CS133 Lab 2 Report

Yanzhe Liu 504153440

Experimental Results:

For size 4096 x 4096 matrices:

Performance (GFlop/s)	np=4	np=8	np=16	np=32
BS=16	12.12	18.26	26.87	32.92
BS=32	28.30	52.76	54.82	50.75
BS=64	28.74	62.77	70.10	58.75

Table 1. Performance with different experimental parameters

We found that block size of 64 is the optimal for cache performance. There is significant improvement with more processors, except for np=16 and np=32. Due to the CS133 server's CPU architecture, having 32 processors will increase the communication overhead.

Data Partition and Computation:

I broadcast entire matrix B to every processor, and use MPI_scatter to partition matrix A row-wise to every processor. In each processor, a partition of matrix C is computed by multiplying matrix A partition and matrix B. After computation, a MPI_gather is called to collect the resulting matrix C partitions from all processors.

Runtime Analysis:

I did not use the classical MPI send and receive in this implementation. But comparing different MPI send communication runtime, we can find that MPI_Isend and MPI_Irecv would have the fastest runtime because it is a non-blocking communication. The send/receive operation would return immediately.

Scalability:

The speedup by increasing the number of processors is not linear, because the CS133 server CPU uses a shared bus to communicate between processors, which does not handle all-to-all communication very well. Once it gets to 16 processors, the performance does not improve that much when increased to 32 processors.

Comparison to Lab1:

The MPI implementation is about the same programming effort as the OpenMP implementation. With the existing knowledge of loop optimization from lab1, however, implementing MPI version is somewhat simpler. In terms of throughput, the OpenMP version is faster (by 10GFlop/s), which is expected.