

Midterm project : Advanced Programming on GPU

Code :

```
#include <cuda_runtime_api.h>
#include "device_launch_parameters.h"
#include "J:\\ami\\ami\\common\\book.h"
#include <cuda_runtime_api.h>
#include "device_launch_parameters.h"
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#include <time.h>
#include "..\\ami\\ami\\common\\cpu_bitmap.h"
#include <math.h>

#define TILE_WIDTH 1
#define rnd( z ) (z * rand() / RAND_MAX)

__global__ void
MatrixMul(float *Md, float *Nd, float *Pd, const int N)
{
    unsigned int col = TILE_WIDTH*blockIdx.x + threadIdx.x;

    unsigned int row = TILE_WIDTH*blockIdx.y + threadIdx.y;

    for (int k = 0; k<N; k++)
    {
        Pd[row*N + col] += Md[row * N + k] * Nd[k * N + col];
    }
}

__global__ void
MatrixAdd(float *Ad, float *Bd, float *Cd, const int N)
{
    unsigned int col = TILE_WIDTH*blockIdx.x + threadIdx.x;

    unsigned int row = TILE_WIDTH*blockIdx.y + threadIdx.y;

    for (int k = 0; k<N; k++)
    {
        Cd[row*N + col] = Ad[row*N + col] + Bd[row*N + col];
    }
}
```

```

int main()
{
    const int N = 3;
    float a[N][N], b[N][N],
          c[N][N], m_c[N][N], a_c[N][N];
    float *dev_a_d, *dev_b_d, *dev_c_d, *m_dev_c_d, *a_dev_c_d;
    int i, j;
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < N; j++)
        {
            a[i][j] = rnd(20);
            b[i][j] = rnd(14);
        }
    }

    cudaMalloc((void **)&dev_a_d, N*N*sizeof (int));

    cudaMalloc((void **)&dev_b_d, N*N*sizeof (int));

    cudaMemcpy(dev_a_d, a, N*N*sizeof (int), cudaMemcpyHostToDevice);

    cudaMemcpy(dev_b_d, b, N*N*sizeof (int), cudaMemcpyHostToDevice);

    cudaMalloc((void **)&dev_c_d, N*N*sizeof (int));

    cudaMalloc((void **)&m_dev_c_d, N*N*sizeof (int));

    cudaMalloc((void **)&a_dev_c_d, N*N*sizeof (int));

    dim3 dimGrid(N / TILE_WIDTH, N / TILE_WIDTH, 1);

    dim3 dimBlock(TILE_WIDTH, TILE_WIDTH, 1);

    MatrixMul << <dimGrid, dimBlock >> > (dev_a_d, dev_b_d, m_dev_c_d, N);
    MatrixAdd << <dimGrid, dimBlock >> > (dev_a_d, dev_b_d, a_dev_c_d, N);

```

```
cudaMemcpy(m_c, m_dev_c_d, N*N*sizeof(int),cudaMemcpyDeviceToHost);
```

```
cudaMemcpy(a_c, a_dev_c_d, N*N*sizeof(int),cudaMemcpyDeviceToHost);
```

```
printf("matrix 1 \n");
```

```
for (i = 0; i<N; i++)
{
    for (j = 0; j < N; j++)
    {
        printf("%f ", a[i][j]);

    }
    printf("\n");
}
```

```
printf("matrix 2 \n");
```

```
for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
    {
        printf("%f ", b[i][j]);

    }
    printf("\n");
}
```

```
printf("multiplying the two arrays \n");
```

```
printf("we get \n");
```

```
for (i = 0; i<N; i++)
{
    for (j = 0; j < N; j++)
    {
        printf("%f ", m_c[i][j]);

    }
    printf("\n");
}
```

```
printf("adding the two matrix \n");
```

```
for (i = 0; i < N; i++)
{
```

```

        for (j = 0; j < N; j++)
        {
            printf("%f ", a_c[i][j]);
        }
        printf("\n");
    }
    system("pause");
}

```

Screenshot :

The screenshot shows the Microsoft Visual Studio IDE with a C program being executed. The program prints two 3x3 matrices, multiplies them, and adds the result to the first matrix. The output window shows the matrices and the resulting values.

Program Output:

```

matrix 1
0.000000 3.000000 11.000000
7.000000 16.000000 3.000000
14.000000 6.000000 1.000000
matrix 2
7.000000 11.000000 6.000000
12.000000 10.000000 12.000000
7.000000 0.000000 5.000000
multiplying the two arrays
we get
113.000000 30.000000 91.000000
262.000000 237.000000 249.000000
177.000000 214.000000 161.000000
adding the two matrix
7.000000 14.000000 17.000000
19.000000 26.000000 15.000000
21.000000 6.000000 6.000000
Press any key to continue . . .

```

Source Code:

```

for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
    {
        printf("%f ", b[i][j]);
    }
    printf("\n");
}

printf("multiplying the two arrays \n");
printf("we get \n");

for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
    {
        printf("%f ", m_c[i][j]);
    }
}

```