Paralelización especulativa basada en software: resultados y problemas abiertos

Marcelo Cintra, Arturo González-Escribano, Diego R. Llanos, David Orden, Belén Palop

Resumen— **La paralelización especulativa es una técnica que permite extraer en tiempo de ejecución el paralelismo inherente a muchos bucles. El bucle se ejecuta en paralelo, comprobando al mismo tiempo que no se producen violaciones de dependencia entre diferentes iteraciones. En caso de producirse, un mecanismo hardware o software se encarga de descartar el trabajo realizado y reiniciar su ejecución en el orden correcto.**

En este trabajo se muestra el funcionamiento de la paralelización especulativa basada en software, así como las contribuciones de los autores en este campo y algunos problemas que permanecen abiertos.

I. Introducción

La difusión de los sistemas paralelos ha experimentado un gran crecimiento en los últimos años. Casi todos los sistemas operativos modernos permiten dedicar diferentes procesadores para realizar tareas simultáneamente, lo que redunda en una mejora de la productividad. Sin embargo, aún no está resuelto de una forma satisfactoria el problema de cómo ejecutar en paralelo una aplicación secuencial. Para ello existen dos técnicas básicas: extraer el paralelismo inherente a la aplicación en tiempo de compilación, o hacerlo en tiempo de ejecución.

La paralelización automática en tiempo de compilación busca fragmentos de código que puedan ejecutarse en paralelo (típicamente iteraciones de un bucle), asignando las iteraciones independientes a distintos procesadores. Para ello, los compiladores paralelizadores realizan un *análisis de dependencias*, que permite determinar si dos iteraciones pueden ejecutarse o no en paralelo. Este análisis de dependencias en tiempo de compilación no siempre es posible: el flujo de control puede verse influenciado por datos de entrada o estructuras de datos no analizables en tiempo de compilación. Los compiladores comerciales se abstienen de paralelizar el código si no se consigue determinar a ciencia cierta si dos fragmentos de código son independientes, lo que limita el rendimiento obtenido por estas técnicas.

La paralelización en tiempo de ejecución, por su parte, utiliza la información existente durante la ejecución para ejecutar la tarea en paralelo. Para ello existen dos técnicas básicas: la de inspector-ejecutor y la paralelización especulativa. La técnica de *inspector-ejecutor* [1], [2] recorre dos veces el bucle objeto del análisis. Una primera pasada, denominada "bucle de inspección", extrae del bucle la información necesaria para realizar el análisis de dependencia e identificar las iteraciones que pueden ejecutarse en paralelo. Tras el, un "bucle de ejecución" ejecuta en paralelo las iteraciones independientes.

La técnica de *ejecución especulativa* sigue una estrategia diferente. En lugar de realizar un análisis previo de dependencias, el bucle se ejecuta en paralelo, suponiendo (de una manera optimista) que no se producirán violaciones de dependencia. Al mismo tiempo, un mecanismo hardware o software debe supervisar la ejecución, comprobando que estas violaciones no se produzcan. En el caso de producirse una violación de dependencia, las iteraciones calculadas incorrectamente se detienen, reiniciándose su ejecución con los valores correctos. Los mecanismos de soporte hardware a la especulación [3], [4], [5], [6], [7] permiten acelerar al máximo la ejecución paralela, pero requieren modificaciones en los procesadores y/o en las memorias caché. Los mecanismos software, por su parte [8], [9], [10], [11], [12], permiten utilizar sistemas paralelos de procesamiento simétrico (SMP) disponibles comercialmente, a costa de una cierta penalización en el tiempo de ejecución paralelo provocada por la necesidad de ejecutar periódicamente rutinas que se encarguen de comprobar si la ejecución paralela se realiza según la semántica secuencial del algoritmo.

En este trabajo describiremos nuestras contribuciones en este campo, junto con algunos resultados experimentales y una enumeración de los principales problemas que quedan pendientes de resolver. El resto del artículo está estructurado como sigue. La sección I resume el funcionamiento del mecanismo de ejecución especulativa desarrollado por los autores. En la sección II se muestran algunos resultados de la aplicación de esta técnica. La sección III enumera algunos problemas que permanecen abiertos, y la sección IV concluye este documento.

Los mecanismos de ejecución especulativa se basan en extraer threads del código secuencial y ejecutarlos en paralelo, en la esperanza de que no se viole la semántica secuencial del algoritmo. El flujo de la versión secuencial impone, por lo tanto, un orden en los threads. En cualquier momento durante la ejecución, el thread que ejecuta el primer fragmento de código según el orden secuencial se denomina thread *no especulativo*", mientras que ejecuta el último es el thread *más especulativo*". El resto de threads ocupan posiciones intermedias, manteniendo un orden total que permite hablar de threads *predecesores* y *sucesores*. Cada uno de estos threads leen y escriben sobre variables que pueden ser *locales* a ellos o estar *compartidas* entre ambos threads. Las variables locales no presentan ningún problema de dependencias, ya

Marcelo Cintra pertenece a la Universidad de Edimburgo (mc@inf.ed.ac.uk). Arturo González-Escribano, Diego R. Llanos y Belén Palop pertenecen a la Universidad de Valladolid ({arturo|b.palop|diego}@infor.uva.es). David Orden pertenece a la Universidad de Alcalá (david.orden@uah.es).

Fig. 1. Ejemplo de ejecución especulativa en paralelo con reenvío y gestión de una violación de dependencia de tipo RAW.

que el thread accede a ellas en orden. Sin embargo, las variables compartidas entre varios threads deben tratarse con cuidado, ya que un thread puede leer de ellas antes de que un predecesor haya escrito el valor correspondiente. En la práctica, cada thread mantiene una *versión* de las variables compartidas que está utilizando. Cuando un thread desea escribir sobre una de ellas, escribe sobre la versión que este thread mantiene. Si lo que desea es leer una variable compartida, el thread debe comprobar si tiene una copia de la misma, y en caso de no disponer de ella debe buscarse la copia más reciente posible de entre las versiones mantenidas por sus predecesores. Cuando un thread finaliza su ejecución, las versiones de las variables compartidas que haya modificado deben almacenarse en la copia principal de dichas variables. Para preservar la semántica secuencial, este almacenamiento debe hacerse siguiendo el orden de los threads, del no especulativo al más especulativo.

A medida que progresa la ejecución, el sistema de ejecución especulativa comprueba los accesos a las diferentes versiones de las variables compartidas con el fin de detectar posibles *violaciones de dependencia*. Existen tres tipos de violaciones de dependencia. Una violación de tipo RAW (*Read-after-write*se produce cuando un thread consume una versión de un dato distinta de la que ha de generar un predecesor. Si los dos accesos ocurren en orden, es decir, primero se genera el dato y luego se consume, este tipo de violación puede prevenirse reenviando el dato generado al thread que va a consumirlo. En caso contrario, el sistema encargado de monitorizar la ejecución deberá interrumpir al thread que ha consumido el valor incorrecto, y reiniciarlo de modo que utilice el nuevo valor. Las violaciones de tipo WAW (*Write-after-write*) y Las violaciones de tipo WAR (*Write-after-read*) no suelen causar problemas, ya que las modificaciones realizadas por los sucesores se almacenan localmente y no son consumidas por los predecesores.

Cuando se detecta una violación de dependencia, el thread que ha consumido el valor incorrecto debe detenerse, así como también todos los threads que hayan podido verse afectados por esta situación. Esto incluye normalmente a todos los sucesores del thread incorrecto. Los datos calculados por estos threads también se descartan, y su ejecución se reinicia de modo que puedan utilizar el valor actualizado de la variable que provocó la violación de dependencia. La figura 1 muestra cómo pueden ejecutarse varios threads de manera especulativa.

En el caso de la ejecución especulativa por software, se le añaden a la propia aplicación diferentes funciones encargadas

II. Resultados experimentales

III. Algunos problemas abiertos

IV. Conclusiones

Referencias

[1] S.-T Leung and John Zahorjan, "Improving the performance of runtime parallelization," in *Proc. Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, May 1993.

[2] J. Saltz, R. Mirchandaney, and K. Crowley, "Run-time parallelization and scheduling of loops," *IEEE Transactions on Computers*, vol. 40, no. 5, pp. 603–611, May 1991.

[3] Lance Hammond, Mark Willey, and Kunle Olukotun, "Data speculation support for a chip multiprocessor," in *Proc. of the 8th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1998, pp. 58–69.

[4] P. Marcuello and A. González, "Clustered speculative multithreaded processors," in *Intl. Conf. on Supercomputing*, June 1999, pp. 365–372.

[5] J. Gregory Steffan, Christopher B. Colohan, Antonia Zhai, and Todd C. Mowry, "A scalable approach to thread-level speculation," in *Proceedings of the 27th annual International Symposium on Computer Architecture (ISCA)*, June 2000.

[6] J.-Y. Tsai, J. Huang, C. Amlo, D. Lilja, and P.-C. Yew, "The superthread processor architecture," *IEEE Trans. on Computers*, vol. 48, no. 9, pp. 881–902, September.

[7] Y. Zhang, L. Rauchwerger, and J. Torrellas, "Hardware for speculative run-time parallelization in distributed shared-memory multiprocessors," in *Proc. Intl. Symp. High-Performance Computer Architecture*, February 1998, pp. 161–173.

[8] Marcelo Cintra and Diego R. Llanos, "Toward efficient and robust software speculative parallelization on multiprocessors," in *Proceedings of the SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, June 2003.

[9] Francis Dang, Hoo Yu, and Lawrence Rauchwerger, "The R-LRPD Test: Speculative Parallelization of Partially Parallel Loops," in *Proc. of the 16th International Parallel and Distributed Processing Symposium (IPDPS '02)*, April 2002.

[10] M. Gupta and R. Nim, "Techniques for run-time parallelization of loops," *Supercomputing*, November 1998.

[11] L. Rauchwerger and D. A. Padua, "The LRPD test: Speculative run-time parallelization of loops with privatization and reduction parallelization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 2, pp. 160–180, 1999.

[12] P. Rundberg and P. Stenström, "An all-software thread-level data dependence speculation system for multiprocessors," *J. Instruction-Level Parallelism*, vol. 3, October.