高性能计算平台

最佳实践

腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

最佳实践
自动伸缩最佳实践
第一性原理计算框架 CONQUEST 的安装与测试

最佳实践
自动伸缩最佳实践

最近更新时间：2023-03-29 12:58:33

本文以使用 VASP 软件进行高性能计算介绍如何配置自动伸缩策略。

背景信息

当您需要每天不定时提交作业，使用 THPC 集群几个小时进行大规模计算， 然后释放节点，您可以针对不同的作业类型，配置不同的伸缩策略。配置伸缩策略后，系统可以根据实时负载自动增加或减少计算节点。可以帮您合理利用资源，减少使用成本。

前置条件

请依照 准备工作 中的指引获取 SecretId 和 SecretKey，并完成 服务角色授权 。

操作步骤

1. 安装最新版本的 TCCLI 命令工具 ，并进行 初始化配置 请参见 THPC 地域列表 。
2. 使用 CreateCluster 接口创建一个有 1 个 管控节点 0 个 计算节点 的 THPC 集群。

> ⚠ 注意
> - 需要把 AutoScalingType 设置为 THPC_AS，调用后续步骤的 SetAutoScalingConfiguration 接口才会生效。
> - Placement 为 TCCLI Region 地域对应可用区。
> - VirtualPrivateCloud 设置需要是集群同可用区的 VPC ID，子网 ID。
> - 管控节点登录密码设置请参见 LoginSettings 。
> - 支持的基础镜像版本请参见 镜像支持 ，或者可以基于基础镜像版本 创建自定义镜像 ，传入自定义镜像 ID。
> - 集群创建时如果您不指定挂载目录 ，将自动为您在 /opt 目录下创建并挂载 CFS，所有节点共享这一目录，您可以在这个目录下安装您业务需要的软件，本文默认您已在 /opt 目录下安装编译完成依赖的 Intel oneAPI 和 VASP 软件。

执行命令：

```
tccli thpc CreateCluster --cli-input-json file://./test.json
```

test.json 文件参数配置可参见：

```
{
    "Placement": {
        "Zone": "ap-chongqing-1"
    },
    "ManagerNodeCount": 1,
    "ManagerNode": {
        "InternetAccessible": {
            "InternetMaxBandwidthOut": 10
        },
        "InstanceName": "ThpcTestSlurmManagerNode",
        "InstanceType": "S5.MEDIUM4"
    },
    "SchedulerType": "SLURM",
    "LoginSettings": {
        "Password": "xxxxxxxx"
    },
    "ImageId": "img-l8og963d",
    "VirtualPrivateCloud": {
        "VpcId": "vpc-r9jw2jzv",
        "SubnetId": "subnet-0v4eybey"
    },
    "AutoScalingType": "THPC_AS"
}
```

提交请求并执行完毕后，您需要记录 ClusterId：

```
[redacted] % tccli thpc CreateCluster --cli-input-json file://./test.json
{
    "ClusterId": "hpc-5b49i4he",
    "RequestId": "9c9a8319-c949-4eff-8595-481173cd8b12"
}
```

使用 ClusterId 查询集群状态：

```
tccli thpc DescribeClusters --ClusterIds '["hpc-xxxxxxxx"]'
```

返回结果，其中：

- **ClusterStatus 为 RUNNING 状态表示集群创建完成，可以开始使用。**
- **ClusterStatus 为 INITING 状态表示集群正在初始化中，需要等待直到为 RUNNING或者 INIT_FAILED 状态之一。**
- **ClusterStatus 为 INIT_FAILED 状态表示集群创建失败，您可以使用 API Explorer 查看集群活动 查看详细集群状态信息。**

```
[redacted] % tccli thpc DescribeClusters --ClusterIds '["hpc-5b49i4he"]'
{
    "ClusterSet": [
        {
            "ClusterId": "hpc-5b49i4he",
            "ClusterStatus": "RUNNING",
            "ClusterName": "unnamed",
            "Placement": {
                "Zone": "ap-chongqing-1"
            },
            "CreateTime": "2022-11-01T11:30:26Z",
            "SchedulerType": "SLURM",
            "ComputeNodeCount": 0,
            "ComputeNodeSet": [],
            "ManagerNodeCount": 1,
            "ManagerNodeSet": [
                {
                    "NodeId": "ins-edzqfkz6"
                }
            ],
            "LoginNodeSet": [],
            "LoginNodeCount": 0
        }
    ],
    "TotalCount": 1,
    "RequestId": "4c8ac0b6-d96a-4d43-855d-e2d39bc945f7"
}
```

3. 使用 SetAutoScalingConfiguration 接口开启 THPC AS 功能。

```
tccli thpc SetAutoScalingConfiguration --cli-input-json
file://./SetAutoScalingConfiguration.json
```

```
{
 "ClusterId": "hpc-xxxxxxxx",
 "ExpansionBusyTime": 120,
 "ShrinkIdleTime": 120,
 "QueueConfigs": [{
    "QueueName": "compute",
    "MinSize": 1,
    "MaxSize": 5,
    "EnableAutoExpansion": true,
    "EnableAutoShrink": true,
    "ExpansionNodeConfigs": [{
       "Placement": {
          "Zone": "ap-chongqing-1"
       },
       "VirtualPrivateCloud": {
          "VpcId": "vpc-r9jw2jzv",
          "SubnetId": "subnet-0v4eybey"
       },
       "InstanceType": "S6.4XLARGE32"
    }, {
       "Placement": {
          "Zone": "ap-chongqing-1"
```

```
        },
        "VirtualPrivateCloud": {
            "VpcId": "vpc-r9jw2jzv",
            "SubnetId": "subnet-0v4eybey"
        },
        "InstanceType": "S6.2XLARGE32"
    }]
 }]
}
```

以上 SetAutoScalingConfiguration.json 文件接口配置好集群自动扩缩容参数如下：

- 开启 Slurm 调度器 compute 队列的自动扩容能力，THPC 会根据扩容策略对连续等待超过120秒的作业进行自动扩容，并且弹性节点的最大值不超过5个。ExpansionBusyTime 参数取值最小值为120s。
- 开启 Slurm 调度器 compute 队列的自动缩容能力，THPC 会对连续空闲超过120s的节点进行自动缩容，并且弹性节点的最小值不小于1个。ShrinkIdleTime 取值最小值为0s。
  compute 队列配置了两个扩容机型。

序号	机型	CPU
1	S6.4XLARGE32	16
2	S6.2XLARGE32	8

4. 登录管控节点提交 VASP 作业：
   submit.sh 脚本内容如下，脚本提交一个 vasp 作业，在一个节点上使用8个核运行：

```
#!/bin/sh
#SBATCH -p compute
#SBATCH -o job.%j.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH -D /opt/graphene

source /opt/intel/oneapi/setvars.sh
/opt/intel/oneapi/mpi/2021.7.0/bin/mpirun -n 16 /opt/vasp.6.3.0/bin/vasp_std
```

提交作业前，集群没有任何节点存在：

```
[root@manager ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      1  down* dummynode0
```

提交作业：

```
sbatch submit.sh
```

5. 集群扩容并执行作业，作业结束后缩容到0节点。
提交作业之后，THPC 会根据扩容策略，扩容出最匹配的两个8核的 S6.2XLARGE32 实例。等待约4～8分钟后，节点加入集群并运行提交的作业。

> **说明**
> 等待的时间主要包括作业连续等待时间 120 秒、创建实例的时间和节点创建完初始化并加入集群的时间。

```
[root@manager ~]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                15   compute submit.s     root  R       0:01      2 ip-172-30-16-[4,8]
[root@manager ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      1  down* dummynode0
compute*     up   infinite      2  alloc ip-172-30-16-[4,8]
```

任务执行完成之后,节点进入空闲（idle）状态：

```
[root@manager ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      1  down* dummynode0
compute*     up   infinite      2   idle ip-172-30-16-[4,8]
```

等待2～3分钟之后，会进行自动缩容到最小节点数：

```
[root@manager ~]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
compute*     up   infinite      1  down* dummynode0
compute*     up   infinite      1   idle ip-172-30-16-4
```

6. 使用完毕后，使用如下命令删除集群，关联的计算实例将会被销毁。 CFS 文件存储不会删除，如果您需要销毁，可前往 CFS 控制台 。

```
tccli thpc DeleteCluster --ClusterId hpc-xxxxxxxx
```

第一性原理计算框架 CONQUEST 的安装与测试

最近更新时间：2024-01-25 11:23:31

> **说明：**
> 本文来自 高性能计算平台 THPC 用户实践征文，仅供学习和参考。

前言

随着计算机的计算能力和运行规模的不断提升，基于第一性原理计算理论的计算材料学科越来越得到重视。但是一般来说这样的模拟对一个包含成千上万的原子、电子而言，所需的计算框架是非常复杂的，计算代价是相当昂贵的。例如为人所熟知的商用类型 第一性原理计算框架 VASP 授权通常需要五六万人民币以上，而且在一个普通超算集群上计算一个完整的体系结构（超过 1000 个原子）可能需要几周，甚至几个月。无论是软件授权成本，还是时间成本，都比较高昂。对于想学习和实践第一性原理计算的小伙伴而言，当然也有比较节省的方式。首先软件可以选用免费的开源第一性原理计算框架，例如说本文中即将介绍到的 CONQUEST，以及 ABINT，SMASH 和 QUANTUM ESPRESSO 等。

对于普通材料专业的学生来说，可能安装任意一个开源第一性原理计算框架都不是一件容易的事，毕竟有些软件所涉及到的依赖库配置确实比较麻烦。另外，大部分这种软件也只能在 Linux 平台下运行，对于 Linux 比较陌生的人使用起来也有一定的困难。东京大学物质科学团队为此将很多第一性原理计算软件安装在一个同一个云服务器中，并在网上公开允许下载该云服务器镜像。您可以在 官网 获知有关下载信息，所支持的软件列表请参见 MaterApps 。

CONQUEST

CONQUEST 是什么?

CONQUEST 是一款基于局域轨道密度泛函理论的、能以出色的缩放比例进行大规模 并行计算 的第一性原理计算软件。它使用局部轨道来表示 Kohn-Sham 本征态或者密度矩阵。 CONQUEST 可以应用于原子、分子、液体和固体，且对于大型系统特别有效。CONQUEST 可以使用哈密尔顿的精确对角化或通过线性缩放的方法来找到基态。它已被验证使用线性缩放时缩放到超过 2000000 个原子和 200000 个核，以及超过 3400 个原子和 850 个具有精确对角化的核。 CONQUEST 可以执行结构弛豫（包括单位晶胞优化）和分子动力学（在具有各种恒温器的 NVE，NVT 和 NPT 集成中）。

为什么选 CONQUEST

大规模模拟

CONQUEST 设计为使用大型对角缩放（使用精确对角化（使用多站点支持函数方法，已经证明了对 3000 多个原子的计算）或线性缩放（已经证明了对超过 2000000 个原子的计算）。此外，相同的代码和基础集可用于对 1 个原子到 1000000 个以上原子的系统进行建模。

高效并行化

CONQUEST 是一种固有的并行代码，可演示将其扩展到 800 多个内核，以实现精确的对角化，并通过线性缩放将近 200000 个内核。这种扩展使高效使用 HPC 设施成为可能。CONQUEST（在线性缩放模式下，以及在一定程度上进行精确的对角化）在弱缩放下缩放效果最佳：固定每个核心（或线程）的原子数，并根据原子数选择核心数。
CONQUEST 还以线性缩放模式提供一些 OpenMP 并行化，每个节点的 MPI 线程数量相对较少，并使用 OpenMP 进行进一步的并行化。

线性缩放

线性缩放的思想已经存在了二十多年，但是事实证明，编写高效、准确的代码来实现这些思想具有挑战性。尽管可以使用的基础集仍然受到一些限制，但 CONQUEST 已证明有效的线性缩放（具有出色的并行缩放）。对于使用 DFT 进行的 5000 至 10000 原子以上的计算，线性缩放是唯一的选择。

基础集（basis set）

CONQUEST 用称为支持函数的局部轨道表示 Kohn–Sham 本征态或密度矩阵（等效）。这些支持函数由两个基本集合之一构成：伪原子轨道（PAO）或 blip 函数（B 样条曲线）；在 CONQUEST 中使用的主要基础函数是 PAO。PAO 生成代码包含在 CONQUEST 发行版中，其中大多数元素具有定义明确且可靠的默认基础集。最简单的选择是为每个支持功能使用一个 PAO（通常这最多可以计算 1,000 个原子）。对于超出此系统大小的对角化，将使用复合基础，其中将多个 PAO 组合为较小的一组支持功能（多站点支持功能或 MSSF）。使用 MSSF，可以在 HPC 平台上计算 3,000 多个原子。对于线性缩放，需要更注意基集（更多详细信息，请参见 Linear Scaling）。

编译安装指南

为了更好地符合可能存在的多种环境需求，笔者就自己所有的一些平台进行了测试和实践，主要分为以下几类，并以独立文章的形式分别介绍：

- ARM篇 ：该类主要包含 Mac M1，树莓派以及 ARM 服务器 。
- Intel 篇：该类主要包括普通 PC 和 x86 服务器。
- Slurm 篇：该类主要是应用于 HPC 环境下，当然也适合个人在高性能服务器上运行。

这里仅以 Intel 篇和腾讯云提供的 THPC 环境为例介绍一下从零开始编译安装 CONQUEST。由于 THPC 目前只支持 CentOS 7 镜像的 SLURM 调度，相应环境也会和个人博文中有所不同。

依赖环境

- Intel 编译器（含 icc、mpiifort）
- Intel MKL library
- FFTW
- LibXC

THPC 创建集群

请参见 THPC Slurm 调度器-快速入门 一文利用 THPC API 一键创建集群。这里采用的 THPC 集群配置信息如下 thpc.json 文件所示，然后使用以下命令即可创建 THPC 集群。

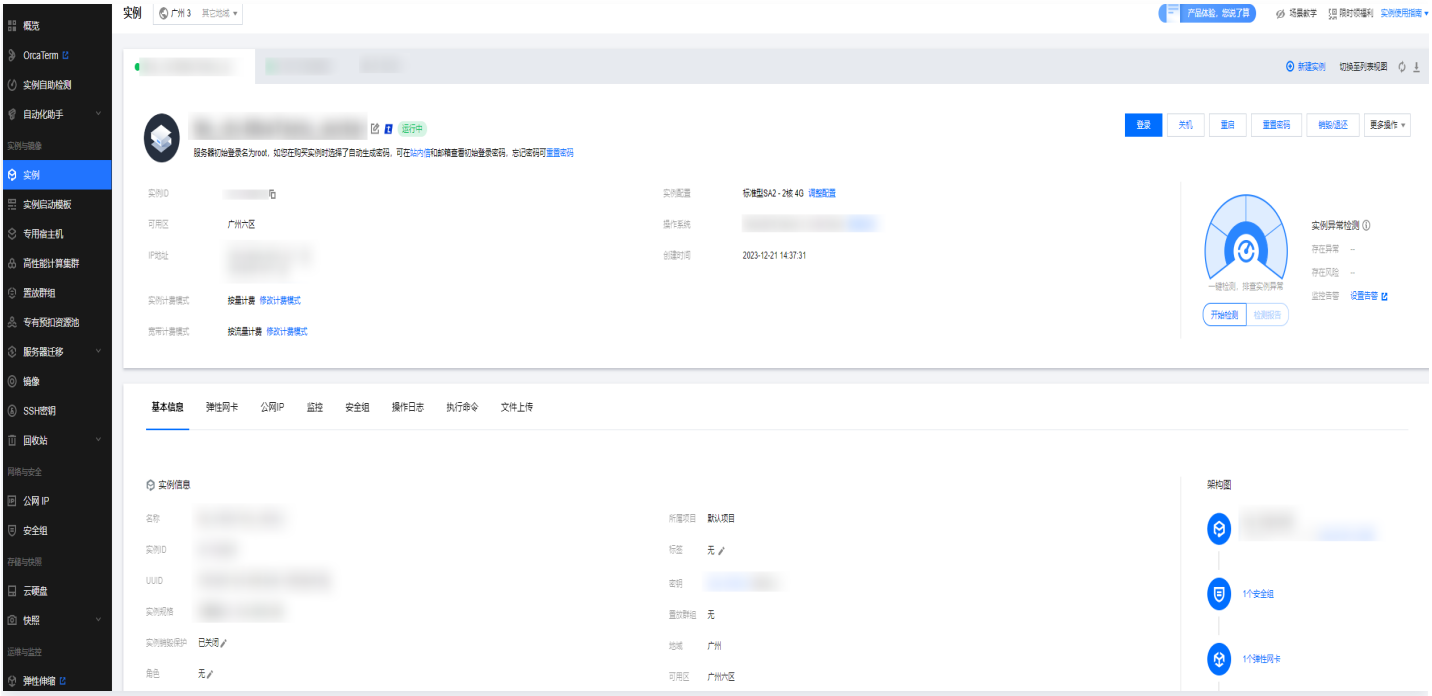
> **说明：**
> 由于是采用的按量付费方式创建集群，所以需要提前往账户里预充值超过 1 小时费用，否则会一直 INIT_FAILED 或出现创建不了 3 台 CVM。另外，请根据区域所提供的实例类型的实际情况选择合适的 InstanceType，否则也将无法正常创建。以下 json 文件中的花括号内的内容请根据实际情况自行修改成对应的内容，由于这并非是本文的目的，不再赘述。

```
{
"ManagerNodeCount": 1,
"ManagerNode": {
   "InternetAccessible": {
      "InternetMaxBandwidthOut": 10
   },
   "InstanceName": "ManagerNode",
   "InstanceType": "SA2.MEDIUM4"
},
"SchedulerType": "SLURM",
"ComputeNodeCount": 2,
"ComputeNode": {
   "InstanceType": "SA2.MEDIUM4",
   "InstanceName": "ComputeNode"
},
"LoginSettings": {
   "Password": "{Password}"
},
"Placement": {
   "Zone": "ap-shanghai-4"
},
"VirtualPrivateCloud": {
   "VpcId": "{vpc-id}",
   "SubnetId": "{subnet-id}"
},
"ImageId": "img-l8og963d",
"StorageOption": {
  "CFSOptions": [{
     "StorageType": "SD",
     "Protocol": "NFS 3.0",
     "LocalPath": "/data",
     "RemotePath": "{ip:/path}"
  }]
}
```

```
}
```

```
# 创建集群
└─$ tccli thpc CreateCluster --cli-input-json file:///Users/zhonger/thpc.json
{
    "ClusterId": "hpc-q41v44zr",
    "RequestId": "ffc824a2-c3d6-444e-97e3-9db9fbe6bf86"
}
# 查看集群状态
└─$ tccli thpc DescribeClusters --ClusterIds '["hpc-q41v44zr"]'
{
    "ClusterSet": [
        {
            "ClusterId": "hpc-q41v44zr",
            "ClusterStatus": "INITING",
            "ClusterName": "unnamed",
            "Placement": {
                "Zone": "ap-shanghai-4"
            },
            "CreateTime": "2022-11-03T04:27:48Z",
            "SchedulerType": "SLURM",
            "ComputeNodeCount": 2,
            "ComputeNodeSet": [
                {
                    "NodeId": "ins-brt5b8sp"
                },
                {
                    "NodeId": "ins-iv8t3tfz"
                }
            ],
            "ManagerNodeCount": 1,
            "ManagerNodeSet": [
                {
                    "NodeId": "ins-2zulgl4x"
                }
            ],
            "LoginNodeSet": [],
            "LoginNodeCount": 0
        }
    ],
    "TotalCount": 1,
    "RequestId": "05e1a766-d217-4bf0-b994-22217c65ce5f"
}
```

创建集群成功后访问 云服务器实例 可以看到如下创建好的实例。请使用 `yum update` 命令先对所有服务器升级软件库到最新版本。



安装 Intel OneAPI HPCKit

> ⚠ **注意：**
>
> - 由于 Intel OneAPI HPCKit 是具有商业版权的，只是允许个人或开发者学习时免费使用，而实际运行在超算或公司内部集群上，则需要购买相应的授权。
> - 软件会被自动安装在 /opt/intel 目录下，常用集群默认云盘大小为 50GB，可能容量不足；如使用 THPC API 创建集群则会自动使用 CFS 来挂载 /opt 目录。
> - CFS 默认空间大小是 10GB，如果一开始就安装 Intel OneAPI HPCKit 可能会提示空间不足，需要在腾讯云提交工单人工扩容到 100GB，或者写一些小文件触发自动扩容机制。
>
> 根据创建集群的管理节点的资源不同，安装 Intel OneAPI HPCKit 套件的时间也会不同，以 2 核 4 G AMD 为例，大概需要 20 分钟左右。

```
# root 用户执行以下内容
# 添加 Intel OneAPI 镜像源
tee > /tmp/oneAPI.repo << EOF
[oneAPI]
name=Intel® oneAPI repository
baseurl=https://yum.repos.intel.com/oneapi
enabled=1
```

```
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://yum.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-
PRODUCTS.PUB
EOF

mv /tmp/oneAPI.repo /etc/yum.repos.d

# 更新软件索引列表并安装软件
yum install -y intel-hpckit


# 使 Intel OneAPI 库生效
cd /opt/intel/oneapi/ && source setvars.sh

# 确认是否生效
icc -V
```

Environment Modules

安装 Envrionment Modules

虽然可以用上面提到的命令使得 Intel OneAPI 库生效，但是还有其他依赖环境存在，还是希望可以使用统一的方式来管理，Environment Modules 就是一个不错的选择。一般来说，超算集群上都是采用这款工具来管理不同软件及不同版本的。

```
# 准备目录
mkdir -p ~/download /opt/modules

# 预安装依赖
yum install tcl tcl-devel

# 使用 GhProxy 代理下载源码压缩包
wget -c https://ghproxy.com/github.com/cea-
hpc/modules/releases/download/v5.1.1/modules-5.1.1.tar.gz

# 解压目录
tar xfz modules-5.1.1.tar.gz

# 配置安装目录
cd modules-5.1.1
./configure --prefix=/opt/modules

# 编译及安装
make && make install
```

```
# 添加以下命令到 ~/.bashrc 中使之生效
tee -a ~/.bashrc << EOF
    source /opt/modules/init/bash
EOF

# 确认是否生效
module ava
```

> ⚠ **注意：**
> 这里由于命令安装了 tcl 依赖，所以需要也在集群的所有节点上执行预安装依赖和最后的命令生效的操作。中间的下载源代码和编译安装操作无须重复。

将 Intel OneAPI 纳入 Modules

```
# 生成 modulefiles 目录
# 执行完可以看到 :: oneAPI modulefiles folder is here: "/opt/intel/oneapi/modulefiles"
cd /opt/intel/oneapi && source modulefiles-setup.sh

# 添加该目录到 Modules（此操作只能临时添加）
module use --append /opt/intel/oneapi/modulefiles

# 确认是否添加成功
module ava
```

如果希望更方便启用 Intel OneAPI 的 modulefiels，可以创建 `/opt/modules/modulefiles/intel/2022.2` 文件，文件内容如下：

```
#%Module1.0#####################################################
##########################
##
## intel/2022.2 modulefile
##
proc ModulesHelp { } {
    puts stderr "\tThis module file will add Intel OneAPI HPCKit modulefiles path"
}

module-whatis   "adds Intel OneAPI HPCKit directory to MODULEPATH"

set basemoddir   /opt/intel/oneapi/modulefiles
module use --append $basemoddir
```

然后使用 `module ava` 命令可以看到有 `intel/2022.2` 了，使用命令加载即可显示所有 Intel OneAPI 的 modulefiles。

```
# 加载 Intel OneAPI Modulefiles
module load intel/2022.2

# 查看所有可用模块
module ava

# 加载后续软件所需编译环境
module load icc mkl mpi

# 确认是否加载成功
icc -V
mpiifort -V
echo $LIBRARY_PATH # 查看 MKL 库路径是否加载
```

FFTW

安装 FFTW

```
# 安装完整编译环境
yum groupinstall "Development Tools"

# 准备 FFTW 安装目录
mkdir -p  /opt/fftw/3.3.10

# 准备源代码目录并下载编译 FFTW
mkdir -p /opt/downloads \
&& cd ~/downloads \
&& wget -c http://www.fftw.org/fftw-3.3.10.tar.gz \
&& tar zxf fftw-3.3.10.tar.gz \
&& cd fftw-3.3.10 \
&& ./configure --prefix=/opt/fftw/3.3.10 CC=icc FC=mpiifort \
&& make \
&& make install
```

将 FFTW 纳入 Modules

```
# 创建特定 Modulefiels 目录
mkdir -p /opt/modules/modulefiles/fftw/

vim /opt/modules/modulefiles/fftw/3.3.10 # 内容如下：
```

```
#%Module
proc ModulesHelp { } {
    puts stderr "\tThis module file will load FFTW 3.3.10"
}

module-whatis  "Enable FFTW 3.3.10"

set basedir /opt/fftw/3.3.10
prepend-path PATH "${basedir}/bin"
prepend-path LIBRARY_PATH "${basedir}/lib"
prepend-path LD_LIBRARY_PATH "${basedir}/lib"
prepend-path INCLUDE_PATH "${basedir}/include"
prepend-path LD_INCLUDE_PATH "${basedir}/include"

# 确认是否生效，是否有 fftw/3.3.10
module ava

# 加载 fftw/3.3.10
module load fftw/3.3.10
```

LibXC

安装 LibXC

```
# 准备 LibXC 安装目录
mkdir -p  /opt/libxc/5.0.0

# 下载编译 LibXC
cd ~/downloads \
&& wget -c http://www.tddft.org/programs/libxc/down.php?file=5.0.0/libxc-5.0.0.tar.gz
-O libxc-5.0.0.tar.gz \
&& tar zxf libxc-5.0.0.tar.gz \
&& cd libxc-5.0.0 \
&& ./configure --prefix=/opt/libxc/5.0.0 CC=icc FC=mpiifort \
&& make \
&& make install
```

将 LibXC 纳入 Modules

```
# 创建特定 Modulefiels 目录
mkdir -p /opt/modules/modulefiles/libxc/

vim /opt/modules/modulefiles/libxc/5.0.0 # 内容如下：
```

```
#%Module
proc ModulesHelp { } {
    puts stderr "\tThis module file will load LibXC 5.0.0"
}

module-whatis  "Enable LibXC 5.0.0"

set basedir /opt/libxc/5.0.0
prepend-path PATH "${basedir}/bin"
prepend-path LIBRARY_PATH "${basedir}/lib"
prepend-path LD_LIBRARY_PATH "${basedir}/lib"
prepend-path INCLUDE_PATH "${basedir}/include"
prepend-path LD_INCLUDE_PATH "${basedir}/include"

# 确认是否生效，是否有 libxc/5.0.0
module ava

# 加载 libxc/5.0.0
module load libxc/5.0.0

# 确认已加载依赖库
module list
# 应该如下所示：
Currently Loaded Modulefiles:
 1) intel/2022.2   2) compiler-rt/latest   3) icc/latest   4) tbb/latest   5) mkl/latest   6)
mpi/latest   7) fftw/3.3.10   8) libxc/5.0.0

Key:
auto-loaded
```

编译安装 CONQUEST

下载 CONQUEST 最新源代码并切换到 develop 分支：

```
cd /opt
git clone https://ghproxy.com/github.com/OrderN/CONQUEST-release conquest
cd /opt/conquest && git checkout develop
```

修改 conquest/src/system.make 编译文件，修改后的文件内容如下所示：

```
# For CONQUEST (2022/10/19 zhonger)

# Set compilers
```

```
FC=mpiifort
F77=$FC

# Linking flags
LINKFLAGS = -L$(MKLROOT)/lib/intel64
$(MKLROOT)/lib/intel64/libmkl_blacs_intelmpi_lp64.a
$(MKLROOT)/lib/intel64/libmkl_lapack95_lp64.a -lmkl_scalapack_lp64 -lmkl_intel_lp64 -
lmkl_sequential -lmkl_core -lmkl_blacs_intelmpi_lp64  -lpthread -lm
#LINKFLAGS= -L/usr/local/lib
ARFLAGS=

# Compilation flags
COMPFLAGS= -I$(MKLROOT)/include/intel64/lp64 -I$(MKLROOT)/include
$(XC_COMPFLAGS)
#COMPFLAGS= -O3 $(XC_COMPFLAGS)
COMPFLAGS_F77= $(COMPFLAGS)

# Set BLAS and LAPACK libraries
#BLAS= -lvecLibFort

# Full library call; remove scalapack if using dummy diag module
LIBS= $(FFT_LIB) $(XC_LIB) $(BLAS)
#LIBS= $(FFT_LIB) $(XC_LIB) -lscalapack $(BLAS)

# LibXC compatibility (LibXC below) or Conquest XC library

# Conquest XC library
#XC_LIBRARY = CQ
#XC_LIB =
#XC_COMPFLAGS =

# LibXC compatibility
# Choose old LibXC (v2.x) or modern versions
#XC_LIBRARY = LibXC_v2
XC_LIBRARY = LibXC_v5
XC_LIB = -L/opt/libxc/5.0.0/lib/ -lxcf90 -lxc
XC_COMPFLAGS = -I/opt/libxc/5.0.0/include

# Set FFT library
FFT_LIB= -L/opt/fftw/3.3.10/lib/ -lfftw3
#FFT_LIB=-lfftw3
FFT_OBJ=fft_fftw3.o

# Matrix multiplication kernel type
MULT_KERN = default
```

```
# Use dummy DiagModule or not
DIAG_DUMMY =
```

使用 `make` 命令执行编译，CONQUEST 正常编译成功后会在源代码目录下的 bin 目录里面看见 `Conquest` 的可执行文件。

使用指南

同上所示，进入 `tools/BasisGeneration` 使用相同 system.make 文件编译后 bin 目录会多出一个 MakeIonFiles 的可执行文件。`Conquest` 命令是用来执行模拟，Makelonfiles 是用来生成模拟所需的原子轨道描述文件。以下就以 Li 为对象介绍一下完整的 CONQUEST 运行流程。

创建测试文件夹

由于后续会尝试使用 SLURM 作业管理系统提交任务，所以必须是在计算节点和管理节点共享的 NFS 目录里准备文件，即在 `/opt` 目录下。

```
# 创建并进入测试文件夹
mkdir -p /opt/test/Li
cd /opt/test/Li
```

准备输入文件

CONQUEST 所需的输入文件一共有三个，分别是 Conquestinput、`Li.in` 和 `Li.ion` 。这三个文件中除了 Li.ion 之外都需要自己编写，首先介绍如何得到 Li.ion 文件。

生成 Li.ion

进入 CONQUEST 源代码目录下的 pseudo-and-pao 目录，可以看到有 LDA，PBE 和 PBEsol 三个文件夹。我们这里采用 PBE 方法来模拟所以进入 PBE/Li 目录。执行 MakeIonFiles 命令就会生成我们所需的 Li.ion 文件。

> ⚠ **注意：**
>
> - MakeIonFiles 命令执行需要引用到正确的路径，否则会提示不存在该命令，所以建议对此命令建立一个别名使用更加方便。例如在 .bashrc 文件尾部中添加如下一行代码并使该配置生效即可。
> - 生成的文件名并不是 Li.ion，可以使用 `mv LiCQ.ion Li.ion` 。
> - 根据想要进行模拟的规模不同，可以分为 SZ(minimal)、SZP(small)、DZP(medium) 和 TZTP(large) 四种，可以在同目录的 Conquest_input 文件中修改 `Atom.BasisSize medium` ，来修改 Li.ion 文件的规模。

```
alias cq="/opt/conquest/bin/Conquest"
```

```
alias cqion="/opt/conquest/bin/MakeIonFiles"
```

编写 Conquest_input

Conquest_input 中完整的参数相当复杂，可以参见 Input tags，这里仅对以下文件中出现的参数做出简要解释。

名称	说明
IO	开头的配置是对输入输出的定义
Iprint	输出文件的打印类型
Title	任务的名字
Coordinates	坐标文件的文件名
General	开头的配置是对基础集、支持函数类型等的定义
FunctionalType 101	指 PBE
Pseudopotential hamann	CONQUEST 最新支持的基础集类型
NumberOfSpecies	原子的类型数
AtomMove	开头的配置是对更新迭代过程的方法的定义
TypeOfRun	运动方式，通常有 static、cg、md 等
minE	开头的配置是收敛方法的定义
SelfConsistent	是否采用自洽方式
SCTolerance	精确度
MaxIters	最多迭代次数
GridCutoff	一个关键性的参数，定义在空间中网格化的大小，随着值的变化所计算的结果也会不一样
Diag.MeshX(MeshY, MeshZ)	是 k-points 的值，也就是单位晶胞 xyz 轴的相对比值
ChemicalSpeciesLabel	定义了原子具体有什么种类，以及它们的编号和原子质量大小。此处的原子质量大小可以从 Li.ion 文件中查到

```
# Conquest_input_Li

IO.Iprint 3
IO.Title Simulation for bulk Li
IO.Coordinates Li.in
IO.FractionalAtomicCoords T

General.FunctionalType 101
General.PseudopotentialType hamann
General.Partitions Hilbert
General.NumberOfSpecies 1
General.ManyProcessors F
General.CheckDFT T

Basis.BasisSet PAOs

minE.VaryBasis F

AtomMove.TypeOfRun static

minE.SelfConsistent T
minE.LTolerance 1.0e-6
minE.SCTolerance 1.0e-6
SC.LinearMixingFactor 0.2
SC.KerkerFactor 0.01
SC.MaxIters  100
SC.MaxEarly  0
SC.MaxPulay  5

Grid.GridCutoff 150

DM.SolutionMethod diagon
Diag.kT   0.002
Diag.MPMesh T
Diag.MPMeshX 15
Diag.MPMeshY 15
Diag.MPMeshZ 15

%block ChemicalSpeciesLabel
 1  6.94  Li
%endblock
```

编写 Li.in

前三行为 Li 的晶格参数 a，b 和 c 的值，因为 Li 是标准的体心立方结构，这三个值相等。具体的值可以从 网站 中查到。注意此处使用的晶格参数的单位是波尔，与 pm 的换算为 `0.5291772 pm = 1 bohr` 。第四行代表在一个单位晶胞中有 2 个 Li 原子（中心 1 个 + 八个角 8 个 1/8），它们的位置一个在原点位置，一个在体心立方结构的中心位置，坐标如第五、六行前三个数字所示。后面的 1 表示这个位置有一个原子，T 表示原子可以运动，三个 T 表示 xyz 三个方向。

```
6.633 0.00  0.00
0.00  6.633 0.00
0.00  0.00  6.633
2
 0.000000000000E+00  0.000000000000E+00  0.000000000000E+00   1 T T T
 0.500000000000E+00  0.500000000000E+00  0.500000000000E+00   1 T T T
```

运行

由于 CONQUEST 定义了并行能使用的最大核数等于原子个数，因此在这里 Li 的计算中最多可以使用双核。如果单独使用编译成功的命令运行的话，默认用的是单核。

单核运行

```
# 在输入文件目录中执行
/opt/conquest/bin/Conquest
```

双核运行

```
# 在输入文件目录中执行
mpirun -np 2 /opt/conquest/bin/Conquest < Conquest_input > Conquest_out
```

SLURM 脚本提交

将以下内容写入 run.sh 文件，完成后使用 sbatch run.sh 命令提交任务。大概过 1 分钟左右可以看到任务完成。

```
#!/bin/bash

#SBATCH --job-name=test_Li
#SBATCH --partition=compute
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=2
#SBATCH --output=test_Li_%j.log
```

```
module load intel/2022.2 mkl mpi
module load fftw/3.3.10 libxc/5.0.0

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
mpirun -np $OMP_NUM_THREADS /opt/conquest/bin/Conquest < Conquest_input > Conquest_out
```

提交任务及查看任务状态截图：

```
[root@manager Li]# sbatch run.sh
Submitted batch job 1
[root@manager Li]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                 1   compute  test_Li     root  R       0:01      1 ip-172-16-4-3
```

集群状态截图：

```
[root@manager Li]# sbatch run.sh
Submitted batch job 1
[root@manager Li]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                 1   compute  test_Li     root  R       0:01      1 ip-172-16-4-3
```

任务执行完成后截图：

```
[root@manager Li]# squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
[root@manager Li]# ls
conquest.bib    Conquest_warnings  hilbert_make_blk.dat  Kmatrix2.i00.p000000  Li.ion
Conquest_input  coord_next.dat     InfoGlobal.i00        Kmatrix2.i00.p000001  run.sh
Conquest_out    eigenvalues.dat    input.log             Li.in                 test_Li_1.log
```

运行结果简要分析

如上图任务执行完成后，会多出 Conquest_out 等文件。如果任务被正常执行可以在 test_Li_1.log 文件中看到 run.sh 中的输出，此处内容为空。Conquest_out 文件包含了较多的结果，此处可以使用以下命令查看一些简单的信息：

```
# 查看 DFT Total Energy
[root@manager Li]# grep "* DFT" Conquest_out
        |* DFT total energy       =      -14.401641923855038 Ha

# 查看 SCF 收敛过程
[root@manager Li]# grep -i "RMS residual" Conquest_out
      Pulay iteration    1 RMS residual:          0.26022E-02
      Pulay iteration    2 RMS residual:          0.20485E-02
      Pulay iteration    3 RMS residual:          0.43764E-04
```

```
      Pulay iteration    4 RMS residual:        0.57800E-05
      Pulay iteration    5 RMS residual:        0.48574E-06
      Pulay iteration    1 RMS residual:        0.48574E-06
```

删除 THPC 集群（可选）

```
# 查看集群状态
# 由于之前的集群已被删除，这里展示的是重新创建的集群
└─$ tccli thpc DescribeClusters --ClusterIds '["hpc-qkd5sayv"]'
{
    "ClusterSet": [
        {
            "ClusterId": "hpc-qkd5sayv",
            "ClusterStatus": "RUNNING",
            "ClusterName": "unnamed",
            "Placement": {
                "Zone": "ap-shanghai-4"
            },
            "CreateTime": "2022-11-03T04:44:07Z",
            "SchedulerType": "SLURM",
            "ComputeNodeCount": 2,
            "ComputeNodeSet": [
                {
                    "NodeId": "ins-8k61ebf3"
                },
                {
                    "NodeId": "ins-qcex0n7z"
                }
            ],
            "ManagerNodeCount": 1,
            "ManagerNodeSet": [
                {
                    "NodeId": "ins-hrucqlcz"
                }
            ],
            "LoginNodeSet": [],
            "LoginNodeCount": 0
        }
    ],
    "TotalCount": 1,
    "RequestId": "64aa4df1-db2e-4c87-96a9-b2c3400e67ef"
}

# 删除指定集群
```

```
└─$ tccli thpc DeleteCluster --ClusterId "hpc-qkd5sayv"
{
    "RequestId": "242a9c73-c488-49d5-a0de-81b6e1b75b73"
}

# 验证集群状态（已经变空了）
└─$ tccli thpc DescribeClusters --ClusterIds '["hpc-qkd5sayv"]'
{
    "ClusterSet": [],
    "TotalCount": 0,
    "RequestId": "9045747d-eadf-4748-a3c9-470418644fc0"
}
```

同时也可以使用浏览器访问 控制台 进行验证（如下图所示）。