A short parallel computing class for C/C++, Part II

Andrew W. Steiner

UTK/ORNL

July 7, 2023

Outline

- MPI
- Slurm

MPI I

- In MPI, different MPI ranks occupy entirely different memory spaces
- MPI tasks appear as separate processes in the operating system, while OpenMP threads do not create separate processes
- The principal way to communicate between MPI "ranks" is via `MPI_()` function calls
- MPI code is more difficult to write, but can be safer because the memory is separate
- Deadlocks and race conditions are still problems

MPI implementations

- A few leading MPI implementations, e.g. MPICH and Open MPI

MPI Compiling and execution

- A "compiler wrapper"

```
mpi_test: mpi_test.o
    mpic++ -o mpi_test mpi_test.o

mpi_test.o: mpi_test.cpp
    mpic++ -o mpi_test.o -c mpi_test.cpp
```

- Different execution pattern: e.g.

```
mpirun -np 4 ./mpi_test
```

MPI - Round Robin

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

using namespace std;

int main(int argc, char *argv[]) {

  int mpi_rank, mpi_size;

  // Init MPI
  MPI_Init(&argc,&argv);

  // Get MPI rank, etc.
  MPI_Comm_rank(MPI_COMM_WORLD,&mpi_rank);
  MPI_Comm_size(MPI_COMM_WORLD,&mpi_size);

  int tag=0, buffer=0;
  if (mpi_size>1 && mpi_rank>=1) {
    MPI_Recv(&buffer,1,MPI_INT,mpi_rank-1,
             tag,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }

  cout << mpi_rank << "/" << mpi_size << endl;

  if (mpi_size>1 && mpi_rank<mpi_size-1) {
    MPI_Send(&buffer,1,MPI_INT,mpi_rank+1,
             tag,MPI_COMM_WORLD);
  }

  MPI_Finalize();

  return 0;
}
```

What is this pattern useful for?

- Reading or writing large data files one rank at a time

MPI Blocking and Communicators

- The functions `MPI_Send` and `MPI_Recv` are "blocking", which means that execution does not proceed until the function is complete
- Communicators are separate MPI objects which manage communication between ranks
- For simple codes, MPI_COMM_WORLD is fine
- You can create sub-communicators for more complex task organization

MPI timing

- Use `MPI_Wtime()` to determine total running time
- This is important for managing output data at the end
- Output simulation data periodically, and well before the end of your allotted time

MPI Pattern 2: Task Manager and Workers

- See mpi_test2.cpp

Useful Tools

- `ldd`: print shared library dependencies
- `nm`: list symbols
- `ltrace` or `strace`: trace system calls
- `pstack, padb`: print stack trace of a running process, latter good with slurm
- `addr2line`: convert address into source code location
- `valgrind`: memory checker
- `gdb, gprof`: Debugger and profilers (but not necessarily parallel-aware
- Also: DDT, Totalview, etc. for debugging

Interactive jobs

- When compiling and executing a code for the first time, it is almost impossible to predict how (and if) it will run on a compute node
- Interactive jobs `interact` allow you to run on compute nodes. This uses your allocated computing time, but is invaluable
- On compute nodes: modules may have a different effect, file systems may look different, and compiler flags may need to be different
- On e.g. bridges, clearly specify the time, the project, the number of nodes/threads you need, the partition you will use, etc.
- On bridges 2, e.g. make sure to use `–ntasks-per-node`

CPU Architechture

- You may need to design your code around the CPU/GPU architechture
- tasks vs. cores vs. ranks vs. threads vs. nodes
- Cores: Typically the actual number of physical processors
- Tasks: On bridges, appears to be the same as cores, but can also be MPI tasks
- Ranks: Typically MPI tasks
- Threads: Typically OpenMP threads
- Nodes: Typically a large quantum of computing power, can be one or multiple CPUs, each with many cores.
- Bridges 2: one node, two CPUs, each with 64 cores

Slurm script

- See example `slurm.scr`

Group Exercises

- Go through Project #1, and prepare to explain how parallelism is used in anneal_para.h.
- Is there a place where the parallelism should be changed? What is the overhead associated with the parallelism in this code?
- For Project #1, propose a makefile target to compile and a separate makefile target to run the code on a non-HPC system
- For Project #1, propose a slurm script for bridges using 1 full node for 24 hours