UNIVERSIDAD TECNOLÓGICA DE PANAMÁ
CENTRO REGIONAL DE VERAGUAS

GLOCAL 2010

PONENCIA:	***PROCESADORES MULTINÚCLEO: REALIDADES Y FALACIAS.***
EXPOSITOR:	***RAÚL ENRIQUE DUTARI DUTARI.***
FECHA:	***21 DE OCTUBRE DE 2010.***
HORA:	***07:00 P. M.***
LUGAR:	***SALÓN DE CONFERENCIAS CARLOS ÁLVAREZ, CENTRO REGIONAL DE VERAGUAS.***
DIRIGIDA A:	***PROFESORES UNIVERSITARIOS, PROFESIONALES Y ESTUDIANTES QUE PARTICIPARON EN EL EVENTO.***
DURACIÓN:	***60 MINUTOS.***

OBJETIVO GENERAL

1. Debatir las características de los escenarios donde se puede explotar toda la potencia de hardware que ofrecen los procesadores con arquitectura Multi-núcleo.

OBJETIVOS ESPECÍFICOS

1. Describir las taxonomías que clasifican a la computación paralela.

2. Establecer la evolución histórica que han seguido los procesadores para llegar a los sistemas multi-núcleo actuales.

3. Debatir los errores más comunes que se cometen al momento de evaluar el rendimiento de los procesadores multi-núcleo, ante sus capacidades reales.

4. Debatir el papel que ejerce el sistema operativo en el aprovechamiento de la potencia del hardware de los procesadores multi – núcleo y mono - núcleo.

5. Evaluar el rol que desempeñan las aplicaciones en el aprovechamiento de la potencia del hardware de los procesadores multi – núcleo y mono - núcleo.

6. Plantear situaciones concretas en las que los procesadores multi núcleo ofrecen un rendimiento superior / inferior al que se esperaría inicialmente de ellos.

TABLA DE CONTENIDOS

RESUMEN DE LA PONENCIA

Los ambientes tecnológicos modernos están acostumbrados a que cada cierto tiempo se incremente, de manera significativa, el rendimiento de las computadoras y sus periféricos.

Por otro lado, el crecimiento de la potencia del hardware, no puede correr en paralelo al incremento de los requerimientos de hardware que demandan los Sistemas Operativos y las aplicaciones indefinidamente.

En tal sentido, uno de los últimos avances más importante de los últimos años ha sido la introducción de las tecnologías que permiten la implementación de procesadores multi - núcleo; sistemas que literalmente *"ponen a trabajar dos o más computadoras donde antes había una única computadora"*.

Sin embargo: ¿es posible que esta sucesión de avances tecnológicos se interrumpa o en algún momento?

O dicho de una forma más puntual: ¿Los procesadores más potentes del mercado pueden provocar que la ejecución de las aplicaciones, se realice de manera más lenta y menos eficiente que en los sistemas tradicionales?

Dependiendo de ciertas circunstancias en las que se desempeñan las aplicaciones y el sistema operativo, conjuntamente con los procesadores, puede darse esta situación.

Los procesadores multi - núcleo ofrecen un rendimiento óptimo, superior al de los procesadores mono - núcleo, cuando tanto el sistema operativo como las aplicaciones se diseñan utilizando hilos de procesamiento de manera intensiva, en lugar de procesos.

Bajo ciertas condiciones, no muy difíciles de alcanzar, los procesadores multi - núcleo pueden llegar a ofrecer un rendimiento inferior al esperado, frente a los procesadores mono - núcleo.

En tal sentido, esta ponencia trata de aclarar algunos de los criterios y parámetros que señalan como y por qué se presenta esta problemática.

1. OBSERVACIONES PRELIMINARES.

La evolución de los procesadores ha pasado por diversas etapas en el transcurso de los años, con el objetivo de dar el máximo de rendimiento posible. Sin entrar mucho en muchos detalles, se pueden distinguir tres etapas.

En la primera etapa, el aumento de rendimiento se obtuvo principalmente aumentando el nivel de integración del procesador. En la segunda etapa el rendimiento siguió incrementándose, aumentando la frecuencia de reloj del procesador, y en la tercera etapa, debido a problemas económicos y técnicos que dificultan seguir incrementando la frecuencia de reloj, se opta por integrar diversos núcleos (cores) en el mismo chip, que es donde se encuentra la industria actualmente. Todo, con el objeto de mejorar el rendimiento de los procesadores; el llamado “Rendimiento Pico Teórico”, que no es más que el rendimiento máximo teórico que un procesador puede alcanzar.

La tendencia de la industria de los procesadores ha sido evolucionar para conseguir un mayor rendimiento Pico Teórico gracias a: el incremento de los niveles de integración, los incrementos de la frecuencia de reloj, las mejoras en la arquitectura interna – tales como: el pipeline, memoria cache, procesamiento superescalar, y los sistemas multi-núcleo/multi-hilo. Todo, con el objeto de mejorar el desempeño de las aplicaciones desarrolladas por los programadores, con simplemente recompilarlas **[STAL06]**.

Sin embargo, hay diversos factores que provocan que esta meta no se alcance. En la siguiente ilustración, se muestra conceptualmente el comportamiento que ha seguido la industria de los procesadores, a lo largo del tiempo. En ella, se observa que la diferencia entre el Rendimiento Pico Teórico de los procesadores

y el rendimiento que se pude obtener sin esfuerzo extra del programador, es decir simplemente recompilando, no tiene un comportamiento lineal **[DOBL09]**[1].

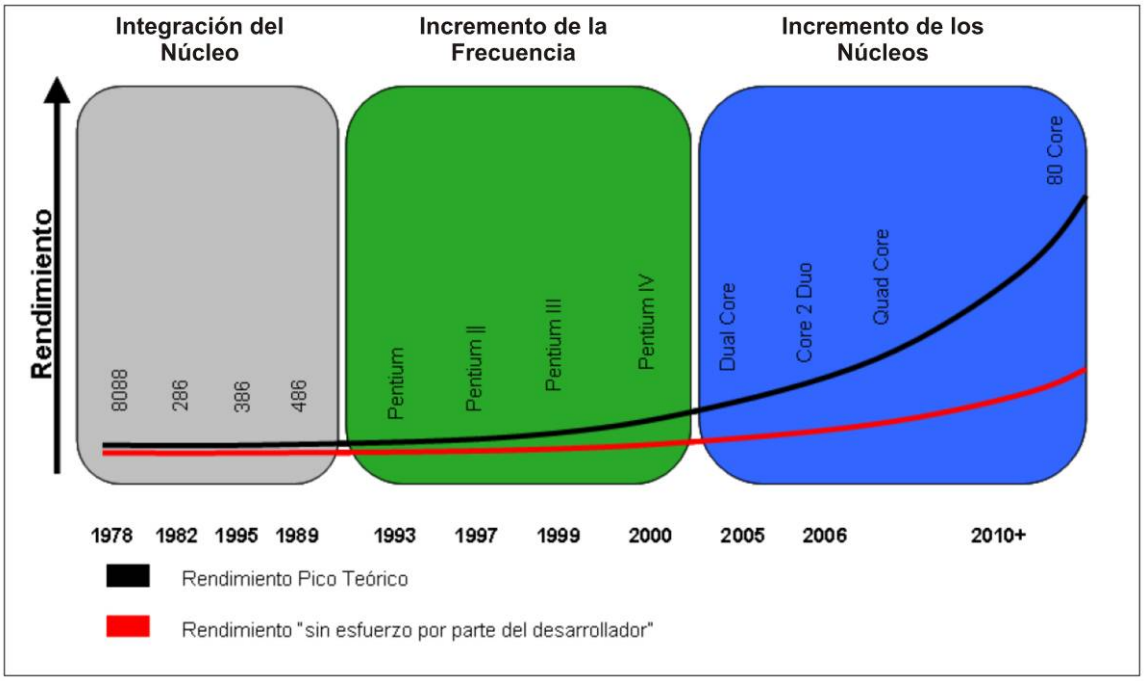

Ilustración 1: Rendimiento Pico Teórico Vs. Rendimiento Sin Esfuerzo Del Desarrollador

Lo cierto es que las mejoras aportadas por los procesadores, como pipeline, ejecución fuera de orden, ejecución especulativa, dificultan el trabajo del compilador y del programador. Esta dificultad añadida y la propia naturaleza de los problemas a implementar, obligan al programador a realizar un esfuerzo de análisis del procesador y del problema a tratar. Sin este esfuerzo extra del programador, el rendimiento obtenido no estará en concordancia con el rendimiento aportado por dichas mejoras.

---

[1] La ilustración toma como referencia a la familia de procesadores Intel, sin embargo, la tendencia en las otras familias de procesadores no es diferente.

Esta ponencia trata de aclarar algunos de los criterios y parámetros que señalan como y por qué se presenta esta problemática.

2. TAXONOMÍAS DE LA COMPUTACIÓN PARALELA.

Existen diferentes taxonomías **[HEPA07]** para clasificar las arquitecturas paralelas existentes, una de las más viejas y populares es la de Flynn.

2.1 CLASIFICACIÓN DE FLYNN.

La clasificación Flynn se basa en dos aspectos:

- **Instrucción:** Son las órdenes que debe ejecutar el procesador.
- **Datos:** son los elementos que se manipulan de acuerdo a la instrucción.

Dependiendo del número de instrucciones a ejecutar y datos a manipular simultáneamente, Flynn plantea la siguiente clasificación.

El más simple de estos sistemas es la computadora secuencial tradicional, donde una instrucción solo manipula un dato a la vez. Flynn llama a este tipo de sistema una sola instrucción, un solo dato (SISD, del inglés Single Instruction Single Data). En la siguiente ilustración se puede observar el modelo de computación SISD.

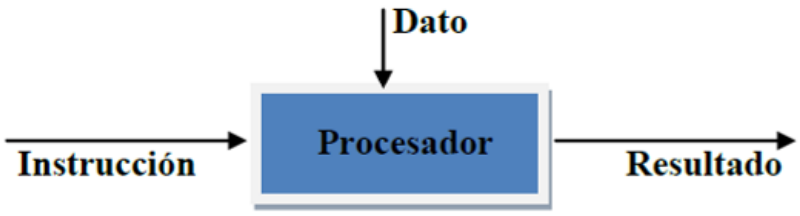

Ilustración 2: Modelo SISD

El sistema de una sola instrucción, múltiple dato (SIMD, del inglés Single Instruction Multiple data) es un sistema en el que la misma instrucción manipula diferentes datos en paralelo. Aquí el número de datos es el número de procesadores trabajando simultáneamente. En la siguiente ilustración, se puede observar el modelo SIMD.

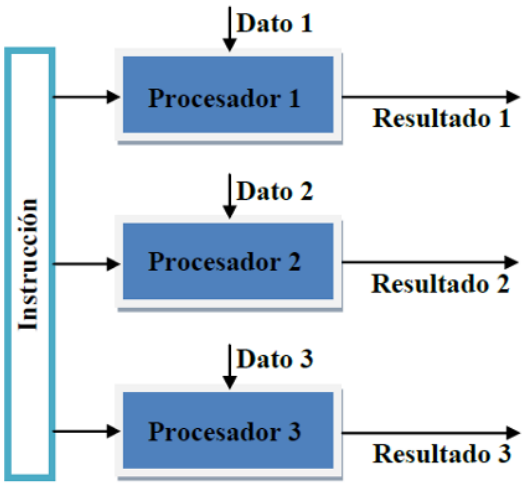

Ilustración 3: Modelo SIMD

Flynn también plantea un modelo de múltiples instrucciones, con un solo dato (MISD, del inglés Multiple Instruction Single Data), un modelo teórico de una maquina que realiza un número de operaciones diferentes con el mismo dato. En siguiente ilustración, se puede observar el modelo MISD.

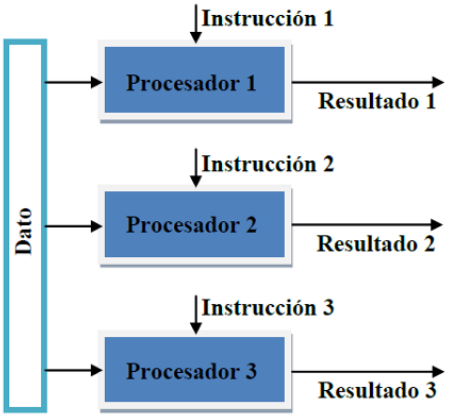

Ilustración 4: Modelo MISD

Finalmente, Flynn presenta el modelo de múltiple instrucción, múltiple dato (MIMD, del inglés Multiple Instruction Multiple Data) se refiere a un sistema multiprocesador, que es un sistema que tiene múltiples procesadores y capaz de trabajar independientemente y producir resultados para el sistema global. Cada procesador es capaz de ejecutar una instrucción diferente con un dato diferente. En la siguiente ilustración, se puede observar el modelo MIMD.

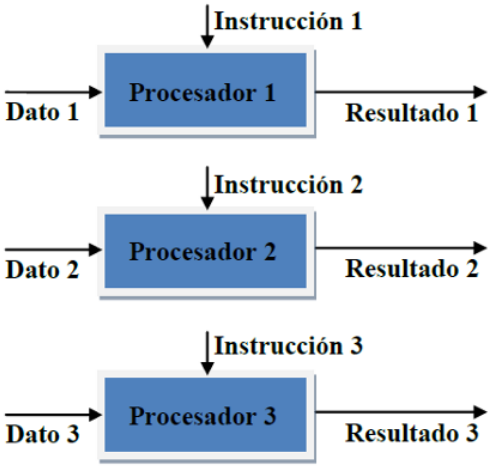

Ilustración 5: Modelo MIMD

2.2 MODELO DE MÁQUINA DE ACCESO ALEATORIO PARALELO.

En el modelo de máquina de acceso aleatorio paralelo (PRAM, del ingles Parallel Random Access Machine) **[HEPA07]**, todos los procesadores están conectados en paralelo con la memoria global. Esta memoria es compartida con todos los procesadores. Este modelo se puede observar en la siguiente ilustración.

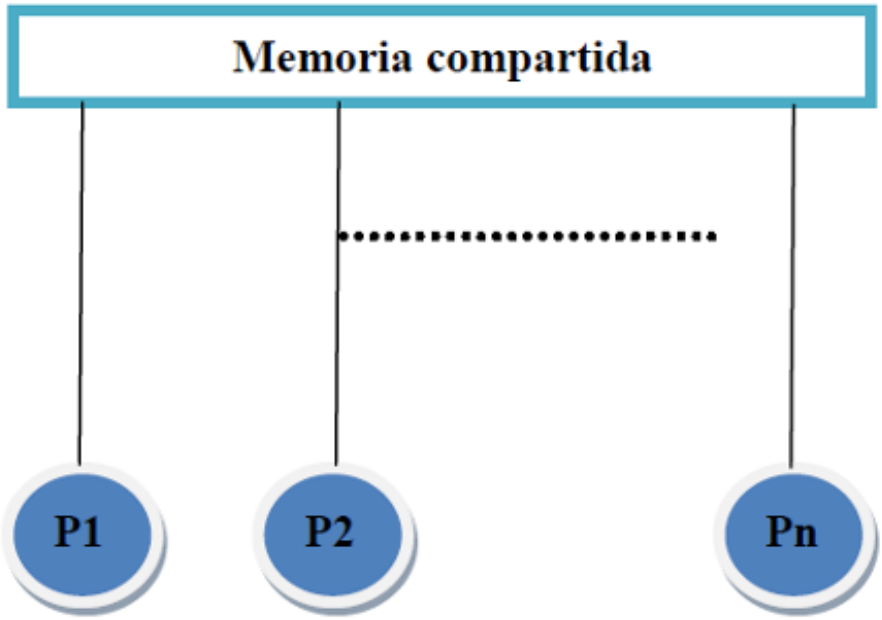

Ilustración 6: Modelo PRAM

Todos los procesadores trabajan síncronamente con un reloj común. Cada procesador es capaz de acceder (lectura/escritura) a la memoria. La comunicación entre procesadores se concreta a través de la memoria. Esto significa que el dato de un procesador *Pi* es comunicado a otro procesador *Pj* siguiendo los pasos siguientes:

- El procesador *Pi* escribe el dato en la memoria global.
- El procesador *Pj* lee el dato de la memoria global.

En el modelo PRAM existen 4 tipos diferentes de arquitecturas, dependiendo la capacidad de que más de un procesador pueda leer/escribir a una localidad de memoria.

- Lectura exclusiva/escritura exclusiva PRAM (EREW).
- Lectura concurrente/escritura exclusiva PRAM (CREW).
- Lectura exclusiva/escritura concurrente PRAM (ERCW).
- Lectura concurrente/escritura concurrente PRAM (CRCW).

El modelo PRAM de lectura exclusiva/escritura exclusiva (EREW) permite que en un instante dado solo un procesador pueda leer/escribir a una localidad de memoria. Por lo tanto, la lectura o escritura simultánea a una localidad de memoria por más de un procesador no es permitida.

Por otro lado, en el modelo PRAM-CREW se permite la lectura concurrente a una localidad de memoria por más de un procesador, pero no permite una escritura concurrente.

En el modelo PRAM-ERCW la lectura se realiza de manera exclusiva, en tanto que se permite la escritura concurrente.

El modelo más eficiente es el PRAM-CRCW ya que permite una lectura concurrente, lo mismo que una escritura concurrente a una localidad de memoria.

Cuando uno o más procesadores intentan leer el contenido de una localidad de memoria concurrentemente, se asume que todos los procesadores tienen éxito en su lectura. Sin embargo, cuando más de un procesador intenta escribir a la misma localidad de memoria concurrentemente, se presenta un conflicto que se

debe resolver apropiadamente. Tres métodos diferentes se han sugerido para resolver el conflicto:

- Resolución de conflicto por igualdad (ECR).
- Resolución de conflicto por prioridad (PCR).
- Resolución de conflicto arbitrario (ACR).

En el caso de ECR, se asume que el procesador tiene una escritura satisfactoria, solo si todos los procesadores intentaron escribir el mismo valor en la localidad de memoria.

En PCR se asume que cada procesador tiene un nivel de prioridad. Cuando más de un procesador intenta escribir a la misma localidad de memoria simultáneamente, el procesador con mayor prioridad es el que escribe.

En ACR se asume que entre los procesadores que intentan escribir simultáneamente, solo uno de ellos logra escribir, y lo hará de manera aleatoria.

La ilustración que se muestra a continuación, ilustra a las arquitecturas que se pueden derivar de la taxonomía PRAM.

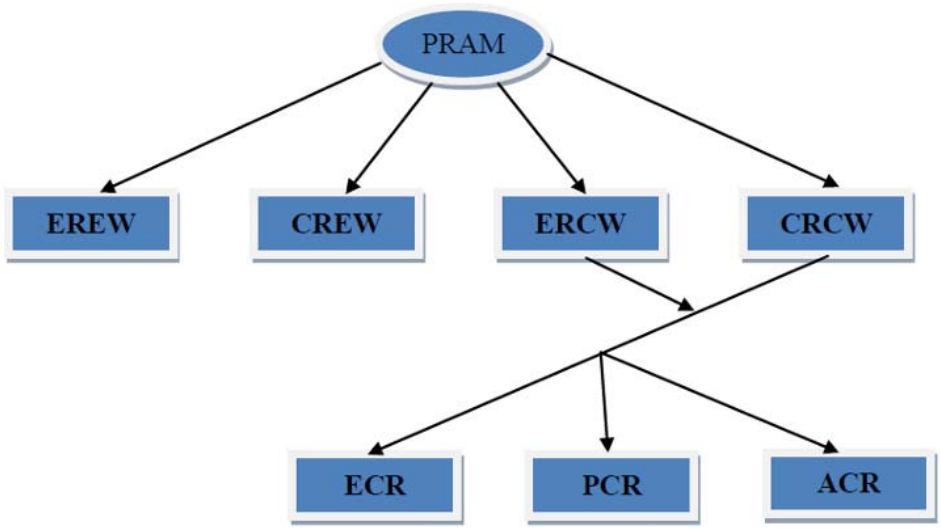

Ilustración 7: Arquitecturas Derivadas Del Modelo PRAM

3. EVOLUCIÓN DEL PROCESADOR MONO-HILADO.

A continuación, se procederá a describir al procesador clásico y sus conceptos asociados, para explicar posteriormente los procesadores multi-hilado y multi-núcleo, así como los problemas para conseguir alto rendimiento en estos sistemas **[STAL06]**.

3.1 PROCESADOR CLÁSICO.

En el procesador clásico, se puede dividir el proceso de ejecución de una instrucción en cuatro etapas **[STAL06]**:

- (BI) búsqueda de la instrucción (prefetch y fetch).
- (DI) decodificación de la instrucción.
- (BO) búsqueda de operandos
- (EI) ejecución de la instrucción (ejecución y escritura de los resultados).

En la siguiente Ilustración, se muestra la ejecución secuencial de dos instrucciones en un procesador clásico. Se puede apreciar que hasta que la primera instrucción no finaliza, no se puede comenzar la búsqueda de la segunda instrucción.

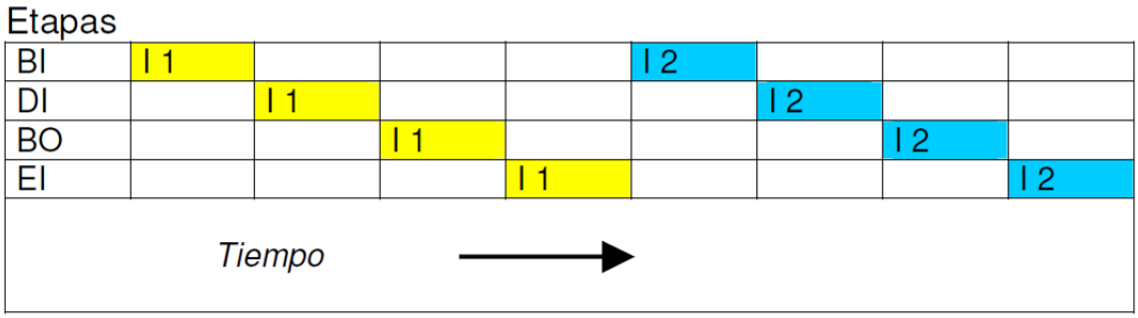

Ilustración 8: Ejecución De Instrucciones En Un Procesador Clásico

El tiempo de ejecución de un programa se puede expresar con la siguiente ecuación de rendimiento:

$$T_{ej} = N \times CPI \times t$$

Donde:

- N = número de instrucciones de un programa. Depende de la arquitectura del computador a través del Repertorio de Instrucciones.
- CPI = número medio de Ciclos Por Instrucción.
- t = tiempo de un ciclo de instrucción.

Para reducir el tiempo de ejecución en esta ecuación independientemente del nivel de integración, existen dos alternativas que son, reducir el número de instrucciones y/o reducir el CPI. Existen dos tipos de filosofías en las arquitecturas de procesadores, que atacan el problema desde distintos puntos de vista.

- **RISC (Computadoras con un conjunto de instrucciones reducido):** Implementación Hardware más simple, por tanto más rápida y eficiente. Se basa en disponer de un repertorio de instrucciones reducido, permitiendo su implementación por hardware. Los programas tendrán un número elevado de instrucciones pero prácticamente la totalidad de ellas se ejecutarán en un ciclo de reloj. En este tipo de arquitectura hay una fuerte dependencia del compilador en la generación eficiente de código máquina.

- **CISC (Computadoras con un conjunto de instrucciones complejo):** Implementación Hardware más compleja y por tanto más lenta e ineficiente. Se basa en disponer de un repertorio de instrucciones amplio y complejo. En ocasiones la ejecución de algunas instrucciones se implementa de forma micro-programada, lo que significa que cada instrucción máquina es interpretada por un micro-programa localizado en una memoria, en el circuito integrado del procesador. El número de instrucciones de un programa es menor que en el RISC, pero el CPI suele ser mayor.

3.1.1 PROCESAMIENTO SEGMENTADO (PIPELINE).

En el procesamiento segmentado se adopta una nueva estrategia con el objetivo de disminuir el tiempo medio de ejecución por instrucción de una aplicación. Se divide internamente el computador en segmentos individuales, cada uno especializado en una de las etapas **[HEPA07]**.

A diferencia del procesador clásico donde todas las etapas tenían que completarse antes de buscar la instrucción siguiente, ahora la existencia de segmentos especializados permite el solapamiento en la ejecución de las instrucciones. Así, un segmento puede empezar a trabajar con una nueva

instrucción sin la necesidad de que la instrucción anterior haya finalizado todas las etapas.

El resultado es un aumento del número de instrucciones ejecutadas por ciclo. Como muestra en la siguiente ilustración, con la ejecución segmentada de instrucciones, un procesador segmentado de cuatro etapas pasa de ejecutar una instrucción cada cuatro ciclos, a una instrucción por ciclo.

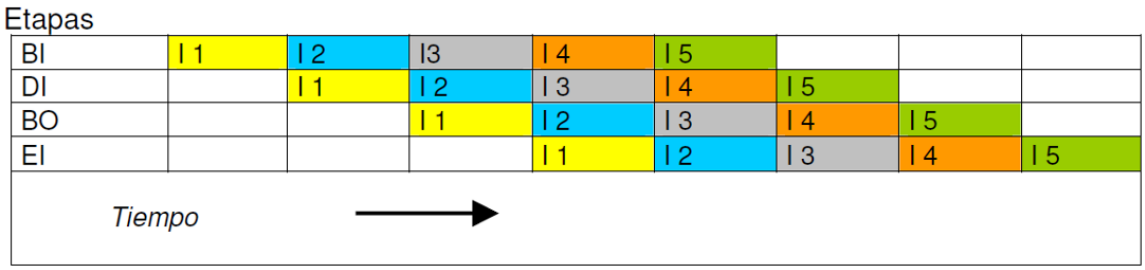

Ilustración 9: Ejecución Segmentada De Instrucciones

No obstante, los saltos y las dependencias de datos limitan esta ganancia.

En el primer caso las instrucciones de bifurcación condicional pueden derivar en diferentes caminos. Por tanto no es posible conocer la siguiente instrucción hasta que no se haya ejecutado por completo la instrucción de bifurcación.

En el segundo caso una instrucción podría quedar detenida en una etapa a la espera del dato que otra instrucción anterior debe calcular.

3.1.2 MEMORIA CACHE.

La memoria cache es una memoria de alta velocidad y menor tamaño que la memoria principal, situada entre el procesador y la memoria principal. Su misión es almacenar temporalmente los contenidos de la memoria principal que estén siendo utilizados por el procesador. De esta forma se reduce el número de

operaciones de acceso a memoria principal, disminuyendo el número de ciclos dedicados a acceso a memoria, y reduciéndose por tanto el CPI **[STAL06]**.

La posibilidad de aumentar las prestaciones incorporando memoria cache se debe a la localidad temporal y a la localidad espacial. Estas son características del comportamiento de las aplicaciones.

En el caso de la localidad temporal existe una alta probabilidad de acceder a posiciones de memoria a las que ya se ha accedido anteriormente.

En el caso de la localidad espacial existe una alta probabilidad que de una vez accedida a una posición de memoria, se acceda a las posiciones de memoria cercanas, en un futuro inmediato.

3.1.3 PROCESADORES SUPERESCALARES.

Un procesador superescalar es capaz de ejecutar más de una instrucción en cada etapa del pipeline del procesador. El número máximo de instrucciones en cada etapa depende del número y del tipo de las unidades funcionales de que disponga el procesador **[HEPA07]**.

Sin embargo, un procesador superescalar sólo es capaz de ejecutar más de una instrucción simultáneamente si las instrucciones no presentan ningún tipo de dependencia. Las dependencias que pueden aparecer son:

- **Estructurales:** cuando dos instrucciones requieren el mismo tipo de unidad funcional pero su número no es suficiente.
- **De datos:**

- **Lectura después de Escritura:** cuando una instrucción necesita el resultado de otra para ejecutarse.

- **Escritura después de Lectura o Escritura:** cuando una instrucción necesita escribir en un registro sobre el que otra instrucción previamente debe leer o escribir.

- **De control:** cuando existe una instrucción de salto que puede variar la ejecución de la aplicación.

Se pueden distinguir diferentes tipos de procesadores por la forma de actuar ante una dependencia estructural o de datos. En un procesador con ejecución en orden las instrucciones quedarán paradas a la espera de que se resuelva la dependencia. Mientras que en un procesador con ejecución fuera de orden las instrucciones dependientes quedarán paradas pero será posible solapar parte de la espera con la ejecución de otras instrucciones independientes que vayan detrás.

En el caso de las dependencias de control, se conoce como ejecución especulativa de instrucciones a la ejecución de instrucciones posteriores a la instrucción de salto (antes de que el PC llegue a la instrucción de salto).

3.2 LIMITACIONES DEL PROCESADOR MONO-HILADO.

A continuación, se describirán una serie de factores que limitan el rendimiento de la ejecución de una aplicación en un procesador mono-hilado **[HEPA07]**.

3.2.1 PROBLEMA DE LA MEMORIA.

La diferencia de velocidad entre procesador y memoria, limita el rendimiento del procesador. Las operaciones de memoria son lentas comparadas con la velocidad del procesador. Los accesos a memoria, por ejemplo en un fallo de cache, pueden consumir de 100 a 1000 ciclos de reloj, durante los cuales el procesador debe esperar a que el acceso a memoria finalice. La ilustración que se muestra a continuación, se pueden observar las fases de ejecución de un programa mono-hilado, ejecutándose sobre un procesador con una frecuencia de funcionamiento dada **[HEPA07]**.

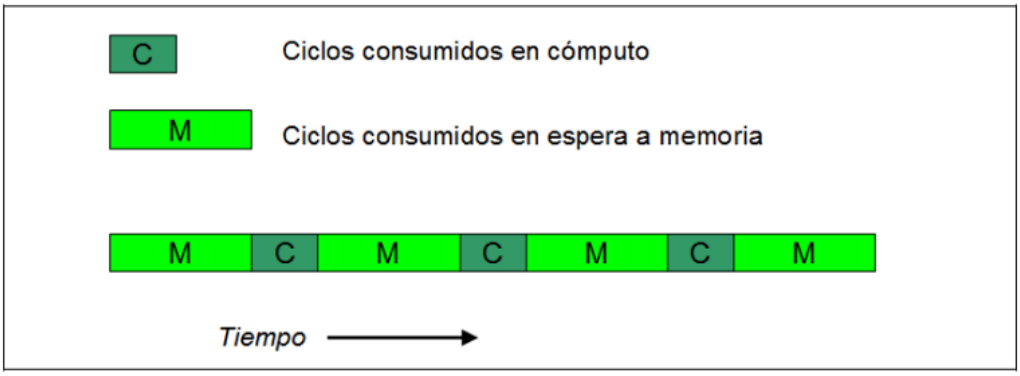

Ilustración 10: Fases De La Ejecución De Un Programa Mono-Hilado

Un aumento de la frecuencia de reloj del procesador sin incrementar la velocidad de la memoria solamente mejoraría el rendimiento en un pequeño porcentaje. Los ciclos de cómputo se realizarían más rápido pero el tiempo de acceso a memoria continuaría siendo el mismo.

Esto se puede apreciar en la ilustración que se presentan a continuación, donde se observan las fases de ejecución de un programa mono-hilado con mayor frecuencia de funcionamiento que el de la Ilustración 10, pero con la velocidad de memoria.

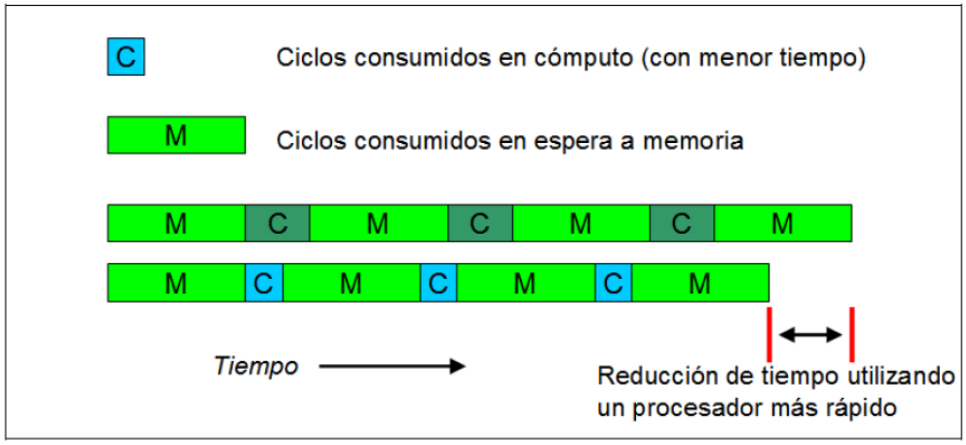

Ilustración 11: Fases De La Ejecución De Un Programa Mono-Hilado Con Un Procesador Más Rápido Que El De La Ilustración 10, Pero Con La Misma Velocidad De Memoria

Además hay que tener en cuenta que a los cientos de ciclos que se consumen en cada acceso a memoria hay que sumarle decenas de ciclos extra, por cada acceso a nuevos niveles de cache (provocados por fallos en el nivel anterior).

La solución para aprovechar los ciclos en los que el procesador esta esperando a que finalice la operación de memoria es el multithreading por hardware. El multithreading por hardware es una propiedad que permite al procesador alternar de un hilo a otro hilo cuando el hilo que esta ocupando el procesador queda parado.

3.2.2 CALOR Y COSTE ASOCIADO.

El incremento de la frecuencia de reloj del procesador implica un aumento de la potencia consumida y del calor generado. En la actualidad los altos valores de frecuencia de reloj de los procesadores suponen un problema, tanto económico (consumo eléctrico, y gasto dedicado a la disipación del calor y refrigeración), como tecnológico (dificultad para disipar la gran cantidad de calor generado, de la pequeña superficie de un procesador) **[HEPA07]**.

Por estos motivos, se abandona la idea de aumentar la frecuencia de reloj del procesador para aumentar el rendimiento, y se opta por añadir más procesadores en el mismo chip. Con esta solución el calor se incrementa de forma lineal y no exponencial como ocurre con el aumento de frecuencia de reloj.

4. EL PROCESADOR MULTI-HILADO Y MULTI-NÚCLEO.

Para comprender los conceptos como el Multithreading por Software, el Multithreading por Hardware y las ventajas de los procesadores multi-núcleo es necesario conocer la diferencia entre hilo y proceso.

4.1 HILO Y PROCESO.

Un hilo o Thread es una secuencia de código ejecutable. Los hilos existen dentro de un proceso y comparten recursos como el espacio de memoria, la pila de ejecución y el estado de la CPU **[SGGA06]**.

Un proceso con múltiples hilos tiene tantos flujos de control como hilos. Cada hilo se ejecuta con su propia secuencia de instrucciones de forma concurrente e independiente.

Un proceso es una instancia de un programa formado por uno o más hilos, con su propio espacio de direccionamiento.

4.1.1 MULTI-THREADING POR SOFTWARE.

En primera instancia, se debe distinguir entre multi-programación y aplicaciones paralelas.

4.1.1.1 MULTI-PROGRAMACIÓN.

La multi-programación es una técnica de planificación que permite tener varios hilos (o procesos) en estado de ejecución. Los hilos comparten los recursos del sistema como la memoria principal y el procesador. Existe la falsa apariencia de que los hilos se están ejecutando simultáneamente, esto es debido a que el sistema operativo es de tiempo compartido. En un escenario de tiempo compartido cada hilo se ejecuta durante un breve intervalo de tiempo. El cambio de contexto se realiza lo suficientemente rápido como para simular la ejecución de varios hilos simultáneamente **[LANC00]**.

En la siguiente ilustración, se observa como el thread 1, se para al producirse una excepción que requiere esperar un tiempo. Inmediatamente el Sistema Operativo cambia de contexto y ejecuta otro thread. Cada columna de la figura representa una instrucción de las cuatro que se pueden llegar a ejecutar en cada ciclo (ejecución superescalar de grado 4).

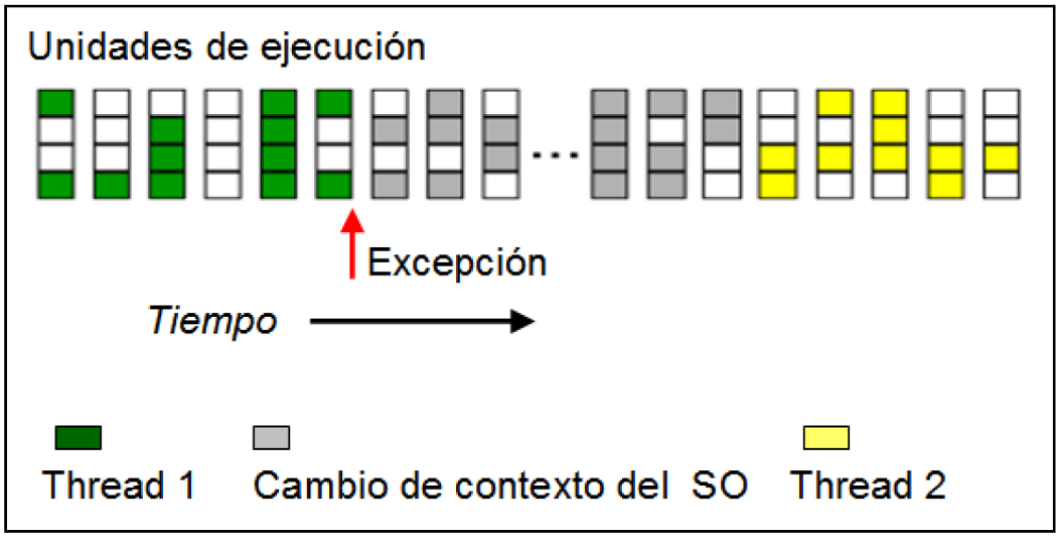

Ilustración 12: Cambio De Thread Del So En Un Entorno Multiprogramado

4.1.1.2 APLICACIONES PARALELAS.

El multi-threading por Software posibilita la realización de aplicaciones paralelas. Es un nuevo modelo de programación que permite a múltiples hilos existir dentro de un proceso.

Los hilos comparten los recursos del proceso pero se ejecutan de forma independiente. El hecho de que sean independientes permite la concurrencia (es decir su ejecución simultanea), y si el procesador lo soporta se podrán ejecutar en paralelo **[LANC00]**.

Las ventajas de realizar la concurrencia a nivel de hilo en lugar de a nivel de proceso son varias:

- Los hilos se encuentran todos dentro de un mismo proceso y por lo tanto pueden compartir los datos globales.
- Además, una petición bloqueante de un hilo no parará la ejecución de otro hilo.
- Por último, si el procesador lo soporta, los diferentes threads están asociados a diferentes conjuntos de registros, por lo que el cambio de contexto del procesador podrá realizase de forma eficiente.

4.1.2 MULTI-THREADING POR HARDWARE.

El multi-threading por Hardware es una técnica que incrementa la utilización de los recursos del procesador. A continuación se analizarán diferentes tipos de multi-threading por Hardware.

4.1.2.1 LARGE-GRAIN MULTI-THREADING.

En el modelo Large-Grain Multithreading el procesador ejecuta el thread de forma habitual y solamente realiza un cambio de contexto cuando ocurre un evento de larga duración (como un fallo de caché). Para que el cambio de contexto sea eficiente es necesario que exista una copia del estado de la arquitectura (PC, registros visibles) para cada thread. Este método tiene la ventaja de ser sencillo de implementar **[HEPA07]**.

En la ilustración que se presenta a continuación, se muestra la ejecución de los diferentes hilos de ejecución o threads, con respecto al tiempo. En dicha ejecución, se puede observar como el procesador cambia de thread cuando se produce un evento que requiere un tiempo de espera por ejemplo un fallo de cache.

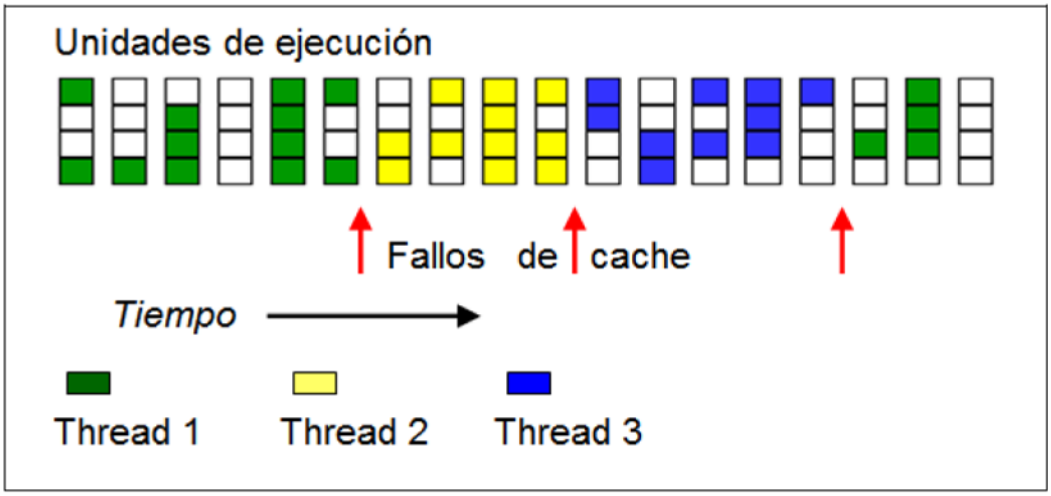

Ilustración 13: Ejecución De Threads Según El Modelo Large-Grain Multithreading

4.1.2.2 FINE-GRAINED MULTITHREADING.

Fine-grained Multi-threading se basa en un cambio “rápido” entre threads, ejecutando en cada ciclo un thread diferente. Es un mecanismo que tiene como base una planificación de la ejecución de las instrucciones en orden. Con el fin de evitar largas latencias por threads bloqueados, se ejecutan instrucciones de diferentes threads. Este enfoque tiene la ventaja de eliminar las dependencias de datos que paran el procesador. Al pertenecer las instrucciones a diferentes threads, las dependencias de datos y de control desaparecen.

En la ilustración que se observa a continuación, se muestra como en cada ciclo se ejecutan instrucciones de threads diferentes. El procesador no espera a que se produzca una interrupción (fallo de caché,....) para saltar al siguiente thread.

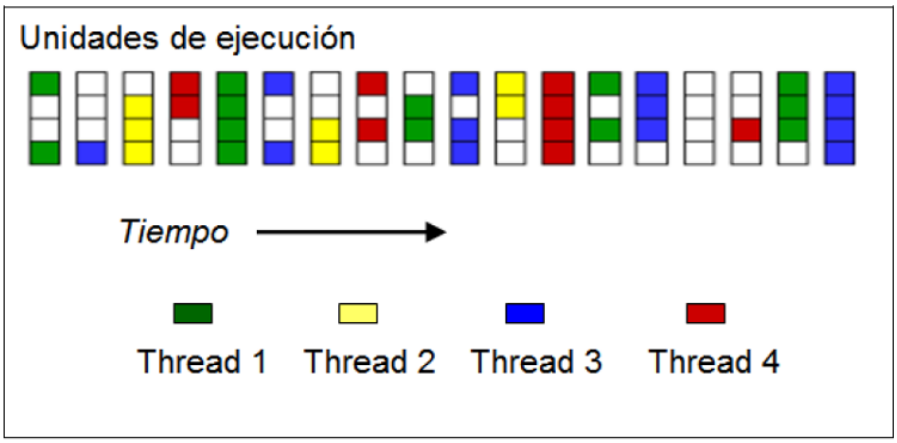

Ilustración 14: Ejecución De Threads Según El Modelo Fine-Grain Multithreading

4.1.2.3 SIMULTANEOUS MULTI-THREADING.

La filosofía de Simultaneous Multi-threading (SMT) consiste en poder ejecutar instrucciones de diferentes threads, en cualquier momento y en cualquier unidad de ejecución. Desarrollar esta tecnología requiere un hardware adicional para toda la lógica. Como consecuencia, su realización para un gran número de

threads aumentaría la complejidad y, por tanto el costo del hardware. Por este motivo en las implementaciones SMT se opta por reducir el número de threads **[HEPA07]**.

Simultaneous Multi-threading muestra a un procesador físico como dos o más procesadores lógicos. Los recursos físicos son compartidos y el estado de la arquitectura es copiada para cada uno de los dos procesadores lógicos. El estado de la arquitectura está formado por un conjunto de registros: registros de propósito general, registros de control, registros del controlador de interrupciones (APIC) y registros de estado.

Los programas verán a los procesadores lógicos como si se tratara de dos o más procesadores físicos diferentes. Sin embargo, desde el punto de vista de la microarquitectura, las instrucciones de los procesadores lógicos se ejecutarán simultáneamente compartiendo los recursos físicos.

4.2 EL PROCESADOR MULTI-NÚCLEO.

Los procesadores multi-núcleo o multi-core combinan dos o más procesadores (a los que nos referiremos como núcleos o cores) en un mismo chip. Estos procesadores mejoran el rendimiento de las aplicaciones paralelas. Cómo ya se mencionó, las aplicaciones paralelas están compuestas por múltiples threads independientes, de forma que es posible la concurrencia. Es decir, los threads se pueden ejecutar al mismo tiempo y en paralelo **[HEPA07]**.

Otra de las ventajas de esta tecnología es poder aplicar el diseño de un núcleo de procesamiento ya probado a los demás cores del procesador. Así se evita el coste que supondría el diseño de nuevos núcleos.

Como inconveniente de esta tecnología, las aplicaciones han de ser correctamente paralelizadas para aprovechar todo el potencial de los procesadores multi-core.

4.2.1 ARQUITECTURAS MULTI-PROCESADOR.

Los procesadores de una arquitectura multi-procesador comparten la memoria principal - es decir, se acoplan a la clasificación PRAM -. Existen dos alternativas para compartir la memoria principal **[HEPA07]**.

- En la primera los procesadores del sistema tienen el mismo tiempo de acceso a memoria, a esta arquitectura se la conoce como arquitectura U.M.A (Acceso uniforme a memoria).
- En la segunda el acceso parcial o total a la memoria es controlado por un único procesador, lo que provoca que este procesador tenga un tiempo de acceso menor, a la memoria controlada por él, que el resto de procesadores. El resto de procesadores debe interactuar con el procesador que controla la memoria para acceder a élla, a esta arquitectura se la conoce como arquitectura N.U.M.A. (Acceso no uniforme a memoria).

En cuanto a la organización de las caches de un multi-core hay varias opciones. En algunas arquitecturas se opta por mantener todos los niveles de cache privados a cada core, mientras que en otras arquitecturas se comparte el último nivel de cache. Estas arquitecturas se ilustrarán gráficamente.

Así, en la ilustración que se observa a continuación, se muestra el esquema de un procesador dual-core, en el que las memorias cache de nivel 1 de

instrucciones y de datos son privadas, la memoria cache de nivel 2 es compartida, y el acceso a memoria principal es de tipo U.M.A.

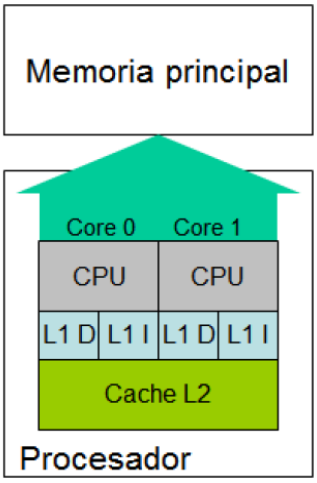

Ilustración 15: Esquema De Procesador Dual-Core Con Cache De Nivel 2 Compartida

Por otro lado, la ilustración que se muestra a continuación, presenta el esquema de un procesador quad-core, en el que las memorias cache de nivel 1 y de nivel 2 son privadas, la memoria cache de nivel 3 es compartida, y el acceso a memoria principal es de tipo U.M.A.

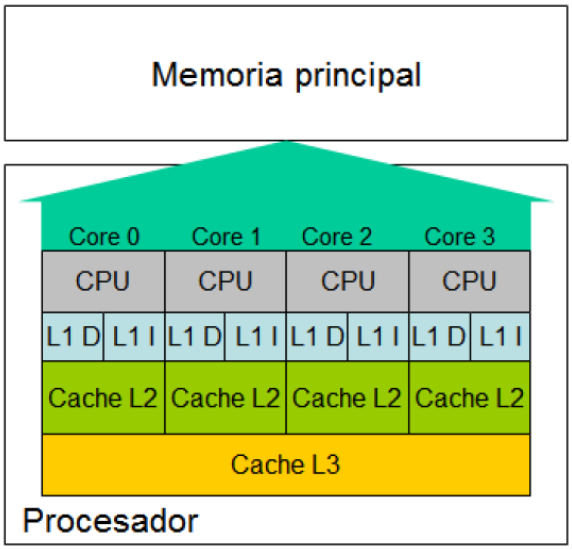

Ilustración 16: Esquema De Procesador Quad-Core Con Cache De Nivel 3 Compartida

Finalmente, la siguiente ilustración muestra el esquema de una arquitectura N.U.M.A, formada por dos procesadores dual-core. Cada uno de los procesadores accede directamente a su memoria principal y para acceder a la memoria principal del otro procesador tiene que interactuar con él. Como en la Ilustración 15, los cores de cada uno de los procesadores tienen caches de nivel 1 privadas y comparten la memoria cache de nivel 2 y la memoria principal que corresponde al procesador.

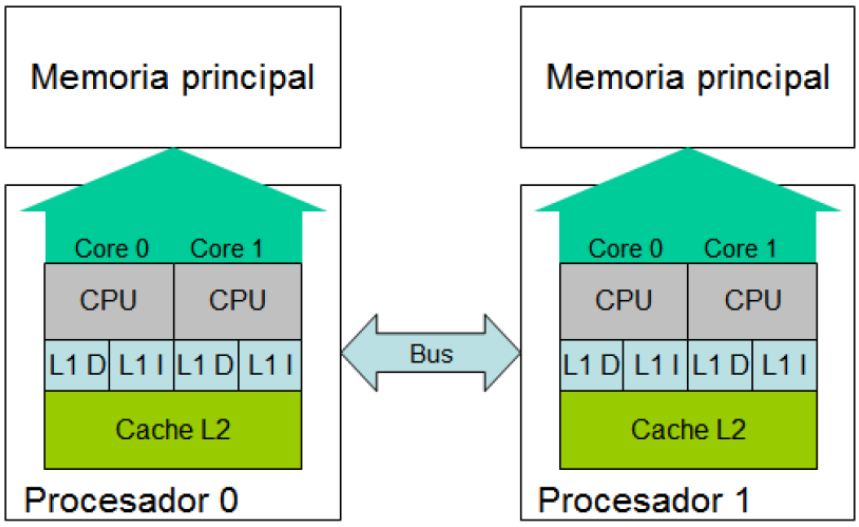

Ilustración 17: Arquitectura N.U.M.A con dos procesadores dual-core

Las ventajas de compartir la memoria cache son las siguientes:

- **Uso eficiente del último nivel de cache:**
  - Si un core está inactivo el otro core puede utilizar toda la cache compartida.
  - Si los dos cores trabajan en paralelo, la memoria cache se reparte proporcionalmente en función de la frecuencia de solicitudes de cada core. Así, se incrementa la utilización de los recursos.

- **Reducción del almacenamiento de los datos:** Los mismos datos utilizados por dos cores que comparten memoria cache, serán almacenados una única vez.

- **Reducción de la complejidad de la coherencia en la memoria cache:**

  - Reduce el problema de *false-sharing.*

  - Menor carga de trabajo para mantener la coherencia, comparado con la arquitectura de memoria cache privada.

5. FACTORES QUE IMPIDEN UN ALTO RENDIMIENTO EN SISTEMAS MULTICORE-MULTITHREAD.

Por todo lo analizado hasta el momento parece que la solución a la necesidad de seguir aumentando el rendimiento en los procesadores, simplemente se puede resolver aumentando el número de cores de un procesador y/o aumentando el número de procesadores, ya sean estos UMA o NUMA **[DOBL09]**.

Si el rendimiento de las aplicaciones en entornos multi-core/multi-thread escalara linealmente al aumentar el número de threads con los que se ejecuta la aplicación, el problema estaría resuelto. Pero lo cierto es que en este tipo de entornos, existen una serie de factores que impiden que esta escalabilidad sea lineal. Entre ellos, resaltan:

- El overhead – sobrecarga del sistema - por creación/eliminación de threads.

- El posible desbalance (en las aplicaciones) del volumen de cómputo por thread (threads esperando a que otros threads finalicen).

- La comunicación entre las memorias de los procesadores.

Estos factores serán descritos, a continuación, con mayor detalle.

5.1 OVERHEADS POR CREACIÓN/ELIMINACIÓN DE THREADS.

Como muestra en la ilustración que se observa a continuación, la creación y posterior eliminación de los threads que trabajan en paralelo, tiene un coste en tiempo (overhead). La importancia de este overhead dependerá de la relación entre el (tiempo total de ejecución del bucle con un thread / nº de threads) y (tiempo que se tarda en crear los threads, repartir el trabajo, recoger el resultado y eliminar los threads)

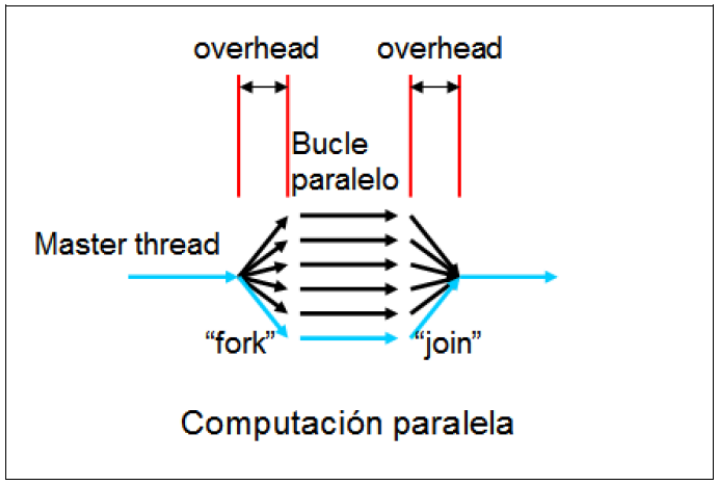

Ilustración 18: Paradigma Fork-Join Con Overheads Por Creación/Eliminación De Threads

El overhead ha de ser pequeño en comparación con (tiempo de ejecución / n threads). Se está suponiendo que el volumen de cómputo se reparte de forma equitativa. En el apartado siguiente se analiza el caso en el que esto no es así.

5.2 DESBALANCEO (EN LAS APLICACIONES).

Una incorrecta distribución del volumen de cómputo por thread implicará que algunos threads finalicen su trabajo antes que otros. Por lo que los threads que han finalizado tendrán que esperar. Esta espera supone un coste en ciclos de reloj del procesador desaprovechados y por lo tanto un overhead.

Esta sobrecarga se puede reducir aunque no eliminar totalmente, asignando el trabajo dinámicamente entre los diferentes threads que participan en la ejecución. Por contrapartida se genera otro pequeño overhead asociado al cómputo necesario para gestionar la asignación dinámica.

En el diseño de aplicaciones paralelas es muy importante una óptima asignación del trabajo a realizar a cada uno de los threads.

5.3 LA COMUNICACIÓN ENTRE LAS MEMORIAS DE LOS CORES.

Los threads de una ejecución multi-thread pueden trabajar de manera independiente y con datos independientes. Pero por las características de las aplicaciones, en algún momento necesitarán intercambiar datos.

Este intercambio de datos se realiza de manera transparente al thread ya que éste únicamente accederá a unas posiciones de memoria que previamente otro thread habrá modificado.

Sin embargo, aunque este intercambio es transparente para el thread, no esta libre de coste en tiempo. Los datos que hayan sido modificados en la cache de un thread tendrán que ser copiados a la cache del thread que los necesita en ese momento.

Hay que tener en cuenta que el coste de comunicar datos modificados por threads que se ejecutan dentro de un mismo procesador es muy inferior al coste de comunicar datos entre threads que se ejecutan en cores de diferentes procesadores.

Estos costes suponen un overhead a considerar a la hora de diseñar una aplicación multithread. Habrá que prestar especial atención a la localidad temporal y espacial de la aplicación y a la descomposición por dominio, intentando primero minimizar las comunicaciones entre threads que se ejecutan en distintos procesadores y segundo minimizar las comunicaciones entre threads que se ejecutan en el mismo procesador.

6. RESULTADOS PRÁCTICOS.

Experimentalmente, se ha comprobado que el rendimiento (en un procesador SPARC T2) escala prácticamente linealmente hasta 16 threads, perdiendo progresivamente hasta un 50% de eficacia por hilo al ir incrementando el número de threads hasta 32 **[DOBL09]**, que no es el comportamiento que idealmente se esperaría de este tipo de tecnología.

El experimento en si, consistió en modelar el problema de los N-cuerpos en su caso más simple, es decir utilizando únicamente dos partículas.

Lo preocupante de estos resultados es que: si en una situación controlada de laboratorio se obtuvieron estos resultados, en situaciones reales de cómputo paralelo, donde muchas aplicaciones compitan por los recursos del sistema, sólo se puede esperar que los niveles de rendimiento de estos procesadores disminuyan significativamente más.

7. CONCLUSIONES

- El modelo actual de crecimiento de la potencia del hardware y el incremento de los requerimientos mínimos del sistema que demandan las aplicaciones actuales no puede ser mantenido de manera indefinida, porque las tecnologías actuales están llegando a situaciones límite, donde simplemente no pueden mejorarse.

- Los procesadores Multi núcleo ofrecen un rendimiento óptimo, cuando: se emplean pocos núcleo de procesamiento, se ejecutan pocos procesos simultáneamente, y tanto los sistema operativos como las aplicaciones han sido diseñadas explícitamente para ejecutar se de manera concurrente.

- El incremento indiscriminado de la cantidad de núcleo de procesamiento, así como de los procesos en ejecución en un momento dado, puede llevar a que el rendimiento de los procesadores Multi núcleo se ubique por debajo del lo que se esperaría de ellos.

- El dominio inadecuado de las técnicas de programación concurrente al momento de diseñar el implementar aplicaciones y sistemas operativos, puede llevar a que se implementen sistemas que, en la práctica, funcionarán y eficientemente en el hardware basado en procesadores Multi núcleo.

8. REFERENCIAS BIBLIOGRÁFICAS

[CANC09] **CANCINO, Rodolfo Jiménez.** *Análisis Del Impacto De Arquitecturas Multi-Núcleo En Cómputo Paralelo.* Tesina. Instituto Politécnico Nacional, Centro De Investigación Y Desarrollo De Tecnología Digital, México, 2009.

[DOBL09] **DOBLADO RUESTA, Héctor Manuel.** *Análisis De Rendimiento De Aplicaciones Paralelas De Memoria Compartida: Problema N-Body.* Tesis. Universidad Autónoma De Barcelona, España, 2009.

[HEPA07] **HENNESSY, John; PATTERSON, David.** *Computer Arquitecture: A Quantitative Approach.* Cuarta Edición, Morgan Kaufman, EUA, 2007.

[LANC00] **LANCHARES DÁVILA, Juan.** *Apuntes De Estructura De Computadores.* Departamento De Arquitectura De Computadores Y Automática. Universidad Complutense De Madrid. 2000.

[MOOR65] **MOORE, Gordon E.** *Cramming more components onto integrated circuits.* Electronics Magazine, Volumen 38, Número 8, publicado el 19 de abril de 1965.

[MUEL10] **MUELLER, Scott.** *Upgrading and Repairing PCs.* Décimo Novena Edición. Pearson Education, EUA, 2010.

[SGGA06] **SILBERSHATZ, Abraham; GALVIN, Peter Baer; GAGNE, Greg.** *Fundamentos De Sistemas Operativos.* Séptima Edición, McGraw-Hill, España, 2006.

[STAL06] **STALLINGS, William.** *Organización y Arquitectura de Computadoras. Principios de Estructura y de Funcionamiento.* Séptima Edición, Pearson Prentice-Hall, España, 2006.

[STAL98] **STALLINGS, William.** *Sistemas Operativos.* Segunda Edición, Pearson Prentice-Hall, España, 1998.

[YUNT09] **YUNTA, Javier Boba.** *Análisis De Prestaciones De Procesadores Multi-Core y Multi-Thread.* Tesis. Universidad Autónoma De Barcelona, España, 2009.