DREXEL UNIVERSITY
Electrical and
Computer Engineering
College of Engineering

Drexel University

Electrical and Computer Engineering Dept.

ECEC-413

Assignment 2:
Pthread Gaussian Elimination

Minjae Park

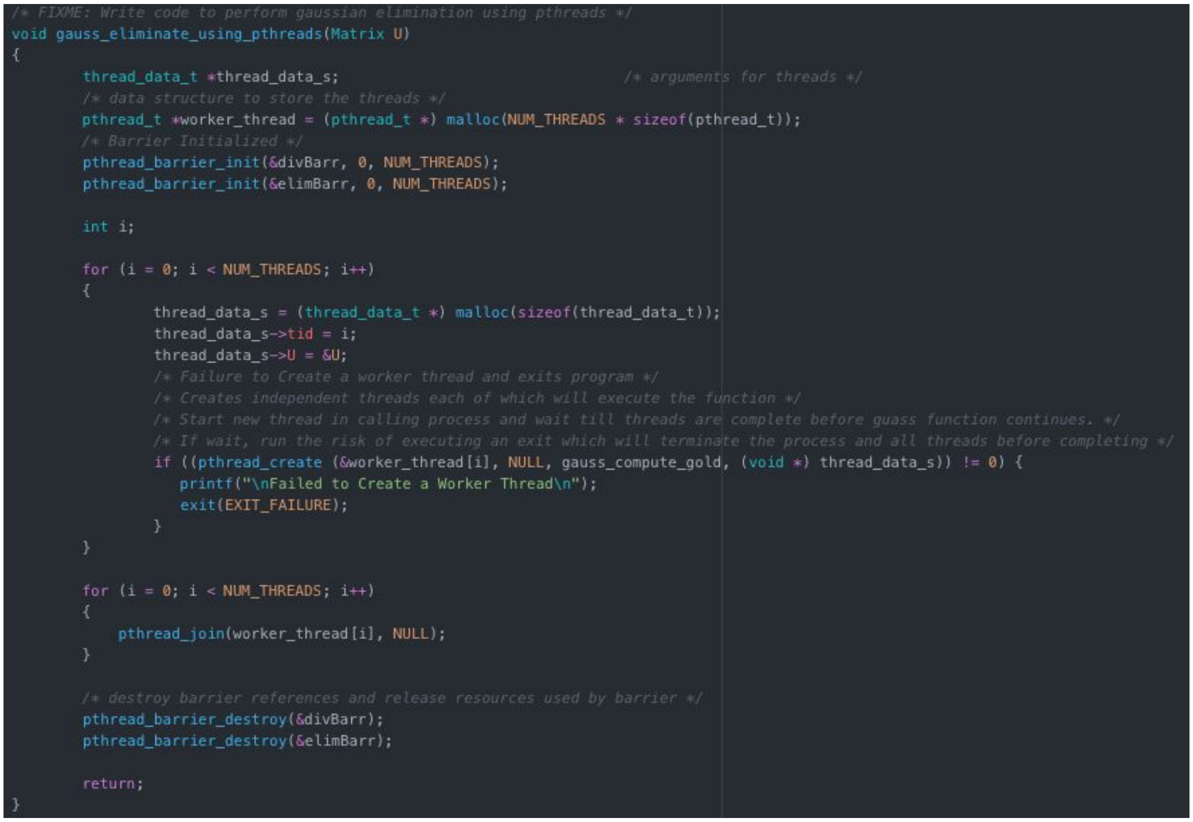
John Truong

Professor Naga Kandasamy

Date: 05/03/2020

Gaussian Elimination using Barriers:

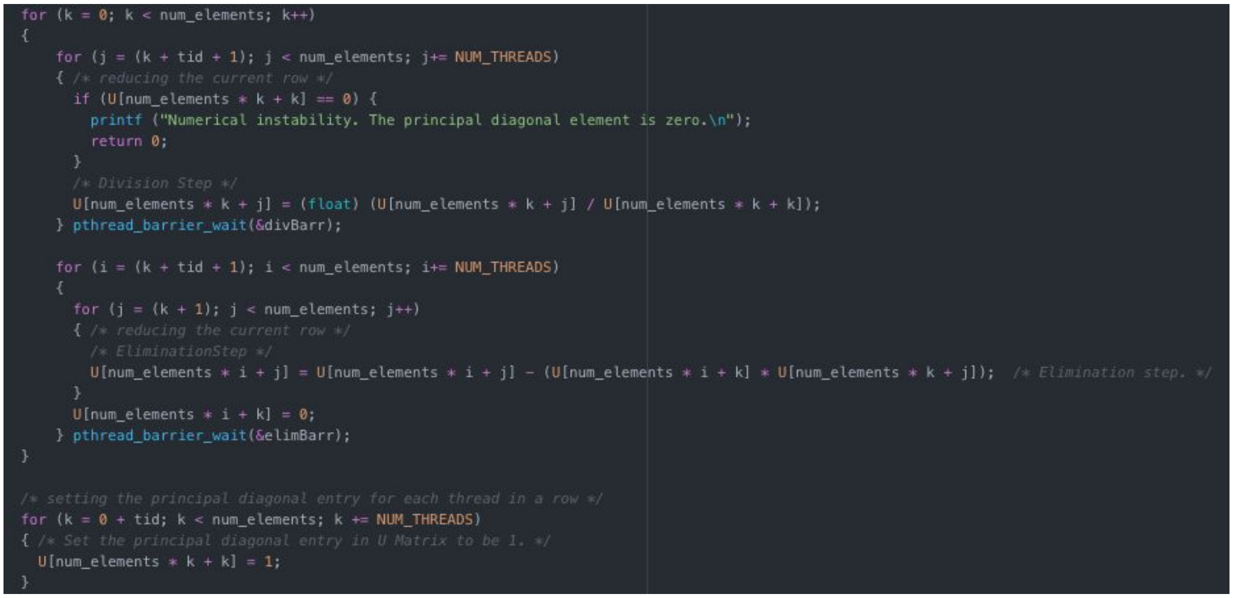
For this assignment, the introduction of barriers were used to extend the serial or single thread component into a multi-thread for the Gaussian Elimination. Barriers are used to make sure the threads are running in parallel or synchronously. Each thread has to wait for the other thread to finish in order to proceed to the next step. First you have to initialize the barriers to be used in the algorithm; in this assignment two are intialized: division step and elimination step. After initializing it, each thread is given a thread ID and will be pointed to the matrix structure. Once all threads are finished going through the Gaussian Elimination, the barriers would be destroyed. The pthread_barrier_destroy function destroys the barriers pointed to by the barrier arguments. Once the barriers are destroyed it releases the resources it used and should be reinitialized with pthread_barrier_init before it is called again.



```
/* FIXME: Write code to perform gaussian elimination using pthreads */
void gauss_eliminate_using_pthreads(Matrix U)
{
        thread_data_t *thread_data_s;                          /* arguments for threads */
        /* data structure to store the threads */
        pthread_t *worker_thread = (pthread_t *) malloc(NUM_THREADS * sizeof(pthread_t));
        /* Barrier Initialized */
        pthread_barrier_init(&divBarr, 0, NUM_THREADS);
        pthread_barrier_init(&elimBarr, 0, NUM_THREADS);

        int i;

        for (i = 0; i < NUM_THREADS; i++)
        {
                thread_data_s = (thread_data_t *) malloc(sizeof(thread_data_t));
                thread_data_s->tid = i;
                thread_data_s->U = &U;
                /* Failure to Create a worker thread and exits program */
                /* Creates independent threads each of which will execute the function */
                /* Start new thread in calling process and wait till threads are complete before guass function continues. */
                /* If wait, run the risk of executing an exit which will terminate the process and all threads before completing */
                if ((pthread_create (&worker_thread[i], NULL, gauss_compute_gold, (void *) thread_data_s)) != 0) {
                   printf("\nFailed to Create a Worker Thread\n");
                   exit(EXIT_FAILURE);
                }
        }

        for (i = 0; i < NUM_THREADS; i++)
        {
            pthread_join(worker_thread[i], NULL);
        }

        /* destroy barrier references and release resources used by barrier */
        pthread_barrier_destroy(&divBarr);
        pthread_barrier_destroy(&elimBarr);

        return;
}
```

Figure 2: Initializing Pthread Barrier and Destroying Barrier

For the pthread computation for this assignment, it is very similar to the serial or single thread logic. Each thread has to go through the division and elimination algorithm at the same time. In the division component, all of the threads are working on the same row while each thread is managing the columns that they are allocated to. After each thread is finished, it will wait at the first barrier till all threads are finished before it proceeds to the next step. In the elimination step, each thread works on the rows which they are assigned to. In this step, the elimination process will set the pivot points in the matrix to be zero. Once threads are finished, it will wait at the barrier till all threads are finished before it will proceed through the loop again. Once all the steps are exhausted and the entire matrix is finished, the principal diagonal entry is set to 1 for each row.



```
for (k = 0; k < num_elements; k++)
{
    for (j = (k + tid + 1); j < num_elements; j+= NUM_THREADS)
    { /* reducing the current row */
      if (U[num_elements * k + k] == 0) {
        printf ("Numerical instability. The principal diagonal element is zero.\n");
        return 0;
      }
      /* Division Step */
      U[num_elements * k + j] = (float) (U[num_elements * k + j] / U[num_elements * k + k]);
    } pthread_barrier_wait(&divBarr);

    for (i = (k + tid + 1); i < num_elements; i+= NUM_THREADS)
    {
      for (j = (k + 1); j < num_elements; j++)
      { /* reducing the current row */
        /* EliminationStep */
        U[num_elements * i + j] = U[num_elements * i + j] - (U[num_elements * i + k] * U[num_elements * k + j]);  /* Elimination step. */
      }
      U[num_elements * i + k] = 0;
    } pthread_barrier_wait(&elimBarr);
}

/* setting the principal diagonal entry for each thread in a row */
for (k = 0 + tid; k < num_elements; k += NUM_THREADS)
{ /* Set the principal diagonal entry in U Matrix to be 1. */
  U[num_elements * k + k] = 1;
}
```

Figure 2: Logic Pthread Computation with Barrier Wait

Results:Speed Up:

Matrix Size	Single Thread	4 Threads	8 Threads	16 Threads	32 Threads
512x512	0.06	0.135	0.109	0.12	0.142
1024x1024	0.381	0.619	0.556	0.547	0.572
2048x2048	2.51	3.845	3.36	3.33	3.32

Table 1: Gaussian Elimination Data on Xunil-05

Matrix Size	Single Thread	4 Threads	8 Threads	16 Threads	32 Threads
512x512		44.44%	55.05%	50.00%	42.25%
1024x1024		61.55%	68.53%	69.95%	66.61%
2048x2048		65.28%	74.40%	75.38%	77.71%

Table 1: Speedup of Gaussian Elimination