universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO CUDA

Autor: Adrián Carral Martínez

ÍNDICE:

1. Introducción.

El presente trabajo tiene como objetivo implementar la multiplicación de matrices cuadradas utilizando CUDA en dos modalidades: usando memoria compartida y usando memoria unificada (Unified Memory). Estas estrategias permiten aprovechar la arquitectura paralela de las GPU para mejorar el rendimiento computacional. Se han trabajado los archivos dgemm_gpu_shared.cu y dgemm_gpu_shared_uvm.cu, completando las secciones TODO, ejecutando los programas y analizando su precisión y rendimiento.

2. Descripción del problema.

El problema consiste en multiplicar dos matrices cuadradas de tamaño n×n empleando CUDA. La computación se paraleliza mediante hilos agrupados en bloques. En el primer ejercicio se optimiza el acceso a memoria usando memoria compartida dentro de la GPU. En el segundo, se aprovecha la memoria unificada, que permite que CPU y GPU accedan a los mismos datos sin necesidad de transferencias explícitas.

3. Desarrollo de la solución.

3.1 Ejercicio 1: Memoria compartida.

3.1.1 Kernel dgemm_gpu_shared.

➢ Memoria compartida:

shared double aSub[BLOCK_SIZE][BLOCK_SIZE];

shared double bSub[BLOCK_SIZE][BLOCK_SIZE];

➢ Cálculo de índices globales:

int idxX = blockIdx.x * blockDim.x + threadIdx.x;

int idxY = blockIdx.y * blockDim.y + threadIdx.y;

➢ Offset del subbloque:

int blockaY = blockIdx.y * BLOCK_SIZE;

int blockbX = blockIdx.x * BLOCK_SIZE;

➢ Carga en memoria compartida desde memoria global o con ceros si fuera de límite.

➢ Sincronización: __syncthreads();

➢ Producto escalar:

```
for (int i = 0; i < blockDim.x; ++i)
        sum += aSub[threadIdx.y][i] * bSub[i][threadIdx.x];
```

➢ Escritura del resultado:

```
if ((idxX < n) && (idxY < n))
        c[idxY * n + idxX] = sum;
```

3.1.2 Función matrixMulOnDevice.

- ➢ Se reservaron d_a, d_b y d_c con cudaMalloc.
- ➢ Se copiaron a y b desde host a device con cudaMemcpy.
- ➢ Se lanzó el kernel y se recuperó c con cudaMemcpy.
- ➢ Se midieron tiempos y se calculó el rendimiento.

3.2 Ejercicio 2: Unified Memory.

3.2.1 Kernel dgemm_gpu_shared.

El kernel usado es exactamente el mismo que en el ejercicio anterior.

3.2.2 Función matrixMulOnDevice.

- ➢ Se lanzó el kernel directamente sobre punteros unificados.

3.2.3 Uso de cudaMallocManaged.

cudaMallocManaged(&a, size);

cudaMallocManaged(&b, size);

cudaMallocManaged(&c, size);

- ➢ No se requirieron transferencias explícitas.

4. Resultados.

Los programas se ejecutaron sobre matrices 1024 x 1024 con los siguientes resultados:

Modalidad	Tiempo (ms)	GFLOPS	maxAbsError	sumAbsError
Memoria compartida	4.152640	517.1370	0.0000	0.0000
Unified Memory	5.577728	385.0105	0.0000	0.0000

Ambas versiones alcanzaron un alto rendimiento, superando los 385 GFLOPS. La versión con memoria compartida fue más rápida, lo cual era esperado por su menor sobrecarga. Sin embargo, la versión con Unified Memory es notablemente más simple de programar y gestionar.

Ambas presentaron resultados numéricamente correctos, con errores absolutos iguales a cero al comparar C = A * B, siendo A la identidad y B una matriz con valores únicos por posición.

5. Validación de la ejecución.

Los ejercicios se ejecutaron en una GPU NVIDIA A100-PCIE-40GB (capacidad de cómputo 8.0). A continuación se muestran las capturas de pantalla de ambas ejecuciones:

```
[ule_formacion_10_10@cn6001 ~]$ nvcc -arch=sm_80 -Xcompiler -fopenmp -o dgemm_gpu_shared dgemm_gp
u_shared.cu
[ule_formacion_10_10@cn6001 ~]$ ./dgem_gpu_shared
bash: ./dgem_gpu_shared: No existe el fichero o el directorio
[ule_formacion_10_10@cn6001 ~]$ ./dgemm_gpu_shared


     Matrix-Multiplication
     =====================================

     Simple DGEMM implemantation on GPU
     Device name = NVIDIA A100-PCIE-40GB, with compute capability 8.0

     Matrix size 1024 x 1024Grid: 64, 64; block:16, 16

Kernel Execution Time: 4.152640 ms (dim C: 1024 * 1024)
This corresponds to: 517.1370 GFLOPS
maxAbsError: 0.0000, sumAbsError: 0.0000

Program terminated SUCCESSFULLY.
```

[Captura ejercicio 1 con memoria compartida]

```
[ule_formacion_10_10@cn6001 ~]$ nvcc -arch=sm_80 -Xcompiler -fopenmp -o dgemm_gpu_shared_uvm dgemm_gpu_shared_uvm.cu
[ule_formacion_10_10@cn6001 ~]$ ./dgemm_gpu_shared_uvm


     Matrix-Multiplication
     =====================================

     Simple DGEMM implemantation on GPU
     Device name = NVIDIA A100-PCIE-40GB, with compute capability 8.0

     Matrix size 1024 x 1024Grid: 64, 64; block:16, 16

Kernel Execution Time: 5.577728 ms (dim C: 1024 * 1024)
This corresponds to: 385.0105 GFLOPS
maxAbsError: 0.0000, sumAbsError: 0.0000

Program terminated SUCCESSFULLY.
```

[Captura ejercicio 2 con memoria unificada]

6. Conclusiones.

Este trabajo ha permitido explorar dos modelos de gestión de memoria en CUDA: compartida y unificada. El primero es más eficiente pero requiere mayor programación manual, mientras que el segundo simplifica el código sin sacrificar demasiado rendimiento.

Se ha reforzado el uso de estructuras paralelas, sincronización de hilos, y optimización de kernels para tareas intensivas como la multiplicación de matrices. En ambos casos se logró un cálculo correcto y eficiente en una arquitectura moderna.

Ambas soluciones son válidas y demuestran el potencial de CUDA para computación de alto rendimiento.