Noah Robinson
Jack Pinkstone
Professor Kandasamy
ECEC 413
15 May 2023

Gaussian Elimination OMP Report

This assignment required the students to use the OMP library to write multi-threaded code to parallelize the gaussian elimination process. We parallelized both division and elimination steps with the striding method.

Design

```
void gauss_eliminate_using_omp(Matrix U, int thread_count)
{
    int num_elements = U.num_rows;
    //int stride = thread_data->num_threads;

    #pragma omp parallel num_threads(thread_count) shared(U, num_elements, thread_count)
    {
        int tid = omp_get_thread_num();

        for (int k = 0; k < num_elements; k++)
        {
            for (int j = (k + tid + 1); j < num_elements; j = j + thread_count)
            {
                U.elements[num_elements * k + j] = (float)(U.elements[num_elements * k + j] / U.elements[num_elements * k + k]);
            }
            #pragma omp barrier

            for (int i = (tid + k + 1); i < num_elements; i = i + thread_count)
            {
                for (int j = (k + 1); j < num_elements; j++)
                {
                    U.elements[num_elements * i + j] -= (U.elements[num_elements * i + k] * U.elements[num_elements * k + j]);
                }
                U.elements[num_elements * i + k] = 0;
            }
            #pragma omp barrier
        }
        for (int k = 0 + tid; k < num_elements; k = k + thread_count)
        {
            U.elements[num_elements * k + k] = 1;
        }
    }

}
```

Above is the OpenMP function that runs in each thread. Num_threads is set to thread_count and we share the variables: U, num_elements, and thread_count between the OpenMP threads. We found that sharing num_elements and thread_count increased performance. Inside the hash pragma we first get the thread ID. In the loop, the division step is performed in parallel, striding down the row. After the divisions are complete, the elimination process is also done in parallel by striding down the rows. Then the principal diagonal entries are set to one. At the end of the hash pragma section, the allocated memory is automatically freed, and the thread is exited.

The stride method is accomplished by having each thread start at a position relative to their “tid”. With each loop, the index that is being operated on increases by the stride value (this value stays constant and is equal to the number of threads). For example, thread zero out of eight starts at index zero and every iteration will increase the index of operation by eight until it reaches the end of scope. With each thread operating at offset, all indexes will be reached.

Performance Comparison

Reference					
	Matrix Size				
	Seconds	512	1024	2048	4096
Threads	4	0.05	0.28	3.63	28.12
	8	0.03	0.27	4.12	28.45
	16	0.03	0.27	4.04	28.91
	32	0.03	0.27	4.21	29.78
Parallelized					
	Matrix Size				
	Seconds	512	1024	2048	4096
Threads	4	0.04	0.24	1.54	13.48
	8	0.03	0.16	1.16	9.37
	16	0.03	0.15	0.82	9.59
	32	0.13	0.31	1.18	10.52

The tables show that the performance of the reference single-threaded solution and the parallelized version with striding vary depending on the number of elements in the matrix and thread count.

With few elements in the array and high thread counts such as 16 and 32, the reference solution is faster than the OMP multi-threaded solutions. As for thread counts four and eight, open MP seems to be optimized so that it has the same runtime in low element cases. Just like the pthreads, as the number of elements to compute increases, the parallel version becomes much faster than the reference. This is because the parallel version can distribute the work across multiple threads, which can significantly reduce the execution time.