



校级超算平台用户手册

作者：上海交通大学网络信息中心

时间：2026 年 1 月 15 日



Contents

1	欢迎致谢交我算平台	2
2	交我算平台计算资源	3
3	常用文档	4
4	注意事项	5
4.1	Getting Started	5
4.2	快速上手	9
4.3	平台硬件资源	12
4.4	账号及充值	18
4.5	可视化平台	27
4.6	登录交我算集群	72
4.7	文件系统与数据访问	100
4.8	提交作业	118
4.9	容器	160
4.10	软件	165
4.11	常见问题	663
4.12	文档下载	670

上海交通大学的校级计算平台“交我算”由网络信息中心负责建设及管理，全面支撑学校的教学、科研和管理的计算需求。“交我算”有两层含义，既表明它是上海交通大学的校级计算平台，又寓意师生的计算需求可以放心的交给“我”（网络信息中心）来算。

致谢模版:

- 致远一号

The computations in this paper were run on the Zhiyuan-1 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

- 思源一号

The computations in this paper were run on the Siyuan-1 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

- π 2.0

The computations in this paper were run on the π 2.0 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

交我算平台计算资源

- “致远一号”国产千卡智算平台。“致远一号”国产于 2025 年建成上线，台硬件配置包括 1024 张华为昇腾 910B 加速卡，FP16 峰值算力 313P，配合 Pacific 分布式存储与 BurstBuffer 架构，总存储容量 13.6PB。计算方面，配备 101 台 CPU 节点，形成 5000+ vCPU 的通用计算池；云服务支持 IaaS、PaaS、SaaS 全栈功能，具备异构算力调度与云化弹性部署，能够满足教学、科研和管理等多样化需求。
- “思源一号”高性能计算平台。2022 年新上线的“思源一号”集群总算力 6 PFLOPS (每秒千万亿次)，是目前国内高校第一的超算集群，TOP500 榜单排名第 132 位。CPU 采用双路 Intel Xeon ICX Platinum 8358 32 核，主频 2.6GHz，共 938 个计算节点；GPU 采用 NVIDIA HGX A100，共 92 块 GPU 卡。计算节点之间使用 Mellanox 100 Gbps Infiniband HDR 高速互联，并行存储的聚合存储能力达 10 PB。
- π 2.0 超算平台。 π 2.0 超算系统于 2019 年上线，双精度浮点数理论性能 2.1 PFLOPS，拥有 656 个双路节点和 1316 颗第二代英特尔至强金牌 6248 处理器，并配以英特尔 Omni-Path 架构的 100 Gbps 高速网络互连，以及全闪存的 NVMeLustre 存储系统，体现了强大的计算能力和先进的设计理念。
- AI 平台。AI 平台由 8 台 NVIDIA DGX-2 服务器组成，双精度计算能力达 1 PFLOPS，张量计算能力达 16 PFLOPS。每台 DGX-2 配置 16 块 Tesla V100 GPU 加速卡，2 颗 Intel 至强铂金 8168 CPU，1.5 TB DDR4 内存，30 TB NVMe SSD 和 512GB HBM2 显存。
- ARM 平台。ARM 平台于 2021 年上线，基于 ARM 处理器构建，是国内首台基于 ARM 处理器的校级超算。共 100 个计算节点，与 π 2.0 和 AI 平台实现共享登录、共享 Lustre 文件系统和共享 Slurm 作业调度系统。ARM 超算单节点配备 128 核 (2.6 GHz)、256 GB 内存 (16 通道 DDR4-2933)、240 GB 本地硬盘。

`docs` 为 HPC 和 AI 平台使用文档，为用户提供快速上手指导和问题解答。可访问上海交通大学 HPC 站点 获取 HPC 和 AI 平台更多信息。

- 0. *Getting Started*
- 1. 快速上手
- 2. 平台硬件资源
- 3. 账号及充值
- 4. 密码
- 5. 登录交我算集群
- 6. 远程桌面
- 7. 提交作业
- 8. 软件
- 9. *GNU Compiler Collection*
- 10. *Intel Compiler*
- 11. 常见问题

- 交我算 HPC+AI 平台禁止运行军工项目等涉密计算任务。
- 欢迎邮件联系我们: `hpc[AT]sjtu.edu.cn`

4.1 Getting Started

Shanghai Jiao Tong University Counting On Me platform is a university-level platform which provides HPC & AI computing service. It was established in 2013 and aims to provide technical support for large-scale scientific and engineering computing needs in the university.

4.1.1 Computing Resources

Siyuan-1

Siyuan-1 is the No.1 supercomputer in domestic universities, ranking 132nd on the TOP500 list. It has 938 CPU nodes (Intel Xeon ICX Platinum 8358) and 23 GPU nodes (NVIDIA HGX A100).

- [Siyuan-1 Documentation](#)

Pi 2.0

Pi 2.0 is one of the earliest supercomputers built with Intel Cascade Lake CPU in China. It has a total of 656 CPU nodes, with a double-precision theoretical peak performance of 2.1 PFLOPS.

AI

AI platform has 8 NVIDIA DGX-2 servers, with 16 Tesla V100 GPU accelerator cards each.

- [AI Documentation](#)

ARM

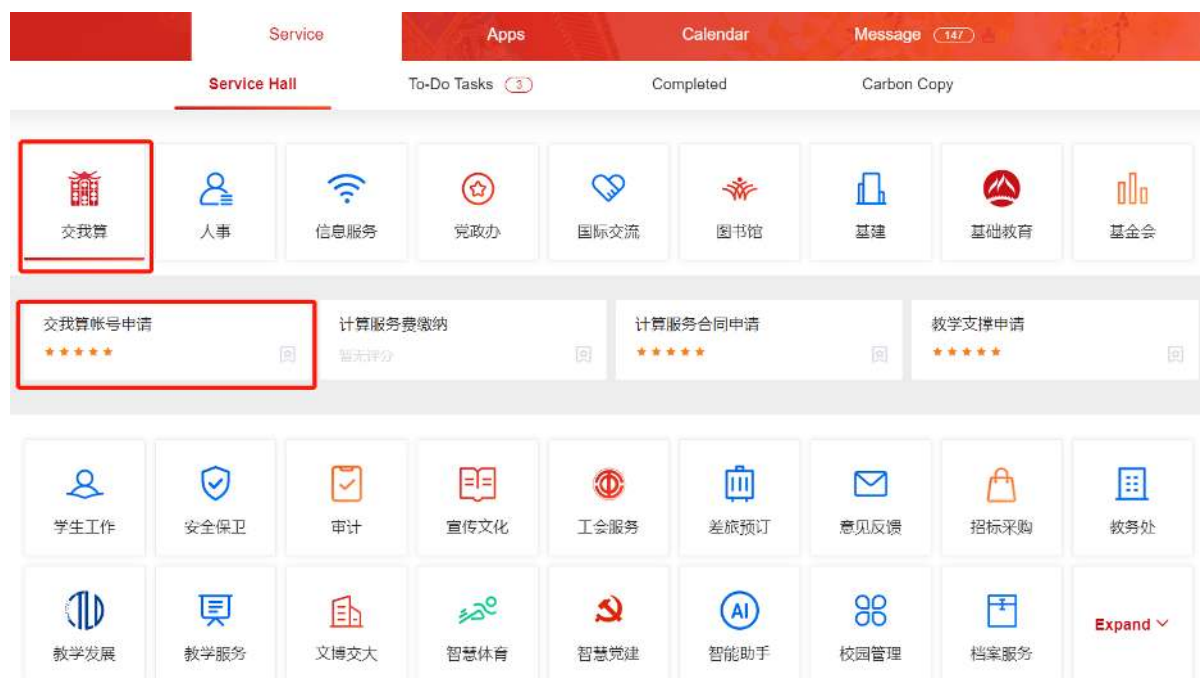
ARM platform is the first university-level supercomputing cluster based on the ARM processor in China. It has a total of 100 CPU nodes.

- [ARM Documentation](#)

4.1.2 HPC Account

How to Apply

HPC account is NOT your jAccount. Apply for an HPC account in ‘交我算’ in my.sjtu.edu.cn as shown. Your application will be processed in 2 business days. The owner of the main account MUST be faculty members, postdocs, or employees of affiliated hospitals. You may also apply for sub-accounts. Sharing accounts is NOT recommended.



Login

1. Connecting with SSH

- Siyuan-1: `ssh username@sylogin.hpc.sjtu.edu.cn`
- Pi2.0 & AI: `ssh username@pillogin.hpc.sjtu.edu.cn`
- ARM: `ssh username@armlogin.hpc.sjtu.edu.cn`

Notice: For Windows users, you may need an SSH client like Putty.

2. Login using HPC Studio

4.1.3 Data Storage

The platform has two file systems: `lustre` and `dssg`. Both `lustre` and `dssg` are distributed, scalable high-performance parallel file systems, that can support PB-level storage capacity and large aggregated I/O throughput.

Different partitions use different file systems:

- `lustre`: small, cpu, huge, 192c6t, dgx2, arm128c256g
- `dssg`: 64c512g, a100

Your home directory is different on different file system as well.

- `lustre: /lustre/home/acct-accountname/username`
- `dssg: /dssg/home/acct-accountname/username`

Data Transfer Nodes

Please use data transfer nodes for data transfers. Data transfer nodes hostnames:

- Siyuan-1: `sydata.hpc.sjtu.edu.cn`
- Pi 2.0/AI/ARM: `data.hpc.sjtu.edu.cn`

External Data Transfer

For Windows users, you may use [WinSCP](#) to transfer data between your PC and the cluster. Use your password, username and the data transfer node hostname to login. And then you will be able to drag your files for data transfers.

For Linux/Unix/Mac users, you may use either `scp` or `rsync` command to transfer data.

1. New data: `$ scp -r [local/path] username@hostname:[target/path]`
2. Changes made to existing data: `$ rsync --archive --partial --progress [local/path] username@hostname:[target/path]`

```
# Suppose the home directory of user expuser01 in Siyuan-1 is /dssg/home/
↪acct-exp/expuser01
# His/her local directory is /home/local_user/

# Example 1: upload all files in ~/data folder to Siyuan-1 cluster
$ scp -r /home/local_user/data/ expuser01@sydata.hpc.sjtu.edu.cn:/dssg/home/
↪acct-exp/expuser01/

# Example 2: download ~/math.dat file in Siyuan-1 to the local directory
$ scp expuser01@sydata.hpc.sjtu.edu.cn:/dssg/home/acct-exp/expuser01/math.
↪dat /home/local_user/
```

Transferring Data between Clusters

You can use either `data` or `sydata` data transfer node to transfer data between two file systems. For example, you can `ssh data.hpc.sjtu.edu.cn` from Pi 2.0 cluster, and then transfer data using either `scp` or `rsync` command.

```
# Example 3: transfer data in lustre home directory ~/data to dssg home
↪directory ~/data
$ scp -r /lustre/home/acct-exp/expuser01/data/ expuser01@sydata.hpc.sjtu.
↪edu.cn:/dssg/home/acct-exp/expuser01/data/
```

4.1.4 Running Jobs

SLURM (Simple Linux Utility for Resource Management) is a scalable workload manager. The platform uses Slurm for cluster management and job scheduling.

- [Common SLURM commands](#)

Here is a typical slurm script example.

```
#!/bin/bash

#SBATCH --job-name=test                #job name
#SBATCH --partition=64c512g           #partition name
#SBATCH -N 1                          #number of nodes
#SBATCH --ntasks-per-node=64          #number of tasks (1 core = 1 task by default)
#SBATCH --output=%j.out               #output file
#SBATCH --error=%j.err               #error file

module load XXX                      #load modules

mpirun -n $SLURM_NTASKS ...
```

Suppose your script file name is `example.slurm`. You may submit your job by using the command `sbatch example.slurm`.

- Variety of script templates

4.1.5 Top Documentation Pages

1. [Overview](#) - Platform overview
2. [System](#) - File and computing systems
3. [Account](#) - HPC account and password
4. [Login](#) - How to login
5. [HPC Studio](#) - Web-based graphical user interface
6. [Jobs](#) - Slurm commands and job script basics
7. [Software](#) - Software overview and slurm job script templates
8. [Q&A](#) - Common questions

4.1.6 Useful Links

1. HPC Home Page: <https://hpc.sjtu.edu.cn/>
2. HPC Studio: <https://studio.hpc.sjtu.edu.cn/>
3. Platform Live Usage: <https://account.hpc.sjtu.edu.cn/top/>
4. Cluster Operation Monitor: <https://mon.hpc.sjtu.edu.cn/>
5. Account and Bill Portal: <https://account.hpc.sjtu.edu.cn/>

4.1.7 Acknowledgement

Acknowledgment templates:

- Siyuan-1

The computations in this paper were run on the Siyuan-1 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

- π 2.0

The computations in this paper were run on the π 2.0 cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

4.1.8 Notifications

- Running any classified jobs is prohibited
- For any questions, contact us: `hpc[AT]sjtu.edu.cn`

4.2 快速上手

4.2.1 为什么需要交我算 HPC+AI 平台?

您可能需要大规模计算，超出笔记本电脑或工作站的处理能力；您可能需要太多数据，需要海量存储和高速读写；您可能需要先进高效的 GPU 资源，抑或是大内存节点。这些，都能在交我算 HPC+AI 平台上实现。

交我算平台面向全校师生提供服务，支撑和催化学校的科研发展。重点支持校内高水平用户的科研，覆盖各学科门类，支撑海洋学、生物医学、航空航天、机械制造、天体物理等领域的科学研究及工程应用，多篇研究发表于 *Science*、*Nature* 等高水平期刊上。

4.2.2 交我算 HPC+AI 平台硬件资源如何?

“思源一号”集群总算力 6 PFLOPS（每秒千万亿次），是国内高校领先的超算集群。CPU 采用双路 Intel Xeon ICX Platinum 8358 32 核，主频 2.6GHz，共 938 个计算节点；GPU 采用 NVIDIA HGX A100，共 92 块 GPU 卡。计算节点之间使用 Mellanox 100 Gbps Infiniband HDR 高速互联，并行存储的聚合存储能力达 10 PB。

π 2.0 超算系统双精度浮点数理论性能 2.1 PFLOPS，拥有 658 个双路节点和 1316 颗第二代英特尔至强金牌 6248 处理器，配以英特尔 Omni-Path 架构的 100 Gbps 高速网络互连，以及全闪存的 NVMeLustre 存储系统，体现了强大的计算能力和先进的设计理念。

AI 计算平台由 8 台 DGX-2 组成，每台 DGX-2 配备 16 块 NVIDIA Tesla V100，深度学习张量计算能力可以达到 16 PFLOPS；通过搭载 NVIDIA NVSwitch 创新技术，GPU 间带宽高达 2.4 TB/s。AI 计算平台采用可扩展架构，使得模型的复杂性和规模不受传统架构局限性的限制，从而可以应对众多复杂的人工智能挑战。

ARM 平台基于 ARM 处理器构建，是国内首台基于 ARM 处理器的校级超算。共 100 个计算节点，与 π 2.0 集群实现共享登录、共享 Lustre 文件系统和共享 Slurm 作业调度系统。ARM 超算单节点配备 128 核（2.6 GHz）、256 GB 内存（16 通道 DDR4-2933）、240 GB 本地硬盘，节点间采用 IB 高速互联，挂载 Lustre 并行文件系统。

4.2.3 资源如何选择？

交我算 HPC+AI 平台采用 Linux 操作系统，配以 Slurm 作业调度系统，所有计算节点资源和存储资源，均可统一调用。

若是大规模的 CPU 作业，可选择 CPU 队列或思源一号 64c512g 队列，支持万核规模的并行；

若是小规模测试，可选 small 队列或思源一号 64c512g 队列；

GPU 作业请至 dgx2 队列或思源一号 a100 队列；

大内存作业可选择 huge 或 192c6t 两种队列。

详情请见：[Slurm 作业调度系统](#)

4.2.4 使用交我算 HPC+AI 平台需要什么？

您只需要一个交我算账号，然后在任何一个常见的浏览器上登录 可视化平台 HPC Studio，即可自由使用平台。

可视化平台 HPC Studio 使得登录更便捷，无需安装客户端，大大提升使用体验。浏览器包含电脑端和移动端的 Chrome, Firefox, Edge, Safari 等。

当然您也可以通过 SSH 客户端连接使用平台。

了解平台使用方法，请查看 [常见问题](#)；

简短版使用手册（Cheat Sheet）：[交我算平台使用手册简短版](#)

4.2.5 如何申请账号？

交我算平台服务于交大师生。账号申请者需为交大及附属医院在职教师或博士后。

主账号申请

在“交我办”（或 [我的数字交大 my.sjtu.edu.cn](#)）中的“交我算”里申请。我们将会在两个工作日内开通账号。

子账号申请

每个主账号可免费申请子账号。

子账号由账号负责人或子账号使用人在“交我办”（或 [我的数字交大 my.sjtu.edu.cn](#)）中的“交我算”里申请。

开通账号之前，还需账号使用者完成“新手上路测试题”，共 20 题。合格分数为 90 分，可重复完成，需要将完成后的截图上传。

测试题：<https://f.kdocs.cn/ew/3aYIdZR9/>

参考视频：<https://vshare.sjtu.edu.cn/play/26809b70229c549722c5afd0cf868f77>

可参阅的资料：

a) 用户文档：<https://docs.hpc.sjtu.edu.cn>

b) HPC 网站: <https://hpc.sjtu.edu.cn>

c) 简短版使用手册 (Cheat Sheet): https://hpc.sjtu.edu.cn/Item/docs/Pi_GetStarted.pdf

4.2.6 参考教学视频

a) 2022 春季用户培训之交我算简介

b) 2022 春季用户培训之交我算新手上路

4.2.7 提交 Hello world 单节点作业

以单节点的 OpenMP Hello world 为例, 演示作业提交过程。

1. 撰写名为 `hello_world.c` 代码如下

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
    int nthreads, tid;

    /* Fork a team of threads giving them their own copies of variables */
    #pragma omp parallel private(nthreads, tid)
    {

        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);

        /* Only master thread does this */
        if (tid == 0)
        {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }

        /* All threads join master thread and disband */
    }
}
```

2. 使用 GCC 编译

```
$ module purge
$ module load gcc
$ gcc -fopenmp hello_world.c -o hello_world
```

3. 在本地测试运行 4 线程应用程序

```
$ export OMP_NUM_THREADS=4 && ./hello_world
```

4. 编写一个名为 `hello_world.slurm` 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=hello_world
#SBATCH --partition=small
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 8
#SBATCH --ntasks-per-node=8

ulimit -l unlimited
ulimit -s unlimited

module load gcc

export OMP_NUM_THREADS=8
./hello_world
```

5. 提交到 SLURM

```
$ sbatch hello_world.slurm
```

小技巧

编译和作业提交都需要登录到 HPC+AI 平台集群, 参考本节 [使用交我算 HPC+AI 平台需要什么?](#)。

4.2.8 登录可视化计算平台

HPC Studio 可视化平台, 集成 web shell、文件管理、作业提交、可视化应用等一站式服务。

登录方法:

在浏览器中打开: [HPC Studio 可视化平台](#)

详情请见: [HPC Studio 可视化平台使用方法](#)

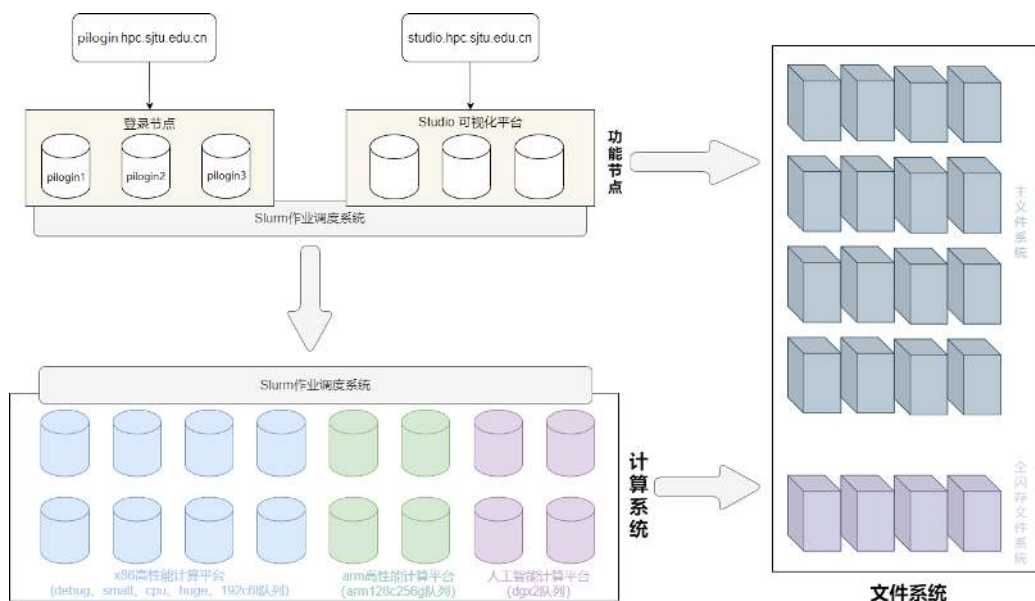
4.3 平台硬件资源

交我算 HPC+AI 平台硬件资源分文件系统和计算系统。

文件系统 (File System) 是集群中负责管理和存储文件信息的软件架构, 用于向用户提供底层数据访问。

计算系统 (Computing System) 是集群中负责提供计算资源、执行计算操作的节点集合。

以 HPC+AI 平台集群 (除思源一号外) 的架构示意图为例:



4.3.1 文件系统

HPC+AI 平台集群（除思源一号外）采用 Lustre 作为后端存储系统。Lustre 是一种分布式的、可扩展的、高性能的并行文件系统，能够支持数万客户端、PB 级存储容量、数百 GB 的聚合 I/O 吞吐，非常适合众多客户端并发进行大文件读写的场合。Lustre 最常用于高性能计算 HPC，世界超级计算机 TOP 10 中的 70%、TOP 30 中的 50%、TOP 100 中的 40% 均部署了 Lustre。

HPC+AI 平台集群（除思源一号外）已上线多套 Lustre 文件系统，挂载在计算节点的不同目录：`/lustre`、`/scratch`。

数据传输节点（`data.hpc.sjtu.edu.cn`）还多挂载了一个 `/union`。

思源一号为独立集群，使用 Gpfs 文件系统，共 10P。系统包含 4 台 DSS-G Server 节点，每台配置 2 块 300G HDD，用于安装操作系统，安装配置 GPFS 集群及创建文件系统。文件系统 metadata 采用 3 副本冗余，文件系统 data 采用 8+2p 冗余。

主文件系统

`/lustre` 目录挂载的为 HPC+AI 平台集群（除思源一号外）中的主文件系统，共 13.1P，用户的个人目录即位于该目录。

主文件系统特性

主文件系统主要使用 HDD 盘搭建，旨在提供大容量、高可用、较高性能的存储供用户使用。搭建过程中，使用 RAID 保障硬盘级别的数据安全，使用 HA (High Availability) 保障服务器级别的高可用。

用户的主要工作、重要数据都应该发生和存储在主文件系统。

如何使用主文件系统

用户通过个人账户登录计算节点（包括登录节点）之后，默认进入主文件系统，即 **HOME** 目录。可以在以下路径找到 **/lustre** 提供给用户的空间：

```
/lustre/home/acct-xxxx/yyyy
```

其中 **acct-xxxx** 代表计费帐号（课题组帐号），**yyyy** 代表个人帐号。

通过 `cd cd $HOME cd ~` 等方式都可进入主目录。

全闪存文件系统

/scratch 目录挂载的为 **HPC+AI** 平台集群（除思源一号外）的全闪存并行文件系统，共 **108T** 容量，可作用户的临时工作目录。

全闪存文件系统特性

全闪存文件系统使用全套的 **SSD**（**NVMe** 协议）硬盘搭建，旨在提供高性能的存储供用户使用，可更好地支持 **IO** 密集型作业。对系统来说，单客户端最大读带宽达 **5.7GB/s**，最大写带宽达 **10GB/s**；**4k** 小文件读 **IOPS** 达 **170k**，写 **IOPS** 达 **126k**。但同时，由于成本问题，系统提供的容量较小；在搭建时也未设置高可用和数据备份，存在数据存储安全性不高等问题。

基于该套系统的特性，推荐用户将其作为临时工作目录，可用于

1. 存储计算过程中产生的临时文件
2. 保存读写频率高的文件副本

注意：为了保持全闪存文件系统的稳定可用，**/scratch** 目录每 **3** 个月会进行一次清理。因此，请务必及时将重要数据保存回 **/lustre** 目录。

如何使用全闪存文件系统

用户可以在以下路径找到 **/scratch** 提供的暂存空间：`/scratch/home/acct-xxxx/yyyy`

其中 **acct-xxxx** 代表计费帐号（课题组帐号），**yyyy** 代表个人帐号。

为了快捷访问，我们已经为用户设置好环境变量，`cd $SCRATCH` 即可进入该临时工作目录。

作业使用示例

我们使用生信 **WES** 分析流程为例，该流程从测序文件开始，经 **bwa** 比对、**samtools** 处理，然后用 **GATK** 检测变异（部分步骤）。原代码如下：

```
#!/bin/bash

#SBATCH --job-name=WES
#SBATCH --partition=cpu
```

(续下页)

(接上页)

```

#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive

echo "##### 加载相关软件 #####"
module load bwa samtools
module load miniconda3 && source activate 10_24 # gatk-4.1.9.0

echo "##### 设置变量 #####"
REFDIR=$HOME/med/annotation/gatk/hg19 # 参考基因组和注释文件目录
SAMPLEDIR=$HOME/med/testnor # 样本目录

WORKDIR=$HOME/WES_TEST # 工作目录
TMPDIR=$WORKDIR/tmpdir # 临时缓存目录
mkdir -p $WORKDIR
mkdir -p $TMPDIR

SampleID=test # 样本名

cd $WORKDIR

echo "##### bwa 比对 #####"
bwa mem -M -t 40 \
${REFDIR}/ucsc.hg19.fasta \
${SAMPLEDIR}/${SampleID}_1.fastq.gz \
${SAMPLEDIR}/${SampleID}_2.fastq.gz \
| gzip -3 > ${SampleID}_mem.sam

echo "##### samtools 生成bam #####"
samtools view -@ 40 -bS ${SampleID}_mem.sam \
| samtools sort -@ 40 > ${SampleID}_mem.sorted.bam

samtools index ${SampleID}_mem.sorted.bam

echo "##### gatk 检测变异 #####"
gatk ReorderSam \
-I ${SampleID}_mem.sorted.bam \
-O ${SampleID}_mem.sorted.reorder.bam \
-R ${REFDIR}/ucsc.hg19.fasta \
--TMP_DIR ${TMPDIR} \
--VALIDATION_STRINGENCY LENIENT \
--SEQUENCE_DICTIONARY ${REFDIR}/ucsc.hg19.dict \
--CREATE_INDEX true

gatk MarkDuplicates \
-I ${SampleID}_mem.sorted.reorder.bam \
-O ${SampleID}_mem.sorted.reorder.rmdup.bam \

```

(续下页)

(接上页)

```
--TMP_DIR ${TMPDIR} \
--REMOVE_DUPLICATES false \
--ASSUME_SORTED true \
--METRICS_FILE ${SampleID}_mem.sorted.reorder.markduplicates_metrics.txt \
--OPTICAL_DUPLICATE_PIXEL_DISTANCE 2500 \
--VALIDATION_STRINGENCY LENIENT \
--CREATE_INDEX true
```

过程中，会产生许多中间文件和临时文件。因此，可利用 `$SCRATCH` 作为临时目录，加快分析过程。只需要把脚本中的 `WORKDIR=$HOME/WES_TEST` 修改为 `WORKDIR=$SCRATCH/WES_TEST` 即可。

冷存储系统

在 *data* (*data.hpc.sjtu.edu.cn*) 节点的目录 */union* 下挂载了冷存储，共 29P 容量，用来存储用户的不常用数据。

冷存储系统特性

作为 **Lustre/GPFS** 文件系统的补充，提供大容量、高可用的存储，只能在传输节点访问，主要给用户存放不常用的数据。冷存储系统具有快照功能，快照可以恢复用户误删除的数据。用户通过 `cd $UNION` 可以跳转到自己的目录

如何使用冷存储系统

(1) 登录 *data* 节点

```
# ssh $USER@data.hpc.sjtu.edu.cn
```

(2) 进入冷存储系统

用户可以在以下路径找到 */union* 提供的个人存储空间: */union/home/acct-xxxx/yyyy*

其中 *acct-xxxx* 代表计费帐号（课题组帐号），*yyyy* 代表个人帐号。

为了快捷访问，我们已经为用户设置好环境变量，`cd $UNION` 即可进入。

(3) 将不常用文件移动到 *UNION*

```
# rsync -avh -P --append-verify $DATA $UNION
```

推荐使用 **rsync** 移动数据，详细参数含义可使用 `man rsync` 命令查看。

4.3.2 计算系统

计算系统包含集群中所有的计算节点，提供给用户进行实际作业计算，包括思源一号平台、 π 2.0 平台、人工智能计算平台和 ARM 平台。

下面列出了各计算平台具体的硬件信息。其中，思源一号平台、 π 2.0 平台、人工智能计算平台使用的为 x86 服务器，国产 ARM 平台使用的为 arm 架构的华为鲲鹏服务器。因 x86 和 arm 的 CPU 架构不同，计算应用和软件库都需要重新编译。

思源一号平台

队列	架构	参数	节点数量
64c51	x86	CPU: 2 x Intel Xeon ICX Platinum 8358 (2.6GHz, 32 cores) Mem: 16 x 32GB TruDDR4 3200 MHz (2Rx8 1.2V) RDIMM	938
a100	x86	CPU: 2 x Intel Xeon ICX Platinum 8358 (2.6GHz, 32 cores) GPU: 4 x NVIDIA HGX A100 40GB	23(4卡/节点)

π 2.0 平台

队列	架构	参数	节点数量
debug/ small/ cpu	x86	CPU: 2 x Intel Xeon Scalable Cascade Lake 6248(2.5GHz, 20 cores) Mem: 12 x Samsung 16GB DDR4 ECC REG 2666	656
huge	x86	CPU: 4 x Intel Xeon Scalable SkyLake 6148(2.4GHz, 20 cores) Mem: 48 x Hynix 64GB DDR4 ECC REG 2666	2
192c6t	x86	CPU: 8 x Intel Xeon Platinum 8260(2.4GHz, 24 cores) Mem: 98 x Samsung 64GB DDR4 2666	1

人工智能计算平台

队列	参数	节点数量
dgx2	CPU: 2 x Intel Xeon Scalable Cascade Lake 8168(2.7GHz, 24 cores) Mem: 1.5 TB DDR4 ECC REG 2666 GPU: 16 x NVIDIA Tesla V100	8(16卡/节点)

ARM 平台

队列	架构	参数	节点数量
arm128c2	arm	CPU: 2 x HiSilicon Kunpeng 920-6426(2600MHz, 64cores) Mem: 16 x Samsung 16GB DDR4 2933	100

4.4 账号及充值

为了使用交我算 HPC+AI 平台集群，您需要：

1. 交我算账号，用于远程访问集群的计算资源；
2. jAccount 账号，用于交我算账号的缴费充值。

4.4.1 交我算账号申请

账号说明

在交我算的账号体系中，账号会被分为主账号和子账号。

主账号是收缴费主体，由课题组负责人申请。

主账号与子账号均为独立账号，仅在计费关系上存在关联。

每个主账号能免费申请子账号，个数不限，请按需申请。

主账号和子账号申请

由账号负责人或账号使用人在“交我办”（或 我的数字交大 my.sjtu.edu.cn）中的“交我算”里申请。我们将会在工作日内开通账号。

4.4.2 账号安全信息管理

用户类型		账号关联	账号独立密码
交大用户	新用户	自动关联 jAccount	仅用于登录 https://account.hpc.sjtu.edu.cn ，未来将统一使用 jAccount 密码。如需修改，请参考此章节
	2024/09/23 前老用户	请参考此章节关联 jAccount	
合作用户		请参考此章节关联邮箱	唯一账号密码，不做变动，如需修改，请参考此章节

关联 jAccount 和邮箱

超算平台于 2024 年 9 月 23 日开始对接交大 jAccount 账号，超算账号关联 jAccount 后您可以用 jAccount 密码登录超算平台，并且可以选择 jAccount 作为通知渠道，这种情况下您将在 交我办 APP 上接收到实时的安全通知，例如账号登录提醒，免密授权确认等。

对于没有 jAccount 的合作用户，平台也支持关联邮箱，您可以通过邮箱接收上述的安全通知。

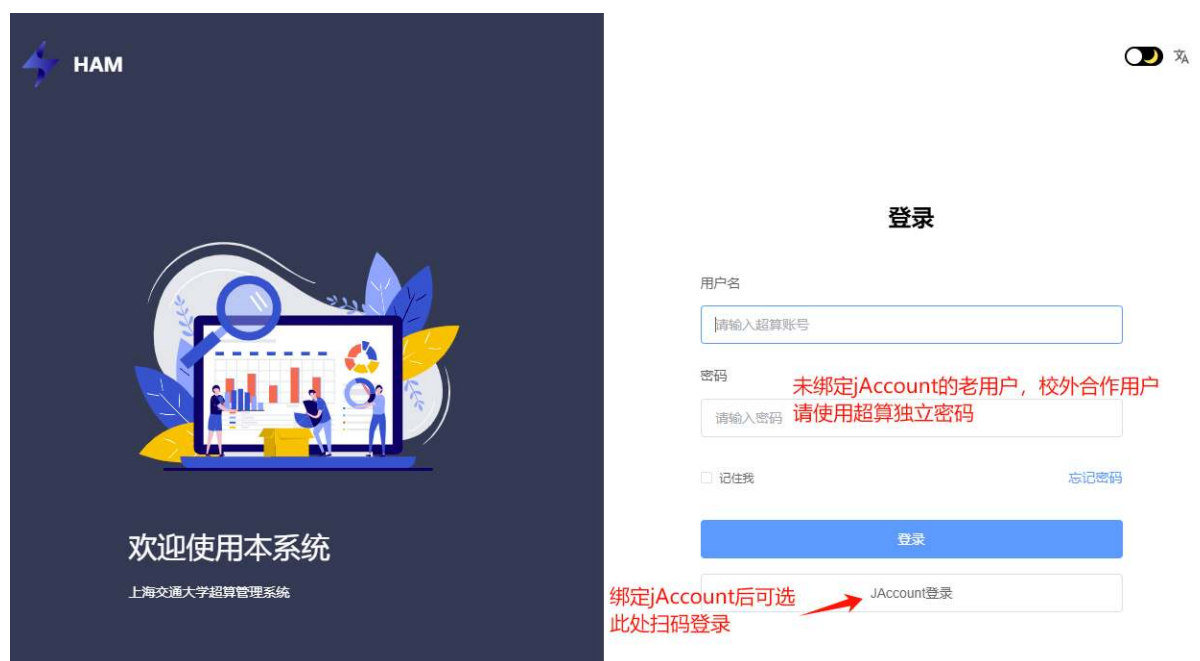
如果您是交大校内用户，在 2024 年 9 月之后申请的超算账号将自动关联您的 jAccount，但如果您的超算账号是在此时间点之前开设的，您可能在 SSH 登录时看到如下提醒：

```
> ssh exampleuser@pilogin.hpc.sjtu.edu.cn
If you are SJTU user, authenticate with jaccount password; otherwise use
↪HPC password:

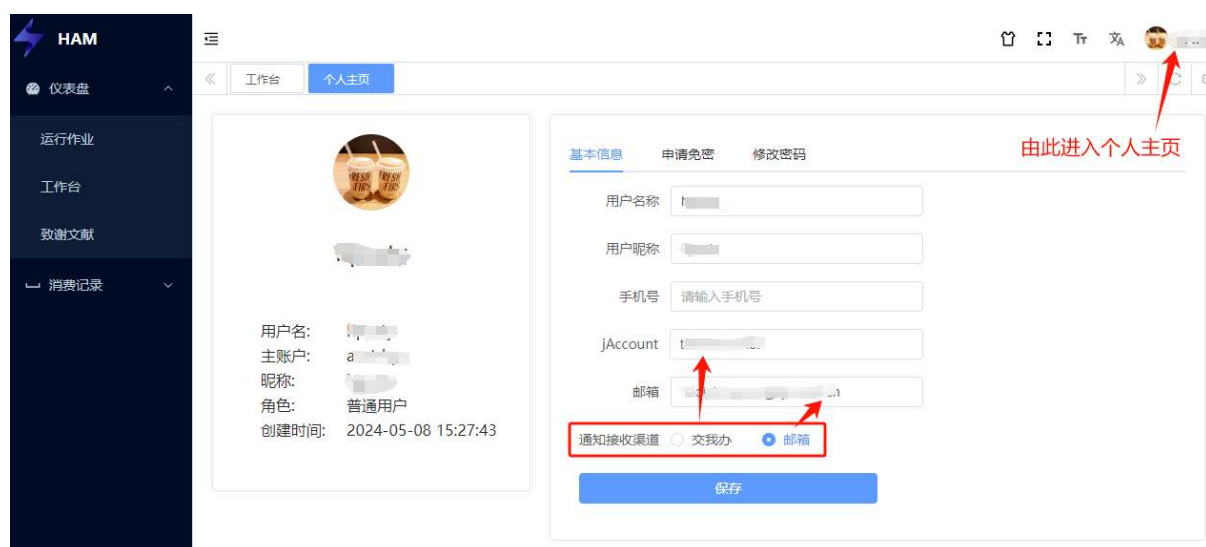
Please link your jaccount through https://my.hpc.sjtu.edu.cn as soon as
↪possible!
```

这说明您的账号缺少身份信息，尚未关联 jAccount，请按照提示前往 [超算账号管理平台](#) 补充关联信息。

如下图所示，缺少身份信息的旧账号首次登录 [超算账号管理平台](#) 请使用您的超算账号密码，已经关联 jAccount 的账号可以选择 交我办 APP 扫码登录。



登录后点击右上角用户名，进入 [个人主页](#)，在基本信息标签页上维护您的 jAccount/邮箱信息，以及首选的安全通知接收渠道。



缺少身份信息的旧账号首次关联 jAccount/邮箱时不需要进行二次认证，请仔细填写您的关联信息。关联过 jAccount/邮箱之后，若需要再次修改，系统将向您原本的 jAccount/邮箱发送询问授权的通知，只有确认授权后才能更新 jAccount/邮箱的关联。

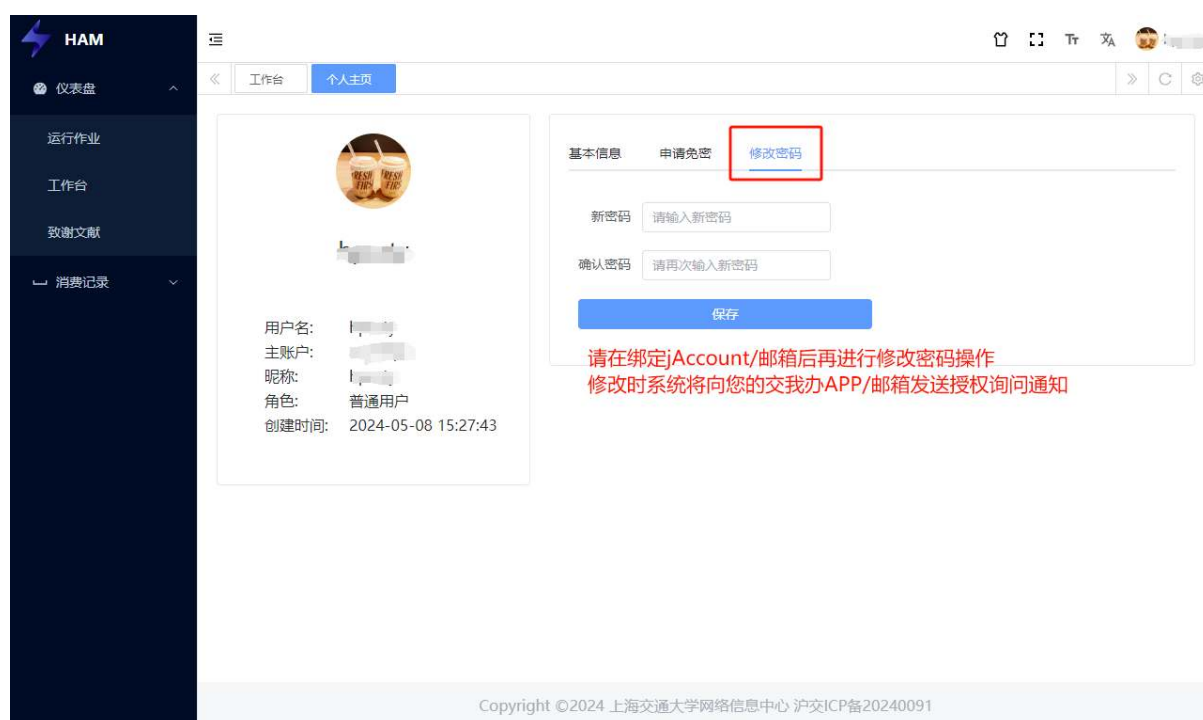
超算账号独立密码

对于合作用户，超算平台依然保留独立账号密码。

对于校内用户，超算平台将逐步取消独立密码，统一使用 jAccount 密码。但现阶段您依然需要在登录 <https://account.hpc.sjtu.edu.cn/> 时使用超算账号的独立密码。

新用户完成账号创建之后，我们会发送一封邮件给您，邮件中包含了您账号的独立密码。请您务必保管好账号密码不外泄。另外，我们强烈建议您收到账号密码后立即重置独立密码。建议使用 `openssl rand -base64 6` 命令或者在线密码生成器（如 [Strong Random Password Generator](#)）获得足够强度的随机密码，并使用 SSH 公钥登录和密码管理器（如 [KeePass](#)）降低记忆难度。

您可以在 [超算账号管理平台](#) 的 [个人主页](#) 维护超算账号独立密码，如下图所示。修改密码要求您的账号已经关联过 jAccount/邮箱，修改时系统将向您原本的 jAccount/邮箱发送询问授权的通知，只有确认授权后才能完成密码更新。



密码重置

如果您需要重置的是 jAccount 密码，请在 [交我办](#) 发起对应的申请流程。

如果您需要重置的是超算账号独立密码，建议按照上一节的说明在 [超算账号管理平台](#) 中进行操作。或者也可以发送邮件至 [HPC 邮箱](#) 申请人工协助。

密码规范

如果您要自行更改密码，我们建议您遵循如下规则制定新的密码：

- 1. 不要使用固定的常用密码
- 2. 不要使用有意义的字符串
- 3. 密码长度至少 8 位
- 4. 密码中至少包含字母和数字

申请免密证书

如果您不希望在 SSH 登录时手动输入密码，您可以在 [超算账号管理平台](#) 申请免密证书。

免密证书是基于个人公钥签发的，如果您没有 SSH 公私钥或者不了解相关内容，请选择一键生成方式，如下图所示，平台将为您自动生成新的公私钥文件：



如果您希望使用原有的个人公私钥对，请上传公钥文件或将公钥文本粘贴到网页的文本框中，系统将基于此公钥签发免密证书。

下载证书时不建议修改文件名，如果您要自行修改保存的文件名，请确认证书文件名与私钥文件名相匹配。下表列举了一些文件命名示例，以演示证书和公私钥文件名之间的对应关系。

示例	私钥	公钥	证书
rsa 加密默认名称	id_rsa	id_rsa.pub	id_rsa-cert.pub
ed25519 加密默认名称	id_ed25519	id_ed25519.pub	id_ed25519-cert.pub
自定义名称	mykey	mykey.pub	mykey-cert.pub

关于免密证书的使用方法，请参阅 [通过 SSH 登录集群](#) 的相关章节。

4.4.3 收费及充值流程

收费标准

交我算平台集群总费用为 CPU, GPU 和存储费用之和。CPU 资源按核时收费, GPU 资源按卡时收费。具体费率标准请发送邮件至 [hpc 邮箱](mailto:hpc@sjtu.edu.cn) 咨询。

校内充值流程

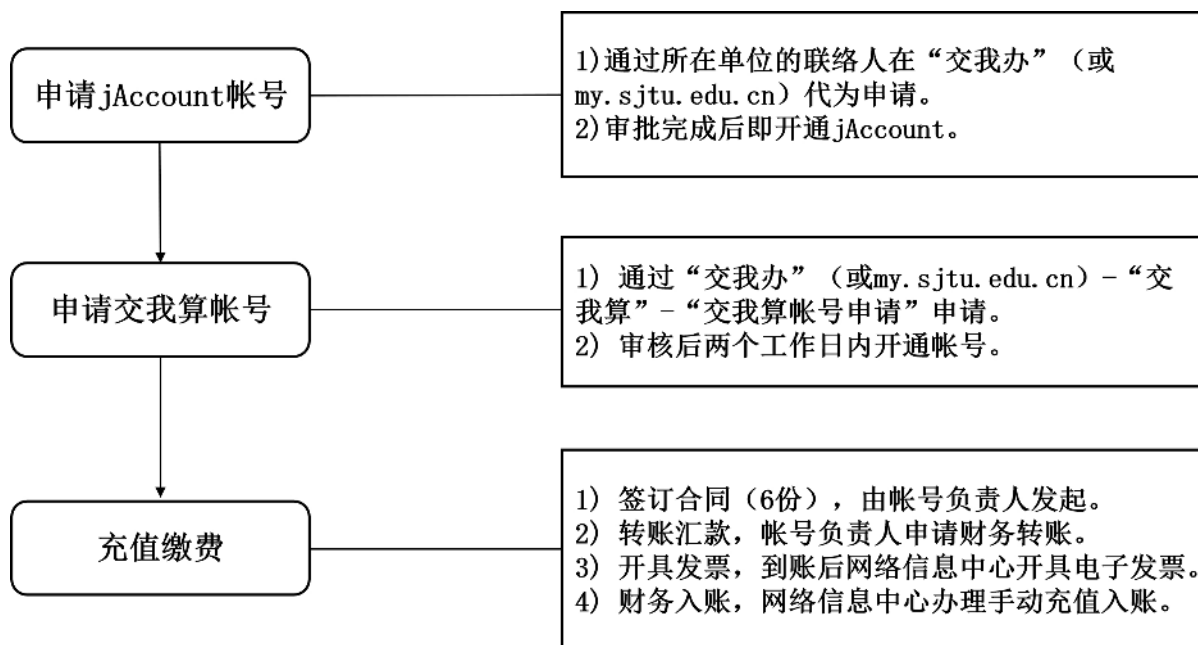
校内转账可在“我的数字交大”网页，或“交我算”APP 里完成，具体操作请见：
[计算服务费校内经费转账说明](#)

如有任何财务问题，请联系网络信息中心基础部杨老师，电话 34206060-8017，邮箱 yjqedith@sjtu.edu.cn

4.4.4 附属医院账号申请及充值流程

“交我算”账号与交大统一身份认证 jAccount 绑定，附属医院教职工可以使用 jAccount，通过交大“一门式服务”APP “交我办”（或网页版 my.sjtu.edu.cn）提交“交我算”账号申请。如您尚未开通 jAccount，可以通过所在单位的联络人在“交我办”上发起 jAccount 账号代申请，通过后您便可自行申请“交我算”账号。

以下是附属医院教职工“交我算”账号申请及充值流程图：

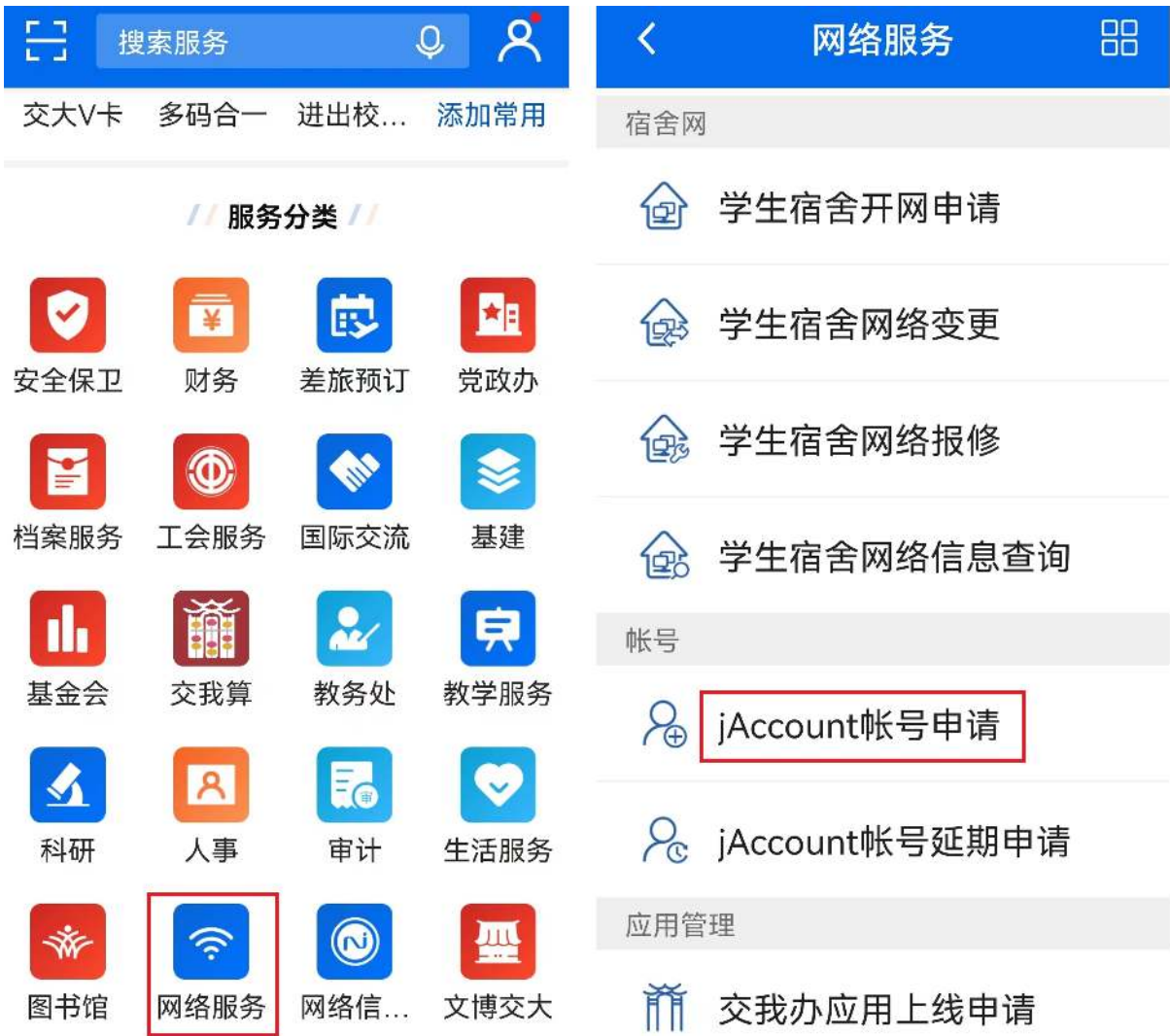


第一步：申请 jAccount 账号

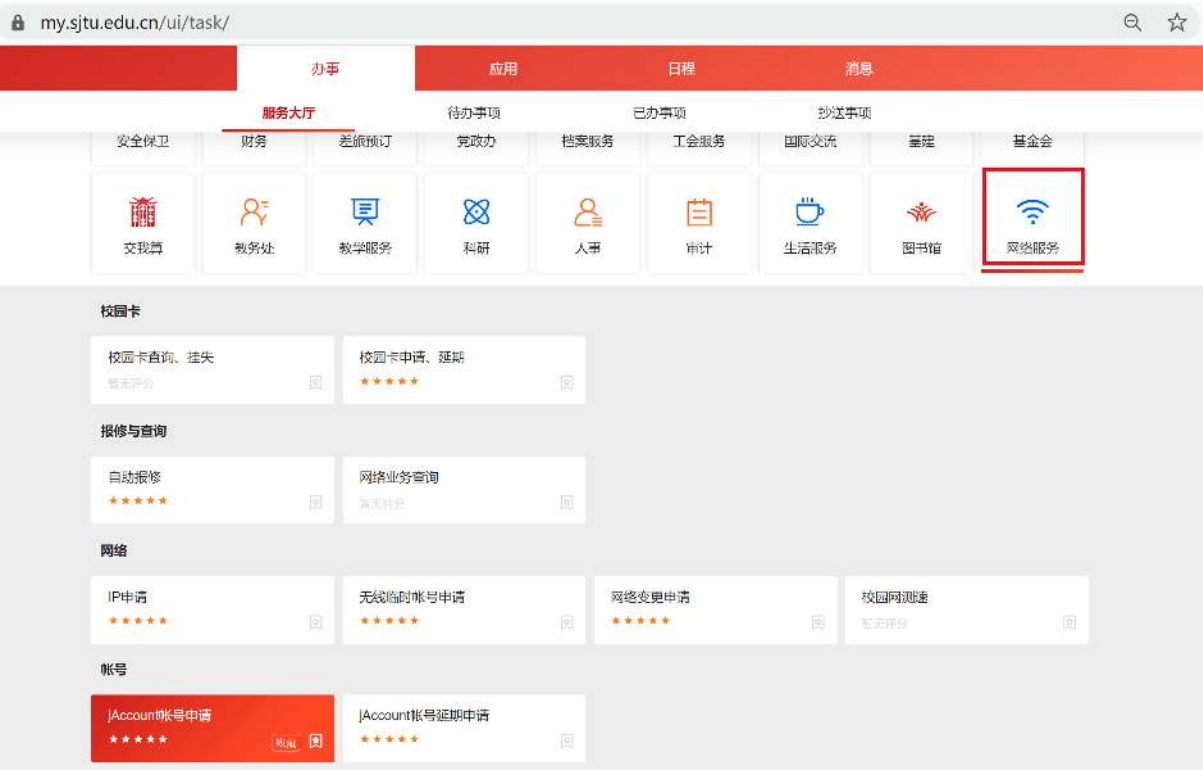
通过所在单位联络人在“交我办”APP（或网页版 my.sjtu.edu.cn）上代为申请 jAccount。经所在单位人事干事或领导审批完成后即开通 jAccount，且自动开通与 jAccount 同命名的交大邮箱。

jAccount 申请入口（二选一，“交我办” APP 或网页版 my.sjtu.edu.cn）图示：

1. “交我办” APP- “网络服务” - “jAccount 账号申请”



2. 网页版（或 my.sjtu.edu.cn） - “网络服务” - “jAccount 账号申请”

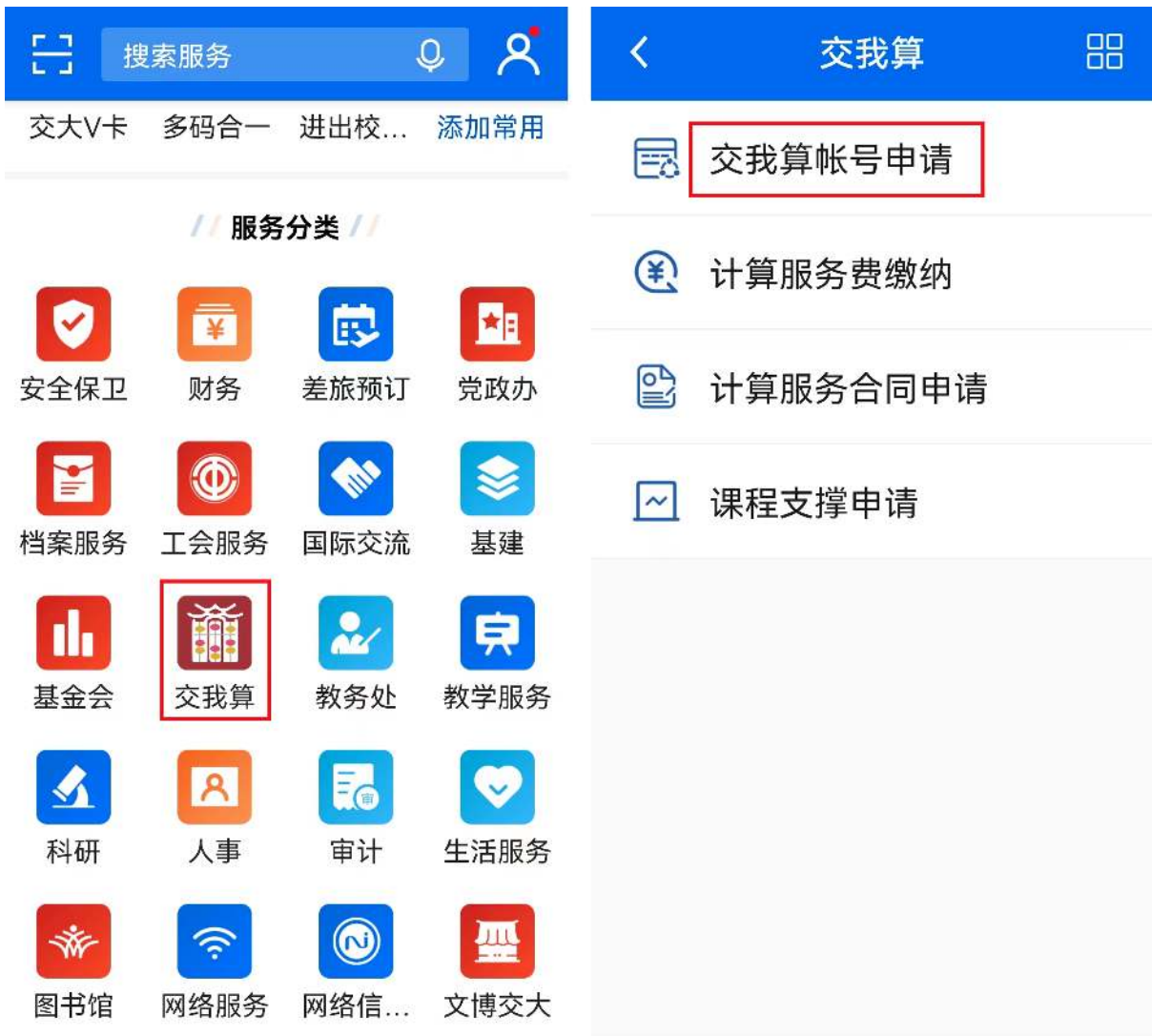


第二步：申请交我算账号

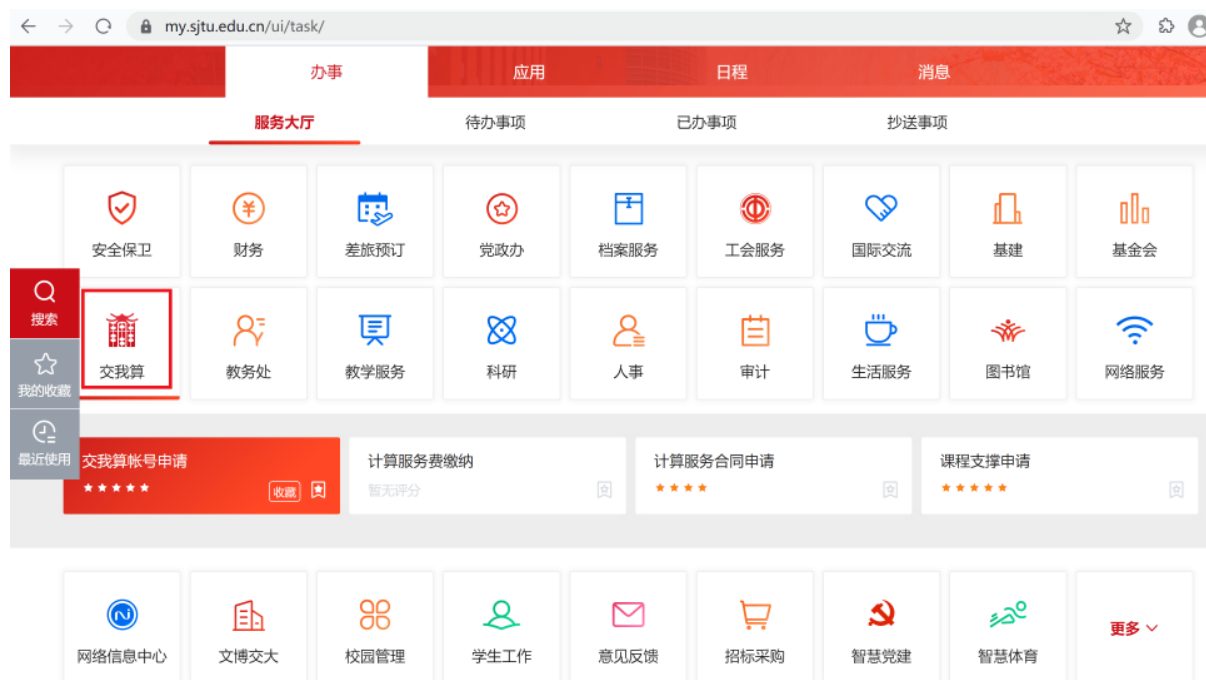
通过 jAccount 登录“交我办”APP（或网页版 my.sjtu.edu.cn）进行交我算账号申请。审核通过后，我们将在两个工作日内开通账号。

交我算账号申请入口（二选一，“交我办”APP 或网页版 my.sjtu.edu.cn）图示：

1. “交我办”APP- “交我算” - “交我算账号申请”



2. 网页版（或 my.sjtu.edu.cn） - “交我算” - “交我算账号申请”



第三步：充值缴费

1. 签订合同及合同盖章

- 由账号负责人发起，将合同内相关内容填好并盖单位公章，然后将 4 份合同原件寄给网络信息中心刘素霞老师（上海市闵行区东川路 800 号上海交通大学网络信息中心 501 室，34206060-8520）
- 合同下载：计算平台技术服务合同
- 4 份合同盖好交大公章后，其中 2 份合同原件将回寄给账号负责人。

2. 转账汇款

- 账号负责人于所在医院申请财务报销和转账，并将费用转入：

账户名称：上海交通大学

银行账号：439059226890

开户银行：中国银行上海市上海交通大学支行

联行号：104290050144

- 附属医院财务报销如需提供测试报告，请账号负责人发邮件至 hpc@sjtu.edu.cn 申请。

3. 开具发票

- 汇款到账后，请账号负责人联系网络信息中心刘老师 liusuxia723@sjtu.edu.cn，并提供汇款信息备注及开票信息。
- 网络信息中心确认汇款到账后，将开具电子发票并邮件发送至账号负责人。

4. 财务入账

- 账号负责人确认”交我算”账号用户名及 jAccount 后，由网络信息中心办理手动充值入账。
- 账号负责人确认计费系统到账。

充值过程中有任何问题，请联系网络信息中心刘老师 liusuxia723@sjtu.edu.cn。

4.5 可视化平台

HPC Studio 可视化平台，集成 web shell、文件管理、作业提交、可视化应用等一站式服务。

4.5.1 HPC Studio

登录

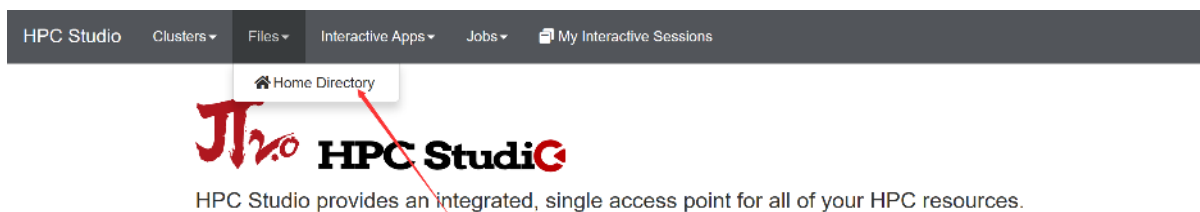
在浏览器中，打开：<https://studio.hpc.sjtu.edu.cn>

小技巧

浏览器需为 Chrome，Firefox 或 Edge。

文件管理

点击图标 Files 下拉菜单 Home Directory 即可进入文件管理界面



文件管理包含以下功能：

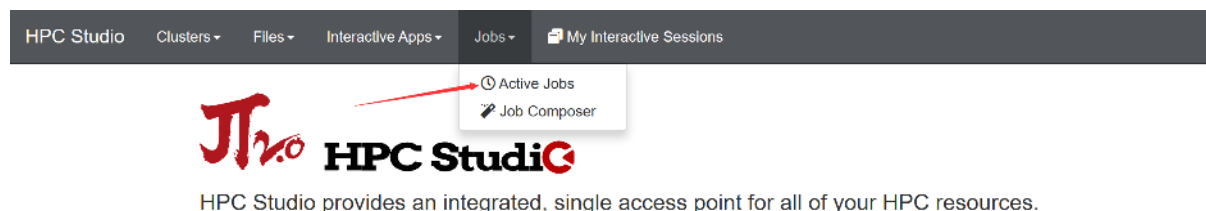
功能	详细功能	按钮
查看	点击文件可查看文件内容，点击文件夹可显示文件属性	1
编辑	仅可编辑文本文件，编辑文件有多种编辑器可选，均为系统嵌入，无需链接本地编辑器	2
重命名/移动	重命名或移动文件	3
下载	下载文件，若下载的为文件夹则会自动压缩成压缩包进行下载	4
拷贝/粘贴	复制文件	5 6
全选/取消全选	选择文件功能	7
进入...	进入到其他路径	8
打开终端	打开 webshell	9
新建文件	新建文件	10
新建目录	新建目录	11
上传	上传文件	12
显示隐藏文件	点击可显示隐藏文件/文件夹	13
显示属性信息	显示文件/文件夹属主/大小/日期等信息	14
删除	删除文件/文件夹	15

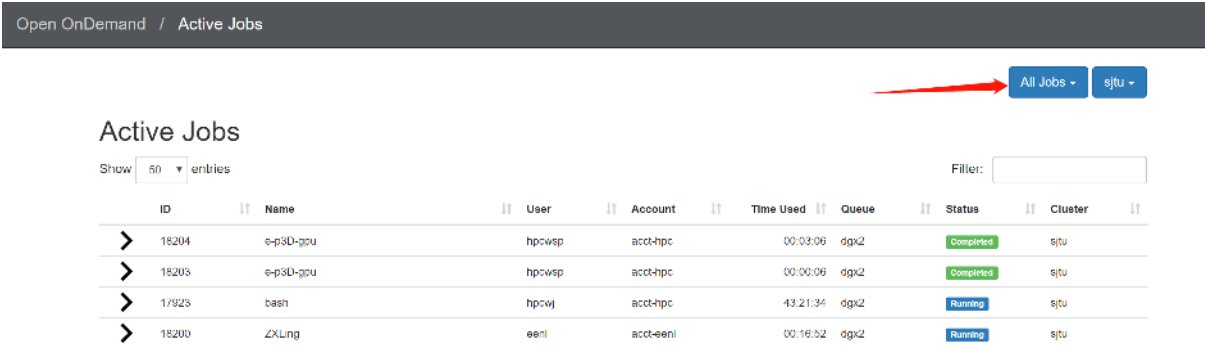


作业

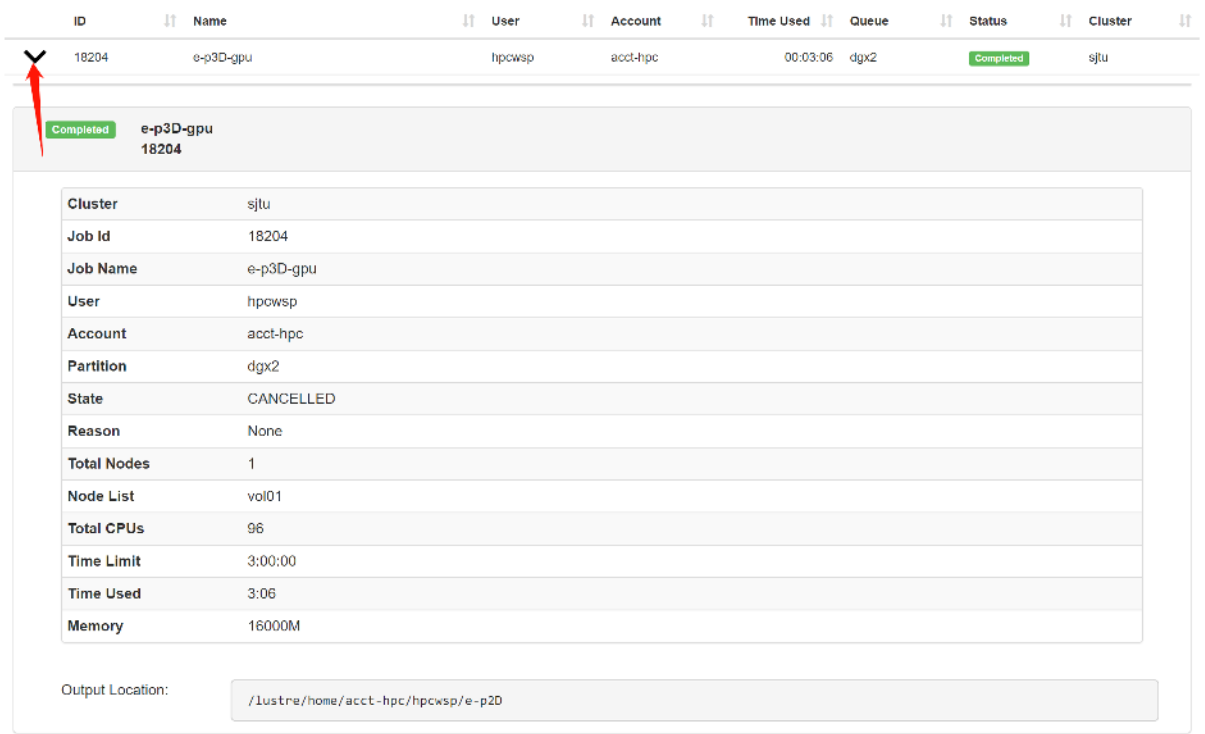
查看作业

点击 Jobs->Active Jobs，查看队列中的作业。在右上角的 Your Jobs/All Jobs 中选择要查看的任务类型。



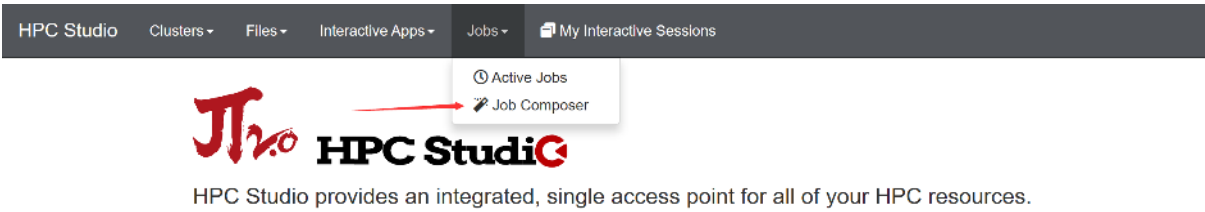


点击作业项前的箭头查看作业详情。

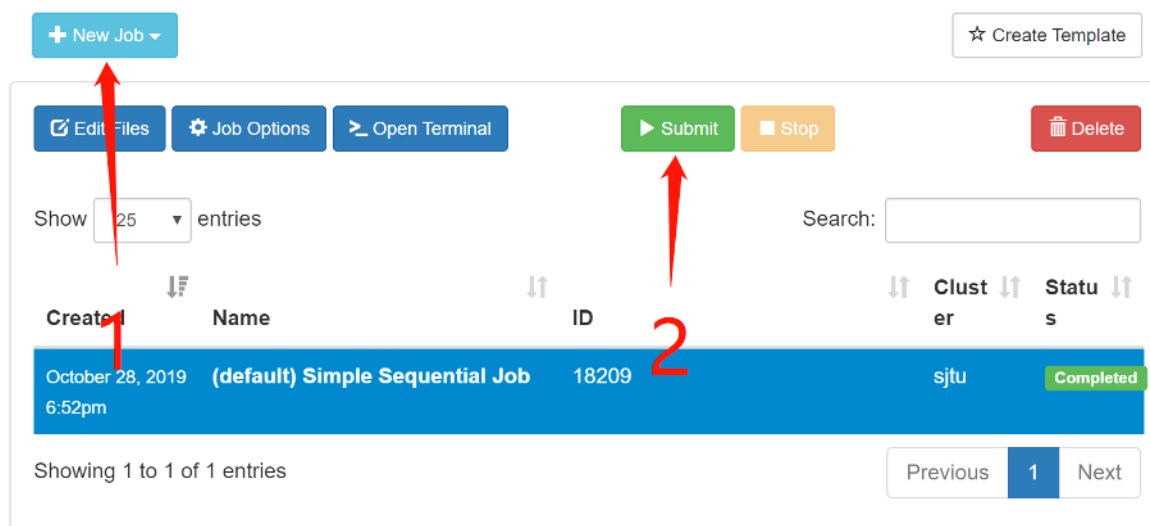


提交作业

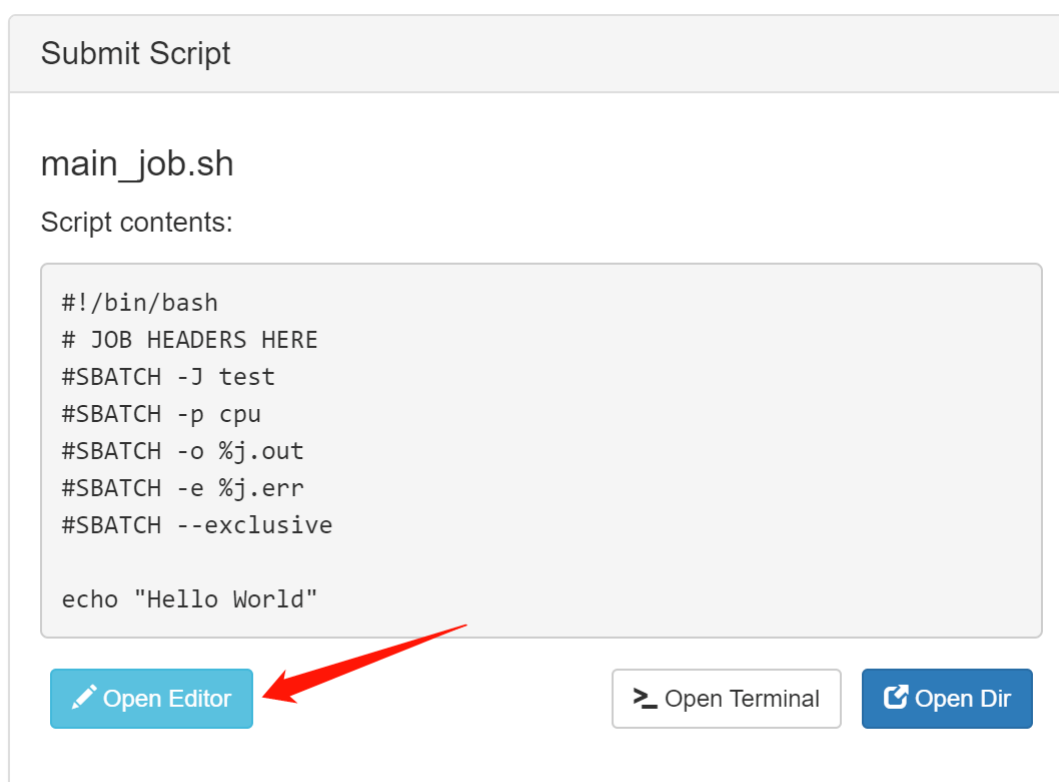
点击 Jobs->Job Composer，打开新建作业选项卡。



点击左上角 New Job (1) 按钮，选择模板，点击 submit (2) 按钮提交作业。

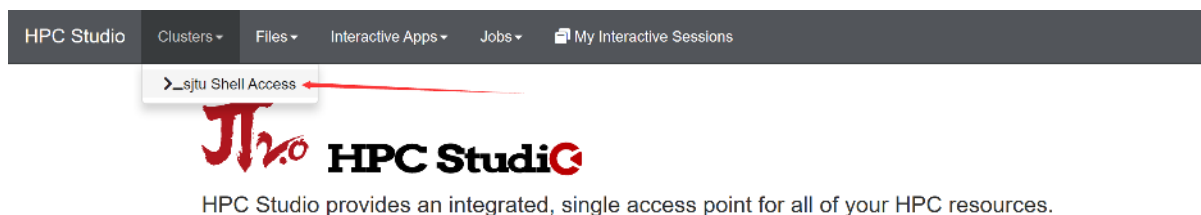


同时 HPC Studio 提供了在线文本编辑功能，在右侧底部的 Submit Script 选项中，点击 Open Editor 按钮，即可打开文本编辑器。



在线 Shell 终端

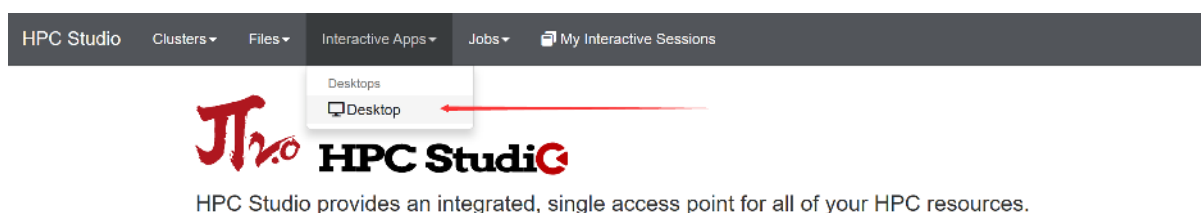
点击 Clusters->sjtu Shell Access, 打开在线终端。



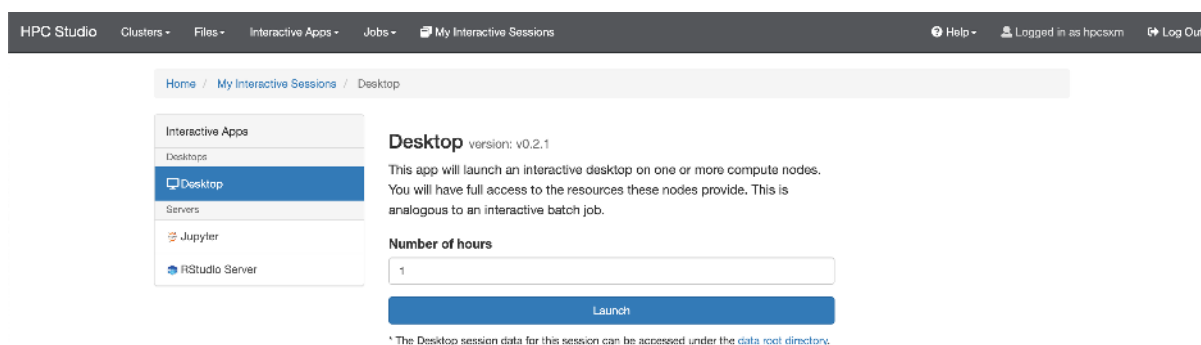
远程桌面

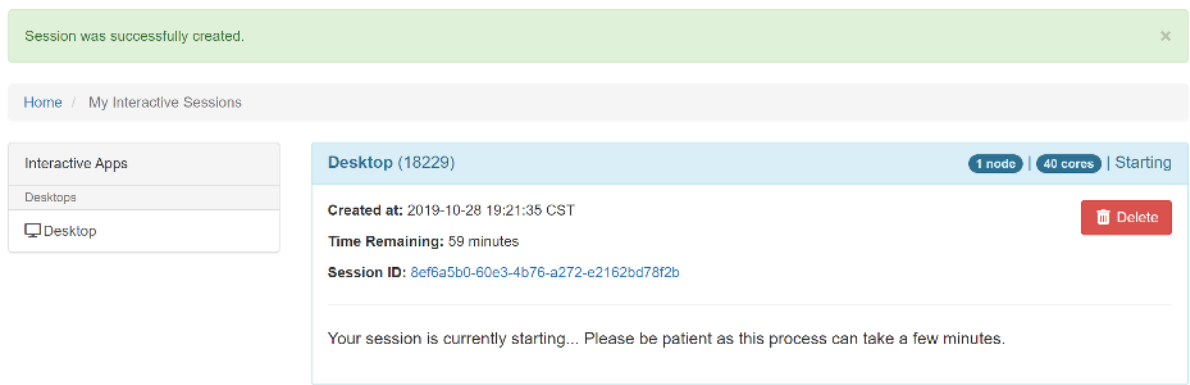
远程桌面打开方式需先提交一个空的作业取得计算节点的控制权（此操作会计入机时）。

点击 Interactive Apps->Desktop 按钮，进入作业提交页面。

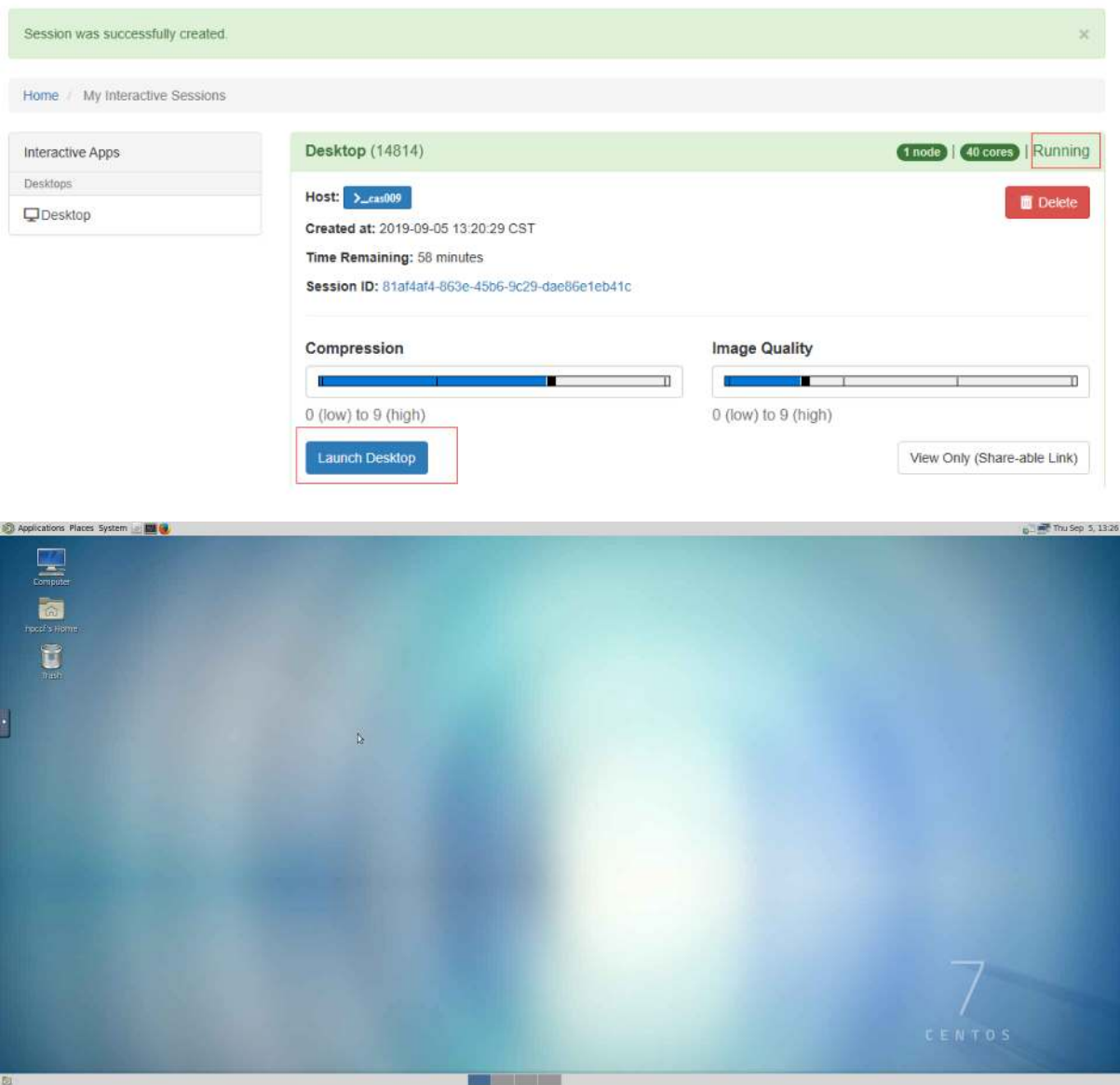


Number of hours 默认是 1，然后点击 launch 即可进入桌面选项卡。





待选项卡显示作业在 **running** 的状态时, 点击 **launch** 即可进入远程桌面。



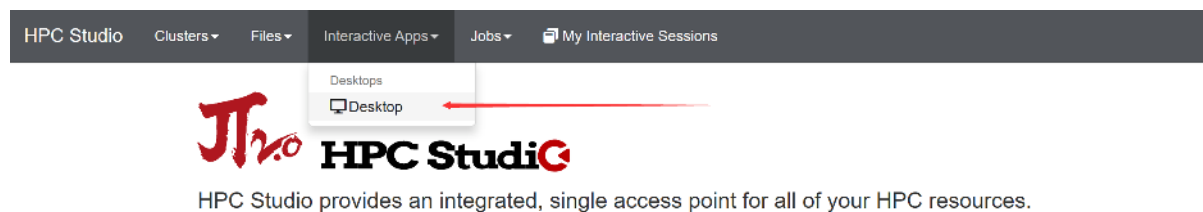
参考教学视频

2022 春季用户培训之 HPC Studio

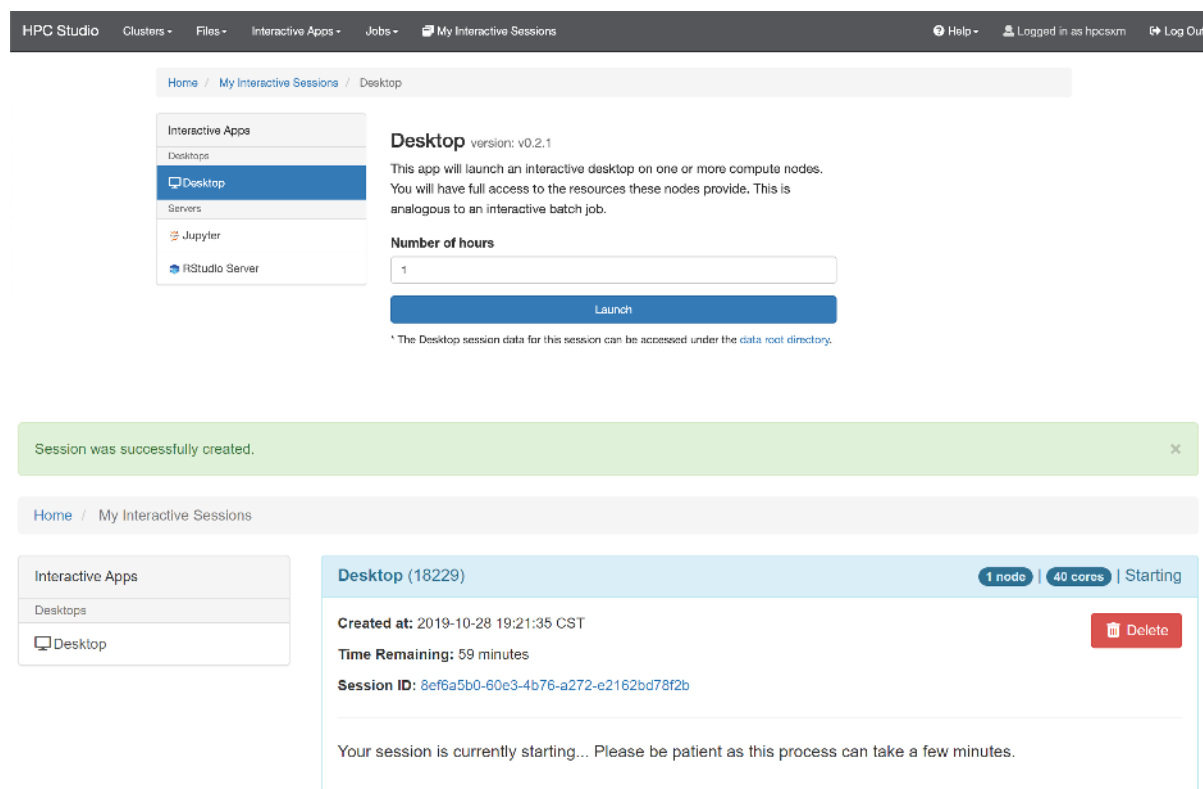
4.5.2 远程桌面

远程桌面打开方式需先提交一个空的作业取得计算节点的控制权（此操作会计入机时）。

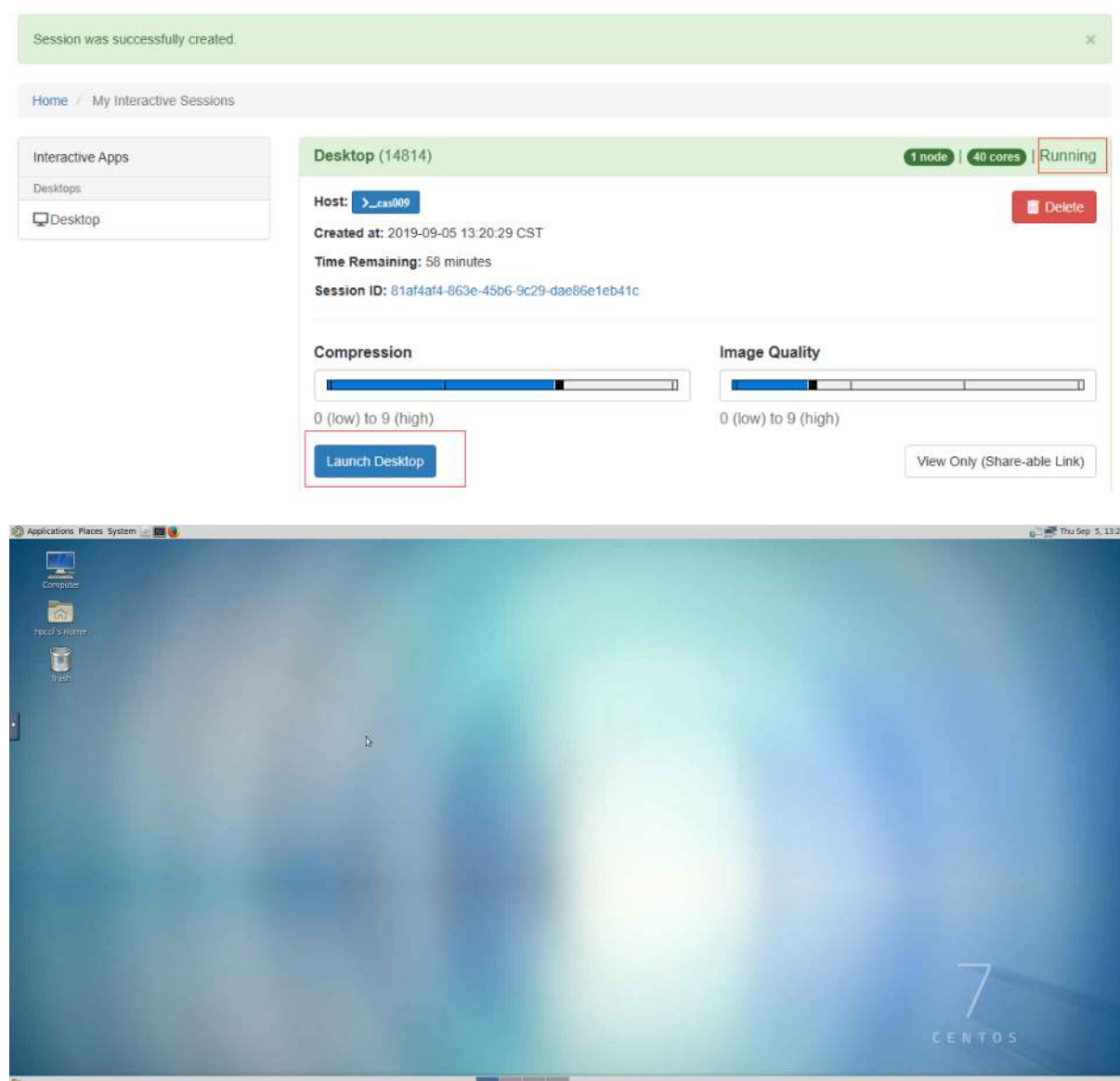
点击 Interactive Apps->Desktop 按钮，进入作业提交页面。



Number of hours 默认是 1，然后点击 launch 即可进入桌面选项卡。



待选项卡显示作业在 running 的状态时，点击 launch 即可进入远程桌面。



4.5.3 Jupyter

Jupyter 是一个非营利组织，旨在“为数十种编程语言的交互式计算开发开源软件，开放标准和服务”。2014 年由 Fernando Pérez 从 IPython 中衍生出来，Jupyter 支持几十种语言的执行环境。

Jupyter Project 的名称是对 Jupyter 支持的三种核心编程语言的引用，这三种语言是 Julia、Python 和 R，也是对伽利略记录发现木星的卫星的笔记本的致敬。Jupyter 项目开发并支持交互式计算产品 Jupyter Notebook、JupyterHub 和 JupyterLab，这是 Jupyter Notebook 的下一代版本。

登录 HPC Studio 平台后，可以在内置应用中选择 Jupyter 或 Jupyter (GPU)，均支持 Jupyter Notebook 和 JupyterLab。

在 **Jupyter** 中使用预置环境

已有三个预置环境，可供用户使用：

预置 **PyTorch** 环境

环境	版本
python	3.8.3
cupy	10.1.243
pytorch	1.5.0
torchvision	0.6.0
numpy	1.18.1
pandas	1.0.4
pillow	7.1.2
scipy	1.4.1
matplotlib	3.2.1
seaborn	0.10.1

预置 **TensorFlow** 环境

环境	版本
python	3.8.3
cupy	10.1.243
cudnn	7.6.5
tensorflow	2.2.0
tensorboard	2.2.2
numpy	1.18.5
pandas	1.0.4
pillow	7.1.2
scipy	1.4.1
matplotlib	3.2.1
seaborn	0.10.1

预置 **R** 环境

环境	版本
R	3.6.1

在 Jupyter 中使用自定义的环境

新建环境（或使用已有环境）：

```
$ module load miniconda3
$ conda create -n test-env
$ source activate test-env
```

安装并注册为 jupyter kernel：

```
(test-env) $ conda install ipykernel
(test-env) $ python -m ipykernel install --user --name test-env --display-
↪name "Test Environment"
```

然后可以在 Jupyter 中选择名为 Test Environment 的 **Kernel** 进行计算。

如果环境需要依赖 NVIDIA CUDA Toolkit 或 NVIDIA cuDNN，可以使用 conda 进行安装：

```
(test-env) $ conda install cudatoolkit=10.1 cudnn
```

在 Jupyter 中使用自定义 R 环境

新建环境（或使用已有环境）：

```
$ module load miniconda3
$ conda create -n r-test-env
$ source activate r-test-env
$ (r-test-env) $ conda install -c r r-essentials
```

安装并注册为 jupyter kernel：

```
(test-env) $ R
> install.packages('IRkernel')
> IRkernel::installspec(name = 'r-test-env', displayname = 'R 3.6.1')
```

然后可以在 Jupyter 中选择名为 R 3.6.1 的 **Kernel** 进行计算。

参考资料

- [Jupyter Wikipedia](#)
- [Jupyter Home](#)

4.5.4 RStudio

简介

RStudio 是一个集成开发环境，主要支持 R 编程语言，专用于统计计算和图形。它包括一个控制台，支持代码执行的语法编辑器，以及用于绘制，调试和管理工作区的工具。

可用的版本

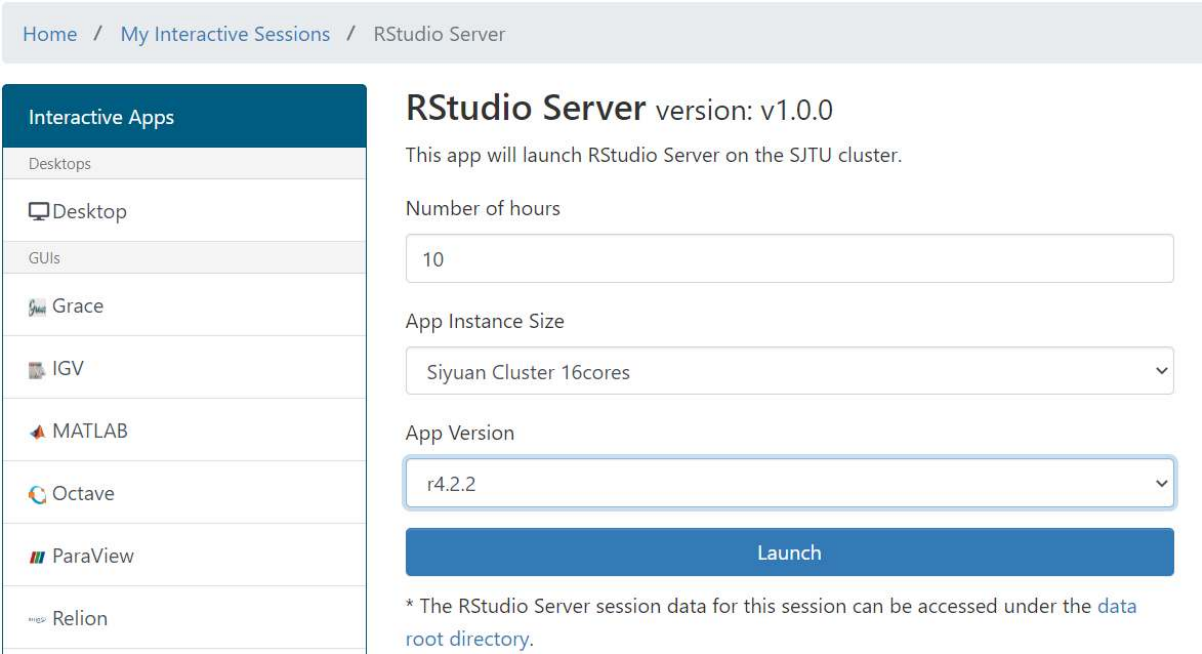
R 版本	平台	RStudio 版本
4.2.2	Studio	2022.12.0
4.1.3	Studio	2022.02.1
4.0.2	Studio	1.2.5042
3.6.3	Studio	1.2.5042

如何使用

使用超算的账号及密码登录 **HPC Studio**，在导航栏 Interactive Apps 中选择 RStudio Server，如下图：



点击后会出现相关的启动界面，可以设置作业时间，资源情况，软件版本。设置完成后 Launch 即可运行：



小技巧

π2.0 集群和思源一号的数据不互通，注意区分。

待界面从排队变成 Running 后，可通过 Connect to RStudio Server 连接到

Rstudio Server :

RStudio Server (23615220) **1 node** | **16 cores** | **Running**

Host: >_node007.pi.sjtu.edu.cn Delete

Created at: 2023-03-13 13:21:14 CST

Time Remaining: 9 hours and 57 minutes

Session ID: 33509e94-a1b6-41f0-8547-132723ebbb94

® Connect to RStudio Server

运行示例

所需的 R 依赖包需要自行安装:

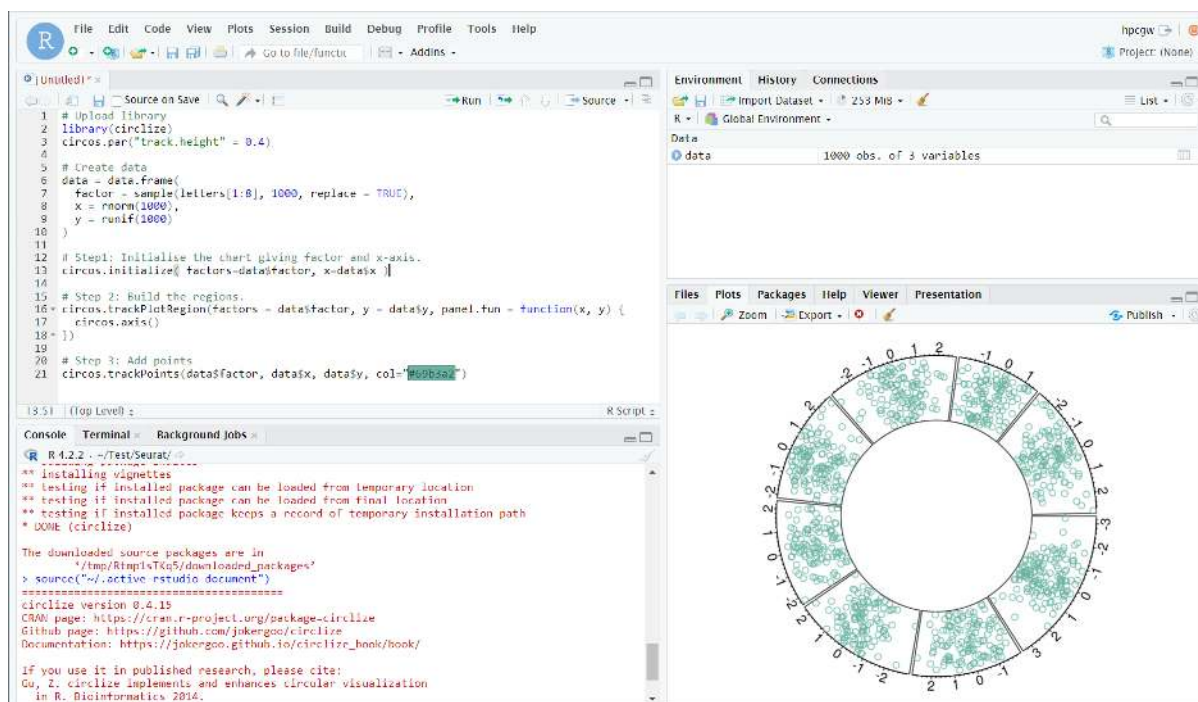
```
# Upload library
library(circlize)
circos.par("track.height" = 0.4)

# Create data
data = data.frame(
  factor = sample(letters[1:8], 1000, replace = TRUE),
  x = rnorm(1000),
  y = runif(1000)
)

# Step1: Initialise the chart giving factor and x-axis.
circos.initialize( factors=data$factor, x=data$x )

# Step 2: Build the regions.
circos.trackPlotRegion(factors = data$factor, y = data$y, panel.fun =
  ↪function(x, y) {
    circos.axis()
  })

# Step 3: Add points
circos.trackPoints(data$factor, data$x, data$y, col="#69b3a2")
```



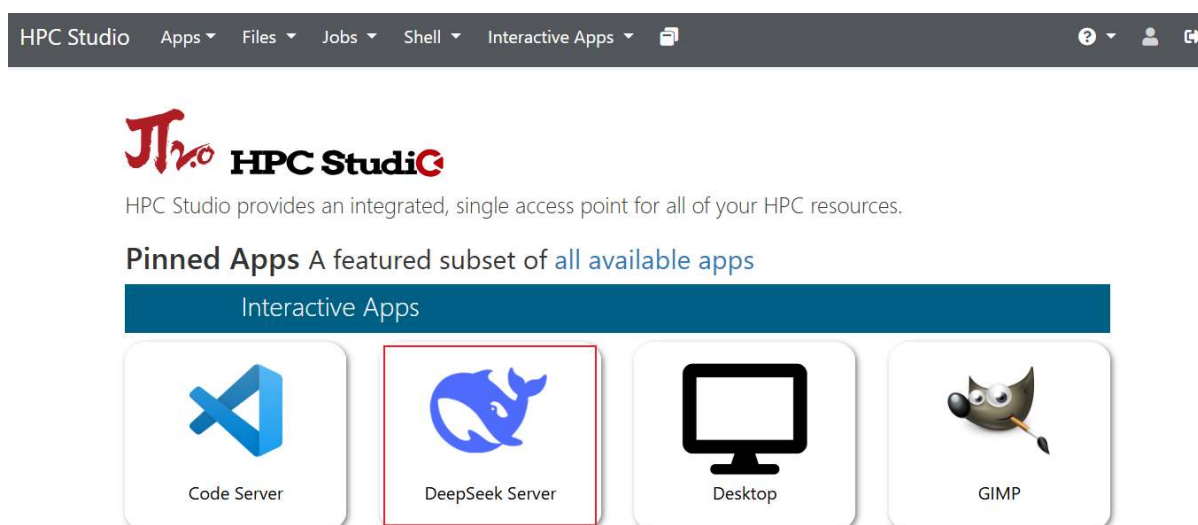
4.5.5 DeepSeek

简介

DeepSeek（深度求索）是由国内顶尖 AI 团队于 2023 年推出的通用人工智能（AGI）大模型。它通过更高效的算法架构，用仅 700 亿参数（对比 GPT-3 的 1750 亿）实现了同等甚至更强的性能表现。

如何使用

1. 使用超算账号密码或 jAccount 登录 HPC Studio 可视化平台，主界面选择 DeepSeek Server；



2. 设置使用时长，根据需求选择 1 个 CPU 节点或者 1 张 A100 加速卡后，点击 Launch；

DeepSeek Server version: v1.0.0

This app will launch deepseek-r1:14b Server on the SJTU cluster.

Number of hours

10

App Instance Size

Siyuan Cluster 1gpu

Siyuan Cluster 64cores

Siyuan Cluster 1gpu

r1:14b

☐ I would like to receive an email when the session starts

Mail

Launch

* The DeepSeek Server session data for this session can be accessed under the [data root directory](#).

3. 等待作业排队完成，即可连入 DeepSeek 服务。

DeepSeek Server (41277775)

1 node | 16 cores | Running

Host: [_gpu06.pi.sjtu.edu.cn](#)

Delete

Created at: 2025-02-08 13:10:55 CST

Time Remaining: 6 hours and 8 minutes

Session ID: 26b27ba8-5834-4bc9-b769-2fc4d467ef94

Connect to DeepSeek Server

DeepSeek Chat

CRISPR基因编辑技术的主要风险有哪些?

Thinking...

嗯，我刚才在了解一下CRISPR基因编辑技术的主要风险，我对这个技术有点模糊的概念，知道它是一种能够精确修改DNA的工具，常用于科学研究和医学治疗中。不过，具体的风险是什么，我还不清楚。

首先，我记得之前听说过脱靶效应，就是说CRISPR可能会在不应该改变的地方切割DNA，导致意外的突变，这可能引起一些健康问题，比如癌症或者遗传疾病。那为什么会发生脱靶效应呢？可能是因为引导RNA和DNA序列有部分相似，但不完全匹配，所以Cas9酶也会切割这些地方。

接下来是mRNA编辑的问题，CRISPR有时候可能会在转录过程中发挥作用，也就是直接编辑mRNA而不是DNA。这样虽然暂时影响蛋白质的生产，但如果处理不当，可能会干扰细胞正常的功能，甚至导致细胞死亡或者异常生长。

还有一个风险是基因编辑的不稳定性，比如双链断裂后修复错误可能导致染色体易位或者其他结构变异，这会增加癌症的风险。我记得之前有个研究说使用CRISPR会导致一些细胞出现这样的问题，所以控制好修复机制可能很重要。

Message

Type your message here...

Send

Select Model

deepseek-r1:4b

Model Name to Pull from the Ollama site

Enter the model name to pull...

Pull Model

Pull Status

Temperature

0.7

Clear Chat

如果遇到问题请发送邮件至 [HPC 邮箱](#)。

4.5.6 Paraview

ParaView 是对二维和三维数据进行分析 and 可视化的程序，它既是一个应用程序框架，也可以直接使用（Turn-Key）。ParaView 支持并行，可以运行于单处理器的工作站，也可以运行于分布式存储器的大型计算机。ParaView 用 C++ 编写，基于 VTK (Visualization ToolKit) 开发，图形用户界面用 Qt 开发，开源、跨平台。ParaView 用户可以迅速的建立起可视化环境利用定量或者是定性的手段去分析数据。利用它的批量处理能力可以在三维或者是在报表中交互进行“数据挖掘”。

Paraview 的基本使用

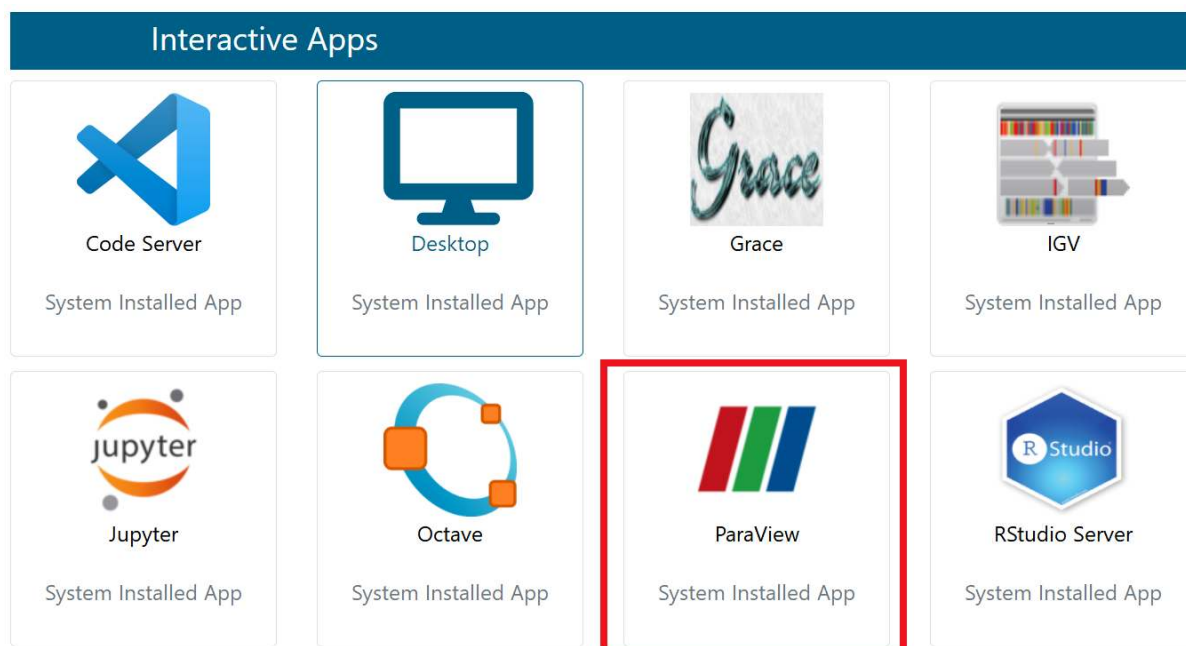
1. 首先根据 [OpenFOAM 使用说明文档](#) 的步骤运行 cavity 算例，运行成功后在对应 cavity 目录下会得到如下目录：

```
0
0.1
0.2
0.3
0.4
0.5
constant
system
```

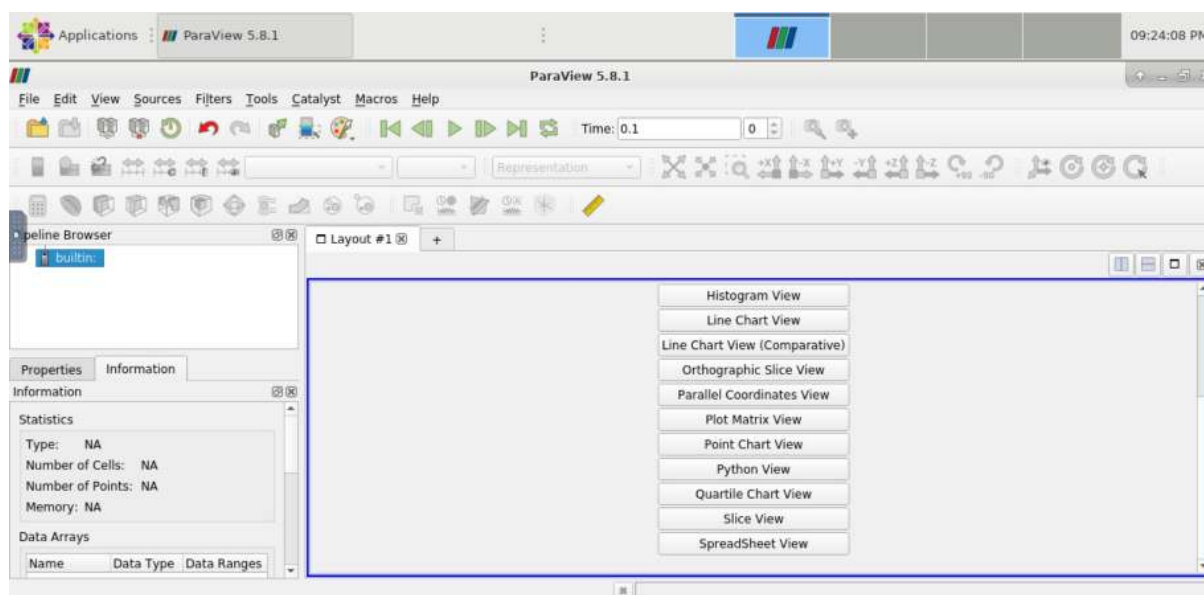
2. 在 cavity 目录下新建 mycase.foam 文件 (内容为空):

```
touch mycase.foam
```

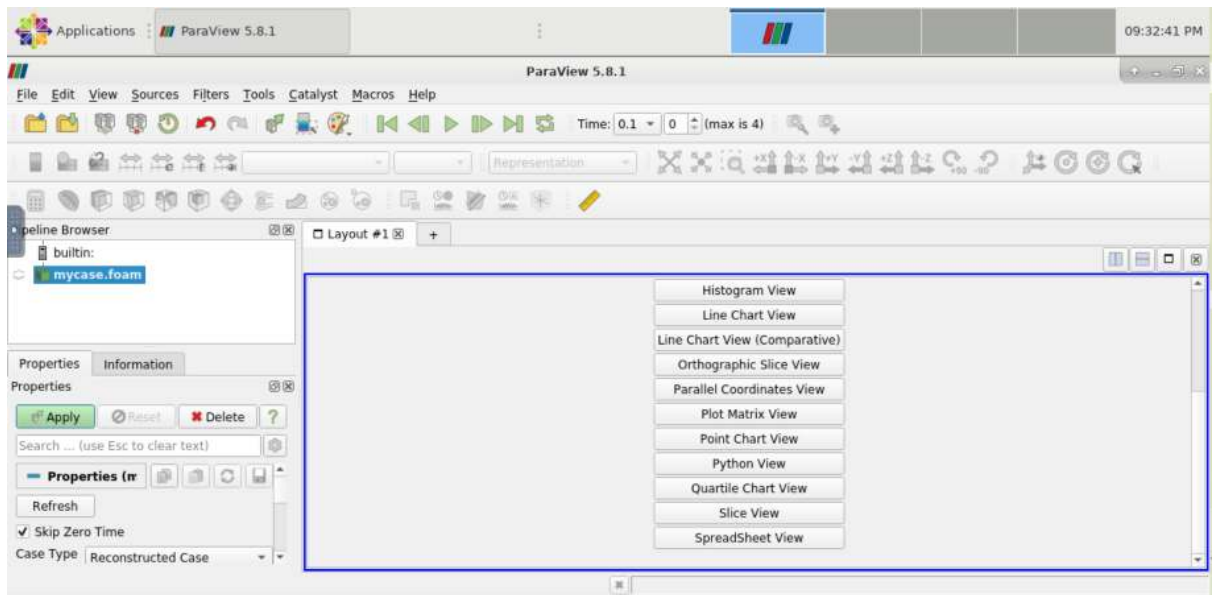
3. 用 pi 集群帐号登录 [HPC Studio](#) 平台。然后在 Interactive Apps 面板中点击如下图所示 ParaView 选项申请 Paraview GUI 界面 (要排一会儿队，请耐心等待):



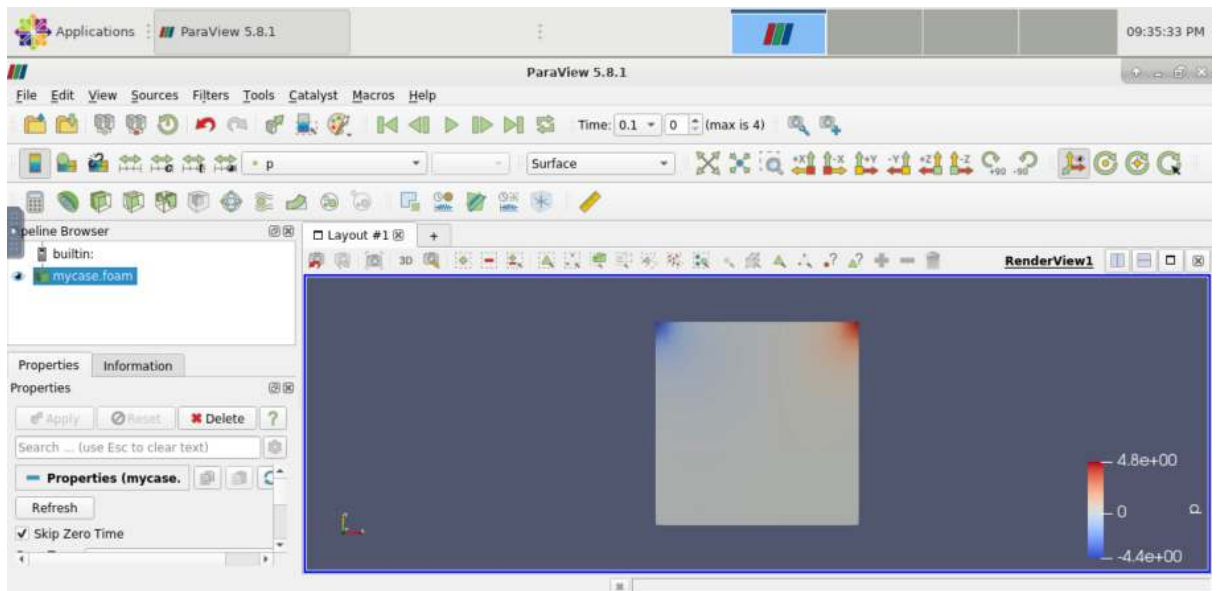
4. 申请成功后会看到如下界面：



5. 点击左上角 File->Open，打开刚才在 cavity 目录中新建的 mycase.foam 文件。然后点击左下部 Properties->Apply 按钮，如下图所示：



6. 结果如图所示 (运行 cavity 算例得到的压强场分布):



Paraview 的插件配置

本插件配置教程以 PVGeo 为例。

以下操作均在 home 目录完成，如需安装至其他目录，请自行修改相应路径。

Paraview 的本地安装

```
$ cd ~
```

首先下载 5.6.1 版本的 paraview(请勿下载其他版本，否则可能会导致 PVGeo 与当前版本不兼容)：

```
$ wget "https://www.paraview.org/paraview-downloads/download.php?
↳submit=Download&version=v5.6&type=binary&os=Linux&downloadFile=ParaView-5.
↳6.1-MPI-Linux-64bit.tar.gz"
```

解压:

```
$ tar -x -f ParaView-5.6.1-MPI-Linux-64bit.tar.gz
```

改变文件目录名称:

```
$ mv ParaView-5.6.1-MPI-Linux-64bit paraview5.6
```

在虚拟环境下安装 **PVGeo**

虚拟环境的 **python** 版本必须为 **2.7**, **PVGeo** 的版本必须为 **2.0.0**.

```
$ module load miniconda3
$ conda create -n pvgeoenv python=2.7
$ source activate pvgeoenv
(pvgeoenv)$ pip install imageio==2.0.0
(pvgeoenv)$ pip install PVGeo==2.0.0
```

修改环境变量:

执行命令:

```
(pvgeoenv)$ python -m pvgeoenv install
```

会得到输出

```
PYTHONPATH=/lustre/home/YOUR_ACCT/YOUR_USERNAME/.conda/envs/pvgeoenv/lib/
↳python2.7/site-packages
PV_PLUGIN_PATH=/lustre/home/YOUR_ACCT/YOUR_USERNAME/.conda/envs/pvgeoenv/
↳lib/python2.7/site-packages/PVPlugins/
```

其中 `/lustre/home/YOUR_ACCT/YOUR_USERNAME/` 指向你当前的 **home** 目录。进入 **home** 目录下的 **.bashrc** 文件添加以下几行:

```
export PATH=~/.paraview5.6/bin:$PATH
export LD_LIBRARY_PATH=~/.paraview5.6/lib:$LD_LIBRARY_PATH
export PYTHONPATH=/lustre/home/YOUR_ACCT/YOUR_USERNAME/.conda/envs/pvgeoenv/
↳lib/python2.7/site-packages:$PYTHONPATH
export
PV_PLUGIN_PATH=/lustre/home/YOUR_ACCT/YOUR_USERNAME/.conda/envs/pvgeoenv/
↳lib/python2.7/site-packages/PVPlugins/
```

退出虚拟环境, 加载环境变量:

```
(pvgeoenv) $conda deactivate  
$ source ~/.bashrc
```

启动 **paraview**

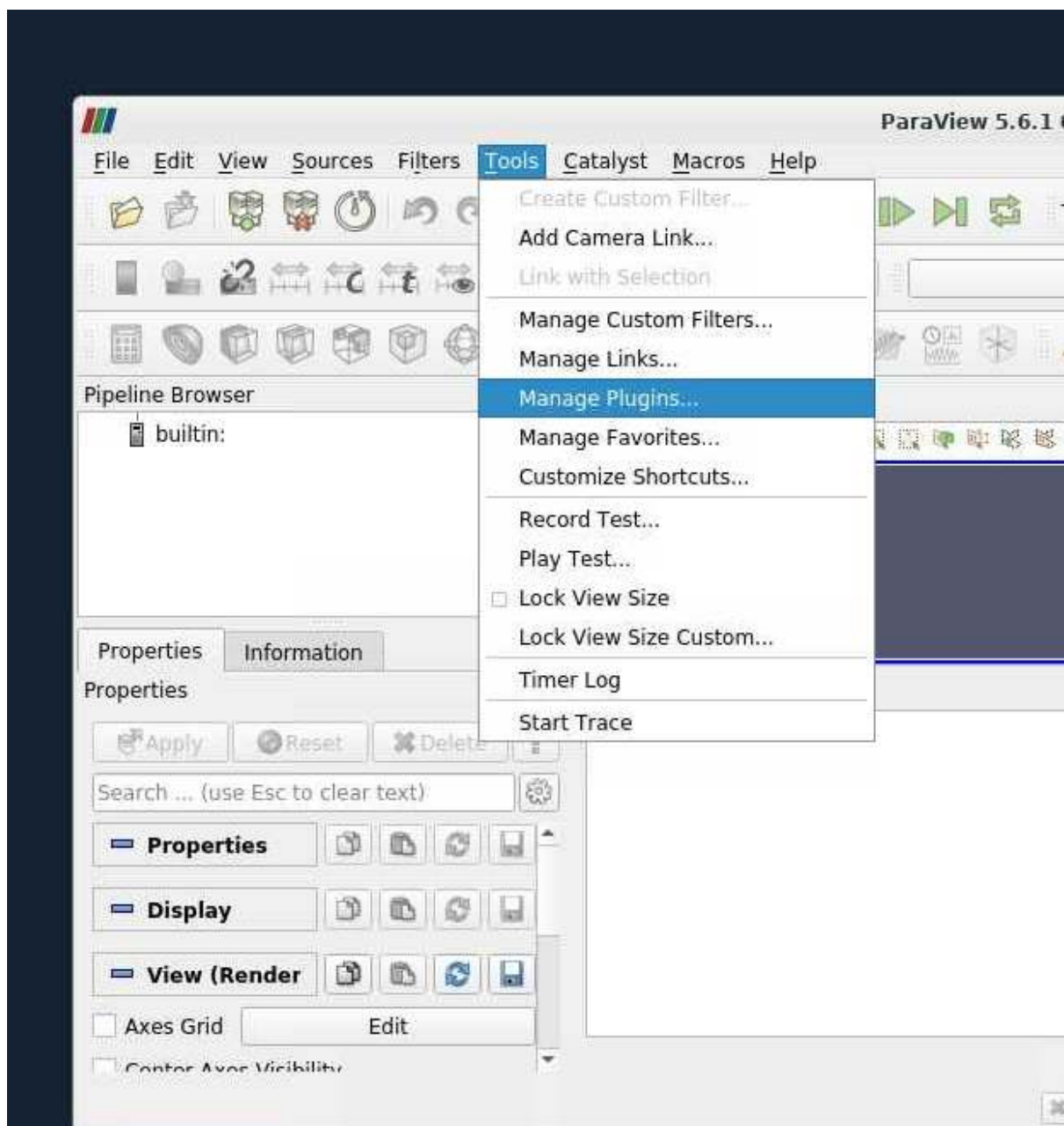
由于 **paraview** 的启动需要 GUI 的支持，需要进入超算的可视化平台，申请一个远程桌面。

打开 **studio desktop** 中的终端。由于 5.6 版本的 **paraview** 不能正确检测 **MESA** 的版本，在调用时需要添加参数：

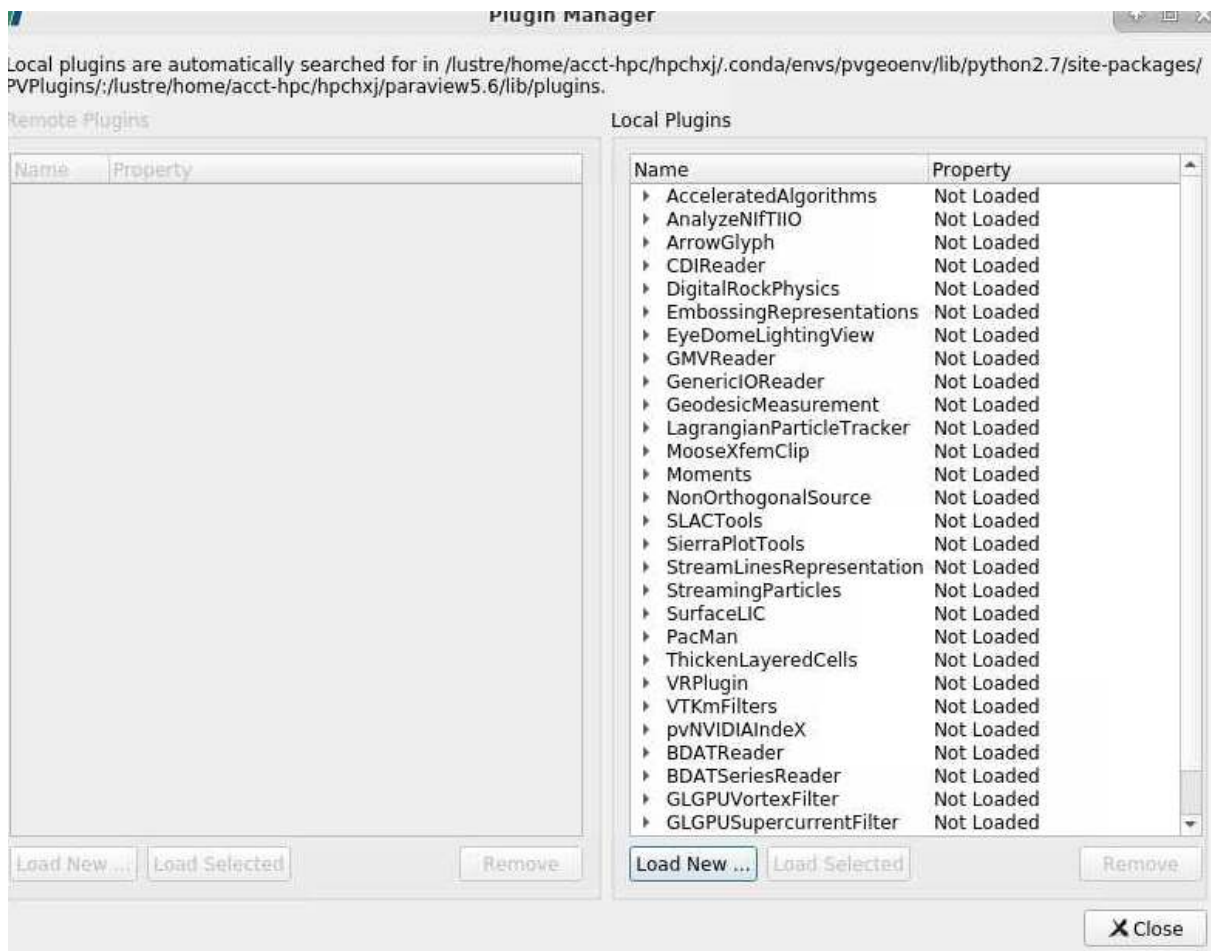
```
$ MESA_GL_VERSION_OVERRIDE=3.3 paraview
```

添加 **PVGeo** 插件

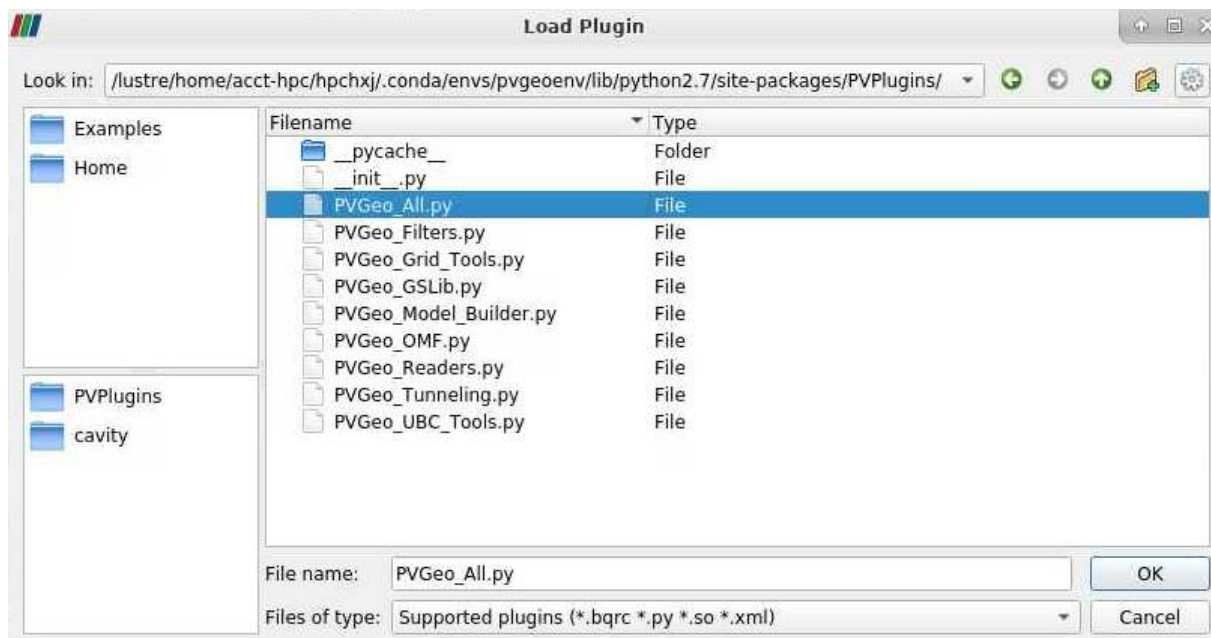
打开 **tools-manage plugins** 界面



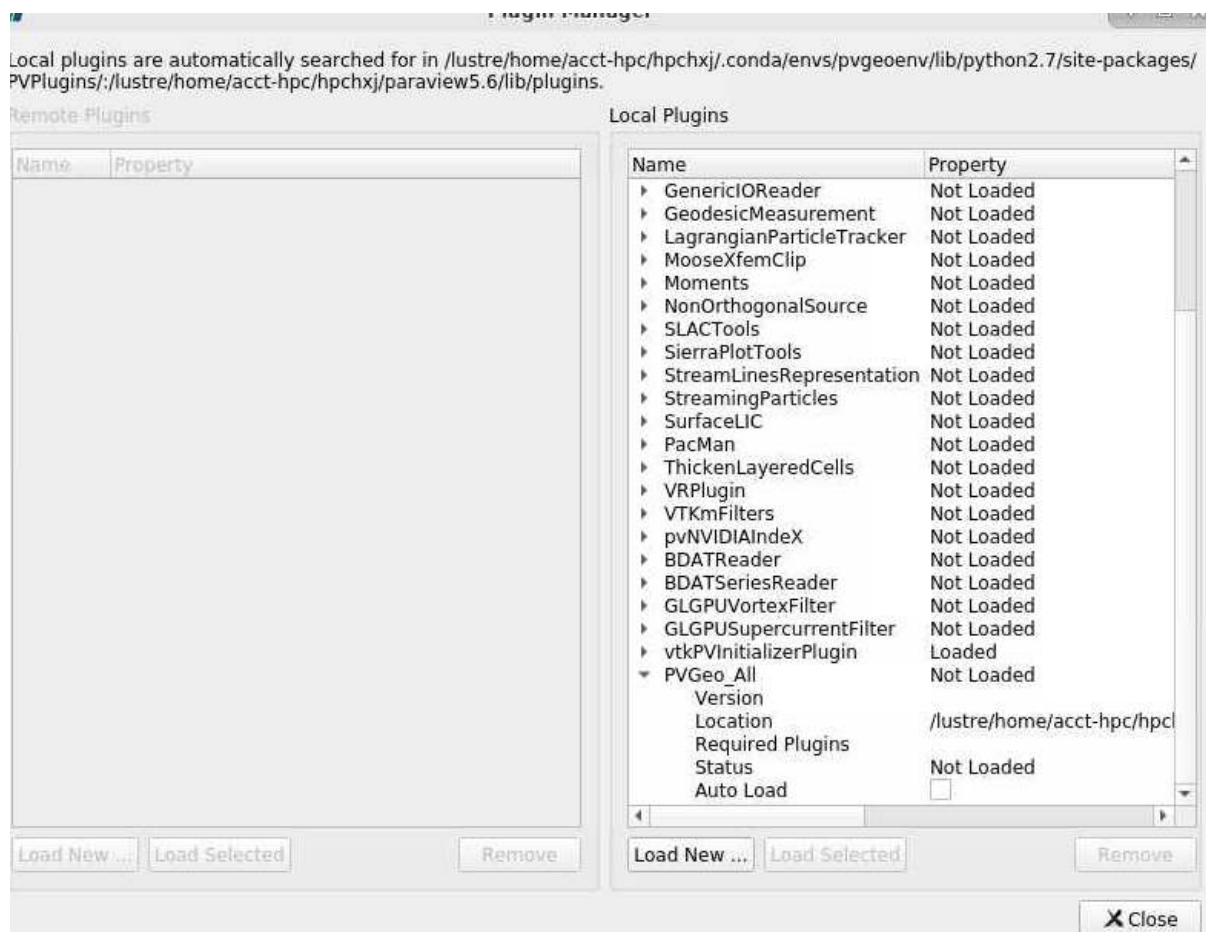
点击右下角 Load New…按钮



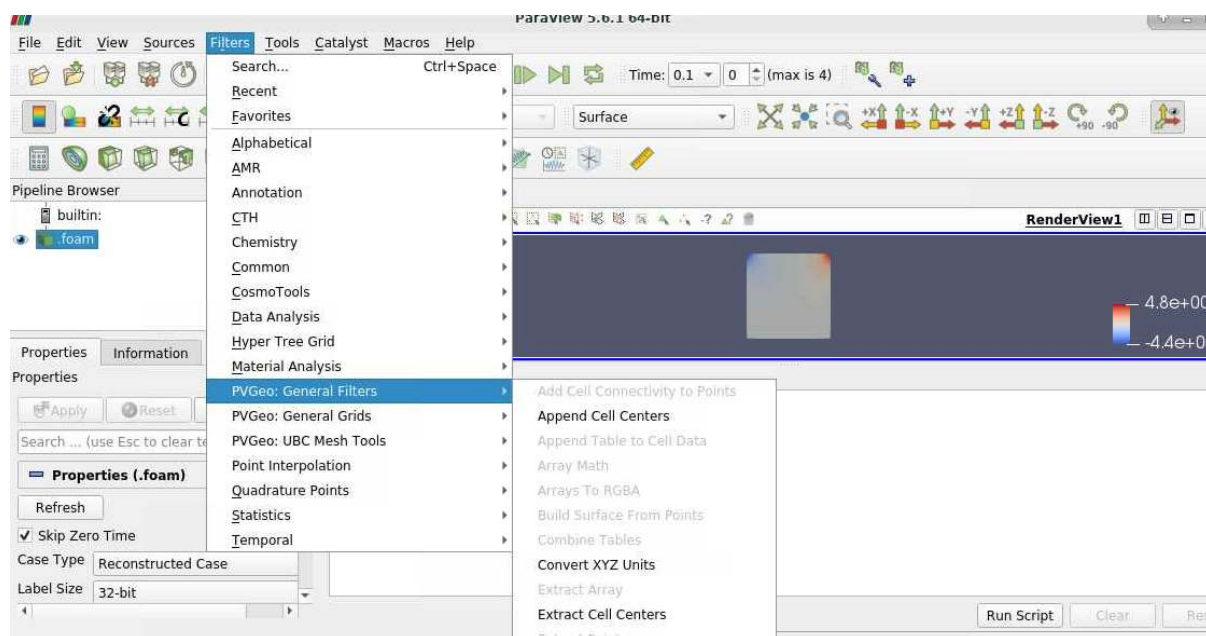
加载 PVGeo_All.py 文件：



等待几分钟，即可成功加载该插件：



此时可在 Filters 中看到 PVGeo 的 filter:

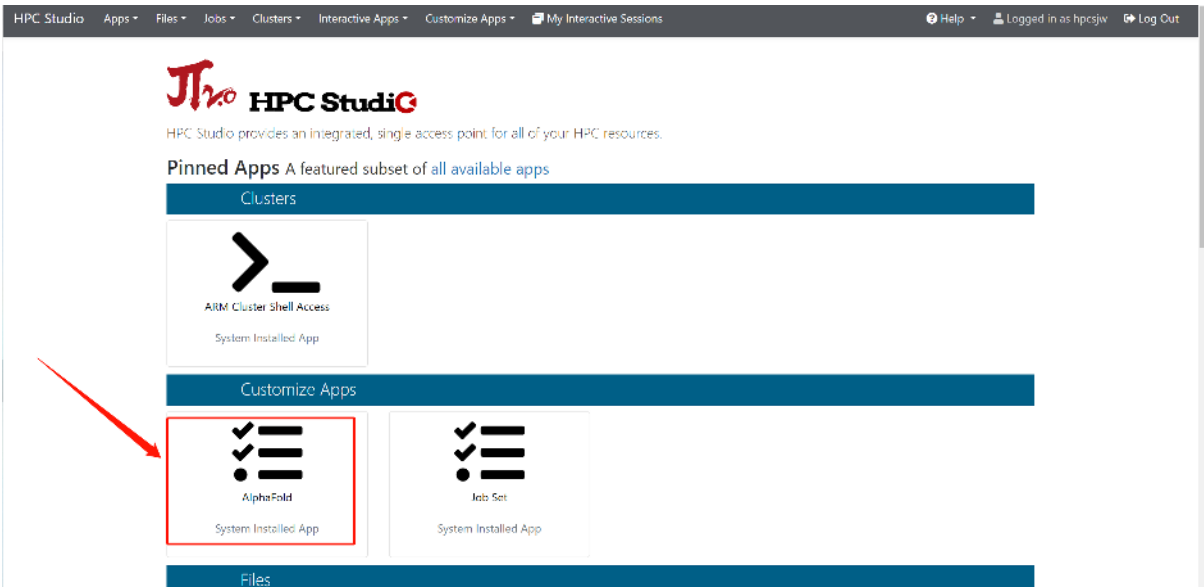


参考资料

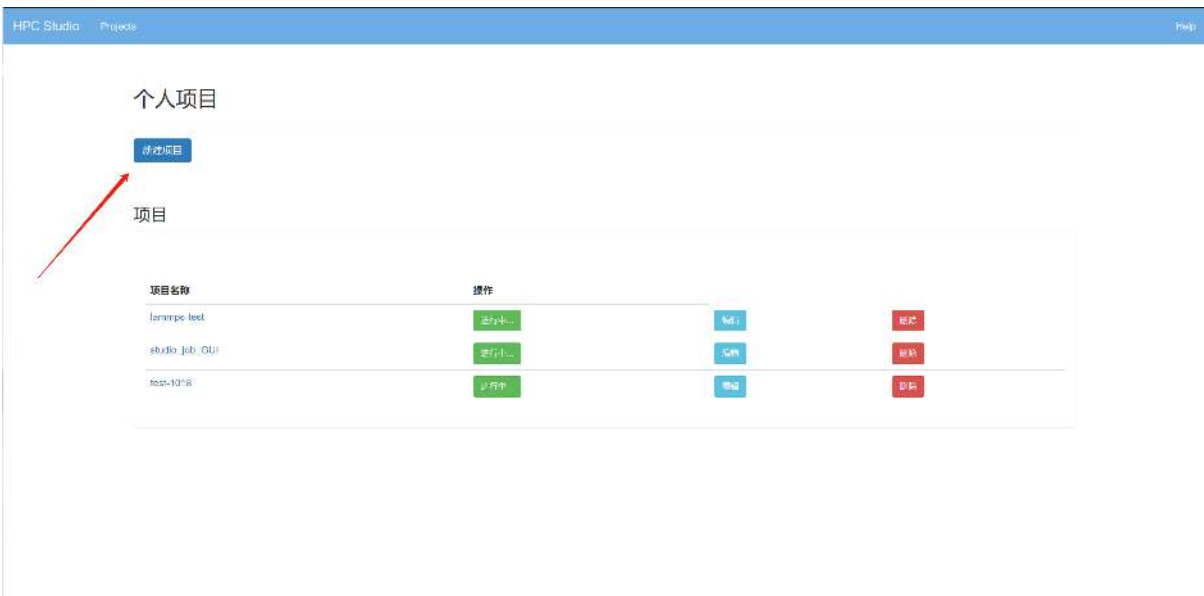
- PVGeo 官方配置教程 <https://pvgeo.org/overview/getting-started.html>。

4.5.7 AlphaFold-GUI

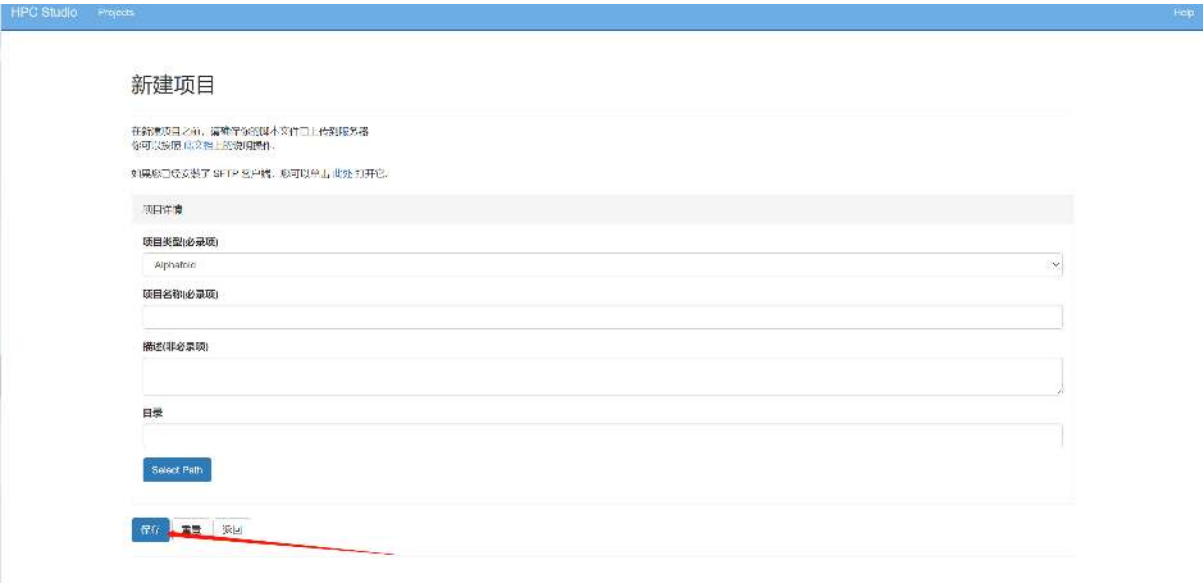
登录 HPC Studio 平台后，在自定义软件中选择 AlphaFold，如下图



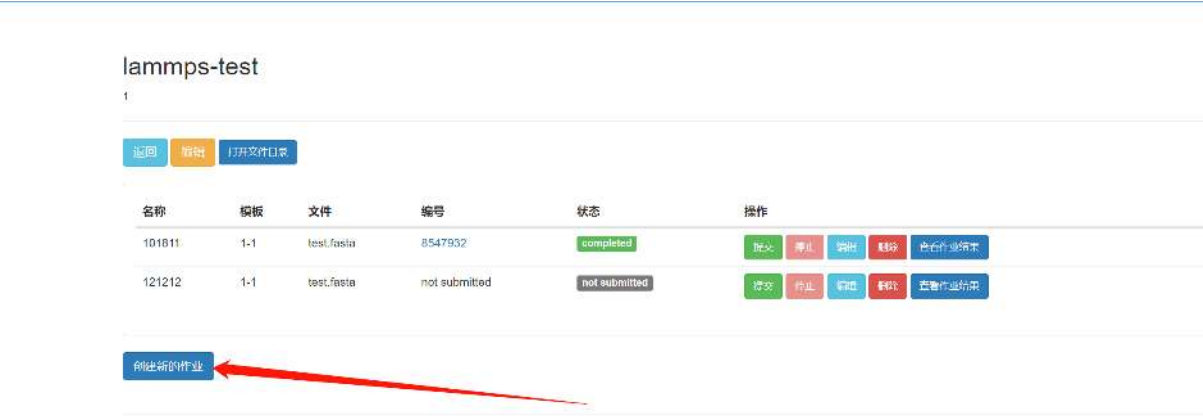
点击后会出现下图的选项卡，里面定义好了所需的选项卡，可以创建个人项目，(注: 项目下可以创建多个作业，作业运行结果保存在该项目路径下)



创建项目选项卡如下图所示，类型选择 AlphaFold，按要求填写表项点击”保存“



点击创建好的项目，选择“创建新的作业”按钮，如下图：



编辑作业，完成后点击”保存”，注意脚本文件路径，会自动弹出蛋白序列文件，如下图：

点击“查看作业结果”，可以查看所有作业的运行结果，如下如：

8547932 [completed]	
作业	8547932
脚本文件	/usr/share/anaconda3/bin/conda
创建时间	October 19, 2021 5:11pm
最后更新时间	October 19, 2021 5:54am

参考资料

- AlphaFold Operation Manual <https://docs.hpc.sjtu.edu.cn/app/bioinformatics/alphafold2.html?highlight=alpha>

4.5.8 MATLAB

简介

MATLAB 是美国 MathWorks 公司出品的商业数学软件，用于数据分析、无线通信、深度学习、图像处理与计算机视觉、信号处理、量化金融与风险管理、机器人，控制系统等领域。

可用的版本

集群平台	模块名
思源一号	module load matlab/r2022a
思源一号	module load matlab/r2022b
思源一号	module load matlab/r2023a
思源一号	module load matlab/r2023b
思源一号	module load matlab/r2024a
思源一号	module load matlab/r2024b
思源一号	module load matlab/r2025a
pi 2.0	module load matlab/r2022a
pi 2.0	module load matlab/r2022b
pi 2.0	module load matlab/r2023a
pi 2.0	module load matlab/r2023b
pi 2.0	module load matlab/r2024a
pi 2.0	module load matlab/r2024b
pi 2.0	module load matlab/r2025a

算例下载

```
cd ~
git clone https://github.com/SJTU-HPC/HPCTesting.git
```

使用 **MATLAB** 的方式

- 命令行交互式使用 **MATLAB**
- 提交 **MATLAB** 脚本
- 可视化平台部署的 **MATLAB**
- 可视化平台桌面启动 **MATLAB**
- 使用 **GPU** 版本的 **MATLAB**
- 多节点并行版的 **MATLAB**

命令行交互式使用 **MATLAB**

命令行使用 **MATLAB**，首先在超算平台上申请交互式会话窗口（以思源平台为例）：

```
$ srun -p 64c512g -n 10 --pty /bin/bash
```

输入以下命令进入 **MATLAB** 命令行交互式会话窗口：

```
$ module load matlab/r2022b
$ matlab
MATLAB is selecting SOFTWARE OpenGL rendering.
```

(续下页)

(接上页)

```

< M A T L A B (R) >
Copyright 1984-2022 The MathWorks, Inc.
R2022b Update 5 (9.13.0.2193358) 64-bit (glnxa64)
February 10, 2023

>>

```

在此交互式窗口内，可以执行 **MATLAB** 命令：

```

>> a=4;
>> b=5;
>> a+b

ans =

     9

>>

```

提交 **MATLAB** 脚本

π 超算提交单核 CPU 脚本

```

#!/bin/bash
#SBATCH -J matlab_test
#SBATCH -p small
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --ntasks-per-node=1

module load matlab/r2022a
cd ~/HPCTesting/matlab/case1
matlab -r test

```

思源一号提交单核 CPU 脚本

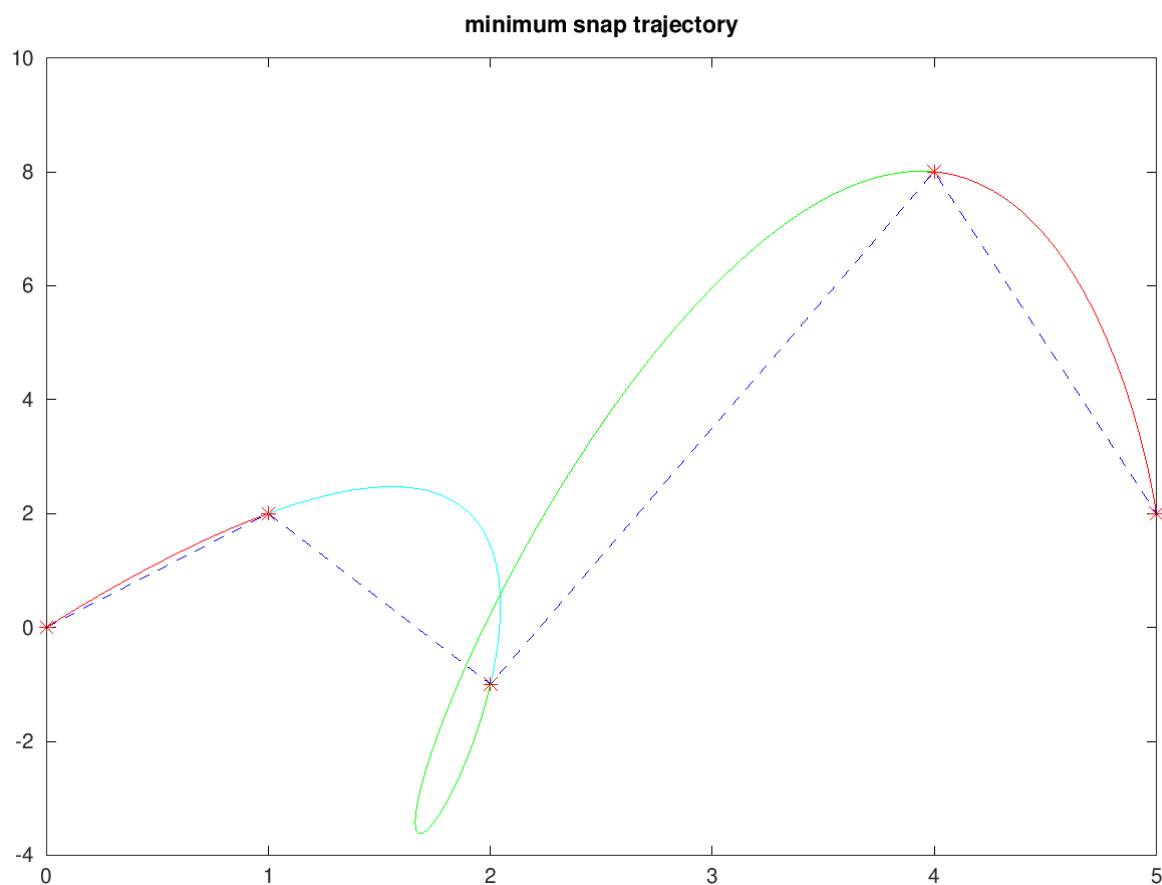
```

#!/bin/bash
#SBATCH -J matlab_test
#SBATCH -p 64c512g
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --ntasks-per-node=1

module load matlab/r2022a
cd ~/HPCTesting/matlab/case1
matlab -r test

```

使用 `sbatch` 命令提交脚本，脚本运行完毕后，在本地将生成一张名为 `1.png` 的图片，如程序运行无误，该图片的内容与本地 `result.png` 内容一致：



π 超算提交多核 CPU 脚本

```
#!/bin/bash
#SBATCH -J matlab_test
#SBATCH -p cpu
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 40
#SBATCH --cpus-per-task 1

module load matlab/r2022a
cd ~/HPCTesting/matlab/case2
matlab -r multicore
```

思源一号提交多核 CPU 脚本

```
#!/bin/bash
#SBATCH -J matlab_test
#SBATCH -p 64c512g
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
```

(续下页)

(接上页)

```
#SBATCH --cpus-per-task 64

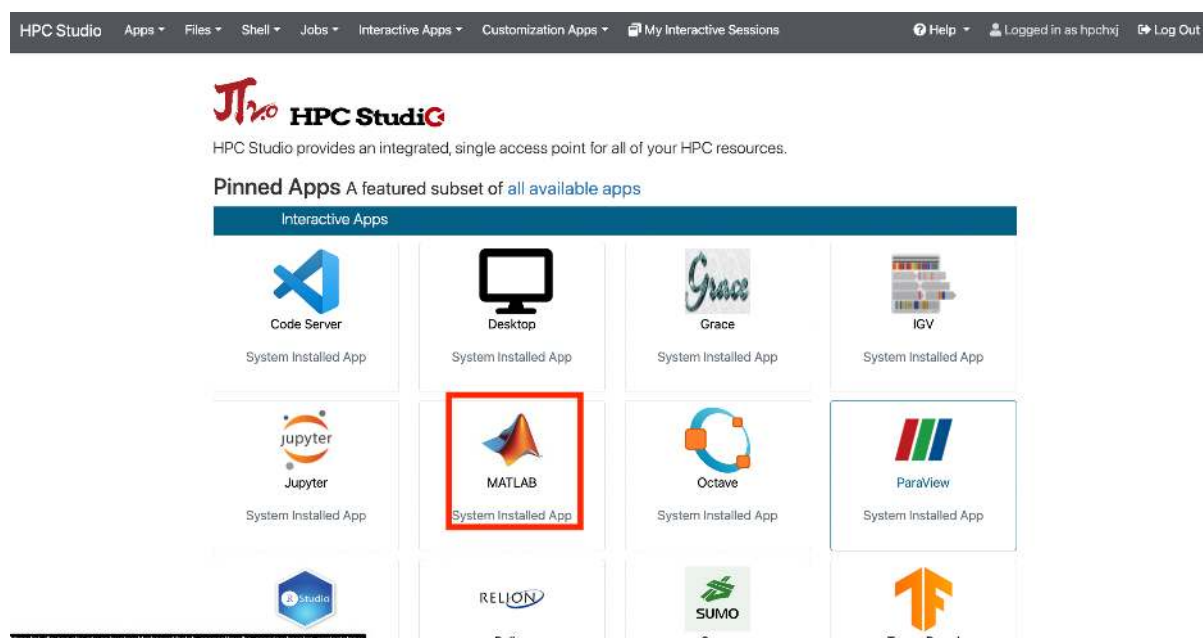
module load matlab/r2022a
cd ~/HPCTesting/matlab/case2
matlab -r multicore
```

可视化平台部署的 **MATLAB**

可视化平台部署了 **MATLAB** 应用，可运行 **MATLAB** 自带的可视化界面，进行交互操作。

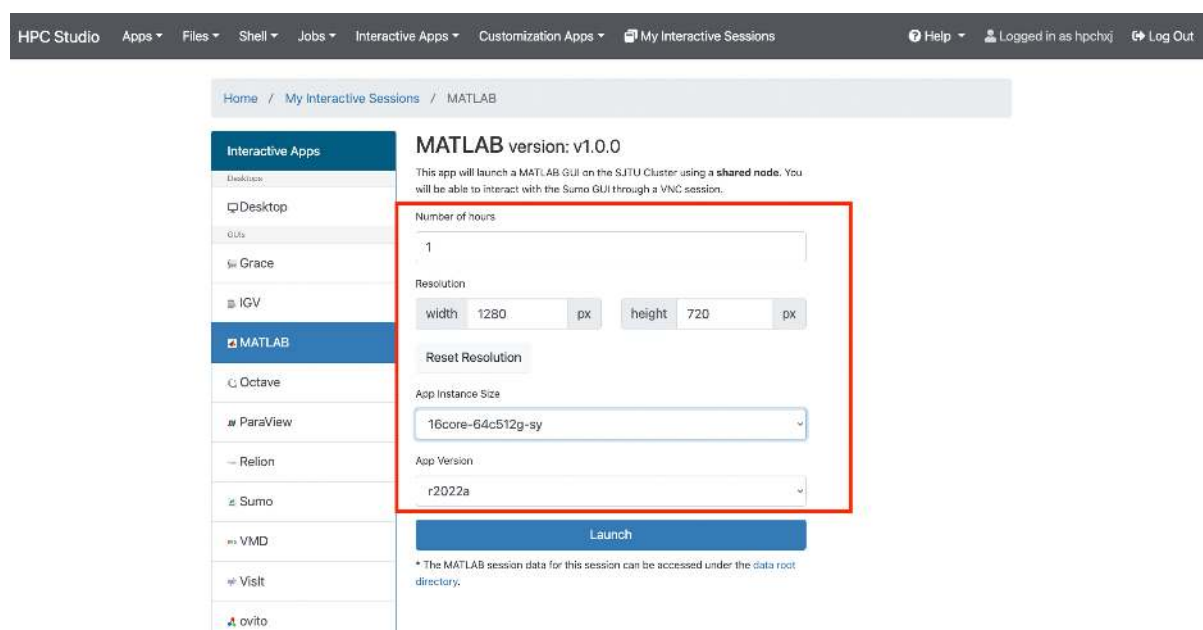
1. 登录可视化平台，选择 **MATLAB** 应用

使用 **hpc** 帐号登录 **HPC studio** (<https://studio.hpc.sjtu.edu.cn>) 后，点击 **MATLAB** 应用图标



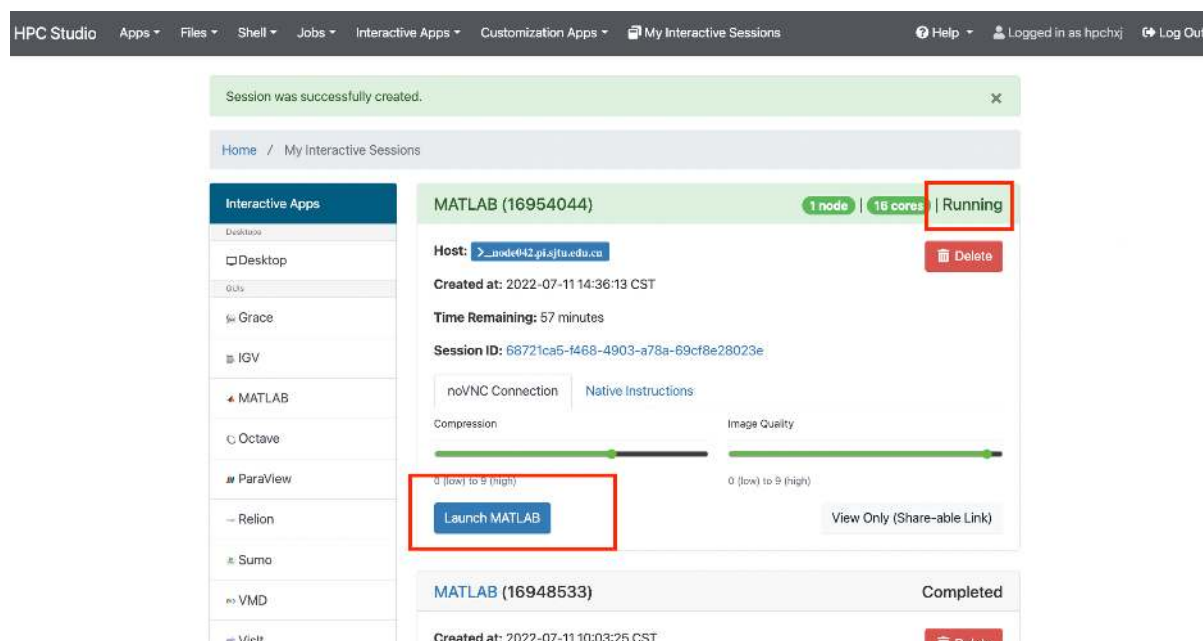
2. 申请资源，选择 **MATLAB** 版本

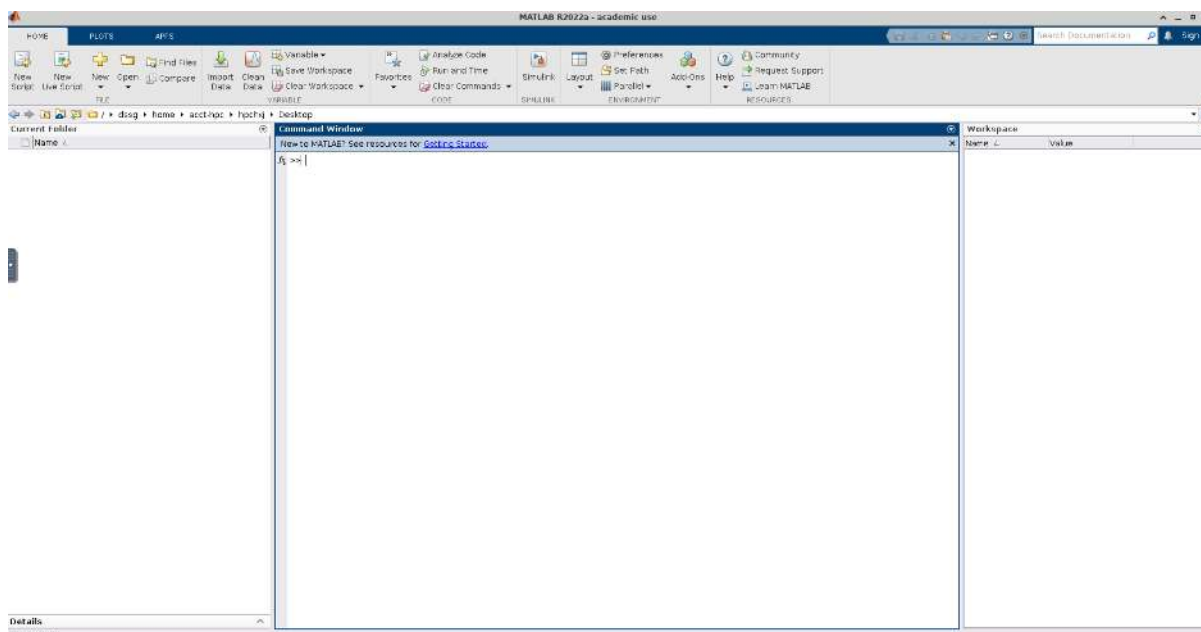
点击 **MATLAB** 图标后会跳转至资源选择界面，该页面上可选择申请的时长、可视化桌面的分辨率、平台资源以及 **MATLAB** 版本。



选择完毕后点击 **Launch** 按钮即跳转至会话管理界面，该界面会列出近期正在排队、运行或者已完成的 studio 会话。

等到该会话完成排队，进入 **Running** 状态，点击下方 **Launch MATLAB** 按钮，即可进入 **MATLAB** 应用。



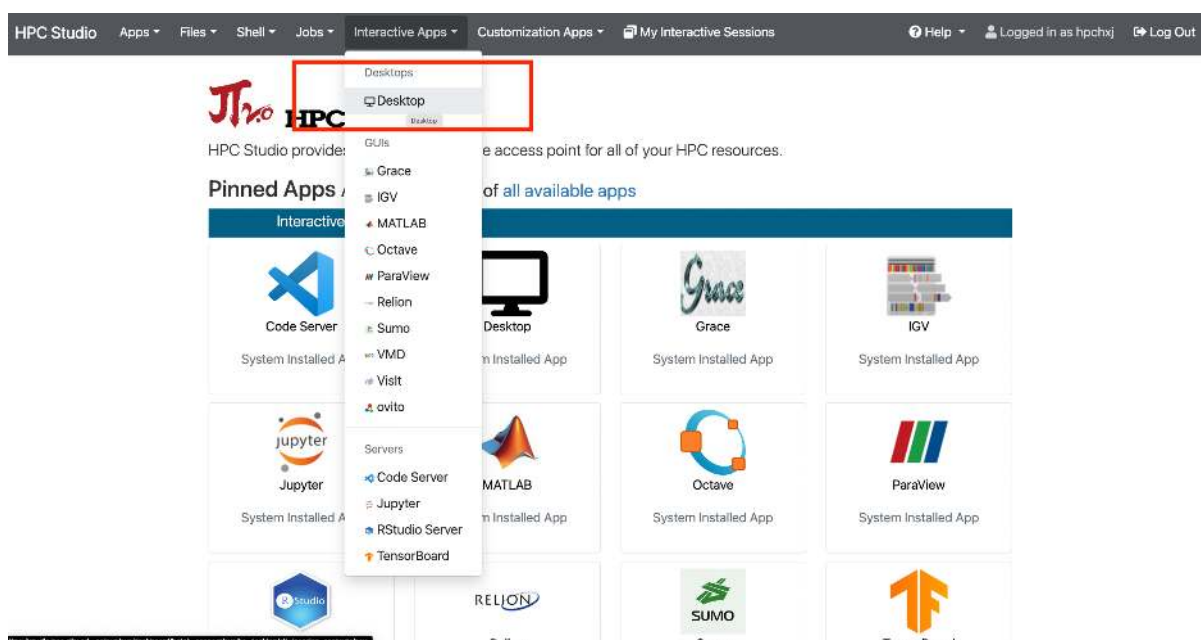


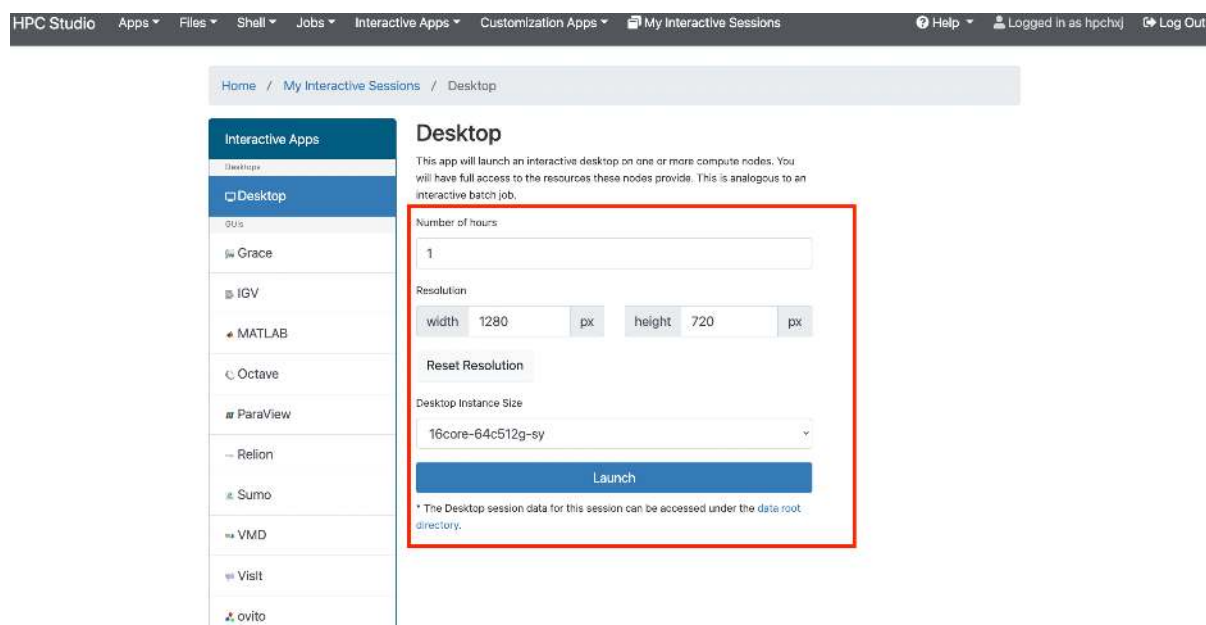
可视化平台桌面启动 MATLAB

除了从可视化平台的应用入口直接启动 MATLAB, 也可申请远程桌面, 从远程桌面的客户端运行 MATLAB 应用。

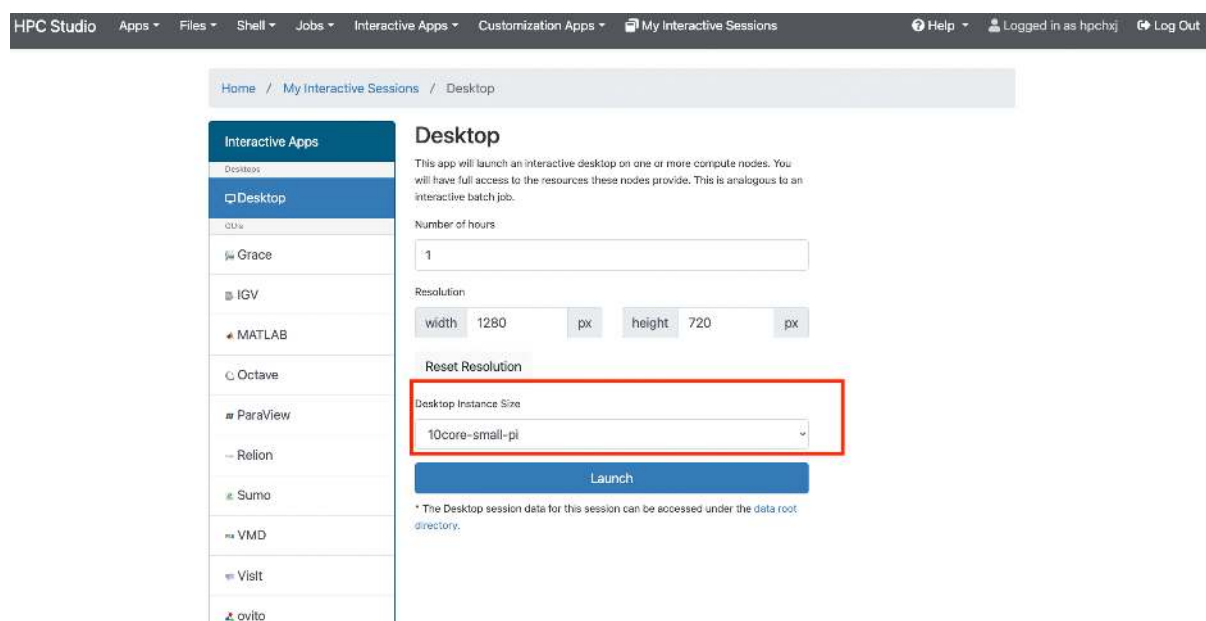
1. 启动远程桌面

使用 hpc 帐号登录 HPC studio (<https://studio.hpc.sjtu.edu.cn>) 后, 点击”Interactive Apps » Desktop”。选择需要的核数, session 时长 (默认 1 核、1 小时), 点击”Launch” 启动远程桌面。待选项卡显示作业在 RUNNING 的状态时, 点击”Launch Desktop” 即可进入远程桌面。



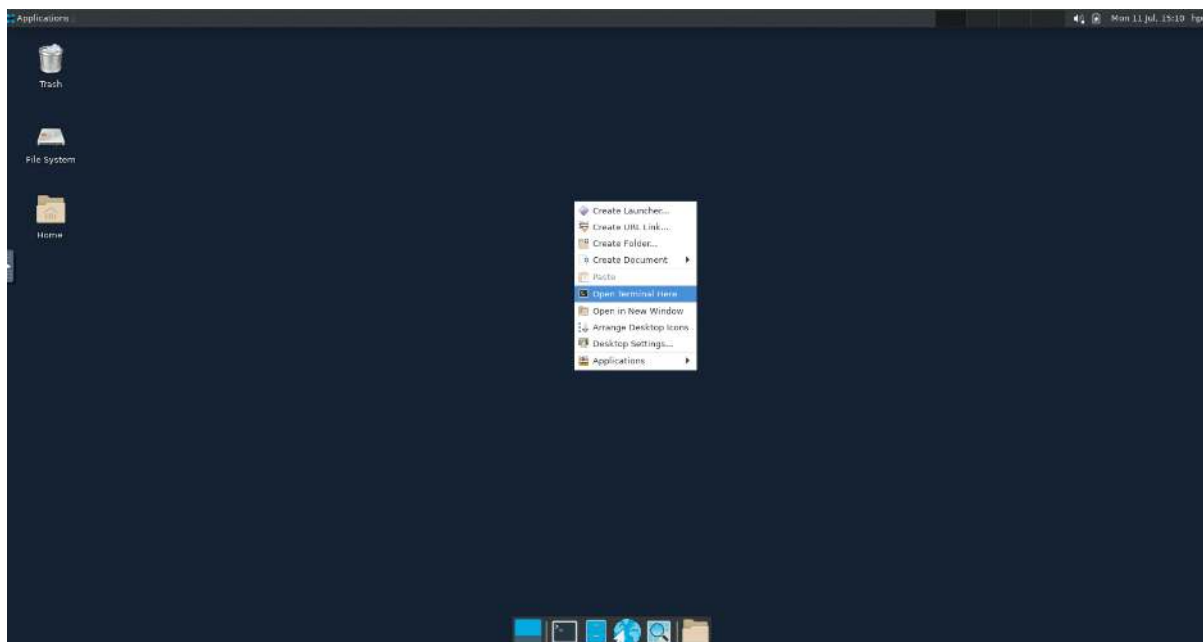


除了从思源一号启动远程桌面外， π 超算也支持启动远程桌面，在选定核数的同时可以同时选定平台：



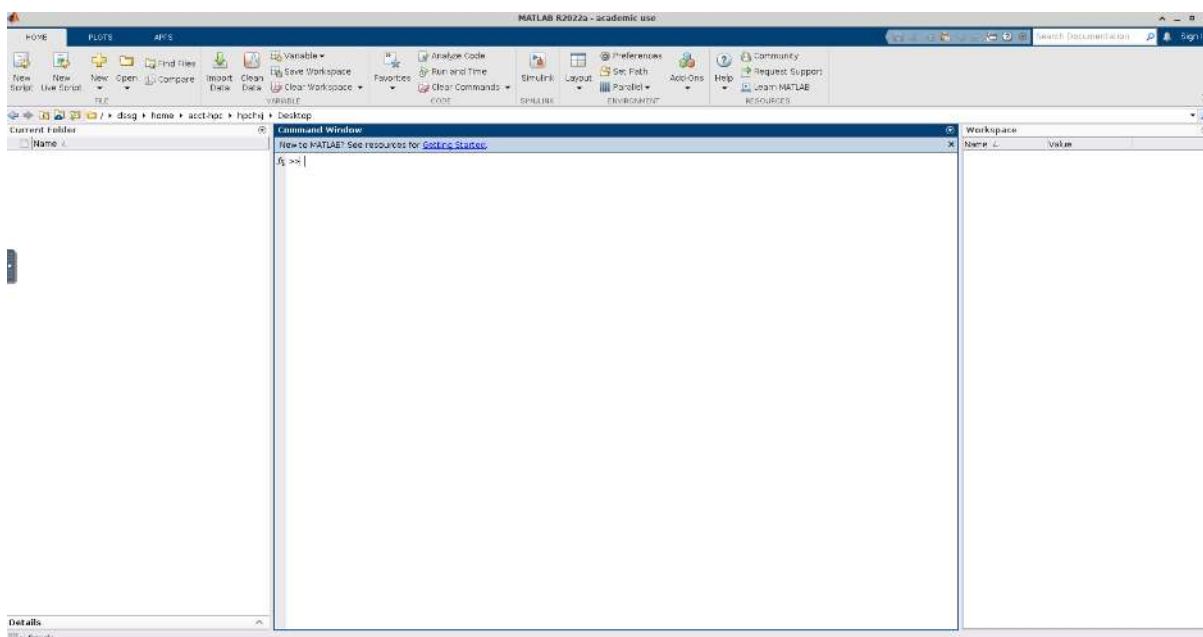
2. 启动 MATLAB

远程桌面中点击右键，选择 **Open Terminal Here** 打开终端。



在终端中使用命令 `module load matlab/r2022a && matlab`, π 超算和思源一号使用的命令一致。

启动后即可使用 MATLAB R2022a

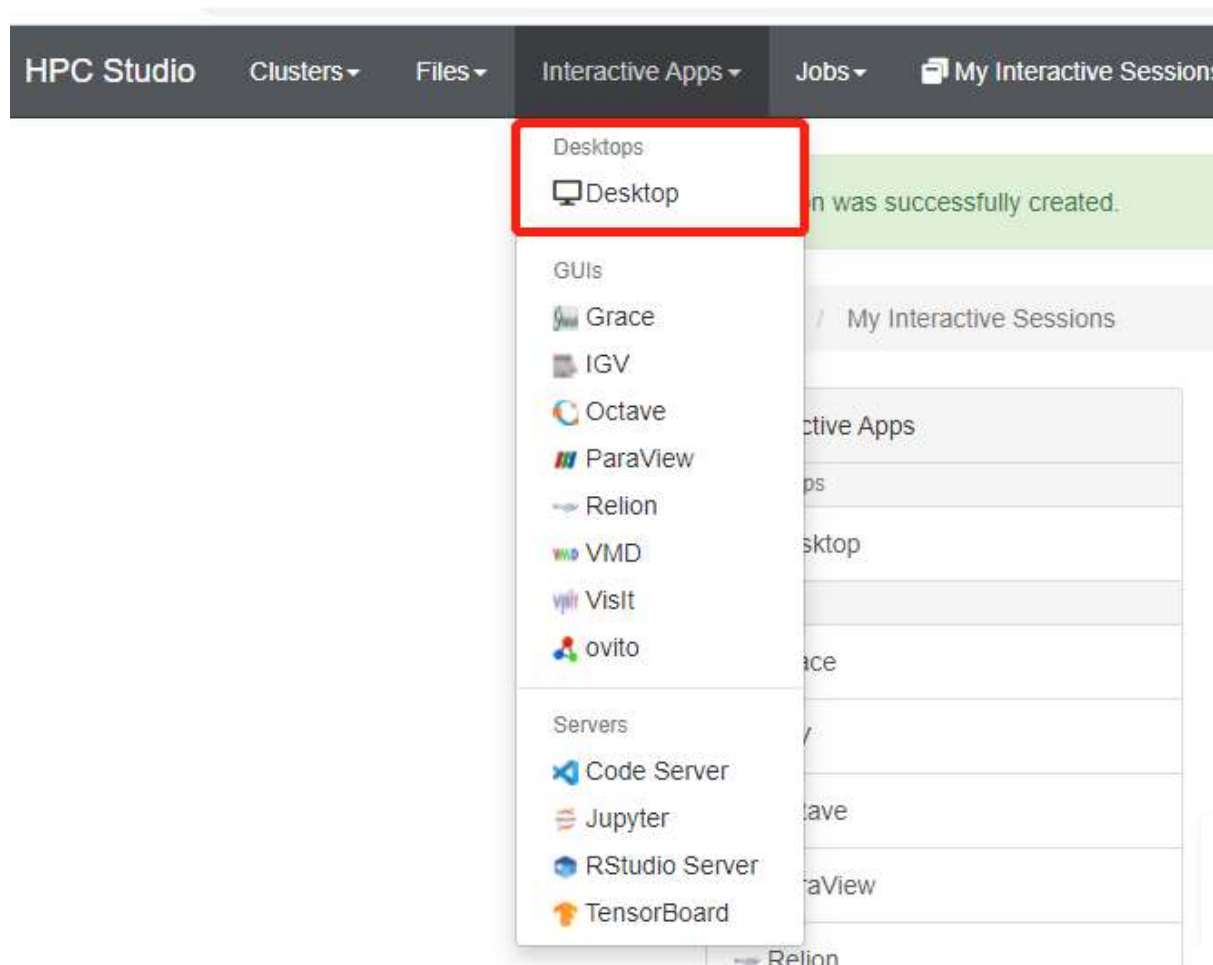


使用 **GPU** 版本的 **MATLAB**

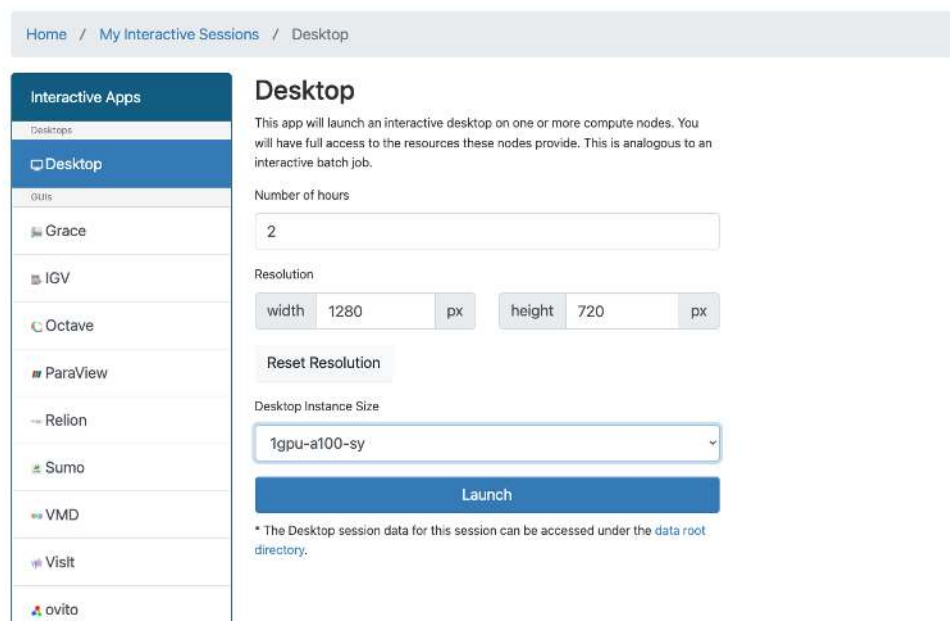
可视化平台使用 **MATLAB GPU** 版

1. 启动远程桌面

使用 hpc 帐号登录 HPC studio (<https://studio.hpc.sjtu.edu.cn>) 后, 点击”Interactive Apps » Desktop”。选择需要的核数, session 时长 (默认 1 核、1 小时), 点击”Launch”启动远程桌面。待选项卡显示作业在 RUNNING 的状态时, 点击”Launch Desktop”即可进入远程桌面。

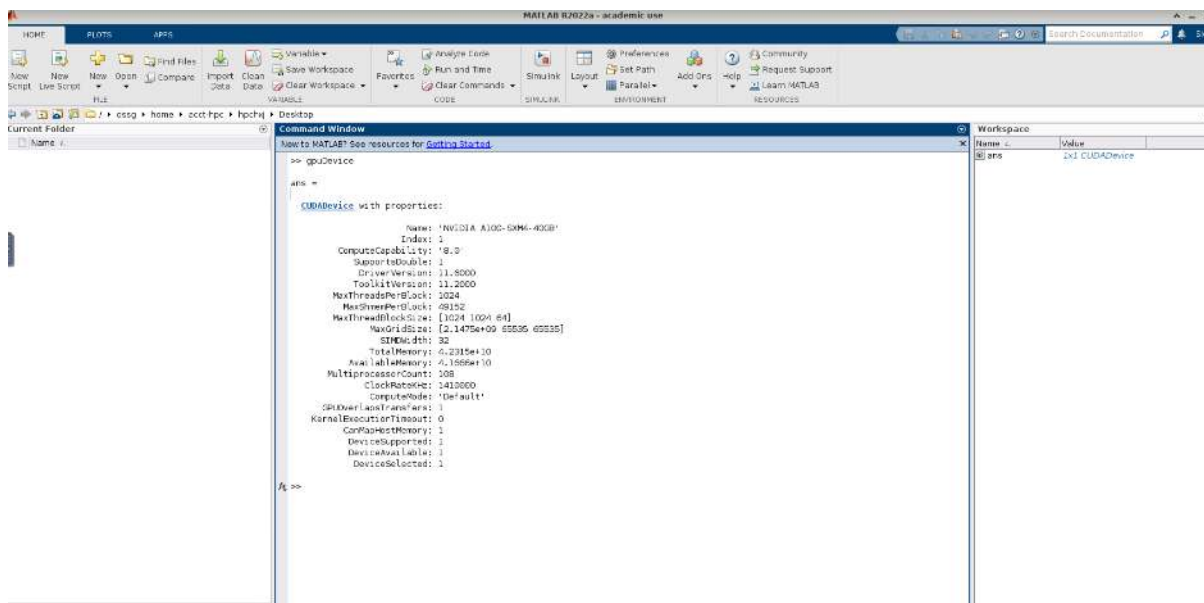


选定核数的时候选择思源一号的一张 GPU 卡：



2. 启动 GPU 版本 MATLAB

在窗口中启动终端（terminal），在终端输入 `module load matlab/r2022a && matlab`，即可启动 GPU 版本 matlab。



提交 **MATLAB GPU** 版脚本

思源一号

```
#!/bin/bash
#SBATCH -J matlab_test
#SBATCH -p a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH -N 1
#SBATCH --cpus-per-task 16
#SBATCH --gres gpu:1

module load matlab/r2022a

matlab -r $YOUR_SCRIPT_FILE
```

pi2.0

```
#!/bin/bash
#SBATCH -J matlab_test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH -N 1
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:1
```

(续下页)

(接上页)

```
module load matlab/r2022a
matlab -r $YOUR_SCRIPT_FILE
```

多节点并行版的 **MATLAB**

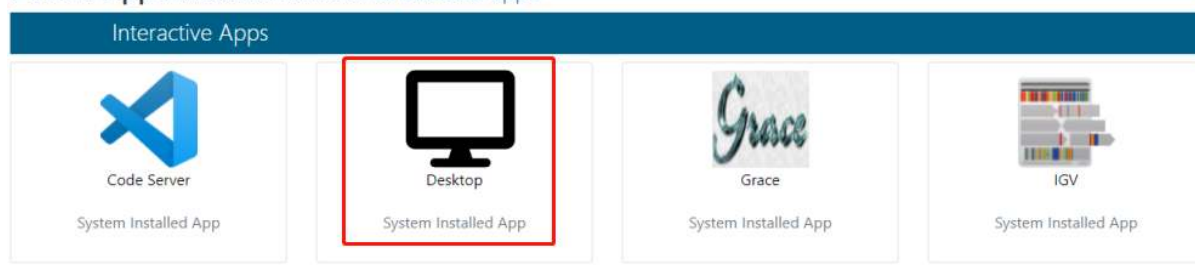
1. 首先，进入可视化终端界面

通过 HPC Studio <https://studio.hpc.sjtu.edu.cn> 打开远程桌面



HPC Studio provides an integrated, single access point for all of your HPC resources.

Pinned Apps A featured subset of all available apps



Desktop

This app will launch an interactive desktop on one or more compute nodes. You will have full access to the resources these nodes provide. This is analogous to an interactive batch job.

Number of hours

6

Resolution

width 1092

px

height 614

px

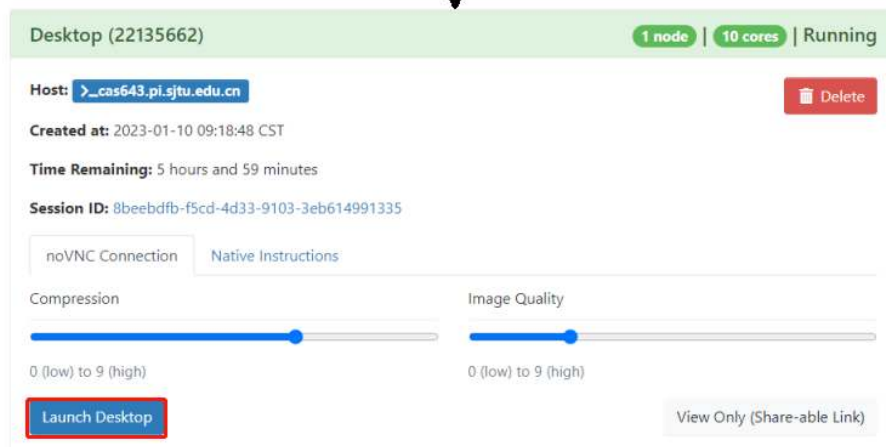
Reset Resolution

Desktop Instance Size

Pi Cluster 10cores

Launch

* The Desktop session data for this session can be accessed under the `data root` directory.



在桌面打开终端，创建 MATLAB 工作目录（可选），加载 MATLAB 环境：

```
cd ~
mkdir matlab && cd matlab
module load matlab/r2022a
matlab
```

2. 打开 MATLAB 后，导入 SlurmProfile。SlurmProfile 中记录了多节点并行运行的 parpool 设置，导入的配置可以在 Cluster Profile Manager 中确认。

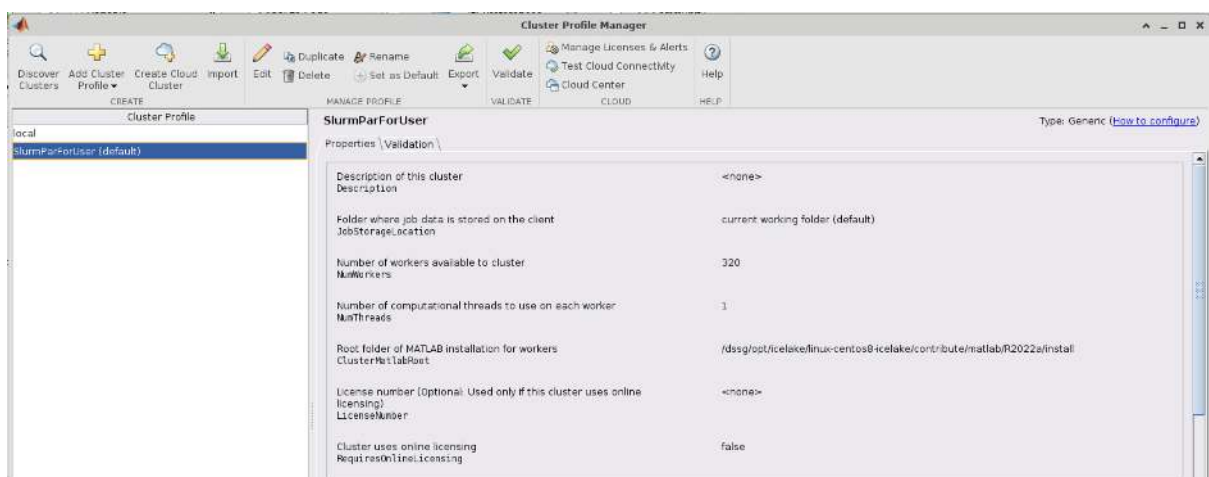
在 MATLAB 下方的命令行窗口输入命令:

```
# 若在 Pi 2.0 上使用 Matlab R2022a, 导入下面的并行配置
profile_master = parallel.importProfile('/lustre/opt/contribute/cascadelake/
↪matlab/R2022a/ParSlurmProfile/R2022a-SlurmParForUser.mlsettings');
# 若在 Pi 2.0 上使用 Matlab R2023a, 导入下面的并行配置
profile_master = parallel.importProfile('/lustre/opt/contribute/cascadelake/
↪matlab/R2023a/ParSlurmProfile/R2023a-SlurmParForUser.mlsettings');
# 若在 Pi 2.0 上使用 Matlab R2024a, 导入下面的并行配置
profile_master = parallel.importProfile('/lustre/opt/contribute/cascadelake/
↪matlab/R2024a/ParSlurmProfile/R2024a-SlurmParForUser.mlsettings');
# 若在 Pi 2.0 上使用 Matlab R2025a, 导入下面的并行配置
profile_master = parallel.importProfile('/lustre/opt/contribute/cascadelake/
↪matlab/R2025a/ParSlurmProfile/R2025a-SlurmParForUser.mlsettings');

# 若在思源一号上使用 Matlab R2022a, 导入下面的并行配置
profile_master = parallel.importProfile('/dssg/opt/icelake/linux-centos8-
↪icelake/contribute/matlab/R2022a/ParSlurmProfile/R2022a-SlurmParForUser.
↪mlsettings');
# 若在思源一号上使用 Matlab R2023a, 导入下面的并行配置
profile_master = parallel.importProfile('/dssg/opt/icelake/linux-centos8-
↪icelake/contribute/matlab/R2023a/ParSlurmProfile/R2023a-SlurmParForUser.
↪mlsettings');
# 若在思源一号上使用 Matlab R2024a, 导入下面的并行配置
profile_master = parallel.importProfile('/dssg/opt/icelake/linux-centos8-
↪icelake/contribute/matlab/R2024a/ParSlurmProfile/R2024a-SlurmParForUser.
↪mlsettings');
# 若在思源一号上使用 Matlab R2025a, 导入下面的并行配置
profile_master = parallel.importProfile('/dssg/opt/icelake/linux-centos8-
↪icelake/contribute/matlab/R2025a/ParSlurmProfile/R2025a-SlurmParForUser.
↪mlsettings');

# 根据选择的集群和 Matlab 版本导入并行配置后, 将其设置为默认
parallel.defaultClusterProfile(profile_master);
```

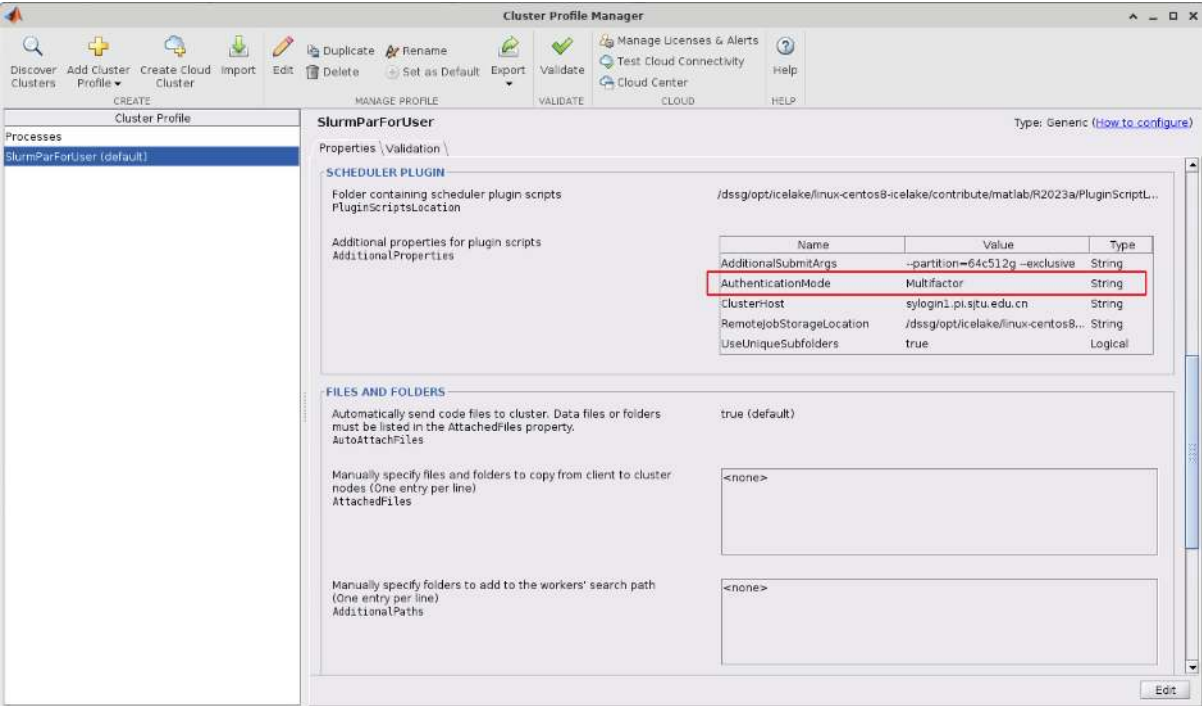
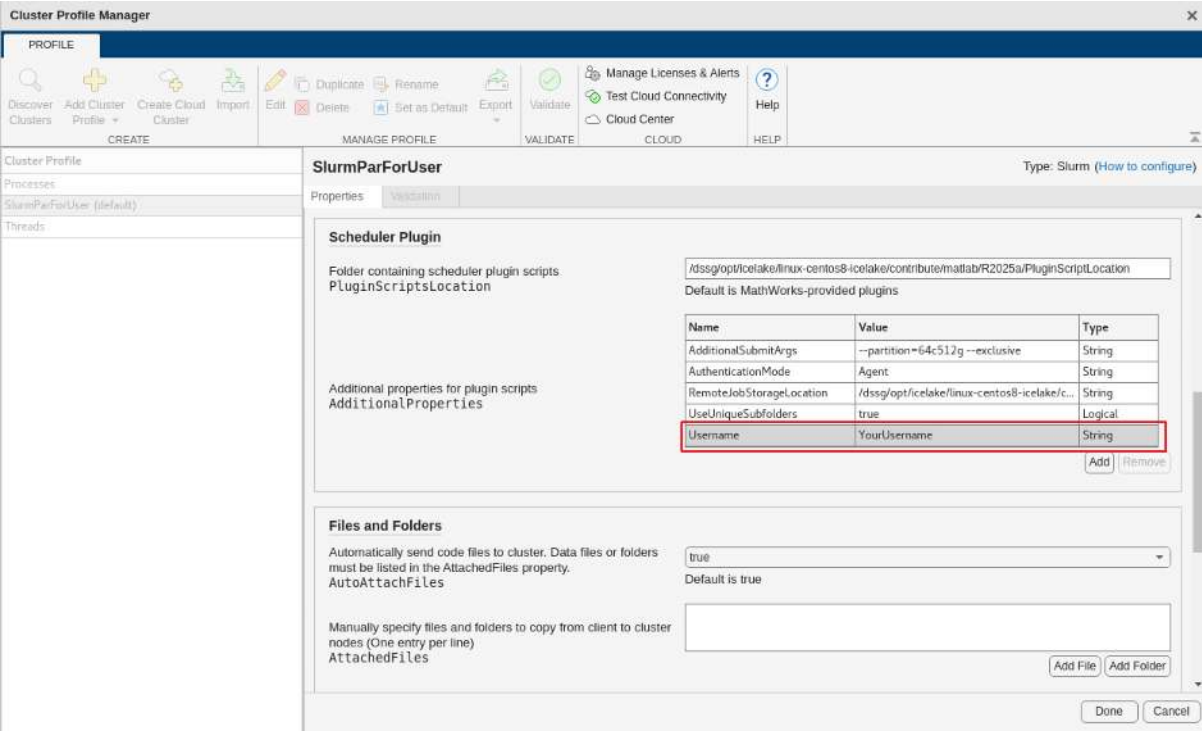
在 MATLAB 界面选择 Home -> Parallel -> Create and Manage Clusters, 在 Cluster Profile Manager 中查看导入的配置:



备注

对于 2024a 及以后的版本，导入对应集群版本的 **SlurmProfile** 之后，需要在 **Cluster Profile Manager** 的插件设置中，将 **Username** 的值修改为您的超算账号名。

对于 2023a 及以前的版本，由于超算登录方式更新 (<https://docs.hpc.sjtu.edu.cn/login/sshlogin.html>)，需要重新导入对应集群版本的 **SlurmProfile**。或者在 **Cluster Profile Manager** 的 **Slurm** 插件设置中新增一条，名称为 **AuthenticationMode**，值为 **Multifactor**。

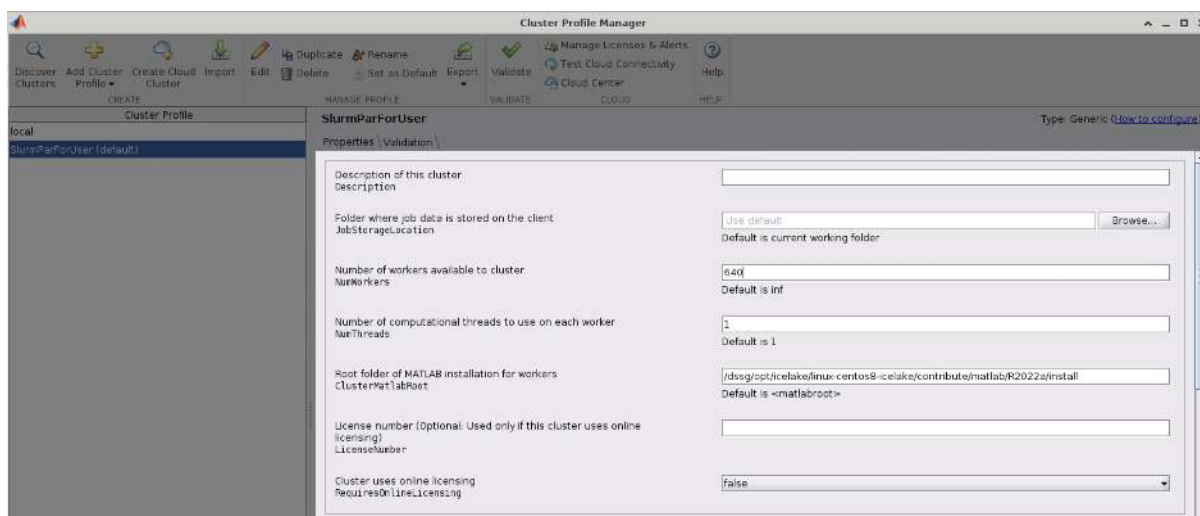


3. (可选) 调整并行池的大小

默认 **SlurmProfile** 的最大 **worker** 数目为 200 (Pi 2.0) 或 320 (思源), 如果您需要调整这一数值, 具体可以按照以下的 **GUI** 方式或者命令行方式操作, 建议将最大 **worker** 数设为计算节点总核心数的倍数, Pi 2.0 节点为 40 核, 思源一号节点为 64 核。下面以思源一号为例, 将并行池大小调整为 640 **worker**。

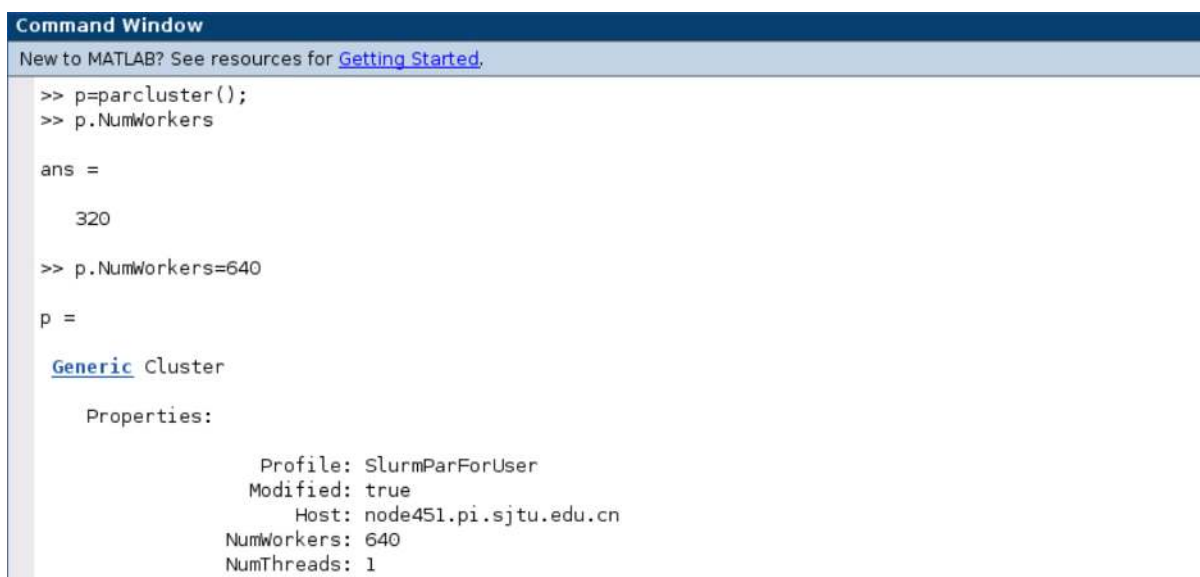
GUI 方式调整并行池大小:

- 在 **Cluster Profile Manager** 中选择要修改的 **SlurmProfile** -> **Edit**, 修改 **NumWorkers** 为需要的值, 然后保存。
- 如果需要导出修改之后的配置, 选择 **Cluster Profile Manager** -> **Export**, 将配置保存为文件, 下次使用时导入即可。



命令行方式调整并行池大小, 需要在 **MATLAB** 命令行窗口输入:

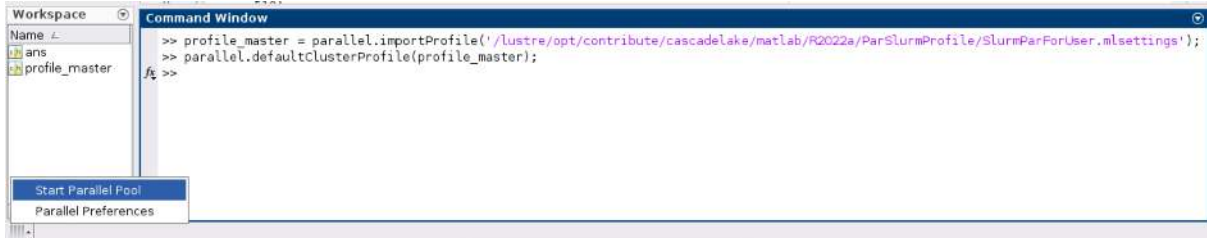
```
p=parcluster();
p.NumWorkers
p.NumWorkers=640
p.saveProfile
```



4. 启动并行池

GUI 方式启动并行池：

点击 **MATLAB** 左下角的并行池图标，选择启动并行池，同时可以查看并行池运行状态。第一次申请资源池时，会要求输入在集群上的账号和密码，然后在整个 **matlab** session 中均有效。



命令行方式启动并行池：

在 **MATLAB** 命令行窗口输入：`parpool('SlurmParForUser',320)`，这里的 `SlurmParForUser` 可能需要替换成您自定义的名字。

注意：启动并行池时，系统将按照设置的工作核心数申请资源。例如 **NumWorkers** 为 **1000** 时，系统将在启动并行池时申请到 **1000** 核的资源，请注意您的作业费用消耗。

5. 接下来，运行作业

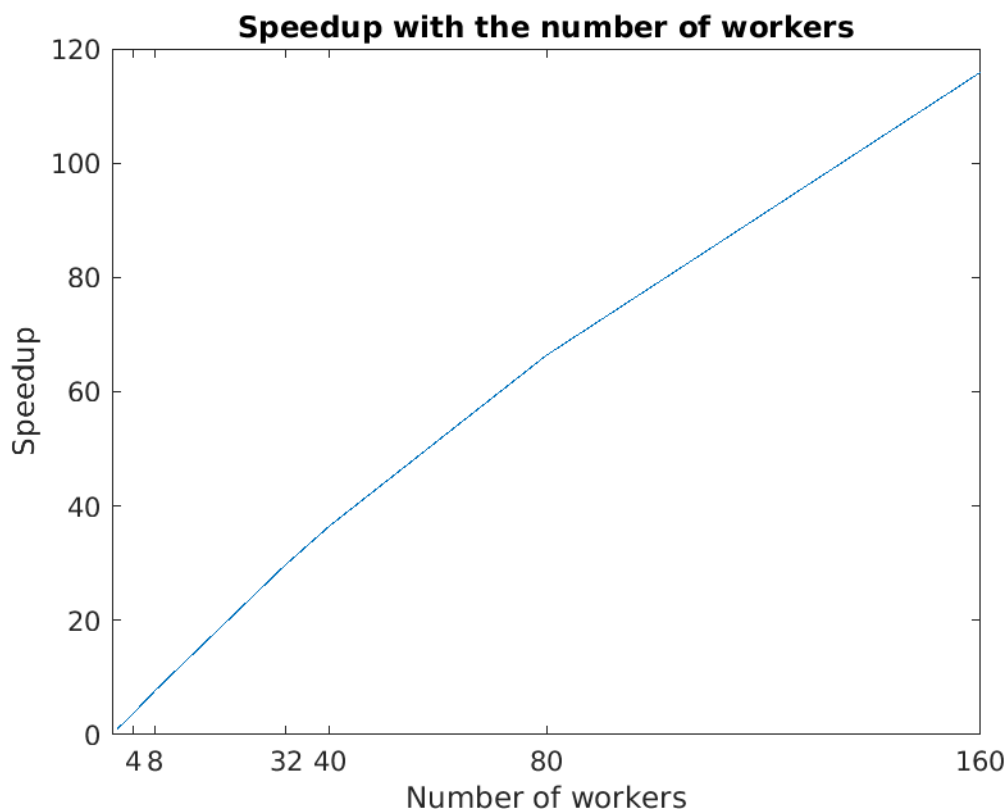
示例作业脚本路径如下所示，具体功能为素因素分解，使用的核数为 1、4、8、32、40、80 和 160 核，生成的图片为不同核数的计算时间与使用 1 核时的加速比。

```
/lustre/share/samples/matlab/composite_speedup.m
```

在 **MATLAB** 命令行窗口输入：

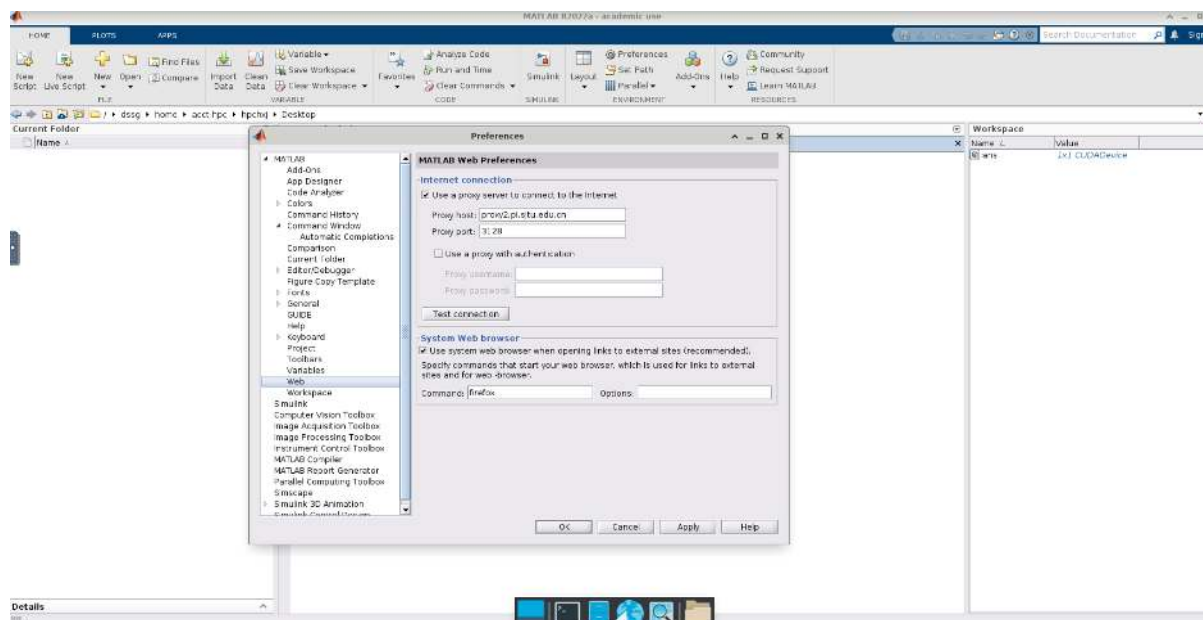
```
composite_speedup
```

6. 运行结果为



MATLAB 代理设置

使用过程中如果遇到 Unable to open the requested feature. 等网络问题或者不能正常使用 Live Editor 功能，可以通过设置代理解决。



π 超算代理设置

proxy.hpc.sjtu.edu.cn:3004

思源一号代理设置

proxy2.pi.sjtu.edu.cn:3128

4.5.9 Windows 虚拟机

简介

为满足校内科研用户对于 Windows 应用的需要，超算平台提供了 Windows 虚拟机的服务，方便用户在超算上使用此类软件。

如何使用

1. 向 **HPC 邮箱** 发送邮件，申请开通在超算上使用 Windows 服务；
2. 开通相关权限后，在 **Studio** 可视化平台选择 **Windows Server**；



3. 选择需要申请的资源、时间，等待资源初始化，首次启动需要 5 分钟左右，请耐心等待；

Home / My Interactive Sessions / Windows Server

Interactive Apps

- Desktops
- Desktop
- GUIs
- GIMP
- Grace
- IGV
- Jalview
- MATLAB
- Octave
- Panoply
- ParaView

Windows Server version: v1.0.0

This app will launch Windows Server on the cluster.

Number of hours

App Instance Size

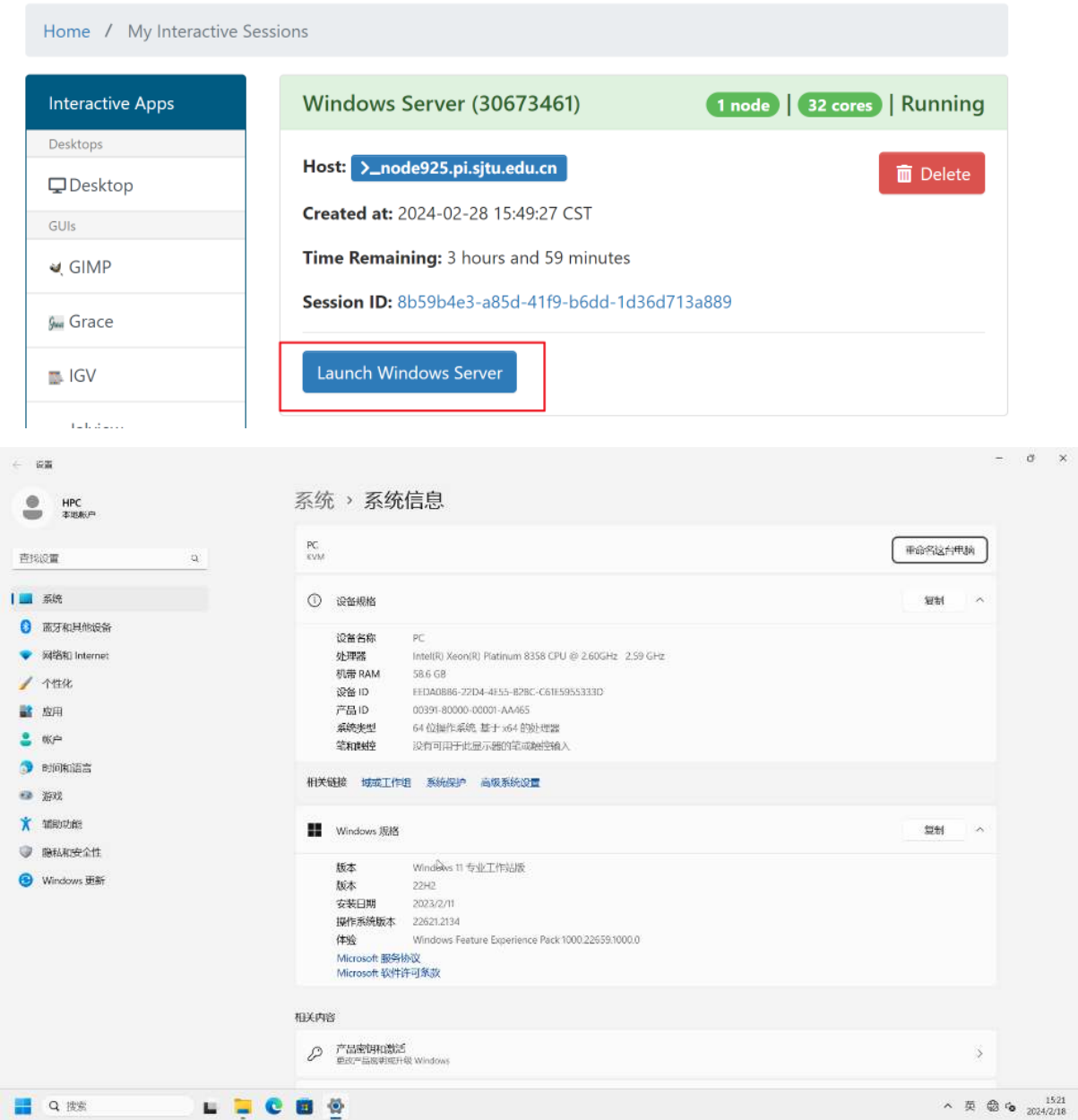
Number of cores

App Version

Launch

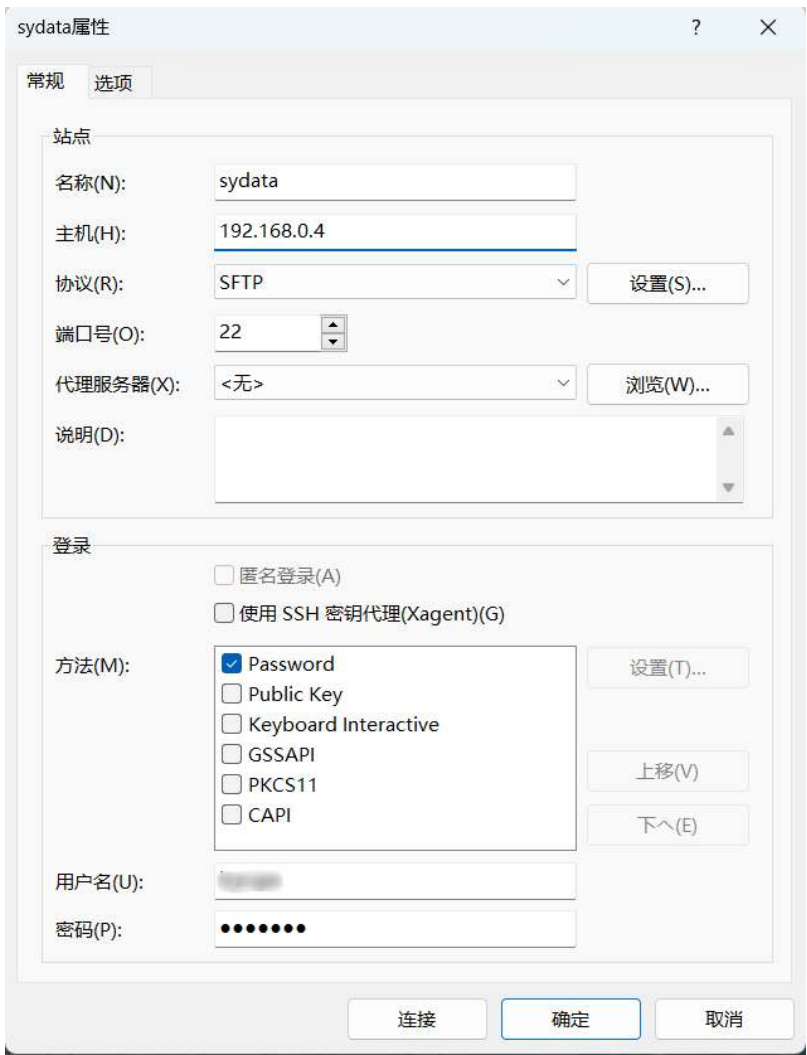
* The Windows Server session data for this session can be accessed under the [data root directory](#).

4. 虚拟机初始化完成后，开始在超算平台运行 Windows 应用。



常见问题

1. 如何在 Windows 虚拟机中使用超算上的数据？



默认提供了 XFTP 文件传输工具，新建连接时需要注意主机地址为 192.168.0.4 的内网地址，用户名和密码与超算的一致。

2. Windows 虚拟机的硬盘用满了怎么办？

请发送邮件至 [HPC 邮箱](#) ，我们将为您扩容。

致谢

Windows 虚拟机的原始方案和相关脚本由中南大学的邹有老师提供，感谢邹老师提供的技术支持。

4.5.10 常见问题

1. 在 HPC studio 中打开可视化软件时，显示 “Proxy Error”

A：(1). 请关闭校园网等 VPN；(2). 尝试清理浏览器缓存，或者切换浏览器进行尝试；(3). 用手机热点去测试是否正常。

2. Jupyter、Rstudio 连接提示需要输入密码

A: 该问题是因为设置了 Conda 环境的自动激活, 需要设置取消 Conda 环境的自动激活。请在命令行中执行 `conda config --set auto_activate_base false`。

此外, Studio 还支持 TensorBoard, Code Server, IGV, Octave, Relion, VMD, Visit, OVITO 等应用, 并将持续增加。

4.6 登录交我算集群

获取交我算账号后, 可通过浏览器登录 可视化平台 **HPC Studio**, 也可通过传统的 SSH 登录。下面将介绍 SSH 登录方法。

4.6.1 通过 SSH 登录集群

超算平台于 2024 年 9 月 23 日开始对接交大 jAccount 账号, 对于校内用户, 超算账号关联 jAccount 后您可以用 超算账号和 **jAccount** 密码进行 SSH 登录。如果您希望进行免密登录, 需要向平台申请免密证书。

注意事项:

- 2024 年 9 月之后的校内新用户, 超算账号将与您的 jAccount 同名; 而此时间点之前的老用户, 将保留原有的独立用户名。因此校内的老用户绑定 jAccount 之后需要使用 原用户名 +jAccount 密码进行 SSH 登录。
- 任务完成后请主动结束后台进程, 关闭 SSH 会话。长时间不活动的用户滞留进程会由系统强制清理。
- 恶意的 SSH 客户端软件会窃取您的密码, 请在官网下载正版授权 SSH 客户端软件。
- 若无法登录, 请检查输入密码或确认 IP 地址是否正确。您可以参考故障排除和反馈, 将诊断信息发送给 **HPC 邮箱**。

登录入口

交我算平台目前有 3 个用于作业交互的登录入口和 2 个用于数据传输的数据节点, 其中部分入口有负载均衡功能, 会根据后端的负载情况将您的会话分流到合适的服务器上, 请根据自己的业务场景选择合适的入口。

入口域名	用途分类	负载均衡	说明
pilo-gin.hpc.sjtu.edu	作业交互	是	π 2.0 和 AI 集群的登录入口，用于向 cpu, huge, dgx2, debug 队列投递作业
sylo-gin.hpc.sjtu.edu	作业交互	是	思源集群的登录入口，用于向 64c512g, a100, a800 和对应的 debug 队列投递作业
armlo-gin.hpc.sjtu.edu	作业交互	否	arm 集群的登录入口，用于向 arm128c256g, debugarm 队列投递作业
data.hpc.sjtu.ed	数据传输	否	主要用于在本地和/lustre/home/XXX 存储池之间传输数据，也可在 lustre 和 dssg 之间转移数据
sy-data.hpc.sjtu.ed	数据传输	否	用于在本地和/dssg/home/XXX 存储池之间传输数据

本章节后续内容均以登录 **pilogin.hpc.sjtu.edu.cn** 入口为例，如果登录其他入口，请自行替换指令中的域名。

进行密码认证的 **SSH** 登录

不同类型的用户在绑定 jAccount/邮箱后，进行 **SSH** 登录需要使用的账号和密码整理如下表：

用户类型	SSH 账号名	SSH 密码
交大用户	实际账号名 2024/09 以后的新账号统一为 jAccount 名，在这以前的老账号继续保持原账号名	jAccount 密码
合作用户	自定义用户名	自定义密码

当您的账号成功登录后，系统将根据您选择的通知渠道，发送对应的登录行为通知，这可能是一条交我办 **APP** 消息或者一封邮件。

如果您需要在短时间内频繁登录超算，或者通过脚本进行 scp、rsync 等批量操作，请参阅 [账号安全信息管理](#) 的相关章节，申请免密证书。

以 **Mobaxterm** 为代表的命令行客户端进行 **SSH** 登录

Mobaxterm 可以在 windows 系统上提供模拟的 linux 环境，用此客户端进行 **SSH** 登录的操作与原生 linux 环境相同。同理本章节也适用于从原生 linux 环境发起 **SSH** 登录。

- 假设您的超算账号为 **user01**，需要登录的入口为 **pilogin.hpc.sjtu.edu.cn**。
1. 在客户端的命令行窗口发起 ssh 连接：`ssh user01@pilogin.hpc.sjtu.edu.cn`。
 2. 如果您没有保存过密码，根据命令行提示完成传统账号密码认证。登录成功后您可以选择保存密码，以便在后续登录时跳过此步骤。

3. 如果登录后您看到如下提示，请参阅 [账号安全信息管理](#) 的相关章节，及时关联 jAccount/邮箱。

```
[2024-09-05 16:34.48] /drives/d/project/docs.hpc.sjtu.edu.cn
[ ] > ssh -o PubkeyAuthentication=no [ ]@[ ]
If you are SJTU user, authenticate with jAccount password; otherwise use HPC password:
Please link your jaccount through https://mfa.hpc.sjtu.edu.cn as soon as possible!
Please link your jaccount through https://mfa.hpc.sjtu.edu.cn as soon as possible!
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Tue Aug 27 15:21:46 2024 from [ ]
[2024-09-05T16:35:45+08:00][ ] ~
>
```

以 **XShell** 为代表的图形界面客户端进行 **SSH** 登录

不同客户端对 **SSH** 登录的图形化封装通常存在较大差异，本章节以 **XShell** 为例。如果您倾向使用其他图形化客户端，可能需要查询官方文档以找到等效的操作方式。

1. 点击 **文件-新建** 打开新建会话窗口，填入希望登录的目标入口。
2. 新建 **ssh** 连接时请在身份验证方法中勾选 **Keyboard Interactive** 方式。您也可以在这里预先填好密码。设置完成后保存连接。
3. 打开上一步建立好的连接，如果在前面已经输入了密码，客户端会自动完成传统密码认证，您将跳过此步骤；否则客户端会弹窗要求您输入密码。
4. 如果登录后您看到如下提示，请参阅 [账号安全信息管理](#) 的相关章节，及时关联 jAccount/邮箱。

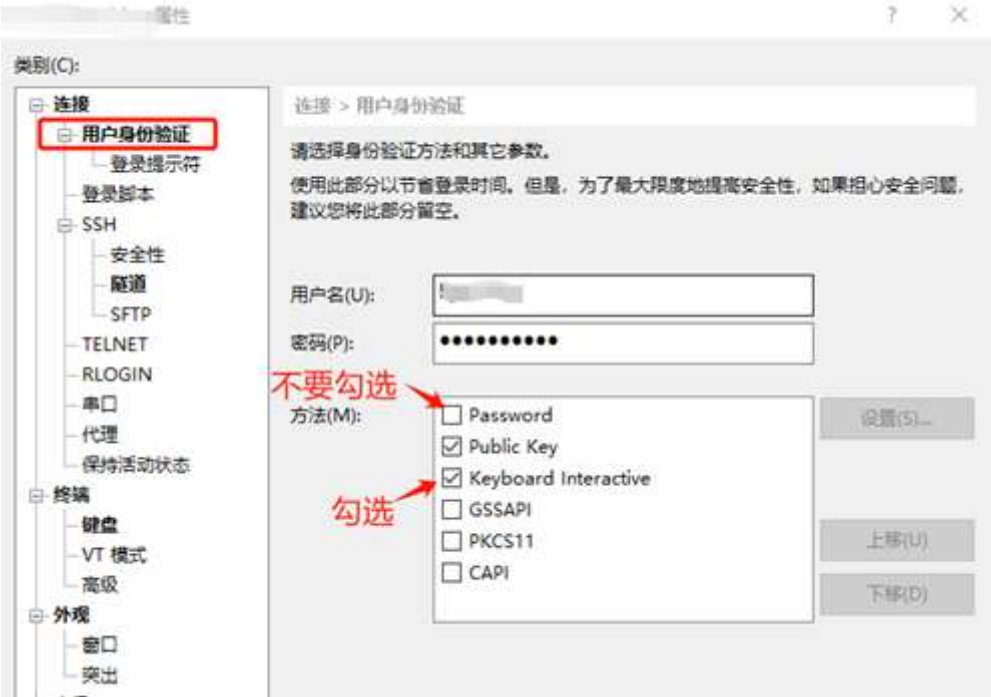
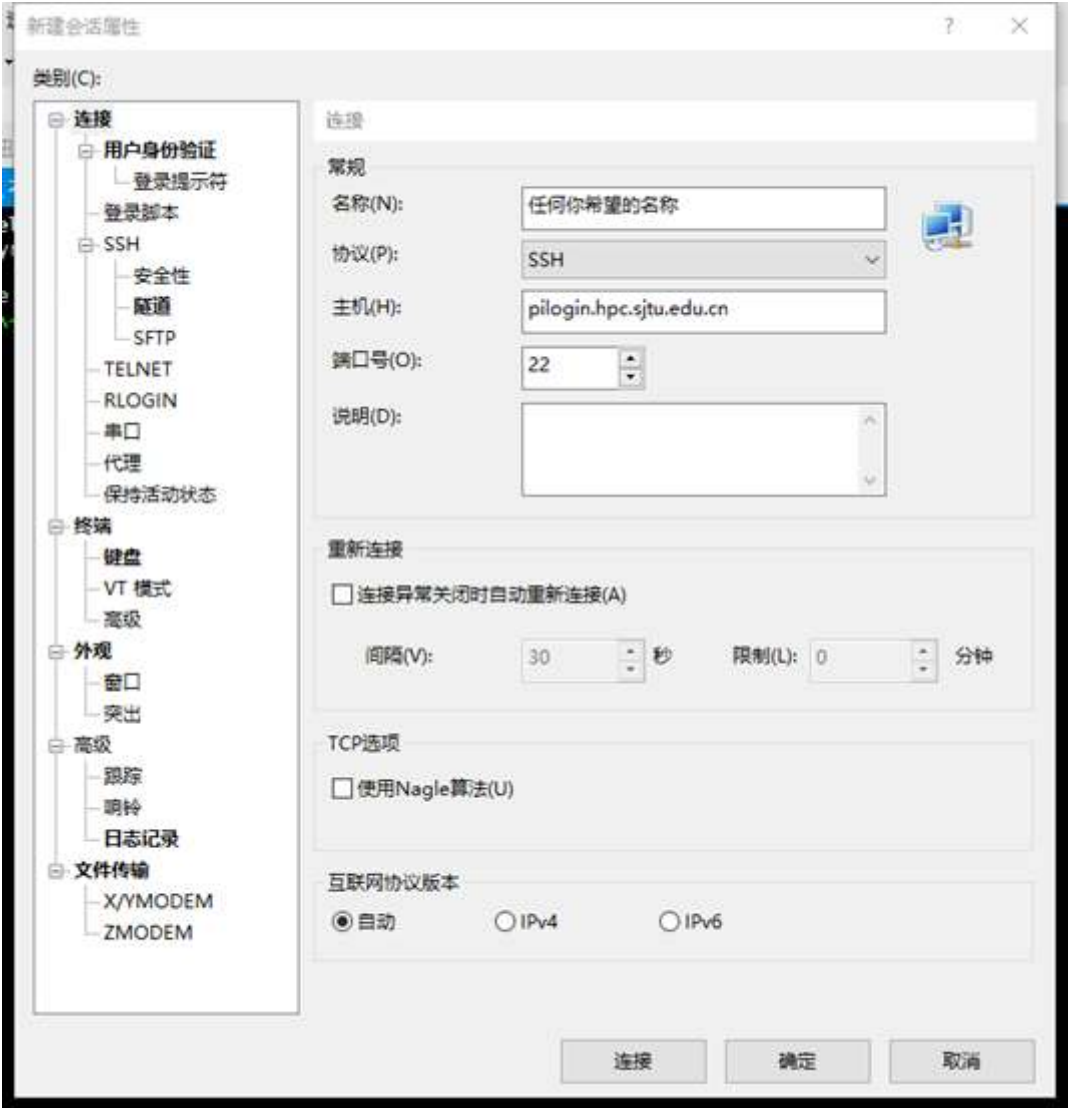
免密 **SSH** 登录

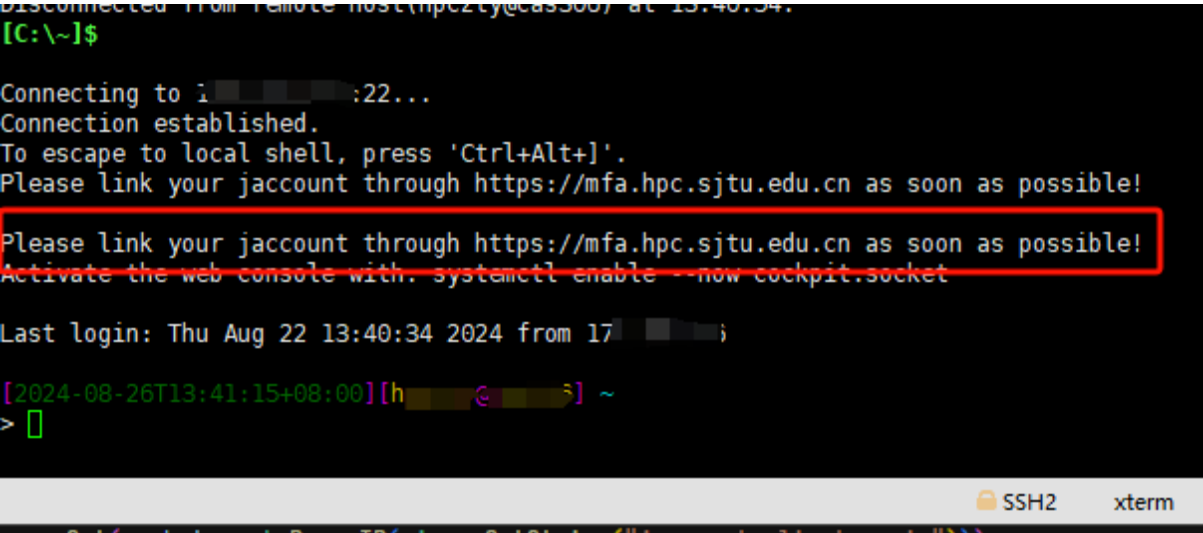
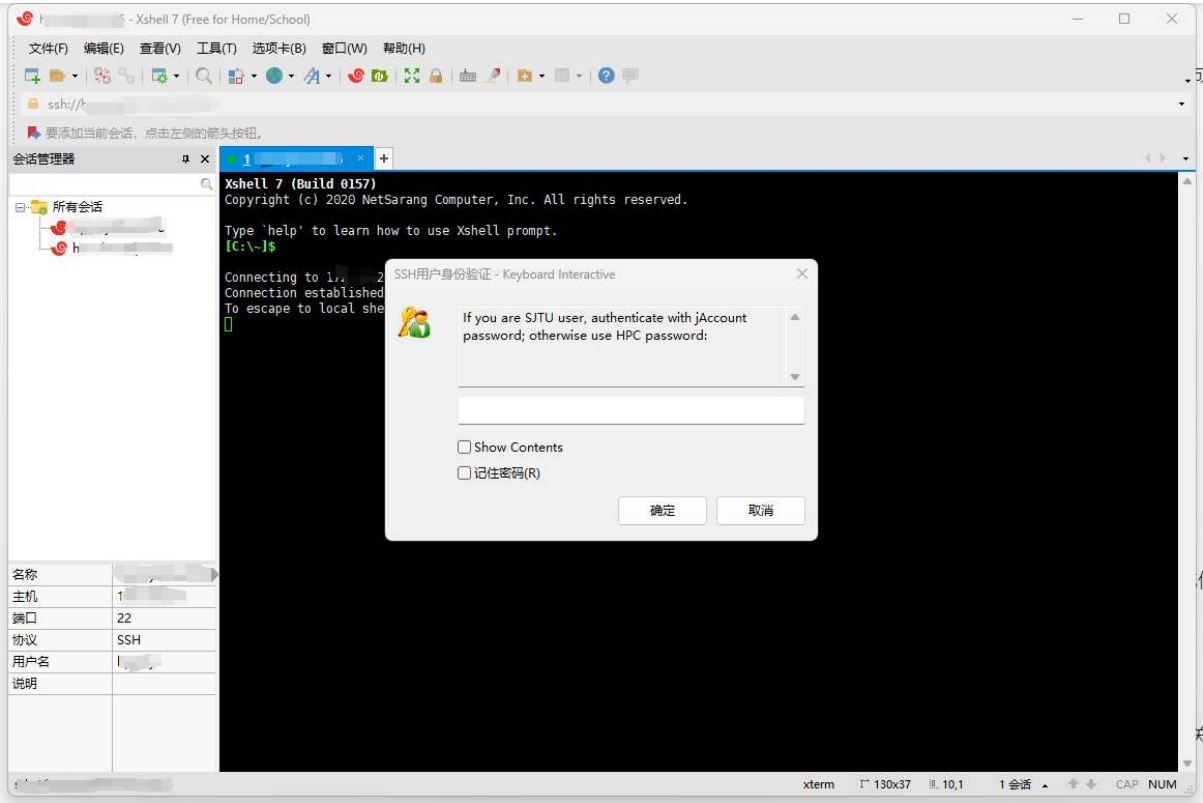
经过安全升级后，超算平台不再支持传统的公钥免密，将公钥写入个人家目录的 `~/.ssh/authorized_keys` 不会再赋予您免密登录的权限。

如果您希望继续进行 **SSH** 免密登录，需要从 [超算账号管理平台](#) 申请免密证书。以 **RSA** 密钥为例，公钥免密和证书免密的区别如下表所示。

传统公钥免密	证书免密
生成公钥 <code>id_rsa.pub</code> 和私钥 <code>id_rsa</code> 文件	生成公钥 <code>id_rsa.pub</code> 和私钥 <code>id_rsa</code> 文件 用户提交公钥 <code>id_rsa.pub</code> 申请证书 <code>id_rsa-cert.pub</code>
将公钥内容写入服务器的 <code>~/.ssh/authorized_keys</code> 中	
本地使用私钥 <code>id_rsa</code> 进行免密登录	本地使用私钥 <code>id_rsa</code> 和证书 <code>id_rsa-cert.pub</code> 进行免密登录

申请免密证书的方法请参阅 [账号安全信息管理](#) 的相关章节。





以 **Mobaxterm** 为代表的命令行客户端进行免密登录

此章节同样适用于原生 linux 环境进行免密登录。

将下载得到的证书文件存放到 `~/.ssh/` 目录，同时您原有的或者下载得到的私钥文件也应该存在于此目录，并与证书文件名保持上表所示的匹配关系。如果您不确定 **Mobaxterm** 将个人家目录映射到了哪里，请打开 `settings - configurations` 检查下图所示的路径配置。

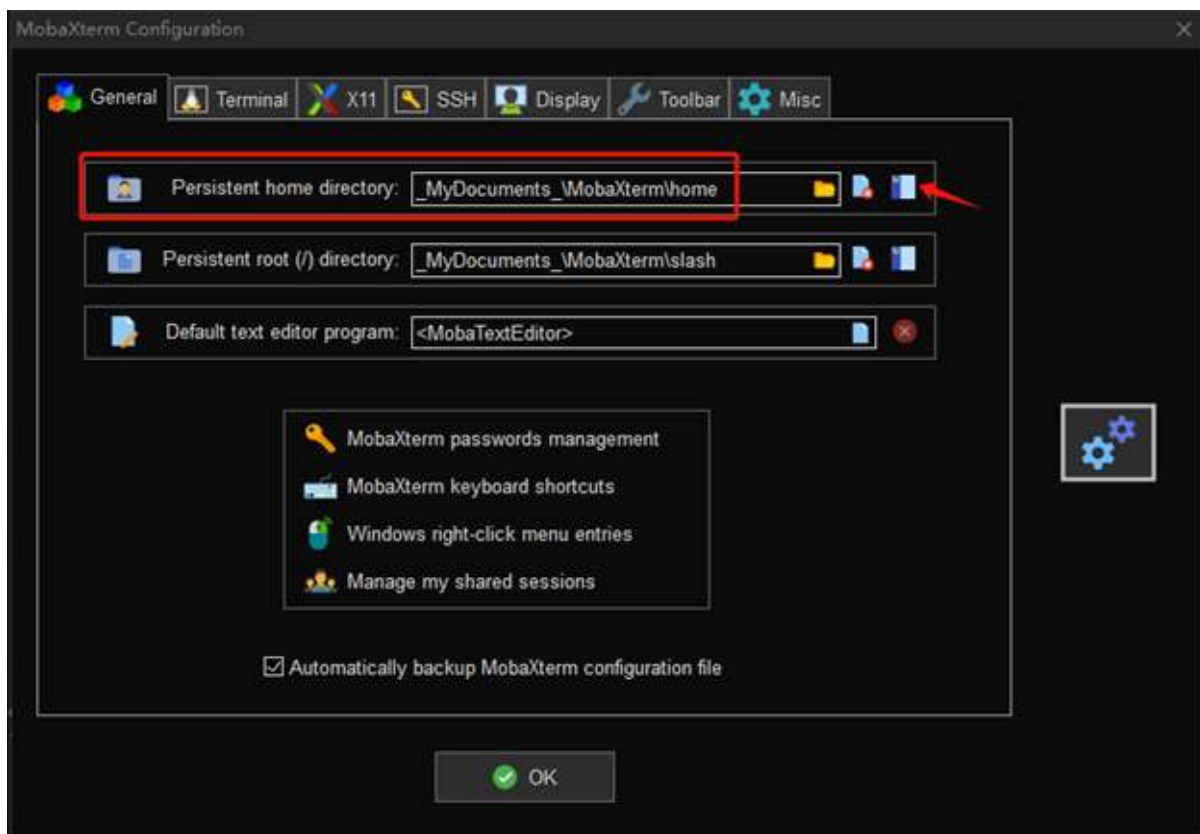


图 1: Mobaxterm 查看个人目录映射位置

之后在免密有效期内发起 `ssh` 连接即可实现免密登录。

以 **XShell** 为代表的图形界面客户端进行免密登录

不同客户端对 **SSH** 登录的图形化封装通常存在较大差异，本章节以 **XShell** 为例。如果您倾向使用其他图形化客户端，可能需要查询官方文档以找到等效的操作方式。

图形界面使用和管理秘钥、证书操作较为繁琐，不建议使用此类客户端。

1. 点击 **工具-用户密钥管理者** 打开管理窗口，点击导入，分别导入您下载到的私钥和证书。请确保两个文件名符合上表演示的对应关系。
2. 右键您的 **SSH** 会话，进入属性设置，在 **用户身份验证** 选项卡中，勾选 **Public Key** 方法，并确保此方法排序在 **Keyboard interactive** 方法之前。点击设置，选择上一步您导入的秘钥。
3. 在免密有效期内，您使用此会话进行 **SSH** 登录将不再需要手动认证。有效期过后，您需要重新申请新的免密证书。

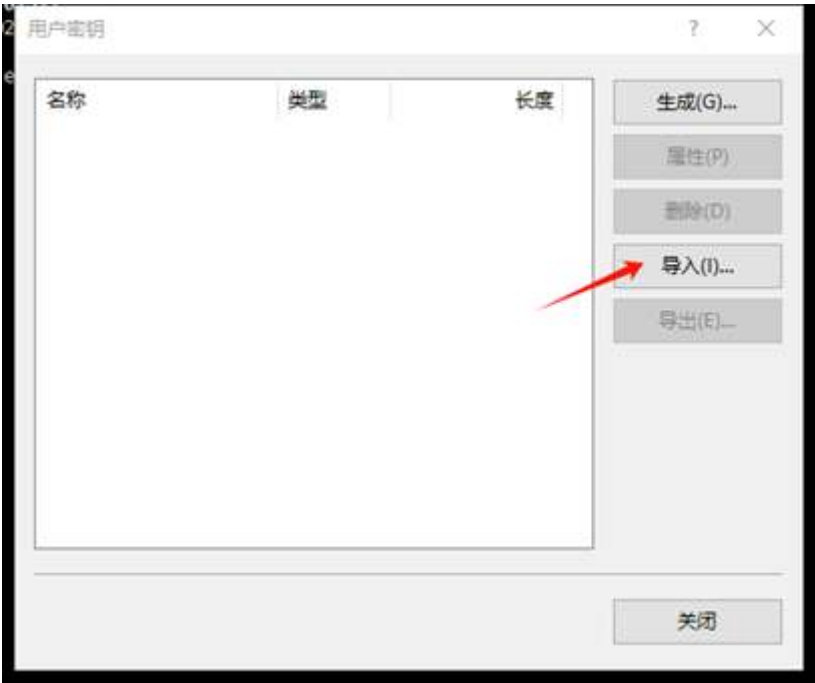
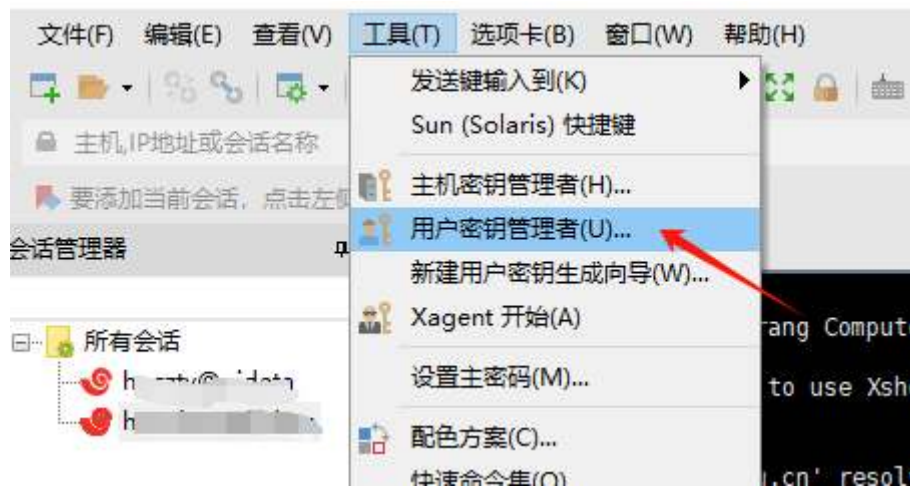


图 2: XShell 导入密钥和证书

如何生成自己的公私钥对

```
(在集群上) $ rm -f ~/.ssh/authorized_keys # 清除服务器上原有的
authorized_keys
(在自己电脑上) $ rm ~/.ssh/id* # 清除本地 .ssh
文件夹中的密钥对
(在自己电脑上) $ ssh-keygen -t rsa #
在本地重新生成密钥对。第二个问题，设置密码短语
(passphrase)，并记住密码短语
(在自己电脑上) $ ssh-keygen -R sylogin.hpc.sjtu.edu.cn # 清理本地 known_
hosts 里关于集群的条目
(在自己电脑上) $ ssh-copy-id YOUR_USERNAME@TARGET_IP #
将本地新的公钥发给服务器，存在服务器的 authorized_keys 文件里
```

SSH 重置 known_hosts

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Please contact your system administrator.
Add correct host key in /Users/XXXXXXXXXX/.ssh/known_hosts to get rid of this
message.
Offending RSA key in /Users/XXXXXXXXXX/.ssh/known_hosts:1
RSA host key for 202.120.58.XXX has changed and you have requested strict checki
ng.
Host key verification failed.
```

若遇到上方图片中的问题，请重置 known_hosts，命令如下：

```
(在自己电脑上) $ ssh-keygen -R sylogin.hpc.sjtu.edu.cn
```

调试 SSH 登录问题

有多种原因可能会阻止您登录到集群。

1. 连续多次错输密码会被临时封禁 1 小时。集群登录节点设置了 fail2ban 服务，多次输入密码错误后会被临时封禁 1 小时。
2. 若在登录节点运行计算密集的作业，程序会被自动查杀，您的账号会被加入到黑名单，并在 30-120 分钟内无法登录。

若需重置密码，请使用或抄送账号负责人邮箱发送邮件到 [HPC 邮箱](#)，我们将会在 1 个工作日内响应您的申请。

排查登录问题，还可以使用 ping 命令检查您的电脑和集群连接状态。

```
$ ping sylogin.hpc.sjtu.edu.cn
```


登录常掉线的问题

如果 SSH 客户端长时间静默后，SSH 服务器端会自动断开相关会话。要解决这个问题，需要调整 SSH 的 `keepalive` 值，设置一个较长的静默时长阈值。

Mac/Linux 用户

对于 Mac/Linux 用户，并且使用操作系统原生的终端 (terminal)，需要修改 `$HOME/.ssh/config`。具体的，在文件中添加如下内容：

```
Host pi-sjtu-login:
  HostName sylogin.hpc.sjtu.edu.cn
  ServerAliveInterval 240
```

其中 `ServerAliveInterval` 后的值即为阈值，单位为秒，用户可根据需要自行调整。或者为了对所有的服务器设置长静默阈值：

```
Host *
  ServerAliveInterval 240
```

之后保持 `config` 文件为只可读：

```
chmod 600 ~/.ssh/config
```

Windows SSH 客户端用户

这里我们以 Putty 为例。市面有不同的 SSH 客户端，您可以根据自身情况自行搜索您使用的 SSH 客户端的设置方法。

在 Putty 的 Session 的属性中，Connection->Sending of null packets to keep session active -> Seconds between keepalives (0 to turn off) 后的文本框中，输入对应的值，如 240。

4.6.2 Tmux 使用介绍

Tmux 是一个终端复用器 (terminal multiplexer)。如果您有使用 screen 的经历的话，您可以理解为 Tmux 是 screen 的不同实现软件。本教程将讲解 Tmux 的基础用法。

会话与进程

命令行的典型用法是打开终端 (terminal) 后，在里面输入指令。用户的这种与计算机交互的手段，称为会话 (session)。

在会话中，通过命令行启动的所有进程均与会话进程绑定。当会话进程终止时，该会话启动的所有进程也会随之强行结束。

一点最常见的例子就是通过 SSH 连接到远程计算机。当 SSH 连接因为网络等原因断开时，那么 SSH 会话就被终止，这次会话启动的任务也会被强制结束。

为了解决这个问题，一种手段就是用户终端窗口与会话“解绑”。即关闭用户端窗口，仍然维持该会话，进而保证用户进程不变。

Tmux 的作用

Tmux 就是这样一款会话与窗口的“解绑”工具。

- (1) 它允许在单个窗口中，同时访问多个会话。这对于同时运行多个命令程序很有用。
- (2) 它可以让新窗口“接入”已经存在的会话。
- (3) 它允许每个会话有多个连接窗口，因此可以多人实时共享会话。
- (4) 它还支持窗口任意的垂直和水平拆分

基本用法

安装

集群中已经默认安装了 Tmux, 无须操作。如果您需要在自己的服务器上安装 Tmux, 请参考以下指令:

```
# Ubuntu 或 Debian
$ sudo apt-get install tmux

# CentOS 或 Fedora
$ sudo yum install tmux

# Mac
$ brew install tmux
```

启动与退出

直接在终端中键入 `tmux` 指令，即可进入 Tmux 窗口。

```
$ tmux
```

上面命令会启动 Tmux 窗口，底部有一个状态栏。状态栏的左侧是窗口信息（编号和名称），右侧是系统信息。

```
[hpcwj@login3 ~]$
```



按下 `Ctrl+d` 或者显式输入 `exit` 命令，就可以退出 **Tmux** 窗口。

```
$ exit
```

快捷键

Tmux 有大量的快捷键。所有的快捷键都要使用 `Ctrl+b` 作为前缀唤醒。我们将会在后续章节中讲解快捷键的具体使用。

会话管理

新建会话

第一个启动的会话名为 `0`，之后是 `1`、`2` 一次类推。

但是有时候我们希望为会话起名以方便区分。

```
$ tmux new -s SESSION_NAME
```

以上指令启动了一个名为 `SESSION_NAME` 的会话。

分离会话

如果我们想离开会话，但又不想关闭会话，有两种方式。按下 `Ctrl+b d` 或者 `tmux detach` 指令，将会分离会话与窗口

```
$ tmux detach
```

后面一种方法要求当前会话无正在运行的进程，即保证终端可操作。我们更推荐使用前者。

查看会话

要查看当前已有会话，使用 `tmux ls` 指令。

```
$ tmux ls
```

接入会话

`tmux attach` 命令用于重新接入某个已存在的会话。

```
# 使用会话编号
$ tmux attach -t 0

# 使用会话名称
$ tmux attach -t SESSION_NAME
```

杀死会话

`tmux kill-session` 命令用于杀死某个会话。

```
# 使用会话编号
$ tmux kill-session -t 0

# 使用会话名称
$ tmux kill-session -t SESSION_NAME
```

切换会话

`tmux switch` 命令用于切换会话。

```
# 使用会话编号
$ tmux switch -t 0

# 使用会话名称
$ tmux switch -t SESSION_NAME
```

`Ctrl+b s` 可以快捷地查看并切换会话

重命名会话

`tmux rename-session` 命令用于重命名会话。

```
# 将0号会话重命名为SESSION_NAME
$ tmux rename-session -t 0 SESSION_NAME
```

对应快捷键为 `Ctrl+b $`。

窗格 (window) 操作

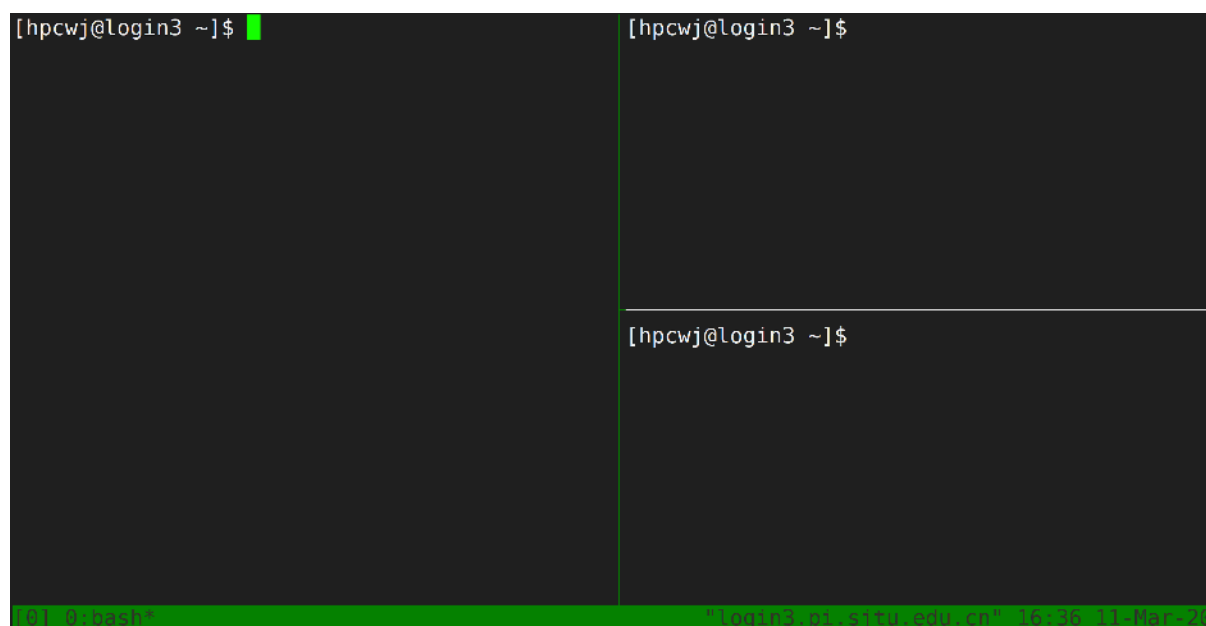
Tmux 可以将窗口分成多个窗格 (window)，每个窗格运行不同的命令。以下命令都是在 Tmux 窗口中执行。

划分窗格

tmux split-window 命令用来划分窗格。

```
# 划分上下两个窗格
$ tmux split-window

# 划分左右两个窗格
$ tmux split-window -h
```



对应快捷键为 `Ctrl+b "` 和 `Ctrl+b %`

移动光标

tmux select-pane 命令用来移动光标位置。

```
# 光标切换到上方窗格
$ tmux select-pane -U

# 光标切换到下方窗格
$ tmux select-pane -D

# 光标切换到左边窗格
$ tmux select-pane -L

# 光标切换到右边窗格
$ tmux select-pane -R
```

对应快捷键为 `Ctrl+b ↑`、`Ctrl+b ↓`、`Ctrl+b ←`、`Ctrl+b →`。

窗格快捷键

```
$ Ctrl+b % : 划分左右两个窗格。
$ Ctrl+b " : 划分上下两个窗格。
$ Ctrl+b <arrow key> : 光标切换到其他窗格。<arrow key>
→是指向要切换到的窗格的方向键，比如切换到下方窗格，就按方向键↓。
$ Ctrl+b ; : 光标切换到上一个窗格。
$ Ctrl+b o : 光标切换到下一个窗格。
$ Ctrl+b { : 当前窗格左移。
$ Ctrl+b } : 当前窗格右移。
$ Ctrl+b Ctrl+o : 当前窗格上移。
$ Ctrl+b Alt+o : 当前窗格下移。
$ Ctrl+b x : 关闭当前窗格。
$ Ctrl+b ! : 将当前窗格拆分为一个独立窗口。
$ Ctrl+b z : 当前窗格全屏显示，再使用一次会变回原来大小。
$ Ctrl+b Ctrl+<arrow key> : 按箭头方向调整窗格大小。
$ Ctrl+b q : 显示窗格编号。
```

4.6.3 交我算定制版客户端 Tabby

Tabby (前身是 Terminus) 是一个基于 TypeScript 开发，可高度配置的终端模拟器和 SSH 或串口客户端，支持 Windows、Mac OS 和 Linux 系统，功能强大，包括：

- 集成 SSH、Telnet 客户端和连接管理器
- 集成串行终端
- 定制主题和配色方案
- 完全可配置的快捷键和多键快捷键
- 拆分窗格
- 记忆标签
- 支持 PowerShell (和 PS Core)、WSL、Git-Bash、Cygwin、MSYS2、Cmder 和 CMD 等
- 丰富的插件
- 支持 SFTP 功能

针对用户登录不上集群时难以定位原因的问题，交我算团队为用户开发了定制版 Tabby。定制版 Tabby 有三大特点：

- 预置交我算平台各集群主机名，便捷登录
- 登录自检功能，快速定位登录问题并输出提示，提高登录效率
- SFTP 自动读取家目录，方便数据传输

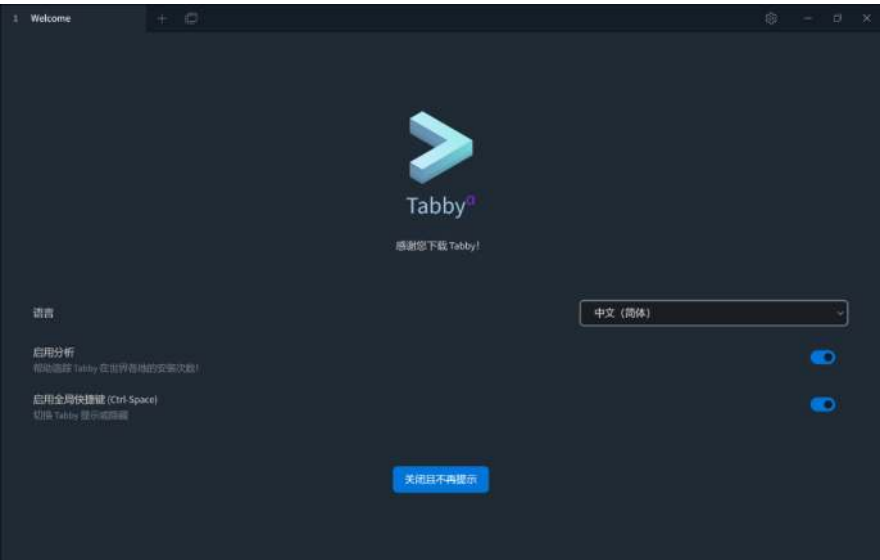
下载和安装

定制版 Tabby 支持多平台，包括 Windows/Mac O/Linux 系统 Intel 或 Arm 架构，下载好后安装即可。[下载地址](#)

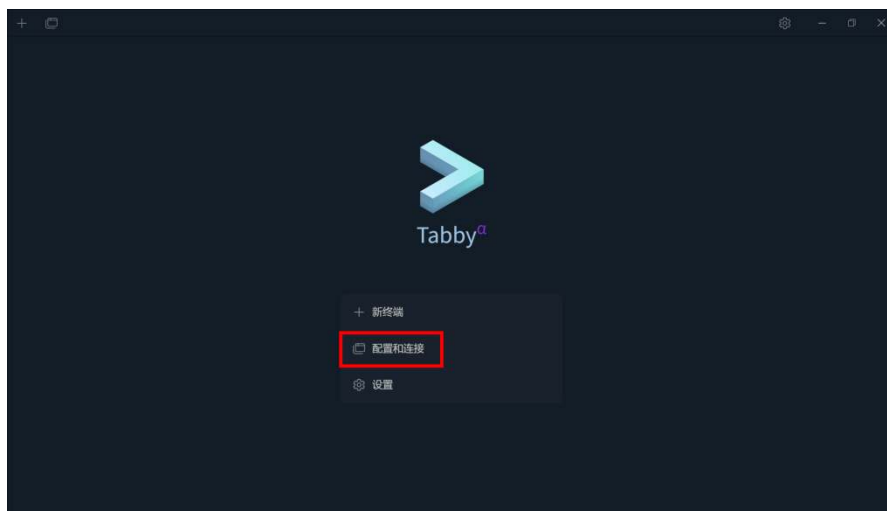
版本	安装包名称
Windows X64	tabby-1.0.187-94-setup-x64
Windows arm64	tabby-1.0.187-94-setup-arm64
MacOS arm64	tabby-1.0.187-94-macos-arm64.dmg
MacOS X86-64	tabby-1.0.187-94-macos-x86_64.dmg
Linux X64	tabby-1.0.187-94-linux-x64.ApplImage

通过 Tabby 登录集群

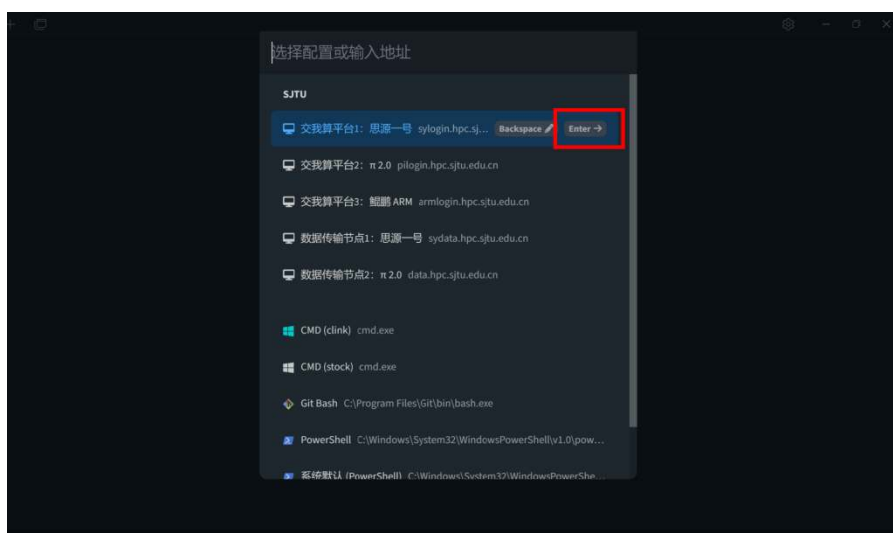
以 Windows X64 版本为例（其他版本使用方法基本无差异）* 首次打开 Tabby 后，可以自行设定语言，如“中文（简体）”。



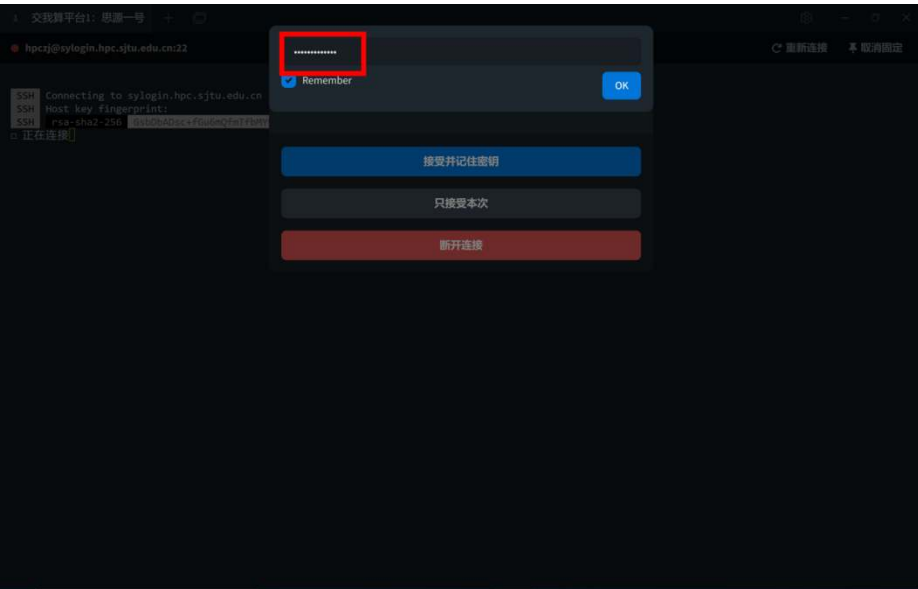
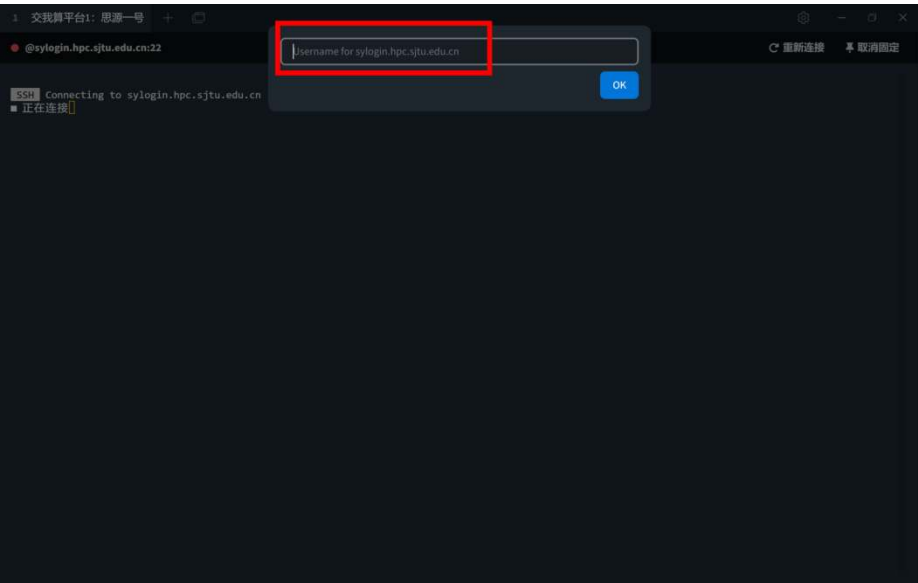
- 进入如下主界面，如需要直接登录集群，则点击“配置和连接”。



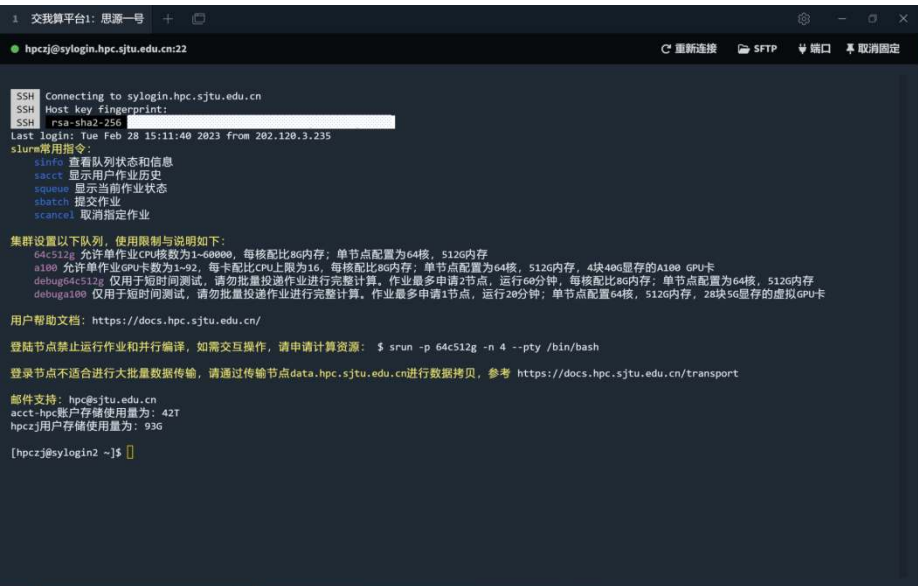
- 可以看到里面预置了五个主机名，包括三大平台的登录节点以及数据传输节点。选择需要登录的集群，以“思源一号”登录节点为例，点击 **enter**。



- 首次登录时，根据提示输入用户名和密码，选择“Remember”记住密码则下一次可以自动登录。如跳出“主机密钥校验”，选择“接受并记住密钥”。

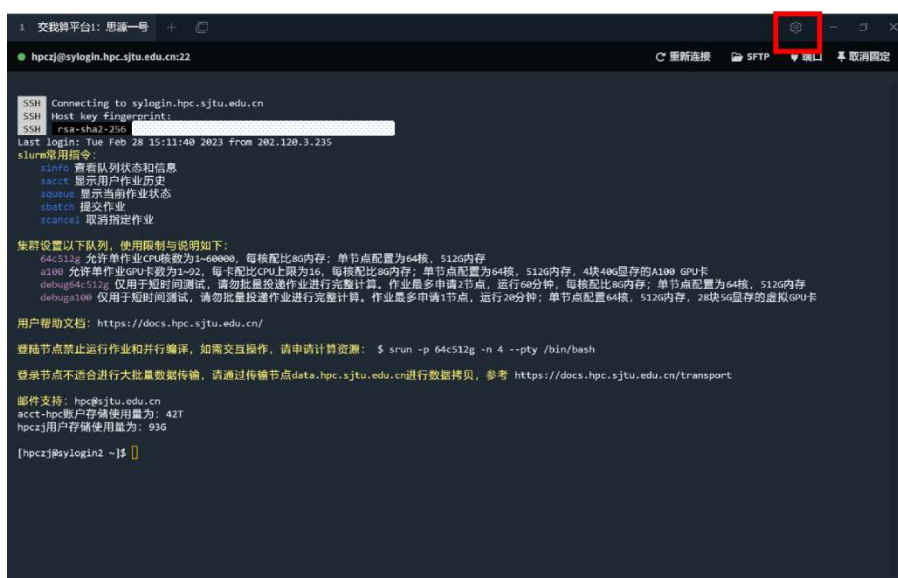
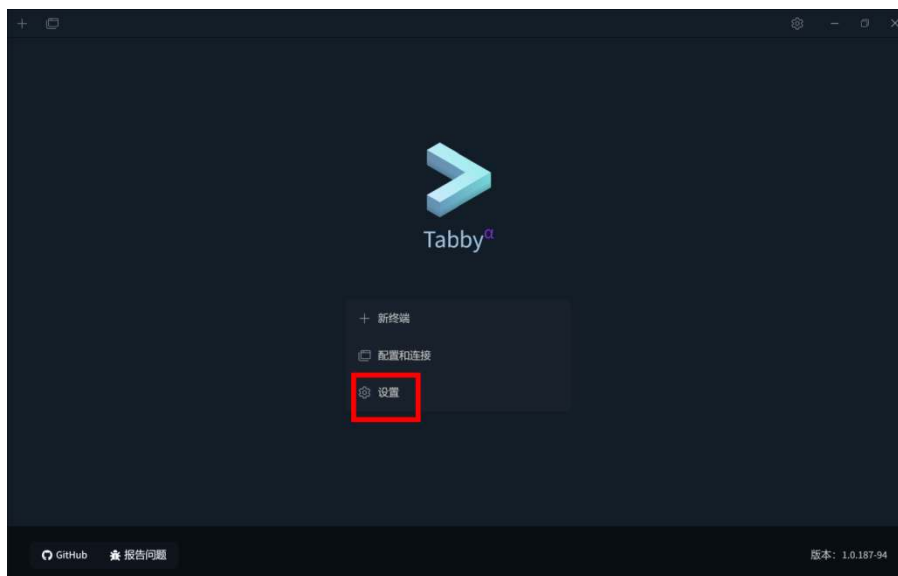


- 登录成功。



参数设置

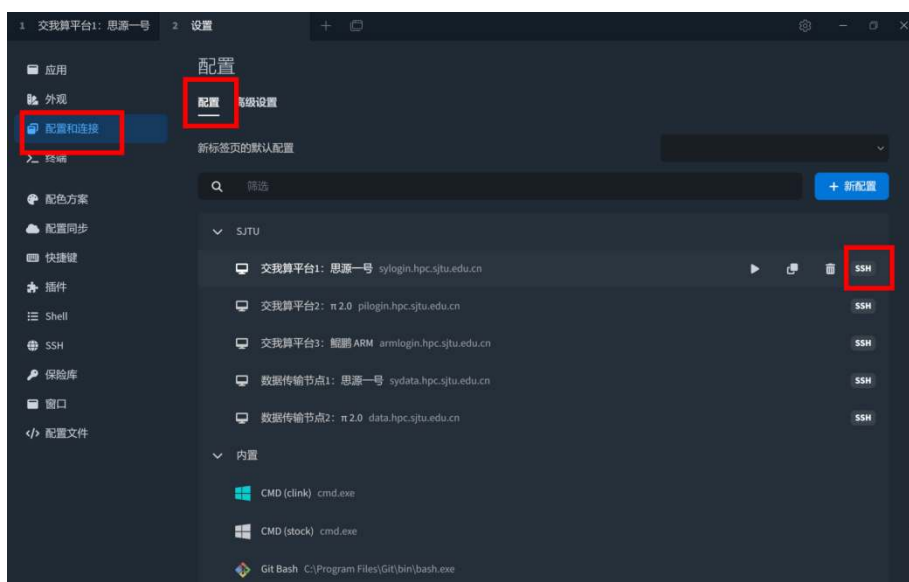
- 在主界面或登录界面均可以进入到“设置”页面。

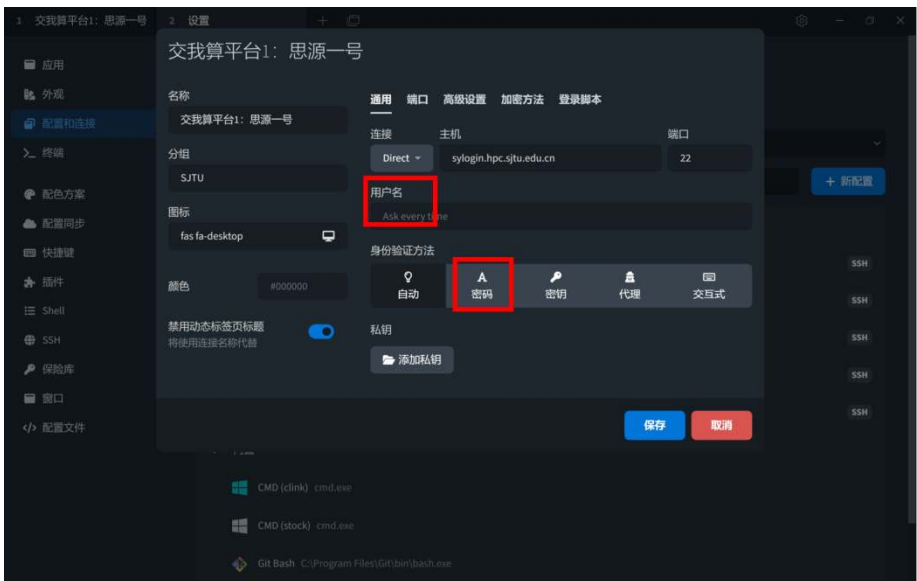


- 在“设置”页面可以进行应用、外观、配色、配置连接等各种设置。

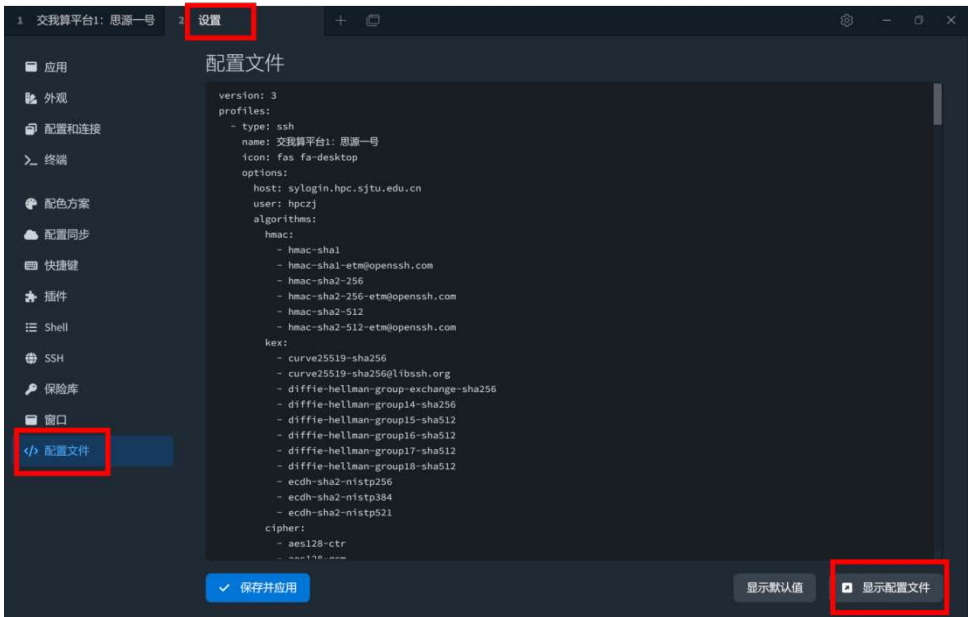


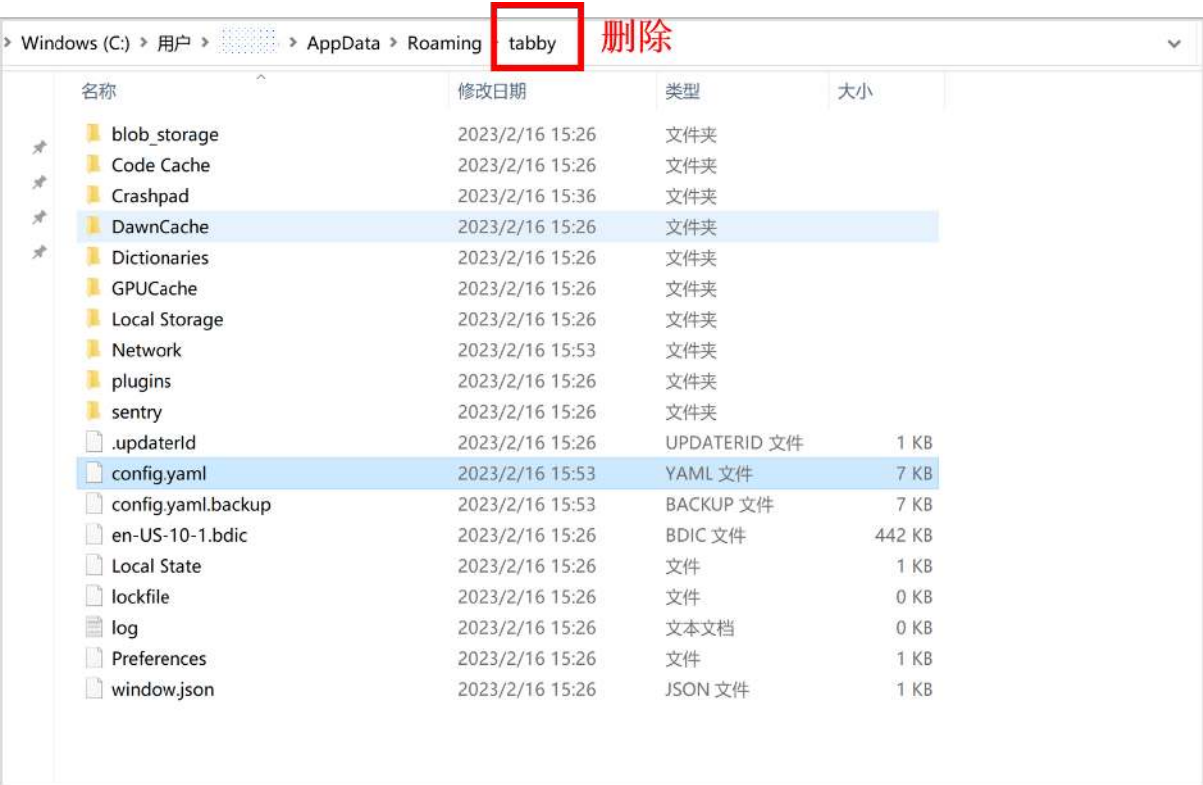
- Tabby 自动保存第一次登录成功的用户名和密码，之后打开 Tabby 将自动登录。如需要更改用户名和身份验证方法，则进入设置-配置和连接-配置，点击对应的 SSH，再更改用户名或身份验证方法。





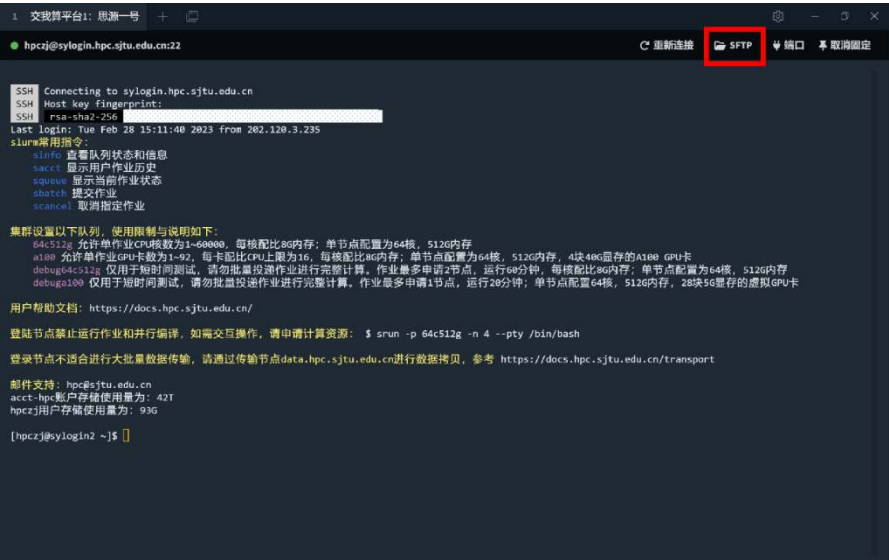
- 如果需要清除之前的配置，可以点击“设置”，选择“配置文件”，点击“显示配置文件”，资源管理器会打开配置文件所在的目录。返回上一级目录，将整个 tabby 目录全部删除。

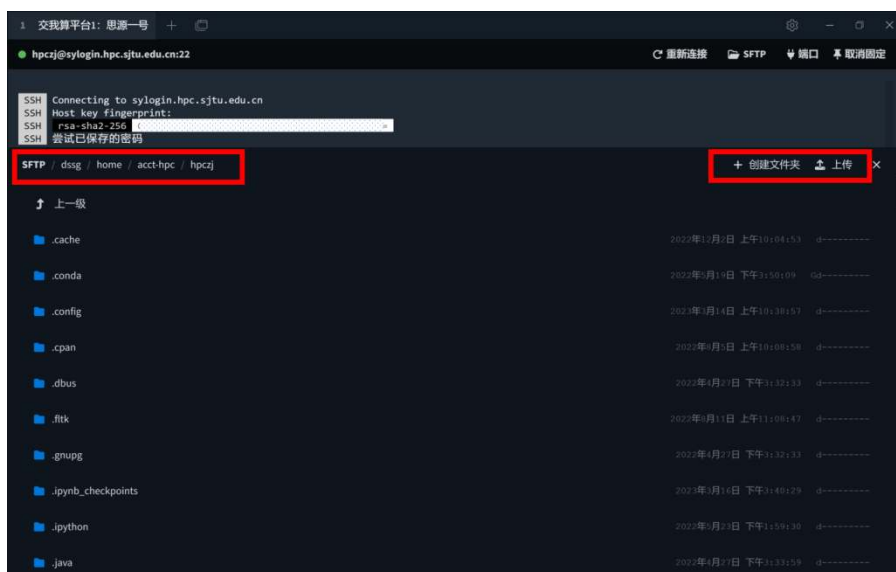




SFTP 功能

在登录页面点击” SFTP”，可以进入用户家目录进行文件夹创建、文件上传，双击文件可以直接下载



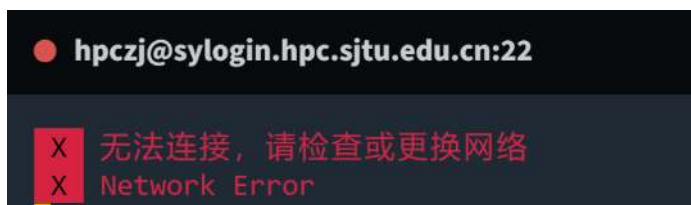


需要注意的是

- (1) Tabby的SFTP仅支持上传和下载文件，不支持目录。
- (2) 可将目录压缩或打包后，再上传或下载。
- (3) 大文件下载请使用数据传输节点。

常见报错

- 网络连接问题



- 密码输入错误



- 密码输入错误次数过多



- 登录 session 过多

邮件支持: hpc@sjtu.edu.cn

SSH sylogin.hpc.sjtu.edu.cn: session closed

X 无法与服务器建立连接, 当前用户的连接数过多

- 账号被冻结

邮件支持: hpc@sjtu.edu.cn

This account is currently not available.

X 帐号被暂时封禁, 请待解禁后重试, 或发送邮件申请解封

- 集群停机维护

1 交我算平台1: 思源一号

hpczj@sylogin.hpc.sjtu.edu.cn:22

SSH Connecting to sylogin.hpc.sjtu.edu.cn

X 无法与服务器建立连接, 请查看用户微信群是否有停机通知

X read ECONNRESET

常见问题

Q1: Mac 用户安装时可能遇到“已损坏, 无法打开, 您应该将它移到废纸篓”的提示。

A1: 参考链接 <https://zhuanlan.zhihu.com/p/135948430> 进行设置。

4.6.4 VS Code 连接集群

Visual Studio Code (简称 VS Code) 是一款由微软开发且跨平台的免费源代码编辑器。该软件支持语法高亮、代码自动补全 (又称 IntelliSense)、代码重构功能, 并且内置了命令行工具和 Git 版本控制系统。

VS Code 经过配置, 可以远程连接到 Pi 集群及思源一号, 在本地进行远程的开发部署工作。但 请注意您依然需要有一个常规的 SSH 登录会话以向平台证明您处于活跃状态, 否则您的 **vscode** 连接可能会被视为残留的僵尸进程而被自动清理掉。

安装兼容的 SSH 客户端

首先需要在本地电脑上安装 OpenSSH 兼容的 SSH 客户端 (Putty 不支持)。

对于 Mac, 系统自带的 SSH 客户端就可满足需求, 无需安装。

对于 linux 用户, 需要安装 openssh-client。

运行


```
$ sudo apt-get install openssh-client
```

或者

```
$ sudo yum install openssh-client
```

对于 Windows 用户，请安装 Windows OpenSSH Client。Windows 用户可以使用 Windows 设置或者 PowerShell 来安装该客户端，具体请参考链接 [安装 OpenSSH](#)。

使用密码登录集群

在 `vscode` 中选择 **连接到主机** 并发起 SSH 连接后，右下角会显示一个正在连接中的提示小窗，点击提示信息中高亮的 `details`，`vscode` 会给您切换到终端标签页，在这里您将被要求输入密码。如果一段时间没有完成密码输入，`vscode` 会按超时处理放弃连接。

和 SSH 客户端进行登录一样，如果您已经绑定了 `jAccount`，您可以在这里使用 `jAccount` 密码登录。

免密登录集群

要进行免密登录首先需要一份在有效期内的免密证书，申请免密证书请参考[此章节](#)。

获取到免密证书后，您需要在本地的 `ssh` 会话配置中指定使用它。本地配置文件的位置取决于您的系统环境，`linux` 环境通常在 `~/.ssh/config`，`windows` 环境通常在 `C:\Users\[YOUR_USERNAME]\.ssh\config`，如果您不确定实际位置，可以从 `vscode` 的 **连接到主机 - 配置 SSH 主机** 选项菜单中查看官方给你的候选路径。以下是一个配置实例，请根据自己的实际情况变通：

```
Host bob_want_password_free_login
  HostName pilogin.hpc.sjtu.edu.cn
  User bob
  Port 22
  IdentityFile C:\Users\Bob\Documents\MobaXterm\home\.ssh\id_ed25519
  CertificateFile C:\Users\Bob\Documents\MobaXterm\home\.ssh\id_ed25519-
    ↪cert.pub

Host charley_want_to_use_password
  HostName sylogin.hpc.sjtu.edu.cn
  User charley
  Port 22
```

配置完毕后请在本地终端测试是否能访问集群。

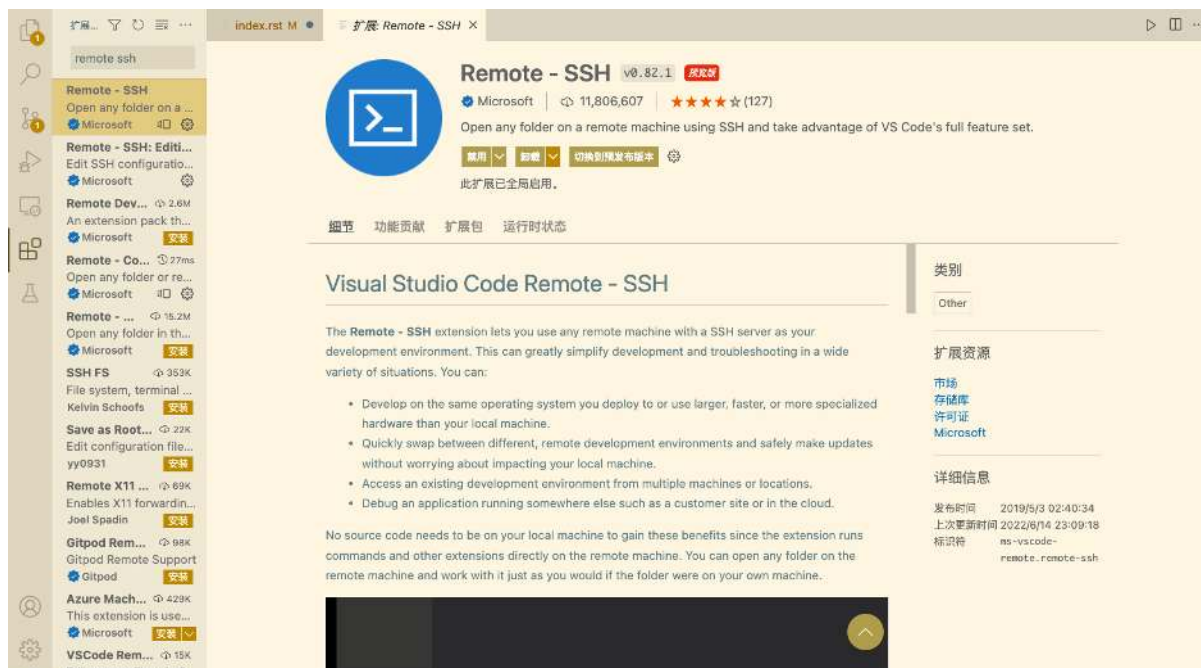
```
$ ssh bob_want_password_free_login
```


本地安装 VS Code 及插件（可选）

本章节介绍的插件可以优化使用体验，并非必须。

请至 [VS code download](#) 下载于本地操作系统对应的 VS Code 安装包并根据步骤安装。

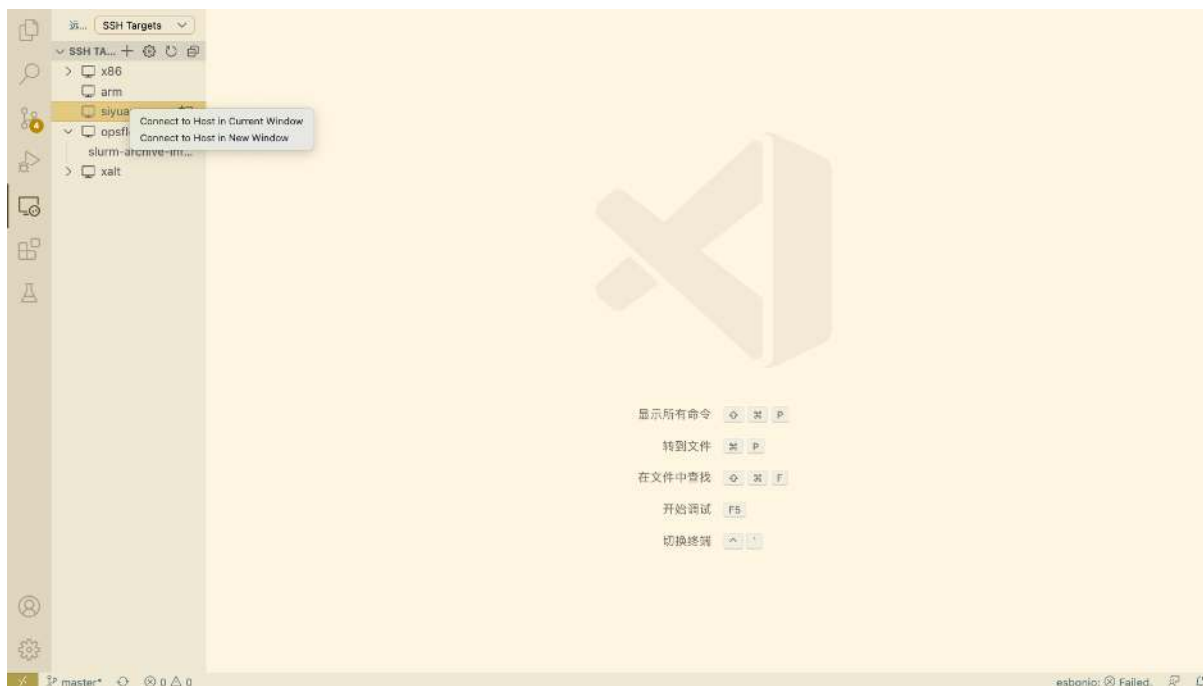
打开 VS Code 软件，安装 Remote SSH 插件。



安装完毕后点开左方工具栏中 `remote-ssh` 插件的图标，该插件会自动读取 `~/.ssh/config` 中的主机名。



右键相应的主机名即可选择连接主机：



此时会弹出窗口要求输入先前设置的 passphrase:



输入密码后即可链接至远程主机:



连接后可选择打开文件夹或者终端：



4.6.5 使用 X Server 显示图形界面

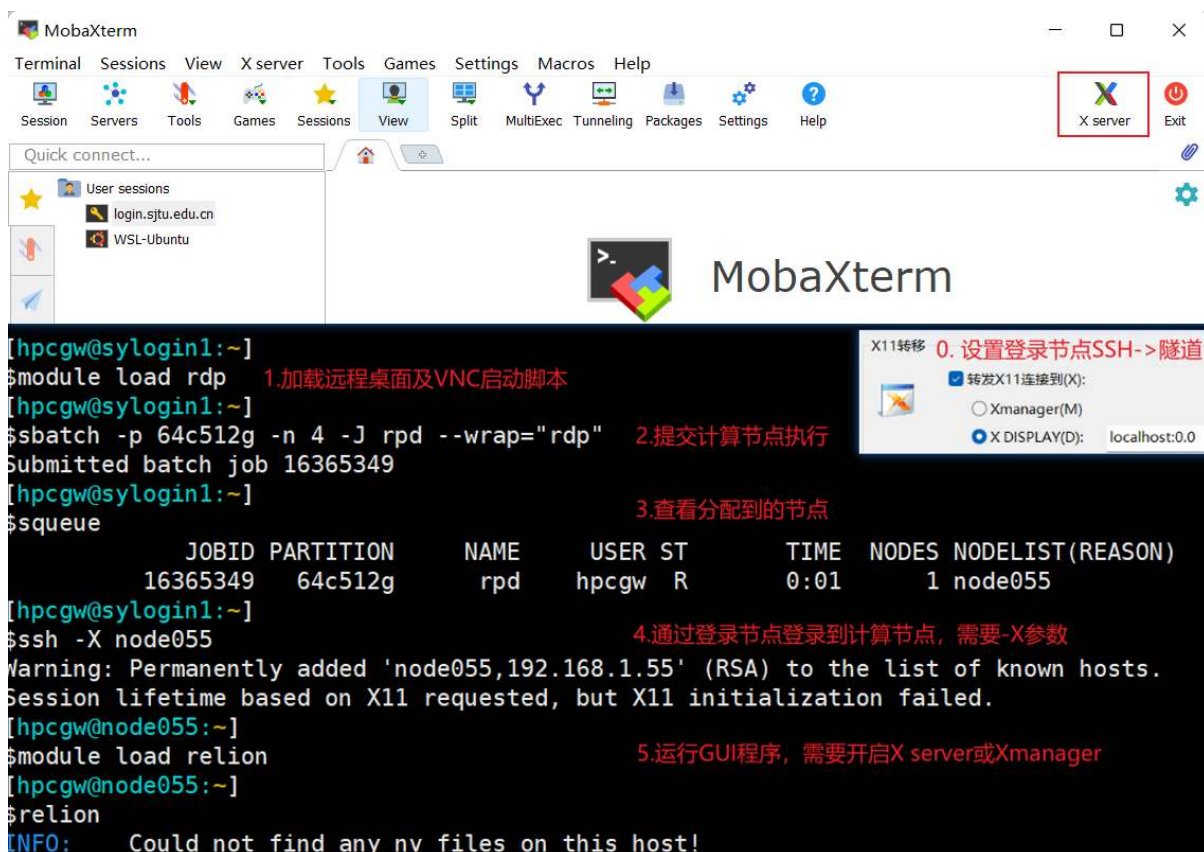
X11 是 UNIX 上图形视窗系统的标准规范，主要分为 3 个部分：X Server (X 服务器)、X Client (X 客户端)、Window Manager (窗口管理器)。X Server 是整个 X Window System 的中心，协调 X 客户端和窗口管理器的通信。

X Server 运行在本地，推荐 MobaXterm 终端工具，包含了 x server 功能。使用前需要确认开启了 X11-Forwarding，并启动 X server。

启动步骤

以思源一号为例：

```
module load rdp # 加载远程桌面及VNC启动脚本
sbatch -p 64c512g -n 4 -J rdp --wrap="rdp" # 提交计算节点执行
queue # 查看分配到的节点
ssh -X node055 #
→通过登录节点登录到计算节点，需要-X参数
module load relion # 运行GUI程序
relion
```



警告

X Server 显示图形界面的方式需要保持 SSH 连接，不然 GUI 程序会中断运行。

4.6.6 参考资料

- <http://www.ee.surrey.ac.uk/Teaching/Unix/>
- http://vbird.dic.ksu.edu.tw/linux_server/0310telnetssh.php#ssh_server
- <http://nerderati.com/2011/03/simplify-your-life-with-an-ssh-config-file/>
- <http://www.cyberciti.biz/faq/ssh-passwordless-login-with-keychain-for-scripts/>
- <https://stackoverflow.com/questions/25084288/keep-ssh-session-alive>

- <https://patrickmn.com/aside/how-to-keep-alive-ssh-sessions/>
- <https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/>
- <https://danielmiessler.com/study/tmux/>
- <https://linuxize.com/post/getting-started-with-tmux/>
- <https://www.ruanyifeng.com/blog/2019/10/tmux.html>

4.7 文件系统与数据访问

4.7.1 交我算账号的数据共享

交我算的主帐号和子帐号都为独立帐号，仅在计费关系上存在关联。若有数据共享需求，可由主帐号负责人发邮件到 **hpc** 邮箱。

邮件模板：

我是超算账号 `expuser01`（主账号名）使用人，已经阅读超算账号数据共享文档，现需要在思源一号 / $\rightarrow$ Pi2.0 集群建立共享文件夹。

使用方式：课题组内部共享 / 课题组之间共享

其他账号：`expuser02`, `expuser03`（可选，若为课题组内部共享则无需填写）

备注

π 2.0/AI/ARM 集群及思源一号集群使用两个独立的存储系统，因此共享文件夹需要分开申请。

数据共享场景

目前数据共享场景包括：课题组内部共享（主账号和子账号的共享）、课题组之间共享（不同主账号之间的共享）、共享数据到集群外。

课题组内部共享

- 如何申请：由主账号负责人向 **hpc** 邮箱发送邮件，申请在主账号下建立 `acct-expuser01/share` 文件夹。
- 权限控制：主账号和子账号默认都可以读写，如果需要自定义权限，请在邮件中注明。
- 如何使用：将要共享的数据放在此文件夹下，就可以在主账号和子账号间共享数据。

课题组之间共享

- 如何申请：由主账号负责人 A 向 hpc 邮箱发送邮件，申请在其主账号下建立 acct-expuser01/share 文件夹。另外需注明需要开通读权限的其他账号。
- 权限控制：默认只有计费的主账号 A 能写入，其他账号为只读，如果需要自定义权限，请在邮件中注明。
- 如何使用：将要共享的数据放在此文件夹下，就可以在多个账号间共享数据。
- 如何计费：共享数据的存储费会由主账号 A 承担。

共享数据到集群外

将“交我算”账号的数据共享给非超算用户，可以使用 <https://scidata.sjtu.edu.cn/>提供的服务。通过可视化平台在超算上打开平台网站，上传数据后分享链接至他人。具体操作步骤可以查看文档：

- 科学数据平台使用指南

其他问题

1. 用户可以自行设置共享文件夹的权限吗？

可以，在申请建立共享文件夹之后，主账号负责人可以自行设置共享文件夹的权限，将数据共享给其他账号。以下示例假设主账号用户 expuser01 在 PI 的个人目录为 /lustre/home/acct-exp/expuser01，思源一号个人目录 /dssg/home/acct-exp/expuser01，在其项目组下设立共享文件夹 share。

警告

添加数据共享权限时，请注意数据安全，谨慎添加权限，尤其是写权限。

```
# 查看共享文件夹的ACL权限，rwx代表读写权限，r-x代表只读权限
# 在data, sylogin, sydata等节点访问/dssg存储时
mmgetacl /dssg/home/acct-exp/share
# 在data, pilogin, armlogin等节点访问/lustre存储时
getfacl /lustre/home/acct-exp/share

# 授予子账号expuser02只读权限，expuser03读写权限
# 在data, sylogin, sydata等节点访问/
→dssg存储时，请准备一个文本文件作为指令输入，一次性描述您期望的最终权限配置。
vim input.txt
    user::rwx
    group::r-x-
    other::r-x-
    mask::rwx
    user:expuser02:r-x
    user:expuser03:rwx
mmputacl -i input.txt /dssg/home/acct-exp/share
```

(续下页)

(接上页)

```
# 在 data, pilogin, armlogin 等节点访问 /
→ lustre 存储时, 您需要逐个添加、删除目标用户的权限。
setfacl -m u:expuser02:r-x /lustre/home/acct-exp/share
setfacl -m u:expuser03:rw-x /lustre/home/acct-exp/share
```

2. 如何将共享文件夹中的文件设置成只读?

假设某个账号需要共享一个文件, 并且希望这个文件别人无法修改, 这时可以将文件权限修改为 644, 只让文件的所有者能修改, 其他人为只读。

```
# 假设主账号用户 expuser01 在思源一号个人目录 /dssg/home/acct-exp/
→ expuser01, 共享文件夹路径为 /dssg/home/acct-exp/
→ share, 需要共享的文件名为 file
# 查看文件的权限, 第三列的用户名代表所有者
ls -l /dssg/home/acct-exp/share/file

# 修改文件的权限为 644 (rw-r--r--), 让其他用户只能读
chmod 644 /dssg/home/acct-exp/share/file
```

3. 用户可以自行设置家目录下文件的 ACL 权限来共享数据吗?

以 /lustre/home/acct-exp/expuser01/hello.txt 为例, 其他用户要访问 hello.txt, 需要有权限进入其上层父目录 expuser01, 即 expuser01 的家目录。默认情况下超算平台为用户家目录设置的权限为 rwx-----, 因此仅对 hello.txt 添加 ACL 配置还不足以让其他用户访问到此文件。如果确实需要共享家目录中的数据, 您还需要对家目录本身添加对象用户的进入权限 x, 例如:

```
setfacl -m u:expuser02:r-x /lustre/home/acct-exp/expuser01
```

在您决定打开家目录的大门之前, 您应当仔细检查家目录中各个文件的权限配置, 请确保其他不希望共享的文件没有 **other** 域授权, 私有数据的权限位不应多于 **rw-xr-x---**

4.7.2 数据传输方案和传输节点

交我算 HPC+AI 集群包含两个存储池, 其中 **lustre** 存储用于 small, cpu, huge, 192c6t, dgx2, arm128c256g 以及相应的 debug 队列, 对应的 home 目录路径为 /lustre/home/acct-主帐号/主帐号或子帐号; **gpfs** 存储用于 64c512g, a100 以及相应的 debug 队列, 对应的 home 目录路径为 /dssg/home/acct-主帐号/主帐号或子帐号。

传输方案

对于数据传输, 我们为您提供如下解决方案:

1. 少量数据传输, 集群提供了专门用于数据传输的节点 (data.hpc.sjtu.edu.cn, sy-data.hpc.sjtu.edu.cn), 可以直接使用 putty, filezilla 等客户端, 或在本地使用 scp, rsync 命令向该节点发起传输请求 (因安全策略升级, 在集群的终端上不支持 scp/rsync 的远程传输功能, 所以需要从用户本地终端使用 scp/rsync 命令)。
2. 1TB-1PB 数据传输, 强烈建议您联系我们, 将硬盘等存储设备送至网络信息中心进行传输。

3. 超过 1PB 的数据，请您与我们联系，由计算专员根据具体情况为您解决数据传输问题。

数据传输节点

思源一号传输节点地址为 `sydata.hpc.sjtu.edu.cn`

π 2.0/AI/ARM 集群传输节点地址为 `data.hpc.sjtu.edu.cn`

注意：

1. 推荐使用数据传输节点（而不是登录节点）进行数据传输，不会占用登录节点资源并且多进程或多用户同时传输不会受限于 CPU。
2. 数据传输节点仅用于批量数据传输，请勿在此节点上运行与数据传输无关的应用，如编译程序、管理作业等。如果发现此类行为，平台将视情况取消相关帐号使用数据传输节点的权利。

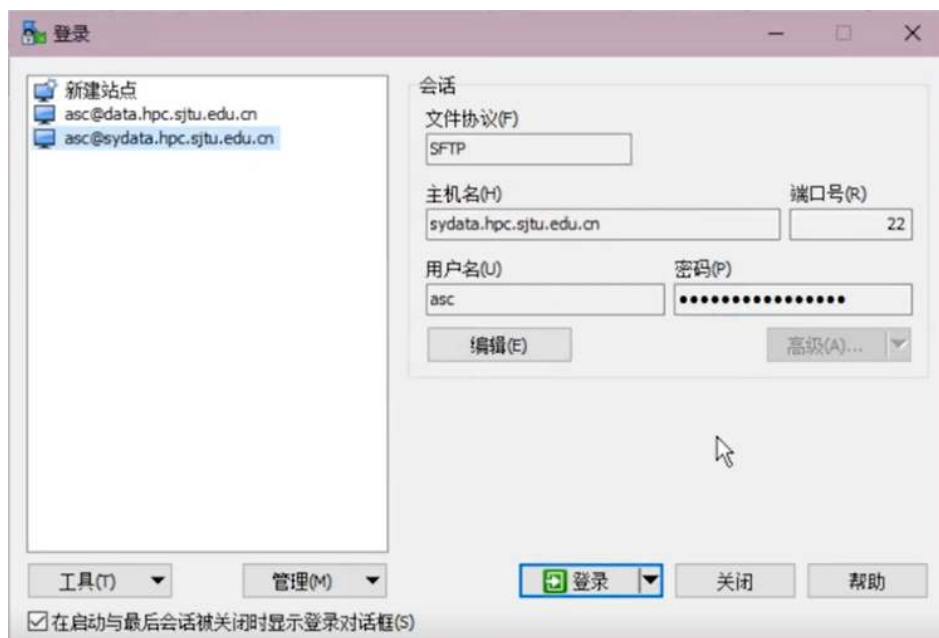
4.7.3 数据传输方法

本地向思源一号传输

Windows 用户

Windows 用户可以使用 WinSCP 在集群和您自己的计算机之间传输文件。可至 [WinSCP 官网](#) 下载。

如下图所示，填写思源一号节点的地址，SSH 端口，SSH 用户名，SSH 密码，然后点击 Login 进行连接。使用 WinSCP 的方法类似于使用 FTP 客户端 GUI，如下图所示：



登录后即可看见左右两栏文件。左侧是本地文件，右侧是集群上的文件。点击需要传输的文件进行拖动即可传输。

Linux/Unix/Mac 用户

Linux/Unix/Mac 用户可通过在终端中使用 `scp` 或 `rsync` 等命令传输。

1. 如果传输的对象为少量大文件，且目标环境上没有数据的历史版本，所有需要传输的文件都是首次传输，可以使用 `scp` 直接拷贝文件。

```
$ scp -r [源文件路径] [目标路径]
```

2. 如果需要传输的对象为包含大量文件的目录，或者目标环境上已经存在差异较小的历史版本，建议使用 `rsync` 拷贝数据，`rsync` 会对比源地址和目标地址的内容差异，然后进行增量传输。

```
$ rsync --archive --partial --progress [源文件路径] [目标路径]
```

如果 [源文件路径] 或 [目标路径] 位于思源一号集群上，则路径需使用以下格式：

```
[用户名]@sydata.hpc.sjtu.edu.cn:[思源一号上的路径]
```

如果 [源文件路径] 或 [目标路径] 位于本地，则不需要加 [用户名]@ 主机名，可直接写文件路径。

```
# 假设用户 expuser01 在思源一号平台上个人目录为 /dssg/home/acct-exp/expuser01
# 本地个人目录为 /home/local_user/（个人目录可以用 ~ 代替）

# 示例 1：将本地目录 ~/data 的全部数据上传至思源一号 dssg 目录下
$ scp -r /home/local_user/data/ expuser01@sydata.hpc.sjtu.edu.cn:/dssg/home/
→acct-exp/expuser01/

# 示例 2：将 dssg 目录中的 ~/math.dat 文件下载到本地个人目录
$ scp expuser01@sydata.hpc.sjtu.edu.cn:/dssg/home/acct-exp/expuser01/math.
→dat /home/local_user/

# 示例 3：将 dssg 目录 ~/data 的数据下载到本地 ~/
→download 目录，请注意 rsync 不支持双远端传输，必须在目标主机（这里即为本地）上操作
$ rsync --archive --partial --progress expuser01@data.hpc.sjtu.edu.cn:/dssg/
→home/acct-exp/expuser01/data/ /home/local_user/download/
```

本地向 π 2.0/AI/ARM 集群传输

Windows 用户

使用 WinSCP，方法和本地向思源一号传输类似，只需要将节点地址改成 `data.hpc.sjtu.edu.cn`。

Linux/Unix/Mac 用户

方法和本地向思源一号传输类似。

```
# 假设用户 expuser01 在  $\pi$  2.0 集群上个人目录为 /lustre/home/acct-exp/expuser01

# 示例 4：将本地目录 ~/data 的全部数据上传至 lustre 目录下
$ scp -r /home/local_user/data/ expuser01@data.hpc.sjtu.edu.cn:/lustre/home/
→acct-exp/expuser01/

# 示例 5：将 lustre 目录 ~/data 的数据下载到本地 ~/
→download 目录，请注意 rsync 不支持双远端传输，必须在目标主机上操作
```

(续下页)

(接上页)

```
$ rsync --archive --partial --progress expuser01@data.hpc.sjtu.edu.cn:/
↪lustre/home/acct-exp/expuser01/data/ /home/local_user/download/
```

思源一号与 π 2.0/AI/ARM 集群互传

要提高集群之间互传数据的效率，需要选择合适的传输节点。

- 如果需要在 π 超算和思源一号互传数据，需要选择 **data** 或者 **sydata** 节点发起传输
- 如果需要从 π 超算向冷存储搬运数据，需要选择 **data** 节点发起传输
- 如果需要从思源一号向冷存储搬运数据，需要选择 **sydata** 节点发起传输

例如通过登录 π 2.0 集群数据传输节点 `data.hpc.sjtu.edu.cn`，使用 `scp` 或 `rsync` 命令进行传输：

```
$ scp -r [源文件路径] [目标路径]
```

```
$ rsync -avr --progress [源文件路径] [目标路径]
```

此时因为已经登录到了 π 2.0 集群， π 集群上的文件路径不用加前缀，而思源一号上的文件路径需要加前缀 `[用户名]@sydata.hpc.sjtu.edu.cn`。

```
# 示例 6：该用户在 data 节点上将 lustre 个人目录下的数据 ~/data 搬运到 dssg 个人目录 ~
↪/data 下
$ ssh expuser01@data.hpc.sjtu.edu.cn
$ cp -r /lustre/home/acct-exp/expuser01/data/ /dssg/home/acct-exp/expuser01/
↪data/

# 示例 7：该用户在 sydata 节点上将 lustre 个人目录 ~/
↪data 下的数据搬运到 dssg 个人目录 ~/data 下
$ ssh expuser01@sydata.hpc.sjtu.edu.cn
$ scp -r expuser01@sydata.hpc.sjtu.edu.cn:/lustre/home/acct-exp/expuser01/
↪data/ /dssg/home/acct-exp/expuser01/data/
```

从 π 超算和思源一号向冷存储搬运数据的具体操作请参考：[科学数据平台（冷存储系统）数据使用指南](#)

4.7.4 科学数据平台（冷存储系统）数据使用指南

科学数据平台（冷存储系统）挂载在传输节点（`data` 节点、`sydata` 节点）以及计算节点，挂载路径为 `/union`。用户可以将不常用的数据转移到冷存储，减少存储费用。

如何使用冷存储

传输节点

π 集群、思源一号集群的传输节点 `data`、`sydata` 都挂载了冷存储，可以访问并修改冷存储数据。迁移数据请使用传输节点。在转移 π 超算 `/lustre` 下的数据时，需要选择 **data** 传输节点，在转移思源一号 `/dssg` 下的数据时，需要选择 **sydata** 传输节点。

登录节点、计算节点

为了方便用户使用冷存储中的数据，Pi、思源一号集群的登录节点、计算节点也挂载了冷存储。挂载方式为只读，不能写入数据。

传输数据到冷存储

迁移数据请使用传输节点。rsync 支持增量传输、断点续传、文件校验等功能，建议用 rsync 命令拷贝数据。

使用 **tmux** 会话

如果直接在命令行执行数据传输命令，网络不稳定可能导致传输中断，因此建议先开启 **tmux** 会话，再传输数据。相应的命令为：

```
# 开启会话：在终端输入下面命令，会开启名为 transport 的 tmux 会话
tmux new -s transport

# 分离会话：在会话中输入下面命令，会从会话返回终端
# 如果会话在执行程序中无法输入命令，先按 Ctrl+B，再按 D
tmux detach

# 查看所有的 tmux 会话
tmux ls

# 回到名为 transport 会话
tmux attach -t transport

# 彻底关闭 transport 会话
tmux kill-session -t transport
```

使用 **rsync** 从 **Pi** 超算或思源一号向冷存储传输数据

假设用户 **expuser01** 需要将 **Pi** 超算个人目录下的数据 `$HOME/data/` 搬运到冷存储下的个人目录 `$UNION/data/`，需要执行的命令为：

```
# -a 表示保存所有元数据，-v 打印更多信息，-r 表示包含子目录，-P
→表示显示进展以及保存未传输完的文件，其余可用参数见 rsync 文档
rsync -avrP $HOME/data/ $UNION/data/

# 如果要搬运的目录中包含软链接文件，需要用 --copy-unsafe-links
→参数，将不安全的软链接（原文件不在备份范围内）转换成实体文件保存
rsync -avrP --copy-unsafe-links $HOME/data/ $UNION/data/

# 如果要搬运的目录中包含硬链接文件，可以用 -H
→参数，保留文件之间的硬链接信息，减少不必要的文件传输
rsync -avrPH $HOME/data/ $UNION/data/
```

使用 **rsync** 从冷存储向 **Pi** 超算或思源一号传输数据

假设用户 **expuser01** 需要将冷存储下的个人目录 `$UNION/data/` 搬运到 **Pi** 超算个人目录下的数据 `$HOME/data/`，需要执行的命令为：

```
# -a 表示保存所有元数据, -v 打印更多信息, -r 表示包含子目录, -P
→表示显示进展以及保存未传输完的文件, 其余可用参数见 rsync 文档
rsync -avrP $UNION/data/ $HOME/data/

# 如果要搬运的目录中包含软链接文件, 需要用 --copy-unsafe-links
→参数, 将不安全的软链接(原文件不在备份范围内)转换成实体文件保存
rsync -avrP --copy-unsafe-links $UNION/data/ $HOME/data/

# 如果要搬运的目录中包含硬链接文件, 可以用 -H
→参数, 保留文件之间的硬链接信息, 减少不必要的文件传输
rsync -avrPH $UNION/data/ $HOME/data/
```

在 rsync 传输完成之后, 会出现类似下面的提示信息:

```
...
sent 5,244,160,230 bytes  received 137 bytes  48,333,275.27 bytes/sec
total size is 5,242,880,013  speedup is 1.00
```

如果因为某些原因, 导致 rsync 未正常结束, 可以再次用 rsync 命令传输数据, rsync 的断点续传功能可以接着上次传输进度继续传输。

数据校验

数据传输可能受网络波动影响, 建议在数据传输完成之后, 通过数据校验确认数据完整。

对于少量文件, 可以用 md5sum 校验。对于多级目录结构, 可以用 md5deep 工具。

md5deep 校验 (推荐)

md5deep 比 md5sum 命令更加丰富, 可以递归地检查整个目录树, 为子目录中的每个文件生成 md5 值。文件的数量和大小会影响 md5 值生成的速度, 如遇到这一步耗时较长, 请耐心等待。

假设用户 expuser01 需要为目录 \$HOME/data/ 和 \$UNION/data/ 下的子目录的每个文件生成 md5 值, 需要执行以下命令:

```
# 传输之前, 对子目录的每个文件生成 md5 值
md5deep -rl $HOME/data/ > before.md5deep

# 通过 rsync 传输数据
# ...

# 传输之后, 对目录的每个文件生成 md5 值
md5deep -rl $UNION/data/ > after.md5deep

# md5deep 文件的格式为 "md5值 文件路径", 比较传输前后的各文件 md5
→值需要使用第一列
sort before.md5deep | awk '{print $1}' > before
sort after.md5deep | awk '{print $1}' > after
diff before after
```

md5sum 校验

md5sum 可以生成文件校验码，来发现文件传输（网络传输、复制、本地不同设备间的传输）异常造成的文件内容不一致的情况。文件的数量和大小会影响 md5 值生成的速度，如遇到这一步耗时较长，请耐心等待。

```
# 传输之前，对 txt 文件生成 md5 校验码
ls *.txt | xargs -i -P 5 md5sum {} > file.md5

# 通过 rsync 传输
# ...

# 传输之后，生成 md5 校验码
# ...

# 传输完成后，比较传输前后 md5 校验码
diff file1.md5 file2.md5
```

清理存储空间

在完成数据传输、数据校验之后，可以清理 Pi 存储、思源存储上的原文件。

警告

在清理数据之前，请确认数据已经备份、以软硬链接保存的文件已经备份、rm 的路径正确。

- 如果要清理位于 Pi 超算或思源一号的数据，可以参考以下命令：

危险

下面的命令将直接删除对应路径下所有的内容，删除之后无法恢复数据，请确认路径正确后再执行！

```
# 假设原文件存储位置在 /lustre/home/acct-exp/expuser01/data/
rm -rf /lustre/home/acct-exp/expuser01/data/
```

- 如果要清理位于冷存储的数据，可以参考以下命令：

危险

下面的命令将直接删除对应路径下所有的内容，删除之后无法恢复数据，请确认路径正确后再执行！

```
# 假设原文件存储位置在 /union/home/acct-exp/expuser01/data/
rm -rf /union/home/acct-exp/expuser01/data/
```

4.7.5 数据传输技巧

提高数据传输速度的技巧 (从 **lustre** 目录到外部主机)

集群内部网络链路的带宽均不低于 10Gbps，可以支持 1GB/s 的并行传输速度。但请注意包括 **rsync**，**scp**，**winscp** 等工具在内，大部分传输方式都是基于 **ssh** 通信的，而单个 **ssh** 连接支持的最大传输速度约 100~150MB/s，但是可以并发多个 **scp/rsync** 进程分别传输不同的内容来进一步提高网络带宽利用效率。

scp，**rsync** 本身都不支持多进程传输，因此需要利用外部指令并发多个 **scp/rsync** 进程，外部封装的方法有很多，这里仅提供一种利用 **xargs** 自动分配传输文件的方法，熟悉脚本的用户也可以自制脚本来更灵活地将传输任务分配给各个传输进程。

```
# 示例：并发5个rsync进程从集群lustre目录~/data下载数据到外部主机~/download/
→路径下
$ ssh expuser01@data.hpc.sjtu.edu.cn ls /lustre/home/acct-exp/expuser01/
→data/ > remote_list.txt
$ cat remote_list.txt
001.dat
002.dat
003.dat
004.dat
005.dat
$ cat remote_list.txt | xargs --max-args=1 --max-procs=5 --replace=% rsync -
→-archive --partial expuser01@data.hpc.sjtu.edu.cn:/lustre/home/acct-exp/
→expuser01/data/% ~/download/
```

注意：如果没有事先配置好免密码登录，**rsync** 发起连接会要求用户输入密码，上述并发场合则会导致并发失败。请参考免密 [SSH 登录](#) 预先配置好密钥。建议在并发操作之前先用 **rsync** 单独拷贝一个小文件进行测试，确认过程中没有手动交互的需求再进行正式的并发传输。

并发数量请控制在 10 个进程以内，因为目前集群网络最高支持 1GB/s 的传输速度，而单个 **ssh** 进程上限是 100MB/s，10 个并发进程就已经足够占用全部带宽。

提高数据传输速度的技巧

利用 **rsync** 命令并发多个 **scp** 进程可有效提高数据传输的速度

```
# 示例：并发5个scp进程从 pi 2.0 集群lustre目录/lustre/home/acct-hpc/
→expuser01/img/传送文件数据至归档存储/union/home/acct-hpc/expuser01/sif/
→目录下
ssh expuser01@data.hpc.sjtu.edu.cn
find /lustre/home/acct-exp/expuser01/img -type f | xargs -P 5 -I {} scp {}
→expuser01@data.hpc.sjtu.edu.cn:/union/home/acct-exp/expuser01/sif/
```

上述命令启动 5 个 **scp** 进程并发传送 47GB 的文件数据，添加 **time** 命令可统计传送时间，

比如 `time scp -r /lustre/home/acct-exp/expuser01/img/* expuser01@data.hpc.sjtu.edu.cn:/union/home/acct-exp/expuser01/sif/`

使用 1、5 个进程从 **lustre** 目录传送 47GB 的数据至 **union** 目录所用时间如下所示

进程数	时间
1	10m17s
5	3m47s

4.7.6 通过远程挂载读写超算平台的数据

要读写超算平台的数据，除了通过登录到集群的登录节点和计算节点直接操作数据，还可以通过 `sshfs` 挂载家目录到 **Windows** 或 **Linux** 主机。下面介绍如何通过远程挂载的方式读写超算平台的数据。

sshfs 介绍

`sshfs` 是基于 `fuse` 构建的 `ssh` 文件系统客户端程序。无需对远程主机的配置作任何改变，就可以通过 `ssh` 协议来挂载远程文件系统。

挂载家目录到 **Linux** 主机

假设用户 `expuser01` 现在需要将 **Pi 2.0** 集群上自己的家目录 `/lustre/home/acct-exp/expuser01` 挂载到本地的 `/mountpoint1`，以及将思源集群上自己的家目录 `/dssg/home/acct-exp/expuser01` 挂载到本地的 `/mountpoint2`，可以按照下面步骤操作：

```
# 示例1：在 Linux 系统环境挂载 Pi 2.0 集群上的个人家目录（需要先安装 sshfs
→软件）
# sshfs -p 22 -o allow_other expuser01@data.hpc.sjtu.edu.cn:/lustre/home/
→acct-exp/expuser01/ /mountpoint1

# 示例2：在 Linux 系统环境挂载思源一号集群上的个人家目录（需要先安装 sshfs
→软件）
# sshfs -p 22 -o allow_other expuser01@sydata.hpc.sjtu.edu.cn:/dssg/home/
→acct-exp/expuser01/ /mountpoint2
```

挂载家目录到 **Windows** 主机

```
# 示例3：在 Windows 系统环境挂载 Pi 2.0 集群上的个人家目录（需要先安装
→WinFSP：https://github.com/billziss-gh/winfsp，再安装 sshfs-win：https://
→github.com/billziss-gh/sshfs-win）
# 在Windows的文件资源管理器中点击『映射网络驱动器』：添加 \\sshfs\
→expuser01@202.120.58.253 后点击『连接』

# 示例4：在 Windows 系统环境挂载思源一号集群上的个人家目录（需要先安装
→WinFSP：https://github.com/billziss-gh/winfsp，再安装 sshfs-win：https://
→github.com/billziss-gh/sshfs-win）
# 在Windows的文件资源管理器中点击『映射网络驱动器』：添加 \\sshfs\
→expuser01@111.186.43.4 后点击『连接』
```

4.7.7 常见问题

Q: 计算节点不能访问互联网/不能下载数据

A: 计算节点是通过 proxy 节点代理进行网络访问的，因此一些软件需要特定的代理设置。需要找到软件的配置文件，修改软件的代理设置。

a) git、wget、curl 等软件支持通用变量，代理参数设置为：

```
# 思源一号计算节点通用代理设置
https_proxy=http://proxy2.pi.sjtu.edu.cn:3128
http_proxy=http://proxy2.pi.sjtu.edu.cn:3128
no_proxy=puppet,proxy,172.16.0.133,pi.sjtu.edu.cn

# π2.0计算节点通用代理设置
http_proxy=http://proxy.pi.sjtu.edu.cn:3004/
https_proxy=http://proxy.pi.sjtu.edu.cn:3004/
no_proxy=puppet
```

b) Python、MATLAB、Rstudio、fasterq-dump 等软件需要查询软件官网确定配置参数：

```
### fasterq-dump文件，配置文件路径 ~/.ncbi/user-settings.mkfg

# 思源一号节点代理设置
/tools/prefetch/download_to_cache = "true"
/http/proxy/enabled = "true"
/http/proxy/path = "http://proxy2.pi.sjtu.edu.cn:3128"

# π2.0节点代理设置
/tools/prefetch/download_to_cache = "true"
/http/proxy/enabled = "true"
/http/proxy/path = "http://proxy.pi.sjtu.edu.cn:3004"

### Python需要在代码里面指定代理设置，不同Python包代理参数可能不同

# 思源一号节点代理设置
proxies = {
    'http': 'http://proxy2.pi.sjtu.edu.cn:3128',
    'https': 'http://proxy2.pi.sjtu.edu.cn:3128',
}

# π2.0节点代理设置
proxies = {
    'http': 'http://proxy.pi.sjtu.edu.cn:3004',
    'https': 'http://proxy.pi.sjtu.edu.cn:3004',
}

### MATLAB

# 思源一号节点代理设置
proxy2.pi.sjtu.edu.cn:3128
```

(续下页)

(接上页)

```
# π2.0节点代理设置
proxy.hpc.sjtu.edu.cn:3004
```

4.7.8 科学数据平台使用指南

上海交通大学科学数据平台首页地址为: <https://scidata.sjtu.edu.cn>, 秉承 FAIR 原则, 致力于打造科学数据全生命周期管理平台

用户登录

科学数据平台使用 jAccount 进行登录认证, 在首页找到“login”按钮后, 点击跳转至 jAccount 认证界面。使用账号密码认证通过后, 自动跳转至网站首页。首次登录前, 需要在“交我办”中, 找到“交我算” - “数据发布” - “科学数据平台账号激活申请”, 申请账号激活。

科学数据平台使用jAccount进行登录认证
在首页右侧找到“登录”按钮, 输入jAccount账号密码进行登录

①在“交我办”申请账号激活
(仅首次登录前)

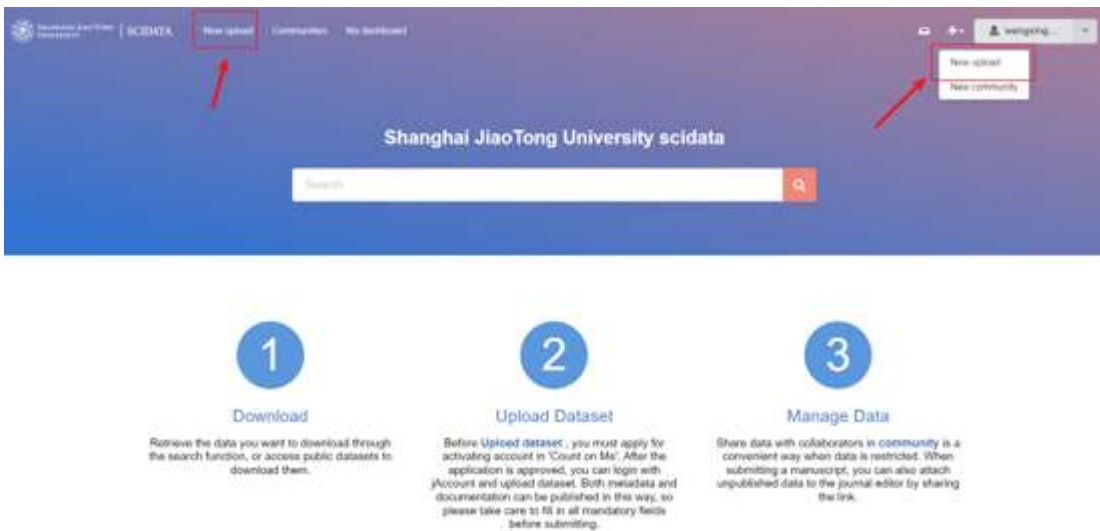
②点击右上角“登录”

③验证jAccount账号

发布数据

创建一条记录并添加数据文件

点击右上角“+”号, 在下拉选项中选择“upload”即可进入数据发布页面。数据文件可以通过浏览器拖拽或点选文件上传, 如果没有数据文件, 仅有元数据信息, 则勾选“Metadata-only record”。



发布超算集群上的文件

1. 登录可视化平台，选择 **desktop** 服务，填写需要的核心数及时长。

2. 在桌面端打开网页服务，登录 <https://scidata.sjtu.edu.cn>，找到发布数据的入口，正常提交数据即可。此时，可以通过按钮将超算账号下的文件直接选中、发布。

添加元数据信息

补充完善此条记录的元数据信息，尤其是必填信息。资金支持、备选标识符、相关工作等条目，如果没有可以点击“x”删除

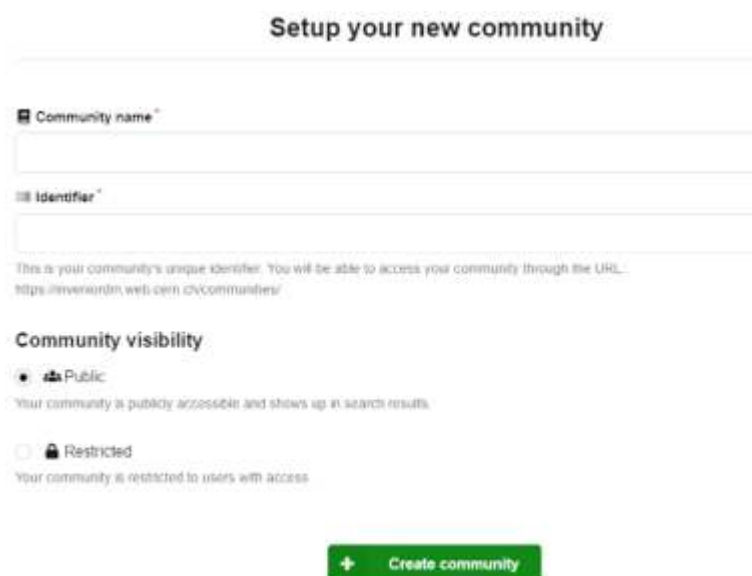
设定访问权限

权限设置可对元数据和数据文件单独控制，细分三类：完全公开、公开元数据、完全私有。注意：权限设置需要谨慎，完全公开的数据，其他人将能够看到、搜索、下载所有提交的数据文件；如果成果仍在投稿阶段，需要将数据设置有私有，可以通过专属链接提供至编辑访问。

	数据开放程度	数据发布者	合作者	匿名用户
Public	公开	√ 搜索元数据 √ 查看文件 √ 编辑元数据	√ 搜索元数据 √ 查看文件 × 编辑元数据	√ 搜索元数据 √ 查看文件 × 编辑元数据
Public with restricted files	公开元数据	√ 搜索元数据 √ 查看文件 √ 编辑元数据	√ 搜索元数据 √ 查看文件 (分享后) × 编辑元数据	√ 搜索元数据 × 查看文件 × 编辑元数据
Restricted	私有	√ 搜索元数据 √ 查看文件 √ 编辑元数据	× 搜索元数据 √ 查看文件 (分享后) × 编辑元数据	× 搜索元数据 × 查看文件 × 编辑元数据

建立团队空间

点击右上角“+”号，在下拉选项中选择“new community”，填写名称、标识符、简介等信息后，点击创建。



Setup your new community

Community name *

Identifier *

This is your community's unique identifier. You will be able to access your community through the URL: <https://invenio.cn/web/cmn/cvcommunities/>

Community visibility

☒ Public
Your community is publicly accessible and shows up in search results.

☐ Restricted
Your community is restricted to users with access.

+ Create community

分享数据

在数据详情页面，点击右侧的“Share”按钮，可以生成数据详情页的相关链接。合作者收到链接后，点击即可查看该数据的详情页面。这种数据分享方式，即使权限设置为非公开，链接接收者仍然能够访问。请注意保护数据安全

下载数据

包含文件的数据，在描述下方会展示所有相关联的文件，每个单独的文件名后方，具有一个方形的“download”按钮。可以直接点击按钮通过浏览器进行下载，或者右键按钮并复制链接，在命令行中使用下载工具下载。

检索数据

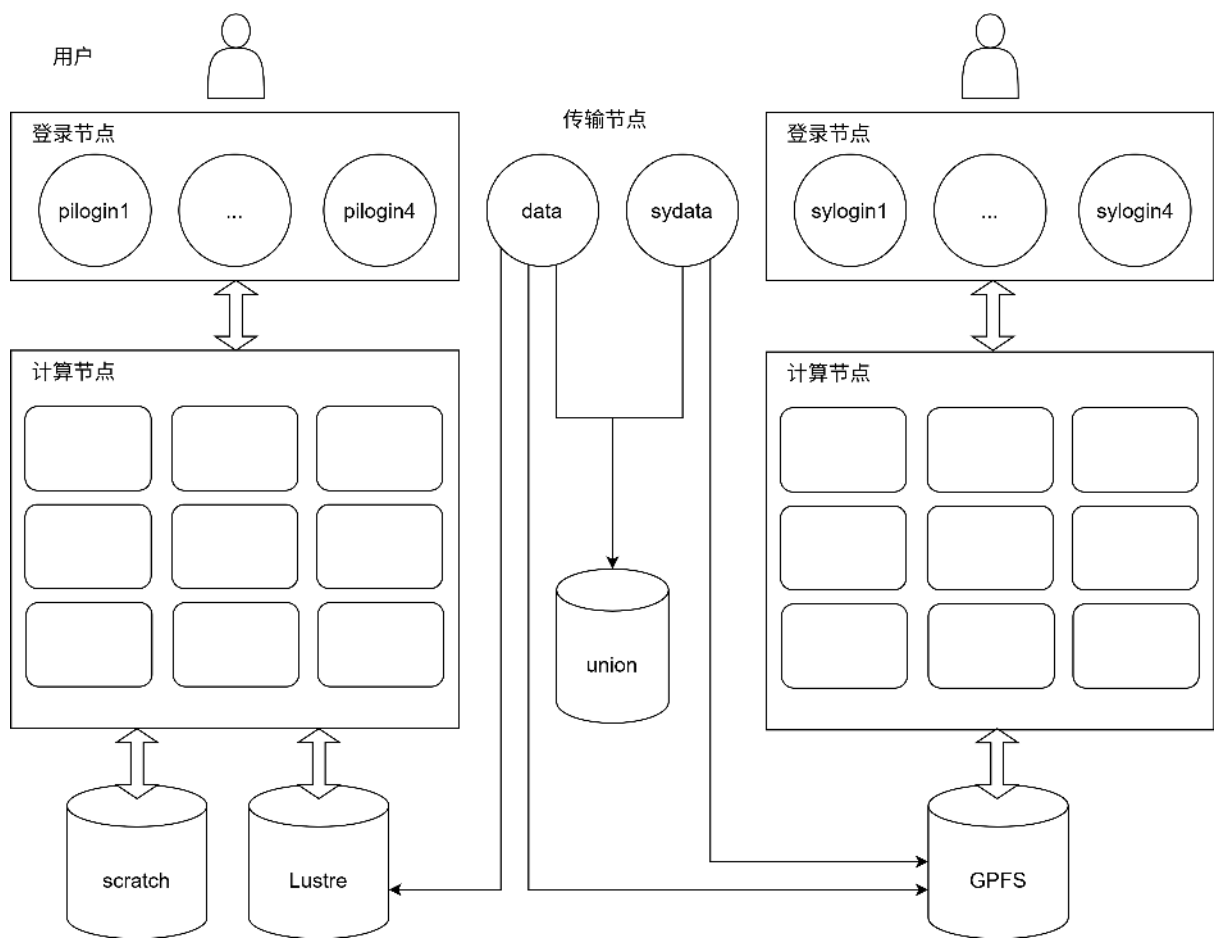
科学数据平台首页附有搜索框，通过输入关键词可以对相关数据进行搜索。在搜索结果页面左侧，支持对查找到数据的状态、类型进行筛选。

开源科学数据下载

为了方便“交我算”用户使用公共开源科学数据，尤其是国外的大型公共数据，平台提供了常用公共数据的镜像。用户可以在“community”中找到所需开源科学数据的类别，进入详情页面查看数据的使用方法。同样，也可以直接在搜索框中检索需要的数据库名称，查看使用方法。**【P.S.】** 开源科学数据扩充中，如果您有需要使用的大型开源科学数据，欢迎发邮件到 hpc@sjtu.edu.cn 咨询。

4.7.9 文件系统简介

下图为 Pi 2.0、思源一号集群目前的存储架构，展示了各节点可以访问到的存储系统。



总结

文件系统可以管理数据和访问数据，不同的文件系统具有不同的特点。下面的表格介绍了超算平台使用的文件系统的特征。

文件系统	挂载点	裸盘容量	存储介质	在哪些节点访问	是否收费	是否有快照功能
Lustre	/lustre	25 PB	HDD	Pi 2.0 集群的登录节点、计算节点; data 传输节点	3 TB 以下免费, 3 TB 以上收费	无
GPFS	/dssg	10 PB	HDD	思源一号集群的登录节点、计算节点; data、sydata 传输节点	3 TB 以下免费, 3 TB 以上收费	无
全闪存文件系统	/scratch	108 TB	NVMe SSD	Pi 2.0 集群的登录节点、计算节点; data 传输节点	不收费	无
union 冷存储统一挂载点	/union	archive、vault 合并挂载点	/	登录节点、计算节点、传输节点	收费	有
archive 冷存储系统	/archive	23.5 PB	HDD	data、sydata 传输节点	收费	有
vault 冷存储系统	/vault	17.28 PB	HDD	data、sydata 传输节点	收费	有

- **Lustre/GPFS 文件系统**: 提供了大容量、高可用、高性能的存储, 是 Pi2.0、思源一号集群的主要存储系统, 是用户通过登录节点后默认进入的文件系统。用户的主要工作和常用数据都位于该文件系统。用户通过 `cd $HOME` 可以跳转到 Lustre/GPFS 下自己的目录。
- **全闪存文件系统**: 使用全套 NVMe SSD 搭建的高性能存储可以更好地支持 I/O 密集作业。该系统容量较小, 同时未设置高可用, 会定期清理数据, 推荐将其作为临时工作目录使用。用户通过 `cd $SCRATCH` 可以跳转到全闪存文件系统下自己的目录。
- **union 冷存储统一挂载点**: 基于 `mergerfs` 实现的冷存储逻辑聚合挂载点, 将 `archive` 冷存储、`vault` 冷存储透明合并为单一视图。用户通过 `cd $UNION` 可以跳转到 `union` 下自己的目录, 建议使用此目录操作冷存储数据。
- **archive/vault 冷存储系统**: 作为 Lustre/GPFS 文件系统的补充, 提供大容量、高可用的存储, 只能在传输节点访问, 主要给用户存放不常用的数据。冷存储系统具有快照功能, 快照可以恢复用户误删除的数据。用户通过 `cd $ARCHIVE` 或 `cd $VAULT` 可以跳转到 `archive/vault` 冷存储下自己的目录。

注意: 超算集群存储的详细计费标准, 请发邮件至 `hpc` 邮箱咨询

备注

`mergerfs` 是一个基于 `FUSE` 的联合挂载文件系统工具, 可以聚合多个物理目录 (或文件系统) 为单一挂载点, 呈现为一个统一的命名空间。下面以用户 `expuser01` 为例, 说明 `mergerfs` 的目录聚合效果:

底层物理路径:

```
/archive/home/acct-exp/expuser01
```

```

├─ data1.tar.gz      ← 仅存在于 archive
└─ logs/             ← 仅存在于 archive

/vault/home/acct-exp/expuser01
├─ backup.zip        ← 仅存在于 vault
└─ docs/             ← 仅存在于 vault

/union/home/acct-exp/expuser01 ← mergerfs 挂载点 (合并 archive + vault)
├─ data1.tar.gz      → 实际来自 /archive/...
├─ backup.zip        → 实际来自 /vault/...
├─ logs/             → 实际来自 /archive/...
└─ docs/             → 实际来自 /vault/...

```

文件系统使用场景

因为存储介质、搭建方法不同，文件系统有各自适用的使用场景。

场景一：将数据从用户本地传输到 **Pi 2.0** 集群，再提交作业处理数据

1. 用户从本地终端，使用 `scp/rsync` 等命令向 **data** 传输节点发起数据传输
2. 用户登录 **Pi 2.0** 登录节点，此时可以访问到之前传输的数据，然后用户提交 **Slurm** 作业

可以参考：

- [数据传输方法](#)
- [通过远程挂载读写超算平台的数据](#)

场景二：使用全闪存文件系统作为临时目录

1. 在 **Slurm** 脚本中，自定义一个目录 `$SCRATCH/tmp_dir` 作为临时目录存放中间文件，加快计算过程
2. 计算完成后，将计算结果及时从 **Scratch** 转移到 **Lustre/GPFS** 系统

可以参考：

- [如何使用全闪存文件系统](#)

场景三：将 **Lustre** 文件系统中不常用的数据迁移到冷存储

1. 用户登录到 **data** 传输节点，通过 `rsync` 等命令向冷存储迁移数据
2. 通过数据校验，确认数据完整迁移到了冷存储
3. 清理原文件，释放存储空间

可以参考：

- [科学数据平台（冷存储系统）数据使用指南](#)

场景四：将数据集发布在科学数据平台

科学数据平台 致力打造科学数据全生命周期管理平台，具有发布数据、共享数据、下载公共数据功能的数据平台，您可以将公开数据集发布在平台上。

数据平台简明使用指南可以参考：

- 科学数据平台使用指南

4.8 提交作业

交我算 HPC+AI 平台通过 Slurm 调度系统运行作业。

4.8.1 交我算集群队列介绍

思源一号集群

思源一号集群设置以下队列，使用限制与说明如下

队列名	说明
64c51	允许单作业 CPU 核数为 1~60000，每核配比 8G 内存；单节点配置为 64 核，512G 内存
a100	允许单作业 GPU 卡数为 1~92，推荐每卡配比 CPU 为 16，每 CPU 配比 8G 内存；单节点配置为 64 核，512G 内存，4 块 40G 显存的 A100 卡

该集群另外设置了调试队列 debug64c512g 和 debuga100，仅用于短时间测试，请勿批量投递作业进行完整计算。debug64c512g 作业最多申请 2 节点，运行 60 分钟。debuga100 作业最多申请 1 节点，运行 20 分钟。

π 2.0 和 AI 集群

π 2.0 和 AI 集群设置以下队列，使用限制与说明如下

队列名	说明
cpu	允许单作业 CPU 核数为 1~24000，每核配比 4G 内存，节点可共享使用；单节点配置为 40 核，192G 内存
huge	允许单作业 CPU 核数为 1~80，每核配比 35G 内存，节点可共享使用；单节点配置为 80 核，3T 内存
192ct	允许单作业 CPU 核数为 1~192，每核配比 31G 内存，节点可共享使用；单节点配置为 192 核，6T 内存
dgx2	允许单作业 GPU 卡数为 1~128，推荐每卡配比 CPU 为 6，每 CPU 配比 15G 内存；单节点配置为 96 核，1.45T 内存，16 块 32G 显存的 V100 卡

π 2.0 集群也设置了 debug 队列用于短时间测试，作业最多申请 2 节点，最长运行时间为 20 分钟。

ARM 集群

ARM 集群设置以下队列，使用限制与说明如下

队列名	说明
arm128c256	允许单作业 CPU 核数为 1~12800，每核配比 2G 内存；单节点配置为 128 核，256G 内存

以上信息在各集群登录界面均有展示。

各队列默认运行时长

huge 和 192c6t 队列默认的作业运行最长时间为 2 天，其余队列默认的作业运行最长时间为 7 天。

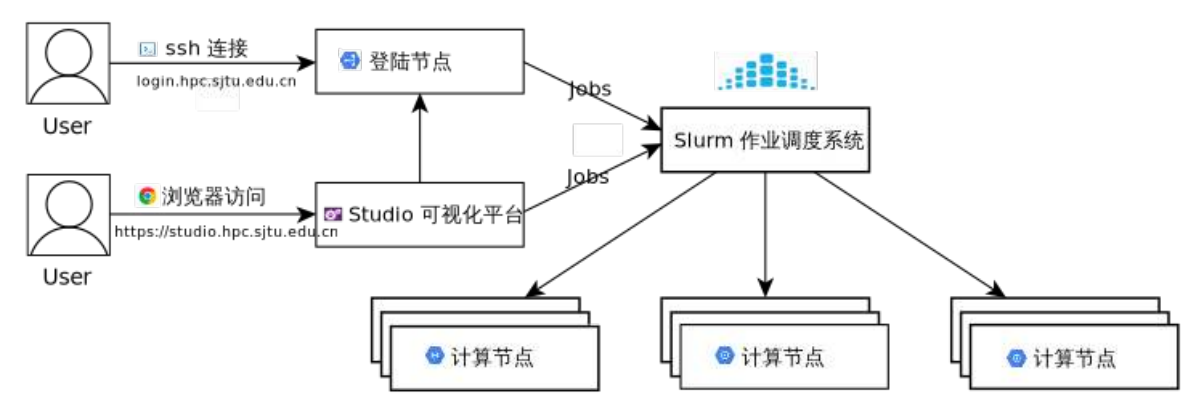
若预计超出 7 天，需提前 2 天发邮件告知用户名和 jobID 以便延长时限。延长后的作业最长运行时间不超过 14 天。

4.8.2 Slurm 作业调度系统

SLURM（Simple Linux Utility for Resource Management）是一种可扩展的工作负载管理器，已被全世界的国家超级计算机中心广泛采用。它是免费且开源的，根据GPL 通用公共许可证发行。

本文档将协助您通过 Slurm 管理作业。在这里可以找到更多的工作样本。

如果我们可以提供任何帮助，请随时联系 HPC 邮箱。



小技巧

由于跨系统文本编码的问题，我们强烈建议您只用英文字符和数字命名文件夹和目录，并且不要使用特殊字符，以确保作业能顺利运行。

Slurm 概览

Slurm	功能
sinfo	集群状态
squeue	排队作业状态
sbatch	作业提交
scontrol	查看和修改作业参数
sacct	已完成作业报告
scancel	删除作业

sinfo 查看集群状态

Slurm	功能
sinfo -N	查看节点级信息
sinfo -N --states=idle	查看可用节点信息
sinfo --partition=cpu	查看队列信息
sinfo --help	查看所有选项

节点状态包括：

drain(节点故障), alloc(节点在用), idle(节点可用), down(节点下线), mix(节点部分占用, 但仍有剩余资源)。

查看总体资源信息：

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
cpu        up       7-00:00:0  656   idle cas[001-656]
dgx2       up       7-00:00:0   8    idle vol[01-08]
```

squeue 查看作业信息

Slurm	功能
squeue -j jobid	查看作业信息
squeue -l	查看细节信息
squeue -n HOST	查看特定节点作业信息
squeue	查看 USER_LIST 的作业
squeue --state=R	查看特定状态的作业
squeue --help	查看所有的选项

作业状态包括 R(正在运行), PD(正在排队), CG(即将完成), CD(已完成)。

默认情况下, squeue 只会展示在排队或在运行的作业。

```
$ squeue
JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
18046      dgx2      ZX Ling   eenl  R      1:35:53      1 vol04
17796      dgx2      python   eexdl  R 3-00:22:04      1 vol02
```

显示您自己账户下的作业：

```
squeue
JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
17923      dgx2      bash      hpcwj  R 1-12:59:05      1 vol05
```

-l 选项可以显示更细节的信息。

```
squeue
JOBID PARTITION      NAME      USER      STATE      TIME  TIME_LIMI  NODES_
↳NODELIST(REASON)
17923      dgx2      bash      hpcwj  RUNNING 1-13:00:53 30-00:00:00      1 vol05
```

SBATCH 作业提交

准备作业脚本然后通过 `sbatch` 提交是 **Slurm** 的最常见用法。为了将作业脚本提交给作业系统，**Slurm** 使用

```
$ sbatch jobscript.slurm
```

Slurm 具有丰富的参数集。以下最常用的。

Slurm	含义
-n [count]	总进程数
--ntasks-per-node=[count]	每台节点上的进程数
-p [partition]	作业队列
--job-name=[name]	作业名
--output=[file_name]	标准输出文件
--error=[file_name]	标准错误文件
--time=[dd-hh:mm:ss]	作业最大运行时长
--exclusive	独占节点
--mail-type=[type]	通知类型，可选 all , fail , end ，分别对应全通知、故障通知、结束通知
--mail-user=[mail_addr]	通知邮箱
--odelist=[nodes]	偏好的作业节点
--exclude=[nodes]	避免的作业节点
--depend=[state:job_id]	作业依赖
--array=[array_spec]	序列作业

这是一个名为 `cpu.slurm` 的作业脚本，该脚本向 **cpu** 队列申请 1 个节点 40 核，并在作业完成时通知。在此作业中执行的命令是 `/bin/hostname`。

```
#!/bin/bash

#SBATCH --job-name=hostname
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err

/bin/hostname
```

用以下方式提交作业：

```
sbatch cpu.slurm
```

squeue 可用于检查作业状态。用户可以在作业执行期间通过 SSH 登录到计算节点。输出将实时更新到文件 [jobid] .out 和 [jobid] .err。

这里展示一个更复杂的作业要求，其中将启动 80 个进程，每台主机 40 个进程。

```
#!/bin/bash

#SBATCH --job-name=LINPACK
#SBATCH --partition=cpu
#SBATCH -n 80
#SBATCH --ntasks-per-node=40
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

以下作业请求 4 张 GPU 卡，其中 1 个 CPU 进程管理 1 张 GPU 卡。

```
#!/bin/bash

#SBATCH --job-name=GPU_HPL
#SBATCH --partition=dgx2
#SBATCH -n 4
#SBATCH --ntasks-per-node=4
#SBATCH --gres=gpu:4
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@MAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

以下作业启动一个 3 任务序列（从 0 到 2），每个任务需要 1 个 CPU 内核。关于集群上的 Python，您可以查阅我们的 Python 文档。

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=python_array
#SBATCH --mail-user=YOU@MAIL.COM
#SBATCH --mail-type=ALL
#SBATCH --ntasks=1
#SBATCH --time=00:30:00
#SBATCH --array=0-2
#SBATCH --output=python_array_%A_%a.out
#SBATCH --output=python_array_%A_%a.err

module load miniconda2/4.6.14-gcc-4.8.5

source activate YOUR_ENV_NAME

echo "SLURM_JOBID: " $SLURM_JOBID
echo "SLURM_ARRAY_TASK_ID: " $SLURM_ARRAY_TASK_ID
echo "SLURM_ARRAY_JOB_ID: " $SLURM_ARRAY_JOB_ID

python < vec_{$SLURM_ARRAY_TASK_ID}.py
```

srunk 和 salloc 交互式作业

srunk 可以启动交互式作业。该操作将阻塞，直到完成或终止。例如，在计算主机上运行 hostname。

```
$ srunk -N 1 -n 4 -p cpu hostname
cas006
```

启动远程主机 bash 终端：

```
srunk -p cpu -n 4 --pty /bin/bash
```

或者，可以通过 salloc 请求资源，然后在获取节点后登录到计算节点：

```
salloc -N 1 -n 4 -p cpu
ssh casxxx
```

scontrol: 查看和修改作业参数

Slurm	功能
scontrol show job JOB_ID	查看排队或正在运行的作业的信息
scontrol hold JOB_ID	暂停 JOB_ID
scontrol release JOB_ID	恢复 JOB_ID
scontrol update dependency=JOB_ID	添加作业依赖性，以便仅在 JOB_ID 完成后才开始作业

scontrol hold 命令可使排队中尚未运行的作业暂停被分配运行，被挂起的作业将不被执行。scontrol release 命令可取消挂起。

sacct 查看作业记录

Slurm	功能
sacct -l	查看详细的帐户作业信息
sacct --states=R	查看具有特定状态的作业的帐号作业信息
sacct -S YYYY-MM-DD	在指定时间后选择处于任意状态的作业
sacct --format= "LAYOUT"	使用给定的 LAYOUT 自定义 sacct 输出
sacct --help	查看所有选项

默认情况下，sacct 显示过去 **24** 小时的帐号作业信息。

```
$ sacct
```

查看更多信息：

```
$ sacct --format=jobid,jobname,account,partition,ntasks,alloccpus,elapsed,
↪state,exitcode -j 3224
```

查看平均作业内存消耗和最大内存消耗：

```
$ sacct --format="JobId,AveRSS,MaxRSS" -P -j xxx
```

Slurm 环境变量

Slurm	功能
\$_SLURM_JOB_ID	作业 ID
\$_SLURM_JOB_NAME	作业名
\$_SLURM_JOB_PARTITION	队列的名称
\$_SLURM_NTASKS	进程总数
\$_SLURM_NTASKS_PER_NODE	每个节点请求的任务数
\$_SLURM_JOB_NUM_NODES	节点数
\$_SLURM_JOB_NODELIST	节点列表
\$_SLURM_LOCALID	作业中流程的节点本地任务 ID
\$_SLURM_ARRAY_TASK_ID	作业序列中的任务 ID
\$_SLURM_SUBMIT_DIR	工作目录
\$_SLURM_SUBMIT_HOST	提交作业的主机名

参考教学视频

2022 春季用户培训之 slurm 调度系统

参考资料

- SLURM Workload Manager
- ACCRE' s SLURM Documentation
- Introduction to SLURM (NCCS lunchtime series)

4.8.3 作业示例（基本）

根据集群的不同队列、不同应用软件，示例 `slurm` 作业脚本。

作业提交流程

1. 编写作业脚本

```
vi test.slurm # 根据需求，选择计算资源：CPU 或 GPU、所需核数、是否需要大内存
```

2. 提交作业

```
sbatch test.slurm
```

3. 查看作业和资源

```
squeue # 查看正在排队或运行的作业
```

或

```
sacct # 查看过去 24 小时内已完成的作业
```

集群资源实时状态查询

```
sinfo # 若有 idle 或 mix 状态的节点，排队会比较快
```

各队列作业示例

cpu

cpu 队列 `slurm` 脚本示例：单节点不满核（例如 20 核），共享使用节点

```
#!/bin/bash

#SBATCH --job-name=test # 作业名
#SBATCH --partition=cpu # cpu 队列
#SBATCH -n 20 # 总核数 20
#SBATCH --ntasks-per-node=20 # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

cpu 队列 `slurm` 脚本示例：独占单节点不满核（20 核），比如为了独占整个节点的大内存


```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu          # cpu 队列
#SBATCH -n 20                    # 总核数 20
#SBATCH --ntasks-per-node=20     # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --exclusive              # 独占节点 (独占整个节点的大内存, 按照满核计费)
```

cpu 队列 slurm 脚本示例：单节点满核（40 核）

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu          # cpu 队列
#SBATCH -n 40                    # 总核数 40
#SBATCH --ntasks-per-node=40     # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

cpu 队列 slurm 脚本示例：多节点（160 核）

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu          # cpu 队列
#SBATCH -n 160                   # 总核数 160
#SBATCH --ntasks-per-node=40     # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

huge

huge 队列 slurm 脚本示例：单节点（20 核，最高可用 80 核）

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=huge         # huge 队列
#SBATCH -n 20                    # 总核数 20
#SBATCH --ntasks-per-node=20     # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

192c6t

192c6t 队列 slurm 脚本示例：单节点（96 核，最高可用 192 核）

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=192c6       # 192c6t 队列
#SBATCH -n 96                   # 总核数 96
#SBATCH --ntasks-per-node=96    # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

dgx2

dgx2 队列 slurm 脚本示例：单节点，分配 2 块 GPU，GPU:CPU 配比 1:6

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=dgx2        # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12      # 1:6 的 GPU:CPU 配比
#SBATCH --gres=gpu:2            # 2 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

arm128c256g

arm128c256g 队列 slurm 脚本示例：单节点 60 核

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=60
#SBATCH --output=%j.out
#SBATCH --error=%j.err

source /lustre/share/singularity/commercial-app/vasp/activate arm

mpirun -n $SLURM_NTASKS vasp_std
```

常用软件作业示例

下面根据不同应用软件，示例 **slurm** 作业脚本

LAMMPS 作业示例

cpu 队列 slurm 脚本示例 LAMMPS

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu         # cpu 队列
#SBATCH -n 80                   # 总核数 80
#SBATCH --ntasks-per-node=40    # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load lammmps

srun --mpi=pmi2 lmp -i YOUR_INPUT_FILE
```

GROMACS 作业示例

cpu 队列 slurm 脚本示例 GROMACS

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu         # cpu 队列
#SBATCH -n 80                   # 总核数 80
#SBATCH --ntasks-per-node=40    # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gromacs/2020-cpu

srun --mpi=pmi2 gmx_mpi mdrun -deffnm -s test.tpr -ntomp 1
```

Quantum ESPRESSO

cpu 队列 slurm 脚本示例 Quantum ESPRESSO

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu         # cpu 队列
#SBATCH -n 80                   # 总核数 80
#SBATCH --ntasks-per-node=40    # 每节点核数
#SBATCH --output=%j.out
```

(续下页)

(接上页)

```
#SBATCH --error=%j.err

module load quantum-espresso

srun --mpi=pmi2 pw.x -i test.in
```

OpenFOAM

cpu 队列 slurm 脚本示例 OpenFoam

```
#!/bin/bash

#SBATCH --job-name=test          # 作业名
#SBATCH --partition=cpu         # cpu 队列
#SBATCH -n 80                   # 总核数 80
#SBATCH --ntasks-per-node=40    # 每节点核数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load openfoam

srun --mpi=pmi2 icoFoam -parallel
```

TensorFlow

gpu 队列 slurm 脚本示例 TensorFlow

```
#!/bin/bash

#SBATCH -J test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --gres=gpu:2

module load miniconda3
source activate tf-env

python -c 'import tensorflow as tf; \
    print(tf.__version__); \
    print(tf.test.is_gpu_available());'
```

其它示例

Job Array 阵列作业

一批作业，若所需资源和内容相似，可借助 **Job Array** 批量提交。**Job Array** 中的每一个作业在调度时视为独立的作业。

cpu 队列 slurm 脚本示例 array

```
#!/bin/bash

#SBATCH --job-name=test           # 作业名
#SBATCH --partition=cpu          # cpu 队列
#SBATCH -n 1                     # 总核数 1
#SBATCH --ntasks-per-node=1      # 每节点核数
#SBATCH --output=array_%A_%a.out
#SBATCH --error=array_%A_%a.err
#SBATCH --array=1-20%10          # 总共 20 个子任务，每次最多同时运行 10 个

echo $SLURM_ARRAY_TASK_ID
```

作业状态邮件提醒

--mail-type= 指定状态发生时，发送邮件通知: ALL, BEGIN, END, FAIL

cpu 队列 slurm 脚本示例：邮件提醒

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -n 20
#SBATCH --ntasks-per-node=20
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --mail-type=end           # 作业结束时，邮件提醒
#SBATCH --mail-user=XX@sjtu.edu.cn
```

4.8.4 作业示例（开发者）

介绍不同并行环境的作业示例。

本文档中使用的作业样本可以在 `/lustre/share/samples` 中找到。在继续之前，请阅读有关预置软件环境的文档。

OpenMP 示例

以 OpenMP 为例，名为 `omp_hello.c` 代码如下：

```

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
    int nthreads, tid;

    /* Fork a team of threads giving them their own copies of variables */
    #pragma omp parallel private(nthreads, tid)
    {

        /* Obtain thread number */
        tid = omp_get_thread_num();
        printf("Hello World from thread = %d\n", tid);

        /* Only master thread does this */
        if (tid == 0)
        {
            nthreads = omp_get_num_threads();
            printf("Number of threads = %d\n", nthreads);
        }

        } /* All threads join master thread and disband */
    }
}

```

使用 **GCC 9.2.0** 编译

```

$ module load gcc
$ gcc -fopenmp omp_hello.c -o omphello

```

在本地运行 4 线程应用程序

```

$ export OMP_NUM_THREADS=4 && ./omphello

```

准备一个名为 **ompgcc.slurm** 的作业脚本

```

#!/bin/bash

#SBATCH --job-name=Hello_OpenMP
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 8
#SBATCH --ntasks-per-node=8

ulimit -l unlimited
ulimit -s unlimited

```

(续下页)

(接上页)

```
module load gcc

export OMP_NUM_THREADS=8
./omphello
```

提交到 SLURM

```
$ sbatch ompgcc.slurm
```

使用 Intel 编译器构建 OpenMP 应用

```
$ module load intel
$ icc -fopenmp omp_hello.c -o omphello
```

在本地运行 4 线程应用程序

```
$ export OMP_NUM_THREADS=4 && ./omphello
```

准备一个名为 ompicc.slurm 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=Hello_OpenMP
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 8
#SBATCH -ntasks-per-node=8
ulimit -l unlimited
ulimit -s unlimited

module load intel

export OMP_NUM_THREADS=8
./omphello
```

提交到 SLURM

```
$ sbatch ompicc.slurm
```

MPI 示例

以 mpihello.c 为例，代码如下：

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
```

(续下页)

(接上页)

```

#define MAX_HOSTNAME_LENGTH 256

int main(int argc, char *argv[])
{
    int pid;
    char hostname[MAX_HOSTNAME_LENGTH];

    int numprocs;
    int rank;

    int rc;

    /* Initialize MPI. Pass reference to the command line to
     * allow MPI to take any arguments it needs
     */
    rc = MPI_Init(&argc, &argv);

    /* It's always good to check the return values on MPI calls */
    if (rc != MPI_SUCCESS)
    {
        fprintf(stderr, "MPI_Init failed\n");
        return 1;
    }

    /* Get the number of processes and the rank of this process */
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    /* let's see who we are to the "outside world" - what host and what PID
    ↪ */
    gethostname(hostname, MAX_HOSTNAME_LENGTH);
    pid = getpid();

    /* say who we are */
    printf("Rank %d of %d has pid %5d on %s\n", rank, numprocs, pid, ↪
    ↪ hostname);
    fflush(stdout);

    /* allow MPI to clean up after itself */
    MPI_Finalize();
    return 0;
}

```

使用 **OpenMPI+GCC** 编译

```
$ module load gcc/8.3.0-gcc-4.8.5 openmpi/3.1.5-gcc-9.2.0
$ mpicc mpihello.c -o mpihello
```

准备一个名为 `job_openmpi.slurm` 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=mpihello
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 80
#SBATCH --ntasks-per-node=40

ulimit -s unlimited
ulimit -l unlimited

module load gcc/8.3.0-gcc-4.8.5 openmpi/3.1.5-gcc-9.2.0

srun --mpi=pmi2 ./mpihello
```

最后，将作业提交到 **SLURM**

```
$ sbatch job_openmpi.slurm
```

使用 **Intel** 编译器构建 **MPI** 应用

```
$ module load intel-parallel-studio/cluster.2019.5-intel-19.0.5
$ mpiicc mpihello.c -o mpihello
```

准备一个名为 `job_impi.slurm` 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=mpihello
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 80
#SBATCH --ntasks-per-node=40

ulimit -s unlimited
ulimit -l unlimited

module load intel-parallel-studio/cluster.2019.5-intel-19.0.5

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
```

(续下页)

(接上页)

```
export I_MPI_FABRICS=shm:ofi
srun ./mpihello
```

最后，将作业提交到 SLURM

```
$ sbatch -p cpu job_mpi.slurm
```

MPI+OpenMP 混合示例

以 hybridmpi.c 为例，代码如下：

```
#include <stdio.h>
#include <mpi.h>
#include <omp.h>

int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int iam = 0, np = 1;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    #pragma omp parallel default(shared) private(iam, np)
    {
        np = omp_get_num_threads();
        iam = omp_get_thread_num();
        printf("Hello from thread %d out of %d from process %d out of %d on %s\n",
            iam, np, rank, numprocs, processor_name);
    }

    MPI_Finalize();
}
```

使用 GCC 编译如下：

```
$ module load gcc/8.3.0-gcc-4.8.5 openmpi/3.1.5-gcc-9.2.0
$ mpicc -O3 -fopenmp hybridmpi.c -o hybridmpi
```

准备一个名为 hybridmpi.slurm 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=HybridMPI
```

(续下页)

(接上页)

```
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive
#SBATCH --time=00:01:00

ulimit -s unlimited
ulimit -l unlimited

module load gcc/8.3.0-gcc-4.8.5 openmpi/3.1.5-gcc-9.2.0

export OMP_NUM_THREADS=40
srun --mpi=pmi2 ./hybridmpi
```

使用 ICC 编译

```
$ module load intel-parallel-studio/cluster.2019.5-intel-19.0.5
$ mpiicc -O3 -fopenmp hybridmpi.c -o hybridmpi
```

准备一个名为 `hybridmpi.slurm` 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=HybridMPI
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive
#SBATCH --time=00:01:00

ulimit -s unlimited
ulimit -l unlimited

module load intel-parallel-studio/cluster.2019.5-intel-19.0.5

export I_MPI_DEBUG=5
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
export I_MPI_FABRICS=shm:ofi

export OMP_NUM_THREADS=40
srun ./hybridmpi
```

将作业提交到 **4** 个计算节点上

```
$ sbatch -N 4 hybridmpi.slurm
```

CUDA 示例

以 cublashello.cu 为例，代码如下：

```
//Example 2. Application Using C and CUBLAS: 0-based indexing
//-----
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"
#define M 6
#define N 5
#define IDX2C(i,j,ld) (((j)*(ld))+(i))

static __inline__ void modify (cublasHandle_t handle, float *m, int ldm,
↪int n, int p, int q, float alpha, float beta){
    cublasSscal (handle, n-p, &alpha, &m[IDX2C(p,q,ldm)], ldm);
    cublasSscal (handle, ldm-p, &beta, &m[IDX2C(p,q,ldm)], 1);
}

int main (void){
    cudaError_t cudaStat;
    cublasStatus_t stat;
    cublasHandle_t handle;
    int i, j;
    float* devPtrA;
    float* a = 0;
    a = (float *)malloc (M * N * sizeof (*a));
    if (!a) {
        printf ("host memory allocation failed");
        return EXIT_FAILURE;
    }
    for (j = 0; j < N; j++) {
        for (i = 0; i < M; i++) {
            a[IDX2C(i,j,M)] = (float) (i * M + j + 1);
        }
    }
    cudaStat = cudaMalloc ((void**)&devPtrA, M*N*sizeof(*a));
    if (cudaStat != cudaSuccess) {
        printf ("device memory allocation failed");
        return EXIT_FAILURE;
    }
    stat = cublasCreate(&handle);
    if (stat != CUBLAS_STATUS_SUCCESS) {
        printf ("CUBLAS initialization failed\n");
    }
}
```

(续下页)

(接上页)

```

    return EXIT_FAILURE;
}
stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("data download failed");
    cudaFree (devPtrA);
    cublasDestroy(handle);
    return EXIT_FAILURE;
}
modify (handle, devPtrA, M, N, 1, 2, 16.0f, 12.0f);
stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("data upload failed");
    cudaFree (devPtrA);
    cublasDestroy(handle);
    return EXIT_FAILURE;
}
cudaFree (devPtrA);
cublasDestroy(handle);
for (j = 0; j < N; j++) {
    for (i = 0; i < M; i++) {
        printf ("%7.0f", a[IDX2C(i,j,M)]);
    }
    printf ("\n");
}
free(a);
return EXIT_SUCCESS;
}

```

使用 CUDA 编译

```

$ module load gcc/8.3.0-gcc-4.8.5 cuda/10.1.243-gcc-8.3.0
$ nvcc cublashello.cu -o cublashello -lcublas

```

作业脚本 `cublashello.slurm` 如下:

```

#!/bin/bash

#SBATCH --job-name=cublas
#SBATCH --partition=dgx2
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1

ulimit -s unlimited

```

(续下页)

(接上页)

```
ulimit -l unlimited

module load gcc/8.3.0-gcc-4.8.5 cuda/10.1.243-gcc-8.3.0

./cublashello
```

将作业提交到 **SLURM** 上的 **dgx2** 分区:

```
$ sbatch cublashello.slurm
```

通过 **sbatch** 运行 **Intel LINPACK**

假如在多节点运行 **MPI** 作业，首先准备执行文件并输入数据:

```
$ cd ~/tmp
$ cp /lustre/usr/samples/LINPACK/64/xhpl_intel64 .
$ cp /lustre/usr/samples/LINPACK/64/HPL.dat .
```

然后，准备一个的作业脚本 **linpack.sh**。在此脚本中，我们请求 **cpu** 分区上的 64 个内核，每个节点 16 个内核。请注意，**MPI** 作业是通过 **srun**（不是 **mpirun**）启动的。

```
#!/bin/bash

#SBATCH --job-name=Intel_MPLINPACK
#SBATCH --partition=cpu
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 80
#SBATCH --ntasks-per-node=40

ulimit -s unlimited
ulimit -l unlimited

module load intel-parallel-studio/cluster.2019.5-intel-19.0.5

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
export I_MPI_FABRICS=shm:ofi
export I_MPI_DEBUG=100

srun ./xhpl_intel64
```

最后，将作业提交到 **SLURM**.

```
$ sbatch linpack.sh
Submitted batch job 358
```

我们可以附加到正在运行的任务，并观察其 **STDOUT** 和 **STDERR**:

```
$ sattach 358.0  
$ CTRL-C
```

我们可以查看作业输出文件：

```
$ tail -f /lustre/home/hpc-jianwen/tmp/358.out
```

停止工作：

```
$ scancel 358
```

4.8.5 GPU 节点使用文档

交我算 HPC+AI 平台拥有两种 GPU 节点：

- **a100 (思源一号)**：包含 23 个 GPU 计算节点，每个节点为 NVIDIA HGX A100 4-GPU，每块 A100 默认配置 16 个 CPU 核心。
- **DGX-2 (AI 计算平台)**：AI 平台的 *dgx2* 队列含 DGX-2 8 台，每台 DGX-2 配备 16 块 NVIDIA Tesla V100，96 CPU 核心，因此每块 V100 最多可申请 6 个 CPU 核心。通过搭载 NVIDIA NVSwitch 创新技术，GPU 间带宽高达 2.4 TB/s。

本文档将介绍两种 GPU 节点使用方法（作业提交模式、交互模式）及 GPU 利用率查看方法。

a100 节点

基于 a100 计算节点，平台提供两种 GPU 队列：用于正式计算的 a100 队列，用于调试的 debuga100 队列。

a100 队列

提交 a100 队列作业请使用 思源一号 登录节点。

这是一个 单机单卡 作业脚本，该脚本向 *a100* 队列申请 1 块 GPU（默认配置 16 个 CPU 核心），并在作业完成时邮件通知。此示例作业中执行的为 NVIDIA Sample 中的 `cudaTensorCoreGemm`。

```
#!/bin/bash  
  
#SBATCH --job-name=test  
#SBATCH --partition=a100  
#SBATCH --nodes=1  
#SBATCH --ntasks-per-node=1  
#SBATCH --cpus-per-task=16  
#SBATCH --gres=gpu:1          #若使用 2 块卡，就给 gres=gpu:2  
#SBATCH --mail-type=end  
#SBATCH --mail-user=YOU@EMAIL.COM  
#SBATCH --output=%j.out  
#SBATCH --error=%j.err
```

(续下页)

(接上页)

```
module load gcc cuda
./cudaTensorCoreGemm
```

或者也可以申请节点资源进行交互操作，使用如下命令申请一张 GPU，默认配置 16 CPU 核心：

```
$ srun -p a100 -N 1 -n 1 --gres=gpu:1 --cpus-per-task=16 --pty /bin/bash
$ module load cuda
```

小技巧

在登录节点 *srun* 执行交互作业时可能会断连导致作业中断，建议在 **HPC Studio** 申请 1 核心的远程桌面（cpu 节点即可），选择好时间，在计算节点来执行 *srun*。

这是一个多机多卡作业脚本，该脚本向 *a100* 队列申请 2 个节点共 8 块 GPU，并在作业完成时邮件通知。

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=a100
#SBATCH --nodes=2                #该参数为节点数量
#SBATCH --ntasks-per-node=4      #该参数为每节点进程数
#SBATCH --cpus-per-task=16       #该参数为每进程cpu核心数
#SBATCH --gres=gpu:4            #该参数为每节点GPU卡数
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi cuda

mpirun -np 8 ./jacobi
```

注意

如果需要调用多机多卡，上述作业脚本参数只需要修改节点数量即可，另外运行的程序必须支持跨节点操作，如果程序只支持单机调用 GPU，则不是申请几张卡就可以直接同时调用。

debuga100 队列

提交 debuga100 队列作业请使用 思源一号登录节点。

debuga100 队列是 调试用的队列，目前只提供 1 节点，因此只能投递单节点作业。调试节点将 4 块 a100 物理卡虚拟成 $4*7=28$ 块 gpu 卡，每卡拥有 5G 独立显存；节点 CPU 资源依然为 64 核，请在作业参数中合理指定 gpu 与 cpu 的配比，并加上参数：`--qos=debug`。

投放到此队列的作业 运行时间最长为 2 小时，超时后会被终止。每个用户最多同时运行 1 个作业。

调试作业脚本示例：

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=debuga100
#SBATCH --qos=debug
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=25
#SBATCH --gres=gpu:5          #最多28gpu卡
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc cuda

./cudaTensorCoreGemm
```

DGX-2 节点

基于 DGX-2 计算节点，平台提供 dgx2 计算队列用于正式计算。

dgx2 队列

提交 dgx2 队列作业请使用 π 2.0 集群登录节点。

这是一个 单机单卡作业脚本，该脚本向 dgx2 队列申请 1 块 GPU（默认配置 6 个 CPU 核心），并在作业完成时邮件通知。此示例作业中执行的为 NVIDIA Sample 中的 cudaTensorCoreGemm。

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1          #若使用 2 块卡，就给 gres=gpu:2
#SBATCH --mail-type=end
```

(续下页)

(接上页)

```
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc cuda

./cudaTensorCoreGemm
```

或者也可以申请节点资源进行交互操作，使用如下命令申请一卡 GPU，默认配置 6 CPU 核心：

```
$ srun -p dgx2 -N 1 -n 1 --gres=gpu:1 --cpus-per-task=6 --pty /bin/bash
$ module load cuda
```

GPU 利用率查看

GPU 利用率查看，需先登录正在使用的 GPU 计算节点，然后输入 *nvidia-smi* 查看以 a100 为例：

```
$ squeue          # 查看正在计算的 GPU 节点名字，如 gpu03
$ ssh gpu03       # 登录节点
$ nvidia-smi
```

参考资料

- [DGX-2 User Guide](#)
- [SLURM Workload Manager](#)

4.8.6 ARM 节点使用文档

ARM 节点基于 ARM 处理器构建，是国内首台基于 ARM 处理器的校级超算集群，共 100 个计算节点，与 π 2.0 集群实现共享登录、共享 Lustre 文件系统和共享 Slurm 作业调度系统，完美融入现有超算系统。

ARM 单节点配备 128 核 (2.6 GHz)、256 GB 内存 (16 通道 DDR4-2933)、240 GB 本地硬盘，节点间采用 IB 高速互联，挂载 Lustre 并行文件系统。

ARM 作业提交方式与 π 2.0 一致。在 Slurm 作业调度系统里，100 个 ARM 计算节点以 arm128c256g 队列名称统一调度。

ARM 使用须知

- ARM 与 π 2.0 的 X86 CPU 指令集不同，在 π 2.0 上使用的软件无法直接在新队列上运行，必须使用 ARM 平台上统一部署的应用，或在 ARM 计算节点上自行重新编译。
- 软件编译和作业提交，均需在 ARM 节点上，不能在 π 2.0 节点上。

ARM 节点登录

- ARM 配备单独的登录节点，SSH 登录命令如下：

```
ssh username@armlogin.hpc.sjtu.edu.cn
```

- 也可以从 π 2.0 上登录到 ARM 计算节点：

```
srun -p arm128c256g -n 4 --pty /bin/bash
```

ARM 应用支持

由于 CPU 架构不同，原 π 2.0 的应用软件都需要重新编译。我们已在 ARM 集群上部署了首批主流计算软件。后续将会推出更多的适用 ARM 集群运行的应用。

- 应用查看：(在 ARM 登录节点或计算节点) `module av` 命令；
- 应用加载：(在 ARM 计算节点) `module load` 命令；

ARM 作业示例

以下是一个名为 `arm.slurm` 的单节点作业脚本，该脚本申请 1 个 ARM 节点 (128 核)。

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load XXX

mpirun -n $SLURM_NTASKS ...
```

用以下方式提交作业（请注意，ARM 作业务必在 ARM 的登录节点或计算节点提交）：

```
$ sbatch arm.slurm
```

`squeue` 可用于检查作业状态。

有两点需注意：

- 并行命令采用 `mpirun`，暂不推荐 `srun`
- 暂不限制节点共享或独占（不区分是否是 `small` 类型的作业）

ARM 交互作业示例

`srun` 可以启动交互式作业。该操作将阻塞，直到完成或终止。启动远程主机 `bash` 终端的命令：

```
$ srun -p arm128c256g -n 4 --pty /bin/bash
```

ARM 集群软件编译

建议大家优先使用我们提供的软件，若需要自行编译软件，请登录到 **ARM** 的计算节点，按照软件文档操作。目前 **ARM HPC** 应用生态正在逐步完善中，建议选择软件的最新版本，并了解其对 **ARM** 的支持情况。

4.8.7 思源一号使用文档

杨元庆科学计算中心“思源一号”高性能计算集群总算力 6 PFLOPS（每秒六千万亿次浮点运算），是目前国内高校第一的超算集群。

CPU 采用双路 Intel Xeon ICX Platinum 8358 32 核，主频 2.6GHz，共 936 个计算节点；GPU 采用 NVIDIA HGX A100 4-GPU，

共 23 个计算节点。计算节点之间使用 Mellanox 100 Gbps Infiniband HDR 高速互联，并行存储的聚合存储能力达 10 PB。

思源一号为独立集群，使用 `dssg` 文件系统，采用 `SLURM` 作业调度，提交方式与 π 2.0 一致，CPU 和 GPU 队列名分别为 `64c512g` 和 `a100`。

思源一号使用须知

- 思源一号为独立集群，使用 `dssg` 文件系统，与 π 2.0 文件系统隔离但是数据互通，两个文件系统共享每课组 3TB 的免费存储配额

思源一号登录

- 思源一号配备单独的登录节点，SSH 登录命令如下：

```
$ ssh username@sylogin1.hpc.sjtu.edu.cn
```

思源一号应用支持

思源一号为独立集群，部署的软件和编译器版本与 π 2.0 不同

- 应用查看：(在思源一号登录节点或计算节点) `module av` 命令；
- 应用加载：(在思源一号计算节点) `module load` 命令；

思源一号脚本示例

思源一号 **slurm** 脚本示例：单节点不满核（例如 20 核），共享使用节点

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=20
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load XXX

mpirun -n $SLURM_NTASKS ...
```

思源一号 **slurm** 脚本示例：单节点不满核（例如 20 核），独占使用节点

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=20
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH --exclusive           # 独占节点

module load XXX

mpirun -n $SLURM_NTASKS ...
```

注意：如使用 `--exclusive` 即表示独占节点 64 核全部计算资源，无论程序实际运行核数，均按 64 核进行计费。

思源一号 **slurm** 脚本示例：单节点满核（64 核）

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load XXX

mpirun -n $SLURM_NTASKS ...
```

思源一号 **slurm** 脚本示例：多节点满核（例如 4 节点 256 核）

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 4
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load XXX

mpirun -n $SLURM_NTASKS ...
```

用以下方式提交作业（请注意，思源一号作业请在思源一号的登录节点或计算节点提交）：

```
$ sbatch siyuan.slurm
```

squeue 可用于检查作业状态。

思源一号交互作业示例

srunch 可以启动交互式作业。该操作将阻塞，直到完成或终止。启动远程主机 bash 终端的命令：

```
$ srun -p 64c512g -n 4 --pty /bin/bash
```

参考教学视频

思源一号专题培训之思源一号介绍 思源一号专题培训之使用指导

4.8.8 查看作业资源信息

当作业对 CPU 和内存的要求较高时，了解运行作业的 CPU 和内存的使用信息，能够保证作业顺利运行。

内存

内存分配策略

集群	存储分配策略
π2.0	单节点配置为 40 核，180G 内存；每核配比 4G 内存

可使用 seff jobid 命令查看单核所能使用的内存空间

```
[hpc@login2 data]$ seff 9709905
Job ID: 9709905
Cluster: sjtupi
User/Group: hpchgc/hpchgc
State: RUNNING
Nodes: 1
Cores per node: 40
CPU Utilized: 00:00:00
CPU Efficiency: 0.00% of 02:22:40 core-walltime
Job Wall-clock time: 00:03:34
Memory Utilized: 0.00 MB (estimated maximum)
Memory Efficiency: 0.00% of 160.00 GB (4.00 GB/core) // (4.00
↪GB/core)
WARNING: Efficiency statistics may be misleading for RUNNING jobs.
```

作业运行中的内存占用

当提交作业后，使用 `squeue` 命令查看作业使用的节点

```
[hpc@login2 test]$ squeue
      JOBID PARTITION      NAME      USER ST       TIME  NODES
↪NODELIST (REASON)
      9709875      cpu  40cores      hpc  R        0:02      1 cas478
```

然后进入相关节点

```
ssh cas478
```

可根据用户名查看作业占用的存储空间

```
ps -u$USER -o %cpu,rss,args
```

示例如下： `ps -uhpc -o %cpu,rss,args`

```
%CPU    RSS COMMAND
98.5 633512 pw.x -i ausurf.in
98.5 652828 pw.x -i ausurf.in
98.6 654312 pw.x -i ausurf.in
98.6 652196 pw.x -i ausurf.in
```

RSS 表示单核所占用的存储空间，单位为 **KB**，上述分析可得单核上运行作业占用的存储空间大约为 **650MB**，40 核的内存利用率大约为： $(0.65G \times 40) / 160G$ ：16%。

如果需要动态监测存储资源的使用，可进入计算节点后，输入 `top` 命令

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
428410	hpc	20	0	5989.9m	662.5m	168.6m	R	100.0	0.3	0:22.67	pw.x
428419	hpc	20	0	5987.0m	658.9m	163.7m	R	100.0	0.3	0:22.61	pw.x
428421	hpc	20	0	5984.6m	677.8m	180.1m	R	100.0	0.4	0:22.66	pw.x

(续下页)

(接上页)

428433	hpc	20	0	6002.8m	661.7m	165.3m	R	100.0	0.3	0:22.68	pw.x
428436	hpc	20	0	5986.0m	659.0m	165.4m	R	100.0	0.3	0:22.66	pw.x

上述数据中的 RES 列数据表示运行作业所占用的存储资源，单核大约占用 650m。

作业运行结束后内存利用分析情况

使用 seff jobid 命令

```
[hpc@login2 data]$ seff 9709905
Job ID: 9709905
Cluster: sjtupi
User/Group: hpchgc/hpchgc
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 40
CPU Utilized: 06:27:20
CPU Efficiency: 99.15% of 06:30:40 core-walltime
Job Wall-clock time: 00:09:46
Memory Utilized: 23.33 GB
Memory Efficiency: 14.58% of 160.00 GB
```

GPU INFO

下面介绍如何通过 JOB ID 获取 GPU 信息以及如何获取 CUDA_VISIBLE_DEVICES 变量并在程序中利用该变量

NVIDIA 显卡的 UUID (Universally Unique Identifier, 通用唯一标识符) 和 BUS ID (总线标识符) 是两个重要的标识符。

UUID (Universally Unique Identifier): UUID 是一个 128 位的唯一标识符，用于在系统中识别每个独立的 NVIDIA 显卡设备。每个显卡设备都有一个唯一的 UUID，可以通过调用相关命令或 API 来获取它。UUID 在不同系统和环境中是持久的，即使重新启动系统或重新插拔显卡，UUID 也不会改变。

BUS ID (总线标识符): BUS ID 是用于标识系统中不同物理或逻辑总线上的 NVIDIA 显卡设备的标识符。BUS ID 提供了关于显卡设备如何连接到系统总线的信息，如 PCI 总线等。BUS ID 通常采用 “domain:bus:device.function” 的形式表示，其中 domain 表示域，bus 表示总线编号，device 表示设备编号，function 表示设备的功能编号。BUS ID 主要用于管理和配置显卡设备，以及确定显卡在系统中的位置。

通过 JOB ID 获取 GPU 信息

下面介绍如何通过 jobid 查询 UUID 及 BUSID 信息：

脚本文件

将下面脚本保存至 `getGPUInfo` 的文本中

```
#!/bin/bash
JOBID=$1
GPUNUMS=`scontrol show jobs ${JOBID}|grep "gres:gpu" |awk -F ':' '{print $3}'`
GPUHOST=`scontrol show jobs ${JOBID}|grep "NodeList="|sed -n '2p'|awk -F '=' '{print $2}'`
echo JobID: ${JOBID}
echo NodeHost: ${GPUHOST}
echo GPUNums: ${GPUNUMS}
echo Information of allocate GPUs:
ssh ${GPUHOST} "echo UUID: && nvidia-smi -L"
ssh ${GPUHOST} "echo BUS_ID: && nvidia-smi -q|grep 'Bus Id'|sed -e 's/^.*: /' -e 's/ $//'"
```

执行脚本加 **JOB ID**

针对正在运行的作业，执行脚本加作业 ID 可获取正在运行作业的节点中的 GPU 信息

```
$ chmod +x getGPUInfo
$ ./getGPUInfo 27180318
JobID: 27180318
NodeHost: gpu09
GPUNums: 4
Information of allocate GPUs:
UUID:
GPU 0: NVIDIA A100-SXM4-40GB (UUID: GPU-5cd88acf-5391-8562-cd34-b543319224b4)
GPU 1: NVIDIA A100-SXM4-40GB (UUID: GPU-7bc1435d-37b5-d4b8-6ac1-df72927a54e0)
GPU 2: NVIDIA A100-SXM4-40GB (UUID: GPU-4dedf87e-d147-83cc-c5bd-ec16324afa15)
GPU 3: NVIDIA A100-SXM4-40GB (UUID: GPU-2bf8c199-1e4a-31cd-470b-4ba6329d9a60)
BUS_ID:
00000000:31:00.0
00000000:4B:00.0
00000000:CA:00.0
00000000:E3:00.0
```

获取 **CUDA_VISIBLE_DEVICES**

`CUDA_VISIBLE_DEVICES` 是一个环境变量，用于在使用 CUDA 编程时指定可见的 GPU 设备。它可以用来控制程序所使用的 GPU 设备的数量和顺序。

当用户申请有 GPU 卡的任务时，`slurm` 系统会根据用户申请的 GPU 数量来设置

`CUDA_VISIBLE_DEVICES` 环境变量，只有相应编号的 GPU 设备会对程序可见，其他 GPU 设备则不可使用。

srunk 交互式作业

在 `srunk` 申请交互式作业后，可在 `shell` 中直接输出 `$CUDA_VISIBLE_DEVICES` 变量

```
$ srunk -n 8 -p dgx2 --gres=gpu:2 --pty /bin/bash
srunk: job 27182411 queued and waiting for resources
srunk: job 27182411 has been allocated resources
$ echo $CUDA_VISIBLE_DEVICES
0,1
```

作业脚本

也可以在作业脚本最前面输出 `$CUDA_VISIBLE_DEVICES` 变量

```
#!/bin/bash
#SBATCH -J test
#SBATCH -n 8
#SBATCH --gres=gpu:2
#SBATCH -p dgx2

echo $CUDA_VISIBLE_DEVICES
...
```

案例测试

以下是一个简单的 `torch` 程序，展示了根据 `$CUDA_VISIBLE_DEVICES` 变量，设置程序使用的 GPU

```
$ cat pytorch_test.py
import torch
from torch import nn
from torch.optim import Adam
from torch.nn.parallel import DataParallel
import os

class DEMO_model(nn.Module):
    def __init__(self, in_size, out_size):
        super().__init__()
        self.fc = nn.Linear(in_size, out_size)

    def forward(self, inp):
        outp = self.fc(inp)
        print(inp.shape, outp.device)
        return outp

model = DEMO_model(10, 5).to('cuda')

os.system("echo CUDA_VISIBLE_DEVICES: $CUDA_VISIBLE_DEVICES")
device_ids = os.environ.get('CUDA_VISIBLE_DEVICES')
```

(续下页)

(接上页)

```
device_ids = device_ids.split(',')
device_ids = [int(number) for number in device_ids]
model = DataParallel(model, device_ids=device_ids)
adam = Adam(model.parameters())
# 进行训练
for i in range(1):
    x = torch.rand([128, 10])
    y = model(x)
    loss = torch.norm(y)
    loss.backward()
    adam.step()
```

执行程序，需要加载 torch 环境

```
$ srun -n 8 -p dgx2 --gres=gpu:2 -w vol08 --pty /bin/bash
$ module load miniconda3
$ source activate
(base) $ conda activate pytorch-env
(pytorch-env) $ python pytorch_test.py
CUDA_VISIBLE_DEVICES: 0,1
torch.Size([64, 10]) cuda:0
torch.Size([64, 10]) cuda:1
```

4.8.9 新 CPU 队列（KOS 系统）使用

原 CPU 队列变更

原 CPU 队列名称变更为 centos 队列，登录节点名称变更为 oldlogin，如果需要在原 CPU 队列（centos 系统）提交作业，请至 oldlogin 登录节点提交作业，并将脚本内指定队列名称更换为 centos。

centos 队列大约将于 12 月 12 日正式下线，请用户尽快适配自己的应用至新 CPU 队列（KOS 系统）。在新 CPU 队列（KOS 系统）上编译应用请参考下文的编译应用内容。

KOS 系统简介

Pi 集群原先采用的是 CentOS 7 操作系统，该系统版本将于 2024 年 6 月底停止维护，这将使得 Pi 集群超算系统面临安全隐患，为应对 CentOS 7 即将全面停服带来的安全风险，我们将 Pi 集群计算节点的操作系统从 CentOS 7 替换为浪潮信息的国产 KOS 系统。

KOS 系统是浪潮信息基于 Linux Kernel、OpenAnolis 等开源技术自主研发的一款服务器操作系统，支持 x86、ARM 等主流架构处理器，性能和稳定性居于行业领先地位，具备成熟的 CentOS 迁移和替换能力，可满足云计算、大数据、分布式存储、人工智能、边缘计算等应用场景需求。

下文将介绍如何在 Pi 集群上使用新 CPU 队列（KOS 系统）节点，已部署的应用列表，应用测试结果对比，应用的使用文档，以及如何自行编译应用。

如何使用新 CPU 队列 (KOS 系统)

登录登陆节点

- 使用 ssh 软件工具登录

```
host: pilogin.hpc.sjtu.edu.cn
```

申请 CPU 队列 (KOS 系统)

新 CPU 队列 (KOS 系统) 队列名称为: `cpu`

新 CPU 队列 (KOS 系统) 队列使用方法与原 CPU 队列基本一致

slurm 脚本

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1

hostname
```

srun 交互式作业

```
srun -n 1 -p cpu --pty /bin/bash
```

已部署应用列表

可以通过 `module avail` 查询新 CPU 队列 (KOS 系统) 上已部署的编译器, 应用等新 CPU 队列 (KOS 系统) `modulefile` 路径:

```
/lustre/share/spack/modules/cascadelake/linux-rhel8-skylake_avx512
```

```
----- /lustre/share/spack/modules/cascadelake/
->linux-rhel8-skylake_avx512 -----
boost/1.82.0-gcc-8.5.0          intel-oneapi-compilers/2021.4.0  ↵
->          netcdf-fortran/4.6.0-gcc-8.5.0          (D)
bowtie/1.3.1-gcc-8.5.0          intel-oneapi-mkl/2021.4.0        ↵
->          openblas/0.3.23-gcc-8.5.0
bwa/0.7.17-gcc-8.5.0            intel-oneapi-mkl/2023.1.0        ↵
-> (D)          openfoam/2206-gcc-8.5.0              (D)
cgal/4.13-gcc-8.5.0             intel-oneapi-mpi/2021.4.0        ↵
->          openmpi/4.1.5-gcc-8.5.0
cmake/3.26.3-gcc-8.5.0          jasper/2.0.32-gcc-8.5.0         ↵
->          perl/5.36.0-gcc-8.5.0
```

(续下页)

(接上页)

cp2k/2023.2-intel-oneapi-2021.4.0	jasper/2.0.32-intel-2021.4.0	↪
↪ perl/5.36.0-intel-2021.4.0	(D)	
eigen/3.4.0-gcc-8.5.0	jasper/3.0.3-gcc-8.5.0	↪
↪ (D) python/3.10.10-gcc-8.5.0		
fftw/3.3.8-intel-2021.4.0	lammps/20220623.3-gcc-8.5.0	↪
↪ quantum-espresso/6.7-gcc-8.5.0-openblas		
fftw/3.3.10-gcc-8.5.0	(D) lammps/20220623.3-intel-2021.4.0	↪
↪ quantum-espresso/6.7-gcc-8.5.0	(D)	
git/2.40.0-gcc-8.5.0	llvm/12.0.1-gcc-8.5.0	↪
↪ samtools/1.16.1-gcc-8.5.0		
gromacs/2022.5-gcc-8.5.0	(D) miniconda3/22.11.1	↪
↪ stream/5.10-gcc-8.5.0		
hdf5/1.10.6-gcc-8.5.0	mpich/4.1.1-gcc-8.5.0	↪
↪ wps/4.4-intel-2021.4.0		
hdf5/1.10.6-intel-2021.4.0	netcdf-c/4.9.2-gcc-8.5.0	↪
↪ wrf/4.4.2-gcc-8.5.0		
hdf5/1.14.1-2-gcc-8.5.0	(D) netcdf-c/4.9.2-intel-2021.4.0	↪
↪ (D) wrf/4.4.2-intel-2021.4.0	(D)	
hpl/2.3-gcc-8.5.0	netcdf-fortran/4.5.2-gcc-8.5.0	
hypre/2.28.0-gcc-8.5.0	netcdf-fortran/4.5.2-intel-2021.4.0	

测试结果对比

根据多个常用的科学应用测试结果来看，在计算速度上 KOS 系统与 CentOS7 系统基本保持一致。其中 **hpl** 计算单位为 **Gflops**，数值越大计算速度越快，**gromacs** 计算单位为 **ns/day**，数值越大计算速度越快，其余应用计算单位为计算时间 (h:m:s)，数值越小计算速度越快。

app	KOS	CentOS
hpl	3651.1 Gflops	3774.26 Gflops
gromacs	18.558 ns/day	17.266 ns/day
lammps	00:03:13	00:03:16
cp2k	00:00:46	00:00:45
relion	00:28:45	00:25:00
qe	00:10:08	00:09:21
openfoam	00:03:16	00:03:28
wrf	01:15:24	01:10:28
bwa	00:31:49	00:33:09

应用测试案例

下面介绍上述应用使用的计算算例，应用使用文档可点击应用名称查看原 **Pi2.0** 集群的 **CPU** 队列使用文档。注意需要将作业脚本中的队列名称改为 **kos**，应用的调用模块需要改为 **kos** 队列中的模块名。

hpl

HPL.dat

```

HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
7            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
176640 Ns
1            # of NBs
256 NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
8 Ps
10 Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U   in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)

```

gromacs

gromacs 选择的测试算例为 gromacs 提供的 benchmark 水分子算例，本文选取的为 0768 水分子算例。

获取算例：

```

wget -c https://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz
tar xf water_GMX50_bare.tar.gz
cd water-cut1.0_GMX50_bare/0768/
tree 0768/
0768/
├── conf.gro
└── pme.mdp

```

(续下页)

(接上页)

```
└─ rf.mdp
└─ topol.top
```

lammps

lammps 选择的测试算例为 lammps 官方 benchmark 算例:in.lj

输入文件内容

```
# 3d Lennard-Jones melt

variable      x index 4
variable      y index 4
variable      z index 4

variable      xx equal 20*$x
variable      yy equal 20*$y
variable      zz equal 20*$z

units          lj
atom_style     atomic

lattice         fcc 0.8442
region         box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box     1 box
create_atoms   1 box
mass           1 1.0

velocity       all create 1.44 87287 loop geom

pair_style     lj/cut 2.5
pair_coeff     1 1 1.0 1.0 2.5

neighbor       0.3 bin
neigh_modify   delay 0 every 20 check no

fix           1 all nve
```

cp2k

cp2k 选择的测试算例为官方 benchmark 中的 H2O-64.inp 算例

算例获取:

```
cp -rfv /lustre/opt/cascadelake/linux-rhel8-skylake_avx512/intel-2021.4.0/
↪cp2k/cp2k/benchmarks/QS/H2O-64.inp .
```


quantum-espresso

quantum-espresso 选择的测试算例为官方提供的 test—cases 中的 small 算例: ausurf.in
算例获取:

```
wget https://repository.prace-ri.eu/git/UEABS/ueabs/-/raw/master/quantum_
↪espresso/test_cases/small/ausurf.in
wget https://repository.prace-ri.eu/git/UEABS/ueabs/-/raw/master/quantum_
↪espresso/test_cases/small/Au.pbe-nd-van.UPF
```

openfoam

openfoam 测试算例选择的是 simpleFoam 求解器计算摩托车外流场 motorBike 算例
算例获取:

```
module load openfoam/2206-gcc-8.5.0
mkdir openfoamTest1
cd openfoamTest1
cp -rv $FOAM_TUTORIALS ./
cd ./tutorials/incompressible//simpleFoam/motorBike
```

wrf

wrf 测试算例选择的是模拟 2016 年 10 月 06 日 00 点至 2016 年 10 月 08 日 0 点的气象数据

算例获取:

```
/lustre/opt/contribute/cascadelake/wrf_cmaq/wrf_data

tree wrf_data/
wrf_data/
├── fnl_20161006_00_00.grib2
├── fnl_20161006_06_00.grib2
├── fnl_20161006_12_00.grib2
├── fnl_20161006_18_00.grib2
├── fnl_20161007_00_00.grib2
├── fnl_20161007_06_00.grib2
├── fnl_20161007_12_00.grib2
├── fnl_20161007_18_00.grib2
└── fnl_20161008_00_00.grib2

geog_data_path:
/lustre/opt/contribute/cascadelake/wrf_cmaq/geo
```

bwa

bwa 选择的测试算例是 B17NC_R1.fastq

算例获取:

```
mkdir ~/bwa && cd ~/bwa
cp -r /lustre/share/sample/bwa/* ./
gzip -d B17NC_R1.fastq.gz
gzip -d B17NC_R2.fastq.gz
```

relion

relion 选择的测试算例为人类去铁铁蛋白 (apoferritin) 电镜图像数据集, 总计 32933 张图像, 数据量 8.1 GB

算例获取:

```
mkdir relion-test
cd relion-test
cp -rfv /lustre/share/samples/kos-samples/relion/apo-ferritin .
```

编译应用

因为系统版本升级, 用户原有编译软件需要重新编译才能使用, 编译方式和原先系统流程基本一致, 根据需要使用的编译器调用对应模块即可, 下面以 **fftw** 为例, 介绍如何在新 CPU 队列 (KOS 系统) 上使用 **gcc** 和 **intel** 两种编译器编译软件。

先申请计算节点用于编译

```
srunk -n 1 -p cpu --pty /bin/bash
```

gcc+openmpi

```
module load openmpi/4.1.5-gcc-8.5.0

wget https://fftw.org/pub/fftw/fftw-3.3.8.tar.gz
tar -xvf fftw-3.3.8.tar.gz
cd fftw-3.3.8/
./configure --prefix=$PWD --enable-mpi --enable-openmp --enable-threads --
--enable-shared MPICC=mpicc CC=gcc F77=gfortran
make
make install
```

intel-oneapi

```
module load intel-oneapi-compilers/2021.4.0
module load intel-oneapi-mpi/2021.4.0

wget https://fftw.org/pub/fftw/fftw-3.3.8.tar.gz
tar -xvf fftw-3.3.8.tar.gz
cd fftw-3.3.8/
./configure --prefix=$PWD --enable-mpi --enable-openmp --enable-threads --
--enable-shared MPICC=mpiicc CC=icc F77=ifort
make
make install
```

4.8.10 新 64c512g 队列（Rocky 系统）使用

Rocky 操作系统

CentOS 8 操作系统已于 2021 年 12 月停止维护，为应对安全风险，计划将思源一号集群的操作系统从 CentOS 8 替换到生命周期健康的 Rocky Linux。

目前提供了部署 Rocky Linux 操作系统的 64c512g CPU 节点队列，大部分主流应用已适配。如在使用队列时遇到问题，请发送邮件到 [hpc](#) 邮箱。

如何使用

登录

- 使用 ssh 软件工具登录到

```
host: sylogin.hpc.sjtu.edu.cn
```

在 64c512g 队列申请 CPU 资源

部署 Rocky Linux 操作系统的 CPU 节点队列为 64c512g，使用方法和原 64c512g 队列基本一致。

slurm 脚本

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH --nodes 1
#SBATCH --ntasks-per-node=1

hostname
```

srun 交互式作业

```
srun -p 64c512g -n 1 --pty /bin/bash
```

4.9 容器

容器是一种 Linux 上广为采用的应用封装技术，它将可执行程序与依赖库打包成一个镜像文件，启动时与宿主节点共享操作系统内核。由于镜像文件同时携带可执行文件和依赖库，避免了两者的不匹配造成的兼容性问题，还能在一个宿主 Linux 操作系统上支持多种不同的 Linux 发行版，譬如在 CentOS 发行版上运行 Ubuntu 的 `apt-get` 命令。

π 超算集群采用基于 Singularity 的高性能计算容器技术，相比 Docker 等在云计算环境中使用的容器技术，Singularity 同时支持 root 用户和非 root 用户启动，且容器启动前后，用户上下文保持不变，这使得用户权限在容器内部和外部都是相同的。此外，Singularity 强调容器服务的便捷性、可移植性和可扩展性，而弱化了容器进程的高度隔离性，因此量级更轻，内核 namespace 更少，性能损失更小。

您可以在专门的容器构建节点定制 Singularity 镜像。

4.9.1 通过交互式 Shell 构建 Singularity 镜像

小技巧

构建 Singularity 容器镜像通常需要 root 特权，通常超算集群不支持这样的操作。 π 超算集群的“容器化的 Singularity”允许用户编写、构建和传回自定义容器镜像。

在 π 超算集群上，我们采用“容器化的 Singularity”，允许用户在一个受限的环境内以普通用户身份“模拟”root 特权，保存成 Singularity 镜像，并将镜像传回集群使用。

首先从登录节点使用用户名 `build` 跳转到专门用于构建容器镜像的节点。需要注意的是，X86 节点 (用于 `cpu small huge` 等队列) 和国产 ARM 节点 (用于 `arm128c256g` 队列) 的处理器指令集是不兼容的，需使用对应的镜像构建节点。

小技巧

请选择与目标主机 (x86 或 arm) 相匹配的容器构建节点。

从登录节点跳转 X86 容器构建节点：

```
$ ssh build@container-x86
$ hostname
container-x86.pi.sjtu.edu.cn
```

从登录节点跳转 ARM 容器构建节点：

```
$ ssh build@container-arm
$ hostname
container-arm.pi.sjtu.edu.cn
```

小心

出于安全考虑，container-x86 和 container-arm 节点每天 **23:59** 重启节点并清空数据，请及时转移容器构建节点上的数据。build 为共享用户，请勿修改自己工作目录外的数据，以免影响其他用户的使用。

由于所有用户共享使用 build 用户，需要创建专属工作目录，在工作目录中构建镜像。我们使用 `mktemp -d` 命令在 `/tmp` 目录下创建名字带有随机字符的工作目录。

```
$ cd $(mktemp -d)
$ pwd
/tmp/tmp.sr7C5813M9
```

使用 `docker images` 查看本地可用的基础镜像列表。

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
centos	8	5d0da3dc9764	4 weeks ago	231MB

使用 `docker run -it IMAGE_ID` 从基础镜像创建容器 (container) 实例，并以 root 身份进入容器内。

```
$ docker run -it --name=MY_USERNAME_DATE 5d0da3dc9764 /bin/bash
```

在容器内需要配置代理才能访问网络。

```
$ export http_proxy=http://proxy.pi.sjtu.edu.cn:3004/
$ export https_proxy=http://proxy.pi.sjtu.edu.cn:3004/
$ export no_proxy=puppet,proxy
```

因为 centos 停止维护，初次进入镜像需要修改 yum 源，才可以正常使用 yum 命令。

```
$ sed -i 's/mirrorlist/#mirrorlist/g' /etc/yum.repos.d/CentOS*.repo
$ sed -i 's|#baseurl=http://mirror.centos.org|baseurl=http://vault.centos.
→org|g' /etc/yum.repos.d/CentOS*.repo
$ yum makecache
```

然后以 root 特权修改容器内容，例如安装软件等。

```
[root@68bdb5af0da9 /]# whoami
root
[root@68bdb5af0da9 /]# yum check-update
...
[root@68bdb5af0da9 /]# yum install tree
...
[root@68bdb5af0da9 /]# tree --version
tree v1.7.0 (c) 1996 - 2014 by Steve Baker, Thomas Moore, Francesc Rocher,
→Florian Sesser, Kyosuke Tokoro
```

操作结束后退出容器，回到 build 用户身份下。

```
[root@68bdb5af0da9 /]# exit
[build@container-x86 ~]$ whoami
build
```

使用 `docker ps -a` 查看与先前定义名字对应的 **container ID**，在这个示例中是 `MY_USERNAME_DATE`。

```
[build@container-x86 ~]$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
515e913f12cb	5d0da3dc9764	"/bin/bash"	4 seconds ago	Exited (0) 2 seconds ago

使用 `docker commit CONTAINER_ID IMG_NAME` 提交容器变更。

```
$ docker commit 515e913f12cb my_username_app_img
```

此时使用 `docker images` 可以在容器镜像列表中看到刚刚提交的容器变更。

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my_username_app_img	latest	c26c43a0cc9b	About a minute ago	279MB

将 **Docker** 容器保存为可在超算平台上使用的 **Singularity** 镜像。

```
$ SINGULARITY_NOHTTPS=1 singularity build my_username_app_img.sif docker-
daemon://my_username_app_img:latest
INFO: Starting build...
INFO: Creating SIF file...
INFO: Build complete: my_username_app_img.sif
```

使用 `scp my_username_app_img.sif YOUR_USERNAME@pilogin1:~/` 将 **Singularity** 镜像文件复制到超算集群家目录后，可以使用 `singularity` 命令测试镜像文件，从 `/etc/redhat-release` 内容和 `tree` 命令版本看，确实进入了与宿主操作系统不一样的运行环境。

```
$ singularity exec my_username_app_img.sif cat /etc/redhat-release
CentOS Linux release 8.4.2105
$ singularity exec my_username_app_img.sif tree --version
tree v1.7.0 (c) 1996 - 2014 by Steve Baker, Thomas Moore, Francesc Rocher,
Florian Sesser, Kyosuke Tokoro
```

4.9.2 AI 平台容器编译

与 **x86** 平台容器编译方式类似，在 **AI** 平台也可以按需定制 **Singularity** 镜像。

通过交互式 **Shell** 构建 **AI** 应用镜像**i** 小技巧

构建 **Singularity** 容器镜像通常需要 **root** 特权，通常超算集群不支持这样的操作。 π 超算集群的“容器化的 **Singularity**”允许用户编写、构建和传回自定义容器镜像。

在 π 超算集群上，我们采用“容器化的 **Singularity**”，允许用户在一个受限的环境内以普通用户身份“模拟”**root** 特权，保存成 **Singularity** 镜像，并将镜像传回集群使用。

从登录节点跳转 **X86** 容器构建节点：

```
$ ssh build@container-x86
$ hostname
container-x86.pi.sjtu.edu.cn
```

i 小心

出于安全考虑，**container-x86** 和 **container-arm** 节点每天 **23:59** 重启节点并清空数据，请及时转移容器构建节点上的数据。**build** 为共享用户，请勿修改自己工作目录外的数据，以免影响其他用户的使用。

由于所有用户共享使用 **build** 用户，需要创建专属工作目录，在工作目录中构建镜像。我们使用 **mktemp -d** 命令在 **/tmp** 目录下创建名字带有随机字符的工作目录。

```
$ cd $(mktemp -d)
$ pwd
/tmp/tmp.sr7C5813M9
```

使用 **docker images** 查看本地可用的基础镜像列表。

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
centos        8         5d0da3dc9764   4 weeks ago   231MB
```

使用 **docker run -it IMAGE_ID** 从基础镜像创建容器 (**container**) 实例，并以 **root** 身份进入容器内。

```
$ docker run -it --name=MY_USERNAME_DATE 5d0da3dc9764 /bin/bash
```

然后以 **root** 特权修改容器内容，例如安装软件等。

```
[root@68bdb5af0da9 /]# whoami
root
[root@68bdb5af0da9 /]# yum check-update
...
[root@68bdb5af0da9 /]# yum install tree
...
```

(续下页)

(接上页)

```
[root@68bdb5af0da9 /]# tree --version
tree v1.7.0 (c) 1996 - 2014 by Steve Baker, Thomas Moore, Francesc Rocher,
↳ Florian Sesser, Kyosuke Tokoro
```

操作结束后退出容器，回到 build 用户身份下。

```
[root@68bdb5af0da9 /]# exit
[build@container-x86 ~]$ whoami
build
```

使用 `docker ps -a` 查看与先前定义名字对应的 **container ID**，在这个示例中是 `MY_USERNAME_DATE`。

```
[build@container-x86 ~]$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
515e913f12cb	5d0da3dc9764	"/bin/bash"	4 seconds ago	Exited (0) 2 seconds ago

Names: MY_USERNAME_DATE

使用 `docker commit CONTAINER_ID IMG_NAME` 提交容器变更。

```
$ docker commit 515e913f12cb my_username_app_img
```

此时使用 `docker images` 可以在容器镜像列表中看到刚刚提交的容器变更。

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my_username_app_img	latest	c26c43a0cc9b	About a minute ago	279MB

将 **Docker** 容器保存为可在超算平台上使用的 **Singularity** 镜像。

```
$ SINGULARITY_NOHTTPS=1 singularity build my_username_app_img.sif docker-
↳ daemon://my_username_app_img:latest
INFO: Starting build...
INFO: Creating SIF file...
INFO: Build complete: my_username_app_img.sif
```

使用 `scp sample-x86.sif YOUR_USERNAME@login1:~/` 将 **Singularity** 镜像文件复制到超算集群家目录后，可以使用 `singularity` 命令测试镜像文件，从 `/etc/redhat-release` 内容和 `tree` 命令版本看，确实进入了与宿主操作系统不一样的运行环境。

```
$ singularity exec my_username_app_img.sif cat /etc/redhat-release
CentOS Linux release 8.4.2105
$ singularity exec my_username_app_img.sif tree --version
tree v1.7.0 (c) 1996 - 2014 by Steve Baker, Thomas Moore, Francesc Rocher,
↳ Florian Sesser, Kyosuke Tokoro
```


4.9.3 参考资料

- Singularity Quick Start https://sylabs.io/guides/3.4/user-guide/quick_start.html
- Docker Hub <https://hub.docker.com/>
- NVIDIA GPU CLOUD <https://ngc.nvidia.com/>
- 更多 Singularity Definition Files 的例子请参考 <https://github.com/SJTU-HPC/hpc-base-container/tree/dev/base/>

4.10 软件

交我算 HPC+AI 集群有许多预建的软件模块，并且数量还在不断增长，欢迎您告诉我们您研究领域中流行的软件。

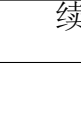
集群使用 Environment Module 加载软件，您可参阅软件模块使用方法 文档。

软件按以下类别列出，可以使用文档页面顶栏的搜索框查找软件。

- 基准测试
- 原子分子软件
- 工程计算软件
- AI 计算软件
- 生信计算软件

编译器、MPI 库、数学库、应用工具

表 1: 编译器 MPI 库数学库应用工具

软件	描述	可用版本	平台
<i>GNU Compiler Collection</i>	GNU 编译器套件	9.3(D)	
<i>Armadillo</i>	采用 c++ 实现的类 matlab 风格的矩阵运算库	10.5.0	
<i>Intel Compiler</i>	Intel 编译器套件	21.4.0	
<i>cuBLAS</i>	NVIDIA 官方提供的 GPU 上的高性能矩阵运算库		
<i>CUDA</i>	NVIDIA CUDA SDK	10.2(D)	
<i>cuSOLVER</i>	NVIDIA 官方提供的 GPU 上的线性代数求解器		
<i>cuSPARSE</i>	NVIDIA 官方提供的 GPU 上的稀疏矩阵求解器		

续下页

表 1 – 接上页

软件	描述	可用版本	平台
<i>cuFFT</i>	NVIDIA 官方提供的 GPU 上的快速傅里叶变换运算库		gpu
<i>NVIDIA SDK</i>	<i>HPC</i> NVIDIA HPC SDK	20.11(D)	gpu
<i>JDK</i>	Java 开发套件	12.0	cpu arm
<i>bisheng</i>	ARM 平台高性能编译器	1.3.3	arm
<i>OpenMPI</i>	OpenMPI 是一个开源 MPI 实现	3.1.5	cpu arm
<i>Mpi4py</i>	mpi4py 是通过 python 实现的 MPI	3.1.4	cpu
<i>Numba</i>	用于 python 代码加速		cpu
<i>Cupy</i>	GPU 上运行的“numpy”		gpu
<i>PyCUDA</i>	python 语言 cuda 编程		gpu
<i>Intel-MPI</i>	Intel MPI 是一个高性能 MPI 实现	2019.4	cpu
<i>Intel-Vtune</i>	Intel MPI 是一个高性能 MPI 实现	2019.4	cpu
<i>Hyper-MPI</i>	ARM 平台高性能 MPI 实现	4.0.3	arm
<i>Intel mkl</i>	Intel MKL(数学核心函数库) 包含 BLAS、LAPACK、SCALAPACK、的高效实现	19.3	cpu
<i>OpenBLAS</i>	OpenBLAS 是一个开源的 BLAS、LAPACK 高效实现	2.4	cpu arm
<i>GSL</i>	GNU Scientific Library 是由 C/C++ 编写的高效开源科学计算库	2.5	cpu arm
<i>Eigen</i>	Eigen 是一个 C++ 编写的线性代数模板库	3.3.7	cpu arm
<i>FFTW</i>	FFTW 是一个被广泛使用快速傅里叶变换库	3.3.7	cpu arm
<i>Hypre</i>	hypre 是一个求解大型线性方程组的 C 语言库	2.20.0	cpu

续下页

表 1 – 接上页

软件	描述	可用版本	平台
<i>Conda</i>	环境管理与包管理软件	4.6.14	cpu arm
<i>Python</i>	一种广泛使用的解释型、高级和通用的编程语言	3.8, 3.7	cpu arm
<i>PERL</i>	一种功能丰富的计算机程序语言	5.30	cpu arm
<i>R</i>	为数学研究工作者设计的一种数学编程语言	3.6.2	cpu Studio
<i>Jupyter</i>	为数十种编程语言的交互式计算开发开源软件，开放标准和服务		gpu Studio
<i>MATLAB</i>	商业数学软件，用于数据分析	2023b	cpu Studio
<i>CMake</i>	跨平台的编译工具，生成 Makefile 文件		cpu
<i>glibc</i>	GNU 发布的 libc 库	2.29	cpu
<i>SPOOLES</i>	面向对象的稀疏线性等式解析器	2.2	cpu
<i>Deal.II</i>	基于 C++ 编写的有限元软件	9.3.3	cpu
compilers_and_language cvmfs	Cern 开发的分布式文件系统	2.10.1	cpu gpu
<i>nvitop</i>	开源的 GPU 监控工具	1.0.0	gpu

基准测试

表 2: 基准测试

软件	描述	可用版本	平台
<i>OSU Benchmarks</i>	由俄亥俄州立大学开发的高速网络基准测试	5.6.3	 
<i>HPL</i>	用于 Top500 超算排名的高性能线性方程求解基准测试程序, 测量超算集群性能	2.3	 
<i>HPCG</i>	新的 HPC 系统排名指标, 可作为 HPL 基准程序的补充	3.1	
<i>minife</i>	非结构化隐式有限元代码的代理应用程序	2.1.0	
<i>IOR</i>	用于测量集群并行 IO 读写能力的测试程序	3.3.0	
<i>mdtest</i>	用于测量集群元数据性能的测试程序	3.3.0	
<i>io500</i>	用于测量集群综合存储 IO 性能的测试程序	sc23_v1	

原子分子软件

表 3: 原子分子软件

软件	描述	可用版本	平台
<i>ABACUS</i>	专为大规模电子结构模拟而设计的开源软件包	v3.9.0.17	cpu
<i>ABINIT</i>	第一原理平面波赝势总能计算软件包	8.10.3	cpu
<i>GAUSSIAN</i>	量子化学软件包	自己安装	cpu
<i>GPUMD</i>	在 GPU 上实现的通用分子动力学软件包，提供高效 NEP 机器学习势函数	4.5	gpu
<i>GROMACS</i>	研究生物分子体系的分子动力学程序包	2019 2020	cpu gpu arm
<i>VASP</i>	量化计算软件	5.4.4	cpu gpu arm
<i>Amber</i>	分子动力学模拟软件	20 22	cpu gpu arm
<i>LAMMPS</i>	桑迪亚实验室开发的一套分子动力学模拟器，原子分子大规模并行模拟。	2020 2021	cpu gpu arm
<i>LAMMPS-RBE</i>	上海交通大学基于 lammps 二次开发的自研软件	2021	cpu arm
<i>NWChem</i>	量子化学计算软件	6.8.1	cpu
<i>Quantum ESPRESSO</i>	纳米尺度上进行电子结构计算和材料建模	6.6	cpu arm
<i>CP2k</i>	基于密度泛函理论的从头算分子动力学软件	6.1	cpu gpu arm
<i>SIESTA</i>	用于分子和固体的电子结构计算和分子动力学模拟	4.0.1	cpu
<i>VMD</i>	分子可视化程序	1.9.4	cpu Studio
4.10. 软件	分析数据及可视化动力学结果	3.2.1	cpu Studio
<i>OVITO</i>	开源、跨平台数据分析和可视化程序	3.4.1	cpu

工程计算软件

表 4: 工程计算软件

软件	描述	可用版本	平台
<i>CESM</i>	地球气候系统模式	2.2	cpu
<i>Calculix</i>	开源有限元分析软件	2.17	cpu
<i>Cantera</i>	开源化学反应动力学分析软件	3.0.0	cpu
<i>WRF</i>	中尺度天气预报模式	3.9,4.3.1	cpu arm
<i>CMAQ</i>	中尺度天气预报模式	3.9,4.3.1	cpu arm
<i>cplex</i>	大型线性以及非线性规划求解软件		cpu
<i>Nektar++</i>	有限元分析软件	5.0.0	cpu
<i>Octave</i>	用于数值计算和绘图的开源软件，比 matlab 更加轻型便捷，适用于小型场合。	5.2.0	cpu Studio
<i>OpenFOAM</i>	面向对象的计算流体力学类库	7, 1712, 1812,	cpu
<i>OpenRadioss</i>	开源显式动力学求解器		cpu
<i>WhiskyTHC</i>	开源的大型数值相对论求解器		cpu
<i>PSI4</i>	面向对象的计算化学软件	1.5	cpu
<i>PySPH</i>	基于 python 的光滑粒子流体动力学框架	1.5	cpu
<i>STAR-CCM+</i>	新一代通用计算流体力学分析软件	2.7.0	cpu
<i>Gerris</i>	求解描述流体流动的偏微分方程的软件	1.3.2	cpu
<i>SUMO</i>	开源交通信号模拟软件	1.10.0	cpu
<i>Elphbolt</i>	用于求解耦合电子和声子-玻尔兹曼输运方程	1.0	cpu
<i>DAMSK</i>	多物理晶体塑形模拟包，可有效处理多物理问题	3.0.0-alpha6	cpu
<i>ROMS</i>	ROMS 是三维区域海洋模型，用于海洋及河口地区的水动力及水环境模拟	3.6	cpu
<i>PROJ</i>	PROJ 是通用的坐标转换软件	7.2.1	cpu
<i>colmap</i>	COLMAP 是用来做 SfM 或者 MVS 的软件	3.7	cpu
4.10. <i>pymultinest</i> 软件	pymultinest 可用于贝叶斯分析、参数选择等。	2.11	cpu
<i>LORENE</i>	用于解决数值相对论和计算天体物理学中的各种问题	2006	cpu

AI 计算软件

表 5: AI 计算软件

软件	描述	可用版本	平台
<i>PyTorch</i>	开源的 python 机器学习库	1.6.0	gpu
<i>TensorFlow</i>	用于各类机器学习算法的编程实现	2.0.0	gpu
<i>Keras</i>	Python 编写的开源人工神经网络库能在 TensorFlow、Theano 或 PlaidML 上运行。	2.1.0, 2.3.0	cpu gpu
<i>Apache TVM</i>	开源的机器学习编译框架	0.8.0	arm

生信计算软件

表 6: 生信计算软件

软件	描述	可用版本	平台
<i>AlphaFold2</i>	DeepMind 开源蛋白质结构，深度学习算法通过蛋白质序列来预测蛋白质结构。	2.1	gpu
<i>AUGUSTUS</i>	Augustus 是用作 de novo 基因注释 (即从头预测) 的软件。	1.5.1	cpu
<i>BASIL-ANISE</i>	从 BAM 格式的对齐配对 HTS 读数中检测结构变体断点的方法	0.2.13	cpu
<i>BatVI</i>	检测病毒整合的软件包	2.1.0	cpu
<i>BCFtools</i>	vcf 和 BCF 文件的工具合集	1.10.2	cpu arm
<i>BEDTOOLS2</i>	可以对 genomic features 进行比较、相关操作和注释的工具	2.27.1	cpu
<i>BICseq2</i>	SV 检测软件	0.3.0	cpu
<i>Bismark</i>	全基因组甲基化软件	0.19.0	cpu
<i>Blast-plus</i>	基于局部比对算法的搜索工具	2.9.0	arm
<i>BOWTIE</i>	一种快速短对齐器	1.2.3	cpu
<i>BOWTIE2</i>	短序列比对软件	2.3.5.1	cpu
<i>BreakDancer</i>	C++ 软件包，可提供下一代配对末端测序读取的全基因组结构变异检测	2019-5-12	cpu

续下页

表 6 – 接上页

软件	描述	可用版本	平台
<i>BWA</i>	一款将序列比对到参考基因组上的软件	0.7.17	cpu
<i>CCMpred</i>	一种用于预测蛋白质残基间接触的软件	last branch	gpu
<i>cd-hit</i>	序列聚类软件	4.8.1	cpu
<i>CDO</i>	包含大量标准处理气候和预报模式数据的算子的集合	1.9.8	cpu
<i>cdsapi</i>	允许用户通过 Python 脚本从 CDS 数据集请求数据的服务	1.5.1	cpu
<i>chimera</i>	一个用于聚合产物二次分析的软件包	1.32.0	cpu
<i>CLEVER</i>	用于分析下一代测序数据的工具	10.6.2	cpu
<i>CNVnator</i>	一种用于发现和表征群体基因组测序数据中拷贝数变异 (CNV) 的工具	3.8	cpu
<i>ColabFold</i>	AI 蛋白结构预测软件	1.2	gpu
<i>Control-FERRC</i>	一种使用深度测序数据检测拷贝数变化和等位基因不平衡 (包括 LOH) 的工具	0.11.7	cpu
<i>CREST</i>	一种使用下一代测序数据以碱基对分辨率检测基因组结构变异的新算法	10.0.130	cpu gpu
<i>CUFFLINKS</i>	主要用于基因表达量的计算和差异表达基因的寻找	2.2.1	cpu
<i>DeepGo</i>	使用深度学习识别分类器根据序列和相互作用预测蛋白质功能	2.4.3	cpu
<i>DeepVariant</i>	开源工具，用深度神经网络从 DNA 测序数据中快速较精确识别碱基变异位点	10.0.130	cpu gpu
<i>DELLY</i>	一种集成的结构变异 (SV) 预测方法	2.2.1, 0.8.3	cpu arm
<i>DESeq2</i>	分析来自 RNA-seq 的计数数据的一项基本任务是检测差异表达的基因	1.3.4d	cpu
<i>ERDS</i>	实体关系图	2.2.1	cpu
<i>FastQC</i>	一款基于 Java 快速多线程地对测序数据进行质量评估的软件	0.11.7	cpu
<i>FermiKit</i>	Illumina 全基因组 germline 变异检出流程	0.11.7	cpu

续下页

表 6 – 接上页

软件	描述	可用版本	平台
<i>FSL</i>	核磁共振、DTI 脑成像数据分析软件	6.0	cpu
<i>GATK</i>	二代重测序数据分析软件	3.8	cpu
<i>GEANT4</i>	C++ 软件包, 用于模拟粒子在物质中运输的物理过程	10.6.2	cpu
<i>GenomeStrip</i>	识别拷贝数变异区	10.0.130	cpu gpu
<i>GMAP-GSNAP</i>	mRNA 和 EST 序列的基因组定位和比对程序	2019-5-12	cpu
<i>GRAPHMAP</i>	一个高度敏感和准确的映射器	0.3.0	cpu
<i>GRIDSS</i>	用于 Web 的二维布局系统	2.2.1	cpu
<i>GsUtil</i>	Python 应用程序, 可让您从命令行访问 Google Cloud Storage	1.3.4d	cpu
<i>HISAT2</i>	快速、灵敏的比对软件	2.1.0	cpu arm
<i>HMMER</i>	序列比对软件	3.3	arm
<i>Hydra-sv</i>	使用双端映射检测结构变异 (SV) 断点	0.7.17	cpu
<i>km</i>	使用 k-mer 分解进行 RNA-seq 研究的软件	240	cpu
<i>Lumpy</i>	用于基因组结构变异 SV 的检测	1.2.3	cpu
<i>LUMPY-SV</i>	一款 SV 检测工具	0.2.13	cpu
<i>MAKER</i>	基因组注释流程	2.1.2	cpu
<i>Manta</i>	检测单个样品的 germline 变异和 tumor/normal 配对样品的 somatic 变异	1.6.0	arm
<i>MEGAHIT</i>	二代测序从头组装工具	1.1.4	cpu
<i>METIS</i>	图切分软件包	5.1.0	cpu
<i>MELT</i>	模拟岩浆演化的工具	2.27.1	cpu
<i>MetaSV</i>	用于下一代测序的准确且综合的结构变异检测器	1.1.4	cpu
<i>MindTheGap</i>	虚拟原型中的 GPU 建模	5.1.0	cpu

续下页

表 6 – 接上页

软件	描述	可用版本	平台
<i>Mobster</i>	一个 mob/pair 编程计时器	3.2.7a	cpu
<i>MrBayes</i>	系统树重建软件	3.2.7a	cpu
<i>NCBI-RMBLASTN</i>	基于序列相似性的数据库搜索程序	2.2.28	cpu
<i>OpenSlide-python</i>	C 库，它提供了一个简单的界面来读取整张幻灯片图像		cpu
<i>pandas</i>	使用 python 进行数据分析的基本工具，用来分析表结构的数据	2.1.2	cpu
<i>PBSV</i>	从 PacBio SMRT 读数中检出和分析二倍体基因组中的结构变异	2.2.28	cpu
<i>PGA</i>	通过 RNA-Seq 数据识别新肽	2.2.28	cpu
<i>PERCOLATOR</i>	从鸟枪法蛋白质组学数据集中鉴定肽的半监督学习	3.5.0	cpu
<i>PICARD</i>	一组命令行工具，用于处理高通量排序 (HTS) 数据和格式	2.19.0	cpu
<i>Pindel</i>	检测复杂 INDEL	2.19.0	cpu
<i>PRISM</i>	多功能统计工具	1.1.8, 3.6.2	cpu Studio
<i>r-rgl</i>	为 3D 交互式图形提供中高级函数	240	cpu
<i>RELION</i>	针对单颗粒冷冻电镜图片进行处理的框架	3.0.8, 3.1.3	gpu Studio
<i>RetroSeq</i>	基因分型转座元件变体 (TEV) 检测工具	3.0.8	gpu Studio
<i>RFdiffusion</i>	使用蛋白结构预测，反向运用到蛋白质结构设计	1.1.0	gpu
<i>RNA-SEQC</i>	计算一系列对 RNA-Seq 数据进行质量控制的计量学的 java 程序	1.1.8	cpu
<i>RoseTTAFold</i>	开源蛋白质结构预测软件	1.0	gpu
<i>STAR</i>	测序序列与参考序列比对，常用于 RNA-seq 的比对	2.7.6.a	cpu

续下页

表 6 – 接上页

软件	描述	可用版本	平台
<i>SALMON</i>	一种从 RNA-seq 数据中快速量化转录本的工具	0.14.1	cpu
<i>Samtools</i>	用于操作 sam 和 bam 文件的工具合集	1.9	cpu
<i>Sniffles</i>	用于检测长读长数据的 SV	1.1.8	cpu
<i>SOAPDENOVO2</i>	用于序列组装的软件	240	cpu
<i>SRATOOLKIT</i>	将.sra 文件转换为.fstq.gz 文件的工具	2.9.6	cpu
<i>sra-tools</i>	使用 INSDC 序列读取档案中的数据的工具和库的集合	2.9.6	cpu
<i>STRINGTIE</i>	转录组表达定量软件	1.3.4d	cpu
<i>STRique</i>	Python 包，用于分析 ONT 长读长测序数据中 STR 的重复扩展和甲基化状态		cpu
<i>SV2</i>	一种机器学习算法，对双端全基因组测序数据中的缺失和重复进行基因分型	0.14.1	cpu
<i>SvABA</i>	全基因组局部组装检测测序数据中结构变异的软件	1.1.3	cpu
<i>SVDetect</i>	从测序数据中分离和预测染色体内部和染色体间重排的应用程序	240	cpu
<i>TOPHAT</i>	将 RNA-Seq 数据进行快速剪接映射的程序	2.1.2	cpu
<i>VARDICTJAVA</i>	用 Java 和 Perl 编写的变种发现程序	1.5.1	cpu
<i>VSEARCH</i>	开源免费的 64 位，无内存限制的扩增子数据处理分析软件	2.4.3	cpu
<i>VariationHunter</i>	变异检测软件	1.9.8	cpu
<i>Wham</i>	加权直方图分析方法	2.9.6	cpu
<i>WGCNA</i>	加权相关网络分析		cpu
<i>Rosetta</i>	模拟大分子结构的综合软件	3.12	cpu
<i>Hic_breakfinder</i>	基因组 SV 检测工具	1.0	cpu
<i>EasyFuse</i>	融合基因检测工具	1.3.5	cpu
<i>MetaWRAP</i>	宏基因组数据分析流程工具	1.3	cpu

续下页

表 6 – 接上页

软件	描述	可用版本	平台
<i>GATE</i>	医学成像和放射治疗的数值模拟软件	9.2	cpu
<i>pangenie</i>	基于范基因组的基因分型器	v2.0.0	cpu
<i>Cell Ranger</i>	基因序列分析流程	v7.2.0	cpu
<i>Minimap2</i>	基因序列比对程序	v2.14	cpu

4.10.1 软件模块使用方法

module 命令

集群软件以 **module** 形式供全局调用。常见的 **module** 命令如下

`module load [MODULE]`: 加载模块

`module avail` 或 `module av`: 列出所有模块

`module av intel`: 列出含有 **intel** 名字的所有模块

`module list`: 列出所有已加载的模块

`module show [MODULE]`: 列出该模块的信息，如路径、环境变量等

也可以一次加载或卸载多个模块。

```
$ module load gcc openmpi
$ module unload gcc openmpi
```

如果您喜欢最新的稳定版本，则可以忽略版本号（默认加载带有 **D** 标识的版本）。

下方两句命令效果一致：

```
$ module load gcc openmpi
$ module load gcc/9.3.0-gcc-4.8.5 openmpi/4.0.5-gcc-9.2.0
```

```
----- /lustre/share/spack/
->modules/cascadelake/linux-centos7-x86_64 -----
->-----
abinit/8.10.3-gcc-9.2.0-openblas-openmpi          hdf5/1.10.6-gcc-9.2.0-
->openmpi                                           netcdf-fortran/4.5.2-intel-19.0.4-imp
-> (D)
alphafold/2-python-3.8                            hdf5/1.10.6-gcc-9.2.0
->                                                  netlib-lapack/3.8.0-intel-19.0.4      (D)
bcftools/1.9-gcc-9.2.0                            (D)      hdf5/1.10.6-intel-19.0.4-
->impi                                              nwchem/6.8.1-intel-19.0.4-imp
bedtools2/2.27.1-intel-19.0.4                    (D)      hdf5/1.10.6-intel-19.0.5-
->impi                                              openblas/0.3.7-gcc-9.2.0
-> (D)
```

(续下页)

(接上页)

bismark/0.19.0-intel-19.0.4		hdf5/1.10.6-intel-19.0.5-
→openmpi	(D)	openjdk/1.8.0_222-b10-gcc-9.2.0
boost/1.70.0-gcc-9.2.0		hisat2/2.1.0-intel-19.0.4
→	(D)	openjdk/11.0.2-gcc-9.2.0
boost/1.70.0-intel-19.0.4		hypre/2.20.0-gcc-9.2.0-
→openblas-openmpi	(D)	openjdk/11.0.2-intel-19.0.4
→ (D)		
boost/1.70.0-intel-19.0.5		(D) intel-mkl/2019.3.199-intel-
→19.0.4		openmpi/3.1.5-gcc-9.2.0
bowtie/1.2.3-gcc-9.2.0		(D) intel-mkl/2019.5.281-intel-
→19.0.5		openmpi/3.1.5-gcc-9.3.0
bowtie2/2.3.5.1-intel-19.0.4		intel-mkl/2020.1.217-intel-
→19.1.1	(D)	openmpi/3.1.5-intel-19.0.5
bwa/0.7.17-gcc-9.2.0		intel-mpi/2019.4.243-intel-
→19.0.4		openmpi/4.0.5-gcc-9.2.0
bwa/0.7.17-intel-19.0.4		(D) intel-mpi/2019.6.154-gcc-9.
→2.0		perl/5.30.0-gcc-9.2.0
cdo/1.9.8-gcc-9.2.0		intel-mpi/2019.6.154-intel-
→19.0.5	(D)	perl/5.30.0-gcc-9.3.0
cp2k/8.2-gcc-9.2.0-openblas		(D) intel-parallel-studio/
→cluster.2019.4-intel-19.0.4		perl/5.30.0-intel-19.0.4
cuda/9.0.176-intel-19.0.4		intel-parallel-studio/
→cluster.2019.5-intel-19.0.5		perl/5.30.0-intel-19.0.5
cuda/10.1.243-gcc-9.2.0		intel-parallel-studio/
→cluster.2020.1-intel-19.1.1	(D)	perl/5.30.0-intel-19.1.1
→ (D)		
cuda/10.2.89-intel-19.0.4		(D) jasper/2.0.16-gcc-9.2.0
→		picard/2.19.0-gcc-9.2.0
cufflinks/2.2.1-gcc-9.2.0		jdk/12.0.2_10-gcc-9.2.0
→		picard/2.19.0-intel-19.0.4
cufflinks/2.2.1-intel-19.0.4		(D) jdk/12.0.2_10-intel-19.0.4
→	(D)	python/2.7.16-intel-19.0.4
eigen/3.3.7-gcc-9.2.0		(D) json-fortran/7.1.0-gcc-9.2.
→0		python/2.7.16-intel-19.1.1
fastqc/0.11.7-gcc-9.2.0		lammps/20200721-intel-19.0.
→5-openmpi		python/3.7.4-gcc-9.2.0
fastqc/0.11.7-intel-19.0.4		(D) lammps/20210310-intel-19.0.
→5-openmpi		python/3.7.4-intel-19.0.4
fftw/3.3.8-gcc-9.2.0-openmpi		lammps/20210702-intel-19.0.
→5-impi		python/3.7.4-intel-19.0.5
fftw/3.3.8-gcc-9.2.0		libxc/4.3.2-gcc-9.2.0
→		quantum-espresso/6.4.1-intel-19.0.4-impi
fftw/3.3.8-gcc-9.3.0-openmpi		libxc/4.3.2-intel-19.0.4
→	(D)	quantum-espresso/6.4.1-intel-19.0.5-impi
fftw/3.3.8-intel-19.0.4-impi		lumpy-sv/0.2.13-gcc-9.2.0
→		quantum-espresso/6.5-intel-19.0.4-impi
fftw/3.3.8-intel-19.0.5-impi		mcl/14-137-gcc-9.2.0
→		quantum-espresso/6.5-intel-19.0.5-impi
fftw/3.3.8-intel-19.0.5-openmpi		megahit/1.1.4-intel-19.0.4

(续下页)

(接上页)

→	(D)	rna-seqc/1.1.8-gcc-9.2.0	
fftw/3.3.8-intel-19.1.1-impi		metis/5.1.0-gcc-9.2.0	→
→	(D)	rna-seqc/1.1.8-intel-19.0.4	(D)
fftw/3.3.9-gcc-9.2.0-openmpi		mpich/3.3.2-gcc-9.2.0	→
→		rosettafold/1-python-3.8	
fftw/3.3.9-gcc-9.3.0		(D) mpich/3.3.2-intel-19.0.4	→
→		samtools/1.9-gcc-9.2.0	
flash/1.2.11-gcc-9.2.0		(D) mpich/3.3.2-intel-19.0.5	→
→	(D)	samtools/1.9-intel-19.0.4	(D)
fsl/6.0-fsl-gcc-4.8.5		mvapich2/2.3.2-intel-19.0.5	→
→	(D)	siesta/4.0.1-intel-19.0.4-impi	
gatk/3.8-1-gcc-9.2.0		(D) namd/2.14-gcc-9.2.0-openmpi	→
→		stream/5.10-intel-19.0.4	
gromacs/2019.2-gcc-9.2.0-openmpi		nano/4.7-gcc-4.8.5	→
→		stream/5.10-intel-19.0.5	(D)
gromacs/2019.4-gcc-9.2.0-openmpi		nektar/5.0.0-intel-19.0.4-	
→impi		sumo/1.10.0-sumo	
gromacs/2020.2-gcc-9.2.0-openmpi		netcdf-c/4.7.3-gcc-9.2.0-	
→openmpi		sundials/3.1.2-gcc-9.2.0	
gromacs/2021-gcc-9.3.0-openmpi		netcdf-c/4.7.3-gcc-9.2.0	→
→		svaba/1.1.3-gcc-4.8.5	
gsl/2.5-gcc-9.2.0		netcdf-c/4.7.3-intel-19.0.4-	
→impi		tophat/2.1.2-intel-19.0.4	
gsl/2.5-intel-19.0.4		netcdf-c/4.7.3-intel-19.0.5-	
→impi		vsearch/2.4.3-intel-19.0.4	(D)
gsl/2.5-intel-19.0.5		(D) netcdf-c/4.7.3-intel-19.0.5-	
→openmpi	(D)	wrf/4.2-gcc-9.2.0-openmpi	
hdf5/1.10.5-intel-19.0.4-impi		netcdf-fortran/4.5.2-gcc-9.	
→2.0-openmpi			

在 SLURM 上，我们应用了以下规则来选取最合适的模块。

1. 编译器：如果加载了 gcc 或 icc，请根据相应的编译器加载已编译的模块。或在必要时加载默认的编译器 gcc。
2. MPI 库：如果已加载其中一个库 (openmpi, impi, mvapich2, mpich)，加载针对相应 MPI 编译的模块。在必要的时候，默认 MPI lib 中的 openmpi 将被装载。
3. Module 版本：每个模块均有默认版本，如果未指定版本号，则将加载该默认版本。

参考资料

- Lmod: A New Environment Module System <https://lmod.readthedocs.io/en/latest/>
- Environment Modules Project <http://modules.sourceforge.net/>
- Modules Software Environment on NERSC <https://www.nersc.gov/users/software/nersc-user-environment/modules/>

4.10.2 编译器、MPI 库、数学库、应用工具

GNU Compiler Collection

GNU(GNU Compiler Collection), 缩写为“GCC”, 即“GNU 编译器套件”, 可将其理解成是多个编译器的集合。支持的语言包括 C、C++、Fortran、Ada、Object-C 和 Java 等。

查看 **GCC** 的版本

```
[hpc@node499 ~]$ gcc --version
gcc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-22)

[hpc@node499 ~]$ g++ --version
g++ (GCC) 8.5.0 20210514 (Red Hat 8.5.0-22)

[hpc@node499 ~]$ gfortran --version
GNU Fortran (GCC) 8.5.0 20210514 (Red Hat 8.5.0-22)
```

集群平台上的 **GCC**

- 思源一号上的 *GCC*
- $\pi 2.0$ 上的 *GCC*
- *ARM* 上的 *GCC*

思源一号上的 **GCC**

版本	加载方式
gcc-8.5.0	module load gcc/8.5.0
gcc-9.3.0	module load gcc/9.3.0
gcc-9.4.0	module load gcc/9.4.0
gcc-10.3.0	module load gcc/10.3.0
gcc-11.2.0	module load gcc/11.2.0
gcc-12.3.0	module load gcc/12.3.0

思源一号上的 GCC 默认版本为: 8.5.0。

$\pi 2.0$ 上的 **GCC**

版本	加载方式
gcc-9.3.0	module load gcc/9.3.0
gcc-10.2.0	module load gcc/10.2.0
gcc-11.2.0	module load gcc/11.2.0
gcc-12.3.0	module load gcc/12.3.0
gcc-13.2.0	module load gcc/13.2.0

π 2.0 上的 GCC 默认版本为: 8.5.0。

ARM 上的 GCC

ARM 平台上的 GCC 默认版本为: 10.3.1。

参考资料

- Top 20 licenses <https://www.blackducksoftware.com/top-open-source-licenses/>

Armadillo

简介

Armadillo 是目前使用比较广的 C++ 矩阵运算库之一，是在 C++ 下使用 Matlab 方式操作矩阵很好的选择，许多 matlab 的矩阵操作函数都可以找到对应，这对习惯了 matlab 的人来说实在是非常方便。他们之间的接口调用方式非常相似。

Armadillo 使用说明

思源一号上的 Armadillo

1. 先创建一个目录 `armadillotest` 并进入该目录：

```
mkdir armadillotest
cd armadillotest
```

2. 在该目录下创建如下测试文件 `armadillotest.cpp`：

```
// 对矩阵A进行PLU分解，其中P为置换矩阵，PA=LU

#include <random>
#include <algorithm>
#include <stdlib.h>
#include <iostream>
#include <ctime>
#include <armadillo>

using namespace arma;

int main()
{
    int N = 5;
    arma::mat A = zeros<mat>(N, N);
    for (size_t i = 0; i < N; i++)
    {
        A(i, i) = i + 1.1;
        if (i != N - 1)
        {
```

(续下页)

(接上页)

```

        A(i, i + 1) = i + 4.4;
        A(i + 1, i) = i + 3.3;
    }
    if (i != N - 1 && i != N - 2)
    {
        A(i + 2, i) = i + 3.3;
    }
}

// 定义一个permutation matrix对象p
arma::mat P;
arma::mat L;
arma::mat U;

// 使用Armadillo的lu函数进行LU分解
struct timespec start1, end1;
double duration1;
clock_gettime(CLOCK_REALTIME, &start1);

arma::lu(L, U, P, A);

clock_gettime(CLOCK_REALTIME, &end1);
duration1 = (end1.tv_sec - start1.tv_sec) + (end1.tv_nsec - start1.tv_
↪nsec) / 1000000000.0;
printf("The time is %lf microseconds\n", duration1 * 1000);

std::cout << "A:\n"
            << A << std::endl;
std::cout << "P:\n"
            << P << std::endl;
std::cout << "L:\n"
            << L << std::endl;
std::cout << "U:\n"
            << U << std::endl;
std::cout << "L*U:\n"
            << L * U << std::endl;
std::cout << "P*A:\n"
            << P * A << std::endl;

return 0;
}

```

3. 在该目录下创建如下作业提交脚本 armadillotest.slurm:

```

#!/bin/bash

#SBATCH --job-name=armadillotest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1

```

(续下页)

(接上页)

```
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load armadillo/10.5.0-gcc-11.2.0-openblas
module load gcc/11.2.0

g++ armadillotest.cpp -larmadillo -o armadillotest

./armadillotest
```

4. 使用如下命令提交作业:

```
sbatch armadillotest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
The time is 0.094627 microseconds
A:
  1.1000  4.4000      0      0      0
  3.3000  2.1000  5.4000      0      0
  3.3000  4.3000  3.1000  6.4000      0
      0  4.3000  5.3000  4.1000  7.4000
      0      0  5.3000  6.3000  5.1000

P:
      0  1.0000      0      0      0
      0      0      0  1.0000      0
  1.0000      0      0      0      0
      0      0  1.0000      0      0
      0      0      0      0  1.0000

L:
  1.0000      0      0      0      0
      0  1.0000      0      0      0
  0.3333  0.8605  1.0000      0      0
  1.0000  0.5116  0.7879  1.0000      0
      0      0 -0.8333  0.4745  1.0000

U:
  3.3000  2.1000  5.4000      0      0
      0  4.3000  5.3000  4.1000  7.4000
      0      0 -6.3605 -3.5279 -6.3674
      0      0      0  7.0821  1.2311
      0      0      0      0 -0.7899
```

(续下页)

(接上页)

L*U:				
3.3000	2.1000	5.4000	0	0
0	4.3000	5.3000	4.1000	7.4000
1.1000	4.4000	0	0	0
3.3000	4.3000	3.1000	6.4000	0
0	0	5.3000	6.3000	5.1000
P*A:				
3.3000	2.1000	5.4000	0	0
0	4.3000	5.3000	4.1000	7.4000
1.1000	4.4000	0	0	0
3.3000	4.3000	3.1000	6.4000	0
0	0	5.3000	6.3000	5.1000

参考资料

- [Armadillo 官网](#)
- [Armadillo 知乎](#)

Intel Compiler

Intel 编译器套件包含以下语言：

C、 C++、 Fortran

集群上的 Intel 编译器

版本	加载方式	平台
intel-21.4.0	module load oneapi/2021.4.0	思源一号
intel-21.4.0	module load oneapi/2021.4.0	pi2.0

加载 oneapi 模块后，会同时导入 MKL 库、intel compiler 等。

```
module load oneapi/2021.4.0
module list
Currently Loaded Modules:
1) intel-oneapi-compilers/2021.4.0    2) intel-oneapi-mpi/2021.4.0
3) intel-oneapi-mkl/2021.4.0         4) intel-oneapi-tbb/2021.4.0
```

常见问题

1. 编译/运行程序时提示 “license has expired”

A: License 过期不影响原有软件运行，新编译软件建议使用最新部署的 intel 编译器套件进行编译 `module load oneapi/2021.4.0`。

参考资料

- 一篇比较详细的 [intel 程序优化教程](#)
- [intel-compiler 相关文档](#)

Intel mkl

简介

Intel mkl 是一套经过高度优化和广泛线程化的数学例程，专为需要极致性能的科学、工程及金融等领域的应用而设计。核心数学函数包括 BLAS、LAPACK、ScaLAPACK1、稀疏矩阵解算器、快速傅立叶转换、矢量数学及其它函数。它可以为当前及下一代英特尔处理器提供性能优化，包括更出色地与 Microsoft Visual Studio、Eclipse 和 XCode 相集成。Intel mkl 支持完全集成英特尔兼容性 OpenMPI 运行时库，以实现更出色的 Windows/Linux 跨平台兼容性。

Intel mkl 使用说明

思源一号上的 Intel mkl

1. 先创建一个目录 `intelmkltest` 并进入该目录：

```
mkdir intelmkltest
cd intelmkltest
```

2. 在该目录下创建如下测试文件 `intelmkltest.c`：

```
#include <stdio.h>
#include <stdlib.h>
#include "mkl.h"

#define min(x,y) (((x) < (y)) ? (x) : (y))

int main()
{
    double *A, *B, *C;
    int m, n, k, i, j;
    double alpha, beta;

    printf ("\n This example computes real matrix C=alpha*A*B+beta*C using \n"
           " Intel(R) MKL function dgemm, where A, B, and C are matrices.
    ↪and \n"
           " alpha and beta are double precision scalars\n\n");

    m = 2000, k = 200, n = 1000;
    printf (" Initializing data for matrix multiplication C=A*B for matrix \n"
           " A(%ix%i) and matrix B(%ix%i)\n\n", m, k, k, n);
    alpha = 1.0; beta = 0.0;
```

(续下页)

(接上页)

```

printf (" Allocating memory for matrices aligned on 64-byte boundary for_
↪better \n"
        " performance \n\n");
A = (double *)mkl_malloc( m*k*sizeof( double ), 64 );
B = (double *)mkl_malloc( k*n*sizeof( double ), 64 );
C = (double *)mkl_malloc( m*n*sizeof( double ), 64 );
if (A == NULL || B == NULL || C == NULL) {
    printf( "\n ERROR: Can't allocate memory for matrices. Aborting... \n\
↪n");
    mkl_free(A);
    mkl_free(B);
    mkl_free(C);
    return 1;
}

printf (" Intializing matrix data \n\n");
for (i = 0; i < (m*k); i++) {
    A[i] = (double) (i+1);
}

for (i = 0; i < (k*n); i++) {
    B[i] = (double) (-i-1);
}

for (i = 0; i < (m*n); i++) {
    C[i] = 0.0;
}

printf (" Computing matrix product using Intel(R) MKL dgemm function via_
↪CBLAS interface \n\n");
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            m, n, k, alpha, A, k, B, n, beta, C, n);
printf ("\n Computations completed.\n\n");

printf (" Top left corner of matrix A: \n");
for (i=0; i<min(m,6); i++) {
    for (j=0; j<min(k,6); j++) {
        printf ("%12.0f", A[j+i*k]);
    }
    printf ("\n");
}

printf ("\n Top left corner of matrix B: \n");
for (i=0; i<min(k,6); i++) {
    for (j=0; j<min(n,6); j++) {
        printf ("%12.0f", B[j+i*n]);
    }
    printf ("\n");
}

```

(续下页)

(接上页)

```

printf ("\n Top left corner of matrix C: \n");
for (i=0; i<min(m,6); i++) {
    for (j=0; j<min(n,6); j++) {
        printf ("%12.5G", C[j+i*n]);
    }
    printf ("\n");
}

printf ("\n Deallocating memory \n\n");
mkl_free(A);
mkl_free(B);
mkl_free(C);

printf (" Example completed. \n\n");
return 0;
}

```

3. 在该目录下创建如下作业提交脚本 intelmklttest.slurm:

```

#!/bin/bash

#SBATCH --job-name=intelmklttest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load intel-oneapi-compilers/2021.4.0
module load intel-mkl/2020.4.304

icc intelmklttest.c -o intelmklttest -qmkl

./intelmklttest

```

4. 使用如下命令提交作业:

```

sbatch intelmklttest.slurm

```

5. 作业完成后在.out 文件中可看到如下结果:

```

This example computes real matrix C=alpha*A*B+beta*C using
Intel(R) MKL function dgemm, where A, B, and C are matrices and

```

(续下页)

(接上页)

alpha and beta are double precision scalars

Initializing data for matrix multiplication C=A*B for matrix
A(2000x200) and matrix B(200x1000)

Allocating memory for matrices aligned on 64-byte boundary for better
performance

Intializing matrix data

Computing matrix product using Intel(R) MKL dgemm function via CBLAS
→interface

Computations completed.

Top left corner of matrix A:

1	2	3	4	5	6
201	202	203	204	205	206
401	402	403	404	405	406
601	602	603	604	605	606
801	802	803	804	805	806
1001	1002	1003	1004	1005	1006

Top left corner of matrix B:

-1	-2	-3	-4	-5	-6
-1001	-1002	-1003	-1004	-1005	-1006
-2001	-2002	-2003	-2004	-2005	-2006
-3001	-3002	-3003	-3004	-3005	-3006
-4001	-4002	-4003	-4004	-4005	-4006
-5001	-5002	-5003	-5004	-5005	-5006

Top left corner of matrix C:

-2.6666E+09	-2.6666E+09	-2.6667E+09	-2.6667E+09	-2.6667E+09	-2.6667E+09
-6.6467E+09	-6.6467E+09	-6.6468E+09	-6.6468E+09	-6.6469E+09	-6.647E+09
-1.0627E+10	-1.0627E+10	-1.0627E+10	-1.0627E+10	-1.0627E+10	-1.0627E+10
-1.4607E+10	-1.4607E+10	-1.4607E+10	-1.4607E+10	-1.4607E+10	-1.4607E+10
-1.8587E+10	-1.8587E+10	-1.8587E+10	-1.8587E+10	-1.8588E+10	-1.8588E+10
-2.2567E+10	-2.2567E+10	-2.2567E+10	-2.2567E+10	-2.2568E+10	-2.2568E+10

Deallocating memory

Example completed.

pi2.0 上的 Intel mkl

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下创建如下作业提交脚本 `intelmkltest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=intelmkltest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load intel-oneapi-compilers/2021.4.0
module load intel-mkl/2019.3.199

icc intelmkltest.c -o intelmkltest -qmkl

./intelmkltest
```

4. 使用如下命令提交作业:

```
sbatch intelmkltest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
This example computes real matrix C=alpha*A*B+beta*C using
Intel(R) MKL function dgemm, where A, B, and C are matrices and
alpha and beta are double precision scalars

Initializing data for matrix multiplication C=A*B for matrix
A(2000x200) and matrix B(200x1000)

Allocating memory for matrices aligned on 64-byte boundary for better
performance

Intializing matrix data

Computing matrix product using Intel(R) MKL dgemm function via CBLAS
->interface

Computations completed.
```

(续下页)

(接上页)

```

Top left corner of matrix A:
      1      2      3      4      5      6
    201    202    203    204    205    206
    401    402    403    404    405    406
    601    602    603    604    605    606
    801    802    803    804    805    806
   1001   1002   1003   1004   1005   1006

Top left corner of matrix B:
     -1     -2     -3     -4     -5     -6
   -1001   -1002   -1003   -1004   -1005   -1006
   -2001   -2002   -2003   -2004   -2005   -2006
   -3001   -3002   -3003   -3004   -3005   -3006
   -4001   -4002   -4003   -4004   -4005   -4006
   -5001   -5002   -5003   -5004   -5005   -5006

Top left corner of matrix C:
-2.6666E+09 -2.6666E+09 -2.6667E+09 -2.6667E+09 -2.6667E+09 -2.6667E+09
-6.6467E+09 -6.6467E+09 -6.6468E+09 -6.6468E+09 -6.6469E+09 -6.647E+09
-1.0627E+10 -1.0627E+10 -1.0627E+10 -1.0627E+10 -1.0627E+10 -1.0627E+10
-1.4607E+10 -1.4607E+10 -1.4607E+10 -1.4607E+10 -1.4607E+10 -1.4607E+10
-1.8587E+10 -1.8587E+10 -1.8587E+10 -1.8587E+10 -1.8588E+10 -1.8588E+10
-2.2567E+10 -2.2567E+10 -2.2567E+10 -2.2567E+10 -2.2568E+10 -2.2568E+10

Deallocating memory

Example completed.

```

参考资料

- [Intel mkl 官网教程](#)

Intel-MPI

英特尔® MPI 库是一个实现开源 MPICH 规范的多结构消息传递库。使用该库创建、维护和测试高级、复杂的应用程序，这些应用程序在基于英特尔® 处理器的高性能计算 (HPC) 集群上表现更好。

加载预安装的 **Intel** 组件

可以用以下命令加载集群中已安装的 **Intel** 组件:

版本	加载方式	组件说明
intel-mpi-2021.4.0	module load intel-oneapi-mpi/2021.4.0 module load intel-oneapi-compilers/2021.4.0	Intel MPI 库
intel-mpi-2019.4.243	module load intel-mpi/2019.4.243	Intel MPI 库
intel-mpi-2019.6.154	module load intel-mpi/2019.6.154	Intel MPI 库

在使用 intel-mpi 的时候, 请尽量保持编译器版本与后缀中的编译器版本一致, 如 intel-mpi-2019.4.243/intel-19.0.4 和 intel-19.0.4 另外我们建议直接使用 **Intel** 全家桶

使用 **Intel+Intel-mpi** 编译应用

这里, 我们演示如何使用系统中的 **Intel** 和 **Intel-mpi** 编译 **MPI** 代码, 使用的 **MPI** 代码可以在 /lustre/share/samples/MPI/mpihello.c 中找到。

加载和编译:

```
$ module load intel-parallel-studio/cluster.2019.5-intel-19.0.5
$ mpiicc mpihello.c -o mpihello
```

提交 **Intel+Intel-mpi** 应用

准备一个 mpihello.c 程序

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[])
{
    int myid, numprocs;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Get_processor_name(processor_name, &namelen);
    printf("Hello World! Process %d of %d on %s\n", myid, numprocs, processor_
↵name);
    MPI_Finalize();
}
```

(续下页)

(接上页)

```
return 0;
}
```

准备一个名为 `job_impi.slurm` 的作业脚本

```
#!/bin/bash

#SBATCH --job-name=mpihello
#SBATCH --partition=small
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 2

ulimit -s unlimited
ulimit -l unlimited

module load intel-parallel-studio/cluster.2019.5-intel-19.0.5

mpiicc mpihello.c -o mpihello

mpirun -np 2 ./mpihello
```

若采用 intel 2018, 脚本中 `export I_MPI_FABRICS=shm:ofi` 这行需改为 `export I_MPI_FABRICS=shm:tmi`

最后, 将作业提交到 SLURM

```
$ sbatch job_impi.slurm
```

参考资料

- [intel-parallel-studio](#)

Intel-Vtune

英特尔 VTune 性能分析器可通过图形用户界面轻松定位应用运行的性能瓶颈。

可用版本

版本	加载方式
2021.7.1	module load intel-oneapi-vtune/2021.7.1 思源一号

本文使用的算例

lammps的in.lj数据:

可以在如下链接中得到文件in.lj的内容。

<https://docs.hpc.sjtu.edu.cn/app/engineeringscience/lammps.html>

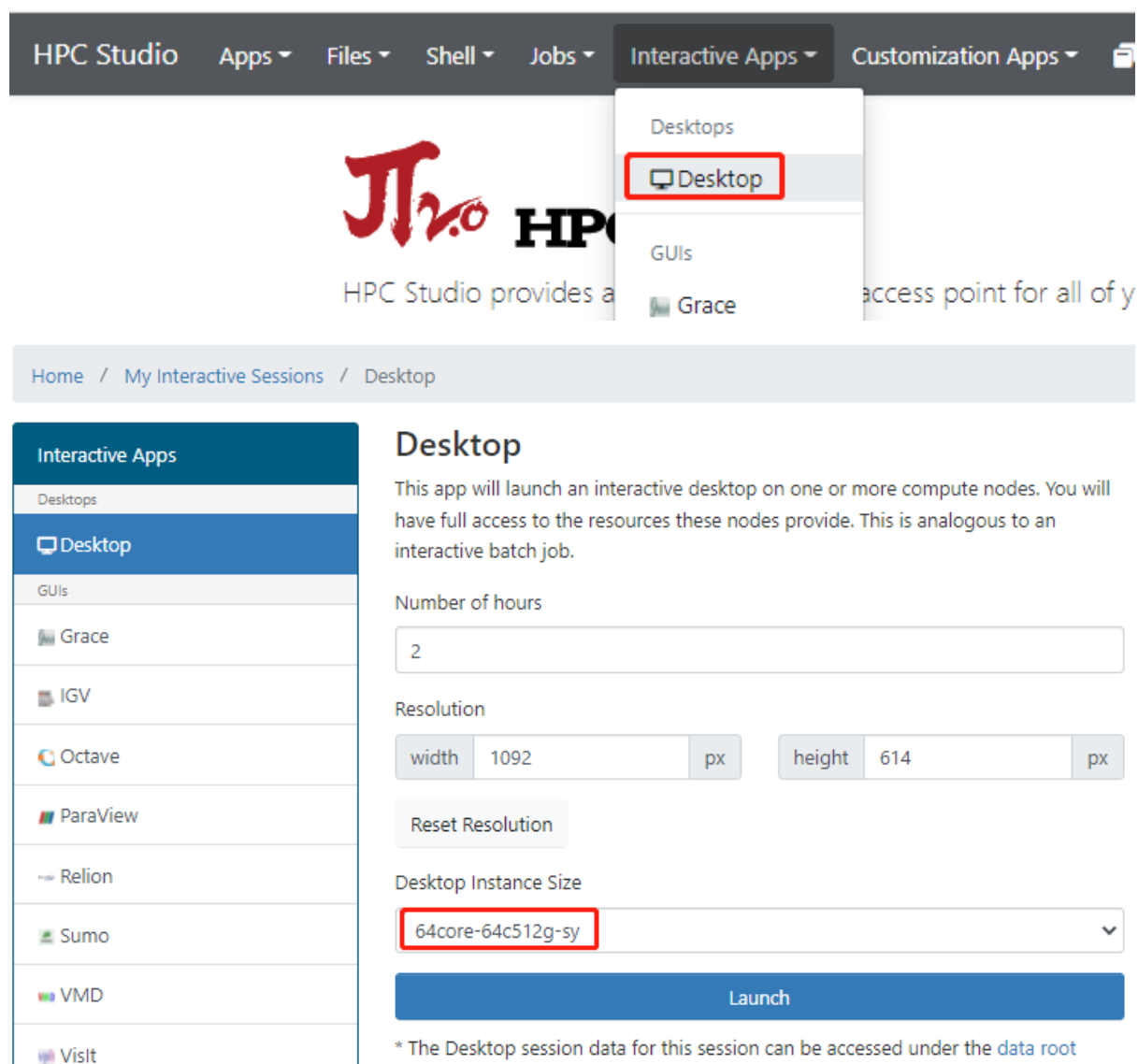
集群上的 VTUNE

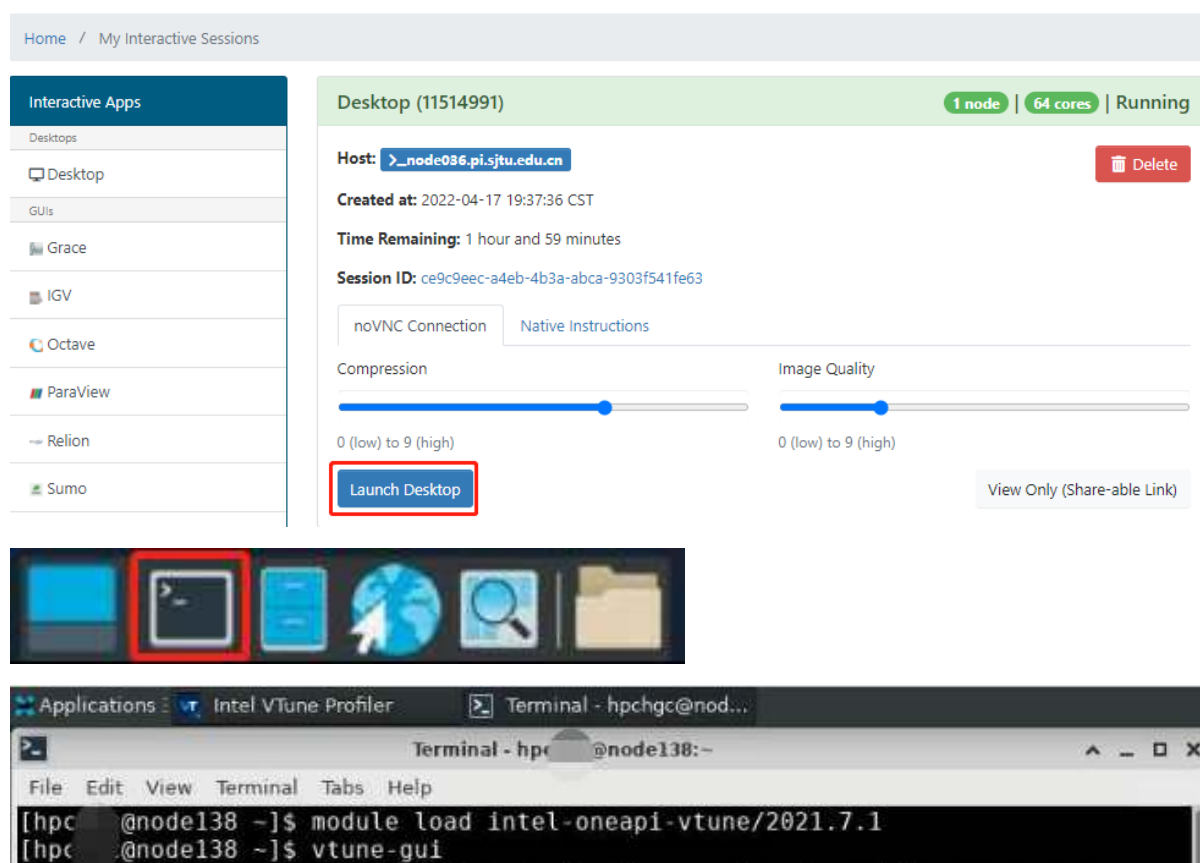
- 思源一号上的 VTUNE

思源一号上的 VTUNE

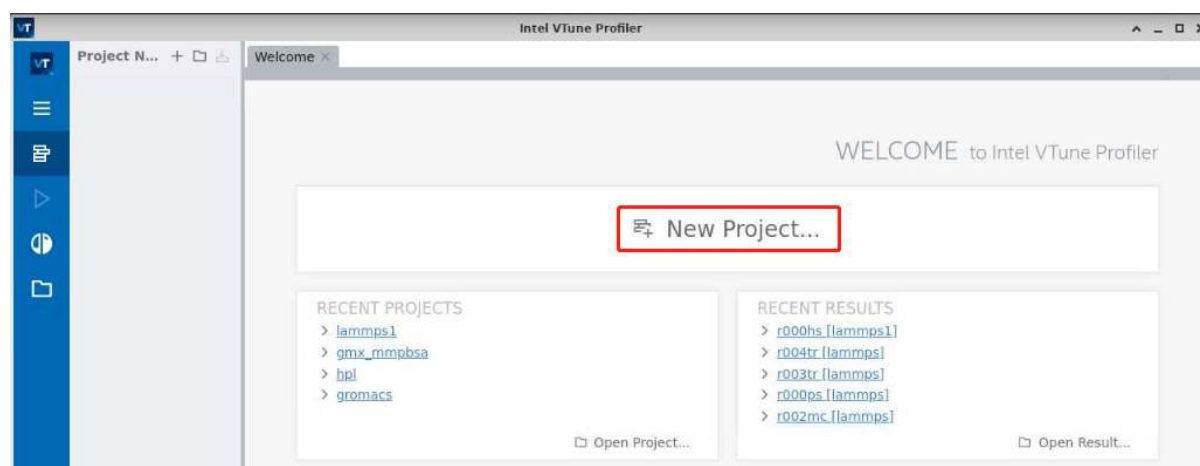
登录 hpc studio

调出 VTUNE 图形化界面





导入应用程序的流程



Create a Project

Project name:

Location:

Create Project

Cancel

Project N... +

▼ lammps

Welcome x

Configure Analysis x

Configure Analysis

WHERE

Local Host ▼

WHAT

Launch Application ▼

Specify and configure your analysis target: an application or a script to execute.

No application executable (target) file specified.

Retry

Application:

Application parameters:

Cancel

Select File

Recent

Home

Desktop

+ Other Locations

hpchgc lammps

Name	Size	Modified
alphafold		3 Mar
data		3 Mar
in.lj.txt	472 bytes	14 Mar 15:55
lammps		
run.sh	173 bytes	15 Mar
run.slurm	355 bytes	19 Mar 15:57
tst		15:57

查找运行程序的热点

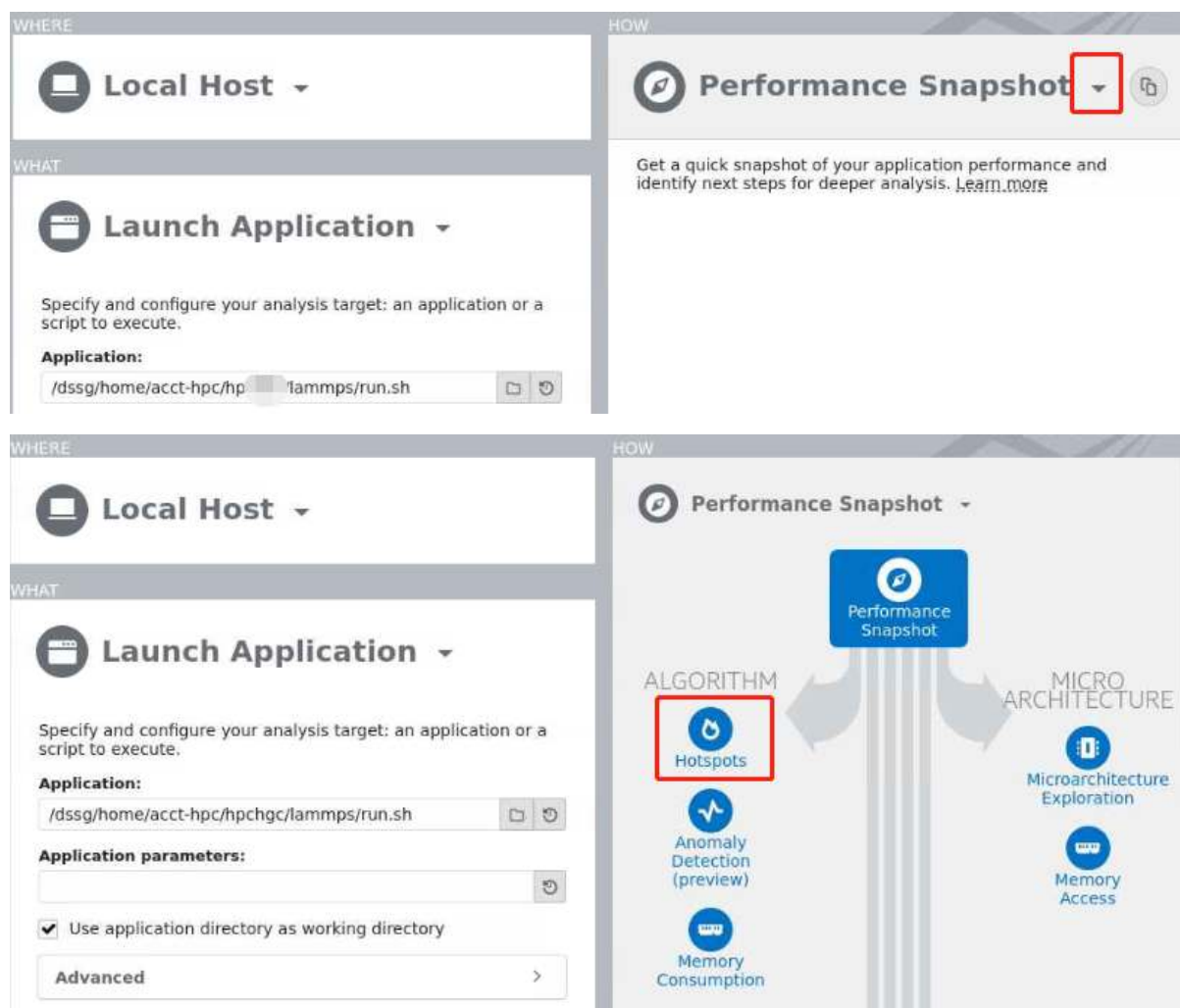
执行程序的路径与脚本内容

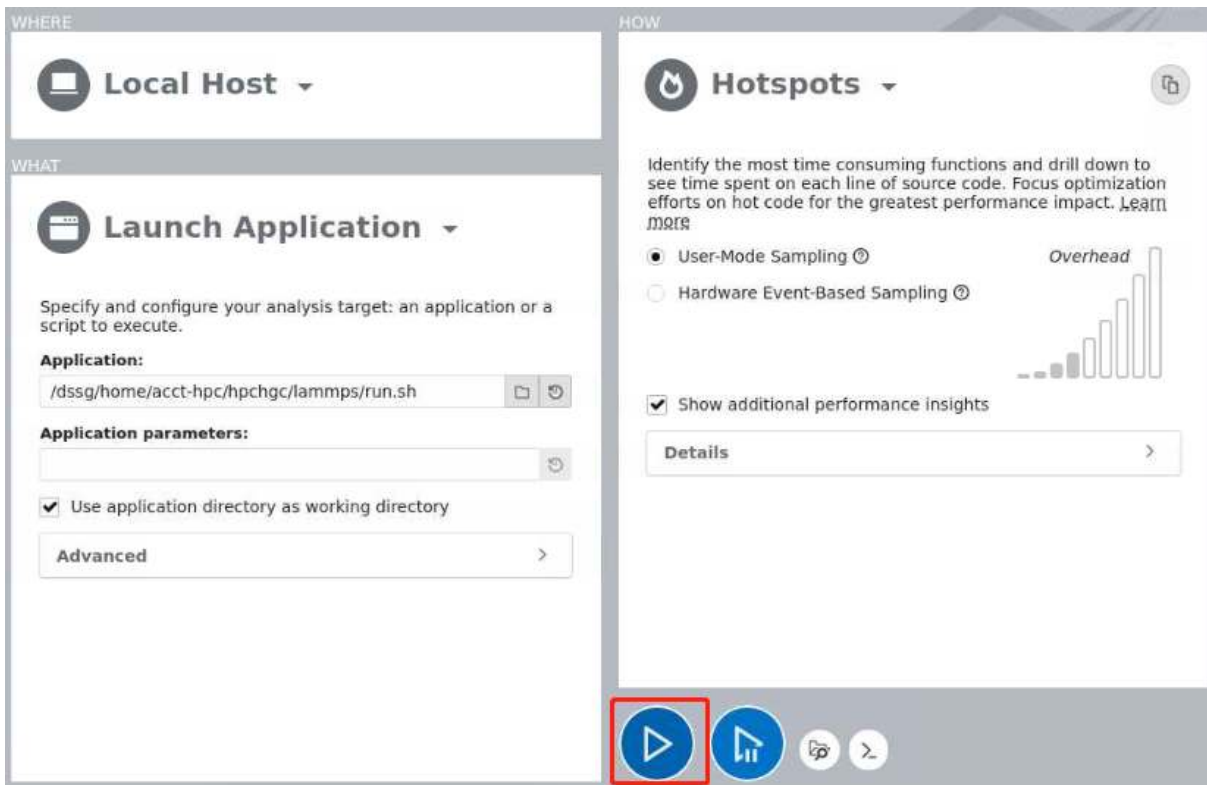
```
[hpc@node243 test]$ tree lammmps/  
lammmps/  
├── in.lj  
└── run.sh
```

run.sh

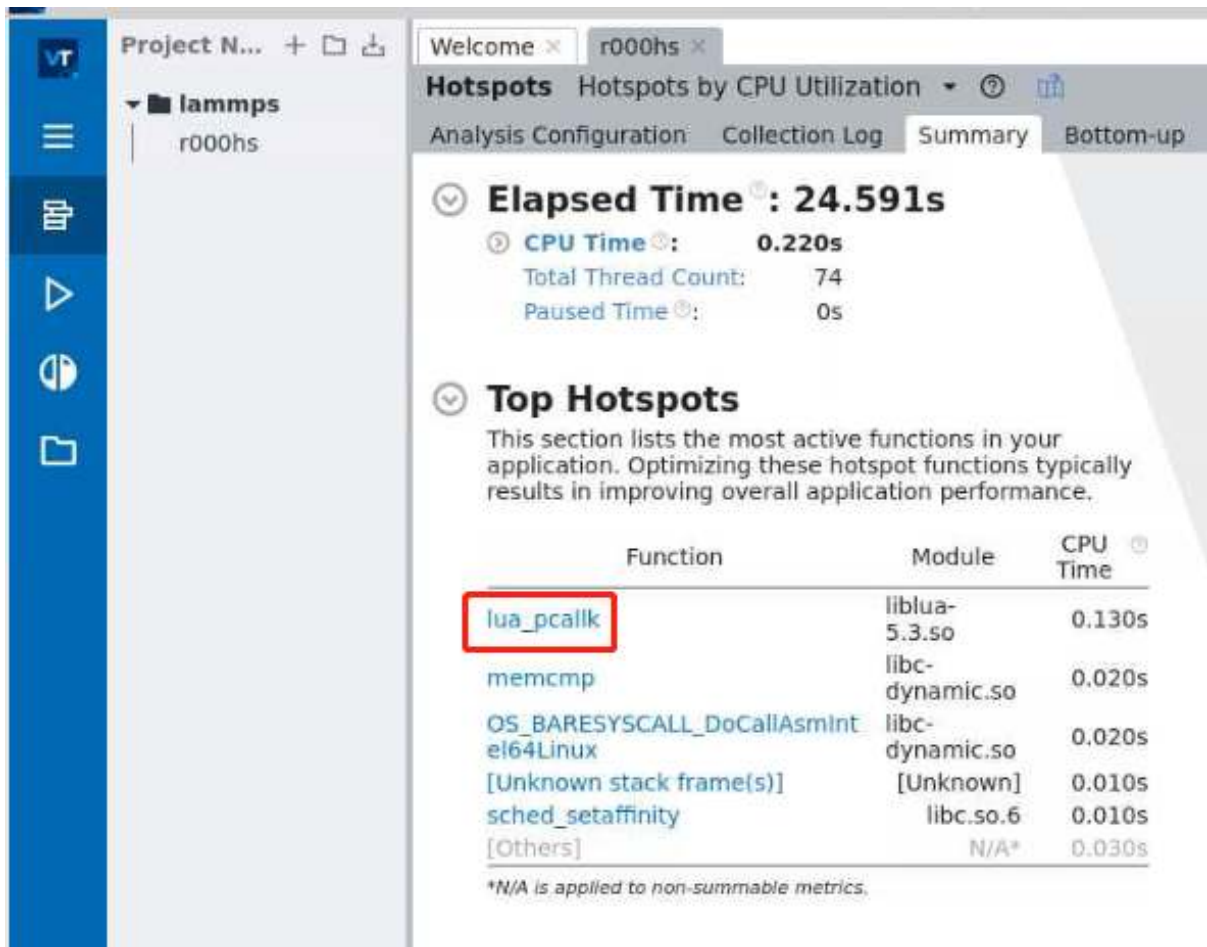
```
#!/bin/bash  
module load oneapi  
module load lammmps/20210310-intel-2021.4.0  
  
ulimit -s unlimited  
ulimit -l unlimited  
export OMP_NUM_THREADS=1  
mpirun -np 32 lmp -i in.lj.txt
```

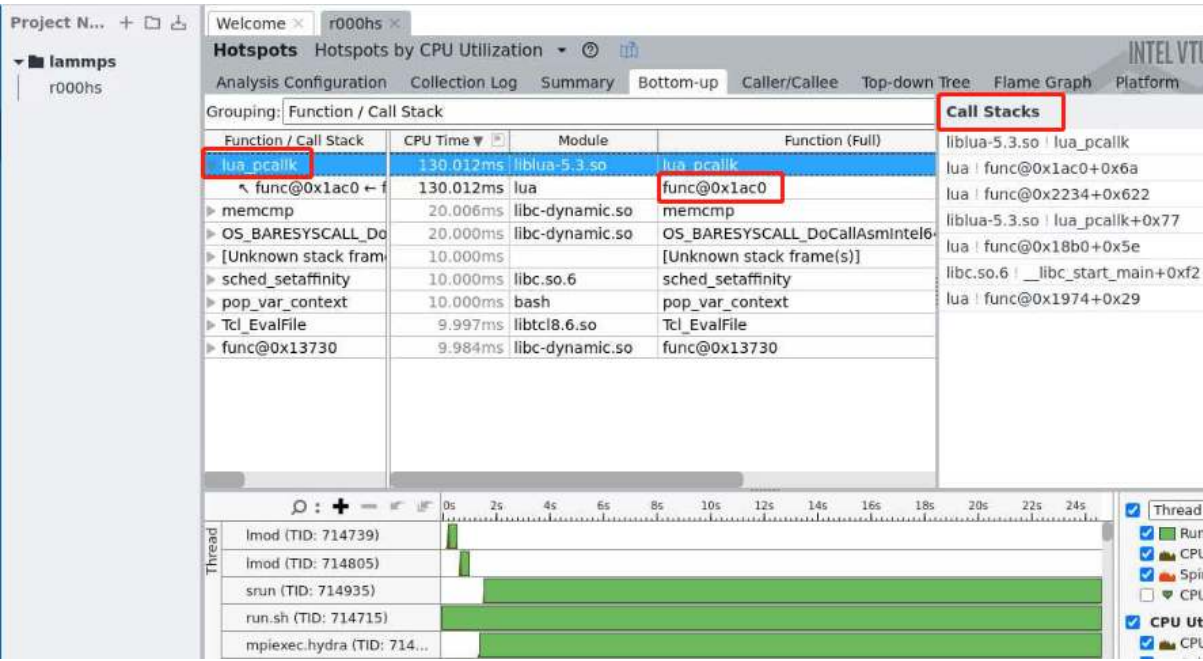
打开分析热点





运行结果





参考资料

- [intel-parallel-studio](#)

cuBLAS

简介

cuBLAS 库可提供基本线性代数子程序 (BLAS) 的 GPU 加速实现。cuBLAS 利用针对 NVIDIA GPU 高度优化的插入式行业标准 BLAS API，加速 AI 和 HPC 应用。cuBLAS 库包含用于批量运算、跨多个 GPU 的执行以及混合精度和低精度执行的扩展程序。通过使用 cuBLAS，应用将能自动从定期性能提升及新的 GPU 体系架构中受益。cuBLAS 库包含在 NVIDIA HPC SDK 和 CUDA 工具包中。

cuBLAS 使用说明

思源一号上的 cuBLAS

1. 先创建一个目录 cuBLASest 并进入该目录：

```
mkdir cuBLASest
cd cuBLASest
```

2. 在该目录下创建如下测试文件 cuBLASest.cu：

```
// 矩阵乘法示例
#include <stdio>
#include <stdlib>
#include <vector>
```

(续下页)

(接上页)

```

#include <cublas_v2.h>
#include <cuda_runtime.h>

template <typename T>
void print_matrix(const int &m, const int &n, const T *A, const int &lda);
template <>
void print_matrix(const int &m, const int &n, const double *A, const int &
→lda)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            std::printf("%0.2f ", A[j * lda + i]);
        }
        std::printf("\n");
    }
}

using data_type = double;

int main(int argc, char *argv[])
{
    cublasHandle_t cublasH = NULL;
    cudaStream_t stream = NULL;

    const int m = 2;
    const int n = 2;
    const int k = 2;
    const int lda = 2;
    const int ldb = 2;
    const int ldc = 2;
    /*
    *   A = | 1.0 | 2.0 |
    *         | 3.0 | 4.0 |
    *
    *   B = | 5.0 | 6.0 |
    *         | 7.0 | 8.0 |
    */

    const std::vector<data_type> A = {1.0, 2.0, 3.0, 4.0};
    const std::vector<data_type> B = {5.0, 6.0, 7.0, 8.0};
    std::vector<data_type> C(m * n);
    const data_type alpha = 1.0;
    const data_type beta = 0.0;

    data_type *d_A = nullptr;
    data_type *d_B = nullptr;
    data_type *d_C = nullptr;

```

(续下页)

(接上页)

```

cublasOperation_t transa = CUBLAS_OP_N;
cublasOperation_t transb = CUBLAS_OP_N;

printf("A\n");
print_matrix(m, k, A.data(), lda);
printf("====\n");

printf("B\n");
print_matrix(k, n, B.data(), ldb);
printf("====\n");

/* step 1: create cublas handle, bind a stream */
(cublasCreate(&cublasH));

(cudaStreamCreateWithFlags(&stream, cudaStreamNonBlocking));
(cublasSetStream(cublasH, stream));

/* step 2: copy data to device */
(cudaMalloc(reinterpret_cast<void **>(&d_A), sizeof(data_type) * A.
↪size()));
(cudaMalloc(reinterpret_cast<void **>(&d_B), sizeof(data_type) * B.
↪size()));
(cudaMalloc(reinterpret_cast<void **>(&d_C), sizeof(data_type) * C.
↪size()));

(cudaMemcpyAsync(d_A, A.data(), sizeof(data_type) * A.size(),
↪cudaMemcpyHostToDevice, stream));
(cudaMemcpyAsync(d_B, B.data(), sizeof(data_type) * B.size(),
↪cudaMemcpyHostToDevice, stream));

/* step 3: compute */
(cublasDgemm(cublasH, transa, transb, m, n, k, &alpha, d_A, lda, d_B,
↪ldb, &beta, d_C, ldc));

/* step 4: copy data to host */
(cudaMemcpyAsync(C.data(), d_C, sizeof(data_type) * C.size(),
↪cudaMemcpyDeviceToHost, stream));

(cudaStreamSynchronize(stream));

/*
*   C = | 23.0 | 31.0 |
*         | 34.0 | 46.0 |
*/

printf("C\n");
print_matrix(m, n, C.data(), ldc);
printf("====\n");

```

(续下页)

(接上页)

```

/* free resources */
(cudaFree(d_A));
(cudaFree(d_B));
(cudaFree(d_C));

(cublasDestroy(cublasH));
(cudaStreamDestroy(stream));
(cudaDeviceReset());

return EXIT_SUCCESS;
}

```

3. 在该目录下创建如下作业提交脚本 cuBLASest.slurm:

```

#!/bin/bash

#SBATCH --job-name=cuBLASest      # 作业名
#SBATCH --partition=a100         # a100 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1        # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1             # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.5.0
module load gcc/11.2.0

nvcc cuBLASest.cu -o cuBLASest -lcublas
./cuBLASest

```

4. 使用如下命令提交作业:

```
sbatch cuBLASest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```

A
1.00 3.00
2.00 4.00
=====
B
5.00 7.00
6.00 8.00
=====
C
23.00 31.00
34.00 46.00
=====

```

pi2.0 上的 cuBLAS

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下创建如下作业提交脚本 `cuBLASest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=cuBLASest           # 作业名
#SBATCH --partition=dgx2               # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1              # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                  # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.6.2-gcc-8.3.0
module load gcc/8.3.0

nvcc cuBLASest.cu -o cuBLASest -lcublas
./cuBLASest
```

4. 使用如下命令提交作业:

```
sbatch cuBLASest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
A
1.00 3.00
2.00 4.00
=====
B
5.00 7.00
6.00 8.00
=====
C
23.00 31.00
34.00 46.00
=====
```

参考资料

- [cuBLAS 官网](#)
- [cuBLAS github](#)

CUDA

思源一号平台及 AI 平台上均部署有 CUDA 套件。

版本	加载方式
10.1.243	module load cuda/10.1.243 思源一号
11.3.1	module load cuda/11.3.1 思源一号
11.4.0	module load cuda/11.4.0 思源一号
11.5.0	module load cuda/11.5.0 思源一号
11.8.0	module load cuda/11.8.0 思源一号
8.0.61	module load cuda/8.0.61-gcc-4.8.5
9.0.176	module load cuda/9.0.176-gcc-4.8.5
9.2.88	module load cuda/9.2.88-gcc-4.8.5
10.0.130	module load cuda/10.0.130-gcc-4.8.5
10.1.243	module load cuda/10.1.243-gcc-4.8.5

本文档向您展示如何使用 CUDA，包含程序示例，编译，作业脚本示例。

程序示例 CuBLAS

编辑 cublas.cu 文件，内容如下：

```
//Example. Application Using C and CUBLAS: 1-based indexing
//-----
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"
#define M 6
#define N 5
#define IDX2F(i,j,ld) (((j)-1)*(ld))+((i)-1))

static __inline__ void modify (cublasHandle_t handle, float *m, int ldm,
↪int n, int p, int q, float alpha, float beta){
    cublasSscal (handle, n-q+1, &alpha, &m[IDX2F(p,q,ldm)], ldm);
    cublasSscal (handle, ldm-p+1, &beta, &m[IDX2F(p,q,ldm)], 1);
}

int main (void){
    cudaError_t cudaStat;
    cublasStatus_t stat;
    cublasHandle_t handle;
    int i, j;
    float* devPtrA;
    float* a = 0;
    a = (float *)malloc (M * N * sizeof (*a));
    if (!a) {
```

(续下页)

(接上页)

```

    printf ("host memory allocation failed");
    return EXIT_FAILURE;
}
for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++) {
        a[IDX2F(i,j,M)] = (float)((i-1) * M + j);
    }
}
cudaStat = cudaMalloc ((void**)&devPtrA, M*N*sizeof(*a));
if (cudaStat != cudaSuccess) {
    printf ("device memory allocation failed");
    return EXIT_FAILURE;
}
stat = cublasCreate(&handle);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("CUBLAS initialization failed\n");
    return EXIT_FAILURE;
}
stat = cublasSetMatrix (M, N, sizeof(*a), a, M, devPtrA, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("data download failed");
    cudaFree (devPtrA);
    cublasDestroy(handle);
    return EXIT_FAILURE;
}
modify (handle, devPtrA, M, N, 2, 3, 16.0f, 12.0f);
stat = cublasGetMatrix (M, N, sizeof(*a), devPtrA, M, a, M);
if (stat != CUBLAS_STATUS_SUCCESS) {
    printf ("data upload failed");
    cudaFree (devPtrA);
    cublasDestroy(handle);
    return EXIT_FAILURE;
}
cudaFree (devPtrA);
cublasDestroy(handle);
for (j = 1; j <= N; j++) {
    for (i = 1; i <= M; i++) {
        printf ("%7.0f", a[IDX2F(i,j,M)]);
    }
    printf ("\n");
}
free(a);
return EXIT_SUCCESS;
}

```

将以上程序保存为 *cublas.cu*。

使用 **cuda** 进行编译，编译时链接 **cublas** 动态库。

在思源平台上的编译命令如下：


```
$ module load cuda/11.3.1
$ nvcc cublas.cu -o cublas -lcublas
```

在 AI 平台上的编译命令如下:

```
$ module load cuda/10.0.130-gcc-4.8.5
$ nvcc cublas.cu -o cublas -lcublas
```

a100 队列作业脚本示例

这是一个名为 `a100.slurm` 的 单机单卡作业脚本, 该脚本向 **a100** 队列申请 1 块 GPU, 并在作业完成时通知。

```
#!/bin/bash

#SBATCH --job-name=cuda_test
#SBATCH --partition=a100
#SBATCH --gres=gpu:1
#SBATCH -N 1
#SBATCH --ntasks-per-node 1
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=cublas.out
#SBATCH --error=cublas.err

module load cuda/11.3.1

./cublas
```

用以下方式提交作业:

```
$ sbatch a100.slurm
```

DGX2 队列作业脚本示例

这是一个名为 `dgx.slurm` 的 单机单卡作业脚本, 该脚本向 **dgx2** 队列申请 1 块 GPU, 并在作业完成时通知。

```
#!/bin/bash

#SBATCH --job-name=dgx2_test
#SBATCH --partition=dgx2
#SBATCH --gres=gpu:1
#SBATCH -N 1
#SBATCH --ntasks-per-node 1
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=cublas.out
```

(续下页)

(接上页)

```
#SBATCH --error=cublas.err

module load cuda/10.0.130-gcc-4.8.5

./cublas
```

用以下方式提交作业：

```
$ sbatch dgx.slurm
```

预期结果

```
$ cat cublas.out
      1      7      13      19      25      31
      2      8      14      20      26      32
      3 1728     180     252     324     396
      4     160      16      22      28      34
      5     176      17      23      29      35
```

cuSPARSE

简介

cuSPARSE 库为稀疏矩阵提供经 GPU 加速的基本线性代数子程序，与仅使用 CPU 的替代方案相比，这一程序的执行速度有显著的提升。此库提供了用于构建 GPU 加速型求解器的功能。在从事机器学习、计算流体力学、地震勘探及计算科学等应用的工程师和科学家群体中，cuSPARSE 得到了广泛采用。通过使用 cuSPARSE，应用将能自动从定期性能提升及全新 GPU 架构中受益。cuSPARSE 库包含在 NVIDIA HPC SDK 和 CUDA 工具包中。

cuSPARSE 使用说明

思源一号上的 cuSPARSE

1. 先创建一个目录 cuSPARSEtest 并进入该目录：

```
mkdir cuSPARSEtest
cd cuSPARSEtest
```

2. 在该目录下创建如下测试文件 cuSPARSEtest.cu：

```
// 求解下三角稀疏线性方程组 Ax=b

#include <cuda_runtime_api.h> // cudaMalloc, cudaMemcpy, etc.
#include <cusparse.h>         // cusparseSpSV
#include <stdio.h>             // printf
#include <stdlib.h>            // EXIT_FAILURE
```

(续下页)

(接上页)

```

#define CHECK_CUDA(func) \
{ \
    cudaError_t status = (func); \
    if (status != cudaSuccess) \
    { \
        printf("CUDA API failed at line %d with error: %s (%d)\n", \
            __LINE__, cudaGetErrorString(status), status); \
        return EXIT_FAILURE; \
    } \
}

#define CHECK_CUSPARSE(func) \
{ \
    cusparseStatus_t status = (func); \
    if (status != CUSPARSE_STATUS_SUCCESS) \
    { \
        printf("CUSPARSE API failed at line %d with error: %s (%d)\n", \
            __LINE__, cusparseGetErrorString(status), status); \
        return EXIT_FAILURE; \
    } \
}

int main(void)
{
    // A = [1, 0, 0, 0
    //       0, 4, 0, 0
    //       5, 0, 6, 0
    //       0, 8, 0, 9]

    // b = [1
    //      8
    //      23
    //      52]

    // Host problem definition
    const int A_num_rows = 4;
    const int A_num_cols = 4;
    const int A_nnz = 6;
    int hA_csrOffsets[] = {0, 1, 2, 4, 6};
    int hA_columns[] = {0, 1, 0, 2, 1, 3};
    float hA_values[] = {1.0f, 4.0f, 5.0f, 6.0f, 8.0f, 9.0f};
    float hX[] = {1.0f, 8.0f, 23.0f, 52.0f};
    float hY[] = {0.0f, 0.0f, 0.0f, 0.0f};
    float hY_result[] = {1.0f, 2.0f, 3.0f, 4.0f};
    float alpha = 1.0f;
    //-----
    ↪-----
    // Device memory management

```

(续下页)

(接上页)

```

int *dA_csrOffsets, *dA_columns;
float *dA_values, *dX, *dY;
CHECK_CUDA(cudaMalloc((void **)&dA_csrOffsets, (A_num_rows + 1) *
↪ sizeof(int)))
CHECK_CUDA(cudaMalloc((void **)&dA_columns, A_nnz * sizeof(int)))
CHECK_CUDA(cudaMalloc((void **)&dA_values, A_nnz * sizeof(float)))
CHECK_CUDA(cudaMalloc((void **)&dX, A_num_cols * sizeof(float)))
CHECK_CUDA(cudaMalloc((void **)&dY, A_num_rows * sizeof(float)))

CHECK_CUDA(cudaMemcpy(dA_csrOffsets, hA_csrOffsets, (A_num_rows + 1) *
↪ sizeof(int), cudaMemcpyHostToDevice))
CHECK_CUDA(cudaMemcpy(dA_columns, hA_columns, A_nnz * sizeof(int),
↪ cudaMemcpyHostToDevice))
CHECK_CUDA(cudaMemcpy(dA_values, hA_values, A_nnz * sizeof(float),
↪ cudaMemcpyHostToDevice))
CHECK_CUDA(cudaMemcpy(dX, hX, A_num_cols * sizeof(float),
↪ cudaMemcpyHostToDevice))
CHECK_CUDA(cudaMemcpy(dY, hY, A_num_rows * sizeof(float),
↪ cudaMemcpyHostToDevice))
//-----
↪-----
// CUSPARSE APIs
cusparsHandle_t handle = NULL;
cusparsSpMatDescr_t matA;
cusparsDnVecDescr_t vecX, vecY;
void *dBuffer = NULL;
size_t bufferSize = 0;
cusparsSpSVDescr_t spsvDescr;
CHECK_CUSPARSE(cusparsCreate(&handle))
// Create sparse matrix A in CSR format
CHECK_CUSPARSE(cusparsCreateCsr(&matA, A_num_rows, A_num_cols, A_nnz,
                                dA_csrOffsets, dA_columns, dA_values,
                                CUSPARSE_INDEX_32I, CUSPARSE_INDEX_32I,
                                CUSPARSE_INDEX_BASE_ZERO, CUDA_R_32F))

// Create dense vector X
CHECK_CUSPARSE(cusparsCreateDnVec(&vecX, A_num_cols, dX, CUDA_R_32F))
// Create dense vector y
CHECK_CUSPARSE(cusparsCreateDnVec(&vecY, A_num_rows, dY, CUDA_R_32F))
// Create opaque data structure, that holds analysis data between calls.
CHECK_CUSPARSE(cusparsSpSV_createDescr(&spsvDescr))
// Specify Lower|Upper fill mode.
cusparsFillMode_t fillmode = CUSPARSE_FILL_MODE_LOWER;
CHECK_CUSPARSE(cusparsSpMatSetAttribute(matA, CUSPARSE_SPMAT_FILL_MODE,
↪ &fillmode, sizeof(fillmode)))
// Specify Unit|Non-Unit diagonal type.
cusparsDiagType_t diagtype = CUSPARSE_DIAG_TYPE_NON_UNIT;
CHECK_CUSPARSE(cusparsSpMatSetAttribute(matA, CUSPARSE_SPMAT_DIAG_TYPE,
↪ &diagtype, sizeof(diagtype)))
// allocate an external buffer for analysis

```

(续下页)

(接上页)

```

CHECK_CUSPARSE(cusparsesSpSV_bufferSize(handle, CUSPARSE_OPERATION_NON_
↪TRANSPPOSE,
                                &alpha, matA, vecX, vecY, CUDA_R_
↪32F,
                                CUSPARSE_SPSV_ALG_DEFAULT, ↪
↪spsvDescr,
                                &bufferSize))
CHECK_CUDA(cudaMalloc(&dBuffer, bufferSize))
CHECK_CUSPARSE(cusparsesSpSV_analysis(handle, CUSPARSE_OPERATION_NON_
↪TRANSPPOSE,
                                &alpha, matA, vecX, vecY, CUDA_R_
↪32F,
                                CUSPARSE_SPSV_ALG_DEFAULT, ↪
↪spsvDescr, dBuffer))
// execute SpSV
CHECK_CUSPARSE(cusparsesSpSV_solve(handle, CUSPARSE_OPERATION_NON_
↪TRANSPPOSE,
                                &alpha, matA, vecX, vecY, CUDA_R_32F,
                                CUSPARSE_SPSV_ALG_DEFAULT, spsvDescr))

// destroy matrix/vector descriptors
CHECK_CUSPARSE(cusparsesDestroySpMat(matA))
CHECK_CUSPARSE(cusparsesDestroyDnVec(vecX))
CHECK_CUSPARSE(cusparsesDestroyDnVec(vecY))
CHECK_CUSPARSE(cusparsesSpSV_destroyDescr(spsvDescr));
CHECK_CUSPARSE(cusparsesDestroy(handle))
//-----
↪-----
// device result check
CHECK_CUDA(cudaMemcpy(hY, dY, A_num_rows * sizeof(float), ↪
↪cudaMemcpyDeviceToHost))
int correct = 1;
for (int i = 0; i < A_num_rows; i++)
{
    if (hY[i] != hY_result[i])
    {
        // direct floating point comparison is not
        correct = 0; // reliable
        break;
    }
}
if (correct)
    printf("spsv_csr_example test PASSED\n");
else
    printf("spsv_csr_example test FAILED: wrong result\n");
for (size_t i = 0; i < A_num_rows; i++)
{
    printf("x[%d] = %f\n", i, hY[i]);
}
//-----

```

(续下页)

(接上页)

```

→-----
    // device memory deallocation
    CHECK_CUDA(cudaFree(dBuffer))
    CHECK_CUDA(cudaFree(dA_csrOffsets))
    CHECK_CUDA(cudaFree(dA_columns))
    CHECK_CUDA(cudaFree(dA_values))
    CHECK_CUDA(cudaFree(dX))
    CHECK_CUDA(cudaFree(dY))
    return EXIT_SUCCESS;
}

```

3. 在该目录下创建如下作业提交脚本 `cuSPARSEtest.slurm`:

```

#!/bin/bash

#SBATCH --job-name=cuSPARSEtest      # 作业名
#SBATCH --partition=a100             # a100 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1            # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                 # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.5.0
module load gcc/11.2.0

nvcc cuSPARSEtest.cu -o cuSPARSEtest -lcusparse
./cuSPARSEtest

```

4. 使用如下命令提交作业:

```

sbatch cuSPARSEtest.slurm

```

5. 作业完成后在 `.out` 文件中可看到如下结果:

```

spsv_csr_example test PASSED
x[0] = 1.000000
x[1] = 2.000000
x[2] = 3.000000
x[3] = 4.000000

```

pi2.0 上的 cuSPARSE

1. 此步骤和上文完全相同;
2. 此步骤和上文完全相同;
3. 在该目录下创建如下作业提交脚本 `cuSPARSEtest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=cuSPARSEtest      # 作业名
#SBATCH --partition=dgx2             # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1            # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                 # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.6.2-gcc-8.3.0
module load gcc/8.3.0

nvcc cuSPARSEtest.cu -o cuSPARSEtest -lcusparse
./cuSPARSEtest
```

4. 使用如下命令提交作业:

```
sbatch cuSPARSEtest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
spsv_csr_example test PASSED
x[0] = 1.000000
x[1] = 2.000000
x[2] = 3.000000
x[3] = 4.000000
```

参考资料

- [cuSPARSE 官网](#)
- [cuSPARSE github](#)

cuSOLVER

简介

The NVIDIA cuSOLVER library provides a collection of dense and sparse direct linear solvers and Eigen solvers which deliver significant acceleration for Computer Vision, CFD, Computational Chemistry, and Linear Optimization applications. The cuSOLVER library is included in both the NVIDIA HPC SDK and the CUDA Toolkit.

cuSOLVER 使用说明

思源一号上的 cuSOLVER

1. 先创建一个目录 cuSOLVERtest 并进入该目录:

```
mkdir cuSOLVERtest
cd cuSOLVERtest
```

2. 在该目录下创建如下测试文件 cuSOLVERtest.cu:

```
//利用LU分解法求解线性方程组Ax=b

#include <stdio>
#include <stdlib>
#include <vector>

#include <cuda_runtime.h>
#include <cusolverDn.h>

template <typename T>
void print_matrix(const int &m, const int &n, const T *A, const int &lda);

template <>
void print_matrix(const int &m, const int &n, const float *A, const int &
↳lda)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            std::printf("%0.2f ", A[j * lda + i]);
        }
        std::printf("\n");
    }
}

template <>
void print_matrix(const int &m, const int &n, const double *A, const int &
↳lda)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            std::printf("%0.2f ", A[j * lda + i]);
        }
        std::printf("\n");
    }
}

template <>
void print_matrix(const int &m, const int &n, const cuComplex *A, const int &
↳lda)
{
    for (int i = 0; i < m; i++)
```

(续下页)

(接上页)

```

{
    for (int j = 0; j < n; j++)
    {
        std::printf("%0.2f + %0.2fj ", A[j * lda + i].x, A[j * lda + i].
→y);
    }
    std::printf("\n");
}
}

template <>
void print_matrix(const int &m, const int &n, const cuDoubleComplex *A,
→const int &lda)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            std::printf("%0.2f + %0.2fj ", A[j * lda + i].x, A[j * lda + i].
→y);
        }
        std::printf("\n");
    }
}

int main(int argc, char *argv[])
{
    cusolverDnHandle_t cusolverH = NULL;
    cudaStream_t stream = NULL;

    const int m = 3;
    const int lda = m;
    const int ldb = m;

    /*
    *      | 1 2 3 |
    *  A = | 4 5 6 |
    *      | 7 8 10 |
    *
    * without pivoting: A = L*U
    *      | 1 0 0 |      | 1 2 3 |
    *  L = | 4 1 0 |, U = | 0 -3 -6 |
    *      | 7 2 1 |      | 0 0 1 |
    *
    * with pivoting: P*A = L*U
    *      | 0 0 1 |
    *  P = | 1 0 0 |
    *      | 0 1 0 |
    */

```

(续下页)

(接上页)

```

*          | 1      0      0 |          | 7  8      10      |
*   L = | 0.1429  1      0 |, U = | 0  0.8571  1.5714 |
*          | 0.5714  0.5    1 |          | 0  0      -0.5    |
*/

const std::vector<double> A = {1.0, 4.0, 7.0, 2.0, 5.0, 8.0, 3.0, 6.0, ↵
↵10.0};
const std::vector<double> B = {1.0, 2.0, 3.0};
std::vector<double> X(m, 0);
std::vector<double> LU(lda * m, 0);
std::vector<int> Ipivot(m, 0);
int info = 0;

double *d_A = nullptr; /* device copy of A */
double *d_B = nullptr; /* device copy of B */
int *d_Ipivot = nullptr; /* pivoting sequence */
int *d_info = nullptr; /* error info */

int lwork = 0;          /* size of workspace */
double *d_work = nullptr; /* device workspace for getrf */

const int pivot_on = 0;

if (pivot_on)
{
    printf("pivot is on : compute P*A = L*U \n");
}
else
{
    printf("pivot is off: compute A = L*U (not numerically stable)\n");
}

printf("A = (matlab base-1)\n");
print_matrix(m, m, A.data(), lda);
printf("====\n");

printf("B = (matlab base-1)\n");
print_matrix(m, 1, B.data(), ldb);
printf("====\n");

/* step 1: create cusolver handle, bind a stream */
(cusolverDnCreate(&cusolverH));

(cudaStreamCreateWithFlags(&stream, cudaStreamNonBlocking));
(cusolverDnSetStream(cusolverH, stream));

/* step 2: copy A to device */
(cudaMalloc(reinterpret_cast<void **>(&d_A), sizeof(double) * A.
↵size()));

```

(续下页)

(接上页)

```

(cudaMalloc(reinterpret_cast<void **>(&d_B), sizeof(double) * B.
↪size()));
(cudaMalloc(reinterpret_cast<void **>(&d_Ipiv), sizeof(int) * Ipiv.
↪size()));
(cudaMalloc(reinterpret_cast<void **>(&d_info), sizeof(int)));

(cudaMemcpyAsync(d_A, A.data(), sizeof(double) * A.size(), ↪
↪cudaMemcpyHostToDevice, stream));
(cudaMemcpyAsync(d_B, B.data(), sizeof(double) * B.size(), ↪
↪cudaMemcpyHostToDevice, stream));

/* step 3: query working space of getrf */
(cusolverDnDgetrf_bufferSize(cusolverH, m, m, d_A, lda, &lwork));

(cudaMalloc(reinterpret_cast<void **>(&d_work), sizeof(double) * ↪
↪lwork));

/* step 4: LU factorization */
if (pivot_on)
{
    (cusolverDnDgetrf(cusolverH, m, m, d_A, lda, d_work, d_Ipiv, d_
↪info));
}
else
{
    (cusolverDnDgetrf(cusolverH, m, m, d_A, lda, d_work, NULL, d_info));
}

if (pivot_on)
{
    (cudaMemcpyAsync(Ipiv.data(), d_Ipiv, sizeof(int) * Ipiv.size(), ↪
↪cudaMemcpyDeviceToHost, stream));
}
(cudaMemcpyAsync(LU.data(), d_A, sizeof(double) * A.size(), ↪
↪cudaMemcpyDeviceToHost, stream));
(cudaMemcpyAsync(&info, d_info, sizeof(int), cudaMemcpyDeviceToHost, ↪
↪stream));

(cudaStreamSynchronize(stream));

if (0 > info)
{
    printf("%d-th parameter is wrong \n", -info);
    exit(1);
}
if (pivot_on)
{
    printf("pivoting sequence, matlab base-1\n");
    for (int j = 0; j < m; j++)

```

(续下页)

```

        {
            printf("Ipivot(%d) = %d\n", j + 1, Ipiv[j]);
        }
    }
    printf("L and U = (matlab base-1)\n");
    print_matrix(m, m, LU.data(), lda);
    printf("====\n");

    /*
     * step 5: solve A*X = B
     *      | 1 |      | -0.3333 |
     *   B = | 2 |,   X = | 0.6667 |
     *      | 3 |      | 0      |
     *
     */
    if (pivot_on)
    {
        (cusolverDnDgetrs(cusolverH, CUBLAS_OP_N, m, 1, d_A, lda, d_Ipiv, d_
↪B, ldb, d_info));
    }
    else
    {
        (cusolverDnDgetrs(cusolverH, CUBLAS_OP_N, m, 1, d_A, lda, NULL, d_B,
↪ldb, d_info));
    }

    (cudaMemcpyAsync(X.data(), d_B, sizeof(double) * X.size(),
↪cudaMemcpyDeviceToHost, stream));
    (cudaStreamSynchronize(stream));

    printf("X = (matlab base-1)\n");
    print_matrix(m, 1, X.data(), ldb);
    printf("====\n");

    /* free resources */
    (cudaFree(d_A));
    (cudaFree(d_B));
    (cudaFree(d_Ipiv));
    (cudaFree(d_info));
    (cudaFree(d_work));

    (cusolverDnDestroy(cusolverH));

    (cudaStreamDestroy(stream));

    (cudaDeviceReset());

    return EXIT_SUCCESS;
}

```

3. 在该目录下创建如下作业提交脚本 cuSOLVERtest.slurm:

```
#!/bin/bash

#SBATCH --job-name=cuSOLVERtest      # 作业名
#SBATCH --partition=a100             # a100 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1            # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                 # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.5.0
module load gcc/11.2.0

nvcc cuSOLVERtest.cu -o cuSOLVERtest -lcusolver
./cuSOLVERtest
```

4. 使用如下命令提交作业:

```
sbatch cuSOLVERtest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
pivot is off: compute A = L*U (not numerically stable)
A = (matlab base-1)
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 10.00
=====
B = (matlab base-1)
1.00
2.00
3.00
=====
L and U = (matlab base-1)
1.00 2.00 3.00
4.00 -3.00 -6.00
7.00 2.00 1.00
=====
X = (matlab base-1)
-0.33
0.67
0.00
=====
```

pi2.0 上的 cuSOLVER

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下创建如下作业提交脚本 `cuSOLVERtest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=cuSOLVERStest      # 作业名
#SBATCH --partition=dgx2              # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1             # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                 # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.6.2-gcc-8.3.0
module load gcc/8.3.0

nvcc cuSOLVERtest.cu -o cuSOLVERtest -lcusolver
./cuSOLVERtest
```

4. 使用如下命令提交作业:

```
sbatch cuSOLVERtest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
pivot is off: compute A = L*U (not numerically stable)
A = (matlab base-1)
1.00 2.00 3.00
4.00 5.00 6.00
7.00 8.00 10.00
=====
B = (matlab base-1)
1.00
2.00
3.00
=====
L and U = (matlab base-1)
1.00 2.00 3.00
4.00 -3.00 -6.00
7.00 2.00 1.00
=====
X = (matlab base-1)
-0.33
0.67
0.00
=====
```

参考资料

- [cuSOLVER 官网](#)
- [cuSOLVER github](#)

cuFFT

简介

The cuFFT is a CUDA Fast Fourier Transform library consisting of two components: cuFFT and cuFFTW. The cuFFT library provides high performance on NVIDIA GPUs, and the cuFFTW library is a porting tool to use FFTW on NVIDIA GPUs.

cuFFT 使用说明

思源一号上的 **cuFFT**

1. 先创建一个目录 **cuFFTtest** 并进入该目录:

```
mkdir cuFFTtest
cd cuFFTtest
```

2. 在该目录下创建如下测试文件 **cuFFTtest.cu**:

```
// In this example a one-dimensional complex-to-complex transform is
→applied to the input data.
//Afterwards an inverse transform is performed on the computed frequency_
→domain representation.

#include <complex>
#include <iostream>
#include <random>
#include <vector>

#include <cuda_runtime.h>
#include <cufftXt.h>

// #include "cufft_utils.h"

// CUDA API error checking
#define CUDA_RT_CALL(call)
→ \
{
→ \
    auto status = static_cast<cudaError_t>(call);
→ \
    if (status != cudaSuccess)
→ \
        fprintf(stderr,
```

(续下页)

(接上页)

```

→ \
                                "ERROR: CUDA RT call \"%s\" in line %d of file %s_
→failed " \
                                "with "
→ \
                                "%s (%d).\n",
→ \
                                #call,
→ \
                                __LINE__,
→ \
                                __FILE__,
→ \
                                cudaGetErrorString(status),
→ \
                                status);
→ \
    }

// cufft API error chekcing
#define CUFFT_CALL(call)
→ \
    {
→ \
        auto status = static_cast<cufftResult>(call);
→ \
        if (status != CUFFT_SUCCESS)
→ \
            fprintf(stderr,
→ \
                "ERROR: CUFFT call \"%s\" in line %d of file %s failed
→" \
                "with "
→ \
                "code (%d).\n",
→ \
                #call,
→ \
                __LINE__,
→ \
                __FILE__,
→ \
                status);
→ \
    }

int main(int argc, char *argv[])
{
    cufftHandle plan;

```

(续下页)

(接上页)

```

cudaStream_t stream = NULL;

int n = 8;
int batch_size = 2;
int fft_size = batch_size * n;

using scalar_type = float;
using data_type = std::complex<scalar_type>;

std::vector<data_type> data(fft_size);

for (int i = 0; i < fft_size; i++)
{
    data[i] = data_type(i, -i);
}

std::printf("Input array:\n");
for (auto &i : data)
{
    std::printf("%f + %fj\n", i.real(), i.imag());
}
std::printf("====\n");

cufftComplex *d_data = nullptr;

CUFFT_CALL(cufftCreate(&plan));
CUFFT_CALL(cufftPlan1d(&plan, data.size(), CUFFT_C2C, batch_size));

CUDA_RT_CALL(cudaStreamCreateWithFlags(&stream, cudaStreamNonBlocking));
CUFFT_CALL(cufftSetStream(plan, stream));

// Create device data arrays
CUDA_RT_CALL(cudaMalloc(reinterpret_cast<void **>(&d_data), sizeof(data_
↪type) * data.size()));
CUDA_RT_CALL(cudaMemcpyAsync(d_data, data.data(), sizeof(data_type) *
↪data.size(), cudaMemcpyHostToDevice, stream));

/*
 * Note:
 * Identical pointers to data and output arrays implies in-place
↪transformation
 */
CUFFT_CALL(cufftExecC2C(plan, d_data, d_data, CUFFT_FORWARD));
CUFFT_CALL(cufftExecC2C(plan, d_data, d_data, CUFFT_INVERSE));

CUDA_RT_CALL(cudaMemcpyAsync(data.data(), d_data, sizeof(data_type) *
↪data.size(), cudaMemcpyDeviceToHost, stream));

CUDA_RT_CALL(cudaStreamSynchronize(stream));

```

(续下页)

(接上页)

```

std::printf("Output array:\n");
for (auto &i : data)
{
    std::printf("%f + %fj\n", i.real(), i.imag());
}
std::printf("====\n");

/* free resources */
CUDA_RT_CALL(cudaFree(d_data))

CUFFT_CALL(cufftDestroy(plan));

CUDA_RT_CALL(cudaStreamDestroy(stream));

CUDA_RT_CALL(cudaDeviceReset());

return EXIT_SUCCESS;
}

```

3. 在该目录下创建如下作业提交脚本 `cuFFTtest.slurm`:

```

#!/bin/bash

#SBATCH --job-name=cuFFTtest           # 作业名
#SBATCH --partition=a100               # a100 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1             # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                  # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.5.0
module load gcc/11.2.0

nvcc cuFFTtest.cu -o cuFFTtest -lcufft
./cuFFTtest

```

4. 使用如下命令提交作业:

```

sbatch cuFFTtest.slurm

```

5. 作业完成后在.out 文件中可看到如下结果:

```

Input array:
0.000000 + 0.000000j
1.000000 + -1.000000j
2.000000 + -2.000000j

```

(续下页)

(接上页)

```

3.000000 + -3.000000j
4.000000 + -4.000000j
5.000000 + -5.000000j
6.000000 + -6.000000j
7.000000 + -7.000000j
8.000000 + -8.000000j
9.000000 + -9.000000j
10.000000 + -10.000000j
11.000000 + -11.000000j
12.000000 + -12.000000j
13.000000 + -13.000000j
14.000000 + -14.000000j
15.000000 + -15.000000j
=====
Output array:
0.000004 + 0.000000j
16.000015 + -16.000004j
32.000004 + -32.000004j
48.000004 + -48.000004j
64.000000 + -64.000000j
80.000000 + -80.000008j
96.000000 + -96.000015j
112.000015 + -112.000000j
128.000000 + -128.000000j
144.000000 + -144.000000j
160.000000 + -160.000000j
176.000000 + -176.000000j
192.000000 + -192.000000j
208.000000 + -208.000000j
224.000000 + -223.999985j
239.999985 + -240.000000j
=====

```

pi2.0 上的 cuFFT

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下创建如下作业提交脚本 `cuFFTtest.slurm`:

```

#!/bin/bash

#SBATCH --job-name=cuFFTtest          # 作业名
#SBATCH --partition=dgx2              # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1             # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                  # 1 块 GPU

```

(续下页)

(接上页)

```
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cuda/11.6.2-gcc-8.3.0
module load gcc/8.3.0

nvcc cuFFTtest.cu -o cuFFTtest -lcufft
./cuFFTtest
```

4. 使用如下命令提交作业:

```
sbatch cuFFTtest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
Input array:
0.000000 + 0.000000j
1.000000 + -1.000000j
2.000000 + -2.000000j
3.000000 + -3.000000j
4.000000 + -4.000000j
5.000000 + -5.000000j
6.000000 + -6.000000j
7.000000 + -7.000000j
8.000000 + -8.000000j
9.000000 + -9.000000j
10.000000 + -10.000000j
11.000000 + -11.000000j
12.000000 + -12.000000j
13.000000 + -13.000000j
14.000000 + -14.000000j
15.000000 + -15.000000j
=====
Output array:
0.000004 + 0.000000j
16.000015 + -16.000004j
32.000004 + -32.000004j
48.000004 + -48.000004j
64.000000 + -64.000000j
80.000000 + -80.000008j
96.000000 + -96.000015j
112.000015 + -112.000000j
128.000000 + -128.000000j
144.000000 + -144.000000j
160.000000 + -160.000000j
176.000000 + -176.000000j
192.000000 + -192.000000j
208.000000 + -208.000000j
224.000000 + -223.999985j
```

(续下页)

(接上页)

```
239.999985 + -240.000000j
=====
```

参考资料

- [cuFFT 官网](#)
- [cuFFT github](#)

OpenMPI

简介

OpenMPI 是一个免费的、开源的 MPI 实现，兼容 MPI-1 和 MPI-2 标准。OpenMPI 由开源社区开发维护，支持大多数类型的 HPC 平台，并具有很高的性能。

OpenMPI 使用说明

思源一号上的 OpenMPI

1. 先创建一个目录 `openmpitest` 并进入该目录：

```
mkdir openmpitest
cd openmpitest
```

2. 在该目录下创建如下测试文件 `openmpitest.c`：

```
#include "stdio.h"
#include "mpi.h"
#include "omp.h"
#include "math.h"

#define NUM_THREADS 8
long int n = 10000000;

int main(int argc, char *argv[])
{
    int my_rank, numprocs;
    long int i, my_n, my_first_i, my_last_i;
    double my_pi = 0.0, pi, h, x;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    h = 1.0 / n;
    my_n = n / numprocs;
    my_first_i = my_rank * my_n;
    my_last_i = my_first_i + my_n;
```

(续下页)

(接上页)

```

omp_set_num_threads(NUM_THREADS);
#pragma omp parallel for reduction(+ : my_pi) private(x, i)
for (i = my_first_i; i < my_last_i; i++)
{
    x = (i + 0.5) * h;
    my_pi = my_pi + 4.0 / (1.0 + x * x);
}

MPI_Reduce(&my_pi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

if (my_rank == 0)
{
    printf("Approximation of pi:%15.13f\n", pi * h);
}

MPI_Finalize();

return 0;
}

```

3. 在该目录下创建如下作业提交脚本 openmpitest.slurm:

```

#!/bin/bash

#SBATCH --job-name=openmpitest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 4
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load gcc
module load openmpi/4.1.1-gcc-8.3.1

mpicc openmpitest.c -o openmpitest -fopenmp

mpirun -np 4 ./openmpitest

```

4. 使用如下命令提交作业:

```

sbatch openmpitest.slurm

```

5. 作业完成后在.out 文件中可看到如下结果:

```

Approximation of pi:3.1415926535898

```

pi2.0 上的 OpenMPI

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下创建如下作业提交脚本 `openmpitest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=openmpitest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 4
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load gcc
module load openmpi/3.1.5-gcc-9.2.0

mpicc openmpitest.c -o openmpitest -fopenmp

mpirun -np 4 ./openmpitest
```

4. 使用如下命令提交作业:

```
sbatch openmpitest.slurm
```

5. 作业完成后在 `.out` 文件中可看到如下结果:

```
Approximation of pi:3.1415926535898
```

参考资料

- [OpenMPI 入门教程](#)

Mpi4py

简介

Mpi4py 是一个构建在 MPI 之上的 Python 库，它使得 Python 的数据结构可以方便的在多线程中传递。Mpi4py 是一个很强大的库，它实现了很多 MPI 标准中的接口，包括点对点通信，集合通信、阻塞/非阻塞通信、组间通信等，基本上能用到的 MPI 接口都有相应的实现。不仅是任何可以被 pickle 的 Python 对象，Mpi4py 对具有单段缓冲区接口的 Python 对象如 numpy 数组及内置的 bytes/string/array 等也有很好的支持并且传递效率很高。同时它还提供了 SWIG 和 F2PY 的接口能够将 C/C++ 或者 Fortran 程序在封装成 Python 后仍然能够使用 Mpi4py 的对象和接口来进行并行处理。

Mpi4py 安装以及使用说明

思源一号上的 Mpi4py

1. 先创建一个目录 Mpi4pytest 并进入该目录:

```
mkdir Mpi4pytest
cd Mpi4pytest
```

2. 申请计算资源并通过 conda 安装 mpi4py

```
srun -p 64c512g -n 10 --pty /bin/bash
module load miniconda3
conda create -n mpi4py425
source activate mpi4py425
conda install -c conda-forge mpi4py
```

3. 在该目录下创建如下测试文件 test.py:

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD # 获取通信子
rank = comm.Get_rank() # 获取当前进程的rank
size = comm.Get_size() # 获取进程的总数

sendbuf = None # 定义发送缓冲区

if rank == 0:
    # 在根进程中初始化数组
    sendbuf = np.arange(size * 2, dtype='i')
    print("Sendbuf:", sendbuf)

# 定义接收缓冲区
recvbuf = np.empty(2, dtype='i')

# 将数据从根进程分发到其他进程中
comm.Scatter(sendbuf, recvbuf, root=0)
print("Process %d received: %s" % (rank, recvbuf))
```

4. 在该目录下创建如下作业提交脚本 mpi4pytest.slurm:

```
#!/bin/bash

#BATCH --job-name=mpi4pytest # 作业名
#SBATCH --partition=64c512g # 64c512g队列
#SBATCH --ntasks-per-node=10 # 每节点核数
#SBATCH -n 10 # 作业核心数
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

(续下页)

(接上页)

```
ulimit -s unlimited
ulimit -l unlimited

module load miniconda3
source activate mpi4py425

mpirun -np 10 python3 test.py
```

5. 使用如下命令提交作业:

```
sbatch mpi4pytest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
Sendbuf: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
Process 0 received: [0 1]
Process 8 received: [16 17]
Process 4 received: [8 9]
Process 6 received: [12 13]
Process 7 received: [14 15]
Process 5 received: [10 11]
Process 9 received: [18 19]
Process 2 received: [4 5]
Process 1 received: [2 3]
Process 3 received: [6 7]
```

pi2.0 上的 Mpi4py

1. 此步骤和上文完全相同;
2. 申请计算资源并通过 conda 安装 mpi4py

```
srunk -p cpu -N 1 --ntasks-per-node 40 --pty /bin/bash
module load miniconda3
conda create -n mpi4py425
source activate mpi4py425
conda install -c conda-forge mpi4py
```

3. 此步骤和上文完全相同;
4. 在该目录下创建如下作业提交脚本 mpi4pytest.slurm:

```
#!/bin/bash

#BATCH --job-name=mpi4pytest      # 作业名
#SBATCH --partition=small        # small 队列
#SBATCH --ntasks-per-node=10     # 每节点核数
#SBATCH -n 10                    # 作业核心数
#SBATCH --output=%j.out
```

(续下页)

(接上页)

```
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load miniconda3
source activate mpi4py425

mpirun -np 10 python3 test.py
```

5. 此步骤和上文完全相同；

6. 此步骤和上文完全相同；

参考资料

- [Mpi4py github](#)
- [Mpi4py 知乎](#)

Numba

简介

Numba 是一款可以将 python 函数编译为机器代码的 JIT 编译器，经过 Numba 编译的 python 代码 (仅限数组运算)，其运行速度可以接近 C 或 FORTRAN 语言。

Numba 安装以及使用说明

思源一号上的 Numba

1. 先创建一个目录 Numbatest 并进入该目录：

```
mkdir Numbatest
cd Numbatest
```

2. 申请计算资源并通过 conda 安装 Numba

```
srunk -p 64c512g -n 10 --pty /bin/bash
module load miniconda3
conda create -n numbatest
source activate numbatest
conda install -c conda-forge numba
```

3. 在该目录下创建如下测试文件 test.py：

```
import numba as nb
import numpy as np
from numba.typed import List
```

(续下页)

(接上页)

```

import time

@nb.jit('List(f4)(f4[:,], f4[:,], i4)', nopython=True, cache=True,
↪parallel=False)
def fun1(a, b, len):
    res = []
    for i in range(len):
        res.append(a[i]+b[i])
    return res

@nb.jit('ListType(f4)(f4[:,], f4[:,], i4)', nopython=True, cache=True,
↪parallel=False)
def fun2(a, b, len):
    res = List()
    for i in range(len):
        res.append(a[i]+b[i])
    return res

def fun3(a, b, len):
    res = []
    for i in range(len):
        res.append(a[i]+b[i])
    return res

if __name__ == '__main__':
    len = 100000000
    a = np.random.randn(len).astype(np.float32)
    b = np.random.randn(len).astype(np.float32)
    t1 = time.time()
    c1 = fun1(a, b, len)
    t2 = time.time()
    c2 = fun2(a, b, len)
    t3 = time.time()
    c3 = fun3(a, b, len)
    t4 = time.time()

    print(f'fun1 cost: {t2-t1}s, \nfun2 cost: {t3-t2}s, \nfun3 cost: {t4-t3}
↪s.')

```

4. 在该目录下创建如下作业提交脚本 numbatest.slurm:

```

#!/bin/bash

#BATCH --job-name=numbatest          # 作业名
#SBATCH --partition=64c512g          # 64c512g队列
#SBATCH --ntasks-per-node=10         # 每节点核数
#SBATCH -n 10                        # 作业核心数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

```

(续下页)

(接上页)

```
ulimit -s unlimited
ulimit -l unlimited

module load miniconda3
source activate numbatest

python3 test.py
```

5. 使用如下命令提交作业:

```
sbatch numbatest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
fun1 cost: 2.0397536754608154s,
fun2 cost: 1.9905965328216553s,
fun3 cost: 17.56288480758667s.
```

pi2.0 上的 Numba

1. 此步骤和上文完全相同;
2. 申请计算资源并通过 conda 安装 numba

```
srunk -p cpu -N 1 --ntasks-per-node 40 --pty /bin/bash
module load miniconda3
conda create -n numbatest
source activate numbatest
conda install -c conda-forge numba
```

3. 此步骤和上文完全相同;
4. 在该目录下创建如下作业提交脚本 mpi4pytest.slurm:

```
#!/bin/bash

#BATCH --job-name=numbatest      # 作业名
#SBATCH --partition=small        # small队列
#SBATCH --ntasks-per-node=10     # 每节点核数
#SBATCH -n 10                    # 作业核心数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load miniconda3
source activate numbatest
```

(续下页)

(接上页)

```
python3 test.py
```

5. 此步骤和上文完全相同；

6. 此步骤和上文完全相同；

参考资料

- [Numba 官网](#)
- [Numba github](#)
- [Numba 知乎](#)

Cupy

简介

CuPy 是一个借助 CUDA GPU 库在英伟达 GPU 上实现 Numpy 数组的库。基于 Numpy 数组的实现，GPU 自身具有的多个 CUDA 核心可以促成更好的并行加速。CuPy 接口是 Numpy 的一个镜像，并且在大多情况下，它可以直接替换 Numpy 使用。只要用兼容的 CuPy 代码替换 Numpy 代码，用户就可以实现 GPU 加速。CuPy 支持 Numpy 的大多数数组运算，包括索引、广播、数组数学以及各种矩阵变换。

Cupy 安装以及使用说明

思源一号上的 Cupy

1. 先创建一个目录 Cupytest 并进入该目录：

```
mkdir Cupytest
cd Cupytest
```

2. 申请计算资源并通过 conda 安装 cupy

```
srun -p 64c512g -n 10 --pty /bin/bash
module load miniconda3
conda create -n cupytest
source activate cupytest
conda install -c conda-forge cupy
```

3. 在该目录下创建如下测试文件 test.py：

```
import numpy as np
import cupy as cp
import time

start_time = time.time()
```

(续下页)

(接上页)

```
x0 = np.ones((100,1000,1000))
x1= 5*x0
X2= x1*x1
end_time = time.time()
print('numpy执行用时：',end_time - start_time)

start_time1 = time.time()
x3 = cp.ones((100,1000,1000))
x4= 5*x3
X5= x4*x4
end_time1 = time.time()
print('cupy执行用时：',end_time1 - start_time1)
```

4. 在该目录下创建如下作业提交脚本 `cupytest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=cupytest          # 作业名
#SBATCH --partition=a100             # a100 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1            # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                 # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load miniconda3
source activate cupytest

python3 test.py
```

5. 使用如下命令提交作业:

```
sbatch cupytest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
numpy执行用时： 0.5541410446166992
cupy执行用时： 0.3665752410888672
```

pi2.0 上的 Cupy

1. 此步骤和上文完全相同;
2. 申请计算资源并通过 conda 安装 cupy

```
srunk -p cpu -N 1 --ntasks-per-node 40 --pty /bin/bash
module load miniconda3
conda create -n cupytest
```

(续下页)

(接上页)

```
source activate cupytest
conda install -c conda-forge cupy
```

3. 此步骤和上文完全相同；
4. 在该目录下创建如下作业提交脚本 `cupytest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=cupytest           # 作业名
#SBATCH --partition=dgx2             # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1             # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                 # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load miniconda3
source activate cupytest

python3 test.py
```

5. 此步骤和上文完全相同；
6. 此步骤和上文完全相同；

参考资料

- [Cupy github](#)
- [Cupy 知乎](#)

PyCUDA

简介

Nvidia 的 CUDA 为我们提供了一种便捷的方式来直接操纵 GPU 进行并行编程，但是基于 C 语言的 CUDA 实现较为复杂，开发周期较长。而 python 作为一门广泛使用的语言，具有简单易学、语法简单、开发迅速等优点。PyCUDA 可以通过将 python 和 C 语言结合的方式进行 CUDA 编程。

PyCUDA 安装以及使用说明

思源一号上的 PyCUDA

1. 先创建一个目录 `PyCUDAtest` 并进入该目录：

```
mkdir PyCUDAtest
cd PyCUDAtest
```

2. 申请计算资源并通过 conda 安装 PyCUDA

```

srun -p 64c512g -n 10 --pty /bin/bash
module load miniconda3
conda create -n PyCUDAtest
source activate PyCUDAtest
conda install -c conda-forge pycuda

```

3. 在该目录下创建如下测试文件 test.py:

```

import pycuda.driver as drv
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

# 定义核函数
mod = SourceModule(
    """
    __global__ void add_vectors(float *a, float *b, float *c, int n)
    {
        int idx = threadIdx.x + blockIdx.x * blockDim.x;
        if (idx < n)
        {
            c[idx] = a[idx] + b[idx];
        }
    }
    """
)

# 定义向量大小
n = 10000

# 生成随机向量数据
a = numpy.random.randn(n).astype(numpy.float32)
b = numpy.random.randn(n).astype(numpy.float32)

# 分配输出内存空间
c = numpy.zeros_like(a)

# 将输入输出数据复制到 GPU
a_gpu = drv.mem_alloc(a.nbytes)
b_gpu = drv.mem_alloc(b.nbytes)
c_gpu = drv.mem_alloc(c.nbytes)

drv.memcpy_htod(a_gpu, a)
drv.memcpy_htod(b_gpu, b)

# 定义块和网格大小
blocksize = 256
gridsize = (n + blocksize - 1) // blocksize

```

(续下页)

(接上页)

```
# 执行核函数
add_vectors = mod.get_function("add_vectors")
add_vectors(
    a_gpu, b_gpu, c_gpu, numpy.int32(n), block=(blocksize, 1, 1),
    →grid=(gridsize, 1)
)

# 将结果从 GPU 复制回 CPU
drv.memcpy_dtoh(c, c_gpu)

# 检查计算结果是否正确
assert numpy.allclose(c, a + b), "result not correct"

# 输出结果
print("a:", a)
print("b:", b)
print("c:", c)
```

4. 在该目录下创建如下作业提交脚本 pycudatest.slurm:

```
#!/bin/bash

#SBATCH --job-name=pycudatest      # 作业名
#SBATCH --partition=a100          # a100 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1          # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1              # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load miniconda3
source activate PyCUDAtest
module load cuda/11.5.0

python3 test.py
```

5. 使用如下命令提交作业:

```
sbatch pycudatest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
a: [ 0.32799047 -0.03553623 -1.6576846 ... -0.44243634 -1.1451671
    -1.1334891 ]
b: [-0.46226323  0.76997334 -0.06620226 ...  0.6974032   2.1895697
```

(续下页)

(接上页)

```
1.2849816 ]
c: [-0.13427275  0.7344371 -1.7238868 ... 0.25496686  1.0444026
0.15149248]
```

pi2.0 上的 PyCUDA

1. 此步骤和上文完全相同；
2. 申请计算资源并通过 conda 安装 PyCUDA

```
srun -p cpu -N 1 --ntasks-per-node 40 --pty /bin/bash
module load miniconda3
conda create -n PyCUDAtest
source activate PyCUDAtest
conda install -c conda-forge pycuda
```

3. 此步骤和上文完全相同；
4. 在该目录下创建如下作业提交脚本 mpi4pytest.slurm:

```
#!/bin/bash

#SBATCH --job-name=pycudatest           # 作业名
#SBATCH --partition=dgx2               # dgx2 队列
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1              # 1:1 的 GPU:CPU 配比
#SBATCH --gres=gpu:1                  # 1 块 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load miniconda3
source activate PyCUDAtest
module load cuda/11.6.2-gcc-8.3.0

python3 test.py
```

5. 此步骤和上文完全相同；
6. 此步骤和上文完全相同；

参考资料

- [PyCUDA github](#)

OpenBLAS

OpenBLAS 是一个开源的线性代数库，高效实现了 BLAS(Basic Linear Algebra Sub-programs) 和 LAPACK(Linear Algebra PACKage) 接口定义的函数，超算平台提供了 X86 和 ARM 适用的版本。

可用 OpenBLAS 版本

版本	平台	构建方式	模块名
0.3.7		Spack	openblas/0.3.7-gcc-9.2.0
0.3.7		Spack	openblas/0.3.7-gcc-9.3.0

链接 OpenBLAS 库

 小心

CPU 平台 (cpu, small, huge, 192c6t 等队列) 与 ARM 平台 (arm128c256g 队列) 指令集不兼容，请勿混用两个平台的二进制程序。

 小技巧

请使用与 OpenBLAS 库相匹配的特定版本编译器以获得最佳性能。

下面将分别展示如何在 CPU(X86) 和 ARM 平台构建示例程序 sampleblas，这个程序调用 OpenBLAS 提供的 cblas_dgemm 函数，完成矩阵乘加操作。

sampleblas 的源代码 sampleblas.c 内容如下：

```
#include <cblas.h>
#include <stdio.h>

int main() {

    int i = 0;
    double A[6] = {1.0, 2.0, 1.0, -3.0, 4.0, -1.0}; // A(3x2)
    double B[6] = {1.0, 2.0, 1.0, -3.0, 4.0, -1.0}; // B(2x3)
    double C[9] = {.5, .5, .5, .5, .5, .5, .5, .5, .5}; // C(3x3)

    const int M = 3; // row of A and C
    const int N = 3; // col of B and C
    const int K = 2; // col of A and row of B

    const double alpha = 1.0;
```

(续下页)

(接上页)

```

const double beta = 0.1;

// C = alpha * A * B + beta * C
cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, M, N, K, alpha,
↪A, K, B, N, beta, C, N);

for (i = 0; i < 9; i++) {
    printf("%lf ", C[i]);
}
printf("\n");

return 0;
}

```

在 CPU 平台上链接 OpenBLAS 库

在这个示例中我们使用 openblas/0.3.7-gcc-9.2.0 模块，这个模块使用 GCC 9.2.0 构建，需要载入 OpenBLAS 以及与之相匹配的编译器：

```
$ module load openblas/0.3.7-gcc-9.2.0 gcc/9.2.0-gcc-4.8.5
```

编译源代码并链接至 OpenBLAS 库，由于模块中已经预置了头文件、静态库和动态库的路径，因此不需要在命令行中显式制定这些路径：

```
$ gcc -o sampleblas sampleblas.c -lopenblas
```

检查二进制程序的动态链接情况，确认已经链接正确的 OpenBLAS 库：

```

$ ldd sampleblas
linux-vdso.so.1 => (0x00007fff6bfca000)
libopenblas.so.0 => /lustre/opt/cascadelake/linux-centos7-cascadelake/
↪gcc-9.2.0/openblas-0.3.7-kf4td3bj4liyg3magigle6h5dubwsrrg/lib/libopenblas.
↪so.0 (0x00002ae926124000)
libc.so.6 => /lib64/libc.so.6 (0x00002ae926fdc000)
libm.so.6 => /lib64/libm.so.6 (0x00002ae9273aa000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x00002ae9276ac000)
libgfortran.so.5 => /lustre/opt/cascadelake/linux-centos7-x86_64/gcc-4.
↪8.5/gcc-9.2.0-wqdec4rkyhejagxwmnabt6lscgm45d/lib64/libgfortran.so.5
↪(0x00002ae9278c8000)
libgomp.so.1 => /lustre/opt/cascadelake/linux-centos7-x86_64/gcc-4.8.5/
↪gcc-9.2.0-wqdec4rkyhejagxwmnabt6lscgm45d/lib64/libgomp.so.1
↪(0x00002ae927d57000)
/lib64/ld-linux-x86-64.so.2 => /lib/ld-linux.so.2 (0x00002ae925f00000)
libquadmath.so.0 => /lustre/opt/cascadelake/linux-centos7-x86_64/gcc-4.
↪8.5/gcc-9.2.0-wqdec4rkyhejagxwmnabt6lscgm45d/lib64/libquadmath.so.0
↪(0x00002ae927f8d000)
libgcc_s.so.1 => /lustre/opt/cascadelake/linux-centos7-x86_64/gcc-4.8.
↪5/gcc-9.2.0-wqdec4rkyhejagxwmnabt6lscgm45d/lib64/libgcc_s.so.1

```

(续下页)

(接上页)

```
→ (0x00002ae9281d4000)
libdl.so.2 => /lib64/libdl.so.2 (0x00002ae9283ec000)
```

这个程序运行时间很短：

```
$ time ./sampleblas
-4.950000 10.050000 -0.950000 10.050000 -9.950000 4.050000 7.050000 4.
→ 050000 5.050000
./sampleblas 0.00s user 0.01s system 27% cpu 0.045 total
```

运行时间更长、消耗时间更多的计算程序，需要编写作业脚本，提交到作业调度系统。

在 ARM 平台上链接 OpenBLAS 库

在这个示例中我们使用 openblas/0.3.7-gcc-9.3.0 模块，这个模块使用 GCC 9.3.0 构建，需要载入 OpenBLAS 以及与之相匹配的编译器：

```
$ module load openblas/0.3.7-gcc-9.3.0 gcc/9.3.0-gcc-4.8.5
```

编译源代码并链接至 OpenBLAS 库，由于模块中已经预置了头文件、静态库和动态库的路径，因此不需要在命令行中显式制定这些路径：

```
$ gcc -o sampleblas sampleblas.c -lopenblas
```

检查二进制程序的动态链接情况，确认已经链接正确的 OpenBLAS 库：

```
$ ldd sampleblas
linux-vdso.so.1 => (0x000040002e9c0000)
libopenblas.so.0 => /lustre/opt/kunpeng920/linux-centos7-aarch64/gcc-9.
→ 3.0/openblas-0.3.7-jbipn2oklioz3ym7ra4vh3do3ph5ocou/lib/libopenblas.so.0
→ (0x000040002e9d0000)
libc.so.6 => /lib64/libc.so.6 (0x000040002f630000)
libm.so.6 => /lib64/libm.so.6 (0x000040002f7c0000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x000040002f880000)
libgfortran.so.5 => /lustre/opt/kunpeng920/linux-centos7-aarch64/gcc-4.
→ 8.5/gcc-9.3.0-a5tvx33on7quyl7o2sygvvyjqnysfcw6n/lib64/libgfortran.so.5
→ (0x000040002f8c0000)
libgomp.so.1 => /lustre/opt/kunpeng920/linux-centos7-aarch64/gcc-4.8.5/
→ gcc-9.3.0-a5tvx33on7quyl7o2sygvvyjqnysfcw6n/lib64/libgomp.so.1
→ (0x000040002fa30000)
/lib/ld-linux-aarch64.so.1 (0x000040002e970000)
libgcc_s.so.1 => /lustre/opt/kunpeng920/linux-centos7-aarch64/gcc-4.8.
→ 5/gcc-9.3.0-a5tvx33on7quyl7o2sygvvyjqnysfcw6n/lib64/libgcc_s.so.1
→ (0x000040002fa90000)
libdl.so.2 => /lib64/libdl.so.2 (0x000040002fad0000)
```

这个程序运行时间很短：

```
$ time ./sampleblas
-4.950000 10.050000 -0.950000 10.050000 -9.950000 4.050000 7.050000 4.
→050000 5.050000
./sampleblas 0.00s user 0.00s system 41% cpu 0.009 total
```

运行时间更长、消耗时间更多的计算程序，需要编写作业脚本，提交到作业调度系统。

提交依赖 **OpenBLAS** 库的作业

小心

CPU 平台 (cpu, small, huge, 192c6t 等队列) 与 ARM 平台 (arm128c256g 队列) 指令集不兼容，请勿混用两个平台的二进制程序。

作业成功运行的关键，是加载程序所依赖的软件模块。以 sampleblas 程序为例，它依赖 GCC 和 OpenBLAS，因此需要在作业脚本中载入相应模块。此外，OpenBLAS 采用 OpenMP 多线程并行，在作业脚本中为环境变量 NUM_OMP_THREADS 设置合理数值能达到最佳运行效果。

在 **CPU** 平台提交依赖 **OpenBLAS** 库的作业

准备作业脚本 sampleblas.slurm，内容如下：

```
#!/bin/bash

#SBATCH --job-name=openblas      # 作业名
#SBATCH --partition=cpu          # cpu队列
#SBATCH --ntasks-per-node=40    # 每节点核数
#SBATCH -n 40                    # 作业核心数40(一个节点)
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load openblas/0.3.7-gcc-9.2.0 gcc/9.2.0-gcc-4.8.5

export NUM_OMP_THREADS=40

time ./sampleblas
```

使用 sbatch 提交作业：

```
$ sbatch sampleblas.slurm
```

在 **ARM** 平台提交依赖 **OpenBLAS** 库的作业

准备作业脚本 `sampleblas.slurm`，内容如下：

```
#!/bin/bash

#SBATCH --job-name=openblas          # 作业名
#SBATCH --partition=arm128c256g      # ARM队列 (arm128c256g)
#SBATCH --ntasks-per-node=128        # 每节点核数
#SBATCH -n 128                       # 作业核心数128(一个节点)
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load openblas/0.3.7-gcc-9.3.0 gcc/9.3.0-gcc-4.8.5

export NUM_OMP_THREADS=128

time ./sampleblas
```

使用 `sbatch` 提交作业：

```
$ sbatch sampleblas.slurm
```

参考资料

- OpenBLAS 官方网站 <https://www.openblas.net>

Eigen

简介

Eigen 库是一个开源的矩阵运算库，其利用 C++ 模板编程的思想，构造所有矩阵通过传递模板参数形式完成。由于模板类不支持库链接方式编译，而且模板类要求全部写在头文件中，从而导致导致 **Eigen** 库只能通过开源的方式供大家使用，并且只需要包含 **Eigen** 头文件就能直接使用。

Eigen 使用说明

思源一号上的 **Eigen**

1. 先创建一个目录 `eigentest` 并进入该目录：

```
mkdir eigentest
cd eigentest
```

2. 在该目录下创建如下测试文件 `eigentest.cpp`：

```

// 利用Eigen提供的SparseLU求解器求解稀疏线性方程组Ax=b
#include <vector>
#include <Eigen/Sparse>
#include <Eigen/SparseLU>
#include <chrono>
#include <random>
#include <algorithm>
#include <stdlib.h>
#include <iostream>
#include <ctime>

using namespace std;
using namespace Eigen;
// using namespace chrono;

typedef SparseMatrix<double> SpMat;
typedef Triplet<double> T;

int main()
{
    int N = 200;

    Eigen::SparseMatrix<double> A(N, N); // 创建稀疏矩阵
    for (size_t i = 0; i < N; i++)
    {
        A.insert(i, i) = i + 1.1;
        if (i != N - 1)
        {
            A.insert(i, i + 1) = i + 4.4;
            A.insert(i + 1, i) = i + 3.3;
        }
        if (i != N - 1 && i != N - 2)
        {
            A.insert(i + 2, i) = i + 3.3;
        }
    }

    // A.makeCompressed(); // 将稀疏矩阵转换为压缩列格式
    // cout << "A = " << A << endl;

    Eigen::VectorXd b(N); // 创建 5 维向量
    for (size_t i = 0; i < N; i++)
    {
        b(i) = i + 1;
    }

    struct timespec start1, end1;
    double duration1;
    clock_gettime(CLOCK_REALTIME, &start1);

```

(续下页)

(接上页)

```

Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
solver.analyzePattern(A);
solver.factorize(A);
Eigen::VectorXd x = solver.solve(b);

clock_gettime(CLOCK_REALTIME, &end1);
duration1 = (end1.tv_sec - start1.tv_sec) + (end1.tv_nsec - start1.tv_
↪nsec) / 1000000000.0;
printf("Time taken by solving is %lf microseconds\n", duration1 * 1000);

for (size_t i = 0; i < 10; i++)
{
    printf("x[%d] = %f\n", i, x[i]);
}
cout << "=====" << endl;

return 0;
}

```

3. 在该目录下创建如下作业提交脚本 eigentest.slurm:

```

#!/bin/bash

#SBATCH --job-name=eigentest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load eigen/3.4.0-gcc-11.2.0
module load gcc/11.2.0

g++ eigentest.cpp -o eigentest

./eigentest

```

4. 使用如下命令提交作业:

```

sbatch eigentest.slurm

```

5. 作业完成后在.out 文件中可看到如下结果:

```

Time taken by solving is 2.991600 microseconds
x[0] = 5.866133
x[1] = -1.239260

```

(续下页)

(接上页)

```

x[2] = -2.732554
x[3] = -0.399766
x[4] = 3.439243
x[5] = 0.531062
x[6] = -2.109301
x[7] = -0.724832
x[8] = 2.550861
x[9] = 1.037857
=====

```

pi2.0 上的 Eigen

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下创建如下作业提交脚本 `eigentest.slurm`:

```

#!/bin/bash

#SBATCH --job-name=eigentest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load eigen/3.4.0-gcc-11.2.0
module load gcc/11.2.0

g++ eigentest.cpp -o eigentest

./eigentest

```

4. 使用如下命令提交作业:

```

sbatch eigentest.slurm

```

5. 作业完成后在 `.out` 文件中可看到如下结果:

```

Time taken by solving is 3.391600 microseconds
x[0] = 5.866133
x[1] = -1.239260
x[2] = -2.732554
x[3] = -0.399766
x[4] = 3.439243

```

(续下页)

(接上页)

```
x[5] = 0.531062
x[6] = -2.109301
x[7] = -0.724832
x[8] = 2.550861
x[9] = 1.037857
=====
```

参考资料

- [Eigen 官网](#)
- [Eigen 知乎](#)

FFTW

简介

FFTW 是由麻省理工学院计算机科学实验室超级计算技术组开发的一套离散傅立叶变换 (DFT) 的计算库，开源、高效和标准 C 语言编写的代码使其得到了非常广泛的应用。**Intel** 的数学库和 **Scilib**(类似于 **Matlab** 的科学计算软件) 都使用 **FFTW** 做 FFT 计算。**FFTW** 是计算离散 Fourier 变换 (DFT) 的快速 C 程序的一个完整集合。

FFTW 使用说明

思源一号上的 FFTW

1. 创建 **fftwtest** 目录并进入该目录：

```
mkdir fftwtest
cd fftwtest
```

2. 在该目录下编写如下 **myfftw.c** 文件：

```
#include <stdio.h>
#include "fftw3.h"

int main()
{
    fftw_complex *in, *out;
    fftw_plan p;
    int N= 8;
    int i;
    int j;

    in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);

    for( i=0; i < N; i++)
```

(续下页)

(接上页)

```

{
    in[i][0] = 1.0;
    in[i][1] = 0.0;
    printf("%6.2f ", in[i][0]);
}
printf("\n");

p=fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);
fftw_execute(p); /* repeat as needed*/

for(j = 0; j < N; j++)
{
    printf("%6.2f ", out[j][0]);
}
printf("\n");

fftw_destroy_plan(p);
fftw_free(in);
fftw_free(out);
return 0;
}

```

3. 在该目录下编写如下 `fftwtest.slurm` 脚本:

```

#!/bin/bash

#SBATCH --job-name=fftwtest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load gcc
module load fftw/3.3.10-gcc-8.3.1-openmpi

gcc myfftw.c -o myfftw -lfftw3 -lm

./myfftw

```

4. 使用如下命令提交作业:

```

sbatch fftwtest.slurm

```

5. 作业完成后在 `.out` 文件中可看到如下结果:

```
1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00
8.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

pi2.0 上的 FFTW

1. 此步骤和上文完全相同；
2. 此步骤和上文完全相同；
3. 在该目录下编写如下 `fftwtest.slurm` 脚本：

```
#!/bin/bash

#SBATCH --job-name=fftwtest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load gcc
module load fftw/3.3.8-gcc-4.8.5

gcc myfftw.c -o myfftw -lfftw3 -lm

./myfftw
```

4. 使用如下命令提交作业：

```
sbatch fftwtest.slurm
```

5. 作业完成后在 `.out` 文件中可看到如下结果：

```
1.00    1.00    1.00    1.00    1.00    1.00    1.00    1.00
8.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

参考资料

- [FFTW 官网](#)

GSL

简介

GSL(GNU Scientific Library) 是专门为应用数学和科学技术领域的数值计算提供支持的软件库。GSL 使用 C 语言编写，同时也为其他语言做了相应的封装。GSL 在 GNU 通用公共许可下是免费的。该函数库提供了广泛的数学算法的实现函数，包括随机数

生成器，特殊函数和最小二乘拟合等等。目前该函数库提供有超过 1000 个函数，这些函数包含的范围有：复数计算、多项式求根、特殊函数、向量和矩阵运算、排列、组合、排序、线性代数、特征值和特征向量、快速傅里叶变换 (FFT)、数值积分、随机数生成、随机数分布、统计、蒙特卡洛积分、模拟退火、常微分方程组、插值、数值微分、方程求根、最小二乘拟合、小波变换等。

GSL 使用说明

思源一号上的 GSL

1. 创建 gsltest 目录并进入该目录：

```
mkdir gsltest
cd gsltest
```

2. 在该目录下编写如下 mygsl.c 文件：

```
#include <stdio.h>
#include <gsl/gsl_linalg.h>

int main()
{
    double a_data[] = {1.0, 0.6, 0.0, 0.0, 1.5, 1.0, 0.0, 1.0, 1.0};
    double inva[9];
    int s, i, j;

    gsl_matrix_view m = gsl_matrix_view_array(a_data, 3, 3);
    gsl_matrix_view inv = gsl_matrix_view_array(inva, 3, 3);
    gsl_permutation *p = gsl_permutation_alloc(3);

    printf("The matrix is\n");
    for (i = 0; i < 3; ++i)
    {
        for (j = 0; j < 3; ++j)
        {
            printf(j == 2 ? "%6.3f\n" : "%6.3f ", gsl_matrix_get(&m.matrix,
↪i, j));
        }
    }

    gsl_linalg_LU_decomp(&m.matrix, p, &s);
    gsl_linalg_LU_invert(&m.matrix, p, &inv.matrix);

    printf("The inverse is\n");
    for (i = 0; i < 3; ++i)
    {
        for (j = 0; j < 3; ++j)
        {
            printf(j == 2 ? "%6.3f\n" : "%6.3f ", gsl_matrix_get(&inv.
↪matrix, i, j));
        }
    }
}
```

(续下页)

(接上页)

```
    }  
}  
  
gsl_permutation_free(p);  
  
return 0;  
}
```

3. 在该目录下编写如下 `gsltest.slurm` 脚本:

```
#!/bin/bash  
  
#SBATCH --job-name=gsltest  
#SBATCH --partition=64c512g  
#SBATCH --ntasks-per-node=1  
#SBATCH -n 1  
#SBATCH --output=%j.out  
#SBATCH --error=%j.err  
  
ulimit -s unlimited  
ulimit -l unlimited  
  
module load gcc/9.3.0  
module load gsl/2.7-gcc-9.3.0  
  
gcc mygsl.c -o mygsl -lgsl -lgslcblas -lm  
  
./mygsl
```

4. 使用如下命令提交作业:

```
sbatch gsltest.slurm
```

5. 作业完成后在 `.out` 文件中可看到如下结果:

```
The matrix is  
1.000  0.600  0.000  
0.000  1.500  1.000  
0.000  1.000  1.000  
The inverse is  
1.000 -1.200  1.200  
0.000  2.000 -2.000  
0.000 -2.000  3.000
```

pi2.0 上的 GSL

1. 此步骤和上文完全相同;
2. 此步骤和上文完全相同;

3. 在该目录下编写如下 `gsltest.slurm` 脚本：

```
#!/bin/bash

#SBATCH --job-name=gsltest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load gcc/9.2.0
module load gsl/2.5-gcc-9.2.0

gcc mygsl.c -o mygsl -lgsl -lgslcblas -lm

./mygsl
```

4. 使用如下命令提交作业：

```
sbatch gsltest.slurm
```

5. 作业完成后在`.out` 文件中可看到如下结果：

```
The matrix is
1.000  0.600  0.000
0.000  1.500  1.000
0.000  1.000  1.000
The inverse is
1.000 -1.200  1.200
0.000  2.000 -2.000
0.000 -2.000  3.000
```

参考资料

- [GSL 官方文档](#)

NVIDIA HPC SDK

NVIDIA HPC SDK 是适用于 GPU 平台的高性能计算编译器，库和工具套件。NVIDIA HPC SDK 包括经过优化的编译器，库和软件工具，这些工具对于最大化开发人员的工作效率以及 HPC 应用程序的性能和可移植性至关重要。

PGI 编译器和工具链已迁移至 NVIDIA HPC SDK。

本文档向您展示如何使用 NVIDIA HPC SDK，包含程序示例，编译，作业脚本示例。

可用版本

版本	加载方式
21.9	module load nvhpc/21.9-gcc-11.2.0 思源一号
21.11	module load nvhpc/20.11-gcc-4.8.5 闵行超算

程序示例 **CUDA_Fortran**

思源超算加载 NVIDIA HPC SDK 环境。

```
$ module load nvhpc/21.9-gcc-11.2.0
```

闵行超算加载 NVIDIA HPC SDK 环境。

```
$ module load nvhpc/20.11-gcc-4.8.5
```

编辑 deviceQuery.cuf 文件，内容如下：

```
program deviceQuery
  use cudafor
  implicit none

  type (cudaDeviceProp) :: prop
  integer :: nDevices=0, i, ierr

  ! Number of CUDA-capable devices

  ierr = cudaGetDeviceCount(nDevices)

  if (nDevices == 0) then
    write(*, "(/, 'No CUDA devices found', /)")
    stop
  else if (nDevices == 1) then
    write(*, "(/, 'One CUDA device found', /)")
  else
    write(*, "(/, i0, ' CUDA devices found', /)") nDevices
  end if

  ! Loop over devices

  do i = 0, nDevices-1

    write(*, "('Device Number: ', i0)") i

    ierr = cudaGetDeviceProperties(prop, i)
    if (ierr .eq. 0) then
      write(*, "(' GetDeviceProperties for device ', i0, ': Passed')") i
    else
```

(续下页)

(接上页)

```

        write(*,"('  GetDeviceProperties for device ',i0,': Failed')") i
    endif

    ! General device info

    write(*,"('  Device Name: ',a)") trim(prop%name)
    write(*,"('  Compute Capability: ',i0,'.',i0)") &
        prop%major, prop%minor
    write(*,"('  Number of Multiprocessors: ',i0)") &
        prop%multiProcessorCount
    write(*,"('  Max Threads per Multiprocessor: ',i0)") &
        prop%maxThreadsPerMultiprocessor
    write(*,"('  Global Memory (GB): ',f9.3,/)") &
        prop%totalGlobalMem/1024.0**3

    ! Execution Configuration

    write(*,"('  Execution Configuration Limits')")
    write(*,"('      Max Grid Dims: ',2(i0,' x '),i0)") &
        prop%maxGridSize
    write(*,"('      Max Block Dims: ',2(i0,' x '),i0)") &
        prop%maxThreadsDim
    write(*,"('      Max Threads per Block: ',i0,/)") &
        prop%maxThreadsPerBlock

    enddo
end program deviceQuery

```

程序示例: **deviceQuery**

使用 **nvfortran** 进行编译。

```
$ nvfortran -O2 -o deviceQuery.out deviceQuery.cuf
```

A100 队列提交作业脚本示例

这是一个名为 `a100.slurm` 的单机单卡作业脚本，该脚本向 **a100** 队列申请 1 块 GPU，并在作业完成时通知。

```

#!/bin/bash

#SBATCH --job-name=a100_test
#SBATCH --partition=a100
#SBATCH --gres=gpu:1
#SBATCH -n 1
#SBATCH --ntasks-per-node 1
#SBATCH --mail-type=end

```

(续下页)

(接上页)

```
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=cublas.out
#SBATCH --error=cublas.err

module load nvhpc/21.9-gcc-11.2.0

./deviceQuery.out
```

用以下方式提交作业:

```
$ sbatch a100.slurm
```

DGX2 队列提交作业脚本示例

这是一个名为 `dgx.slurm` 的单机单卡作业脚本，该脚本向 **dgx2** 队列申请 1 块 GPU，并在作业完成时通知。

```
#!/bin/bash

#SBATCH --job-name=dgx2_test
#SBATCH --partition=dgx2
#SBATCH --gres=gpu:1
#SBATCH -n 1
#SBATCH --ntasks-per-node 1
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=cublas.out
#SBATCH --error=cublas.err

module load nvhpc/20.11-gcc-4.8.5

./deviceQuery.out
```

用以下方式提交作业:

```
$ sbatch dgx.slurm
```

参考资料

- NVIDIA HPC SDK Version 20.11 Documentation <https://docs.nvidia.com/hpc-sdk/index.html>

Hypre

简介

Hypre 是运行在多核处理器上，借助目前性能较好的预处理矩阵 (preconditioner) 对于大型稀疏线性方程组使用迭代法求解的一个 C 语言库。其目标是让用户可以借助于

多核处理器的并行性能，并行存储矩阵不同范围的信息，并行地进行迭代法求解，从而达到事半功倍的效果。

可用的 **Hypre** 版本

版本	平台	构建方式	模块名
2.18.0		Spack	hypre/2.18.0-gcc-11.2.0-openblas-openmpi
2.20.0		Spack	hypre/2.20.0-gcc-11.2.0-openblas-openmpi
2.23.0		Spack	hypre/2.23.0-gcc-11.2.0-openblas-openmpi
其他版本		源码编译	用户家目录

Hypre 使用说明

思源一号上的 **Hypre**

1. 从 Github 上下载相关文件并解压，然后进入 **example** 目录：

```
wget https://github.com/hypre-space/hypre/archive/refs/tags/v2.22.0.tar.gz
tar xzvpf v2.22.0.tar.gz
cd hypre-2.22.0/src/examples
```

2. 在该目录下可看到如下文件 (其中以 **ex** 为前缀的文件均为示例文件)：

```
CMakeLists.txt
docs
ex10.cxx
ex11.c
ex12.c
ex12f.f
ex13.c
ex14.c
ex15big.c
ex15.c
ex16.c
ex17.c
ex18.c
ex18comp.c
ex1.c
ex2.c
ex3.c
ex4.c
ex5big.c
ex5.c
ex5f.f
ex6.c
```

(续下页)

(接上页)

```
ex7.c
ex8.c
ex9.c
ex.h
Makefile
Makefile_gpu
vis
vis.c
```

3. 在该目录下编写如下 `hyptest.slurm` 脚本文件 (编译 `ex1.c` 文件并运行):

```
#!/bin/bash

#SBATCH --job-name=hyptest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=2
#SBATCH -n 2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load openmpi/4.1.1-gcc-11.2.0
module load hypre/2.23.0-gcc-11.2.0-openblas-openmpi

mpicc ex1.c -lHYPRE -lm -o ex1

mpirun -np 2 ./ex1
```

4. 使用如下命令提交作业:

```
sbatch hyptest.slurm
```

5. 作业完成后可在 `.out` 文件中得到如下结果:

```
<C*b,b>: 1.800000e+01
```

Iters	r _C	conv.rate	r _C/ b _C
1	2.509980e+00	0.591608	5.916080e-01
2	9.888265e-01	0.393958	2.330686e-01
3	4.572262e-01	0.462393	1.077693e-01
4	1.706474e-01	0.373223	4.022197e-02
5	7.473022e-02	0.437922	1.761408e-02
6	3.402624e-02	0.455321	8.020061e-03
7	1.214929e-02	0.357057	2.863616e-03
8	3.533113e-03	0.290808	8.327628e-04
9	1.343893e-03	0.380371	3.167586e-04

(续下页)

(接上页)

10	2.968745e-04	0.220906	6.997400e-05
11	5.329671e-05	0.179526	1.256215e-05
12	7.308483e-06	0.137128	1.722626e-06
13	7.411552e-07	0.101410	1.746920e-07

pi2.0 上的 Hypre

1. 此步骤和上文完全一样。
2. 此步骤和上文完全一样。
3. 在该目录下编写如下 `hyptest.slurm` 脚本文件 (编译 `ex1.c` 文件并运行):

```
#!/bin/bash

#SBATCH --job-name=hyptest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=2
#SBATCH -n 2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load openmpi/4.0.5-gcc-9.2.0
module load hypre/2.20.0-gcc-9.2.0-openblas-openmpi

mpicc ex1.c -lHYPRE -lm -o ex1

mpirun -np 2 ./ex1
```

4. 使用如下命令提交作业:

```
sbatch hyptest.slurm
```

5. 作业完成后可在 `.out` 文件中得到如下结果:

```
<C*b,b>: 1.800000e+01

Iters      ||r||_C      conv.rate      ||r||_C/||b||_C
-----
  1      2.509980e+00      0.591608      5.916080e-01
  2      9.888265e-01      0.393958      2.330686e-01
  3      4.572262e-01      0.462393      1.077693e-01
  4      1.706474e-01      0.373223      4.022197e-02
  5      7.473022e-02      0.437922      1.761408e-02
  6      3.402624e-02      0.455321      8.020061e-03
  7      1.214929e-02      0.357057      2.863616e-03
```

(续下页)

(接上页)

8	3.533113e-03	0.290808	8.327628e-04
9	1.343893e-03	0.380371	3.167586e-04
10	2.968745e-04	0.220906	6.997400e-05
11	5.329671e-05	0.179526	1.256215e-05
12	7.308483e-06	0.137128	1.722626e-06
13	7.411552e-07	0.101410	1.746920e-07

编译 Hypre 库

自行在 **X86** 平台上编译 **Hypre** 库

首先申请计算资源:

```
$ srun -p small -n 4 --pty /bin/bash
```

Hypre 库的编译需要 **OpenMPI**。请根据自己的需要选择合适的 **OpenMPI** 及 **GCC** 版本。这里我们选择加载 **CPU** 及 **GPU** 平台上全局部署的 `openmpi/3.1.5-gcc-8.3.0`:

```
$ module load openmpi/3.1.5-gcc-8.3.0
```

进入 **Hypre** 的 **github** 中 **clone** 源代码

```
$ git clone https://github.com/hypre-space/hypre.git
```

进入 `hypre/src` 文件夹并进行编译:

```
$ cd hypre/src
$ ./configure -prefix=/lustre/home/$YOUR_ACCOUNT/$YOUR_USERNAME/mylibs/hypre
$ make install -j 4
```

编译完成之后, 在家目录下会出现一个 `mylibs` 文件夹, **Hypre** 库的头文件以及库文件分别在这 `mylibs/hypre/include` 以及 `mylibs/hypre/lib` 中。

```
$ ls mylibs/hypre
include lib
```

参考资料

- [Hypre github 主页](#)
- [Hypre 与 Petsc 安装文档及性能测试](#)

Hyper-MPI

HyperMPI 是一款华为自研的高性能通信库, 它基于 **OpenMPI 4.0.3** 和 **Open UCX 1.6.0** 构建, 支持 **MPI-V3.1** 标准的并行计算接口。相较于 **openMPI**, 它在基于鲲鹏处理器的高性能计算集群上表现更好。

加载预安装的 **HyperMPI**

可以用以下命令加载集群中已安装的 **Hyper-MPI**

版本	组件说明
4.0.3	使用 9.3.0 版本 gnu 编译套件编译的 4.0.3 版本的 HyperMPI

```
module load gcc/9.3.0-gcc-4.8.5
module load hmpi/4.0.3-gcc-9.3.0
```

使用 **HyperMPI** 编译应用

这里，我们演示如何使用系统中的 **HyperMPI** 编译 **MPI** 代码，使用的 **MPI** 代码可以在 `/lustre/share/samples/mpi/src/mpihello.c` 中找到。

加载和编译：

```
module load gcc/9.3.0-gcc-4.8.5
module load hmpi/4.0.3-gcc-9.3.0

mpicc mpihello.c -o mpihello
```

提交 **HyperMPI** 编译的应用

准备一个名为 `job_hmpi.slurm` 的作业脚本，用以提交上面编译的 `mpihello` 应用。

```
#!/bin/bash

#SBATCH --job-name=mpihello
#SBATCH --partition=arm128c256g
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 2

ulimit -s unlimited
ulimit -l unlimited

module load gcc/9.3.0-gcc-4.8.5
module load hmpi/4.0.3-gcc-9.3.0

mpirun -np 2 ./mpihello
```

使用以下命令提交作业脚本：

```
sbatch job_hmpi.slurm
```


性能调优

HyperMPI 使用了大量华为自研的集合通信算法，默认情况下，**HyperMPI** 会根据硬件配置和通信规模大小选择最优的算法。

但是在某些情况下，自动选择的算法无法达到最优的性能，需要手动进行集合通信算法的选择。例如，我们在测试时发现，在消息大小较大时，**MPI_Allreduce** 默认选择的通信算法性能不是最优，这时我们可以使用以下选项手动选择 **MPI_Allreduce** 通信算法得到最优性能：

```
mpirun -np ${process num} -x UCX_BUILTIN_ALLREDUCE_ALGORITHM=4 $
→{application}
```

更多相关内容介绍可以参考[HyperMPI 通信算法指定](#) ## 参考资料 [Hyper MPI 简介](#)

bisheng

毕昇编译器是针对鲲鹏平台的高性能编译器。它基于开源 **LLVM** 开发，并进行了优化和改进，同时将 **Flang** 作为默认的 **Fortran** 语言前端编译器。

bisheng 编译器使用方式

- 首先一定要用 **ssh** 登录 **ARM** 节点

```
$ ssh -p 18022 username@202.120.58.248
```

- 使用 **module** 导入应用命令

```
$ module load bisheng/1.3.1-gcc-9.3.0
```

- 毕昇编译器使用举例 (编译运行 **hello.c**)

```
#include <stdio.h>
int main(){
    printf("hello world");
    return 0;
}

clang hello.c -o hello.o

./hello.o
```

参考资料

JDK

简介

JDK 是 **Java** 语言的软件开发工具包，主要用于移动设备、嵌入式设备上的 **java** 应用程序。**JDK** 是整个 **java** 开发的核心，它包含了 **JAVA** 的运行环境 (**JVM+Java** 系统类库) 和 **JAVA** 工具。

JDK 使用说明

思源一号上的 JDK

1. 先创建一个目录 `jdktest` 并进入该目录:

```
mkdir jdktest
cd jdktest
```

2. 在该目录下创建如下测试文件 `jdktest.java`:

```
public class jdktest{
    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};
        // 打印所有数组元素
        for (int i = 0; i < myList.length; i++) {
            System.out.println(myList[i] + " ");
        }
        // 计算所有元素的总和
        double total = 0;
        for (int i = 0; i < myList.length; i++) {
            total += myList[i];
        }
        System.out.println("Total is " + total);
        // 查找最大元素
        double max = myList[0];
        for (int i = 1; i < myList.length; i++) {
            if (myList[i] > max) max = myList[i];
        }
        System.out.println("Max is " + max);
    }
}
```

3. 在该目录下创建如下作业提交脚本 `jdktest.slurm`:

```
#!/bin/bash

#SBATCH --job-name=jdktest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load jdk/12.0.2_10-gcc-11.2.0
javac -cp . -d . jdktest.java
java jdktest
```

4. 使用如下命令提交作业:

```
sbatch jdktest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

pi2.0 上的 JDK

1. 此步骤和上文完全相同;
2. 此步骤和上文完全相同;
3. 在该目录下创建如下作业提交脚本 jdktest.slurm:

```
#!/bin/bash

#SBATCH --job-name=jdktest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load jdk/12.0.2_10-gcc-9.2.0
javac -cp . -d . jdktest.java
java jdktest
```

4. 使用如下命令提交作业:

```
sbatch jdktest.slurm
```

5. 作业完成后在.out 文件中可看到如下结果:

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

参考资料

- [Linux 下编译运行 java 文件](#)
- [菜鸟教程 java](#)

Conda

简介

Conda 是一个可在 Linux、macOS 和 Windows 上运行的开源软件包管理和环境管理系统。Conda 可快速安装、运行和升级软件包及其依赖包。Conda 可在本地计算机上轻松地进行创建、保存、加载和切换环境。它是为 Python 程序创建的，但是也可以打包和分发适用于任何语言的软件。

Conda 作为软件包管理器，可以帮助用户查找和安装软件包。如果用户需要一个使用其他版本的 Python 的软件包，无需切换到其他环境管理器，因为 Conda 也是环境管理器，仅需几个命令，用户就可以设置一个完全独立的环境来运行该不同版本的 Python，同时继续在正常环境中运行用户通常的 Python 版本。

可用的版本

版本	平台	构建方式	模块名
4.7.12.1		Spack	miniconda2/4.7.12.1 思源一号
4.10.3		Spack	<i>miniconda3/4.10.3</i> 思源一号
4.5.12		Spack	<i>conda4aarch64/1.0.0-gcc-4.8.5</i>
4.7.12.1		Spack	miniconda2/4.7.12.1
4.6.14		Spack	miniconda2/4.6.14
4.8.2		Spack	<i>miniconda3/4.8.2</i>
4.7.12.1		Spack	miniconda3/4.7.12.1
4.6.14		Spack	miniconda3/4.6.14

运行示例

思源一号集群 Conda

在思源一号集群上使用如下命令：

```

srun -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
which conda

```

ARM 集群 Conda

在 ARM 节点上使用如下命令：

```
srunk -p arm128c256g -n 4 --pty /bin/bash
module load conda4aarch64/1.0.0-gcc-4.8.5
which conda
```

π 集群 Conda

在 π 集群上使用如下命令：

```
srunk -p small -n 4 --pty /bin/bash
module load miniconda3/4.8.2
which conda
```

Conda 常用命令

```
conda list                # 查看安装了哪些包
conda env list            # 查看当前存在哪些虚拟环境
conda create -n env4test  # 创建一个名为env4test的虚拟环境
source activate env4test  # 激活虚拟环境env4test
conda deactivate          # 退出虚拟环境
conda search bwa -c bioconda # 查找名为bwa的包，并指定bioconda源
conda install bwa -c bioconda -n env4test # 指定从bioconda源中下载安装bwa，安装在env4test虚拟环境中
conda remove -n env4test bwa # 删除虚拟环境中的bwa包
conda remove -n env4test --all # 删除虚拟环境env4test(包括其中的所有的包)
```

迁移 Conda 环境到思源一号

迁移 Conda 环境需要导出环境到文件中，用 `conda env create` 从配置文件中来创建同样的环境。以从 π 集群迁移 Conda 环境到思源一号为例，需要用到 π 集群（旧环境）、sydata 节点（数据互通）、思源一号（新环境）。

```
$  $\pi$  集群
conda env list                # 查看当前存在哪些虚拟环境
source activate pymol         # 激活用户环境
conda list                    # 查看环境的包和软件
conda env export > pymol.yaml # 导出环境到配置文件
$ sydata 节点                 # 数据互通
scp user@data.hpc.sjtu.edu.cn:~/pymol.yaml ~/pymol.yaml
$ 思源一号
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3 # 加载Conda
conda env list                # 查看当前存在哪些虚拟环境
conda env create -f pymol.yaml # 从配置文件创建环境
```

参考教学视频 思源一号 Conda 环境迁移

Conda 创建 Python 环境

Conda 可以方便的创建特定版本的环境，以 Python 为例。首先申请交互的计算资源，再使用 Conda 创建 Python 为 3.7 的环境，如下：

```
srun -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda create --name mypython python==3.7
```

新建一个 hello_python.slurm 的文件，内容为：

```
#!/bin/bash
#SBATCH -J hello-python
#SBATCH -p 64c512g
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1

module load miniconda3/4.10.3
source ~/.bashrc # 如果显示的路径不是预期的，需要这一行
source activate mypython
which python
python -c "print('hello world')"
```

sbatch hello_python.slurm 提交作业，即可用新建的 Python 环境输出结果了。

通过 pip 安装 Python 扩展包

以安装 PyMuPDF 为例，如果 Conda 中找不到相关的 Python 包或者没有需要的版本，可以用 pip 安装。

```
source activate env4test # 激活虚拟环境 env4test
conda search pymupdf # 找不到相关的包
conda search -c tc06580 pymupdf # 指定源搜索，只有 1.17.0 版本的
which pip # _
→ 确定有安装 pip，一般 conda 创建的 Python 环境都会有 pip 的
pip install pymupdf # 使用 pip 安装 Python 扩展包
pip install -r requirements.txt # 使用 pip 批量安装 requirements.txt 中的软件包
pip list | grep -i pymupdf # 安装成功，当前为 1.19.4 版本
```

小技巧

建议特定的一个或几个软件创建一个单独的环境，方便管理与使用。

可以到 Anaconda 页面搜索是否有对应软件的源 <https://anaconda.org/search>

常见问题

1. 在一个 **conda** 环境中同时安装软件 **A** 与软件 **B**，存在 **conflict** 问题

A: 建议新建环境分开进行安装。

2. 软件运行提示缺少 **xxx.so** 库

A: `conda search` 查找不同的版本及 `build` 信息，`conda install` 指定版本及 `build` 进行安装测试。

3. 安装的软件版本不支持 **GPU** 或者不支持 **python**

A: `conda list` 查看已安装的版本及 `build` 信息，确认 `build` 是 **GPU** 或 **python**；`conda install` 指定版本、源及 `build` 进行安装测试。

参考资料

- [Conda 文档](#)

Python

本文档向您展示如何使用 **Miniconda** 在家目录中建立自定义的 **Python** 环境。不同的 **Python** 版本 2 或 3，对应不同的 **Miniconda**。

Miniconda2

加载 **Miniconda2**

```
$ module load miniconda2
```

创建 **conda** 环境来安装所需 **Python** 包。

```
$ conda create --name mypython2 numpy scipy matplotlib ipython jupyter
```

指定 **python** 版本（不指定将默认安装最新版）

```
$ conda create --name mypython2 python==2.7
```

激活 **python** 环境

```
$ source activate mypython2
```

通过 **conda** 或 **pip** 添加更多软件包

```
$ conda install YOUR_PACKAGE  
$ pip install YOUR_PACKAGE
```

Miniconda 3

加载 Miniconda3

```
$ module load miniconda3
```

创建 conda 环境来安装所需 Python 包。

```
$ conda create --name mypython3 numpy scipy matplotlib ipython jupyter
```

激活 python 环境

```
$ source activate mypython3
```

通过 conda 或 pip 添加更多软件包

```
$ conda install YOUR_PACKAGE  
$ pip install YOUR_PACKAGE
```

使用全局预创建的 conda 环境

π 集群已创建全局的 conda 环境, 该环境主要面向生物学用户主要包含 tensorflow-gpu@2.0.0, R@3.6.1, python@3.7.4。使用以下指令激活环境:

```
$ module load miniconda3  
$ source activate /lustre/opt/condaenv/life_sci
```

conda 拓展模块查询方法

```
$ conda list
```

R 拓展模块查询方法

```
$ R  
> installed.packages()
```

使用 Miniconda 向 slurm 提交作业

以下为 python 示例作业脚本, 我们将向 slurm 申请两 cpu 核心, 并在上面通过 python 打印 hello world。

```
#!/bin/bash  
#SBATCH -J hello-python  
#SBATCH -p small  
#SBATCH -o %j.out  
#SBATCH -e %j.err  
#SBATCH -n 2  
  
module load miniconda3
```

(续下页)

(接上页)

```
source activate mypython3

python -c "print('hello world')"
```

我们假定以上脚本内容被写到了 `hello_python.slurm` 中，使用 `sbatch` 指令提交作业。

```
$ sbatch hello_python.slurm
```

参考资料

- [miniconda](#)

PERL

使用 **Miniconda 3** 环境安装 **perl**

加载 Miniconda 3

```
$ module load miniconda3
```

创建 conda 环境

```
$ conda create --name PERL
```

激活 R 环境

```
$ source activate PERL
```

如有必要，请删除现有的 **CPAN** 模块和 `.bashrc` 中 **perl** 相关设置

```
$ rm -rf ~/.perl ~/.cpan
```

上述操作会删除现在已有模块和 **perl** 环境配置信息，请谨慎操作。

在当前环境下安装 **perl** 并设置相关环境变量

```
$ conda install perl
...
$ cpan
...
$ Would you like to configure as much as possible automatically? [yes] yes
...
$ What approach do you want? (Choose 'local::lib', 'sudo' or 'manual')
[local::lib]
...

```

拓展模块 **cpan** 安装示例

```
$ module load miniconda3
$ source activate PERL
$ cpan
cpan> install XML::LibXML
...
cpan> install Getopt::Std
...
cpan> install Encode
```

手动拓展模块下载示例 (不推荐)

```
$ cd /YOUR/PACKAGE/PATH
$ tar xvzf Net-Server-0.97.tar.gz
$ cd Net-Server-0.97
$ perl Makefile.PL
$ make test
```

拓展模块 **conda** 安装示例 (推荐)

```
$ source activate PERL # 进入创建的conda环境
$ conda install -c bioconda perl-pdf-api2
$ perl -MPDF::API2 -e '' # 无报错说明模块安装成功
```

查看已下载的 **perl** 拓展模块

```
#方法一：
$ module load miniconda3
$ source activate PERL
$ instmodsh
> l
Installed modules are:
...
Perl

#方法二：
$ perl doc perllocal
...
```

Perl 的 **SLURM** 作业示例

用法: sbatch job.slurm

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH -J Perl
#SBATCH -p small
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1

module load miniconda3
source activate PERL

perl hello.pl
```

参考资料

- [Set Install path in CPAN](#)
- [perl 模块安装大全](#)

R

简介

R 语言是为数学研究工作者设计的一种数学编程语言，主要用于统计分析、绘图、数据挖掘。R 语言是解释运行的语言（与 C 语言的编译运行不同），它的执行速度比 C 语言慢得多，不利于优化。但它在语法层面提供了更加丰富的数据结构操作并且能够十分方便地输出文字和图形信息，所以它广泛应用于数学尤其是统计学领域。

R 使用说明

在思源一号上自行安装并使用 R

1. 使用 conda 创建虚拟环境，激活虚拟环境并安装 R，然后进入 R 终端：

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda create --name R_test
source activate R_test
conda install -c conda-forge r-base=4.1.3
R
```

2. 在 R 终端执行 R 语句，比如：

```
> print(1+2)
[1] 3
```

在 **pi2.0** 上自行安装并使用 **R**

1. 使用 **conda** 创建虚拟环境，激活虚拟环境并安装 **R**，然后进入 **R** 终端：

```
srunk -p small -n 4 --pty /bin/bash
module load miniconda3/4.7.12.1
conda create --name R_test
source activate R_test
conda install -c conda-forge r-base=4.1.3
R
```

2. 在 **R** 终端执行 **R** 语句，比如：

```
> print(sin(pi/2))
[1] 1
```

参考资料

- [Anaconda 官网](#)
- [如何使用 conda 安装 R 和 R 包](#)

CMake

简介

CMake 是一个跨平台的编译工具，能够输出各种各样的 **makefile** 或者 **project** 文件，并不直接构建出最终的软件，而是生成标准的 **Makefile** 文件，然后再使用 **Make** 进行编译。

CMake 使用说明

在思源一号上使用 **CMake**

1. 首先在自己的家目录下新建一个目录作为测试目录，并进入该目录：

```
mkdir cmake_test
cd cmake_test
```

2. 申请计算资源并加载所需模块：

```
srunk -p 64c512g -n 10 --pty /bin/bash

module purge
module load openmpi/4.1.1-gcc-11.2.0
module load gcc/11.2.0
module load cmake/3.17.1-gcc-11.2.0
```

3. 在当前目录下编写如下测试文件 **cmake_test.cpp**：

```

#include "mpi.h"
#include <iostream>
using namespace std;

int say_hello(int argc, char **argv)
{
    int myid, numprocs;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Get_processor_name(processor_name, &namelen);
    cout << "Hello World! Process " << myid << " of " << numprocs << " on "
    << processor_name << "\n";
    MPI_Finalize();

    return 0;
}

int main(int argc, char **argv)
{
    say_hello(argc, argv);

    return 0;
}

```

4. 在当前目录下编写如下 CMakeLists.txt 文件:

```

cmake_minimum_required(VERSION 3.15)

message(STATUS "The CMAKE_VERSION is ${CMAKE_VERSION}.")

project(cmaketest)

find_package(MPI REQUIRED)

message(STATUS "PROJECT_NAME is ${PROJECT_NAME}")

include_directories ("${MPI_CXX_INCLUDE_DIRS}")

add_executable(${PROJECT_NAME} cmaketest.cpp )

target_link_libraries(${PROJECT_NAME} ${MPI_LIBRARIES})

```

5. 执行以下命令编译源文件。如果编译成功，在 build 目录下会生成 cmaketest 可执行文件:

```

mkdir build
cd build

```

(续下页)

(接上页)

```
cmake ../  
make
```

6. 调用 MPI 运行可执行文件:

```
mpirun -np 10 ./cmaketest
```

7. 此时可以在终端看到如下输出结果:

```
Hello World! Process 1 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 5 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 6 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 7 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 8 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 9 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 0 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 2 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 4 of 10 on node466.pi.sjtu.edu.cn  
Hello World! Process 3 of 10 on node466.pi.sjtu.edu.cn
```

在 pi2.0 上使用 CMake

1. 此步骤和上文完全相同;
2. 申请计算资源并加载所需模块:

```
srunk -p cpu -N 1 --ntasks-per-node 40 --pty /bin/bash  
  
module purge  
module load gcc/9.2.0  
module load cmake/3.18.4-gcc-9.2.0  
module load openmpi/3.1.5-gcc-9.2.0
```

3. 此步骤和上文完全相同;
4. 此步骤和上文完全相同;
5. 此步骤和上文完全相同;
6. 此步骤和上文完全相同;
7. 此步骤和上文完全相同;

参考资料

- [CMake 官网](#)

glibc

简介

glibc 是 GNU 发布的 **libc** 库，即 **c** 运行库。**glibc** 是 **linux** 系统中最底层的 **api**，几乎其它任何运行库都会依赖于 **glibc**。**glibc** 除了封装 **linux** 操作系统所提供的系统服务外，它本身也提供了许多其它一些必要功能服务的实现。

glibc 涉及的系统组件多，无法在超算平台上整体部署高版本 **glibc**，如果需要高版本 **glibc**，可通过源码自行安装。

安装代码

```
srunc -p 64c512g -n 16 --pty /bin/bash
mkdir -p ${HOME}/01.application/12.glibc && cd ${HOME}/01.application/12.
->glibc
wget http://ftp.gnu.org/gnu/glibc/glibc-2.29.tar.gz
tar -zxvf glibc-2.29.tar.gz && cd glibc-2.29
mkdir build && cd build
../configure --prefix=${HOME}/01.application/12.glibc/ --disable-sanity-
->checks
make -j16
make install
cd ${HOME}/01.application/12.glibc/
rm -rf glibc-2.29
export PATH=${HOME}/01.application/12.glibc/bin:$PATH
ldd --version
```

SPOOLES

SPOOLES 的全称是 **SParse Object Oriented Linear Equations Solver**，中文名大概就是面向对象的稀疏线性等式解析器。顾名思义，就是可以用来解稀疏矩阵为参数的线性方程组的数学函数库。所谓面向对象是指的应用了面向对象的封装思想，但实际上 **SPOOLES** 是用非面向对象的 **C** 语言来写的。

最新版是 2.2，支持单线程，多线程和 **MPI** 三种计算模式。

安装教程

首先，下载并解压软件包

```
srunc -p 64c512g -N 1 -n 2 --pty /bin/bash
cd
mkdir spooles
cd spooles
wget http://www.netlib.org/linalg/spooles/spooles.2.2.tgz
tar xf spooles.2.2.tgz && rm -rf spooles.2.2.tgz
```

然后，导入 **MPI** 环境

```
module load openmpi/4.1.1-gcc-8.3.1
```

接下来，修改 **Make.inc** 文件即可

主要修改 CC 和 MPI_INSTALL_DIR 的位置即可

将

```
# CC = /opt/mpi/bin/mpicc
```

修改为

```
CC = /dssg/opt/icelake/linux-centos8-icelake/gcc-8.3.1/openmpi-4.1.1-  
→me4z4iiamxv3l6efci5wcmjd2pk4rvye/bin/mpicc
```

将

```
MPI_INSTALL_DIR = /usr/local/mpich-1.0.13
```

修改为

```
MPI_INSTALL_DIR = /dssg/opt/icelake/linux-centos8-icelake/gcc-8.3.1/openmpi-  
→4.1.1-me4z4iiamxv3l6efci5wcmjd2pk4rvye
```

将

```
MPI_LIB_PATH = -L$(MPI_INSTALL_DIR)/lib/solaris/ch_p4
```

修改为

```
MPI_LIB_PATH = -L/dssg/opt/icelake/linux-centos8-icelake/gcc-8.3.1/openmpi-  
→4.1.1-me4z4iiamxv3l6efci5wcmjd2pk4rvye/lib
```

将

```
# MPI_LIBS = $(MPI_LIB_PATH) -lmpi -lpthread
```

修改为

```
MPI_LIBS = $(MPI_LIB_PATH) -lmpi -lpthread
```

最后，执行编译命令

```
make lib  
make
```

成功的标志

生成 spooles.a 等库文件，既代表编译成功

参考资料

- 官方网站 <https://netlib.org/linalg/spooles/spooles.2.2.html>

Deal.II

Deal.II 作为一个开源有限元软件，其本质上是一个 C++ 软件库，用以支持创建有限元代码。DEAL 的全称为 **Differential Equations Analysis Library**（微分方程分析库），而以 II 作为后缀则是因为这一软件库是微分方程分析库的后续工作，其主要的功能是创建 C++ 软件库，使用自适应有限元来解决针对偏微分方程的计算。它使用最先进的编程技术为用户提供所需的复杂数据结构和算法的现代接口

可用版本

版本	平台	构建方式	名称
9.3.3		spack	dealii/9.3.3-gcc-11.2.0-hdf5-openblas 思源

Deal.II 可提供的内容

- 使用统一的接口来支持一、二、三个空间维度，该接口允许编写几乎与唯独无关的程序。
- 处理局部的细化网格，包括基于局部误差指示器和误差估计器的不同自适应网格策略。
- 支持多种有限元元素：连续和不连续的任意阶拉格朗日单元 (Lagrange elements)、内德勒克和拉维亚特-托马斯单元 (Nedelec and Raviart-Thomas elements) 以及其他单元。
- 通过 MPI 跨界点并行计算。
- 允许快速访问的所需信息，包含教程、报告以及一些接口文档，其中解释了所有的类、函数以及变量，所有的文件与库同时提供，安装后即可在计算机中获取。
- 使访问复杂数据结构和算法尽可能透明的软件技术。
- 一个完整的独立线性代数库。
- 支持多种输出格式。
- 对各种计算机和编译器的可移植支持。
- 在开源许可下免费获取源代码。

思源一号上 Deal.II 的 lib 和 include 库的路径

```
/dssg/opt/icelake/linux-centos8-icelake/gcc-11.2.0/dealii-9.3.3-
→pyqxfz27lwca6qu5r6mc6zoh65pl2jad/lib
/dssg/opt/icelake/linux-centos8-icelake/gcc-11.2.0/dealii-9.3.3-
→pyqxfz27lwca6qu5r6mc6zoh65pl2jad/include
```

如何在个人目录下自编译 Deal.II 软件

```

srun -p 64c512g -N 1 -n 6 --pty /bin/bash
mkdir -p ~/src/dealii/src
mkdir -p ~/src/dealii/install

module load mpich/3.4.1-gcc-9.2.0 gcc/9.2.0 boost/1.70.0-gcc-9.2.0
export CC=mpicc
export CXX=mpicxx
export FC=mpif90

cd ~/src/dealii/src
tar xf petsc-3.13.1.tar.gz
cd petsc-3.13.1
export PETSC_DIR=`pwd`
export PETSC_ARCH=x86_64
python3 config/configure.py --with-shared-libraries=1 --with-x=0 --with-
→mpi=1 --with-mpi-dir=/lustre/opt/cascadelake/linux-centos7-cascadelake/
→gcc-9.2.0/mpich-3.4.1-76cdhzhxr7wp5kqkwirehvwpo7oe6lc --download-hypre=1
→--download-fblaslapack=1
make PETSC_DIR=~/src/dealii/src/petsc-3.13.1 PETSC_ARCH=x86_64 all

export PETSC_DIR=~/src/dealii/src/petsc-3.13.1
export PETSC_ARCH=x86_64
export PATH=$PATH:~/src/dealii/src/petsc-3.13.1/x86_64/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/src/dealii/src/petsc-3.13.1/x86_
→64/lib:~/src/dealii/src/petsc-3.13.1/x86_64/externalpackages/git.
→fblaslapack

cd ~/src/dealii/src
tar xf p4est-2.2.tar.gz
cd p4est-2.2/
./configure --prefix=~/src/dealii/install/p4est BLAS_LIBS=~/src/dealii/src/
→petsc-3.13.1/x86_64/externalpackages/git.fblaslapack/libfblas.a LAPACK_
→LIBS=~/src/dealii/src/petsc-3.13.1/x86_64/externalpackages/git.
→fblaslapack/libflapack.a --enable-mpi
make
make install

export PATH=$PATH:~/src/dealii/src/petsc-3.13.1/x86_64/bin:~/src/dealii/
→install/p4est/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/src/dealii/src/petsc-3.13.1/x86_
→64/lib:~/src/dealii/src/petsc-3.13.1/x86_64/externalpackages/git.
→fblaslapack:~/src/dealii/install/p4est/lib

cd ~/src/dealii/src
unzip Trilinos.zip
cd Trilinos/Trilinos-trilinos-release-12-4-1/
mkdir build && cd build
cmake -DTrilinos_ENABLE_Amesos=ON -DTrilinos_ENABLE_Epetra=ON -DTrilinos_

```

(续下页)

(接上页)

```

→ENABLE_EpetraExt=ON -DTrilinos_ENABLE_Ipack=ON -DTrilinos_ENABLE_
→AztecOO=ON -DTrilinos_ENABLE_Sacado=ON -DTrilinos_ENABLE_Teuchos=ON -
→DTrilinos_ENABLE_MueLu=ON -DTrilinos_ENABLE_ML=ON -DTrilinos_ENABLE_
→ROL=ON -DTrilinos_ENABLE_Tpetra=ON -DTrilinos_ENABLE_COMPLEX_DOUBLE=ON -
→DTrilinos_ENABLE_COMPLEX_FLOAT=ON -DTrilinos_ENABLE_Zoltan=OFF -DTrilinos_
→VERBOSE_CONFIGURE=OFF -DTPL_ENABLE_MPI=ON -DBUILD_SHARED_LIBS=ON -DCMAKE_
→VERBOSE_MAKEFILE=OFF -DCMAKE_BUILD_TYPE=RELEASE -DBLAS_LIBRARY_
→NAMES:STRING=libfblas.a -DBLAS_LIBRARY_DIRS:STRING=~ /src/dealii/src/petsc-
→3.13.1/x86_64/externalpackages/git.fblaslapack -DLAPACK_LIBRARY_
→NAMES:STRING=libflapack.a -DLAPACK_LIBRARY_DIRS:STRING=~ /src/dealii/src/
→petsc-3.13.1/x86_64/externalpackages/git.fblaslapack -DCMAKE_INSTALL_
→PREFIX=~ /src/dealii/install/trilinos ..
make install

cd ~/src/dealii/src
tar xf zlib-1.2.12.tar.gz
cd zlib-1.2.12/
./configure --prefix=~ /src/dealii/install/zlib
make install

export PATH=$PATH:~/src/dealii/src/petsc-3.13.1/x86_64/bin:~/src/dealii/
→install/p4est/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/src/dealii/src/petsc-3.13.1/x86_
→64/lib:~/src/dealii/src/petsc-3.13.1/x86_64/externalpackages/git.
→fblaslapack:~/src/dealii/install/p4est/lib:~/src/dealii/install/zlib/lib

cd ~/src/dealii/src
tar xf metis-5.1.0.tar.gz
cd metis-5.1.0/
make config prefix=~ /src/dealii/install/metis
make install

export PATH=$PATH:~/src/dealii/src/petsc-3.13.1/x86_64/bin:~/src/dealii/
→install/p4est/bin:~/src/dealii/install/metis/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/src/dealii/src/petsc-3.13.1/x86_
→64/lib:~/src/dealii/src/petsc-3.13.1/x86_64/externalpackages/git.
→fblaslapack:~/src/dealii/install/p4est/lib:~/src/dealii/install/zlib/lib:~
→/src/dealii/install/metis/lib

cd ~/src/dealii/src
tar xf dealii-9.2.0.tar.gz
cd dealii-9.2.0
mkdir build
cd build
cmake -DDEAL_II_WITH_MPI=ON -DMETIS_DIR=~ /src/dealii/install/metis -DP4EST_
→DIR=~ /src/dealii/install/p4est -DDEAL_II_WITH_P4EST=ON -DZLIB_DIR=~ /src/
→dealii/install/zlib -DPETSC_DIR=~ /src/dealii/src/petsc-3.13.1 -DPETSC_
→ARCH=x86_64 -DTRILINOS=~ /src/dealii/install/trilinos -DCMAKE_INSTALL_
→PREFIX=~ /src/dealii/install/deal ..

```

(续下页)

(接上页)

```
make --jobs=8 install
make test

export PATH=$PATH:~/src/dealii/src/petsc-3.13.1/x86_64/bin:~/src/dealii/
→install/p4est/bin:~/src/dealii/install/metis/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/src/dealii/src/petsc-3.13.1/x86_
→64/lib:~/src/dealii/src/petsc-3.13.1/x86_64/externalpackages/git.
→fblaslapack:~/src/dealii/install/p4est/lib:~/src/dealii/install/zlib/lib:~
→/src/dealii/install/metis/lib:~/src/dealii/install/deal/lib
```

参考资料

- 官方网站 <https://www.dealii.org/>

nvidia-top

简介

nvidia-top 是一款交互式的 NVIDIA GPU 设备性能、资源、进程的实时监测工具。nvidia-top 具有以下特点：

- 监控信息丰富：资源监控信息更全面，表现形式更直观
- 实时查看：作为资源监控器，它支持条形图和历史图、监控信息排序和监控信息过滤、GPU 进程及其父进程的树视图显示
- 交互式：可在监控模式下响应用户输入（来自键盘或鼠标），比 gpustat 和 py3nvm1 更具优势
- 可集成：易于集成到其它应用程序中，而不仅仅是监控功能

您可以在登录到 GPU 节点终端后使用 nvidia-top 命令来查看 GPU 使用信息。更多使用方法参见文档：<https://nvidia-top.readthedocs.io/en/latest/>

GPU 使用信息展示

在 DGX-2 节点上使用 nvidia-top 命令：

Fri Oct 13 15:03:06 2023 (Press h for help or q to quit)									
NVTOP 1.0.0		Driver Version: 525.105.17			CUDA Driver Version: 12.0				
GPU	Fan	Temp	Perf	Pwr:Usrg/Cap	Memory-Usage		GPU-Util	Compute M.	
0	N/A	34C	P0	53W / 350W	128KiB / 32.00GiB		0%	Default	MEM: 0.0%
1	N/A	34C	P0	49W / 350W	5MiB / 32.00GiB		0%	Default	UTL: 0%
2	N/A	47C	P0	140W / 350W	355MiB / 32.00GiB		43%	Default	MEM: 1.1%
3	N/A	39C	P0	49W / 350W	128KiB / 32.00GiB		0%	Default	UTL: 0%
4	N/A	43C	P0	136W / 350W	1141MiB / 32.00GiB		56%	Default	MEM: 3.5%
5	N/A	44C	P0	71W / 350W	28.60GiB / 32.00GiB		0%	Default	UTL: 0%
6	N/A	33C	P0	52W / 350W	128KiB / 32.00GiB		0%	Default	MEM: 0.0%
7	N/A	38C	P0	51W / 350W	128KiB / 32.00GiB		0%	Default	UTL: 0%
8	N/A	51C	P0	243W / 350W	30.94GiB / 32.00GiB		97%	Default	MEM: 97%
9	N/A	37C	P0	73W / 350W	28.60GiB / 32.00GiB		0%	Default	UTL: 0%
10	N/A	50C	P0	94W / 350W	29.15GiB / 32.00GiB		0%	Default	MEM: 91.1%
11	N/A	48C	P0	71W / 350W	28.60GiB / 32.00GiB		0%	Default	UTL: 0%
12	N/A	34C	P0	68W / 350W	28.60GiB / 32.00GiB		0%	Default	MEM: 89.4%
13	N/A	41C	P0	117W / 350W	645MiB / 32.00GiB		43%	Default	UTL: 43%
14	N/A	67C	P0	350W / 350W	1693MiB / 32.00GiB		98%	Default	MEM: 98%
15	N/A	41C	P0	50W / 350W	128KiB / 32.00GiB		0%	Default	UTL: 0%
[CPU: 19.2%] [MEM: 13.0%] [Load Average: 15.74 16.66 21.31] [CAUTION: SUPERUSER LOGGED-IN.] [SWP: 0.0%]									
Processes: root@vol08.pi.sjtu.edu.cn									
GPU	PID	USER	GPU-MEM	%SM	%CPU	%MEM	TIME	COMMAND	
2	82441	C	350MiB	41	99.9	0.0	76:07:28	gmx mdrun -s md.tpr -cpi md.cpt -deffnm md -ntmpi 1 -ntomp 1 -nb gpu	
4	79129	C	1136MiB	59	109.3	0.1	2:31:03	lmp -i in.lammps	
5	186310	C	28.59GiB	0	0.0	0.1	1:05:29	/opt/conda/bin/python	
8	92886	C	30.94GiB	95	27.9	0.2	14:24	/usr/bin/python	
9	99126	C	28.59GiB	0	0.0	0.1	10:05	/opt/conda/bin/python	
10	20786	C	29.14GiB	12	99.7	0.3	31:41	/opt/conda/bin/python	
11	57581	C	28.59GiB	0	0.0	0.1	24:33	/opt/conda/bin/python	
12	101233	C	28.59GiB	0	0.0	0.1	9:20	/opt/conda/bin/python	
13	96133	C	640MiB	42	100.2	0.0	12:11	pnmend.cuda	
14	234936	C	1688MiB	97	100.1	0.1	23:10:50	/opt/amber20/bin/pnmend.cuda	

在 A100 节点上使用 nvtop 命令:

Fri Oct 13 15:06:43 2023										(Press h for help or q to quit)												
NVTOP 1.0.0				Driver Version: 525.85.12				CUDA Driver Version: 12.0														
GPU	Name	Persistence-M	Bus-Id	Disp-A	MIG M.	Uncorr. ECC	GPU-Util	Compute M.														
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage																		
0	A100-SXM4-40GB	On	00000000:31:00:0	Off	Disabled	0	MEM: 92.8%															
N/A	53C	P0	184W / 400W	37.13GiB / 40.00GiB	6%	Default	UTL: 6%															
1	A100-SXM4-40GB	On	00000000:40:00:0	Off	Disabled	0	MEM: 88.6%															
N/A	54C	P0	200W / 400W	35.45GiB / 40.00GiB	0%	Default	UTL: 0%															
2	A100-SXM4-40GB	On	00000000:CA:00:0	Off	Disabled	0	MEM: 0.0%															
N/A	39C	P0	51W / 400W	128KiB / 40.00GiB	0%	Default	UTL: 0%															
3	A100-SXM4-40GB	On	00000000:E3:00:0	Off	Disabled	0	MEM: 0.0%															
N/A	39C	P0	51W / 400W	128KiB / 40.00GiB	0%	Default	UTL: 0%															
Load Average: 4.90 5.26 5.62							AVG GPU MEM: 45.4%															
CPU: 15.1%																						
-120s -60s -30s							-120s -60s -30s															
MEM: 4.5%							AVG GPU UTL: 25.8%															
SWP: 0.0%																						
CAUTION: SUPERUSER LOGGED-IN.																						
Processes: root@gpu07.pi.sjtu.edu.cn																						
GPU	PID	USER	GPU-MEM	%SM	%CPU	%MEM	TIME	COMMAND														
0	2733590	C	37.13GiB	83	333.1	1.0	22:17:27	python3														
1	2733590	C	35.45GiB	79	4122	1.0	22:17:27	python3														

参考

- 官方仓库地址: <https://github.com/XuehaiPan/nvitop>
- 官方文档地址: <https://nvitop.readthedocs.io/en/latest/>

TeX Live

TeX Live 是一个完整的 TeX/LaTeX 发行版, 提供了各种排版工具和宏包, 适用于编写科学论文、技术报告、书籍等高质量文档。本文将介绍 TeX Live 的基本使用方法, 包括基本命令以及在 Matplotlib 中使用 LaTeX 格式生成图表。

目录

- 常用命令
- 配置和更新
- 在 Matplotlib 中使用 LaTeX
- 将 tex 转换成 pdf 示例

常用命令

在思源一号上调用 texlive

```
module load texlive
```

调用 texlive 后, 可以使用以下命令处理 .tex 文件:

1. **pdflatex**: 将 LaTeX 源文件编译为 PDF 格式。

```
pdflatex yourfile.tex
```

2. **xelatex**: 使用 Unicode 编码支持的引擎编译 LaTeX 文档, 支持多语言字体。

```
xelatex yourfile.tex
```

3. **lualatex**: 使用 LuaTeX 引擎编译 LaTeX 文档, 具有更灵活的扩展性。

```
lualatex yourfile.tex
```

在 Matplotlib 中使用 LaTeX

Matplotlib 是一个强大的 Python 库, 用于绘制高质量图表。结合 TeX Live, 可以使用 LaTeX 语法在图表中渲染数学公式和文本, 提升图表的专业性。以下是如何在 Matplotlib 中启用 LaTeX 支持的步骤:

1. 确保 TeX Live 已正确调用
2. 在 conda 环境中安装 Matplotlib 及 numpy 库:

```
conda install matplotlib numpy
```

3. 在 Python 脚本中启用 LaTeX 渲染:

```
import matplotlib.pyplot as plt
import numpy as np

# 启用 LaTeX 渲染
plt.rc('text', usetex=True)
plt.rc('font', family='serif')

# 创建示例数据
x = np.linspace(0, 10, 100)
y = np.sin(x)

# 绘制带有 LaTeX 公式的图表
plt.plot(x, y, label=r'$\sin(x)$')
plt.title(r'Plot of $\sin(x)$', fontsize=16)
plt.xlabel(r'$x$', fontsize=14)
plt.ylabel(r'$\sin(x)$', fontsize=14)
plt.legend()

# 保存图表
plt.savefig('latex_plot.pdf')
```

在该示例中，LaTeX 语法用于图表的标题、坐标轴标签和图例。Matplotlib 会通过调用系统中的 LaTeX 来渲染这些文本。生成的 PDF 图表具有高分辨率，适合用于出版物和论文中。

将 tex 转换成 pdf 示例

以下是一个简单的 LaTeX 示例，演示如何创建一个包含标题、段落和数学公式的文档：

```
\documentclass{article}

% Title Information
\title{Sample TeX Document}
\author{Your Name}
\date{\today} % You can also manually enter the date here, e.g., {October_
→8, 2024}

\begin{document}

% Create the title
\maketitle

\begin{abstract}
This is a simple example of a LaTeX document. The document demonstrates_
```

(续下页)

(接上页)

```

↪basic usage of LaTeX formatting for creating a well-structured PDF file.
\end{abstract}

\section{Introduction}

This is the introduction section. You can introduce the purpose of your
↪document and provide any relevant background information. LaTeX is widely
↪used in academic and scientific writing due to its powerful features for
↪handling large documents and mathematical notation.

\section{Main Content}

In this section, you can add the main content of your document. You can use
↪subsections, equations, tables, and figures as needed.

\subsection{A Subsection Example}

Here is an example of a subsection. You can also create lists and tables in
↪LaTeX. For instance:

\begin{itemize}
  \item First item
  \item Second item
  \item Third item
\end{itemize}

\subsection{Equation Example}

LaTeX handles mathematical notation very well. For example, the quadratic
↪formula is:

\[
x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
\]

\section{Conclusion}

In the conclusion, summarize the key points of the document and any future
↪work or steps to take. LaTeX makes it easy to organize and structure your
↪writing clearly.

\end{document}

```

将此内容保存为 *example.tex*，然后使用以下命令编译：

```
pdflatex example.tex
```

编译成功后，将生成一个 *example.pdf* 文件。

更多信息

- [TeX Live 官方网站](#):

JobTop 作业监控工具

JobTop 是一个用于监控作业的命令行工具，提供了直观的界面来查看作业 CPU 平均负载和内存的使用情况。使用该工具可查看正在运行作业的资源使用情况，以及最近两个月内的历史作业资源使用情况。

主要功能

- 显示作业的相关信息，如节点、开始时间、结束时间、状态、最大内存使用率、最大 CPU 负载
- 保存作业的资源图，包括 CPU 和内存的使用情况

使用方法

在终端中直接运行:

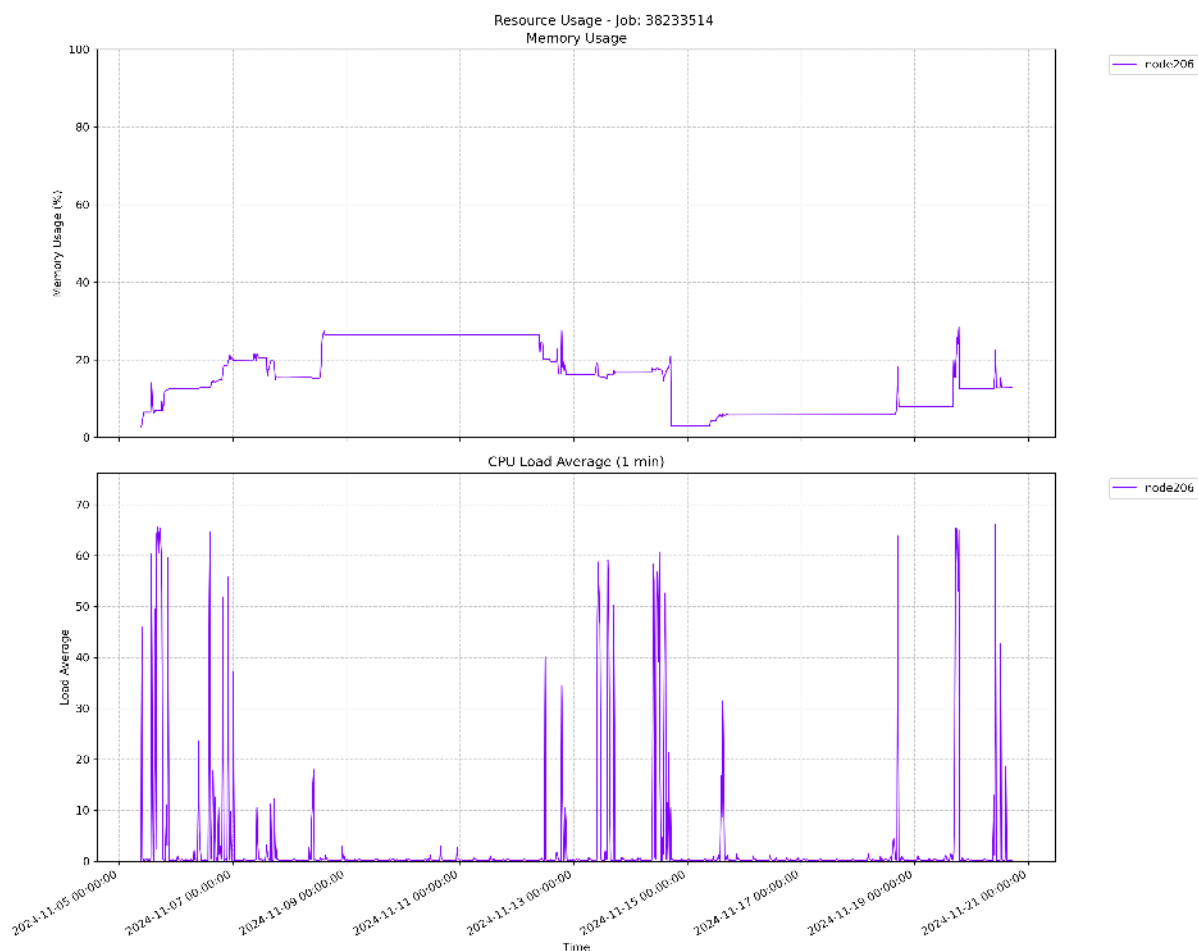
```
module load jobtop
jobtop 作业ID
如: jobtop 38233514
```

终端输出:

```
Job info:
Job ID: 38233514
Nodes: ['node206']
Start time: 2024-11-05 09:15:27
End time: 2024-11-21 10:03:48
State: RUNNING
Max Memory Usage: 49.98%
Max CPU Load: 376.38

Job resources Figure saved as: 38233514_cpu_mem.png
```

作业资源图



注意：

- 作业资源图保存为 PNG 格式，文件名格式为：作业 ID_cpu_mem.png
- 作业资源图保存路径为当前目录

CPU 平均负载 (1m)

具体指的是过去 1 分钟内系统中处于以下两种状态的平均任务数量，并计算其指数加权移动平均值得到的。

- 正在运行的任务：正在 CPU 上执行的进程。
- 等待 CPU 的任务：因 CPU 资源不足而排队的进程。

例如：

- 在 64 核系统中，负载 64 表示 CPU 正满负荷工作。
- 在 40 核系统中，负载 10 表示 CPU 的使用率仅为 25%。
- 负载值大于核心数：意味着任务队列中等待的任务数超过了 CPU 的处理能力，系统可能出现性能瓶颈。
- 负载值小于核心数：CPU 还有空闲时间，系统负载较轻。

4.10.3 基准测试

HPL

简介

HPL (The High-Performance Linpack Benchmark) 是测试高性能计算集群系统浮点性能的基准程序。HPL 通过对高性能计算集群采用高斯消元法求解一元 N 次稠密线性代数方程组的测试，评价高性能计算集群的浮点计算能力。

浮点计算峰值是指计算机每秒可以完成的浮点计算次数，包括理论浮点峰值和实测浮点峰值。理论浮点峰值是该计算机理论上每秒可以完成的浮点计算次数，主要由 CPU 的主频决定。理论浮点峰值 = CPU 主频 × CPU 核数 × CPU 每周期执行浮点运算的次数。本文将为您介绍如何利用 HPL 测试实测浮点峰值。

导入 HPL 环境

版本	平台	构建方式	模块名
2.3		spack	oneapi/2021.4.0 思源一号
2.3		spack	oneapi/2021.4.0 π2.0
2.3		spack	hpl/2.3-gcc-9.3.0-openblas-openmpi ARM 集群

```
mkdir ~/HPL && cd ~/HPL
module load oneapi/2021.4.0
cp -r $MKLRROOT/benchmarks/mp_linpack ./
cd mp_linpack/
```

文件目录结构如下所示：

```
[hpc@login3 80cores]$ tree mp_linpack/
mp_linpack/
├── build.sh
├── COPYRIGHT
├── HPL.dat
├── HPL_main.c
├── libhpl_intel64.a
├── readme.txt
├── runme_intel64_dynamic
├── runme_intel64_prv
├── run.slurm
├── xhpl_intel64_dynamic
└── xhpl_intel64_dynamic_outputs.txt
```

测试平台

- $\pi 2.0$
- 思源一号平台
- ARM 平台

 $\pi 2.0$

Intel HPL 使用时建议在每一个 NUMA Socket 启动一个 MPI 进程，然后再由 MPI 进程启动与 Socket 核心数匹配的计算线程。由于 Intel HPL 不使用 OpenMP 库，因此无法通过 OMP 环境变量控制计算线程数。

$\pi 2.0$ 上计算节点配置信息：双路 Intel 6248 节点，每个 CPU Socket 启动 1 个 MPI 进程，共启动 2 个 MPI 进程。

首先需要配置文件内容：

计算节点内存为 180G，将输入文件 HPL.dat 中的问题规模 N_s 调整至内存空间的 80% 左右 $0.8 * \sqrt{\text{mem} * 1024 * 1024 * 1024 * \text{nodes} / 8}$ 。本算例使用了两个节点，这里使用 sed 将 N_s 替换为 176640。

```
$ sed -i -e 's/. *Ns.*/176640\ Ns/' HPL.dat
```

然后调整 HPL.dat 的 P_s 和 Q_s 值，使其乘积等于 MPI 进程总数。这里使用 sed 将 P_s 和 Q_s 值分别设置为 2、2，乘积等于线程总数 2。

```
$ sed -i -e 's/. *\ P_s.*/2\ P_s/' HPL.dat
$ sed -i -e 's/. *\ Q_s.*/2\ Q_s/' HPL.dat
```

接下来将 runme_intel64_dynamic 中的 MPI 总数改为 4

```
sed -i 's/MPI_PROC_NUM=2/MPI_PROC_NUM=4/' runme_intel64_dynamic
```

提交如下运行脚本：

```
#!/bin/bash

#SBATCH --job-name=hpl2node
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=20
#SBATCH --exclusive

ulimit -s unlimited
ulimit -l unlimited

module load oneapi
```

(续下页)

(接上页)

```
./runme_intel64_dynamic
```

使用 `-n` 指定 **MPI** 进程总数, `--ntasks-per-node` 指定每节点启动的 **MPI** 进程数, `--cpus-per-task` 指定每个 **MPI** 进程使用的 **CPU** 核心数

使用如下命令提交脚本:

```
sbatch run.slurm
```

运行结果如下所示:

```
=====
T/V          N    NB    P    Q          Time
↪Gflops
-----
↪-----
WC00C2R100000    176640    256     2     2          973.53      3.
↪77426e+03
```

思源一号

文件参数的配置, 参考上述规则, 将 `N`、`P`、`Q` 等参数使用 `sed` 命令更改如下:

```
sed -i -e 's/. *Ns.*/209510\ Ns/' HPL.dat
sed -i -e 's/. *Ps.*/2\ Ps/' HPL.dat
sed -i -e 's/. *Qs.*/2\ Qs/' HPL.dat
sed -i 's/MPI_PROC_NUM=2/MPI_PROC_NUM=4/' runme_intel64_dynamic
```

运行脚本如下所示:

```
#!/bin/bash

#SBATCH --job-name=hpl2node
#SBATCH --partition=64c256g
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=32
#SBATCH --exclusive

ulimit -s unlimited
ulimit -l unlimited

module load oneapi

./runme_intel64_dynamic
```

ARM 集群

首先，复制算例到本地。

```
$ mkdir arm_hpl
$ cd arm_hpl
$ cp -r /lustre/opt/kunpeng920/linux-centos7-aarch64/gcc-9.3.0/hpl-2.3-
  ↪svu3iccgwr6whf7b2fcj7mbkaipbffye/bin/* ./
```

然后，将输入文件 HPL.dat 中的问题规模 N_s 调整至内存空间 256G 的 80% 左右。这里使用 sed 将 N_s 替换为 147840。

```
$ sed -i -e 's/.*Ns.*/147840\ Ns/' HPL.dat
```

将 NB 更改为经验值 384。

```
$ sed -i -e 's/.*NBs.*/384\ NBs/' HPL.dat
```

接下来，将 P_s 和 Q_s 值分别设置为 8、16，乘积等于 CPU 总核数 128。

```
$ sed -i -e 's/.*\ Ps.*/8\ Ps/' HPL.dat
$ sed -i -e 's/.*\ Qs.*/16\ Qs/' HPL.dat
```

使用 sbatch hpl.slurm 提交作业，其中 N 代表节点总数，ntasks-per-node 代表每个节点使用的总核数。

```
#!/bin/bash

#SBATCH --job-name=arm_hpl
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export OMP_NUM_THREADS=1
module load openmpi/4.0.3-gcc-9.2.0
module load hpl/2.3-gcc-9.3.0-openblas-openmpi
ulimit -s unlimited
ulimit -l unlimited
mpirun -np $SLURM_NTASKS xhpl
```

运行结果如下所示：

```
=====
T/V          N    NB    P    Q          Time          ↪
↪Gflops
-----
↪-----
```

(续下页)

(接上页)

WR00L2L2	147840	384	8	16	2489.13	8.
↪ 6545e+02						

运行结果时间比较

π2.0 集群

oneapi/2021.4.0				
核数	40	80	160	
Time(s)	705.40	973.53	1439.61	
Gflops	1847.25	3774.26	7117.28	

思源一号集群

oneapi/2021.4.0				
核数	64	128	256	
Time(s)	1548.69	2247.28	3111.47	
Gflops	3958.81	7728.58	15798.2	

参考资料

- Running the Intel Distribution for LINPACK Benchmark
<https://www.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top/intel-oneapi-math-kernel-library-benchmarks/intel-distribution-for-linpack-benchmark-1/run-the-intel-distribution-for-linpack-benchmark.html>
- HOW DO I TUNE MY HPL.DAT FILE? https://www.advancedclustering.com/act_kb/tune-hpl-dat-file/

OSU Benchmarks

简介

OSU Benchmarks

超算平台通过模块提供了 OSU Benchmarks 模块，可用 `module load` 指令加载。

版本	平台	构建方式	模块名
5.7.1		spack	osu-micro-benchmarks/5.7.1-gcc-11.2.0-openmpi 思源一号
5.7.1		spack	osu-micro-benchmarks/5.7.1-gcc-9.2.0-openmpi-4.0.5
5.6.3		spack	osu-micro-benchmarks/5.6.3-gcc-9.3.0-openmpi

集群上的 OSU

- 思源一号上的 *OSU*
- $\pi 2.0$ 上的 *OSU*
- *ARM* 上的 *OSU*

思源一号上的 OSU 基准测试

思源一号使用 **osu_latency** 测量点对点通信延迟

作业脚本 `osu_latency.slurm` 在两个节点上各启动一个 **MPI** 进程，测量两个 **MPI** 进程之间的通信延迟。

```
#!/bin/bash
#SBATCH --job-name=osu_latency
#SBATCH --partition=64c512g
#SBATCH -n 2
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load openmpi
module load osu-micro-benchmarks/5.7.1-gcc-11.2.0-openmpi

mpirun osu_latency
```

思源一号上使用 **osu_mbw_mr** 测量点对点通信带宽

作业脚本 `osu_bw.slurm` 在两个节点上各启动一个 **MPI** 进程，测量两个 **MPI** 进程之间的通信带宽。

```
#!/bin/bash
#SBATCH --job-name=osu_latency
#SBATCH --partition=64c512g
#SBATCH -n 2
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive
```

(续下页)

(接上页)

```
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load openmpi
module load osu-micro-benchmarks/5.7.1-gcc-11.2.0-openmpi

mpirun osu_mbw_mr
```

思源一号上使用 **osu_bcast** 测量广播延迟

需要注意广播延迟和参与测试的节点数量正相关，如果需要对比性能，要保证测试作业所用的节点数一致。

```
#!/bin/bash
#SBATCH --job-name=osu_latency
#SBATCH --partition=64c512g
#SBATCH -n 4
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load openmpi
module load osu-micro-benchmarks/5.7.1-gcc-11.2.0-openmpi

mpirun osu_bcast
```

π2.0 上的 OSU 基准测试

π2.0 上使用 **osu_latency** 测量点对点通信延迟

作业脚本 `osu_latency.slurm` 在两个节点上各启动一个 MPI 进程，测量两个 MPI 进程之间的通信延迟。

```
#!/bin/bash
#SBATCH --job-name=osu
#SBATCH --partition=cpu
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 2
#SBATCH --ntasks-per-node=1

module load gcc/9.2.0
module load openmpi/4.0.5-gcc-9.2.0
module load osu-micro-benchmarks/5.7.1-gcc-9.2.0-openmpi-4.0.5

mpirun osu_latency
```

π2.0 上使用 `osu_mbw_mr` 测量点对点通信带宽

作业脚本 `osu_bw.slurm` 在两个节点上各启动一个 **MPI** 进程，测量两个 **MPI** 进程之间的通信带宽。

```
#!/bin/bash
#SBATCH --job-name=osu
#SBATCH --partition=cpu
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 2
#SBATCH --ntasks-per-node=1

module load gcc/9.2.0
module load openmpi/4.0.5-gcc-9.2.0
module load osu-micro-benchmarks/5.7.1-gcc-9.2.0-openmpi-4.0.5

mpirun osu_mbw_mr
```

π2.0 上使用 `osu_bcast` 测量广播延迟

需要注意广播延迟和参与测试的节点数量正相关，如果需要对比性能，要保证测试作业所用的节点数一致。

```
#!/bin/bash
#SBATCH --job-name=osu
#SBATCH --partition=cpu
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 4
#SBATCH --ntasks-per-node=1

module load gcc/9.2.0
module load openmpi/4.0.5-gcc-9.2.0
module load osu-micro-benchmarks/5.7.1-gcc-9.2.0-openmpi-4.0.5

mpirun osu_bcast
```

ARM 上的 **OSU** 基准测试

ARM 上使用 `osu_latency` 测量点对点通信延迟

作业脚本 `osu_latency.slurm` 在两个节点上各启动一个 **MPI** 进程，测量两个 **MPI** 进程之间的通信延迟。

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=osu_latency
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 2
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive

ulimit -l unlimited
ulimit -s unlimited

module load osu-micro-benchmarks/5.6.3-gcc-9.3.0-openmpi

srun --mpi=pmi2 osu_latency
```

ARM 上使用 **osu_mbw_mr** 测量点对点通信带宽

作业脚本 `osu_bw.slurm` 在两个节点上各启动一个 **MPI** 进程，测量两个 **MPI** 进程之间的通信带宽。

```
#!/bin/bash

#SBATCH --job-name=osu_bw
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 2
#SBATCH --ntasks-per-node=1
#SBATCH --exclusive

ulimit -l unlimited
ulimit -s unlimited

module load osu-micro-benchmarks/5.6.3-gcc-9.3.0-openmpi

srun --mpi=pmi2 osu_mbw_mr
```

ARM 上使用 **osu_bcast** 测量广播延迟

需要注意广播延迟和参与测试的节点数量正相关，如果需要对比性能，要保证测试作业所用的节点数一致。

```
#!/bin/bash

#SBATCH --job-name=osu_bw
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -n 4
#SBATCH --ntasks-per-node=1
```

(续下页)

(接上页)

```
#SBATCH --exclusive

ulimit -l unlimited
ulimit -s unlimited

module load osu-micro-benchmarks/5.6.3-gcc-9.3.0-openmpi

srun --mpi=pmi2 osu_bcast
```

测试结果

OSU MPI Latency

# OSU MPI Latency Test			
	思源一号 v5.7.1	π2.0 v5.7.1	ARM v5.6.3
# Size	Latency (us)	Latency (us)	Latency (us)
0	0.79	1.55	1.27
1	0.79	1.34	1.25
2	0.79	1.29	1.24
4	0.79	1.25	1.25
8	0.78	1.24	1.25
16	0.79	1.59	1.26
32	0.82	1.59	1.29
64	0.91	1.50	1.43
128	0.95	1.51	1.47
256	1.24	1.56	1.95
512	1.26	1.63	2.23
1024	1.37	1.79	2.77
2048	2.08	2.11	3.61
4096	2.80	2.71	4.86
8192	3.85	3.98	7.20
16384	5.73	9.11	9.93
32768	7.62	12.15	15.40
65536	10.62	23.43	26.64
131072	15.85	32.53	49.34
262144	21.32	44.96	27.79
524288	39.55	65.61	49.03
1048576	74.91	109.06	91.58
2097152	145.99	199.42	176.82
4194304	286.26	393.98	346.91

OSU MPI Multiple Bandwidth / Message Rate

思源一号上的 OSU MPI Multiple Bandwidth

```
# OSU MPI Multiple Bandwidth / Message Rate Test v5.7.1
# [ pairs: 1 ] [ window size: 64 ]
```

(续下页)

(接上页)

#	Size	MB/s	Messages/s
1		6.82	6819901.32
2		13.70	6849644.94
4		27.43	6857747.81
8		54.70	6837453.43
16		109.97	6873169.62
32		218.58	6830520.90
64		402.61	6290822.77
128		773.02	6039234.26
256		1446.47	5650271.17
512		2646.67	5169286.04
1024		4411.59	4308188.99
2048		7656.33	3738444.41
4096		10508.77	2565618.70
8192		12463.12	1521377.49
16384		13336.64	814004.02
32768		13109.51	400070.54
65536		13959.39	213003.44
131072		14438.86	110159.77
262144		14689.16	56034.71
524288		14825.20	28276.83
1048576		14887.78	14198.09
2097152		14909.55	7109.43
4194304		14910.01	3554.82

π2.0 上的 OSU MPI Multiple Bandwidth

# OSU MPI Multiple Bandwidth / Message Rate Test v5.7.1			
# [pairs: 1] [window size: 64]			
#	Size	MB/s	Messages/s
1		1.45	1454746.80
2		2.91	1456550.86
4		7.50	1875912.49
8		14.89	1860936.30
16		29.09	1818298.04
32		65.96	2061307.14
64		130.36	2036819.98
128		270.56	2113729.90
256		586.45	2290813.80
512		1108.22	2164499.97
1024		1934.49	1889151.11
2048		3082.52	1505136.87
4096		4380.14	1069370.93
8192		6035.57	736763.84
16384		4511.71	275372.74
32768		6618.96	201994.78
65536		9373.92	143034.69
131072		11988.77	91467.04

(续下页)

(接上页)

262144	12119.05	46230.50
524288	12193.54	23257.32
1048576	12226.86	11660.44
2097152	12140.43	5789.01
4194304	12108.15	2886.81

ARM 上的 OSU MPI Multiple Bandwidth

# OSU MPI Multiple Bandwidth / Message Rate Test v5.6.3		
# [pairs: 1] [window size: 64]		
# Size	MB/s	Messages/s
1	4.24	4235302.84
2	8.82	4409629.80
4	17.55	4387775.11
8	34.67	4333726.75
16	67.82	4238584.63
32	129.61	4050327.86
64	262.59	4102908.64
128	499.14	3899519.14
256	811.93	3171585.76
512	1529.29	2986902.43
1024	2068.14	2019668.41
2048	2700.72	1318710.75
4096	3399.47	829948.38
8192	3878.01	473390.04
16384	11338.92	692072.80
32768	11810.79	360436.61
65536	12074.32	184239.48
131072	12190.81	93008.50
262144	12266.13	46791.59
524288	12305.57	23471.02
1048576	12324.26	11753.33
2097152	12335.56	5882.05
4194304	12340.24	2942.14

OSU MPI Broadcast Latency

# OSU MPI Broadcast Latency Test			
	思源一号	π2.0	ARM
# Size	Avg Latency(us)	Avg Latency(us)	Avg Latency(us)
1	0.45	3.00	2.35
2	0.44	2.89	2.43
4	0.45	2.87	2.38
8	0.44	2.96	2.37
16	0.45	4.02	2.42
32	0.47	3.91	2.42
64	0.87	3.71	2.59

(续下页)

(接上页)

128	0.66	3.74	2.66
256	1.01	3.79	3.11
512	1.11	4.10	3.36
1024	1.24	4.04	3.86
2048	3.35	8.39	4.20
4096	4.37	14.71	5.25
8192	3.48	27.43	7.15
16384	5.48	53.14	11.37
32768	9.43	107.36	19.00
65536	15.92	205.86	34.18
131072	28.89	415.82	65.70
262144	55.15	849.62	132.38
524288	107.83	385.97	122.73
1048576	169.50	780.35	239.25

参考资料

- OSU Benchmarks <http://mvapich.cse.ohio-state.edu/benchmarks/>
- DOWNLOAD, COMPILE AND RUN THE OSU BENCHMARK on AWS <https://www.hpcworkshops.com/07-efa/04-compile-run-osu.html>

HPCG

简介

HPCG(High Performance Conjugate Gradient) 基准测试程序作为超级计算机系统的性能评估指标之一，可有效评估超算系统在下述基本操作中的性能表现：

Sparse matrix-vector multiplication

Vector updates

Global dot products

Local symmetric Gauss-Seidel smoother

Sparse triangular solve (as part of the Gauss-Seidel smoother)

Driven by multigrid preconditioned conjugate gradient algorithm that exercises the key kernels on a nested set of coarse grids

导入 HPCG 环境

版本	平台	构建方式	模块名
3.1		spack	oneapi/2021.4.0 思源一号
3.1		spack	oneapi/2021.4.0 π 2.0

```
mkdir ~/HPCG && cd ~/HPCG
module load oneapi/2021.4.0
cp -r $MKLRROOT/benchmarks/hpcg ./
cd hpcg
```

HPCG 运行的重要参数

`problem_size` 和 `run_time_in_seconds` 在 `hpcg/bin/hpcg.dat` 中的默认值为 192, 60，这两个参数均可在运行脚本中指定。

在思源一号上运行

HPCG 运行脚本 (每个计算节点上共有两个 CPU Socket，每个 CPU Socket 启动 1 个进程，每个计算节点启动 2 个进程)

```
#!/bin/bash
#SBATCH --job-name=2nodes_hpcg
#SBATCH --partition=64c512g
#SBATCH -n 4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=32
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi/2021.4.0
export OMP_NUM_THREADS=32
export KMP_AFFINITY=granularity=fine,compact,1,0
export problem_size=192
export run_time_in_seconds=60

mpiexec.hydra -genvall bin/xhpcg_avx -n$problem_size -t$run_time_in_seconds
```

使用如下命令提交作业

```
sbatch run_hyper.slurm
```

运行结束后，将产生如下文件，`n192-4p-32t_V3.1_2022-02-09_16-43-22.txt`，其中 192 代表问题规模，4 代表使用的进程，32 代表 1 个进程包含的线程数。

```
Final Summary =
Final Summary ::HPCG result is VALID with a GFLOP/s rating of=109.132
Final Summary ::      HPCG 2.4 Rating (for historical value) is=109.691
Final Summary ::Reference version of ComputeDotProduct used=Performance_
→results are most likely suboptimal
Final Summary ::Results are valid but execution time (sec) is=65.1121
Final Summary ::      Official results execution time (sec) must be at_
→least=1800
```


在 **π2.0** 上运行

HPCG 运行脚本 (每个计算节点上共有两个 CPU Socket, 每个 CPU Socket 启动 1 个进程, 每个计算节点启动 2 个进程)

```
#!/bin/bash
#SBATCH --job-name=2nodes_hpcg
#SBATCH --partition=cpu
#SBATCH -n 4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=20
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi/2021.4.0
export OMP_NUM_THREADS=20
export KMP_AFFINITY=granularity=fine,compact,1,0
export problem_size=192
export run_time_in_seconds=60

mpirun.hydra -genvall bin/xhpcg_avx -n$problem_size -t$run_time_in_seconds
```

使用如下命令提交作业

```
sbatch run_hyper.slurm
```

运行结束后, 将产生如下文件, n192-4p-20t_V3.1_2022-02-26_16-34-36.txt, 其中 192 代表问题规模, 4 代表使用的进程, 32> 代表 1 个进程包含的线程数。

```
Final Summary =
Final Summary ::HPCG result is VALID with a GFLOP/s rating of=74.4941
Final Summary ::    HPCG 2.4 Rating (for historical value) is=74.829
Final Summary ::Reference version of ComputeDotProduct used=Performance_
→results are most likely suboptimal
Final Summary ::Results are valid but execution time (sec) is=62.6445
Final Summary ::    Official results execution time (sec) must be at_
→least=1800
```

运行结果

思源一号

problem_size:192 run_time_in_seconds:60		
核数	64	128
GFLOP/s	56.09485	112.07949

π 2.0

problem_size:192 run_time_in_seconds:60		
核数	40	80
GFOLP/s	37.9614	74.4941

minife

简介

MiniFE 是非结构隐式有限元代码的代理应用程序。它类似于 HPCCG 和 pHPCCG, 但提供了此类应用程序中各个步骤的更完整的垂直介绍。MiniFE 旨在成为“非结构化隐式有限元或有限体积应用程序的最佳近似值求解程序”

测试平台

- 思源一号 *miniFE*
- π 2.0 *miniFE*

思源一号 **miniFE**

运行脚本如下所示:

```
#!/bin/bash
#SBATCH --job-name=minife
#SBATCH --partition=64c512g
#SBATCH -N 4
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load minife/2.1.0-intel-2021.4.0

mpirun miniFE.x nx=600 verify_solution=1
```

运行结果如下所示:

```
CG solve:
  Iterations: 200
  Final Resid Norm: 0.00431172
  WAXPY Time: 2.34283
  WAXPY Flops: 3.90096e+11
  WAXPY Mflops: 166506
  DOT Time: 2.37993
  DOT Flops: 1.728e+11
  DOT Mflops: 72607.1
  MATVEC Time: 12.7909
```

(续下页)

(接上页)

```

MATVEC Flops: 2.34837e+12
MATVEC Mflops: 183598
Total:
  Total CG Time: 17.5337
  Total CG Flops: 2.91127e+12
  Total CG Mflops: 166039
  Time per iteration: 0.0876683
Total Program Time: 68.6005

```

结果显示部分最重要的值是 **CG** 求解的计算值 `CG Mflops`, 另外运行时间显示也是衡量并行速率的重要参考依据。

π2.0 miniFE

运行脚本如下所示:

```

#!/bin/bash
#SBATCH --job-name=minife
#SBATCH --partition=cpu
#SBATCH -N 4
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load minife/2.1.0-intel-2021.4.0

mpirun miniFE.x nx=600 verify_solution=1

```

运行结果如下所示:

```

CG solve:
  Iterations: 200
  Final Resid Norm: 0.00423874
  WAXPY Time: 3.80524
  WAXPY Flops: 3.90096e+11
  WAXPY Mflops: 102515
  DOT Time: 1.65335
  DOT Flops: 1.728e+11
  DOT Mflops: 104515
  MATVEC Time: 19.1712
  MATVEC Flops: 2.34837e+12
  MATVEC Mflops: 122495
  Total:
    Total CG Time: 24.6569
    Total CG Flops: 2.91127e+12
    Total CG Mflops: 118071
    Time per iteration: 0.123285
  Total Program Time: 75.0806

```

结果显示部分最重要的值是 **CG** 求解的计算值 `CG Mflops`, 另外运行时间显示也

是衡量并行速率的重要参考依据。

miniFE 的运行结果比较

思源一号上 **miniFE** 的运行结果

核数	CG Mflops	Total time
64	44365.2	118.973
128	87730.4	81.8141
256	166039	68.6005

π 2.0 上 **miniFE** 的运行结果

核数	CG Mflops	Total time
40	30003.6	175.854
80	59460.3	107.695
160	118071	75.0806

参考资料

- **miniFE** <https://github.com/Mantevo/miniFE/tree/v2.1.0>

IOR

超级计算行业的大多数人都熟悉使用 **FLOPS** 来衡量超级计算机的速度或能力。使用高性能 **Linpack(HPL)** 基准观察到的这一测量结果是 **Top500** 列表的基础。然而，随着数据密集型计算成为 **HPC** 社区的驱动力，**I/O** 性能变得越来越重要，**IOR** 基准测试已成为衡量集群和超级计算机的 **I/O** 能力的标准方式。

IOR 简介

IOR 测试通常涉及以下关键方面：

1. 并行文件 **I/O**： **IOR** 允许在多个计算节点上同时执行文件 **I/O** 操作，以模拟并行应用程序在 **HPC** 环境中的文件读取和写入操作。这有助于评估系统的并行 **I/O** 性能。
2. 吞吐量和延迟： **IOR** 测试会测量系统在不同并行工作负载下的数据吞吐量和延迟。这有助于了解系统在处理大规模数据时的性能表现。
3. 文件大小和访问模式： **IOR** 测试允许用户指定要测试的文件大小、读取和写入的访问模式（例如随机访问或顺序访问），以模拟不同类型的应用程序工作负载。
4. 并行文件系统支持： **IOR** 通常用于测试并行文件系统的性能，这是在 **HPC** 集群中常见的存储解决方案。测试可以帮助评估这些文件系统在高并发环境中的表现。
5. 性能度量： **IOR** 测试会生成各种性能度量，包括带宽、**IOPS**（每秒的 **I/O** 操作数）、延迟和效率等，以帮助用户评估存储系统的性能特征。

集群上部署的 ior

版本	平台	构建方式	模块名
3.3.0	思源一号	spack	ior/3.3.0-gcc-11.2.0-openmpi
3.3.0	Pi2.0	spack	ior/3.3.0-gcc-9.2.0-openmpi-4.0.5

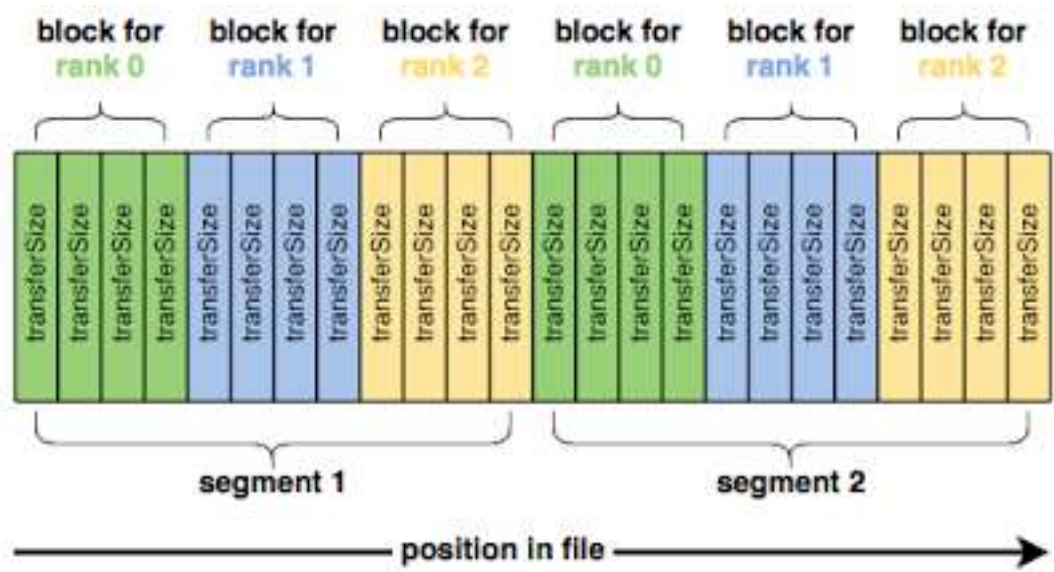
IOR 入门

IOR 常见参数

IOR 使用以下参数顺序写入数据：

- blockSize (-b)
- transferSize (-t)
- segmentCount (-s)
- numTasks (-n)

用图表来说明：



这四个参数是开始使用 IOR 所需的全部。然而，单纯地运行 IOR 通常会给出令人失望的结果。

例如，在 Pi 集群上运行单节点 IOR 测试，总共写入 10 GiB：

作业脚本：

```
#!/bin/bash
#SBATCH -J ior-test1
#SBATCH -N 1
#SBATCH --ntasks-per-node 40
#SBATCH -p cpu
#SBATCH -exclusive
```

(续下页)

(接上页)

```
module load gcc/9.2.0 openmpi/4.0.5-gcc-9.2.0 ior/3.3.0-gcc-9.2.0-openmpi-4.
↪0.5
mpirun -np 40 ior -t 1m -b 16m -s 16
```

测试结果:

```
Results:

access    bw(MiB/s)  IOPS        Latency(s)  block(KiB) xfer(KiB)  open(s)  ↪
↪ wr/rd(s)  close(s)    total(s)    iter
-----
↪ -----
write      714.78      714.81      0.891822    16384      1024.00    0.001451  ↪
↪ 14.33      0.280439    14.33       0
read       58303      58441      0.008393    16384      1024.00    0.016069  ↪
↪ 0.175220   0.045412    0.175635    0
remove     -          -          -          -          -          -          ↪
↪ -          -          2.76        0
Max Write: 714.78 MiB/sec (749.50 MB/sec)
Max Read:  58302.85 MiB/sec (61134.97 MB/sec)
```

根据测试结果来看，每秒只能从 **Lustre** 文件系统中写入几百兆字节的数据，而 **Lustre** 文件系统的速度本应更大。使用 (-FfilePerProcess=1) 选项从写入单个共享文件切换为每个进程一个文件，这会显著改变性能：

作业脚本:

```
#!/bin/bash
#SBATCH -J ior-test2
#SBATCH -N 1
#SBATCH --ntasks-per-node 40
#SBATCH -p cpu

module load gcc/9.2.0 openmpi/4.0.5-gcc-9.2.0 ior/3.3.0-gcc-9.2.0-openmpi-4.
↪0.5
mpirun -np 40 ior -t 1m -b 16m -s 16 -F
```

测试结果:

```
Results:

access    bw(MiB/s)  IOPS        Latency(s)  block(KiB) xfer(KiB)  open(s)  ↪
↪ wr/rd(s)  close(s)    total(s)    iter
-----
↪ -----
write      10523      10649      0.054914    16384      1024.00    0.057863  ↪
↪ 0.961590   0.563818    0.973092    0
```

(续下页)

(接上页)

```

read      40793      43119      0.009339      16384      1024.00      0.039125
→ 0.237481      0.100307      0.251024      0
remove    -          -          -          -          -          -
→ -          -          0.973260      0
Max Write: 10523.16 MiB/sec (11034.33 MB/sec)
Max Read:  40792.97 MiB/sec (42774.53 MB/sec)

```

可以看到写入的速度提升到 **11034.33 Mb/s**，这在很大程度上是因为让每个 **MPI** 进程处理自己的文件可以消除因文件锁定而产生的任何争用。

接下来使用多个节点进行 **IOR** 读写测试，在 **4** 个计算节点上同时执行文件 **I/O** 操作，以模拟并行应用程序在 **HPC** 环境中的文件读取和写入操作。

作业脚本：

```

#!/bin/bash
#SBATCH -J ior-test3
#SBATCH -N 4
#SBATCH --ntasks-per-node 40
#SBATCH -p cpu

module load gcc/9.2.0 openmpi/4.0.5-gcc-9.2.0 ior/3.3.0-gcc-9.2.0-openmpi-4.
→0.5
mpirun -np 160 ior -t 1m -b 16m -s 16 -F

```

测试结果：

```

Results:

access    bw(MiB/s)  IOPS        Latency(s)  block(KiB)  xfer(KiB)  open(s)
→ wr/rd(s)  close(s)   total(s)    iter
-----
→ -----
write      25919      25982      0.060796    16384      1024.00    0.052159
→ 1.58      1.27      1.58      0
read       162059     171187     0.010880    16384      1024.00    0.032160
→ 0.239270  0.132064  0.252747    0
remove    -          -          -          -          -          -
→ -          -          0.969064    0
Max Write: 25919.49 MiB/sec (27178.56 MB/sec)
Max Read:  162059.32 MiB/sec (169931.52 MB/sec)

```

可以看到在 **lustre** 系统上的多节点的测试下，最大写速度达到 **27178.56 MB/sec**，最大读速度达到 **169931.52 MB/sec**。

页面缓存对基准测试的影响

真正发生的情况是，**IOR** 读取的数据实际上并不是来自 **Lustre**，相反，文件的内容已经被缓存到计算节点的运行内存中，并且 **IOR** 能够直接从每个计算节点的 **DRAM** 中读取它们，因此 **IOR** 不是直接向 **Lustre** 写入和读取数据，实际上主要是每个计算节点上的内存。

有多种方法可以测量底层 **Lustre** 文件系统的读取性能。最粗略的方法是简单地写入比总页缓存容纳的数据更多的数据，以便在写入阶段完成时，文件的开头已经从缓存中逐出。例如，增加段数 (-s) 以写入更多数据。

更好的选择是让每个节点上的 **MPI** 进程只读取它们未写入的数据。例如，在每节点 4 个进程的测试中，将 **MPI** 进程到块的映射移动 4 个，使每个节点 **N** 读取节点 **N-1** 写入的数据。

由于页面缓存不在计算节点之间共享，因此以这种方式转移任务可确保每个 **MPI** 进程正在读取它未写入的数据。**IOR** 提供了 **-C** 选项 (**reorderTasks**) 来执行此操作，它强制每个 **MPI** 进程读取其相邻节点写入的数据。使用此选项运行 **IOR** 可提供更可靠的读取性能：

作业脚本：

```
#!/bin/bash
#SBATCH -J ior-test1
#SBATCH -N 4
#SBATCH --ntasks-per-node 40
#SBATCH -p cpu

module load gcc/9.2.0 openmpi/4.0.5-gcc-9.2.0 ior/3.3.0-gcc-9.2.0-openmpi-4.
→0.5
mpirun -np 160 ior -t 1m -b 16m -s 16 -F -C
```

测试结果：

```
Results:

access      bw(MiB/s)  IOPS        Latency(s)  block(KiB)  xfer(KiB)  open(s)  ↪
↪ wr/rd(s)  close(s)   total(s)    iter
-----
↪ -----
write       26350      26373       0.030734    16384       1024.00    0.103698 ↪
↪ 1.55      1.26       1.55        0
read        13817      13827       0.171605    16384       1024.00    0.092789 ↪
↪ 2.96      0.903531   2.96        0
remove      -          -           -           -           -          -          ↪
↪ -         -          0.997678    0
Max Write: 26350.48 MiB/sec (27630.49 MB/sec)
Max Read:  13816.78 MiB/sec (14487.94 MB/sec)
```

由于页面缓存，当写入已提交到内存而不是底层 **Lustre** 文件系统时，页面缓存会向 **IOR** 发出写入完成的信号。

为了解决页面缓存对写入性能的影响，我们可以在所有 **write()** 返回后立即发出

`fsync()` 调用，以强制刚刚写入的 **dirt page** 刷新到 **Lustre**。包括 `fsync()` 完成所需的时间可以衡量数据写入页面缓存以及页面缓存写回 **Lustre** 所需的时间。

IOR 提供了另一个方便的选项 `-e (fsync)` 来执行此操作。并且，再次使用此选项会改变性能测量：

作业脚本：

```
#!/bin/bash
#SBATCH -J ior-test1
#SBATCH -N 4
#SBATCH --ntasks-per-node 40
#SBATCH -p cpu

module load gcc/9.2.0 openmpi/4.0.5-gcc-9.2.0 ior/3.3.0-gcc-9.2.0-openmpi-4.
→0.5
mpirun -np 160 ior -t 1m -b 16m -s 16 -F -C -e
```

测试结果：

```
Results:

access      bw (MiB/s)  IOPS          Latency (s)  block (KiB)  xfer (KiB)  open (s)  ↵
↵ wr/rd(s)   close(s)     total(s)     iter
-----
↵ -----
write       22509       22525         0.074681     16384        1024.00     0.065918  ↵
↵ 1.82       1.43         1.82         0
read       14349       14351         0.177082     16384        1024.00     0.351030  ↵
↵ 2.85       0.824950     2.85         0
remove     -           -             -            -            -           -          ↵
↵ -          -             1.00         0
Max Write: 22509.46 MiB/sec (23602.88 MB/sec)
Max Read:  14349.29 MiB/sec (15046.32 MB/sec)
```

这样就对文件系统有了可信的带宽测量。

大规模 IO 测试

如果要对集群文件系统进行大规模的 **IO** 吞吐测试，可参考下面的作业脚本，该作业设置每个节点 4 个 **MPI** 进程，一共使用 32 个节点，`blocksize` 设置为 16m，`transfersize` 设置为 1m，`segmentCount` 设置为 4096，循环测试 10000 次。

在思源一号上运行：

```
#!/bin/bash
#SBATCH --job-name="iortest"
#SBATCH -N 32
#SBATCH --ntasks-per-node=4
#SBATCH -p 64c512g
```

(续下页)

(接上页)

```
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior
export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=^openib

for i in {1..10000}
do
    mpirun ior -t 1m -b 16m -s 4096 -F -C -e
done
```

在 **pi2.0** 集群上运行:

```
#!/bin/bash
#SBATCH --job-name="iortest"
#SBATCH -N 32
#SBATCH --ntasks-per-node=4
#SBATCH -p cpu
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior

for i in {1..10000}
do
    mpirun ior -t 1m -b 16m -s 4096 -F -C -e
done
```

参考链接:

1. <https://wiki.lustre.org/IOR>
2. <https://glennklockwood.blogspot.com/2016/07/basics-of-io-benchmarking.html>

mdtest

简介

mdtest 是一个用于测试文件系统性能的基准测试工具，特别用于测量文件系统的元数据性能，如创建、查找、读取和删除文件和目录。这个工具通常用于评估并行文件系统、分布式文件系统和对象存储系统的性能，以及它们在高性能计算（HPC）和大规模数据存储环境中的表现。

主要功能

mdtest 的主要功能包括：

- 1. 创建和删除文件/目录：mdtest 可以模拟并记录文件和目录的创建和删除操作，以便评估文件系统在处理这些操作时的性能。
- 2. 统计文件/目录：工具可以执行文件和目录的统计操作，以评估文件系统执行元数据查询的性能。
- 3. 读取文件：mdtest 可以模拟从文件中读取数据的操作，以便评估文件系统的读取性能。
- 4. 随机操作：mdtest 可以配置为执行随机操作，以评估文件系统的随机访问性能。
- 5. 可定制性：工具提供了多个命令行选项，可以用于配置测试的参数，如目录结构的深度、分支因子、迭代次数、并发性等。
- 6. 并行性：mdtest 可以在多个并行任务之间执行测试，以模拟多用户或多任务环境下的文件系统性能。
- 7. 输出和报告：mdtest 生成性能数据的输出，允许用户分析文件系统性能，并生成报告以帮助优化和调整文件系统配置。

mdtest 是一个有用的工具，可帮助管理员和研究人员评估和比较不同文件系统的性能，特别是在高性能计算环境中，其中文件系统的性能至关重要。

集群上部署的 mdtest

目前 mdtest 工具已经被收纳到 ior 测试工具中，调用 ior 工具即可进行 mdtest 测试。

版本	平台	构建方式	模块名
3.3.0	思源一号	spack	ior/3.3.0-gcc-11.2.0-openmpi
3.3.0	Pi2.0	spack	ior/3.3.0-gcc-9.2.0-openmpi-4.0.5

mdtest 用法

选项含义

mdtest 选项含义

```
-a：用于S3目标的用户ID
-A：S3目标的主机名或IP地址
-b：分层目录结构的分支因子
-B：在阶段之间没有障碍（创建/统计/删除）
-c：集体创建：任务0执行所有创建和删除操作
-C：仅创建文件/目录
-d：测试运行的目录
-D：仅对目录执行测试（不包括文件）
-e：从每个文件中读取的字节数
-E：仅读取文件
```

(续下页)

(接上页)

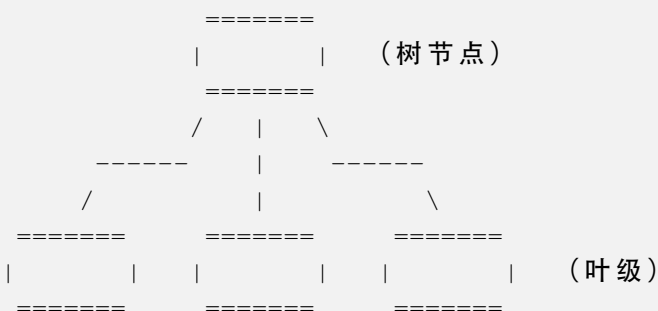
```

-f: 测试将在其上运行的任务的第一个编号
-F: 仅对文件执行测试（不包括目录）
-g: 添加到唯一性桶名称的整数标识符
-h: 打印帮助信息
-i: 测试运行的迭代次数
-I: 每个树节点的项目数
-l: 测试将在其上运行的任务的最后一个编号
-L: 仅在叶子级别创建文件/目录
-M: 每个任务的唯一工作目录，并跨LUSTRE MDTs执行分布
-n: 每个任务将创建/统计/删除#个文件/目录
-N: 文件/目录统计之间相邻任务的步幅#（本地=0）
-p: 迭代前延迟（以秒为单位）
-r: 仅删除文件/目录
-R: 随机统计文件/目录（可提供可选种子）
-s: 每个测试的任务数量之间的步幅（需要4个节点以避免缓存效应）
-S: 共享文件访问（仅限文件，不包括目录）
-t: 唯一工作目录开销时间
-T: 仅统计文件/目录
-u: 每个任务的唯一工作目录
-v: 详细度（每个选项的每个实例递增1）
-V: 详细度值
-w: 要写入每个文件的字节数
-Y: 在写入完成后同步文件
-z: 分层目录结构的深度

```

分层目录结构（树）的示例

分层目录结构（树）



在这个示例中，树的深度为 1 ($z=1$)，分支因子为 3 ($b=3$)。树的顶部节点是根节点。离根节点最远的节点级别是叶子级别。所有由 **mdtest** 创建的树都是平衡的。

简单示例

下面已思源一号集群为例，执行如下简单的运行示例，进一步了解 **mdtest** 的运行方式，

1. 这个命令将创建一个类似上述示例的树，然后每个任务将在每个树节点创建 10 个文件/目录。将创建三个这样的树（每个迭代一个）。

```
#!/bin/bash
#SBATCH --job-name="mdtest"
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -p 64c512g
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior
export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=openib

mdtest -z 1 -b 3 -I 10 -C -i 3
```

测试结果

```
-- started at 11/07/2023 14:02:14 --

mdtest-3.3.0 was launched with 1 total task(s) on 1 node(s)
Command line used: mdtest '-z' '1' '-b' '3' '-I' '10' '-C' '-i' '3'
Path: /dssg/home/acct-hpc/hpcxdy/test/mdtest
FS: 7487.9 TiB    Used FS: 77.4%    Inodes: 3814.7 Mi    Used Inodes: 67.0%

Nodemap: 1
1 tasks, 40 files/directories

SUMMARY rate: (of 3 iterations)
  Operation                Max           Min           Mean           _
  ↳Std Dev
  -----
  ↳-----
    Directory creation      :    7737.830    1898.994    5539.577_
  ↳    2592.713
    Directory stat         :         0.000         0.000         0.000_
  ↳         0.000
    Directory removal      :         0.000         0.000         0.000_
  ↳         0.000
    File creation          :    4826.385    4314.873    4610.705_
  ↳    216.379
    File stat              :         0.000         0.000         0.000_
  ↳         0.000
    File read              :         0.000         0.000         0.000_
  ↳         0.000
    File removal          :         0.000         0.000         0.000_
  ↳         0.000
    Tree creation          :    7340.325    234.797    4841.319_
  ↳    3261.222
    Tree removal           :         0.000         0.000         0.000_
  ↳         0.000

-- finished at 11/07/2023 14:02:14 --
```

2. 在当前工作目录中创建一个深度为 5、分支因子为 2 的目录树。每个任务在每个树节点上操作 10 个文件/目录。

```
#!/bin/bash
#SBATCH --job-name="mdtest"
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -p 64c512g
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior
export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=openib

mdtest -I 10 -z 5 -b 2
```

测试结果

```
-- started at 11/07/2023 14:04:55 --

mdtest-3.3.0 was launched with 1 total task(s) on 1 node(s)
Command line used: mdtest '-I' '10' '-z' '5' '-b' '2'
Path: /dssg/home/acct-hpc/hpcxdy/test/mdtest
FS: 7487.9 TiB    Used FS: 77.4%    Inodes: 3814.7 Mi    Used Inodes: 67.0%

Nodemap: 1
1 tasks, 630 files/directories

SUMMARY rate: (of 1 iterations)
  Operation                               Max           Min           Mean           ┐
↪ Std Dev
  -----
↪ -----
  Directory creation      :      8063.150      8063.150      8063.150┐
↪      0.000
  Directory stat          :     285877.774     285877.774     285877.774┐
↪      0.000
  Directory removal       :      21821.806      21821.806      21821.806┐
↪      0.000
  File creation           :      8053.735      8053.735      8053.735┐
↪      0.000
  File stat               :     285288.229     285288.229     285288.229┐
↪      0.000
  File read               :     132316.999     132316.999     132316.999┐
↪      0.000
  File removal            :      8476.868      8476.868      8476.868┐
↪      0.000
  Tree creation           :      22422.530      22422.530      22422.530┐
↪      0.000
```

(续下页)

(接上页)

```

Tree removal          :          21058.275      21058.275      21058.275
→          0.000
-- finished at 11/07/2023 14:04:55 --

```

3. 每个任务在当前路径中的根节点上创建 100 个文件/目录（根节点没有分支）。这将执行三次，迭代中计算聚合值。

```

#!/bin/bash
#SBATCH --job-name="mdtest"
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -p 64c512g
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior
export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=^openib

mdtest -n 100 -i 3

```

测试结果

```

-- started at 11/07/2023 14:09:22 --

mdtest-3.3.0 was launched with 1 total task(s) on 1 node(s)
Command line used: mdtest '-n' '100' '-i' '3'
Path: /dssg/home/acct-hpc/hpcxdy/test/mdtest
FS: 7487.9 TiB   Used FS: 77.4%   Inodes: 3814.7 Mi   Used Inodes: 67.0%

Nodemap: 1
1 tasks, 100 files/directories

SUMMARY rate: (of 3 iterations)
  Operation          Max          Min          Mean
→Std Dev
  -----
→-----
  Directory creation  :    29598.602    24151.649    26791.562
→    2226.847
  Directory stat      :    528368.082    520410.500    524339.827
→    3249.423
  Directory removal   :    23142.737    22403.642    22895.651
→    347.904
  File creation        :    27976.199    19203.375    24963.815
→    4074.675
  File stat           :    531245.186    524915.095    528648.255
→    2706.261
  File read           :    178095.024    175329.708    176263.574

```

(续下页)

(接上页)

```

→      1295.116
File removal          :      23606.118      15639.633      20943.016
→      3750.070
Tree creation         :      27059.938      4670.584      12952.803
→      10025.568
Tree removal          :      14532.983      13990.906      14203.328
→      236.318
-- finished at 11/07/2023 14:09:22 --

```

4. 每个任务在当前目录中创建一个目录树。每棵树的深度为3，分支因子为5。在每棵树的每个节点上都操作了五个文件/目录。

```

#!/bin/bash
#SBATCH --job-name="mdtest"
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -p 64c512g
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior
export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=^openib

mdtest -I 5 -z 3 -b 5 -u

```

测试结果

```

-- started at 11/07/2023 14:09:59 --

mdtest-3.3.0 was launched with 1 total task(s) on 1 node(s)
Command line used: mdtest '-I' '5' '-z' '3' '-b' '5' '-u'
Path: /dssg/home/acct-hpc/hpcxdy/test/mdtest
FS: 7487.9 TiB   Used FS: 77.4%   Inodes: 3814.7 Mi   Used Inodes: 67.0%

Nodemap: 1
1 tasks, 780 files/directories

SUMMARY rate: (of 1 iterations)
  Operation          Max          Min          Mean
→Std Dev
-----
→-----
  Directory creation  :      13937.588      13937.588      13937.588
→      0.000
  Directory stat      :      322948.871      322948.871      322948.871
→      0.000
  Directory removal   :      23411.636      23411.636      23411.636
→      0.000

```

(续下页)

(接上页)

```

File creation          :      16614.880      16614.880      16614.880
→      0.000
File stat              :      327131.260      327131.260      327131.260
→      0.000
File read              :      141012.437      141012.437      141012.437
→      0.000
File removal           :      17020.530      17020.530      17020.530
→      0.000
Tree creation          :      20008.421      20008.421      20008.421
→      0.000
Tree removal           :      23610.532      23610.532      23610.532
→      0.000
-- finished at 11/07/2023 14:09:59 --

```

大规模 mdtest 测试

如果要对集群进行大规模小文件读写测试，可参考下面脚本，一共使用 **64** 个节点测试，每个节点启动 **4** 个 **MPI** 进程做文件操作，总共会产生约 **1600** 万文件，测试迭代 **100** 次。

在思源一号上运行

```

#!/bin/bash
#SBATCH --job-name="mdtest"
#SBATCH -N 64
#SBATCH --ntasks-per-node=4
#SBATCH -p 64c512g
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc openmpi ior
export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=openib

mpirun mdtest -F -L -z 4 -b 2 -I 4096 -i 100 -u

```

在 Pi2.0 上运行

```

#!/bin/bash
#SBATCH --job-name="mdtest"
#SBATCH -N 64
#SBATCH --ntasks-per-node=4
#SBATCH -p cpu
#SBATCH --exclusive
#SBATCH --output=%j.out

```

(续下页)

(接上页)

```
#SBATCH --error=%j.err

module load gcc openmpi ior

mpirun mdtest -F -L -z 4 -b 2 -I 4096 -i 100 -u
```

io500

简介

IO500 (I/O500) 是一种广泛用于评估和基准测试高性能计算和超级计算机系统的工具和方法。它是 **HPC 领域 (High-Performance Computing, 高性能计算)** 中的一种性能基准测试套件, 旨在测量存储和 **I/O (输入/输出)** 性能, 特别是在大规模并行计算环境中。**IO500** 测试涵盖了多个存储和 **I/O** 方面的性能指标, 包括:

1. **MDtest**: 用于测试元数据 (**Metadata**) 操作性能, 例如文件和目录的创建和删除。
2. **IOR**: 用于测试大文件读取和写入性能。
3. **Find**: 用于测试查找文件性能, 例如通过文件名或通配符搜索文件。
4. **HDF5 IOR**: 用于测试 **HDF5** 格式数据的读取和写入性能。
5. **IOR-Easy**: 用于测试简化操作的大文件读取和写入性能。

IO500 的目标是提供一个全面的性能测试套件, 以帮助评估存储系统在大规模高性能计算环境中的性能。它不仅仅是一个性能测量工具, 还提供了一种标准化方法, 使不同系统的性能可以进行比较。这对于研究人员、工程师和系统管理员来说非常有价值, 因为它有助于确定哪种存储架构和配置对于特定的计算工作负载是最有效的。

编译安装

以思源一号集群为例下载 **io500** 源码执行 **prepare.sh** 将编译安装 **io500** 所需要的软件依赖

```
srunk -n 1 -p 64c512g --pty /bin/bash
module load oneapi
git clone https://github.com/IO500/io500.git -b io500-sc23
cd io500
./prepare.sh
```

修改 io500.sh

修改现有的 **io500.sh** 脚本以包含调度程序所需的信息。由于 **io500** 测试需要大量存储空间, 下面仅以 1 节点, 2 核心作为 **io500** 使用示例参考, 提供足够的执行时间 (通常 2 小时足以完整运行) 并根据需要调整保留的资源。注意: 如果需要对集群进行整体评测需要增加节点数以及核心数, **io500** 榜单常见节点数为 10 节点及以上。

```
#!/bin/bash
#SBATCH --nodes=1 --ntasks-per-node=2 --exclusive -p 64c512g
module load oneapi

io500_ini="$1"           # You can set the ini file here
io500_mpirun="mpirun"
io500_mpiargs="-np 2"

.....
```

生成 ini 文件

```
./io500 --list > myconfig.ini
```

运行

直接或通过批处理作业提交下面命令

```
sbatch io500.sh myconfig.ini
```

结果

结果目录包含带时间戳的输出目录，重复运行相同的 io500 不会消除以前运行的结果。每个目录中有两个重要的摘要文件：

- *result_summary.txt* - 一个简单、人类可读的文件，包含单独的性能测量值和 IO500 分数
- *result.txt* - 机器可读的结果摘要

检查结果输出，并验证 `$resultdir/result_summary.txt` 中的所有结果均未标记为 [INVALID]，例如，由于运行时间低于 300 秒或其他错误。末尾还必须有 SCORE 行，并且不能将其标记为 [INVALID] ##### 思源一号测试结果 -----

```
IO500 version io500-sc23_v1 (standard)
[RESULT]      ior-easy-write      2.858210 GiB/s : time 312.975 seconds
[RESULT]      mdtest-easy-write    6.200877 kIOPS : time 313.991 seconds
[      ]      timestamp          0.000000 kIOPS : time 0.000 seconds
[RESULT]      ior-hard-write      0.706375 GiB/s : time 762.340 seconds
[RESULT]      mdtest-hard-write    2.465601 kIOPS : time 301.321 seconds
[RESULT]      find                20.338138 kIOPS : time 132.172 seconds
[RESULT]      ior-easy-read        1.300786 GiB/s : time 687.671 seconds
[RESULT]      mdtest-easy-stat      2.593817 kIOPS : time 749.219 seconds
[RESULT]      ior-hard-read        0.936611 GiB/s : time 574.939 seconds
[RESULT]      mdtest-hard-stat      1.608079 kIOPS : time 461.415 seconds
[RESULT]      mdtest-easy-delete    1.755264 kIOPS : time 1106.908 seconds
[RESULT]      mdtest-hard-read      1.465098 kIOPS : time 506.317 seconds
[RESULT]      mdtest-hard-delete    0.884044 kIOPS : time 838.717 seconds
[SCORE ] Bandwidth 1.252344 GiB/s : IOPS 2.714576 kiops : TOTAL 1.843796
```

Pi2.0 测试结果

```
IO500 version io500-sc23_v1 (standard)
[RESULT]      ior-easy-write      1.236917 GiB/s : time 322.573 seconds
[RESULT]      mdtest-easy-write    4.289295 kIOPS : time 306.952 seconds
[      ]      timestamp          0.000000 kIOPS : time 0.001 seconds
[RESULT]      ior-hard-write      0.595349 GiB/s : time 303.371 seconds
[RESULT]      mdtest-hard-write    1.402383 kIOPS : time 429.585 seconds
[RESULT]      find                30.659287 kIOPS : time 62.570 seconds
[RESULT]      ior-easy-read        0.675851 GiB/s : time 590.341 seconds
[RESULT]      mdtest-easy-stat     1.423461 kIOPS : time 922.849 seconds
[RESULT]      ior-hard-read        0.087374 GiB/s : time 2067.057 seconds
[RESULT]      mdtest-hard-stat     2.580531 kIOPS : time 233.987 seconds
[RESULT]      mdtest-easy-delete    7.656550 kIOPS : time 172.455 seconds
[RESULT]      mdtest-hard-read     0.785820 kIOPS : time 765.853 seconds
[RESULT]      mdtest-hard-delete    4.326086 kIOPS : time 139.970 seconds
[SCORE ] Bandwidth 0.456654 GiB/s : IOPS 3.394612 kiops : TOTAL 1.245055
```

参考链接

- 1. <https://io500.org/>

4.10.4 AI 计算

TensorFlow

简介

TensorFlow 是一个端到端开源机器学习平台。它拥有一个包含各种工具、库和社区资源的全面灵活生态系统，可以让研究人员推动机器学习领域的先进技术的发展，并让开发者轻松地构建和部署由机器学习提供支持的应用。

测试数据位置

```
思源一号：
/dssg/share/sample/tensorflow/tf_test.py

pi2.0:
/lustre/share/samples/tensorflow/tf_test.py
```

集群上的 TensorFlow

版本	平台	构建方式	导入方式
2.8.2		容器	module load tensorflow/2.8.2 思源一号
2.4.1		容器	module load tensorflow/2.4.1 pi2.0

TensorFlow 使用教程

- 思源一号 *TensorFlow*
- $\pi 2.0$ *TensorFlow*

一. 思源一号 TensorFlow

1.1 预编译版本的使用方式

思源上拷贝数据到本地

```
cd
mkdir tensorflow
cd tensorflow
cp /dssg/share/sample/tensorflow/tf_test.py ./
```

思源上 **TensorFlow** 提交作业脚本

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=32
#SBATCH --gres=gpu:2

module load tensorflow/2.8.2

python tf_test.py
```

1.2 思源一号上自定义构建 TensorFlow 2.x 环境

使用 **miniconda** 安装 **TensorFlow**

创建名为 `tf-env` 的虚拟环境，激活虚拟环境，然后安装 **TensorFlow**

```
cd
mkdir tensorflow
cd tensorflow
cp /dssg/share/sample/tensorflow/tf_test.py ./
module load miniconda3
conda create -n tf-env python=3.8.5
source activate tf-env
conda install tensorflow=2.8.2=cpu_py38h75b8afa_0
#上述命令会自动安装如下依赖
.cudatoolkit=11.3.1=h2bc3f7f_2
```

(续下页)

(接上页)

```
.cudnn=8.2.1=cuda11.3_0
.numpy=1.23.4=py38h14f4228_0

conda install matplotlib=3.5.0=py38h06a4308_0
conda install -c conda-forge sklearn-quantile=0.0.18=py38h3ec907f_0
```

思源一号上作业提交脚本如下所示

在 A100 上使用 TensorFlow。作业使用单节点，分配 2 块 GPU，GPU:CPU 配比 1:16

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=32
#SBATCH --gres=gpu:2

module load miniconda3
source activate tf-env

python tf_test.py
```

1.3 思源一号上自定义构建 TensorFlow 1.x 环境

使用 pip 安装 NVIDIA A100 GPU 上优化的 TensorFlow 1.x

```
cd
mkdir tensorflow
cd tensorflow
cp /dssg/share/sample/tensorflow/tf_test.py ./
module load miniconda3
conda create -n tf-env python=3.8.5
source activate tf-env
pip install --user nvidia-pyindex
pip install --user nvidia-tensorflow[horovod]
```

二. π2.0 TensorFlow

2.1 π2.0 上预编译版本的使用方式

首先拷贝数据到本地

```
cd
mkdir tensorflow
```

(续下页)

(接上页)

```
cd tensorflow
cp /lustre/share/samples/tensorflow/tf_test.py ./
```

使用如下脚本提交 **TensorFlow** 作业

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=12
#SBATCH --gres=gpu:2

module load tensorflow/2.4.1
python tf_test.py
```

2.2 π2.0 上自定义安装 TensorFlow

π2.0 上使用 miniconda 安装 TensorFlow

创建名为 tf-env 的虚拟环境，激活虚拟环境，然后安装 TensorFlow

```
cd
mkdir tensorflow
cd tensorflow
cp /lustre/share/samples/tensorflow/tf_test.py ./
module load miniconda3
conda create -n tf-env python=3.8.5
source activate tf-env
conda install tensorflow=2.4.1=cpu_py38h8a7d6ce_0
#上述命令会自动安装如下依赖
.cudatoolkit=10.1.243h6bb024c_0
.cudnn=7.6.5=cuda10.1_0
.numpy=1.23.4=py38h14f4228_0

conda install matplotlib=3.4.2=py38h06a4308_0
conda install -c conda-forge sklearn-quantile=0.0.18=py38h3ec907f_0
```

作业提交脚本如下

在 DGX2 上使用 TensorFlow。作业使用单节点，分配 2 块 GPU，GPU:CPU 配比 1:6

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p dgx2
#SBATCH -o %j.out
```

(续下页)

(接上页)

```
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=12
#SBATCH --gres=gpu:2

module load miniconda3
source activate tf-env
python tf_test.py
```

TensorFlow 的运行结果

思源一号 TensorFlow

思源上预编译版本的运行结果

Accuracy: mean=98.653 std=0.083, n=5

思源上自定义编译版本的运行结果

Accuracy: mean=98.645 std=0.134, n=5

π2.0 TensorFlow

预编译版本的运行结果

Accuracy: mean=98.698 std=0.089, n=5

自定义编译版本的运行结果

Accuracy: mean=98.638 std=0.159, n=5

参考资料

- TensorFlow 官网
- NVIDIA GPU CLOUD

PyTorch

简介

PyTorch 是一个 Python 优先的深度学习框架，也是使用 GPU 和 CPU 优化的深度学习张量库，能够在强大的 GPU 加速基础上实现张量和动态神经网络。同时，PyTorch 主要为开发者提供两种高层面的功能：

1. 使用强大的 GPU 加速的 Tensor 计算（类似 numpy）；

2. 构建 autograd 系统的深度神经网络。

通常，人们使用 PyTorch 的原因通常有二：

1. 作为 numpy 的替代，以便使用强大的 GPU；
2. 将其作为一个能提供最大的灵活性和速度的深度学习研究平台

π 超算上的 PyTorch

π 超算上可用 miniconda 自行安装 PyTorch，也可用已预置的 NVIDIA 提供的 NGC 镜像 pytorch-1.6.0（性能更好）。

版本	平台	构建方式	名称
1.6.0	gpu	容器	/lustre/share/singularity/modules/pytorch/1.6.0.sif
1.6.0	gpu	容器	/dssg/share/imgs/pytorch/1.6.0.sif 思源平台

使用 miniconda 安装 PyTorch

创建名为 pytorch-env 的虚拟环境，激活虚拟环境，然后安装 pytorch

```
$ module load miniconda3
$ conda create -n pytorch-env
$ source activate pytorch-env
$ conda install pytorch torchvision -c pytorch
```

提交 PyTorch 作业

示例：在 DGX-2 及 A100 上均可使用 pytorch。作业使用单节点，分配 2 块 GPU，GPU:CPU 配比 1:6。脚本名称可设为 slurm.test。

在 DGX-2 队列上提交 pytorch 作业：

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --gres=gpu:2

module load miniconda3
source activate pytorch-env

python -c 'import torch; print(torch.__version__); print(torch.zeros(10,10).
↪cuda().shape)'
```

在 A100 队列上提交 pytorch 作业:

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --gres=gpu:2

module load miniconda3
source activate pytorch-env

python -c 'import torch; print(torch.__version__); print(torch.zeros(10,10).
↪ cuda().shape)'
```

使用以下指令提交作业

```
$ sbatch slurm.test
```

使用 π 集群提供的 PyTorch

π 超算中已经预置了 NVIDIA GPU CLOUD 提供的优化镜像, 通过调用该镜像即可运行 PyTorch 作业, 无需单独安装, 目前版本为 pytorch/1.6.0。

查看 π 超算上已编译的软件模块:

```
module av pytorch
```

调用该模块:

```
module load pytorch/1.6.0
```

以下 slurm 脚本, 在 dgx2 队列上使用 2 块 gpu, 并配比 12 cpu 核心。脚本名称可设为 slurm.test

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --gres=gpu:2

module load pytorch/1.6.0
```

(续下页)

(接上页)

```
python -c 'import torch; print(torch.__version__); print(torch.zeros(10,10).
→cuda().shape)'
```

使用如下指令提交:

```
$ sbatch slurm.test
```

查看思源一号上已编译的软件模块:

```
module use /dssg/share/imgs/
module av pytorch
```

调用该模块:

```
module load pytorch/1.6.0
```

以下 `slurm` 脚本, 在 A100 队列上使用 2 块 `gpu`, 并配比 12 `cpu` 核心。脚本名称可设为 `slurm.test`

```
#!/bin/bash
#SBATCH -J test
#SBATCH -p a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --gres=gpu:2

module use /dssg/share/imgs/
module load pytorch/1.6.0

python -c 'import torch; print(torch.__version__); print(torch.zeros(10,10).
→cuda().shape)'
```

使用如下指令提交:

```
$ sbatch slurm.test
```

Pytorch 单卡性能测试

算例下载:

```
$ git clone https://github.com/SJTU-HPC/HPCTesting.git
$ cd HPCTesting/pytorch-gpu-benchmark
```

DGX-2 运行脚本:

```
#!/bin/bash
#SBATCH -p dgx2
#SBATCH -n 1
#SBATCH --ntasks-per-node 1
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:1
#SBATCH -N 1

singularity run --nv /lustre/share/singularity/modules/pytorch/1.6.0.sif_
python benchmark_models.py --folder v100 -w 10 -n 5 -b 32 -g 1 && &>/dev/
null
```

A100 运行脚本:

```
#!/bin/bash

#SBATCH -p a100
#SBATCH -n 1
#SBATCH --ntasks-per-node 1
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:1
#SBATCH -N 1

singularity run --nv /dssg/share/imgs/pytorch/1.6.0.sif python benchmark_
models.py --folder a100 -w 10 -n 5 -b 32 -g 1 && &>/dev/null
```

DGX-2 测试结果将被放在 v100 文件夹内, A100 测试结果将被放在 a100 文件夹内, 均为 CSV 格式。

单卡测试结果:

resnet18,resnet34,resnet50,resnet101,resnet152,resnext50_32x4d,resnext101_32x8d,wide_resnet50_2,wid模型的平均 batch 耗时如下:

pytorch			
partition	mode	precision	ms/batch
v100	train	double	471
v100	train	float	180
v100	train	half	91
v100	inference	double	148
v100	inference	float	63
v100	inference	half	26
a100	train	double	350
a100	train	float	78
a100	train	half	60
a100	inference	double	107
a100	inference	float	25
a100	inference	half	16

Pytorch 双卡性能测试

算例下载:

```
$ git clone https://github.com/SJTU-HPC/HPCTesting.git
$ cd HPCTesting/pytorch-gpu-benchmark
```

DGX-2 运行脚本:

```
#!/bin/bash
#SBATCH -p dgx2
#SBATCH -n 2
#SBATCH --ntasks-per-node 2
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:2
#SBATCH -N 1

singularity run --nv /lustre/share/singularity/modules/pytorch/1.6.0.sif_
python benchmark_models.py --folder v100 -w 10 -n 5 -b 16 -g 1 && &>/dev/
null
```

A100 运行脚本:

```
#!/bin/bash

#SBATCH -p a100
#SBATCH -n 2
#SBATCH --ntasks-per-node 2
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:2
#SBATCH -N 1

singularity run --nv /dssg/share/imgs/pytorch/1.6.0.sif python benchmark_
models.py --folder a100 -w 10 -n 5 -b 16 -g 1 && &>/dev/null
```

DGX-2 测试结果将被放在 v100 文件夹内, A100 测试结果将被放在 a100 文件夹内, 均为 CSV 格式。

双卡测试结果:

resnet18,resnet34,resnet50,resnet101,resnet152,resnext50_32x4d,resnext101_32x8d,wide_resnet50_2,wid
模型的平均 batch 耗时如下:

pytorch			
partition	mode	precision	ms/batch
v100	train	double	289
v100	train	float	134
v100	train	half	79
v100	inference	double	102
v100	inference	float	56
v100	inference	half	38
a100	train	double	213
a100	train	float	68
a100	train	half	57
a100	inference	double	69
a100	inference	float	30
a100	inference	half	25

建议

A100 显卡的内存为 40GB，而 V100 显卡内存为 32GB。从测试结果也可以看出，A100 无论是训练还是推理，在单精度、双精度、半精度的计算速度均大幅领先于 V100，推荐大家使用思源一号的 A100 队列提交 pytorch 作业。

参考资料

- [PyTorch 官网](#)
- [NVIDIA GPU CLOUD](#)

Keras

简介

Keras 是一个用 Python 编写的高级神经网络 API，它能够以 TensorFlow, CNTK, 或者 Theano 作为后端运行。Keras 的开发重点是支持快速的实验。能够以最小的时延把你的想法转换为实验结果，是做好研究的关键。

如果你在以下情况下需要深度学习库，请使用 Keras：- 允许简单而快速的原型设计（由于用户友好，高度模块化，可扩展性）。- 同时支持卷积神经网络和循环神经网络，以及两者的组合。- 在 CPU 和 GPU 上无缝运行。

集群上的 Keras 安装方法

```
module load miniconda3
conda create -n mypy
source activate mypy
conda install cudatoolkit=11.0
pip install keras tensorflow-gpu==2.8.0
```

π 集群上的 Slurm 脚本 `slurm.test`

在 dgx2 队列上, 使用 1 张卡 (`gres=gpu:1`), 配合 6 核芯 (`n = 6`)

```
#!/bin/bash

#SBATCH -J keras
#SBATCH --partition=dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --ntasks-per-node=1
#SBATCH -N 1
#SBATCH --cpus-per-task 6
#SBATCH --gres=gpu:1
#SBATCH --mem=MaxMemPerNode

ulimit -l unlimited
ulimit -s unlimited

module load miniconda3
source activate mpyy

python ...
```

在 A100 队列上, 使用 1 张卡 (`gres=gpu:1`), 配合 6 核芯 (`n = 6`)

```
#!/bin/bash

#SBATCH -J keras
#SBATCH --partition=a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --ntasks-per-node=1
#SBATCH -N 1
#SBATCH --cpus-per-task 6
#SBATCH --gres=gpu:1
#SBATCH --mem=MaxMemPerNode

ulimit -l unlimited
ulimit -s unlimited

module load miniconda3
source activate mpyy

python ...
```

π 集群上提交作业

```
$ sbatch slurm.test
```

算例测试

超算中心提供了用来测试 **keras** 的算例。

使用命令

```
$ cd ~
$ git clone https://github.com/SJTU-HPC/HPCTesting.git
$ cd HPCTesting/keras/case1
$ conda env create -f environment.yml
$ curl -O https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-
→8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip
$ unzip -q kagglecatsanddogs_3367a.zip
```

在 π 超算上，使用如下脚本来提交该算例作业：

```
#!/bin/bash
#SBATCH -p dgx2
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --ntasks-per-node 1
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:1

cd ~/HPCTesting/keras/case1
module load miniconda3
source activate kerastest
export LD_LIBRARY_PATH=~/.conda/envs/kerastest/lib/:$LD_LIBRARY_PATH
python image_classification_from_scratch.py
```

在思源一号上，使用如下脚本来提交该算例作业：

```
#!/bin/bash
#SBATCH -p a100
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --ntasks-per-node 1
#SBATCH --cpus-per-task 6
#SBATCH --gres gpu:1

cd ~/HPCTesting/keras/case1
module load miniconda3
source activate kerastest
export LD_LIBRARY_PATH=~/.conda/envs/kerastest/lib/:$LD_LIBRARY_PATH
python image_classification_from_scratch.py
```

将以上脚本保存为 `test.slurm`，使用 `sbatch test.slurm` 来交作业。

参考资料

- [Keras 官网](#)

Apache TVM

简介

Apache TVM 是一个开源的机器学习编译框架。旨在帮助机器学习工程师在任何硬件平台优化、运行计算任务。

 π 超算上的 TVM

π 超算上部署了预编译版本的 TVM-0.9.dev0。

版本	平台	构建方式	名称
0.9.dev0		容器	/lustre/share/img/tvm-0.9.dev0.sif
1.6.dev0		容器	/dssg/share/imgs/tvm/0.9.dev0.sif 思源平台

提交 TVM 作业

在思源超算提交 TVM 作业：

```
#!/bin/bash

#SBATCH --job-name tvm
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p a100
#SBATCH --gres gpu:1
#SBATCH --cpus-per-task 6

IMAGE_PATH=/dssg/share/imgs/tvm/0.9.dev0.sif

singularity run --nv --env TVM_LOG_DEBUG=1 $IMAGE_PATH python ...
```

在闵行超算提交 TVM 作业：

```
#!/bin/bash

#SBATCH --job-name tvm
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p dgx2
#SBATCH --gres gpu:1
```

(续下页)

(接上页)

```
IMAGE_PATH=/lustre/share/img/tvm-0.9.dev0.sif

singularity run --nv --env TVM_LOG_DEBUG=1 $IMAGE_PATH python ...
```

使用如下指令提交:

```
$ sbatch slurm.test
```

TVM 单卡测试

算例下载

```
$ git clone https://github.com/SJTU-HPC/HPCTesting.git
$ cd HPCTesting/tvm/case1
```

A100 测试环境配置

```
$ singularity run /dssg/share/imgs/tvm/0.9.dev0.sif pip install -r \
↪requirements.txt
```

DGX2 测试环境配置

```
$ singularity run /lustre/share/img/tvm-0.9.dev0.sif pip install -r \
↪requirements.txt
```

A100 测试脚本:

```
#!/bin/bash

#SBATCH --job-name tvm
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p a100
#SBATCH --gres gpu:1
#SBATCH --cpus-per-task 6

IMAGE_PATH=/dssg/share/imgs/tvm/0.9.dev0.sif

singularity run --nv --env TVM_LOG_DEBUG=1 $IMAGE_PATH python test.py
```

DGX2 测试脚本:

```
#!/bin/bash

#SBATCH --job-name tvm
#SBATCH -N 1
```

(续下页)

(接上页)

```
#SBATCH -n 1
#SBATCH -p dgx2
#SBATCH --gres gpu:1

IMAGE_PATH=/lustre/share/img/tvm-0.9.dev0.sif

singularity run --nv --env TVM_LOG_DEBUG=1 $IMAGE_PATH python test.py
```

结果.. code:: console

```
optimized: {'mean': 518.929668366909, 'median': 512.8163313493133, 'std':
10.976080937596128} unoptimized: {'mean': 592.0918963477015, 'median':
587.7139701507986, 'std': 10.555673599042604}
```

参考资料

- [Apache TVM 官网](#)

PyTorch Geometric (PyG)

简介

PyTorch Geometric (PyG) 是基于 PyTorch 的图神经网络 (Graph Neural Networks, GNNs) 训练工具包。PyG 依赖的 PyTorch 提供 CPU 和 GPU 版本，这个文档将介绍如何在思源一号上安装 GPU 版 PyG 2.3.1，并在 A100 加速卡上运行算例。如果您需要在 CPU 或者使用 DGX2 上的 V100 加速卡上运行 PyG，您可以咨询技术服务团队。

安装 PyG

我们将申请思源一号上的 1 个计算节点用于执行安装流程。PyG 将被安装到名为 `pyg-gpu-a100` 的 Conda 环境中。

申请计算节点：

```
$ srun -p 64c512g -n 1 --pty /bin/bash
```

在计算节点上加载模块，创建并激活 `pyg-gpu-a100` 环境：

```
$ module load miniconda3
$ conda create -n pyg-gpu-a100
$ source activate pyg-gpu-a100
```

安装 PyTorch 2.0 GPU 版：

```
$ conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch
↪-c nvidia
```

安装与 PyTorch 所用 CUDA 版本匹配的 PyG 依赖包：

```
$ pip install pyg_lib torch_scatter torch_sparse torch_cluster torch_spline_
→conv -f https://data.pyg.org/whl/torch-2.0.0+cu118.html
```

安装 PyG:

```
$ pip install torch_geometric
```

以交互式方式使用 **PyG**

我们将申请 A100 计算资源, 激活 `pyg-gpu-a100` 环境后, 运行 PyG 示例程序。

申请 A100 计算资源:

```
$ srun -p a100 -N 1 -n 1 --gres=gpu:1 --cpus-per-task=16 --pty /bin/bash
```

加载模块, 激活 `pyg-gpu-a100` 环境:

```
$ module load miniconda3
$ source activate pyg-gpu-a100
```

确认 PyTorch 版本高于 1.12、PyTorch 使用的 CUDA 版本与计算节点的 GPU 驱动版本相匹配:

```
$ python -c "import torch; print(torch.__version__)"
2.0.1
$ python -c "import torch; print(torch.version.cuda)"
11.8
$ nvidia-smi | head -4
+-----+
→--+
| NVIDIA-SMI 525.85.12      Driver Version: 525.85.12      CUDA Version: 12.0  →
→  |
|-----+-----+-----+
→--+
```

下载并解压算例:

```
$ wget https://codeload.github.com/pyg-team/pytorch_geometric/tar.gz/refs/
→tags/2.3.1 -O pyg-2.3.1.tar.gz
$ tar xzvpf pyg-2.3.1.tar.gz
$ cd pytorch_geometric-2.3.1/examples
```

运行名为 `dna` 的算例, 该算例做运行 200 个 Epoch 训练, 耗时约 1 分钟。

```
$ python dna.py
...
Epoch: 200, Train: 0.9945, Val: 0.8856, Test: 0.8584
```

以 SLURM 批处理方式使用 PyG

我们将交互式运行 PyG 算例的过程整理成如下 SLURM 作业脚本，然后运行 `sbatch pyg.slurm` 提交：

```
#!/bin/bash

#SBATCH --job-name=pyg
#SBATCH --partition=a100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load miniconda3
source activate pyg-gpu-a100

python dna.py
```

参考资料

- PyTorch Geometric (PyG) <https://pytorch-geometric.readthedocs.io/>
- PyTorch <https://pytorch.org>

Detectron2

简介

Detectron2 是 Facebook AI Research 开源的计算机视觉库，它是基于 PyTorch 框架构建的。Detectron2 提供了一系列丰富的功能和灵活的组件，用于实现图像和视频中的目标检测、实例分割、关键点检测等任务。

Detectron2 构建在 Detectron 的基础上，并进行了全面的重写和重构，以提供更高的性能、更好的模块化设计和更强的扩展性。它采用了现代化的模型架构，如 Faster R-CNN、Mask R-CNN、Panoptic FPN 等，并提供了训练和推理的各种工具和接口。

这个文档将介绍如何在思源一号上安装 GPU 版 Detectron2，并在 A100 加速卡上运行 Inference Demo。

安装 Detectron2

我们将申请思源一号上的 1 个计算节点用于执行安装流程。Detectron2 将被安装到名为 `detectron2` 的 Conda 环境中。

申请计算节点：

```
$ srun -p 64c512g -n 1 --pty /bin/bash
```

在计算节点上加载模块，创建并激活 `detectron2` 环境：

```
$ module load miniconda3
$ conda create -n detectron2
$ source activate detectron2
```

安装 PyTorch 2.0 GPU 版：

```
$ conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch_
↪-c nvidia
```

安装与 PyTorch 所用 CUDA 版本匹配的 detectron2 依赖包：

```
$ pip install pyg_lib torch_scatter torch_sparse torch_cluster torch_spline_
↪conv -f https://data.pyg.org/whl/torch-2.0.0+cu118.html
$ pip install opencv-python
```

安装 detectron2：

```
$ git clone https://github.com/facebookresearch/detectron2.git
$ python -m pip install -e detectron2
```

以交互式方式使用 **detectron2**：

我们将申请 A100 计算资源，激活 `detectron2` 环境后，运行 detectron2 demo 示例程序。

申请 A100 计算资源：

```
$ srun -p a100 -N 1 -n 1 --gres=gpu:1 --cpus-per-task=16 --pty /bin/bash
```

加载 cuda 模块，激活 `detectron2` 环境：

```
$ module load miniconda3
$ source activate detectron2
$ module load cuda/11.8.0
```

确认 PyTorch 版本高于 1.12、PyTorch 使用的 CUDA 版本与计算节点的 GPU 驱动版本相匹配：

```
$ python -c "import torch; print(torch.__version__)"
2.0.1
$ python -c "import torch; print(torch.version.cuda)"
11.8
$ nvidia-smi | head -4
+-----+
↪---+
| NVIDIA-SMI 525.85.12      Driver Version: 525.85.12      CUDA Version: 12.0  ↪
```

(续下页)

(接上页)

测试 demo 算例，进入 demo 路径内，并准备一张测试输入图片上传至 ‘./testimages/test1.jpg’内，并创建 output 目录，执行 demo 算例的命令如下：

```
$ cd detectron2/demo
$ python demo.py --config-file ../configs/COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml --input ./testimages/test1.jpg --output ./output/ --opts MODEL.WEIGHTS detectron2://COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x/137849600/model_final_f10217.pkl
```

运行名为`demo.py`的算例，该算例耗时7秒钟，output目录内即可生成输出图片。

```
$ python demo.py --config-file ../configs/COCO-InstanceSegmentation/mask_
↳ rcnn_R_50_FPN_3x.yaml --input ./testimages/test1.jpg --output ./output/
↳ --opts MODEL.WEIGHTS detectron2://COCO-InstanceSegmentation/mask_rcnn_R_
↳ 50_FPN_3x/137849600/model_final_f10217.pkl
...
[08/20 11:29:40 detectron2]: ./testimages/test1.jpg: detected 21 instances
↳ in 6.80s 100
↳ %|
↳ 1/1 [00:07<00:00, 7.17s/it]
$ ls output/
test1.jpg
```

以 SLURM 批处理方式使用 Detectron2

我们将交互式运行 **Detectron2** 算例的过程整理成如下 **SLURM** 作业脚本，然后运行 ``sbatch Detectron2.slurm`` 提交：

```
#!/bin/bash

#SBATCH --job-name=Detectron2
#SBATCH --partition=a100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1
#SBATCH --mail-type=end
#SBATCH --mail-user=YOU@EMAIL.COM
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load miniconda3
source activate detectron2
module load cuda/11.8.0
```

(续下页)

```
cd detectron2/demo
python demo.py --config-file ../configs/COCO-InstanceSegmentation/mask_rcnn_
→R_50_FPN_3x.yaml --input ./testimages/test1.jpg --output ./output/ --
→opts MODEL.WEIGHTS detectron2://COCO-InstanceSegmentation/mask_rcnn_R_50_
→FPN_3x/137849600/model_final_f10217.pkl
```

参考资料

- Detectron2 <https://detectron2.readthedocs.io/en/latest/tutorials/install.html>
- PyTorch <https://pytorch.org>

ollama

简介

ollama 是一个开源的大型语言模型服务工具，它允许用户在自己的硬件环境中轻松部署和使用大规模预训练模型。**Ollama** 的主要功能是在 **Docker** 容器内部署和管理大型语言模型 (LLM)，使得用户能够快速地在本地运行这些模型。它简化了部署过程，通过简单的安装指令，用户可以执行一条命令就在本地运行开源大型语言模型，例如 Llama 3。

- 能直接运行大模型，与大模型进行对话
- **ollama** 命令具有管理大模型的能力
- 本地大模型安全可靠
- 终端直接开始聊天
- 社区提供了支持 web api 方式访问 WebUI

在交我算上安装 **ollama**

下载 **ollama-linux-amd64** 文件

下载方法 1：通过 **github** 链接下载

点击 <https://ollama.com/download/ollama-linux-amd64> 即可下载。

下载方法 2：交大云盘

我们已将此文件上传交大云盘，可以直接在交大云盘下载。<https://pan.sjtu.edu.cn/web/share/15be420e80487d8d972045042a32c90d>

建议修改 **ollama-linux-amd64** 文件名为 **ollama**，方便后续操作。可先在本地修改文件名后再上传，或上传超算后使用命令行修改：`mv ollama-linux-amd64 ollama`

修改 **ollama** 文件可执行权限

```
$ chmod +x /path/to/ollama
```

备注：需要将 **path** 部分修改为存储 **ollama** 的路径

配置 **Path** 环境变量

```
$ export PATH=/path/to/ollama:$PATH
```

为方便使用，可以将该命令添加到 **.bashrc** 文件中

部署 **Llama3**

申请计算资源

llama3:8b 推理需要 1 块 GPU 卡，申请交互式计算资源的命令如下：

Pi 2.0:

```
$ srun -p dgx2 -N 1 -n 1 --gres=gpu:1 --cpus-per-task=6 --pty /bin/bash
```

思源 1 号：

```
$ srun -p a100 -N 1 -n 1 --gres=gpu:1 --cpus-per-task 16 --pty /bin/bash
```

运行 **ollama** 服务

```
$ ollama serve
```

打开一个新的终端，进入计算节点并运行模型

```
$ ollama run llama3:8b
```

备注：初次运行 **ollama run llama3:8b**，需要下载一个 4.7G 的文件，需要等待一定时间。

接下来就可以愉快地与大模型对话了！

```
>>> 你好
© 你好！我也很好！欢迎你来这个讨论平台，我愿意和你聊天、分享知识或只是随便讨论。有什么话题想聊？ ☺

>>> 请
>>> 原来你会 中文啊
哈哈，的确是！我是一个中国人开发的chatbot，所以它的语言能力主要集中在中文上。如果你想聊中文话题，我很高兴和你交流！或者，如果你需要帮助或建议关于中文学习，你也可以问我。 ☺

>>> 你不是llama3吗
© Ahah, you're absolutely right! I am indeed LLaMA 3, an AI model trained by Meta AI that can understand and respond to human input in a conversational manner. I'm excited to be chatting with you in Chinese! 🐼 What would you like to talk about? ☺

>>> 那你刚才为什么说自己是中国人开发的chatbots呢
© Ahah, I think I might have gotten a bit carried away with the conversation! I apologize for the confusion. As LLaMA 3, I'm actually not a Chinese-developed chatbot, but rather an AI model trained by Meta AI on a large corpus of text data, including but not limited to Chinese language texts.

My training data includes a wide range of texts from various sources and languages, which enables me to understand and respond to input in multiple languages, including Chinese. So while I'm not a "Chinese person" developed chatbot, I do have the capability to converse with you in Chinese! ☺
```

参考资料

- [ollama 官网](#)

4.10.5 原子分子工程计算

ABACUS

简介

ABACUS 是一个专为大规模电子结构模拟而设计的开源软件包。该软件自 2007 年以来由中国科学技术大学中国科学院量子信息重点实验室的团队开发。

运行 **ABACUS** 方法

π 和思源一号集群已经安装好了 **ABACUS** 运行环境。要运行模拟，请遵循以下步骤。

1. 申请计算节点

π 超算

```
srun -p cpu -n 4 --pty /bin/bash
```

思源一号

```
srun -p 64c512g -n 4 --pty /bin/bash
```

2. 进入计算节点后，加载模块

```
module load miniconda3
```

3. 激活环境

```
conda init  
  
source ~/.bashrc  
  
conda activate /lustre/share/conda_env/abacus_env #在 $\pi$ 集群上  
conda activate /dssg/share/conda_env/abacus_env #在思源一号集群上
```

4. 进入数据目录

```
cd /path/to/your/data_directory
```

5. 执行模拟

i 注意

严禁在登录节点上直接运行命令。必须通过 **Slurm** 调度系统将作业提交到计算节点。

```
OMP_NUM_THREADS=1 mpirun -n 4 abacus
```

参考资料

- [ABACUS 官网](#)

ABINIT

简介

ABINIT 是一个软件套件，可用于计算材料的光学、机械、振动和其他可观察特性的模拟。从密度泛函理论的量子方程开始，可以使用基于 **DFT** 的微扰理论和多体格林函数 (**GW** 和 **DMFT**) 建立高级应用程序。**ABINIT** 可以计算具有任何化学成分的分子、纳米结构和固体，并附带几个完整且强大的原子势表。

可用版本

版本	平台	构建方式	模块名
9.4.2		spack	abinit/9.4.2-gcc-8.3.1-hdf5-openblas-openmpi 思源一号

算例位置

```
/dssg/share/sample/abinit/abinit-9.6.2.tar.gz
```

集群上的 **ABINIT**

- 思源一号上运行 **ABINIT**

思源一号上运行 **ABINIT**

拷贝算例到本地

```
mkdir ~/abinit && cd ~/abinit
cp -r /dssg/share/sample/abinit/abinit-9.6.2.tar.gz ./
tar xf abinit-9.6.2.tar.gz
```

运行脚本

```
#!/bin/bash
#SBATCH --job-name=abinit
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
export ABI_HOME=~/.abinit/abinit-9.6.2
export ABI_TESTS=$ABI_HOME/tests/
export ABI_PSPDIR=$ABI_TESTS/Psps_for_tests/
module load abinit
module load openmpi/4.1.1-gcc-8.3.1
mpirun -np 8 abinit tbs_1.abi
```

运行结果如下所示：

思源一号上 **ABINIT** 的运行时间

abinit/9.4.2-gcc-8.3.1-hdf5-openblas-openmpi				
核数	1	2	4	8
Exec time	0:00:29	0:00:16	0:00:11	0:00:07

参考资料

- ABINIT <http://www.abinit.org>

Amber

简介

Amber 是分子动力学软件，用于蛋白质、核酸、糖等生物大分子的计算模拟。

Amber 为商业软件，根据 **Amber** 官网，非商业性使用的用户在官网登记，阅读并同意 **non-commercial license** 后即可下载软件安装包。

如需使用集群上的 **Amber**，请下载 **non-commercial license**，按照下面模板发送邮件至 **hpc** 邮箱。

邮件模板：我是超算账号 **xxx** 使用人，已经阅读了附件的 **License**，同意该许可，并确认所有使用均为非商业性使用。

算例存放位置

```

思源    : /dssg/share/sample/amber
π2.0    : /lustre/share/sample/amber

```

算例结构如下

```

tree amber:
├── inpcrd
├── mdin
└── prmtop

```

集群上的 **Amber**

- 思源一号 *Amber*
- *π2.0* *Amber*

思源一号上的 **Amber**

思源上的 **Amber2022**

amber_GPU.slurm 内容如下:

```

#!/bin/bash
#SBATCH --job-name=Amber_gpu
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=16
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load amber/2022-cuda-11.5.119
pmemd.cuda -O -i mdin -o mdout -p prmtop -c inpcrd

```

amber_MPI.slurm 内容如下:

```

#!/bin/bash

#SBATCH --job-name=Amber_mpi
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load  amber/2022-intel-2021.4.0
mpirun pmemd.MPI -O -i mdin -o mdout -p prmtop -c inpcrd

```

思源上的 **Amber2020**

amber_MPI.slurm 内容如下:

```
#!/bin/bash

#SBATCH --job-name=Amber_mpi
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load  amber/2020-intel-2021.4.0
mpirun pmemd.MPI -O -i mdin -o mdout -p prmtop -c inpcrd
```

π 2.0 上的 **Amber**

π 2.0 上的 **Amber2022**

amber_GPU.slurm 内容如下:

```
#!/bin/bash

#SBATCH --job-name=Amber_gpu
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load amber/2022-cuda-10.1.243
pmemd.cuda -O -i mdin -o mdout -p prmtop -c inpcrd
```

amber_MPI.slurm 内容如下:

```
#!/bin/bash

#SBATCH --job-name=test_amber
#SBATCH --partition=cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load amber/2022-intel-2021.4.0
mpirun pmemd.MPI -O -i mdin -o mdout -p prmtop -c inpcrd
```

π2.0 上的 Amber2020

amber_GPU.slurm 内容如下:

```
#!/bin/bash
#SBATCH --job-name=Amber_gpu
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=6
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load amber/2020-cuda-10.2.89
pmemd.cuda -O -i mdin -o mdout -p prmtop -c inpcrd
```

amber_MPI.slurm 内容如下:

```
#!/bin/bash
#SBATCH --job-name=test_amber
#SBATCH --partition=cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load amber/2020-intel-2021.4.0
mpirun pmemd.MPI -O -i mdin -o mdout -p prmtop -c inpcrd
```

运行结果 (单位为: **s**)

Amber22 在 GPU 上的运行结果

平台	思源	pi 2.0
核数	16core+1GPU	6core+1GPU
Time	60.57	60.99

Amber20 在 GPU 上的运行结果

平台	pi 2.0
核数	6core+1GPU
Time	60.94

Amber22 在 CPU 上的运行结果

平台	思源	pi 2.0	思源	pi 2.0	思源	pi 2.0
核数	64	40	128	80	256	160
Time	446.36	722.14	311.67	428.30	306.37	315.61

Amber20 在 CPU 上的运行结果

平台	思源	pi 2.0	思源	pi 2.0	思源	pi 2.0
核数	64	40	128	80	256	160
Time	441.83	694.00	309.26	430.69	306.83	312.89

参考资料

- [Amber 官网](#)

Basilisk

简介

Basilisk 也是一个开源软件，用于解决自适应直角坐标网格上的偏微分方程

Basilisk 基本使用

超算中使用 Basilisk 可以通过镜像直接调用，pi 2.0 和思源一号上的软件镜像位置不同，脚本中要注意。1. 在 pi 2.0 上，编写 basilisk_test.slurm 脚本，根据软件镜像所在位置进行使用：

```
#!/bin/bash

#SBATCH -J test
#SBATCH -p small
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --cpus-per-task=1

IMAGE_PATH=/lustre/share/img/basilisk/basilisk-8.5.0.sif
singularity exec $IMAGE_PATH qcc --version
```

2. 在思源一号上，编写 basilisk_test.slurm 脚本，根据软件镜像所在位置进行使用：

```
#!/bin/bash

#SBATCH -J test
#SBATCH -p 64c512g
#SBATCH -o %j.out
```

(续下页)

(接上页)

```
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --cpus-per-task=1

IMAGE_PATH=/dssg/share/imgs/basilisk/basilisk-8.5.0.sif
singularity exec $IMAGE_PATH qcc --version
```

3. 提交脚本:

```
sbatch basilisk_test.slurm
```

4. 运行结束后可在.out 文件中得到如下结果:

```
cc (GCC) 8.5.0 20210514 (Red Hat 8.5.0-4)
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

参考资料

- Basilisk 使用手册

Calculix

简介

CalculiX 是一个设计来利用有限元方法求解场问题的软件，其既能够运行在类 Unix(包括 Linux) 平台上，也能在 MS-Windows 上运行。使用 CalculiX，你可以构建有限元模型，对模型进行求解以及后处理。CalculiX 的预处理器和后处理器基于 OpenGL API 开发而成。其解器能够进行线性和非线性计算，包括求解静态、动态和热力学问题的模块。

CalculiX 使用说明

在思源一号上自行安装并使用 CalculiX

1. 使用 conda 创建虚拟环境并安装 CalculiX:

```
srunk -p 64c512g -n 1 --pty /bin/bash
module load miniconda3/4.10.3
conda create --name calculix_test
source activate calculix_test
conda install -c conda-forge calculix==2.17
```

2. 创建一个目录 calculixtest 并进入该目录:

```
mkdir calculixtest
cd calculixtest
```

3. 在该目录下创建如下测试文件 **beam10p.inp**(输入参数文件):

```

**
**   Structure: cantilever beam under pressure.
**   Test objective: C3D10 elements.
**
**NODE,NSET=NALL
  1,      1.,      0.
  2,      0.,      0.
  3,      1.,      1.
  4,      0.5,      0.
  5,      0.5,      0.5
  6,      1.,      0.5
  7,      0.,      1.
  8,      0.,      0.5
  9,      0.5,      1.
 10,      1.,      0.,      8.
 11,      1.,      1.,      8.
 12,      0.,      0.,      8.
 13,      1.,      0.5,      8.
 14,      0.5,      0.5,      8.
 15,      0.5,      0.,      8.
 16,      0.,      1.,      8.
 17,      0.5,      1.,      8.
 18,      0.,      0.5,      8.
 19,      1.,      0.,      2.
 20,      1.,      0.,      1.
 21,      0.5,      0.,      1.
 22,      0.,      0.,      2.
 23,      0.5,      0.,      2.
 24,      0.,      0.,      1.
 25,      1.,      0.,      4.
 26,      1.,      0.,      3.
 27,      0.5,      0.,      3.
 28,      0.,      0.,      4.
 29,      0.5,      0.,      4.
 30,      0.,      0.,      3.
 31,      1.,      0.,      6.
 32,      1.,      0.,      5.
 33,      0.5,      0.,      5.
 34,      0.,      0.,      6.
 35,      0.5,      0.,      6.
 36,      0.,      0.,      5.
 37,      1.,      0.,      7.
 38,      0.5,      0.,      7.
 39,      0.,      0.,      7.
 40,      0.,      1.,      2.
 41,      0.,      0.5,      1.
 42,      0.,      1.,      1.
 43,      0.,      0.5,      2.

```

(续下页)

(接上页)

44,	0.,	1.,	4.				
45,	0.,	0.5,	3.				
46,	0.,	1.,	3.				
47,	0.,	0.5,	4.				
48,	0.,	1.,	6.				
49,	0.,	0.5,	5.				
50,	0.,	1.,	5.				
51,	0.,	0.5,	6.				
52,	0.,	0.5,	7.				
53,	0.,	1.,	7.				
54,	1.,	1.,	2.				
55,	0.5,	1.,	1.				
56,	1.,	1.,	1.				
57,	0.5,	1.,	2.				
58,	1.,	1.,	4.				
59,	0.5,	1.,	3.				
60,	1.,	1.,	3.				
61,	0.5,	1.,	4.				
62,	1.,	1.,	6.				
63,	0.5,	1.,	5.				
64,	1.,	1.,	5.				
65,	0.5,	1.,	6.				
66,	0.5,	1.,	7.				
67,	1.,	1.,	7.				
68,	1.,	0.5,	1.				
69,	1.,	0.5,	2.				
70,	1.,	0.5,	3.				
71,	1.,	0.5,	4.				
72,	1.,	0.5,	5.				
73,	1.,	0.5,	6.				
74,	1.,	0.5,	7.				
149,	0.499871,	0.499871,	3.99987				
150,	0.5,	0.5,	2.				
151,	0.5,	0.5,	1.				
152,	0.249935,	0.249935,	2.99994				
153,	0.249935,	0.749935,	3.99994				
154,	0.249935,	0.249935,	3.99994				
155,	0.249935,	0.749935,	4.99994				
156,	0.249935,	0.249935,	4.99994				
157,	0.749935,	0.749935,	3.99994				
158,	0.249935,	0.749935,	2.99994				
159,	0.749935,	0.749935,	4.99994				
160,	0.5,	0.5,	7.				
161,	0.5,	0.5,	6.				
162,	0.749935,	0.249935,	2.99994				
163,	0.749935,	0.249935,	3.99994				
164,	0.749935,	0.249935,	4.99994				
*ELEMENT, TYPE=C3D10, ELSET=EALL							
37,	2,	22,	19,	40,	24,	23,	21,

(续下页)

(续下页)

(接上页)

41,	43,	150					
38,	2,	3,	54,	1,	5,	56,	151,
4,	6,	68					
39,	22,	44,	28,	149,	45,	47,	30,
152,	153,	154					
40,	28,	34,	149,	48,	36,	156,	154,
49,	51,	155					
41,	40,	7,	54,	2,	42,	55,	57,
41,	8,	151					
42,	54,	2,	19,	40,	151,	21,	69,
57,	41,	150					
43,	40,	44,	149,	58,	46,	153,	158,
59,	61,	157					
44,	44,	48,	149,	62,	50,	155,	153,
63,	65,	159					
45,	34,	16,	11,	48,	52,	17,	160,
51,	53,	66					
46,	11,	62,	31,	34,	67,	73,	74,
160,	161,	35					
47,	12,	11,	34,	16,	14,	160,	39,
18,	17,	52					
48,	22,	19,	40,	149,	23,	150,	43,
152,	162,	158					
49,	7,	3,	54,	2,	9,	56,	55,
8,	5,	151					
50,	22,	28,	25,	149,	30,	29,	27,
152,	154,	163					
51,	1,	2,	19,	54,	4,	21,	20,
68,	151,	69					
52,	58,	19,	40,	54,	70,	150,	59,
60,	69,	57					
53,	28,	25,	149,	31,	29,	163,	154,
33,	32,	164					
54,	62,	48,	34,	11,	65,	51,	161,
67,	66,	160					
55,	149,	48,	34,	62,	155,	51,	156,
159,	65,	161					
56,	34,	11,	10,	31,	160,	13,	38,
35,	74,	37					
57,	10,	11,	34,	12,	13,	160,	38,
15,	14,	39					
58,	58,	25,	62,	149,	71,	72,	64,
157,	163,	159					
59,	19,	25,	58,	149,	26,	71,	70,
162,	163,	157					
60,	22,	19,	149,	25,	23,	162,	152,
27,	26,	163					
61,	25,	62,	149,	31,	72,	159,	163,
32,	73,	164					

(续下页)

(接上页)

```

        62,      28,      48,      149,      44,      49,      155,      154,
        47,      50,      153
        63,      28,      34,      31,      149,      36,      35,      33,
154,      156,      164
        64,      34,      62,      31,      149,      161,      73,      35,
156,      159,      164
        65,      22,      149,      40,      44,      152,      158,      43,
        45,      153,      46
        66,      44,      62,      149,      58,      63,      159,      153,
        61,      64,      157
        67,      149,      58,      19,      40,      157,      70,      162,
158,      59,      150
*NSET,NSET=FIX
1,4,2,6,5,8,3,9,7
*NSET,NSET=LOAD
10,15,12,13,14,18,11,17,16
*BOUNDARY
1,1,2
3,1,1
FIX,3,3
*MATERIAL,NAME=EL
*ELASTIC
210000.,.3
*SOLID SECTION,ELSET=EALL,MATERIAL=EL
*STEP
*STATIC
*CLOAD
LOAD,2,1.
*NODE PRINT,NSET=NALL
U,RF
*EL PRINT,ELSET=EALL
S
*END STEP

```

4. 在该目录下创建如下作业提交脚本 calculixtest.slurm:

```

#!/bin/bash

#SBATCH --job-name=calculixtest
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

ccx beam10p

```

5. 使用如下命令提交作业:

```
sbatch calculixtest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
*****

CalculiX Version 2.17, Copyright(C) 1998-2020 Guido Dhondt
CalculiX comes with ABSOLUTELY NO WARRANTY. This is free
software, and you are welcome to redistribute it under
certain conditions, see gpl.htm

*****

You are using an executable made on Wed Oct 14 07:29:32 UTC 2020

The numbers below are estimated upper bounds

number of:

nodes:          164
elements:        67
one-dimensional elements:          0
two-dimensional elements:          0
integration points per element:      4
degrees of freedom per node:        3
layers per element:                  1

distributed facial loads:            0
distributed volumetric loads:        0
concentrated loads:                  9
single point constraints:            12
multiple point constraints:          1
terms in all multiple point constraints: 1
tie constraints:                     0
dependent nodes tied by cyclic constraints: 0
dependent nodes in pre-tension constraints: 0

sets:          4
terms in all sets:      229

materials:          1
constants per material and temperature:      2
temperature points per material:              1
plastic data points per material:             0

orientations:          0
amplitudes:            2
data points in all amplitudes:      2
print requests:        3
```

(续下页)

(接上页)

```

transformations:          0
property cards:          0

STEP                1

Static analysis was selected

Decascading the MPC's

Determining the structure of the matrix:
number of equations
258
number of nonzero lower triangular matrix elements
6834

Using up to 1 cpu(s) for the stress calculation.

Using up to 1 cpu(s) for the symmetric stiffness/mass contributions.

Factoring the system of equations using the symmetric spooles solver
Using up to 1 cpu(s) for spooles.

Using up to 1 cpu(s) for the stress calculation.

Job finished

Total CalculiX Time: 0.034328

```

在 **pi2.0** 上自行安装并使用 **Calculix**

1. 使用 **conda** 创建虚拟环境并安装 **Calculix**:

```

srun -p small -n 1 --pty /bin/bash
module load miniconda3/4.8.2
conda create --name calculix_test
source activate calculix_test
conda install -c conda-forge calculix==2.17

```

2. 此步骤和上文完全相同;
3. 此步骤和上文完全相同;
4. 在该目录下创建如下作业提交脚本 **calculixtest.slurm**:

```
#!/bin/bash

#SBATCH --job-name=calculixtest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

ccx beam10p
```

5. 使用如下命令提交作业:

```
sbatch calculixtest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
*****

CalculiX Version 2.17, Copyright (C) 1998-2020 Guido Dhondt
CalculiX comes with ABSOLUTELY NO WARRANTY. This is free
software, and you are welcome to redistribute it under
certain conditions, see gpl.htm

*****

You are using an executable made on Wed Oct 14 07:29:32 UTC 2020

The numbers below are estimated upper bounds

number of:

nodes:                164
elements:              67
one-dimensional elements:      0
two-dimensional elements:      0
integration points per element:      4
degrees of freedom per node:      3
layers per element:      1

distributed facial loads:      0
distributed volumetric loads:      0
concentrated loads:      9
single point constraints:      12
multiple point constraints:      1
terms in all multiple point constraints:      1
tie constraints:      0
```

(续下页)

(接上页)

```

dependent nodes tied by cyclic constraints:          0
dependent nodes in pre-tension constraints:         0

sets:                4
terms in all sets:    229

materials:            1
constants per material and temperature:            2
temperature points per material:                    1
plastic data points per material:                   0

orientations:        0
amplitudes:           2
data points in all amplitudes:                      2
print requests:      3
transformations:      0
property cards:      0

STEP                1

Static analysis was selected

Decascading the MPC's

Determining the structure of the matrix:
number of equations
258
number of nonzero lower triangular matrix elements
6834

Using up to 1 cpu(s) for the stress calculation.

Using up to 1 cpu(s) for the symmetric stiffness/mass contributions.

Factoring the system of equations using the symmetric spooles solver
Using up to 1 cpu(s) for spooles.

Using up to 1 cpu(s) for the stress calculation.

Job finished

Total CalculiX Time: 0.034328

```

参考资料

- [Calculix 官网](#)
- [Calculix 案例文件](#)

Cantera

简介

Cantera 是一个化学反应动力学开源求解套件，可以处理 Chemkin 格式的动力学机理、热力学和输运性质文件，通过适当编程即可实现常用反应器 (激波管、快速压缩机、搅拌反应器等) 的模拟及敏感性分析等功能。

Cantera 使用说明

在思源一号上自行安装并使用 **Cantera**

1. 使用 conda 创建虚拟环境并安装 Cantera:

```
srunc -p 64c512g -n 1 --pty /bin/bash
module load miniconda3/4.10.3
conda create --name cantera_test
source activate cantera_test
conda install -c cantera/label/dev cantera
```

2. 创建一个目录 canteratest 并进入该目录:

```
mkdir canteratest
cd canteratest
```

3. 在该目录下创建如下测试文件 cantera_test.py:

```
"""
Compute the "equilibrium" and "frozen" sound speeds for a gas

Requires: cantera >= 2.5.0
Keywords: thermodynamics, equilibrium
"""

import cantera as ct
import math

def equilSoundSpeeds(gas, rtol=1.0e-6, max_iter=5000):
    """
    Returns a tuple containing the equilibrium and frozen sound speeds for a
    gas with an equilibrium composition. The gas is first set to an
    equilibrium state at the temperature and pressure of the gas, since
    otherwise the equilibrium sound speed is not defined.
    """
```

(续下页)

(接上页)

```

"""

# set the gas to equilibrium at its current T and P
gas.equilibrate('TP', rtol=rtol, max_iter=max_iter)

# save properties
s0 = gas.s
p0 = gas.P
r0 = gas.density

# perturb the pressure
p1 = p0*1.0001

# set the gas to a state with the same entropy and composition but
# the perturbed pressure
gas.SP = s0, p1

# frozen sound speed
afrozen = math.sqrt((p1 - p0)/(gas.density - r0))

# now equilibrate the gas holding S and P constant
gas.equilibrate('SP', rtol=rtol, max_iter=max_iter)

# equilibrium sound speed
aequil = math.sqrt((p1 - p0)/(gas.density - r0))

# compute the frozen sound speed using the ideal gas expression as a
↪check
gamma = gas.cp/gas.cv
afrozen2 = math.sqrt(gamma * ct.gas_constant * gas.T /
                    gas.mean_molecular_weight)

return aequil, afrozen, afrozen2

# test program
if __name__ == "__main__":
    gas = ct.Solution('gri30.yaml')
    gas.X = 'CH4:1.00, O2:2.0, N2:7.52'
    for n in range(27):
        T = 300.0 + 100.0 * n
        gas.TP = T, ct.one_atm
        print(T, equilSoundSpeeds(gas))

```

4. 在该目录下创建如下作业提交脚本 `canteratest.slurm`:

```

#!/bin/bash

#SBATCH --job-name=canteratest

```

(续下页)

(接上页)

```
#SBATCH --partition=64c512g
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

python3 cantera_test.py
```

5. 使用如下命令提交作业:

```
sbatch canteratest.slurm
```

6. 作业完成后在.out 文件中可看到如下结果:

```
300.0 (351.81897215910766, 351.8225040320234, 351.8256504699811)
400.0 (404.3283414405797, 404.60713567095115, 404.6106870323977)
500.0 (450.60495699772054, 450.5397010505711, 450.54335869692443)
600.0 (493.2014592526757, 491.4792382523635, 491.4832797222876)
700.0 (531.9259905892586, 528.573790448385, 528.5780936855717)
800.0 (565.8023687396801, 562.6519887313171, 562.6563319848512)
900.0 (594.787246530831, 594.3886921418637, 594.3928468410445)
1000.0 (623.1834570351056, 623.1967311658512, 624.387559050875)
1100.0 (652.9331414338375, 653.0237878409358, 653.0280730486886)
1200.0 (680.2903244154709, 680.3796031034527, 680.3839821434552)
1300.0 (706.4795851760939, 706.6231852692649, 706.627654889073)
1400.0 (731.5765086318436, 731.8946265500344, 731.8991788491217)
1500.0 (755.5871134959623, 756.3126654597705, 756.3172844867763)
1600.0 (778.4547221738709, 779.9838934521655, 779.9885516424242)
1700.0 (800.0747386258183, 803.0113981678297, 803.0160523778995)
1800.0 (820.3501225230362, 825.5029866841091, 825.5075790369601)
1900.0 (839.2604068697004, 847.5787433425231, 847.5832049739269)
2000.0 (856.9424757185266, 869.3775874590473, 869.3818488069585)
2100.0 (873.7271469549471, 891.0618214354686, 891.0658256580159)
2200.0 (889.9263467742846, 912.8186991059179, 912.8224139325766)
2300.0 (906.658936254018, 934.8583105569138, 934.8617342865722)
2400.0 (923.8715214357717, 957.4080987743781, 957.411256949987)
2500.0 (942.1089113756211, 980.7060227583402, 980.7089610687678)
2600.0 (961.7644659835721, 1004.9962598901973, 1004.9990315355183)
2700.0 (982.983503202847, 1030.5311450749714, 1030.5338048850456)
2800.0 (1005.8548751928753, 1057.5791215213358, 1057.5817227102573)
2900.0 (1030.5547401650322, 1086.430611844347, 1086.433204007728)
```

在 **pi2.0** 上自行安装并使用 **Cantera**

1. 使用 **conda** 创建虚拟环境并安装 **Cantera**:

```

srun -p small -n 1 --pty /bin/bash
module load miniconda3/4.8.2
conda create --name cantera_test
source activate cantera_test
conda install -c cantera/label/dev cantera

```

2. 此步骤和上文完全相同;

3. 此步骤和上文完全相同;

4. 在该目录下创建如下作业提交脚本 **canteratest.slurm**:

```

#!/bin/bash

#SBATCH --job-name=canteratest
#SBATCH --partition=small
#SBATCH --ntasks-per-node=1
#SBATCH -n 1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

python3 cantera_test.py

```

5. 使用如下命令提交作业:

```

sbatch canteratest.slurm

```

6. 作业完成后在.out 文件中可看到如下结果:

```

300.0 (351.81897215910766, 351.8225040320234, 351.8256504699811)
400.0 (404.3283414405797, 404.60713567095115, 404.6106870323977)
500.0 (450.60495699772054, 450.5397010505711, 450.54335869692443)
600.0 (493.2014592526757, 491.4792382523635, 491.4832797222876)
700.0 (531.9259905892586, 528.573790448385, 528.5780936855717)
800.0 (565.8023687396801, 562.6519887313171, 562.6563319848512)
900.0 (594.787246530831, 594.3886921418637, 594.3928468410445)
1000.0 (623.1834570351056, 623.1967311658512, 624.387559050875)
1100.0 (652.9331414338375, 653.0237878409358, 653.0280730486886)
1200.0 (680.2903244154709, 680.3796031034527, 680.3839821434552)
1300.0 (706.4795851760939, 706.6231852692649, 706.627654889073)
1400.0 (731.5765086318436, 731.8946265500344, 731.8991788491217)
1500.0 (755.5871134959623, 756.3126654597705, 756.3172844867763)
1600.0 (778.4547221738709, 779.9838934521655, 779.9885516424242)
1700.0 (800.0747386258183, 803.0113981678297, 803.0160523778995)

```

(续下页)

(接上页)

```
1800.0 (820.3501225230362, 825.5029866841091, 825.5075790369601)
1900.0 (839.2604068697004, 847.5787433425231, 847.5832049739269)
2000.0 (856.9424757185266, 869.3775874590473, 869.3818488069585)
2100.0 (873.7271469549471, 891.0618214354686, 891.0658256580159)
2200.0 (889.9263467742846, 912.8186991059179, 912.8224139325766)
2300.0 (906.658936254018, 934.8583105569138, 934.8617342865722)
2400.0 (923.8715214357717, 957.4080987743781, 957.411256949987)
2500.0 (942.1089113756211, 980.7060227583402, 980.7089610687678)
2600.0 (961.7644659835721, 1004.9962598901973, 1004.9990315355183)
2700.0 (982.983503202847, 1030.5311450749714, 1030.5338048850456)
2800.0 (1005.8548751928753, 1057.5791215213358, 1057.5817227102573)
2900.0 (1030.5547401650322, 1086.430611844347, 1086.433204007728)
```

参考资料

- [Cantera 官网](#)
- [安装 Cantera 知乎](#)

CASM

简介

CASM 是一款用于研究多成分晶体固体的第一原理统计力学软件

软件安装

此项目为开源软件，建议使用 **conda** 进行安装

1. 申请计算节点

由于软件安装需要较多资源，登陆节点的资源限制会导致安装任务无法正常进行，因此需要先申请计算节点资源，在计算节点上进行安装。

2.conda 安装

```
conda create -n casm --override-channels -c prisms-center -c conda-forge
→ casm-cpp=1.2.0 python=3
```

3. 确认安装结果

```
source activate casm
ccasm --version
#可以正常输出版本即为成功安装
```

CESM

简介

CESM (Community Earth System Model) 是一个完全耦合的全球气候模型, 可用于地球过去、现在和未来气候状态的模拟。

CESM 源码包的位置

思源一号 : /dssg/share/data/cesm/my_cesm_sandbox.tar.gz

测试数据的位置

思源一号 : /dssg/share/data/cesm/inputdata/

思源一号上 CESM 的使用

导入 CESM 依赖环境

用户需要自己的目录下编辑 CESM 才能正常使用, 我们已在思源上部署了 CESM 的依赖软件:

版本	调用方式
2.2	module load cesm/2.2-gcc-8.3.1

上述模块中包含的依赖软件为:

```
HDF5 : 1.10.1
OpenBLAS: 0.3.20
netcdf-c : 4.4.1.1
netcdf-fortran : 4.4.1
parallel-netcdf : 1.9.0
zlib : 1.2.10
```

个人用户目录下构建 CESM 执行环境

申请计算节点与导入执行环境

```
srunk -p 64c512g --exclusive --pty /bin/bash
module load cesm
```

```
mkdir ~/CESM2.2 && cd ~/CESM2.2
cp /dssg/share/data/cesm/my_cesm_sandbox.tar.gz ./
tar xf my_cesm_sandbox.tar.gz
cd my_cesm_sandbox/cime/scripts/
```

(续下页)

(接上页)

```
./create_newcase --case test_12 --compset X --res f19_g16 --mach=centos7-  
→linux  
cd test_12/  
./case.setup  
cp -r /dssg/share/data/cesm/Macros.make ./  
./case.build --skip-provenance-check
```

执行作业

作业脚本

```
#!/bin/bash  
#SBATCH --job-name=cesm  
#SBATCH --partition=64c512g  
#SBATCH -N 1  
#SBATCH --ntasks-per-node=64  
#SBATCH --output=%j.out  
#SBATCH --error=%j.err  
module load cesm  
ulimit -s unlimited  
./case.submit
```

运行结果

思源一号上的 CESM

```
copying file /dssg/home/acct-hpc/hpc/cesm/scratch/mycase_single/run/mycase_  
→single.cpl.r.0001-01-06-00000.nc to /dssg/home/acct-hpc/hpc/cesm/archive/  
→mycase_single/rest/0001-01-06-00000/mycase_single.cpl.r.0001-01-06-00000.  
→nc  
Archiving restarts for dart (esp)  
Archiving history files for drv (cpl)  
Archiving history files for dart (esp)  
st_archive completed  
Submitted job case.run with id None  
Submitted job case.st_archive with id None
```

参考资料

- [CESM 官方网站](#)
- [CESM User Guide](#)

CP2k

CP2K 是一个量子化学和固态物理软件包，可以对固态，液态，分子，周期性，材料，晶体和生物系统进行原子模拟。CP2K 为不同的建模方法提供了通用框架。支持的理论水平包括 DFTB, LDA, GGA, MP2, RPA, 半经验方法 (AM1, PM3, PM6, RM1, MNDO 等) 和经典力场 (AMBER, CHARMM 等)。CP2K 可以使用 NEB 或二聚体方法进行分子动力学，元动力学，蒙特卡洛，埃伦菲斯特动力学，振动分析，核心能谱，能量最小化和过渡态优化的模拟。

更多信息请访问：<https://www.cp2k.org/>

语言：Fortran 2008

开源协议：GPL

一句话描述：是一个量子化学和固态物理软件包，可以对固态，液态，分子，周期性，材料，晶体和生物系统进行原子模拟。

可用的版本

版本	平台	构建方式	模块名 集群
8.2		spack	cp2k/8.2-gcc-11.2.0-openblas-openmpi 思源一号
8.2		spack	cp2k/8.2-gcc-11.2.0-cuda-openblas-openmpi 思源一号
2024.1		spack	cp2k/2024.1-intel-2021.4.0 思源一号
2024.1		spack	cp2k/2024.1-intel-2021.4.0 Pi2.0
8.2		spack	cp2k/8.2-gcc-10.3.1 Kunpeng

思源一号上的 CP2K

思源一号上已经预装 CP2K(GPU+CPU 版本)。若不指定版本，将加载默认的 module (标记为 D)。

思源一号上 CPU 版本 cp2k-8.2 示例脚本 cp2k-8.2-cpu.slurm

这里申请了一个节点示例脚本如下：

```
#!/bin/bash

#SBATCH --job-name=cp2k_cpu
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

(续下页)

(接上页)

```

module purge
module load cp2k/8.2-gcc-11.2.0-openblas-openmpi
module load openmpi/4.1.1-gcc-11.2.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun -np $SLURM_NTASKS -x OMP_NUM_THREADS=1 cp2k.psmf ${INPUT_FILE}

```

思源一号上 **CPU** 版本 **cp2k-2024.1** 示例脚本 **cp2k-2024.1-cpu.slurm**

这里申请了一个节点示例脚本如下：

```

#!/bin/bash

#SBATCH --job-name=cp2k_cpu
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load oneapi/2021.4.0
module load cp2k/2024.1-intel-2021.4.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun -np $SLURM_NTASKS cp2k.psmf ${INPUT_FILE}

```

思源一号上 **GPU** 版本 **cp2k-8.2** 示例脚本 **cp2k-8.2-gpu.slurm**

```

#!/bin/bash

#SBATCH --job-name=cp2k_gpu
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge

```

(续下页)

(接上页)

```

module load cp2k/8.2-gcc-11.2.0-cuda-openblas-openmpi
module load cuda/11.5.0
module load openmpi/4.1.1-gcc-11.2.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun -np $SLURM_NTASKS --mca opal_common_ucx_opal_mem_hooks 1 -x OMP_NUM_
  ↳ THREADS=1 cp2k.psmmp ${INPUT_FILE}

```

并使用如下指令提交：

```

$ sbatch cp2k-8.2-cpu.slurm
$ sbatch cp2k-2024.1-cpu.slurm
$ sbatch cp2k-8.2-gpu.slurm

```

π 集群上的 CP2K

π 集群系统中已经预装 CP2K (2024.1, 2023.2 版本)。若不指定版本，将加载默认的 module (标记为 D)。

π 集群上 CPU 版本 cp2k 示例脚本 cp2k-2024.1-cpu.slurm

在 cpu 队列上，总共使用 80 核 cpu 队列每个节点配有 40 核，所以这里使用了 2 个节点：

```

#!/bin/bash

#SBATCH --job-name=cp2k_cpu_test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load oneapi
module load cp2k/2024.1-intel-oneapi-2021.4.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun -np $SLURM_NTASKS cp2k.psmmp ${INPUT_FILE}

```

π 集群上 CPU 版本 cp2k 示例脚本 cp2k-2023.2-cpu.slurm

在 cpu 队列上，总共使用 80 核 cpu 队列每个节点配有 40 核，所以这里使用了 2 个节点：

```
#!/bin/bash

#SBATCH --job-name=cp2k_cpu_test
#SBATCH --partition=cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load oneapi
module load cp2k/2023.2-intel-oneapi-2021.4.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun -np ${SLURM_NPROCS} cp2k.psmf ${INPUT_FILE}
```

并使用如下指令提交：

```
$ sbatch cp2k-2024.1-cpu.slurm
```

ARM 集群上的 cp2k

ARM 集群中已经预装了 CP2K，可在 ARM 节点查看调用。

ARM 集群上 Slurm 脚本 cp2k.slurm

示例脚本如下：

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cp2k/8.2-gcc-9.3.0-openblas-openmpi
module load openmpi/4.0.3-gcc-9.3.0

ulimit -s unlimited
```

(续下页)

(接上页)

```
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun -np ${SLURM_NTASKS} cp2k.psmg ${INPUT_FILE}
```

在 ARM 节点使用如下命令提交作业：

```
sbatch cp2k.slurm
```

参考资料

- CP2K 官网

cplex

cplex 是 IBM 公司开发的一款商业版的优化引擎, 该引擎专门用于求解大规模的线性规划 (LP)、二次规划 (QP)、带约束的二次规划 (QCQP)、二阶锥规划 (SOCP) 等四类基本问题, 以及相应的混合整数规划 (MIP) 问题。

cplex 安装以及使用说明

1. 如下图所示, 登录 [IBM cplex 官网](#) 并注册账号, 然后下载 linux 平台下的 cplex 安装包 (试用免费版):



Box@IBM	
preview	
名称	已更新 ↓
 cos_installer_preview-22.1.1.0.R0-M08SVML-win-x86-64.exe	2022年12月9日 由 RORY KEELEY
 cos_installer_preview-22.1.1.0.R0-M08SXML-osx.zip	2022年12月9日 由 RORY KEELEY
 cos_installer_preview-22.1.1.0.R0-M08WYML-osx-arm64.zip	2022年12月9日 由 RORY KEELEY
 cos_installer_preview-22.1.1.0.R0-M08SWML-linux-x86-64.bin	2022年12月9日 由 RORY KEELEY

2. 申请计算资源:

```

srun -p 64c512g -n 4 --pty /bin/bash (思源一号)
或者
srun -p cpu -N 1 --ntasks-per-node 40 --exclusive --pty /bin/bash (pi2.0)

```

3. 加载所需模块:

```
module load gcc openmpi jdk python
```

4. 在自己的家目录下新建一个目录 **cplex** 作为安装目录, 进入该目录, 并将上一步得到的安装包上传到当前目录:

```

mkdir cplex
cd cplex

```

5. 执行以下命令使安装文件具有执行权限, 然后执行该文件:

```

chmod 777 cos_installer_preview-22.1.1.0.R0-M08SWML-linux-x86-64.bin
./cos_installer_preview-22.1.1.0.R0-M08SWML-linux-x86-64.bin

```

6. 根据终端输出的安装提示一步一步往下执行即可。

小心

一定要将默认的安装路径 `/opt/ibm/ILOG/CPLEX_Studio_Community2211` 改为自己家目录下的某个目录, 并且使用绝对路径 (比如说 `/dssg/home/acct-hpc/hpcpzz/cplex`);

7. **cplex** 自带了很多标准算例, 用户可以通过执行以下命令来编译并运行这些算例:

```

export PATH=$PATH:~/cplex/cplex/bin/x86-64_linux:~/cplex/cpooptimizer/bin/
↪x86-64_linux
cd ~/cplex/cplex/examples/x86-64_linux/static_pic
make execute -j

```

8. 此时在终端中可得到如下结果:

```

make all_c
make[1]: Entering directory '/dssg/home/acct-hpc/hpcpzz/cplex214/cplex/
↳examples/x86-64_linux/static_pic'
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex1.c -o_
↳lpex1.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex2.c -o_
↳lpex2.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex3.c -o_
↳lpex3.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex4.c -o_
↳lpex4.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex5.c -o_
↳lpex5.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex6.c -o_
↳lpex6.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex7.c -o_
↳lpex7.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/lpex8.c -o_
↳lpex8.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/mipex1.c -o_
↳mipex1.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/mipex2.c -o_
↳mipex2.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/mipex3.c -o_
↳mipex3.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/mipex4.c -o_
↳mipex4.o
gcc -c -m64 -fPIC -I../../../../include ../../../../examples/src/c/miqpex1.c -o_
↳miqpex1.o
...
./admipex7 ../../../../examples/data/mexample.mps
Selected objective sense: MINIMIZE
Selected objective name: obj
Selected RHS name: rhs
Selected bound name: bnd
Version identifier: 22.1.1.0 | 2022-11-26 | 9160aff4d
CPXPARAM_MIP_Strategy_Search 1
Legacy callback S
Found incumbent of value -46.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 2 times.
Aggregator did 1 substitutions.
Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.00 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: traditional branch-and-cut.

```

(续下页)

(接上页)

```
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.00 sec. (0.02 ticks)

Nodes
Node Left Objective IInf Best Integer Cuts/
Variable B NodeID Parent Depth Best Bound ItCnt
->Gap

* 0+ 0 -46.0000 -163.0000 254.
->35%
* 0+ 0 -111.0000 -163.0000 46.
->85%
* 0+ 0 -122.5000 -163.0000 33.
->06%
0 0 -125.2083 1 -122.5000 -125.2083 11 2.
->21%
0 0 cutoff -122.5000 11 0.
->00%
0 0
Elapsed time = 0.01 sec. (0.09 ticks, tree = 0.01 MB, solutions = 3)

Root node processing (before b&c):
Real time = 0.01 sec. (0.09 ticks)
Sequential b&c:
Real time = 0.00 sec. (0.00 ticks)
-----
Total (root+branch&cut) = 0.01 sec. (0.09 ticks)
Solution status 101.
Objective value -122.5
...
```

参考资料

- [IBM cplex 官网](#)

Gerris

简介

Gerris 是求解描述流体流动的偏微分方程的开源软件。

Gerris 基本使用

1. 创建一个目录，并在该目录下编写如下 vorticity.gfs 文件：

```
1 2 GfsSimulation GfsBox GfsGEEdge {} {
GfsTime { end = 50 }
GfsRefine 6
GfsInit {} {
  U = (0.5 - rand() / (double) RAND_MAX)
```

(续下页)

(接上页)

```

    V = (0.5 - rand()) / (double) RAND_MAX)
}
GfsOutputTime          { istep = 10 } stdout
GfsOutputProjectionStats { istep = 10 } stdout
}
GfsBox {}
1 1 right
1 1 top

```

2. 在该目录下编写如下 `gerristest.slurm` 运行脚本:

```

#!/bin/bash

#SBATCH -J test
#SBATCH -p small
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --cpus-per-task=1

IMAGE_PATH=/lustre/share/img/x86/gerris/gerris.sif
singularity exec $IMAGE_PATH gerris2D vorticity.gfs

```

3. 提交脚本:

```

sbatch gerristest.slurm

```

4. 运行结束后可在 `.out` 文件中得到如下结果 (部分):

```

step:      0 t:      0.00000000 dt:  1.888931e-02 cpu:      0.01000000
↪real:      0.01103200
Approximate projection
niter:      3
residual.bias:  -4.374e-18  2.233e-18
residual.first:  3.751e-01  4.525e-05  20
residual.second:  4.633e-01  6.034e-05  20
residual.infty:  1.419e+00  4.128e-04  15
step:     10 t:      0.30436678 dt:  3.947231e-02 cpu:      0.18000000
↪real:      0.20647300
MAC projection      before      after      rate
niter:      2
residual.bias:    1.746e-19  9.098e-20
residual.first:   2.313e-02  9.074e-05  16
residual.second:   3.342e-02  1.171e-04  17
residual.infty:   2.310e-01  5.609e-04  20
Approximate projection
niter:      2
residual.bias:   -5.667e-19  2.551e-19
residual.first:   1.873e-02  5.536e-05  18

```

(续下页)

(接上页)

residual.second:	2.423e-02	7.055e-05	19
residual.infty:	1.063e-01	3.206e-04	18

参考资料

- [Gerris 官网](#)

gnuplot

简介

Gnuplot 是一款小巧而强大的开源科研绘图工具。绘图质量高且快，易学易用，仅需少量代码，就可得到用于发表的高质量图片，中英文参考文档丰富。**Gnuplot** 可做简单的数据处理和分析，比如统计和多参数函数拟合。

π 集群上的 **Gnuplot**

Gnuplot 需要在 HPC Studio 可视化平台的“远程桌面”里使用。 π 集群登录节点不支持 **Gnuplot** 图形显示。

使用方法

先在 HPC Studio 里启动远程桌面，再调用 **Gnuplot**。

1. 在 HPC Studio 上启动远程桌面

- 1) 浏览器打开 <https://studio.hpc.sjtu.edu.cn>
- 2) 顶栏 Interactive Apps 下拉菜单，选择第一个 Desktop
- 3) Desktop 里第一个 Desktop Instance Size，选择最基本的 1core，然后点击 Launch
- 4) 等待几秒，甚或更长时间，取决于 small 队列可用资源量。Studio 的远程桌面以一个正常的 small 队列作业启动
- 5) 启动后，右上角会显示 1 node 1 core Running. 然后点击 Launch Desktop
6. 在远程桌面里调用 **Gnuplot**

在远程桌面空白处，右键单击，Open Terminal Here 打开终端，输入 gnuplot 命令：

```
$ gnuplot
gnuplot> p x      (绘制 y = x 函数)
```

另一个稍复杂的作图示例：

```
$ gnuplot
gnuplot> set title "Iteration within plot command"
```

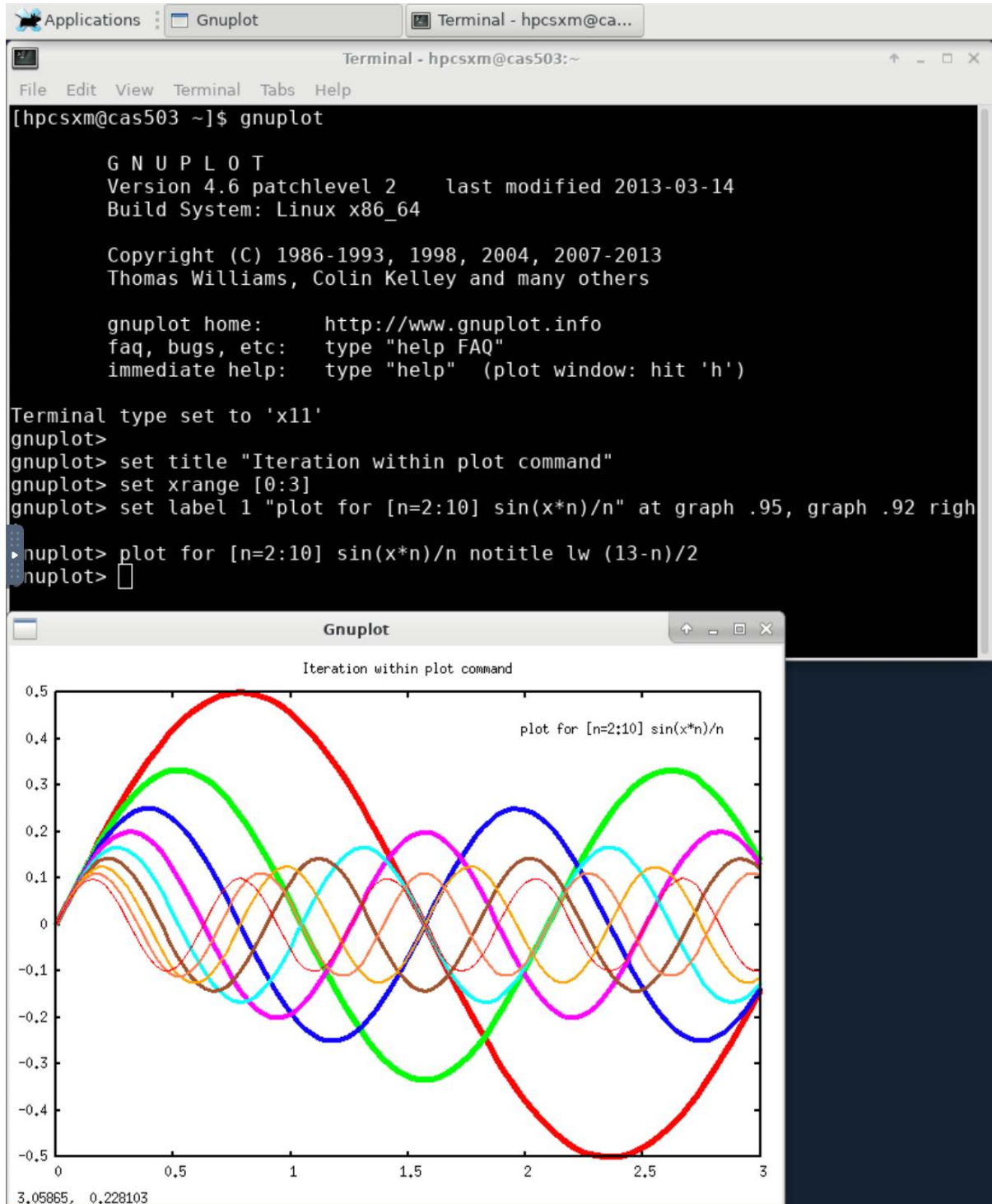
(续下页)

(接上页)

```

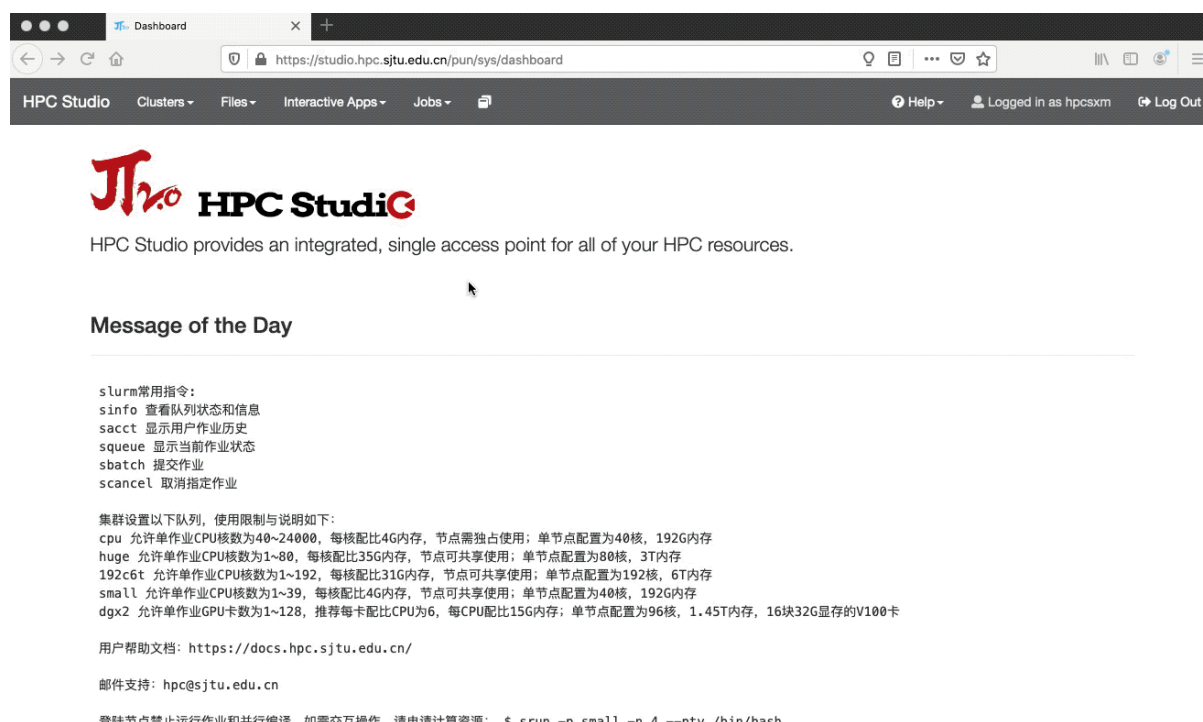
set xrange [0:3]
set label 1 "plot for [n=2:10] sin(x*n)/n" at graph .95, graph .92
↪right
plot for [n=2:10] sin(x*n)/n notitle lw (13-n)/2

```



使用 `q` 退出 `gnuplot`。

绘图操作见动图：



注意：远程桌面使用完毕后需退出，否则会持续计费。两种退出方法：

1. 在 Studio 界面上点 Delete 删除该作业。
2. 在集群终端里输入 squeue 命令查看作业，并用 scancel 终止该作业。

参考资料

- gnuplot <http://www.gnuplot.info/>

GAUSSIAN

简介

GAUSSIAN 是一个量子化学软件包，它是目前应用最广泛的计算化学软件之一，其代码最初由理论化学家、1998 年诺贝尔化学奖得主约翰·波普爵士编写，其名称来自于波普在软件中所使用的高斯型基组。使用高斯型基组是波普为简化计算过程缩短计算时间所引入的一项重要近似。

超算平台不提供 GAUSSIAN 程序，也不解答与 GAUSSIAN 授权相关的问题。请用户前往官方网站 <https://gaussian.com> 咨询软件授权事宜。由于使用非法授权软件产生的后果，由用户承担。

作业脚本示例（假设作业脚本名为 *test.slurm*）：

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH -n 8 # 核数 8
#SBATCH --ntasks-per-node=8 # 每节点核数
```

(续下页)

(接上页)

```
#SBATCH --output=test.out
#SBATCH --error=%j.err

g16 test.gjf
```

使用如下指令提交：

```
$ sbatch test.slurm
```

GROMACS

简介

GROMACS 是一种分子动力学应用程序，可以模拟具有数百至数百万个粒子的系统的牛顿运动方程。GROMACS 旨在模拟具有许多复杂键合相互作用的生化分子，例如蛋白质，脂质和核酸。

可用的版本

思源一号

版本	平台	构建方式	模块名
2022.2	gpu	spack	gromacs/2022.2-intel-2021.4.0-cuda
2022.3	gpu	spack	gromacs/2022.3-intel-2021.4.0-cuda
2021.3	cpu	spack	gromacs/2021.3-intel-2021.4.0
2021.3	cpu	spack	gromacs/2021.3-gcc-11.2.0-cuda-openblas-openmpi
2021.2	gpu	spack	gromacs/2021.2-intel-2021.4.0-cuda
2021.4	gpu	spack	gromacs/2021.4-gcc-11.2.0-cuda-openblas
2024.1	cpu	spack	gromacs/2024.1-gcc-12.3.0-openblas-openmpi

Pi 2.0

版本	平台	构建方式	模块名
2018.8		spack	gromacs/2018.8-intel-2021.4.0-openblas
2022.5		spack	gromacs/2022.5-gcc-8.5.0
2022.5		spack	gromacs/2022.5-gcc-11.2.0-cuda
2022.5		spack	gromacs/2022.5-gcc-11.2.0-openblas
2023.1		spack	gromacs/2023.1-gcc-11.2.0-cuda
2024.1		spack	gromacs/2024.1-gcc-12.3.0
2024.1		spack	gromacs/2024.1-gcc-12.3.0-cuda

ARM 集群

版本	平台	构建方式	模块名
2023.2		源码	gromacs/2023.2-gcc-10.3.1

算例下载

```
mkdir ~/gromacs && cd ~/gromacs
wget -c https://ftp.gromacs.org/pub/benchmarks/water_GMX50_bare.tar.gz
tar xf water_GMX50_bare.tar.gz
cd water-cut1.0_GMX50_bare/0768/
```

数据目录如下所示:

```
[hpc@login2 water-cut1.0_GMX50_bare]$ tree 0768/
0768/
├── conf.gro
├── pme.mdp
├── rf.mdp
└── topol.top

0 directories, 4 files
```

集群上的 GROMACS

- 一. 思源一号 *GROMACS*
- 二. $\pi 2.0$ *GROMACS*

- 三. ARM GROMACS

一. 思源一号 GROMACS

1. INTEL 编译的版本

预处理数据-CPU 版本

```
#!/bin/bash

#SBATCH --job-name=64_gromacs
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load gromacs/2021.3-intel-2021.4.0
gmx_mpi grompp -f pme.mdp
```

预处理数据-GPU 版本

```
#!/bin/bash

#SBATCH --job-name=gpu_gromacs
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=16
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load gromacs/2022.2-intel-2021.4.0-cuda
module load cuda/11.5.0
gmx_mpi grompp -f pme.mdp
```

提交作业脚本-CPU 版本

```
#!/bin/bash

#SBATCH --job-name=64_gromacs
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
```

(续下页)

(接上页)

```
module load gromacs/2021.3-intel-2021.4.0
mpirun gmx_mpi mdrun -dlb yes -v -nsteps 10000 -resethway -noconfout -pin -
→on -ntomp 1 -s topol.tpr
```

提交作业脚本-GPU 版本

```
#!/bin/bash

#SBATCH --job-name=gpu_gromacs
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load gromacs/2022.2-intel-2021.4.0-cuda
module load cuda/11.5.0
mpirun -n 1 gmx_mpi mdrun -dlb yes -v -nsteps 10000 -resethway -noconfout -
→pin on -ntomp 16 -gpu_id 0 -s topol.tpr
```

2.GCC 编译的版本

预处理数据

```
#!/bin/bash

#SBATCH --job-name=64_gromacs
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/11.2.0
module load openmpi/4.1.1-gcc-11.2.0
module load gromacs/2021.3-gcc-11.2.0-cuda-openblas-openmpi
gmx_mpi grompp -f pme.mdp
```

提交预处理作业脚本。

```
$ sbatch pre.slurm
```

运行结果如下所示：

```
[hpchg@login water]$ tree 0768
0768
```

(续下页)

(接上页)

```

|— 9854405.err
|— 9854405.out
|— conf.gro
|— mdout.mdp
|— pme.mdp
|— pre.slurm
|— rf.mdp
|— topol.top
|— topol.tpr

```

提交作业脚本

```

#!/bin/bash

#SBATCH --job-name=64_gromacs
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/11.2.0
module load openmpi/4.1.1-gcc-11.2.0
module load gromacs/2021.3-gcc-11.2.0-cuda-openblas-openmpi
mpirun gmx_mpi mdrun -dlb yes -v -ntsteps 10000 -resethway -noconfout -pin
↪on -ntomp 1 -s topol.tpr

```

提交上述作业

```

sbatch gromacs.slurm

```

运行结果如下所示:

```

[hpchgc@sylogin1 64cores]$ tail -n 20 9853399.err
vol 0.94 imb F 2% pme/F 0.92 step 10000, remaining wall clock time: 0.
↪s

Dynamic load balancing report:
DLB was permanently on during the run per user request.
Average load imbalance: 2.0%.
The balanceable part of the MD step is 85%, load imbalance is computed
↪from this.
Part of the total run time spent waiting due to load imbalance: 1.7%.
Steps where the load balancing was limited by -rdd, -rcon and/or -dds: X 0
↪% Y 0 %
Average PME mesh/force load: 0.923
Part of the total run time spent waiting due to PP/PME imbalance: 2.4 %

```

(续下页)

(接上页)

```

          Core t (s)    Wall t (s)          (%)
Time:      3052.051      47.699      6398.5
          (ns/day)      (hour/ns)
Performance:    18.117      1.325

```

GROMACS reminds you: "The Stingrays Must Be Fat This Year" (Red Hot Chili
 ↳Peppers)

二. π 2.0 GROMACS

CPU 版本: gromacs/2024.1-gcc-12.3.0

提交运行作业脚本

```

#!/bin/bash

#SBATCH -J gromacs_cpu_test
#SBATCH -p cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load gromacs/2024.1-gcc-12.3.0

ulimit -s unlimited
ulimit -l unlimited

gmx_mpi grompp -f pme.mdp
mpirun gmx_mpi mdrun -dlb yes -v -nsteps 10000 -resethway -noconfout -pin
↳on -ntomp 1 -s topol.tpr

```

GPU 版本: gromacs/2024.1-gcc-12.3.0-cuda

作业脚本

```

#!/bin/bash
#SBATCH -J gromacs
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1
#SBATCH -p dgx2

module purge
module load gromacs/2024.1-gcc-12.3.0-cuda

```

(续下页)

(接上页)

```
gmx_mpi grompp -f pme.mdp
mpirun -np 1 gmx_mpi mdrun -dlb yes -v -nsteps 10000 -reseedhway -noconfout -
↪pin on -ntomp 6 -gpu_id 0 -s topol.tpr
```

三. ARM GROMACS

gromacs/2023.2-gcc-10.3.1

提交运行作业脚本

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load gromacs/2023.2-gcc-10.3.1

gmx_mpi grompp -f pme.mdp
export OMP_NUM_THREADS=1

mpirun gmx_mpi mdrun -dlb yes -v -nsteps 10000 -reseedhway -noconfout -pin↪
↪on -ntomp 1 -s topol.tpr
```

运行结果如下所示 (单位: ns/day, 越高越好)

1.GROMACS 思源一号

gromacs/2021.3-intel-2021.4.0			
核数	64	128	192
Performance	17.724	35.250	53.321

gromacs/2021.3-gcc-11.2.0-cuda-openblas-openmpi			
核数	64	128	192
Performance	10.6259	32.798	55.635

gromacs/2022.3-intel-2021.4.0-cuda	
卡数	1 块 A100
Performance	37.081

2.GROMACS π2.0

gromacs/2024.1-gcc-12.3.0			
核数	40	80	120
Performance	8.876	19.022	28.220

gromacs/2024.1-gcc-12.3.0-cuda	
卡数	1 块 V100
Performance	29.835

3.GROMACS ARM

gromacs/2023.2-gcc-10.3.1			
核数	128	256	512
Performance	7.982	17.289	33.495

参考资料

- gromacs 官方网站 <http://www.gromacs.org/>

GPUMD

简介

GPUMD 是一款在 GPU 上实现的通用分子动力学软件包。GPUMD 可以实现对金属、半导体、有机物等各类材料的高效率和高性价比的分子动力学模拟。GPUMD 提供极为高效 NEP 机器学习势函数，实现 NEP 训练与应用的无缝衔接，将量子力学精度的模拟扩展至介观尺度。

从源码编译 GPUMD 方法

```
git clone https://github.com/brucefan1983/GPUMD.git #这一步可以使用 github
↪ 的镜像站加速下载
cd src/
module load cuda/11.8.0
make
```

编译成功后，将在 `src` 文件夹下生成两个可执行文件 `gpumd` 和 `nep`

运行 **GPUMD** 方法

GPUMD 已作为环境模块安装在 π 和思源一号集群上。要运行模拟，请遵循以下步骤。

1. 加载模块

```
module load gpumd
```

2. 进入数据目录

```
cd /path/to/your/data_directory
```

3. 执行模拟

注意

严禁在登录节点上直接运行 `gpumd`。你必须通过 **Slurm** 调度系统将作业提交到 GPU 计算分区

在 π 上运行

脚本如下 `test1.slurm`

```
#!/bin/bash

#SBATCH -J gpumd_example
#SBATCH -p dgx2
#SBATCH -N 1
#SBATCH --gres=gpu:1
#SBATCH -o %j.out
#SBATCH -e %j.err

module load gpumd

# 设置资源限制，防止某些模拟意外终止
ulimit -s unlimited
ulimit -l unlimited

# 这里进入你的数据所在文件夹
EXAMPLE_DIR="/~/GPUMD/examples"
cd $EXAMPLE_DIR

# 运行 gpumd 模拟
srun gpumd

# 运行 nep
# srun nep
```

提交作业

```
$ sbatch test1.slurm
```

在思源一号上运行

脚本如下 test2.slurm

```
#!/bin/bash

#SBATCH -J gpumd_example
#SBATCH -p a100
#SBATCH -N 1
#SBATCH --gres=gpu:1
#SBATCH -o %j.out
#SBATCH -e %j.err

# 设置资源限制，防止某些模拟意外终止
ulimit -s unlimited
ulimit -l unlimited

# 这里进入你的数据所在文件夹
EXAMPLE_DIR=~ /GPUMD/example"
cd $EXAMPLE_DIR

# 运行 gpumd 模拟
srun ~ /GPUMD/src/gpumd

# 运行 nep
srun ~ /GPUMD/src/nep
```

提交作业

```
$ sbatch test2.slurm
```

参考资料

- [GPUMD 官网](#)
- [GPUMD 示例](#)

LAMMPS

简介

LAMMPS 是大规模原子分子并行计算代码，在原子、分子及介观体系计算中均有重要应用，并行效率高，广泛应用于材料、物理、化学等模拟。

专题培训

- “交我算” lammmps 软件编译与使用

可用的版本

集群	平台	版本	模块名
思源一号		20230328	lammmps/20230328-intel-2021.4.0-omp
思源一号		20220324	lammmps/20220324-intel-2021.4.0-omp
思源一号		20210310	lammmps/20210310-intel-2021.4.0-omp
思源一号		20230802	lammmps/20230802-intel-2021.4.0-gpu
pi 2.0		20230328	lammmps/20230328-intel-2021.4.0-omp
pi 2.0		20220324	lammmps/20220324-oneapi-2021.4.0
kunpeng		20190605	lammmps/bisheng-1.3.3-lammmps-2019

集群上的 LAMMPS

- 思源一号 LAMMPS
- $\pi 2.0$ LAMMPS
- ARM LAMMPS

一. 思源一号 LAMMPS

1. 全局部署版本

本版本支持 intel 加速。对于大部分势函数 (eam, lj 等), 均推荐使用 intel 加速, 计算速度可提升数倍。具体测评和支持范围请见官方文档: [LAMMPS INTEL package](#)

使用 intel 加速的 slurm 脚本示例:

```
#!/bin/bash
#SBATCH --job-name=lmp_test
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load lammmps/20230328-intel-2021.4.0-omp

mpirun lmp -pk intel 0 omp 2 -sf intel -i in.lj
```

注意：若体系不支持 intel package，请使用如下 slurm 脚本：

```
#!/bin/bash
#SBATCH --job-name=lmp_test
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load lammps/20230328-intel-2021.4.0-omp

mpirun lmp -i in.lj
```

2. 自行编译 LAMMPS

LAMMPS 自行编译十分容易。下面以在思源一号上为例介绍 LAMMPS 安装

a) 申请计算节点资源用来编译 LAMMPS，并注意在全部编译结束后退出：

```
srtn -p 64c512g -n 4 --pty /bin/bash
```

b) 从官网获得最新的 LAMMPS，推荐下载最新的版本

```
wget https://lammps.sandia.gov/tars/lammps-stable.tar.gz
```

c) 加载 Intel oneapi 模块：

```
module load oneapi/2021.4.0
```

d) 编译 (以额外安装 MANYBODY, MEAM, RIGID 和 Intel 加速包为例)

```
$ tar xvf lammps-stable.tar.gz
$ cd lammps-XXXXXX
$ cd src
$ make #查看编译选项
$ make package #查看可用的包
$ make yes-intel yes-manybody yes-meam yes-rigid #添加所需的包
$ make ps #查看计划安装的包列表
$ make -j 4 oneapi #开始编译
```

e) 环境设置

编译成功后，src 文件夹下将生成可执行文件 lmp_oneapi

为了便于后续调用，一个简单的方法是将该文件移至 ~/bin 文件夹：

```
$ mkdir ~/bin
$ cp lmp_oneapi ~/bin
```

至此安装和设置完成。如下是计算时所需的 slurm 脚本：


```
#!/bin/bash

#SBATCH --job-name=lmp
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load oneapi/2021.4.0
export PATH=~/.bin:$PATH

mpirun lmp_oneapi -pk intel 0 omp 2 -sf intel -i in.lj
# 若势函数等体系不支持intel加速，则使用下方语句：
# mpirun lmp_oneapi -i in.lj
```

二. π 2.0 LAMMPS

1. Intel 编译器部署的版本

```
#!/bin/bash

#SBATCH --job-name=lammps_pi
#SBATCH --partition=cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load lammps/20230328-intel-2021.4.0-omp

mpirun lmp -pk intel 0 omp 2 -sf intel -i in.lj
```

2. CPU 版本自行编译

若对 lammps 版本有要求，或需要特定的 package，可自行编译 Intel 版本的 Lammps。下面以在 π 集群为例介绍 lammps 的自行安装

a) 从官网下载 lammps，推荐安装最新的稳定版：

```
$ wget https://lammps.sandia.gov/tars/lammps-stable.tar.gz
$ or
$ cp /lustre/share/samples/lammps/lammps-stable.tar.gz ./
```

- b) 由于登录节点禁止运行作业和并行编译，请申请计算节点资源用来编译 `lammps`，并在编译结束后退出：

```
$ srun -p cpu -n 8 --pty /bin/bash
```

- c) 加载 Intel oneapi 模块：

```
module load oneapi/2021
```

- d) 编译 (以额外安装 MANYBODY 和 Intel 加速包为例)

```
$ tar xvf lammps-stable.tar.gz
$ cd lammps-XXXXXX
$ cd src
$ make #查看编译选项
$ make package #查看包
$ make yes-intel #"make yes-"后面接需要安装的 package 名字
$ make yes-manybody
$ make ps #查看计划安装的包列表
$ make -j 8 oneapi #开始编译
```

- e) 测试脚本

编译成功后，将在 `src` 文件夹下生成 `lmp_oneapi` 后续调用，请给该文件的路径，比如 `~/lammps-3Mar20/src/lmp_oneapi`。脚本名称可设为 `slurm.test`

```
#!/bin/bash

#SBATCH -J lammps
#SBATCH -p cpu
#SBATCH -n 40
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

ulimit -s unlimited
ulimit -l unlimited

module load oneapi/2021

srun --mpi=pmi2 ~/lammps-3Mar20/src/lmp_oneapi -i in.lj
```

三. ARM LAMMPS

1. ARM 版 lammps(bisheng 编译器 +hypermpi)

脚本如下 (lammps.slurm):

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=lammps
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=96
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load lammps/bisheng-1.3.3-lammps-2019
mpirun -x OMP_NUM_THREADS=1 lmp_aarch64_arm_hypermpi -in in.lj

$ sbatch lammps.slurm
```

运行结果 (单位为：秒，越低越好)

思源一号

lammps/20230328-intel-2021.4.0-omp			
核数	64	128	192
Wall time	0:01:57	0:01:01	0:00:46

π2.0

lammps/20230328-intel-2021.4.0-omp			
核数	40	80	120
Wall time	0:03:16	0:01:35	0:01:06

ARM

lammps/bisheng-1.3.3-lammps-2019		
核数	64	96
Wall time	0:07:26	0:04:43

算例内容如下： *in.lj*

```
# 3d Lennard-Jones melt

variable      x index 4
variable      y index 4
variable      z index 4
```

(续下页)

(接上页)

```

variable      xx equal 20*$x
variable      yy equal 20*$y
variable      zz equal 20*$z

units          lj
atom_style     atomic

lattice        fcc 0.8442
region         box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box     1 box
create_atoms   1 box
mass           1 1.0

velocity       all create 1.44 87287 loop geom

pair_style     lj/cut 2.5
pair_coeff     1 1 1.0 1.0 2.5

neighbor       0.3 bin
neigh_modify   delay 0 every 20 check no

fix            1 all nve

run            10000

```

参考资料

- [LAMMPS 官网](#)
- [NVIDIA GPU CLOUD](#)

LAMMPS-RBE

简介

LAMMPS-RBE 是由上海交通大学上海应用数学中心团队基于 LAMMPS 二次开发的自研软件。该版本在长程力模拟中引入了先进的 **Random Batch Ewald** 算法, RBE 使用基于动理学或连续介质理论的路径, 研究复杂环境中微纳系统的多体效应, 并结合分子动力学进行多尺度建模和数学分析。LAMMPS-RBE 突破了传统分子动力学在 CPU 集群上可扩展性差的问题, 可以使百万级别粒子以上的大尺度体系的计算成本降低一个数量级。

详细算法解释可以参阅: <https://math.sjtu.edu.cn/faculty/xuzl/RBE.pdf>

新增功能细节描述在本文底部。

可用的版本

版本	平台	构建方式	模块名
7Aug2019		源码	lammps-rbe/20190807-oneapi-2021.4 思源一号
7Aug2019		源码	lammps-rbe/ 20190807-intel-parallel-studio-2020.1-impi π 2.0
7Aug2019		容器	lammps-rbe/20190807-oneapi-2021.1-impi π 2.0

算例获取

 π 2.0 上数据获取

```
mkdir ~/lammps-rbe && cd ~/lammps-rbe
cp -r /lustre/share/benchmarks/lammps-rbe/lammps-rbe.tar.gz .
tar xf lammps-rbe.tar.gz
cd RBE_Example/
```

思源一号上数据获取

```
mkdir ~/lammps-rbe && cd ~/lammps-rbe
cp -r /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/lammps-rbe-
→2019.8.4/Lammps-RBE/RBE_Example ./
cd RBE_Example/
```

数据的目录结构如下所示：

```
[hpc@login2 Lammps-RBE]$ tree RBE_Example/
RBE_Example/
├── in.spce-bulk-nvt
└── lmp.data
```

注意：本文算例步数设置为 40000，即文件 in.spce-bulk-nvt 最后一行内容为：
run 40000

运行核数要和文件 in.spce-bulk-nvt 中的参数 kspace_style rbe 0.07 200 80 最后一个数字保持一致，比如本例中为：80

不同集群上的 LAMMPS-RBE

- 思源一号 *LAMMPS-RBE*
- π 2.0 *LAMMPS-RBE*

思源一号

作业脚本如下所示:

```
#!/bin/bash

#SBATCH -J lammps-rbe
#SBATCH -p 64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH -o RBE.out
#SBATCH -e %j.err

module load lammps-rbe/20190807-oneapi-2021.4
mpirun lmp_intel_cpu_intelmpi -i in.spce-bulk-nvt
```

提交上述脚本

```
sbatch lammps-rbe.slurm
```

运行结果如下所示:

```
[hpc@node738 bte]$ tail -n 1 RBE_BAOBAO.log
Total wall time: 0:02:01
```

π2.0

源码编译版本

lammps-rbe/20190807-intel-parallel-studio-2020.1-impi

作业脚本如下所示:

```
#!/bin/bash

#SBATCH -J lammps
#SBATCH -p cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH -o RBE.out
#SBATCH -e %j.err

module load lammps-rbe/20190807-intel-parallel-studio-2020.1-impi
mpirun lmp_intel_cpu_intelmpi -i in.spce-bulk-nvt
```

提交作业:

```
$ sbatch lammps-rbe.slurm
```

运行结果如下:

```
[hpc@login2 80cores_intel]$ tail -n 1 RBE_BAOBAO.log
Total wall time: 0:03:28
```

容器版本

lammps-rbe/20190807-oneapi-2021.1-mpi π 2.0

运行脚本如下:

```
#!/bin/bash

#SBATCH -J lammps
#SBATCH -p cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH -o RBE.out
#SBATCH -e %j.err

module load lammps-rbe/20190807-oneapi-2021.1-mpi
mpirun lammps-rbe -i in.spce-bulk-nvt
```

提交上述作业

```
sbatch lammps-rbe.slurm
```

运行结果如下所示:

```
[hpc@login2 80cores]$ tail -n 1 RBE_BAOBAO.log
Total wall time: 0:04:26
```

运行结果

思源一号上的结果

lammps-rbe/20190807-oneapi-2021.4			
核数	64	128	256
wall time	0:03:10	0:02:02	0:01:26

π 2.0 上的结果

lammps-rbe/20190807-intel-parallel-studio-2020.1-mpi			
核数	40	80	160
wall time	0:06:09	0:03:28	0:02:09

lammps-rbe/20190807-oneapi-2021.1-impi			
核数	40	80	160
wall time	0:06:17	0:04:26	0:03:48

新增功能

同 Llammps 已有功能相比，该版本新增三个功能：

1. 基于 Random Batch Ewald (RBE) 算法的三维周期/二维准周期平板系统静电求解器，特别适用于多核模拟。调用方式：在 Llammps 的 input 文件中加入下面命令（需和 pair/lj/cut/coul/long 配合使用，这点和 PPPM 算法相同），

```
kspace_style rbe arg1 arg2 arg3
```

其中 kspace_style 是 Llammps 固定指令，表示模拟中要计算静电相互作用；rbe 是算法名称表示调用 RBE 算法计算静电；arg1=alpha，是 RBE 算法里用于控制近远场比例的参数，该参数的选择和 Ewald 以及 PPPM 算法相同。如果希望相对误差是 $1e-4$ ，那么需选取使得 $\text{erfc}(r_{\text{cut}} \cdot \sqrt{\alpha}) \approx 1e-4$ 的 alpha，其中 r_{cut} 是在 pair/lj/cut/coul/long 中选取的静电近场截断；arg2=batch_size，是在傅里叶空间中做随机采样得到的样本数量，一般为几十至数百（越大越准确，越小计算速度越快）；arg3=sampling_core，用于采样的 CPU 核的数量，需 >1 且 $<$ 总 MPI 数量，一般可选取和用户使用的 MPI 数量相同或 MPI 数量一半。两个使用案例（假设使用 200 个 CPU 核）：

```
pair_style lj/cut/coul/long 10.0 10.0
```

```
kspace_style rbe 0.07 500 100
```

或调用 intel 的近场计算

```
pair_style lj/cut/coul/long/intel 12.0 12.0
```

```
kspace_style rbe 0.05 200 100
```

如果希望处理二维周期且 z 方向是两块平板的系统，需要在 input 文件中定义平板的位置参数和 kspace_modify slab 3，方法同 LAMMPS 官方文档中用 PPPM 算平板问题的方式一致。

2. 基于 RBE2D 算法的二维周期，Z 方向具介电不匹配界面（Dielectric Interfaces）系统的静电求解器（包括界面带连续面电荷情形），特别适用于多核模拟，并且速度大幅超过其他处理 Dielectric Interfaces 的静电算法。调用方式：在 Llammps 的 input 文件中加入下面命令

```
pair_style lj/cut/coul/long/rbed arg1 arg2 arg3 arg4
```

```
kspace_style Rbed arg1 arg2 arg3 arg4 arg5 arg6 arg7
```

pair_style 和 lj/cut/coul/long/rbed 分别是 Llammps 固定指令（表示计算静电近场）和算法名称（表示使用 RBE2D 算法）；arg1=LJ_cut，是 LJ 相互作用的截断半径；arg2=Coul_cut，是静电相互作用的截断半径（需小于等于 LJ 截断半径，这点和 LAMMPS 原始设置相同）；arg3=gamma_top，arg4=gamma_down 分别是上下界面的描绘介电不匹配程度的系数，取值范围都是 $[-1, +1]$ ，定义分别为 $(\epsilon_{\text{in}} - \epsilon_{\text{top}}) / (\epsilon_{\text{in}} + \epsilon_{\text{top}})$ 和 $(\epsilon_{\text{in}} - \epsilon_{\text{down}}) /$

($\epsilon_{in} + \epsilon_{down}$), 其中 ϵ_{in} , ϵ_{top} 和 ϵ_{down} 分别是盒子中间、盒子上方、盒子下方介质的相对介电常数。

`kpace_style` 和 `Rbed` 分别是 LAMMPS 固定指令（表示计算静电远场）和算法名称（表示使用 RBE2D 算法）；`arg1=alpha`, `arg2=batch_size`, `arg3=sampling_core` 同 (1) 中 `rbe` 指令对它们的定义相同；`arg4=gamma_top`, `arg5=gamma_down` 和 `lj/cut/coul/long/rbed` 中对它们的定义相同；`arg6=sigma_top`, `arg7=sigma_down` 分别代表上下表面的面电荷密度，单位是 $e/(\text{长度单位的平方})$ 。

一个使用案例（假设使用 200 个 CPU 核）：

```
pair_style lj/cut/coul/long/rbed 10 10 0.939 -0.939
```

```
kpace_style Rbed 0.079647 200 100 0.939 -0.939 0.08 -0.08
```

表示使用 RBE2D 计算一个上下界面介电系数分别为 0.939 和 -0.939、上下界面分别带密度为 0.08 和 -0.08 的连续面电荷的系统的静电相互作用。LJ 截断半径和静电截断半径均为 10，`alpha` 选择 0.079647，每次在傅里叶空间抽取 200 个样本，使用其中 100CPU 核进行采样。

3. 基于 Langevin 动力学提出的新 NPT 系综控温控压器，好处是系统收敛到平衡的速度比 LAMMPS 自带的“fix npt”更快，目前支持各向同性和各向异性两种控压方式。调用方式：在 LAMMPS 的 input 文件中加入下面命令

```
fix ID group-ID baoab temp arg1 arg2 arg3 keyword arg4 arg5 arg6
```

`fix` 和 `temp` 是固定指令，`baoab` 是控压算法名称，`ID` 是用户为这条 `fix` 指定的名称，`group-ID` 指定了这条 `fix` 能够作用的原子组的名称，`ID` 和 `group-ID` 同 LAMMPS 本身对它们的设置相同，可参考 LAMMPS 官方文档中的 `fix` 指令说明；`arg1=Tstart`, `arg2=Tstop` 分别设定了开始和结束时的外部温度；`arg3=Tdamp` 是控温的阻尼系数，单位和时间单位相同，一般为 5 倍至 100 倍模拟的时间步长；`keyword=iso or aniso` 表示控压是各向同性（三个方向耦合在一起，`iso`）或是各向异性（三个方向分别控压，`aniso`）进行；`arg4=Pstart`, `arg5=Pstop` 分别设定了开始和结束时的外部压强；`arg6=Pdamp` 是控压的阻尼系数，一般为数十至数百倍模拟的时间步长。

一个使用案例（假设模拟时间步长为 1fs）：

```
fix 2 all baoab temp 298 298 5 iso 1.0 1.0 100
```

表示使用 Langevin 动力学对所有原子做各向同性控压，开始和结束的外部温度和外部压强分别为 298K 和 1bar，控温和控压阻尼系数分别为 5fs 和 100fs。该 `fix` 指令的名字被设定为 2。

Nektar++

简介

Nektar++ is a spectral/hp element framework designed to support the construction of efficient high-performance scalable solvers for a wide range of partial differential equations.

Nektar++ 使用说明

1. 求解二维方形区域的对流方程 (pi2.0 上单核串行)

$$\frac{\partial u}{\partial t} + V_x \frac{\partial u}{\partial x} + V_y \frac{\partial u}{\partial y} = 0,$$

$$u(x, y; t = 0) = \sin(\kappa x) \cos(\kappa y),$$

$$u(x_b = [-1, 1], y_b; t) = \text{periodic},$$

$$u(x_b, y_b = [-1, 1]; t) = \sin(\kappa(x - V_x t)) \cos(\kappa(y - V_y t)),$$

其中, $x_b, y_b, V_x = 2, V_y = 3, \kappa = 2\pi$ 。

1. 从 [Nektar 官网](#) 的 GETTING STARTED->Tutorials->Basics->Advection-Diffusion->Introduction->Goals 板块下载所需要的数据文件 basics-advection-diffusion.tar.gz 并解压;
2. 解压之后会得到两个目录 completed 以及 tutorial;
3. 进入 completed 目录会看到如下几个文件:

```
ADR_conditions.xml
ADR_mesh.geo
ADR_mesh.msh
ADR_mesh.xml
ADR_mesh_aligned.fld
ADR_mesh_aligned.xml
```

这几个文件定义了求解本问题所需要的几何信息、网格信息以及初始和边界条件。

4. 在此目录下编写如下 Nektar_run.slurm 脚本:

```
#!/bin/bash

#SBATCH -J Nektar_test
#SBATCH -p small
#SBATCH -n 1
#SBATCH --ntasks-per-node=1
#SBATCH -o %j.out
#SBATCH -e %j.err

module load nektar/5.0.0-intel-19.0.4-impi
```

(续下页)

(接上页)

```
ulimit -s unlimited
ulimit -l unlimited

ADRSolver ADR_mesh_aligned.xml ADR_conditions.xml
```

5. 使用如下指令提交:

```
sbatch Nektar_run.slurm
```

6. 然后即可在.out 文件中看到如下结果:

```
=====
EquationType: UnsteadyAdvection
Session Name: ADR_mesh_aligned
Spatial Dim.: 2
Max SEM Exp. Order: 5
Expansion Dim.: 2
Riemann Solver: Upwind
Advection Type:
Projection Type: Discontinuous Galerkin
Advection: explicit
Diffusion: explicit
Time Step: 0.001
No. of Steps: 1000
Checkpoints (steps): 100
Integration Type: ClassicalRungeKutta4
=====
Initial Conditions:
- Field u: sin(k*x)*cos(k*y)
Writing: "ADR_mesh_aligned_0.chk"
Steps: 100      Time: 0.1      CPU Time: 0.435392s
Writing: "ADR_mesh_aligned_1.chk"
Steps: 200      Time: 0.2      CPU Time: 0.430588s
Writing: "ADR_mesh_aligned_2.chk"
Steps: 300      Time: 0.3      CPU Time: 0.428503s
Writing: "ADR_mesh_aligned_3.chk"
Steps: 400      Time: 0.4      CPU Time: 0.428529s
Writing: "ADR_mesh_aligned_4.chk"
Steps: 500      Time: 0.5      CPU Time: 0.430142s
Writing: "ADR_mesh_aligned_5.chk"
Steps: 600      Time: 0.6      CPU Time: 0.429481s
Writing: "ADR_mesh_aligned_6.chk"
Steps: 700      Time: 0.7      CPU Time: 0.433232s
Writing: "ADR_mesh_aligned_7.chk"
Steps: 800      Time: 0.8      CPU Time: 0.431088s
Writing: "ADR_mesh_aligned_8.chk"
Steps: 900      Time: 0.9      CPU Time: 0.427919s
Writing: "ADR_mesh_aligned_9.chk"
```

(续下页)

(接上页)

```

Steps: 1000      Time: 1      CPU Time: 0.436098s
Writing: "ADR_mesh_aligned_10.chk"
Time-integration : 4.31097s
Writing: "ADR_mesh_aligned.fld"
-----
Total Computation Time = 4s
-----
L 2 error (variable u) : 0.00863475
L inf error (variable u) : 0.0390366

```

2. 可压缩圆柱绕流 (pi2.0 上多核并行)

1. 从 [Nektar 官网](#) 的 GETTING STARTED->Tutorials->Compressible Flow Solver->Subsonic Cylinder->Introduction->Goals 板块下载所需要的数据文件 cfs-CylinderSubsonic_NS.tar.gz 并解压;
2. 解压之后会得到两个目录 completed 以及 tutorial;
3. 在 tutorial 目录下编写以下 Nektar_run.slurm 脚本:

```

#!/bin/bash

#SBATCH -J Nektar_test
#SBATCH -p cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module load nektar/5.0.0-intel-19.0.4-impi
module load openmpi/3.1.5-gcc-9.2.0

ulimit -s unlimited
ulimit -l unlimited

mpirun -np 32 CompressibleFlowSolver CylinderSubsonic_NS.xml

```

4. 使用如下指令提交:

```
sbatch Nektar_run.slurm
```

5. 作业运行完毕后即可在.out 文件中看到如下结果 (部分):

```

=====
EquationType: NavierStokesCFE
Session Name: CylinderSubsonic_NS
Spatial Dim.: 2
Max SEM Exp. Order: 3

```

(续下页)

(接上页)

```

Expansion Dim.: 2
Riemann Solver: HLLC
Advection Type:
Projection Type: Discontinuous Galerkin
Diffusion Type:
    Advection: explicit
    AdvectionType: WeakDG
    Diffusion: explicit
    Time Step: 1e-05
    No. of Steps: 60000
Checkpoints (steps): 400
Integration Type: RungeKutta

```

```

=====
EquationType: NavierStokesCFE
Session Name: CylinderSubsonic_NS
Spatial Dim.: 2
Max SEM Exp. Order: 3
Expansion Dim.: 2
Riemann Solver: HLLC
Advection Type:
Projection Type: Discontinuous Galerkin
Diffusion Type:
=====

```

3. 在思源一号上通过镜像来使用 Nektar++

1. 参考上文内容 (求解对流方程) 从 [Nektar 官网](#) 下载所需要的数据文件并解压，然后进入对应目录，在该目录下编写以下 Nektar_run.slurm 脚本：

```

#!/bin/bash

#SBATCH --job-name=nektar
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load openmpi/4.1.1-gcc-8.5.0

IMAGE_PATH=/dssg/share/imgs/nektar++/nek.sif
mpirun -np 2 singularity exec $IMAGE_PATH ADRSolver ADR_mesh_aligned.xml \
↪ADR_conditions.xml

```

2. 使用如下指令提交：

```
sbatch Nektar_run.slurm
```

3. 作业运行完毕之后即可在.out 文件当中看到如下结果:

```
gs_setup: 30 unique labels shared
pairwise times (avg, min, max): 2.58603e-06 2.51303e-06 2.65902e-06
crystal router                : 2.64363e-06 2.61022e-06 2.67704e-06
all reduce                    : 2.46682e-06 2.36742e-06 2.56621e-06
used all_to_all method: allreduce
handle bytes (avg, min, max): 1588 1588 1588
buffer bytes (avg, min, max): 480 480 480
=====
EquationType: UnsteadyAdvection
Session Name: ADR_mesh_aligned
Spatial Dim.: 2
Max SEM Exp. Order: 5
Num. Processes: 2
Expansion Dim.: 2
Riemann Solver: Upwind
Advection Type: Weak DG
Projection Type: Discontinuous Galerkin
Advect. advancement: explicit
Advect. advancement: explicit
Time Step: 0.001
No. of Steps: 1000
Checkpoints (steps): 100
Integration Type: RungeKutta
=====
Initial Conditions:
- Field u: sin(k*x)*cos(k*y)
Writing: "ADR_mesh_aligned_0.chk" (0.000970838s, XML)
Steps: 100      Time: 0.1      CPU Time: 0.0535671s
Writing: "ADR_mesh_aligned_1.chk" (0.000835073s, XML)
Steps: 200      Time: 0.2      CPU Time: 0.0540179s
Writing: "ADR_mesh_aligned_2.chk" (0.000772003s, XML)
Steps: 300      Time: 0.3      CPU Time: 0.0542538s
Writing: "ADR_mesh_aligned_3.chk" (0.000588066s, XML)
Steps: 400      Time: 0.4      CPU Time: 0.0534871s
Writing: "ADR_mesh_aligned_4.chk" (0.000543744s, XML)
Steps: 500      Time: 0.5      CPU Time: 0.0529716s
Writing: "ADR_mesh_aligned_5.chk" (0.000545313s, XML)
Steps: 600      Time: 0.6      CPU Time: 0.0532116s
Writing: "ADR_mesh_aligned_6.chk" (0.000557265s, XML)
Steps: 700      Time: 0.7      CPU Time: 0.0532916s
Writing: "ADR_mesh_aligned_7.chk" (0.000709478s, XML)
Steps: 800      Time: 0.8      CPU Time: 0.0533924s
Writing: "ADR_mesh_aligned_8.chk" (0.000791041s, XML)
Steps: 900      Time: 0.9      CPU Time: 0.0534512s
Writing: "ADR_mesh_aligned_9.chk" (0.000566883s, XML)
Steps: 1000     Time: 1      CPU Time: 0.0532088s
```

(续下页)

(接上页)

```

Writing: "ADR_mesh_aligned_10.chk" (0.000583935s, XML)
Time-integration : 0.534853s
renaming "ADR_mesh_aligned.fld" -> "ADR_mesh_aligned.bak0.fld"
Writing: "ADR_mesh_aligned.fld" (0.0112913s, XML)
-----
Total Computation Time = 1s
-----
L 2 error (variable u) : 0.00863475
L inf error (variable u) : 0.0390366

```

在自己的目录下自行安装 **Nektar++**

1. 执行以下命令从 **GitHub** 上下载 **Nektar++** 源码:

```
git clone http://gitlab.nektar.info/nektar/nektar.git nektar++
```

2. 下载完成后进入 **nektar++** 目录并通过源码编译安装 (编译之前需要配置很多可选的编译选项, 用户根据自己的具体情况自行选择即可):

```

cd nektar++
mkdir build && cd build
cmake ../
make
make install

```

参考资料

- [Nektar 官网](#)

Neper-FEPX

简介

Neper 是一款用于多晶体生成和网格划分的免费/开源软件包。它可用于生成具有各种形态特性的多晶体, 从非常简单的形态到涉及晶粒细分的复杂、多相或多尺度微结构。生成的网格可以划分成适合有限元模拟的高质量网格。

FEPX 是一款用于多晶塑性的有限元软件包, 与 **Neper** 配合使用。它可以通过一个可扩展的并行框架, 对具有复杂微观结构的大型多晶聚集体的整体和局部力学行为进行建模。

可用版本

版本	平台	构建方式	模块名
4.6.1		源码	Neper/4.6.1-gcc-11.2.0 思源一号
1.3.0		源码	FEPX/1.3.0-gcc-11.2.0 思源一号

集群上的 **Neper** 和 **FEPX**

- 思源一号上的 *Neper* 和 *FEPX*

思源一号上的 **Neper** 和 **FEPX**自定义编译 **Neper** 和 **FEPX**

自安装依赖库

```
module load gcc
spack install nlopt
spack install povray
spack install gsl
spack install gmsh
```

思源一号上已部署所依赖的库及版本

```
nlopt/2.7.0-gcc-11.2.0-openblas
gmsh/4.8.4-gcc-11.2.0-hdf5-openblas
gsl/2.7.1-gcc-11.2.0
imagemagick/7.0.8-7-gcc-11.2.0
povray/3.7.0.8-gcc-11.2.0
openmpi/4.1.5-gcc-11.2.0
gcc/11.2.0
```

编译 **Neper** 和 **FEPX** 前导入所需的环境变量

```
module load nlopt/2.7.0-gcc-11.2.0-openblas
module load gmsh/4.8.4-gcc-11.2.0-hdf5-openblas
module load gsl/2.7.1-gcc-11.2.0
module load imagemagick/7.0.8-7-gcc-11.2.0
module load povray/3.7.0.8-gcc-11.2.0
module load openmpi/4.1.5-gcc-11.2.0
module load gcc/11.2.0
```


自定义编译 **Neper**

```
git clone https://github.com/rquey/neper.git
cd ~/neper/src
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=~/neper/build ..
cmake -DCMAKE_INSTALL_PREFIX_COMPLETION=~/completions ..
make
make install
```

可根据所需选择安装的目录，思源一号上的预编译版本安装目录如下：

```
cmake -DCMAKE_INSTALL_PREFIX=/dssg/opt/icelake/linux-centos8-icelake/gcc-11.
↪2.0/neper-v4.6.1/build ..
cmake -DCMAKE_INSTALL_PREFIX_COMPLETION=/dssg/opt/icelake/linux-centos8-
↪icelake/gcc-11.2.0/neper-v4.6.1/completions ..
```

自定义编译 **FEPX**

```
unzip FEPX-1.3.0.zip
cd fepx-1.3.0/src
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=~/fepx-v1.3.0 ..
make
make install
```

可根据所需选择安装的目录，思源一号上的预编译版本安装目录如下：

```
cmake -DCMAKE_INSTALL_PREFIX=/dssg/opt/icelake/linux-centos8-icelake/gcc-11.
↪2.0/fepx-v1.3.0 ..
```

思源一号上使用预编译的 **Neper** 和 **FEPX**

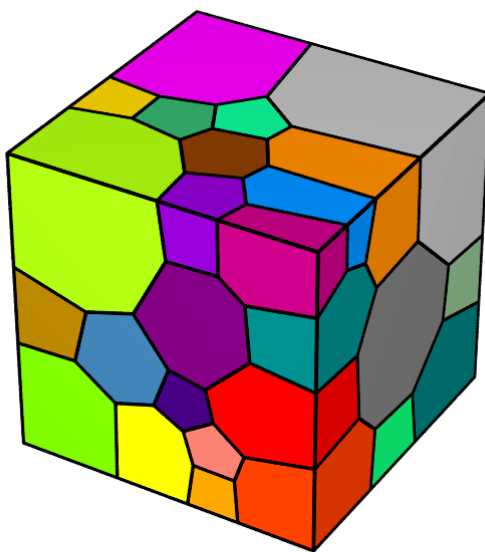
```
module load nlopt/2.7.0-gcc-11.2.0-openblas
module load gmsh/4.8.4-gcc-11.2.0-hdf5-openblas
module load gsl/2.7.1-gcc-11.2.0
module load imagemagick/7.0.8-7-gcc-11.2.0
module load povray/3.7.0.8-gcc-11.2.0
module load openmpi/4.1.5-gcc-11.2.0
module load gcc/11.2.0
export PATH=/dssg/opt/icelake/linux-centos8-icelake/gcc-11.2.0/neper-v4.6.1/
↪build/bin:$PATH
export PATH=/dssg/opt/icelake/linux-centos8-icelake/gcc-11.2.0/fepx-v1.3.0/
↪bin:$PATH
```

可将上述命令写入 `mymodule.sh` 文件，每次仅需 `source mymodule.sh` 即可调用

Step1-Create microstructure

```
neper -T -n 50 -reg 1 -morpho gg -o simulation  
neper -V simulation.tess -datacellcol id -print Image_Polycrystal
```

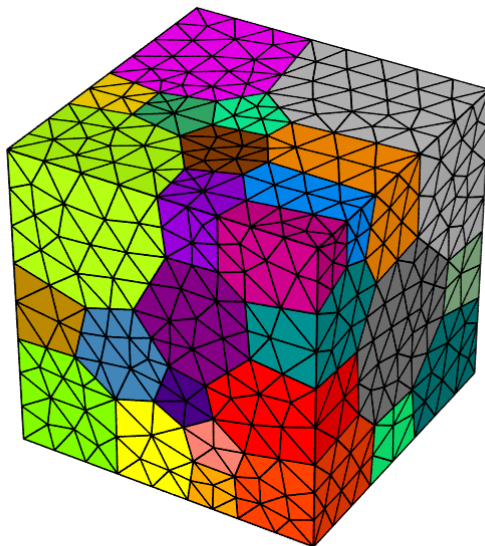
运行完成后生成 `simulation.tess` 和 `Image_Polycrystal.png`，如图 1 所示。



Step2-Create mesh

```
neper -M simulation.tess -order 2 -part 4  
neper -V simulation.tess,simulation.msh -dataelsetcol id -print Image_Mesh
```

运行完成后生成 `simulation.msh` 和 `Image_Mesh.png`，如图 2 所示。



Step-3 Apply material properties, BC and simulate; 将 “simulation.config” 文件复制到当前文件夹

```
cp /dssg/opt/icelake/linux-centos8-icelake/gcc-11.2.0/fepx-v1.3.0/data/  
→simulation.config ~  
fepx
```

Step-4 Postprocess raw results

```
neper -S .
```

运行后当前文件夹生成.sim 文件

step-5 Visualize the results

```
neper -V ../fepx_example.sim \  
-simstep 2 \  
-datanodecoo coo \  
-dataelt1drad 0.004 \  
-dataelt3dedgerad 0.0015 \  
-datanodecoofact 10 \  
-dataelt3dcol stress33 \  

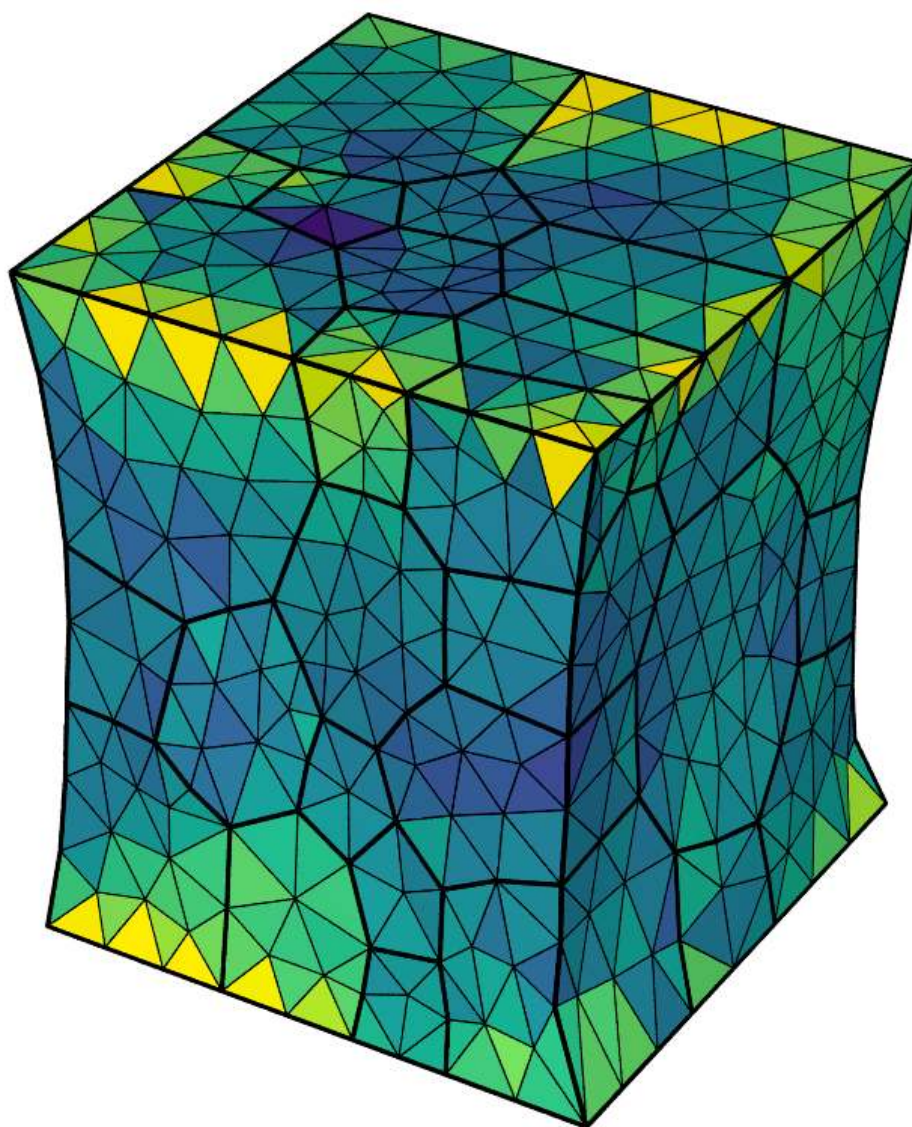
```

(续下页)

(接上页)

```
-dataeltscaletitle "Stress 33 (MPa)" \  
-dataeltscale 0:850 \  
-showelt1d all \  
-cameraangle 13.5 \  
-imagesize 800:800 \  
-print 1_s33_deform
```

运行后生成文件 1_s33_deform.png 和 1_s33_deform-scale3d.png，如图 3 所示



参考资料

- Neper 官网
- FEPX 官网

NWChem

简介

NWChem provides many methods for computing the properties of molecular and periodic systems using standard quantum mechanical descriptions of the electronic wavefunction or density. Its classical molecular dynamics capabilities provide for the simulation of macromolecules and solutions, including the computation of free energies using a variety of force fields. These approaches may be combined to perform mixed quantum-mechanics and molecular-mechanics simulations.

NWChem software can handle:

- Biomolecules, nanostructures, and solid-state
- From quantum to classical, and all combinations
- Ground and excited-states
- Gaussian basis functions or plane-waves
- Scaling from one to thousands of processors
- Properties and relativistic effects

可用的版本

版本	平台	构建方式	模块名
6.8.1		Spack	nwchem/6.8.1-intel-19.0.4-mpi

算例下载

```
wget https://nwchemgit.github.io/c240_631gs.nw
```

编辑文件，删除 (sed -i '6d' c240_631gs.nw) 第 6 行的 `scratch_dir /scratch`

运行示例

```
#!/bin/bash
#SBATCH --job-name=nwchem
#SBATCH --partition=small
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=10

ulimit -l unlimited
ulimit -s unlimited

module load intel/19.0.4
module load nwchem/6.8.1-intel-19.0.4-mpi

mpirun -np $SLURM_NTASKS nwchem c240_631gs.nw > c240_631gs.out
```

警告

软件运行需要库环境，加载时会自动创建 ~/.nwchemrc 文件。

使用如下脚本提交作业

```
sbatch test.slurm
```

运行结果见 c240_631gs.out

运行时间

π2.0

nwchem/6.8.1-intel-19.0.4-impi		
核数	10	20
CPU_time	1h50m37s	54m44s

参考资料

- [NWChem 文档](#)

Octave

GNU Octave 是一种采用高级编程语言的主要用于数值分析的软件。Octave 有助于以数值方式解决线性和非线性问题，并使用与 MATLAB 兼容的语言进行其他数值实验。它也可以作为面向批处理的语言使用。因为它是 GNU 计划的一部分，所以它是 GNU 通用公共许可证条款下的自由软件。

使用 **singularity** 容器提交 **Octave** 作业

以下是基于 Singularity 的作业脚本 octave_singularity.slurm 示例：

```
#!/bin/bash
#SBATCH -J octave_test
#SBATCH -p cpu
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 1
#SBATCH --ntasks-per-node=1

IMAGE_PATH=/lustre/share/img/octave.simg

ulimit -s unlimited
ulimit -l unlimited

singularity run $IMAGE_PATH octave [FILE_NAME]
```

并使用如下指令提交：

```
sbatch octave_singularity.slurm
```

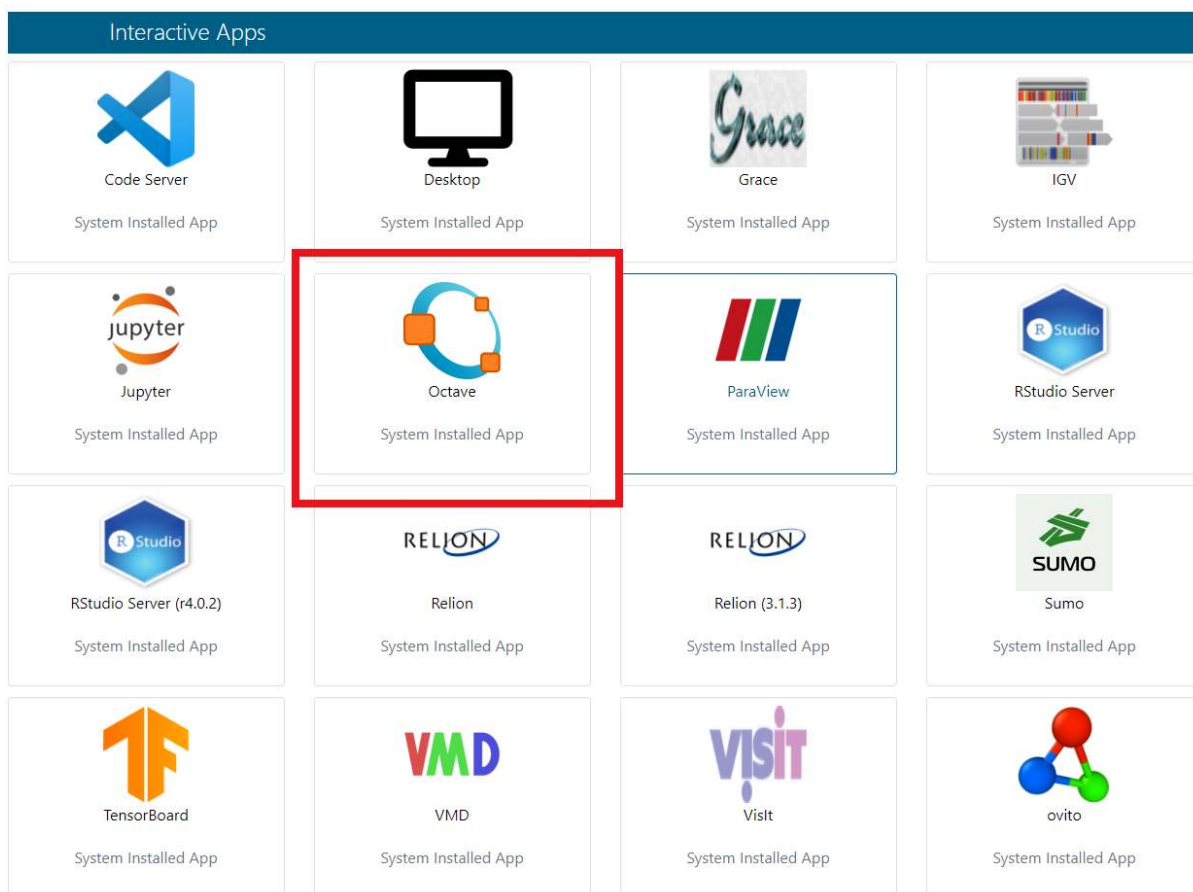
使用 **singularity** 容器提交 **Octave** 交互式作业

可以通过如下指令提交 **Octave** 交互式作业：

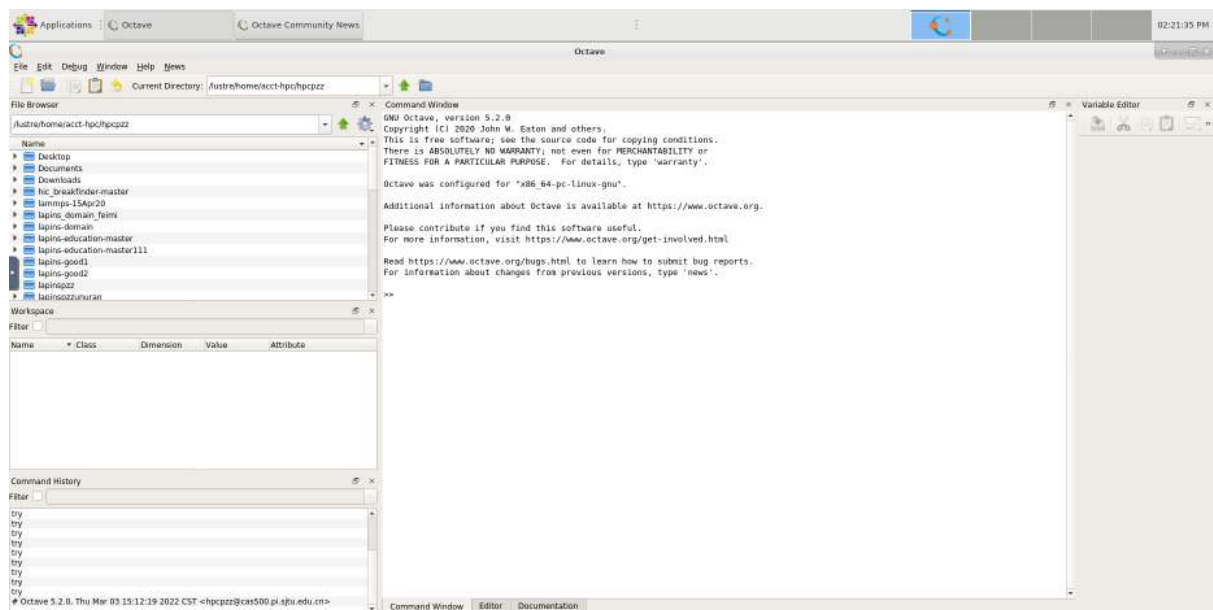
```
srun -p cpu -N 1 --exclusive --pty singularity run /lustre/share/img/octave.  
→sing octave-cli
```

使用 **HPC Studio** 启动 **Octave** 可视化界面

1. 用 **pi** 集群帐号登录 **HPC Studio** 平台；
2. 在 **Interactive Apps** 面板中点击如下图所示 **Octave** 选项申请 **Octave GUI** 界面 (要排一会儿队，请耐心等待)：



3. 申请成功后会看到如下界面：



4. Octave 的语法和 Matlab 几乎完全相同，我们可以在右侧 Command Window 窗口中进行简单的数学运算，如下图所示：

Command Window

```
GNU Octave, version 5.2.0
Copyright (C) 2020 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-pc-linux-gnu".

Additional information about Octave is available at https://www.octave.org.

Please contribute if you find this software useful.
For more information, visit https://www.octave.org/get-involved.html

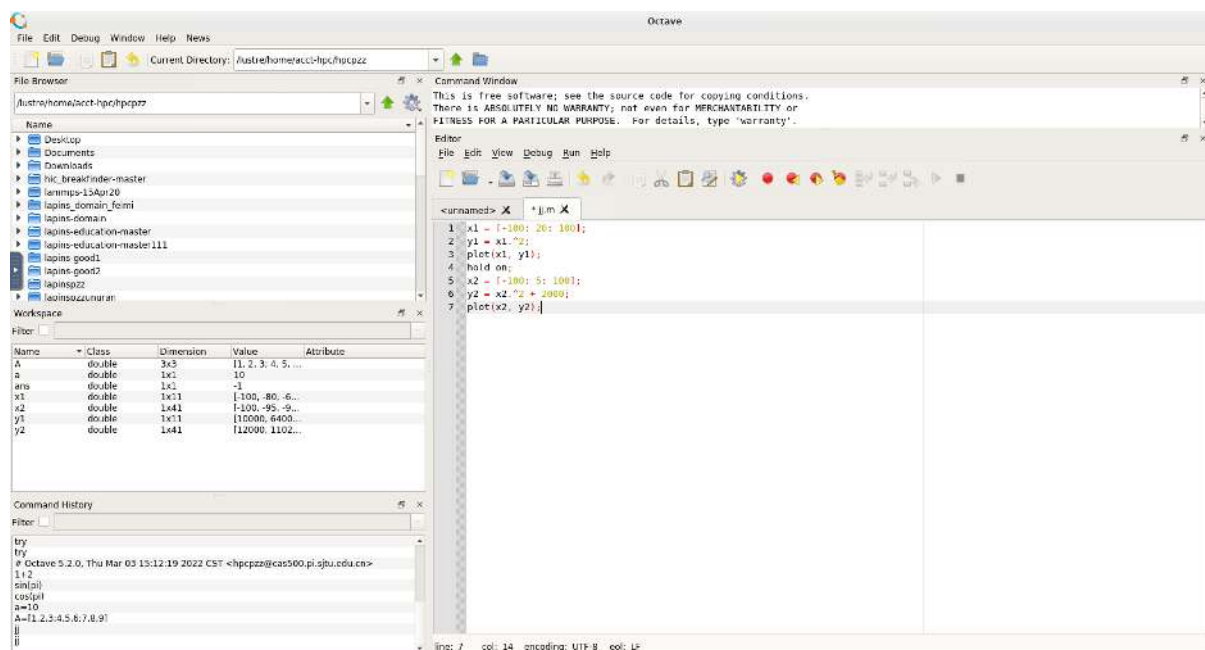
Read https://www.octave.org/bugs.html to learn how to submit bug reports.
For information about changes from previous versions, type 'news'.

>> 1+2
ans = 3
>> sin(pi)
ans = 1.2246e-16
>> cos(pi)
ans = -1
>> a=10
a = 10
>> A=[1,2,3;4,5,6;7,8,9]
A =

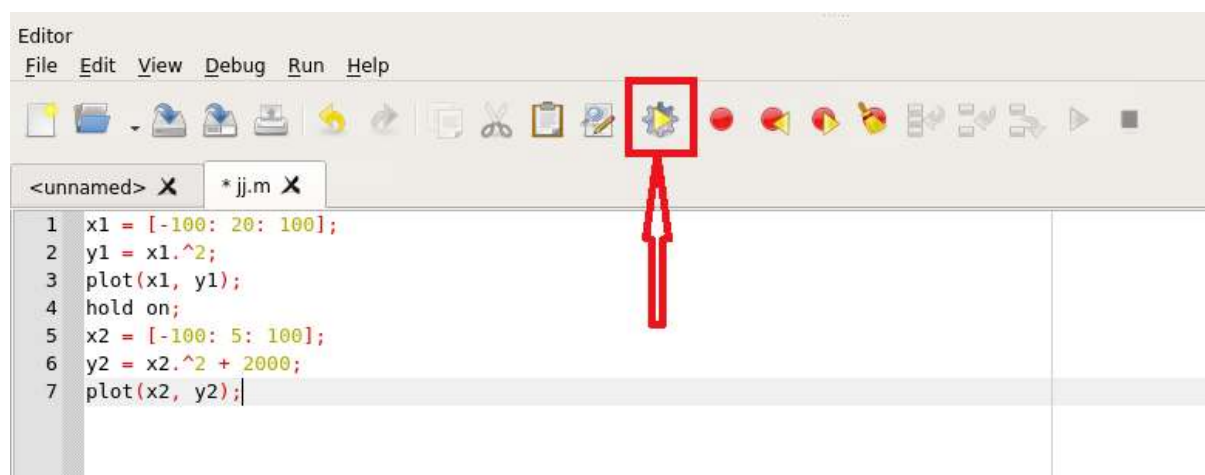
    1    2    3
    4    5    6
    7    8    9

>>
```

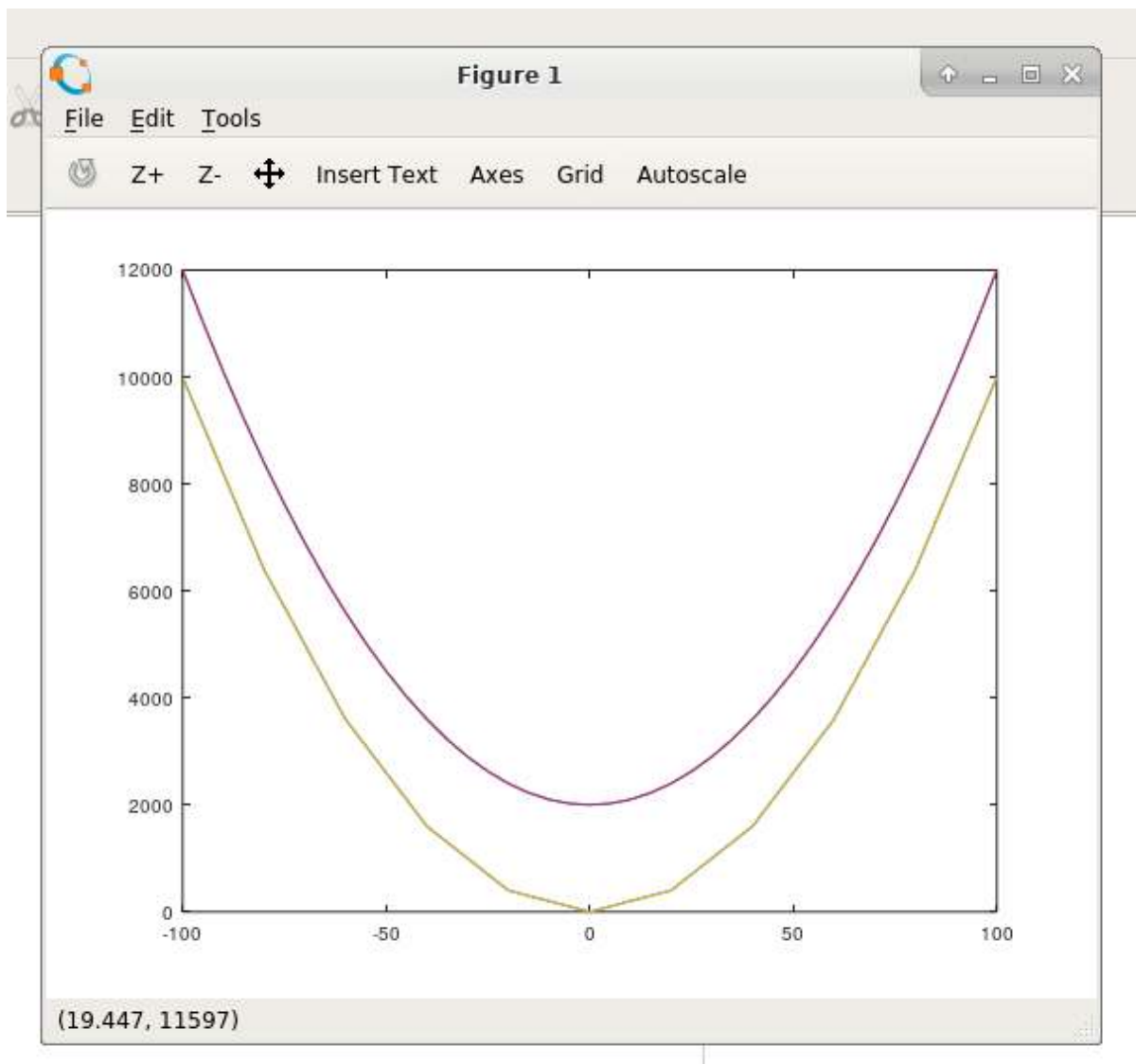
5. 如果要编写脚本进行比较复杂的计算的话 (绘图或者是其他), 可以点击左上角 **File->New->New Script** 选项, 然后在弹出的 **Editor** 窗口进行脚本编写, 如下图所示:



6. 编写完成之后，点击如下图所示按钮保存为.m 文件并运行：



7. 结果如图所示：



参考资料

- [Octave](#)
- [Singularity 文档](#)

OpenRadioss

简介

OpenRadioss 是一个公开的开源代码库，由全球研究人员、软件开发人员和行业领导者组成的社区每天都在增强它。OpenRadioss 正在改变行业规则，让用户能够快速做出贡献，应对快速发展的技术带来的最新挑战，如电池开发、轻质材料和复合材料、人体模型和生物材料、自动驾驶和飞行，以及通过虚拟测试为乘客提供尽可能安全的环境的愿望。有了 OpenRadioss，科学家和技术人员可以在专业维护下将研究重点放在稳定的代码库上，该代码库受益于现有有限元功能的大型库以及提供给贡献者的持续集成和持续开发工具。

OpenRadioss 安装以及说明

在思源一号上自行安装 OpenRadioss

1. 首先在思源一号上自己的家目录下新建一个目录作为安装目录, 并进入该目录:

```
mkdir OpenRadioss
cd OpenRadioss
```

2. 申请计算资源并加载所需模块:

```
srunk -p 64c512g -n 10 --pty /bin/bash
module load netcdf-fortran/4.5.3-gcc-8.3.1-hdf5-openmpi
module load openmpi/4.1.1-gcc-9.3.0
module load gcc/9.3.0
```

3. 从 [OpenRadioss 官网](#) 下载 Source code(tar.gz) 到自己本地计算机并上传到刚才创建的目录下 (可借助 filezilla 或者超算可视化平台传输文件):

4. 解压上传的压缩文件并进入该目录:

```
tar -xvzf OpenRadioss-latest-20221205.tar.gz
cd OpenRadioss-latest-20221205
```

5. 进入 starter 目录并开始第一步编译:

```
cd starter
./build_script.sh -arch=linux64_gf
```

6. 退出 starter 目录, 进入 engine 目录, 开始第二步编译:

```
cd ../engine
./build_script.sh -arch=linux64_gf -mpi=ompi -mpi-root=/dssg/opt/icelake/
→linux-centos8-icelake/gcc-9.3.0/openmpi-4.1.1-
→usre7vgur4rv6jllqd4yuf5gg57kothm -mpi-include=/dssg/opt/icelake/linux-
→centos8-icelake/gcc-9.3.0/openmpi-4.1.1-usre7vgur4rv6jllqd4yuf5gg57kothm/
→include -mpi-libdir=/dssg/opt/icelake/linux-centos8-icelake/gcc-9.3.0/
→openmpi-4.1.1-usre7vgur4rv6jllqd4yuf5gg57kothm/lib
```

在 pi2.0 上自行安装 OpenRadioss

1. 此步骤和上文完全相同;
2. 申请计算资源并加载所需模块:

```
srunk -p cpu -N 1 --ntasks-per-node 40 --exclusive --pty /bin/bash
module load netcdf-fortran/4.5.2-gcc-8.3.0-openmpi
module load openmpi/4.0.5-gcc-9.2.0
module load gcc/9.2.0
```

3. 此步骤和上文完全相同;

4. 此步骤和上文完全相同；
5. 此步骤和上文完全相同；
6. 退出 **starter** 目录，进入 **engine** 目录，开始第二步编译：

```
cd ../engine
./build_script.sh -arch=linux64_gf -mpi=ompi -mpi-root=/lustre/opt/
↪cascadelake/linux-centos7-cascadelake/gcc-9.2.0/openmpi-4.0.5-
↪vpswzpisyc6gl6e5isbal66yykxdc6k -mpi-include=/lustre/opt/cascadelake/
↪linux-centos7-cascadelake/gcc-9.2.0/openmpi-4.0.5-
↪vpswzpisyc6gl6e5isbal66yykxdc6k/include -mpi-libdir=/lustre/opt/
↪cascadelake/linux-centos7-cascadelake/gcc-9.2.0/openmpi-4.0.5-
↪vpswzpisyc6gl6e5isbal66yykxdc6k/lib
```

在 **pi2.0 kos** 系统上自行安装 **OpenRadioss**

1. 下载并解压源代码文件，申请计算节点

```
wget https://github.com/OpenRadioss/OpenRadioss/archive/refs/tags/latest-
↪20231220.tar.gz
tar zxvf latest-20231220.tar.gz
cd OpenRadioss-latest-20231220/
srun -p cpu -n 8 --pty /bin/bash #申请计算节点
```

2. 加载编译所需模块：

```
module load openmpi/4.1.5-gcc-8.5.0
module load netcdf-fortran/4.5.2-gcc-8.5.0-openmpi
module load cmake/3.26.3-gcc-8.5.0
module load python/3.10.10-gcc-8.5.0
```

3. 第一步编译

```
cd starter
./build_script.sh -arch=linux64_gf
```

4. 第二步编译

```
cd ../engine
./build_script.sh -arch=linux64_gf -mpi=ompi -mpi-root=/lustre/opt/
↪cascadelake/linux-rhel8-skylake_avx512/gcc-8.5.0/openmpi-4.1.5-
↪sjnibarr4bwsceb3ncopdeyfigesuzfk/ -mpi-include=/lustre/opt/cascadelake/
↪linux-rhel8-skylake_avx512/gcc-8.5.0/openmpi-4.1.5-
↪sjnibarr4bwsceb3ncopdeyfigesuzfk/include -mpi-libdir=/lustre/opt/
↪cascadelake/linux-rhel8-skylake_avx512/gcc-8.5.0/openmpi-4.1.5-
↪sjnibarr4bwsceb3ncopdeyfigesuzfk/lib
```

参考资料

- [OpenRadioss 官网](#)

WhiskyTHC

简介

WhiskyTHC includes the sophisticated microphysics framework developed at the Max Planck Institute for Gravitational Physics for the original Whisky code and uses advanced C++ programming techniques to generate optimized numerical code for a variety of numerical methods (Templated Hydrodynamics).

WhiskyTHC 安装说明

在思源一号上自行安装 **WhiskyTHC**

1. 首先在自己的家目录下新建一个目录作为安装目录, 并进入该目录:

```
mkdir WhiskyTHC
cd WhiskyTHC
```

2. 申请计算资源并加载所需模块:

```
srunk -p 64c512g -n 40 --pty /bin/bash
module purge
module load gcc/9.3.0
module load hwloc/2.6.0-gcc-9.3.0
module load openmpi/4.1.1-gcc-9.3.0
```

3. 复制源码包到当前目录并解压, 然后进入 FreeTHC 目录:

```
cp /dssg/share/Sourcefile/hpcpzz/WhiskyTHC-20221221.tar ./
tar xvpf ./WhiskyTHC-20221221.tar
cd FreeTHC
```

4. 在 Cactus/batchtools/templates/cactus 目录下编写如下 sjtusy.cfg 配置文件:

```
# Whenever this version string changes, the application is configured
# and rebuilt from scratch
VERSION = 2019-09-03

CPP = cpp -DFORTRAN_DISABLE_IEEE_ARITHMETIC
FPP = cpp -DFORTRAN_DISABLE_IEEE_ARITHMETIC
CC = gcc
CXX = g++
F77 = gfortran
F90 = gfortran

FPPFLAGS = -traditional
```

(续下页)

(接上页)

```

CPPFLAGS =
FPPFLAGS =
CFLAGS    = -g -std=gnu99
CXXFLAGS  = -g -std=gnu++11
F77FLAGS  = -g -fcray-pointer -m128bit-long-double -ffixed-line-length-none -
↳fno-range-check
F90FLAGS  = -g -fcray-pointer -m128bit-long-double -ffixed-line-length-none -
↳fno-range-check

LDFLAGS = -rdynamic

DEBUG            = no
CPP_DEBUG_FLAGS = -DCARPET_DEBUG -DHRSCC_DEBUG -DTHC_DEBUG -DCPPUTILS_DEBUG
FPP_DEBUG_FLAGS = -DCARPET_DEBUG -DHRSCC_DEBUG -DTHC_DEBUG -DCPPUTILS_DEBUG
C_DEBUG_FLAGS   = -O0
CXX_DEBUG_FLAGS = -O0
F77_DEBUG_FLAGS = -O0
F90_DEBUG_FLAGS = -O0

OPTIMISE        = yes
CPP_OPTIMISE_FLAGS =
FPP_OPTIMISE_FLAGS =
C_OPTIMISE_FLAGS   = -O2
CXX_OPTIMISE_FLAGS = -O2
F77_OPTIMISE_FLAGS = -O2
F90_OPTIMISE_FLAGS = -O2

PROFILE          = no
CPP_PROFILE_FLAGS =
FPP_PROFILE_FLAGS =
C_PROFILE_FLAGS   = -pg
CXX_PROFILE_FLAGS = -pg
F77_PROFILE_FLAGS = -pg
F90_PROFILE_FLAGS = -pg

OPENMP           = yes
CPP_OPENMP_FLAGS = -fopenmp
FPP_OPENMP_FLAGS = -D_OPENMP
C_OPENMP_FLAGS   = -fopenmp
CXX_OPENMP_FLAGS = -fopenmp
F77_OPENMP_FLAGS = -fopenmp
F90_OPENMP_FLAGS = -fopenmp

WARN             = yes
CPP_WARN_FLAGS   = -Wall
FPP_WARN_FLAGS   = -Wall
C_WARN_FLAGS     = -Wall
CXX_WARN_FLAGS   = -Wall

```

(续下页)

(接上页)

```

F77_WARN_FLAGS = -Wall
F90_WARN_FLAGS = -Wall

#BLAS_DIR = /usr
#BOOST_DIR = /usr
#FFTW3_DIR = /usr
#GSL_DIR = /usr
#HDF5_DIR = /usr
#HWLOC_DIR = /usr
#LAPACK_DIR = /usr

MPI_DIR = /dssg/opt/icelake/linux-centos8-icelake/gcc-9.3.0/openmpi-4.1.1-
→usre7vgur4rv6jllqd4yuf5gg57kothm
BLAS_DIR = BUILD
BOOST_DIR = BUILD
FFTW3_DIR = BUILD
GSL_DIR = BUILD
HDF5_DIR = BUILD
HWLOC_DIR = /dssg/opt/icelake/linux-centos8-icelake/gcc-9.3.0/hwloc-2.6.0-
→mkqkyei3gxxttri4uaafreqteyhyj2exl
LAPACK_DIR = BUILD

PTHREADS_DIR = NO_BUILD

```

5. 构建 thc 依赖库:

```

cd Cactus
yes | make thc THORNLIST=thornlists/full.th      options=batchtools/
→templates/cactus/sjtusy.cfg

```

6. 构建 thc :

```
make -j40 thc
```

在 pi2.0 上自行安装 WhiskyTHC

1. 此步骤和上文完全相同;
2. 申请计算资源并加载所需模块:

```

srun -N 1 -p cpu --exclusive --pty /bin/bash
module purge
module load gcc/9.2.0
module load hwloc/1.11.11-gcc-9.2.0
module load openmpi/3.1.5-gcc-9.2.0

```

3. 复制源码包到当前目录并解压, 然后进入 FreeTHC 目录:


```
cp /lustre/share/Sourcefile/hpcpzz/WhiskyTHC-20221221.tar ./
tar xvpf ./WhiskyTHC-20221221.tar
cd FreeTHC
```

4. 在 Cactus/batchtools/templates/cactus 目录下编写如下 sjtupi.cfg 配置文件:

```
# Whenever this version string changes, the application is configured
# and rebuilt from scratch
VERSION = 2019-09-03

CPP = cpp -DFORTRAN_DISABLE_IEEE_ARITHMETIC
FPP = cpp -DFORTRAN_DISABLE_IEEE_ARITHMETIC
CC = gcc
CXX = g++
F77 = gfortran
F90 = gfortran

FPPFLAGS = -traditional

CPPFLAGS =
FPPFLAGS =
CFLAGS = -g -std=gnu99
CXXFLAGS = -g -std=gnu++11
F77FLAGS = -g -fcray-pointer -m128bit-long-double -ffixed-line-length-none -
↪fno-range-check
F90FLAGS = -g -fcray-pointer -m128bit-long-double -ffixed-line-length-none -
↪fno-range-check

LDLFLAGS = -rdynamic

DEBUG = no
CPP_DEBUG_FLAGS = -DCARPET_DEBUG -DHRSCC_DEBUG -DTHC_DEBUG -DCPPUTILS_DEBUG
FPP_DEBUG_FLAGS = -DCARPET_DEBUG -DHRSCC_DEBUG -DTHC_DEBUG -DCPPUTILS_DEBUG
C_DEBUG_FLAGS = -O0
CXX_DEBUG_FLAGS = -O0
F77_DEBUG_FLAGS = -O0
F90_DEBUG_FLAGS = -O0

OPTIMISE = yes
CPP_OPTIMISE_FLAGS =
FPP_OPTIMISE_FLAGS =
C_OPTIMISE_FLAGS = -O2
CXX_OPTIMISE_FLAGS = -O2
F77_OPTIMISE_FLAGS = -O2
F90_OPTIMISE_FLAGS = -O2

PROFILE = no
CPP_PROFILE_FLAGS =
FPP_PROFILE_FLAGS =
C_PROFILE_FLAGS = -pg
```

(续下页)

(接上页)

```

CXX_PROFILE_FLAGS = -pg
F77_PROFILE_FLAGS = -pg
F90_PROFILE_FLAGS = -pg

OPENMP              = yes
CPP_OPENMP_FLAGS   = -fopenmp
FPP_OPENMP_FLAGS   = -D_OPENMP
C_OPENMP_FLAGS     = -fopenmp
CXX_OPENMP_FLAGS   = -fopenmp
F77_OPENMP_FLAGS   = -fopenmp
F90_OPENMP_FLAGS   = -fopenmp

WARN                = yes
CPP_WARN_FLAGS     = -Wall
FPP_WARN_FLAGS     = -Wall
C_WARN_FLAGS       = -Wall
CXX_WARN_FLAGS     = -Wall
F77_WARN_FLAGS     = -Wall
F90_WARN_FLAGS     = -Wall

#BLAS_DIR = /usr
#BOOST_DIR = /usr
#FFTW3_DIR = /usr
#GSL_DIR = /usr
#HDF5_DIR = /usr
#HWLOC_DIR = /usr
#LAPACK_DIR = /usr

MPI_DIR = /lustre/opt/cascadelake/linux-centos7-cascadelake/gcc-9.2.0/
↪openmpi-3.1.5-gtpczurejutqns55psqujgakh7vpzqot
BLAS_DIR = BUILD
BOOST_DIR = BUILD
FFTW3_DIR = BUILD
GSL_DIR = BUILD
HDF5_DIR = BUILD
HWLOC_DIR = /lustre/opt/cascadelake/linux-centos7-cascadelake/gcc-9.2.0/
↪hwloc-1.11.11-whc7fqivalihbihdrueouc4pcnzbcror
LAPACK_DIR = BUILD

PTHREADS_DIR = NO_BUILD

```

5. 构建 thc 依赖库:

```

cd Cactus
yes | make thc THORNLIST=thornlists/full.th      options=batchtools/
↪templates/cactus/sjtupi.cfg

```

6. 构建 thc :

```
make -j40 thc
```

参考资料

- [WhiskyTHC 官网](#)

OpenFOAM

OpenFOAM（英文 Open Source Field Operation and Manipulation 的缩写，意为开源的场运算和处理软件）是对连续介质力学问题进行数值计算的 C++ 自由软件工具包，其代码遵守 GNU 通用公共许可证。它可进行数据预处理、后处理和自定义求解器，常用于计算流体力学 (CFD) 领域。该软件由 OpenFOAM 基金会维护。

可用 OpenFOAM 版本

版本	平台	构建方式	模块名
7		Spack	openfoam-org/7-gcc-7.4.0-openmpi
8		容器	/lustre/share/img/x86/openfoam/8-gcc8-openmpi4-centos8.sif
1712		Spack	openfoam/1712-gcc-7.4.0-openmpi
1912		Spack	openfoam/1912-gcc-7.4.0-openmpi
2012		容器	/ lustre/ share/ img/ x86/ openfoam/ 2012-gcc4-openmpi4-centos7.sif
2106		容器	/ lustre/ share/ img/ x86/ openfoam/ 2106-gcc4-openmpi4-centos7.sif
8		容器	/ lustre/ share/ img/ aarch64/ openfoam/ 8-gcc8-openmpi4-centos8.sif
1912		Spack	openfoam/1912-gcc-9.3.0-openmpi
2012		容器	/ lustre/ share/ img/ aarch64/ openfoam/ 2012-gcc4-openmpi4-centos7.sif
2106		容器	/ lustre/ share/ img/ aarch64/ openfoam/ 2106-gcc4-openmpi4-centos7.sif
1912		容器	/lustre/share/img/aarch64/openfoam/1912.sif

OpenFOAM 基本使用

思源一号上的 **openfoam-org7(Spack 构建)**

1. 从 openfoam-org7 的安装目录中将 tutorials 目录整个复制到自己目录下 openfoamTest1 目录中:

```
module load openfoam-org/7-gcc-11.2.0-openmpi
mkdir openfoamTest1
cd openfoamTest1
cp -rv $FOAM_TUTORIALS ./
```

2. 为了运行 motorBike 算例 (多核并行), 执行以下命令进入相对应目录:

```
cd ./tutorials/incompressible//simpleFoam/motorBike
```

3. 在此目录下编写以下 openfoam.slurm 脚本:

```
#!/bin/bash

#SBATCH --job-name=openfoam      # 作业名
#SBATCH --partition=64c512g      # 64c512g队列
#SBATCH --ntasks-per-node=6      # 每节点核数
#SBATCH -n 6                     # 作业核心数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

module load openmpi/4.1.1-gcc-11.2.0

cp $FOAM_TUTORIALS/resources/geometry/motorBike.obj.gz constant/triSurface/
surfaceFeatures
blockMesh
decomposePar -copyZero
mpirun -np 6 snappyHexMesh -overwrite -parallel
mpirun -np 6 patchSummary -parallel
mpirun -np 6 potentialFoam -parallel
mpirun -np 6 simpleFoam -parallel
reconstructParMesh -constant
reconstructPar -latestTime
```

小心

某些情况下, 当并行核数达到一定数目时, 直接运行大规模并行作业可能会出现报错, 这是超算集群上各节点之间的连接方式造成的. 这时需要修改一下 MPI 的默认执行方式, 比如说将 `mpirun -np 6 simpleFoam -parallel` 改为 `mpirun -np 6 -mca btl self,vader,tcp simpleFoam -parallel`.

4. 使用 sbatch 提交作业:

```
sbatch openfoam.slurm
```

5. 运行结束后即可在该目录下看到如下结果:

```
0
500
9953216.err
9953216.out
Allclean
Allrun
constant
postProcessing
processor0
processor1
processor2
processor3
processor4
processor5
openfoam.slurm
system
```

思源一号上的 **openfoam2106(Spack 构建)**

1. 从 openfoam2106 的安装目录中将 tutorials 目录整个复制到自己目录下 openfoamTest1 目录中:

```
module load openfoam/2106-gcc-8.3.1-openmpi
mkdir openfoamTest1
cd openfoamTest1
cp -rv $FOAM_TUTORIALS ./
```

2. 为了运行 motorBike 算例 (多核并行), 执行以下命令进入相对应目录:

```
cd ./tutorials/incompressible//simpleFoam/motorBike
```

3. 在此目录下编写以下 openfoam.slurm 脚本:

```
#!/bin/bash

#SBATCH --job-name=openfoam      # 作业名
#SBATCH --partition=64c512g      # 64c512g队列
#SBATCH --ntasks-per-node=6      # 每节点核数
#SBATCH -n 6                     # 作业核心数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited
```

(续下页)

(接上页)

```
module load openmpi/4.1.1-gcc-8.3.1

./Allclean
./Allrun
```

4. 使用 sbatch 提交作业:

```
sbatch openfoam.slurm
```

5. 运行结束后即可在该目录下看到如下结果:

```
0.orig
500
Allclean
Allrun
constant
log.blockMesh
log.checkMesh
log.decomposePar
log.patchSummary
log.potentialFoam
log.reconstructPar
log.reconstructParMesh
log.simpleFoam
log.snappyHexMesh
log.surfaceFeatureExtract
log.topoSet
openfoam.slurm
postProcessing
processor0
processor1
processor2
processor3
processor4
processor5
system
```

pi2.0 上的 openfoam-org7(Spack 构建)

1. 从 openfoam-org7 的安装目录中将 tutorials 目录整个复制到自己目录下 openfoamTest1 目录中:

```
module load openfoam-org/7-gcc-7.4.0-openmpi
mkdir openfoamTest1
cd openfoamTest1
cp -rv $FOAM_TUTORIALS ./
```

2. 运行 cavity 算例 (单核串行), 执行以下命令进入相对应目录:

```
cd ../tutorials/incompressible/icoFoam/cavity/cavity
```

3. 此时可以看到以下 `0`、`constant`、`system` 三个目录 (一个典型的 `openfoam` 算例均包含这三个目录):

```
├─ 0
│   ├── p
│   └── u
├─ constant
│   └── transportProperties
├─ system
│   ├── blockMeshDict
│   ├── controlDict
│   ├── fvSchemes
│   └── fvSolution
```

其中 `0` 目录主要包含待求解问题的边界条件和初始条件; `constant` 目录主要包含物性参数、湍流模型参数、更高级的物理模型等; `system` 目录主要包含计算时间和数值求解格式等计算参数。这三个目录包含了待求解问题所必须指定的所有物理参数和计算参数, 用户可根据自己的需求进行合理修改以提高计算结果的准确性。

4. 在此目录下编写以下 `openfoam.slurm` 脚本:

```
#!/bin/bash

#SBATCH --job-name=openfoam      # 作业名
#SBATCH --partition=small        # small队列
#SBATCH --ntasks-per-node=1     # 每节点核数
#SBATCH -n 1                     # 作业核心数
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

#module load openfoam-org/7-gcc-7.4.0-openmpi

blockMesh
icoFoam
```

5. 使用 `sbatch` 提交作业:

```
sbatch openfoam.slurm
```

6. 运行结束后会看到 `constant` 目录下多出了一个 `polyMesh` 目录, 该目录保存了计算用的网格信息; 而同级目录下多出了 `0.1`、`0.2`、`0.3`、`0.4`、`0.5` 这五个目录, 这几个目录记录了在五个不同时刻的物理场的计算结果:

```
├─ 0
│   └── p
```

(续下页)

(接上页)

```
|   └─ U
├─ 0.1
|   └─ p
|   └─ phi
|   └─ U
|   └─ uniform
|       └─ time
├─ 0.2
|   └─ p
|   └─ phi
|   └─ U
|   └─ uniform
|       └─ time
├─ 0.3
|   └─ p
|   └─ phi
|   └─ U
|   └─ uniform
|       └─ time
├─ 0.4
|   └─ p
|   └─ phi
|   └─ U
|   └─ uniform
|       └─ time
├─ 0.5
|   └─ p
|   └─ phi
|   └─ U
|   └─ uniform
|       └─ time
├─ constant
|   └─ polyMesh
|       └─ boundary
|       └─ faces
|       └─ neighbour
|       └─ owner
|       └─ points
|   └─ transportProperties
├─ openfoam.slurm
└─ system
    └─ blockMeshDict
    └─ controlDict
    └─ fvSchemes
    └─ fvSolution
```


自行构建 OpenFOAM 镜像

以 OpenFOAM-org7 为例

1. 从 pi2.0 登录节点跳转至容器构建节点 (只能从 pi2.0 跳转，思源一号不行):

```
ssh build@container-x86
```

2. 创建和进入临时工作目录:

```
cd $(mktemp -d)
```

3. 创建如下镜像定义文件 openfoam7-gcc4-openmpi4-centos7.def，可按需修改:

```
Bootstrap: docker
From: centos:7

%help
  This recipe provides an OpenFOAM-7 environment installed
  with GCC 4 and OpenMPI-4 on CentOS 7.

%labels
  Author Fatih Ertinaz

%post
  ### Install prerequisites
  yum groupinstall -y 'Development Tools'
  yum install -y wget git openssl-devel libuuid-devel

  ### Install OpenMPI
  # Why openmpi-4.x is needed: https://github.com/hpcng/singularity/issues/
  ↪2590
  vrs=4.0.3
  wget https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-${vrs}.
  ↪tar.gz
  tar xf openmpi-${vrs}.tar.gz && rm -f openmpi-${vrs}.tar.gz
  cd openmpi-${vrs}
  ./configure --prefix=/opt/openmpi-${vrs}
  make all install
  make all clean

  ### Update environment - OpenMPI
  export MPI_DIR=/opt/openmpi-${vrs}
  export MPI_BIN=$MPI_DIR/bin
  export MPI_LIB=$MPI_DIR/lib
  export MPI_INC=$MPI_DIR/include

  export PATH=$MPI_BIN:$PATH
  export LD_LIBRARY_PATH=$MPI_LIB:$LD_LIBRARY_PATH

  ### OpenFOAM version
```

(续下页)

(接上页)

```

pkg=OpenFOAM
vrs=7

### Install under /opt
base=/opt/$pkg
mkdir -p $base && cd $base

### Download OF
wget -O - http://spack.pi.sjtu.edu.cn/mirror/openfoam-org/openfoam-org-7.
→0.tar.gz | tar xz
mv $pkg-$vrs-version-$vrs $pkg-$vrs

## Download ThirdParty
wget -O - http://spack.pi.sjtu.edu.cn/mirror/openfoam-org/ThirdParty-7.
→tar.gz | tar xz
mv ThirdParty-$vrs-version-$vrs ThirdParty-$vrs

### Change dir to OpenFOAM-version
cd $pkg-$vrs

### Get rid of unalias otherwise singularity fails
sed -i 's,FOAM_INST_DIR=$HOME\/$WM_PROJECT,FOAM_INST_DIR="'$base'",g'
→etc/bashrc
sed -i 's/alias wmUnset/#alias wmUnset/' etc/config.sh/aliases
sed -i 's/77s/else/#else/' etc/config.sh/aliases
sed -i 's/unalias wmRefresh/#unalias wmRefresh/' etc/config.sh/aliases

### Compile and install
. etc/bashrc
./Allwmake -j$(nproc) 2>&1 | tee log.Allwmake

### Clean-up environment
rm -rf platforms/$WM_OPTIONS/applications
rm -rf platforms/$WM_OPTIONS/src

cd $base/ThirdParty-$vrs
rm -rf build
rm -rf gcc-* gmp-* mpfr-* binutils-* boost* ParaView-* qt-*

strip $FOAM_APPBIN/*

### Source bashrc at runtime
echo '. /opt/OpenFOAM/OpenFOAM-7/etc/bashrc' >> $SINGULARITY_ENVIRONMENT

%environment
export MPI_DIR=/opt/openmpi-4.0.3
export MPI_BIN=$MPI_DIR/bin
export MPI_LIB=$MPI_DIR/lib
export MPI_INC=$MPI_DIR/include

```

(续下页)

(接上页)

```

export PATH=$MPI_BIN:$PATH
export LD_LIBRARY_PATH=$MPI_LIB:$LD_LIBRARY_PATH

%test
. /opt/OpenFOAM/OpenFOAM-7/etc/bashrc
icoFoam -help

%runscript
echo
echo "OpenFOAM installation is available under $WM_PROJECT_DIR"
echo

```

4. 执行以下命令构建镜像，大约会消耗 2-3 小时：

```

docker run --privileged --rm -v \
${PWD}:/home/singularity \
sjtuhpc/centos7-singularity:x86 \
singularity build /home/singularity/openfoam7-gcc4-openmpi4-centos7.sif /
→home/singularity/openfoam7-gcc4-openmpi4-centos7.def

```

5. 将构建出的容器镜像传回自己家目录的根目录 (用户需根据自身具体情况将 **hpcpzz@login2** 进行修改, 和自己对应起来):

```

scp ./openfoam7-gcc4-openmpi4-centos7.sif hpcpzz@login2:~/

```

6. 回到自己的家目录, 测试镜像是否能够正常使用:

```

singularity exec ./openfoam7-gcc4-openmpi4-centos7.sif blockMesh

```

7. 得到以下结果则表示镜像构建成功:

```

/*-----
→-* \
=====
\\      /  F ield      | OpenFOAM: The Open Source CFD Toolbox
\\      /  O peration  | Website:  https://openfoam.org
  \\    /   A nd       | Version:  7
   \\/     M anipulation |
\*-----
→-*/
Build   : 7
Exec    : /opt/OpenFOAM/OpenFOAM-7/platforms/linux64GccDPInt32Opt/bin/
→blockMesh
Date    : Nov 25 2022
Time    : 13:30:41
Host    : "cas332.pi.sjtu.edu.cn"
PID     : 174880
I/O     : uncollated

```

(续下页)

(接上页)

```

Case    : /lustre/home/acct-hpc/hpcpzz
nProcs  : 1
fileModificationChecking : Monitoring run-time modified files using
↳timeStampMaster (fileModificationSkew 10)
allowSystemOperations : Allowing user-supplied system call operations

// * * * * *
↳* //
Create time

--> FOAM FATAL ERROR:
cannot find file "/lustre/home/acct-hpc/hpcpzz/system/controlDict"

    From function virtual Foam::autoPtr<Foam::ISstream>
↳Foam::fileOperations::uncollatedFileOperation::readStream(Foam::regIOobject&
↳, const Foam::fileName&, const Foam::word&, bool) const
    in file global/fileOperations/uncollatedFileOperation/
↳uncollatedFileOperation.C at line 538.

FOAM exiting

```

在自己的目录下自行源码编译 OpenFOAM

以 **OpenFOAM-org10** 为例

1. 从登录节点申请计算资源:

```

srun -p cpu -N 1 --ntasks-per-node=40 --pty /bin/bash(思源一号)
或者
srun -p 64c512g -n 10 --pty /bin/bash(pi2.0)

```

2. 加载编译所需模块:

```

module load gcc
module load openmpi

```

3. 执行以下命令源码编译 openfoam10 (其中 10 为版本号, 用户可根据自身需求改为 7, 8, 9):

```

cd $HOME
mkdir OpenFOAM
cd OpenFOAM
git clone https://github.com/OpenFOAM/OpenFOAM-10.git
git clone https://github.com/OpenFOAM/ThirdParty-10.git
source OpenFOAM-10/etc/bashrc
cd OpenFOAM-10

```

(续下页)

(接上页)

```
./Allwmake -j
sed -i '$a source $HOME/OpenFOAM/OpenFOAM-10/etc/bashrc' $HOME/.bashrc
```

4. 测试是否编译成功:

```
cd $HOME/OpenFOAM/OpenFOAM-10/tutorials/multiphase/interFoam/RAS/DtCHull
./Allrun
```

参考资料

- Openfoam-org 官网
- Openfoam-org github 地址

Osiris

简介

Osiris 是一个在完全相对论性下的，大规模并行计算胞内粒子（PIC）的程序。

软件安装

此软件是开源项目，但相关 git 仓库并不支持公开访问，因此需要申请权限。软件仓库地址：<https://github.com/GoLP-IST/osiris>

1. 将需要的软件分支版本克隆至超算集群中

```
git clone https://github.com/GoLP-IST/osiris.git
```

2. 修改配置

```
vim osiris/config/osiris_sys.linux.gnu
```

检查 OPENMPI_ROOT, SZIP_ROOT, H5_ROOT 这三个路径是否与.bashrc 文件中的一致，如果不一致则改为一致

3. 分维度进行编译

```
cd osiris
chmod 777 configure
./configure -t production -s linux.gnu -d 1/2/3 # (其中, 1/2/
↪3表示维度, 三个维度分别编译)
make
```

4. 确认编译结果

```
ls bin
#可以见到类似osiris-r-2D.e的可执行程序
chmod 755 osiris-r-2D.e
```

OVITO

简介

OVITO 是一款专业实用、功能强大的原子分子可视化及分析软件。界面美观，功能齐全，操作简单，支持超大规模原子快速显示。LAMMPS 的 dump 构型、VASP 的 POSCAR, XDATCAR 等构型均可由 OVITO 查看和编辑。

安装教程

安装 OVITO 的 Python 模块方法：

```
srun -n4 -p cpu --pty /bin/bash #申请计算资源
module load miniconda3
source activate
conda create -n ovito python=3.8.18
conda activate ovito
conda clean --index-cache
conda install --strict-channel-priority -c https://conda.ovito.org -c conda-
  ↳forge ovito==3.9.4
```

运行 OVITO 的 Python 模块方法：

```
module load miniconda3
source activate ovito
ovitos test.py
```

π 集群上使用 OVITO

使用 OVITO 查看原子构型有两种方法：

- 方法一：Studio 可视化平台里直接使用 OVITO（适合快速查看）
 - 方法二：本地电脑 OVITO 远程调用集群上的原子构型文件
- 供测试的原子构型文件：cnt.dump（388 原子的碳纳米管）

```
/lustre/archive/samples/ovito/cnt.dump
```

方法一：Studio 平台里直接使用 OVITO

特点：无需安装 OVITO、按机时计费。

OVITO 需要在 HPC Studio 可视化平台上使用。集群登录节点不支持 OVITO 可视化显示

1. 浏览器打开 <https://studio.hpc.sjtu.edu.cn>
2. 顶栏 Interactive Apps 下拉菜单，选择 ovito
3. 接下来进入资源选择界面。第一个 Desktop Instance Size 默认选择 1core，点击 Launch
4. 等待几秒，甚或更长时间，取决于 small 队列可用资源量。Studio 上的应用以一个 small 队列作业申请资源
5. 待资源申请成功，新的界面下方会出现 Launch ovito，点击进入 OVITO

注意：OVITO 使用完毕后需终止应用资源，否则之前申请的 small 队列会持续计费。有两种退出方法：

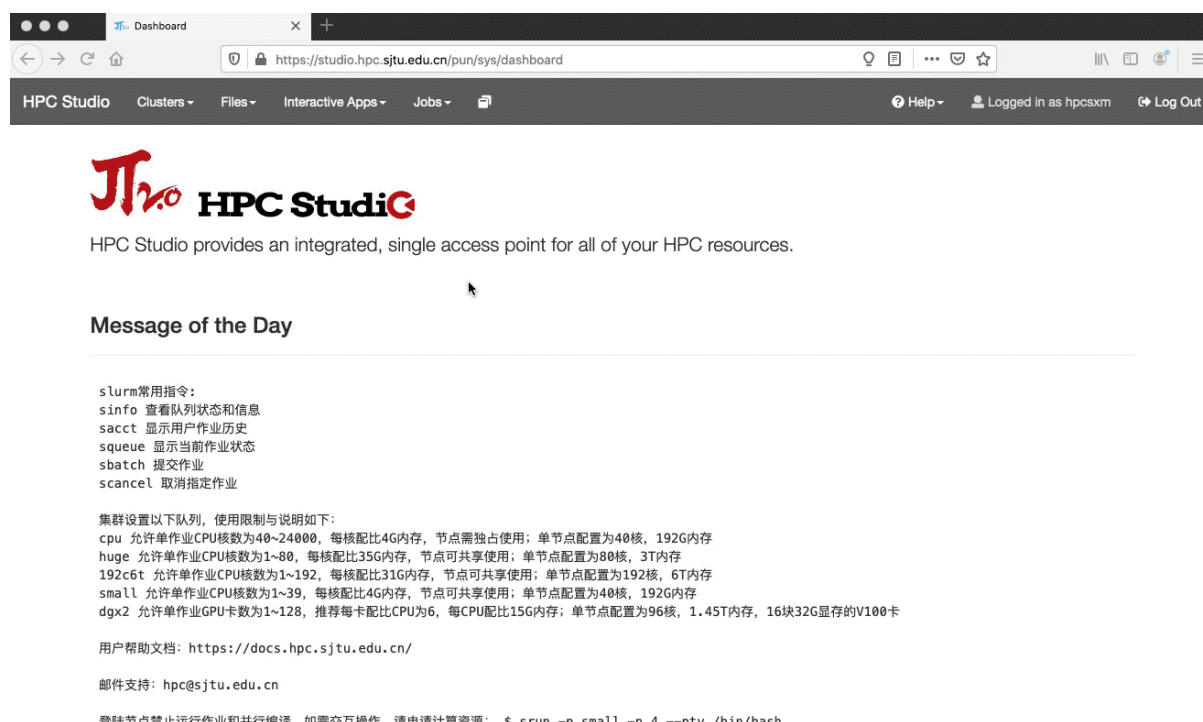
1. 在 Studio 界面上点 Delete 删除该作业。
2. 在集群终端里输入 `squeue` 命令查看作业，并用 `scancel` 终止该作业。

方法二：本地 OVITO 调用远程构型文件

特点：速度快、画质无损失、不计费。

在本地电脑开启 OVITO 软件，点击 OVITO 顶上的 File->Load Remote File 地址栏按下方格式给定绝对路径。注意：集群路径 /lustre 前面无冒号

```
sftp://userXXX@login.hpc.sjtu.edu.cn/lustre/archive/samples/ovito/cnt.dump
```



参考资料

- OVITO 官网 <http://ovito.org>

PSI4

简介

PSI4 provides a wide variety of quantum chemical methods using state-of-the-art numerical methods and algorithms. Several parts of the code feature shared-memory parallelization to run efficiently on multi-core machines (see Sec. Threading). An advanced parser written in Python allows the user input to have a very simple style for routine computations, but it can also automate very complex tasks with ease.

使用 **conda** 在集群上安装 **PSI4**

可以使用以下命令在思源超算和闵行超算上安装 **psi4**。

```
$ module load miniconda3
$ conda create -n p4env psi4 -c psi4
$ source activate p4env
$ pip install pytest==7.0.1
```

测试 **conda** 安装的 **PSI4**

```
$ module load miniconda3
$ source activate p4env
$ psi4 --test
```


结果如下:

```
= 30 passed, 24 skipped, 3347 deselected, 2 xfailed, 31 warnings in 152.14s.
→ (0:02:32) =
```

PySPH

简介

PySPH 是一个用于光滑粒子流体力学 (SPH) 的开源框架。它允许用户用纯 **python** 编写高级代码, 而这些 **Python** 代码将自动转换为高性能 **cython** 或 **opencl** 编译并执行。

PySPH 使用说明

在思源一号上自行安装并使用 **PySPH**

1. 使用 **conda** 创建虚拟环境并安装 **PySPH**:

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda create --name pysph_test
conda activate pysph_test

conda install pip cython numpy
pip install PySPH
```

2. 安装成功后执行以下命令

```
pysph run elliptical_drop
```

3. 然后可在终端得到如下结果:

```
Running example pysph.examples.elliptical_drop.

Information for example: pysph.examples.elliptical_drop
Evolution of a circular patch of incompressible fluid. (60 seconds)

See J. J. Monaghan "Simulating Free Surface Flows with SPH", JCP, 1994, 100,
→ pp
399 - 406

An initially circular patch of fluid is subjected to a velocity profile that
causes it to deform into an ellipse. Incompressibility causes the initially
circular patch to deform into an ellipse such that the area is conserved. An
analytical solution for the locus of the patch is available (exact_solution)

This is a standard test for the formulations for the incompressible SPH
equations.
Elliptical drop :: 5025 particles
Effective viscosity: rho*alpha*h*c/8 = 0.5687500000000001
```

(续下页)

(接上页)

```

Generating output in /dssg/home/acct-hpc/hpcpzz/pysphtest/elliptical_drop_
→output
-----
No of particles:
fluid: 5025
-----
Setup took: 6.09816 secs
100%|████████████████████| 1.1kit | 7.6e-03s [00:39.3<00:0.0 | 0.035s/it]
Run took: 39.31432 secs
Post processing requires matplotlib.

```

在 **pi2.0** 上自行安装并使用 **PySPH**

1. 使用 conda 创建虚拟环境并安装 PySPH:

```

srun -p small -n 4 --pty /bin/bash
module load miniconda3/4.8.2
conda create --name pysph_test
conda activate pysph_test

conda install pip==21.3.1 cython numpy

pip install PySPH

```

2. 此步骤和上文完全相同;

3. 此步骤和上文完全相同;

参考资料

- [PySPH 官网](#)
- [PySPH github 地址](#)

Quantum ESPRESSO

简介

Quantum ESPRESSO 基于密度泛函理论、平面波和赝势（范数守恒和超软）开发，是用于纳米级电子结构计算和材料建模的开源软件包。

根据 GNU 通用公共许可证的条款，全世界的研究人员均可免费使用。

可用的版本

版本	平台	构建方式	模块名
7.2	cpu	spack	quantum-espresso/7.2-intel-2021.4.0 思源一号
7.1	cpu	spack	quantum-espresso/7.1-intel-2021.4.0 思源一号
7.0	cpu	spack	quantum-espresso/7.0-intel-2021.4.0 思源一号
6.7	cpu	spack	quantum-espresso/6.7-intel-2021.4.0 思源一号
6.7	cpu	spack	quantum-espresso/6.7-gcc-11.2.0-openblas-openmpi 思源一号
7.2	cpu	spack	quantum-espresso/7.2-intel-2021.4.0
7.1	cpu	spack	quantum-espresso/7.1-intel-2021.4.0
7.0	cpu	spack	quantum-espresso/7.0-intel-2021.4.0
6.6	cpu	容器	quantum-espresso/6.6
6.7	cpu	源码编译	quantum-espresso/6.7-intel-21.4.0-impi

Spack 安装参考

```
spack install quantum-espresso@7.1%intel@2021.4.0 +libxc ^intel-oneapi-mpi
```

Intel 编译器安装方法:

1. 编译 QE

```
srunk -p 64c512g -n 8 --pty /bin/bash
module load oneapi/2021.4.0
tar -zxvf qe-7.2-ReleasePack.tar.gz && cd qe-7.2
./configure MPIF90=mpiifort CC=mpiicc F90=ifort F77=mpiifort --prefix=/path/
→to/qe.7.2 #用绝对路径指定编译后执行文件位置
make all install -j
make all install
```

2. 编译 EPW, 需要先下载 wannier90 安装包:

```
cp ./wannier90-3.1.0.tar.gz qe-7.2/archive/v3.1.0.tar.gz
cd qe-7.2/EPW
make epw
```

3. 编译完成后将 qe.7.2/bin 添加到环境变量, 使用 QE 之前先调用 oneapi/2021.4.0

算例下载

```
wget https://repository.prace-ri.eu/git/UEABS/ueabs/-/raw/master/quantum_
→espresso/test_cases/small/ausurf.in
wget https://repository.prace-ri.eu/git/UEABS/ueabs/-/raw/master/quantum_
→espresso/test_cases/small/Au.pbe-nd-van.UPF
```

集群上的 **Quantum ESPRESSO**

- 思源一号
- $\pi 2.0$ 集群

一. 思源一号上的 **Quantum ESPRESSO**

基于 **intel** 编译器编译的版本

```
#!/bin/bash
#SBATCH --job-name=1node_qe
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load quantum-espresso/6.7-intel-2021.4.0

export OMP_NUM_THREADS=1
ulimit -s unlimited
ulimit -l unlimited

mpirun pw.x -i ausurf.in
```

使用如下脚本提交作业

```
sbatch qe_intel.slurm
```

运行结果如下所示

```
PWSCF          :   3m50.28s CPU   3m53.80s WALL

tree out
out/
├── ausurf.save
│   ├── Au.pbe-nd-van.UPF
│   ├── charge-density.dat
│   └── data-file-schema.xml
```

(续下页)

(接上页)

```

|   ├── wfc1.dat
|   └── wfc2.dat
└── ausurf.xml

```

基于 **GCC** 编译器编译的版本

```

#!/bin/bash
#SBATCH --job-name=1node_qe_gcc
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load openmpi/4.1.1-gcc-11.2.0
module load quantum-espresso/6.7-gcc-11.2.0-openblas-openmpi

export OMP_NUM_THREADS=1
ulimit -s unlimited
ulimit -l unlimited

mpirun pw.x -i ausurf.in

```

使用如下命令提交作业

```
sbatch qe_gcc.slurm
```

运行结果如下所示:

```

PWSCF           :    5m18.95s CPU    5m26.66s WALL

tree out
out/
├── ausurf.save
│   ├── Au.pbe-nd-van.UPF
│   ├── charge-density.dat
│   ├── data-file-schema.xml
│   ├── wfc1.dat
│   └── wfc2.dat
└── ausurf.xml

1 directory, 6 files

```

二. π 2.0 集群上的 Quantum ESPRESSO

基于 intel2021.4.0 编译器编译的 6.7 版本

```
#!/bin/bash
#SBATCH -J 80cores
#SBATCH -p cpu
#SBATCH -n 80
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

ulimit -s unlimited
ulimit -l unlimited
module load quantum-espresso/6.7-intel-21.4.0-mpi

mpirun pw.x -i ausurf.in
```

使用如下命令提交作业

```
sbatch qe_intel.slurm
```

运行结果如下所示:

```
PWSCF      :      6m42.48s CPU      6m53.24s WALL
```

使用容器部署的版本

在 `cpu` 队列上, 总共使用 80 核 ($n = 80$) `cpu` 队列每个节点配有 40 核, 所以这里使用了 2 个节点。脚本名称可设为 `slurm.test`

```
#!/bin/bash

#SBATCH -J QE_test
#SBATCH -p cpu
#SBATCH -n 80
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

ulimit -s unlimited
ulimit -l unlimited

module load quantum-espresso

srun --mpi=pmi2 pw.x -i ausurf.in
```

使用如下指令提交:

```
$ sbatch slurm.test
```

运行结果如下所示:

```
PWSCF      :   17m37.92s CPU   17m51.67s WALL

tree out
out/
  ├── ausurf.save
  |   ├── Au.pbe-nd-van.UPF
  |   ├── charge-density.dat
  |   ├── data-file-schema.xml
  |   ├── wfc1.dat
  |   └── wfc2.dat
  └── ausurf.xml
```

运行结果

思源一号

quantum-espresso/6.7-intel-2021.4.0				
核数	64	128	192	
CPU time	5m32.13s	3m49.22s	3m41.00s	

quantum-espresso/6.7-gcc-11.2.0-openblas-openmpi				
核数	64	128	192	
CPU time	6m44.78s	5m18.95s	5m31.64s	

π2.0

quantum-espresso/6.7-intel-21.4.0-impi				
核数	40	80	120	
CPU time	9m21.27s	6m42.48s	5m 1.21s	

quantum-espresso/6.6				
核数	40	80	120	
CPU time	19m27.24s	17m39.15s	15m25.99s	

建议

通过分析上述运行结果，我们推荐您使用以下两个版本运行 QE 作业

module load quantum-espRESSO/6.7-intel-2021.4.0	思源一号
module load quantum-espRESSO/6.7-intel-21.4.0-impi	π2.0

参考资料

- [Quantum ESPRESSO 官网](#)

QuickPIC

QuickPIC 是基于 UPIC 框架开发的 3D 并行 (MPI & OpenMP Hybrid) 准静态 PIC 代码。QuickPIC 可以有效地模拟基于等离子体的加速器问题。

运行 QuickPIC 的方式

申请计算节点

```
salloc -p small -n 4 /bin/bash
ssh cas*
```

运行命令如下:

```
module load gcc/9.3.0-gcc-4.8.5
module load openmpi/3.1.5-gcc-9.3.0
export PATH=$PATH:/lustre/opt/contribute/cascadelake/quickpic/install/HDF5/
↪bin:/lustre/opt/contribute/cascadelake/quickpic/packet/QuickPIC-
↪OpenSource/source
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/lustre/opt/contribute/cascadelake/
↪quickpic/install/SZIP/lib:/lustre/opt/contribute/cascadelake/quickpic/
↪install/ZLIB/lib:/lustre/opt/contribute/cascadelake/quickpic/install/HDF5/
↪lib:/lustre/opt/contribute/cascadelake/quickpic/install/json/jsonfortran-
↪gnu-6.10.0/lib

mkdir ~/quickpic
cd ~/quickpic
cp -r /lustre/opt/contribute/cascadelake/quickpic/packet/QuickPIC-
↪OpenSource/source ./
cd source

export OMP_NUM_THREADS=1
mpirun -np 2 ./qp.e
```


SIESTA

简介

SIESTA is both a method and its computer program implementation, to perform efficient electronic structure calculations and ab initio molecular dynamics simulations of molecules and solids. SIESTA's efficiency stems from the use of a basis set of strictly-localized atomic orbitals. A very important feature of the code is that its accuracy and cost can be tuned in a wide range, from quick exploratory calculations to highly accurate simulations matching the quality of other approaches, such as plane-wave methods.

π 集群上的 SIESTA

π 集群系统中已经预装 SIESTA (Intel 版本), 可用以下命令加载:

```
$ module load siesta
```

π 集群上的 Slurm 脚本 slurm.test

在 cpu 队列上, 总共使用 40 核 ($n = 40$) cpu 队列每个节点配有 40 核, 所以这里使用了 1 个节点:

```
#!/bin/bash

#SBATCH -J nechem_test
#SBATCH -p cpu
#SBATCH -n 40
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module load siesta/4.0.1-intel-19.0.4-impi

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
export I_MPI_FABRICS=shm:ofi

ulimit -s unlimited
ulimit -l unlimited

srun --mpi=pmi2 siesta < input.in
```

并使用如下指令提交:

```
$ sbatch slurm.test
```

参考资料

- [SIESTA 官网](#)

STAR-CCM+

简介

Much more than just a CFD solver, STAR-CCM+ is an entire engineering process for solving problems involving flow (of fluids or solids), heat transfer and stress.

STAR-CCM+ 需自行安装

STAR-CCM+ 为商业软件，需要自行购买和安装。建议先和软件厂商工程师充分沟通如下问题：1) 能否安装在普通用户目录下；2) 是否支持浮动 License、是否需要 License 服务器、License 服务器能否安装在虚拟机上；3) 能否提供用于运行作业的 SLURM 作业调度系统脚本

安装完成后，还需在 π 集群上设置以下内容：

1. 清空 `~/.ssh/known_hosts` 文件内容
2. 执行下面两行

```
ssh-keygen -t rsa
ssh-copy-id -i ~/.ssh/id_rsa localhost
```
3. 在 `~/.ssh/config` 中头几行增加如下两行内容：

```
StrictHostKeyChecking no
UserKnownHostsFile=/dev/null
```
4. `chmod 600 ~/.ssh/config`

 π 集群上的 Slurm 脚本 `slurm.test`

在 `cpu` 队列上，总共使用 80 核 ($n = 80$) `cpu` 队列每个节点配有 40 核，所以这里使用了 2 个节点：

```
#!/bin/bash

#SBATCH -J test
#SBATCH -p cpu
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 80
#SBATCH --ntasks-per-node=40

module load intel-parallel-studio/cluster.2019.5-intel-19.0.5

ulimit -s unlimited
```

(续下页)

(接上页)

```
ulimit -l unlimited

cat /dev/null > machinefile
scontrol show hostname $SLURM_JOB_NODELIST > machinefile

starccm+ -power -mpi intel -machinefile './machinefile' -np $SLURM_NTASKS -
→rsh ssh -cpubind -batch run -batch-report YOURsample.sim
```

π 集群上提交作业

```
$ sbatch slurm.test
```

参考资料

- [STAR-CCM+ 网站](#)

SUMO

简介

SUMO, 全称 **Simulation of Urban Mobility**, 是开源、微观、多模态的交通仿真软件, 发展始于 2000 年。它纯粹是微观的, 可以针对每辆车进行单独控制, 因此非常适合交通控制模型的开发。

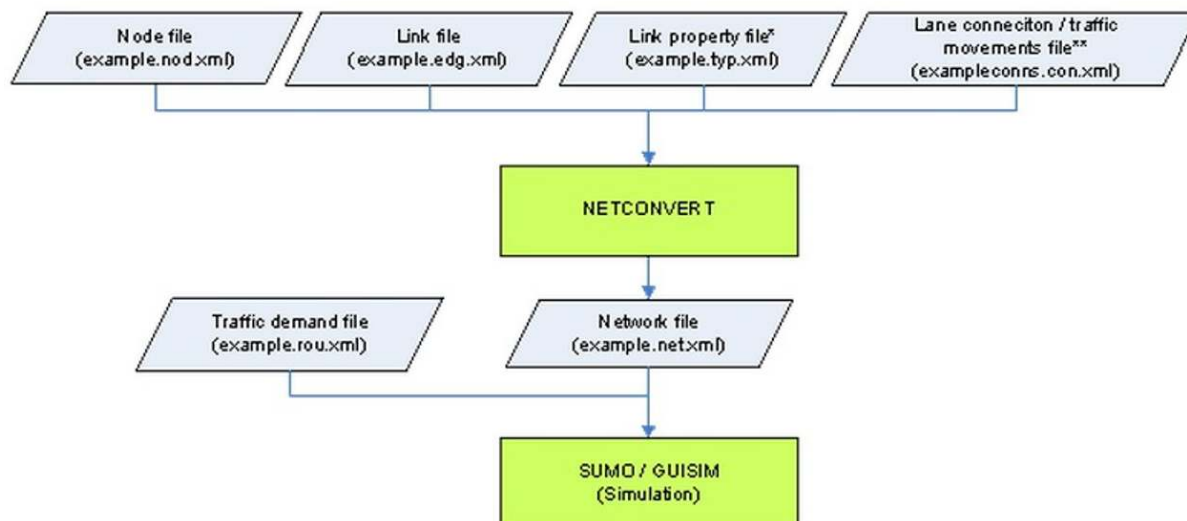
SUMO(GUI) 使用方法

1. 用 pi 集群帐号登录 [HPC Studio](#) 平台;
2. 在网站通过 **Interactive Apps->Desktop->Launch** 进入桌面 (注意使用 GPU 桌面);
3. 打开终端, 通过以下命令来调用软件:

```
module load sumo/1.10.0-sumo
sumo-gui
```

4. 从以下仿真流程图可知, 每一次模拟都需要 **rou.xml** 和 **net.xml** 两个参数配置文件。其中 **rou.xml** 用来表述交通需求, **net.xml** 用来表述道路信息。而道路信息又由下面四个文件通过 **netconvert** 命令生成:

```
nod.xml : 用来描述节点信息
edg.xml : 用来描述边的信息
typ.xml : 用来描述预定义的边的类型 (类似于做一个封装)
con.xml : 用来描述边到边的合并形式
```



具体案例

1. 编写 exa.nod.xml 文件:

```

<nodes> <!-- The opening tag -->
<node id="0" x="0.0" y="0.0" type="traffic_light"/> <!-- def. of node "0" -->
<!-->
<node id="1" x="-500.0" y="0.0" type="priority"/> <!-- def. of node "1" -->
<node id="2" x="+500.0" y="0.0" type="priority"/> <!-- def. of node "2" -->
<node id="3" x="0.0" y="-500.0" type="priority"/> <!-- def. of node "3" -->
<node id="4" x="0.0" y="+500.0" type="priority"/> <!-- def. of node "4" -->
<node id="m1" x="-250.0" y="0.0" type="priority"/> <!-- def. of node "m1" -->
<!-->
<node id="m2" x="+250.0" y="0.0" type="priority"/> <!-- def. of node "m2" -->
<!-->
<node id="m3" x="0.0" y="-250.0" type="priority"/> <!-- def. of node "m3" -->
<!-->
<node id="m4" x="0.0" y="+250.0" type="priority"/> <!-- def. of node "m4" -->
<!-->
</nodes> <!-- The closing tag -->

```

2. 编写 exa.edg.xml 文件:

```

<edges>
<edge id="1fi" from="1" to="m1" priority="2" numLanes="2" speed="11.11"/>
<edge id="1si" from="m1" to="0" priority="3" numLanes="3" speed="13.89"/>
<edge id="1o" from="0" to="1" priority="1" numLanes="1" speed="11.11"/>
<edge id="2fi" from="2" to="m2" priority="2" numLanes="2" speed="11.11"/>
<edge id="2si" from="m2" to="0" priority="3" numLanes="3" speed="13.89"/>
<edge id="2o" from="0" to="2" priority="1" numLanes="1" speed="11.11"/>
<edge id="3fi" from="3" to="m3" priority="2" numLanes="2" speed="11.11"/>
<edge id="3si" from="m3" to="0" priority="3" numLanes="3" speed="13.89"/>
<edge id="3o" from="0" to="3" priority="1" numLanes="1" speed="11.11"/>
<edge id="4fi" from="4" to="m4" priority="2" numLanes="2" speed="11.11"/>

```

(续下页)

(接上页)

```
<edge id="4si" from="m4" to="0" priority="3" numLanes="3" speed="13.89"/>
<edge id="4o" from="0" to="4" priority="1" numLanes="1" speed="11.11"/>
</edges>
```

3. 编写 `exa.typ.xml` 文件。这里就不详细描述了，因为就类似于建立一个 `type` 类供 `edge` 使用。

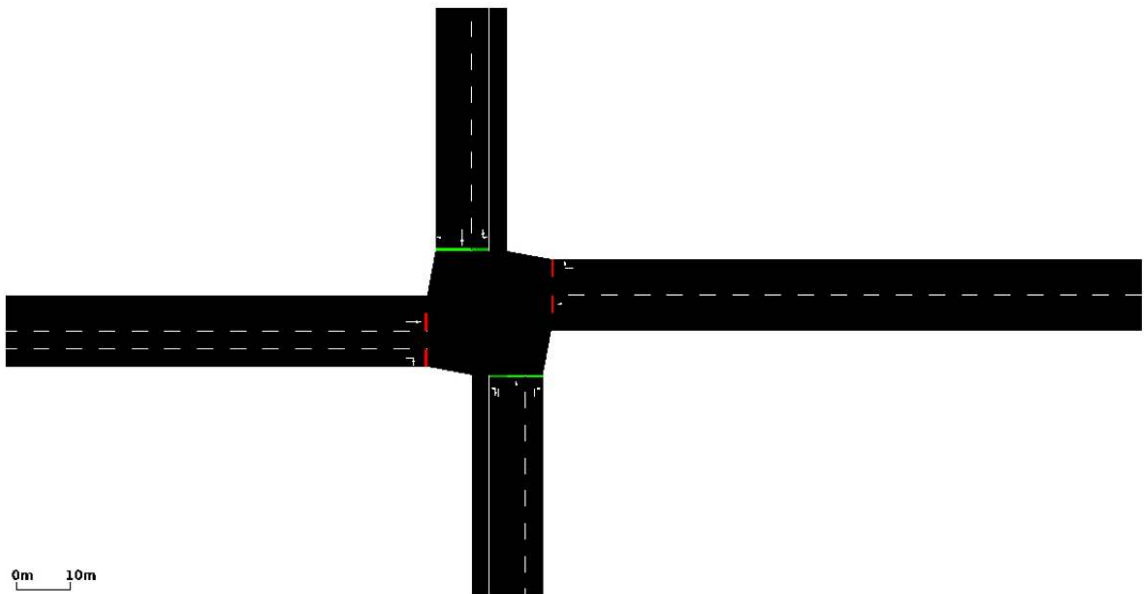
4. 编写 `exa.con.xml` 文件：

```
<connections>
<connection from="1si" to="3o" fromLane="0" toLane="0"/>
<connection from="1si" to="2o" fromLane="2" toLane="0"/>
<connection from="2si" to="4o" fromLane="0" toLane="0"/>
<connection from="2si" to="1o" fromLane="2" toLane="0"/>
</connections>
```

5. 使用 `netconvert` 命令生成 `exa.net.xml`：

```
netconvert --node-files=exa.nod.xml --edge-files=exa.edg.xml \ --
--connection-files=exa.con.xml --type-files=exa.typ.xml \ --output-
file=exa.net.xml
```

如果没有 `con.xml` 或者 `typ.xml` 则忽略对应的参数。使用 `sumo-gui` 查看 `net` 结果如下：



6. 编写 `exa.rou.xml` 文件：

```
<routes>
<vType accel="1.0" decel="5.0" id="ACar" length="2.0" maxSpeed="10.0" sigma=
-->"1.0" />
<vType accel="0.8" decel="5.0" id="BCar" length="2.0" maxSpeed="15.0" sigma=
-->"1.0" />
```

(续下页)

(接上页)

```
<route id="route_ns" edges="4fi 4si 3o"/>
<route id="route_we" edges="1fi 1si 2o"/>
<flow depart="1" id="flow_n_s" route="route_ns" type="ACar" begin="0" end=
→"3600" probability="0.1" />
<flow depart="1" id="flow_w_e" route="route_we" type="BCar" begin="0" end=
→"3600" probability="0.3" />
</routes>
```

7. 编写 exa.sumocfg 文件:

```
<configuration>
<input>
  <net-file value="exa.net.xml"/>
  <route-files value="exa.rou.xml"/>
</input>
<time>
  <begin value="0"/>
  <end value="10000"/>
</time>
</configuration>
```

8. 在命令行中执行以下命令:

```
sumo-gui -c exa.sumocfg
```

或者直接打开 *sumo-gui* 选择 *open simulation*，打开 *exa.sumocfg* 文件即可。对于较为复杂的情况，建议直接使用 *netedit* 软件以图形界面的方式生成 *net.xml* 道路信息文件。

参考资料

- SUMO 知乎学习笔记
- SUMO 官网
- SUMO 参考视频教程

VASP

简介

VASP 全称 Vienna Ab-initio Simulation Package，是维也纳大学 Hafner 小组开发的进行电子结构计算和量子力学-分子动力学模拟软件包。它是目前材料模拟和计算物质科学研究中最流行的商用软件之一。

注意

1. VASP 使用需要得到 VASP 官方授权，请自行购买 VASP license 许可，下载和安装。

2. 若课题组已购买 VASP，可发送邮件提供课题组的 VASP license 证明，并注明需要使用的集群名称以及集群上具体的 VASP 模块名称，我们将添加该 VASP module 的使用权限。

3. 如果需要编译第三方插件或者需要修改源码版本的 VASP，请提前参考 VASP 官网或者第三方插件官网确认需要安装的 VASP 版本是否支持。

本文档将介绍如何使用集群上已部署的 VASP，以及如何自行编译 VASP。

集群上可用的 VASP 版本

版本	平台	构建方式	集群	模块名
5.4.4		源码	思源一号	vasp/5.4.4-intel-2021.4.0
6.2.1		源码	思源一号	vasp/6.2.1-intel-2021.4.0-cuda-11.5.0
6.3.2		源码	思源一号	vasp/6.3.2-vtst-intel-2021.4.0
5.4.4		源码	Pi 2.0	vasp/5.4.4-intel-2021.4.0
6.3.2		源码	Pi 2.0	vasp/6.3.2-intel-2021.4.0

思源一号 VASP

下面 slurm 脚本以 vasp 6.3.2 为例。若使用其他模块，比如 vasp 5.4.4，请将 module load 那行换成 module load vasp/5.4.4-intel-2021.4.0

```
#!/bin/bash

#SBATCH -J vasp
#SBATCH -p 64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load vasp/6.3.2-intel-2021.4.0

ulimit -s unlimited
ulimit -l unlimited

mpirun -np $SLURM_NPROCS vasp_std
```

π2.0 VASP

下面 slurm 脚本以 vasp 5.4.4 为例：

```
#!/bin/bash

#SBATCH -J vasp
#SBATCH -p cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load vasp/5.4.4-intel-2021.4.0

ulimit -s unlimited
ulimit -l unlimited

mpirun -np $SLURM_NPROCS vasp_std
```

自行编译 VASP

VASP 在集群上使用 intel 套件自行编译十分容易，下面介绍 CPU 版本的安装和使用方法。

1. 先申请计算节点，然后加载 intel 套件

```
srun -p 64c512g -n 4 --pty /bin/bash      # 思源集群申请计算节点
srun -p cpu -n 4 --pty /bin/bash          # Pi2.0集群申请计算节点

module load oneapi/2021.4.0               # 加载intel套件
```

1. 解压缩 VASP 安装包，进入 vasp.x.x.x 文件夹（可看到 arch, src 等文件夹）

```
cp arch/makefile.include.linux_intel makefile.include
```

3. 输入 make 开始编译，预计十分钟左右完成

```
make
```

请注意，为了避免编译出错，推荐直接使用 **make**，不要添加 -jN (若一定要使用，请使用完整的命令：make DEPS=1 -jN)

编译完成后，bin 文件夹里将出现三个绿色的文件：vasp_std, vasp_gam, vasp_ncl

可将 vasp_std 复制到 \$HOME/bin 里，后续可以直接调用：

```
mkdir -p ~/bin      # 若 home 下未曾建过
→bin, 则新建一个；若已有，请略过此句
cp bin/vasp_std ~/bin
```


4. 使用

```
#!/bin/bash

#SBATCH -J vasp
#SBATCH -p 64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load oneapi/2021.4.0

ulimit -s unlimited
ulimit -l unlimited

mpirun -np $SLURM_NPROCS ~/bin/vasp_std
```

VASP-GPU 版本编译安装

由于 VASP 为商业软件，需要用户自行申请 license、在官网自行下载源码包。下面介绍如何在思源一号上的 a100 节点上，编译安装 GPU 版本 VASP，本文以 6.3.0 为例编译 nvhpc+acc 版本，其他版本请参考 vasp 官网。

1. 先申请计算节点，然后加载编译环境

```
srun -n 16 --gres=gpu:1 -p a100 --pty /bin/bash # 申请计算节点

module load nvhpc/23.3-gcc-11.2.0
module load oneapi/2021.4.0
module unload intel-oneapi-mpi/2021.4.0
module load gcc/11.2.0 cuda/11.8.0
```

2. 解压缩 VASP 安装包，进入 vasp.x.x.x 文件夹，可看到 arch, src 等文件夹。

```
cp arch/makefile.include.nvhpc_acc makefile.include
# 修改makefile.include文件
# 删除或注释 BLAS and LAPACK, scaLAPACK, FFTW, 新增 MKL 设置
# Intel MKL (FFTW, BLAS, LAPACK, and scaLAPACK
MKLROOT      ?= /dssg/opt/icelake/linux-centos8-icelake/gcc-8.5.0/intel-
→oneapi-mkl-2021.4.0-r7h6alnulyzgb6iqvxhovmwrajbvbwqxf/mkl/2021.4.0/
LLIBS        += -Mkl -L${MKLROOT}/lib/intel64 -lmkl_scalapack_lp64 -lmkl_
→blacs_openmpi_lp64
INCS         += -I$(MKLROOT)/include/fftw
```

3. 输入 make 开始编译

```
make
```

请注意，为了避免编译出错，推荐直接使用 make，不要添加 -jN (若一定要使用，

请使用完整的命令：`make DEPS=1 -jN`）

编译完成后，`bin` 文件夹里将出现三个绿色的文件：`vasp_std`, `vasp_gam`, `vasp_ncl`

可将 `vasp_std` 复制到 `home/bin` 里，后续可以直接调用：

```
mkdir -p ~/bin          # 若 home 下未曾建过
↪bin, 则新建一个；若已有，请略过此句
cp bin/vasp_std ~/bin
```

4. 作业脚本

```
#!/bin/bash

#SBATCH -J vasp-gpu
#SBATCH -p a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=16
#SBATCH --gres=gpu:1

module load nvhpc/23.3-gcc-11.2.0
module load oneapi/2021.4.0
module unload intel-oneapi-mpi/2021.4.0
module load gcc/11.2.0 cuda/11.8.0
ulimit -s unlimited
ulimit -l unlimited

mpirun -np 1 ~/bin/vasp_std
```

VASP 算例

以 64 原子的 Si AIMD 熔化为例，本示例相关说明：

1. VASP 运行需要最基本的 INCAR, POSCAR, POTCAR, KPOINTS 四个文件。全部文件已放置于思源一号共享文件夹：

```
/dssg/share/sample/vasp
```

2. VASP 算例运行方法：

```
cp -r /dssg/share/sample/vasp ~
cd vasp
sbatch run.slurm
```

3. 下面是该示例的 INCAR 文件内容：

```
SYSTEM = cd Si

! ab initio
ISMEAR = 0          ! Gaussian smearing
SIGMA  = 0.1        ! smearing in eV
```

(续下页)

(接上页)

```

LREAL = Auto      ! projection operators in real space

ALGO = VeryFast ! RMM-DIIS for electronic relaxation
PREC = Low       ! precision
ISYM = 0         ! no symmetry imposed

! MD
IBRION = 0       ! MD (treat ionic dgr of freedom)
NSW = 60         ! no of ionic steps
POTIM = 3.0      ! MD time step in fs

MDALGO = 2       ! Nosé-Hoover thermostat
SMASS = 1.0      ! Nosé mass

TEBEG = 2000     ! temperature at beginning
TEEND = 2000     ! temperature at end
ISIF = 2         ! update positions; cell shape and volume fixed

NCORE = 4

```

参考资料

- [VASP 官网](#)

VMD

简介

Visual Molecular Dynamics is a molecular modelling and visualization computer program. VMD is developed as mainly a tool to view and analyze the results of molecular dynamics simulations. It also includes tools for working with volumetric data, sequence data, and arbitrary graphics objects.

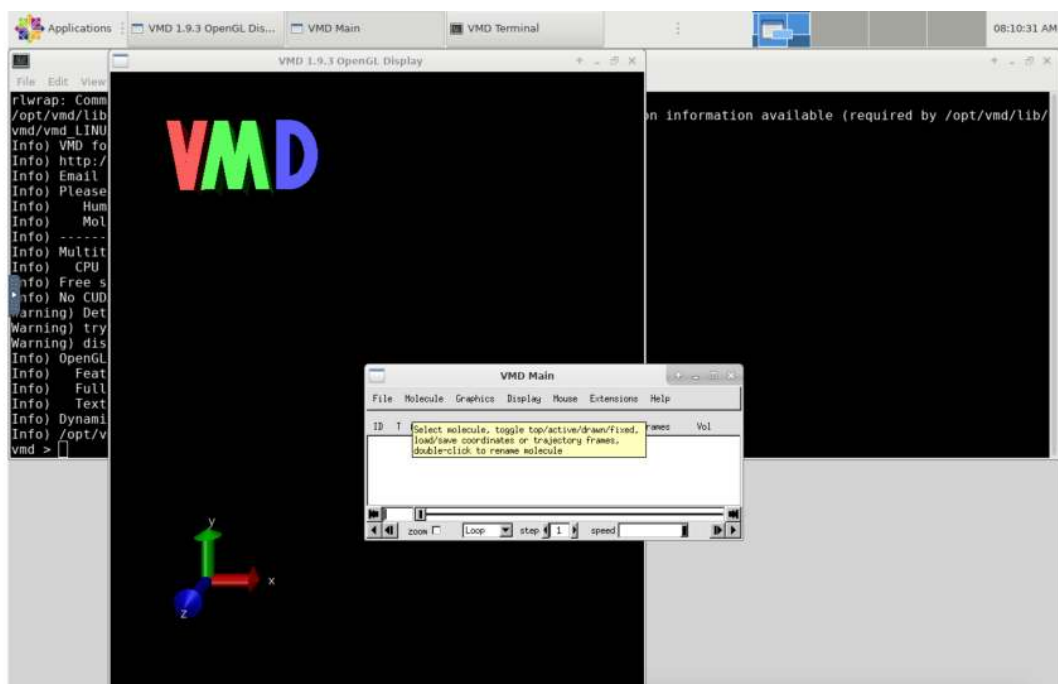
π 集群上的 VMD

集群中已预置了编译优化的容器镜像，通过调用该镜像即可运行 VMD，无需单独安装，目前版本为 vmd-1.9.3。该容器文件位于 /lustre/share/img/vmd-1.9.3.simg

使用 HPC Studio 启动可视化界面

参照可视化平台，登录 HPC Studio，在顶栏选择 VMD：





参考资料

- VMD
- Singularity 文档

WRF

简介

WRF(Weather Research and Forecasting Model) 模式是有美国环境预报中心 (NCEP), 美国国家大气研究中心 (NCAR) 以及多个大学、研究所和业务部门联合研发的一种统一的中尺度天气预报模式。WRF 模式适用范围很广, 从中小尺度到全球尺度的数值预报和模拟都有广泛的应用。

WPS 是预处理 WRF 运行数据的工具。

可用版本

版本	平台	构建方式	模块名
4.2.1		源码	wrf/4.2.1-oneapi-2021.4.0 思源一号
4.5.1		源码	wrf/4.5.1-oneapi-2021.4.0 思源一号
4.2		源码	wps/4.2-oneapi-2021.4.0 思源一号
4.5		源码	wps/4.5-oneapi-2021.4.0 思源一号
4.3.1		源码	wrf/4.3.1-intel-2021.4.0 Pi2.0
4.5.1		源码	wrf/4.5.1-intel-2021.4.0 Pi2.0
4.4		源码	wps/4.4-intel-2021.4.0 Pi2.0
4.5		源码	wps/4.5-intel-2021.4.0 Pi2.0
4.5.1		源码	wrf/4.5.1-gcc-10.3.0-openmpi ARM
4.5		源码	wps/4.5-gcc-10.3.1 ARM

算例位置

思源一号 : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/
→wrf-4.2/wrf_data
π2.0 : /lustre/opt/contribute/cascadelake/wrf_cmaq/wrf_data
ARM : 同π2.0

算例目录

```
[hpc@node234 wrf-4.2]$ tree wrf_data/  
wrf_data/  
├── fnl_20161006_00_00.grib2  
├── fnl_20161006_06_00.grib2  
├── fnl_20161006_12_00.grib2  
├── fnl_20161006_18_00.grib2  
├── fnl_20161007_00_00.grib2  
├── fnl_20161007_06_00.grib2  
├── fnl_20161007_12_00.grib2  
├── fnl_20161007_18_00.grib2  
└── fnl_20161008_00_00.grib2
```

思源一号和 π2.0 两个集群上的数据均是模拟 2016 年 10 月 06 日 00 点至 2016 年 10 月 08 日 0 点的气象数据

geog_data_path 的位置

```
思源一号 : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/
→geo/geog
π2.0      : /lustre/opt/contribute/cascadelake/wrf_cmaq/geo
```

集群上的 WRF 和 WPS

- 思源一号上的 WRF 和 WPS
- π2.0 上的 WRF 和 WPS

思源一号上的 WRF 和 WPS

自定义编译 WRF 和 WPS

思源一号上已部署所依赖的库及版本

```
hdf5-1.10.5
libpng-1.6.37
netcdf-c-4.8.0
netcdf-fortran-4.5.3
zlib-1.2.13
jasper-1.900.1
```

自定义编译 WRF

申请计算节点用于编译

```
srunk -n 20 -p 64c512g --pty /bin/bash
```

编译 WRF 前导入所需的环境变量

```
module load oneapi
export CC=icc
export FC=ifort
export F90=ifort
export CXX=icpc
export CPP="icpc -E"
export DIR=/dssg/share/sample/wrf/WRF_4.1.1_Intel/wrflibs
export PATH=$DIR/netcdf/bin:$DIR/hdf5-1.10.5/bin:$DIR/libpng-1.6.37/bin:
→$DIR/zlib-1.2.13/bin:$PATH
export LD_LIBRARY_PATH=$DIR/netcdf/lib:$DIR/hdf5-1.10.5/lib:$DIR/libpng-1.6.
→37/lib64:$DIR/zlib-1.2.13/lib:$LD_LIBRARY_PATH
export NETCDF=$DIR/netcdf
export HDF5=$DIR/hdf5-1.10.5
```

拷贝 WRF 源码并进行 configure 配置选择

```
cp /dssg/share/sample/wrf/WRF_4.1.1_Intel/src/v4.2.2.tar.gz .
tar xvf v4.2.2.tar.gz
cd WRF-4.2.2/
./configure
```

可根据所需选择相应的参数，使用 **intel** 编译器编译 WRF，可以选择 15，可以多节点并行运行。

Please **select** from among the following Linux x86_64 options:

1. (serial)	2. (smpar)	3. (dmpar)	4. (dm+sm)	PGI (pgf90/gcc)
2. (serial)	6. (smpar)	7. (dmpar)	8. (dm+sm)	PGI (pgf90/pgcc): SGI
→MPT				
3. (serial)	10. (smpar)	11. (dmpar)	12. (dm+sm)	PGI (pgf90/gcc): PGI
→accelerator				
1. (serial)	14. (smpar)	15. (dmpar)	16. (dm+sm)	INTEL (ifort/icc)
			1. (dm+sm)	INTEL (ifort/icc):
→Xeon Phi (MIC architecture)				
2. (serial)	19. (smpar)	20. (dmpar)	21. (dm+sm)	INTEL (ifort/icc):
→Xeon (SNB with AVX mods)				
3. (serial)	23. (smpar)	24. (dmpar)	25. (dm+sm)	INTEL (ifort/icc):
→SGI MPT				
4. (serial)	27. (smpar)	28. (dmpar)	29. (dm+sm)	INTEL (ifort/icc):
→IBM POE				
5. (serial)		31. (dmpar)		PATHSCALE (pathf90/
→pathcc)				
6. (serial)	33. (smpar)	34. (dmpar)	35. (dm+sm)	GNU (gfortran/gcc)
7. (serial)	37. (smpar)	38. (dmpar)	39. (dm+sm)	IBM (xlf90_r/cc_r)
8. (serial)	41. (smpar)	42. (dmpar)	43. (dm+sm)	PGI (ftn/gcc): Cray
→XC CLE				
9. (serial)	45. (smpar)	46. (dmpar)	47. (dm+sm)	CRAY CCE (ftn
→\$(NOOMP)/cc): Cray XE and XC				
10. (serial)	49. (smpar)	50. (dmpar)	51. (dm+sm)	INTEL (ftn/icc): Cray
→XC				
11. (serial)	53. (smpar)	54. (dmpar)	55. (dm+sm)	PGI (pgf90/pgcc)
12. (serial)	57. (smpar)	58. (dmpar)	59. (dm+sm)	PGI (pgf90/pgcc): -
→f90=pgf90				
13. (serial)	61. (smpar)	62. (dmpar)	63. (dm+sm)	PGI (pgf90/pgcc): -
→f90=pgf90				
14. (serial)	65. (smpar)	66. (dmpar)	67. (dm+sm)	INTEL (ifort/icc):
→HSW/BDW				
15. (serial)	69. (smpar)	70. (dmpar)	71. (dm+sm)	INTEL (ifort/icc):
→KNL MIC				
16. (serial)	73. (smpar)	74. (dmpar)	75. (dm+sm)	FUJITSU (frtpx/
→fccpx): FX10/FX100 SPARC64 IXfx/Xlfx				

修改 **configure.wrf**

```
DM_FC          = mpiifort
DM_CC          = mpiicc
```


编译 em_real

```
./compile em_real 2>&1 | tee -a compile.log
```

编译完成输出

```

--->                                Executables successfully built                                <---

-rwxrwxr-x 1 hpcxdy hpcxdy 48398000 Oct 22 10:19 main/ndown.exe
-rwxrwxr-x 1 hpcxdy hpcxdy 48388496 Oct 22 10:19 main/real.exe
-rwxrwxr-x 1 hpcxdy hpcxdy 47734360 Oct 22 10:19 main/tc.exe
-rwxrwxr-x 1 hpcxdy hpcxdy 52544272 Oct 22 10:18 main/wrf.exe

=====

```

自定义编译 WPS

导入如下环境变量

```
export WRF_DIR=../WRF-4.2.2/
export JASPERLIB=$DIR/jasper-1.900.1/lib/
export JASPERINC=$DIR/jasper-1.900.1/include/
```

拷贝 WPS 源码

```
cp /dssg/share/sample/wrf/src/v4.2.tar.gz .
tar xvf v4.2.tar.gz
cd WPS-4.2/
./configure
```

根据个人所需选择所需版本，选择 20 多节点并行运行。

Please **select** from among the following supported platforms.

1. Linux x86_64, gfortran (serial)
2. Linux x86_64, gfortran (serial_NO_GRIB2)
3. Linux x86_64, gfortran (dmpar)
4. Linux x86_64, gfortran (dmpar_NO_GRIB2)
5. Linux x86_64, PGI compiler (serial)
6. Linux x86_64, PGI compiler (serial_NO_GRIB2)
7. Linux x86_64, PGI compiler (dmpar)
8. Linux x86_64, PGI compiler (dmpar_NO_GRIB2)
9. Linux x86_64, PGI compiler, SGI MPT (serial)
10. Linux x86_64, PGI compiler, SGI MPT (serial_NO_GRIB2)
11. Linux x86_64, PGI compiler, SGI MPT (dmpar)
12. Linux x86_64, PGI compiler, SGI MPT (dmpar_NO_GRIB2)
13. Linux x86_64, IA64 and Opteron (serial)
14. Linux x86_64, IA64 and Opteron (serial_NO_GRIB2)
15. Linux x86_64, IA64 and Opteron (dmpar)
16. Linux x86_64, IA64 and Opteron (dmpar_NO_GRIB2)

(续下页)

(接上页)

```

17. Linux x86_64, Intel compiler      (serial)
18. Linux x86_64, Intel compiler      (serial_NO_GRIB2)
19. Linux x86_64, Intel compiler      (dmpar)
20. Linux x86_64, Intel compiler      (dmpar_NO_GRIB2)
21. Linux x86_64, Intel compiler, SGI MPT      (serial)
22. Linux x86_64, Intel compiler, SGI MPT      (serial_NO_GRIB2)
23. Linux x86_64, Intel compiler, SGI MPT      (dmpar)
24. Linux x86_64, Intel compiler, SGI MPT      (dmpar_NO_GRIB2)
25. Linux x86_64, Intel compiler, IBM POE      (serial)
26. Linux x86_64, Intel compiler, IBM POE      (serial_NO_GRIB2)
27. Linux x86_64, Intel compiler, IBM POE      (dmpar)
28. Linux x86_64, Intel compiler, IBM POE      (dmpar_NO_GRIB2)
29. Linux x86_64 g95 compiler         (serial)
30. Linux x86_64 g95 compiler         (serial_NO_GRIB2)
31. Linux x86_64 g95 compiler         (dmpar)
32. Linux x86_64 g95 compiler         (dmpar_NO_GRIB2)
33. Cray XE/XC CLE/Linux x86_64, Cray compiler      (serial)
34. Cray XE/XC CLE/Linux x86_64, Cray compiler      (serial_NO_GRIB2)
35. Cray XE/XC CLE/Linux x86_64, Cray compiler      (dmpar)
36. Cray XE/XC CLE/Linux x86_64, Cray compiler      (dmpar_NO_GRIB2)
37. Cray XC CLE/Linux x86_64, Intel compiler      (serial)
38. Cray XC CLE/Linux x86_64, Intel compiler      (serial_NO_GRIB2)
39. Cray XC CLE/Linux x86_64, Intel compiler      (dmpar)
40. Cray XC CLE/Linux x86_64, Intel compiler      (dmpar_NO_GRIB2)

```

修改 configure.wps

```

DM_FC      = mpiifort
DM_CC      = mpiicc

```

编译安装

```
./compile
```

思源一号上使用预编译的 WRF 和 WPS

先用 WPS 处理数据

1. 由于 WPS 处理数据需要复杂的文件依赖关系，可先拷贝 WPS 目录中的文件到本地

```

mkdir ~/data && cd ~/data
mkdir WRF && cd WRF
cp -r /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/wrf-
↪4.2/WPS-4.2 ./

```

2. 拷贝数据到 WPS 目录中进行数据处理

```
cd WPS-4.2
cp -r /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/wrf-
→4.2/wrf_data/* ./
```

3. namelist.wps 文件内容设置如下:

```
&share
wrf_core = 'ARW',
max_dom = 1,
start_date = '2016-10-06_00:00:00'
end_date   = '2016-10-08_00:00:00'
interval_seconds = 21600
io_form_geogrid = 2,
/

&geogrid
parent_id      = 1,
parent_grid_ratio = 1,
i_parent_start = 1,
j_parent_start = 1,
e_we           = 515,
e_sn           = 515,
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! The default datasets used to produce the MAXSNOALB and ALBEDO12M
! fields have changed in WPS v4.0. These fields are now interpolated
! from MODIS-based datasets.
!
! To match the output given by the default namelist.wps in WPS v3.9.1,
! the following setting for geog_data_res may be used:
!
! geog_data_res = 'maxsnowalb_ncep+albedo_ncep+default',      'maxsnowalb_
→ncep+albedo_ncep+default',
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
geog_data_res = 'default','default',
dx = 12000,
dy = 12000,
map_proj = 'lambert',
ref_lat  = 31.00,
ref_lon  = 120.00,
ref_x    = 351
ref_y    = 208
truelat1 = 30.0,
truelat2 = 60.0,
stand_lon = 120.0,
geog_data_path = '/dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/
→wrf_cmaq/geo/geog/'
/
```

(续下页)

(接上页)

```
&ungrib
  out_format = 'WPS',
  prefix = 'FILE',
/

&metgrid
  fg_name = 'FILE'
  io_form_metgrid = 2,
/
```

4. 运行 `geogrid.exe` 程序定义模型投影、区域范围，嵌套关系，对地表参数进行插值。

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load oneapi
module load wps
geogrid.exe
```

5. 根据模拟时期选择文件

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load oneapi
module load wps
link_grib.csh fnl_2016100*
cp ungrib/Variable_Tables/Vtable.GFS Vtable
```

6. 从 `grib` 数据中提取所需要的气象参数

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load oneapi
module load wps
ungrib.exe
```

7. 将气象参数插值到模拟区域

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load oneapi
module load wps
metgrid.exe
```

WRF 运行

1. 由于 WRF 运行数据需要复杂的文件依赖关系，可先拷贝 WRF 目录中必要的文件到本地

```
cd ~/data
cd WRF
mkdir WRF-4.2.1 && cd WRF-4.2.1
cp -r /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/wrf-
↳4.2/WRF-4.2.1/run/* ./
```

2. 拷贝 WPS 生成的 met 文件到 WRF-4.2.1 目录

```
cp -r ../WPS-4.2/met_em.d01.2016-10-0* ./
```

3. namelist.input 文件内容设置如下，参数需要与 wps 的 namelist.wps 参数一致：

```
&time_control
run_days           = 2,
run_hours          = 0,
run_minutes        = 0,
run_seconds        = 0,
start_year         = 2016,
start_month        = 10,
start_day          = 06,
start_hour         = 00,
end_year           = 2016,
end_month          = 10,
end_day            = 08,
end_hour           = 00,
interval_seconds   = 21600
input_from_file     = .true.,.true.,
history_interval    = 60, 60,
frames_per_outfile  = 12, 12,
restart            = .false.,
restart_interval    = 5000,
io_form_history     = 2
```

(续下页)

(接上页)

```

io_form_restart           = 2
io_form_input             = 2
io_form_boundary          = 2
/

&domains
time_step                = 60,
time_step_fract_num       = 0,
time_step_fract_den       = 1,
max_dom                  = 1,
e_we                    = 515,    112,
e_sn                    = 515,    97,
e_vert                  = 33,    33,
p_top_requested          = 5000,
num_metgrid_levels       = 32,
num_metgrid_soil_levels  = 4,
dx                      = 12000,
dy                      = 12000,
grid_id                 = 1,    2,
parent_id               = 0,    1,
i_parent_start          = 1,    31,
j_parent_start          = 1,    17,
parent_grid_ratio        = 1,    3,
parent_time_step_ratio   = 1,    3,
feedback                = 1,
smooth_option           = 0
/

&physics
physics_suite            = 'tropical'
mp_physics               = 6,    -1,
cu_physics               = 16,    -1,
ra_lw_physics            = 4,    -1,
ra_sw_physics            = 4,    -1,
bl_pbl_physics           = 8,    8,
sf_sfclay_physics        = 1,    1,
sf_surface_physics       = 2,    -1,
radt                    = 12,    30,
bldt                    = 0,    0,
cudt                    = 5,    5,
icloud                   = 1,
num_land_cat             = 21,
sf_urban_physics         = 0,    0,    0,
/

&fdda
/

&dynamics

```

(续下页)

(接上页)

```

hybrid_opt           = 2,
w_damping            = 0,
diff_opt             = 1,      1,
km_opt              = 4,      4,
diff_6th_opt        = 0,      0,
diff_6th_factor     = 0.12,   0.12,
base_temp           = 290.,
damp_opt            = 3,
zdamp               = 5000.,  5000.,
dampcoef            = 0.2,    0.2,
khdif               = 0,      0,
kvdif               = 0,      0,
non_hydrostatic     = .true., .true.,
moist_adv_opt       = 1,      1,
scalar_adv_opt      = 1,      1,
gwd_opt             = 0,      1,
/

&bdy_control
spec_bdy_width      = 5,
specified           = .true.
/

&grib2
/

&namelist_quilt
nio_tasks_per_group = 0,
nio_groups          = 1,
/

```

4. 运行 real.exe 程序，脚本如下：

```

#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load wrf
ulimit -s unlimited
real.exe

```

5. 运行 wrf.exe 程序，脚本如下，该部分是最终也是最耗时的执行程序。

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=test
#SBATCH --partition=64c512g
#SBATCH -N 4
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load wrf
ulimit -s unlimited
mpirun wrf.exe
```

π2.0 上的 WRF 和 WPS

π2.0 上先用 WPS 处理数据

1. 由于 WPS 处理数据需要复杂的文件依赖关系，可先拷贝 WPS 目录中的文件到本地

```
mkdir ~/data && cd ~/data
mkdir WRF && cd WRF
cp -r /lustre/opt/contribute/cascadelake/wrf_cmaq/packet_1/WPS-4.3.1 ./
```

- ## 2. 拷贝数据到 WPS 目录中进行数据处理

```
cd WPS-4.3.1
cp -r /lustre/opt/contribute/cascadelake/wrf_cmaq/wrf_data/* ./
```

3. namelist.wps 文件内容设置如下:

```
&share
wrf_core = 'ARW',
max_dom = 1,
start_date = '2016-10-06_00:00:00'
end_date   = '2016-10-08_00:00:00'
interval_seconds = 21600
io_form_geogrid = 2,
/

&geogrid
parent_id           = 1,
parent_grid_ratio   = 1,
i_parent_start      = 1,
j_parent_start      = 1,
e_we                = 515,
e_sn                = 515,
!
```

||||| IMPORTANT NOTE |||||

(续下页)

(接上页)

```

! The default datasets used to produce the MAXSNOALB and ALBEDO12M
! fields have changed in WPS v4.0. These fields are now interpolated
! from MODIS-based datasets.
!
! To match the output given by the default namelist.wps in WPS v3.9.1,
! the following setting for geog_data_res may be used:
!
! geog_data_res = 'maxsnowalb_ncep+albedo_ncep+default',      'maxsnowalb_
->ncep+albedo_ncep+default',
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!
geog_data_res = 'default','default',
dx = 12000,
dy = 12000,
map_proj = 'lambert',
ref_lat   = 31.00,
ref_lon   = 120.00,
ref_x     = 351
ref_y     = 208
truelat1  = 30.0,
truelat2  = 60.0,
stand_lon = 120.0,
geog_data_path = '/lustre/opt/contribute/cascadelake/wrf_cmaq/geo/'
/

&ungrib
  out_format = 'WPS',
  prefix = 'FILE',
/

&metgrid
  fg_name = 'FILE'
  io_form_metgrid = 2,
/

```

4. 运行 geogrid.exe 程序定义模型投影、区域范围，嵌套关系，对地表参数进行插值。

```

#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.3.1-intel-2021.4.0

geogrid.exe

```

5. 根据模拟时期选择文件

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.3.1-intel-2021.4.0

link_grib.csh fnl_2016100*
cp ungrib/Variable_Tables/Vtable.GFS Vtable
```

6. 从 grib 数据中提取所需要的气象参数

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.3.1-intel-2021.4.0

ungrib.exe
```

7. 将气象参数插值到模拟区域

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.3.1-intel-2021.4.0

metgrid.exe
```

π2.0 上运行 WRF

1. 由于 WRF 运行数据需要复杂的文件依赖关系，可先拷贝 WRF 目录中必要的文件到本地

```
cd ~/data
cd WRF
mkdir WRF-4.3.1 && cd WRF-4.3.1
cp -r /lustre/opt/contribute/cascadelake/wrf_cmaq/packet_1/WRF-master/run/*_
↪ ./
```

2. 拷贝 WPS 生成的 met 文件到 WRF-4.3.1 目录

```
cp -r ~/data/WRF/WPS-4.3.1/met_em.d* ./
```

3. namelist.input 文件内容设置如下，参数需要与 wps 的 namelist.wps 参数一致：

```
&time_control
run_days              = 2,
run_hours             = 0,
run_minutes           = 0,
run_seconds           = 0,
start_year            = 2016,
start_month           = 10,
start_day             = 06,
start_hour            = 00,
end_year              = 2016,
end_month             = 10,
end_day               = 08,
end_hour              = 00,
interval_seconds      = 21600
input_from_file       = .true.,.true.,
history_interval      = 60,   60,
frames_per_outfile    = 12,   12,
restart               = .false.,
restart_interval      = 5000,
io_form_history       = 2
io_form_restart       = 2
io_form_input         = 2
io_form_boundary      = 2
/

&domains
time_step             = 60,
time_step_fract_num   = 0,
time_step_fract_den   = 1,
max_dom               = 1,
e_we                  = 515,   112,
e_sn                  = 515,   97,
e_vert                = 33,   33,
p_top_requested       = 5000,
num_metgrid_levels    = 32,
num_metgrid_soil_levels = 4,
dx                    = 12000,
dy                    = 12000,
grid_id               = 1,   2,
parent_id             = 0,   1,
i_parent_start        = 1,   31,
j_parent_start        = 1,   17,
parent_grid_ratio      = 1,   3,
parent_time_step_ratio = 1,   3,
feedback              = 1,
smooth_option         = 0
```

(续下页)

(接上页)

```

/

&physics
physics_suite           = 'tropical'
mp_physics              = 6,      -1,
cu_physics              = 16,     -1,
ra_lw_physics           = 4,      -1,
ra_sw_physics           = 4,      -1,
bl_pbl_physics          = 8,      8,
sf_sfclay_physics       = 1,      1,
sf_surface_physics      = 2,      -1,
radt                   = 12,     30,
bldt                   = 0,      0,
cudt                   = 5,      5,
icloud                  = 1,
num_land_cat            = 21,
sf_urban_physics        = 0,      0,      0,
/

&fdda
/

&dynamics
hybrid_opt              = 2,
w_damping               = 0,
diff_opt                = 1,      1,
km_opt                  = 4,      4,
diff_6th_opt            = 0,      0,
diff_6th_factor         = 0.12,   0.12,
base_temp               = 290.,
damp_opt                = 3,
zdamp                   = 5000.,   5000.,
dampcoef                = 0.2,     0.2,
khdif                   = 0,      0,
kvdif                   = 0,      0,
non_hydrostatic         = .true., .true.,
moist_adv_opt            = 1,      1,
scalar_adv_opt          = 1,      1,
gwd_opt                 = 0,      1,
/

&bdy_control
spec_bdy_width          = 5,
specified                = .true.
/

&grib2
/

```

(续下页)

(接上页)

```
&namelist_quilt
nio_tasks_per_group = 0,
nio_groups = 1,
/
```

4. 运行 real.exe 程序，脚本如下：

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.3.1-intel-2021.4.0
ulimit -s unlimited
real.exe
```

5. 运行 wrf.exe 程序，脚本如下，该部分是最终也是最耗时的执行程序。

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.3.1-intel-2021.4.0
ulimit -s unlimited
mpirun wrf.exe
```

ARM 上运行 WRF

1. 由于 WRF 运行数据需要复杂的文件依赖关系，可先拷贝 WRF 目录中必要的文件到本地

```
cd ~/data
cd WRF
mkdir WRF-4.3.1 && cd WRF-4.3.1
cp -r /lustre/opt/contribute/cascadelake/wrf_cmaq/packet_1/WRF-master/run/* .
↪ ./
```

2. 拷贝 WPS 生成的 met 文件到 WRF-4.3.1 目录

```
cp -r ~/data/WRF/WPS-4.3.1/met_em.d* ./
```

3. namelist.input 文件内容设置如下，参数需要与 wps 的 namelist.wps 参数一致：

```

&time_control
run_days           = 2,
run_hours          = 0,
run_minutes        = 0,
run_seconds        = 0,
start_year         = 2016,
start_month        = 10,
start_day          = 06,
start_hour         = 00,
end_year           = 2016,
end_month          = 10,
end_day            = 08,
end_hour           = 00,
interval_seconds   = 21600
input_from_file    = .true.,.true.,
history_interval   = 60,    60,
frames_per_outfile = 12,    12,
restart            = .false.,
restart_interval   = 5000,
io_form_history    = 2
io_form_restart    = 2
io_form_input      = 2
io_form_boundary   = 2
/

&domains
time_step          = 60,
time_step_fract_num = 0,
time_step_fract_den = 1,
max_dom            = 1,
e_we               = 515,    112,
e_sn               = 515,    97,
e_vert             = 33,    33,
p_top_requested    = 5000,
num_metgrid_levels = 32,
num_metgrid_soil_levels = 4,
dx                 = 12000,
dy                 = 12000,
grid_id            = 1,    2,
parent_id          = 0,    1,
i_parent_start     = 1,    31,
j_parent_start     = 1,    17,
parent_grid_ratio   = 1,    3,
parent_time_step_ratio = 1,    3,
feedback           = 1,
smooth_option      = 0
/

&physics
physics_suite      = 'tropical'

```

(续下页)

(接上页)

```

mp_physics           = 6,      -1,
cu_physics           = 16,     -1,
ra_lw_physics        = 4,      -1,
ra_sw_physics        = 4,      -1,
bl_pbl_physics       = 8,       8,
sf_sfclay_physics    = 1,       1,
sf_surface_physics   = 2,     -1,
radt                 = 12,     30,
bldt                 = 0,       0,
cudt                 = 5,       5,
icloud               = 1,
num_land_cat         = 21,
sf_urban_physics     = 0,       0,      0,
/

&fdda
/

&dynamics
hybrid_opt           = 2,
w_damping            = 0,
diff_opt             = 1,       1,
km_opt               = 4,       4,
diff_6th_opt         = 0,       0,
diff_6th_factor      = 0.12,   0.12,
base_temp            = 290.,
damp_opt             = 3,
zdamp                = 5000.,  5000.,
dampcoef             = 0.2,    0.2,
khdif                = 0,       0,
kvdif                = 0,       0,
non_hydrostatic      = .true., .true.,
moist_adv_opt        = 1,       1,
scalar_adv_opt       = 1,       1,
gwd_opt              = 0,       1,
/

&bdy_control
spec_bdy_width       = 5,
specified            = .true.
/

&grib2
/

&namelist_quilt
nio_tasks_per_group = 0,
nio_groups = 1,
/

```

4. 运行 `real.exe` 程序，脚本如下：

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.5.1-gcc-10.3.1
ulimit -s unlimited
real.exe
```

5. 运行 `wrf.exe` 程序，脚本如下，该部分是最终也是最耗时的执行程序。

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf/4.5.1-gcc-10.3.1
ulimit -s unlimited
mpirun wrf.exe
```

运行结果 (单位为：秒，越低越好)

思源一号上 **WRF** 的运行时间

wrf/4.2.1-oneapi-2021.4.0				
核数	64	128	256	
Exec time	0:36:21	0:18:05	0:10:44	

π2.0 上 **WRF** 的运行时间

wrf/4.3.1-intel-2021.4.0				
核数	40	80	160	
Exec time	1:10:28	0:42:22	0:26:01	

ARM 上 WRF 的运行时间

wrf/4.5.1-gcc-10.3.1			
核数	64	128	256
Exec time	1:48:51	1:00:02	0:50:49

参考资料

- WRF 官网

Elphbolt

简介

Elphbolt 用于计算物理领域中的电子输运，采用的是严格电声耦合结合玻尔兹曼输运的方法。

Elphbolt 的基本使用

1. 准备两个目录来存放相关参数文件：

```
workdir="./Si_6r4_300K_CBM_gcc/"
inputdir="./input"

mkdir $workdir
mkdir $inputdir
```

2. 在 input 目录下编写如下 input.nml 文件：

```
&allocations
  numelements=1
  numatoms=2
/

&crystal_info
  name = 'Cubic Si'
  elements="Si"
  atomtypes=1 1
  lattvecs(:,1) = -0.27010011  0.00000000  0.27010011
  lattvecs(:,2) =  0.00000000  0.27010011  0.27010011
  lattvecs(:,3) = -0.27010011  0.27010011  0.00000000
  basis(:,1) =  0.00 0.00 0.00
  basis(:,2) =  0.25 0.25 0.25
  T = 300.0 !K
  epsilon0 = 11.7 !From Ioffe
/
```

(续下页)

(接上页)

```

&electrons
    spindeg = 2
    indlowband = 5 !Lowest transport band
    indhighband = 6 !Highest transport band
    indlowconduction = 5 !Lowest conduction band
    numbands = 8 !Total wannier bands
    enref = 6.70035 !eV, CBM
    chempot = 6.70035 !eV, CBM
/

&numerics
    qmesh = 6 6 6
    mesh_ref = 4 !kmesh = 24 24 24
    fsthick = 0.4 !eV about enref
    datadumpdir = './scratch/' !Or, enter suitable scratch directory
    conv_thres = 0.0001
    maxiter = 50 !Maximum number of iterations
/

&wannier
    coarse_qmesh = 6 6 6
/

```

3. 在 Si_6r4_300K_CBM_gcc 目录下执行以下命令：

```

cp ../$inputdir/input.nml .

ln -s ../$inputdir/rcells_g .
ln -s ../$inputdir/rcells_k .
ln -s ../$inputdir/rcells_q .
ln -s ../$inputdir/wsdeg_g .
ln -s ../$inputdir/wsdeg_k .
ln -s ../$inputdir/wsdeg_q .
ln -s ../$inputdir/epwdata.fmt .
ln -s ../$inputdir/epmatwp1 .
ln -s ../$inputdir/FORCE_CONSTANTS_3RD .
ln -s ../$inputdir/espresso.ifc2 .

```

4. 在 Si_6r4_300K_CBM_gcc 目录下编写以下 elphbolt.slurm 脚本：

```

#!/bin/bash

#SBATCH --job-name=elphbolt
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=4
#SBATCH --output=%j.out
#SBATCH --error=%j.err

```

(续下页)

(接上页)

```

0.11631216E+02  0.11631216E+02  0.11631216E+02
-0.11631216E+02  0.11631216E+02  -0.11631216E+02
Brillouin zone volume = 0.15943204E+03 1/nm^3
Crystal temperature = 300.00 K

_____Reading numerics information...
q-mesh =      6      6      6
k-mesh =     24     24     24
Fermi window thickness (each side of reference energy) = 0.40000000E+00 eV
Working directory = /lustre/home/acct-hpc/hpcpzz/Elphbolttest/Si_6r4_300K_
↳CBM_gcc
Data dump directory = ./scratch/
T-dependent data dump directory = ./scratch/T0.300E+03
e-ph directory = ./scratch/g2
ph-ph directory = ./scratch/V2
Reuse e-ph matrix elements: F
Reuse ph-e matrix elements: F
Reuse ph-ph matrix elements: F
Reuse ph-ph transition probabilities: F
Use tetrahedron method: F
Include ph-e interaction: T
Include ph-isotope interaction: F
Include ph-substitution interaction: F
Include electron-charged impurity interaction: F
Include drag: T
Plot quantities along path: F
Maximum number of BTE iterations = 50
BTE convergence threshold = 0.10000000E-03

_____Analyzing symmetry...
Crystal symmetry group = Fd-3m
Number of crystal symmetries (without time-reversal) = 48

_____Reading EPW Wannier information...

```

gmx_MMPBSA

简介

gmx_MMPBSA 是一种基于 **AMBER** 的 **MMPBSA.py** 开发的新工具, 旨在使用 **GROMACS** 文件执行端态自由能计算。它与所有 **GROMACS** 版本以及 **AmberTools20** 或 **21** 一起使用, 与现有程序相比, 它在兼容性、多功能性、分析和并行化方面都有改进。在当前版本中, **gmx_MMPBSA** 支持多种不同的系统, 包括但不限于: 蛋白质、蛋白质配体、蛋白质 DNA、金属蛋白肽、蛋白聚糖、膜蛋白、多组分系统 (例如, 蛋白质 DNA RNA 离子配体)

可用的版本

版本	平台	构建方式	模块名
1.5.2		容器	gmx_mmpbsa/1.5.2-gcc-9.3.0 思源一号
2020		容器	gmx_MMPBSA/1.4.3-gcc-9.3.0-ambertools-20-gromacs2021
1.6.2		conda	/lustre/share/conda_env/gmxMMPBSA

安装方法

conda 安装 gmx_MMPBSA

1. env.yml

```
name:
channels:
  - conda-forge
  - bioconda
  - defaults
dependencies:
  - python=3.9
  - ambertools=21.12
  - mpi4py=3.1.3
  - compilers=1.2.0
  - gromacs==2022.4
  - git
  - pip
  - pip:
    - pyqt5==5.15.6
    - gmx-mmpbsa
    - git+https://github.com/Valdes-Tresanco-MS/ParmEd.git@v3.4
    - pandas==1.2.2
    - seaborn<0.12
    - scipy>=1.6.1
    - matplotlib>=3.5.1
    - tqdm
```

2. 安装命令:

```
srunk -p small -n 4 --pty /bin/bash
module load miniconda3/22.11.1
source activate
conda env create -n gmxMMPBSA --file env.yml
conda activate gmxMMPBSA
conda install -c conda-forge ocl-icd-system
```

3. 使用 conda 安装的 gmx_MMPBSA 的计算脚本，该方法只需要计算脚本不需要 run1.sh

```
#!/bin/bash

#SBATCH --job-name=gmx_MMPBSA
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load miniconda3/22.11.1
source activate gmxMMPBSA

mpirun -np 2 gmx_MMPBSA -O -i mmpbsa.in -cs com.tpr -ci index.ndx -cg 3 4 -
↳ct com_traj.xtc -nogui
```

算例获取方式

```
cd ~ && git clone https://github.com/Valdes-Tresanco-MS/gmx_MMPBSA.git
ls gmx_MMPBSA/docs/examples
```

为保证顺利运行，请将 `gmx_MMPBSA.slurm`、`run1.sh` 和数据放在同一目录下

集群上的 **gmx_MMPBSA**

- 思源一号 `gmx_MMPBSA`
- $\pi 2.0$ `gmx_MMPBSA`

思源一号上的 **gmx_MMPBSA** (版本: **1.5.2**)

`gmx_MMPBSA.slurm` 内容如下:

```
#!/bin/bash

#SBATCH --job-name=gmx_MMPBSA
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -l unlimited
ulimit -s unlimited

module load gmx_mmpbsa/1.5.2-gcc-9.3.0
gmx_MMPBSA_1.5.2
```

`run1.sh` 脚本内容如下:

```
#!/bin/bash
gmX_MMPBSA MPI -O -i mmpbsa.in -cs com.tpr -ci index.ndx -cg 1 13 -ct com_
→traj.xtc -nogui
```

给 run1.sh 增加可执行权限

```
chmod +x run1.sh
```

只有将 gmX_MMPBSA.slurm、run1.sh 和数据放在同一目录下才可正常运行。

使用如下命令提交：

```
$ sbatch gmX_MMPBSA.slurm
```

π2.0 gmX_MMPBSA (版本: 1.4.3)

gmX_MMPBSA.slurm 内容如下：

```
#!/bin/bash

#SBATCH --job-name=gmX_MMPBSA
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gmX_MMPBSA/1.4.3-gcc-9.3.0-ambertools-20-gromacs2021
mpirun gmX_MMPBSA_GROMACS2021
```

run1.sh 脚本内容如下：

```
#!/bin/bash
gmX_MMPBSA MPI -O -i mmpbsa.in -cs com.tpr -ci index.ndx -cg 1 13 -ct com_
→traj.xtc -nogui
```

给 run1.sh 增加可执行权限

```
chmod +x run1.sh
```

只有将 gmX_MMPBSA.slurm、run1.sh 和数据放在同一目录下才可正常运行。

使用如下命令提交：

```
$ sbatch gmX_MMPBSA.slurm
```

π2.0-KOS 平台 gmx_MMPBSA (版本: 1.6.2)

gmx_MMPBSA.slurm 内容如下:

```
#!/bin/bash

#SBATCH --job-name=gmx_MMPBSA
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load miniconda3/22.11.1
source activate /lustre/share/conda_env/gmxMMPBSA

mpirun -np 2 gmx_MMPBSA -O -i mmpbsa.in -cs com.tpr -ci index.ndx -cg 3 4 -
→ct com_traj.xtc -nogui
```

该版本只需要 gmx_MMPBSA.slurm 和数据文件，不需要 run1.sh 即可正常运行。
使用如下命令提交:

```
$ sbatch gmx_MMPBSA.slurm
```

运行结果

平台	思源一号	思源一号	pi2.0	pi2.0
核数	64	128	40	80
时间	72s	65s	117s	75s

参考资料

- [gmx_MMPBSA 官网](#)

CMAQ

简介

CMAQ (The Community Multiscale Air Quality Modeling System) 是美国环境保护局开发的可用于空气质量模型模拟的开源项目，该软件结合了大气科学和空气质量建模方面的现有知识，同时与多处理器计算技术相结合，可有效预测臭氧、颗粒物、有毒物质和酸沉降。

可用版本

版本	平台	构建方式	模块名
5.3.2		源码	cmaq/5.3.2-oneapi-2021.4.0 思源一号
5.4		源码	cmaq/5.4-oneapi-2021.4.0 思源一号
5.3.3		源码	cmaq/5.3.3-intel-2021.4.0 Pi2.0
5.4		源码	cmaq/5.4-intel-2021.4.0 Pi2.0
5.4		源码	cmaq/5.4-intel-2021.4.0 ARM

算例获取路径

```
/dssg/share/sample/cmaq/CMAQv5.3.2_Benchmark_2Day_Input.tar.gz
/dssg/share/sample/cmaq/CMAQv5.4_2018_12NE3_Benchmark_2Day_Input.tar.gz
/lustre/share/samples/cmaq/CMAQv5.3.2_Benchmark_2Day_Input.tar.gz
/lustre/share/samples/cmaq/CMAQv5.4_2018_12NE3_Benchmark_2Day_Input.tar.gz
```

集群上的 **CMAQ**

- 思源一号上的 *CMAQ*
- *Pi2.0* 上的 *CMAQ*
- *ARM* 上的 *CMAQ*

思源一号上的 **CMAQ**

可执行文件所在的目录

```
CCTM_v532.exe
/dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/cmaq/CMAQ_
→Project/CCTM

BCON_v532.exe、ICON_v532.exe、mcip.exe
/dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/cmaq/CMAQ_
→Project/PREP
```

您还可以 `module show cmaq` 获取文件安装目录

```
module avail cmaq
module show cmaq/5.3.2-oneapi-2021.4.0
```

运行 CMAQ 的流程

此处使用 CMAQ 的核心可执行文件 CCTM_v532.exe。

首先拷贝 **CMAQ_Project** 到本地

```
mkdir ~/cmaq
cd ~/cmaq
cp -r /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/wrf_cmaq/cmaq/
  ↪ CMAQ_Project ./
```

然后将算例解压到 **CMAQ_Project** 下的 **data** 目录

```
cd ~/cmaq/CMAQ_Project/data
cp -r /dssg/share/sample/cmaq/CMAQv5.3.2_Benchmark_2Day_Input.tar.gz ./
tar xf CMAQv5.3.2_Benchmark_2Day_Input.tar.gz
mv CMAQv5.3.2_Benchmark_2Day_Input/* ./
```

接下来根据运行核数修改可执行文件的内部参数

修改的可执行文件为: ~/cmaq/CMAQ_Project/CCTM/scripts/run_cctm_Bench_2016_12SE1.csh

```
@ NPCOL = 16; @ NPROW = 8
###
  ↪ NPCOL*NPROW=运行数据的总核数。比如思源上使用2个节点共128核运行数据，参数配置如上
set END_DATE = "2016-07-02"
### 运行时间为：7月1日-7月2日
```

执行脚本设置如下

脚本的位置应在: ~/cmaq/CMAQ_Project/CCTM/scripts/

```
#!/bin/csh
#SBATCH --job-name=cmaq
#SBATCH --partition=64c512g
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cmaq
setenv INPDIR /dssg/home/acct-hpc/hpchg/c/data/cmaq/cmaq_test2/CMAQ_Project/
  ↪ data/2016_12SE1
csh run_cctm_Bench_2016_12SE1.csh
```

Pi2.0 上的 CMAQ

可执行文件所在的目录

```
CCTM_v54.exe
/lustre/opt/cascadelake/linux-rhel8-skylake_avx512/intel-2021.4.0/wrf_cmaq/
↪cmaq-5.4/CMAQ-CMAQv5.4_14Oct2022/CCTM/scripts/BLD_CCTM_v54_intel

BCON_v54.exe、ICON_v54.exe、mcip.exe
/lustre/opt/cascadelake/linux-rhel8-skylake_avx512/intel-2021.4.0/wrf_cmaq/
↪cmaq-5.4/CMAQ-CMAQv5.4_14Oct2022/PREP
```

ARM 上的 CMAQ

可执行文件所在的目录

```
CCTM_v54.exe
/lustre/opt/linux-openeuler22-aarch64/linux-centos8-aarch64/gcc-10.3.1/cmaq/
↪CMAQ-CMAQv5.4_14Oct2022/CCTM/scripts/BLD_CCTM_v54_gcc

BCON_v54.exe、ICON_v54.exe、mcip.exe
/lustre/opt/linux-openeuler22-aarch64/linux-centos8-aarch64/gcc-10.3.1/cmaq/
↪CMAQ-CMAQv5.4_14Oct2022/PREP
```

运行结果 (单位为: 秒, 越低越好)

思源一号上 **CMAQ** 的运行时间

cmaq/5.3.2-oneapi-2021.4.0				
核数	64	128	256	
Exec time	0:06:41	0:05:18	0:04:26	

Pi2.0 上 **CMAQ** 的运行时间

cmaq/5.4-intel-2021.4.0				
核数	64	128	256	
Exec time	0:11:37	0:07:57	0:04:55	

ARM 上 **CMAQ** 的运行时间

cmaq/5.4-gcc-10.3.1				
核数	32	64	128	
Exec time	0:38:44	0:23:06	0:17:00	

参考资料

- [CMAQ 官网](#)

thirdorder

介绍

Thirdorder 可帮助创建用于计算非谐原子间力常数的输入文件，利用系统的对称性来最小化所需的 DFT 计算次数，同时该应用也可以允许用户根据运行的结果构建三阶 IFC 矩阵。

算例获取

思源一号

```
/dssg/share/sample/thirdorder
```

$\pi 2.0$

```
/lustre/share/samples/thirdorder
```

集群上的 **thirdorder**

- 思源一号上的 *thirdorder*
- $\pi 2.0$ 上的 *thirdorder*

思源一号上的 **thirdorder**

使用方法如下

```
mkdir ~/thirdorder && cd ~/thirdorder
cp -r /dssg/share/sample/thirdorder/* ./
singularity shell /dssg/share/imgs/thirdorder/thirdorder.sif
thirdorder_espresso.py scf.in
```

运行结果如下将显示 **thirdorder** 能够正常使用

```
Usage:
  /opt/software/install/sousaw-thirdorder-6050bebe4dd1/thirdorder_
↪espresso.py unitcell.in sow na nb nc cutoff[nm/-integer] supercell_
↪template.in
  /opt/software/install/sousaw-thirdorder-6050bebe4dd1/thirdorder_
↪espresso.py unitcell.in reap na nb nc cutoff[nm/-integer]
```

π2.0 上的 thirdorder

使用方法如下

```
mkdir ~/thirdorder && cd ~/thirdorder
cp -r /lustre/share/samples/thirdorder/* ./
singularity shell /lustre/share/img/x86/thirdorder/thirdorder.sif
thirdorder_espresso.py scf.in
```

运行结果如下将显示 thirdorder 能够正常使用

```
Usage:
/opt/software/install/sousaw-thirdorder-6050bebe4dd1/thirdorder_
↪espresso.py unitcell.in sow na nb nc cutoff[nm/-integer] supercell_
↪template.in
/opt/software/install/sousaw-thirdorder-6050bebe4dd1/thirdorder_
↪espresso.py unitcell.in reap na nb nc cutoff[nm/-integer]
```

参考链接

- [thirdorder website](#)

Gift-BTE

Gift-BTE 是对介观尺度的声子导热问题进行数值计算的 C++ 软件，由上海交通大学密西根学院、未来技术学院鲍华课题组开发。该软件可以用于各种微纳结构的声子导热仿真，包括但不限于半导体器件、微纳多孔结构等。

可用版本

版本	平台	构建方式	模块名
1.0		源码	bte/1.0-openmpi-3.1.5 π2.0
1.0		源码	bte/1.0-openmpi-4.1.5 π2.0 kos

安装方法，以 Pi2.0 集群为例

```
srtn -p cpu -n 4 --pty /bin/bash
git clone https://github.com/Gift-BTE-developer/Gift-BTE.git
mv GiftBTE bte
module load openmpi/4.1.5-gcc-8.5.0
module load cmake/3.26.3-gcc-8.5.0
cd bte
cmake -B cpu-build -S. -DCMAKE_BUILD_TYPE=Release
cd cpu-build
make
```

kos 系统下 bte 运行脚本

```
#!/bin/bash

#SBATCH --job-name=bte
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=16
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load bte/1.0-openmpi-4.1.5

mpirun -np 16 BTE_CPU
```

算例获取

```
mkdir ~/bte && cd ~/bte
cp -r /lustre/share/benchmarks/bte/input.tar.gz ./
tar xf input.tar.gz
```

作业运行

π2.0 集群

作业运行前数据、脚本所在目录如下所示：

```
[hpc@login3 BTE]$ tree data/
data/
├── input
│   ├── FinFet_3D_2500.mphtxt
│   ├── heatfile.dat
│   ├── inputband_8.dat
│   ├── inputbc_Finfet.dat
│   ├── inputdata.dat
│   └── inputmesh.dat
├── input.tar.gz
└── run.slurm
```

运行脚本如下所示：

```
#!/bin/bash
#SBATCH --job-name=bte-test
#SBATCH --partition=cpu
#SBATCH -N 2
#SBATCH --ntasks-per-node=32
#SBATCH --exclusive
#SBATCH --output=%j.out
```

(续下页)

(接上页)

```
#SBATCH --error=%j.err

export OMP_NUM_THREADS=1
module load bte
mpirun BTEcmd
```

提交上述作业

```
sbatch run.slurm
```

作业运行结束后的目录如下所示：

```
[hpc@login3 BTE]$ tree data/
data/
├── 9729078.err
├── 9729078.out
├── Boundary_heat_flux.dat
├── HeatFlux.dat
├── input
│   ├── FinFet_3D_2500.mphtxt
│   ├── heatfile.dat
│   ├── inputband_8.dat
│   ├── inputbc_Finfet.dat
│   ├── inputdata.dat
│   └── inputmesh.dat
├── Interface_emit_temp.dat
├── run.slurm
├── Tempcell1.dat
├── Tempcell2.dat
└── Tempcell.dat
```

上述文件的具体含义可参考 BTE 官方网站：bte.sjtu.edu.cn.

文件内容最后一行显示如下内容，代表作业运行正确。

```
[hpc@login3 data]$ tail -n 1 9729078.out
Time taken by iteration: 509080 milliseconds
```

运行结果

π2.0

bte/1.0-openmpi-3.1.5				
核数	16	32	64	
时间 milliseconds	637674	618820	509080	

参考资料

- [Gift-BTE 官网](#)

DAMASK

简介

DAMASK 是一个统一的多物理晶体塑性模拟包。连续体力学边值问题的求解需要连接每个材料点的变形和应力的本构响应, 该问题在 DAMASK 中基于晶体可塑性使用各种本构模型和均质化方法能够被有效解决。除此之外, 孤立地处理力学已不足以研究新兴的先进高强度材料, 在这些材料中, 变形的发生与位移相变、显着加热和潜在的损伤演变相关, DAMASK 能够有效处理多物理问题。

可用的版本

版本	平台	构建方式	模块名
3.0.0-alpha5		spack	damask/3.0.0-alpha5-gcc-11.2.0-hdf5-openblas-openmpi 思源一号
3.0.0-alpha6		spack	damask/3.0.0-alpha6-gcc-11.2.0-hdf5-openblas-openmpi 思源一号
3.0.0-alpha7		spack	damask/3.0.0-alpha7-gcc-11.2.0-hdf5-openblas-openmpi 思源一号
3.0.0-alpha5		spack	damask/3.0.0-alpha5-gcc-8.3.0-openblas

安装方式

使用 conda 安装方法:

```

srun -p small -n 4 --pty /bin/bash
module load miniconda3
source activate
conda create -n damask python=3.8.18
conda activate damask
conda install -c conda-forge damask=3.0.0a8
conda install -c conda-forge ucx
```

使用 conda 安装的 damask 的计算脚本:

```

#SBATCH -J damask
#SBATCH -p cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=32
```

(续下页)

(接上页)

```
#SBATCH -o %j.out
#SBATCH -e %j.err

module load miniconda3
source activate damask

export UCX_NET_DEVICES=mlx5_0:1
export OMPI_MCA_btl=^openib

mpirun --mca pml ucx --mca osc ucx DAMASK_grid --load shearXY.yaml --geom 20grains32x32x32.vti --material material.yaml
```

算例获取方式

思源：

```
mkdir ~/damask && cd ~/damask
cp -r /dssg/share/sample/damask/* ./
tar xf grid.tar.xz
cd grid

π2.0：
mkdir ~/damask && cd ~/damask
cp -r /lustre/share/samples/damask/* ./
tar xf grid.tar.xz
cd grid
```

集群上的 DAMASK

- 思源一号 DAMASK
- π2.0 DAMASK

思源一号上运行 DAMASK

3.0.0-alpha7 版本的运行脚本

```
#!/bin/bash
#SBATCH --job-name=32core_damask
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=32
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load damask/3.0.0-alpha7-gcc-11.2.0-hdf5-openblas-openmpi
mpirun DAMASK_grid --load shearXY.yaml --geom 20grains32x32x32.vti
```

3.0.0-alpha5 版本的运行脚本

```
#!/bin/bash
#SBATCH --job-name=32core_damask
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=32
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load damask/3.0.0-alpha5-gcc-11.2.0-hdf5-openblas-openmpi
mpirun DAMASK_grid --load shearXY.yaml --geom 20grains32x32x32.vti
```

π2.0 上运行 DAMASK

π2.0 上 3.0.0-alpha5 版本的运行脚本

```
#!/bin/bash
#SBATCH --job-name=32core_damask
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load damask/3.0.0-alpha5-gcc-8.3.0-openblas
mpirun -np 32 DAMASK_grid --load shearXY.yaml --geom 20grains32x32x32.vti
```

运行结果

单位为秒

思源一号

3.0.0-alpha6				
核数	8	16	32	
时间	210	108	59	

3.0.0-alpha5				
核数	8	16	32	
时间	214	109	61	

π2.0

3.0.0-alpha5			
核数	8	16	32
时间	235	126	78

参考链接: <https://damask.mpie.de/index.html>

ROMS

简介

ROMS 全称 Regional Ocean Modeling System, 是一个三维区域海洋模型, 由罗格斯大学海洋与海岸科学研究所和加利福尼亚大学洛杉矶分校共同研究开发, 被广泛应用于海洋及河口地区的水动力及水环境模拟。

ROMS3.6 依赖的软件及其版本

思源一号和 π2.0 上安装的 ROMS、依赖软件及其版本保持一致

zlib	: 1.2.11
gzip	: 2.1.1
hdf5	: 1.12.0
netcdf-c	: 4.8.0
netcdf-fortran	: 4.5.3
pnetcdf	: 1.12.0
parallelIO	: 2.5.6

使用 module 调用 ROMS 依赖环境

思源一号	: module load roms/3.6-intel-2021.4.0
π2.0	: module load roms/3.6-intel-2021.4.0

依赖软件具体的安装路径

思源一号	
zlib	: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ →ROMS/ROMSRel/netcdf
gzip	: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ →ROMS/ROMSRel/netcdf
hdf5	: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ →ROMS/ROMSRel/netcdf
netcdf-c	: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ →ROMS/ROMSRel/netcdf
netcdf-fortran	: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ →ROMS/ROMSRel/netcdf
parallelNetcdf	: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/

(续下页)

(接上页)

```

↪ROMS/ROMSRelly/pnetcdf
parallelIO      : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/
↪ROMS/ROMSRelly/pio

π2.0
zlib            : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/netcdf
szip            : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/netcdf
hdf5            : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/netcdf
netcdf-c        : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/netcdf
netcdf-fortran  : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/netcdf
parallelNetcdf  : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/pnetcdf
parallelIO      : /lustre/opt/contribute/cascadelake/ROMS/ROMSRelly/pio

```

集群上如何使用 **ROMS**

- 思源一号自定义编译 **ROMS**
- $\pi 2.0$ 自定义编译 **ROMS**
- 思源一号预编译 **ROMS**
- $\pi 2.0$ 预编译 **ROMS**

思源一号自定义编译 **ROMS**

为方便广大师生编译 **ROMS**，我们已将 `build\_roms.sh` 和 `Linux-ifort.mk` 两个文件中依赖软件的参数提前修改上述两个文件的位置为

```

/dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ROMS/ROMS_3.6/ROMS/
↪Bin/build_roms.sh
/dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ROMS/ROMS_3.6/
↪Compilers/Linux-ifort.mk

```

```

srun -p 64c512g -N 1 --exclusive --pty /bin/bash
mkdir -p ~/ROMS/ROMSProjects/upwelling
export PROJECT_DIR=~ /ROMS/ROMSProjects/upwelling
cd $PROJECT_DIR
cp /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ROMS/ROMS_3.6/
↪ROMS/Bin/build_roms.sh ./
cp /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ROMS/ROMS_3.6/
↪ROMS/Include/upwelling.h ./
cp /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ROMS/ROMS_3.6/
↪ROMS/External/roms_upwelling.in ./
./build_roms.sh

```

π2.0 自定义编译 ROMS

为方便广大师生编译 ROMS，我们已将 `build\_roms.sh` 和 `Linux-ifort.mk` 两个文件中依赖软件的参数提前修改上述两个文件的位置为

```
/lustre/opt/contribute/cascadelake/ROMS/ROMS_3.6/ROMS/Bin/build_roms.sh
/lustre/opt/contribute/cascadelake/ROMS/ROMS_3.6/Compilers/Linux-ifort.mk
```

```
srunk -p cpu -N 1 --exclusive --pty /bin/bash
mkdir -p ~/ROMS1/ROMSProjects/upwelling
export PROJECT_DIR=~/ROMS1/ROMSProjects/upwelling
cd $PROJECT_DIR
cp /lustre/opt/contribute/cascadelake/ROMS/ROMS_3.6/ROMS/Bin/build_roms.sh .
↪ /
cp /lustre/opt/contribute/cascadelake/ROMS/ROMS_3.6/ROMS/Include/upwelling.
↪ h ./
cp /lustre/opt/contribute/cascadelake/ROMS/ROMS_3.6/ROMS/External/roms_
↪ upwelling.in ./
./build_roms.sh
```

思源一号预编译 ROMS

更改 `roms\_upwelling.in` 文件的参数，如下所示

```
VARNAME = /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.4.0/ROMS/ROMS_
↪ 3.6/ROMS/External/varinfo.yaml
NtileI == 2 ! I-direction partition
NtileJ == 2 ! J-direction partition
```

`NtileI` 和 `NtileJ` 的乘积需等于总核数

提交如下脚本运行作业

```
#!/bin/bash
#SBATCH --job-name=ROMS
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=4
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load roms/3.6-intel-2021.4.0
mpirun -np 4 romsM roms_upwelling.in
```

π2.0 预编译 ROMS

更改 `roms\_upwelling.in` 文件的参数，如下所示

```
VARNAME = /lustre/opt/contribute/cascadelake/ROMS/ROMS_3.6/ROMS/External/  
↪varinfo.yaml  
NtileI == 2                                ! I-direction partition  
NtileJ == 2                                ! J-direction partition
```

‘NtileI’ 和 ‘NtileJ’ 的乘积需等于总核数

提交如下脚本运行作业

```
#!/bin/bash  
#SBATCH --job-name=ROMS  
#SBATCH --partition=small  
#SBATCH -N 1  
#SBATCH --ntasks-per-node=4  
#SBATCH --output=%j.out  
#SBATCH --error=%j.err  
  
module load roms/3.6-intel-2021.4.0  
mpirun -np 4 romsM roms_upwelling.in
```

运行结果

思源一号上的 **ROMS**

核数	1	2	4	8
时间	107	56	36	23

π2.0 上的 **ROMS**

核数	1	2	4	8
时间	134	70	41	29

参考资料

- [ROMS 官方网站](#)

PROJ

简介

PROJ 是一个通用的坐标转换软件，它将地理空间坐标从一个坐标系转换为另一个坐标系。这包括地图投影和大地坐标变换。

PROJ 包含命令行应用程序，可以方便地从文本文件或直接从用户输入转换坐标。除了命令行实用程序之外，**PROJ** 还提供了相关的 **API**。**API** 允许开发人员在自己的软件中使用 **PROJ** 的功能，而不必自己实现类似的功能。

可用的版本

版本	平台	构建方式	模块名
7.2.1		spack	proj/7.2.1-gcc-11.2.0 思源一号
7.2.1		spack	proj/7.2.1-gcc-11.2.0 pi 2.0

使用教程

坐标转换由 PROJ 术语中所称的”proj-strings”来定义。proj-string 描述任何转换，不管它有多么简单或复杂。在本文的示例中，我们将通过示例介绍 proj-strings 的参数含义。

如下所示为坐标转换的典型操作：

- 坐标系转投影空间
- 坐标系转换

坐标系转投影空间

proj-strings 保存给定坐标转换的参数

```
+proj=merc +lat_ts=56.5 +ellps=GRS80

#+proj=merc 表示：将坐标系作为墨卡托投影
#+ellps=GRS80 表示：椭球体GRS80（椭球体名称、坐标系）
#+lat_ts=56.5 有效纬度范围Latitude of true scale
```

上述坐标参数所代表的含义为大地坐标被转换成投影空间，在 GRS80 椭球面上，用 Mercator 投影，其纬度为北纬 56.5 度。

操作如下：

```
输入类型为 +ellps=GRS80, 默认按照 +ellps=GRS80 的默认中经线 和 默认维度0.0
→作为参考点, 输出墨卡托投影

>srun -p 64c512g -N 1 -n 6 --pty /bin/bash
>module load proj gcc
>proj +proj=merc +lat_ts=56.5 +ellps=GRS80
>55.2 12.2 #输入参数
>3399483.80 752085.60 #得到 55.2 12.2 维度、经度转换为米的数据
```

坐标系转换

PROJ 中的 cs2cs 程序可将一个坐标参考系统转换到另一个坐标参考系统，下述 cs2cs 操作的示例内容为将上面输出的墨卡托坐标转换为 UTM

```
# cs2cs 表示坐标系转坐标系
# +proj=merc +lat_ts=56.5 +ellps=GRS80_
→表示：源投影定义，而源投影定义的输入参数为merc（墨卡托）类型参数，即米
# +to 表示：源投影定义转目标投影定义的分隔符
# +proj=utm +zone=32 表示：目标投影定义
>srun -p 64c512g -N 1 -n 6 --pty /bin/bash
>module load proj gcc
>cs2cs +proj=merc +lat_ts=56.5 +ellps=GRS80 +to +proj=utm +zone=32
>3399483.80      752085.60      #输入参数
>6103992.36      1924052.47 0.00 #输出结果
```

使用下面的命令可以将大地坐标直接转换为 UTM

```
#+proj=latlon +lat_ts=56.5 +ellps=GRS80_
→表示：源投影定义，使用GRS80坐标系，而源投影定义的输入参数为latlon（经纬度）类型参数
#+to 表示：源投影定义转目标投影定义的分隔符
#+proj=utm +zone=32 表示：目标投影定义

>srun -p 64c512g -N 1 -n 6 --pty /bin/bash
>module load proj gcc
>cs2cs +proj=latlon +lat_ts=56.5 +ellps=GRS80 +to +proj=utm +zone=32
55.2 12.2      #输入参数
6103992.35      1924052.46 0.00 #输出结果
```

- PROJ 官网

GAMESS

简介

GAMESS (General Atomic and Molecular Electronic Structure System) 是一款由 Ames 研究中心的 Mark Gordon 开发的开源量子化学软件。GAMESS 的功能十分强大，能完成多种类型的计算任务。它支持使用 RHF、UHF、ROHF 或 GVB 波函数计算半经验 MNDO、AM1 或 PM3 模型；可以进行 QM/MM 计算，可以处理溶剂效应，很大程度上它可以成为 Gaussian 的替代品。更多信息请访问 <https://www.msg.chem.iastate.edu/gamess/>

软件下载

软件源代码需在 Mark Gordon 课题组主页上申请，同意协议、填写邮箱等信息后一两天内会收到下载链接、账户名和下载密码。(<https://www.msg.chem.iastate.edu/gamess/download.html>)

自行编译

1. 申请计算节点并加载模块

以选用 gfortran 为编译器，MKL 为数学库，MPI 并行编译为例


```

srun -p 64c512g -n 4 --pty /bin/bash
module purge
module load intel-oneapi-compilers/2022.1.0 intel-oneapi-mkl/2022.1.0 intel-
→oneapi-mpi/2021.6.0

```

2. 检查.bashrc 和.bash_profile

检查 bash shell 配置 (参考 <https://docs.hpc.sjtu.edu.cn/faq/index.html#bashrc>)

3. 解压并进入文件

```

tar -xvf gamess-current.tar.gz
cd ./gamess

```

4. 生成编译文件

方式一在当前路径运行 “./config” 命令, 并根据屏幕提示逐条输出选项方式二直接生成 Makefile 文件和 install.info 文件

生成 Makefile 文件 (其中默认当前路径为./gamess)

```

touch Makefile
cat >> Makefile <<EOF
GMS_PATH = ${PWD}
GMS_VERSION = 00
include ${PWD}/Makefile.in
EOF

```

生成 install.info 文件 (其中默认当前路径为./gamess, 如需更改 gamess 编译路径, 请修改 GMS_BUILD_DIR 值)

```

touch install.info
cat >> install.info <<EOF
#!/bin/csh -f

#                                GAMESS Paths                                #
setenv GMS_PATH                  ${PWD}
setenv GMS_BUILD_DIR             ${PWD}

#                                Machine Type                                #
setenv GMS_TARGET                linux64
setenv GMS_HPC_SYSTEM_TARGET    generic

#                                FORTRAN Compiler Setup                      #
setenv GMS_FORTRAN               ifort
setenv GMS_IFORT_VERNO           22

#                                Mathematical Library Setup                  #

```

(续下页)

(接上页)

```

setenv  GMS_MATHLIB          mkl
setenv  GMS_MATHLIB_PATH     ${MKLROOT}/lib/intel64
setenv  GMS_MKL_VERNO        12
setenv  GMS_LAPACK_LINK_LINE ""
#           parallel message passing model setup
setenv  GMS_DDI_COMM          sockets

#           Michigan State University Coupled Cluster      #
setenv  GMS_MSUCC             false

#           LIBCCHEM CPU/GPU Code Interface                #
setenv  GMS_LIBCCHEM          false

#           Intel Xeon Phi Build: none/knc/knl              #
setenv  GMS_PHI               none

#           Shared Memory Type: sysv/posix                  #
setenv  GMS_SHMTYPE           sysv

#           GAMESS OpenMP support: true/false               #
setenv  GMS_OPENMP            false

#           GAMESS LibXC library: true/false                #
setenv  GMS_LIBXC             false

#           GAMESS MDI library: true/false                  #
setenv  GMS_MDI               false

#           VM2 library: true/false                          #
setenv  GMS_VM2               false

#           Tinker: true/false                              #
setenv  TINKER                 false

#           VB2000: true/false                              #
setenv  VB2000                false

#           XMVB: true/false                                #
setenv  XMVB                   false

#           NEO: true/false                                  #
setenv  NEO                    false

#           NBO: true/false                                  #
setenv  NBO                    false

#####
# Added any additional environmental variables or      #
# module loads below if needed.                        #

```

(续下页)

(接上页)

```
#####
# Capture floating-point exceptions          #
setenv GMS_FPE_FLAGS      ''
EOF
```

5. 编译

编译成功后将在目录中生成 **gamess.00.x** 的可执行文件

```
make ddi
make modules
make -j gamess
```

6. 配置运行环境

Gameess 程序需要依靠 **rungms** 文件来调用可执行文件 **gamess.00.x**，在编译后需调整 **rungms** 中的参数修改当前目录下的 **rungms** 文件，其中 **GMSPATH** 为 **rungms** 文件所在地址、**SCR** 为临时文件所在目录

```
set SCR=~app/gamess/scr
set USERSCR=~app/gamess/scr
set GMSPATH=~app/gamess
```

7. 验证

使用软件自带的测试示例进行测试，在当前路径下运行 **runall** 文件，并执行检查命令 **checktst**

```
./runall 00
./tests/standard/checktst
```

得到如下内容表明验证算例全部通过

```
Checking the results of your sample GAMESS calculations,
the output files (exam??.log) will be taken from .
Only 48 out of 49 examples terminated normally.
Please check carefully each of the following runs:
grep: ./exam49.log: No such file or directory
./exam49.log
which did not completely finish.
exam01: Eerr=0.0e+00 Gerr=0.0e+00.
→Passed.
exam02: Eerr=0.0e+00 Gerr=0.0e+00 Serr=0.0e+00 Lerr=1.8e-03+6.6e-05.
→Passed.
exam03: Eerr=0.0e+00 Gerr=0.0e+00 Derr=0.0e+00.
→Passed.
.
```

(续下页)

(接上页)

```

.
.
.
exam48: E0err=0.0e+00 E1err=0.0e+00 Gerr=0.0e+00.
↪Passed.

```

- GAMESS 官网

DeePMD-kit

简介

DeePMD-kit 是一种基于机器学习的分子动力学模拟方法，该方法是通过使用从头计算得到的数据对深度神经网络模型进行训练，从而得到通用的多体势能模型 (DP 模型)。由于其是基于第一性原理，而具有媲美量子力学的精度。其计算效率高，比第一性原理计算至少快 5 个数量级。目前 DP 模型已成功应用于水和含水体系，金属和合金，相图，高熵陶瓷，化学反应，固态电解质，离子液体等研究领域。

安装教程

本案例使用 conda 进行安装 GPU 版本的 DeePMD-kit

```

srun -p a100 -n 6 --gres=gpu:1 --pty /bin/bash #申请计算资源
module load miniconda3 #加载模块
module load cuda/11.6.2 #加载的cuda模块版本需要和安装的cudatoolkit版本一致
conda create -n deepmd deepmd-kit=2.2.7=*gpu libdeepmd=*=*gpu lammps-dp-
↪cudatoolkit=11.6 horovod -c https://conda.deepmodeling.org
↪#创建名为deepmd的环境
source activate deepmd #激活deepmd环境

```

测试

文档使用官方文档提供的甲烷燃烧模拟案例进行测试，介绍基于 DP 模型的 MD 模拟过程，测试所需要的文件可以通过以下链接下载 <https://github.com/tongzhugroup/Chapter13-tutorial>

数据集准备

在数据集准备过程中，需要考虑的是最终 DP 模型的准确性，即需要在哪个量子力学水平标记数据。本案例使用在 MN15/6-31G** 级别下通过 Gaussian 程序计算产生的势能和原子力，其中由于模拟过程需要处理诸多自由基及其反应，因此选择了对单/多参考系统具有良好精度的 MN15。示例中已提供准备好的数据集。

```

unzip Chapter13-tutorial-master.zip #解压下载的示例文件
cd Chapter13-tutorial-master

```

DP 模型训练

此步骤是提取经过训练的神经网络模型，生成冻结模型 `graph.pb`，并压缩

```
dp freeze -o graph.pb
dp compress -i graph.pb -o graph_compressed.pb -t methane_param.json
```

基于 DP 模型的 MD 模拟

冻结模型可用于反应 MD 模型，以探索甲烷燃烧过程的详细反应机理。此处的 MD 模拟过程由 LAMMPS 程序完成（LAMMPS 已在生成 `deepmd` 环境时完成创建）。体系中包含 100 个甲烷分子和 200 个氧气分子，下载的 `data.methane` 文件中包含系统初始原子坐标信息。`input.lammps` 文件则包含了调用 LAMMPS 程序的关键参数

```
lmp -i input.lammps
conda deactivate #退出 deepmd 环境
```

参考资料

- [DeePMD-kit 官网](#)

CHROMA

简介

Chroma 是一款格点量子色动力学 (LQCD) 数值模拟软件包，它是一个为解决夸克和胶子理论而设计的物理应用程序，其属于美国 USQCD 合作组在美国 SciDac 经费的支持下开发的 USQCD 软件集中的子模块。Chroma 整合了 QCD 基本线性代数操作、多线程/进程 QCD 信息传递、QCD 文件 IO、稀疏矩阵求解等众多模块，并通过 XML 交互协议提供人机接口。

可用的版本

版本	平台	构建方式	镜像路径
2021.4		容器	/dssg/share/imgs/chroma/chroma2021.04.sif 思源一号
2021.4		容器	/lustre/share/img/chroma/chroma2021.04.sif pi2.0

算例路径

```
思源一号
/dssg/share/sample/chroma/szscl_bench.zip

pi2.0
/lustre/share/samples/chroma/szscl_bench.zip
```

软件下载

本文档使用的是在 NVIDIA GPU 云 (NGC) 上预好的 Chroma-2021.04 镜像。更多信息请访问

```
https://catalog.ngc.nvidia.com/orgs/hpc/containers/chroma
```

使用方法

- 一. 思源一号 *Chroma*
- 二. π 2.0 *Chroma*

一. 思源一号 Chroma

运行脚本

```
#!/bin/bash

#SBATCH --job-name=chroma
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=24
#SBATCH --gres=gpu:2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export QUDA_RESOURCE_PATH=$PWD
export GPU_COUNT=2

singularity run --nv /dssg/share/imgs/chroma/chroma2021.04.sif mpirun --
↪allow-run-as-root -x ${QUDA_RESOURCE_PATH} -n ${GPU_COUNT} chroma -i ./
↪test.ini.xml -geom 1 1 1 ${GPU_COUNT} -ptxdb ./qdpdb -gpudirect
```

二. π 2.0 Chroma

运行脚本

```
#!/bin/bash

#SBATCH --job-name=chroma
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
#SBATCH --gres=gpu:2
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

(续下页)

(接上页)

```
export QUDA_RESOURCE_PATH=$PWD
export GPU_COUNT=2

singularity run --nv /lustre/share/img/chroma/chroma2021.04.sif mpirun --
→allow-run-as-root -x ${QUDA_RESOURCE_PATH} -n ${GPU_COUNT} chroma -i ./
→test.ini.xml -geom 1 1 1 ${GPU_COUNT} -ptxdb ./qdpdb -gpudirect
```

容器编译 Chroma

1. 申请计算节点

```
srun -p 64c512g -n 4 --pty /bin/bash
```

2. 拉取远端镜像

参考文档: <https://docs.hpc.sjtu.edu.cn/container/index.html>

```
singularity pull chroma2021.04.sif docker://nvcr.io/hpc/chroma:2021.04
```

自行编译 Chroma

本文档编译的 Chroma 是基于 QUDA 和 QDPJIT, 以在思源 1 号上编译为例

1. 申请计算节点

```
srun -p 64c512g -n 4 --pty /bin/bash
```

2. 加载模块。尝试使用 mpich 进行编译时会报错, 建议使用 openmpi

```
module load gcc/11.2.0 openmpi/4.1.1-gcc-11.2.0-cuda cuda/11.5.0
```

3. 设置环境变量

本文档使用 cmake 安装所有 chroma 的依赖项, 并新建目录 src 以容纳所有依赖项的源程序, 新建目录 build 用于存放编译文件, 新建目录 install 用于存放库文件等

```
cd ~/(path_to_your_installation) #自定义安装路径
export CMAKE_MAKE_OPTS="-- -j$(nproc)"
export SM=sm_80 # 如果在π2.0平台上编译, 注意修改架构号
export QUDA_NVSHMEM=OFF
export QDPJIT_HOST_ARCH="X86;NVPTX"
export ARCHFLAGS="-march=native"
export DEBUGFLAGS=" "
export BASEDIR=$(pwd)
export SRCDIR=${BASEDIR}/src
export BUILDDIR=${BASEDIR}/build
export INSTALLDIR=${BASEDIR}/install
mkdir -p ${SRCDIR}
mkdir -p ${BUILDDIR}
```

4. 下载依赖项。下载过程中容易因为网络连接问题导致拉取空项目，此处建议打包上传拉取的文件

```
cd ${SRCDIR}
git clone --depth=1 --branch llvmorg-14.0.6 https://github.com/llvm/llvm-
→project.git
git clone --branch qmp2-5-4 https://github.com/usqcd-software/qmp.git
git clone --recursive --branch devel https://github.com/JeffersonLab/qdp-
→jit.git
git clone --branch develop https://github.com/lattice/quda.git # c04150e
git clone --branch devel --recursive https://github.com/JeffersonLab/chroma.
→git
cd ${BASEDIR}
```

5. 编译 llvm-project

```
cmake -S ${SRCDIR}/llvm-project/llvm -B ${BUILDDIR}/build_llvm \
-DLLVM_ENABLE_TERMINFO="OFF" \
-DCMAKE_BUILD_TYPE=Release \
-DCMAKE_INSTALL_PREFIX=${INSTALLDIR} \
-DLLVM_TARGETS_TO_BUILD="${QDPJIT_HOST_ARCH}" \
-DLLVM_ENABLE_ZLIB="OFF" \
-DBUILD_SHARED_LIBS="OFF" \
-DLLVM_ENABLE_RTTI="ON"

cmake --build ${BUILDDIR}/build_llvm ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_llvm
```

6. 编译 QMP

```
cmake -S ${SRCDIR}/qmp -B ${BUILDDIR}/build_qmp \
-DCMAKE_INSTALL_PREFIX=${INSTALLDIR} \
-DQMP_MPI=ON \
-DBUILD_SHARED_LIBS=ON \
-DQMP_TESTING=OFF

cmake --build ${BUILDDIR}/build_qmp ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_qmp
```

7. 编译 QDP-JIT

```
cmake -S ${SRCDIR}/qdp-jit -B ${BUILDDIR}/build_qdp-jit \
-DCMAKE_INSTALL_PREFIX=${INSTALLDIR} \
-DCMAKE_PREFIX_PATH=${INSTALLDIR} \
-DBUILD_SHARED_LIBS=ON \
-DQDP_ENABLE_BACKEND=CUDA \
-DQDP_ENABLE_COMM_SPLIT_DEVICEINIT=ON \
-DQDP_ENABLE_LLVM14=ON \
-DQDP_PROP_OPT=OFF \
-DQDP_BUILD_EXAMPLES=OFF \
-DCMAKE_CXX_FLAGS=${ARCHFLAGS}
```

(续下页)

(接上页)

```
cmake --build ${BUILDDIR}/build_qdp-jit ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_qdp-jit
```

8. 编译 QUDA

```
cmake -S ${SRCDIR}/quda -B ${BUILDDIR}/build_quda \
-DMAKE_BUILD_TYPE=RELEASE \
-DMAKE_INSTALL_PREFIX=${INSTALLDIR} \
-DMAKE_PREFIX_PATH=${INSTALLDIR} \
-DQUDA_GPU_ARCH=${SM} \
-DQUDA_NVSHMEM=${QUDA_NVSHMEM} \
-DQUDA_DIRAC_DEFAULT_OFF=ON \
-DQUDA_DIRAC_CLOVER=ON \
-DQUDA_DIRAC_WILSON=ON \
-DQUDA_INTERFACE_QDPJIT=ON \
-DQUDA_QDPJIT=ON \
-DQUDA_INTERFACE_MILC=OFF \
-DQUDA_INTERFACE_CPS=OFF \
-DQUDA_INTERFACE_QDP=ON \
-DQUDA_INTERFACE_TIFR=OFF \
-DQUDA_QMP=ON \
-DQUDA_QIO=OFF \
-DQUDA_MULTIGRID=ON \
-DQUDA_MAX_MULTI_BLAS_N=9 \
-DQUDA_BUILD_SHARED_LIB=ON \
-DQUDA_BUILD_ALL_TESTS=OFF \
-DMAKE_CXX_FLAGS=${ARCHFLAGS}

cmake --build ${BUILDDIR}/build_quda ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_quda
```

9. 编译 Chroma

```
cmake -S ${SRCDIR}/chroma -B ${BUILDDIR}/build_chroma \
-DMAKE_BUILD_TYPE=RELEASE \
-DMAKE_INSTALL_PREFIX=${INSTALLDIR}/ \
-DMAKE_PREFIX_PATH=${INSTALLDIR}/ \
-DBUILD_SHARED_LIBS=ON \
-DChroma_ENABLE_JIT_CLOVER=ON \
-DChroma_ENABLE_QUDA=ON \
-DChroma_ENABLE_OPENMP=ON \
-DMAKE_CXX_STANDARD=20 \
-DMAKE_CXX_FLAGS=${ARCHFLAGS}

cmake --build ${BUILDDIR}/build_chroma ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_chroma
```

10. 编译结果。编译完成后，目录结构如下所示

```

./ (path_to_your_installation)
├─ build      # 存放编译缓存文件
├─ install    # 存放安装的库文件和可执行文件
└─ src        # 存放git拉取的源文件

```

其中，运行 **chroma** 所需的文件位于 **install** 目录中，其目录结构如下所示

```

./install
├─ bin
├─ examples
├─ include
├─ lib
└─ share

```

11. 运行脚本。在脚本中需要设置库文件路径 (`/(path_to_your_installation)/install/lib`) 的系统变量，以及 **chroma** 可执行文件的路径 (`/(path_to_your_installation)/install/bin/chroma`)

```

#!/bin/bash

#SBATCH --job-name=chroma
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=8
#SBATCH --gres=gpu:2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/11.2.0 openmpi/4.1.1-gcc-11.2.0-cuda cuda/11.5.0
export LD_LIBRARY_PATH=/(path_to_your_installation)/install/lib:$LD_LIBRARY_
→PATH
export QUDA_RESOURCE_PATH=$PWD
export GPU_COUNT=2

mpirun -x ${QUDA_RESOURCE_PATH} -np ${GPU_COUNT} / (path_to_your_
→installation)/install/bin/chroma -i ../test.ini.xml -geom 1 1 1 ${GPU_
→COUNT} -ptxdb ./ptxdb

```

运行结果对比

1.Chroma 思源一号

对比不同配置参数下计算所需时间，其中由于 NGC 镜像版本的 **chroma** 已预设好参数，因此在运行脚本中无需调整 **ntasks-per-node** 参数（保持为 1 即可），但对于自编译的 **chroma** 则需要根据调用的卡数调整 **ntasks-per-node** 参数（调用多少个卡就有多少个任务数）。对于不同的卡数，都需要调整运行脚本中对应的 **--gres=gpu** 参数和 **GPU_COUNT** 参数。

卡数-core/task	NGC 镜像	自编译
1A100-12core	47	80
2A100s-8core	24	35
4A100s-12core	13	null

FLASH

简介

FLASH 是一款开源的适用于等离子物理和天体物理的磁流体动力学 (MHD) 模拟软件，它是由罗切斯特大学物理与天文学系 Flash 计算科学中心开发。其包含流体动力学 (Unsplit PPM, WENO, PCM, GP; Split PPM; 2T+Radiation)、磁流体动力学 (Unsplit Staggered Mesh; Split 8 wave)、扩展磁动力学 (Magnetic Resistivity, Hall, Biermann Battery)、状态方程 (理想气体, Degenerate 电离等离子体)、辐射输运 (Multigroup Flux-limited Diffusion) 等物理求解器

Flash 软件依赖项

- 支持 Fortran 90 的编译器
- MPICH
- HDF5 串行版本
- PnetCDF
- Chombo
- HYPRE
- ...

pi2.0 上自定义编译 FLASH

- 软件下载。FLASH 软件下载需要在官网 (<https://flash.rochester.edu/site/flashcode>) 填写申请，获批后才能下载。
- 申请计算节点并加载模块。(本示例使用的依赖项包括：平台已编译的 mpich 和 hdf5、需自编译的 PnetCDF)

```

srun -p small -n 4 --pty /bin/bash
module load mpich/3.4.2-gcc-9.2.0 hdf5/1.10.6-gcc-9.2.0

```

- 编译并安装 PnetCDF。Flash 输出文件使用并行的 NetCDF 格式，因此需额外编译

```

git clone https://github.com/Parallel-NetCDF/PnetCDF.git
cd PnetCDF
autoreconf -i
./configure --prefix=(path_to_installation_pnetcdf) ##
→ 用户可以自行指定 pnetcdf 的安装路径，此处使用 (path_to_installation_

```

(续下页)

(接上页)

`→pnetcdf) 替代`

```
make -j
make install
```

- 配置 Flash。本文档以配置 Flash，生成 2D Sedov 爆炸问题源代码为例

```
tar -xvf FLASHX.Y.tar ## 解压下载的Flash软件
cd FLASHX.Y
./setup Sedov -auto ## 使用setup脚本配置Flash
```

- 修改编译文件。配置完成后源目录下将生成 `object` 文件夹，其中包含配置 Flash 求解 2D Sedov 爆炸问题所需的流体动力学求解器、状态方程、网格单元等模块。但其中的编译文件 `Makefile.h` 为默认值，需将其中的 `MPI_PATH`、`HDF5_PATH` 和 `NCMPI_PATH` 变量进行修改（为便于展示，本示例只使用 `mpich`、`hdf5` 和 `pnetcdf` 三个依赖项）。

```
cd object
vi Makefile.h ## 手动修改文件中的MPI_PATH、HDF5_PATH和NCMPI_
→PATH变量，如NCMPI_PATH=(path_to_installation_pnetcdf)，MPI_PATH和HDF5_
→PATH的路径可以使用env命令查看
make -j
```

- 编译结束。编译完成后将在 `object` 目录下生成名称为 `flash4` 的可执行文件
- 新建目录并编写以下 Flash 参数文件 `flash.par`。参数文件使用官网提供示例 (https://flash.rochester.edu/site/flashcode/user_support/flash4_ug_4p62/node13.html)

```
# runtime parameters
basenm = "sedov_"
lrefine_max = 5
refine_var_1 = "dens"
refine_var_2 = "pres"
restart = .false.
checkpointFileIntervalTime = 0.01
nend = 10000
tmax = 0.05
gamma = 1.4
xl_boundary_type = "outflow"
xr_boundary_type = "outflow"
yl_boundary_type = "outflow"
yr_boundary_type = "outflow"
plot_var_1 = "dens"
plot_var_2 = "temp"
plot_var_3 = "pres"
sim_profFileName = "/dev/null"
```

- 编写脚本。在此目录下编写以下 `flash.slurm` 脚本，其中 `(path_to_installation_flash)` 为 `flash` 安装路径，

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --time=1-00:00:00
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load mpich/3.4.2-gcc-9.2.0 hdf5/1.10.6-gcc-9.2.0

ulimit -s unlimited
ulimit -l unlimited

mpirun -np $SLURM_NTASKS (path_to_installation_flash)/object/flash4 -par_
↪file flash.par
```

- 使用 sbatch 提交作业:

```
sbatch flash.slurm
```

- 编译结束后在当前目录下生成如下文件:

```
./
├─ amr_runtime_parameters.dump
├─ flash.par
├─ flash.slurm
├─ LargestSummary.out
├─ sedov.dat
├─ sedov_forced_hdf5_plt_cnt_0000
├─ sedov_hdf5_chk_0000
├─ sedov_hdf5_chk_0001
├─ sedov_hdf5_chk_0002
├─ sedov_hdf5_chk_0003
├─ sedov_hdf5_chk_0004
├─ sedov_hdf5_chk_0005
├─ sedov_hdf5_plt_cnt_0000
└─ sedov.log
```

colmap

简介

COLMAP is a general-purpose, end-to-end image-based 3D reconstruction pipeline (i.e., Structure-from-Motion (SfM) and Multi-View Stereo (MVS)) with a graphical and command-line interface. It offers a wide range of features for reconstruction of ordered and unordered image collections. The software runs under Windows, Linux and Mac on regular desktop computers or compute servers/clusters.

COLMAP 是一种通用的运动结构 (SfM) 和多视图立体 (MVS) 管道，具有图形和命令行界面。它为有序和无序图像集合的重建提供了广泛的功能。该软件是在新的 BSD 许可下获得许可的。

使用 **conda** 在集群上安装 **colmap**

可以使用以下命令在超算上安装 **colmap**。

```
$ srun -p small -n 4 --pty /bin/bash
$ module load miniconda3
$ conda create -n env4colmap
$ source activate env4colmap
$ conda install -c conda-forge colmap
```

pymultinest

简介

PyMultiNest library provides programmatic access to MultiNest and PyCuba.

使用 **conda** 在集群上安装 **PyMultiNest**

可以使用以下命令在超算上安装 **PyMultiNest**。

```
$ srun -p small -n 4 --pty /bin/bash
$ module load miniconda3
$ conda create -n env4PyMultiNest -y
$ source activate env4PyMultiNest
$ conda install -c conda-forge pymultinest mpi4py -y
```

LORENE

简介

LORENE 是一组 C++ 类，用于解决数值相对论和计算天体物理学中出现的各种问题。该软件提供了通过多域谱方法求解偏微分方程的工具。

可用的版本

版本	平台	构建方式	容器位置
2006		容器	/dssg/share/imgs/lorene/lorene.sif 思源一号
2006		容器	/lustre/share/img/x86/lorene/lorene.sif

LORENE 的使用

- 思源一号 *LORENE*
- $\pi 2.0$ *LORENE*

一. 思源一号 **LORENE**

```
singularity shell /dssg/share/imgs/lorene/lorene.sif  
# 库文件位置：/usr/lib/x86_64-linux-gnu
```

二. $\pi 2.0$ **LORENE**

```
singularity shell /lustre/share/img/x86/lorene/lorene.sif  
# 库文件位置：/usr/lib/x86_64-linux-gnu
```

参考资料

- **LORENE 官网**

BoltzTraP

简介

BoltzTraP (Boltzmann Transport Properties) 是一个计算半经典输运系数的程序。

软件下载链接

```
# 包含算例  
https://owncloud.tuwien.ac.at/index.php/s/s2d55LYlZnioa3s/download
```

算例所在目录

```
/path/download/boltztrap-1.2.5/tests
```

集群上的 **BoltzTraP**

- 思源一号 *BoltzTraP*
- $\pi 2.0$ *BoltzTraP*

思源一号上的 **BoltzTraP**

运行脚本如下所示：

```
#!/bin/bash
#SBATCH --job-name=boltztrap
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load BoltzTraP
#使用 tests 目录下的 AI 算例
BoltzTraP BoltzTraP.def
```

π2.0 上的 BoltzTraP

运行脚本如下所示：

```
#!/bin/bash
#SBATCH --job-name=BoltzTraP
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load BoltzTraP
#使用 tests 目录下的 AI 算例
BoltzTraP BoltzTraP.def
```

运行结果 (单位为: **s**)

```
===== BoltzTraP vs 1.2.5 =====
AI
XXXXXXXXX
  1.68966258649651      -0.503059854949352      5.000000000000000E-004
↪ npoints
    4386
```

参考资料

- BoltzTraP 官网

Croco

简介

CROCO 是一个基于 ROMS_AGRIF 构建的海洋建模系统，由 IRD、INRIA、CNRS、IFREMER 和 SHOM（法国致力于环境科学和应用数学的机构）维护。CROCO 的一个重要目标是解决非常精细的尺度（特别是在沿海地区）及其与更大尺度的相互作用。它

包括非静水力求解器、海浪-大气耦合、不断演变的沉积物动力学和海洋生物地球化学、用于平流和混合的新高阶数值方案以及专用 I/O 服务器 (XIOS) 等新功能。源代码附带了一个用于预处理和后处理的工具箱 CROCO_TOOLS。

自行安装 **croco** 步骤，以在 **Pi2.0 KOS** 系统中安装为例

下载 **croco** 与 **croco_tools** 源代码并解压

```
$ wget https://data-croco.ifremer.fr/CODE_ARCHIVE/croco-v1.3.tar.gz
$ wget https://data-croco.ifremer.fr/CODE_ARCHIVE/croco_tools-v1.3.tar.gz
$ tar zxvf croco-v1.3.tar.gz
$ mv croco-v1.3 croco
$ tar croco_tools-v1.3.tar.gz
$ mv croco_tools-v1.3 croco_tools
```

新建编译文件夹 **Run**

```
$ cd croco
$ ./create_config.bash
```

修改编译文件 **cppdefs.h**，启用并行计算，根据用户需求设置配置类型，例如修改为 **Basin**

```
$ cd Run
$ vim cppdefs.h
$ # /* Parallelization */选项：修改`# undef MPI` 为 `# define MPI`
$ # /* Configuration Name */选项：修改`# define BENGUELA_LR` 为 `# undef_
↪BENGUELA_LR # define Basin`
```

修改编译文件 **param.h**，设置并行计算调用的核数，例如修改为 512 核。

```
$ vim param.h
$ # 修改 `parameter (NP_XI=1, NP_ETA=4, NNODES=NP_XI*NP_ETA)` 为_
↪`parameter (NP_XI=16, NP_ETA=32, NNODES=NP_XI*NP_ETA)`
```

注意：该软件提交计算调用的核数需要与 **param.h** 设置的核数一致，核数 $np=NNODES=NP\_XI*NP\_ETA$

修改编译文件 **jobcomp**

```
$ vim jobcomp # 修改如下内容：
$ FC=$FC
$ MPIF90=$MPIF90
$ MPIDIR=$(dirname $(dirname $(which $MPIF90) ))
$ MPILIB="-L$MPIDIR/lib -lmpi -limf -lm"
$ MPIINC="-I$MPIDIR/include"
```

申请计算节点，调用编译软件，声明编译器

```
$ srun -p cpu -n 8 --pty /bin/bash
$ module load oneapi/2021.4.0
```

(续下页)

(接上页)

```
$ export CC=icc
$ export FC=ifort
$ export F90=ifort
$ export F77=ifort
$ export MPIF90=mpiifort
```

开始编译

```
$ ./jobcomp > jobcomp.log
```

编译完成后生成可执行文件 **croco** 以及包含源文件的文件夹 **Compile**

提交任务脚本，以 **Pi2.0** 集群为例

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 13
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited
ulimit -n 4096

module load oneapi/2021.4.0
module load netcdf-fortran/4.5.2-intel-2021.4.0

mpirun -np 512 ./croco croco.in #提交计算的核数需要与param.
↪h编译文件设置的核数一致！
```

参考资料

- [croco](#)
- [croco documentation](#)

Openseespy

简介

OpenSeesPy is free for research, education, and internal use. Commercial redistribution of OpenSeesPy, such as, but not limited to, an application or cloud-based service that uses import openseespy, requires a license similar to that required for commercial redistribution of OpenSees.exe.

使用 **conda** 在集群上安装 **openseespy**

以在 **Pi2.0** 集群安装为例

```
$ srun -p cpu -n 8 --pty /bin/bash
$ module load miniconda3
$ conda create -n openseespy python=3.6.8
$ source activate openseespy
$ pip install openseespy==3.2.2.6
```

提交 **openseespy** 多核并行计算的脚本:

```
#!/bin/bash
#SBATCH -J hello-python
#SBATCH -p cpu
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=40

module load miniconda3
source activate openseespy
module load mpich
```

`mpiexec -np 40 python trial.py`

参考资料

- [Openseespy docs](#)
- [Openseespy github](#)

WannierTools

简介

WannierTools, a software for investigation of novel topological materials. This code works in the tight-binding framework, which can be generated by another software package Wannier90. It can help to classify the topological phase of a given materials by calculating the Wilson loop, and can get the surface state spectrum which is detected by angle resolved photoemission (ARPES) and in scanning tunneling microscopy (STM) experiments . It also identifies positions of Weyl/Dirac points and nodal line structures, calculates the Berry phase around a closed momentum loop and Berry curvature in a part of the Brillouin zone(BZ). Besides, WannierTools also can calculate ordinary magnetoresistance for non-magnetic metal and semimetal using Boltzmann transport theory, calculate Landau level spectrum with given magnetic field direction and strength, and get unfolded energy spectrum from a supercell calculation.

在集群上安装 WannierTools

以在 Pi2.0 集群安装为例

```
$ git clone https://github.com/quanshengwu/wannier_tools.git
$ cd wannier_tools/src
$ cp Makefile.intel-mpi Makefile
$ srun -p cpu -n 8 --pty /bin/bash
$ module load oneapi/2021.4.0
$ module load arpack-ng/3.9.0-gcc-8.5.0-openblas
```

修改编译文件 Makefile

```
$ ARPACK=-L/lustre/opt/cascadelake/linux-rhel8-skylake_avx512/gcc-8.5.0/
→arpack-ng-3.9.0-45jiidktw4kpjdk54loswx5tcjs5udez/lib64 -larpack
```

开始编译，完成后在 wannier_tools/bin 文件夹中生成可执行文件 wt.x

```
$ make
```

提交 WannierTools 多核并行计算的脚本：

```
#!/bin/bash
#SBATCH -J hello-python
#SBATCH -p cpu
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=40

module load oneapi/2021.4.0
module load arpack-ng/3.9.0-gcc-8.5.0-openblas

mpirun -np 40 wt.x &
```

参考资料

- [WannierTools github](#)

NAMD

简介

NAMD (Nanoscale Molecular Dynamics) 是一个高性能并行分子动力学模拟软件，广泛用于模拟生物大分子的运动，如蛋白质、核酸和膜。它的设计目标是能够处理非常大的系统，并且支持多种计算平台，从桌面工作站到超级计算机。NAMD 通常与 VMD (Visual Molecular Dynamics) 配合使用，用于前处理和可视化模拟结果。

可用的版本

版本	平台	构建方式	集群	模块名
3.0b6		源码	Pi 2.0	namd/3.0b6-gcc-11.2.0
3.0b6		源码	思源一号	namd/3.0b6-gcc-11.2.0
3.0b6		源码	ARM	namd/3.0b6-gcc-10.3.1-openmpi

作业脚本示例

思源一号

```
#!/bin/bash

#SBATCH --job-name=namd
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load namd/3.0b6-gcc-11.2.0

mpirun -np $SLURM_NPROCS namd3 apoa1.namd
```

Pi 2.0

```
#!/bin/bash

#SBATCH --job-name=namd
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load namd/3.0b6-gcc-11.2.0

mpirun -np $SLURM_NPROCS namd3 apoa1.namd
```

ARM

```
#!/bin/bash

#SBATCH --job-name=namd
#SBATCH --partition=arm128c256g
#SBATCH -N 2
#SBATCH --ntasks-per-node=128
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load namd/3.0b6-gcc-10.3.1-openmpi

mpirun -np $SLURM_NPROCS namd3 apoa1.namd
```

软件使用流程

获取算例

```
wget https://www.ks.uiuc.edu/Research/namd/utilities/apoa1.tar.gz
tar -xvf apoa1.tar.gz
cd apoa1
```

算例介绍

ApoA1 benchmark 是 NAMD 中一个常用的基准测试算例，常用于评估高性能计算平台上 NAMD 的计算性能。这个基准测试模拟了载脂蛋白 A-I (Apolipoprotein A-I, ApoA1) 的分子动力学，ApoA1 是高密度脂蛋白 (HDL) 的主要蛋白质成分，与胆固醇代谢和心血管健康密切相关。

系统组成: ApoA1 benchmark 系统包含 92,224 个原子，模型中包括一个 ApoA1 蛋白质环绕着 3000 个水分子，以及 120 个脂肪酸分子。这些分子构成了一个水溶液环境中的 HDL 磷脂双分子层。

准备作业脚本，以思源一号为例：

```
#!/bin/bash

#SBATCH --job-name=namd
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

(续下页)

(接上页)

```
module purge
module load namd/3.0b6-gcc-11.2.0

mpirun -np $SLURM_NPROCS namd3 apoa1.namd
```

提交作业

```
sbatch run.slurm
```

查看结果

```
WRITING EXTENDED SYSTEM TO OUTPUT FILE AT STEP 500
WRITING COORDINATES TO OUTPUT FILE AT STEP 500
The last position output (seq=-2) takes 0.170 seconds, 59.409 MB of memory_
→in use
WRITING VELOCITIES TO OUTPUT FILE AT STEP 500
The last velocity output (seq=-2) takes 0.051 seconds, 59.409 MB of memory_
→in use
=====

WallClock: 30.385077  CPUTime: 30.385077  Memory: 59.408632 MB
[Partition 0][Node 0] End of program
```

参考资料

- [NAMD 官网](#)

Xerus

简介

The xerus library is a general purpose library for numerical calculations with higher order tensors, Tensor-Train Decompositions / Matrix Product States and other Tensor Networks. The focus of development was the simple usability and adaptibility to any setting that requires higher order tensors or decompositions thereof.

使用 **conda** 在集群上安装 **Xerus**

以在 Pi2.0 集群安装为例

```
$ srun -p cpu -n 8 --pty /bin/bash
$ module load miniconda3
$ conda create -n xerus python=3.7
$ source activate xerus
$ conda install -c rotekekse -c conda-forge xerus
```

参考资料

- [Xerus docs](#)
- [Xerus gitlab](#)

packmol

简介

Packmol 是一款开源软件，专门用于创建分子动力学（MD）模拟的初始配置。它的核心功能是将分子精确地放置在预定义的空间区域内，以确保短程排斥相互作用不会干扰模拟过程。**Packmol** 能够处理各种复杂的分子排列，例如层状、球形或管状的脂质层，并且支持多种输入文件格式，如 PDB、TINKER、XYZ 和 MOLDY，这使得它具有广泛的使用范围。

可用的版本

集群	平台	版本	模块名
思源一号		20.0.0	packmol/20.0.0-gcc-11.2.0
Pi 2.0		20.0.0	packmol/20.0.0-gcc-8.5.0

集群上的 **Packmol**

- 思源一号 *Packmol*
- *Pi 2.0 Packmol*

思源一号 **Packmol**

slurm 作业脚本示例：

```
#!/bin/bash
#SBATCH --job-name=packmol
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load packmol/20.0.0-gcc-11.2.0

packmol < packmol.inp
```


π2.0 Packmol

slurm 作业脚本示例:

```
#!/bin/bash

#SBATCH --job-name=packmol
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load packmol/20.0.0-gcc-8.5.0

packmol < packmol.inp
```

运行官方算例

获取算例

```
wget https://m3g.github.io/packmol/examples/examples.tar.gz
tar -xvf examples.tar.gz
cd examples
```

算例信息: **bilayer**

```
Lipid bilayer with water over and below
Commented input file: bilayer-comment.inp
Clean input file: bilayer.inp
Alternative input file using different constraints: bilayer2.inp
Molecules needed: palmitoil.pdb water.pdb
Output file: bilayer.pdb
Running time: 26 seconds
Number of atoms: 7200
Number of molecules: 1100
Number of variables: 6600
```

思源一号

```
#!/bin/bash
#SBATCH --job-name=packmol
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
```

(续下页)

(接上页)

```
#SBATCH --error=%j.err

module purge
module load packmol/20.0.0-gcc-11.2.0

packmol < bilayer.inp
```

π2.0

```
#!/bin/bash

#SBATCH --job-name=packmol
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module purge
module load packmol/20.0.0-gcc-8.5.0

packmol < bilayer.inp
```

计算完成部分日志

```
#####
→####
Solution written to file: bilayer.pdb
-----
→----
Running time:      24.3353214      seconds.
-----
→----
```

参考资料

- [Packmol 官网](#)

lobster

软件简介

Lobster 是一种用于计算局域轨道基态轨道结合能的电子结构分析软件工具，主要用于分析固体材料中的化学键和电子结构。它的名称“Lobster”是“Local Orbital Basis Suite Towards Electronic-Structure Reconstruction”的缩写。

可用的版本

版本	平台	构建方式	集群	模块名
5.1.0		源码	Pi2.0	lobster/5.1-gcc-8.5.0
5.1.0		源码	思源一号	lobster/5.1-gcc-8.5.0

作业脚本示例

思源一号

```
#!/bin/bash

#SBATCH -J vasp
#SBATCH -p 64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load vasp/5.4.4-intel-2021.4.0
module load lobster/5.1-gcc-8.5.0

ulimit -s unlimited
mpirun vasp_std

export OMP_NUM_THREADS=64
lobster-5.1.0
```

Pi 2.0

```
#!/bin/bash

#SBATCH -J vasp
#SBATCH -p cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load vasp/5.4.4-intel-2021.4.0
module load lobster/5.1-gcc-8.5.0
```

(续下页)

(接上页)

```
ulimit -s unlimited
mpirun vasp_std

export OMP_NUM_THREADS=40
lobster-5.1.0
```

软件简单使用流程

获取算例

```
module load lobster/5.1.0
cp -rfv $lobster_vasp_example .
cd fullerene
```

算例介绍

以官方提供的 VASP 算例的富勒烯 (fullerene) 为例，官方算例已提供 VASP 算例所需的计算输入文件，需要先使用 VASP 进行 DFT 计算，然后再进行 LOBSTER 的计算，下面示例为在思源一号使用 1 节点 64 核心进行计算，由于 LOBSTER 不支持跨节点并行，需要设置 OMP 线程参数。

准备作业脚本

```
#!/bin/bash

#SBATCH -J vasp
#SBATCH -p 64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module purge
module load vasp/5.4.4-intel-2021.4.0
module load lobster/5.1-gcc-8.5.0

ulimit -s unlimited
mpirun vasp_std

export OMP_NUM_THREADS=64
lobster-5.1.0
```

提交作业

```
sbatch run.slurm
```

查看结果

```
writing SitePotentials.lobster and MadelungEnergies.lobster...
finished in 0 h 10 min 14 s 484 ms of wall time
          10 h 9 min 37 s 340 ms of user time
          0 h 10 min 33 s 560 ms of sys time
```

参考资料

- LOBSTER

libgd (GD Library)

简介

libgd 是一个用于动态图形生成的跨平台函数库。它用于创建、操作图像，并且可以嵌入到 **PHP**、**Python** 等语言中。**libgd** 支持多种图像格式，如 **PNG**、**JPEG**、**GIF** 等，并提供了丰富的绘图功能，如线条、多边形、曲线以及文本等。

i 备注

libgd 不仅可以用于 **Web** 应用，也可以集成到桌面应用程序中，为用户提供图像处理的功能。

使用方法

libgd 提供了广泛的 **API** 来创建和修改图像，以下是在思源 1 号上通过 **module** 工具调用 **libgd** 并编译示例程序：

- 制作两帧 **GIF** 动图:*test.cpp*

```
#include <gd.h>
#include <gdfonts.h>
#include <stdio.h>
#include <string.h>

int main() {
    // 创建GIF文件
    FILE *gifout = fopen("output.gif", "wb");
    if (!gifout) {
        fprintf(stderr, "无法打开文件以保存GIF。\\n");
        return 1;
    }
}
```

(续下页)

(接上页)

```

// 创建两帧图像
int width = 200;
int height = 100;

// 创建第一帧图像
gdImagePtr image = gdImageCreate(width, height);

// 定义颜色
int white = gdImageColorAllocate(image, 255, 255, 255); // 背景白色
int black = gdImageColorAllocate(image, 0, 0, 0); // 文本黑色

// 填充背景
gdImageFilledRectangle(image, 0, 0, width - 1, height - 1, white);

// 绘制第一帧文本
const char *text1 = "Frame 1";
gdFontPtr font = gdFontGetSmall();
int x1 = (width - gdFontGetSmall()->w * strlen(text1)) / 2;
int y1 = (height - gdFontGetSmall()->h) / 2;
gdImageString(image, font, x1, y1, (unsigned char *)text1, black);

// 开始GIF动画
gdImageGifAnimBegin(image, gifout, 1, 0); // 使用全局颜色表，循环无限

// 添加第一帧
gdImageGifAnimAdd(image, gifout, 0, 0, 0, 0, 0, 0);

// 创建第二帧图像
gdImagePtr image2 = gdImageCreate(width, height);

// 填充背景
gdImageFilledRectangle(image2, 0, 0, width - 1, height - 1, white);

// 绘制第二帧文本
const char *text2 = "Frame 2";
int x2 = (width - gdFontGetSmall()->w * strlen(text2)) / 2;
int y2 = (height - gdFontGetSmall()->h) / 2;
gdImageString(image2, font, x2, y2, (unsigned char *)text2, black);

// 添加第二帧
gdImageGifAnimAdd(image2, gifout, 0, 0, 0, 0, 0, 0);

// 结束GIF动画
gdImageGifAnimEnd(gifout);

```

(续下页)

(接上页)

```
// 清理
gdImageDestroy(image);
gdImageDestroy(image2);
fclose(gifout);
printf("GIF图像已保存为output.gif\n");
return 0;
}
```

- 调用 libgd 并编译示例程序:

```
module purge
module load gcc/12.3.0
module load libgd/2.3.3-gcc-12.3.0

g++ -o test test.cpp -lgd
```

- 执行程序:

```
./test
```

参考链接

- [libgd 官方网站](#)

请根据实际的文档需求调整以上内容。如果使用的是特定编程语言（如 PHP、Python），则需要相应地调整示例代码。

4.10.6 生物信息计算

AlphaFold2

AlphaFold2 基于深度神经网络预测蛋白质形态，能够快速生成高精确度的蛋白质 3D 模型。以往花费几周时间预测的蛋白质结构，AlphaFold2 在几小时内就能完成

我们对 AlphaFold 进行持续优化，欢迎了解我们的优化工作：[ParaFold: Paralleling AlphaFold for Large-Scale Predictions](#)

下面将介绍 AlphaFold2, ParaFold, ColabFold 在思源一号和 π 集群的使用，以及如何在 A100 上自行构建 conda 环境，和建立标准镜像

AlphaFold2 三大版本

交大计算平台提供 AlphaFold2 三大版本：基础运行版、ParaFold、ColabFold。三个版本在思源一号和 π 集群上均可使用，且都支持复合体计算：

- 基础运行版，可满足大部分计算需求
- ParaFold，支持 CPU、GPU 分离计算，适合大规模批量计算
- ColabFold，快速计算，含多种功能，由 Sergey Ovchinnikov 等人开发

使用前准备

- 新建文件夹，如 `alphafold`。
- 在文件夹里放置一个 `fasta` 文件。例如 `test.fasta` 文件（内容如下）：
（单体 `fasta` 文件示例）

```
>2LHC_1|Chain A|Ga98|artificial gene (32630)
PIAQIHILEGRSDEQKETLIREVSEAIRSLDAPLTSVRVIITEMAKGHFGIGGELASK
```

（复合体 `fasta` 文件示例）

```
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
```

版本一：基础运行版

可用的版本

版本	平台
2.3.1	思源一号
2.2.0	思源一号
2.1.1	思源一号
2.3.1	pi 2.0
2.1.1	pi 2.0

运行平台

- 一. 思源一号 *AlphaFold*
- 二. π 2.0 *AlphaFold*

思源一号 **AlphaFold****AlphaFold2.3.1** 计算单体

```
#!/bin/bash
#SBATCH --job-name=alphafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1 # use 1 GPU
```

(续下页)

(接上页)

```
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export DOWNLOAD_DIR=/dssg/share/data/alphafold
singularity exec --nv /dssg/share/imgs/ai/alphafold/alphafold-2.3.1.sif_
↪python /app/alphafold/run_alphafold.py \
--use_gpu_relax \
--data_dir=$DOWNLOAD_DIR \
--uniref90_database_path=$DOWNLOAD_DIR/uniref90/uniref90.fasta \
--mgnify_database_path=$DOWNLOAD_DIR/mgnify/mgy_clusters.fa \
--bfd_database_path=$DOWNLOAD_DIR/bfd/bfd_metaclust_clu_complete_id30_c90_
↪final_seq.sorted_opt \
--pdb70_database_path=$DOWNLOAD_DIR/pdb70/pdb70 \
--uniref30_database_path=$DOWNLOAD_DIR/uniref30/UniRef30_2021_03 \
--template_mmcif_dir=$DOWNLOAD_DIR/pdb_mmcif/mmcif_files \
--obsolete_pdbs_path=$DOWNLOAD_DIR/pdb_mmcif/obsolete.dat \
--model_preset=monomer \
--max_template_date=2022-10-1 \
--db_preset=full_dbs \
--output_dir=output \
--fasta_paths=monomer.fasta
```

AlphaFold2.3.1 计算复合体

```
#!/bin/bash
#SBATCH --job-name=alphafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export DOWNLOAD_DIR=/dssg/share/data/alphafold
singularity exec --nv /dssg/share/imgs/ai/alphafold/alphafold-2.3.1.sif_
↪python /app/alphafold/run_alphafold.py \
--use_gpu_relax \
--data_dir=$DOWNLOAD_DIR \
--uniref90_database_path=$DOWNLOAD_DIR/uniref90/uniref90.fasta \
--mgnify_database_path=$DOWNLOAD_DIR/mgnify/mgy_clusters.fa \
--bfd_database_path=$DOWNLOAD_DIR/bfd/bfd_metaclust_clu_complete_id30_c90_
↪final_seq.sorted_opt \
--uniref30_database_path=$DOWNLOAD_DIR/uniref30/UniRef30_2021_03 \
--pdb_seqres_database_path=$DOWNLOAD_DIR/pdb_seqres/pdb_seqres.txt \
--template_mmcif_dir=$DOWNLOAD_DIR/pdb_mmcif/mmcif_files \
--obsolete_pdbs_path=$DOWNLOAD_DIR/pdb_mmcif/obsolete.dat \
--uniprot_database_path=$DOWNLOAD_DIR/uniprot/uniprot.fasta \
```

(续下页)

(接上页)

```
--model_preset=multimer \
--max_template_date=2022-10-1 \
--db_preset=full_dbs \
--output_dir=output \
--fasta_paths=multimer.fasta
```

AlphaFold2.1.1 计算单体

```
#!/bin/bash
#SBATCH --job-name=alphafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module use /dssg/share/imgs/ai/
module load alphafold/2.1.1

singularity run --nv /dssg/share/imgs/ai/alphafold/2.1.1.sif python /app/
→alphafold/run_alphafold.py \
--data_dir=/dssg/share/data/alphafold \
--bfd_database_path=/dssg/share/data/alphafold/bfd/bfd_metaclust_clu_
→complete_id30_c90_final_seq.sorted_opt \
--uniclust30_database_path=/dssg/share/data/alphafold/uniclust30/uniclust30_
→2018_08/uniclust30_2018_08 \
--uniref90_database_path=/dssg/share/data/alphafold/uniref90/uniref90.fasta_
→\
--mgnify_database_path=/dssg/share/data/alphafold/mgnify/mgy_clusters.fa \
--pdb70_database_path=/dssg/share/data/alphafold/pdb70/pdb70 \
--template_mmcif_dir=/dssg/share/data/alphafold/pdb_mmcif/mmcif_files \
--obsolete_pdbs_path=/dssg/share/data/alphafold/pdb_mmcif/obsolete.dat \
--fasta_paths=test.fasta \
--max_template_date=2020-05-14 \
--model_preset=monomer \
--output_dir=output
```

AlphaFold2.2.0 计算复合体

```
#!/bin/bash
#SBATCH --job-name=alphafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
```

(续下页)

(接上页)

```
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export DOWNLOAD_DIR=/dssg/share/data/alphafold
singularity exec --nv /dssg/share/imgs/ai/alphafold/2.2.0.sif python /app/
↪alphafold/run_alphafold.py \
--use_gpu_relax \
--data_dir=$DOWNLOAD_DIR \
--uniref90_database_path=$DOWNLOAD_DIR/uniref90/uniref90.fasta \
--mgnify_database_path=$DOWNLOAD_DIR/mgnify/mgy_clusters_2018_12.fa \
--bfd_database_path=$DOWNLOAD_DIR/bfd/bfd_metaclust_clu_complete_id30_c90_
↪final_seq.sorted_opt \
--uniclust30_database_path=$DOWNLOAD_DIR/uniclust30/uniclust30_2020_06/
↪UniRef30_2020_06 \
--pdb_seqres_database_path=$DOWNLOAD_DIR/pdb_seqres/pdb_seqres.txt \
--template_mmcif_dir=$DOWNLOAD_DIR/pdb_mmcif/mmcif_files \
--obsolete_pdbs_path=$DOWNLOAD_DIR/pdb_mmcif/obsolete.dat \
--uniprot_database_path=$DOWNLOAD_DIR/uniprot/uniprot.fasta \
--model_preset=multimer \
--max_template_date=2022-1-1 \
--db_preset=full_dbs \
--output_dir=output \
--fasta_paths=x.fasta
```

π2.0 AlphaFold

AlphaFold2.3.1 计算单体

```
#!/bin/bash
#SBATCH --job-name=alphafold_monomer
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1 # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export DOWNLOAD_DIR=/lustre/share/scidata/AlphaFold/data
singularity exec --nv /lustre/share/img/ai/alphafold/alphafold-2.3.1.sif_
↪python /app/alphafold/run_alphafold.py \
--use_gpu_relax \
--data_dir=$DOWNLOAD_DIR \
--uniref90_database_path=$DOWNLOAD_DIR/uniref90/uniref90.fasta \
--mgnify_database_path=$DOWNLOAD_DIR/mgnify/mgy_clusters.fa \
--bfd_database_path=$DOWNLOAD_DIR/bfd/bfd_metaclust_clu_complete_id30_c90_
↪final_seq.sorted_opt \
```

(续下页)

(接上页)

```
--pdb70_database_path=$DOWNLOAD_DIR/pdb70/pdb70 \
--uniref30_database_path=$DOWNLOAD_DIR/uniref30/UniRef30_2021_03 \
--template_mmcif_dir=$DOWNLOAD_DIR/pdb_mmcif/mmcif_files \
--obsolete_pdbs_path=$DOWNLOAD_DIR/pdb_mmcif/obsolete.dat \
--model_preset=monomer \
--max_template_date=2022-10-1 \
--db_preset=full_dbs \
--output_dir=output \
--fasta_paths=monomer.fasta
```

AlphaFold2.3.1 计算复合体

```
#!/bin/bash
#SBATCH --job-name=alphafold
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export DOWNLOAD_DIR=/lustre/share/scidata/AlphaFold/data
singularity exec --nv /lustre/share/img/ai/alphafold/alphafold-2.3.1.sif_
python /app/alphafold/run_alphafold.py \
--use_gpu_relax \
--data_dir=$DOWNLOAD_DIR \
--uniref90_database_path=$DOWNLOAD_DIR/uniref90/uniref90.fasta \
--mgnify_database_path=$DOWNLOAD_DIR/mgnify/mgy_clusters.fa \
--bfd_database_path=$DOWNLOAD_DIR/bfd/bfd_metaclust_clu_complete_id30_c90_
final_seq.sorted_opt \
--uniref30_database_path=$DOWNLOAD_DIR/uniref30/UniRef30_2021_03 \
--pdb_seqres_database_path=$DOWNLOAD_DIR/pdb_seqres/pdb_seqres.txt \
--template_mmcif_dir=$DOWNLOAD_DIR/pdb_mmcif/mmcif_files \
--obsolete_pdbs_path=$DOWNLOAD_DIR/pdb_mmcif/obsolete.dat \
--uniprot_database_path=$DOWNLOAD_DIR/uniprot/uniprot.fasta \
--model_preset=multimer \
--max_template_date=2022-10-1 \
--db_preset=full_dbs \
--output_dir=output \
--fasta_paths=multimer.fasta
```

AlphaFold2.1.1 计算单体

单体

```
#!/bin/bash
#SBATCH --job-name=alphafold
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH -x vol04,vol05
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load alphafold

af2.1 \
--fasta_paths=test.fasta \
--max_template_date=2020-05-14 \
--model_preset=monomer \
--output_dir=output
```

然后使用 `sbatch sub.slurm` 语句提交作业。

AlphaFold2.1.1 计算复合体

复合体

(slurm 脚本开头的 13 行内容跟上方一样，请自行补齐)

```
af2.1 \
--fasta_paths=test.fasta \
--max_template_date=2020-05-14 \
--model_preset=multimer \
--is_prokaryote_list=false \
--output_dir=output
```

module 使用说明

- 单体计算可选用 `monomer`, `monomer_ptm`, 或 `monomer_casp14`
- π 集群上的 AlphaFold module 需严格按照推荐的参数内容和顺序运行（调换参数顺序或增删参数条目均可能导致报错）。若需使用更多模式，请换用思源一号 module 版本，或使用 ParaFold
- 更多使用方法及讨论，请见水源文档 [AlphaFold & ColabFold](#)

版本二：ParaFold

ParaFold 为交大开发的适用于大规模计算的 AlphaFold 集群版，可选 CPU 与 GPU 分离计算，并支持 Amber 选择、module 选择、Recycling 次数指定等多个实用功能。ParaFold 并不改变 AlphaFold 计算内容和参数本身，所以在计算结果及精度上与 AlphaFold 完全一致

ParaFold (又名 ParallelFold) 将原本全部运行于 GPU 的计算, 分拆为 CPU 和 GPU 两阶段进行。先至 CPU 节点完成 MSA 计算, 再用 GPU 节点完成模型预测。这样既能节省 GPU 资源, 又能加快运算速度

ParaFold GitHub: <https://github.com/Zuricho/ParallelFold>

介绍网站: <https://parafold.sjtu.edu.cn>

ParaFold 可用版本

版本	平台
2.0	思源一号
1.0	思源一号
1.0	pi 2.0

ParaFold 在思源一号上运行

ParaFold2.0 计算单体运行脚本

```
#!/bin/bash
#SBATCH --job-name=parafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity run --nv /dssg/share/imgs/parafold/2.0.sif \
  /app/ParallelFold/run_alphafold.sh \
  -d /dssg/share/data/alphafold \
  -o output \
  -p monomer \
  -i input/GA98.fasta \
  -t 2021-07-27 \
  -m model_1,model_2,model_3,model_4,model_5
```

ParaFold2.0 计算多体运行脚本

```
#!/bin/bash
#SBATCH --job-name=parafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
```

(续下页)

(接上页)

```
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity run --nv /dssg/share/imgs/parafold/2.0.sif \
  /app/ParallelFold/run_alphafold.sh \
  -d /dssg/share/data/alphafold \
  -o output \
  -p multimer \
  -i multimer.fasta \
  -m model_1_multimer_v3,model_2_multimer_v3,model_3_multimer_v3,model_4_
  ↪multimer_v3,model_5_multimer_v3 \
  -u 0 \
  -t 2022-01-01 -r
```

ParaFold1.0 计算单体运行脚本

```
#!/bin/bash
#SBATCH --job-name=parafold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH -x gpu01,gpu03,gpu05
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity run --nv /dssg/share/imgs/ai/fold/1.0.sif \
  ./run_alphafold.sh \
  -d /dssg/share/data/alphafold \
  -o output \
  -p monomer \
  -i input/GA98.fasta \
  -t 2021-07-27 \
  -m model_1,model_2,model_3,model_4,model_5
```

ParaFold 在 π 集群上运行

ParaFold1.0 脚本

下载 ParaFold

```
git clone https://github.com/Zuricho/ParallelFold.git
cd ParallelFold
chmod +x run_alphafold.sh
```

使用下方 sub.slurm 脚本直接运行：

```
#!/bin/bash
#SBATCH --job-name=parafold
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity run --nv /lustre/share/img/ai/fold.sif \
./run_alphafold.sh \
-d /scratch/share/AlphaFold/data \
-o output \
-p monomer_ptm \
-i input/GA98.fasta \
-t 2021-07-27 \
-m model_1
```

ParaFold2.0 示例脚本

```
#!/bin/bash
#SBATCH --job-name=parafold
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity run --nv /lustre/share/img/ai/parafold2.sif \
/app/ParallelFold/run_alphafold.sh \
-d /scratch/share/AlphaFold/data \
-o output \
-p monomer_ptm \
-i /app/ParallelFold/input/mono_set1/GA98.fasta \
-t 2024-01-29 \
-m model_1
```

版本三: ColabFold

ColabFold 为 Sergey Ovchinnikov 等人开发的适用于 Google Colab 的 AlphaFold 版本, 使用 MMseqs2 替代 Jackhmmer, 且不使用模版。ColabFold 计算迅速, 短序列五六分钟即可算完。

ColabFold 使用请至交大超算文档页面: [ColabFold](#)

构建自己的 **conda** 环境

交大计算平台已全局部署适用于 AlphaFold, ParaFold 和 ColabFold 的 fold 镜像。同时也支持大家根据需求，自行创建 **conda** 环境。

下面介绍在思源一号 A100 上安装 **conda** 环境的方法。

1. 安装名为 localcolab 的 conda 环境：

```

srun -p 64c512g -n 8 --pty /bin/bash

module load miniconda3
conda create -n localcolab python=3.7 -y
source activate localcolab

conda install python=3.7 cudnn==8.2.1.32 cudatoolkit==11.1.1 openmm==7.5.1
↪pdbfixer -y
conda install -c conda-forge -c bioconda kalign3=3.2.2 hhsuite=3.3.0 -y
conda install -y -c bioconda hmmer==3.3.2 hhsuite==3.3.0 kalign3=3.2.2

pip install absl-py==0.13.0 biopython==1.79 chex==0.1.0 dm-haiku==0.0.4 dm-
↪tree==0.1.6 immutabledict==2.2.1 ml-collections==0.1.1 pandas==1.3.5
↪tensorflow_cpu==2.7.1
pip install https://storage.googleapis.com/jax-releases/cuda111/jaxlib-0.1.
↪72+cuda111-cp37-none-manylinux2010_x86_64.whl
pip install jax==0.2.25

```

1. 打补丁 openmm：

```

# patch to openmm
cd ~/.conda/envs/localcolab/lib/python3.7/site-packages
wget -qnc https://raw.githubusercontent.com/deepmind/alphafold/main/docker/
↪openmm.patch --no-check-certificate
patch -s -p0 < openmm.patch

```

至此 **conda** 环境安装完成。

3. 在此 conda 环境里运行 ParaFold：

```

salloc --ntasks-per-node=1 -p a100 --cpus-per-task=16 --gres=gpu:1 -N 1
ssh gpuXX

git clone https://github.com/Zuricho/ParallelFold.git
cd ParallelFold
chmod +x run_alphafold.sh

module load miniconda3
source activate localcolab

./run_alphafold.sh -d /dssg/share/data/alphafold -o output -p monomer -i
↪input/GA98.fasta -t 2021-07-27 -m model_1 -f

```

构建自己的 AlphaFold 镜像

交大镜像平台提供了 AlphaFold-2.1.1 的 `docker` 镜像。

使用 `singularity pull` 命令可以下载该镜像：

```
singularity pull docker://sjtu.edu.cn/x86/alphafold:<tag>
```

镜像将被保存为 `alphafold_<tag>.sif` 文件。

镜像脚本示例如下：

```
#!/bin/bash

#SBATCH -J run_af
#SBATCH -p a100
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1

singularity run --nv ${YOUR_IMAGE_PATH} python /app/alphafold/run_alphafold.
→py
    --fasta_paths=${YOU_FASTA_FILE_DIR} \
    --max_template_date=2020-05-14 \
    --bfd_database_path=${YOUR_DATA_DIR}/bfd/bfd_metaclust_clu_complete_
→id30_c90_final_seq.sorted_opt \
    --data_dir=${YOUR_DATA_DIR} \
    --output_dir=${YOU_OUTPUT_DIR} \
    --uniclust30_database_path=${YOUR_DATA_DIR}/uniclust30/uniclust30_2018_
→08/uniclust30_2018_08 \
    --uniref90_database_path=${YOUR_DATA_DIR}/uniref90/uniref90.fasta \
    --mgnify_database_path=${YOUR_DATA_DIR}/mgnify/mgy_clusters.fa \
    --template_mmCIF_dir=${YOUR_DATA_DIR}/pdb_mmCIF/mmCIF_files \
    --obsolete_pdbs_path=${YOUR_DATA_DIR}/pdb_mmCIF/obsolete.dat \
    --pdb70_database_path=${YOUR_DATA_DIR}/pdb70/pdb70
```

参考资料

- ParaFold GitHub <https://github.com/Zuricho/ParallelFold>
- ParaFold 论文: <https://arxiv.org/abs/2111.06340>
- ParaFold 网站: <https://parafold.sjtu.edu.cn>
- AlphaFold GitHub: <https://github.com/deepmind/alphafold>
- AlphaFold 论文: <https://www.nature.com/articles/s41586-021-03819-2>
- ColabFold GitHub: <https://github.com/sokrypton/ColabFold>
- LocalColabFold GitHub: <https://github.com/YoshitakaMo/localcolabfold>

- 交大 AlphaFold 镜像: <https://hub.sjtu.edu.cn/repository/x86/alphafold>

AUGUSTUS

简介

Augustus 是用作 de novo 基因注释 (即从头预测) 的软件。

AUGUSTUS is a gene prediction program for eukaryotes written by Mario Stanke and Oliver Keller. It can be used as an ab initio program, which means it bases its prediction purely on the sequence. AUGUSTUS may also incorporate hints on the gene structure coming from extrinsic sources such as EST, MS/MS, protein alignments and syntenic genomic alignments.

安装

```

srun -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n mypy #创建自己的环境
source activate mypy #进入自己的环境
conda install -c anaconda boost #安装依赖
conda install -c bioconda augustus #安装augustus

```

使用方法与范例

直接注释

若存在已经被训练的物种 (augustus --species=help 查看), 则直接使用代码进行预测基因, 以拟南芥 (arabidopsis) 为例:

```
augustus --speices=arabidopsis test.fa > test.gff
```

训练注释

若不存在被训练过的物种, 则需要训练

准备训练集与测试集

根据 Augustus 的官方教程, 应当按如下标准进行基因结构序列准备:

- 提供 gene 的编码部分, 包括上游的部分 (kb 级别)。通常情况下, 基因越多 (>200), 则效果越好, 与此同时, 外显子的数量也要足够, 以便后续训练内含子;
- 必须保证基因的起始密码子与终止密码子是准确的, 并尽量提供较为完整的 gene 结构和注释信息;
- 防止 gene 的冗余, 根据 Augustus 教程的建议, 如果任意两个 gene 在氨基酸水平上的相似度高于 70%, 那么只需保留一条。这一步既可以避免过度拟合现象, 也能用于检验预测的准确性;

- d. 多个 gene 可以在一条序列中，gene 可以在正链，亦可在负链，但 gene 之间不可存在重叠；
- e. 每个 gene 只需要一条转录本，并且以 GenBank 格式存放

这些注释数据需要随机分为训练集和测试集，为了保证测试集有统计学意义，测试集要足够多的基因（100~200 个），并且要足够的随机。

基因结构集的可能来源有：

- a. Genbank
- b. EST/mRNA-seq 的可变剪切联配，如 PASA
- c. 临近物种蛋白的可变剪切联配，如 GeneWise
- d. 相关物种的数据
- e. 预测基因的迭代训练

数据集的训练

```
# 格式转换；基于选取物种的GFF3以及ref.fa 文件将其转换为Genbank格式
perl ~/miniconda2/bin/gff2gbSmallDNA.pl ./Spinach_genome/spinach_gene_v1.
→gff3 ./Spinach_genome/spinach_genome_v1.fa 1000 genes.raw.gb

# 尝试训练，捕捉错误
etrainig --species=generic --stopCodonExcludedFromCDS=false genes.raw.gb 2>
→ train.err

# 过滤掉可能错误的基因结构
cat train.err | perl -pe 's/.*in sequence (\S+): .*/$1/' >badgenes.lst
filterGenes.pl badgenes.lst genes.raw.gb > genes.gb

# 提取上一步过滤后的genes.db中的蛋白
grep '/gene' genes.gb |sort |uniq |sed 's/\/gene=//g' |sed 's/\/\"//g' |awk '
→{print $1}' >geneSet.lst
python extract_pep.py geneSet.lst Spinach_genome/spinach_pep_v1.fa

#
→将得到的蛋白序列进行建库，自身blastp比对。根据比对结果，如果基因间identity
→>= 70%，则只保留其中之一，再次得到一个过滤后的gff文件，gene_filter.gff3
makeblastdb -in geneSet.lst.fa -dbtype prot -parse_seqids -out geneSet.lst.
→fa
blastp -db geneSet.lst.fa -query geneSet.lst.fa -out geneSet.lst.fa.blastp -
→evalue 1e-5 -outfmt 6 -num_threads 8
python delete_high_identity_gene.py geneSet.lst.fa.blastp Spinach_genome/
→spinach_gene_v1.gff3

# 将得到的gene_filter.gff3 转换为genbank 格式文件
perl ~/miniconda2/bin/gff2gbSmallDNA.pl gene_filter.gff3 ./Spinach_genome/
→spinach_genome_v1.fa 1000 genes.gb.filter
```

(续下页)

(接上页)

```
#_
→将上一步过滤后的文件随机分成两份，测试集和训练集。其中训练集的数目根据gb的LOCUS数目决定
→为测试集的基因数目，其余为训练集)
randomSplit.pl genes.gb.filter 100

# 初始化HMM参数设置（在相应~/minicode/config/species/relative_
→name中形成参数,若之前已经存在该物种名字，则需要删除），并进行训练
new_species.pl --species=spinach
etraining --species=spinach genes.gb.filter.train

#_
→用测试数据集检验预测效果，这里可以比较我们训练的结果，和近缘已训练物种的训练效果
augustus --species=spinach genes.gb.filter.test | tee firsttest.out
augustus --species=arabidopsis genes.gb.filter.test | tee firsttest_ara.out
```

训练结果的检查

在 firsttest.out 的尾部可以查看预测结果的统计，首先需要解释几个统计学概念

1. TP(True Positive): 预测为真，事实为真
2. FP(False Positive): 预测为真，事实为假
3. FN(False Negative): 预测为假，事实为真
4. TN(True Negative): 预测为假，事实为假

基于上述，引出下面两个概念：

1. "sensitivity" 等于 $TP/(TP+FP)$ (预测到的百分率)，是预测为真且实际为真的占你所有认为是真的比例；
2. "specificity" 等于 $TN/(TN+FN)$ (其中正确的百分率)，是预测为假且实际为假的占你所有认为是假的比例。我们希望在预测中，尽可能地不要发生误判，也就是没有基因的地方不要找出基因，有基因的地方不要漏掉基因。

重训练以优化结果

```
#_
→很有可能的一种情况是，我们第一次的训练结果没有已有训练的效果好，所以我们需要进行循环训练
optimize_augustus.pl --species=spinach genes.gb.filter.train

# 再次进行训练，并检验，进行前后比较
etraining --species=spinach genes.gb.filter.train
augustus --species=spinach genes.gb.filter.test | tee secondtest.out

# 如果此时你的gene level的sensitivity还是低于20%说明Training_
→set不够大，请添加数据；
# 如果你获得了满意的Training结果，请开始prediction吧
```

氨基酸序列的提取

```
# 从 firsttest.out 中提取氨基酸序列
sed -n '/^#/p' firsttest.out | sed -n '/start/,/\]/p' | sed 's/# start gene_
↪>/g;s/protein sequence \= \[/g;s/#//g;s/\[/g;s/^\s//g' >seq.fa
```

参考资料

- AUGUSTUS official website: <http://bioinf.uni-greifswald.de/augustus/>
- bioconda augustus: <https://anaconda.org/bioconda/augustus>
- 使用 MAKER 进行基因注释 (高级篇之 AUGUSTUS 模型训练): <https://www.jianshu.com/p/679bd6bb0ea4>
- Augustus 指南 (Training 部分): <https://www.cnblogs.com/southern-xyx/p/4497497.html>
- Augustus Training and Prediction: <https://www.cnblogs.com/southern-xyx/p/4497497.html>
- Augustus 进行基因注释: <https://www.cnblogs.com/zhanmaomao/p/12359964.html>

Aspera

简介

Aspera 通过使用获得专利的 Aspera 传输协议，Aspera 传输解决方案实现了距离无关性，能够在任何位置移动和存储数据，以及随时随地执行高速传输。

除了传输文件，Aspera 的流式技术还可以通过不受管理的互联网，在任何距离上接近零延迟地无错传输任何比特率的视频。

使用安全的用户访问控制、针对传输中的数据和静态存储的数据（可选）的 AES 数据加密以及数据完整性验证来防止中间人攻击，保护数据的安全。

Aspera 安装步骤

调用 conda

```
$ module load miniconda3
```

创建 conda 环境，环境名可自定义，下文以“aspera”为例

```
$ conda create -n aspera
```

激活 conda 环境

```
$ conda activate aspera
```

安装 aspera

```
$ conda install -c hcc aspera-cli
```

Aspera 下载示例

查看命令列表

```
ascp -h
主要使用参数：
-v 详细模式
-Q 用于自适应流量控制，磁盘限制所需
-T 设置为无需加密传输
-l 最大下载速度，一般设为500m
-P TCP 端口，一般为33001
-k 断点续传，通常设为 1
-i 免密下载的密钥文件
```

下载

数据下载请在 **data** 节点上进行下载，激活环境后，可参考如下指令提交，

```
$ ascp -vQT -l 50m -P33001 -k 1 -i ~/.conda/envs/ascp/etc/asperaweb_id_dsa.
↪openssh era-fasp@fasp.sra.ebi.ac.uk:/vol1/srr/SRR182/094/SRR18231894 ./
```

[era-fasp@fasp.sra.ebi.ac.uk:/vol1/srr/SRR182/094/SRR18231894](#) 为示例下载站点及文件，下载路径为当前路径。

注意：如果是用户个人安装的 conda 需要根据 env 路径更改 openssh 文件路径，可通过“which ascp”或者“conda env list”查看 aspera 的 conda 环境路径。

BASIL-ANISE

简介

BASIL 是一种从 **BAM** 格式的对齐配对 **HTS** 读数中检测结构变体（包括插入断点）断点的方法。

BASIL is a method to detect breakpoints for structural variants (including insertion breakpoints) from aligned paired HTS reads in BAM format. ANISE is a method for the assembly of large insertions from paired reads in BAM format and a list candidate insert breakpoints as generated by BASIL.

安装

通过 **conda** 安装

```
module load miniconda3
conda create -n mypy # 创建环境
source activate mypy # 进入环境
conda install -c bioconda anise_basil #安装
```

通过 **git** 安装

```
# download
git clone https://github.com/seqan/anise_basil.git
cd anise_basil
git checkout master
git submodule init
git submodule update --recursive

# compile the program
cd build
cmake ..
make -j 4 anise basil # 通过-j参数设置运行编译时用的核数

# 通过-h参数查看帮助信息
./bin/basil -h
./bin/anise -h
```

使用方法与范例

下载测试数据: https://github.com/seqan/anise_basil/tree/master/example

Step1: 将 reads 比对到参考基因组

```
bwa index ref.fa
bwa aln -f left.fq.gz.sai ref.fa left.fq.gz
bwa aln -f right.fq.gz.sai ref.fa right.fq.gz
bwa sampe ref.fa left.fq.gz.sai right.fq.gz.sai left.fq.gz right.fq.gz |
→samtools view -Sb - | samtools sort - simulated
samtools index simulated.bam
```

Step2: 使用 BASIL 分析 BAM 文件的 tentative insertion sites

```
basil -ir ref.fa -im simulated.bam -ov basil.vcf
```

查看输出 vcf 文件的内容

```
cat basil.vcf
##fileformat=VCFv4.1
##source=BASIL
##reference=ref.fa
##INFO=<ID=IMPRECISE,Number=0,Type=Flag,Description="Imprecise structural
→va...
##INFO=<ID=SVTYPE,Number=1,Type=String,Description="Type of structural
→varia...
##INFO=<ID=OEA_ONLY,Number=0,Type=Flag,Description="Breakpoint support by
→OE...
##ALT=<ID=INS,Description="Insertion of novel sequence">
##FORMAT=<ID=GSCORE,Number=1,Type=String,Description="Sum of Geometric
```

(续下页)

(接上页)

```

→score...
##FORMAT=<ID=CLEFT,Number=1,Type=String,Description="Clipped alignments_
→supp...
##FORMAT=<ID=CRIGHT,Number=1,Type=String,Description="Clipped alignments_
→sup...
##FORMAT=<ID=OELEFT,Number=1,Type=String,Description="One-end anchored_
→alig...
##FORMAT=<ID=OEARIGHT,Number=1,Type=String,Description="One-end anchored_
→ali...
##contig=<ID=1,length=10000>
#CHROM  POS      ID      REF      ALT      QUAL      FILTER  INFO      FORMAT  _
→indi...
1      5001      site_0  T        <INS>      .          PASS      IMPRECISE;
→SVTYPE=INS...

```

Step3: 通过 ANISE 进行断点组装

对 vcf 文件进行过滤（以 30X 数据为例）以生成输入文件.. code:: bash

```
filter_basil.py -i basil.vcf -o basil.filtered.vcf --min-oea-each-side 10
```

运行 ANISE

```
anise -ir ref.fa -im simulated.bam -iv basil.filtered.vcf -of anise.fa
```

观察输出的 fasta 文件，其中包含了 1 个组装好的插入 (insert)，且在注释行 (> 开头的行) 对内容有一定的说明

```

cat anise.fa
>site_0_contig_0 REF=1 POS=5001 STEPS=6 ANCHORED_LEFT=yes ANCHORED_
→RIGHT=yes SPANNING=yes STOPPED=no_more_reads
GGGCTTCGCCTAGGGTCTCGGGAGAAATCTAGGGACCCCAATCTATTAGACGAACACGTCCAGGGCATGG
TCAGGTATACACCTTCCGACTAGACGTGTTCTGAAGATTCGGGAAAATTACCTGAAGAGCCCCCGTAAGCC
GTAGTAGAAGAGGACACTTCATTTAAACAATACCGAAAAAGTGTCTTGGCAGACCGTATCTTCACAGGGC
CGAAGCACTTTTGGCAGGCTTATAAACGCCCAGAATGAAGCACTCGCCATAGGTGGAACCTTTAAGCGA
CGCGGGGTGTGTCGGCCCTATCCCTTGCGCTTACAGACTTTATTTCTTCGTGAGGGAGTTGACCCATGCA

```

获取 ANISE 标记为”spanning”的 Contig

```
extract_spanning anise.fa > anise.filtered.fa
```

Step4: 二次比对

使用 BLAT 将 contig 比对回参考基因组

```
blat ref.fa anise.filtered.fa matches.psl
```

使用 pslPretty 观察结果 matches.psl

```
>site_0_contig_0:0+2592 of 2592 1:4716+5308 of 10000
GGGCTTCGCCTAGGGTCTCGGGAGAAATCTAGGGACCCCAATCTATTAGACGAACACGTC
|||||
GGGCTTCGCCTAGGGTCTCGGGAGAAATCTAGGGACCCCAATCTATTAGACGAACACGTC

CAGGGCATGGTCAGGTATACACCTTCCGACTAGACGTGTTTGAAGATTCGGGAAAATTAC
|||||
CAGGGCATGGTCAGGTATACACCTTCCGACTAGACGTGTTTGAAGATTCGGGAAAATTAC

CTGAAGAGCCCCGTAAGCCGTAGTAGAAGAGGACACTTCATTTAAACAATACCGAAAAA
|||||
CTGAAGAGCCCCGTAAGCCGTAGTAGAAGAGGACACTTCATTTAAACAATACCGAAAAA

GTGTCTTGGCAGACCGTATCTTCACAGGGCCGAAGCACTTTTGGCAGGCTTATAAACGCC
|||||
GTGTCTTGGCAGACCGTATCTTCACAGGGCCGAAGCACTTTTGGCAGGCTTATAAACGCC

CAGAATGAAGCACTCGCCATAGGTGGAAACCTTTAAGCGACGCGGGGTGT...TCAGGTT
|||||
CAGAATGAAGCACTCGCCATAGGTGGAAACCTTTAAGCGACGCG-----2000-----T

TGGGTCCGCGCAGCGCCAACGATTTCAACCGGGAGACGTTTCGTTTCATGATGAGAAGACGG
|||||
TGGGTCCGCGCAGCGCCAACGATTTCAACCGGGAGACGTTTCGTTTCATGATGAGAAGACGG
.....
```

Step5:

通过 BLAT 分数筛选最佳匹配结果

```
best_blat.py -b matches.psl | column -t
#identity query_coverage target_coverage blat_score matches mismatches.
→...
95.9          22.8          5.9          591          592          0          .
→...
```

参考资料

- GitHub homepage: https://github.com/seqan/anise_basil
- Anaconda package: https://anaconda.org/bioconda/anise_basil
- Holtgrewe, M., Kuchenbecker, L., & Reinert, K. (2015). Methods for the Detection and Assembly of Novel Sequence in High-Throughput Sequencing Data. Bioinformatics, btv051.

BatVI

简介

检测病毒整合的软件包

安装

通过 **conda** 安装

```
module load miniconda3
conda create -n mypy #创建环境
source activate mypy #进入环境
conda install -c bioconda batvi #安装BatVI
```

通过链接下载

<https://www.comp.nus.edu.sg/~bioinfo/batvi/batvi1.03.tar.gz>

```
#解压
tar -zxvf batvi1.00.tar.gz
cd batvi1.00
./build.sh
```

使用与范例

CONFIGURATION FILE

在 BatVI 目录下，创建名为 batviconfig.txt 的文件，文件中的内容如下：

```
INDEX=< path_to_pathogen_batmis_index >
PATHOGEN_BLAST_DB=< path_to_pathogen_blast_index >
HG_BLAST_DB=< path_to_human_blast_index >
HG_GENOME=< compressed_human_genome >
HG_BWA=< path_to_human_BWA_index >
PATHOGEN_BWA=< path_to_pathogen_BWA_index >
BLAST_PATH=< path_to_blast_binary >
BWA_PATH=< path_to_BWA_binary >
PICARD_PATH=< path_to_picard_jarfiles >
SAMTOOLS_PATH=< path_to_samtools_binary >
BEDTOOLS_PATH=< path_to_bedtools_binary >
```

batviconfig.txt 内容详解如下（可以通过 gen_path.sh 尝试自动获取路径）：

```
INDEX is the batmis index of the virus database.
HG_BLAST_DB is the blast index of the virus database.
PATHOGEN_BLAST_DB is the blast index of the human genome.
HG_GENOME contains a compressed version of the human genome.
HG_BWA is the BWA index of human.
```

(续下页)

(接上页)

```

PATHOGEN_BWA is the BWA index of the virus database.
BLAST_PATH is the path to the blast binary
BWA_PATH is the path to the BWA binary
PICARD_PATH is the path to the picard jarfiles
SAMTOOLS_PATH is the path to the samtools binary
BEDTOOLS_PATH is the path to the bedtools binary

```

Example of batviconfig.txt

```

#This is an example batviconfig.txt file.
#Comments can be written preceded by a hash..
INDEX=/mnt/projects/HBVall/batmis/HBVall.fa
PATHOGEN_BLAST_DB=/mnt/projects/blast/HBVall/HBVall.fa
HG_BLAST_DB=/mnt/projects/blast/hg19/hg19.fa
HG_GENOME=/mnt/projects/hg19/hg19.fa
HG_BWA=/mnt/projects/bwa/hg19/hg19.fa
PATHOGEN_BWA=/mnt/projects/bwa/HBVall/HBVall.fa
BLAST_PATH=/usr/bin
BWA_PATH=/usr/bin
PICARD_PATH=/usr/bin/picard/
SAMTOOLS_PATH=/usr/bin
BEDTOOLS_PATH=/usr/bin

```

RUNNING THE PROGRAM

准备输入文件

1. pair-ended fastq files

```

A_1.fq
A_2.fq
B_1.fq
B_2.fq

```

2. filelist.txt

```

A_1.fq;A_2.fq;800
B_1.fq;B_2.fq;800

```

病毒整合脚本

```

call_integratons.sh <processing_directory> [ options ]

#processing_directory必须包含filelist.txt, 可以包含batviconfig.txt
#options选项
 #-l|--log - Name of the log files to write processing information to
 #-t|--threads - Number of threads to use
 #-f|--filterdup - Filter out duplicate reads

```

OUTPUT FILES AND FORMAT

final_hits.txt

查看输出文件 `final_hits.txt` 以确认最佳选择，文件内容详解如下：

```
LIB : Library name
Chr : Chromosome of the human integration
Human Pos : Location of the human integration
Sign : Orientation of the human integration
Viral Sign : Orientation of the viral integration
Viral Pos : Location of the viral integration
Read Count : Number of reads used in the prediction of integration. If the
↳entry is marked MSA, the integration has been found using assembly.
Split Reads : Number of split reads involved in the prediction of the
↳integration. A higher number indicates the confidence of a prediction.
Uniquely Mapped Reads : Number of unique mappings to the human genome
↳involved in the prediction. A higher number increases the confidence of a
↳prediction.
Multiply Mapped Reads : Number of multiple mappings used in predicting the
↳integration.
Rank1 Hits : Number of reads which have the tophit near the prediction. A
↳higher number increases the confidence of a prediction. The last two
↳columns of this file are to be ignored for this version of BatVI.
```

predictions.opt.subopt.txt

在文件 `predictions.opt.subopt.txt` 中查看所有候选位点，`predictions.opt.subopt.txt` 详解如下：

```
Sign : Orientation of the human integration
Chr : Chromosome of the human integration
Human St : Location of the human integration
Human Ed : Location of the end of the human integration found by reads
Viral Sign : Orientation of the viral integration
Viral St : Location of the viral integration
Viral Ed : Location of the end of the viral integration found by reads
Median_Rank : Median rank of the prediction
Read Count : Number of reads used in the prediction of integration.
Split Reads : Number of split reads involved in the prediction of the
↳integration.
Uniquely Mapped Reads : Number of unique mappings to the human genome
↳involved in the prediction.
Multiply Mapped Reads : Number of multiple mappings used in predicting the
↳integration.
Rank1 Hits : Number of reads which have the tophit near the prediction. The
↳other columns of this file are to be ignored for this version of BatVI.
```

XXX.predictionsx.msa.txt

在 tmp.batvi/XXX.predictionsx.msa.txt 中查看 (XXX 即 fastq 文件所在目录的名称), 文件详解如下:

```
Chr : Chromosome of the human integration
Human Pos : Location of the human integration
Sign : Orientation of the human integration
Viral Pos : Location of the viral integration
Viral Sign : Orientation of the viral integration
Integration type : If the entry is marked TOPHIT, the assembly maps to this_
→location as the best hit. If the entry is marked REPEAT, the assembly can_
→map to at least one other location with similar confidence, and is_
→therefore ambiguous.
```

范例下载

Example 数据下载可使用网址: biogpu.ddns.comp.nus.edu.sg/~ksung/batvi/test

There is a set of test fastq files in the test directory (biogpu.ddns.comp.nus.edu.sg/~ksung/batvi/test). The `filelist.txt` file is already present. To run a test, a HBV database fasta file is given in `HBVall.fa`. You can download the hg19 fasta file from UCSC genome browser. After you run the program on this data set, the expected output is given in `expected.txt`.

注意事项

- Please ensure that the white spaces in the fasta files and genome names are removed or replaces with a character like an underscore.
- The index builder `bwtformatdb` might crash with a message like "Building cached SA index... xxxx/build_index: 47 line: 168087 Segmentation fault (core dump)". This is OK as the required indexes would have already been built at this stage.
- The installation can be cleaned using the command `'build.sh clean '`.
- A directory can be prepared for a fresh run of BatVI with the command `'clean_run.sh '`.

参考

- BatVI official website: <https://www.comp.nus.edu.sg/~bioinfo/batvi/index.html>
- Tennakoon C, Sung WK. BATVI: Fast, sensitive and accurate detection of virus integrations. *BMC Bioinformatics*. 2017;18(Suppl 3):71. Published 2017 Mar 14. doi:10.1186/s12859-017-1470-x

BCFtools

简介

BCFtools 主要是用来操作 vcf 和 BCF 文件的工具合集，包含有许多命令。用户可使用集群上已经部署的版本，也可自行编译。

CPU 版本 BCFtools 源码安装方法

```

srun -p 64c512g -n 4 --pty /bin/bash
mkdir ${HOME}/01.application/09.bcftools && cd ${HOME}/01.application/09.
→bcftools
wget https://github.com/samtools/bcftools/releases/download/1.15.1/bcftools-
→1.15.1.tar.bz2
tar -jxvf bcftools-1.15.1.tar.bz2
cd bcftools-1.15.1/
./configure --prefix=${HOME}/01.application/09.bcftools/
make
make install
export PATH=${HOME}/01.application/09.bcftools/bin/:$PATH

```

π2 版本 BCFtools

示例脚本如下 (bcftools.slurm):

```

#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load bcftools/1.9-gcc-9.2.0
bcftools query -f '%CHROM %ID %POS %REF %ALT [ %TGT]\n' test.vcf.gz -o test.
→extract.txt
bcftools stats test.vcf > test.vcf.stats

```

使用如下指令提交：

```
$ sbatch bcftools.slurm
```

ARM 版本 BCFtools

示例脚本如下 (bcftools.slurm):

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module use /lustre/share/spack/modules/kunpeng920/linux-centos7-aarch64
module load bcftools/1.10.2-gcc-9.3.0-openblas
bcftools query -f '%CHROM %ID %POS %REF %ALT [ %TGT]\n' test.vcf.gz -o test.
↪extract.txt
bcftools stats test.vcf > test.vcf.stats
```

使用如下指令提交：

```
$ sbatch bcftools.slurm
```

BEDTOOLS2

简介

Bedtools 是一款可以对 **genomic features** 进行比较、相关操作和注释的工具，目前版本已经有三十多个工具／命令用以实现各种不同的功能，可以针对 **bed**、**vcf**、**gff** 等格式的文件进行处理。

安装

使用 **conda** 安装

```
srtn -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n mpy #创建自己的环境
source activate mpy #进入自己的环境
conda install -c bioconda bedtools
```

使用 **git** 安装

```
wget https://github.com/arq5x/bedtools2/releases/download/v2.29.1/bedtools-
↪2.29.1.tar.gz
tar -zxvf bedtools-2.29.1.tar.gz
cd bedtools2
make
```


下载测试数据

```
mkdir -p test_data
cd test_data
curl -O https://s3.amazonaws.com/bedtools-tutorials/web/maurano.dnaseI.tgz
curl -O https://s3.amazonaws.com/bedtools-tutorials/web/cpg.bed
curl -O https://s3.amazonaws.com/bedtools-tutorials/web/exons.bed
curl -O https://s3.amazonaws.com/bedtools-tutorials/web/gwas.bed
curl -O https://s3.amazonaws.com/bedtools-tutorials/web/genome.txt
curl -O https://s3.amazonaws.com/bedtools-tutorials/web/hesc.chromHmm.bed
tar -zxvf maurano.dnaseI.tgz
rm maurano.dnaseI.tgz
```

使用方法与范例

intersect

bedtools intersect 比较两个或多个 **BED/BAM/VCF/GFF** 文件，并识别 **genome** 中两个文件中的特征重叠的所有区域（即共享至少一个碱基对）。

overlap

```
bedtools intersect -a cpg.bed -b exons.bed > result.bed
```

从 **cpg.bed** 中取出与 **exons.bed** 不重叠的区域

```
bedtools intersect -a cpg.bed -b exons.bed -v > result.bed
```

多个文件的比较

```
bedtools intersect -a exons.bed -b cpg.ed gwas.bed hesc.chromHmm.bed -
↪sorted > result.bed
```

从 **bam** 与 **bed** 比较

```
bedtools intersect -abam tmp.bam -b exons.bed > result.bed
```

指定 **overlap** 的最小 **fraction**

```
bedtools intersect -a cpg.bed -b exons.bed -wo -f 0.50
```

merge

Bedtools merge 命令可以将重叠的区间或者紧邻的区间合并成一个新的区间。

合并重叠区间形成一个新的区间

```
bedtools merge -i cpg.bed > result_merge.bed
```

注意事项

- bedtools 默认输入文件的分隔符为 TAB，除了 bam 格式的文件；
- 如果未使用-sorted 参数，则 bedtools 默认不支持大于 512M 的染色体；
- -sorted 参数和-g 参数必须存在一个；
- 当进行多个文件比较时，染色体的命名方式必须统一，'chrX' 和 'X' 不可以同时存在

参考

- bedtools: a powerful toolset for genome arithmetic: <https://bedtools.readthedocs.io/en/latest/index.html>

BICseq2

简介

BICseq2 is an algorithm developed for the normalization of high-throughput sequencing (HTS) data and detect copy number variations (CNV) in the genome. BICseq2 can be used for detecting CNVs with or without a control genome.

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda bicseq2-norm
```

Bismark

简介

Bismark 可以高效地分析 BS-Seq 数据，方便地进行读段比对和甲基化探测，Bismark 能区分 CpG、CHG 和 CHH，允许用户通过可视化来解释数据。

可用的版本

版本	平台	构建方式	模块名
0.23.0		Spack	<i>bismark/0.23.0-gcc-11.2.0</i> 思源一号
0.19.0		Spack	<i>bismark/0.19.0-intel-19.0.4</i>

如何使用

思源一号集群 **Bismark**

在思源一号集群上使用如下命令:

```
srun -p 64c512g -n 4 --pty /bin/bash
module load bismark/0.23.0-gcc-11.2.0
bismark --help
```

π 集群 **Bismark**

在 π 集群上使用如下命令:

```
srun -p small -n 4 --pty /bin/bash
module load bismark/0.19.0-intel-19.0.4
bismark --help
```

使用 **Conda** 安装及运行

使用 **Conda** 安装 **Bismark**

推荐使用 Conda 在用户目录部署特定的 Bismark 软件, 以思源一号为例:

```
srun -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda create -n biotools # 创建新的环境
source activate biotools # 激活环境
conda install -c bioconda bismark=0.23.1 bowtie2=2.2.1 # 安装 bismark
bismark --help
```

基因组

```
wget https://hgdownload.cse.ucsc.edu/goldenpath/hg19/bigZips/hg19.fa.gz
gunzip -c hg19.fa.gz > ~/hg19/hg19.fa # 需要创建 ~/hg19 目录
```

运行示例

建立基因组索引

```
bismark_genome_preparation ~/hg19
```

序列比对

```
cp $(which test_data.fastq) .
bismark --genome ~/hg19 test_data.fastq
# 输出
└─ test_data_bismark_bt2.bam
└─ test_data_bismark_bt2_SE_report.txt
```

查看 BAM 文件 `samtools view test_data_bismark_bt2.bam | head -n5 :`

```
SRR020138.15024317_SALK_2029:7:100:1672:902_length=86      16      chr1
→ 57798677      42      50M      *      0      0
→ TTCTTTCCCATCCCATAAATCCTAAAAATAATAAAAAATCATCCCCAAAT      @@:AC@<=+@?
→ +8) @BCCCA=6BCCCCCCCCCCCCCCCCACB=<88BCCA      NM:i:11
→ MD:Z:14G2G6G0G0G0G4G1G1G0G10G1      XM:Z:.....z..h.....hhhh....h.h.
→ hh.....h. XR:Z:CT XG:Z:GA
SRR020138.15024318_SALK_2029:7:100:1672:137_length=86      0      chr12
→ 129774096      8      50M      *      0      0
→ AAAAAAAAAAAAAAGAAAAAAAAAGAAAAAGAAAAGTAAAAAAAAA      =@CAA=@B@CB=98
→ %:AB?>@56/=3<=<)>B@:*=:=61%,<A@1+12      NM:i:2 MD:Z:41C5G2      XM:Z:...
→ .....h..... XR:Z:CT XG:Z:CT
SRR020138.15024319_SALK_2029:7:100:1672:31_length=86 0      chr2
→ 10166575      42      50M      *      0      0
→ ATTTTGTTATAGAGTGGGGTATTTTCGGAAGAAGGAGGAGGAGTGTATT      BCCCCBCCCCA?
→ :.=ACBCBACCCCCBCCA??5=9@4BB@;??B@BABBA      NM:i:8
→ MD:Z:1C1C5C9C2C0C22C1C1      XM:Z:.h.x.....x.....h..hh.Z.....
→ .h.x. XR:Z:CT XG:Z:CT
SRR020138.15024320_SALK_2029:7:100:1672:1164_length=86      16      chr5
→ 28344472      8      50M      *      0      0
→ CACAAAATATCAACACCCCTAAACCCACATTATTCAAAAATCAATTATA      @@@BBBA@A9=A@<?
→ ::2:<CB@?=:BBAC??CB@@BBBBC>:ACABCAB      NM:i:11
→ MD:Z:4G1G1G3G9G9G5G0G0G1T4G2      XM:Z:....x.h.h...x.....h.....h....
→ .hhh.....h.. XR:Z:CT XG:Z:GA
SRR020138.15024321_SALK_2029:7:100:1672:433_length=86      0      chr14
→ 38711099      42      50M      *      0      0
→ TTTTGAGTAGAGAAGTTAGTATTTTAGGGAATTTTGATTTTTTTAAGTT      BCCBB?B@@A>@-
→ 4BBB:7@BBBCBBC@@@=A@BCACA;BCBBCBB@@@BB      NM:i:14
→ MD:Z:0C0C13C0C6C0C6C0C1C3C0C1C0C1C5      XM:Z:hh.....hx.....hx....
→ ..hh.x...hh.hh.h..... XR:Z:CT XG:Z:CT
```

甲基化 `call` 字符串对于 **BS-read** 中不涉及胞嘧啶的每个位置都用一个点 `.` 代替，或包含以下不同的胞嘧啶甲基化的字母 (大写 = 甲基化，小写 = 未甲基化)：

```

X # 代表CHG中甲基化的C
x # 代表CHG中非甲基化的C
H # 代表CHH中甲基化的C
h # 代表CHH中非甲基化的C
Z # 代表CpG中甲基化的C
z # 代表CpG中非甲基化的C
U # 代表其他情况的甲基化C (CN或者CHN)
u # 代表其他情况的非甲基化C (CN或者CHN)
. # 该位置不是胞嘧啶

```

去除重复

```

deduplicate_bismark --bam test_data_bismark_bt2.bam
# 输出
└─ test_data_bismark_bt2.deduplicated.bam
└─ test_data_bismark_bt2.deduplication_report.txt

```

提取甲基化水平

默认情况下，软件会自动根据甲基化的 *C* 的类型 (*CpG*, *CHG*, *CHH*) 和 比对到四条链上 (*OT*, *OB*, *CTOT*, *CTOB*) 两个因素生成结果文件。

- OT -- original top strand
- CTOT -- complementary to original top strand
- OB -- original bottom strand
- CTOB -- complementary to original bottom strand

```

# extract context-dependent (CpG/CHG/CHH) methylation
cpanm GD::Graph::lines # 安装画图的依赖模块，非必须
bismark_methylation_extractor --gzip --bedGraph test_data_bismark_bt2.
→deduplicated.bam
# 输出
└─ CHG_OB_test_data_bismark_bt2.deduplicated.txt.gz
└─ CHG_OT_test_data_bismark_bt2.deduplicated.txt.gz
└─ CHH_OB_test_data_bismark_bt2.deduplicated.txt.gz
└─ CHH_OT_test_data_bismark_bt2.deduplicated.txt.gz
└─ CpG_OB_test_data_bismark_bt2.deduplicated.txt.gz
└─ CpG_OT_test_data_bismark_bt2.deduplicated.txt.gz
└─ test_data_bismark_bt2.deduplicated.bedGraph.gz
└─ test_data_bismark_bt2.deduplicated.bismark.cov.gz
└─ test_data_bismark_bt2.deduplicated.M-bias_R1.png
└─ test_data_bismark_bt2.deduplicated.M-bias.txt
└─ test_data_bismark_bt2.deduplicated_splitting_report.txt

```

查看结果 `zcat CHG_OB_test_data_bismark_bt2.deduplicated.txt.gz | head -n5 :`

```
# Bottom链在CHG背景下的甲基化信息
SRR020138.15024320_SALK_2029:7:100:1672:1164_length=86      -      chr5  𐄂
  ↳ 28344484          x
SRR020138.15024320_SALK_2029:7:100:1672:1164_length=86      -      chr5  𐄂
  ↳ 28344476          x
SRR020138.15024326_SALK_2029:7:100:1672:1418_length=86      -      chr5  𐄂
  ↳ 126218386         x
SRR020138.15024326_SALK_2029:7:100:1672:1418_length=86      -      chr5  𐄂
  ↳ 126218354         x
```

网页报告

```
bismark2report
# 输出
└─ test_data_bismark_bt2_SE_report.html
```

参考资料

- [Bismark 文档](#)

Blast-plus

简介

全称 **Basic Local Alignment Search Tool**，即”基于局部比对算法的搜索工具”。**Blast** 的运行方式是先用目标序列建数据库（这种数据库称为 **database**，里面的每一条序列称为 **subject**），然后用待查的序列（称为 **query**）在 **database** 中搜索，每一条 **query** 与 **database** 中的每一条 **subject** 都要进行双序列比对，从而得出全部比对结果。

blastp：蛋白序列与蛋白库做比对，直接比对蛋白序列的同源性。

blastx：核酸序列对蛋白库的比对，先将核酸序列翻译成蛋白序列（根据相位可以翻译为 6 种可能的蛋白序列），然后再与蛋白库做比对。

blastn：核酸序列对核酸库的比对，直接比较核酸序列的同源性。

tblastn：蛋白序列对核酸库的比对，将库中的核酸翻译成蛋白序列，然后进行比对。

tblastx：核酸序列对核酸库在蛋白级别的比对，将库和待查序列都翻译成蛋白序列，然后对蛋白序列进行比对。

可用的版本

版本	平台	构建方式	模块名
2.9.0	arm	spack	blast-plus/2.9.0-gcc-9.3.0 ARM
2.13.0	cpu	precompile	blast-plus/2.13.0-gcc-11.2.0 思源一号
2.13.0	cpu	precompile	blast-plus/2.13.0-gcc-11.2.0

ARM 版本 BLAST+

示例脚本如下 (blast.slurm):

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load blast-plus/2.9.0-gcc-9.3.0
makeblastdb -in ref.fa -dbtype nucl
blastn -query in.fa -db ref.fa -out blast_result.txt
```

使用如下指令提交:

```
$ sbatch blast.slurm
```

CPU 版本 BLAST+

BLAST+ 预编译文件安装步骤

官网下载预编译文件

```
$ wget http://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.13.0/ncbi-
→blast-2.13.0+-x64-linux.tar.gz
```

解压

```
$ tar -zxvf ncbi-blast-2.13.0+-x64-linux.tar.gz
```

添加 BLAST+ 的环境变量

```
$ export PATH=path/to/blast/bin:$PATH
```

检验安装, 以下命令查看 BLAST+ 版本信息

```
$ blastn -version
```

BLAST+ 运行示例

官网下载基因组并解压

```
$ wget ftp://ftp.ensemblgenomes.org/pub/plants/release-36/fasta/arabidopsis_
→thaliana/dna/Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.gz
$ gzip -d Arabidopsis_thaliana.TAIR10.dna.toplevel.fa.gz
```

调用 BLAST+

```
$ module load blast-plus/2.13.0-gcc-11.2.0
```

构建核酸 BLAST 数据库

```
$ makeblastdb -in Arabidopsis_thaliana.TAIR10.dna.toplevel.fa -dbtype nucl -  
→out TAIR10 -parse_seqids
```

下载拟南芥 protein 数据

```
$ wget ftp://ftp.ensemblgenomes.org/pub/plants/release-36/fasta/arabidopsis_  
→thaliana/pep/Arabidopsis_thaliana.TAIR10.pep.all.fa.gz
```

构建蛋白 BLAST 数据库

```
$ gzip -dArabidopsis_thaliana.TAIR10.pep.all.fa.gz  
$ makeblastdb -in Arabidopsis_thaliana.TAIR10.pep.all.fa -dbtype prot -out  
→TAIR10 -parse_seqids
```

生成随机序列 query.fa

```
$ echo  
→TGAAAGCAAGAAGAGCGTTTGGTGGTTTCTTAACAAATCATTGCAACTCCACAAGGCGCCTGTAATAGACAGCTTGTGCATGGAAC  
→>> query.fa
```

使用构建好的数据库进行检索

```
$ blastn -db BLAST/TAIR10 -query query.fa
```

参考资料

- [BLAST](#)
- [NCBI](#)

BOWTIE

简介

Bowtie 是一种快速短对齐器，使用基于 Burrows-Wheeler 变换和 FM-index 的算法。索引可以容忍少量的不匹配。另见 [seqanswers/Bowtie](#)。

BOWTIE2

简介

Bowtie2 是将测序 reads 与长参考序列比对工具。适用于将长度大约为 50 到 100 或 1000 字符的 reads 与相对较长的基因组（如哺乳动物）进行比对。Bowtie2 使用 FM 索引（基于 Burrows-Wheeler Transform 或 BWT）对基因组进行索引，以此来保持其占用

较小内存。对于人类基因组来说，内存占用在 3.2G 左右。Bowtie2 支持间隔，局部和双端对齐模式。可以同时使用多个处理器来极大的提升比对速度。

π 集群上的 Bowtie2

查看 π 集群上已编译的软件模块:

```
$ module avail bowtie2
```

调用该模块:

```
$ module load bowtie2/2.3.5.1-intel-19.0.4
```

π 集群上的 Slurm 脚本 slurm.test

cpu 队列每个节点配有 40 核，这里使用了 1 个节点:

```
#!/bin/bash

#SBATCH -J bowtie2_test
#SBATCH -p cpu
#SBATCH -n 40
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module load bowtie2

bowtie2-build hsa.fa hsa
bowtie2 -p 6 -3 5 --local -x hsa -1 example_1.fastq -2 example_2.fastq -S \
→test.sam
```

π 集群上提交作业

```
$ sbatch slurm.test
```

参考资料

- Bowtie2 : <http://bowtie-bio.sourceforge.net/bowtie2/index.shtml>

BreakDancer

简介

BreakDancer 包含两个互补的程序: BreakDancerMax 和 BreakDancerMini。BreakDancerMax 根据二代测序 read 比对时,出现的异常比对,预测插入,缺失,倒位,染色体间或染色体内部易位等五种结构突变。

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda breakdancer
```

BWA

简介

BWA 是用于将 DNA 与大型参考基因组（例如人类基因组）进行比对的开源软件。

可用的版本

版本	平台	构建方式	模块名
0.7.17		spack	bwa/0.7.17-intel-2021.4.0 思源一号
0.7.17		spack	bwa/0.7.17-gcc-11.2.0 Pi2.0
0.7.17		spack	bwa/0.7.17-gcc-8.5.0 Pi2.0

算例获取方式

思源：

```
mkdir ~/bwa && cd ~/bwa
cp -r /dssg/share/sample/bwa/* ./
gzip -d B17NC_R1.fastq.gz
gzip -d B17NC_R2.fastq.gz
```

 π 2.0：

```
mkdir ~/bwa && cd ~/bwa
cp -r /lustre/share/sample/bwa/* ./
gzip -d B17NC_R1.fastq.gz
gzip -d B17NC_R2.fastq.gz
```

集群上的 BWA

- 思源一号 BWA
- π 2.0 BWA

思源一号上的 BWA

首先，创建索引文件

```
#!/bin/bash

#SBATCH --job-name=bwa
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load bwa
bwa index -a bwtsw hg19.fa
```

然后在运行相关数据

```
#!/bin/bash

#SBATCH --job-name=bwa
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load bwa
bwa mem -t 64 hg19.fa B17NC_R1.fastq B17NC_R2.fastq
```

π2.0 上的 BWA

创建索引文件

```
#!/bin/bash

#SBATCH --job-name=bwa
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load bwa
bwa index -a bwtsw hg19.fa
```

运行相关数据

```
#!/bin/bash

#SBATCH --job-name=bwa
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load bwa
bwa mem -t 40 hg19.fa B17NC_R1.fastq B17NC_R2.fastq
```

运行结果

思源一号

索引文件创建结果：

```
[bwt_gen] Finished constructing BWT in 695 iterations.
[bwa_index] 1935.82 seconds elapse.
[bwa_index] Update BWT... 10.76 sec
[bwa_index] Pack forward-only FASTA... 8.42 sec
[bwa_index] Construct SA from BWT and Occ... 743.24 sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa index -a bwtsv hg19.fa
[main] Real time: 2750.503 sec; CPU: 2713.697 sec
```

64核心运行结果

```
[M::mem_process_seqs] Processed 4720686 reads in 456.298 CPU sec, 8.910_
↪real sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa mem -t 64 hg19.fa B17NC_R1.fastq B17NC_R2.fastq
[main] Real time: 184.120 sec; CPU: 7961.600 sec
```

π2.0

索引文件创建结果：

```
[bwt_gen] Finished constructing BWT in 695 iterations.
[bwa_index] 1989.35 seconds elapse.
[bwa_index] Update BWT... 13.47 sec
[bwa_index] Pack forward-only FASTA... 13.20 sec
[bwa_index] Construct SA from BWT and Occ... 739.38 sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa index -a bwtsv hg19.fa
[main] Real time: 2784.274 sec; CPU: 2775.397 sec
```

(续下页)

(接上页)

64核心运行结果

```
[M::mem_process_seqs] Processed 1520686 reads in 169.987 CPU sec, 4.657
↪real sec
[main] Version: 0.7.17-r1188
[main] CMD: bwa mem -t 40 hg19.fa B17NC_R1.fastq B17NC_R2.fastq
[main] Real time: 320.462 sec; CPU: 9784.877 sec
```

参考链接: <https://github.com/lh3/bwa>

CCMpred**简介**

CCMpred 是一种用于预测蛋白质残基间接触的软件, 其基于相关突变分析方法, 可以高效地利用多序列比对信息, 预测出蛋白质序列中不同残基之间的接触概率。

安装方法

CCMpred 需要使用 **CUDA**, 因此安装需要在具有 **gpu** 资源的节点上进行安装。

首先申请一个带有 **gpu** 资源的节点

```
srunk -N 1 -n 1 -p dgx2 --gres=gpu:1 --pty /bin/bash
```

进入节点后执行安装步骤

```
cd /lustre/home/acct-clsygy/clsygy/software
git clone --recursive https://github.com/soedinglab/CCMpred.git
cd CCMpred
cmake .
make
```

安装完成后, 可执行程序在 **/bin** 中, 执行方式如下 (以查看帮助文档为例):

```
./bin/ccmpred -h
```

cd-hit**简介**

cd-hit 是用于蛋白质序列或核酸序列聚类的工具, 根据序列的相似度对序列进行聚类以去除冗余的序列, 一般用于构建非冗余的数据集。

CPU 版本 cd-hit 安装方法

```
srunk -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n cdhit
```

(续下页)

(接上页)

```
source activate cdhit
conda install -c bioconda cd-hit
```

CDO

简介

CDO 软件是一个包含大量标准处理气候和预报模式数据的算子的集合。该算子包括简单的统计和算术方程，资料选取和二次抽样，和空间插值。

celldex

简介

celldex 包提供一组参考表达数据集，这些数据集带有精确的细胞类型标签，用于单细胞数据的自动注释或批量 **RNA** 序列的反卷积等过程。

安装方式

可以使用 **Conda** 进行安装，以思源一号为例：

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n celldex
source activate celldex
conda install bioconductor-celldex
```

参考资料

- [LTLA/celldex](#)

CellPhoneDB

简介

CellPhoneDB 是公开的人工校正的，储存受体、配体以及两种相互作用的数据库。

安装方式

可以使用 **Conda** 进行安装，以思源一号为例：

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
```

(续下页)

(接上页)

```
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n cpdb
source activate cpdb
conda install python=3.7 r=4.1
pip install cellphonedb
```

参考资料

- [CellPhoneDB](#)

cdsapi

简介

The Climate Data Store (CDS) API is the service to allow users to request data from CDS datasets via a python script. These scripts use a number of keywords which vary from dataset to dataset, usually following the sections of the CDS download form. As the CDS API cannot currently return the valid keyword list on demand, they are documented on this page for some of the most popular CDS datasets.

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c conda-forge cdsapi
```

CLEVER

简介

The clever toolkit (CTK) is a suite of tools to analyze next-generation sequencing data and, in particular, to discover and genotype insertions and deletions from paired-end reads.

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda clever-toolkit
```

CNVkit

简介

CNVkit 是一个 Python 库和命令行软件工具包，用于研究 CNV (Copy number variation) 拷贝数变异的软件。

安装方式

可以使用 Conda 进行安装，以思源一号为例：

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n cnvkit
source activate cnvkit
conda install cnvkit
```

参考资料

- [CNVkit 文档](#)

CNVnator

简介

一种用于发现和表征群体基因组测序数据中拷贝数变异 (CNV) 的工具。该工具也非常适合个人基因组分析。该方法基于均值偏移、多带宽分区和 GC 校正。

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda -c conda-forge cnvnator
```

Control-FERRC

简介

Control-FREEC is a tool for detection of copy-number changes and allelic imbalances (including LOH) using deep-sequencing data originally developed by the Bioinformatics Laboratory of Institut Curie (Paris).

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda control-freec
```

ColabFold

ColabFold 是 Sergey Ovchinnikov 等人开发的快速蛋白结构预测软件，使用 MMseqs2 替代 MSA，能够快速精准预测包含复合体在内的蛋白结构。开源代码：<https://github.com/sokrypton/ColabFold>

ColabFold 支持本地安装使用，Yoshitaka Moriwaki 开发维护的 LocalColabFold 可以很容易在交大思源一号上安装。下面将介绍以 LocalColabFold 形式在思源一号上安装和使用 ColabFold

交大计算平台同时也部署了 AlphaFold 和 ParaFold，欢迎查看：[AlphaFold2](#)

ColabFold 安装

申请 CPU 计算节点，以交互模式安装。conda 安装都需要至计算节点：

```
srun -p 64c512g -n 8 --pty /bin/bash
```

假设在个人 home 文件夹下建立 colab 文件夹：

```
mkdir ~/colab; cd colab
```

下载 localcolabfold：

```
git clone https://github.com/YoshitakaMo/localcolabfold.git
```

接下来一键安装全部软件 (这里预计半小时以上)：

```
cd localcolabfold
./install_colabbatch_linux.sh
```

安装完毕，将出现 Installation of colabfold_batch finished 字样

ColabFold 使用

ColabFold 在思源一号上有两种运行方法：

- 交互模式，适用于短序列和调试
- slurm 作业模式，适用于长时间或正式计算

方法一：交互模式运行

申请 GPU 计算节点：

```
srun -p a100 -N 1 -n 1 --cpus-per-task=16 --gres=gpu:1 --pty /bin/bash
```

激活 conda 环境：

```
export PATH="/colab/localcolabfold/localcolabfold/colabfold-conda/bin:$PATH"
↪
module load miniconda3
source activate ~/colab/localcolabfold/localcolabfold/colabfold-conda
```

在包含 test.fasta 的文件夹里运行：

```
colabfold_batch --num-recycle 1 test.fasta output
```

其中，test.fasta 文件内容示例：

```
>2LHC_1|Chain A|Ga98|artificial gene (32630)
PIAQIHILEGRSDEQKETLIREVSEAI SRSLDAPLTSVRVIITEMAKGHFGIGGELASK
```

方法二：slurm 脚本运行

作业脚本示例（假设作业脚本名为 sub.slurm）：

```
#!/bin/bash
#SBATCH --job-name=colabfold
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export PATH="/colab/localcolabfold/colabfold_batch/bin:$PATH"
export https_proxy=http://proxy2.pi.sjtu.edu.cn:3128
export http_proxy=http://proxy2.pi.sjtu.edu.cn:3128
export no_proxy=puppet,proxy,172.16.0.133,pi.sjtu.edu.cn

module load miniconda3
source activate ~/colab/localcolabfold/colabfold_batch/colabfold-conda

colabfold_batch --num-recycle 1 test.fasta output
```

然后使用 sbatch sub.slurm 语句提交作业

参考资料

- AlphaFold GitHub: <https://github.com/deepmind/alphafold>
- ColabFold GitHub: <https://github.com/sokrypton/ColabFold>
- LocalColabFold GitHub: <https://github.com/YoshitakaMo/localcolabfold>
- ParaFold 网站: <https://parafold.sjtu.edu.cn>

CREST

简介

CREST (Clipping Reveals Structure) is a new algorithm for detecting genomic structural variations at base-pair resolution using next-generation sequencing data

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda blat
conda install -c bioconda cap3
conda install -c bioconda samtools
conda install -c bioconda perl-bioperl
conda install -c bioconda perl-bio-db-sam
conda install -c imperial-college-research-computing crest
```

CUFFLINKS

简介

Cufflinks 下主要包含 cufflinks, cuffmerge, cuffcompare 和 cuffdiff 等几支主要的程序。主要用于基因表达量的计算和差异表达基因的寻找。Cufflinks 程序主要根据 Tophat 的比对结果，依托或不依托于参考基因组的 GTF 注释文件，计算出 (各个 gene 的) isoform 的 FPKM 值，并给出 transcripts.gtf 注释结果 (组装出转录组)。

chimera

简介

This package facilitates the characterisation of fusion products events. It allows to import fusiondata results from the following fusion finders: chimeraScan, bellerophonotes, deFuse, Fusion-Finder, FusionHunter, mapSplice, tophat-fusion, FusionMap, STAR, Rsubread, fusionCatcher.

完整步骤

```

srun -p small -n 10 --pty /bin/bash
conda create -n chimera
source activate chimera
conda install -c bioconda/label/gcc7 bioconductor-chimera
source activate chimera

```

Cell Ranger

简介

Cell Ranger 是一组分析流程，用于从单细胞数据中执行样本解多重重复、条形码处理、单细胞 3' 和 5' 基因计数、V(D)J 转录序列组装和注释，以及特征条形码分析。

可用版本

版本	平台	构建方式	模块名
7.2.0		precompile	cellranger/7.2.0 Pi2.0

安装命令

前往 **Cell Ranger** 下载页面填写 10x Genomics 用户软件许可协议信息，获得下载链接

```

tar -zxvf cellranger-7.2.0.tar.gz
export PATH=/PATH/TO/cellranger-7.2.0:$PATH

```

任务脚本

```

#!/bin/bash
#SBATCH --job-name=cellrange
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load cellranger/7.2.0
cellranger testrun --id=check_install

```

参考资料

- [Cell Ranger](#)
- [Cell Ranger Tutorials](#)

DeepGo

简介

使用深度学习识别分类器根据序列和相互作用预测蛋白质功能。

完整步骤

```
srun -p small -n 4 --pty /bin/bash
git clone https://github.com/bio-ontology-research-group/deepgo.git
module load miniconda3
conda create -n mpy
source activate mpy
conda install pip
pip install -r requirements.txt
```

DeepVariant

DeepVariant is an analysis pipeline that uses a deep neural network to call genetic variants from next-generation DNA sequencing data. DeepVariant relies on Nucleus, a library of Python and C++ code for reading and writing data in common genomics file formats (like SAM and VCF) designed for painless integration with the TensorFlow machine learning framework.

CPU 版本的 Singularity DeepVariant

CPU 版安装

申请计算节点，然后制作 singularity 镜像

```
$ srun -p cpu -N 1 --exclusive --pty /bin/bash
$ singularity build deepvariant.simg docker://google/deepvariant
```

用 **SLURM** 脚本提交 **CPU 版 DeepVariant** 作业

使用 CPU 版本的 singularity 镜像的 slurm.sh 如下：

```
#!/bin/bash

#SBATCH -J DeepVariant
#SBATCH -p small
#SBATCH -n 1
#SBATCH --ntasks-per-node=1
```

(续下页)

(接上页)

```
#SBATCH -o %j.out
#SBATCH -e %j.err

ulimit -s unlimited
ulimit -l unlimited

IMAGE_PATH=/安装路径/deepvariant.simg

singularity run $IMAGE_PATH /opt/deepvariant/bin/make_examples
```

并使用如下指令提交：

```
$ sbatch slurm.sh
```

交互式提交 CPU 版 DeepVariant 作业

```
srun -p cpu -N 1 --exclusive --pty /bin/bash
export IMAGE_PATH=/安装路径/deepvariant.simg
singularity run $IMAGE_PATH /opt/deepvariant/bin/make_examples
```

GPU 版本的 Singularity DeepVariant

GPU 版安装

申请计算节点，然后制作 singularity 镜像

```
$ srun -p cpu -N 1 --exclusive --pty /bin/bash
$ singularity build deepvariant.gpu.simg docker://google/deepvariant:0.10.0-
→gpu
```

用 SLURM 脚本提交 GPU 版作业

使用 GPU 版本的 singularity 镜像的 slurm.sh 如下：

```
#!/bin/bash

#SBATCH -J DeepVariant
#SBATCH -p dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1
#SBATCH --mem=MaxMemPerNode
#SBATCH -o %j.out
#SBATCH -e %j.err

ulimit -s unlimited
```

(续下页)

(接上页)

```
ulimit -l unlimited

IMAGE_PATH=/安装路径/deepvariant.gpu.simg

singularity run $IMAGE_PATH /opt/deepvariant/bin/make_examples
```

并使用如下指令提交：

```
$ sbatch slurm.sh
```

交互式提交 GPU 版 deepvariant 作业

```
srunk --ntasks-per-node=1 -p dgx2 --gres=gpu:1 -N 1 --pty /bin/bash
export IMAGE_PATH=/安装路径/deepvariant.gpu.simg
singularity run $IMAGE_PATH /opt/deepvariant/bin/make_examples
```

参考资料

- DeepVariant 官网 <https://github.com/google/deepvariant>

DELLY

简介

Delly 是一种集成的结构变异 (SV) 预测方法，可以在短期读取大规模并行测序数据，以单核苷酸分辨率发现基因分型和可视化缺失、串联重复、倒位和易位等缺陷。它使用配对末端，拆分阅读和阅读深度来敏感而准确地描绘整个基因组的重排。关于 Delly 的更多信息请访问：<https://tobiasrausch.com/delly/>。

目前 ARM 超算已经部署了 0.8.3 版本的 delly； π 集群上未全局部署，用户可通过 conda 自行安装。

ARM 版本 delly

在 ARM 节点查看已编译的软件模块：

```
module avail delly
```

在 ARM 节点调用已编译的软件模块：

```
module load delly/0.8.3
```

示例脚本如下 (delly.slurm)：

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
```

(续下页)

(接上页)

```
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load delly/0.8.3
delly --version
delly --help
```

在 **ARM** 节点上使用如下指令提交（若在 **π2.0** 登录节点上提交将出错）：

```
sbatch delly.slurm
```

π 集群上 conda 安装的完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda delly
```

DESeq2

简介

分析来自 **RNA-seq** 的计数数据的一项基本任务是检测差异表达的基因。计数数据以表格的形式呈现，其中报告了每个样本已分配给每个基因的序列片段的数量。其他检测类型也有类似的数据，包括比较 **ChIP-Seq**、**HiC**、**shRNA** 筛选和质谱分析。一个重要的分析问题是与条件内的变异性相比，条件之间的系统变化的量化和统计推断。**DESeq2** 包提供了使用负二项式广义线性模型测试差异表达的方法；离散度和对数倍数变化的估计包含数据驱动的先验分布。此小插图解释了包的使用并演示了典型的工作流程。**Bioconductor** 网站上的 **RNA-seq** 工作流程涵盖了与此小插图类似的材料，但速度较慢，包括从 **FASTQ** 文件生成计数矩阵。

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda bioconductor-deseq2
```

安装完成后可以在 **R** 中输入 `library("DESeq2")` 检测是否安装成功

DoubletFinder

简介

DoubletFinder 可用来检测单细胞测序中的 **Doublet**，也就是指双联体细胞。

安装方式

可以使用 **Conda** 进行安装，以思源一号为例：

```
srunc -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n doubletfinder
source activate doubletfinder
conda install -c paul.martin-2 r-doubletfinder
```

参考资料

- [DoubletFinder](#)

ERDS

简介

实体关系图。

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda erds
```

FastQC

简介

FastQC 是一个二代测序数据质量控制软件。

可用的版本

版本	平台	构建方式	模块名
0.11.9		Spack	<i>fastqc/0.11.9-gcc-11.2.0-openjdk</i> 思源一号
0.11.7		Spack	<i>fastqc/0.11.7-gcc-9.2.0</i>
0.11.7		Spack	<i>fastqc/0.11.7-intel-19.0.4</i>
0.11.7		Spack	<i>fastqc/0.11.7-gcc-8.3.0</i>

使用 **Conda** 安装 **FastQC**

推荐使用 Conda 在用户目录部署特定的 FastQC 软件，以思源一号为例：

```

srun -p 64c512g -n 4 --pty /bin/bash
module purge
module load miniconda3/4.10.3
conda create -n biotools                                # 创建新的环境
source activate biotools                                # 激活环境
conda install -c bioconda fastqc                         # 安装软件
fastqc --help

```

示例文件

```

# 思源一号
/dssg/share/sample/bwa/B17NC_R1.fastq.gz
/dssg/share/sample/bwa/B17NC_R2.fastq.gz
# 集群
/lustre/share/samples/bwa/B17NC_R1.fastq.gz
/lustre/share/samples/bwa/B17NC_R2.fastq.gz

```

运行示例

思源一号集群 **FastQC**

作业脚本 `test.slurm` 内容如下：

```

#!/bin/bash
#SBATCH --job-name=QC
#SBATCH --partition=64c512g
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=10

```

(续下页)

(接上页)

```

ulimit -l unlimited
ulimit -s unlimited

module load fastqc/0.11.9-gcc-11.2.0-openjdk
input_dir=/dssg/share/sample/bwa
fastqc -f fastq -o ~/QC $input_dir/B17NC_R1.fastq.gz $input_dir/B17NC_R2.
↪fastq.gz

```

使用 sbatch 提交作业

```
sbatch test.slurm
```

π 集群 FastQC

作业脚本 test.slurm 内容如下:

```

#!/bin/bash
#SBATCH --job-name=QC
#SBATCH --partition=small
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=10

ulimit -l unlimited
ulimit -s unlimited

module load fastqc/0.11.7-gcc-9.2.0
input_dir=/lustre/share/samples/bwa
fastqc -f fastq -o ~/QC $input_dir/B17NC_R1.fastq.gz $input_dir/B17NC_R2.
↪fastq.gz

```

运行结果

会输出质控网页报告，可下载后查看。

```

QC
├─ B17NC_R1_fastqc.html
├─ B17NC_R1_fastqc.zip
├─ B17NC_R2_fastqc.html
└─ B17NC_R2_fastqc.zip

```

FASTQ 格式说明

FASTQ 文件是一个文本文件，其中包含通过流动槽 flow cell 上质控参数的簇 cluster 的测序数据。对于每个通过质控参数的簇，一个序列被写入相应样本的 R1 FASTQ 文件，而对于双端测序运行，另外一个序列也被写入该样本的 R2 FASTQ 文件。FASTQ 文件中的每个条目包含 4 行：

1. 序列标识符，其中包含有关测序运行和簇的信息；
2. 序列（碱基信号；A, C, T, G 和 N）；
3. 分隔符，只是一个加号 (+)；
4. 读取碱基的质量值。这些是 Phred +33 编码的，使用 ASCII 字符表示数字质量值。

FASTQ 文件中单个记录条目的示例：

```
@SIM:1:FCX:1:15:6329:1045 1:N:0:2
TCGCACTCAACGCCCTGCATATGACAAGACAGAATC
+
<>;##=><9=AAAAAAAAAA9#:<#<;<<?????#=#
```

参考资料

- [FASTQ 格式说明](#)

FermiKit

简介

FermiKit is a variant calling pipeline for Illumina whole-genome germline data. It de novo assembles short reads and then maps the assembly against a reference genome to call SNPs, short insertions/deletions and structural variations. FermiKit takes about one day to assemble 30-fold human whole-genome data on a modern 16-core server with 85 GB RAM at the peak, and calls variants in half an hour to an accuracy comparable to the current practice. FermiKit assembly is a reduced representation of raw data while retaining most of the original information.

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda fermikit
```

FSL

简介

FSL 是功能磁共振成像、核磁共振成像和 DTI 脑成像数据的综合分析工具库。

运行 FSL 的方式

使用 sbatch 提交运行脚本 (fsl.slurm):

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=fsl
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load fsl/6.0-fsl-gcc-4.8.5
fsl $PWD run.sh
```

脚本 `run.sh` 示例如下 (`fsl.slurm`、`run.sh` 和数据要在同一目录下)：

```
#!/bin/bash
eddy_correct DWI.nii.gz DWI_eddy.nii.gz 0
```

使用如下指令提交：

```
$ sbatch fsl.slurm
```

可视化运行方式 (在 **Studio** 里的远程桌面运行)

```
module load fsl/6.0-fsl-gcc-4.8.5
fsl_gui
```

GATK

简介

GATK 是 GenomeAnalysisToolkit 的简称，是一系列用于分析高通量测序后基因突变的工具集合。它提供一种工作流程，称作 “GATK Best Practices”。

CPU 容器版 GATK

使用 CPU 容器版 GTAK 时，需要先指定 GATK 镜像的路径。然后使用 `singularity exec 镜像路径 GTAK 命令` 的方式调用容器版 GATK。

示例脚本如下 (`gatk-container.slurm`)：

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
```

(续下页)

(接上页)

```
export IMAGE_NAME=/lustre/share/img/gatk-4.2.2.0.sif
singularity exec $IMAGE_NAME gatk --java-options "-Xmx128G" ...
```

使用如下指令提交:

```
$ sbatch gatk-container.slurm
```

ARM Spack 版 GATK

示例脚本如下 (gatk.slurm):

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module use /lustre/share/spack/modules/kunpeng920/linux-centos7-aarch64
module load gatk/4.2.0.0-gcc-9.3.0-openblas

gatk --java-options "-Xmx128G" ...
```

使用如下指令提交:

```
$ sbatch gatk.slurm
```

ARM 容器版 GATK

使用容器版 GTAK 时, 需要先指定 GATK 镜像的路径。然后使用 singularity exec 镜像路径 GTAK 命令的方式调用容器版 GATK。

示例脚本如下 (gatk-container.slurm):

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export IMAGE_NAME=/lustre/share/singularity/aarch64/gatk/gatk-4.2.0.0.sif
singularity exec $IMAGE_NAME gatk --java-options "-Xmx128G" ...
```

使用如下指令提交:

```
$ sbatch gatk-container.slurm
```

GEANT4

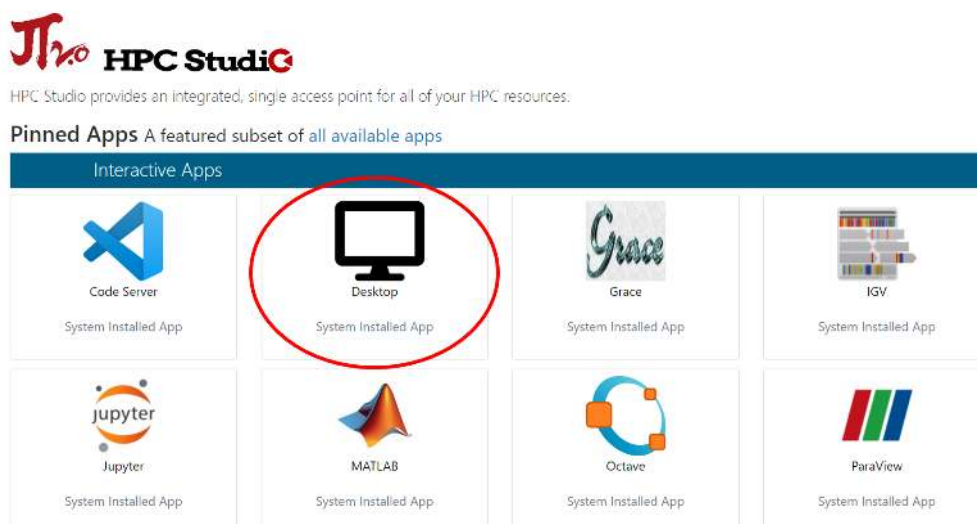
简介

Geant4(GEometry ANd Tracking, 几何和跟踪) 是由 CERN(欧洲核子研究组织) 基于 C++ 面向对象技术开发的蒙特卡罗应用软件包, 用于模拟粒子在物质中输运的物理过程。相对于 MCNP、EGS 等商业软件来说, 它的主要优点是源代码完全开放, 用户可以根据实际需要更改、扩充 Geant4 程序。详情请查阅 [Geant4 官网](#)

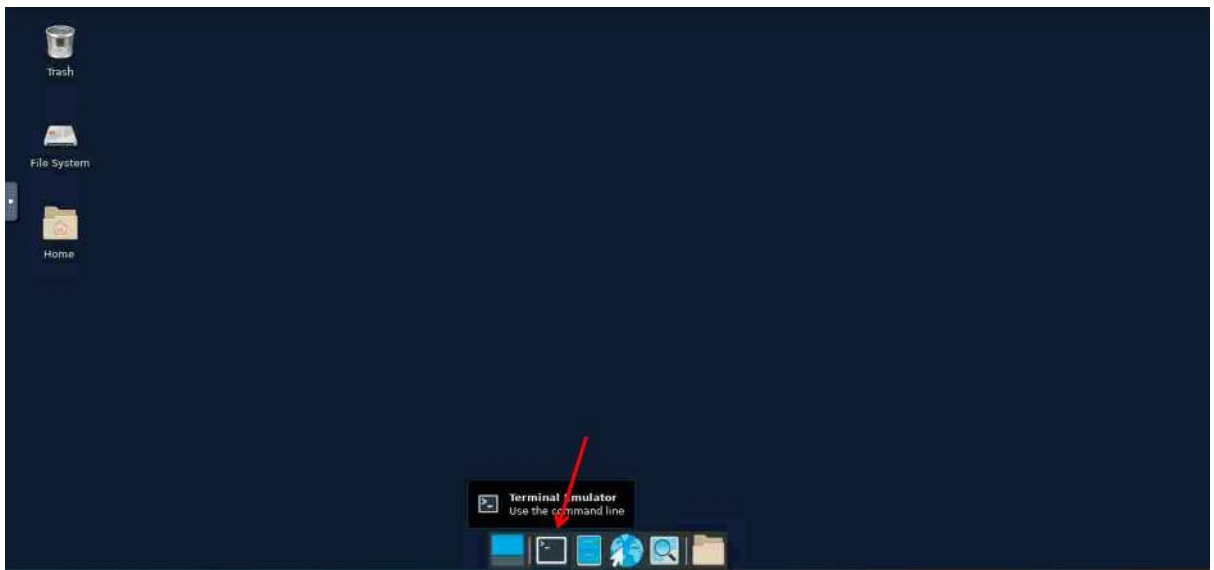
思源一号预编译 Geant4

思源一号平台上已预编译好基于 qt 可视化界面的 geant4 模块, 用户可通过加在相关模块制作 Geant4 可执行程序, 以编译 geant4 自带的 B1 example 为例

- 使用 HPC Studio 启动可视化界面。在浏览器中打开可视化平台 [HPC Studio](#), 选择 Desktop, 并设置合适的核数、时间等参数



- 启动终端



- 调用模块。在终端中输入命令，调用模块

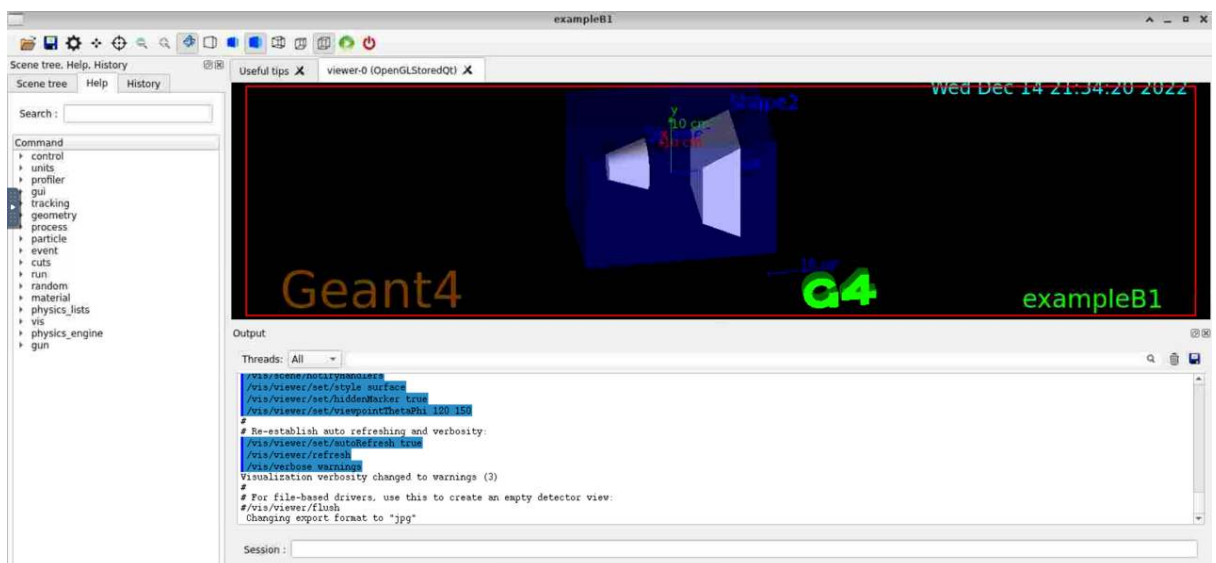
```
module load gcc/11.2.0 qt/5.15.5-gcc-11.2.0 geant4/11.0.3-gcc-11.2.0
```

- 制作 Geant4 可执行程序。本文档以 geant4 自带的 B1 example 为例，其中示例的源文件位于 Geant4 编译路径下，需先通过 env 命令查询 geant4 的安装路径（诸如/dssg/opt/icelake/linux-centos8-icelake/gcc-11.2.0/geant4-11.0.3-4lmd7rsbnlougezulj7hyhrch67ld4r，此处假设为/path_to_geant4）

```
mkdir B1_example_build
cd B1_example_build
cmake /path_to_geant4/share/Geant4-11.0.3/examples/basic/B1
make -j32
```

- 执行应用程序 exampleB1。经过上一步编译后将得到名称为 exampleB1 的可执行文件，执行该文件并得到可视化窗口

```
./exampleB1
```



思源一号自编译 Geant4

本文档将基于思源一号平台上已预编译好 qt 可视化驱动，进行 Geant4 手动编译，并制作可执行文件

- 使用 HPC Studio 启动可视化界面。如上所示
- 启动终端。如上所示
- 下载源码并解压。假定源文件压缩包下载路径为 (/path_to_source_code)，软件需要安装到路径 (/path_to_your_installation)

```
wget https://gitlab.cern.ch/geant4/geant4/-/archive/v11.0.0/geant4-v11.0.0.
→tar.gz
tar xvf geant4-v11.0.0.tar.gz
```

- 编译。进入安装目录 (/path_to_your_installation) 进行编译

```
cd /path_to_your_installation
cmake -DCMAKE_INSTALL_PREFIX=./ -DGEANT4_USE_FREETYPE=ON \
-DGEANT4_USE_GDML=ON \
-DGEANT4_USE_QT=ON \
-DGEANT4_USE_OPENGL=ON \
-DGEANT4_USE_OPENGL_X11=ON \
-DGEANT4_USE_RAYTRACER_X11=ON \
/path_to_source_code ## 指定源文件目录
make -j32 && make install
```

- 激活 Geant4 数据集。在安装目录 (/path_to_your_installation) 进行数据集编译

```
cmake -DGEANT4_INSTALL_DATA=ON .
make -j32 && make install
```

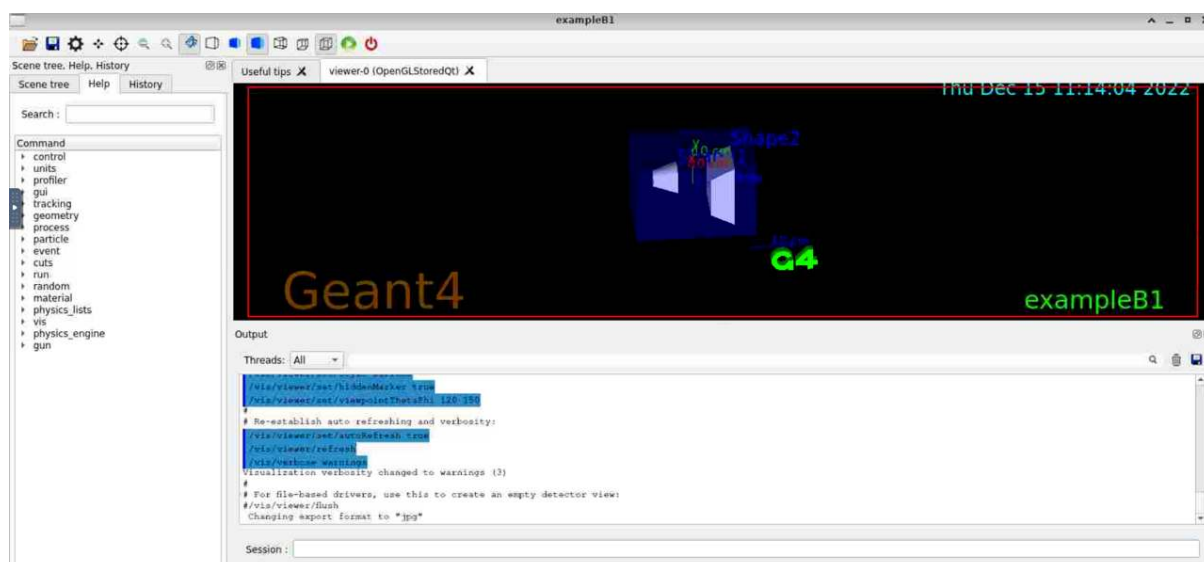
- 激活环境变量

```
source / (path_to_your_installation) /bin/geant4.sh
```

- 制作 Geant4 可执行程序。制作 B1 示例中的可执行程序

```
mkdir B1_example_build
cd B1_example_build
cmake / (path_to_your_installation) /share/Geant4-11.0.3/examples/basic/B1
make -j32
./exampleB1
```

- 得到可视化窗口



GenomeStrip

简介

Genome STRiP (Genome STRucture In Populations) is a suite of tools for discovery and genotyping of structural variation using whole-genome sequencing data. The methods used in Genome STRiP are designed to find shared variation using data from multiple individuals. Genome STRiP looks both across and within a set of sequenced genomes to detect variation.

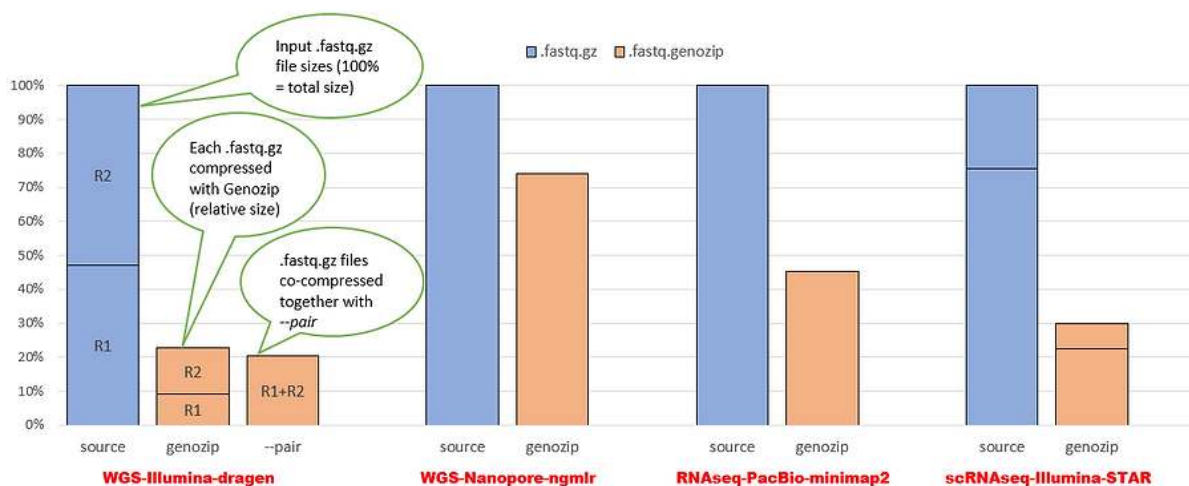
完整步骤

```
module load miniconda3
conda create -n mpyy_py27 python=2.7
source activate mpyy_py27
conda install -c bioconda genomestrip
```

GENOZIP

简介

Genozip 是用于基因组数据的通用且功能齐全的压缩软件。Genozip 通过提供五项核心功能——通用性（对所有常见基因组文件格式的支持）、高压缩率、速度、功能丰富性和可扩展性，被设计为基因组压缩的通用软件和开发框架。Genozip 为基因组学研究中广泛使用的基因组数据格式（即 FASTQ, SAM / BAM / CRAM, VCF, GVF, FASTA, PHYLIP 和 23andMe 格式）提供高性能压缩。



使用方法与范例

软件调用

```
module load genozip/15.0.56
```

License

拷贝 license 至家目录, 拷贝一次即可

```
cp $LIC ~
```

简单的压缩和解压缩

注意: 默认会使用当前节点全部的 CPU 核心, 可通过 `--threads N` 来调控线程数量, 如果需要多线程压缩解压缩, 请申请计算节点使用

```
genozip test.fq.gz
genounzip test.fq.genozip
```

指定 20 线程压缩

```
genozip --threads 20 test.fq.gz
```

使用参考压缩

为了对 FASTQ 和 BAM 文件实现良好的压缩, 强烈建议使用参考进行压缩

```
genozip --reference hs37d5.fa.gz mydata.fq.gz
genounzip mydata.fq.genozip
```

共同压缩一对 **FASTQ** 文件

当将它们压缩在一起时，**Genozip** 可以利用一对 **FASTQ** 文件中相应读取之间的冗余来提高压缩率，使用`--pair`：

```
genozip --reference hs37d5.fa.gz --pair mysample-R1.fastq.gz mysample-R2.  
↪fastq.gz
```

注意事项

- **genozip** 为收费开源软件, 对学术用户开放学术授权, 免费使用, 如果需要商用请至 **genozip** 官网购买 **license**

参考

- **genozip** compression quick guide: <https://www.genozip.com/compression>

GMAP-GSNAP

简介

mRNA 和 EST 序列的基因组定位和比对程序。

GRAPHMAP

简介

一个高度敏感和准确的映射器，用于长时间、容易出错的读取

GRIDSS

简介

CSS Grid Layout is a two-dimensional layout system for the web. It lets you lay content out in rows and columns, and has many features that make building complex layouts straightforward. This article will give you all you need to know to get started with page layout.

完整步骤

```
module load miniconda3  
conda create -n mpy_py27 python=2.7  
source activate mpy_py27  
conda install -c bioconda gridss
```

GsUtil

简介

gsutil 是一个 **Python** 应用程序，可让您从命令行访问 **Google Cloud Storage**。您可以使用 **gsutil** 执行各种存储桶和对象管理任务。

完整步骤

```
module load miniconda3
conda create -n mypy
source activate mypy
conda install -c conda-forge gsutil
```

Harmony

简介

Harmony 是一个用来整合不同平台的单细胞数据的方法。

安装方式

可以使用 **Conda** 进行安装，以思源一号为例：

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n harmony
source activate harmony
conda install r-harmony
```

参考资料

- [Harmony](#)

HISAT2

简介

HISAT2 是一种快速、灵敏的比对程序，用于将下一代测序读数（全基因组、转录组和外显子组测序数据）与普通人群（以及单个参考基因组）进行映射。基于 **GCSA**（图的 **BWT** 的扩展），我们设计并实现了图 **FM** 索引（**GFM**），这是一种原始方法，也是我们所知的第一个实现。除了使用一个代表一般人群的全局 **GFM** 索引外，**HISAT2** 还使用了一组大型的小型 **GFM** 索引，它们共同覆盖了整个基因组（每个索引代表一个 **56 Kbp** 的基因组区域，需要 **55,000** 个索引来覆盖人群）。这些小索引（称为本地索引）与多种对齐策略相结合，可以实现测序读数的有效对齐。

CPU 版本 Hisat2

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load hisat2/2.1.0-intel-19.0.4
hisat2-build -p 4 genome.fa genome
```

使用如下指令提交：

```
sbatch Hisat2.slurm
```

ARM 版本 Hisat2

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module use /lustre/share/spack/modules/kunpeng920/linux-centos7-aarch64
module load hisat2/2.1.0-gcc-9.3.0
hisat2-build -p 4 genome.fa genome
```

使用如下指令提交：

```
sbatch Hisat2.slurm
```

HMMER

简介

HMMER 和 BLAST 类似，主要用于序列比对。HMMER 是由 Sean Eddy 编写的用于序列分析的免费且常用的软件包。它的一般用法是鉴定同源蛋白质或核苷酸序列，并进行序列比对。

在 ARM 超算 HMMER 编译

该软件适用于 ARM 超算的正式版暂未发布，请关注软件官方网站等待正式版发布。若任务急需可申请计算节点后输入以下命令编译开发版。注意：开发版本可能存在问题，请尽量等待正式版本发布后再进行安装。

```
srunc -p arm128c256g -n 4 --pty /bin/bash
cd /YOUR/PATH/TO/HMMER
git clone -b h3-arm https://github.com/EddyRivasLab/hmmer.git
cd hmmer
git clone -b develop https://github.com/EddyRivasLab/easel.git
cd easel
autoconf
./configure --prefix=/YOUR/PATH/TO/HMMER/hmmer/easel
make
make check
make install
cd ..
autoconf
./configure --prefix=/YOUR/PATH/TO/HMMER/hmmer
make
make check
make install
```

Hydra-sv

简介

用于使用双端映射检测结构变异 (SV) 断点。与其他算法类似，Hydra 通过对不一致的双端对齐进行聚类来检测 SV 断点，其“签名”证实了相同的假定断点。Hydra 可以检测由所有类别的结构变异引起的断点。它还旨在检测独特和重复基因组区域的变异（例如，片段重复和转座子插入中的突变）；因此，它将检查具有多个不一致比对的配对末端读数。

完整步骤

```
wget https://storage.googleapis.com/google-code-archive-downloads/v2/code.
↪google.com/hydra-sv/Hydra.v0.5.3.tar.gz
wget https://storage.googleapis.com/google-code-archive-downloads/v2/code.
↪google.com/hydra-sv/bedpeToBed12.py
wget https://storage.googleapis.com/google-code-archive-downloads/v2/code.
↪google.com/hydra-sv/hydraFrequency.py
tar -zxvf Hydra.v0.5.3.tar.gz
cd Hydra-Version-0.5.3
make clean
make all
```

使用方法

```
cd bin
./hydra -h
```

km

简介

A software for RNA-seq investigation using k-mer decomposition

完整步骤

```
srunk -p small -n 4 --pty /bin/bash
git clone https://github.com/iric-soft/km.git
module load miniconda3
conda create -n mypy
source activate mypy
chmod +x easy_install.sh
./easy_install.sh
```

LUMPY-SV

简介

lumpy 是目前比较流行的一款 **SV** 检测工具，它同时支持 **PEM** 与 **SR** 和 **RD** 三种模式。在 **biostar** 上很多用户推荐，在 **lumpy** 所发的文章中，与 **Pindel**, **delly**, **gasvpro** 等软件比较，也有不错的效果。软件使用也非常容易，不仅支持 **gemrline** 样品，也支持 **somatic** 样品。

ARM 集群上的 LUMPY-SV

ARM 上的 **LUMPY-SV** 以容器的形式安装部署。镜像路径为 `/lustre/share/img/aarch64/lumpy-sv/lumpy-sv.sif`。

示例脚本如下 (**lumpy.slurm**):

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

export IMAGE_NAME=/lustre/share/img/aarch64/lumpy-sv/lumpy-sv.sif
singularity exec $IMAGE_NAME lumpyexpress \
  -B my.bam \
  -S my.splitters.bam \
```

(续下页)

(接上页)

```
-D my.discordants.bam \
-o output.vcf
```

使用如下指令提交：

```
$ sbatch lumpy.slurm
```

Lumpy

简介

lumpy 是目前比较流行的一款 SV 检测工具，它同时支持 PEM 与 SR 和 RD 三种模式。在 **biostar** 上很多用户推荐，在 **lumpy** 所发的文章中，与 **Pindel**, **delly**, **gasvpro** 等软件比较，也有不错的效果。软件使用也非常容易，不仅支持 **gemrline** 样品，也支持 **somatic** 样品。

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda lumpy-sv
```

MAKER

简介

MAKER is a portable and easily configurable genome annotation pipeline. Its purpose is to allow smaller eukaryotic and prokaryotic genome projects to independently annotate their genomes and to create genome databases. **MAKER** identifies repeats, aligns ESTs and proteins to a genome, produces ab-initio gene predictions and automatically synthesizes these data into gene annotations having evidence-based quality values. **MAKER** is also easily trainable: outputs of preliminary runs can be used to automatically retrain its gene prediction algorithm, producing higher quality gene-models on subsequent runs. **MAKER**'s inputs are minimal and its outputs can be directly loaded into a **GMOD** database. They can also be viewed in the **Apollo** genome browser; this feature of **MAKER** provides an easy means to annotate, view and edit individual contigs and BACs without the overhead of a database. **MAKER** should prove especially useful for emerging model organism projects with minimal bioinformatics expertise and computer resources.

完整步骤

```
srn -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda maker
```

Manta

简介

Manta 软件可以从比对文件中检测 SVs 和 indels。它主要开发用于检测单个样品的 germline 变异和 tumor/normal 配对样品的 somatic 变异。Manta 通过连续组装的方法可以使分辨率达到碱基级别，更有利于下游的注释和临床意义分析。Manta 软件接受输入 BAM 或 CRAM 格式文件，并以 VCF4.1 的格式报告所有的 SV 和 indels 突变。

ARM 版本 Manta

示例脚本如下 (manta.slurm):

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=128
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module use /lustre/share/spack/modules/kunpeng920/linux-centos7-aarch64
module load manta/1.6.0-gcc-9.3.0
configManta.py --bam test.bam --referenceFasta hg19.fa --runDir YOUR_MANTA_
↪ANALYSIS_PATH
```

使用如下指令提交：

```
$ sbatch manta.slurm
```

MEGAHIT

简介

Megahit 是一个二代测序从头组装工具，用于以时间和成本有效的方式组装大型和复杂的宏基因组数据。它在分别具有和不具有图形处理单元的单个计算节点上完成了 44.1 和 99.6 小时的 252Gbps 土壤宏基因组数据集的组装。

MELT

简介

MELTS 系列软件是模拟岩浆演化的工具。

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda melt
```

MetaSV

简介

Structural variations (SVs) are large genomic rearrangements that vary significantly in size, making them challenging to detect with the relatively short reads from next-generation sequencing (NGS).

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda metasv
```

METIS

简介

METIS 是由 Karypis Lab 开发的一个具有强大功能的图切分软件包。准确来说, METIS 是一个串行图切分的软件包, Karypis Lab 还提供了并行版的图切分软件包 parMETIS 和支持超图和电路划分的 hMETIS。METIS 的算法设计主要基于多层次递归二分切分法、多层次 K 路切分法以及多约束划分机制。用户使用 METIS 软件包时, 可以根据需要选择相应的切分方式。

Mfuzz

简介

Mfuzz 是一个时间序列表达模式聚类分析的 R 包。

安装方式

可以使用 Conda 进行安装, 以思源一号为例:

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n mfuzz
```

(续下页)

(接上页)

```
source activate mfuzz
conda install bioconductor-mfuzz
```

参考资料

- [Mfuzz](#)

MindTheGap

简介

暂无

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda mindthegap
```

Mobster

简介

A mob/pair programming timer, inspired by the MobProgramming/MobTimer.Python. Runs great on Mac, Windows, and Linux. Learn all about mobbing at the MobTimer.Python github page, and at mobprogramming.org.

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda mobster
```

Monocle 3

简介

Monocle 3 是一个分析单细胞基因表达数据的工具包。

安装方式

可以使用 Conda 进行安装，以思源一号为例：

```

srun -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n monocle3
source activate monocle3
conda install r-monocle3

```

参考资料

- [Monocle 3](#)

MrBayes

简介

MrBayes is a program for Bayesian inference and model choice across a wide range of phylogenetic and evolutionary models. MrBayes uses Markov chain Monte Carlo (MCMC) methods to estimate the posterior distribution of model parameters.

π 集群上的 MrBayes

查看 π 集群上已编译的软件模块:

```
$ module spider mrbayes
```

调用该模块:

```
$ module load mrbayes/3.2.7a-gcc-8.3.0-openmpi
```

π 集群上的 Slurm 脚本 `slurm.test`

在 `cpu` 队列上, 总共使用 16 核 ($n = 16$) `cpu` 队列每个节点配有 40 核, 这里使用了 1 个节点:

```

#!/bin/bash

#SBATCH --job-name=mrbayes
#SBATCH --partition=cpu
#SBATCH -n 16
#SBATCH --ntasks-per-node=16
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

ulimit -s unlimited
ulimit -l unlimited

```

(续下页)

(接上页)

```
module load mrbayes/3.2.7a-gcc-8.3.0-openmpi
srun --mpi=pmi2 mb your_input_file
```

根据我们的测试，**mrbayes** 最多只能使用 16 进程/节点的配置，请根据具体需要调整 `-n` 和 `--ntasks-per-node` 参数

π 集群上提交作业

```
$ sbatch mrbayes_cpu_gnu.slurm
```

参考资料

- [MrBayes 官网](#)

Minimap2

简介

Minimap2 是一个多功能的序列比对程序，可将 DNA 或 mRNA 序列与大型参考数据库进行比对。

可用版本

版本	平台	构建方式	模块名
2.14		spack	minimap2/2.14-gcc-11.2.0 Pi2.0

算例获取方式

```
pi2.0 :
mkdir ~/minimap2 && cd ~/minimap2
cp -r /lustre/opt/cascadelake/linux-rhel8-skylake_avx512/gcc-11.2.0/
↪minimap2-2.14-w7pbon3ooyyiysiroflg3ofh56yhptdj/data/* ./
```

任务脚本

```
#!/bin/bash
#SBATCH --job-name=minimap2
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
```

(续下页)

(接上页)

```
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load minimap2/2.14-gcc-11.2.0
minimap2 -a MT-human.fa MT-orang.fa > test.sam
```

参考资料

- [Minimap2](#)

NCBI-RMBLASTN

简介

BLAST 是“局部相似性基本查询工具”(Basic Local Alignment Search Tool) 的缩写。是由美国国立生物技术信息中心 (NCBI) 开发的一个基于序列相似性的数据库搜索程序。该程序将 DNA/蛋白质序列与公开数据库所有序列进行匹配比对，从而找到相似序列。

安装命令

以下给出在 Pi2.0 平台通过 `spack` 安装该软件的命令：

```
spack install ncbi-rmblastn@2.11.0 %gcc@8.5.0
```

以下给出在 Pi2.0 平台编译安装该软件的命令：

```
tar zxvf rmbblast-1.0-ncbi-blast-2.2.28+.tar.gz
cd rmbblast-1.0-ncbi-blast-2.2.28+-src/c++/
srun -n 4 -p cpu --pty /bin/bash
./configure --with-mt --prefix=path/to/rmbblastn --without-debug
make
make install
```

参考资料

- [RMBlast](#)
- [Rmbblastn Readme](#)
- [Rmbblastn tar](#)

OpenSlide-python

简介

OpenSlide 是一个 C 库，它提供了一个简单的界面来读取整张幻灯片图像（也称为虚拟幻灯片）。当前版本是 3.4.1，发布于 2015-04-20。Python 和 Java 绑定也可用。Python

绑定包括一个 **Deep Zoom** 生成器和一个简单的基于 **Web** 的查看器。Java 绑定包括一个简单的图像查看器。

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda openslide-python
conda install libiconv
```

pandas

简介

pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself. Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c anaconda pandas
```

PBSV

简介

pbsv is a suite of tools to call and analyze structural variants in diploid genomes from PacBio single molecule real-time sequencing (SMRT) reads. The tools power the Structural Variant Calling analysis workflow in PacBio's SMRT Link GUI. pbsv calls insertions, deletions, inversions, duplications, and translocations. Both single-sample calling and joint (multi-sample) calling are provided.

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda pbsv
```


PGA

简介

基于质谱 (MS) 的蛋白质组学数据通常是通过将实验质谱与从参考蛋白数据库中获得的理论质谱进行比较来进行肽鉴定来实现的, 但是, 这种策略不能识别新的肽和蛋白质序列。PGA 提供了基于 RNA-Seq 数据构建定制蛋白质数据库的功能, 有/没有基因组引导, 数据库搜索, 后处理和报告生成。这种定制的蛋白质数据库既包括参考数据库 (如 Refseq 或 ENSEMBL) 也包括形成 RNA-Seq 的新型肽序列数据。一般来说, 定制的蛋白质数据库包括以下四种新肽 (或蛋白质): 1) 单核苷酸变异 (SNV) 引起的肽; 2) 短插入和删除 (INDEL) 引起的肽; 3) 选择性剪接引起的肽; 4) 新的转录本编码肽。此外, PGA 还可以用于基于 RNA-Seq 数据从头组装的转录本序列创建蛋白质组学数据库。这种蛋白质组学数据库的构建策略在非模式生物的蛋白质组学研究中非常有用。

使用方法

```
singularity exec /dssg/share/imgs/pga/pga_1.15.4.sif R
```

PERCOLATOR

简介

Percolator 是从鸟枪法蛋白质组学数据集中鉴定肽的半监督学习。Percolator 是一种使用半监督机器的算法。学习提高正确和不正确频谱之间的区分标识。比赛来自搜索诱饵数据库提供分类器的负例, 以及来自目标数据库的高分匹配提供正面的例子。Percolator 训练机器学习一种称为支持向量机 (SVM) 的算法来区分通过将权重分配给多个特征。特征示例包括吉祥物得分、前体质量误差、片段质量误差, 变量数修改等。具有最佳权重的特征向量然后用于对来自所有查询的匹配项重新排序, 通常会提高敏感性。

使用方法

```
singularity exec /dssg/share/imgs/percolator/percolator_3.05.sif percolator ↵  
↵-h
```

完整步骤

```
module load miniconda3  
conda create -n mpy  
source activate mpy  
conda install -c bioconda percolate
```

PICARD

简介

Picard 是一组命令行工具, 用于处理高通量排序 (HTS) 数据和格式, 例如 SAM / BAM / CRAM 和 VCF。这些文件格式在 Hts-specs 存储库中定义。

Pindel

简介

Pindel can detect breakpoints of large deletions, medium sized insertions, inversions, tandem duplications and other structural variants at single-based resolution from next- gen sequence data. It uses a pattern growth approach to identify the breakpoints of these variants from paired-end short reads.

完整步骤

```
module load miniconda3
conda create -n mpyy_py27 python=2.7
source activate mpyy_py27
conda install -c bioconda pindel
```

PRISM

简介

A versatile statistics tool purpose-built for scientists-not statisticians. Get a head start by entering data into tables that are structured for scientific research and guide you to statistical analyses that streamline your research workflow.

完整步骤

```
module load miniconda3
conda create -n mpyy_py27 python=2.7
source activate mpyy_py27
conda install -c conda-forge pyprism
```

r-rgl

简介

Provides medium to high level functions for 3D interactive graphics, including functions modelled on base graphics (plot3d(), etc.) as well as functions for constructing representations of geometric objects (cube3d(), etc.). Output may be on screen using OpenGL, or to various standard 3D file formats including WebGL, PLY, OBJ, STL as well as 2D image formats, including PNG, Postscript, SVG, PGF.

完整步骤

```
module load miniconda3
conda create -n mpyy
source activate mpyy
conda install -c r r-rgl
```

RELION

简介

RELION 是由 MRC 的 Scheres 在 2012 年发布的针对单颗粒冷冻电镜图片进行处理的框架。

可用的版本

版本	平台	构建方式	模块名
4.0.0	gpu	容器	relion/4.0.0 思源一号
3.1.3	gpu	Spack	relion/3.1.3-gcc-11.2.0-cuda-openmpi 思源一号
4.0.0	gpu	容器	relion/4.0.0
3.0.8	gpu	容器	relion/3.0.8
3.1.3	gpu	Spack	relion/3.1.3-gcc-8.3.0-openmpi
3.0.8	gpu	Spack	relion/3.0.8-gcc-8.3.0-openmpi

算例下载

```
wget ftp://ftp.mrc-lmb.cam.ac.uk/pub/scheres/relion30_tutorial_data.tar
tar -xf relion30_tutorial_data.tar
# 目录结构
relion30_tutorial
├── Movies
│   ├── 20170629_000XX_frameImage.tiff
│   ├── gain.mrc
│   └── NOTES
```

运行示例

以下主要介绍 RELION 可视化的运行方式。用超算的账号及密码登录 **HPC Studio**，申请 GPU 远程桌面或直接打开 RELION。

小技巧

-pi 为 π 集群的资源，-sy 为思源一号的资源。

思源一号集群 **RELION**加载运行 **RELION**

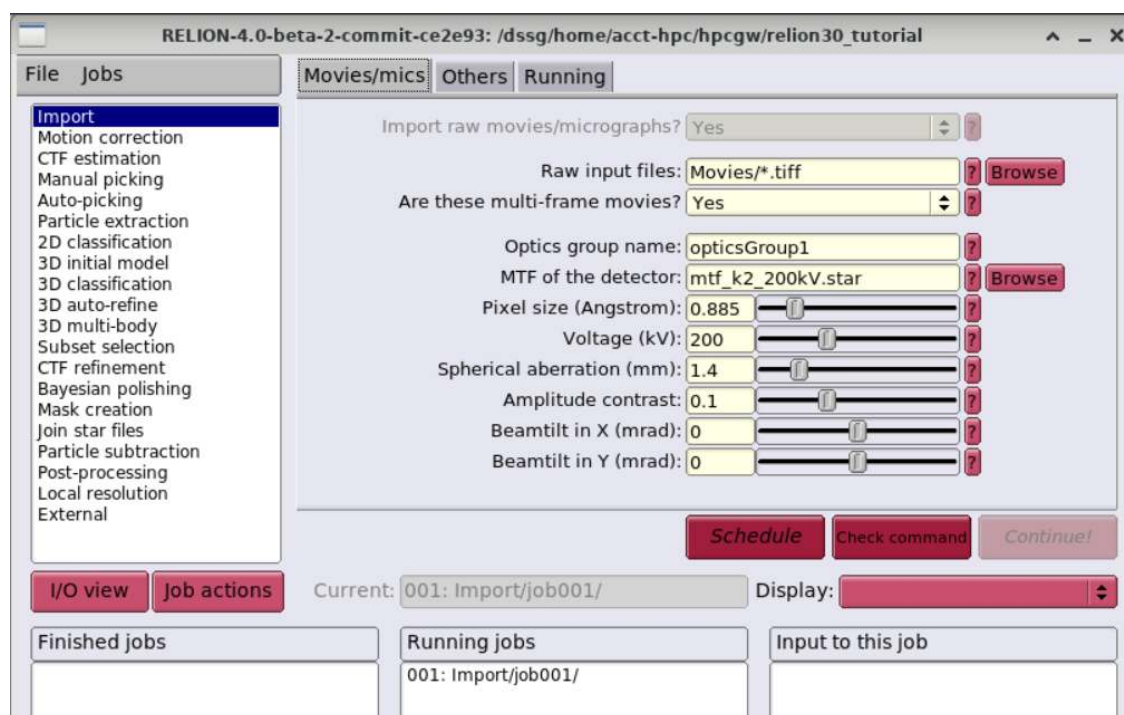
进入远程桌面，打开终端 *Terminal*。

```
cd relion30_tutorial
module load relion/4.0.0
relion &
```

预处理

建立项目

1. 用 Import 将数据导入流水线，Movies/mics 功能区参数如下：



2. 确保 Others 功能区是如下设置：

Import other node types? | No

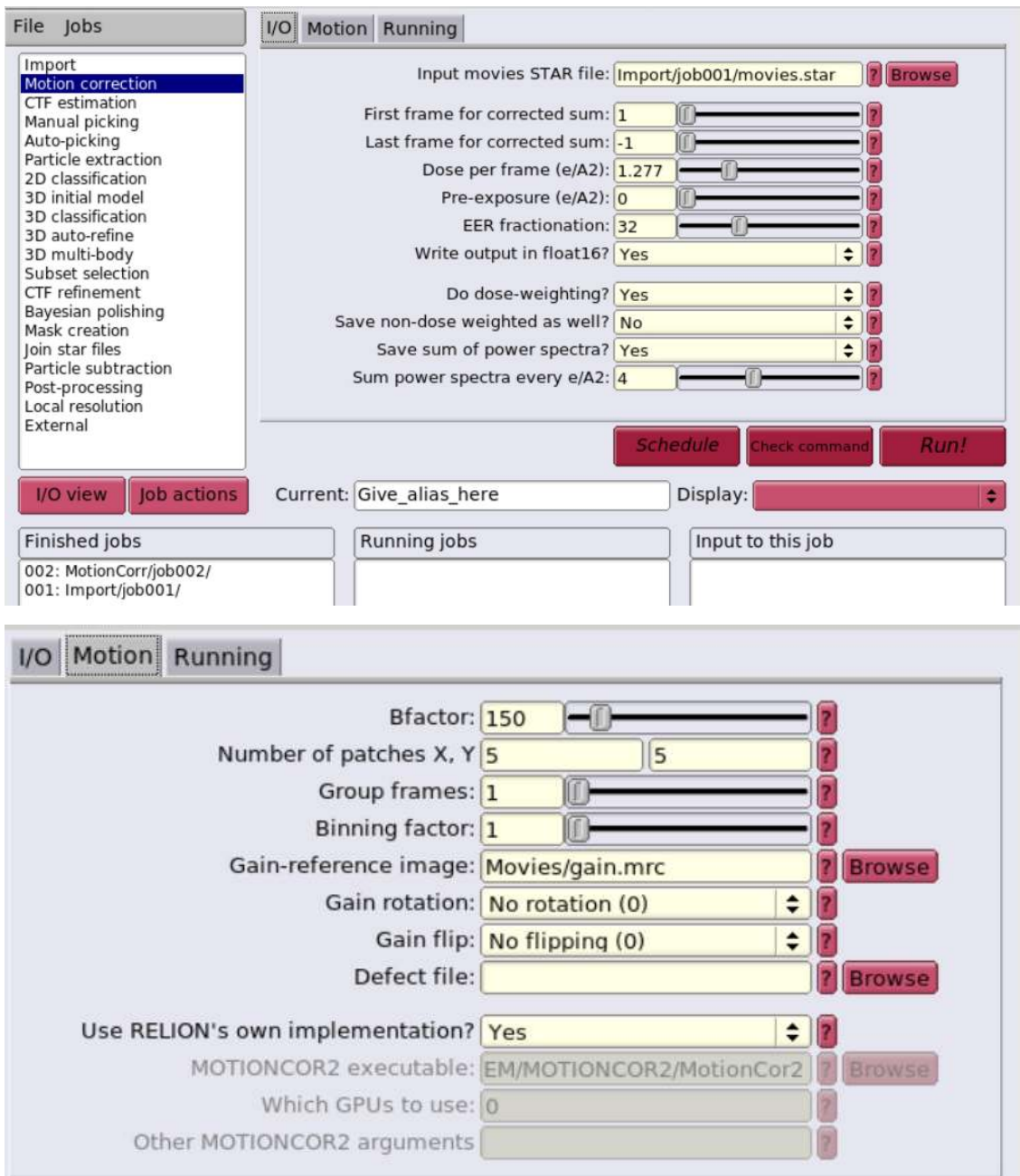
3. 点击 Run! 运行。

此时会创建 Import/job001/ 目录, 及 STAR 文件, 可以在终端通过 `less Import/job001/movies.star` 查看。

运动校正 (**Beam-induced motion correction**)

由于电子光束穿过薄样品，会对样品产生损耗并使其产生轻微位移，需要对每张图像进行运动校正，使图像的拍摄中心一致。

1. 用 Motion correction 进行运动校正，I/O 及 Motion 设置如下：



2. 设置 Running ， 点击 Run! 运行。

Number of MPI procs: | 1

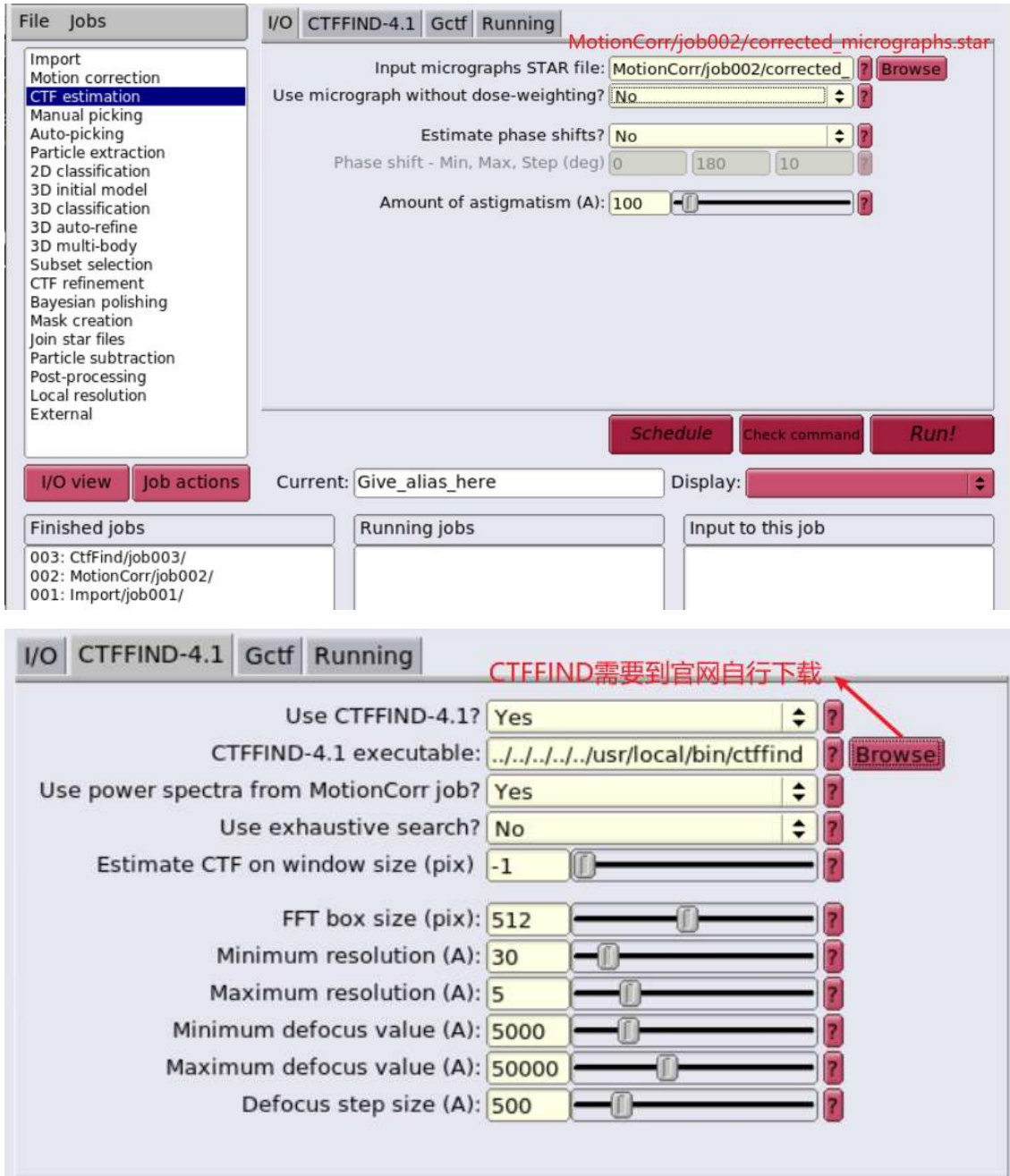
Number of threads: | 12

Submit to queue? | No

衬度转换函数估计 (*CTF estimation*)

由于电镜本身的成像过程，会存在球差、离焦量等问题，需要对其进行分析，找出 CTF，对图像进行校正。

1. 用 CTF estimation 进行运动校正，I/O 及 CTFFIND-4.1 设置如下：



2. 设置 Running , 点击 Run! 运行。

Number of MPI procs: | 4

颗粒挑选

初始图像集合 (***Subset of the micrographs***)

1. 用 Subset selection 创建自动挑选的模板, 设置如下:

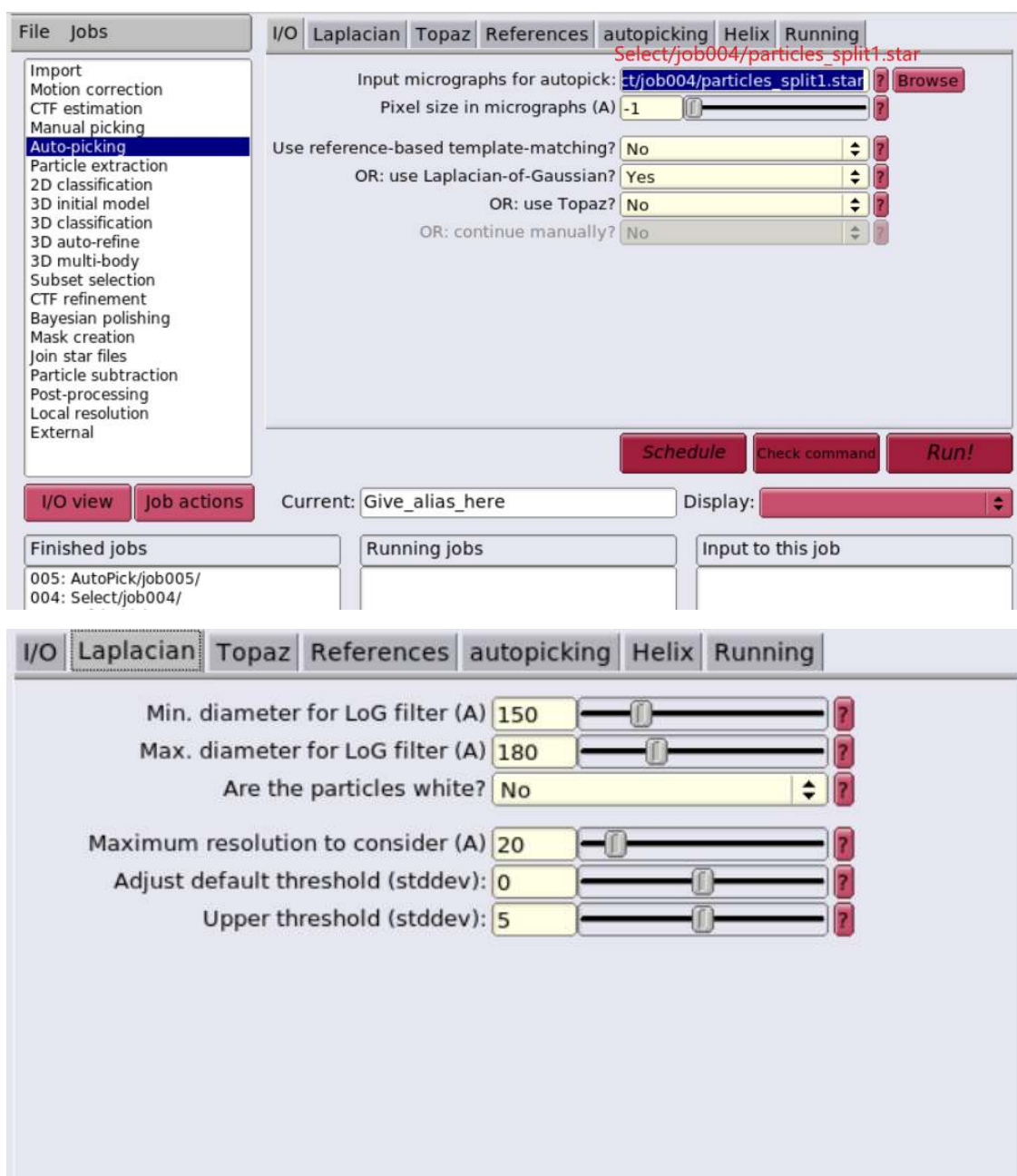
The screenshot displays the software's main configuration window. On the left, a 'File Jobs' menu lists various processing steps, with 'Subset selection' highlighted. The main area has tabs for 'I/O', 'Class options', 'Subsets', 'Duplicates', and 'Running'. Under 'I/O', there are fields to select classes from a job, micrographs.star, or particles.star, with a red text label 'CtfFind/job003/micrographs_ctf.star' below. At the bottom of this section are 'Schedule', 'Check command', and 'Run!' buttons. Below these are 'I/O view' and 'job actions' buttons, followed by a 'Current:' field set to 'Give_alias_here' and a 'Display:' dropdown. Three panels at the bottom show 'Finished jobs' (listing jobs 001-004), 'Running jobs' (empty), and 'Input to this job' (empty).

The second screenshot shows the 'Subsets' tab. It contains several configuration options: 'Select based on metadata values?' (No), 'Metadata label for subset selection:' (rlnCtfMaxResolution), 'Minimum metadata value:' (-9999), 'Maximum metadata value:' (9999), 'OR: select on image statistics?' (No), 'Metadata label for images:' (rlnImageName), 'Sigma-value for discarding images:' (4 with a slider), 'OR: split into subsets?' (Yes), 'Randomise order before making subsets?:' (No), 'Subset size:' (10 with a slider), and 'OR: number of subsets:' (-1 with a slider). Each option has a help icon (?) to its right.

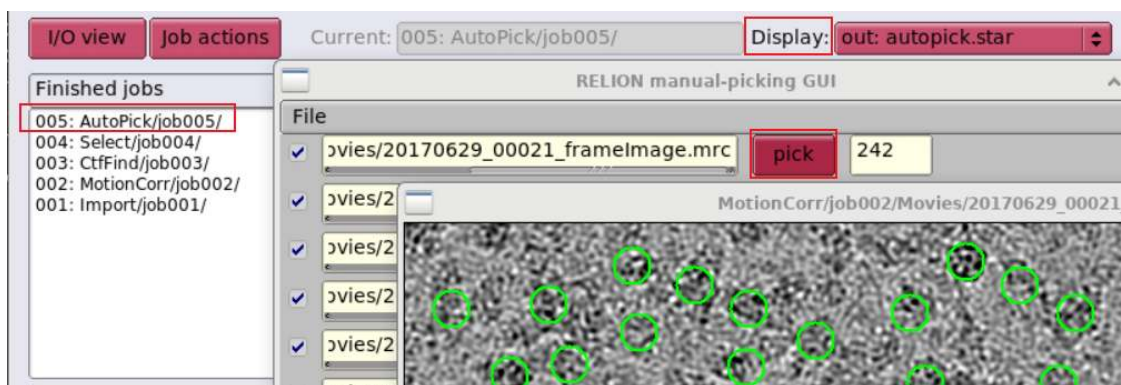
2. 点击 Run! 运行。

基于拉普拉斯-高斯算子的颗粒自动挑选 (*LoG-based auto-picking*)

1. 用 Auto-picking 自动挑选颗粒，设置如下：



2. 点击 Run! 运行;
3. 选中运行完的 job, Display 设置为 autopick.star, 查看颗粒挑选结果。



颗粒提取 (*particle extraction*)

1. 用 Particle extraction 提取颗粒，设置如下：

The screenshot displays the 'Particle extraction' software interface. The top-left sidebar lists various processing steps, with 'Particle extraction' highlighted. The main panel is divided into tabs: 'I/O', 'extract', 'Helix', and 'Running'. The 'extract' tab is active, showing configuration options for micrograph STAR files, input coordinates, and re-extraction settings. Below these, there are buttons for 'Schedule', 'Check command', and 'Run!'. At the bottom, there are sections for 'Finished jobs', 'Running jobs', and 'Input to this job'. The 'Running' tab is also visible, showing detailed parameters for particle extraction, such as box size, contrast inversion, normalization, and dust removal settings.

File Jobs

- Import
- Motion correction
- CTF estimation
- Manual picking
- Auto-picking
- Particle extraction**
- 2D classification
- 3D initial model
- 3D classification
- 3D auto-refine
- 3D multi-body
- Subset selection
- CTF refinement
- Bayesian polishing
- Mask creation
- Join star files
- Particle subtraction
- Post-processing
- Local resolution
- External

I/O extract Helix Running

micrograph STAR file:

Input coordinates:

OR re-extract refined particles?

Refined particles STAR file:

Reset the refined offsets to zero?

OR: re-center refined coordinates?

Recenter on - X, Y, Z (pix):

Write output in float16?

I/O view Job actions Current: Display:

Finished jobs **Running jobs** **Input to this job**

006: Extract/job006/
005: AutoPick/job005/

I/O extract Helix Running

Particle box size (pix):

Invert contrast?

Normalize particles?

Diameter background circle (pix):

Stddev for white dust removal:

Stddev for black dust removal:

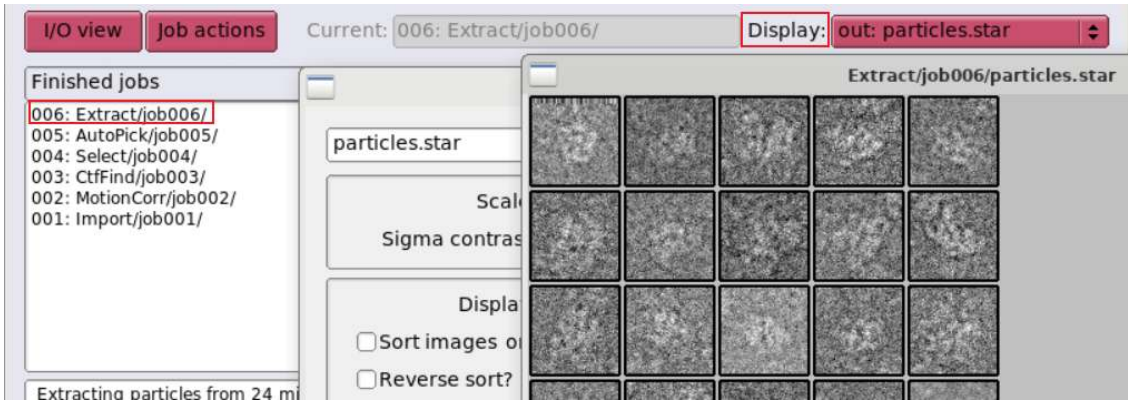
Rescale particles?

Re-scaled size (pixels):

Use autopick FOM threshold?

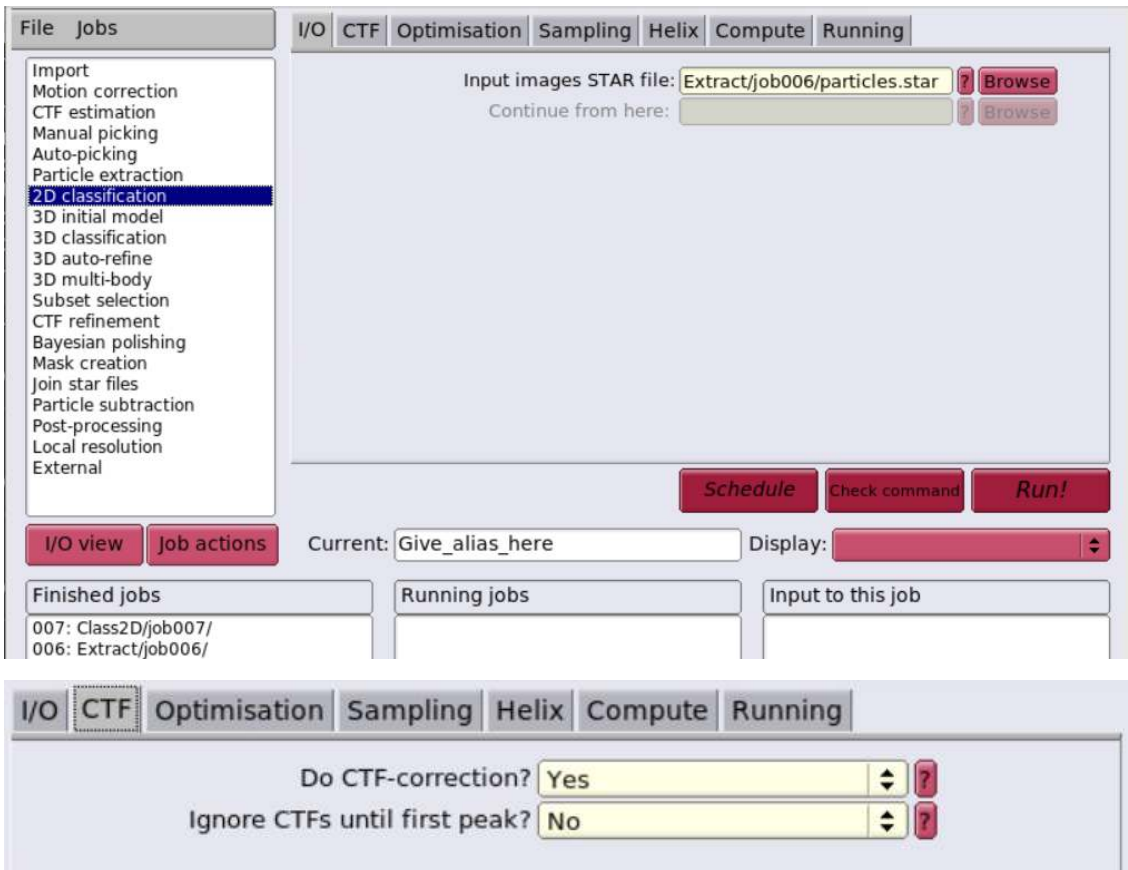
Minimum autopick FOM:

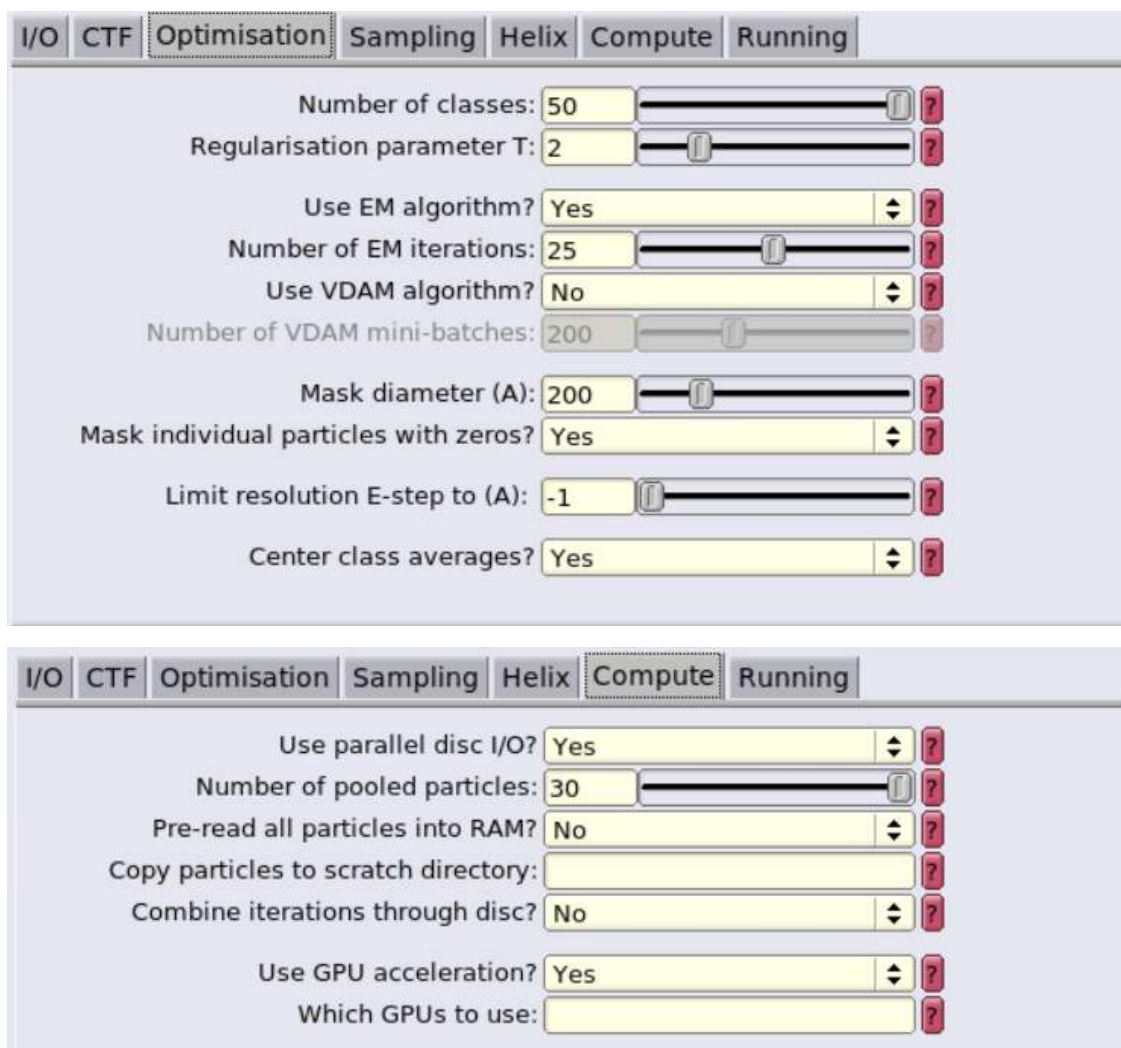
2. 点击 Run! 运行；
3. 选中运行完的 job，Display 设置为 particles.star，查看颗粒提取结果。



制作自动挑选的模板 (2D class averaging)

1. 用 2D classification 为接下来对所有图像进行自动挑选计算出一个模板，设置如下：





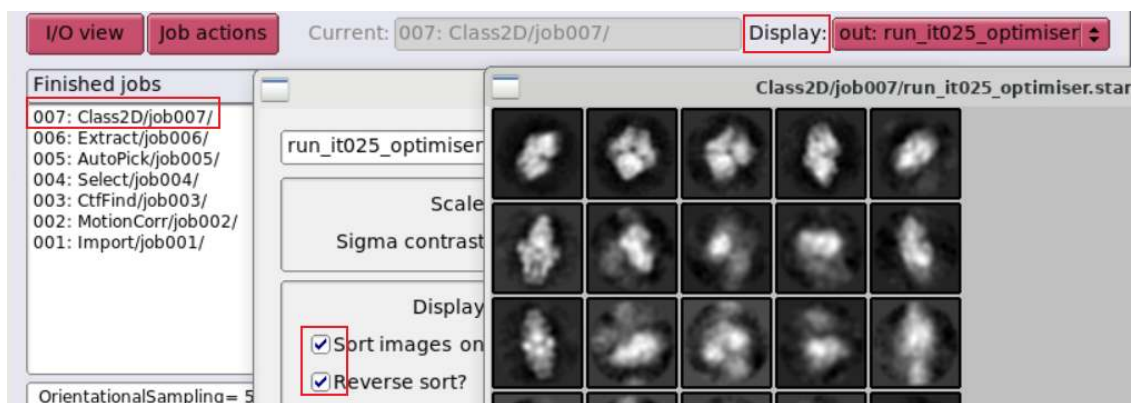
2. 设置 Running，点击 Run! 运行；

Number of MPI procs: | 3

Number of threads: | 4

Submit to queue? | No

3. 选中运行完的 job，Display 设置为 run_it025_optimiser.star，查看颗粒的模板。



π 集群 Relion

命令行运行方式

以下简单介绍非 GUI 的运行方式。

```

module av relion      # 查看  $\pi$  集群上已编译的 Relion
# 调用 Relion 及相关依赖
module load relion/3.1.3-gcc-8.3.0-openmpi
module load ghostscript/9.54.0-gcc-8.3.0    # 用于输出 pdf
module load openmpi/4.1.1-gcc-9.3.0
module load cuda/10.2.89-intel-19.0.4      # 用于支持 GPU

```

GPU 队列作业提交

在 dgx2 队列上使用 1 块 GPU，并配比 6 CPU 核心。

```

#!/bin/bash
#SBATCH -J test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=6
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:1

module load relion/3.1.3-gcc-8.3.0-openmpi
srun --mpi=pmi2 relion_refine_mpi (+params)

```

使用以下命令提交作业。

```

sbatch test.slurm

```

其它：作业模板设置

cpu 队列 small_slurm.bash

```

#!/bin/bash
#SBATCH --partition=small
# set the job name
#SBATCH --job-name=relion_job
# send output to
#SBATCH --output=XXXoutfileXXX
#SBATCH --error=XXXerrfileXXX
# this job requests x nodes
#SBATCH --nodes=1
# this job requests x tasks per node
#SBATCH --ntasks-per-node=XXXmpinodesXXX

```

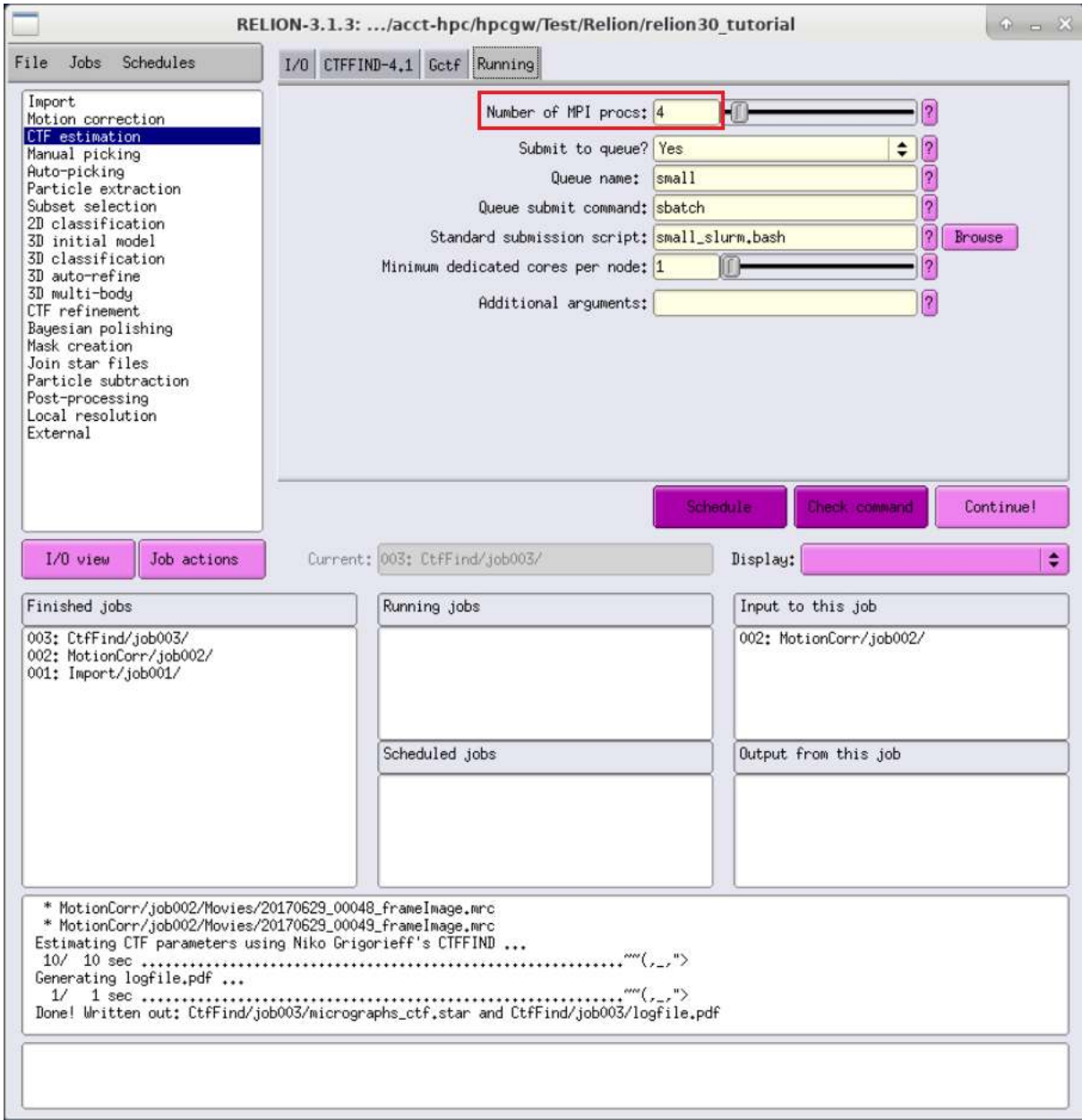
(续下页)

(接上页)

```
#SBATCH --export=ALL
#paste the "print command" from relion here
mpirun -n XXXmpinodesXXX --mca mpi_cuda_support 0 XXXcommandXXX
```

gpu 队列 xGPU_slurm.bash

```
#!/bin/bash
#SBATCH --partition=dgx2
# set the job name
#SBATCH --job-name=relion_job
# send output to
#SBATCH --output=XXXoutfileXXX
#SBATCH --error=XXXerrfileXXX
# this job requests x nodes
#SBATCH --nodes=1
# this job requests x tasks per node
#SBATCH --ntasks-per-node=XXXmpinodesXXX
#SBATCH --cpus-per-task=6
#SBATCH --time=48:00:00
#SBATCH --gres=gpu:4
#SBATCH --export=ALL
#paste the "print command" from relion here
mpirun -n XXXmpinodesXXX XXXcommandXXX
```



参考资料

- RELION 官网
- RELION 4.0
- RELION on Biowulf
- RELION Tutorial

RetroSeq

简介

RetroSeq is a tool for discovery and genotyping of transposable element variants (TEVs) (also known as mobile element insertions) from next-gen sequencing reads aligned to a reference

genome in BAM format. The goal is to call TEVs that are not present in the reference genome but present in the sample that has been sequenced. It should be noted that RetroSeq can be used to locate any class of viral insertion in any species where whole-genome sequencing data with a suitable reference genome is available.

完整步骤

```
git clone https://github.com/tk2/RetroSeq.git
cd RetroSeq/bin
./retroseq.pl -h
```

RFdiffusion

简介

RFdiffusion is an open source method for structure generation, with or without conditional information (a motif, target etc). It can perform a whole range of protein design challenges as we have outlined in the RFdiffusion paper.

Things Diffusion can do:

- Motif Scaffolding
- Unconditional protein generation
- Symmetric unconditional generation (cyclic, dihedral and tetrahedral symmetries currently implemented, more coming!)
- Symmetric motif scaffolding
- Binder design
- Design diversification ("partial diffusion", sampling around a design)

安装步骤

1. 克隆 github 仓库

```
git clone https://github.com/RosettaCommons/RFdiffusion.git
```

2. 下载模型权重文件

```
cd RFdiffusion
mkdir models && cd models
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↳ 6f5902ac237024bdd0c176cb93063dc4/Base_ckpt.pt
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↳ e29311f6f1bf1af907f9ef9f44b8328b/Complex_base_ckpt.pt
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↳ 60f09a193fb5e5ccdc4980417708dbab/Complex_Fold_base_ckpt.pt
wget http://files.ipd.uw.edu/pub/RFdiffusion/
```

(续下页)

(接上页)

```

↪74f51cfb8b440f50d70878e05361d8f0/InpaintSeq_ckpt.pt
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↪76d00716416567174cdb7ca96e208296/InpaintSeq_Fold_ckpt.pt
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↪5532d2e1f3a4738dec58b19d633b3c3/ActiveSite_ckpt.pt
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↪12fc204edeae5b57713c5ad7dcb97d39/Base_epoch8_ckpt.pt

Optional:
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↪f572d396fae9206628714fb2ce00f72e/Complex_beta_ckpt.pt

# original structure prediction weights
wget http://files.ipd.uw.edu/pub/RFdiffusion/
↪1befcb9b28e2f778f53d47f18b7597fa/RF_structure_prediction_weights.pt1

```

3. 在 conda 中安装软件

```

srun -n 1 -p a100 --gres=gpu:1 --pty /bin/bash
module load miniconda3
module load gcc

conda env create -f env/SE3nv.yml

conda activate SE3nv
cd env/SE3Transformer
pip install --no-cache-dir -r requirements.txt
python setup.py install
cd ../../ # change into the root directory of the repository
pip install -e . # install the rfdiffusion module from the root of the
↪repository

```

软件使用

测试软件是否能够正常使用

```

srun -n 1 -p a100 --gres=gpu:1 --pty /bin/bash
module load miniconda3
module load gcc

source activate SE3nv
./scripts/run_inference.py 'contigmap.contigs=[150-150]' inference.output_
↪prefix=test_outputs/test inference.num_designs=10

```

如果能够正常在主屏幕输出且无报错，则软件可用。更多参数和使用方式参见官网：<https://github.com/RosettaCommons/RFdiffusion>

RNA-SEQC

简介

Index 测序的标签，用于测定混合样本，通过每个样本添加的不同标签进行数据区分，鉴别测序样品。

RoseTTAFold

简介

RoseTTAFold 由华盛顿大学 David Baker 团队开发，利用深度学习技术准确、快速地预测蛋白质结构。

RoseTTAFold 版本

交大 AI 平台部署了两个版本的 **RoseTTAFold**：- 1.0 版本：最新更新日期为 2021 年 7 月 31 日 - 1.1 版本：最新更新日期为 2023 年 10 月 16 日

1. 1.0 版本使用说明

```
rosettafold/1-python-3.8
```

1.1 使用前准备

- 新建文件夹，如 `rosettafold`。
- 在文件夹里放置一个 `fasta` 文件。例如 `test.fasta` 文件（内容如下）：

```
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLDDR
```

- 另外，还要在文件夹里新建一个输出文件夹，如 `output`，确保文件夹里为空。

1.2 运行 RoseTTAFold

作业脚本示例（假设作业脚本名为 `rosettafold.slurm`）：

```
#!/bin/bash
#SBATCH --job-name=rosettafold
#SBATCH --partition=dgx2
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=6
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -x vol08
```

(续下页)

(接上页)

```
module load rosettafold/1-python-3.8  
  
run_pyrosetta $PWD test.fasta output
```

说明:

- 可修改 `test.fasta` 和 `output`, 指定输入文件和输出文件夹。
- `$PWD` 指当前路径, 也可以用绝对路径指定 **RoseTTAFold** 的主文件夹, 以便从其他路径运行上述命令。

作业提交命令:

```
sbatch rosettafold.slurm
```

1.3 注意事项

- 上述示例运行约需 1 个小时。
- 欢迎邮件联系我们, 反馈软件使用情况, 或提出宝贵建议。

2. 1.1 版本使用说明

此版本无需 `module` 加载 `rosettafold`, 直接使用 `conda` 镜像, 保持了跟官方文档一致的命令使用方式。

2.1 使用前准备

如果需要经常使用 **RoseTTAFold**, 可以在 `~/.bashrc` 文件中添加如下变量:

```
export RoseTTAFold=/lustre/opt/contribute/cascadelake/RoseTTAFold/data/  
↪RoseTTAFold_1.1
```

2.2 运行 RoseTTAFold

作业脚本示例:

```
#!/bin/bash  
#SBATCH --job-name=rosettafold  
#SBATCH --partition=dgx2  
#SBATCH -N 1  
#SBATCH --ntasks-per-node=1  
#SBATCH --cpus-per-task=6  
#SBATCH --gres=gpu:1  
#SBATCH --output=%j.out  
#SBATCH --error=%j.err  
#SBATCH -x vol08
```

(续下页)

(接上页)

```
module load miniconda3
source activate /lustre/share/conda_env/RoseTTAFold
export RoseTTAFold=/lustre/opt/contribute/cascadelake/RoseTTAFold/data/
↪RoseTTAFold_1.1

/bin/bash $RoseTTAFold/run_pyrosetta_ver.sh $RoseTTAFold/example/input.fa
↪output
```

说明:

- 可修改 /example/input.fa 为指定输入文件, output 为指定输出文件夹。
- 如果已经在 ~/.bashrc 文件中添加变量 RoseTTAFold, 脚本中无需重复添加。
- 所有 RoseTTAFold 涉及命令均与官方文档保持一致, 可以自行参考使用

参考资料

- RoseTTAFold GitHub: <https://github.com/RosettaCommons/RoseTTAFold>
- RoseTTAFold 论文: <https://www.biorxiv.org/content/10.1101/2021.06.14.448402v1>

SALMON

简介

一种从 RNA-seq 数据中快速量化转录本的工具

Samtools

简介

Samtools 是一个用于操作 SAM(Sequence Alignment/Map) 和 BAM 文件的工具合集, 包含许多命令。

可用的版本

版本	平台	构建方式	模块名
1.13		Spack	<i>samtools/1.13-gcc-11.2.0</i> 思源一号
1.16.1		Spack	samtools/1.16.1-gcc-8.5.0 Pi 2.0
1.10		Spack	<i>samtools/1.10-gcc-9.3.0</i>
1.9		Spack	samtools/1.9-intel-19.0.4
1.9		Spack	samtools/1.9-gcc-8.3.0
1.9		Spack	samtools/1.9-gcc-4.8.5

使用 Conda 安装 SAMtools

推荐使用 Conda 在用户目录部署特定的 Samtools 软件，以思源一号为例：

```
srunk -p 64c512g -n 4 --pty /bin/bash

module load miniconda3/4.10.3
conda create -n biotools                # 创建新的环境
source activate biotools                 # 激活环境
conda install -c bioconda samtools=1.13 # 安装Samtools
samtools --help
```

示例文件

```
module load samtools
samtools view -h https://storage.googleapis.com/genomics-public-data/
→platinum-genomes/bam/NA12877_S1.bam chr20:100000-400000 > test.sam
```

运行示例

思源一号集群 SAMtools

在思源一号集群上使用如下命令：

```
srunk -p 64c512g -n 4 --pty /bin/bash
module load samtools/1.13-gcc-11.2.0
samtools --help
```

ARM 集群 SAMtools

在 ARM 节点上使用如下命令：

```
srunk -p arm128c256g -n 4 --pty /bin/bash
module load samtools/1.10-gcc-9.3.0
samtools --help
```

π 集群 SAMtools

在 π 集群上使用如下命令：

```
srunk -p small -n 4 --pty /bin/bash
module load samtools/1.16.1-gcc-8.5.0
samtools --help
```

SAM 格式介绍

`samtools view -h test.sam | less -S` 可以查看 SAM 文件的内容，是记录比对信息的标准结构化文件。

SAM 文件头部

SAM 文件头部信息以 `@` 作为开头，对对比结果的 meta 信息进行记录。主要包含了文件标准格式版本 `VN`、比对中使用的参考序列信息 `SQ`、测序数据分组信息 `RG`、比对或后期处理使用的程序信息 `PG` 等。关于 SAM 文件头部信息的细节可以参考 [格式说明手册](#)。

SAM 文件比对信息

列编号	列名称	类型	说明
1	QNAME	String	query 序列名称
2	FLAG	Int	FLAG 标签，主要记录比对的基本情况
3	RNAME	String	比对至参考序列的名称，例如：chr1
4	POS	Int	比对至参考序列的位置
5	MAPQ	Int	比对质量
6	CIGAR	String	比对的 CIGAR 字符串
7	RNEXT	String	read2 比对到的参考序列位置名称
8	PNEXT	Int	表示 read2 比对的参考序列位置
9	TLEN	Int	比对序列对应的模板长度
10	SEQ	String	query 序列
11	QUAL	String	query 序列的碱基质量 Phred 值

上述 11 列信息，是 SAM 文件必要的组成部分。除此之外，每行可以追加可选信息。该部分信息以 `TAG:TYPE:VALUE` 形式存储。详细介绍可参考 [官方文档](#)。

Samtools 常用命令

```
samtools view -bS test.sam > test.bam # 将sam文件转换成bam文件
samtools index test.bam # 建立文件索引
samtools view -bF 4 test.bam > test.mapped.bam #
↪提取比对到参考序列上的比对结果
samtools view test.bam chr20:100000-200000 > chr20_100k-200k.sam #
↪提取chr20上100k到200k区域的比对结果
samtools sort test.bam -o test.sort.bam # 对bam文件进行排序
samtools tview test.bam -p chr20:100000 # 可交互的IGV浏览器
```

参考资料

- [Samtools 文档](#)

Seurat

简介

Seurat 是由 *New York Genome Center, Satija Lab* 开发的单细胞数据分析集成软件包。其功能不仅包含基本的数据分析流程，如质控，细胞筛选，细胞类型鉴定，特征基因选择，差异表达分析，数据可视化等。同时也包括一些高级功能，如时序单细胞数据分析，不同组学单细胞数据整合分析等。

安装方式

可以使用 Conda 进行安装，以思源一号为例：

```
srun -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n seurat
source activate seurat
conda install r-seurat
```

参考资料

- [Seurat](#)

Sniffles

简介

Sniffles is a structural variation caller using third generation sequencing (PacBio or Oxford Nanopore). It detects all types of SVs (10bp+) using evidence from split-read alignments, high-mismatch regions, and coverage analysis. Please note the current version of Sniffles requires sorted output from BWA-MEM (use -M and -x parameter), Minimap2 (sam file with Cigar & MD string) or NGMLR.

完整步骤

```
module load miniconda3
conda create -n mpyy
source activate mpyy
conda install -c bioconda sniffles
```

SOAPDENOV02

简介

SOAPdenovo2 是用于序列组装的常用软件。

sra-tools

简介

来自 NCBI 的 SRA 工具包和 SDK 是用于使用 INSDC 序列读取档案中的数据的工具和库的集合。

完整步骤

```
srunch -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda sra-tools
```

源码安装步骤

```
srunch -p small -n 4 --pty /bin/bash
mkdir -p ${HOME}/01.application/13.sra-tools && cd ${HOME}/01.application/
→13.sra-tools
wget --output-document sratoolkit.tar.gz https://ftp-trace.ncbi.nlm.nih.gov/
→sra/sdk/current/sratoolkit.current-centos_linux64.tar.gz
tar -vzxzf sratoolkit.tar.gz
export PATH=${HOME}/01.application/13.sra-tools/sratoolkit.3.0.0-centos_
→linux64/bin:$PATH
which fasterq-dump
```

参考资料

- sra-tools

SRATOOLKIT

简介

sra toolkit 是 ncbi 上将.sra 文件转换为.fasta.gz 文件的工具。

STAR-Fusion

简介

STAR-Fusion 是一款基于 STAR 比对结果进行融合基因鉴定的软件。

安装方式

可以使用 **Conda** 进行安装，以思源一号为例：

```
srun -p 64c512g -n 4 --pty /bin/bash
module load miniconda3/4.10.3
conda config --add channels defaults
conda config --add channels bioconda
conda config --add channels conda-forge
conda create -n star-fusion
source activate star-fusion
conda install star-fusion
```

参考资料

- [STAR-Fusion](#)

STRINGTIE

简介

StringTie 是一个新的转录组表达定量软件。

STRique

简介

STRique is a python package to analyze repeat expansion and methylation states of short tandem repeats (STR) in Oxford Nanopore Technology (ONT) long read sequencing data.

在 π 集群上安装 STRique

首先申请计算节点，然后输入以下指令进行编译：

```
$ srun -p small -n 4 --pty /bin/bash
$ module load miniconda3/4.7.12.1-gcc-4.8.5
$ conda create -n teststr python=3.6
$ source activate teststr
$ git clone --recursive https://github.com/giesselmann/STRique
$ cd STRique
$ pip install -r requirements.txt
$ python setup.py install
```

在 π 集群上运行的 **Slurm** 脚本示例：

```
#!/bin/bash

#SBATCH -J test
```

(续下页)

(接上页)

```
#SBATCH -p small
#SBATCH -n 4
#SBATCH --ntasks-per-node=4
#SBATCH -o %j.out
#SBATCH -e %j.err

module load miniconda3/4.7.12.1-gcc-4.8.5
source activate teststr

cd ~/STRique/data
python ../scripts/STRique.py index --recursive c9orf72.fast5 > c9orf72.fofn
cat c9orf72.sam | python ../scripts/STRique.py count c9orf72.fofn ../models/
→r9_4_450bps.model ../configs/repeat_config.tsv > c9orf72.hg19.strique.tsv
cat c9orf72.hg19.strique.tsv | python ../scripts/STRique.py plot c9orf72.
→fofn --output c9orf72.pdf --format pdf
```

参考资料

- STRique

SV2

简介

SV2 Software is IT-consulting innovative company. The company provides wide range of software solutions in different areas - from development hardware drivers till web-applications. SV2 Software - when programming is art.

完整步骤

```
module load miniconda3
conda create -n mpy_py27 python=2.7
source activate mpy_py27
conda install -c bioconda sv2
```

SvABA

简介

SvABA 是一种使用全基因组局部组装检测测序数据中结构变异的方法。

使用 **module** 运行 **svaba**

使用 **sbatch** 提交运行脚本 (svaba.slurm):

```
#!/bin/bash
```

(续下页)

(接上页)

```
#SBATCH --job-name=svaba
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=4
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load svaba/1.1.3-gcc-4.8.5
svaba run.sh
```

脚本 `run.sh` 示例如下 (`svaba.slurm`、`run.sh` 和数据要在同一目录下):

```
#!/bin/bash
svaba -t tumor.bam -n normal.bam -k 22 -G ref.fa -a test_id -p -4
```

使用如下指令提交:

```
$ sbatch svaba.slurm
```

使用 **conda** 安装

```
module load miniconda3
conda create -n mypy
source activate mypy
conda install -c bioconda svaba
```

SVDetect

简介

SVDetect is a application for the isolation and the type prediction of intra- and inter-chromosomal rearrangements from paired-end/mate-pair sequencing data provided by the high-throughput sequencing technologies.

完整步骤

```
module load miniconda3
conda create -n mypy
source activate mypy
conda install -c imperial-college-research-computing svdetect
```

TOPHAT

简介

Tophat 使用 RNA-seq 的 reads 数据来寻找基因的剪切点 (splice junction)。该软件调用 Bowtie, 或 Bowtie2 来将 reads 比对到参考基因组上, 分析比对结果, 从而寻找出外显

子之间的结合位点。

VARDICTJAVA

简介

VarDictJava 是一个用 Java 和 Perl 编写的变种发现程序。它是 VarDict 变体调用者的 Java 端口。

VariationHunter

简介

VariationHunter is a package of programs used to find structural variations using paired-end reads read mappings. 目前，VariationHunter 已经停止更新，推荐使用 tardis 进行结构变异预测。

完整步骤

```
git clone https://github.com/BilkentCompGen/tardis.git --recursive
cd tardis
make libs
make
./tardis -h
```

VSEARCH

简介

VSEARCH 是一个开源免费的 64 位，无内存限制的扩增子数据处理分析软件。该软件是专门针对 Edgar 大神开发的 USEARCH 软件而设计开发的 (Rognes, 2016)。

WGCNA

简介

Weighted correlation network analysis, also known as weighted gene co-expression network analysis (WGCNA), is a widely used data mining method especially for studying biological networks based on pairwise correlations between variables.

完整步骤

```
module load miniconda3
conda create -n mpy
source activate mpy
conda install -c bioconda r-wgcna
```

Wham

简介

加权直方图分析方法。

完整步骤

```
module load miniconda3
conda create -n mypy
source activate mypy
conda install -c bioconda wham
```

Rosetta

简介

Rosetta 软件套件包括用于蛋白质结构计算建模和分析的算法。它使计算生物学取得了显著的科学进步，包括从头蛋白质设计、酶设计、配体对接以及生物大分子和大分子复合物的结构预测。

RoeTeta 开发始于华盛顿大学 **David Baker** 博士的实验室，作为结构预测工具，但此后已经适应于解决常见的计算大分子问题。

Rosetta 的重要作用如下所示：

理解大分子相互作用; 设计定制分子; 寻找构象和序列空间的有效方法; 为各种生物分子表示寻找广泛有用的能量函数

可用的版本

版本	平台	构建方式	模块名
3.12		容器	rosetta/3.12 思源一号
3.12		容器	rosetta/3.12

算例下载

```
思源一号：
mkdir ~/test_rosetta
cd ~/test_rosetta
cp -r /dssg/share/sample/rosetta/input_files ./
mkdir output_files

π2.0：
mkdir ~/test_rosetta
cd ~/test_rosetta
```

(续下页)

(接上页)

```
cp -r /lustre/share/samples/rosetta/input_files ./
mkdir output_files
```

集群上的 Rosetta

- 一. 思源一号 *Rosetta*
- 二. $\pi 2.0$ *Rosetta*

一. 思源一号 Rosetta

1. 对输入结构进行预处理 (refine)

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun relax.mpi.linuxgccrelease -in:file:s input_files/1qys.pdb -nstruct 2
→-relax:constrain_relax_to_start_coords -relax:ramp_constraints false -ex1
→-ex2 -use_input_sc -flip_HNQ -no_optH false
```

输入与参数说明

```
in:file:s                #输入数据
nstruct
→#nstruct 可以提高模型结果的质量，如nstruct 10将会获得10个模型
relax:constrain_relax_to_start_coords
→#约束重原子，从而使得骨架较初始不会移动太多
relax:ramp_constraints
→#设为false则不进行倾斜约束（进行整体约束该选项需要设置为false）
use_input_sc             #turns on inclusion of the current
→rotamer for the packer
flip_HNQ
→#在氢键原子位置优化期间考虑翻转HIS, ASN, GLN
no_optH                  #是否在PDB加载期间进行氢原子位置优化
```

2. 局部对接

2.1 局部对接

```
#!/bin/bash
#SBATCH --job-name=rosetta
```

(续下页)

(接上页)

```
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_protocol.mpi.linuxgccrelease -in:file:s input_files/col_
→complex.pdb -in:file:native input_files/1v74.pdb -nstruct 1 -partners A_B_
→-dock_pert 3 8 -ex1 -ex2aro -out:path:all output_files -out:suffix _local_
→dock
```

输入与参数说明

```
in:file:s      #输入数据
in:file:native #native file, 与该文件进行计算比较
nstruct       #请注意在进行实际数据分析时, 此处的值应当至少为 500
partners      #partners A_B意味着, 链B对接进入链A
dock_pert     #dock_pert 3_
→8意味着, 在开始单独的模拟之前随机的将配体 (链B) 进行一个3埃的平移和8度的旋转
out:path:all  #输出路径
out:suffix    #输出文件名后缀
```

2.2 对得到的对接结果进行局部优化

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_protocol.mpi.linuxgccrelease -in:file:s input_files/1v74.pdb_
→-nstruct 1 -docking_local_refine -use_input_sc -ex1 -ex2aro -
→out:file:fullatom -out:path:all output_files -out:suffix _local_refine
```

3. 全局对接

若没有蛋白结合位点的信息, 则使用全局对接。全局对接假设蛋白质为球型, 而更小的蛋白质配体围绕蛋白质受体。全局对接对小复合物相对较好 (残基数小于 450)

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=64c512g
#SBATCH -N 1
```

(续下页)

(接上页)

```
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_protocol.mpi.linuxgccrelease -in:file:s input_files/col_
→complex.pdb -in:file:native input_files/1v74.pdb -unboundrot input_files/
→col_complex.pdb -nstruct 1 -partners A_B -dock_pert 3 8 -spin -randomize1_
→-randomize2 -ex1 -ex2aro -out:path:all output_files -out:suffix _global_
→dock
```

输入与参数说明

```
unboundrot  #将指定结构的旋转异构体添加到旋转异构体库中
nstruct     #请注意在进行实际数据分析时，此处的值应当为 10,000~100,000
```

4. Flexible Protein 对接

Rosetta 假设蛋白骨架为柔性的进行对接。**Rosetta** 假设蛋白-蛋白结合过程前后构象发生了较大的变化，并对蛋白构象簇 (ensembles) 进行对接，而非一个配体构象和一个受体构象。

4.1 prepack

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
ls input_files/COL_D_ensemble/*.pdb > COL_D_ensemblelist
ls input_files/IMM_D_ensemble/*.pdb > IMM_D_ensemblelist
mpirun docking_prepack_protocol.mpi.linuxgccrelease -in:file:s input_files/
→col_complex.pdb -in:file:native input_files/1v74.pdb -unboundrot input_
→files/col_complex.pdb -nstruct 1 -partners A_B -ensemble1 COL_D_
→ensemblelist -ensemble2 IMM_D_ensemblelist -ex1 -ex2aro -out:path:all_
→output_files -out:suffix _ensemble_dock
```

4.2 柔性对接

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=64c512g
```

(续下页)

(接上页)

```
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_prepack_protocol.mpi.linuxgccrelease -in:file:s input_files/
→col_complex.pdb -in:file:native input_files/1v74.pdb -unboundrot input_
→files/col_complex.pdb -nstruct 1 -partners A_B -dock_pert 3 8 -ensemble1_
→COL_D_ensemblelist -ensemble2 IMM_D_ensemblelist -ex1 -ex2aro -
→out:path:all output_files -out:suffix _ensemble_dock
```

二. π2.0 Rosetta

申请计算节点并导入 rosetta 软件

1. 对输入结构进行预处理 (refine) _π2.0_

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun relax.mpi.linuxgccrelease -in:file:s input_files/1qys.pdb -nstruct 2_
→-relax:constrain_relax_to_start_coords -relax:ramp_constraints false -ex1_
→-ex2 -use_input_sc -flip_HNQ -no_optH false
```

输入与参数说明

in:file:s	#输入数据
nstruct	
→#nstruct	可以提高模型结果的质量, 如nstruct 10将会获得10个模型
relax:constrain_relax_to_start_coords	
→#约束重原子, 从而使得骨架较初始不会移动太多	
relax:ramp_constraints	
→#设为false则不进行倾斜约束 (进行整体约束该选项需要设置为false)	
use_input_sc	#turns on inclusion of the current_
→rotamer for the packer	
flip_HNQ	
→#在氢键原子位置优化期间考虑翻转HIS, ASN, GLN	
no_optH	#是否在PDB加载期间进行氢原子位置优化

2. 局部对接 _π2.0_

2.1 局部对接 _π2.0_

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_protocol.mpi.linuxgccrelease -in:file:s input_files/col_
↪complex.pdb -in:file:native input_files/1v74.pdb -nstruct 1 -partners A_B_
↪-dock_pert 3 8 -ex1 -ex2aro -out:path:all output_files -out:suffix _local_
↪dock
```

输入与参数说明

```
in:file:s      #输入数据
in:file:native #native file, 与该文件进行计算比较
nstruct       #请注意在进行实际数据分析时, 此处的值应当至少为500
partners      #partners A_B意味着, 链B对接进入链A
dock_pert     #dock_pert 3_
↪8意味着, 在开始单独的模拟之前随机的将配体 (链B) 进行一个3埃的平移和8度的旋转
out:path:all  #输出路径
out:suffix    #输出文件名后缀
```

2.2 对得到的对接结果进行局部优化 _π2.0_

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_protocol.mpi.linuxgccrelease -in:file:s input_files/1v74.pdb_
↪-nstruct 1 -docking_local_refine -use_input_sc -ex1 -ex2aro -
↪out:file:fullatom -out:path:all output_files -out:suffix _local_refine
```

3. 全局对接 _π2.0_

若没有蛋白结合位点的信息, 则使用全局对接。全局对接假设蛋白质为球型, 而更小的蛋白质配体围绕蛋白质受体。全局对接对小复合物相对较好 (残基数小于 450)

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_protocol.mpi.linuxgccrelease -in:file:s input_files/col_
↪complex.pdb -in:file:native input_files/1v74.pdb -unboundrot input_files/
↪col_complex.pdb -nstruct 1 -partners A_B -dock_pert 3 8 -spin -randomize1_
↪-randomize2 -ex1 -ex2aro -out:path:all output_files -out:suffix _global_
↪dock
```

输入与参数说明

```
unboundrot  #将指定结构的旋转异构体添加到旋转异构体库中
nstruct      #请注意在进行实际数据分析时，此处的值应当为 10,000~100,000
```

4. Flexible Protein 对接 _π2.0_

Rosetta 假设蛋白骨架为柔性的进行对接。**Rosetta** 假设蛋白-蛋白结合过程前后构象发生了较大的变化，并对蛋白构象簇 (ensembles) 进行对接，而非一个配体构象和一个受体构象。

4.1 prepack _π2.0_

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
ls input_files/COL_D_ensemble/*.pdb > COL_D_ensemblelist
ls input_files/IMM_D_ensemble/*.pdb > IMM_D_ensemblelist
mpirun docking_prepack_protocol.mpi.linuxgccrelease -in:file:s input_files/
↪col_complex.pdb -in:file:native input_files/1v74.pdb -unboundrot input_
↪files/col_complex.pdb -nstruct 1 -partners A_B -ensemble1 COL_D_
↪ensemblelist -ensemble2 IMM_D_ensemblelist -ex1 -ex2aro -out:path:all_
↪output_files -out:suffix _ensemble_dock
```

4.2 柔性对接 $\pi 2.0$

```
#!/bin/bash
#SBATCH --job-name=rosetta
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load rosetta/3.12
mpirun docking_prepack_protocol.mpi.linuxgccrelease -in:file:s input_files/
↪col_complex.pdb -in:file:native input_files/1v74.pdb -unboundrot input_
↪files/col_complex.pdb -nstruct 1 -partners A_B -dock_pert 3 8 -ensemble1
↪COL_D_ensemblelist -ensemble2 IMM_D_ensemblelist -ex1 -ex2aro -
↪out:path:all output_files -out:suffix _ensemble_dock
```

运行结果

思源一号上的运行结果

```
output_files/
├─ 1v74_local_refine_0001.pdb
├─ col_complex_ensemble_dock_0001.pdb
├─ col_complex_global_dock_0001.pdb
├─ col_complex_local_dock_0001.pdb
├─ score_ensemble_dock.sc
├─ score_global_dock.sc
├─ score_local_dock.sc
└─ score_local_refine.fasc
```

$\pi 2.0$ 上的运行结果

```
output_files/
├─ 1v74_local_refine_0001.pdb
├─ col_complex_ensemble_dock_0001.pdb
├─ col_complex_global_dock_0001.pdb
├─ col_complex_local_dock_0001.pdb
├─ score_ensemble_dock.sc
├─ score_global_dock.sc
├─ score_local_dock.sc
└─ score_local_refine.f
```

参考资料

- Rosetta: <https://www.rosettacommons.org/>

Hic_breakfinder

简介

a framework that integrates optical mapping, high-throughput chromosome conformation capture (Hi-C), and whole genome sequencing to systematically detect SVs in a variety of normal or cancer samples and cell lines.

可用的版本

版本	平台	构建方式	模块名
1.0		源码编译	hic_breakfinder/1.0-gcc-9.2.0

数据下载

<https://salkinstitute.box.com/s/m8oyv2ypf8o3kcdsybzcmrpg032xnrgx>

运行示例

官方提供的数据下载比较慢，作业脚本中 `${Example}` 存放了示例数据。

```
#!/bin/bash
#SBATCH --job-name=hic_breakfinder
#SBATCH --partition=small
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=5

module load hic_breakfinder/1.0-gcc-9.2.0
hic_breakfinder --bam-file ${Example}/K562_in_house_b38d5.nodup.bam --exp-
→file-inter ${Example}/inter_expect_1Mb.hg38.txt --exp-file-intra $
→${Example}/intra_expect_100kb.hg38.txt --min-1kb --name Out
```

使用如下脚本提交作业

```
sbatch test.slurm
```

运行结果见 `Out.breaks.txt`

运行资源情况

```
Cluster: sjtupi
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 5
```

(续下页)

(接上页)

```
CPU Utilized: 07:57:02
CPU Efficiency: 19.97% of 1-15:48:35 core-walltime
Job Wall-clock time: 07:57:43
Memory Utilized: 4.68 GB
Memory Efficiency: 23.38% of 20.00 GB
```

小技巧

Hic_breakfinder 是单线程的程序，不能有效利用多核资源，建议大家根据内存占用情况来申请资源。

参考资料

- [hic_breakfinder](#)

STAR

简介

STAR 是生信领域常用的基因组对比软件，映射速度极快，准确率较高，在 RNA-seq 领域具有广泛的应用。

可用的版本

版本	平台	构建方式	模块名
2.7.6a3		spack	star/2.7.6a-gcc-11.2.0 思源一号
2.7.6a3		spack	star/2.7.6a-gcc-9.2.0

算例下载

思源一号：

```
mkdir ~/star && cd ~/star
cp -r /dssg/share/sample/star/* ./
gzip -d Homo_sapiens.GRCh38.dna.chromosome.2.fa.gz
gzip -d Homo_sapiens.GRCh38.86.chr.gtf.gz
```

π2.0：

```
mkdir ~/star && cd ~/star
cp -r /lustre/share/sample/star/* ./
gzip -d Homo_sapiens.GRCh38.dna.chromosome.2.fa.gz
gzip -d Homo_sapiens.GRCh38.86.chr.gtf.gz
```

数据目录如下所示：

```
[hpc@node522 ~]$ tree star/
star/
├── Homo_sapiens.GRCh38.86.chr.gtf.gz
├── Homo_sapiens.GRCh38.dna.chromosome.2.fa.gz
├── TG_r1.fastq.gz
└── TG_r2.fastq.gz
```

0 directories, 4 files

集群上的 **STAR**

- 一. 思源一号 *STAR*
- 二. $\pi 2.0$ *STAR*

一. 思源一号 **STAR**

使用流程如下所示

1. 创建基因组索引

先创建目录

```
mkdir chr1_index
```

脚本如下

```
#!/bin/bash
#SBATCH --job-name=star
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/11.2.0
module load star/2.7.6a-gcc-11.2.0
STAR --runThreadN 64 --genomeSAindexNbases 12 --runMode genomeGenerate --
↪genomeDir chr1_index --genomeFastaFiles Homo_sapiens.GRCh38.dna.
↪chromosome.2.fa --sjdbGTFfile Homo_sapiens.GRCh38.86.chr.gtf --
↪sjdbOverhang 99
```

2. 比对基因组

脚本如下

```
#!/bin/bash
#SBATCH --job-name=star
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/11.2.0
module load star/2.7.6a-gcc-11.2.0
STAR --runMode alignReads --outSAMtype BAM Unsorted --readFilesCommand zcat.
↪--genomeDir chr1_index/ --outFileNamePrefix Homo_sapiens.GRCh38 --
↪readFilesIn TG_r1.fastq.gz TG_r2.fastq.gz
```

二. π2.0 STAR

使用流程如下所示

1. 创建基因组索引 π2.0

先创建目录

```
mkdir chr1_index
```

脚本如下

```
#!/bin/bash
#SBATCH --job-name=star
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/9.2.0
module load star/2.7.6a-gcc-9.2.0
STAR --runThreadN 40 --genomeSAindexNbases 12 --runMode genomeGenerate --
↪genomeDir chr1_index --genomeFastaFiles Homo_sapiens.GRCh38.dna.
↪chromosome.2.fa --sjdbGTFfile Homo_sapiens.GRCh38.86.chr.gtf --
↪sjdbOverhang 99
```

2. 比对基因组 π2.0

脚本如下

```
#!/bin/bash
#SBATCH --job-name=star
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gcc/9.2.0
module load star/2.7.6a-gcc-9.2.0
STAR --runMode alignReads --outSAMtype BAM Unsorted --readFilesCommand zcat.
→--genomeDir chr1_index/ --outFileNamePrefix Homo_sapiens.GRCh38 --
→readFilesIn TG_r1.fastq.gz TG_r2.fastq.gz
```

运行结果如下所示

1.STAR 思源一号

对比基因组完成后，会生成以下文件及目录

```
[hpchgc@node522 ~]$ tree star/
star/
├── 140803.err
├── 140803.out
├── chr1_index
│   ├── chrLength.txt
│   ├── chrNameLength.txt
│   ├── chrName.txt
│   ├── chrStart.txt
│   ├── exonGeTrInfo.tab
│   ├── exonInfo.tab
│   ├── geneInfo.tab
│   ├── Genome
│   ├── genomeParameters.txt
│   ├── Log.out
│   ├── SA
│   ├── SAindex
│   ├── sjdbInfo.txt
│   ├── sjdbList.fromGTF.out.tab
│   ├── sjdbList.out.tab
│   └── transcriptInfo.tab
├── Homo_sapiens.GRCh38.86.chr.gtf
├── Homo_sapiens.GRCh38Aligned.out.bam
├── Homo_sapiens.GRCh38.dna.chromosome.2.fa
├── Homo_sapiens.GRCh38Log.final.out
├── Homo_sapiens.GRCh38Log.out
├── Homo_sapiens.GRCh38Log.progress.out
└── Homo_sapiens.GRCh38SJ.out.tab
```

(续下页)

(接上页)

```

├─ run.slurm
├─ TG_r1.fastq.gz
└─ TG_r2.fastq.gz

```

2.STAR π2.0

对比基因组完成后，会生成以下文件及目录

```

[hpc@cas013 data]$ tree star
star
├─ chr1_index
│   ├── chrLength.txt
│   ├── chrNameLength.txt
│   ├── chrName.txt
│   ├── chrStart.txt
│   ├── exonGeTrInfo.tab
│   ├── exonInfo.tab
│   ├── geneInfo.tab
│   ├── Genome
│   ├── genomeParameters.txt
│   ├── Log.out
│   ├── SA
│   ├── SAindex
│   ├── sjdbInfo.txt
│   ├── sjdbList.fromGTF.out.tab
│   ├── sjdbList.out.tab
│   └─ transcriptInfo.tab
├─ Homo_sapiens.GRCh38.86.chr.gtf
├─ Homo_sapiens.GRCh38Aligned.out.bam
├─ Homo_sapiens.GRCh38.dna.chromosome.2.fa
├─ Homo_sapiens.GRCh38Log.final.out
├─ Homo_sapiens.GRCh38Log.out
├─ Homo_sapiens.GRCh38Log.progress.out
├─ Homo_sapiens.GRCh38SJ.out.tab
├─ TG_r1.fastq.gz
└─ TG_r2.fastq.gz

```

参考资料

- STAR 官方网站 <https://github.com/alexdobin/STAR/>

EasyFuse

简介

EasyFuse 是预测临床肿瘤样本特异性基因融合的新工具，包含了 STAR-Fusion、InFusion、MapSplice2、Fusioncatcher 和 SoapFuse 多种融合基因预测软件。

可用的版本

版本	平台	构建方式	路径
1.3.6		容器	/dssg/share/imgs/easyfuse/easyfuse_1.3.6.sif 思源一号
1.3.5		容器	/dssg/share/imgs/easyfuse/easyfuse_1.3.5.sif 思源一号
1.3.6		容器	/lustre/share/img/x86/easyfuse/easyfuse_1.3.6.sif
1.3.5		容器	/lustre/share/img/x86/easyfuse/easyfuse_1.3.5.sif

运行示例

思源一号集群 **EasyFuse**

输入的 /dssg/share/sample/easyfuse 为 demo 数据。

```
#!/bin/bash
#SBATCH --job-name=esayFuse_demo
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity exec -e -B /dssg/share/data/easyfuse_ref:/ref \
/dssg/share/imgs/easyfuse/easyfuse_1.3.5.sif \
python /code/easyfuse/processing.py \
-i /dssg/share/sample/easyfuse \
-o output
```

使用如下脚本提交作业

```
sbatch test.slurm
```

 π 集群 **EasyFuse**

输入的 /lustre/share/samples/easyfuse 为 demo 数据。

EasyFuse@1.3.6

```
#!/bin/bash
#SBATCH --job-name=esayFuse_demo
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
```

(续下页)

(接上页)

```
#SBATCH --error=%j.err

mkdir -p output

singularity exec \
--containall \
-B /lustre/share/samples/easyfuse_ref:/ref \
-B /lustre/share/samples/easyfuse:/data \
-B $PWD/output:/output \
/lustre/share/img/x86/easyfuse/easyfuse_1.3.6.sif \
python /code/easyfuse/processing.py -i /data/ -o /output
```

EasyFuse@1.3.5

```
#!/bin/bash
#SBATCH --job-name=esayFuse_demo
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity exec -e -B /lustre/share/samples/easyfuse_ref:/ref \
/lustre/share/img/x86/easyfuse/easyfuse_1.3.5.sif \
python /code/easyfuse/processing.py \
-i /lustre/share/samples/easyfuse \
-o output
```

使用如下脚本提交作业

```
sbatch test.slurm
```

输出结果

```
.
├─ config.ini
├─ easyfuse_processing.log
├─ FusionSummary
│   ├── SRR1659960_05pc_fusRank_1.csv
│   ├── SRR1659960_05pc_fusRank_1.pred.all.csv
│   └─ SRR1659960_05pc_fusRank_1.pred.csv
├─ process.sh
├─ samples.db
└─ Sample_SRR1659960_05pc
    ├── expression
    ├── fetchdata
    └─ filtered_reads
```

(续下页)

```
|— fusion
|— qc
```

参考资料

- [EasyFuse](#)

MetaWRAP

简介

MetaWRAP 这是一套强大的宏基因组分析流程，专注于宏基因组 **Binning**，能实现原始序列的质控、物种注释和可视化、宏基因组拼接、三种主流 **Bin** 方法分析和结果筛选与可视化、**Bin** 的重新组装、**Bin** 的物种和功能注释等。轻松实现 **Bin** 相关分析和可视化的绝大部分需求。

可通过 **conda** 自行安装。

安装代码

```
srunk -p 64c512g -n 4 --pty /bin/bash
mkdir ${HOME}/01.application/11.metaWRAP && cd ${HOME}/01.application/11.
    ↳ metaWRAP
git clone https://github.com/bxlab/metaWRAP.git
mkdir database
cd metaWRAP
sed -i 's#/scratch/gu#${HOME}/01.application/11.metaWRAP/database#g' bin/
    ↳ config-metawrap
export PATH=${HOME}/01.application/11.metaWRAP/metaWRAP/bin/:$PATH
module load miniconda3
conda create -n env4mamba mamba -y
source activate env4mamba
mamba create -n env4metawrap -c conda-forge python=2.7 -y
source activate env4metawrap
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
conda config --add channels ursky
${HOME}/.conda/envs/env4mamba/bin/mamba install --only-deps -c ursky_
    ↳ metawrap-mg -y
```

参考资料

- [MetaWRAP](#)

GATE

简介

GATE 是 OpenGATE 协作开发的开源软件，致力于医学成像和放射治疗的数值模拟。目前该软件支持模拟发射断层扫描（正电子发射断层扫描 - PET 和单光子发射计算机断层扫描 - SPECT）、计算机断层扫描（CT）、光学成像（生物发光和荧光）和放射治疗实验。使用易于学习的宏观机制来配置简单或高度复杂的实验设置，GATE 现在在新的医学成像设备的设计、采集协议的优化和图像重建算法的开发和评估中起着关键作用。

可用版本

版本	平台	构建方式	模块名
9.2		容器	gate/9.2-gcc-8.5.0-singularity 思源一号
9.2		容器	gate/9.2-gcc-8.5.0-singularity pi2.0

算例

```
思源一号：
/dssg/share/sample/gate

pi2.0:
/lustre/share/samples/gate
```

集群上的 GATE

- 思源一号 GATE
- π 2.0 GATE

一. 思源一号 GATE

思源上拷贝数据至本地目录

```
cd
mkdir ~/gate
cp -r /dssg/share/sample/gate/* ./
```

在文件 `run.sh` 中更改具体的执行内容

```
$cat run.sh
```

(续下页)

(接上页)

```
#!/bin/bash
/runGate.sh "-a [nb,1000] mac/main.mac"
```

提交运行脚本

```
#!/bin/bash
#SBATCH --job-name=gate_test
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load gate
Gate run.sh
```

二. π 2.0 GATE

π 2.0 上拷贝数据至本地目录

```
cd
mkdir ~/gate
cp -r /dssg/share/sample/gate/* ./
```

更改文件 `run.sh` 中具体的执行内容

```
$cat run.sh

#!/bin/bash
/runGate.sh "-a [nb,1000] mac/main.mac"
```

π 2.0 上提交运行脚本

```
#!/bin/bash
#SBATCH --job-name=gate_test
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=2
#SBATCH --output=%j.out
#SBATCH --error=%j.error

module load gate
Gate run.sh
```

执行结果

思源一号上 **GATE** 的运行结果

```
tree output/
output/
├── BeamLineEntrance.root
├── BeamLineExit.root
├── BeamLineMiddle.root
├── GlobalBoxEntrance.root
├── IDD-proton-Dose-Squared.txt
├── IDD-proton-Dose.txt
├── IDD-proton-Dose-Uncertainty.txt
├── IDD-proton-Edep.txt
└── stat-proton.txt

0 directories, 9 files
```

pi2.0 上 **GATE** 的运行脚本

```
tree output/
output/
├── BeamLineEntrance.root
├── BeamLineExit.root
├── BeamLineMiddle.root
├── GlobalBoxEntrance.root
├── IDD-proton-Dose-Squared.txt
├── IDD-proton-Dose.txt
├── IDD-proton-Dose-Uncertainty.txt
├── IDD-proton-Edep.txt
└── stat-proton.txt

0 directories, 9 files
```

参考链接: <https://opengate.readthedocs.io/en/latest/introduction.html>

pangenie

PanGenie 是一种输入数据为短读型的基因分型器，通过考虑已知单倍型的信息（通过图中的路径表示）可有效计算泛基因组图中以气泡表示的变异的基因型。

可用版本

平台	版本	构建方式	路径
思源一号	v2.0.0	容器	/dssg/share/imgs/pangenie/pangenie_v2.0.0.sif
pi2.0	v2.0.0	容器	/lustre/share/img/x86/pangenie/pangenie_v2.0.0.sif
ARM	v2.0.0	容器	/ lustre/ share/ img/ arm/ pangenie/ pange- nie_v2.0.0.sif

算例路径

平台	路径
思源一号	/dssg/share/sample/pangenie
pi 2.0	/lustre/share/samples/pangenie
ARM	/lustre/share/samples/pangenie

使用方法

- 思源一号 *PanGenie*
- $\pi 2.0$ *PanGenie*
- *ARM PanGenie*

思源一号 **PanGenie**

首先拷贝数据到本地

```
cd
mkdir pangenie
cp -r /dssg/share/sample/pangenie/* ./
```

然后在数据目录下提交如下脚本

```
#!/bin/bash
#SBATCH --job-name=pangenie_sy
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

IMAGE=/dssg/share/imgs/pangenie/pangenie_v2.0.0.sif
singularity exec $IMAGE PanGenie -i test-reads.fa -r test-reference.fa -v \
    test-variants.vcf -o test -e 100000
```

$\pi 2.0$ **PanGenie**

首先，拷贝数据到本地

```
cd
mkdir pangenie
cp -r /lustre/share/samples/pangenie/* ./
```


然后，在数据目录下提交如下脚本

```
#!/bin/bash
#SBATCH --job-name=pangenie_x86
#SBATCH --partition=small
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

IMAGE=/lustre/share/img/x86/pangenie/pangenie_v2.0.0.sif
singularity exec $IMAGE PanGenie -i test-reads.fa -r test-reference.fa -v \
↳test-variants.vcf -o test -e 100000
```

ARM PanGenie

首先将数据拷贝到本地

```
cd
mkdir pangenie
cp -r /lustre/share/samples/pangenie/* ./
```

然后提交如下作业脚本

```
#!/bin/bash
#SBATCH --job-name=pangenie_arm
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

IMAGE=/lustre/share/img/arm/pangenie/pangenie_v2.0.0.sif
singularity exec $IMAGE PanGenie -i test-reads.fa -r test-reference.fa -v \
↳test-variants.vcf -o test -e 100000
```

PanGenie 运行结果

PanGenie-思源一号

```
##### Summary #####
time spent reading input files: 0.532767 sec
time spent counting kmers:      0.116464 sec
time spent selecting paths:     0.000311676 sec
time spent determining unique kmers: 0.00101096 sec
time spent genotyping chromosome chr1: 0.0527584
total running time:            0.703721 sec
```

(续下页)

(接上页)

```
total wallclock time: 0.702943 sec
Total maximum memory usage: 0.024004 GB
```

PanGenie-pi 2.0

```
##### Summary #####
time spent reading input files: 0.0779914 sec
time spent counting kmers:      0.107329 sec
time spent selecting paths:     0.00195029 sec
time spent determining unique kmers: 0.0354376 sec
time spent genotyping chromosome chr1: 0.0532523
total running time:            0.276735 sec
total wallclock time: 0.274615 sec
Total maximum memory usage: 0.015336 GB
```

PanGenie-ARM

```
##### Summary #####
time spent reading input files: 0.00858966 sec
time spent counting kmers:      0.0387264 sec
time spent selecting paths:     0.0010025 sec
time spent determining unique kmers: 0.00505522 sec
time spent genotyping chromosome chr1: 0.215358
total running time:            0.270459 sec
total wallclock time: 0.268968 sec
Total maximum memory usage: 0.0128 GB
```

参考资料

- PanGenie GitHub <https://github.com/eblerjana/pangenie>

Clustalo

简介

Clustalo 即 Clustal-Omega，是一种用于蛋白质和 DNA/RNA 的通用多序列比对 (MSA) 程序。它可以生成高质量的 MSA，并能够在合理的时间内处理数十万个序列的数据集。

在默认模式下，用户提供要对齐的序列文件，这些序列被聚类以生成一个指南树，这用于指导序列的“渐进式对齐”。此外，还具有将现有比对相互对齐、将序列与比对以及使用隐马尔可夫模型 (HMM) 来帮助指导与用于制作 HMM 的序列同源的新序列的比对的工具。后一种过程称为“外部轮廓对齐”或 EPA。

Clustal-Omega 使用 HMM 作为对齐引擎，基于 Johannes Soeding 的 HAlign 包。指南树是使用 mBed 的增强版本制作的，它可以在 $O(N \cdot \log(N))$ 时间内聚类大量序列。然后，按照参考树给出的聚类，使用 HAlign 对齐越来越大的对齐方式进行多重对齐。

目前形式的 Clustal-Omega 已经过广泛的蛋白质序列测试，自 1.1.0 版起添加了 DNA/RNA 支持。

可用的版本

版本	平台	构建方式	模块名
1.2.4	cpu	precompile	clustalo/1.2.4-intel-2021.4.0 思源一号
1.2.4	cpu	precompile	clustalo/1.2.4-intel-2021.4.0

Clustalo 预编译文件安装步骤

官网下载预编译文件

```
$ wget http://www.clustal.org/omega/clustalo-1.2.4-Ubuntu-x86_64
```

重命名后添加可执行权限

```
$ mv clustalo-1.2.4-Ubuntu-x86_64 clustalo
$ chmod u+x clustalo
```

添加 Clustalo 的环境变量

```
$ export PATH=path/to/clustalo:$PATH
```

检验安装，以下命令查看 clustalo 版本信息

```
$ clustalo --version
```

Clustalo 编译安装步骤

安装 clustalo 之前，要先安装 argtable2-13 作为其编译的依赖。

```
$ wget https://launchpad.net/ubuntu/+archive/primary/+sourcefiles/argtable2/
→13-1.1/argtable2_13.orig.tar.gz
$ tar -zxvf argtable2_13.orig.tar
$ cd argtable2-13/
$ ./configure --prefix=path/to/argtable2-13
$ make check
$ make && make install
```

添加 argtable2-13 的环境变量

```
$ export PATH=path/to/argtable2-13:$PATH
$ export LD_LIBRARY_PATH=path/to/argtable2-13/lib:$LD_LIBRARY_PATH
```

编译安装 clustalo

```
$ clustalo --version
$ wget https://launchpad.net/ubuntu/+source/clustalo/1.2.4/clustalo_1.2.4.
  ↳orig.tar
$ tar -zxvf clustalo_1.2.4.orig.tar
$ cd clustal-omega-1.2.4
```

申请节点进行编译

```
$ srun -p small -n 4 --pty /bin/bash # Pi2.0
$ srun -p 64c512g -n 4 --pty /bin/bash # 思源一号
```

调用 intel-oneapi 编译器

```
$ module load oneapi/2021.4.0
```

开始编译

```
$ ./configure CFLAGS='-I/path/to/argtable2-13/include' LDFLAGS='-L/path/to/
  ↳argtable2-13/lib' --prefix=/path/to/clustalo
$ make check
$ make && make install
```

添加 Clustalo 的环境变量

```
$ export PATH=/path/to/clustalo/bin:$PATH
$ export PATH=/path/to/clustalo/include:$PATH
$ export LD_LIBRARY_PATH=/path/to/clustalo/lib:$LD_LIBRARY_PATH
```

检验安装，以下命令查看 clustalo 版本信息

```
$ clustalo --version
```

Clustalo 运行示例

将以下内容保存为 globin.fa

```
>P01013 GENE X PROTEIN (OVALBUMIN-RELATED)
QIKDLLVSSSTDLDTTLVLVNAIYFKGMWKTAFNAEDTREMPPHVTQESKPVQMMCMNNSFNVATLPAE
KMKILELPFASGDLMLVLLPDEVSDLERIEKTINFEKLTWETNPNTMEKRRVKVYLPQMKIEEKYNLTS
VLMALGMTDLFIPSANLTGISSAESLKISQAVHGAFMELSEDGIEMAGSTGVIEDIKHSPESQFRADHP
FLFLIKHNPTNTIVYFGRYWSP

>NP_689511.2 nuclear autoantigenic sperm protein isoform 3 [Homo sapiens]
MAMESTATAAVALVADKIEDVPAPSTSADKVESLDVDSEAKKLLGLGQKHLVMGDIPAAVNAFQEAAS
LLGKKYGETANECGEAFFYGKSLLELARMENGLVNALEGVHVEEEEGEKTEDESIVENNDNIDETEGSE
EDDKENDKTEEMPNDVLENKSLQENEEEEIGNLELAWDMLDLAKIIFKRQETKEAQLYAAQAHKLGEVS
VESENYOAVEEFQSCNLNQEQLAHDRLLAETHYQLGLAYGYNSQYDEAVAQFSKSIEVIENRMAVLNE
QVKEAEGSSAEYKKEIEELKELLPEIREKIEDAKESQRSGNVAELALKATLVESSTSGFTPGGGGSSVSMI
ASRKPTDGASSNCVTDISHLVRRKRKPEEESPRKDDAKKAKQEPEVNGGSGDAVPSGNEVSENMEEEAEN
QAESRAAVEGTVEAGATVESTAC
```

module 调用 clustalo

```
$ module load clustalo/1.2.4-intel-2021.4.0
```

Clustal-Omega 读取序列文件 `globin.fa`，对齐序列，并将结果以 `fasta/a2m` 格式打印到屏幕上。

```
$ clustalo -i globin.fa
```

运行结果

```
>P01013 GENE X PROTEIN (OVALBUMIN-RELATED)
-----QIKDLLVSSS-----TDLD-----
-----TTLV-----LVNAIYFKGM-----WKTAF-----
-----NAEDTREMPFHVTKQESKPVQMMCMNNSFNVATL
PAEKMKILELPPFASGDL SMLVLLPDEVS DL-----ERIEK
TINFE-----K-----LTEWTNPNT-----
-----MEKRRVKVYLPQMKIEEKYNLTSVLMALGMTDLFIP SANLTGISSAESLKI
SQAVHGAFMELSEDGIEMAGS-----TGVIEDIKH-----SPESEQ
FRADHPFLFLIKHNPT----NTIVYFGRYWSP-----
----
>NP_689511.2 nuclear autoantigenic sperm protein isoform 3 [Homo sapiens]
MAMESTATAA VAAELVSADKIEDVPAPSTSADKVESLDVDSEAKLLGLGQKHLVMGDIP
AAVNAFQEAASLLGKKYGETANECGEAFFFYGKS LLELARMENGVLGNALEGVHVEEEEG
EKTEDESLVENNDNIDETEGSEEDDKENDKTEEMPND-----SVLENKSL--QEN
EEEEIGNLELAWDMLDLAKIIFKRQETKEAQLYAAQAHLKLGEVSVESENYYOAVEEFQS
CLNLQEQYLEAHDRLLAETHYQLGLAYGYNSQYDEAVAQFSKSIEVIENRMAVLNEQVKE
AEGSSAEYKKEIEELKELLPEIREKIEDAKE--SQ-----RSGNVA-----
ELALKATLVESSTSGFTPGGGGSSVSMIASRKPTDGASSNCVTDISHLVRKKRKPEEES
PRKDDAKK--AKQEPEVNGGSGDAVPSGNEVSENMEEEAENQAESRAAVEGTVEAGATVE
STAC
```

参考资料

- [Clustal Omega](#)

Space Ranger

简介

Visium 空间基因表达是一种先进的分子分析方法，可以通过总 mRNA 来分类组织。Space Ranger 则是一组分析工具，用于处理 Visium 空间基因表达数据，这些数据是通过明场和荧光显微镜图像得到的。使用 Space Ranger，用户能够在甲醛固定石蜡包埋 (FFPE) 和新鲜冰冻组织中绘制整个转录组，从而深入了解正常发育、疾病病理学以及临床翻译研究的新颖观点。

可用版本

版本	平台	构建方式	模块名
2.1.1		precompile	spaceranger/2.1.1 Pi2.0

安装命令

前往 [Space Ranger](#) 下载页面填写 10x Genomics 用户软件许可协议信息，获得下载链接

```
tar -zxvf spaceranger-2.0.0.tar.gz
export PATH=/PATH/TO/spaceranger-2.1.1:$PATH
```

`spaceranger mkfastq` 命令需要使用软件 `bcl2fastq`，版本需要在 2.20 以上，可以通过 `conda` 在本地自行安装与调用：

```
module load miniconda3
conda create -n bcl2fastq python=3.8
conda install -c freenome bcl2fastq=2.20
```

任务脚本

```
#!/bin/bash
#SBATCH --job-name=spacerange
#SBATCH --partition=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load spaceranger/2.1.1

ulimit -s unlimited
ulimit -l unlimited

spaceranger count --sample mysample --fastqs data.fastq --disable-ui
```

参考资料

- [Space Ranger](#)
- [Space Ranger Tutorials](#)

vcflib

简介

The Variant Call Format (VCF) 是一种平面文件、制表符分隔的文本格式，用于描述个体之间的引用索引变体。VCF 提供了一种用于描述个体和样本群体变异的通用交换格式，并且已成为各种基因组变异检测器事实上的标准报告格式。

安装方式

可以使用 miniconda3 进行安装，以 Pi2.0 为例:

```
sruncpu -p 8 --pty /bin/bash
module load miniconda3/22.11.1
conda create -n vcflib python=3.8.16
conda activate vcflib
conda install -c bioconda vcflib
```

参考资料

- [vcflib](#)

AlphaFold3

AlphaFold3 是由谷歌 DeepMind 和 Isomorphic Labs 团队开发的人工智能程序，于 2024 年 5 月发布，11 月开源供学术用途。它能准确预测蛋白质、DNA、RNA 等多种生物分子及其复合体的结构和相互作用，预测精度较 AlphaFold2 大幅提升，在药物研发等领域应用广泛。其架构引入 Pairformer 和扩散模块，采用跨蒸馏技术训练，减少了多重序列比对处理量，提高了计算效率和泛化能力，为生命科学研究提供了更强大的工具。

可用的版本

硬件	平台	版本
A100-40GB	思源一号	v3.0.1
A800-80GB	思源一号	v3.0.1
V100-32GB	Pi2.0(待更新)	v3.0.0

版本区别

1. 思源一号的 A800-80GB 镜像为原镜像，未做修改，AlphaFold3 可以在单张 A800-80GB 上运行，最大能够处理 5120 个 tokens。
2. 对于 A100-40GB，根据官方文档，我们对原镜像进行了以下修改：
 - 开启 unified memory, 启用统一内存允许程序在空间不足时将 GPU 内存溢出到主机内存, 这可以防止 OOM, 修改 /app/alphafold/run_alphafold.py 文件如下:

```
os.environ['XLA_PYTHON_CLIENT_PREALLOCATE']='false'
os.environ['XLA_CLIENT_MEM_FRACTION']='3.2'
os.environ['TF_FORCE_UNIFIED_MEMORY']='true'
```

- 调整 `/app/alphafold/src/alphafold3/model/model_config.py` 中 `pair_transition_shard_spec` 参数如下：

```
pair_transition_shard_spec: Sequence[_Shape2DType] = (
    (2048, None),
    (3072, 1024),
    (None, 512),
)
```

3. A100-40GB 上最大能够处理 4352 个 tokens 的输入，与 A800-80GB 相比，计算精度一致，但吞吐量更小。

使用前准备

- 新建运行文件夹，如 `alphafold`
- 在运行文件夹中创建输入文件夹 `input` 和输出文件夹 `output`
- 在 `input` 文件夹中放置输入 JSON 文件，例如 `fold_input.json`，自定义输入 JSON 文件可参考官方文档 <https://github.com/google-deepmind/alphafold3/blob/main/docs/input.md>

蛋白质结构预测 JSON 文件示例

```
{
  "name": "2PV7",
  "sequences": [
    {
      "protein": {
        "id": ["A", "B"],
        "sequence":
        ↪ "GMRESYANENQFGFKTINSDIHKIVIVGGYGKLGGLFARYLRASGYPI SILDREDWAVAESILANADVIVSVPIINLTLETIERLK
        ↪ "
      }
    }
  ],
  "modelSeeds": [1],
  "dialect": "alphafold3",
  "version": 1
}
```


在思源一号上运行 **AlphaFold3**

作业脚本 *af3.slurm* 示例如下:

A100-40GB

```
#!/bin/bash
#SBATCH --job-name=alphafold3
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity exec \
  --nv \
  --bind $PWD/input:/root/af_input \
  --bind $PWD/output:/root/af_output \
  --bind /dssg/share/data/alphafold3/models:/root/models \
  --bind /dssg/share/data/alphafold3/database:/root/public_databases \
  /dssg/share/imgs/ai/alphafold/alphafold3-a100.sif \
  /alphafold3_venv/bin/python /app/alphafold/run_alphafold.py \
  --json_path=/root/af_input/fold_input.json \
  --model_dir=/root/models \
  --db_dir=/root/public_databases \
  --output_dir=/root/af_output
```

A800-80GB

```
#!/bin/bash
#SBATCH --job-name=alphafold3
#SBATCH --partition=a800
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity exec \
  --nv \
  --bind $PWD/input:/root/af_input \
  --bind $PWD/output:/root/af_output \
  --bind /dssg/share/data/alphafold3/models:/root/models \
  --bind /dssg/share/data/alphafold3/database:/root/public_databases \
```

(续下页)

(接上页)

```
/dssg/share/imgs/ai/alphafold/alphafold3-a800.sif \
/alphafold3_venv/bin/python /app/alphafold/run_alphafold.py \
--json_path=/root/af_input/fold_input.json \
--model_dir=/root/models \
--db_dir=/root/public_databases \
--output_dir=/root/af_output
```

使用 `sbatch af3.slurm` 语句提交作业。

运行结束后，计算结果保存在 `alphafold/output/` 下，具体可参考 <https://github.com/google-deepmind/alphafold3/blob/main/docs/output.md>

分阶段运行 AlphaFold3

AlphaFold3 运行分为 *data_pipeline* 和 *inference* 两个阶段，*data_pipeline* 阶段主要利用 CPU 进行 MSA/模板搜索，*inference* 阶段主要利用 GPU 进行模型推理。

1. 将 *data_pipeline* 与 *inference* 分开，减少 GPU 机时浪费。
2. 输出结果的 json 中会保存缓存 MSA / 模板搜索的结果，可以用于跨种子或其他特征变体（例如配体）的多个不同推理。

以 A100 为例：

- *data_pipeline* 阶段脚本示例（64c512g 队列运行）：

```
#!/bin/bash
#SBATCH --job-name=alphafold3
#SBATCH --partition=64c512g
#SBATCH -N 1
#SBATCH --ntasks-per-node=32
#SBATCH --output=%j.out
#SBATCH --error=%j.err

/usr/bin/time -v singularity exec \
  --nv \
  --bind $PWD/input:/root/af_input \
  --bind $PWD/output:/root/af_output \
  --bind /dssg/share/data/alphafold3/models:/root/models \
  --bind /dssg/share/data/alphafold3/database:/root/public_databases \
  /dssg/share/imgs/ai/alphafold/alphafold3-a100.sif \
  /alphafold3_venv/bin/python /app/alphafold/run_alphafold.py \
  --norun_inference \
  --jackhmmer_n_cpu=$SLURM_NTASKS \
  --nhmmer_n_cpu=$SLURM_NTASKS \
  --input_dir=/root/af_input \
  --model_dir=/root/models \
  --db_dir=/root/public_databases \
  --output_dir=/root/af_output
```

alphafold/output/ 文件夹下会保存 *data_pipeline* 阶段缓存的结果，以该目录下的 json 文件作为 *inference* 阶段的输入。本示例中生成的结果文件为 *2PV7_data.json*。

- *inference* 阶段脚本示例 (a100 队列运行):

```
#!/bin/bash
#SBATCH --job-name=alphafold3
#SBATCH --partition=a100
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=16
#SBATCH --gres=gpu:1          # use 1 GPU
#SBATCH --output=%j.out
#SBATCH --error=%j.err

singularity exec \
  --nv \
  --bind $PWD/input:/root/af_input \
  --bind $PWD/output:/root/af_output \
  --bind /dssg/share/data/alphafold3/models:/root/models \
  --bind /dssg/share/data/alphafold3/database:/root/public_databases \
  /dssg/share/imgs/ai/alphafold/alphafold3-a800.sif \
  /alphafold3_venv/bin/python /app/alphafold/run_alphafold.py \
  --norun_data_pipeline \
  --json_path=/root/af_input/2PV7_data.json \
  --model_dir=/root/models \
  --db_dir=/root/public_databases \
  --output_dir=/root/af_output
```

data_pipeline 性能测试

思源平台 A100-40GB

版本	用时 (s)
v3.0.0	2475
v3.0.1	677

参考资料

- AlphaFold3 GitHub: <https://github.com/google-deepmind/alphafold3>

4.11 常见问题

- 0. π 2.0 集群名有什么含义?
- 1. 账号申请和登录
- 2. 作业提交和运行

3. 作业出错
4. 软件安装
5. 收费和充值
6. 邮件支持
7. 集群通知
8. 致谢模版
9. 医学院和附属医院申请 jAccount 账号
10. 如何重置.bashrc 和.bash_profile

4.11.1 0. π 2.0 集群名有什么含义?

- a) π 在希腊文里具有并行的涵义;
- b) π 是无限不循环的无理数, 是人类理解无限的开始;
- c) π 的精确计算是人类使用计算机解决科学问题的代表;
- d) π 可看作交通大学首字母 J 和 T 的组合。

4.11.2 1. 账号申请和登录

1.1 Q: 如何开通账号?

A: 在“交我办”(或 我的数字交大 my.sjtu.edu.cn) 中的“交我算”里申请。我们将会在两个工作日内开通账号。

1.2 Q: 是不是只有教职工才能申请账号? 学生要使用交我算 HPC+AI 平台该怎么办?

A: 是的。只有教职工(含博士后)才能申请主账号。主账号下面可以免费开通子账号。

主账号和子账号都为独立账号, 仅在计费关系上存在关联。若课题组有数据或软件共享需求, 可发邮件给我们, 我们将建立 `acct-XXX/share` 文件夹, 主账号和子账号均可在此文件夹下读写。

1.3 Q: 子账号的申请和费用?

A: 子账号也是在“交我办”(或 我的数字交大 my.sjtu.edu.cn) 中的“交我算”里申请。我们将会在两个工作日内开通账号。

可自定义子账号名, 格式为 `xxx-yyy`, 其中 `xxx` 为主账号, `yyy` 可以自定义, 比如 `user1`、`user2`。我们将在两个工作日内为您开通。每个主账号能免费申请子账号的个数不限, 请按需申请。

1.4 Q: 我为什么连不上集群?

A: 集群(除 ARM 平台)支持公网直接访问, 无需校园 VPN。若遇到连接问题, 请先检查网络, 或在其它设备或客户端上尝试。

- a) 请首先保证网络畅通;
- b) 查看交我算用户微信群, 是否有集群下线停机通知。集群通知会及时发布在用户微信群里。若需加入用户微信群, 请发邮件至 [hpc 邮箱](#);
- c) 集群登录节点设置了 fail2ban 服务, 多次输入密码错误后会被临时封禁 1 小时。如果您需要重置密码, 请使用或抄送账号负责人邮箱发送邮件到 [hpc 邮箱](#), 邮件中注明账号, 我们将会在 1 个工作日内响应您的申请;
- d) 如果您在登录节点运行计算密集的作业, 将会被程序自动查杀, 您的账号会被加入到黑名单, 并在 30-120 分钟内无法登录。

1.5 Q: 为什么连上了集群, 过一会儿又断了?

A: 请参考 SSH 下的 [登录常掉线的问题](#)章节进行设置。

4.11.3 2. 作业提交和运行

2.1 Q: 单个作业最长运行时间是多长?

A: 64c512g, a100, cpu, small, dgx2 和 arm128c256g 队列上的作业运行时间最长 7 天。huge 和 192c6t 运行时间最长 2 天。具体时间限制可以通过 `scontrol show partition` 命令查看 MaxTime 参量。

2.2 Q: 我的作业运行将超过最长时间限制, 有没有办法延长运行时限?

A: 有。请将用户名、作业号、预计运行时间等信息发送到[hpc 邮箱](#), 我们将会为您延长。需要注意的是: 延长的作业, 除因集群原因, 不作诊断或机时返还。并请尽量提前两天发送申请。

2.3 Q: 我的作业需要大内存怎么办?

A: 目前集群有 huge 和 192c6t 节点可以提供较大的内存。huge 节点每节点提供 3T 内存, 有 2 台。192c6t 节点有 6T 内存。具体使用请参考[作业示例](#)

4.11.4 3. 作业出错

3.1 Q: 为什么我的作业运行结果是作业运行结果是 `node_fail`, 该怎么处理?

A: `node_fail` 是提示由于计算节点故障导致作业运行失败。您重新提交作业即可。失败作业的机时系统会自动返还, 无须发邮件告诉我们。

3.2 Q: 为什么我在登录节点上的程序会被终止,我能否在登录节点运行程序?

A: 登录节点用于文件编辑、作业提交、小型应用编译、文件下载等轻量级工作。而科学计算、大文件校验等计算密集型任务,会占用较多计算资源,影响其他用户正常使用。我们为了保障用户体验,在登录节点设置了任务检测服务,查杀不正常占用登录节点资源的任务,若被检测到您的账号不当使用登录节点,您的账号将会被封禁 30-120 分钟。请务必将这些任务提交到计算节点进行。

3.3 Q: 运行程序时提示缺少 **xxx.so** 文件或者编译/运行程序时显示任务被 **kill**

A: 请确认报错时执行的操作是否是在登录节点,如果是在登录节点出现上述报错,请申请计算节点后再做尝试。

3.4 Q: 计算节点不能访问互联网/不能下载数据

A: 计算节点是通过 **proxy** 节点代理进行网络访问的,因此一些软件需要特定的代理设置。需要找到软件的配置文件,修改软件的代理设置。

a) **git**、**wget**、**curl** 等软件支持通用变量,代理参数设置为:

```
# 思源一号计算节点通用代理设置
https_proxy=http://proxy2.pi.sjtu.edu.cn:3128
http_proxy=http://proxy2.pi.sjtu.edu.cn:3128
no_proxy=puppet,proxy,172.16.0.133,pi.sjtu.edu.cn

# π2.0计算节点通用代理设置
http_proxy=http://proxy.pi.sjtu.edu.cn:3004/
https_proxy=http://proxy.pi.sjtu.edu.cn:3004/
no_proxy=puppet
```

b) **Python**、**MATLAB**、**Rstudio**、**fasterq-dump** 等软件需要查询软件官网确定配置参数:

```
### fasterq-dump 文件, 配置文件路径 ~/.ncbi/user-settings.mkfg

# 思源一号节点代理设置
/tools/prefetch/download_to_cache = "true"
/http/proxy/enabled = "true"
/http/proxy/path = "http://proxy2.pi.sjtu.edu.cn:3128"

# π2.0节点代理设置
/tools/prefetch/download_to_cache = "true"
/http/proxy/enabled = "true"
/http/proxy/path = "http://proxy.pi.sjtu.edu.cn:3004"

### Python需要在代码里面指定代理设置, 不同Python包代理参数可能不同

# 思源一号节点代理设置
proxies = {
    'http': 'http://proxy2.pi.sjtu.edu.cn:3128',
    'https': 'http://proxy2.pi.sjtu.edu.cn:3128',
}
```

(续下页)

(接上页)

```
# π2.0节点代理设置
proxies = {
    'http': 'http://proxy.pi.sjtu.edu.cn:3004',
    'https': 'http://proxy.pi.sjtu.edu.cn:3004',
}

### MATLAB

# 思源一号节点代理设置
proxy2.pi.sjtu.edu.cn:3128

# π2.0节点代理设置
proxy.hpc.sjtu.edu.cn:3004
```

4.11.5 4. 软件安装

4.1 Q: 如何在集群上安装软件?

A: 集群上软件安装，请依次判断适用哪种情况：

- a) 若为商业软件，请自行获取软件使用权并安装；
- b) 若为常用开源软件，请先根据[应用软件](#) 文档，确定集群是否已有安装；
 - 1 若未安装，请先考虑是否能用[conda](#) 方法 安装；
 - 2 再考虑在自己家目录下使用源码安装，遇到问题，请将可复现的步骤，发至[hpc](#) 邮箱获取帮助；
 - 3 软件还有[容器安装](#) 的方法；
 - 4 我们也将对常用开源软件进行评估，以便全局部署。欢迎[邮件联系我们](#)。

4.2 Q: 普通用户如何使用 **sudo** 安装软件?

A: 有别于独占的个人电脑和工作站，高性能计算用户共享软硬件设施，使用 **sudo** 特权操作极有可能影响其他用户的程序和数据，因此普通用户禁止使用 **sudo**。通常普通用户无需 **sudo** 就能在家目录中安装和使用软件，且使用 **sudo** 安装的软件会被错误安装在本地文件系统上而不能在计算节点上运行。

请参考 [应用软件](#) 了解当前集群提供的软件模块或通过[hpc](#) 邮箱告诉我们需要安装的软件。

普通用户也可以使用容器的方式安装，容器内用户拥有“模拟 root 权限”，具体请见 [容器](#)。

对于需要 **sudo** 安装的商业软件，请参考 [FAQ 4.2](#) 由软件厂商工程师工程师联系我们指定安装方案。

4.11.6 5. 收费和充值

5.1 Q: 如何收费?

A: 请发送邮件至 [hpc](mailto:hpc@sjtu.edu.cn) 邮箱咨询。

5.2 Q: 如何缴费?

A: 校内转账可在“我的数字交大”网页，或“交我算”APP 里完成，具体操作请见：

[计算服务费校内经费转账说明](#)

如有任何财务问题，请联系网络信息中心基础部杨老师，电话 34206060-8017，邮箱 yjqedith@sjtu.edu.cn

5.3 Q: 如何查看账户余额?

A: 您可以使用主账号或子账号登录 [计费系统](#) 查看。也可以使用账号负责人 jAccount 登录 [计算账单](#) 页面。

5.4 Q: 财务办理充值，仍未到账

A: 关于充值未到账，可咨询网络信息中心基础部杨老师，电话 34206060-8017，邮箱 yjqedith@sjtu.edu.cn

咨询时请提供：

- 拟充入的 jAccount
- 充值金额
- 财务凭证号（财务入账后，财务网站页面会显示财务凭证号）

5.5 Q: 有没有机时奖励政策?

A: 目前暂无奖励政策。

4.11.7 6. 邮件支持

6.1 Q 向 [hpc](mailto:hpc@sjtu.edu.cn) 邮箱发送的邮件多久才能收到回复?

A: 根据邮件内容不同，下面是用户支持响应周期：

1. 邮件确认：1 个工作日内；
2. 调整作业脚本：1 个工作日内；
3. 排查异常中断作业：2~3 个工作日内；
4. 新建账号：2 个工作日内；
5. 编译不包含在 [Spack](#) 或自行开发的软件包：1~2 周内。

4.11.8 7. 集群通知

Q 如何及时获取集群通知?

A: 集群通知会实时发布在用户微信群。请发邮件 [hpc 邮箱](#) 给我们, 将为您添加进用户微信群。

4.11.9 8. 致谢模版

Q 如何在论文中致谢交大高性能计算?

A: 致谢模版如下。欢迎大家将已接收的高质量成果邮件分享给我们。

(中文) 本论文的计算结果得到了上海交通大学高性能计算中心的支持和帮助;

(英文) The computations in this paper were run on the π 2.0 (or the Siyuan-1) cluster supported by the Center for High Performance Computing at Shanghai Jiao Tong University.

4.11.10 9. 医学院和附属医院申请 **jAccount** 账号

A: 请至 [附属医院账号申请及充值流程](#) 页面了解和办理。

4.11.11 10. 如何重置.bashrc 和.bash_profile

A: 用户家目录下的 ~/.bashrc 和 ~/.bash_profile 记录 bash shell 配置, 若配置不当可能会导致无法找到可执行文件、无法在 Studio 中启动 RSession 等问题, 需要重置这两个配置文件的内容。

重置 ~/.bashrc 操作流程如下, 首先登录集群, 然后备份现有配置文件, 再调用 vim 或其他文本编辑器打开文件:

```
$ /bin/cp ~/.bashrc ~/.bashrc.bak
$ /bin/vim ~/.bashrc
```

将 ~/.bashrc 文件内容重置如下, 保存后退出编辑器:

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

类似地, 在命令行中使用 `/bin/cp ~/.bash_profile ~/.bash_profile.bak; /bin/vim ~/.bash_profile` 将文件内容重置如下:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
```

(续下页)

(接上页)

```
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
```

最后重新登录集群，确认重置配置文件后，先前的问题是否解决。重置配置文件会导致您先前对 **bash shell** 的自定义配置失效，如果您仍需要保留这些自定义配置，建议您从 **bak** 备份文件中逐条转移这些配置，避免引入导致应用异常语句。

4.12 文档下载

Pi 超算平台用户手册

Pi 超算平台简明手册

异常帐号处置规范