

1. Arquitecturas paralelas: Clasificación y Prestaciones

1.1

Clasificación del paralelismo implícito en una aplicación.

7.1 Libro grande.

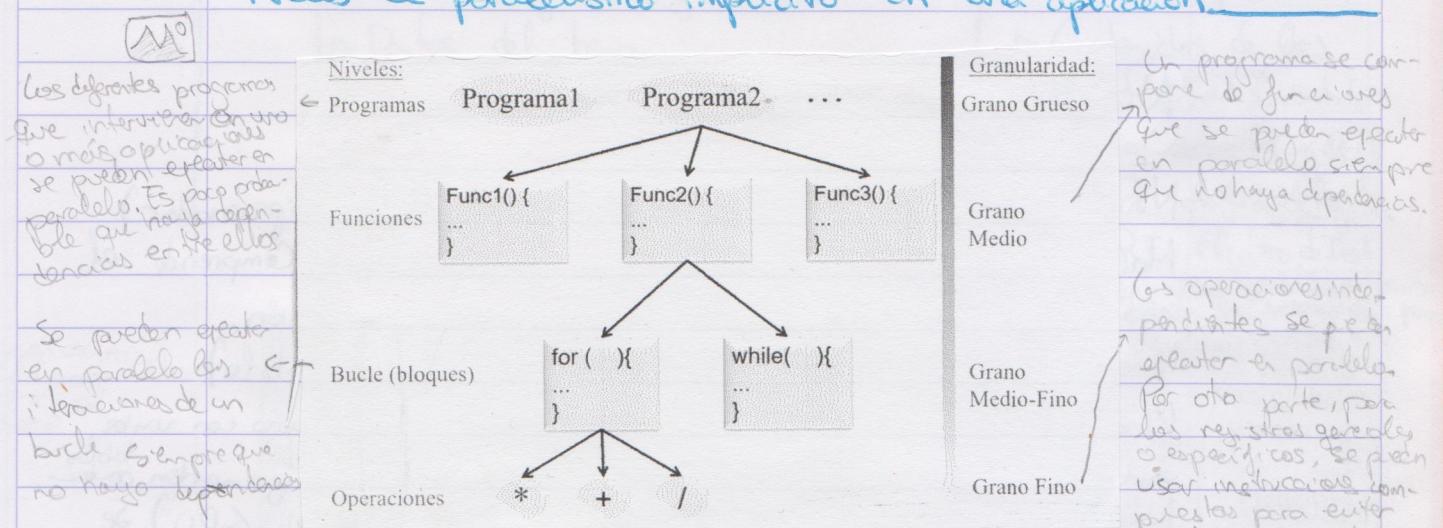
Paralelismo implícito Aquellas cosas que se pueden hacer en paralelo dentro de una tarea. Se puede clasificar según tres cosas:

Nivel de paralelismo, es decir, el nivel de abstracción dentro del código secuencial.

Paralelismo de tareas y datos, o sea, la capacidad de parallelizar una instrucción ^{sol} con varios datos, o dos tareas simultáneas.

Granularidad, es decir, el ámbito donde se da el paralelismo: operaciones / programas...

Niveles de paralelismo implícito en una aplicación.



Dependencias de datos.

Cuando dos operaciones requieren un dato compartido, hay que terminar una para que comience otro, rompiendo el paralelismo.

Condiciones

para que Bz presente dependencia respecto Bz.

Hacen referencia a la misma posición de memoria. Bz aparece en la secuencia de código antes que Bz.

(Dependencia verdadera
Dep. de salida)
Dependencia

Tipos

RAW Read After Write
WAW Write After Write
WAR Write After Read

$$\begin{aligned}a &= b + c \\d &= a + c \\a &= b + c \\a &= b + d \\b &= a + c \\a &= d + c\end{aligned}$$

Paralelismo implícito en una aplicación

los bloques
ser funciones
y las conexiones
entre ellos
reflejan el flujo
de datos entre
operaciones.

Paralelismo de tareas. Paralelismo a nivel de función.
Se encuentra extrayendo la estructura lógica de funciones
de una aplicación.

Extrae
paralelismo
analizando las
operaciones
en la
estructura de
los datos
en los
diferentes
bloques
del bucle.

Paralelismo de datos. Paralelismo a nivel de bucles.
Podría paralelizarse el mismo código del bucle para
los datos que se tratan dentro. Se encuentra implícito
en las operaciones con estructuras de datos (vectores y
matrices). Se puede extraer de la representación matemática
de la aplicación. Ejemplo $V_1 = V_2 + V_3$ engloba sumas de
escalares.

Niveles de detección

(→: Se extrae)

Paralelismo a nivel de programa → Nivel de procesos TLP (Multicomputador, multiprocesador)

Paralelismo entre bucles → Mayor unidad de ejecución que gestiona el SO. Comprende el
código del programa y lo necesario para su ejecución.

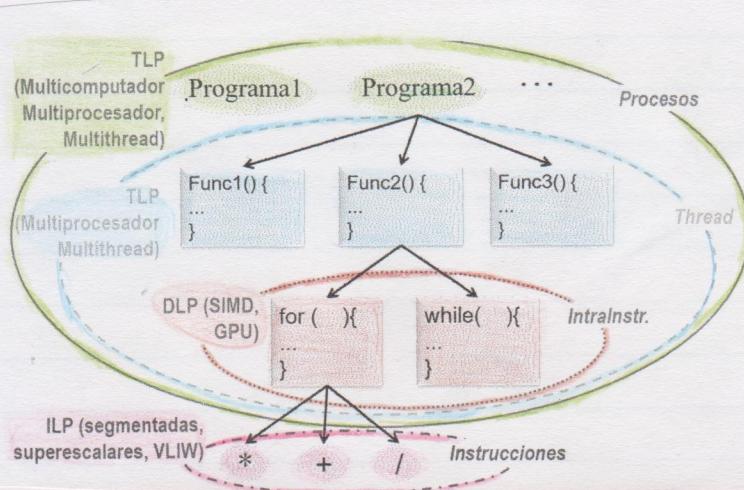
Es más fácil usar
hebras a
procesos. → Nivel de Thread TLP (Multiprocesador, multihilo). Aprovecha
el paralelismo
entre
gránulos de
ejecución.

Paralelismo a nivel de bucle → Una hebra es una estructura similar al proceso pero con varios
estados asociados al proceso, pudiendo ejecutarse en paralelo y compartir datos.

Paralelismo a nivel de instrucciones (bucle) DLP (SIMD, GPU) → Nivel de instrucciones (bucle) DLP (SIMD, GPU) Se
puede explotar dentro de una instrucción vectorial para ser aprovechada
por arquitecturas SIMD o vectoriales.

Paralelismo entre operaciones → Nivel de instrucciones ILP (Procesadores segmentados, superescalares,
VLIW). La unidad de control de un core o procesador
gestiona la ejecución de instrucciones por la unidad de procesamiento.
Puede ejecutar en paralelo las operaciones reburadas a operaciones independientes.

2 x 29





Threads vs. procesos. El paralelismo explícito

El paralelismo explícito se basa en procesos y threads

Proceso

Hace referencia a todo el código del programa y todos los datos y recursos necesarios para su ejecución.

Para comunicarse entre procesos se usan llamadas al SO

Cada proceso tiene:

- ▷ Datos en pila
- ▷ Variables globales y estáticas
- ▷ Datos del heap
- ▷ Contenido de los registros
- ▷ Tabla de páginas
- ▷ Tabla de ficheros abiertos

Thread / Proceso ligero

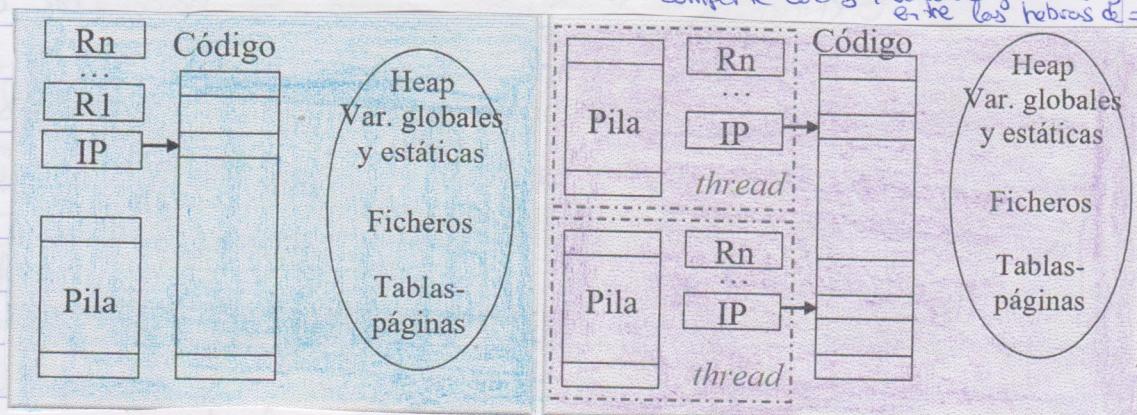
Cada uno de los flujos de control que conforman el proceso.

Para comunicarse entre threads se usa la memoria que comparten entre todos ~~los~~ ellos.

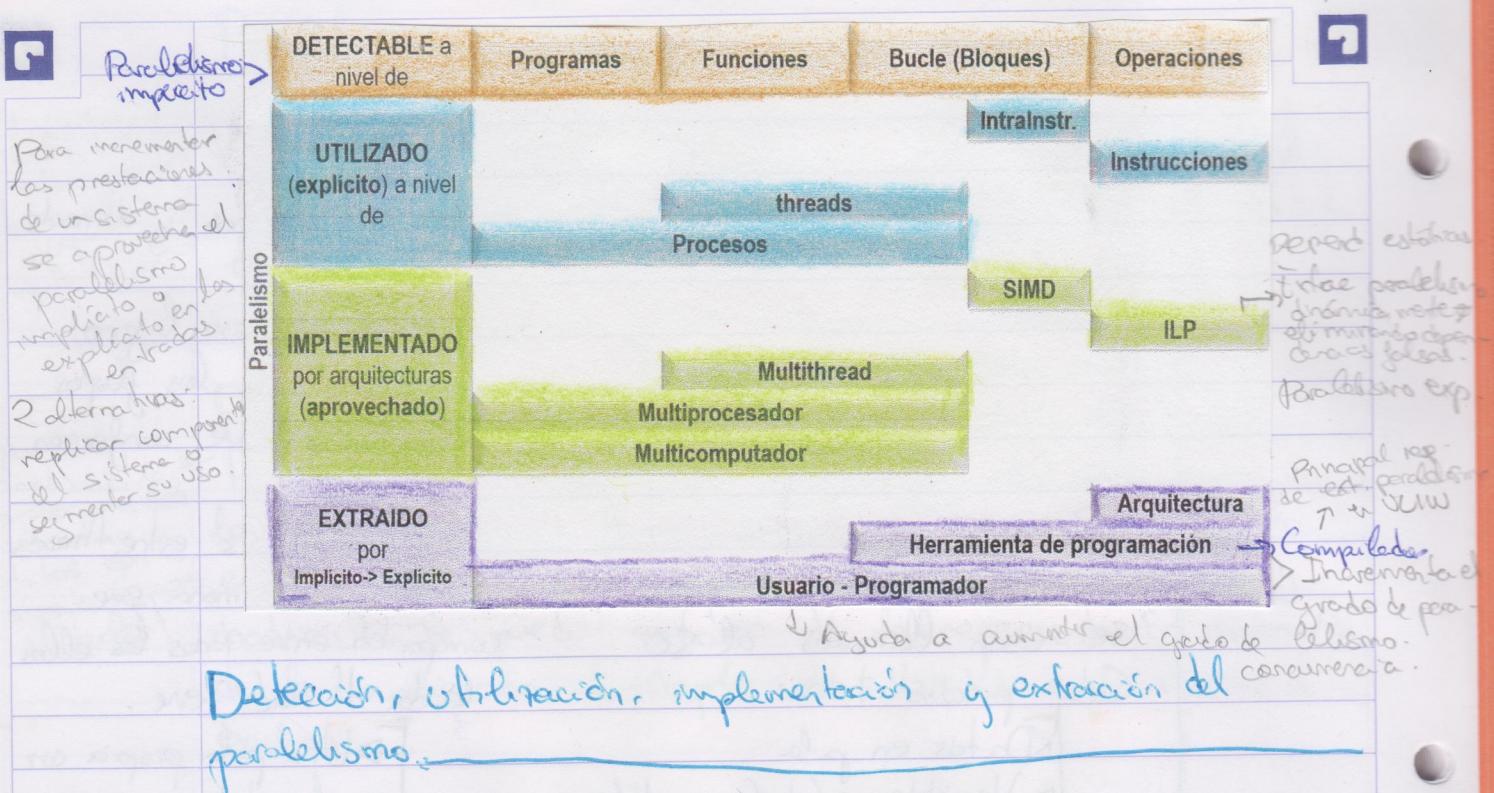
Cada thread tiene:

- ▷ Una pila propia con los datos que use.
- ▷ Contenidos de los registros ^{propios} en su ámbito (sobre todo, PC y RI)
- * PC: Contador de Programa
- * RI: Registro de Inst.

Comparte código, variables globales y recursos entre las hebras de = proceso



Con las características de las hebras se permite crear y destruir hebras, y comutar entre ellas en un tiempo menor a los procesos. La comunicación entre hebras también sería de menor tiempo. Esto permite que las hebras tengan una granularidad menor que la de los procesos.



- **Detección**: Clasifica la granularidad y el paralelismo implícito.

Utilización Maravilla de explicar el paralelismo

Implementación Aprovechamiento de las arquitecturas para usar el paralelismo en el hardware

Extracción: Se refiere a quién hace la extracción del paralelismo a partir de un código secuencial.

Grado de paralelismo: No tienen nº de entradas del control que se pueden parallelizar.

1.2

Computación paralela Estudia los aspectos hardware y software relacionados con el desarrollo y ejecución de aplicaciones en un sistema de cómputo compuesto de varios cores, procedimientos o computadores vistos externamente por como una unidad autónoma. Por ejemplo, si se manejan a través de un SO común.

Computación distribuida Estudia los aspectos en un sistema distribuido, es decir, en una colección de sistemas autónomos situados en distintas localizaciones físicas. En ese caso, la granularidad debería ser gruesa para extraer el paralelismo.

Criterios de clasificación

Comercial : Segmentación del mercado

Educación e investigación: Flujos de control y datos.

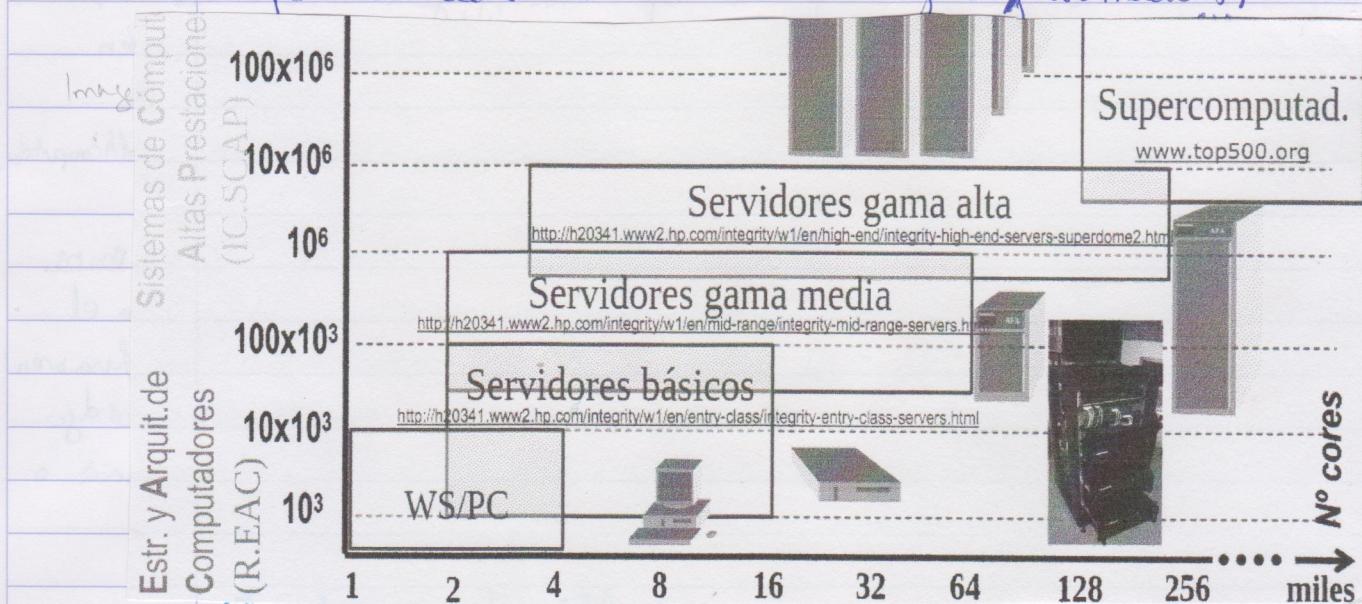
Sistemas de memoria, nivel de paralelismo e paralelo.

Clasificación comercial

Computadores empotrados Dispositivos que tienen un procesador para controlar y obtener información, como móviles, relojes inteligentes. Son económicos y tienen un propósito específico. Hace falta que haya un menor consumo de potencia y generación de calor.

Por ejemplo, la PlayStation tenía un procesador SIMD que usaba, sobre todo, para gráficos, como vectores y matrices.

Computadores de uso general: Por ejemplo; Ordenadores de sobremesa y portátiles, servidores clusters... Se usan para todo tipo de aplicaciones (oficina, entretenimiento, procesamiento de transacciones, soporte de decisiones, e-commerce, científicos y animaciones)



Clasificación de Flynn

Divide en una tabla el paralelismo de flujos de datos y de instrucciones, siendo una tabla con dos filas y dos columnas. En un computador se pueden ejecutar uno o varios flujos de datos y uno o varios flujos de instrucciones.

Flujos de instrucciones

	Single (cho)	Multiple
Flujos de datos	SISD	MISD
Multiple	SIMD	MIMD

Single Instruction Single Data (SISD): Un único

flujo de instrucciones procesa operandos y genera resultados, definiendo un único flujo de datos. Correspondiente a los computadores con un núcleo solo en su CPU.

Single Instruction Multiple Data (SIMD) El único flujo de

instrucciones procesa operandos y genera resultados pero definiendo varios flujos de datos, de forma que cada instrucción codifica varias instrucciones operaciones iguales sobre datos distintos. Esto se da en GPUs y en repetidores multiparalela de Intel y AMD. Entre las arquitecturas destacan los procesadores matriciales y vectoriales.

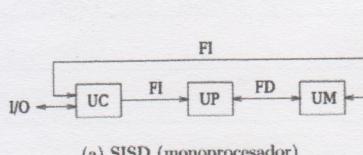
(MIMO)

Multiple Instruction Multiple Data (MIMD) El computador ejecuta

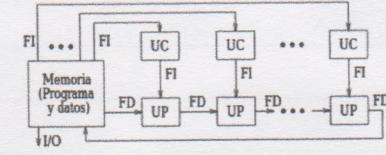
varias secuencias o flujos distintos de instrucciones y cada uno de ellas procesa operandos y genera resultados definiendo un único flujo de instrucciones, de forma que existen también varios flujos de datos uno por cada flujo de instrucciones. Correspondiente con multinúcleos, multiprocesadores y multicamputadores.

Multiple Instruction Single Data (MISD) Se ejecutan varios

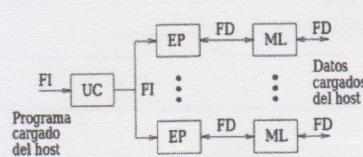
flujos distintos de instrucciones, aunque todos actúan sobre el mismo flujo de datos. No existen computadores que funcionen según este modelo pero se puede simular en un código este modelo para aplicaciones que procesan una secuencia o flujo de datos.



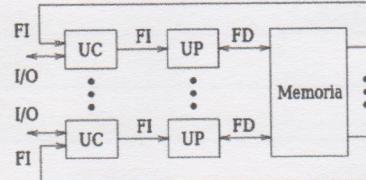
(a) SISD (monoprocesador)



(b) MISD (array sistólico)



(c) SIMD (procesador matricial)



(d) MIMD (multiprocesador)

Figura 4.1: Clasificación de Flynn de las arquitecturas de computadores. (UC=Unidad de Control, UP=Unidad de Procesamiento, UM=Unidad de Memoria, EP=Elemento de Proceso, ML=Memoria Local, FI=Flujo de Instrucciones, FD=Flujo de datos.)

Clasificación de Computadores Paralelos MIND según sistema de memoria

Multiprocesador Todos los procesadores comparten el mismo espacio de direcciones. Esto genera un cuello de botella con respecto a la memoria ya que es posible que dos procesadores quieran acceder a la misma memoria, pero así el programador no necesita saber dónde están los datos. La red de interconexión necesita un buen ancho de datos.

Se pueden llamar SMP (Symmetric Multiprocessor), y tiene una mayor latencia (al acceder a memoria hay que pasar siempre por la red de interconexión) y una menor escalabilidad (nos costaría más trabajo hacer más grande y potente este sistema) y la red debería tener mayor ancho de banda para soportar esa escalabilidad (lo que es complejo y costoso). La comunicación es implícita mediante variables compartidas.

Requiere primitivas de sincronización y la distribución código-datos no es necesaria. Añadiendo la programación, generalmente, es más sencilla.

Multi-computador Cada procesador compone un nodo con su propia espacio de direcciones. Tiene una menor latencia y mayor escalabilidad. La comunicación debe realizarse mediante un paso de mensajes, ya que no hay memoria compartida por los nodos. La sincronización dependería, entonces, de un software para gestionar ese paso de mensajes, y la distribución código-datos se requiere.⁽³⁾ La programación, en consecuencia, puede ser más difícil.

⁽³⁾ Eso implica el uso de herramientas de programación más sofisticadas.

! ¿Cómo puedo mejorar la escalabilidad de sistemas multiprocesador?

Para ello el objetivo principal es no saturar la red de interconexión. Para ello:

Aumentar caché del procesador ~~Con ello solo habría~~ que cargar una vez los datos por la red y de ahí permanecer en el caché.

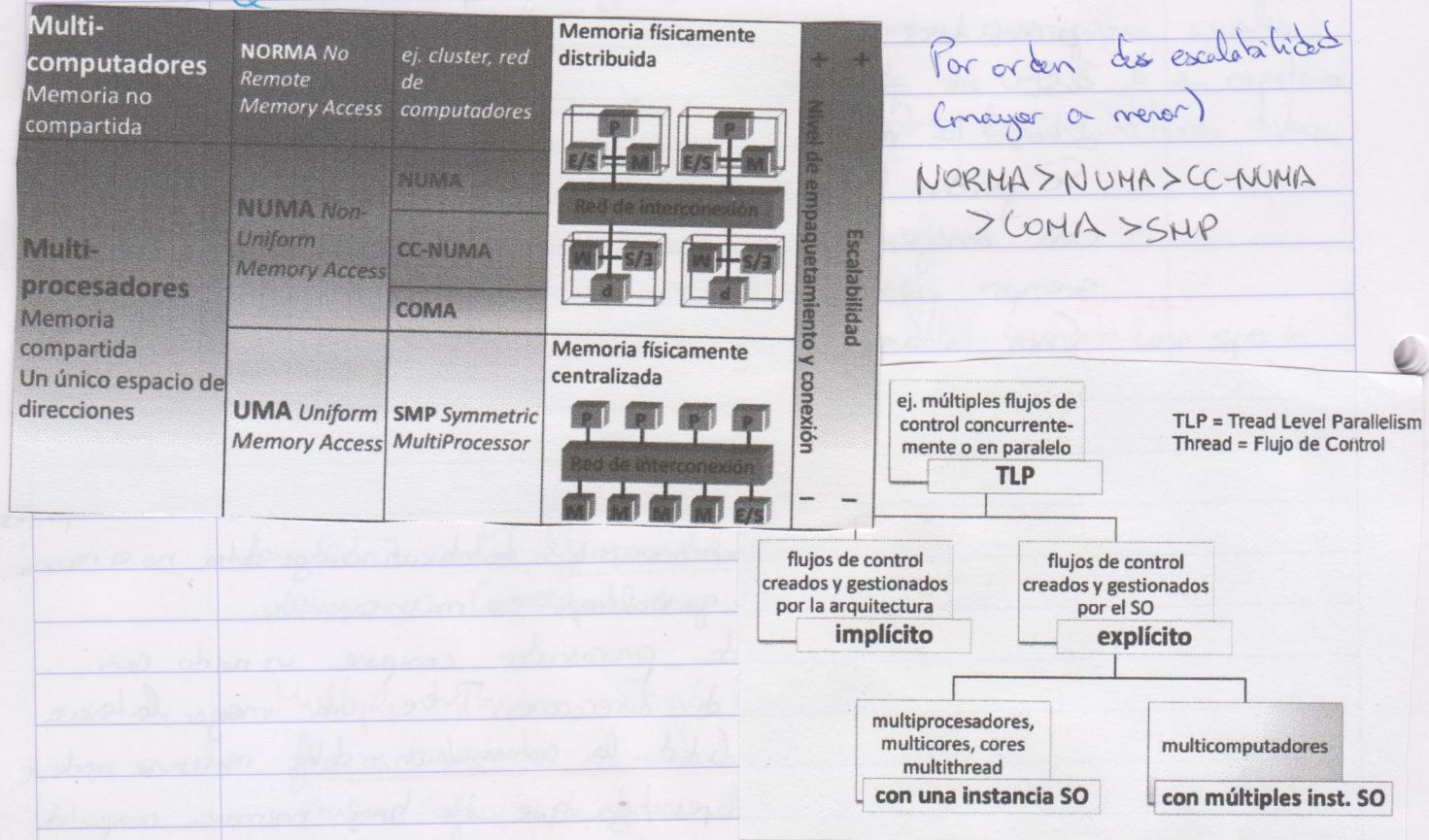
Usar redes de menor latencia y mayor ancho de banda.

Por ejemplo: buses (conecta todos los proc. con la memoria), barras cruzadas (red "completamente N en cada y M salidas, permite) o multi etapa (comunicación que pasa hasta $\min(N,M)$ generaciones por varias etapas para llegar al destino).

Distribuir regiones de memoria entre los procesadores

Si bien la memoria sigue siendo compartida, se evitan conflictos de bus al intentar hacer que dos procesadores vayan a por ella misma zona de memoria.

Clasificación completa de computadoras según el sistema de memoria



Clasificación de arquitecturas con múltiples flujos de control o threads

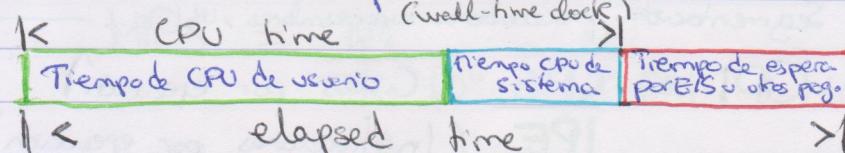
Arq. con DLP (Data Level Parallelism)	Arq. con ILP (Instruction Level Parallelism)	Arq. con TLP (Thread Level Parallelism) explícito y una instancia de SO	Arq. con TLP explícito y múltiples instancias SO
<p>Tema 3</p> <p>Ejecutan las operaciones de una instrucción concurr. o en paralelo</p> <p>Unidades funcionales vectoriales o SIMD</p>	<p>Tema 4</p> <p>Ejecutan múltiples instrucciones concurr. o en paralelo</p> <p>Cores escalares segmentados, superescalares o VLIW/EPIC</p>	<p>Temas 3, 5</p> <p>Ejecutan múltiples flujos de control concurr. o en paralelo</p> <p>Cores que modifican la arquit. escalar segmentada, superescalar o VLIW/EPIC para ejecutar threads concurr. o en paralelo</p>	<p>IC-SCAP</p> <p>Ejec. múltiples flujos de control en paralelo</p> <p>Multi-procesadores: ejecutan threads en paralelo en un computador con múltiples cores (incluye multicores)</p> <p>Multi-computadores: ejecutan threads en paralelo en un sistema con múltiples computadores</p>

1.3 Evaluación de prestaciones de una arquitectura.

TJ: libro pequeño.

Tiempo de respuesta

Se mide en tiempo real, dividiéndose en tres partes.



$$\text{Tiempo de CPU} = \text{Tiempo usuario} + \text{Tiempo sistema}$$

Si hay un flujo de control, $\text{elapsed} \geq \text{Tiempo CPU}$

Si hay venios, $\text{elapsed} < \text{Tiempo CPU}$ pero $\text{elapsed} \geq \text{Tiempo CPU}$

Tiempo de CPU

$$T_{CPU} = \frac{\text{Ciclos del programa}}{\text{Frecuencia CPU (f)}} \cdot T_{ciclo} = \frac{\text{Ciclos del programa}}{\text{Número de Instrucciones (NI)}}$$

$$\text{Ciclos por instrucción (CPI)} = \frac{\text{Ciclos del programa}}{\text{Número de Instrucciones (NI)}}$$

Con ambas fórmulas podemos extraer $T_{CPU} = NI \cdot CPI \cdot T_{ciclo}$
y:

$$T_{ciclo} = \frac{1}{f}, \text{ ya que la frecuencia va por ciclos segundo.}$$

$$CPI = \frac{1}{IPC} \quad (\text{Instrucciones por ciclo})$$

Haciendo esto podemos encontrar maneras de calcular T_{CPU}

$$T_{CPU} = NI \cdot \frac{CPI}{f}$$

$$T_{CPU} = CPI \cdot \frac{NI}{f}$$

Con la idea de bajar el tiempo de CPU surgieron 2 tipos de repertorios

RISC (Complex Instruction Set) Con las mejores tecnologías se hacen instrucciones más complejas de forma que habrá menos instrucciones, pero sobres los ciclos por instrucción, tratando de compensarse con la frecuencia y las mejores.

RISC (Reduced Instruction Set) Su objetivo es aprovechar mejor las recursos con un menor repertorio de instrucciones sencillas, de forma que los ciclos por instrucción bajen, pero al ser sencillas, se necesitan muchas instrucciones para hacer algo, que se dron de compensar con las frecuencias.

CPI NI

Segmentación, procesadores superscalares, VLIWs...

$$\text{CPI} = \frac{\text{CPE}}{\text{IPE}}$$

(Ciclos por ejecución)
(Instrucciones por ejecución)

Ciclos por ejecución Es el número de ciclos entre emisiones. Se calculaiendo cuántos ciclos hay entre una ejecución y siguiente.

Instrucciones por ejecución Es el número de instrucciones que se ejecutan cada vez que se emite. Se calcula dividiendo cuantas instrucciones se ejecutan en el mismo ciclo.

Complejidad del repertorio

$$\text{NI} = \frac{\text{Nº de operaciones (que realiza el programa)}}{\text{Operaciones por instrucción (Nº de operaciones que puede codificar cada instrucción)}}$$

¿En qué influye (...) al tiempo de CPU?

$$T_{CPU} = NI \cdot CPI \cdot T_{ciclo}$$

- Tecnología (Transistores, diseño físico)
- Estructura y organización (Diseño lógico)
- Repertorio de instrucciones (Número de instrucciones y complejidad de estos)
- Compilador (Optimizaciones y conversión a ensamblador)

Instrucciones por segundo.

MIPS (Millones de Instrucciones por Segundo)

$$\begin{aligned} \text{MIPS} &= \frac{NI}{T_{CPU} \cdot 10^6} = \frac{NI}{NI \cdot CPI \cdot T_{CPU} \cdot 10^6} = \frac{1}{CPI \cdot T_{CPU} \cdot 10^6} \\ &= \frac{f}{CPI \cdot 10^6} = \frac{f \cdot IPC}{10^6} \end{aligned}$$

La cantidad puede depender del repertorio de instrucciones que se use. El programa tendrá un número diferente según la arquitectura (en general). Además, cada programa puede tener distinto número de MIPS aunque se ejecute en un ordenador con la misma arquitectura, lo que no sirve para caracterizar la máquina.

Puede variar inversamente con las prestaciones: haciendo que muchos MIPS representen peores prestaciones.

▷ Si $NI_1 = \frac{1}{2} NI_2$ y $T_{CPU_2} = 1.5 \cdot T_{CPU_1}$, ¿Qué máquina es mejor?

$$\text{MIPS}_2 = \frac{NI_2}{T_{CPU_2} \cdot 10^6} = \frac{2NI_1}{1.5 T_{CPU_1} \cdot 10^6} = 1.33 \cdot \frac{NI_1}{T_{CPU_1} \cdot 10^6}$$

$$= 1.33 \text{ MIPS}_1.$$

La mejor máquina deberá ser la 2.

MFLOPS (Millones de operaciones de coma flotante por Segundo)

$$\text{MFLOPS} = \frac{\text{Operaciones en coma flotante}}{10^6}$$

No es una medida constante para todos los programas, además de que no es constante en máquinas diferentes y la potencia de las operaciones en coma flotante no es igual para todos los operadores. Es necesario, pues, una normalización de las instrucciones en coma flotante.

A la hora de experimentar, las instrucciones podrán tener entadas y salidas, por lo que deberán entrarse a la hora de hacer un programa de test. Además, el tiempo de CPU es elapsed, por lo que habrá que quitar cierto tiempo por tareas de gestión del SO.

Programas benchmark

Son programas que evalúan las prestaciones, las propiedades exigidas son la:

Fidabilidad: Deben ser reproducibles.

Permitir comparar diferentes realizaciones de un sistema, o varios sistemas.

Tipos:

(depreciado) Bajo nivel / Micro benchmark Prueban cosas a nivel casi hardware
- Kernels Usan instrucciones que suelen ser privilegiadas.

Sintéticos

Programas reales Programas cotidianos a usar.

Aplicaciones diseñadas Por ej. predicción de tiempo

los benchmarks evalúan componentes o conjuntos de estos; o bien emular situaciones reales con carga (para el computador completo) para poder establecer las comparaciones.

En muchos casos es importante estimar el tiempo de CPU sin considerar E/S que podría requerir un programa a partir de las características del computador que se use, las operaciones y los accesos a memoria. Trata de aproximarse al valor mínimo de tiempo de CPU con el objetivo de ver como rechaza este programa y si su ejecución se ajustará a las restricciones de tiempo previstas.

Ganancia de prestaciones

$$Sp = \frac{V_p}{V_i} = \frac{T_i}{T_p}$$

Velocidad Tiempo

$p =$ Máquina mejorada
 $i =$ Máquina base

$$\text{Velocidad (V)} = \frac{W \text{ (constante)}}{\text{Tiempo}}$$

Ley de Homología

La mejora de velocidad que se puede obtener al mejorar un recurso de una máquina en un factor p está limitada por:

$$S \leq \frac{p}{1 + f(p-1)}$$

Siendo p el factor de mejora y f la fracción del tiempo donde no puedo emplear el recurso que mejora las prestaciones.

Al final, el tiempo para una arquitectura con recursos mejorados es:

$$T_p = f \cdot T_p + \frac{(1-f)T_p}{p}$$

[] []

Tiempo sin posibilidad de mejora

Tiempo con posibilidad de mejora

$$Sp = \frac{T_1}{T_p} \leq \frac{T_1}{T_1 f + \frac{T_1(1-f)}{p}} = \frac{T_1}{T_1 \left(f + \frac{1-f}{p} \right)}$$