Universidad Nacional de San Agustin de Arequipa

Año de la recuperación y consolidación de la economía peruana

Escuela Profesional de Ciencia de la Computación

Topicos en Inteligencia Artificial

LABORATORIO 1

Programación en CUDA

A-2025

Profesor: MSc. R. Jesús Cárdenas Talavera

Javier Wilber Quispe Rojas

Índice

1. Implementar el Hola Mundo desde un kernel de CUDA

```
#include<stdio.h>
__global__ void hello(void)
{
    printf("GPU: Hola Mundo!\n");
}
int main(int argc,char **argv)
{

    hello<<<1,10>>>();
    cudaDeviceReset();
    return 0;
}
```

Listing 1: Codigo ejercicio 1

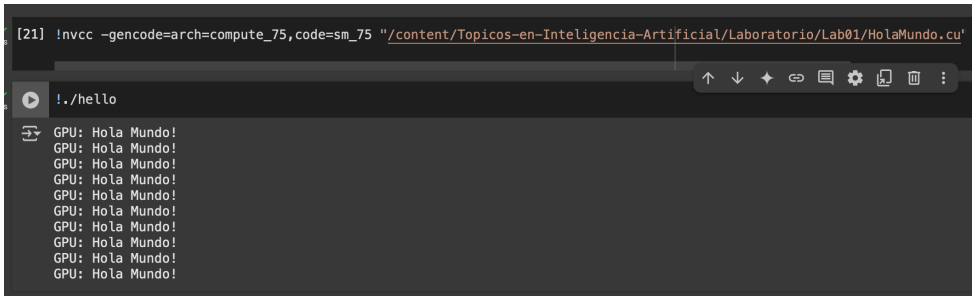

Figura 1: Ejecución

2. Elaborar un programa para CPU que sume dos vectores de valores flotantes. Mida el tiempo de ejecución para sumar vectores mayores a 1000 elementos.

```
#include "iostream"
#include <cstdlib>
#include <vector>
#include <math.h>
#include <chrono>

using namespace std;
const int N=1000;
std::vector<float> A(N), B(N), C(N);
void CrearVectores(vector<float>& A, vector<float>& B){
    srand(time(NULL));
    for(int i=0; i<N ; i++){
        A[i] = (rand() % 1000)/10.0;
        B[i] = (rand() % 1000)/10.0;

    }
}
float Tiempo(vector<float>& A, vector<float>& B , vector<float>& C){
    float time=0.0;
    auto start = std::chrono::high_resolution_clock::now();

    for (int i = 0; i < N; ++i) {
        C[i] = A[i] + B[i];
    }

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<float, std::milli> duration = end - start;


    time=duration.count();

    return time;
```

```
33
34  }
35
36  int main(){
37      CrearVectores(A,B);
38
39      float time =Tiempo(A,B,C);
40
41
42      for(int i=0; i<1000 ; i++)
43      {
44          cout<<"A"<<i<<" : "<<A[i]<<" +"<<"B"<<i<<" : "<<B[i]<<" ="<<"C"<<i<<" : "<<C[i]<<
                endl;
45      }
46
47      cout<<"El tiempo en segundos fue: "<<time<<endl;
48      return 0;
49  }
```

Listing 2: Codigo ejercicio 2

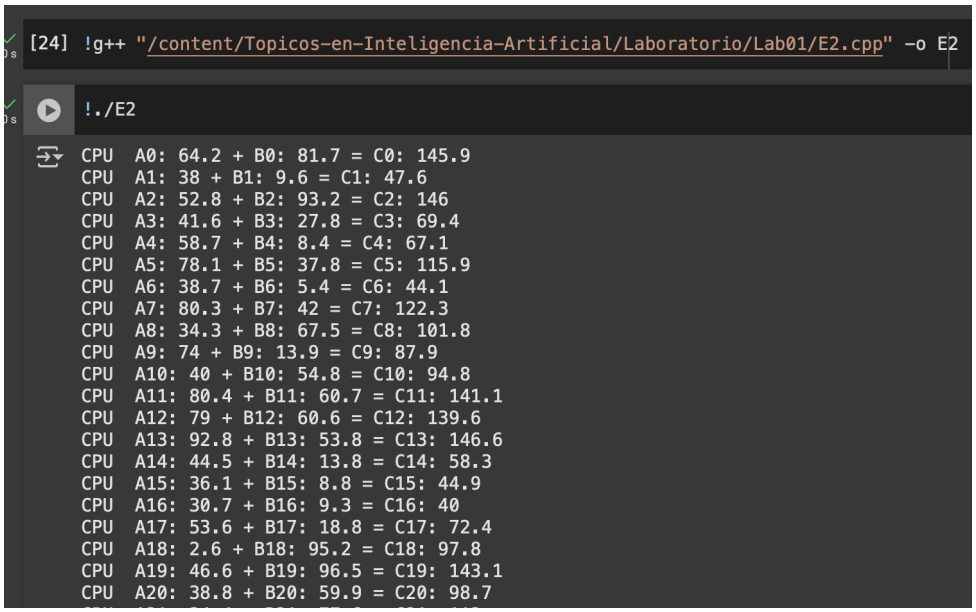

Figura 2: Ejecución ejercicio 2

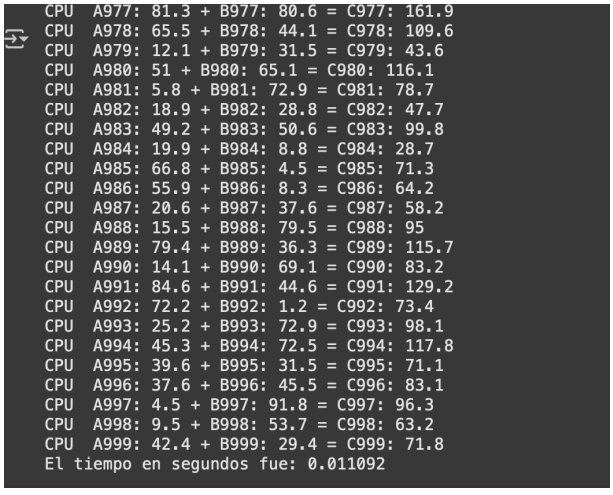

Figura 3: Ejecución ejercicio 2

3. Elaborar un programa que utilice la GPU para la suma de dos vectores de valores flotantes. Mida el tiempo de ejecución para sumar vectores mayores a 1000 elementos.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <cuda_runtime.h>
#include <chrono>

using namespace std;

const int N = 1000;
vector<float> A(N), B(N), C(N);


__global__ void vectorAddGPU(const float* A, const float* B, float* C, int N) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}


void CrearVectores(vector<float>& A, vector<float>& B) {
    srand(time(NULL));
    for (int i = 0; i < N; i++) {
        A[i] = (rand() % 1000) / 10.0f;
        B[i] = (rand() % 1000) / 10.0f;
    }
}


float TiempoGPU(vector<float>& A, vector<float>& B, vector<float>& C) {
    float *d_A, *d_B, *d_C;
    size_t size = N * sizeof(float);

    cudaMalloc(&d_A, size);
    cudaMalloc(&d_B, size);
    cudaMalloc(&d_C, size);

    cudaMemcpy(d_A, A.data(), size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B.data(), size, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

    auto start = std::chrono::high_resolution_clock::now();

    vectorAddGPU<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
    cudaDeviceSynchronize();

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<float, std::milli> duration = end - start;

    cudaMemcpy(C.data(), d_C, size, cudaMemcpyDeviceToHost);

    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);

    return duration.count();
}

int main() {
    CrearVectores(A, B);

    float time = TiempoGPU(A, B, C);

    for (int i = 0; i < 1000; i++) {
        cout << "GPU-"<<"A" << i << ":-" << A[i]
             << "-+-B" << i << ":-" << B[i]
```

```
68                 << " = C" << i << " : " << C[i] << endl;
69         }
70
71         cout << "El tiempo de ejecuci n en GPU fue: " << time << " ms" << endl;
72
73         return 0;
74 }
```

Listing 3: Codigo ejercicio 3

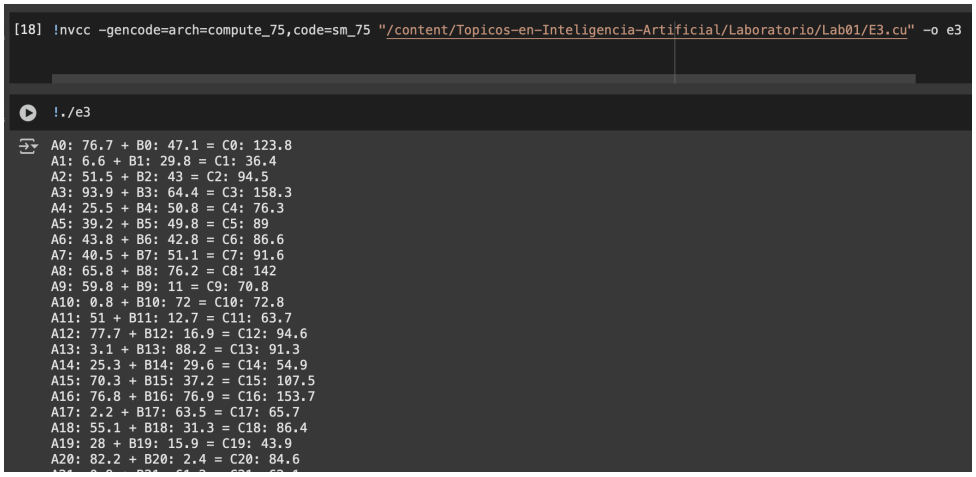

Figura 4: Ejecución ejercicio 3

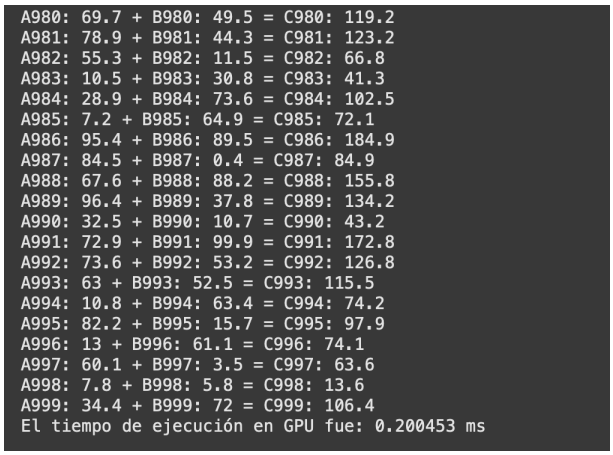

Figura 5: Ejecución ejercicio 3