UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

Práctica 2

Parte 1: Compilando Programas en C

Máster Universitario en Ingeniería de Computadores y Redes

1 de diciembre de 2025

Pablo González Segarra

Índice

Ejercicio 1 y 2

Debido a la similitud entre los ejercicios 1 y 2, se presentan juntos en esta sección. A continuación se muestra el código modificado necesario para ejecutar el programa en la GPU utilizando CUDA y comprobar que los resultados son correctos.

```
#define N 100000000

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

__global__ void vector_add(float *out, float *a, float *b, int n) {
    for(int i = 0; i < n; i++){
        out[i] = a[i] + b[i];
    }
}

int main(){
    float *a, *b, *out;

    // Allocate memory
    a   = (float*)malloc(sizeof(float) * N);
    b   = (float*)malloc(sizeof(float) * N);
    out = (float*)malloc(sizeof(float) * N);

    // Initialize array
    for(int i = 0; i < N; i++){
        a[i] = 1.0f; b[i] = 2.0f;
    }

    float *a_cuda = NULL, *b_cuda = NULL, *out_cuda = NULL;

    // Allocate device memory
    cudaMalloc((void**)&a_cuda, sizeof(float) * N);
    cudaMalloc((void**)&b_cuda, sizeof(float) * N);
    cudaMalloc((void**)&out_cuda, sizeof(float) * N);

    // Copy inputs to device
    cudaMemcpy(a_cuda, a, sizeof(float) * N, cudaMemcpyHostToDevice);
    cudaMemcpy(b_cuda, b, sizeof(float) * N, cudaMemcpyHostToDevice);

    // Main function
    vector_add<<<1,1>>>(out_cuda, a_cuda, b_cuda, N);

    cudaMemcpy(out, out_cuda, sizeof(float) * N,
    cudaMemcpyDeviceToHost);

    cudaDeviceSynchronize();

    // Verify result
    const float MAX_ERR = 1e-6f;
    for(int i = 0; i < N; i++){
        if(fabsf(out[i] - 3.0f) > MAX_ERR){
            printf("Error at index %d: %f (diff=%f)\n", i, out[i],
    fabsf(out[i] - 3.0f));
```

```
50                 return -1;
51             }
52         }
53
54         printf("Success! All values are correct.\n");
55
56         // Free memory
57         cudaFree(a_cuda);
58         cudaFree(b_cuda);
59         cudaFree(out_cuda);
60
61         free(a);
62         free(b);
63         free(out);
64 }
```

Listing 1: Código CUDA modificado para ejecutar en GPU

Ejercicio 3

El codigo modificado para poder obtener los tiempos de ejecución es el siguiente:

```
1  #define N 100000000
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <math.h>
6  #include <cuda_runtime.h>
7
8  __global__ void vector_add(float *out, float *a, float *b, int n) {
9      for(int i = 0; i < n; i++){
10         out[i] = a[i] + b[i];
11     }
12 }
13
14 int main(){
15     float *a, *b, *out;
16
17     // Allocate memory
18     a   = (float*)malloc(sizeof(float) * N);
19     b   = (float*)malloc(sizeof(float) * N);
20     out = (float*)malloc(sizeof(float) * N);
21
22     // Initialize array
23     for(int i = 0; i < N; i++){
24         a[i] = 1.0f; b[i] = 2.0f;
25     }
26
27     float *a_cuda = NULL, *b_cuda = NULL, *out_cuda = NULL;
28
29     // Create events for timing
30     cudaEvent_t start, end;
31     cudaEventCreate(&start);
32     cudaEventCreate(&end);
33
```

```
34      // Start timing
35      cudaEventRecord(start);
36
37      // Allocate device memory
38      cudaMalloc((void**)&a_cuda, sizeof(float) * N);
39      cudaMalloc((void**)&b_cuda, sizeof(float) * N);
40      cudaMalloc((void**)&out_cuda, sizeof(float) * N);
41
42      // Copy inputs to device
43      cudaMemcpy(a_cuda, a, sizeof(float) * N, cudaMemcpyHostToDevice);
44      cudaMemcpy(b_cuda, b, sizeof(float) * N, cudaMemcpyHostToDevice);
45
46      // Main function
47      vector_add<<<1,1>>>(out_cuda, a_cuda, b_cuda, N);
48
49      cudaMemcpy(out, out_cuda, sizeof(float) * N,
        cudaMemcpyDeviceToHost);
50
51      // End timing
52      cudaEventRecord(end);
53      cudaEventSynchronize(end);
54      float milliseconds = 0;
55      cudaEventElapsedTime(&milliseconds, start, end);
56      printf("Time elapsed: %f ms\n", milliseconds);
57
58      // Verify result
59      const float MAX_ERR = 1e-6f;
60      for(int i = 0; i < N; i++){
61          if(fabsf(out[i] - 3.0f) > MAX_ERR){
62              printf("Error at index %d: %f (diff=%f)\n", i, out[i],
        fabsf(out[i] - 3.0f));
63              return -1;
64          }
65      }
66
67      printf("Success! All values are correct.\n");
68
69      // Free memory
70      cudaFree(a_cuda);
71      cudaFree(b_cuda);
72      cudaFree(out_cuda);
73
74      free(a);
75      free(b);
76      free(out);
77  }
```

Listing 2: Código CUDA modificado para medir tiempos de ejecución

Se ha ejecutado 5 veces para evitar variaciones ajenas al rendimiento del código. El tiempo de ejecucion tiene en cuenta tanto la transferencia de datos entre host y device como la ejecución del kernel.

Resultado de las ejecuciones:

1. 5887.4 ms
2. 5864.1 ms

3. 5918.6 ms
4. 5856.9 ms
5. 5868.1 ms

Siendo el mejor tiempo 5856.9 ms.

Ejercicio 4

extbfVersión	**Tiempo de ejecución (ms)**	**Speedup GPU vs CPU**
1 thread / block		
16 threads / block		
32 threads / block		
256 threads / block		

Tabla 1: Tiempos de ejecución y speedup para distintas configuraciones de hilos/ bloque

Ejercicio 5

Ejercicio 6

Ejercicio 7