VECTOR ADDITION

```
!nvcc --version

code = """
#include<iostream>
#include<cstdlib>
using namespace std;

__global__ void vectorAdd(int *a, int *b, int *result, int n) {
int tid = blockIdx.x*blockDim.x + threadIdx.x;
if(tid <= n) {
result[tid] = a[tid] + b[tid];
}
}

void print_array(int *a, int N) {
for(int i=0; i<N; i++) {
cout<<" "<<a[i];
}
cout<<endl;
}

void init_array(int *a, int N) {
for(int i=0; i<N; i++) {
a[i] = rand()%10 + 1;
}
}

int main() {
int *a, *b, *c;
int *a_dev, *b_dev, *c_dev;
int n = 64; //24
a = (int*)malloc(n * sizeof(n));
b = (int*)malloc(n * sizeof(n));
c = (int*)malloc(n * sizeof(n));

int size = n * sizeof(int);
cudaMalloc(&a_dev, size);
cudaMalloc(&b_dev, size);
cudaMalloc(&c_dev, size);
init_array(a, n);
init_array(b, n);
print_array(a, n);
print_array(b, n);
//cudaEvent_t start, end;
//cudaEventCreate(&start);
//cudaEventCreate(&end);
cudaMemcpy(a_dev, a, size, cudaMemcpyHostToDevice);
cudaMemcpy(b_dev, b, size, cudaMemcpyHostToDevice);
//int threads = 1024;
//int blocks = (n+threads-1)/threads;
//cudaEventRecord(start);
//vectorAdd<<<blocks,threads>>>(a_dev, b_dev, c_dev, n);
vectorAdd<<<1,1024>>>(a_dev, b_dev, c_dev, n);
//cudaEventRecord(end);
//cudaDeviceSynchronize();
//float time = 0.0;
//cudaEventElapsedTime(&time, start, end);
cudaMemcpy(c, c_dev, size, cudaMemcpyDeviceToHost);
cout<<"Results : "<<endl;
```

```
print_array(c, n);
//cout<<"Time elapsed : "<<time<<endl;
cudaFree(a_dev);
cudaFree(b_dev);
cudaFree(c_dev);
return 0;
}
"""

text_file = open("ass2.cu", "w")
text_file.write(code)
text_file.close()
!nvcc ass2.cu
!./a.out
!nvprof ./a.out
```

OUTPUT::

```
Vector1-
4 7 8 6 4 6 7 3 10 2 3 8 1 10 4 7 1 7 3 7 2 9 8 10 3 1 3 4 8 6 10 3 3 9 10 8 4 7 2 3 10 4 2 10
5 8 9 5 6 1 4 7 2 1 7 4 3 1 7 2 6 6 5 8
vector2-
7 6 7 10 4 8 5 6 3 6 5 8 5 5 4 1 8 9 7 9 9 5 4 2 5 10 3 1 7 9 10 3 7 7 5 10 6 1 5 9 8 2 8 3 8 3
3 7 2 1 7 2 6 10 5 10 1 10 2 8 8 2 2 6

Results : 11 13 15 16 8 14 12 9 13 8 8 16 6 15 8 8 9 16 10 16 11 14 12 12 8 11 6 5 15 15 20 6
10 16 15 18 10 8 7 12 18 6 10 13 13 11 12 12 8 2 11 9 8 11 12 14 4 11 9 10 14 8 7 14
```

VECTOR MATRIX MULTIPLICATION:

```
!nvcc --version


code = """#include<iostream>

using namespace std;

__global__
void matrixVector(int *vec, int *mat, int *result, int n, int m)
{
int tid = blockIdx.x*blockDim.x + threadIdx.x;
int sum=0;
if(tid <= n) {
for(int i=0; i<n; i++) {
sum += vec[i]*mat[(i*m) + tid];
}
result[tid] = sum;
}
}

void init_array(int *a, int n) {
for(int i=0; i<n; i++)
a[i] = rand()%n + 1;
}

void init_matrix(int *a, int n, int m) {
for(int i=0; i<n; i++) {
for(int j=0; j<m; j++) {
a[i*m + j] = rand()%n + 1;
}
```

```
}
}

void print_array(int *a, int n) {
for(int i=0; i<n; i++) {
cout<<" "<<a[i];
}
cout<<endl;
}

void print_matrix(int *a, int n, int m) {
for(int i=0; i<n; i++) {
for(int j=0; j<m; j++)
cout<<" "<<a[i*m + j];
cout<<endl;
}
}

int main() {
int *a, *b, *c;
int *a_dev, *b_dev, *c_dev;
int n = 16;
int m = 32;
a = new int[n];
b = new int[n*m];
c = new int[m];
init_array(a, n);
init_matrix(b, n, m);
cout<<"Initial array : "<<endl;
print_array(a, n);
cout<<"Initial matrix : "<<endl;
print_matrix(b, n, m);
cout<<"Initial resultant array : "<<endl;
print_array(c, m);
cout<<endl;
cudaMalloc(&a_dev, sizeof(int)*n);
cudaMalloc(&b_dev, sizeof(int)*n*m);
cudaMalloc(&c_dev, sizeof(int)*m);
cudaMemcpy(a_dev, a, sizeof(int)*n, cudaMemcpyHostToDevice);
cudaMemcpy(b_dev, b, sizeof(int)*n*m, cudaMemcpyHostToDevice);
matrixVector<<<m/256+1, 256>>>(a_dev, b_dev, c_dev, n, m);
cudaMemcpy(c, c_dev, sizeof(int)*m, cudaMemcpyDeviceToHost);
cout<<"Results : "<<endl;
print_array(c, m);
cudaFree(a_dev);
cudaFree(b_dev);
cudaFree(c_dev);
delete[] a;
delete[] b;
delete[] c;
return 0;
}


"""

text_file = open("ass2.cu", "w")
text_file.write(code)
text_file.close()
!nvcc ass2.cu
!./a.out
!nvprof ./a.out
```

OUTPUT::

```
Initial array : 8 7 10 4 2 16 11 13 10 14 11 12 3 12 4 7
Initial matrix : 13 3 5 9 12 9 8 14 7 11 15 4 4 16 10 11 7 3 14 8 2 9 4 11 6 14 6 8 9 10 15 5
12 3 14 7 12 5 5 2 15 3 5 2 2 14 13 8 1 10 15 2 2 2 13 8 15 2 15 8 11 13 12 7 16 9 13 11 13 1
12 12 3 16 13 5 13 9 12 13 3 11 15 4 12 11 11 11 12 10 2 7 6 13 13 5 6 9 15 2 10 10 13 12 9 10
16 6 2 12 2 4 6 16 8 1 10 2 11 6 11 12 12 1 9 8 5 14 16 4 15 9 13 12 5 6 5 4 11 6 15 12 10 4 12
1 5 5 2 15 10 13 11 5 13 3 12 1 16 12 4 14 4 1 9 8 6 13 12 16 3 10 11 12 14 6 12 2 11 13 16 4 9
10 9 5 12 4 6 11 15 9 9 3 9 1 10 14 14 5 13 16 15 8 11 12 13 6 13 7 2 12 11 11 6 3 15 1 6 4 12
5 13 4 7 5 4 16 3 1 5 15 16 3 6 10 14 3 15 10 9 1 5 3 11 10 5 9 11 11 13 6 15 9 9 5 13 12 4 15
13 8 14 12 10 3 6 7 5 4 16 14 4 5 16 14 14 5 7 8 15 3 13 13 11 5 1 7
Initial resultant array : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Results : 1308 1005 1197 1044 1592 1366 1370 1280 1143 988 1663 1122 1334 1216 1666 1013 1338 0
```

MATRIX MULTIPLICATION

```
!nvcc --version


code = """
#include<iostream>

using namespace std;

__global__
void matrixMultiplication(int *a, int *b, int *c, int m, int n, int k)
{
int row = blockIdx.y*blockDim.y + threadIdx.y;
int col = blockIdx.x*blockDim.x + threadIdx.x;
int sum=0;
if(col<k && row<m) {
for(int j=0;j<n;j++)
{
sum += a[row*n+j] * b[j*k+col];
}
c[k*row+col]=sum;
}
}

void init_result(int *a, int m, int k) {
for(int i=0; i<m; i++) {
for(int j=0; j<k; j++) {
a[i*k + j] = 0;
}
}
}

void init_matrix(int *a, int n, int m) {
for(int i=0; i<n; i++) {
for(int j=0; j<m; j++) {
a[i*m + j] = rand()%10 + 1;
}
}
}

void print_matrix(int *a, int n, int m) {
for(int i=0; i<n; i++) {
for(int j=0; j<m; j++) {
cout<<" "<<a[i*m + j];
}
```

```
cout<<endl;
}
cout<<endl;
}

int main()
{
int *a,*b,*c;
int *a_dev,*b_dev,*c_dev;
int m=5, n=4, k=3;
a = new int[m*n];
b = new int[n*k];
c = new int[m*k];
init_matrix(a, m, n);
init_matrix(b, n ,k);
init_result(c, m, k);
cout<<"Initial matrix : "<<endl;
print_matrix(a, m, n);
print_matrix(b, n, k);
print_matrix(c, m, k);
cudaMalloc(&a_dev, sizeof(int)*m*n);
cudaMalloc(&b_dev, sizeof(int)*n*k);
cudaMalloc(&c_dev, sizeof(int)*m*k);
cudaMemcpy(a_dev, a, sizeof(int)*m*n, cudaMemcpyHostToDevice);
cudaMemcpy(b_dev, b, sizeof(int)*n*k, cudaMemcpyHostToDevice);
dim3 dimGrid(1,1);
dim3 dimBlock(16,16);
matrixMultiplication<<<dimGrid, dimBlock>>>(a_dev,b_dev,c_dev, m, n, k);
cudaMemcpy(c, c_dev, sizeof(int)*m*k, cudaMemcpyDeviceToHost);
cout<<"Result : "<<endl;
print_matrix(c, m, k);
cudaFree(a_dev);
cudaFree(b_dev);
cudaFree(c_dev);
delete[] a;
delete[] b;
delete[] c;
return 0;
}
"""


text_file = open("ass2.cu", "w")
text_file.write(code)
text_file.close()
!nvcc ass2.cu
!./a.out
!nvprof ./a.out
```