Review: MPI_Send and MPI_Recv Deadlocks

- In MPI, both MPI_Send and MPI_Recv functions are blocking operations. A deadlock occurs when two processes attempt to send messages to each other simultaneously, as neither can receive the other's message due to the blocking nature of these functions. To resolve this issue without , processes should alternate between sending and receiving operations.

MPI Communicators

- MPI Communicators represent a group or set of processes.
- Each process within a communicator is assigned a unique identifier from 0 to n-1, known as its rank.
- Every MPI program starts with the default communicator MPI_COMM_WORLD, which encompasses all processes and has a global rank.
- It is possible to create subcommunicators, which are subsets of the main communicator processes. These subcommunicators have their own local rank.

MPI Datatypes

- MPI supports various predefined data types, and it also allows for the creation of custom data types.
- To send multiple items/objects in a single message, one can create a C struct with multiple fields and then create an array of these struct objects.

Heterogeneous Processors and MPI

- MPI can handle systems with heterogeneous processors ("machines with different processors").
- Successful operation is generally contingent on the operating systems being the same across the different processors.

Downsides to Using Blocking Calls

- Blocking calls, such as those in MPI for sending or receiving messages, stop the processing of other tasks that can be serially processed. When a process is blocked, it cannot perform any other work that may need to be executed.

Non-blocking Calls in MPI

- Non-blocking calls were introduced after the initial design of MPI and are used for point-to-point communication.
- MPI_Isend and MPI_Irecv are the non-blocking versions of MPI_Send and MPI_Recv, respectively.
- Using non-blocking calls involves two steps:
    - Post the non-blocking operation
    - Wait for its results at a later point using MPI_Wait or MPI_Test. This allows a process to continue with other tasks until it's ready to block and wait for the results

        - MPI_Wait is used with the same request object provided to the send function and can also return a status object, which provides information about the communication. If the status is not needed, use MPI_STATUS_IGNORE.
- The advantage of non-blocking communication is that it allows for tasks not requiring communication to be processed concurrently.

2D Stencil Computation

- In 2D stencil computation, each process must send its top and bottom layers to other processes to ensure all necessary data (ghost layers) is available.
- Generally, you want to post, receives first, ensuring that whenever other processes send messages, the receiving process is ready to accept them
    - This is more efficient then sending then receiving
- Data can be stored in column-major (consecutive elements in columns stored adjacently) or row-major (consecutive elements in rows stored adjacently) order.
- To avoid waiting for sends to complthete, data can be copied into two buffers.
- The middle rows can be computed post-send and receive, as they do not require communication, while the first and last rows should be computed after the wait, since they require communication.
- How memory is handled also effects performance
    - With column major, consecutive elements in columns are stored next to each other
    - With row major, consecutive elements in rows are stored next to each other

Other MPI Calls

- MPI_Test is used to check if an operation is completed, indicated by a flag set to true.

Collective Operations

- Collective operations involve all processes in a communicator.
- MPI_Barrier is a blocking call that ensures all processes in the communicator reach a certain point before any can proceed. It helps synchronize work across processes.
- MPI_Bcast is used to send data from a root process to all other processes in the communicator. It's important to use the correct MPI_Comm with this operation to set the level of communication.
- int MPI_Reduce
    - reduce data from all processes to the root
    - sendbuf should be valid on all processes
    - recvbuf only needs to exist on root
- MPI_Allreduce
    - If you want to send the result back to all processes
    - Scatter
        - different than broadcast because you want to send different data to different processes
    - Gather

- gather all data from all processes to the root

double MPI_Wtime(void)

- returns elapsed time
- see code on slides
- starttime = MPI_Wtime();
- enddtime = MPI_Wtime();