Informe 01 - Algoritmos Paralelos

Lorena Xiomara Castillo Galdos

Marzo 2017

Contents

1 Hardware paralelo **2**

2 Taxonomía de Flynn **2**
2.1 Single Instruction, Single Data (SISD) 2
2.1.1 Concepto 2
2.1.2 Ejemplo 2
2.2 Single Instruction, Multiple Data(SIMD) 3
2.2.1 Concepto 3
2.2.2 Ejemplo 3
2.3 Multiple Instruction, Single Data(MISD) 3
2.3.1 Concepto 3
2.3.2 Ejemplo 3
2.4 Multiple Instruction, Multiple Data(MIMD) 4
2.4.1 Concepto 4
2.4.2 Ejemplo 4

3 Redes interconectadas **4**
3.1 Interconexiones de memoria compartida 4
3.1.1 Bus 4
3.1.2 Switch 5
3.2 Interconexiones de memoria distribuida 6
3.2.1 Interconexiones directas 6
3.2.2 Interconexiones indirectas 6

4 Latencia y Bandwidth **9**

1 Hardware paralelo

Multiple issue y *pipelining* pueden ser claramente considerados como hardware paralelo, dado que las unidades funcionales son replicadas. Sin embargo dado que esta forma de paralelización no es usualmente visible para el programador, las tratamos como extensiones del modelo básico de von Neumann, y para éste propósito, el hardware paralelo será limitado a hardware que es visible para el programador. Es decir, si puede modificar su código fuente para hacer *exploit*, o si debe modificar su código para hacer *exploit*, entonces, consideraremos al hardware como paralelo.[1]

2 Taxonomía de Flynn

En computación paralela, la taxonomía de Flynn es frecuentemente usada para clasificar arquitecturas de computadora. Clasifica un sistema de acuerdo al número de flujo de instrucciones y el número de flujos de datos que puede manejar simultáneamente. Un sistema clásico de von Neumann es por lo tanto un solo flujo de instrucciones, un solo flujo de datos, o un sistema SISD. Dado que ejecuta una sola instrucción a la vez y puede buscar o almacenar un ítem de datos a la vez.

2.1 Single Instruction, Single Data (SISD)

2.1.1 Concepto

Un sistéma clásico de von Neumann es el sistema *single instruction stream, single data stream* o sistema SISD, ya que ejecuta una sola instrucción a la vez y puede recuperar o almacenar un ítem de datos a la vez.[2]

2.1.2 Ejemplo

Para realizar un cambio de brillo en una imagen, la gran diferencia es que en un sistema SISD cada píxel debe ser operado individualmente requiriendo un gran número de operaciones en una estructura de bucle. Por ejemplo en una imagen de 6 megapíxeles, una imagen que consiste en aproximadamente 6 millones de píxeles. Si tenemos un sólo procesador haciendo las operaciones, tendremos que hacer un bucle 6 millones de veces para aumentar el brillo de la imagen, pero si usamos un sistema SIMD con muchos procesadores, siendo estos muchos procesadores un número que varía con el sistema particular SIMD (pueden ser 1024). Ahora para realizar la operación en muchos píxeles en un solo ciclo de reloj contra un píxel por ciclo de reloj.

Las primeras computadoras caseras fueron sistemas SISD ya que contenían solo un procesador y un banco de memoria.

2.2 Single Instruction, Multiple Data(SIMD)

2.2.1 Concepto

Los sistemas *single instruction, multiple data* o SIMD son sistemas paralelos. Como su nombre sugiere, los sistemas SIMD operan en múltiples flujos de datos aplicando la misma instrucción a múltiple ítems de datos, así que puede pensarse que un sistema SIMD como uno que tiene unidades de control simple y múltiples ALUs.

2.2.2 Ejemplo

SIMD es un método para lograr la paralelización a nivel de datos, un ejemplo perfecto de un sistema SIMD es una Unidad de Procesamiento Gráfico (tarjeta de video). Un sistema SIMD toma como entrada una serie de parámetros de datos (una lista de puntos a dibujar en una malla) y como salida, series de parámetros de datos (una lista estructurada de puntos). El sistema SIMD fue hecho popular en las primeras supercomputadoras a larga escala. Hoy en día, sin embargo, muchas computadoras de escritorio son clasificadas en el reino de SIMD.

2.3 Multiple Instruction, Single Data(MISD)

2.3.1 Concepto

Es un método de arquitectura de computación paralela donde muchas unidades de procesamiento realizan diferentes cálculos en el mismo dato de entrada. Existen muchas arquitecturas que pueden ser clasificadas en este tipo. Entre las más populares están la arquitectura *pipeline*, y computadoras tolerantes a fallas. Una arquitectura *pipeline* es un sistema que toma un conjunto de datos y realiza una serie de operaciones sobre éstos. Los procesadores individuales en una unidad de procesamiento gráfico caen dentro de esta categoría, pero la unidad de procesamiento en sí es un sistema MIMD.

2.3.2 Ejemplo

El método MISD es comúnmente utilizado para hacer estudios de parámetros durante simulaciones científicas. Por ejemplo en el diseño de una antena de transmisión de radio hay varias variables que afectan el desempeño de la antena, éstas necesitan ser estudiadas durante la fase de diseño. Un programador puede desarrollar un simple pedazo de código, y un solo conjunto de datos que modele la antena. Éste data set es luego sujeto de múltiples ejecuciones resultando en una matriz de resultados de cálculo, la cual asiste al diseñador de la antena en determinar el mejor sistema. Muchas aplicaciones basadas en GRID son basadas en un sistema MISD, estas pasarán el mismo conjunto de datos a múltiples nodos de la GRID, cada nodo de GRID calculará algo diferente basado en un parámetro generado aleatoriamente, o un parámetro basado en el número particular de

nodos. Los resultados pueden ser entonces combinados a determinar la mejor solución a un problema.

2.4 Multiple Instruction, Multiple Data(MIMD)

2.4.1 Concepto

El sistema MIMD es una técnica utilizada para lograr paralelismo máximo. Un sistema diseñado con este tipo permitirá a múltiples procesadores realizar diferentes instrucciones en diferentes datos simultáneamente. Los procesadores en un sistema MIMD operan completamente independiente uno de otro, pero pueden confiar en los resultados del otro. Éste es por mucho, el modelo más versátil de programación paralela. Un sistema diseñado basado en este modelo puede ser usado para implementar proyectos de programación de los cuatro modelos, o una mezcla de modelos.

2.4.2 Ejemplo

A partir de 2013 todos los top 10 y más de las top 500 supercomputadoras son basadas en arquitectura MIMD lo cual ha causado que algunas sean divididas de ésta categoría en dos subcategorías. Programas simples, flujos de datos múltiples (SPMD) donde múltiples procesadores ejecutan el mismo programa simultáneamente a diferentes puntos en diferentes datos, ésto difiere de sistemas SIMD en que los sistemas SIMD estan bloqueados en el paso uno con otro.

Las arquitecturas MIMD son utilizadas en una variedad de aplicaciones desde diseño asistido por computadora, simulación, modelado, vision computacional y muchas más.

3 Redes interconectadas

La interconexión juega un rol decisivo en el desempeño de ambos sistemas distribuídos y que comparten memoria (*shared-memory*), incluso si los procesadores y la memoria tienen desempeño virtualmente ilimitado, una interconexión lenta degradará seriamente el desempeño global de todos hasta el programa más simple.

3.1 Interconexiones de memoria compartida

3.1.1 Bus

Es una colección de comunicación de cables paralelos juntos con algún hardware que controla el acceso al bus. La característica clave de un bus es que, los cables de comunicación son compartidos por los dispositivos que están conectados a él. Los buses tienen la virtud de bajo costo y flexibilidad; múltiples dispositivos pueden ser conectados a un bus con un costo adicional pequeño.

Si conectamos un gran número de procesadores a un bus, esperaremos que los procesadores tendrían frecuentemente que esperar para acceder a la memoria principal. Además, como el tamaño de los sistemas de memoria compartida aumenta, los buses son rápidamente reemplazados por interconexiones *switched*.

3.1.2 Switch

Como su nombre sugiere, éstas interconexiones usan *switches* para controlar el enrutamiento de los datos entre los dispositivos conectados. Los *switches* individuales pueden asumir una de dos configuraciones, con estos *switches* y al menos suficientes módulos de memoria como procesadores, solo habrá conflicto entre dos núcleos intentando acceder a memoria si dos núcleos intentan acceder simultáneamente al mismo módulo de memoria.

En la figura 1 se muestra un *crossbar*. Las líneas son enlaces de comunicación bidireccionales, los cuadrados son núcleos o módulos de memoria, y los círculos son *switches*.
Los *switches* individuales pueden asumir una de las dos configuraciones mostradas en la figura 2. En la figura 3 se muestra accesos de memoria simultáneos por los procesadores.

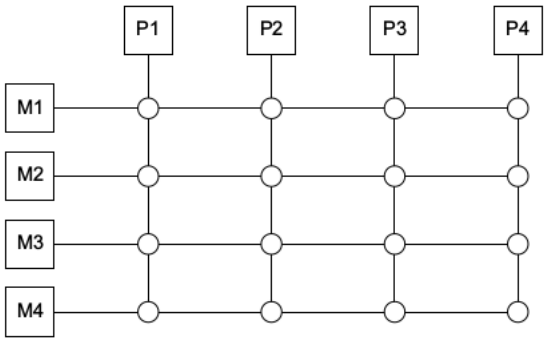

Figure 1: Crossbar

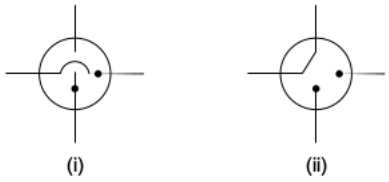

Figure 2: Configuraciones de switch

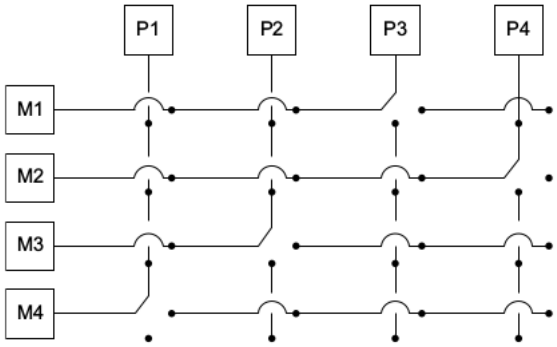

Figure 3: Accesos de memoria simultáneos

Las *crossbars* permiten la comunicación simultánea entre diferentes dispositivos, por lo que son mucho más rápidos que los *buses*. Sin embargo, el costo de los *switches* y enlaces es relativamente alto. Un pequeño sistema basado en bus será mucho menos costoso que un sistema del mismo tamaño basado en *crossbar*.

3.2 Interconexiones de memoria distribuida

3.2.1 Interconexiones directas

En una conexión directa cada *switch* está directamente conectado a un par de memoria-procesador, y los *switches* son conectados entre ellos.

- **Ring**

  Un *ring* es superior a un *bus* dado que permite múltiples comunicaciones simultáneas, si hay p procesadores el número de enlaces es $2p$. Un ejemplo se muestra en la figura 4 a).

- **Thoroidal-mesh**

  Es más expansiva que el *ring* porque los *switches* son más complejos, deben soportar cinco enlaces en lugar de tres, y si hay p procesadores el número de enlaces es $3p$. Un ejemplo se muestra en la figura 4 b).

3.2.2 Interconexiones indirectas

Proveen una alternativa a las conexiones directas. En una interconexión indirecta, los *switches* pueden no estar directamente conectados a un procesador. A menudo son mostrados con enlaces unidireccionales y una colección de procesadores, cada uno tiene un enlace de salida y de entrada, y una *switching network*, como se muestra en la figura 5.

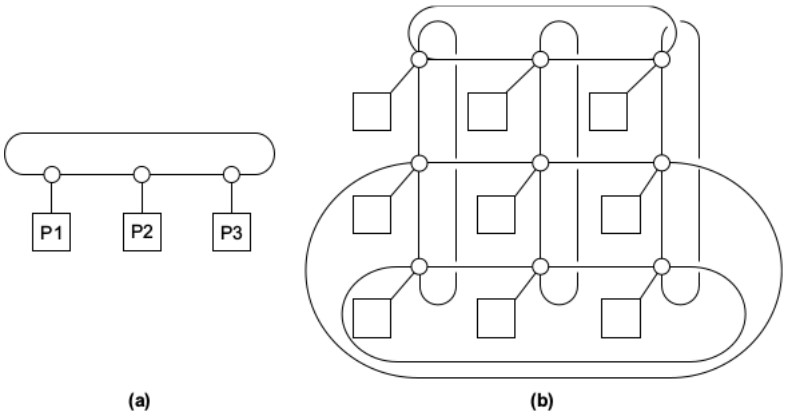

Figure 4: Ring y Thoroidal mesh

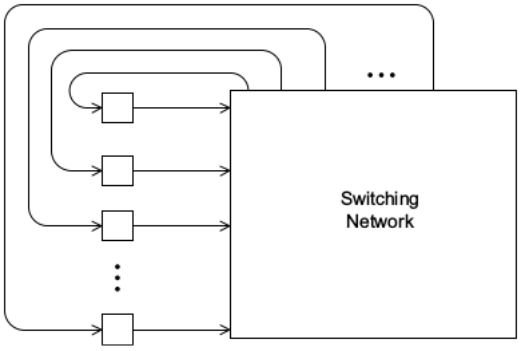

Figure 5: Switching network

- **Crossbar y red omega**

  Son ejemplos relativamente simples de redes indirectas. Como vimos anteriormente en la figura 1 una *crossbar* con enlaces bidireccionales. El diagrama de una *crossbar* con memoria distribuida como se muestra en la figura 6 tiene enlaces unidireccionales. Mientras dos procesadores no intenten comunicarse simultáneamente con el mismo procesador, todos los procesadores pueden comunicarse simultáneamente con otro procesador.

  En la figura 7 se muestra una red omega. Los *switches* son *crossbars* de dos por dos (figura 8). Se observa que a diferencia del *crossbar* existen comunicaciones que no pueden ocurrir simultáneamente.

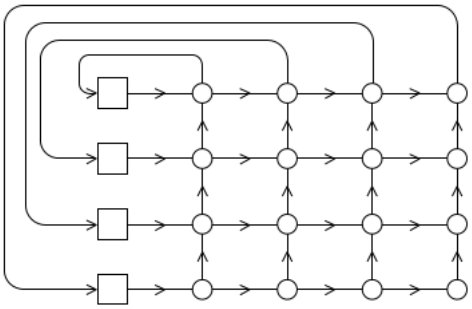

Figure 6: Crossbar interconectada para memoria distribuida

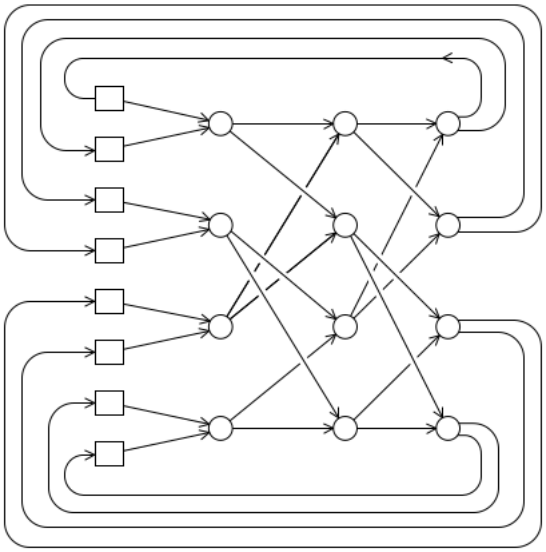

Figure 7: red omega

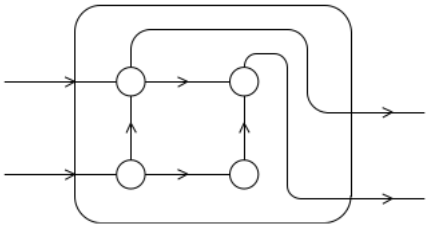

Figure 8: Switch en una red omega

4 Latencia y Bandwidth

Cada vez que los datos son trasmitidos, estamos interesados en cuánto tardará para los datos alacanzar su destino. Esto es cierto si hablamos de transmitir datos entre memoria principal y cache, cache y registro, disco duro y memoria, o entre dos nodos en un sistema de memoria distribuida híbrido.

Existen dos figuras que a menudo son usadas para describir el desempeño de una interconexión (sin importar lo que se esté conectando), la **latencia** y el **bandwidth**.

La latencia es el tiempo transcurrido entre el principio de la fuente a transmitir los datos y los datos en el destino empezando a recibir el primer byte. El *bandwidth* es la tasa en la cual el destino recibe los datos después de que ha empezado a recibir el primer byte. Así que la latencia de una interconexión es l segundos y el *bandwidth* es b bytes por segundo, entonces el tiempo T que demora transmitir un mensaje de n bytes es la ecuación 1.

$$T = l + n/b \tag{1}$$

La latencia es usada algunas veces para describir el tiempo total de transmisión del mensaje. También es a menudo usada para describir el tiempo requerido para cualquier sobrecarga fija involucrada en la transmisión de datos. Por ejemplo, si estamos enviando un mensaje entre dos nodos de una forma en un sistema de memoria distribuida, un mensaje no es sólo datos en bruto. Podría incluir los datos a ser trasnmitidos, una dirección de destino, alguna información que especifique el tamaño del mensaje, alguna información para la corrección de errores, y así sucesivamente. En este contexto, la latencia puede ser el tiempo que se tarda en reunir el mensaje en el lado de envío, el tiempo necesario para combinar las distintas partes, el tiempo para desmontar el mensaje en el lado receptor, el tiempo necesario para extraer los datos brutos del mensaje y almacenarlos en su destino.

References

[1] S.L. Hamilton. *An Introduction to Parallel Programming.* LULU Press, 2013.

[2] Peter Pacheco. *An introduction to parallel programming.* Elsevier, 2011.