Assignment 4

**Add large vectors**

#include<iostream>  
#include<bits/stdc++.h>  
#include<cuda.h>  
#define BLOCK\_SIZE 16  
using namespace std;  
  
void fill\_array(int \*arr,int size){  
for(int i = 0;i < size; i++){  
arr[i] = rand() % 100;  
}  
}  
  
void add\_cpu(int \*arr1, int \*arr2, int \*result, int size){  
for(int i = 0;i < size; i++){  
result[i] = arr1[i] + arr2[i];  
}  
}  
  
void print\_matrix(int \*arr, int size){  
for(int i = 0; i < size; i++){  
cout << arr[i] << " ";  
}  
cout << endl;  
}  
  
\_\_global\_\_ void add(int \*arr1, int \*arr2, int \*arr3,int size){  
int block\_id = blockIdx.x \* blockDim.x + threadIdx.x;  
if(block\_id < size){  
arr3[block\_id] = arr1[block\_id] + arr2[block\_id];  
}  
}  
  
int main(){  
int \*arr1\_cpu,\*arr2\_cpu,\*result\_cpu;  
int size;  
cout << "Enter size of vector: ";  
cin >> size;  
  
arr1\_cpu = new int[size];  
arr2\_cpu = new int[size];  
result\_cpu = new int[size];  
  
fill\_array(arr1\_cpu,size);  
cout << "Array 1: ";  
print\_matrix(arr1\_cpu,size);  
fill\_array(arr2\_cpu,size);  
cout << "Array 2: ";  
print\_matrix(arr2\_cpu,size);  
  
int \*arr1\_gpu,\*arr2\_gpu,\*result\_gpu;  
  
cudaMallocManaged(&arr1\_gpu, size \* sizeof(int));  
cudaMallocManaged(&arr2\_gpu, size \* sizeof(int));  
cudaMallocManaged(&result\_gpu, size \* sizeof(int));  
  
cudaMemcpy(arr1\_gpu,arr1\_cpu,size \* sizeof(int),cudaMemcpyHostToDevice);  
cudaMemcpy(arr2\_gpu,arr2\_cpu,size \* sizeof(int),cudaMemcpyHostToDevice);  
cudaEvent\_t start,stop;  
float elapsedTime;  
  
dim3 dimGrid(size + BLOCK\_SIZE - 1 / BLOCK\_SIZE);  
dim3 dimBlock(BLOCK\_SIZE);  
  
cudaEventCreate(&start);  
cudaEventCreate(&stop);  
cudaEventRecord(start,0);  
  
add<<<dimGrid,dimBlock>>>(arr1\_gpu,arr2\_gpu,result\_gpu,size);  
cudaEventRecord(stop,0);  
cudaEventSynchronize(stop);  
cudaEventElapsedTime(&elapsedTime,start,stop);  
cudaEventDestroy(start);  
cudaEventDestroy(stop);  
cudaMemcpy(result\_cpu,result\_gpu,size \* sizeof(int),cudaMemcpyDeviceToHost);  
cout << "GPU result:\n";  
print\_matrix(result\_cpu,size);  
cout<<"Elapsed Time = "<<elapsedTime<<" milliseconds" << endl;  
cudaFree(arr1\_gpu);  
cudaFree(arr2\_gpu);  
cudaFree(result\_gpu);  
  
cudaEventCreate(&start);  
cudaEventCreate(&stop);  
cudaEventRecord(start,0);  
  
add\_cpu(arr1\_cpu,arr2\_cpu,result\_cpu,size);  
cudaEventRecord(stop,0);  
cudaEventSynchronize(stop);  
cudaEventElapsedTime(&elapsedTime,start,stop);  
cudaEventDestroy(start);  
cudaEventDestroy(stop);  
cout << "CPU result:\n";  
print\_matrix(result\_cpu,size);  
cout<<"Elapsed Time = "<<elapsedTime<<" milliseconds" << endl;  
  
return 0;  
}

**Matrix Mutiplication**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define row1 2 /\* Number of rows of first matrix \*/

#define col1 3 /\* Number of columns of first matrix \*/

#define row2 3 /\* Number of rows of second matrix \*/

#define col2 2 /\* Number of columns of second matrix \*/

\_\_global\_\_ void matproductsharedmemory(int \*l,int \*m, int \*n)

{

int x=blockIdx.x;

int y=blockIdx.y;

\_\_shared\_\_ int p[col1];

int i;

int k=threadIdx.x;

n[col2\*y+x]=0;

p[k]=l[col1\*y+k]\*m[col2\*k+x];

\_\_syncthreads();

for(i=0;i<col1;i++)

n[col2\*y+x]=n[col2\*y+x]+p[i];

}

int main()

{

int a[row1][col1];

int b[row2][col2];

int c[row1][col2];

int \*d,\*e,\*f;

int i,j;

// printf("\n Enter elements of first matrix of size 2\*3\n");

for(i=0;i<row1;i++)

{

for(j=0;j<col1;j++)

{

a[i][j] = i+j+1;

printf("%d\t",a[i][j]);

}

printf("\n");

}

// printf("\n Enter elements of second matrix of size 3\*2\n");

for(i=0;i<row2;i++)

{

for(j=0;j<col2;j++)

{

b[i][j] = j+2;

printf("%d\t",b[i][j]);

}

printf("\n");

}

cudaMalloc((void \*\*)&d,row1\*col1\*sizeof(int));

cudaMalloc((void \*\*)&e,row2\*col2\*sizeof(int));

cudaMalloc((void \*\*)&f,row1\*col2\*sizeof(int));

cudaMemcpy(d,a,row1\*col1\*sizeof(int),cudaMemcpyHostToDevice);

cudaMemcpy(e,b,row2\*col2\*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);

/\* Here we are defining two dimensional Grid(collection of blocks) structure. Syntax is dim3 grid(no. of columns,no. of rows) \*/

matproductsharedmemory<<<grid,col1>>>(d,e,f);

cudaMemcpy(c,f,row1\*col2\*sizeof(int),cudaMemcpyDeviceToHost);

printf("\n Product of two matrices:\n ");

for(i=0;i<row1;i++)

{

for(j=0;j<col2;j++)

{

printf("%d\t",c[i][j]);

}

printf("\n");

}

cudaFree(d);

cudaFree(e);

cudaFree(f);

return 0;

}

// nvcc filename -o objFileName

// ./objFileName