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

PRACTICA 1

CUDA Programming

A-2025

Profesor: MSc. PERCY MALDONADO QUISPE

Javier Wilber Quispe Rojas

Índice

1. Get information about the device where you are running the code.

```cpp
#include <stdio.h>

int main()
{
    int noOfDevices;

    cudaGetDeviceCount (&noOfDevices);

    cudaDeviceProp prop;
    for (int i = 0; i < noOfDevices; i++)
    {

        cudaGetDeviceProperties (&prop, i);

        printf("Device Name:\t%s\n", prop.name);
        printf("Total global memory:\t%ld\n", prop.totalGlobalMem);
        printf("No. of SMs:\t%d\n", prop.multiProcessorCount);
        printf("Shared memory / SM:\t%ld\n", prop.sharedMemPerBlock);
        printf("Registers / SM:\t%d\n", prop.regsPerBlock);
    }

    return 1;
}
```

Listing 1: Codigo ejercicio 1

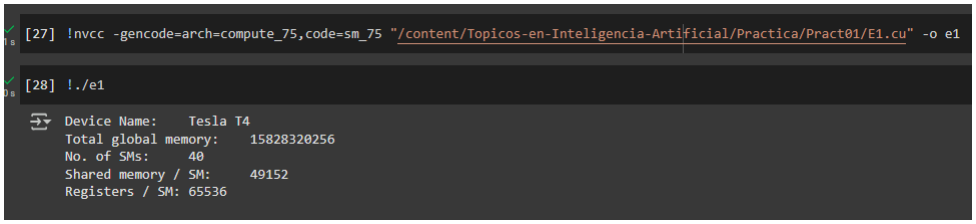

Figura 1: Ejecución ejercicio 1

2. Implements vector addition in CUDA.

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>


__global__ void suma_vectores(float *c, float *a, float *b, int N) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) {
        c[idx] = a[idx] + b[idx];
    }
}

int main(void) {

    float *a_h, *b_h, *c_h;

    float *a_d, *b_d, *c_d;
    int N=10;
    size_t size = N * sizeof(float);


    a_h = (float *)malloc(size);
    b_h = (float *)malloc(size);
    c_h = (float *)malloc(size);


```

```
    for (int i = 0; i < N; i++) {
        a_h[i] = (float)i;
        b_h[i] = (float)(i + 1);
    }

    printf("\nArreglo a:\n");
    for (int i = 0; i < N; i++) printf("%f ", a_h[i]);
    printf("\nArreglo b:\n");
    for (int i = 0; i < N; i++) printf("%f ", b_h[i]);


    cudaMalloc((void **) &a_d, size);
    cudaMalloc((void **) &b_d, size);
    cudaMalloc((void **) &c_d, size);


    cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
    cudaMemcpy(b_d, b_h, size, cudaMemcpyHostToDevice);


    int block_size = 8;
    int n_blocks = (N + block_size - 1) / block_size;
    suma_vectores<<<n_blocks, block_size>>>(c_d, a_d, b_d, N);


    cudaMemcpy(c_h, c_d, size, cudaMemcpyDeviceToHost);


    printf("\nResultado c:\n");
    for (int i = 0; i < N; i++) printf("%f ", c_h[i]);
    printf("\n");


    free(a_h);
    free(b_h);
    free(c_h);


    cudaFree(a_d);
    cudaFree(b_d);
    cudaFree(c_d);

    return 0;
}
```

Listing 2: Codigo ejercicio 2

Figura 2: Ejecución ejercicio 2

3. Implements matrix multiplication in CUDA.

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>

#define BLOCK_SIZE 16

int div_up(int x, int y) {
    return (x + y - 1) / y;
}
```

```
10
11  // Funci n Kernel que se ejecuta en el Device
12  __global__ void Multiplica_Matrices_GM(float *C, float *A, float *B, int nfil, int ncol)
        {
13      int idx = blockIdx.x * blockDim.x + threadIdx.x;
14      int idy = blockIdx.y * blockDim.y + threadIdx.y;
15      int index = idy * ncol + idx;
16
17      if (idy < nfil && idx < ncol) {
18          float sum = 0.0f;
19          for (int k = 0; k < ncol; k++) {
20              sum += A[idy * ncol + k] * B[k * ncol + idx];
21          }
22          C[index] = sum;
23      }
24  }
25
26  int main(void) {
27
28      float *A_h, *B_h, *C_h;
29
30      float *A_d, *B_d, *C_d;
31
32      int nfil = 5;
33      int ncol = 5;
34      int N = nfil * ncol;
35
36      size_t size = N * sizeof(float);
37
38      cudaEvent_t start, stop;
39      float time;
40      cudaEventCreate(&start);
41      cudaEventCreate(&stop);
42
43
44      A_h = (float *)malloc(size);
45      B_h = (float *)malloc(size);
46      C_h = (float *)malloc(size);
47
48
49      for (int i = 0; i < nfil; i++) {
50          for (int j = 0; j < ncol; j++) {
51              A_h[i * ncol + j] = 1.0f;
52              B_h[i * ncol + j] = 2.0f;
53          }
54      }
55
56
57      cudaMalloc((void **) &A_d, size);
58      cudaMalloc((void **) &B_d, size);
59      cudaMalloc((void **) &C_d, size);
60
61
62      cudaMemcpy(A_d, A_h, size, cudaMemcpyHostToDevice);
63      cudaMemcpy(B_d, B_h, size, cudaMemcpyHostToDevice);
64
65
66      dim3 block_size(BLOCK_SIZE, BLOCK_SIZE);
67      dim3 n_blocks(div_up(ncol, BLOCK_SIZE), div_up(nfil, BLOCK_SIZE));
68      cudaEventRecord(start);
69      Multiplica_Matrices_GM<<<n_blocks, block_size>>>(C_d, A_d, B_d, nfil, ncol);
70      cudaEventRecord(stop);
71
72
73      cudaEventSynchronize(stop);
74
75
76      cudaMemcpy(C_h, C_d, size, cudaMemcpyDeviceToHost);
77
78
79      printf("\nMatriz A:\n");
80      for (int i = 0; i < nfil; i++) {
81          for (int j = 0; j < ncol; j++) {
```

```
82              printf("%2.2f-", A_h[i * ncol + j]);
83          }
84          printf("\n");
85      }
86
87      printf("\nMatriz-B:\n");
88      for (int i = 0; i < nfil; i++) {
89          for (int j = 0; j < ncol; j++) {
90              printf("%2.2f-", B_h[i * ncol + j]);
91          }
92          printf("\n");
93      }
94
95      printf("\nMatriz-C:\n");
96      for (int i = 0; i < nfil; i++) {
97          for (int j = 0; j < ncol; j++) {
98              printf("%2.2f-", C_h[i * ncol + j]);
99          }
100         printf("\n");
101     }
102
103
104     cudaEventElapsedTime(&time, start, stop);
105     printf("Tiempo-de-ejecuci n:-%3.1f-ms\n", time);
106
107
108     free(A_h);
109     free(B_h);
110     free(C_h);
111
112
113     cudaFree(A_d);
114     cudaFree(B_d);
115     cudaFree(C_d);
116
117     return 0;
118 }
```

Listing 3: Codigo ejercicio 3

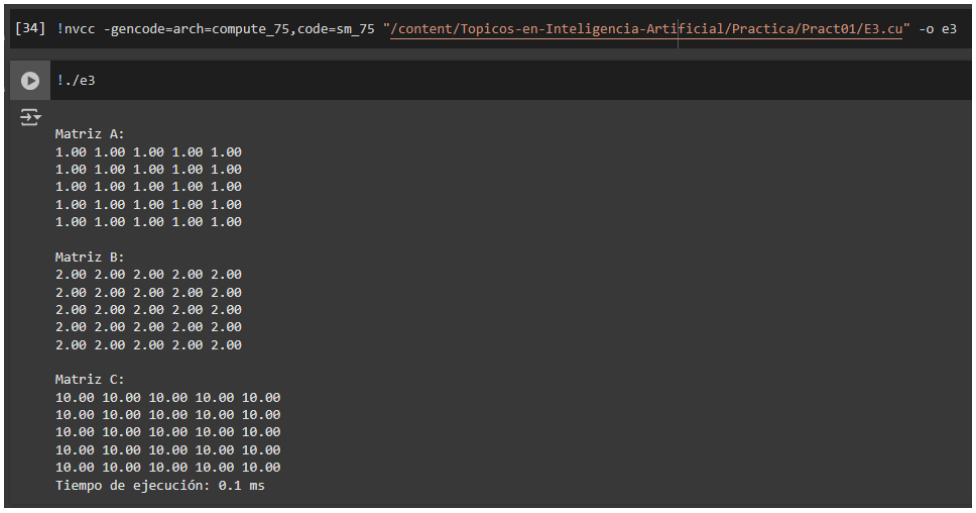

Figura 3: Ejecución ejercicio 3

4. Link de colab

Para poder ejecutar los codigos en el colab debe cambiar entorno de ejecucion **GPU T4** , luego ejecutar todo.
`https://colab.research.google.com/drive/1OeqldS_mWLVCoVe_-3lTho44W_QBz6rC?usp=sharing`