

ARQUITECTURA DE COMPUTADORES

1.CÓDIGO

```
/*
 * ARQUITECTURA DE COMPUTADORES
 * 2º Grado en Ingenieria Informatica
 *
 * ENTREGA Básico 2:
 * >> Función kernel
 *
 * Alumno: Antonio Alonso Briones
 * Fecha: 18/09/2024
 *
 */

// includes
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>

// declaración de funciones
__host__ void generar_vector_random(int* vector, int N)
{
    for (int i = 0; i < N; i++)
    {
        vector[i] = rand() % 10; // Valores entre 0 y 9
    }
}

__global__ void invertir_y_sumar_vectores(int* vector1, int* vector2, int*
resultado, int N)
{
    int id = threadIdx.x;
    if (id < N)
    {
        vector2[id] = vector1[N - id - 1]; // Invertir el vector
        resultado[id] = vector1[id] + vector2[id];
    }
}

int main(int argc, char** argv)
{
    // Declaración de variables
    int N;
    printf("Introduce el tamaño de los vectores: ");
    scanf("%d", &N);

    // Obtener el número de dispositivos CUDA
    int deviceCount;
    cudaGetDeviceCount(&deviceCount);
    if (deviceCount == 0)
    {
        printf("No se han encontrado dispositivos CUDA compatibles.\n");
        return 1;
    }
}
```

```

    // Obtener propiedades del dispositivo (suponemos que usamos el
dispositivo 0)
    cudaDeviceProp deviceProp;
    cudaGetDeviceProperties(&deviceProp, 0);

    // Cálculo del número total de núcleos (SP) dependiendo de la capacidad
de cómputo
    int cudaCores;
    int SM = deviceProp.multiProcessorCount;
    int major = deviceProp.major;
    int minor = deviceProp.minor;
    const char* archName;
    switch (major)
    {
    case 1: // Tesla
        archName = "TESLA";
        cudaCores = 8;
        break;
    case 2: // Fermi
        archName = "FERMI";
        if (minor == 0)
            cudaCores = 32;
        else
            cudaCores = 48;
        break;
    case 3: // Kepler
        archName = "KEPLER";
        cudaCores = 192;
        break;
    case 5: // Maxwell
        archName = "MAXWELL";
        cudaCores = 128;
        break;
    case 6: // Pascal
        archName = "PASCAL";
        if (minor == 1 || minor == 2)
            cudaCores = 128;
        else
            cudaCores = 64;
        break;
    case 7: // Volta y Turing
        if (minor == 0)
            archName = "VOLTA";
        else
            archName = "TURING";
        cudaCores = 64;
        break;
    case 8: // Ampere
        archName = "AMPERE";
        cudaCores = 64;
        break;
    case 9: // Hopper
        archName = "HOPPER";
        cudaCores = 128;
        break;
    default: // Arquitectura desconocida
        archName = "DESCONOCIDA";
        cudaCores = 0;
        break;
    }

    int totalCores = SM * cudaCores;
    float globalMemMiB = deviceProp.totalGlobalMem / (1024.0 * 1024.0);

```

```

// Mostrar propiedades del dispositivo
printf("> Propiedades del dispositivo seleccionado:\n");
printf("Nombre: %s\n", deviceProp.name);
printf("Arquitectura CUDA: %s\n", archName);
printf("Capacidad de cómputo: %d.%d\n", major, minor);
printf("Número de multiprocesadores: %d\n", SM);
printf("Número de núcleos CUDA (%d x %d): %d\n", SM, cudaCores,
totalCores);
printf("Tamaño de la memoria global: %.2f MiB\n", globalMemMiB);

// Obtener el máximo número de hilos por bloque
int maxThreadsPerBlock = deviceProp.maxThreadsPerBlock;

// Verificar que N no exceda el máximo de hilos por bloque
if (N > maxThreadsPerBlock)
{
    printf("N excede el máximo número de hilos por bloque (%d). Se
limitará N a %d.\n", maxThreadsPerBlock, maxThreadsPerBlock);
    N = maxThreadsPerBlock;
}

// Declaración de punteros para host y device
int* hst_vector1, * hst_vector2, * hst_resultado;
int* dev_vector1, * dev_vector2, * dev_resultado;

// Reserva de memoria en el host
hst_vector1 = (int*)malloc(N * sizeof(int));
hst_vector2 = (int*)malloc(N * sizeof(int));
hst_resultado = (int*)malloc(N * sizeof(int));

// Reserva de memoria en el device
cudaMalloc((void*)&dev_vector1, N * sizeof(int));
cudaMalloc((void*)&dev_vector2, N * sizeof(int));
cudaMalloc((void*)&dev_resultado, N * sizeof(int));

// Inicialización del primer vector
printf("> Generando el vector 1...\n");
srand(0); // Inicializar la semilla con un valor fijo
generar_vector_random(hst_vector1, N);

// Copiar el vector 1 al device
cudaMemcpy(dev_vector1, hst_vector1, N * sizeof(int),
cudaMemcpyHostToDevice);

// Lanzamiento del kernel
printf("> Generando el vector 2 y realizando la suma...\n");
invertir_y_sumar_vectores << <1, N >> > (dev_vector1, dev_vector2,
dev_resultado, N);

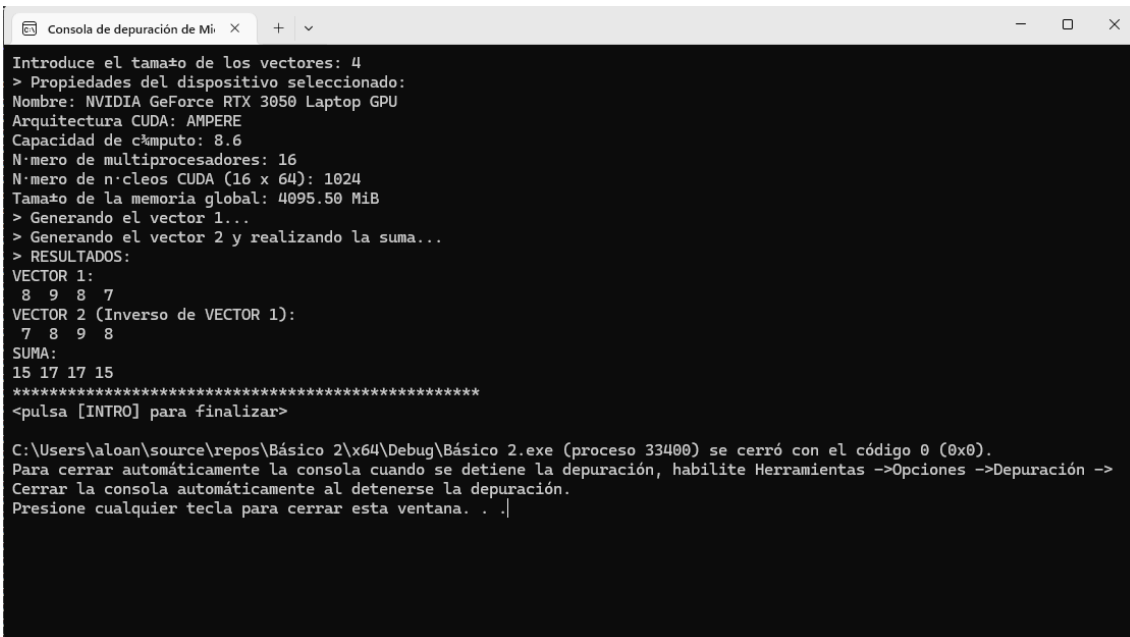
// Sincronizar para esperar a que el kernel termine
cudaDeviceSynchronize();

// Copiar datos desde el device al host
cudaMemcpy(hst_vector2, dev_vector2, N * sizeof(int),
cudaMemcpyDeviceToHost);
cudaMemcpy(hst_resultado, dev_resultado, N * sizeof(int),
cudaMemcpyDeviceToHost);

// Impresión de resultados
printf("> RESULTADOS:\n");
printf("VECTOR 1:\n");
for (int i = 0; i < N; i++)

```


3. SALIDA POR PANTALLA



```
Consola de depuración de Mi... X + v
Introduce el tamaño de los vectores: 4
> Propiedades del dispositivo seleccionado:
Nombre: NVIDIA GeForce RTX 3050 Laptop GPU
Arquitectura CUDA: AMPERE
Capacidad de cómputo: 8.6
Número de multiprocesadores: 16
Número de núcleos CUDA (16 x 64): 1024
Tamaño de la memoria global: 4095.50 MiB
> Generando el vector 1...
> Generando el vector 2 y realizando la suma...
> RESULTADOS:
VECTOR 1:
8 9 8 7
VECTOR 2 (Inverso de VECTOR 1):
7 8 9 8
SUMA:
15 17 17 15
*****
<pulsa [INTRO] para finalizar>

C:\Users\alooan\source\repos\Básico 2\x64\Debug\Básico 2.exe (proceso 33400) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->
Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . . |
```