Multithreading en Entornos de Ejecución Multitarea[1]

J.C. Moure, Dolores Rexachs y Emilio Luque

Departamento de Informática. Unidad de Arquitectura de Computadores y Sistemas Operativos.
Universidad Autónoma de Barcelona. España.
e-mail: (iarqp, d.rexachs, e.luque)@cc.uab.es

Resumen

Un amplio espectro de los entornos en los que se usan las estaciones de trabajo y los computadores personales requiere de la ejecución de múltiples tareas diferentes durante un mismo intervalo de tiempo. Sin embargo, la mayoría de los procesadores actuales están diseñados para minimizar el tiempo de ejecución de una única tarea, e implementan un modelo de ejecución superescalar que usa el paralelismo entre las instrucciones de la tarea para ejecutar varias instrucciones simultáneamente. Estos procesadores ejecutan varias tareas diferentes intercambiando la tarea en ejecución a lo largo del tiempo. El objeto de este artículo es mostrar que para los entornos de ejecución multitarea existen mejores alternativas que los procesadores actuales. Analizamos los requerimientos de estos entornos y las ventajas que presentan los procesadores *multithreaded* (MT). En primer lugar, la medida fundamental de las prestaciones ya no es el tiempo de ejecución final de cada tarea sino la cantidad de tareas realizadas por unidad de tiempo o *productividad* ("*throughput*"). Los procesadores MT soportan a nivel del repertorio de instrucciones la ejecución explícita de varios flujos de control, o *threads*, reduciendo el tiempo que se pierde al realizar los cambios de contexto. Si además el procesador MT es capaz de ejecutar varias instrucciones de tareas diferentes durante cada ciclo de reloj, compartiendo los recursos del procesador, se puede usar este paralelismo extra para incrementar la *productividad* del sistema.

Una opción básica de diseño es cómo compartir los recursos del procesador entre las tareas activas del procesador. Existen microarquitecturas que permiten compartir entre las tareas prácticamente todos los recursos del procesador, como el *simultaneous multithreading* (SMT), y otras donde un grupo de recursos está reservado a una única tarea, como el multiprocesamiento en un chip (CMP). En este artículo analizamos otras soluciones intermedias, con un mejor compromiso entre la complejidad del SMT y el relativamente bajo rendimiento del CMP. La asignación de los recursos compartidos del procesador a las tareas activas se realiza dinámicamente, en tiempo de ejecución y ciclo a ciclo. Se analizan los problemas ocasionados por la interferencia entre tareas que comparten recursos, especialmente la memoria cache, y se proponen formas de organizar esta compartición para reducir los problemas. Finalmente, se plantea un modelo cualitativo tanto de las microarquitecturas como de las aplicaciones, con objeto de detectar posibles cuellos de botella, de hacer una primera valoración de las diferentes opciones de diseño y de realizar una aproximación teórica a los resultados cuantitativos que se esperan obtener.

Palabras Clave: Multitarea, *Multithreading*, *Simultaneous Multithreading*, Modelado de aplicaciones

Introducción

Un amplio espectro de los entornos en los que se usan las estaciones de trabajo y los computadores personales requiere de la ejecución de múltiples tareas diferentes durante un mismo intervalo de tiempo, especialmente en el dominio de la ingeniería, los negocios y la informática personal. Sin embargo, la mayoría de los procesadores actuales están diseñados para minimizar el tiempo de ejecución de una única tarea, poniendo el énfasis en técnicas para reducir el tiempo medio de ejecución de las instrucciones y sin considerar demasiado la eficiencia con que se aprovechan los recursos del procesador. Existe la presunción, más o menos implícita, de que la mejor forma de ejecutar varias tareas diferentes es dedicar todos los recursos del procesador a una única tarea, e ir intercambiando las tareas en ejecución a lo largo del tiempo. Sin embargo, cuando se pretende ejecutar varias tareas concurrentemente, la medida fundamental de las prestaciones del sistema ya no es el tiempo de ejecución final de cada tarea

[1] Este trabajo ha sido realizado con la ayuda del CICYT, con contrato TIC 95-0868 y la ayuda parcial del "Comisionat per a Universitats i Recerca" (Grups de Recerca Consolidats SGR 1997)

sino la cantidad de tareas realizadas por unidad de tiempo o *productividad* ("throughput"). En este caso, la utilización eficiente de los recursos resulta crucial.

El objeto de este artículo es mostrar que para los entornos de ejecución multitarea existen mejores alternativas que los procesadores actuales. Las mejoras en la tecnología de fabricación de procesadores permiten integrar cada vez un mayor número de recursos dentro del procesador, posibilitando la ejecución de varias instrucciones en paralelo. Los procesadores actuales implementan un modelo de ejecución superescalar y usan técnicas dinámicas de análisis de dependencias entre instrucciones que tratan de aprovechar el paralelismo que existe entre instrucciones más o menos cercanas de una única aplicación para poder ejecutar más de una instrucción por ciclo. Estas técnicas son, entre otras, la ejecución especulativa de instrucciones, la ejecución de instrucciones fuera de orden y el renombrado dinámico de registros para evitar las "falsas" dependencias entre instrucciones. Sin embargo, a medida que aumentamos el número de recursos en el procesador, estas técnicas se van haciendo relativamente menos efectivas, debido a las dependencias que existen entre las instrucciones cercanas y a que los accesos a memoria principal son cada vez relativamente más lentos, [6]. Además, el hardware de control que se necesita para implementar el modelo superescalar es cada vez más y más complejo, llegando incluso a incidir en el tiempo de ciclo del procesador, [10]. El resultado final es que el incremento en el número de recursos internos del procesador y en su complejidad da lugar a un incremento en el rendimiento cada vez menor.

En este artículo analizamos los requerimientos de los entornos multitarea y las ventajas que en ellos presentan los procesadores *multithreaded* (MT). Estos procesadores soportan a nivel del repertorio de instrucciones la ejecución explícita de varios flujos de control, o *threads*. Si además el procesador MT es capaz de ejecutar varias instrucciones de tareas diferentes durante cada ciclo de reloj, compartiendo los recursos del procesador, se puede llegar a conseguir una mayor utilización de los recursos y una mayor *productividad*. Puesto que en este articulo sólo consideramos el caso en que cada *thread* es una tarea completa e independiente, vamos a usar los términos *thread* y tarea como sinónimos.

En el artículo se exploran varias de las posibles alternativas de diseño de la microarquitectura de un procesador MT. Una opción básica de diseño es cómo compartir los recursos del procesador entre las tareas que están activas dentro del procesador. Por un lado, se ha de considerar la lógica de control y los caminos de datos que permitan compartir un determinado recurso. Por otro lado, es necesario añadir mecanismos que arbitren el uso de los recursos compartidos.
La asignación de los recursos compartidos del procesador a las tareas activas se lleva a cabo dinámicamente, en tiempo de ejecución, y debe estar guiada por el objetivo de maximizar el rendimiento y la *productividad*. Esta gestión de los recursos internos del procesador se realiza ciclo a ciclo, permitiendo un alto grado de control en la asignación de prioridades a las tareas. También se analiza en este artículo los problemas ocasionados por la interferencia entre tareas que comparten recursos, especialmente en el caso de la memoria cache, y se proponen formas de organizar la compartición de los recursos para reducir estos problemas. Finalmente, se plantea un modelo cualitativo tanto de las microarquitecturas como de las aplicaciones, con el objetivo de detectar los posibles cuellos de botella, de hacer una primera valoración de las diferentes opciones de diseño y de realizar una aproximación teórica a los resultados cuantitativos que se esperan obtener.

Un caso extremo de microarquitectura que permite compartir entre todas las tareas activas prácticamente todos los recursos del procesador sería el llamado *simultaneous multithreading* (SMT), [11] y [5]. El otro caso extremo, donde un grupo de recursos está reservado a una única tarea activa, vendría representado por el multiprocesamiento en un chip (CMP), [9]. En este artículo analizamos otras soluciones intermedias que pueden proporcionar un mejor compromiso entre la complejidad del SMT y el relativamente bajo rendimiento del CMP.

En la primera sección se analizan los requerimientos específicos impuestos por los entornos de ejecución multitarea. En la segunda sección se bosquejan las tendencias básicas que han determinado el diseño de los procesadores actuales y se muestran las deficiencias que éstos presentan para conseguir alcanzar unas elevadas prestaciones en el contexto de los entornos multitarea. La siguiente sección describe la propuesta de los procesadores MT y estudia las cuestiones relativas a su arquitectura. A continuación hay un apartado que analiza las diferentes opciones en la microarquitectura de estos procesadores, con especial énfasis en las propuestas de tipo SMT y CMP. La penúltima sección trata sobre el modelado de las aplicaciones y de las diferentes microarquitecturas para poder predecir las prestaciones. Finalmente, se describe la metodología experimental que se pretende utilizar.

Caracterización de los Entornos de Ejecución Multitarea

Las características peculiares de los programas que se pretenden ejecutar en un determinado computador imponen diferentes requisitos al procesador. En este apartado exploramos las características de los entornos de ejecución multitarea y las implicaciones que se deducen de ellas a la hora de diseñar un procesador adecuado para su ejecución. Aunque el término multitarea indica, únicamente, que se ejecutará más de una aplicación independiente de forma concurrente, en este apartado vamos a delimitar el tipo de entornos multitarea que vamos a considerar. Para ello, vamos a utilizar la clasificación de Moore [8], que distingue entre aplicaciones de tiempo real, aplicaciones de *máximo esfuerzo* (“best effort”) y aplicaciones de *calidad de servicio* (“quality of service”) o multimedia (ver figura 1).

Las aplicaciones en tiempo real requieren que los resultados se produzcan en un tiempo limitado en relación con las entradas de datos externos. Estas aplicaciones deben asegurar no sólo cumplir sus requerimientos funcionales sino también sus requerimientos temporales. En este tipo de aplicaciones es de vital importancia una planificación en la ejecución de las tareas y en el uso de los recursos que asegure que ningún dato se produce fuera de tiempo. De todos modos, las aplicaciones de tiempo real tienen unos requisitos muy dispares que, en ocasiones, requieren de hardware específico, mientras que en este trabajo nos planteamos el diseño de un procesador de propósito general. Por lo tanto, no vamos a considerar este tipo de aplicaciones en nuestro estudio.

Otro tipo de aplicaciones son aquellas donde se espera que el computador realice el máximo esfuerzo para ejecutarlas lo antes posible. Son aplicaciones donde no hay una limitación en el tiempo máximo de ejecución: como mucho, si una aplicación tarda demasiado tiempo, será una molestia para el usuario. Ejemplos de aplicaciones de *máximo esfuerzo* son la compilación de un programa, la compresión de un fichero, o la simulación de un sistema de moléculas. La mayor parte de las aplicaciones que se llevan a cabo en estaciones de trabajo y computadores personales son de este tipo.

Finalmente, las aplicaciones de tipo multimedia, como la generación de imágenes de vídeo o el reconocimiento de voz o de imágenes, y de creciente importancia en el ámbito de los entornos multitarea [2], presentan requerimientos diferentes a los sistemas de tiempo real o de *máximo esfuerzo*. Estas aplicaciones de cómputo intensivo pueden saturar fácilmente el procesador. Para reducir esta demanda se suelen usar técnicas de compresión de datos en forma de capas, de modo que la información de baja calidad se procesa en primer lugar, como primera capa, y a continuación se van procesando las capas siguientes para mejorar la calidad inicial. De esta manera, una porción mínima de los recursos es suficiente para proporcionar un servicio mínimo, que estará degradado pero será mejor que nada. Si se dispone de recursos adicionales (más tiempo de procesamiento), la calidad de servicio de la aplicación puede mejorarse.

En un entorno de ejecución en el que existe una única aplicación para ser ejecutada por un computador, todos los recursos están asignados a esa aplicación y la única medida de prestaciones que tiene interés es el tiempo de ejecución de la aplicación. Un ejemplo típico de

esta situación sería el caso de una aplicación científica de alto rendimiento, como las dedicadas a predecir el tiempo meteorológico. Sin embargo, en un entorno donde simultáneamente coexisten varias tareas que requieren una cierta proporción de uso del procesador, la medida fundamental de prestaciones deja de ser el tiempo de ejecución de una tarea y pasa a ser la *productividad*, es decir, el número de tareas que se completan por unidad de tiempo. En este tipo de entornos, algunas tareas tienen mayor prioridad que otras y se mezclan requisitos propios de las aplicaciones de *máximo esfuerzo* con los de las aplicaciones de *calidad de servicio*. Es muy importante el uso coordinado y eficiente de los recursos del procesador, con mecanismos de planificación que permitan un control fino de la asignación de los recursos del computador a las tareas. Además, el computador debe ser capaz de adaptarse tanto a periodos de tiempo con un alto número de tareas activas como a periodos con falta de tareas activas. Aunque tradicionalmente el sistema operativo ha sido el encargado de realizar tanto la asignación de tareas como el manejo de los recursos, con escaso soporte por parte del procesador, el uso en los procesadores venideros de mecanismos específicos para soportar la multitarea debe ser estudiado a la luz de las nuevas tendencias que acaecen en el ámbito del diseño de procesadores.

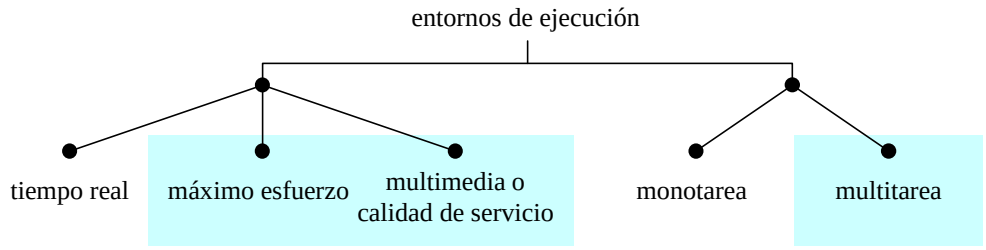

Figura 1. Clasificación de los entornos de ejecución

Tendencias Actuales en el Diseño de Procesadores

Los avances tecnológicos en la fabricación de chips permiten una mayor integración de transistores, lo cual permite, por una parte, la inclusión en un único chip de más recursos dedicados al cómputo, al almacenamiento interno de datos y a la transmisión de datos entre los recursos internos, y, por otra parte, reduce el tiempo de funcionamiento de las puertas lógicas, permitiendo unas frecuencias de reloj mayores. Los ingenieros en el área de arquitectura de computadores investigan nuevas formas de organizar esta cantidad creciente de transistores para reducir el tiempo de ejecución de las aplicaciones y, a la vez, reducir al máximo el impacto que los cambios internos en el procesador puedan provocar en el funcionamiento del software existente, tanto en las aplicaciones en sí como en las herramientas para diseñar nuevas aplicaciones.

La organización interna de los procesadores, lo que se conoce como microarquitectura del procesador, tiende a ser cada vez más distribuida y a replicar cada vez un mayor número de recursos internos con el objeto de poder ejecutar varias instrucciones por ciclo de forma simultánea. Sobre esta microarquitectura distribuida, los procesadores comerciales actuales integran una cantidad considerable de lógica de control que implementa, a partir de una secuencia de instrucciones extraída dinámicamente del programa en ejecución, un modelo reducido de ejecución de flujo de datos (“*dataflow*”), con objeto de encontrar instrucciones independientes que poder ejecutar en paralelo. De esta manera, la extracción y el uso del paralelismo que existe internamente en las aplicaciones es transparente al repertorio de instrucciones particular del procesador, preservando la compatibilidad del código existente. Sin embargo, cuantos más recursos de cómputo se integran en un único chip, mayor es la cantidad de lógica de control que debe añadirse para poder alimentar estos recursos con instrucciones independientes. Este hardware de control es muy centralizado y se vuelve cada vez más y más complejo, haciendo muy difícil poder conseguir altas frecuencias de reloj. Por otro lado, esta estrategia automática de extracción de paralelismo se va haciendo relativamente menos efectiva a medida que se aumenta el número de recursos en el procesador, debido fundamentalmente a

las dependencias que existen entre instrucciones cercanas entre sí y a que los accesos a memoria principal son cada vez relativamente más lentos. El resultado final es que el incremento en el número de recursos internos del procesador y en su complejidad da lugar a un incremento en las prestaciones cada vez menor.

La mala escalabilidad que es característica del modelo centralizado de ejecución superescalar puede ser evitada con una organización de la microarquitectura que proporcione un mejor balance entre el número de ciclos necesarios para ejecutar un programa, el tiempo de ciclo del procesador y la complejidad del diseño, [10]. Las soluciones complejas del tipo de las utilizadas en los procesadores superescalares son capaces de ejecutar muchas instrucciones por ciclo y por lo tanto reducir el número total de ciclos de ejecución, pero a expensas de incrementar el tiempo de ciclo del procesador y el tiempo de diseño y validación de los procesadores. Por el contrario, soluciones excesivamente simples dan lugar a frecuencias de reloj mucho más elevadas y requieren mucho menos tiempo de desarrollo, pero suelen aprovechar poco el paralelismo existente en los programas y necesitan muchos ciclos de ejecución para ejecutar el mismo programa. La cuestión clave de diseño consiste en asegurar que el tiempo de ciclo que se toma como base está determinado, únicamente, por la lógica de cómputo que se ha de utilizar por la mayor parte de las instrucciones y no por la lógica de control, que debería ser transparente, o por la lógica de cómputo que sólo utilizan algunas instrucciones de forma infrecuente. Si a esto se añade que la tendencia en la tecnología de fabricación de chips es que los retardos de las lineas de interconexión dominen pronto a los retardos de las puertas lógicas, [7], algunos estudios llegan a la conclusión de que sólo será posible alcanzar altas frecuencias de reloj si se organiza el procesador en módulos muy rápidos que trabajen más o menos de forma asíncrona, [10].

Básicamente, el modelo multitarea que se ejecuta sobre un procesador actual está soportado por el sistema operativo. El sistema operativo divide los recursos del computador entre las aplicaciones que están ejecutándose concurrentemente, usando estrategias más o menos inteligentes para maximizar el uso de los recursos y aumentar así el número de tareas que se completan por unidad de tiempo. Mientras los periféricos del computador pueden ser utilizados por varias tareas de forma simultánea (discos duros, tarjeta de red, etc.), el procesador, sin embargo, se considera como un único recurso que es asignado a una única tarea activa. Cada tarea tiene asignada una prioridad y el sistema operativo asigna el procesador a una de las tareas con mayor prioridad entre todas aquellas que estén activas. El procesador se comparte entre las diferentes tareas listas a lo largo del tiempo, usando una estrategia que se conoce como ejecución en tiempo compartido. Un reloj externo permite interrumpir al procesador para que el sistema operativo pueda desalojar a la tarea que está en ejecución y evitar así que se monopolice el uso del procesador. Del mismo modo, todas las interacciones del procesador con los periféricos de entrada y salida se realizan mediante interrupciones que desalojan la tarea en ejecución, salvan su estado en memoria, y ponen en ejecución el proceso del sistema operativo adecuado para responder a la interrupción. Este mecanismo de gestión de interrupciones representa el único soporte que proporciona el procesador para adecuarse a la ejecución multitarea.

Las desventajas de los procesadores actuales en el ámbito de los entornos multitarea pueden clasificarse en dos tipos. En primer lugar, estos procesadores no son apropiados para trabajar en unas condiciones donde se exige un cambio de contexto frecuente y un soporte rápido para responder a eventos asíncronos. En segundo lugar, los procesadores superescalares no aprovechan la existencia del paralelismo que existe entre las instrucciones de tareas independientes, que es además muy fácilmente manejable (en contrapartida al paralelismo que explotan los modelos superescalares, que es costoso de manejar).

Respecto al alto coste incurrido en los cambios de contexto, cabe considerar dos importantes factores. En primer lugar, para desalojar una tarea en ejecución se ha de salvar en memoria el estado de los registros que forman parte del modelo de programación del procesador, y para

continuar la ejecución de esta tarea se ha de volver a recuperar estos valores en los registros. Este mismo proceso se ha de realizar tanto para cambiar de una tarea a otra como para permitir al sistema operativo responder a algún evento externo. El tiempo perdido en los cambios de contexto y por el sistema operativo planificando la gestión del evento provoca que ciertos recursos del sistema no se utilicen a su máxima capacidad. En particular, la utilización del procesador también se degrada. Sin embargo, es importante considerar un segundo factor, quizás más importante que el anterior, y que provoca que el coste de los cambios de contexto sea mayor que el tiempo necesario para iniciar la ejecución de una segunda tarea. Los procesadores superescalares, en su afán de minimizar el tiempo de ejecución de una única tarea, utilizan grandes memorias cache para evitar las largas latencias de los accesos a memoria principal, utilizan grandes tablas de predicción para reducir los efectos de las dependencias de control, y empiezan a utilizar otras tablas de predicción para reducir otros tipos de penalizaciones (cache de trazas, predicción de conjuntos en una cache de segundo nivel, etc.). Estas técnicas necesitan un cierto periodo de transición para capturar la "localidad" que manifiestan los programas, o dicho de otro modo, para "memorizar" el comportamiento del programa, y comenzar a reportar sus beneficios. En un entorno en el que los cambios de contexto sean demasiado frecuentes, estas técnicas de "memorización" pueden ser contraproducentes, ya que cada cambio de contexto descarta toda la información recogida para optimizar la ejecución de la tarea que se desaloja, y esta información debe ser reconstruida de nuevo cuando la tarea vuelva a entrar en ejecución

Como resumen de este apartado diremos que los procesadores superescalares están dirigidos, básicamente, a entornos de ejecución monotarea, donde el único requerimiento es ejecutar un programa secuencial lo más rápido posible. Hasta ahora, la estrategia de asignar todos los recursos del chip a una única aplicación, incluso en los entornos multitarea, ha sido más o menos efectiva. Sin embargo, la capacidad que proporciona la tecnología actual para poder ejecutar muchas instrucciones por ciclo no está siendo suficientemente explotada por los procesadores actuales, que proporcionan unos incrementos decrecientes en las prestaciones a medida que se aumenta su coste y complejidad. En los entornos multitarea, estos problemas se agudizan debido a la sensibilidad que presentan al alto coste de los cambios de contexto en un procesador superescalar. Se ha de tener en cuenta, además, que el elevado grado de paralelismo existente en el sistema que no es suficientemente aprovechado por el procesador.

Procesadores 'Multithreaded'

Una de las alternativas para conseguir alcanzar un alto rendimiento con las microarquitecturas distribuidas actuales es utilizar el paralelismo que existe entre secuencias de instrucciones independientes. En lugar de analizar un programa secuencial instrucción por instrucción en la búsqueda de paralelismo, se pueden usar diferentes contadores de programa para seguir la ejecución de múltiples flujos de control, o *threads*, consiguiendo ensanchar de forma efectiva la ventana de instrucciones que pueden ser analizadas para su ejecución en paralelo, y sobrepasando los límites impuestos por las dependencias locales entre instrucciones cercanas entre sí. Los procesadores *multithreaded* (MT) soportan la especificación explícita de *threads* en su repertorio de instrucciones y la ejecución concurrente de estos *threads*. Si además se permite que múltiples instrucciones de *threads* diferentes puedan coexistir en el *pipeline* del procesador, es posible enmascarar la latencia de las operaciones que requieren más de un ciclo de ejecución y reducir la penalización que producen las dependencias de control y de datos. En este caso, el procesador MT puede obtener el paralelismo necesario para mejorar sus prestaciones tanto entre las instrucciones de una misma tarea como entre las instrucciones de diferentes tareas. Esta característica es especialmente importante en el caso de las grandes latencias que incurren los accesos a memoria que no pueden ser satisfechos por la memoria cache.

Los procesadores MT representan un intento de incrementar las prestaciones de un procesador, pero también se pueden considerar una aproximación para proporcionar un modelo general de programación paralela. De hecho, los procesadores MT implementan en su repertorio de

instrucciones un modelo de programación paralela donde el recurso de procesamiento ha sido virtualizado, y se considera que el procesador consta de muchos procesadores virtuales que comparten los recursos del chip. Los procesadores MT extienden el modelo de ejecución tradicional de un único flujo de control, facilitando la programación asíncrona (también conocida como programación orientada a eventos) y el manejo concurrente de todos los recursos del computador. Además, como el repertorio de instrucciones simplemente se aumenta con las instrucciones necesarias para manejar *threads*, la compatibilidad con las aplicaciones existentes se mantiene.

En este artículo, no obstante, se analiza la capacidad de ejecución *multithread* en el contexto de los entornos multitarea. Esta premisa simplifica enormemente el diseño de procesadores MT. En primer lugar, la identificación de flujos de control independientes es directa (cada tarea independiente es un *thread*) y no involucra ni al programador ni al compilador. En segundo lugar, las instrucciones para el manejo de *threads* pueden ser privilegiadas y ejecutarse únicamente en modo supervisor, de forma que afecten al código del sistema operativo pero no al código de las aplicaciones de usuario. Finalmente, gracias a la elevada granularidad del paralelismo entre tareas de alto nivel, no son críticos los mecanismos para crear y manejar *threads* y para permitir la comunicación y sincronización entre ellos (ya que son invocados con muy poca frecuencia). Por el contrario, cuando los *threads* forman parte de una misma aplicación, son relativamente cortos, y están fuertemente relacionados, es necesario un soporte complejo en el procesador que permita una comunicación y una sincronización rápida y eficaz entre los *threads*.

En un entorno multitarea, el principal objetivo de un procesador MT es ejecutar el máximo número de tareas por unidad de tiempo pero, a ser posible, sin penalizar demasiado el tiempo de ejecución de cada tarea particular. Con este objetivo, el procesador puede usar el paralelismo que existe entre las instrucciones de una única tarea para reducir su tiempo de ejecución, y puede usar el paralelismo que existe entre las instrucciones de diferentes tareas para incrementar el número total de instrucciones ejecutadas por unidad de tiempo. Para simular la ejecución simultánea de tareas, en lugar de ir alternando rápidamente la tarea que está activa en el procesador, éste mantiene internamente el contexto de varias tareas en ejecución. De esta manera, el procesador puede ejecutar instrucciones de diferentes tareas de forma realmente simultánea y evitar totalmente el tiempo perdido en los cambios de contexto. El número óptimo de tareas activas que el procesador es capaz de mantener internamente dependerá del número total de instrucciones por ciclo que el procesador sea capaz de ejecutar. De todos modos, como el número de tareas activas en el sistema puede ser mayor que el número de contextos disponibles físicamente, es necesario proporcionar algún mecanismo que permita salvar y recuperar tareas en memoria. Este mecanismo es necesario para asegurar que todas las tareas activas tienen oportunidad de ser ejecutadas y evitar así que se produzca *deadlock*. Este mecanismo debería ser accesible al sistema operativo por medio del repertorio de instrucciones.

La interacción entre las tareas de usuario y las del sistema operativo debe ser analizada con cuidado, porque puede tener una influencia importante en el rendimiento de los sistemas multitarea. En los procesadores superescalares actuales el mecanismo de comunicación entre tareas de usuario y tareas del sistema operativo se realiza mediante la generación de interrupciones desde el programa de usuario que provocan un cambio de contexto y la invocación de un programa del sistema operativo. Este mecanismo, que es el mismo que se utiliza para responder a interrupciones externas, no es útil en un procesador MT, que permite la ejecución simultánea de varias tareas. En este caso, lo que se necesita es la capacidad de ejecutar simultáneamente tareas de usuario, con acceso restringido a ciertos recursos del procesador, y tareas en modo supervisor, que pueden ejecutar cualquier instrucción privilegiada. La finalización de una transferencia de entrada/salida, o la demanda de un servicio del sistema operativo por parte de una tarea de usuario, podría entonces ser llevada a cabo simplemente mediante la activación de una tarea nueva, en modo supervisor, responsable de tratar ese servicio. Las tareas en modo supervisor no necesitan ser interrumpibles, ya que el control del

deadlock en el sistema operativo se puede realizar de forma estática. Lo que sí es necesario es asegurar que al menos una tarea en modo supervisor puede ser ejecutada siempre, bajo cualquier circunstancia.

Por otro lado y tal como hemos dicho anteriormente, las instrucciones para crear, terminar y cambiar la prioridad de las tareas deben ser implementadas en modo supervisor. Es también interesante disponer de al menos dos posibles modos de ejecución. Un primer modo de ejecución debería permitir especificar una tarea de usuario como la de máxima prioridad, limitando a las otras tareas activas a utilizar recursos del procesador sólo cuando éstos no sean utilizados por la tarea más prioritaria. Este modo de ejecución permitiría minimizar el tiempo de ejecución de la aplicación prioritaria y, sin embargo, mantener un alto grado de utilización de los recursos del procesador usando el resto de tareas no prioritarias. Un segundo modo de ejecución permitiría que todas las tareas tuvieran idéntica prioridad, siendo el propio procesador el responsable de organizar y planificar la asignación de sus recursos a las tareas con el objetivo de maximizar el rendimiento del sistema. En este caso, el tiempo final de ejecución de cada tarea deja de tener importancia y se prima la ejecución del mayor número total de instrucciones. Será responsabilidad del sistema operativo decidir en cuál de los dos modos debe trabajar el procesador y, en su caso, que tarea se considera prioritaria. Finalmente, como suponemos que las tareas son prácticamente independientes, para permitir la comunicación de información y la sincronización entre las tareas de usuario es suficiente con disponer de mecanismos generales para compartir páginas dentro del espacio de direccionamiento de la memoria virtual.

Como resumen señalaremos que los procesadores MT permiten explotar el paralelismo que existe entre las diferentes tareas en un entorno multitarea, aumentando la *productividad* del procesador y usando de forma eficiente el conjunto de recursos adicionales que la tecnología permite integrar en un chip. Estos procesadores proporcionan una respuesta rápida a los eventos y permiten un nivel de control muy fino de la ejecución de las tareas. Finalmente, y como mostraremos en la sección siguiente, con un diseño cuidadoso de la microarquitectura, los procesadores MT pueden ser capaces de adaptarse fácilmente tanto al exceso como a la falta de paralelismo entre tareas.

Microarquitectura de los Procesadores *Multithreaded*

En esta sección se explora el espacio de diseño de los procesadores MT en un entorno multitarea. Como hemos comentado en una sección anterior, la microarquitectura debe balancear adecuadamente la complejidad del diseño, que permite ejecutar un mayor número de instrucciones por ciclo, con la posibilidad de aumentar la frecuencia del reloj. En este apartado, para simplificar las explicaciones, vamos a considerar un procesador con capacidad de ejecutar hasta 8 instrucciones por ciclo. Escogemos este tamaño porque parece ser el que deberían presentar los procesadores de la próxima generación, ya que los de la actual generación son capaces de ejecutar hasta 4 instrucciones por ciclo.

La microarquitectura del procesador queda determinada por el conjunto de unidades funcionales, incluyendo las memorias cache, y por la lógica de control. La elección de las unidades funcionales, probablemente, es una de las cuestiones más simples, y este problema de diseño no ofrece diferencia alguna con el mismo problema aplicado a un procesador superescalar. Básicamente, el número y tipo de unidades funcionales viene determinado por la frecuencia de cada uno de los tipos de instrucciones que contienen las aplicaciones que se prevé van a ejecutarse en el procesador. Por ejemplo, si la frecuencia de instrucciones simples de tipo entero (sumas, restas y operaciones lógicas) es de un 35%, se deberán incluir al menos 3 sumadores enteros en el procesador, [4]. Respecto al tamaño de la memoria cache incluida en el procesador, es probable que sea necesario aumentar su capacidad. Sin embargo, la elección de este parámetro parece depender únicamente de las características de las aplicaciones escogidas y no parece que pueda verse influido por otras decisiones de diseño relativas a la microarquitectura. Por último, el hardware de control dedicado a extraer instrucciones

independientes y alimentar el hardware de cómputo sí que presenta numerosas alternativas que se influyen unas a las otras. Las alternativas básicas para "invertir" en este hardware de control son:

1. lógica de tipo superescalar para explotar el paralelismo existente en una única aplicación
2. elementos de almacenamiento para mantener el estado de varias tareas simultáneamente
3. lógica de control para permitir compartir recursos entre diferentes tareas

Tipo de Paralelismo a Explotar

Uno de los compromisos que debe afrontar el diseño de una microarquitectura MT es balancear el hardware de control dedicado a explotar el paralelismo interno de cada tarea y el hardware de control dedicado a explotar el paralelismo existente entre las tareas. Por una parte, a partir de un cierto nivel de prestaciones, mucha parte del hardware que se utiliza para mejorar la ejecución de una única tarea no es usado de forma eficiente. En otras palabras, hay un cierto punto de diseño donde la razón entre el incremento de las prestaciones y el incremento del costo comienza a disminuir. Por otra parte, el hardware utilizado para mantener el estado de varias tareas en ejecución y permitir el lanzamiento de instrucciones de forma concurrente es malgastado cuando sólo existen unas pocas tareas activas. Este hardware no sólo deja de ser utilizado sino que incluso, si se encuentra dentro de algún camino crítico, puede limitar la velocidad del reloj y, por tanto, afectar negativamente a la ejecución monotarea.

Los procesadores superescalares representan uno de los extremos dentro del espacio de diseño. En realidad, un procesador superescalar se puede ver como un caso particular de procesador MT completamente sesgado al caso en que la ocurrencia simultánea de más de una tarea activa es muy infrecuente. Otra propuesta que estaría en el otro extremo del espacio de diseño de las microarquitecturas MT serían los multiprocesadores en un chip (CMP), [9]. Los procesadores CMP integran en un único chip varios procesadores superescalares relativamente simples, donde cada uno de ellos utiliza técnicas para explotar el paralelismo interno de una única tarea, mientras que la ejecución simultánea de una tarea en cada uno de los procesadores permite aprovechar el paralelismo entre tareas. En este caso, los CMPs representan un nuevo caso particular de procesadores MT, sesgados para el caso en que existan al menos tantas tareas activas como procesadores integrados en el chip.

Finalmente, los procesadores SMT ("*simultaneous multithreading*", [11], [5]), combinan las dos aproximaciones previas. Estos procesadores parten de un diseño superescalar típico y lo aumentan con lógica que permite a varias tareas compartir prácticamente todos los recursos del procesador en cada ciclo de reloj. Un procesador SMT se comporta como un procesador superescalar cuando sólo existe una tarea activa, y se puede comportar como un procesador CMP cuando existen varias tareas independientes activas. Los procesadores SMT son los más flexibles y los que consiguen ejecutar un mayor número de instrucciones por ciclo, ya que se adaptan mejor al tipo de paralelismo existente en el entorno. Cuando hay un número insuficiente de tareas, varios de los núcleos de procesamiento integrados en el procesador CMP permanecerán parados, mientras que el procesador SMT puede dedicar esos recursos a extraer el paralelismo interno de las pocas tareas disponibles. Por el contrario, cuando hay falta de paralelismo interno en las tareas, el procesador SMT puede utilizar muchas tareas para maximizar el uso de sus recursos mientras que un procesador CMP tiene limitado el número de tareas a utilizar. En este último caso, un procesador superescalar estará aún más limitado, al no poder ejecutar mas que una única tarea.

Alternativas para Compartir los Recursos del Procesador

La comparación anterior entre procesadores SMT y CMP se podía haber expresado en términos de compartición de recursos. Los procesadores SMT representan la solución más compleja, donde todas las tareas comparten todos los recursos. Los procesadores CMP, sin embargo, representan la solución más simple, donde cada recurso es accesible a una única tarea. En esta

subsección analizaremos las posibilidades que existen a la hora de compartir los recursos del procesador.

La ventaja de la microarquitectura de tipo SMT fue comentada en la anterior subsección: dada su flexibilidad para usar cualquier tipo de paralelismo, los procesadores SMT pueden ejecutar las aplicaciones en un número menor de ciclos. Estas microarquitecturas, no obstante, son muy centralizadas y complejas, y no encajan muy bien con la tecnología actual de fabricación de procesadores. La complejidad extra de tratar con instrucciones de diferentes tareas se añade a la ya de por sí altamente compleja estructura superescalar, pudiendo llegar a afectar al tiempo de ciclo del procesador. Por el contrario, los procesadores CMP, debido a su microarquitectura más simple y distribuida, permiten unas frecuencias de reloj mayores y pueden ser diseñados con una mayor facilidad relativa. Las ganancias de los procesadores SMT en el número de ciclos de ejecución se realizan a costa de un incremento en el tiempo de ciclo, mientras que los procesadores CMP sacrifican el número de instrucciones que se ejecutan por ciclo para reducir el tiempo de ciclo.

En nuestra investigación proponemos otras aproximaciones intermedias que intentan combinar las ventajas de los procesadores SMT respecto al número de instrucciones ejecutadas por ciclo con las altas frecuencias de reloj alcanzables por las microarquitecturas de tipo CMP. Algunos de los elementos de la microarquitectura que son los más críticos para determinar la frecuencia máxima de reloj son la lógica que detecta cuándo una instrucción está libre de dependencias y la lógica de selección de instrucciones activas (sin dependencias), cuya complejidad y tiempo de funcionamiento crecen cuadráticamente con el número de instrucciones ejecutables por ciclo y con el tamaño de la ventana de instrucciones. Otro de los elementos críticos es la lógica que permite utilizar los resultados de operaciones antes de que estos resultados sean escritos en los registros destino ("*bypass logic*"), que también crece cuadráticamente con el número de instrucciones ejecutables por ciclo. Una forma de reducir esta complejidad es limitar el número de tareas que pueden lanzar instrucciones a ejecutar a cada una de las unidades funcionales. Por ejemplo, se pueden agrupar las unidades funcionales en bloques y asignar varias tareas a un único bloque. Las tareas asignadas al mismo bloque de unidades funcionales pueden compartir todos los recursos del bloque, como en el modelo de ejecución SMT. Sin embargo, el acceso por parte de una tarea a un recurso que pertenezca a otro bloque puede comportar varios ciclos de penalización o, incluso, puede no ser posible. Un ejemplo de organización MT por bloques se presenta en la arquitectura Concurro, [3].

Otra posibilidad para organizar una arquitectura MT es considerar que algunas tareas privilegiadas tengan acceso a un mayor número de recursos del procesador, mientras que otras tareas sólo tengan acceso a un número más reducido de recursos. En este caso se pueden contemplar muchas variaciones, tales como el nivel de jerarquía de las tareas (hasta cuántos tipos de tareas con privilegios diferentes se consideran), o también permitir un acceso más rápido o más lento a un determinado recurso en función del privilegio asignado a la tarea. Un caso especial de procesadores MT no homogéneos podría combinar un procesador superescalar con un procesador CMP compuesto de varios núcleos de procesamiento relativamente muy simples. Esta opción es bastante barata en cuanto a costes de hardware y costes de diseño pero permite obtener un alto rendimiento tanto cuando hay una única tarea en el sistema como cuando hay muchas. Otro ejemplo de procesador MT no homogéneo podría permitir que algunas tareas utilicen un modelo superescalar con ejecución de instrucciones fuera de orden, y restringir otras tareas a un modelo de ejecución en orden. En la figura 2 se resumen las posibles alternativas.

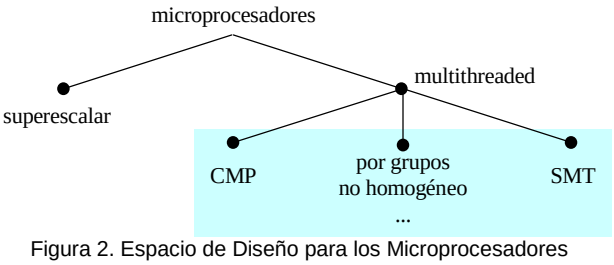

Figura 2. Espacio de Diseño para los Microprocesadores

Gestión de Recursos

Durante la ejecución, los recursos compartidos deben ser asignados a las tareas de una manera tal que se maximice el rendimiento del procesador. Como el procesador ejecuta instrucciones de diferentes tareas cada ciclo de reloj, es posible implementar mecanismos muy precisos que respeten las prioridades asignadas a las tareas. Uno de los recursos compartidos clave es la lógica de decodificación de instrucciones, ya que limita el número de instrucciones de cada tarea que pueden entrar en el *pipeline* de ejecución. Una estrategia muy prometedora para maximizar el rendimiento del procesador, [11], consiste en asignar más recursos de decodificación (y por tanto, permitir que entren más instrucciones en el *pipeline*) a aquellas tareas que tienen menos instrucciones en ejecución, favoreciendo las tareas que tienen menos dependencias entre sus propias instrucciones (esto es, un mayor paralelismo interno). Esta estrategia, sin embargo, presupone que todas las tareas tienen la misma prioridad y que el único requerimiento que se le pone al procesador es maximizar la utilización de los recursos y el número de instrucciones ejecutadas por ciclo. En nuestro trabajo pretendemos considerar también la asignación de prioridades a las tareas, de forma que algunas de ellas tengan más oportunidades para conseguir recursos que otras.

La memoria cache del procesador es un recurso que merece una atención especial. Las memorias cache se usan para mitigar el problema de la disparidad creciente entre la velocidad de los procesadores y la velocidad de acceso de las memorias. La memoria cache es una memoria muy rápida que permite evitar muchos de los accesos a memoria principal y, por lo tanto, reducir el tiempo medio de acceso a memoria. En un sistema multitarea, no obstante, las memorias cache deben ser diseñadas teniendo en cuenta que el conjunto de datos con los que cada tarea trabaja de forma más frecuente debe poder ser almacenado en la cache para todas las tareas activas. También se debe tener en cuenta la posible interferencia causada por la coexistencia de varias tareas, que acceden a datos totalmente independientes. En resumen, la localidad de referencia típica de los programas puede sufrir debido a la dispersión provocada por accesos que no están relacionados entre sí, provocando que las memorias cache sean menos efectivas en reducir la cantidad de accesos a memoria principal.

Hay, sin embargo, dos hechos positivos al compartir la cache entre varias tareas. En primer lugar, la dispersión en las direcciones de memoria accedidas por tareas diferentes reduce la posibilidad de conflictos sistemáticos, permitiendo, por ejemplo, el uso eficiente de organizaciones con varios bancos de memoria en forma entrelazada ("*interleaving*"), que son sustancialmente menos costosas que el añadir varios puertos de acceso a una memoria cache. En segundo lugar, se consigue un mayor grado de flexibilidad en ciertos aspectos del diseño de la jerarquía de memoria. Por ejemplo, el tamaño de las líneas de la memoria cache suele ser relativamente grande para poder aprovechar la localidad espacial de los datos y reducir el tiempo medio de acceso a memoria. Sin embargo, a medida que el tamaño de la línea de cache es mayor y que se aprovecha más la localidad espacial, también es mayor la cantidad de datos que se traen de memoria principal a la memoria cache para luego no ser utilizados. Por el contrario, con un tamaño de línea más pequeño se consigue ahorrar buena parte del ancho de banda utilizado entre el procesador y la memoria principal, a costa de incrementar la latencia media de los accesos. Los procesadores MT presentan la capacidad de solapar el acceso a memoria principal por parte de una tarea con la ejecución de instrucciones o el acceso a

memoria principal por parte de otras tareas. De esta manera se pueden tolerar latencias en el acceso a memoria mucho mayores y se pueden relajar los requerimientos sobre el ancho de banda de la memoria.

Modelado de las Prestaciones

En esta sección investigamos el modelado cualitativo de los entornos de ejecución multitarea y de las microarquitecturas MT. La descripción de los parámetros que conforman el modelo se completa con una descripción del método experimental necesario para obtener medidas concretas de estos parámetros. Este modelo pretende ser apropiado para predecir los cuellos de botella potenciales del sistema e identificar las características críticas del espacio de diseño de los procesadores MT y las características de las aplicaciones que más influencian el rendimiento de los procesadores MT. El propósito de esta sección es proporcionar el soporte teórico necesario para razonar acerca de las ventajas cuantitativas de las diferentes opciones descritas en el apartado anterior para la microarquitectura de un procesador MT, sin tener que depender únicamente de los resultados obtenidos por métodos de simulación.

Consideramos la ecuación $T=N \cdot CPI \cdot T_c$, que define el tiempo necesario para completar un programa (T) como el número de instrucciones ejecutadas (N) multiplicado por el número de ciclos necesarios, de media, para ejecutar cada instrucción (CPI) y por el tiempo de ciclo del procesador (T_c), [4]. Como en este estudio suponemos que no hay cambios en el repertorio de instrucciones y que no se modifica la codificación de las aplicaciones (las instrucciones extras que se han de ejecutar para el manejo de threads, al ser tan infrecuentes, pueden ignorarse), el número total de instrucciones ejecutadas para un determinado conjunto de aplicaciones (N) siempre se mantiene. También suponemos que la tecnología de implementación es la misma para todas las opciones analizadas y, por lo tanto, que las diferencias que puedan producirse en el tiempo de ciclo (T_c) de los procesadores resultantes serán debidas únicamente a la complejidad de la microarquitectura. Finalmente, el número de ciclos por instrucción (CPI) depende tanto de las propiedades de las aplicaciones ejecutadas (su paralelismo intrínseco), como de la habilidad de la microarquitectura para explotar el paralelismo que existe en estas aplicaciones.

Caracterización de los Entornos Multitarea

La creciente complejidad de las microarquitecturas aumenta la cantidad de características de una aplicación que potencialmente pueden limitar el valor del CPI alcanzable, haciendo que las viejas clasificaciones queden obsoletas. En principio, las aplicaciones pueden clasificarse en función del recurso del computador que se espera que sea el cuello de botella. Así, podemos clasificar las aplicaciones, por ejemplo, como limitadas ("*bounded*") por entrada/salida, limitadas por los accesos a memoria, limitadas por la capacidad de cálculo en punto flotante o limitadas por la capacidad de cálculo en punto fijo (o cálculo entero). Sin embargo, esta clasificación pasa por alto una característica que es básica para determinar el rendimiento de una aplicación ejecutada en un procesador superescalar actual: el patrón de dependencias entre las instrucciones de la aplicación. Cuando estas dependencias afectan a instrucciones que están relativamente lejanas entre sí durante la ejecución del programa, es relativamente fácil entonces encontrar suficientes instrucciones independientes con las que alimentar al procesador. En este caso, consideramos que el grado de paralelismo interno del programa es alto. Por el contrario, si las dependencias afectan frecuentemente a instrucciones contiguas o muy cercanas, entonces será difícil llenar el pipeline con suficientes instrucciones independientes y el grado de paralelismo interno será bajo. Estas aplicaciones se pueden clasificar como aplicaciones limitadas por su paralelismo interno, y pueden ser clasificadas a su vez en función del tipo de dependencias que más limitan el paralelismo, bien las dependencias de control o bien las dependencias de datos.

No es sencillo caracterizar una aplicación de forma estática, a partir de su código, ya que éste no refleja directamente el comportamiento dinámico del programa. Para obtener unas medidas significativas es necesario obtener la secuencia real de ejecución de instrucciones a partir de un conjunto representativo de sus datos de entrada. Esta secuencia real se obtiene al ejecutar el programa, bien de forma real o en forma de simulación. Toda la información relativa al programa se puede expresar mediante el grafo dinámico de dependencias. Sin embargo, este grafo es demasiado complejo para ser útil. Lo que se necesita es un conjunto reducido de valores que representen las características más importantes del programa. Una posible medida para caracterizar la aplicación sería el grado de paralelismo interno de la aplicación y que nos proporcionaría la cota máxima del número de instrucciones por ciclo que podría ejecutar un procesador superescalar.

Con objeto de imponer el menor número posible de restricciones ajenas a la propia aplicación y encontrar su máximo grado de paralelismo, consideramos un modelo superescalar más o menos ideal. Asumiremos que se permite la ejecución de instrucciones fuera de orden, con una ventana de instrucciones grande (aunque no infinita), con renombrado de registros para evitar las dependencias "falsas", con predicción de saltos perfecta y con resolución perfecta de las dependencias en memoria. También supondremos que el número de unidades funcionales es grande y que todas ellas tienen una latencia de un ciclo, incluyendo las unidades de acceso a memoria. Simulamos la ejecución de la aplicación en esta microarquitectura ideal y obtenemos el número de instrucciones que, de media, se ejecutan por ciclo, utilizando este valor para cuantificar el grado de paralelismo inherente a la aplicación. Como este paralelismo (el número de instrucciones ejecutadas cada ciclo) varía a lo largo de la ejecución del programa, con periodos con exceso de paralelismo y periodos con falta de paralelismo, es importante cuantificar también la desviación estándar de este valor para modelar su variabilidad, que tiene una importancia fundamental en las prestaciones.

Como este valor no es más que un valor límite, que probablemente no podrá ser alcanzado por ninguna microarquitectura realista, es necesario considerar el comportamiento de la aplicación cuando los recursos de la microarquitectura están limitados. Para ello, se puede analizar la sensibilidad que tiene el valor obtenido al simular la aplicación cuando se varían ciertos parámetros fundamentales del modelo de ejecución superescalar. Por ejemplo, para cuantificar la sensibilidad de las prestaciones al tamaño de la ventana de instrucciones que se analizan en cada ciclo de reloj, se puede reducir (o aumentar) este parámetro en el simulador y obtener el porcentaje de degradación (o mejora) que sufren las prestaciones de la aplicación respecto al caso ideal. Este tipo de análisis se puede usar para clasificar las aplicaciones en función de su sensibilidad a los diferentes recursos del sistema, por ejemplo, a la latencia de los accesos a memoria, a la latencia de las operaciones de punto flotante, al porcentaje de acierto en la predicción de saltos, etc.

Caracterización de las microarquitecturas MT

La microarquitectura determina el tiempo de ciclo (T_c) y la habilidad de explotar tanto el paralelismo interno como el paralelismo entre tareas. El tiempo de ciclo sólo puede determinarse tras un análisis detallado, y nosotros, en este artículo, no vamos a intentar modelarlo. La habilidad para explotar el paralelismo de las aplicaciones, sin embargo, sí que puede ser modelada y depende, fundamentalmente, de la latencia y el ancho de banda de ejecución de los diferentes tipos de operaciones, y de la organización que permite compartir recursos entre las diferentes tareas.

Para cada tipo de operación, la latencia de ejecución determina la penalización por dependencias. Si una instrucción necesita el resultado producido por una instrucción anterior, la latencia de la primera instrucción determinará cuándo se puede comenzar a ejecutar la segunda instrucción y, por lo tanto, para un cierto grafo de dependencias entre instrucciones, determinará el camino crítico y el grado de solapamiento en la ejecución simultánea de instrucciones. El

ancho de banda para cada operación determina, por otra parte, la cantidad de operaciones de un mismo tipo que pueden realizarse por ciclo. Estos valores limitan el modelo de ejecución ideal, en el que se considera que todas las operaciones tienen latencia de un ciclo y el ancho de banda es ilimitado. Una forma de cuantificar este valor sería calcular la media de las latencias usando la frecuencia relativa de cada tipo de operación. El sistema de memoria funciona de forma mucho más compleja que el resto de unidades funcionales y debería, por tanto, modelarse con un mayor número de parámetros.

Por otra parte, la organización que permite compartir recursos entre las tareas en ejecución podría modelarse usando tres valores:
(1) *el grado del procesador*, que es el límite superior para el número de instrucciones que cada ciclo pueden ser lanzadas a la etapa de ejecución,
(2) *el grado de paralelismo externo*, que es el máximo número de tareas que pueden lanzar instrucciones de forma concurrente, y
(3) el *grado de paralelismo interno*, que es el número de instrucciones que, como máximo, pueden ser lanzadas cada ciclo desde una tarea particular.

Un modelo MT con parámetros *d*:*t*:*k* permite ejecutar hasta *k* instrucciones (grado de paralelismo interno) de cualquiera de las *t* tareas (grado de paralelismo externo), para un máximo de *d* instrucciones por ciclo (grado del procesador). Usando esta nomenclatura, un procesador SMT sería un modelo *d:t:d*, donde cada una de las *t* tareas puede lanzar cualquier número de instrucciones, con la restricción de que el número total de instrucciones lanzadas no puede ser mayor que *d*. Esta sería la organización más flexible. Por otra parte, un procesador CMP sería un modelo *d:t:k*, con $k=d/t$, esto es, las *d* instrucciones que se pueden lanzar cada ciclo están repartidas estáticamente entre todas las tareas. Esta nomenclatura no sería suficiente para reflejar estructuras de compartición de recursos más complejas, por ejemplo organizaciones por grupos u organizaciones no homogéneas pero, dada su simplicidad, puede servir como punto de partida del estudio.

La compartición simultánea por varias tareas de los diferentes recursos del procesador permite compensar y reducir los cuellos de botella que se producen en cada aplicación. Por ejemplo, ya hemos comentado que la falta de paralelismo interno de una tarea en un determinado periodo de tiempo puede compensarse con un periodo de exceso de paralelismo interno en otras tareas. En un modelo de tipo SMT, el paralelismo total del conjunto de las aplicaciones será la suma del paralelismo interno de cada aplicación, y la desviación estándar del paralelismo del sistema será relativamente menor que la desviación estándar de cada aplicación particular. En un modelo donde la compartición de recursos está más limitada, se ha de analizar la capacidad del sistema para compensar las variaciones en el paralelismo de las tareas individuales.

Metodología Experimental

Hemos desarrollado un simulador funcional detallado de un procesador superescalar con ejecución fuera de orden, ejecución especulativa y memoria cache no bloqueante. El simulador ha sido construido usando la versión 2.0 de las herramientas de simulación SimpleScalar, [1], con una implementación completamente modificada del núcleo del procesador. Esta implementación sacrifica la velocidad de simulación por la simplicidad y modularidad que permita extender el simulador, de forma relativamente simple, para poder modelar una microarquitectura de tipo MT. Cada módulo lógico de la microarquitectura que se está simulando se corresponde con un módulo software dentro del programa de simulación. De esta manera, para modificar ciertos módulos de la microarquitectura simulada sólo hay que modificar aquellos módulos del programa encargados de simularlos. Además, se ha hecho un esfuerzo importante por simplificar al máximo la microarquitectura subyacente y hacerla lo más distribuida posible. Por ejemplo, la funcionalidad relacionada con la ejecución especulativa, que se suele distribuir por todo el procesador y suele afectar a todos los módulos lógicos del sistema, ha sido encapsulada al máximo dentro del módulo encargado de gestionar los saltos.

Actualmente estamos desarrollando un entorno de ejecución multitarea sobre este simulador superescalar. Algunos de los parámetros del sistema podrán ser seleccionados por el usuario, como el tiempo máximo entre cambios de contexto o el tiempo medio que necesita el sistema operativo para completar las peticiones de entrada/salida de las aplicaciones. También estamos ampliando el simulador para incluir las diferentes microarquitecturas MT descritas en la anterior sección de este artículo. Para ello, se duplicarán muchos de los módulos lógicos del sistema, como la unidad de búsqueda de instrucciones, la unidad de predicción de saltos, los bancos de registros, las unidades de ejecución, etc. Otros módulos lógicos pueden reusarse simplemente cambiando los parámetros que definen el módulo o con pequeñas modificaciones. Finalmente, deben diseñarse módulos especiales de control para orquestar el funcionamiento concurrente de todos los módulos.

Nuestro próximo objetivo es seleccionar algunas de las aplicaciones de los SPEC para formar un entorno multitarea y utilizarlo como ejemplo para tomar medidas en el simulador. En primer lugar se tomarán medidas en el procesador superescalar para obtener una caracterización de cada una de las aplicaciones de entorno, tal y como fue descrito en el apartado anterior. Luego se analizará la ejecución multitarea de estas mismas aplicaciones sobre el mismo procesador superescalar para medir el impacto en las prestaciones de los cambios de contexto y de las llamadas al sistema operativo. En este punto se espera poder determinar los puntos críticos del sistema y evaluar la ganancia que se puede esperar gracias a una microarquitectura mejor adaptada a las condiciones de la multitarea. Finalmente se simulará la ejecución multitarea de este mismo conjunto de aplicaciones sobre el simulador MT y se evaluarán cuantitativamente las diferentes opciones del espacio de diseño descrito en este artículo.

Bibliografía

[1] Burger, D., and Austin, T.M. "*The SimpleScalar Tool Set, Version 2.0*" University of Wisconsin-Madison Computer Science Department, Technical Report #1342, June, 1997

[2] Diefendorff, K., and Dubey,P.K. "*How Multimedia Workloads Will Change Processor Design*" IEEE Computer, pp. 43-45, September 1997

[3] Gunther, Bernard K. "*Multithreading with Distributed Functional Units*" IEEE Trans. On Comp. Vol. 46, no. 4, April 1997

[4] Hennessy, J.L., and Patterson, D., "*Computer Architecture: a Quantitative Approach*" Morgan Kaufmann, 1994

[5] Hirata, H., Kimura, K., Nagamine, S., Mochizuki, Y., Nishimura, A., Nakase, Y., Nishizawa, T. "*An Elementary Processor Architecture with Simultaneous Instruction Issuing from Multiple Threads*" ISCA-19, pp. 136-145, May 1992

[6] Lam, M.D., and Wilson, R.P. "*Limits of Control Flow on Parallelism*" ISCA-19, pp. 46-57, 1992

[7] Matzke, Doug "*Will Physical Scalability Sabotage Performance Gains?*" IEEE Computer, pp. 37-39, September 1997

[8] Moore, Simon W. "*Multithreaded Processor Design*" Kluwer Academic Publishers, Norwell, MA, USA, 1996

[9] Olukotun, K., Nayfeh, B.A., Hammond, L., Wilson, K., and Chang, K. "*The Case for a Single-Chip Multiprocessor*" Proc. ASPLOS VII, pp. 2-11, October 1996

[10] Palacharla, S., Jouppi, N.P., and Smith, J.E. "*Complexity-Effective Superscalar Processors*" Proc. ISCA-24, pp. 206-218, 1997

[11] Tullsen, D.M., Eggers, S.J., and Levy, H.M. "*Simultaneous multithreading: Maximizing on-chip parallelism*" Proc. ISCA-22, pp. 392-403, 1995