Tesina del Máster en Ingeniería de Computadores

Diseño e implementación de planificadores con ahorro energético para tareas de tiempo real estrictas y no estrictas en procesadores multicore multihilo

Diana Bautista Rayo

Directores:
Julio Sahuquillo Borrás
Husein Hassan Mohamed

10 de diciembre de 2008

Resumen

Debido al incremento en las necesidades computacionales, los microprocesadores de altas prestaciones, como por ejemplo procesadores multihilo y multicore, están siendo utilizados para trabajar con sistemas de tiempo real empotrados. Por desgracia el uso de estos microprocesadores más complejos introduce algunos incenvenientes cuando se utilizan para trabajar en sistemas de tiempo real. El primer problema que presentan es que provocan que el cálculo del WCET (Worst Case Execution Time) sea más complicado. Por otro lado el uso de esto microprocesadores introduce nuevas necesidades energéticas. Para hacer frente a estas necesidades, se están aplicando nuevas técnicas de reducción de la energía, como por ejemplo Dynamic Voltage Scaling (DVS). Por desgracia, para los sistemas de tiempo de real, la aplicación de estas técnicas conlleva problemas en la planificabilidad de las tareas.

En este trabajo se presentan dos planificadores con reducción energética para la planificación de tareas de tiempo real basándose en procesadores multicore multihilo de grano grueso. Ambos planificadores implementan las técnicas de DVS para ahorrar energía sin comprometer la planificabilidad del sistema. El primero de ellos es un planificador básico dirigido a tareas de tiempo real no estrictas que ha sido evaluado con diferentes alternativas de distribución de tareas (basadas en el WCET de las tareas) y un modelo de frecuencias de 3 niveles. El segundo planificador está enfocado a la planificación de tareas de tiempo real estrictas. Para su evaluación se han creado diferentes heurísticas de distribución de tareas y la planificación se ha basado en el algoritmo EDF (Earliest Deadline First). Este planificador ha sido evaluado para 3 modelos de frecuencias diferentes (5, 3 y 2 niveles) y diferentes heurísticas de particionado (basadas en el equilibrado de los recursos necesitados por las tareas), que fomentan el solapamiento entre los recursos de memoria y procesador, que junto con las técnicas de DVS reducen el consumo energético del sistema.

Los resultados experimentales de ambos modelos muestran que utilizando un planificador que implemente técnicas DVS de ahorro energético los beneficios aumentan con respecto a un planificador que no implemente dichas técnicas. También se puede observar que cuanto mayor es el número de niveles que implemente el DVS mayor serán los beneficios obtenidos. Si las técnicas de DVS se combinan con buenas heurísticas de particionado, el consumo energético del sistema puede verse reducido a la mitad.

Índice general

1. Introducción **7**

2. Sistemas de tiempo real **13**

3. Trabajo relacionado **19**

4. Planificador con ahorro energético para tareas de tiempo real no estrictas **23**
4.1. Conjunto de tareas y modelo del sistema 23
4.2. Planificador con ahorro energético 24
4.3. Particionado de tareas 26

5. Planificador con ahorro energético para tareas estrictas de tiempo real **29**
5.1. Conjunto de tareas y modelo del sistema 29
5.2. Planificador con ahorro energético 31
5.3. Heurísticas de particionado 33

6. Entorno experimental **35**
6.1. Diseño y caracterización de la carga 36

7. Resultado Experimentales **41**
7.1. Modelo de tareas de tiempo real no estrictas 41
7.2. Modelo de tareas de tiempo real estrictas 43

8. Conclusiones y trabajo futuro **49**

A. Nuevas heurísticas de particionado **51**

Capítulo 1

Introducción

En la actualidad los procesadores están experimentando un crecimiento del número de funcionalidades que incorporan. Su uso se ha extendido considerablemente y se hallan en todos los campos, desde aparatos electrodomésticos, periféricos y controladores hasta llegar a los grandes equipos computacionales que requieren aplicaciones informáticas más complejas. Los procesadores evolucionan de manera continua, ofreciendo configuraciones hardware más pequeñas, rápidas, fiables y baratas, permitiendo generalizar su uso. Entre otras cosas, pueden encontrase en teléfonos móviles, PDAs, video consolas, ordenadores portátiles, sistemas de navegación en coches y en sitios no tan obvios como los procesadores que se encuentran en la mayoría de electrodomésticos modernos. A este tipo de aplicaciones se las denomina genéricamente aplicaciones de tiempo real o empotradas [1].

Figura 1.1. Dispositivos con procesadores empotrados.

Siguiendo esta evolución de los procesadores, las aplicaciones que ejecutan los sistemas de tiempo real pueden ser más complejas y precisas puesto que se dispone de una mayor velocidad de proceso. Como hemos mencionado anteriormente, los procesadores empotrados han incrementado el número de funcionalidades actuales y, debido a esto, incluyen mecanismos complejos desarrollados para arquitecturas de altas prestaciones, como por ejemplo predicción dinámica de saltos, ejecución fuera de orden, multihilo dinámico, etc. Algunos ejemplos de este tipo de procesadores serían el Ubicom IP3023 [2], el cual tiene implementados 8 hilos hardware o el ARM11 [3] el cual incorpora un pipeline de 8 estados, caches y predicción dinámica de saltos. Además, existen implementaciones del ARM11 con multicores [4].

Desafortunadamente, estas técnicas de altas prestaciones dificultan el cálculo del WCET (Worst Case Execution Time) de las tareas de tiempo real. Una de las principales preocupaciones cuando trabajamos con sistemas de tiempo real es garantizar las limitaciones impuestas por el deadline de las tareas de tiempo real ejecutadas en un hilo cuando este hilo tiene que ejecutarse de manera concurrente con otros hilos [5, 6]. Una solución muy simple, como por ejemplo ejecutar las tareas de tiempo real solas, podría degradar el rendimiento del sistema total [7]. Para hacer frente a estos aspectos, el planificador debe ser capaz de controlar de una manera muy precisa el intercambio de los recursos internos del procesador entre las tareas de tiempo real.

Por otro lado, el desarrollo de procesadores más complejos y potentes y la aplicación de técnicas de altas prestaciones en los procesadores empotrados requieren, a su vez, de un consumo energético mayor, lo que provoca problemas de durabilidad de los sistemas. Además, este consumo energético es un gran inconveniente para aquellos sistemas que precisen del uso de baterías (sistemas móviles), y en menor medida para todos los sistemas en general. Por ejemplo, existen sistemas que pueden estar localizados en entornos donde el ahorro energético es esencial para asegurar la operabilidad y duración del sistema, como las aplicaciones espaciales, donde un menor consumo se traduce en menores paneles solares y

menores baterías, es decir en menor peso y dimensión. Debido a esto, la investigación en técnicas de ahorro de consumo energético para procesadores empotrados ha ido ganando importancia [8, 9, 10].

En este contexto, la técnica de ahorro energético Dynamic Voltage Scaling (DVS) [8, 11, 12] es ampliamente conocida. Por ejemplo, el Transmeta Crusoe [13], el Intel Xeon [14] y el AMD DuronTM Móvil [15] la implementan. Esta técnica, permite la reducción de la energía consumida por el procesador, a base de reducir el voltaje suministrado al procesador y al mismo tiempo, reduciendo proporcionalmente la velocidad de ejecución de las tareas. Consideraciones generales sobre la latencia de los circuitos implican una relación lineal entre el voltaje suministrado y la velocidad [16]. Por lo tanto, utilizando la técnica DVS, se puede reducir la energía consumida para ejecutar un determinado conjunto de tareas reduciendo la velocidad del procesador mediante el voltaje y consiguiendo así un decremento cuadrático en la potencia disipada. Esto implica que un pequeño decremento en la velocidad del procesador, tiene como consecuencia un importante ahorro energético. Sin embargo, estos importantes ahorros energéticos debidos a la reducción de la velocidad implican un incremento en los tiempos de ejecución de las tareas. Teniendo en cuenta que en los sistemas con restricciones temporales, como lo son los sistemas de tiempo real, el correcto funcionamiento de la aplicación depende en gran medida del tiempo en el que se producen los resultados, se debe ser muy cuidadoso en el uso de esta técnica, utilizándola únicamente cuando el sistema no requiere la máxima capacidad del procesador ya que si no se podrían presentar situaciones donde el sistema se encuentre sobrecargado, cosa que podría poner en peligro la planificación del sistema[17, 18, 19].

A diferencia de los sistemas informáticos en donde las tareas comienzan a ejecutarse y terminan, en los sistemas de tiempo real, sistemas de control, las tareas tienen un funcionamiento periódico continuo, es decir, no acaban nunca. Si una tarea es ejecutada a mayor velocidad terminará antes que si es ejecutada a una velocidad más reducida. Como es de suponer, el consumo de energía es mayor a mayor velocidad de ejecución de las ta-

reas, pero podría darse el caso que si la tarea termina más rápidamente consuma menor cantidad de energía que si termina más tarde. Sin embargo, en las aplicaciones de tiempo real, al ser tareas periódicas, éstas no finalizan, sino que al finalizar una instancia, deben esperar la siguiente instanciación de la tarea para continuar la ejecución, con lo que resultará energéticamente favorable ejecutar estas aplicaciones a la velocidad mínima que garantice los plazos temporales a la vez que el tiempo en que el procesador está libre es minimizado.

Debido a las razones descritas anteriormente, se necesitan nuevos planificadores de tiempo real que dirigiremos a entornos empotrados multicore multihilo. Estos planificadores deberán mantener bajo el consumo de energía, mientras adaptan la velocidad del procesador a las necesidades de tiempo real de las cargas. En otras palabras, el planificador deberá hacer un balance entre garantizar las restricciones de tiempo real a la vez que el consumo energético es mínimo [17, 18, 19]. Para mantener el consumo de energía mínimo mientras garantiza los deadlines, el planificador no debe simplemente seleccionar la tarea que debe ser ejecutada sino también el nivel energético del DVS mínimo que garantice las restricciones de tiempo real.

En un entorno multicore, cada core puede tener su propio regulador DVS, lo que llamaríamos un DVS local, o puede tener un único regulador para todos los núcleos, forzando los mismos a trabajar a la misma velocidad (DVS global). Tener un regulador para cada núcleo es más caro a la vez que complejo, ya que se necesitan más reguladores y la red de distribución energética es más compleja. Además se ha demostrado [20] que si las tareas están distribuidas de forma equilibrada, un DVS global puede ser tan eficiente como un DVS local.

Los procesadores de grano grueso multihilo son un área prometedora de investigación en relación con los sistemas de tiempo real, ya que ofrecen un buen equilibrio entre el consumo y la planificabilidad. Estos procesadores son capaces de cambiar el hilo cuando se ejecuta un evento de larga latencia (por ejemplo, un acceso a la memoria principal). Esta

característica intrínseca de estos procesadores, actúa como un catalizador para aumentar el tiempo que el procesador y la memoria pueden trabajar simultáneamente (tiempo de solapamiento). Si el tiempo de solapamiento aumenta, el tiempo global de ejecución de un conjunto de tareas puede reducirse, con lo que la frecuencia podría ser reducida y el ahorro de energía aumentado. Si las tareas que se ejecutan en diferentes hilos de un mismo procesador son complementarías atendiendo a sus requisitos de procesador y memoria, el tiempo de solapamiento aumentaría, por lo tanto, estos procesadores ofrecen un escenario natural para explotar las características DVS.

La presente tesina se estructura de la siguiente manera. En el capítulo 2 se hace una breve descripción de los sistemas de tiempo real. Los distintos tipos de modelos de tareas, las características de las mismas y una descripción del planificador EDF, en el que está basado uno de los planificadores de ahorro energético propuestos. En el Capítulo 3, se comenta el estado del arte en lo que a planificadores con reducción energética en sistemas de tiempo real se refiere. A continuación se presentan los dos modelos de planificadores propuestos (Capítulos 4 y 5) y se detallan el modelo del sistema y de tareas utilizado para cada uno de ellos así como la implementación del planificador. A continuación se describe el entorno experimental en el cual se ha trabajado así como el diseño y caracterización de la carga. En el Capítulo 7 se muestran los resultados obtenidos tras los experimentos para cada uno de los planificadores con ahorro energético y por último en el Capítulo 8 se comentan las conclusiones obtenidas y el trabajo futuro.

Capítulo 2

Sistemas de tiempo real

La característica fundamental de los sistemas de tiempo real es que su correcto funcionamiento no depende únicamente del resultado lógico de la computación sino también del tiempo en el que se producen los resultados. A veces, es más conveniente tener un resultado impreciso en un cierto instante a tenerlo con gran precisión pero demasiado tarde. Por tanto, el parámetro de rendimiento más importante para estos sistemas es el cumplimiento de las restricciones temporales impuestas por la aplicación en tiempo de diseño. Los sistemas de tiempo real, según el tipo de restricción temporal que poseen, pueden clasificarse en:

- Sistemas de tiempo real estricto (Hard real time)
- Sistemas de tiempo real firme (Firm real time)
- Sistemas de tiempo real no estricto (Soft real time)

Los sistemas de tiempo real estricto son aquellos en los que es absolutamente imperativo el cumplimiento de todos los plazos temporales, debido a que su incumplimiento puede dar lugar a una catástrofe, en vidas humanas o económicas. Básicamente, estos sistemas se hallan en aplicaciones de control industrial, aplicaciones aeronáuticas, etc. En los sistemas de tiempo real estrictos, es importante poder asegurar, antes de poner en funcionamiento el

sistema, que se respetarán todas las restricciones temporales de las tareas, es decir, que el sistema sea planificable.

En los sistemas de tiempo real no estrictos, el tiempo de finalización es importante, pero el sistema seguirá funcionando aunque los plazos no se cumplan, simplemente habrá una degradación de la calidad de servicio, un ejemplo de estos sistemas son los sistemas multimedia en general. En la Figura 2.1 se puede observar un ejemplo de sistemas de tiempo real.

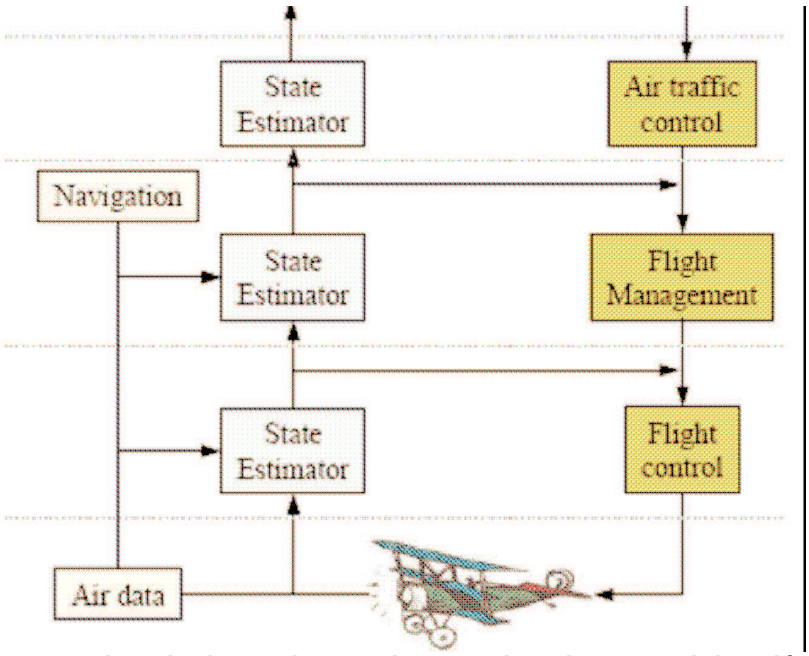


Figura 2.1. Ejemplo de un sistema de control en tiempo real de aviónica.

Los sistemas de tiempo real mixto o firme, son aquellos en los que si se pierde algún plazo de vez en cuando, hay una degradación del servicio que puede compensarse usando mecanismos de control de calidad, por ejemplo, en las aplicaciones de control industriales, como seria el caso de las cadenas de producción, el control de una cinta transportadora, etc.

Las aplicaciones de tiempo real estricto, necesitan tener una habilidad especial para coordinar un gran número de actividades en paralelo, pero de complejidad baja. Cada una de estas actividades esta implementada en forma de tarea, que la mayoría de veces será de ejecución cíclica o periódica, por ejemplo la evaluación del estado de un sensor cada cierto

tiempo. Sin embargo, existen algunas tareas que no pueden ser periódicas por su propia naturaleza sino que deberán ser aperiódicas, tareas que se ejecutan como respuesta a un evento externo a la aplicación por ejemplo alarmas y/o peticiones de un operador de consola, o como respuesta a un evento interno a la propia aplicación (acciones asíncronas requeridas por el propio programa).

En general, las tareas implementadas para estas aplicaciones pueden tener plazos temporales estrictos (hard deadlines) o no estrictos (soft deadlines). A las tareas aperiódicas con plazos temporales se las llama tareas esporádicas, siendo las tareas aperiódicas aquellas que no los tienen. Normalmente, las aplicaciones de tiempo real tendrán una mezcla de todo tipo de tareas. A menudo las tareas esporádicas son tratadas como tareas periódicas en el que el periodo viene fijado por el tiempo mínimo entre llegadas de estas tareas, de manera que estas tareas serán siempre aceptadas.

Realizando una simplificación importante, se puede decir que las tareas que ejecutan los sistemas de tiempo real tienen como características más importantes:

- Periodo de activación (T)
- Plazo temporal (D)
- Peor tiempo de ejecución (WCET)

El periodo de activación de una tarea es la separación temporal entre dos activaciones sucesivas de la tarea, el plazo temporal es el tiempo que tiene la tarea para emitir una respuesta una vez activada y el peor tiempo de ejecución es el máximo tiempo que tardará la tarea en realizar sus cálculos. Gráficamente, pueden verse representados estos tiempos en la Figura 2.2.

El periodo y el plazo temporal de las tareas vienen impuesto por las características del sistema físico que se quiera controlar y se determina en tiempo de diseño de la aplicación de tiempo real, sin embargo, el cálculo del peor tiempo de finalización es más complejo.

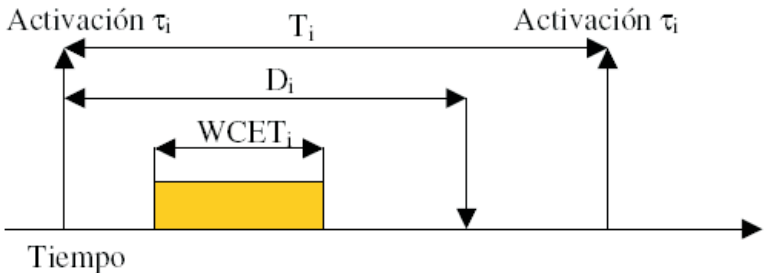


Figura 2.2. Características de la Tarea Ti.

En el artículo de P. Puschner y A. Burns [21] se encuentran las referencias básicas de cómo se puede realizar este cálculo. El WCET se calcula en el peor de los casos, por tanto, es una cota superior del tiempo máximo de ejecución de una determinada tarea. Esta cota temporal es dependiente de la arquitectura de la máquina donde se ejecuta, y no debe tener en cuenta las posibles interferencias entre las distintas tareas del sistema, tales como apropiaciones del procesador por una tarea de mayor prioridad, tiempos de bloqueo en el acceso a un recurso compartido, y demás tiempos que dependen del tipo de planificador que se use.

Los algoritmos de planificación son los encargados de decidir qué tarea se tiene que ejecutar en cada instante. Los planificadores pueden ser planificadores off-line en los que previo al funcionamiento del sistema se ha establecido un plan de ejecución de las tareas o planificadores on-line, en los que la decisión de qué tarea se ejecuta se realiza durante el funcionamiento del sistema.

En nuestro trabajo trabaremos con un planificador on-line. Los planificadores on-line establecen en tiempo de ejecución del sistema que tarea se debe ejecutar en cada momento, por tanto son más flexibles al entorno de la aplicación, pudiéndose adaptar a la carga real del sistema. Estos planificadores normalmente basan la elección de la tarea a ejecutar en distintos niveles de prioridades, eligiendo siempre la tarea con mayor prioridad. Pueden ser expulsivos, de manera que si se activa una tarea de mayor prioridad que la tarea que se está ejecutando, la nueva tarea se apodera del procesador, o no expulsivos [22]. La asignación de prioridades puede ser estática, fija para cada tarea durante toda la ejecución del sistema, o dinámica, la prioridad que se asigna a una tarea va variando en función del

estado de la tarea, del instante de activación o del estado del sistema. Los principales planificadores on-line con asignación estática de prioridades son: los planificadores expulsivos de prioridades fijas en los que la asignación de prioridades es RM (Rate Monotonic) o DM (Deadline Monotonic). Los principales planificadores con asignación dinámica de prioridades son el EDF (Earliest Deadline First) y el LLF (Least Laxity First). Concretamente nosotros vamos a trabajar con el planificador EDF.

El planificador Earliest Deadline First [23] asigna las prioridades a las tareas dinámicamente en función de sus plazos temporales y del estado del sistema. De manera que la tarea con un plazo temporal absoluto más inmediato será la tarea que tenga mayor prioridad. Es decir, ejecuta siempre la tarea más urgente independientemente del tiempo de cálculo de esa tarea. En el momento de activación de una tarea, deben recalcularse las prioridades para dar cabida a la nueva tarea. Si esta nueva tarea tiene su plazo temporal más inmediato pasara a ser la tarea que se ejecute adueñándose en este momento del procesador. En el caso de la asignación dinámica de prioridades, la prueba de planificabilidad suficiente y necesaria es la que viene dada por la ecuación $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ que indica que en EDF siempre que la utilización del procesador sea menor o igual que 1 el conjunto de tareas es planificable con los plazos temporales de las tareas igual a sus periodos.

Capítulo 3

Trabajo relacionado

Hay tres modelos principales que se utilizan para implementar las habilidades de los procesadores multihilo comerciales actuales: grano fino, grano grueso y simultáneo [24]. Por ejemplo El procesador de Intel Montecito [25] tiene un procesador de grano grueso mientras que la arquitectura del multicore Niagara de Sun presenta un grano fino por cada core [26]. En el mercado también existen diversos procesadores simultáneos [27, 28].

A pesar de que, generalmente, los procesadores de grano fino y los simultáneos ofrecen mejores prestaciones porque permiten compartir dinámicamente los recursos del pipeline de ciclo en ciclo, el cálculo del WCET de las tareas de tiempo real compartiendo el procesador en estos modelos se hace muy complicado. Debido a esto, en este trabajo se ha asumido un procesador de grano grueso, el cual cambia el uso del procesador entre los hilos durante los eventos de larga latencia de la misma manera que hacen los sistemas operativos multitarea clásicos.

Muchos han sido los autores que han investigado el consumo energético en sistemas de tiempo real uniprocesador. La mayoría de ellos se centran en sistemas de tareas periódicas como por ejemplo [19, 29], pero también se han publicado propuestas donde el sistema está compuestos por tareas aperiódicas [30] y esporádicas [31].

En las plataformas multiprocesador podemos aplicar dos categorías principales de plani-

ficación: la planificación global y la planificación particionada. En la planificación global, una tarea se sitúa en una cola de tareas ejecutables que es compartida por todos los procesadores del sistema y en cada momento, las tareas con mayor prioridad son seleccionadas para ejecutarse en los procesadores disponibles. En ese modelo, está permitida la migración de tareas entre los procesadores. En la planificación particionada, una tarea se asigna de forma permanente a un procesador dado y no tiene permitido migrar a otro. Esta técnica divide el problema de la planificación en una plataforma multiprocesador a varias plataformas uniprocesador. Debido a esto, muchos de los algoritmos utilizados en teoría uniprocesador (como por ejemplo EDF y RM) pueden ser adoptados.

En un artículo de Buttazzo y otros, [19], se presenta un algoritmo para la gestión de la energía que está basado en DVS y que integra un planificador elástico para procesador de niveles energéticos discretos. Algunos estudios de investigación han abordado el problema de la gestión de la energía en multiprocesadores cuando se ejecutan tareas de tiempo real [32, 17, 18]. Por ejemplo en [32] se presenta el problema de la reducción de la energía para tareas periódicas de tiempo real en multiprocesadores simétricos utilizando la técnica del DVS. El algoritmo EDF analiza los efectos de las heurísticas de particionado. En [17], Al Enawy y otros, consideran el problema de la reducción de la energía para tareas reemplazables estrictas de tiempo real. Estas tareas son planificadas en una plataforma simétrica de multiprocesadores con capacidad DVS. Ellos adoptan una planificación particionada y asumen que las tareas tienen asignadas prioridades estáticas mediante el algoritmo RM. En [18], Baruah aborda el problema de la síntesis de sistemas de tareas de tiempo real en multiprocesadores idénticos usando un planificador global y el algoritmo EDF.

Algunos estudios se han centrado también en procesadores multihilo. En [7] Cazorla y otros, presentan una arquitectura donde un procesador SMT interactúa con el sistema operativo para mejorar las prestaciones. El sistema operativo especifica los hilos cuyo rendimiento es previsible (PPT) para que sean ejecutados en una frecuencia dada, mientras que los hilos que no son PPT sólo pueden ejecutar los recursos que no van a ser utilizados por

los hilos PPT. Sin embargo, el problema de garantizar las restricciones de tiempo real junto con el ahorro energético no esta resuelto. En [5] Rotenberg afirma que las restricciones de tiempo real rara vez son garantizadas por los procesadores SMT y propone una arquitectura superescalar virtual en-orden que permite la virtualización de un procesador superescalar en varios procesadores. Esta arquitectura incorpora un planificador estático que ejecuta las tareas periódicas. Comparan su planificador con un planificador EDF para multiprocesadores en términos de pérdida de deadlines. La planificabilidad de las tareas estrictas de tiempo real no está garantizada y la gestión de la energía no se aborda.

Con respecto al solapamiento entre el procesador y la memoria de las tareas, El-Haj-Mahmoud y otros, [33], proporcionan una técnica para la explotación de un procesador de grano grueso con el propósito de tolerar las latencias de memoria en los sistemas de tiempo real estrictos. En este trabajo, obtienen un test de planificación cerrado para determinar si un conjunto de tareas estrictas de tiempo real es planificable en un contexto de planificación Round-Robin-Ponderado en un procesador multihilo. Sin embargo, en este trabajo no se aprovechan las ventajas del solapamiento para hacer frente al ahorro energético.

Capítulo 4

Planificador con ahorro energético para tareas de tiempo real no estrictas

4.1. Conjunto de tareas y modelo del sistema

El conjunto de tareas propuesto para este primer trabajo está formado por un conjunto de m tareas no estrictas de tiempo real $T = \{T_1, ..., T_n\}$, concretamente tareas esporádicas, que son tratadas como tareas periódicas en las que el periodo viene fijado por el tiempo mínimo entre llegadas de estas tareas. Este tiempo se ha modelado como el parámetro λ. Las tareas son lanzadas a ejecución cada cierto tiempo λ, es decir, el tiempo entre llegadas de estas tareas es $1/\ \lambda$ unidades de tiempo.

El modelo de sistema propuesto para este trabajo está formado por una procesador multicore multihilo, un planificador de tareas y un cola de tareas preparadas que alimenta el planificador. La Figura 4.1 muestra el esquema del sistema. Por motivos de evaluación, para los experimentos se ha considerado que el procesador es bi-core, aunque este trabajo puede extenderse a procesadores con cualquier número de cores. El algoritmo DVS implementa tres niveles de frecuencia (100Mhz, 200Mhz y 400Mhz) y se ha asumido que es un DVS global ya que, como se ha visto en la introducción, si la carga está bien equilibrada

las prestaciones proporcionadas por un DVS global pueden ser igual de buenas que las de un DVS local.

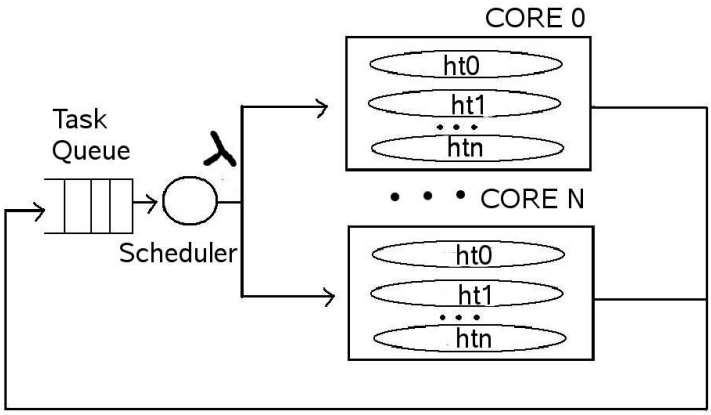


Figura 4.1. Cola de tareas y sistema multicore.

El sistema modelado trabaja de la siguiente manera: Cuando una tarea finaliza su ejecución, vuelve a la cola de tareas preparadas. El planificador lee esta cola y lanza las tareas preparadas al core correspondiente. Como hemos comentado anteriormente, el planificador debe esperar un tiempo λ para lanzar dos tareas consecutivas. Para analizar el efecto que este tiempo tiene sobre el consumo energético del sistema se han elegido diferentes valores de λ. Hemos asumido que el tiempo de llegada base λ es igual al tiempo de ejecución de la tarea mas lenta del conjunto de tareas a ejecutar en el sistema. Sin embargo, como la eficacia del procesador depende de este tiempo, se han analizado diferentes escenarios. Para ello, se multiplica el tiempo entre llegadas base (λ) por un factor dado al que llamaremos K cuyos valores serán 1, 1.5 y 2.

4.2. Planificador con ahorro energético

El algoritmo de planificación con ahorro energético propuesto para este modelo tiene como finalidad minimizar la energía consumida por el sistema siempre y cuando las restricciones de tiempo real se cumplan. Para ello se ha diseñado un modelo simple donde se asume que un core trabajando a la mínima frecuencia puede garantizar las restricciones de

tiempo real de una única tarea ejecutándose sobre él. Si en el core se ejecutan dos tareas, para garantizar la planificabilidad de las mismas, el sistema deberá trabajar a la frecuencia media y si el número de tareas en un mismo core es igual o mayor que tres, el sistema deberá trabajar a la máxima frecuencia.

```
Estado Inicial: Ambos cores se encuentran apagados

Llenando Cores: Se aplica cuando una tarea Ti se
                encuentra preparada

Paso 1: Alguno de los cores se encuentra vacio?
              Si: Si el core A no tiene tareas,
                  enviar Ti al core. En caso contrario,
                  enviarla al core B
              No: Ir al paso 2

Paso 2: Enviar Ti al core menos cargado,
              Si ambos cores se encuentran igual de
              cargados entonces incrementar la
              frecuencia y enviar Ti al core A

Reduciendo la frecuencia: se aplica cuando una tarea
              Ti termina

Paso 3: Ambos cores se encuentran igual de cargados?
              Si: Reducir la frecuencia
```

Cuadro 4.1. Algoritmo de planificación.

En el estado inicial, ambos cores están apagados, por lo tanto no consumen energía. Cuando una tarea llega al sistema, uno de los cores se enciende y empieza a trabajar a la mínima frecuencia. Si otra tarea llega al sistema, el planificador debe estimar si el core actual puede satisfacer las restricciones de tiempo real de ambas tareas (la que está ejecutándose y la tarea nueva que ha llegado al sistema) con la velocidad actual de trabajo. Si no es posible, se encenderá un segundo core que trabajará a la mínima frecuencia también. A partir de este punto, sino se pueden garantizar las restricciones de tiempo real de las

tareas que vayan llegando, el planificador incrementará la frecuencia del procesador hasta llegar a la máxima posible. Una vez que las tareas finalizan, se revierte este proceso. El planificador reduce la frecuencia, desde la máxima hasta la mínima, hasta que finalmente los cores son apagados. En la Tabla 4.1 se puede ver una descripción detallada del algoritmo de planificación.

4.3. Particionado de tareas

Como hemos comentado en el apartado 4.1, el sistema modelado no tiene un particionador que aplique heurísticas de distribución de las tareas explicítamente, ya que este primer modelo es un modelo muy sencillo. En este caso, la distribución de las tareas se lleva a cabo teniendo en cuenta el tiempo de ejecución de las mismas. Para entender el porqué de esta distribución veremos un ejemplo. Imaginemos que el planificador selecciona dos tareas con un tiempo de ejecución muy largo y las envía al core A mientras que otras dos tareas con tiempos de ejecución más cortos son enviadas al core B. Con esta distribución, cuando las tareas del core B (tareas cortas) terminen su ejecución, las tareas del core A (tareas largas) aún estarán ejecutándose. Recordemos que en este modelo se ha implementado un DVS global, es decir, la frecuencia de ambos cores tiene que ser la misma. Debido a que las tareas del core A están todavía en ejecución, la velocidad del procesador no puede ser reducida y se estaría gastando energía a pesar de que el core B está vacío. Por tanto, si las tareas se distribuyen de forma adecuada entre los cores (hay tareas largas en ambos cores), la velocidad del procesador podrá ser reducida tan pronto como estas tareas terminen su ejecución.

Este ejemplo ilustra dos estrategias de particionado opuestas. Nos referiremos a una política de particionado *equilibrada* a aquella política que envía tareas complementarias (cortas y largas) al mismo core y como política de particionado *desequilibrada* a aquella que no tiene en cuenta si las tareas son complementarias o no.

En la Tabla 4.2 se pueden ver los resultados obtenidos para una misma mezcla que ha sido ejecutada utilizando políticas de particionado *equilibradas* y *desequilibradas*. Las columnas F400, F200 y F100 de la tabla indican el porcentaje de tiempo que el procesador ha estado ejecutándose a 400Mhz, 200Mhz y 100Mhz respectivamente. Esta mezcla está compuesta por 6 benchmarks (véase el apartado 6.1) *decode, encode, mpeg, rawd, adpcm* y *crc*. En la política de particionado *equilibrada*, los benchmarks son distribuidos de manera equitativa entre los dos cores, es decir, en ambos cores podemos encontrar benchmarks con tiempos de ejecución largos y cortos. Por el contrario, en la política de particionado *desequilibrada*, los benchmarks con tiempos de ejecución más largos son enviados al mismo core. Como consecuencia de esto, el procesador está la mayoría del tiempo ejecutándose a frecuencia máxima ($F400 > 90\,\%$) ya que el planificador no puede bajar la frecuencia. Como puede observarse el tiempo en el que el procesador está a la máxima frecuencia está fuertemente relacionado con el tiempo entre llegadas (Factor K comentado en el apartado 4).

Cuadro 4.2. Utilizando diferentes estrategias de balanceo.

Política de particionado	*K*	Tiempo de ejecución (M)	Tiempo de memoria (%)	F400 (%)	F200 (%)	F100 (%)
Equilibrado	1	59.10	0.71	0.94	0.05	0.01
	1.5	71.67	0.57	0.60	0.37	0.03
	2	83.03	0.49	0.40	0.58	0.02
	2.5	88.39	0.47	0.33	0.63	0.04
	3	90.93	0.45	0.30	0.66	0.04
Desequilibrado	1	56.44	0.69	0.97	0.01	0.02
	1.5	55.42	0.62	0.95	0.03	0.02
	2	61.21	0.57	0.93	0.03	0.04
	2.5	64.24	0.57	0.91	0.04	0.05
	3	67.45	0.40	0.90	0.05	0.05

Capítulo 5

Planificador con ahorro energético para tareas estrictas de tiempo real

5.1. Conjunto de tareas y modelo del sistema

El modelo de tareas para este sistema consiste en un conjunto de n tareas periódicas estrictas de tiempo real, $T = \{T_1, ..., T_n\}$, ejecutadas en un procesador con m cores multihilo, $H = \{H_1, ..., H_m\}$. Cada tarea de tiempo real T_i, como hemos visto en el Capítulo 2, está caracterizada por tres parámetros principales que son $T_i = \{WCET_i, P_i, D_i\}$.

El WCET es un prerrequisito de las aplicaciones de tiempo real [31] y tiene que ser conocido a priori. Para ello puede ser medido o estimado de una forma analítica. En nuestro caso, el $WCET_i$ ha sido obtenido ejecutando la $tarea_i$ a la mínima frecuencia posible del procesador. Como uno de los objetivos de este modelo de planificador es explotar el solapamiento entre el procesador y la memoria para reducir la energía consumida por el conjunto de tareas, el WCET se ha dividido en dos componentes atendiendo a los requisitos de memoria y CPU que consume la tarea. Con esta división la caracterización de una tarea quedará de la siguiente manera: $T_i = WCET_i(cpu_i, m_i), P_i, D_i$. Por último mencionaremos que para simplificar el modelo de tareas, de la misma manera que muchas

implementaciones del algoritmo EDF [32], en este trabajo se ha asumido que ($P_i = D_i$).

Las tareas llegan al sistema de forma dinámica desde la fuente — el número de tareas varia de acuerdo con los cambios de modo de las aplicaciones de tiempo real [34]— , se ejecutan un número de periodos (a los que llamaremos periodos activos) y salen del sistema dirigiéndose al sumidero.

El sistema modelado está compuesto por un procesador multicore multihilo de grano grueso, un particionador, un planificador con ahorro energético (compuesto por planificadores EDF y un DVS hardware) y la fuente y el sumidero de las tareas de tiempo real. Este sistema está representado en la Figura 5.1. Cada uno de los cores es un procesador multihilo de grano grueso. Al igual que en el primer modelo, este sistema se puede generalizar a m cores, pero por motivos de evaluación se han asumido dos.

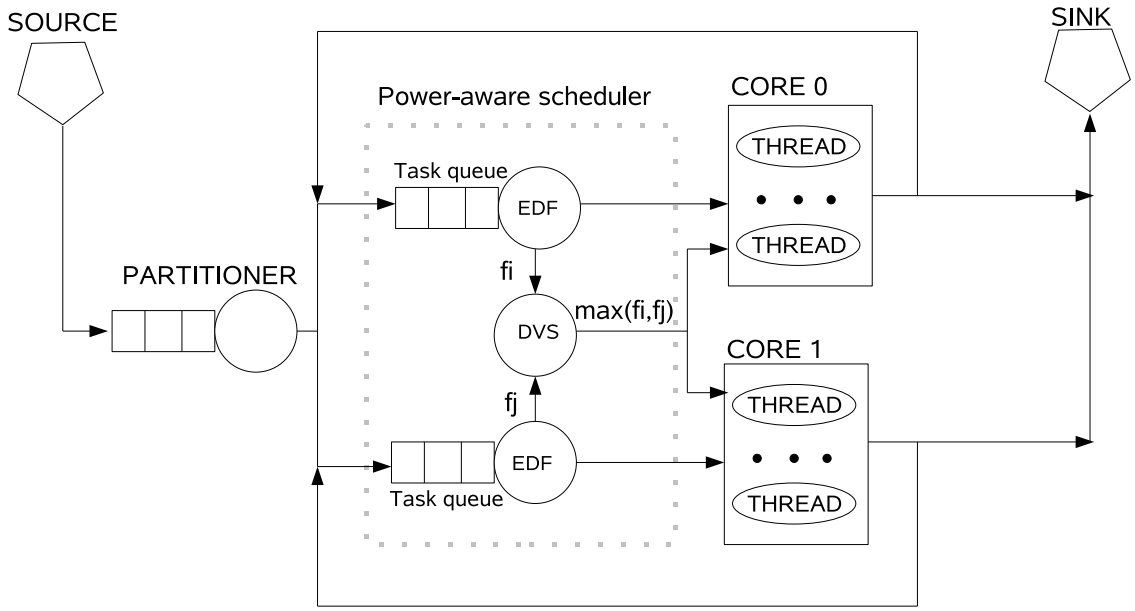


Figura 5.1. Modelo del sistema.

Ambos cores modelan un procesador basado en un ARM11 que implementa 3 hilos hardware, los cuales comparten un regulador DVS global. El particionador distribuye las las tareas preparadas para ejecución, de acuerdo a las heurísticas de particionado, entre los planificadores EDF intentando mejorar la viabilidad del sistema y reduciendo el consumo energético mediante el equilibrado de la carga. El planificador EDF ejecuta las tareas

de su cola de preparados de manera ordenada, es decir, la tarea con una mayor prioridad será la primera en ser ejecutada. Cada planificador trabaja de forma independiente y sólo el particionador se preocupa de la distribución de tareas. De esta manera el proceso de planificación se simplifica significativamente comparado con un planificador global. Cada uno de los planificadores escribe en un puerto de entrada/salida que es leído por el DVS. Una vez leído el puerto de entrada/salida, el hardware DVS ajusta la frecuencia del sistema a los requisitos de los planificadores.

```
Particionado de tareas:
(a) Conjunto inicial de tareas: distribuido de acuerdo a una
    heuristica
(b) Nuevas tareas que llegan: son asignadas al core con menor
    utilizacion

Planificacion (por core):
    Comportamiento EDF:
(c)           -ordena la cola
(d)           -lanza las tareas y aplica los reemplazos
    Comportamiento de ahorro de consumo:
(e)           -selecciona la menor frecuencia Fi
               cuando cambia el conjunto de tareas

Comportamiento DVS:
(f) Selecciona la maxima frecuencia Fi/Vi y la suministra a
    los cores
```

Figura 5.2. Principales tareas del sistema.

5.2. Planificador con ahorro energético

La Figura 5.2 resume las funciones básicas que lleva a cabo el sistema. Inicialmente el particionador distribuye el conjunto inicial de tareas (etiqueta a) entre los cores (es decir, en su cola de tareas) de acuerdo a una heurística dada (descritas en la Sección 5.3).

El comportamiento del algoritmo EDF se divide en dos funciones principales. Primero, la cola de tareas de cada core siempre mantiene las tareas ordenadas de acuerdo al algoritmo

EDF (etiqueta c), es decir, las tareas se insertan en el lugar adecuado cuando llegan a la cola. Segundo, una tarea preparada puede ejecutarse tan pronto como alcanza la cabeza de la cola, siempre y cuando haya algún hilo de ejecución libre en el core (etiqueta d). Si no hubiera ninguno libre y la tarea que está preparada es más prioritaria que alguna de las tareas que se están ejecutando, esta última sería reemplazada.

Atendiendo a las características de ahorro energético del sistema, se distribuyen en dos componentes principales, el planificador y la hardware DVS. El primero selecciona la mínima frecuencia Fi a la que el sistema es planificable (etiqueta e) y envía este valor al hardware DVS. El hardware DVS selecciona el máximo valor recibido (de entre los dos valores enviados por los planificadores) y lo envía (frecuencia y voltaje) a ambos cores (etiqueta f).

El planificador selecciona la frecuencia más baja a la que el sistema es planificable cada vez que el conjunto de tareas que se están ejecutando en el sistema cambia, es decir, cada vez que llega una nueva tarea o una de las tareas que se estaban ejecutando termina. Esto se hace así por dos razones: i) Dado que la carga del sistema ha cambiado puede que se necesite una nueva frecuencia (más alta o más baja) y ii) puede suceder que debido a un buen solapamiento, las tareas que continúan ejecutándose en el core tomen un tiempo extra (*slack time*) que permita al procesador garantizar las restricciones de tiempo real trabajando con una frecuencia más baja. Cabe mencionar que el número de veces que el planificador envía una frecuencia nueva es despreciable comparado con el tiempo total de ejecución (por ejemplo 8 veces durante 30M de ciclos). Debido a esto, se ha asumido que la sobrecarga producida por los cambios de frecuencia es nula.

Por último cuando una nueva tarea llega al sistema, el particionador la coloca en el core con menos utilización (etiqueta b). Esta política de distribución se llama Worst Fit [17] y lo que se pretende con ella es equilibrar la carga del sistema. Como consecuencia, puede darse el caso en que una nueva tarea llegue al sistema y no se necesite subir la frecuencia para garantizar las restricciones de tiempo real.

5.3. Heurísticas de particionado

Como hemos comentado antes, los procesadores de grano grueso cambian el procesador entre los hilos de ejecución cuando ocurre un evento de alta latencia, como por ejemplo un fallo de cache, en ese caso, la actividad de memoria de un hilo determinado puede solaparse con el tiempo de procesador de otro hilo del mismo core permitiendo al planificador conseguir un *slack time*. Por eso, cuanto más alto sea el tiempo de solapamiento entre las tareas, más altos pueden llegar a ser los beneficios obtenidos en cuanto al consumo energético. Nótese que se puede alcanzar un elevado ratio de solapamiento cuando las tareas ejecutadas en los hilos de un mismo core son complementarias en términos de requisitos de procesador y memoria. Por ejemplo una de ellas requiere una gran actividad de procesador mientras que la otra requiere una gran actividad de memoria. Por el contrario, si las tareas de un mismo core requieren en su mayoría una gran actividad o de procesador o de memoria no podrá aparecer solapamiento entre ellas.

Para evaluar el ahorro de consumo proporcionado por el planificador de ahorro energético, se han diseñado dos heurísticas de particionado, llamadas BM (Memoria equilibrada) y BC (CPU equilibrada). Estas heurísticas distribuyen las tareas de acuerdo a las actividades de la memoria y el procesador que requieren de una forma equilibrada (es decir, lo más cercano posible a cincuenta-cincuenta) entre ambos cores. De esta manera, ambas heurísticas actúan como catalizadores para incrementar el solapamiento entre el procesador y la memoria. Además para evaluar las prestaciones que producen las heurísticas, se ha implementado una heurística desquilibrada llamada UR (Recursos Desequilibrados).

Capítulo 6

Entorno experimental

Ambos planificadores han sido evaluados sobre el entorno de simulación ciclo a ciclo de multicore multhilo Multi2Sim [35], el cual ha sido extendido para dar soporte a tareas de tiempo real tanto estrictas como no estrictas. Cada uno de los cores modela un microprocesador empotrado ARM 11 [3]. La Tabla 6.1 resume los parámetros de la arquitectura.

Cuadro 6.1. Parámetros de la máquina.

Microprocessor core	
Política de issue	En orden
Tipo de fetch	Switch on event
Predicción de saltos	Dos-niveles histórico global 256 entradas BTB, 4096 2-bit contadores GHB 13 ciclos de penalización
Ancho de banda de issue	2 instrucciones/ciclo
# de ALU's enteros, multiplicadores/divisores	2,1
# de ALU's flotantes, multiplicadores/divisores	2,1
Jerarquía de memoria	
Memoria cache	Desabilitada
Latencia de memoria	100 cycles

6.1. Diseño y caracterización de la carga

Para realizar este trabajo se han utilizado un conjunto de benchmarks de las suites EEMBC [36], Clab, Mediabench [37] y del repositorio Real-Time Systems Benchmark [38]. Con estos benchmarks se ha diseñado un conjunto de mezclas para explorar de forma experimental los posibles ahorros energéticos. Para ello se ha seguido una metodología formada por 4 pasos principales:

1. Caracterización de los benchmarks.
2. Clasificación de los benchmarks.
3. Seleccionar los benchmarks que formarán una mezcla.
4. Planificación de las mezclas (sólo para tiempo real estricto).

La tabla 6.2 resume los benchmarks utilizados para los experimentos y muestra para cada uno de ellos los requisitos de procesador y de memoria asociados. Los requisitos de procesador han sido medidos en millones (M) de ciclos mientras que los requisitos de memoria han sido cuantificados como el porcentaje de instrucciones de memoria (loads) ejecutadas. Como puede observarse, los requisitos de memoria y procesador difieren ampliamente entre los benchmarks. Debido a estas diferencias, los benchmarks han sido clasificados (*columna clas.*) ateniendo al consumo que realizan. Para cada recurso (procesador y memoria) los benchmarks han sido clasificados en tres grupos: bajo (L), alto (H) y medio (M), dependiendo si tienen mucho, poco o un consumo intermedio de recursos respectivamente.

Una vez que los benchmarks han sido clasificados, se pueden diseñar mezclas heterogéneas, formadas por distintos tipos de benchmarks, seleccionando benchmarks de diferentes grupos.

Para los experimentos del primer modelo (Capítulo 4), las mezclas diseñadas pueden verse en la tabla 6.3. Dado que en este modelo no existía un particionador explícito, el orden en que las tareas llegan al sistema es importante para la distribución de las mismas. De

Cuadro 6.2. Descripción de los benchmarks, recusos y clasificación

Nombre Suite	Nombre Benchmarks	Descripción	Procesador		Memoria	
			(M)	Clas.	(%)	Clas.
MediaBench	Adpcm	Compresión y descompresión del habla	10.1	H	21	L
	Mpeg	Decodificador de video compresión Lossymotion	18.9	H	22	L
	G721	Codificador de compression de voz	21.8	H	6.4	L
	Rawc	Algoritmo de compresión del habla	9.4	H	26	M
CLab	Cnt	Cuenta números en una matriz	0.43	M	22	L
WCET	Compress	Programa de compresión de datos	0.41	M	26	M
	Cover	Programa de testeo de caminos	0.32	M	22	L
	Duff	Copia un array de 43 bits	0.28	L	24	L
	Expint	Expansión de series para funciones integrales	0.35	M	24	L
	Ludcmp	Algoritmo de descomposición LU	0.39	M	28	M
	Insertsort	Inserción ordenada en un array	0.28	L	24	L
	Ns	Búsqueda en un array multidimensional	0.67	M	22	L
	Nsichneu	Simulación y extension de una red Petri	0.41	M	33	H
	Statemate	Código generado automáticamente	0.27	L	29	H
EEMBC	Fac	Factorial de un número	0.25	L	22	L
	Fibcall	Cálculo iterativo de un número Fibonacci	0.25	L	27	M
	Bs	Búsqueda binaria en un array	0.24	L	22	L
	Prime	Cálculo de números primarios	0.60	M	24	L
	Ndes	Código empotrado complejo	2.11	H	29	H
	Fir	Filtro de impulsos de respuesta finitos	0.34	M	24	M
	CRC	Comprueba la redundancia cíclica en un array de 40 bits	1.01	H	22	L
	FDCT	Transformada discreta del coseno	0.32	M	25	M
	FFT1	Transformada de Furier de 1024-puntos	0.34	M	25	M

la misma manera que el tiempo que se consume ejecutándose en cada una de las frecuencias depende de la mezcla que vaya a ejecutarse, el ahorro energético muestra la misma dependencia. Estas mezclas se han diseñado para explorar diferentes escenarios sobre las ganancias que el algoritmo de planificación puede proporcionar. Todos estas mezclas han sido diseñadas de acuerdo a la política de particionado *equilibrada* ya que en todas ellas las tareas de mayor duración están distribuidas entre ambos cores.

La Tabla 6.4 muestra los experimentos diseñados para evaluar el planificador para tareas de tiempo real estrictas. Estas mezclas se han diseñado de manera que la utilización de los

Cuadro 6.3. Mezclas para el modelo de tareas no estrictas

Nombre	Nombre de los benchamrks
M0	crc, ludcmp, adpcm, fir, fdct, fft1
M1	fft1, crc, fir, ludcmp, adpcm, fdct
M2	crc, fft1, fir, adpcm, ludcmp, fdct
M3	fir, ludcmp, crc, fdct, fft1, adpcm
M4	crc, adpcm, fdct, ludcmp, fir, fft1
M5	ludcmp, adpcm, fdct, fft1, crc, fir
M6	ludcmp, crc, fir, fdct, fft1, adpcm
M7	fft1, fir, adpcm, ludcmp, crc, fdct

cores fluctúe entre un 40 % y un 80 %. Estas mezclas son distribuidas entre los cores por el particionador de acuerdo a una heurística de planificación de las vistas anteriormente (Apartado 5.3). Cada heurística proporciona una distribución diferente de los recursos de memoria y CPU para cada uno de los cores como muestra la Tabla 6.5. En esta tabla, los valores se presentan como un par en cada una de las celdas de la tabla. El primer valor de cada par representa el porcentaje de los requisitos de memoria para un core dado con respecto al total de los requisitos de memoria para ambos cores. El segundo valor se refiere a los requisitos de procesador. Por ejemplo, los pares (50,61), (50,39), que se corresponden a la heurística BM para la Mezcla 4, quieren decir que la actividad de memoria está distribuida de forma equitativa entre ambos cores mientras que la actividad de procesador es aproximadamente un 20 % más en el core 1 que en el core 0 (61 vs. 39).

Cuadro 6.4. Mezclas para el modelo de tareas estrictas

Nombre	Nombre de los Benchmarks
Mix 1	Ndes, Statemate, Duff, CRC, Ludcmp, Nsichneu
Mix 2	Compress, Cnt, Fibcall, Fac, Fir and Bs
Mix 3	Duff, Nsichneu, FDCT, Expint, Fibcall, Cover
Mix 4	Prime, CRC, FFT, Bsort, Bs and Insertsort
Mix 5	Cover, Ns, Nsichneu, Ndes, Cnt, Statemate

Por último, como paso final, se ha llevado a cabo un planning de la ejecución de cada uno de los mixes. En las aplicaciones de control críticas de tiempo real, las tareas no

Cuadro 6.5. Distribución de los requisitos de memoria y CPU para cada una de las heurísticas de particionado (%Mem, %CPU).

Heurística	Mix 1 Core 0 , Core 1	Mix 2 Core 0 , Core 1	Mix 3 Core 0 , Core 1	Mix 4 Core 0 , Core 1	Mix 5 Core 0 , Core 1
BM	(53,48),(47,52)	(44,43),(56,57)	(52,39),(48,61)	(50,61),(50,39)	(49,58),(51,42)
BC	(59,51),(41,49)	(44,43),(56,57)	(48,60),(52,40)	(30,47),(70,53)	(43,52),(57,48)
UR	(65,66),(35,39)	(62,68),(38,32)	(66,67),(34,33)	(77,59),(23,41)	(78,78),(22,22)

sólo se ejecutan contínuamente en el sistema si no que puede haber tareas que entren y salgan dinámicamente del sistema (cambios de modo) y esta es la razón por la que se ha de realizar este planning de ejecución. Además para evaluar estos sistemas, no necesitamos simplemente seleccionar los benchmarks que pertenecen a cada una de las mezclas, también se ha de diseñar como se van a comportar estos benchmarks a lo largo del hiperperiodo (mínimo común múltiplo de los periodos de los benchmarks que forman una mezcla). La distribución de los periodos activos de una mezcla ha sido diseñada de forma aleatoria para introducir variaciones de la carga en el sistema y poder aplicar el algoritmo de ahorro energético. La Figura 6.1 muestra los diagramas de Gannt correspondientes a las cinco mezclas de este modelo durante sus correspondientes hiperperiodos. En estos diagramas se puede apreciar la relación entre el periodo (activo o no) de cada una de las tareas, el número de veces que se ha ejecutado y la relación entre periodo e hiperperiodo. Para evitar tiempos de ejecución innecesariamente elevados, los valores de los periodos han sido seleccionados como divisores relatívamente pequeños del hiperperiodo. Por ejemplo, la Mezcla 4 está compuesta por seis benchmarks que llegan al sistema al mismo tiempo y cuyo hiperperiodo es 50 millones de ciclos. Los benchmarks *BS, Insertsort, CRC,* y *FFT* permanecen en el sistema durante todo el hiperperiodo mientras que *Prime* y *Bsort* permanecen en el activos en el sistema durante un número de periodos (en este caso 1 y 5 respectivamente) y después salen del sistema para llegar al sumidero.

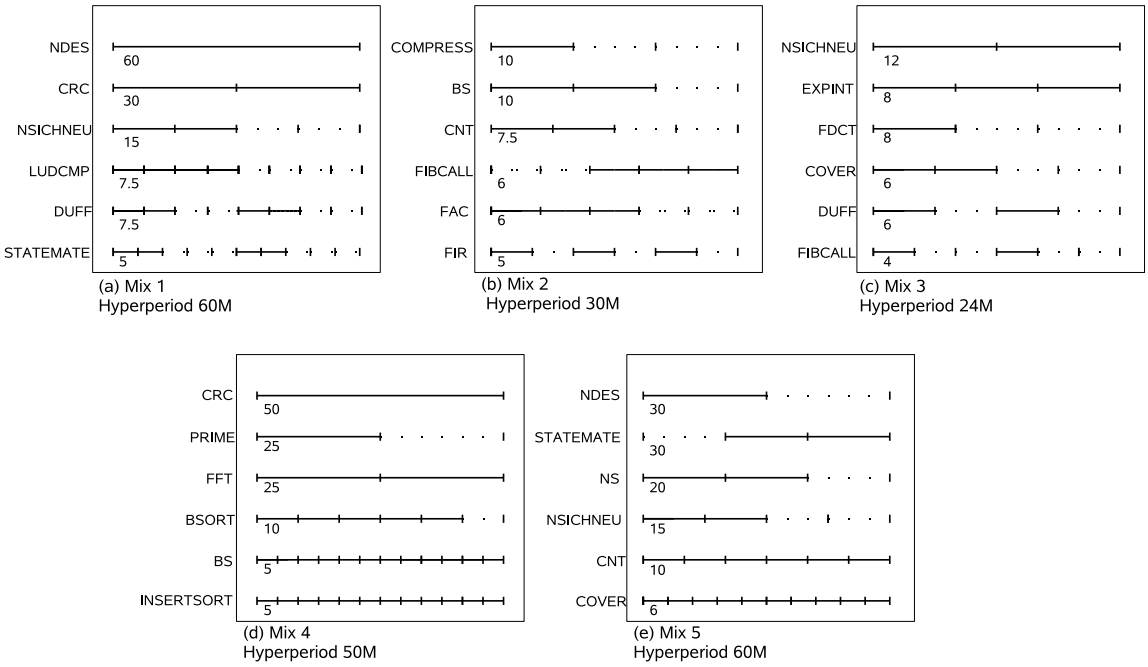


Figura 6.1. Diagramas de Gannt para las mezclas diseñadas. Las líneas continuas significan que el periodo está activo. Las líneas discontinuas que la tarea está fuera del sistema (periodo inactivo).

Capítulo 7

Resultado Experimentales

7.1. Modelo de tareas de tiempo real no estrictas

Como hemos comentado en el apartado 4.1, en este modelo se ha asumido que el procesador puede trabajar con tres niveles de energía. Con respecto a la relación entre la frecuencia y el voltaje del procesador, cada uno de estos niveles de energía utiliza diferentes niveles de voltaje. La Tabla 7.1 muestra las asunciones sobre la energía que se consume por ciclo cuando trabajamos a 400MHz, 200MHz y 100MHz respectivamente. Estos valores han sido elegidos de acuerdo con los valores del procesador Pentium M [39, 40].

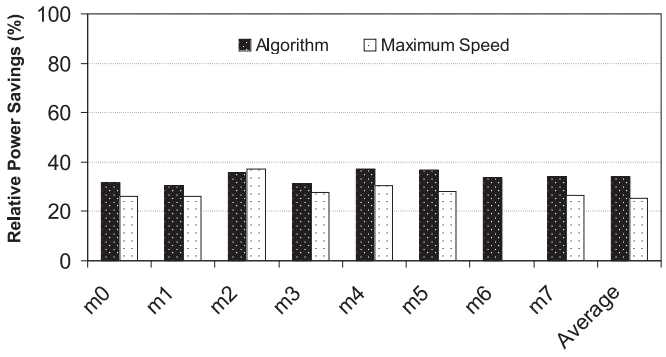
Cuadro 7.1. Energía consumida por frecuencia.

frequency [MHz]	500	400	300	200	100
energy [pJ/cycle]	450	349.2	261.5	186.3	123.8

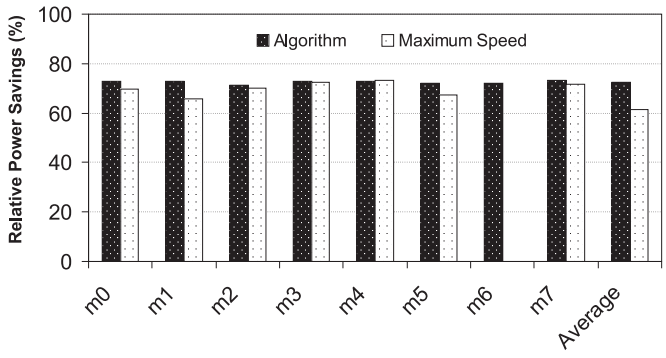
Los resultados de energía han sido obtenidos ejecutando simulaciones de 3 millones de ciclos de duración durante las cuales las mezclas se ejecutaban repetidamente. Para cada una de las ejecuciones, se ha medido el número de ciclos en los que el sistema trabaja a una frecuencia dada y se ha multiplicado este valor por la energía que consume por ciclo esa frecuencia.

Para poder comparar el ahorro energético de cada factor, se ha considerado como unidad

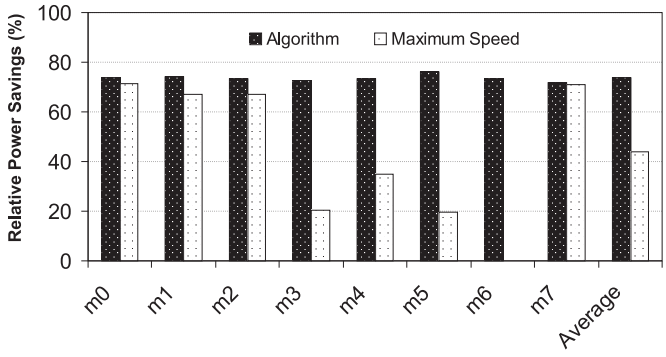
de medida base la mezcla que más consume (es decir, M6) cuando se ejecuta el procesador a máxima frecuencia. A partir de ahí se han obtenido los resultados de energía relativos para las demás mezclas. La Figura 7.1 muestra los resultados.



(a) Factor 1



(b) Factor 1.5



(c) Factor 2

Figura 7.1. Consumo Relativo.

Tres son las principales conclusiones a las que podemos llegar observando los resultados

obtenidos:

- Si nos fijamos en el algoritmo propuesto, generalmente, el ahorro energético es mayor a medida que el tiempo entre llegadas es mayor (es decir, los periodos son más largos). Esto se debe a que la utilizacion del sistema es menor, y por tanto el algoritmo se aplica más veces. Por ejemplo, si el sistema estuviera sobrecargado la mayoría del tiempo, el algoritmo aportaría menos beneficios.

- Independientemente del tiempo entre tareas (factor 1, 1.5 y 2), la política de particionado por si sola permite un ahorro del consumo. Los resultados muestran que sólo variando la política de particionado, el ahorro energético proporcionado, de media, puede alcanzar valores sobre un 25 %, 61 %, y 44 %, para el factor 1, factor 1.5 y factor 2, respectivamente.

- Los beneficios del algoritmo no sólo dependen del algoritmo en si, si no también del particionado base, es decir, puede darse el caso, de que aplicando el algoritmo con una mala partición inidial de las tareas, el consumo energético se vea incrementado. Sin embargo, combinando ambos, el algoritmo y un buen particionador el algoritmo siempre conlleva importantes beneficios. Los resultados muestran que, de media, estos beneficios han sido 34 %, 73 %, y 74 %, para el factor 1, factor 1.5 y factor 2, respectivamente.

7.2. Modelo de tareas de tiempo real estrictas

Para este modelo, el planificador con ahorro energético puede trabajar para cualquier rango de niveles de frecuencia. Los beneficios energéticos dependerán, por tanto, de cómo de amplio sea este rango y del número de niveles que soporte el regulador de DVS. En este trabajo se han evaluado diferentes modelos de frecuencia y voltaje. Concretamente se han evaluado tres modelos los cuales soportan 2, 3 y 5 niveles de frecuencia y voltaje a los

que llamaremos 2L, 3L y 5L respectivamente. El modelo 2L es un modelo que trabaja sólo con los niveles de frecuencia extremos (100MHz y 500MHz) mientras que el modelo 3L también soporta una frecuencia intermedia (300MHz). Por último el modelo 5L permite trabajar con las cinco frecuencias. Los valores de energía como hemos visto en el modelo anterior han sido seleccionados asemejándose a los del procesador Pentium M [40]. La Tabla 7.1 muestra los valores mencionados.

Por motivos de comparación, se han evaluado dos modelos más. Un modelo base (1L) y un modelo planificador simple de dos niveles de frecuencia (2LN). El modelo base asume que el procesador siempre trabaja a la máxima frecuencia. El modelo 2LN asume que el procesador siempre trabaja a la máxima frecuencia excepto cuando no hay ninguna tarea ejecutándose en los cores (están vacíos). En ese instante la frecuencia es reducida a la mínima. Cuando una tarea vuelve a empezar su ejecución o llega una nueva tarea al sistema, la velocidad del sistema vuelve a ser la que teníamos antes de bajarla a la mínima. Hemos de mencionar que esta mejora se aplica a todos los demás modelos diseñados.

La energía consumida se ha obtenido ejecutando cada mezcla durante su hiperperiodo. De la misma manera que para el modelo de tareas no estrictas, para cada caso, se mide el tiempo que el procesador esta trabajando a cada frecuencia y la energía acumulada se obtiene multiplicando estos valores por la energía que consume cada nivel de frecuencia por ciclo.

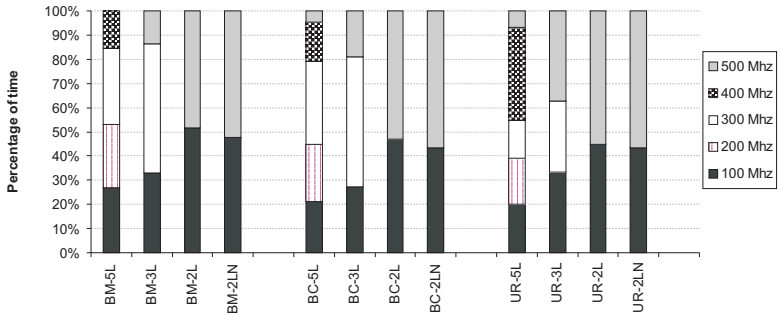


Figura 7.2. Porcentaje del tiempo de ejecución para cada heurística.

La Figura 7.2 muestra el porcentaje de tiempo que cada una de las heurísticas consume,

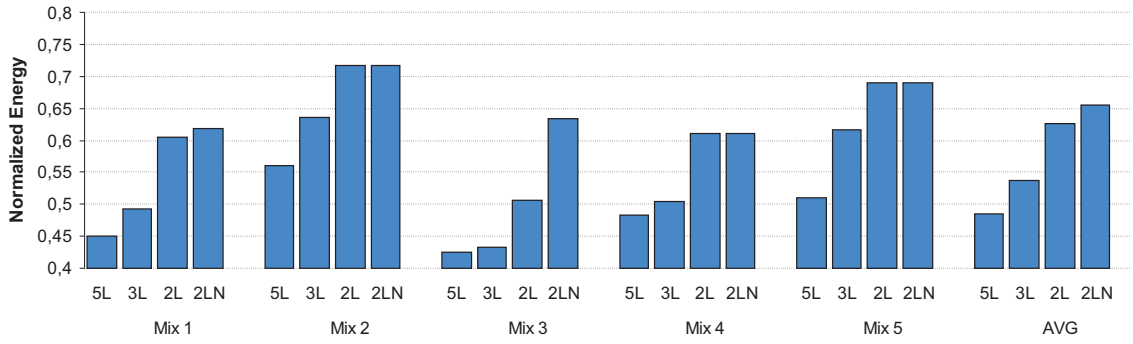
de media, en cada uno de los niveles de frecuencia dependiendo del sistema modelado (5L, 3L, 2L y 2LN). Como puede observarse, el planificador con ahorro energético propuesto, cuando se aplican diferentes niveles permite a ambos cores trabajar a lo largo de todo el rango de frecuencias, lo que demuestra la bondad de las mezclas diseñadas. Obsérvese que el planificador con ahorro energético, independientemente del modelo del sistema, sólo selecciona un nivel de frecuencia y voltaje i si las restricciones de tiempo real de la tarea no pueden ser satisfechas con un nivel inferior j ($j < i$). Este hecho puede observarse en la figura, donde el tiempo durante el qué se trabaja a frecuencia máxima es menor en el modelo 5L que en el modelo 3L, el cual es menor que en el modelo 2L. Además, puede observarse como en algunos casos (BM-5L) la frecuencia máxima nunca se alcanza.

La observación previa es importante en relación con el consumo energético ya que la energía normalizada para una tecnología dada (por ejemplo 70 nm) crece exponencialmente con la frecuencia de reloj [41]. Por eso, el planificador debe concentrar el tiempo de ejecución en la frecuencia más eficiente como hace la presente propuesta. De esta manera, el consumo de energía se reduce reduciendo el tiempo consumido a la máxima frecuencia.

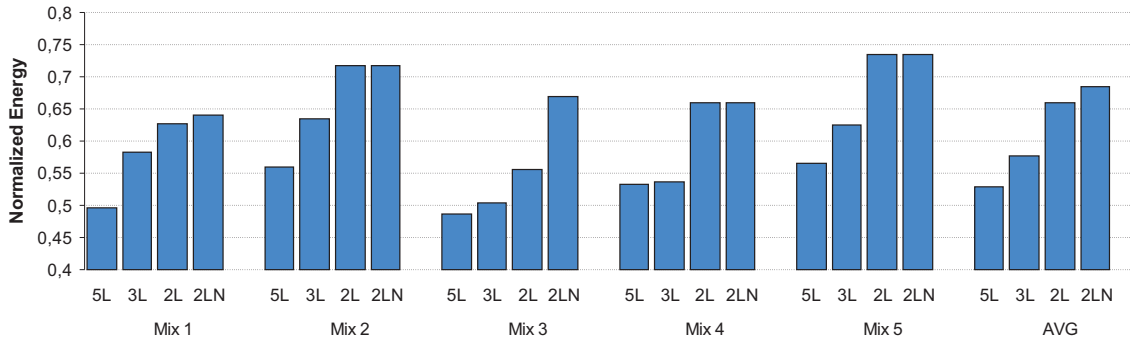
La Figura 7.3 muestra los resultado de la energía consumida relativa al modelo base. Es decir, los valores de energía han sido normalizados de acuerdo a un procesador trabajando a la máxima frecuencia (lo que representaría un valor de energía de 1). Los beneficios energéticos pueden deducirse de manera directa. Por ejemplo, un valor de 0.45 significa que para una determinada heurística se necesita un 45 % de la energía que se necesitaría para el modelo base, asimismo la heurística alcanza un ahorro energético del 55 % ($1 - 0{,}45$).

Tres conclusiones principales se pueden extraer observando la figura.

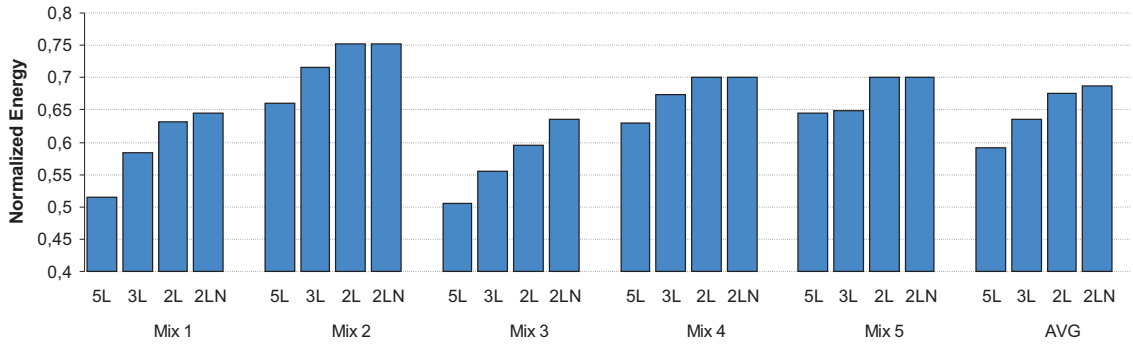
- Atendiendo a las heurísticas de particionado, la heurística cuyo consumo normalizado es menor, de media, es la heurística BM (Figura 7.3(a)). Esta heurística presenta sobre un 10 % ($0{,}59 - 0{,}49$) menos de consumo que la heurística desequilibrada para el modelo 5L lo que supone sobre un 21 % de ahorro energético. Esto se debe a que el tiempo de memoria es significativamente mayor que el tiempo de procesador



(a) Balanced Memory



(b) Balanced CPU



(c) Unbalanced Resource

Figura 7.3. Normalized Energy.

(unas cuatro veces mayor de media). Debido a esto, el tiempo de solapamiento aumenta cuando el tiempo de memoria está equilibrado, permitiendo de esta manera al planificador reducir la frecuencia más veces que la heurística BC.

- Sin tener en cuenta la heurística aplicada, los mejores resultados son los resultados

obtenidos cuando trabajamos con un modelo con un mayor número de niveles de frecuencia y voltaje. Esto es debido a que cuando el sistema tiene un mayor rango de frecuencias donde elegir, existen más oportunidades de encontrar una frecuencia que se ajuste mejor a los requisitos del sistema. Por ejemplo si un conjunto de tareas necesita al menos una frecuencia de 350MHz, el modelo 3L seleccionará la frecuencia de 500MHz para poder garantizar la planificabilidad mientras que el modelo 5L seleccionará la frecuencia de 400MHz. Por eso, este último podrá ahorrar más energía. Los resultados muestran que si trabajamos con un modelo 5L los beneficios energéticos son, de media, sobre un 32 % mayores que frente a un modelo 2L (ambos trabajando con la heurística BM).

Las diferencias de energía entre el modelo 2L y el modelo 2LN no son muy significativas (de media un 4 % mejores usando el modelo 2L con la heurística BM). La razón es que un modelo 2L no tiene muchas oportunidades para cambiar el nivel de frecuencia y voltaje cada vez que el algoritmo se aplica. Sin embargo, incluso en para este modelo, la heurística BM alcanza beneficios de un 24 % (ver Figura 7.3(a), Mezcla 3).

- Por último, la conclusión principal a la que podemos llegar es que usando el planificador de ahorro energético con un elevado número de niveles de frecuencia (5L) y una buena heurística de equilibrado de la carga (BM o BC), los beneficios obtenidos son un 40 % mayores que si utilizamos un modelo 2LN con ninguna heurística.

Capítulo 8

Conclusiones y trabajo futuro

En este trabajo se han presentado dos planificadores con ahorro energético para sistemas de tareas de tiempo real estrictas y no estrictas basados en un procesador multicore multihilo. Ambos modelos aplican la técnica de Dynamic Voltage Scaling y ajustan la frecuencia del procesador a la carga que se esté ejecutando y han sido evaluados usando diferentes conjuntos de mezclas formadas por benchmarks en un procesador empotrado bi-core multihilo.

Para el primer modelo presentado (Capítulo 4), los resultados experimentales muestran que los algoritmos de planificación pueden diseñarse para un sistema con un único nivel de frecuencia y voltaje, sin embargo, si aplicamos las técnicas DVS los beneficios pueden ser mucho más elevados. Los resultados, obtenidos explorando diferentes escenarios (mediante la variación del tiempo entre llegadas de las tareas), muestran que, de media, se pueden alcanzar beneficios de aproximadamente un 34 %, 18 % y 67 %.

En el segundo modelo presentado (Capítulo 5), se han introducido más modelos (5L, 3L y 2L) con más niveles de energía y una serie de heurísticas de equilibrado de la carga, para distribuir las tareas de tiempo real entre los cores, que actúan como catalizadores del solapamiento, reduciendo así la energía consumida por el sistema. Los experimentos muestran que las heurísticas tienen un fuerte impacto en el consumo energético. Los re-

sultados muestran que la heurística BM es la mejor heurística y que para un nivel dado de frecuencia y voltaje, proporciona sobre un 21 % de ahorro energético más que si ninguna heurística es aplicada. Además, para la misma heurística, puede observarse que el número de frecuencias implementadas también afecta a la energía consumida. Por ejemplo, el ahorro energético alcanzado con 5 niveles de frecuencia alcanza, de media, un 32 % en relación a un sistema de 2 niveles. Por último si combinamos la heurística BM con un amplio rango de frecuencias (5L), la energía consumida es, de media, un 51 % menos que para un sistema que trabaje sólo con un rango de energía.

Como puede deducirse, por los resultados obtenidos de los dos modelos presentados, una buena distribución de la carga es muy importante de cara a reducir el consumo. Por este motivo, actualmente estamos trabajando en nuevas heurísticas de particionado que en tiempo de ejecución determinen la distribución idónea. En estos momentos estamos explorando distintas políticas de particionado que distribuyan las tareas entre los cores atendiendo a diferentes factores como los recursos requeridos por cada tarea (memoria y procesador) y la utilización del sistema consumida por ellas. En el Anexo A se detallan estas nuevas heurísticas de particionado.

También se ha ampliado el modelo para trabajar con procesadores de 4 y 6 cores. Ampliando el número de cores del procesador, existe un rango de posibilidades a la hora de implementar el algoritmo DVS, como por ejemplo aplicar un DVS global para todos los cores, o aplicar DVS locales para cada par de cores, pudiendo ampliar los beneficios obtenidos.

Apéndice A

Nuevas heurísticas de particionado

Las heurísticas de distribución de las tareas tienen un fuerte impacto a la hora de ahorrar consumo cuando ejecutamos un conjunto dado de tareas, como puede deducirse de los resultados obtenidos tras evaluar los diferentes planificadores con ahorro energético propuestos.

En el modelo de tareas de tiempo real no estrictas, se realizaba una distribución de tareas muy simple, basándonos en el WCET de las tareas y distribuyendo las tareas más largas en ambos core. Esta distribución, aunque sencilla, influía a la hora del gasto energético. En el modelo de tareas de tiempo real estrictas ya se introducen heurísticas de particionado más complejas que basándose en los requisitos de memoria y CPU del conjunto de tareas, distribuyen las tareas de forma que el solapamiento entre ellas se ve incentivado. En ambos modelos, el particionado de tareas se hacía de manera off-line, es decir, una vez que las tareas iniciales comenzaban su ejecución, las heurísticas dejaban de aplicarse y pasaban a aplicarse heurísticas más sencillas como el WF.

Como se ha observado que las heurísticas de planificación son muy importantes a la hora de ahorrar energía cuando se ejecuta un conjunto de tareas, se han diseñado heurísticas de planificación nuevas, que basándose también en los requisitos de memoria y procesador, tienen en cuenta la utilización del sistema y se aplican a la distribución de las tareas a

lo largo de todo su tiempo de ejecución (on-line). Estas heurísticas se han llamado BR (Equilibrado de recursos) y LBR (Equilibrado de recursos con límite) y cada una de ella se ha implementado en dos versiones. La versión en la qué prima distribuir los requisitos de procesador (BR-C y LBR-C) y la versión en la qué lo que prima es la distribución de los requisitos de memoria (BR-M y LBR-M).

La heurística BR distribuye la carga del sistema atendiendo al porcentaje global de memoria/cpu de cada core, es decir, cuando llega una tarea nueva al sistema, el particionador selecciona el core cuyas tareas tienen menos requisitos de memoria/cpu. Antes de asignar este core a la tarea entrante, el particionador comprueba que el core pueda garantizar la planificabilidad del sistema con esta nueva tarea, es decir que la utilización actual del core más la utilización de la tarea que va a recibir sea menor que 1. En caso de no ser así, el particionador asignará a la nueva tarea el core con menor utilización de los cores del sistema (el más descargado). Esta comprobación es necesaria debido a que pueden darse situaciones en las que el core con menos requisitos de memoria/cpu no sea el core con menor utilización y de no realizarse las restricciones de tiempo real podrían no garantizarse. Una descripción del algoritmo BR puede verse en la Figura A.1.

Con la heurística anterior se pretende aumentar el tiempo de solapamiento mediante la distribución de los recursos. El problema que presenta esta heurística es que se pueden dar situaciones donde un core podría estar muy cargado mientras que otro no tanto si las tareas que se ejecutan en él consumen muchos recursos. Dado que el planificador se basa en un DVS global hemos de procurar que ambos cores tengan un nivel de utilización similar. Para solucionar ese problema se ha diseñado la heurística LBR.

La heurística LBR distribuye, en la medida de lo posible, la carga del sistema entre los cores del mismo a la vez que incentiva el solapamiento distribuyendo los recursos de las tareas. Esta heurística garantiza la planificabilidad del sistema y mantiene la velocidad del procesador tan baja como es posible. Funciona de la siguiente manera: Primero se ordenan las tareas por la cantidad de recursos que consumen. Una vez ordenadas, se realiza una

1: **Algoritmo: Equilibrado de Recurso (BR)**
2: **Entradas:** $tareas$: lista de tareas que se van a distribuir
3: **Salidas:** $par(tarea, core\_asignado)$: Cores asignados a las correspondientes tareas
4: $lista\_recursos\_tarea \leftarrow \oslash$;
5: **for all** $tasks$ **do**
6: insertar $tarea$ en $lista\_recursos\_tarea$ atendiendo a los recursos que necesite
7: **end for**
8: **while** $lista\_recursos\_tarea$ no esté vacía **do**
9: $utilizacion$ = max_utilizacion($lista\_recursos\_tarea$)
10: $tarea$ = primero($lista\_recursos\_tarea$)
11: $core\_seleccionado$=core_menos_recursos()
12: **if** $utilizacion\_core\_seleccionado$ es mayor que $1 - utilizacion$ **then**
13: $core\_seleccionado$=core_menos_utilizacion()
14: **end if**
15: $utilizacion\_core\_seleccionado$=$utilizacion\_core\_seleccionado$+$utilizacion$
16: **if** $utilizacion\_core\_seleccionado$ es mayor que 1 **then**
17: Finalizar: El sistema no es planificable
18: **end if**
19: **end while**

Figura A.1. Heuristica de equilibrado de recursos

distribución de las tareas basándonos en este criterio, es decir, el core seleccionado será el que menos recursos esté consumiendo (como en la heurística BR) pero, cuando uno de los cores del sistema pasa un cierto umbral de utilización (en este caso la utilización media del sistema), el core seleccionado a partir de ese momento será el de menor utilización del sistema. Con esta heurística se pretende equilibrar lo máximo posible la utilización (para que todos los cores necesiten la misma frecuencia para garantizar los deadlines de las tareas que están ejecutándose) a la vez que se distribuyen los recursos que consumen las tareas, permitiendo así aumentar el solapamiento. La Figura A.2 muestra una descripción detalla del algoritmo para esta heurística.

Ambas heurísticas han sido comparadas con la heurística Worst Fit cuyo objetivo es distribuir la carga del sistema entre los cores del mismo y además es conocida por ser la heurística con mejores prestaciones de entre las existentes [17].

1: **Algoritmo:** Equilibrado Limitado de Recursos (LBR)
2: **Entradas:** $tareas$: lista de tareas que se van a distribuir
3: **Salidas:** $par(tarea, core\_asignado)$: Cores asignados a las correspondientes tareas
4: $lista\_recursos\_tarea \leftarrow \oslash$;
5: **for all** $tasks$ **do**
6: insertar $tarea$ en $lista\_recursos\_tarea$ atendiendo a los recursos que necesita
7: **end for**
8: $utilizacion\_limite$=utilizacion_media($tareas$)
9: **while** $lista\_recursos\_tarea$ no esté vacía **do**
10: $tarea$ = primero($lista\_recursos\_tarea$)
11: $core\_seleccionado$=core_menos_recursos()
12: $utilizacion\_predicha$=$utilizacion\_core\_seleccionado$+$utilizacion\_tarea$
13: **if** $utilizacion\_predicha$ es mayor que $utilizacion\_limite$ **then**
14: $core\_seleccionado$=core_menos_utilizacion()
15: **end if**
16: $utilizacion\_core\_seleccionado$=$utilizacion\_core\_seleccionado$+$utilizacion\_tarea$
17: **if** $utilizacion\_core\_seleccionado$ es mayor que 1 **then**
18: Fin: El sistema no es planificable
19: **end if**
20: **end while**

Figura A.2. Heurística de equilibrado limitado de recursos

Bibliografía

[1] A. Burns and A. Wellings. *Sistemas de tiempo real y lenguajes de programación*. AddisonWesley, 2001.

[2] Inc. Ubicom. The Ubicom IP3023 wireless network processor. Technical report, 2003.

[3] ARM Limited. ARM11 MPCore Processor. Technical reference manual. Technical report, 2006.

[4] K. Hirata and J. Goodacre. ARM MPCore; The streamlined and scalable ARM11 processor core. In *Proceedings of the 2007 conference on Asia South Pacific design automation*, pages 747–748, 2007.

[5] A. El-Haj-Mahmoud, A.AL-Zawawi, A. Anantaraman, and E. Rotenberg. Virtual multiprocessor: an analyzable, high-performance architecture for real-time computing. In *Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 213–224, 2005.

[6] A. Anantaraman, K. Seth, K. Patil, E. Rotenberg, and F. Mueller. Virtual simple architecture (visa): Exceeding the complexity limit in safe real-time systems. In *Proceedings of the 30th International Symp. On Computer Architecture*, pages 350–361, 2003.

[7] F. Cazorla, P. Knijnenburg, R. Sakellariou, E. Fernández, A. Ramirez, and M. Valero. Predictable performance in SMT processors: Synergy between the OS and SMTs. *IEEE Transactions on computers*, 55(7):785–799, 2006.

[8] C. Hung, J. Chen, and T. Kuo. Energy-efficient real-time task scheduling for a DVS system with a Non-DVS processing element. In *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 303–312, 2006.

[9] R. Ubal, J. Sahuquillo, S. Petit, H. Hassan, and P. López. Power reduction in advanced embedded IPC processors. *International Journal of Intelligent Automation and Soft Computing*, Special Issue on Embedded System and Software for Intelligent Pervasive Computing(In Press), 2008.

[10] M. Verma, L. Wehmeyer, and P. Marwedel. Cache-aware scratchpad-allocation algorithms for energy-constrained embedded systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 25(10):2035–2051, 2006.

[11] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the 1pARM microprocessor system. In *Proceedings of the 1998 International Symp. On Low Power Electronics and Desing*, pages 96–101, 2000.

[12] J. Pouwelse, K. Langedoen, and H. Sips. Application-directed voltage scaling. *IEEE Transactions on Very Scale integration (TVLSI)*.

[13] Transmeta Corp. Transmeta Crusoe. Technical report, [Online]. Available: http://www.transmeta.com.

[14] INTEL Corp. INTEL-XEON. Technical report, [Online]. Available: http://www.intel.com/products/processor/xeon.

[15] AMD Corporation. AMD Duron TM . Technical report, [Online]. Available: http://www.amd.com.

[16] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. *Design Automation Conference*, 0:134–139, 1999.

[17] T. AlEnawy and H. Aydin. Energy-aware task allocation for Rate Monotonic scheduling. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 213–223, 2005.

[18] J. Anderson and S. Baruah. Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms. In *Proceedings of the 24th IEEE International Conference on Distributed Computing*, pages 428–435, 2004.

[19] M. Marinoni and G. Buttazzo. Elastic DVS management in processors with discrete voltage/frequency modes. *IEEE Transactions on Industrial Informatics*, 3(1):51–62, 2007.

[20] J. Donald and M. Martonosi. Techniques for multicore thermal management: classification and new exploration. In *Proceedings of the 33rd annual international symposium on Computer Architecture*, pages 78–88, 2006.

[21] P. Pushner and A. Burns. A review of Worst-Case Execution-Time analysis. *The international Journal of Time-Critical computing Systems*, 18:115–128, 2000.

[22] J. Stankovic, M. Spuri, M. DiNatale, and G. Butazzo. Implications of Classical Scheduling Results for Real-Time Systems . *IEEE Computer*, 28:16–25, 1995.

[23] J. Stankovic, M. Spuri, K. Ramamritham, and G. Butazzo. Deadline Scheduling for Real-Time systems EDF and related Algorithms . *Kluwer Academic Publishers*, 1998.

[24] D. Tullsen, S. Eggers, and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 392–403, 1995.

[25] C. McNairy and R. Bhatia. Montecito: a dual-core, dual-thread Itanium processor. *IEEE Micro*, 25(2):10–20, 2005.

[26] P. Kongetira and and K. Olukotun K. Aingaran. Niagara: a 32-way multithreaded Sparc processor. *IEEE Micro*, 25(2):21–29, 2005.

[27] G. Hinton et al. The microarchitecture of the Pentium 4 processor. Intel Technology Journal. 2001.

[28] R. Kalla, S. Balaram, and J.M. Tendler. IBM Power5 chip: a dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[29] Y.Zhu and F. Mueller. ”feedback EDF scheduling of real-time tasks exploiting dynamic voltage scaling”. *Real-Time Systems Journal*, 31(1-3):33–63, 2005.

[30] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS management in web servers. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 52–63, 2003.

[31] A. Qadi, S. Goddard, and S. Farritor. A dynamic voltage scaling algorithm for sporadic tasks. In *Proceedings of the 24th IEEE Real-Time Systems Symposium.*, pages 52–62, 2003.

[32] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium, Workshop on Parallel and Distributed Real-Time Systems*, 2003.

[33] A. El-Haj-Mahmoud and E. Rotenberg. Safely exploiting multithreaded processors to tolerate memory latency in real-time systems. In *Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 2–13, 2004.

[34] G. R. Goud, N. Sharma, K. Ramamritham, and S. Malewar. Efficient real-time support for automotive applications: A case study. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 335–341, 2006.

[35] R. Ubal, J. Sahuquillo, S. Petit, and P. López. A simulation framework to evaluate multicore-multithreaded processors. In *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*, pages 62–68, 2007.

[36] Embedded Microprocessor Benchmarking Consortium. Automotive and telecomunication benchmark suites. Technical report, [Online]. Available: http://www.eembc.org/.

[37] C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. Technical report, [Online]. Available: http://euler.slu.edu/ fritts/mediabench/mb1/.

[38] Maarladen Real time Research Center. WCET analysis project. WCET Benchmark programs. Technical report, [Online]. Available: http:/www.mrtc.mdh.se/projects/wcet/benchmarks.html, 2006.

[39] K. Krewell. Pentium m hits the street. microprocessor report. Technical report, [Online]. Available: http://www.transmeta.com, March 2003.

[40] R. Watanabe, M.Kondo, M.Imai, H.Nakamura, and T.Nanya. Task scheduling under performance constraints for reducing the energy consumption of the GALS multiprocessor SoC. In *Proceedings of the Design Automation and Test in Europe*, 2007.

[41] E. Seo, J. Jeong, S. Park, and J. Lee. Energy efficient scheduling of Real-Time tasks on multicore processors. *IEEE Transactions on parallel and distributed systems*, 19(11), 2008.