Hadoop/Cuda Lab
Assignment - 6
CSE 328

8 April 2022

Gyanendra Kumar Shukla
CSE 1
191112040

Department of Computer Science
NIT, Bhopal

Contents

1 Matrix Multiplication

For all the programs, the matrix multiplication kernel would be the same, only the size of the input matrix and the block count was changing.

So, I wrote a program that takes in matrix size and block size as argument, I then passed these values for different questions and analyzed the output.

The *__global__ void matrixMul(int *a, int *b, int *c, int N)* function does the actual matrix multiplication. It takes in pointer to two matrix *a* and *b* and stores the result in third matrix *c*.

The *void init_matrix(int *m, int N, int x)* function initializes the array of size n with values between 1 and x.

The *void verify_matrix(int *a, int *b, int *c, int N)* performs the matrix multiplication on cpu and verifies if the result obtained is correct or not.

The main function expects 2 arguments. The size of the matrix and the number of blocks we want to use. If these parameters are not provided, the program uses a default size of 64*64 and block size of 32.

```
#include <cstdlib>
#include <iostream>

__global__ void matrixMul(int *a, int *b, int *c, int N){
    // calculate global row and column for each thread
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    // check if the thread is valid
    if (row < N && col < N) {
        // calculate the value of the element
        int sum = 0;
        for(int i=0; i<N; i++){
            sum += a[row * N + i] * b[i * N + col];
        }

        // store the value in the result matrix
        c[row * N + col] = sum;
    }
}

// initialize a square matrix between 1 and X
void init_matrix(int *m, int N, int x) {
    for (int i=0; i<N*N; i++) {
        m[i] = rand() % x + 1;
    }
}

// verify the result on cpu
void verify_matrix(int *a, int *b, int *c, int N) {
    for (int i=0; i<N; i++) {
        for (int j=0; j<N; j++) {
            int sum = 0;
```

```
            for (int k=0; k<N; k++) {
                sum += a[i * N + k] * b[k * N + j];
            }
            if (sum != c[i * N + j]) {
                printf("Error: (%d, %d) = %d, should be %d\n", i, j, c[i * N + j],
                ↪  sum);
            }
        }
    }
}

int main(int argc, char **argv) {
    int N;
    int x = 8;
    int blck_size = 32;
    if (argc != 3) {
        N = 1 << 6;
    } else{
        N = atoi(argv[1]);
        blck_size = atoi(argv[2]);
    }

    // square matrix dimension (N*N)
    int bytes = N*N*sizeof(int);


    float gpu_elapsed_time, cpu_elapsed_time;

    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);

    // allocate memory for matrices
    int *a, *b, *c;
    cudaMallocManaged(&a, bytes);
    cudaMallocManaged(&b, bytes);
    cudaMallocManaged(&c, bytes);

    // initialize matrices
    init_matrix(a, N, x);
    init_matrix(a, N, x);

    // set our block and grid dimensions
    int threads = 1*1;
    int blocks = blck_size; // (N + threads - 1) / threads;
    std:: cout << "Matrix size: " << N << "*" << N << std::endl;
    std:: cout << "Block size: " << blck_size << std::endl;
    std:: cout << "Threads: " << threads << "\n\n";


    // setup kernel parameters
    dim3 THREADS(threads, threads);
    dim3 BLOCKS(blocks, blocks);
```

```
    cudaEventRecord(start, 0);

    matrixMul<<<BLOCKS, THREADS>>>(a, b, c, N);
    cudaDeviceSynchronize();

    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);

    cudaEventElapsedTime(&gpu_elapsed_time, start, stop);

    std::cout << "GPU Elapsed Time: " << gpu_elapsed_time << " ms\n";

    cudaEventRecord(start, 0);
    verify_matrix(a, b, c, N);
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&cpu_elapsed_time, start, stop);

    std::cout << "CPU Elapsed Time: " << cpu_elapsed_time << " ms\n";

    std::cout << "Program Completed Successfully\n";
}
```

All the programs were compiled on nvidia gtx 1650 with smi information -

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 471.11       Driver Version: 471.11       CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:01:00.0  On |                  N/A |
| N/A   46C    P8     2W /  N/A |    872MiB /  4096MiB |      5%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

For compilation, use -

```
nvcc .\programname.cu -o outname -ccbin "C:\Program Files (x86)\Microsoft
Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30037\bin\Hostx64\x64"
```

1.1 Question 1

Consider matrix M of size 8x8and N of size 8x8. Assign values to all elements of both the matrices randomly between 1 to 8.Assuming that you have only one processor which can run only one thread, determine execution time for MxN.

1.1.1 OUTPUT

```
.\build\mm  8 1

Matrix size: 8*8
Block size: 1
Threads: 1

GPU Elapsed Time: 0.149568 ms
CPU Elapsed Time: 0.56176 ms
Program Completed Successfully
```

1.1.2 Screenshot

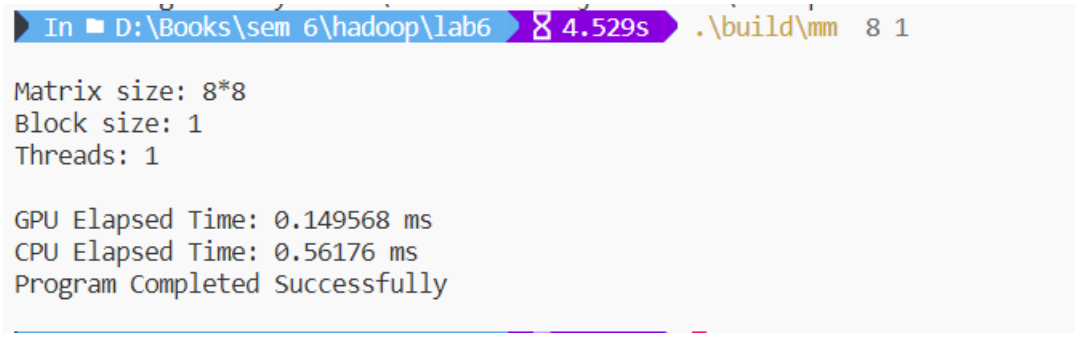

Figure 1: Matrix size = 8*8, block count = 1

1.2 Question 2

Consider matrix M of size 64x64 and N of size 64x64. Assign values to all elements of both the matrices randomly between 1 to 8. Assuming that you have only one processor which can run only one thread, determine execution time for MxN.

1.2.1 OUTPUT

```
.\build\mm  64 1

Matrix size: 64*64
Block size: 1
Threads: 1

GPU Elapsed Time: 0.1536 ms
CPU Elapsed Time: 1.31165 ms
Program Completed Successfully
```

1.2.2 Screenshot

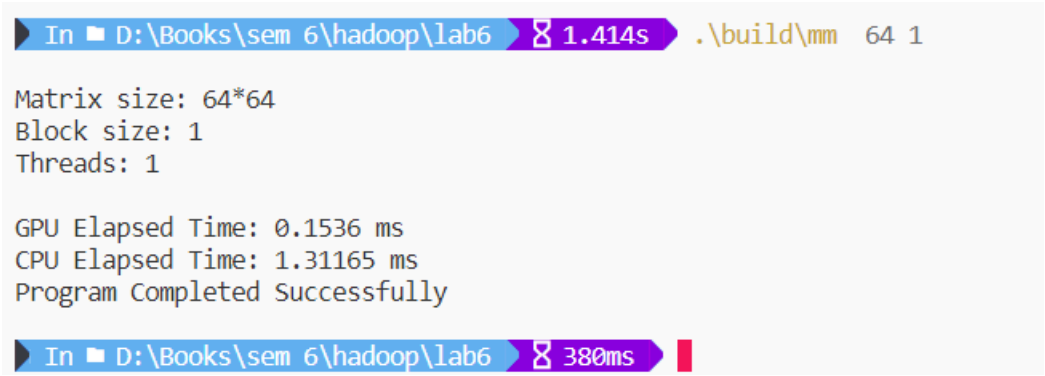

Figure 2: Matrix size = 64*64, block count = 1

1.3 Question 3

Consider matrix M of size 8x8 and N of size 8x8. Assign values to all elements of both the matrices randomly between 1 to 8. Assuming that you have fourprocessors and eachcan run only one thread, determine execution time for MxN

1.3.1 OUTPUT

```
.\build\mm  8 4

Matrix size: 8*8
Block size: 4
Threads: 1

GPU Elapsed Time: 0.161792 ms
CPU Elapsed Time: 0.738048 ms
Program Completed Successfully
```

1.3.2 Screenshot

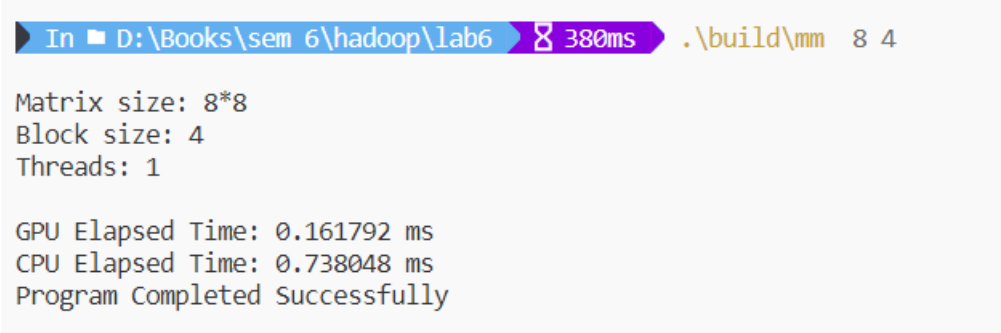

Figure 3: Matrix size = 8*8, block count = 4

1.4 Question 4

Consider matrix M of size 64x64 and N of size 64x64. Assign values to all elements of both the matrices randomly between 1 to 8. Assuming that you have eight processors and eachcan run only one thread, determine execution time for MxN.

1.4.1 OUTPUT

```
.\build\mm  64 4

Matrix size: 64*64
Block size: 4
Threads: 1

GPU Elapsed Time: 0.154656 ms
CPU Elapsed Time: 1.37859 ms
Program Completed Successfully
```

1.4.2 Screenshot

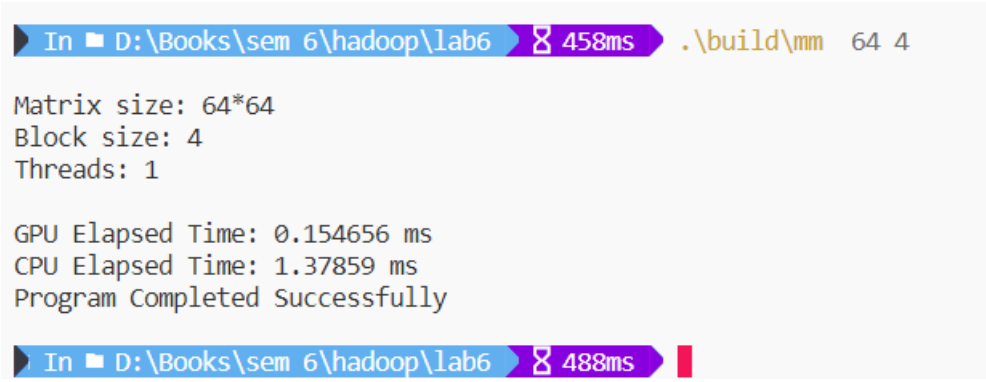

Figure 4: Matrix size = 64*64, block count = 4