Tam Phi
Hanh Do Phung

Assignment 7: Gaussian Elimination parallelization with CUDA

I. Introduction

Gaussian elimination is a method of solving a system of n-linear equations of the form Ax = b in which x is a (n x 1) vector of all x_1 to x_n solutions and A is an n x n sized matrix representing the coefficients of variables x_1 to x_n.

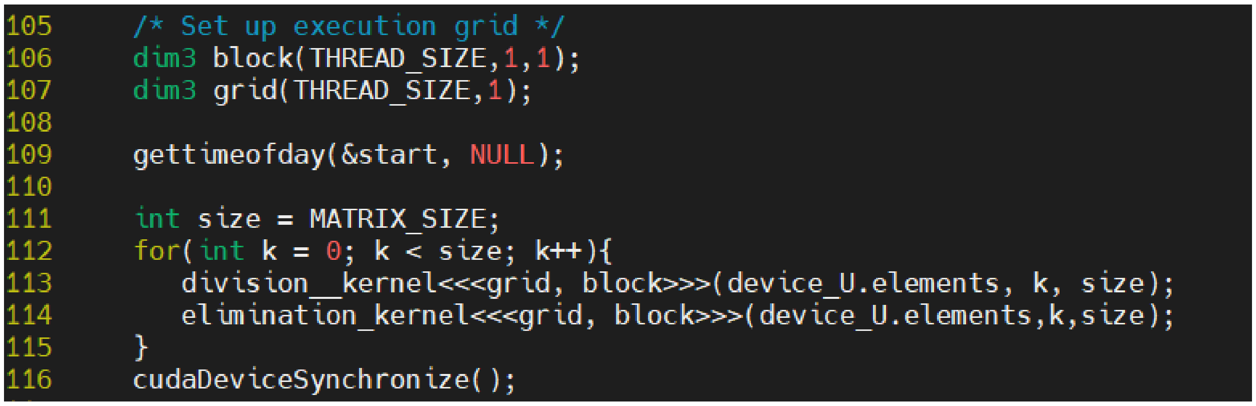
II. Design and Implementation

There are 2 steps in implementation.

1) Exec() function:

-Allocate memory on GPU for the output matrix U, call the kernel function to perform parallelization

```
Pseudo code{
  /*Allocate memory on device */
  device_U = Allocate on GPU for output matrix
  /* Copy image to device */
  Copy elements from A to device_U
  /* Create 1d thread blocks of size THREAD_SIZE/
  /* Each 1d grid of size THREAD_SIZE */
```



```
    /* Set up execution grid */
    dim3 block(THREAD_SIZE,1,1);
    dim3 grid(THREAD_SIZE,1);

    gettimeofday(&start, NULL);

    int size = MATRIX_SIZE;
    for( int k = 0; k < size; k++){
       division__kernel<<<grid, block>>>(device_U.elements, k, size);
       elimination_kernel<<<grid, block>>>(device_U.elements,k,size);
    }
    cudaDeviceSynchronize();
```

2) GPU kernel functions:

- **division_kernel():** A function that performs division of current row by pivot element

```
Pseudo code {
/* Due to 1d thread & grid block */
  tid = blockDim.x * blockIdx.x + threadIdx.x;
```

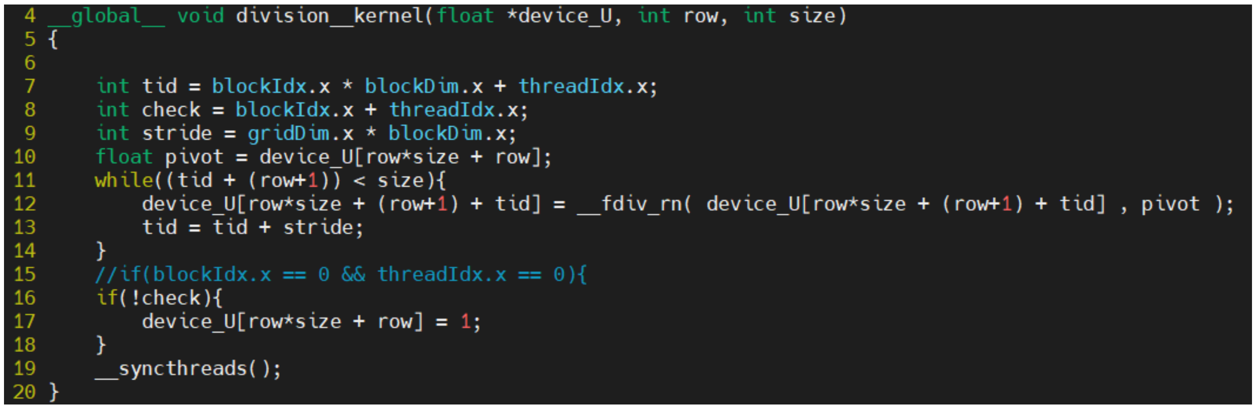
/* while(tid + (row+1)) = for(j = k+1; k < matrix_size; j++) */

stride = THREAD_SIZE * THREAD_SIZE;

tid = tid + stride;

/* set pivot to 1 */

}



```
__global__ void division_kernel(float *device_U, int row, int size)
{

    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int check = blockIdx.x + threadIdx.x;
    int stride = gridDim.x * blockDim.x;
    float pivot = device_U[row*size + row];
    while((tid + (row+1)) < size){
        device_U[row*size + (row+1) + tid] = __fdiv_rn( device_U[row*size + (row+1) + tid] , pivot );
        tid = tid + stride;
    }
    //if(blockIdx.x == 0 && threadIdx.x == 0){
    if(!check){
        device_U[row*size + row] = 1;
    }
    __syncthreads();
}
```

- **elimination_kernel():** A function that performs reduction of under rows

Pseudo code {

/* while(row + 1 + blockIdx.x) = for (int i = k+1; i < matrix_size; i++)

/* Due to 1d thread & grid block */

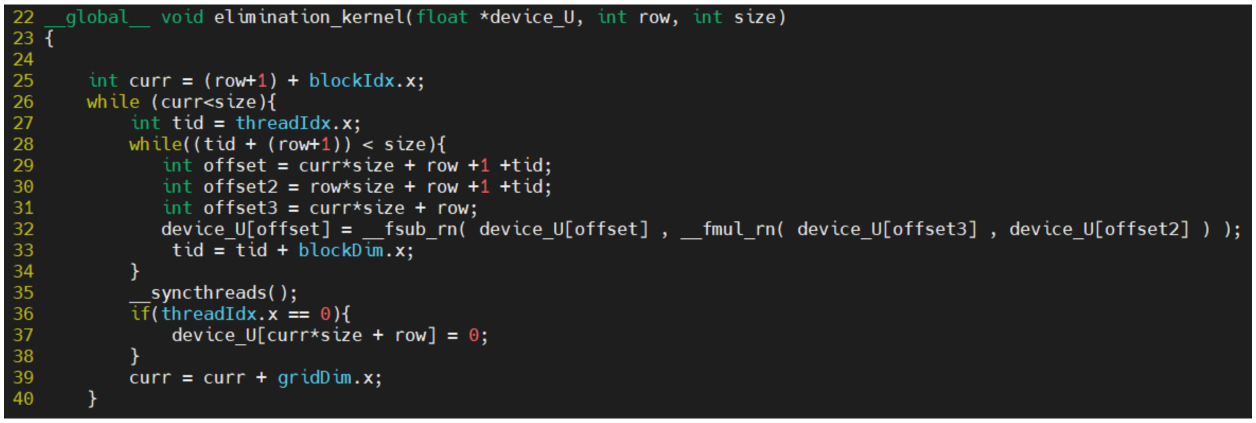
tid = threadIdx.x;

/* while(tid + (row+1)) = for(j = k+1; k < matrix_size; j++) */

set lower diagonal to 0

move current row (curr) by 1 execution grid;

}

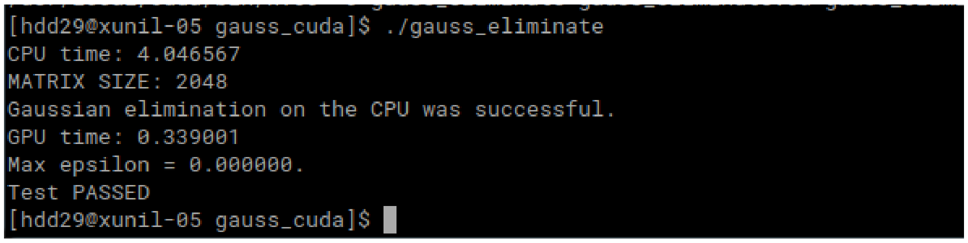


```
__global__ void elimination_kernel(float *device_U, int row, int size)
{

    int curr = (row+1) + blockIdx.x;
    while (curr<size){
        int tid = threadIdx.x;
        while((tid + (row+1)) < size){
            int offset = curr*size + row +1 +tid;
            int offset2 = row*size + row +1 +tid;
            int offset3 = curr*size + row;
            device_U[offset] = __fsub_rn( device_U[offset] , __fmul_rn( device_U[offset3] , device_U[offset2] ) );
             tid = tid + blockDim.x;
        }
        __syncthreads();
        if(threadIdx.x == 0){
            device_U[curr*size + row] = 0;
        }
        curr = curr + gridDim.x;
    }
```

Result and discussion

All accuracy test was passed for a variety of matrix size as suggested in the figure below.



```
[hdd29@xunil-05 gauss_cuda]$ ./gauss_eliminate
CPU time: 4.046567
MATRIX SIZE: 2048
Gaussian elimination on the CPU was successful.
GPU time: 0.339001
Max epsilon = 0.000000.
Test PASSED
[hdd29@xunil-05 gauss_cuda]$
```

<u>Execution time</u>

Matrix size	CPU time	GPU time
512	0.082656	0.007294
1024	0.551565	0.037728
2048	4.04656	0.339001
4096	31.705435	2.843954

Conclusion

As the data moving step between the CPU and GPU is included in the execution time measurements, the trend of more speed up is observed as the matrix size increases. This is because of the one -time overhead of cudaMalloc() and cudaMemcopy() executed is more significant overall compared to the execution time of the algorithm itself on the CPU/GPU.