Tema 2. Modelos de computadores paralelos.

1. Plataformas secuenciales

John Von Neumann sentó las bases de la computación moderna a la que pertenecen los ordenadores secuenciales, también denominados computadores serie. Este modelo se basa en la Unidad Central de Procesamiento o CPU (*Central Processing Unit*), en la memoria donde se almacenan datos e instrucciones. En esta arquitectura las instrucciones se ejecutan en serie que tratan con un única secuencia de datos.

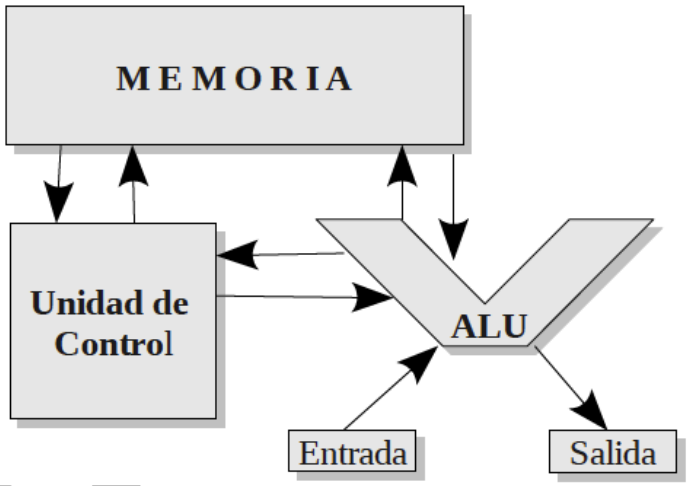

Figura 1. Arquitectura de Von Neumann (CPU y memoria)

Esta arquitectura presenta dos factores limitadores a tener en cuenta: la velocidad a la que se ejecutan las instrucciones y la velocidad de transferencia de datos e instrucciones entre la memoria y la CPU. En el primer factor están implicados los componentes y la tecnología electrónica (frecuencia de reloj, etc.). No obstante, desde los años noventa se han desarrollado nuevas estrategias para aumentar el rendimiento. La más conocida es la segmentación que consiste en ejecutar varias partes de un número determinado de instrucciones al mismo tiempo en varias unidades funcionales de cálculo (multiplicadores, sumadores, unidades de carga y almacenamiento,...). Esto se consigue dividiendo las partes de ejecución de una instrucción en etapas,

alcanzando un rendimiento bastante mejorado con respecto a una máquina no segmentada o secuencial. En la figura 2 podemos ver el funcionamiento de una CPU segmentada.

Instrucción	Ciclo de reloj								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
i	IF	ID	EX	MEM	**WB**				
i+1		IF	ID	EX	**MEM**	WB			
i+2			IF	ID	**EX**	MEM	WB		
i+3				IF	**ID**	EX	MEM	WB	
i+4					**IF**	ID	EX	MEM	WB

Figura 2. Segmentación RISC

Donde la etapa de ejecución IF es la búsqueda de instrucción en la caché de instrucciones o en la memoria principal, ID es la decodificación de la instrucción y lectura de los registros, EX es la etapa ejecución aritmético lógica, MEM es la lectura y escritura en memoria, y finalmente WB escribe el resultado en los registros. Como el lector puede apreciar, la ventaja de esta estrategia radica en que es posible adelantar la ejecución de una instrucción en cada ciclo de reloj. En consecuencia en cada ciclo de reloj se finaliza una instrucción y el incremento del rendimiento, sin tener en cuenta los riesgos de la segmentación, es de un factor de cinco con respecto a una máquina que ejecuta las instrucciones en serie. Por contra, aunque el incremento de la productividad es muy alto con segmentación, el tiempo de CPU que se le dedica a una instrucción individual es superior al de su homólogo secuencial. Esto es debido a:

- Los cerrojos que hay que colocar en la circuitería para separar la información entre etapas.
- La duración de las etapas es similar y viene determinada por la etapa más larga.
- Los riesgos estructurales, de datos y de control que introducen detenciones en el cauce de ejecución.

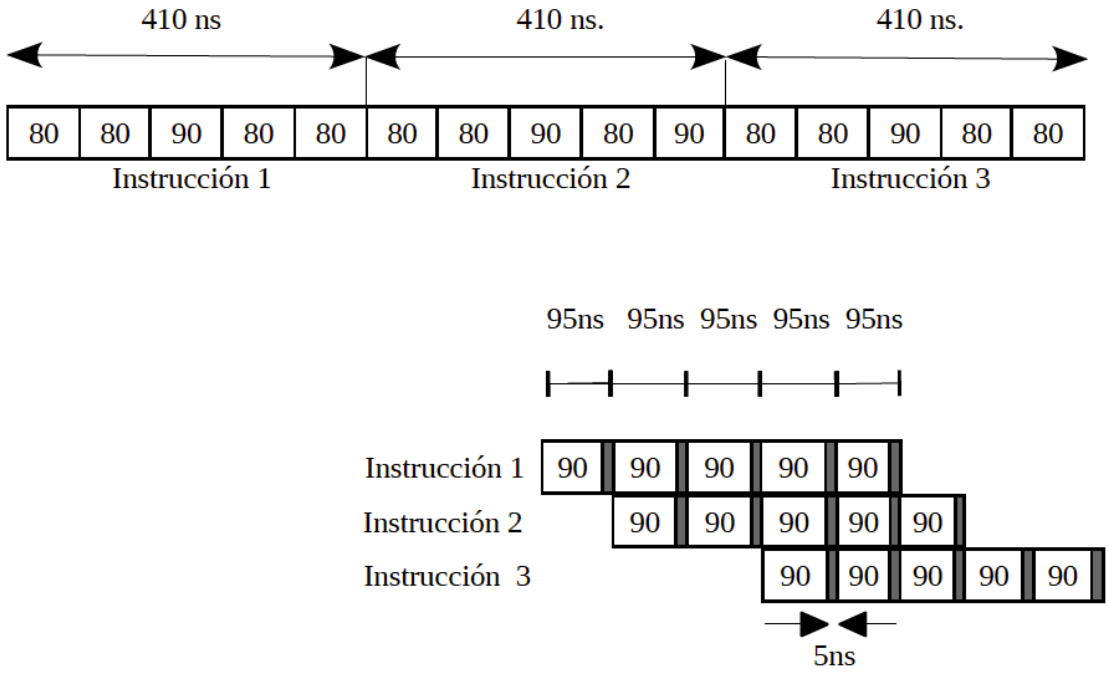

Figura 3. Ve rsión No segmentada (arriba) y segmentada de tres instrucciones (abajo)

Como puede observarse en la figura 3, la instrucción no segmentada consume 410 nanosegundos, mientras que su equivalente segmentada consume 475 nanosegundos, a causa de la etapa más lenta que es 90 ns, en este caso suponiendo una segmentación ideal y un retardo de 5 ns. Por los cerrojos de cada etapa segmentada. Esto implica un aumento de productividad que no siempre está asociado a un incremento de velocidad de ejecución. Con respecto al segundo problema de la segmentación, éste admitiría dos soluciones que radican principalmente en acortar el tiempo de acceso a memoria. En primer lugar se utiliza memoria entrelazada (memoria interleaving) Esta estrategia arquitectónica permite dividir la memoria en bloques, cada uno con acceso independiente, tal y como puede verse en la siguiente figura 4.

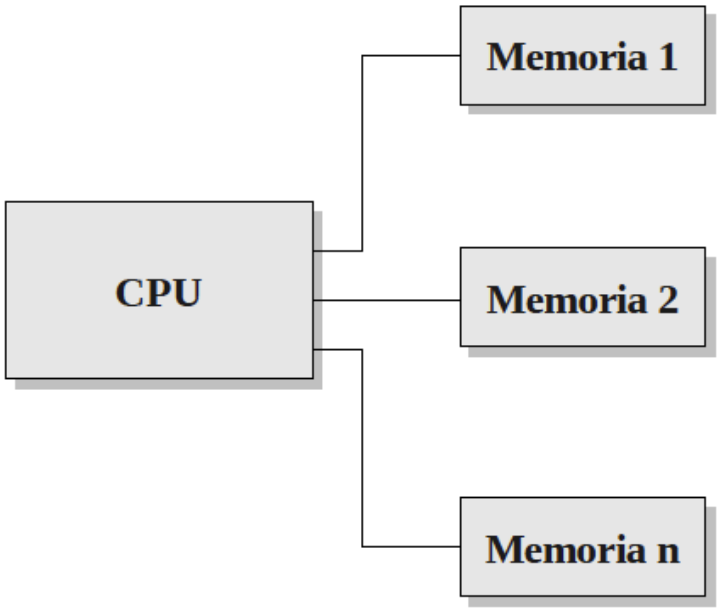

Figura 4. Memory interleaving

Otra estrategia para reducir el tiempo de acceso a memoria es el uso de memorias cachés. Con esta memoria se consigue mayor velocidad de intercambio en un espacio de direcciones más pequeño, en detrimento del coste de la tecnología. Con este técnica se consigue el objetivo del principio de localidad, en la caben dos posibles casos: La localidad espacial que consiste en el hecho de que la instrucción siguiente a la que se está ejecutando está en la cercanía del bloque en uso, La localidad temporal que consiste en que los datos o instrucciones que serán accedidos a continuación serán los que han sido accedidos recientemente.

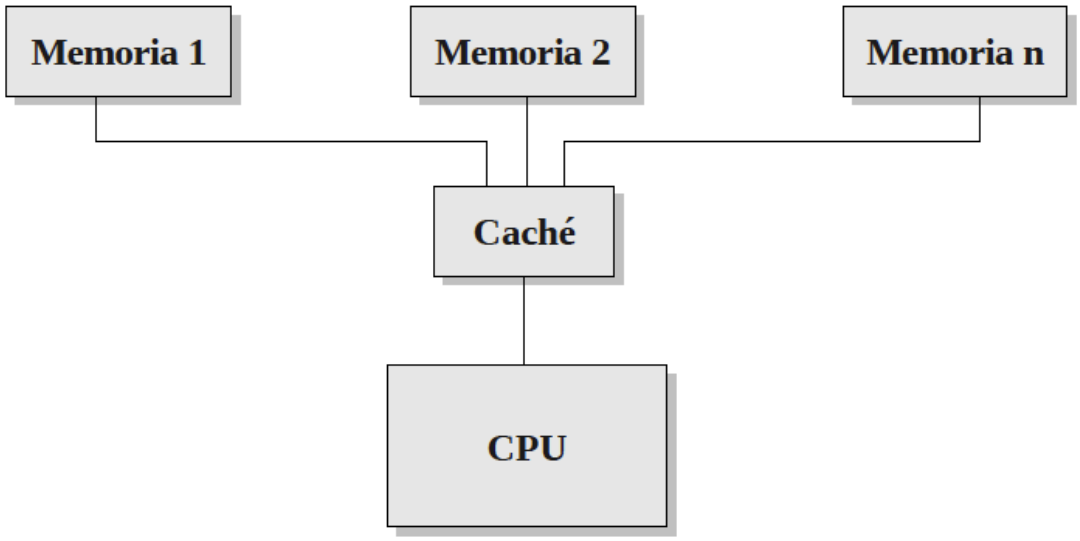

Figura 5. Memory interleaving y caché.

Los procesadores segmentados son útiles para aplicaciones científicas y de ingeniería. Un procesador segmentado, en término generales, siempre va a utilizar una memoria caché para reducir la latencia de acceso a memoria, es decir, evitar que las instrucciones de acceso a memoria sean muy lentas. No obstante, en las aplicaciones científicas, los datos de grandes dimensiones son accedidos con baja localidad, y de esta manera el rendimiento que se consigue en la jerarquía de memoria es bastante pobre y con un efecto resultante de mal rendimiento en el uso de la caché. Una posible solución alternativa sería prescindir de las cachés, siempre y cuando fuera posible determinar qué patrones de acceso son los que definen el acceso a memoria y realizar la segmentación en consecuencia.

2. Taxonomía de Flynn

Para clasificar las arquitecturas paralelas existen diferentes clasificaciones; aunque una de las más utilizadas es la que definió el ingeniero Michael J. Flynn en 1972 y es conocida como la taxonomía de Flynn. Esta clasificación distingue arquitecturas multiprocesadores de acuerdo a dos dimensiones independientes: datos e instrucciones, donde cada una de estas dimensiones puede tener sólo dos posibles estados: **simple** (single) y **multiple** (múltiple).
En la Figura 6 se muestra esta clasificación.

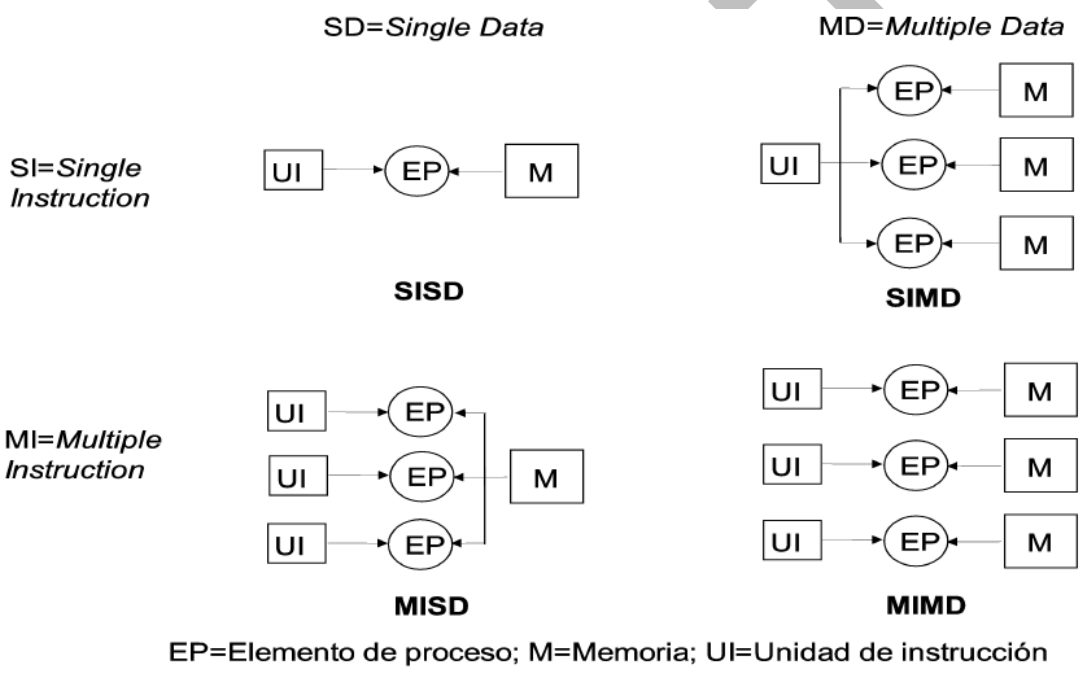

Figura 6. Taxonomia de Flynn

En primer lugar se sitúa la categoría SISD, Single Data, Single Instruction. Dentro de esta categoría se encuadran la mayoría de los computadores secuenciales disponibles actualmente: PC, Estaciones de Trabajo y antiguos Mainframe. Las instrucciones se ejecutan secuencialmente, pero pueden utilizar segmentación y más de una unidad de cálculo. Siempre harán uso de una única unidad de control que gobierna todas las instrucciones. Es traducido como instrucción única, datos únicos. La secuenciación de las instrucciones es producida por cada pulso de reloj, haciendo que la CPU obtenga un dato o un instrucción de memoria para su ejecución secuencial.

En segundo lugar tenemos la categoría SIMD (Single Intructon Multiple Data). Es un tipo de arquitectura paralela donde todas las unidades ejecutan la misma instrucción en un ciclo de reloj y cada unidad opera sobre una parte diferente de los datos. Esta arquitectura es especialmente apropiada para problemas de procesamiento de imágenes y gráficos, como es el caso de CUDA, que veremos en esta asignatura. En esta clasificación se encuadran los procesadores matriciales en los que existen más de una unidad de procesamiento trabajando sobre flujos de datos distintos, pero ejecutando la misma instrucción proporcionada por una única unidad de control.

La tercera categoría es MISD (Multiple Instruction Single Data). Esta organización se caracteriza por el hecho de la existencia de varias unidades de procesamiento cada una ejecutando una instrucción diferente pero sobre el mismo dato. Es una arquitectura teórica y no se conoce ninguna materialización de esta categoría.

Finalmente tenemos la organización MIMD (Multiple Instruction Multple Data). Dentro de esta categoría se encuadran la mayoría de sistemas multiprocesadores y multicomputadores, donde cada programa se ejecuta en cada unidad de procesamiento. Esta arquitectura implica una coordinación entre procesadores, puesto que todos los flujos de memoria se obtienen de un espacio compartido para todos ellos. Los procesadores en esta arquitectura trabajan de manera independiente. Por ejemplo, dos procesadores podrían estar realizando la Transformada Rápida de Fourier, mientras otro realiza la salida gráfica. Esto es lo que habitualmente se denomina descomposición de control. Aunque no debemos pasar por alto la descomposición de datos, en la que los datos del problema se reparten entre los distintos procesadores y todos ejecutan el mismo programa sobre la zona de datos que se le ha asociado. Éste modelo es mucho más potente que las arquitecturas SIMD.

La gran ventaja de los sistemas SIMD con respecto a los MIMD es que necesitan menos hardware, ya que sólo requieren una unidad de control. Los sistemas MIMD son más independientes y ejecutan cada uno de ellos una copia del programa y del Sistema Operativo. Además de esto, los sistemas SIMD necesitan menos tiempo de inicio para las comunicaciones entre procesadores, debido a que son transferencias entre registros de los procesadores. Como conclusión, aunque los sistemas SIMD aparentemente sean más baratos que los MIMD, esto no es cierto, pues los sistemas MIMD pueden construirse con ordenadores convencionales de baja potencia.

3. Organización del espacio de direcciones de memoria

El sistema de memoria es el lugar donde se almacena la información del programa; tanto datos como instrucciones. Dependiendo de cómo se organice el sistema de memoria, los procesadores se comunicarán de diferentes formas. Atendiendo a esta organización, se podrán encontrar las siguientes arquitecturas:

3.1 Sistemas de memoria compartida

Los sistemas de memoria compartida se caracterizan principalmente en que los procesadores comparten físicamente la memoria y, por tanto, pueden acceder al mismo espacio de direcciones de la misma. Por ejemplo, si un procesador escribe en una de las direcciones de memoria, otro procesador del mismo sistema puede acceder a esa misma posición directamente. Consecuentemente, los cambios realizados por un procesador en la memoria común son visibles por los otros procesadores del sistema a través de una red de interconexión, como puede verse en la figura 7:

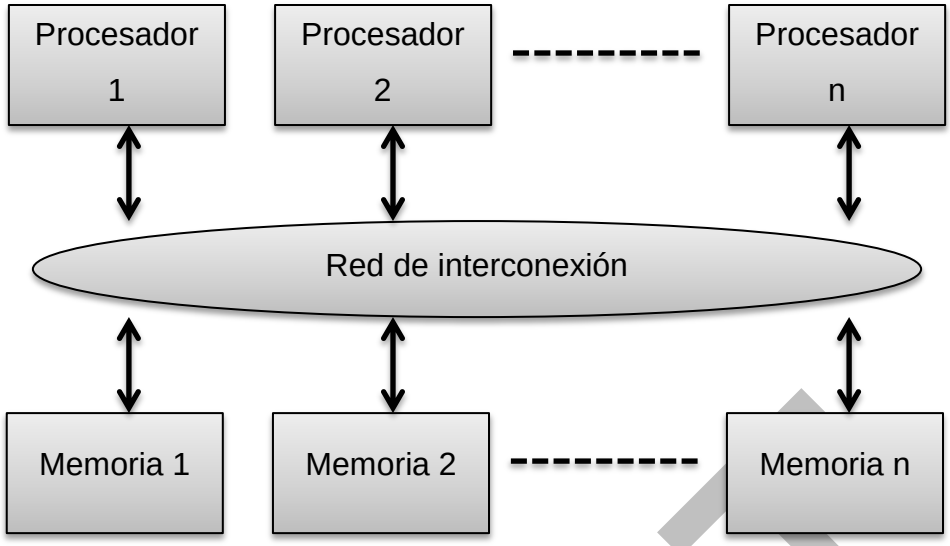

Figura 7 Sistema de memoria compartida

A su vez, los sistemas de memoria compartida se pueden clasificar en dos tipos fundamentales dependiendo el tiempo de acceso a la memoria por parte de los distintos procesadores. En primer lugar los sistemas de acceso uniforme a memoria (UMA, Uniform memory access). Estos sistemas se caracterizan por tener el mismo tiempo de acceso a memoria para todos los procesadores. La mayoría de los sistemas de memoria compartida se distinguen por incorporar una memoria caché local en cada procesador, consiguiendo por tanto un incremento en el ancho de banda entre el procesador y la memoria local. Estos sistemas se denominan CC-UMA (Cache Coherent UMA) son sistemas UMA donde la coherencia de caché debe intentar que si un procesador modifica una dirección de memoria, todos los procesadores del sistema deben estar informados al respecto.

Una de las limitaciones de la arquitectura UMA es la escalabilidad y normalmente encontramos estos sistemas con un número de procesadores comprendido entre 16 y 32; aunque algunos sistemas han logrado alcanzar hasta los 64 procesadores como el Sun E10000. Actualmente, los sistemas UMA se utilizan para la construcción de arquitecturas multiprocesador de bus

compartido con un número reducido de procesadores. Estos sistemas son comúnmente representados por los multiprocesadores simétricos.

Por otro lado, los sistemas de acceso no uniforme a memoria (NUMA). En estos sistemas se consigue una mejora del rendimiento eliminando la memoria global a la que acceden los procesadores. En estas arquitecturas los procesadores no tienen un tiempo igual de acceso a las memorias. En este caso el problema de escalabilidad no es tan serio como en el caso anterior.

3.2 Sistemas de memoria distribuida

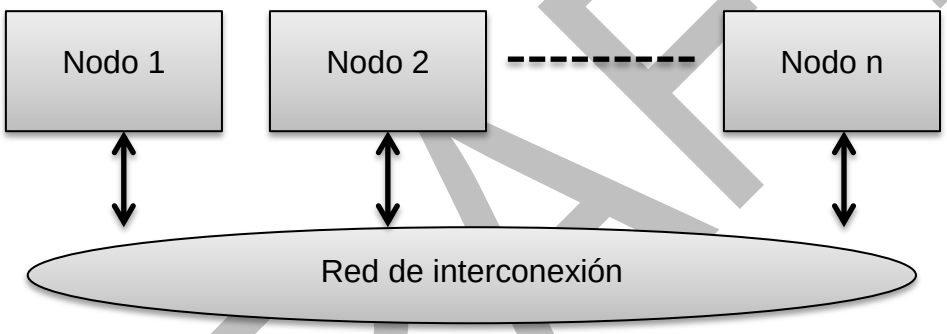

Figura 8. Arquitectura de memoria distribuida

Los sistemas de memoria distribuida se caracterizan porque cada nodo contiene su propio procesador y su propia memoria, comunicándose cada sistema entre sí por medio de paso de mensajes a través de una red de interconexión. Los sistemas de memoria distribuida pueden ser interconexiones entre distintos nodos de un clúster, estaciones de trabajo, etc. Estos nodos se conectan mediante una red de interconexión. Puesto que la latencia de la red puede ser alta, hay que tener este factor a la hora del desarrollo del programa paralelo. En este caso existe una situación de compromiso entre el tiempo requerido para realizar una operación básica de comunicación y el tiempo requerido para realizar una operación básica de cálculo de los procesos.

De esta forma, la estrategia de programación de los sistemas de memoria distribuida debe tener siempre como objetivo disminuir el sobrecosto de las comunicaciones, es decir, minimizar el número de mensajes y maximizar el tamaño. Cuando se combinan los sistemas MIMD con programación paralela de paso de mensajes se le denomina sistemas clúster.

4. Arquitecturas heterogéneas.

Las arquitecturas heterogéneas son aquellas que incluyen elementos computacionales de diversas naturalezas. En el mismo entorno de computación se pueden encontrar procesadores orientados a latencia, orientados a rendimiento y de propósito especifico que pueden ser utilizados para computar nuestras tareas. Además, la adopción de elementos computacionales heterogéneos en este momento es obligatorio para la eficiencia energética. Estas arquitecturas usan varios núcleos con diferente funcionalidad, rendimiento y eficiencia energética. De esta manera, el chip puede explotar las características de cada tarea para ejecutar de manera eficiente desde ambas perspectivas: tiempo de ejecución y consumo de energía. El runtime es todavía inmaduro para asignar eficientemente a los procesadores los cálculos que mejor se adaptan a sus necesidades, y por lo tanto los programadores desempeñan un papel fundamental en este paisaje emergente de computación para extraer el máximo rendimiento. No es una tarea fácil, ya que tienen que lidiar con diferentes componentes de hardware, conjuntos de instrucciones y modelos de programación, mientras se mantiene el consumo de energía en un umbral razonable. En la figura 9 se muestra un sistema heterogéneo disponible en la actualidad en cualquier ordenador.

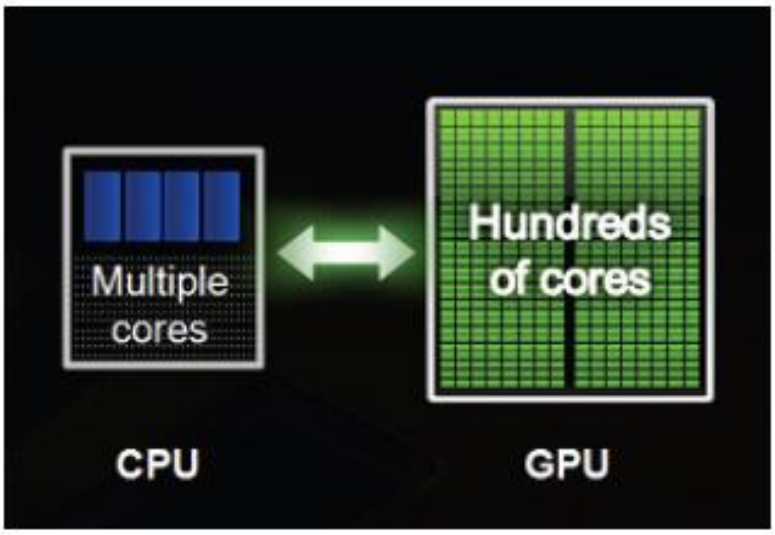

Figura 9. Arquitecturas heterogéneas.

4.1 CUDA: Compute Unified Device Architecture

La Unidad de Procesamiento gráfico está compuesta por un gran conjunto de procesadores que directamente pueden acceder a memoria, tanto de lectura como de escritura. Esto permite que el modelo de programación sea mucho más flexible y general que en las generaciones anteriores de GPUs. En esta sección vamos a describir la arquitectura de Nvidia CUDA (Computed Unified Device Architecture) centrándonos, sobre todo en los aspectos relevantes para la programación de propósito general de altas prestaciones.

La Figura 10 muestra una organización general de un computador. La tarjeta gráfica se encuentra conectada al sistema a través del Bus PCI Express que proporciona un ancho de banda de 4GB/s. Para acceder a la tarjeta gráfica tenemos que acceder mediante llamadas al sistema operativo que debe ejecutar el procesador principal.

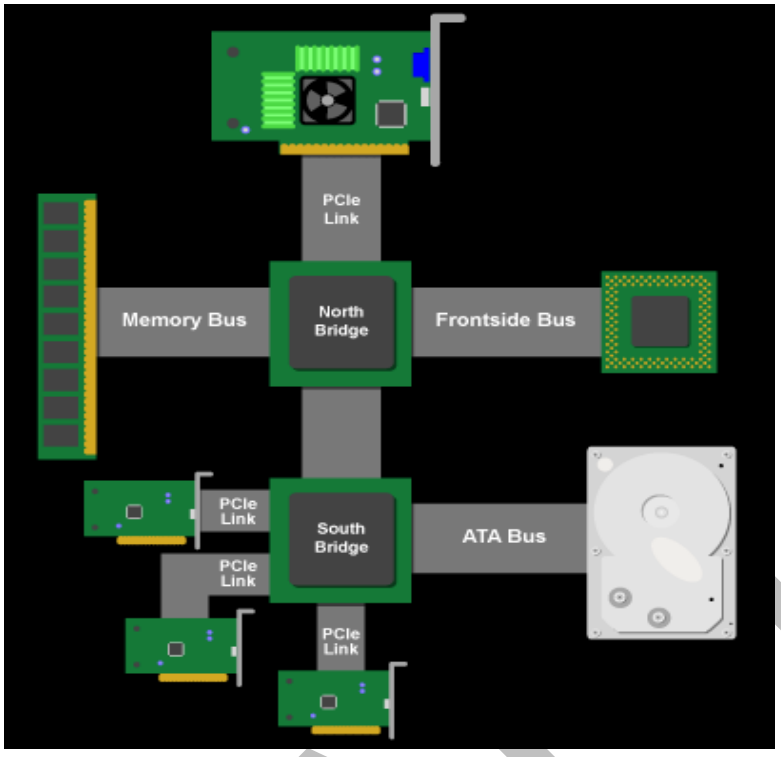

Figura 10. Organización general del ordenador

La figura 11 muestra la arquitectura grafica CUDA de Nvidia. Esta arquitecutra incluye una serie de multiprocesadores (SMs, Streaming multiprocessor). Estos multiprocesadores están compuestos por una serie de procesadores mas sencillos denominados en la terminología CUDA SPs, Streaming processor.

El número de multiprocesadores en la tarjeta gráfica y el número de procesadores por cada SM varia en función de la generación de GPU que estemos tratando. Además la tarjeta gráfica tiene su propia jerarquía de memoria, independiente de la jerarquía de memoria principal.

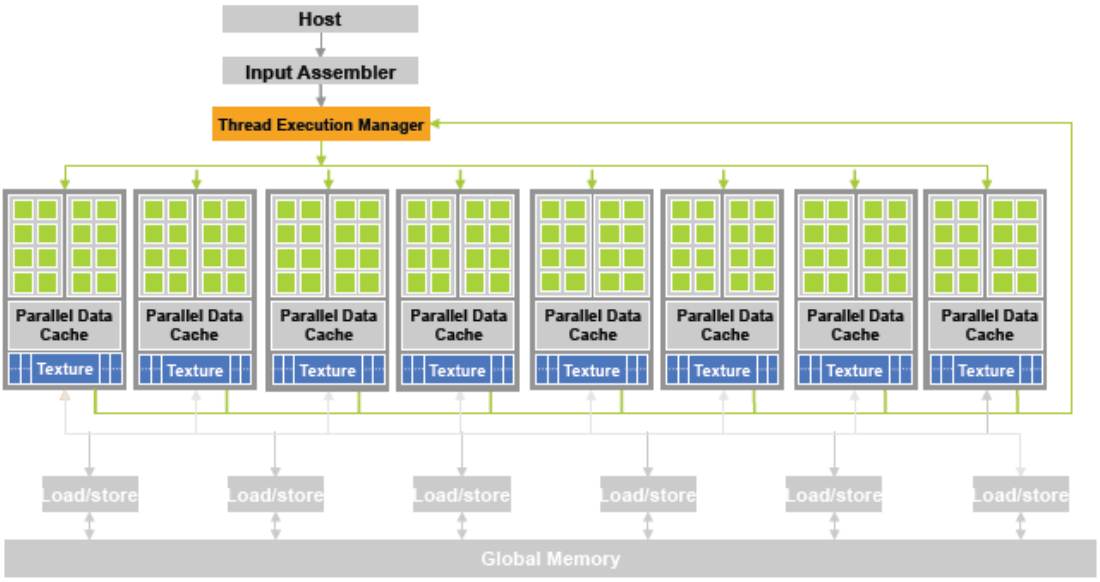

Figura 11. Arquitectura general de la GPU