

Arquitectura de Computadores. Tema 1

Lección 1. Clasificación del paralelismo implícito en una aplicación.

Clasificación del paralelismo

Según su naturaleza, podemos clasificar el paralelismo implícito en tres categorías.

Nivel de paralelismo implícito

El paralelismo puede darse a diferentes niveles: a nivel de programa, a nivel de funciones, a nivel de bloques, a nivel de operaciones, etc.

Granularidad

Es el volumen de trabajo, y será más fina o más gruesa si el volumen de trabajo es menor o mayor. Podemos decir que es el tamaño del conjunto de operaciones que pueden realizarse en paralelo.

Paralelismo de tareas vs paralelismo de datos

No es lo mismo trabajar en paralelo únicamente con datos, únicamente con tareas, o mezclando ambos.

El paralelismo de tareas se encuentra extrayendo la estructura lógica de funciones. Es por esto que está relacionado con el paralelismo *a nivel de función*. El paralelismo de datos se encuentra implícito en las operaciones con estructuras de datos, y relacionado con el paralelismo a nivel del *bucle*.

Condiciones de paralelismo

Para poder realizar ejecuciones en paralelo entre dos bloques de código, deben presentar una independencia de datos.

Dependencia de datos

Decimos que un bloque de código B_2 presenta dependencia de datos con respecto a otro bloque B_1 si:

- Hacen referencia a una misma posición de memoria M (variable).
- B_1 aparece en la secuencia de código antes que B_2 .

Podemos clasificar la dependencia de datos en tres tipos, según la secuencia de operaciones de lectura/escritura que ocurran en los dos bloques con respecto a M .

RAW (Read After Write): dependencia verdadera. No se puede evitar.

WAW (Write After Write): dependencia de salida. Podría evitarse guardando la variable inicial en una variable temporal.

WAR (Write After Read): antidependencia. Podría evitarse realizando la lectura en una variable temporal que contuviese la variable inicial.

El caso *RAR (Read After Read)* no presenta dependencia, ya que simplemente se consulta el dato, sin modificarlo en ningún caso.

Paralelismo explícito

Un computador puede aprovechar el paralelismo entre diferentes entidades de forma explícita, a saber:

- Instrucciones.
- Hebras (*threads*).
- Procesos.

Hebras vs procesos

Un **proceso** comprende el código del programa, y todo lo necesario para su ejecución (datos en pila, segmentos, registros, tabla de páginas, ...). Para comunicar procesos hay que usar llamadas al SO.

Un proceso puede constar de múltiples **hebras** que controlan el flujo de control. Cada hebra tiene su propia pila, y también el contenido de los registros. Para comunicarse entre ellos, utilizan la memoria que comparten.

En general, las operaciones de creación, destrucción, conmutación y comunicación en las hebras consumen menos tiempo.

Lección 2. Clasificación de estructuras paralelas

Computación paralela y computación distribuida

La **computación paralela** estudia el desarrollo y ejecución de aplicaciones en un sistema compuesto por múltiples cores/procesadores que es visto externamente como una unidad autónoma.

Mientras que la **computación distribuida** estudia los aspectos del desarrollo y ejecución de aplicaciones en un sistema distribuido, es decir, en una colección de recursos autónomos situados en distintas localizaciones físicas.

Esta última puede ser a **baja escala**, si se encuentran situados en diversas localizaciones unidas por una misma infraestructura de red local. O bien **computación grid**, un conjunto de dominios administrativos distribuidos en distintas localidades geográficas y que están unidas con una infraestructura de telecomunicaciones.

Ejemplos de esto son la *computación cloud*, que se desarrollan en un *sistema cloud*, un conjunto de recursos virtuales (pay-per-use y con interfaz de auto-servicio) que abstraen los recursos físicos utilizados, con recursos que son captados/liberados de manera inmediata, con acceso múltiple de clientes y con métodos de conexión estándar independientes de la plataforma de acceso.

Criterios de clasificación de computadores

Por lo general la clasificación de computadores se basa en el número de núcleos que posean y operaciones que sean capaces de hacer, lo cual influye directamente en el precio.

Clasificación de Flynn

Se basa en el flujo de instrucciones, si son capaces de trabajar con una única instrucción o con varias y si son capaces de trabajar con un único flujo de datos o con varios.

SISD

Single Instruction, Single Data. Existe un único flujo de datos y una única instrucción a ejecutar en cada instante. Corresponde a los computadores uni-procesador. <!-- (algo de único flujo de datos y una única instrucción, completar) -->

SIMD

Single Instruction, Multiple Data. La instrucción que se codifica va a cada uno de los procesadores (es la misma para todos), donde se le indica de dónde tiene que captar los datos, hace tantas operaciones como unidades de procesamiento se tengan. Por tanto, aprovecha el paralelismo de datos.

MISD

Multiple Instruction, Single Data. No es un sistema que se implemente en la realidad, cada unidad de control estaría conectada con una única unidad de procesamiento con cada flujo de datos.

MIMD

Multiple Instruction, Multiple Data. Múltiples flujos de datos que permiten realizar múltiples instrucciones. Aprovecha, además del paralelismo de datos, el paralelismo funcional. <!-- (algo de múltiples flujos de datos y múltiples instrucciones, completar) -->

Sistema de memoria

Existen dos tipos de máquinas según esta clasificación: multiprocesadores y multicomputadores.

En los **multiprocesadores** todos los procesadores comparten el mismo espacio de direcciones, el programador no necesita conocer dónde están almacenados los datos. Tienen mayor latencia y son poco escalables. Tienen comunicación implícita mediante variables compartidas y los datos no duplicados están en memoria principal. Es necesario implementar primitivas de sincronización, no es necesaria la distribución de código y datos entre procesadores y su programación es mas sencilla.

En los **multicomputadores** cada procesador tiene su propio espacio de direcciones. Están compuestos por computadores completos conectados entre sí por una interfaz de red. El programador necesita conocer donde están almacenados los datos. Tienen menor latencia y son escalables. Tienen comunicación explícita mediante software para paso de mensajes y los datos duplicados están en memoria principal. Su sincronización es mediante un software específico de comunicación, es necesaria la distribución de código y datos entre procesadores y su programación es más difícil.

Comunicación uno-a-uno en un multiprocesador

El nodo fuente manda a memoria el dato, se procesa y se genera una respuesta que es devuelta al nodo fuente. Después el nodo destino realiza la petición del dato a memoria, se procesa y la memoria devuelve el dato a el flujo destino.

Se debe garantizar que el flujo de control *consumidor* del dato lea la variable compartida cuando el *productor* haya escrito en la variable el dato.

Comunicación uno-a-uno en un multicomputador

El nodo fuente copia los datos que desea enviar a un buffer y mediante la red de interconexión los hace llegar a el nodo destino que está en ejecución y a la espera de que le lleguen los datos. Cuando llegan los datos se cargan en memoria de usuario y continua con la ejecución.

Incremento de escalabilidad en multiprocesadores

Se debe aumentar la caché del procesador, usar redes de menor latencia y mayor ancho de banda que un bus y distribuir físicamente los módulos de memoria entre los procesadores compartiendo el espacio de direcciones.

Clasificación completa de computadores según el sistema de memoria

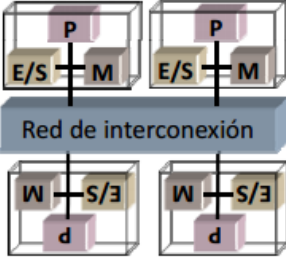
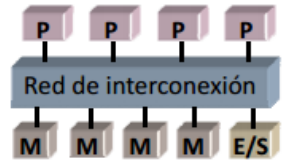
Multi-computadores Memoria no compartida	NORMA <i>No Remote Memory Access</i>	<i>ej. cluster, red de computadores</i>	Memoria físicamente distribuida	+	+
Multi-procesadores Memoria compartida Un único espacio de direcciones	NUMA <i>Non-Uniform Memory Access</i>	NUMA		Escalabilidad	-
		CC-NUMA			
		COMA			
	UMA <i>Uniform Memory Access</i>	SMP <i>Symmetric MultiProcessor</i>	Memoria físicamente centralizada	-	-
					

Figure 1: Clasificación

Arquitecturas con DLP, ILP y TLP

(Thread=Flujo de control)

Lección 3. Evaluación de prestaciones de una arquitectura

Tiempos de CPU

Tiempo de CPU

$$T_{cpu} = Ciclos * T_{ciclo} = Ciclos / Frecuencia \text{ de reloj}$$

Ciclos por Instrucción

$$CPI = Ciclos / NI$$

Donde NI es el n° de instrucciones y donde los ciclos son la suma de los CPI de cada instrucción. Así vemos que hay varias formas de calcular el tiempo de CPU:

$$T_{cpu} = NI * (CPE / IPE) * T_{1ciclo}$$

$$T_{cpu} = NI * CPI * T_{ºciclo}$$

$$T_{cpu} = (N^{º}operaciones / OPI) * CPI * T_{ºciclo}$$

OPI: Número de operaciones que puede codificar una instrucción

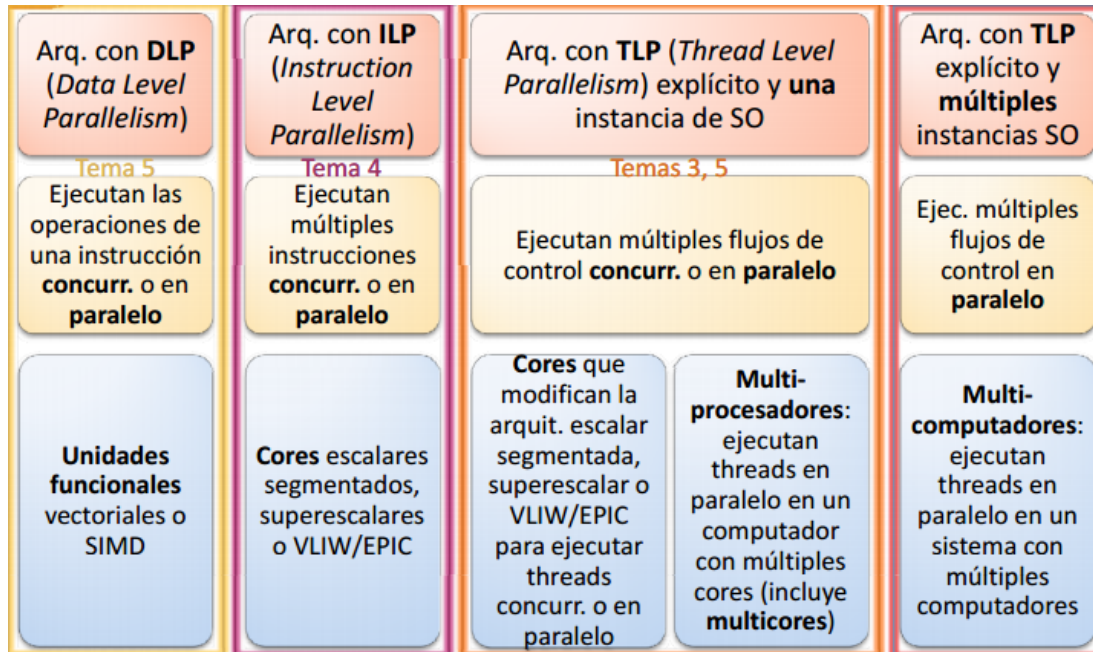


Figure 2: Arquitecturas

CPE: Número mínimo de ciclos transcurridos entre los instantes instantes en que el procesador procesador puede emitir instrucciones instrucciones

IPE: Instrucciones que pueden emitirse (para empezar su ejecución) cada vez que se produce dicha emisión

MIPS

MIPS: Millones de intrucciones por segundo

$$\text{MIPS} = \text{NI} / (\text{Tcpu} * 10^6) = \text{Frecuencia} / (\text{CPI} * 10^6)$$

Depende del repertorio de instrucciones. Puede variar con el programa. Puede variar inversamente con las prestaciones (mayor valor de MIPS corresponde a peores prestaciones)

MFLOPS

MFLOPS: Millones de operaciones en coma flotante por segundo

$$\text{MFLOPS} = \text{Operaciones en coma flotante} / (\text{Tcpu} * 10^6)$$

No es una medida adecuada para todos los programas. El conjunto de operaciones en coma flotante no es constante en máquinas diferentes y la potencia de las operaciones en coma flotante no es igual para todas las operaciones

Conjunto de programas de prueba (Benchmark)

Tipos

- De bajo nivel o microbenchmark
- Kernels
- Sintéticos
- Programas reales
- Aplicaciones diseñadas

Ganancia en prestaciones

Speed-Up

El incremento de velocidad que se consigue en la nueva situación con respecto a la previa (máquina base) se expresa mediante la ganancia de velocidad o speed-up:

$$S_p = V_p / V_1 = T_1 / T_p$$

V_1 Velocidad de la máquina base

V_p Velocidad de la máquina mejorada (un factor p en uno de sus componentes)

T_1 Tiempo de ejecución en la máquina base

T_p Tiempo de ejecución en la máquina mejorada

Ley de Amdahl

La mejora de velocidad, S , que se puede obtener cuando se mejora un recurso de una máquina en un factor p está limitada por:

$$S \leq p / (1 + f(p - 1))$$

donde f es la fracción del tiempo de ejecución en la máquina sin la mejoradurante el que no se puede aplicar esa mejora.

Lo vemos con un ejemplo:

Si un programa programa pasa un 25% de su tiempo de ejecución en una máquina realizando instrucciones de coma flotante, y se mejora la máquina haciendo que estas instrucciones se ejecuten en la mitad de tiempo, entonces $p=2$, $f=0.75$ y $S \leq 2 / (1 + 0.75) = 1.14$

¡Hay que mejorar el caso más frecuente (lo que más se usa)!

Ley enunciada por Amdahl en relación con la eficacia de los computadores paralelos: dado que en un programa hay código secuencial que no puede paralelizarse, los procesadores no se podrían utilizar eficazmente