MPI 1

CSCI 4850/5850 High-Performance Computing

Spring 2018

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
Learning Objectives

● Recap the architecture of distributed memory system and get started MPI.
Shared Memory System

- There is a single shared address space for all processors.
- All processors share the same view of memory.
Distributed Memory System

- Each processor has its own private memory.
- A network connects all the processors.

![Diagram of Distributed Memory System]
3-tier parallelism, hybrid architecture.
Node Parallelism

- One process per node.
- Private memory.
- Communication between nodes:
  - Message-passing.
  - Basic operations: send, receive.
  - Collective operations.

Message Passing Interface (MPI)

send(M) → M → receive(M)
Core Parallelism

- One thread per core.
- Shared-memory.
- Synchronization between cores:
  - Basic operations: fork, join.
  - Bulk-synchronous parallel (BSP).

OpenMP

```c
#pragma omp parallel for
for (i=0; i<N; i++)
  x[i] += y[i]
```
Accelerator Parallelism

- Massively parallel thread architecture.
- Shared memory.
- Offload mechanism to spawn kernel on accelerator.
- Manage memory copy operations.

```c
#pragma acc kernels
for (i = 0; i < N; i++)
    x[i] = foo(i)
```
Research of Parallel Programming

- Researches -> programming models, languages, and libraries
  - a prototype available and encourage use by others
  - Users: commitment, support, portability
  - Not all research groups can provide this

- Failure to achieve critical mass of users can limit impact of research
  - MPI and only few others succeeded
Standardization

- The community has resorted to “community- based” standards
  - MPI
  - OpenMP
  - CUDA, OpenCL → OpenACC

- Portable, portable, and portable
Standardization Benefits

- Multiple implementations promote competition
- Vendors get clear direction on where to devote effort
- Users get portability for applications
- Wide use consolidates the research, Prepares community for next round of research
Message-Passing Paradigm

- A parallel program is decomposed into processes, called ranks.
- Each rank holds a portion of the program’s data into its private memory.
- Communication among ranks is made explicit through messages.
- Channels honor first-in-first-out (FIFO) ordering.

- **Message passing** is for communication among processes, which have separate address spaces (vs share addr.)
Single-Program Multiple-Data (SPMD)

- All processes run the same program, each accesses a different portion of data.
- All processes are launched simultaneously.
- Communication:
  - Point-to-point messages.
  - Collective communication operations.
- Inter-process communication consists of
  - Synchronization
  - Interaction: movement of data from one process’s address space to another’s;
- SPMD (VS SIMD, MIMD)
  - Processors run a personal copy of a program
Features of Message Passing

- **Simplicity**: the basics of the paradigm are traditional communication operations.
- **Generality**: can be implemented on most parallel architectures.
- **Performance**: the implementation can match the underlying hardware.
- **Scalability**: the same program can be deployed on larger systems.
What is Message Passing Interface (MPI)?

- A message-passing library specification
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Designed to provide access to advanced parallel hardware for end users, library writers, and tool developers
Message Passing Interface (MPI)

- Standard for operations in message passing.
- Led by MPI Forum (academia & industry).
  - MPI-1 standard (1994)
  - MPI-2 standard (1997)
  - MPI-3 standard (2012)
- Implementations:
  - Open-source: MPICH, OpenMPI.
  - Proprietary: Cray, IBM, Intel.
- Languages:
- There is no automatic sequential-equivalent to an MPI program.
Novel Features of MPI

- **Communicators** encapsulate communication spaces for library safety
- **Datatypes** reduce copying costs and permit heterogeneity
- Multiple communication **modes** allow precise buffer management
- Extensive **collective operations** for scalable global communication
- **Process topologies** permit efficient process placement, user views of process layout
- **Good interface** encourages portable tools
MPI is simple

- Many parallel programs can be written using just these six functions, only two of which are non-trivial:
  - MPI_INIT
  - MPI_FINALIZE
  - MPI_COMM_SIZE
  - MPI_COMM_RANK
  - MPI_SEND
  - MPI_RECV
MPI Ranks

- Ranks have private memory.
- Each rank has a unique identification number.
- Ranks are numbered sequentially: [0, n-1].
MPI Primer

- Initialization:
  
  C:   int MPI_Init(int *argc, char ***argv)

  C++: void MPI::Init(int& argc, char**& argv)

- Get communicator size:
  
  int MPI_Comm_size(MPI_Comm comm, int *size)

  int Comm::Get_size() const

- Get rank number:
  
  int MPI_Comm_rank(MPI_Comm comm, int *rank)

  int Comm::Get_rank() const

- Finalization:
  
  int MPI_Finalize(void)

  void Finalize()
C, Fortran, C++

- Check https://www.open-mpi.org/doc/v2.0/

**Name**

`MPI_Init` - Initializes the MPI execution environment

**Syntax**

**C Syntax**

```c
#include <mpi.h>
int MPI_Init(int *argc, char **argv)
```

**Fortran Syntax**

```fortran
INCLUDE 'mpi.f'
MPI_INIT(IERRO)
   INTEGER IERROR
```

**C++ Syntax**

```cpp
#include <mpi.h>
void MPI::Init(int& argc, char**& argv)
void MPI::Init()
```
#include <iostream>
#include <cstdlib>              // has exit(), etc.
#include <mpi.h>                // MPI header file
#include <unistd.h>             // for sleep()

using namespace std;

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

    // initialize for MPI (should come before any other calls to
    //     MPI routines)
    MPI::Init(argc, argv);

    // get number of processes
    int nprocs = MPI::COMM_WORLD.Get_size();

    // get this process's number (ranges from 0 to nprocs - 1)
    int myid = MPI::COMM_WORLD.Get_rank();

    double starttime, endtime;
    if (myid == 0) {
        starttime = MPI::Wtime();
    }

    // print a greeting
    cout << "hello from process " << myid << " of " << nprocs << endl;
    sleep(5);

    if (myid == 0) {
        endtime = MPI::Wtime();
        cout << "Elapsed time = " << endtime-starttime << endl;
    }

    // clean up for MPI
    MPI::Finalize();

    return EXIT_SUCCESS;
}
MPI – How to build MPI binary?

[ahnt@apex Lab1]$ mpiCC
bash: mpiCC: command not found...
[ahnt@apex Lab1]$ which mpiCC
/usr/bin/which: no mpiCC in
(/cm/shared/apps/slurm/15.08.6/sbin:/cm/shared/apps/slurm/15.08.6/bin:/cm/local/apps/gcc/5.2.0/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/usr/sbin:/usr/sbin:/usr/sbin:/cm/local/apps/environment-modules/3.2.10/bin:/home/ahnt/.local/bin:/home/ahnt/bin)

[ahnt@apex Lab1]$ module list
Currently Loaded Modulefiles:
  1) gcc/5.2.0    2) slurm/15.08.6

[ahnt@apex Lab1]$ module available
[ahnt@apex Lab1]$ module add openmpi/gcc/64/1.10.1
[ahnt@apex Lab1]$ mpiCC -o mpihelloworld mpihelloworld.cpp
Lab 1 – Run mpi-helloworld at kepler.slu.edu

- [https://sites.google.com/a/slu.edu/atg/getting-started](https://sites.google.com/a/slu.edu/atg/getting-started)
- Prepare a script to submit the mpi helloworld program.
- Submit the job and check the result.
- Useful SGE commands:
  - `qhost`: You can see the status of available compute nodes
  - `qconf`: Modify and check the current SGE configuration
  - `qconf -sql`: show a list of all queues
  - `qconf -spl`: show all parallel environments
  - `qsub`: submit a batch job to Sun Grid Engine.
  - `qstat`: show the status of Sun Grid Engine jobs and queues
Lab: Run mpi-helloworld at apex.slu.edu

- [Link](https://sites.google.com/a/slu.edu/atg/apex)

- Prepare a script to submit the mpi helloworld program.

- Submit the job (`sbatch`) and check (`squeue`) the result.

- Useful SLURM commands:
  - `squeue` - lets you see what's in the SLURM queue
  - `squeue -u <username>` - as above but for a specific user
  - `scancel <jobid>` - cancels a certain job id
  - `scancel -u <username>` - cancels all jobs for a certain user