Universidad de Nacional de San Agustín
Facultad de Producción y Servicios
Escuela de Ciencia de la Computación

Lab - Resumen capítulo 1

Integrantes: Kelvin Paul Pucho Zevallos
Docente: Alvaro Henry Mamani Aliaga

Fecha de entrega: 17 de diciembre de 2022
Arequipa, Perú

Índice de Contenidos

Índice de Figuras

1. HETEROGENEOUS PARALLEL COMPUTING

Hace uso de una lógica de control sofisticada para permitir instrucciones de un solo subproceso para ejecutar en paralelo o incluso fuera de su orden secuencial manteniendo la apariencia de ejecución secuencial

y por medio de las grandes memorias caché son proporcionado para reducir las latencias de instrucción y acceso a datos de grandes aplicaciones complejas. Los microprocesadores multinúcleo de uso general de gama alta suelen tener ocho o más núcleos de procesador grandes y muchos megabytes de almacenamiento en chip memorias caché diseñadas para ofrecer un sólido rendimiento de código secuencial.

Una GPU debe ser capaz de mover cantidades extremadamente grandes de datos dentro y fuera de su principal Dynamic Random Access Memory (DRAM) debido a los requisitos de búfer de cuadros de gráficos.

Para programas que tienen uno o muy pocos subprocesos, las CPU con las latencias de operación más bajas pueden lograr un rendimiento mucho mayor que las GPU.
Un programa tiene una gran cantidad de subprocesos, GPU con mayor rendimiento de ejecución

Según la historia relatada en el libro el modelo ming, presentado por NVIDIA en 2007, está diseñado para admitir CPU-GPU conjuntas. La demanda de soporte para la ejecución conjunta CPU-GPU es se refleja aún más en modelos de programación más recientes como OpenCL, OpenACC y C++AMP

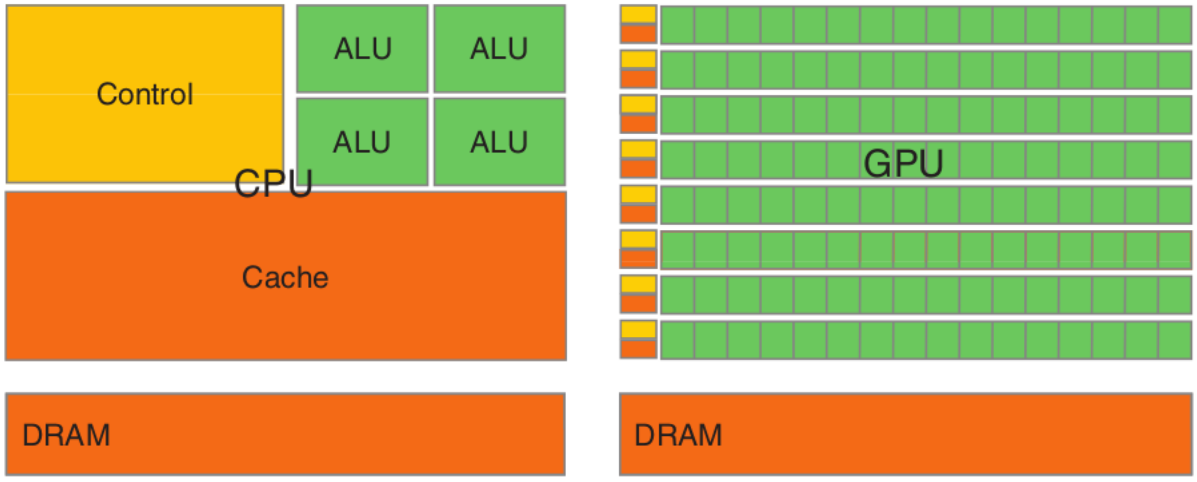

Figura 1: Comparacion CPU y GPU

2. ARCHITECTURE OF A MODERN GPU

La arquitectura comun se muestra en la siguiente figura:

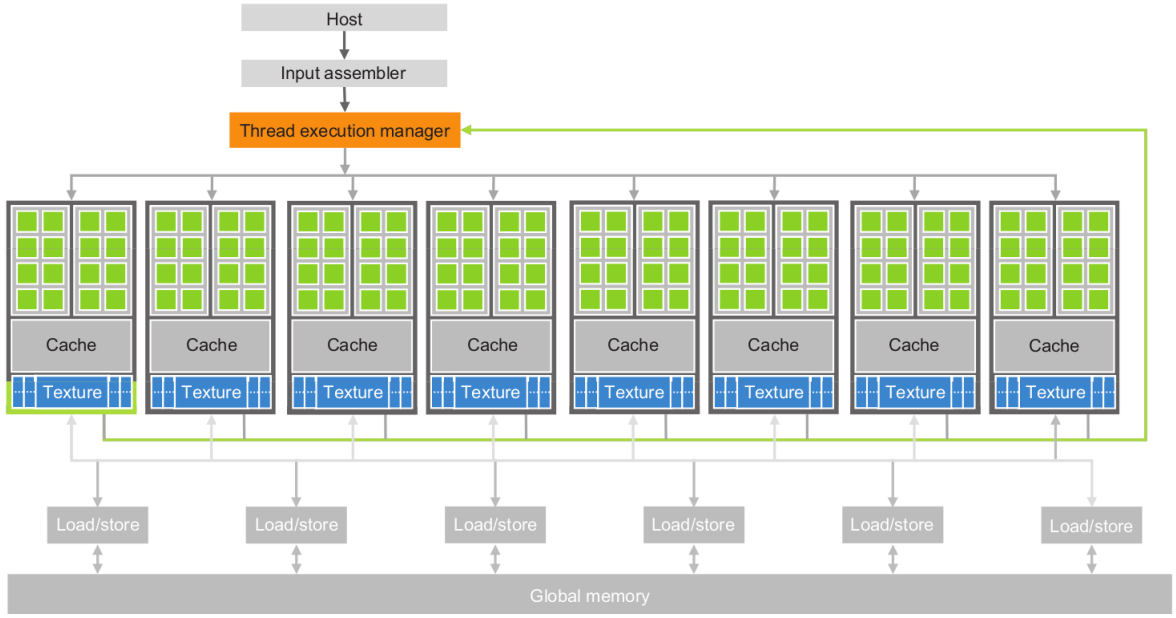

Figura 2: CUDA-capable GPU

Esta organizado dentro de una arreglo de ligeros threading de multiprocesos de flujo (SM) donde cada SM tiene un numero de SP (Streaming processors) que comparten la logica e instrucciones del cache. Una serie es el G80 introdujo la arquitectura CUDA y tenía 86,4 GB/s de ancho de banda de la memoria, además de un enlace de comunicación a la lógica del núcleo de la CPU a través de una interfaz PCI-Express Generation 2 . Sobre PCI-E Gen2, una aplicación CUDA puede transferir datos desde la memoria del sistema al memoria global a 4 GB/s y, al mismo tiempo, cargar datos de nuevo en la memoria del sistema a 4 GB/s. También se espera que el ancho de banda de comunicación crezca a medida que crezca el ancho de banda del bus de la CPU de la memoria del sistema en el futuro.

3. WHY MORE SPEED OR PARALLELISM?

La principal motivación para la programación paralela masiva es que las aplicaciones disfruten de un aumento continuo de la velocidad en las futuras generaciones de hardware. Muchas aplicaciones continuarán exigir mayor velocidad y hoy en día parecen funcionar lo suficientemente rápido. En el mundo de hoy existencia lo que se denomina super-aplicaciones. Por ejemplo en la comunidad de investigadores biólogos se centra mas y mas en el nivel molecular.

En este ambito ellos utilizan los microscopios para verificar la moleculas al detalles pero estos tambien tienes sus limitaciones establecidas con el instrumento y lo que se quiere es la reaccion al tiempo de respuesta tolerable.

En el futuro las tecnologías requieren un buen proceso en paralelo para utilizar imágenes en 3D y visualizaciones ya que la síntesis y la alta resolución demandan mas poder computacional.
Las mejoras tambien se utilizan en las interfaces mediante velocidades informáticas mejoradas.

moderno inteligente los usuarios de teléfonos disfrutan de una interfaz más natural con pantallas táctiles de alta resolución que rivalizar con los televisores de pantalla grande. Si el jugador automóvil chocó con obstáculos, el comportamiento del automóvil no cambió para reflejar el daño. Solo cambia la puntuación del juego, y la puntuación determina el ganador. Podemos esperar ver más de estos efectos realistas en el futuro: las colisiones dañarán sus ruedas y la experiencia de conducción del jugador será mucho más realista.

4. SPEEDING UP REAL APPLICATIONS

¿Qué tipo de aceleración podemos esperar al paralelizar una aplicación? Depende de la parte de la aplicación que se puede paralelizar. De hecho, incluso una cantidad infinita de aceleración en la parte paralela solo puede reducir el tiempo de ejecución en un 30 %, logrando no más de aceleración 1.43X. El hecho de que el nivel de aceleración que se puede lograr a través de la ejecución en paralelo pueda verse severamente limitado por la parte paralelizable de la aplicación se conoce como Ley de Amdahl. Esto le da a toda la aplicación una aceleración de 50X. Los investigadores han logrado aceleraciones de más de 100X para algunas aplicaciones. En la práctica, la paralelización directa de aplicaciones a menudo satura el ancho de banda de la memoria , lo que resulta en una aceleración de solo 10 veces. Sin embargo, se debe optimizar aún más el código para evitar limitaciones como la capacidad limitada de memoria en el chip.

La mayoría de las aplicaciones tienen partes que la CPU puede ejecutar mucho mejor. Las CPU son bastante buenas con estas porciones. La buena noticia es que estas partes, aunque pueden ocupar una gran parte del código, tienden a representar solo una pequeña parte del tiempo de ejecución de las superaplicaciones. El resto es lo que llamamos las porciones de «carne de durazno».

Estas porciones son fáciles de paralelizar, al igual que algunas de las primeras aplicaciones gráficas. La programación paralela en sistemas informáticos heterogéneos puede mejorar drásticamente la velocidad de estas aplicaciones. Como se ilustra en la figura 1.3, las primeras GPGPU cubren solo una pequeña porción de la sección de carnes, que es análoga a una pequeña porción de las aplicaciones más emocionantes. Como veremos, el modelo de programación CUDA está diseñado para cubrir una sección mucho más grande de las porciones de carne de durazno de aplicaciones emocionantes.

De hecho, como veremos en el Capítulo 20, Más información sobre computación CUDA y GPU, estos modelos de programación y su hardware subyacente siguen evolucionando a un ritmo acelerado para permitir la paralelización eficiente de secciones de aplicaciones aún más grandes.

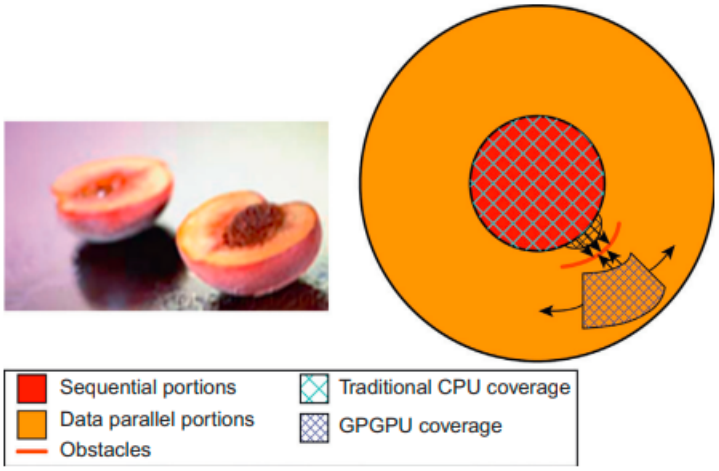

Figura 3: Cobertura de porciones de aplicaciones secuenciales y paralelas.

5. CHALLENGES IN PARALLEL PROGRAMMING

Alguien dijo una vez que si no te importa el rendimiento, la programación paralela es muy fácil. Este libro aborda varios desafíos para lograr un alto rendimiento en la programación paralela. En segundo lugar, la velocidad de ejecución de muchas aplicaciones está limitada por la velocidad de acceso a la memoria. Nos referimos a estas aplicaciones como vinculadas a la memoria, a diferencia de las vinculadas a la computación, que están limitadas por la cantidad de instrucciones realizadas por byte de datos.

Lograr una ejecución paralela de alto rendimiento en aplicaciones vinculadas a la memoria a menudo requiere métodos novedosos para mejorar la velocidad de acceso a la memoria.

En tercer lugar, la velocidad de ejecución de los programas paralelos suele ser más sensible a las características de los datos de entrada que sus homólogos secuenciales. Muchas aplicaciones del mundo real necesitan manejar entradas con características muy diversas, como erráticos. O velocidades de datos impredecibles, y velocidades de datos muy altas. El rendimiento de los programas paralelos a veces puede variar drásticamente con estas características.

6. PARALLEL PROGRAMMING LANGUAGES AND MODELS

Los más utilizados son la interfaz de paso de mensajes para computación de clúster escalable y OpenMP para sistemas multiprocesador de memoria compartida. Una implementación de OpenMP consta de un compilador y un tiempo de ejecución. Un programador especifica directivas y pragmas sobre un bucle al compilador OpenMP. Con estas directivas y pragmas, los compiladores de OpenMP generan código paralelo.

OpenMP fue diseñado originalmente para la ejecución de la CPU.

La principal ventaja de OpenACC es que proporciona automatización del compilador y soporte de tiempo de ejecución para abstraer muchos detalles de programación paralela de los programadores. Es por eso que enseñamos programación OpenACC en el Capítulo 19, Programación paralela con OpenACC. Sin embargo, la programación efectiva en OpenACC aún requiere que los programadores entiendan todos los conceptos detallados de programación paralela involucrados. Debido a que CUDA brinda a los programadores un control explícito de estos detalles de programación paralela, es un excelente vehículo de aprendizaje incluso para alguien a quien le gustaría usar OpenMP y OpenACC como su principal interfaz de programación.

El libro indica que aquellos que están familiarizados con OpenCL y CUDA saben que existe una notable similitud entre los conceptos y características clave de OpenCL y CUDA. Es decir, un programador de CUDA puede aprender a programar OpenCL con un esfuerzo mínimo. Más importante aún, prácticamente todas las técnicas aprendidas usando CU

7. OVERARCHING GOALS

El objetivo principal ensenar a cómo programar masivamente en paralelo procesadores para lograr un alto rendimiento, y nuestro enfoque no requerirá una gran mucha experiencia en hardware. Por lo tanto, vamos a dedicar muchas páginas a las técnicas para desarrollar programas paralelos de alto rendimiento. En particular, nos centraremos en técnicas de pensamiento computacional que le permiten pensar en los problemas de manera que sean susceptibles de alto rendimiento computación paralela. La programación paralela de alto rendimiento en la mayoría de los procesadores requerirá algunos conocimiento de cómo funciona el hardware.

Como parte de nuestras discusiones sobre técnicas de programación paralela de alto rendimiento.

8. ORGANIZATION OF THE BOOK

En el capitulo 2 la programación en C. Primero presenta CUDA C como una extensión simple y pequeña de C que admite la computación conjunta CPU/GPU heterogénea y el modelo de programación paralela de datos múltiples de programa único ampliamente utilizado.
El Capítulo 3, Ejecución paralela escalable, presenta más detalles del modelo de ejecución paralela de CUDA. Brinda suficiente información sobre la creación, organización, vinculación de recursos, vinculación de datos y programación de subprocesos para permitir que el lector implemente Cálculo sofisticado utilizando CUDA C y razonamiento sobre el comportamiento de rendimiento de su código CUDA.

En el capitulo 4 Presentamos las características del lenguaje CUDA que asignan y utiliza estos recuerdos. El uso apropiado de estas memorias puede mejorar drásticamente la rendimiento de acceso a datos y ayuda a aliviar la congestión de tráfico en la memoria sistema.

En el capitulo 5, se hace consideraciones en el hardware CUDA actual. En particular, da más detalles en patrones deseables de ejecución de subprocesos, accesos a datos de memoria y asignación de recursos.

En el Capítulo 6, se hace consideraciones numéricas, presenta los conceptos de formato, preci-

sión y exactitud de números de punto flotante IEEE-754. Muestra por qué diferentes paralelos los arreglos de ejecución pueden dar como resultado diferentes valores de salida
Mientras que en los siguientes capítulos como son capítulos 7, Patrones paralelos: convolución, capítulo 8, Patrones paralelos: prefijo sum, capítulo 9, Patrones paralelos—cálculo de histogramas en paralelo, capítulo 10, Patrones paralelos: cálculo matricial disperso, capítulo 11, Patrones paralelos: combinación ordenar, capítulo 12, Patrones paralelos: búsqueda de gráficos, presentar seis paralelos importantes patrones de cálculo que brindan a los lectores una mayor comprensión de la programación paralela técnicas y mecanismos de ejecución en paralelo. Capítulo 8, Paralelo patrones: suma de prefijos, presenta un árbol de reducción y suma de prefijos, o escaneo, un importante patrón de computación paralela que convierte la computación secuencial en paralela cálculo. conjuntos También cubrimos la operación de fusión, un patrón ampliamente utilizado en dividir y coincidir. estrategias de reparto del trabajo. El Capítulo 10, Patrones paralelos: cálculo matricial disperso, presenta el cálculo matricial disperso, un patrón utilizado para procesar conjuntos de datos Este capítulo introduce al lector a los conceptos de reorganización de datos para acceso paralelo más eficiente: compresión de datos, relleno, clasificación, transposición y regularización Capítulo 11, Patrones paralelos: clasificación por fusión, introduce la clasificación por fusión, e identificación y organización de datos de entrada dinámicos.