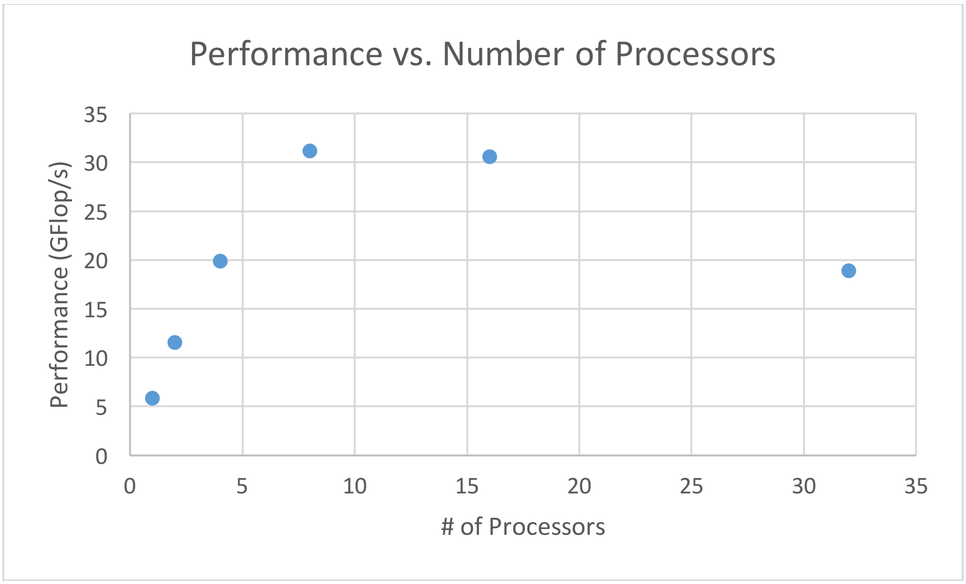
Justin Liu | 504-487-373
COM SCI 133 | Dis 1B
2015-05-11

Lab 2

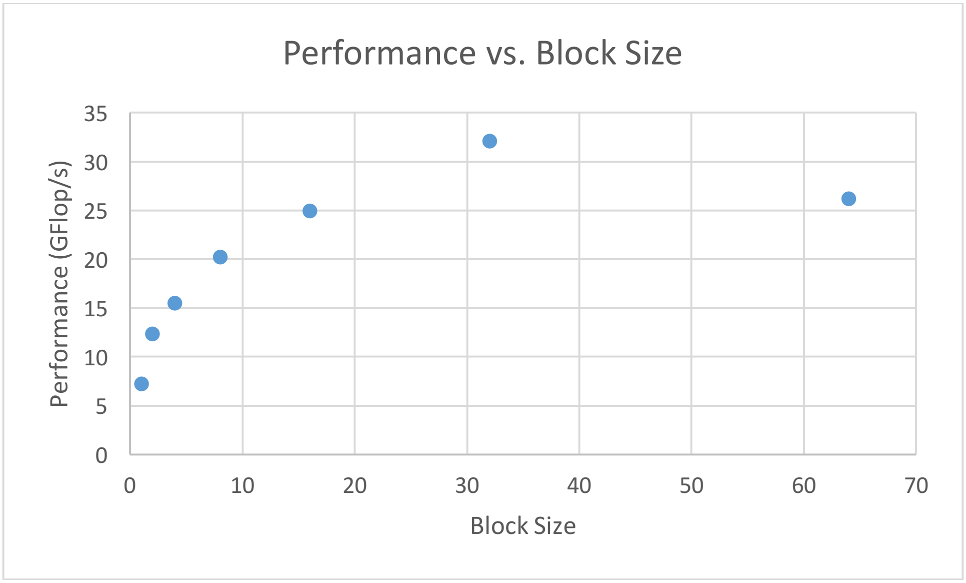
Results



# of processors	Performance (GFlop/s)
1	5.85
2	11.53
4	19.87
8	31.14
16	30.55
32	18.88

It appears that we had the best performance when using 8 processors. However, the server was under variable load at the time of testing, so it is possible that 16 processors is actually faster.

Block Size	Performance (GFlop/s)
1	7.17
2	12.32
4	15.48
8	20.2
16	24.9
32	32.1
64	26.15



From this, we can deduce the optimal block size is 32. Results obtained using size of 1024 and 16 processors.

Partitioning

Let n be the number of processors in the system. Then the root processor, 0, scatters 1/n of the matrix A to every processor, divided by rows, and broadcasts all of matrix B to all processors. Each processor multiplies the matrices together to form 1/n of the matrix C each. The root processor then gathers each segment of C and writes it to the C pointer.

Blocking, Buffered Blocking, and Non-Blocking

I did not use MPI_Send or MPI_Recv, instead using MPI_Scatter, MPI_Bcast, and MPI_Gather. The scatter, broadcast, and gather functions do not have buffered or non-blocking versions that can be used on our compiler. I chose to use each of those functions instead because, as discussed in class, the complexity is much better compared to sequential sends/receives, and essentially automates most of the optimizations I would have made to the send/receive functions.

If I were to use the send/receive functions, I would want the root processor to be able to send each segment of the matrices out in parallel. Therefore, I would use MPI_ISend and MPI_IRecv to avoid having the root processor wait between each send. MPI_Bsend and MPI_IRecv would also work similarly, but with more overhead.

Scalability

We do not have linear speedup because the performance begins decreasing after 16 processors. This is most likely because the root processor must communicate with all other processors in the system, and as the number of processors grows then the communication delays become more significant.

MPI vs. OpenMP

Although MPI is slower and more verbose than OpenMP, I prefer MPI over OpenMP because I can visualize what is happening better. Being able to see what information is being passed between processors makes more sense to me than simply adding a pragma and having the rest of it abstracted away.