CPEN502 Assignment 2

Jiaqi Zhang (63174551)

1. Implementation

In this assignment, we use pthreads to implement Forward Gaussian Elimination. The main idea is to divide the matrix to P (thread number) parts so that different rows could be processed by corresponding threads at the same time. Barrier was applied. Each row would be processed at some stage and then eliminate other rows with normalization.

Cyclic stripped mapping was implemented and greatly resolved the load-balancing issue. Each thread can only process the rows whose mod values are the same as current thread id.

2. Performance

In general, larger matrix size can benefit from multiple threads, but for small matrix there is no big difference since pthread's power is not fully exploited.

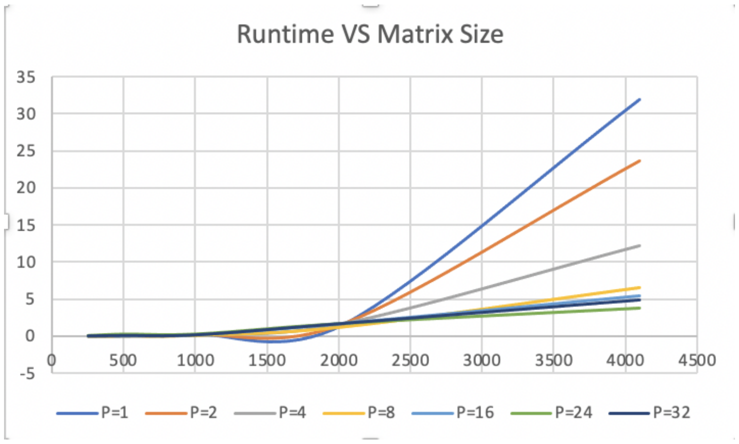

Figure 1. Runtime vs Matrix Size

2.1 Observations from Optimization Settings:

1) The bar chart below is for matrix size is 4096, thread number 32 and optimization setting -Ofast.
2) The performance of parallel implementation of Gaussian Elimination has improved a lot by using custom flags.
3) Custom Optimization Setting is about 8 times faster than the one without optimization setting.

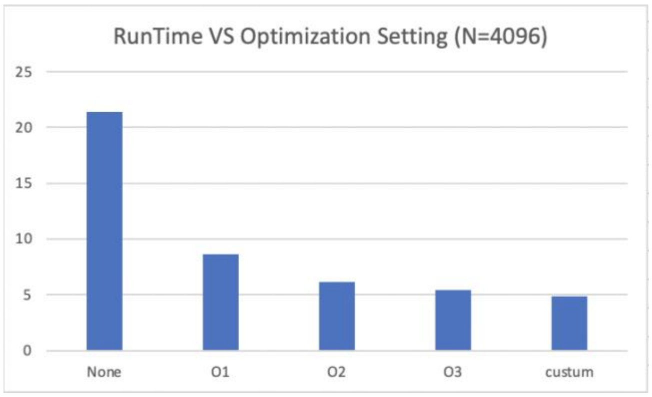

Figure 2. Runtime vs optimization Setting

2.2 Observations from number of processors

1) The bar chart below is for matrix size 4096, thread number 32 and optimization setting -Ofast. The baseline of speedup is vanilla Gaussian elimination.
2) As we can see from the line chart, an obvious "bump" could be found in the bar chart. This is because for large matrixes, if thread number is small, there is not much parallelism, but if thread number is too large, barrier waiting time will increase a lot and turn down performance.
3) The best speedup for matrix size 4096 is 8.27x by using 24 threads.

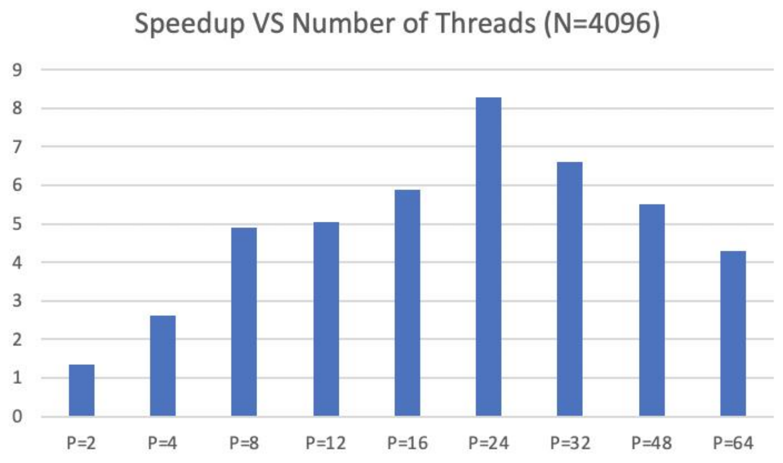

Figure 3. Speedup vs Number of threads

3. pthread VS MPI

Both pthread and MPI are power tools for parallel programming. The basic intuition is similar by dividing large amount of work into many small parts and small parts can be executed at the same time. However, the strategies to achieve their goal are different. In MPI, communications between groups were set up by sending and receiving messages. In pthreads, different flows of work were controlled by settings like locks and barriers.

My pthread version of Gaussian Elimination is slower than MPI. This is because my MPI version does not require waiting time - each submatrix can check its rank with local rank, and do sending and receiving accordingly. However, the pthread implementation must wait until all the threads have come to the same stage before elimination. I believe it can be improved by pipeline mapping or improving loop structure by grouping several rows into a block in order to reduce the times of using barrier.

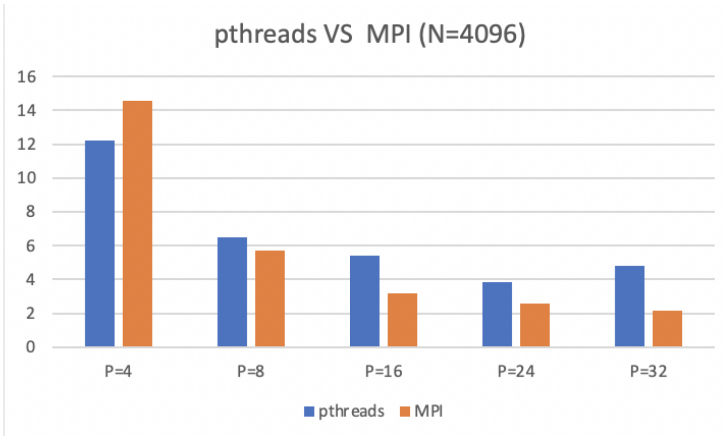

Figure 4. Pthread vs MPI

4. Compile and Run

Parallel Compile: *gcc -Ofast -o ap1 assign2_pthreads.c -pthread*
Parallel Run: *./ap1 4096 32*

Best result for parallel version: 3.691539 seconds. (24 threads)
Best result for sequential version: 14.711683 seconds (same as MPI implementation).