UNIVERSIDAD CARLOS III DE MADRID

Guía Presentación Bloque 3 (Temas 4, 5 y 6)

Departamento de Ingeniería de Sistemas y Automática

RAÚL PÉRULA MARTÍNEZ
LUIS ENRIQUE MORENO LORENTE
ALBERTO BRUNETE GONZALEZ
CESAR AUGUSTO ARISMENDI GUTIERREZ
DOMINGO MIGUEL GUINEA GARCIA ALEGRE
JOSÉ CARLOS CASTILLO MONTOYA

Universidad Carlos III de Madrid

Guía de presentación del bloque III (Tema 4. Condiciones para el paralelismo: Dependencias; tema 5. Segmentación: conceptos básicos; y tema 6. Introducción a la segmentación avanzada: Riesgos)

Para ejecutar diferentes partes de un código en paralelo es necesario asegurarse de que son independientes. Esto requiere el análisis de las dependencias existentes entre dichas partes donde la herramienta que se propone es la realización de un grafo que sintetice las relaciones dentro del código. Este análisis se realiza de forma estática, antes de la ejecución. Entre distintas instrucciones puede haber diferentes dependencias. Por este motivo es necesario clarificar las diferencias entre aquellas dependencias que solo afectan a los datos, de aquellas que afectan a los bucles y saltos, así como las relacionadas con los recursos disponibles en el computador (memoria, unidades de ejecución, etc.).

En cuanto a la automatización de este proceso de detección de dependencias, este proceso requiere la verificación de unas relaciones de dependencias, definidas mediantes las **condiciones de Bernstein**, un conjunto de condiciones formales que determinan si dos procesos (o segmentos de un programa) pueden ejecutarse en paralelo. Las condiciones de Bernstein pueden extenderse del nivel de instrucción a niveles superiores como segmentos de código, subrutinas, procesos o programas.

Una vez entendida la utilidad del análisis de dependencias, es posible continuar con el proceso de optimización dentro de la ejecución del código. Para ello en el tema 5 se define el concepto de segmentación, es decir, la división de la ejecución de instrucciones en etapas, consiguiendo que una instrucción empiece a ejecutarse antes de que hayan terminado las anteriores, lo que implica la ejecución simultánea de varias instrucciones dentro de la CPU. Ya que todas las etapas deben de tardar lo mismo en su ejecución, el tiempo de ciclo será el de la etapa más lenta, más el del retardo provocado por la utilización de los registros intermedios. Comparando este esquema con el secuencial, el tiempo de ciclo será más lento, pero el CPI será menor, lo que provoca un aumento del rendimiento. Ya que si no tenemos en cuenta los riesgos estructurales, tendríamos que en cada ciclo de reloj, termina de ejecutarse una instrucción (CPI=1).

Para obtener estas prestaciones, es necesario conocer el funcionamiento interno de las distintas etapas dentro del procesador. Cada etapa debe ser independiente, por lo que resulta imprescindible prestar atención a los registros que comunican unas con otras para, entre otras cosas, mantener los valores de los datos y las señales de control como los campos de la instrucción descodificada o señales para controlar la lógica interna del procesador. De la misma manera, es importante conocer algunos de los problemas

asociados a estas arquitecturas segmentadas, como es la limitación a la hora de acceder al banco de registros y a la caché de datos.

En el caso ideal, el speedup de una unidad segmentada coincide con el número de etapas del cauce. Esto rara vez ocurre debido a la existencia de operaciones cuyos tiempo de ejecución difieren (p. ej. sumas vs multiplicaciones), por la falta de recursos hardware o porque los datos no estén disponibles cuando se requieran. A estos límites se les conoce por el nombre de riesgos y de manera análoga a las dependencias, pueden afectar a los datos, al hardware y a las instrucciones que cambian el flujo de ejecución. La solución más común (y menos eficiente) en estos casos consiste en detener el cauce hasta que el riesgo desaparezca, causando la pérdida de ciclos dentro del cauce. En el tema 6 se detallan estos riesgos y se proporcionan ejemplos de código que los contienen, así como algunas técnicas para mitigar el impacto de los mismos. Estas técnicas se clasifican según si tienen lugar en tiempo de compilación (técnicas software) o en tiempo de ejecución (técnicas hardware). Algunas de las técnicas más conocidas se revisan en este tema, como son la anticipación o adelantamiento, donde se intenta utilizar los recursos en cuanto estén disponibles, sin esperar a que recorran todo el cauce de ejecución. Del mismo modo, es necesario estudiar las modificaciones hardware que conlleva la implementación de dichas técnicas de resolución de riesgos.