"Año de la universalización de la salud".

Universidad Nacional de San Agustín de Arequipa

Escuela Profesional de Ciencia de la Computación

Computación Paralela y Distribuida.

MODIFICACIONES AL MODELO VON NEUMANN

Alumnos:
Miguel Alexander, Herrera Cooper

Docente :
Dr. Alvaro Mamani Aliaga

17 de septiembre del 2020

Índice

Índice de figuras

1. Memoria Cache

1.1. Definición

> La memoria caché es una memoria de tamaño reducido, de alta velocidad, que se coloca entre la memoria principal(Static RAM) y la CPU.

1.2. Principio de Localidad

> Se basa en la idea de que los programas tienden a usar datos e instrucciones que están físicamente cerca de los datos y las instrucciones usados recientemente.

Presenta dos tipos:

1. **Localidad Espaccial:** El programa normalmente accederá a una ubicación cercana
2. **Localidad Temporal:** El programa normalmente accederá un futuro próximo

Para aprovechar el principio de localidad, el sistema utiliza una interconexión más amplia para acceder a los datos y las instrucciones. Es decir, un acceso a la memoria funcionará eficazmente en bloques de datos e instrucciones en lugar de instrucciones individuales y elementos de datos individuales. Estos bloques se denominan bloques de caché o líneas de caché.

1.3. Niveles de Caché

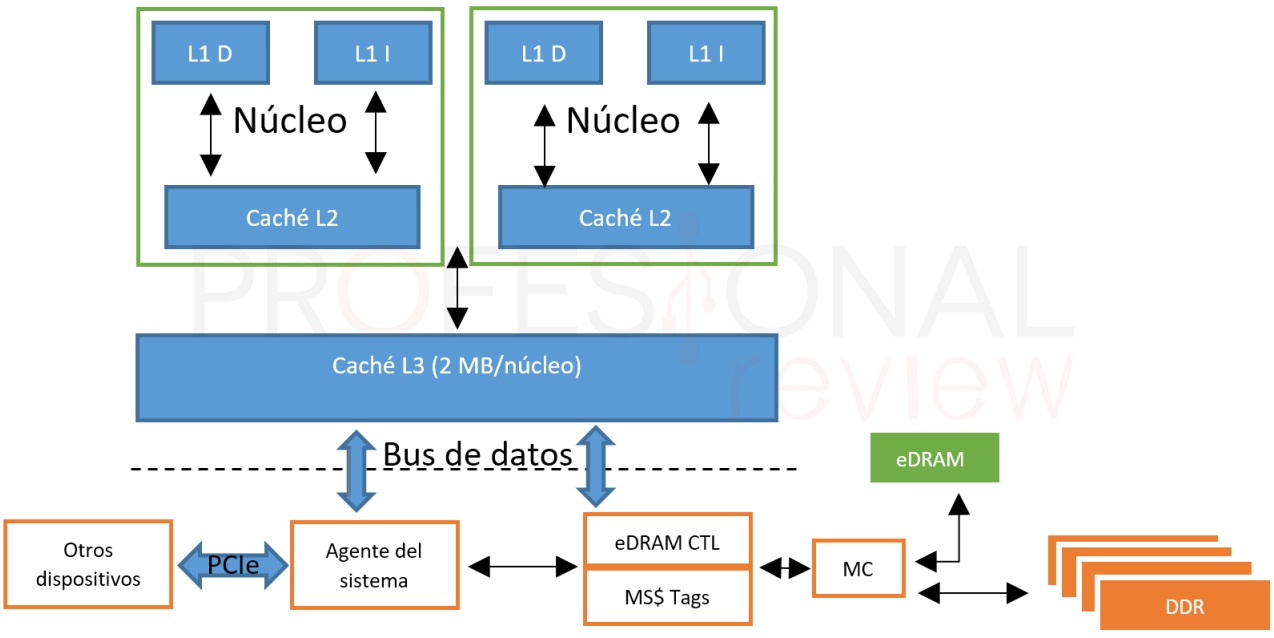

Figura 1: Niveles de la Memoria Caché

1.3.1. Nivel L1

- La caché L1 es la configuración más rápida, la que se encuentra más cerca de los núcleos. Ésta almacena los datos que inmediatamente van a ser usados por la CPU, y es por ello que las velocidades están en torno a los 1150 GB/s y la latencia en tan solo 0,9 ns.
- El tamaño de esta memoria caché está en torno a los 256 KB en total, aunque según la potencia de la CPU (y coste) será menor o mayor, de hecho, los procesadores de Workstation como el Intel Core i9-7980 XE cuentan con unos 1152 KB en total.
- Esta caché L1 se divide en dos tipos, la caché L1 de datos y la caché L1 de instrucciones, la primera se encarga de almacenar los datos que se procesarán y la segunda almacena la información sobre la operación a realizar (suma, resta, multiplicación, etc).
- Además, cada núcleo cuenta con sus propias memorias caché L1, así que, si tenemos un procesador de 6 núcleos, tendremos 6 caché L1 divididas en L1 D y L1 I. En los procesadores Intel cada una de ellas son de 32 KB, y en los procesadores AMD también son de 32 KB o 64 KB en la L1 I. Por supuesto variarán según la calidad y potencia, como siempre.

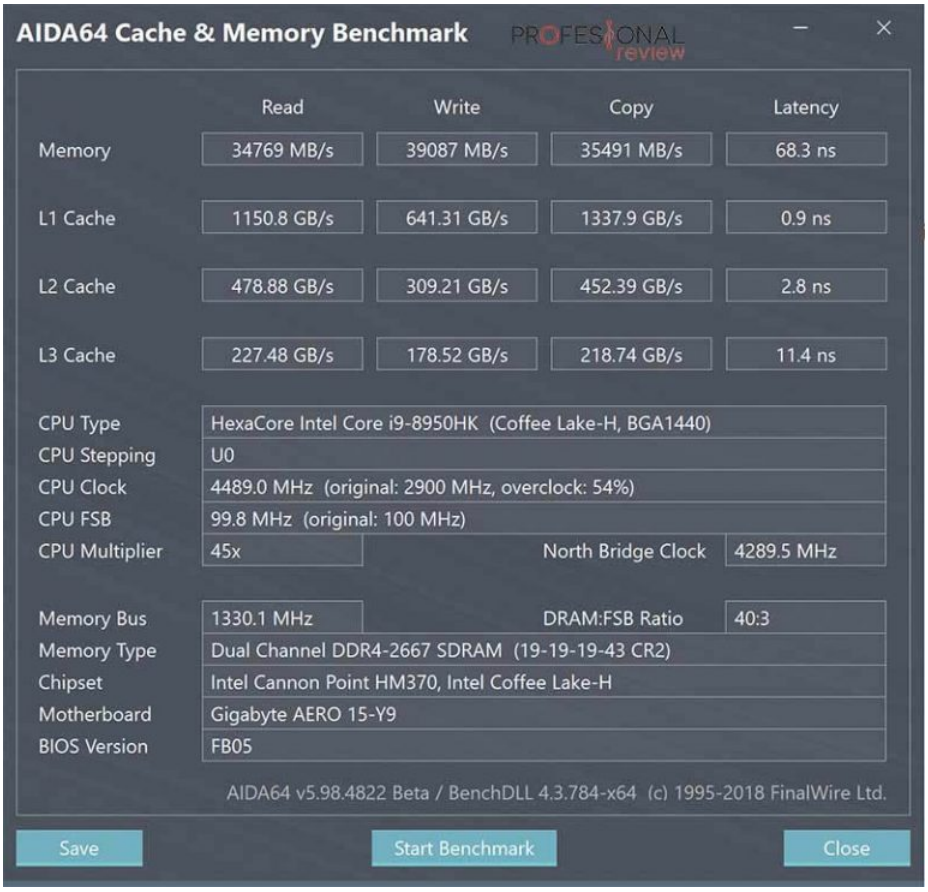

Figura 2: Visualizacion de los Tipos de Memoria Caché

1.3.2. Nivel L2

- Tiene mayor capacidad de almacenamiento, aunque será un poco más lenta, de unos 470 GB/s y 2,8 ns de latencia.
- El tamaño de almacenamiento suele variar entre los 256 KB y los 18 MB. Ya vemos que son capacidades considerables para las velocidades que manejamos.
- En ella se almacenan las instrucciones y datos que pronto serán utilizadas por la CPU y en este caso no está dividida en Instrucciones y datos.
- Tenemos una caché L2 por cada núcleo, generalmente en los procesadores más relevante y por cada núcleo, suele haber 256, 512 o hasta 1024 KB.

1.3.3. Nivel L3

- Esta memoria iene un espacio dedicado para ella en el chip del procesador. Presentan el mayor tamaño y también es la más lenta, hablamos de más de 200 GB/s y 11 ns de latencia.
- En la actualidad un procesador tiene 4 MB de caché L3, y puede verse unidades de hasta 64 MB.
- La L3 se reparte normalmente en unos 2 MB por cada núcleo, pero digamos que no está dentro de cada núcleo, así que hay un bus de datos para comunicarse con ellos. De este bus y del propio de la memoria RAM depende en gran medida la solvencia y velocidad de una CPU, y es donde Intel saca su poderío frente a AMD.

1.4. Tipos de Memoria Caché de acuerdo a la Disponibilidad de Información

1.4.1. Cache Hit

Si la información es viable.

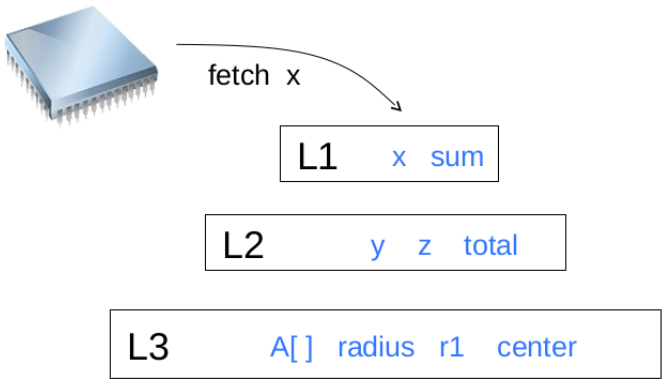

Figura 3: Cache Hit

1.4.2. Cache Miss

Si la informacion no es admisible

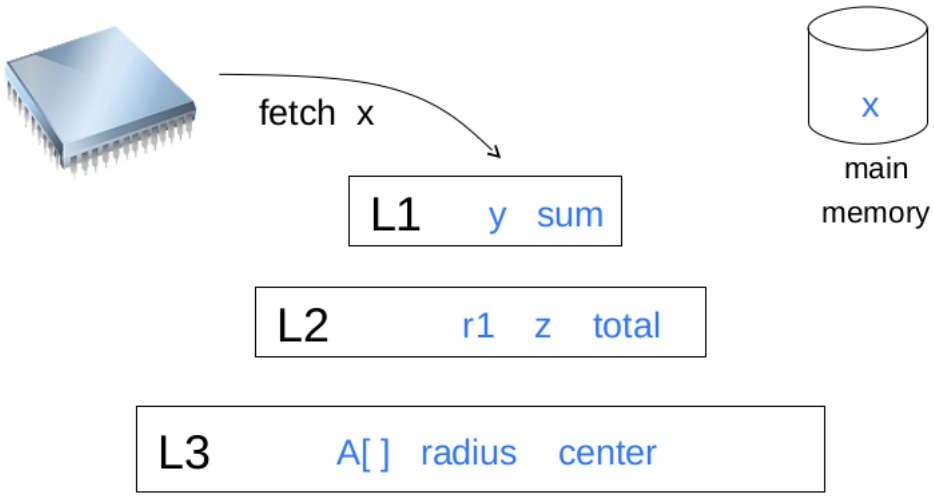

Figura 4: Cache Miss

1.5. Problemas con la caché

> Cuando la CPU escribe datos en un caché, el valor en el caché y el valor en la memoria principal son diferentes o inconsistentes.

Hay dos enfoques básicos para lidiar con la inconsistencia.

1.5.1. Cachés de escritura simultánea

La línea se escribe en la memoria principal cuando se escribe en la caché.

1.5.2. Cachés de escritura diferida

Los datos no se escriben de inmediato.

Los datos actualizados en la caché se marcan como sucios, y cuando la línea de la caché se reemplaza por una nueva línea de la memoria caché, la línea sucia se escribe en la memoria.

1.6. Función de Correspondencia (Mapping)

Este elemento de diseño determina como se asocia una cierta posición de memoria con su posible ubicación en la memoria cache. La memoria cache está formada por un cierto conjunto de líneas. En cada línea se puede almacenar un bloque de memoria.

Por tanto :

> La función de correspondencia establece para cada bloque de memoria cuales son las líneas posibles de ser utilizadas en la cache para almacenarlo.

La relación puede ser desde que cualquier línea de la cache puede recibir a un bloque dado de memoria (denominada correspondencia "completamente asociativa") hasta que solo una línea determinada puede recibir un bloque dado (denominada correspondencia "directa"), pasando por que cierto conjunto de n líneas puede alojar un bloque dado (es la correspondencia "asociativa por conjuntos de n vías").

2. Memoria Virtual

2.1. Definición

La memoria virtual es el mecanismo más general para la ejecución de programas no enteros en memoria. Se basa en un sistema de paginación (o combinado) en el que sólo un subconjunto de las páginas del programa están cargadas en memoriay resto reside en un dispositivo de almacenamiento

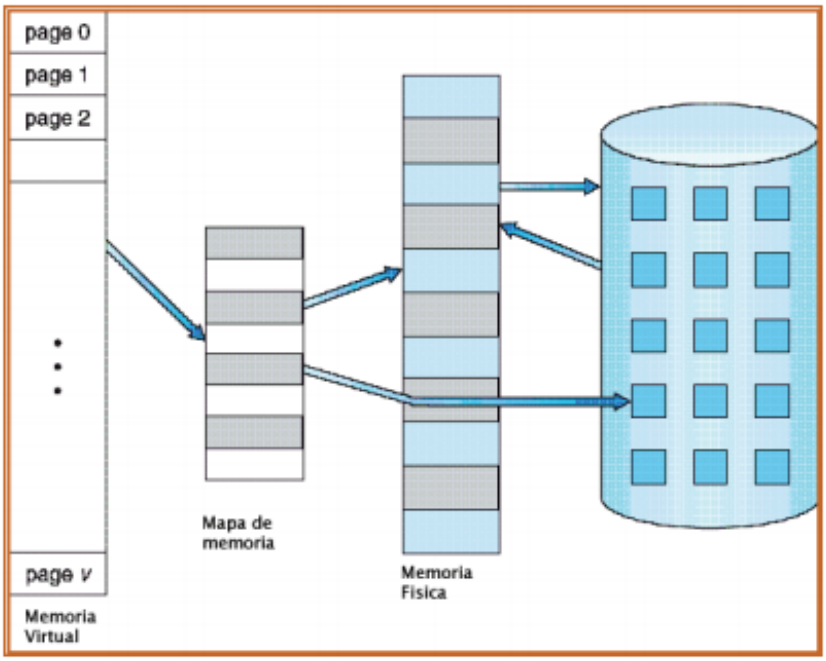

Figura 5: Esquema general de la memoria virtual

2.2. Swapping

- **Problema**
  La memoria virtual es el mecanismo más general para la ejecución de programas no enteros en memoria. Se basa en un sistema de paginación (o combinado) en el que sólo un subconjunto de las páginas del programa están cargadas en memoria. El resto reside en un dispositivo de almacenamiento.

- **Soluciones**

1. Incrementar la memoria principal
   - Es caro
   - Más memoria = programas mas grandes
2. Intercambio (Swapping)

2.2.1. Características

1. Se tiene una cola(queue) a lo largo del proceso de solicitudes en el disco.
2. Los procesos se traen a la memoria principal a medida que hay espacio disponible.
3. A medida de que un proceso se completa se retira de la memoria principal.
4. Puede pasar que ninguno de los procesos de memoria este listo, como por ejemplo : todos esten esperando una operancion de Entrada/Salida.
    - En lugar de estar parado, el procesador intercambia uno de esos procesos, ubicandolo en el disco en una cola intermedia.
    - Trae otro proceso de la cola intermedia o acepta otra petición de proceso de la cola de largo plazo.
    - Pero el intercambio es otro proceso de E/S.

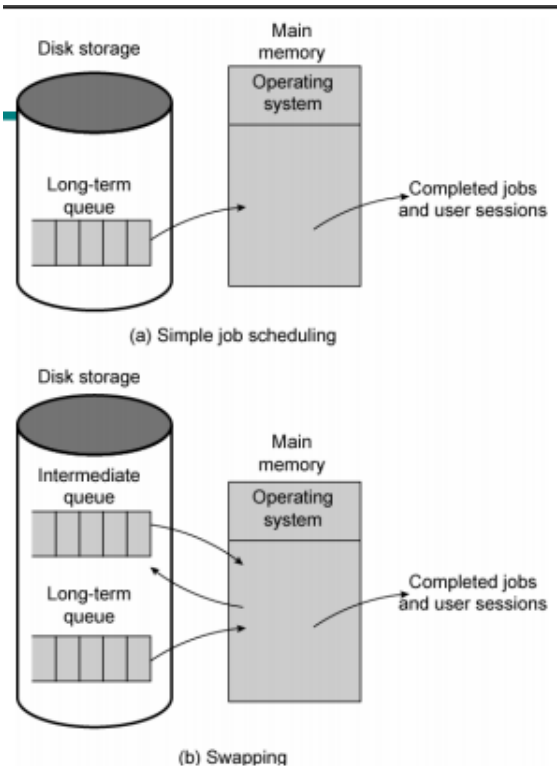

Figura 6: Swapping

2.3. Paginación

1. Se divide la memoria en pequeños trozos iguales y de tamaño fijo - frames (marcos).
2. Se dividen los programas (procesos) en pequeños trozos iguales – pages (páginas).
3. Se asignan el número requerido de frames a un proceso.
4. El Sistema Operativo mantiene una lista de frames libres.
5. Un proceso no requiere que sus páginas se alojen en frames contiguos.
6. Se usa una tabla de páginas que indica en que frame se aloja cada página del proceso.

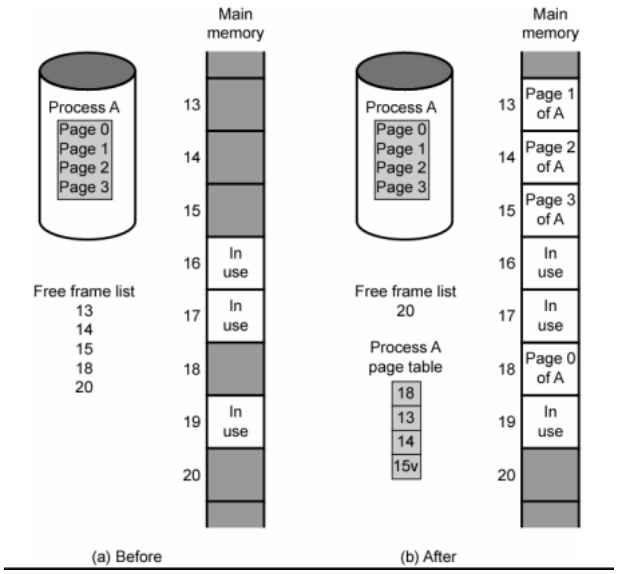

Figura 7: Asignación de Marcos (Frames) libres

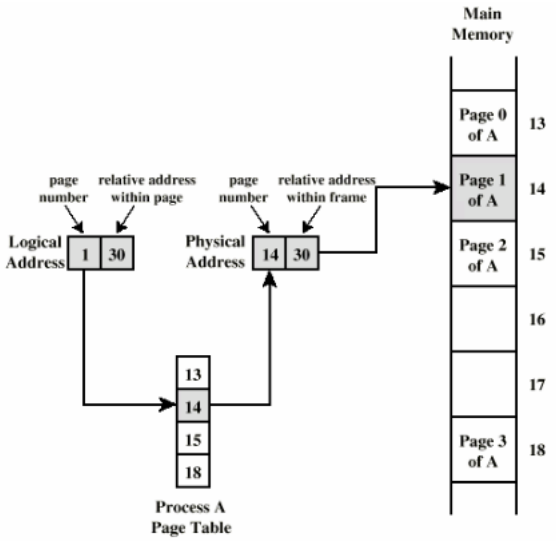

Figura 8: Direcciones lógicas y físicas - Paginación

2.4. Buffer de Traducción Anticipada

(Translation Lookaside Buffer)

1. Toda referencia a memoria virtual puede ocasionar dos accesos a la memoria física:
   - Para tomar el elemento de la tabla de páginas apropiada.
   - Para tomar el dato correspondiente.
2. Esto duplicaría el tiempo de acceso a memoria
3. Para evitar esto, se usa una cache especial para los elementos de la tabla de páginas.
   - TLB

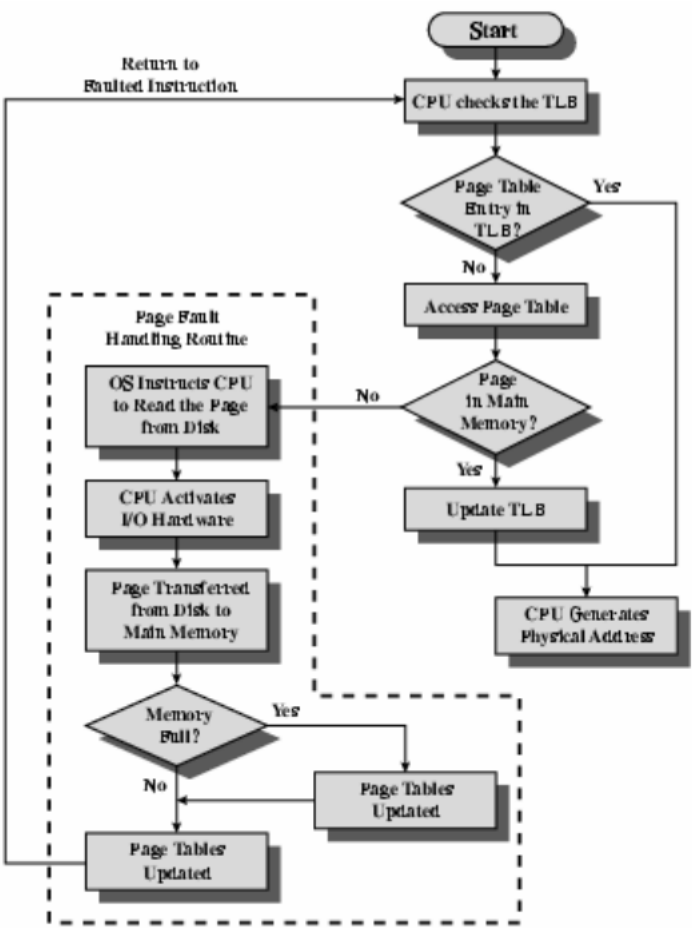

Figura 9: Operación de TLB

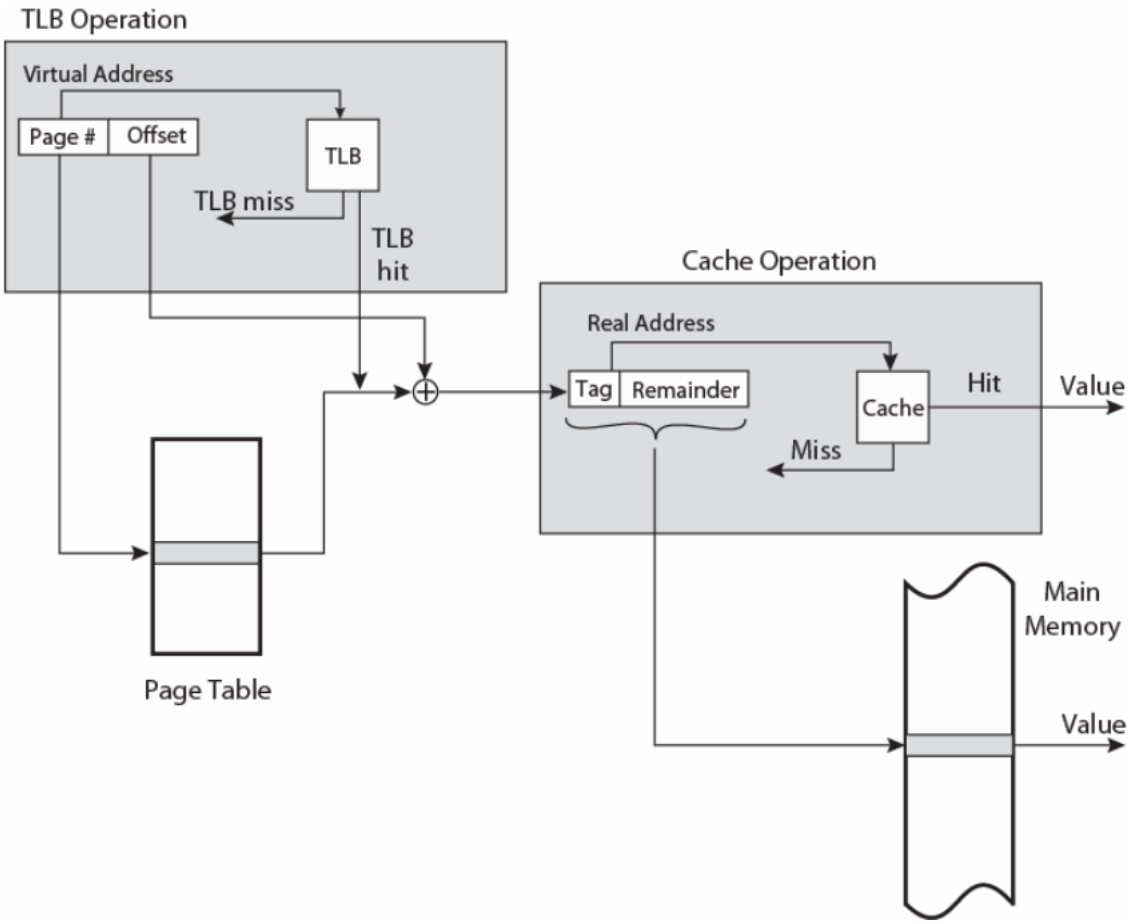

Figura 10: Operación de TLB y Caché

3. Paralelismo a Nivel de Instrucción

3.1. Segmentación

También llamado pipelinin es una técnica de implementación que se basa en dividir la ejecución de cada instrucción en varias etapas. Esta técnica no afecta (o afecta mínimamente) a la latencia de cada instrucción, pero permite incrementar el throughput, puesto que permite nalizar (idealmente) una instrucción por ciclo.

3.2. Técnicas de compilación e ILP

- El paralelismo de instrucción es aplicable dentro de cada bloque básico (secuencia de instrucciones sin saltos). Sin embargo, la longitud media de un bloque básico es de 3 a 6 instrucciones lo que reduce bastante su posible aprovechamiento.
- Una técnica para mejorar el aprovechamiento del ILP dentro de un bucle se conoce como desenrollamiento de bucles.
- El desenrollamiento de bucles consiste en entrelazar la ejecución de instrucciones de varias iteraciones de un bucle. Al tratarse de instrucciones no relacionadas no generan dependencias y permiten un mejor aprovechamiento de la arquitectura segmentada.

3.3. Técnicas de emisión múltiple

Las técnicas de emisión múltiple se basan en la emisión de más de una instrucción por ciclo con el objetivo de alcanzar un CPI menor que uno (lo que equivale a más de una instrucción por ciclo). Este da lugar a tres soluciones: el uso de procesadores superescalares planificados estáticamente, procesadores superescalares planificados dinámicamente y procesadores VLIW.

3.4. Paralelismo a nivel de hilo

- En un procesador multi-hilo los distintos hilos (hardware) de ejecución comparten las unidades funcionales del procesador lo que hace necesario su replicación.
- Se distinguen tres aproximaciones:
  1. Multi-hilo de grano fino.
  2. Multi-hilo de grano grueso.
  3. Multi-hilo simultaneo.

3.4.1. Multi-hilo de grano fino.

Se alterna entre hilos en cada instrucción

3.4.2. Multi-hilo de grano grueso.

Se producen alternancias en las detenciones largas (como los fallos en caché L2).

3.4.3. Multi-hilo simultáneo.

permite la ejecución de instrucciones de varios hilos de forma simultánea.

3.5. Especulación

- Para mejorar los resultados de la predicción de saltos, el siguiente paso es la especulación sobre el resultado de las bifurcaciones y la ejecución asumiendo que la especulación ha sido correcta.
- Esto requiere un mecanismo de tratamiento para los casos en que la especulación es incorrecta.
- Los componentes básicos de la especulación son la predicción dinámica de saltos, la especulación y la planifcación dinámica.
- Para conseguirlo, se separa la finalización de una instrucción del el paso del resultado de una instrucción que produce un resultado a otra que lo usa.

3.6. Pipelining por Software

El pipelining por software es una técnica que explota la habilidad de una máquina para ejecutar varias instrucciones a la vez. Las iteraciones del ciclo son calendarizadas para comenzar en pequeños intervalos, que permita que el ciclo pueda ejecutarse con rapidez, aprovechando al máximo el paralelismo entre iteraciones. Esto resulta de gran importancia, ya que hay relativamente poco paralelismo entre las instrucciones en una sola iteración.

- El desenrollamiento de ciclos mejora el rendimiento, debido a que traslapa el cálculo de las iteraciones, pero aún así el límite del ciclo desenrollado sigue actuando como barrera para el movimiento de código. Es decir la mejora en rendimiento podría ser aún mayor si esto es tratado adecuadamente.
- La técnica de pipelining por software superpone una serie de iteraciones consecutivas continuas hasta que se agotan, produciendo un código compacto y eficiente.
- Tómese como ejemplo el ciclo:

```
for ( i = 0;  i < n; i++)
        D [ i ] = A[ i ] * B[ i ] + c;
```

Figura 11: Código de Ejemplo

- Las iteraciones en el ciclo escriben en distintas localidades de memoria, que por sí solas son diferentes de cualquiera de las ubicaciones leídas, por lo tanto, no hay dependencias de memoria y todas pueden proceder en paralelo.

- En general, se obtiene un mejor uso del hardware al desenrollar varias iteraciones de un ciclo. El problema que esto conlleva es el aumento en tamaño de código, lo que a su vez puede tener un impacto negativo sobre el rendimiento en general. Es por ello que se debe elegir un número finito de veces para desenrollar un ciclo, que pueda obtener la mayoría de la mejora en el rendimiento sin que se expanda el código demasiado.
- El desenrollamiento de un ciclo coloca varias iteraciones del mismo en un bloque básico grande, con lo cual puede utilizarse un algoritmo simple de calendarización por lista para las operaciones a ejecutar en paralelo.

4. Paralelismo a nivel de hilo de ejecución o multithreading

El rendimiento de las aplicaciones no escala proporcionalmente en la cantidad de recursos añadidos. Incluso, en muchos casos, por más recursos que se añadan, el rendimiento de la aplicación se ve incrementado solo ligeramente.

Este factor es el que motivó la aparición de arquitecturas que consideran la ejecución simultánea de diferentes hilos de ejecución en un mismo procesador: paralelismo a nivel de hilo o thread level parallelism (TLP) (Lo, 1997). En estas arquitecturas:

- Diferentes hilos de ejecución comparten las unidades funcionales del procesador (por ejemplo, unidades funcionales).
- El procesador debe tener estructuras independientes para cada uno de los hilos que ejecuta, como registro de renombre o contador de programa, entre otros.
- Si los hilos pertenecen a diferentes procesos, el procesador debe facilitar mecanismos para que puedan trabajar con diferentes tablas de páginas.

La mayoría de aplicaciones actuales tienen un alto nivel de paralelismo y su rendimiento escala añadiendo más paralelismo a nivel de procesador.

El objetivo es mejorar la productividad de los sistemas que pueden correr aplicaciones que son inherentemente paralelas. Ejemplos de este tipo de aplicaciones son, entre otros, navegadores, bases de datos o servidores web.

En estos entornos actuales, sacar rendimiento explotando el TLP puede ser mucho más efectivo en términos de coste/rendimiento que explotando el ILP.

En la mayoría de casos, cuanto más paralelismo se facilite, más rendimiento se podrá sacar. En cualquier caso, tal como se estudia más adelante, la mayoría de técnicas que se habían empleado para sistemas ILP también son empleadas en sistemas TLP.

4.1. Ejecución multihilo simultánea 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.

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.

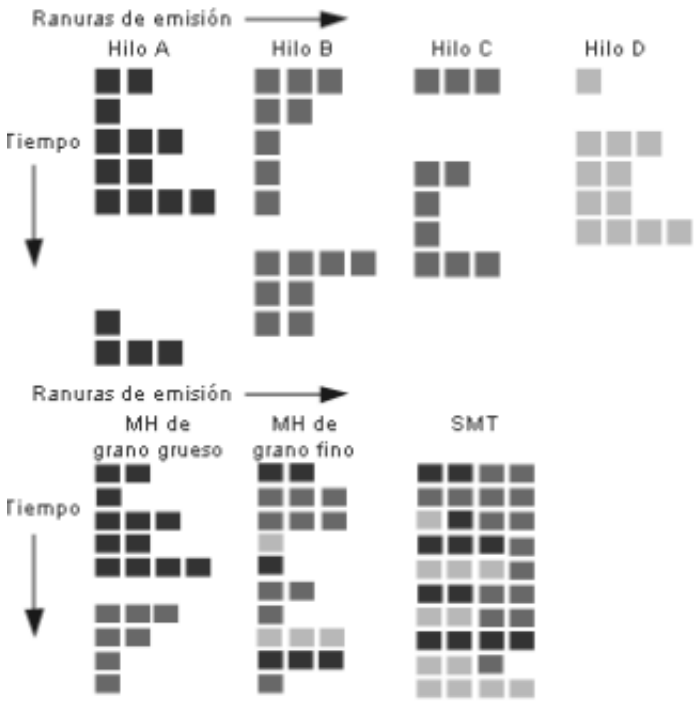

Figura 12: Diferencias en la capacidad del procesador para explotar los recursos superescalares

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.

5. Referencias

1. Professional Review - Tendencias en Memoria Caché.
2. Universidad de las Palmas de Gran Granaria - Sistemas Operativos.
3. Tecnologo en Informatica Fing/CETP - Arquitectura de Computadores.
4. Pontificia Universidad Católica de Chile -Departamento de Ciencia de la Computación - Pipeline: Paralelismo a nivel de instrucción
5. Universidad de Costa Rica - Escuela de Ciencias de la Computación e Informática - Paralelismo a Nivel de Instrucción
6. Universidad Carlos III de Madrid - Departamento de Informática - Paralelismo a nivel de instrucción.
7. Hennessy, John L; Patterson, David A: Computer Architecture: a quantitative approach , Tercera Edición, Morgan Kaufman, San Francisco, USA, 2003.
8. Pontificia Universidad Javeriana - Procesos, hilos y Recursos
9. Universidad de Las Palmas de Gran Canaria - Programación Concurrente