CMSC416 Introduction to Parallel Computing
Topic: Collective Operations
Date: 2/15/2024

Started with general announcements about Project1 & updates to the slides.

Definition: point-to-point operations occur between two individual processes.
Definition: collective operations occur between all processes within a given communicator.

At the base level, all collective operations are made up of many point-to-point operations. For example, with MPI_Bcast it is comprised of many process to process sends and receives.

MPI_Barrier(MPI_Comm comm)

This function can be used to synchronize processes if they are out of sync. Once a process reaches the barrier (executes the MPI_Barrier call), it will block until every other process within the communicator reaches and enters the barrier. At this point, the call is finished and processes will continue operating.

Question: can two processes have the same MPI barrier?
Answer: An MPI barrier call takes in a communicator. This barrier is shared among all processes in the communicator. If these two processes are within the same communicator then they will be using the same barrier.

MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

This function can be used to send data from a root process to all of the other processes in the communicator.

The buffer is used to transmit and spread the same piece of data to each process within the communicator. The sending process will first write data into its buffer. Then this data will appear in the receiving process's copy of the buffer.

MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

This function is used to collect and reduce data from all processes in the communicator and send it to the root process. It will reduce the data according to whatever operation was passed in through the ‘op’ parameter. For example, if I used a reduce call with the sum operation then each piece of data will get summed together with the end result being passed to the root process.

The sendbuf parameter should be valid for all processes involved. Each process can write their own unique data to this buffer. The recvbuf parameter must be valid for the root process that receives the data.

Similarly, the MPI_Allreduce function will collect and reduce data from processes to a root node, then it will redistribute this value back to all processes in the communicator.

MPI_Scatter(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

This function also acts as a way to send data. The difference between scatter and broadcast is that instead of sending one value to all processes, scatter will take an array of values and distribute them evenly between processes.

The amount of data to send from the root process is specified by the sendcount parameter. Similarly, all receiving processes will specify how much data it will accept through the recvcount parameter. The scatter call will see these counts and distribute the data points accordingly.

For each process that accepts data, including the root, the data will appear in the recvbuf buffer.

MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

This function acts as another way to gather data. Instead of reducing all data points into a single value, this function will append the data together into an array. The data will be appended in order of the processes's rank and will be sent to the root process.

MPI_Wtime(void)

This function returns the elapsed time between two points in time. It starts timing at the first wtime call, and ends at the next wtime call.