

ENTENDIENDO EL PARALELISMO - INTRODUCCIÓN*

José Enrique Alvarez Estrada

Translated By:

José Enrique Alvarez Estrada

Based on *Understanding Parallelism - Introduction*[†] by

Charles Severance

Kevin Dowd

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 3.0[‡]

En cierto sentido, hemos estado hablando acerca de paralelismo desde el inicio del libro. Sólo que en vez de llamarlo "paralelismo", hemos usado términos como "entubamiento", "superescalar" y "flexibilidad del compilador". Conforme nos adentramos en la programación de multiprocesadores, debemos incrementar nuestro entendimiento del paralelismo, para poder comprender cómo programar estos sistemas de forma efectiva. En corto, mientras obtenemos más recursos paralelos, necesitamos encontrar más paralelismo en nuestro código.

Cuando hablamos de paralelismo, necesitamos entender el concepto de granularidad. La granularidad del paralelismo indica el tamaño de los cálculos que se están realizando simultáneamente entre sincronizaciones. Algunos ejemplos de paralelismo, ordenados de acuerdo al tamaño de grano, son:

- Cuando realiza una suma de enteros de 32 bits, usando un sumador con acarreo por desplazamiento, puede usted sumar parcialmente los bits 0 y 1 al mismo tiempo que los bits 2 y 3.
- En un procesador con entubamiento, mientras se decodifica una instrucción, puede recuperarse de la memoria la siguiente.
- En un procesador escalar de dos vías puede ejecutarse, en un solo ciclo, cualquier combinación de una instrucción entera y de punto flotante.
- En un multiprocesador, puede usted dividir las iteraciones de un ciclo entre los cuatro procesadores del sistema.
- Puede usted dividir un arreglo grande entre cuatro estaciones de trabajo conectadas en red. Cada estación puede operar sobre su propia información local, y luego intercambiar los valores de frontera al final de cada paso de tiempo.

*Version 1.1: Mar 29, 2011 1:54 pm -0500

[†]<http://cnx.org/content/m32775/1.3/>

[‡]<http://creativecommons.org/licenses/by/3.0/>

En este capítulo, comenzamos con el *paralelismo a nivel de instrucciones* (entubamiento y supesescalar) y avanzamos hacia el *paralelismo a nivel de hilos de ejecución*, que es el que necesitamos para los sistemas multiprocesador. Es importante señalar que estos niveles distintos de paralelismo generalmente no entran en conflicto. Incrementar el paralelismo a nivel de hilos en problemas de grano grueso, a menudo hace que salga a flote el paralelismo de grano más fino.

El siguiente ciclo está lleno de paralelismo:

```
DO I=1,16000
  A(I) = B(I) * 3.14159
ENDDO
```

Hemos expresado el ciclo en una forma que parece implicar que debe calcularse A(1) primero, seguido por A(2), y así sucesivamente. Sin embargo, una vez completado el ciclo, no tiene importancia si A(16000) se calculó antes o después que A(15999). Bien pudiera haberse calculado primero todos los valores pares de I y luego los nones. Tampoco hubiera hecho una diferencia si las 16,000 iteraciones se hubieran calculado simultáneamente, usando un procesador superescalar de 16,000 vías.¹ Si el compilador es flexible en cuanto al orden en que ejecuta las instrucciones que componen el programa, puede realizar todas estas simultáneamente siempre que haya disponible hardware paralelo.

Una técnica usada por los científicos computacionales para analizar formalmente el paralelismo potencial de un algoritmo, consiste en caracterizar cuan rápidamente se ejecutaría con un procesador superescalar de "infinito número de vías".

No todos los ciclos contienen tanto paralelismo como este sencillo que analizamos. Necesitamos identificar aquellas cosas que limitan el paralelismo en nuestro código, y quitarlas siempre que sea posible. En capítulos previos hemos identificado y retirado el desorden y reescrito los ciclos para simplificarlos.

Este capítulo también proporciona , en muchas formas. Revisaremos la mecánica de la compilación de código, la cuál se aplica aquí, pero no daremos respuesta a todos los "porqués". Las técnicas básicas de análisis de bloques forman la base del trabajo que realiza el compilador cuando trata de lograr mayor paralelismo. Al observar dos piezas de datos, instrucciones, o datos e instrucciones, un compilador debe responder la pregunta: "¿dependen las unas de las otras?" Existen tres respuesta posibles: sí, no, y no lo sabemos. La tercera respuesta es, en la práctica, la misma que sí, porque el compilador debe comportarse conservadoramente cuando no puede garantizar que sea seguro reordenar las instrucciones.

Ayudar al compilador a reconocer el paralelismo es uno de los enfoques básicos que toman los especialistas para afinar el código. Una ligera reescritura de un ciclo, o cierta información suplementaria dada al compilador, pueden convertir una respuesta de "no sabemos" en una oportunidad de lograr paralelismo. Es seguro que hay otras facetas en este proceso de afinación, tales como optimizar los patrones de acceso a memoria de forma que se adapten mejor al hardware, o reelaborar un algoritmo. Y no existe una sola forma correcta de hacerlo para todos los problemas; cualquier esfuerzo de mejora debe involucrar una combinación de técnicas.

¹Curiosamente, esta idea no es tan descabellada como pareciera. Sobre una computadora SIMD (una sola instrucción, múltiples datos) como la Connection CM-2 con 16,384 procesadores, tomaría tres ciclos de instrucción procesar este ciclo completo.