Advanced Topics in Numerical Analysis: High Performance Computing

MATH-GA 2012.001 & CSCI-GA 2945.001

Georg Stadler
Courant Institute, NYU
stadler@cims.nyu.edu

Spring 2017, Thursday, 5:10–7:00PM, WWH #512

April 6, 2017

Outline

Organization

Summary from last class

Coding and running code time!

Organization

- 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.

Organization

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

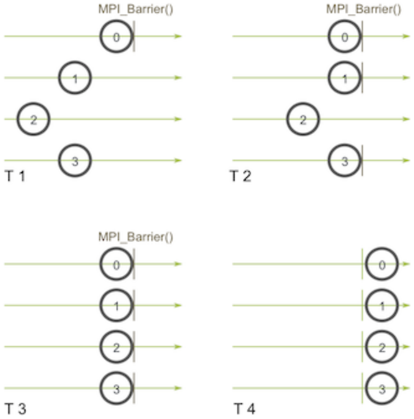
Summary from last class

Coding and running code time!

MPI Barrier

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

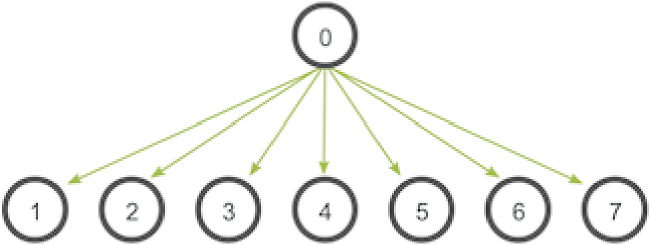
```
MPI_Barrier(MPI_Comm communicator)
```



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)
```

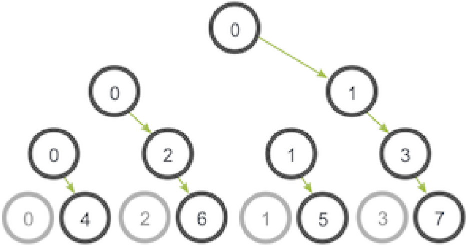


Actual implementation depends on MPI library.

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.

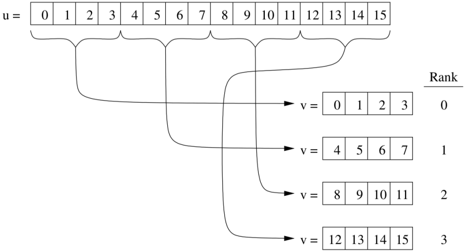
MPI_MINLOC: Returns the minimum value and the rank of the process that owns it.

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

MPI Scatter

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

```
MPI_Scatter(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 sendbuf can be NULL. Send/recv count are per processor.

MPI Gather

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

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

Variant:

`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

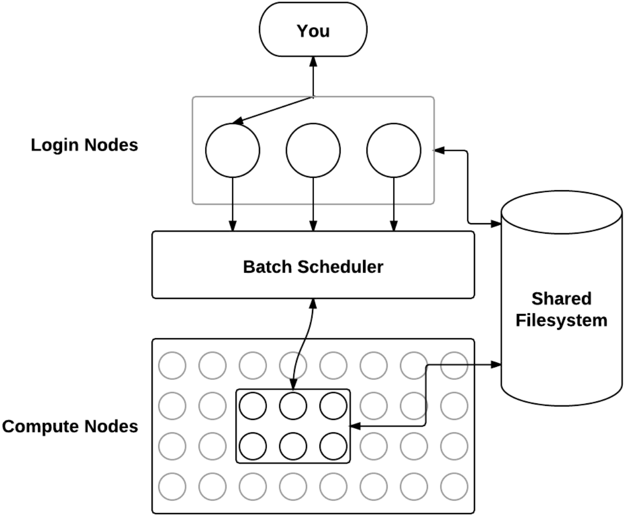
Let's compare a naive implementation of `MPI_Bcast` with the system implementation:

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

. . . and let's do it on Stampede!

Submitting jobs on Stampede

Overview of HPC cluster



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`

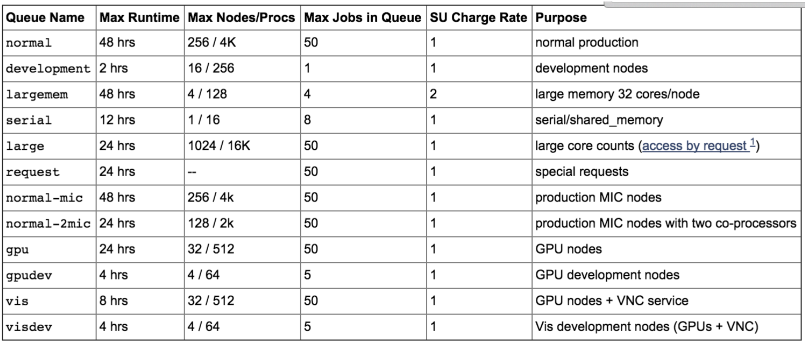
Submitting jobs on Stampede

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



Queue Name	Max Runtime	Max Nodes/Procs	Max Jobs in Queue	SU Charge Rate	Purpose
normal	48 hrs	256 / 4K	50	1	normal production
development	2 hrs	16 / 256	1	1	development nodes
largemem	48 hrs	4 / 128	4	2	large memory 32 cores/node
serial	12 hrs	1 / 16	8	1	serial/shared_memory
large	24 hrs	1024 / 16K	50	1	large core counts (access by request [1])
request	24 hrs	--	50	1	special requests
normal-mic	48 hrs	256 / 4k	50	1	production MIC nodes
normal-2mic	24 hrs	128 / 2k	50	1	production MIC nodes with two co-processors
gpu	24 hrs	32 / 512	50	1	GPU nodes
gpudev	4 hrs	4 / 64	5	1	GPU development nodes
vis	8 hrs	32 / 512	50	1	GPU nodes + VNC service
visdev	4 hrs	4 / 64	5	1	Vis development nodes (GPUs + VNC)

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