

به نام خدا
دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

برنامه‌نویسی چندهسته‌ای

گزارش کار آزمایش ۶

سید محمد حجازی حسینی

۹۷۳۳۰۲۰

بهار ۱۴۰۱

گام (۱)

درستی کد برای ۳ در ۳:

```
Microsoft Visual Studio Debug Console
GPU Device 0: "GeForce GT 1030" with compute capability 6.1

[-] N = 3
MatrixA(3,3), MatrixB(3,3)
Computing result using CUDA Kernel...
Elapsed time in msec = 0.007168
Array A:
1.000000 7.000000 4.000000
0.000000 9.000000 4.000000
8.000000 8.000000 2.000000
Array B:
4.000000 5.000000 5.000000
1.000000 7.000000 1.000000
1.000000 5.000000 2.000000
Array C:
15.000000 74.000000 20.000000
13.000000 83.000000 17.000000
42.000000 106.000000 52.000000

C:\Users\sayedmohammad\source\repos\Lab6\x64\Release\Lab6.exe (process 1008) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

گام (۲)

روش یک

برای $n = 2048$

$32 \times 32 \times 64 \times 64 = 2048 \times 2048$ پس tile 64×64 یک thread و هر thread = (32, 32, 1)

```
Microsoft Visual Studio Debug Console
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "GeForce GT 1030" with compute capability 6.1

[-] N = 2048
MatrixA(2048,2048), MatrixB(2048,2048)
Computing result using CUDA Kernel...
Elapsed time in msec = 7476.853027

C:\Users\sayedmohammad\source\repos\Lab6\x64\Release\Lab6.exe (process 9996) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

روش دوم)

$n = 2048$

باید $\text{grid} = (64, 64, 1)$ و $\text{thread} = (32, 32, 1)$ باشد تا $32 \times 32 \times 64 \times 64 = 2048 \times 2048$

```
Microsoft Visual Studio Debug Console
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "GeForce GT 1030" with compute capability 6.1

[-] N = 2048
MatrixA(2048,2048), MatrixB(2048,2048)
Computing result using CUDA Kernel...
Elapsed time in msec = 304.964569

C:\Users\seyedmohammad\source\repos\Lab6\x64\Release\Lab6.exe (process 20340) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

روش سوم)

$n = 2048$

باید $\text{grid} = (4, 4, 1)$ و $\text{thread} = (32, 32, 1)$ و $\text{tile} = 16 \times 16$ باشد. البته می‌توانیم مقدار tile را کاهش و grid را افزایش دهیم. مثلاً $\text{grid} = (16, 16, 1)$ و $\text{tile} = 4 \times 4$ ، به همین صورت یعنی $\text{tile} = 4 \times 4$ پیش می‌رویم:

```
Microsoft Visual Studio Debug Console
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "GeForce GT 1030" with compute capability 6.1

[-] N = 2048
MatrixA(2048,2048), MatrixB(2048,2048)
Computing result using CUDA Kernel...
Elapsed time in msec = 2161.592773

C:\Users\seyedmohammad\source\repos\Lab6\x64\Release\Lab6.exe (process 9460) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

نتایج:

روش ۱ = 7476.85

روش ۲ = 304.96

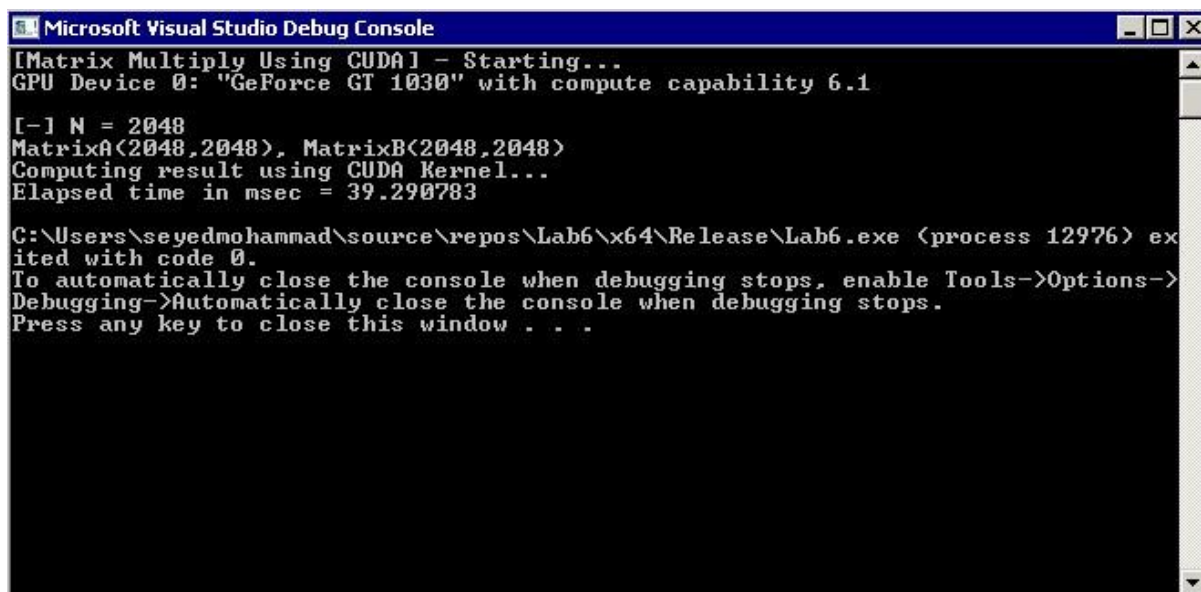
روش ۳ = 2161.59

پس روش دوم از همه بهتر عمل کرده است.

گام ۳

tile = 4 x 4, thread = (32, 32, 1), grid = (16, 16, 1)

با استفاده از share memory سرعت بهبود پیدا می کند:



```
Microsoft Visual Studio Debug Console
[Matrix Multiply Using CUDA] - Starting...
GPU Device 0: "GeForce GT 1030" with compute capability 6.1

[-] N = 2048
MatrixA(2048,2048), MatrixB(2048,2048)
Computing result using CUDA Kernel...
Elapsed time in msec = 39.290783

C:\Users\seyedmohammad\source\repos\Lab6\x64\Release\Lab6.exe (process 12976) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

کد مربوط به kernel:

```
#define TILE_WIDTH 4
#define BLOCK_DIM 32
__global__ void
matrixMulCUDA3(float *C, float *A, float *B, int n)
{
    int start_row = blockDim.y * blockIdx.y + threadIdx.y * TILE_WIDTH;
    int end_row = start_row + TILE_WIDTH;
    int start_col = blockDim.x * blockIdx.x + threadIdx.x * TILE_WIDTH;
    int end_col = start_col + TILE_WIDTH;

    for (int row = start_row; row < end_row; row++) {
        for (int col = start_col; col < end_col; col++) {
            float CValue = 0;

            __shared__ float As[BLOCK_DIM][BLOCK_DIM];
            __shared__ float Bs[BLOCK_DIM][BLOCK_DIM];

            for (int z = 0; z < n / (BLOCK_DIM * TILE_WIDTH); z++) {

                As[threadIdx.y][threadIdx.x] = A[row * n + z * BLOCK_DIM + threadIdx.x];
                Bs[threadIdx.y][threadIdx.x] = B[(z * BLOCK_DIM + threadIdx.y) * n + col];

                __syncthreads();

                for (int k = 0; k < BLOCK_DIM; ++k)
                    CValue += As[threadIdx.y][k] * Bs[k][threadIdx.x];

                __syncthreads();
            }

            C[row * n + col] = CValue;
        }
    }
}
```

توضیح:

هر نخ همان tile خود را محاسبه می‌کند. اما با این تفاوت که اینبار برای استفاده از share memory عملیات جمع روی ضرب روی درایه‌ها را به تعداد block ها تقسیم می‌کنیم. در هر iteration یک block از global memory خوانده می‌شود و در share memory ذخیره می‌شود. سپس با استفاده از آن یک متغیر میانی را حساب می‌کنیم. در iteration بعدی جواب block بعدی را با قبلی جمع می‌کنیم و همینطور پیش می‌رویم. در نهایت همه‌ی جمع‌ها را در خانه‌ی مربوطه‌ی global memory ذخیره می‌کنیم.