CMSC416: Introduction to Parallel Computing

Topic: Collective operations

Date: 2/15/2024

Collective operations. all processes in the group call the communication routine, with matching arguments.

Point to point operations is single process to single process.
Collective operations have multiple processes either as the sending or as the receiver.
MPI_Barrier.
Wait until all processes in communicator reaches the barrier and then they can all proceed to next instruction.
MPI_Bcast.
Send data from root to all other processors. The receive part is still using the same MPI_Bcast. Root argument will differentiate operation of root vs receiver. Bcast must also use same buffer name for the same data.

Collective operations are implemented as point to point operations. Under the hood, MPI dissolves them into point to point messages.
The MPI library takes care of the abstraction.

MPI_Reduce.
Opposite of barrier. Suppose you have some data in each process, that you want to aggregate. The simplest aggregate is sum. Eg. collect data and sum them onto one process. You can bring them into one process.
Sendbuf must be valid on all processes. Even root because it is also contributing.
You can do custom operations witht the MPI_Op, not just sum.
Recvbuf only needs to exist on root, though when compiling, the recvbuf has to at least be defined on the other processes.
You can aggregate an array of numbers. Specified with the count argument.

MPI_Allreduce. If want to send result back to all processes. The result of the reduce is sent to all process.

MPI_Scatter
Scatter will sender to different processes, but it sends different parts. Must have same amount of data. Difference between bcast, is the data is different. Bcast will send same exact data to receivers. MPI_Scatter is also sent in order of MPI_Rank

MPI_Gather. Takes different data from many processes to root. Perhaps turn many different elements and combine them into an array on root. The array is not random order, it is done in order of MPI_Rank.

Double MPI_Wtime(void)
Start = MPI_Wtime()
End = MPI_Wtime()
elapsed time = start - end;
Time is in seconds.

Embarrassingly parallel program. Each loop iteration does not depend on other iterations.
You could round robin the iterations. Give one to each and go the the processes. Eg. 0 4 8 12 … goes to process 1. 1 5 9 … goes to process 2.
Another decomposition method is block division of work. 0-25 go to process 0 25-50 to process 1 etc.

In the end, we MPI_Reduce with MPI_SUM to get final answer. Without it, each process only has partial answer. We want to make sure the semantics of parallel program does not change from the serial version.