Lecture 5 Summary

This lecture was talking about MPI (message passing interface). The topics involves principles of message-passing programming, the building blocks and MPI itself.

The first part was about principles of message-passing programming. The logical view of a message-passing supported machine has x processes and each process has its own exclusive address space. Each data element must belong to one of the partitions of the space, and all interactions require cooperation of two processes between these interactions. Mostly, message-passing programs uses asynchronous or loosely synchronous paradigms, and SPMD (single program multiple data) model.

There are three kinds of operations: non-buffered blocking message passing operations, buffered blocking message passing operations, and non-blocking message passing operations. For non-buffered blocking one, the send operation does not return until a matching receive has been encountered, which causes major issues of idling and deadlocks. For buffered blocking one, the send operation copies the data into the designated buffer and returns after the copy operation has been completed. However, there is a tradeoff between copying overhead and idling overhead, and deadlocks are still possible with this kind of operation because that receive operations still block. For non-blocking message passing operations, the send or receive operations return before it is semantically safe to do so. Therefore, non-blocking operations are generally with a check-status operation.

The last part was about MPI, which defines a standard library for message-passing for developing portable message-passing programs. And minimum of 6 routines is required to write a fully-functional message-passing programs: MPI\_init(), which initializes MPI, MPI\_Finalize(), which terminates MPI, MPI\_Comm\_size(), which determines the number of processes, MPI\_Comm\_rank(), which determines the label of calling process, MPI\_send(), which sends a “unbuffered/blocking” message, and MPI\_Recv(), which receives a “unbuffered/blocking” message. Communicators, which are MPI\_Comm functions for here, define a communication domain. Communicators are used as arguments to all message transfer MPI routines and a process can belong to many different domains. MPI\_Recv() and MPI\_Send() are for sending and receiving messages in MPI. These functions can deadlock, therefore avoid using in large scale codes. MPI also provides equivalent datatypes for all C datatypes. To exchange messages, MPI\_Sendrecv can be called.