Intro to Parallel Processing with MPI/MPICH2

M. Bellis[1]

[1]Department of Physics
Carnegie Mellon University

January 30, 2007

Outline

1. What is MPI?
2. Running MPICH code
   - Starting the daemon
   - Hello World!
3. More involved examples
4. Calculate π example
5. Fitting with Minuit

What is MPI?

- MPI (Message Passing Interface)
  - Set of standards to allow parallel processing. http://www.mpi-forum.org/
- MPICH2 (Message Passing Interface)
  - Implementation of these standards.
  - Original MPICH was in FORTRAN.
  - Now has C/C++ bindings for message calls.
  - Also comes with MPI daemons (`mpd`) which must run on processors. http://www-unix.mcs.anl.gov/mpi/mpich/index.htm

Starting the daemon

- MPICH requires a daemon running on the CPU's
  - Can just run `mpd` at the prompt on each machine.
- Better way
  - `mpdboot -n <n cpu's> -f <hosts file>`

Starting the daemon

Sample `mpd.hosts` file

```
albert.phys.cmu.edu
max.phys.cmu.edu
qcd002.phys.cmu.edu
#qcd004.phys.cmu.edu
```

- Note that you can specify less cpu's on the command line than are actually in the file, or comment some out.
- When you want to shut down:
  - `mpdcleanup -f <hosts file>`

Hello World!

```
#include <mpi.h>
#include <iostream>

using namespace std;

int main (int argc, char **argv)
{
  char name[256];
  int namelength;
  MPI::Status status;
  MPI::Init( argc, argv );
  int num_procs = MPI::COMM_WORLD.Get_size();
  int rank = MPI::COMM_WORLD.Get_rank();
  MPI::Get_processor_name (name, namelength);
  cerr << "Process " << rank << " of " << num_procs
       << " is running on " << name << endl;
  MPI::Finalize ();
  return 0;
}
```

Makefile

```
CC = /usr/local/bin/mpicc
CPP = /usr/local/bin/mpicxx

CCFLAGS = -DMPICH_IGNORE_CXX_SEEK
LIBS =
LDFLAGS = -lm

%: %.c
  $(CC) $(CCFLAGS) -c -o objects/$*.o $*.c
  $(CC) objects/$*.o $(LIBS) $(LDFLAGS) -o $(HOME)/bin/$*

%: %.cc
  $(CPP) $(CCFLAGS) -c -o objects/$*.o $*.cc
  $(CPP) objects/$*.o $(LIBS) $(LDFLAGS) -o $(HOME)/bin/$*
```

- Note the use of `mpicc/mpicxx` which just call gcc/g++ with some default MPICH libraries.
  - `mpiexec -n <# of processes> <executable>`
- You want to make sure that the executable is in the `$PATH`

Hello World output

- To run this, must use `mpiexec`
  - `mpiexec -n <# of processes> <executable>`
- Output of above hello world.

```
Process 0 of 4 is running on albert.phys.cmu.edu
Process 2 of 4 is running on qcd020.phys.cmu.edu
Process 1 of 4 is running on qcd022.phys.cmu.edu
Process 3 of 4 is running on qcd021.phys.cmu.edu
```

Basic calls

- Make sure you include `mpi.h`!
- `MPI::Status status`
  - Required for some calls, as we will see.
- `MPI::Init(argc, argv)`
  - Required for running MPI.
  - Needs `argc` and `argv`.
- `MPI::Finalize()`
  - Must call this at the end of the code to clean up lingering MPI calls, or executable will hang.
- When in doubt: `mpdhelp`

Basic calls

- `int MPI::COMM_WORLD.Get_size()`
  - Returns the number of processors.
- `int MPI::COMM_WORLD.Get_rank()`
  - Returns the rank of the current processor.
- `void MPI::Get_processor_name (char *name, int &namelength)`
  - Accesses the name of the machine on which that process is running.

Basic calls

- `int void MPI::COMM_WORLD.Send(const void* buf, int count, const Datatype& datatype, int dest, int tag) const`
  - Sends some buffer (`buf`) of some size (`count`) and some type (`datatype`) to some processor (`dest`) that matches some tag (`tag`).
- `void MPI::COMM_WORLD.Recv(void* buf, int count, const Datatype& datatype, int source, int tag, Status& status) const`
  - Recieves some buffer (`buf`) of some size (`count`) and some type (`datatype`) from some processor (`source`) that matches some tag (`tag`).
- Note that these *do not* return until the buffer has been fully sent/recieved.
- If you do not want to wait, use `Isend`/`Ireceive`
- See `helloWorldMPI2.cc`

Basic calls

- See helloWorldMPI2.cc

```
[bellis@albert ~]$ mpiexec -n 4 helloWorldMPI2
Greetings from process 1 - qcd022.phys.cmu.edu
Greetings from process 2 - qcd020.phys.cmu.edu
Greetings from process 3 - qcd021.phys.cmu.edu
```

Basic calls

- `int void MPI::COMM_WORLD.Bcast(void* buffer, int count, const MPI::Datatype& datatype, int root) const = 0`
    - Broadcasts some buffer (`sendbuf`) of some size (`count`) and some type (`datatype`) to all processor from a particular process (`root`).
- `int void MPI::COMM_WORLD.Reduce(const void* sendbuf, void* recvbuf, int count, const MPI::Datatype& datatype, const MPI::Op& op, int root) const = 0`
    - Receives some buffer (`sendbuf`) of some size (`count`) and some fills another buffer (`recvbuf`) from all processors and performs some operation (`op` SUM, PRODUCT, etc) on them when it fills.
- See `helloWorldMPI2.cc`

Calculate π example

- See `calcPiMPI.cc`

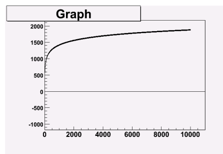
Fitting with Minuit

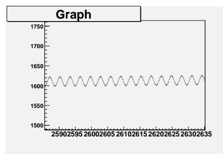
- Generate points along line

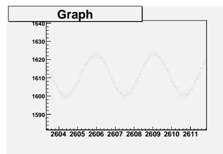
$$y = a \log x + b \sin^2(x) + c \cos^s(x)$$

  and smear with $\sigma = 1.0$

- Generated 1e6 and 1e7 events.







- Fit for a, b and c with Minuit
- See `fitMyExample.cc`

Skeleton of Minuit code

```
fcn()
{
  loop over each point and calc Chi2
}

main()
{
  Initialize minuit

  Read in the data points

  Call the minimization and fcn
}
```

Fitting with Minuit

- When it runs on one processor with 10,000,000 events
- 45-50 seconds read-in time
- 280 seconds to run.

Fitting with Minuit and MPICH2

- Must set up loop in `main`.
- Only run `Minuit` on master node.
- Calculate each point's χ^2 on the subnodes, but still in `main`.

Skeleton of Minuit code

```
fcn()
{
  if(rank == master)
   Send the current parameter values to each subnode
  else
   Receive the current parameter values

  if(rank == master)
   Receive the contributions to chisquare from each subnode
   and sum them
}
main()
{
  Initialize minuit
  if(rank == master)
   Read in the data points and send to subnodes
  else
   Receive the data points

  if(rank == master)
   Call the minimization and fcn
  else
   Caclulate each points contribution to chisquare
}
```

Fitting with MPI and Minuit

- When it runs on 21 processors with 10,000,000 events
- 45-50 seconds read-in time
- 15 seconds to run.
- Scales with number of processors!

Resources

- mpdhelp
- http://www.mpi-forum.org/
- http://www-unix.mcs.anl.gov/mpi/mpich/index.htm
- http://www.scs.fsu.edu/ burkardt/cpp_src/mpi/mpi.html
- http://www.llnl.gov/computing/tutorials/mpi/#Getting_Started
- http://www-unix.mcs.anl.gov/mpi/www/index.html
- http://www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-2.0/mpi2-report.htm#Node0
- http://webct.ncsa.uiuc.edu:8900/SCRIPT/MPI/scripts/serve_home
- http://beige.ovpit.indiana.edu/I590/node60.html