MPI: Terminology and Examples

Aaron Weeden
Shodor Education Foundation, Inc.
2015

Review

It is recommended to first review the slides on Parallel Computing and OpenMP. They cover some key terms that will be used in these slides.

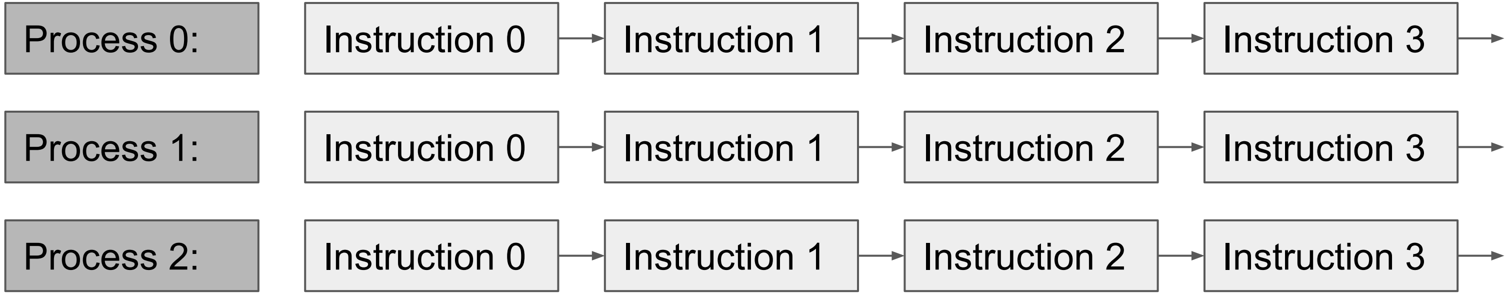
MPI

- Standard for distributed memory parallelism.
- Allows for multiple nodes (or just multiple cores) to run a program in parallel.
- Stands for Message Passing Interface (more on that on the next slide).
- Utilizes function calls as opposed to compiler directives.
- Syntax example: send a message:

```
MPI_Send(&buffer, count, MPI_INT, destination, tag,
    MPI_COMM_WORLD);
```

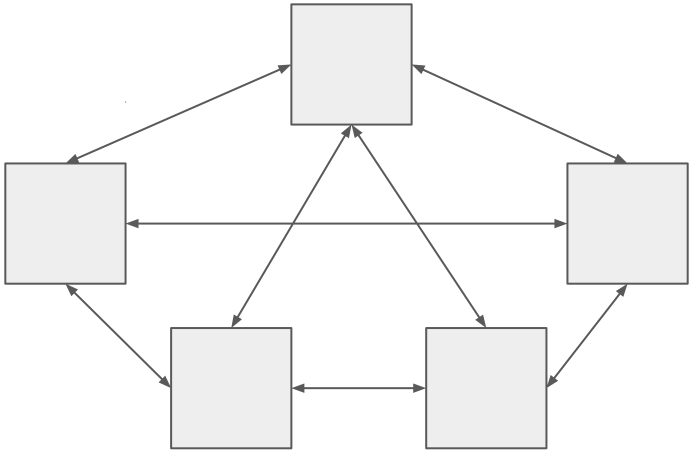
Key Term: **Process**

- MPI entity that can use a core to execute instructions.
- Does NOT share memory with other processes.
- Can send and receive messages to and from other processes (hence Message Passing Interface).
- Each process executes a copy of the same program.



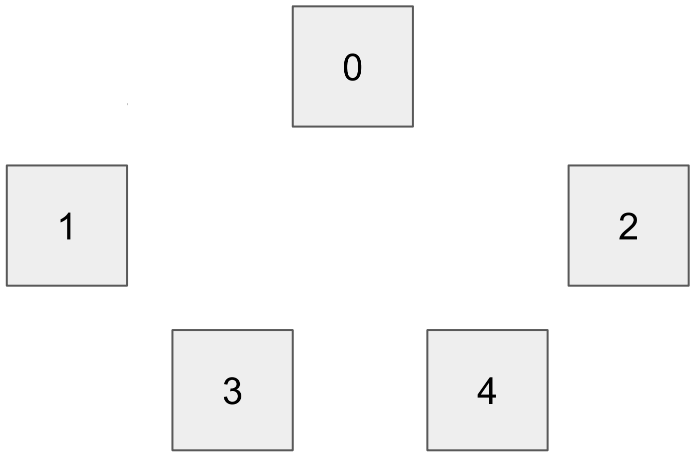
Key Term: **Communicator**

- A collection of MPI processes that can send and receive messages to and from each other.
- Normally this is all of the processes, and there is a constant defined for it, `MPI_COMM_WORLD`.



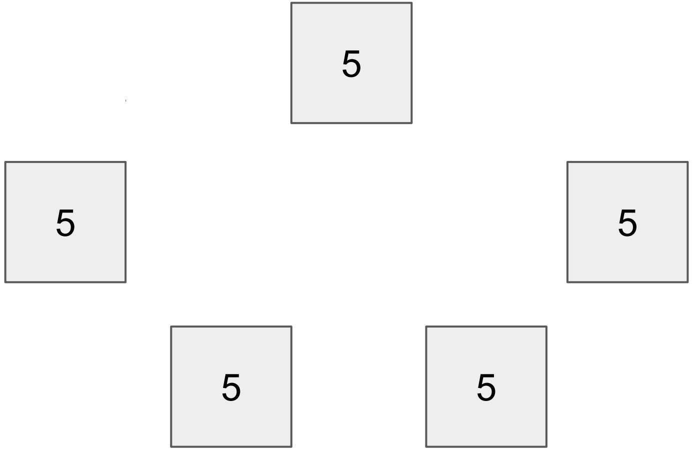
Key Term: **Rank**

- Unique identifier for each process in the communicator.
- Usually an integer starting at 0 and counting upwards.



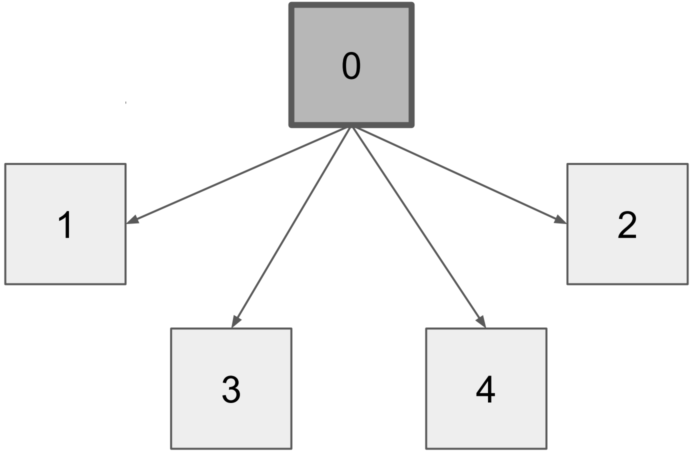
Key Term: **Size**

- Number of processes in a communicator.
- Same for all processes in the communicator.



Key Term: **Boss/Master**

- Optional, single process in the communicator with different tasks to do than the others.
- Sometimes assigns work to the other processes (hence **boss**).
- Usually rank 0.



MPI Program Execution

- Each process executes a copy of the same program with a different value for the rank.
- If only certain processes should do certain instructions, use the rank to distinguish these.
- Example:

```
if (rank == BOSS) {
  /* send work */
}
else {
  /* receive work */
}
```

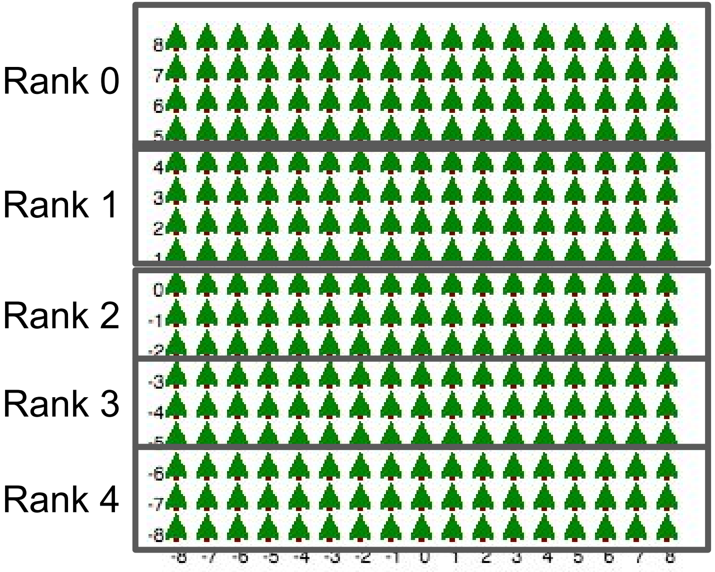
The 6 Basic MPI Routines

1. Initialize the communicator.
2. Set my rank.
3. Set the size.
4. Send a message.
5. Receive a message.
6. Finalize the communicator.

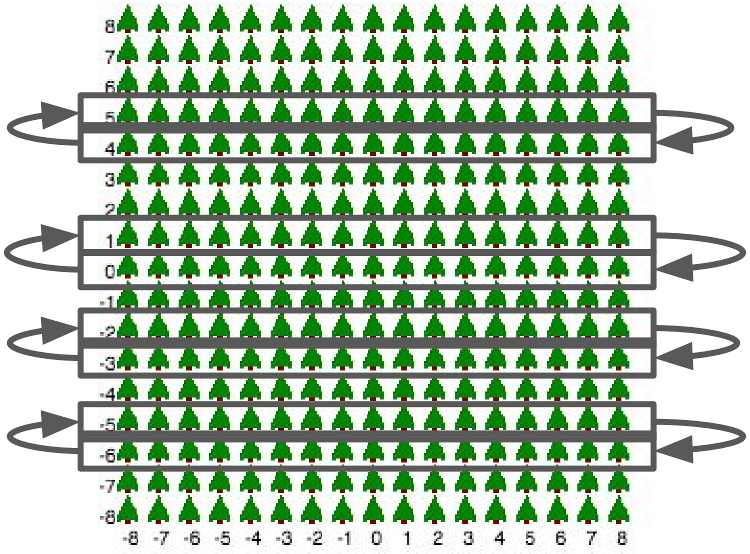
MPI Algorithm for Forest Fire Model

- Same basic model as OpenMP version, with a few extra tasks (shown below in blue).
- Data
  - **Trees** (for checking trees)
  - **NewTrees** (for changing trees)
- Tasks
  - **DistributeRows**: Each process assigns itself some of the rows of the forest.
  - **InitData**: One of the processes lights the center tree on fire.
  - For each time step:
    - **ContinueBurning**: For trees already burning that haven't burnt out, burn another step.
    - **CommunicateBoundaries**: Each process communicates tree data to its neighbor processes.
    - **BurnNew**: For trees next to a burning neighbors, catch on fire with some probability.
    - **AdvanceTime**: Copy NewTrees into Trees.

MPI Algorithm for Forest Fire Model



DistributeRows

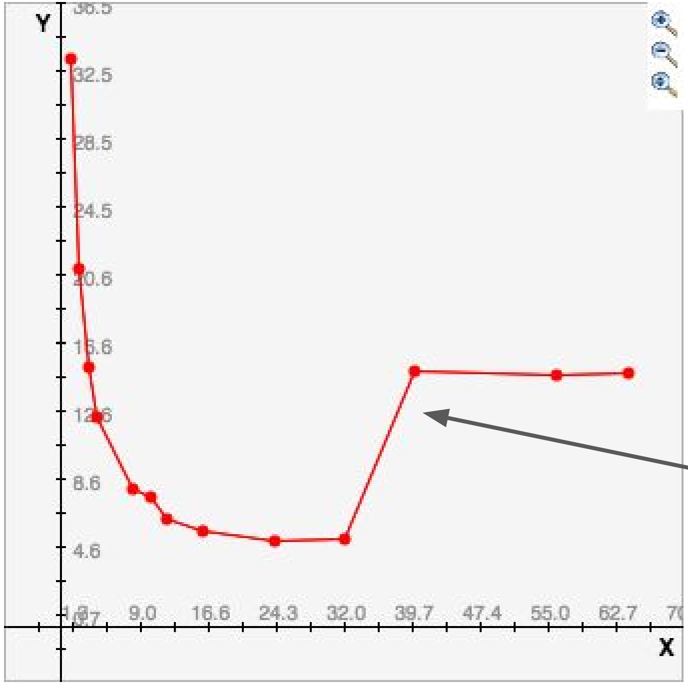


CommunicateBoundaries

Differences with OpenMP

- Processes instead of threads.
- Memory is distributed, not shared.
- For a process to read a value in another process' memory, a message has to be sent and received.
- No need for locks, because one process cannot read another process' memory directly.
- Requires writing more code (not just sticking directives above `for` loops).
- Allows a program to be scaled across more than 1 node.

Strong Scaling of MPI Forest Fire

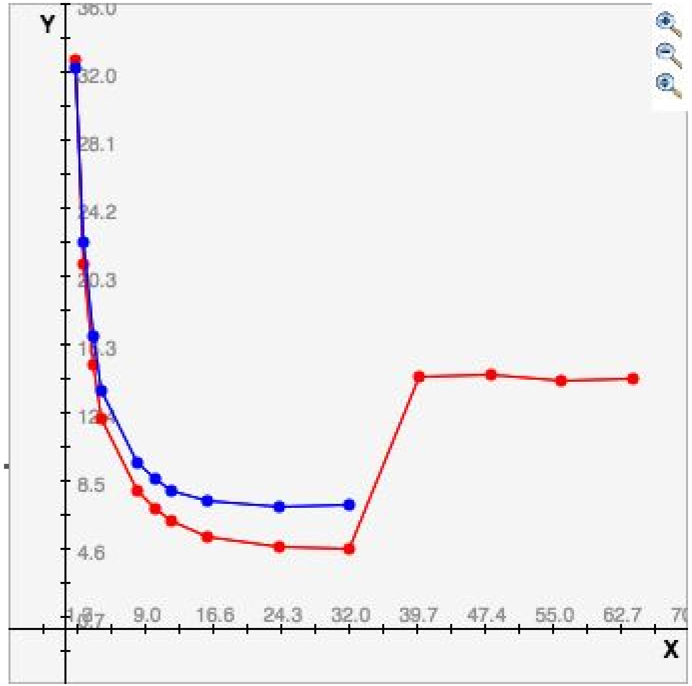


Jump from using 1 node to using 2 nodes : same flat line with extra time for inter-node communication

Strong Scaling Comparison of OpenMP and MPI

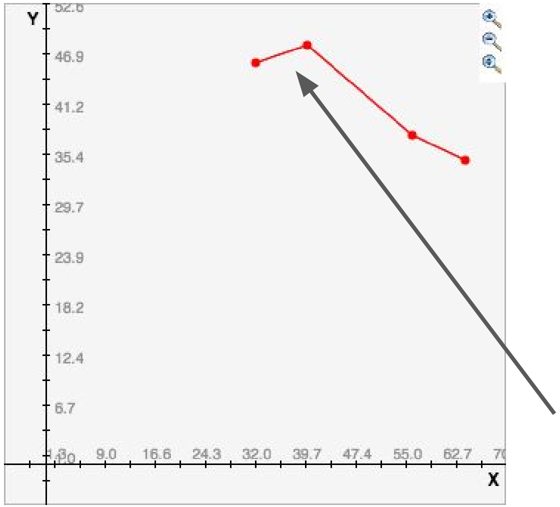
- MPI: Red
- OpenMP: Blue

- Note: not every MPI program is faster than its OpenMP counterpart, though that is the case here.



Strong Scaling with a Bigger Problem Size

- Before the problem size was 1300 rows, columns, and time steps.
- Now the problem size is 4000 rows, columns, and time steps.
- Just looking at process counts of 32, 40, 56, and 64, we see a slight slowdown from 32 to 40, but then a continual speedup.
- OpenMP is unable to scale beyond 1 node (32 cores for Blue Waters), so it cannot achieve the speed of 64 MPI processes for this problem size.

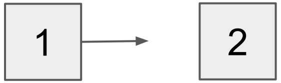


Jump from using 1 node
to using 2 nodes

MPI Example Communication: **Send**

- One process sends a buffer of data to another process.
- Forest fire example: process 1 sends its bottom row of trees to process 2.
- C syntax:

```
MPI_Send(SendBuf, /* buffer */
 NColsPlusBounds, /* count */
         MPI_INT, /* type */
               2, /* rank of receiver */
               0, /* message tag */
 MPI_COMM_WORLD); /* communicator */
```



MPI Example Communication: **Receive**

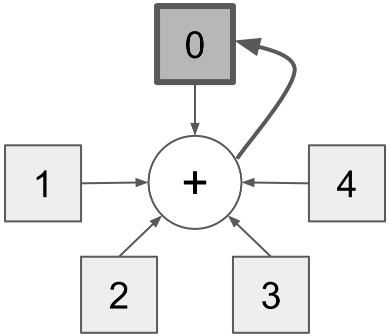
- One process receives a buffer of data from another process.
- Forest fire example: process 2 receives its top row of trees from process 1.
- C syntax:

```
MPI_Recv(RecvBuf, /* buffer */
 NColsPlusBounds, /* count */
         MPI_INT, /* type */
               1, /* rank of sender */
               0, /* message tag */
  MPI_COMM_WORLD, /* communicator */
        &status); /* struct with info about sent message */
```

MPI Example Communication: **Reduce**

- All processes contribute a buffer of data.
- An operation is performed on the data (e.g. sum, min, max).
- One process receives the result.
- Forest fire example: sum the number of burned trees.
- C syntax:

```
MPI_Reduce(&NBurnedTrees, /* send buffer */
                      &i, /* receive buffer */
                       1, /* count */
                 MPI_INT, /* type */
                 MPI_SUM, /* operation */
                       0, /* rank of receiver */
          MPI_COMM_WORLD); /* communicator */
```



MPI Example Communication: **Broadcast**

- One process sends a buffer of data to all processes (including itself).
- All processes replace their copy of the buffer with the sender's copy.
- Forest fire example: boss broadcasts whether input parameters are valid.
- C syntax:

```
MPI_Bcast(&AreParamsValid, /* buffer */
                        1, /* count */
                  MPI_INT, /* type */
                        0, /* rank of sender */
           MPI_COMM_WORLD); /* communicator */
```