Week 11: Introduction to Message Passing Interface (MPI)

Computational Tools and Techniques in STEM

Apr 16-18, 2019

Learning Goals

- **L1**: Basics of high performance computing (HPC).
- **L2**: Distributed and shared memory programming.
- **L3**: Basics of parallelism.
- **L4**: Introduction to MPI.
- **L5**: Example.

High Peformance Computing (HPC)

It refers to using the combined power of computing resources for handling computation and data intensive tasks, such as data processing, visualization and running simulations.

Broadly speaking, there are two types of hardwares that are present:

1. Shared memory machines
2. Distributed memory clusters

Distributed and shared memory programming

Shared memory machines: RAM can be accessed by all the processing units.

Distributed memory machines

- Memory is inaccessible between different processing units, or nodes.
- When using a distributed memory setup, there has to be an interconnection between the nodes.
- When running distributed memory simulations over several nodes, the computational units must use the network to communicate.

Hybrid machines

- Modern HPC systems are a a hybrid of these two paradigms.
- Some units share a common memory space and some do not.
- Generally, MPI+X is used for hybrid programming.
- MPI for the inter-node communications and X within the same node.
- X can be OpenMP or some other shared memory programming paradigm.

Parallelism

Parallelism

- Some parallelism can improve performance of many applications, even without knowing the nitty-gritty of the underlying computer architecture.
- One way to “parallelize” your problem is simply to chop into pieces and run those pieces separately. This of course depends on problem complexity and inter-dependence among the pieces.

Speedup

- Ratio of execution time on the parallel system and the execution time on the serial system.
- The upper limit of the speed depends on how well the problem can be parallelized.

Degrees of parallelization

- Embarassingly parallel problems. E.g. random walk.
- Almost embarassingly parallel problems. E.g. numerical integral of a polynomial using the trapezoid rule.
- Communication intensive problems. E.g. solving partial differential equations numerically using a finite difference approach.

Let's see if we can parallelize the trapezoid problem that we did earlier. What category does it fall into?

Message Passing Interface (MPI)

Mutlicore machines: Machines containing more than one core. For e.g., our laptops!

Process: A process is a software notion, not the same thing as a processor, which is a more physical concept. We will be making use of multiprocessing to make use of multicore machines.

MPI is not a programming language but is a protocol for communicating between the processes. These processes can be either intra-node or inter-node.

Multiprocessing

- MPI spawns multiple processes on the system that run independently of each other.
- Each process has its own pool of memory that it does not share.
- There is an overhead with creating a process, but if there are enough calculations to be performed, it's worth the effort.
- The processes only share data when explicitly asked to do so via a communication call.
- Commuication in MPI is expensive!

Launching MPI processes

Given, N chips and M cores on each of those chips, then you can launch N*M MPI processes running at full speed.

- A single core can only run one process at a time.
- If you have multiple cores, each process will run on a separate core.
- If you ask for more processes than the available cores, it will still run, but with a lower efficiency.

Hyperthreading: It allows multiple processes to share the resources of a single core. The package that supports multiprocessing is mpi4py. It contains all the standard MPI calls.

Two types of parallelism

- Data parallelism: Each process does the same exact work but on a completely different data. This approach is also known as SPMD (single program multiple data). E.g. solving numerical p.d.e. using finite difference is often done using domain decomposition, which is basically just data division among processes.
- Task parallelism: Each process executes parts of the code that are independent of each other.

Note: We will be doing a few examples with data parallelism in this class.

Types of communication

Point to point communication: Transmission of data between a pair of processes, where one process sends and the other receives. We will be covering just a subcategory of this known as blocking communications.

Collective communication: Transmission of data between multiple processes of a group simultaneously. The mechanics are very similar to point to point communications.

Some MPI commands

from mpi4py import MPI: import module mpi4py

MPI_COMM_WORLD: it is the default communicator and includes all the MPI processes. Within a communicator, every process has its own unique rank.

MPI_COMM_WORLD.rank: ID of the current MPI process

MPI_COMM_WORLD.size: Total number of MPI processes
MPI.Init(): Module function to initialize MPI

MPI.Finalize(): Module function to finalize MPI

MPI send and recv

comm.send(buf, dest=0, tag=0)
buf: what needs to be sent
dest: process id of the destination process
tag: unique tag associated with the message

comm.recv(obj=None, source=0, tag=0, status=None)
This function returns the object to be received.
obj: object to be received
source: process id where the message is originatin from
tag: unique id of the message that it needs to be receiving

Exercise

- Run the file mpi_rank.py using the run script run_mpi_rank.slurm
- Run the file mpi_p2p.py using the run script run_mpi_p2p.slurm