Introduction to Parallel Processing

Exam Review

Professor Amanda Bienz

Exam Format

- 50 minutes, in-person, in-class
- Written
- Short answers (not multiple choice, multiple answer, fill-in-the-blank, etc)
- Covers MPI, all lectures up to Monday Oct 2 (Performance Modeling)
  - Content from homework 2 and 3

Point-to-Point

- Have one process post MPI_Send while another posts MPI_Recv
- Data is split into packets and sent across the links of the network
- Send may wait for matching receive to be posted, depending on implementation
  - Eager (small messages)
  - Rendezvous (large messages)

Point-to-Point

- Have one process post MPI_Send while another posts MPI_Recv
- Data is split into packets and sent across the links of the network
- Send may wait for matching receive to be posted, depending on implementation
  - Eager : sending process assumes necessary buffer space is available and goes ahead and sends the message
  - Rendezvous : cannot assume buffer space is available, tells the receiving process it wants to send a message of this size and waits for receiving process to say it’s ready before sending data

Non-Blocking

- Have one process post MPI_Isend while another posts MPI_Irecv
- When do calls to MPI_Isend and MPI_Irecv return?
- Can you reuse buffers before wait?
- MPI_Send = MPI_Isend + MPI_Wait
- MPI_Recv = MPI_Irecv + MPI_Wait

Non-Blocking

- Have one process post MPI_Isend while another posts MPI_Irecv
- Both MPI_Isend and MPI_Irecv return immediately, regardless of whether matching operation has been posted
- However, you cannot use the send buffer or the receive buffer until after you have called wait
- MPI_Send = MPI_Isend + MPI_Wait
- MPI_Recv = MPI_Irecv + MPI_Wait

Does this code work?

Assume each process has the following code:

```
double val = rand() / RAND_MAX;
if (rank % 2) proc = rank - 1;
else proc = rank + 1;
MPI_Send(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD);
MPI_Recv(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
```

What if we post Recv first?

Assume each process has the following code:

```
double val = rand() / RAND_MAX;
if (rank % 2) proc = rank - 1;
else proc = rank + 1;
MPI_Recv(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
         MPI_STATUS_IGNORE);
MPI_Send(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD);
```

How about sending with Isend?

Assume each process has the following code:

```
double val = rand() / RAND_MAX;
if (rank % 2) proc = rank - 1;
else proc = rank + 1;
MPI_Isend(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        &send_request);
MPI_Wait(&send_request, MPI_STATUS_IGNORE);
MPI_Recv(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
```

How about sending with Isend?

Assume each process has the following code:

```
double val = rand() / RAND_MAX;
if (rank % 2) proc = rank - 1;
else proc = rank + 1;
MPI_Isend(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        &send_request);
MPI_Recv(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
MPI_Wait(&send_request, MPI_STATUS_IGNORE);
```

How about using only Irecv?

Assume each process has the following code:

```
double val = rand() / RAND_MAX;
if (rank % 2) proc = rank - 1;
else proc = rank + 1;
MPI_Send(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD);
MPI_Irecv(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        &recv_request);
MPI_Wait(&recv_request, MPI_STATUS_IGNORE);
```

How about using only Irecv?

Assume each process has the following code:

```
double val = rand() / RAND_MAX;
if (rank % 2) proc = rank - 1;
else proc = rank + 1;
MPI_Irecv(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD,
        &recv_request);
MPI_Send(&val, 1, MPI_DOUBLE, proc, 1234, MPI_COMM_WORLD);
MPI_Wait(&recv_request, MPI_STATUS_IGNORE);
```

Going back to Isend…

Assume each process has the following code:

```
for (int i = 0; i < n_msgs; i++)
{
    val = send_data[i];
    MPI_Isend(&val, 1, MPI_DOUBLE, i, 1234, MPI_COMM_WORLD,
            &(send_requests[i]));
    MPI_Irecv(&recv_data[i], 1, MPI_DOUBLE, i, 1234,
            MPI_COMM_WORLD, &(recv_requests[i]));
}
MPI_Waitall(n_msgs, send_requests, MPI_STATUSES_IGNORE);
MPI_Waitall(n_msgs, recv_requests, MPI_STATUSES_IGNORE);
```

Going back to Isend…

Assume each process has the following code:

```
for (int i = 0; i < n_msgs; i++)
{
    MPI_Isend(&send_data[i], 1, MPI_DOUBLE, i, 1234,
            MPI_COMM_WORLD, &(send_requests[i]));
    MPI_Irecv(&recv_data[i], 1, MPI_DOUBLE, i, 1234,
            MPI_COMM_WORLD, &(recv_requests[i]));
}
MPI_Waitall(n_msgs, send_requests, MPI_STATUSES_IGNORE);
MPI_Waitall(n_msgs, recv_requests, MPI_STATUSES_IGNORE);
```

Collectives

- May require a barrier, or all processes to start the operation before any can complete
- Implemented very efficiently… you don’t need to memorize the algorithms but do need to know what the collective does
  - E.g. Know an Allreduce reduces data from all processes, gives results to all processes. Don’t need to know that this is implemented via recursive-doubling or ring algorithms.

MPI_Reduce

Which is faster? Reducing a list and then summing the final reduction, or summing locally and then reducing local sums?

1

```
for (int i = 0; i < n; i++)
    local_sum += list[i];
MPI_Reduce(&local_sum, &sum, 1, MPI_DOUBLE, MPI_SUM, 0,
        MPI_COMM_WORLD);
```

2

```
MPI_Reduce(list, sum_list, n, MPI_DOUBLE, MPI_SUM, 0,
        MPI_COMM_WORLD);
if (rank == 0)
    for (int i = 0; i < n; i++)
        sum += sum_list[i];
```

MPI_Barrier

What if I have process 0 send a message to every other process:

```
if (rank == 0)
{
    for (int i = 1; i < num_procs; i++)
    {
        MPI_Send(&i, 1, MPI_INT, i, 1234, MPI_COMM_WORLD);
    }
}
else MPI_Recv(&val, 1, MPI_INT, 0, 1234, MPI_COMM_WORLD);
```

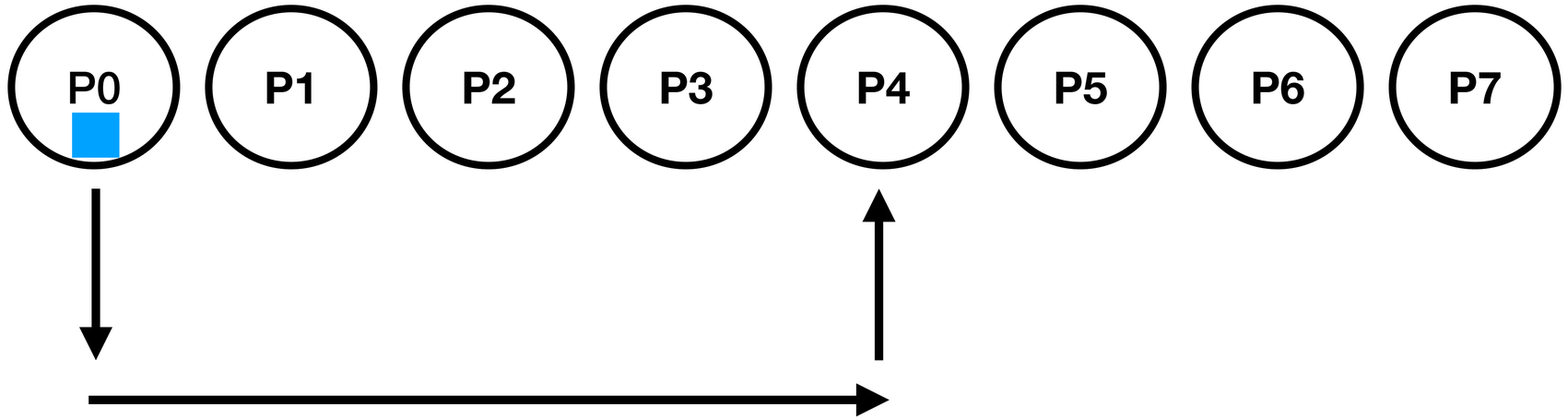
MPI_Barrier

Does this work correctly for process 0? Does it have to wait for every other process?

```
if (rank == 0)
{
    for (int i = 1; i < num_procs; i++)
    {
        MPI_Send(&i, 1, MPI_INT, i, 1234, MPI_COMM_WORLD);
    }
}
else MPI_Recv(&val, 1, MPI_INT, 0, 1234, MPI_COMM_WORLD);
```

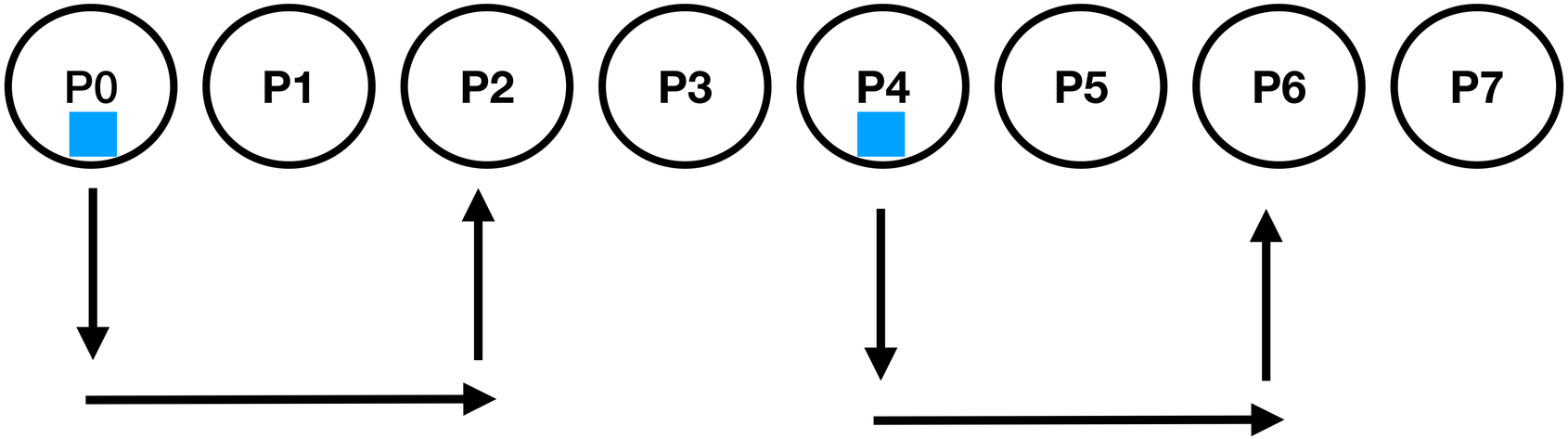
Will a binomial tree algorithm create a barrier?

Binomial Tree Algorithm



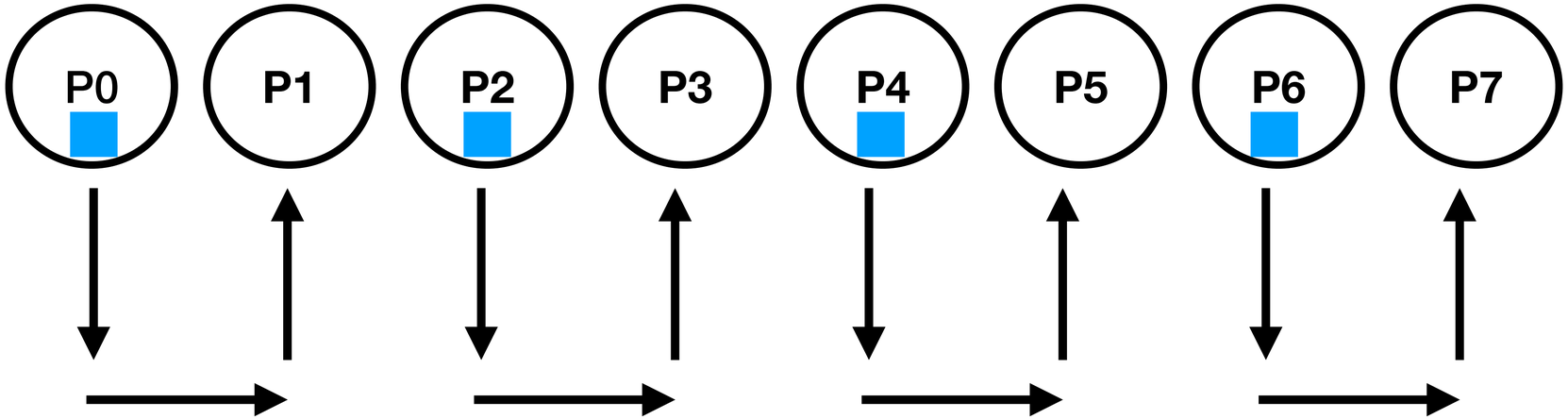
Will a binomial tree algorithm create a barrier?

Binomial Tree Algorithm



Will a binomial tree algorithm create a barrier?

Binomial Tree Algorithm



Which of these create a barrier?

- MPI_Bcast
- MPI_Reduce
- MPI_Scatter
- MPI_Gather
- MPI_Allreduce
- MPI_Allgather
- MPI_Alltoall
- MPI_Alltoallv

Synchronization costs?

- Let’s think about the MPI_Bcast
- This does not create a barrier
- But what about synchronization…
  - Is there an additional synchronization cost on all processes?
  - On any processes?
  - Why do we talk about synchronization being so costly for collectives that do not create a barrier?

Performance Models

- Postal model : T = alpha * n + beta * s
- Does the latency or bandwidth dominate the cost of messages?
- If I use the postal model to estimate the cost of an MPI_Allgather operation, what is missing from the model?
- If I use the postal model to estimate the cost of an MPI_Allreduce, what else is missing from the model?

Performance

- We know, based on ping-pong tests, that there is a difference between intra-socket, intra-node, and inter-node
- Intra-socket : processes on socket share cache
- Intra-node : processes on node share main memory
- Inter-node : injected into the network, sent across links of the network

Performance

- Let’s assume process 0 sends a single MPI_FLOAT to process p.
- Which is cheaper:
  - Process p is on the same socket as process 0
  - Process p is on the same node, but different socket than process 0

Performance

- Let’s assume process 0 sends **1 million** MPI_FLOAT values to process p.
- Which is cheaper:
  - Process p is on the same socket as process 0
  - Process p is on the same node, but different socket than process 0