Dynamic power-aware techniques for real-time multicore embedded systems

JOSÉ LUIS MARCH CABRELLES

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Universitat Politècnica de València

Ph.D. Thesis

Dynamic Power-Aware Techniques for Real-Time Multicore Embedded Systems

Author:
José Luis March Cabrelles

Advisors:
Prof. Salvador V. Petit Martí
Prof. Julio Sahuquillo Borrás

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
(Computer Engineering)

in the
Parallel Architectures Group
Department of Computer Engineering

December 2014

Collection Doctoral Thesis

© 2015, of the present edition: Editorial Universitat Politècnica de València
Telf.: 963 877 012 / www.lalibreria.upv.es

ISBN: 978-84-9048-327-5 (printed version)

This guy's walking down a street when he falls in a hole. The walls are so steep, he can't get out. A doctor passes by, and the guy shouts up, "Hey you, can you help me out?" The doctor writes a prescription, throws it down in the hole and moves on. Then a priest comes along, and the guy shouts up "Father, I'm down in this hole, can you help me out?" The priest writes out a prayer, throws it down in the hole and moves on. Then a friend walks by. "Hey Joe, it's me, can you help me out?" And the friend jumps in the hole. Our guy says, "Are you stupid? Now we're both down here." The friend says, "Yeah, but I've been down here before, and I know the way out."

Leo McGarry (played by John Spencer), The West Wing.

TECHNICAL UNIVERSITY OF VALENCIA

Abstract

School of Computer Engineering
Department of Computer Engineering

Doctor of Philosophy
(Computer Engineering)

Dynamic Power-Aware Techniques for Real-Time Multicore Embedded Systems

by José Luis March Cabrelles

The continuous shrink of transistor sizes has allowed more complex and powerful devices to be implemented in the same area, which provides new capabilities and functionalities. However, this complexity increase comes with a considerable rise in power consumption. This situation is critical in portable devices where the energy budget is limited and, hence, battery lifetime defines the usefulness of the system. Therefore, power consumption has become a major concern in the design of real-time multicore embedded systems.

This dissertation proposes several techniques aimed to save energy without sacrifying real-time schedulability in this type of systems. The proposed techniques deal with different main components of the system. In particular, the techniques affect the task partitioner and the scheduler, as well as the memory controller.

Some of the techniques are especially tailored for multicores with shared *Dynamic Voltage and Frequency Scaling* (DVFS) domains. Workload balancing among cores in a given domain has a strong impact on power consumption, since all the cores sharing a DVFS domain must run at the speed required by the most loaded core.

In this thesis, a novel **workload partitioning** algorithm is proposed, namely *Load-bounded Resource Balancing* (*LRB*). The proposal allocates tasks to cores to balance a given resource (processor or memory) consumption among cores, improving real-time schedulability by increasing overlapping between processor and memory. However, distributing tasks in this way regardless the individual core utilizations could lead to unfair load distributions. That is, one of the cores could become much loaded than the others. To avoid this scenario, when a given utilization threshold is exceeded, tasks are assigned to the least loaded core.

Unfortunately, workload partitioning alone is sometimes not able to achieve a good workload balance among cores. Therefore, this work also explores novel **task migration** approaches. Two task migration heuristics are proposed. The first heuristic, referred to as *Single Option Migration* (*SOM*), attempts to perform only one migration when the workload changes to improve utilization balance. Three variants of the *SOM* algorithm have been devised, depending on the point of time the migration attempt is performed: when a task arrives to the system (SOM_{in}), when a task leaves the system (SOM_{out}), and in both cases (SOM_{in-out}). The second heuristic, referred to as *Multiple Option Migration* (*MOM*) explores an additional alternative workload partitioning before performing the migration attempt.

Regarding the memory controller, **memory controller scheduling policies** are devised. Conventional policies used in *Non Real-Time* (NRT) systems are not appropriate for systems providing support for both *Hard Real-Time* (HRT) and *Soft Real-Time* (SRT) tasks. Those policies can introduce variability in the latencies of the memory requests and, hence, cause an HRT deadline miss that could lead to a critical failure of the real-time system. To deal with this drawback, a simple policy, referred to as *HR-first*, which prioritizes requests of HRT tasks, is proposed. In addition, a more advanced approach, namely *ATR-first*, is presented. *ATR-first* prioritizes only those requests of HRT tasks that are necessary to ensure real-time schedulability, improving the *Quality of Service* (QoS) of SRT tasks.

Finally, this thesis also tackles **dynamic execution time estimation**. The accuracy of this estimation is important to avoid deadline misses of HRT tasks but also to increase QoS in SRT systems. Besides, it can also help to improve the schedulability of the systems and reduce power consumption. The *Processor-Memory* (*Proc-Mem*) model, that dynamically predicts the execution time of real-time application for each frequency level, is proposed. This model measures at the first hyperperiod, making use of *Performance Monitoring Counters* (PMCs) at run-time, the portion of time that each core is performing computation (*CPU*), waiting for memory (*MEM*), or both (*OVERLAP*). This information will be used to estimate the execution time at any other working frequency.

UNIVERSIDAD POLITÉCNICA DE VALENCIA

Resumen

Escuela Técnica Superior de Ingeniería Informática
Departamento de Informática de Sistemas y Computadores

Doctor en Filosofía
(Ingeniería Informática)

Técnicas Dinámicas con Control de Consumo para Sistemas Empotrados Multinúcleo de Tiempo Real

por José Luis March Cabrelles

La continua reducción del tamaño de transistor ha permitido la implementación de dispositivos más complejos y potentes en una misma área, lo que proporciona una mayor capacidad y funcionalidad. Sin embargo, este incremento en la complejidad conlleva un considerable aumento del consumo energético. Esta situación es crítica en dispositivos móviles, donde la capacidad energética está limitada y, por lo tanto, la vida útil de la batería define la usabilidad del sistema. En consecuencia, el consumo energético se ha convertido en un aspecto crucial en el diseño de sistemas empotrados multinúcleo de tiempo real.

Esta disertación propone diversas técnicas con el objetivo de ahorrar energía sin sacrificar la planificabilidad de tiempo real en este tipo de sistemas. Las técnicas propuestas actúan sobre diferentes componentes principales del sistema. En particular, estas técnicas afectan al particionador de tareas y al planificador, así como al controlador de memoria.

Algunas de estas técnicas están especialmente concebidas para sistemas multinúcleo con dominios DVFS (*Dynamic Voltage and Frequency Scaling*) compartidos. Equilibrar la carga entre los núcleos en un dominio dado tiene un fuerte impacto en el consumo de energía, ya que todos los núcleos que comparten un dominio DVFS deben funcionar a la velocidad requerida por el núcleo más cargado.

En esta tesis se propone un nuevo algoritmo de **particionado de carga**, denominado *Load-bounded Resource Balancing* (*LRB*). La propuesta asigna las tareas a los núcleos para equilibrar el consumo de un recurso dado (procesador o memoria) entre los núcleos, mejorando la planificabilidad de tiempo real al incrementar el solapamiento entre procesador y memoria. Sin embargo, distribuir las tareas de esta forma sin tener en cuenta

la utilización individual de los núcleos podría dar lugar a distribuciones de carga desequilibradas. Es decir, uno de los núcleos podría estar mucho más cargado que el resto. Con tal de evitar este escenario, cuando se supera un umbral de utilización dado, las tareas son asignadas al núcleo menos cargado.

Desafortunadamente, sólo con el particionado de carga a veces no es suficiente para conseguir un buen equilibrado de carga entre los núcleos. Por lo tanto, este trabajo también explora nuevas aproximaciones basadas en **migración de tareas**. Se proponen dos heurísticas con migración de tareas. La primera heurística, llamada *Single Option Migration* (*SOM*), intenta realizar sólo una migración cuando la carga cambia para mejorar el equilibrio de la utilización. Se han ideado tres variantes del algoritmo *SOM*, en función del instante de tiempo en el que se realiza el intento de migración: cuando una tarea llega al sistema (SOM_{in}), cuando una tarea sale del sistema (SOM_{out}), y en ambos casos (SOM_{in-out}). La segunda heurística, denominada *Multiple Option Migration* (*MOM*), explora un particionado de carga alternativo adicional antes de realizar el intento de migración.

Respecto al controlador de memoria, se han concebido dos **políticas de planificación para el controlador de memoria**. Las políticas convencionales usadas en sistemas que no son de tiempo real no son apropiadas para sistemas que dan soporte a tareas tanto de tiempo real estricto (*Hard Real-Time* o HRT) como de tiempo real flexible (*Soft Real-Time* o SRT). Esas políticas pueden introducir variabilidad en las latencias de las peticiones de memoria y, por consiguiente, causar una pérdida de deadlines que podría conllevar un fallo crítico del sistema de tiempo real. Para tratar este inconveniente, se propone una política simple, denominada *HR-first*, que prioriza peticiones de tareas HRT. Además, se presenta una aproximación más elaborada, llamada *ATR-first*. *ATR-first* prioriza sólo aquellas tareas HRT que son necesarias para asegurar la planificabilidad de tiempo real, mejorando la calidad de servicio (*Quality of Service* o QoS) de las tareas SRT.

Finalmente, esta tesis también aborda la **estimación dinámica del tiempo de ejecución**. La precisión de esta estimación es importante para evitar la pérdida de deadlines de tareas HRT, pero también para incrementar la QoS en sistemas SRT. Además, también puede ayudar a mejorar la planificabilidad del sistema y reducir el consumo de energía. Se propone el modelo *Processor-Memory* (*Proc-Mem*), que dinámicamente predice el tiempo de ejecución de aplicaciones de tiempo real para cada nivel de frecuencia. Este modelo mide en el primer hiperperiodo, haciendo uso de *Performance Monitoring Counters* (PMCs) en tiempo de ejecución, la porción de tiempo que cada núcleo está realizando cómputos (*CPU*), esperando a memoria (*MEM*), o ambos (*OVERLAP*). Esta información será usada para estimar el tiempo de ejecución a cualquier otra frecuencia.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Resum

Escola Tècnica Superior d'Enginyeria Informàtica
Departament d'Informàtica de Sistemes i Computadors

Doctor en Filosofia
(Enginyeria Informàtica)

Tècniques Dinàmiques amb Control de Consum per a Sistemes Encastats Multinucli de Temps Real

per José Luis March Cabrelles

La contínua reducció de la grandària dels transistors ha permès la implementació de dispositius més complexes i potents en una mateixa àrea, proporcionant una major capacitat i funcionalitat. No obstant, aquest increment en la complexitat comporta un considerable augment del consum energètic. Aquesta situació és crítica en dispositius mòbils, on la capacitat energètica està limitada i, per tant, la vida útil de la bateria defineix la usabilitat del sistema. En conseqüència, el consum energètic s'ha convertit en un aspecte crucial en el disseny de sistemes encastats multinucli de temps real.

Aquesta dissertació proposa diverses tècniques amb l'objectiu d'estalviar energia sense sacrificar la planificabilitat de temps real en aquest tipus de sistemes. Les tècniques proposades actuen sobre diferents components principals del sistema. En particular, aquestes tècniques afecten el particionador de tasques i el planificador, així com el controlador de memòria.

Algunes d'aquestes tècniques estan especialment concebudes per a sistemes multinucli amb dominis DVFS (*Dynamic Voltage and Frequency Scaling*) compartits. Equilibrar la càrrega entre els nuclis en un domini donat té un fort impacte en el consum d'energia, ja que tots els nuclis que comparteixen un domini DVFS han de funcionar a la velocitat requerida pel nucli més carregat.

En aquesta tesi es proposa un nou algorisme de **particionat de càrrega**, anomenat *Load-bounded Resource Balancing* (*LRB*). La proposta assigna les tasques als nuclis per equilibrar el consum d'un recurs donat (processador o memòria) entre els nuclis, millorant la planificabilitat de temps real en incrementar el solapament entre processador i memòria. No obstant, distribuir les tasques d'aquesta manera sense tenir en compte la

utilització individual dels nuclis podria donar lloc a distribucions de càrrega desequilibrades. És a dir, un dels nuclis podria estar molt més carregat que la resta. Per tal d'evitar aquest escenari, quan se supera un llindar d'utilització donat, les tasques són assignades al nucli menys carregat.

Desafortunadament, només amb el particionat de càrrega de vegades no és suficient per aconseguir un bon equilibrat de càrrega entre els nuclis. Per tant, aquest treball també explora noves aproximacions basades en **migració de tasques**. Es proposen dues heurstiques amb migració de tasques. La primera heurística, anomenada *Single Option Migration* (*SOM*), intenta realitzar només una migració quan la càrrega canvia per millorar l'equilibri de la utilització. S'han ideat tres variants de l'algorisme *SOM*, en funció de l'instant de temps en què es realitza l'intent de migració: quan una tasca arriba al sistema (SOM_{in}), quan una tasca surt del sistema (SOM_{out}), i en ambdós casos (SOM_{in-out}). La segona heurística, anomenada *Multiple Option Migration* (*MOM*), explora un particionat de càrrega alternatiu addicional abans de fer l'intent de migració.

Respecte al controlador de memòria, s'han concebut dos **polítiques de planificació per al controlador de memòria**. Les poltiques convencionals usades en sistemes que no són de temps real no són apropiades per a sistemes que donen suport a tasques tant de temps real estricte (*Hard Real-Time* o HRT) com de temps real flexible (*Soft Real-Time* o SRT). Aquestes polítiques poden introduir variabilitat en les latències de les peticions de memòria i, per tant, causar una pèrdua de deadlines que podria comportar una fallada crítica del sistema de temps real. Per tractar aquest inconvenient, es proposa una política simple, anomenada *HR-first*, que prioritza peticions de tasques HRT. A més, es presenta una aproximació més elaborada, anomenada *ATR-first*. *ATR-first* prioritza només aquelles tasques HRT que són necessàries per a assegurar la planificabilitat de temps real, millorant la qualitat de servei (*Quality of Service* o QoS) de les tasques SRT.

Finalment, aquesta tesi també aborda la **estimació dinàmica del temps d'execució**. La precisió d'aquesta estimació és important per evitar la pèrdua de deadlines de tasques HRT, però també per incrementar la QoS en sistemes SRT. A més, també pot ajudar a millorar la planificabilidad del sistema i reduir el consum d'energia. Es proposa el model *Processor-Memory* (*Proc-Mem*), que dinàmicament prediu el temps d'execució d'aplicacions de temps real per a cada nivell de freqüència. Aquest model mesura en el primer hiperperíode, fent ús de *Performance Monitoring Counters* (PMCs) en temps d'execució, la porció de temps que cada nucli està realitzant còmputs (*CPU*), esperant a memòria (*MEM*), o ambdós (*OVERLAP*). Aquesta informació serà usada per estimar el temps d'execució a qualsevol altra freqüència.

Acknowledgements

This work would have never been possible without the support of many people throughout these years. First of all, I have to mention my parents and my sister, without whom I would not be who I am. Second, I also have to acknowledge my advisors Salva Petit and Julio Sahuquillo, as well as Houcine Hassan, for their guidance and help in achieving this goal.

Looking back, I started my journey in this university ten years ago. Now, after an Engineering Degree, a Master's Degree and a Ph.D., I am happy to remember a lot of moments and experiences, and so I want to thank everyone in the university that helped me feel really comfortable all these years. *Thanks*.

Regarding my research group mates, I appreciate the great atmosphere they all have built, not only at work but also going out together for lunch or dinner, japanese food, *paella* or *torrà* at Carlos' place, karting, etc. Although the full list would be longer, I would like to mention Roberto, Crispín, Carlos, Josué, José María, Davide, Javi, José Cano, etc. Moreover, working in my laboratory allowed me to build up a good friendship with Raúl and Mónica. I had interesting and deep discussions with them about everything including, of course, food and running. The three of us became an excellent team facing the daily problems of work and life. Finally, a special mention deserve both *biondi* Alejandro Valero and Mario Lodde, with whom I shared a lot of conversations, laughs, travels, concerts, conferences and, in the end, great moments. I met Mario when studying our Master's Degree, he taught me italian and music stuff and, as years went by, we were turning into closer *amici*. Àlex and I became so good friends that we even shared an apartment (with a *Death Room* and a *Rat's Room*) for almost two years of our lives. Currently, we are far away from each other due to our jobs, however, I know with them I have two trusty friends no matter where we are. *Grazie Mille a Tutti*.

I would like to thank Blas for introducing me to *Fútbol Xtreme*, where I have not only enjoyed playing football but I have also made a lot of friends. As I have been able to confirm throughout my life (*C.I.L.*, *C.D. Deportes Arnau*, *C.D. Serranos*, *Fac-In*, *Los de Siempre*, *Alfonso*, *HideSport*, *Nottingham Por*, etc.), that is the main result of playing football: to make friends. In fact, I have to acknowledge football itself, as well as running, music, movies and TV series, for being such a useful tool to face stress periods during the development of this work. *Gracias*.

From my stay at the MDH (Mälardalens Högskola) in Västerås (Sweden), I am very grateful to Thomas Nolte and Moris Behnam. They were very kind, introducing me to

their research group and making me feel like anyone else on the team. By the way, an excellent group of people not only in research duties but also in a personal sense. In the same way, several nice words should be said about some friends I made in Västerås. Sama and Richi were gentle and helpful in the time we spent together. Miguel and Laura are my family in Sweden, and I know I can count on them. And with Berta I had deep and enriching discussions about life and muffins. *Tack Så Mycket.*

Here in Spain I also had friends that supported me during all these years: Roman, Caraltu, Cenci, Maci, Toni, Kent, Amparo, Yasmin, Paula, Cedric, etc. I remember with special affection the year I shared an apartment with Francesco and Silvia. More recently, last year I shared another apartment with Jaume and Dani, and they quickly turned from flatmates into actual friends. Back to the past again, I met Irene fifteen years ago at school, and despite we now live in different cities, she is always there to state a helpful *judge*ment in anything that happens in my life. *Gràcies.*

Crossing the Atlantic Ocean until Brazil, I can not forget Dra. Dra. Juliana Dos Anjos, who was a really close friend during her stay in Valencia. If we go further through the Pacific Ocean we will arrive in Japan and find Dr. Pablo Lamilla (a.k.a. Buci). He was a constant support when we were at university in Valencia studying together and also later when he moved to Kōchi. *Muito Obrigado* & *Dōmo Arigatō.*

And last but not least, I really would like to thank my most true friends Alejandro Arcos and Jonatan Linares (a.k.a. Xona). Arcos is always there to contradict me, even when he agrees with me. We shared many moments, at university, in the football field or traveling. And he hosted me many times in London and Madrid, where I felt like home. With Xona I also shared many experiences, working together in university projects, talking about movies and soundtracks, having lunch at Tony's or driving a Jeep through the caribbean jungle. Besides, he is a dynamic entrepreneurial endeavour with many start-up projects in his mind, in which I try to help him. The three of us shared many great nights, starting with *bravas* at La Rosa Negra having deep discussions and laughs, and ending where the *flow* takes us. I know Arcos and Xona for ten years, and they were always with me in *the good, the bad and the ugly* times.

Contents

Abstract v

Resumen vii

Resum ix

Acknowledgements xi

Contents xiii

List of Figures xvii

List of Tables xix

Acronyms xxi

1 Introduction 1
- 1.1 Motivation . . . 1
  - 1.1.1 Power-Aware Multicore Processors . . . 1
  - 1.1.2 Real-Time Task Partitioners and Schedulers . . . 2
  - 1.1.3 Memory Controller Scheduling Policies . . . 3
  - 1.1.4 Dynamic Execution Time Estimation . . . 4
- 1.2 Contributions of the Thesis . . . 4
- 1.3 Thesis Outline . . . 7

2 Related Work 9
- 2.1 Real-Time Scheduling in Power-Aware Multicore Processors . . . 9
  - 2.1.1 Task Partitioning . . . 11
  - 2.1.2 Task Migration . . . 11
- 2.2 Memory Controller Scheduling Policies . . . 12
- 2.3 Execution Time Estimation . . . 13

3 System Model 17
- 3.1 Baseline Design . . . 17
  - 3.1.1 Real-Time Tasks . . . 19
  - 3.1.2 Power-Aware Scheduler . . . 19
- 3.2 The Multi2Sim Simulation Framework . . . 21
  - 3.2.1 Simulation Models . . . 22

3.2.2 Main Proposed Extensions . . . 22
3.2.2.1 Task Repetition . . . 23
3.2.2.2 Priority . . . 24
3.2.2.3 Frequency . . . 25
3.2.2.4 Latency of Frequency Changes . . . 25
3.2.2.5 Task Partitioner . . . 26

4 Task Partitioning **29**
4.1 Introduction . . . 29
4.2 Partitioning Heuristics . . . 30
4.2.1 HRT Heuristic . . . 30
4.2.2 Power-Aware HRT Scheduler . . . 31
4.2.3 Providing Support for SRT Tasks . . . 32
4.3 Experimental Evaluation . . . 35
4.3.1 Designing and Planning Mix Execution for HRT Tasks . . . 36
4.3.2 Designing Hybrid Mixes . . . 39
4.3.3 Energy Savings for HRT Mixes . . . 39
4.3.4 Energy Savings versus Deadline Misses for Hybrid Mixes . . . 42
4.4 Conclusions . . . 46

5 Task Migration **47**
5.1 Introduction . . . 47
5.2 Proposed Task Migration Heuristics . . . 47
5.2.1 *Single Option Migration* Policies . . . 48
5.2.2 *Multiple Option Migration* Dynamic Partitioner . . . 49
5.3 Experimental Results . . . 51
5.3.1 Impact of Applying Migrations at Specific Points of Time . . . 53
5.3.2 Comparing *MOM* versus *SOM* Variants . . . 56
5.4 Conclusions . . . 59

6 Memory Controller Scheduling Policies **61**
6.1 Introduction . . . 61
6.2 Power-Aware Scheduler . . . 62
6.3 Memory Controller . . . 62
6.3.1 *HRT Requests First* . . . 63
6.3.2 *Active Task Requests First* . . . 64
6.4 Experimental Results . . . 65
6.5 Conclusions . . . 69

7 Dynamic Execution Time Estimation **71**
7.1 Introduction . . . 71
7.2 System Architecture . . . 72
7.2.1 Partitioning and Scheduling . . . 73
7.2.2 Memory System . . . 73
7.3 *Processor-Memory* Model . . . 74
7.4 Model Validation . . . 76
7.5 *Frequency Selection Policy* based on the *Proc-Mem* Model . . . 80
7.5.1 Scheduler Working Behavior . . . 80

7.5.2 Experimental Results 81
7.6 Conclusions 83

8 Conclusions **85**
8.1 Contributions 85
8.2 Publications 87

References **91**

List of Figures

1.1 Components of the studied system and thesis contributions 5

3.1 Modeled system. 18
3.2 Active and inactive periods example. 23
3.3 Decreasing and increasing frequency. 26

4.1 *Load-bounded Resource Balancing* heuristic. Legend: RES refers to the resource being balanced. 31
4.2 HRT and SRT frequency calculation. 33
4.3 Frequency adjustment due to SRT deadline misses. 34
4.4 Example of the Gantt chart for mix 6. Continuous line means that the period is active. Discontinuous line means that the task is out of the system (inactive period). 38
4.5 Normalized energy (2 cores). 40
4.6 Normalized energy (4 cores). 41
4.7 Normalized deadline misses and energy (2 cores). 43
4.8 Normalized deadline misses and energy (4 cores). 44

5.1 Example of task migrations to balance the system workload. 49
5.2 *Migration Attempt* algorithm. 50
5.3 *Multiple Option Migration* dynamic partitioner algorithm. 51
5.4 SOM_{in-out} versus *MOM* working example. 52
5.5 *SOM* variants comparison for different DVFS levels and number of cores. 54
5.6 Effective action of the SOM_{in} partitioning algorithm. 55
5.7 Differences of the required frequencies in the 3-core system for mix 4. . . . 57
5.8 SOM_{in-out} versus *MOM* for different DVFS levels and number of cores. . 58

6.1 *HR-first* request scheduling policy. 64
6.2 *ATR-first* request scheduling policy. 65
6.3 Normalized energy and deadline misses for 2 cores. 67
6.4 Normalized energy and deadline misses for 4 cores. 69

7.1 System model. 72
7.2 Execution time model. 74
7.3 Execution overlap between processor and memory for two different frequencies in a superscalar architecture. 75
7.4 Estimates of the *Proc-Mem* model in stand-alone execution in the single-core superscalar architecture. 77

7.5 Maximum deviation in processor cycles in a single-core superscalar architecture. 78
7.6 Average of maximum deviations in a multicore superscalar architecture. . 78
7.7 Estimates of the *Proc-Mem* model in a multicore processor. 79
7.8 Power-aware scheduler actions of the system across the hyperperiods. . . . 80
7.9 Normalized energy consumption of *Proc-Mem* and *CMAT* with respect to a system working at the maximum speed. 82

List of Tables

3.1 Machine parameters. 19
3.2 Benchmark description. 20
3.3 Frequency (F), voltage (V) and power (P) for each DVFS level. 21

4.1 Benchmarks requirements and classification in categories H, M, L. 35
4.2 Benchmark mixes. 36
4.3 Benchmarks classification intervals attending to their processor requirements and memory reference instructions. 37
4.4 Distribution of resource requirements and utilization (core 0, core 1) for each scheduling heuristic for mix 6. 37
4.5 QoS requirements trade-off. 45

5.1 Benchmarks and mixes. Legend: * the benchmark appears more than once in the mix. 53
5.2 Algorithms action on workload changes. 56
5.3 Average and standard deviation of task utilization. 59

6.1 Benchmarks and mixes. Legend: * the benchmark appears more than once in the mix. 66

7.1 Mix composition: benchmarks and instances of each benchmark. 81
7.2 Deadline misses in the *CMAT* and *Proc-Mem* models and active periods of the mixes. 83

Acronyms

ATR-first	**A**ctive **T**ask **R**equests **first**
BF	**B**est **F**it
CGMT	**C**oarse-**G**rain **M**ulti**T**hreading
CMAT	**C**onstant **M**emory **A**ccess **T**ime
DVFS	**D**ynamic **V**oltage and **F**requency **S**caling
EDF	**E**arliest **D**eadline **F**irst
FF	**F**irst **F**it
FSP	**F**requency **S**election **Policy**
FGMT	**F**ine-**G**rain **M**ulti**T**hreading
HRT	**H**ard **R**eal-**T**ime
HR-first	**H**RT **R**equests **first**
LRB	**L**oad-bounded **R**esource **B**alancing
MOM	**M**ultiple **O**ption **M**igration
MPSoC	**M**ulti**P**rocessor **S**ystem-**o**n-**C**hip
NRT	**N**on **R**eal-**T**ime
OS	**O**perating **S**ystem
PDA	**P**ersonal **D**igital **A**ssistant
PMC	**P**erformance **M**onitoring **C**ounter
Proc-Mem	**Proc**essor-**Mem**ory
QoS	**Q**uality **o**f **S**ervice
RMS	**R**ate **M**onotonic **S**cheduling
SMP	**S**ymmetric **M**ultiprocessor **P**latform
SMT	**S**imultaneous **M**ulti**T**hreading
SOM	**S**ingle **O**ption **M**igration
SRT	**S**oft **R**eal-**T**ime

WCET	**W**orst **C**ase **E**xecution **T**ime
WF	**W**orst **F**it
WRR	**W**eighted **R**ound-**R**obin

Chapter 1

Introduction

1.1 Motivation

This chapter introduces the motivation and contributions of this thesis. The motivation deals with four main elements in a power-aware real-time multicore system: i) the power-aware multicore processor, ii) the real-time task partitioner and scheduler, iii) the memory controller, and iv) dynamic execution time estimation. These four elements are highlighted in bold letters in Figure 1.1. Next, we motivate the significance of these elements for the performance and power consumption of real-time systems.

1.1.1 Power-Aware Multicore Processors

Because of technology advances power consumption has emerged up as an important design issue in modern microprocessors. This fact is especially critical when these processors are designed for real-time systems (e.g., robotics, sensor networks) and embedded systems (e.g., PDAs, mobile phones), where the battery lifetime is a crucial issue [1]. As a consequence, research on reducing power consumption has become a hot research topic when designing embedded computing systems [2–4].

A straightforward way to reduce power consumption consists on using processors that do not implement the most power-hungry microarchitectural mechanisms (e.g., the out-of-order issue logic). However, this situation does not always match the design constraints since processors must support a growing number of applications. Thus, many research

works have concentrated on attacking hot spots [5] or reducing consumption in the larger microprocessor components like the cache memory [6]. A different power management technique which applies on the whole microprocessor die and that is being implemented in most current microprocessors is *Dynamic Voltage and Frequency Scaling* (DVFS) [2]. It applies on the whole microprocessor and allows the system to improve its energy consumption by reducing the frequency when the processor has a low activity level (e.g., a mobile phone that is not being actively used). This technique allows the system to work at different frequency/voltage levels, and it has been implemented in many modern microprocessors like the Transmeta Crusoe [7], the Intel Xeon [8], or the Mobile AMD Duron [9].

Depending on either each core has its own DVFS regulator or the same regulator is used to supply voltage to all cores (forcing them to work at the same speed), DVFS can be classified as local and global, respectively. Local DVFS is more complex and expensive because more frequency/voltage regulators are required in the power delivery network. In this way, many multicore processors use a single clock domain since using independent clock domains is much more expensive, as stated in [1]. In addition, it has been shown that if the workload is properly balanced among the cores, global DVFS can be as efficient as local DVFS [10]. Thus, there are several high-performance processors, such as the IBM Power 7 [11], that implement global DVFS. However, recent manycore processors are incorporating multiple frequency/voltage domains, where each domain applies to a subset of the processor cores.

1.1.2 Real-Time Task Partitioners and Schedulers

The problem of power aggravates with the increasing number of cores in the processors die [12], which is the current trend. In such processors, where different computational units are implemented, real-time systems include a workload partitioner in charge of distributing the task set among the available computational units according to a given heuristic [13, 14]. Typically, these heuristics only addressed the workload partitioning in the past; nevertheless, and due to energy concerns, power-aware heuristics have been recently proposed. In this context, several studies have analyzed the energy consumption by revisiting previous heuristics like the *Worst Fit* (WF) or *Best Fit* (FF) [15, 16].

Unfortunately, the nature of some workload mixes prevents the partitioner from achieving a good balance. To deal with this drawback some systems allow task migration in order to move their execution from one core to another, which results in energy saving improvements. Note that nowadays an *Operating System* (OS) is typically used to govern any embedded system (even in the form of a micro/pico kernel), however this study is abstracting the workload as a set of periodic tasks.

Regarding real-time constraints, the current embedded market requires these systems to host both *Hard Real-Time* (HRT) and *Soft Real-Time* (SRT) tasks, as well as tasks without timing constraints (i.e., *Non Real-Time* or NRT tasks). Unlike HRT environments where task deadlines must be guaranteed, in SRT environments this constraint is not mandatory, that is, deadline misses can be tolerated [17, 18]. Since in embedded systems HRT tasks are used to model critical applications due to safety and correctness reasons, SRT and NRT tasks are usually assigned the lowest priority. Nevertheless, a key requirement for market penetration of commodity embedded systems such as smartphones or tablets is to ensure not only a correct operation, but also the highest performance for SRT applications such as video streaming.

1.1.3 Memory Controller Scheduling Policies

Besides, due to packaging restrictions, which are often accentuated in embedded systems, memory controllers are shared among a set of its computing cores. In this context, the cores sharing the same memory controller are said to belong to the same memory domain. Applications executing in the same memory domain may suffer contention when accessing the shared memory controller, which leads to unpredictable behavior. Thus, memory controller policies must be specially designed to tackle this issue.

To avoid unpredictable behavior in the execution of memory requests from critical applications, conventional memory controller scheduling policies used in NRT systems (e.g., FIFO) are not appropriate, since these policies can introduce a wide variability in the latencies of the memory requests, which is likely to cause HRT deadline misses. Recall that a deadline miss in HRT could lead to an important damage of the real-time system.

1.1.4 Dynamic Execution Time Estimation

In addition, real-time systems require to estimate the *Worst Case Execution Time* (WCET) of the applications they run to ensure the system schedulability. The WCET must be estimated with the highest accuracy in order to provide either high *Quality of Service* (QoS) in SRT systems or to prevent possible damages due to deadline misses in HRT systems. Also, a high estimation accuracy allows the system to save power, improve the schedulability, or both.

As commented above, there are severals aspects difficulting the estimation of the execution time of tasks in these processors. First, the running threads compete for shared resources such as the memory controller. Second, the aforementioned DVFS technique, aimed at managing power consumption more efficiently, difficults the execution time estimations, since it must be taken into account the range of frequencies that the regulator supports.

Typically, estimations of the WCET can be performed off-line by running the jobs in an isolated way. However, advances in processor design make this technique not appropriate for its use in most current multithreaded microprocessor generations. In addition, some research works assume that the memory access time (quantified in processor cycles) is constant regardless the processor frequency. This assumption considers that all the processor components scale their speed at the same pace. Nevertheless, it can bring important deviations to the estimation of the execution time since main memory devices have their own power supply and work independently of the processor DVFS regulator.

1.2 Contributions of the Thesis

The goal of this thesis is to devise multiple techniques to reduce power consumption in a multicore embedded real-time system. This thesis offers four major contributions that are enclosed with dotted lines in Figure 1.1: i) task partitioning, ii) task migration, iii) memory request scheduling policies, and iv) dynamic execution time estimation. These contributions are described below:

1. **Task Partitioning**.

   A partitioning heuristic aimed at increasing the overlapping time between memory

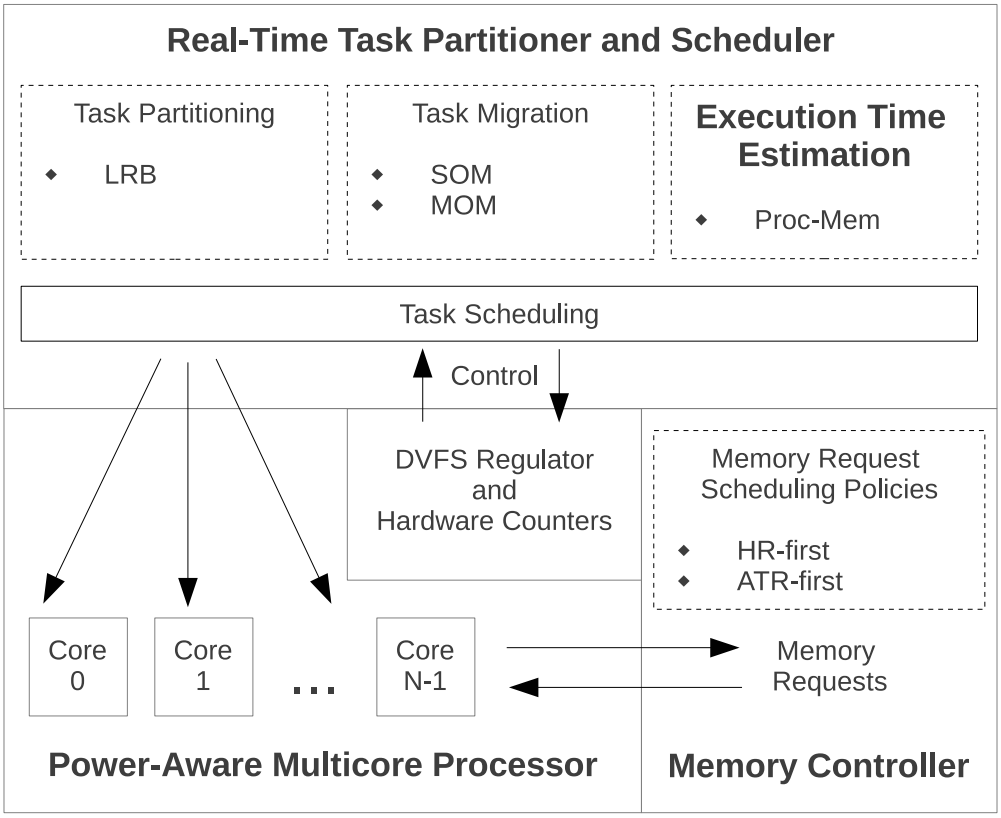


FIGURE 1.1: Components of the studied system and thesis contributions

access and computation time is proposed. This algorithm distributes tasks, both HRT and SRT, among cores balancing the demand of a major system component (processor or memory) until a given utilization threshold is reached. Then, as all cores are assumed to work at the same speed (global DVFS), the less loaded core policy is followed to better balance speed requirements. In this context, there is a trade-off between the system frequency and the percentage of deadline misses. In other words, the lower the frequency, the less the energy consumption but also a higher number of deadline misses will rise and viceversa. The proposed algorithm, referred to as *Load-bounded Resource Balancing* (*LRB*), addresses this trade-off by varying the system speed, and uses a threshold as an indicator of acceptable deadline misses. To this end, different aggressiveness ways to increase and decrease the frequency have been devised. This technique varies the frequency to reduce power consumption while guaranteeing HRT deadlines and allowing the miss of some SRT deadlines.

2. **Task Migration**.

   Two algorithms allowing task migration are proposed. The simpler algorithm,

namely *Single Option Migration* (*SOM*), checks just one target core before performing a migration. In contrast, the *Multiple Option Migration* (*MOM*) searches the optimal target core. To address energy savings, the devised schedulers follow two main rules: (i) migrations are allowed only in those points of time when the workload changes, that is, when tasks enter or leave the system, and (ii) only one task is allowed to migrate each time because analyzing all the possible task migrations may result in a prohibitive overhead. Thes partitioner module is in charge of readjusting possible workload imbalances at run-time that may occur at arrivals or exits of tasks by applying task migration. To keep overhead low and to study the impact of the point of time when the algorithm is applied, three variants of the *SOM* algorithm have been devised, depending on the point of time the scheduler is applied: when a task arrives to the system (SOM_{in}), when a task leaves the system (SOM_{out}), and in both cases (SOM_{in-out}).

3. **Memory Controller Scheduling Policies**.

   A simple memory controller scheduling policy would prioritize requests of HRT tasks over the remaining ones, from now on *HR-first*. This would increase the number of deadline misses of SRT tasks and the execution time of NRT tasks, negatively impacting on the QoS of SRT and non-critical applications. Moreover, a high number of SRT deadline misses during HRT peak activity may leave the system unresponsive for significant periods of time. To overcome this problem, a novel memory controller policy, referred to as *ATR-first*, is proposed. This policy ensures *Earliest Deadline First* (EDF) schedulability without sacrificing QoS of non-critical applications. The proposed approach prioritizes only those requests of HRT tasks that are critical to accomplish real-time schedulability. For dynamic scheduling algorithms, such as EDF, this set of tasks varies during execution and depends on the running workload as well as the computing power of the system (i.e., number of cores and multithreading capabilities of each core).

4. **Dynamic Execution Time Estimation**.

   A model that dynamically predicts the execution time of real-time applications is proposed. The proposal, referred to as *Processor-Memory* (*Proc-Mem*), executes all the workload at the maximum speed during the first hyperperiod, which is used to collect the required inputs to the model. When this hyperperiod expires, the

model is used to estimate the lowest (i.e., the most energy-efficient) working frequency that fulfills the deadline through the different applications' periods. These input values are saved and used by the scheduler for subsequent hyperperiods. Notice that selecting the optimal frequency not only can bring important energy savings but also improves system schedulability. Besides, a *Frequency Selection Policy* (*FSP*) based on the model is also devised and compared with the *Constant Memory Access Time* (*CMAT*) model, where the memory access time (quantified in processor cycles) is constant regardless the processor frequency.

1.3 Thesis Outline

This thesis is structured in 8 chapters. Chapter 2 discusses the related research. Chapter 3 describes the baseline system model and presents the simulation framework used for the experiments.. Chapter 4 introduces the *LRB* task partitioning algorithm. Chapter 5 explains both *SOM* and *MOM* task migration algorithms. Chapter 6 analyzes the proposed memory controller policies *HR-first* and *ATR-first*. Chapter 7 describes the *Proc-Mem* model to predict the execution time of real-time tasks. Finally, Chapter 8 summarizes the thesis and presents some concluding remarks.

Chapter 2

Related Work

This chapter presents the related work with the thesis contributions as described in Chapter 1.

2.1 Real-Time Scheduling in Power-Aware Multicore Processors

Numerous research papers have explored energy management on uniprocessor real-time systems. Most of them focus on periodic task systems [19, 20], but also some proposals have been published dealing with soft aperiodic tasks [21] and sporadic tasks [22]. Buttazzo et al. [19] present an algorithm for energy management based on DVFS that integrates the elastic scheduler for discrete voltage mode processors. In [20], AlEnawy et al. analyze the performance optimization problems for real-time systems that have to rely on a fixed energy budget during an operation. They adopt the weakly-hard real-time scheduling paradigm to ensure a predictable performance for all the tasks; that is, they minimize the number of dynamic failures (in terms of (m,k)-firm deadline constraints) while remaining within the energy budget.

To deal with both computational and power management requirements, many systems use multicore processors. These processors allow a more efficient power management than complex monolithic processors for a given performance level. In multiprocessors platforms, where energy efficient scheduling of real-time tasks is an NP-Hard problem [1, 15], two main scheduling categories exist depending on whether the task queue is

shared among all the processors (global scheduling) or not (partitioned scheduling). In the former case, tasks are allowed to migrate among processors and the highest priority executable tasks are selected for execution. For instance, in [23], Kato et al. present a global EDF based scheduler [24] for sporadic tasks. In the latter case, a task is assigned permanently to a given processor and it is not allowed to migrate. When using this technique the multiprocessor scheduling problem is split into several uniprocessor scheduling problems. Therefore, well established algorithms such as EDF and RMS (*Rate Monotonic Scheduling*) from uniprocessor theory can been adopted. For example, in [16], AlEnawy et al. consider the problem of energy minimization for periodic preemptive HRT tasks scheduled on a *Symmetric Multiprocessor Platform* (SMP) with DVFS capability. In [25] Zikos et al. present an energy aware algorithm for clusters of heterogeneous processors, based on favoring job allocation to the most energy efficient processors. Notice that these works do not model multithreaded processors, which are standard nowadays.

Many manufacturers (e.g., Intel, IBM, Sun, etc.) deliver processors providing multithreading capabilities, that is, they provide support to run several threads simultaneously. Some examples of current multithreaded processors are Intel Montecito [26] and IBM Power 5 [27]. Also, leading manufacturers of the embedded sector, like ARM, plan to include multithreading technology in next-generation processors [28]. Regarding research in these multithreaded systems, in [29], Park et al. provide a performance guaranteed energy management for multithreaded applications in multicore processors. In [30] Cazorla et al. present an architecture where a *Simultaneous MultiThreading* (SMT) processor interacts with the OS to improve real-time predictability. However, none of these works consider real-time schedulability and energy management as a whole. In [31], El-Haj-Mahmoud et al. state that SMT processors hardly provide HRT guarantees, and proposes virtualizing them into multiple single-threaded superscalar processors. This architecture incorporates a static scheduler to execute periodic tasks. They compare their scheduler to a multiprocessor implementing EDF in terms of rate of deadline misses. Nevertheless, this work does not tackle energy consumption.

2.1.1 Task Partitioning

With respect to partitioning heuristics, many proposals have recently been published. In [32] Wei et al. exploit parallelism of multimedia tasks on a multicore platform combining DVFS with switching-off cores to reduce energy consumption. In [15], Aydin et al. develop a framework where load balancing is used to produce energy-efficient partitionings of real-time tasks in a SMP system with DVFS. These partitionings are evaluated using the EDF algorithm. Schranzhofer et al. [33] presented a method for allocating tasks to cores in a multiprocessor platform, aimed at minimizing the average power consumption, however, the application is modeled without considering timing constraints. In [16], authors focus on the impact of heuristics on feasibility and energy consumption, and propose a new one that reserves a subset of processors for light tasks. In [34] Brandenburg et al. present an empirical evaluation of several global and partitioned scheduling algorithms. The evaluation was conducted on a Sun Niagara multicore platform with 32 logical CPUs (eight cores and four hardware threads per core). Although each tested algorithm proved to be a feasible choice for some subset of the considered workload categories, they do not take into account power saving aspects. Finally, concerning memory-processor overlapping, El-Haj-Mahmoud et al. [35] devise a technique for tolerating memory latencies in HRT systems implemented in *Coarse-Grain MultiThreading* (CGMT) processors. They derive a closed-form schedulability test to determine whether a HRT task set is schedulable in the context of *Weighted Round-Robin* (WRR) scheduling on a multithreaded processor. Unfortunately, this work does not deal with energy management and multiple cores.

2.1.2 Task Migration

Some proposals dealing with task migration can be found in the bibliography. Brandenburg et al. [34] evaluate global and partitioned scheduling algorithms in terms of scalability, although power consumption was not investigated. In [36], Zheng divides tasks into fixed and migration tasks, allocating each of the latter category to two cores, so they can migrate from one to another. Unlike this thesis, that paper does not consider dynamic workload changes, instead, all tasks are assumed to reach the system at the same instant, so migrations can be scheduled off-line. In [37] Brião et al. analyze how migrating SRT tasks affects NoC-based MPSoCs in terms of deadline misses and

energy consumption for non-threaded architectures. Seo et al. [1] present a dynamic repartitioning algorithm with migration to balance the workload and reduce power consumption. They perform a theoretical exploration assuming parameters like number of cores and number of tasks, but neither computational core nor real-time benchmarks are used through their evaluation. Thus, their main contribution is the theoretical estimation of benefits. Fisher and Baruah [38] derived near-optimal sufficient tests for determining whether a given collection of jobs with precedence constraints can feasibly meet all deadlines upon a specified multiprocessor platform allowing task migration under global EDF scheduling. However, this work does not tackle reduction of energy consumption, which is major concern of this research. Notice that in multicores, task migration may cause additional traffic for the coherence protocol too; this problem has been studied previously and reduced by modification to the coherence scheme [39].

2.2 Memory Controller Scheduling Policies

Schliecker et al. [40] and Pellizzoni et al. [41] analyze the delay of memory access in systems where several simultaneously-running tasks share the main memory. Their proposals require a detailed profiling of application memory access patterns and a deep understanding of the memory scheduling policy.

Predator [42] is a memory controller for multiprocessors that guarantees a bandwidth requirement to a given task and requires the user to assign a fixed priority to each task. The controller is implemented in the network interface of a network on chip targeted for a specific DRAM device, a JEDEC-compliant 32 Mb 16 DDR2-400B SDRAM. This solution fits well in streaming or multimedia real-time applications, in which a bandwidth QoS requirement can be easily defined.

In [43], Paolieri et al. present a JEDEC-compliant DDRx SDRAM analyzable memory controller for multicore architectures that reduces the impact that a memory request can suffer due to the memory interferences introduced by other tasks, allowing the computation of tight WCET estimations. This research is orthogonal to the memory controller scheduling policy proposed on this thesis, that improves QoS of SRT tasks in a heavily loaded real-time system.

In [44], a shared memory multicore that allows concurrent execution of HRT and non-HRT applications is proposed. They give an Upper Bound Delay to memory requests of HRT tasks, so that these tasks can meet their deadlines while providing high-performance for non-HRT tasks. However, the power consumption of the proposal is not tackled. To the best of our knowledge, our work addresses for the first time the trade-off between the energy consumption and SRT QoS while guaranteeing HRT constraints.

2.3 Execution Time Estimation

A number of static WCET analysis methods have been designed in the last two decades, mainly for monoprocessors [45]. These research works use execution time estimates to choose the optimal DVFS level to enhance power savings and reduce deadline misses. Seth et al. [46] study the effects of DVFS on static timing analysis taking into account power consumption. They calculate the execution time in any frequency range using a parametric model that depends on the number of cache misses. However, their approach is static and needs the source files of the applications. Another model is proposed by Snowdon et al. [47]. They perform an on-line evaluation of application characteristics using performance counters, nevertheless, it is combined with an off-line characterization of the hardware platform. Miftakhutdinov et al [48] propose a DVFS performance predictor for memory systems with a streaming prefetcher. They also take into account that memory latencies (as measured in seconds) are not affected by frequency scaling. However, their proposal is static, without real-time constraints and does not consider multiple cores. Unlike these works, our proposal dynamically estimates the execution time and focuses on multicore processors.

Some work has dealt with the estimation of the execution time of concurrent tasks. Several studies focus on timing analysis for single-threaded architectures by using static code analysis. Schaefer et al. [49] propose to measure execution time at basic block level and using this data to estimate WCET of the entire program. Wenzel et al. [50] propose a decomposition of the program into segments performing a timing analysis for each segment. Authors also propose an approach for program segmentation that balances the number of program segments with the average number of paths per segment. These studies propose improvements of measurement-based timing techniques for single-threaded processors.

Regarding multithreading, Cullmann et al. [51] analyze the design of future multithreaded processors for time-critical systems. They show that some processor designs make the timing analysis infeasible and suggest design principles for making multithreaded architectures predictable. Based on the theoretical analysis, authors give guidelines for designing predictable architectures.

Finally, in [52], Radojković et al. propose a method that quantifies the slowdown that concurrent tasks may experience due to contention in shared processor resources. The presented method is used to determine if a given multithreaded processor is a good candidate for systems with timing requirements. They use the method to analyze three multithreaded architectures exhibiting different configurations of resource sharing. However, they do not tackle the schedulability analysis of the system.

Execution time estimation for multicore platforms has been the subject of rather few studies. Most of them focused on cache-aware methods. Hardy et al. [53] present a WCET estimation method for multicore platforms with shared instruction caches. The proposed method provides estimates through the control of the contents of the shared instruction caches. More precisely, by caching only the blocks statically known to be reused and bypassing from shared caches the other blocks.

There are some works that study the effect of the main memory in the estimation of the WCET for multithreaded and multicore architectures. Shah et al. [54] make use of bank interleaving and applying *Priority based Budget Scheduling* (PBS) to share SDRAM among multiple masters. This technique permits to bound the WCET of an application accessing a shared SDRAM using the worst case access pattern. They implemented the memory system in an FPGA. Their proposal produces safe and low WCET bounds. Ungerer et al. [55] build a predictable multicore architecture for mixed critical applications. Predictability is achieved by giving the highest priority to the HRT tasks while on the shared bus access latency is bounded by a *Round Robin* (RR) scheduling policy. The memory is accessed through an *Analyzable Memory Controller* (AMC), which implements bank interleaving. Through theoretical analysis, latency parameters are extracted to calculate the WCET. The AMC applies the maximum of Read/Write and Write/Read switching latencies as a constant worst case latency on every access. Such assumption, while making the analysis simple, cannot produce precise bounds. Moreover, the RR policy with one request per master cannot satisfy the need

of different bandwidth requirements. If more than one request per master is assigned, the WCET is severely degraded [54].

Chapter 3

System Model

This chapter presents the baseline multicore system used in this thesis. We also detail the extensions that have been carried out to the Multi2Sim [56] simulation framework to model real-time constraints.

3.1 Baseline Design

The multicore system studied in this thesis consists of a source and a sink of real-time tasks, a workload partitioner, a power-aware scheduler, and a multicore processor with m cores. Figure 3.1 depics a block diagram of the modeled system.

When a task arrives to the system, a partitioner module allocates it into a task queue associated to a given core, which contains the tasks that are ready for execution in that core. These queues are components of the power-aware scheduler that controls a global DVFS regulator [2]. In this scheme, the scheduler is in charge of adjusting the working frequency of the cores in order to satisfy the workload requirements.

The system consists of multiple superscalar cores that can issue more than one instruction to execution every cycle. Besides, due to energy constraints of embedded systems, an in-order issue logic has been assumed, as deployed in embedded processors like the Intel Atom [57]. Table 3.1 summarizes the simulated machine architectural parameters of each core.

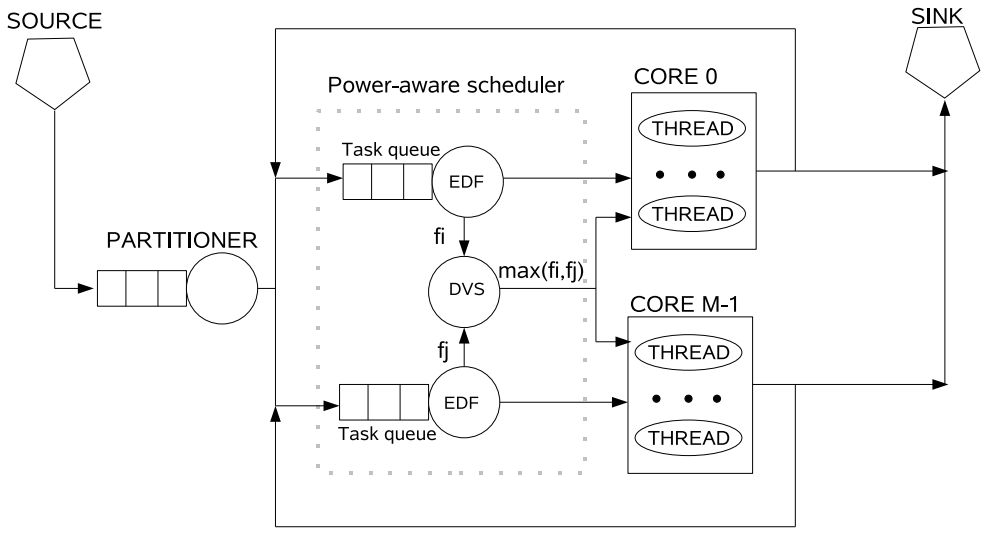


FIGURE 3.1: Modeled system.

Many commercial systems are implementing multithreading capabilities in multicore processors. To this end, three main paradigms can be used: *Simultaneous Multi-Threading* (SMT), *Coarse-Grain MultiThreading* (CGMT) and *Fine-Grain MultiThreading* (FGMT) [58]. For instance, Intel's Montecito [26] multicore processor implements CGMT while Sun's UltraSPARC T2 [59] uses FGMT. Several SMT implementations also exist in commercial processors, like the IBM Power 5 [27] or the Intel Atom [60].

SMT processors are more complex and less predictable. Because of these reasons, this work focuses on CGMT which offers a good trade-off between power consumption and real-time schedulability [35]. CGMT processors provide multithreading capabilities by switching the running thread when a long latency event occurs (e.g., a memory access). In such a case, a new thread takes the processor control while the other performs the memory access, so during the long latency event both threads are allowed to overlap their execution. In this context, the saved time can be devoted to reduce frequency in order to save energy. If the new thread also stalls due to a long latency memory event, then the issue slots are temporarily reassigned to the highest priority thread among those that are not waiting for memory, until the event is resolved. This thread switch can occur among the available hardware threads. That is, the issue slots are always assigned to the hardware thread executing the task with the highest real-time priority, which varies according to the scheduling algorithm.

Microprocessor core	
Issue policy	In order
Fetch kind	Switch on event
Branch Prediction	Two-level global history 256 entries BTB, 4096 2-bit saturating counters GHB
Issue bandwidth	2 instructions/cycle
# Int ALUs, mult/div	2,1
# FP ALUs, mult/div	2,1

TABLE 3.1: Machine parameters.

3.1.1 Real-Time Tasks

The system workload executes periodic real-time tasks. There is no task dependency and each task has its own period of computation. A task can be launched to execute at the beginning of each active period and it must end its execution before reaching its deadline. The end of the period and the deadline of a task are assumed to be the same for a more tractable scheduling process. There are also some periods where tasks do not execute since they are not active (i.e., inactive periods). In short, a task arrives to the system, executes during several active periods repeatedly, leaves the system, remains out of the system for some inactive periods, and then it enters the system again. This sequence of consecutive active and inactive periods allows modeling real systems with mode changes. Table 3.2 shows the benchmarks from WCET Analysis Project [61] that have been used to prepare real-time workload mixes.

Besides its period and deadline, a task is also characterized by its WCET. The WCET is obtained by executing the task in isolation. To avoid WCET variability due to concurrent execution, we assume that the processor and the memory controller support priorities.

The task utilization is obtained as $U = \frac{WCET}{Period}$ and it is used by several schedulers and partitioners to check whether schedulability of the task set is feasible or not.

3.1.2 Power-Aware Scheduler

The power-aware scheduler is composed of an EDF scheduler per core and the global DVFS regulator [10]. Once a task is allocated to a core, it is inserted into the task queue of that core. These queues are ordered according to the EDF policy, which prioritizes

Name	Function Description
Adpcm	Adaptive pulse code modulation algorithm
Bitcnts	Test program for bit counting functions
Bs	Binary search for a 15-element array
Bsort100	Bubblesort program
Cnt	Counts non-negative numbers in a matrix
Compress	Data compression program
Cover	Program for testing many paths
Crc	Cyclic redundancy check on 40-byte data
Duff	Copy 43-byte array
Edn	FIR filter calculations
Expint	Series expansion for integral function
Fac	Factorial of a number
Fdct	Fast Discrete Cosine Transform
Fft1	1024-point Fast Fourier Transform
Fibcall	Simple iterative Fibonacci calculation
Fir	Finite impulse response filter
Insertsort	Insertion sort on a reversed array of size 10
Janne_complex	Nested loop program
Jfdctint	Discrete-cosine transformation
Lcdnum	Read ten values, output half to LCD
Lms	LMS adaptive signal enhancement
Loop3	Function with diverse loops
Ludcmp	LU decomposition algorithm
Matmult	Matrix multiplication of two 20x20 matrices
Minmax	Minimum and maximum functions
Minver	Inversion of floating point matrix
Ndes	Complex embedded code
Ns	Search in a multi-dimensional array
Nsichneu	Simulate an extended Petri Net
Prime	Calculates whether numbers are prime
Qsort-exam	Non-recursive version of quick sort algorithm
Qurt	Root computation of quadratic equations
Select	Nth largest number in a floating point array
Sqrt	Square root function
Statemate	Automatically generated code
Ud	Calculation of matrixes

Table 3.2: Benchmark description.

the tasks whose deadlines expire earlier. Thus, the tasks with the closest deadlines will be the ones mapped into the hardware threads implemented in each core. If some thread context is occupied by a lower priority task, preemption is applied.

Once the control logic of the global DVFS receives the speed requirements from all the EDF schedulers in the system, it supplies to the cores an appropriate frequency/voltage level to fulfill these requirements. The target frequencies are recalculated only when

F[MHz]	1700	1500	1400	1300	1200	1100	900	600
V[Volts]	1.48	1.48	1.48	1.39	1.18	1.18	1	0.96
P[Watts]	24.5	24.5	22	22	12	12	7	6

TABLE 3.3: Frequency (F), voltage (V) and power (P) for each DVFS level.

the workload changes, that is, when a task arrives to and/or leaves the system. In the former case, a higher speed can be required because the workload increases. In the latter, it could happen that a lower frequency could satisfy the deadline requirements of the remaining tasks.

Table 3.3 shows the different frequency/voltage values considered for the power-aware scheduler based on the frequency levels of a Pentium M [62]. Frequency changes are not considered instantaneous since some time is needed to overcome the voltage gap between two different DVFS levels. In this sense, the latency of changing the DVFS level has been modeled according to a voltage transition rate of $1mv/1\mu s$ [63]. To model this latency and the power overhead caused by these changes, the worst case for that transition has been assumed. That is, during a frequency transition the speed of the lowest frequency and the power consumption of the highest one are considered.

To calculate the energy consumption, the number of cycles working at each frequency is multiplied by the energy required per cycle at that frequency. Then, this value is normalized using as baseline the energy consumed by the system working always at the maximum speed. Notice that it is important to reduce the time spent at the higher frequency/voltage levels, since the normalized power for a given technology (e.g., 45nm) grows exponentially with the processor clock [1]. Thus, the scheduler must concentrate on the most efficient frequency/voltage levels.

3.2 The Multi2Sim Simulation Framework

Multi2Sim [56] is a cycle-by-cycle execution driven simulation framework for evaluating processors, which is being used by manufacturer companies like AMD or NVIDIA. As in commercial processors, three main parts can be distinguished: the core, the cache hierarchy and the interconnection network. Multithreaded cores can be modeled on this simulator with three multithreading paradigms: FGMT, CGMT and SMT. Distinct

configurations are allowed to represent different sharing strategies of pipeline stages and resources. It models different memory hierarchies with diverse sharing strategies and cache levels among cores and threads. A cache coherence protocol (MOESI) for sharing data among cores is also implemented. Finally, several interconnection network topologies can be configured.

3.2.1 Simulation Models

In Multi2Sim three different simulation techniques are used: *Functional Simulation*, *Detailed Simulation* and *Event-Driven Simulation* (timing simulation is performed by the latter two). *Functional Simulation* is implemented as an autonomous module that provides an interface to the rest of the simulator. It does not consider any hardware structure, as cores or threads, it just deals with software contexts. Its main functions are to create and destroy contexts, perform program loading, enumerate existing contexts and consult their status, execute machine instructions and handle speculative execution. In *Detailed Simulation* the specific hardware microarchitecture is taken into account, with elements as pipeline structures (stage resources, instruction queues, reorder buffer, etc), branch predictor, cache memories or segmented functional units. Each cycle the detailed simulation module uses the functional simulation module interface to update the context status. The *Detailed Simulation* module analyzes the recently executed instructions accounting the operation latencies caused by hardware structures. With functional and detailed simulation built in independent modules, the implementation of machine instructions behaviour can be centralized in a single file (functional simulation), while function calls that activate hardware components (detailed simulation) return the latency required to be completed. In some situations that latency cannot be calculated when the function is called, it needs to be simulated cycle by cycle. That is the case of the interconnection network and cache memories. In that situation an *Event-Driven Simulation* module is required to obtain delays of message transfers caused by memory accesses.

3.2.2 Main Proposed Extensions

In order to make Multi2Sim able to support real-time tasks and, at the same time, model a power-aware system, many extensions have been implemented in an additional

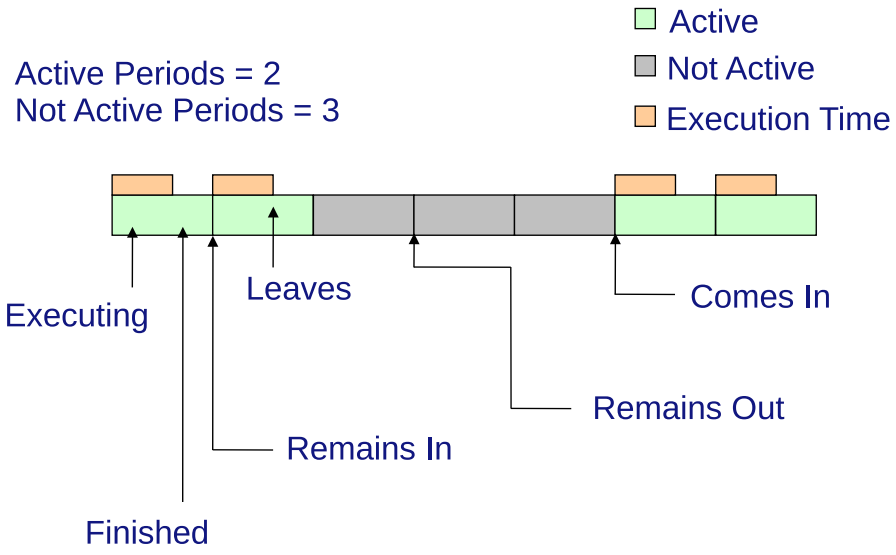


FIGURE 3.2: Active and inactive periods example.

module. This power-aware real-time module is in charge of: i) manage periodic tasks repetition with its alternate active and inactive periods; ii) create a deadline based task priority system; iii) support processor frequencies; iv) model penalty latency cycles when frequency changes; and v) provide a task partitioner to distribute the workload among cores. These assignments can be seen as sub-modules, and a more detailed description of each one is presented below.

3.2.2.1 Task Repetition

In the original Multi2Sim, tasks (benchmarks) are just executed once. In contrast, in real-time systems tasks are entering and leaving the system constantly, alternating a number of consecutive (periodic task repetition) active and inactive periods, until the end of the simulation. When a task finishes, it can be scheduled for another execution period or leave the system (active-inactive transition). In the latter case, the task will remain out of the system for some consecutive periods, and then it will enter it again (inactive-active transition). Figure 3.2 shows an example of a task with 2 active and 3 inactive periods. In this submodule, the chosen approach to model periodic task repetition is repeating the program loading process. When the last instruction of a task finishes its commit stage, that task is removed, and immediately it is reloaded again. This includes creating context, restoring data, arguments, environment variables, etc.

3.2.2.2 Priority

In order to model a more realistic system, it is also necessary to increase the number of tasks that the simulator is able to schedule because the original Multi2Sim only accepts as many tasks as the total number of hardware threads in the system, that is, the number of hardware threads per core multiplied by the number of cores. This change implies implementing a task queue for each core, so active tasks can wait for a chance to use the processor when a task that is running finishes. These are priority queues, based on the EDF algorithm, although other algorithms can be also applied. In this context, tasks priorities regarding EDF are taken into account. As *Coarse-Grain* is the assumed multithreading paradigm, thread switches occur when a long latency event appears (e.g., main memory accesses). On the other hand, we do not consdier the *timeslice* thread switch implemented in real CGMT processors.

In this way, at the beginning of the simulation the highest priority tasks are launched to execution. A task switch caused by a long latency event from the highest priority thread enables the highest priority active task among the remaining ones to use the processor. When the long latency event is resolved, preemption is applied to allow the highest priority thread that was stalled to continue execution. In short, the processor must be occupied by the task with the highest priority not stalled by a long latency event. Moreover, preemption must be also applied when a higher priority task becomes active and all the threads of the core are occupied. In that case, the lowest priority task among the executing ones will be replaced by the new one. This expulsion requires saving the execution state of the replaced task, so it can be reloaded from that point later when any mapped task finishes. The aim is to ensure that, the tasks with the closest deadlines will run regardless of when they arrived to the system.

Unlike the other extensions proposed, priority thread switching requires to alter the original Mult2Sim pipeline because the instruction fetch stage needs to be modified so task deadlines are considered in the decision of whether to change the current thread or not, since it is in that stage where thread switching occurs.

3.2.2.3 Frequency

In Multi2Sim there is not a model of the processor clock frequency. However, the number of latency cycles requested for a main memory access is a configurable parameter. Therefore, we use the relative speed between CPU and memory is used to model different processor frequency levels. In this way, a faster processor will stall during more processor cycles than a slower one for a given memory latency. That is, for each processor cycle, there will be potentially more memory events serviced in the slower processor than in the faster one. Once the system is able to run tasks with different frequencies, the assumed global DVFS controls system speed depending on the workload requirements. Note that the frequency will only be recalculated when a workload change occurs, that is, when a task enters or leaves the system to perform a consecutive sequence of active or inactive periods, respectively.

3.2.2.4 Latency of Frequency Changes

Frequency changes are not instantaneous, some time is needed to overcome the voltage difference between the previous frequency and the new one. Besides, another consideration is the power overhead caused by these changes. In real processors frequency changes are gradual, but in this case it is modeled a simpler approach. Only the old and the new frequencies are considered, with a latency depending on the worst case. If a frequency increase is required, the system will remain at the same speed during some penalty cycles before actually changing the working frequency. On a decrease request this sequence is reversed, that is, first the working frequency is decreased; after that the penalty cycles are accounted. In both cases, the number of penalty cycles will be proportional to the voltage difference between the two frequencies.

If the system is able to run tasks at more than two different frequencies, then it is possible that a frequency change implies passing through some intermediate frequency levels. In that case we repeat the aforementioned sequence for each two pair of intermediate frequency levels. For example, in a system with 3 frequency levels it could be requested a change of the working frequency from the maximum to the minimum one, and that will be modeled decreasing the frequency from the maximum speed to the intermediate one, waiting the corresponding penalty cycles, and then arriving finally to the minimum

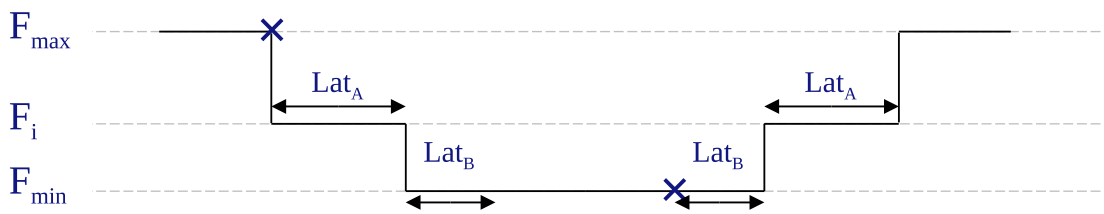


FIGURE 3.3: Decreasing and increasing frequency.

frequency, followed by some additional penalty cycles. The left part of Figure 3.3 represents the frequency transition commented above. In addition, the right part of the figure represents the transition from the minimum to the maximum frequency.

Frequency changes cause more power consumption. To model this power overhead it is assumed that during a frequency transition power consumption is the one corresponding to the highest frequency among the ones involved in the change, independently of whether it is an increase or a decrease. That is, the worst case is assumed for every frequency change. Note that we assume the worst case for both speed and power consumption. Regarding speed, the worst case means that the system works at the lowest frequency during the transition, while regarding power it means that the system consumes the same power that the highest frequency would consume during the whole transition.

3.2.2.5 Task Partitioner

The last extension is a task partitioner module. This component is in charge of allocating all the tasks to the available cores as they arrive at the system. This distribution is a major concern since it will have a very important influence on the system efficiency, in terms of power consumption, tasks execution time and schedulability. This is because many different partitioning algorithms can be applied, and the workload balance among the cores will depend on the selected algorithm. Good algorithms will be those that can better balance workload so the system can work at lower frequencies reducing power consumption. For instance, if the system has two cores, a hypothetical balanced workload could present 0.5 utilization for each core, while another algorithm could allocate tasks to cores causing a 0.2 utilization for one core and 0.8 for the other one. The former algorithm would be better than the latter, since that task distribution could allow the

system to reduce power consumption by working at a global lower speed, while in the second algorithm the system may not be able to work at a low frequency because that would imply missing deadlines in the most loaded core.

Chapter 4

Task Partitioning

4.1 Introduction

This chapter presents a partitioning heuristic, referred to as *Load-bounded Resource Balancing* (*LRB*), aimed at increasing the overlapping time between memory access time and computation time; thus reducing energy consumption in CGMT multicore processors working on a global DVFS regulator. The algorithm distributes tasks among cores balancing the demand of a given system component (core or memory) until a given utilization threshold is reached. Then, as all cores work at the same speed (global DVFS), the less loaded core policy is followed to balance speed requirements.

Both HRT and SRT tasks are considered, since most current embedded systems run both kinds of applications, e.g., avionics and robotics systems. Unlike HRT environments where task deadlines must be guaranteed, in SRT environments this constraint is not mandatory, that is, deadline misses can be tolerated [17, 18]. In this context, there is a trade-off between the system frequency and the percentage of deadline misses. In other words, the lower the frequency, the less the energy consumption but also a higher number of deadline misses will rise and viceversa. The proposed algorithm addresses this trade-off by varying the system speed, and uses a threshold as an indicator of acceptable deadline misses. To this end, different ways with distinct aggressiveness have been devised to increase and decrease the frequency. To sum up, the proposed algorithm varies the frequency to reduce power consumption while guaranteeing HRT deadlines and allowing the miss of some SRT deadlines.

4.2 Partitioning Heuristics

As mentioned above, CGMT processors switch the running thread when a long latency event rises. This feature allows overlapping the execution of diverse tasks, provided that one of them is consuming CPU while the other ones are accessing to main memory or waiting for it. In a real-time system, the consequence of this overlapping is that non priority tasks are allowed to either execute processor instructions or access to main memory. Thus, they can finish earlier their execution than when executing in a non-threaded processor. This property allows the system to get slack time that can be used to reduce the processor speed and save energy. Note that this slack time can be enlarged when the co-running task threads are complementary regarding processor and memory requirements. In short, the main goal of the proposed heuristic is to increase the processor-memory overlapping time. With this aim, the proposed heuristic distributes complementary tasks among the available cores.

However, distributing tasks in this way regardless the individual core utilizations could lead to an unfair load distribution. That is, one of the cores could become much loaded than the others. In this situation, it would be not feasible to globally reduce the consumption by lowering the system speed, since the accumulated utilization of one core would force to execute its workload at a higher speed in order to guarantee the deadlines of the tasks assigned to that core. Therefore, to avoid this situation the heuristic should first provide a good resource (CPU or memory) balancing among cores in order to increase overlapping, while bounding the utilization of the individual cores.

4.2.1 HRT Heuristic

The proposed *LRB* policy, pursues to balance a given resource utilization. Two different variants have been devised depending on the balanced resource (memory and CPU). The variant balancing memory will be referred to as *LRB-M*, and that balancing CPU as *LRB-C*. Figure 4.1 summarizes the *LRB* policy. The algorithm distributes tasks among cores by assigning the most resource-consuming task to the core with least accumulated consumption for the resource being balanced. Then, it updates the task set to be distributed, as well as the accumulated resource consumption and utilization of the target core.

1: **Algorithm:** *Load-bounded Resource Balancing* (LRB)
2: **Input:** $Tasks$: task set to be distributed
3: **Output:** $Tasks_1, Tasks_2, ..., Tasks_m$: tasks sets assigned to the different cores
4: Let $U_{th} = \frac{\sum_{i=1}^{n} U_i}{m}$
5: **while** $Tasks$ is not empty **do**
6: $target\_task \leftarrow max : \forall task \in Tasks, RES_{max} \geq RES_{task}$
7: $target\_core \leftarrow min : \forall core, RES_{min} \leq RES_{core}$
8: **if** $U_{target\_core} + U_{target\_task} > U_{th}$ **then**
9: $target\_core \leftarrow min : \forall core, U_{min} \leq U_{core}$
10: **end if**
11: $U_{target\_core} \leftarrow U_{target\_core} + U_{target\_task}$
12: $RES_{target\_core} \leftarrow RES_{target\_core} + RES_{target\_task}$
13: $Tasks_{target\_core} \leftarrow Tasks_{target\_core} \cup \{target\_task\}$
14: $Tasks \leftarrow Tasks - \{target\_task\}$
15: **end while**

FIGURE 4.1: *Load-bounded Resource Balancing* heuristic. Legend: RES refers to the resource being balanced.

This behavior is maintained whenever the accumulated utilization of the target core after the assignment does not exceed a given threshold, calculated by averaging the utilization of the cores assuming that the workload is perfectly balanced. If this threshold is exceeded, then the task is assigned to the least loaded core. In this way, the algorithm pursues to balance the workload, enabling global speed and energy savings.

Finally, to evaluate how the proposed heuristics (*LRB-M* and *LRB-C*) perform they have been compared to the WF heuristic. The latter pursues to balance only the system workload, and it is known that it achieves the best overall performance among the partitioning heuristics in the literature [16].

Once the workload is partitioned, the power-aware scheduler must deal with scheduling the workload as well as energy savings.

4.2.2 Power-Aware HRT Scheduler

The power-aware scheduler is composed of an EDF scheduler per core and the global DVFS controller. Each EDF scheduler has two main jobs. First, the scheduler selects the minimum speed at which the task set running on its core is schedulable following the EDF algorithm [64] and sends this requirement to the global DVFS controller. Second, each core is assumed to execute three hardware threads, thus it can launch to execution up to three ready tasks, those with higher priority in its task set. Regarding EDF, these

tasks are those whose deadlines expire earlier. If any thread context is occupied by a low priority task, preemption is applied.

Recall that these jobs are performed each time a task arrives to or leaves the system due to two main reasons. The first one is because when the tasks set changes, a different speed can be required, and the second reason is that it may happen that because of a good overlapping of tasks running on the same core, when a task ends its execution there is more slack time (i.e., the remaining tasks will finish earlier), allowing the EDF scheduler to fullfil real-time constraints at lower speeds.

The power-aware scheduler has been modeled working with different levels (2, 3, and 5) of frequency and voltage. These models will be referred to as 2L, 3L, and 5L respectively. Frequency has been assumed ranging from 100 MHz to 500 MHz in steps of 100 MHz.[1] The 5L model allows the system to work at all the five frequency/voltage levels. The 2L only supports the extreme frequencies (i.e., 100 MHz and 500 MHz), and the 3L model also supports the intermediate one (i.e., 300 MHz).

In order to evaluate the performance of the proposed partitioner, two different baseline schedulers have been also evaluated: a single-level scheduler (1L), which always works at the maximum processor speed, and a naive two-level scheduler (2LN). The latter scheduler assumes that the processor is always working at the maximum speed (500 Mhz) except when there is no task running on any core. In that case, the system frequency is dropped down to the minimum one (100 Mhz).

4.2.3 Providing Support for SRT Tasks

When running a workload composed of only HRT tasks, the frequency might be set higher than needed due to the elapsed memory-processor overlapping time. However, setting a lower frequency could incur in deadline misses, which is not allowed in HRT workloads. In contrast, when supporting SRT tasks, relaxing the target frequency (i.e., setting it lower or much lower than required by the EDF scheduler) could bring important energy benefits. This can be done since SRT tasks can tolerate a certain percentage of deadline misses.

[1] Energy values have been extrapolated from the ones described in Chapter 3.

```
FREQUENCY_CALCULATION()
 1  if workload_change
 2    then
 3          FREQ_HRT ← HRT_calculation()
 4          switch
 5            case mode_1 :
 6                 FREQ ← FREQ_HRT − 100
 7            case mode_2 :
 8                 FREQ ← FREQ_HRT − 200
 9            case mode_3 :
10                 FREQ ← FREQ_HRT − 300
11            case mode_4 :
12                 FREQ ← FREQ_HRT − 400
13  if activations = w
14    then
15          activations ← 0
16          if lost_deadlines > th
17            then FREQ ← FREQ + 100
18          lost_deadlines ← 0
19
```

FIGURE 4.2: HRT and SRT frequency calculation.

When combining HRT and SRT tasks in the same workload, the target frequency is limited by the frequency required by HRT workloads to be schedulable. In addition, SRT tasks will have a lower priority than HRT tasks. Therefore, SRT tasks are allowed to run only when there is not any HRT task requiring processor computation. In short, SRT tasks are scheduled in background.

To support this behavior, the power-aware scheduler has been extended in several ways. First, the scheduler task queue has been split into two different queues, one for HRT tasks, and the other one for SRT tasks. Second, to obtain the working frequency, the scheduler calculates the frequency required by the whole workload (both HRT and SRT tasks) that avoids deadline misses. Then, this frequency is reduced depending on the selected power-saving execution mode. Four different modes have been implemented with different degrees of aggressiveness. These modes are referred as mode-1, mode-2, mode-3 and mode-4, where the number indicates how many frequency levels are reduced with respect to the frequency required by the EDF scheduler. For instance, if the required frequency is 400 MHz then mode-1, and mode-2 would reduce this frequency to 300 and 200 MHz, respectively. The first conditional structure of the algorithm detailed in Figure 4.2 shows how these values are calculated.

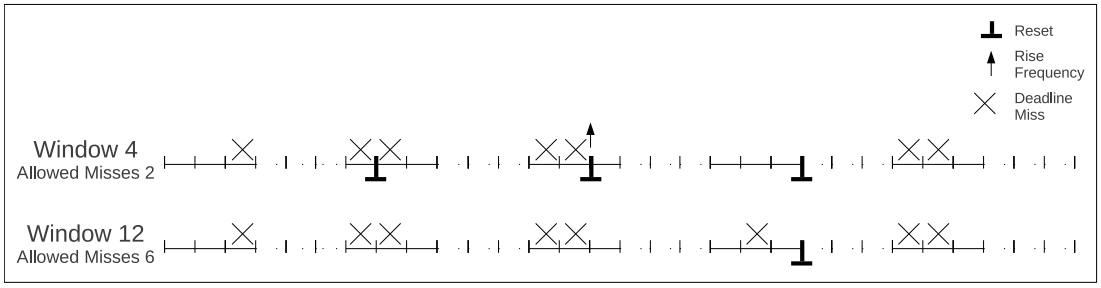


FIGURE 4.3: Frequency adjustment due to SRT deadline misses.

In this system, the frequency must be checked (and changed if required) in two main cases. The first case corresponds to the case described in Subsection 4.2.2. In addition, the frequency must be also checked when a SRT task misses its deadline. In the latter case, frequency will be risen to the next level when the amount of deadline misses reach a given threshold th. The algorithm checks the number of missed deadlines for a given task after w executions of that task, namely the window size. Missed deadline counters for each task are increased each time a deadline miss occurs and reset at the end of the window.

Figure 4.3 shows two examples of frequency adjustment due to SRT deadline misses varying w and th. For instance, a $w = 4$ window size rises the frequency at the end of the second window, because the *allowed misses* threshold is exceeded (i.e., 2), whereas a $w = 12$ window size with $th = 6$ does not modify the system frequency, since the number of deadlines missed is only 6 in the first window and 2 in the second one. Notice that, in this example, using a $w = 4$ window size avoids one deadline miss with respect to the $w = 12$ window size (sixth deadline miss).

When the frequency is risen because the threshold is reached, this new frequency level will remain stable during a minimum period of time, in order to avoid that the relaxed execution modes slow down too much the frequency which can bring an unacceptable number of deadline misses. This period has been fixed to 0.5M cycles in the experiments after checking a wide range of values.

Benchmark Name	Processor (M of cycles)	Memory (%)
Adpcm	6.41 H	24 L
Bitcnts	1.90 H	14 L
Bs	0.24 L	22 L
Bsort100	0.29 L	23 L
Cnt	0.43 M	22 L
Compress	0.41 M	26 M
Cover	0.32 M	22 L
Crc	1.01 H	22 L
Duff	0.28 L	24 L
Edn	2.91 H	30 H
Expint	0.35 M	24 L
Fac	0.25 L	22 L
Fdct	0.32 M	25 M
Fft1	0.34 M	25 M
Fibcall	0.25 L	27 M
Fir	0.34 M	24 M
Insertsort	0.28 L	24 L
Janne_complex	0.21 L	22 L
Jfdctint	0.32 M	23 L
Lms	13.21 H	27 H
Ludcmp	0.39 M	28 M
Matmult	9.62 H	27 M
Minver	0.29 L	24 L
Ndes	2.11 H	29 H
Ns	0.67 L	22 L
Nsichneu	0.41 M	33 H
Prime	0.60 M	24 L
Qsort-exam	0.25 L	24 L
Qurt	0.23 L	23 L
Select	0.25 L	24 M
Statemate	0.27 L	29 H
Ud	0.34 M	23 M

TABLE 4.1: Benchmarks requirements and classification in categories H, M, L.

4.3 Experimental Evaluation

For evaluation purposes, the system has been modeled on top of the Multi2Sim simulation framework, which was extensively extended as it has been explained in the previous chapter. More details about the required extensions to model this kind of systems can be found in [65].

To perform the experiments with two and four cores, a wide set of benchmarks from [61] have been used. Different mixes have been designed to explore the benefits on energy

Mix	Benchmarks
Mix 1	Cover, Duff, Fir, Janne_complex, Ludcmp, Ndes, Ns, Nsichneu, Statemate, Ud.
Mix 2	Cover, Duff, Expint, Fibcall, Jfdctint, Lms, Nsichneu, Ud.
Mix 3	Adpcm, Edn, Fft1, Lms, Matmult, Ndes, Ns, Qsort-exam, Qurt, Select.
Mix 4	Bitcnts, Bsort100, Compress, Crc, Fac, Jfdctint, Matmult, Minver, Prime, Qurt.
Mix 5	Bitcnts, Bs, Bsort100, Cnt, Compress, Cover, Crc, Duff, Edn, Ndes.
Mix 6	Minver, Ndes, Ns, Nsichneu, Prime, Qsort-exam, Qurt, Select, Statemate, Ud.
Mix 7	Adpcm, Bitcnts, Cnt, Compress, Cover, Crc, Duff, Edn, Expint, Fac, Fdct, Fft1, Fibcall, Fir, Insertsort, Janne_complex, Jfdctint.
Mix 8	Duff, Expint, Fdct, Fibcall, Insertsort, Janne_complex, Jfdctint, Lms, Matmult, Nsichneu, Qurt, Select, Statemate.
Mix 9	Bitcnts, Bs, Bsort100, Cnt, Compress, Cover, Crc, Duff, Edn, Expint, Fdct, Fibcall, Janne_complex, Jfdctint, Ndes, Nsichneu, Ud.
Mix 10	Adpcm, Bitcnts, Bsort100, Compress, Crc, Edn, Fac, Fft1, Jfdctint, Lms, Matmult, Minver, Ndes, Ns, Prime, Qsort-exam, Select.

TABLE 4.2: Benchmark mixes.

savings as described below.

4.3.1 Designing and Planning Mix Execution for HRT Tasks

A methodology consisting on the following main steps has been applied to design the benchmark mixes:

1. Benchmark characterization.
2. Benchmark classification.
3. Benchmark selection for a given mix.
4. Mix execution planning.

Table 4.1 shows the benchmarks used through the experiments, the corresponding memory and processor requirements, and Table 4.2 the mixes and their composition. These mixes were designed aimed at achieving core utilizations falling in between 30% and 90%. Memory requirements indicate the percentage of executed memory reference instructions (load instructions) and processor requirements are quantified in millions (M)

Criterion	Interval
Processor	3 > High > 1 0.7 > Medium > 0.4 0.4 > Low > 0.2
Memory	34 > High > 30 30 > Medium > 25 24 > Low > 20

TABLE 4.3: Benchmarks classification intervals attending to their processor requirements and memory reference instructions.

of cycles. Benchmarks have been classified attending to resource consumption. For each resource (processor and memory), benchmarks have been broken down into three main groups: low -L- (benchmarks having few resource requirements), high -H- (those having exceptional requirements), and medium size -M-. Table 4.3 shows the intervals considered for this classification.

The utilization value of each core depends on the applied partitioning heuristic since tasks distribution among cores varies according to the heuristic. As an example, Table 4.4 shows the results for mix 6 in a 2-core system. Values are presented as two pairs in each cell of the table. The first value of each pair represents the requirement of a given resource in core 0, expressed in percentage, with respect to the total requirements of both cores. The second one indicates such percentage for the second core. For instance, comparing the values corresponding to the WF heuristic (first column), with the ones corresponding to the *LRB-M* (second column), it can be seen that the first one distributes quite uniformly but in an unfair way resource usage while the second one improves the distribution of utilization and memory between both cores.

In the last step of the mix design, a planning of the execution of each mix is performed. The main aim behind this step is to simulate that in critical real-time applications, tasks are dynamically entering and leaving the system at different rates [12], as explained in

	Heuristic		
Resource	WF	*LRB-M*	*LRB-C*
(%) Memory	(63,37)	(51,49)	(48,52)
(%) CPU	(60,40)	(51,49)	(49,51)
Utilization	(84,77)	(85,76)	(78,83)

TABLE 4.4: Distribution of resource requirements and utilization (core 0, core 1) for each scheduling heuristic for mix 6.

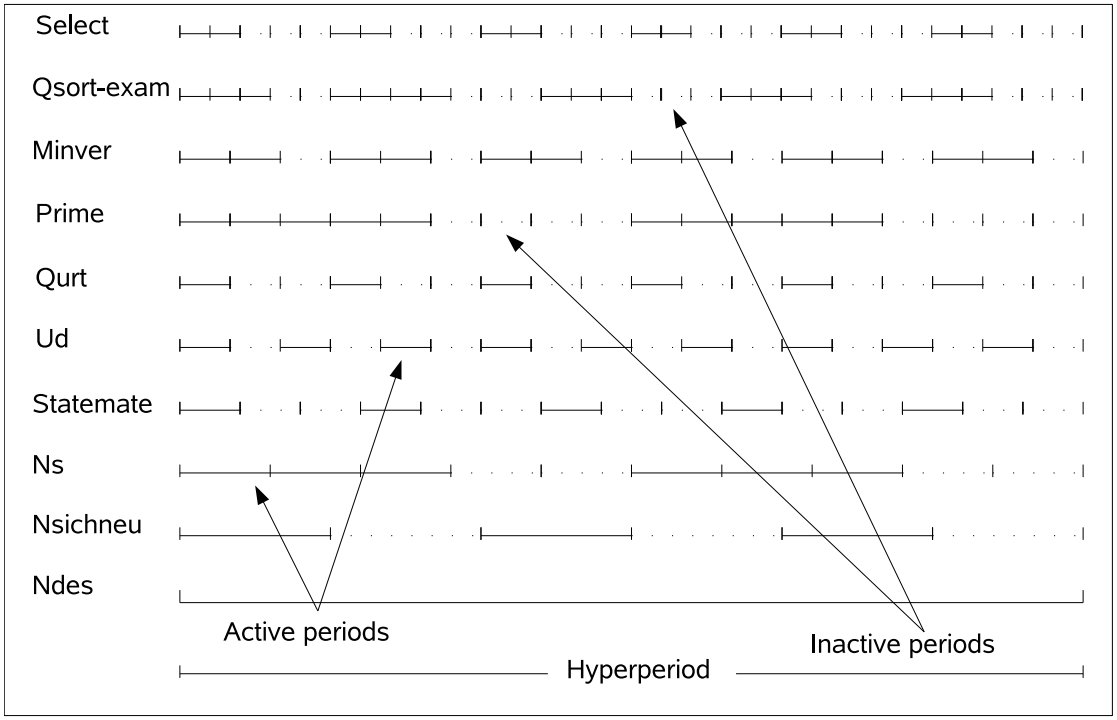


FIGURE 4.4: Example of the Gantt chart for mix 6. Continuous line means that the period is active. Discontinuous line means that the task is out of the system (inactive period).

Subsection 3.2.2.1. Due to this fact, the task period can be either active or inactive depending on if the corresponding task restarts the execution or not. Therefore, to evaluate these systems we need not only to select which benchmarks belong to each mix, but also how these benchmarks behave along the hyperperiod (lowest common multiple of the benchmarks period). The distribution of active and inactive periods for a given task has been randomly designed in order to introduce workloads variations which enable the power-aware scheduler to change the DVFS frequency/voltage level.

Figure 4.4 shows an example of a Gantt chart corresponding to mix 6 during its hyperperiod. This chart shows the relationship between the period of each task and the hyperperiod of the mix. It also shows the distribution of active (continuous line) and inactive (discontinuous line) periods. For instance, this mix is composed of ten benchmarks, which arrive to the system at the same time and whose hyperperiod is 90M cycles. Notice that the benchmark *Ndes* is running during the whole hyperperiod. On the contrary, the rest of the benchmarks remain active a given number of periods, log off the system, and after a given number of inactive periods enter the system again.

4.3.2 Designing Hybrid Mixes

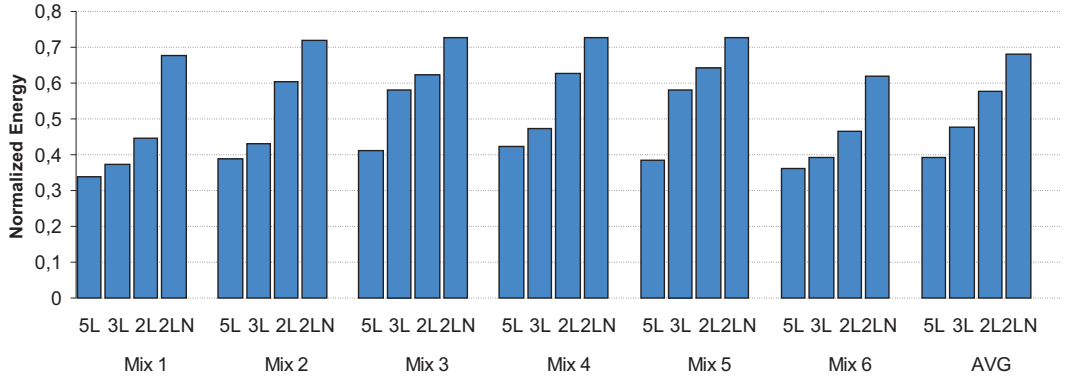
To evaluate the impact of combining HRT and SRT tasks (i.e., hybrid mixes), new mixes have been designed. Unlike in previous subsection where only HRT task are considered, the aim of these hybrid mixes is to check how the performance of SRT tasks may be altered due to the presence of HRT tasks in the system. This is a difficult task since HRT and SRT workloads may interact among them. For simplification purposes, it has been assumed that HRT tasks have not only the highest priority but also that their schedulability for the most aggressive execution mode is guaranteed. The methodology followed to design hybrid mixes matches the described above with the only exception that the focus is on SRT tasks since HRT tasks are assumed to have the same execution behavior. In addition, mixes have been designed to simulate how the system performs on overloaded conditions. In this way, the required processor speed will be one of the highest of the system, so the algorithm must tradeoff between energy savings and deadline misses.

4.3.3 Energy Savings for HRT Mixes

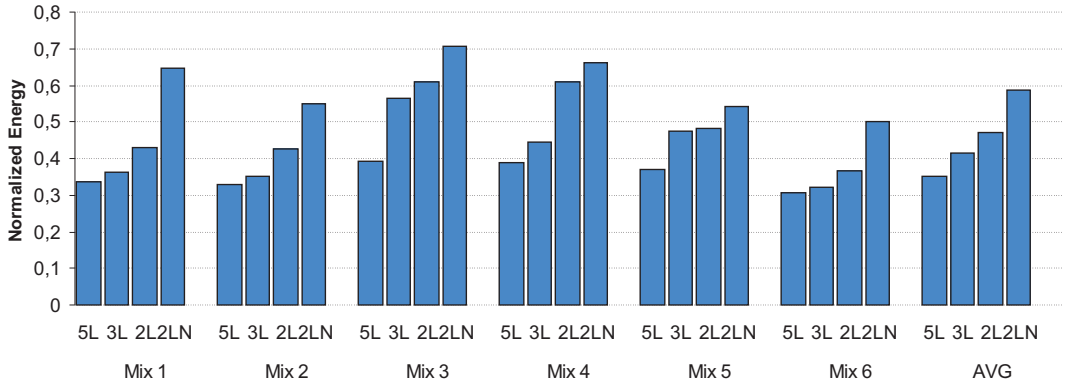
Figure 4.5 depicts the normalized energy consumption for the set of mixes (mix 1 to mix 6) in a two-core system. Energy values have been normalized with respect to the baseline model (1L) whose energy consumption has been assumed to be equal to 1. From these values, power benefits can straightforwardly be deduced. For instance, a 0.4 value means that a given heuristic requires 40% the energy consumed by the baseline. Thus, the heuristic achieves energy savings by 60% (i.e., $1 - 0.4$). In other words, it requires 2.5 (1/0.4) times less energy.

Attending to these results, three main conclusions can be drawn. Regarding the partitioning heuristics, the proposed heuristic (*LRB-M*) consumes about 4% (i.e., $0.39 - 0.35$) less energy than the WF heuristic for the 5L model, which represents about 10% relative energy savings.

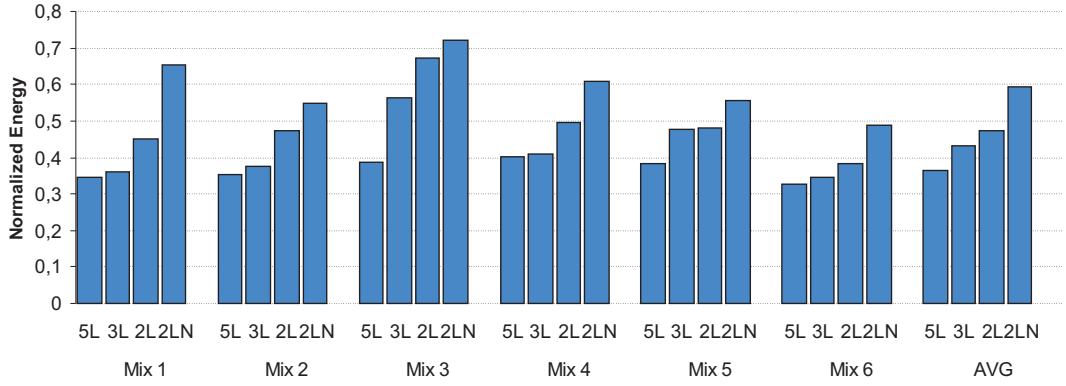
Regarding the distribution of memory/processor requirements, the *LRB-M* heuristic achieves, on average, higher energy savings than the *LRB-C* heuristic. For instance, normalized energy consumed by the mix 2 with the *LRB-M* heuristic is about 3% less than the consumed by the *LRB-C*. This is because the memory time is significantly



(a) *Worst Fit* (WF)



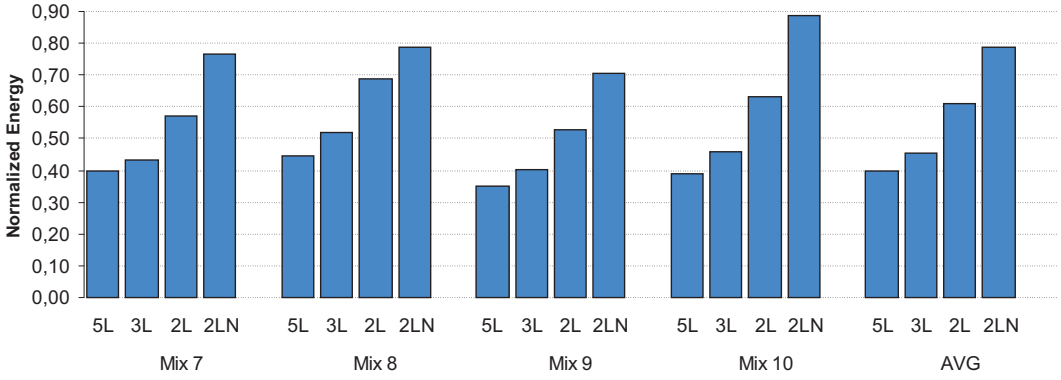
(b) *Load-bounded Resource Balancing* - Memory (*LRB-M*)



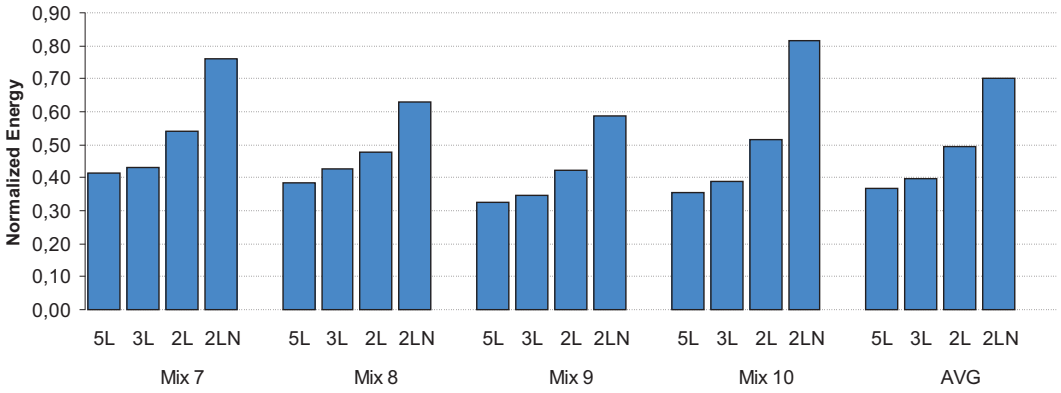
(c) *Load-bounded Resource Balancing* - CPU (*LRB-C*)

FIGURE 4.5: Normalized energy (2 cores).

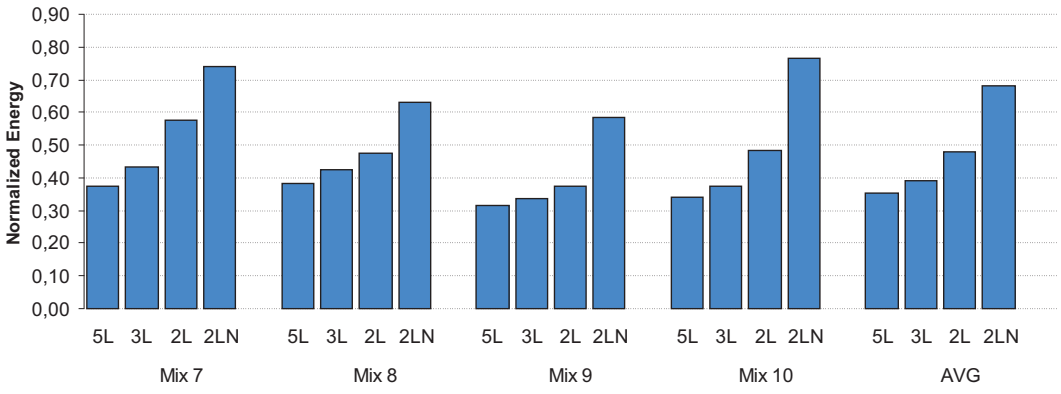
higher (about four times) than the processor time. Therefore, the overlapping time highly increases when the memory time is balanced, so allowing the scheduler to turn down the speed.



(a) *Worst Fit* (WF)



(b) *Load-bounded Resource Balancing* - Memory (*LRB-M*)



(c) *Load-bounded Resource Balancing* - CPU (*LRB-C*)

FIGURE 4.6: Normalized energy (4 cores).

Regardless the applied heuristic, the best results are achieved when working with the higher number of frequency/voltage levels. This is because when the system has a wide range of frequencies, the scheduler adjusts better the speed of the system to the task

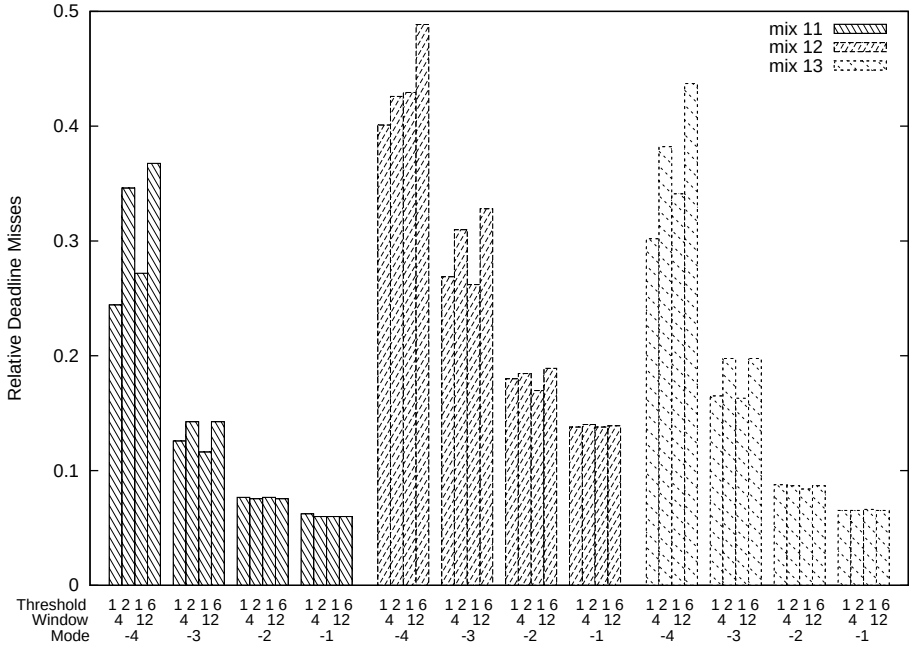
set requirements. For instance, if the task set requires at least a 150 MHz frequency, the 3L model would select the 300 MHz frequency while the 5L model would select 200 MHz. Therefore, the latter model saves more energy. Results show that working with the 5L model provides, on average, by about 29% more relative power savings than working with the 2L model. Energy differences among the 2L and the 2LN levels (see Subsection 4.2.2) are significant, on average, about 18% which shows that applying the power-aware scheduler has an important effect on power-consumption.

The proposed policies can work on any number of cores. Notice that a higher number of cores, which is the current trend in multicore systems [1, 11], will waste an important percentage of the energy budget when using global DVFS. Nevertheless, when applying the algorithm in a m-core system, the algorithm must guarantee that the utilization per core and the task resource consumption is equally distributed. Figure 4.6 shows the experimental results for different mixes (mix 7 to mix 12) when executing in a four-core system. As the mixes executed in the four-core system have similar characteristics (i.e., similar utilization values and resource requirements) to the ones shown in Figure 4.5 (two-core system) the results are also in the same line.

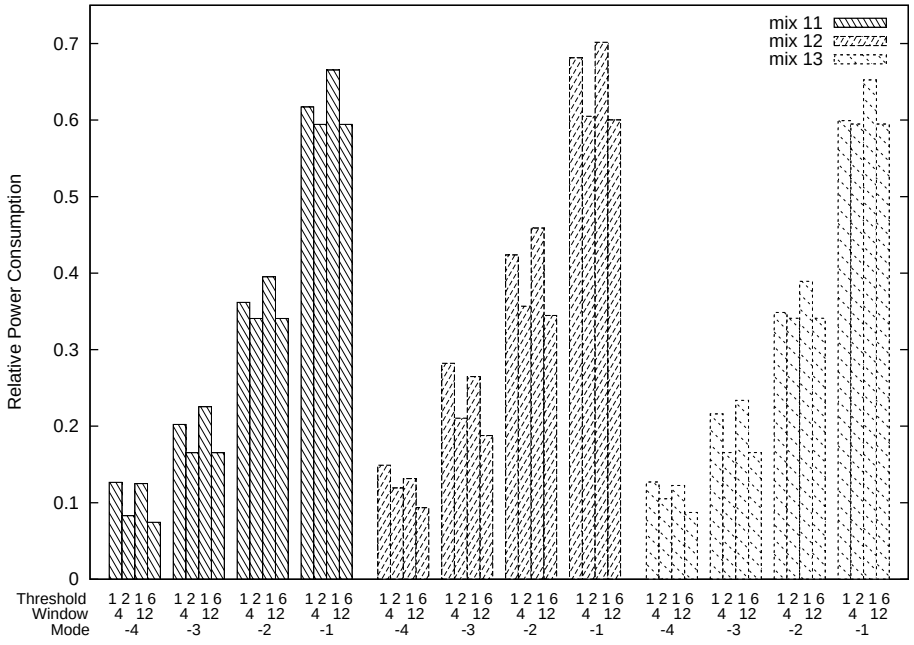
Finally, the main conclusion that can be drawn is that by using the power-aware scheduler with a high number of frequency levels (5L) and a fair heuristic strategy (*LRB-M*), normalized energy can be 65% lower than working with the baseline model.

4.3.4 Energy Savings versus Deadline Misses for Hybrid Mixes

When SRT tasks are introduced in the system, not only the energy savings must be evaluated, but also the QoS in terms of deadline misses as well. The experiments have been run varying the window size ($w = 2$ and $w = 4$), the threshold ($th = 1$ and $th = 2$ for $w = 4$, and $th = 1$ and $th = 6$ for $w = 12$) and the execution mode (-1, -2, -3 and -4). Figures 4.7 and 4.8 show the relative deadline misses and power consumption for the extended *LRB-M* heuristic when applied in a system supporting HRT and SRT tasks, for 2 and 4 cores, respectively. Notice that only the 5L configuration has been considered for hybrid experiments since it makes no sense to have more execution modes than available frequency levels. The percentage of deadline misses is obtained by dividing the number of lost deadlines by the total number of SRT periods executed.



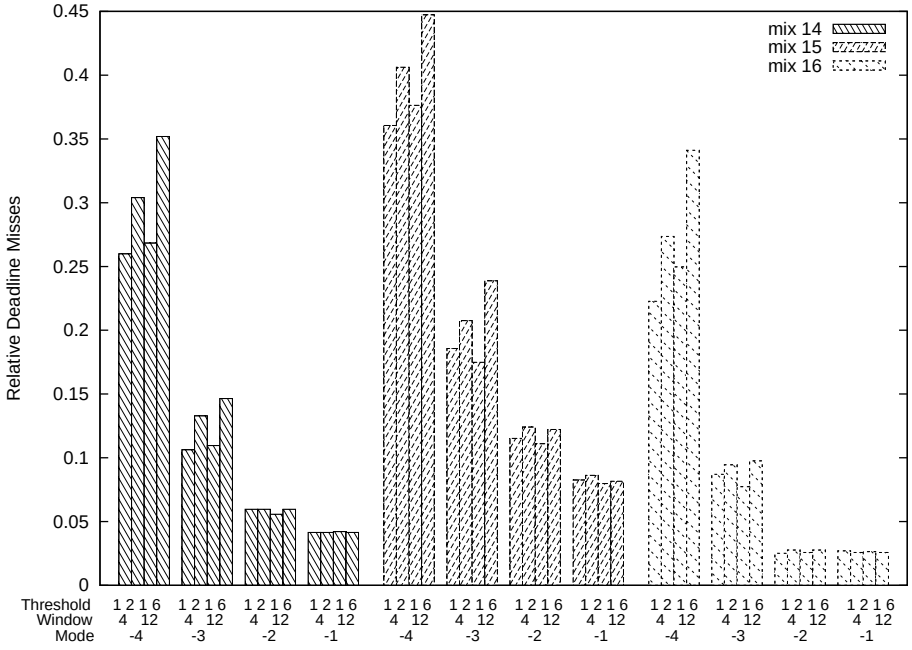
(a) Deadline Misses.



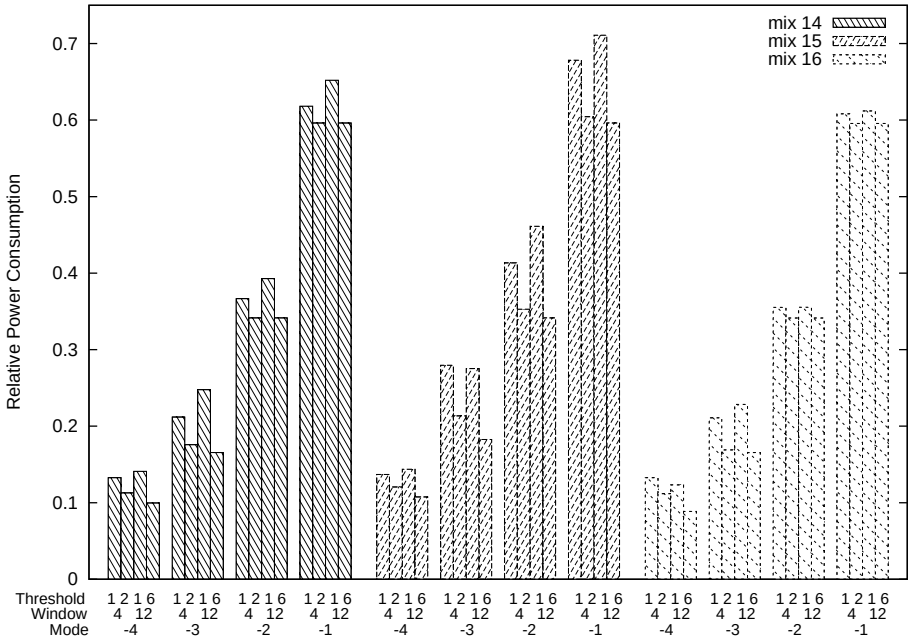
(b) Energy Consumption.

FIGURE 4.7: Normalized deadline misses and energy (2 cores).

The results show that both deadline misses and power consumption graphics have opposite forms, that is, the more the power consumption, the less the number of deadline misses. This is due to the fact that when the frequency is risen, more slack time is available and hence SRT tasks are easily scheduled, however, more energy is consumed.



(a) Deadline Misses.



(b) Energy Consumption.

FIGURE 4.8: Normalized deadline misses and energy (4 cores).

Looking at the three parameters shown in Figures 4.7 and 4.8 (threshold, window size, and mode) it can be pointed out that the mode parameter is the one that has the strongest impact on both power consumption and deadline misses. If the main objective of the system is to minimize the number of deadline misses, the best choice

Power Consumption	Deadline Misses
High	Low
Intermediate-High	Intermediate-Low
Intermediate-Low	Intermediate-High
Low	High

TABLE 4.5: QoS requirements trade-off.

is mode-1, whereas if the priority is to reduce power consumption, mode-4 should be the selected mode. For intermediate requirements mode-2 and mode-3 would be more suitable choices.

Notice that safer modes (-1 and -2) lose less deadlines but the most aggressive present less power consumption. As stated above, the system has to trade-off between these two parameters to establish the frequency value. In this sense, the system uses the threshold parameter to determine the risk assumed in terms of deadline misses. Recall that a higher threshold can reduce power consumption, but at the expense of allowing more deadline misses. The influence of the threshold is larger in the two more risky frequency modes (-4 and -3), as it can be seen in Figure 4.7(a). This is because in the safer modes (-1 and -2) working at higher frequencies prevents the system from losing an excesive number of deadlines, since the threshold is not reached as many times as in the risky modes. It can be also seen that given a threshold for mode-4, a larger window size implies more energy savings although the number of deadline misses increases.

In spite of the aforementioned considerations, the used window lenghts obtain important energy savings and this parameter might be tuned depending on the workload characteristics, that is, considering the number of task repetitions. For example, in a system where most benchmarks execute from 5 to 8 times during the hyperperiod, a 4 window size would be more appropriate and would achieve better results than a 12 window size. Once a mode is chosen, the system can prioritize the QoS requirements in a different manner during its lifetime (Table 4.5). For instance, the system could start by setting a low number of deadline misses, but at the end of the system mission, when the battery is going down, it can be more useful allowing more deadline misses in order to reduce power consumption and make a better use of the remaining energy budget.

Regarding the scalability, the mixes (14, 15 and 16) launched in the four-core processor were designed following the same criteria as the mixes (11, 12 and 13) launched in the

two-core processor. Comparing Figure 4.7 and Figure 4.8, it can be noticed that the scheduler shows a good behavior with the number of cores in both deadline misses and power consumption.

4.4 Conclusions

This chapter has presented HRT and SRT power-aware partitioner and scheduler for a CGMT multicore processor implemented in a wide range of embedded systems. The scheduler guarantees HRT task deadlines and saves energy applying DVFS techniques. Regarding the partitioner, a new heuristic to partition the workload has been introduced. The proposed heuristic distributes the task set attending to the CPU or memory requirements among the system cores in order to increase the overlapping time and hence get extra slack time that can be devoted to reduce power consumption.

Results show that applying the proposed heuristic (*LRB-M*) in HRT task environments, energy consumption is by 10% lower than when applying the WF heuristic with a power-aware scheduler and 5 DVFS frequency/voltage levels. If no power-aware scheduler is applied (2LN), the results provided by the *LRB-M* heuristic achieve about 14% more energy benefits than WF.

Regarding the results of the extended heuristic scheduler when executing hybrid mixes composed of HRT and SRT tasks, energy consumption can vary from 8% to 70%, and deadline misses from 3% to 48%, depending on the allowed deadline misses and on the selected frequency. Therefore, the required QoS is the key issue to determine both power consumption and deadline misses. Then, a higher number of allowed deadline misses of SRT tasks can help to reduce much more power consumption at the expense of increasing the deadline misses and viceversa.

Chapter 5

Task Migration

5.1 Introduction

This chapter presents two task migration algorithms to reduce energy consumption in multicore embedded systems with real-time constraints implementing DVFS capabilities. The simpler algorithm, namely, *Single Option Migration* (*SOM*) only checks just one target core before performing a migration. In contrast, the *Multiple Option Migration* (*MOM*) searches the optimal target core. To address energy savings, the devised algorithms follow two main rules: (i) migrations are allowed only in those points of time when the workload changes, that is, when tasks enter or leave the system, and (ii) only one task is allowed to migrate each time because checking all the possible task migrations may result in a prohibitive overhead. The partitioner module is in charge of readjusting possible workload imbalances at run-time that may occur at arrivals or exits of tasks by applying task migration. Three variants of the *SOM* algorithm have been devised, depending on the point of time the scheduler is applied: when a task arrives to the system (SOM_{in}), when a task leaves the system (SOM_{out}), and in both cases (SOM_{in-out}).

5.2 Proposed Task Migration Heuristics

There are several partitioning heuristics that can be used to distribute tasks among cores as they arrive to the system. As commented in Subsection 4.2.1, the WF partitioning

heuristic is considered one of the best choices in order to balance the workload [16], yielding to improved energy savings. WF balances the workload by assigning each incoming task to the least loaded core. If more than one task arrives to the system at the same time, WF arranges the incoming tasks in a decreasing utilization order and assigns them to the cores starting with the task with the highest utilization. This algorithm was originally used in partitioned scheduling, and it does not support any task migration among cores by design. In other words, once WF assigns an incoming task to a given core, the task remains in that core until it leaves the system (i.e., it has executed all its active periods). To allow migration, *SOM* policies are devised in the next subsection.

5.2.1 *Single Option Migration* Policies

Figure 5.1 shows an example of how task migration could improve workload balance. At the beginning of the execution (time *t0*), *task 0* and *task 1* are the only tasks assigned to *core 0* and *core 1*, respectively. *Task 0* presents an utilization by 33%, while the utilization of *task 1* is around 25% (i.e., its WCET occupies a quarter of its period). At point *t2*, *task 2*, whose utilization is around 66%, arrives to the system. The WF algorithm would assign it to *core 1* (since it is the least loaded core); leading the system to a high workload imbalance since the global utilization of *core 0* and *core 1* would be 33% and 91%, respectively. This imbalance problem could be solved by allowing task migration. For instance, allowing *task 1* to migrate to *core 0*, would provide a much fair balance (58% in *core 0* versus 66% in *core 1*).

This work assumes that the running workload dynamically changes at run-time. In this context, the system can mainly become strongly unbalanced when the workload changes, that is, when a task enters or leaves the system, as seen in the previous example. Thus, in the evaluated system migration policies should apply in these points in order to maximize benefits due to migration. For this purpose, we have devised three policies based on the WF policy to explore energy benefits: SOM_{in}, SOM_{out}, and SOM_{in-out}. The first one, SOM_{in}, allows migration only when a new task arrives to the system, SOM_{out} when a task leaves the system, and the last one, SOM_{in-out}, allows migration in both cases.

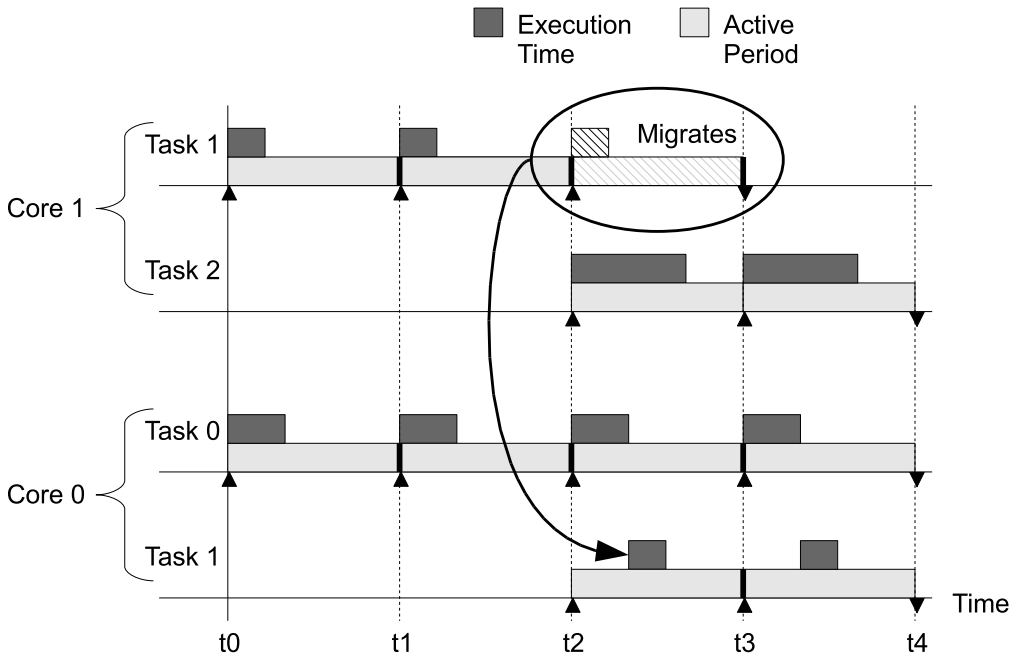


FIGURE 5.1: Example of task migrations to balance the system workload.

Since workload changes only at two points of time (when a task arrives to the system or when a task leaves the system) it only makes sense to apply migration at these events. In addition, to avoid performing an excessive number of migrations, which could lead to an unacceptable overhead, the number of migrations is limited to only one.

Figure 5.2 illustrates the devised *Migration Attempt* (*MA*) algorithm. This algorithm calculates the imbalance by subtracting the utilization of the least loaded core from the utilization of the most loaded one. This result is divided by two to obtain a theoretical utilization value that represents the amount of work that should migrate to achieve a perfect balance between both cores, and hence, a better global balance. Then, it searches the task in the most loaded core whose utilization is the closest one to this value. Notice that if the utilization of the selected task is not close enough, the migration could yield to a worse imbalance; therefore, the algorithm performs the migration only if it effectively reduces the imbalance.

5.2.2 *Multiple Option Migration* Dynamic Partitioner

This subsection presents the *MOM* dynamic partitioner algorithm, which applies both at tasks' arrivals and exits. When a task arrives to the system, *MOM* selects the target

1: $imbalance \leftarrow max\_core\_utilization - min\_core\_utilization$
2: $target\_utilization \leftarrow imbalance/2$
3: $minimum\_difference \leftarrow MAX\_VALUE$
4: **for all** $task$ in $most\_loaded\_core$ **do**
5: **if** $|U_{task} - target\_utilization| < minimum\_difference$ **then**
6: $minimum\_difference \leftarrow |U_{task} - target\_utilization|$
7: $candidate \leftarrow task$
8: **end if**
9: **end for**
10: $new\_max\_core\_utilization \leftarrow max\_core\_utilization - U_{candidate}$
11: $new\_min\_core\_utilization \leftarrow min\_core\_utilization + U_{candidate}$
12: $new\_imbalance \leftarrow |new\_max\_core\_utilization - new\_min\_core\_utilization|$
13: **if** $new\_imbalance < imbalance$ **then**
14: $migrate(candidate)$
15: **end if**

FIGURE 5.2: *Migration Attempt* algorithm.

core and performs a *migration attempt* acording to the *MA* algorithm discussed above. When a task leaves the system, *MOM* checks if a *migration attempt* would provide energy improvements.

MOM (Figure 5.3) arranges the tasks arriving to the system in decreasing utilization order. Then, it iteratively performs a temptative assignment of the task showing more utilization to each core in order to find which assignment provides the minimum utilization (U_{min} variable in the figure) for the most loaded core. Notice that all the possible assignments include a *migration attempt* according to the *MA* algorithm discussed above. Finally, the task assignment that provides the best overall balance is applied and the algorithm continues with the next task.

Figure 5.4 depicts an example where the *MOM* heuristic improves the behavior of SOM_{in-out} on a task arrival. The SOM_{in-out} allocates the incoming task to core 0 and then performs a *migration attempt*, but in this case, there is not any possible migration enabling a better workload balance. Thus, the final imbalance becomes 40% (i.e., $90\% - 50\%$). In contrast, when *MOM* is applied, it also checks the result of allocating the new task to core 1 (arrow labeled as *MOM* B) and then considering one migration. In this case, the task migration enables a better balance since both cores remain equally loaded with 70% of utilization, which is the distribution selected by *MOM*.

To sum up, the main difference between SOM_{in-out} and *MOM* is that the former selects only one core and performs a *migration attempt*, whereas the proposed heuristic checks

1: **Algorithm:** *Multiple Option Migration* Dynamic Partitioner (*MOM*)
2: **Input:** $Task\_set(Task_0, Task_1, ..., Task_{T-1})$: task set to be distributed;
3: **Input:** T: number of tasks
4: **Input:** $Core\_set(Core_0, Core_1, ..., Core_{M-1})$: cores in the system
5: **Input:** M: number of cores
6: **Input/Output:** $Tasks_0, Tasks_1, ..., Tasks_{M-1}$: tasks sets assigned to the different M cores
7: **while** $Task\_set$ is not empty **do**
8: $target\_task \leftarrow Task_i : (Task_i) \geq MAX(U(Task_0), U(Task_1), ..., U(Task_{T-1}))$
9: $U_{min} \leftarrow \infty$
10: $initial\_task\_assignment = Tasks_0, Tasks_1, ..., Tasks_{M-1}$
11: **for all** $target\_core$ in $Core\_set$ **do**
12: $Tasks_{target\_core} \leftarrow Tasks_{target\_core} \cup \{target\_task\}$
13: $Migration\_Attempt()$
14: **if** $U_{min} > MAX(U(Core_0), U(Core_1), ..., U(Core_{M-1}))$ **then**
15: $U_{min} \leftarrow MAX(U(Core_0), U(Core_1), ..., U(Core_{M-1}))$
16: $best\_task\_assignment \leftarrow Tasks_0, Tasks_1, ..., Tasks_{M-1}$
17: **end if**
18: $Tasks_0, Tasks_1, ..., Tasks_{M-1} \leftarrow initial\_task\_assignment$
19: **end for**
20: $Tasks_0, Tasks_1, ..., Tasks_{M-1} \leftarrow best\_task\_assignment$
21: **end while**

FIGURE 5.3: *Multiple Option Migration* dynamic partitioner algorithm.

different cores, and then choses the best option in terms of workload balance.

5.3 Experimental Results

Experimental evaluation has been conducted on the extended Multi2Sim simulation framework. This section evaluates a multicore processor with two, three and four cores, implementing three hardware threads each. Regarding the migration overhead, a 10.000 cycles penalty has been assumed [66]. This penalty is applied each time a running context moves its execution to another core.

Because of energy constraints, embedded systems are still limited to a lower number of cores than their high-performance counterparts. Therefore, energy evaluation results focus on a low number of cores: two, three and four cores. Some examples are the bi-core Intel Atom [60], the tri-core Marvell ARMADA^{TM} 628 [67] or the quad-core ARM 11 MPCore [68]. On the other hand, this chapter assumes a relatively wide number of frequency/voltage levels (up to eight) in order to approach the results to real systems. The 8L configuration allows the system to work at all the frequencies indicated

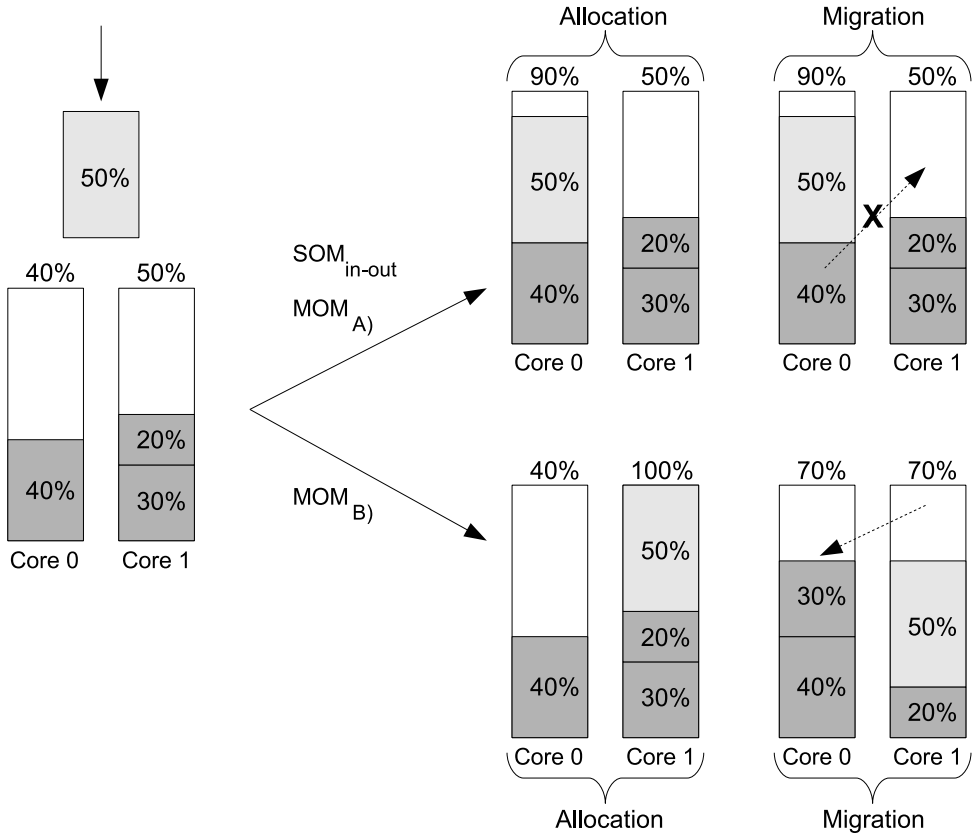


FIGURE 5.4: SOM_{in-out} versus *MOM* working example.

in Chapter 3, whereas the 4L mode permits running tasks at 1700, 1400, 1100 and 600 MHz. The last DVFS configuration, referred to as 2L, only supports the extreme frequencies (i.e., 600 and 1700 MHz).

Table 5.1 shows the benchmarks from [61] that have been used to prepare real-time workload mixes (a benchmark name with an asterisk means that the benchmark is used in the mix more than once). These experiments focus on a HRT system, so deadline misses are not allowed. Each mix is composed of a set of benchmarks whose number ranges from 7 to 34, running concurrently depending on the number of cores. Mixes 1, 2 and 3 are executed in a 2-core system, mixes 4, 5 and 6 in a system with three cores, and mixes 7, 8 and 9 in a 4-core system. To calculate the WCET, each benchmark has been executed alone in the modeled system working always at the lowest frequency. The WCET is measured in number of cycles, that is, it accounts the total number of cycles that a thread takes to execute.

In order to test the algorithms behavior across a wide range of situations, when designing

Mix	Benchmarks
Mix 1	Bs, Fac, Fibcall, Janne_complex, Lcdnum, Sqrt, Statemate.
Mix 2	Cnt, Compress, Expint, Fac, Fft1, Janne_complex, Jfdctint, Ludcmp, Qurt.
Mix 3	Bs, Bsort100, Cnt, Compress, Duff, Expint, Fac, Fft1, Fibcall, Insertsort, Lcdnum, Loop3, Minver, Ns, Statemate.
Mix 4	Bs*, Cnt, Compress, Duff*, Expint*, Fac*, Fft1*, Fibcall*, Insertsort*, Lcdnum*, Minver*, Statemate*.
Mix 5	Cover*, Fdct, Fir, Janne_complex*, Jfdctint, Ludcmp*, Minmax*, Nsichneu*, Qsort-exam*, Qurt*, Select*, Sqrt*.
Mix 6	Fibcall*, Lcdnum*, Loop3*, Minmax*, Select*, Sqrt*.
Mix 7	Cnt*, Compress*, Crc*, Edn*, Expint*, Fac*, Fft1*, Janne_complex*, Jfdctint*, Ludcmp*, Qurt*.
Mix 8	Cover*, Fdct*, Fir*, Janne_complex*, Jfdctint*, Ludcmp*, Minmax*, Nsichneu*, Qsort-exam*, Qurt*, Select*, Sqrt*.
Mix 9	Bs*, Fac*, Fibcall*, Janne_complex*, Lcdnum*, Sqrt*, Statemate*.

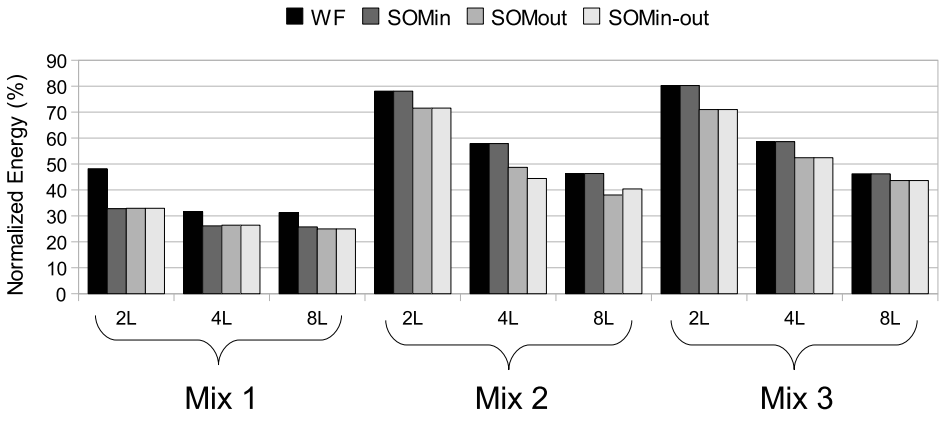
TABLE 5.1: Benchmarks and mixes. Legend: * the benchmark appears more than once in the mix.

mixes it has been taken into account the task utilization, the task WCET and the sequence of active and inactive periods. To this end, the global system utilization is set ranging from 35% to 95% in a single execution, task periods from 100K to 18M cycles, the number of times that a task arrives to and leaves the system from 1 to 21, and the consecutive number of active periods of a task from 1 to 70.

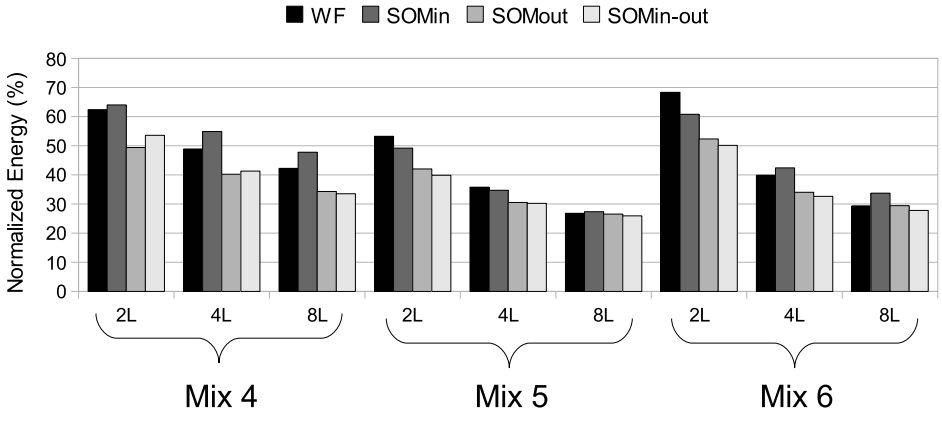
5.3.1 Impact of Applying Migrations at Specific Points of Time

This section analyzes the three devised *SOM* variants (SOM_{in}, SOM_{out} and SOM_{in-out}). The main goal is to identify the best points of time to carry out migrations. Figure 5.5 shows, for different benchmark mixes, the relative energy consumption compared to the energy consumed by the system working always at the maximum speed varing the DVFS configurations and number of cores.

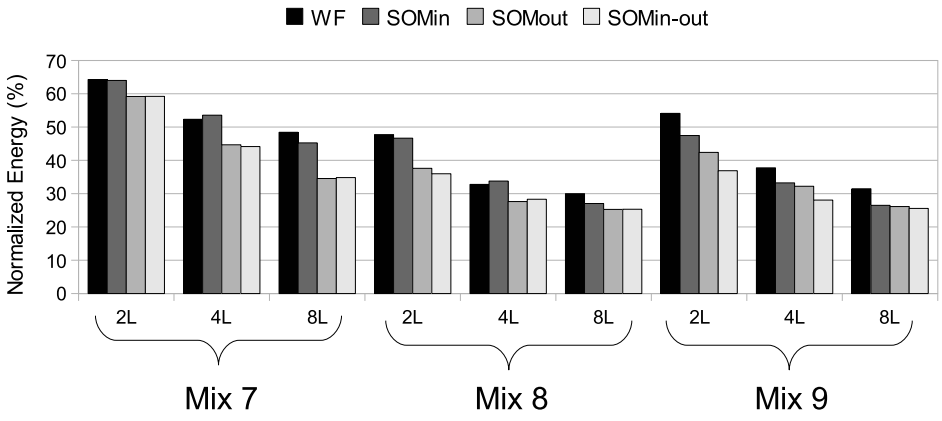
As it can be observed in the results of the 2-core system (Figure 5.5(a)), migration can provide important energy savings with respect to no migration (WF). For instance, for mix 2 in the 4L case with task migration, both when a task arrives to and leaves the system, the energy consumption can be reduced up to by 23.27% compared to the execution without migration. Notice that the system behaves in a similar way regardless of the number of cores, that is, the benefits of migration that are observed in



(a) 2 Cores



(b) 3 Cores



(c) 4 Cores

FIGURE 5.5: *SOM* variants comparison for different DVFS levels and number of cores.

a system with two cores are also similar in systems with three or four cores, as shown in Figures 5.5(b) and 5.5(c). This fact makes the proposal a good candidate for commercial systems attending to the current industry trend of increasing the core count.

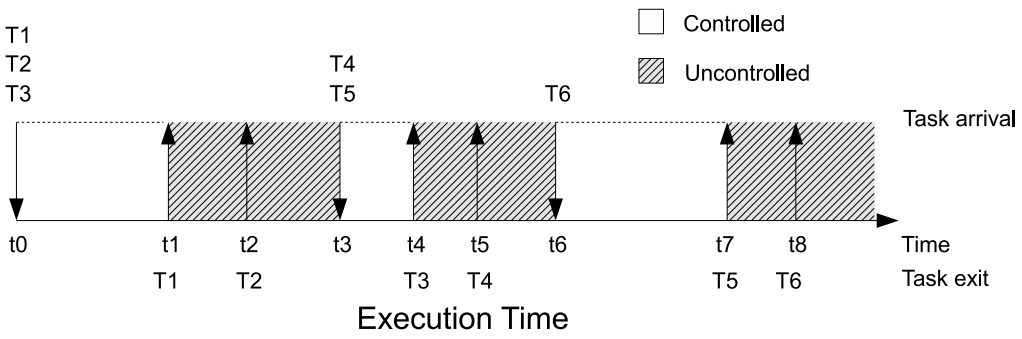


FIGURE 5.6: Effective action of the SOM_{in} partitioning algorithm.

An interesting observation is that, in some mixes, the SOM_{in} variant consumes more power than the classical WF algorithm with no migration. For example, in the 3-core system (Figure 5.5(b)) allowing migrations only at tasks' arrivals turns out in harmful effects for mix 4 in terms of power consumption, where SOM_{in} consumes 12.27% more energy than WF for 4L configuration. The reason is related to the fraction of time length that the system is *controlled* by the partitioning algorithm. That is, the SOM_{in} partitioning heuristic only applies at tasks' arrivals. Therefore, as soon as a task leaves the system, the workload imbalance will rise since SOM_{in} does not apply on such events.

Figure 5.6 illustrates an example. At *time t0 tasks T1*, *T2*, and *T3* arrive to the system, and the scheduler selects the frequency/voltage level that best fits the workload requirements. Lets assume that the workload is perfectly balanced in a 2-core system. Then at *time t1*, *task T1* leaves the system, so workload imbalance will rise (dashed area), in algorithms such as WF or SOM_{in} where migration is not performed, so yielding to energy wasting. Notice that this area is *uncontrolled* since the set of tasks running has changed. On the contrary, the *controlled* time periods are those where the set of tasks running matches the set used to perform the scheduling actions. Moreover, further imbalance would rise if the next task *T2* leaves the system from the same core. This imbalance will remain until the algorithm applies, which happen only on tasks' arrivals in WF and SOM_{in} (in *t3*). This drawback is solved in the algorithms which allow task migration at such points like SOM_{out}, SOM_{in-out} and *MOM*. Table 5.2 shows which actions are performed by the different algorithms both when a task arrives to and leaves the system.

The longer the algorithm controls the running workload, the better the workload balance. Consequently, the frequency levels requested by the different cores will be similar, so

Algorithm	Task Arrival	Task Exit
WF	WF	-
SOM_{in}	WF, Task Migration	-
SOM_{out}	WF	Task Migration
SOM_{in-out}	WF, Task Migration	Task Migration
MOM	*MOM*	Task Migration

TABLE 5.2: Algorithms action on workload changes.

avoiding energy wastings. Figure 5.7 shows, for mix 4, in a 3-core system with 8 DVFS levels, the difference among frequencies required by the cores along the execution time (in percentage). For instance, label 0 means that both cores require the same frequency and label 2 means that the core with less frequency/voltage requirements requested level i to the DVFS regulator, while the core with the maximum requirements requested level i+2. This figure explains the curious behavior identified above, where SOM_{in} performed worse than WF. As observed, both partitioners yield the system to spend a similar amount of time with all the cores requiring a similar speed (i.e., with a difference less or equal than 1 level). Nevertheless, the main reason why SOM_{in} consumes more power than WF is that, in this mix, there is a significant amount of time where the difference in speed required by the cores in SOM_{in} is 3 and 4 levels, while in WF most of this time the difference is only 2 levels. Notice that SOM_{out} and SOM_{in-out} balance the workload in a better way (area associated to label 0 is much longer) than WF and SOM_{in}, the reason is due to the former *control* the system both at tasks' arrivals and exits.

As expected, if the system implements more DVFS frequency levels, then more energy savings can be potentially obtained since the system can select a frequency closer to the optimal estimated by the scheduler. However, despite this fact, in some cases energy benefits due to migration in a system with few frequency levels can reach or even surpass the benefits of having more levels without migration. For example, the energy consumption of SOM_{in-out} for mix 2 in the 4L 2-core system is around 44% the consumption of the baseline, whereas the same value of WF in the 8L system is 46%.

5.3.2 Comparing *MOM* versus *SOM* Variants

This section analyzes the energy improvements of the proposed *MOM* algorithm over the *SOM* variants. For comparison purposes the best *SOM* variant (SOM_{in-out}), on

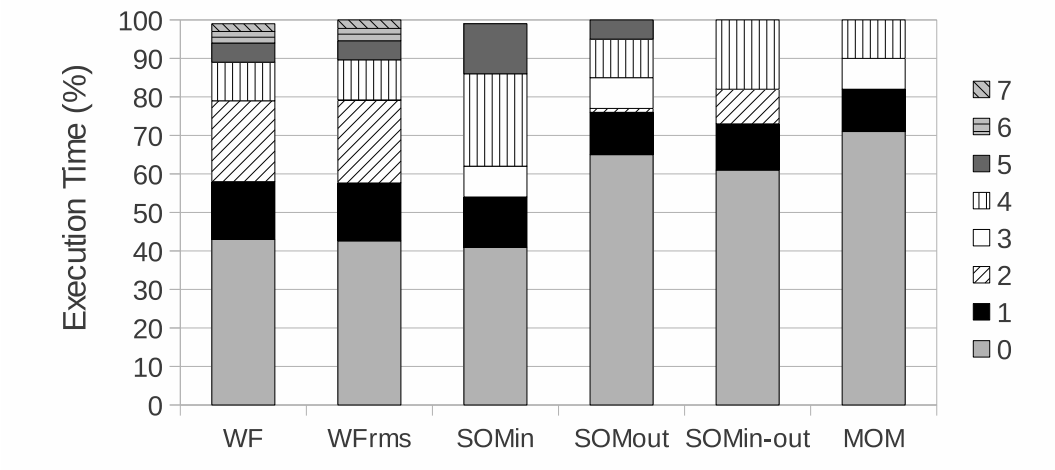


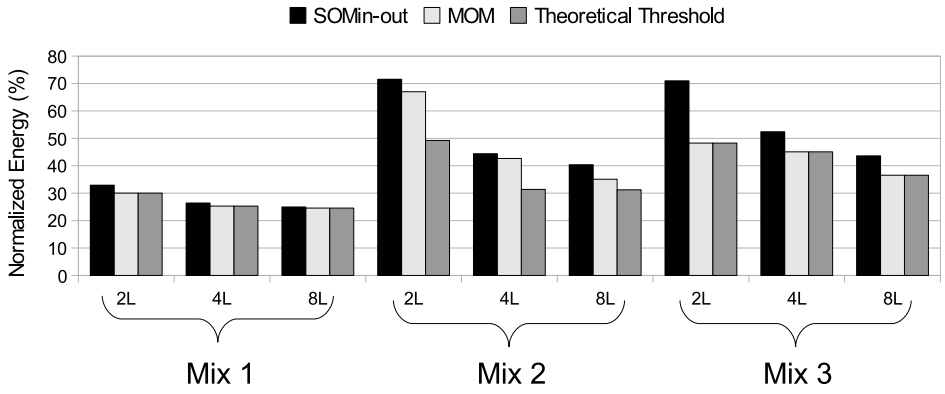
FIGURE 5.7: Differences of the required frequencies in the 3-core system for mix 4.

average, and a theoretical threshold have been also included in the plots. This theoretical threshold is a value that represents the maximum energy savings that can be achieved in a system where the number of task migrations is not limited, they have no cost, and they can be performed at any point of the execution time. That is, a system with perfect task balancing and without penalties due to migration. Figure 5.8 shows the energy results for two, three and four cores, normalized with respect to the energy consumed by the system working always at the maximum speed.

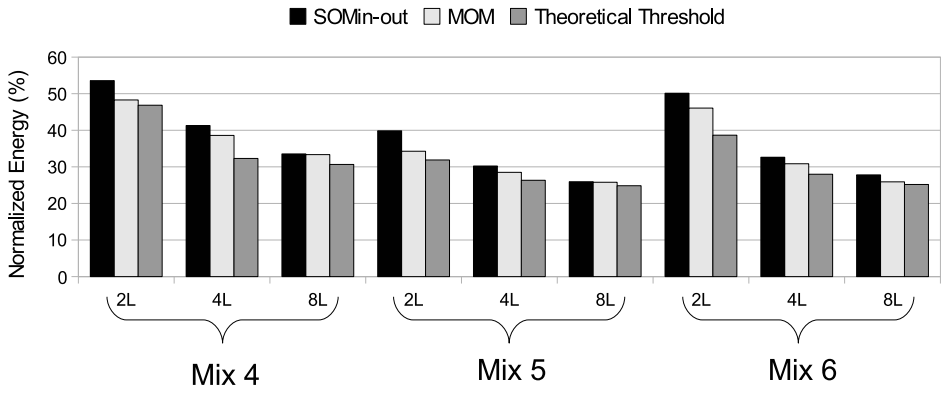
Results show that regardless of the number of cores, the mix, and the number of frequency levels, *MOM* saves more energy than SOM_{in-out}. For example, when running mix 3 in the 2L 2-core system, *MOM* consumes 60.17% and 68.01% of the energy consumed by WF and by SOM_{in-out}, respectively. The reason is that *MOM* enables the cores of the system to work at a similar frequency for longer than any *SOM* variant. This can be also observed in Figure 5.7.

As the proposal combines scheduling with partitioning, one could ask which part of the benefits comes from the scheduler and which ones from the partitioner. To discern this aspect, we implemented the RMS scheduler for energy comparison purposes. Looking at Figure 5.7, one can appreciate that benefits mainly come from the partitioner side instead of the scheduler side, since RMS with WF partitioning presents similar results as the EDF with the same partitioner heuristic.

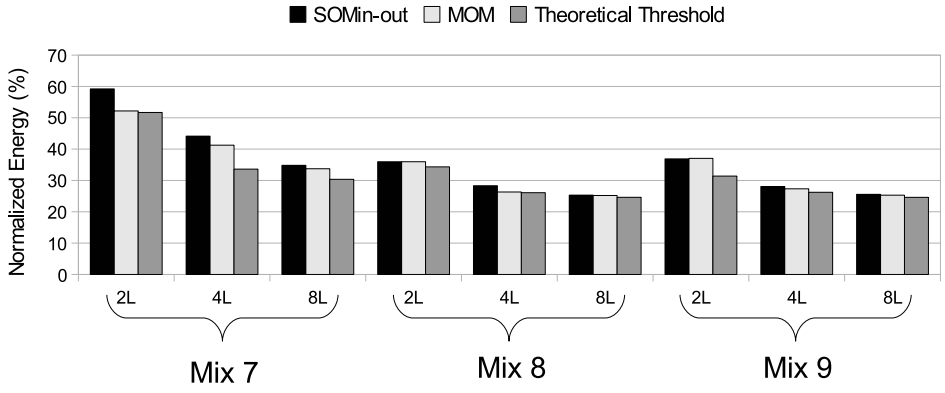
Comparing the working behavior of *MOM* with SOM_{in-out} it can be appreciated that both algorithms perform the same action when a task leaves the system (see Table 5.2).



(a) 2 Cores



(b) 3 Cores



(c) 4 Cores

FIGURE 5.8: SOM_{in-out} versus MOM for different DVFS levels and number of cores.

Therefore, differences in benefits between them come from applying the algorithm at tasks' arrivals. The reason is that SOM_{in-out} first allocates the incoming task and then makes one *migration attempt*, whereas MOM checks for each core which combination of task-to-core allocation plus a single *migration attempt* would achieve a better workload balance. Thus, MOM examines a wider range of possible distributions.

Mix	1	2	3	4	5	6	7	8	9
Average	30.54	24.29	15.32	14.96	14.86	20.28	19.94	13.33	16.06
Standard Deviation	13.86	8.12	5.29	3.39	2.76	4.44	12.07	3.64	10.46

TABLE 5.3: Average and standard deviation of task utilization.

Moreover, in some mixes (e.g., mix 3) *MOM* results fall very close to the energy savings of the theoretical threshold. However, if the utilizations of the tasks in a given mix widely differ among them and depending on how run-time conditions evolve, the results of any practical scheduler may be far from the theoretical threshold (e.g., mix 7, mix 9). The standard deviation of the task utilization (see Table 5.3) in these mixes is relatively high since a few tasks have a huge utilization. This fact prevents *SOM* and *MOM* from achieving a perfect balancing in some scenarios, as done by the theoretical threshold. Notice that mix 1 for a 2-core system also presents a high standard deviation value, but in this case it is due to a single task with much higher utilization. On the other hand, in mix 3 most tasks present similar utilizations within a limited range (10%-17%). Thus, it is more feasible that practical schedulers can obtain a perfect balancing.

Finally, as the number of cores and voltage levels increase (4 cores), a *SOM* algorithm is enough to achieve important energy savings, although *MOM* can slightly improve those results. Moreover, these results fall close to the theoretical maximum. Thus, in this scenario, a possible choice to enhance energy savings is to change the voltage regulator domain (i.e., to implement several regulators, each one shared by a subset of cores).

5.4 Conclusions

Workload balancing has been proved to be an efficient power technique in multicore systems. Unfortunately, unexpected workload imbalances can rise at run-time provided that the workload changes dynamically since new tasks arrive to or leave the system. To palliate this shortcoming, this chapter has analyzed the impact on energy consumption due to task migration strategies in a multicore embedded system implementing DVFS.

Two power-aware migration heuristics working with real-time constraints, namely *SOM* and *MOM* have been devised, which check only one target core or the optimal core before performing a migration, respectively. To prevent excessive overhead, task migration has been strategically applied at three specific execution points of time where the workload

changes: at tasks' arrivals, at tasks' exits, and in both cases. Three variants of *SOM* algorithm are devised depending on the point of time the algorithm applies.

Experimental evaluation has been performed using sets of mixes of real-time benchmarks executed on a modeled ARM11 MPCore processor. A first observation is that applying the algorithm at tasks' exits achieves better energy savings than applying it at tasks' arrivals, but the highest benefits are obtained when the algorithm is applied in both cases. On the other hand, *MOM* performs in general better than *SOM*, however as the number of cores and frequency/voltage levels increases, the differences among energy benefits are reduced. Results show that task migration allows the proposed schedulers to achieve important energy benefits over WF. These benefits are, on average, by 17% and 24% over WF, for *SOM* and *MOM*, respectively. Moreover, in some cases *MOM*'s benefits are up to 40%.

This chapter has also shown how task migration combined with DVFS can allow important energy savings. Thus, benefits come from both techniques. Analyzing the results one can notice that migration is a powerful technique since it allows reducing energy consumption compared to a system with more voltage levels without migration.

A final remark is that improving the workload balance by supporting task migration, not only energy savings can be enhanced, but since the utilization of the most loaded core is also reduced, then also a wider set of tasks could be scheduled.

Chapter 6

Memory Controller Scheduling Policies

6.1 Introduction

Memory requests compete to access to the main memory at the memory controller. In order to satisfy real-time constraints of HRT tasks, a simple memory controller scheduling policy would prioritize requests of these tasks, over SRT tasks. This policy, from now on *HR-first*, will increase the number of deadline misses of SRT tasks. This way negatively impacts on the QoS of SRT applications. Moreover, a high number of SRT deadline misses during peak activity of HRT tasks may leave the system unresponsive for significant periods of time. This situation will be perceived by the user as unreliability and bad performance (e.g., movie blinking) of the device. To overcome this problem, DVFS mechanisms could be used to adjust system performance temporarily, increasing energy consumption and battery discharge rate during peak activity.

This chapter presents a scheduler for memory requests at the memory controller; this policy, referred to as *ATR-first*, is aimed for a multicore multithreaded embedded real-time system with DVFS support. The proposed policy ensures EDF schedulability without sacrificing QoS of non-critical applications. This approach prioritizes only those requests of HRT tasks that are critical to accomplish real-time schedulability. For dynamic scheduling algorithms, such as EDF, this set of tasks varies during execution and

depends on the running workload as well as the computing power of the system (i.e., number of cores and multithreading capabilities of each core).

6.2 Power-Aware Scheduler

Notice that although both HRT and SRT tasks are considered in this chapter, the scheduler must only guarantee HRT task deadlines. Therefore, the task set considered for estimating the target working frequency must include at least all the HRT tasks. Recall that HRT tasks have always more priority than SRT tasks, regardless their deadlines. Of course, the lower the number of SRT task deadline misses, the higher QoS of the system. However, there are situations (e.g., low battery) where maintaining the battery charge is preferable over improving QoS.

In this sense, this work proposes two scheduling policies: *H-mode* and *H+S-mode*. The former considers only HRT tasks for estimating the target machine speed. The latter, considers both HRT and SRT tasks for frequency estimation. As experimental results will show, *H-mode* presents lower energy consumption than *H+S-mode* but at expense of QoS of SRT tasks. On the other hand, *H+S-mode* consumes more energy since higher frequencies are required to meet the deadlines of the larger workload demands. Depending on the situation (e.g., the battery charge level) the system could dynamically select one or another scheduling strategy.

This work evaluates a DVFS with 8, 4 and 2 frequency/voltage levels. With the 8-level configuration (8L) the system can run tasks at the frequencies indicated in Chapter 3. The 4-level (4L) one allows to work at 1700, 1400, 1100 and 600 MHz. Finally, the 2L configuration only supports the extreme frequencies (i.e., 600 and 1700 MHz).

6.3 Memory Controller

The memory controller manages memory requests from all the threads in the system. The modeled memory controller includes advanced features like *load forwarding* and *load bypassing*. That is, if the address of a load from a given thread matches the address of a previous pending store, the requested data is forwarded from the store queue to the load. On the other hand, if no address of a previous store matches the load address, then

the load is allowed to bypass the previous stores. The controller also gives precedence to loads over stores from any thread since stores do not stall the processor activity, so long store latencies do not affect the system schedulability.

In order to support priorities among requests from different threads, as done in [42], different queues have been modeled in the memory controller. This feature is required since a simple FCFS memory-request scheduling policy will cause HRT deadline misses as it introduces unexpected variability in load latencies. The model also assumes as many memory ports as computing cores, avoiding that memory contention afects schedulability of critical HRT tasks. In addition to the proposed *ATR-first*, a more intuitive *HR-first* policy has been implemented for comparison purposes. Below both policies are described.

6.3.1 *HRT Requests First*

In this policy, the memory controller is configured to prioritize requests from HRT tasks over requests from SRT tasks. Requests from the same kind of tasks (HRT or SRT) are ordered according to EDF scheduling. For instance, if a new memory request from an HRT task arrives to the memory controller queue, it will be issued before any pending memory request from SRT tasks but after other requests from HRT tasks with higher priority. This policy can be implemented with the same priority order as the one used by the power-aware scheduler.

A working example of this scheduling policy is shown in Figure 6.1. The requests are labeled in the queue according to the memory operation (load or store) and their relative priority (1 is the highest priority). The background color of the request entry distinguishes HRT requests from SRT ones. As explained above, loads take precedence over stores, so they have the highest priority values. Among loads, those from HRT tasks are prioritized over those from SRT tasks. Regarding stores, they are issued after all the loads, and stores from HRT tasks are issued first. Thus, the older store request in the queue (priority 6) is the last request issued to memory since it is a store from an SRT task.

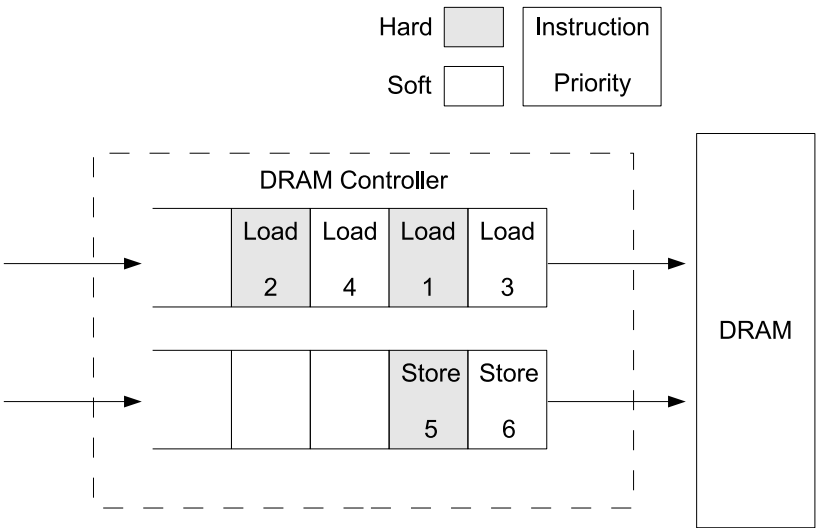


FIGURE 6.1: *HR-first* request scheduling policy.

6.3.2 *Active Task Requests First*

The priority ordering used in *HR-first* prioritizes load requests of HRT tasks. Assuming EDF scheduling, this condition is sufficient to ensure schedulability of HRT tasks but it is not necessary. The necessary condition is that requests from HRT tasks with the closest deadline get the highest priority. From now on, we refer to the task with the closest deadline of a given task queue as the *active* task. This task is mapped to one of the threads of the core associated with the task queue.

Therefore, requests from SRT tasks can be issued before requests from HRT tasks that are not active, so improving their QoS. In this sense, *ATR-first* issues first requests from active HRT tasks, then requests from (active and non-active) SRT tasks, and finally requests from the remaining (non-active) HRT tasks. As in *HR-first*, loads take precedence and requests from the same type of tasks (active HRT, active SRT, non-active SRT, or non-active HRT) are ordered according to EDF.

An example of how this policy works is depicted in Figure 6.2. Memory requests can be classified in two groups depending on whether their source task is active or not. For instance, loads from active tasks in the figure have the highest priorities (1 and 2), and among them, the load from the active HRT task takes precedence. In contrast, the priority of loads from non-active tasks is reversed. That is, the load from the non-active SRT task presents higher priority (3) than the load from the non-active HRT task (4). Finally, here are only two stores in the example, one from a non-active SRT task and

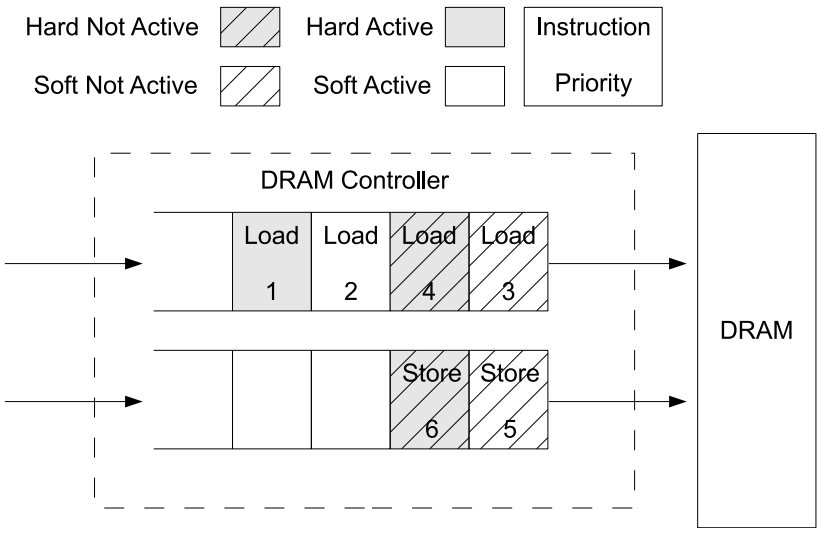


FIGURE 6.2: *ATR-first* request scheduling policy.

another from a non-active HRT task. In this case, the store from the non-active SRT task presents higher priority (5).

6.4 Experimental Results

In order to evaluate the proposal, the Multi2Sim simulation framework has been extended to support the memory controller. In this case, a system with two and four cores supporting three hardware threads per core has been evaluated.

Similar criteria as in previous chapters has been considered to prepare real-time mixes, also using benchmarks from [61] (see Table 6.1). The number of tasks executed in a 2-core system (mixes 1, 2 and 3) ranges from 7 to 15, whereas in a system with four cores (mixes 4, 5 and 6) the number of tasks vary from 20 to 31.

For all the mixes, there is a subset of tasks that are considered HRT tasks. At the maximum speed, the cumulative utilization of this subset ranges from 50% to 95%. Since the HRT task with the closest deadline has the maximum priority, the set of HRT tasks is feasible by the EDF scheduler. However, as the total mix utilization is over 100%, there will be SRT tasks that will not meet their deadlines. This mix design allows us to compare different memory controller policies in terms of number of deadline misses. Experimental results are presented in terms of normalized energy consumption and SRT deadline misses.

Mix	Benchmarks
Mix 1	Cnt, Compress, Expint, Fac, Fft1, Janne_complex, Jfdctint, Ludcmp, Qurt.
Mix 2	Bs, Bsort100, Cnt, Compress, Duff, Expint, Fac, Fft1, Fibcall, Insertsort, Lcdnum, Loop3, Minver, Ns, Statemate.
Mix 3	Bs, Fac, Fibcall, Janne_complex, Lcdnum, Sqrt, Statemate.
Mix 4	Cnt*, Compress*, Crc*, Expint*, Fac*, Fft1*, Janne_complex*, Jfdctint*, ludcmp*, Qurt*.
Mix 5	Cover*, Fdct*, Fir*, Janne_complex*, Jfdctint*, Ludcmp*, Minmax*, Nsichneu*, Qsort-exam*, Qurt*, Select*, Sqrt*.
Mix 6	Bs*, Fac*, Fibcall*, Janne_complex*, Lcdnum*, Sqrt*, Statemate*.

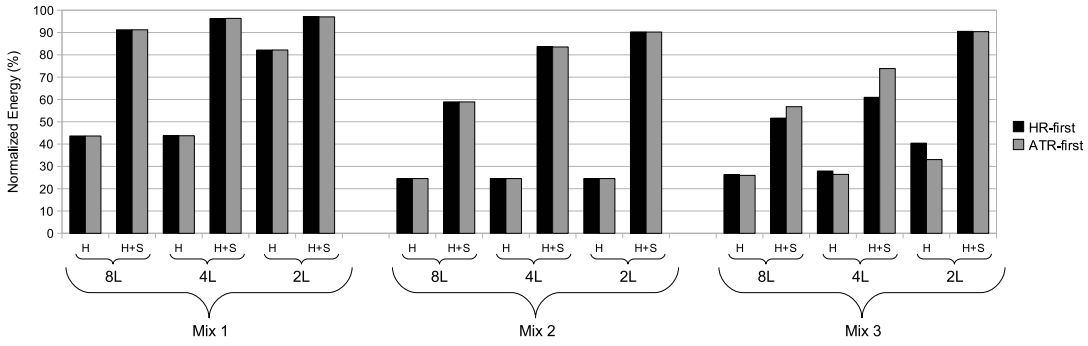
TABLE 6.1: Benchmarks and mixes. Legend: * the benchmark appears more than once in the mix.

Figures 6.3(a) and 6.3(b) show, for different benchmark mixes, the normalized energy consumption and SRT deadline misses of a system with two cores when varying the number of DVFS levels (2L, 4L, and 8L), the task set considered for calculating the working frequency (*H-mode* and *H+S-mode*), and the memory controller policies (*HR-first* and *ATR-first*).

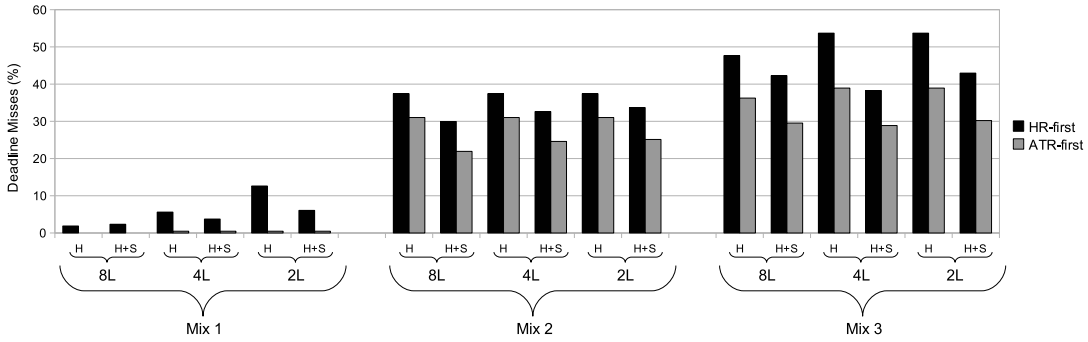
As observed, *ATR-first* does not incur in significant power overheads. That is, the energy consumed by *ATR-first* is similar to *HR-first*, regardless the task set mode and the number of DVFS levels. In fact, the average difference between both policies is only by 1%. A higher energy consumption can be mainly appreciated in mix 3, where the energy consumption increases by 17% for 4-level DVFS in *H+S-mode*. Nevertheless, for 2-level DVFS in *H-mode*, *ATR-first* reduces energy consumption about 22%.

As usual, the number of available DVFS levels has a strong impact on energy consumption. In this sense, the best results are achieved when working with the highest number of DVFS levels since, as mentioned in Subsection 4.3.3, this allows the scheduler to fit better the system speed to the workload requirements.

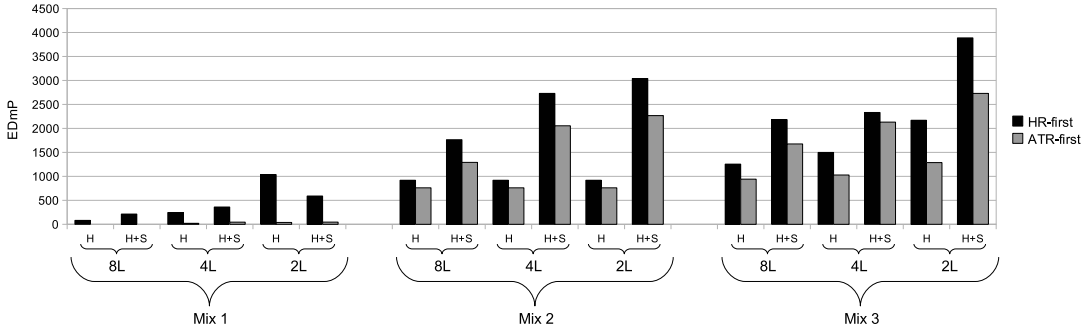
This effect is more pronounced when the system is neither overloaded (most of the time working at the maximum speed) nor underused (most of the time working at the minimum speed). The latter may happen with the *H-mode*, whereas the former with the *H+S-mode*. Note that the *H-mode* always consumes much less energy than the *H+S-mode* since less tasks are taken into account for the frequency calculation. Results show



(a) Normalized energy



(b) Deadline misses



(c) Energy per deadline misses

FIGURE 6.3: Normalized energy and deadline misses for 2 cores.

that working with the eight frequency/voltage levels can reduce power consumption by 43% compared when working with two levels.

Regarding deadline misses, *ATR-first* achieves significant benefits for any frequency calculation mode and DVFS configuration. This proposal reduces on average around 48% SRT deadline misses, reaching in some cases the fulfillment of all deadlines as observed in mix 1 for the 8L system configuration. Moreover, despite choosing the *H+S-mode* consumes more energy, it misses less deadlines than the *H-mode* across all the evaluated mixes.

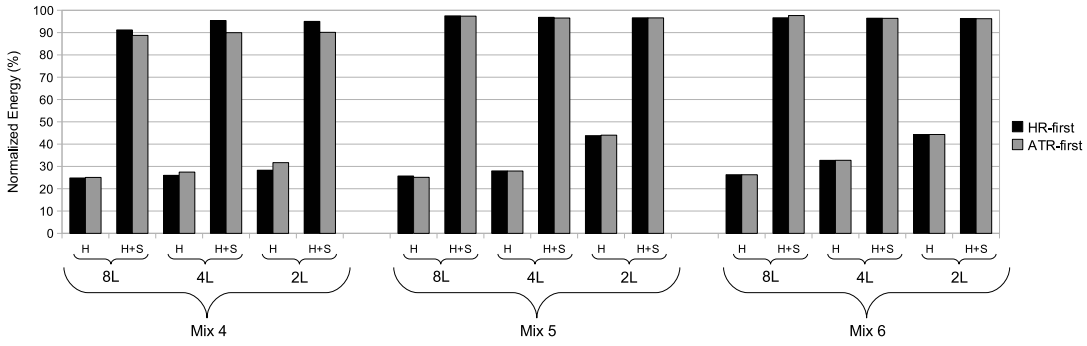
An interesting remark is that *ATR-first* can allow reducing both deadline misses and power consumption, when it is carefully combined with a frequency calculation mode and a given number of DVFS levels. For instance, *ATR-first* misses 31% of the deadlines and consumes by 25% of power of the baseline system for mix 2 in a 4L system with the *H-mode*, whereas for the same mix and number of frequencies using *HR-first* and the *H+S-mode* provides 33% of deadline misses and 84% of power consumption. The former case also gets better results than *HR-first* in the same mix in a 2L system with the *H+S-mode*, which achieves 34% of deadline misses and 90% of power consumption.

In order to evaluate the *efficiency* of the design in terms of energy and QoS we propose the metric *Energy Deadline misses Product* (*EDmP*) as the product of the relative energy consumption and the relative deadline misses. This metric makes sense when energy consumption is pretty much the same or rather close, and provides insights in which is the best scheduling strategy. In other words, when evaluating the *EDmP* there is an upper bound in the QoS that should not be surpassed. For instance, a huge reduction in power consumption would not be acceptable with a 90% of relative deadline misses in normal battery charges. *EDmP* closely resembles the *Energy Delay Product* (EDP) widely used to identify the best hardware design.

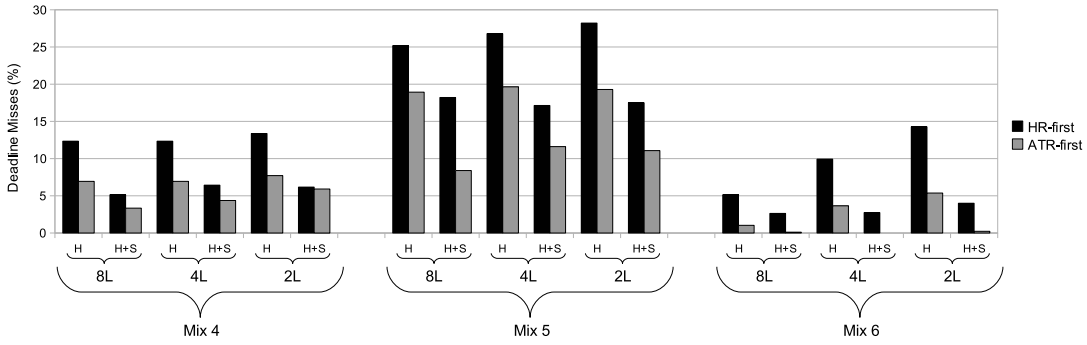
Figure 6.3(c) shows the *EDmP* results of the evaluated schedulers. Notice that the maximum percentage of deadline misses is about 54%, in some cases when *HR-first* is applied. As it can be appreciated, this metric clearly identifies the *ATR-first* policy as the best scheduler design with a noticeable difference with respect to *HR-first*.

Figure 6.4(a) shows the normalized energy consumption and Figure 6.4(b) the number of deadline misses for a system with four cores. As observed, energy values corresponding to the *HR-first* and *ATR-first* memory controller policies are also very close, with an average difference of 0.4%. The differences in normalized energy come again from the speed mode, depending on whether SRT tasks are considered or not, and from the number of DVFS levels. Notice that the *EDmP* metric in a 4-core system (Figure 6.4(c)) also remarks the benefits obtained by *ATR-first* in any case.

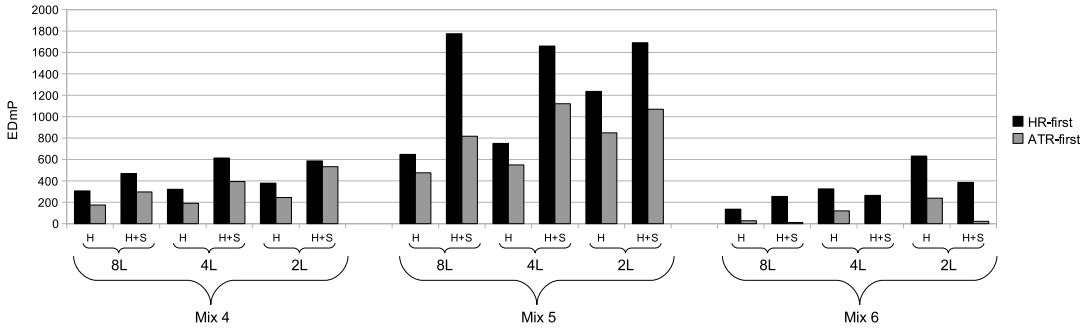
Furthermore, *ATR-first* obtains around 50% reduction in deadline misses, on average, achieving again all deadlines fulfillment in some scenarios, such as in mix 6 in a 4L system with the *H+S-mode*. Based on these results, it can be said that the results obtained by the proposal follow the same trend as in the 2-core system.



(a) Normalized energy



(b) Deadline misses



(c) Energy per deadline misses

FIGURE 6.4: Normalized energy and deadline misses for 4 cores.

6.5 Conclusions

The design of multicore embedded systems with real-time constraints should pursue three main aims: i) to reduce power consumption, ii) to ensure deadlines fulfillment of HRT tasks, and iii) to improve SRT tasks QoS. To this end, memory latencies must be addressed. A key component to handle memory latencies is the memory controller, which manages the access to main memory among requests from the different tasks.

This chapter has proposed the *ATR-first* policy to schedule memory requests in the

memory controller device. This policy relaxes the constraints of a more intuitive *HR-first* policy prioritizing always HRT task requests over SRT ones. In contrast, *ATR-first* prioritizes only the HRT task requests that are crucial to achieve workload schedulability. For the remaining tasks, memory requests from SRT tasks take precedence. This design improves QoS of SRT tasks while ensuring HRT tasks deadlines fulfillment.

Results show that using the proposed *ATR-first* policy consumes pretty much the same power as the *HR-first* policy. In addition, *ATR-first* always reduces the number of SRT deadline misses with respect to *HR-first*. This reduction is on average around 49% and, in some cases, can allow the system to fulfill all of the deadlines.

Finally, to remark that the *ATR-first* policy is shown to be more energy-efficient in all the experiments, as demonstrated by the *Energy Deadline misses Product* metric. To the best of our knowledge, this is the first time a metric measuring the trade-off between energy consumption an deadline misses is proposed.

Chapter 7

Dynamic Execution Time Estimation

7.1 Introduction

Estimating the execution time for a real-time task dynamically can help the system provide better schedulability and energy savings. This fact becomes difficult in current microprocessors working with different frequency levels. An additional problem rises because processor cores and main memory work at different speeds.

Some research works assume that the memory access time (quantified in processor cycles) is constant regardless the processor frequency. We will refer to this model as *Constant Memory Access Time* (*CMAT*) [69]. This model assumes that all the processor components scale their speed at the same pace, which can bring important deviations to estimate the execution time since main memory devices have their own power supply and work independently of the DVFS regulator. To deal with this shortcoming, other researchers have devised alternative models [46, 47] to achieve a better estimation. These models, however, are static and rely on either analyzing the workload source code [46] or performing off-line characterization of the architectural parameters [47].

This chapter presents a model that predicts at run-time the execution time of real-time applications running in multicores supporting different *frequency domains* (i. e., local DVFS), without the need of analyzing any source code or hardware platform. Instead,

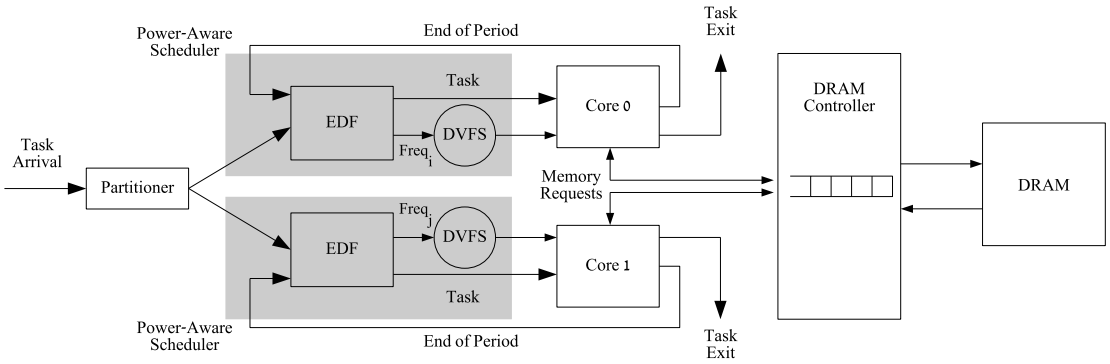


FIGURE 7.1: System model.

the proposed model uses the first hyperperiod to investigate the workload characteristics. Then, this information is used by the power-aware scheduler to choose the most suitable frequency for the following hyperperiods. In this way, not only important energy savings can be achieved but also system schedulability is improved. For instance, if the system utilization decreases and more slack time is available, this extra time could be used to reduce the frequency for energy savings or to introduce additional tasks in the system.

The presented model, referred to as *Processor-Memory* (*Proc-Mem*), predicts the execution time for each task and frequency level. To this end, *Proc-Mem* uses *Performance Monitoring Counters* (PMCs) to measure the time that each core spends performing computation (*CPU*), waiting for memory (*MEM*), and overlapping time (*OVERLAP*) between computation and memory access. Since the overlapping time of a given task depends not only of itself but also on the co-running tasks, the input values of the model must be taken at run-time. The proposed model uses the first hyperperiod to gather the required values. Then, the scheduler uses the model estimates to choose the most suitable working frequency in each active period of the following hyperperiods to address both energy and deadline misses.

7.2 System Architecture

The baseline model has been slightly modified, as shown in Figure 7.1. This modeled system consists of a single-threaded superscalar multicore. These cores can work at a different range of frequency levels, as adopted in recent embedded systems [70].

The working frequency of each core is controlled by a local DVFS regulator [10], that is, cores can run at different speeds. The considered DVFS local regulators implement 7 and 4 frequency levels for core 0 and core 1, respectively. The 7L configuration allows the system to work at all the frequencies ranging from 1.1 GHz to 1.7 GHz in steps of 100 MHz, whereas the 4L (Low-Power) mode permits running tasks at four low frequencies (1.4, 1.3, 1.2 and 1.1 GHz).

7.2.1 Partitioning and Scheduling

The WF algorithm implemented in the partitioner module balances the workload by assigning each incoming task to the least loaded core. Since in the modeled system each core works with a different set of frequency levels, the WF policy must be properly adjusted. For this purpose, the following extension has been adopted. In case that the partitioner detects that core 1 is fully loaded (100% utilization) when working at its maximum working frequency (1.4 GHz), then the new incoming tasks are allocated to core 0, even if this action introduces imbalance into de system, since core 0 can work at higher frequencies than core 1.

The devised schedulers are also in charge of calculating the required target speed of each core according to its utilization. In this sense, the power-aware EDF scheduler implemented in each core chooses the minimum frequency that fulfills the temporal constraints of its task set in order to minimize power consumption.

7.2.2 Memory System

Regarding the memory system, all cores send their memory requests to a common memory controller that handles the accesses to a shared scratchpad memory at the maximum speed (1.7 GHz). A fair memory access scheduling policy has been implemented in the memory controller to minimize inter-core interferences. The memory controller handles its internal request queues according to the FCFS-RR (First-Come First-Served, Round-Robin) policy [71].

The scratchpad memory is composed of eight banks. Bank conflicts are taken into account so that if a bank is being accessed and a younger request demands the same

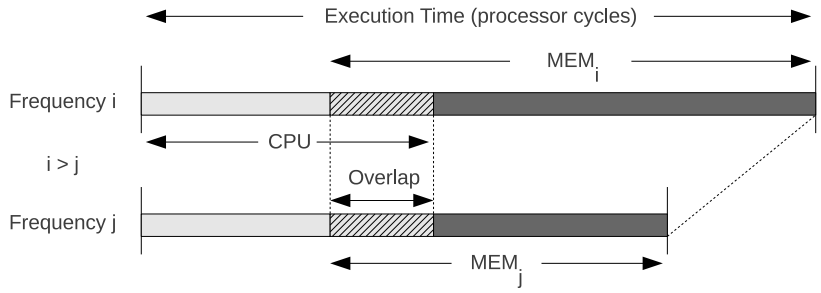


FIGURE 7.2: Execution time model.

bank, this request waits at the memory controller until the previous access finishes. Otherwise, the new request can proceed.

7.3 *Processor-Memory* Model

The execution time of a task can be considered as composed of two main components, (*CPU*) time and memory (*MEM*) time. The former can vary according to data, structural and control hazards presented by the application at the core side, while the latter grows with the number of memory accesses. Both times can overlap since the core can dispatch instructions while the memory is being accessed. This time, from now on referred to as *OVERLAP*, is defined as the time the processor is executing non-dependent instructions while a memory request is being served.

Figure 7.2 depicts a simplified overview of the execution time components. The model assumes that PMCs [72] typically implemented in most current processors, are available in the multicore system. These registers allow the processor to gather multiple variables. This chapter assumes that the variables required by the scheduler can be gathered in the target processor.

Since the *MEM* value is gathered in processor cycles, the smaller the processor cycle time (i.e., the higher the processor frequency) the higher the *MEM* value gathered in the corresponding performance counter. Therefore, the *MEM* value for a target frequency j can be estimated from the *MEM* value collected for the current frequency i as given by Equation 7.1. This effect can be appreciated in Figure 7.2. It also can be appreciated that if the elapsed time is quantified in processor cycles, the *CPU* value remains constant.

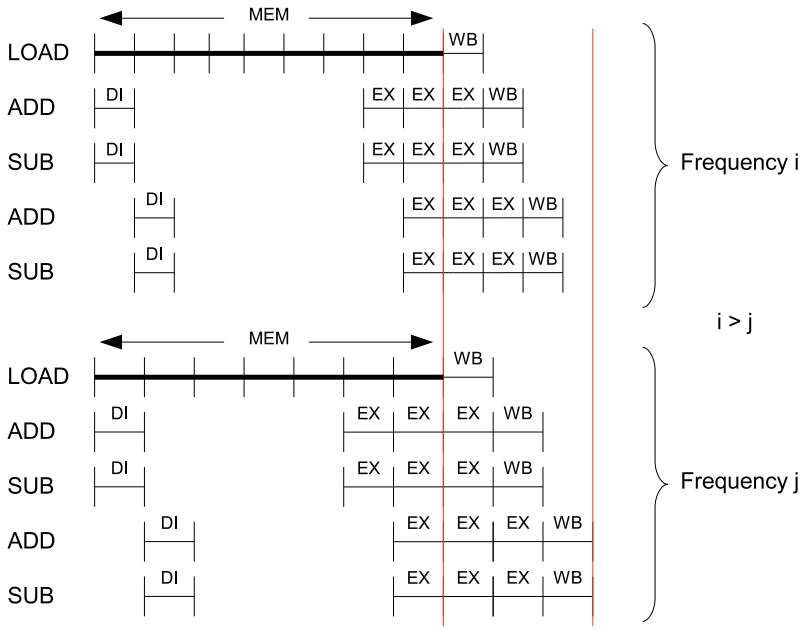


FIGURE 7.3: Execution overlap between processor and memory for two different frequencies in a superscalar architecture.

$$MEM_j = MEM_i \times \frac{Freq_j}{Freq_i} \tag{7.1}$$

On the other hand, in-order processors write the results to the corresponding destination register in program order at the writeback (WB) stage. This means that after a long latency event (e.g. a memory access), the execution (EX) stages of subsequent instructions are delayed to perform the WB stage in program order. The memory latency in current systems, regardless of the working frequency, is typically much longer than that of arithmetic operators. Thus, the number of instructions that can be executed while a previous memory request is being processed remains constant to comply with the WB order. In other words, the $OVERLAP$ time, measured in processor cycles, can be assumed to be constant.

Figure 7.3 illustrates this behavior. It depicts, a possible instructions-time diagram corresponding to the execution of five instructions (one memory reference and four arithmetics) that overlap their execution in a 2-instruction issue width superscalar processor working at two different frequencies i (upper side) and j (lower side). This example assumes 3-cycle latency for arithmetic operations. The number of overlapped cycles is exactly the same in both frequencies due to the part of CPU time overlapping with the

memory access is fixed (all stages DI, two stages EX of the two first arithmetic instructions and one EX stage of the two latter arithmetic instructions). This fact is taken into account in Equation 7.2, which estimates the total execution time in cycles for a given working frequency j. Note that both CPU and $OVERLAP$ values do not depend on the working frequency.

$$Texe_j = CPU - OVERLAP + MEM_j \tag{7.2}$$

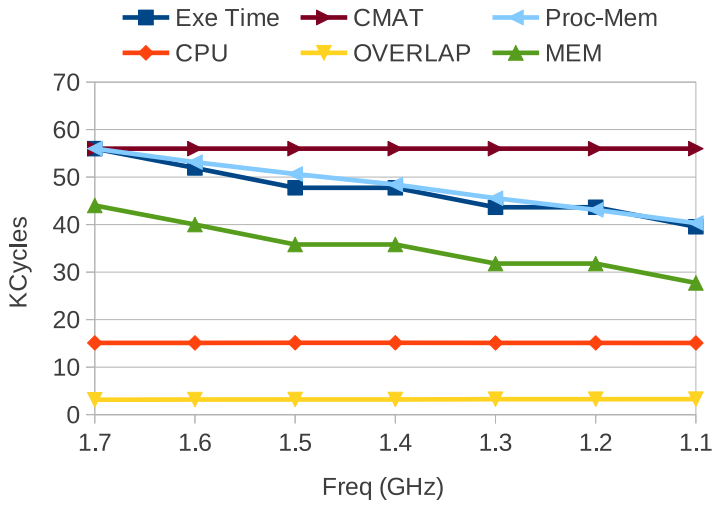
Substituting MEM_j from Equation 7.1 in Equation 7.2, we can derive Equation 7.3, which estimates the overall execution time for any working frequency j as a function of the inputs of the model gathered during the program execution at a different working frequency i.

$$Texe_j = CPU - OVERLAP + MEM_i \times \frac{Freq_j}{Freq_i} \tag{7.3}$$

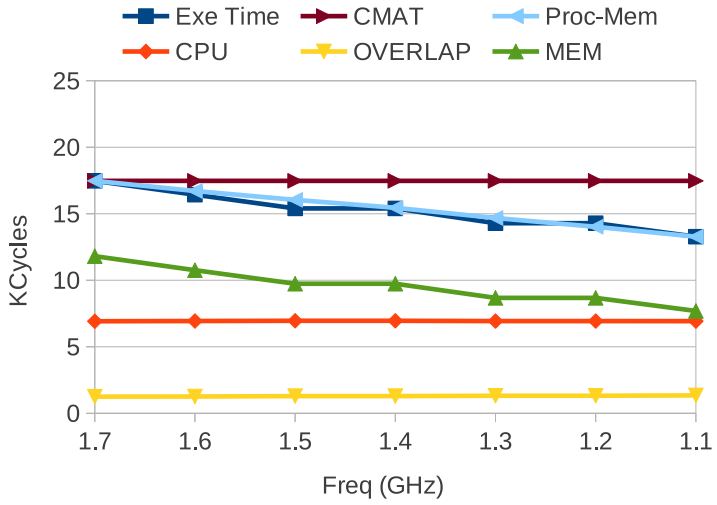
Finally, it is important to note that in multicore processors, where different cores compete for shared resources, new interferences appear. These interferences mainly rise when memory requests from different cores reach the memory controller, where they are scheduled for main memory access. Consequently, the individual execution time (measured either in cycles or in temporal units) of each task increases with respect to stand-alone execution in a single-core system. In spite of this fact, as shown in the next section, the proposed model remains mostly valid in this scenario since the extra time waiting at the memory controller is already included in the MEM component.

7.4 Model Validation

To evaluate the goodness of the model the Multi2Sim simulation framework has been used. The system executes only SRT tasks, and experiments have been conducted with 20 different benchmarks from WCET Malardalen Project [61], although for illustrative purposes, only the results of two representative benchmarks (*fir* and *sqrt*) are presented. These benchmarks are executed in 2-way superscalar cores that include multicycle operators whose latencies range from 1 cycle to 3 cycles.



(a) Fir



(b) Sqrt

FIGURE 7.4: Estimates of the *Proc-Mem* model in stand-alone execution in the single-core superscalar architecture.

Figure 7.4 compares the execution time estimates provided by the *Proc-Mem* and the *CMAT* models for fir (left side) and sqrt (right side) benchmarks. The latter assumes a constant number of processor cycles for each memory access (i.e. the memory speed depends on the DVFS frequency). The stand-alone execution time (labeled as *Exe Time*) of both benchmarks in a single-core superscalar architecture is also represented. Model inputs were taken when running the processor at the highest frequency, that is, at 1.7 GHz. The remaining points of the plot (from 1.6 GHz down to 1.1 GHz) were estimated with the studied models. For comparison purposes, the three main components (*CPU*, *OVERLAP* and *MEM*) of the execution time are represented. As observed, *CPU* and

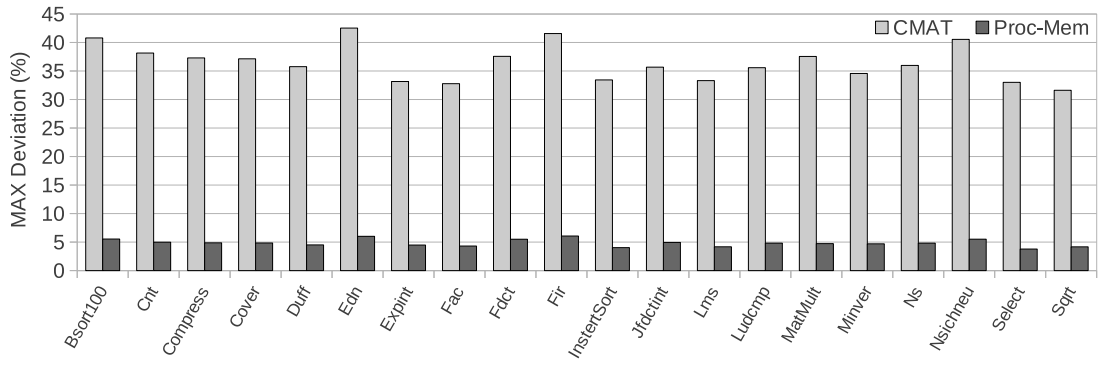


FIGURE 7.5: Maximum deviation in processor cycles in a single-core superscalar architecture.

OVERLAP values keep constant for both benchmarks, while *MEM* increases with the frequency, which illustrates that *Proc-Mem* is suitably designed to model the target system.

Proc-Mem estimates closely follow the execution time regardless the target frequency; in contrast, the error introduced by the *CMAT* model grows as the frequency moves away from 1.7 Ghz. *Proc-Mem* deviation is on average around 2.5% and 1.5% depending on the benchmark, and never exceeds 5.5%. In contrast, the *CMAT* approach incurs in a noticeable higher error, which is already around 7% for both benchmarks in the first 100 MHz away from 1.7 GHz. Moreover, this deviation worsens as the frequency decreases, exceeding 30% in both benchmarks at 1.1 GHz.

The maximum deviation for *Proc-Mem* and *CMAT* accross a relatively wide set of 20 evaluated benchmarks is shown in Figure 7.5. *Proc-Mem* error is always around 5%, while that incurred by *CMAT* is always over 30%.

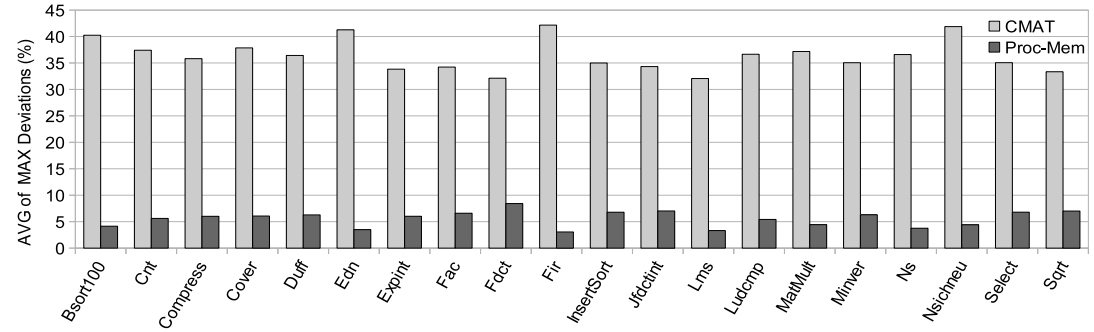
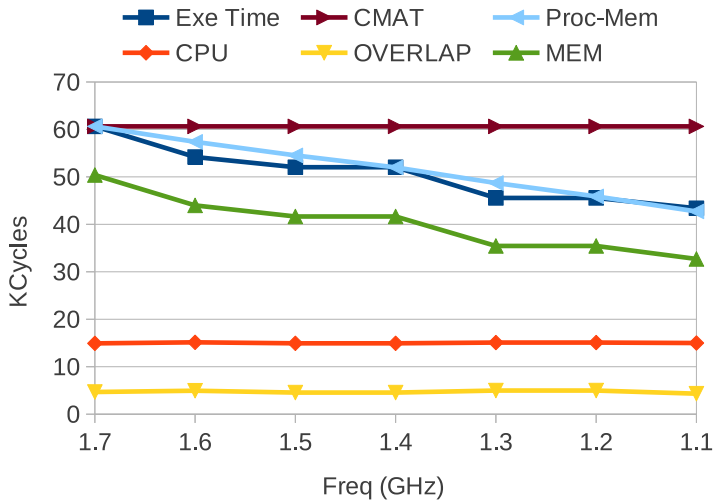
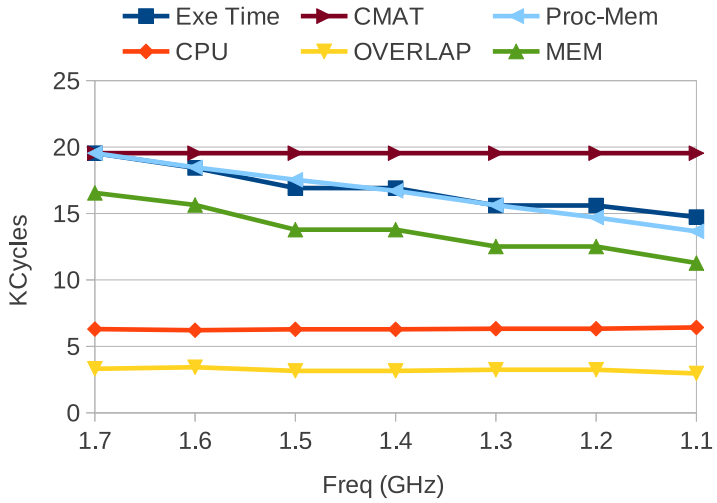


FIGURE 7.6: Average of maximum deviations in a multicore superscalar architecture.



(a) Fir



(b) Sqrt

FIGURE 7.7: Estimates of the *Proc-Mem* model in a multicore processor.

Once it has been proven that the proposal provides good estimates for single-core execution, *Proc-Mem* results are explored in a two-core architecture. To this end, we have performed an exhaustive study with multiple experiments by concurrently executing all the possible pairs of benchmarks for the different working frequencies. In these experiments we assume that both cores work at the same frequency. Figure 7.6 shows the average of the maximum deviations incurred by *CMAT* and *Proc-Mem* for each benchmark across all the experiments. The deviation of *Proc-Mem* for all benchmarks is, on average, 5.6% with respect to the measured execution time, while the deviation of the *CMAT* model is, on average, 36.4%.

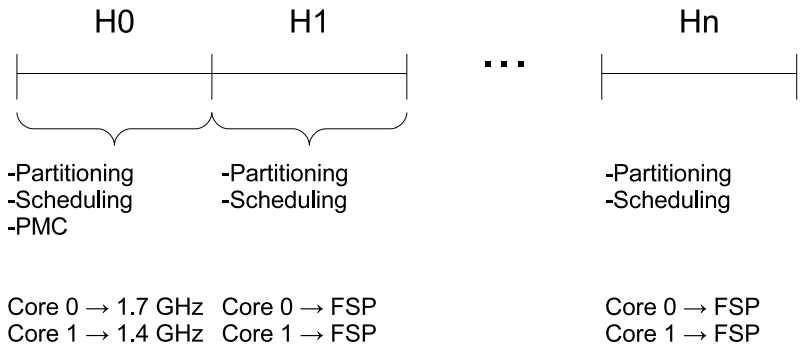


FIGURE 7.8: Power-aware scheduler actions of the system across the hyperperiods.

For illustrative purposes, Figure 7.7 presents the results of the previously studied benchmarks (*fir* and *sqrt*) when executing concurrently for the different frequencies. In this case, the maximum deviation of the *CMAT* model is 39.8% for *fir* and 32.7% for *sqrt*, whereas in the *Proc-Mem* model the maximum deviation is 6.9% and 7.3% for *fir* and *sqrt* respectively. Moreover, the plots confirm the robustness of the proposed model even in the presence of interferences due concurrent execution across all the experiments.

7.5 *Frequency Selection Policy* based on the *Proc-Mem* Model

7.5.1 Scheduler Working Behavior

The *Proc-Mem* model provides estimates of the execution time for the different instances of the tasks in each target frequency. These estimates can be used for several purposes by the scheduler such as to predict the task utilization or to choose the target frequency. As an example, this section illustrates how the estimates provided by the model can help the power-aware scheduler to choose the most suitable DVFS levels to both save energy or address deadline misses. This component of the scheduler will be referred to as *Frequency Selection Policy* or *FSP*. Next, the proposed policy is presented.

The actions performed by the scheduler at run-time using the devised *FSP* are depicted in Figure 7.8 for the studied two-core processor. Actions mainly differ depending on the considered hyperperiod; the first one, namely H0, is used as a sampling period to obtain the workload characteristics to be used by the model in the following hyperperiods.

Mix	Benchmark (instances)				
Mix 1	Bsort100 (1)	Cnt (16)			
Mix 2	Bsort100 (1)	Cnt (90)			
Mix 3	Bsort100 (1)	Cnt (6)	Compress (3)	Cover (3)	
Mix 4	Bsort100 (1)	Cnt (10)	Compress (10)	Cover (10)	
Mix 5	Bsort100 (1)	Cnt (6)	Compress (10)	Cover (16)	
Mix 6	Duff (5)	Edn (1)	Expint (13)	Fac (13)	Fdct (3)
Mix 7	Duff (12)	Edn (1)	Expint (15)	Fac (24)	Fdct (15)
Mix 8	Duff (12)	Edn (1)	Expint (16)	Fac (32)	Fdct (15)

TABLE 7.1: Mix composition: benchmarks and instances of each benchmark.

In hyperperiod H0, PMCs are used to gather the main components of the execution time (*CPU*, *OVERLAP*, and *MEM*). This is done for each active period and task. Each core of the bi-core system is assumed to work at the same frequency during the whole hyperperiod. This initial frequency can be any of the available in the DVFS regulators that ensures meeting all task deadlines, although this example assumes that each core works at its maximum frequency (i.e. core 0 works at 1.7 GHz and core 1 at 1.4 GHz.)

When hyperperiod H0 expires, the inputs required by the *Proc-Mem* model (i.e., *CPU*, *OVERLAP*, and *MEM*) have already been gathered for each active period. Then, the proposed *FSP* is applied to select the target core frequencies to be used in subsequent active periods of the following hyperperiods. The devised policy uses the model to estimate the execution time at each frequency for each task and period. Taking into account these estimates, the policy chooses the lowest frequency for each period that fulfills its deadline while maximizing energy savings. In addition, the obtained estimates are also used by the power-aware scheduler to correct possible deviations in the WCET and calculate tasks utilizations, which are required to perform partitioning and scheduling actions.

7.5.2 Experimental Results

To evaluate the proposed scheduler, it has been compared to a variant using the *CMAT* model in terms of energy and deadline misses. To this end, we have designed eight mixes consisting of benchmarks that are executed multiple times in different active periods across the hyperperiod. Table 7.1 presents the mix composition and the number of instances of each individual benchmark in one hyperperiod. Benchmarks were randomly

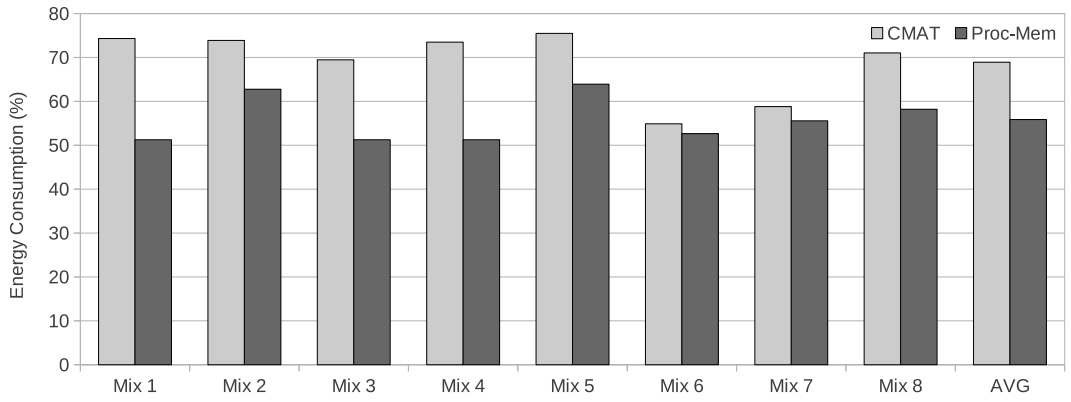


FIGURE 7.9: Normalized energy consumption of *Proc-Mem* and *CMAT* with respect to a system working at the maximum speed.

selected to build the mixes. We designed mixes with different number of tasks (2, 4, and 5 tasks) and different number of active periods per task (varying from 1 up to 90 active periods) to explore the behavior of the proposal in a wide range of scenarios. Notice that we start with few benchmarks in order to have a good control and feedback of the experiment to correct deviations.

Figure 7.9 shows the normalized energy consumption during hyperperiod H1 of both proposals with respect to a system working always at the maximum speed for eigth different mixes of benchmarks. Compared to the *CMAT* model, *Proc-Mem* significantly reduces power consumption across all the mixes. The reason is that *Proc-Mem* estimates the execution time much more accurately than the *CMAT* model, which allows the scheduler to select lower frequencies. Energy reduction is as high as 31% in some cases, and around 18% on average. Notice that in mixes 1, 3 and 4, *Proc-Mem* consumes half the energy of the system working at the maximum speed. The reason is that when executing these mixes with the *Proc-Mem* based scheduler the system runs all the time at the minimum speed available in both local DVFS regulators, whereas when using *CMAT* higher frequencies are used due to higher deviations in the execution time estimates. We found that deviations of *CMAT* grow with the weight of the memory access time over the execution time.

As mentioned above, this work focuses on SRT systems, where deadline misses are allowed. In these systems, a critical issue is that the implemented schedulers tradeoff energy to deadline misses; that is, energy saving must be achieved but guaranteeing a minimum quality of service (deadline misses). Table 7.2 presents the number of misses in

Mix	Deadline Misses		# Active Periods
	CMAT	*Proc-Mem*	
Mix 1	0	0	17
Mix 2	0	2	91
Mix 3	0	0	13
Mix 4	0	0	31
Mix 5	1	1	33
Mix 6	0	0	35
Mix 7	5	5	67
Mix 8	1	1	76

TABLE 7.2: Deadline misses in the *CMAT* and *Proc-Mem* models and active periods of the mixes.

the experiments of the *Proc-Mem* and *CMAT* models for each mix. The number of active periods of the tasks of a mix during a hyperperiod is also included. As observed, in four mixes there is not any deadline miss neither in *Proc-Mem* nor in *CMAT*. Nevertheless, in three of them (i.e., mix 1, mix 3 and mix 4) *Proc-Mem* model energy savings are higher (i.e. by 22%) than when using *CMAT*.

Some deadline misses appear in the remaining mixes, where two cases can be distinguished. On the one hand, the same number of misses rises in mixes 5, 7 and 8. However, in this case, *Proc-Mem* is more energy efficient (by 10%) than *CMAT*. On the other hand, in mix 2, *Proc-Mem* misses two deadlines while no deadline is missed with *CMAT*. Comparing the latter result to performance, *Proc-Mem* misses by 2% of the active periods' deadlines, while savings in energy consumption are by 15%. Therefore, we can conclude that estimating tight execution times can incur in a limited amount of additional deadline misses. However, in this case, important energy savings can be brought.

7.6 Conclusions

Accurately estimating task execution time in a real-time multicore embedded system supporting DVFS is a critical issue for enhancing the schedulability of the system and improving energy consumption. Since this kind of systems support multiple core speeds, different execution times should be estimated, one for each speed. This chapter has proposed the *Proc-Mem* model that estimates the execution times, for each instance of a real-time task, at the available frequencies in the multicore. These estimates are used

by a power-aware scheduler to choose the most suitable working frequency to address both energy and deadline misses.

To provide accurate execution times, *Proc-Mem* uses *Performance Monitoring Counters* (PMCs) to measure the time that each core spends performing computation (CPU), waiting for memory (MEM), and overlapping time ($OVERLAP$) between computation and memory access. Based on this information, *Proc-Mem* estimates the execution times for each task and frequency level.

We have devised a *Frequency Selection Policy* that uses the *Proc-Mem* model to reduce energy consumption while incurring in scarce deadline misses. Compared to the *Constant Memory Access Time* model used in recent works, the use of *Proc-Mem* allows the power-aware EDF scheduler to significantly improve energy savings without significantly increasing the number of deadline misses. Experiments show that the accuracy of the proposed model allows the system to reach energy savings by 18% on average, and up to 31% in some workloads.

Chapter 8

Conclusions

This thesis has proposed several dynamic power-aware techniques to reduce power consumption in real-time multicore embedded systems supporting DVFS where power consumption is a major design issue. Three main issues of the system are covered in this thesis: i) *workload balancing*, ii) *memory controller*, and iii) *execution time estimation*. Regarding *workload balancing*, two aproaches have been studied to help reducing the frequency, and hence, to bring energy savings. First, a partitioning algorithm that increases the overlapping time between CPU and memory has been devised. Second, different partitioning algorithms that allow task migration have been also proposed. In addition, this thesis also has covered the memory access problem at the *memory controller* by proposing two policies that reorder memory requests depending on real-time priorities. Regarding *estimation of execution time*, a model to dynamically estimate the execution time of a task for different frequency levels has been proposed. All these techniques have been implemented and evaluated on top of the extensively used Multi2Sim [56] simulation framework. In this final chapter, the conclusions of this dissertation are analyzed, summarizing the contributions of each chapter. Finally, we also expose the results in terms of scientific publications derived from the work presented in this dissertation.

8.1 Contributions

In Chapter 4, a power-aware partitioner and scheduler for a CGMT multicore processor has been presented. Using DVFS techniques, the scheduler can guarantee HRT tasks'

deadlines and reduce power consumption. The proposed partitioner, *LRB*, distributes the workload attending to the CPU or memory requirements among cores of the system. As a consequence, the overlapping time increases which produces extra slack time that can be used to save energy. Experimental results in HRT environments show that the proposed heuristic can reduce energy consumption compared to the WF heuristic. Regarding the results of the extended heuristic scheduler when executing hybrid tasks composed of HRT and SRT tasks, the required QoS is the key issue to determine both power consumption and deadline misses. That is, a higher number of allowed deadline misses of SRT tasks can help reducing much more power consumption at the expense of increasing the deadline misses and viceversa.

Chapter 5, has introduced two partitioning algorithms, namely *SOM* and *MOM*, that consider task migration. The former checks only one target core, whereas the latter searches the optimal core before performing a migration. Three variants of *SOM* algorithm are devised depending on the point of time the algorithm applies (at tasks' arrivals, at tasks' exits, and in both cases). A first observation is that applying the algorithm at tasks' exits achieves better energy savings than applying it only at tasks' arrivals, but the highest benefits are obtained when the algorithm is applied in both cases. On the other hand, *MOM* performs in general better than *SOM*, however as the number of cores and frequency/voltage levels increase, differences among energy benefits are reduced. Results show that task migration allows the proposed schedulers to achieve important energy benefits over the WF policy. In addition, using task migration to improve the workload balance not only can bring energy savings, but also a wider set of tasks could be scheduled since the utilization of the most loaded core is also reduced.

Chapter 6 has presented two policies for the memory controller. The first and simpler *HR-first* policy prioritizes always scheduling memory requests at requests of HRT tasks over SRT ones. The second and more accurate approach, *ATR-first*, gives priority only to the requests of HRT tasks that are necessary to maintain schedulability. For the remaining tasks, memory requests from SRT tasks take precedence. This design improves QoS of SRT tasks while ensuring HRT tasks deadlines fulfillment. Experimental results show that the power consumption of the system when using either *HR-first* or *ATR-first* is similar. Besides, with *ATR-first* the number of SRT deadline misses is always reduced compared to *HR-first*, allowing the system to fulfill all deadlines in some cases.

Finally, in Chapter 7, the *Proc-Mem* model to estimate the execution time of a real-time task has been discussed. Since a multicore platform supporting multiple core speeds is considered, for each instance of a task different execution times should be estimated, one for each speed. *Proc-Mem* uses *Performance Monitoring Counters* (PMCs) to measure the time that each core spends performing computation (CPU), waiting for memory (MEM), and overlapping time ($OVERLAP$) between computation and memory access. Compared to the *Constant Memory Access Time* model used in recent works, the use of *Proc-Mem* allows the power-aware EDF scheduler to significantly improve energy savings without significantly increasing the number of deadline misses.

8.2 Publications

The following papers related with this dissertation were submitted and accepted for publication in different international journals and conferences.

Journals:

- J. L. March, J. Sahuquillo, H. Hassan, S. Petit and J. Duato. "A New Energy-Aware Dynamic Task Set Partitioning Algorithm for Soft and Hard Embedded Real-Time Systems". *The Computer Journal*. Volume 54. Issue 8. Pages 1282-1294. ISSN: 0010-4620. Oxford University Press. August 2011. (JCR 2^{nd} Quartile)

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "Power-Aware Scheduling with Effective Task Migration for Real-Time Multicore Embedded Systems". *Concurrency and Computation: Practice and Experience*. Volume 25. Issue 14. Pages 1987-2001. ISSN: 1532-0626. Wiley-Blackwell. September 2013. (JCR 2^{nd} Quartile)

Conferences:

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "A Dynamic Power-Aware Partitioner with Task Migration for Multicore Embedded Systems". In *Proceedings of the 17^{th} International European Conference on Parallel and Distributed*

Computing (Euro-Par). Pages 218-229. ISBN: 978-3-642-23399-9. Springer-Verlag. Bordeaux, France. 29 August - 2 September 2011. (CORE A)

- J. L. March, S. Petit, J. Sahuquillo, H. Hassan and J. Duato. "Efficiently Handling Memory Accesses to Improve QoS in Multicore Systems under Real-Time Constraints". In *Proceedings of the 24^{th} International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD). Pages 286-293. ISBN: 978-1-4673-4790-7. IEEE. New York, NY, USA. 24-26 October 2012. (CORE B)

- J. L. March, S. Petit, J. Sahuquillo, H. Hassan and J. Duato. "Dynamic WCET Estimation for Real-Time Multicore Embedded Systems Supporting DVFS". In *Proceedings of the 16^{th} International Conference on High Performance Computing and Communications* (HPCC). Pages 27-33. ISBN: 978-1-4799-6123-8. IEEE. Paris, France. 20-22 August 2014. (CORE B)

- J. L. March, J. Sahuquillo, H. Hassan, S. Petit and J. Duato. "Extending a Multicore Multithread Simulator to Model Power-Aware Hard Real-Time Systems". In *Proceedings of the 10^{th} International Conference on Algorithms and Architectures for Parallel Processing* (ICA3PP), *Workshop on Multicore and Multithreaded Architectures and Algorithms* (M2A2). Pages 444-453. 978-3-642-13135-6. Springer-Verlag. Busan, Korea. 21-23 May 2010. (CORE B)

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "How to Model Real-Time Task Constraints on a High-Performance Processor Simulator". *7^{th} HiPEAC International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems* (ACACES). Pages 301-304. ISBN: 978-90-382-1798-7. Academia Press. Fiuggi, Italy. July 2011.

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "Dynamic Virtual Migration to Reduce Power Consumption in Multicore Embedded Systems". *8^{th} HiPEAC International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems* (ACACES). Pages 241-244. ISBN: 978-90-382-1987-5. Academia Press. Fiuggi, Italy. July 2012

In addition, also in domestic conferences some related papers have been published:

- J. L. March, J. Sahuquillo, H. Hassan, S. Petit and J. Duato. "Ampliación de un simulador de sistemas multinúcleo para la ejecución de tareas de tiempo real con control de consumo". *XXI Jornadas de Paralelismo*. Pages 391-398. ISBN: 978-84-92812-49-3. Ibergarceta Publicaciones. Valencia, Spain. 7-10 September 2010.

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "Real-Time Task Migration with Dynamic Partitioning to Reduce Power Consumption". *XXII Jornadas de Paralelismo*. Pages 185-190. ISBN: 978-84-694-1791-1. Servicio de Publicaciones de la Universidad de La Laguna. Tenerife, Spain. 7-9 September 2011.

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "Balanceo Dinámico con Control de Consumo en Sistemas Multinúcleo de Tiempo Real". *XXIII Jornadas de Paralelismo*. Pages 206-211. ISBN: 978-84-695-4471-6. Servicio de Publicaciones de la Universidad Miguel Hernández. Elche, Spain. 19-21 September 2012.

- J. L. March, S. Petit, J. Sahuquillo, H. Hassan and J. Duato. "Políticas para el Controlador de Memoria en Sistemas Multinúcleo de Tiempo Real". *XXIV Jornadas de Paralelismo*. Pages 30-35. ISBN: 978-84-695-8330-2. Servicio de Publicaciones de la Universidad Complutense de Madrid. Madrid, Spain. 17-20 September 2013.

The following paper has been submitted and it is currently under review process:

- J. L. March, S. Petit, J. Sahuquillo, H. Hassan and J. Duato. "A Dynamic Execution Time Estimation Model to Save Energy in Heterogeneous Multicore Embedded Systems". *Journal of Systems Architecture*. Submitted.

All the works listed above are exclusively related with this thesis. The specific contributions of the Ph.D. candidate reside mostly in the implementation of the system model and the proposed techniques, the setup and execution of all simulation experiments, and the writing of the paper drafts describing the work. Along with these processes, the co-authors have repeatedly provided useful hints and advices, which the Ph.D. candidate has then applied to make the work evolve into its final version. All the conference papers

listed above were presented and defended by the Ph.D. candidate, except [65] in Busan (Korea) and [73] in New York City (USA).

Finally, the acquired skills by the Ph.D. candidate during the development of this work have been also applied at laboratory sessions in the Advanced Computer Architectures Course of the Computer Engineer Degree offered by the School of Computer Engineering at *Universitat Politècnica de València* during 2012-2013 and 2013-2014 academic years.

References

[1] 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):1540–1552, 2008.

[2] 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 Real-Time Systems Symposium*, pages 303–312, Rio de Janeiro, Brazil, 5-8 December 2006. IEEE Computer Society.

[3] R. Ubal, J. Sahuquillo, S. Petit, H. Hassan, and P. López. Power Reduction in Advanced Embedded IPC Processors. *Intelligent Automation and Soft Computing*, 15(3):495–507, 2009.

[4] M. Moncusí, A. Arenas, and J. Labarta. Energy Aware EDF Scheduling in Distributed Hard Real Time Systems. In *Proceedings of the Real-Time Systems Symposium*, pages 101–103, Cancun, Mexico, 3-5 December 2003. IEEE Computer Society.

[5] A. Iyer and D. Marculescu. Power Aware Microarchitecture Resource Scaling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 190–196, Munich, Germany, 13-16 March 2001. IEEE Computer Society.

[6] Q. Zhu and Y. Zhou. Power-Aware Storage Cache Management. *IEEE Transactions on Computers*, 54(5):587–602, 2005.

[7] *Transmeta Crusoe*. Santa Clara, CA, USA, Transmeta Corp. [Online]. Available: http://www.transmeta.com.

[8] *INTEL-XEON*. Santa Clara, CA, USA, INTEL Corp.. [Online]. Available: http://www.intel.com/products/processor/xeon.

[9] *AMD Duron*. Sunnyvale, CA, USA, AMD Corp. [Online]. Available: http://www.amd.com.

[10] J. Donald and M. Martonosi. Techniques for Multicore Thermal Management: Classification and New Exploration. In *Proceedings of the 33rd International Symposium on Computer Architecture*, pages 78–88, Boston, MA, USA, 17-21 June 2006. IEEE Computer Society.

[11] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd. Power7: IBM's Next-Generation Server Processor. *IEEE Micro*, 30(2):7–15, 2010.

[12] 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 International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 335–341, Sydney, Australia, 16-18 August 2006. IEEE Computer Society.

[13] J. López, J. Díaz, M. García, and D. García. Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 25–33, Stockholm, Sweden, 19-21 June 2000. IEEE Computer Society.

[14] A. Burchard, J. Liebeherr, Y. Oh, and S. Son. New strategies for Assigning Real-Time Tasks to Multiprocessor Systems. *IEEE Transactions on Computers*, 44(12): 1429–1442, 1995.

[15] 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*, page 113, Nice, France, 22-26 April 2003. IEEE Computer Society.

[16] T. A. AlEnawy and H. Aydin. Energy-Aware Task Allocation for Rate Monotonic Scheduling. In *Proceedings of the 11th Real Time and Embedded Technology and Applications Symposium*, pages 213–223, San Francisco, CA, USA, 7-10 March 2005. IEEE Computer Society.

[17] C.L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[18] J. A. Stankovic and K. Ramamritham. Editorial: What is Predictability for Real-Time Systems? *Real-Time Systems*, 2(4):247–254, 1990.

[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] T. A. AlEnawy and H. Aydin. Energy-Constrained Scheduling for Weakly-Hard Real-Time Systems. In *Proceedings of the 26th Real-Time Systems Symposium*, pages 376–385, Miami, FL, USA, 6-8 December 2005. IEEE Computer Society.

[21] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu. Power-aware QoS Management in Web Servers. In *Proceedings of the 24th Real-Time Systems Symposium*, pages 63–72, Cancun, Mexico, 3-5 December 2003. IEEE Computer Society.

[22] A. Qadi, S. Goddard, and S. Farritor. A Dynamic Voltage Scaling Algorithm for Sporadic Tasks. In *Proceedings of the 24th Real-Time Systems Symposium*, pages 52–62, Cancun, Mexico, 3-5 December 2003. IEEE Computer Society.

[23] S. Kato and N. Yamasaki. Global EDF-based Scheduling with Efficient Priority Promotion. In *Proceedings of the 14th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 197–206, Kaohisung, Taiwan, 25-27 August 2008. IEEE Computer Society.

[24] H. Chetto and M. Chetto. Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989.

[25] S. Zikos and H.D. Karatza. Performance and Energy Aware Cluster-Level Scheduling of Compute-Intensive Jobs with Unknown Service Times. *Simulation Modelling Practice and Theory*, 19(1):239–250, 2010.

[26] C. McNairy and R. Bhatia. Montecito: A Dual-Core, Dual-Thread Itanium Processor. *IEEE Micro*, 25(2):10–20, 2005.

[27] R. Kalla, B. Sinharoy, and J.M. Tendler. IBM Power5 Chip: A Dual-Core Multithreaded Processor. *IEEE Micro*, 24(2):40–47, 2004.

[28] Agam Shah. *Arm plans to add multithreading to chip design.* ITworld, 2010. [Online]. Available: http://www.itworld.com/hardware/122383/arm-plans-add-multithreading-chip-design.

[29] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing Energy-Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures. In *Proceedings of the SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 169–180, San Diego, CA, USA, 12-16 June 2007. ACM.

[30] 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.

[31] 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 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 213–224, San Francisco, CA, USA, 24-27 September 2005. ACM Press.

[32] Y. Wei, C. Yang, T. Kuo, and S. Hung. Energy-Efficient Real-Time Scheduling of Multimedia Tasks on Multi-Core Processors. In *Proceedings of the 25th Symposium on Applied Computing*, pages 258–262, Sierre, Switzerland, 22-26 March 2010. ACM.

[33] A. Schranzhofer, J.-J. Chen, and L. Thiele. Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms. *IEEE Transactions on Industrial Informatics*, 6(4):692–707, 2010.

[34] B. B. Brandenburg, J. M. Calandrino, and J. H. Anderson. On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study. In *Proceedings of the 29th Real-Time Systems Symposium*, pages 157–169, Barcelona, Spain, 30 November - 3 December 2008. IEEE Computer Society.

[35] A. El-Haj-Mahmoud and E. Rotenberg. Safely Exploiting Multithreaded Processors to Tolerate Memory Latency in Real-Time Systems. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 2–13, Washington, DC, USA, 22-25 September 2004. ACM Press.

[36] Liu Zheng. A Task Migration Constrained Energy-Efficient Scheduling Algorithm for Multiprocessor Real-time Systems. In *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing*, pages 3055–3058, Shanghai, China, 21-25 September 2007. IEEE Computer Society.

[37] E. Brião, D. Barcelos, F. Wronski, and F. R. Wagner. Impact of Task Migration in NoC-based MPSoCs for Soft Real-time Applications. In *Proceedings of the International Conference on VLSI*, pages 296–299, Atlanta, GA, USA, 15-17 October 2007. IEEE Computer Society.

[38] N. Fisher and S. Baruah. The Feasibility of General Task Systems with Precedence Constraints on Multiprocessor Platforms. *Real-Time Systems*, 41(1):1–26, 2009.

[39] R. Giorgi and C. A. Prete. PSCR: A Coherence Protocol for Eliminating Passive Sharing in Shared-Bus Shared-Memory Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):742–763, 1999.

[40] Simon Schliecker, Mircea Negrean, Gabriela Nicolescu, Pierre Paulin, and Rolf Ernst. Reliable Performance Analysis of a Multicore Multithreaded System-on-Chip. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, CODES+ISSS '08, pages 161–166, New York, NY, USA, 2008. ACM.

[41] R. Pellizzoni, A. Schranzhofery, J.-J. Cheny, M. Caccamo, and L. Thiele. Worst Case Delay Analysis for Memory Interference in Multicore Systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 741 –746, March 2010.

[42] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: a Predictable SDRAM Memory Controller. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 251–256, New York, NY, USA, 2007. ACM.

[43] M. Paolieri, E. Quiñones, F.J. Cazorla, and M. Valero. An Analyzable Memory Controller for Hard Real-Time CMPs. *Embedded Systems Letters, IEEE*, 1(4):86 –90, dec. 2009.

[44] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Guillem Bernat, and Mateo Valero. Hardware Support for WCET Analysis of Hard Real-Time Multicore

Systems. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 57–68, New York, NY, USA, 2009. ACM.

[45] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.*, 7(3):36:1–36:53, May 2008.

[46] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-Aware Static Timing Analysis. In *Proceedings of the 24th International Real-Time Systems Symposium*, pages 40–51, Cancun, Mexico, 3-5 December 2003. IEEE Computer Society.

[47] D. C. Snowdon, G. Van Der Linden, and S. M. Petters. Accurate Run-Time Prediction of Performance Degradation under Frequency Scaling. In *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, Pisa, Italy, 4-6 July 2007.

[48] Rustam Miftakhutdinov, Eiman Ebrahimi, and Yale N. Patt. Predicting Performance Impact of DVFS for Realistic Memory Systems. In *Proceedings of the 45th International Symposium on Microarchitecture*, pages 155–165, 2012.

[49] Stefan Schaefer, Bernhard Scholz, Stefan M. Petters, and Gernot Heiser. Static Analysis Support for Measurement-based WCET Analysis. In *In 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Work-in-Progress Session*, 2006.

[50] Ingomar Wenzel, Raimund Kirner, Bernhard Rieder, and Peter Puschner. Measurement-based Timing Analysis. In *Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 430–444, Porto Sani, Greece, 2008.

[51] Christoph Cullmann, Christian Ferdinand, Gernot Gebhard, Daniel Grund, Claire Maiza Burguière, Jan Reineke, Benoît Triquet, and Reinhard Wilhelm. Predictability Considerations in the Design of Multi-Core Embedded Systems. *Proceedings of Embedded Real Time Software and Systems*, 2010. http://www.erts2010.org.

[52] Petar Radojković, Sylvain Girbal, Arnaud Grasset, Eduardo Quiñones, Sami Yehia, and Francisco J. Cazorla. On the Evaluation of the Impact of Shared Resources in Multithreaded COTS Processors in Time-Critical Environments. *ACM Trans. Archit. Code Optim. TACO*, 8(4):34:1–34:25, January 2012.

[53] D. Hardy, T. Piquet, and I. Puaut. Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches. In *30th IEEE Real-Time Systems Symposium, 2009*, pages 68–77, 2009.

[54] H. Shah, A. Raabe, and A. Knoll. Bounding WCET of applications using SDRAM with Priority Based Budget Scheduling in MPSoCs. In *Design, Automation Test in Europe Conference Exhibition*, pages 665–670, 2012.

[55] T. Ungerer, F.J. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiones, M. Gerdes, M. Paolieri, J. Wolf, H. Cass, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability. *Micro, IEEE*, 30(5):66–75, 2010.

[56] R. Ubal, J. Sahuquillo, S. Petit, and P. López. Multi2Sim: 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, Gramado, RS, Brazil, 24-27 October 2007. IEEE Computer Society.

[57] Tom R Halfhill. Intel's Tiny Atom: New Low-power Microarchitecture Rejuvenates the Embedded x86. *Micro Report*, 22(4):1, 2008.

[58] 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, Santa Margherita Ligure, Italy, 22-24 June 1995. IEEE Computer Society.

[59] M. Shah et al. UltraSPARC T2: A Highly-Threaded, Power-Efficient, SPARC SOC. In *Proceedings of the IEEE Asian Solid-State Circuits Conference*, pages 22–25, Jeju, Korea, 12-14 November 2007. IEEE Computer Society.

[60] *Intel Atom Processor Microarchitecture*. Santa Clara, CA, USA, INTEL Corp.. [Online]. Available: http://www.intel.com.

[61] *WCET Analysis Project. WCET Benchmark Programs*. Mälardalen Real-Time Research Center, Västerås, Sweden, 2006. [Online]. Available: http://www.mrtc.mdh.se/projects/wcet.

[62] *Intel Pentium M Processor Datasheet*. INTEL Corp., Santa Clara, CA, USA, 2004. [Online]. Available: http://download.intel.com/support/processors/mobile/pm/sb/25261203.pdf.

[63] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 271–282, Barcelona, Spain, 12-16 November 2005. IEEE Computer Society.

[64] H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. Melhem. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *Proceedings of the 22nd Real-Time Systems Symposium*, pages 95–105, London, UK, 2-6 December 2001. IEEE Computer Society.

[65] J.L. March, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. Extending a Multicore Multithread Simulator to Model Power-Aware Hard Real-Time Systems. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing*, pages 444–453, Busan, Korea, 21-23 May 2010. Springer-Verlag, Berlin.

[66] P. Chaparro, J. González, G. Magklis, Qiong Cai, and A. González. Understanding the Thermal Implications of Multi-Core Architectures. *IEEE Transactions on Parallel and Distributed Systems*, 18(8):1055–1065, 2007.

[67] *Marvell ARMADA^{TM} 628*. Santa Clara, CA, USA, Marvell Semiconductor, Inc. [Online]. Available: http://www.marvell.com/company/press_kit/assets/Marvell_ARMADA_628_Release_FINAL3.pdf.

[68] K. Hirata and J. Goodacre. ARM MPCore; The Streamlined and Scalable ARM11 Processor Core. In *Proceedings of the Conference on Asia South Pacific Design Automation*, pages 747–748, Yokohama, Japan, 23-26 January 2007. IEEE Computer Society.

[69] Marco E. T. Gerards and Jan Kuper. Optimal DPM and DVFS for Frame-based Real-Time Systems. *ACM Transactions on Architecture and Code Optimization*, 9 (4):41:1–41:23, January 2013.

[70] *ARM big.LITTLE Processing*, ARM Holdings. [Online]. Available: http://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[71] E. Ipek, O. Mutlu, J.F. Martinez, and R. Caruana. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *35th International Symposium on Computer Architecture, ISCA*, pages 39–50, 2008.

[72] Stéphane Eranian. What Can Performance Counters Do for Memory Subsystem Analysis? In *Proceedings of ACM SIGPLAN workshop on memory systems performance and correctness (ASPLOS'08)*, pages 26–30. ACM, 2008.

[73] J.L. March, S. Petit, J. Sahuquillo, H. Hassan, and J. Duato. Efficiently Handling Memory Accesses to Improve QoS in Multicore Systems under Real-Time Constraints. In *IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 286–293, New York, NY, USA, 24-26 October 2012. IEEE. ISBN 978-1-4673-4790-7.