UNIVERSIDAD DE CARABOBO

FACULTAD EXPERIMENTAL DE CIENCIAS Y TECNOLOGÍA

DEPARTAMENTO DE COMPUTACIÓN

ARQUITECTURA DEL COMPUTADOR

Paralelización

Profesor

Jose Canache

Estudiantes

Alarcón Z. Jeanmarco J. C.I: 27.117.926

Alejandro Cerpa C.I: 30334870

Naguanagua, Noviembre 2023

La paralelización se refiere a la capacidad de los dispositivos y sistemas digitales de realizar diversas tareas a la vez, esto normalmente se encuentra presente en diversas características del procesador de un sistema digital, aunque a veces también puede ser visto por alguna característica peculiar del procesador no dedicada a la función de la paralelización en sí como la lectura y escritura de audio y video, antes de hablar de ello se destaca el comienzo de la paralelización y sus razones de existencia. A partir de la creación del procesador Pentium la curva de aumento cuyo tope de procesamiento llegó a los a picos en el año 2004 con la creación del pentium 4 de 3.6 GHZ, después de esto el aumento en la velocidad de procesamiento de los procesadores se hace disminuido considerablemente ya que ha mayor velocidad se produce más calor en el procesador, la solución de la industria fue los microprocesadores multi-hilo de varios núcleos que permitían ejecutar varias tareas a la vez dividiendo la carga de trabajo entre varios núcleos, sin embargo esta característica se ve opacada muchas veces gracias a la falta de soporte para la programación paralela y este “nuevo” paradigma de programación, otros sistemas sistemas como los de procesamiento digital ya incorporan antes programación paralela incluso sin microprocesadores multi-hilos de varios núcleos, esto se debe a la particularidad de los sistemas digitales que trabajan en medidas más grandes que 1 byte, los sistemas de procesamiento gráfico por ejemplo al usar 1 byte por píxel por pantalla en un sistema cuya unidad principal es 4 bytes, pueden manipular y mostrar 4 píxeles por operación creando el primer antecedente de la programación paralela, ahora llevando esto a lugares más amplios y complicados hoy en día se han creado tecnologias que permiten mediante nuevas mecanicas la programacion paralela usando los diferentes nucleos del microprocesador, a continuacion se hablara un poco de algunas de estas tecnologias (que destacan más en la parte de software).

Tecnologías para la ayuda de la programación paralela

La interfaz de paso de mensajes o MPI:

MPI, también conocido como la Interfaz de Paso de Mensajes, es un estándar que dicta las reglas para las funciones en una biblioteca de paso de mensajes. Aunque no es un estándar oficial reconocido por organizaciones como ISO o IEEE, MPI es ampliamente aceptado y utilizado debido a su código abierto.

Esta biblioteca se utiliza en programas que se benefician de la existencia de múltiples procesadores y estos programas normalmente están escritos en lenguajes como C, C++, Fortran y Ada. La creación de MPI fue un esfuerzo colaborativo en un foro abierto con la participación de más de 40 organizaciones, que incluyen proveedores de hardware, investigadores, académicos, desarrolladores de software y usuarios.

Lo que distingue a MPI de otras bibliotecas de paso de mensajes es su portabilidad y eficiencia. Los programas que utilizan MPI pueden funcionar en casi cualquier arquitectura de memoria distribuida. Además, cada implementación de la biblioteca se optimiza para el hardware específico en el que se ejecuta, lo que mejora su eficiencia.

Con MPI, antes de que se inicie la ejecución del programa, se determina el número de procesos requeridos, y no se generan procesos adicionales durante la ejecución de la aplicación. Se asigna una variable conocida como rango a cada proceso, que se utiliza para identificar a cada proceso en un rango de 0 a p-1, donde p es el número total de procesos. Esta variable de rango controla la ejecución del programa al determinar qué proceso ejecuta qué segmento de código.

En MPI, se define un comunicador como un grupo de procesos que pueden intercambiar mensajes entre sí. El comunicador básico, denominado MPI_COMM_WORLD, se define mediante una macro en el lenguaje C. Este comunicador agrupa a todos los procesos que están activos durante la ejecución de una aplicación.

OpenMp:

OpenMP es una API diseñada para facilitar la programación multiproceso de memoria compartida en una variedad de plataformas, incluyendo Unix y Microsoft Windows. Esta API, que fue definida en colaboración por proveedores de hardware y software, permite agregar concurrencia a los programas escritos en C, C++ y Fortran, utilizando el modelo de ejecución fork-join.

El modelo de programación de OpenMP puede aplicarse tanto en memoria compartida como en memoria distribuida en multiprocesadores. Este modelo se basa en la disponibilidad de múltiples hilos de ejecución concurrentes que pueden acceder a variables en áreas de memoria a las que cada uno de ellos tiene acceso. Esto simplifica la programación al evitar la necesidad de intercambio explícito de datos entre los diferentes procesadores.

OpenMP es un modelo de programación portátil y escalable que proporciona a los programadores una interfaz sencilla y flexible para el desarrollo de aplicaciones paralelas. Estas aplicaciones pueden ejecutarse en una variedad de plataformas, desde computadoras de escritorio hasta supercomputadoras. Además, una aplicación construida con un modelo de programación paralela híbrido puede ejecutarse en un clúster de computadoras utilizando OpenMP y MPI, o a través de las extensiones de OpenMP para sistemas de memoria distribuida.

OpenMP es un sistema de programación basado en la memoria compartida y en

directivas de compilación. En este sistema, el programador introduce directivas en el código secuencial para orientar al compilador en la paralelización del código. Además, se incluyen algunas llamadas específicas a funciones. Cuando se introduce una directiva de compilador OpenMP en el código, se incorpora automáticamente una sincronización en todo el bloque de código. Esto significa que el bloque de código se marca como paralelo y se generan hilos según las características especificadas por la directiva. Al finalizar el bloque, existe una barrera para sincronizar los diferentes hilos, a menos que se especifique lo contrario con la directiva nowait. Este patrón de ejecución se conoce como fork-join.

Cuda:

CUDA, que significa Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo), es una plataforma de computación paralela desarrollada por Nvidia. Esta plataforma incluye un compilador y un conjunto de herramientas de desarrollo que permiten a los programadores codificar algoritmos en las GPU de Nvidia utilizando una variante del lenguaje de programación C, conocida como CUDA C.

Además de C/C++, se pueden utilizar otros lenguajes de programación como Python, Fortran, Julia y Java a través de wrappers.

CUDA es compatible con todas las GPU Nvidia de la serie G8X en adelante, lo que incluye las líneas GeForce, Quadro, ION y Tesla. Nvidia asegura que los programas desarrollados para la serie GeForce 8 funcionarán sin modificaciones en todas las futuras tarjetas Nvidia, gracias a la compatibilidad binaria proporcionada por el conjunto de instrucciones PTX (Parallel Thread Execution).

La plataforma CUDA busca aprovechar las ventajas de las GPU sobre las CPU de propósito general al utilizar el paralelismo que ofrecen sus múltiples núcleos. Esto permite el lanzamiento de un gran número de hilos simultáneos. Por lo tanto, si una aplicación está diseñada utilizando numerosos hilos que realizan tareas independientes (que es lo que hacen las GPU al procesar gráficos, su tarea natural), una GPU puede ofrecer un alto rendimiento en diversas áreas, desde la biología computacional hasta la criptografía.

El modelo de programación CUDA se ha diseñado para permitir que las aplicaciones escalen su paralelismo de manera fluida para aumentar el número de núcleos de procesamiento. Este enfoque se basa en tres aspectos fundamentales: la jerarquía de grupos de hilos, las memorias compartidas y las barreras de sincronización.

La estructura utilizada en este modelo se define por una cuadrícula, que contiene bloques de hilos compuestos por un máximo de 1024 hilos diferentes. Cada hilo en un bloque se identifica con un identificador único, al que se accede mediante la variable threadIdx. Esta variable es esencial para distribuir el trabajo

entre diferentes hilos. threadIdx tiene tres componentes (x, y, z), que corresponden a las dimensiones de los bloques de hilos. De esta manera, cada elemento de una matriz podría ser procesado por su correspondiente en un bloque de hilos de dos dimensiones.

Al igual que los hilos, los bloques se identifican mediante blockIdx (en sus componentes x, y, z). Otras funciones útiles incluyen blockDim, que permite acceder al tamaño del bloque, y gridDim, que permite acceder a la forma de la cuadrícula.

Finalmente, CUDA busca maximizar el uso del paralelismo extenso y la alta capacidad de memoria de las GPU para aplicaciones que requieren una gran cantidad de cálculos en comparación con los accesos frecuentes a la memoria principal, que podrían generar un cuello de botella.