LP5- High Performance Computing Practical 4

Write a CUDA Program for :

1. Addition of two large vectors
2. Matrix Multiplication using CUDA C

Name : Kaustubh Shrikant Kabra

Rollno : 37

BE COMP 1

In [1]:
```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

In [ ]:
```
"""
Removing all  the previous version
"""
# !apt-get --purge remove cuda nvidia* libnvidia-*
# !dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge
# !apt-get remove cuda-*
# !apt autoremove
# !apt-get update
```

In [ ]:
```
!wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-ubur
!sudo mv cuda-ubuntu2204.pin /etc/apt/preferences.d/cuda-repository-pin-600
!wget https://developer.download.nvidia.com/compute/cuda/11.8.0/local_installers/cuda-repo
!sudo dpkg -i cuda-repo-ubuntu2204-11-8-local_11.8.0-520.61.05-1_amd64.deb
!sudo cp /var/cuda-repo-ubuntu2204-11-8-local/cuda-*-keyring.gpg /usr/share/keyrings/
!sudo apt-get update
!sudo apt-get -y install cuda
```

Installing Libraries

In [3]:
```
!pip install git+https://github.com/andreinechaev/nvcc4jupyter.git
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Collecting git+https://github.com/andreinechaev/nvcc4jupyter.git
  Cloning https://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-aw95zdbs
  Running command git clone --filter=blob:none --quiet https://github.com/andreinechaev/nv
cc4jupyter.git /tmp/pip-req-build-aw95zdbs
  Resolved https://github.com/andreinechaev/nvcc4jupyter.git to commit aac710a35f52bb78ab3
4d2e52517237941399eff
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-py3-none-any.whl size=4305 sha25
6=e1c6e1b43bde1123b2e4491419a96b8a67c29c0acf87dfb5fd635abe1c3f268b
  Stored in directory: /tmp/pip-ephem-wheel-cache-jkhue099/wheels/a8/b9/18/23f8ef71ceb0f63
297dd1903aedd067e6243a68ea756d6feea
Successfully built NVCCPlugin
```

```
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
```

In [4]:

```
%load_ext nvcc_plugin
```

```
created output directory at /content/src
Out bin /content/result.out
```

Vector Addition

In [6]:

```
%%cu
#include "stdio.h"
#include <iostream>
#include <cuda.h>
#include <cuda_runtime.h>

// Defining number of elements in Array
#define N 5

// Defining Kernel function for vector addition
__global__ void gpuAdd(int *d_a, int *d_b, int *d_c)
{
    // Getting block index of current kernel
    int tid = blockIdx.x; // handle the data at this index
    if (tid < N)
        d_c[tid] = d_a[tid] + d_b[tid];
}
// Main program
int main(void)
{
    // Defining host arrays
    int h_a[N], h_b[N], h_c[N];
    // Defining device pointers
    int *d_a, *d_b, *d_c;
    // allocate the memory
    cudaMalloc((void**)&d_a, N * sizeof(int));
    cudaMalloc((void**)&d_b, N * sizeof(int));
    cudaMalloc((void**)&d_c, N * sizeof(int));
    // Initializing Arrays
    for (int i = 0; i < N; i++) {
        h_a[i] = 2*i*i;
        h_b[i] = i ;
    }

    // Copy input arrays from host to device memory
    cudaMemcpy(d_a, h_a, N * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, N * sizeof(int), cudaMemcpyHostToDevice);

    // Calling kernels with N blocks and one thread per block, passing
    // device pointers as parameters
    gpuAdd <<<N, 1 >>>(d_a, d_b, d_c);

    // Copy result back to host memory from device memory
    cudaMemcpy(h_c, d_c, N * sizeof(int), cudaMemcpyDeviceToHost);
    printf("Vector addition on GPU \n");

    // Printing result on console
    for (int i = 0; i < N; i++) {
        printf("The sum of %d element is %d + %d = %d\n",
            i, h_a[i], h_b[i],h_c[i]);
    }
    // Free up memory
    cudaFree(d_a);
```

```
    cudaFree(d_b);
    cudaFree(d_c);
    return 0;
}
```

```
Vector addition on GPU
The sum of 0 element is 0 + 0 = 0
The sum of 1 element is 2 + 1 = 3
The sum of 2 element is 8 + 2 = 10
The sum of 3 element is 18 + 3 = 21
The sum of 4 element is 32 + 4 = 36
```

Matrix Multiplication

In [7]:

```
%%cu
#include <iostream>
#include <cuda.h>

using namespace std;

#define BLOCK_SIZE 2

__global__ void gpuMM(float *A, float *B, float *C, int N)
{
    // Matrix multiplication for NxN matrices C=A*B
    // Each thread computes a single element of C
    int row = blockIdx.y*blockDim.y + threadIdx.y;
    int col = blockIdx.x*blockDim.x + threadIdx.x;

    float sum = 0.f;
    for (int n = 0; n < N; ++n)
        sum += A[row*N+n]*B[n*N+col];

    C[row*N+col] = sum;
}

int main(int argc, char *argv[])
{int N;float K;
    // Perform matrix multiplication C = A*B
    // where A, B and C are NxN matrices
    // Restricted to matrices where N = K*BLOCK_SIZE;
        cout<<"Enter a Value for Size/2 of matrix";
        cin>>K;

    K = 2;
    N = K*BLOCK_SIZE;

    cout << "Executing Matrix Multiplcation" << endl;
    cout << "Matrix size: " << N << "x" << N << endl;

    // Allocate memory on the host
    float *hA,*hB,*hC;
    hA = new float[N*N];
    hB = new float[N*N];
    hC = new float[N*N];

    // Initialize matrices on the host
    for (int j=0; j<N; j++){
        for (int i=0; i<N; i++){
           hA[j*N+i] = 2;
           hB[j*N+i] = 4;
```

```
        }
    }

    // Allocate memory on the device
    int size = N*N*sizeof(float);    // Size of the memory in bytes
    float *dA,*dB,*dC;
    cudaMalloc(&dA,size);
    cudaMalloc(&dB,size);
    cudaMalloc(&dC,size);

    dim3 threadBlock(BLOCK_SIZE,BLOCK_SIZE);
    dim3 grid(K,K);
    cout<<"\nInput Matrix 1 \n";
    for (int row=0; row<N; row++){
            for (int col=0; col<N; col++){

                    cout<<hA[row*col]<<" ";

            }
            cout<<endl;
        }
    cout<<"\nInput Matrix 2 \n";
    for (int row=0; row<N; row++){
            for (int col=0; col<N; col++){

                    cout<<hB[row*col]<<" ";

            }
            cout<<endl;
        }
    // Copy matrices from the host to device
    cudaMemcpy(dA,hA,size,cudaMemcpyHostToDevice);
    cudaMemcpy(dB,hB,size,cudaMemcpyHostToDevice);

    //Execute the matrix multiplication kernel

    gpuMM<<<grid,threadBlock>>>(dA,dB,dC,N);

    // Now do the matrix multiplication on the CPU
   /*float sum;
    for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){
            sum = 0.f;
            for (int n=0; n<N; n++){
                sum += hA[row*N+n]*hB[n*N+col];
            }
            hC[row*N+col] = sum;
            cout << sum <<"     ";


        }
        cout<<endl;
    }*/

    // Allocate memory to store the GPU answer on the host
    float *C;
    C = new float[N*N];

    // Now copy the GPU result back to CPU
    cudaMemcpy(C,dC,size,cudaMemcpyDeviceToHost);

    // Check the result and make sure it is correct
    cout <<"\n\n\n\n\n Resultant matrix\n\n";
    for (int row=0; row<N; row++){
        for (int col=0; col<N; col++){
```

```
                cout<<C[row*col]<<"      ";

        }
        cout<<endl;
    }

    cout << "Finished." << endl;
}
```

```
Enter a Value for Size/2 of matrixExecuting Matrix Multiplcation
Matrix size: 4x4

Input Matrix 1
2       2       2       2
2       2       2       2
2       2       2       2
2       2       2       2

Input Matrix 2
4       4       4       4
4       4       4       4
4       4       4       4
4       4       4       4




 Resultant matrix

32      32      32      32
32      32      32      32
32      32      32      32
32      32      32      32
Finished.
```