中国科学技术大学超级计算中心

曙光TC4600百万亿次超级计算系统使用指南-简略版

李会民

2016年9月14日

目录

1 前言

本用户使用指南简略版主要将对在中国科学技术大学超级计算中心曙光TC4000百万亿次超级计算系统上进行编译以及运行作业做一基本介绍，详细信息请参看《曙光TC4600百万亿次超级计算系统用户使用指南》等相应的文档。

为了便于查看，主要排版约定如下：

- 文件名：*/path/file*
- 环境变量：*MKLROOT*
- 命令：*command parameters*
- 脚本文件内容或长命令：

```
export OPENMPI=/opt/openmpi/1.8.2_intel-compiler-2015.1.133
export PATH=$OPENMPI/bin:$PATH
export MANPATH=$MANPATH:$OPENMPI/share/man
```

- 命令输出：

```
QUEUE_NAME    PRIO STATUS       MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
serial         50  Open:Active    -   16    -    -     0    0    0    0
long           40  Open:Active    -    -    -    -     0    0    0    0
normal         30  Open:Active    -    -    -    -     0    0    0    0
```

由于受水平和时间所限，错误和不妥之处在所难免，欢迎指出错误和改进意见，本人将尽力完善。本指南会经常更新，请从超算中心主页上下载更新后的手册。

2 曙光TC4600百万亿次超级计算系统简介

中国科学技术大学超级计算中心曙光TC4600百万亿次超级计算系统，主要由曙光TC4600E刀片服务器、曙光I620机架式服务器及曙光Parastor100并行存储构成，共计一个管理节点、一个用户登录节点、五个存储节点及300个计算节点，其中计算节点为7200个CPU核心，总双精度峰值计算能力为每秒288万亿次。主要参数为：

- 刀片计算节点：
  - 300台曙光CB60-G21刀片
  - 节点名：node1-node300
  - 各节点配置：
    * 两颗64位主频2.5GHz、30MB高速L3缓存的Intel Xeon E5-2680 v3 x86_64 12核CPU（共24核），支持AVX2指令[1]
    * 64GB DDR4 2133MHz内存
    * 一块300GB SAS硬盘
    * 一块56Gbps FDR InfiniBand卡
  - 刀片机箱型号：TC4600E
- 存储系统：
  - 曙光DS800-G10机架式存储，34块4TB SATA硬盘，实际可用空间102TB
  - IO节点：
    * 节点名：io1-io4
    * 四台曙光I620-G10机架式服务器
    * 各节点配置：
      · 两颗64位主频2.1GHz的Intel Xeon E5-2620 v2 x86_64 6核CPU，共12核
      · 32GB DDR3 1600MHz内存
      · 两块600GB SAS硬盘
      · 一块56Gbps FDR InfiniBand卡
      · 一块8Gbps FC卡
    * 节点名：io5
      · 曙光I620-G10机架式服务器
      · 两颗64位主频2.1GHz的Intel Xeon E5-2640 v3 x86_64 6核CPU，共16核
      · 64GB DDR3 1600MHz内存

[1] 高级矢量扩展，理论性能：整数和浮点性能翻倍

· 12块4TB SATA硬盘，实际可用空间102TB
· 一块56Gbps FDR InfiniBand卡

– 文件系统：lustre并行文件系统

- 计算网络：56Gbps FDR InfiniBand准全线速高速网
- 操作系统：x86_64架构的64位CentOS 6.7 Linux
- 编译器：Intel、PGI和GNU等C/C++ Fortran编译器
- 数值函数库：Intel MKL
- 并行环境：Intel MPI和Open MPI等，支持MPI并行程序；各节点内的CPU共享内存，节点内既支持分布式内存的MPI并行方式，也支持共享内存的OpenMP并行方式；同时支持在节点内部共享内存，节点间分布式内存的混合并行模式。
- 资源管理和作业调度：IBM Platform LSF
- 常用公用软件安装目录：*/opt*。请自己查看有什么软件，有些软件需要在自己*~/.bashrc*等配置文件中设置后才可以使用。

图 1: 曙光TC4600百万亿次超级计算系统

3 用户登录与文件传输

本超算系统的操作系统为x86_64架构的64位CentOS 6.7 Linux，不支持TELNET方式登录，用户需以SSH方式（在MS Windows下可利用PuTTY、Xshell等支持SSH协议的客户端软件[2]）登录到用户登录节点（节点名tc4600）后进行编译、提交作业等操作。用户数据可以利用FTP和SFTP协议进行数据传输。为了安全，用户若短时间内3次密码错误登录，那么登录时所使用IP将被自动封锁10分钟，可以等待10分钟后再尝试，或换个IP登录，或联系超算中心老师解封。

本超算系统可从校内IP登录，校外一般无法直接访问。如果需要从校外等登录，可以使用学校的VPN（教师的网络通带有此功能，学生的不带），或者申请校超算中心VPN（http://scc.ustc.edu.cn/vpn/）。超算中心VPN只能用户访问超算服务器，无法访问校内外其它资源。

用户可以在登录节点上运行*yppasswd*命令修改密码（利用*passwd*命令修改密码无效）。请不要设置简单密码和向无关人员泄漏密码，以免给用户造成损失。如果忘记密码，请邮件（sccadmin@ustc.edu.cn）联系中心老师申请重置，并需提供帐号名、所在服务器系统名称等必要信息。

用户登录进来的默认语言环境为zh_CN.UTF-8中文[3]，以方便查看登录后的中文提示。如果希望使用英文或GBK中文，可以在自己的~/*.bashrc*中添加export LC_ALL=C或export LC_ALL=zh_CN.GBK。

针对zh_CN.UTF-8的PuTTY客户端配置修改如下[4]：

- 打开PuTTY主程序，选择SESSION登入到服务器
- 点击左上角change setting....，打开设置面板
- 选择window − > Appearance − > Font settings − > Change...，选择Fixedsys字体，字符集选择CHINESE_GB2312
- 在window − > Appearance − > Translation中，Received data assumed to be in which character set中，把Use font encoding改为UTF-8
- 切换到session选项，选中常用的那个，点击SAVE，把这些设置保存在session里面（否则下次打开又不支持中文）。

登录进来后请注意登录后的中文提示，或运行*cat /etc/motd*查看登录提示，也可以运行*faq*命令查看常见问题的回答。

[2]客户端下载：http://scc.ustc.edu.cn/yhsq/dlrjxz/
[3]SSH Secure Shell Client不支持UTF-8中文，不建议使用
[4]PuTTY中文设置：http://scc.ustc.edu.cn/yhsq/dlrjxz/200910/t20091014_13029.html

账户开设时，默认每个用户最大可用50GB磁盘存储空间。运行*lfs quota -h /home*命令可以查看当前自己的磁盘空间使用情况：

```
Disk quotas for user hmli (uid 503):
     Filesystem  used  quota  limit  grace  files  quota  limit  grace
          /home  6.9G    50G    50G      -  26096      0      0      -
Disk quotas for group nic (gid 505):
     Filesystem  used  quota  limit  grace  files  quota  limit  grace
          /home 164.8G     0k     0k      - 143674      0      0      -
```

其中第三行显示的是用户的磁盘情况，第六行为用户所在组的磁盘情况。

- Filesystem：使用的文件系统
- used：实际使用的磁盘空间
- quota：磁盘空间软限额
- limit：磁盘空间硬限额

您也可运行*du -hs 目录*可以查看目录占用的空间。请及时清除不需要的文件，以便释放空间。如需要更大存储空间，请与管理人员联系。

超算中心不提供数据备份服务，数据一但丢失或误删将无法恢复，请务必及时下载保存自己的数据。

本超算系统采用x86_64的64位CentOS 6.7 Linux操作系统。CentOS(Community ENTerprise Operating System)是Linux主流发行版之一，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。由于出自同样的源代码，因此有些要求高度稳定性的服务器以CentOS替代商业版的Red Hat Enterprise Linux使用。两者的不同在于CentOS并不包含封闭源代码软件。一般来说可以用*man 命令*或命令加-h或–help等选项来查看该命令的详细用法，详细信息可参考CentOS、Red Hat Enterprise Linux手册或通用Linux手册。

4 利用module设置编译及运行环境

本系统安装了多种编译环境及应用等，为方便用户使用，配置有Environment Modules工具，用户可以利用*module*命令设置、查看所需要的环境等。一般编译和运行程序时可用*module load modulefile*加载对应的模块，如不想每次都手动加，可将其设置在*~/.bashrc*或*~/.modulerc*文件中：

- *~/.bashrc*，只Bash启动时设置：

```
module load intel/2016.3.210
```

- *~/.modulerc*，每次module命令启动时都设置：

```
#%Module1.0
module load intel/2016.3.210
```

注意第一行#%Module1.0是需要的。

module基本语法为：*module [switches] [sub-command] [sub-command-args]*

常用开关参数（switches）：

- –help, -H：显示帮助。
- –force, -f：强制激活依赖解决。
- –terse, -t：以短格式显示。
- –long, -l：以长格式显示。
- –human, -h：以易读方式显示。
- –verbose, -v：显示module命令执行时的详细信息。
- –silent, -s：静默模式，不显示出错信息等。
- –icase, -i：搜索时不区分大小写。

常用子命令（sub-command）：

- *avail [path...]*：显示MODULEPATH环境变量中设置的目录中的某个目录下可用的模块，如有参数指定，则显示MODULEPATH中符合这个参数的路径。如*module avail*：

```
------------------------------------------------------------/opt/Modules/app ------------------------------------------------------------
abacus/1.0.0/intelmpi/5.0.2.044_intel-compiler-2015.1.133      matlab/2015a
abacus/1.0.0/openmpi/5.0.2.044_intel-compiler-2015.1.133       matlab/2016a
ansys/cfx/15.0.17                                              namd/2016.5.4
ansys/cfx/16.1.0                                               R/3.2.3
ansys/fluent/15.0.17                                           siesta/3.2-p1-5/avx2
ansys/fluent/16.1.0                                            siesta/3.2-p1-5/noavx2
cp2k/2.5.1                                                     vasp/5.2.11/intel-mpi-5.0.2.044_intel-compiler-2015.1.133
espresso/5.1                                                   vasp/5.2.11/intel-mpi-5.0.2.044_intel-compiler-2015.1.133-without-DNGhalf
gaussian/g09/D01                                               vasp/5.3.5/intel-mpi-5.0.2.044_intel-compiler-2015.1.133-with-avx
gromacs/5.0.4/avx256                                           vasp/5.3.5/intel-mpi-5.0.2.044_intel-compiler-2015.1.133-without-avx
gromacs/5.0.4/sse2                                             vasp/5.3.5/intel-mpi-5.1.1.109_intel-compiler-2016.0.109
lammps/2014-12-09                                              vasp/5.3.5/intel-mpi-5.1.3.181_intel-compiler-2016.2.181
lammps/2015-05-15                                              vasp/5.3.5/openmpi-2.0.0_intel-compiler-2016.2.181
lammps/2016-05-14                                              voro++/0.4.6
MaterialsStudio/6.0

------------------------------------------------------------/opt/Modules/compiler ------------------------------------------------------------
gcc/3.4.6     gcc/4.8.5     intel/2015.0.090 intel/2016.0.109 intel/2016.3.210 pgi/14.10
gcc/4.4.7     gcc/4.9.2     intel/2015.1.133 intel/2016.2.181 intel/2017.0.098 pgi/16.7

------------------------------------------------------------/opt/Modules/lib ------------------------------------------------------------
fftw/3.2.1/gcc/4.4.7    mkl/11.2               mkl/11.3               mkl/2017.0
fftw/3.3.4/intel/2015.1.133 mkl/11.2.1         mkl/11.3.3

------------------------------------------------------------/opt/Modules/mpi ------------------------------------------------------------
intelmpi/2017.0.098        intelmpi/5.1.1.109         openmpi/1.10.2/pgi/16.7    openmpi/2.0.0/intel/2016.2.181
intelmpi/5.0.1.035         intelmpi/5.1.3.181         openmpi/1.6.5/pgi/14.10    openmpi/2.0.0/pgi/14.10
intelmpi/5.0.2.044         openmpi/1.10.2/intel/2016.2.181 openmpi/1.8.2/pgi/14.10

------------------------------------------------------------/opt/Modules/python ------------------------------------------------------------
python/2.7.12  python/3.4.0

------------------------------------------------------------/opt/Modules/tool ------------------------------------------------------------
advisor/2015.1.0.367266 clck/3.1.2.006      inspector/2015.1.2.379161 itac/9.0.2.045    vtune/2015.1.0.367959
advisor/2015.1.10.380555 cmake/3.1.1        inspector/2017.1.0.475470 itac/9.1.1.017    vtune/2015.1.1.380310
advisor/2017.1.0.477503 cmake/3.6.1         itac/2017.0.020       julia/0.4.5           vtune/2016.3.0.463186
clck/2017.0.014       inspector/2015.1.0.366509 itac/9.0.1.033    rar/5.20b4            vtune/2017.3.0.463186
```

解释:

- /opt/Modules/mpi：模块所在的目录，由MODULEPATH环境变量中设定。
- openmpi/2.0.0/intel/2015.3.187：模块名或模块文件modulefile，此表示此为2.0.0版本Open MPI，而且是采用2015.3.187版本Intel编译器编译的。

- *help␣[modulefile...]*:显示每个子命令的用法,如给定modulefile参数,则显示modulefile中的帮助信息。
- *add|load␣modulefile...*:加载modulefile中设定的环境,如*modue␣load␣intelmpi/5.1.3.210*。
- *rm|unload␣modulefile...*:卸载已加载的环境modulefile,如*modue␣unload␣intelmpi/5.1.3.210*。
- *swap|switch␣[modulefile1]␣modulefile2*:用modulefile2替换当前已加载的modulefile1,如modulefile1没指定,则交换与modulefile2同样根目录下的当前已加载modulefile。
- *show|display␣modulefile...*： 显示modulefile环境变量信息。如 *module␣show␣openmpi/2.0.0/intel/2015.3.187*

```
/opt/Modules/mpi/openmpi/2.0.0/intel/2015.3.187:

module-whatis  Open MPI 2.0.0 utilities work with Intel compiler 2015.3.187
module         load intel/2015.3.187
prepend-path   PATH /opt/openmpi/2.0.0/intel/2015.3.187/bin
prepend-path   LD_LIBRARY_PATH /opt/openmpi/2.0.0/intel/2015.3.187/lib
prepend-path   INCLUDE /opt/openmpi/2.0.0/intel/2015.3.187/include
prepend-path   MANPATH /opt/openmpi/2.0.0/intel/2015.3.187/share/man
```

  - 第一行是modulefile具体路径
  - module-whatis：模块说明，后面可用子命令whatis、apropos、keyword等显示或搜索
  - module load：表示自动加载的模块
  - prepend-path：表示将对应目录加到对应环境变量的前面

- *clear*：强制module软件相信当前没有加载任何modulefiles。
- *purge*：卸载所有加载的modulefiles
- *refresh*：强制刷新所有当前加载的不安定的组件。一般用于aliases需要重新初始化，但环境比那两已经被当前加载的模块设置了的派生shell中。
- *whatis [modulefile...]*:显示modulefile中module-whatis命令指明的关于此modulefile的说明，如果没有指定modulefile，则显示所有modulefile的。
- *apropos|keyword string*:在modulefile中module-whatis命令指明的关于此modulefile的说明中搜索关键字，显示符合的modulefile。

其它一些不常用命令及参数，请*man module*查看，用户也可自己生成自己所需要的modulefile文件，并用module来设置，具体请*man module*及*man modulefile*。

5 串行及OpenMP程序编译及运行

用户只需在登录节点(tc4600)上以相应的编译命令和选项进行编译即可。当前安装的编译环境主要为：

- C/C++、Fortran编译器：Intel、PGI和GNU编译器，支持OpenMP并行。
- MPI并行环境：Intel MPI和Open MPI并行环境。

当前设置为默认使用Intel Parallel Studio XE 2015集群版（含Intel C/C++/Fortran编译器2015.1.133、Intel MKL 11.2和Intel MPI 5.0.2.044等），安装目录为*/opt/intel/composer_xe_2015.1.133*。用户也可以单独设置自己所需的串行编译环境运行，如：

- *module␣load␣intel/2016.3.210*
- 在*~/.bashrc*中设置（设置完成后需要*source␣~/.bashrc*或重新登录以便设置生效）：

```
. /opt/intel/parallel_studio_xe_2015/bin/psxevars.sh intel64
```

注意：在*~/.bashrc*中设置的级别有可能要高于使用*module␣load*设置的，可以运行*icc␣-v*或*which␣icc*等命令查看实际使用的编译环境。

5.1 串行C/C++程序的编译

- Intel C/C++编译器：
  - 编译C和C++程序的编译命令分别为*icc*和*icpc*
  - 主要编译格式为：*icc/icpc␣[options]␣file1␣[file2␣...]*
  - 编译举例：
    * 将C程序yourprog.c编译为可执行文件yourprog(-o)：
      icc␣-o␣yourprog␣yourprog.c
    * 将C++程序yourprog.cpp编译为可执行文件yourprog(-o)：
      icpc␣-o␣yourprog␣yourprog.cpp
    * 将C程序yourprog.c编译为对象文件yourprog.o(-c)而不是可执行文件：
      icc␣-c␣yourprog.c
    * 将C程序yourprog.c编译为汇编文件yourprog.s(-S)而不是可执行文件：
      icc␣-S␣yourprog.c
    * 生成带有调试信息的可执行文件以用于调试(-g)：
      icc␣-g␣yourprog.c␣-o␣yourprog

* 指定头文件路径(-I)编译：
  icc -I/alt/include -o yourprog yourprog.c
* 指定库文件路径(-L)及库名(-l)编译：
  icc -L/alt/lib -lxyz -o yourprog yourprog.c

- PGI C/C++编译器：
  - 编译C和C++程序的编译命令分别为*pgcc*和*pgCC*
  - 主要编译格式为：*pgcc/pgCC [-flag]... sourcefile...*
  - 编译举例：
    * 将C程序yourprog.c编译为可执行文件yourprog：
      pgcc -o yourprog yourprog.c
    * 将C++程序yourprog.cpp编译为可执行文件yourprog：
      pgCC -o yourprog yourprog.cpp
    * 将C程序yourprog.c编译为对象文件yourprog.o(-c)而不是可执行文件：
      pgcc -c yourprog.c
    * 将C程序yourprog.c编译为汇编文件yourprog.s(-S)而不是可执行文件：
      pgcc -S yourprog.c
    * 生成带有调试信息的可执行文件以用于调试(-g)：
      pgcc -g yourprog.c -o yourprog
    * 指定头文件路径(-I)编译：
      pgcc -I/alt/include -o yourprog yourprog.c
    * 指定库文件路径(-L)及库名(-l)编译：
      pgcc -L/alt/lib -lxyz -o yourprog yourprog.c

- GNU C/C++编译器：
  - 编译C和C++程序的编译命令分别为*gcc*和*g++*
  - 主要编译格式为：*gcc/g++ [options]... sourcefile...*
  - 编译举例：
    * 将C程序yourprog.c编译为可执行文件yourprog:*gcc -o yourprog yourprog.c*
    * 将C++程序yourprog.cpp编译为可执行文件yourprog：
      g++ -o yourprog yourprog.cpp
    * 将C程序yourprog.c编译为对象文件yourprog.o(-c)而不是可执行文件：
      gcc -c yourprog.c
    * 将C程序yourprog.c编译为汇编文件yourprog.s(-S)而不是可执行文件：
      gcc -S yourprog.c

* 生成带有调试信息的可执行文件以用于调试(-g):
  gcc␣-g␣yourprog.c␣-o␣yourprog
* 指定头文件路径(-I)编译:
  gcc␣-I/alt/include␣-o␣yourprog␣yourprog.c
* 指定库文件路径(-L)及库名(-l)编译:
  gcc␣-L/alt/lib␣-lxyz␣-o␣yourprog␣yourprog.c

5.2 串行Fortran程序的编译

- Intel Fortran编译器:
  - 编译Fortran程序的编译命令为*ifort*
  - 主要编译格式为: *ifort␣[options]␣file1␣[file2␣...]*
  - 编译举例:
    * 将Fortran 77程序yourprog.for编译为可执行文件yourprog(-o):
      ifort␣-o␣yourprog␣yourprog.for
    * 将Fortran 90程序yourprog.f90编译为可执行文件yourprog(-o):
      ifort␣-o␣yourprog␣yourprog.f90
    * 将Fortran 90程序yourprog.90编译为对象文件yourprog.o(-c)而不是可执行文件:
      ifort␣-c␣yourprog.f90
    * 将Fortran程序yourprog.f90编译为汇编文件yourprog.s(-S)而不是可执行文件:
      ifort␣-S␣yourprog.f90
    * 生成带有调试信息的可执行文件以用于调试(-g):
      ifort␣-g␣yourprog.f90␣-o␣yourprog
    * 指定头文件路径(-I)编译:
      ifort␣-I/alt/include␣-o␣yourprog␣yourprog.f90
    * 指定库文件路径(-L)及库名(-l)编译:
      ifort␣-L/alt/lib␣-lxyz␣-o␣yourprog␣yourprog.f90
- PGI Fortran编译器:
  - 编译Fortran 77和90程序的编译命令分别为*pgf77*和*pgf90*
  - 主要编译格式为: *pgf77/pgf90␣[␣-flag␣]...␣sourcefile...*
  - 编译举例:
    * 将Fortran 77程序yourprog.for编译为可执行文件yourprog:
      pgf77␣-o␣yourprog␣yourprog.for

* 将Fortran 90程序yourprog.f90编译为可执行文件yourprog：
  pgf90␣-o␣yourprog␣yourprog.f90
* 将Fortran程序yourprog.f90编译为对象文件yourprog.o(-c)而不是可执行文件：
  pgf90␣-c␣yourprog.f90
* 将Fortran程序yourprog.f90编译为汇编文件yourprog.s(-S)而不是可执行文件：
  pgf90␣-S␣yourprog.f90
* 生成带有调试信息的可执行文件以用于调试(-g)：
  pgf90␣-g␣yourprog.f90␣-o␣yourprog
* 指定头文件路径(-I)编译：
  pgf90␣-I/alt/include␣-o␣yourprog␣yourprog.f90
* 指定库文件路径(-L)及库名(-l)编译：
  pgf90␣-L/alt/lib␣-lxyz␣-o␣yourprog␣yourprog.f90

- GNU Fortran编译器：
  - 编译Fortran程序的编译命令为*g77*或*gfortran*
  - 主要编译格式为：*g77/gfortran␣[␣options␣]...␣sourcefile...*
  - 编译举例：
    * 将Fortran 77程序yourprog.for编译为可执行文件yourprog：
      · gcc 4.x系列：*gfortran␣-o␣yourprog␣yourprog.for*
      · gcc 3.x系列：*g77␣-o␣yourprog␣yourprog.for*
    * 将Fortran 90程序yourprog.f90编译为可执行文件yourprog：
      gfortran␣-o␣yourprog␣yourprog.f90
    * 将Fortran程序yourprog.f90编译为对象文件yourprog.o(-c)而不是可执行文件：
      gfortran␣-c␣yourprog.f90
    * 将Fortran程序yourprog.f90编译为汇编文件yourprog.s(-S)而不是可执行文件：
      gfortran␣-S␣yourprog.f90
    * 生成带有调试信息的可执行文件以用于调试(-g)：
      gfortran␣-g␣yourprog.f90␣-o␣yourprog
    * 指定头文件路径(-I)编译：
      gfortran␣-I/alt/include␣-o␣yourprog␣yourprog.f90
    * 指定库文件路径(-L)及库名(-l)编译：
      gfortran␣-L/alt/lib␣-lxyz␣-o␣yourprog␣yourprog.f90

注意：*g77*既不支持OpenMP，也不支持Fortran 90及之后的标准。

5.3　OpenMP程序的编译与运行

5.4　OpenMP程序的编译

Intel、PGI和GNU编译器都支持OpenMP并行，只需利用相关编译命令结合必要的OpenMP编译选项编译即可。对应此三种编译器的OpenMP编译选项：

- Intel编译器：
  - -openmp：2015之前版的选项，在2015版也支持，但属于过时选项
  - -qopenmp：2015及之后版的选项，建议使用
- PGI编译器：-mp
- GNU编译器：-fopenmp

采用这三种编译器的编译例子如下：

- Intel编译器：
  - 2015版
    * 将OpenMP的C程序yourprog-omp.c编译为可执行文件yourprog-omp：
      icc -qopenmp -o yourprog-omp yourprog.c
    * 将OpenMP的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp：
      ifort -qopenmp -o yourprog-omp yourprog.f90
  - 2015之前版
    * 将OpenMP的C程序yourprog-omp.c编译为可执行文件yourprog-omp：
      icc -openmp -o yourprog-omp yourprog.c
    * 将OpenMP的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp：
      ifort -openmp -o yourprog-omp yourprog.f90
- PGI编译器：
  - 将OpenMP的C程序yourprog-omp.c编译为可执行文件yourprog-omp：
    pgcc -mp -o yourprog-omp yourprog.c
  - 将OpenMP的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp：
    pgf90 -mp -o yourprog-omp yourprog.f90
- GNU编译器：
  - 将OpenMP的C程序yourprog-omp.c编译为可执行文件yourprog-omp：
    gcc -fopenmp -o yourprog-omp yourprog.c
  - 将OpenMP的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp：
    gfortran -fopenmp -o yourprog-omp yourprog.f90

5.5 OpenMP程序的运行

OpenMP程序的运行一般是通过在运行前设置环境变量*OMP_NUM_THREADS*来控制线程数，比如在bash中利用*export OMP_NUM_THREADS=24*设置使用24个线程运行。

注意，本系统为节点内共享内存节点间分布式内存的架构，因此只能在一个节点上的CPU之间运行同一个OpenMP程序作业，在提交作业时需要使用相应选项以保证在同一个节点运行。

6 MPI并行程序编译及运行

本系统安装有两种MPI实现：Intel MPI和Open MPI，并可与不同编译器相互配合使用，安装目录分别在*/opt/intel/impi*和*/opt/openmpi/*[5]。系统默认设置使用Intel编译器与Intel MPI的组合，安装目录为*/opt/intel/impi/5.0.2*。

用户可以运行*module␣apropos␣MPI*或*module␣avail*查看可用MPI环境，用类似命令设置所需的MPI环境：*module␣load␣intelmpi/5.0.2.044*，使用此命令时会自动加载对应的编译器>等版本。

6.1 MPI并行程序的编译

6.2 Intel MPI库

安装的Intel MPI库[6]5.0版本库实现了MPI V3.0标准。

6.2.1 编译命令

请注意，Intel MPI与Open MPI等MPI实现不同，*mpicc*、*mpif90*和*mpifc*命令默认使用GNU编译器，如需指定使用Intel编译器等，请使用对应的*mpiicc*、*mpiicpc*和*mpiifort*命令。下表为Intel MPI编译命令及其对应关系。

6.2.2 编译举例

对于MPI并行程序，对应不同类型源文件的编译命令如下：

- 调用默认C编译器将C语言的MPI并行程序yourprog-mpi.c编译为可执行文件yourprog-mpi：
  mpicc␣-o␣yourprog-mpi␣yourprog-mpi.c
- 调用Intel C编译器将C语言的MPI并行程序yourprog-mpi.c编译为可执行文件yourprog-mpi：
  mpiicc␣-o␣yourprog-mpi␣yourprog-mpi.cpp
- 调用Intel C++编译器将C++语言的MPI并行程序yourprog-mpi.cpp编译为可执行文件yourprog-mpi：
  mpiicxx␣-o␣yourprog-mpi␣yourprog-mpi.cpp

[5]具有不同版本的Open MPI与编译器的组合。
[6]主页：http://software.intel.com/en-us/intel-mpi-library/

表 1: Intel MPI编译命令及其对应关系

编译命令	调用的默认编译器命令	支持的语言	支持的应用二进制接口
通用编译器			
mpicc	gcc, cc	C	32/64 bit
mpicxx	g++	C/C++	32/64 bit
mpifc	gfortran	Fortran77*/Fortran 95*	32/64 bit
GNU* Compilers Versions 3 and Higher			
mpigcc	gcc	C	32/64 bit
mpigxx	g++	C/C++	32/64 bit
mpif77	g77	Fortran 77	32/64 bit
mpif90	gfortran	Fortran 95	32/64 bit
Intel Fortran, C++ Compilers Versions 11.1 and Higher			
mpiicc	icc	C	32/64 bit
mpiicpc	icpc	C++	32/64 bit
mpiifort	ifort	Fortran77/Fortran 95	32/64 bit

- 调用GNU Forttan编译器将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.f
- 调用Intel Fortran编译器将Fortran 90语言的MPI并行程序yourprog-mpi.f90编译为可执行文件yourprog-mpi：
  mpiifort␣-o␣yourprog-mpi␣yourprog-mpi.f90

6.3 Open MPI库

Open MPI[7]库是另一种非常优秀MPI实现，用户如需使用可以自己通过运行*module␣load␣模块名*选择与openmpi相关的项自己设置即可。

Open MPI的安装目录在*/opt/openmpi*，的编译命令主要为：

- C程序：*mpicc*
- C++程序：*mpic++*、*mpicxx*、*mpiCC*

[7]主页：http://www.open-mpi.org/

- Fortran 77程序：*mpif77*、*mpif90*、*mpifort*
- Fortran 90程序：*mpif90*
- Fortran程序：*mpifort*[8]

*mpifort*为1.8系列引入的编译Fortran程序的命令，1.6系列不支持此命令。

*mpif77*和*mpif90*为1.6系列和1.8系列的编译Fortran程序的命令，但在1.8系列中已经为过时的命令。

对于MPI并行程序，对应不同类型源文件的编译命令如下：

- 将C语言的MPI并行程序yourprog-mpi.c编译为可执行文件yourprog-mpi：
  mpicc -o yourprog-mpi yourprog-mpi.c
- 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*：
  mpicxx -o yourprog-mpi yourprog-mpi.cpp
- 将Fortran 90语言的MPI并行程序yourprog-mpi.f90编译为可执行文件yourprog-mpi：
  mpifort -o yourprog-mpi yourprog-mpi.f90
- 将Fortran 77语言的MPI并行程序yourprog-mpi.f编译为可执行文件yourprog-mpi：
  mpif77 -o yourprog-mpi yourprog-mpi.f
- 将Fortran 90语言的MPI并行程序yourprog-mpi.f90编译为可执行文件yourprog-mpi：
  mpif90 -o yourprog-mpi yourprog-mpi.f90

编译命令的基本语法为：*编译命令 [-showme|-showme:compile|-showme:link] ...*

6.4 与编译器相关的编译选项

MPI编译环境的编译命令实际上是调用Intel、PGI或GCC编译器进行编译，具体选项等，请参看Intel MPI、Open MPI以及Intel、PGI和GCC编译器手册。

6.5 MPI并行程序的运行

MPI程序最常见的并行方式类似为：*mpirun -n 24 yourmpi-prog*。

在本超算系统上，MPI并行程序需结合IBM Platform LSF作业调度系统的作业提交命令*bsub*来调用作业脚本运行，基本格式为*bsub -q normal -n 24 mpijob executable*，请参看作业调度管理系统。

[8]注意为mpifort，而不是Intel MPI的mpiifort

7 Intel MKL数值函数库

当前安装的Intel MKL版本为Intel Parallel Studio XE 2015版本编译器自带的11.2版本(安装在*/opt/intel/composer_xe_2015.1.133/mkl*)。在bash下可以通过运行*module␣load␣mkl/版本号*设置，或者在*~/.bashrc*之类的环境变量设置文件中添加类似下面代码设置Intel MKL所需的环境变量*INCLUDE*、*LD_LIBRARY_PATH*和*MANPATH*等：

```
../opt/intel/composer_xe_2015.1.133/bin/compilervars.sh␣intel64
```

7.1 Intel MKL主要内容

Intel MKL主要包含如下内容：

- 基本线性代数子系统库（BLAS, level 1, 2, 3）和线性代数库（LAPACK）：提供向量、向量-矩阵、矩阵-矩阵操作。
- ScaLAPACK分布式线性代数库：含基础线性代数通信子程序（Basic Linear Algebra Communications Subprograms, BLACS）和并行基础线性代数子程序（Parallel Basic Linear Algebra Subprograms, PBLAS）
- PARDISO直接离散算子：一种迭代离散算子，支持用于求解方程的离散系统的离散BLAS (level 1, 2, and 3)子函数，并提供可用于集群系统的分布式版本的PARDISO。
- 快速傅立叶变换方程（Fast Fourier transform, FFT）：支持1、2或3维，支持混合基数（不局限与2的次方），并有分布式版本。
- 向量数学库（Vector Math Library, VML）：提供针对向量优化的数学操作。
- 向量统计库（Vector Statistical Library, VSL）：提供高性能的向量化随机数生成算子，可用于一些几率分布、剪辑和相关例程和汇总统计功能。
- 数据拟合库（Data Fitting Library）：提供基于样条函数逼近、函数的导数和积分，及搜索。
- 扩展本征解算子（Extended Eigensolver）：基于FEAST的本征值解算子的共享内存版本的本征解算子。

7.2 Intel MKL目录内容

Intel MKL的主要目录内容见表2。

表 2: Intel MKL目录内容

目录	内容
<mkl_dir>	MKL主目录，如*/opt/intel/composer_xe_2013.2.146/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>/bin	包含设置MKL环境变量的脚本
<mkl_dir>/bin/ia32	包含针对IA-32架构设置MKL环境变量的脚本
<mkl_dir>/bin/intel64	包含针对Intel 64架构设置MKL环境变量的脚本
<mkl_dir>/examples	一些例子，可以参考学习
<mkl_dir>/include	含有INCLUDE文件
<mkl_dir>/include/ia32	含有针对ia32 Intel编译器的Fortran 95 .mod文件
<mkl_dir>/include/intel64/ilp64	含有针对Intel64 Intel编译器ILP64接口[9]的Fortran 95 .mod文件
<mkl_dir>/include/intel64/lp64	含有针对Intel64 Intel编译器LP64接口的Fortran 95 .mod文件
<mkl_dir>/include/mic/ilp64	含有针对MIC架构ILP64接口的Fortran 95 .mod文件，本系统未配置M
<mkl_dir>/include/mic/lp64	含有针对MIC架构LP64接口的Fortran 95 .mod文件，本系统未配置M
<mkl_dir>/include/fftw	含有FFTW2和3的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/fftw2x_cdft	包含2.x版集群FFTW(MPI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/fftw3x_cdft	包含3.x版集群FFTW(MPI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/ia32	包含IA32架构的静态库和共享目标文件
<mkl_dir>/lib/intel64	包含EM64T架构的静态库和共享目标文件
<mkl_dir>/lib/mic	用于MIC协处理器，本系统未配置MIC
<mkl_dir>/tests	一些测试文件
<mkl_dir>/tools	工具及插件
<mkl_dir>/tools/builder	包含用于生成定制动态可链接库的工具
<mkl_dir>/../Documentation/en_US/mkl	MKL文档目录

7.3 链接Intel MKL

7.4 快速入门

7.4.1 利用-mkl编译器参数

Intel Composer XE编译器支持采用-mkl[10]参数链接Intel MKL：

- -mkl或-mkl=parallel：采用标准线程Intel MKL库链接；
- -mkl=sequential：采用串行Intel MKL库链接；
- -mkl=cluster：采用Intel MPI和串行MKL库链接；
- 对Intel 64架构的系统，默认使用LP64接口链接程序。

7.4.2 使用链接行顾问

Intel提供了网页和命令行方式的链接行顾问帮助用户设置所需要的MKL链接参数。访问http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor或执行*mkl_link_tool -interactive*，按照提示输入所需要信息即可获取链接Intel MKL时所需要的参数。

7.5 链接举例

7.5.1 在Intel 64架构上链接

在这些例子中：

- MKLPATH=$MKLROOT/lib/intel64
- MKLINCLUDE=$MKLROOT/include

如果已经设置好环境变量，那么在所有例子中可以略去-I$MKLINCLUDE，在所有动态链接的例子中可以略去-L$MKLPATH。

- 使用LP64接口的并行Intel MKL库静态链接myprog.f：

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE
-Wl,--start-group $MKLPATH/libmkl_intel_lp64.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
```

[10]是-mkl，不是-lmkl，其它编译器未必支持此-mkl选项。

- 使用LP64接口的并行Intel MKL库动态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE
-lmkl_intel_lp64␣-lmkl_intel_thread␣-lmkl_core␣-liomp5␣-lpthread␣-lm
```

- 使用LP64接口的串行Intel MKL库静态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE
-Wl,--start-group␣$MKLPATH/libmkl_intel_lp64.a␣$MKLPATH/libmkl_sequential.a
$MKLPATH/libmkl_core.a␣-Wl,--end-group␣-lpthread␣-lm
```

- 使用LP64接口的串行Intel MKL库动态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE
-lmkl_intel_lp64␣-lmkl_sequential␣-lmkl_core␣-lpthread␣-lm
```

- 使用ILP64接口的并行Intel MKL库动态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE
-Wl,--start-group␣$MKLPATH/libmkl_intel_ilp64.a␣$MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a␣-Wl,--end-group␣-liomp5␣-lpthread␣-lm
```

- 使用ILP64接口的并行Intel MKL库动态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE
-lmkl_intel_ilp64␣-lmkl_intel_thread␣-lmkl_core␣-liomp5␣-lpthread␣-lm
```

- 使用串行或并行（调用函数或设置环境变量选择线程或串行模式，并设置接口）Intel MKL库动态链接myprog.f:

```
ifort␣myprog.f␣-lmkl_rt
```

- 使用Fortran 95 LAPACK接口和LP64接口的并行Intel MKL库静态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE␣-I$MKLINCLUDE/intel64/lp64
-lmkl_lapack95_lp64␣-Wl,--start-group␣$MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_intel_thread.a␣$MKLPATH/libmkl_core.a
-Wl,--end-group␣-liomp5␣-lpthread␣-lm
```

- 使用Fortran 95 BLAS接口和LP64接口的并行Intel MKL库静态链接myprog.f:

```
ifort␣myprog.f␣-L$MKLPATH␣-I$MKLINCLUDE␣-I$MKLINCLUDE/intel64/lp64
-lmkl_blas95_lp64␣-Wl,--start-group␣$MKLPATH/libmkl_intel_lp64.a
$MKLPATH/libmkl_intel_thread.a␣$MKLPATH/libmkl_core.a
-Wl,--end-group␣-liomp5␣-lpthread␣-lm
```

7.6 链接细节

下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$MKLPATH/libmkl_core.a代替-lmkl_core，其中$MKLPATH为用户定义的指向MKL库目录的环境变量。

```
<files␣to␣link>

-L<MKL␣path>␣-I<MKL␣include>
[-I<MKL␣include>/{ia32|intel64|{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_core
-liomp5␣[-lpthread]␣[-lm]␣[-ldl]
```

注：*[]*内的表示可选，|表示其中之一、*{}*表示含有。在静态链接时，在分组符号（如，*-Wl,--start-group␣$MKLPATH/libmkl_cdft_core.a␣$MKLPATH/libmkl_blacs_intelmpi_ilp64.a␣$MKLPATH/libmkl_intel_ilp64.a␣$MKLPATH/libmkl_intel_thread.a␣$MKLPATH/libmkl_core.a␣-Wl,--end-group*）封装集群组件、接口、线程和计算库。

列出库的顺序是有要求的，除非是封装在上面分组符号中的。

8 作业调度管理系统

本系统利用IBM Platform LSF Express 8.3进行资源和作业调度管理，所有需要运行的作业均必须通过作业提交命令*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(tc4600)上不通过作业调度管理系统直接运行作业（编译等日常操作除外），以免影响其余用户的正常使用。
- 如果不通过作业调度管理系统直接在计算节点上运行将会被监护进程直接杀掉。

8.1 作业运行的条件

作业提交后需要一段时间等待作业调度系统调度运行，一般为先提交的先运行，并且作业运行需要满足多个基本条件：

- 系统有空闲资源，满足程序运行需要。可以利用*bhosts*命令查看，ok状态的才可以接受作业运行。
- 用户作业没有超过系统设置的允许用户运行的作业数，可以利用*busers*命令查看。
- 用户作业没有超过所使用的作业队列的允许作业核数的限制，可以利用*bqueues*命令查看。
- 用户作业没有被挂起等。利用*bjobs*命令查看。
- 作业调度管理系统工作正常。

当系统作业繁忙时，如果提交需要核数很多的作业，也许需要长时间才可以运行甚至根本无法获取到足够资源来运行，请考虑选择合适的并行规模。

作业如不运行，请先请运行*bjobs -l JobID*或*bjobs -p JobID*查看输出信息中PENDING REASONS部分及提交时设置的参数等，并结合运行上述几个命令查看原因。

如果上述条件都符合，也许作业调度管理系统存在问题，请与超算中心工作人员联系处理。

8.2 常见命令

- 查看队列情况：bqueues
- 提交作业：bsub
- 终止作业：bkill
- 查看作业的排队和运行情况：bjobs
- 查看运行中作业的屏幕正常输出：bpeek
- 查看各节点的运行情况：lsload
- 查看各节点的空闲情况：bhosts
- 查看用户信息：busers
- 挂起作业：bstop
- 继续运行被挂起的作业：bresume
- 设置作业最先运行：btop
- 设置作业最后运行：bbot
- 修改排队中的作业选项：bmod

8.3 查看队列情况：bqueues

利用*bqueues*可以查看现有队列信息[11]，例如：

bqueues

将输出：

```
QUEUE_NAME PRIO STATUS    MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
QUEUE_NAME     PRIO STATUS          MAX JL/U JL/P JL/H NJOBS PEND RUN SUSP
serial          50  Open:Active       -   16    -    -     0    0    0    0
long            40  Open:Active       -    -    -    -     0    0    0    0
normal          30  Close:Active      -    -    -    -     0    0    0    0
```

其中，主要列的含义为：

- QUEUE_NAME：队列名

[11]以下仅为举例，具体队列会根据需要更改，请看登录后的提示，或运行bqueues -l查看

- 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

8.4 提交作业：bsub

用户需要利用*bsub*提交作业，其基本格式为*bsub␣[options]␣command␣[arguments]*。其中options设置队列、CPU核数等的选项，必需在command之前，否则将作为command的参数；arguments为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。下面将给出常用的几种提交方式。

8.4.1 提交到特定队列：bsub -q

利用-q选项可以指定提交到哪个队列，作业队列会针对运行情况进行修改，请注意参看登录后的提示或运行*bqueues␣-l*命令查看，现有的队列为：

- normal：所需要的CPU核数大于1个且不超过16个时
- long：所需要的CPU核数超过32个时
- serial：所需要的CPU核数为一个时

比如想提交到normal队列使用2个CPU核运行程序executable1，可以：

bsub␣-q␣normal␣-n␣2␣executable1

如果提交成功，将显示类似下面的输出：

```
Job <79722> is submitted to 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。

8.4.2 运行串行作业：bsub -q serial

运行串行作业，请使用串行队列serial，比如：

bsub␣-q␣serial␣executable-serial

本超算系统鼓励运行并行作业，允许运行的串行作业资源较少。

8.4.3 指明所需要的CPU核数：bsub -n

利用-n选项指定所需要的CPU核数（一般来说核数和进程数一致），比如下面指定利用16个CPU核（由-n 16指定）运行MPI（由*mpijob*指明为运行MPI程序）程序：

bsub␣-q␣normal␣-n␣16␣mpijob␣executable-mpi1

如需要的核数多于16个，需利用-q long指明使用long队列。

由于每个节点的CPU核数为16，建议单个作业所使用的核数最好为16或8的整数倍，以尽量保证自己的程序占据独立的节点或半个节点，尽量避免相互影响。

8.4.4 运行MPI作业：bsub -n NUM mpijob

如果需要运行MPI作业，需要利用*mpijob*调用MPI可执行程序，并用-n选项指定所需的CPU核数，比如下面指定利用64颗CPU核运行MPI程序executable-mpi1：

bsub␣-q␣long␣-n␣64␣mpijob␣executable-mpi1

注意：

- *mpijob*命令已经封装了Intel MPI和Open MPI的运行MPI作业的脚本，用户一般无需再特别按照Intel MPI和Open MPI的原始*mpirun*、*mpiexec*等命令使用。
- 提交作业时在*mpijob*之前的参数将传递给LSF,之后的参数将传递给原始的*mpirun*或*mpiexec*等。
- 如用户对LSF和Intel MPI、Open MPI等不熟悉，请勿擅自添加其它参数，如有需要请与超算中心联系。

8.4.5 运行OpenMP共享内存作业：bsub -q

由于只能在同一个节点内部运行OpenMP共享内存的作业，此时需要添加利用-q参数指定队列（normal队列会自动保证只在单个节点内运行，long队列跨节点运行，要除外）选项：

bsub␣-q␣normal␣-n␣16␣executable-omp1

8.4.6 运行MPI和OpenMP共享内存混合并行作业

需要针对需求利用LSF环境变量特殊处理，如果自己不清楚怎么处理，请联系管理人员。

8.4.7 运行排他性作业：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␣8␣executable-omp1

注意：排他性作业在运行期间，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。

8.4.8 指明输出、输出文件运行：bsub -i -o -e

作业的正常屏幕输入文件（指的是类似 命令 < 输入文件 方式的文件）、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。比如指定executable1的输入、正常和错误屏幕输出文件分别为：*executable1.input*、*executable-作业号.log*和*executable1-作业号.err*：

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

建议打开-o和-e参数，以便查看作业为什么出问题等。如果需要管理人员协助解决，请告知这些输出，以及运行目录，怎么运行的等，以便管理人员能获取足够的信息及时处理。

8.4.9 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或利用调试器手动调试程序等需要进行交互时），需要结合-I参数。建议只是在调试期间使用，一般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

8.5 LSF作业脚本

如果作业比较复杂，还需要设置环境变量，做其它处理等，可用以在LSF脚本中设置队列等参数方式提交，如*my_script.lsf*

```
#!/bin/sh
#BSUB -q long
#BSUB -o %J.log -e %J.err
#BSUB -n 64
source my.sh
mpijob ./mympi-prog1
cd newworkdir
mpijob ./mympi-prog2
```

注意，采用此方式时：

- 不得以直接*./my_script.lsf*等常规脚本运行方式运行。
- 需要传递给*bsub*命令运行：*bsub < my_script.lsf*。
- 如果bsub后面更-q等LSF参数，将会覆盖掉LSF脚本中的设置。
- 一般用户，没必要写此类脚本，直接通过命令行传递LSF参数即可。
- 对于当前设置满足不了作业需求，且用户比较了解LSF中的各规定，对shell脚本编写比较在行，那么用户完全可自己编写脚本提交作业，比如提交特殊需求的MPI与GPU结合的作业。

LSF作业脚本主要有以下常见变量比较常用，在作业运行后，这些变量存储对应的作业信息，具体的请参看LSF官方手册：

- *LS_JOBPID*：作业进程号
- *LSB_HOSTS*：存储系统分配的节点名
- *LSB_JOBFILENAME*：作业脚本文件名
- *LSB_JOBID*：作业号
- *LSB_QUEUE*：作业队列
- *LSB_JOBPGIDS*：作业进程组号组
- *LSB_JOBPIDS*：作业进程号组

8.6 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
```

8.7 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.
```

此命令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，因此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。

8.8 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.
```

8.9 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.
```

8.10 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.
```

8.11 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long队列，可以：

bmod␣-Z␣executable2␣-q␣long␣79727

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

8.12 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，比如有哪些作业在运行，哪些在排队，某个作业运行在哪个节点上，以及为什么没有运行等，例如：

bjobs

```
JOBID USER STAT QUEUE   FROM_HOST  EXEC_HOST JOB_NAME SUBMIT_TIME
79726 hmli RUN  normal  tc4600  2*node31 *executab1 Mar 12 19:20
                                 1*node18
                                 1*node4
79727 hmli PEND long    tc4600           *executab2 Mar 12 19:20
```

上面显示作业79726在运行，分别在node31、node18和node4上运行2、1、1个进程；而作业79727处于排队中尚未运行，查看未运行的原因可以利用：

bjobs␣-l␣79727

```
Job Id <79727>, tc4600 <hmli>, Project <default>, Status <PEND>,
Queue <long> , Command <executab2>
Sun Mar 12 14:15:07: Submitted from host <tc4600>,
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   -    -    -    -     -     -
```

以下为另外几个常用参数：

- -u username：查看某用户的作业，如username为all，则查看所有用户的作业。
- -q queuename：查看某队列上的作业。
- -m hostname：查看某节点上的作业。

8.13 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 >>
Energy: 3.0keV
Angles: 13.0, 0.0
```

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。

如果想连续查看某个作业的输出，请添加-f参数。

8.14 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
node1        ok      0.0   0.0   0.0   0%   0.5   0   25  227G   32G   62G
node2        locku   0.1   0.0   0.0   0%   0.4   0   60  227G   32G   62G
node4        unavail  -     -     -    -     -    -   -     -     -     -
```

- HOST_NAME：节点名。
- status：状态。
  - ok：正常状态。
  - busy：超负载运行。

  - lockW：被运行窗口锁定。
  - lockU：有作业在进行排他性运行。
  - unavaild：节点有问题，需要联系超算中心老师处理。
  - unlicensed：节点有问题，需要联系超算中心老师处理。
- r15s：近15秒平均系统负载。
- r1m：近1分钟系统负载。
- r15m：近15分钟系统负载。
- pg：上一分钟的内存分页比率，单位为每页/秒。
- tmp：*/tmp*目录空闲大小，单位为MB。
- swp：可用虚拟内存大小，单位为MB。
- mem：可用内存大小，单位为MB。

8.15 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  -    16      2   2     0     0   0
node10       ok      -    16      2   1     0     0   0
node14       ok      -    16      2   1     0     0   0
```

- HOST_NAME：节点名。
- STATUS：状态。
  - ok：可以接收新作业。
  - closed：已经被占满或被超算中心老师关闭，无法接受新作业。
  - unavail：节点出错，需要超算中心老师处理。
  - unreach：节点出错，需要超算中心老师处理。
  - closed_Cu_excl：运行排他性计算单元作业的成员节点。
- JL/U：允许每个用户的作业核数。-表示未限制。

- MAX：允许最大作业核数。
- NJOBS：当前运行的作业数。
- RUN：当前运行作业占据的核数。
- SSUSP：被系统挂起的作业占据的核数。
- USUSP：被用户挂起的作业占据的核数。
- RSV：预留的核数。

即使有节点状态为ok状态，也不一定表示您的作业可以运行，具体运行条件参见8.1作业运行条件。

8.16 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          -    320    40    32   8     0     0   0
```

其中：

- USER/GROUP：用户或组名。
- JL/U：允许每个用户的作业核数。-表示未限制。
- MAX：允许最大作业核数。
- NJOBS：当前运行的作业数。
- PEND：当前挂起作业占据的核数。
- RUN：当前运行作业占据的核数。
- SSUSP：被系统挂起的作业占据的核数。
- USUSP：被用户挂起的作业占据的核数。
- RSV：预留的核数。

9 联系方式

- 超级计算中心：
  - 电话：0551-63602248
  - 信箱：sccadmin@ustc.edu.cn
  - 主页：http://scc.ustc.edu.cn
  - 办公室：中国科大东区新图书馆一楼东侧超级计算中心126室
- 李会民：
  - 电话：0551-63600316
  - 信箱：hmli@ustc.edu.cn
  - 主页：http://hmli.ustc.edu.cn
  - 办公室：中国科大东区新科研楼网络信息中心204室