//Assignment No. 1

1]min.cu

#include <cuda.h>

#include <stdio.h>

#include <time.h>

#include<math.h>

#define TPB 512

#define SIZE 2048

\_\_global\_\_ void min(int \*a , int \*c)

// kernel function definition

{

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

// initialize i to thread ID

a[2\*i] < a[2\*i+1]? c[i] = a[2\*i]:c[i] = a[2\*i+1];

}

int main()

{

int i;

srand(time(NULL)); //makes use of the computer's internal clock to control the choice of the

int a[SIZE];

// int c[SIZE];

int \*dev\_a, \*dev\_c; //GPU / device parameters

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

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

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

{

a[i] = SIZE - i; // rand()% 1000 + 1;

}

//assign memory to parameters on GPU

// input the numbers

//copy the array from CPU to GPU

for(i=1;i<log2((double)SIZE); i++)

{

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

min<<<1,SIZE/pow(2,i)>>>(dev\_a,dev\_c);

// call kernel function

cudaMemcpy(&a, dev\_c, SIZE\*sizeof(int),cudaMemcpyDeviceToHost);

}

a[0] < a[1]? printf("\nMin: %d", a[0]): printf("\nMin: %d", a[1]);

//printf("\nmax = %d ",a[0]);

//cudaFree(dev\_a);

// Free the allocated memory

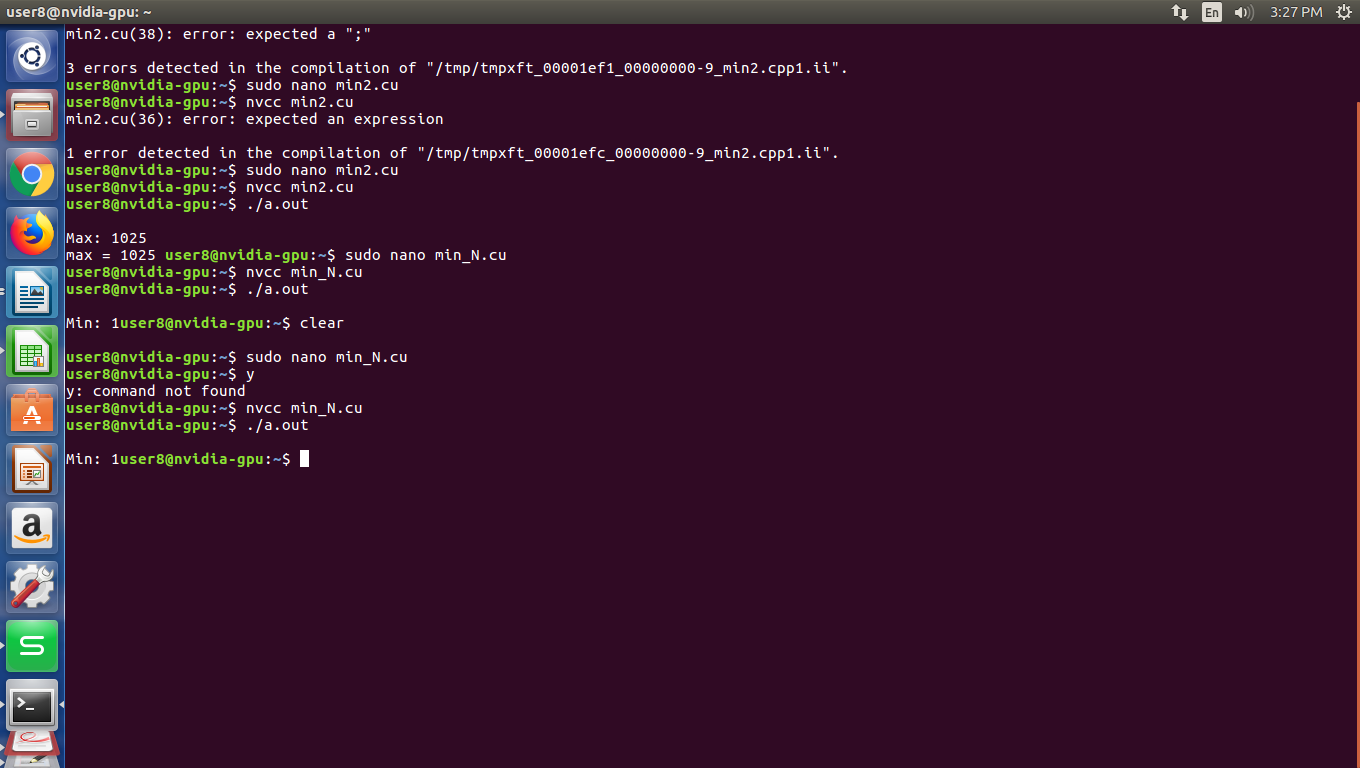
//cudaFree(dev\_c);

// printf("");

return 0;

}

OUTPUT:



2]max.cu

#include <cuda.h>

#include <stdio.h>

#include <time.h>

#include<math.h>

#define TPB 512

#define SIZE 2048

\_\_global\_\_ void max(int \*a , int \*c) // kernel function definition

{

int i = blockIdx.x \* blockDim.x + threadIdx.x; // initialize i to thread ID

a[2\*i] > a[2\*i+1]? c[i] = a[2\*i]:c[i] = a[2\*i+1];

}

int main()

{

int i;

srand(time(NULL)); //makes use of the computer's internal clock to control the choice of the seed

int a[SIZE];

// int c[SIZE];

int \*dev\_a, \*dev\_c; //GPU / device parameters

cudaMalloc((void \*\*) &dev\_a, SIZE\*sizeof(int)); //assign memory to parameters on GPU

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

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

{

a[i] = SIZE - i; // rand()% 1000 + 1; // input the numbers

}

//copy the array from CPU to GPU

for(i=1;i<log2((double)SIZE); i++)

{

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

max<<<1,SIZE/pow(2,i)>>>(dev\_a,dev\_c);

// call kernel function <<<number of blocks, number of threads

cudaMemcpy(&a, dev\_c, SIZE\*sizeof(int),cudaMemcpyDeviceToHost); // copy the result back from GPU to CPU

}

a[0] > a[1]? printf("\nMax: %d", a[0]): printf("\nMax: %d", a[1]);

// printf("\nmax = %d ",a[0]);

//cudaFree(dev\_a); // Free the allocated memory

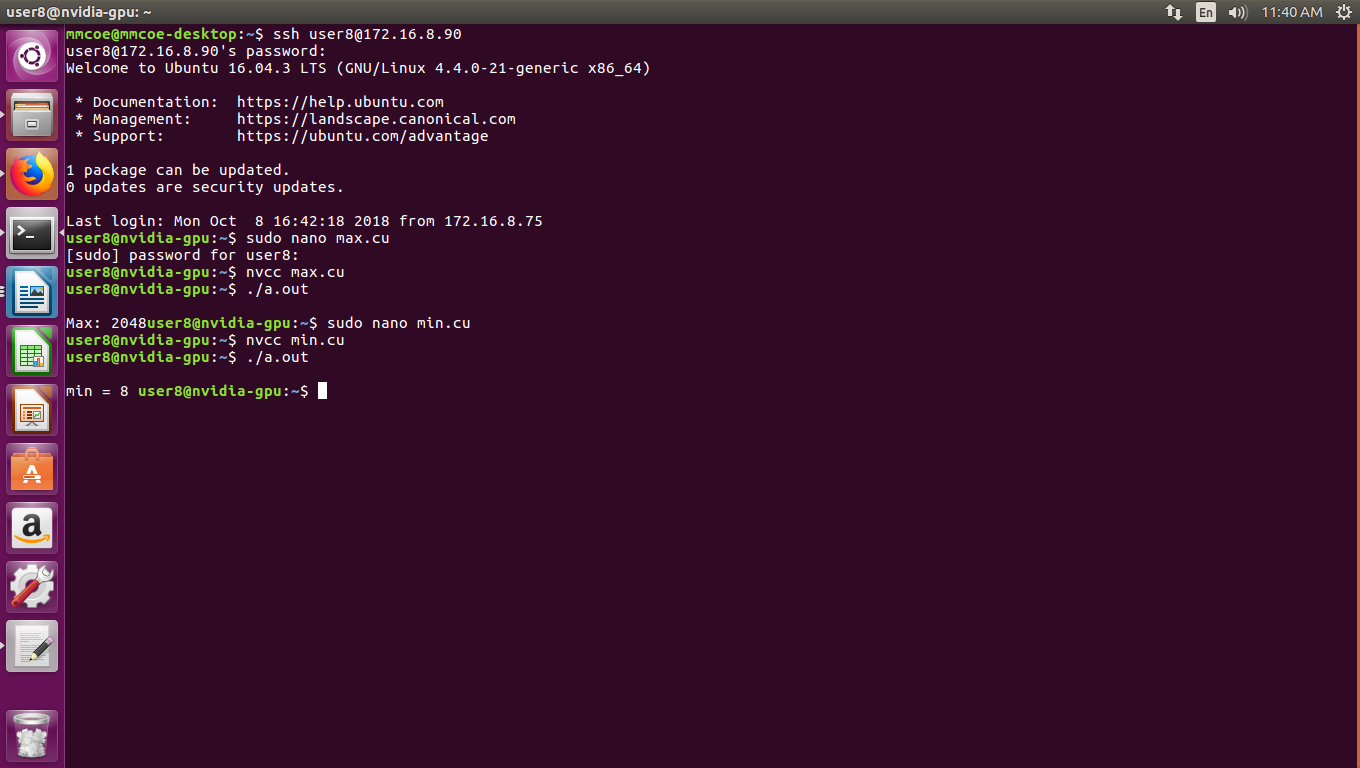
//cudaFree(dev\_c);

// printf("");

return 0;

}

OUTPUT:



3]mean.cu

#include <cuda.h>

#include <stdio.h>

#include <time.h>

#include<math.h>

#define TPB 512

#define SIZE 32

\_\_global\_\_ void max(int \*a , int \*c) // kernel function definition

{

int i = blockIdx.x \* blockDim.x + threadIdx.x; // initialize i to thre$

c[i] = a[i\*2] + a[i\*2+1];

}

int main()

{

int i;

srand(time(NULL)); //makes use of the computer's internal clock to control$

int a[SIZE];

// int c[SIZE];

int \*dev\_a, \*dev\_c; //GPU / device parameters

cudaMalloc((void \*\*) &dev\_a, SIZE\*sizeof(int)); //assign memory to parameters on GPU

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

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

{

a[i] = rand()% 1000 + 1; // input the numbers

}

//copy the array from CPU to GPU

for(i=1;i<log2((double)SIZE); i++)

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

max<<<1,SIZE/pow(2,i)>>>(dev\_a,dev\_c);

// call kernel function <<<number of blocks, number of threads

cudaMemcpy(&a, dev\_c, SIZE\*sizeof(int),cudaMemcpyDeviceToHost); // copy the result back from GPU to CPU

}

int sum = a[0] + a[1];

printf("Sum is :%d\n",sum);

float avg = (float)sum/SIZE;

printf("Average is : %f",avg);

cudaFree(dev\_a); // Free the allocated memory

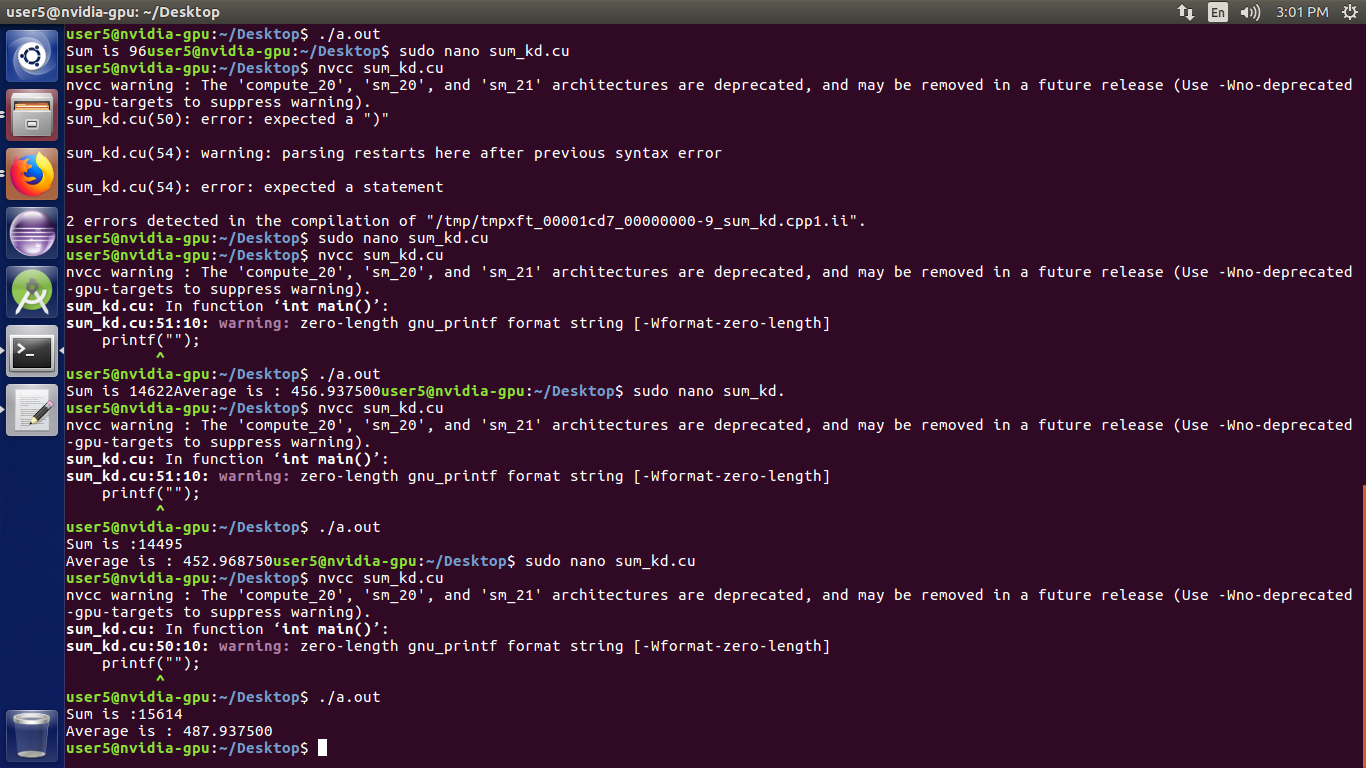
cudaFree(dev\_c);

printf("");

return 0;

}

OUTPUT:



4]std.cu

#include <cuda.h>

#include <stdio.h>

#include <time.h>

#include<math.h>

#define TPB 512

#define SIZE 512

\_\_global\_\_ void sum1(float \*a , float \*c) // kernel function definition

{

int i = blockIdx.x \* blockDim.x + threadIdx.x; // initialize i to thre$

c[i] = a[i\*2] + a[i\*2+1];

}

\_\_global\_\_ void std1(float \*a , float \*c,float mean) // kernel function definition

{

int i = blockIdx.x \* blockDim.x + threadIdx.x; // initialize i to thre$

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

}

int main()

{

int i;

srand(time(NULL)); //makes use of the computer's floaternal clock to control$

float a[SIZE],a\_copy[SIZE];

float c[SIZE];

float \*dev\_a, \*dev\_c; //GPU / device parameters

cudaMalloc((void \*\*) &dev\_a, SIZE\*sizeof(float)); //assign memory to parameters on GPU

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

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

{

a[i] = i+1 ;// rand()% 1000 + 1; // input the numbers

c[i] = (float)a[i];

a\_copy[i] = a[i];

}

//copy the array from CPU to GPU

for(i=1;i<log2((double)SIZE); i++)

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

sum1<<<1,SIZE/pow(2,i)>>>(dev\_a,dev\_c);

// call kernel function <<<number of blocks, number of threads

cudaMemcpy(&a, dev\_c, SIZE\*sizeof(float),cudaMemcpyDeviceToHost); // copy the result back from GPU to CPU

}

float sum = a[0] + a[1];

printf("Sum is :%f\n",sum);

float avg = (float)sum/SIZE;

printf("Average is : %f \n",avg);

cudaFree(dev\_a); // Free the allocated memory

cudaFree(dev\_c);

cudaMalloc((void \*\*) &dev\_a, SIZE\*sizeof(float)); //assign memory to parameters on GPU

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

cudaMemcpy(dev\_a , c, SIZE\*sizeof(float),cudaMemcpyHostToDevice);

std1<<<SIZE/TPB,TPB>>>(dev\_a,dev\_c,avg);

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

cudaFree(dev\_a); // Free the allocated memory

cudaFree(dev\_c);

cudaMalloc((void \*\*) &dev\_a, SIZE\*sizeof(float)); //assign memory to parameters on GPU

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

for(i=1;i<log2((double)SIZE); i++)

{ cudaMemcpy(dev\_a , c, SIZE\*sizeof(float),cudaMemcpyHostToDevice);

sum1<<<1,SIZE/pow(2,i)>>>(dev\_a,dev\_c); // call kernel function <<<number of blocks, number of threads

cudaMemcpy(&a, dev\_c, SIZE\*sizeof(float),cudaMemcpyDeviceToHost); // copy the result back from GPU to CPU

}

float std\_sum = a[0] + a[1];

printf("std sum is : %f\n",std\_sum);

float std\_final = sqrt(std\_sum/(float)SIZE);

printf("STD is : %f \n",std\_final);

}

OUTPUT:

