SISTEMAS PARALELOS

Clase 2 – Plataformas para programación paralela

UNIVERSIDAD
NACIONAL
DE LA PLATA

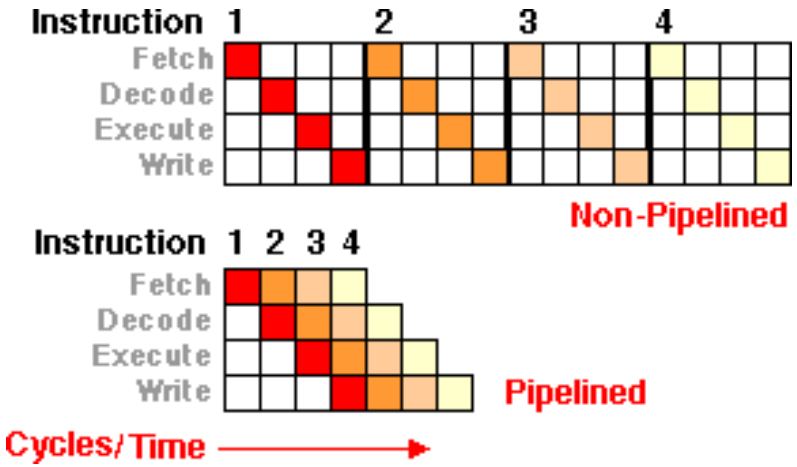
EVOLUCIÓN DE LOS PROCESADORES: PARALELISMO IMPLÍCITO

Evolución de los procesadores: paralelismo implícito

- Vista lógica tradicional de una computadora secuencial: una ***memoria*** conectada a un ***procesador*** a través de un ***bus de comunicaciones***.
- Estos tres componentes presentan cuellos de botellas que afectan al rendimiento global del sistema.
- A lo largo de los años, se han diseñado diferentes innovaciones en la arquitectura de los procesadores que permiten atacarlos
- Una de ellas es la ***multiplicidad***: múltiples procesadores, múltiples memorias, múltiples buses de conexión.
  - ¿El programador siempre los ve?

Pipelining

- El *pipelining* consiste básicamente en solapar las diferentes etapas de la ejecución de instrucciones, reduciendo el tiempo de ejecución total



16 ciclos de reloj

7 ciclos de reloj

Pipelining

- El modelo de referencia es el de “línea de montaje”
  - Supongamos que el armado de un auto requiere 100 unidades de tiempo y que puede ser dividido en 10 etapas en forma de pipeline de 10 unidades cada una → se armaría 1 auto cada 10 unidades de tiempo
- Para aumentar la velocidad del pipeline, podemos dividirlo en etapas más pequeñas incrementado su profundidad.
  - En el contexto de los procesadores, esto permite a su vez mayores frecuencias de reloj.
  - Tener en cuenta que la velocidad de un pipeline está limitado por la duración de su etapa más costosa → Etapas más pequeñas aceleran el pipeline.
- Pero, ¿podemos dividir infinitamente? ¿Es útil?
  - Estadísticamente cada 5/6 instrucciones hay un salto condicional!
  - Ejecución especulativa puede ayudar

Pipelining

- Pipelines profundos requieren de técnicas efectivas que sean capaces de predecir los saltos en forma especulativa
  - La penalización de un salto mal predicho se incrementa a medida que el pipeline es más profundo → más instrucciones adelantadas son las que se pierden
  - Estos factores limitan la profundidad de pipeline y su posible ganancia de rendimiento
- ¿Cómo mejorar entonces la tasa de ejecución de instrucciones?
- Una forma obvia consiste en usar múltiples pipelines
  - Durante cada ciclo de reloj, múltiples instrucciones son emitidas en paralelo, las cuales son ejecutadas en múltiples unidades funcionales

Pipelining y ejecución superescalar

```
1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

(i)

```
1. load R1, @1000
2. add R1, @1004
3. add R1, @1008
4. add R1, @100C
5. store R1, @2000
```

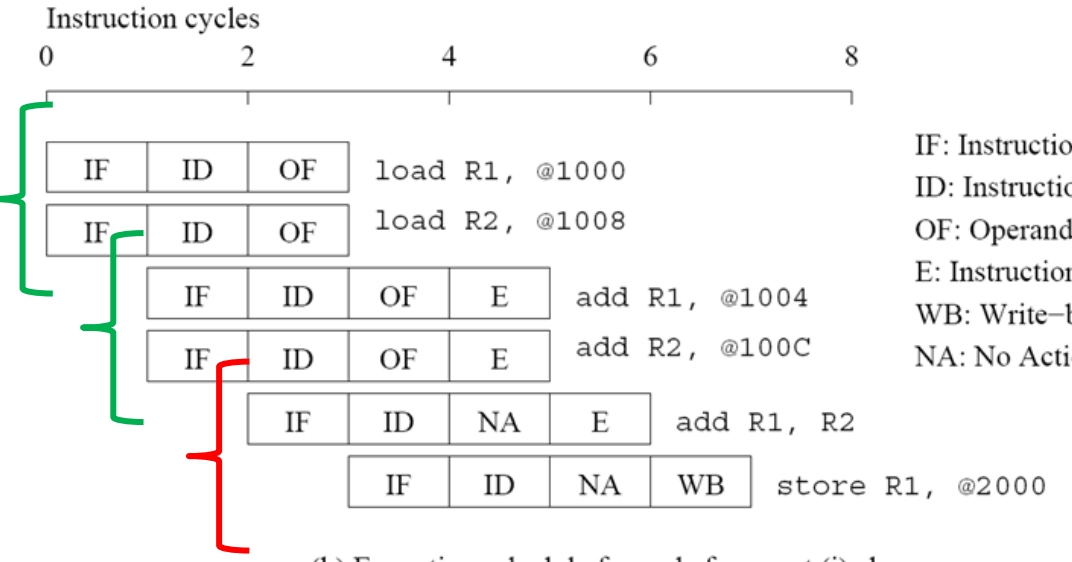
(ii)

```
1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

(iii)

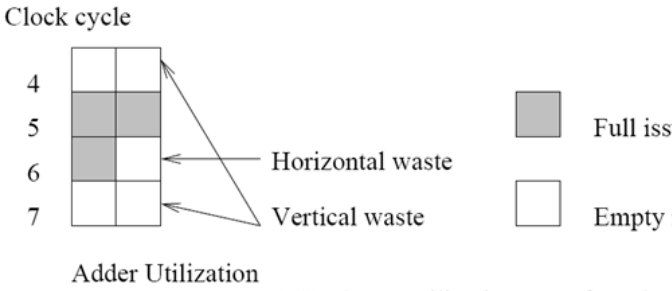
(a) Three different code fragments for adding a list of four numbers.

Procesador con 2 pipelines y la habilidad de emitir simultáneamente 2 instrucciones (*superescalar*)



(b) Execution schedule for code fragment (i) above.

IF: Instruction Fetch
ID: Instruction Decode
OF: Operand Fetch
E: Instruction Execute
WB: Write–back
NA: No Action



(c) Hardware utilization trace for sched

Pipelining y ejecución superescalar

- No hay una única manera de escribir un programa y la misma tiene incidencia en el rendimiento final
- El ideal sería que todas las etapas estén activas en todo momento (máximo paralelismo).
  - En la práctica, es muy difícil que ocurra
- Al momento de realizar la planificación de instrucciones, se deben tener en cuenta:
  - Dependencia verdadera de datos (*True data dependency*): el resultado de una instrucción es la entrada para la siguiente
  - Dependencia de recurso (*Resource dependency*): dos operaciones requieren el mismo recurso (por ejemplo, unidad de punto flotante)
  - Dependencia de salto (*Branch dependency*): las instrucciones a ejecutar después de un salto condicional no pueden ser determinadas a priori sin tener margen de error.

Pipelining y ejecución superescalar

- El planificador (hardware) analiza un conjunto de instrucciones de la cola de instrucciones a ejecutar y emite aquellas que pueden ser ejecutadas en forma concurrente, teniendo en cuenta las dependencias.
  - Si las instrucciones son ejecutadas en el orden en que aparecen en la cola, se dice que la emisión es *en orden* → Simple pero limita significativamente la emisión de instrucciones
  - Si el procesador tiene la habilidad de reordenar las instrucciones en la cola, entonces se puede alcanzar el máximo rendimiento posible → este modelo se conoce como *fuera de orden* (o de *emisión dinámica*) y, aunque es más complejo, es el que se usa en la actualidad

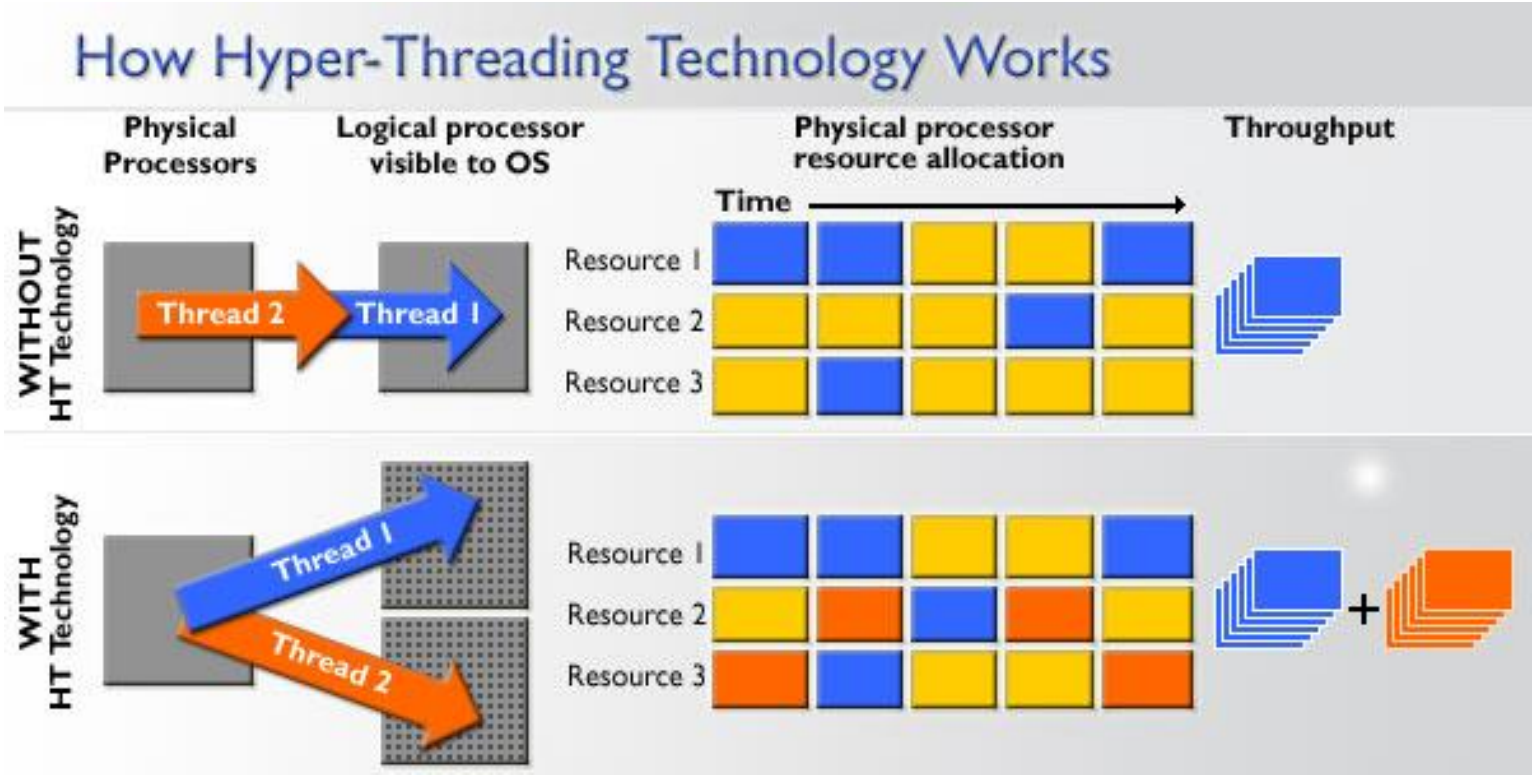
- En resumen, el rendimiento de un procesador superescalar está limitado por la cantidad disponible de paralelismo a nivel de instrucciones.

Multi-hilado a nivel hardware

- En general, un único hilo de ejecución no resulta suficiente para aprovechar la potencia de los procesadores superescalares
- La técnica *Simultaneous Multi-Threading* (SMT) consiste en mantener más de un hilo de ejecución al mismo tiempo en el procesador[1]
  - Los recursos asociados al estado del procesador son replicados una o más veces (contador de programa, registros, punteros, pila, etc) manteniendo el número original de recursos de ejecución (unidades funcionales, cachés, interfaces de memoria, etc) → sólo requiere un pequeño incremento en el tamaño del chip
  - Con esta replicación, el procesador parece tener múltiples núcleos (a veces llamados *procesadores lógicos*) y por lo tanto puede ejecutar múltiples flujos (hilos) en paralelo, sin importar si pertenecen al mismo programa o a diferentes
  - El número de replicaciones de estados determina el número de procesadores lógicos del procesador

[1]En los procesadores de Intel esta técnica se denomina Hyper-Threading

Multi-hilado a nivel hardware



Multi-hilado a nivel hardware

- SMT puede mejorar la productividad del procesador (instrucciones ejecutadas por ciclo) siempre y cuando sea posible intercalar instrucciones de múltiples hilos entre los pipelines
- El escenario ideal sería tener múltiples hilos que usen recursos de ejecución diferentes → Lamentablemente esto no es común en la práctica
  - En ocasiones, el número de referencias a memoria de un programa escala con el número de hilos por lo que se puede dar un mejor aprovechamiento del ancho de banda si tenemos un gran número de hilos
- Desventaja de SMT: Si los hilos usan exactamente los mismos recursos, podría haber perdida de rendimiento por la competencia entre ellos. Por ejemplo, programa sensibles al tamaño de caché.

La ganancia por el uso de SMT depende fuertemente del programa a ejecutar. Generalmente, lo que conviene es hacer pruebas con y sin uso de SMT para evaluar su posible beneficio.

Procesadores *Very Long Instruction Word*

- El costo y complejidad del hardware del planificador en los procesadores superescalares crece significativamente a medida que aumenta la emisión de instrucciones, lo que se vuelve una limitación
- El enfoque de los procesadores VLIW consiste en resolver las dependencias y la disponibilidad de recursos a nivel de compilación para determinar aquellas instrucciones que pueden ejecutarse en forma concurrente
- Estas instrucciones son empaquetadas y ejecutadas en paralelo, lo que da lugar a la noción de VLIW

Procesadores Very Long Instruction Word

- Ventajas de los VLIW
  - Como la planificación se hace a nivel de software, los mecanismos de decodificación y emisión de instrucciones son más simples
  - El compilador maneja un contexto más grande para seleccionar las instrucciones y puede usar una variedad más amplia de optimizaciones para lograr mayor paralelismo comparado a una unidad de hardware
- Desventajas
  - Los compiladores no disponen de información de ejecución para tomar decisión (por ejemplo, historial de saltos) → lo que lleva a que consideren políticas conservadoras de planificación
  - Otras situaciones en tiempo de ejecución son muy difíciles de predecir , como los *stalls* debido a fallos de caché → limita el alcance y el rendimiento de la planificación basada en compiladores estáticos
- En resumen, el rendimiento de un procesador VLIW es completamente dependiente de la capacidad del compilador de extraer paralelismo de las instrucciones de un programa

LIMITACIONES EN EL RENDIMIENTO DEL SISTEMA DE MEMORIA

Limitaciones en el rendimiento del sistema de memoria

- En muchas aplicaciones, la limitación se encuentra en el sistema de memoria y no en la velocidad del procesador
- Los dos parámetros fundamentales del sistema de memoria a tener en cuenta son la **latencia** y el **ancho de banda**
- La latencia es el tiempo que transcurre desde que se solicita el dato hasta que el mismo está disponible
- El ancho de banda es la velocidad con la cual el sistema puede *alimentar* al procesador
  - Analogía con manguera y agua

Latencia de la memoria: un ejemplo

- Consideremos un procesador capaz de ejecutar 1 instrucción de punto flotante por nanosegundo (ns) conectado a una memoria con una latencia de 100 ns (sin cachés)
  - El rendimiento pico teórico de este sistema es de 1GFlops

- Supongamos que debemos multiplicar 2 matrices A y B de nxn elementos cada una
  - Para esto se requieren n^3 operaciones de multiplicación/suma → $2n^3$ ns de cómputo
  - Para cada operación de multiplicación/suma se requiere acceder a memoria para buscar ambos operandos → $200n^3$ ns de acceso a memoria
  - Para guardar los resultados en memoria se requiere un acceso por cada elemento → $100n^2$ ns

Latencia de la memoria: un ejemplo

- Si n=32 entonces:
  - Se requieren $2x32^3$ operaciones de multiplicación/suma → $2x2^{15}$ ns de cómputo.
  - Para cada operación de multiplicación/suma se requiere acceder a memoria para buscar ambos operandos → $200x2^{15}$ ns de acceso a memoria
  - Para guardar los resultados en memoria se requiere un acceso por cada elemento → $100x2^{10}$ ns
  - En total: $102{,}64x2^{16}$ ns para resolver 2^{16} operaciones → 9,75 Mflops

El rendimiento baja de 1 Gflop (1000 Mflops) a 9,75 Mflops

Reduciendo la latencia usando cachés

- La memoria caché se caracteriza por ser más rápida y tener menor capacidad que la memoria RAM. También es más costosa.
- El objetivo es disminuir la latencia del sistema de memoria maximizando el número de datos que se acceden desde de la caché.
- Un *cache hit* se da cuando se solicita un dato que está en la caché. La tasa de hits es importante porque incide directamente en la latencia global del sistema

Impacto del uso de caché: un ejemplo

- Consideremos el mismo sistema del ejemplo anterior pero ahora con una memoria caché de 32Kb con latencia de 4 ns y ancho de banda igual a un elemento de la matriz
- Se debe resolver la misma multiplicación de matrices con n=32. Como ambas matrices (A y B) entran en caché, se accede a memoria únicamente una vez por cada elemento:
  - Para esto se requieren n^3 operaciones de multiplicación/suma → $2n^3$ ns de cómputo
  - Para cada operación de multiplicación/suma se requiere acceder a memoria para buscar ambos operandos → Para cada elemento se accede una vez a memoria (100 ns) y el resto a caché (4x(n-1) ns) → en total se requieren (96+4xn) $2n^2$ ns de acceso a memoria
  - Para guardar los resultados en memoria se requiere un acceso por cada elemento → $100n^2$ ns

Impacto del uso de caché: un ejemplo

- Si n=32 entonces:
  - Se requieren $2x32^3$ operaciones de multiplicación/suma → $2x2^{15}$ ns de cómputo.
  - Para cada operación de multiplicación/suma se requiere acceder a memoria o caché para buscar ambos operandos → $448x2^{10}$ ns de acceso a operandos
  - Para guardar los resultados en memoria se requiere un acceso por cada elemento → $100x2^{10}$ ns
  - En total: $9{,}5625x2^{16}$ ns para resolver 2^{16} operaciones → 104,6 Mflops

El rendimiento sube de 9,75 a 104,6 Mflops

  - La mejora obtenida por la inclusión de la caché se basa en la suposición de que habrá una repetición de referencias a determinados datos en una ventana de tiempo pequeña (*localidad temporal*)

Impacto del ancho de banda

- El ancho de banda es la velocidad con la que los datos pueden ser transferidos desde la memoria al procesador y está determinado por el ancho de banda del bus de memoria como de las unidades de memoria

- Una técnica común para mejorar el ancho de banda del sistema consiste en incrementar el tamaño de los bloques de memoria que se transfieren por ciclo de reloj

- El sistema de memoria requiere *l* unidades de tiempo (latencia) para obtener *b* unidades de datos (b es el tamaño del bloque medido en bits, bytes o words)

Impacto del ancho de banda: un ejemplo

- Consideremos el mismo sistema del ejemplo anterior (memoria caché de 32Kb con latencia de 4 ns) pero con ancho de banda igual a 4 elementos de la matriz
- Se debe resolver la misma multiplicación de matrices con $n=32$. Como ambas matrices (A y B) entran en caché, se accede a memoria únicamente una vez por cada 4 elementos:
  - Para esto se requieren n^3 operaciones de multiplicación/suma → $2n^3$ ns de cómputo
  - Para cada operación de multiplicación/suma se requiere acceder a memoria para buscar ambos operandos → Para cada 4 elementos se accede una vez a memoria (100 ns) y el resto a caché (4x(n-1) ns del primero y 3x4xn ns el resto) → en total se requieren (24+4xn) $2n^2$ ns de acceso a memoria
  - Para guardar los resultados en memoria se requiere un acceso por cada 4 elementos → $25n^2$ ns

Impacto del ancho de banda: un ejemplo

- Si n=32 entonces:
  - Se requieren $2x32^3$ operaciones de multiplicación/suma → $2x2^{15}$ ns de cómputo.
  - Para cada operación de multiplicación/suma se requiere acceder a memoria o caché para buscar ambos operandos → $304x2^{10}$ ns de acceso a operandos
  - Para guardar los resultados en memoria se requiere un acceso por cada 4 elementos → $25x2^{10}$ ns
  - En total: $6{,}140625x2^{16}$ ns para resolver 2^{16} operaciones → 162,8 Mflops

El rendimiento sube de 104,6 a 162,8 Mflops

  - La mejora de rendimiento es posible dado que las sucesivas instrucciones usan datos que se encuentran consecutivos en la memoria (*localidad espacial*)

Impacto del acceso no lineal

- El siguiente código computa la suma total por columnas de B:

```
for (i = 0; i < 1000; i++) {
        suma[i] = 0.0;
        for (j = 0; j < 1000; j++)  suma[i] += B[j][i];
}
```

- ¿Cuál es el problema si la matriz B está en memoria por filas?

- ¿Qué sucede al restructurar el código de la siguiente forma?:

```
for (i = 0; i < 1000; i++) suma[i] = 0.0;
for (j = 0; j < 1000; j++)
        for (i = 0; i < 1000; i++)
                suma[i] += B[j][i];
```

- ¿Qué sucede si la matriz es muy grande y no entra en la caché? ¿Cómo hacer para aprovechar la localidad?

Resumen de ideas para mejorar el rendimiento del sistema de memoria

- Explotar localidad espacial y temporal de datos es crítico para amortizar la latencia e incrementar el ancho de banda efectivo
- La relación entre el número de instrucciones y el número de accesos a memoria es un buen indicador temprano del rendimiento efectivo del sistema
- La organización de los datos en la memoria y la forma en que se estructura el código pueden impactar significativamente en el rendimiento final del sistema

PLATAFORMAS DE CÓMPUTO PARALELO

Clasificación de las plataformas de cómputo paralelo

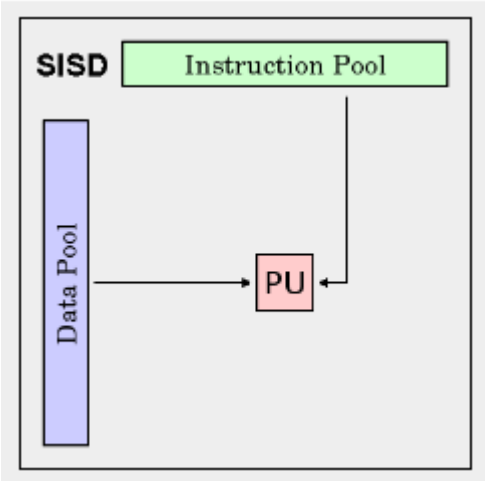
- Las plataformas de cómputo paralelo se pueden clasificar por:
  - Organización lógica: desde el punto de vista del programador
    - Mecanismo de control: forma de expresar tareas paralelas
    - Modelo de comunicación: mecanismo para especificar el intercambio de datos entre tareas
  - Organización física: desde el punto de vista del hardware
    - Espacio de direcciones
    - Red de interconexión

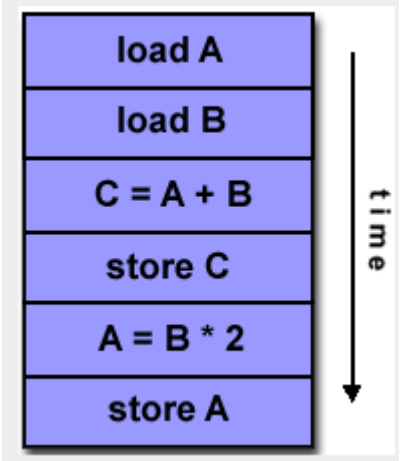
Clasificación por mecanismo de control

- Flynn clasifica las computadoras según el número de instrucciones y datos que se pueden procesar simultáneamente
  - SISD (Single Instruction, Single Data)
  - SIMD (Single Instruction, Multiple Data)
  - MISD (Multiple Instruction, Single Data)
  - MIMD (Multiple Instruction, Multiple Data)

Clasificación por mecanismo de control: SISD

- Las instrucciones son ejecutadas en forma secuencial, una por ciclo de reloj
- En cualquier ciclo de reloj, los datos afectados son aquellos que hace referencia la instrucción en cuestión
- La ejecución se vuelve determinística
- Es el tipo más antiguo de computadoras
- Ejemplos: mainframes, computadoras monoprocesador



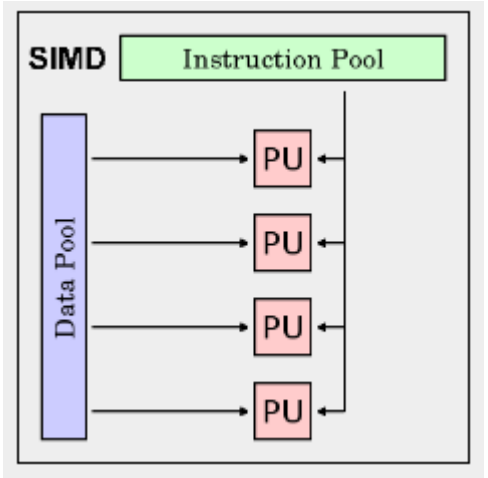


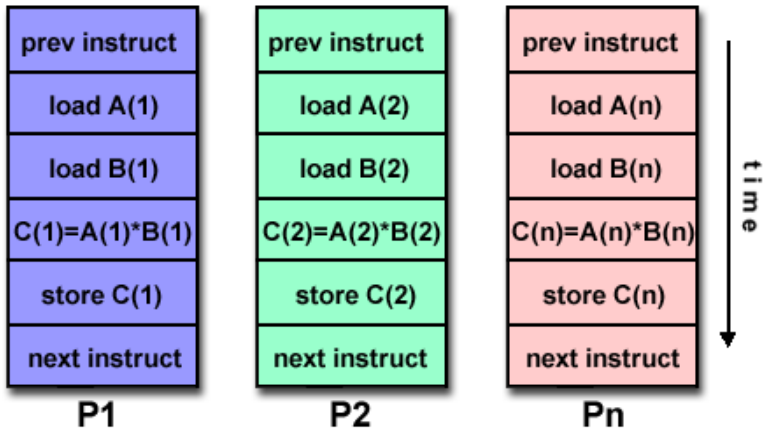

UNIVAC1


IBM 360

Clasificación por mecanismo de control: SIMD

- Todas las unidades de procesamiento ejecutan la misma instrucción sobre diferentes datos
- Ejecución sincrónica y determinística
- Hardware simplificado (comparten control)
- Pueden deshabilitarse selectivamente algunas unidades para que ejecuten o no instrucciones
- Resulta adecuado para problemas con alto grado de regularidad (por ejemplo, procesamiento de imágenes)

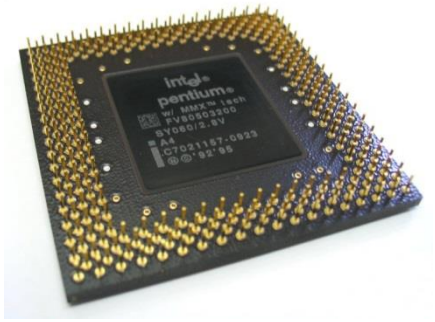




Clasificación por mecanismo de control: SIMD

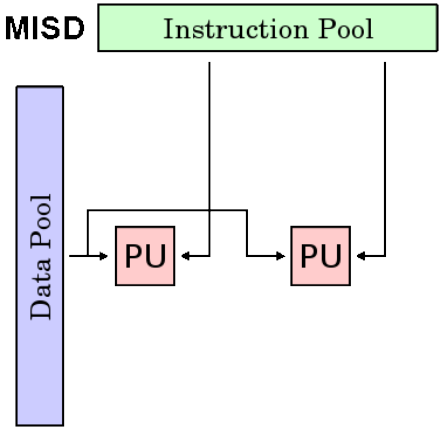
- Ejemplos: Arreglos de procesadores (*Processor Arrays*) como el Thinking Machines CM-2
- En la actualidad, no existen procesadores SIMD “puros” pero sí hay algunos que incluyen componentes basados en este modo de procesamiento
  - Unidades vectoriales (MMX,SSE,AVX,AVX512,etc), GPUs

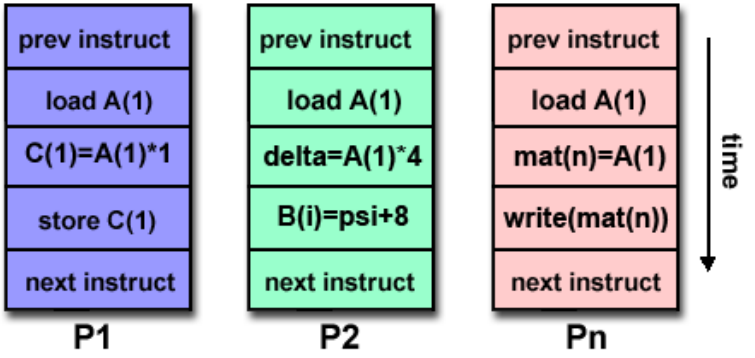

Thinking Machines CM-2



Clasificación por mecanismo de control: MISD

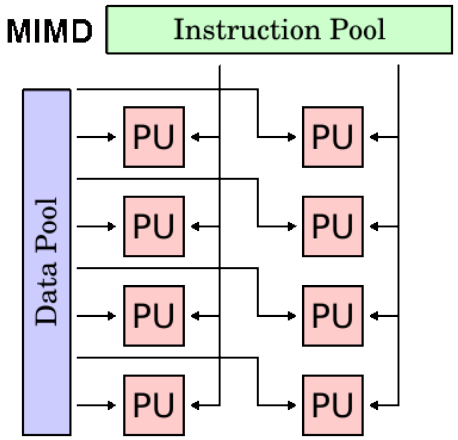
- Las unidades de procesamiento ejecutan diferentes instrucciones sobre el mismo dato
- No existen máquinas reales basadas en este modo de procesamiento
- Podrían usarse para:
  - Múltiples filtros de frecuencia aplicados a una misma señal
  - Múltiples algoritmos de cifrado intentando decodificar el mismo mensaje

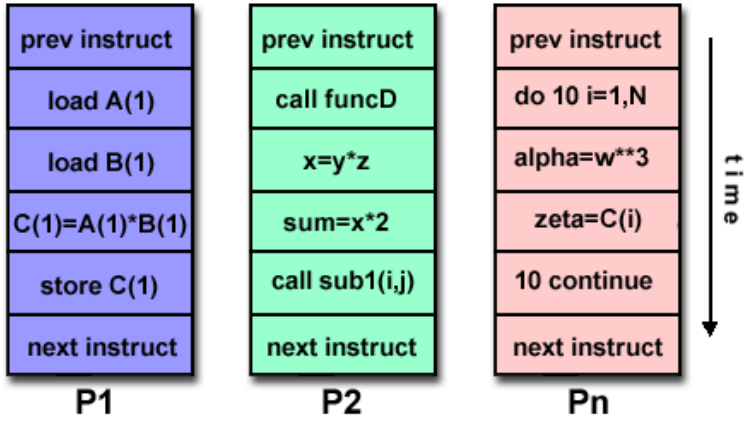




Clasificación por mecanismo de control: MIMD

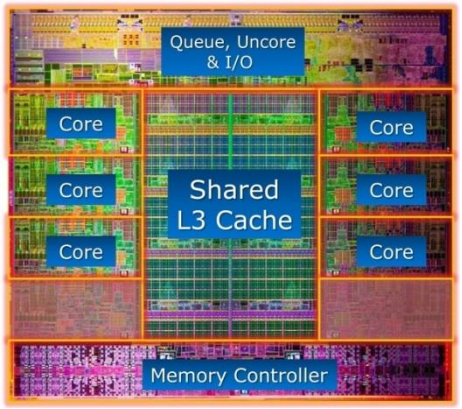
- Las unidades de procesamiento ejecutan diferentes instrucciones sobre diferentes datos
- La ejecución puede ser sincrónica o asincrónica, determinística o no determinisítica
- Pueden ser máquinas de memoria compartida o de memoria distribuida
- Es las clase más común de máquina paralela

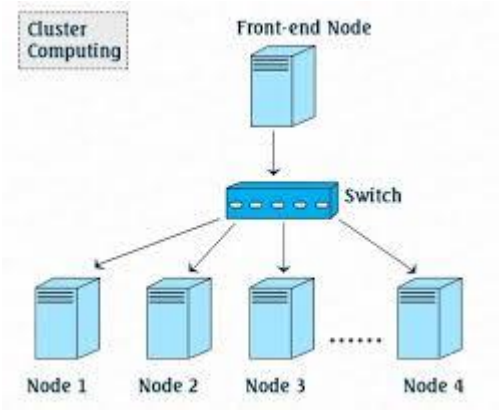




Clasificación por mecanismo de control: MIMD

- Ejemplos: procesadores multicore, clusters, multiprocesadores, grids, supercomputadoras





Clasificación por el modelo de comunicación

- Las tareas paralelas se pueden comunicar de dos formas:
  - Accediendo a una memoria compartida
  - Intercambiando mensajes
- Para ambos modelos de comunicación existen plataformas de cómputo asociadas, aunque en algunos casos también es posible emplear ambos modelos

Clasificación por el modelo de comunicación: memoria compartida

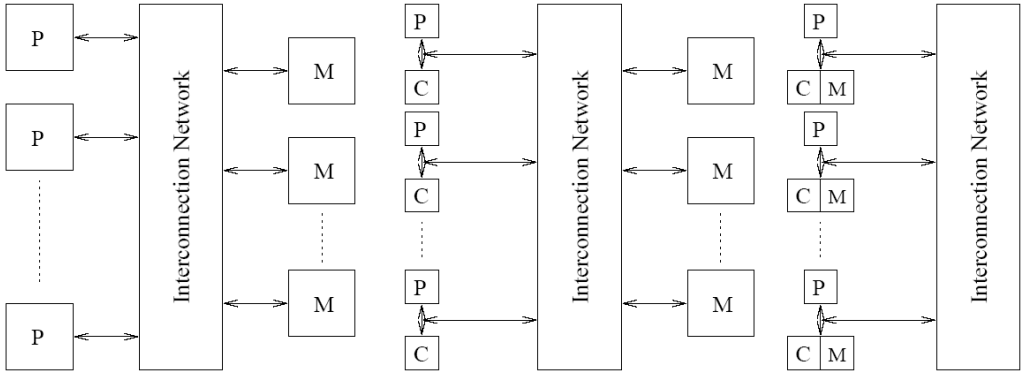
- Las tareas se comunican leyendo y escribiendo variables en un espacio de datos común (memoria compartida)
- Suele ser el modelo más simple de programar (comunicación implícita)
- Se requiere algún mecanismo de sincronización (por ejemplo, semáforos, locks, variables condición, etc) para controlar el acceso a la memoria, resolver competencias y prevenir condiciones de carrera y deadlocks.
- Suele emplearse en las plataformas de memoria compartida

Clasificación por el modelo de comunicación: pasaje de mensajes

- Cada tarea paralela cuenta con su propio espacio de datos y no existe un espacio común entre ellas
- Las tareas paralelas se comunican intercambiando mensajes tanto para sincronización como para comunicación
- Se suele emplear en plataformas de memoria distribuida

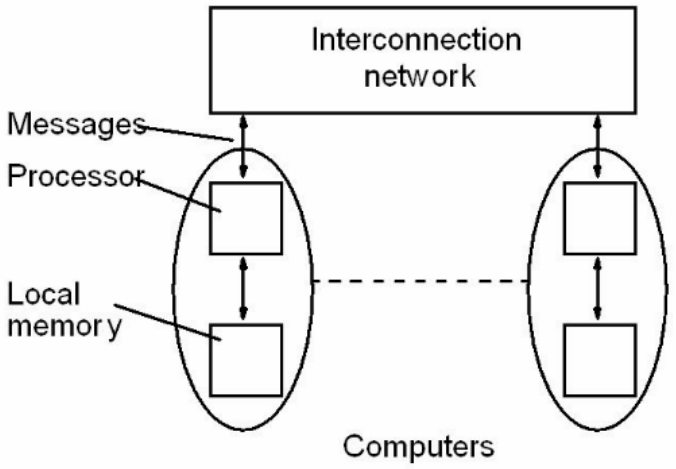
Clasificación por el espacio de direcciones: memoria compartida

- Los procesadores se comunican leyendo y escribiendo variables en un espacio de datos común (memoria compartida)
- Los módulos de memoria pueden ser locales (exclusivos a un procesador) o globales (comunes a todos los procesadores)
- Sub-clasificación por modo de acceso a memoria
  - Acceso uniforme a memoria (UMA)
  - Acceso no uniforme a memoria (NUMA)
- Se necesita un mecanismo de coherencia de caché
- Modelo de programación asociado: memoria compartida. ¿Pasaje de mensajes?



Clasificación por el espacio de direcciones: memoria distribuida

- La vista consiste de *p* nodos de procesamiento, cada uno con su propio espacio de datos
  - Cada nodo puede contener un procesador único o un multiprocesador (variante actual)
- Las nodos se comunican intercambiando mensajes tanto para sincronización como para comunicación
- ¿Hay problemas de consistencia?
- Modelo de programación asociado: pasaje de mensajes. ¿Memoria compartida?



Clasificación por la red de interconexión

- Las redes de interconexión proveen mecanismos de comunicación para los nodos de procesamiento (memoria distribuida) o entre los procesadores y las memorias (memoria compartida)

- Las redes de interconexión se clasifican en:
  - Redes estáticas: enlaces de comunicación punto-a-punto entre los nodos de procesamiento
  - Redes dinámicas: se construyen usando switches y enlaces de comunicación. Los enlaces se conectan unos a otros en forma dinámica usando los switches para establecer los caminos entre los nodos de procesamiento y los bancos de memoria

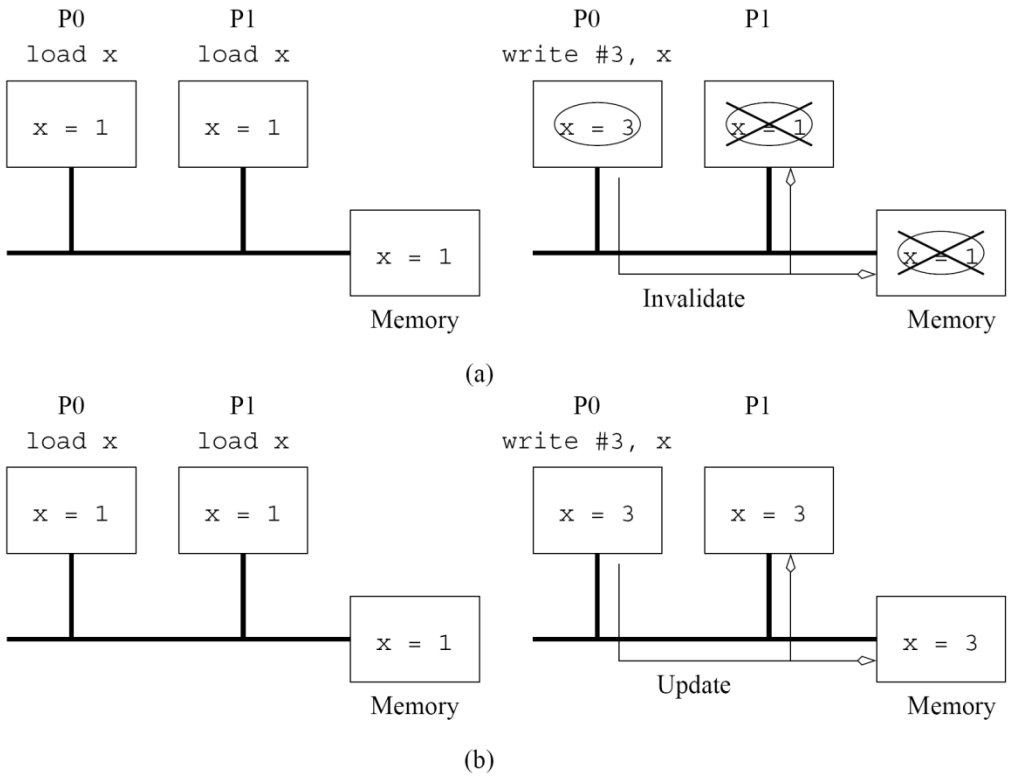
COHERENCIA DE CACHE EN SISTEMAS MULTIPROCESADOR

Coherencia de caché en sistemas multiprocesador

- Las redes de interconexión proveen mecanismos para comunicar datos
- En máquinas de memoria compartida pueden existir múltiples copias del mismo dato → se requiere hardware específico para mantener la consistencia entre estas copias
- El mecanismo de coherencia debe asegurar que todas las operaciones realizadas sobre las múltiples copias son *serializables* → tiene que existir algún orden de ejecución secuencial que se corresponde con la planificación paralela

Coherencia de caché en sistemas multiprocesador

Protocolos de coherencia de caché: (a) invalidación (b) actualización

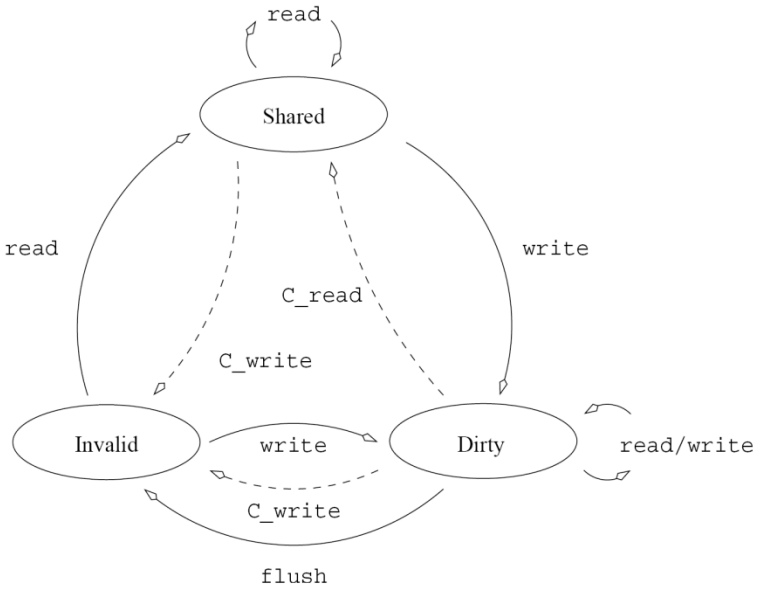


Protocolos de coherencia de caché

- Si un procesador lee un dato una vez y no vuelve a usarlo, un protocolo de actualización puede generar un overhead innecesario
  - En este caso es mejor un protocolo de invalidación
- Si dos procesadores trabajan sobre la misma variable en forma alternada, un protocolo de actualización será mejor opción
  - Se evita/reduce ocio por espera del dato actualizado
- Relación costo-beneficio: los protocolos de actualización pueden producir overhead por comunicaciones innecesarias mientras que los de invalidación pueden producir ocio ante la espera de actualización de un dato
- En la actualidad, la mayoría de los protocolos se basan en invalidación

Protocolos de coherencia de caché basados en invalidación

- Cada copia de un dato se asocia con uno de tres estados: *shared*, *invalid* o *dirty*
- En el estado *shared*, hay múltiples copias válidas del dato (en diferentes memorias). Ante una escritura, pasa a estado *dirty* donde se produjo mientras que el resto se marca como *invalid.*
- En el estado *dirty*, la copia es válida y se trabaja con esta.
- En el estado *invalid*, la copia no es válida. Ante una lectura, se actualiza a partir de la copia válida (la que está en estado *dirty*)



Protocolos de coherencia de caché basados en invalidación: ejemplo

Time ↓

Instruction at Processor 0	Instruction at Processor 1	Variables and their states at Processor 0	Variables and their states at Processor 1	Variables and their states in Global mem.
				x = 5, D y = 12, D
read x		x = 5, S		x = 5, S
	read y		y = 12, S	y = 12, S
x = x + 1		x = 6, D		x = 5, I
	y = y + 1		y = 13, D	y = 12, I
read y		y = 13, S	y = 13, S	y = 13, S
	read x	x = 6, S	x = 6, S	x = 6, S
x = x + y		x = 19, D	x = 6, I	x = 6, I
	y = x + y	y = 13, I	y = 19, D	y = 13, I
x = x + 1		x = 20, D		x = 6, I
	y = y + 1		y = 20, D	y = 13, I

COSTOS DE COMUNICACIÓN

Costos de comunicación

- Uno de los mayores overheads en los programas paralelos proviene de la comunicación entre unidades de procesamiento

- El costo de la comunicación depende de múltiples factores y no sólo del medio físico: modelo de programación, topología de red, manejo y ruteo de datos, protocolos de software asociados.

- Costos diferentes según la forma de comunicación:
  - Modelo simplificado para pasaje de mensajes: $t_{comm} = t_s + m\ t_w$, donde t_s es el tiempo requerido para preparar el mensaje, m es el tamaño del mensaje medido en palabras (*words*) y t_w es el tiempo requerido para transmitir una palabra
  - Memoria compartida: resulta difícil modelar costos por múltiples factores que escapan al control del programador (los tiempos de acceso dependen de la ubicación del dato, el overhead de los protocolos de coherencia son difíciles de estimar, la localidad espacial es difícil de modelar, competencia generada por accesos compartidos depende de la planificación en ejecución, entre otros).

Bibliografía usada para esta clase

- Capítulo 2, “Introduction to Parallel Computing”. Grama, Gupta, Karypis, Kumar. Addison Wesley (2003).

- Capítulo 1, “Introduction to High Performance Computing for Scientists and Engineers”. Georg Hager, Gerard Wellein. CRC Press (2011).

- “Introduction to parallel computing” Blaise Barney, Lawrence Livermore National Laboratory. https://computing.llnl.gov/tutorials/parallel_comp