上海交通大学校级计算平台

Pi 2.0 用户文档手册

https://docs.hpc.sjtu.edu.cn

2020 年 11 月

Contents

上海交通大学交我算平台是校级计算公共服务平台，成立于 2013 年，旨在对校内大规模科学与工程计算需求提供技术支撑。

1 欢迎致谢交我算平台

致谢模板及优秀论文见平台成果展示：https://hpc.sjtu.edu.cn/Item/Science.htm

2

交我算 **HPC+AI** 平台

- “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。

3 文档使用

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。

1. 快速上手
2. 系统
3. 帐号
4. 密码
5. 登录
6. 远程桌面
7. 作业
8. 软件
9. *GNU Compiler Collection*
10. *Intel Compiler*
11. 常见问题

4 注意事项

- 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 快速上手

4.1.1 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；您可能有太多数据，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。

4.1.2 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 计算平台是国内高校计算力能最强的人工智能计算平台。该平台由 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。

4.1.3 资源如何选择？

交我算 HPC+AI 平台采用 CentOS 的操作系统，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

4.1.4 使用交我算 HPC+AI 平台需要什么？

您只需要一个交我算帐号，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

4.1.5 如何申请帐号？

交我算平台服务于交大师生。帐号申请者需为交大及附属医院在职教师或博士后。

主帐号申请

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帐号。

子帐号申请

每个主帐号可免费申请子帐号。

子帐号由帐号负责人或子帐号使用人在“交我办”(或 我的数字交大 my.sjtu.edu.cn) 中的“交我算”里申请。

开通帐号之前，还需帐号使用者完成“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

4.1.6 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

4.1.7 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)
```

(下页继续)

(续上页)

```
    {

     /* 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？ 。

4.1.8 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

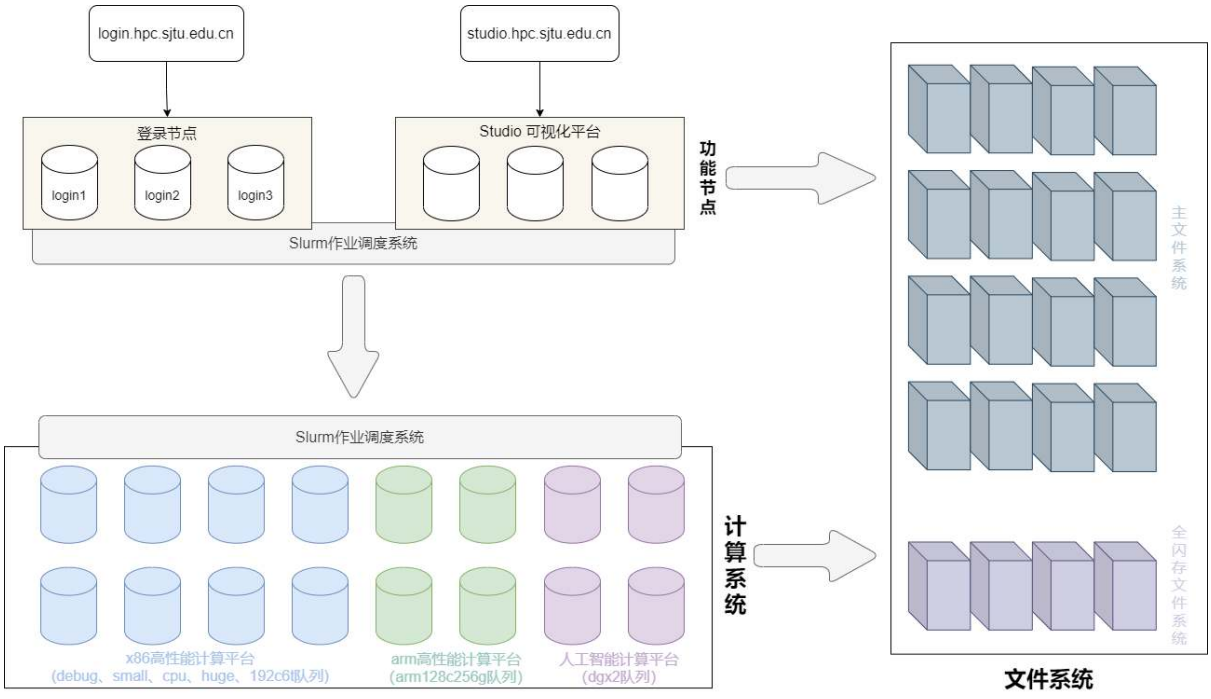
4.2 系统

交我算 HPC+AI 平台集群有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：



4.2.1 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 /archive。

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，可用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
#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 #####"
```

(下页继续)

(续上页)

```
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 \
--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 \
```

(下页继续)

(续上页)

```
--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。

归档文件系统

在 *data* （*data.hpc.sjtu.edu.cn*）节点的目录 */archive* 下挂载了挂挡存储，共 3P 容量，用来存储用户的不常用数据。

归档文件系统特性

归档文件系统主要使用机械硬盘搭建，可提供大容量、高可用的存储供用户使用。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。归档文件系统作为主文件系统的一个补充，主要提供给用户存储不常用的数据（冷数据），从而释放主文件系统的存储空间、缓解主文件系统的存储压力。

** 注意：和主文件系统以及全闪存文件系统不同，归档文件系统只能在 *data* 节点访问，无法在计算节点和登录节点访问，也就是说保存在该文件系统的数据不能在计算节点读取并参与计算，因此只推荐保存不常使用的数据。**

如何使用归档文件系统

(1) 登录 *data* 节点

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

(2) 进入归档文件系统

用户可以在以下路径找到 /archive 提供的个人存储空间: `/archive/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 $ARCHIVE` 即可进入。

(3) 将不常用文件移动到 *$ARCHIVE*

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

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

4.2.2 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
64c512g	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
arm128c256gm	arm	CPU: 2 x HiSilicon Kunpeng 920-6426(2600MHz, 64cores) Mem: 16 x Samsung 16GB DDR4 2933	100

4.3 帐号

上海交通大学交我算平台是校级计算平台，面向全校师生提供科研与教学支持。

4.3.1 帐号申请

准备工作

为了使用交我算 HPC+AI 平台集群，您需要：

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

帐号说明

在交我算的帐号体系中，帐号会被分为主帐号和子帐号。

主帐号是收缴费主体，由课题组负责人申请。

主帐号与子帐号均为独立帐号，仅在计费关系上存在关联。

每个主帐号能免费申请子帐号，个数不限，请按需申请。

主帐号和子帐号申请

由帐号负责人或帐号使用人在“交我办”（或 我的数字交大 my.sjtu.edu.cn ）中的“交我算”里申请。我们将会在两个工作日内开通帐号。

4.3.2 密码

密码与帐号保护

当完成帐号创建之后，我们会发送一封邮件给您，邮件中包含了您帐号的密码。请您务必保管好帐号密码不外泄。另外，我们强烈建议您收到帐号密码后立即重置密码。

集群安全策略要求用户密码满足如下全部三项要求：

1. 大于等于 8 位；
2. 包含大写字母、小写字母、数字、符号中至少两种类型字符；
3. 密码中不能包含用户名子字符串 (>=3 字符)。

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。

密码自助更改

如果您知道您的密码，并且需要更改密码，需要：

1. 连接到集群的登录节点
2. 使用 `passwd` 指令
3. 终端会提示您输入一次旧密码，并输入新密码

至此，您已经更新了您的密码，请妥善保存新的密码。

忘记密码

如果您因为种种原因忘记了密码，请发送邮件至 HPC 邮箱 ，在邮件中注明您要重置密码的帐号。如果您不是帐号申请人，为了确认您的身份，请务必抄送帐号申请人的邮箱。我们会在确保本次重置并非恶意重置之后完成密码重置工作，并发送邮件通知您新的密码。

密码规范

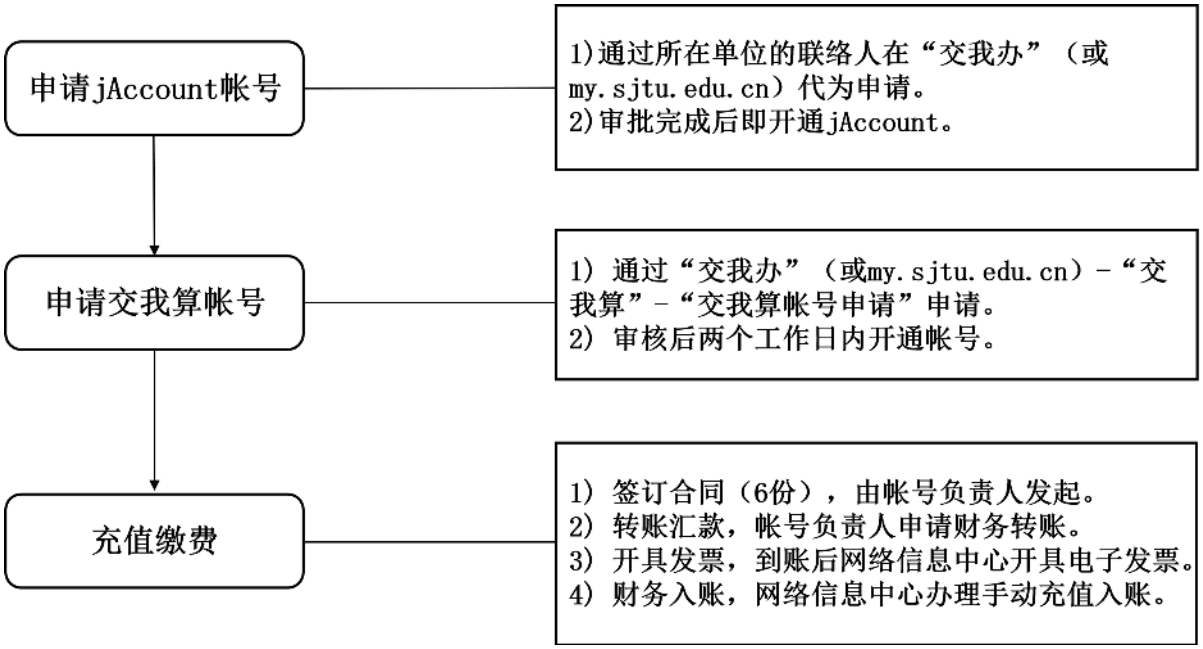
如果您要自行更改密码，我们建议您遵循如下规则制定新的密码：

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

4.3.3 附属医院教职工帐号申请及充值流程

“交我算”帐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”帐号申请。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 帐号代申请，通过后您便可自行申请“交我算”帐号。

以下是附属医院教职工“交我算”帐号申请及充值流程图：

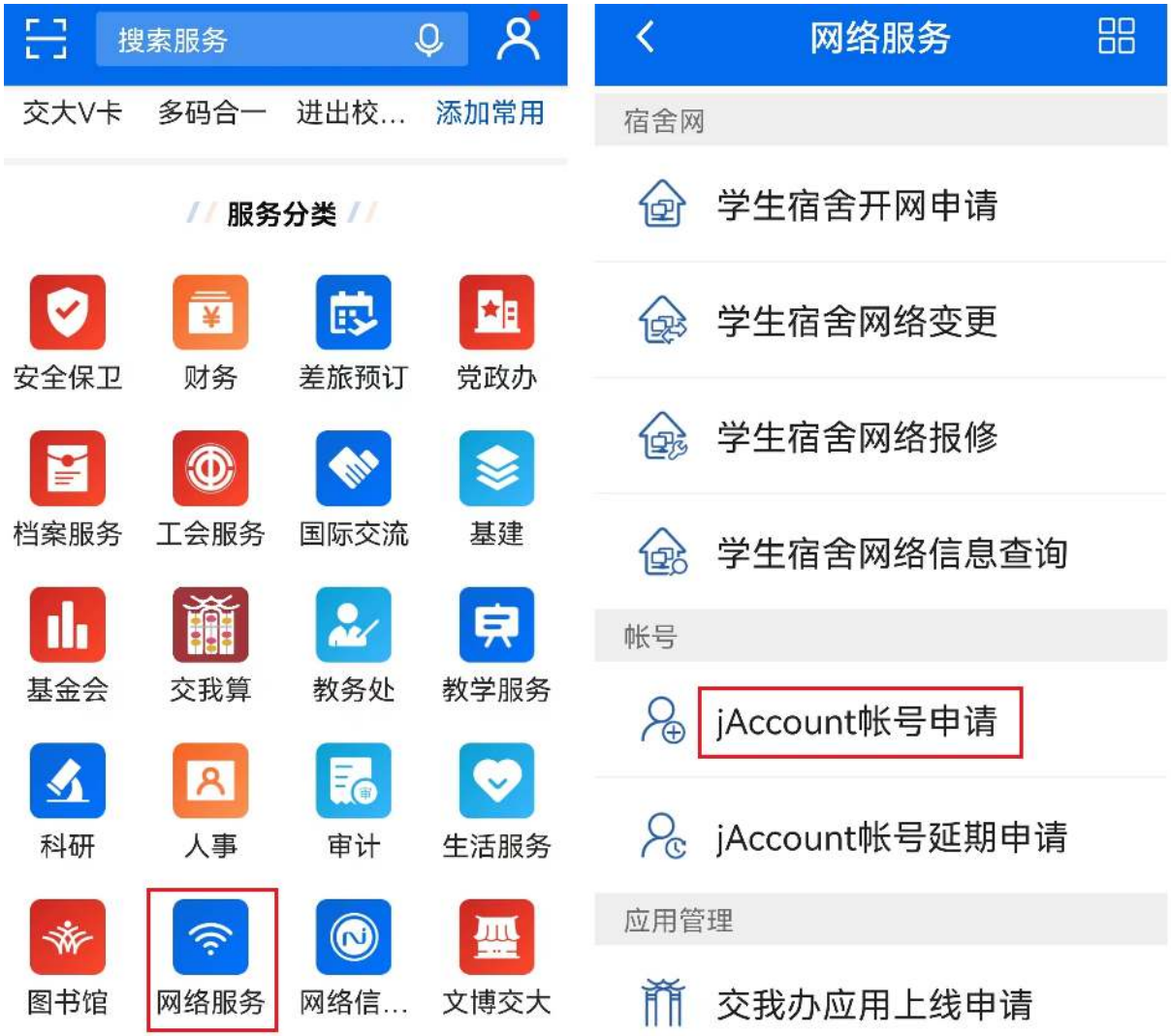


第一步：申请 jAccount 帐号

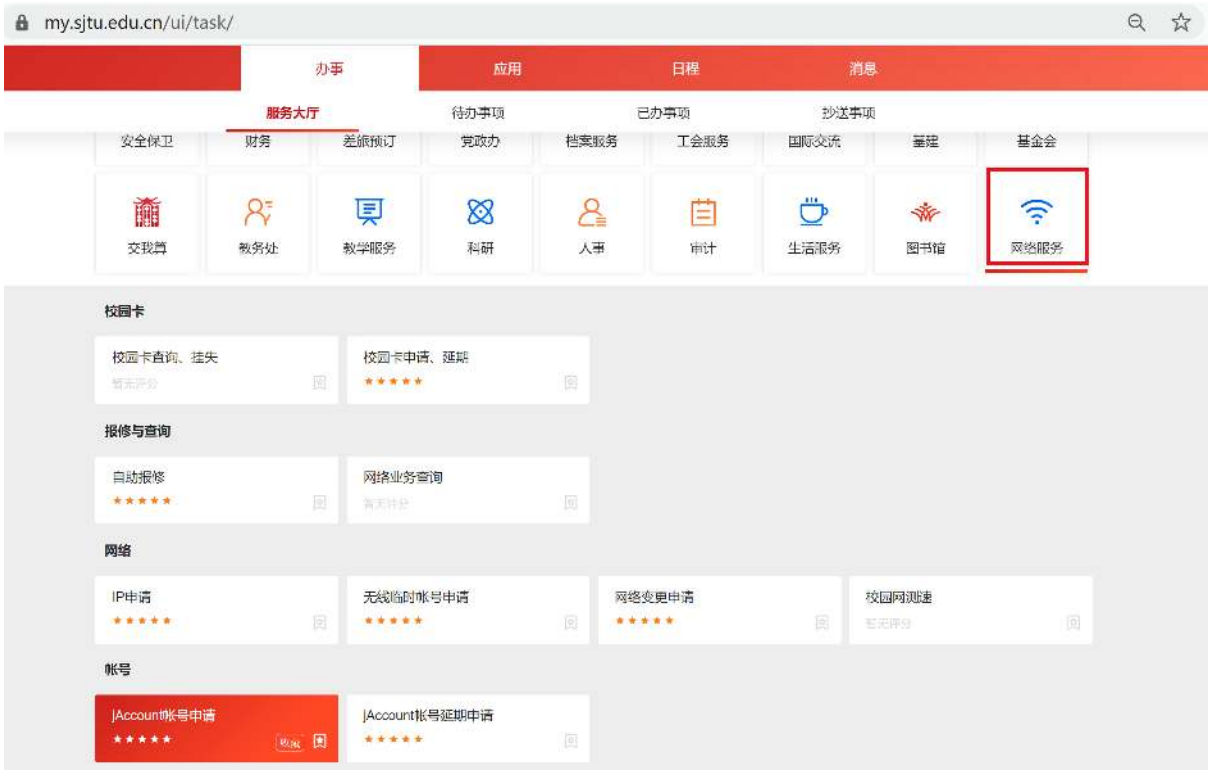
通过所在单位联络人在“交我办”APP（或网页版 my.sjtu.edu.cn）上代为申请 jAccount。经所在单位人事干事或领导审批完成后即开通 jAccount，且自动开通与 jAccount 同命名的交大邮箱。

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 帐号申请”



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

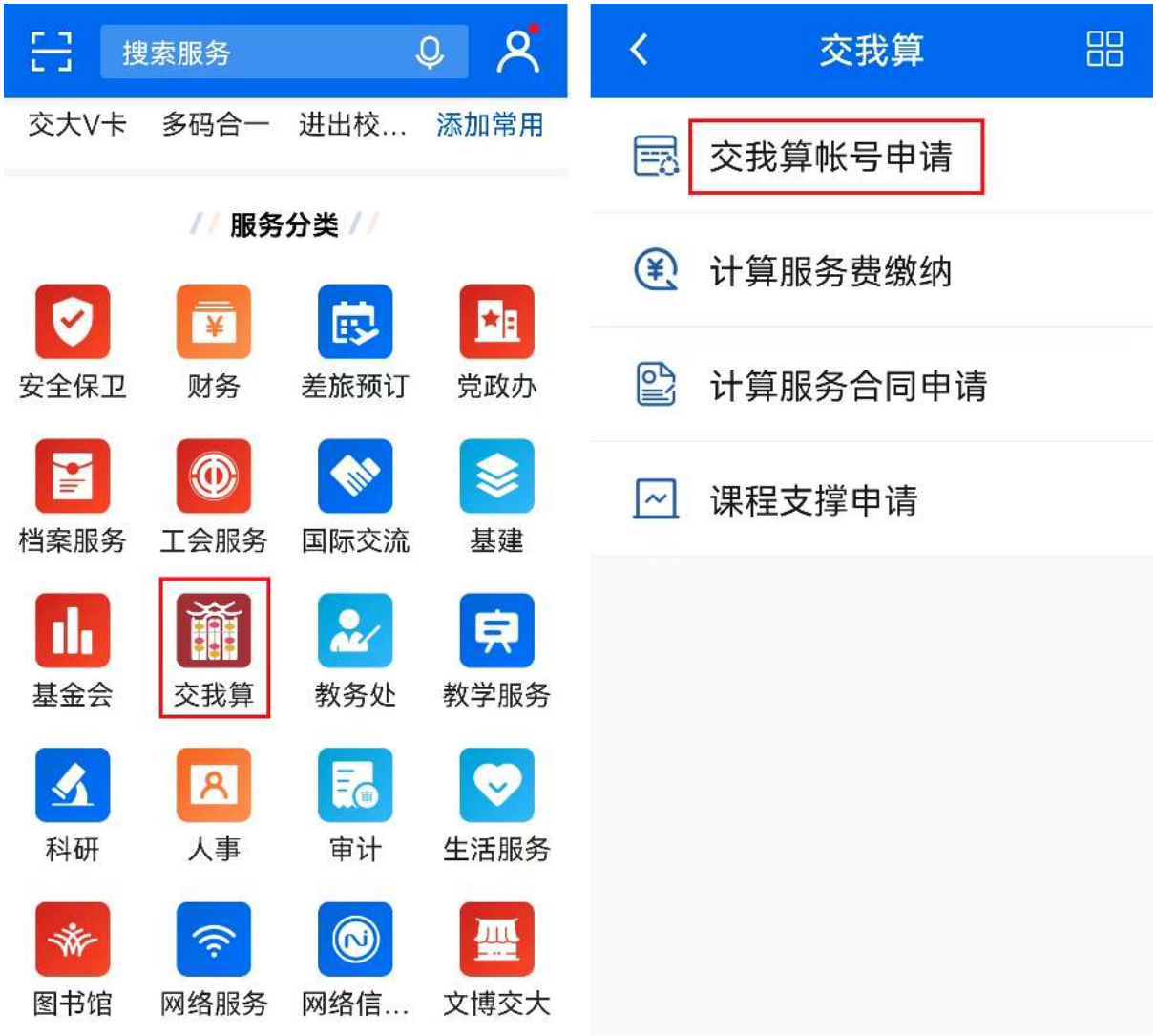


第二步：申请交我算帐号

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

交我算帐号申请入口（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”-“交我算帐号申请”



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

第三步：充值缴费

1. 签订合同及合同盖章

- 由帐号负责人发起，将合同内相关内容填好并盖单位公章，然后将 6 份合同原件寄给网络信息中心王思婷老师（上海市闵行区东川路 800 号上海交通大学网络信息中心 101 室，34206060-8011）
- 合同下载：计算平台技术服务合同
- 6 份合同盖好交大公章后，其中 3 份合同原件将回寄给帐号负责人。

2. 转账汇款

- 帐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，请帐号负责人发邮件至 hpc@sjtu.edu.cn 申请。

3. 开具发票

- 汇款到账后，请帐号负责人联系网络信息中心王老师 stwangecho@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帐号负责人。

4. 财务入账

- 帐号负责人确认”交我算”帐号用户名及 jAccount 后，由网络信息中心办理手动充值入账。
- 帐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王老师 stwangecho@sjtu.edu.cn，电话 34206060-8011。

4.4 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。

Studio 支持的应用如下，并持续增加:

- RStudio
- Jupyter
- TensorBoard
- Code Server
- IGV
- Octave
- ParaView
- Relion
- VMD
- Visit
- OVITO
- MATLAB
- AlphaFold

4.4.1 HPC Studio

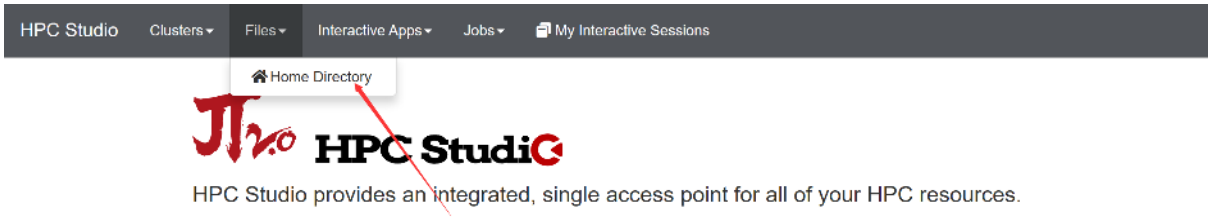
登录

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。

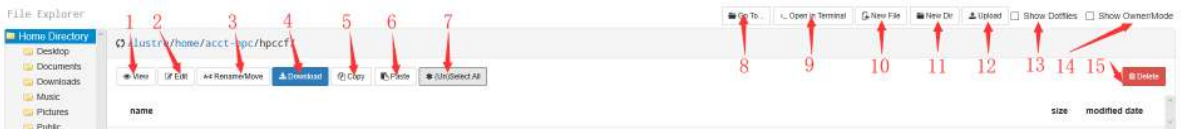
文件管理

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：

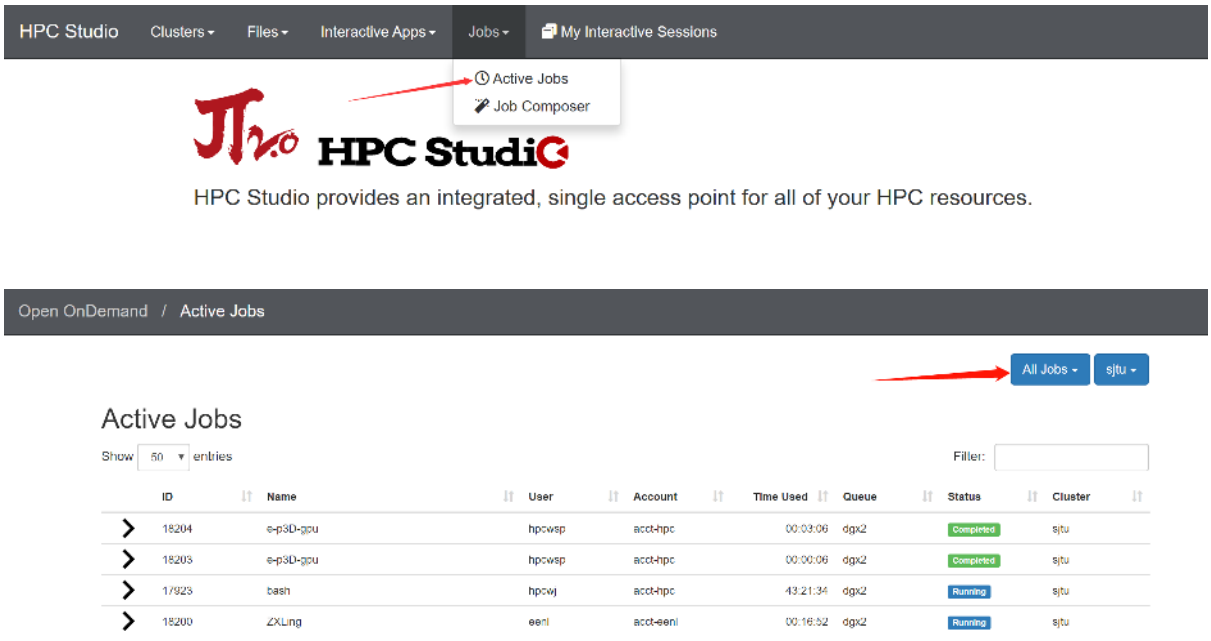
功能	详细功能	按钮
查看	点击文件可查看文件内容，点击文件夹可显示文件属性	1
编辑	仅可编辑文本文件，编辑文件有多种编辑器可选，均为系统嵌入，无需链接本地编辑器	2
重命名/移动	重命名或移动文件	3
下载	下载文件，若下载的为文件夹则会自动压缩成压缩包进行下载	4
拷贝/粘贴	复制文件	5 6
全选/取消全选	选择文件功能	7
进入 ···	进入到其他路径	8
打开终端	打开 webshell	9
新建文件	新建文件	10
新建目录	新建目录	11
上传	上传文件	12
显示隐藏文件	点击可显示隐藏文件/文件夹	13
显示属主信息	显示文件/文件夹属主/大小/日期等信息	14
删除	删除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



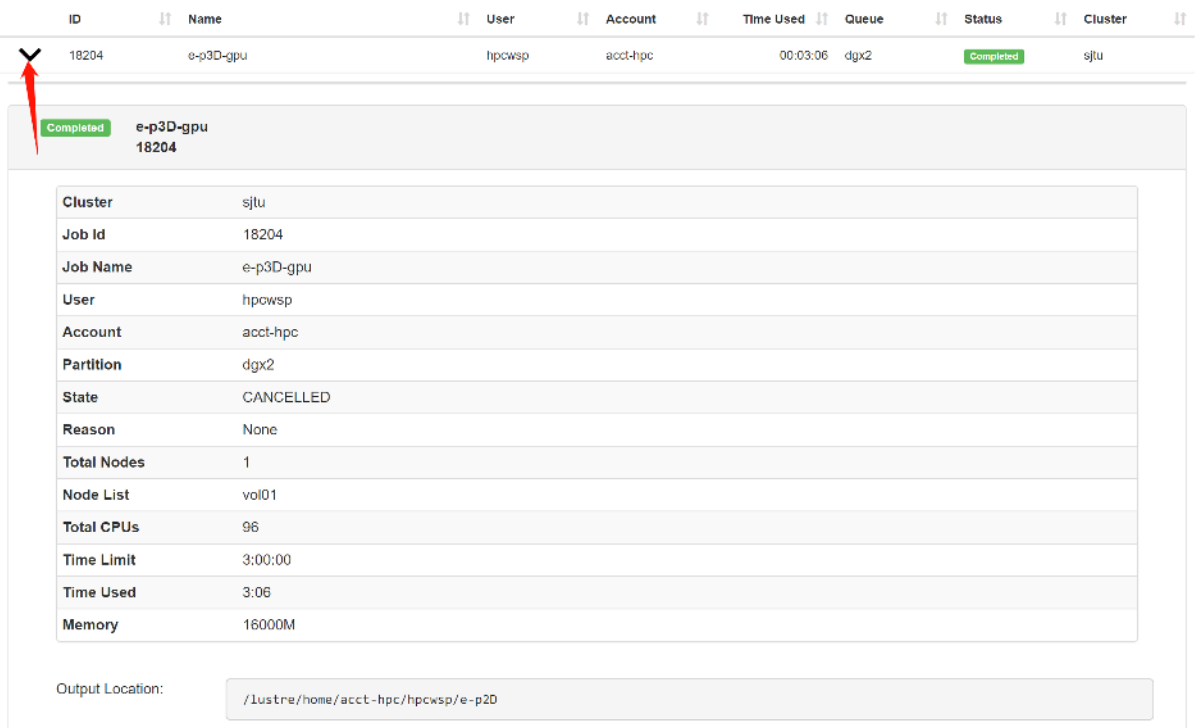
作业

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。

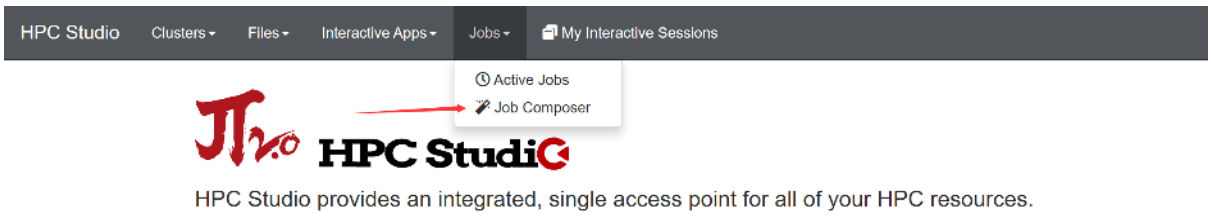


点击作业项前的箭头查看作业详情。

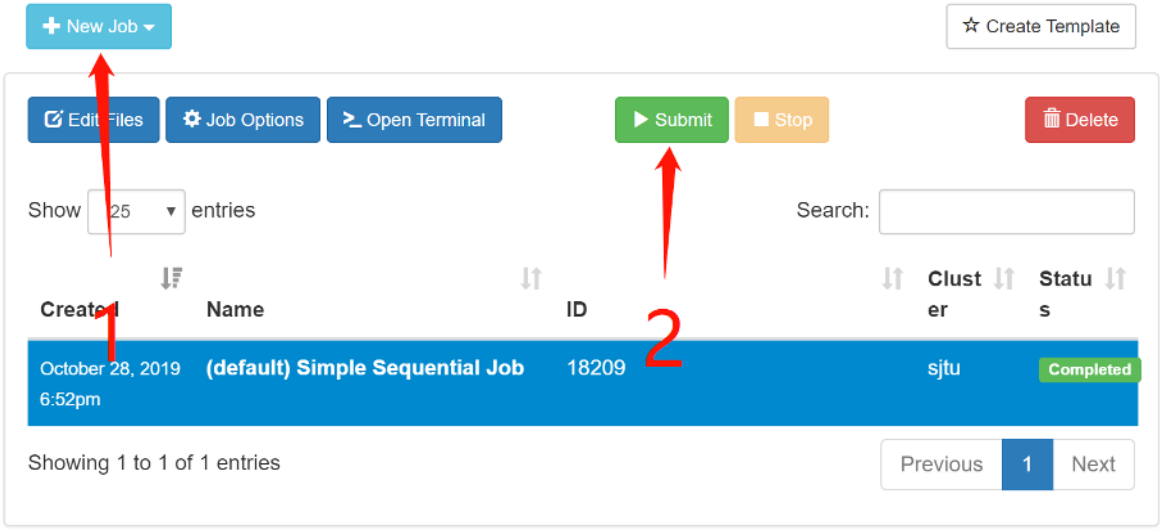


提交作业

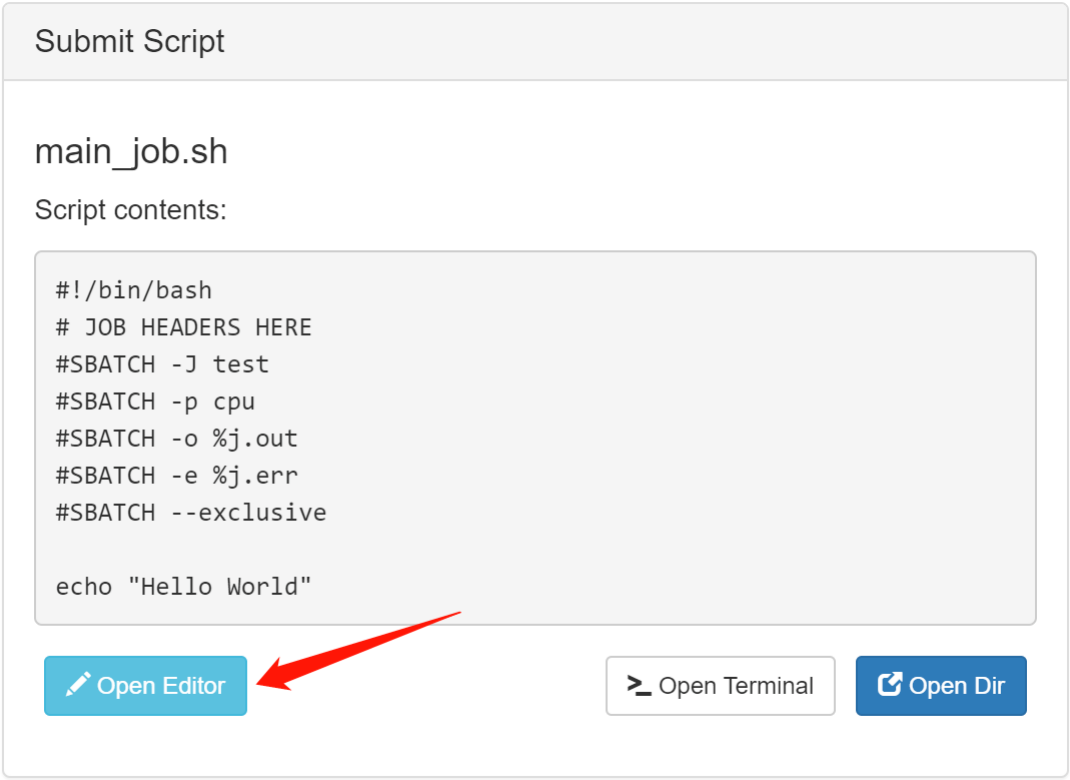
点击 `Jobs->Job Composer`，打开新建作业选项卡。

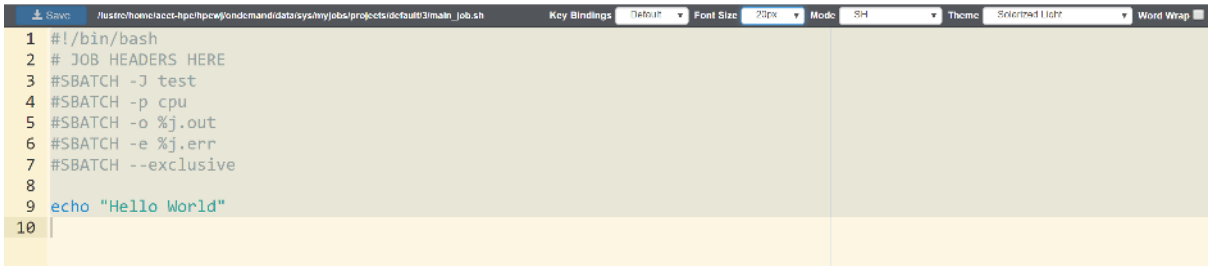


点击左上角 `New Job`（1）按钮，选择模板，点击 `submit`（2）按钮提交作业。



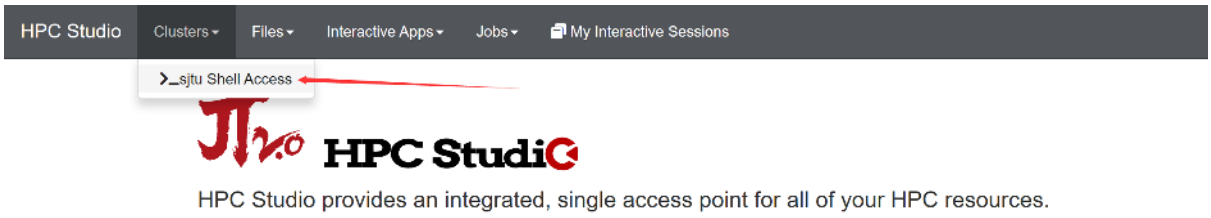
同时 HPC Studio 提供了在线文本编辑功能，在右侧底部的 `Submit Script` 选项卡中，点击 `Open Editor` 按钮，即可打开文本编辑器。





在线 **Shell** 终端

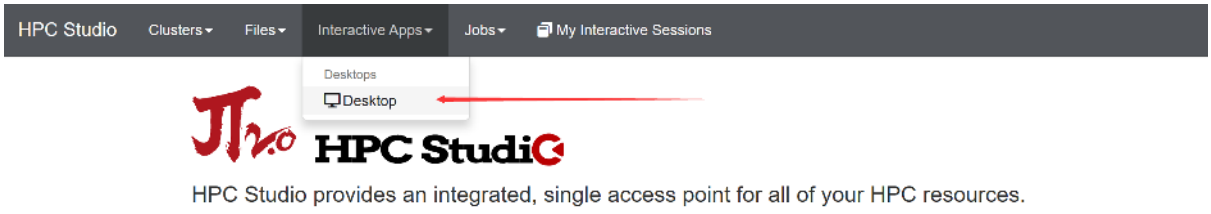
点击 `Clusters->sjtu Shell Access`，打开在线终端。



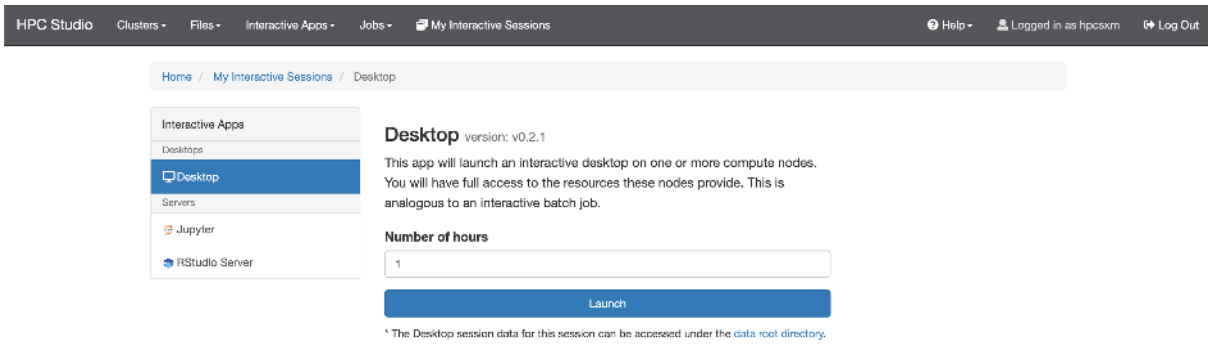
远程桌面

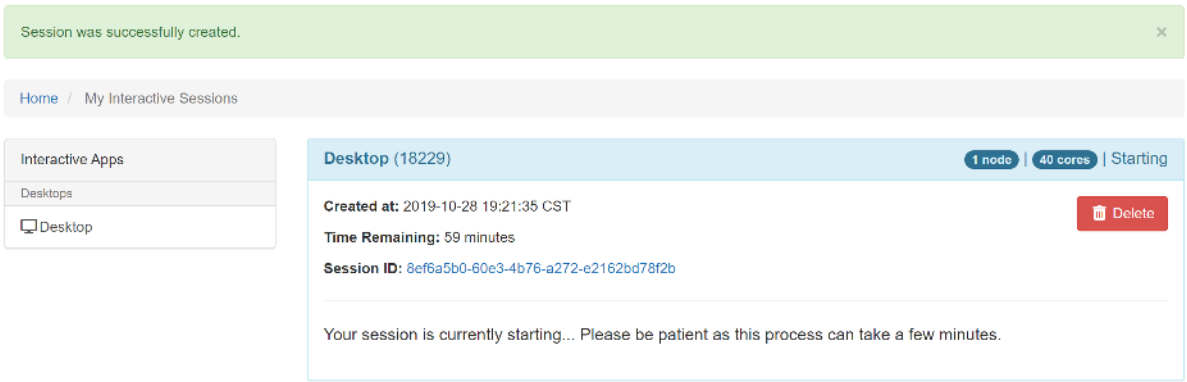
远程桌面打开方式需先提交一个空的作业取得计算节点的控制权（此操作会计入机时）。

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

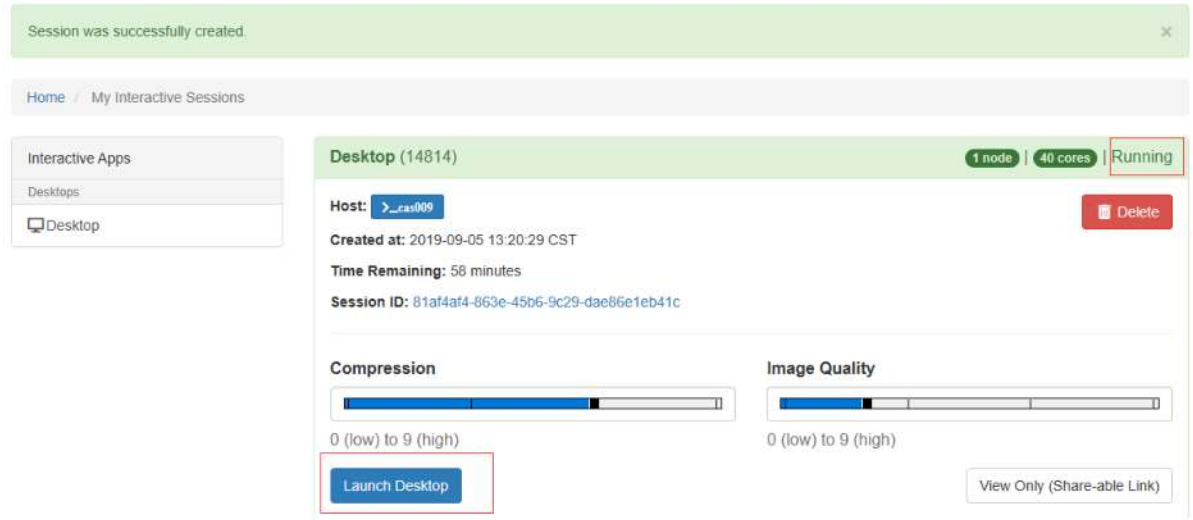


Number of hours 默认是 1，然后点击 launch 即可进入桌面选项卡。





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。



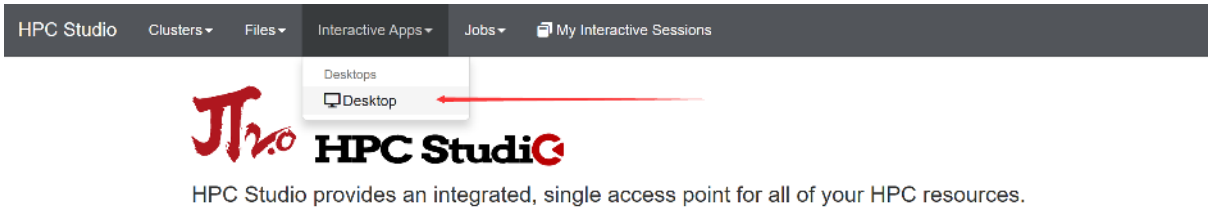
参考教学视频

2022 春季用户培训之 HPC Studio

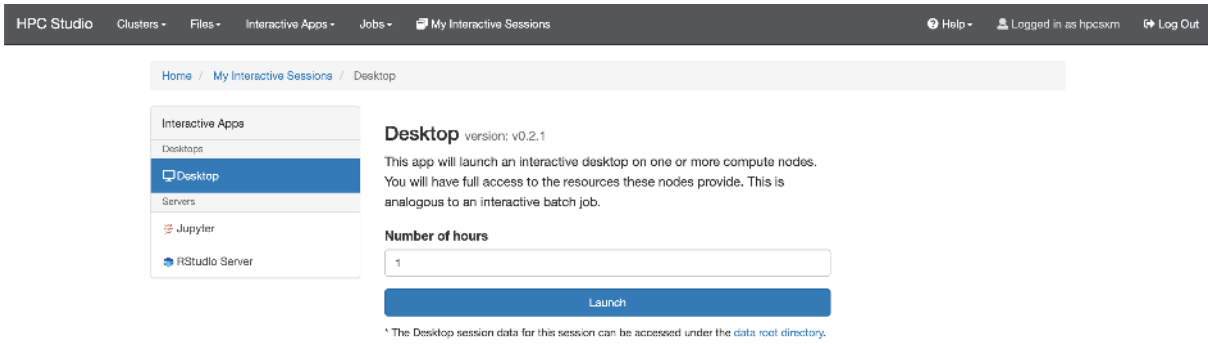
4.4.2 远程桌面

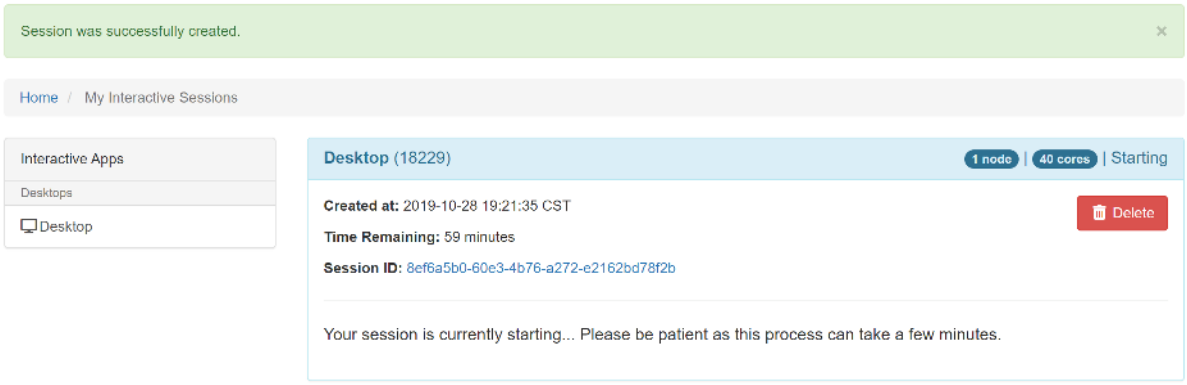
远程桌面打开方式需先提交一个空的作业取得计算节点的控制权（此操作会计入机时）。

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

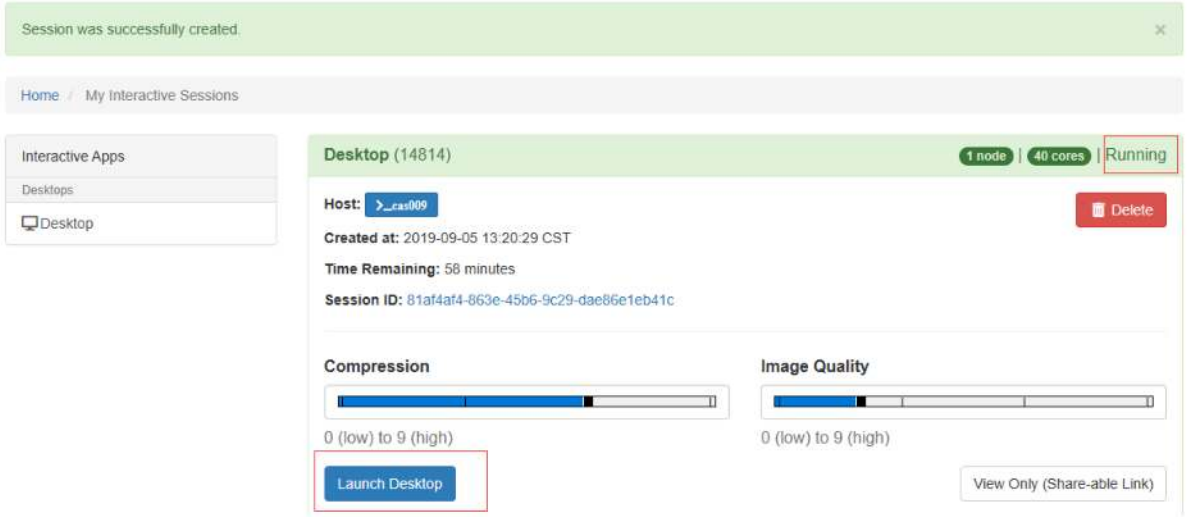


Number of hours 默认是 1，然后点击 launch 即可进入桌面选项卡。





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。



4.4.3 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 `Jupyer (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
cudatoolkit	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
cudatoolkit	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 jupter 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 jupter 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 Wikepedia
- Jupyter Home

4.4.4 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。

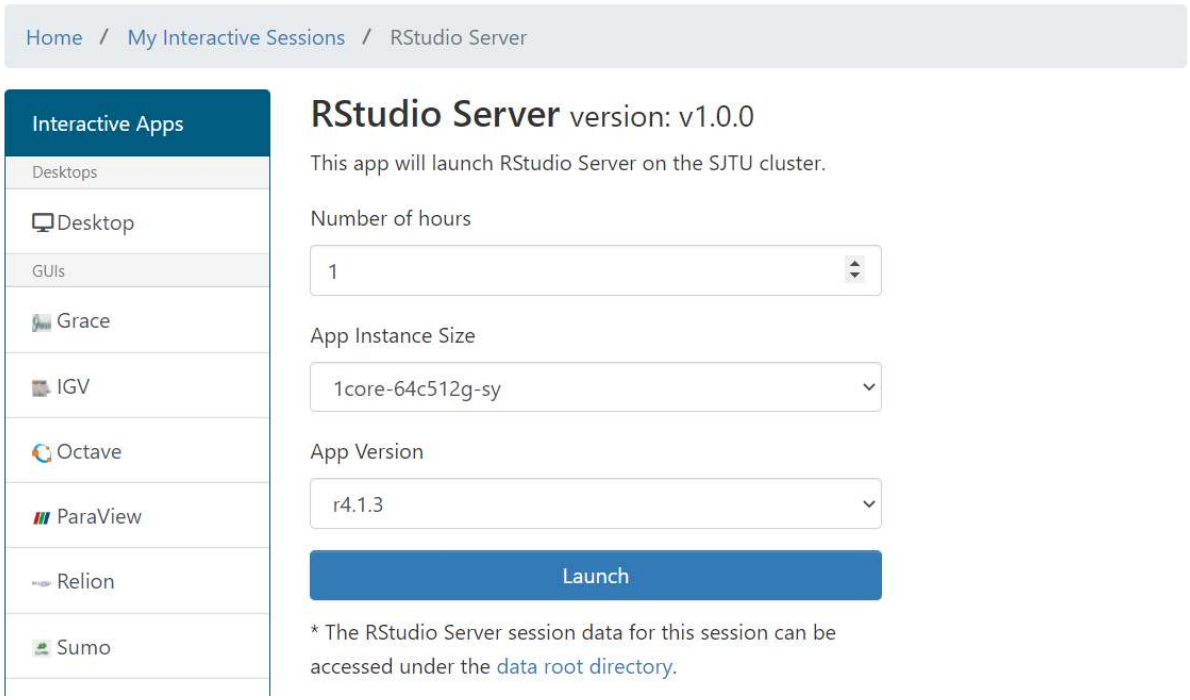
可用的版本

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

如何使用

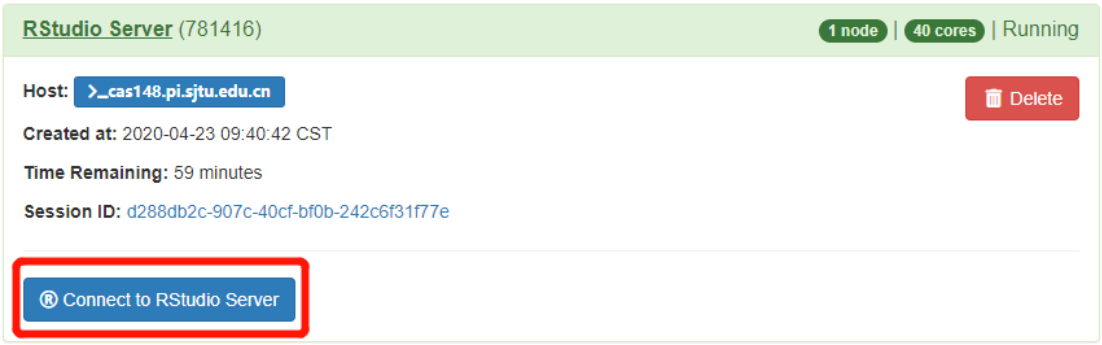
用超算的账号及密码登录 HPC Studio，在内置应用中选择 `RStudio Server`，如下图：

点击后会出现相关的选项卡，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：*-pi 为 π 集群的资源，*-sy 为思源一号的资源。

待界面从等待变成 Running 后，可使用 Connect to RStudio Server 连接到 Rstudio Server：



运行示例

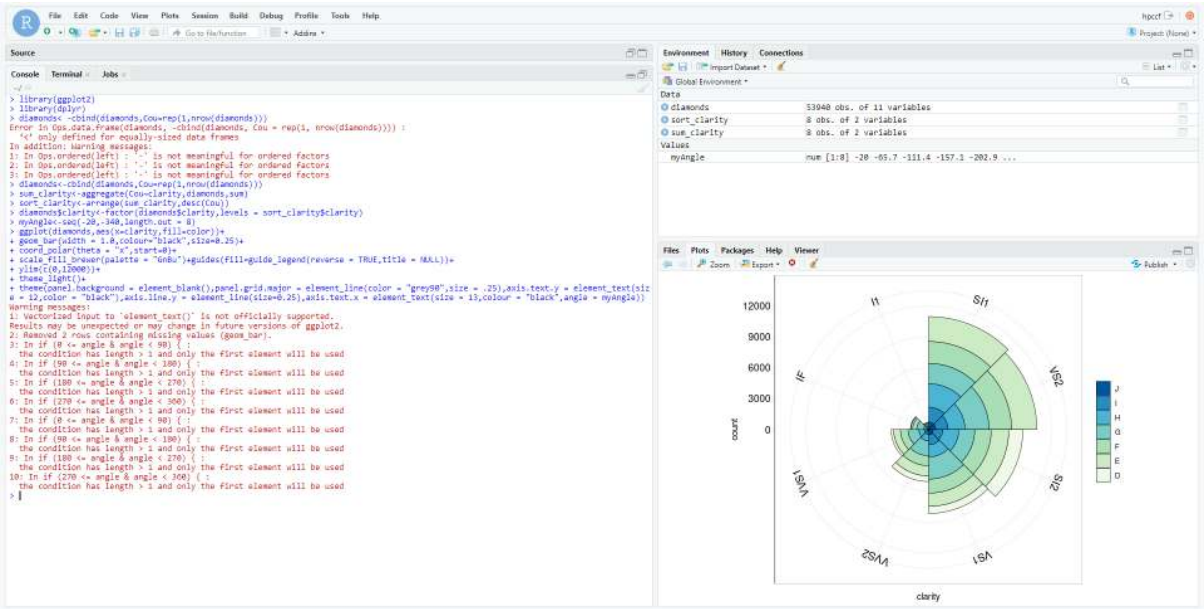
所需的 R 依赖包需要自行下载：

```
library(ggplot2)
library(dplyr)
diamonds< -cbind(diamonds,Cou=rep(1,nrow(diamonds)))
sum_clarity<-aggregate(Cou~clarity,diamonds,sum)
sort_clarity<-arrange(sum_clarity,desc(Cou))
diamonds$clarity<- factor(diamonds$clarity, levels = sort_clarity
↪$clarity)
myAngle <-seq(-20,-340,length.out = 8)
```

(下页继续)

(续上页)

```
ggplot(diamonds,aes(x=clarity,fill=color))+
geom_bar(width=1.0,colour="black",size=0.25)+
coord_polar(theta = "x",start=0)+
scale_fill_brewer(palette="GnBu")+guides(fill=guide_
↪legend(reverse=TRUE,title=NULL))+ ylim(c(0,12000))+
theme_light()+
theme( panel.background = element_blank(),
     panel.grid.major = element_line(colour = "grey80",size=.25),
     axis.text.y = element_text(size = 12,colour="black"),
     axis.line.y = element_line(size=0.25),
     axis.text.x=element_text (size = 13,colour="black",angle =␣
↪myAngle))
```



4.4.5 Paraview

ParaView 是对二维和三维数据进行分析和可视化的程序，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

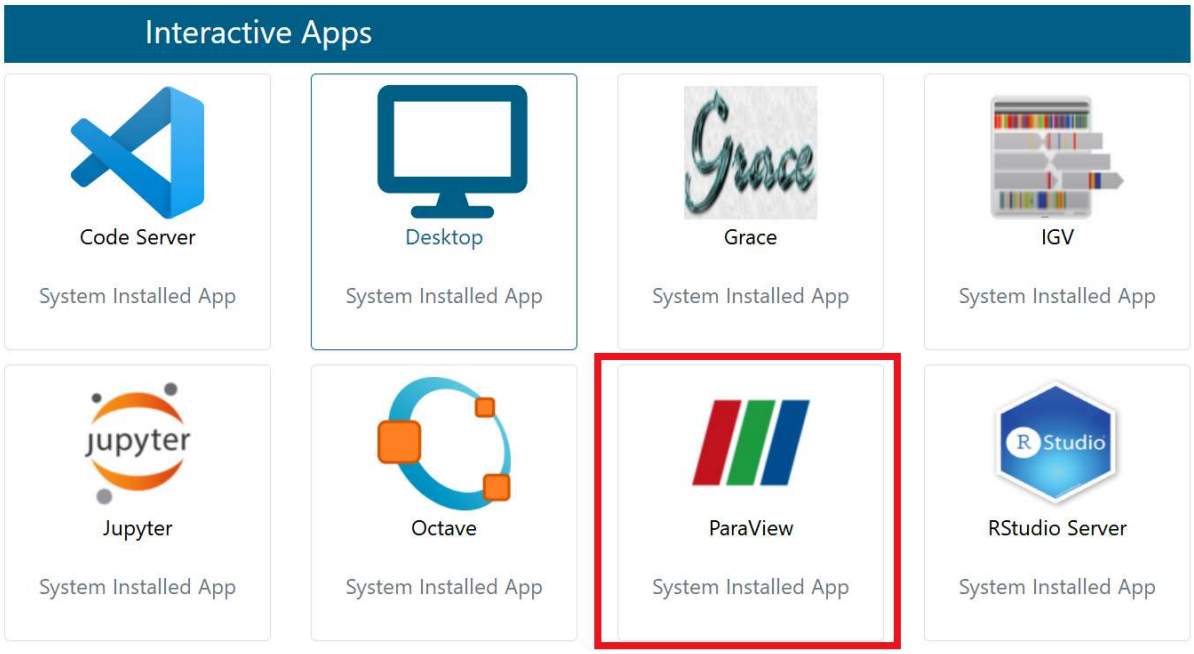
1. 首先根据 OpenFOAM 使用说明文档 的步骤运行 cavity 算例，运行成功后在对应 cavity 目录下会得到如下目录：

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

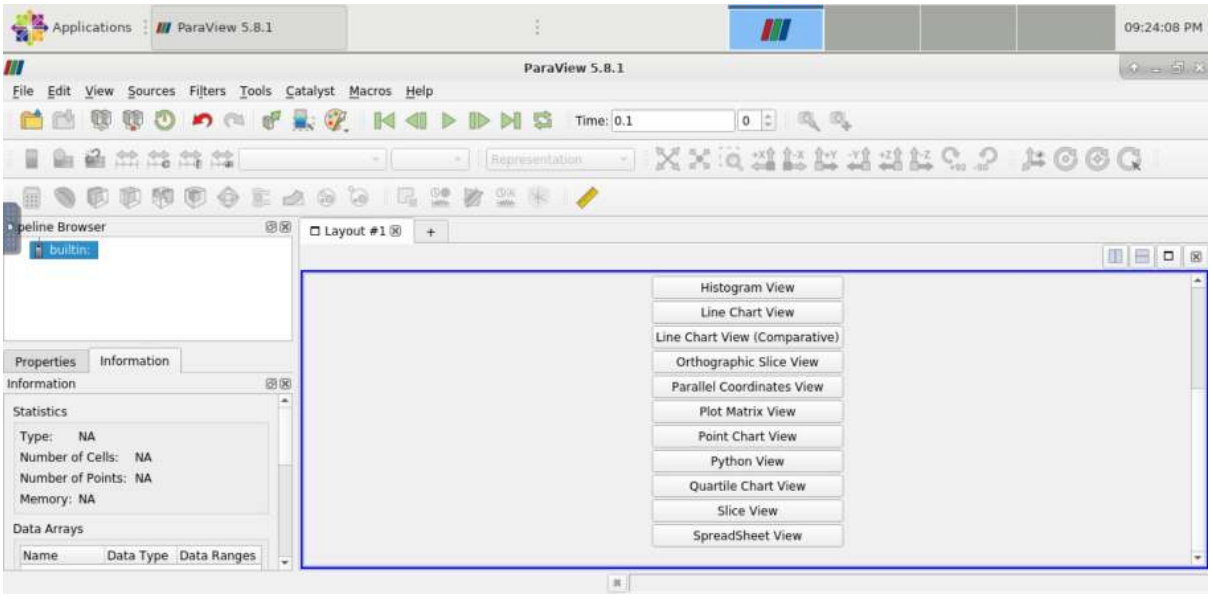
2. 在 cavity 目录下新建 mycase.foam 文件 (内容为空)：

```
touch mycase.foam
```

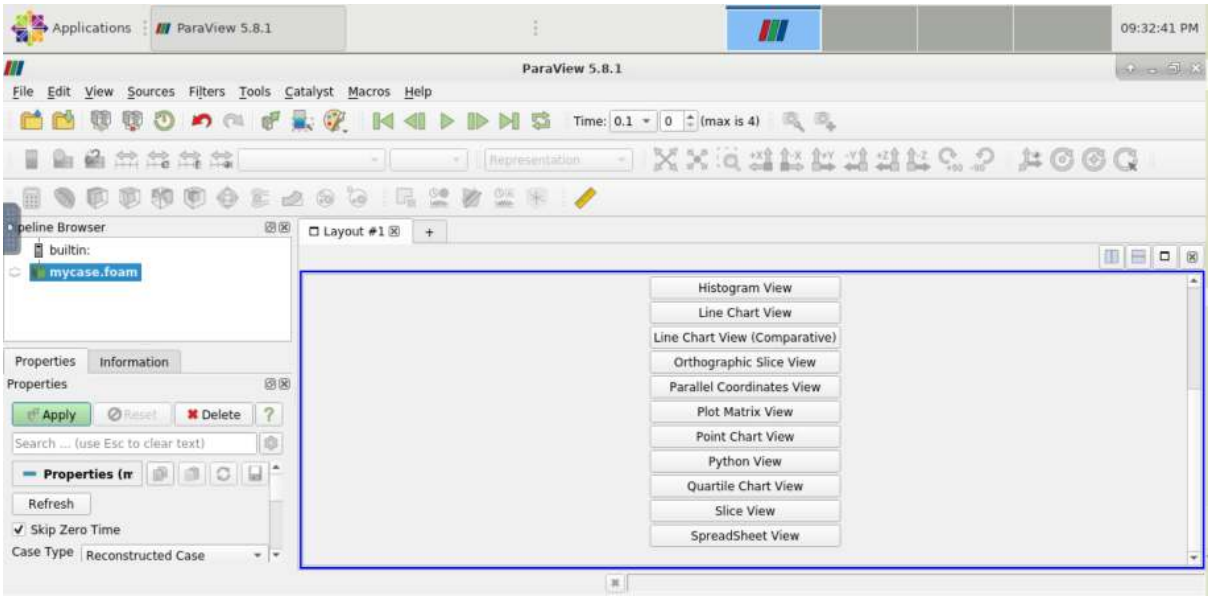
3. 用 pi 集群帐号登录 HPC Studio 平台。然后在 Interactive Apps 面板中点击如下图所示 ParaView 选项申请 Paraview GUI 界面 (要排一会儿队，请耐心等待)：



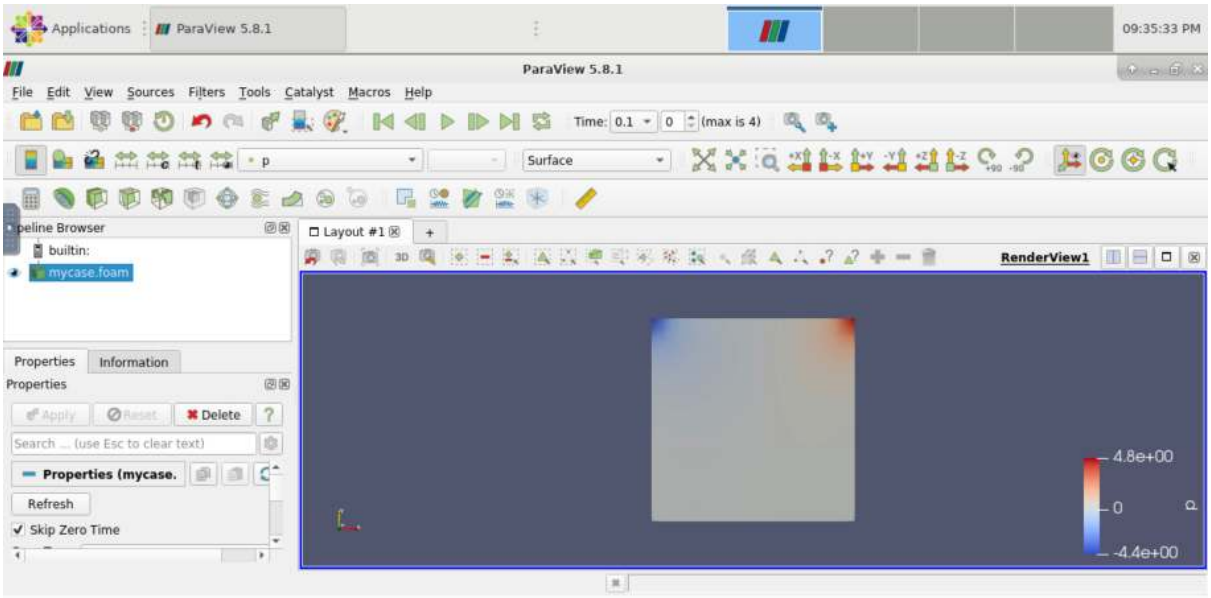
4. 申请成功后会看到如下界面：



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 imegio==2.0.0
(pvgeoenv)$ pip install PVGeo==2.0.0
```

修改环境变量:

执行命令:

```
(pygeoenv)$ 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**

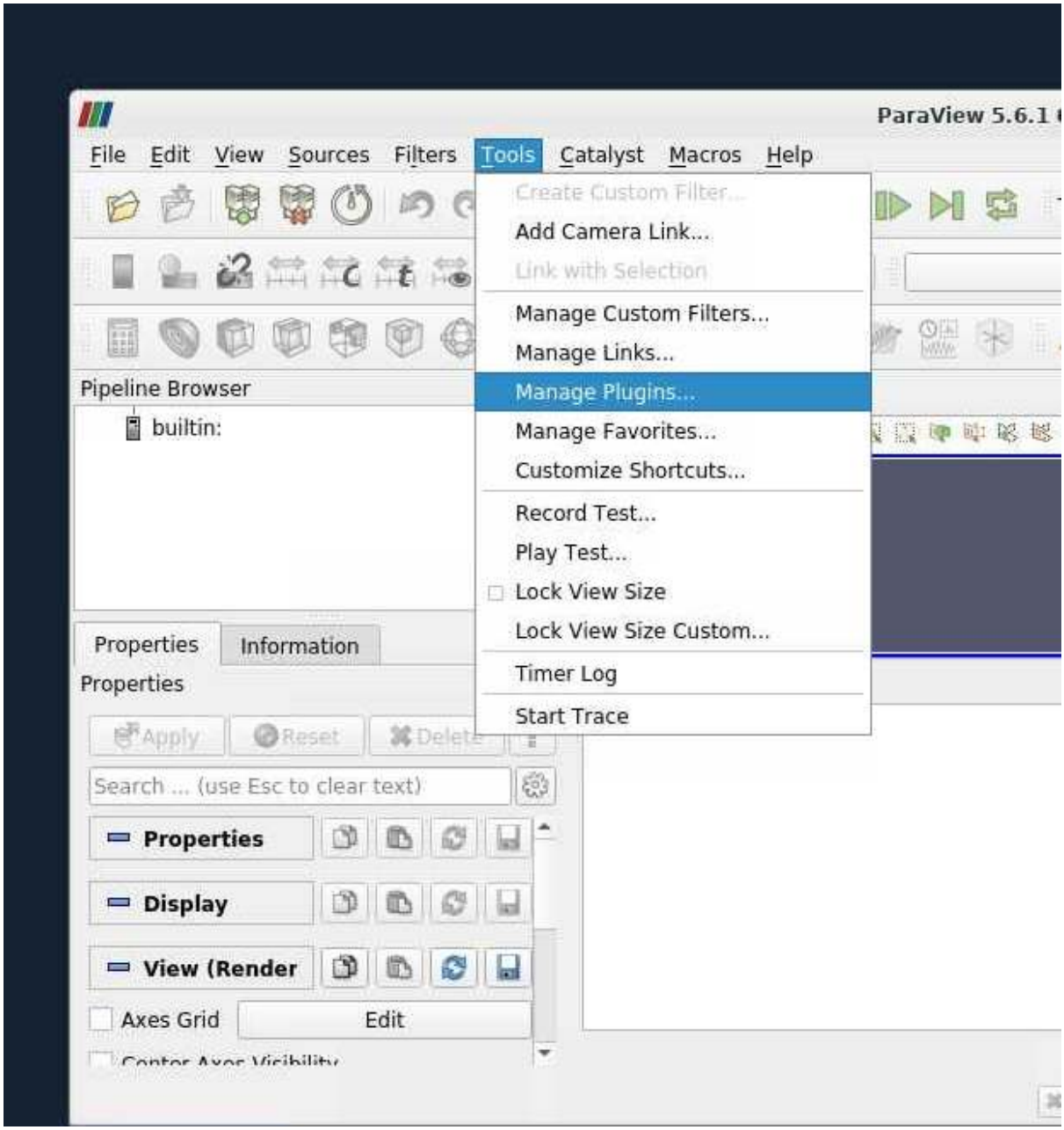
由于 paraview 的启动需要 GUI 的支持，需要进入超算的可视化平台，申请一个远程桌面。

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：

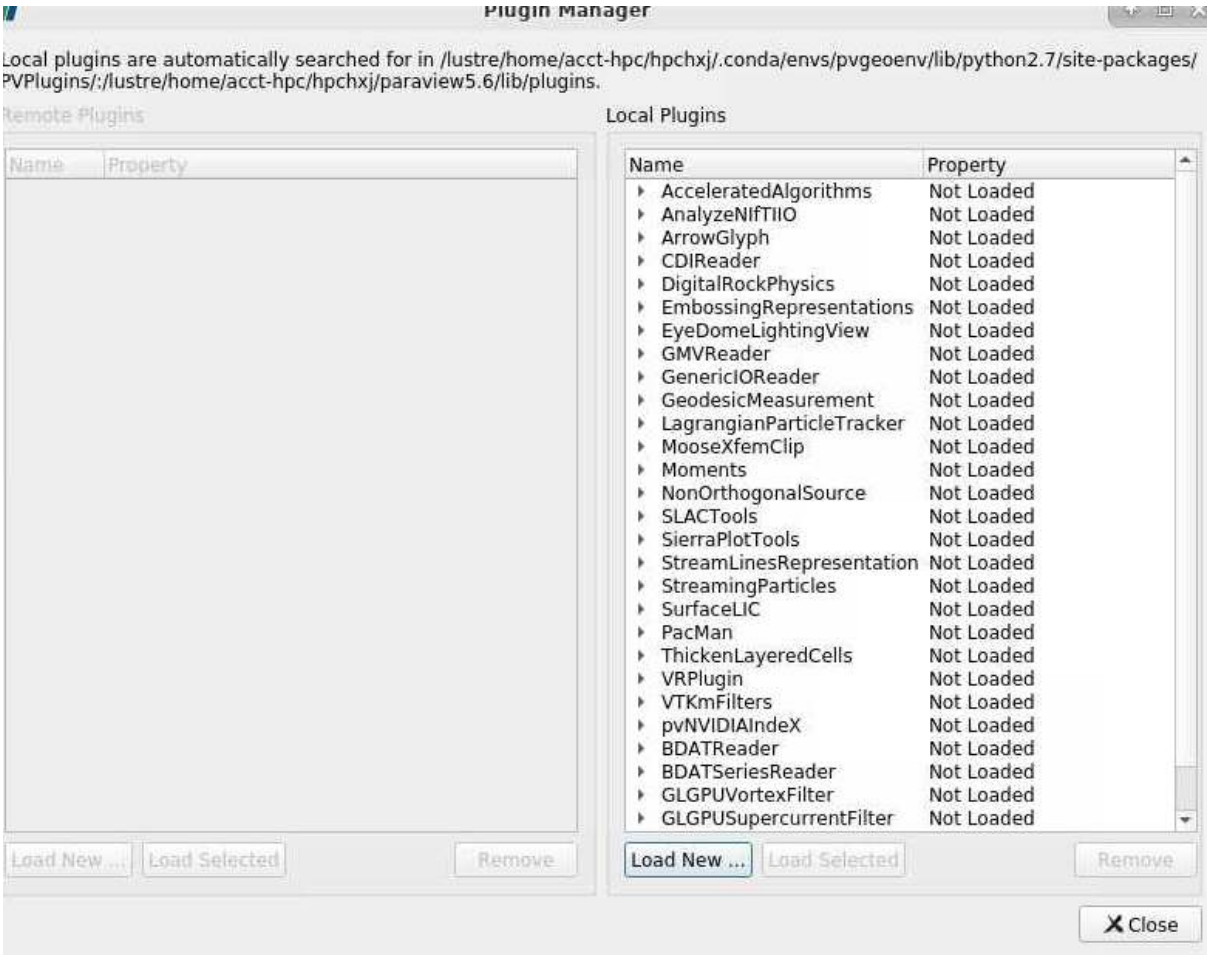
```
$ MESA_GL_VERSION_OVERRIDE=3.3 paraview
```

添加 **PVGeo** 插件

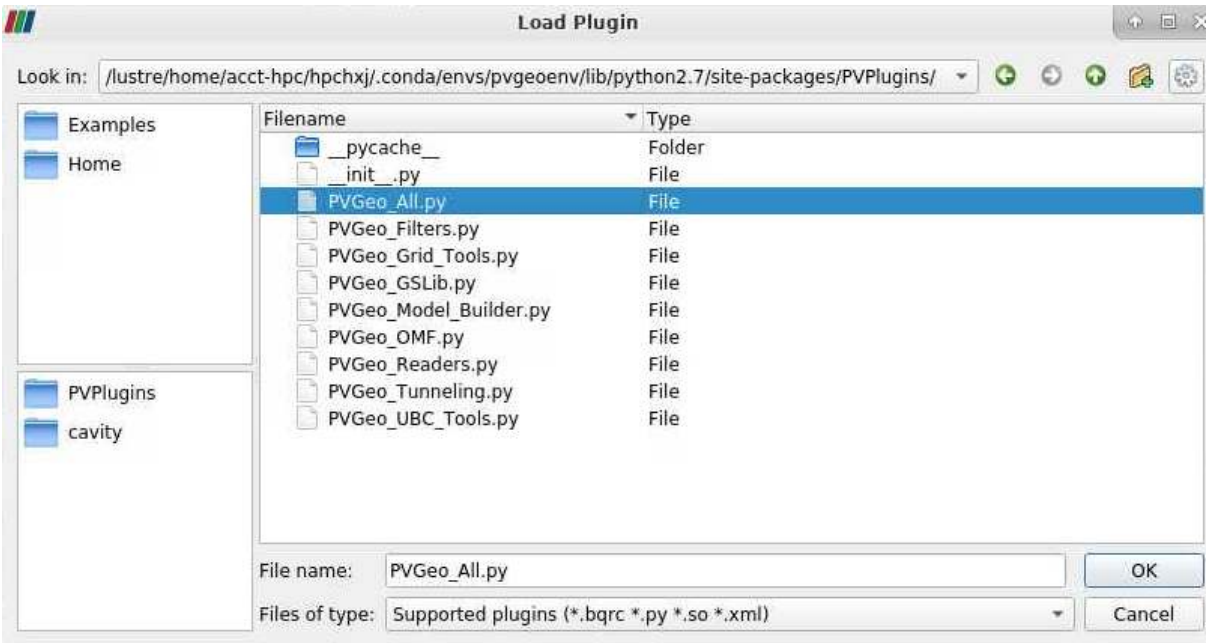
打开 tools-manage plugins 界面



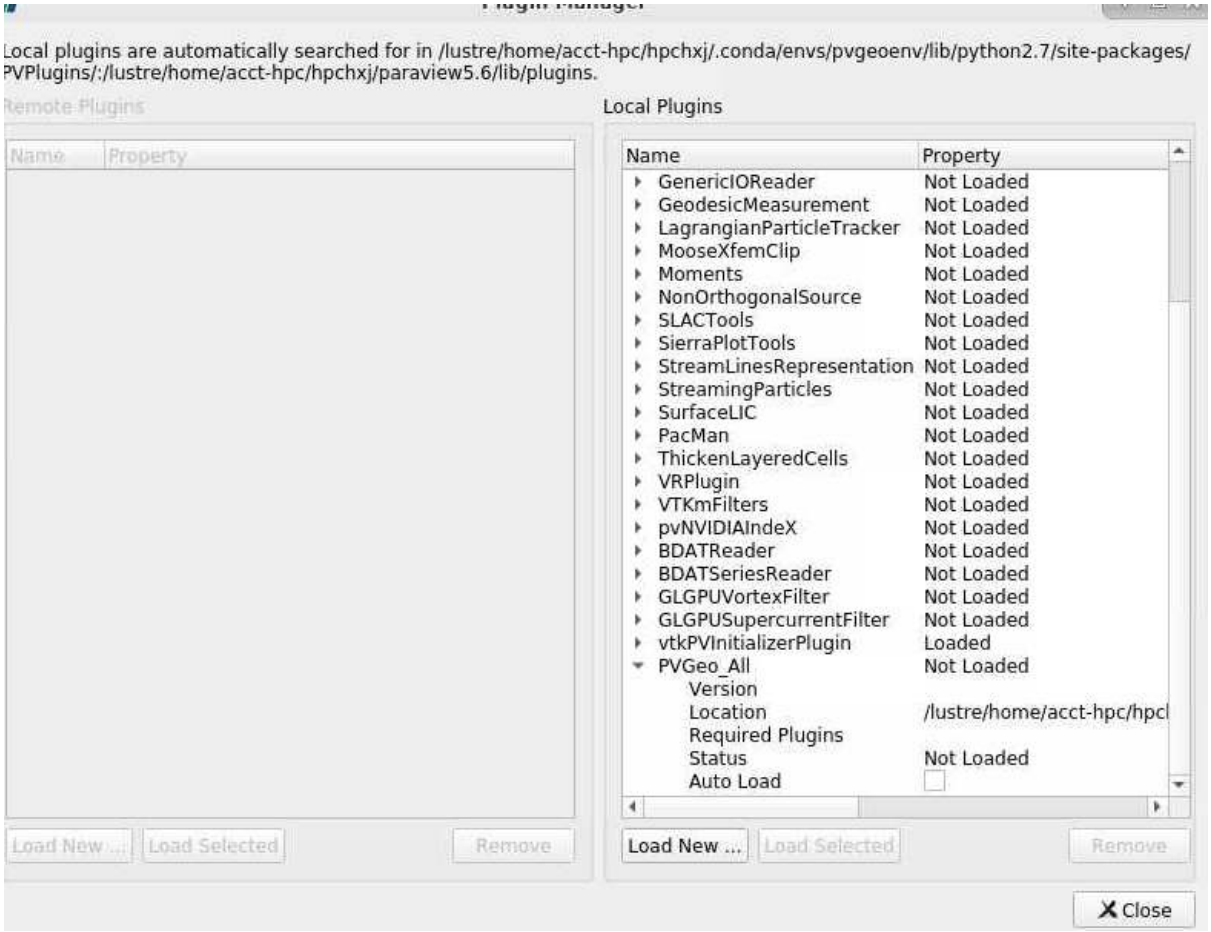
点击右下角 Load New…按钮



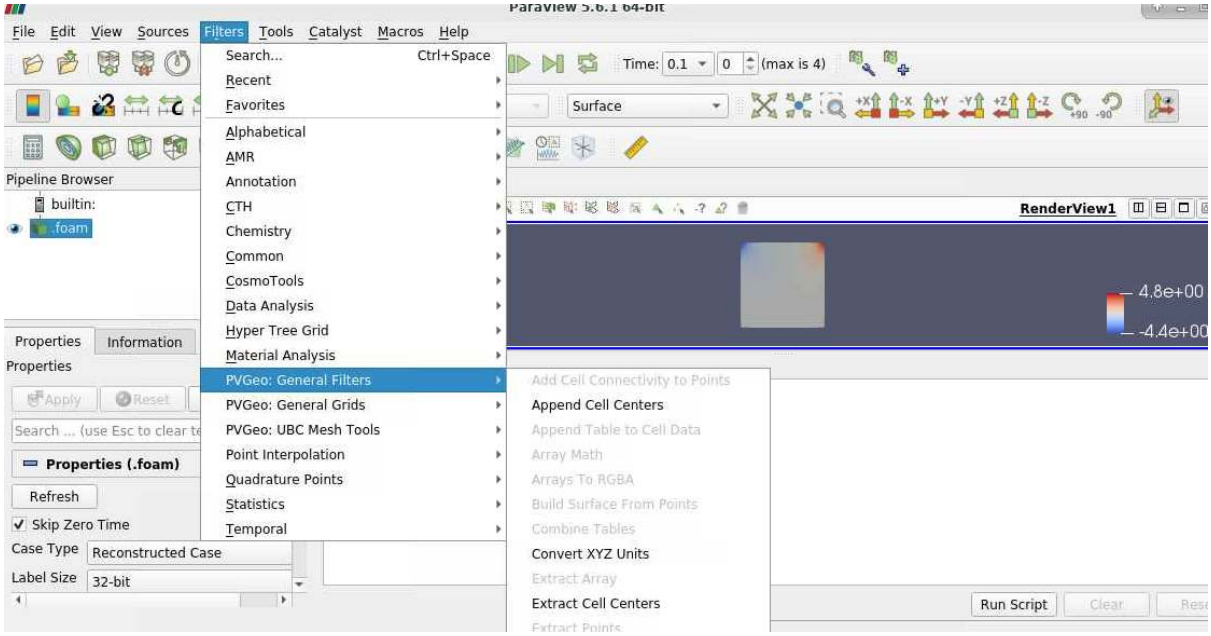
加载 PVGeo_All.py 文件:



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:

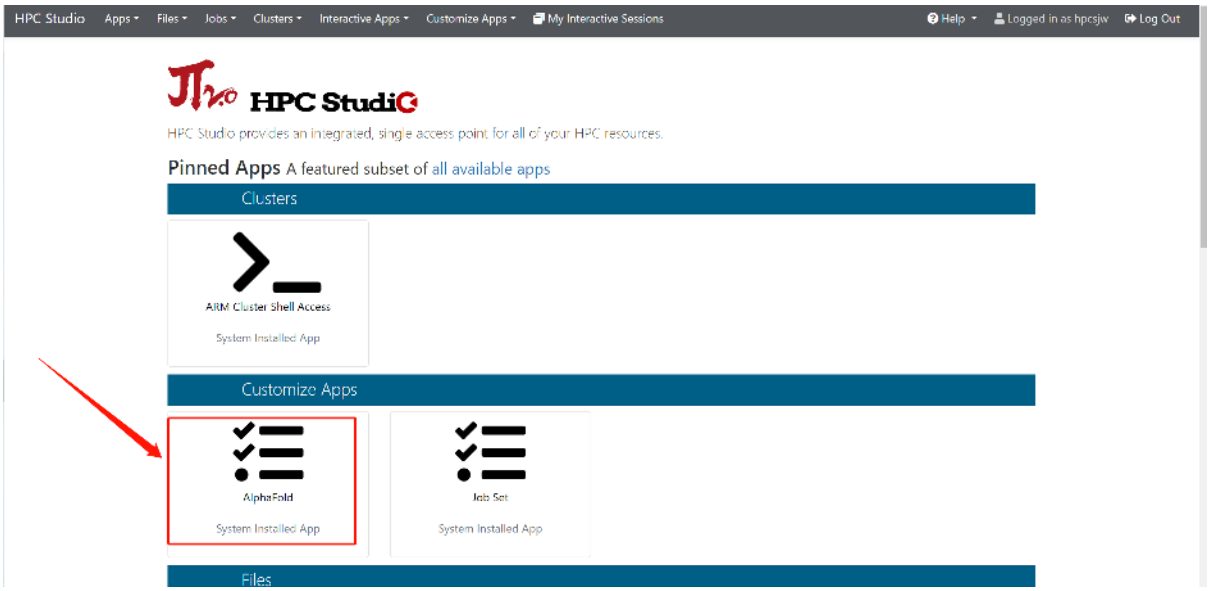


参考资料

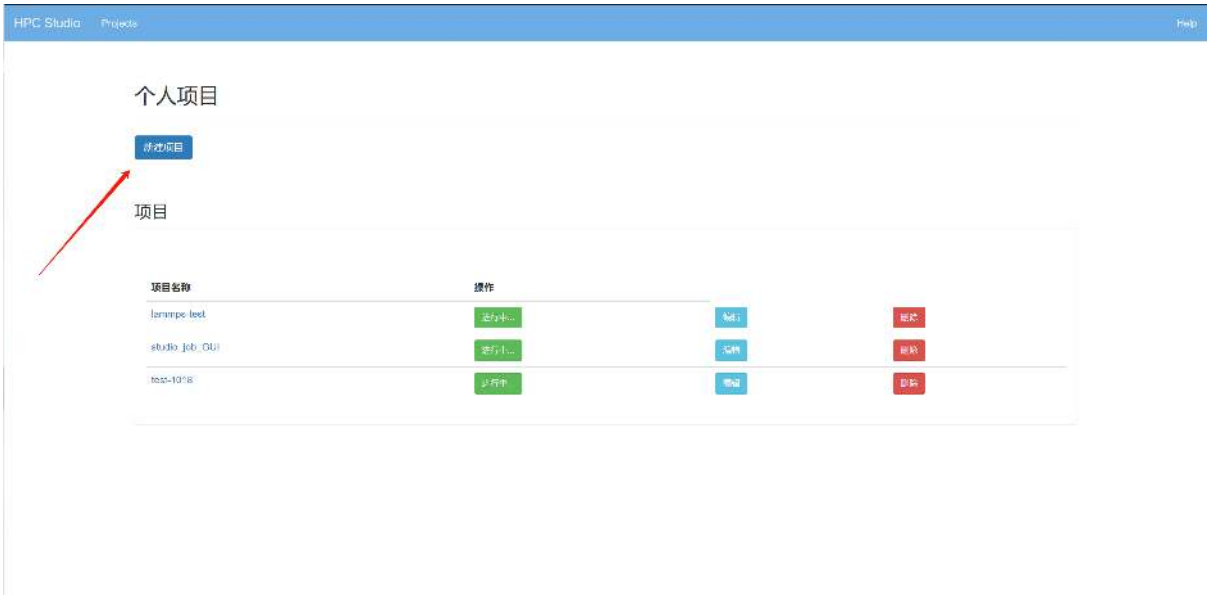
- PVGeo 官方配置教程 https://pvgeo.org/overview/getting-started.html。

4.4.6 AlphaFold-GUI

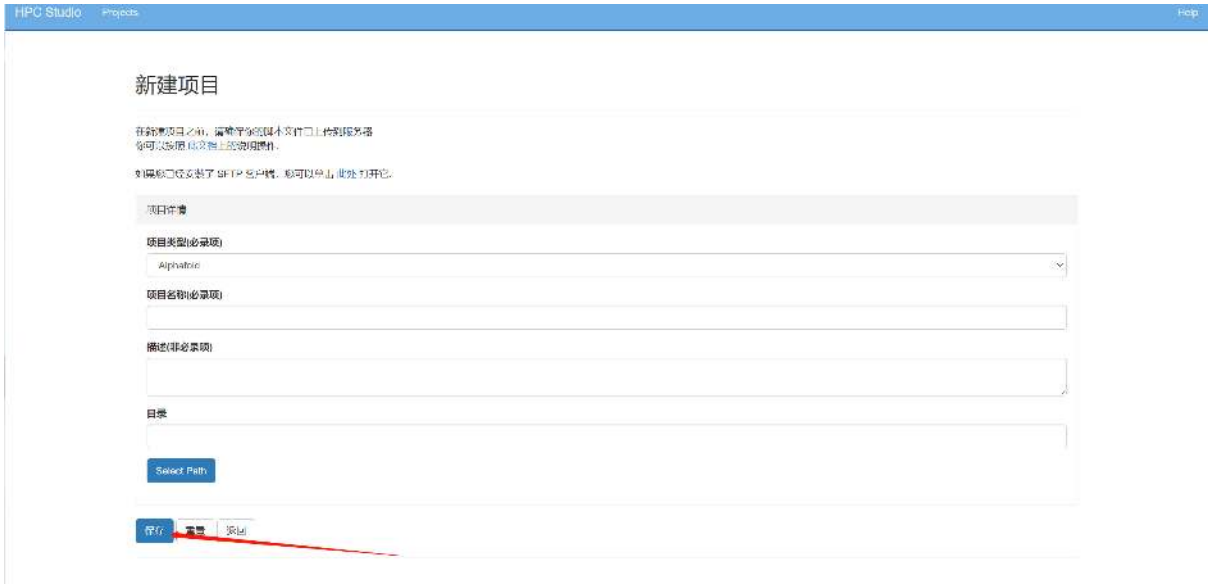
登录 HPC Studio 平台后，在自定义软件中选择 AlphaFold，如下图



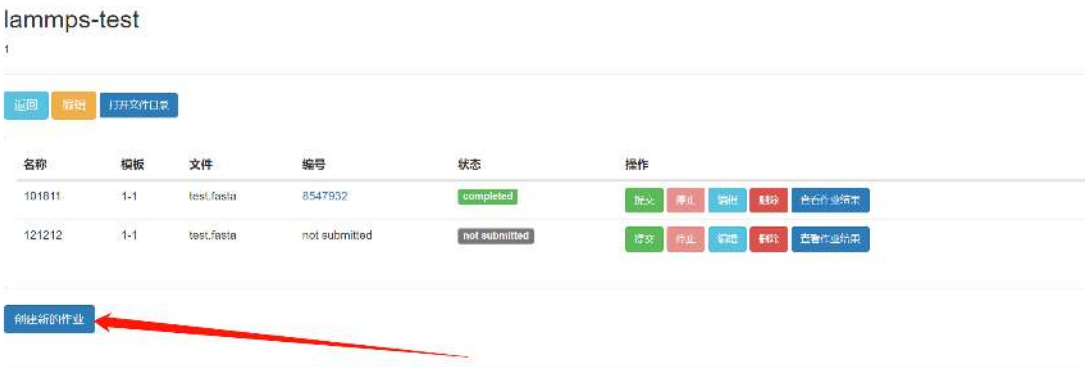
点击后会出现下图的选项卡，里面定义好了所需的选项卡，可以创建个人项目，(注: 项目下可以创建多个作业，作业运行结果保存在该项目路径下)



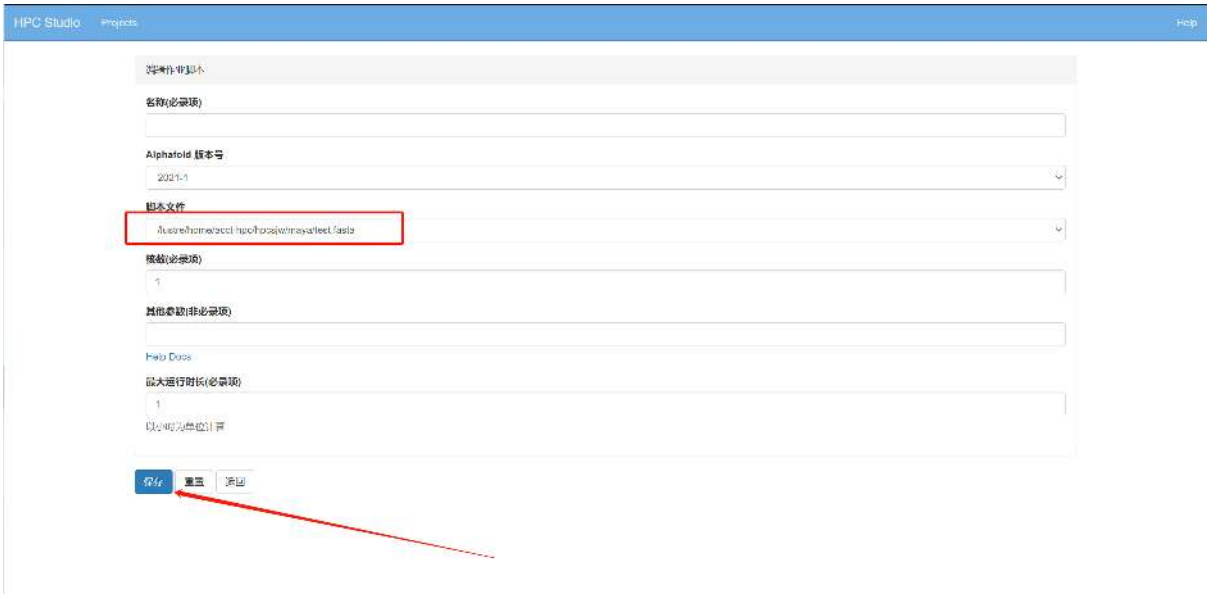
创建项目选项卡如下图所示，类型选择 AlphaFold，按要求填写表项点击”保存“



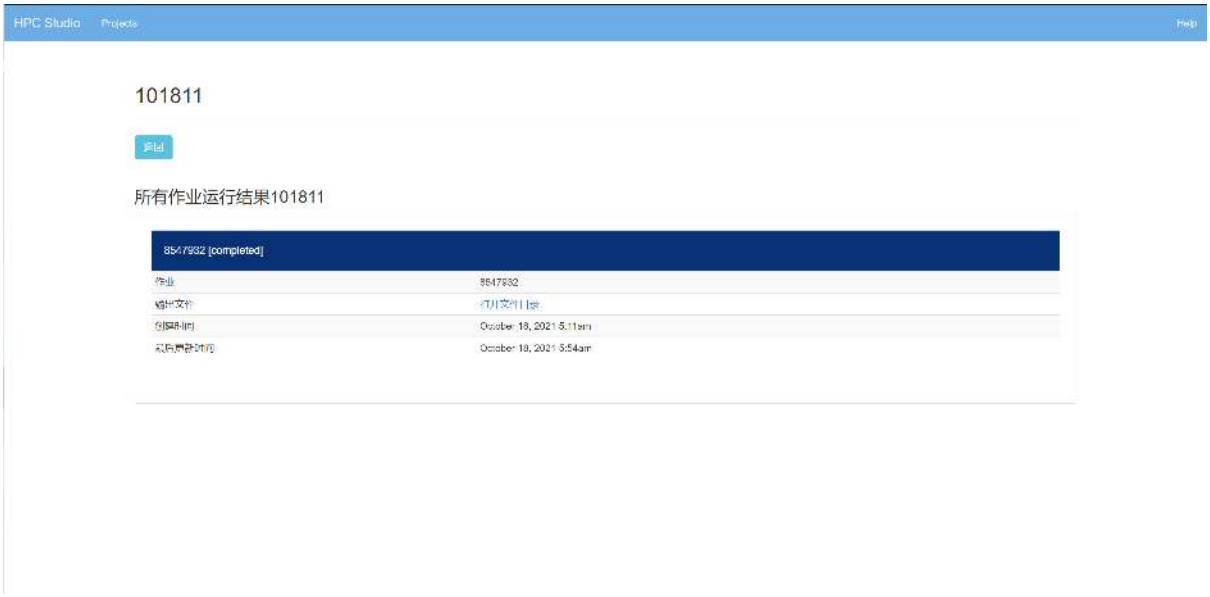
点击创建好的项目，选择“创建新的作业”按钮，如下图:



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：



参考资料

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

4.4.7 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`。

4.4.8 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

版本	平台	构建方式	名称
2022a	cpu	容器	/lustre/share/img/matlab_latest.sif
2022a	cpu	容器	/dssg/share/imgs/matlab/matlab_latest.sif 思源
2022b	cpu	容器	/lustre/share/img/matlab_r2022b.sif
2022b	cpu	容器	/dssg/share/imgs/matlab/matlab_r2022b.sif 思源
2021a	cpu	容器	/lustre/share/img/matlab_r2021a.sif
2021a	cpu	容器	/dssg/share/imgs/matlab/matlab_r2021a.sif 思源

超算上的 MATLAB

超算上的 CPU 及 GPU 平台均支持 MATLAB 软件，在 π 超算及思源一号均有提供。

超算上的 MATLAB 授权由网络授权服务器自动检查，超算用户无需用户名密码登录，打开即可使用。

MATLAB 既可被可视化调用（需启动 HPC Studio Desktop），也可从命令行调用。

- 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：

```
$ singularity run /dssg/share/imgs/matlab/matlab_r2022b.sif matlab
MATLAB is selecting SOFTWARE OPENGL rendering.

                            < M A T L A B (R) >
                  Copyright 1984-2022 The MathWorks, Inc.
               R2022b Update 1 (9.13.0.2080170) 64-bit (glnxa64)
                               September 28, 2022


To get started, type doc.
For product information, visit www.mathworks.com.

>>
```

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

1. 算例下载

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

2. 脚本提交

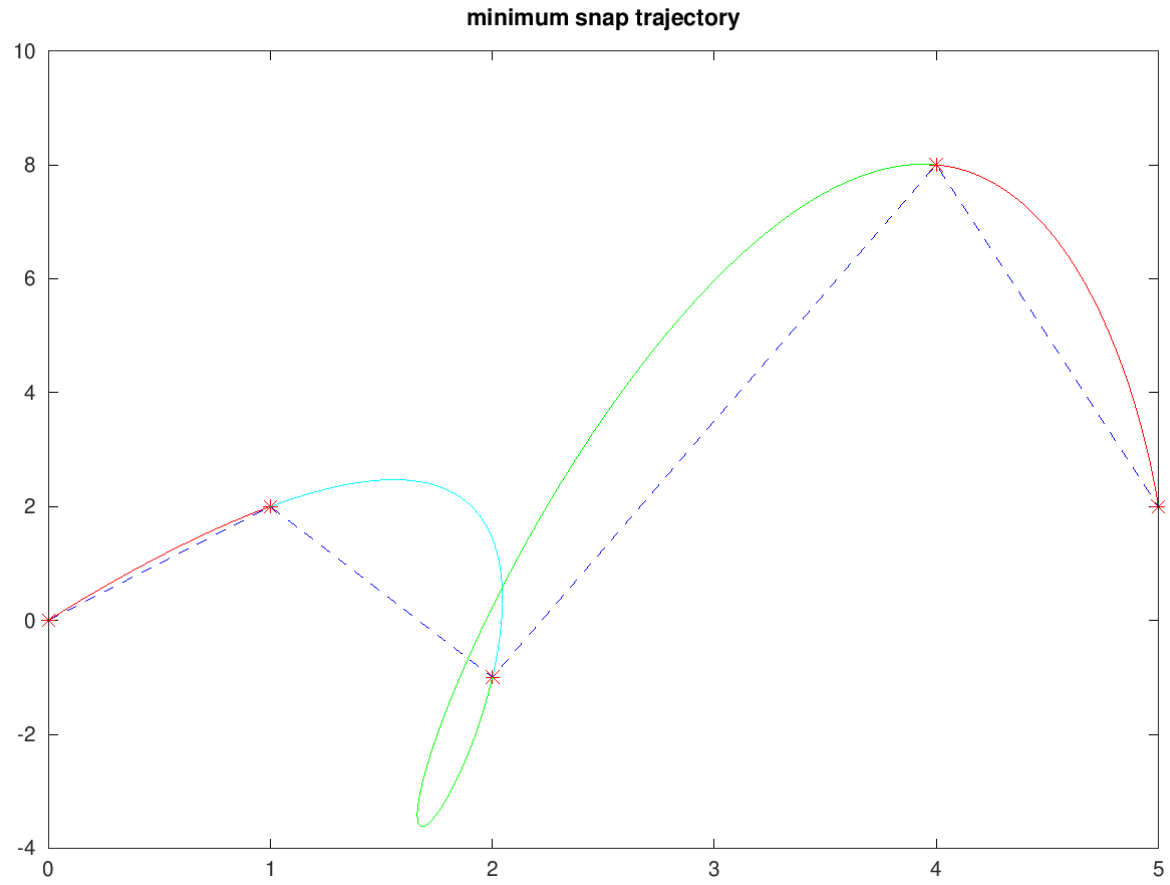
π 超算提交单核 CPU 脚本

```
#!/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

IMAGE_PATH=/lustre/share/img/matlab_r2022b.sif

ulimit -s unlimited
ulimit -l unlimited
cd ~/HPCTesting/matlab/case1
singularity exec $IMAGE_PATH 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

IMAGE_PATH=/dssg/share/imgs/matlab/matlab_r2022b.sif

ulimit -s unlimited
ulimit -l unlimited
cd ~/HPCTesting/matlab/case1
singularity exec $IMAGE_PATH 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 small
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -n 40
#SBATCH --cpus-per-task 1

IMAGE_PATH=/lustre/share/img/matlab_r2022b.sif

ulimit -s unlimited
ulimit -l unlimited
cd ~/HPCTesting/matlab/case2
singularity exec $IMAGE_PATH 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
```

(下页继续)

(续上页)

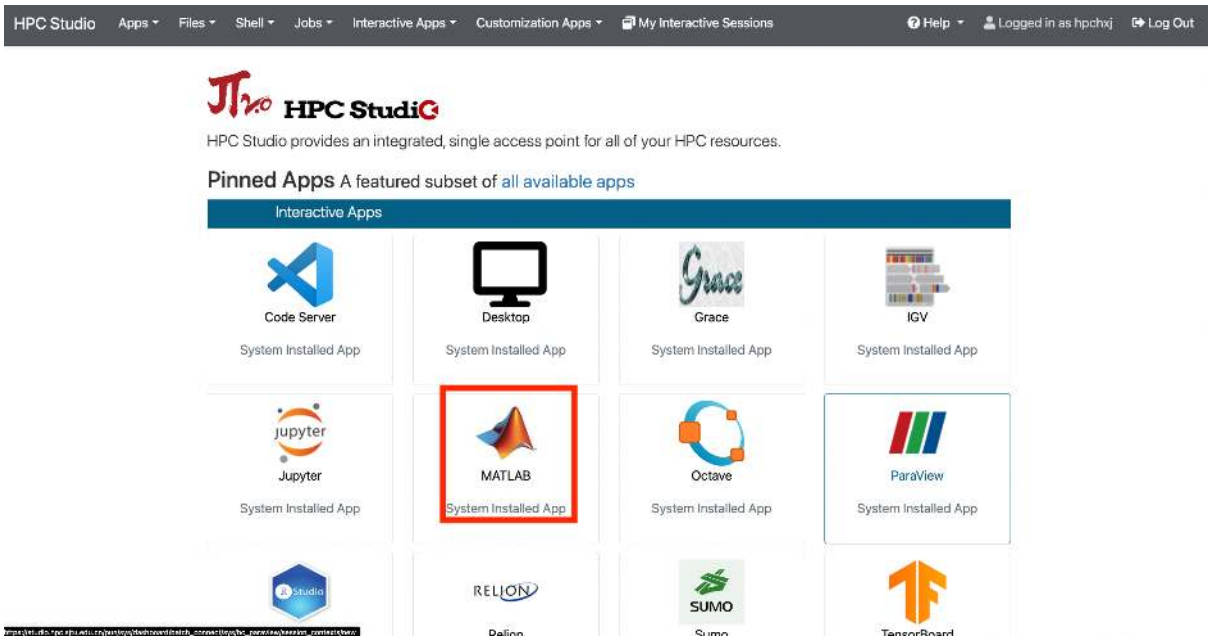
```
#SBATCH --cpus-per-task 64

IMAGE_PATH=/dssg/share/imgs/matlab/matlab_r2022b.sif

ulimit -s unlimited
ulimit -l unlimited
cd ~/HPCTesting/matlab/case2
singularity exec $IMAGE_PATH 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。

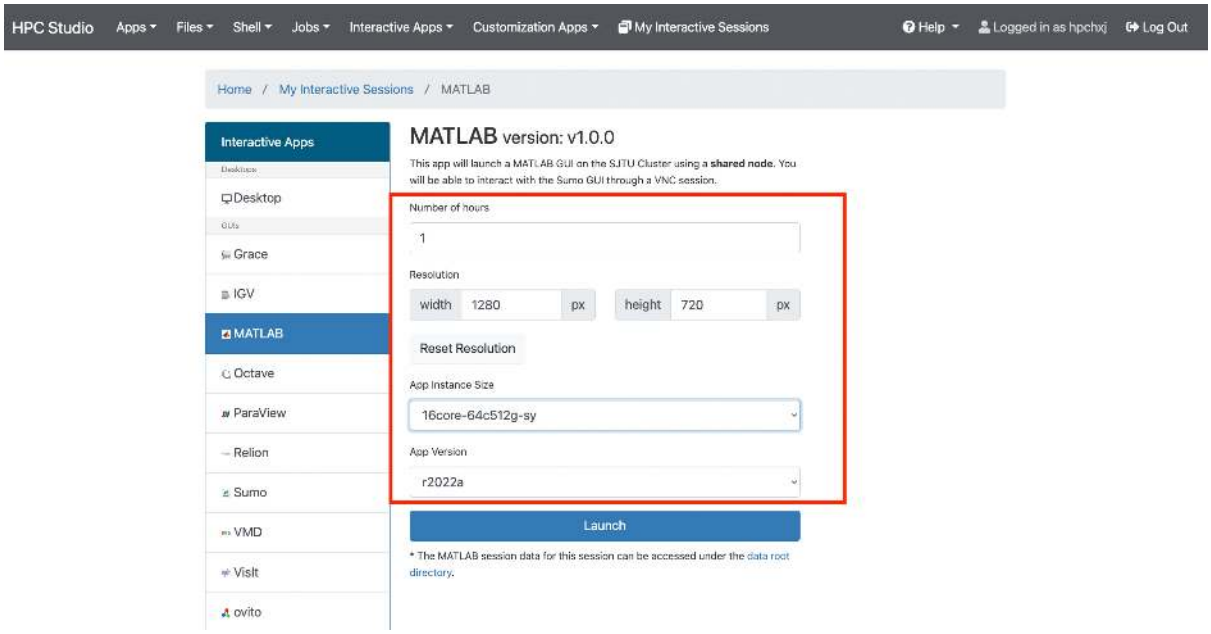
1. 登录可视化平台，选择 MATLAB 应用

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



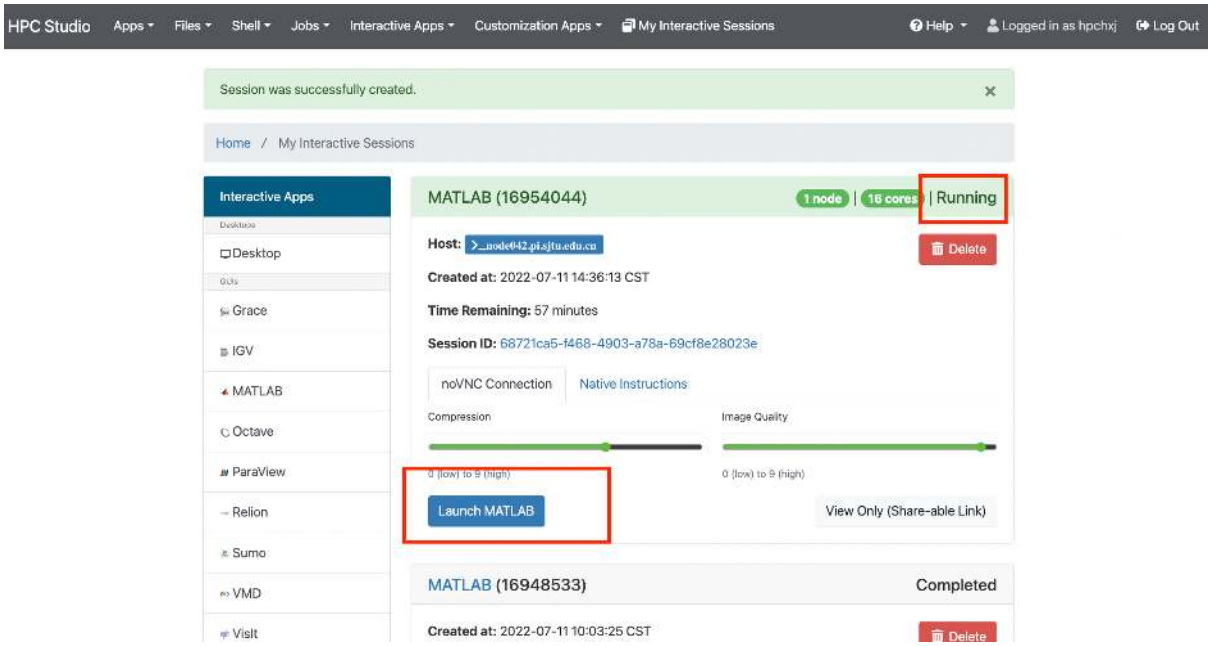
2. 申请资源，选择 MATLAB 版本

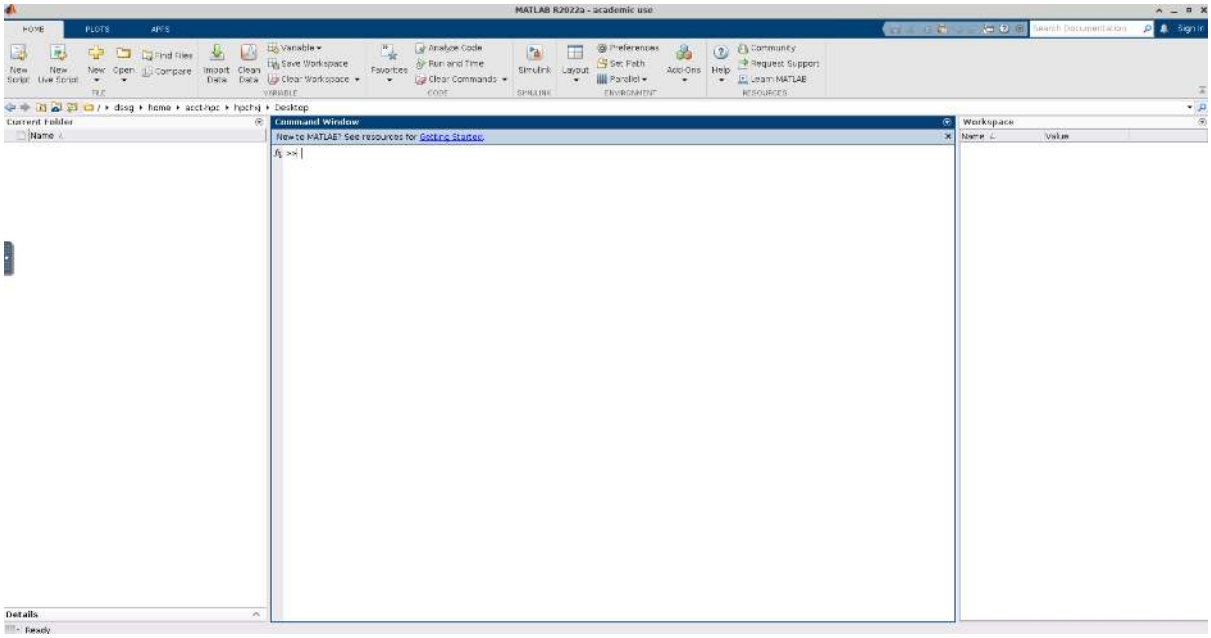
点击 MATLAB 图标后会跳转至资源选择界面，该页面上可选择申请的时长、可视化桌面的分辨率、平台资源以及 MATLAB 版本。



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。



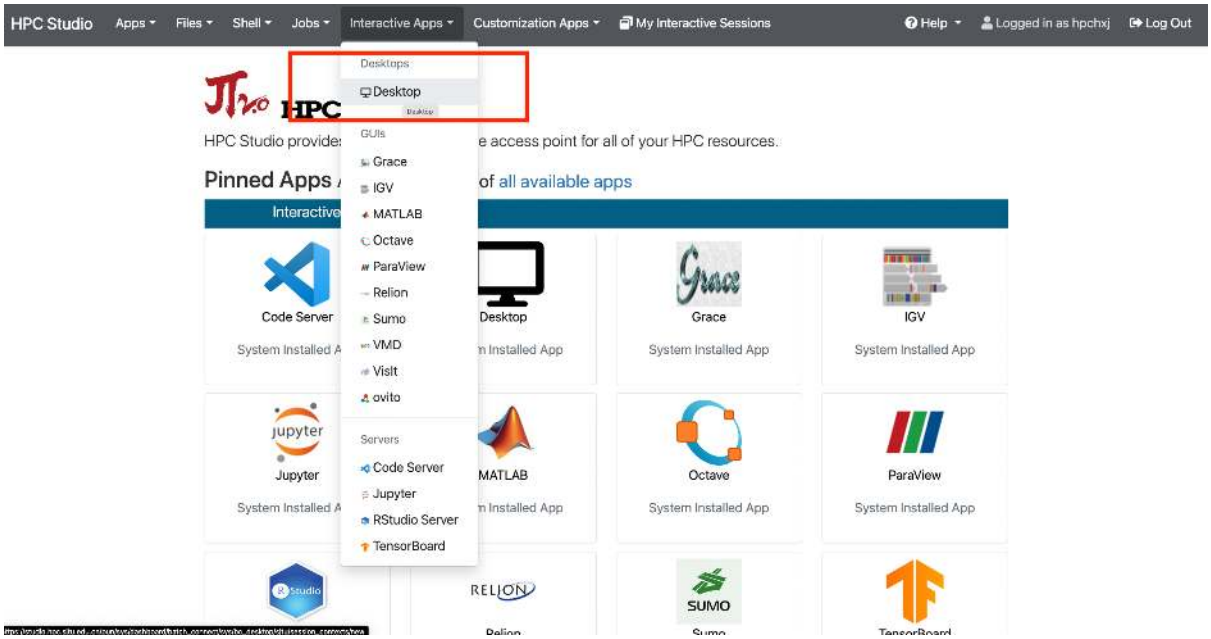


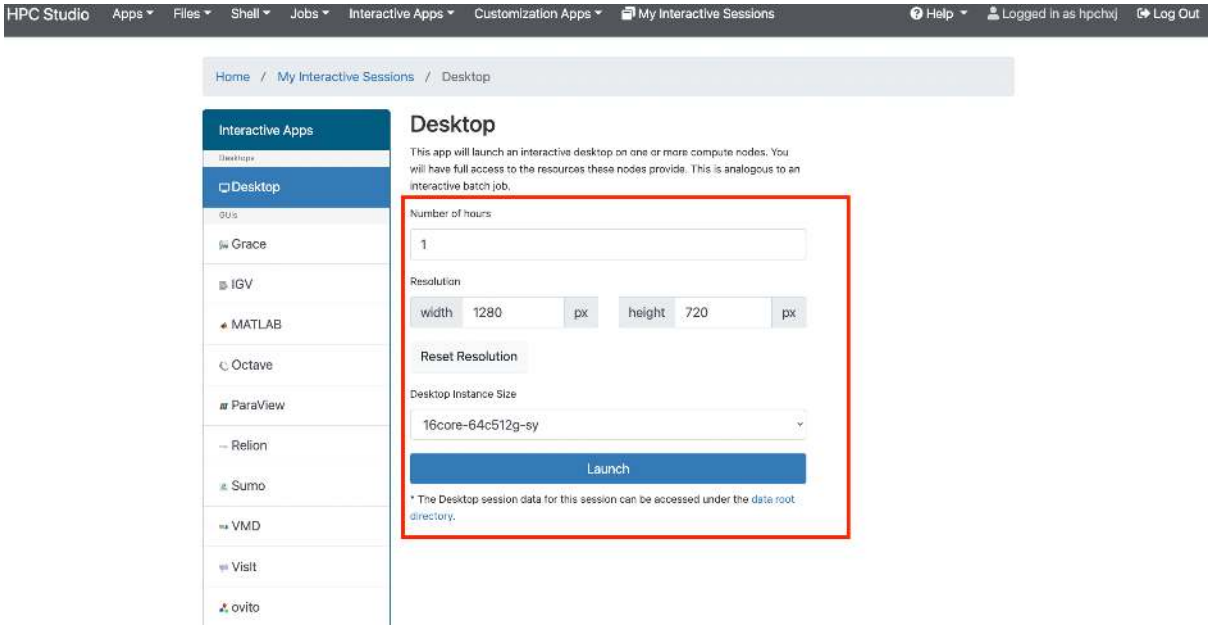
可视化平台桌面启动 **MATLAB**

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。

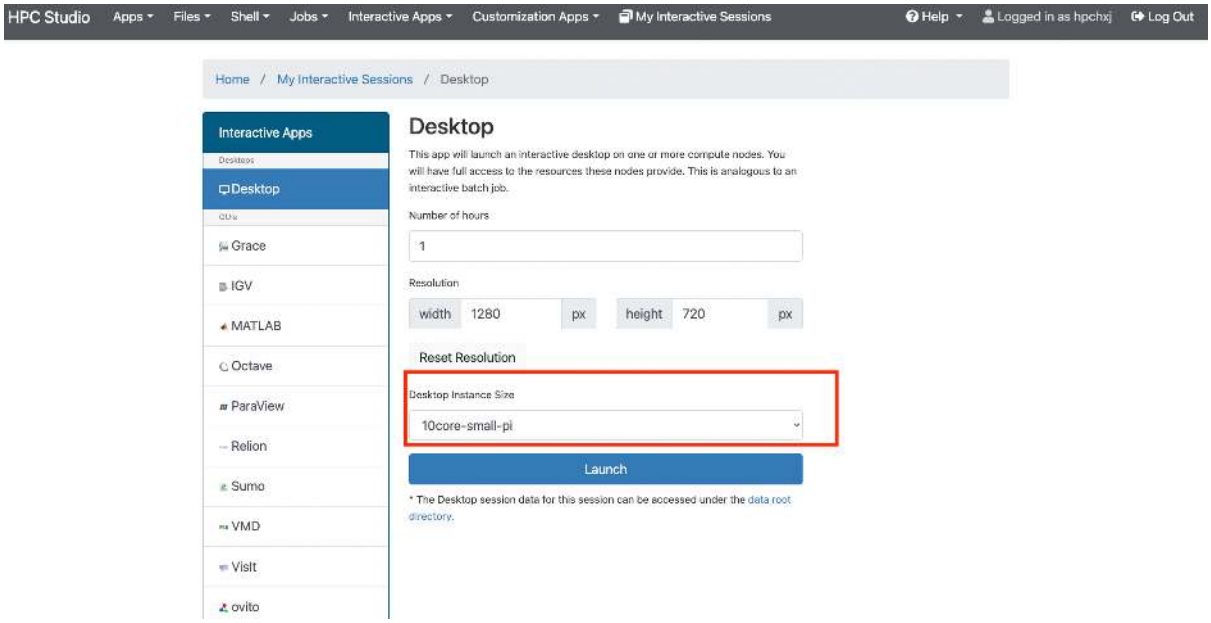
1. 启动远程桌面

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。



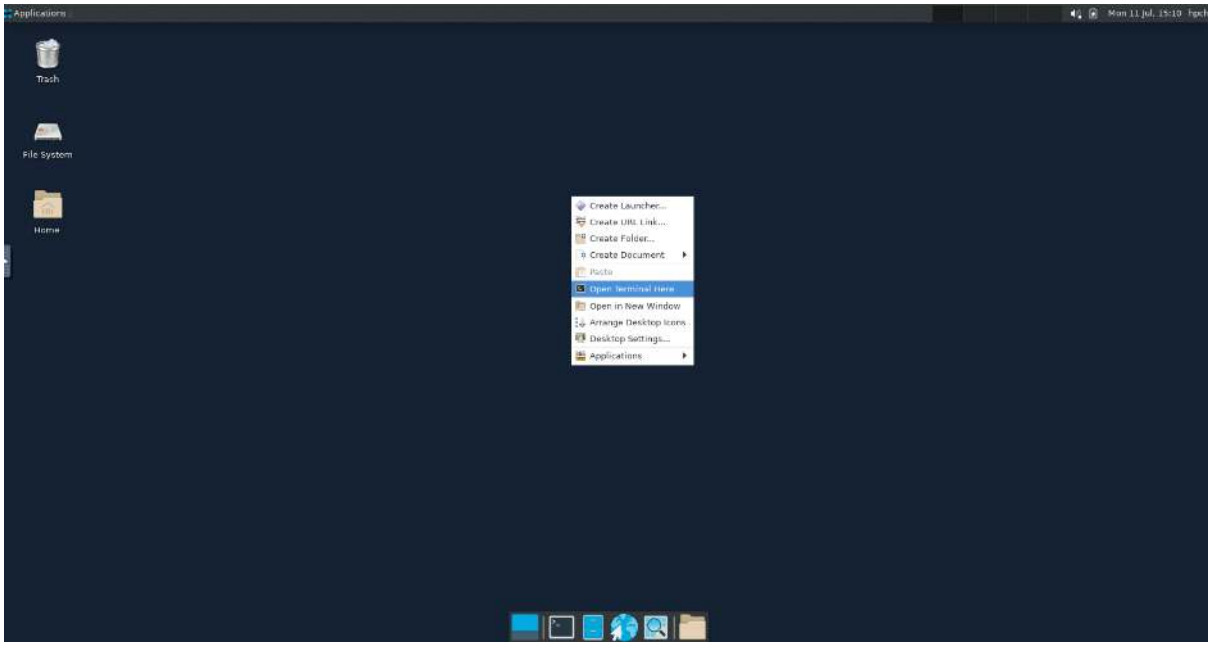


除了从思源一号启动远程桌面外，π 超算也支持启动远程桌面，在选定核数的同时可以同时选定平台：



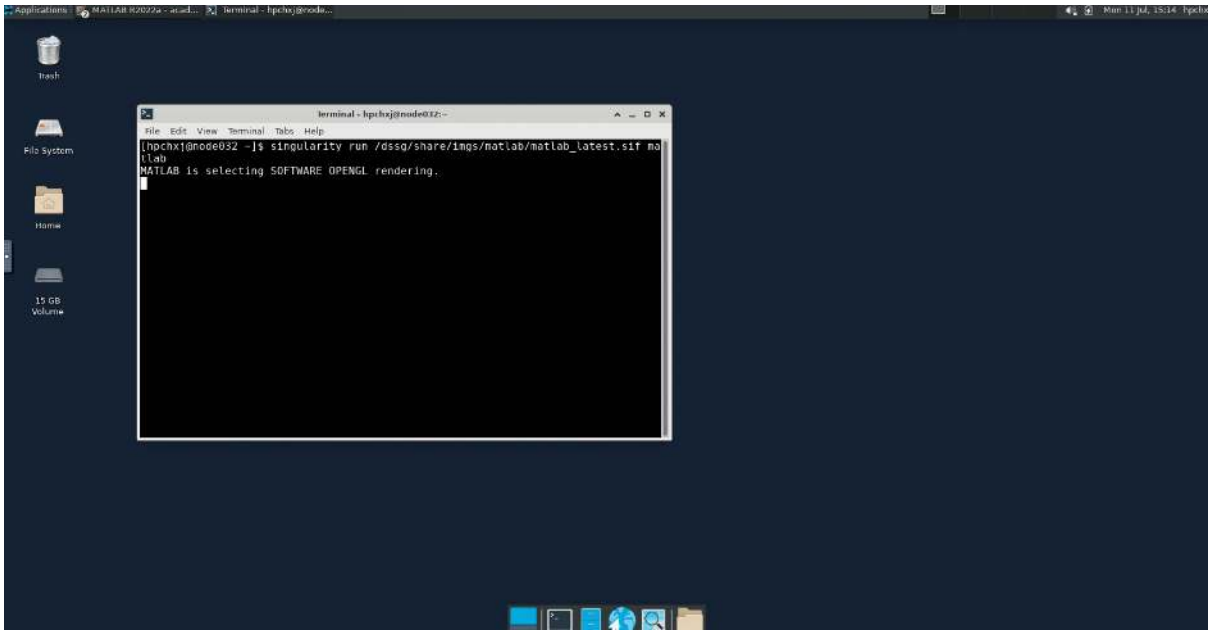
2. 启动 MATLAB

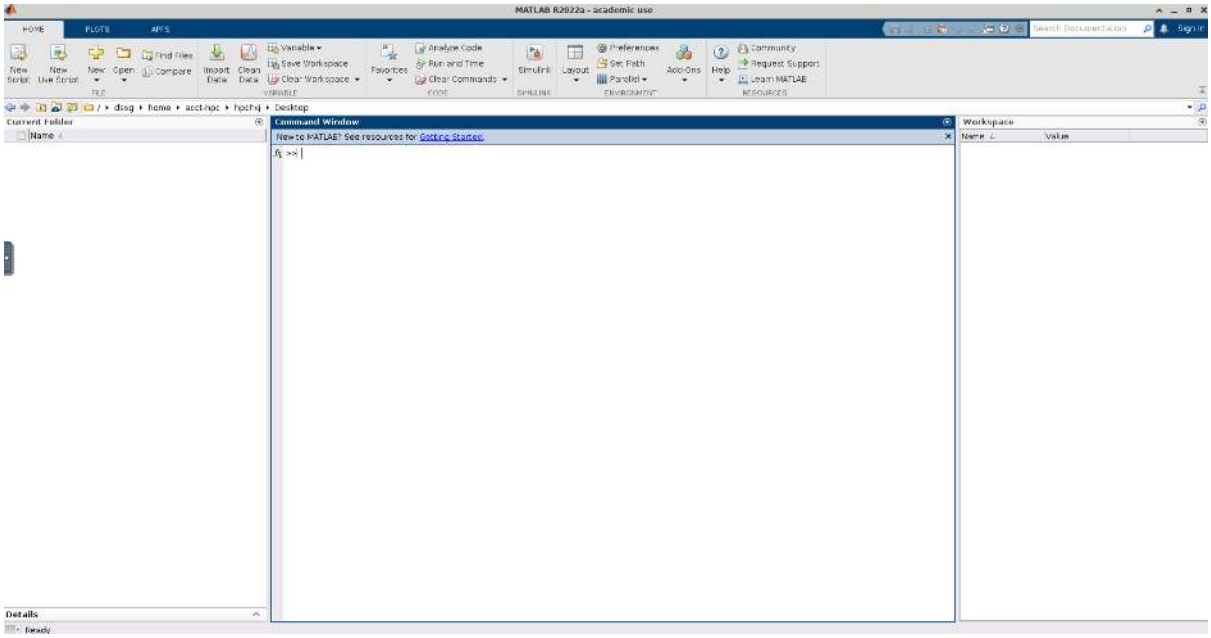
远程桌面中点击右键，选择 Open Terminal Here 打开终端。



在终端中使用命令 `singularity run /dssg/share/imgs/matlab/matlab_r2022b.sif matlab`，π 超算需使用命令 `singularity run /lustre/share/img/matlab_r2022b.sif matlab`。

启动后即可使用 MATLAB R2022b





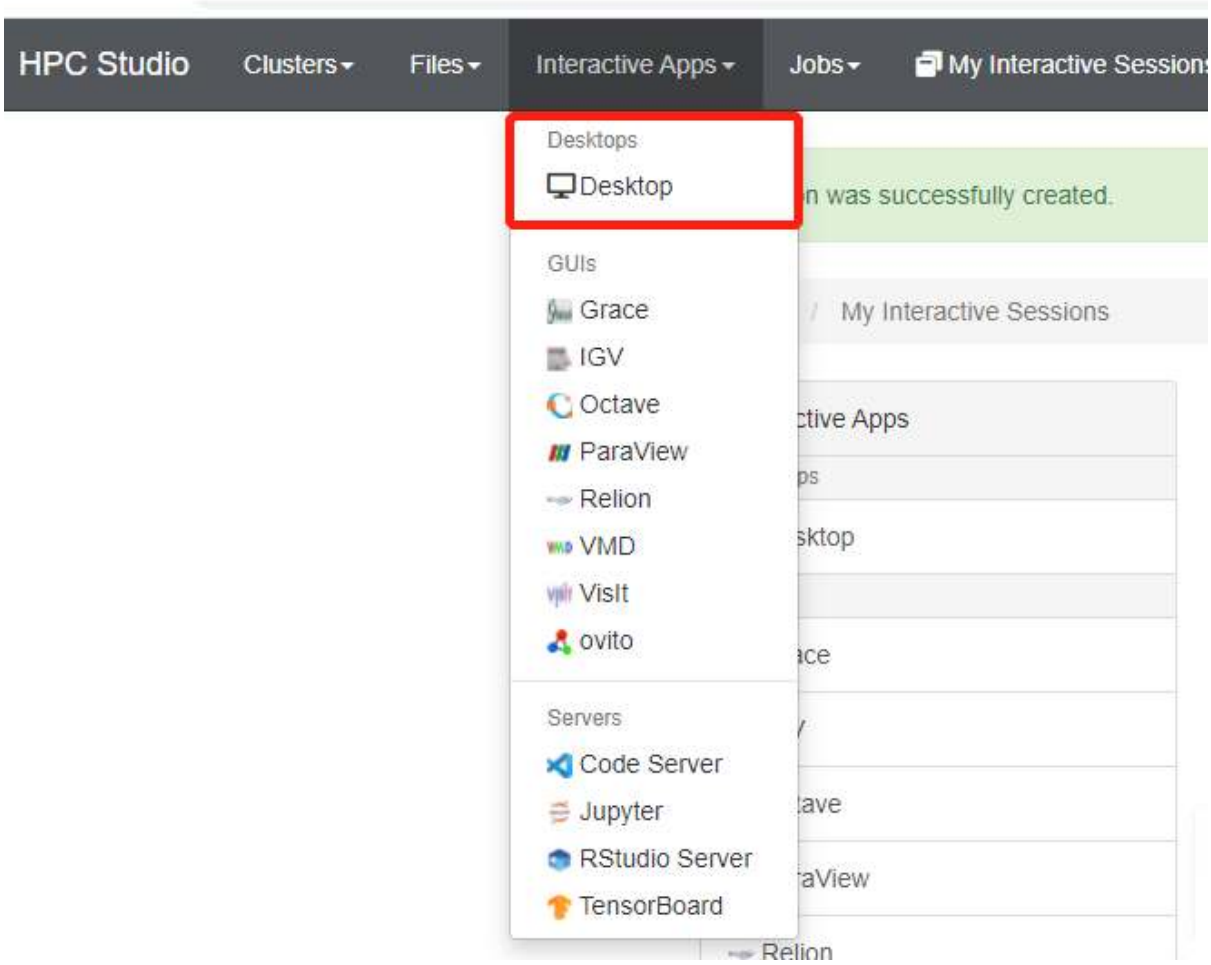
使用 GPU 版本的 MATLAB

使用 GPU 版本的 MATLAB 需要 CUDA11，因此该版本只能在思源一号使用。

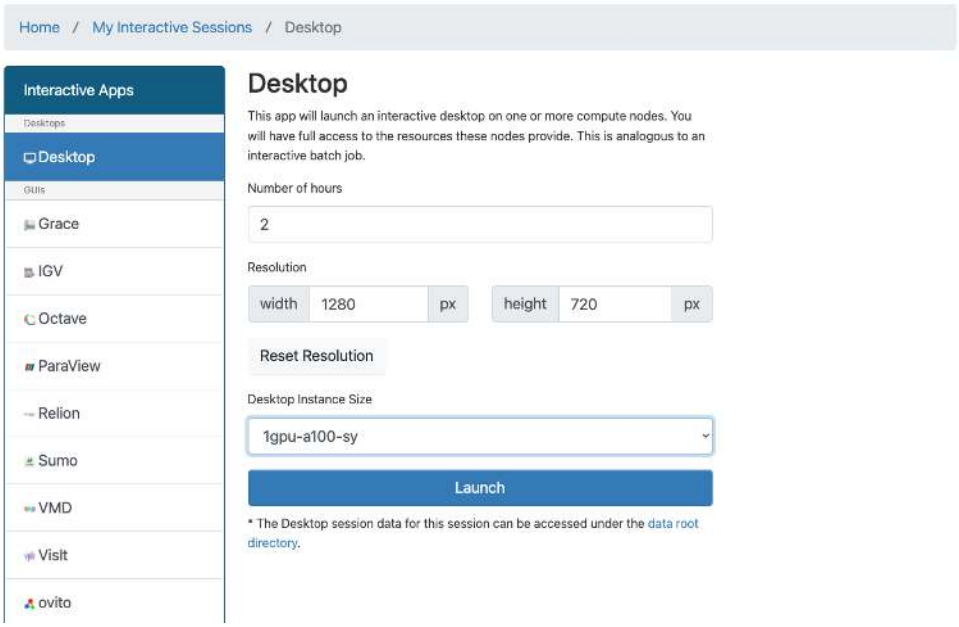
可视化平台使用 MATLAB GPU 版

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。

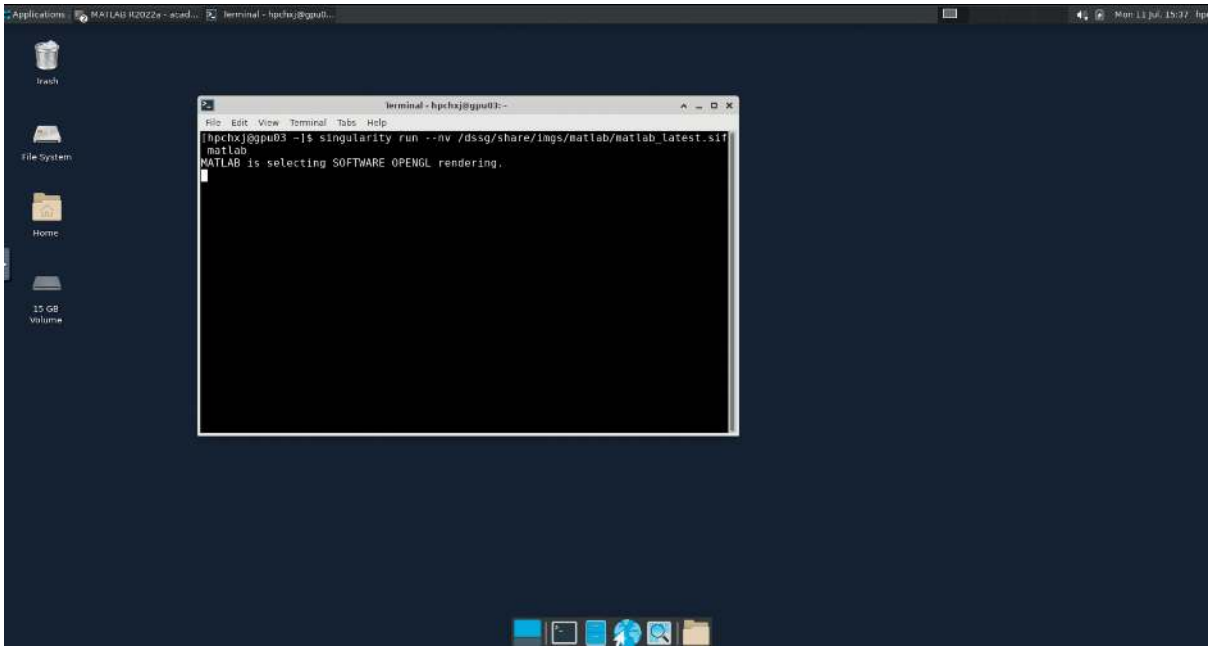


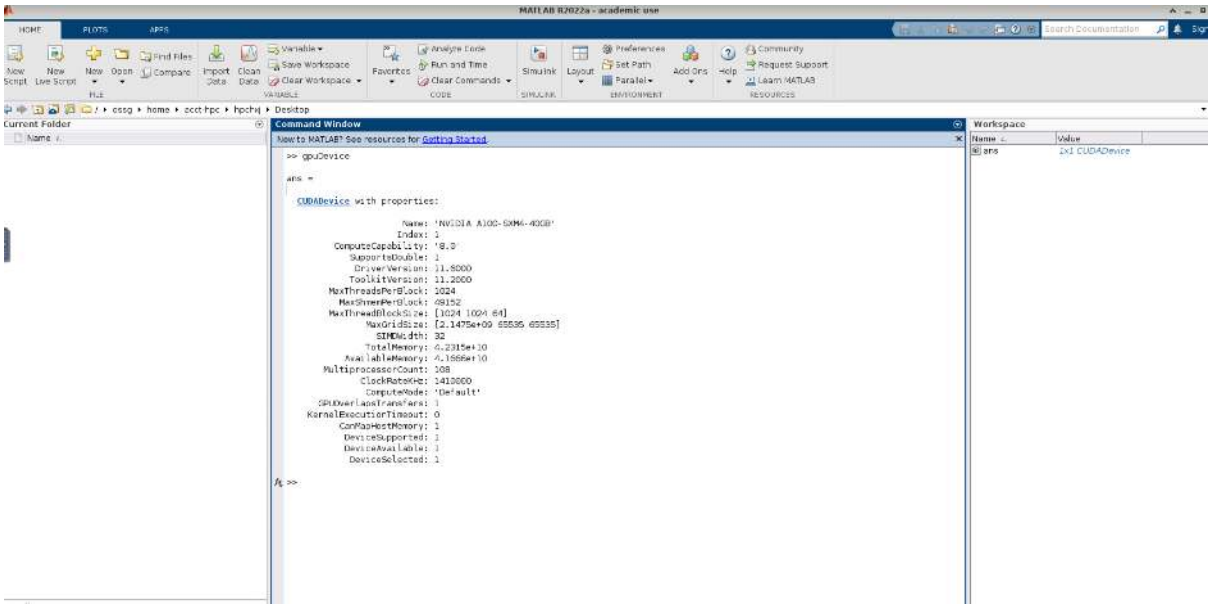
选定核数的时候选择思源一号的一张 GPU 卡：



2. 启动 GPU 版本 MATLAB

在窗口中启动终端（terminal），在终端输入 `singularity run --nv /dssg/share/imgs/matlab/matlab_r2022b.sif`，即可启动 GPU 版本 matlab。





提交 **MATLAB GPU** 版脚本

```
#!/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 6
```

(下页继续)

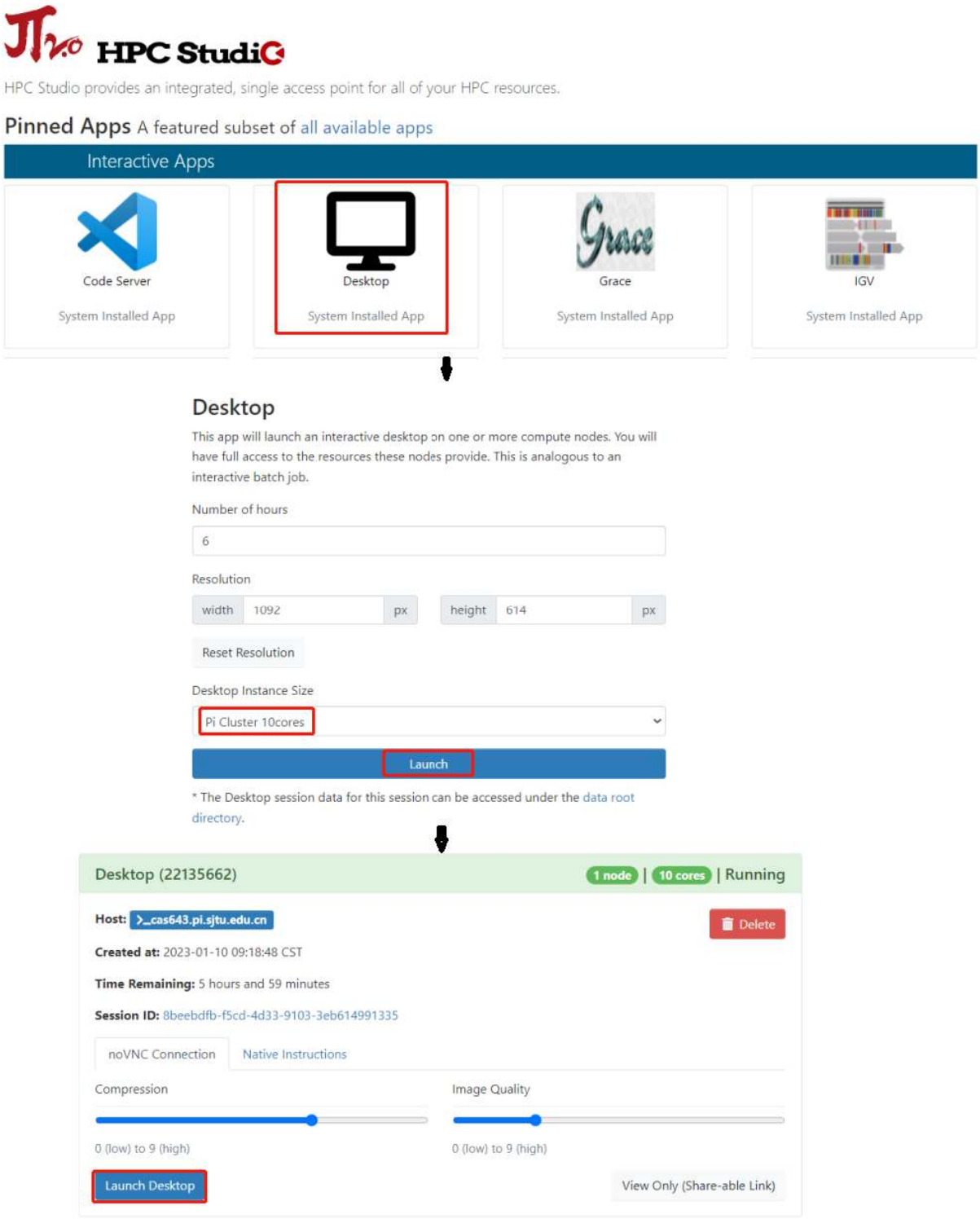
(续上页)

```
#SBATCH --gres gpu:1

IMAGE_PATH=/dssg/share/imgs/matlab/matlab_r2022b.sif

ulimit -s unlimited
ulimit -l unlimited

singularity run --nv $IMAGE_PATH 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 matlab 可视化终端




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

2. 然后，导入 SlurmProfile

输入如下命令:

```
profile_master = parallel.importProfile('/lustre/opt/contribute/
↪cascadelake/matlab/R2022a-new/ParSlurmProfile/SlurmParForUser.
↪mlsettings');
parallel.defaultClusterProfile(profile_master);
```

3. 接下来，运行作业

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。

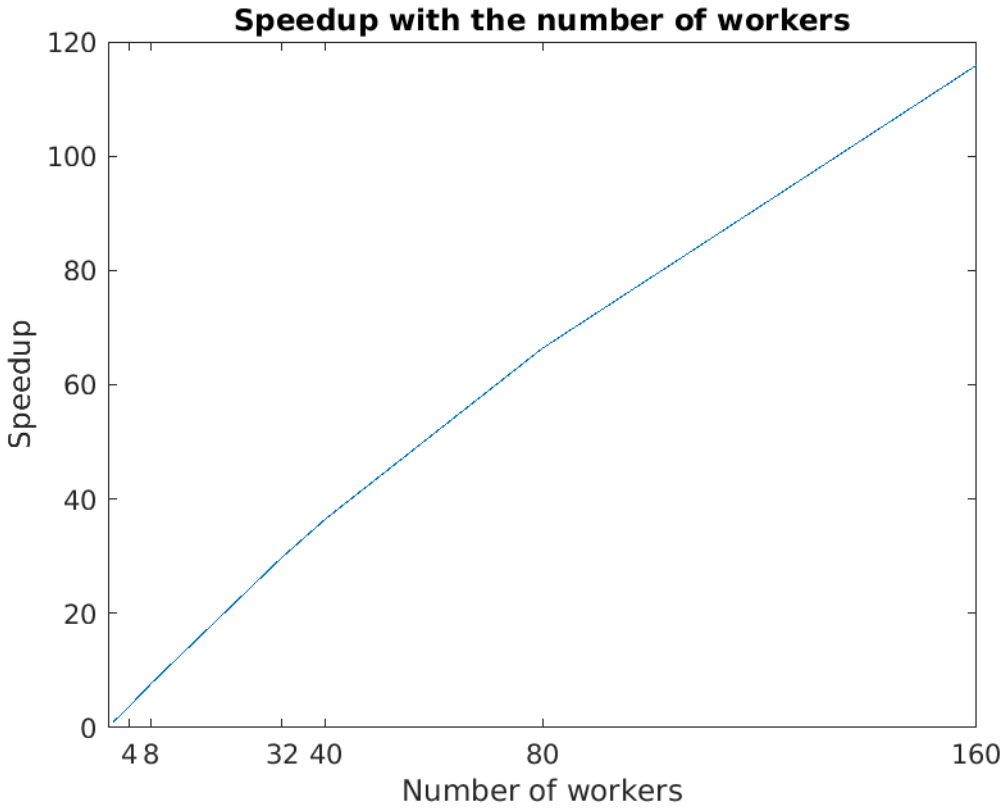
```
/lustre/share/samples/matlab/composite_speedup.m
```

输入：

```
composite_speedup
```

注：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。

4. 运行结果为



MATLAB Parallel Computing Toolbox

利用 Parallel Computing Toolbox™，可以使用多核处理器、GPU 和计算机集群来解决计算问题和数据密集型问题。利用并行 for 循环、特殊数组类型和并行化数值算法等高级别构造，无需进行 CUDA 或 MPI 编程即可对 MATLAB® 应用程序进行并行化。通过该工具箱可以使用 MATLAB 和其他工具箱中支持并行的函数。你可以将该工具箱与 Simulink 配合使用，并行运行一个模型的多个仿真。程序和模型可以在交互模式和批处理模式下运行。

集群上部署的 MATLAB 镜像均已安装 Parallel Computing Toolbox 并获取相关授权，打开 MATLAB 即可使用相应功能。

了解更多 MATLAB Parallel Computing Toolbox 在超算上的使用，请跳转至文档*MATLAB Parallel Computing Toolbox*.

单节点性能对比

算例为路径 `~/HPCTesting/matlab/case2` 。

运行时间

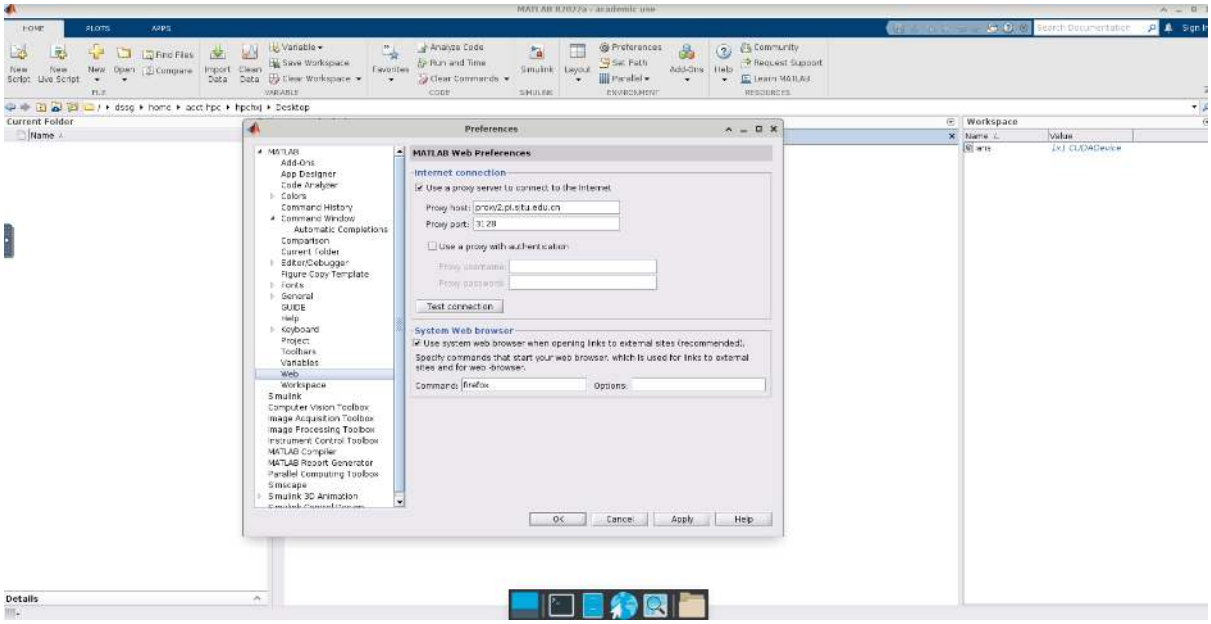
版本	平台	时间 (s)
2021a	思源	105
2021a	π 超算	176

建议

思源超算单节点拥有更多核心、更大内存。在运行多核心任务时推荐使用思源平台。

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`

自定义添加 MATLAB 插件

首先拷贝集群上的镜像到本地

```
cp /lustre/share/img/matlab_latest.sif ~/
```

接下来需要在镜像中添加基础编译环境（该操作可以在 build@container-x86 中操作）

```
Bootstrap:localimage
From:/home/singularity/matlab_latest.sif

%post
    echo y | apt-get update -y
    echo y | apt-get install gcc -y
    echo y | apt-get install g++ -y
```

最后在添加自定义的库时，需要先进入容器

```
singularity shell matlab_latest_self.sif
```

4.5 登录

获取交我算帐号后，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。

4.5.1 通过 SSH 登录集群

本文主要内容：

- 使用 SSH 登录集群的注意事项；
- 首次登录准备，如信息采集、客户端下载、SSH 登录、SSH 文件传输、无密码登录等；
- 故障排除和反馈。

注意事项

- 交我算帐号仅限于同一课题组的成员使用，请勿将帐号借给他人使用。
- 请妥善保管好您的帐号密码，不要告知他人。管理员不会要求您提供密码。
- 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。
- 登录集群后，请不要跳转到其他登录节点。任务完成后请关闭 SSH 会话。
- 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。

SSH 登录

帐号开通后您会收到含有以下内容的邮件，包含帐号用户名和密码用于登录集群。登录集群有两种方式：SSH 命令行登录、SSH 客户端登录。

```
Username: YOUR_USERNAME
Password: YOUR_PASSWORD
```

登录方法一：命令行登录

- 思源一号

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

- π 2.0 和 AI 平台

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

- ARM 平台（限校内 IP，或使用 SJTU VPN）

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

说明：

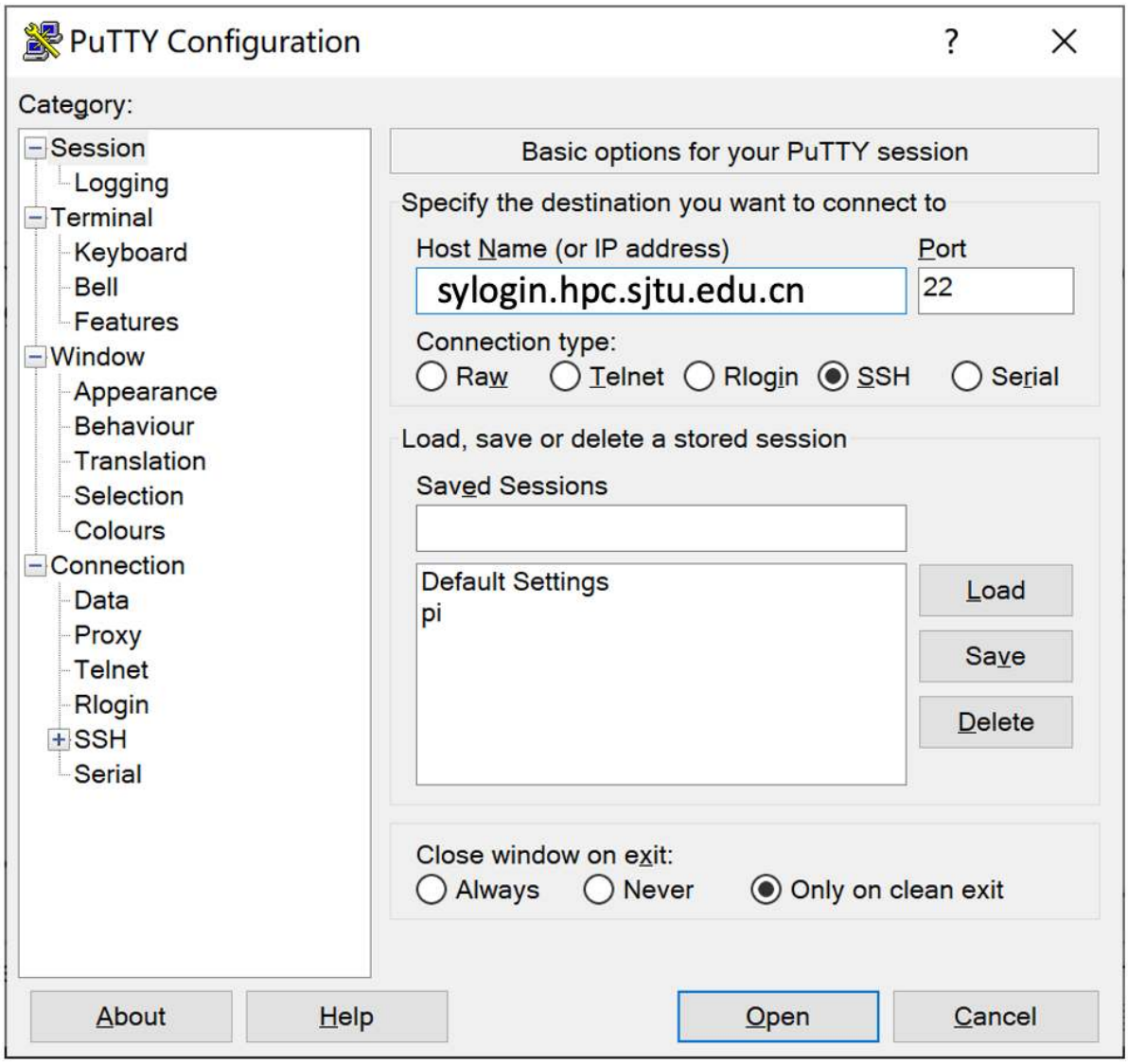
- 登录节点 IP 地址（或主机名）分别为 sylogin.hpc.sjtu.edu.cn（思源一号）、pilogin.hpc.sjtu.edu.cn（π 2.0 和 AI)、armlogin.hpc.sjtu.edu.cn（ARM）
- 通过上述命令登录，会自动分配到多个登录节点之一
- SSH 端口均为默认值 22

登录方法二：客户端登录

Windows 推荐使用 Putty 免费客户端，下载后双击即可运行使用。可至 Putty 官网下载。Linux / Unix / Mac 操作系统拥有自己的 SSH 客户端，包括 ssh, scp, sftp 等。

下面介绍 Windows 用户使用 Putty 客户端登录思源一号。

启动客户端 Putty，填写登录节点地址 sylogin.hpc.sjtu.edu.cn（适用于思源一号），或 pilogin.hpc.sjtu.edu.cn （适用于 π 2.0 和 AI 集群），端口号采用默认值 22，然后点 Open 按钮，如下图所示：



在终端窗口中，输入您的 SSH 用户名和密码进行登录：

提示：输入密码时，不显示字符，请照常进行操作，然后按回车键登录。

通过 SSH 传输文件

登录节点资源有限，不推荐在登录节点直接进行大批量的数据传输。交我算 HPC+AI 平台提供了专门用于数据传输的节点，登录该节点后可以通过 rsync，scp 等方式将个人目录下的数据下载到本地，或者反向上传本地数据到个人目录。详情请参考具体请参考数据传输 。

无密码登录

提示：“无密码登录”仅适用于使用 *SSH* 命令行工具的 *Linux/ UNIX / Mac* 用户

“无密码登录”使您无需输入用户名和密码即可登录，它还可以作为服务器的别名来简化使用。无密码登录需要建立从远程主机（集群的登录节点）到本地主机（您自己的计算机）的 SSH 信任关系。建立信任关系后，双方将通过 SSH 密钥对进行身份验证。

首先，您需要在本地主机上生成的 SSH 密钥对。为安全起见，集群要求使用密码短语 (passphrase) 来保护密钥对。使用密码短语来保护密钥对，每次双方身份验证时都需要输入密码。

```
$ ssh-keygen -t rsa
```

接下来屏幕会显示：

```
Generating public/private rsa key pair.
Enter file in which to save the key (/XXX/XXX/.ssh/id_rsa):    #␣
↪存储地址，默认回车即可
```

(下页继续)

(续上页)

```
Enter passphrase (empty for no passphrase):                    #␣
↪请设置密码短语，并记住。输入的时候屏幕无显示
Enter same passphrase again:                                   #␣
↪再输入一遍密码短语
```

在无密码短语的情况下，您的私钥未经加密就存储在您的硬盘上，任何人拿到您的私钥都可以随意的访问对应的 SSH 服务器。

ssh-keygen 将在 ~/.ssh 中生成一个密钥对，包含两个文件：id_rsa(需保留的私钥)，和 id_rsa.pub 可作为您的身份发送的公钥)。然后，使用 ssh-copy-id 将本地主机的公钥 id_rsa.pub 添加到远程主机的信任列表中。实际上，ssh-copy-id 所做的就是将 id_rsa.pub 的内容添加到远程主机的文件 ~/.ssh/authorized_keys 中。

```
（在自己电脑上）$ ssh-copy-id YOUR_USERNAME@TARGET_IP
```

若手动自行在服务器上添加 authorized_keys 文件，需确保 authorized_keys 文件的权限为 600：

```
（在集群上）$ chmod 600 ~/.ssh/authorized_keys
```

我们还可以将连接参数写入 ~/.ssh/config 中，以使其简洁明了。新建或编辑文件 ~/.ssh/config：

```
$ EDIT ~/.ssh/config
```

还需分配以下内容：主机分配远程主机的别名，主机名是远程主机的真实域名或 IP 地址，端口分配 SSH 端口，用户分配 SSH 用户名。

```
Host hpc
HostName TARGET_IP
User YOUR_USERNAME
```

您需要确保此文件的权限正确：

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

然后，您只需输入以下内容即可登录集群：

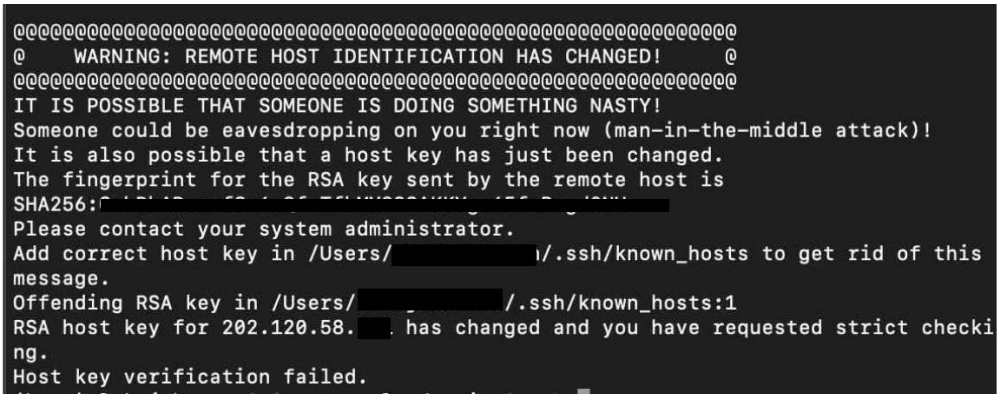
```
$ ssh hpc
```

当 *SSH* 密钥对发生泄漏，请立即清理本地电脑*.ssh* 文件夹里的密钥对，并重新在本地生成密钥对（生成时请设置密码短语）。另外请删除集群上的 *~/.ssh/authorized_keys* 文件。

如何重新生成密钥对

```
（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



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，您的帐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，请使用或抄送帐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，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。

4.5.2 Tmux

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。

Tmux 是什么？

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行程序很有用。

（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
```

(下页继续)

(续上页)

```
# 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。

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 $`。

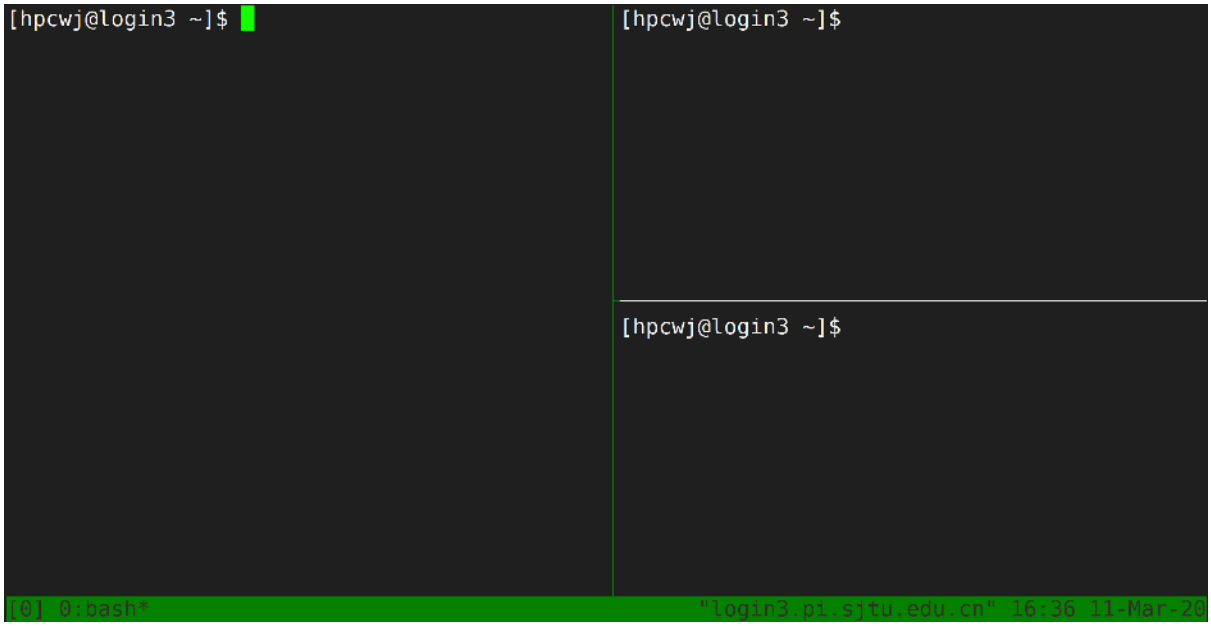
窗格（**window**）操作

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。
```

(下页继续)

(续上页)

```
$ 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。
```

4.5.3 VS Code

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。

使用 VS Code 连接集群

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。配置 VS Code 进行远程开发主要分为 4 个步骤：

1. 在本地电脑安装兼容的 SSH 客户端；
2. 配置 SSH 免密登录集群；
3. 在本地电脑安装 VS Code 客户端并安装相应插件；
4. 使用 VS Code 远程访问；

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。

SSH 免密登录集群

关于 SSH 免密登录的配置请参考*ssh 免密登录*。

一个可参考的 *~/.ssh/config* 文件内容如下：

```
Host x86
  HostName pilogin.hpc.sjtu.edu.cn
  User $YOUR_USERNAME
  Port 22

Host arm
  HostName kplogin1.hpc.sjtu.edu.cn
  User $YOUR_USERNAME
  Port 22


Host siyuan
  HostName sylogin.hpc.sjtu.edu.cn
  User $YOUR_USERNAME
  Port 22
```

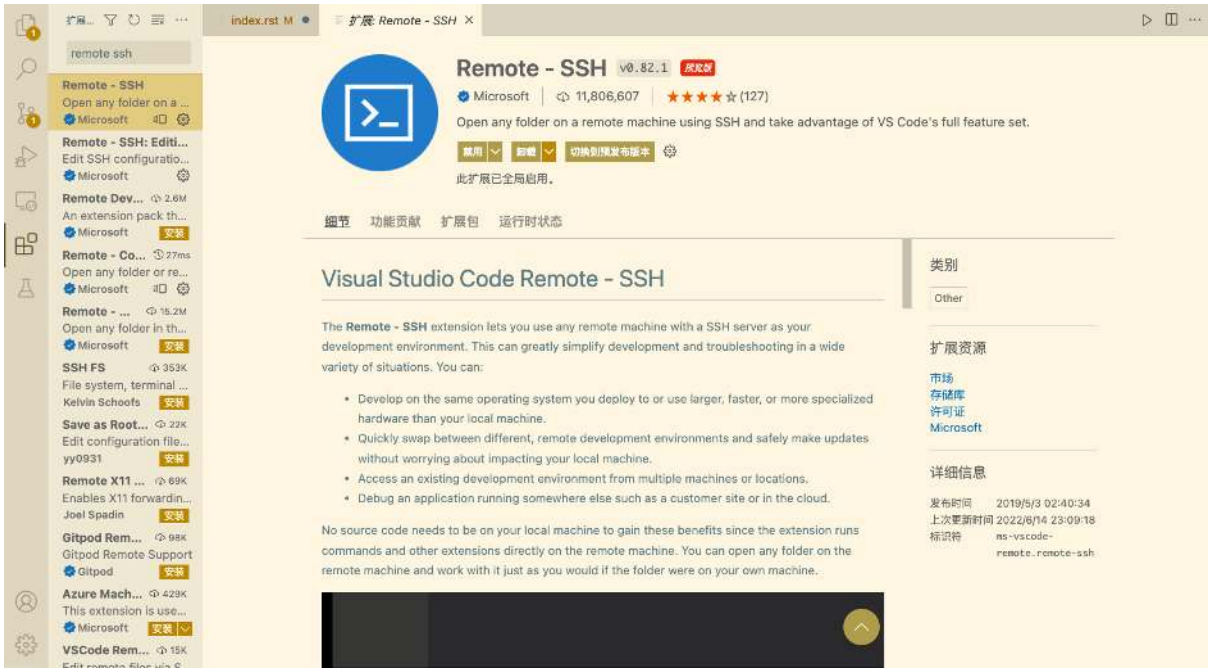
配置完毕后请在本地终端测试是否能访问集群。

```
$ ssh siyuan
Enter passphrase for key '/Users/YOUR_HOME/.ssh/id_rsa':
Last failed login: Wed Jun 22 18:34:38 CST 2022 from xxx.xxx.xxx.xx␣
 ↪on ssh:notty
There were 2 failed login attempts since the last successful login.
Last login: Wed Jun 22 18:28:52 2022 from xxx.xxx.xxx.xx
```

本地安装 VS Code 及插件

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。



使用 **VS Code** 访问集群

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：

4.5.4 X Server

图形视窗系统

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。

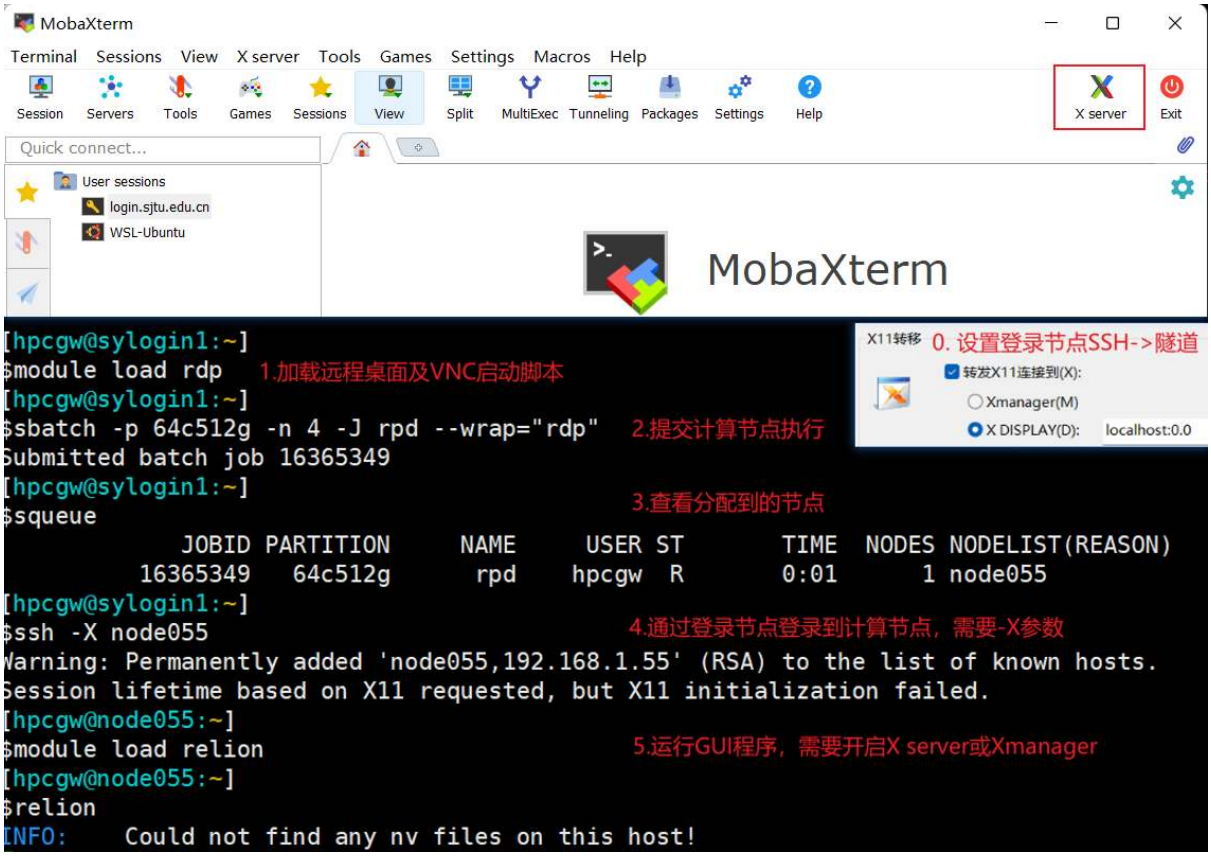
使用 X Server 显示图形界面

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
squeue                                             # 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。

4.5.5 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

4.6 数据传输

目前交我算 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，应当在 `login, kplogin, data` 系列节点上查看、使用存放的数据；**gpfs** 存储用于 `64c512g, a100` 以及相应的 debug 队列，应当在 `sylogin, sydata` 节点上查看、使用存放的数据。使用时请注意自己 home 目录的实际路径。

推荐使用 data 节点进行数据传输，不占用 login 节点资源并且多进程或多用户同时传输不会受限于 CPU。**data** 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、校验数据等。如果发现此类行为，中心将视情况取消相关帐号使用传输节点的权利。

4.6.1 传输节点

思源一号 data 节点地址为 sydata.hpc.sjtu.edu.cn

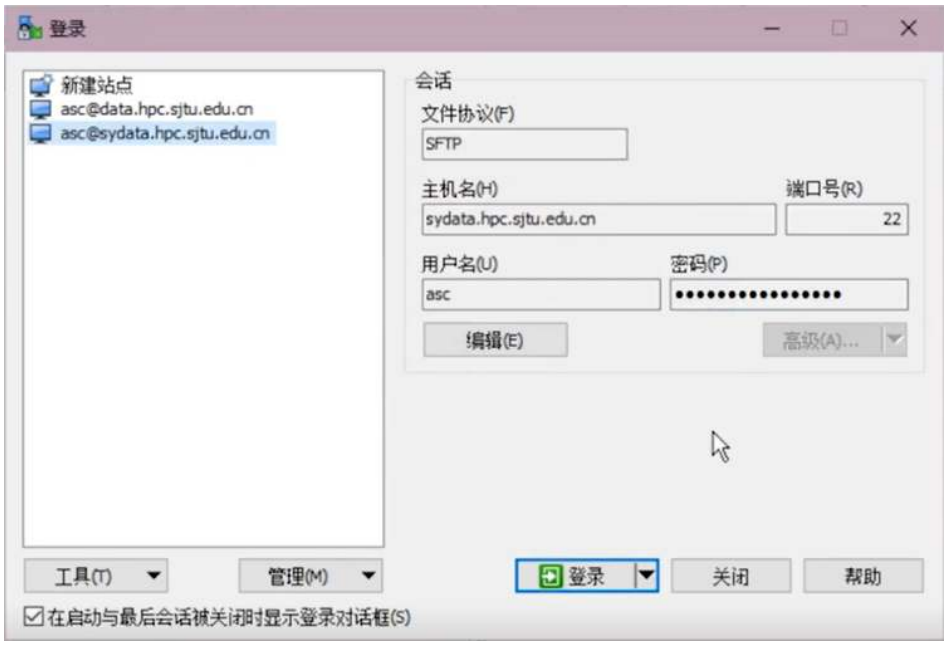
π 2.0/AI/ARM 集群 data 节点地址为 data.hpc.sjtu.edu.cn

4.6.2 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/
```

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

4.6.4 思源一号与 π 2.0/AI/ARM 集群互传

如果是在 lustre 和 dssg 直接跨存储池搬运数据，可以任选 data 或者 sydata 节点发起传输。例如通过登录 π2.0 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 π2.0 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
$ scp -r /lustre/home/acct-exp/expuser01/data/ expuser01@sydata.hpc.
↪sjtu.edu.cn:/dssg/home/acct-exp/expuser01/data/
```

4.6.5 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, sydata.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。

4.6.6 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
```

(下页继续)

(续上页)

```
$ 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。请参考无密码登录 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。

4.6.7 提高数据传输速度的技巧 (从 lustre 目录到 archive 目录)

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/archive/home/acct-hpc/
↪expuser01/sif/目录下
ssh expuser01@data.hpc.sjtu.edu.cn
find /lustre/home/acct-hpc/expuser01/img -type f | xargs -P 5 -I {}␣
↪scp {} hpc@data.hpc.sjtu.edu.cn:/archive/home/acct-hpc/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-hpc/expuser01/img/* hpchgc@data.hpc.sjtu.edu.cn:/archive/home/acct-hpc/expuser01/sif/`

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

进程数	时间
1	10m17s
5	3m47s

4.6.8 通过 sshfs 挂载家目录

```
# 假设用户expuser01在π 2.0集群上个人目录为/lustre/home/acct-exp/
↪expuser01 ; 思源一号的家目录为/dssg/home/acct-exp/expuser01

# 示例1: 在linux系统环境挂载π 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/ /mountpoint

#␣
↪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/ /mountpoint

# 示例3: 在windows系统环境挂载π 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: http
↪/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』
```

4.6.9 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

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

4.7 作业

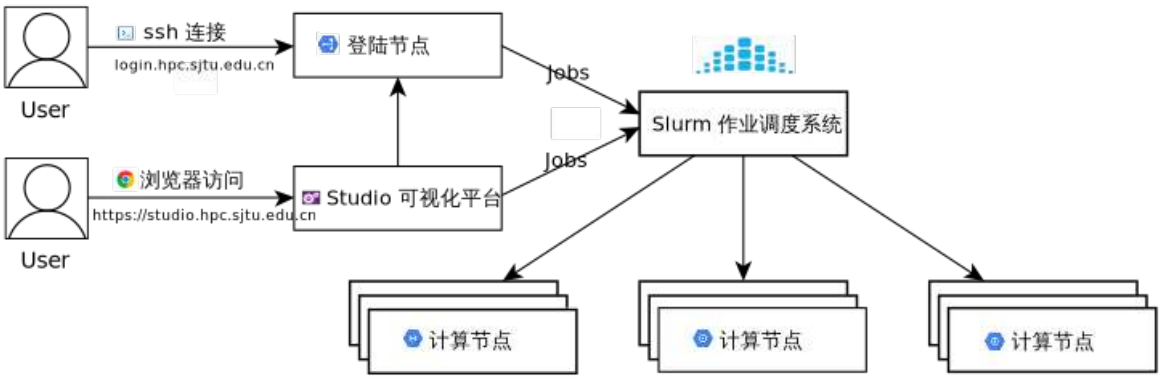
交我算 HPC+AI 平台通过 Slurm 调度系统运行作业。

4.7.1 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)。

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
64c512g	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

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 40~24000，每核配比 4G 内存，节点需独占使用；单节点配置为 40 核，192G 内存
huge	允许单作业 CPU 核数为 6~80，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
192c6t	允许单作业 CPU 核数为 48~192，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
small	允许单作业 CPU 核数为 1~20，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
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

ARM 集群设置以下队列，使用限制与说明如下

队列名	说明
arm128c256g	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

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   ZXLing     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_address]	通知邮箱
--nodelist=[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
```

(下页继续)

(续上页)

```
#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

#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
```

srun 和 salloc 交互式作业

srun 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。

```
$ srun -N 1 -n 4 -p small hostname
cas006
```

启动远程主机 bash 终端：

```
srun -p small -n 4 --exclusive --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 small
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

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
```

查看更多的信息：

```
$ 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)

4.7.2 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
```

集群队列介绍

集群上现有 64c512g, a100, small, cpu, huge, 192c6t, dgx2 和 arm128c256g 队列。

`scontrol show partition` 查看集群队列介绍

`sinfo` 查看集群资源实时状态

队列名	说明
64c512g	允许使用 CPU 核数为 1~60000，每核配比 8G 内存，节点可共享使用；单节点配置为 64 核，512G 内存
a100	允许单作业 GPU 卡数为 1~92，每卡配比 CPU 上限为 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
small	允许使用 CPU 核数为 1~20，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
cpu	允许使用 CPU 核数为 40~24000，每核配比 4G 内存，节点需独占使用；单节点配置为 40 核，192G 内存。用户需独占节点，用满 40 核，或使用部分核心的时候加上 `--exclusive` 命令
huge	允许使用 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
192c6t	允许使用 CPU 核数为 96~192，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	允许使用 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
arm128c256g	允许使用 CPU 核数为 1~12800，每核配比 2G 内存，节点可共享使用；单节点配置为 128 核，256G 内存

64c512g, a100, small, cpu, dgx2 队列允许的作业运行最长时间为 7 天, arm128c256g 为 3 天, huge 和 192c6t 为 2 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

各队列作业示例

下面根据不同队列，示例 slurm 作业脚本

small

small 队列 slurm 脚本示例

```
#!/bin/bash

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

cpu

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
```

cpu 队列 slurm 脚本示例：单节点（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：单节点（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
```

(下页继续)

(续上页)

```
#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（核数小于 40, cpu␣
↪队列必须加上此命令）
```

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
```

(下页继续)

(续上页)

```
#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 lammps

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
#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
```

(下页继续)

(续上页)

```
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=small            # small 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 --output=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
```

(下页继续)

(续上页)

```
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

small 队列 slurm 脚本示例：邮件提醒

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=small
#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
```

4.7.3 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 */
```

(下页继续)

(续上页)

```
    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=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
./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=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 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>

#define MAX_HOSTNAME_LENGTH 256

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

    int numprocs;
```

(下页继续)

(续上页)

```
    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
```

(下页继续)

(续上页)

```
#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
export I_MPI_FABRICS=shm:ofi

srun ./mpihello
```

最后，将作业提交到 SLURM

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

ulimit -s unlimited
```

(下页继续)

(续上页)

```
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);
```

(下页继续)

(续上页)

```
    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, 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
```

(下页继续)

(续上页)

```
#SBATCH --gres=gpu:1

ulimit -s unlimited
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
```

4.7.4 AI 平台使用文档

交大 AI 计算平台是国内高校计算力能最强的人工智能计算平台。AI 平台拥有两种 GPU 计算资源：

- 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：Pi 上的 *dgx2* 队列含 DGX-2 8 台，每台 DGX-2 配备 16 块 NVIDIA Tesla V100，每块 V100 默认配置 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 使用方法（作业提交模式、交互模式）及 GPU 利用率查看方法。

A100 使用

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

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 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*。

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。

投放到此队列的作业 运行时间最长为 **20** 分钟，超时后会被终止。

调试作业脚本示例：

```
#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=debuga100
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=25
#SBATCH --gres=gpu:5              #最多28gpu卡
```

(下页继续)

(续上页)

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

module load gcc cuda

./cudaTensorCoreGemm
```

DGX-2 使用

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
#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

4.7.5 ARM 使用文档

ARM 超算基于 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，完美融入现有超算系统。

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。

ARM 超算作业提交方式与 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 超算与 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 超算登录

- ARM 超算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 超算上部署了首批主流计算软件。后续将会推出更多的适用 ARM 超算运行的应用。

- 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 超算软件编译

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 超算的计算节点，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。

4.7.6 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；

思源一号作业示例

以下是一个名为 `siyuan.slurm` 的 单节点作业脚本，该脚本申请 1 个思源一号 CPU 节点（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 ...
```

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。

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

`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 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

4.7.7 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*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
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
```

(下页继续)

(续上页)

```
Memory Utilized: 23.33 GB
Memory Efficiency: 14.58% of 160.00 GB
```

4.8 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。

您选择如下其中一种方法使用集群提供的 Singularity 容器镜像能力：

1. 使用 Singularity 加载集群预编译的镜像。
2. 使用 Singularity 拉取远端镜像。
3. 按需定制 Singularity 镜像。

4.8.1 使用 Singularity 加载集群预编译的镜像

π 集群拥有丰富的预编译镜像资源。针对不同的硬件架构，我们制作了不同的基础镜像与应用镜像。您可以访问我们的 Docker Hub 主页 查看已经制作的镜像。

目前我们在 **Docker Hub** 上开源的镜像仓库主要有：

- x86 平台基础镜像
- x86 平台应用镜像
- ARM 平台应用镜像

Singularity 可以从 Docker Hub(以 `docker://sjtuhpc/` 开头) 拉取 π 集群预编译镜像。如下命令从 Docker Hub 拉取预编译的 lammps 镜像，保存成名为 `lammps-intel-2020.sif` 的镜像文件：

```
$ unset XDG_RUNTIME_DIR
$ unset SINGULARITY_BIND
$ unset MODULEPATH
$ singularity pull lammps-intel-2020.sif docker://sjtuhpc/hpc-app-
↪container:lammps-intel-2020
```

查看生成的镜像文件

```
$ ls
lammps-intel-2020.sif
```

使用自己制作的镜像文件运行提交 lammps 作业：

```
#!/bin/bash
#SBATCH --job-name=lmp_test
#SBATCH --partition=cpu
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 2
#SBATCH --ntasks-per-node=40


ulimit -s unlimited
ulimit -l unlimited

module load gcc/9.3.0-gcc-4.8.5
module  load openmpi
mpirun -n 80 singularity run YOUR_IMAGE_PATH lmp -i YOUR_INPUT_FILE
```

小技巧：π 集群预编译基础镜像根据操作系统、GCC 版本、openmpi 版本等，有不同的版本。x86 平台预编译应用镜像以及 ARM 平台预编译应用镜像也拥有多个软件版本。请根据需要合理选择。

4.8.2 使用 Singularity 拉取远端镜像

您可以使用 `singularity pull` 拉取远端预编译的镜像，从而直接使用外部预预编译镜像仓库提供的丰富软件资源。Singularity 可以从 Docker Hub(以 `docker://` 开头)、Singularity Hub(以 `shub://` 开头) 等地址拉取镜像。如下命令从 Docker Hub 拉取 CentOS 8 标准镜像，保存成名为 `cento8.sif` 的镜像文件：

```
$ singularity pull centos8.sif docker://centos:centos8
```

查看生成的镜像文件

```
$ ls centos8.sif
centos8.sif
```

加载容器镜像，并且在容器环境中运行 `cat` 程序，查看容器内 `/etc/redhat-release` 文件的内容，然后在宿主环境中运行同样命令，对比结果：

```
$ singularity exec centos.sif cat /etc/redhat-release
CentOS Linux release 8.3.2011
$ cat /etc/redhat-release
CentOS Linux release 7.7.1908 (Core)
```

运行结果显示，我们成功在 CentOS 7 操作系统上加载了一个 CentOS 8 容器镜像。

小技巧：Singularity 镜像文件 (Singularity Image File, sif) 是一种内容只读的文件格式，其文件内容不能被修改。

4.8.3 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` 下载基础操作系统镜像。

```
$ docker pull centos:8
```

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
```

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 mackcache
```

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␣
 ↪                    PORTS     NAMES
515e913f12cb   5d0da3dc9764   "/bin/bash"   4 seconds ago    Exited␣
 ↪(0) 2 seconds ago             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 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
```

4.8.4 通过 Definition File 构建 Singularity 镜像

Singularity 还可以使用“镜像定义文件”(Definition File) 描述镜像构建过程。镜像定义文是一个文本文件，描述了构建镜像使用的基本镜像、构建过程执行的命令，如软件包管理命令 `yum`, `apt-get` 等等。

小技巧：上一节交互式操作的命令操作过程，就是镜像定义文件的主要内容。

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
```

我们准备一个镜像定义文件 sample.def，这个定义文件使用 CentOS 8 为基本镜像，安装编译器、OpenMPI 等工具，编译 OpenFOAM 8，内容如下：

```
Bootstrap: docker
From: centos:8

%help
    This recipe provides an OpenFOAM-8 environment installed
    with GCC and OpenMPI-4.

%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
```

(下页继续)

(续上页)

```
    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
    pkg=OpenFOAM
    vrs=8

    ### 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-8.0.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-8.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 '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-*
```

(下页继续)

(续上页)

```
    strip $FOAM_APPBIN/*

    ### Source bashrc at runtime
    echo '. /opt/OpenFOAM/OpenFOAM-8/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

    export PATH=$MPI_BIN:$PATH
    export LD_LIBRARY_PATH=$MPI_LIB:$LD_LIBRARY_PATH

%test
    . /opt/OpenFOAM/OpenFOAM-8/etc/bashrc
    icoFoam -help

%runscript
    echo
    echo "OpenFOAM installation is available under $WM_PROJECT_DIR"
    echo
```

调用“容器化的 Singularity”构建镜像，由于指令集的差异，使用的镜像标签也有 x86 和 arm 分别。

小技巧： 在 container-x86 上请使用 sjtuhpc/centos7-singularity:x86 ，在 container-arm 上请使用 sjtuhpc/centos7-singularity:arm 。

在 container-x86 节点上上构建镜像，构建的镜像保存在当前目录 sample-x86.sif :

```
$ docker run --privileged --rm -v \
    ${PWD}:/home/singularity \
    sjtuhpc/centos7-singularity:x86 \
    singularity build /home/singularity/sample-x86.sif /home/
↪singularity/sample.def
```

在 container-arm 节点上上构建镜像，构建的镜像保存在当前目录 sample.sif :

```
$ docker run --privileged --rm -v \
    ${PWD}:/home/singularity \
    sjtuhpc/centos7-singularity:arm \
    singularity build /home/singularity/sample-arm.sif /home/
↪singularity/sample.def
```

在镜像构建过程中“模拟”了 root 特权，因此生成镜像文文件属主是 root：

```
$  ls -alh *.sif
-rwxr-xr-x 1 root root 475M Jun  3 22:43 sample-x86.sif
```

将构建出的镜像从 container 节点传回登录节点的家目录中：

```
$ scp sample-x86.sif YOUR_USERNAME@login1:~/
```

然后编写作业脚本提交到作业调度系统。下面这个作业脚本示例使用刚才构建的 OpenFOAM 镜像，完成了网格划分、模型求解、后处理等操作：

```
#!/bin/bash

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

ulimit -s unlimited
ulimit -l unlimited

module load openmpi/4.0.3-gcc-9.3.0

export IMAGE_NAME=./8-centos8.sif

singularity exec $IMAGE_NAME surfaceFeatures
singularity exec $IMAGE_NAME blockMesh
singularity exec $IMAGE_NAME decomposePar -copyZero
mpirun -n $SLURM_NTASKS singularity exec $IMAGE_NAME snappyHexMesh -
 ↪overwrite -parallel
mpirun -n $SLURM_NTASKS singularity exec $IMAGE_NAME potentialFoam -
 ↪parallel
mpirun -n $SLURM_NTASKS singularity exec $IMAGE_NAME simpleFoam -
 ↪parallel
```

4.8.5 AI 平台容器编译

与 x86 平台容器编译方式类似，在 AI 平台也有三种容器构建方式：

1. 使用 Singularity 加载 AI 平台预编译的镜像。
2. 使用 Singularity 拉取远端镜像。
3. 按需定制 Singularity 镜像。

拉取 AI 平台预编译的镜像

AI 平台的基础与应用镜像分别托管在以下 **Docker Hub** 仓库：

- x86 平台基础镜像
- x86 平台应用镜像

如该镜像的 tag 带有 gpu 字样，则该镜像为 AI 平台预编译镜像，此时可用 `singularity pull` 命令拉去该镜像到本地直接使用。

以下示例拉取预编译的 GPU 版 gromacs 镜像到本地：

```
$ unset XDG_RUNTIME_DIR
$ unset SINGULARITY_BIND
$ unset MODULEPATH
$ singularity pull docker://sjtuhpc/hpc-app-container:gromacs-gpu-
↪2019
```

查看生成的镜像文件

```
$ ls
gromacs-gpu-2019.sif
```

使用自己制作的镜像文件运行提交 gromacs 作业：

```
#!/bin/bash
#SBATCH -J gromacs_gpu_test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:2

srun --mpi=pmi2 singularity run --nv gromacs-gpu-2019.sif gmx_mpi␣
↪mdrun -deffnm benchmark -ntomp 1 -s ./ion_channel.tpr
```

使用 Singularity 拉取外部预编译应用镜像

您可以使用 `singularity pull` 拉取远端预编译的镜像，从而直接使用外部预预编译镜像仓库提供的丰富软件资源。Singularity 可以从 Docker Hub(以 `docker://` 开头)、Singularity Hub(以 `shub://` 开头) 等地址拉取镜像。如下命令从 Docker Hub 拉取 NGC 构建的 Pytorch 镜像，保存成名为 `pytorch_21.10-py3.sif` 的镜像文件：

```
$ singularity pull docker://nvcr.io/nvidia/pytorch:21.10-py3
```

查看生成的镜像文件

```
$ ls centos8.sif
pytorch_21.10-py3.sif
```

申请 GPU 节点资源，加载容器镜像，并且在容器环境中运行 `python -c "import torch"` 命令查看有无报错：

```
$ singularity run --nv  pytorch_21.10-py3.sif python -c "import␣
↪torch"
```

没有报错说明镜像中的 pytorch 已经加载成功。

通过交互式 Shell 构建 AI 应用镜像

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。

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
```

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` 下载基础操作系统镜像。

```
$ docker pull centos:8
```

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
...
[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␣
↪                    PORTS     NAMES
515e913f12cb   5d0da3dc9764   "/bin/bash"    4 seconds ago    Exited␣
↪(0) 2 seconds ago             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
```

通过 Definition File 构建 AI 应用镜像

Singularity 还可以使用“镜像定义文件”(Definition File) 描述镜像构建过程。镜像定义文件是一个文本文件，描述了构建镜像使用的基本镜像、构建过程执行的命令，如软件包管理命令 yum, apt-get 等等。

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
```

> 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
```

我们准备一个镜像定义文件 sample.def ，内容如下：

```
BootStrap: docker
From: sjtuhpc/hpc-base-container:gcc-8.cuda-10.2.ompi-4.0
Stage: build
%post
    . /.singularity.d/env/10-docker*.sh

%post
    yum install -y \
        epel-release
    yum install -y \
        cmake3 \
```

(下页继续)

(续上页)

```
        make \
        wget
    ln -s /usr/bin/cmake3 /usr/bin/cmake
    rm -rf /var/cache/yum/*

# Gromacs version 2020
%post
    mkdir -p /var/tmp && wget -q -nc --no-check-certificate -P /var/
↪tmp http://ftp.gromacs.org/pub/gromacs/gromacs-2019.6.tar.gz
    mkdir -p /var/tmp && tar -x -f /var/tmp/gromacs-2019.6.tar.gz -
↪C /var/tmp -z
    mkdir -p /var/tmp/gromacs-2019.6/build && cd /var/tmp/gromacs-
↪2019.6/build
    cmake -DCMAKE_INSTALL_PREFIX=/opt/gromacs -D CUDA_TOOLKIT_ROOT_
↪DIR=/usr/local/cuda \
          -D CMAKE_BUILD_TYPE=Release -D GMX_SIMD=AVX_512 \
          -D GMX_BUILD_OWN_FFTW=ON -D GMX_GPU=ON -D GMX_MPI=ON -D␣
↪GMX_OPENMP=ON \
          -D GMX_PREFER_STATIC_LIBS=ON /var/tmp/gromacs-2019.6
    cmake --build /var/tmp/gromacs-2019.6/build --target all -- -j
↪$(nproc)
    cmake --build /var/tmp/gromacs-2019.6/build --target install --␣
↪-j$(nproc)
    rm -rf /var/tmp/gromacs-2019.6 /var/tmp/gromacs-2019.6.tar.gz
%environment
    export LD_LIBRARY_PATH=/opt/gromacs/lib64:$LD_LIBRARY_PATH \
    epxort PATH=/opt/gromacs/bin:$PATH


BootStrap: docker
From: sjtuhpc/hpc-base-container:gcc-8.cuda-10.2.ompi-4.0
%post
    . /.singularity.d/env/10-docker*.sh

# Gromacs version 2020
%files from build
    /opt/gromacs /opt/gromacs
%environment
    export LD_LIBRARY_PATH=/opt/gromacs/lib64:$LD_LIBRARY_PATH
    export PATH=/opt/gromacs/bin:$PATH
```

构建的镜像保存在当前目录 `sample-x86.sif`：

```
$ docker run --privileged --rm -v \
     ${PWD}:/home/singularity \
     sjtuhpc/centos7-singularity:x86 \
     singularity build /home/singularity/sample-x86.sif /home/
↪singularity/sample.def
```

在镜像构建过程中“模拟”了 root 特权，因此生成镜像文文件属主是 root：

```
$  ls -alh *.sif
-rwxr-xr-x 1 root root 475M Jun  3 22:43 sample-x86.sif
```

将构建出的镜像从 container 节点传回登录节点的家目录中:

```
$ scp sample-x86.sif YOUR_USERNAME@login1:~/
```

然后编写作业脚本提交到作业调度系统。下面这个作业脚本示例使用刚才构建的 Gromacs GPU 版镜像:

```
#!/bin/bash
#SBATCH -J gromacs_gpu_test
#SBATCH -p dgx2
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=12
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:2

srun --mpi=pmi2 singularity run --nv sample-x86.sif gmx_mpi mdrun -
↪deffnm benchmark -ntomp 1 -s ./ion_channel.tpr
```

4.8.6 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/

4.9 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
GNU Compiler Collection	GNU 编译器套件	9.3(D)	cpu arm
Intel Compiler	Intel 编译器套件	21.4.0	cpu
CUDA	NVIDIA CUDA SDK	10.2(D)	cpu
NVIDIA HPC SDK	NVIDIA HPC SDK	20.11(D)	gpu
JDK	Java 开发套件	12.0	cpu arm
bisheng	ARM 平台高性能编译器	1.3.3	arm
OpenMPI	OpenMPI 是一个开源 MPI 实现	3.1.5	cpu arm
Intel-MPI	Intel MPI 是一个高性能 MPI 实现	2019.4	cpu
Intel-Vtune	Intel MPI 是一个高性能 MPI 实现	2019.4	cpu
Hyper-MPI	ARM 平台高性能 MPI 实现	4.0.3	arm
Intel mkl	Intel MKL(数学核心函数库) 包含 BLAS、LAPACK、SCALAPACK、的高效实现	19.3	cpu
OpenBLAS	OpenBLAS 是一个开源的 BLAS、LAPACK 高效实现	2.4	cpu arm
GSL	GNU Scientific Library 是由 C/C++ 编写的高效开源科学计算库	2.5	cpu arm
Eigen	Eigen 是一个 C++ 编写的线性代数模板库	3.3.7	cpu arm
FFTW	FFTW 是一个被广泛使用快速傅里叶变换库	3.3.7	cpu arm
Hypre	hypre 是一个求解大型线性方程组的 C 语言库	2.20.0	cpu
Conda	环境管理与包管理软件	4.6.14	cpu arm
Python	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
PERL	一种功能丰富的计算机程序语言	5.30	cpu arm

基准测试

表 2: 基准测试

软件	描述	可用版本	平台
OSU Benchmarks	由俄亥俄州立大学开发的高速网络基准测试	5.6.3	cpu arm
HPL	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	cpu arm
HPCG	新的 HPC 系统排名指标, 可作为 HPL 基准程序的补充	3.1	cpu
minife	非结构化隐式有限元代码的代理应用程序	2.1.0	cpu

原子分子软件

表 3: 原子分子软件

软件	描述	可用版本	平台
ABINIT	第一原理平面波赝势总能计算软件包	8.10.3	cpu
GAUSSIAN	量子化学软件包	自己安装	cpu
GROMACS	研究生物分子体系的分子动力学程序包	2019 2020	cpu gpu arm
VASP	量化计算软件	5.4.4	cpu gpu arm
Amber	分子动力学模拟软件	20 22	cpu gpu arm
LAMMPS	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
LAMMPS-RBE	上海交通大学基于 lammps 二次开发的自研软件	2021	cpu arm
NWChem	量子化学计算软件	6.8.1	cpu
Quantum ESPRESSO	纳米尺度上进行电子结构计算和材料建模	6.6	cpu arm
CP2k	基于密度泛函理论的从头算分子动力学软件	6.1	cpu gpu arm
SIESTA	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
VMD	分子可视化程序	1.9.4	cpu Studio
OVITO	分析数据及可视化动力学结果	3.2.1	cpu Studio
Paraview	开源、跨平台数据分析和可视化程序	0.4.1	cpu Studio
gnuplot	开源、跨平台数据分析和可视化程序	4.6	cpu Studio

工程计算软件

表 4: 工程计算软件

软件	描述	可用版本	平台
CESM	地球气候系统模式	2.2	cpu
Calculix	开源有限元分析软件	2.17	cpu
Cantera	开源化学反应动力学分析软件	3.0.0	cpu
WRF	中尺度天气预报模式	3.9,4.3.1	cpu arm
CMAQ	中尺度天气预报模式	3.9,4.3.1	cpu arm
cplex	大型线性以及非线性规划求解软件		cpu
Nektar++	有限元分析软件	5.0.0	cpu
Octave	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
OpenFOAM	面向对象的计算流体力学类库	7, 1712, 1812,	cpu
OpenRadioss	开源显式动力学求解器		cpu
WhiskyTHC	开源的大型数值相对论求解器		cpu
PSI4	面向对象的计算化学软件	1.5	cpu
PySPH	基于 python 的光滑粒子流体动力学框架	1.5	cpu
STAR-CCM+	新一代通用计算流体力学分析软件	2.7.0	cpu
Gerris	求解描述流体流动的偏微分方程的软件	1.3.2	cpu
SUMO	开源交通信号模拟软件	1.10.0	cpu
Elphbolt	用于求解耦合电子和声子-玻尔兹曼输运方程	1.0	cpu
DAMSK	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
ROMS	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
PROJ	PROJ 是通用的坐标转换软件	7.2.1	cpu
colmap	COLMAP 是用来做 SfM 或者 MVS 的软件	3.7	cpu
pymultinest	pymultinest 可用于贝叶斯分析、参数选择等。	2.11	cpu

AI 计算软件

表 5: AI 计算软件

软件	描述	可用版本	平台
PyTorch	开源的 python 机器学习库	1.6.0	gpu
TensorFlow	用于各类机器学习算法的编程实现	2.0.0	gpu
Keras	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
Apache TVM	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
AlphaFold2	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
AUGUSTUS	Augustus 是用作 de novo 基因注释 (即从头预测) 的软件。	1.5.1	cpu
BASIL-ANISE	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
BatVI	检测病毒整合的软件包	2.1.0	cpu
BCFtools	vcf 和 BCF 文件的工具合集	1.10.2	cpu arm
BEDTOOLS2	可以对 genomic features 进行比较、相关操作和注释的工具	2.27.1	cpu
BICseq2	SV 检测软件	0.3.0	cpu
Bismark	全基因组甲基化软件	0.19.0	cpu
Blast-plus	基于局部比对算法的搜索工具	2.9.0	arm
BOWTIE	一种快速短对齐器	1.2.3	cpu
BOWTIE2	短序列比对软件	2.3.5.1	cpu
BreakDancer	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
BWA	一款将序列比对到参考基因组上的软件	0.7.17	cpu

下页继续

表 6 – 续上页

软件	描述	可用版本	平台
cd-hit	序列聚类软件	4.8.1	cpu
CDO	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
cdsapi	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
chimera	一个用于聚合产物二次分析的软件包	1.32.0	cpu
CLEVER	用于分析下一代测序数据的工具	10.6.2	cpu
CNVnator	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
ColabFold	AI 蛋白结构预测软件	1.2	gpu
Control-FERRC	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
CREST	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
CUFFLINKS	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
DeepGo	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
DeepVariant	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
DELLY	一种集成的结构变异（SV）预测方法	2.2.1, 0.8.3	cpu arm
DESeq2	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
ERDS	实体关系图	2.2.1	cpu
FastQC	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
FermiKit	Illumina 全基因组 germline 变异检出流程	0.11.7	cpu
FSL	核磁共振、DTI 脑成像数据分析软件	6.0	cpu
GATK	二代重测序数据分析软件	3.8	cpu

下页继续

表 6 – 续上页

软件	描述	可用版本	平台
GEANT4	C++ 软件包，用于模拟粒子在物质中输运的物理过程	10.6.2	cpu
GenomeStrip	识别拷贝数变异区	10.0.130	cpu gpu
GMAP-GSNAP	mRNA 和 EST 序列的基因组定位和比对程序	2019-5-12	cpu
GRAPHMAP	一个高度敏感和准确的映射器	0.3.0	cpu
GRIDSS	用于 Web 的二维布局系统	2.2.1	cpu
GsUtil	Python 应用程序，可让您从命令行访问 Google Cloud Storage	1.3.4d	cpu
HISAT2	快速、灵敏的比对软件	2.1.0	cpu arm
HMMER	序列比对软件	3.3	arm
Hydra-sv	使用双端映射检测结构变异 (SV) 断点	0.7.17	cpu
km	使用 k-mer 分解进行 RNA-seq 研究的软件	240	cpu
Lumpy	用于基因组结构变异 SV 的检测	1.2.3	cpu
LUMPY-SV	一款 SV 检测工具	0.2.13	cpu
MAKER	基因组注释流程	2.1.2	cpu
Manta	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
MEGAHIT	二代测序从头组装工具	1.1.4	cpu
METIS	图切分软件包	5.1.0	cpu
MELT	模拟岩浆演化的工具	2.27.1	cpu
MetaSV	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
MindTheGap	虚拟原型中的 GPU 建模	5.1.0	cpu
Mobster	一个 mob/pair 编程计时器	3.2.7a	cpu
MrBayes	系统树重建软件	3.2.7a	cpu

下页继续

表 6 – 续上页

软件	描述	可用版本	平台
NCBI-RMBLASTN	基于序列相似性的数据库搜索程序	2.2.28	cpu
OpenSlide-python	C 库，它提供了一个简单的界面来读取整张幻灯片图像		cpu
pandas	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
PBSV	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
PICARD	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
Pindel	检测复杂 INDEL	2.19.0	cpu
PRISM	多功能统计工具	1.1.8, 3.6.2	cpu Studio
r-rgl	为 3D 交互式图形提供中高级函数	240	cpu
RELION	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
RetroSeq	基因分型转座元件变体 (TEV) 检测工具	3.0.8	gpu Studio
RNA-SEQC	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
RoseTTAFold	开源蛋白质结构预测软件	1.0	gpu
STAR	测序序列与参考序列比对，常用于 RNA-seq 的比对	2.7.6.a	cpu
SALMON	一种从 RNA-seq 数据中快速量化转录本的工具	0.14.1	cpu
Samtools	用于操作 sam 和 bam 文件的工具合集	1.9	cpu
Sniffles	用于检测长读长数据的 SV	1.1.8	cpu
SOAPDENOVO2	用于序列组装的软件	240	cpu
SRATOOLKIT	将.sra 文件转换为.fstaq.gz 文件的工具	2.9.6	cpu

下页继续

表 6 – 续上页

软件	描述	可用版本	平台
sra-tools	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
STRINGTIE	转录组标表达定量软件	1.3.4d	cpu
STRique	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
SV2	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
SvABA	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
SVDetect	从测序数据中分离和预测染色体内和染色体间重排的应用程序	240	cpu
TOPHAT	将 RNA-Seq 数据进行快速剪接映射的程序	2.1.2	cpu
VARDICTJAVA	用 Java 和 Perl 编写的变种发现程序	1.5.1	cpu
VSEARCH	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
VariationHunter	变异检测软件	1.9.8	cpu
Wham	加权直方图分析方法	2.9.6	cpu
WGCNA	加权相关网络分析		cpu
Rosetta	模拟大分子结构的综合软件	3.12	cpu
Hic_breakfinder	基因组 SV 检测工具	1.0	cpu
EasyFuse	融合基因检测工具	1.3.5	cpu
MetaWRAP	宏基因组数据分析流程工具	1.3	cpu
GATE	医学成像和放射治疗的数值模拟软件	9.2	cpu
pangenie	基于范基因组的基因分型器	v2.0.0	cpu

4.9.1 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-impi    (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-
↪impi
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)
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
```

(下页继续)

(续上页)

```
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  ␣
↪                  (D)
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  ␣
↪                (D)
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                                (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)
```

(下页继续)

(续上页)

```
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/

4.9.2 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。

判断 **GCC** 支持的编译语言方法

```
[hpc@node499 ~]$ gcc --version
gcc (GCC) 8.3.1 20191121 (Red Hat 8.3.1-5)

[hpc@node499 ~]$ g++ --version
g++ (GCC) 8.3.1 20191121 (Red Hat 8.3.1-5)

[hpc@node499 ~]$ gfortran --version
GNU Fortran (GCC) 8.3.1 20191121 (Red Hat 8.3.1-5)
```

集群平台上的 **GCC**

- 思源一号上的 *GCC*
- *π2.0* 上的 *GCC*
- *ARM* 上的 *GCC*

思源一号上的 **GCC**

版本	加载方式
gcc-8.3.1	module load gcc/8.3.1
gcc-8.5.0	module load gcc/8.5.0
gcc-9.3.0	module load gcc/9.3.0
gcc-10.3.0	module load gcc/10.3.0
gcc-11.2.0	module load gcc/11.2.0

思源一号上的 GCC 默认版本为: 8.3.1。

π2.0 上的 GCC

版本	加载方式
gcc-5.5.0	module load gcc/5.5.0
gcc-7.4.0	module load gcc/7.4.0
gcc-8.3.0	module load gcc/8.3.0
gcc-9.2.0	module load gcc/9.2.0
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

π2.0 上的 GCC 默认版本为: 4.8.5。

ARM 上的 GCC

版本	加载方式
gcc-8.4.0	module load gcc/8.4.0
gcc-8.5.0	module load gcc/8.5.0
gcc-9.3.0	module load gcc/9.3.0

ARM 平台上的 GCC 默认版本为: 4.8.5。

参考资料

- Top 20 licenses https://www.blackducksoftware.com/top-open-source-licenses/

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;

  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;
  }
```

(下页继续)

(续上页)

```
  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");
  }

  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);
```

(下页继续)

(续上页)

```
  printf (" Example completed. \n\n");
  return 0;
}
```

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=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  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
```

(下页继续)

(续上页)

```
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.
```

(下页继续)

(续上页)

```
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();

     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
```

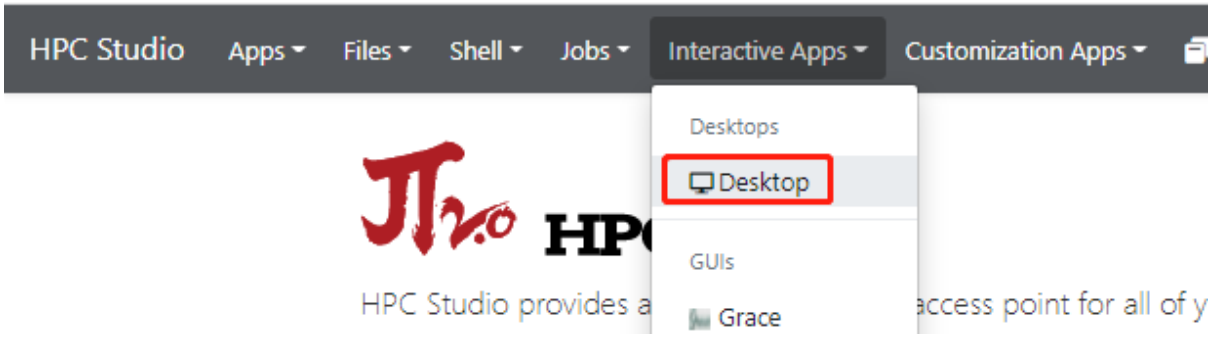
集群上的 **VTUNE**

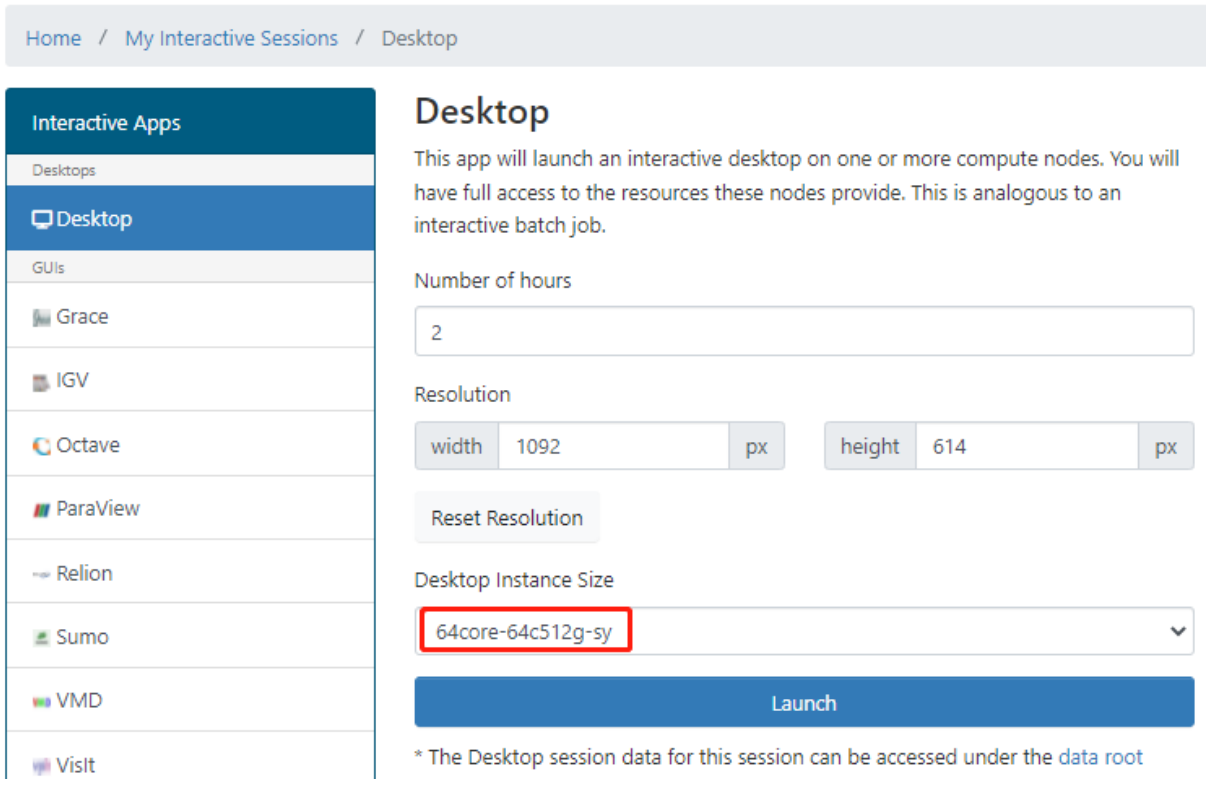
- 思源一号上的 *VTUNE*

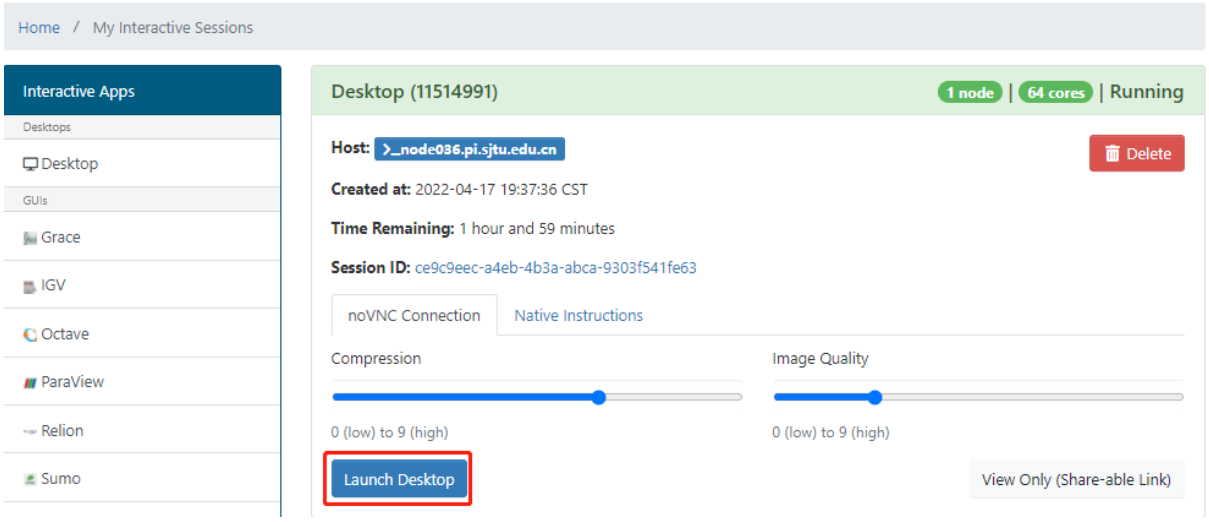
思源一号上的 **VTUNE**

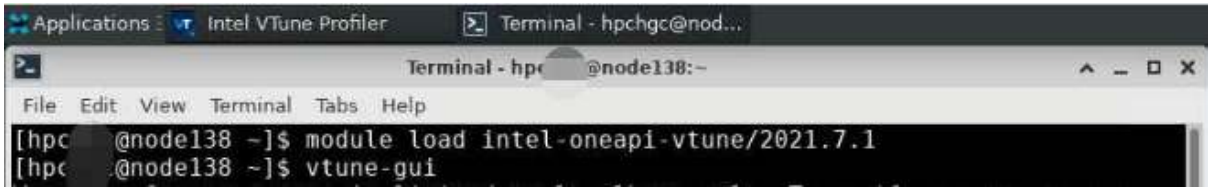
登录 hpc studio

调出 **VTUNE** 图形化界面

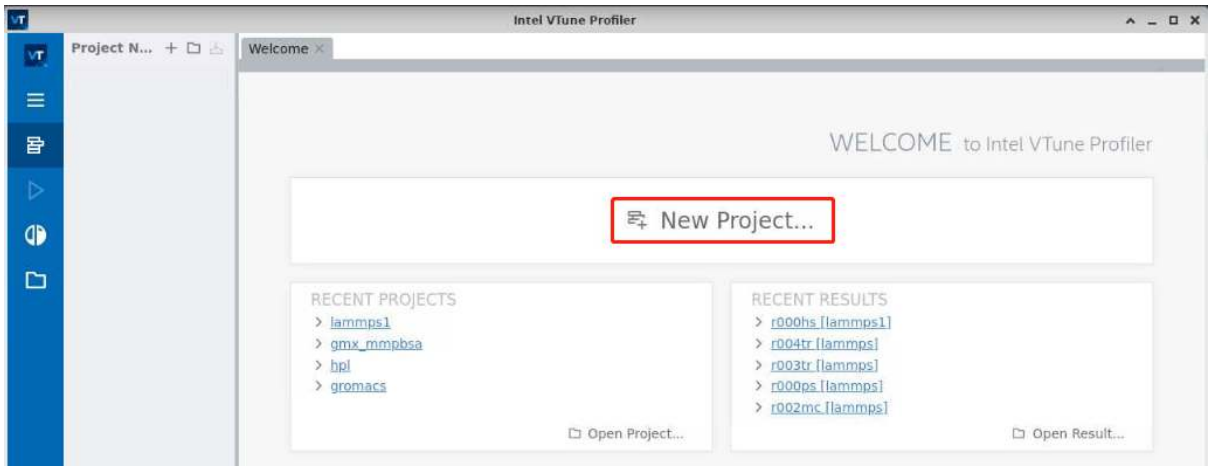


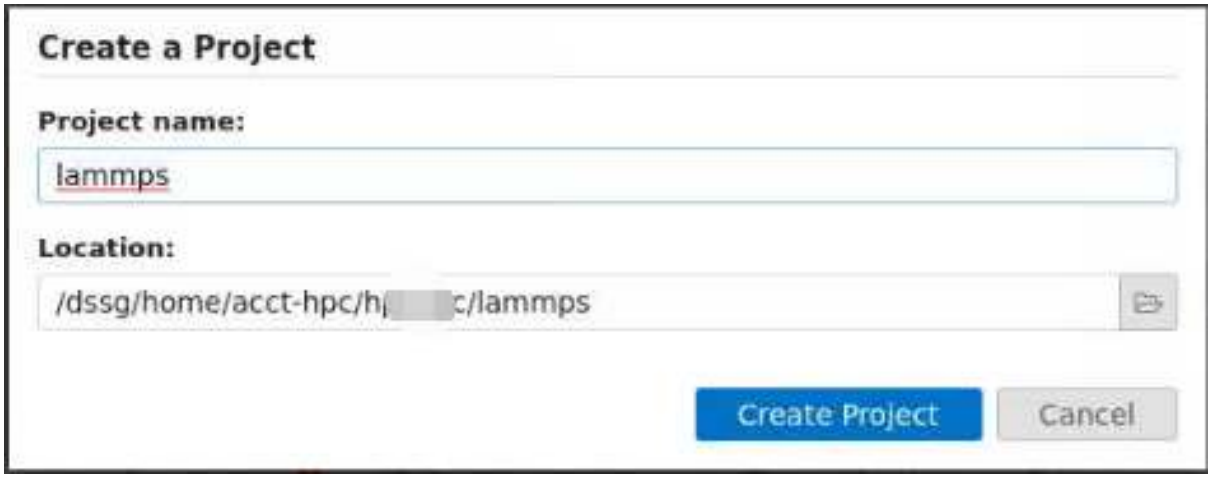

Home / My Interactive Sessions / Desktop
Interactive Apps
Desktops
Desktop
GUIs
Grace
IGV
Octave
ParaView
Relion
Sumo
VMD
VisIt
Desktop
This app will launch an interactive desktop on one or more compute nodes. You will have full access to the resources these nodes provide. This is analogous to an interactive batch job.
Number of hours
2
Resolution
width 1092 px
height 614 px
Reset Resolution
Desktop Instance Size
64core-64c512g-sy
Launch
* The Desktop session data for this session can be accessed under the data root

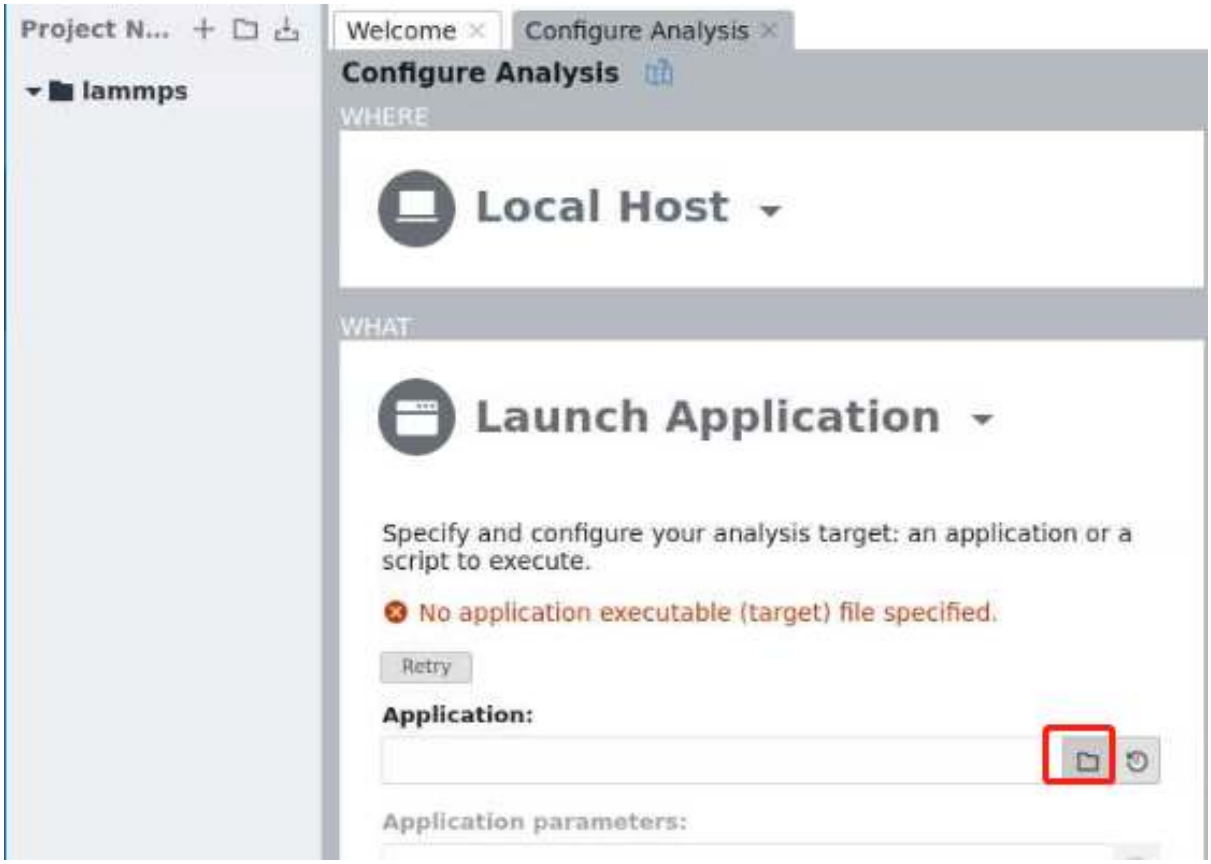

Home / My Interactive Sessions
Interactive Apps
Desktops
Desktop
GUIs
Grace
IGV
Octave
ParaView
Relion
Sumo
Desktop (11514991)
1 node | 64 cores | Running
Host: >_node036.pi.sjtu.edu.cn
Delete
Created at: 2022-04-17 19:37:36 CST
Time Remaining: 1 hour and 59 minutes
Session ID: ce9c9eec-a4eb-4b3a-abca-9303f541fe63
noVNC Connection
Native Instructions
Compression
Image Quality
0 (low) to 9 (high)
0 (low) to 9 (high)
Launch Desktop
View Only (Share-able Link)

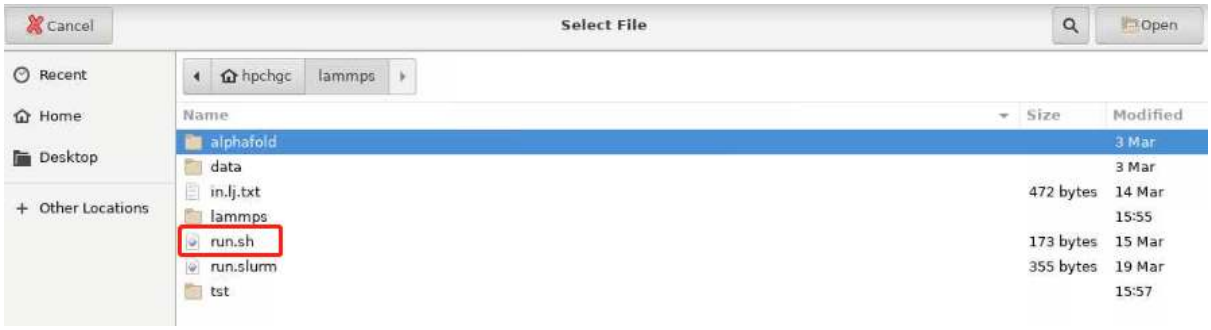

Applications
Intel VTune Profiler
Terminal - hpchgc@nod...
Terminal - hpc @node138:~
File Edit View Terminal Tabs Help
[hpc @node138 ~]$ module load intel-oneapi-vtune/2021.7.1
[hpc @node138 ~]$ vtune-gui

导入应用程序的流程


Intel VTune Profiler
Project N...
Welcome
WELCOME to Intel VTune Profiler
New Project...
RECENT PROJECTS
lammps1
gmx_mmpbsa
hpl
gromacs
Open Project...
RECENT RESULTS
r000hs [lammps1]
r004tr [lammps]
r003tr [lammps]
r000ps [lammps]
r002mc [lammps]
Open Result...


Create a Project
Project name:
lammps
Location:
/dssg/home/acct-hpc/h c/lammps
Create Project
Cancel





查找运行程序的热点

执行程序的目录与脚本内容

```
[hpc@node243 test]$ tree lammps/
lammps/
├── in.lj
└── run.sh
```

run.sh

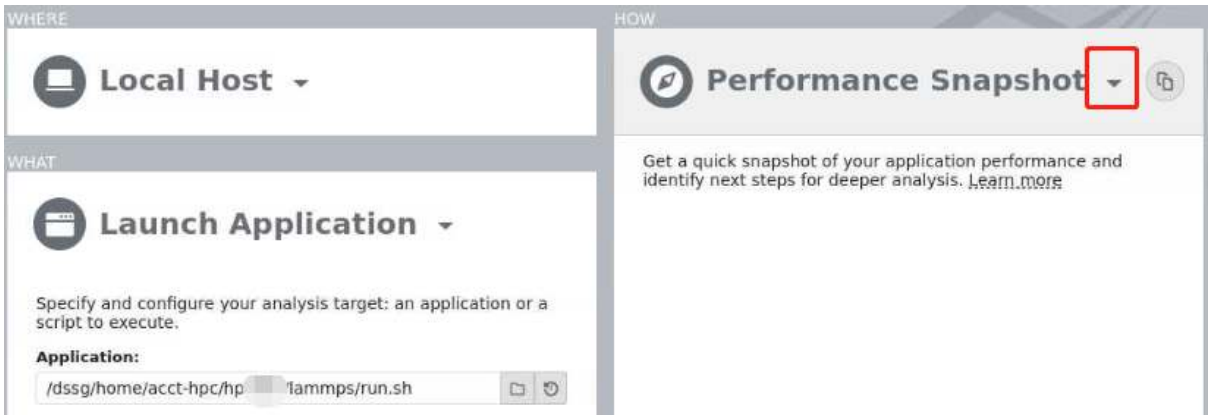
```
#!/bin/bash
module load oneapi
module load lammps/20210310-intel-2021.4.0
```

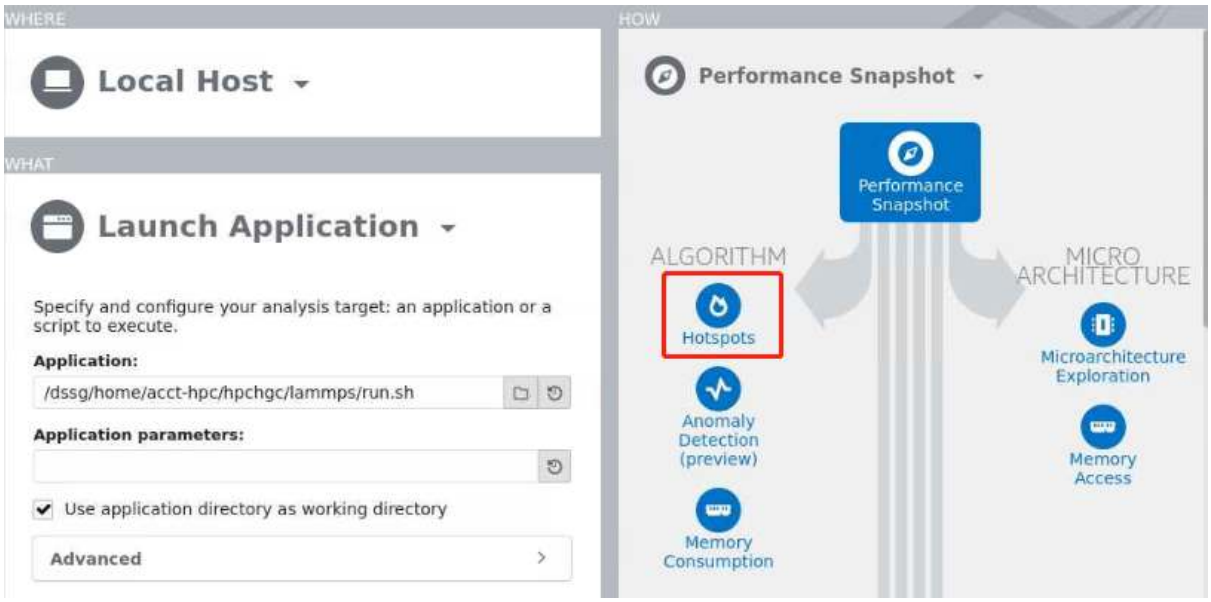
(下页继续)

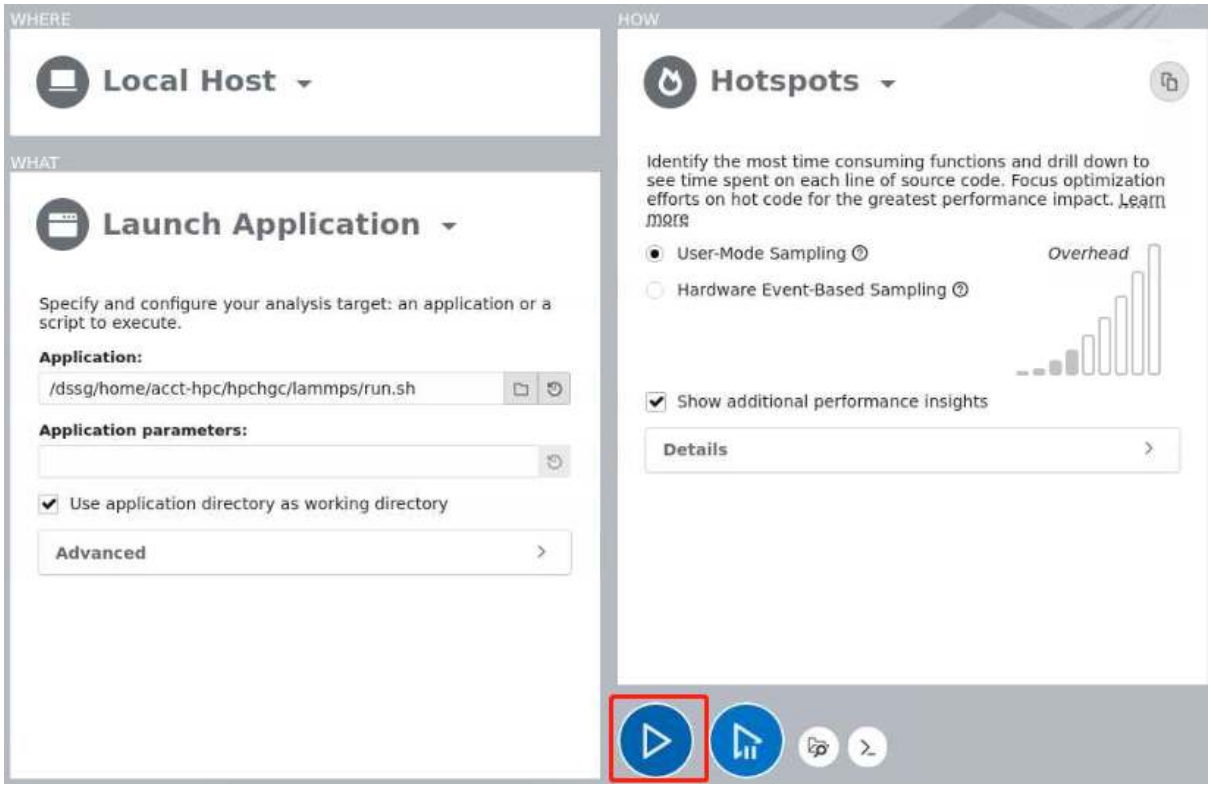
(续上页)

```
ulimit -s unlimited
ulimit -l unlimited
export OMP_NUM_THREADS=1
mpirun -np 32  lmp -i in.lj.txt
```

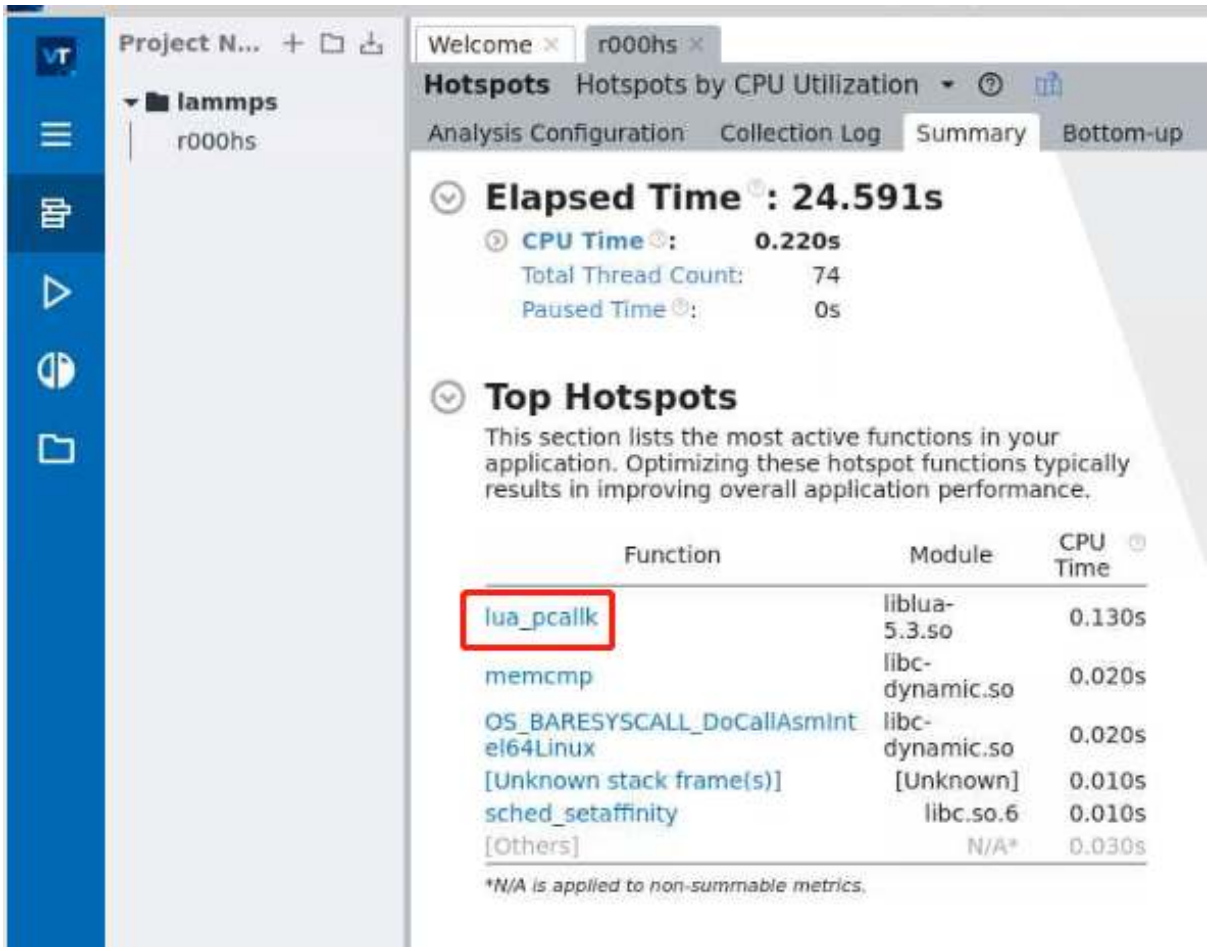
打开分析热点

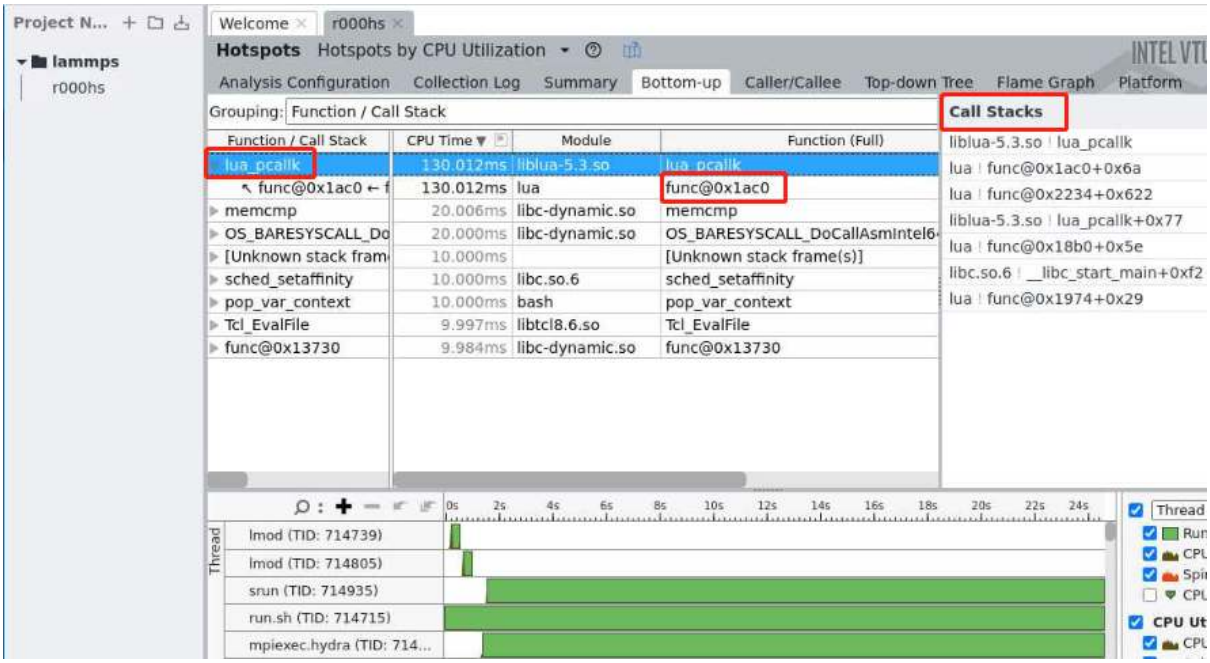





WHERE
Local Host
WHAT
Launch Application
Specify and configure your analysis target: an application or a script to execute.
Application:
/dssg/home/acct-hpc/hpchgc/lammps/run.sh
Application parameters:
Use application directory as working directory
Advanced
HOW
Hotspots
Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. Learn more
User-Mode Sampling
Hardware Event-Based Sampling
Overhead
Show additional performance insights
Details

运行结果


Project N...
lammps
r000hs
Welcome
r000hs
Hotspots
Hotspots by CPU Utilization
Analysis Configuration
Collection Log
Summary
Bottom-up
Elapsed Time: 24.591s
CPU Time: 0.220s
Total Thread Count: 74
Paused Time: 0s
Top Hotspots
This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.
Function
Module
CPU Time
lua_pcallk
liblua-5.3.so
0.130s
memcmp
libc-dynamic.so
0.020s
OS_BARESYSCALL_DoCallAsmIntel64Linux
libc-dynamic.so
0.020s
[Unknown stack frame(s)]
[Unknown]
0.010s
sched_setaffinity
libc.so.6
0.010s
[Others]
N/A*
0.030s
*N/A is applied to non-summable metrics.


Project N...
lammps
r000hs
Welcome
r000hs
Hotspots
Hotspots by CPU Utilization
Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Flame Graph
Platform
Grouping: Function / Call Stack
Call Stacks
Function / Call Stack
CPU Time
Module
Function (Full)
lua_pcallk
130.012ms
liblua-5.3.so
lua_pcallk
func@0x1ac0
130.012ms
lua
func@0x1ac0
memcmp
20.006ms
libc-dynamic.so
memcmp
OS_BARESYSCALL_Do
20.000ms
libc-dynamic.so
OS_BARESYSCALL_DoCallAsmIntel6
[Unknown stack fram
10.000ms
[Unknown stack frame(s)]
sched_setaffinity
10.000ms
libc.so.6
sched_setaffinity
pop_var_context
10.000ms
bash
pop_var_context
Tcl_EvalFile
9.997ms
libtcl8.6.so
Tcl_EvalFile
func@0x13730
9.984ms
libc-dynamic.so
func@0x13730
liblua-5.3.so ! lua_pcallk
lua ! func@0x1ac0+0x6a
lua ! func@0x2234+0x622
liblua-5.3.so ! lua_pcallk+0x77
lua ! func@0x18b0+0x5e
libc.so.6 ! __libc_start_main+0xf2
lua ! func@0x1974+0x29
Thread
lmod (TID: 714739)
lmod (TID: 714805)
srun (TID: 714935)
run.sh (TID: 714715)
mpiexec.hydra (TID: 714...

参考资料

- intel-parallel-studio

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
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;
```

(下页继续)

(续上页)

```
    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 = 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
```

(下页继续)

(续上页)

```
#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/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
```

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：

```
// This is a mpi-openmp-hybrid parallel program to calculate the␣
↪value of pi ! !

#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;
  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
```

(下页继续)

(续上页)

```
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 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

OpenBLAS

OpenBLAS 是一个开源的线性代数库，高效实现了 BLAS(Basic Linear Algebra Subprograms) 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	cpu	Spack	openblas/0.3.7-gcc-9.2.0
0.3.7	arm	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
```

(下页继续)

(续上页)

```
    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;
    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-wqdecm4rkyyhejagxwmnabt6lscgm45d/lib64/
↪libgfortran.so.5 (0x00002ae9278c8000)
     libgomp.so.1 => /lustre/opt/cascadelake/linux-centos7-x86_64/
↪gcc-4.8.5/gcc-9.2.0-wqdecm4rkyyhejagxwmnabt6lscgm45d/lib64/
↪libgomp.so.1 (0x00002ae927d57000)
```

(下页继续)

(续上页)

```
    /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-wqdecm4rkyyhejagxwmnabt6lscgm45d/lib64/
↪libquadmath.so.0 (0x00002ae927f8d000)
    libgcc_s.so.1 => /lustre/opt/cascadelake/linux-centos7-x86_64/
↪gcc-4.8.5/gcc-9.2.0-wqdecm4rkyyhejagxwmnabt6lscgm45d/lib64/libgcc_
↪s.so.1 (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-a5tvx33on7quyl7o2sygvyjqnysfcw6n/
↪lib64/libgfortran.so.5 (0x000040002f8c0000)
    libgomp.so.1 => /lustre/opt/kunpeng920/linux-centos7-aarch64/
↪gcc-4.8.5/gcc-9.3.0-a5tvx33on7quyl7o2sygvyjqnysfcw6n/lib64/
↪libgomp.so.1 (0x000040002fa30000)
```

(下页继续)

(续上页)

```
    /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-a5tvx33on7quyl7o2sygvyjqnysfcw6n/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
```

(下页继续)

(续上页)

```
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 myeigen.cpp：

```
#include <iostream>
#include <Eigen/Dense>
using Eigen::MatrixXd;
int main()
{
    MatrixXd m(2,2);
    m(0,0) = 3;
    m(1,0) = 2.5;
    m(0,1) = -1;
    m(1,1) = m(1,0) + m(0,1);
    std::cout << m << std::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
```

(下页继续)

(续上页)

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

ulimit -s unlimited
ulimit -l unlimited

module load eigen/3.4.0-gcc-8.3.1

g++  myeigen.cpp -o myeigen

./myeigen
```

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：

```
3  -1
2.5 1.5
```

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/3.3.7-gcc-8.3.0

g++  myeigen.cpp -o myeigen

./myeigen
```

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：

```
3  -1
2.5 1.5
```

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;
```

(下页继续)

(续上页)

```
  in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
  out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
  for( i=0; i < N; i++)
  {
      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 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 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));
    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
```

(下页继续)

(续上页)

```
#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
```

(下页继续)

(续上页)

```
        else
            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
```

(下页继续)

(续上页)

```
#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 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	cpu	Spack	hypre/2.18.0-gcc-11.2.0-openblas-openmpi
2.20.0	cpu	Spack	hypre/2.20.0-gcc-11.2.0-openblas-openmpi
2.23.0	cpu	Spack	hypre/2.23.0-gcc-11.2.0-openblas-openmpi
其他版本	cpu	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
```

(下页继续)

(续上页)

```
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
ex7.c
ex8.c
ex9.c
ex.h
Makefile
Makefile_gpu
vis
vis.c
```

3. 在该目录下编写如下 hypretest.slurm 脚本文件 (编译 ex1.c 文件并运行):

```
#!/bin/bash

#SBATCH --job-name=hypretest
#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 hypretest.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
 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 hypretest.slurm 脚本文件 (编译 ex1.c 文件并运行)：

```
#!/bin/bash

#SBATCH --job-name=hypretest
#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 hypretest.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
 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

Hyper-MPI 是

Hyper-MPI 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 hmpi/4.0.3-gcc-9.3.0
```

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");
```

(下页继续)

(续上页)

```
    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++) {
```

(下页继续)

(续上页)

```
        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
```

(下页继续)

(续上页)

```
#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	cpu	Spack	miniconda2/4.7.12.1 思源一号
4.10.3	cpu	Spack	*miniconda3/4.10.3* 思源一号
4.5.12	arm	Spack	*conda4aarch64/1.0.0-gcc-4.8.5*
4.7.12.1	cpu	Spack	miniconda2/4.7.12.1
4.6.14	cpu	Spack	miniconda2/4.6.14
4.8.2	cpu	Spack	*miniconda3/4.8.2*
4.7.12.1	cpu	Spack	miniconda3/4.7.12.1
4.6.14	cpu	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:

```
srun -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:

```
srun -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）。

```
$ π 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
srun -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医学用户主要包含 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

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：
$ perldoc perllocal
...
```

Perl 的 SLURM 作业示例

用法：sbatch job.slurm

```
#!/bin/bash

#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：

```
srun -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：

```
srun -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 cmaketest
cd  cmaketest
```

2. 申请计算资源并加载所需模块：

```
sallc -p 64c512g -n 10
# salloc: Nodes nodexxx are ready for job
ssh nodexxx

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 cmaketest.cpp：

```
#include "mpi.h"
#include <iostream>
using namespace std;

int say_hello( int argc, char ** argv );

```

(下页继续)

(续上页)

```
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
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：

```
salloc -p cpu -N  1 --ntasks-per-node 40  --exclusive
#salloc: Nodes casxxx are ready for job
ssh casxxx

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

MATLAB Parallel Computing Toolbox

利用 Parallel Computing Toolbox™，可以使用多核处理器、GPU 和计算机集群来解决计算问题和数据密集型问题。利用并行 for 循环、特殊数组类型和并行化数值算法等高级别构造，无需进行 CUDA 或 MPI 编程即可对 MATLAB® 应用程序进行并行化。通过该工具箱可以使用 MATLAB 和其他工具箱中支持并行的函数。你可以将该工具箱与 Simulink 配合使用，并行运行一个模型的多个仿真。程序和模型可以在交互模式和批处理模式下运行。

集群上部署的 MATLAB 镜像均已安装 Parallel Computing Toolbox 并获取相关授权，打开 MATLAB 即可使用相应功能。

MATLAB Parallel Computing Toolbox 支持四种模式：

1. 运行后台进程
2. 分布式计算
3. 并行计算
4. *GPU* 计算

本文档将给出各种计算模式的示例。示例均以交互式命令行形式展现，需要先申请计算资源：

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

在计算节点上运行 MATLAB 镜像：

```
$ singularity run /dssg/share/imgs/matlab/matlab_latest.sif matlab
MATLAB is selecting SOFTWARE OPENGL rendering.

                                      < M A T L A B (R) >
                            Copyright 1984-2022 The MathWorks, Inc.
                          R2022a Update 2 (9.12.0.1956245) 64-bit␣
↪(glnxa64)
                                            May 11, 2022


To get started, type doc.
For product information, visit www.mathworks.com.

>>
```

更多的 MATLAB 使用方式，请参考文档*MATLAB*.

运行后台进程

在使用 MATLAB 交互式命令行的时候，可以让一个进程在后台运行，从而不影响继续使用交互式命令行。

请至 gabor_patch_avi 将该段代码拷贝至本地并命名为 *gabor_patch_avi.m*。这段代码耗时 10-60 秒，并将在本地生成一个 *.avi* 文件。

函数 *gabor_patch_avi* 可在后台被运行：

```
>> jid = batch('gabor_patch_avi');
>> ls
gabor_patch_avi.m
>> a=5;b=10;
>> a+b

ans =

    15

>> ls
098824838.avi  gabor_patch_avi.m
```

可以看到，*gabor_patch_avi* 函数和命令行命令并不是顺序执行的。

分布式计算

如果进程之间无需通信，那么可以使用分布式计算模式。

以下代码同时在后台运行 4 个 *gabor_patch_avi* 函数。

```
>> job_num=4;
>> clust_obj = parcluster;
>> clust_obj.NumWorkers = job_num;
>> job_obj = clust_obj.createJob;
>> for ii = 1:job_num
        job_obj.createTask(@gabor_patch_avi, 0);
   end
>> job_obj.submit;
```

并行计算

MATLAB 进程可以多核心并行、共享内存并且在线程间通信。

parfor

最简单的启用并行计算的方法是使用 MATLAB 的 *parfor* 关键字。

以下示例使用 10 个 cpu 核并行计算了 *y* 数列的值。

```
>>  pc = parcluster('local');
>> parpool(pc, 10);
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 10).
>> n = 2000;
>> y = zeros(n,1);
>> parfor i = 1:n
       y(i) = max(svd(randn(i)));
   end
>>
```

GPU 计算

GPU 计算需要先申请 GPU 计算资源：

```
$ srun -p a100 -N 1 --gres gpu:1 -n 12 --pty /bin/bash
```

使用如下命令启动 GPU 版本 MATLAB:

```
$ singularity run --nv /dssg/share/imgs/matlab/matlab_latest.sif␣
↪matlab
```

使用 *gpuArray* 将数据存入 GPU，即可在 GPU 上进行运算。使用 *gather* 可将数据从 GPU 重新传回 CPU:

```
>> X = [1,2,3];
>> G = gpuArray(X);
>> isgpuarray(G)

ans =

logical

1

>> GSq = G.^2;
>> XSq = gather(GSq)

XSq =

     1     4     9

>> isgpuarray(XSq)
```

(下页继续)

(续上页)

```
ans =

logical

0

>>
```

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

```
srun -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

```
srun -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
```

(下页继续)

(续上页)

```
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	cpu	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`
```

(下页继续)

(续上页)

```
export PETSC_ARCH=x86_64
python3 config/configure.py --with-shared-libararies=1 --with-x=0 --
 ↪with-mpi=1 --with-mpi-dir=/lustre/opt/cascadelake/linux-centos7-
 ↪cascadelake/gcc-9.2.0/mpich-3.4.1-
 ↪76cdhzxhzr7wp5kqkwirehvwpo7oe6lc --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_ENABLE_EpetraExt=ON -DTrilinos_ENABLE_Ifpack=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 .
 ↪.
```

(下页继续)

(续上页)

```
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 ..
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/

4.9.3 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	cpu	spack	oneapi/2021.4.0 思源一号
2.3	cpu	spack	oneapi/2021.4.0 π2.0
2.3	cpu	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 $MKLROOT/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
```

(下页继续)

(续上页)

```
├── runme_intel64_prv
├── run.slurm
├── xhpl_intel64_dynamic
└── xhpl_intel64_dynamic_outputs.txt
```

测试平台

- *π2.0*
- 思源一号平台
- *ARM* 平台

π2.0

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。

π2.0 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 `Ns` 调整至内存空间的 80% 左右 `0.8*sqrt(mem*1024*1024*1024*nodes/8)` 。本算例使用了两个节点，这里使用 sed> 将 Ns 替换为 176640。

```
$ sed -i -e 's/.*Ns.*/176640\ Ns/' HPL.dat
```

然后调整 HPL.dat 的 `Ps` 和 `Qs` 值，使其乘积等于 MPI 进程总数。这里使用 sed 将 `Ps` 和 `Qs` 值分别设置为 2、2，乘积等于线程总数 2。

```
$ sed -i -e 's/.*\ Ps.*/2\ Ps/' HPL.dat
$ sed -i -e 's/.*\ Qs.*/2\ Qs/' 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
```

(下页继续)

(续上页)

```
#SBATCH -n 4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=20
#SBATCH --exclusive

ulimit -s unlimited
ulimit -l unlimited

module load oneapi

./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
```

(下页继续)

(续上页)

```
#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 `Ns` 调整至内存空间 256G 的 80% 左右。这里使用 sed 将 Ns 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，将将 `Ps` 和 `Qs` 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
```

(下页继续)

(续上页)

```
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
--------------------------------------------------------------------------------
↪------------
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 Benchmarsk 模块，可用 `module load` 指令加载。

版本	平台	构建方式	模块名
5.7.1	cpu	spack	osu-micro-benchmarks/5.7.1-gcc-11.2.0-openmpi 思源一号
5.7.1	cpu	spack	osu-micro-benchmarks/5.7.1-gcc-9.2.0-openmpi-4.0.5
5.6.3	cpu	spack	osu-micro-benchmarks/5.6.3-gcc-9.3.0-openmpi

集群上的 **OSU**

- 思源一号上的 *OSU*
- *π2.0* 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
#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
```

(下页继续)

(续上页)

```
#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

#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
```

(下页继续)

(续上页)

```
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
#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
```

(下页继续)

(续上页)

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 ]
# 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
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
```

(下页继续)

(续上页)

```
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
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-complie-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	cpu	spack	oneapi/2021.4.0 思源一号
3.1	cpu	spack	oneapi/2021.4.0 π2.0

```
mkdir ~/HPCG && cd ~/HPCG
module load oneapi/2021.4.0
cp -r $MKLROOT/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

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-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
GFOLP/s	56.09485	112.07949

π2.0

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*
- *π2.0 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
```

(下页继续)

(续上页)

```
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
  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
```

(下页继续)

(续上页)

```
  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

π2.0 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

4.9.4 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

π2.0:
/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	gpu	容器	module load tensorflow/2.8.2 思源一号
2.4.1	gpu	容器	module load tensorflow/2.4.1 pi2.0

TensorFlow 使用教程

- 思源一号 *TensorFlow*
- *π2.0 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 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=gpu_py38h75b8afa_0
#上述命令会自动安装如下依赖
.cudatoolkit=11.3.1=h2bc3f7f_2
.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
```

(下页继续)

(续上页)

```
#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
```

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
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=gpu_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
#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)'
```

(下页继续)

(续上页)

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
```

(下页继续)

(续上页)

```
#SBATCH --gres=gpu:2

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
```

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
```

(下页继续)

(续上页)

```
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。

π 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 mypy

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 mypy

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
```

(下页继续)

(续上页)

```
#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	gpu	容器	/lustre/share/img/tvm-0.9.dev0.sif
1.6.dev0	gpu	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


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
#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

4.9.5 原子分子工程计算

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	cpu	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，需购买授权使用。

如需使用集群上的 Amber，请发邮件至 hpc 邮箱，附上课题组购买 Amber 的证明，并抄送超算帐号负责人。

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
```

π2.0 上的 Amber

π2.0 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
```

(下页继续)

(续上页)

```
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

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：

```
srun -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.
```

(下页继续)

(续上页)

```
    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.
    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.
```

(下页继续)

(续上页)

```
   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,
    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,
```

(下页继续)

(续上页)

```
    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
    62,      28,      48,     149,      44,      49,     155,  ␣
↪154,
    47,      50,     153
    63,      28,      34,      31,     149,      36,      35,  ␣
↪33,
   154,     156,     164
```

(下页继续)

(续上页)

```
    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
```

(下页继续)

(续上页)

```
 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
```

(下页继续)

(续上页)

```
 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
 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
```

(下页继续)

(续上页)

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：

```
srun -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
```

(下页继续)

(续上页)

```
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.
    """

    # 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')
```

(下页继续)

(续上页)

```
    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
#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, 404.61068703239 77)
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)
```

(下页继续)

(续上页)

```
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)
```

(下页继续)

(续上页)

```
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)
```

参考资料

- Cantera 官网
- 安装 Cantera 知乎

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

用户需要在自己的目录下编辑 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

```
srun -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/
./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
8.2	cpu	spack	cp2k/8.2-gcc-11.2.0-openblas-openmpi 思源一号
8.2	gpu	spack	cp2k/8.2-gcc-11.2.0-cuda-openblas-openmpi 思源一号
8.2	cpu	spack	cp2k/8.2-gcc-9.2.0-openblas
8.2	gpu	spack	cp2k/8.2-gcc-8.3.0-openblas
8.2	arm	spack	cp2k/8.2-gcc-9.3.0-openblas-openmpi

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** 示例脚本 **cp2k_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 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 --allow-run-as-root -np $SLURM_NTASKS -x OMP_NUM_THREADS=1␣
↪cp2k.psmp ${INPUT_FILE}
```

(下页继续)

(续上页)

思源一号上 **GPU** 版本 **cp2k** 示例脚本 **cp2k_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 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 --allow-run-as-root -np $SLURM_NTASKS --mca opal_common_ucx_
 ↪opal_mem_hooks 1 -x OMP_NUM_THREADS=1 cp2k.psmp ${INPUT_FILE}
```

并使用如下指令提交：

```
$ sbatch cp2k_cpu.slurm
$ sbatch cp2k_gpu.slurm
```

π 集群上的 CP2K

π 集群系统中已经预装 CP2K (GPU+CPU 版本)。若不指定版本，将加载默认的 module（标记为 D)。

π 集群上 CPU 版本 cp2k 示例脚本 cp2k_cpu.slurm

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 --job-name=cp2k_cpu_test
```

(下页继续)

(续上页)

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

module load cp2k/8.2-gcc-9.2.0-openblas
module load openmpi/4.0.5-gcc-9.2.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun --allow-run-as-root -np $SLURM_NTASKS -x OMP_NUM_THREADS=1␣
↪cp2k.psmp ${INPUT_FILE}
```

π 集群上 GPU 版本 cp2k 示例脚本 cp2k_gpu.slurm

```
#!/bin/bash

#SBATCH --job-name=cp2k_gpu_test
#SBATCH --partition=dgx2
#SBATCH --output=%j.out
#SBATCH --error=%j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=6
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:1


module load cp2k/8.2-gcc-8.3.0-openblas
module load cuda/10.1.243-gcc-8.3.0
module load openmpi/4.0.5-gcc-8.3.0

ulimit -s unlimited
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun --allow-run-as-root -np $SLURM_NTASKS --mca opal_common_ucx_
↪opal_mem_hooks 1 -x OMP_NUM_THREADS=1 cp2k.psmp ${INPUT_FILE}
```

并使用如下指令提交：

```
$ sbatch cp2k_cpu.slurm
$ sbatch cp2k_gpu.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
ulimit -l unlimited

INPUT_FILE=H2O-256.inp
mpirun --allow-run-as-root -np $SLURM_NTASKS -x OMP_NUM_THREADS=1␣
↪cp2k.psmp ${INPUT_FILE}
```

在 ARM 节点使用如下命令提交作业：

```
sbatch cp2k.slurm
```

参考资料

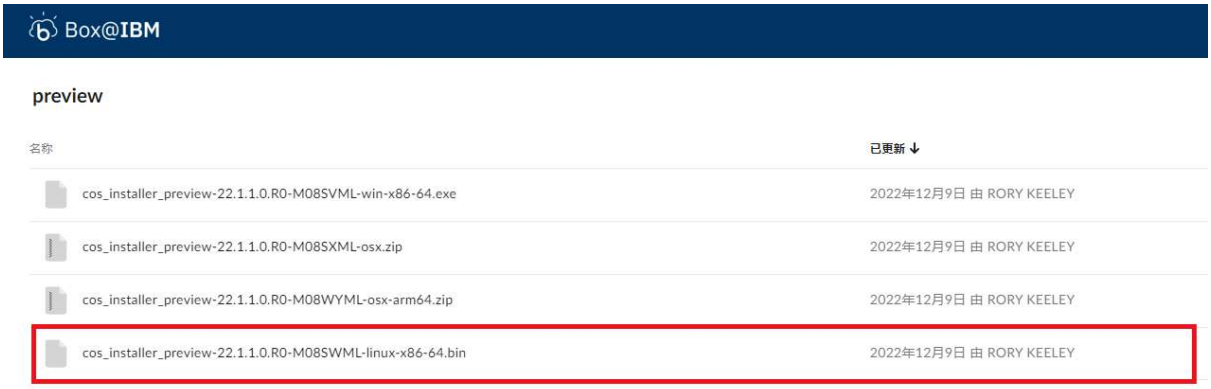
- CP2K 官网

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)：



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. 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
```

4. 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
```

5. 根据终端输出的安装提示一步一步往下执行即可。需要注意的是一定要将默认的安装路径 /opt/ibm/ILOG/CPLEX_Studio221 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)；

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 GfsGEdge {} {
GfsTime { end = 50 }
GfsRefine 6
GfsInit {} {
  U = (0.5 - rand()/(double)RAND_MAX)
  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
```

(下页继续)

(续上页)

```
#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
  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`

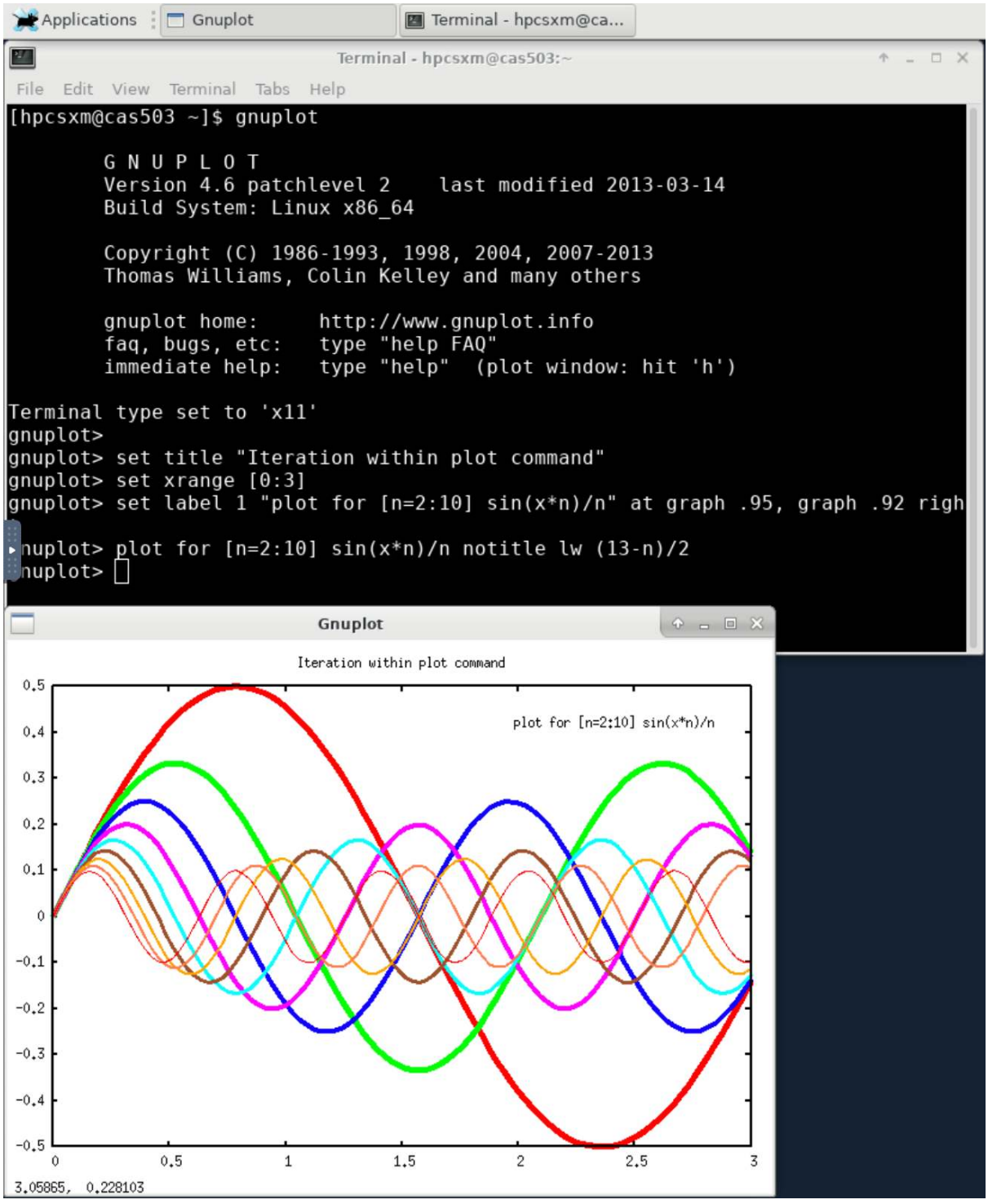
6. 在远程桌面里调用 Gnuplot

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)
```

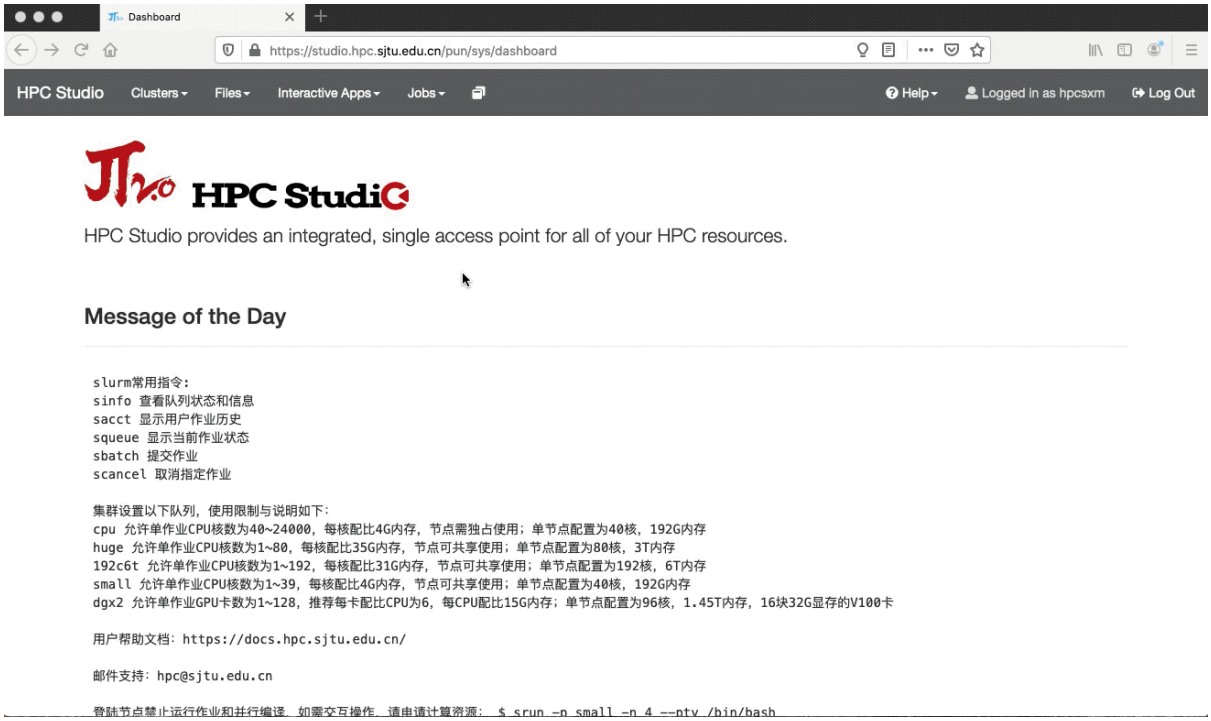
另一个稍复杂的作图示例：

```
$ gnuplot
gnuplot> set title "Iteration within plot command"
         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
```

(下页继续)

(续上页)

```
#SBATCH -N 1
#SBATCH -n 8                          # 核数 8
#SBATCH --ntasks-per-node=8           # 每节点核数
#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

版本	平台	构建方式	模块名
2022.3	gpu	spack	gromacs/2022.3-intel-2021.4.0-cuda 思源一号
2021.3	cpu	spack	gromacs/2021.3-intel-2021.4.0 思源一号
2021.3	cpu	spack	gromacs/ 2021.3-gcc-11.2.0-cuda-openblas-openmpi 思源一号
2021.2	gpu	spack	gromacs/2021.2-intel-2021.4.0-cuda 思源一号
2021.6	cpu	spack	gromacs/2021.6-gcc-9.2.0-openmpi-4.0.5
2019.4	cpu	spack	gromacs/2019.4-gcc-9.2.0-openmpi
2020	cpu	容器	gromacs/2020-cpu-double
2022	arm	源码	gromacs/2022-gcc-9.3.0

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. *π2.0 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/2022.3-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
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=16
#SBATCH --gres=gpu:1
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load oneapi
module load gromacs/2022.3-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:

```
[hpchgc@login water]$ tree 0768
0768
├── 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
```

(下页继续)

(续上页)

```
mpirun gmx_mpi mdrun -dlb yes -v -nsteps 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 %


               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. π2.0 GROMACS

1.gromacs/2021.6-gcc-9.2.0-openmpi-openmpi-4.0.5

提交预处理脚本

```
#!/bin/bash

#SBATCH -J gromacs_cpu_test
#SBATCH -p cpu
#SBATCH -n 40
```

(下页继续)

(续上页)

```
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module load gromacs/2021.6-gcc-9.2.0-openmpi-4.0.5
module load openmpi/4.0.5-gcc-9.2.0
module load gcc/9.2.0

ulimit -s unlimited
ulimit -l unlimited
gmx_mpi grompp -f pme.mdp
```

提交运行作业脚本

```
#!/bin/bash

#SBATCH -J gromacs_cpu_test
#SBATCH -p cpu
#SBATCH -n 40
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module load gromacs/2021.6-gcc-9.2.0-openmpi-4.0.5
module load openmpi/4.0.5-gcc-9.2.0
module load gcc/9.2.0

ulimit -s unlimited
ulimit -l unlimited
mpirun gmx_mpi mdrun -dlb yes -v -nsteps 10000 -resethway -
↪noconfout -pin on -ntomp 1 -s topol.tpr
```

2.gromacs/2020-cpu-double

提交预处理脚本

```
#!/bin/bash

#SBATCH -J gromacs_cpu_test
#SBATCH -p cpu
#SBATCH -n 40
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module load gromacs/2019.4-gcc-9.2.0-openmpi
```

(下页继续)

(续上页)

```
ulimit -s unlimited
ulimit -l unlimited
gmx_mpi grompp -f pme.mdp
```

提交运行作业脚本

```
#!/bin/bash

#SBATCH -J gromacs_cpu_test
#SBATCH -p cpu
#SBATCH -n 40
#SBATCH --ntasks-per-node=40
#SBATCH -o %j.out
#SBATCH -e %j.err

module load gromacs/2020-cpu-double

ulimit -s unlimited
ulimit -l unlimited
srun --mpi=pmi2 gmx_mpi_d mdrun -dlb yes -v -nsteps 10000 -
↪resethway -noconfout -pin on -ntomp 1 -s topol.tpr
```

三. ARM GROMACS

1.module load gromacs/2022-gcc-9.3.0

提交预处理脚本

```
#!/bin/bash

#!/bin/bash

#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gromacs/2022-gcc-9.3.0

gmx_mpi grompp -f pme.mdp
```

提交运行作业脚本

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

(下页继续)

(续上页)

```
#SBATCH --job-name=test
#SBATCH --partition=arm128c256g
#SBATCH -N 2
#SBATCH --ntasks-per-node=128
#SBATCH --exclusive
#SBATCH --output=%j.out
#SBATCH --error=%j.err

module load gromacs/2022-gcc-9.3.0
export OMP_NUM_THREADS=1
mpirun gmx_mpi mdrun -dlb yes -v -nsteps 10000 -resethway -
↪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/2021.6-gcc-9.2.0-openmpi-4.0.5			
核数	40	80	120
Performance	8.584	17.266	30.913

gromacs/2020-cpu-double			
核数	40	80	120
Performance	4.441	8.388	16.701

3.GROMACS ARM

gromacs/2022-gcc-9.3.0			
核数	128	256	512
Performance	7.754	15.466	30.650

参考资料

- gromacs 官方网站 http://www.gromacs.org/

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。

可用的版本

版本	平台	构建方式	模块名
2022	cpu	源码	lammps/20220324-intel-2021.4.0-omp 思源一号
2021	cpu	源码	lammps/20210310-intel-2021.4.0-omp 思源一号
2020	cpu	容器	直接使用镜像
2019	arm	容器	lammps/bisheng-1.3.3-lammps-2019

集群上的 **LAMMPS**

- *思源一号 LAMMPS*
- *π2.0 LAMMPS*
- *ARM LAMMPS*

一. 思源一号 LAMMPS

1. 全局部署版本 20220324 20210310

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 lammps/20220324-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/20220324-intel-2021.4.0-omp

mpirun lmp -i in.lj
```

1. 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，并请注意在全部编译结束后退出：

```
srun -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

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. π2.0 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/20220324-oneapi-2021.4.0

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 small -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
```

(下页继续)

(续上页)

```
$ 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

#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/20220324-intel-2021.4.0-omp			
核数	64	128	192
Wall time	0:01:28	0:00:45	0:00:37

lammps/20210310-intel-2021.4.0-omp			
核数	64	128	192
Wall time	0:01:26	0:00:46	0:00:36

π2.0

intel 加速版本			
核数	40	80	120
Wall time	0:02:37	0:01:16	0:00:52

ARM

lammps/bisheng-1.3.3-lammps-2019		
核数	64	96
Wall time	0:07:26	0:04:43

建议

通过分析上述结果，我们推荐您使用如下两个版本提交作业。

```
module load lammps/20210310-intel-2021.4.0-omp                ␣
↪思源一号
/lustre/share/singularity/modules/lammps/20-user-intel.sif    π2.0
```

算例内容如下：***in.lj***

```
# 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

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	cpu	源码	lammps-rbe/20190807-oneapi-2021.4 思源一号
7Aug2019	cpu	源码	lammps-rbe/ 20190807-intel-parallel-studio-2020.1-impi π2.0
7Aug2019	cpu	容器	lammps-rbe/20190807-oneapi-2021.1-impi π2.0

算例获取

π2.0 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*
- *π2.0 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-impi π2.0

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-impi
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

π2.0 上的结果

lammps-rbe/20190807-intel-parallel-studio-2020.1-impi			
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 Lammps 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 Lammps 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 Lammps 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 erfc(r_cut*sqrt{alpha})≈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 Lammps 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 Lammps 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 (ε_in-ε_top)/ (ε_in+ε_top) 和 (ε_in-ε_down)/ (ε_in+ε_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。

kspace_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/（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

kspace_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)

$$\frac{\partial u}{\partial t} + \mathrm{V}_x \frac{\partial u}{\partial x} + \mathrm{V}_y \frac{\partial u}{\partial y} = 0,$$

$$u(x, y;\ t = 0) = \sin(\kappa x)\cos(\kappa y),$$

$$u(x_b = [-1, 1], y_b;\ t) = \text{periodic},$$

$$u(x_b, y_b = [-1, 1];\ t) = \sin(\kappa(x - \mathrm{V}_x\, t))\cos(\kappa(y - \mathrm{V}_y\, t)),$$

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

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
```

(下页继续)

(续上页)

```
               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"
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
```

(下页继续)

(续上页)

```
#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
             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:
```

(下页继续)

(续上页)

```
=============================================================================
```

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
```

(下页继续)

(续上页)

```
       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
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. 执行以下从命令从 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
ccmake ../
make
make install
```

参考资料

- Nektar 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	cpu	Spack	nwchem/6.8.1-intel-19.0.4-impi

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-impi

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
```

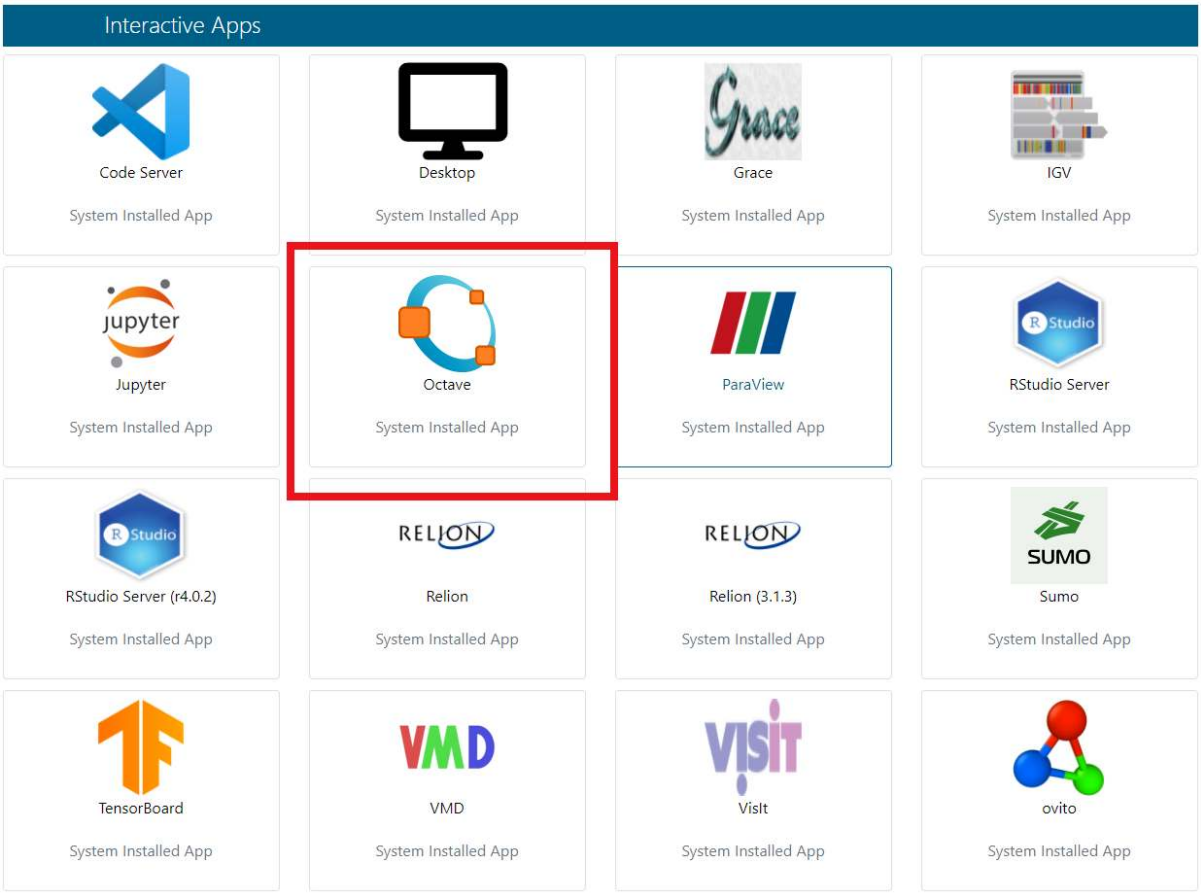
使用 **singularity** 容器提交 **Octave** 交互式作业

可以通过如下指令提交 Octave 交互式作业：

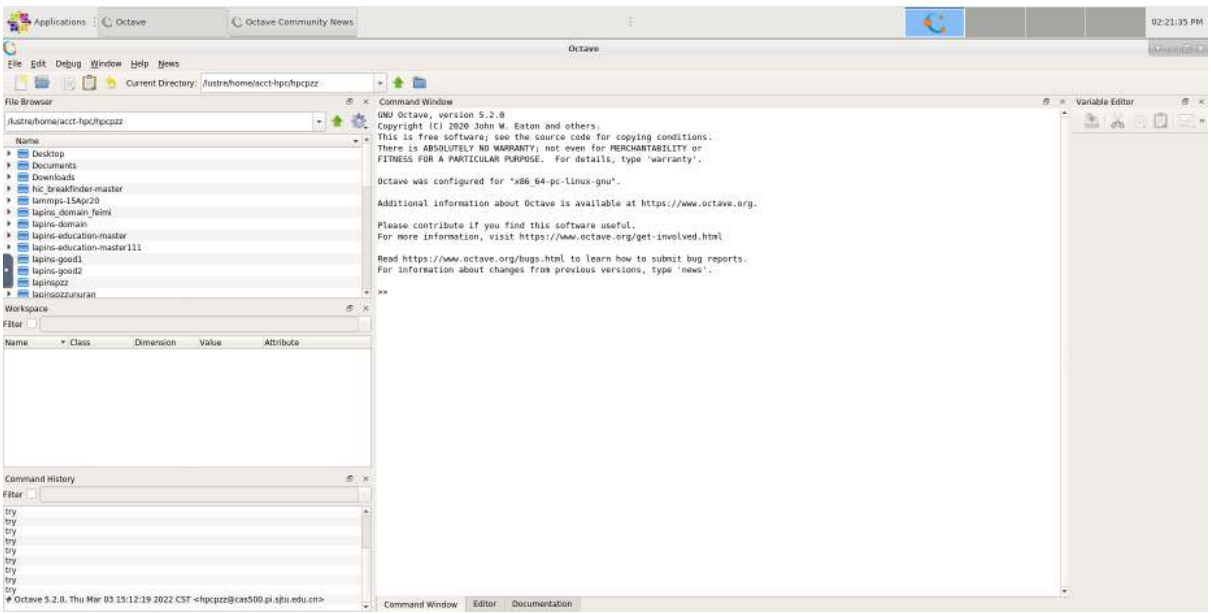
```
srun -p cpu -N 1 --exclusive --pty singularity run /lustre/share/
↪img/octave.simg octave-cli
```

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):



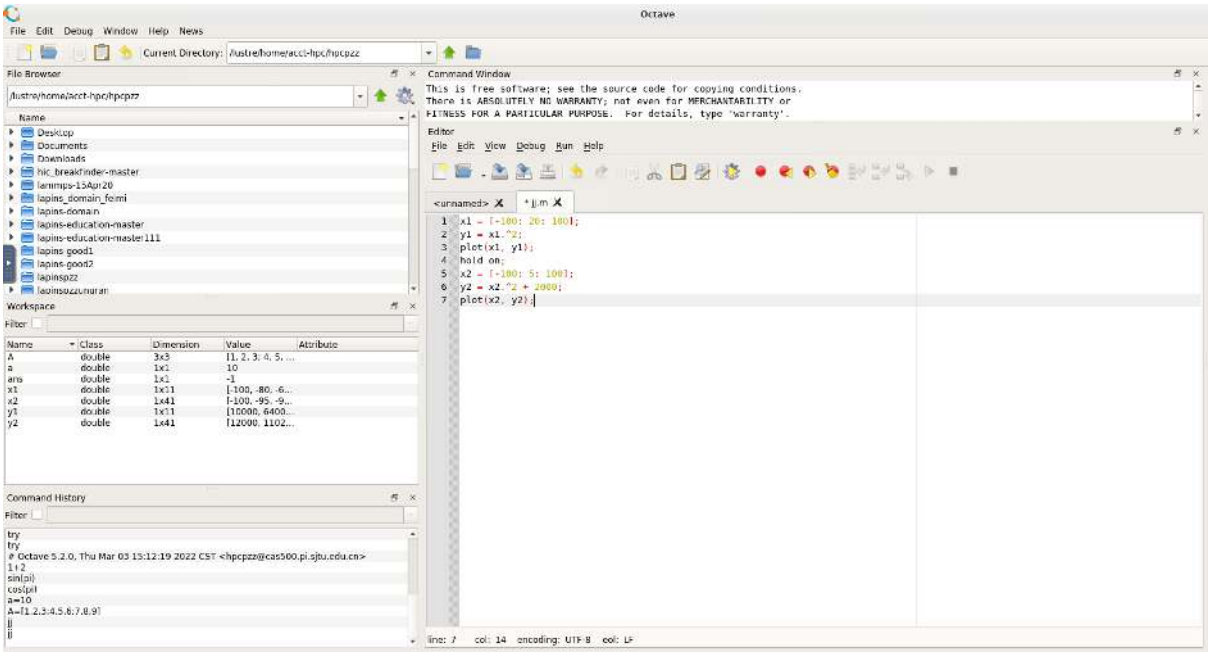
3. 申请成功后会看到如下界面:



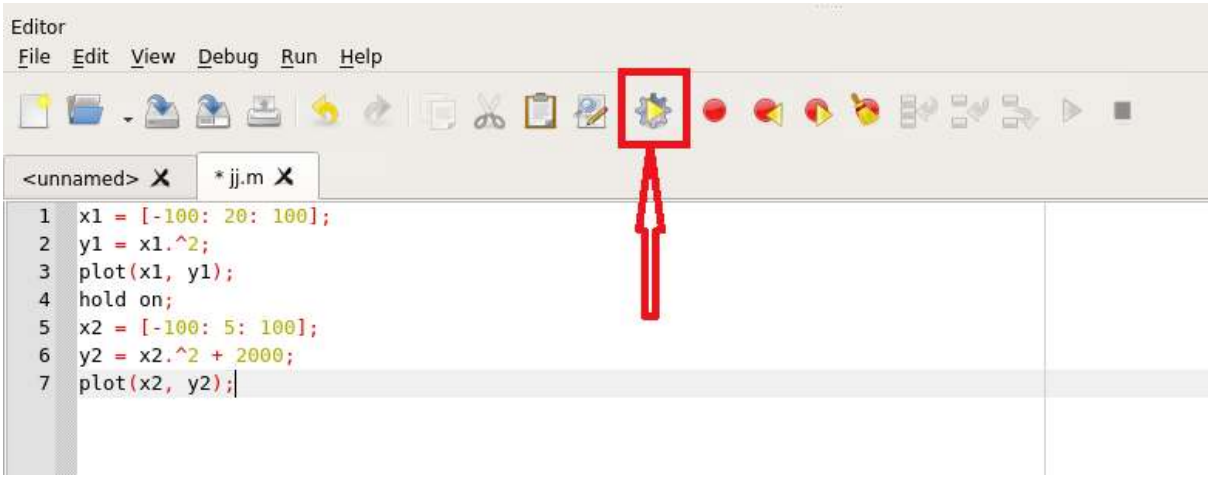
4. Octave 的语法和 Matlab 几乎完全相同，我们可以在右侧 Command Window 窗口中进行简单的数学运算，如下图所示：

```
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

>>
```

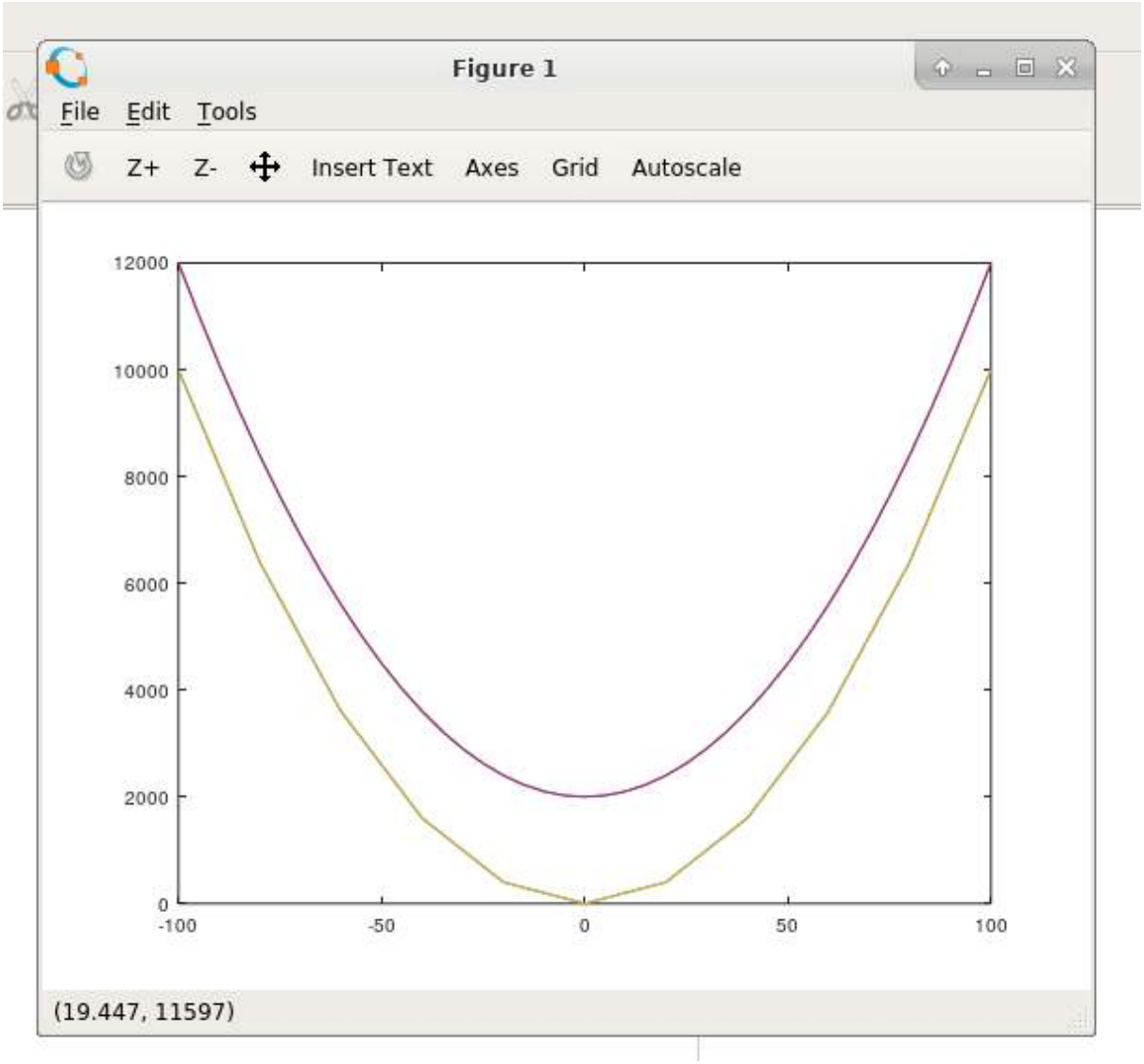
5. 如果要编写脚本进行比较复杂的计算的话 (绘图或者是其他)，可以点击左上角 File->New->New Script 选项，然后在弹出的 Editor 窗口进行脚本编写，如下图所示：



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 is the publicly available open-source code base that a worldwide community of researchers, software developers, and industry leaders are enhancing every day. OpenRadioss is changing the game by empowering users to make rapid contributions that tackle the latest challenges brought on by rapidly evolving technologies like battery development, lightweight materials and composites, human body models and biomaterials, autonomous driving and flight, as well as the desire to give passengers the safest environment possible via virtual testing.

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:

```
srun -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 -xzvf 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:

```
srun -p cpu -N  1 --ntasks-per-node 40  --exclusive  --pty /bin/bash
module load netcdf-fortran/4.5.2-gcc-8.3.0-openmpi
```

(下页继续)

(续上页)

```
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-
↪vpswzpisyoc6gl6e5isbal66yykxdc6k -mpi-include=/lustre/opt/
↪cascadelake/linux-centos7-cascadelake/gcc-9.2.0/openmpi-4.0.5-
↪vpswzpisyoc6gl6e5isbal66yykxdc6k/include -mpi-libdir=/lustre/opt/
↪cascadelake/linux-centos7-cascadelake/gcc-9.2.0/openmpi-4.0.5-
↪vpswzpisyoc6gl6e5isbal66yykxdc6k/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:

```
srun -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

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
```

(下页继续)

(续上页)

```
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
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-mkqkyei3gxxtri4uaafreqteyhyj2exl
LAPACK_DIR = BUILD
```

(下页继续)

(续上页)

```
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 =
```

(下页继续)

(续上页)

```
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
```

(下页继续)

(续上页)

```
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 =/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	cpu	Spack	openfoam-org/7-gcc-7.4.0-openmpi
8	cpu	容器	/lustre/share/img/x86/openfoam/8-gcc8-openmpi4-centos8.sif
1712	cpu	Spack	openfoam/1712-gcc-7.4.0-openmpi
1912	cpu	Spack	openfoam/1912-gcc-7.4.0-openmpi
2012	cpu	容器	/lustre/share/img/x86/openfoam/2012-gcc4-openmpi4-centos7.sif
2106	cpu	容器	/lustre/share/img/x86/openfoam/2106-gcc4-openmpi4-centos7.sif
8	arm	容器	/lustre/share/img/aarch64/openfoam/8-gcc8-openmpi4-centos8.sif
1912	arm	Spack	openfoam/1912-gcc-9.3.0-openmpi
2012	arm	容器	/lustre/share/img/aarch64/openfoam/2012-gcc4-openmpi4-centos7.sif
2106	arm	容器	/lustre/share/img/aarch64/openfoam/2106-gcc4-openmpi4-centos7.sif
1912	arm	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
```

(下页继续)

(续上页)

```
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
```

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
```

(下页继续)

(续上页)

```
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

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_PROJECT_DIR/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
```

(下页继续)

(续上页)

```
└── 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
│   └── U
├── 0.1
│   ├── p
│   ├── phi
│   ├── U
│   └── uniform
│       └── time
├── 0.2
│   ├── p
```

(下页继续)

(续上页)

```
|    ├── 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
   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
```

(下页继续)

(续上页)

```
   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 '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

   export PATH=$MPI_BIN:$PATH
   export LD_LIBRARY_PATH=$MPI_LIB:$LD_LIBRARY_PATH

%test
   . /opt/OpenFOAM/OpenFOAM-7/etc/bashrc
   icoFoam -help
```

(下页继续)

(续上页)

```
%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
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

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *␣
↪* * * * * //
```

(下页继续)

(续上页)

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

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 $HOME
mkdir OpenFOAM
cd OpenFOAM
git clone https://github.com/OpenFOAM/OpenFOAM-10.git
git clone https://github.com/OpenFOAM/ThirdParty-10.git
source OpenFOAM-10/etc/bashrc
cd OpenFOAM-10
./Allwmake -j
sed -i '$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。

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。

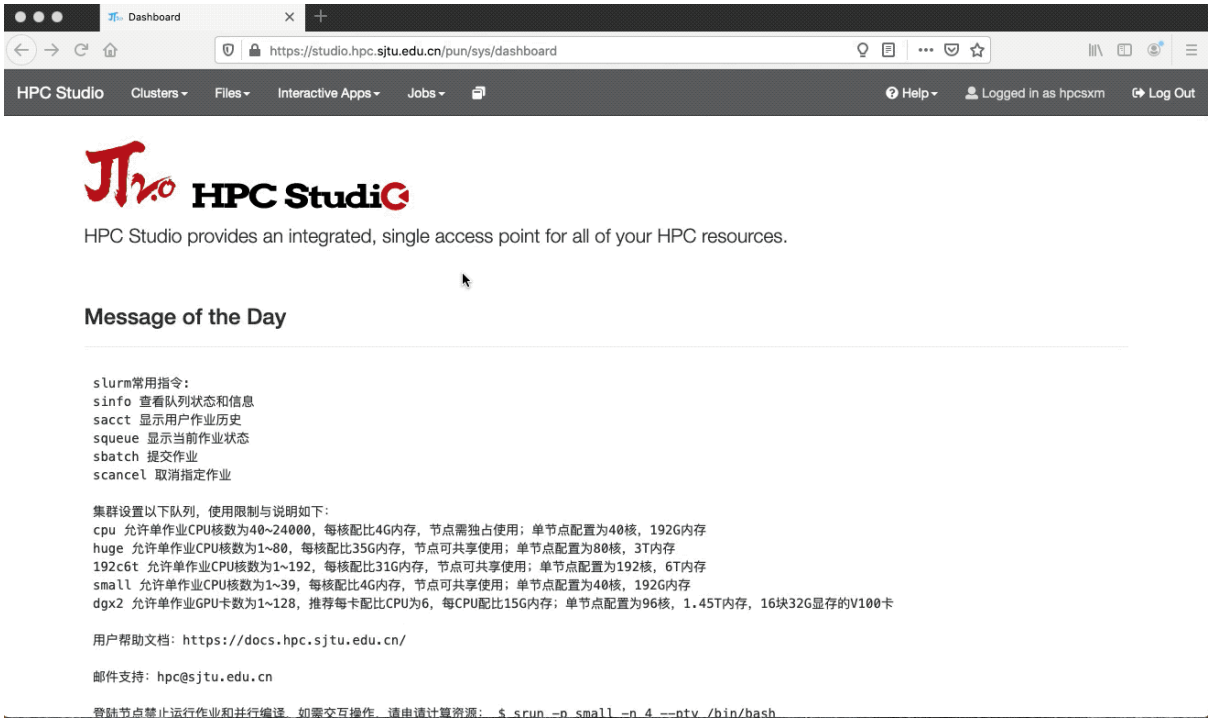
方法二：本地 OVITO 调用远程构型文件

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：

```
srun -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
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
7.1	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
7.1	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
```

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
- *π2.0* 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
│   ├── 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. π2.0 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
```

(下页继续)

(续上页)

```
#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-impi

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 ./qpic.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
```

(下页继续)

(续上页)

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

ulimit -s unlimited
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。

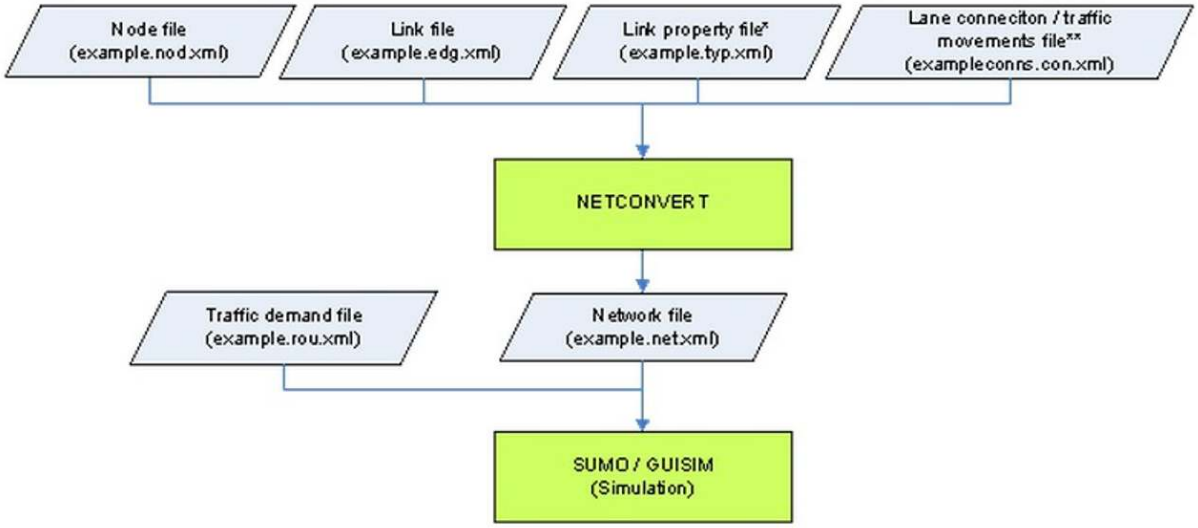
SUMO(GUI) 使用方法

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"/>
<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。

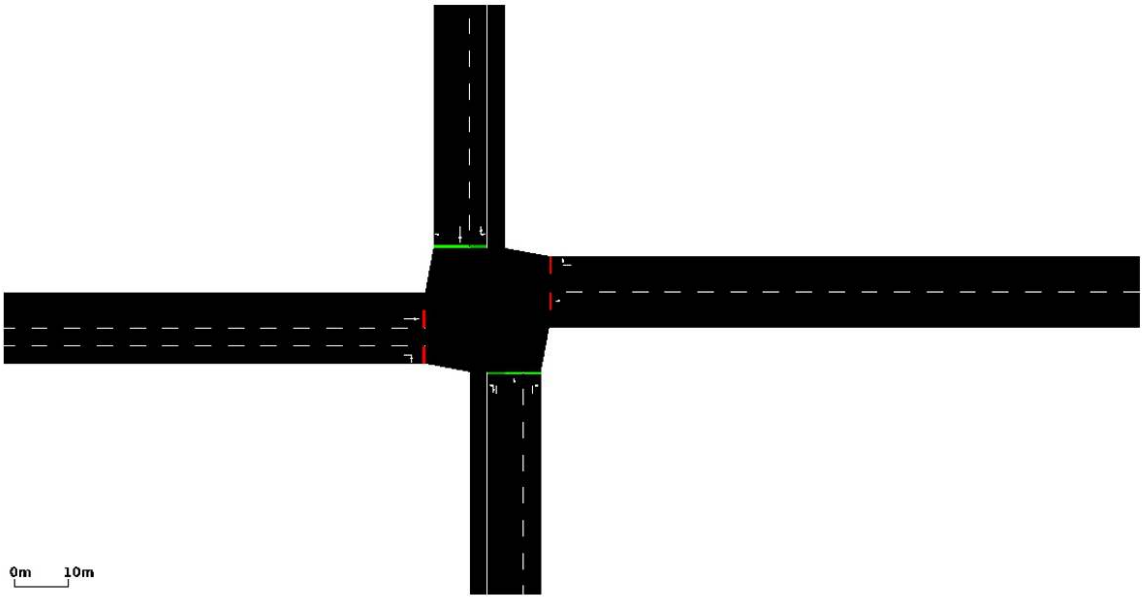
4. 编写 exa.con.xml 文件：

```
   <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" />
<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。

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。如需协助安装或使用，请发邮件联系我们，附上课题组拥有 VASP license 的证明。

本文档将介绍如何使用集群上已部署的 VASP 5.4.4 和 6.2.1，以及如何自行编译 VASP。

集群上的 **VASP**

- 思源一号 *VASP*
- *π2.0 VASP*
- *ARM VASP*

思源一号 **VASP**

若已拥有 VASP license，请邮件联系我们，提供课题组拥有 VASP license 的证明，并注明是 VASP5 还是 VASP6，我们将添加该 VASP module 的使用权限

经测试，思源一号使用默认的 `OMP_NUM_THREADS=1` 速度比其它设置更好，故无需额外设置该参数

下面 slurm 脚本以 vasp 6.2.1 为例。若使用 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
```

(下页继续)

(续上页)

```
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module load vasp/6.2.1-intel-2021.4.0-cuda-11.5.0

ulimit -s unlimited

mpirun vasp_std
```

π2.0 VASP

若已拥有 VASP license，请邮件联系我们，提供课题组拥有 VASP license 的证明，并注明是 VASP5 还是 VASP6，我们将添加该 VASP module 的使用权限

请注意，π2.0 上推荐使用 `OMP_NUM_THREADS=2`，速度较默认的 `OMP_NUM_THREADS=1` 提升近 20%

slurm 里，若使用 CPU 节点，须确保 `OMP_NUM_THREADS * ntasks-per-node` = **总核数**。例如：

- 1 个 CPU 节点，`OMP_NUM_THREADS=2`，`ntasks-per-node=20`
- 2 个 CPU 节点，`OMP_NUM_THREADS=2`，`ntasks-per-node=40`

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=20
#SBATCH --exclusive
#SBATCH -o %j.out
#SBATCH -e %j.err

module use /lustre/share/singularity/commercial-app
module load vasp/5.4.4-intel

ulimit -s unlimited
ulimit -l unlimited

export OMP_NUM_THREADS=2

srun --mpi=pmi2 vasp_std
```

ARM VASP

若已拥有 VASP license，请邮件联系我们，提供课题组拥有 VASP license 的证明，并注明是 VASP5 还是 VASP6，我们将添加该 VASP module 的使用权限

```
#!/bin/bash

#SBATCH -J vasp_arm
#SBATCH -p arm128c256g
#SBATCH -o %j.out
#SBATCH -e %j.err
#SBATCH -N 1
#SBATCH --ntasks-per-node=128

module load openmpi/4.0.3-gcc-9.2.0
mpirun singularity exec /lustre/share/singularity/commercial-app/
↪vasp/5.4.4-arm.sif 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。下面以思源一号为例，介绍安装和使用方法。

1. 先申请计算节点，然后加载 intel 套件

```
srun -p 64c512g -n 4 --pty /bin/bash        # 申请计算节点

module load intel-oneapi-compilers/2021.4.0
module load intel-oneapi-mpi/2021.4.0
module load intel-oneapi-mkl/2021.4.0
```

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.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 ~/bin        # 若 home 下未曾建过␣
↪bin，则新建一个；若已有，请略过此句
cp 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 load intel-oneapi-compilers/2021.4.0
module load intel-oneapi-mpi/2021.4.0
module load intel-oneapi-mkl/2021.4.0

ulimit -s unlimited

mpirun ~/vasp_std
```

VASP 算例及测试

以 64 原子的 Si AIMD 熔化为例，各使用 40 核，思源一号与 π 2.0 的测试结果：

setting	思源一号 40 核	π 2.0 40 核
OMP_NUM_THREADS	CPU time used (sec)	CPU time used (sec)
1	19	94
2	23	31
4	39	39

测试结果说明：

- 思源一号推荐使用 `OMP_NUM_THREADS=1`
- π 2.0 推荐使用 `OMP_NUM_THREADS=2`
- 思源一号 VASP 计算速度明显优于 π 2.0

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 slurm.sub
```

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

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

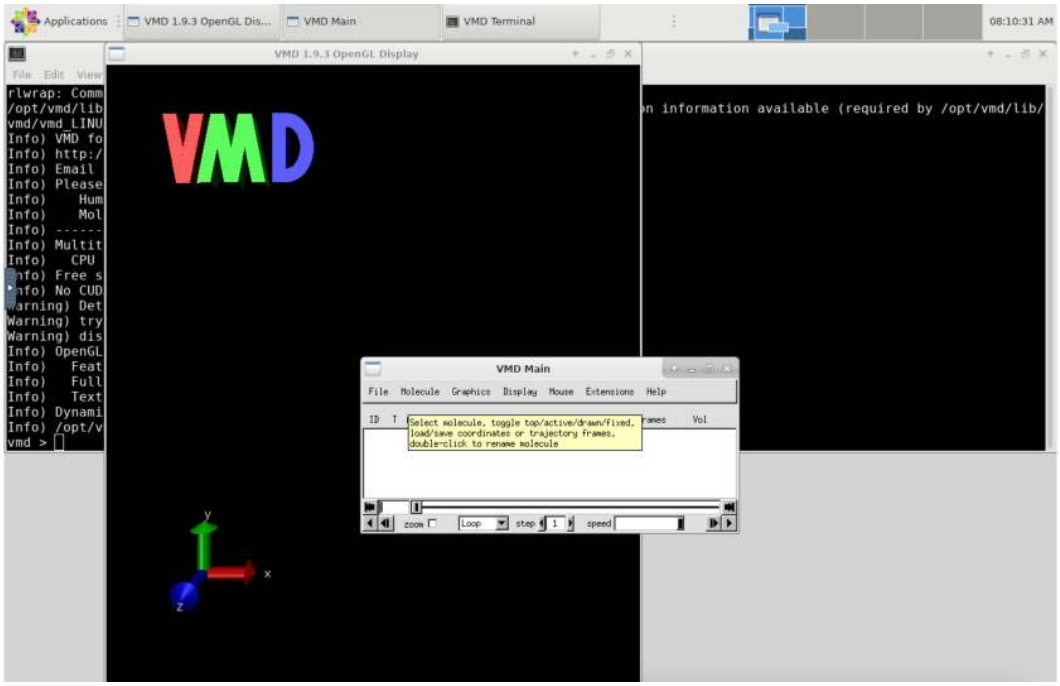
Visual Molecular Dynamics is a molecular modelling and visualization computer program. VMD is developed as mainly a tool to view and analyze the results of molecular dynamics simulations. It also includes tools for working with volumetric data, sequence data, and arbitrary graphics objects.

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	cpu	源码	wrf/4.2.1-oneapi-2021.4.0 思源一号
4.2	cpu	源码	wps/4.2-oneapi-2021.4.0 思源一号
4.3.1	cpu	源码	wrf_cmaq/5.3.3-wrf-4.3.1

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
```

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

```
hdf5-1.12.0
libpng-1.6.37
netcdf-c-4.8.1
```

(下页继续)

(续上页)

```
netcdf-fortran-4.5.3
zlib-1.2.11
jasper-1.900.29
```

自定义编译 WRF

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 DIR=/dssg/home/acct-hpc/hpchgc/software/wrf/WRF_4.1.1_Intel/
↪library
export LD_LIBRARY_PATH=$DIR/wrf_libs_intel/lib:$LD_LIBRARY_PATH
export LDFLAGS=-L$DIR/wrf_libs_intel/lib
export CPPFLAGS=-I$DIR/wrf_libs_intel/include
export NETCDF=$DIR/wrf_libs_intel/
export HDF5=$DIR/wrf_libs_intel/
export NETCDF=$DIR/wrf_libs_intel/
export HDF5=$DIR/wrf_libs_intel/
```

```
tar xvf v4.2.1.tar.gz
cd WRF-4.2.1/
./configure
```

可根据所需选择相应的参数，思源一号上的预编译版本选择的是 20 号，使用 intel 编译器编译 WRF，并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)
  5. (serial)   6. (smpar)   7. (dmpar)   8. (dm+sm)   PGI (pgf90/
↪pgcc): SGI MPT
  9. (serial)  10. (smpar)  11. (dmpar)  12. (dm+sm)   PGI (pgf90/
↪gcc): PGI accelerator
 13. (serial)  14. (smpar)  15. (dmpar)  16. (dm+sm)   INTEL (ifort/
↪icc)
                                        17. (dm+sm)   INTEL (ifort/
↪icc): Xeon Phi (MIC architecture)
 18. (serial)  19. (smpar)  20. (dmpar)  21. (dm+sm)   INTEL (ifort/
↪icc): Xeon (SNB with AVX mods)
 22. (serial)  23. (smpar)  24. (dmpar)  25. (dm+sm)   INTEL (ifort/
↪icc): SGI MPT
 26. (serial)  27. (smpar)  28. (dmpar)  29. (dm+sm)   INTEL (ifort/
↪icc): IBM POE
```

(下页继续)

(续上页)

```
 30. (serial)               31. (dmpar)                PATHSCALE␣
↪(pathf90/pathcc)
 32. (serial)  33. (smpar)  34. (dmpar)  35. (dm+sm)   GNU␣
↪(gfortran/gcc)
 36. (serial)  37. (smpar)  38. (dmpar)  39. (dm+sm)   IBM (xlf90_r/
↪cc_r)
 40. (serial)  41. (smpar)  42. (dmpar)  43. (dm+sm)   PGI (ftn/
↪gcc): Cray XC CLE
 44. (serial)  45. (smpar)  46. (dmpar)  47. (dm+sm)   CRAY CCE␣
↪(ftn $(NOOMP)/cc): Cray XE and XC
 48. (serial)  49. (smpar)  50. (dmpar)  51. (dm+sm)   INTEL (ftn/
↪icc): Cray XC
 52. (serial)  53. (smpar)  54. (dmpar)  55. (dm+sm)   PGI (pgf90/
↪pgcc)
 56. (serial)  57. (smpar)  58. (dmpar)  59. (dm+sm)   PGI (pgf90/
↪gcc): -f90=pgf90
 60. (serial)  61. (smpar)  62. (dmpar)  63. (dm+sm)   PGI (pgf90/
↪pgcc): -f90=pgf90
 64. (serial)  65. (smpar)  66. (dmpar)  67. (dm+sm)   INTEL (ifort/
↪icc): HSW/BDW
 68. (serial)  69. (smpar)  70. (dmpar)  71. (dm+sm)   INTEL (ifort/
↪icc): KNL MIC
 72. (serial)  73. (smpar)  74. (dmpar)  75. (dm+sm)   FUJITSU␣
↪(frtpx/fccpx): FX10/FX100 SPARC64 IXfx/Xlfx

Enter selection [1-75] :
```

根据个人所需可选择 mpi 进行编译，思源一号部署的预编译版本的更改参数如下：

更改文件configure.wrf**的参数**

```
DM_FC           =       mpiifort
DM_CC           =       mpiicc
```

自定义编译 WPS

导入如下环境变量

```
export WRF_DIR=../WRF-4.2.1/
export JASPERLIB=$DIR/wrf_libs_intel/lib/
export JASPERINC=$DIR/wrf_libs_intel/include/
```

```
tar xvf v4.2.tar.gz
cd WPS-4.2/
./configure
```

根据个人所需选择所需版本，思源一号上部署的预编译版本选择的 19 号，可多节点并行运行。(一般情况下选择 17 串行版即可满足计算所需)

```
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)
  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)

Enter selection [1-40] :
```

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,
 !
 !!!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT NOTE !!!!!!!!!!!!!!!!!!!!!!
↪!!!!!
 ! 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',
```

(下页继续)

(续上页)

```
 !
 !!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT NOTE !!!!!!!!!!!!!!!!!!!!!!
↪!!!!!
 !
 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/'
/

&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
```

(下页继续)

(续上页)

```
#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
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
/
```

(下页继续)

(续上页)

```
&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
/

&namelist_quilt
nio_tasks_per_group = 0,
```

(下页继续)

(续上页)

```
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
#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 !!!!!!!!!!!!!!!!!!!!!!
 ↪!!!!!
 ! 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',
 !
 !!!!!!!!!!!!!!!!!!!!!!!!!! IMPORTANT NOTE !!!!!!!!!!!!!!!!!!!!!!
 ↪!!!!!
 !
 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,
```

(下页继续)

(续上页)

```
 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_cmaq/5.3.3-wrf-4.3.1

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_cmaq/5.3.3-wrf-4.3.1

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
```

(下页继续)

(续上页)

```
module load wrf_cmaq/5.3.3-wrf-4.3.1

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_cmaq/5.3.3-wrf-4.3.1

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,
```

(下页继续)

(续上页)

```
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'
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,
```

(下页继续)

(续上页)

```
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=cpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=40
#SBATCH --output=%j.out
#SBATCH --error=%j.err
module load wrf_cmaq/5.3.3-wrf-4.3.1
ulimit -s unlimited
```

(下页继续)

(续上页)

```
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_cmaq/5.3.3-wrf-4.3.1
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_cmaq/5.3.3-wrf-4.3.1			
核数	40	80	160
Exec time	1:10:28	0:42:22	0:26:01

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
/

&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
```

(下页继续)

(续上页)

```
      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

module load gcc/8.3.0
module load openmpi/4.0.4-gcc-8.3.0
module load netlib-lapack/3.8.0-gcc-8.3.0
module load openblas/0.3.7-gcc-8.3.0
module load elphbolt/1.0.0-gcc-8.3.0-openmpi-4.0.4

cafrun -n 2 elphbolt.x
```

5. 使用如下指令提交作业：

```
sbatch elphbolt.slurm
```

6. 作业完成后可在.out 文件中看到如下结果 (部分):

```
+-------------------------------------------------------------------
 ↪------+
| \                                                                 ␣
 ↪       |
|  \                                                                ␣
 ↪       |
|   \   \                                                          ␣
 ↪       |
|    \   \                                                         ␣
 ↪       |
|   __\   \               _         _    _            _    _      ␣
 ↪       |
|   \      \          ___|.|       |.|  | |__   ___ |.|_ / /__    ␣
 ↪       |
|    \    __\        / _ \.|   _   |.|_ | '_ \ / _ \|.|_  ___/    ␣
 ↪       |
|     \  \          |  __/.| |/ \_|/  \| |_) : (_) |.|/ /__       ␣
 ↪       |
|      \ \           \___|_|/|__/ |   /| ___/ \___/|_|\___/       ␣
 ↪       |
|       \ \                 /|                                     ␣
 ↪       |
|        \\                 \|                                     ␣
 ↪       |
|          \\                '                                     ␣
 ↪       |
|           \                                                      ␣
 ↪       |
|            \                                                     ␣
 ↪       |
| A solver for the coupled electron-phonon Boltzmann transport␣
 ↪equations. |
| Copyright (C) 2020- Nakib Haider Protik.                         ␣
 ↪       |
|                                                                  ␣
 ↪       |
| This is a 'free as in freedom'[*] software, distributed under the␣
 ↪GPLv3.|
| [*] https://www.gnu.org/philosophy/free-sw.en.html               ␣
 ↪       |
+-------------------------------------------------------------------
 ↪------+

Number of coarray images =      2
-------------------------------------------------------------------
 ↪--------
```

(下页继续)

(续上页)

```
                                                          Setting up␣
 ↪crystal...
Material: Cubic Si
Isotopic average of masses will be used.
Si mass =   0.28085510E+02 u
Lattice vectors [nm]:
-0.27010011E+00   0.00000000E+00   0.27010011E+00
0.00000000E+00   0.27010011E+00   0.27010011E+00
-0.27010011E+00   0.27010011E+00   0.00000000E+00
Primitive cell volume =  0.39409804E-01 nm^3
Reciprocal lattice vectors [1/nm]:
-0.11631216E+02  -0.11631216E+02   0.11631216E+02
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
```

(下页继续)

(续上页)

```
_____
↪_____
_____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	cpu	容器	gmx_mmpbsa/1.5.2-gcc-9.3.0 思源一号
2020	cpu	容器	gmx_MMPBSA/1.4.3-gcc-9.3.0-ambertools-20-gromacs2021

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*
- *π2.0 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
```

(下页继续)

(续上页)

```
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
```

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	cpu	源码	cmaq/5.3.2-oneapi-2021.4.0 思源一号

算例获取路径

```
/dssg/share/sample/cmaq/CMAQv5.3.2_Benchmark_2Day_Input.tar.gz
```

集群上的 **CMAQ**

- 思源一号上的 *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
```

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
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/hpchgc/data/cmaq/cmaq_test2/CMAQ_
↪Project/data/2016_12SE1
csh run_cctm_Bench_2016_12SE1.csh
```

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

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
```

π2.0

```
/lustre/share/samples/thirdorder
```

集群上的 **thirdorder**

- 思源一号上的 *thirdorder*
- *π2.0* 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	cpu	源码	bte/1.0-openmpi-3.1.5 π2.0

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
#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

DAMSK

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	cpu	spack	damask/ 3.0.0-alpha5-gcc-11.2.0-hdf5-openblas-openmpi 思源一号
3.0.0-alpha6	cpu	spack	damask/ 3.0.0-alpha6-gcc-11.2.0-hdf5-openblas-openmpi 思源一号
3.0.0-alpha7	cpu	spack	damask/ 3.0.0-alpha7-gcc-11.2.0-hdf5-openblas-openmpi 思源一号
3.0.0-alpha5	cpu	spack	damask/3.0.0-alpha5-gcc-8.3.0-openblas

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
```

(下页继续)

(续上页)

```
#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
szip           : 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/ROMSRely/netcdf
szip           : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.
↪4.0/ROMS/ROMSRely/netcdf
hdf5           : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.
↪4.0/ROMS/ROMSRely/netcdf
netcdf-c       : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.
↪4.0/ROMS/ROMSRely/netcdf
netcdf-fortran: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.
↪4.0/ROMS/ROMSRely/netcdf
parallelNetcdf: /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.
↪4.0/ROMS/ROMSRely/pnetcdf
parallelIO     : /dssg/opt/icelake/linux-centos8-icelake/oneapi-2021.
↪4.0/ROMS/ROMSRely/pio

π2.0
zlib           : /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/
↪netcdf
szip           : /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/
↪netcdf
hdf5           : /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/
↪netcdf
netcdf-c       : /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/
↪netcdf
netcdf-fortran: /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/
↪netcdf
parallelNetcdf: /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/
↪pnetcdf
```

(下页继续)

(续上页)

```
parallelIO     : /lustre/opt/contribute/cascadelake/ROMS/ROMSRely/pio
```

集群上如何使用 ROMS

- 思源一号自定义编译 *ROMS*
- *π2.0* 自定义编译 *ROMS*
- 思源一号预编译 *ROMS*
- *π2.0* 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
```

```
srun -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	cpu	spack	proj/7.2.1-gcc-11.2.0 思源一号
7.2.1	cpu	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 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
#+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                #
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
```

(下页继续)

(续上页)

```
#       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.                     #
#####################################################
# 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

Gamess 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.
.
.
.
.
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

```
srun -p small -n 4 --pty /bin/bash #申请计算资源
module load miniconda3 #加载模块
conda create -n deepmd deepmd-kit=*=*gpu libdeepmd=*=*gpu lammps-dp␣
↪cudatoolkit=11.3 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	cpu	容器	/dssg/share/imgs/chroma/chroma2021.04.sif 思源一号
2021.4	cpu	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

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. π2.0 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

export QUDA_RESOURCE_PATH=$PWD
export GPU_COUNT=2
```

(下页继续)

(续上页)

```
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
```

(下页继续)

(续上页)

```
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} \
```

(下页继续)

(续上页)

```
 -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}

cmake --build ${BUILDDIR}/build_qdp-jit ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_qdp-jit
```

8. 编译 QUDA

```
cmake -S ${SRCDIR}/quda -B ${BUILDDIR}/build_quda \
 -DCMAKE_BUILD_TYPE=RELEASE \
 -DCMAKE_INSTALL_PREFIX=${INSTALLDIR} \
 -DCMAKE_PREFIX_PATH=${INSTALLDIR} \
 -DQUDA_GPU_ARCH=${SM} \
 -DQUDA_NVSHMEM=${QUDA_NVSHMEM} \
 -DQUDA_DIRAC_DEFAULT_OFF=ON \
 -DQUDA_DIRAC_CLOVER=ON \
 -DQUDA_DIRAC_WILSON=ON \
 -DQUDA_INTERFACE_QDPJIT=ON \
 -DQUDA_QDPJIT=ON \
 -DQUDA_INTERFACE_MILC=OFF \
 -DQUDA_INTERFACE_CPS=OFF \
 -DQUDA_INTERFACE_QDP=ON \
 -DQUDA_INTERFACE_TIFR=OFF \
 -DQUDA_QMP=ON \
 -DQUDA_QIO=OFF \
 -DQUDA_MULTIGRID=ON \
 -DQUDA_MAX_MULTI_BLAS_N=9 \
 -DQUDA_BUILD_SHAREDLIB=ON \
 -DQUDA_BUILD_ALL_TESTS=OFF \
 -DCMAKE_CXX_FLAGS=${ARCHFLAGS}

cmake --build ${BUILDDIR}/build_quda ${CMAKE_MAKE_OPTS}
cmake --install ${BUILDDIR}/build_quda
```

9. 编译 Chroma

```
cmake -S ${SRCDIR}/chroma -B ${BUILDDIR}/build_chroma \
 -DCMAKE_BUILD_TYPE=RELEASE \
 -DCMAKE_INSTALL_PREFIX=${INSTALLDIR}/ \
 -DCMAKE_PREFIX_PATH=${INSTALLDIR}/ \
 -DBUILD_SHARED_LIBS=ON \
 -DChroma_ENABLE_JIT_CLOVER=ON \
 -DChroma_ENABLE_QUDA=ON \
 -DChroma_ENABLE_OPENMP=ON \
```

(下页继续)

(续上页)

```
 -DCMAKE_CXX_STANDARD=20 \
 -DCMAKE_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_
↪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"
```

(下页继续)

(续上页)

```
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
```

(下页继续)

(续上页)

```
├── 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
```

4.9.6 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：module 标准版、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：

- module 标准版，加载即用，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)
PIAQIHILEGRSDEQKETLIREVSEAISRSLDAPLTSVRVIITEMAKGHFGIGGELASK
```

（复合体 fasta 文件示例）

```
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
>2MX4
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
```

版本一：**module** 标准版

module 标准版加载后即可使用，支持复合体计算

module 在思源一号上运行

作业脚本示例（假设作业脚本名为 `sub.slurm`）：

```
#!/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 \
```

(下页继续)

(续上页)

```
--model_preset=monomer \
--output_dir=output
```

思源一号上调用 AlphaFold2.2.0 的 multimer 模块的脚本

```
#!/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
#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
```

然后使用 `sbatch sub.slurm` 语句提交作业。

module 在 π 集群上运行

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。

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 在思源一号上运行

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=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

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
```

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。ColaFold 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
```

(下页继续)

(续上页)

```
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 synthenic 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，则需要进行训练

准备训练集与测试集

根据 Augutus 的官方教程，应当按如下标准进行基因结构序列准备：

a. 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；
b. 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；
c. 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
etraining --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
```

(下页继续)

(续上页)

```
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

#␣
 ↪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LOCU
 ↪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
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
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Trainning␣
↪set不够大，请添加数据；
# 如果你获得了满意的Trainning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（Trainning 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

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 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=OEALEFT,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
TCAGGTATACACCTTCCGACTAGACGTGTTCGAAGATTCGGGAAAATTACCTGAAGAGCCCCCGTAAGCC
GTAGTAGAAGAGGACACTTCATTTAAACAATACCGAAAAAGTGTCTTGGCAGACCGTATCTTCACAGGGC
CGAAGCACTTTTGGCAGGCTTATAAACGCCCAGAATGAAGCACTCGCCATAGGTGGAAACCTTTAAGCGA
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

CAGGGCATGGTCAGGTATACACCTTCCGACTAGACGTGTTCGAAGATTCGGGAAAATTAC
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
CAGGGCATGGTCAGGTATACACCTTCCGACTAGACGTGTTCGAAGATTCGGGAAAATTAC

CTGAAGAGCCCCCGTAAGCCGTAGTAGAAGAGGACACTTCATTTAAACAATACCGAAAAA
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
CTGAAGAGCCCCCGTAAGCCGTAGTAGAAGAGGACACTTCATTTAAACAATACCGAAAAA

GTGTCTTGGCAGACCGTATCTTCACAGGGCCGAAGCACTTTTGGCAGGCTTATAAACGCC
```

(下页继续)

(续上页)

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
GTGTCTTGGCAGACCGTATCTTCACAGGGCCGAAGCACTTTTGGCAGGCTTATAAACGCC

CAGAATGAAGCACTCGCCATAGGTGGAAACCTTTAAGCGACGCGGGGTGT...TCAGGTT
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
CAGAATGAAGCACTCGCCATAGGTGGAAACCTTTAAGCGACGCG-----2000------T

TGGGTCCGCGCAGCGCCAACGATTTCAACCGGGAGACGTTCGTTCATGATGAGAAGACGG
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TGGGTCCGCGCAGCGCCAACGATTTCAACCGGGAGACGTTCGTTCATGATGAGAAGACGG
......
```

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.
PATHOGEN_BWA is the BWA index of the virus database.
BLAST_PATH is the path to the blast binary
```

(下页继续)

(续上页)

```
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
```

(下页继续)

(续上页)

```
#-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 predicion of␣
↪the integration. A higher number indicreases 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
```

(下页继续)

(续上页)

```
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 predicion 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

#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

```
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 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‘不可以同事存在

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 mypy
source activate mypy
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	cpu	Spack	*bismark/0.23.0-gcc-11.2.0* 思源一号
0.19.0	cpu	Spack	*bismark/0.19.0-intel-19.0.4*

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      ␣
↪AAAAAAAAAAAAAAGAAAAAAAAGAAAAAGAAAAGGAAAAGTAAAAAAAA      ␣
↪=@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      ␣
↪ATTTTGTTATAGAGTGGGGTATTTTCGGGAAGAAGGAGGAGGAGTGTATT      ␣
↪BCCCCBCCCCA?:=ACCBCABCCCCCBCCA??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
```

(下页继续)

(续上页)

```
SRR020138.15024320_SALK_2029:7:100:1672:1164_length=86        16     ␣
↪ chr5     28344472         8         50M     *         0         0       ␣
↪CACAAAATATCAACACCCCTAAACCCCACATTATTCAAAAATCAATTATA      ␣
↪@@@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       ␣
↪TTTTGAGTAGAGAAGTTAGTATTTTAGGGAATTTTTGATTTTTTTTAAGTT       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。

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
```

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。领结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易位等五种结构突变。

完整步骤

```
module load miniconda3
conda create -n mypy
source activate mypy
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	cpu	spack	bwa/0.7.17-intel-2021.4.0 思源一号
0.7.17	cpu	spack	bwa/0.7.17-intel-19.0.4

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

π2.0:
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*
- *π2.0 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
```

(下页继续)

(续上页)

```
#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 bwtsw 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 bwtsw hg19.fa
[main] Real time: 2784.274 sec; CPU: 2775.397 sec

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

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

```
srun -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n cdhit
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:

```
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 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:

```
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 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 mypy
source activate mypy
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 mypy_py27 python=2.7
source activate mypy_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:

```
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 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/colabfold_batch/bin:$PATH"
module load miniconda3
source activate ~/colab/localcolabfold/colabfold_batch/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)
PIAQIHILEGRSDEQKETLIREVSEAISRSLDAPLTSVRVIITEMAKGHFGIGGELASK
```

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 mypy_py27 python=2.7
source activate mypy_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 trascripts.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, bellerophontes, deFuse, FusionFinder,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
```

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 mypy
source activate mypy
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 singluarity 镜像的 slurm.sh 如下：

```
#!/bin/bash

#SBATCH -J DeepVariant
#SBATCH -p small
#SBATCH -n 1
#SBATCH --ntasks-per-node=1
#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 singluarity 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
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 deepvarant 作业

```
srun --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
#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 mypy_py27 python=2.7
source activate mypy_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 mypy
source activate mypy
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:

```
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 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	cpu	Spack	*fastqc/0.11.9-gcc-11.2.0-openjdk* 思源一号
0.11.7	cpu	Spack	*fastqc/0.11.7-gcc-9.2.0*
0.11.7	arm	Spack	fastqc/0.11.7-intel-19.0.4
0.11.7	cpu	Spack	fastqc/0.11.7-gcc-8.3.0

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
# π 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
```

(下页继续)

(续上页)

```
#SBATCH -N 1
#SBATCH --ntasks-per-node=10

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

#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

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

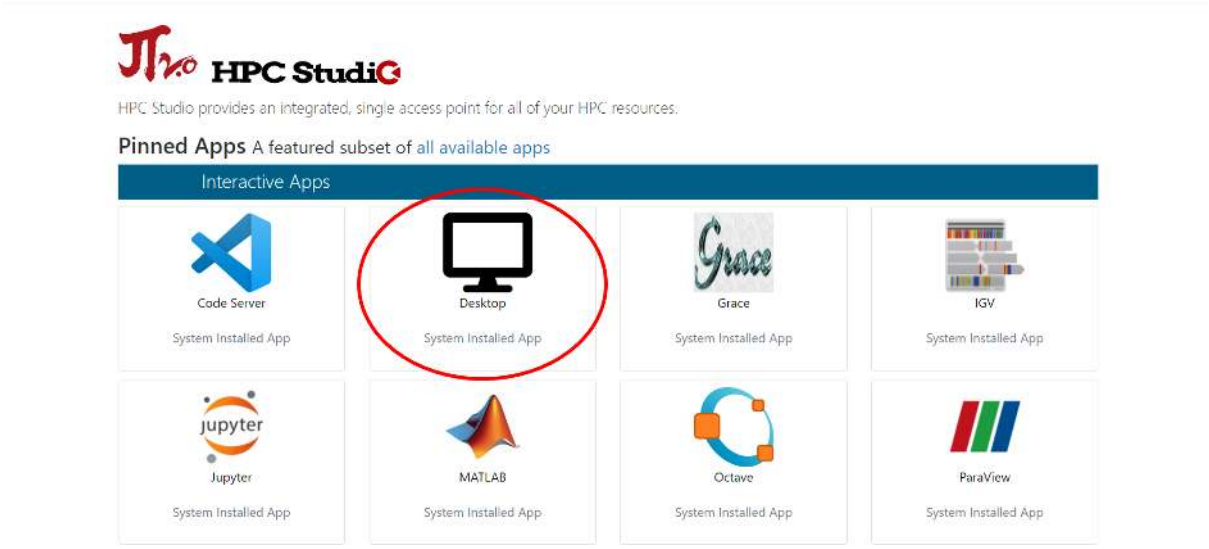
简介

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

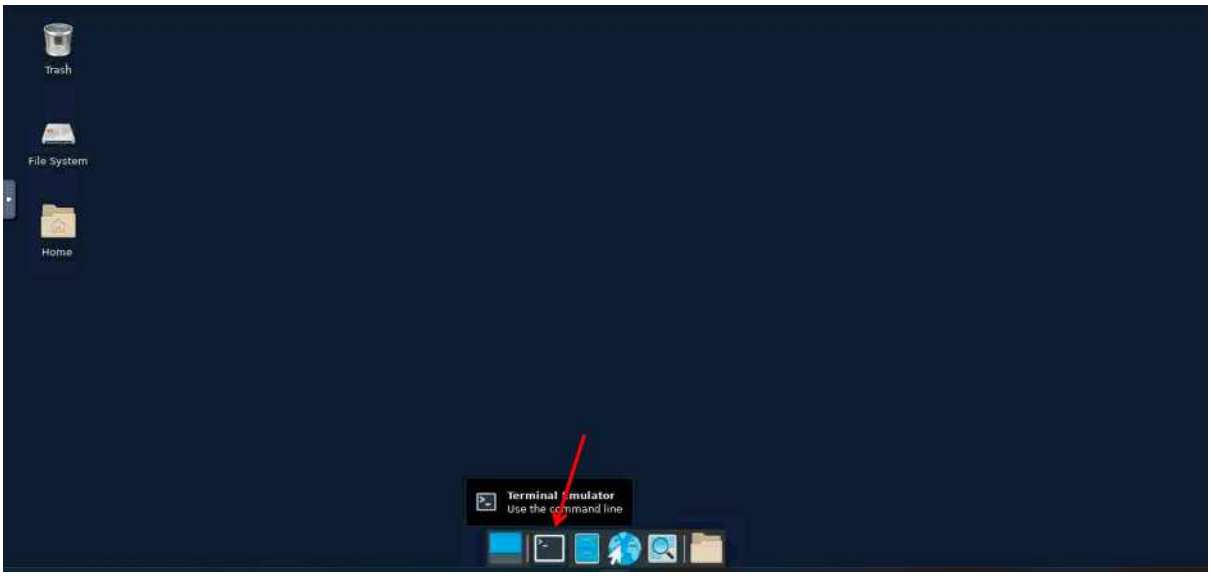
思源一号预编译 Geant4

思源一号平台上已预编译好基于 qt 可视化界面的 geant4 模块，用户可通过加在相干模块制作 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

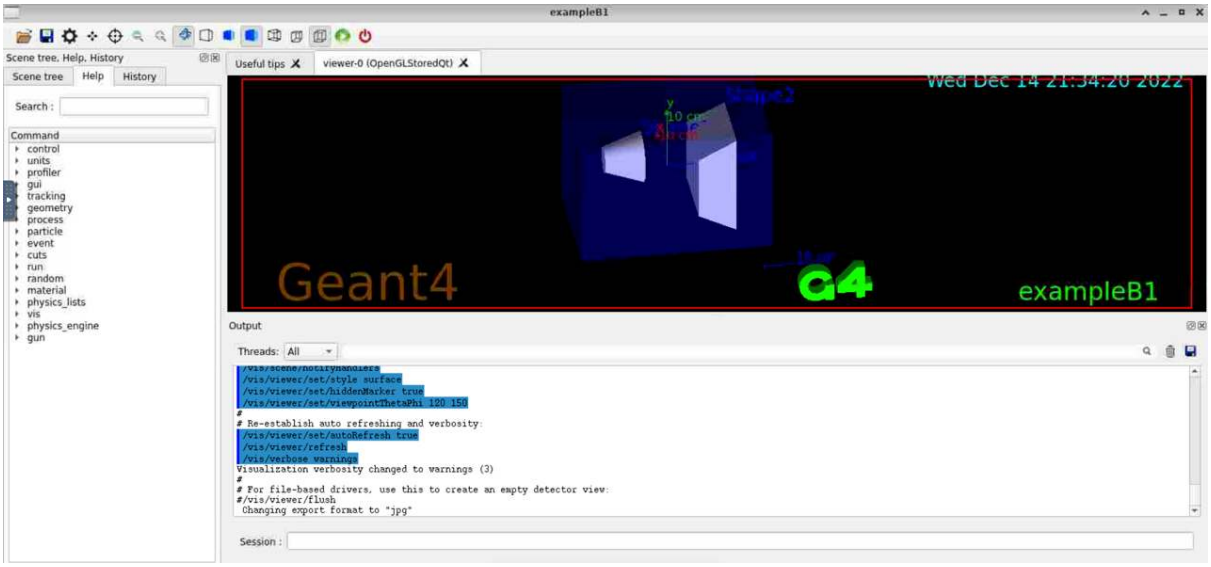
```
module load gcc/11.2.0 qt/5.15.5-gcc-11.2.0 geant4/11.0.3-gcc-11.2.0
```

- 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/geant4-11.0.3-4lmd7rsrbnlougezulj7hyhrch67ld4r，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
```

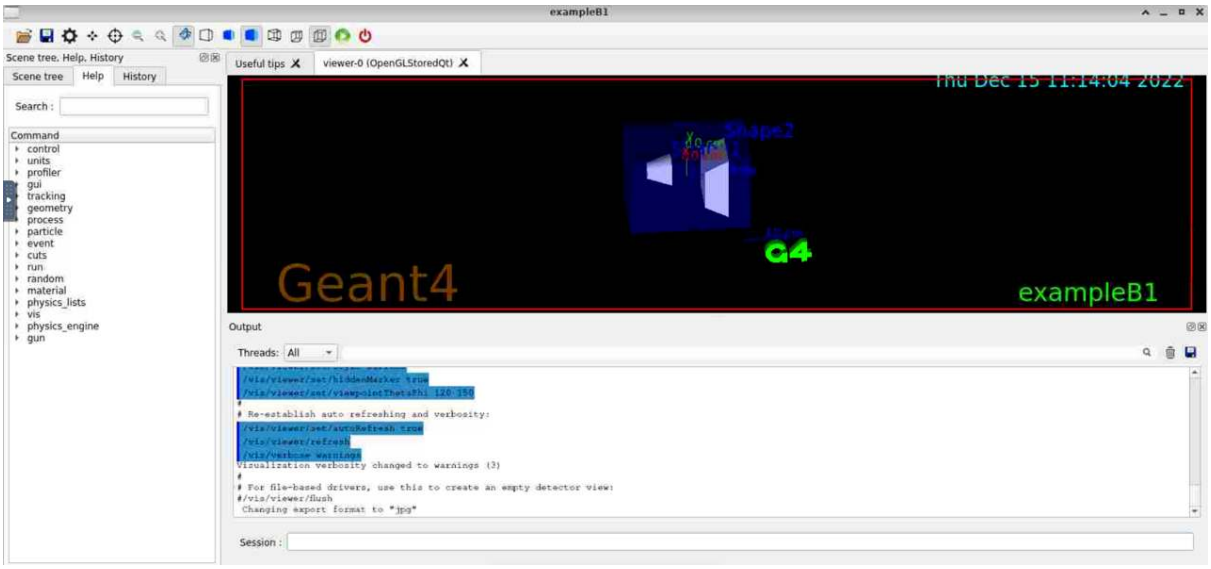
- 激活环境变量

```
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 mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda genomestrip
```

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 contentout 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 mypy_py27 python=2.7
source activate mypy_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:

```
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 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。

```
srun -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

```
srun -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 \
 -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 mypy_py27 python=2.7
source activate mypy_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 seusequent runs. MAKER's inputs are minimal and its ouputs 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

```
srun -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n mypy
source activate mypy
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
```

(下页继续)

(续上页)

```
#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地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 mypy_py27 python=2.7
source activate mypy_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 mypy_py27 python=2.7
source activate mypy_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:

```
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 mfuzz
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
```

(下页继续)

(续上页)

```
#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

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

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。

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 mypy
source activate mypy
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 mypy
source activate mypy
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, duplic ations, and translocations. Both single-sample calling and joint (multi-sample) calling are provided.

完整步骤

```
module load miniconda3
conda create -n mypy_py27 python=2.7
source activate mypy_py27
conda install -c bioconda pbsv
```

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 mypy_py27 python=2.7
source activate mypy_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 mypy_py27 python=2.7
source activate mypy_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 mypy
source activate mypy
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**

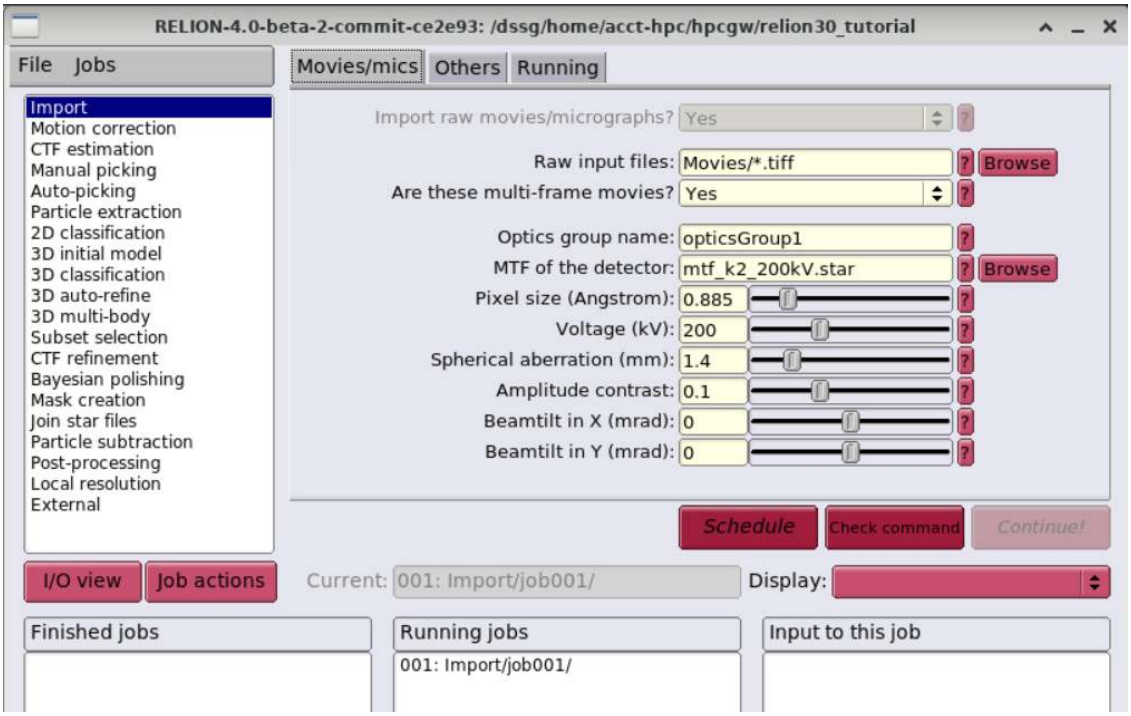
进入远程桌面，打开终端 *Terminal*。

```
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

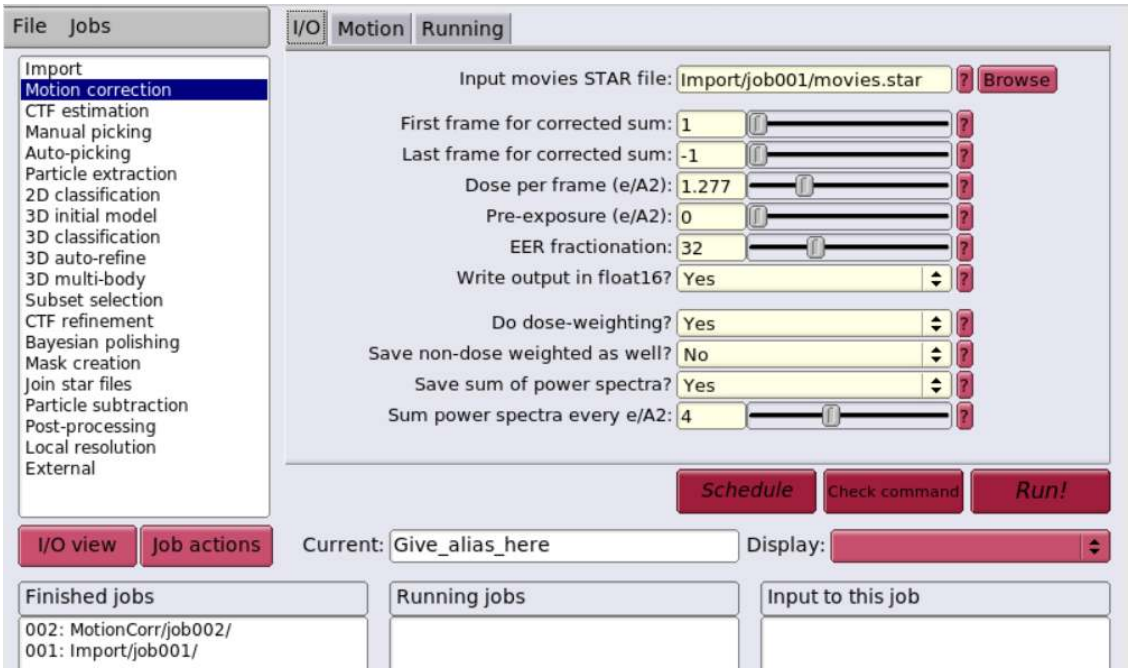
3. 点击 Run! 运行。

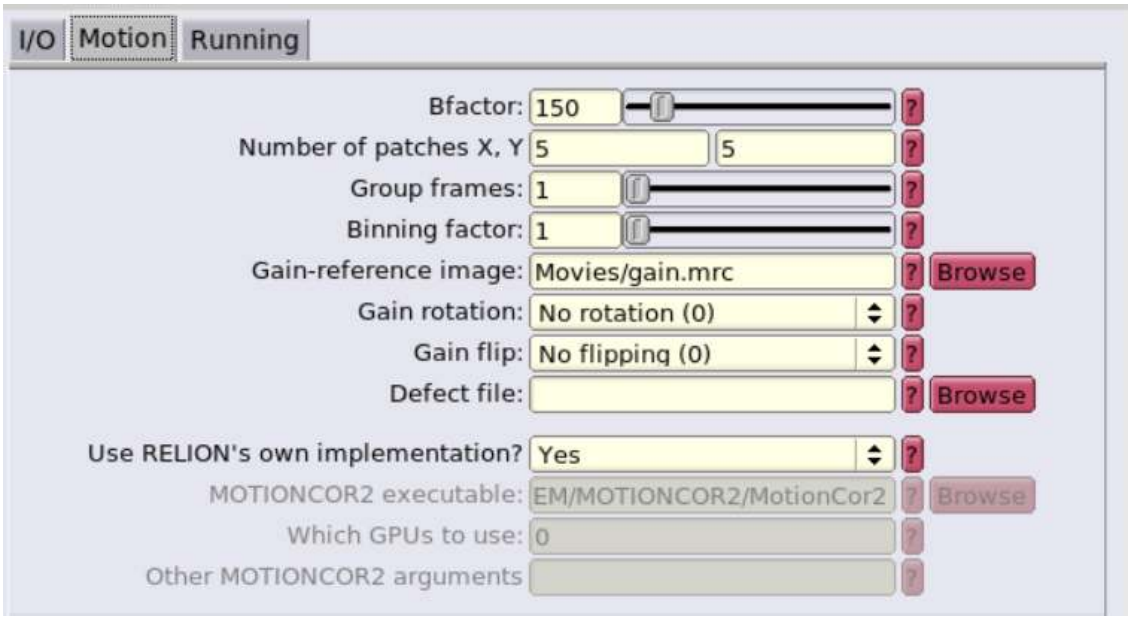
   此时会创建 Import/job001/ 目录，及 STAR 文件，可以在终端通过 less Import/job001/movies.star 查看。

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`

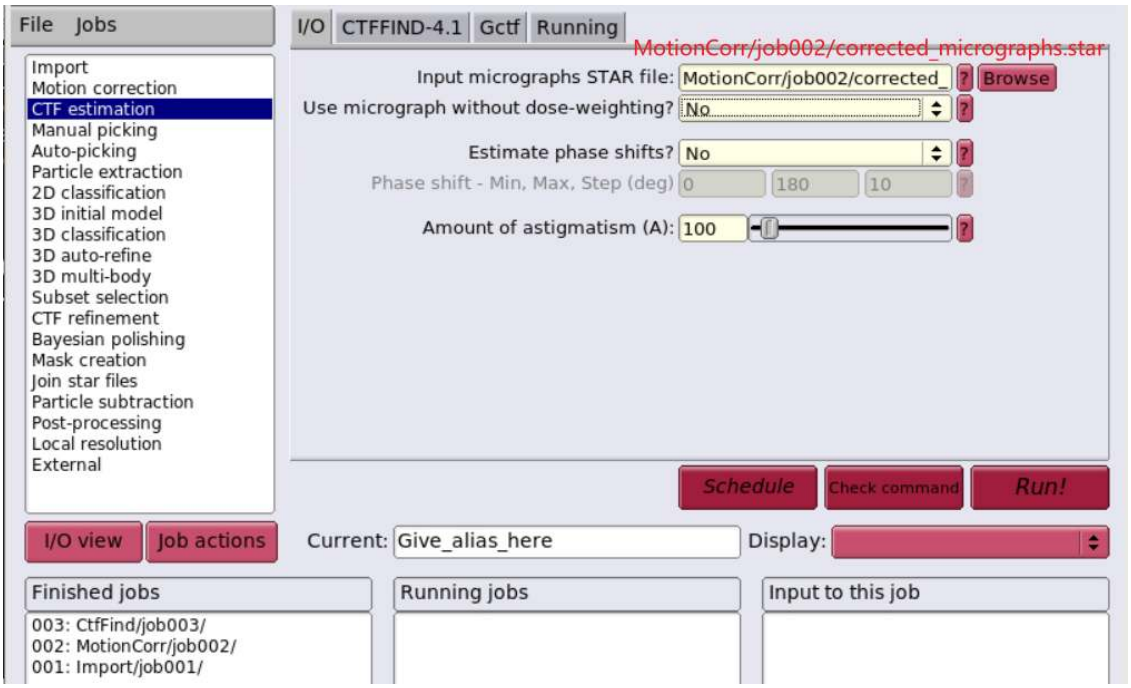
   Number of threads: | `12`

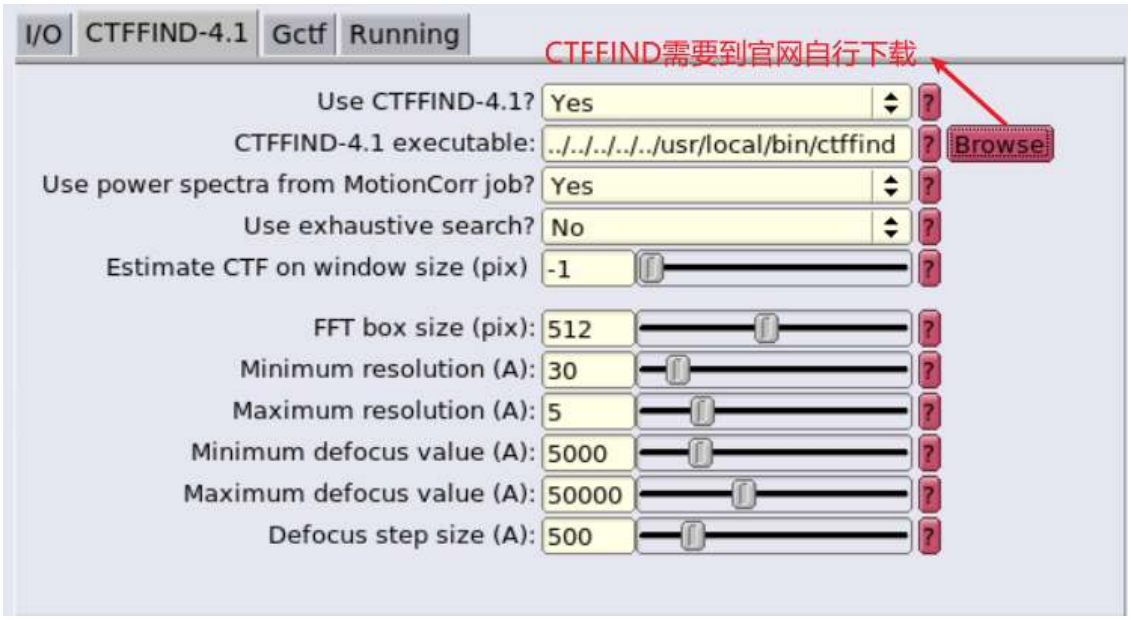
   Submit to queue? | `No`

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CFT，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：





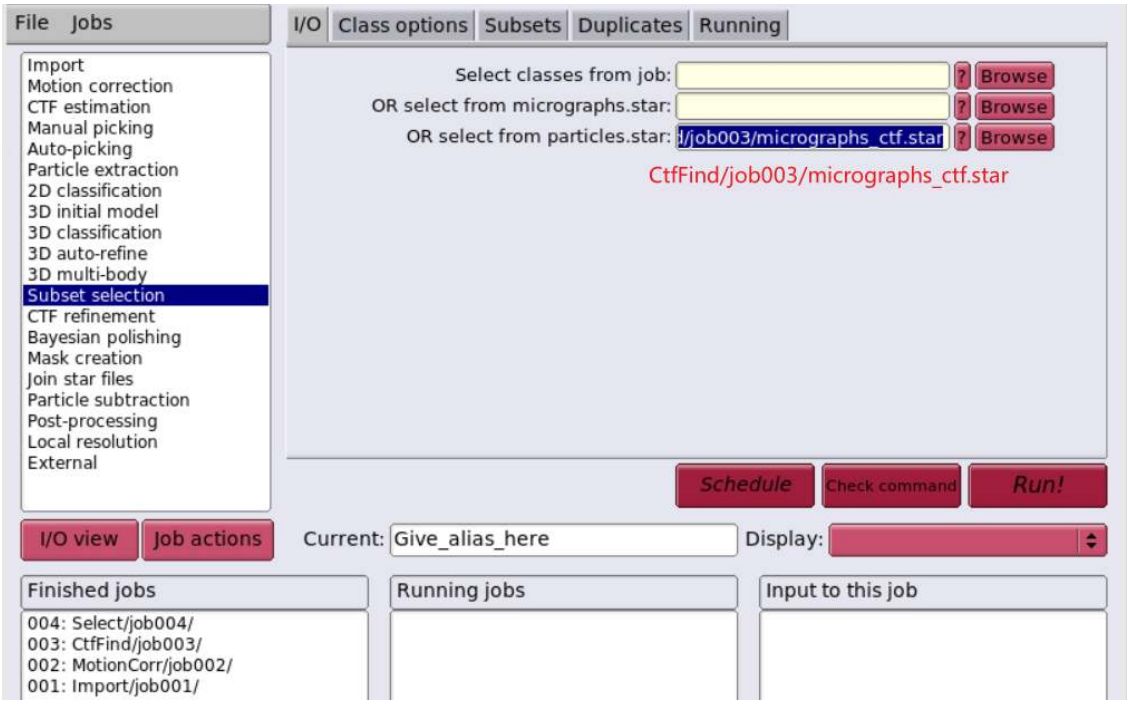
2. 设置 `Running`，点击 `Run!` 运行。

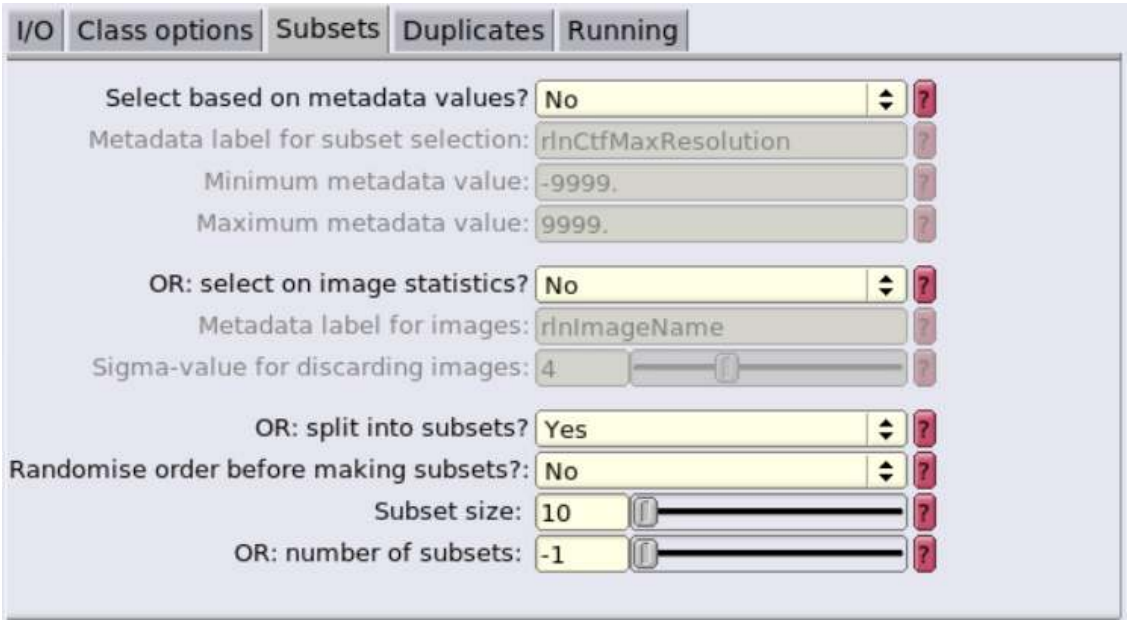
   Number of MPI procs: `4`

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：

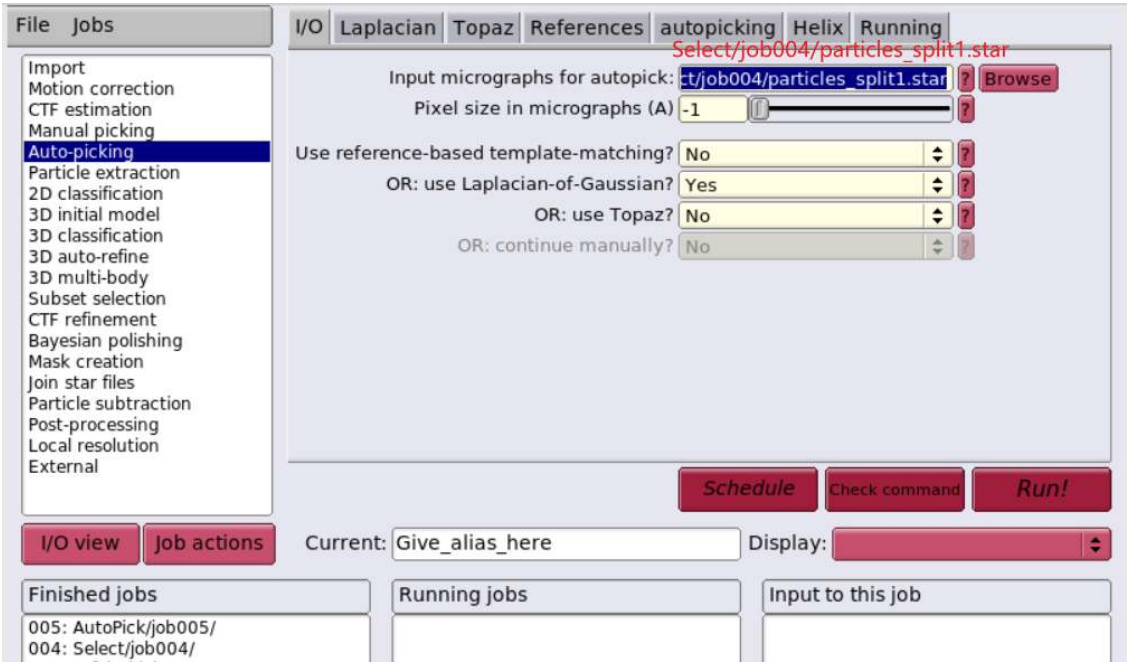


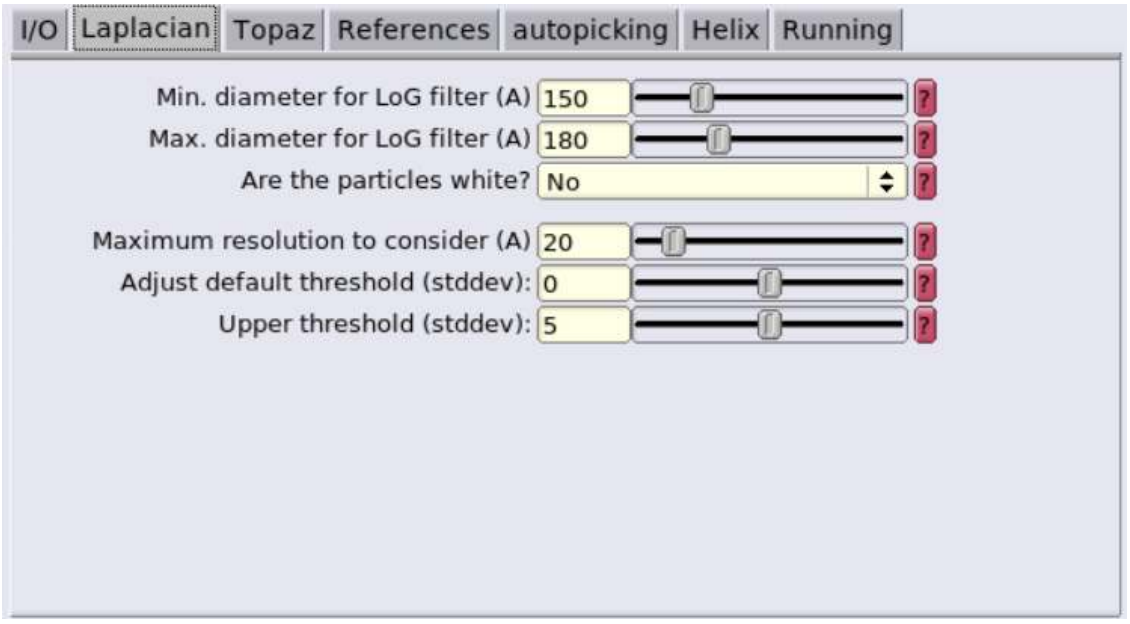


2. 点击 `Run!` 运行。

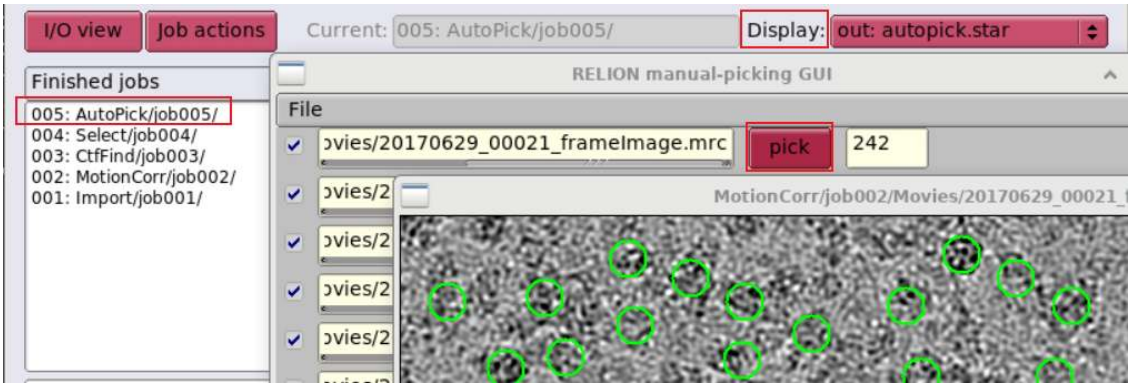
基于拉普拉斯-高斯算子的颗粒自动挑选 (*LoG-based auto-picking*)

1. 用 `Auto-picking` 自动挑选颗粒，设置如下：



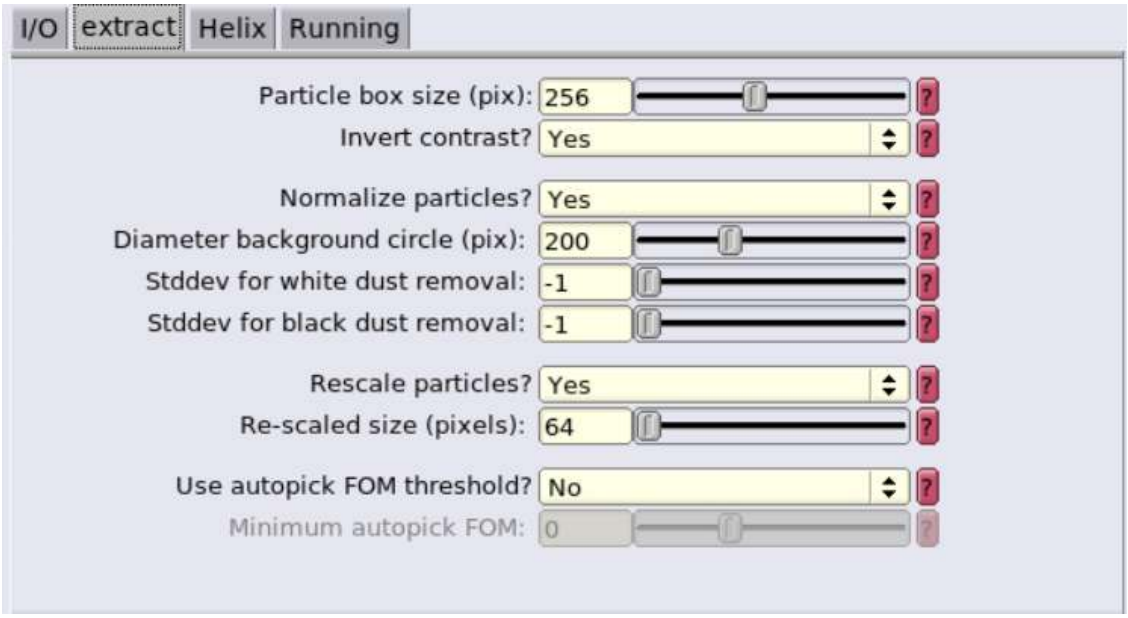


2. 点击 `Run!` 运行；
3. 选中运行完的 job，`Display` 设置为 `autopick.star` ，查看颗粒挑选结果。

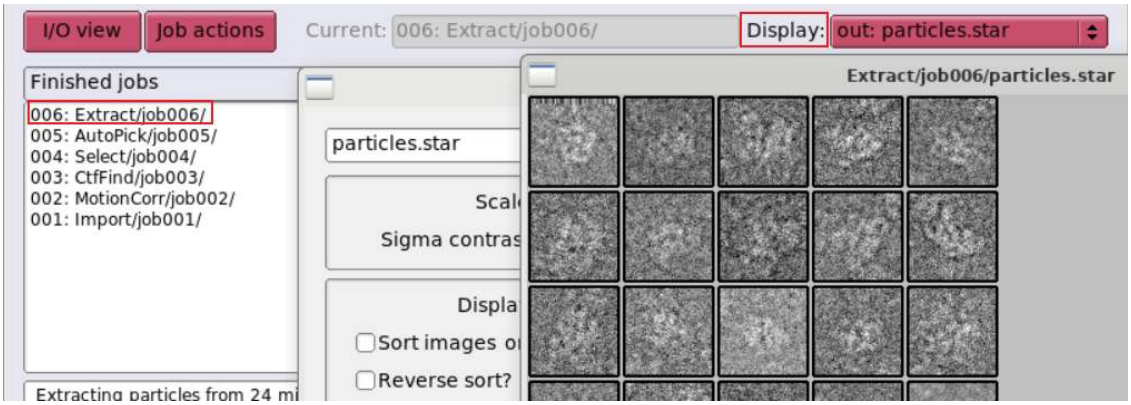


颗粒提取 (*particle extraction*)

1. 用 `Particle extraction` 提取颗粒，设置如下：

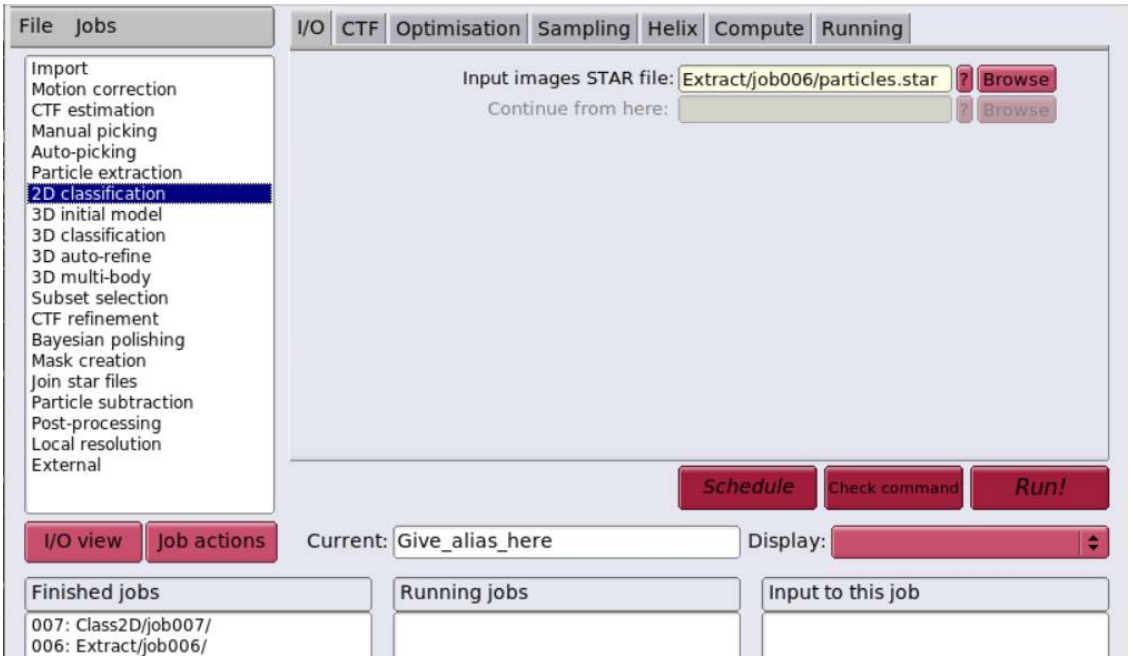


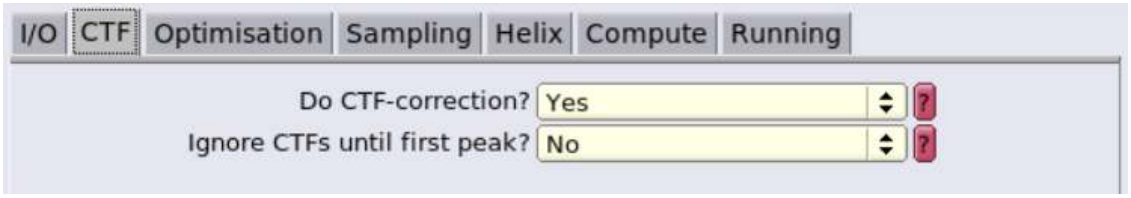
2. 点击 `Run!` 运行；
3. 选中运行完的 job，`Display` 设置为 `particles.star`，查看颗粒提取结果。

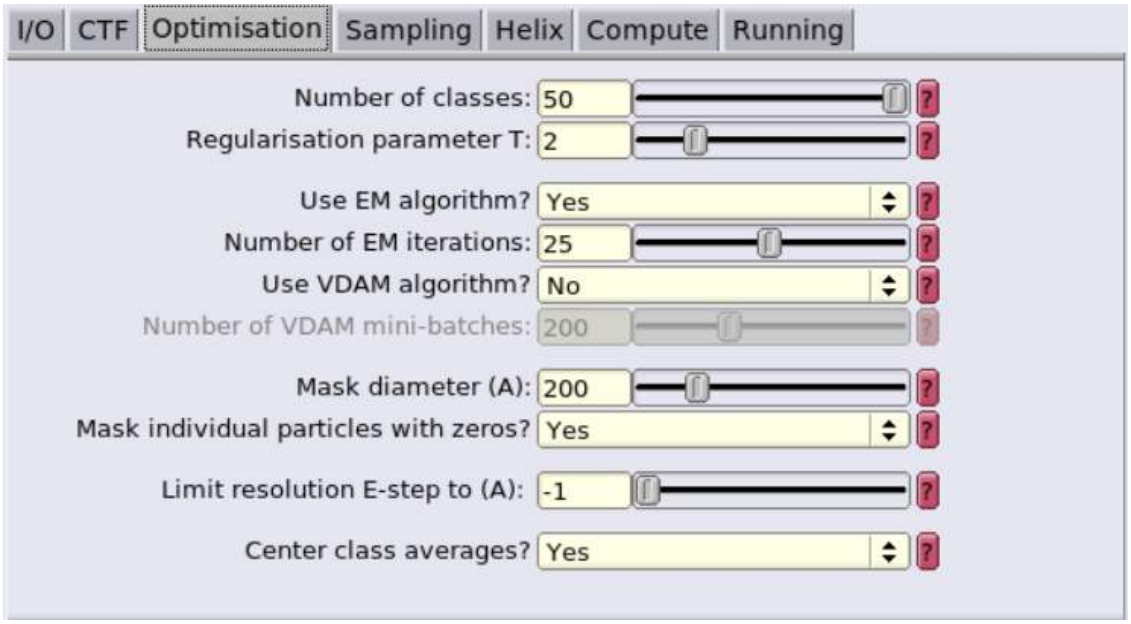


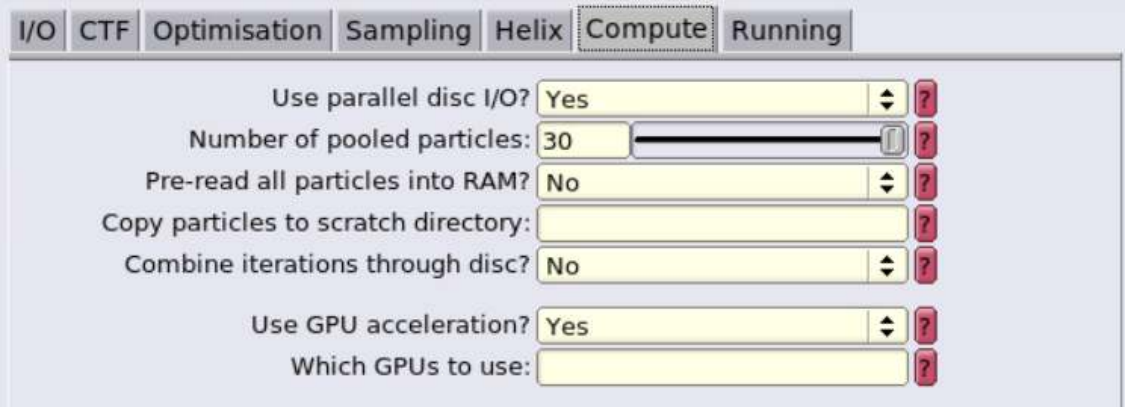
制作自动挑选的模板 (2D class averaging)

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：


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 CTF Optimisation Sampling Helix Compute Running
Input images STAR file: Extract/job006/particles.star Browse
Continue from here: Browse
Schedule Check command Run!
I/O view Job actions
Current: Give_alias_here
Display:
Finished jobs
007: Class2D/job007/
006: Extract/job006/
Running jobs
Input to this job


I/O CTF Optimisation Sampling Helix Compute Running
Do CTF-correction? Yes
Ignore CTFs until first peak? No


I/O CTF Optimisation Sampling Helix Compute Running
Number of classes: 50
Regularisation parameter T: 2
Use EM algorithm? Yes
Number of EM iterations: 25
Use VDAM algorithm? No
Number of VDAM mini-batches: 200
Mask diameter (A): 200
Mask individual particles with zeros? Yes
Limit resolution E-step to (A): -1
Center class averages? Yes



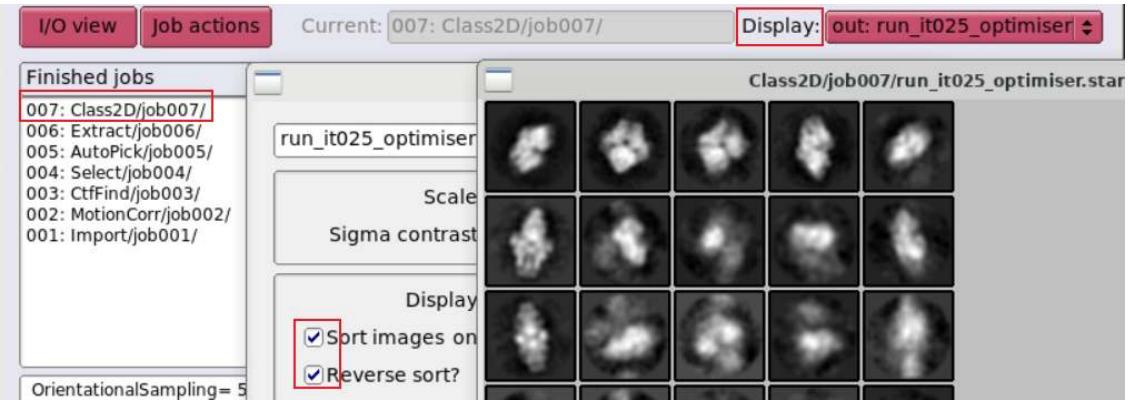
2. 设置 `Running`，点击 `Run!` 运行；

   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 π 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
```

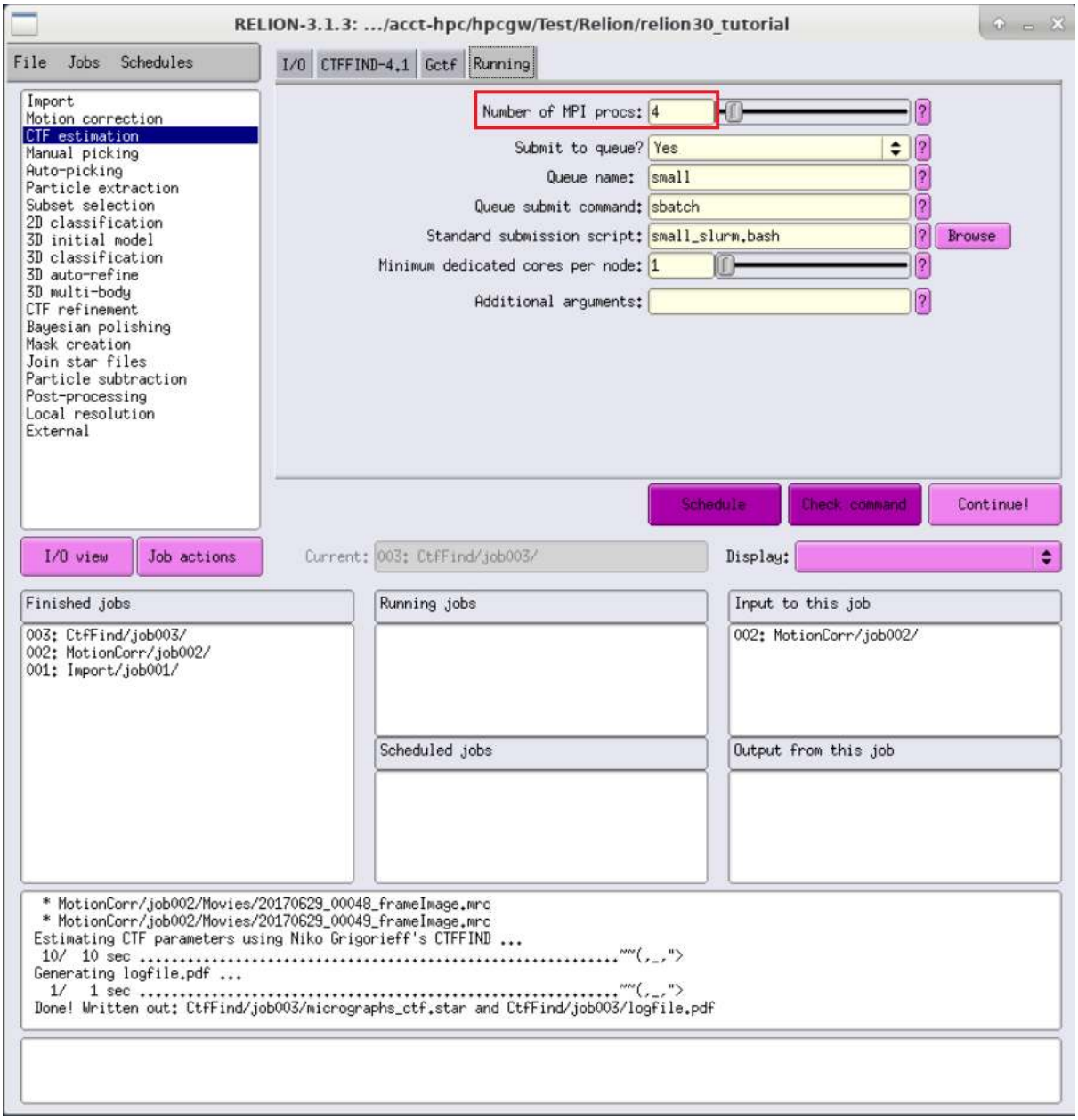
其它：作业模板设置

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
#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
```

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 RoseTTAFold 的 module，最新更新日期：2021 年 7 月 31 日

```
rosettafold/1-python-3.8
```

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
PTRTVAISDAAQLPHDYCTTPGGTLFSTTPGGTRIIYDRKFLLDR
```

- 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。

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

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
```

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。

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	cpu	Spack	*samtools/1.13-gcc-11.2.0* 思源一号
1.9	cpu	Spack	*samtools/1.9-gcc-9.2.0*
1.10	arm	Spack	*samtools/1.10-gcc-9.3.0*
1.9	cpu	Spack	samtools/1.9-intel-19.0.4
1.9	cpu	Spack	samtools/1.9-gcc-8.3.0
1.9	cpu	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：

```
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 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:

```
srun -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：

```
srun -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:

```
srun -p small -n 4 --pty /bin/bash
module load samtools/1.9-gcc-9.2.0
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，对比对结果的 `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 mypy
source activate mypy
conda install -c bioconda sniffles
```

SOAPDENOVO2

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

```
srun -p small -n 4 --pty /bin/bash
module load miniconda3
conda create -n mypy
source activate mypy
conda install -c bioconda sra-tools
```

源码安装步骤

```
srun -p small -n 4 --pty /bin/bash
mkdir -p ${HOME}/01.application/13.sra-tools && cd ${HOME}/01.
↪application/13.sra-tools
wget --output-document sratoolkit.tar.gz https://ftp-trace.ncbi.nlm.
↪nih.gov/sra/sdk/current/sratoolkit.current-centos_linux64.tar.gz
tar -vxzf sratoolkit.tar.gz
export PATH=${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.fstaq.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 是一个新的转录组标表达定量软件。

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
#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 mypy_py27 python=2.7
source activate mypy_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

#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 correla tions between variables.

完整步骤

```
module load miniconda3
conda create -n mypy
source activate mypy
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	cpu	容器	rosetta/3.12 思源一号
3.12	cpu	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
cp -r /lustre/share/samples/rosetta/input_files ./
mkdir output_files
```

集群上的 **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）
```

(下页继续)

(续上页)

```
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
#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
```

(下页继续)

(续上页)

```
#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
#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
#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
```

(下页继续)

(续上页)

```
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
```

(下页继续)

(续上页)

```
#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 _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_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
```

π2.0 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	cpu	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
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	cpu	spack	star/2.7.6a-gcc-11.2.0 思源一号
2.7.6a3	cpu	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**

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
```

(下页继续)

(续上页)

```
#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
```

(下页继续)

(续上页)

```
|    ├── 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
├── 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
```

(下页继续)

(续上页)

```
|    ├── 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	cpu	容器	/dssg/share/imgs/easyfuse/easyfuse_1.3.6.sif 思源一号
1.3.5	cpu	容器	/dssg/share/imgs/easyfuse/easyfuse_1.3.5.sif 思源一号
1.3.6	cpu	容器	/lustre/share/img/x86/easyfuse/easyfuse_1.3.6.sif
1.3.5	cpu	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
#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 \
```

(下页继续)

(续上页)

```
/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
    ├── 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

```
srun -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	cpu	容器	gate/9.2-gcc-8.5.0-singularity 思源一号
9.2	cpu	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*
- *π2.0 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

#!/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. π2.0 GATE

π2.0 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"
```

π2.0 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
```

π2.0 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/ pangenie_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*
- *π2.0 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
```

π2.0 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
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

4.10 常见问题

4.10.1 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。

4.10.2 1. 帐号申请和登录

1.1 Q：如何开通帐号？

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帐号。

1.2 Q：是不是只有教职工才能申请帐号？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）才能申请主帐号。主帐号下面可以免费开通子帐号。

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，主帐号和子帐号均可在此文件夹下读写。

1.3 Q：子帐号的申请和费用？

A：子帐号也是在“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帐号。

可自定义子帐号名，格式为 xxx-yyy，其中 xxx 为主帐号，yyy 可以自定义，比如 user1、user2。我们将在两个工作日内为您开通。每个主帐号能免费申请子帐号的个数不限，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，请使用或抄送帐号负责人邮箱发送邮件到 hpc 邮箱，邮件中注明帐号，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，您的帐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。

4.10.3 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。192ct6 节点有 6T 内存。具体使用请参考作业示例

4.10.4 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，若被检测到您的帐号不当使用登录节点，您的帐号将会被封禁 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
```

(下页继续)

(续上页)

```
 # 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

# π2.0节 点 代 理 设 置
proxy.hpc.sjtu.edu.cn:3004
```

4.10.5 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：集群上是否提供商业软件？

A：目前暂不提供商业软件，不过您可以自行购买后安装。以下是注意事项：

商业软件 License 通常需要使用专用的 License 服务器，在购买商业软件并尝试在集群上部署 License 服务器前，请与我们以及软件厂商进行充分沟通。

a) 不要把 License 绑定到集群的登录节点；

b) 请购买浮动授权，即计算程序可以在集群上的任意一个节点启动，通常需要安装特定的 License 服务器；

c) 询问 License 服务器是否可以部署在虚拟机上，这样我们可以专门开一台虚拟机运行您的 License 服务器；

d) 与厂商充分沟通 License 服务器安装模式、授权数量、使用限制、更换 MAC 地址的费用以及厂商具备基本的技术支持能力。如果需要了解集群的软硬件信息，可以在交流过程中抄送 hpc 邮箱。

4.3 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。

4.10.6 5. 收费和充值

5.1 Q：如何收费？

A：请发送邮件至 hpc 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王老师，电话 34206060-8011，邮箱 stwangecho[AT]sjtu.edu.cn

5.3 Q：如何查看账户余额？

A：您可以使用主帐号或子帐号登录 计费系统 查看。也可以使用帐号负责人 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王老师，电话 34206060-8011，邮箱 stwangecho[AT]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。在此之前已有的奖励机时仍然生效。

5.6 Q: 电子信息与电气工程学院优惠政策怎么申请？（仅适用于电院老师）

A：电院优惠政策请参考 http://dzb.seiee.sjtu.edu.cn/dzb/info/15820.htm

4.10.7 6. 邮件支持

6.1 Q 向 hpc 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. 新建帐号：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。

4.10.8 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。

4.10.9 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.

4.10.10 9. 医学院和附属医院申请 jAccount 帐号

A：请至 上海交通大学医学院网络信息中心 页面了解和办理。

4.10.11 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
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。

4.11 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