HPCL Ass.4 : BE_34_Samruddhi Khairnar 15/04

Write a CUDA Program for :

1. Addition of two large vectors

2. Matrix Multiplication using CUDA C.

Code :

```
// HPCL_4_BE_34 - Vector addition, Matrix Multiplication - CUDA C

# include <stdio.h>

__global__ void add_vectors(int *g_a, int *g_b, int *g_c)
{
  int i = threadIdx.x + blockDim.x * blockIdx.x;
  g_c[i] = g_a[i] + g_b[i];
}

void run_vector_addition()
{
    printf("Enter the length of both the vectors : ");
    int n;
    scanf("%d", &n);
    int c_a[n], c_b[n], c_c[n];
    for (int i=0;i<n;i++)
    {
        printf("Enter element %d of Vector 1 : ", i);
        scanf("%d", &c_a[i]);
        printf("Enter element %d of Vector 2 : ", i);
        scanf("%d", &c_b[i]);
    }

  int *g_a, *g_b, *g_c;

    int size = n*sizeof(int);
    cudaMalloc(&g_a, size);
    cudaMalloc(&g_b, size);
    cudaMalloc(&g_c, size);
    cudaMemcpy(g_a, c_a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(g_b, c_b, size, cudaMemcpyHostToDevice);
    cudaMemcpy(g_c, c_c, size, cudaMemcpyHostToDevice);
    add_vectors<<<5,5>>>(g_a, g_b, g_c); // 5 Blocks * 5 Threads = 25 Threads in total.
    cudaDeviceSynchronize();
    cudaMemcpy(c_c, g_c, size, cudaMemcpyDeviceToHost);

    cudaFree(g_a);
    cudaFree(g_b);
    cudaFree(g_c);

    printf(" ");
    for (int i=0;i<n;i++)
        printf("%d ", c_a[i]);
    printf("\n+");
    for (int i=0;i<n;i++)
        printf("%d ", c_b[i]);
    printf("\n=");
    for (int i=0;i<n;i++)
        printf("%d ", c_c[i]);
    printf("\n");
}

__global__ void matrix_multiply(int *g_a, int *g_b, int *g_c, int size)
{

/*

I have converted the 2D array of 3*3 elements into a 1D array of 9 elements hence multiplying the block dimension to the row index of each of the matrices - A, B, C. Shinde ma'am has used 2D array hence x,y dimensions.
    0   1   2
0 {A00,A01,A02}   {B00,B01,B02}   {C00,C01,C02}
1 {A10,A11,A12} X {B10,B11,B12} = {C10,C11,C12}
2 {A20,A21,A22}   {B20,B21,B22}   {C20,C21,C22}

C00 = A00*B00 + A01*B10 + A02*B20
C01 = A00*B01 + A01*B11 + A02*B21

Hence, we must have a loop - from 0 to 2:
C00, C01 = 0, 0

for i in range(0,3):
    C00 += A0i * Bi0
    C01 += A0i * Bi1

Hence, in a loop, C[Block_id][thread_id] += A[Block_id][i] *
```

```
B[i][thread_id]
And as I have converted all of them into 1D arrays, The left index is
multiplied by Block_dim (no.of threads in a block)

for i in range(0,3):
    C[Block_id*Block_dim + thread_id] += A[Block_id*Block_dim + i] *
B[i*Block_dim +thread_id]

*/

  int idx = blockDim.x * blockIdx.x + threadIdx.x;
  for(int i=0; i<size; i++)
      g_c[idx] += g_a[blockDim.x * blockIdx.x + i] * g_b[i * blockDim.x +
threadIdx.x];
}

void run_matrix_multiplication()
{
    int n=3;
    int c_c[3][3] = {{0,0,0},
                     {0,0,0},
                     {0,0,0}};

    int c_a[n][n], c_b[n][n];
    for (int i=0;i<n;i++)
    {
        for (int j=0;j<n;j++)
        {
            printf("Enter element %d,%d of Matrix A : ", i, j);
            scanf("%d", &c_a[i][j]);
            printf("Enter element %d,%d of Matrix B : ", i, j);
            scanf("%d", &c_b[i][j]);
        }
    }

  int *g_a, *g_b, *g_c;

  int size = n*n*sizeof(int);
  cudaMalloc(&g_a, size);
  cudaMalloc(&g_b, size);
  cudaMalloc(&g_c, size);
  cudaMemcpy(g_a, c_a, size, cudaMemcpyHostToDevice);
  cudaMemcpy(g_b, c_b, size, cudaMemcpyHostToDevice);
  cudaMemcpy(g_c, c_c, size, cudaMemcpyHostToDevice);

  matrix_multiply<<<3,3>>>(g_a, g_b, g_c, size); // 3 Blocks, each having
```

```
3 threads
  cudaDeviceSynchronize();
  cudaMemcpy(c_c, g_c, size, cudaMemcpyDeviceToHost);

  cudaFree(g_a);
  cudaFree(g_b);
  cudaFree(g_c);

    printf("A = \n");
    for (int i=0;i<n;i++)
    {
        for (int j=0;j<n;j++)
            printf("%d,", c_a[i][j]);
        printf("\n\n");
    }
    printf("B = \n");
    for (int i=0;i<n;i++)
    {
        for (int j=0;j<n;j++)
            printf("%d,", c_b[i][j]);
        printf("\n\n");
    }
    printf("Multiplication = \n");
    for (int i=0;i<n;i++)
    {
        for (int j=0;j<n;j++)
            printf("%d,", c_c[i][j]);
        printf("\n\n");
    }
}

int main()
{
    int ch;
    while(true)
    {
        printf("Enter 1-Vector Addition | 2-Matrix Multiplication | 0-Exit
: ");
        scanf("%d", &ch);
        if(ch==1)
            run_vector_addition();
        else if(ch==2)
            run_matrix_multiplication();
        else if(ch==0)
        {
            printf("Exited Successfully.\n");
            break;
        }
        else
            printf("Invalid input.\n");
    }

    return 0;
}
```

Output :

```
comp-proj-sys05@compprojsys05-OptiPlex-3010:~/Downloads$ nvcc HPCL_4_BE_34.cu
comp-proj-sys05@compprojsys05-OptiPlex-3010:~/Downloads$ ./a.out
Enter 1-Vector Addition | 2-Matrix Multiplication | 0-Exit : 1
Enter the length of both the vectors : 5
Enter element 0 of Vector 1 : 1
Enter element 0 of Vector 2 : 1
Enter element 1 of Vector 1 : 2
Enter element 1 of Vector 2 : 2
Enter element 2 of Vector 1 : 3
Enter element 2 of Vector 2 : 3
Enter element 3 of Vector 1 : 4
Enter element 3 of Vector 2 : 4
Enter element 4 of Vector 1 : 2
Enter element 4 of Vector 2 : 2
 1 2 3 4 2
+1 2 3 4 2
=2 4 6 8 4
Enter 1-Vector Addition | 2-Matrix Multiplication | 0-Exit : 2
Enter element 0,0 of Matrix A : 1
Enter element 0,0 of Matrix B : 1
Enter element 0,1 of Matrix A : 2
Enter element 0,1 of Matrix B : 0
Enter element 0,2 of Matrix A : 3
Enter element 0,2 of Matrix B : 0
Enter element 1,0 of Matrix A : 4
Enter element 1,0 of Matrix B : 0
Enter element 1,1 of Matrix A : 5
Enter element 1,1 of Matrix B : 1
Enter element 1,2 of Matrix A : 6
Enter element 1,2 of Matrix B : 0
Enter element 2,0 of Matrix A : 7
Enter element 2,0 of Matrix B : 0
Enter element 2,1 of Matrix A : 8
Enter element 2,1 of Matrix B : 0
Enter element 2,2 of Matrix A : 9
Enter element 2,2 of Matrix B : 1
A =
1,2,3,

4,5,6,

7,8,9,

B =
1,0,0,

0,1,0,

0,0,1,

Multiplication =
1,2,3,

4,5,6,

7,8,9,

Enter 1-Vector Addition | 2-Matrix Multiplication | 0-Exit : 0
Exited Successfully.
```