中国科学技术大学超级计算中心

ChinaGrid高性能计算集群使用指南

李会民

2016年5月27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ChinaGrid高性能计算集群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指南。

为了便于查看，主要排版约定如下：

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

```
export OPENMPI=/opt/openmpi-1.6.4_intel-13.1.0.146
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。请从超算中心主页上下载更新后的手册。

Part II

ChinaGrid高性能计算集群简介

中国科学技术大学超级计算中心ChinaGrid高性能计算集群,主要由浪潮NF5260M3机架式服务器、浪潮NX5440刀片服务器、曙光TC4600刀片服务器、联想B710刀片服务器构成，共计一个管理和用户登录节点、一个存储节点及44个计算节点，其中计算节点为704个CPU核心，总双精度峰值计算能力为每秒14.64万亿次。具体参数为：

- 管理和用户登录节点：
  - 用户主登录节点，可以进行编译与通过作业调度系统提交作业。禁止直接在此节点上运行作业。
  - 浪潮NF5260M3机架式服务器一台，两颗64位主频2.60GHz的Intel Xeon E5-2670 x86_64 8核CPU，共16核，32GB内存。
  - 节点名为chinagrid。
- 存储节点：浪潮SA5212H2机架式存储服务器，6块2TB SATA硬盘，可用空间9TB。
- 刀片计算节点：
  - 各节点配置：两颗64位主频2.60GHz的Intel Xeon E5-2670 x86_64 8核CPU（共16核）、32GB内存及300GB SAS硬盘。
  - 节点名：node1-node44
    * node1-node10：浪潮NF5260M3机架式服务器，10台
    * node11-node20：浪潮NX5440刀片服务器，10台
    * node21-node30：联想B710刀片服务器，10台
    * node31-node44：曙光TC4600刀片服务器，14台
- 计算网络：56Gbps InfiniBand高速网。
- 管理网络：千兆以太网。
- 操作系统：x86_64架构的64位CentOS 6.5 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: ChinaGrid高性能计算集群

Part III

用户登录与文件传输

本系统的操作系统为x86_64的64位CentOS 6.4 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、SSH Secure Shell Client等支持SSH协议的客户端软件[1]）登录到用户登录节点（节点名chinagrid）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，或联系管理员解封。

用户可以在登录节点上运行*passwd*命令修改密码（利用*passwd*命令在登录节点之外的节点上修改密码无效）。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。

用户登录进来的默认语言环境为zh_CN.UTF-8中文，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[2]：

- 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）。

账户开设时，默认最大可用50GB磁盘存储空间。用户请及时清除不需要的文件，以便释放空间。运行*quota␣-s*命令可以查看当前自己的磁盘空间使用情况，*du␣-hs 目录*可以查看目录占用的空间。如需要更大存储空间，请与管理人员联系。

超算中心无法提供数据备份服务，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。

[1]客户端下载：http://scc.ustc.edu.cn/yhsq/dlrjxz/

[2]PuTTY客户端中文设置：

- http://scc.ustc.edu.cn/yhsq/dlrjxz/200910/t20091014_13029.html

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设置了只能访问超算服务器，无法访问校内外其它资源。

本超算系统采用x86_64的64位CentOS 6.4 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*␣命令或者命令加-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

串行及OpenMP程序编译及运行

在本系统上可运行C/C++、Fortran的串行程序，以及与OpenMP和MPI结合的并行程序。用户只需在登录节点(chinagrid)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。

系统当前设置为默认使用Intel 13.1.0.146 C/C++/Fortran编译器与Intel MPI 4.1.0的组合，用户也可以单独设置自己所需的串行编译环境（如在*~/.bashrc*中设置），也可直接运行*mpi-selector-menu*[3]命令按照提示选择自己使用的MPI和串行编译环境，注意数字后需要加u，设置完成后最好重新登录以便设置生效：

```
Current system default : intel -mpi-4.1.0.030 _intel -compiler-13.1.0.146
Current user default :   <none>

    ”u” and ”s” modifiers can be added to numeric and ”U”
    commands to specify ”user” or ”system-wide”.

1. intel -mpi-4.0.3.008 _intel -compiler-13.0.1.117
2. intel -mpi-4.1.0.030 _intel -compiler-13.0.1.117
3. intel -mpi-4.1.0.030 _intel -compiler-13.1.0.146
4. openmpi-1.6.3_intel -compiler-13.0.1.117
5. openmpi-1.6.3_intel -compiler-13.1.0.146
6. openmpi-1.6.4_gcc-4.4.7
7. openmpi-1.6.4_intel -compiler-13.1.0.146
8. openmpi-1.6.4_pgi-10.6
U. Unset default
Q. Quit

Selection (1-8[us], U[us], Q):
```

注意：在*~/.bashrc*中设置的级别要高于使用*mpi-selector-menu*设置的。

本节主要介绍串行程序和OpenMP并行程序的编译，MPI并行程序的编译将在后面介绍。

[3]此命令虽然是针对MPI设计的，但设置后也会更改串行编译环境。

1 C/C++/Fortran编译器简介

1.1 Intel C/C++ Fortran编译器

Intel 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。

系统已经安装的是Intel Parallel Studio XE 2013带有的，并设置默认使用64位的13.1.0.146版本的Intel编译器，安装目录为*/opt/intel/composer_xe_2013.2.146*，用户直接使用即可，无需自己设置。

系统还安装有64位13.0.1.117版本的编译器,安装在*/opt/intel/composer_xe_2013.1.117*，用户如想使用，可以在自己的*~/.bashrc*之类环境设置文件中添加代码（␣表示空格）：

```
../opt/intel/composer_xe_2013.1.117/bin/compilervars.sh␣intel64
```

或者运行*mpi-selector-menu*命令设置。

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。

官方手册目录：

- 13.1.0.146版本：*/opt/intel/composer_xe_2013.2.146/Documentation*。
- 13.0.1.117版本：*/opt/intel/composer_xe_2013.1.117/Documentation*。

1.2 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2010 v10.6，安装在*/opt/pgi/linux86-64/10.6*。

用户要使用，可以运行*mpi-selector-menu*命令设置，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/linux86-64/10.6/bin:$PATH
MANPATH=$MANPATH:/opt/pgi/linux86-64/10.6/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*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：*/opt/pgi/linux86-64/10.6/doc*。

1.3 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。

2 C/C++程序的编译

本节主要介绍C/C++源程序的常用编译方式。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。

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。

2.2 Intel C/C++编译器重要编译选项

- -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。
- -c：仅编译成目标文件（.o文件）。
- -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。
- -g：包含调试信息。
- -ip：在单个文件中进行过程间优化(Interprocedural Optimizations-IPO)。

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

- -ipo[n]：在多文件中进行过程间优化，非负整数n为可生成的目标文件数。
- -I<头文件目录>：指明头文件的搜索路径。
- -L<库目录>：指明库的搜索路径。
- -mkl<lib>：链接时自动链接Intel MKL库，默认为不起用。<lib>可以为：
  - parallel：采用线程化部分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。
- -l<库文件>：指明所需链接的库名，如库名为libxyz.a，则可用-lxyz指定。
- -o file：指定生成的文件名。
- -openmp：编译OpenMP程序。注意：在本系统上只能在同一个节点内的CPU上运行OpenMP程序，提交作业时请结合相应选项，以保证在同一个节点运行。
- -O<级别>：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。
- -p：进行概要导向优化(Profile Guided Optimization-PGO)。
- -shared: 生成共享目标而不是可执行文件, 必须在编译每个目标文件时使用-fpic选项。
- -static ： 静态链接所有库。
- -std=<标准>：标准可以为c89、c99、gnu89、gnu++98或c++0x，分别对应相应标准。
- -w：编译时不显示任何警告，只显示错误。
- -wall：编译时显示所有警告。
- -x␣<类型>：类型可以为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。
- -Xhost：告诉编译器以编译时主机的最高指令集指令编译。

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

2.3 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*pgCC*命令有稍微不同，建议仔细查看PGI相关资料。

- 一般选项：
  - -#：显示编译器、汇编器、链接器的调用信息。
  - -c：仅编译成目标文件（.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。
- 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目标文件（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。

- 调试选项：
  - -g：包含调试信息。

- 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<头文件目录>：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。

• 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<库文件>：指明所需链接的库名。如库为libxyz.a，则可用-lxyz指定。
- -L<库目录>：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共享目标的路径。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目标文件。
- -R<directory>：对共享目标文件总搜索directory目录。
- -pgf90libs：链接时添加pgf90运行库，以允许混合编程。
- -shared：生成共享目标而不是可执行文件，必须在编译每个目标文件时使用-fpic选项。
- -soname<name>：生成共享目标时，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。

• 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。

- 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目标小于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nehalem-64等，默认与编译时的平台一致。

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。本系统采用的是Intel Xeon E5-2670 CPU，需要仔细选择，特别是保证结果的正确性。

2.4 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。

- 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决定。

  - -c：仅编译成目标文件（.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。

- 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=：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。

- 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。

- 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。

- 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。

- 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<头文件目录>：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。

- 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：生成共享目标而不是可执行文件，必须在编译每个目标文件时使用-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<头文件目录>：指明头文件的搜索路径。
  - -l<库文件>：指明所需链接的库名，如库为libxyz.a，则可用-lxyz指定。
  - -L<库目录>：指明库的搜索路径。
  - -B<路径>：设置寻找可执行文件、库、头文件、数据文件等路径。

- 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。
- 约定成俗的选项：
  - -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。

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。

2.5 C/C++程序编译举例

- Intel C/C++编译器编译举例：
  - *icc -o yourprog yourprog.c*
    将C程序yourprog.c编译为可执行文件yourprog。
  - *icpc -o yourprog yourprog.cpp*
    将C++程序yourprog.cpp编译为可执行文件yourprog。
  - *icc -o yourprog-omp -openmp yourprog.c*
    将OpenMP指令并行的C程序yourprog-omp.c编译为可执行文件yourprog-omp。
- PGI C/C++编译器编译举例：
  - *pgcc -o yourprog yourprog.c*
    将C程序yourprog.c编译为可执行文件yourprog。
  - *pgCC -o yourprog yourprog.cpp*
    将C++程序yourprog.cpp编译为可执行文件yourprog。
  - *pgcc -o yourprog-omp -mp yourprog.c*
    将OpenMP指令并行的C程序yourprog-omp.c编译为可执行文件yourprog-omp。

- GNU C/C++编译器编译举例：
  - *gcc -o yourprog yourprog.c*
    将C程序yourprog.c编译为可执行文件yourprog。
  - *g++ -o yourprog yourprog.cpp*
    将C++程序yourprog.cpp编译为可执行文件yourprog。
  - *gcc -o yourprog-omp -fopenmp yourprog.c*
    将OpenMP指令并行的C程序yourprog-omp.c编译为可执行文件yourprog-omp。

3 Fortran程序的编译

3.1 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。

3.2 Intel Fortran编译器重要编译选项

Intel编译器选项非常多，下面仅仅是列出一些本人认为常用的编译Fortran程序的*ifort*编译命令重要选项，建议仔细看相关资料。

- -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。
- -Bdynamic：运行时动态链接所需要的库。
- -Bstatic：静态链接用户生成的库。
- -c：仅编译成目标文件（.o文件）。
- -convert [关键字]:转换无格式数据的类型,比如关键字为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。
- -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,选项size可为72、80和132。也可直接用-72、-80和-132指定，默认为72字符。

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

- -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。
- -fixed：指明Fortran源代码为固定格式，默认由文件后缀决定格式类别。
- -fpic：生成地址无关代码，当编译成共享目标文件时必须使用此选项，等价于-fPIC，默认为-fno-pic。
- -free：指明Fortran源程序为自由格式，默认由文件后缀决定格式类别。
- -g：包含调试信息。
- -ip：在单个文件中进行过程间优化(Interprocedural Optimizations-IPO)。
- -ipo[n]：在多文件中进行过程间优化，非负整数n为可生成的目标文件数。
- -I<头文件目录>：指明头文件的搜索路径。
- -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。
- -L<库目录>：指明库的搜索路径。
- -l<库文件>：指明所需链接的库名，如库文件为libxyz.a，则可用-lxyz指定。
- -mkl<lib>：链接时自动链接Intel MKL库，默认为不起用。<lib>可以为：
  - parallel：采用线程化部分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。
- -nofree：指明Fortran源程序为固定格式。
- -openmp：编译OpenMP指令并行程序，注意：在本系统上只能在同一个节点内CPU上运行OpenMP程序，提交作业时请结合相应选项，以保证在同一个节点上运行。
- -O<级别>：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。
- -p：进行概要导向优化(Profile Guided Optimization-PGO)。
- -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。

- -shared: 生成共享目标而不是可执行文件，必须在编译每个目标文件时使用-fpic选项。
- -stand <标准>：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。标准可为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<标准>，此时标准不带f，可为03、90、95。
- -static：静态链接所有库。
- -unroll[n]：循环最大可展开的层数，与性能相关。
- -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。
- -w：编译时不显示任何警告，只显示错误。
- -wall：编译时显示所有警告。
- -X：编译时不用默认的头文件搜索目录，与-I结合可使用指定的头文件目录。
- -Xhost：告诉编译器以编译时主机的最高指令集指令编译。

建议仔细查看编译器手册中关于程序优化的部分，特别是IPO、PGO和HLO部分，多加测试，选择适合自己程序的编译选项以提高性能。本系统采用的是Intel Xeon E5-2670 CPU，建议参考本CPU相关资料，并仔细选择参数。

3.3 PGI Fortran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编译Fortran 9x程序的*pgf90*重要选项，编译Fortran 77程序的*pgf77*等编译命令也许有部分不同，建议仔细看PGI相关资料。

- 一般选项：
  - -#：显示编译器、汇编器、链接器的调用。
  - -c：仅编译成目标文件（.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参数。
  - -help[=option]：显示帮助信息，option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继续进行编译；默认为显示错误信息并终止编译。
- -o file：指定生成的文件名。
- -show：显示现有pgf90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，显示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。

• 优化选项：

- -fast：编译时选择针对目标平台的普通优化参数，用pgf90 -fast -help可以查看等价的开关。优化级别至少为O2，参看-O参数。
- -fastsse：对支持SSE和SSE2指令的CPU（如Opteron）编译时选择针对目标平台的普通优化参数，用pgcc -fastsse -help可以查看等价的开关。优化级别至少为O2，参看-O参数。
- -fpic或-fPIC：编译器生成地址无关代码，以便可以用于生成共享目标文件（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,…]]：显示有用的信息到标准错误输出，选项可以为all、autoinline、inline、ipa、loop或opt、mp、time或stat。
- -Mipa [=option[,option,…]]和-Mnoipa：对过程间分析启用和指定参数，默认为-Mnoipa。
- -Mneginfo=option[,option…]：使编译器生成关于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以收集运行时的统计信息以用于子序列的编译中。-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：启动概要导向优化，此时必须在当前目录下存在概要文件pgfi.out。
- -Mprof[=option[,option,…]]：设置性能功能概要选项。用此选项可使结果执行生成性能概要，以便PGPROF性能概要器可以分析。

  - -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。

- 调试选项：
  - -g：包含调试信息。

- 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name为def。
  - -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，但只显示源文件中已定义的宏，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<头文件目录>：指明头文件的搜索路径。
  - -M：显示make的依赖关系到标准输出。
  - -MD：显示make的依赖关系到文件file.d，其中file是编译文件的根名字。
  - -MM：显示make的依赖关系到标准输出，但忽略系统头文件。
  - -MMD：显示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>：预处理去除时name的初始定义。

- 链接选项：
  - -Bdynamic：运行时动态链接所需的库。
  - -Bstatic：静态链接所需的库。
  - -Bstatic_pgi：对动态链接系统库时静态链接PGI库。
  - -g77libs：允许链接GNU g77或gcc生成的库。
  - -l<库文件>：指明所需链接的库名，如库为libxyz.a，则可用-lxyz指定。
  - -L<库目录>：指明库的搜索路径。
  - -m：显示链接拓扑。

- -Mrpath和-Mnorpath：默认为-rpath，以设置包含PGI共享目标的路径。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目标文件。
- -R<directory>：对共享目标文件总搜索directory目录。
- -pgf90libs：链接时添加pgf90运行库，以允许混合编程。
- -shared：生成共享目标而不是可执行文件，必须在编译每个目标文件时使用-fpic选项。
- -soname<name>：生成共享目标时，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。

• 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主程序的目标文件。
- -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。

- 平台相关选项：
  - -Kieee和-Knoieee：浮点操作是否严格按照IEEE 754标准，默认为不。使用-Kieee时一些优化处理被禁止，并且使用更加精确的数值库，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。选项可以为divz、fp、align、denorm、inexact、inv、none、ovf、unf，默认为none。
  - -Mlarge_arrays和-Mnolarge_arrays：是否允许数组大于2GB，默认不允许。当使用-mcmodel=medium时暗含-Mlarge_arrays选项。
  - -mcmodel=small|medium：使用内存模型是否限制目标小于2GB(small)或允许数据块大于2GB(medium)，medium时暗含-Mlarge_arrays选项。
  - -Msecond_underscore和-Mnosecond_underscore：是否对已经有_的Fortran函数名添加第二个_。主要在与*g77*兼容时使用，*g77*默认给符号添加第二个_。
  - -Mvarargs和-Mnovarargs：是否生成从Fortran调用C程序时用变量参数调用序列，默认为否。
  - -tp target：target可以为nehalem-64等，默认与编译时的平台一致。

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。本系统采用的是Intel Xeon E5-2670 CPU，需要仔细选择，特别是保证结果的正确性。

3.4 GNU Fortran编译器重要编译选项

GNU Fortran编译器是Linux系统自带的Fortran编译器，系统安装的版本为4.4.7，支持大部分*gcc*选项，下面仅仅是列出一些针对4.4.7的*gfortran*本人认为常用的重要选项，建议仔细看GNU Fortran和gcc的相关资料。

- 控制Fortran语言类型的选项：
  - -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。
  - -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。
  - -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。
  - -fimplicit-none：禁止变量的隐式声明，所有变量都需要显式声明。
  - -fcray-pointer：支持Cray指针扩展。
  - -fopenmp：编译OpenMP并行程序。
  - -std=<std>：指明Fortran标准，std可以为f95、f2003、legacy。
  - -M<dir>和-J<dir>：指定编译时保存生成的模块文件目录。
  - -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。
- 一般选项：
  - -c：仅编译成目标文件（.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。
- 警告选项：

  - -fsyntax-only：仅仅检查代码的语法错误，并不进行其余操作。
  - -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。

- 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。

- 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。

- 预处理选项：
  - -C：保留预处理的C源文件中的注释。
  - -D name：在预处理中定义宏name的值为1。
  - -D name=def：在预处理中定义name为def。
  - -U name：去除预处理中的任何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。
  - -dM：显示预处理时源文件中定义的宏及值，含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：预处理各文件，将结果发给标准输出，不进行编译、汇编或链接。
  - -I<头文件目录>：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，但忽略系统包含。
  - -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。

- 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：生成共享目标而不是可执行文件，必须在编译每个目标文件时使用-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，强制连接一个库模块来定义它。
  - -I<头文件目录>：指明头文件的搜索路径。
  - -l<库文件>：指明所需链接的库名，如库为libxyz.a，则可用-lxyz指定。
  - -L<库目录>：指明库的搜索路径。
  - -B<路径>：设置寻找可执行文件、库、头文件、数据文件等路径。

- Intel 386和AMD 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。

- 约定成俗的选项：
  - -fno-automatic：将每个程序单元的本地变量和数组声明具有SAVE属性。
  - -ff2c：与*g77*和*f2c*命令生成的代码兼容。
  - -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。
  - -funderscoring：对外部函数名没有_的加_，以与一些Fortran编译器兼容。
  - -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。
  - -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。
  - -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。

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。

3.5 Fortran程序编译举例

- Intel Fortran编译器编译举例：
  - *ifort -o yourprog yourprog.for*
    将Fortran 77程序yourprog.for编译为可执行文件yourprog。
  - *ifort -o yourprog -static yourprog.f90*
    将Fortran 90程序yourprog.f90静态编译为可执行文件yourprog。
  - *ifort -o yourprog-omp -openmp yourprog.f90*
    将OpenMP指令并行的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp。
- PGI Fortran编译器编译举例：
  - *pgf77 -o yourprog yourprog.for*
    将Fortran 77程序yourprog.for编译为可执行文件yourprog。
  - *pgf90 -o yourprog -static yourprog.f90*
    将Fortran 90程序yourprog.f90静态编译为可执行文件yourprog。
  - *pgf90 -o yourprog-omp -mp yourprog.f90*
    将OpenMP指令并行的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp。
- GNU Fortran编译器编译举例：
  - *g77 -o yourprog yourprog.for*
    将Fortran 77程序yourprog.for编译为可执行文件yourprog。
  - *gfortran -o yourprog -static yourprog.f90*
    将Fortran 90程序yourprog.f90静态编译为可执行文件yourprog。
  - *gfortran -o yourprog-omp -fopenmp yourprog.f90*
    将OpenMP指令并行的Fortran 90程序yourprog-omp.f90编译为可执行文件yourprog-omp。

注意：*g77*既不支持OpenMP，也不支持Fortran 90标准。

4 OpenMP程序的编译与运行

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编译选项分别为-openmp、-mp和-fopenmp。

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

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=16*设置使用16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 V

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。

本系统安装有两种MPI实现：Intel MPI和Open MPI，并可与不同编译器相互配合使用，安装目录分别在*/opt/intel/impi*和*/opt/openmpi-**[4]。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/4.1.0.030*，建议使用。

用户可以运行*mpi-selector-menu*命令按照提示选择自己使用的MPI环境（注意数字后需要加u），设置完成后最好重新登录以便设置生效：

```
Current system default : intel -mpi-4.1.0.030 _intel -compiler-13.1.0.146
Current user default : <none>

    ”u” and ”s” modifiers can be added to numeric and ”U”
    commands to specify ”user” or ”system-wide”.

1. intel -mpi-4.0.3.008 _intel -compiler-13.0.1.117
2. intel -mpi-4.1.0.030 _intel -compiler-13.0.1.117
3. intel -mpi-4.1.0.030 _intel -compiler-13.1.0.146
4. openmpi-1.6.3_intel -compiler-13.0.1.117
5. openmpi-1.6.3_intel -compiler-13.1.0.146
6. openmpi-1.6.4_gcc-4.4.7
7. openmpi-1.6.4_intel -compiler-13.1.0.146
8. openmpi-1.6.4_pgi-10.6
U. Unset default
Q. Quit

Selection (1-8[us], U[us], Q):
```

上述选项的格式：MPI实现名-MPI实现版本_编译器名-编译器版本。以上述为例，如果想使用1.6.4版本的Open MPI与13.1.0.146版本的Intel编译器的组合，那么请输入7u后回车。

[4]具有不同版本的Open MPI与编译器的组合。

5 MPI并行程序的编译

5.1 Intel MPI库

Intel MPI库[5]是一种多模消息传递接口(MPI)库，所安装的4.1版本Intel MPI库实现了MPI V2.2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。

5.1.1 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：

- 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。

5.1.2 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或-parallel参数，那么使用线程安全库。

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

表 5: 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

  - 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
    * -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。

<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

- -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。

5.1.3 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。

5.1.4 编译举例

对于并行程序，对应不同类型源文件的编译命令如下：

- *mpicc -o yourprog-mpi yourprog-mpi.c*
  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。

- *mpiicxx -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。

- *mpif90 -o yourprog-mpi yourprog-mpi.f*
  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。

- *mpiifort -o yourprog-mpi yourprog-mpi.f90*
  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。

5.2 Open MPI库

Open MPI[6]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的编译命令主要为：

- C程序：*mpicc*
- C++程序：*mpic++*、*mpicxx*、*mpiCC*
- Fortran 77程序：*mpif77*、*mpif90*
- Fortran 90程序：*mpif90*

对于并行程序，对应不同类型源文件的编译命令如下：

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

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。

[6]主页：http://www.open-mpi.org/

- –showme:libs：显示调用的编译器所使用的库名，以空格分隔。
- –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

5.3 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。

6 MPI并行程序的运行

在本系统上，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␣16␣mpijob␣executable*，请参看VIII作业调度系统介绍。

Part VI

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。

7 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。

8 准备所需要调试的程序

8.1 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。

8.2 准备编译器和连接器环境

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。

8.3 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。

8.4 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。

8.5 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

9 开始调试程序

9.1 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*

9.2 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。

9.3 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。

9.4 运行程序

在命令行中输入run，将开始运行程序。

9.5 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。

9.6 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。

9.7 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。

9.8 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。

9.9 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。

9.10 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

9.11 退出调试器

在命令行中输入quit，将退出调试器。

10 传递命令给调试器

10.1 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”

10.2 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。

10.3　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。

10.4　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。

11 调试并行程序

11.1 与线程和进程组一起工作

11.1.1 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。

11.1.2 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

11.1.3　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

11.1.4　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。

11.1.5　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。

11.1.6 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。

11.1.7 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。

11.2 调试多线程应用

11.2.1 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。

11.2.2　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

11.2.3　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。

11.3 调试大规模并行应用

11.3.1 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。

11.3.2 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。

11.3.3 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。

11.3.4 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。

11.3.5 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。

11.3.6 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

11.3.7 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。

11.3.8　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;
```

11.3.9 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]) at ~/test/ cpi. c:8
%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:52
```

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/ cpi. c:8
%4 [0:7] #0 0x0000000000400a41 in f (a=[0.0050000000000000001;0.074999999999999997]) at ~/test/ cpi. c:8
%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:52
```

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
```

11.3.10 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=""
prognamemain=""
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 VII

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。

12 Intel MKL

当前安装的Intel MKL版本为13.1.0.146版本编译器自带的11.0.2版本（安装在*/opt/intel/composer_xe_2013.2.146/mkl*）和13.0.1.117版本编译器自带的11.0.1版本（安装在*/opt/intel/composer_xe_2013.1.117/mkl*），都具有intel64和ia32版本。在bash下可以通过运行*mpi-selector-menu*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：

- 11.0.2版本：

```
../opt/intel/composer_xe_2013.2.146/bin/compilervars.sh intel64
```

- 11.0.1版本：

```
../opt/intel/composer_xe_2013.1.117/bin/compilervars.sh intel64
```

13 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)
- 离散基本线性代数库(Sparse BLAS)
- 线性代数库(LAPACK)
- 可扩展性线性代数库(ScaLAPACK)
- 并行基本线性代数子系统库(PBLAS)
- 离散求解程序(Sparse Solver routines)

- 扩展本征求解子程序(Extended Eigensolver Routines)
- 向量数值库函数(Vector Mathematical Library functions, VML)
- 统计库函数(Statistical Library functions)
- 传统傅立叶变换程序和集群傅立叶变换程序(Conventional DFTs and Cluster DFTs)
- 偏微分方程(Partial Differential Equations)
- 非线性忧患问题求解器(Non-Linear Optimization Problem Solvers)
- 数据拟合函数(Data Fitting Functions)
- 支持函数(Support Functions)
- 基本线性代数通信子程序库(Basic Linear Algebra Communication Subprograms, BLACS)

14 Intel MKL目录内容

Intel MKL的主要目录内容见表6。

15 链接Intel MKL

15.1 快速入门

15.1.1 利用-mkl编译器参数

Intel Composer XE编译器支持采用-mkl[8]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。

[8]是-mkl，不是-lmkl，其它编译器未必支持此-mkl选项。

表 6: 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>/../Documentation/en_US/mkl	MKL文档目录
<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[7]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/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/builder	包含用于生成定制动态可链接库的工具

15.1.2 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15.3.2动态选择接口和线程层部分。

15.1.3 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15.3.2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

15.1.4 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。

15.1.5 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。

15.2 链接举例

15.2.1 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
```

15.2.2 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
```

15.3 链接细节

15.3.1 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。在静态连接时，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。

15.3.2 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。

15.3.3 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。

15.3.4 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	

15.3.5 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。

15.3.6 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。

15.3.7 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）。

16 性能优化等

请参见Intel MKL官方手册。

Part VIII

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(chinagrid)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。

17 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。

18 查看队列情况：bqueues

利用*bqueues*可以查看现有队列信息，例如：

bqueues

将输出：

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

其中，主要列的含义为：

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

19 提交作业：bsub

用户需要利用*bsub*提交作业，其基本格式为*bsub␣[options]␣command␣[arguments]*。其中options设置队列、CPU核数等的选项，必需在command之前，否则将作为command的参数；arguments为设置作业的可执行程序本身所需要的参数，必须在command之后，否则将作为设置队列等的选项。下面将给出常用的几种提交方式。

19.1 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。

19.2 运行串行作业：bsub -q serial

运行串行作业，请使用串行队列serial，比如：

bsub -q serial executable-serial

本超算系统鼓励运行并行作业，允许运行的串行作业资源较少。

19.3 指明所需要的CPU核数：bsub -n

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

bsub -q normal -n 16 mpijob executable-mpi1

如需要的核数多于16个，需利用-q long指明使用long队列。

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

19.4 运行MPI作业：bsub -n NUM mpijob

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

bsub -q long -n 64 mpijob executable-mpi1

注意：

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

19.5 运行OpenMP共享内存作业：bsub -q

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

bsub -q normal -n 16 executable-omp1

19.6 运行MPI和OpenMP共享内存混合并行作业

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

19.7 运行排他性作业：bsub -x

如果需要独占节点运行，此时需要添加-x选项：

bsub -x -q normal -n 8 executable-omp1

注意：排他性作业在运行期间，不允许其余的作业提交到运行此作业的节点，并且只有在某节点没有任何其余的作业在运行时才会提交到此节点上运行。如果不需要采用排他性运行，请不要使用此选项，否则将导致作业必须等待完全空闲的节点才会运行，也许将增加等待时间。

19.8 指明输出、输出文件运行：bsub -i -o -e

作业的正常屏幕输入文件（指的是类似 命令 < 输入文件 方式的文件）、正常屏幕输出到的文件和错误屏幕输出的文件可以利用-i、-o和-e选项来分别指定，运行后可以通过查看指定的这些输出文件来查看运行状态，文件名可利用%J与作业号挂

钩。比如指定executable1的输入、正常和错误屏幕输出文件分别为：*executable1.input*、*executable-作业号.log*和*executable1-作业号.err*：

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

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

19.9 交互式运行作业：bsub -I

如果需要运行交互式的作业（如在运行期间需要手动输入参数或利用调试器手动调试程序等需要进行交互时），需要结合-I参数。建议只是在调试期间使用，一般作业还是尽量不要使用此选项，类似选项还有-Ip和-Is：

bsub -I executable1

20 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
```

21 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。

22 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.
```

23 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.
```

24 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.
```

25 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.
```

26 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   chinagrid   2*node31  *executab1  Mar 12 19:20
                                       1*node18
                                       1*node4
79727 hmli   PEND long     chinagrid              *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>, chinagrid <hmli>, Project <default >, Status <PEND>,
Queue <long> , Command <executab2>
Sun Mar 12 14:15:07: Submitted from host <chinagrid >,
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。

27 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。

28 查看各节点的运行情况：lsload

利用*lsload*命令可查看当前各节点的运行情况，例如：

lsload

HOST_NAME	status	r15s	r1m	r15m	ut	pg	ls	it	tmp	swp	mem
node1	ok	0.0	0.0	0.0	0%	0.5	0	25	227G	32G	62G
node2	locku	0.1	0.0	0.0	0%	0.4	0	60	227G	32G	62G
node4	unavail	-	-	-	-	-	-	-	-	-	-

- HOST_NAME：节点名。
- status：状态。
  - ok：正常状态。
  - busy：超负载运行。
  - lockW：被运行窗口锁定。
  - lockU：有作业在进行排他性运行。
  - unavaild：节点有问题，需要联系管理员处理。
  - unlicensed：节点有问题，需要联系管理员处理。
- r15s：近15秒平均系统负载。
- r1m：近1分钟系统负载。
- r15m：近15分钟系统负载。
- pg：上一分钟的内存分页比率，单位为每页/秒。
- tmp：*/tmp*目录空闲大小，单位为MB。
- swp：可用虚拟内存大小，单位为MB。
- mem：可用内存大小，单位为MB。

29 查看各节点的空闲情况：bhosts

利用*bhosts*命令可查看当前各节点的空闲情况，例如：

bhosts

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
node12	closed	-	16	2	2	0	0	0
node10	ok	-	16	2	1	0	0	0
node14	ok	-	16	2	1	0	0	0

- HOST_NAME：节点名。
- STATUS：状态。
  - ok：可以接收新作业。
  - closed：已经被占满或被管理员关闭，无法接受新作业。
  - unavail：节点出错，需要管理员处理。
  - unreach：节点出错，需要管理员处理。
  - closed_Cu_excl：运行排他性计算单元作业的成员节点。
- JL/U：允许每个用户的作业核数。-表示未限制。
- MAX：允许最大作业核数。
- NJOBS：当前运行的作业数。
- RUN：当前运行作业占据的核数。
- SSUSP：被系统挂起的作业占据的核数。
- USUSP：被用户挂起的作业占据的核数。
- RSV：预留的核数。

即使有节点状态为ok状态，也不一定表示您的作业可以运行，具体运行条件参见17作业运行条件。

30 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。

Part IX

联系方式

- 超算中心：
  - 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