Outline

Organization

Summary from last class

Coding and running code time!
Homework 4 due next week.

Have you logged into Stampede and tried to run there? Please try asap (to give us time to work out issues).

Final projects: Expect individual feedback for your projects in the next couple of days.

Final project presentations: May 10/11 (most likely). Recall that you’ve to give a 10 minute presentation.

And please come to your colleagues’ presentations.
Planned material for remainder of course (you are welcome to give input!):

- Hybrid computing (MPI + OpenMP; today)
- GPU computing with OpenCL; possibly Intel Xeon Phi accelerators
- Algorithms: Multigrid (?) FMM (?)
- Tools: Some debugging; Visualization with paraview; load balancing tools (?)
- Homeworks: Expect either two more short homeworks, or one longer one.
Outline

Organization

Summary from last class

Coding and running code time!
MPI Barrier

Synchronizes all processes. Other collective functions implicitly act as a synchronization. Used for instance for timing.

`MPI_BARRIER(MPI_Comm communicator)`
MPI Broadcast

Broadcasts data from one to all processors. Every processor calls same function (although its effect is different).

MPI_Bcast(void* data, int count, MPI_Datatype datatype, int root, MPI_Comm communicator)

Actual implementation depends on MPI library.
MPI Broadcast

Broadcasts data from one to all processors. Every processor calls same function (although its effect is different).

`MPI_Bcast(void* data, int count, MPI_Datatype datatype, int root, MPI_Comm communicator)`

Actual implementation depends on MPI library.
MPI Reduce

Reduces data from all to one processors. Every processor calls same function.

MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm communicator)

Possible Reduce operators:
- MPI_MAX: Returns the maximum element.
- MPI_MIN: Returns the minimum element.
- MPI_SUM: Sums the elements.
- MPI_PROD: Multiplies all elements.
- MPI_LAND: Performs a logical and across the elements.
- MPI_LOR: Performs a logical or across the elements.
- MPI_BAND: Performs a bitwise and across the bits of the elements.
- MPI_BOR: Performs a bitwise or across the bits of the elements.
- MPI_MAXLOC: Returns the maximum value and the rank of the process that owns it.
- MPI_MINLOC: Returns the minimum value and the rank of the process that owns it.

MPI_Allreduce(): Provides result of reduction too all processors.
MPI Scatter

 Broadcasts different data from one to all processors. Every processor calls same function.

\[
\text{MPI Scatter(} \text{void* sendbuff, int sendcount, MPI Datatype sendtype, void* recvbuf, int recvcount, MPI Datatype recvtype, int root, MPI Comm communicator) } \]

Send arguments must be provided on all processors, but sendbuff can be NULL. Send/recv count are per processor.
Gathers **different** data from all to one processors. Every processor calls same function.

```c
MPI_Gather(void* sendbuff, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm communicator)
```

**Variant:**

```c
MPI_Allgather() gathers from all processors to all processors.
```
MPI_Bcast comparison

Let’s compare a naive implementation of MPI_Bcast with the system implementation:

https://github.com/NYU-HPC17/lecture8
MPI_Bcast comparison

Let’s compare a naive implementation of MPI_Bcast with the system implementation:

https://github.com/NYU-HPC17/lecture8

…and let’s do it on Stampede!
Submitting jobs on Stampede

Overview of HPC cluster

You

Login Nodes

Batch Scheduler

Shared Filesystem

Compute Nodes
Submitting jobs on Stampede

Stampede user guide:
https://portal.tacc.utexas.edu/user-guides/stampede

Batch facilities: SGE, LSF, SLURM. Stampede uses SLURM, and these are some of the basic commands:

- submit/start a job: `sbatch` `jobscript`
- see status of my job: `squeue -u` `USERNAME`
- cancel my job: `scancel` `JOBID`
- see all jobs on machine: `showq | less`
Submitting jobs on Stampede

Some basic rules:

- Don’t run on the login node!
- Don’t abuse the shared file system.
Submitting jobs on Stampede

Available queues on Stampede

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Max Runtime</th>
<th>Max Nodes/Procs</th>
<th>Max Jobs in Queue</th>
<th>SU Charge Rate</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>48 hrs</td>
<td>256 / 4K</td>
<td>50</td>
<td>1</td>
<td>normal production</td>
</tr>
<tr>
<td>development</td>
<td>2 hrs</td>
<td>16 / 256</td>
<td>1</td>
<td>1</td>
<td>development nodes</td>
</tr>
<tr>
<td>largemem</td>
<td>48 hrs</td>
<td>4 / 128</td>
<td>4</td>
<td>2</td>
<td>large memory 32 cores/node</td>
</tr>
<tr>
<td>serial</td>
<td>12 hrs</td>
<td>1 / 16</td>
<td>8</td>
<td>1</td>
<td>serial/shared_memory</td>
</tr>
<tr>
<td>large</td>
<td>24 hrs</td>
<td>1024 / 16K</td>
<td>50</td>
<td>1</td>
<td>large core counts (<a href="#">access by request</a>)</td>
</tr>
<tr>
<td>request</td>
<td>24 hrs</td>
<td>–</td>
<td>50</td>
<td>1</td>
<td>special requests</td>
</tr>
<tr>
<td>normal-mic</td>
<td>48 hrs</td>
<td>256 / 4K</td>
<td>50</td>
<td>1</td>
<td>production MIC nodes</td>
</tr>
<tr>
<td>normal-2mic</td>
<td>24 hrs</td>
<td>128 / 2k</td>
<td>50</td>
<td>1</td>
<td>production MIC nodes with two co-processors</td>
</tr>
<tr>
<td>gpu</td>
<td>24 hrs</td>
<td>32 / 512</td>
<td>50</td>
<td>1</td>
<td>GPU nodes</td>
</tr>
<tr>
<td>gpudev</td>
<td>4 hrs</td>
<td>4 / 64</td>
<td>5</td>
<td>1</td>
<td>GPU development nodes</td>
</tr>
<tr>
<td>vis</td>
<td>8 hrs</td>
<td>32 / 512</td>
<td>50</td>
<td>1</td>
<td>GPU nodes + VNC service</td>
</tr>
<tr>
<td>visdev</td>
<td>4 hrs</td>
<td>4 / 64</td>
<td>5</td>
<td>1</td>
<td>Vis development nodes (GPUs + VNC)</td>
</tr>
</tbody>
</table>
Submitting jobs on Stampede
Example job script (in git repo for lecture5)

#!/bin/bash
#SBATCH -J myMPI  \# job name
#SBATCH -o myMPI.o  \# output and error file name
#SBATCH -n 32  \# total number of mpi tasks
#SBATCH -p development  \# queue -- normal, development, etc.
#SBATCH -t 01:30:00  \# run time (hh:mm:ss) - 1.5 hours
#SBATCH --mail-user=username@tacc.utexas.edu
#SBATCH --mail-type=begin  \# email me when the job starts
#SBATCH --mail-type=end  \# email me when the job finishes
ibrun ./a.out  \# run the MPI executable
Outline

Organization

Summary from last class

Coding and running code time!
1D MPI Jacobi

- Blocking Send/Recv
- Nonblocking Send/Recv; overlapping computation and communication
- MPI-OpenMP hybrid on Stampede