Advanced Topics in Numerical Analysis: High Performance Computing
MATH-GA 2012.003 and CSCI-GA 2945.003

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

Spring 2015, Monday, 5:10–7:00PM, WWH #202

Mar. 23, 2015
Outline

Organization issues

Last week summary

Submitting jobs through a scheduler

Debugging and profiling
Organization and today’s class

Organization:

- Homework #2 is due next week. Please finish Problem 1 (finding MPI bugs) before our class next week.
- I’ll show today how to run code on Stampede.
- Any homework questions?
Organization and today’s class

Organization:
- Homework #2 is due next week. Please finish Problem 1 (finding MPI bugs) before our class next week.
- I’ll show today how to run code on Stampede.
- Any homework questions?

Today:
- Wrap up collective MPI communication.
- Submitting jobs through a scheduler (on Stampede)
- Debugging/profiling
- valgrind & cachegrind
Outline

Organization issues

Last week summary

Submitting jobs through a scheduler

Debugging and profiling
MPI Collectives—Literature

Recommended online resource:  
http://mpitutorial.com/

Recommended reading: Chapter 5 in  
*Collective Communication: Theory, Practice, and Experience* by E. Chan, M. Heimlich, A. Purkayastha and R. van de Geijn  

MPI 3.0 Standard documentation  
http://www.mpi-forum.org/docs/mpi-3.0/
MPI Barrier

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

MPI_Bcast(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 bit-wise 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-HPC15/lecture5
MPI_Bcast comparison

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

https://github.com/NYU-HPC15/lecture5

...and let’s do it on Stampede!
Outline

Organization issues

Last week summary

Submitting jobs through a scheduler

Debugging and profiling
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

### 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>
#!/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 issues

Last week summary

Submitting jobs through a scheduler

Debugging and profiling
Debugging tools
... with emphasis on parallel debuggers

**Debugging** targets correctness of code

**Profiling** analyzes the code with target to make it faster

Compile with flag `-g` to generate source-level debug information

```
gcc -g code_with_bug.c
```

(Don’t) Use code optimization flags `-O1,-O2,-O3`

In **parallel**, studying program statistics can be useful, i.e.:
- memory usage per processor
- # of floating point operations
Debugging/profiling tools

- `printf` — fishing for bug, but sometimes useful
- `gdb` — GNU debugger: serial, but can also be attached to a parallel task
- `Totalview` (commercial debugger/profiler, available on many HPC resources)
- `DDT` (another commercial debugger/profiler; available on Stampede)
- `TAU`: Tuning and Analysis Utility
- `PAPI`: Performance Application Programming Interface
- `HPCtoolkit/perfexpert`: Suite of (open source) analysis and profiling tools
- `valgrind` (say: “val-grinned”) and `cachegrind`: memory/cache debugger and profiler
Debugging/profiling tools

Tau visualization with paraprof
Valgrind and cachegrind

Valgrind

- memory management tool and suite for debugging, also in parallel
- profiles heap (not stack) memory access
- simulates a CPU in software
- running code with valgrind makes it slower by factor of 10-100
- only available in development version on latest version of Mac OS
- Documentation: http://valgrind.org/docs/manual/

memcheck
finds leaks
inval. mem. access
uninitialize mem.
incorrect mem. frees

cachegrind

cache profiler
sources of cache misses

callgrind
extension to cachegrind
function call graph
Valgrind and cachegrind

Usage (see examples):

Run with valgrind (no recompile necessary!)

```bash
mpirun -np 2 valgrind --tool=memcheck [options] ./a.out [args]
```

Test examples for valgrind memcheck:

```bash
https://github.com/NYU-HPC15/lecture6
```

Valgrind can be linked against a wrapper MPI library to verify MPI communication; see

```bash
```
Valgrind and cachegrind

Run cachegrind profiler:
valgrind --tool=cachegrind [options] ./a.out [args]

Visualize results of cachegrind:
cg_annotate --auto=yes cachegrind.out***

To illustrate the use of cachegrind, we reuse the vector multiplication problem:

https://github.com/NYU-HPC15/lecture2

valgrind --tool=cachegrind
   ./inner-mem vec_size no_of_reps skip