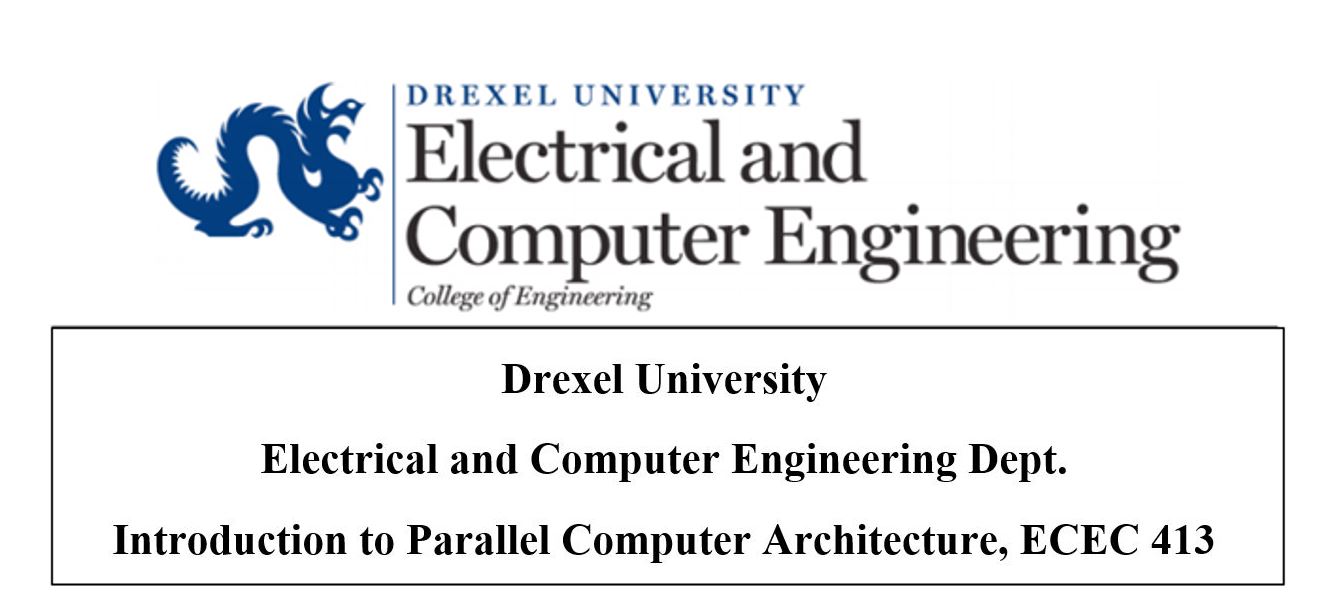
****

**Title:** Gaussian Elimination using OpenMP

**Names:** Avik Bag, Shannon Miles

**TA:**  Vasil Pano

**Section:** 001

**Date Performed:** 1/20/17

**Date Due:** 1/29/17

**Date Received:** 1/20/17

**Introduction:**The object of this assignment is to take a serial implementation of the Gaussian elimination algorithm and develop a parallel formulation of the gauss\_eliminate C-file using OpenMP. This will be accomplished by modifying the gauss\_eliminate\_using\_openmp() function. For matrix sizes of 1024 × 1024, 2048 × 2048, 4096 × 4096, and 8192 × 8192, the parallelized code will be compared to the serial implementation and the speed up will be reported when using 2, 4, 8, and 16 threads. (Please see README on how to run program.)

**Graphs:**

Graph 1………………………………..

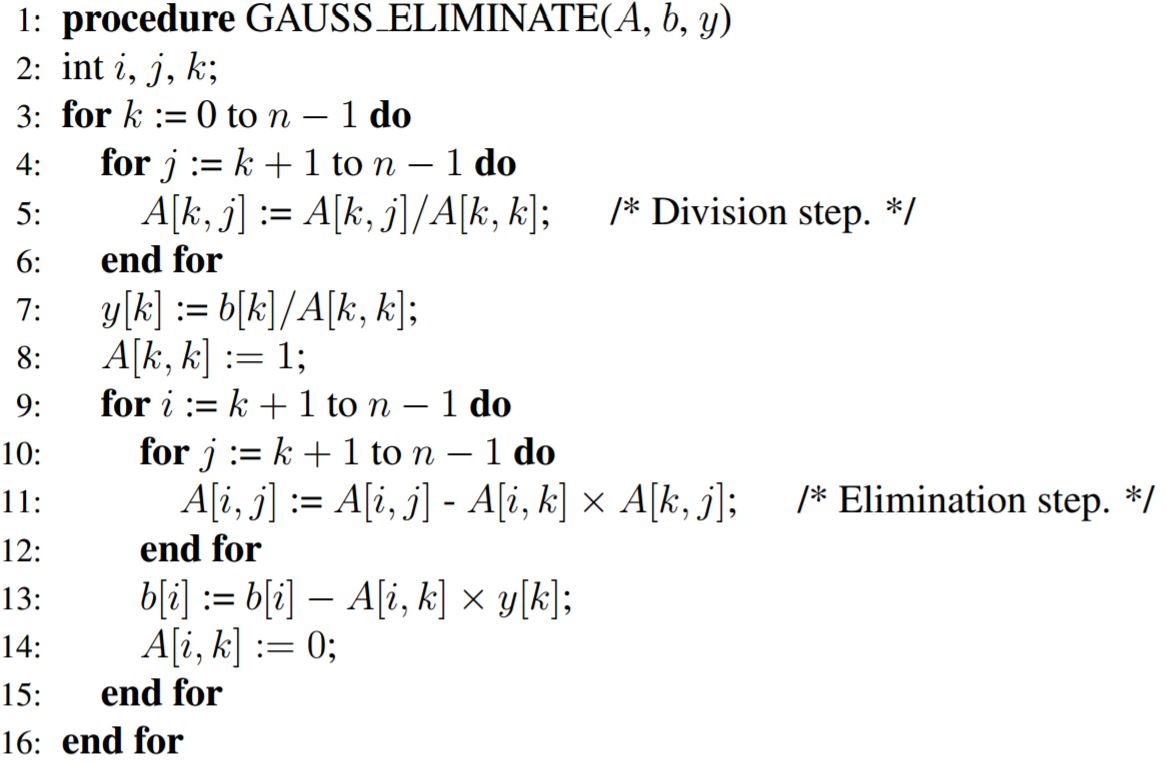
*Figure 1. This graph relates the execution time to the matrix size when using a serial implementation to that of 2, 4, 8 and 16 threads.*

Graph 2………………………………..

*Figure 2. This graph relates the speed up to the number of threads.*

**Discussion**:

The Gaussian elimination code to be parallelized is shown below:



This was accomplished by choosing the correct variables that should be shared or made private, and by parallelizing the correct for loops during the parallelization stage. …………. (Can list the code that was modified and what the code accomplished.

This modification allowed us to get speed ups of ………………….for threads 2, 4, 8, and 16, respectively, which can be seen in figure 2. As we can see from that figure, no more speed ups can be gained after using …………….threads. Also, from figure 1, it can be seen that the more threads that are added, the faster the execution time. After ………threads are used the execution time of the multithreaded program is improved over the serial implementation.

To make sure our implementation will run on Drexel’s servers, the parallelized code designed for the gauss\_eliminate\_using\_openmp() function was built and tested on xunil-05.coe.drexel.edu.

**Conclusion:**

Parallelizing this code will not always lead to a speed up or faster runtime. However, the more threads used, the faster the runtime was to a point. There was a limiting number of threads that could be used to see a speed up.