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

李会民

2016年9月14日

目录

Part I

前言

本用户使用指南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。

Part II

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颗CPU核心，总双精度峰值计算能力为每秒288万亿次。具体参数为：

- 用户登录节点：
  - 节点名为tc4600
  - 用于用户登录、编译与通过作业调度系统提交管理作业等。禁止在此节点上不通过作业调度系统直接运行作业。
  - 曙光I620-G10机架式服务器
    * 两颗64位主频2.4GHz的Intel Xeon E5-2620 v3 x86_64 6核CPU，共12核
    * 32GB DDR4 2133MHz ECC内存
    * 两块600GB SAS硬盘
- 管理节点：
  - 节点名为tcadmin
  - 用于系统管理，普通用户无权登录
  - 曙光I620-G10机架式服务器
    * 两颗64位主频2.4GHz的Intel Xeon E5-2620 v3 x86_64 6核CPU，共12核
    * 32GB DDR4 2133MHz ECC内存
    * 两块600GB SAS硬盘
- 刀片计算节点：
  - 300台曙光CX50-G20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

[1]高级矢量扩展，理论性能：整数和浮点性能翻倍

    * 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
      · 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，一台216口和108口Mellonax 56Gbps交换机各一台
- 管理网络：千兆以太网
- 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

Part III

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）。

[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

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。

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，请与超算中心老师联系。

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。

Part IV

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*module load intelmpi/5.1.3.210*。
- *rm|unload modulefile...*:卸载已加载的环境modulefile,如*module 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*。

Part V

串行及OpenMP程序编译及运行

在本超算系统上可运行C/C++、Fortran的串行程序，以及与OpenMP和MPI结合的并行程序。编译程序时，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（用户不应到其余节点上进行编译，以免影响系统效率。其它节点一般只设置了运行作业所需要的库路径等，未必设置了编译环境）。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。

本超算系统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。

建议采用对一般程序来说性能较好的Intel编译器，用户也可以选择适合自己程序的编译器，以取得更好的性能。

本部分主要介绍串行C/C++ Fortran源程序和OpenMP并行程序的编译，MPI并行程序的编译将在后面介绍。

1 串行C/C++程序的编译

1.1 输入输出文件后缀与类型的关系

编译器默认将按照输入文件的后缀判断文件类型，输入文件的后缀与类型的关系见表1。

编译器默认将输出按照文件类型与后缀相对应，输出文件的后缀与类型的关系见表2。

表 1: 输入文件后缀与类型的关系

文件名	解释	动作
filename.c	C源文件	传给编译器
filename.C filename.CC filename.cc filename.cpp filename.cxx	C++源文件	传给编译器
filename.a filename.so	库文件	传递给链接器
filename.i	已预处理的文件	传递给标准输出
filename.o	目标文件	传递给链接器
filename.s	汇编文件	传递给汇编器

表 2: 输出文件后缀与文件类型的关系

文件名	解释
filename.i	已预处理的文件，由使用-p选项生成
filename.o	目标文件，由添加-c选项生成
filename.s	汇编文件，由添加-s选项生成
a.out	默认生成的可执行文件

1.2 串行C/C++程序编译举例

- Intel C/C++编译器：
  - 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：
  - 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：

- 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

2 串行Fortran程序的编译

2.1 输入输出文件后缀与类型的关系

编译器默认将按照输入文件的后缀判断文件类型，输入文件的后缀与类型的关系见表3。

编译器默认将输出按照文件类型与后缀相对应，输出文件的后缀与类型的关系见表4。

表 3: 输入文件后缀与文件类型的关系

文件名	解释	动作
filename.a	目标库文件	传给编译器
filename.f filename.for filename.ftn filename.i	固定格式的Fortran源文件	被Fortran编译器编译
filename.fpp filename.FPP filename.F filename.FOR filename.FTN	固定格式的Fortran源文件	自动被Fortran编译器预处理后再被编译
filename.f90 filename.i90	自由格式的Fortran源文件	被Fortran编译器编译
filename.F90	自由格式的Fortran源文件	自动被Fortran编译器预处理后再被编译
filename.s	汇编文件	传递给汇编器
filename.so	库文件	传递给链接器
filename.o	目标文件	传递给链接器

表 4: 输出文件后缀与类型的关系

文件名	解释	生成方式
filename.o	目标文件	编译时添加-c选项生成
filename.so	共享库文件	编译时指定为共享型，如添加-shared，并不含-c
filename.mod	模块文件	编译含有MODULE声明时的源文件生成
filename.s	汇编文件	编译时添加-S选项生成
a.out	默认生成的可执行文件	编译时没有指定-c时生成

2.2 串行Fortran程序编译举例

- Intel Fortran编译器：
  - 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：
  - 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：
  - 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。

3 OpenMP程序的编译与运行

3.1 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

3.2 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。

Part VI

Intel、PGI及GNU C/C++ Fortran编译器介绍

4 Intel C/C++ Fortran编译器

4.1 Intel C/C++ Fortran编译器简介

Intel Parallel Studio XE 2015版C/C++ Fortran编译器，是一种主要针对Inetl平台的高性能编译器，可用于开发复杂且要进行大量计算的C/C++、Fortran程序。

系统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*。官方手册目录为*/opt/intel/composer_xe_2015.1.133/Documentation*。

Intel编译器编译C和C++源程序的编译命令分别为*icc*和*icpc*；编译Fortran源程序的命令为*ifort*。*icpc*命令使用与*icc*命令相同的编译器选项，利用*icpc*编译时将后缀为.c和.i的文件看作为C++文件；而利用*icc*编译时将后缀为.c和.i的文件则看作为C文件。用*icpc*编译时，总会链接C++库；而用*icc*编译时，只有在编译命令行中包含C++源文件时才链接C++库。

编译命令格式为:*command [options] [@response_file] file1 [file2...]*,其中*response_file*为文件名，此文件包含一些编译选项，请注意调用时前面有个@。

在Intel数学库(Intel®math)中的许多函数针对Intel微处理器相比针对非Intel微处理器做了非常大的优化处理。

为了使用Intel数学库中的函数，需要在程序源文件中包含头文件*mathimf.h*，例如使用实函数:

```
// real_math.c
#include <stdio.h>
#include <mathimf.h>

int main() {
 float fp32bits;
 double fp64bits;
 long double fp80bits;
 long double pi_by_four = 3.14159265358979323 8/4.0;
```

```
// pi/4 radians is about 45 degrees
 fp32bits = (float) pi_by_four; // float approximation to pi/4
 fp64bits = (double) pi_by_four; // double approximation to pi/4
 fp80bits = pi_by_four; // long double (extended) approximation to pi/4

// The sin(pi/4) is known to be 1/sqrt(2) or approximately .7071067
 printf("When␣x␣=␣%8.8f,␣sinf(x)␣=␣%8.8f␣\n", fp32bits, sinf(fp32bits));
 printf("When␣x␣=␣%16.16f,␣sin(x)␣=␣%16.16f␣\n", fp64bits, sin(fp64bits));
 printf("When␣x␣=␣%20.20Lf,␣sinl(x)␣=␣%20.20f␣\n", fp80bits, sinl(fp80bits));

 return 0;
}
```

编译：*icc real_math.c*

4.2 编译错误

C/C++程序编译时的出错信息类似以下：

```
netlog.c(140): error: identifier "hhh" is undefined
                    for(int hhh=domain_cnt+1;hhh>TMP;hhh--){
                                             ^

netlog.c(156): error: expected an expression
    for(int i=0;i<32;i++)for(int j=0;j<256;j++)if(ip1[i][j]!=0)fprintf(fin);
        ^
```

Fortran程序编译时的出错信息类似以下：

```
NOlihm.f90(146): error #6404: This name does not have a type, and must have
an explicit type.  [NPR]
  n2nd=0; npr=0
-----------^
NOlihm.f90(542): remark #8290: Recommended relationship between field width
'W' and the number of fractional digits 'D' in this edit descriptor is 'W>=D+3'.
6060 format(/i2,'-th layer',i2,'-th element: z=',i3,' a=',f9.5/' Ef=',f7.5)
-------------------------------------------------------------------------^
```

编译错误的格式为：

- 源文件名(行数): 错误类型:具体说明

- 源代码，^指示出错位置

错误类型可以为：

- Warning：警告，报告对编译有效但也许存在问题的语法，请根据信息及程序本身判断，不一定需要处理。
- Error：存在语法或语义问题，必须要处理。
- Fatal Error：报告环境错误，如磁盘空间没有了。

4.3 Fortran程序运行错误

根据运行时错误代码可以在官方手册中查找对应错误解释：

- 逐步链接：
  User and Reference Guide for the Intel®Fortran Compiler 15.0->Compiler Reference->Error Handling->Handling Run-Time Errors->List of Run-Time Error Messages
- 直接链接：
  http://scc.ustc.edu.cn/zlsc/tc4600/intel/2015.1.133/compiler_f/GUID-44448B78-2B8htm

4.4 Intel Parallel Studio XE 2015版重要编译选项

Intel编译器选项分为几类，可以用*icc -help 类别*查看对应的选项，类别与选项对应关系如下：

advanced - Advanced Optimizations，高级优化

codegen - Code Generation，代码生成

compatibility - Compatibility，兼容性

component - Component Control，组件控制

data - Data，数据

deprecated - Deprecated Options，过时选项

diagnostics - Compiler Diagnostics，编译器诊断

float - Floating Point，浮点

help - Help，帮助

inline - Inlining，内联

ipo - Interprocedural Optimization (IPO)，过程间优化

language - Language，语言

link - Linking/Linker，链接/链接器

misc - Miscellaneous，杂项

opt - Optimization，优化

output - Output，输出

pgo - Profile Guided Optimization (PGO)，概要导向优化

preproc - Preprocessor，预处理

reports - Optimization Reports，优化报告

openmp - OpenMP and Parallel Processing，OpenMP和并行处理

可以运行*icc -help help*查看选项分类情况。

4.4.1 Intel Parallel Studio XE 2015版新增选项

表 5: Intel Parallel Studio XE 2015版过时选项

过时的选项	描述	默认
-check-pointers-narrowing[-]	是否编译器启用限制指针到结构域。	-[no-]check-pointers-narrowing
-diag-enable=power	是否启用诊断针对可能影响能耗的低效代码的诊断。	-diag-disable=power
-f[no-]eliminate-unused-debug-types	是否对在编译单元中的类型声明显示调试信息。	-feliminate-unused-debug-types
-f[no-]fat-lto-objects	是否在过程间优化（-c -ipo）时生成包含中间语言及对象代码的胖链接时优化对象。	-ffat-lto-objects

-[no-]gcc-include-dir	是否将gcc特性的头文件目录放在系统头文件路径中。	-gcc-include-dir
-prof-gen=[no]threadsafe	是否生成含有PGO数据的检测对象文件用于更高级的并行化。	-prof-gen=nothreadsafe
-[no-]qopt-dynamic-align	是否启用动态数据对齐优化。	-opt-dynamic-align
-qopt-report-embed	是否在生成时将特定循环信息保存在汇编文件中。	OFF
-qopt-report-filter	让编译器寻找含有特定子集信息的部分，并针对此部分生成优化报告。	OFF
-qopt-report-format	设定优化报告的格式。	OFF
-qopt-report-names	设定是否在优化报告中显示重整或未重整的名字。	OFF
-qopt-report-per-object	让编译器针对每个对象文件生成单独的优化报告文件。	OFF

4.4.2 Intel Parallel Studio XE 2015版已过时选项

表 6: Intel Parallel Studio XE 2015版已过时选项

过时的选项	替代选项
-axS	-axSSE4.1
-axT	-axSSSE3
-check-uninit	-check=uninit
-diag-enable=sc	None
-diag-enable=sc-enums	None
-diag-enable=sc-include	None

-diag-enable=sc-single-file	None
-diag-sc-dir	None
-fms-dialect=9	None
-fmudflap	None; 请考虑使用指针检查器选项（如 check）
-Kc++	-x c++
-march=pentiumii	None
-march=pentiumiii	-march=pentium3
-mcpu	-mtune
-msse	-mia32
-offload	-qoffload
-offload-attribute-target	-qoffload-attribute-target
-offload-option	-qoffload-option
-openmp	-qopenmp
-openmp-lib	-qopenmp-lib
-qopenmp-link、-openmp-link	None
-openmp-offload	-qopenmp-offload
-openmpP	-qopenmp
-openmp-report	-qopenmp-report
-openmpS	-qopenmp-stubs
-openmp-simd	-qopenmp-simd
-openmp-stubs	-qopenmp-stubs
-openmp-task	-qopenmp-task
-openmp-threadprivate	-qopenmp-threadprivate
-opt-args-in-regs	-qopt-args-in-regs

-opt-assume-safe-padding	-qopt-assume-safe-padding
-opt-block-factor	-qopt-block-factor
-opt-calloc	-qopt-calloc
-opt-class-analysis	-qopt-class-analysis
-opt-dynamic-align	-qopt-dynamic-align
-opt-gather-scatter-unroll	-qopt-gather-scatter-unroll
-opt-jump-tables	-qopt-jump-tables
-opt-malloc-options	-qopt-malloc-options
-opt-matmul	-qopt-matmul
-opt-mem-layout-trans	-qopt-mem-layout-trans
-opt-multi-version-aggressive	-qopt-multi-version-aggressive
-opt-prefetch	-qopt-prefetch
-opt-prefetch-distance	-qopt-prefetch-distance
-opt-ra-region-strategy	-qopt-ra-region-strategy
-opt-report	-qopt-report
-opt-report-embed	-qopt-report-embed
-opt-report-file	-qopt-report-file
-opt-report-filter	-qopt-report-filter
-opt-report-format	-qopt-report-format
-opt-report-help	-qopt-report-help
-opt-report-per-object	-qopt-report-per-object
-opt-report-phase	-qopt-report-phase
-opt-report-routine	-qopt-report-routine
-opt-streaming-cache-evict	-qopt-streaming-cache-evict

-opt-streaming-stores	-qopt-streaming-stores
-opt-subscript-in-range	-qopt-subscript-in-range
-opt-threads-per-core	-qopt-threads-per-core
-use-asm	None
-wd	-diag-disable
-we	-diag-error
-wn	-diag-error-limit
-wo	-diag-once id[,id,...]
-wr	-diag-remark
-ww	-diag-warning
-xH	-xSSE4.2
-xS	-xSSE4.1
-xSSE3_ATOM	-xATOM_SSSE3
-xSSSE3_ATOM	-xATOM_SSSE3
-xT	-xSSSE3

4.4.3 Intel Parallel Studio XE 2015版编译器已废弃选项

表 7: Intel Parallel Studio XE 2015版编译器已废弃选项

废弃的选项	替代选项
-A-	-undef
-0f_check	None
-alias-args	-fargument-alias
-axB	-axSSE2
-axH	-axSSE4.2

-axi	None
-axK 无精确替代者；升级到	-msse2
-axM	None
-axN	-axSSE2
-axP	-axSSE3
-axW	-msse2
-c99	-std=c99
-create-pch	-pch-create
-cxxlib-gcc[=dir]	-cxxlib[=dir]
-cxxlib-icc	None
-diag-enable sv[<n>]	-diag-enable sc[<n>]
-diag-enable sv-include	-diag-enable sc-include
-diag-sv	-diag-enable sc[<n>]
-diag-sv-error	-diag-disable warning
-diag-sv-include	-diag-enable sc-include
-diag-sv-level	None
-diag-sv-sup	-diag-disable <v>[,<v2,...]
-export	None
-export-dir	None
-F	-P
-fdiv_check	None
-fp	-fno-omit-frame-pointer
-fpstkchk	-fp-stack-check
-func-groups	-prof-func-groups

-fvisibility=internal	-fvisibility=hidden
-fwritable-strings	None
-gcc-version No exact replacement; use	-gcc-name
-i-dynamic	-shared-intel
-i-static	-static-intel
-inline-debug-info	-debug inline-debug-info
-ipo-obj (and -ipo_obj)	None
-ipp-link=static-thread	None
-Knopic,-KNOPIC	-fpic
-Kpic,-KPIC	-fpic
-mp	-fp-model
-no-alias-args	-fargument-noalias
-no-c99	-std=c89
-no-cpprt	-no-cxxlib
-nobss-init	-no-bss-init
-norestrict	-no-restrict
-Ob	-inline-level
-openmp-lib legacy	None
-openmp-profile	None
-opt-report-level	-qopt-report
-prefetch	-qopt-prefetch
-prof-format-32	None
-prof-gen-sampling	None
-prof-genx	-prof-gen=srcpos

-qp	-p
-rct	None
-shared-libcxa	-shared-libgcc
-ssp	None
-static-libcxa	-static-libgcc
-syntax	-fsyntax-only
-tpp1	None
-tpp2	-mtune=itanium2
-tpp5	None
-tpp6	None
-tpp7	-mtune=pentium4
-tprofile	None
-use-pch	-pch-use
-Wpragma-once	None
-xB	-xSSE2
-xi	None
-xK 无精确替代者；升级到	-msse2
-xM	None
-xN	-xSSE2
-xO	-msse3
-xP	-xSSE3
-xW	-msse2

4.4.4 优化选项

- -fast：最大化整个程序的速度。这里是所谓的最大化，还是需要结合程序本身使用合适的选项，默认不使用此选项。

- -nolib-inline：取消标准库和内在函数的内联展开。
- -O*n*：设定优化级别，默认为O2。O与O2相同，推荐使用；O3为在O2基础之上增加更激进的优化，比如包含循环和内存读取转换和预取等，但在有些情况下速度反而慢，建议在具有大量浮点计算和大数据处理的循环时的程序使用。
- -Ofast：设定一定的优化选项提高程序性能，设定-O3, -no-prec-div和-fp-model fast=2。在Linux系统上提供与gcc的兼容。
- -Os：启用优化，但不增加代码大小，并且产生比-O2优化小的代码。它取消了一些优化不明显却增大了代码的优化选项。

4.4.5 代码生成选项

- -ax*code*：在有性能提高时，生成针对Intel®处理器的多特征面向的自动调度代码路径。*code*可为：
  - MIC-AVX512：生成针对Intel®处理器的Intel®Advanced Vector Extensions 512 (Intel®AVX-512)基础指令，Intel®AVX-512冲突检测指令，Intel®AVX-512指数和倒数指令，Intel®AVX-512预处理指令，以及CORE-AVX2启用的指令。
  - CORE-AVX2:生成Intel®Advanced Vector Extensions 2 (Intel®AVX2)、Intel®AVX、SSE4.2、SSE4.1、SSE3、SSE2、SSE和SSSE3指令。本系统使用的Intel Xeon E5 2680v3 CPU支持此指令集，建议使用。
  - CORE-AVX-I:生成Float-16转换指令和RDRND(随机数)指令、Intel®Advanced Vector Extensions (Intel®AVX)、Intel®SSE4.2、SSE4.1、SSE3、SSE2、SSE和SSSE3指令。
  - AVX:生成Intel®Advanced Vector Extensions (Intel®AVX)、Intel®SSE4.2、SSE4.1、SSE3、SSE2、SSE和SSSE3指令。
  - SSE4.2：生成Intel®SSE4.2、SSE4.1、SSE3、SSE2、SSE和SSSE3指令。
  - SSE4.1：生成Intel®SSE4.1、SSE3、SSE2、SSE和SSSE3指令
  - SSSE3：生成SSSE3指令和Intel®SSE3、SSE2和SSE指令。
  - SSE3：生成Intel®SSE3、SSE2和SSE指令。
  - SSE2：生成Intel®SSE2和SSE指令。
- -fexceptions、-fno-exceptions：是否生成异常处理表。
- -x*code*：设置启用编译目标的特征，包含采取何种指令集和优化。
  - MIC-AVX512

  - CORE-AVX2（建议）
  - CORE-AVX-I
  - AVX
  - SSE4.2
  - SSE4.1
  - SSSE3
  - SSE3
  - SSE2

- -m*code*：需要生成目标特征的指令集。*code*可为：
  - avx:生成Intel®Advanced Vector Extensions (Intel®AVX)、Intel®SSE4.2、SSE4.1、SSE3、SSE2、SSE和SSSE3指令。
  - sse4.2：生成Intel®SSE4.2、SSE4.1、SSE3、SSE2、SSE和SSSE3指令。
  - sse4.1：生成Intel®SSE4.1、SSE3、SSE2、SSE和SSSE3指令。
  - ssse3：生成SSSE3指令和Intel®SSE3、SSE2和SSE指令。
  - sse3：生成Intel®SSE3、SSE2和SSE指令。
  - sse2：生成Intel®SSE2和SSE指令。
  - sse：已过时，现在与ia32一样。
  - ia32：生成与IA-32架构兼容的x86/x87通用代码。取消任何默认扩展指令集，任何之前的扩展指令集。并且取消所有面向特征的优化及指令。此值仅在Linux系统上使用IA-32架构时有效。
- -m32和-m64：生成IA-32或Intel®64位代码，默认由主机系统设定。
- -march=*processor*：生成支持某种处理器特定特征的代码。*processor*可为：
  - generic
  - core-avx2（建议）
  - core-avx-i
  - corei7-avx
  - corei7
  - atom
  - core2
  - pentium-mmx

  - pentiumpro
  - pentium4m
  - pentium-m
  - pentium4（默认）
  - pentium3
  - pentium

- -mtune=*processor*：针对特定处理器优化。*processor*可为：
  - generic（默认）
  - core-avx2（建议）
  - core-avx-i
  - corei7-avx
  - corei7
  - atom
  - core2
  - pentium-mmx
  - pentiumpro
  - pentium4m
  - pentium-m
  - pentium4
  - pentium3
  - pentium

- -xHost：生成编译主机处理器能支持的最高指令集。建议使用。

4.4.6 过程间优化（IPO）选项

- -ip：在单个文件中进行过程间优化(Interprocedural Optimizations-IPO)。
- -ip-no-inlining：禁止过程间优化时启用的全部和部分内联。
- -ip-no-pinlining：禁止过程间优化时启用的部分内联。
- -ipo*[n]*、-no-ipo：是否在多文件中进行过程间优化，非负整数*n*为可生成的对象文件数。

- -ipo-c：在多文件中进行过程间优化，并生成一个对象文件。
- -ipo-jobs*n*：指定在过程间优化的链接阶段时的命令（作业）数。
- -ipo-S：在多文件中进行过程间优化，并生成一个汇编文件。
- -ipo-separate：在多文件中进行过程间优化，并为每个文件分别生成一个对象文件。

4.4.7 高级优化选项

- -funroll-all-loops：即使在循环次数不确定的情况下也展开所有循环。默认为否。
- -guide*[=n]*：设置自动向量化、自动并行及数据变换的指导级别。*n*为1到4，1为标准指导，4为最高指导，如果*n*忽略，则默认为4。默认为不启用。
- -guide-data-trans*[=n]*：设置数据变换时的指导级别。*n*为1到4，1为标准指导，4为最高指导，如果*n*忽略，则默认为4。默认为不启用。
- -guide-file*[=filename]*：将自动并行的结果输出到文件*filename*中。
- -guide-file-append*[=filename]*：将自动并行的结果追加到文件*filename*中。
- -guide-par*[=n]*：设置自动并行的指导级别。*n*为1到4，1为标准指导，4为最高指导，如果*n*忽略，则默认为4。默认为不启用。
- -guide-vec*[=n]*：设置自动向量化的指导级别。*n*为1到4，1为标准指导，4为最高指导，如果*n*忽略，则默认为4。默认为不启用。
- -mkl*[lib]*：链接时自动链接Intel MKL库，默认为不启用。*lib*可以为：
  - parallel：采用线程化部分的MKL库链接，此为*lib*如果没指明时的默认选项。
  - sequential：采用未线程化的串行MKL库链接。
  - cluster：采用集群部分和串行部分MKL链接。
- -simd、-no-simd：是否启用SIMD编译指示的编译器解释。
- -unroll*[=n]*：设置循环展开的最大层级。
- -unroll-aggressive、-no-unroll-aggressive：设置对某些循环执行激进展开。默认不启用。
- -vec、-no-vec：是否启用向量化。默认启用。

4.4.8 概要导向优化（PGO）选项

- -p：使用gprof编译和链接函数。
- -prof-dir *dir*：设定存储概要导向优化信息的文件目录。
- -prof-file *filename*：设定概要摘要文件名。

4.4.9 优化报告选项

- -qopt-report*[=n]*：设定显示优化报告信息的级别，为每个对象文件生成一个对应的文件。*n*为0（不显示）到5（最详细）。
- -qopqopt-report-file=*keyword*：设定报告文件名。*keyword*可以为：
  - filename：保存输出的文件名。
  - stderr：输出到标准错误输出。
  - stdout：输出到标准输出。
- -qopt-report-filter=*string*:设置报告的过滤器。*string*可以为filename、routine、range等。
- -qopt-report-format=*keyword*：设置报告的格式。*keyword*可以为text和vs，分别对应纯文本和Visual Studio格式。
- -qopt-report-help：显示使用-qopt-report-phase选项时可用于报告生成的各优化阶段，并显示各级别报告的简短描述。
- -qopt-report-per-object：为各对象文件生成独立的报告文件。
- -qopt-report-phase：对生成的优化报告指明一个或多个优化阶段。*phase*可以为：cg、ipo、loop、openmp、par、pgo、tcollect、vec和all等。
- -qopt-report-routine=*substring*：让编译器对含有*substring*的子程序生成优化报告。
- -qopt-report-names=*keyword*：是否在优化报告中显示重整的或未重整的名字。*keyword*可以为：mangled和unmangled。
- -tcheck：对线程应用启用分析。
- -tcollect*[lib]*：插入测试探测调用Intel Trace Collector API。*lib*为一种Intel Trace Collector库，例如：VT、VTcs、VTmc或VTfs。
- -tcollect-filter*filename*：对特定的函数启用或禁止测试。
- -vec-report*[=n]*：设置向量化诊断信息详细程度。*n*为0（不显示）到7（最详细）。

4.4.10 OpenMP和并行处理选项

- -fmpc-privatize、-fno-mpc-privatize：是否启用针对多处理器计算环境（MPC）所有静态数据私有。
- -par-affinity=*[modifier,...]type[,permute][,offset]*：设定线程亲和性。
  - modifier:可以为以下值之一:granularity=fine|thread|core、[no]respect、[no]verbose、[no]warnings、proclist=proc_list。默认为granularity=core, respect, noverbose。
  - type：指示线程亲和性。此选项是必需的，并且需为以下之一：compact、disabled、explicit、none、scatter、logical、physical。默认为none。logical和physical已经过时。分别使用compact和scatter，并且没有permute值。
  - permute：非负整数。当type设置为explicit、none或disabled时，不能使用此选项。默认为0。
  - offset：非负整数。当type设置为explicit、none或disabled时，不能使用此选项。默认为0。
- -par-num-threads=*n*：设定并行区域内的线程数。
- -par-report*n*：设定自动并行时诊断信息的显示级别。*n*可以为0到5。
- -par-runtime-control*n*、-no-par-runtime-control：设定是否对符号循环边界的循环执行运行时检查代码。
- -par-schedule-*keyword[=n]*：设定循环迭代的调度算法。*keyword*可以为：
  - auto：由编译器或者运行时系统设定调度算法。
  - static：将迭代分割成连续块。
  - static-balanced：将迭代分割成偶数大小的块。
  - static-steal：将迭代分割成偶数大小的块，但允许线程从临近线程窃取部分块。
  - dynamic：动态获取迭代集。
  - guided：设定迭代的最小值。
  - guided-analytical：使用指数分布或动态分布分割迭代。
  - runtime：直到运行时才设定调度分割。

  *n*为每个迭代数或块大小。此设置，只能配合static、dynamic和guided使用。
- -par-threshold*n*：设定针对循环自动并行的阈值。*n*为一个在0到100间的整数，限定针对循环自动并行的阈值：

  - 如*n*为0，则循环总会被并行。
  - 如*n*为100，则循环只有在基于编译器分析应用的数据能达到预期收益时才并行。
  - 1到99为预期可能的循环加速百分比。

- -parallel：让自动并行器针对可以安全并行执行的循环生成多线程代码。
- -parallel-source-info=*n*、-no-parallel-source-info：设定当生成OpenMP或自动并行代码是否显示源位置。*n*为显示级别：
  - 0：禁止显示源位置信息。
  - 1：显示子程序名和行信息。
  - 2：显示路径、文件名、子程序名和行信息。
- -openmp和-qopenmp：编译OpenMP程序。注意：在一般只能在同一个节点内的CPU上运行OpenMP程序。
  - -openmp：2015之前版本的Intel编译器编译OpenMP的选项，在2015版本可以使用，但已过时；
  - -qopenmp：2015及之后版本的Intel编译器编译OpenMP的选项，之前的版本不支持此选项；
  - 很多-openmp-*及-opt-*选项都需要替换成对应的-qopenmp-*和-qopt-*选项。
- -qopenmp-lib=*type*：设定链接时使用的OpenMP运行时库。当前*type*只能设定为compat。
- -qopenmp-link=*library*：设定采用动态还是静态链接OpenMP运行时库。*library*可以为static和dynamic，分别表示静态和动态链接OpenMP运行时库。
- -qopenmp-report*n*：设定OpenMP并行器的诊断信息的显示级别。*n*可以为0、1和2。
- -qopenmp-simd、-no-qopenmp-simd：设定是否启用OpenMP SIMD编译。
- -qopenmp-stubs：使用串行模式编译OpenMP程序。
- -qopenmp-task=*model*：设定OpenMP的任务模型。*model*可以为：
  - intel：让编译接受Intel任务序列指导指令（#pragma intel_omp_taskq和#pragma intel_omp_task）。OpenMP API 3.0将被忽略。
  - omp：让编译接受OpenMP API 3.0任务序列指导指令（#pragma omp_task）。Intel任务序列指导指令将被忽略。
- -qopenmp-threadprivate=*type*：设定OpenMP线程私有的实现。*type*可以为：

  - legacy：让编译器继承使用以前Intel编译器使用的OpenMP线程私有实现。
  - compat：让编译器使用基于对每个私有线程变量应用__declspec(thread)属性的兼容OpenMP线程私有实现。

4.4.11 浮点选项

- -fast-transcendentals：让编译器使用超越函数代替，超越函数是较快但精度较低的实现。
- -fimf-absolute-error=*value[:funclist]*：定义对于数学函数返回值允许的最大绝对误差的值。*value*为正浮点数,*funclist*为函数名列表。如:-fimf-absolute-error=0.00001:sin,sinf。
- -fimf-accuracy-bits=*bits[:funclist]*：定义数学函数返回值的相对误差，包含除法及开方。*bits*为正浮点数，指明编译器应该使用的正确位数，*funclist*为函数名列表。如：-fimf-accuracy-bits=23:sin,sinf。bits与ulps之间的变换关系为：$ulps = 2^{p-1-bits}$，其中p为目标格式尾数$bits$的位数（对应单精度、双精度和长双精度分别为23、53和64）。
- -fimf-max-error=*ulps[:funclist]*：定义对于数学函数返回值的最大允许相对误差，包含除法及开方。*value*为正浮点数，指定编译器可以使用的最大相对误差，*funclist*为函数名列表，如：-fimf-max-error=4.0:sin,sinf。
- -fimf-precision*[=value[:funclist]]*：当设定使用何种数学库函数时，定义编译器应该使用的精度。*value*可以为：
  - high：等价于max-error = 0.6
  - medium：等价于max-error = 4
  - low：等价于accuracy-bits = 11（对单精度）和accuracy-bits = 26（对双精度）

  *funclist*为函数名列表，如：-fimf-precision=high:sin,sinf。
- -fma、-no-fma：是否对存在融合乘加（fused multiply-add-FMA）的目标处理器启用融合乘加。此选项只有在-x或-march参数设定CORE-AVX2或更高时才有效。
- -fp-model *keyword*：控制浮点计算的语义，*keyword*可以为：
  - precise：取消浮点数据的非值安全优化。
  - fast[=1|2]：对浮点数据启用更加激进的优化。
  - strict：启用精度和异常，禁止收缩，启用编译指示stdc和fenv_access。
  - source：四舍五入中间结果到源定义精度。
  - double：四舍五入中间结果到53-bit（双）精度。

  - extended：四舍五入中间结果到64-bit（扩展）精度。
  - [no-]except：定义严格浮点异常编译指令是否启用。

  *keyword*可以分成以下三组使用：

  - precise, fast, strict
  - source, double, extended
  - except

- -fp-port、-no-fp-port：是否对浮点操作启用四舍五入。
- -fp-speculation=*mode*：设定推测浮点操作时使用的模式。*mode*可以为：
  - fast：让编译器推测浮点操作。
  - safe：让编译器在推测浮点操作有可能存在浮点异常时停止推测。
  - strict：让编译器禁止浮点操作时推测。
  - off：与strict相同。
- -fp-trap=*mode[,mode,...]*：设置主函数的浮点异常捕获模式。*mode*可以为：
  - [no]divzero：是否启用被0除时的IEEE捕获。
  - [no]inexact：是否启用不精确结果时的IEEE捕获。
  - [no]invalid：是否启用无效操作时的IEEE捕获。
  - [no]overflow：是否启用上溢时的IEEE捕获。
  - [no]underflow：是否启用下溢时的IEEE捕获。
  - [no]denormal：是否启用非正规时的IEEE捕获。
  - all：启用上述所有的IEEE捕获。
  - none：禁止启用上述所有的IEEE捕获。
  - common：启用最常见的IEEE捕获：被0除、无效操作和上溢。
- -fp-trap-all=*mode[,mode,...]*：设置所有函数的浮点异常捕获模式。*mode*可以为：
  - [no]divzero：是否启用被0除时的IEEE捕获。
  - [no]inexact：是否启用不精确结果时的IEEE捕获。
  - [no]invalid：是否启用无效操作时的IEEE捕获。
  - [no]overflow：是否启用上溢时的IEEE捕获。
  - [no]underflow：是否启用下溢时的IEEE捕获。

  - [no]denormal：是否启用非正规时的IEEE捕获。
  - all：启用上述所有的IEEE捕获。
  - none：禁止启用上述所有的IEEE捕获。
  - common：启用最常见的IEEE捕获：被0除、无效操作和上溢。
- -ftz：赋值非常规操作结果为0。
- -mp1：提高浮点操作的精度和一致性。
- -pc*n*：设定浮点尾数精度。*n*可以为：
  - 32：四舍五入尾数到24位（单精度）。
  - 64：四舍五入尾数到53位（双精度）。
  - 80：四舍五入尾数到64位（扩展精度）。
- -prec-div、-no-prec-div：是否提高浮点除的精度。
- -prec-sqrt、-no-prec-sqrt：是否提高开根的精度。
- -rcd：启用快速浮点数到整数转换。

4.4.12 内联选项

- -gnu89-inline：设定编译器在C99模式时使用C89定义处理内联函数。
- -finline、-fno-inline：是否对__inline声明的函数进行内联，并执行C++内联。
- -finline-functions、-fno-inline-functions：对单个文件编译时启用函数内联。
- -finline-limit=*n*：设定内联函数的最大数。*n*为非负整数。
- -inline-calloc、-no-inline-calloc:是否设定编译器内联调用calloc()为调用malloc()和memset()。
- -inline-factor、-no-inline-factor：是否设定适用于所有内联选项定义的上限的比例乘法器。
- -inline-level=*n*：设定内联函数的展开级别。*n*可以为0、1、2.

4.4.13 输出、调试及预编译头文件（PCH）选项

- -c：仅编译成对象文件（.o文件）。
- -debug *[keyword]*：设定是否生成调试信息。*keyword*可以为：
  - none：不生成调试信息。
  - full或all：生成完全调试信息。
  - minimal：生成最少调试信息。
  - [no]emit_column：设定是否针对调试生成列号信息。
  - [no]expr-source-pos：设定是否在表达式粒度级别生成源位置信息。
  - [no]inline-debug-info：设定是否针对内联代码生成增强调试信息。
  - [no]macros：设定是否针对C/C++宏生成调试信息。
  - [no]pubnames：设定是否生成DWARF debug_pubnames节。
  - [no]semantic-stepping：设定是否编译器生成针对断点和单步的增强调试信息。
  - [no]variable-locations：设定是否编译器生成有助于寻找标量局部变量的增强型调试信息。
  - extended：设定关键字值semantic-stepping和variable-locations。
  - [no]parallel：设定是否编译器生成并行调试代码指令以有助于线程数据共享和可重入调用探测。
- -g：包含调试信息。
- -g0：禁止生成符号调试信息。
- -gdwarf-*n*：设定生成调试信息时的DWARF版本，*n*可以为2、3、4。
- -o file：指定生成的文件名。
- -pch：设定编译器使用适当的预编译头文件。
- -pch-create *filename*：设定生成预编译头文件。
- -pch-dir *dir*：设定搜索预编译头文件的目录。
- -pch-use *filename*：设定使用的预编译头文件。
- -print-multi-lib：打印哪里系统库文件应该被发现。
- -S：设定编译器只是生成汇编文件但并不进行链接。

4.4.14 预处理选项

- -B*dir*：设定头文件、库文件及可执行文件的搜索路径。
- -D*name[=value]*：设定编译时的宏及其值。
- -dD：输出预处理的源文件中的#define指令。
- -dM：输出预处理后的宏定义。
- -dN：与-dD类似，但只输出的#define指令的宏名。
- -E：设定预处理时输出到标注输出。
- -EP：设定预处理时输出到标注输出，忽略#line指令。
- -gcc、-no-gcc、-gcc-sys：判定确定的GNU宏（__GNUC__、__GNUC_MINOR__和__GNUC_PATCHLEVEL__）是否定义。
- -gcc-include-dir、-no-gcc-include-dir：设定是否将gcc设定的头文件路径加入到头文件路径中。
- -H：编译时显示头文件顺序并继续编译。
- -I：设定头文件附加搜索路径。
- -icc、-no-icc：设定Intel宏（__INTEL_COMPILER）是否定义。
- -idirafter*dir*：设定*dir*路径到第二个头文件搜索路径中。
- -imacros *filename*：允许一个头文件在编译时在其它头文件前面。
- -iprefix *prefix*：指定包含头文件的参考目录的前缀。
- -iquote *dir*：在搜索的头文件路径前面增加*dir*目录以供那些使用引号而不是尖括号的文件使用。
- -isystem*dir*：附加*dir*目录到系统头文件的开始。
- -iwithprefix*dir*：附加*dir*目录到通过-iprefix引入的前缀后，并将其放在头文件目录末尾的头文件搜索路径中。
- -iwithprefixbeforex*dir*:除头文件目录*dir*放置的位置与-I声明的一样外,与-iwithprefix类似。
- -M：让编译器针对各源文件生成makefile依赖行。
- -MD：预处理和编译，生成后缀为.d包含依赖关系的输出文件。

- -MF*filename*：让编译器在一个文件中生成makefile依赖信息。
- -MG：让编译器针对各源文件生成makefile依赖行。与-M类似，但将缺失的头文件作为生成的文件。
- -MM：让编译器针对各源文件生成makefile依赖行。与-M类似，但不包含系统头文件。
- -MMD：预处理和编译，生成后缀为.d包含依赖关系的输出文件。与-M类似，但不包含系统头文件。
- -MP：让编译器对每个依赖生成伪目标。
- -MQ*target*：对依赖生成改变默认目标规则。*target*是要使用的目标规则。与-MT类似，但引用特定Make字符。
- -MT*target*：对依赖生成改变默认目标规则。*target*是要使用的目标规则。
- -nostdinc++：对C++不搜索标准目录下的头文件，而搜索其它标准目录。
- -P：停止编译处理，并将结果写入文件。
- -pragma-optimization-level=*interpretation*：指定如没有前缀指定时，采用何种优化级别编译指令解释。*interpretation*可以为：
  - Intel：Intel解释。
  - GCC：GCC解释。
- -U*name*：取消某个宏的预定义。
- -undef：取消所有宏的预定义。
- -X：从搜索路径中去除标准搜索路径。

4.4.15 C/C++语言选项

- -ansi：与gcc的-ansi选项兼容。
- -check=*keyword[, keyword...]*：设定在运行时检查某些条件。*keyword*可以为：
  - [no]conversions：设定是否在转换成较小类型时进行检查。
  - [no]stack：设定是否在堆栈帧检查。
  - [no]uninit：设定是否对未初始化变量进行检查。
- -fno-gnu-keywords：让编译器不将typeof作为一个关键字。

- -fpermissive：让编译器允许非一致性代码。
- -fsyntax-only：让编译器仅作语法检查，不生成目标代码。
- -funsigned-char：将默认字符类型变为无符号类型。
- -help-pragma：显示所有支持的编译指令。
- -intel-extensions、-no-intel-extensions：是否启用Intel C和C++语言扩展。
- -restrict、-no-restrict：设定是否采用约束限定进行指针消岐。
- -std=*val*：*val*可以为c89、c99、gnu89、gnu++98或c++0x，分别对应相应标准。
- -stdlib*[=keyword]*：设定链接时使用的C++库。*keyword*可以为：
  - libstdc++：链接使用GNU libstdc++库。
  - libc++：链接使用libc++库。
- -strict-ansi：让编译器采用严格的ANSI一致性语法。
- -x *type*：*type*可以为c、c++、c-header、cpp-output、c++-cpp-output、assembler、assembler-with-cpp或none，分别表示c源文件等，以使所有源文件都被认为是此类型的。
- -Zp*[n]*: 设定结构体在字节边界的对齐。n是字节大小边界，可以为1、2、4、8和16。

4.4.16 Fortran语言选项

- -auto-scalar：INTEGER、REAL、COMPLEX和LOGICAL内在类型变量，如未声明有SAVE属性，将分配到运行时堆栈中，下次调用此函数时变量赋值。
- -allow *keyword*：设定编译器是否允许某些行为。*keyword*可以为[no]fpp_comments，声明fpp预处理器如何处理在预处理指令行中的Fortran行尾注释。
- -altparam、-noaltparam：设定是否允许不同的语法（不带括号）PARAMETER声明。
- -assume *keyword[, keyword...]*：设定某些假设。*keyword*可以为：none、[no]bscc、[no]buffered_io、[no]buffered_stdout、[no]byterecl、[no]cc_omp、[no]dummy_aliases、[no]fpe_summary、[no]ieee_fpe_flags、[no]minus0、[no]old_boz、[no]old_ldout_format、[no]old_logical_ldio、[no]old_maxminloc、[no]old_unit_star、[no]old_xor、[no]protect_constants、[no]protect_parens、[no]realloc_lhs、[no]source_include、[no]std_intent_in、[no]std_minus0_rounding、[no]std_mod_proc_name、[no]std_value、[no]underscore、[no]2underscores、[no]writeable-strings等

- -ccdefault *keyword*：设置文件显示在终端上时的回车类型。*keyword*可以为：
  - none：设定编译器使用无回车控制预处理。
  - default：设定编译器使用默认回车控制设定。
  - fortran：设定编译器使用通常的第一个字符的Fortran解释。如字符0使得在输出一个记录时先输出一个空行。
  - list：设定编译器记录之间出输出换行。
- -check=*keyword[, keyword...]*：设定在运行时检查某些条件。*keyword*可以为：
  - none：禁止所有检查。
  - [no]arg_temp_created：设定是否在子函数调用前检查实参。
  - [no]assume：设定是否在测试在ASSUME指令中的标量布尔表达式为真，或在ASSUME_ALIGNED指令中的地址对齐声明的类型边界时进行检查。
  - [no]bounds：设定是否对数组下标和字符子字符串表达式进行检查。
  - [no]format：设定是否对格式化输出的数据类型进行检查。
  - [no]output_conversion：设定是否对在指定的格式描述域内的数据拟合进行检查。
  - [no]pointers：设定是否对存在一些分离的或未初始化的指针或为分配的可分配目标时进行检查。
  - [no]stack：设定是否在堆栈帧检查。
  - [no]uninit：设定是否对未初始化变量进行检查。
  - all：启用所有检查。
- -cpp：对源代码进行预处理，等价于-fpp。
- -extend-source*[size]*：指明固定格式的Fortran源代码宽度，*size*可为72、80和132。也可直接用-72、-80和-132指定，默认为72字符。
- -fixed：指明Fortran源代码为固定格式，默认由文件后缀设定格式类别。
- -free：指明Fortran源程序为自由格式，默认由文件后缀设定格式类别。
- -nofree：指明Fortran源程序为固定格式。
- -implicitnone：指明默认变量名为未定义。建议在写程序时添加implicit none语句，以避免出现由于默认类型造成的错误。
- -names *keyword*：设定如何解释源代码的标志符和外部名。*keyword*可以为：

  - lowercase：让编译器忽略标识符的大小写不同，并转换外部名为小写。
  - uppercase：让编译器忽略标识符的大小写不同，并转换外部名为大写。
  - as_is：让编译器区分标识符的大小写，并保留外部名的大小写。

- -pad-source、-nopad-source：对固定格式的源码记录是否采用空白填充行尾。
- -stand *keyword*：以指定Fortran标准进行编译，编译时显示源文件中不符合此标准的信息。*keyword*可为f03、f90、f95和none，分别对应显示不符合Fortran 2003、90、95的代码信息和不显示任何非标准的代码信息，也可写为-std*keyword*，此时*keyword*不带f，可为03、90、95。
- -standard-semantics：设定编译器的当前Fortran标准行为是否完全实现。
- -syntax-only：仅仅检查代码的语法错误，并不进行其它操作。
- -wrap-margin、-no-wrap-margin：提供一种在Fortran列表输出时禁止右边缘包装。
- -us：编译时给外部用户定义的函数名添加一个下划线，等价于-assume underscore，如果编译时显示_函数找不到时也许添加此选项即可解决。

4.4.17 数据选项

- 共有选项
  - -fcommon、-fno-common：设定编译器是否将common符号作为全局定义。
  - -fpic、-fno-pic：是否生成位置无关代码。
  - -fpie：类似-fpic生成位置无关代码，但生成的代码只能链接到可执行程序。
    * -gcc：定义GNU宏。
    * -no-gcc：取消定义GNU宏。
    * -gcc-sys：只有在编译系统头文件时定义GNU宏。
  - -mcmodel=*mem_model*:设定生成代码和存储数据时的内存模型。*mem_model*可以为：
    * small：让编译器限制代码和数据使用最开始的2GB地址空间。对所有代码和数据的访问可以使用指令指针（IP）相对地址。
    * medium：让编译器限制代码使用最开始的2GB地址空间，对数据没有内存限制。对所有代码的访问可以使用指令指针（IP）相对地址，但对数据的访问必须采用绝对地址。
    * large：对代码和数据不做内存限制。所有访问都得使用绝对地址。
  - -mlong-double-*n*：覆盖掉默认的长双精度数据类型配置。*n*可以为：

  * 64：设定长双精度数据为64位。
  * 80：设定长双精度数据为80位。

- C/C++专有选项
  - -auto-ilp32：让编译器分析程序设定能否将64位指针缩成32位指针，能否将64位长整数缩成32位长整数。
  - -auto-p32：让编译器分析程序设定能否将64位指针缩成32位指针。
  - -check-pointers=*keyword*：设定编译器是否检查使用指针访问的内存边界。*keyword*可以为：
    * none：禁止检查，此为默认选项。
    * rw：检查通过指针读写的内存边界。
    * write：只检查通过内存写的内存边界。
  - -check-pointers-dangling*keyword*：设定编译器是否对悬挂（dangling）指针参考进行检查。*keyword*可以为：
    * none：禁止检查悬挂指针参考，此为默认选项。
    * heap：检查heap的悬挂指针参考。
    * stack：检查stack的悬挂指针参考。
    * all：检查上述所有的悬挂指针参考。
  - -fkeep-static-consts、-fno-keep-static-consts：设定编译器是否保留在源文件中没有参考的变量分配。

- Fortran专有选项
  - -convert *[keyword]*:转换无格式数据的类型,比如*keyword*为big_endian和little_endian时,分别表示无格式的输入输出为big_endian和little_endian格式，更多格式类型,请看编译器手册。
  - -double-size *size*：设定DOUBLE PRECISION和DOUBLE COMPLEX声明、常数、函数和内部函数的默认KIND。*size*可以为64或128,分别对应KIND=8和KIND=16。
  - -dyncom *"common1,common2,..."*：对指定的common块启用运行时动态分配。
  - -fzero-initialized-in-bss、-fno-zero-initialized-in-bss：设定编译器是否将数据显式赋值为0的变量放置在DATA块内。
  - -intconstant、-nointconstant：让编译器使用FORTRAN 77语法设定整型常数的KIND参数。
  - -integer-size *size*：设定整型和逻辑变量的默认KIND。*size*可以为16、32或64,分别对应KIND=2、KIND=4或KIND=8。

- -no-bss-init：让编译器将任何未初始化变量和显式初始化为0的变量放置在DATA块。默认不启用，放置在BSS块。
- -real-size *size*：设定实型变量的默认KIND。*size*可以为32、64或18，分别对应KIND=4、KIND=8或KIND=16。
- -save:强制变量值存储在静态内存中。此选项保存递归函数和用AUTOMATIC声明的所有变量（除本地变量外）在静态分配中，下次调用时可继续用。默认为-auto-scalar，内在类型INTEGER、REAL、COMPLEX和LOGICAL变量分配到运行时堆栈中。
- -zero、-nozero：是否将所有保存的但未初始化的内在类型INTEGER、REAL、COMPLEX或LOGICAL的局部变量值初始为0。

4.4.18 编译器诊断选项

- -diag-*type*=*diag-list*：控制显示的诊断信息。*type*可以为：
  - enable：启用一个或一组诊断信息。
  - disable：禁用一个或一组诊断信息。
  - error：让编译器将诊断信息变为错误。
  - warning：让编译器将诊断信息变成警告
  - remark：让编译器将诊断信息变为备注。

  *diag-list*可为：driver、port-win、thread、vec、par 、openmp 、warn 、error 、remark 、cpu-dispatch 、 id[,id,...] 、 tag[,tag,...]等。
- -traceback、-notraceback：编译时在对象文件中生成额外的信息使得在运行出错时可以提供源文件回朔信息。
- -w：编译时不显示任何警告，只显示错误。
- -w*n*：设置编译器生成的诊断信息级别。*n*可以为：
  - 0：对错误生成诊断信息，屏蔽掉警告信息。
  - 1：对错误和警告生成诊断信息。此为默认选项。
  - 2：对错误和警告生成诊断信息，并增加些额外的警告信息。
  - 3：对备注、错误和警告生成诊断信息，并在级别2的基础上再增加额外警告信息。建议对产品使用此级别。
  - 4：在级别3的基础上再增加一些警告和备注信息，这些增加的信息一般可以安全忽略。

- -Wabi、-Wno-abi：设定生成的代码不是C++ ABI兼容时是否显示警告信息。
- -Wall：编译时显示警告和错误信息。
- -Wbrief：采用简短方式显示诊断信息。
- -Wcheck：让编译器在对特定代码在编译时进行检查。
- -Werror：将所有警告信息变为错误信息。
- -Werror-all将所有警告和备注信息变为错误信息。
- -Winline：设定编译器显示哪些函数被内联，哪些未被内联。
- -Wunused-function、-Wno-unused-functio：设定是否在声明的函数未使用时显示警告信息。
- -Wunused-variable、-Wno-unused-variable：设定是否在声明的变量未使用时显示警告信息。

4.4.19 兼容性选项

- -f66：FORTRAN程序特有。使用FORTRAN 66标准，默认为使用 Fortran 95标准。
- -f77rtl、-nof77rtl：FORTRAN程序特有。是否使用FORTRAN 77运行时行为，默认为使用Intel Fortran运行时行为。控制以下行为：
  - 当unit没有与一个文件对应时，一些INQUIRE说明符将返回不同的值：
    * NUMBER= 返回0；
    * ACCESS= 返回’UNKNOWN’；
    * BLANK= 返回’UNKNOWN’；
    * FORM= 返回’UNKNOWN’。
  - PAD= 对格式化输入默认为”NO’。
  - NAMELIST和列表输入的字符串必需用单引号或双引号分隔。
  - 当处理NAMELIST输入时：
    * 每个记录的第一列被忽略。
    * 出现在组名前的’$’或’&’必须在输入记录的第二列。
  - -fpscomp [keyword[, keyword...]]、-nofpscomp：FORTRAN程序特有。设定是否某些特征与Intel®Fortran或Microsoft* Fortran PowerStation兼容。*keyword*可以为：
    * none：没有选项需要用于兼容性。

  * [no]filesfromcmd：设定当OPEN声明中FILE=说明符为空时的兼容性。
  * [no]general：设定当Fortran PowerStation和Intel®Fortran语法存在不同时的兼容性。
  * [no]ioformat：设定列表格式和无格式IO时的兼容性。
  * [no]libs：设定可移植性库是否传递给链接器。
  * [no]ldio_spacing：设定是否在运行时在数值量后字符值前插入一个空白。
  * [no]logicals：设定代表LOGICAL值的兼容性。
  * all：设定所有选项用于兼容性。

- -fabi-version=*n*：设定使用指定版本的ABI实现。*n*可以为：
  - 0：使用最新的ABI实现。
  - 1：使用gcc 3.2和gcc 3.3使用的ABI实现。
  - 2：使用gcc 3.4及更高的gcc中使用的ABI实现。
- -gcc-name=*name*：设定使用的gcc编译器的名字。
- -gxx-name*name*：设定使用的g++编译器的名字。

4.4.20 链接和链接器选项

- -Bdynamic：在运行时动态链接所需要的库。
- -Bstatic：静态链接用户生成的库。
- -cxxlib*[=dir]*、-cxxlib-nostd、-no-cxxlib：设定是否使用gcc提供的C++运行时库及头文件。*dir*为gcc二进制及库文件的顶层目录。
- -I*dir*：指明头文件的搜索路径。
- -L*dir*：指明库的搜索路径。
- -l*string*：指明所需链接的库名，如库名为libxyz.a，则可用-lxyz指定。
- -no-libgcc：禁止使用特定gcc库链接。
- -nodefaultlibs：禁止使用默认库链接。
- -nostartfiles：禁止使用标准启动文件链接。
- -nostdlib：禁止使用标准启动和库文件链接。
- -pie、-no-pie：设定编译器是否生成需要链接进可执行程序的位置独立代码

- -pthread：对多线程启用pthreads库。
- -shared：生成共享对象文件而不是可执行文件，必须在编译每个对象文件时使用-fpic选项。
- -shared-intel：动态链接Intel库。
- -shared-libgcc：动态链接GNU libgcc库。
- -static：静态链接所有库。
- -static-intel：静态链接Intel库。
- -static-libgcc：静态链接GNU libgcc库。
- -static-libstdc++：静态链接GNU libstdc++库。
- -u *symbol*：设定指定的符号未定义。
- -v：显示驱动工具编译信息。
- -Wa,*option1[,option2,...]*：传递参数给汇编器进行处理。
- -Wl,*option1[,option2,...]*：传递参数给链接器进行处理。
- -Wp,*option1[,option2,...]*：传递参数给预处理器。
- -Xlinker *option*：将option信息传递给链接器。

4.4.21 其它选项

- -help *[category]*：显示帮助。
- -sox*[=keyword[,keyword]]*、-no-sox：设定是否让编译时在生成的可执行文件中保存编译选项和版本等信息，也可以指定是否保存子程序等信息。
  - inline：包含在各目标文件中的内联子程序名。
  - profile：包含编译时采用-prof-use的子程序列表，以及存储概要信息的.dpi文件名和指明使用的和忽略的概要信息。

  存储的信息可以使用以下方法查看：
  - *objdump -sj .comment a.out*
  - *strings -a a.out | grep comment:*
- -V：显示版本信息。

编译工具	语言或函数	命令
PGF77	ANSI FORTRAN 77	pgf77
PGFORTRAN	ISO/ANSI Fortran 2003	pgfortran、pgf90、pgf95
PGCC	ISO/ANSI C11 and K&R C	pgcc
PGC++	ISO/ANSI C++14 with GNU compatibility	pgc++
PGDBG	Source code debugger	pgdbg
PGPROF	Performance profiler	pgprof

- –version：显示版本信息。
- -watch*[=keyword[, keyword...]]*、-nowatch：设定是否在控制台显示特定信息。*keyword*可以为：
  - none：禁止cmd和source。
  - [no]cmd：设定是否显示驱动工具命令及执行。
  - [no]source：设定是否显示编译的文件名。
  - all：启用cmd和source。

5 PGI C/C++ Fortran编译器

5.1 PGI C/C++ Fortran编译器简介

PGI C/C++ Fortran编译器是一种针对多种CPU与操作系统的高性能编译器，可用于开发复杂且要进行大量计算的程序。当前安装的版本为2016.7和2014.10，分别安装在*/opt/pgi-16.7/linux86-64/16.7*、*/opt/pgi/linux86-64/14.10*。安装在*/opt/intel*，可用*module avail*查看，用*moudle load 模块名*使用，或在自己的*~/.bashrc*之类环境设置文件中添加以下代码设置：

```
PATH=/opt/pgi/linux86-64/14.10/bin:$PATH
MANPATH=$MANPATH:/opt/pgi/linux86-64/14.10/man
export PATH MANPATH
```

PGI编译器编译C、C++、Fortran 77源程序的命令分别为*pgcc*、*pgCC|pgc++*[5]和*pgf77*，编译Fortran 90（为了描述方便，本手册中将Fortran 90、95、2003、2008标准统称为Fortran 90）的源程序的命令有*pgf90*、*pgf901*、*pgf902*、*pgf90_ex*、*pgf95*和*pgfortran*。

官方手册目录：*安装目录/doc*。

[5]2016版为pgc++，之前版本为pgCC

5.2 编译错误

编译时的出错信息类似以下：

```
PGF90-S-0034-Syntax error at or near * (NOlihm.f90: 2)
```

编译错误的格式为：

大写的编译命令-严重级别-错误编号-解释，含指明位置(文件名: 行号)

错误严重级别分为：

- I：信息
- W：警告
- S：严重
- F：致命
- V：其它

Fortran程序编译错误解释,参见:PGI Compiler Reference Guide->Chapter 9. MESSAGES->9.3. Fortran Compiler Error Messages->9.3.2. Message List

5.3 Fortran程序运行错误

Fortran程序运行时错误解释，参见：PGI Compiler Reference Guide->Chapter 9. MESSAGES->9.4. Fortran Run-time Error Messages->9.4.2. Message List

5.4 PGI C/C++编译器重要编译选项

PGI编译器选项非常多，下面仅仅是列出一些本人认为常用的关于编译C程序的*pgcc*命令的重要选项。编译C++程序的*pgc++*|*pgCC*命令有稍微不同，建议仔细查看PGI相关资料。建议仔细查看编译器手册中关于程序优化的部分，多加测试，选择适合自己程序的编译选项以提高性能。

5.4.1 一般选项

- -#：显示编译器、汇编器、链接器的调用信息。
- -c：仅编译成对象文件（.o文件）。
- -defaultoptions和-nodefaultoptions：是否使用默认选项，默认为使用。

- -flags：显示所有可用的编译选项。
- -help*[=option]*：显示帮助信息，option*n*可以为groups、asm、debug、language、linker、opt、other、overall、phase、phase、prepro、suffix、switch、target和variable。
- -Minform=*level*：控制编译时错误信息的显示级别。level可以为fatal、file、severe、warn、inform，默认为-Minform=warn。
- -noswitcherror：显示警告信息后，忽略未知命令行参数并继续进行编译。默认显示错误信息并且终止编译。
- -o file：指定生成的文件名。
- -show：显示现有pgcc命令的配置信息。
- -silent：不显示警告信息，与-Minform=severe等同。
- -v：详细模式，在每个命令执行前显示其命令行。
- -V：显示编译器版本信息。
- -w：编译时不显示任何警告，只显示错误。

5.4.2 优化选项

- -fast：编译时选择针对目标平台的普通优化选项。用*pgcc -fast -help*可以查看等价的开关。优化级别至少为O2，参看-O选项。
- -fastsse：对支持SSE和SSE2指令的CPU（如Intel Xeon CPU）编译时选择针对目标平台的优化选项。用*pgcc -fastsse -help*可以查看等价的开关，优化级别至少为O2，参看-O选项。
- -fpic或-fPIC：编译器生成地址无关代码，以便可用于生成共享对象文件（动态链接库）。
- -Kpic或-KPIC：与-fpic或-fPIC相同，为了与其余编译器兼容。
- -Minfo *[=option[,option,...]]*：显示有用信息到标准错误输出，选项可为all、autoinline、inline、ipa、loop或opt、mp、time或stat。
- -Mipa*[=option[,option,...]]*和-Mnoipa：启用指定选项的过程间分析优化，默认为-Mnoipa。
- -Mneginfo=*option[,option...]*：使编译器显示为什么特定优化没有实现的信息。选项包括concur、loop和all。

- -Mnoopenmp：当使用-mp选项时，忽略OpenMP并行指令。
- -Mnosgimp：当使用-mp选项时，忽略SGI并行指令。
- -Mpfi：生成概要导向工具，此时将会包含特殊代码收集运行时的统计信息以用于子序列编译。-Mpfi必须在链接时也得使用。当程序运行时，会生成概要导向文件pgfi.out。
- -Mpfo：启用概要导向优化，此时必须在当前目录下有概要文件pgfi.out。
- -Mprof*[=option[,option,...]]*：设置性能功能概要选项。此选项可使得结果执行生成性能概要，以便PGPROF性能概要器分析。
- -mp*[=option]*：打开对源程序中的OpenMP并行指令的支持。
- -O*[level]*：设置优化级别。level可设为0、1、2、3、4，其中4与3相同。
- -pg：使用gprof风格的基于抽样的概要刨析。

5.4.3 调试选项

- -g：包含调试信息。

5.4.4 预处理选项

- -C：预处理时保留C源文件中的注释。
- -D*name[=def]*：预处理时定义宏name为def。
- -dD：打印源文件中已定义的宏及其值到标准输出。
- -dI：打印预处理中包含的所有文件信息，含文件名和定义时的行号。
- -dM：打印预处理时源文件已定义的宏及其值，含定义时的文件名和行号。
- -dN：与-dD类似，但只打印源文件已定义的宏，而不打印宏值。
- -E：预处理每个.c文件，将结果发送给标准输出，但不进行编译、汇编或链接等操作。
- -I*dir*：指明头文件的搜索路径。
- -M：打印make的依赖关系到标准输出。
- -MD：打印make的依赖关系到文件file.d，其中file是编译文件的根名字。

- -MM：打印make的依赖关系到标准输出，但忽略系统头文件。
- -MMD：打印make的依赖关系到文件file.d，其中file是编译的文件的根名字，但忽略系统头文件。
- -P：预处理每个文件，并保留每个file.c文件预处理后的结果到file.i。
- -U*name*：去除预处理中的任何name的初始定义。

5.4.5 链接选项

- -Bdynamic：在运行时动态链接所需的库。
- -Bstatic：静态链接所需的库。
- -Bstatic_pgi ：动态链接系统库时静态链接PGI库。
- -g77libs：允许链接GNU *g77*或*gcc*命令生成的库。
- -l*string*：指明所需链接的库名。如库为libxyz.a，则可用-lxyz指定。
- -L*dir*：指明库的搜索路径。
- -m：显示链接拓扑。
- -Mrpath和-Mnorpath:默认为-rpath,以给出包含PGI共享对象的路径。用-Mnorpath可以去除此路径。
- -pgf77libs：链接时添加pgf77运行库，以允许混合编程。
- -r：生成可以重新链接的对象文件。
- -R*directory*：对共享对象文件总搜索directory目录。
- -pgf90libs：链接时添加pgf90运行库，以允许混合编程。
- -shared: 生成共享对象而不是可执行文件,必须在编译每个对象文件时使用-fpic选项。
- -soname*name*：生成共享对象时，用内在的DT_SONAME代替指定的name。
- -u*name*：传递给链接器，以生成未定义的引用。

5.4.6 C/C++语言选项

- -B：源文件中允许C++风格的注释，指的是以//开始到行尾内容为注释。除非指定-C选项，否则这些注释被去除。
- -c8x或-c89：对C源文件采用C89标准。
- -c9x或-c99：对C源文件采用C99标准。

5.4.7 Fortran语言选项

- -byteswapio或-Mbyteswapio：对无格式Fortran数据文件在输入输出时从大端（big-endian)到小端(little-endian)交换比特,或者相反。此选项可以用于读写Sun或SGI等系统中的无格式的Fortran数据文件。
- -i2：将INTEGER变量按照2比特处理。
- -i4：将INTEGER变量按照4比特处理。
- -i8：将默认的INTEGER和LOGICAL变量按照4比特处理。
- -i8storage：对INTEGER和LOGICAL变量分配8比特。
- -Mallocatable*[=95|03]*：按照Fortran 95或2003标准分配数组。
- -Mbackslash和-Mnobackslash：将反斜线(\)当作正常字符（非转义符）处理，默认为-Mnobackslash。-Mnobackslash导致标准的C反斜线转义序列在引号包含的字串中重新解析。-Mbackslash则导致反斜线被认为和其它字符一样。
- -Mextend：设置源代码的行宽为132列。
- -Mfixed、-Mnofree和-Mnofreeform：强制对源文件按照固定格式进行语法分析，默认.f或.F文件被认为固定格式。
- -Mfree和-Mfreeform：强制对源文件按照自由格式进行语法分析，默认.f90、.F90、.f95或.F95文件被认为自由格式。
- -Mi4和-Mnoi4:将INTEGER看作INTEGER*4。-Mnoi4将INTEGER看作INTEGER*2。
- -Mnomain：当链接时，不包含调用Fortran主程序的对象文件。
- -Mr8和-Mnor8：将REAL看作DOUBLE PRECISION，将实(REAL)常数看作双精度(DOUBLE PRECISION)常数。默认为否。
- -Mr8intrinsics *[=float]*和-Mnor8intrinsics:将CMPLX看作DCMPLX,将REAL看作DBLE。添加float选项时，将FLOAT看作DBLE。

- -Msave和-Mnosave：是否将所有局部变量添加SAVE声明，默认为否。
- -Mupcase和-Mnoupcase：是否保留名字的大小写。-Mnoupcase导致所有名字转换成小写。注意，如果使用-Mupcase，那么变量名X与变量名x不同，并且关键字必须为小写。
- -Mcray=*pointer*：支持Cray指针扩展。
- -module *directory*：指定编译时保存生成的模块文件的目录。
- -r4：将DOUBLE PRECISION变量看作REAL。
- -r8：将REAL变量看作DOUBLE PRECISION。

5.4.8 平台相关选项

- -Kieee和-Knoieee：浮点操作是否严格按照IEEE 754标准。使用-Kieee时一些优化处理将被禁止，并且使用更精确的数值库。默认为-Knoieee，将使用更快的但精确性低的方式。
- -Ktrap=*[option,[option]...]*：控制异常发生时CPU的操作。选项可为divz、fp、align、denorm、inexact、inv、none、ovf、unf，默认为none。
- -Msecond_underscore和-Mnosecond_underscore：是否对已有_的Fortran函数名添加第二个_。与*g77*编译命令兼容时使用，因为*g77*默认符号后添加第二个_。
- -mcmodel=*small*|*medium*：使内存模型是否限制对象小于2GB(small)或允许数据块大于2GB(medium)。medium时暗含-Mlarge_arrays选项。
- -tp *target*：target可以为haswell等，默认与编译时的平台一致。

6 GNU C/C++ Fortran编译器

6.1 GNU C/C++ Fortran编译器简介

GNU C/C++ Fortran(GCC)编译器为系统自带的编译器，当前安装的版本为4.4.7和3.4.6。默认为4.4.7版本，用户无需特殊设置即可使用。GNU编译器编译C、C++源程序的命令分别为4.4.7版本的*gcc*和*g++*及3.4.6版本的*gcc34*和*g++34*；4.4.7版本的*gfortran*可以直接编译Fortran 77、90源程序。3.4.6版本的*g77*只能编译F77程序，不可编译Fortran 90源程序。

6.2 编译错误

C/C++程序编译时错误信息类似:

```
netlog.c: In function 'main':
netlog.c:84:7: error: 'for' loop initial declarations are only allowed in C99 mode
netlog.c:84:7: note: use option -std=c99 or -std=gnu99 to compile your code
```

编译错误的格式为:

- 源文件名: 函数中
- 源文件名:行数:列数:错误类型:具体说明
- 源文件名:行数:列数:注解:解决办法

Fortran程序编译时错误信息类似:

```
NOlihm.f90:146.14:

  n2nd=0; npr=0
             1
Error: Symbol 'npr' at (1) has no IMPLICIT type
```

编译错误的格式为:

- 源文件名:行数:列数:
- 源文件代码
- 1指示出错位置
- 错误类型: 具体说明

6.3 GNU C/C++编译器GCC重要编译选项

GNU编译器GCC是Linux系统自带的编译器，系统安装的版本为4.4.7和3.4.6，选项非常多，下面仅仅是列出一些针对4.4.7本人认为常用的重要选项，建议仔细看GCC相关资料。

建议仔细查看编译器手册中关于程序优化的部分，多加测试，选择适合自己程序的编译选项以提高性能。

6.3.1 控制文件类型的选项

- -x language：明确指定而非让编译器判断输入文件的类型。language可为：
  - c、c-header、c-cpp-output
  - c++、c++-header、c++-cpp-output
  - objective-c、objective-c-header、objective-c-cpp-output
  - objective-c++、objective-c++-header、objective-c++-cpp-output
  - assembler、assembler-with-cpp
  - ada
  - f95、f95-cpp-input
  - java
  - treelang

  当language为none时，禁止任何明确指定的类型，其类型由文件名后缀设定。
- -c：仅编译成对象文件（.o文件），并不进行链接。
- -o file：指定生成的文件名。
- -v：详细模式，显示在每个命令执行前显示其命令行。
- -###：显示编译器、汇编器、链接器的调用信息但并不进行实际编译，在脚本中可以用于捕获驱动器生成的命令行。
- –help：显示帮助信息。
- –target-help：显示目标平台的帮助信息。
- –version：显示编译器版本信息。

6.3.2 C/C++语言选项

- -ansi：C模式时，支持所有ISO C90指令。在C++模式时，去除与ISO C++冲突的GNU扩展。
- -std=*val*：控制语言标准，可以为c89、iso9899:1990、iso9899:199409、c99、c9x、iso9899:1999、iso9899:199x、gnu89、gnu99、gnu9x、c++98、gnu++98。
- -B：在源文件中允许C++风格的注释，指的是以//开始到行尾内容为注释。除非指定-C选项，否则这些注释被去除。

- -c8x或-c89：对C源文件采用C89标准。
- -c9x或-c99：对C源文件采用C99标准。

6.3.3 Fortran语言选项

- -fconvert=*conversion*：指定对无格式Fortran数据文件表示方式，其值可以为：native，默认值；swap，在输入输出时从大端（big-endian）到小端（little-endian）交换比特，或者相反；big-endian，用大端方式读写；little-endian，用小端方式读写。
- -ff2c：与*g77*和*f2c*命令生成的代码兼容。
- -ffree-form和-ffixed-form：声明源文件是自由格式还是固定格式，默认从Fortran 90起的源文件为自由格式，之前的Fortran 77等的源文件为固定格式。
- -fdefault-double-8：设置DOUBLE PRECISION类型为8比特。
- -fdefault-integer-8：设置INTEGER和LOGICAL类型为8比特。
- -fdefault-real-8：设置REAL类型为8比特。
- -fno-backslash：将反斜线(\)当作正常字符（非转义符）处理。
- -fno-underscoring：不在名字后添加_。注意：*gfortran*默认行为与*g77*和*f2c*不兼容，为了兼容需要加-ff2c选项。除非使用者了解与现有系统环境的集成，否则不建议使用-fno-underscoring选项。
- -ffixed-line-length-*n*：设置固定格式源代码的行宽为n。
- -ffree-line-length-*n*：设置自由格式源代码的行宽为n。
- -fimplicit-none：禁止变量的隐式声明，所有变量都需要显式声明。
- -fmax-identifier-length=*n*：设置名称的最大字符长度为n，Fortran 95和200x的长度分别为31和65。
- -fno-automatic：将每个程序单元的本地变量和数组声明具有SAVE属性。
- -fcray-pointer：支持Cray指针扩展。
- -fopenmp：编译OpenMP并行程序。
- -M*dir*和-J*dir*：指定编译时保存生成的模块文件目录。
- -fsecond-underscore：默认*gfortran*对外部函数名添加一个_，如果使用此选项，那么将添加两个_。此选项当使用-fno-underscoring选项时无效。此选项当使用-ff2c时默认启用。

- -std=*val*：指明Fortran标准，val可以为f95、f2003、legacy。
- -funderscoring：对外部函数名没有_的加_，以与一些Fortran编译器兼容。

6.3.4 警告选项

- -fsyntax-only：仅仅检查代码的语法错误，并不进行其它操作。
- -w：编译时不显示任何警告，只显示错误。
- -Wfatal-errors：遇到第一个错误就停止，而不尝试继续运行显示更多错误信息。

6.3.5 调试选项

- -g：包含调试信息。
- -ggdb：包含利用gbd调试时所需要的信息。

6.3.6 优化选项

- -O*[level]*：设置优化级别。优化级别level可以设置为0、1、2、3、s。

6.3.7 预处理选项

- -C：预处理时保留C源文件中的注释。
- -D *name*：预处理时定义宏name的值为1。
- -D *name=def*：预处理时定义name为def。
- -U *name*：预处理时去除的任何name初始定义。
- -undef：不预定义系统或GCC声明的宏，但标准预定义的宏仍旧被定义。
- -dD：显示源文件中定义的宏及其值到标准输出。
- -dI：显示预处理中包含的所有文件，包括文件名和定义时的行号信息。
- -dM：显示预处理时源文件中定义的宏及其值，包括定义时文件名和行号。
- -dN：与-dD类似，但只显示源文件中定义的宏，而不显示宏值。
- -E：预处理各.c文件，将结果发给标准输出，不进行编译、汇编或链接。
- -I*dir*：指明头文件的搜索路径。

- -M：打印make的依赖关系到标准输出。
- -MD：打印make的依赖关系到文件file.d，其中file是编译文件的根名字。
- -MM：打印make的依赖关系到标准输出，但忽略系统头文件。
- -MMD：打印make的依赖关系到文件file.d，其中file是编译的文件的根名字，但忽略系统头文件。
- -P：预处理每个文件，并保留每个file.c文件预处理后的结果到file.i。

6.3.8 链接选项

- -pie：在支持的目标上生成地址无关的可执行文件。
- -s：从可执行文件中去除所有符号表。
- -rdynamic：添加所有符号表到动态符号表中。
- -static：静态链接所需的库。
- -shared：生成共享对象文件而不是可执行文件，必须在编译每个对象文件时使用-fpic选项。
- -shared-libgcc：使用共享libgcc库。
- -static-libgcc：使用静态libgcc库。
- -u *symbol*：确保符号symbol未定义，强制链接一个库模块来定义它。
- -I*dir*：指明头文件的搜索路径。
- -l*string*：指明所需链接的库名，如库为libxyz.a，则可用-lxyz指定。
- -L*dir*：指明库的搜索路径。
- -B*dir*：设置寻找可执行文件、库、头文件、数据文件等路径。

6.3.9 i386和x86-64平台相关选项

- -mtune=*cpu-type*：设置优化针对的CPU类型，可为：generic、core2、opteron、opteron-sse3、bdver1、bdver2等，bdver1为针对本系统AMD Opteron CPU的。
- -march=*cpu-type*：设置指令针对的CPU类型，CPU类型与上行中一样。
- -mieee-fp和-mno-ieee-fp：浮点操作是否严格按照IEEE标准。

6.3.10 约定成俗选项

- -fpic：生成地址无关的代码以用于共享库。
- -fPIC：如果目标机器支持，将生成地址无关的代码。
- -fopenmp：编译OpenMP并行程序。
- -fpie和-fPIE：与-fpic和-fPIC类似，但生成的地址无关代码，只能链接到可执行文件中。

Part VII

MPI并行程序编译及运行

本系统的通信网络有两种：56Gbps InfiniBand高速计算网络和千兆以太网。InfiniBand网络相比千兆以太网具有高带宽低延迟的特点，通信性能比千兆以太网要高很多，建议使用。

本系统安装有多种MPI实现，主要有：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/**[6]。系统默认设置使用的InfiniBand、Intel编译器与Intel MPI的组合。默认Intel MPI的安装目录为*/opt/intel/impi/5.0.2*，建议使用。

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。

7 MPI并行程序的编译

7.1 Intel MPI库

Intel MPI库[7]是一种多模消息传递接口(MPI)库，所安装的5.0版本Intel MPI库实现了MPI V3.0标准。Intel MPI库可以使开发者采用新技术改变或升级其处理器和互联网络而无需改编软件或操作环境成为可能。主要包含以下内容：

- Intel MPI库运行时环境(RTO)：具有运行程序所需要的工具，包含多功能守护进程(MPD)、Hydra及支持的工具、共享库(.so)和文档。
- Intel MPI库开发套件(SDK)：包含所有运行时环境组件和编译工具，含编译器命令，如*mpiicc*、头文件和模块、静态库(.a)、调试库、追踪库和测试代码。

7.1.1 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。

其中：

[6]具有不同版本的Open MPI与编译器的组合。

[7]主页：http://software.intel.com/en-us/intel-mpi-library/

表 8: 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

- ia32：IA-32架构。
- intel64：Intel 64(x86_64, amd64)架构。
- 移植现有的MPI程序到Intel MPI库时，请重新编译所有源代码。
- 如需显示某命令的简要帮助，可以不带任何参数直接运行该命令。

7.1.2 编译命令参数

- -mt_mpi：采用以下级别链接线程安全的MPI库：MPI_THREAD_FUNNELED, MPI_THREAD_SERIALIZED或MPI_THREAD_MULTIPLE。

  Intel MPI库默认使用MPI_THREAD_FUNNELED级别线程安全库。

  注意：

  – 如使用Intel C编译器编译时添加了-openmp、-qopenmp或-parallel参数，那么使用线程安全库。

  – 如果用Intel Fortran编译器编译时添加了如下参数，那么使用线程安全库：

    * -openmp

    * -qopenmp
    * -parallel
    * -threads
    * -reentrancy
    * -reentrancy threaded

- -static_mpi：静态链接Intel MPI库，并不影响其它库的链接方式。
- -static：静态链接Intel MPI库，并将其传递给编译器，作为编译器参数。
- -config=*name*：使用的配置文件。
- -profile=*profile_name*：使用的MPI分析库文件。
- -t或-trace：链接Intel Trace Collector库。
- -check_mpi：链接Intel Trace Collector正确性检查库。
- -ilp64：打开局部ILP64支持。对于Fortran程序编译时如果使用-i8选项，那么也需要此ILP64选项。
- -dynamic_log：与-t组合使用链接Intel Trace Collector库。不影响其它库链接方式。
- -g：采用调试模式编译程序，并针对Intel MPI调试版本生成可执行程序。可查看官方手册Environment variables部分*I_MPI_DEBUG*变量查看-g参数添加的调试信息。采用调试模式时不对程序进行优化，可查看*I_MPI_LINK*获取Intel MPI调试版本信息。
- -link_mpi=*arg*：指定链接MPI的具体版本，具体请查看*I_MPI_LINK*获取Intel MPI版本信息。此参数将覆盖掉其它参数，如-mt_mpi、-t=log、-trace=log和-g。
- -O：启用编译器优化。
- -fast：对整个程序进行最大化速度优化。此参数强制使用静态方法链接Intel MPI库。*mpiicc*、*mpiicpc*和*mpiifort*编译命令支持此参数。
- -echo：显示所有编译命令脚本做的信息。
- -show：仅显示编译器如何链接，但不实际执行。
- -{cc,cxx,fc,f77,f90}=*compiler*：选择使用的编译器。如：*mpicc -cc=icc -c test.c*。
- -gcc-version=*nnn*，设置编译命令*mpicxx*和*mpiicpc*编译时采用部分GNU C++环境的版本，如nnn的值为340，表示对应GNU C++ 3.4.x。
- -compchk：启用编译器设置检查，以保证调用的编译器配置正确。
- -v：显示版本信息。

<nnn>值	GNU* C++版本
320	3.2.x
330	3.3.x
340	3.4.x
400	4.0.x
410	4.1.x
420	4.2.x
430	4.3.x
440	4.4.x
450	4.5.x
460	4.6.x
470	4.7.x

7.1.3 环境变量

- *I_MPI_{CC,CXX,FC,F77,F90}_PROFILE*和*MPI{CC,CXX,FC,F77,F90}_PROFILE*:
  - 默认分析库。
  - 语法：*I_MPI_{CC,CXX,FC,F77,F90}_PROFILE=<profile_name>*。
  - 过时语法：*MPI{CC,CXX,FC,F77,F90}_PROFILE=<profile_name>*。
- *I_MPI_TRACE_PROFILE*:
  - 设定-trace参数使用的默认分析文件。
  - 语法：*I_MPI_TRACE_PROFILE=<profile_name>*
  - *I_MPI_{CC,CXX,F77,F90}_PROFILE*环境变量将覆盖掉*I_MPI_TRACE_PROFILE*。
- *I_MPI_CHECK_PROFILE*:
  - 设定-check_mpi参数使用的默认分析。
  - 语法：*I_MPI_CHECK_PROFILE=<profile_name>*。
- *I_MPI_CHECK_COMPILER*:
  - 设定启用或禁用编译器兼容性检查。
  - 语法：*I_MPI_CHECK_COMPILER=<arg>*。
    * <arg>为enable | yes | on | 1时打开兼容性检查。

* <arg>为disable | no | off | 0时，关闭编译器兼容性检查，为默认值。

- *I_MPI_{CC,CXX,FC,F77,F90}*和*MPICH_{CC,CXX,FC,F77,F90}*：
  - 语法：*I_MPI_{CC,CXX,FC,F77,F90}*=<*compiler*>。
  - 过时语法：*MPICH_{CC,CXX,FC,F77,F90}*=<*compiler*>。
  - <compiler>为编译器的编译命令名或路径。
- *I_MPI_ROOT*：
  - 设置Intel MPI库的安装目录路径。
  - 语法：*I_MPI_ROOT*=<*path*>。
  - <path>为Intel MPI库的安装后的目录。
- *VT_ROOT*：
  - 设置Intel Trace Collector的安装目录路径。
  - 语法：*VT_ROOT*=<*path*>。
  - <path>为Intel Trace Collector的安装后的目录。
- *I_MPI_COMPILER_CONFIG_DIR*：
  - 设置编译器配置目录路径。
  - 语法：*I_MPI_COMPILER_CONFIG_DIR*=<*path*>。
  - <path>为编译器安装后的配置目录，默认值为<installdir>/<arch>/etc。
- *I_MPI_LINK*：
  - 设置链接MPI库版本。
  - 语法：*I_MPI_LINK*=<*arg*>。
  - <arg>可为：
    * opt：优化的单线程版本Intel MPI库；
    * opt_mt：优化的多线程版本Intel MPI库；
    * dbg：调试的单线程版本Intel MPI库；
    * dbg_mt：调试的多线程版本Intel MPI库；
    * log：日志的单线程版本Intel MPI库；
    * log_mt：日志的多线程版本Intel MPI库。

7.1.4 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
- 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

7.1.5 调试

使用以下命令对Intel MPI库调用GDB调试器：*mpirun -gdb -n 4 ./testc*

可以像使用GDB调试串行程序一样调试。

也可以使用以下命令附着在一个运行中的作业上：

mpirun -n 4 -gdba <pid>

其中<pid>为运行中的MPI作业进程号。

环境变量I_MPI_DEBUG提供一种获得MPI应用运行时信息的方式。可以设置此变量的值从0到1000，值越大，信息量越大。

mpirun -genv I_MPI_DEBUG 5 -n 8 ./my_application

更多信息参见程序调试章节。

7.1.6 追踪

使用-t或-trace选项链接调用Intel Trace Collector库生成可执行程序。此与当在mpiicc或其它编译脚本中使用-profile=vt时具有相同的效果。

mpiicc -trace test.c -o testc

在环境变量VT_ROOT中包含用Intel Trace Collector库路径以便使用此选项。设置I_MPI_TRACE_PROFILE为<profile_name>环境变量指定另一个概要库。如设置I_MPI_TRACE_PROFILE为vtfs，以链接fail-safe版本的Intel Trace Collector库。

7.1.7 正确性检查

使用-check_mpi选项调用Intel Trace Collector正确性检查库生成可执行程序。此与当在mpiicc或其它编译脚本中使用-profile=vtmc时具有相同的效果。

mpiicc -profile=vtmc test.c -o testc

或

mpiicc -check_mpi test.c -o testc

在环境变量VT_ROOT中包含用Intel Trace Collector库路径以便使用此选项。设置I_MPI_CHECK_PROFILE为<profile_name>环境变量指定另一个概要库。

7.1.8 统计收集

如果想收集在应用使用的MPI函数统计，可以设置I_MPI_STATS环境变量的值为1到10。设置好后再运行MPI程序，则在stats.txt文件中存储统计信息。

7.2 Open MPI库

Open MPI[8]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*mpi-selector-menu*选择与openmpi相关的项自己设置即可。

Open MPI的安装目录在*/opt/openmpi*下。

Open MPI的编译命令主要为：

- C程序：*mpicc*
- C++程序：*mpic++*、*mpicxx*、*mpiCC*
- Fortran 77程序：*mpif77*、*mpif90*、*mpifort*

[8]主页：http://www.open-mpi.org/

- Fortran 90程序：*mpif90*
- Fortran程序：*mpifort*[9]

*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] ...*

编译参数可以为：

- –showme：显示所调用的编译器所调用编译参数等信息。
- –showme:compile：显示调用的编译器的参数
- –showme:link：显示调用的链接器的参数
- –showme:command：显示调用的编译命令
- –showme:incdirs：显示调用的编译器所使用的头文件目录，以空格分隔。
- –showme:libdirs：显示调用的编译器所使用的库文件目录，以空格分隔。
- –showme:libs：显示调用的编译器所使用的库名，以空格分隔。

[9]注意为mpifort，而不是Intel MPI的mpiifort

- –showme:version：显示Open MPI的版本号。

默认使用配置Open MPI时所用的编译器及其参数，可以利用环境变量来改变。环境变量格式为*OMPI_value*，其*value*可以为：

- *CPPFLAGS*：调用C或C++预处理器时的参数
- *LDFLAGS*：调用链接器时的参数
- *LIBS*：调用链接器时所添加的库
- *CC*：C编译器
- *CFLAGS*：C编译器参数
- *CXX*：C++编译器
- *CXXFLAGS*：C++编译器参数
- *F77*：Fortran 77编译器
- *FFLAGS*：Fortran 77编译器参数
- *FC*：Fortran 90编译器
- *FCFLAGS*：Fortran 90编译器参数

7.3 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，具体优化选项等，请参看Intel MPI、Open MPI以及Intel、PGI和GCC编译器手册。

8 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。

Part VIII
程序调试

此部分主要介绍Intel调试器，其它的，如GNU调试器和PGI调试器，很多调试命令类似Intel调试器，请自己查看相关资料。

9 Intel调试器简介

Intel调试器(IDB)是一个全功能的象征性源代码应用程序调试器，包括以下功能：

- 调试C/C++和Fortran程序
- 反汇编和检查机器码和机器寄存器值
- 调试多线程应用（只支持在Linux主机上）
- 调试Intel众核(MIC)应用（只支持在Linux主机上）

Intel调试器在Linux系统上有图形界面(GUI)和命令行(command line)两种方式，其内建命令具有IDB（Intel调试）和GDB（GNU调试）两种模式。

Intel调试器的特性主要包含：

- C/C++语言支持；
- 汇编语言支持；
- Fortran语言支持，包含Fortran 95/90标准；
- 访问程序访问的寄存器；
- 修改寄存器的位字段编辑器；
- 与Intel Inspector XE的内存错误分析特性兼容。

在Linux主机上，基于Intel C++编译器、Intel Click Plus或OpenMP运行时环境，Intel调试器有助于在应用开发中引入并行。Intel调试器提供以下并行调试特性：

- 线程数据共享分析，用于探测不同线程对相同数据元素的访问（针对C/C++和Fortran）；
- 智能断点，用于在从不同线程重入函数调用上停止程序执行；

- 显示向量寄存器的视图，如Intel Streaming SIMD Extensions (Intel SSE)寄存器，具有利用单指令多数据(SIMD)指令集和广泛的格式和编辑选项调试并行数据；
- 模拟串行执行OpenMP代码或Intel Cilk Plus代码的模式；
- OpenMP运行时信息视图集，用于高级OpenMP程序状态分析。

图形界面的Intel调试器提供了完全的调试进程控制，通过点击工具栏按钮，就可以使用大多数基本函数，如单步(single-step)执行、执行完函数的单步执行(step-through-function)、运行和显示内存。图形界面的Intel调试器支持多源代码窗口、计算表达式和改变其值，拖曳表达式到计算窗口等。

图形界面的Intel调试器相对简单，本手册主要介绍基于命令行的Intel调试器，但部分功能需要图形界面的调试器。

10 准备所需要调试的程序

10.1 准备调试代码源代码

调试程序时，一般无需修改程序源代码，但是在程序中建议做如下改变：

- 如果程序运行后，利用调试器难于终止，请设置一个初始停止点；
- 在源代码增加一些断言，以便帮助定位错误。

10.2 准备编译器和链接器环境

调试信息被编译器存储在.o文件。信息的级别和格式由编译器选项控制。

对于Intel C/C++或Fortran编译器，采用-g选项，例如：

- *icc -g hello.c*
- *icpc -g hello.cpp*
- *ifort -g hello.f90*

对于GCC编译器，采用-g选项。对于一些较老版本的GCC，此选项也许会产生DWARF-1标准的调试信息，如果这样，请使用-gdwarf-2选项，例如：

- *gcc -gdwarf-2 hello.c*
- *g++ -gdwarf-2 hello.cpp*

- *gfortran -gdwarf-2 hello.f90*

调试信息将通过*ld*命令导入到a.out（可执行程序）或.so（共享库）文件中。

如果是在调试优化编译的代码，采用-g选项将自动增加-O0选项。

请参看调试优化编译的代码部分中关于-g和相关扩展调试选项及它们的与优化之间的关系。

10.3 调试优化编译的代码

Intel调试器可以通过使用-g参数帮助调试优化编译的程序。但是关于此程序的信息也许并不准确，尤其是变量的地址和值经常没有被正确报告，这是因为通用调试信息模式无法全部表示-O1、-O2、-O3及其它优化选项的复杂性。

为了避免此限制，采用Intel编译器编译程序时在所需的-O1、-O2或-O3优化选项同时指明-g和-debug扩展选项。这会产生具有更多高级但更少通用支持的调试信息，主要激活以下：

- 给出变量的正确地址和值，不管其是在寄存器或不同时间在不同地址时。注意：
  - 在程序中，一些变量可能被优化掉或转换成不同类型的数据，或其地址没有在所有点都被记录。在这些情形下，打印变量时将显示无值$< novalue >$。
  - 否则，这些值和地址将正确，但这些寄存器没有地址，调试器中print &i命令将打印一条警告。
  - 尽管break main命令通常将在程序开始处理后停止，但程序大多数变量和参数在程序的开始处理和结束处理时是未定义的。
- 在堆栈追踪中显示内联函数，这通过使用inline关键词识别。注意：
  - 只有在堆栈顶端和通常（非内联）调用的函数显示指令指针，其原因在于其它函数与其调用的内联函数共享硬件定义的堆栈帧。
  - 返回指令将只返回对那些采用调用指令时是非内联调用函数的控制，其原因在于内联调用没有定义返回地址。
  - up、down和call命令以通常方式工作。
- 允许在内联函数中设置断点。

Intel调试器存在以下限制：

优化经常导致产生的对某一行源代码的指令的顺序与源代码的顺序不一致；针对某一行的指令也许与对其它源码行的指令混合。在这些代码中单步追踪时，程序将趋向于不顺次在每一行源代码上停止，而是在源代码行变化发生时停止。

10.4 准备所需要调试的并行程序

必须用Intel编译器编译源代码才可以使用Intel调试器特性，比如分析共享数据或在重入函数调用中停止。

为了使用并行调试特性，需要：

- 如果存在makefile编译配置文件，请对它进行编辑。
- 在命令行添加编译器选项-debug parallel。
- 重编译程序。

10.5 编译所要调试的程序

下面以常做为例子的hello程序为例介绍。

- hello.c例子：

```
#include <stdio.h>
int main()
{
   printf("Hello␣World!\n");
   return 0;
}
```

编译：

icc -debug -O0 helloworld.c -o helloworld

- hello.f90例子：

```
program main
print *,"Hello␣World!"
end program main
```

编译：

ifort -debug -O0 helloworld.f90 -o helloworld

11 开始调试程序

11.1 启动基于命令行的Intel调试器

在Linux系统上Intel调试器的*idb*命令默认使用图形界面启动，为了使用命令行方式启动，请运行命令*idbc*。

Intel调试器有两种调试模式：IDB和GDB模式，可以分别以以下方式启动：

- *idbc -idb*
- *idbc -gdb*

查看可用的选项，可以运行命令*idbc --help*。

启动所需要调试的程序，有几种方式：

- 启动调试器时运行程序：*idbc yourprogram*
- 启动调试器后调用程序：
  - 启动调试器：*idbc*
  - 在idbc命令行中启动程序：file yourprogram
- 调试出错时生成的core文件：*idbc yourprogram corefilename*
- 启动调试器时附着到某个进程：*idbc -p processid*

11.2 在调试器中卸载程序

- file（只用于GDB模式）。不跟任何参数将卸载当前可执行程序，只卸载当前可执行程序。
- unload [pid|filename]（IDB模式）。指明进程号或文件名卸载。
- detach。以附着到某个进程启动调试器时取消附着。

11.3 显示源代码

在调试器启动后的命令行中输入list命令可以显示源代码，如输入list main，将显示main函数的代码。

11.4 运行程序

在命令行中输入run，将开始运行程序。

11.5 设置和删除断点

- 设置断点：
  - 输入以下命令：break main
    此时在程序main处设置了一个断点。
  - 输入run再次运行程序
    应用将停止在设置的断点处。
- 删除断点：
  - 列出所有设置的断点ID号：info breakpoints
    调试器将显示所有存在的断点。
  - 指明所要删除的断点ID号。如果从开始调试后没有设置其它断点，那么只有1个断点，其ID号为1。
  - 删除此断点：delete breakpoint 1
    那么将删除设置断点1。
  - 重新运行程序。
    那么程序将运行并显示“Hello World!”，并退出程序。

11.6 控制进程环境

用户可以：1、对进程的环境变量进行设置或者取消设置以便在将来使用；2、设置与当前调试器环境和启动调试器的shell不同的环境。设置的变量将影响后续调试的新进程。环境命令不影响当前运行进程。设置的环境变量不改变或显示调试器的环境变量，它们只影响新产生的进程。

- 显示当前集的所有环境变量，可采用以下方式之一：
  - show environment（仅gdb模式）
  - export（仅idb模式）
  - printenv（仅idb模式）
  - setenv（仅idb模式）

- 增加或改变环境变量，可采用以下命令之一：
  - set environment（仅gdb模式）
  - export（仅idb模式）
  - setenv（仅idb模式）
-
  - 取消一个环境变量，可采用以下命令之一：
    * unset environment（仅gdb模式）
    * unsetenv（仅idb模式）

注意：Intel调试器没有命令可以简单回到调试器启动时的环境变量的初始状态，用户必须正确设置和取消环境变量。

11.7 执行一行代码

如果源代码当前行是函数调用，那么可以步入(step into)或者跨越(step voer)此函数。

1. 步进执行一行源代码，步入到此函数中：

   (a) 使用step命令
   应用执行一行代码。如果当前行是函数调用，那么应用步入到函数中，即不执行完此函数调用。

2. 步进执行一行源代码，跨越执行完此函数：

   (a) 使用next命令
   应用执行一行代码。如果当前行是函数调用，那么应用跨越此函数，即执行完此函数调用。

11.8 执行代码直到

运行代码直到某行或某个表达式，可用until命令。

11.9 执行一行汇编指令

如果应用的当前指令为函数调用，那么可以步入或者跨越此函数。

1. 步进执行一行汇编指令，步入到此函数中：

(a) 使用stepi命令

应用执行一行指令。如果下一行指令是函数调用，那么应用步入到函数中。

2. 步进执行一行汇编指令，跨越此执行完此函数：

(a) 使用nexti命令

应用执行到下一汇编代码。如果当前行是跳出或调用，那么应用跨越过它。

11.10 显示变量或表达式值

利用print命令可以显示变量值或表达式的值。如：

- 显示变量val2的当前值：print val2
- 显示表达式val2*2的值：print val2*2

11.11 退出调试器

在命令行中输入quit，将退出调试器。

12 传递命令给调试器

12.1 调试多进程

图形界面Intel调试器不支持调试多进程，采用命令行方式代替。

调试器可以同时发现和控制多个进程，由于以下原因之一：

- 它产生的此进程。
- 它附着的此进程。

在任何时刻，利用Intel调试器只可以检查或执行一个单独的所控制的进程。剩余的被停止，用户必须精确地将调试器变换到想要调试的进程，并停止正在控制的进程。

显示调试器所控制的进程：

1. 如没有在IDB模式下，利用以下命令转换到IDB模式：

(a) $cmdset = ”idb”

(b) 输入process或show process命令
调试器显示它所控制的任何进程。

2. 如果想变换到GDB模式，利用以下命令：

(a) $cmdset = ”gdb”

3. 将调试器变换到某一特定进程：

(a) 如没有在IDB模式下，利用以下命令转换到IDB模式：
(idb) $cmdset = ”idb”

(b) 输入process命令
变换离开的进程将保持停止，直到或者调试器退出或者变换到它并继续它。

4. 如果想变换到GDB模式，利用以下命令：

(a) $cmdset = ”gdb”

12.2　支持多调用框架、线程和源

进程包含一个或多个执行线程。线程执行函数。函数是指令的序列，是由编译器从源文件的源行中编译成的。

在进入Intel调试器调试进程时，也许会非常乏味地重复指定哪个线程、源文件等，以便应用命令。为了避免这样，每次调试器停止进程时，会重新建立用户命令的静态上下文和动态上下文。动态上下文的组件依赖于用户运行的程序，静态上下文独立于用户的运行。

用户可以使用调试器变量显示上下文的组件或通过使用其它命令显示。

静态上下文包含下表中的：

当前程序	info sharedlibrary（GDB模式）或listobj（IDB模式），info file
当前文件	print $curfile
当前行	print $curline

动态上下文包含下表中的：

Intel调试器保持静态和动态上下文组件一致作为上下文改变。调试器决定当前文件和行对应进程停止的位置，但用户也可通过下面命令直接改变动态上下文：

- frame（GDB模式）

where	当前调用框架
print $curprocess	当前进程
print $curthread	当前线程
thread	导致调试器获得进程控制的线程执行事件

- up或down
- func（IDB模式）
- process（IDB模式）
- thread

用户可以利用file（GDB模式）或unload（IDB模式）命令卸载程序。

12.3　命令、文件名和变量补全

Intel调试器支持命令、文件名和变量的补全。在Intel调试器命令行中开始键入一个命令、文件名或变量名，然后按Tab键。如果有不只一个备选，调试器会发出铃声。再一次按Tab键，将列出备选。

利用单引号和双引号影响可能备选集。利用单引号填充C++名字，包含特殊字符的(":", "<", ">", "("等)。利用双引号告诉调试器在文件名中查看备选。

12.4　自定义命令

Intel调试器支持用户自定义命令。

- GDB模式：

  用户定义的命令支持在定义体内包含if、while、loop_break和loop_continue命令。用户定义的命令最多可有10个参数，以空白分割。参数名依次为$arg0, $arg1, $arg2, . . ., $arg9。参数总数存储在$argc中。

  其步骤为：

  – 输入define commandname
  – 每行输入一个命令
  – 输入end

- IDB模式：

  利用alias命令来定义或显示用户自己的命令。

  定义可以包含：

  – 另一个alias的名字，如果在定义中嵌套的alias是第一个识别符。
  – 引号包含的字符串，如果在指定的引号前加了反斜线。两个引用标记不能一起用；它们必须用一个空格或至少一个字符分割。

13 调试并行程序

13.1 与线程和进程组一起工作

13.1.1 总览

线程是进程内部单个、串行控制流。每个线程包含单个执行点。线程在单个地址空间中（共享）执行；因此，进程的线程可以读写相同的内存地址。

多个进程执行时，当用户需要关注某个进程时，它却恼人地或不切实际地枚举所有进程。

当为了设置代码断点而定义停止线程和线程过滤器时，用户需要定义线程集。

用户可以以紧凑方式指定进程或线程集，集可包含一个或多个范围。用户可以对每个进程集执行普通操作，调试器变量既可以存储集也可以存储范围以便操作、引用和查看。

13.1.2 进程和线程集表示法

- 指明进程和线程集

  进程或线程集包含进程或线程ID的一个或多个连续范围，以逗号(,)分割。

  1. 为了指明一个进程集，采用下面表示法：

     [range {,. . .}]

  2. 声明一个线程集，采用下面表示法：

     t:[range {,. . .}}]

  3. 利用空括号显示空集：

     [ ]

  4. 可以利用表达式、通配符及合并线程集来指明进程和线程集。

例如：

- 下面例子包含当前进程中的前三个线程：
  t:[1,2,3]
- 下面例子采用表达式来指明线程集：
  t:[1:3+foo()]
- 下面例子指明了合并后的进程集：
  :[*] -t:[1]
- 下面例子包含当前进程中的所有线程：
  t:[*]
- 下面例子包含当前进程中除线程1和6之外的所有线程：
  t:[2:5, 7:]

- 指明进程或线程的范围

  为了指明进程或线程的一段范围，采用以下表示法：

  - *：指明所有进程或线程
  - expression：
    * 如果expression的值可以计算出或强制转换成一个整数p，那么此集只包含具有ID p的线程或进程。
    * 如果expression可以计算成一个范围r，那么集与r相同。
  - { expression } : { expression }：指明进程或线程的连续范围。
    例如：[10:12]指明具有进程号10、11和12的进程；t:[10:12]指明具有线程号10、11和12的线程。

  注意：Intel调试器忽略下限大于上限的范围。

  下限和上限都是可选的，因此可以类似下面指明：

  - :n：所有ID号不大于n的进程或线程。
    例如，[:5]代表所有进程号不大于5的进程。
  - n:：所有ID号不小于n的进程或线程。
    例如，t:[20:]代表所有线程号不小于20的线程。
  - :：所有进程或线程

13.1.3　在调试器中存储进程和线程集

在Intel调试器中利用set命令可以在调试器变量中存储进程和线程集。如：

set $set1 = [:7, 10, 15:20, 30:]

可以用print和show process set命令显示存储的值。

print $set1

如果用户没有指明集名，或指明了所有，调试器显示所有存储在调试器变量中的进程集。例如：

- 设置set2：set $set2 = [8:9, 5:2, 22:27]
- 显示set2值：show process set $set2将显示：

  $set2 = [8:9, 22:27]
- 显示所有进程集：show process set *将显示：

  $set1 = [:7, 10, 15:20, 30:]

  $set2 = [8:9, 22:27]

下面例子设置变$myset2，包含线程3、10-20、50和$myset1集：

set $myset2 = t:[3, 10:20, 50:] + $myset1

13.1.4　进程和线程集操作

Intel调试器中的进程和线程集支持下表中的操作：

操作	代表	指定动作
+	并集	作用在S1和S2集上，返回包含在S1集或包含在S2集中的元素。
-	差集	作用在S1和S2集上，返回包含在S1集，但不包含在S2集中的元素。
单目操作符-	负号	作用在集S上，返回全集与S集的差集。

13.1.5　预定义的线程集

默认，Intel调试器包含以下调试器变量集，以便使用户能方便访问几个线程集。用户可以用这些变量来定义用户自己的线程集：

$allthreads	所有存在调试线程。
$currentlockstepthreads	具有相同程序计数器作为当前线程的线程集。
$currentopenmpteam	在OpenMP中，并行区域产生一个线程组队。当当前线程是一个线程队的成员时，$currentopenmpteam是最里面队里的所有线程与当前线程。
$currentthread	当前线程。当一个事件发生时，调试器将当前线程设置为事件线程。利用thread命令可以将一个线程设置为当前线程。
$frozenthreads	当前冻结的线程
$lasteventingthread	引发最后事件的线程。调试事件可以是断点、同步点、信号发生或异常。
$uninterruptedthreads	不可终止的线程。

13.1.6 查看线程与线程集

在Intel调试器命令行上可以使用info threads命令查看线程与线程集。

13.1.7 改变当前进程集

在Intel调试器命令行上可以使用focus命令改变当前进程集。

13.2 调试多线程应用

13.2.1 在OpenMP和串行代码中寻找bug

为了判定bug是由并发导致的还是在算法内部导致的，串行执行OpenMP并行代码区域并限制这些代码区域在针对每个区域的单个线程中执行非常有用。用户可以动态串行化这些代码区域，因此无需重新编译或重启OpenMP应用。

注意：当用户在程序在并行区域执行时启用串行化，此区域并不会串行化执行，只是随后的区域将是串行化执行。当用户终止串行化，只有当前地址后的区域重新回到并行。串行化所选择的区域时，在此区域之前或之后设置断点非常有用。这将帮助用户在选择区域被执行前打开串行化，及在其它并行区域前关闭串行化。利用这种选择性串行化，应用的其余部分仍旧可以并行，这将缩短执行时间。

串行化一个OpenMP并行区域：

- 到向串行化的代码区域。
- 在此区域之前的行处和之后行处分别设置一个断点。此步帮助用户串行化此特别的代码区域。

- 运行或重新运行应用。应用将停止在第一个断点处。
- 启用串行化：输入命令*idb set openmp-serialization on*。
- 继续调试。应用停止在下一个断点处。只有单个线程执行此区域。
- 终止串行化执行：输入命令*idb set openmp-serialization off*。所有随后的OpenMP并行区域被多个线程执行，直到用户重新启用串行化。

注意：每一次用户想串行运行相同区域时，用户都必须在第一个断点处启用串行，在第二个断点处终止串行。

13.2.2　查看OpenMP信息

当调试OpenMP应用时，Intel调试器提供OpenMP的锁、队和线程信息。

为了启用OpenMP支持，必须保证调试器访问了共享库*libomp_db.so*，默认此库在编译器安装目录的*lib/intel64*或*lib/ia32*目录下。

当Intel调试器探测到是在调试OpenMP程序时，会自动启动OpenMP支持。为了关闭OpenMP支持，需要执行下面命令：

set $threadlevel=”native”

如要重新启用，则执行下面命令：

set $threadlevel=”openmp”

图形界面Intel调试器支持查看以下OpenMP应用信息：

信息	利用的窗口	使用的命令
线程	Threads Window	*idb info thread*
任务	Tasks Window	*idb info task*
栅栏（同步）	Barriers Window	*idb info barrier*
任务等待	Taskwaits Window	*idb info taskwait*
锁	Locks Window	*idb info locks*
队	Team Window	*idb info teams*
父/子关系	Spawn Tree Window	*idb info openmp thread tree*

13.2.3　线程数据共享探测

- 关于线程数据共享探测：

通常，Intel调试器探测所有不同线程重叠访问相同数据，并尝试在线程间同步。

Intel调试器支持：

- pthread同步函数
- OpenMP同步结构
- OpenMP任务和并行区域
- Intel Threading Building Blocks (Intel TBB)中同步结构子集

用户可以在原生线程、OpenMP线程以及Intel TBB和Intel Cilk Plus应用中使用的线程中使用数据共享探测。

如果Intel调试器不理解同步机制，调试器也许可以探测正确同步的线程的共享访问。在Windows和Linux系统上，可以使用压缩(Suppress)过滤器忽略此种不想要的探测。过滤器按照每个解决存储，无需在每次运行时重复产生这些过滤器。

多线程访问相同数据元素时会导致间歇性数据损坏。利用Intel调试器，可以类似部分通常调试会话一样探测和分析这些线程数据共享事件。探测到的事件显示在Thread Data Sharing Events窗口中。

- 探测线程共享事件：

  为了探测线程数据共享事件，在Intel调试器命令行中输入如下序列命令：

  - *idb sharing on*

    此命令启动探测数据共享事件。

  - *idb sharing stop on*

    此命令在当数据共享事件发生时停止调试。此方式是默认的，如没有声明*idb sharing stop off*，那么可以停止此步。

  - run

  - *idb sharing event expand*

    此命令显示数据共享探测事件的详细信息。

  Intel调试器执行仪表化（可测试）应用时，并且在数据共享事件发生时停止执行。所有在程序执行过程的线程数据共享事件或输入*idb sharing event expand*时，显示在Thread Data Sharing Events窗口中。

  注意：如用户不希望应用在数据共享事件发生时停止，请输入*idb sharing stop off*。

- 从探测中排除线程数据共享事件：

  防止调试器探测部分线程数据共享事件和将其显示在Thread Data Sharing Events窗口中非常有用，例如当事件显示为假阳性结果时。用户可以过滤此类针对不同访问类型的线程数据共享分析，并从进一步的探测中排除。

为了从进一步探测中排除数据共享事件：

1. *idb sharing filter add file filename*：忽略命名的文件的数据共享事件
2. *idb sharing filter add function function_name*：忽略命名的函数的数据共享事件
3. *idb sharing filter add range start_address, end_address*：忽略声明的地址范围内的数据共享事件
4. *idb sharing filter add variable variable [, size]*：忽略指定变量的数据共享事件

Intel调试器将从选择的类型忽略任何进一步的访问，并不探测其数据共享事件。

- 在重入调用时断开：

  重入调用(re-entrant call)发生在当不只一个线程在同一时间访问同一个表达式时。用户可以使Intel调试器在这些重入调用时断开此代码。

  输入以下命令断开：

  idb reentrancy specifier

  此命令使得在行号、函数或地址上启动重入探测。当重入探测被启用，Intel调试器在这些重入调用处断开代码执行。

  当不只一个线程同时访问指明的表达式或地址时，应用执行并停止。

13.3 调试大规模并行应用

13.3.1 总览

调试大规模并行程序的最大挑战为从调试器复制大量输出以控制并行应用进程。Intel调试器可以通过归聚相似数据成组来帮助管理这些输出。调试器利用以下两个策略归聚输出：

1. 浓缩同样的输出消息到单个输出消息。当调试器显示归聚的消息时，调试器在这些消息的前面添加进程ID的前缀。在那些范围内的进程无需连续。调试器通过将相同输出变为单个并最终输出来归聚所有进程。例如，在下面消息中，[0-41]是进程范围：

   [0-41] Intel® Debugger, Version XX

2. 具有不同十六进制的数字，但是在其它方面相同的输出，通过归聚不同的数字到一个范围。例如：在下面消息中，[0-41]是进程范围，[0;41]是值范围：

   [0-41]>2 0x120006d6c in feedback(myid=[0;41],np=42,name=0x11fffe018=”mytest”) ”mytest.c”:41

调试大规模并行应用的另外挑战为使用调试器以一致方式控制所有应用进程，或进程子集。Intel调试器通过一个单一用户接口提供这种控制。

Intel调试器在开始并行调试会话的开端：

1. 调试器探测用户应用的拓扑并附着调试器到此应用的每个进程。
2. 调试器构建n进制树，调试器作为根和叶，被称为聚合的特定进程则在根和叶中间。用户可以指定此树的分支因子和聚合时间延迟。

根调试器对应启动并行应用并作为用户的应用接口。聚合输出归聚先前描述的输出。叶调试器控制和查询用户应用的进程。

分支因子是用于构建n进制树和决定树中聚合数的因子。例如，对于16个进程：

- 用8作为分支因子产生3个聚合。
- 用2作为分支因子产生15个聚合。

用户可以在Intel调试器初始文件中设置分值因子$parallel_branchingfactor变量，其值可以从默认值8到等于或大于2。

当用户从初始文件中删除$parallel_branchingfactor时，在启动机制中的分支因子为其默认值。

聚合延迟指定了当没有接收到全部期望的消息时，在它们聚合和发送消息到下一水平之前的等待时间。

用户可以在调试器初始文件中改变聚合延迟$parallel_aggregatordelay的值，默认值为3000毫秒。

当用户从调试器初始文件中删除$parallel_aggregatordelay时，在启动机制中聚合延迟采用其默认值。

注意：

- 用户只能在启动调试器时在调试器初始文件中改变$parallel_branchingfactor和$parallel_aggregatordelay的值。在调试器启动后，用户不能改变其值。
- 默认，在树结构中，调试器利用rsh产生叶调试器和聚合进程。为了使用不同的远程shell产生这些进程，需要设置环境变量*IDB_PARALLEL_SHELL*为所需shell的路径。确保集群中每个节点具有访问所有其它集群节点的权限以便设置树结构。

13.3.2　在调试MPI应用之前

在调试之前，需要确保环境变量*IDB_HOME*设置为调试器的安装目录。

如果使用MPICH，确保mpirun_dbg.idb脚本在MPICH安装目录的*/bin/*目录下。

如果使用Intel MPI 3.0及以上，确保环境变量*MPIEXEC_DEBUG*已经定义，因此MPI进程挂起其执行等待调试器附着到它们上。

13.3.3　开始MPI调试会话

为了开始一个新的在调试器控制下的MPI作业：

- 如果用户使用MPICH，在shell中输入下面命令：

  *mpirun -dbg=idb -np number_of_processes [other_MPICH_options] *
  executable_filename [application_arguments]

- 如果用户使用Intel MPI 3.0及以上，在shell中输入下面命令：

  *mpiexec -idb -n number_of_processes [other_Intel_MPI_options] *
  executable_filename [application_arguments]

- 如果用户使用prun：

  *idb [idb_options] -parallel ‘which prun‘ -n number_of_processes *
  -N Number_of_nodes [other_prun_options] application [application_arguments]

当Intel调试器启动用户的并行应用，它探测和附着到用户应用的所有进程。在此点，用户应用在执行任何用户代码前停止执行，并且调试器显示提示符。

现在用户可以设置所需要的断点，并使用continue命令继续执行用户的应用。

13.3.4　附着到已经存在的MPI进程

注意：当前Intel调试器不支持附着到prun和Intel MPI库。

附着调试器到MPICH进程，在shell中采用如下命令：

idb -pid spawner_pid -parallelattach spawner_filename

spawner_pid是产生此作业所有进程的ID号，在Linux下，可以利用*ps -xf*命令查看此进程的ID号。spawner_filename是产生器可执行程序的名字。

13.3.5 在并行调试会话中的可用命令

用户可以使用绝大多数调试命令，就像用户调试非并行应用一样。多数命令被传递到叶调试器，用户可以在用户用户接口上看到聚合输出。然而有一些重要的例外。

所有命令被送到叶调试器以便并行调试，除了以下：

- 本地命令：在并行调试时，命令没有被送到叶调试器，但是被本地调试器使用。
- 远程和本地命令：在并行调试时，命令送到叶调试器，并也被本地调试器使用。
- 禁用命令：对并行调试禁用的命令。

下表显示的是调试器命令：只是本地、即是远程也是本地、禁用命令。

在上述列出的命令之外，focus命令可以辅助并行调试。

13.3.6 与聚合消息一起工作

根调试器从叶调试器收集输出，并聚合后输出。在大多数情形下，聚合会工作得很好。但是如果用户希望知道从特定叶调试器的准确输出时，这也许是个障碍。

为了弥补这个，调试器给每个聚合消息赋值一个唯一的ID号，并将消息存储在消息ID列表中。用户可以使用下面命令查看消息列表和展开其入口：

- show aggregated message
- expand aggregated message

13.3.7 并行调试技巧

- 获取更好聚合输出：

  如果调试器输出没有如用户期望的进行聚合，用户可以提高聚合延迟时间变量$parallel_aggregatordelay的值。此值代表的是过期时间,单位为毫秒,针对对于每个聚合器当聚合器没有收集到所有期望的消息时。因$parallel_aggregatordelay默认值为3000毫秒，用户通常不会存在聚合延迟问题。

- 同步进程：

  如果进程在调试会话中变得没有同步，例如用户对总集的子集采用focus命令，然后使用next或其它继续执行命令的情形，使得进程恢复一起运行的最简单的方式为继续执行到所有进程都要到达的某个位置。下面例子显示从进程得到的输出是如何不同的，因为在此程序中不同进程在不同的地址。使用GDB模式的until或IDB模式的cont to命令同步这些进程并聚合消息。

本地命令	本地和远程命令	禁用命令
!	export	attach
alias	output	detach
define	pwd	file
edit	quit	idb freeze
expand aggreagted message	set	idb set openmp-serialization
help	set environment	idb show openmp-serialization
history	set variable	idb stopping threads
playback input	setenv	idb synchronze
record	sh	idb target threads
set editing	shell	idb thaw
set height	show convenience	idb uninterrupt
set max-user-call-depth	show environment	load
set prompt	unset	patch
set width	unsetenv	printenv
show aggregated message	unset environment	rerun
show commands		run
show editing		set args
show height		target core
show max-user-call-depth		unload
show process set		
show prompt		
show user		
show width		
source		
unalias		
unrecord		

```
(idb) next
(idb)   [4:5,12] stopped at [int feedbackToDebugger(int, int, char*):17 0x120006bf4]
   [0:3,6:11] [3] stopped at [int feedbackToDebugger(int, int, char*):15 0x120006bf0]
   [4:5,12]    17  int pathSize = 1000;
   [0:3,6:11]    15  int i = 0;
(idb) l
(idb)   [0:3,6:11]    16  char path[1000];
   [4:5,12]    18  char hostname[1000];
   [0:3,6:11]    17  int pathSize = 1000;
   [4:5,12]    19  int hostnameSize = 1000;
   [0:3,6:11]    18  char hostname[1000];
   [4:5,12]    20
   [0:3,6:11]    19  int hostnameSize = 1000;
   [4:5,12]    21  volatile int debuggerAttached = 0;
   [0:3,6:11]    20
   [4:5,12]    22
   [0:3,6:11]    21  volatile int debuggerAttached = 0;
   [4:5,12]    23  gethostname(hostname,hostnameSize);
%3 [0:12]    [22;24]
   [0:3,6:11]    23  gethostname(hostname,hostnameSize);
   [4:5,12]    25  getcwd(path,pathSize);
   [0:3,6:11]    24
   [4:5,12]    26  strcat(path,"/");
   [0:3,6:11]    25  getcwd(path,pathSize);
   [4:5,12]    27  strcat(path,name);
   [0:3,6:11]    26  strcat(path,"/");
   [4:5,12]    28
   [0:3,6:11]    27  strcat(path,name);
   [4:5,12]    29  // Print myid pid into idbAttach.myid
   [0:3,6:11]    28
   [4:5,12]    30  sprintf(filename,"idbAttach.%d",myid);
   [0:3,6:11]    29  // Print myid pid into idbAttach.myid
   [4:5,12]    31  file = fopen(filename,"w");
   [0:3,6:11]    30  sprintf(filename,"idbAttach.%d",myid);
   [4:5,12]    32  if (file == NULL) {
   [0:3,6:11]    31  file = fopen(filename,"w");
   [4:5,12]    33    fprintf(stderr,"smg98: can't open %s for %s\n",filename, "w");
   [0:3,6:11]    32  if (file == NULL) {
   [4:5,12]    34    exit(1)
   [0:3,6:11]    33    fprintf(stderr,"smg98: can't open %s for %s\n",filename, "w");
   [4:5,12]    35  }
   [12]     36  fprintf(file," %ld %ld %s %s\n", myid, getpid(), hostname, path);
```

```
   [12]     37  fclose(file);
   [12]     38
   [4:5]     36  fprintf(file," %ld %ld %s %s\n", myid, getpid(), hostname, path);
   [0:3,6:11]    34    exit(1);
   [0:3,6:11]    35  }
   [4:5]     37  fclose(file);
   [0:3,6:11]    36  fprintf(file," %ld %ld %s %s\n", myid, getpid(), hostname, path);
   [4:5]     38
(idb) until 36
   [0:13] stopped at [int feedbackToDebugger(int, int, char*):36 0x120006cb8]
   [0:13]     36  fprintf(file," %ld %ld %s %s\n", myid, getpid(), hostname, path);
(idb) next
(idb)   [0:13] stopped at [int feedbackToDebugger(int, int, char*):37 0x120006d0c]
   [0:13]     37  fclose(file);
```

注意：所有以(idb)开始的为需要输入的命令，其余为输出，后面类似。

13.3.8 在并行调试中查找源文件

Intel调试器如果无法在应用二进制文件指定的目录或在二进制所在目录中发现源文件就无法显示源代码。

在命令行中指-I选项可以避免此问题。当利用mpirun命令开始调试会话时，此选项应该跟随-idbopt选项。

代替的，在所有叶调试器中使用use或map source directory命令可以克服此问题。

例如：

```
(idb) w
Source file not found or not readable, tried...
    ./cpi.c
    /usr/users/smith/idb-sandbox/test/src/common/Funct/bin/cpi.c
(Cannot find source file mpirun.c)
(idb) use /usr/proj/debug/idb/test/src/common/Funct/src
   [0:7] Directory search path for source files:
   [0:7] . /usr/users/smith/idb-sandbox/test/src/common/Funct/bin
       /usr/proj/debug/idb/test/src/common/Funct/src
(idb) w
   [0:7]     20
   [0:7]     21 double f(double);
   [0:7]     22
   [0:7]     23 int main(int argc, char *argv[])
   [0:7]     24 {
   [0:7]     25     int done = 0, n, myid, numprocs, i;
```

```
   [0:7]     26    double PI25DT = 3.14159265358979323846264 3;
   [0:7]     27    double mypi, pi, h, sum, x;
   [0:7]     28    double startwtime = 0.0, endwtime;
   [0:7]     29    int  namelen;
```

13.3.9 并行调试举例

下面命令使用8个进程和Intel MPI启动并行调试会话。

```
% mpiexec -idb -n 8 cpi
Intel® Debugger for applications running on Intel® 64, Version X
Attaching to program: /usr/bin/python, process 17717
Reading symbols from /usr/bin/python...(no debugging symbols found)...done.
[New Thread 182902515936 (LWP 17717)]__select_nocancel () in /lib64/tls/libc-2.3.2.so
Info: Optimized variables show as <no value> when no location is allocated.
Continuing.
MPIR_Breakpoint () at /tmp/vgusev.xtmpdir.svsmpi020.1167/mpi2.32e.svsmpi020.2008
0917/dev/src/pm/mpd/mtv.c:100
No source file named /tmp/vgusev.xtmpdir.svsmpi020.1167/mpi2.32e.svsmpi020.20080
917/dev/src/pm/mpd/mtv.c.
(idb)
```

下面是进程0到7的消息。

```
[0:7] Intel® Debugger for applications running on Intel® 64, Version X
%1 [0:7] Attaching to program: ~/test/cpi, process [17729
;17737]
   [0:7] Reading symbols from ~/test/cpi...done.
```

下面聚合消息包含不同部分的消息，2是消息ID号。在此情形下，LWP ID从进程到进程是不同的。

```
%2 [0:7] [New Thread 182908720320 (LWP [17729;17737])]
   [3,5] syscall () in /lib64/tls/libc-2.3.2.so
   [0:2,4,6:7] MPIR_WaitForDebugger () at /tmp/vgusev.xtmpdir.svsmpi020.1167/mpi
2.32e.svsmpi020.20080917/dev/src/mpi/debugger/dbginit.c:139
(idb)
   [0:7] stopped at [int main(int, char**):22 0x0000000000400ab1]
   [0:7]     22    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
(idb)
   [0:7]     18    char processor_name[MPI_MAX_PROCESSOR_NAME];
   [0:7]     19    int gate = 0;
```

```
   [0:7]     20
   [0:7]     21    MPI_Init(&argc,&argv);
   [0:7] >   22    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   [0:7]     23    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   [0:7]     24    MPI_Get_processor_name(processor_name,&namelen);
   [0:7]     25
   [0:7]     26    fprintf(stderr,"Process %d on %s\n",
(idb)
(idb) b f
(idb)
   [0:7] Breakpoint 1 at 0x400a41: file ~/test/cpi.c, line 8.
(idb) c
(idb)
   [0:7] Continuing.
   [0:7]
%3 [0:7] Breakpoint 1, f (a=[0.0050000000000000001;0.074999999999999997]) at
~/test/cpi.c:8
   [0:7] 8        return (4.0 / (1.0 + a*a));
(idb) where
(idb)
%4 [0:7] #0 0x0000000000400a41 in f (a=[0.0050000000000000001;0.074999999999999997])
%5 [0:7] #1 0x0000000000400bf3 in main (argc=1, argv=0x7fbfe7d358) at ~/test/cpi.c:52
```

下面命令设置当前进程集包含进程4、5、6和7。

```
(idb) focus [4:7]
(idb) c
(idb)
```

下面提示符显示当前进程集。

```
   [4:7] Continuing.
   [4:7]
%6 [4:7] Breakpoint 1, f (a=[0.125;0.155]) at ~/test/cpi.c:8
   [4:7] 8        return (4.0 / (1.0 + a*a));
(idb) where
(idb)
%7 [4:7] #0 0x0000000000400a41 in f (a=[0.125;0.155]) at ~/cchen
15/test/cpi.c:8
%8 [4:7] #1 0x0000000000400bf3 in main (argc=1, argv=0x7fbff7d7d8) at ~/test/cpi.c:52
(idb) focus [*]
(idb) n
```

```
(idb)
%9 [0:7] main (argc=1, argv=0x7fbff2a468) at ~/test/cpi.c
:52
   [0:7] 52                 sum += f(x);
(idb) where
(idb)
%10 [0:7] #0 0x0000000000400bf3 in main (argc=1, argv=0x7fbfe7d358) at ~/test/cpi.c:5
```

下面命令显示所有消息列表中存储的聚合消息。

```
(idb) show aggregated message
%1 [0:7] Attaching to program: ~/test/cpi, process [17729;17737]
%2 [0:7] [New Thread 182908720320 (LWP [17729;17737])]
%3 [0:7] Breakpoint 1, f (a=[0.0050000000000000001;0.074999999999999997]) at ~/test/c
%4 [0:7] #0 0x0000000000400a41 in f (a=[0.0050000000000000001;0.074999999999999997])
%5 [0:7] #1 0x0000000000400bf3 in main (argc=1, argv=0x7fbfe7d358) at ~/test/cpi.c:52
%6 [4:7] Breakpoint 1, f (a=[0.125;0.155]) at ~/test/cpi.c:8
%7 [4:7] #0 0x0000000000400a41 in f (a=[0.125;0.155]) at ~/tesast/cpi.c:8
%8 [4:7] #1 0x0000000000400bf3 in main (argc=1, argv=0x7fbff7d7d8) at ~/test/cpi.c:52
%9 [0:7] main (argc=1, argv=0x7fbff2a468) at ~/test/cpi.c:52
%10 [0:7] #0 0x0000000000400bf3 in main (argc=1, argv=0x7fbfe7d358) at ~/test/cpi.c:5
```

下面命令使用消息ID 1扩展聚合消息。

```
(idb) expand aggregated message 1
%1 [0:7] Attaching to program: ~/test/cpi, process [17729;17737]
 [3] Attaching to program: ~/test/cpi, process 17732
 [5] Attaching to program: ~/test/cpi, process 17734
 [2] Attaching to program: ~/test/cpi, process 17730
 [4] Attaching to program: ~/test/cpi, process 17733
 [0] Attaching to program: ~/test/cpi, process 17737
 [1] Attaching to program: ~/test/cpi, process 17729
 [7] Attaching to program: ~/test/cpi, process 17736
 [6] Attaching to program: ~/test/cpi, process 17735
(idb) disable 1
(idb)
(idb) c
(idb)
    [0:7] Continuing.s
pi is approximately 3.1416009869231245, Error is 0.0000083333333314
wall clock time = 120.800664
    [0:7] Program exited normally.
```

```
(idb)
(idb) quit
```

13.3.10 使用mpirun_dbg.idb启动文件

最新的MPICH发布应含有Intel调试器启动文件mpirun_dbg.idb。如果不存在，或用户使用较旧的MPICH发布，用户可以通过存储下面代码到mpirun所在的目录的mpirun_dbg.idb文件中生成。

```
#! /bin/sh
cmdLineArgs=""
p4pgfile=""
p4workdir=""
progname main=""
p4ssport=""
processedCmdLineArgs=""
#
# Extract -p4ssport info from the string passed in via -cmdlineargs.
#
function processCmdLineArgs()
{
    while [ 1 -le $# ] ; do
        arg=$1
        shift
        case $arg in
            -p4ssport)
               p4ssport="-p4ssport␣$1"
               shift
               ;;
            *)
               processedCmdLineArgs="$processedCmdLineArgs␣$arg"
               ;;
      esac
    done
}
while [ 1 -le $# ] ; do
  arg=$1
  shift
  case $arg in
     -cmdlineargs)
```

```
        cmdLineArgs="$1"
        shift
    ;;
    -p4pg)
        p4pgfile="$1"
    shift
    ;;
    -p4wd)
        p4workdir="$1"
    shift
    ;;
    -progname)
        prognamemain="$1"
    shift
    ;;
  esac
done
#
#
# Need to `eval echo $cmdLineArgs` to undo evil quoting done in mpirun.args
#
processCmdLineArgs `eval echo $cmdLineArgs`
#
if [ -n "$IDB_HOME" ] ; then
    ldbdir=$IDB_HOME
    idb=$ldbdir/idb
    if [ -f $ldbdir/idb.cat ] && [ -r $ldbdir/idb.cat ] ; then
      if [ -n "$NLSPATH" ]; then
        nlsmore=$NLSPATH
      else
        nlsmore=""
      fi
      NLSPATH=$ldbdir/$nlsmore
    fi
else
    idb="idb"
fi
#
$idb -parallel $prognamemain -p4pg $p4pgfile -p4wd $p4workdir -mpichtv $p4ssport \
$processedCmdLineArgs
```

Part IX

Intel MKL数值函数库

本系统上安装的数值函数库主要有Intel核心数学库(Math Kernel Library, MKL)，用户可以直接调用，以提高性能、加快开发。

14 Intel MKL

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*），具有intel64和ia32版本。在bash下可以通过运行*module load*选择Intel编译器时设置，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：

11.2版本：

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

15 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。

16 Intel MKL目录内容

Intel MKL的主要目录内容见表9。

17 链接Intel MKL

17.1 快速入门

17.1.1 利用-mkl编译器参数

Intel Composer XE编译器支持采用-mkl[11]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。

17.1.2 使用单一动态库

可以通过使用Intel MKL Single Dynamic Library(SDL)来简化链接行。

为了使用SDL库，请在链接行上添加libmkl_rt.so。例如

icc application.c -lmkl_rt

SDL使得可以在运行时选择Intel MKL的接口和线程。默认使用SDL链接时提供：

- 对Intel 64架构的系统，使用LP64接口链接程序；
- Intel线程。

如需要使用其它接口或改变线程性质，含使用串行版本Intel MKL等，需要使用函数或环境变量来指定选择，参见17.3.2动态选择接口和线程层部分。

[11]是-mkl，不是-lmkl，其它编译器未必支持此-mkl选项。

表 9: 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[10]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	含针对MIC架构ILP64接口的Fortran 95 .mod文件，本系统未配置MIC
<mkl_dir>/include/mic/lp64	含针对MIC架构LP64接口的Fortran 95 .mod文件，本系统未配置MIC
<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

17.1.3 选择所需库进行链接

选择所需库进行链接，一般需要：

- 从接口层(Interface layer)和线程层(Threading layer)各选择一个库；
- 从计算层(Computational layer)和运行时库(run-time libraries, RTL)添加仅需的库。

链接应用程序时的对应库参见下表。

	接口层	线程层	计算层	运行库
IA-32架构，静态链接	libmkl_intel.a	libmkl_intel_thread.a	libmkl_core.a	libiomp5.so
IA-32架构，动态链接	libmkl_intel.so	libmkl_intel_thread.so	libmkl_core.so	libiomp5.so
Intel 64架构，静态链接	libmkl_intel_lp64.a	libmkl_intel_thread.a	libmkl_core.a	libiomp5.so
Intel 64架构，动态链接	libmkl_intel_lp64.so	libmkl_intel_thread.so	libmkl_core.so	libiomp5.so

SDL会自动链接接口、线程和计算库，简化了链接处理。下表列出的是采用SDL动态链接时的Intel MKL库。参见17.3.2动态选择接口和线程层部分，了解如何在运行时利用函数调用或环境变量设置接口和线程层。

	SDL	运行时库
IA-32和Intel 64架构	libmkl_rt.so	libiomp5.so

17.1.4 使用链接行顾问

Intel提供了网页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,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。

17.1.5 使用命令行链接工具

使用Intel MKL提供的命令行链接工具可以简化使用Intel MKL编译程序。本工具不仅可以给出所需的选项、库和环境变量，还可执行编译和生成可执行程序。*mkl_link_tool*命令安装在*<mkl directory>/tools*，主要有三种模式：

- 查询模式：返回所需的编译器参数、库或环境变量等：
  - 获取Intel MKL库：*mkl_link_tool -libs [Intel MKL Link Tool options]*

  - 获取编译参数：*mkl_link_tool -opts [Intel MKL Link Tool options]*
  - 获取编译环境变量：*mkl_link_tool -env [Intel MKL Link Tool options]*

- 编译模式：可编译程序。
  - 用法：*mkl_link_tool [options] <compiler> [options2] file1 [file2 ...]*

- 交互模式：采用交互式获取所需要的参数等。
  - 用法：*mkl_link_tool -interactive*

参见http://software.intel.com/en-us/articles/mkl-command-line-link-tool。

17.2 链接举例

17.2.1 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
```

- 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
```

17.2.2 在IA-32架构上链接

在这些例子中:

- MKLPATH=$MKLROOT/lib/ia32
- MKLINCLUDE=$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。

- 使用并行Intel MKL库静态链接myprog.f:

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

- 使用并行Intel MKL库动态链接myprog.f:

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

- 使用串行Intel MKL库静态链接myprog.f:

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

- 使用串行Intel MKL库动态链接myprog.f:

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

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

```
ifort myprog.f -lmkl_rt
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/ia32
-lmkl_lapack95
-Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
```

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

```
ifort myprog.f -L$MKLPATH -I$MKLINCLUDE -I$MKLINCLUDE/ia32
-lmkl_blas95
-Wl,--start-group $MKLPATH/libmkl_intel.a $MKLPATH/libmkl_intel_thread.a
$MKLPATH/libmkl_core.a -Wl,--end-group -liomp5 -lpthread -lm
```

17.3 链接细节

17.3.1 在命令行上列出所需库链接

注意：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。

17.3.2 动态选择接口和线程层链接

SDL接口使得用户可以动态选择Intel MKL的接口和线程层。

- 设置接口层

  可用的接口与系统架构有关，对于Intel 64架构，可使用LP64和ILP64接口。在运行时设置接口，可调用mkl_set_interface_layer函数或设置*MKL_INTERFACE_LAYER*环境变量。下表为可用的接口层的值。

接口层	*MKL_INTERFACE_LAYER*的值	mkl_set_interface_layer的参数值
LP64	LP64	MKL_INTERFACE_LP64
ILP64	ILP64	MKL_INTERFACE_ILP64

如果调用了mkl_set_interface_layer函数,那么环境变量*MKL_INTERFACE_LAYER*的值将被忽略。默认使用LP64接口。

- 设置线程层

  在运行时设置线程层，可以调用mkl_set_threading_layer函数或者设置环境变量*MKL_THREADING_LAYER*。下表为可用的线程层的值。

线程层	*MKL_INTERFACE_LAYER*的值	mkl_set_interface_layer的参数值
Intel线程	INTEL	MKL_THREADING_INTEL
串行线程	SEQUENTIAL	MKL_THREADING_SEQUENTIAL
GNU线程	GNU	MKL_THREADING_GNU
PGI线程	PGI	MKL_THREADING_PGI

如果调用了mkl_set_threading_layer函数,那么环境变量MKL_THREADING_LAYER的值被忽略。默认使用Intel线程。

17.3.3 使用接口库链接

- 使用ILP64接口 vs. LP64接口

  Intel MKL ILP64库采用64-bit整数（索引超过含有$2^{31}-1$个元素的大数组时使用），而LP64库采用32-bit整数索引数组。

  LP64和ILP64接口在接口层实现，分别采用下面接口层链接使用LP64或ILP64：

  – 静态链接：libmkl_intel_lp64.a或libmkl_intel_ilp64.a
  – 动态链接：libmkl_intel_lp64.so或libmkl_intel_ilp64.so

  ILP64接口提供以下功能：

  – 支持大数据数组（具有超过$2^{31}-1$个元素）；
  – 添加-i8编译器参数编译Fortran程序。

  LP64接口提供与以前Intel MKL版本的兼容，因为LP64对于仅提供一种接口的版本低于9.1的Intel MKL来说是一个新名字。如果用户的应用采用Intel MKL计算大数据数组或此库也许在将来会用到时请选择使用ILP64接口。

  Intel MKL提供的ILP64和LP64头文件路径是相同的。

  – 采用LP64/ILP64编译

    下面显示如何采用ILP64和LP64接口进行编译：

* Fortran：
  ILP64： *ifort -i8 -I<mkl directory>/include ...*
  LP64： *ifort -I<mkl directory>/include ...*
* C/C++：
  ILP64： *icc -DMKL_ILP64 -I<mkl directory>/include ...*
  LP64： *icc -I<mkl directory>/include ...*

注意，采用-i8或-DMKL_ILP64选项链接LP64接口库时也许将会产生预想不到的错误。

– 编写代码

如果不使用ILP64接口，无需修改代码。

为了移植或者新写代码使用ILP64接口，需要使用正确的Intel MKL函数和子程序的参数类型：

整数类型	Fortran	C/C++
32-bit整数	INTEGER*4或INTEGER(KIND=4)	int
针对ILP64/LP64的通用整数（ILP64使用64-bit，其余32-bit）	INTEGER，不指明KIND	MKL_INT
针对ILP64/LP64的通用整数（64-bit整数）	INTEGER*8或INTEGER(KIND=8)	MKL_INT64
针对ILP64/LP64的FFT接口	INTEGER，不指明KIND	MKL_LONG

– 局限性

所有Intel MKL函数都支持ILP64编程，但是针对Intel MKL的FFTW接口：

* FFTW 2.x封装不支持ILP64；
* FFTW 3.2封装通过专用功能函数plan_guru64支持ILP64。

- 使用Fortran 95接口库

libmkl_blas95*.a和libmkl_lapack95*.a库分别含有BLAS和LAPACK所需的Fortran 95接口，并且是与编译器无关。在Intel MKL包中，已经为Intel Fortran编译器预编译了，如果使用其它编译器，请在使用前先编译。

17.3.4 使用线程库链接

- 串行库模式

采用Intel MKL串行（非线程化）模式时，Intel MKL运行非线程化代码。它是线程安全的（除了LAPACK已过时的子程序?lacon），即可以在用户程序

的OpenMP代码部分使用。串行模式不要求与OpenMP运行时库的兼容，环境变量*OMP_NUM_THREADS*或其Intel MKL等价变量对其也无影响。

只有在不需要使用Intel MKL线程时才应使用串行模式。当使用一些非Intel编译器线程化程序或在需要非线程化库（比如使用MPI的一些情况时）的情形使用Intel MKL时，串行模式也许有用。为了使用串行模式，请选择*sequential.*库。

对于串行模式，由于*sequential.*依赖于pthread，请在链接行添加POSIX线程库(pthread)。

- 选择线程库层

  一些Intel MKL支持的编译器使用OpenMP线程技术。Intel MKL支持这些编译器提供OpenMP技术实现，为了使用这些支持，需要采用正确的线程层和编译器支持运行库进行链接。

  – 线程层

    每个Intel MKL线程库包含针对同样的代码采用不同编译器(Intel、GNU和PGI编译器）分别编译的库。

  – 运行时库

    此层包含Intel编译器兼容的OpenMP运行时库libiomp。在Intel编译器之外，libiomp提供在Linux操作系统上对更多线程编译器的支持。即，采用GNU编译器线程化的程序可以安全地采用intel MKL和libiomp链接。

    下表有助于解释在不同情形下使用Intel MKL时选择线程库和运行时库（仅静态链接情形）：

编译器	应用是否线程化	线程层	推荐的运行时库	备注
Intel	无所谓	libmkl_intel_thread.a	libiomp5.so	
PGI	Yes	libmkl_pgi_thread.a 或libmkl_sequential.a	由PGI*提供	使 用libmkl_sequential.a从Intel MKL调用中去除线程化
PGI	No	libmkl_intel_thread.a	libiomp5.so	
PGI	No	libmkl_pgi_thread.a	由PGI*提供	
PGI	No	libmkl_sequential.a	None	
GNU	Yes	libmkl_gnu_thread.a	libiomp5.so或GNU OpenMP运行时库	libiomp5提供监控缩放性能
GNU	Yes	libmkl_sequential.a	None	
GNU	No	libmkl_intel_thread.a	libiomp5.so	
other	Yes	libmkl_sequential.a	None	
other	No	libmkl_intel_thread.a	libiomp5.so	

17.3.5 使用计算库链接

- 如不使用Intel MKL集群软件在链接应用程序时只需要一个计算库即可，其依赖于链接方式：
  - 静态链接：libmkl_core.a
  - 动态链接：libmkl_core.so
- 采用Intel MKL集群软件的计算库

  ScaLAPACK和集群Fourier变换函数(Cluster FFTs)要求更多的计算库，其也许依赖于架构。下表为列出的针对Intel 64架构的使用ScaLAPACK或集群FFTs的计算库：

函数域	静态链接	动态链接
ScaLAPACK，LP64接口	libmkl_scalapack_lp64.a和 libmkl_core.a	libmkl_scalapack_lp64.so和 libmkl_core.so
ScaLAPACK，ILP64接口	libmkl_scalapack_ilp64.a和 libmkl_core.a	libmkl_scalapack_ilp64.so和 libmkl_core.so
集群FFTs	libmkl_cdft_core.a和 libmkl_core.a	libmkl_cdft_core.so和 libmkl_core.so

下表为列出的针对IA-32架构的使用 ScaLAPACK或集群FFTs的计算库:

函数域	静态链接	动态链接
ScaLAPACK	libmkl_scalapack_core.a和libmkl_core.a	libmkl_scalapack_core.so和libmkl_core.so
集群FFTs	libmkl_cdft_core.a和libmkl_core.a	libmkl_cdft_core.so和libmkl_core.so

注意:对于ScaLAPACK和集群FFTs,当在MPI程序中使用时,还需要添加BLACS库。

17.3.6 使用编译器运行库链接

甚至在静态链接其它库时，也可动态链接libiomp5、兼容的OpenMP运行时库。

静态链接libiomp5也许会存在问题，其原因由于操作环境或应用越复杂，将会包含更多多余的库的复本。这将不仅会导致性能问题，甚至导致不正确的结果。

动态链接libiomp5时，需确保*LD_LIBRARY_PATH*环境变量设置正确。

17.3.7 使用系统库链接

使用Intel MKL的FFT、Trigonometric Transforn或Poisson、Laplace和Helmholtz求解程序时，需要通过在链接行添加-lm参数链接数学支持系统库。

在Linux系统上，由于多线程libiomp5库依赖于原生的pthread库，因此，在任何时候，libiomp5要求在链接行随后添加-lpthread参数（列出的库的顺序非常重要）。

17.3.8 冗长（Verbose）启用模式链接

如果应用调用了MKL函数，您也许希望知道调用了哪些计算函数，传递给它们什么参数，并且花费多久执行这些函数。当启用Intel MKL冗长（Verbose）模式时，您的应用可以打印出这些信息。可以打印出这些信息的函数称为冗长启用函数。并不是所有Intel MKL函数都是冗长启用的，请查看Intel MKL发布说明。

在冗长模式下，每个冗长启用函数的调用都将打印人性化可读行描述此调用。如果此应用在此函数调用中终止，不会有针对此函数的信息打印出来。第一个冗长启用函数调用将打印一版本信息行。

为了对应用启用Intel MKL冗长模式，需要执行以下两者之一:

- 设置环境变量MKL_VERBOSE为1，在bash下可以执行*export MKL_VERBOSE=1*
- 调用支持函数mkl_verbose(1)

函数调用mkl_verbose(0)将停止冗长模式。调用启用或禁止冗长模式的函数将覆盖掉环境变量设置。关于mkl_verbose函数，请参看Intel MKL Reference Manual。

Intel MKL冗长模式不是线程局域的，而是全局状态。这意味着如果一个应用从多线程中改变模式，其结果将是未定义的。

18　性能优化等

请参见Intel MKL官方手册。

Part X
应用程序的编译与安装

应用程序一般有两种方式发布：

- 二进制方式:用户无需编译,只要解压缩后设置相关环境变量等即可。如Gaussian[12]，国内用户只能购买到已经编译好的二进制可执行文件，有些国家和地区能购买到源代码。
- 源代码方式：
  - 用户需要自己编译，并且可以按照需要修改编译参数以编译成最适合自己的可执行程序，之后再设置环境变量等使用，如VASP[13]。
  - 源代码编译时经常用到的编译命令为*make*，编译配置文件为*Makefile*，请查看*make*命令用法及*Makefile*文件说明。

应用程序一般都有官方的安装说明，建议在安装前，首先仔细查看一下，比如到其主页或者查看解压缩后的目录中的类似：install*、readme*等文件。

19 二进制程序的安装

以二进制方式发布的程序，安装相对简单，一般只要解压缩后设置好环境变量即可，以Gaussian09为例：

- 将压缩包复制到某个地方，如*/opt*
- 解压缩：*tar xvf gaussian09.tar.gz*[14]
- 设置环境变量：修改*~/.bashrc*，添加：

```
##Add for g09
export g09root="/opt"
export GAUSS_SCRDIR="/tmp"
. $g09root/g09/bsd/g09.profile
##End for g09
```

- 刷新环境设置：*. ~/.bashrc*或重新登录下。

[12]Gaussian主页：http://www.gaussian.com/
[13]VASP主页：http://www.vasp.at/
[14]当前主流Linux系统，*tar*命令已经能自动识别.gz和.bz压缩，无需再明确添加z或j参数来指定。

20 源代码程序的安装

以源代码发布的程序安装相对复杂，需了解所采用的编译环境，并对配置等做相应修改（主要修改编译命令、库、头文件等编译参数等）。以VASP为例：

- 查看安装说明:其主页上的文档:http://www.vasp.at/index.php/documentation
- 解压缩文件：
  - *tar xvf vasp.5.lib.tar.gz*
  - *tar xvf vasp.5.2.tar.gz*
- 查看说明：install、readme文件等，VASP解压后的目录中未含有，可以参见上述主页文档安装。
- 生成默认配置文件：*./configure*。
  - VASP不需要*./configure*命令生成*Makefile*，而是提供了几个针对不同系统和编译器的makefile模板，可以复制*makefile.linux_ifc_P4*成*Makefile*
  - 其它程序也许需要*./configure*生成所需要的*Makefile*，在运行*./configure*之前，一般可以运行*./configure -h*查看其选项。如对Open MPI 1.6.4，可以运行以下命令生成*Makefile*：

    F77=ifort FC=ifort CC=icc CXX=icpc ./configure --prefix=/opt/openmpi-1.6.4

    其中：
    * F77：编译Fortran77源文件的编译器命令
    * FC：编译Fortran90源文件的编译器命令
    * CC：编译C源文件的编译器命令
    * CXX：编译C++源文件的编译器命令
    * –prefix：安装到的目录前缀

    另外一些在Makefile中常见变量为：
    * CPP：预处理参数
    * CLAGS：C程序编译参数
    * CXXFLAGS：C程序编译参数
    * F90：编译Fortran90及以后源文件的编译器命令
    * FFLAGS：Fortran编译参数
    * OFLAG：优化参数
    * INCLUDE：头文件参数
    * LIB：库文件参数

    * LINK：链接参数

- 修改*Makefile*文件配置，设定编译环境等：
  - 对*vasp.5.lib/Makefile*做如下修改：
    * 设定编译Fortran的编译器命令为Intel Fortran编译器命令：FC=ifort
  - 对*vasp.5/Makefile*做如下修改：
    * 设定BLAS库使用Intel MKL中的BLAS：BLAS=-mkl[15]
    * 打开FFT3D支持：去掉FFT3D = fft3dfurth.o fft3dlib.o前的#[16]
    * 设定MPI Fortran编译器为Intel MPI编译器：FC=mpiifort
- 编译：*make*
  - 先在*vasp.5.lib*目录中执行*make*
  - 如未出错，则再在*vasp.5.2*目录中执行*make*
- 安装：*make install*。VASP不需要，有些程序需要执行此步。
- 设置环境变量：比如在*~/.bashrc*中设置安装后的可执行程序目录在环境变量*PATH*中：

```
export PATH=$PATH:/opt/vasp.5.2
```

[15]因为2013版本的Intel编译器支持-mkl选项自动Intel MKL库，因此可以这么设置。
[16]在Makefile中#表示注释

Part XI

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。

21 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。

22 查看队列情况：bqueues

用户在使用时，首先需要了解哪些队列可以使用，利用*bqueues*可以查看现有队列信息。

具体队列会根据需要更改，请注意登录系统后的提示，或运行*bqueues -l*查看。

bqueues

将输出：

```
QUEUE_NAME PRIO STATUS    MAX  JL/U  JL/P  JL/H NJOBS PEND RUN  SUSP
long       59  Open:Active 1200 1200  -     -     0     0     0     0
normal     58  Open:Inact  1440 192   -     -   360     0   360     0
small      57  Closed:Active 720 120  -     -  1976  1476   500     0
```

其中，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：队列开放，可以接受提交新作业
  - Active：队列已激活，队列中未开始运行的作业可以开始运行
  - Closed：队列已关闭，不接受提交新作业
  - Inact：队列未激活，可接受提交新作业，但队列中的等待运行的作业不会开始运行
- MAX：队列对应的最大作业槽数（Job Slot，一般与CPU核数一致，以下通称CPU核数），-表示无限
- JL/U：单个用户同时可以使用的CPU核数
- JL/P：每个处理器可以接受的CPU核数
- JL/H：每个节点可以接受的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
- RSV：为排队作业预留的CPU核数

22.1 查看队列详细情况：bqueues -l

队列策略也许会调整，利用*bqueues -l [队列名]*查看各队列的详细情况：

```
QUEUE: small
  -- Maximum number of job slots that each user can use in this queue is 720.
   on nodes: node1~node67

PARAMETERS/STATISTICS
PRIO NICE STATUS       MAX JL/U JL/P JL/H NJOBS PEND RUN SSUSP USUSP RSV
 57   20  Open:Active   720 120    -    - 1964 1464  500     0     0    0
Interval for a host to accept two jobs is 0 seconds

 PROCLIMIT
 48

SCHEDULING PARAMETERS
          r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched  -     -     -     -       -     -    -     -     -      -      -
 loadStop   -     -     -     -       -     -    -     -     -      -      -

         adapter_windows   poe nrt_windows
 loadSched           -      -           -
 loadStop            -      -           -

SCHEDULING POLICIES: EXCLUSIVE

USERS: paiduigroup/ scc_group/
HOSTS: node1 node2 node3 node4 node5 node6 node7 node8 node9 node10 node11 node12
RES_REQ: span[ptile=24]
```

- QUEUE：队列名，跟着的下一行是描述
- PRIO：优先值，越大越优先
- NICE：作业运行时的nice值，即作业运行时的操作系统调度优先值，从-20到19，越小越优先
- RUNLIMIT：作业单CPU运行时间限制，以系统中的某个节点为基准，如为1440.0 min则表示可以运行一天（60*24）
- CPULIMIT：单个作业运行时间限制，以系统中的某个节点E5410作为参考，运行机时（核数*墙上时间）为345600.0 CPU分钟，即如用8 CPU核计算，允许运行30天

- PROCLIMIT：单个作业核数限制，2␣4␣8，表示使用此队列时，最少使用2个核，最多使用8核，如提交时没用-n指定具体核数，则使用默认4核
- PROCESSLIMIT：单个作业最大核数限制，为8
- MEMLIMIT：每个进程能使用的内存数，默认以KB为单位
- THREADLIMIT：作业最大线程数
- USERS：有权使用的用户
- HOSTS：对应的节点

23 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。
- status列：
  - ok：表示可以接受新作业，只有这种状态可以接受新作业
  - closed：表示系统在运行，但已被调度系统关闭，不接受新作业
  - locku：表示在进行排他性运行
  - busy：表示负载超过限定
  - unavail、-ok：作业调度系统服务有问题
- r15s、r1m、r15m列：分别表示15秒、1分钟、15分钟平均负载
- ut：利用率
- tmp：*/tmp*目录大小
- swp：swp虚拟内存大小

- mem：内存大小
- io：硬盘读写（-l选项时才出现）

查看node2节点：*lsload node2*

24 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：表示可以接收新作业，只有这种状态可以接受新作业
  - closed：表示已被作业占满，不接受新作业
  - unavail和unreach：系统停机或作业调度系统服务有问题
- 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。

查看node1节点：*bhosts node1*

*bhosts -l*会显示节点详细信息，其中slots表示目前最大可以接受作业槽数（默认一般与CPU核数一致）

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21作业运行条件。

25 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。

26 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，必须在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。

注意：

- 作业提交后，应经常检查一下作业的CPU、内存等利用率，判断实际运行效率：
  - 可以*ssh*到对应运行作业的节点运行*top*命令；
  - 查看Ganglia系统监控：http://scc.ustc.edu.cn/ganglia。
- 请不要ssh到节点后直接运行作业，以免影响作业调度系统分配到此节点的作业。

26.1 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。

如提交不成功，则显示相应提示。

26.2 提交串行作业：bsub -n 1

我校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，有些系统不允许运行串行作业。

运行串行作业，请使用支持串行队列的队列（假如serial队列支持串行），比如：

bsub -q serial -n 1 executable-serial

26.3 指明所需要的CPU核数：bsub -n

利用*-n min_proc[,max_proc]*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24个CPU核（由*-n 24*指定）运行MPI（由*mpijob*指明为运行MPI程序）程序：

bsub -q small -n 24 mpijob executable-mpi1

最少采用24最大采用72 CPU核数运行：*bsub -n 24,72*

当作业调度尝试开始运行时：

- 如空闲资源满足72 CPU核，那么将采用72 CPU核运行；
- 如空闲资源不满足72 CPU核，但满足24 CPU核，则用24 CPU核运行。

由于每个节点的CPU核数为24，建议单个作业所使用的核数最好为24或12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。以上仅仅是建议，具体申请核数应考虑作业实际情况。即使同一个计算软件，在计算不同条件时，也有可能不一样，请务必仔细研究自己所使用的软件。

26.4 提交到特定节点：bsub -m “host_name”

利用*-m*选项可以指定作业在特定节点上运行，如：

bsub -m ‘‘node1 node3’’

如非必要，建议不要添加此选项，以免导致作业无法及时运行。

26.5 提交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。
- 如您了解所使用的MPI环境与LSF的配合,也可不用*mpijob*提交,如直接用*mpirun*等命令运行。

26.6 指明需要某种资源作业提交：bsub -R

*-R ‘‘res_req’’ [-R ‘‘res_req’’ ...]*可以使得作业在需要满足某种条件的节点上运行，如：

- *-R ‘‘span[hosts=1]’’*：指定需要在同一个节点内运行；
- *-R ‘‘span[ptile=8]’’*:指定需要在每一个节点内运行多少核,如*bsub -n 16 -R ‘‘span[ptile=8]’’*则会分给2个节点，每个节点8核；

- *-R ‘‘1*{mem>5000} + 9*{mem>1000}’’*：一个CPU核需要至少5GB内存，另外9个每个至少需要1GB内存。

26.7 提交OpenMP等共享内存作业:bsub -R “span[hosts=1]” OMP_NUM_THREAD

由于只能在同一个节点内部运行OpenMP共享内存的作业，如Gaussian程序，此时需要添加利用*-R ‘‘span[hosts=1]’’*参数指定使用一个节点，并用OMP_NUM_THREADS设定指定的线程数，一般应与申请的核数一致[17]：

指定利用8 CPU核运行OpenMP程序：

bsub -q normal -n 8 -R ‘‘span[hosts=1]’’ OMP_NUM_THREADS=8 executable-omp1

26.8 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。

26.9 给作业起个名字：bsub -P project_name

提交时可以利用*-P*选项给作业起个名字方便查看，如：

bsub -P VASPJOB

26.10 运行排他性作业：bsub -x

如需要独占节点运行，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CPU核，也将按照此节点内的所有CPU核数进行机时计算。

[17]有些程序可以在输入文件中设置，那么可以不填加OMP_NUM_THREADS

26.11 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

*-o*和*-e*及变种*-oo*和*-eo*：

- -o：如日志原文件存在，正常屏幕输出将追加到原文件后
- -oo：如日志原文件存在，正常屏幕输出将覆盖原文件
- -e：如日志原文件存在，出错时屏幕输出将追加到原文件后
- -eo：如日志原文件存在，出错时屏幕输出将覆盖原文件

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。

26.12 指明输出目录提交：bsub -outdir output_directory

默认情况下，屏幕输出等存放在提交作业时的目录下，如想将其放到其它目录可以采用*bsub -outdir output_directory*方式，如：

bsub -outdir ‘‘%U/%J_%I’’ myprog

结合以下常用变量，可以方便结合作业号等设置输出目录及日志文件名等：

- %J：作业号
- %JG：作业组
- %I：作业组中的索引
- %EJ：执行作业号
- %EI：作业组中的执行作业索引
- %P：作业名
- %U：用户名
- %G：用户组名

26.13 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

26.14 满足依赖关系运行作业：bsub -w

利用*-w*选项可以使得新提交的作业在满足一定条件时才运行，比如与其它作业的关联：

- done(job_ID |“job_name” ...)：作业结束时状态为DONE时运行
- ended(job_ID | “job_name”)：作业结束时状态为DONE或EXIT时运行
- exit(job_ID | “job_name” [,[operator] exit_code])：作业结束时状态为EXIT，且退出代码满足一定条件时运行
- external(job_ID | “job_name”, “status_text”)：作业状态变为某状态时运行，如变为SUSP
- 支持的条件之间的条件表达式：&&（和）、||（或）、!（否）
- 支持的条件内的条件算子：>、>=、<、<=、==、!=

如：*bsub -w ‘‘done(1456)’’*

26.15 指定时间运行：bsub -b time

利用*-b [[year:][month:]day:]hour:minute*可以使得新提交的作业在特定时间运行，系统会预留资源给此作业，如果在时间到达时所需资源满足则会运行，不满足则继续等待。如：

bsub -b 2016:01:09:09:20

26.16 指定运行时长：bsub -W time

利用*-W [hour:]minute*可以使得提交的作业在运行超过设定时长后终止，如：

bsub -W 1:30

26.17 在运行前执行特定命令：bsub -E “pre_command”

利用*-E ‘‘pre_exec_command [arguments ...]’’*可以使得作业在运行前，在所分配的节点上运行特定命令。

如利用modinput.sh此修改程序输入参数：*bsub -E ‘‘./modinput.sh 10’’ -n 2 exec1*

26.18 在运行后执行特定命令：bsub -Ep “post_command”

利用*-Ep ‘‘post_exec_command [arguments ...]’’*可以使得作业在运行结束时，在所分配的节点上运行特定命令。

如利用此删除core文件：*bsub -Ep ‘‘bin/rm -f core.*’’ -n 2 exec1*

26.19 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OpenMP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

27 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
```

28 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。

29 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.
```

30 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.
```

31 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.
```

32 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.
```

33 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。

34 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。

Part XII

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