基于龙芯 2 号国产万亿次高性能计算机 KD-50-I 用户使用手册

中国科学技术大学 KD-50-I 项目组

2007 年 12 月

目录

1 概述

基于龙芯 2 号国产万亿次高性能计算机 KD-50-I（以下简称 KD-50-I）是第一个用国产 CPU 搭建的万亿次高性能机，具有一个登录节点和 336 个处理单元，每个处理单元包含一颗主频 750 MHz 的龙芯 2F CPU 和 1G 内存，处理单元之间通过千兆以太网连接，采用的是 Debian/GNU Linux 无盘系统，支持 C/C++ 和 Fortran77/90/95 程序及 MPI 并行，利用 TORQUE 和 Maui 进行作业调度。

本指南主要将对基于源代码的各类应用如何在 KD-50-I 上运行做基本介绍。熟悉并行机环境、编译器和并行计算的用户可按此指南直接使用 KD-50-I，同时 KD-50-I 项目组的人员也将对用户需要提供技术支持。

2 用户登录与文件传输

KD-50-I 登录节点的 IP 为 202.38.95.50（2001:da8:d800:95:2e0:81ff:fe4b:7f07 为其 IPV6 地址），用户需以 ssh 方式登录上此节点后进行编译、运行等操作（不支持 telnet 协议，用户在 MS Windows 下可利用 putty 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）。

3 程序编译

KD-50-I 支持串行程序和 MPI 的并行程序，用户只需要在登录节点上以交叉编译命令进行编译即可，无需登录到龙芯处理单元上进行编译。

3.1 串行程序的编译

对串行程序，利用 GNU 编译器 GCC 在登录节点上以交叉编译方式进行编译，当前的 GCC 版本为 4.2.2，安装在 /opt/toolchain-gcc4.2.2 下，编译命令全部以 mips64el-linux- 开头[1]。C、C++、Fortran77/9x 的串行源代码与其对应的编译命令如下：

- C：mips64el-linux-gcc -o yourprog yourprog.c
- C++：mips64el-linux-g++ -o yourprog yourprog.cpp
- F77：mips64el-linux-gfortran -o yourprog yourprog.f
- F90：mips64el-linux-gfortran -o yourprog yourprog.f90

具体编译优化参数等，请参考 GCC 编译器与龙芯 2F 的相关手册。

3.2 并行程序的编译

KD-50-I 的 MPI 环境使用的是 MPICH2，当前的版本为 1.0.6p1，安装在 /opt /mpich2 下，与串行程序一样也是在登录节点上以交叉编译的方式进行编译，但并行编译命令不带 mips64el-linux- 前缀。

对于并行程序，源文件类型和编译命令的对应关系见下表：

- C：mpicc -o yourprog yourprog.c
- C++：mpicxx -o yourprog yourprog.cpp
- F77：mpif77 -o yourprog yourprog.f
- F90：mpif90 -o yourprog yourprog.f90

[1]默认的 gcc、g++、gfortran 等命令是登录节点的 X86 平台的命令，用此编译命令编译的程序无法在龙芯节点上运行，对于源程序自带的编译配置文件 Makefile 等也必须修改为以 mips64el-linux- 开头的对应命令。

4 作业运行

KD-50-I 利用 TORQUE 和 Maui 进行资源和作业管理，所有需要运行的作业无论是用于程序调试还是业务计算均必须通过 qsub 命令提交，提交后可以利用 TORQUE 和 Maui 的相关命令查询作业状态等。为了利用 qsub 提交作业，用户需针对此作业创建提交脚本，在脚本里面设定需要运行的作业参数等。在此分别给出串行和并行的简单脚本，用户可以修改此脚本以适用于自己的作业，如需要更加高级的功能请参考 TORQUE 手册。

4.1 串行作业

对于串行程序，用户可编写命名为 serial_job.sh（此脚本名可以按照用户喜好命名）的串行作业脚本，其内容如下：

```
#!/bin/sh
#PBS -N job_name
#PBS -o job.log
#PBS -e job.err
#PBS -q dque
cd yourworkdir
echo Running on hosts `hostname`
echo Time is `date`
echo Directory is $PWD
echo This job runs on the following nodes:
cat $PBS_NODEFILE
echo This job has allocated 1 node
./yourprog
```

注意[2]，TORQUE 建立在 PBS 作业管理系统之上，PBS 的参数需在作业提交脚本中利用 #PBS 设置。上述脚本利用 qsub 命令提交后，表示进入 yourworkdir 目录后，提交到 dque 队列，其作业名为 job_name，标准输出和错误输出将分别存在此目录下的 job.log 和 job.err 文件中。上述脚本中以 #PBS 开头的几行的 -N、-o、-e、-q 参数后分别设置的是这个作业的名字 job_name、标准输出定向到的文件名 job.log、

[2]此脚本中 `hostaname` 等中的是键盘左上角的反引号 `，不是右侧的 '

标准错误输出定向到的文件名 job.err、作业使用的队列名 dque。

作业脚本编写完成后，可以按照下面命令提交作业：

user@kd50:~ /work$ **qsub ser_job.sh**

如果成功，将有类似下面的输出：

```
37.kd50
```

其中 37.kd50 表示的是作业号，由两部分组成，37 表示的是作业序号，kd50 表示的是作业管理系统的主机名，也就是登录节点名，之后可以用此作业号来查询作业及终止此作业等。

4.2 并行作业

与串行作业类似，对于并行作业，则需要编写类似下面脚本 par_job.sh：

```
#!/bin/sh
#PBS -N job_name
#PBS -o job.log
#PBS -e job.err
#PBS -q dque
#PBS -l nodes=16
cd yourworkdir
echo Time is `date`
echo Directory is $PWD
echo This job runs on the following nodes:
cat $PBS_NODEFILE
NPROCS=`wc -l<$PBS_NODEFILE`
echo This job has allocated $NPROCS nodes
mpiexec -machinefile $PBS_NODEFILE -np $NPROCS ./yourprog
```

与串行程序的脚本相比，主要不同之处在于在#PBS 开头的 -l 参数后设置：nodes=所需要的进程数，另外请注意需采用 mpiexec 的命令格式提交并行可

执行程序。

与串行作业类似，可使用下面方式提交：

user@kd50:~ /work$ **qsub par_job.sh**

4.3 常用作业管理命令

用户常用的与作业相关的 TORQUE 和 Maui 命令主要有：

- canceljob：取消已存在的作业
- checkjob：显示作业状态、资源需求、环境、限制、信任、历史、已分配资源和资源利用等
- nqs2pbs：将 nqs 作业脚本转换为 pbs 作业脚本
- pbsnodes：显示节点信息
- printjob：显示指定作业脚本中的作业信息
- qdel：取消指定的作业
- qhold：挂起一个作业
- qmove：将一个作业从一个队列移到另一个队列中
- qnodes：pbsnodes 的别名，显示节点信息
- qorder：交换两个作业的排队顺序
- qrls：将被挂起的作业送入准备运行的队列中
- qselect：显示符合条件的作业的作业号
- qstat：显示队列、服务器和作业的信息
- qsub：提交作业

- showbf：显示有特殊资源需求的资源的可用性
- showq：显示已激活和空闲的作业的优先级细节
- showstart：显示空闲作业的估计开始时间
- tracejob：追踪作业信息

具体请参考 TORQUE 和 Maui 用户手册。

4.3.1 查看队列中的作业状态：qstat

利用 qstat 可以查看作业的运行状态：

user@kd50:~ /work$ **qstat**

输入上面命令后，将给出类似下面的输出：

Job id	Name	User	Time Use	S	Queue
48.kd50	job_name4	user	0	E	dque
49.kd50	job_name1	user	00:00:00	R	dque
50.kd50	job_name2	user	0	H	dque
51.kd50	job_name3	user	0	Q	dque

上面几列的含义分别为：作业号、作业名、用户名、使用的时间、状态、队列名，其中状态中的 E、Q、H 和 R 分别表示作业处于退出、挂起、排队和运行中。

4.3.2 挂起作业：qhold

qhold 命令可以挂起作业，被挂起的作业将不被执行，这样可以让其余作业优先得到资源运行，被挂起的作业在用 qstat 命令查询时显示的状态标志为 H，下面命令将挂起作业号为 50.kd50 的作业：

user@kd50:~ /work$ **qhold 50.kd50**

4.3.3 取消挂起：qrls

被挂起的作业可以利用 qrls 来取消挂起，重新进入等待运行状态：

user@kd50:~ /work$ **qrls 50.kd50**

4.3.4 终止作业：qdel 和 canceljob

用户如果想终止一个作业，可以利用 qdel 或 canceljob 来取消：

user@kd50:~ $ **qdel 50.kd50**

user@kd50:~ $ **canceljob 51.kd50**

4.3.5 查看作业状态：checkjob

利用 checkjob 可以查看作业的状态：

user@kd50:~ $ **checkjob 51.kd50**

```
checking job 51

State: Hold
Creds:  user:user  group:user  class:dque  qos:DEFAULT
WallTime: 00:00:00 of 99:23:59:59
SubmitTime: Sun Dec  2 19:22:19
  (Time Queued  Total: 00:46:13  Eligible: 00:24:40)

Total Tasks: 16
Req[0]  TaskCount: 16  Partition: ALL
Network: [NONE]  Memory >= 0  Disk >= 0  Swap >= 0
Opsys: [NONE]  Arch: [NONE]  Features: [NONE]

IWD: [NONE]  Executable:  [NONE]
Bypass: 0  StartCount: 0
PartitionMask: [ALL]
Flags:       RESTARTABLE
```

```
PE:  16.00  StartPriority:  24
cannot select job 51 for partition DEFAULT (non-idle state 'Hold')
```

从上面的 State: Hold 可以看出作业已被挂起。

user@kd50:~ $ **checkjob 49.kd50**

```
checking job 49
State: Running
Creds:  user:user  group:user  class:dque  qos:DEFAULT
WallTime: 1:07:14 of 99:23:59:59
SubmitTime: Sun Dec  2 19:02:10
  (Time Queued  Total: 00:00:01  Eligible: 00:00:01)
StartTime: Sun Dec  2 19:02:11
Total Tasks: 16
Req[0]  TaskCount: 16  Partition: DEFAULT
Network: [NONE]  Memory >= 0  Disk >= 0  Swap >= 0
Opsys: [NONE]  Arch: [NONE]  Features: [NONE]
NodeCount: 16
Allocated Nodes:
[node16:1][node15:1][node14:1][node13:1]
[node12:1][node11:1][node10:1][node09:1]
[node08:1][node07:1][node06:1][node05:1]
[node04:1][node03:1][node02:1][node01:1]

IWD: [NONE]  Executable:  [NONE]
Bypass: 0  StartCount: 1
PartitionMask: [ALL]
Flags:       RESTARTABLE

Reservation '49' (-1:06:52 -> 99:22:53:07  Duration: 99:23:59:59)
PE:  16.00  StartPriority:  1
```

从上面的 State: Running 可看出作业处于运行中，并且可看到占用的资源状态。

4.3.6 交换两个作业的排队顺序：qorder

利用 qorder 可以交换两个作业的排队顺序：

user@kd50:~ $ **qstat**

Job id	Name	User	Time Use	S	Queue
52.kd50	job_name1	user	0	H	dque
53.kd50	job_name2	user	0	Q	dque
54.kd50	job_name3	user	0	Q	dque

user@kd50:~ $ **qorder 53.kd50 54.kd50**

user@kd50:~ $ **qstat**

Job id	Name	User	Time Use	S	Queue
52.kd50	job_name1	user	0	H	dque
54.kd50	job_name3	user	0	Q	dque
53.kd50	job_name2	user	0	Q	dque

可见 qorder 53.kd50 54.kd50 后，作业 53.kd50 和 54.kd50 的排队顺序相互对换了，这样作业 54.kd50 将优先于 53.kd50 运行。

4.3.7 选择符合特定条件的作业的作业号：qselect

qselect 可以用来显示符合一定条件的作业的作业号，比如选择被挂起的作业，可以用下面的命令：

user@kd50:~ $ **qselect -s H**

```
52.kd50
```

4.3.8 显示队列中作业的信息：showq

user@kd50:~ $ **showq**

```
ACTIVE JOBS--------------------
JOBNAME   USERNAME      STATE   PROC     REMAINING            STARTTIME
```

```
52              user   Running     16 99:22:44:09  Sun Dec 2 21:04:37
     1 Active Job      16 of    16 Processors Active (100.00%)
IDLE JOBS----------------------
JOBNAME  USERNAME     STATE  PROC      WCLIMIT             QUEUETIME

54              user      Idle     16 99:23:59:59  Sun Dec 2 21:04:45 1
Idle Job

BLOCKED JOBS-------------------
JOBNAME  USERNAME     STATE  PROC      WCLIMIT             QUEUETIME

53              user      Hold     16 99:23:59:59  Sun Dec 2 21:04:37
Total Jobs: 3  Active Jobs: 1   Idle Jobs: 1   Blocked Jobs: 1
```

4.3.9 显示节点信息：pbsnodes 和 qnodes

利用 pbsnodes 和 qnodes（实际两者是同一个命令的两个名字）可以显示系统各个节点的信息，比如空闲（free）、当机（down）、离线（offline）。例如：显示所有空闲的节点：

user@kd50:~ $ **pbsnodes -l free**

其输出为：

```
node0101                free
node0102                free
node0104                free
node0105                free
```

5 数学库使用

KD-50-I 上目前安装的数学库主要有：BLAS、LAPACK、ScaLAPACK 和 FFTW，用户可以直接调用。

5.1 基本线形代数运算库：BLAS

5.1.1 简介

基本线形代数运算库（BLAS）提供基本的向量矩阵运算，可分为三层：

第一层包含如下形式的向量运算：

$$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$$

以及向量点乘、向量求模运算等。

第二层包含如下形式的矩阵向量运算等：

$$\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$$

$$A \leftarrow \alpha\mathbf{x}\mathbf{y}^T + A$$

第三层包含如下形式的矩阵运算等。

$$C \leftarrow \alpha AB^T + \beta C$$

KD-50-I 上的 BLAS 是经过针对龙芯 2F 体系结构优化过的版本，使用此运算库，可以得到更高的性能。

5.1.2 使用范例

我们给出一个在 C 程序中调用 BLAS 矩阵相乘函数的例子。

gemm.c:

```
#include "stdio.h"
#include "stdlib.h"
```

```
void dgemm_(char*,char*, const int* M, const int* N,
            const int* K, const double* alpha, const double *A,
            const int* lda, const double *B, const int* ldb,
            const double* beta, double *C, const int* ldc);
int main(int argc, char **argv) {
char trans='T';
double A[M*K]={};
double B[K*N]={};
double C[M*N]={};
double alpha=...;
double beta=...;
int lda=M;
int ldb=K;
int ldc=M;
dgemm_(&ntrans,&ntrans,&M,&N,&K,
       &alpha,A,&lda,B,&ldb,&beta,C,&ldc);
/* C= alpha * A * B + beta * C */
}
```

在 KD-50-I 上编译：

user@kd50:~ $ **mips64el-linux-gcc -o gemm gemm.cpp -lblas -lgfortran -lm**

MPI 程序中使用此数值函数库与此类似。

5.2 自适应线性代数库：ATLAS

ATLAS 是根据体系结构自适应优化的 BLAS，包含 BLAS 的所有函数和一些 LAPACK 函数，一般来说具有较高的效率。

在 KD-50-I 上编译：

user@kd50:~ $ **mips64el-linux-gcc -o gemm gemm.cpp -lcblas -latlas -lm**

5.3 线性代数库：LAPACK

线性代数库（LAPACK）提供解线性方程，最小二乘法问题，特征值问题和奇异值问题等多种函数调用。

下面给出一个求解对称矩阵广义特征值与特征向量的 C 程序调用 LAPACK 的例子。

sygvd.c：

```
#include "stdio.h"
#include "stdlib.h"
int dsygvd_(int *itype, char *jobz, char *uplo, int *n,
            double *a, int *lda, double *b, int *ldb,
            double *w, double *work, int *lwork, int *iwork,
            int *liwork, int *info );
int main(int argc, char **argv) {
char aa='V';
char bb='U';
int itype=1;
int N=...;
double A[N*N]={}; /*symmetric*/
double B[N*N]={}; /*symmetric ,positive definite */
int wl=1 + 6*N + 2*N*N;
int iw=3+5*N;
double W[N]={}; /*OUTPUT,eigenvalues in ascending order*/
double WL[wl]={};/*work space*/
double IW[iw]={};/*work space*/
int k;/*OUTPUT,INFO*/
dsygvd_(&itype,&aa,&bb,&N,A,&N,B,&N,W,WL,&wl,IW,&iw,&k);
/* Solving A*x=(lambda)*B*x Eigen-Problem */
return 0;
}
```

在 KD-50-I 上编译：

user@kd50:~ $ **mips64el-linux-gcc -o sygvd sygvd.c -llapack -lblas -lgfortran -lm**

5.4 可扩展线性代数库：ScaLAPACK

ScaLAPACK 是 LAPACK 的并行版本，支持 MPI，可用于求解线性方程系统、线性最小二乘问题和奇异值问题等。下面给出一个调用 PCHEEVX 求 Hermitian 矩阵的特征值和特征向量的 Fortran77 例子。

pcheevx.f:

```
      INTEGER LWORK, MAXN, LIWORK
      REAL    ZERO, MONE
      PARAMETER ( LWORK = 100000, MAXN = 100, LIWORK = 5000, ZERO = 0.0E+0 )
      INTEGER LDA, MAXPROCS
      PARAMETER ( LDA = MAXN, MONE = -1.0E+0, MAXPROCS = 512 )
      INTEGER LRWORK
      PARAMETER ( LRWORK = 100000 )
      INTEGER CONTEXT, I, IAM, INFO, M, MYCOL, MYROW, N, NB, NPCOL, NPROCS, NPROW, NZ
      INTEGER DESCA( 50 ), DESCZ( 50 ),ICLUSTR( MAXPROCS*2 ), IFAIL( MAXN ),IWORK( LIWORK )
      REAL    GAP( MAXPROCS ), RWORK( LRWORK ), W( MAXN )
      COMPLEX A( LDA, LDA ), WORK( LWORK ), Z( LDA, LDA )
      EXTERNAL BLACS_EXIT, BLACS_GET, BLACS_GRIDEXIT, BLACS_GRIDINFO, BLACS_GRIDINIT,
     $ BLACS_PINFO, BLACS_SETUP, DESCINIT, PCHEEVX, PCLAMODHILB, PCLAPRNT
      N = 4
      NB = 1
      NPROW = 2
      NPCOL = 1
      CALL BLACS_PINFO( IAM, NPROCS )
      IF( ( NPROCS.LT.1 ) ) CALL BLACS_SETUP( IAM, NPROW*NPCOL )
      CALL BLACS_GET( -1, 0, CONTEXT )
      CALL BLACS_GRIDINIT( CONTEXT, 'R', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( CONTEXT, NPROW, NPCOL, MYROW, MYCOL )
      IF( MYROW.EQ.-1 ) GO TO 20
      CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, CONTEXT, LDA, INFO )
      CALL DESCINIT( DESCZ, N, N, NB, NB, 0, 0, CONTEXT, LDA, INFO )
      CALL PCLAMODHILB( N, A, 1, 1, DESCA, INFO )
      CALL PCHEEVX( 'V', 'A', 'U', N, A, 1, 1, DESCA, ZERO, ZERO, 13,-13, MONE, M, NZ, W,
     $ MONE,Z,1,1,DESCZ, WORK,LWORK,RWORK,LRWORK,IWORK,LIWORK,IFAIL,ICLUSTR,GAP,INFO )
      CALL PCLAPRNT( N, N, Z, 1, 1, DESCZ, 0, 0, 'Z', 6, WORK )
      CALL BLACS_GRIDEXIT( CONTEXT )
   20 CONTINUE
      CALL BLACS_EXIT( 0 )
 9999 FORMAT( 'W=diag([', 4D16.12, ']);' )
      STOP
      END
      SUBROUTINE PCLAMODHILB( N, A, IA, JA, DESCA, INFO )
      INTEGER BLOCK_CYCLIC_2D, DLEN_, DT_, CTXT_, M_, N_, MB_, NB_, RSRC_, CSRC_, LLD_
      PARAMETER ( BLOCK_CYCLIC_2D = 1, DLEN_ = 9, DT_ = 1,CTXT_ = 2, M_ = 3, N_ = 4,
     $ MB_ = 5, NB_=6, RSRC_ = 7, CSRC_ = 8, LLD_ = 9 )
      REAL    ONE
      PARAMETER ( ONE = 1.0E+0 )
      INTEGER IA, INFO, JA, N
      INTEGER DESCA( * )
      COMPLEX A( * )
      INTEGER I, J, MYCOL, MYROW, NPCOL, NPROW
      EXTERNAL BLACS_GRIDINFO, PCELSET
      INTRINSIC CMPLX, REAL
      IF( BLOCK_CYCLIC_2D*CSRC_*CTXT_*DLEN_*DT_*LLD_*MB_*M_*NB_*N_* RSRC_.LT.0 )RETURN
      INFO = 0
      CALL BLACS_GRIDINFO( DESCA( CTXT_ ), NPROW, NPCOL, MYROW, MYCOL )
      IF( IA.NE.1 ) THEN
         INFO = -3
      ELSE IF( JA.NE.1 ) THEN
         INFO = -4
      END IF
```

```
      DO 20 J = 1, N
         DO 10 I = 1, N
            IF( I.EQ.J ) THEN
               CALL PCELSET( A, I, J, DESCA, CMPLX( ONE / ( REAL( I+J )-ONE ) )+CMPLX( ONE ) )
            ELSE
               CALL PCELSET( A, I, J, DESCA, CMPLX( ONE / ( REAL( I+J )-ONE ), REAL( J-I ) ) )
            END IF
   10    CONTINUE
   20 CONTINUE
      END
```

其编译命令为：

user@kd50:~ $ **mpif77 -o pcheevx pcheevx.f -lscalapack -lblacsF77init_MPI-LINUX-0 -lblacs_MPI-LINUX-0 -lblas**

5.5 快速傅立叶变换：FFTW

FFTW 库（Fastest Fourier Transform in the West）用于计算快速傅立叶变换。最新稳定版本是 FFTW3.1.2，但因为 FFTW3 并不向下兼容，很多遗留程序使用的是 FFTW2，所以 KD-50-I 上也安装了 FFTW2 的稳定版本 2.1.5。

FFTW2 提供 MPI 并行接口，FFTW3 稳定版不支持 MPI 并行，FFTW3 最新版本 3.2 alpha3 提供 MPI 并行接口。

下面给出一个 MPI 的 C 程序调用 FFTW2 的例子。

fft.c：

```
#include "mpi.h"
#include "fftw.h"
#include "fftw_mpi.h"
int main(int argc, char **argv)
{
    MPI_Init(&argc,&argv);
    fftwnd_mpi_plan mpi_plus_plan;
    double *psicmpi;
    int nx,ny,nz;
    int local_n , local_start , local_n_after_transform ,
        local_start_after_transform ,fftmy;
```

```
    mpi_plus_plan = fftw3d_mpi_create_plan(
                        MPI_COMM_WORLD,nx,ny,nz,
                        FFTW_BACKWARD,
                        FFTW_MEASURE|FFTW_IN_PLACE|
                        FFTW_THREADSAFE|FFTW_USE_WISDOM);
    fftwnd_mpi_local_sizes (mpi_plus_plan,
                            &local_n, &local_start,
                            &local_n_after_transform,
                            &local_start_after_transform,&fftmy);
    psicmpi = (double*)malloc(sizeof(double)*fftmy);
    fftwnd_mpi(mpi_plus_plan, 1, psicmpi,
               NULL, FFTW_NORMAL_ORDER);
    MPI_Finalize();
    return 0;
}
```

在 KD-50-I 上编译：

user@kd50:~ $ **mpicc -o fft fft.cpp -lfftw_mpi -lfftw**

6 程序调试指南

在 KD-50-I 上，我们可以使用 GNU 工具软件 GDB 来调试用 C 、 C++ 以及 Fortran 写成的串行程序和并行程序。

6.1 串行程序调试

为了使程序能够被 GDB 调试，在编译时需加上 -g 编译选项。

输入 gdb 进入 gdb 调试环境，然后用 file yourpro 加载文件，或者直接输入 gdb yourpro 来加载文件。

gdb 的常用命令如下：

命令	缩写	功　　能
break	b	设置断点
step	s	执行一行源代码（进入函数内部）
next	n	执行一行源代码（不进入函数内部）
list	l	列出10行源代码
print	p	行打印数据内容
shell		执行shell命令（不退出GDB）
run	r	执行当前被调试的程序
continue	c	继续执行源代码直到下一个断点
set		设置程序参数
quit	q	退出GDB

更多命令可以在 shell 中执行 man gdb 查看。

我们用 5.1.2 节中的程序作为例子。

user@kd50:~ $ **gcc -g -o gemm gemm.c -lblas -lgfortran**

user@kd50:~ $ **gdb**

```
(gdb) file gemm
Reading symbols from /home/mmsp/eigen/gemm...done.
Using host libthread_db library "/lib/libthread_db.so.1".
(gdb) l
2       #include "stdlib.h"
3
4
5       void dgemm_(char*,char*, const int* M, const int* N,
6            const int* K, const double* alpha, const double *A,
7            const int* lda, const double *B, const int* ldb,
8            const double* beta, double *C, const int* ldc);
9
10      int main(int argc, char **argv) {
11              char trans='T';
(gdb)
12              int M=2;
13              int N=3;
14              int K=1;
15              double A[2]={1,2};
```

```
16              double B[4]={1,2,3};
17              double C[6]={0,0,0,0,0,0};
18              double alpha=1.0;
19              double beta=0.0;
20              int lda=K;
21              int ldb=N;
(gdb) b main
Breakpoint 1 at 0x400928: file gemm.c, line 11.
(gdb) l
22              int ldc=M;
23              dgemm_(&trans,&trans,&M,&N,&K,
24                     &alpha,A,&lda,B,&ldb,&beta,C,&ldc);
25              int i;
26              for(i=0;i<M*N;i++) printf("%f\n",C[i]);
27              return 0;
28      }
(gdb) b 22
Breakpoint 2 at 0x400a6c: file gemm.c, line 22.
(gdb) r
Starting program: /home/mmsp/eigen/gemm
Breakpoint 1, main (argc=1, argv=0x7ef03aa4) at gemm.c:11
11              char trans='T';
(gdb) c
Continuing.
Breakpoint 2, main (argc=1, argv=0x7ef03aa4) at gemm.c:22
22              int ldc=M;
(gdb) n
23              dgemm_(&trans,&trans,&M,&N,&K,
(gdb) s
26              for(i=0;i<M*N;i++) printf("%f\n",C[i]);
(gdb) p A
$1 = {1, 2}
(gdb) c
Continuing.
1.000000
2.000000
2.000000
4.000000
3.000000
6.000000

Program exited normally.
(gdb) q
```

6.2 并行程序调试

并行程序的常用的调试方法是程序员在代码中插入输出语句打印即时数据，从而进行程序调试，如非必要应启动尽量少的进程以方便调试。现在在 MPICH2 中，并行程序同样也可以用 gdb 进行调试，通过 mpiexec 启动 gdb ，可以用命令 z 进行各进程间的转换。下面用 mpi 计算 π 值的示例程序展示 gdb 调试 mpi 并行程序。

user@kd50:~ $ **mpicc -g -o cpi cpi.c**

user@kd50:~ $ **mpiexec -gdb -np 2 ./cpi**

```
0-1:  (gdb) l
0-1:  24
0-1:  25              fprintf(stderr,"Process %d on %s\n",
0-1:  26                      myid, processor_name);
0-1:  27
0-1:  28              n = 0;
0-1:  29              while (!done)
0-1:  30              {
0-1:  31                  if (myid == 0)
0-1:  32                  {
0-1:  33          /*
0-1:  (gdb) b 28
0-1:  Breakpoint 2 at 0x405434: file cpi.c, line 28.
0-1:  (gdb) r
0-1:  Continuing.
0:  Process 0 on node0111
1:  Process 1 on node0112
0-1:
1:  Breakpoint 2, main (argc=1, argv=0x7eba39e4) at cpi.c:28
0:  Breakpoint 2, main (argc=1, argv=0x7fb639e4) at cpi.c:28
0-1:  28              n = 0;
0-1:  (gdb) 0-1:  (gdb)
0-1:  (gdb) p myid
0:  $1 = 0
1:  $1 = 1
0-1:  (gdb) z 0
0:  (gdb) p myid
0:  $2 = 0
0:  (gdb) z 1
1:  (gdb) p myid
```

```
1:  $2 = 1
1:  (gdb) z
0-1:  (gdb) c
0-1:  Continuing.
0:  pi is appro 3.1416009869231241, Error: 0.0000083333333309
1:
0:  wall clock time = 0.007154
1:  Program exited normally.
0:
1:  (gdb) 0:  Program exited normally.
0:  (gdb) q
0-1:  MPIGDB ENDING
```

7 注意事项

- KD-50-I 系统为 Little Endian 构架，如果发现程序运行有问题，请检查二进制输入文件是否有问题，一般来说可通过在此平台下重新生成来解决。
- 程序编译在登录节点上必须以交叉编译的方式进行，串行编译命令以 mips64el-linux- 开始，并行编译命令与常用的一样以 mpi 开始。
- 用户 configure，Makefile 等配置文件需修改其编译选项设为对应的交叉编译器。
- 作业提交和管理需要在登录节点上进行，在处理单元上无法运行相关命令。

8 相关文档

- 龙芯 CPU：http://www.loongson.cn/loongson/down/
- 编译器：
  - GCC：http://gcc.gnu.org/onlinedocs/
  - MPICH2：http://www.mcs.anl.gov/research/projects/mpich2/documentation/files/mpich2-doc-user.pdf
- 数学函数库：
  - BLAS：http://www.netlib.org/blas/
  - ATLAS：http://www.netlib.org/atlas/
  - LAPACK：http://www.netlib.org/lapack/
  - ScaLAPACK：http://www.netlib.org/scalapack/slug/
  - FFTW：http://www.fftw.org
- 作业调度与资源管理：
  - TORQUE：http://www.clusterresources.com/torquedocs21/
  - Maui：http://www.clusterresources.com/products/maui/docs/mauiusers.shtml

9 技术支持

中国科学技术大学基于龙芯 2 号国产万亿次高性能计算机 KD-50-I 的主页为：http://kd50.ustc.edu.cn 和 http://www.kd50.ustc.edu.cn（请选择适合自己的网址，校外用户一般来说 http://www.kd50.ustc.edu.cn 会快一点）。KD-50-I 项目小组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：+86-551-3602248
  - 电邮：hmli@ustc.edu.cn
- 方维
  - 电话：+86-551-3602442
  - 电邮：fangwei@mail.ustc.edu.cn