Jack Pinkstone

Noah Robinson

Professor Kandasamy

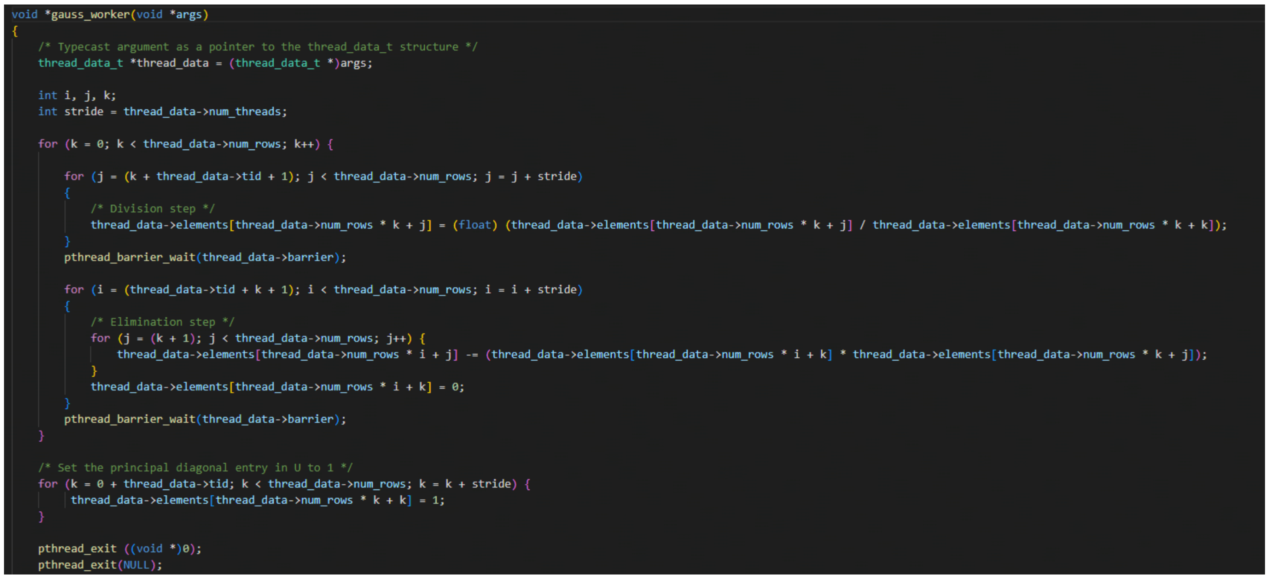
ECEC 413

7 May 2023

Gaussian Elimination Report

This assignment required the students to use the pthread library to write multi-threaded code to parallelize the gaussian elimination process. There were two implementations to solve this problem. One way uses “chunking” and the other uses “striding”. We parallelized both division and elimination steps with the striding method.

Design:

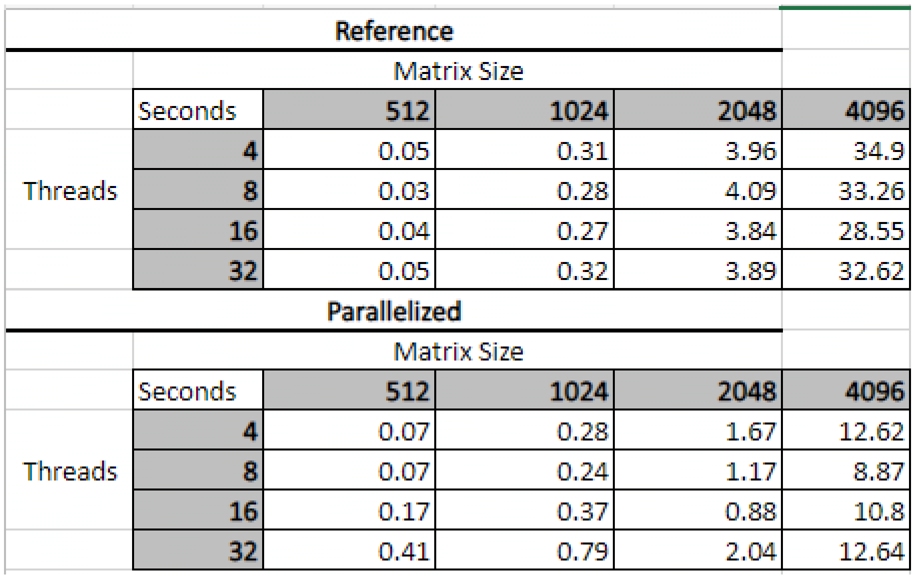


```c
void *gauss_worker(void *args)
{
    /* Typecast argument as a pointer to the thread_data_t structure */
    thread_data_t *thread_data = (thread_data_t *)args;

    int i, j, k;
    int stride = thread_data->num_threads;

    for (k = 0; k < thread_data->num_rows; k++) {

        for (j = (k + thread_data->tid + 1); j < thread_data->num_rows; j = j + stride)
        {
            /* Division step */
            thread_data->elements[thread_data->num_rows * k + j] = (float) (thread_data->elements[thread_data->num_rows * k + j] / thread_data->elements[thread_data->num_rows * k + k]);
        }
        pthread_barrier_wait(thread_data->barrier);

        for (i = (thread_data->tid + k + 1); i < thread_data->num_rows; i = i + stride)
        {
            /* Elimination step */
            for (j = (k + 1); j < thread_data->num_rows; j++) {
                thread_data->elements[thread_data->num_rows * i + j] -= (thread_data->elements[thread_data->num_rows * i + k] * thread_data->elements[thread_data->num_rows * k + j]);
            }
            thread_data->elements[thread_data->num_rows * i + k] = 0;
        }
        pthread_barrier_wait(thread_data->barrier);
    }

    /* Set the principal diagonal entry in U to 1 */
    for (k = 0 + thread_data->tid; k < thread_data->num_rows; k = k + stride) {
        thread_data->elements[thread_data->num_rows * k + k] = 1;
    }

    pthread_exit ((void *)0);
    pthread_exit(NULL);
```

Above is the worker function that runs in each thread. First, it gets its thread ID. Next, it type casts its arguments for use in the function. Then, the offset from where to start the computation is found. 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 principle diagonal entries are set to one. At the end, the allocated memory is 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 every iteration will increase the index of operation by a stride of 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	0.31	3.96	34.9
	8	0.03	0.28	4.09	33.26
	16	0.04	0.27	3.84	28.55
	32	0.05	0.32	3.89	32.62

Parallelized

		Matrix Size			
	Seconds	512	1024	2048	4096
Threads	4	0.07	0.28	1.67	12.62
	8	0.07	0.24	1.17	8.87
	16	0.17	0.37	0.88	10.8
	32	0.41	0.79	2.04	12.64

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, the reference solution is much faster than the multi-threaded solutions. This is because the overhead of thread synchronization and communication outweighs the benefits of parallelization. However, 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.

Striding with less elements takes longer because of cache misses causing delays in the cores. When one thread adds its complete value to the array, it invalids the cache line on the other core’s cache. The other threads will have cache missed and therefore must fetch the line again from memory.