Instituto Politécnico Nacional

“Escuela Superior de Computo”

Integrantes:

- Diaz Ruiz Israel
- Ignacio Cortés Atzin Maxela
- Ríos Rivera Fernanda Anahí

Unidad de aprendizaje: Computo paralelo

Profesor: Luis Alberto Ibáñez Zamora

Grupo: 6BM2

“Reporte de práctica 1”

índice

Índice de figuras

Marco teórico

1. Arquitectura CUDA CUDA (Compute Unified Device Architecture) es una plataforma de computación paralela desarrollada por NVIDIA que permite a los desarrolladores usar la GPU para procesamiento general. En lugar de limitarse al procesamiento gráfico, CUDA permite ejecutar tareas computacionales paralelas de propósito general mediante kernels lanzados desde la CPU hacia la GPU. Una GPU CUDA está compuesta por múltiples Streaming Multiprocessors (SMs), que a su vez contienen núcleos CUDA (Scalar Processors), registros, unidades de control y memoria compartida. Los programas en CUDA se ejecutan en forma de miles de hilos que se agrupan en bloques y grids, facilitando la ejecución masiva de tareas paralelas.

2. Inspección de Hardware con PyCUDA El paquete PyCUDA permite el desarrollo de programas CUDA desde Python, incluyendo la inspección del hardware disponible. Mediante la clase pycuda.driver.Device, es posible obtener parámetros esenciales de la GPU como: Nombre del dispositivo: facilita la identificación del modelo específico de la tarjeta gráfica. Número de multiprocesadores (SMs): indica cuántas unidades de cómputo en paralelo posee la GPU. Hilos por bloque: define el límite superior de hilos que se pueden lanzar dentro de un solo bloque de ejecución. Tamaño máximo de bloque (max_block_dim_x, y, z): establece las dimensiones máximas permitidas para un bloque en cada eje. Memoria global (total_memory): espacio total disponible para el almacenamiento de datos accesible desde todos los hilos. Memoria compartida por bloque: espacio reservado por cada SM para que los hilos de un bloque puedan compartir datos de forma rápida. Hilos por multiprocesador: número máximo de hilos activos que un SM puede mantener simultáneamente. Tamaño de warp (warp size): grupo fijo de hilos (generalmente 32) que se ejecutan en paralelo bajo el modelo SIMT. Compute Capability: versión de arquitectura CUDA soportada por la GPU, que determina las características disponibles como el tamaño de memoria, tipos de instrucciones o soporte para Tensor Cores.

3. Importancia del Perfil de Hardware Conocer estos atributos es esencial para optimizar el desempeño de aplicaciones CUDA. Por ejemplo: El tamaño de warp impacta la eficiencia si hay divergencia entre hilos. El uso efectivo de memoria compartida puede reducir la latencia respecto a la memoria global. Saber cuántos hilos por bloque y por SM se permiten, facilita el diseño de kernels con alta ocupación y balance de carga.

Inspeccionar tu GPU

Utilizar la biblioteca **PyCUDA** para consultar e interpretar las propiedades fundamentales del hardware CUDA presente en la GPU del sistema, tales como número de multiprocesadores, hilos por bloque, tamaño de memoria compartida y global, tamaño de warp, entre otros.

Este ejercicio te permitirá comprender la estructura jerárquica de ejecución CUDA y cómo las capacidades del hardware influyen en la organización y rendimiento de los kernels.

Objetivos particulares

Implementar un script en Python con PyCUDA que imprima en consola:

- El **nombre del dispositivo GPU** detectado.
- La **cantidad de multiprocesadores (SMs)** disponibles.
- El **número máximo de hilos por bloque** permitido.
- Las **dimensiones máximas de bloque** (x, y, z).
- La **memoria global total** disponible.
- El **tamaño máximo de memoria compartida por bloque**.
- El **número de hilos por multiprocesador**.
- El **tamaño de warp**.
- La **versión de compute capability** de la GPU.

Ejecutar el script y tomar una captura de pantalla de la salida para documentar tu GPU.

```
[ ] import pycuda.driver as cuda
    import pycuda.autoinit

    device = cuda.Device(0)  # Asume que tienes al menos una GPU

    print(f" Nombre de GPU: {device.name()}")
    print(f"  Número de multiprocesadores (SMs): {device.get_attribute(cuda.device_attribute.MULTIPROCESSOR_COUNT)}")
    print(f"  Hilos por bloque: {device.get_attribute(cuda.device_attribute.MAX_THREADS_PER_BLOCK)}")
    print(f"  Tamaño máximo de bloque (x,y,z): {device.max_block_dim_x}, {device.max_block_dim_y}, {device.max_block_dim_z}")
    print(f"  Tamaño total de memoria global: {device.total_memory() / (1024**2):.2f} MB")
    print(f"  Memoria compartida por bloque: {device.get_attribute(cuda.device_attribute.SHARED_MEMORY_PER_BLOCK)} bytes")
    print(f"  Hilos por multiprocesador: {device.get_attribute(cuda.device_attribute.MAX_THREADS_PER_MULTIPROCESSOR)}")
    print(f"  Warp size (tamaño de warp): {device.get_attribute(cuda.device_attribute.WARP_SIZE)}")
    print(f"  Versión de Compute Capability: {device.compute_capability()}")
```

```
Nombre de GPU: Tesla T4
 Número de multiprocesadores (SMs): 40
 Hilos por bloque: 1024
 Tamaño máximo de bloque (x,y,z): 1024, 1024, 64
 Tamaño total de memoria global: 15095.06 MB
 Memoria compartida por bloque: 49152 bytes
 Hilos por multiprocesador: 1024
 Warp size (tamaño de warp): 32
 Versión de Compute Capability: (7, 5)
```

Figura 1 Ejecución del programa

Interpretar los resultados obtenidos, por ejemplo:

1.- ¿Cuántos bloques puede ejecutar tu GPU simultáneamente?

La GPU tiene 40 multiprocesadores (SMs) y cada SM puede ejecutar hasta 1024 hilos simultáneamente. Pero el número de bloques simultáneos depende de varios factores:

- Hilos por bloque
- Uso de memoria compartida
- Registros usados por hilo
- Máximo teórico (simplificado, sin considerar limitaciones de registros o recursos):

Si se usaran bloques de 256 hilos → cada SM puede manejar 1024 / 256 = 4 bloques simultáneamente.

Entonces:

40 SMs × 4 bloques = 160 bloques simultáneamente

2.- ¿Qué tan grande puede ser un kernel en cuanto a hilos?

La cantidad total de hilos por bloque está limitada a:

1024 hilos por bloque (máximo absoluto)

Además, por dimensión:

- Dim X ≤ 1024
- Dim Y ≤ 1024
- Dim Z ≤ 64
- X × Y × Z ≤ 1024

Ejemplos válidos:

- (1024, 1, 1)
- (32, 32, 1) = 1024 hilos
- (16, 16, 4) = 1024 hilos

Así que el tamaño máximo de un kernel en cuanto a hilos por bloque es 1024.

3.- ¿Cómo afectaría el tamaño del warp a tu código CUDA?

El tamaño de warp es 32 hilos.

Implicaciones:

La GPU ejecuta hilos en grupos de 32 (warp). Todos los hilos en un warp ejecutan la misma instrucción al mismo tiempo (SIMT).

Si el número de hilos por bloque no es múltiplo de 32, habrá hilos "inactivos" que consumen recursos sin contribuir al cómputo (ineficiencia).

Colab

https://colab.research.google.com/drive/1SjulWCgykm3yZfTInlbailyd2aT25xB7?usp=sharing

Referencias

- Klöckner, A. (2010, Septiembre). PyCUDA: Even simpler GPU programming with Python. In Proceedings of the GPU Technology Conference, Berkeley, CA, USA (pp. 20-23).
- Pérez Sánchez, J. A. (2013). Desarrollo CUDA en Java y Python.