ICS 632
Hualiang Li
Assignment 3

Part 1: Simgrid
Activity #1:

Type	Wall-clock Time
Default_bcast	1.375 seconds
Naïve_bcast	4.335 seconds
Ring_bcast	4.175 seconds

Table 1. Wall-clock Time for three different implementation on 50-processor ring.

From the table, I can see that the naïve broadcast and ring broadcast is 3 times long as the default broadcast.

Moreover, we can see that the ring broadcast has no significant improvement compared with the naive broadcast. This is because the processor does not send the message until all the message is received from its previous processor. So, there is no pipeline at all in this implementation. And because the network latency, it does not improve the performance.

Activity #2:

	20-Processor	35-Processor	50 Processor
100000	0.408	0.416	0.429
500000	0.227	0.236	0.249
1000000	0.211	0.226	0.241
5000000	0.250	0.316	0.383
10000000	0.320	0.450	0.580
50000000	0.896	1.537	2.177
100000000	1.619	2.897	4.175

Table 2. Wall-clock Time for three ring platforms with different chunk size.

From the table, we can see that the best chunk size for all kinds of ring are all 1000000.

The best chunk size in this case does not depend on the platform. We can observe that the wall-clock time is minimized for chunk size that is neither smallest nor largest. This is because when the chunk size is too small, the MPI and network latency overhead is too huge. When the chunk size is too big, the parallelization is bad.

For a 100-processor ring, when the chunk is 1000000, the wall-clock time is 0.317 seconds. When the chunk size is 100000000, the wall-clock time is 8.436. The speedup is about 27.

The performance compared to the default broadcast is much better. The speed up is about 5.

The pipeline communication for the purpose of ring-based broadcast on a ring-shaped physical platform is much better than the default implementation of broadcast.

Activity #3:

Chunk size	Wall-clock time
100000	0.228
500000	0.141
1000000	0.145
5000000	0.299
10000000	0.501
50000000	2.133
100000000	4.174

Table 3. Wall-clock time for asynchronous pipelined ring broadcast.

From the table, we can see the best chunk size for asynchronous pipeline ring broadcast is 500000. This is different from the regular pipeline implementation.

When using MPI_Isend the best wall-clock time is 0.141 seconds, while using MPI_Send, the wall-clock time is 0.241 seconds. The performance improves about 41%.

The asynchronous pipelined ring broadcast is fast than the default broadcast. The speedup is about 9.75 compared to the default one.

The asynchronous pipeline communication for the purpose of ring-based broadcast on a ring-shaped physical platform is much better than the default implementation of broadcast.

Activity #4:

Implementation type	Wall-clock Time
Asynchronous pipeline ring bcast	0.241
Asynchronous pipeline bintree bcast	2.398

Table 4. Wall-clock time on ring platform for different implementation.

Implementation type	Wall-clock Time
Asynchronous pipeline ring bcast	0.982
Asynchronous pipeline bintree bcast	0.199

Table 5. Wall-clock time on tree platform for different implementation.

From the table 4, we can see that the ring bcast is much fast than the bintree bcast on a ring platform. This is expected because the physical platform match the implementation, which make the implementation taking the advantage of the communication channel.

From the table 5, we can see that, the bintree bcast is faster than the ring bcast on a ring platform. However, the advantage is much smaller compared to it on ring platform. I think it is still worth to implement my own bintree broadcast on the bintree platform, because the ring broadcast is about 5 times faster.

Activity #5:

Implementation type	Wall-clock Time
Asynchronous pipeline ring bcast	0.325
Asynchronous pipeline bintree bcast	0.422
Default bcast	0.196

Table 6. Wall-clock time on crossbar platform for different implementation.

Implementation type	Wall-clock Time
Asynchronous pipeline ring bcast	2.692
Asynchronous pipeline bintree bcast	2.627
Default bcast	2.528

Table 7. Wall-clock time on backbone platform for different implementation.

From table 6 and 7, we can see that the default broadcast is as good as the ring broadcast and bintree broadcast. Therefore, with the reality that the real world has different physical network channel model, it is not worth to implement our own broadcast. However, if the physical network topology is fixed, we can implement a certain model to improve the performance.

Conclusion:

In this assignment, the most difficult part for me is how to visualize the network topology in favor of our own implementation. To solve this issue, I draw pictures to make my implementation easier. Also, I found the parameter for the MPI API is confusing me because different API call has different parameter. In particular, the MPI_Bcast do both send and receive for the sender and receiver, which surprised me a lot. I thought I needed MPI_Recv to receive the MPI_Bcast message, which is not true.

A potential improvement for the MPI I think of is, we can implement an iteration for different network topology, such that we do not have to consider the different toplogy. Something like:

```
MPI_Send(buffer, size, type, it->next, ….)
MPI_Recv(buffer, size, type, it->pre, …)
```

Part 2: Cray

I run a data size of 10^9 bytes, and use chunk size of 1000000.

Implementation type	Wall-clock Time
Default bcast	3.23
Naïve_bcast	27.1
Asynchronous pipeline ring bcast	1.92
Asynchronous pipeline bintree bcast	8.64

Table 8. Wall-clock time for different implementation on Cray.

From table 8, we can see that the default broadcast is much better than my own implementation of naïve bcast and 3 time faster than the bintree bcast. However, the asynchronous pipeline ring bcast is the fastest. But the advantage is not that significant. Therefore, it is not worth to implement our own broadcast. I guess this is because in the real world, we have no idea what network topology nodes the system scheduler will be assigned to us. There is no advantage to do so.