EC527: High Performance Programming with Multicore and GPUs

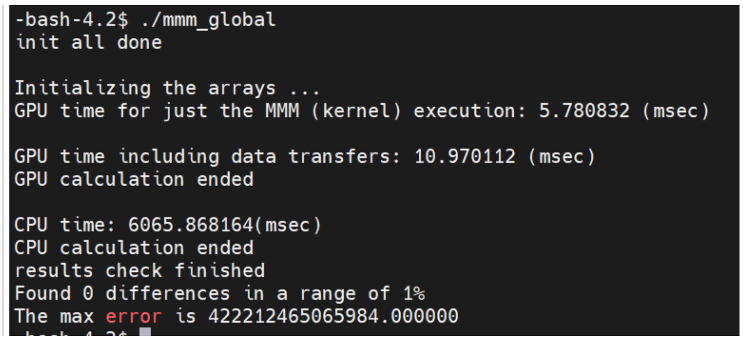
Programming Assignment 8

Part1

After doing the Matrix-Matrix Multiply using the global memory, the total result went well. I tried to set the maximum difference range to be 1%, and the total error is 0. And I generated the matrix element by using the srand(). The function is shown below:

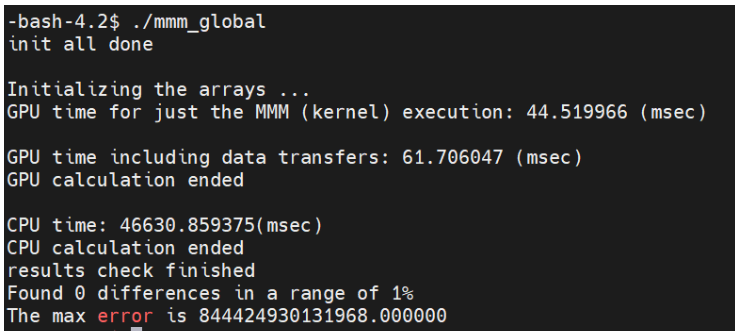
```
void initialize_Array_1D(float *arr, int len, int seed)
{
    int i, j;
    float randNum;
    srand(seed);

    for (i = 0; i < len; i++)
    {
        for (j = 0; j < len; j++)
        {
            randNum = (float)rand();
            arr[i * MATRIX_SIZE + j] = randNum;
        }
    }
}
```

And the maximum difference and running time for the matrix of 1024*1024 is shown below:



```
-bash-4.2$ ./mmm_global
init all done

Initializing the arrays ...
GPU time for just the MMM (kernel) execution: 5.780832 (msec)

GPU time including data transfers: 10.970112 (msec)
GPU calculation ended

CPU time: 6065.868164(msec)
CPU calculation ended
results check finished
Found 0 differences in a range of 1%
The max error is 422212465065984.000000
```

And the result for the matrix of 2048*2048 is:



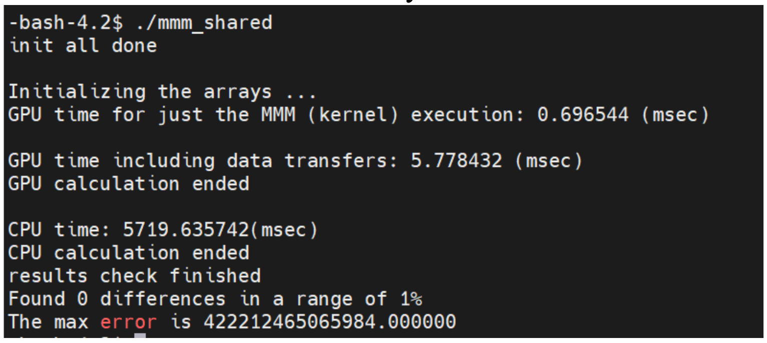
```
-bash-4.2$ ./mmm_global
init all done

Initializing the arrays ...
GPU time for just the MMM (kernel) execution: 44.519966 (msec)

GPU time including data transfers: 61.706047 (msec)
GPU calculation ended

CPU time: 46630.859375(msec)
CPU calculation ended
results check finished
Found 0 differences in a range of 1%
The max error is 844424930131968.000000
```

Part2

The process of matrix multiply for the shared memory is only a bit difference. The only difference is that I use the tile array to calculate and finally put them into the arr3. The function is implemented as below:

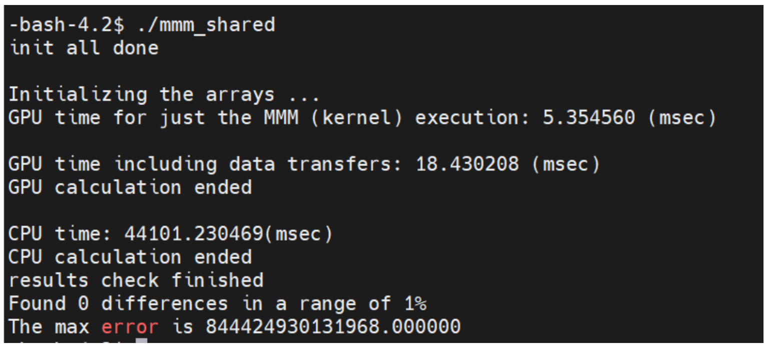
```
__global__ void kernel_mmm(float *arr1, float *arr2, float *arr3, int
arrlen)
{

    const int len = TILE_WIDTH;
    __shared__ float Mds[len][len];
    __shared__ float Nds[len][len];
    int bx = blockIdx.x;
    int by = blockIdx.y;
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    int Row = by * TILE_WIDTH + ty;
    int Col = bx * TILE_WIDTH + tx;

    float Pvalue = 0;
    for (int m = 0; m < arrlen / TILE_WIDTH; ++m)
    {
        Mds[ty][tx] = arr1[Row * arrlen + (m * TILE_WIDTH + tx)];
        Nds[ty][tx] = arr2[Col + (m * TILE_WIDTH + ty) * arrlen];
        __syncthreads();

        for (int k = 0; k < TILE_WIDTH; ++k)
        {
            Pvalue += Mds[ty][k] * Nds[k][tx];
        }
        __syncthreads();
    }

    arr3[Row * arrlen + Col] = Pvalue;
}
```

The result for the matrix of array size of 1024 is:



```
-bash-4.2$ ./mmm_shared
init all done

Initializing the arrays ...
GPU time for just the MMM (kernel) execution: 0.696544 (msec)

GPU time including data transfers: 5.778432 (msec)
GPU calculation ended

CPU time: 5719.635742(msec)
CPU calculation ended
results check finished
Found 0 differences in a range of 1%
The max error is 422212465065984.000000
```

And the result for the 2048 is:

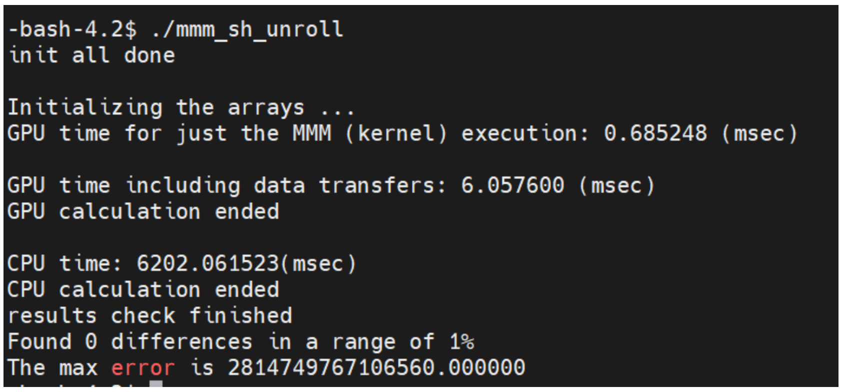


```
-bash-4.2$ ./mmm_shared
init all done

Initializing the arrays ...
GPU time for just the MMM (kernel) execution: 5.354560 (msec)

GPU time including data transfers: 18.430208 (msec)
GPU calculation ended

CPU time: 44101.230469(msec)
CPU calculation ended
results check finished
Found 0 differences in a range of 1%
The max error is 844424930131968.000000
```

By comparing them with part1 above, we can see that when using the shared memory, it is much faster.

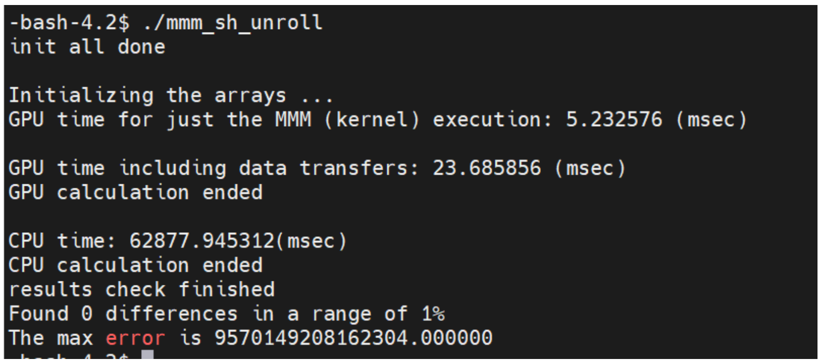
Part3

When using the unrolling factors, I used a factor of 2 and the result shows that it is not faster than the not using unrolling ones. Here is the result for the 1024*1024 matrix.



```
-bash-4.2$ ./mmm_sh_unroll
init all done

Initializing the arrays ...
GPU time for just the MMM (kernel) execution: 0.685248 (msec)

GPU time including data transfers: 6.057600 (msec)
GPU calculation ended

CPU time: 6202.061523(msec)
CPU calculation ended
results check finished
Found 0 differences in a range of 1%
The max error is 2814749767106560.000000
```

And here is the result for the 2048*2048 matrix.



```
-bash-4.2$ ./mmm_sh_unroll
init all done

Initializing the arrays ...
GPU time for just the MMM (kernel) execution: 5.232576 (msec)

GPU time including data transfers: 23.685856 (msec)
GPU calculation ended

CPU time: 62877.945312(msec)
CPU calculation ended
results check finished
Found 0 differences in a range of 1%
The max error is 9570149208162304.000000
```