Quang Anh Hoang (qh62@drexel.edu)

# Gaussian Elimination using OpenMP

ECEC 622: Parallel Computer Architecture
Professor Naga Kandasamy

1. Introduction
   Gaussian elimination is a method which is used for solving a matrix equation: Ax = B, where A is an n x n matrix, and x and B are n x 1 matrices. This method's goal is to achieve a specific form of A, which is called "row echelon form". In this form, the coefficients lying on the diagonal line of the matrix is reduced to 1. Those coefficients on the lower half of the diagonal is reduced to 0, and those on the other half will have different values so that the matrix does not change its value.
   Gaussian elimination involves two steps to reduce each row consecutively: division step and elimination step. In division step, every coefficient is divided by the leading coefficient (the one lying on the diagonal line of the matrix). This will make the leading coefficient's value to be 1, while the remainings have a specific value so that the ratio between each coefficient and the leading one of that row does not change. In elimination step, for each row below the divided one, each coefficient is reduced by a multiple of that from the same column but on the divided row, and the multiplier will be the one from that row sharing the column with the leading coefficient in that iteration. This will make that coefficient's value to be 0, and eventually all coefficients below the diagonal line will be 0.
   The main task of this assignment is parallelizing the division and elimination steps with a barrier synchonization in between to speedup the program. For each iteration, from the top row to bottom one, there will be a number of threads generated to execute dividing function in parallel (using chunking method on coefficients on the right of leading one). Then barrier sync is used to wait for all threads to finish their task before moving on to the eliminating function, where each thread takes a number of rows (approximately the same in chunking method) to reduce the coefficients.
2. Implementation Design
   Although the algorithm is very similar to Gaussian Elimination using pthread library, in this assignment, using OpenMP makes the program much simpler to implement, as there is no need to use a main function to create threads and then assign task to them. In OpenMP, both divide and eliminate functions are reduced to two "for" loops, and they are parallelized by a parallel directive following this format:
   ```
   #pragma omp parallel num_threads(num_threads)
   {
           #pragma omp for
           // Divide loop //
           #pragma omp for
           // Eliminate loop //
   }
   ```
   Since by the end of each FOR directive, all threads are synchronized by a barrier sync before moving on to the next task, there is no need to add a barrier sync between two loops.
3. Result
   There are sixteen different configurations used in this assignment to examine the performance of parallelizing Gaussian Elimination, with four different values of number of threads: 4, 8, 16 and 32; four different values of matrix size: 512 x 512, 1024 x 1024, 2048 x 2048, and 4096 x 4096. Each configuration is run for 20 times and speedups are recorded and shown in the tables below for each method:

| | 4 threads | 8 threads | 16 threads | 32 threads |
|---|---|---|---|---|
| 512 x 512 | 1.226783 | 0.217145 | 0.089016 | 0.042474 |
| 1024 x 1024 | 2.359133 | 0.892171 | 0.400049 | 0.176938 |
| 2048 x 2048 | 0.958888 | 0.947925 | 0.821744 | 0.671339 |
| 4096 x 4096 | 0.880680 | 0.935251 | 0.891756 | 0.890608 |

**Table 1:** Average speedup of multi-thread Gaussian Elimination function, chunking method for both divide and eliminate functions

According to the collected result, the average speedup would surpass 1 if the matrices' dimension is 512 x 512 or 1024 x 1024, and the program uses 4 threads to execute. The speedup is enormous for 1024 x 1024, as on average, the multi-threaded program runs more than twice as fast single-threaded one, at 2.36 times. Among different configurations with number of threads, 4-thread program probably has the highest average speedup, and only when the volume of data is too large with 4096 x 4096 matrix A, 8-thread program has higher speedup than 4-thread one.
However, there are only two out of sixteen configurations have led to a gain in performance (with speedup larger than 1), this implementation still needs improvements to reduce runtime and elevate the performance.