Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

UNIDAD DE CONTROL SEGMENTADA

PROCESAMIENTO PARALELO

Índice

Presentación

Hasta ahora, hemos estudiado algunos conceptos sobre multiprocesamiento, incluso hemos visto el caso concreto de Intel, en el que podíamos comprobar cómo conseguir multiprocesamiento a nivel de chip.

En este tema, daremos un salto más amplio hacia las máquinas multiprocesadores, dando un formato más consecuente.

Para empezar, conoceremos los puntos clave para entender el porqué de la necesidad de máquinas supercomputadores, y los puntos clave en los que se basan para conseguir un alto rendimiento.

Posteriormente, daremos paso a cómo clasificar los ejemplares de máquinas según el tipo de paralelización que implementen.

Por tanto, los puntos en los que nos centraremos en este tema son:

- Introducción: proceso, hilo, cambio de contexto.
- Clasificación y niveles de paralelismo implementados en las máquinas supercomputadores.
- Diferentes tipos de ejecución multihilo a nivel hardware.

Introducción

Para empezar a hablar del paralelismo, antes necesitamos comprender cuáles son los niveles de paralelismo y dónde podemos hacer paralelismo.

Primero, preguntémonos por la necesidad. Los computadores masivamente paralelos (MPP) están en pleno desarrollo. Cada año se instalan más de 10 de estos supercomputadores en todo el mundo. Tal vez parezcan pocos, pero es que su coste supera los 1,25 mil millones de euros y unos 10 millones de coste al año de funcionamiento (consume 10MW de electricidad).

Pero su fabricación está justificada por las aplicaciones sobre las áreas en las que actualmente están trabajando, que según el HPCC son:

- Diseños de medicamentos principalmente orientados a lucha contra el cáncer y el SIDA.
- El origen de la materia y el universo.
- Avances en biomedicina, como el estudio del cerebro humano.
- Modelado de comportamiento oceánico, la dinámica de la capa de ozono, cambio climático, etc.
- Nuevos materiales y la creación de la energía
- Próxima generación de tecnología de fabricación
- Para defensa, que siempre está presente, aunque no sea de las más populares.

Pero para definir bien el alcance de esta asignatura, aclaremos que paralelismo es un término muy amplio, pero podemos definirlo como la capacidad de realizar operaciones al mismo tiempo. Destacamos dos niveles fundamentales:

Paralelismo interno	Multiplicidad de unidades funcionales, segmentación (de instrucciones, operaciones aritméticas, accesos a memoria).
Paralelismo externo	Fuera del procesador. Sistemas con muchos procesadores (computadores paralelos).

En esta unidad nos centraremos únicamente en el paralelismo interno, no sin antes repasar con más detalle, la necesidad del multiprocesamiento.

HPCC

HPCC son las siglas de *High-Performance Computing and Communication program*.

Rendimiento de un supercomputador

Como curiosidad, decir que un supercomputador de última generación resuelve en un día lo que a nuestro ordenador de sobremesa le llevaría del orden de 200 años. Extrapolar este calculo, y pensar en lo que podría resolver en un año de funcionamiento y ¿todo esto por solo 10 millones de dólares?

Conceptos básicos

Antes de comenzar, vamos a recordar ciertos aspectos sobre materias estudiadas con anterioridad. Repasemos las siguientes definiciones.

Programa	Para el programador, es un conjunto ordenado de instrucciones. Pare el sistema operativo, es un archivo ejecutable almacenado en memoria secundaria.
Proceso	También llamado tarea. Programa en ejecución, con su contexto asociado que especifica (espacio de direcciones que ocupa, recursos del compilador que utiliza, estatus, etc.).
Ciclo de vida de un proceso	Creación, ejecución y terminación. Con sus subfunciones de establecimiento de la descripción del proceso, asignación del espacio de memoria, carga del programa en el espacio de memoria, paso de la descripción del proceso al planificador, etc.

El proceso es una entidad pesada, es decir, tiene muchos recursos asociados. Además, hoy en día, un proceso no solo realiza una operación a la vez, sino que realiza multitud de acciones simultáneamente. Es decir, tiene varios flujos de ejecución simultánea.

Ejemplo: pensar en Word; al mismo tiempo que atiende el teclado para interpretar la teclas que el usuario escribe, por debajo, está formateando la página para mostrar cómo está quedando el documento, haciendo una corrección ortográfica instantánea, etc. No hace mucho, esto solo se podría logar teniendo un proceso para cada una de estas actividades simultáneas, trabajando sobre los mismos datos de entrada (lo que el usuario tecleaba).

En la actualidad, se ha creado un nuevo concepto: la idea de **thread**. El thread es la parte ejecutiva de un proceso. Por tanto, el proceso sigue siendo igual de complejo que antes, pero dentro de un mismo proceso, podemos tener más de un flujo (hilo) de ejecución, y todos estos hilos comparten los recursos del proceso.

Si deseas ver las formas básicas de crear y terminar threads en los lenguajes de programación, puedes consultar el siguiente enlace:

Documentos Crear y terminar threads

El proceso y su parte ejecutiva thread en la aplicación Word

Con thread, nuestro Word ya no necesita lanzar un gran número de procesos hijos para realizar estas tareas, sino que puede lanzar threads hijos dentro de un único contexto. Esto hace que la multitarea a nivel de proceso sea mucho más simple y liviana, pero su gestión es similar a la de los procesos, con la salvedad de que el cambio de contexto entre procesos es cara en tiempo, mientras que entre hilos es más fácil y rápida.

Clasifación de paralelismos

A continuación, vamos a hablar del paralelismo aplicado a modelos computacionales. Para empezar, un modelo computacional es un sistema que maneja datos para obtener resultados mediante la realización de unas determinadas operaciones en base a esos datos.

En relación con esta idea, podemos definir cuatro modelos de paralelismo.

SSID (Single Instruction Single Data)	Es la máquina no paralela. En cada paso se emite una instrucción, que opera sobre un único dato. Es típica de las máquinas con una sola unidad funcional, y es óptima para algoritmos secuenciales.
MISD (Multiple Instruction Single Data)	Un solo dato, al que se le dedican simultáneamente múltiples operaciones. Es un tipo de paralelismo que no se implementa, ya que sus usos y aplicaciones son muy pocas. Además, convertir esta máquina a una MIMD es fácil y no resulta muy caro.
SIMD (Single Instruction Single Data)	Una sola instrucción sobre una colección de datos. Es típica de máquina con la unidad de procesamiento de datos replicada (ALU). Todas estas unidades funcionales están coordinadas por una única unidad de control y de manera síncrona (todas a la vez). Estas máquinas son conocidas como matriciales, y son óptimas para operaciones vectoriales.
MIMD (Multiple Instruction Multiple Data)	Tienen los procesadores y las unidades operativas replicadas, pudiendo ejecutar múltiples instituciones sobre múltiples datos de manera simultánea, y la ejecución de instrucciones puede ser asíncrona.

Por tanto, podemos resumir:

- **Máquinas convencionales**: SISD.
- **Máquinas Vectoriales/Matriciales**: SIMD.
- **Máquinas MIMD**: Multiprocesadores/Multicomputadores.
- **Supercomputador**: máquinas más potentes en cada momento. Hoy MIMD.

Niveles de paralelismo

Para conseguir eficiencia en el paralelismo se han asignado multitud de recursos, tanto desde el punto de vista software como en el diseño de compiladores y sistemas operativos y hardware en el desarrollo de nuevas arquitecturas. Estos estudios dan un contexto de paralelismo dividido en dos grupos diferentes pero solapados.

Por un lado, encontramos el siguiente grupo.

Paralelismo en tiempo de diseño de programa (available parallelism)	Este paralelismo se centra en el momento de hacer el programa, por lo que los sistemas operactivos y el compilador también tienen su importancia. Este paralelismo, se estudia conjuntamente con otras asignaturas como las relacionas con la concurrencia y los sistemas operativos.
Paralelismo en tiempo de ejecución (utilized parallelism)	Es un paralelismo que se realiza en el momento de ejecutar el programa, identificando las partes que se pueden solapar y que permiten la ejecución simultanea.

Por otro lado, encontramos este otro grupo.

Paralelismo Funcional	Se consigue a partir de la lógica de solución de un problema, es decir, cómo se plantea el diagrama de flujo, el programa, el código que resuelve el problema. • Ejemplo1: A= (B+C)*(C+D); La resolución del problema implica realizar dos sumas y multiplicar los resultados de ambas. De la lógica de la solución se puede deducir que las dos sumas se podrían realizar en paralelo. • Ejemplo2: A= proc(func1(B,C),func2(C,D)); De igual forma en este ejemplo se deduce que las funciones func1 y func2 se pueden resolver en paralelo.

Paralelismo de Datos	Se consigue a partir de las estructuras de datos que se empleen. Existen estructuras concretas que permiten un paralelismo claro de los datos, como los vectores o las matrices. • Ejemplo: sean X, A y B vectores de n elementos, la suma de A+B puede ser: para i∈ [0..n] hacer $X_i = A_i+B_i$ ≠ calcular $X_{[0..n]} = A_{[0,,n]}+B_{[0..n]}$. La primera parte es secuencial, mientras que la segunda es paralela.

Paralelismo funcional y de datos

Paralelismo funcional

A nivel de software, existen muchos niveles de paralelizar una aplicación, como por ejemplo: dos instrucciones que se pueden realizar al mismo tiempo, dos funciones o incluso dos hilos distintos. Por tanto, podemos conseguir paralelización a muchos niveles, que es una realidad a la que llamamos **grano de paralelización**. Así, se diferencian cuatro niveles:

Paralelismo a nivel de instrucción (grano fino)	Permite que las instrucciones individuales de un programa se puedan ejecutar en paralelo. Las instrucciones pueden venir en ensamblador o con el formato de los lenguajes de alto nivel. Normalmente, el paralelismo a nivel de instrucción se comprende mejor en el nivel ensamblador.
Paralelismo a nivel de bucle (grano medio)	Cuando realizamos en un programa tenemos múltiples iteraciones sobre un bucle, este puede ser un trozo de código paralelizable. Sin embargo, la recurrencia, debido a la dependencia entre los datos entre una iteración y la subsiguiente, puede hacer más restrictivo y menos eficaz el paralelismo
Paralelismo a nivel de procedimiento (grano medio)	Cuando dos procedimientos o funciones se pueden realizar simultáneamente, ya que no hay dependencias entre ellos y el orden no es determinante.
Paralelismo a nivel de programa (grano grueso)	El paralelismo a nivel de programa consiste en tener más de un proceso independiente entre sí, ejecutándose simaltáneamente.

Podemos conseguir el paralelismo a nivel de instrucción empleando arquitecturas ILP.

Por término general, el paralelismo a nivel de bucle es un procedimiento que se implementa mediante thread.

Paralelismo de datos

El paralelismo de datos se puede implementar de dos maneras diferentes:

- Por un lado, una posibilidad consiste en conseguir dicho paralelismo directamente a través de arquitecturas dedicadas, llamadas DP-architectures (*Data Parallelism*), que permitan paralelizar operaciones sobre datos.
- Por otro lado, se trata de conseguir el paralelismo de datos remplazando las operaciones sobre vectores en bucles, y aplicar paralelismo funcional a nivel de bucle.

Arquitecturas ILP

Las arquitecturas **ILP** (*Instruction-level parallelism*) suelen ser transparentes al programador, siendo un compilador (los compiladores están preparados para detectar y optimizar el paralelismo) o el propio microprocesador quien se encarga de realizar la labor.

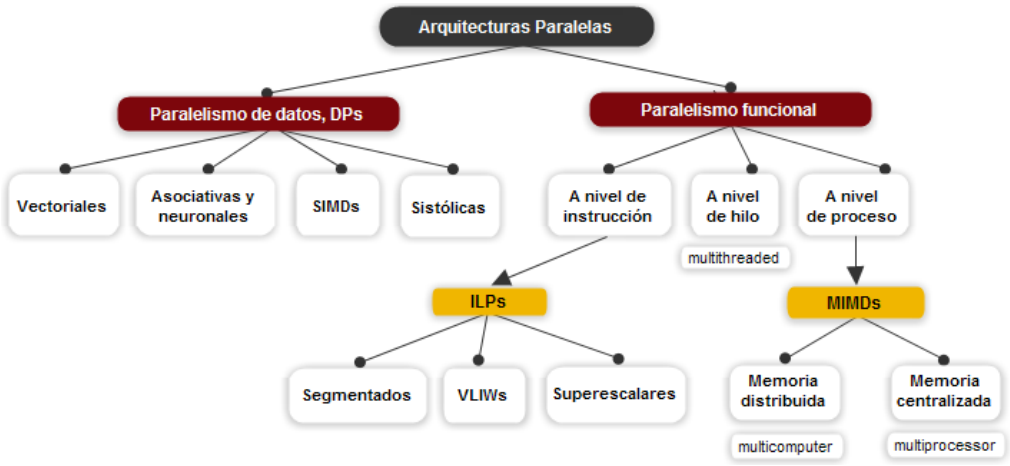
Implementación mediante thread

El procedimiento del paralelismo a nivel de bucle se implementa mediante thread, dado que estos se pueden crear manualmente por los programadores (si el lenguaje lo soporta) o por sistemas operativos que soporten la multitarea. Además, algunos compiladores paralelos pueden organizar de manera automática el código para maximizar la paralelización a estos niveles.

La paralelización a nivel de thread se puede implementar físicamente en la máquina mediante dos formas: bien porque la arquitectura hardware soporta la ejecución paralela por tener replicados las unidades funcionales (máquinas multithread o MIMD), o bien de manera secuencial (uno tras otro, con intercambios muy rápidos), en máquinas con sistemas operativos que soporten multitarea.

Criterios de clasificación

Existen muchas formas de clasificar las arquitecturas paralelas, y cada bibliografía que estudiemos propone una distinta. Pero la clasificación más interesante, por ser la más cercana al estudio de esta materia, es la señalada por **Dezsö Sima**. Esta clasificación se basa en el tipo de paralelismo que explota la arquitectura de la máquina. Al contrario que otras categorías, esta empieza por la división a nivel de funcional y de datos, lo que no suele ser muy habitual.



Dentro de la rama de paralelismo a nivel de datos, encontramos los vectoriales y los SIMD.

El paralelismo a nivel funcional solo diferencia tres grupos. Los basados en bucle y procedimiento se agrupan en un grano medio, llamado **nivel de hilo** (thread). Dentro de este nivel de paralelización funcional encontramos los **ILPs**, a los que le dedicaremos los temas siguientes de esta unidad, dejando los de MIMD para el resto de unidades, donde podremos obtener una visión más profunda de los mismos.

Datos vectoriales y SIMD

Esta materia la estudiaremos más adelante.

Ejecución multihilo en hardware

La ejecución multihilo en hardware (*hardware multithreading*) permite que varios hilos (threads) compartan las unidades funcionales del procesador y se ejecuten simultáneamente. Para facilitar la compartición de los recursos del procesador, cada hilo debe tener su propio estado. Por ejemplo, cada hilo debería tener una copia del banco de registros y el PC. La memoria puede compartirse gracias a la memoria virtual, que ya incluye soporte para multiprogramación. Además, el hardware debe tener la capacidad de conmutar de forma rápida la ejecución entre diferentes hilos.

Existen dos alternativas para la ejecución multihilo en hardware.

Ejecución multihilo de grano fino (fine-grained multithreading)	Se cambia de hilo en cada instrucción, y como resultado, se obtiene una ejecución entrelazada de varios hilos. Este entrelazado se hace habitualmente siguiendo un turno rotatorio entre los hilos, saltándose los hilos parados en ese momento. Para que esta ejecución sea práctica, el procesador debe ser capaz de cambiar de hilo en cada ciclo del reloj. Una ventaja clave de tipo de ejecución es su capacidad de ocultar las pérdidas de prestaciones debidas a paradas largas y cortas, porque cuando un hilo se para, se pueden ejecutar instrucciones de otros hilos. Por el contrario, la principal desventaja, es la pérdida de velocidad en la ejecución de un hilo individual, porque un hilo que esté listo para ejecutarse sin paradas se verá interrumpido por instrucciones de otros hilos.
Ejecución multihilo de grano grueso (coarse-grained multithreading)	La conmutación entre hilos se realiza solo cuando hay paradas largas; por ejemplo, en un fallo de la caché del segundo nivel. Este cambio alivia el requisito de cambios de hilos sin coste, y la probabilidad de hacer más lenta la ejecución de un hilo individual es mucho menor, porque solamente se ejecutan instrucciones de otros hilos cuando se produce una parada larga en el hilo en cuestión. Sin embargo, este tipo de ejecución tiene un importante inconveniente: su limitada capacidad de reducir las pérdidas en las prestaciones debidas a paradas cortas. Esto se debe al sobrecoste del llenado del pipeline, donde la ejecución multihilo de grano grueso es más útil para reducir la penalización de las paradas de alto coste, y el tiempo de llenado del pipeline es muy pequeño comparado con el tiempo de parada.

A continuación, vemos un resumen de lo anteriormente expuesto:

Conmutación de diferentes hilos

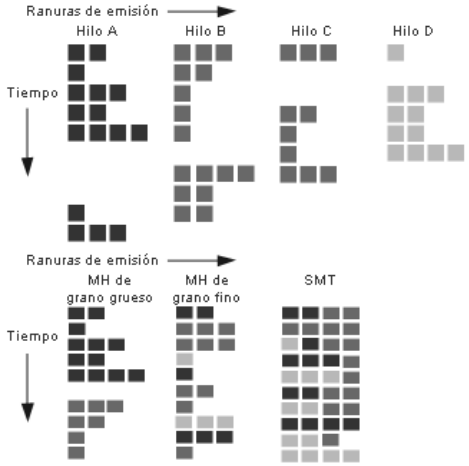
En particular, el cambio de hilo debe ser mucho más rápido que el cambio de proceso, que típicamente necesita cientos o miles de ciclos del procesador, mientras que el cambio de hilo debe ser instantáneo.

- **Ejecución multihilo en hardware**: aumento de la utilización del procesador mediante la conmutación a otro hilo cuando un hilo se para.
- **Ejecución multihilo de grano fino**: versión de la ejecución multihilo en hardware que conmuta entre hilos después de cada instrucción.
- **Ejecución multihilo de grano grueso**: versión de la ejecución multihilo en hardware que conmuta entre hilos solo después de un suceso importante, por ejemplo, un fallo de caché.

Ejecución multihilo simultánea

La **ejecución multihilo simultánea** (*simultaneous multithreading*, SMT) es una variante de la ejecución multihilo en hardware que utiliza los recursos de un procesador con planificación dinámica y emisión múltiple de instrucciones para explotar el paralelismo a nivel de hilo al mismo tiempo que explota el paralelismo a nivel de instrucción. El punto clave que ha motivado el desarrollo de los SMT, es que los procesadores con emisión múltiple de instrucciones tienen, a menudo, un número mayor de unidades funcionales que las que puede utilizar de forma efectiva un único hilo. Además, con el renombrado de registros y la planificación dinámica se pueden emitir varias instrucciones de diferentes hilos sin preocuparnos de las dependencias entre ellas; las dependencias se resuelven gracias a la planificación dinámica.

Como se confía en la existencia de mecanismos dinámicos, los SMT no conmutan en cada ciclo, sino que están siempre ejecutando instrucciones de varios hilos, dejando que el hardware determine qué instrucciones se ejecutan y asocie los registros renombrados al hilo adecuado.



En la imagen se ilustra, de forma conceptual, las diferencias en la capacidad del procesador para explotar los recursos superescalares para las siguientes configuraciones. La parte superior muestra cómo se ejecutarían de forma independiente cuatro hilos en un procesador sin soporte para ejecución multihilo. La parte inferior muestra cómo se combinan los cuatro hilos para ejecutarse de forma más eficiente con las tres alternativas de ejecución multihilo, y que nombramos a continuación:

- Procesador superescalar con ejecución multihilo de grano grueso.
- Procesador superescalar con ejecución multihilo de grano fino.
- Procesador superescalar con ejecución multihilo simultánea.

Resumen

Las palabras clave que tenemos que asimilar de este tema, son las siguientes:

- **Programa**, **proceso y ciclo de vida**. Que se refiere al software y cómo conseguir el multiprocesamiento.
- **SSID**, **MISD**, **SIMD** y **MIMD**. Que se refiere al tipo de paralelismo logrado, sobre datos y sobre instrucciones.
- **Paralelismo funcional** y de **datos**.
- **Ejecución multihilo**, de **grano fino** o **grueso**.

Por su parte, hemos categorizado de manera formal los tipos de multiprocesadores y hemos representado, en un esquema gráfico, cómo se podrían colocar todos los tipos de ordenadores, paralelos y no paralelos.

Finalmente, hemos aprendido cómo conseguir una paralelización a nivel de hilo (o proceso).