Introducción a GPU Computing

Grado en Ingeniería Informática

UNIVERSIDAD NEBRIJA

CPU vs GPU

- CPU (Central Processing Unit): Diseñadas para ser un procesador multipropósito
- GPU (Graphics Processing Unit): Diseñadas específicamente para el procesamiento de gráficos

CPU vs GPU

- CPU (Central Processing Unit): Diseñadas para ser un procesador multipropósito
- GPU (Graphics Processing Unit): Diseñadas específicamente para el procesamiento de gráficos

	CPU	GPU
Número de núcleos	2-8	Miles

Ejemplo

- Supongamos que queremos dibujar un cuadrado:

Ciclo	1 núcleo	4 núcleos
Ciclo 1	Calcula la primera línea	Cada núcleo calcula una línea
Ciclo 2	Calcula la segunda línea	Cada núcleo dibuja una línea
Ciclo 3	Calcula la tercera línea	
Ciclo 4	Calcula la cuarta línea	
Ciclo 5	Dibuja la primera línea	
Ciclo 6	Dibuja la segunda línea	
Ciclo 7	Dibuja la tercera línea	
Ciclo 8	Dibuja la cuarta línea	

CPU vs GPU

- CPU (Central Processing Unit): Diseñadas para ser un procesador multipropósito
- GPU (Graphics Processing Unit): Diseñadas específicamente para el procesamiento de gráficos

	CPU	GPU
Número de núcleos	2-8	Miles
Frecuencia	**3,5 GHz**	**1500 MHz**

CPU vs GPU

- CPU (Central Processing Unit): Diseñadas para ser un procesador multipropósito
- GPU (Graphics Processing Unit): Diseñadas específicamente para el procesamiento de gráficos

	CPU	GPU
Frecuencia	**3,5 GHz**	**1500 MHz**
Tipo de ejecución	Secuencial	Paralelo
Datos por ciclo	Pocos	Muchos
Ancho de banda	Pequeño	Grande

CPU vs GPU

- CPU (Central Processing Unit): Diseñadas para ser un procesador multipropósito
- GPU (Graphics Processing Unit): Diseñadas específicamente para el procesamiento de gráficos

	CPU	GPU
Número de núcleos	2-8	Miles
Frecuencia	3,5 GHz	1500 MHz
Controlador de memoria	**64 bits**	**32 bits**

GPU Computing

- Dos tipos de dispositivos: Host (CPU) y Device (GPU)
- Se utiliza para hacer cálculos masivos sobre las GPUs
- Problema: Lenguajes diseñados para realizar gráficos en pantalla
- Solución: CUDA (Arquitectura Unificada de Dispositivos de Cómputo)

CUDA

- Programación en lenguajes de alto nivel como C o C++
- Permite especificar el código que queremos que se ejecute en el device
- Se creó una única unidad de ejecución con el mismo set de instrucciones y mismos recursos

Secuencia de operaciones CUDA

1. Declarar y asignar memoria para host y device.
2. Inicializar datos del host.
3. Transferir datos del host al device.
4. Ejecutar uno o más kernels.
5. Transferir datos desde el device al host.
6. Liberar recursos.

Ejemplo: Suma de vectores

1. Declarar y asignar memoria para host y device.

```
//Allocate host memory
a = (float*)malloc(sizeof(float) * N);
b = (float*)malloc(sizeof(float) * N);
out = (float*)malloc(sizeof(float) * N);
//Allocate device memory
cudaMalloc((void**)&d_a, sizeof(float) * N);
cudaMalloc((void**)&d_b, sizeof(float) * N);
cudaMalloc((void**)&d_out, sizeof(float) * N);
```

Ejemplo: Suma de vectores

2. Inicializar datos del host.

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

Ejemplo: Suma de vectores

3. Transferir datos del host al device.

```
//Transfer data from host to device memory
cudaMemcpy(d_a, a, sizeof(float) * N, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, sizeof(float) * N, cudaMemcpyHostToDevice);
```

Ejemplo: Suma de vectores

4. Ejecutar uno o más kernels:

```
función<<<Número de bloques, número de hilos por bloque>>>
(parámetros);
```

```
//Executing kernel
vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```

Ejemplo: Suma de vectores

4. Ejecutar uno o más kernels:

```
función<<<Número de bloques, número de hilos por bloque>>>
(parámetros);
```

```
//Executing kernel
vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```

Núcleo: formafo por bloques

Bloque: formado por hilos

Ejemplo: Suma de vectores

5. Transferir datos desde el device al host.

```
//Transfer data back to host memory
cudaMemcpy(out, d_out, sizeof(float) * N, cudaMemcpyDeviceToHost);
```

Ejemplo: Suma de vectores

6. Liberar memoria.

```
//Deallocate device memory
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_out);

//Deallocate host memory
free(a);
free(b);
free(out);
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x;
    int stride = blockDim.x;
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x;
    int stride = blockDim.x;
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x;
    int stride = blockDim.x;
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x;
    int stride = blockDim.x;
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x;
    int stride = blockDim.x;
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}
```

```
vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x; → de 0 a 255
    int stride = blockDim.x;
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}

vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```

Ejemplo: Suma de vectores

- Función suma de vectores

```
__global__ void vector_add(float *out, float *a, float *b, int n) {
    int index = threadIdx.x; → de 0 a 255
    int stride = blockDim.x; → 256
    for(int i = index; i < n; i += stride){
        out[i] = a[i] + b[i];
    }
}

vector_add<<<1,256>>>(d_out, d_a, d_b, N);
```