Universidad
Europea de Madrid

LAUREATE INTERNATIONAL UNIVERSITIES

MULTIPROCESADORES

EL SOFTWARE EN LOS MULTIPROCESADORES

© Todos los derechos de propiedad intelectual de esta obra pertenecen en exclusiva a la Universidad Europea de Madrid, S.L.U. Queda terminantemente prohibida la reproducción, puesta a disposición del público y en general cualquier otra forma de explotación de toda o parte de la misma.

La utilización no autorizada de esta obra, así como los perjuicios ocasionados en los derechos de propiedad intelectual e industrial de la Universidad Europea de Madrid, S.L.U., darán lugar al ejercicio de las acciones que legalmente le correspondan y, en su caso, a las responsabilidades que de dicho ejercicio se deriven.

Índice

Presentación

En este tema, veremos cómo afecta al software que una máquina sea multiprocesador. Tanto al sistema operativo como a las aplicaciones en general.

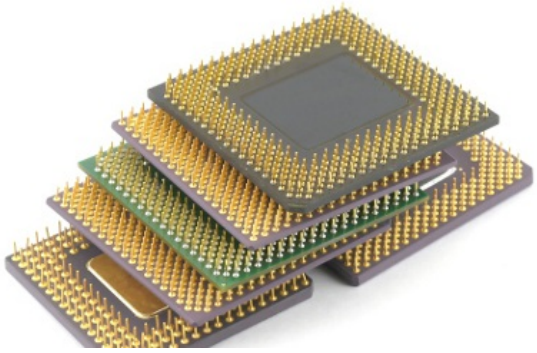
Veremos cuáles son los puntos fundamentales en el diseño de aplicaciones para que estas aprovechen las características de máquinas multiprocesador.

Para terminar, haremos un estudio de una de las familias más extendidas tanto dentro de los ordenadores personales como de los grandes supercomputadores: la familia Intel.

Analizaremos cuales son las tendencias más modernas que Intel implementa para optimizar la ejecución de código y explicaremos por qué el de Intel es un buen procesador tanto para uso personal como para uso intensivo.

El tema lo desglosaremos en los siguientes puntos fundamentales:

- Cómo crear programas de procesamiento paralelo y los tipos de paralelización.
- Describiremos el multiprocesamiento en un solo chip y en multiprocesadores.
- Caso real: Intel.



La dificultad de crear programas de procesamiento paralelo

La dificultad del paralelismo no solo está en el hardware, tal vez la principal dificultad esté en el software. Hay muy pocas aplicaciones que se hayan reescrito para completar sus tareas en menos tiempo en un multiprocesador. Es difícil escribir software que utilice varios procesadores para terminar una tarea de forma más rápida, y el problema empeora al aumentar el número de procesadores.

¿Por qué es más difícil desarrollar programas de procesamiento paralelo que programas secuenciales?

Como es lógico pensar, para que un programa sea eficaz en una máquina multiprocesador, debe aprovechar al máximo las características de una arquitectura paralela. Por el contrario, en un monoprocesador es preferible utilizar un programa secuencial pues la programación es más sencilla y el aumento de eficiencia es poco significativo. De hecho, las técnicas de diseño de monoprocesadores, tales como súper escalar y ejecución fuera de orden, aprovechan el paralelismo a nivel de instrucción (que veremos en este tema) normalmente sin la participación del programador.

¿Por qué es difícil escribir programas de procesamiento paralelo que sean rápidos, especialmente cuando aumenta el número de procesadores?

Porque para que sea eficiente, el software tiene que poder dividirse en tantas subtareas como procesadores tenga la máquina. Pensemos en lo que cuesta hacer un trabajo entre un grupo de compañeros de prácticas cuando dicho grupo no tiene dos o tres, sino 50 o 60 integrantes y, a continuación, supongamos que la tarea es difícil de dividir, ya que para que un compañero pueda hacer su parte del trabajo, necesita resultados de otro compañero. O supongamos que el trabajo es tan pequeño que se tarda más tiempo en dividirlo y juntarlo, que en que una sola persona lo haga completamente.

Otro riesgo para las prestaciones se produciría cuando hay que emplear demasiado tiempo en comunicarse con sus compañeros en lugar de trabajar en su parte de la tarea. Podemos extraer algunas conclusiones de esta analogía con la programación paralela, los desafíos incluyen la planificación de las tareas, el equilibrio de la carga, el tiempo de sincronización y la sobrecarga de las comunicaciones entre colaboradores. El reto es todavía más difícil cuantos más compañeros participen en la resolución de la actividad y cuantos más procesadores estén disponibles para la programación paralela.

No nos olvidemos de lo visto sobre la ley de Amdahl. Esta ley nos recuerda que incluso las partes pequeñas de un programa deben paralelizarse si se quiere hacer un buen uso de los muchos núcleos disponibles.

División

Y dividirlo cumpliendo unas premisas, como por ejemplo que la división sea equitativa en tiempo y complejidad, ya que si no, un compañero puede terminar más tarde que los demás, lo que la mejora es poco significativo.

Técnicas:súper escalar y ejecución fuera de orden

Estas técnicas disminuyen la demanda de reescritura de programas para multiprocesadores, porque sin que los programadores tengan que hacer nada más, sus programas secuenciales se ejecutarán más rápido en los nuevos computadores.

Escalado fuerte-débil

En un multiprocesador, es más difícil obtener buenas aceleraciones manteniendo el tamaño del problema fijo que incrementando el tamaño del problema. Esto nos permite introducir dos términos que describen dos formas diferentes de escalado.

Escalado fuerte	Hace referencia a la aceleración con un tamaño de problema fijo
Escalado débil	Implica que el tamaño del problema aumenta proporcionalmente al número de procesadores.

Supongamos que el tamaño del problema, M, es el conjunto de trabajo en memoria principal y que el número de procesadores de que se dispone es P. Entonces, la memoria por procesador en escalado fuerte es aproximadamente M/P y, en escalado débil, es aproximadamente M.

Dependiendo del tipo de aplicación, se puede defender cualquier tipo de escalado. Por ejemplo, los programas de pruebas benchmark de una base de datos crédito-débito requieren que el número de cuentas de clientes aumente para aumentar el número de transacciones por minuto.

El argumento utilizado reside en lo absurdo que resulta pensar que un número dado de clientes, de repente, va a utilizar los cajeros automáticos 100 veces al día simplemente porque el banco tiene un computador más rápido. En lugar de esto, si se quiere demostrar que el sistema puede multiplicar por 100 el número de transacciones por minuto, se debe hacer el experimento multiplicando por 100 el número de clientes.

Este último ejemplo muestra la importancia del equilibrio de la carga. A continuación, puedes encontrar las definiciones correspondientes a estos dos tipos de escalado.

Escalado fuerte	Aceleración obtenida en un multiprocesador sin aumentar el tamaño del problema.
Escalado débil	Aceleración obtenida en un multiprocesador aumentando el tamaño del problema proporcionalmente al número de procesadores.

Consideraciones de diseño de un sistema operativo de multiprocesador

Un sistema operativo de SMP (*Symmetric Multiprocessor*) gestiona los procesadores y demás recursos del computador para que el usuario perciba un solo sistema operativo controlando los recursos del sistema. De hecho, el computador debería parecer un sistema monoprocesador con multiprogramación.

Tanto en un sistema multiprocesador como en un sistema monoprocesador, pueden estar activos varios trabajos o procesos al mismo tiempo, y es responsabilidad del sistema operativo planificar su ejecución y asignar los recursos.

Un usuario puede desarrollar aplicaciones que utilicen varios procesos o varios hilos dentro de un proceso sin tener en cuenta si se dispone de uno o de varios procesadores.

> Así, un sistema operativo de multiprocesador debe proporcionar toda la funcionalidad de un sistema operativo con multiprogramación y, además, las características adicionales que permitan utilizar varios procesadores.

Entre los puntos clave de diseño están los siguientes.

Procesos concurrentes simultáneos	Las rutinas del sistema operativo deben ser reentrantes para permitir que varios procesadores puedan ejecutar simultáneamente el mismo código IS. Con varios procesadores ejecutando la misma o distintas partes del sistema operativo, las tablas y las estructuras de gestión del sistema operativo deben manejarse apropiadamente para evitar bloqueos u operaciones no válidas.
Planificación	La planificación puede realizarla cualquier procesador, por lo que deben evitarse los conflictos. El planificador debe asignar los procesos preparados a los procesadores disponibles.
Sincronización	Puesto que hay varios procesos que pueden acceder a espacios de memoria y a recursos de E/S compartidos, debe proporcionarse una sincronización efectiva. La sincronización asegura la exclusión mutua y la ordenación de eventos.

Gestión de memoria	La gestión de memoria en un multiprocesador debe comprender todos los aspectos propios de los computadores monoprocesadores, discutidos en una unidad anterior. Además, el sistema operativo debe explotar el paralelismo que proporciona el hardware, por ejemplo las memorias multipuerto, para obtener mejores prestaciones. Los mecanismos de paginación en procesadores distintos deben coordinarse, para mantener la consistencia y para decidir sobre el reemplazo de páginas, cuando varios procesadores comparten una página o un segmento.
Fiabilidad y tolerancia a fallos	El sistema operativo debería hacer posible una degradación gradual cuando se produce un fallo en un procesador. El planificador y otros elementos del sistema operativo deben reconocer la pérdida de un procesador y reestructurar las tablas de gestión en consecuencia.

Procesamiento multithread y multiprocesadores monochip

Para un procesador, la medida de prestaciones más importante es la velocidad a la que ejecuta instrucciones. Esta se puede expresar como:

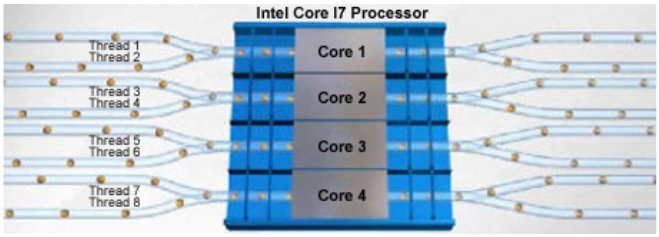
Velocidad (MIPS) = f * *IPC*

Donde f es la frecuencia de reloj del procesador, en MHz, e IPC (instrucciones por ciclo) es el promedio de instrucciones ejecutadas por ciclo. En consecuencia, los diseñadores han buscado la mejora de prestaciones en dos frentes:

- Incrementando la frecuencia de reloj.
- Incrementando el número de instrucciones ejecutadas o, más propiamente, el número de instrucciones que se completan en un ciclo de reloj.

Una alternativa, que permite un paralelismo entre instrucciones elevado sin incrementar ni la complejidad de los circuitos ni el consumo de potencia, es el procesamiento multithread. Básicamente, la secuencia de instrucciones se divide en secuencias más pequeñas, denominadas threads (hilos), que pueden ejecutarse en paralelo.

La diversidad de diseños multithread que se han realizado, tanto en procesadores comerciales como experimentales, es considerable. En estos temas, proporcionamos una breve revisión de los conceptos principales.



Fuente: Intel Corporation

Procesamiento multithread implícito y explícito

El concepto de thread utilizado para estudiar los procesadores multithread puede ser o puede no ser el mismo que el concepto de thread en un sistema operativo multiprogramado. Por lo tanto, es útil definir brevemente una serie de términos.

Proceso	Programa en ejecución en un computador. Un proceso reúne dos características clave: • Propiedad de recursos: un proceso dispone de un espacio de direcciones virtuales, los datos, la pila y demás atributos que definen el proceso. En determinadas ocasiones, recursos tales como memoria principal, canales de E/S, dispositivos de E/S y ficheros. • Planificación/ejecución: la ejecución de un proceso sigue un camino de ejecución (traza) a través de uno o más programas.
Conmutación de proceso	Operación de cambio del proceso que se está ejecutando en el procesador por otro proceso. Para ello, se almacenan todos los datos de control, registros y demás información del primer proceso y se reemplazan con la información correspondiente al segundo.

Thread	Una unidad de trabajo, dentro de un proceso, que se puede asignar al procesador. Incluye un contexto de procesador y su propia área de datos para la pila. Un thread se ejecuta secuencialmente y puede interrumpirse para que el procesador pase a ejecutar otro thread.
Conmutación de thread	El control del procesador pasa de un thread a otro dentro de un mismo proceso. Usualmente, este tipo de conmutación es mucho menos costosa que la de procesos.

Así, mientras los threads aparecen involucrados en la planificación y ejecución, los procesos tienen que ver tanto con la planificación/ejecución como con la asignación de recursos. Los threads de un mismo proceso comparten los mismos recursos. Esta es la razón por la que un cambio de thread consume mucho menos tiempo que la conmutación de procesos.

Hasta ahora, todos los procesadores comerciales han utilizado procesamiento multithread explícito. Estos sistemas ejecutan concurrentemente instrucciones diferentes de threads explícitos, bien entremezclando instrucciones de threads diferentes en cauces compartidos o mediante ejecución paralela y cauces paralelos. Es decir, thread explicito es aquel que es creado por el programador en tiempo de codificación.

El procesamiento multithread implícito hace referencia a la ejecución concurrente de varios threads extraídos de un único programa secuencial. Estos threads implícitos pueden ser definidos estáticamente por el compilador o dinámicamente por el hardware. Para seguir hablando de software, el thread que más nos importa en esta sección es el explícito.

Threads

La mayoría de los sistemas operativos actuales (Linux, Unix, Windows, etc.), permiten el uso de threads.

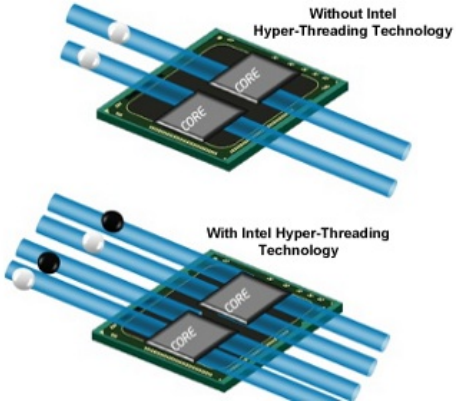
Caso real: Intel I

Cuando hablo de esta familia en concreto, me gusta comenzar diciendo que la elección de este microprocesador en concreto, no es porque sea el mejor/peor, el más barato/caro, etc. Dicha elección, se hace simplemente porque su uso está muy extendido en computadores comerciales, es fácil encontrar documentación sobre el microprocesador en cuestión y nos podemos hacer una imagen mental de lo que estamos hablando.

Si Intel es bueno en algo, es en su capacidad de hacer marketing sobre una imagen. Pero el marketing sería insuficiente si Intel no supiera adecuarse al mercado (a las nuevas tecnologías y tendencias) para sacar el mayor rendimiento a los ordenadores actuales. De hecho, me gusta aclarar que normalmente, aunque no siempre, el fabricante de hardware es el que implementa nuevas funcionalidades a su hardware y el sistema operativo es el que anda a remolque, para darle soporte a estas nuevas tecnologías.

¿Cómo se sitúa Intel dentro de esta arquitectura multiprocesador/multithread de la que hablamos?

Pues nada mal, de hecho, la mayor parte procesadores instalados en los supercomputadores que se fabrican hoy son fabricados entre Intel y su principal competidor: AMD.



Fuente: Intel Corporation

Intel, en la actualidad, fabrica procesadores multinúcleo (cores) que permiten ejecutar procesos en paralelo de manera totalmente simultanea, como por ejemplo los i3, i5, o los i7. Todos estos procesadores se fabrican con de 2 a 8 cores en un solo chip y con memoria cache de diferente nivel, integrando tanto compartida como no compartida.

Además, Intel incorpora una nueva tecnología que le ha dado mucho éxito para ordenadores monoprocesadores, ya que permite aumentar considerablemente la eficiencia con un bajo coste: el Hyper-Threading, que consiste en la ejecución de varios threads de un proceso, en un solo núcleo (o core).

Caso real: Intel II

¿Cómo se logra esto?

Aprovechando los tiempos muertos en la ejecución de un thread. Tal como se ha visto antes, los procesos tienen tiempos muertos en la ejecución secuencial de un código. Estos tiempos muertos pueden ser de tiempo largo o de tiempo corto. Como veremos próximamente, los huecos de tiempo largo son fácilmente recuperables mediante un cambio de contexto.

¿Pero qué paso con los retrasos por tiempos cortos?

Se llama tiempo corto al ciclo o dos ciclos de reloj que hay que esperar para solucionar fallos en el conjunto de instrucciones que, por coste, no se solucionan con hardware (como los fallos en la segmentación que trataremos posteriormente).

Pero Intel, en vez de destinar presupuesto para mejorar el hardware de sus procesadores, invento un método para aprovechar esos tiempos de espera conmutando la ejecución de ese hilo con otro del mismo proceso. De esta manera, entre los dos hilos de ejecución no se desperdicia ningún ciclo de reloj.

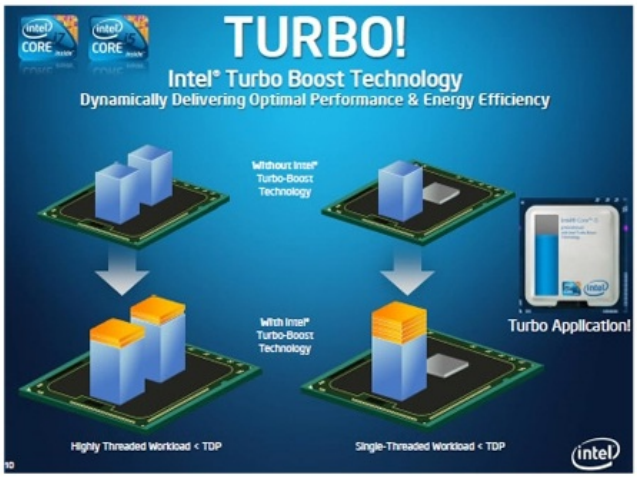
Esta técnica, que necesita de un tiempo de conmutación, o tiempo que se tarda en pasar de ejecutar un thread a ejecutar otro, muy rápido (tiempo=0). Con este método se consigue simultanear, en más de un 30%, la ejecución de un segundo thread sobre el thread principal, lo que supone un gran aumento de la eficiencia con un coste muy moderado.

1/4

El Hyper-Threading se implementa internamente al microprocesador, de manera que el S.O. reconoce un único core como dos cores virtuales diferentes.

Pero como ya se ha descrito en la primera pantalla de este tema, la creación de software que utilice todos los cores eficientemente es escaso, y mucho más escaso en los ordenadores personales. De los cuales Intel, también tiene un nicho de mercado muy importante.

Por tanto, ha dotado a sus procesadores de otra mejora importante: el turbo boost.



2/4

¿En qué consiste el Turbo Boost?

Mucho del software que se ejecuta ahora mismo en un procesador es secuencial monohilo. Por lo que tener 8 cores para que 7 estén parados y solo uno funcionando, es un gasto de recursos y energético importante.

Con esta nueva tecnología, Intel puede apagar los cores que no son necesarios. Este apagado permite mantener el procesador más frio y consumir menos energía. Cosa que se utiliza para sobre-forzar el core que se mantiene encendido, aumentando su ciclo de reloj y, también considerablemente, la ejecución monohilo.

Con estas dos técnicas, Intel cubre las dos situaciones actuales más típicas.

- Procesos multihilo de poca escalabilidad (2-6 hilos), muy abundantes en uso domestico, mediante el Hyper-Threading.
- Software antiguo monohilo, ahorrando energía y haciendo que sus procesad¬ores puedan aumentar sus prestaciones cuando no se utilizan todas sus características, mediante el turbo boost.

3/4

¿Cuáles son los puntos débiles de Intel?

Hoy en día, los puntos débiles de Intel son los buses de acceso a dispositivos de E/S y, principalmente, a memoria. El problema reside en que Intel utiliza un único canal de transmisión de datos con la memoria, independientemente del número de cores que este tenga, por lo que se convierte en un cuello de botella que limita mucho la escalabilidad de sus procesadores (6-8 es el límite eficiente de escalabilidad).

Este problema no lo tiene por ejemplo AMD, ya que cada núcleo de sus procesadores tiene sus propios canales de comunicación con la memoria, aunque no implementa tecnología Hyper-Threading ni Turbo-Boost. Estos procesadores, por el contrario, tienen un comportamiento muy bueno en maquinas clúster multiprocesador, ya que son muy optimas en la ejecución de procesos independientes, no en procesos multithread.

4/4

Resumen

En este tema hemos visto la importancia del balanceo de carga y de la división en tareas paralelas de las aplicaciones, y del sistema operativo, para conseguir una buena *paralelización* y, por consiguiente, un buen uso de todas las características de un ordenador multiprocesador.

Hemos analizado que los microprocesadores actuales pueden implementar multiprocesamiento a nivel de chip, ya que suelen incluir más de un core internamente, lo que permite ejecutar más de un proceso simultáneamente. Recordemos la jerarquía de memoria dentro de chip, con la jerarquía de cache y los problemas que esto trae.

Hemos descubierto las técnicas utilizadas por los procesadores Intel para aumentar al máximo sus prestaciones, no perdiendo ni un ciclo de reloj, gracias a la tecnología Hyper-Threading, y no perdiendo tampoco efectividad en sistemas monoproceso o de proceso preferente, como en los ordenadores personales monousuario, en los que claramente solo hay un proceso que consume mucho más de la mitad del procesador.

En estos casos, gracias al turbo boost, hace un overclocking de algunos cores, al tiempo que puede desactivar por completo otros.