**CUDA PROGRAMMING – PARALLEL AND DISTRIBUTED COMPUTING**

**(PCS – 106)**



**SUBMITTED TO: SUBMITTED BY:**

**Ms RUHI SAXENA ARUN KUSHWAHA**

**CSE Deptt. 801732005**

**ME(CS1)**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **SNO** | **EXERCISES** | **PAGE NO.** | **REMARKS** |
| 1 | CUDA Program to print “Hello World”. | 1 |  |
| 2 | CUDA Program for matrix addition. | 2 |  |
| 3 | CUDA Program for matrix multiplication. | 5 |  |
| 5 | . CUDA Program for even-odd sorting | 8 |  |
| 4 | . CUDA Program for vector addition | 11 |  |
| 6 | CUDA Program to calculate execution time by increasing  blocks. | 13 |  |
| 7 | CUDA Program to calculate execution time by increasing threads. | 16 |  |
| 8 | CUDA Program for swapping two numbers. | 19 |  |
| 9 | CUDA Program for Bitonic Sorting. | 21 |  |
| 10 | CUDA Program for vector addition through sine function. | 25 |  |
| 11 | CUDA Program for vector addition through cosine  function. | 27 |  |
| 12 | CUDA Program for matrix addition through random  function. | 29 |  |

# PROGRAM 1

**Write a simple program in cuda programming**

#include<stdio.h>

\_\_global\_\_ void Kernel(void)

{

}

int main(void)

{

Kernel<<<1,1>>>();

printf("Hello World/\n");

return 0;

}

**Output**

****

**PROGRAM 2**

**Write a program of matrix addition in cuda programming**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<math.h>

\_\_global\_\_ void addMat(int a[],int b[],int c[],int m,int n)

{

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

if(blockIdx.x<m && threadIdx.x<n) {

c[index]=a[index]+b[index];

}}

void readMat(int a[],int r,int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

scanf("%d",&a[i\*c+j]);

}}}

void printMat(int a[],int r,int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

printf("%d ",a[i\*c+j]);

}}}

int main()

{

int r=3,c=3;

int \*ha,\*hb,\*hc,\*da,\*db,\*dc;

size\_t size;

size=r\*c\*sizeof(int);

ha=(int\*) malloc(size);

hb=(int\*) malloc(size);

hc=(int\*) malloc(size);

printf("Matrix A (3\*3): ");

readMat(ha,r,c);

printf("Matrix B (3\*3): ");

readMat(hb,r,c);

cudaMalloc(&da,size);

cudaMalloc(&db,size);

cudaMalloc(&dc,size);

cudaMemcpy(da,ha,size,cudaMemcpyHostToDevice);

cudaMemcpy(db,hb,size,cudaMemcpyHostToDevice);

addMat<<<r,c>>>(da,db,dc,r,c);

cudaMemcpy(hc,dc,size,cudaMemcpyDeviceToHost);

printf("Matrix after addition Matrix C: ");

printMat(hc,r,c);

cudaFree(da);

cudaFree(db);

cudaFree(dc);

free(ha);

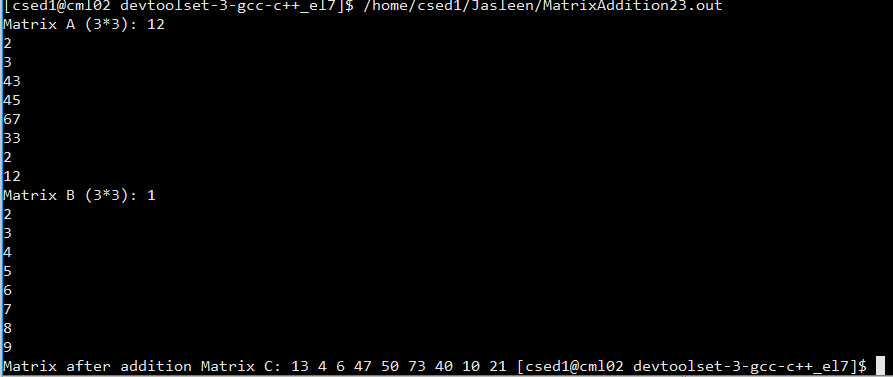
free(hb);

free(hc);

return 0;

}

**Output**

****

**PROGRAM 3**

**Write a program of matrix multiplication in cuda programming**

#include<cuda.h>

#include<stdio.h>

\_\_global\_\_ void MatrixMulKernel(int \*Md, int \*Nd, int \*Pd, int Width) {

int tId = threadIdx.x;

int bId = blockIdx.x;

int Pvalue = 0;

for(int k = 0; k < Width ; ++k) {

int Mdelement = Md[bId\*Width + k];

int Ndelement = Nd[k\*Width + tId];

Pvalue += (Mdelement\*Ndelement);

}

Pd[bId\*Width + tId] = Pvalue;

}

void readMat(int a[], int Width)

{

for(int i=0;i<Width;i++)

{

for( int j=0;j<Width;j++)

{

scanf("%d", &a[i\*Width+j]);

}}}

void printMat(int a[], int Width)

{

for(int i=0;i<Width;i++)

{

for(int j=0;j<Width;j++)

{

printf("%d ", a[i\*Width+j]);

}}}

int main() {

void MatrixMultiplication(int \*, int \*, int \*, int);

const int Width = 3;

int M[Width\*Width], N[Width\*Width], P[Width\*Width];

printf("\n Enter a 3\*3 Matrix M \n");

readMat(M, Width);

printf("\n Enter a 3\*3 Matrix N \n");

readMat(N, Width);

MatrixMultiplication(M, N, P, Width);

printf("\n Matrix M ");

printMat(M,Width);

printf("\n Matrix N ");

printMat(N,Width);

printf("\n Matrix P ");

printMat(P,Width);

return 0;

}

void MatrixMultiplication(int \*M, int \*N, int \*P, int Width) {

int size = Width\*Width\*sizeof(int);

int \*Md, \*Nd, \*Pd;

cudaMalloc((void\*\*)&Md, size);

cudaMemcpy(Md,M,size,cudaMemcpyHostToDevice);

cudaMalloc((void\*\*)&Nd, size);

cudaMemcpy(Nd,N,size,cudaMemcpyHostToDevice);

cudaMalloc((void\*\*)&Pd,size);

MatrixMulKernel<<<Width,Width>>>(Md,Nd,Pd,Width);

cudaMemcpy(P,Pd,size,cudaMemcpyDeviceToHost);

cudaFree(Md);

cudaFree(Nd);

cudaFree(Pd);

}

**Output**

****

# Program 4

**Write a program of odd even sorting using cuda programming**

#include<stdio.h>

#include<cuda.h>

\_\_global\_\_ void even(int \*darr, int n )

{

int k=threadIdx.x;

int t;

k=k\*2;

if(k<=n-2)

{

if(darr[k]> darr[k+1])

{

t=darr[k];

darr[k]=darr[k+1];

darr[k+1]=t;

}

}

}

\_\_global\_\_ void odd(int \*darr, int n )

{

int k = threadIdx.x;

int t;

k=k\*2+1;

if(k<=n-2)

{

if(darr[k]> darr[k+1])

{

t=darr[k];

darr[k]=darr[k+1];

darr[k+1]=t;

}

}

}

int main()

{

int \*arr, \*darr;

int n,i;

printf("how many numbers to enter:");

scanf("%d", &n);

arr=(int\*)malloc(n\*sizeof(int));

printf("Enter the nos");

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

{

scanf("%d", &arr[i]);

}

cudaMalloc(&darr, n\*sizeof(int));

cudaMemcpy(darr, arr, n\*sizeof(int), cudaMemcpyHostToDevice);

for(i=0;i<=n/2;i++)

{

even <<<1,n>>> (darr,n);

odd <<<1,n>>> (darr,n);

}

cudaMemcpy( arr, darr, n\*sizeof(int), cudaMemcpyDeviceToHost);

printf("Sorted Array");

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

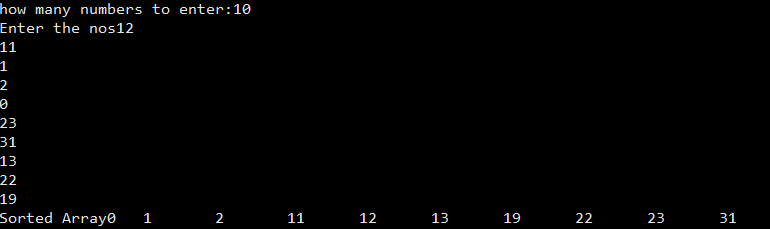
{

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

}

return 0;

}

**Output:**

# Program 5

**Write a program of vector addition in cuda**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<cuda\_runtime.h>

#define N 10

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

{

int bid=blockIdx.x;

if(bid<N)

c[bid]=a[bid]+b[bid];

}

int main(void)

{

int i,a[N],b[N],c[N];

int \*dev\_a,\*dev\_b,\*dev\_c;

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

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

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

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

{

a[i]=-(i\*i);

b[i]=(i\*i\*i);

}

cudaMemcpy(dev\_a,a,N\*sizeof(int),cudaMemcpyHostToDevice);

cudaMemcpy(dev\_b,b,N\*sizeof(int),cudaMemcpyHostToDevice);

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

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

printf("\n a+b=c \n");

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

{

printf("%d+%d=%d\n",a[i],b[i],c[i]);

}

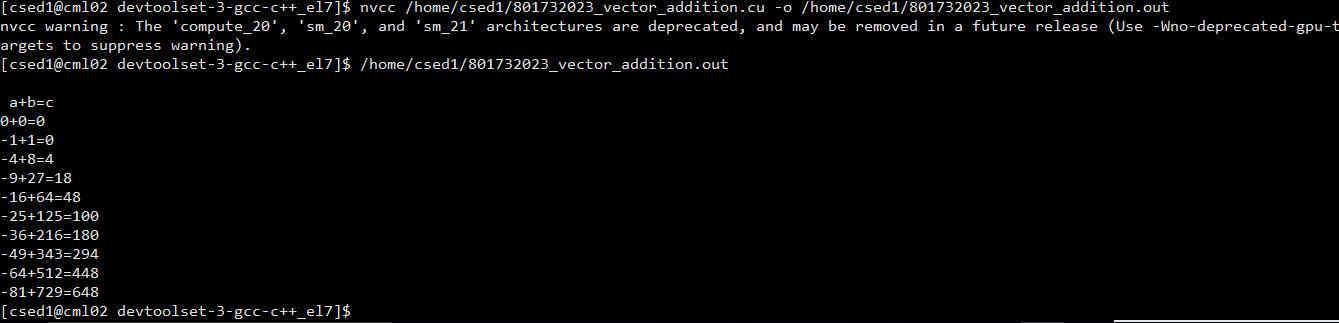
cudaFree(dev\_a);

cudaFree(dev\_b);

cudaFree(dev\_c);

}

**Output:**



**Program 6**

**WAP to calculate execution time by increasing blocks**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<math.h>

#include <time.h>

\_\_global\_\_ void addMat(int a[],int b[],int c[],int m,int n)

{

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

if(blockIdx.x<m && threadIdx.x<n) {

c[index]=a[index]+b[index];

}

}

void readMat(int a[],int r,int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

scanf("%d",&a[i\*c+j]);

}

}

}

void printMat(int a[],int r,int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

printf("%d ",a[i\*c+j]);

}

}

printf("\n");

}

int main()

{

int r=3,c=3;

int \*ha,\*hb,\*hc,\*da,\*db,\*dc;

size\_t size;

size=r\*c\*sizeof(int);

ha=(int\*) malloc(size);

hb=(int\*) malloc(size);

hc=(int\*) malloc(size);

printf("Matrix A: ");

readMat(ha,r,c);

printf("Matrix B: ");

readMat(hb,r,c);

cudaMalloc(&da,size);

cudaMalloc(&db,size);

cudaMalloc(&dc,size);

cudaMemcpy(da,ha,size,cudaMemcpyHostToDevice);

cudaMemcpy(db,hb,size,cudaMemcpyHostToDevice);

clock\_t tic = clock();

addMat<<<9,1>>>(da,db,dc,r,c);

clock\_t toc = clock();

cudaMemcpy(hc,dc,size,cudaMemcpyDeviceToHost);

printf("Matrix after addition Matrix C: ");

printMat(hc,r,c);

cudaFree(da);

cudaFree(db);

cudaFree(dc);

printf("Elapsed: %f seconds\n", (double)(toc - tic) / CLOCKS\_PER\_SEC);

free(ha);

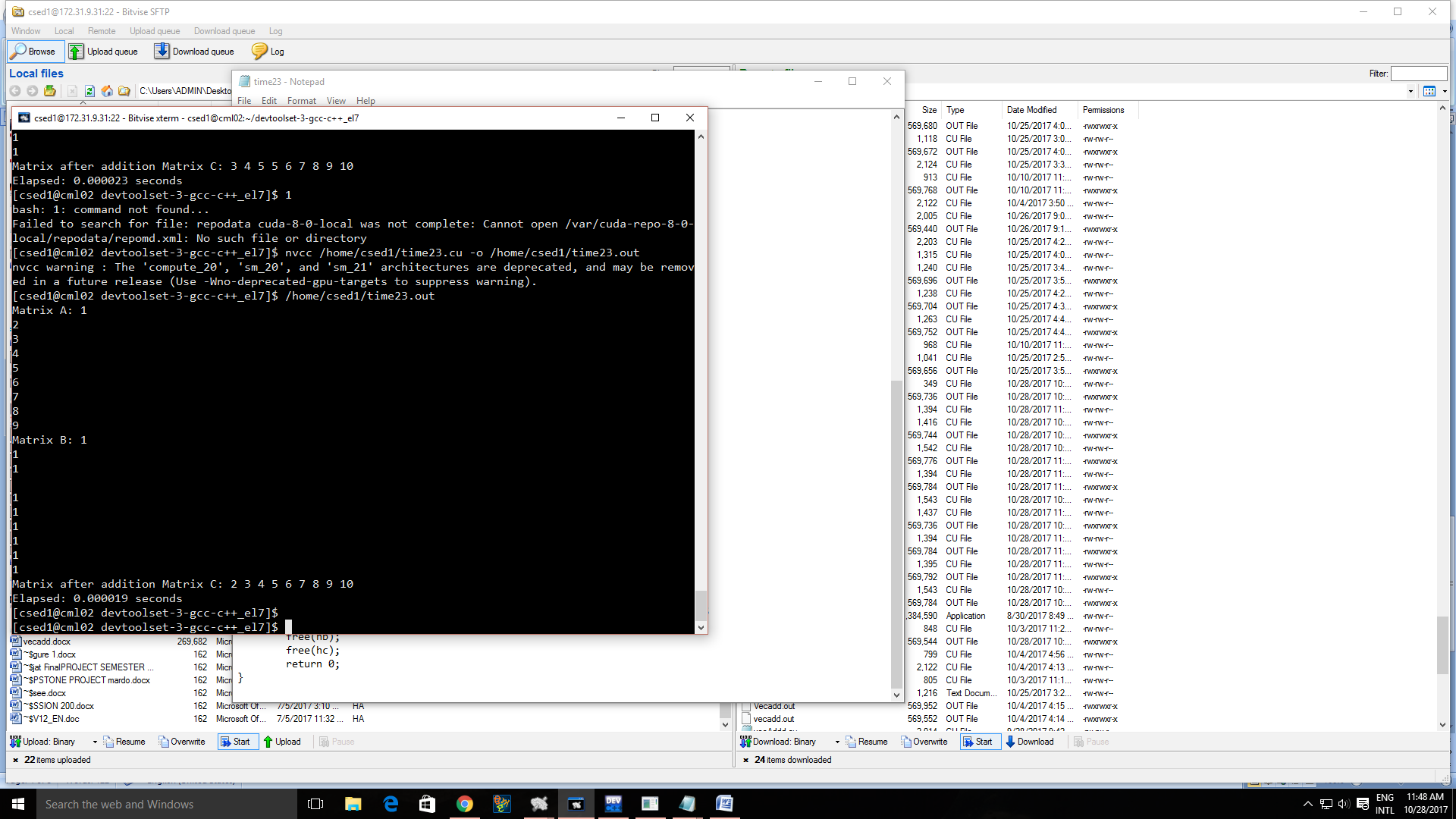
free(hb);

free(hc);

return 0;

}

**Output:**



# Program 7

**WAP to calculate CPU execution time by increasing threads**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<math.h>

#include <time.h>

\_\_global\_\_ void addMat(int a[],int b[],int c[],int m,int n)

{

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

if(blockIdx.x<m && threadIdx.x<n) {

c[index]=a[index]+b[index];

}

}

void readMat(int a[],int r,int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

scanf("%d",&a[i\*c+j]);

}

}

}

void printMat(int a[],int r,int c)

{

for(int i=0;i<r;i++)

{

for(int j=0;j<c;j++)

{

printf("%d ",a[i\*c+j]);

}

}

printf("\n");

}

int main()

{

int r=3,c=3;

int \*ha,\*hb,\*hc,\*da,\*db,\*dc;

size\_t size;

size=r\*c\*sizeof(int);

ha=(int\*) malloc(size);

hb=(int\*) malloc(size);

hc=(int\*) malloc(size);

printf("Matrix A: ");

readMat(ha,r,c);

printf("Matrix B: ");

readMat(hb,r,c);

cudaMalloc(&da,size);

cudaMalloc(&db,size);

cudaMalloc(&dc,size);

cudaMemcpy(da,ha,size,cudaMemcpyHostToDevice);

cudaMemcpy(db,hb,size,cudaMemcpyHostToDevice);

clock\_t tic = clock();

addMat<<<1,9>>>(da,db,dc,r,c);

clock\_t toc = clock();

cudaMemcpy(hc,dc,size,cudaMemcpyDeviceToHost);

printf("Matrix after addition Matrix C: ");

printMat(hc,r,c);

cudaFree(da);

cudaFree(db);

cudaFree(dc);

printf("Elapsed: %f seconds\n", (double)(toc - tic) / CLOCKS\_PER\_SEC);

free(ha);

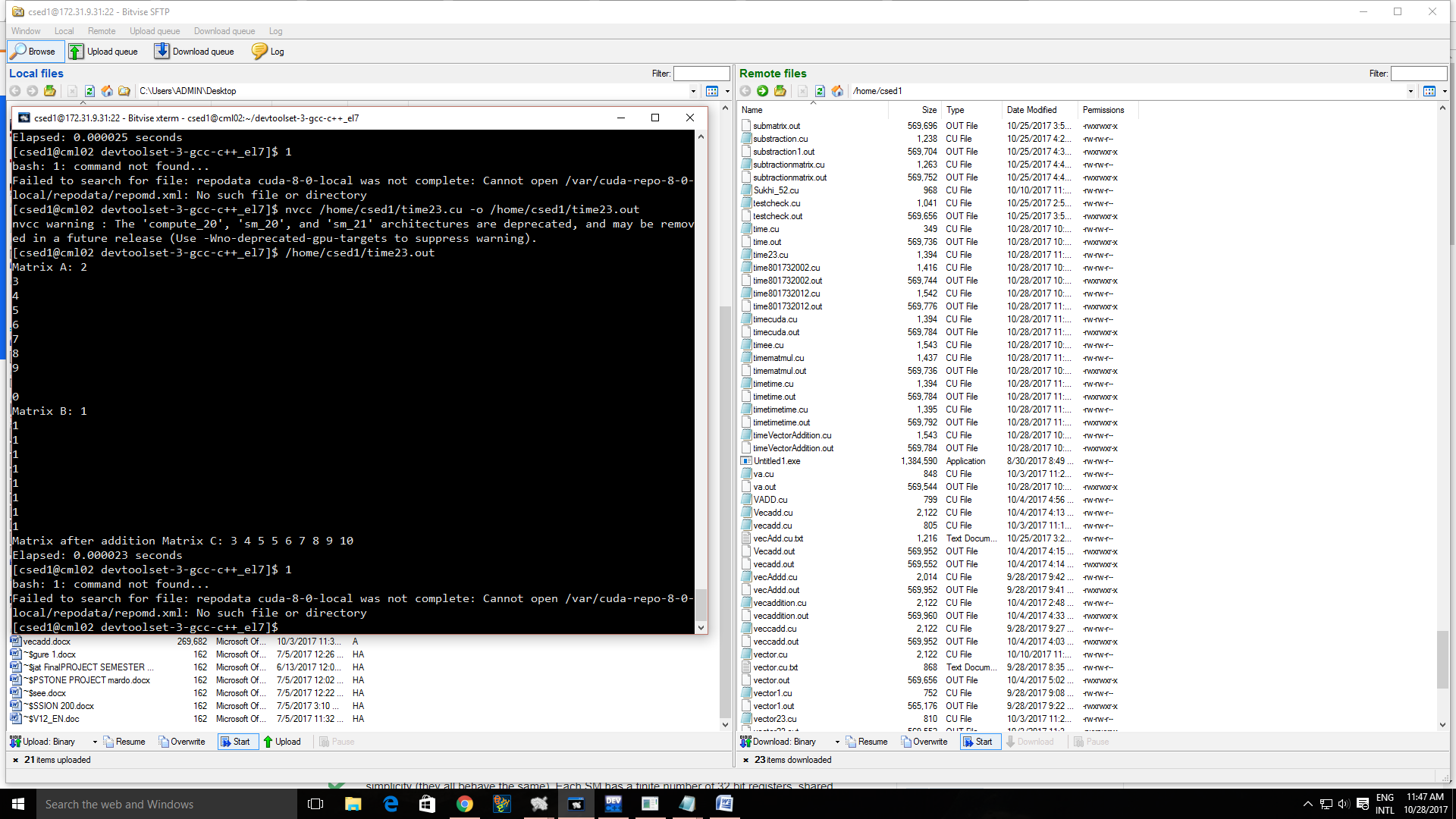
free(hb);

free(hc);

return 0;

}

**Output:**



# Program 8

**WAP to swap two numbers using cuda**

#include<stdio.h>

#include<stdlib.h>

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

{

\*c=\*a;

\*a=\*b;

\*b=\*c;

}

int main(void)

{

int a,b,c;

int \* dev\_a,\*dev\_b,\*dev\_c;

int size=sizeof(int);

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

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

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

a=2;

b=7;

printf("Values before Swaping\n");

printf("%d is the value of a\n",a);

printf("%d is the value of b\n",b);

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

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

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

cudaMemcpy(&a,dev\_a,size,cudaMemcpyDeviceToHost);

cudaMemcpy(&b,dev\_b,size,cudaMemcpyDeviceToHost);

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

cudaFree(dev\_a);

cudaFree(dev\_b);

cudaFree(dev\_c);

printf("Values after Swaping\n");

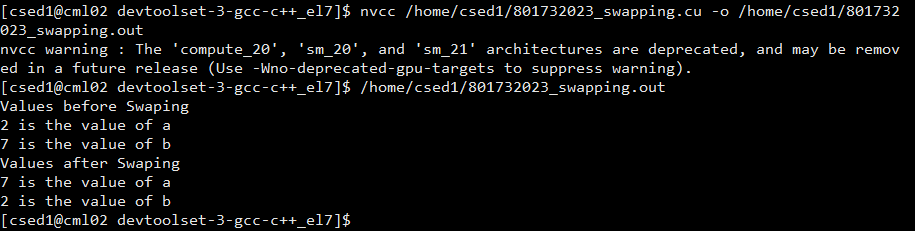
printf("%d is the value of a\n",a);

printf("%d is the value of b\n",b);

return 0;

}

**Output:**

****

**Program 9**

**WAP to implement bitonic sort in cuda programming**

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

/\* Every thread gets exactly one value in the unsorted array. \*/

#define THREADS 2

#define BLOCKS 4

#define NUM\_VALS THREADS\*BLOCKS

void print\_elapsed(clock\_t start, clock\_t stop)

{

double elapsed = ((double) (stop - start)) / CLOCKS\_PER\_SEC;

printf("Elapsed time: %.3fs\n", elapsed);

}

float random\_float()

{

return (float)rand()/(float)RAND\_MAX;

}

void array\_print(float \*arr, int length)

{

int i;

for (i = 0; i < length; ++i) {

printf("%1.3f ", arr[i]);

}

printf("\n");

}

void array\_fill(float \*arr, int length)

{

srand(time(NULL));

int i;

for (i = 0; i < length; ++i) {

arr[i] = random\_float();

}

}

\_\_global\_\_ void bitonic\_sort\_step(float \*dev\_values, int j, int k)

{

unsigned int i, ixj; /\* Sorting partners: i and ixj \*/

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

ixj = i^j;

/\* The threads with the lowest ids sort the array. \*/

if ((ixj)>i) {

if ((i&k)==0) {

/\* Sort ascending \*/

if (dev\_values[i]>dev\_values[ixj]) {

/\* exchange(i,ixj); \*/

float temp = dev\_values[i];

dev\_values[i] = dev\_values[ixj];

dev\_values[ixj] = temp;

}

}

if ((i&k)!=0) {

/\* Sort descending \*/

if (dev\_values[i]<dev\_values[ixj]) {

/\* exchange(i,ixj); \*/

float temp = dev\_values[i];

dev\_values[i] = dev\_values[ixj];

dev\_values[ixj] = temp;

}

}

}

}

/\*\*

\* Inplace bitonic sort using CUDA.

\*/

void bitonic\_sort(float \*values)

{

float \*dev\_values;

size\_t size = NUM\_VALS \* sizeof(float);

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

cudaMemcpy(dev\_values, values, size, cudaMemcpyHostToDevice);

dim3 blocks(BLOCKS,1); /\* Number of blocks \*/

dim3 threads(THREADS,1); /\* Number of threads \*/

int j, k;

/\* Major step \*/

for (k = 2; k <= NUM\_VALS; k <<= 1) {

/\* Minor step \*/

for (j=k>>1; j>0; j=j>>1) {

bitonic\_sort\_step<<<blocks, threads>>>(dev\_values, j, k);

}

}

cudaMemcpy(values, dev\_values, size, cudaMemcpyDeviceToHost);

cudaFree(dev\_values);

}

int main(void)

{

clock\_t start, stop;

float \*values = (float\*) malloc( NUM\_VALS \* sizeof(float));

array\_fill(values, NUM\_VALS);

printf(" Array with random numbers ");

array\_print(values, NUM\_VALS) ;

start = clock();

bitonic\_sort(values); /\* Inplace \*/

stop = clock();

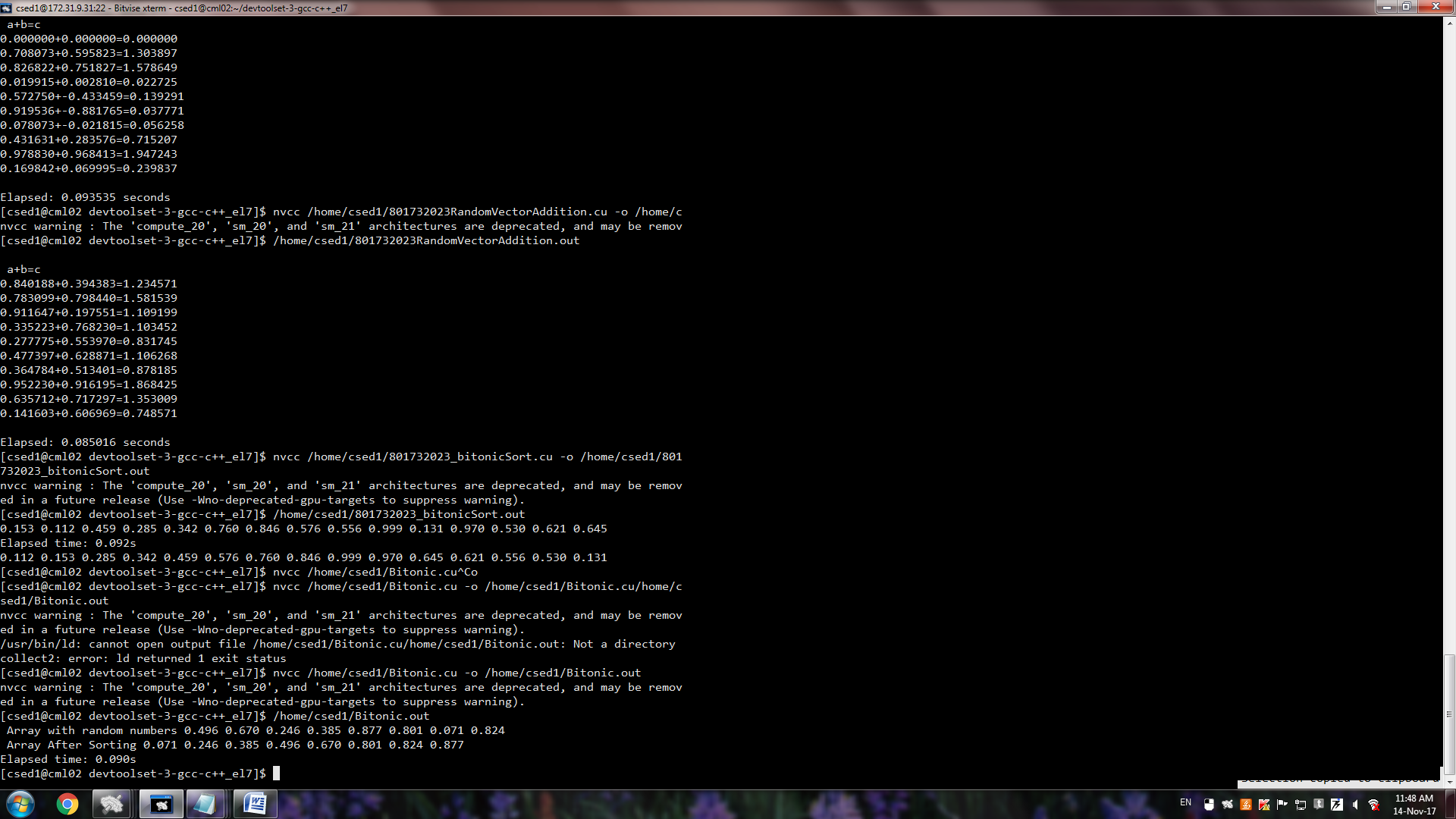
printf(" Array After Sorting ");

array\_print(values, NUM\_VALS) ;

print\_elapsed(start, stop);

}

**Output:**



**Program 10**

**WAP to implement vector addition using sine function in cuda programming**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<cuda\_runtime.h>

#define N 10

\_\_global\_\_ void add(float \*a,float \*b,float \*c)

{

int bid=blockIdx.x;

if(bid<N)

c[bid]=a[bid]+b[bid];

}

int main(void)

{

int i;

float a[N],b[N],c[N];

float \*dev\_a,\*dev\_b,\*dev\_c;

clock\_t tic = clock();

cudaMalloc((void\*\*)&dev\_a,N\*sizeof(float));

cudaMalloc((void\*\*)&dev\_b,N\*sizeof(float));

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

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

{

a[i]=-sin(i)\*sin(i);

b[i]=sin(i)\*sin(i)\*sin(i);

}

cudaMemcpy(dev\_a,a,N\*sizeof(float),cudaMemcpyHostToDevice);

cudaMemcpy(dev\_b,b,N\*sizeof(float),cudaMemcpyHostToDevice);

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

cudaMemcpy(c,dev\_c,N\*sizeof(float),cudaMemcpyDeviceToHost);

printf("\n a+b=c \n");

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

{

printf("%f+%f=%f\n",a[i],b[i],c[i]);

}

clock\_t toc = clock();

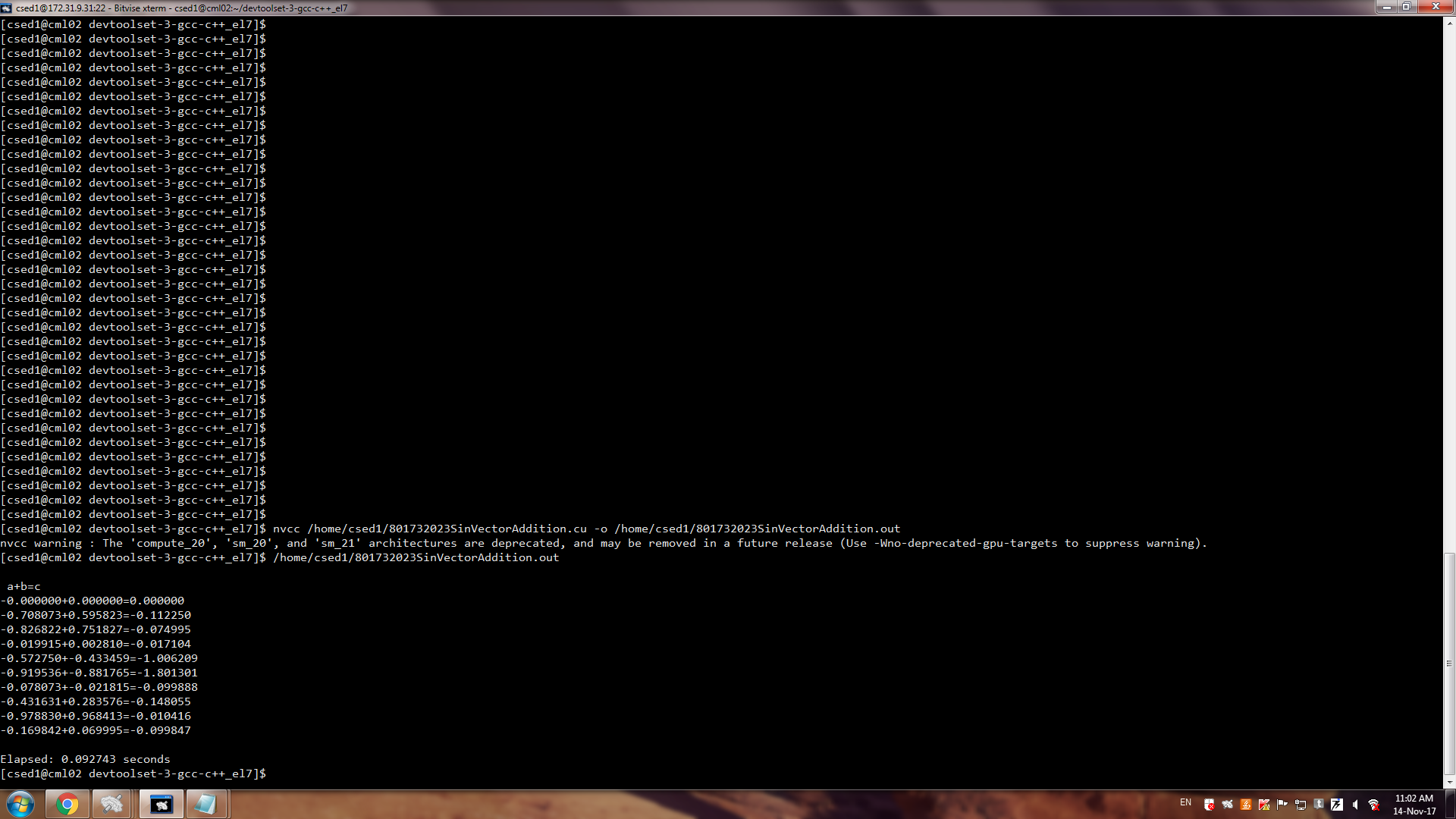
cudaFree(dev\_a);

cudaFree(dev\_b);

cudaFree(dev\_c);

printf("\nElapsed: %f seconds\n", (double)(toc - tic) / CLOCKS\_PER\_SEC);

}



**Program 11**

**WAP to implement vector addition using cosine function**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<cuda\_runtime.h>

#define N 10

\_\_global\_\_ void add(float \*a,float \*b,float \*c)

{

int bid=blockIdx.x;

if(bid<N)

c[bid]=a[bid]+b[bid];

}

int main(void)

{

int i;

float a[N],b[N],c[N];

float \*dev\_a,\*dev\_b,\*dev\_c;

clock\_t tic = clock();

cudaMalloc((void\*\*)&dev\_a,N\*sizeof(float));

cudaMalloc((void\*\*)&dev\_b,N\*sizeof(float));

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

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

{

a[i]=-cos(i)\*cos(i);

b[i]=cos(i)\*cos(i)\*cos(i);

}

cudaMemcpy(dev\_a,a,N\*sizeof(float),cudaMemcpyHostToDevice);

cudaMemcpy(dev\_b,b,N\*sizeof(float),cudaMemcpyHostToDevice);

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

cudaMemcpy(c,dev\_c,N\*sizeof(float),cudaMemcpyDeviceToHost);

printf("\n a+b=c \n");

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

{

printf("%f+%f=%f\n",a[i],b[i],c[i]);

}

clock\_t toc = clock();

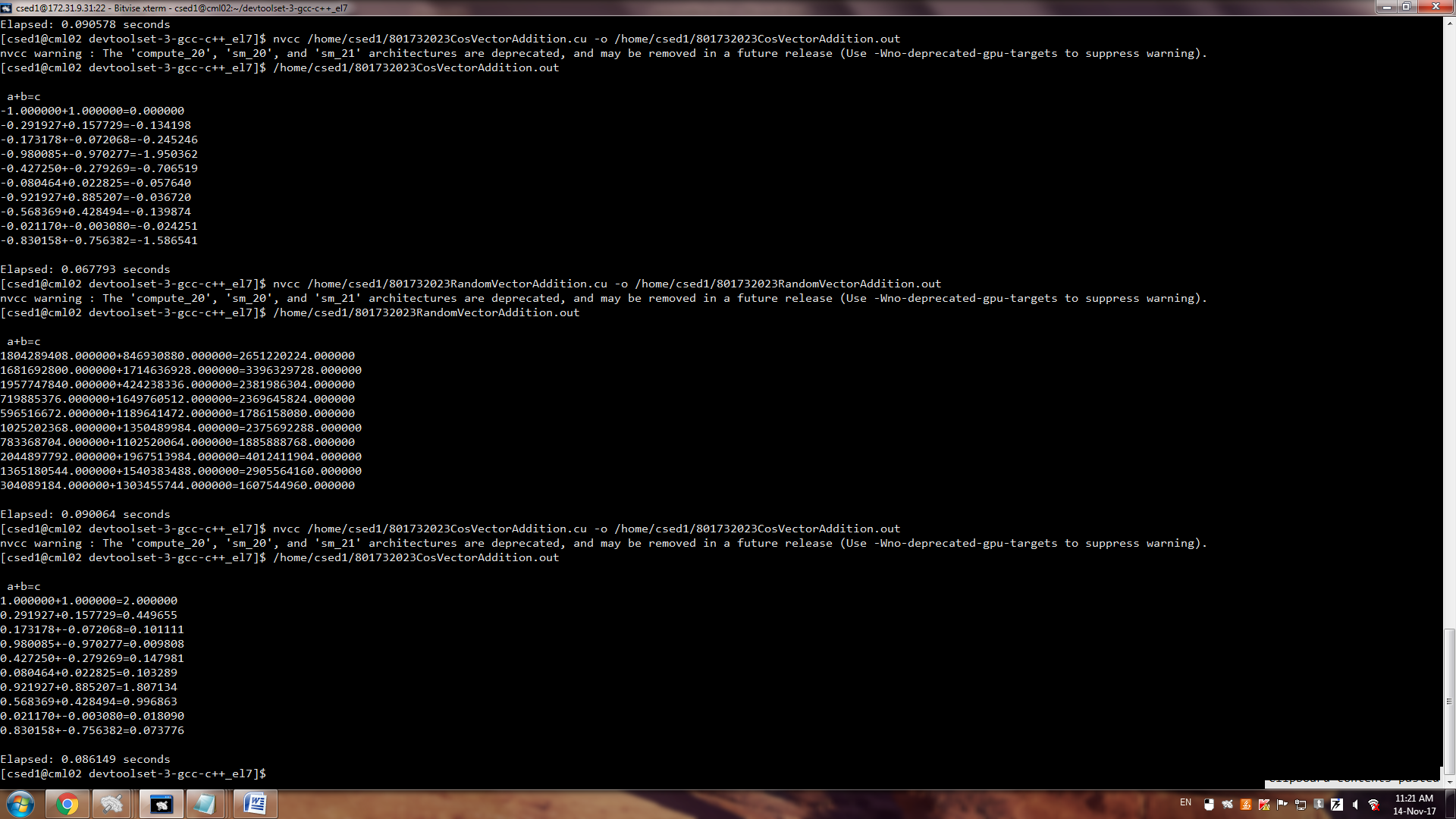
cudaFree(dev\_a);

cudaFree(dev\_b);

cudaFree(dev\_c);

printf("\nElapsed: %f seconds\n", (double)(toc - tic) / CLOCKS\_PER\_SEC);

}



**Program 12**

**WAP to implement vector addition using random function**

#include<stdio.h>

#include<stdlib.h>

#include<cuda.h>

#include<cuda\_runtime.h>

#define N 10

\_\_global\_\_ void add(float \*a,float \*b,float \*c)

{

int bid=blockIdx.x;

if(bid<N)

c[bid]=a[bid]+b[bid];

}

int main(void)

{

int i;

float a[N],b[N],c[N];

float \*dev\_a,\*dev\_b,\*dev\_c;

clock\_t tic = clock();

cudaMalloc((void\*\*)&dev\_a,N\*sizeof(float));

cudaMalloc((void\*\*)&dev\_b,N\*sizeof(float));

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

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

{

a[i]=rand();

b[i]=rand();

}

cudaMemcpy(dev\_a,a,N\*sizeof(float),cudaMemcpyHostToDevice);

cudaMemcpy(dev\_b,b,N\*sizeof(float),cudaMemcpyHostToDevice);

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

cudaMemcpy(c,dev\_c,N\*sizeof(float),cudaMemcpyDeviceToHost);

printf("\n a+b=c \n");

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

{

printf("%f+%f=%f\n",a[i],b[i],c[i]);

}

clock\_t toc = clock();

cudaFree(dev\_a);

cudaFree(dev\_b);

cudaFree(dev\_c);

printf("\nElapsed: %f seconds\n", (double)(toc - tic) / CLOCKS\_PER\_SEC);

}

