MPI Basics Cont.

MPI Communicators

Communicators are a set/group of processes numbered from 0 to n-1. Every program has a default communicator called `MPI_COMM_WORLD`, which is a group that contains all the processes. You can create sub-communicators that can contain groups of specific processes.

Example: 3x3 Board. We can assign sub-communicators to group each row, where each process in each communicator is labeled 0-2.

MPI Datatypes

MPI has default datatypes and derived/user-defined datatypes. Derived or user-defined datatypes can include things like user-defined structs, which allow for sending multiple datatypes at once.

Advanced MPI

Non-blocking calls

MPI_Send and MPI_Recv are blocking calls, meaning the program must wait until the message is sent/received. While waiting for a message, no other work can be completed during this time.

An alternative to this is to use non-blocking calls, which allow us to continue computing while communicating to other processes. The two routines we use to achieve this behavior are: `MPI_Isend` and `MPI_Irecv`. The procedure for using them can be broken into two parts:

1. Post the operation (`MPI_Isend, MPI_Irecv`)
2. Wait for results later using `MPI_Wait` or `MPI_Test`
    a. for every `MPI_Irecv` there should be a `MPI_Wait`/`MPI_Test` that waits to receive the data

Essentially, we are splitting a blocking operation (`MPI_Send, MPI_Recv`) into a non-blocking operation (`MPI_Isend, MPI_Irecv`) and a wait operation (`MPI_Wait, MPI_Test`) that receives the data.

Note that non-blocking calls vs blocking calls are for purely performance advantages. The semantics and correctness of the program should remain consistent.

MPI_Wait

We want to make sure that the call is completed, so we use `MPI_Wait` to ensure that we can use the received buffer.

Good practice is to have wait calls on every non-blocking operation, in order to ensure that the data is sent/received. This may be less important for non-blocking send operations, but is essential for non-blocking receive operations in order to ensure that the data we are receiving is in the receive buffer and can be used.

Example: 2D stencil computation

Consider a board split into 4 chunks row-wise. Each process must communicate with its two neighbors in order to send/receive the ghost rows it needs. Ex: Process 1 sends its top row to Process 0 and bottom row to Process 2. Process 1 requests the bottom row of Process 0 and top row of Process 2.

Note: Data being sent in MPI send operations must be contiguous in memory. So in order to send data that is not contiguous in memory, you need to manually create a buffer and send that instead.

To take advantage of the non-blocking calls. We should split the computation portion into two parts:

A. Compute on the cells that do not require additional communication between processes (innermost rows)
B. Compute on cells that require additional communication to neighboring processes in order to acquire necessary ghost row data to compute on cells (outermost rows)

In this scenario, we should attempt to do part A work while waiting to receive communication from neighboring processes in part B.

MPI_Test

`MPI_Test` is not a blocking call, so why would we use it? Well, we might not want to wait and instead want to do computation that requires a receive communication immediately when we receive it. We can use `MPI_Test` to do this by using a while loop where we compute non-required-communication work within the loop, but once the data is received we can do the required-communication work.

Collective operations

MPI_Barrier

`MPI_Barrier` is a blocking call used to synchronize the processes in a communicator. This is useful especially for asynchronous programs, because we force processes to wait until all other processes in the communicator have reached a certain point in the program (the routine).

MPI_Bcast

`MPI_Bcast` sends data from root to all other processes in a communicator. This operation is similar to send and receive calls, but now the root process only has to post one operation (`MPI_Bcast`) to send the data to all other processes in the communicator.