!nvcc --version

!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git

%load\_ext nvcc\_plugin

from google.colab import drive

drive.mount('/content/drive')

#1 block 1 thread Assign3 Q2

%%cu

#include<stdio.h>

\_\_global\_\_ void myKernel(){

    printf("Hello World! from Kernel\n");

}

int main(){

    myKernel<<<1,1>>>();

    printf("Hello World! from main\n");

    return 0;

}

#multiple blocks 1 thread vector addition Assign3 Q3

%%cu

#define N 512

#include<iostream>

#include<cmath>

#include<fstream>

using namespace std;

\_\_global\_\_ void vecAdd(int \*a, int\* b, int\* c);

int main(){

    ifstream fin;

    fin.open("/content/drive/My Drive/CSV data/Database.txt", ios::in);

    int a[N], b[N], c[N];

    int \*dev\_a, \*dev\_b, \*dev\_c;

    for(int i = 0;i<N;i++){

        fin>>a[i];

        fin>>b[i];

    }

    int size = N\*sizeof(int);

    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);

    vecAdd<<<N,1>>>(dev\_a,dev\_b,dev\_c);

    cudaMemcpy(c, dev\_c, size,cudaMemcpyDeviceToHost);

    cout<<"\na[]\n";

    for(int i=0;i<N;i++){

        cout<<a[i]<<" ";

    }

    cout<<"\nb[]\n";

    for(int i=0;i<N;i++){

        cout<<b[i]<<" ";

    }

    cout<<"\nc[]\n";

    for(int i=0;i<N;i++){

        cout<<c[i]<<" ";

    }

    fin.close();

    cudaFree(dev\_a);

    cudaFree(dev\_b);

    cudaFree(dev\_c);

    exit (0);

}

\_\_global\_\_ void vecAdd (int \*a, int \*b, int \*c) {

    int i = blockIdx.x \* blockDim.x + threadIdx.x;

    if(i<N){

      c[i] = a[i] + b[i];

    }

}

#1 block multiple threads vector subtraction Assign3 Q4

%%cu

#define N 512

#include<iostream>

#include<cmath>

#include<fstream>

using namespace std;

\_\_global\_\_ void vecSub(int \*a, int\* b, int\* c);

int main(){

    ifstream fin;

    fin.open("/content/drive/My Drive/CSV data/Database.txt", ios::in);

    int a[N], b[N], c[N];

    int \*dev\_a, \*dev\_b, \*dev\_c;

    for(int i = 0;i<N;i++){

        fin>>a[i];

        fin>>b[i];

    }

    int size = N\*sizeof(int);

    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);

    vecSub<<<1,N>>>(dev\_a,dev\_b,dev\_c);

    cudaMemcpy(c, dev\_c, size,cudaMemcpyDeviceToHost);

    cout<<"\na[]\n";

    for(int i=0;i<N;i++){

        cout<<a[i]<<" ";

    }

    cout<<"\nb[]\n";

    for(int i=0;i<N;i++){

        cout<<b[i]<<" ";

    }

    cout<<"\nc[]\n";

    for(int i=0;i<N;i++){

        cout<<c[i]<<" ";

    }

    fin.close();

    cudaFree(dev\_a);

    cudaFree(dev\_b);

    cudaFree(dev\_c);

    exit (0);

}

\_\_global\_\_ void vecSub (int \*a, int \*b, int \*c) {

    int i = blockIdx.x \* blockDim.x + threadIdx.x;

    if(i<N){

      c[i] = a[i] - b[i];

    }

}

#multiple block multiple thread, multiply each element by 5 Assign4 Q1

%%cu

#define T 1024

#define N 10000

#include<iostream>

#include<cmath>

#include<fstream>

using namespace std;

\_\_global\_\_ void vecMul5(int \*a, int\* c);

int main(){

    ifstream fin;

    fin.open("/content/drive/My Drive/CSV data/Database.txt", ios::in);

    int a[N], c[N];

    int \*dev\_a, \*dev\_c;

    for(int i = 0;i<N;i++){

        fin>>a[i];

    }

    int size = N\*sizeof(int);

    cudaMalloc((void\*\*)&dev\_a, size);

    cudaMalloc((void\*\*)&dev\_c, size);

    cudaMemcpy(dev\_a, a, size, cudaMemcpyHostToDevice);

    vecMul5<<<((N+T-1)/T),T>>>(dev\_a,dev\_c);

    cudaMemcpy(c, dev\_c, size,cudaMemcpyDeviceToHost);

    cout<<"\na[]\n";

    for(int i=0;i<N;i++){

        cout<<a[i]<<" ";

    }

    cout<<"\nc[]\n";

    for(int i=0;i<N;i++){

        cout<<c[i]<<" ";

    }

    fin.close();

    cudaFree(dev\_a);

    cudaFree(dev\_c);

    exit (0);

}

\_\_global\_\_ void vecMul5 (int \*a, int \*c) {

    int i = blockIdx.x \* blockDim.x + threadIdx.x;

    if(i<N){

      c[i] = a[i]\*5;

    }

}

#multiple block multiple thread, vector Pairwise Sum Assign4 Q2

%%cu

#define T 512

#define N 10000

#define m -1

#include<iostream>

#include<cmath>

#include<fstream>

using namespace std;

\_\_global\_\_ void vecPairSum(int \*a, int\* c);

int main(){

    ifstream fin;

    fin.open("/content/drive/My Drive/CSV data/Database.txt", ios::in);

    int a[N], c[N];

    int \*dev\_a, \*dev\_c;

    for(int i = 0;i<N;i++){

        fin>>a[i];

    }

    int size = N\*sizeof(int);

    cudaMalloc((void\*\*)&dev\_a, size);

    cudaMalloc((void\*\*)&dev\_c, size);

    cudaMemcpy(dev\_a, a, size, cudaMemcpyHostToDevice);

    vecPairSum<<<((N+T-1)/T),T>>>(dev\_a,dev\_c);

    cudaMemcpy(c, dev\_c, size,cudaMemcpyDeviceToHost);

    cout<<"\na[]\n";

    for(int i=0;i<N;i++){

        cout<<a[i]<<" ";

    }

    cout<<"\nc[]\n";

    for(int i=1;i<N;i++){

        cout<<c[i]<<" ";

    }

    fin.close();

    cudaFree(dev\_a);

    cudaFree(dev\_c);

    exit (0);

}

\_\_global\_\_ void vecPairSum (int \*a, int \*c) {

    \_\_shared\_\_ int temp[T+1];

    int g = threadIdx.x + blockIdx.x \* blockDim.x;

    int l = threadIdx.x + 1;

    if( l < N ) {

        temp [ l ] = a [ g ];

        if (threadIdx.x < 1 ) {

            if(g > 0){

                temp[l+m] = a [g+m] ;

            }

            else{

                temp[l+m] = 0;

            }

        }

        \_\_syncthreads();

        int result = 0;

        for (int offset = -1 ; offset <= 0 ; offset++)

            result += temp[l+offset];

        c[g] = result;

    }

}

# 1 block multiple threads, Dot Product Assign4 Q3

%%cu

#define N 32

#define m -1

#include<iostream>

#include<cmath>

#include<fstream>

using namespace std;

\_\_global\_\_ void vecDotProduct(int \*a, int\* b, int\* c);

int main(){

    ifstream fin;

    fin.open("/content/drive/My Drive/CSV data/Database.txt", ios::in);

    int a[N], b[N];

    int \*c = (int \*)malloc(sizeof(int));

    int \*dev\_a, \*dev\_b, \*dev\_c;

    for(int i = 0;i<N;i++){

        fin>>a[i];

        fin>>b[i];

    }

    int size = N\*sizeof(int);

    cudaMalloc((void\*\*)&dev\_a, size);

    cudaMalloc((void\*\*)&dev\_b, size);

    cudaMalloc((void\*\*)&dev\_c, sizeof(int));

    cudaMemcpy(dev\_a, a, size, cudaMemcpyHostToDevice);

    cudaMemcpy(dev\_b, b, size,cudaMemcpyHostToDevice);

    vecDotProduct<<<1,N>>>(dev\_a,dev\_b,dev\_c);

    cudaMemcpy(c, dev\_c, sizeof(int),cudaMemcpyDeviceToHost);

    cout<<"\na[]\n";

    for(int i=0;i<N;i++){

        cout<<a[i]<<" ";

    }

    cout<<"\nb[]\n";

    for(int i=0;i<N;i++){

        cout<<b[i]<<" ";

    }

    cout<<"\nDot Product is: "<<c[0];

    fin.close();

    free(c);

    cudaFree(dev\_a);

    cudaFree(dev\_b);

    cudaFree(dev\_c);

    exit (0);

}

\_\_global\_\_ void vecDotProduct (int \*a, int \*b, int \*c) {

    \_\_shared\_\_ int temp[N];

    int i = blockIdx.x \* blockDim.x + threadIdx.x;

    if(i<N){

      temp[i] = (a[i] \* b[i]);

      \_\_syncthreads();

      if(i==(N+m)){

          int s = 0;

          for(int i=0;i<N;i++){

              s = s + temp[i];

          }

          \*c = s;

      }

    }

}