HP RX2600 集群用户使用指南

中国科学技术大学 超级运算中心 李会民

2008 年 5 月

目录

1 HP RX2600 集群概述

中国科学技术大学超级运算中心的峰值计算能力为每秒 3840 亿次 HP Integrity RX2600 集群（以下简称 RX2600 集群）于 2003 年建成，具有一个管理节点、一个登录节点和 32 个计算节点，其具体参数为：

- 每计算节点：2 颗 1.5GHz Itanium 2 Madison 64 位处理器，1 块 36GB SCSI 硬盘，其中 node1 和 node2 各为 16GB 内存，node3 - node32 各为 2GB 内存
- 计算网络：Myrinet 2000
- 存储：2TB 的 HDS9520 磁盘阵列
- 峰值计算能力：每秒 3840 亿次
- 操作系统：Red Hat Enterprise Linux AS release 4 (Nahant)
- 编译器：Intel C/C++ Fortran 编译器
- 数学函数库：Intel MKL、VNI IMSL
- 并行环境：MPICH-GM，支持 MPI 程序；节点内两颗 CPU 为共享内存，节点内可运行 OpenMP 程序，节点间不支持
- 作业管理：Platform LSF

本指南主要将对在此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指南，超算中心也将为用户需要提供必要的技术支持。

2 用户登录与文件传输

RX2600 集群操作系统为 Red Hat Enterprise Linux AS release 4 (Nahant)，用户需以 ssh 或 telnet 方式登录上此系统后进行编译、运行等操作（建议采用 ssh 协议，用户在 MS Windows 下可利用 putty[1] 等支持 ssh 协议的软件进行登录），用户数据则可以利用 ftp和 sftp 协议进行传输（建议在客户端设置使用安全的 sftp 协议）。

[1]putty 下载：http://scc.ustc.edu.cn/download/putty.exe

用户登录进来后默认的 shell 为 bash，可以利用 **chsh** 命令修改为自己喜欢的 shell，修改密码可以在登录节点(hpc1)上运行 **yppasswd** 命令（利用 **passwd** 命令在登录节点上修改密码无效）。

Red Hat Enterprise Linux AS 为 Linux 系统的一种，可以用 **man**␣命令（注意 ␣ 表示空格）或者命令加 -h 或 –help 等选项来查看该命令的详细用法，详细信息请参考 Red Hat Enterprise Linux AS 手册或一般 Linux 手册。

Linux 相关资料：http://scc.ustc.edu.cn/docs/doc-main.php?id=rx2600

3 串行及 OpenMP 程序编译

RX2600 集群上可运行 C/C++、Fortran 的串行程序，以及与 OpenMP 和 MPI 结合的并行程序。编译时，用户只需要在登录节点上以相应的编译命令和选项进行编译即可。当前安装的编译环境为：

- C/C++ 编译器：Intel C++ Compiler for Linux
- Fortran 编译器：Intel Fortran Compiler for Linux
- MPI 并行环境：MPICH-GM

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

3.1 C/C++ 程序的编译

3.1.1 Intel C++ Compiler for Linux 简介

Intel C++ Compiler for Linux 是一种针对 Inetl 平台的高性能的高级编译器，可用于开发复杂且要进行大量计算的程序，也可对 Fortran 程序进行语言间调用。

Intel C++ Compiler for Linux 编译器安装在 /opt/intel/cc/xxx 下，其中 xxx 为版本号，比如当前版本为 10.1.013，安装在 /opt/intel/cc/10.1.013 下，用户可以利用 **ls /opt/intel/cc** 查看还安装有哪些版本。如果想自己指定使用的版本，可以在 ~/.bashrc 之类的文件中添加：.␣<install−dr>/bin/iccvars.sh，其中 <install-dr> 为 /opt/intel/cc/xxx。。注意 ␣ 表示空格，而且前面有个 .，以后类似。

C 和 C++ 程序的编译命令分别为 **icc** 和 **icpc**。**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。

3.1.2 输入输出文件后缀与类型的关系

输入文件的后缀与类型的关系见表 1。

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	预处理文件	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

输出文件的后缀与类型的关系见表 2。

表 2: 输出文件后缀与文件类型的关系

文件名	解释
filename.i	预处理文件由添加 -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

3.1.3 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，在 IA 64 平台上相当于 -O3 -ipo -static，在 IA 32 上相当于 -O3 -ipo -static -no-prec-div -xP。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。
- -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，n 为可产生的目标文件数，为非负整数。
- -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。
- -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。
- -mtune=itanium2：指定针对 Itanium 2 处理器进行优化。
- -openmp：编译 OpenMP 程序，注意：在 RX2600 集群上只能在同一个节点的两个 CPU 上跑 OpenMP 程序，提交作业时请结合 -x 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)。
- -shared：产生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，分别对应不同的标准。
- -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。

建议仔细看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。

3.1.4 C/C++ 程序编译举例

- **icc -o yourprog yourprog.c**
  将 C 程序 yourprog.c 编译为可执行程序 yourprog。
- **icpc -o yourprog yourprog.cpp**
  将 C++ 程序 yourprog.cpp 编译为可执行程序 yourprog。
- **icc -o yourprog-omp -openmp -mtune=itanium2 yourprog.c**
  将 OpenMP 的 C 程序 yourprog-omp.c 编译为针对硬件平台 Itanium2 自动优化的可执行程序 yourprog-omp。

3.1.5 Intel 数学库

Intel C++ Compiler for Linux 附带提供了 Intel 数学库，为了使用此库，需要在程序中包含头文件 mathimf.h。

一个简单例子：

```
// 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 -o real_math

下面是一个包含头文件 complex.h 以使用复数函数的例子：

```
// complex_math.c
#include <stdio.h>
#include <complex.h>
int main()
{
float _Complex c32in,c32out;
double _Complex c64in,c64out;
double pi_by_four= 3.141592653589793238/4.0;
c64in = 1.0 + I* pi_by_four;
// Create the double precision complex number 1 + (pi/4) * i
// where I is the imaginary unit.
c32in = (float _Complex) c64in;
// Create the float complex value from the double complex value.
c64out = cexp(c64in);
c32out = cexpf(c32in);
// Call the complex exponential,
// cexp(z) = cexp(x+iy) = e^ (x + i y) = e^x * (cos(y) + i sin(y))
printf("When z = %7.7f + %7.7f i, cexpf(z) = %7.7f + %7.7f i \n"
, crealf(c32in),cimagf(c32in), crealf(c32out),cimagf(c32out));
printf("When z = %12.12f + %12.12f i, cexp(z) = %12.12f + %12.12f i \n"
, creal(c64in),cimag(c64in),creal(c64out),cimagf(c64out));
return 0;
}
```

编译：

icc -std=c99 complex_math.c -o complex_math

3.1.6 相关文档

- 超算中心主页上的 Intel C++ Compiler for Linux 文档：http://scc.ustc.edu.cn/docs/doc-main.php?id=rx2600
- Intel C++ Compiler for Linux 文档：http://www.intel.com/cd/software/products/asmo-na/eng/346158.htm
- Intel C++ Compiler for Linux 论坛：http://softwarecommunity.intel.com/isn/Community/en-US/forums/1016/ShowForum.aspx

3.2 Fortran 程序的编译

3.2.1 Intel Fortran Compiler for Linux 简介

Intel Fortran Compiler for Linux 是一种针对 Inetl 平台的高性能的高级编译器，可用于开发复杂且要进行大量计算的程序，也可对 C/C++ 程序进行语言间调用。

Intel Fortran Compiler for Linux 编译器安装在 /opt/intel/fc/xxx 下，其中 xxx 为版本号，比如当前版本为 10.1.013，安装在 /opt/intel/fc/10.1.013 下，用户可以利用 **ls /opt/intel/fc** 查看还安装有哪些版本。如果想自己指定使用的版本，可以在 ~/.bashrc 之类的文件中添加：.␣<install−dr>/bin/ifortvars.sh，其中 <install-dr> 为 /opt/intel/fc/xxx。。注意 ␣ 表示空格，而且前面有个 .，以后类似。

Fortran 程序的编译命令为 **ifort**。

3.2.2 输入输出文件后缀与类型的关系

输入文件的后缀与类型的关系见表 3。

输出文件的后缀与类型的关系见表 4。

3.2.3 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）。

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

- -convert [关键字]：转换无格式数据的类型，比如关键词为 big_endian 和 little_endian 时，分别表示无格式的为 big_endian 和 little_endian，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 可选为 72、80 和 132，也可直接用 -72、-80 和 -132 指定，默认为 72 字符。
- -fast：最大化整个程序的速度，在 IA 64 平台上相当于 -O3 -ipo -static，在 IA 32 上相当于 -O3 -ipo -static -no-prec-div -xP。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：生成位置无关代码，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，n 为可产生的目标文件数，为非负整数。
- -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，如库名为 libxyz.a，则可用 -lxyz 指定。
- -mtune=itanium2：指定针对 Itanium 2 处理器进行优化。
- -nofree：指明 Fortran 源程序为固定格式的。

- -openmp：编译 OpenMP 程序，注意：在 RX2600 集群上只能在同一个节点的两个 CPU 上跑 OpenMP 程序，提交作业时请结合 -x 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)。
- -shared：产生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>：编译时显示源文件中不符合此标准的信息，标准可以为 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，为 03、90、95。
- -static：静态链接所有库。
- -unroll[n]：最大循环可展开的层数，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。

建议仔细看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。

3.2.4 Fortran 程序编译举例

- **ifort -o yourprog yourprog.for**
  将 Fortran 77 程序 yourprog.for 编译为可执行程序 yourprog。

- **ifort -o yourprog -static yourprog.f90**
  将 Fortran 90 程序 yourprog.f90 静态编译为可执行程序 yourprog。

- **ifort -o yourprog-omp -openmp -mtune=itanium2 yourprog.f90**
  将 OpenMP 的 Fortran 90 程序 yourprog-omp.f90 编译为针对硬件平台 Itanium2 自动优化的可执行程序 yourprog-omp。

3.2.5 相关文档

- 超算中心主页上的 Intel Fortran Compiler for Linux 文档：http://sfc.ustc.edu.cn/docs/doc-main.php?id=rx2600

- Intel Fortran Compiler for Linux 文档：http://www.intel.com/cd/software/products/asmo-na/eng/346152.htm

- Intel Fortran Compiler for Linux 论坛：http://softwarecommunity.intel.com/isn/Community/en-US/forums/1011/ShowForum.aspx

3.3 OpenMP 程序的编译与运行

Intel C++ 和 Fortran for Linux 编译器支持 OpenMP 并行，只需要利用编译命令结合 -openmp 编译选项进行编译即可，比如：

- **icc -o yourprog-omp -openmp yourprog.c**
  将 OpenMP 的 C 程序 yourprog-omp.c 编译为可执行程序 yourprog-omp。

- **ifort -o yourprog-omp -openmp yourprog.f90**
  将 OpenMP 的 Fortran 90 程序 yourprog-omp.f90 编译为可执行程序 yourprog-omp。

OpenMP 的运行一般是在运行前通过设置环境变量 **OMP_NUM_THREADS** 来控制进程数，比如在 bash 中利用 **export OMP_NUM_THREADS=2** 设置或

在 C shell 中利用 **setenv OMP_NUM_THREADS 2** 设置来用 2 个进程运行，注意，RX2600 集群为分布式内存架构，同一个节点上的两个 CPU 之间为共享内存，因此只能在一个节点上的两个 CPU 之间运行 OpenMP 程序，在提交作业时需要添加 -x 选项以保证在同一个节点运行。

4 MPI 并行环境：MPICH-GM

RX2600 集群的通信网络为 myrinet，在此上安装的 MPI 的并行环境为 MPICH 针对 myrinet 的版本 MPICH-GM，用法等基本等同于 MPICH。

4.1 MPI 并行程序的编译

MPICH-GM 包含的编译命令主要为：**mpicc**、**mpic++**、**mpicxx**、**mpiCC**、**mpif77** 和 **mpif90**，对于并行程序，源文件类型和编译命令的对应关系如下：

- **mpicc -o yourprog-mpi yourprog-mpi.c**
  将 C 语言的 MPI 程序 yourprog-mpi.c 编译为可执行程序 yourprog-mpi。

- **mpicxx -o yourprog-mpi yourprog-mpi.cpp**
  将 C++ 语言的 MPI 程序 yourprog-mpi.cpp 编译为可执行程序 yourprog-mpi，也可换为 **mpic++** 或 **mpiCC**。

- **mpif77 -o yourprog-mpi yourprog-mpi.f**
  将 Fortran 77 语言的 MPI 程序 yourprog-mpi.f 编译为可执行程序 yourprog-mpi。

- **mpif90 -o yourprog-mpi yourprog-mpi.f90**
  将 Fortran 90 语言的 MPI 程序 yourprog-mpi.f90 编译为可执行程序 yourprog-mpi。

MPICH-GM 中的编译命令实际上是调用 Intel C++ 和 Fortran Compiler for Linux 进行编译，具体优化选项等，请参看 Intel C++ 和 Fortran Compiler for Linux 的手册。

4.2 MPI 并行程序的运行

在 RX2600 集群上，MPI 并行程序需结合作业调度系统 LSF 的作业提交命令 **bsub** 来运行，基本用法为 **bsub -q normal -o log -e err_file -a mpich_gm -n 4 ./yourprog-mpi**，必须添加 -a mpich_gm 选项以表示使用 MPICH-GM 并行环境，详细用法将在作业调度部分进行说明。

4.3 MPI 并行程序调试

集群缺乏专业的并行程序调试工具，并行程序的调试一般来说只能利用打印语句来逐步定位错误，建议利用尽量少的进程数来调试以方便进行追踪。

4.4 MPICH-GM 的相关文档

- Myricom 公司的文档：http://www.myri.com/scs/documentation.html
- MPICH1 主页文档：http://www-unix.mcs.anl.gov/mpi/mpich1/download.html

5 数学函数库

RX2600 集群上安装的数学函数库主要有 Intel Math Kernel Library(MKL) 和 VNI International Mathematics and Statistics Library(IMSL)，用户可以直接调用，以提高性能、加快开发。

5.1 Intel MKL

MKL 安装在 /opt/intel/mkl/xxx 下，其中 xxx 为包号，比如 /opt/intel/mkl/10.0.2.0218，在 bash 下可以通过 .␣/opt/intel/mkl/10.0.2.018/tools/environment/mklvars64.sh 来设置 MKL 所需的环境变量 INCLUDE、LD_LIBRARY_PATH 和 MANPATH 等。

5.1.1 MKL 主要内容

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)
- 离散求解程序(Sparse Solver routines)
- 向量数学库函数(Vector Mathematical Library functions)
- 向量统计库函数(Vector Statistical Library functions)
- 傅立叶变换程序(Fourier Transform functions (FFT))
- 集群版傅立叶变换程序(Cluster FFT)
- 区间求解程序(Interval Solver routines)
- 三角变换程序(Trigonometric Transform routines)
- 泊松、拉普拉斯和哈密顿求解程序(Poisson, Laplace, and Helmholtz Solver routines)

- 优化（信赖域）求解程序(Optimization (Trust-Region) Solver routines)

5.1.2 MKL 目录内容

MKL 的主要目录内容如下见表 5。

表 5: MKL 目录内容

目录	解释
<mkl_dir>	MKL 主目录，比如 /opt/intel/mkl/10.0.2.0218
<mkl_dir>/benchmarks/linpack	包含 OpenMP 版的 LINPACK 的基准程序
<mkl_dir>/benchmarks/mp_linpack	包含 MPI 版的 LINPACK 的基准程序
<mkl_dir>/doc	MKL 文档目录
<mkl_dir>/examples	一些例子，建议用户参考学习
<mkl_dir>/include	含有 INCLUDE 文件
<mkl_dir>/interfaces/blas95	包含 BLAS 的Fortran 90 封装及用于编译成库的 makefile
<mkl_dir>/interfaces/lapack95	包含 LAPACK 的Fortran 90 封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw2xc	包含 2.x 版 FFTW(C 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw2xf	包含 2.x 版 FFTW(Fortran 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw3xc	包含 3.x 版 FFTW(C 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw3xf	包含 3.x 版 FFTW(Fortran 接口)封装及用于编译成库的 makefile
<mkl_dir>/interfaces/fftw2x_cdft	包含 2.x 版 MPI FFTW(集群 FFT)封装及用于编译成库的 makefile
<mkl_dir>/lib/32	包含 IA32 架构的静态库和共享目标文件
<mkl_dir>/lib/64	包含 IA64 架构的静态库和共享目标文件
<mkl_dir>/lib/em64t	包含 EM64T 架构的静态库和共享目标文件
<mkl_dir>/man/man3	MKL 的 man 文档
<mkl_dir>/tests	一些测试文件
<mkl_dir>/tools/builder	包含用于产生定制动态可链接库的工具
<mkl_dir>/tools/environment	包含用于设置环境变量的 shell 脚本
<mkl_dir>/tools/support	包含使用 Intel Premier 支持时所需要的包 ID 和许可代码等信息

5.1.3 链接 MKL

为了在程序中链接 MKL 库中的 libyyy.a 或 libyyy.so，可以采用两种方式：

- 在链接行中，列举含有相对或绝对路径的库名，比如：
  **<ld> myprog.o /opt/intel/mkl/10.0.xxx/lib/64/libmkl_solver.a **
  **/opt/intel/mkl/10.0.xxx/lib/64/libmkl_intel.a **
  **/opt/intel/mkl/10.0.xxx/lib/64/libmkl_intel_thread.a **
  **/opt/intel/mkl/10.0.xxx/lib/64/libmkl_core.a **
  /opt/intel/mkl/10.0.xxx/lib/64/libguide.so -lpthread
  其中 <ld> 为链接命令，比如 **ld**，myprog.o 是用户的目标文件。

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

如果已经利用前面所说的 mklvars64.sh 设置好 MKL 环境变量，上面则可以简化为无需指定库所在的绝对路径，只需要利用 -l<库名>指明所需要的库即可。

链接 MKL 库时指明库的路径和库名如下：

−L<MKL path> −I<MKL path>
[−lmkl_lapack95] [−lmkl_blas95]
[cluster components]
[{−lmkl_{intel, intel_ilp64 , intel_lp64 , intel_sp2dp, gf, gf_ilp64 , gf_lp64}]
[−lmkl_{intel_thread, sequential}]
[{−lmkl_solver, −lmkl_solver_lp64, −lmkl_solver_ilp64}]
{{[−lmkl_lapack] −lmkl_{ia32, em64t, ipf}},
−lmkl_core}}
[{−lguide, −liomp5}] [−lpthread] [−lm]

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

5.1.4 相关资料

- 超算中心主页上的 MKL 资料：http://scc.ustc.edu.cn/docs/doc-main.php?id=rx2600

- Intel MKL 文档：http://www.intel.com/cd/software/products/asmo-na/eng/345631.htm
- Intel MKl 论坛：http://softwarecommunity.intel.com/isn/Community/en-US/forums/1273/ShowForum.aspx

5.2 VNI IMSL

IMSL 是 VNI 公司的一套完整的数学与统计程序库数值程序库，能够让使用者嵌入至他们的应用系统中。IMSL 提供高效能的计算能力并提供所有专家所需要开发与建置的精密数值分析应用程序。这些程序库能够让用户直接使用已经写好的数学与统计算法，而免去自我撰写程序的麻烦，并能够让用户轻易的嵌入至用户所使用的 C 与 Fortran 语言程序中。

IMSL 与 MKL 相比主要多了积分、随机数以及统计等方面的函数。

5.2.1 IMSL C 语言版

IMSL C 数值函数库(CNL)包含数学和统计两部分，其帮助文件为：

- MATH/LIBRARY：数学库和特殊函数库
  - MATH/LIBRARY (Volumes 1, 2, and 3)
  - MATH/LIBRARY Special Functions
- STAT/LIBRARY statistics (Volumes 1, and 2)，统计库

大多数程序都有单精度和双精度的版本，许多线性求解和本征系统的程序含有复数和双精度复数版本，从一般的个人计算机到超级计算机，其接口是一样的。

5.2.2 IMSL CNL 数学库内容

IMSL CNL 数学库主要包含以下内容：

- 线性系统(Linear Systems)
- 本征系统分析(Eigensystem Analysis)
- 插值与近似(Interpolation and Approximation)

- 积分(Quadrature)
- 微分方程(Differential Equations)
- 变换(Transforms)
- 非线性方程(Nonlinear Equations)
- 优化(Optimization)
- 特殊函数(Special Functions)
- 统计和随机数生成器(Statistics and Random Number Generator)
- 打印函数(Printing Functions)
- 实用程序(Utilities)

5.2.3 IMSL CNL 统计库内容

IMSL CNL 统计库主要包含以下内容：

- 基本统计(Basic Statistics)
- 回归(Regression)
- 相关性与协方差(Correlation and Covariance)
- 变量分析和设计实验(Analysis of Variance and Designed Experiments)
- 绝对和离散数据分析(Categorical and Discrete Data Analysis)
- 非参数统计(Nonparametric Statistics)
- 拟合良好度测试(Tests of Goodness-of-Fit)
- 时间序列和预测(Time Series and Forecasting)
- 多变量分析(Multivariate Analysis)
- 生存和健壮性分析(Survival and Reliability Analysis)

- 几率分布函数和其反换式(Probability Distribution Functions and Inverses)
- 随机数生成器(Random Number Generator)
- 神经网络(Neural Networks)
- 打印函数(Printing Functions)
- 实用程序(Utilities)

5.2.4 IMSL CNL 安装后目录内容

IMSL CNL 安装在 /opt/vni 下，即 <VNI_DIR> 目录，具体目录包含的文件见表 6。

表 6: IMSL CNL 目录

目录	解释
<VNI_DIR>/imsl/cnl600/help	帮助
<VNI_DIR>/imsl/cnl600/itanium/bin	设置环境变量的脚本
<VNI_DIR>/imsl/cnl600/itanium/examples	一些例子
<VNI_DIR>/imsl/cnl600/itanium/include	头文件
<VNI_DIR>/imsl/cnl600/itanium/lib	库文件
<VNI_DIR>/imsl/cnl600/itanium/notes	说明文件

5.2.5 链接 IMSL CNL

使用 CNL 时，需要在程序的顶部包含对应的头文件：

- 数学库：#include<imsl.h>
- 统计库：#include<imsls.h>

用户可以利用 **. <VNI_DIR>/imsl/cnl600/itanium/bin/cnlsetup.sh** 来设置 CNL 环境变量，运行此命令后，可使用如下环境变量进行编译：

- $CC：C 程序的编译命令
- $CFLAGS：C 编译选项

- $LINK_CNL_SHARED：共享链接的 IMSL 库的选项
- $LINK_CNL_STATIC：静态链接的 IMSL 库的选项
- $LINK_CNL_SHARED_SMP：共享链接的 OpenMP IMSL 库的选项，支持 BLAS 和 LAPACK
- $LINK_CNL：等价于 $LINK_CNL_SHARED
- $LINK_CNL_SMP：等价于 $LINK_CNL_SHARED_SMP

用户可以用 **echo** 命令来查看指定环境变量的具体内容，比如 **echo $LINK_CNL**，也可用 env 命令查看所有已设置的环境变量的内容。

IMSL C 程序可用下面方式进行编译：

$CC −o <executable> $CFLAGS <main.c> $LINK_CNL

详细信息请参看 IMSL CNL-training.pdf(IMSL C Numerical Library 6.0 User Training USTC)以及相关 IMSL 手册。

5.2.6 IMSL Fortran 语言版

IMSL Fortran 数值函数库(FNL)包含数学和统计两部分，其帮助文件为：

- MATH/LIBRARY：数学库和特殊函数库
  - MATH/LIBRARY (Volumes 1, 2, and 3)
  - MATH/LIBRARY Special Functions
- STAT/LIBRARY statistics (Volumes 1, and 2)，统计库

大多数程序都有单精度和双精度的版本，许多线性求解和本征系统的程序含有复数和双精度复数的版本，从一般的个人计算机到超级计算机，其接口是一样的。

5.2.7 IMSL FNL 数学库内容

IMSL FNL 数学库主要包含以下内容：

- 线性系统(Linear Systems)
- 本征系统分析(Eigensystem Analysis)

- 插值与近似(Interpolation and Approximation)
- 积分与微分(Integration and Differentiation)
- 微分方程(Differential Equations)
- 变换(Transforms)
- 非线性方程(Nonlinear Equations)
- 优化(Optimization)
- 基本矩阵/向量操作(Basic Matrix/Vector Operations)
- 线性代数操作和通用函数(Linear Algebra Operators and Generic Functions)
- 实用程序(Utilities)

5.2.8 IMSL FNL 统计库内容

IMSL FNL 统计库主要包含以下内容：

- 基本统计(Basic Statistics)
- 回归(Regression)
- 相关性(Correlation)
- 变量分析(Analysis of Variance)
- 变量分析和设计实验(Analysis of Variance and Designed Experiments)
- 绝对和离散数据分析(Categorical and Discrete Data Analysis)
- 非参数统计(Nonparametric Statistics)
- 拟合良好度测试(Tests of Goodness-of-Fit)
- 时间序列和预测(Time Series and Forecasting)
- 协方差构造和因子分析(Covariance Structures and Factor Analysis)

- 判别式分析(Discriminant Analysis)
- 集群分析(Cluster Analysis)
- 抽样(Sampling)
- 生存分析、生命测试和健壮性分析(Survival Analysis, Life Testing and Reliability)
- 多维缩放(Multidimensional Scaling)
- 密度和哈泽德估计(Density and Hazard Estimation)
- 线性打印机图像(Line Printer Graphics)
- 几率分布函数和其反换式(Probability Distribution Functions and Inverses)
- 随机数生成器(Random Number Generator)
- 实用程序(Utilities)

5.2.9 IMSL FNL 安装后目录内容

IMSL FNL 安装在 /opt/vni 下，即 <VNI_DIR> 目录，具体目录包含的文件见表 7。

表 7: IMSL FNL 目录

目录	解释
<VNI_DIR>/imsl/cnl600/help	帮助
<VNI_DIR>/imsl/cnl600/itanium/bin	设置环境变量的脚本
<VNI_DIR>/imsl/cnl600/itanium/examples	一些例子
<VNI_DIR>/imsl/cnl600/itanium/include	头文件
<VNI_DIR>/imsl/cnl600/itanium/lib	库文件
<VNI_DIR>/imsl/cnl600/itanium/notes	说明文件

5.2.10 链接 IMSL FNL

用户可以利用 **. <VNI_DIR>/imsl/fnl600/itanium/bin/fnlsetup.sh** 来设置 FNL 环境变量，运行此命令后，可使用如下环境变量进行编译：

- $F90：Fortran 程序的编译命令
- $MPIF90：Fortran 90 MPI 程序的编译命令
- $F90FLAGS：自由格式的 Fortran 90 编译选项
- $FFLAGS：固定格式的 Fortran 90 编译选项
- $LINK_F90_SHARED：共享链接的 IMSL 库的选项
- $LINK_FNL_SHARED：等价于 $LINK_F90_SHARED
- $LINK_F90_STATIC：静态链接的 IMSL 库的选项
- $LINK_FNL_STATIC：等价于 $LINK_F90_STATIC
- $LINK_F90：等价于 $LINK_F90_SHARED
- $LINK_FNL：等价于 $LINK_F90
- $LINK_MPI：静态链接 IMSL MPI 库，IMSL 库如可能将多个进程一起计算的并行方式
- $LINK_MPIS：静态链接 IMSL MPI 库，强制 IMSL 库在一个进程内使用串行方式

IMSL Fortran 程序编译举例：

- 编译链接 Fortran 程序：
  $F90 −o <executable> $F90FLAGS <main> $LINK_F90
- 编译链接 Fortran MPI 程序：
  $MPIF90 −o <executable> $F90FLAGS <main> $LINK_MPI

注意：在 MPI 程序中使用 IMSL 库时，需要用 IMSL 的 MPI 的初始及结束函数等代替 MPI_INIT 和 MPI_FINALIZE，其余取得进程号和进程数的函数等也需要替换，具体请查看 IMSL 手册。

详细信息请参看 IMSL FNL-training.pdf(IMSL Fortran Numerical Library 5.0 User Training USTC)以及相关 IMSL 手册。

5.2.11 相关资料

- 超算中心主页的 IMSL 资料：http://scc.ustc.edu.cn/docs/doc-main.php?id=rx2600
- VNI IMSL 技术支持：http://www.vni.com/tech/imsl/index.php
- VNI IMSL 论坛：http://forums.vni.com/forumdisplay.php?f=8

6 作业管理系统

RX2600 集群利用 Platform 公司的 LSF 进行资源和作业管理，所有需要运行的作业均必须通过作业提交命令 **bsub** 提交，提交后可利用相关命令查询作业状态等。为了利用 **bsub** 提交作业，需要在 **bsub** 中指定各选项和需要执行的程序。

注意：

- 不要在登录节点直接运行（编译除外），以免影响其余用户的正常使用
- 如果不通过作业调度系统直接在计算节点上运行将会被监护进程直接杀掉

6.1 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 和 arguments 分别为可设置队列、CPU 数等的选项和作业本身所需要的参数，下面将给出常用的几种提交方式。

6.1.1 提交到特定队列：bsub -q

RX2600 集群共 32 个节点，每个节点含有两颗 Itanium2 CPU，其中 node1 和 node2 上为两颗共享 16GB 内存，其余为共享 2GB 内存。利用 -q 选项可以指定提交到哪个队列，现有的队列为：

- normal：当所需要的 CPU 数不超过 4 个时，作业将在 node3 - node32 上运行，此为默认队列。
- long：当所需要的 CPU 数超过 4 个并不多于 16 个时，作业将在 node3 - node32 上运行。
- hugemem：作业只在各含有 16GB 内存的 node1 和 node2 上运行，当所要运行作业需要的内存普通节点无法满足时，需指明此队列。注意：队列只有 node1 和 node2 节点，如果不需要大内存，请勿使用此队列，一方面避免用户作业长时间等待这两个节点空出，另一方面也避免造成真正需要大内存的作业在等待。

比如想提交到 normal 队列运行串行程序 executable1，可以：

bsub -q normal executable1 或 bsub executable1

如果提交成功，将显示类似下面的输出：

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

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

6.1.2 指明所需要的 CPU 数：bsub -n

利用 -n 选项指定可指定所需要的 CPU 数，比如下面指定利用 4 颗 CPU（由 -n 4 指定）运行 MPI（由 -a mpich_gm 指定）程序：

bsub -q normal -a mpich_gm -n 4 executable-mpi1

如果需要的 CPU 数多于 4 颗并不多于 16 颗时（RX2600 集群最大允许运行的单个作业 CPU 数为 16），需要利用 -q long 指明使用 long 队列。

6.1.3 运行 MPI 作业：bsub -a mpich_gm

如果需要运行 MPI 作业，需要添加 -a mpich_gm 选项，并用 -n 选项指定所需的 CPU 数，比如下面指定利用 16 颗 CPU 运行 MPI 程序 executable-mpi1：

bsub -q long -a mpich_gm -n 16 executable-mpi1

6.1.4 运行共享内存作业或排他性运行作业：bsub -x

集群只能在同一个节点内部运行共享内存（比如 OpenMP）的作业，此时需要添加 -x 选项进行排他性运行（只能在 normal 和 hugemem 队列上使用，long 队列尚未放开此权限），并用 -n 2 选项指定所需的 CPU 数为 2：

bsub -x -q hugemem -n 2 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。

6.1.5 指明输出、输出文件运行：bsub -i -o -e

作业的输入文件、正常屏幕输出到的文件和错误屏幕输出的文件可以利用 -i、-o 和 -e 选项来分别指定，运行后可以通过查看指定的这些输出文件来

查看运行状态。比如指定 executable1 的输入、正常和错误屏幕输出文件分别为：executable1.input、executable1.log 和 executable1.err：

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

6.2 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
```

6.3 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。

6.4 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.
```

6.5 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.
```

6.6 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.
```

6.7 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 hugemem 队列，可以：

bmod -Z "executable2" -q "hugemem" 79727

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

6.8 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    hpc1.ustc.e  2*node31    *executab1  Apr 27 19:20
                                             1*node18
                                             1*node4
79727  hmli   PEND   hugemem   hpc1.ustc.e              *executab2    Apr 27 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>, User <hmli>, Project <default>, Status <PEND>,
Queue <hugemem> , Command <executab2>
Sun Apr 27 14:15:07: Submitted from host <hpc1.ustc.edu.cn>,
CWD <$HOME>, Requested Resources <type==any && swp>35>;
PENDING REASONS:
SCHEDULING PARAMETERS:
            r15s  r1m  r15m   ut  pg    io   ls    it   tmp   swp   mem
loadSched   -     0.7  1.0    -  4.0    -    -     -    -     -     -
loadStop    -     1.5  2.5    -  8.0    -    -     -    -     -     -
```

6.9 查看运行中作业的屏幕正常输出：bpeek

利用 **bpeek** 命令可查看运行中作业的屏幕正常输出，例如：

bpeek 79727

```
<< output from stdout >>
------------------------------------------------

Radius(nm): 300.000
```

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

6.10 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
node12             ok   1.0   2.0   2.1  100%  0.0   0  7356  1878M  2024M  722M
node10         locku    1.0   1.0   1.2   51%  0.0   0  7664  1878M  2026M  407M
```

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

6.11 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
node1       closed      -     4       2     2       0       0     0
```

node10	ok	–	2	2	1	0	0	0
node11	closed	–	2	2	2	0	0	0

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

6.12 查看队列情况：bqueues

利用 **bqueues** 可以查看现有队列信息，例如：

bqueues

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
priority	43	Open:Active	–	64	–	–	0	0	0	0
test	40	Open:Active	–	–	–	–	0	0	0	0
normal	30	Open:Active	–	6	–	–	22	2	20	0
long	30	Open:Active	–	20	–	–	52	12	40	0
hugemem	30	Open:Active	–	4	–	–	3	0	3	0
idle	20	Closed: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 数，- 表示无限，以下类似
- JL/U：单个用户同时可以的 CPU 数
- NJOBS：排队、运行和被挂起的总作业所占 CPU 数
- PEND：排队中的作业所需 CPU 数
- RUN：运行中的作业所占 CPU 数
- SUSP：被挂起的作业所占 CPU 数

6.13 查看用户信息：buser

利用 **buser** 可以查看用户信息，例如：

busers hmli

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

6.14 相关资料

- 超算中心主页的 LSF 资料：http://scc.ustc.edu.cn/docs/doc-main.php?id=rx2600
- Platform 公司 LSF 资料：http://www.platform.com/Products/platform-lsf-family

7 技术支持

中国科学技术大学超级运算中心主页为：http://scc.ustc.edu.cn。超级运算中心将为用户提供相应的技术支持，如有需要请联系：

李会民

- 电话：0551-3602248
- 电邮：hmli@ustc.edu.cn