Jason Gallagher

Steven Kravitsky

**OpenMP Assignment 1**

This assignment was parallelizing Gaussian Elimination using the OpenMP library. OpenMP is used to parallelize two for loops using a user specified number of threads. It also allowed us to use variables in either a shared or variable format and dynamically allocate the threads to the code. The code was fairly straightforward to parallelize, with the serial code only needing to be slightly changed by using pragmas.



Figure 1: The first pragma used to initilize the threads and elements



Figure 2: The second pragma creating the threads and executing the loops

In Figure 1 it can be seen that the first pragma is being run on a user specified number of threads, thread\_count, and gets the variables num\_elements and U which are going to be used by the threads themselves. The variables i, j, and k are going to be used inside the threads but need to be private for each thread so that they are not being overwritten.

In Figure 2 it can be seen that the second pragma is dynamically creating the threads and executing the loops. The threads will be assigned as they become available, which was easier to code but does come with the cost of additional overhead. The threads are executing on a row at a time, going through a for loop to compute the elements individually. There doesn’t need to be any locking or worry of race conditions because there is a built-in barrier at the end of the loop that stops forces the program to wait for the threads to finish before continuing.

The speedup seen in Figure 3 shows about the results that we were expecting to see when this test was parallelized. At 1024x1024 elements the speedup just about doubles every time the number of cores doubles. This is more than likely due to the small size of the array since this is where the serial code would run the fastest. At 2048x2048 we had some weird odd results, with the 4 core result being the lowest speed up and the 16 core result being the highest speed up. This could have been due to an issue with the testing script or been due to a larger or smaller load being run on the server when the code for this was run. At 4096x4096 and 8192 the speed ups are again larger, due to the larger matrix being more suited to the parallelized code.

Figure : OpenMP data