CPEN 512 Assignment 1

Parallel Gaussian Elimination using OpenMPI

Muchen He, 44638154, Oct 5, 2020

1. Introduction

In this assignment, we implement the forward-elimination portion of the Gaussian Elimination [1]. The objective of the assignment is to explore message-passing parallelization using OpenMPI, and observe its performance benefits over the serial program. Single-precision floating-point values are used, and all metrics are measured form the cpen512.ece.ubc.ca computer.

2. Background

The forward elimination (FE) involves reducing a given matrix A into its row-echelon form (REF), where rows have a non-zero leading coefficient with every element below the coefficient being 0 such that it forms an inverted triangle of non-zero values. For a M by N matrix of non-zero values, the end result is

$$A' = \begin{bmatrix} 1 & a_{1,2} & a_{1,3} & \cdots & a_{1,N} \\ 0 & 1 & a_{2,3} & \dots & a_{2,N} \\ 0 & 0 & 1 & \dots & a_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Traditionally, the FE algorithm designates a pivot point with corresponding row and column (defined and initialized as $h = 1$ and $k = 1$ respectively). First the current row indexed h is divided by the leading coefficient indexed k. Then every row below pivot subtracts a multiple of the pivot row such that every element in column k below row h is 0. This process repeats until either h or k reaches the matrix boundary M or N. The algorithm for FE can be simplified if we make the assumption the matrix is investable and reducible such that we don't need to perform row-swapping. This way, we don't have to swap rows but still ensure output matrix is in REF.

3. Parallel Implementation

The parallelized algorithm is based on a naïve implementation from CoffeeBeforeArch[2]. The intuition is to divide each row operation such that many different rows can be reduced at the same time. The memory allocated to the matrix can be assumed to be divided equally amongst C different processes. Therefore each process $P_{0,1,\dots,C-1}$ gets M/C contiguous rows. This is done using the built-in `MPI_Scatter` function. As reduction finishes on row R_1, parallel computation can begin immediately on rows R_2 to R_M. However, because MPI relies on message passing, we need to send the content of the entire row to other processes that's operating on different rows — this is done using `MPI_Bcast`. While waiting for the rows above to be processed, processes that operate on later rows also call on `MPI_Bcast` to receive reduced row information in a loop. The received content gets put into a local buffer and then the local rows the processor p_n is responsible for, will be manipulated using the content in the buffer. Finally, all processes synchronize with `MPI_Barrier`, and then all the rows are brought back together into a single block of memory that represents the final matrix A' using `MPI_Gather` function.

There is room for improvement as we can use cyclic mapping and pipelined pipelined communication as shown in [2] and [3] to significantly reduce overheads due to parallel communications. However due to lack of time, I did not implement them.

Table 1. Total Execution Time

Matrx Size	Serial	2	4	8	16	32
128	4.85e-04	9.21e-04	1.42e-03	2.20e-03	3.01e-03	3.53e-02
512	3.26e-02	2.44e-02	1.70e-02	1.33e-02	1.23e-02	3.29e-02
1024	2.52e-01	1.84e-01	1.10e-01	6.88e-02	4.77e-02	6.04e-02
2048	3.01e+0	2.51e+00	1.70e+00	1.15e+00	6.92e-01	7.56e-01
4096	2.67e+01	3.22e+01	1.94e+01	1.02e+01	5.78e+00	5.57e+00

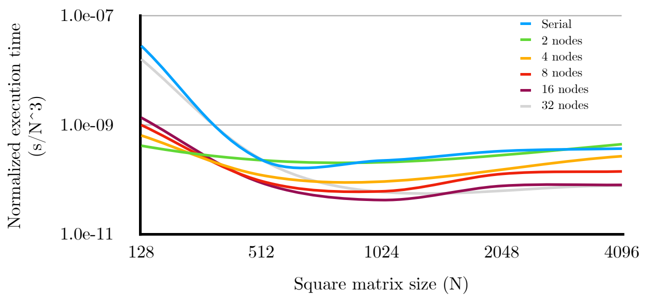

Figure 2. Normalized execution time

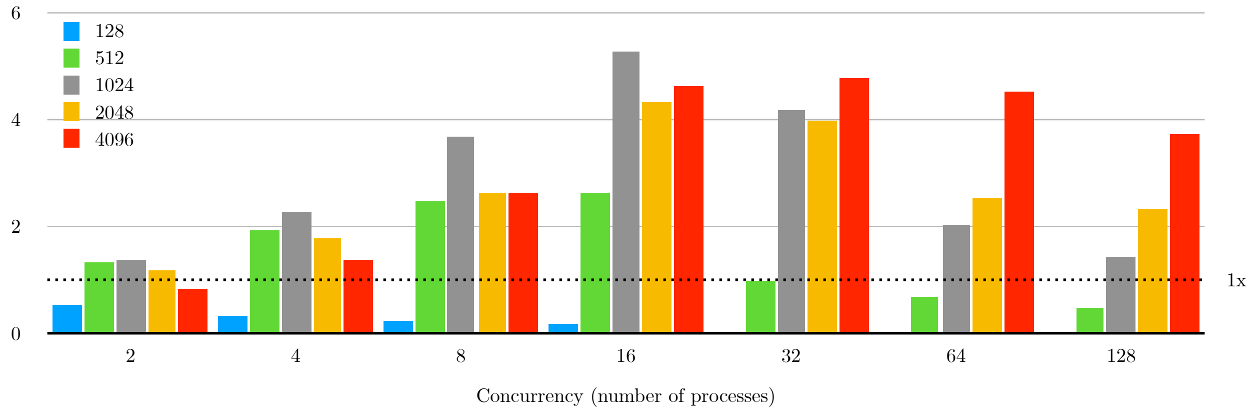

Figure 3. Speedup vs. Concurrency

4. Performance

In this section, I evaluate the performance benefits of parallelized code compared to the serial case. The program is compiled using `mpicc` using a set of custom compiler flags[1]. Table 1 shows the total execution time (in seconds) of FE algorithm using square matrix with various matrix sizes and number of processes (specified using `-np C` as arguments). Figure 1 shows the average normalized execution time (execution time $\div N^3$ for a square matrix) and we observe that serial has the worst normalized execution time.

When matrix size is large, as we increase number of processes operating concurrently, we generally observe a shift down in normalized execution time. For $C = 2$, we see a significant performance boost for small matrix sizes, but no speedup for large C. For increasing large C, the performance on small matrices decrease due to excessive communication overheads, but this is traded with significant throughput for large N (seen clearly for $C = 16$ or 32 in Figure 1).

Figure 2 shows the speedup of the FE algorithm compared to serial base case. We also observe that we get the most speedup when using more processes for larger matrices. Conversely, for smaller matrices, more processes becomes slower. Note that as we increase number of processes used beyond 32 and 64, we observe a decrease in speedup due to excess communication overheads.

[1] Custom compiler flags: `-O3 -m64 -march=barcelona -mtune=barcelona -ftree-vectorize -fmerge-all-constants -fmodulo-sched -faggressive-loop-optimizations -floop-interchange -funroll-all-loops -ftree-loop-distribution -funswitch-loops -finline-functions`

5. Conclusion

In this assignment, we explored using OpenMPI to turn a serial FE algorithm to perform Gaussian elimination to a parallel program. We observe that for large matrices, parallel programs with large number of concurrent processes excels, while suffers for small matrices due to communication overheads. The best speedup is observed for a square matrix with $N = 1024$ and using $C = 16$ concurrent processes with a 5.3x speedup.

References

(1) "CoffeeBeforeArch - Overview," GitHub. [Online]. Available: https://github.com/coffeebeforearch. [Accessed: 05-Oct-2020].

(2) "Gaussian elimination," Wikipedia, 12-Sep-2020. [Online]. Available: https://en.wikipedia.org/wiki/Gaussian_elimination. [Accessed: 05-Oct-2020].

(3) J. Howe and S. Bratcher, Parallel Gaussian Elimination, 1996. [Online]. Available: http://cseweb.ucsd.edu/classes/fa98/cse164b/Projects/PastProjects/LU/abstract.html. [Accessed: 05-Oct-2020].