青能所超级计算平台用户 2011 年 3 月培训

李会民

hmli@ustc.edu.cn

中国科学院青岛生物能源与过程研究所 超级计算中心

2011 年 03 月

培训内容

超算平台硬件系统简介

超算系统主要由曙光天潮 TC2600 刀片服务器、天阔 A950r-F 机架服务器和天阔 A620r-FX 机架服务器组成，总计算能力达每秒 3.6 万亿次：

- TC2600 刀片：38 台，每刀片两颗主频 2.1GHz 的 AMD Opteron 2352 x86_64 四核处理器（共八核），16GB 内存，38 台刀片总浮点计算能力为每秒 2.55 万亿次
- A950r-F 服务器：两台，每台八颗主频 2.2GHz 的 AMD Opteron 8354 x86_64 四核处理器（共 32 核），64GB 内存，每台计算能力为每秒 0.28 万亿次，两台合计每秒 0.56 万亿次
- IO 兼计算节点：六台 A620r-FX 机架服务器，每台两颗主频 2.1GHz 的 AMD Opteron 2352 x86_64 四核处理器（共八核），8GB 内存，六台总计算能力每秒 0.4 万亿次
- 管理及用户登录节点：各一台 A620r-FX 机架服务器，每台两颗主频 2.1GHz 的 AMD Opteron 2352 x86_64 四核处理器（共八核），16GB 内存，两台合计总计算能力每秒 0.13 万亿次
- 计算网络：1、InfiniBand，单向 10Gb/s；2、千兆以太网
- 存储与备份：各一台 5TB（八块 750GB SATA 硬盘）的 HDS WMS 100 存

超算平台操作系统及软件简介

已部署软件：

- 操作系统：CentOS 5.4[1]
- 编译器：Intel、PGI、GNU 等 C/C++、Fortran 编译器
- 并行环境：Open MPI、MVAPICH、MVAPICH2：
  - 节点内和节点间 MPI 并行
  - 刀片和 IO 节点内的八核，A950r-F 节点内的 32 核可 OpenMP 并行
  - 节点内共享内存，节点间分布式内存的 MPI OpenMP 混合并行模式
- 数学函数库：ACML、MKL 及 BLAS、Goto BLAS、ATLAS、LAPACK、ScaLAPACK 等
- 作业管理：Platform LSF
- 运行监控：Ganglia 及自主开发软件

[1]CentOS(Community ENTerprise Operating System)是 Linux 发行版之一，它来自于 Red Hat Enterprise Linux 依照开放源代码规定释出的源代码所编译而成（主要去除 Red Hat 版权所有的一些东西）。一般可以利用 man 命令或命令加 -h 或 --help 等选项来查看其详细用法，建议参考 CentOS、Red Hat Enterprise Linux AS 手册或通用 Linux 手册。

用户登录与文件传输

- 只有已被允许的 IP 才可以登录到超算平台（所内 IP 已被允许，所外 IP 需要申请才被允许），如需要从所外登录且 IP 不是固定的，可以申请 VPN 帐户（所内 IP 已被禁止使用 VPN）。
- 用户需以 ssh[2]方式（不支持 telnet 方式登录）登录后进行编译、提交作业等操作
- 用户数据可以利用 ftp和 sftp 协议进行传输，ftp 客户端也许需要关闭被动(passive)模式。
- 用户登录进来后默认的 shell 为 bash，用户可以利用 ypchsh 命令修改为所需的 shell；用户可以在登录节点上运行 yppasswd 命令修改密码（注意：利用 passwd 命令在登录节点上修改密码无效）。
- 用户默认免费存储空间为 100GB，如果有更多需要，请向管理员申请，并支付相应费用。

[2]在 MS Windows 下可利用 putty 等支持 ssh 协议的客户端软件进行登录，putty 主页：http://www.chiark.greenend.org.uk/~sgtatham/putty/

串行及 OpenMP 程序编译

青能所超算平台可运行采用 C/C++、Fortran 编写的串行程序，及与 OpenMP 结合的并行程序。用户只需要在登录节点上以相应的编译命令和选项进行编译即可[3]。当前安装的编译环境主要为：

- Intel C/C++ Fortran 编译器
  - 系统默认使用 11.1.064 版本
  - 用户可在 ~/.bashrc 添加如下内容使用 2011.2.137(12.0.2.137)版本：

```
. /opt/intel/composerxe-2011.2.137/bin/compilervars.sh intel64
```

- PGI C/C++ Fortran 编译器
  - 系统默认使用 7.1 版本
  - 用户可在 ~/.bashrc 添加如下内容以使用 2011(11.1)版本：

```
PGI=/opt/pgi
export PGI
PATH=/opt/pgi/linux86-64/11.1/bin:$PATH
export PATH
MANPATH=$MANPATH:/opt/pgi/linux86-64/11.1/man
export MANPATH
```

- GNU C/C++ Fortran 编译器

Intel C/C++ 编译器编译举例

- icc -o yourprog yourprog.c
  将 C 程序 yourprog.c 编译为可执行文件 yourprog。
- icpc -o yourprog yourprog.cpp
  将 C++ 程序 yourprog.cpp 编译为可执行文件 yourprog。
- icc -o yourprog-omp -openmp yourprog.c
  将 OpenMP 指令并行的 C 程序 yourprog-omp.c 编译为可执行文件 yourprog-omp。

PGI C/C++ 编译器编译举例

- pgcc -o yourprog yourprog.c
  将 C 程序 yourprog.c 编译为可执行文件 yourprog。
- pgCC -o yourprog yourprog.cpp
  将 C++ 程序 yourprog.cpp 编译为可执行文件 yourprog。
- pgcc -o yourprog-omp -mp yourprog.c
  将 OpenMP 指令并行的 C 程序 yourprog-omp.c 编译为可执行文件 yourprog-omp。

GNU C/C++ 编译器编译举例

- gcc -o yourprog yourprog.c
  将 C 程序 yourprog.c 编译为可执行文件 yourprog。
- g++ -o yourprog yourprog.cpp
  将 C++ 程序 yourprog.cpp 编译为可执行文件 yourprog。
- gcc -o yourprog-omp -fopenmp yourprog.c
  将 OpenMP 指令并行的 C 程序 yourprog-omp.c 编译为可执行文件 yourprog-omp。

Intel Fortran 编译器编译举例

- ifort -o yourprog yourprog.for
  将 Fortran 77 程序 yourprog.for 编译为可执行文件 yourprog。
- ifort -o yourprog -static yourprog.f90
  将 Fortran 90 程序 yourprog.f90 静态编译为可执行文件 yourprog。
- ifort -o yourprog-omp -openmp yourprog.f90
  将 OpenMP 指令并行的 Fortran 90 程序 yourprog-omp.f90 编译为可执行文件 yourprog-omp。

PGI Fortran 编译器编译举例

- pgf77 -o yourprog yourprog.for
  将 Fortran 77 程序 yourprog.for 编译为可执行文件 yourprog。
- pgf90 -o yourprog -static yourprog.f90
  将 Fortran 90 程序 yourprog.f90 静态编译为可执行文件 yourprog。
- pgf90 -o yourprog-omp -mp yourprog.f90
  将 OpenMP 指令并行的 Fortran 90 程序 yourprog-omp.f90 编译为可执行文件 yourprog-omp。

GNU Fortran 编译器编译举例

- g77 -o yourprog yourprog.for
  将 Fortran 77 程序 yourprog.for 编译为可执行文件 yourprog。
- gfortran -o yourprog -static yourprog.f90
  将 Fortran 90 程序 yourprog.f90 静态编译为可执行文件 yourprog。
- gfortran -o yourprog-omp -fopenmp yourprog.f90
  将 OpenMP 指令并行的 Fortran 90 程序 yourprog-omp.f90 编译为可执行文件 yourprog-omp。

注意：g77 不支持 OpenMP，也不支持 Fortran 90 开始的标准，但 gfortran 支持 Fortran 77 标准。

MPI 并行程序编译

青能所超算平台的通信网络为 InfiniBand（建议使用）和千兆以太网（建议仅在不支持 InfiniBand 网络的情况下使用），系统上安装的 MPI 的并行环境为 Open MPI 和基于 MPICH 针对 InfiniBand 的版本 MVAPICH（1 和 2 分支），且都有分别对应 Intel、PGI、GNU 编译器的版本。

系统设置默认使用 /usr/mpi/intel/openmpi-1.4.1，用户可以运行 mpi-selector-menu 命令按照提示选择自己使用的 MPI 环境，注意数字后需要加 u。

MPI 并行程序编译举例

Open MPI 和 MVAPICH 的编译命令主要为：mpicc、mpic++、mpicxx、mpiCC、mpif77 和 mpif90，不同类型程序的编译命令如下：

- mpicc -o yourprog-mpi yourprog-mpi.c
  将 C 语言的 MPI 并行程序 yourprog-mpi.c 编译为可执行文件 yourprog-mpi。
- mpicxx -o yourprog-mpi yourprog-mpi.cpp
  将 C++ 语言的 MPI 并行程序 yourprog-mpi.cpp 编译为可执行文件 yourprog-mpi，也可换为 mpic++ 或 mpiCC。
- mpif77 -o yourprog-mpi yourprog-mpi.f
  将 Fortran 77 语言的 MPI 并行程序 yourprog-mpi.f 编译为可执行文件 yourprog-mpi。
- mpif90 -o yourprog-mpi yourprog-mpi.f90
  将 Fortran 90 语言的 MPI 并行程序 yourprog-mpi.f90 编译为可执行文件 yourprog-mpi。

MPI 编译命令实际上是调用 Intel、PGI、GCC 编译器进行编译。

数学函数库

青能所超算平台安装的数学函数库主要有 Intel Math Kernel Library (MKL)、AMD Core Math Library(ACML)及我们编译的数学函数库（在 /opt/lib 下，注意名称，如 libgoto_barcelonap-r1.26-intel.a 表示用 Intel 编译器编译的 goto 库），用户可以直接调用，以提高性能、加快开发。

Intel MKL

当前默认使用的是 /opt/intel/Compiler/11.1/064/mkl/lib，版本为 11.1.064，具有 i386 和 AMD64(em64t) 两个版本，分别对应的目录为 32 和 em64t，以下将以 AMD64 系统为例介绍。
在 bash 下可以通过在 ~/.bashrc 之类的文件中添加下面代码设置 MKL 所需的环境变量 INCLUDE、LD_LIBRARY_PATH 和 MANPATH 等：

```
. /opt/intel/Compiler/11.1/064/mkl/tools/environment/mklvarsem64t.sh
```

用户可在 ~/.bashrc 添加如下内容使用 2011.2.137(12.0.2.137)版本：

```
. /opt/intel/composerxe-2011.2.137/bin/compilervars.sh intel64
```

注：以下内容以 11.1.064 版本介绍。

MKL 主要内容

- 基本线性代数子系统库(BLAS)
- 离散基本线性代数库(Sparse BLAS)
- 线性代数库(LAPACK)
- 可扩展性线性代数库(ScaLAPACK)
- 离散求解程序(Sparse Solver routines)
- 向量数学库函数(Vector Mathematical Library functions)
- 向量统计库函数(Vector Statistical Library functions)
- 傅立叶变换程序(Fourier Transform functions (FFT))
- 集群版傅立叶变换程序(Cluster FFT)
- 区间求解程序(Interval Solver routines)
- 三角变换程序(Trigonometric Transform routines)
- 泊松、拉普拉斯和哈密顿求解程序(Poisson, Laplace, and Helmholtz Solver routines)
- 优化（信赖域）求解程序(Optimization (Trust-Region) Solver routines)

MKL 目录内容

目录	内容
<mkl_dir>	MKL 主目录，比如 /opt/intel/Compiler/11.1/064/mkl
<mkl_dir>/benchmarks/linpack	包含 OpenMP 版的 LINPACK 的基准程序
<mkl_dir>/benchmarks/mp_linpack	包含 MPI 版的 LINPACK 的基准程序
<mkl_dir>/doc	MKL 文档目录
<mkl_dir>/examples	一些例子，建议用户参考学习
<mkl_dir>/include	含有 INCLUDE 文件
<mkl_dir>/interfaces/blas95	包含 BLAS 的 Fortran 90 封装及用于编译成库的 makefile
<mkl_dir>/interfaces/LAPACK95	包含 LAPACK 的Fortran 90 封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw2xc	包含 2.x 版 FFTW(C 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw2xf	包含 2.x 版 FFTW(Fortran 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw3xc	包含 3.x 版 FFTW(C 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw3xf	包含 3.x 版 FFTW(Fortran 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw2x_cdft	包含 2.x 版 MPI FFTW(集群 FFT)封装及用于编译成库的 makefile
<mkl_dir>/lib/32	包含 IA32 架构的静态库和共享目标文件
<mkl_dir>/lib/em64t	包含 EM64T 架构的静态库和共享目标文件
<mkl_dir>/man/man3	MKL 的 man 文档
<mkl_dir>/tests	一些测试文件
<mkl_dir>/tools/builder	包含用于生成定制动态可链接库的工具
<mkl_dir>/tools/environment	包含用于设置环境变量的 shell 脚本
<mkl_dir>/tools/support	包含使用 Intel Premier 支持时所需要的包 ID 和许可代码等信息

注：2011.2.137(12.0.2.137)版本的主目录变更为：
/opt/intel/composerxe-2011.2.137/mkl

链接 MKL

可以采用两种方式链接：

- 在链接行中，列举含有相对或绝对路径的库名，比如：

```
<ld> myprog.o /opt/intel/Compiler/11.1/064/mkl/lib/em64t/libmkl_solver.a \
/opt/intel/Compiler/11.1/064/mkl/lib/em64t/libmkl_intel.a \
/opt/intel/Compiler/11.1/064/mkl/lib/em64t/libmkl_intel_thread.a \
/opt/intel/Compiler/11.1/064/mkl/lib/em64t/libmkl_core.a \
/opt/intel/Compiler/11.1/064/lib/intel64/libguide.so -lpthread
```

  其中 <ld> 为链接命令，比如 ld，myprog.o 是用户的目标文件。

- 首先列举所需的 MKL 库，然后跟着系统库 libpthread：
  在链接行中，利用 -L<path> 列举含有相对或绝对路径的库名（指明搜索库的路径），和 -I<include>（指明搜索头文件的路径）。

链接 MKL

如已利用前面所说的设置好 MKL 环境变量，上面则可以简化为无需指定库所在的绝对路径，只需要利用 -l<库名>指明所需要的库即可：

```
<files to link>
-L<MKL path> -I<MKL include>
[-I<MKL include>/{32|em64t|{ilp64|lp64}|64/{ilp64|lp64}}]
[-lmkl_blas{95|95_ilp64|95_lp64}]
[-lmkl_lapack{95|95_ilp64|95_lp64}]
[<cluster components>]
-lmkl_{intel|intel_ilp64|intel_lp64|intel_sp2dp|gf|gf_ilp64|gf_lp64}
-lmkl_{intel_thread|gnu_thread|pgi_thread|sequential}
[-lmkl_lapack] -lmkl_core
{-liomp5|-lguide} [-lpthread] [-lm]
```

注意：上面是动态链接的命令，如果想静态链接，需要将含有 -l 的库名用含有库文件的路径来代替，比如用 $MKLPATH/libmkl_core.a 代替 -lmkl_core，$MKLPATH 为用户定义的指向 MKL 库目录的环境变量。

ACML 简介

AMD Core Math Library (ACML) 是一些数值函数的组合，并且特别针对 AMD64 平台处理器（如 Opteron）等做了优化。ACML 函数库具有 FORTRAN 77 和 C 语言接口，主要包含：

- BLAS - 基本线性代数子系统库，包含级别一的离散基本线性代数 (Sparse BLAS)
- LAPACK - 线性代数库
- FFT - 傅立叶变换程序
- RNG - 随机数发生器和统计分布函数

ACML 安装目录

当前 ACML 安装在 /opt/acml4.4.0，按照编译器等分为几个目录：

目录	内容
ifort32	针对 64 位 Intel 的编译器，不启用 OpenMP 并行
ifort32_mp	针对多核下 32 位 Intel 的编译器，启用 OpenMP 并行
ifort64	针对 64 位 Intel 的编译器，不启用 OpenMP 并行指令
ifort64_mp	针对多核下 64 位 Intel 的编译器，启用 OpenMP 并行
pgi32	针对 64 位 PGI 的编译器，不启用 OpenMP 并行
pgi32_mp	针对多核下 32 位 PGI 的编译器，启用 OpenMP 并行
pgi64	针对 64 位 PGI 的编译器，不启用 OpenMP 并行
pgi64_mp	针对多核下 64 位 PGI 的编译器，启用 OpenMP 并行
gfortran32	针对 64 位 gfortran 的编译器，不启用 OpenMP 并行
gfortran32_mp	针对多核下 32 位 gfortran 的编译器，启用 OpenMP 并行
gfortran64	针对 64 位 gfortran 的编译器，不启用 OpenMP 并行
gfortran64_mp	针对多核下 64 位 gfortran 的编译器，启用 OpenMP 并行

链接编译举例

注意：

- 32 和 64 分别表示编译成 32 和 64 位的可执行程序
- 含有 _mp 的表示编译成 OpenMP 并行的可执行程序
- 需要在自己的环境设置文件（如 ~/.bashrc）中设置以下环境变量，以保证运行时能找到此库，其中 COMPILER 为类似 ifort64 的目录：

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/acml4.4.0/COMPILER/lib
```

Intel 编译器编译命令

- ifort driver.f -L/opt/acml4.4.0/ifort64/lib -lacml
- gcc -c -I/opt/acml4.4.0/ifort64/include driver.c
  ifort -nofor-main driver.o -L/opt/acml4.4.0/ifort64/lib -lacml
- ifort -openmp driver.f -L/opt/acml4.4.0/ifort64_mp/lib -lacml_mp
- ifort -openmp driver.f -L/opt/acml4.4.0/ifort32_mp/lib -lacml_mp

注意：因为太长，以 ifort 开头的行实际与紧接着的下一行为同一行

PGI 编译器编译命令

- pgf77 -tp=k8-64 -Mcache_align driver.f -L/opt/acml4.4.0/pgi64/lib -lacml
- pgf77 -tp=k8-32 -Mcache_align driver.f -L/opt/acml4.4.0/pgi32/lib -lacml
- pgf77 -tp=k8-64 -mp -Mcache_align driver.f -L/opt/acml4.4.0/pgi64_mp/lib -lacml_mp
- pgf77 -tp=k8-32 -mp -Mcache_align driver.f -L/opt/acml4.4.0/pgi32_mp/lib -lacml_mp
- pgcc -c -tp=k8-64 -mp -Mcache_align -I/opt/acml4.4.0/pgi64_mp/include driver.c
  pgcc -tp=k8-64 -mp -Mcache_align driver.o -L/opt/acml4.4.0/pgi64_mp/lib -lacml_mp -lpgftnrtl -lm

注意：因为太长，以 pgf77 开头的行实际与紧接着的下一行为同一行

GNU 编译器编译命令

- gfortran -m64 driver.f -L/opt/acml4.4.0/gfortran64/lib -lacml
- gfortran -m64 driver.f -L/opt/acml4.4.0/gfortran64/lib -static -lacml
- gfortran -m64 driver.f /opt/acml4.4.0/gfortran64/lib/libacml.a
- gfortran -m32 driver.f -L/opt/acml4.4.0/gfortran32/lib -lacml
- gfortran -fopenmp -m64 driver.f -L/opt/acml4.4.0/gfortran64_mp/lib -lacml_mp
- gfortran -fopenmp -m32 driver.f -L/opt/acml4.4.0/gfortran32_mp/lib -lacml_mp
- gcc -m64 -I/opt/acml4.4.0/gfortran64/include driver.c -L/opt/acml4.4.0/gfortran64/lib -lacml -lgfortran

注意：因为太长，以 gfortran 或 gcc 开头的行实际与紧接着的下一行为同一行

ACML_MV：快速数学和向量数学库

- ACML_MV 是一些常用数学函数的快速和/或向量化的库，比如 sin、cos、exp 等
- ACML_MV 库充分发挥了 AMD64 架构性能优势，这些函数精度非常高，现在只有 64 位的 ACML
- 快速数学库的函数名，一般以 fast 开头，如对应 cos 函数的 fastcos
- 向量数学函数库一般以 vrd2_、vrd4_、vrda_、vrs4_、vrs8_、vrsa_ 开始，其中 d 表示双精度，s 表示单精度，a 表示数组，2、4、8 分别表示输入参数的数目，例如 vrd4_log 表示具有 4 个双精度输入参数的向量化的 log 函数
- 调用针对 Intel 64 位编译器的库：
  -L/opt/acml4.4.0/ifort64/lib -lacml_mv

作业管理系统 LSF 的使用

青能所超算平台利用 Platform 公司的 LSF 进行资源和作业管理，所有需要运行的作业均必须通过作业提交命令（如 bsub ）提交，提交后可利用相关命令查询作业状态等。为了利用 bsub 提交作业，需要在 bsub 中指定各选项和需要执行的程序。注意：

- 不要在登录节点直接运行（编辑、编译、查看文件、压缩解压缩等非计算任务除外），以免影响其余用户的正常使用
- 如果不通过作业调度系统直接在计算节点上运行将会被监护进程直接杀掉
- 申请的计算资源必需不少于实际使用的资源，否则作业（如申请4核，实际使用6核的计算资源）将会被监护进程直接杀掉

提交作业：bsub

bsub 是用户最常用的作业提交命令，其基本格式为：

bsub [options] command [arguments]

- options 设置队列、CPU 核数等 LSF 的选项，必需在 command 之前，否则将作为 command 的参数
- arguments 设置作业的可执行程序本身所需要的参数，必须在 command 之后，否则将作为设置队列等的选项

提交到特定队列：bsub -q

-q 选项可以指定提交到哪个队列，青能所超算平台现有的队列为[4]：

- normal：所需要的 CPU 核数不超过八个时，作业将在 io1 - io5，node10 - node47 上运行，此为默认队列。
- long：所需要的 CPU 核数超过八个时，作业将在 io1 - io5，node10 - node46 上运行。
- serial：所需要的 CPU 核数为一个时，作业将在 node47 上运行。
- fat：所需要的 CPU 核数超过八个，但不超过 32 个并需共享内存时，作业将只在 node48 上运行。

提交到 normal 队列运行串行程序 executable1：

bsub -q normal executable1 或 bsub executable1

如果提交成功，将显示类似下面的输出：

```
Job <79722> is submitted to default queue <normal>.
```

其中 79722 为此作业的作业号，以后可利用此作业号来进行查询及终止等操作。

[4]队列会针对运行情况修改，请参看登录后提示或运行 bqueues -l 命令查看

指明所需要的 CPU 核数：bsub -n

- -n 选项指定可所需要的 CPU 核数（一般来说核数和进程数一致）
- 指定利用八个核（由 -n 8 指定）运行 MPI（由 mpiibjob 或 mpip4job 指定）程序：

```
bsub -q normal -n 8 mpiibjob executable-mpi1
```

运行串行作业：bsub -q serial

运行串行作业，请使用 serial 队列：

```
bsub -q serial executable-serial
```

运行 MPI 作业：bsub -n NUM mpiibjob|mpip4job

- 运行 MPI 作业，需要利用 mpiibjob 或 mpip4job 调用可执行程序，并用 -n 选项指定所需的 CPU 核数
- 利用 16 颗核运行使用 InfiniBand 网络的 MPI 程序 executable-mpi1：

  bsub -q long -n 16 mpiibjob executable-mpi1
- 利用 16 颗核运行使用千兆以太网的 MPI 程序 executable-mpi1：

  bsub -q long -n 16 mpip4job executable-mpi1

注：mpiibjob 和 mpip4job 分别指定调用使用 InfiniBand 和千兆以太网的应用，具体需使用哪个命令，不能任意决定，需由编译成的可执行文件使用的网络决定，系统配置的默认编译环境为使用 InfiniBand 网络（性能比千兆以太网高，建议使用），但二进制发布的非在本系统上编译的可执行程序也许使用千兆以太网。

运行 OpenMP 共享内存作业：bsub -a openmp -R ``span[hosts=1]''

集群只能在同一个节点内部运行 OpenMP 共享内存的作业，此时需要添加 -a openmp -R ``span[hosts=1]'' 选项：

```
bsub -a openmp -R ``span[hosts=1]'' -q normal -n 8
executable-omp1
```

运行排他性或共享内存作业：bsub -x

如果需要独占节点运行，此时需要添加 -x 选项：

```
bsub -x -q normal -n 4 executable-omp1
```

注意：

- 排他性运行在运行期间，不允许其余的作业提交到运行此作业的节点，并且只有在某节点没有任何其余的作业在运行时才会提交到此节点上运行
- 如果不需要采用排他性运行，请不要使用此选项，否则将导致作业必须等待完全空闲的节点才会运行，也许将增加等待时间
- 另外使用排他性运行时，哪怕只使用某节点内的一个核，也将按照此节点内的所有 CPU 核数进行收费

指明输入、输出文件运行：bsub -i -o -e

- 作业的输入文件、正常屏幕输出到的文件和错误屏幕输出的文件可以利用 -i、-o 和 -e 选项来分别指定，运行后可以通过查看指定的这些输出文件来查看运行状态，文件名可利用 %J 与作业号挂钩
- 如指定 executable1 的输入、正常和错误屏幕输出文件分别为 executable1.input、executable1-%J.log 和 executable1-%J.err：

```
bsub -i executable1.input -o executable1-%J.log -e
executable1-%J.err executable1
```

交互式运行作业：bsub -I

如果需要运行交互式的作业（如在运行期间需要手动输入参数等进行交互），需要结合 -I 参数，建议只是在调试期间使用，平常作业还是尽量不要使用此选项，类似选项还有 -Ip 和 -Is：

```
bsub -I executable1
```

终止作业：bkill

利用 bkill 命令可以终止某个运行中或者排队中的作业，比如：

bkill 79722

运行成功后，将显示类似下面的输出：

```
Job <79722> is being terminated
```

挂起作业：bstop

利用 bstop 命令可临时挂起某个作业以让别的作业先运行，例如：
bstop 79727
运行成功后，将显示类似下面的输出：

```
Job <79727> is being stopped.
```

- 可以将排在队列前面的作业临时挂起，以让后面的作业先运行。
- 虽然也可以作用于运行中的作业，但并不会因为此作业被挂起而允许其余作业占用此作业所占用的 CPU 运行，实际资源不会释放，建议不要随便对运行中的作业进行挂起操作
- 如果运行中的作业不再想继续运行，请用 bkill 终止。

继续运行被挂起的作业：bresume

利用 bresume 命令可继续运行某个挂起某个作业，例如：

bresume 79727

运行成功后，将显示类似下面的输出：

```
Job <79727> is being resumed.
```

设置作业最先运行：btop

利用 btop 命令可最先运行排队中的某个作业，例如：

btop 79727

运行成功后，将显示类似下面的输出：

```
Job <79727> has been moved to position 1 from top.
```

设置作业最后运行：bbot

利用 bbot 命令可设定最后运行排队中的某个作业，例如：

bbot 79727

运行成功后，将显示类似下面的输出：

```
Job <79727> has been moved to position 1 from bottom.
```

修改排队中的作业选项：bmod

利用 bmod 命令可修改排队中的某个作业的选项，比如想将排队中的运行作业号为 79727 的作业的执行命令修改为 executable2 并且换到 fat 队列，可以：

```
bmod -Z executable2 -q fat 79727
```

```
Parameters of job <79727> are being changed.
```

查看作业的排队和运行情况：bjobs

利用 bjobs 可以查看作业的运行情况，例如：

bjobs

```
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
79726 hmli RUN normal user    2*node31 *executab1 Mar 12 19:20
                              1*node4
79727 hmli PEND long user               *executab2 Mar 12 19:20
```

显示作业 79726 分别在 node31 和 node4 上运行 2、1 个进程；作业 79727 处于排队中尚未运行，查看详细可以利用：

bjobs -l 79727

```
Job Id <79727>, User <hmli>, Project <default>, Status <PEND>,
Queue <long> , Command <executab2>
Sun Mar 12 14:15:07: Submitted from host <user.qibebt.ac.cn>,
CWD <$HOME>, Requested Resources <type==any && swp>35>;
PENDING REASONS:
SCHEDULING PARAMETERS:
           r15s r1m r15m ut pg  io  ls   it   tmp   swp   mem
loadSched -    0.7 1.0  - 4.0 -   -    -    -     -     -
loadStop  -    1.5 2.5  - 8.0 -   -    -    -     -     -
```

从上面 PENDING REASONS: 后的信息可以看出没运行的原因，也可直接运行 bjobs -p 79727 查看没运行原因。

查看运行中作业的屏幕正常输出：bpeek

利用 bpeek 命令可查看运行中作业的屏幕正常输出，例如：

bpeek 79727

<< output from stdout >>

Radius(nm): 300.000

如果在运行中用 -o 和 -e 分别指定了正常和错误屏幕输出，也可以通过直接查看指定的文件的内容来查看屏幕输出。

查看各节点的运行情况：lsload

利用 lsload 命令可查看当前各节点的运行情况，例如：

```
lsload
```

```
HOST_NAME status r15s r1m r15m ut pg ls it tmp swp mem
node10    ok     0.0  0.0 0.0 0% 3.5 0 2050 9032M 4000M 16G
node11    locku  0.0  0.0 0.0 0% 3.5 0 2050 9032M 4000M 16G
```

ut 列表示利用率，status 列中的 locku 表示在进行排他性运行。

查看各节点的空闲情况：bhosts

利用 bhosts 命令可查看当前各节点的空闲情况，例如：

bhosts

```
HOST_NAME STATUS JL/U MAX NJOBS RUN SSUSP USUSP RSV
node12    closed   -   4     2   2     0     0    0
node10    ok       -   2     2   1     0     0    0
```

STATUS 列中的 ok 表示可以接收新作业，closed 表示已经被占满。

查看队列情况：bqueues

利用 bqueues 可以查看现有队列信息，例如：

bqueues

```
QUEUE_NAME PRIO STATUS          MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
normal          30  Open:Active   -  336    -    -     77    0   77    0
long            30  Open:Active   -  336    -    -     64    0   64    0
fat             29  Open:Active   -   32    -    -     32    0   32    0
serial          20  Open:Active   -    8    -    -      0    0    0    0
gene            20  Open:Active   -   32    -    -      8    0    8    0
```

主要列的含义为：

- QUEUE_NAME：队列名
- PRIO：优先级，数字越大优先级越高
- STATUS：状态。Open:Active 表示已激活，可使用；Closed:Active 表示已关闭，不可使用
- MAX：队列对应的最大 CPU 核数，- 表示无限，以下类似
- JL/U：单个用户同时可以的 CPU 核数
- NJOBS：排队、运行和被挂起的总作业所占 CPU 核数
- PEND：排队中的作业所需 CPU 核数
- RUN：运行中的作业所占 CPU 核数
- SUSP：被挂起的作业所占 CPU 核数

查看用户信息：busers

利用 busers 可以查看用户信息，例如：

```
busers hmli
```

```
USER/GROUP  JL/P  MAX  NJOBS  PEND  RUN  SSUSP  USUSP  RSV
hmli          -   22     40    32    8      0      0    0
```