Manthan Gopal Dhole
201080020
Subject :Parallel Computing

Lab Experiment 1

Aim :

Introduction to Cuda

Write a CUDA program for

1. Matrix Addition
2. Matrix Transpose
3. Matrix Multiplication

run it in Jupyter notebook in google colab

Theory :

CUDA is a model created by Nvidia for parallel computing platforms and application programming interfaces. CUDA is the parallel computing architecture of NVIDIA which allows for dramatic increases in computing performance by harnessing the power of the GPU.
Google Colab is a free cloud service and the most important feature able to distinguish Colab from other free cloud services is; Colab offers GPU and is completely free! With Colab you can work on the GPU with CUDA C/C++ for free!

CUDA code will not run on AMD CPU or Intel HD graphics unless you have NVIDIA hardware inside your machine. On Colab you can take advantage of the Nvidia GPU as well as being a fully functional Jupyter Notebook with pre-installed Tensorflow and some other ML/DL tools.

CUDA (Compute Unified Device Architecture) is a parallel computing platform and API created by NVIDIA. CUDA provides a set of functions to execute code on GPUs (Graphical Processing Units) to perform complex computations much faster than a CPU (Central Processing Unit) alone. Some of the main CUDA functions are:

1. cudaMalloc() - This function is used to allocate memory on the GPU.
2. cudaMemcpy() - This function is used to copy data from the host to the device or vice versa.
3. cudaFree() - This function frees memory that has been previously allocated on the GPU.
4. global - This keyword is used to specify a function that runs on the GPU and can be called from the host code.
5. device - This keyword is used to specify a function that runs on the GPU and can only be called from other GPU functions.
6. threadIdx - This variable contains the thread ID within a block.
7. blockIdx - This variable contains the block ID within a grid.
8. blockDim - This variable contains the dimensions of a block.
9. gridDim - This variable contains the dimensions of a grid.

These functions and variables are used to write and execute CUDA code on GPUs.

Link:

https://colab.research.google.com/drive/1swGAydL_inxtdS5BryEjwQ0aAckEVIDT?usp=sharing

Code :

Addition of Matrix

```
%%cu
#include <stdio.h>
#include <stdlib.h>

#define N 5
#define BLOCK_DIM 10

__global__ void matrixAdd (int *a, int *b, int *c) {
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;

    int index = col + row * N;

    if (col < N && row < N) {
        c[index] = a[index] + b[index];
    }

}

int main() {
    int a[N][N], b[N][N], c[N][N];
    int *dev_a, *dev_b, *dev_c;

    int size = N * N * sizeof(int);
    for(int i=0; i<N; i++)
        for (int j=0; j<N; j++){
            a[i][j] = 10;
            b[i][j] = 2;
        }

    cudaMalloc((void**)&dev_a, size);
    cudaMalloc((void**)&dev_b, size);
    cudaMalloc((void**)&dev_c, size);

    cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

    dim3 dimBlock(BLOCK_DIM, BLOCK_DIM);
    //dim3 dimGrid((int)ceil(N/dimBlock.x),(int)ceil(N/dimBlock.y));
    dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x,
(N+dimBlock.y-1)/dimBlock.y);
    printf("dimGrid.x = %d, dimGrid.y = %d\n", dimGrid.x, dimGrid.y);
    matrixAdd<<<dimGrid,dimBlock>>>(dev_a,dev_b,dev_c);
    cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);
    printf("Matrix a is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", a[i][j] );
        }
        printf("\n");
    }

    printf("Matrix b is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", b[i][j] );
        }
        printf("\n");
    }

    printf("Result after a*b is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
                printf("%d\t", c[i][j] );
            }
        printf("\n");
    }


    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
}
```

Output:

```
Matrix a is:
10      10      10      10      10
10      10      10      10      10
10      10      10      10      10
10      10      10      10      10
10      10      10      10      10
Matrix b is:
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
Result after a*b is:
12      12      12      12      12
12      12      12      12      12
12      12      12      12      12
12      12      12      12      12
12      12      12      12      12
```

Multiplication of Matrix

```
%%cu
#include <stdio.h>
#include <stdlib.h>

#define N 5
#define BLOCK_DIM 10

__global__ void matrixMult (int *a, int *b, int *c) {
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;



    if (col < N && row < N) {
        int tempSum=0;

        for(int i=0;i<N;i++){
            tempSum += a[row * N + i] * b[i * N + col];
        }

        c[row * N + col] = tempSum;
    }

}

int main() {
    int a[N][N], b[N][N], c[N][N];
    int *dev_a, *dev_b, *dev_c;

    int size = N * N * sizeof(int);

    for(int i=0; i<N; i++)
        for (int j=0; j<N; j++){
            a[i][j] = 3;
            b[i][j] = 2;
        }

    cudaMalloc((void**)&dev_a, size);
    cudaMalloc((void**)&dev_b, size);
    cudaMalloc((void**)&dev_c, size);

    cudaMemcpy(dev_a, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b, size, cudaMemcpyHostToDevice);

    dim3 dimBlock(BLOCK_DIM, BLOCK_DIM);
    //dim3 dimGrid((int)ceil(N/dimBlock.x),(int)ceil(N/dimBlock.y));
    dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x,
(N+dimBlock.y-1)/dimBlock.y);
    printf("dimGrid.x = %d, dimGrid.y = %d\n", dimGrid.x, dimGrid.y);
    matrixMult<<<dimGrid,dimBlock>>>(dev_a,dev_b,dev_c);
    cudaMemcpy(c, dev_c, size, cudaMemcpyDeviceToHost);
    printf("Matrix a is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", a[i][j] );
        }
        printf("\n");
    }

    printf("Matrix b is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", b[i][j] );
        }
        printf("\n");
    }

    printf("Result after a*b is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", c[i][j] );
        }
        printf("\n");
    }


    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
}
```

Output:

```
Matrix a is:
3       3       3       3       3
3       3       3       3       3
3       3       3       3       3
3       3       3       3       3
3       3       3       3       3
```

```
Matrix b is:
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
2       2       2       2       2
Result after a*b is:
30      30      30      30      30
30      30      30      30      30
30      30      30      30      30
30      30      30      30      30
30      30      30      30      30
```

.

Transpose of Matrix

```
%%cu
#include <stdio.h>
#include <stdlib.h>

#define N 5
#define BLOCK_DIM 10

__global__ void matrixTrans(int *ain, int *aout) {
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int row = blockIdx.y * blockDim.y + threadIdx.y;



    if (col < N && row < N) {
        aout[col*N+row]=ain[row*N+col];
    }

}

int main() {
    int a_in[N][N], a_out[N][N];;
    int *dev_ain, *dev_aout;
    int size = N * N * sizeof(int);

    for(int i=0; i<N; i++)
        for (int j=0; j<N; j++){
            a_in[i][j] = i*i+j;
        }

    cudaMalloc((void**)&dev_ain, size);
    cudaMalloc((void**)&dev_aout, size);

    cudaMemcpy(dev_ain, a_in, size, cudaMemcpyHostToDevice);
    cudaMemcpy(dev_aout, a_out, size, cudaMemcpyHostToDevice);

    dim3 dimBlock(BLOCK_DIM, BLOCK_DIM);
    //dim3 dimGrid((int)ceil(N/dimBlock.x),(int)ceil(N/dimBlock.y));
    dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x,
(N+dimBlock.y-1)/dimBlock.y);
    printf("dimGrid.x = %d, dimGrid.y = %d\n", dimGrid.x, dimGrid.y);
    matrixTrans<<<dimGrid,dimBlock>>>(dev_ain,dev_aout);
    cudaMemcpy(a_out, dev_aout, size, cudaMemcpyDeviceToHost);
    printf("Input Matrix is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", a_in[i][j] );
        }
        printf("\n");
    }

    printf("Transpose Matrix is:\n");
    for(int i=0; i<N; i++){
        for (int j=0; j<N; j++){
            printf("%d\t", a_out[i][j] );
        }
        printf("\n");
    }

}
```

Output:

```
Input Matrix is:
1       2       3       4       5
2       3       4       5       6
5       6       7       8       9
10      11      12      13      14
17      18      19      20      21
Transpose Matrix is:
1       2       5       10      17
2       3       6       11      18
3       4       7       12      19
4       5       8       13      20
5       6       9       14      21
```

Conclusion :

Thus, in this experiment we studied cuda programming and understood how parallel programming works and the importance of the gpu in parallel computing.
We used google colab platform to run programs for matrix addition, transpose and multiplication.
We studied various functions such as cudaMalloc for memory allocation, cuda Free for memory deallocation and the built-in device variables blockDim, blockldx, and threadIdx used to identify and differentiate GPU threads that execute the kernel in parallel.