Universitat Politècnica de València

Master Thesis

# A Dynamic Power-Aware Partitioner with Real-Time Task Migration for Embedded Multicore Processors

*Author:*
José Luis March Cabrelles

*Advisors:*
Dr. Julio Sahuquillo Borrás
Dr. Salvador V. Petit Martí

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

*Master of Science*

*in the*
Parallel Architectures Group
Department of Computer Engineering

June 2012

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# *Abstract*

Escuela Tècnica Superior de Ingeniería Informática
Department of Computer Engineering

Master of Science

**A Dynamic Power-Aware Partitioner with Real-Time Task Migration for Embedded Multicore Processors**

by José Luis March Cabrelles

A major design issue in embedded systems is reducing the power consumption since batteries have a limited energy budget. For this purpose, several techniques such as Dynamic Voltage and Frequency Scaling (DVFS) or task migration are being used. DVFS circuitry allows reducing power by selecting the optimal voltage supply, while task migration achieves this effect by balancing the workload among cores.

This work focuses on power-aware scheduling allowing task migration to reduce energy consumption in multicore embedded systems implementing DVFS capabilities. To address energy savings, the devised schedulers follow two main rules: migrations are allowed at specific points of time and only one task is allowed to migrate each time.

Two algorithms have been proposed working under real-time constraints. The simpler algorithm, namely, Single Option Migration (SOM) only checks one target core before performing a migration. In contrast, the Multiple Option Migration (MOM) searches the optimal target core.

In general, the MOM algorithm achieves better energy savings than the SOM algorithm, although differences are wider for a reduced number of cores and frequency/voltage levels. Moreover, the MOM algorithm reduces energy consumption as much as 40% over the typical Worst Fit (WF) strategy.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| **EDF** | **E**arliest **D**eadline **F**irst |
| **DVFS** | **D**ynamic **V**oltage (and) **F**requency **S**caling |
| **FF** | **F**irst **F**it |
| **MOM** | **M**ultiple **O**ption **M**igration |
| **MPSoC** | **M**ulti**P**rocessor **S**ystem-**o**n-**C**hip |
| **OS** | **O**perating **S**ystem |
| **PDA** | **P**ersonal **D**igital **A**ssistant |
| **RMS** | **R**ate **M**onotonic **S**cheduling |
| **SMT** | **S**imultaneous **M**ulti**t**hreaded |
| **SOM** | **S**ingle **O**ption **M**igration |
| **WF** | **W**orst **F**it |

# Chapter 1

# Introduction

Embedded systems are an important segment of the microprocessor market as they are becoming ubiquitous in our life. Systems like PDAs, smartphones, and automotive devices, provide an increasing number of functionalities such as navigation, multimedia or gaming, so that computational power is becoming more important every day. However, increasing computational power impacts on battery lifetime. Therefore, a major design concern is power management and optimization [1, 2].

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. Moreover, 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 [3] and IBM Power 7 [4]. Also, leading manufacturers of the embedded sector, like ARM, plan to include multithreading technology in next processor generations [5].

A power management technique that is being implemented in most current microprocessors is Dynamic Voltage and Frequency Scaling (DVFS) [6]. This technique 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). In a multicore system, the DVFS regulator can be shared by several cores, also referred to as *global*, which means that leakage power consumption is mostly the same in all the cores. On the contrary, some systems have a local or *private* DVFS regulator for each

individual core. In the former case, all cores are forced to work at the same speed but less regulators are required so it is a cheaper solution. The latter case could enable more energy savings since each core frequency can be properly tuned to its applications requirements but it is more expensive [7].

On the other hand, energy consumption in systems with a global DVFS regulator can be further improved by properly balancing the workload [1, 8]. To this end, a *partitioner* module is in charge of distributing the workload (i.e., the set of tasks) according to a given algorithm, such as the Worst Fit (WF) or First Fit(FF) [9], that selects the target core to run each task. Unfortunately, the nature of some workload mixes prevents the partitioner from achieving a good balance. To deal with this drawback some systems allow tasks to migrate and move their execution from one core to another, which results in energy saving improvements.

In this work, two algorithms allowing task migration to reduce energy consumption in multicore embedded systems with real-time constraints implementing DVFS capabilities 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 at specific points of time because analyzing all the possible task migrations may result in a prohibitive overhead; and (ii) only one task is allowed to migrate each time. This work focuses on multicore processors where the scheduler includes a partitioner module to distribute tasks among cores. This partitioner 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 studying the impact of the point of time when the algorithm is applied, three variants of the SOM algorithm have been devised, depending on when the scheduler acts: at each task arrival ($SOM_{in}$), when a task leaves the system ($SOM_{out}$), and in both cases ($SOM_{in-out}$).

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 realistic number of cores: two, three and four cores. Some examples are the bi-core Intel Atom [10], the tri-core Marvell ARMADA$^{TM}$ 628 [11] or the quad-core

ARM 11 MPCore [12]. On the other hand, this work assumes a relatively wide number of frequency/voltage levels (up to eight) in order to approach the results to real systems.

Experimental results show that applying the algorithm at tasks exits can achieve better energy savings than applying only at tasks arrivals, but the highest benefits are obtained when the algorithm is applied in both cases. In addition, the MOM algorithm achieves better energy savings than the SOM algorithm. Differences are wider for a reduced number of cores and frequency/voltage levels. Both algorithms show that migration allows achieving important energy benefits. These benefits are, on average, as much as 17% and 24% for the SOM and MOM algorithms, respectively, over the WF algorithm. An interesting observation is that global DVFS regulators minimize differences among the scheduling strategies for a high number of cores and frequency/voltage levels; showing that, in such a case, SOM achieves many times energy savings close to an idealized scheduler.

The remainder of this work is structured as follows. Chapter 2 discusses the related research on energy management and task migration for embedded systems. Chapter 3 describes the modeled system, including the partitioner and the power-aware scheduler. Chapter 4 presents the proposed workload partitioning algorithms. Chapter 5 analyzes experimental results. Finally, Chapter 6 presents some concluding remarks.

# Chapter 2

# Related Work

Scheduling in multiprocessor systems can be performed in two main ways depending on the task queue management: *global scheduling*, where a single task queue is shared by all the processors, or *partitioned scheduling*, which uses a private queue for each processor. The former allows task migration by design since all the processors share the same task queue. In the latter case, the scheduling in each processor can be performed by applying well-established uniprocessor algorithms such as EDF (Earliest Deadline First) or RMS (Rate Monotonic Scheduling). An example of a modern global scheduling proposal can be found in [13].

## 2.1 Partitioned Scheduling

In the *partitioned scheduling* case, research can focus either on the partitioner or the scheduler. Acting in the partitioner, recent works have addressed the energy-aware task allocation problem [9, 14, 15]. For instance, Wei et al. [14] reduce energy consumption by exploiting parallelism of multimedia tasks on a multicore platform combining DVFS with switching-off cores.

Aydin et al. [15] showed that the problem of minimizing energy-consumption on partitioned systems remains NP-Hard even when the feasibility is guaranteed a priori. They also stated that tasks with large utilization values must be allocated to separate processors in the optimal solution for the load balancing problem. Therefore, they proposed an

algorithm that reserves a subset of processors for the execution of tasks with utilization not exceeding a threshold.

Schranzhofer et al. [16] presented a method for allocating tasks to a MultiProcessor System-on-Chip (MPSoC) platform, aimed at minimizing the average power consumption. A dynamic mapping strategy is devised, where static mappings for scenario sequences are computed and stored as templates on the system. Then, a manager observes mode changes at runtime and chooses an appropriate precomputed template to assign newly arriving tasks to processing units. However, their application is modeled without considering timing constraints.

Unlike this work, none of the previous techniques analyzes the power benefits of task migration among cores.

In [9] the problem of energy minimization for periodic preemptive hard real-time tasks scheduled with RMS in identical multiprocessor platform is proposed. That work uses static priorities based on the RMS, which is not a suitable scheduler for a dynamic system like the one proposed in this work where tasks enter and leave dynamically the system. In contrast to EDF, which is used in our proposal, for RMS no polynomial-time nor exact feasibility tests exist as of today (i.e. one that provides necessary and sufficient conditions for feasibility). Moreover, the algorithms in [9] run in pseudo-polynomial time, which increases the overhead.

## 2.2 Task Migration

Some proposals dealing with task migration can be found in the literature. Brandenburg et al. [17] evaluate global and partitioned scheduling algorithms in terms of scalability based upon empirically-derived overheads. They conclude that each tested algorithm proved to be a viable choice for some subset of the workload categories considered. However, power consumption was not investigated.

In [18], 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. Nevertheless, his work does not consider dynamic workload changes, instead, all tasks are assumed to arrive at the same time, so migrations can be scheduled off-line.

In [19] Brião et al. analyze how migrating soft tasks affects NoC-based MPSoCs in terms of deadline misses and energy consumption. They assume a model of task migration where the task is stalled and then all its code and data are transferred through network links. Besides, task migration is triggered only when the allocation heuristic is executed to balance the system, that is, when a new load of tasks is available. However, they focus on non-threaded architectures.

Seo et al. [7] present a dynamic repartitioning algorithm with migration to balance the workload and reduce consumption. Their work differs from this in that 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.

## 2.3 Scheduling in Multicore and Multithreaded Systems

Regarding the scheduler, in [20] the authors virtualize a Simultaneous Multithreaded (SMT) processor into multiple single-threaded superscalar processors with the aim of combining high performance with real-time formalism. A simple real-time scheduling approach concentrates scheduling within a small time interval, producing a simple repeating space/time schedule that orchestrates virtualization. Experiments show that more task sets are provably schedulable on their proposal than on conventional rigid multiprocessors with equal aggregate resources, and the advantage only intensifies with more demanding task sets.

In order to improve real-time tasks predictability, Cazorla et al. [21] devise an interaction technique between the Operating System (OS) and an SMT processor. Their approach enables the OS to run time-critical jobs without dedicating all internal resources to them so that non-time-critical jobs can make significant progress as well and without significantly compromising overall throughput.

Fisher and Baruah [22] 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 under global EDF scheduling.

In [23], the authors propose a methodology for abstracting the computing power available on a multicore platform by a set of virtual processors in order to allocate real-time tasks. The set of tasks is partitioned into a set of subgraphs that are selected to minimize either the required computational power or the number of cores. Their task model considers precedence relations while this work considers independent tasks and dynamic workload changes. Besides, authors do not undertake the power consumption issue, which is the main focus of this research by using DVFS. Moreover, they focus on a theoretical analysis instead of modeling a detailed cycle-by-cycle multicore architecture as this work presents.

# Chapter 3

# System Model

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. In this scheme, the scheduler is in charge of adjusting the working frequency of the cores in order to satisfy the workload requirements. Figure 3.1 shows a block diagram of the modeled system.

Processor cores, modelled as an ARM11 MPCore, implement the coarse-grain multi-threading paradigm that switches the running thread when a long latency event occurs (i.e., a main memory access). Thus, the running thread issues instructions to execute while another thread is performing the memory access, so overlapping their execution. In the modeled system, the issue slots are always assigned to the thread executing the task with the highest real-time priority. If this thread stalls due to a long latency memory event, then the issue slots are temporarily reassigned until the event is resolved.

## 3.1 Real-Time Task Behavior

The system workload executes periodic hard 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 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

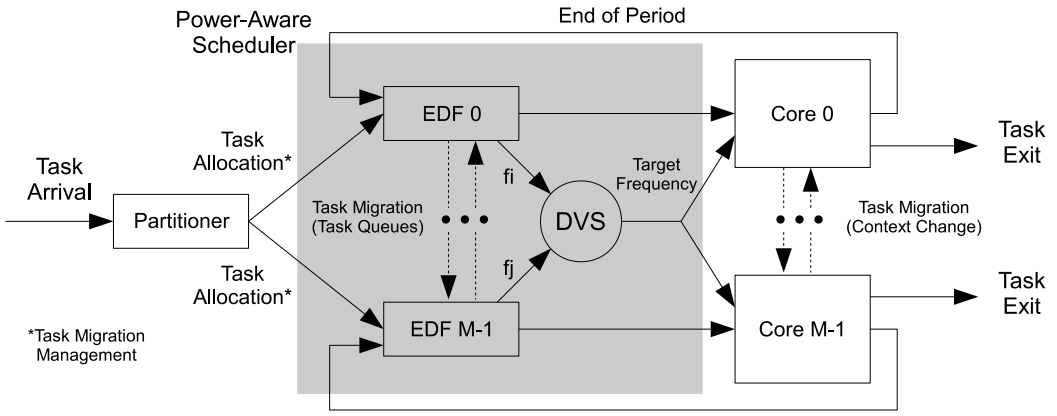



FIGURE 3.1: Modeled system.

execute since they are not active (i.e., inactive periods). In short, a task arrives to the system, executes several times repeatedly, leaves the system, remains out of the system for some periods, and then it enters the system again. This sequence of consecutive active and inactive periods allows modelling real systems with mode changes.

Besides its period and deadline, a task is also characterized by its Worst Case Execution Time (WCET). The task utilization is obtained as $U = \frac{WCET}{Period}$ and is used by several schedulers and partitioners to check whether schedulability of the task set is feasible or not.

## 3.2 Power-Aware Scheduler

Once a task is allocated to a core, it is inserted into the task queue of that core, where incoming tasks are ordered according to the EDF policy, which prioritizes the tasks with the closest deadlines. Thus, the tasks with the closest deadlines will be mapped into the hardware threads of cores.

The scheduler is also in charge of calculating the required target speed of each core according to the tasks requirements. In this sense, the EDF scheduler of each core choses the minimum frequency that fulfills the temporal contraints of its task set in order to minimize power consumption. This information is sent to the global DVFS regulator that selects the maximum frequency/voltage level among the requested by the EDF schedulers.

TABLE 3.1: Frequency (F) vs Power (P).

| F[MHz] | 1700 | 1500 | 1400 | 1300 | 1200 | 1100 | 900 | 600 |
|---|---|---|---|---|---|---|---|---|
| P[Watts] | 24.5 | 24.5 | 22 | 22 | 12 | 12 | 7 | 6 |

The target frequencies are recalculated only when 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.

Different frequency values are considered for the power-aware scheduler, based on the frequency levels of a Pentium M [24] which are shown in Table 3.1. This work evaluates the benefits of a DVFS with 8, 4 and 2 frequency/voltage levels. The 8L configuration allows the system to work at all the frequencies indicated in the table, 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). In addition, the overhead of changing the frequency/voltage level has been modeled according to a voltage transition rate of $1mv/1\mu s$ [2].

# Chapter 4

# Partitioning Heuristics

There are several partitioning heuristics that can be used to distribute tasks among cores as they arrive to the system. The Worst Fit (WF) partitioning heuristic is considered as one of the best choices in order to balance the workload [9], so 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 deal with 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, Single Option Migration (SOM) policies are devised in the next section.

## 4.1 Single Option Migration Policies

Figure 4.1 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 around 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

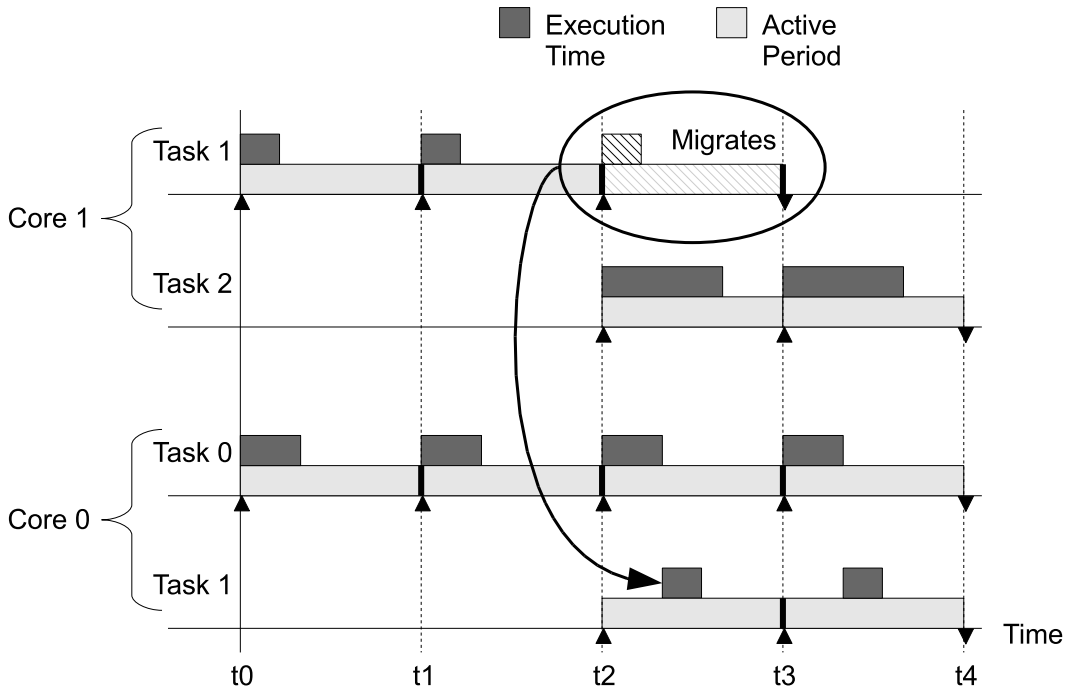



FIGURE 4.1: Example of task migrations to balance the system.

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 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. To avoid performing an excessive number of migrations, which could lead to an unacceptable overhead, the number of migrations is limited to only one which can be performed when a task arrives to or leaves the system.

Figure 4.2 illustrates the devised Migration Attempt (MA) algorithm. This algorithm calculates the imbalance by subtracting the utilization of the least loaded core from the

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 4.2: Migration Attempt algorithm.

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 the two cited 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.

## 4.2 Multiple Option Migration Dynamic Partitioner

This section presents the Multiple Option Migration (MOM) dynamic partitioner algorithm, which applies both at tasks arrivals and exits. When a task arrives to the system, MOM selects the target 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 4.3) 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 for the most loaded core ($U_{min}$ variable in the figure). Notice that all the possible assignments include a migration attempt according to the MA algorithm discussed above.

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 4.3: Multiple Option Migration dynamic partitioner algorithm.

Finally, the task assignment that provides the best overall balance is applied and the algorithm continues with the next task.

Figure 4.4 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 different cores, and then choses the best option in terms of workload balance.

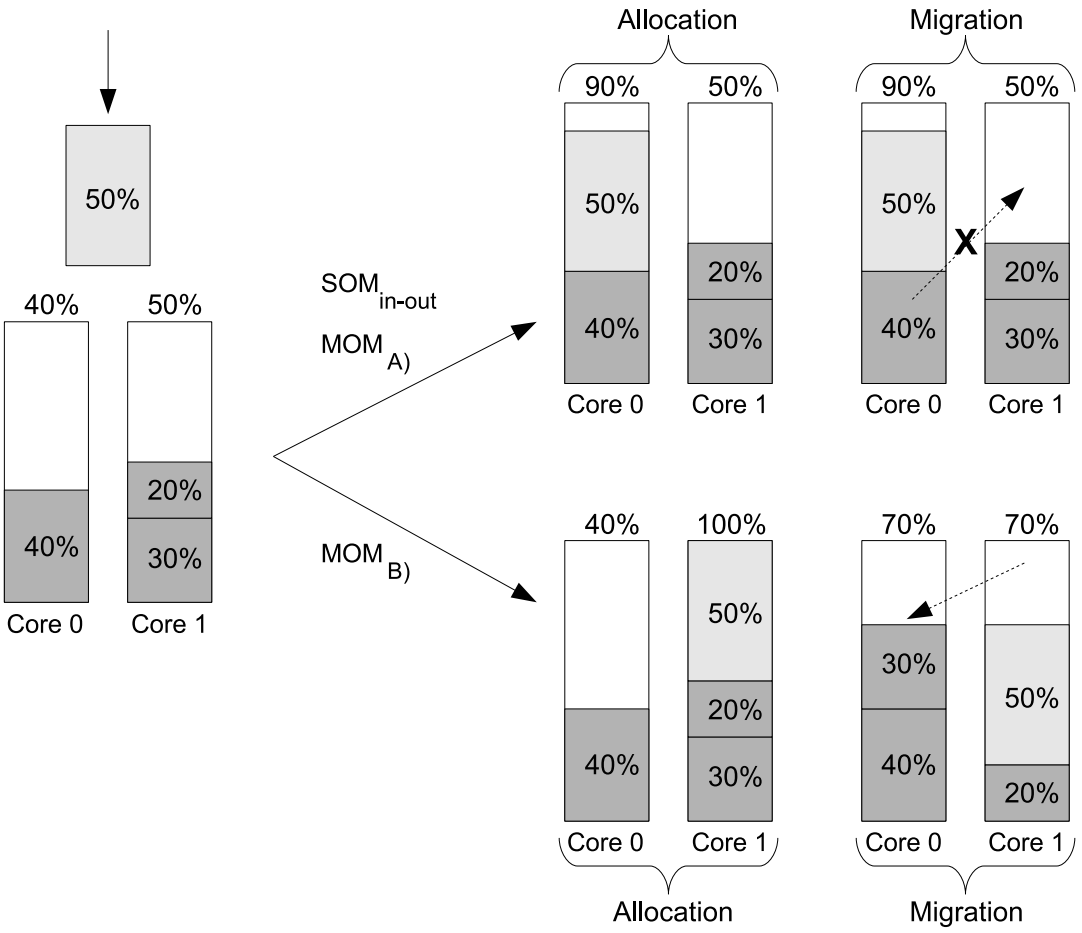



FIGURE 4.4: $SOM_{in-out}$ vs MOM working example.

# Chapter 5

# Experimental Results

Experimental evaluation has been conducted on Multi2Sim [25], a cycle-by-cycle execution driven simulation framework for evaluating multicore multithreaded processors, which has been extended to model the system described in Chapter 3. This Chapter evaluates a multicore processor with two, three and four cores, implementing three hardware threads each. Internal core features have been modeled like an ARM11 MPCore based processor, modified to work as a coarse-grain multithreaded processor with in-order execution, two-instruction issue width, and a 34-cycle memory latency. Regarding the migration overhead, a 10.000 cycles penalty has been assumed [26]. This penalty is applied each time a running context switches its execution to another core.

Since some time is needed to overcome the voltage difference between two different DVFS levels, frequency changes are not instantaneous. 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 taken into account.

Table 5.1 shows the benchmarks from [27] that have been used to prepare real-time workload mixes (a mix number with an asterisk means that the benchmark is used in the mix more than once). Each mix is composed of a set of benchmarks whose number ranges from 7 to 34, running concurrently depending on the number of cores executed. 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. These mixes have been designed considering aspects such as task utilization, task periodicity, and the sequence of active

TABLE 5.1: Benchmark description and mixes.

| Name | Function Description | Mix |
|---|---|---|
| bs | Binary search for a 15-element array | 1, 3, 4*, 9* |
| bsort100 | Bubblesort program | 3 |
| cnt | Counts non-negative numbers in a matrix | 2, 3, 4, 7* |
| compress | Data compression program | 2, 3, 4, 7* |
| cover | Program for testing many paths | 5*, 8* |
| crc | Cyclic redundancy check on 40-byte data | 7* |
| duff | Copy 43-byte array | 3, 4* |
| edn | FIR filter calculations | 7* |
| expint | Series expansion for integral function | 2, 3, 4*, 7* |
| fac | Factorial of a number | 1, 2, 3, 4*, 7*, 9* |
| fdct | Fast Discrete Cosine Transform | 5, 8* |
| fft1 | 1024-point Fast Fourier Transform | 2, 3, 4*, 7* |
| fibcall | Simple iterative Fibonacci calculation | 1, 3, 4*, 6*, 9* |
| fir | Finite impulse response filter | 5, 8* |
| insertsort | Insertion sort on a reversed array of size 10 | 3, 4* |
| janne_complex | Nested loop program | 1, 2, 5*, 7*, 8*, 9* |
| jfdctint | Discrete-cosine transformation | 2, 5*, 7*, 8* |
| lcdnum | Read ten values, output half to LCD | 1, 3, 4*, 6*, 9* |
| loop3 | Function with diverse loops | 3, 6* |
| ludcmp | LU decomposition algorithm | 2, 5*, 7*, 8* |
| minmax | Minimum and maximum functions | 5*, 6*, 8* |
| minver | Inversion of floating point matrix | 3, 4* |
| ns | Search in a multi-dimensional array | 3 |
| nsichneu | Simulate an extended Petri Net | 5*, 8* |
| qsort-exam | Non-recursive version of quick sort algorithm | 5*, 8* |
| qurt | Root computation of quadratic equations | 2, 5*, 7*, 8* |
| select | Nth largest number in a floating point array | 5*, 6*, 8* |
| sqrt | Square root function | 1, 5*, 6*, 8*, 9* |
| statemate | Automatically generated code | 1, 3, 4*, 9* |

Legend: * the benchmark appears more than once in the mix.

and inactive periods. Task periods range from 100.000 to 18.000.000 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. The global system utilization varies in a single execution from 35% to 95% in order to test the algorithms behavior across a wide range of situations. In addition, all results are presented and analyzed for a system implementing two, four and eight voltage levels.

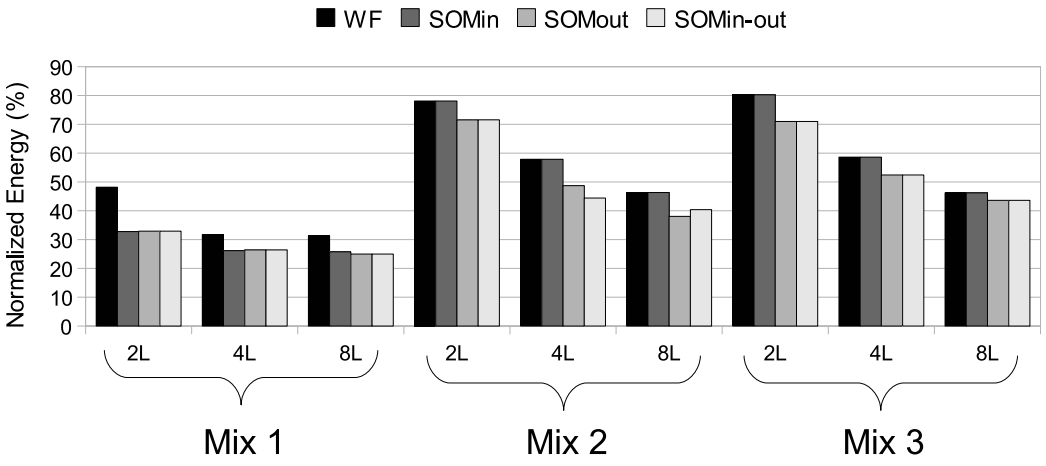
## 5.1 Impact of Migrations at Specific Points of Time

This section analyzes the three devised Single Option Migration 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 5.1 shows the relative energy consumption compared to the energy consumed by the system working always at the maximum speed for different benchmark mixes, DVFS configurations, and number of cores. The results are obtained by multiplying the number of cycles working at each frequency by the energy required per cycle at that frequency.

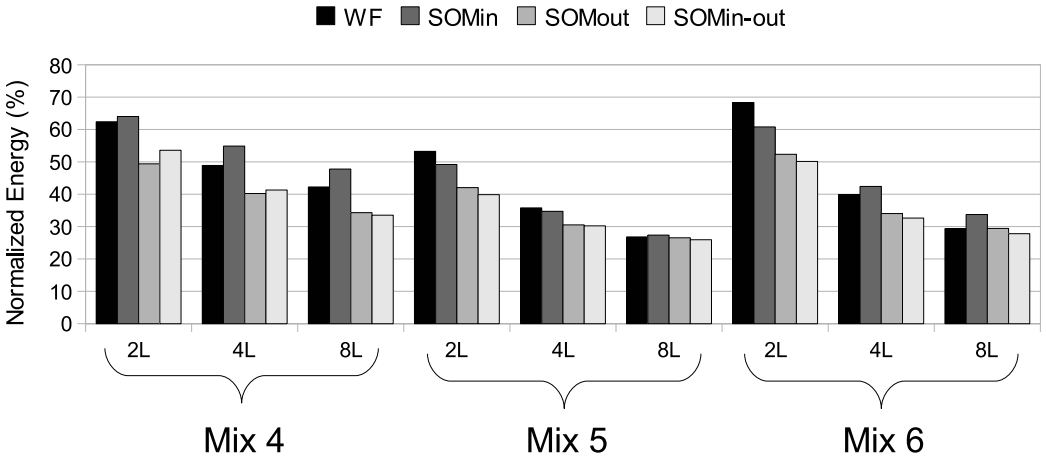
As it can be observed in the results of the 2-core system (Figure 5.1(a)), migration can provide important energy savings with respect to no migration (WF). For instance, for mix 2 in the 4L case with task migration, both when a task arrives to and leaves the system, the energy consumption can be reduced by up to 23.27% compared to the execution without migration.

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 5.1(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 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.

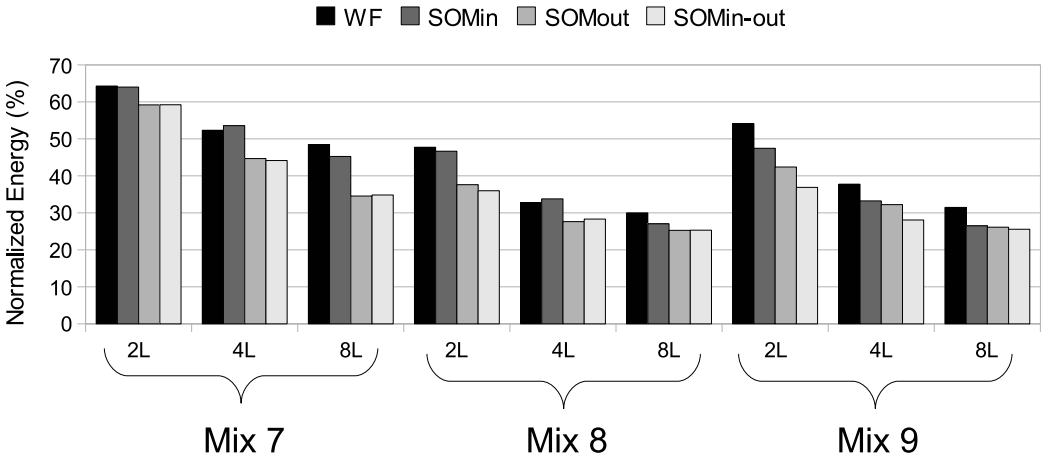
Figure 5.2 illustrates an example. At *time t0*, tasks *T1*, *T2*, and *T3* arrive to the system, and the scheduler selects the frequency/voltage level that best fits the workload requirements. Lets assume that the workload is perfectly balanced in a 2-core system. Then, at *time t1*, task *T1* leaves the system, so workload imbalance will rise (dashed area) in algorithms like WF or $SOM_{in}$ where no migration is performed, 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 happens only on tasks arrivals




(a) 2 Cores




(b) 3 Cores




(c) 4 Cores

FIGURE 5.1: Single Option Migration variants comparison for different DVFS levels and number of cores.

TABLE 5.2: Algorithms action on workload changes.

| 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 |

in WF and $SOM_{in}$ (in *t3*). This drawback is solved in the algorithms which allow migration when a task leaves the system 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 avoiding energy wasting. Figure 5.3 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

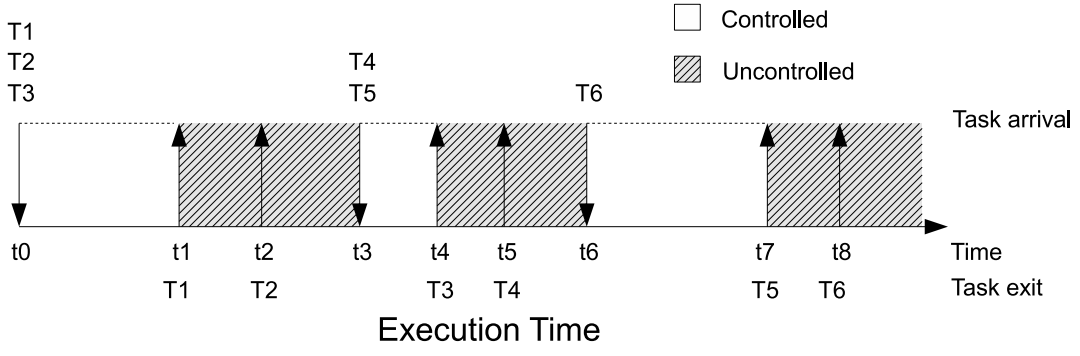



FIGURE 5.2: Effective action of the $SOM_{in}$ partitioning algorithm.

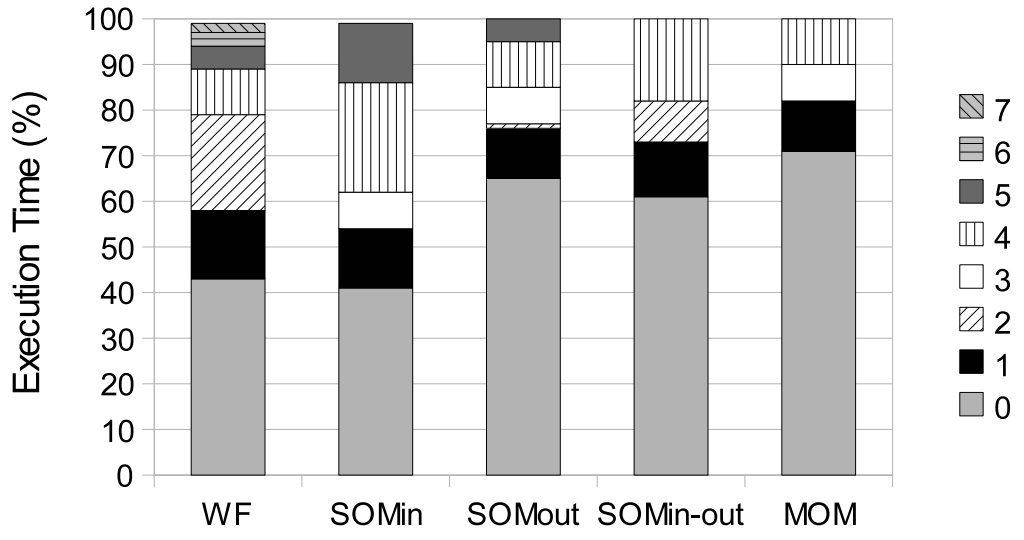



FIGURE 5.3: Differences of the required frequencies.

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 algorithm *controls* the system both at tasks arrivals and exits.

Another interesting remark is that 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%.

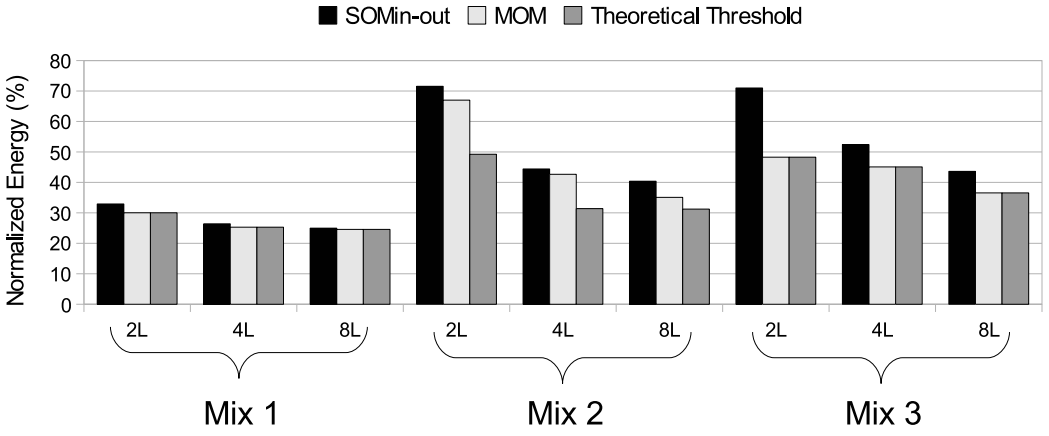
Finally, it can be noticed that the system behaves in a similar way regardless of the number of cores, that is, the benefits of migration that are observed in systems with two or three cores are also similar in a system with four cores, as shown in Figure 5.1(c). This fact makes the proposal a good candidate for commercial systems attending to the current industry trend of increasing the number of cores.

## 5.2 Comparing MOM versus SOM Variants

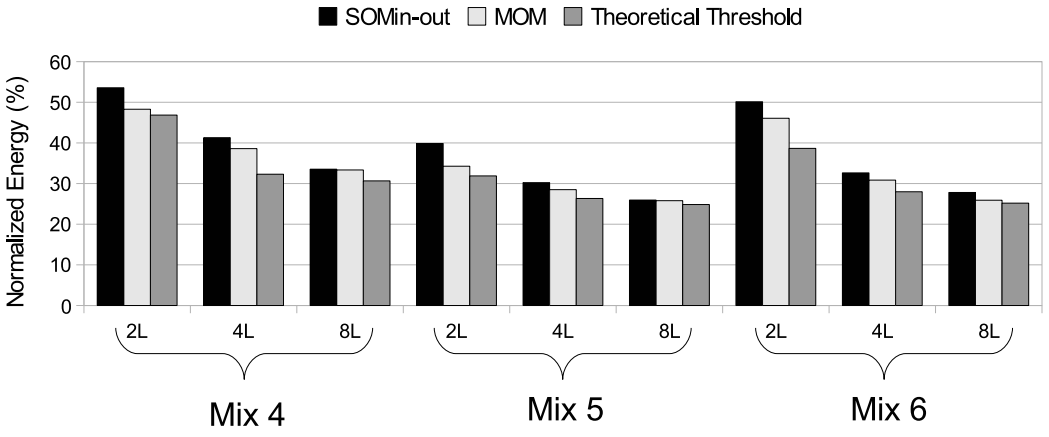
This section analyzes the energy improvements of the proposed MOM algorithm over the variants of the SOM algorithm. For comparison purposes the best SOM variant on average ($SOM_{in-out}$), and a theoretical threshold have been also included in the plots. This theoretical threshold is a value that represents 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 5.4 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 processor cores to work at a similar frequency for longer than any SOM variant. This can be also observed in the example of Figure 5.3. 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). 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.

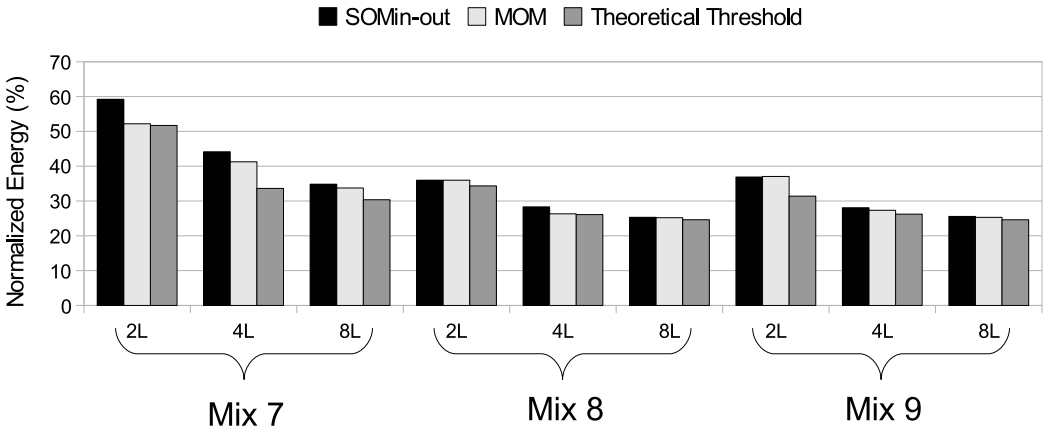
Moreover, in some mixes (e.g., mix 3) MOM results are very close to the energy savings of the theoretical threshold. However, if the utilizations of the tasks in a given mix widely differ among them, and depending on how run-time conditions evolve, the results of any practical scheduler may be far from the theoretical threshold (e.g., mix 7 and 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 requiring a much higher utilization than




(a) 2 Cores




(b) 3 Cores




(c) 4 Cores

FIGURE 5.4: $SOM_{in-out}$ versus MOM for different DVFS levels and number of cores.

TABLE 5.3: Average and standard deviation of task utilization.

| 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 |
| Standar deviation | 13.86 | 8.12 | 5.29 | 3.39 | 2.76 | 4.44 | 12.07 | 3.64 | 10.46 |

the others. 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 increases (4 cores), a Single Option Migration 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 the cores).

# Chapter 6

# 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 work has analyzed the impact on energy consumption due to scheduling strategies in a multicore embedded system implementing DVFS.

Two power-aware schedulers 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 have been 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 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 increases, the differences among energy benefits are reduced. Results show that task migration allows the proposed schedulers to achieve important energy benefits over the WF. These benefits are, on average, by 17% and 24% over the WF, for the SOM and MOM, respectively. Moreover, in some cases MOM benefits are up to 40%.

This work has 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.

## 6.1 Current and Future Work

As for future work we plan to extend this research, whose main topic is reducing power consumption in multicore embedded real-time systems. Besides task migration and load balancing we plan to focus on reducing energy consumption at the memory controller. More precisely we plan to devise different scheduling policies based on real-time priorities, or taking advantage of structures like the row buffer.

Moreover, since sometimes the nature of the workload prevents the partitioner from achieving a good balancing, having several frequency/voltage domains (each one for a subset of cores) can also help to enhance energy savings.

## 6.2 Publications

The following papers related with this work were submitted and accepted for publication in different international conferences and 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. Vol. 54. Num. 8. Pages 1282-1294. ISSN: 0010-4620. Oxford University Press. August 2011. (JCR $1^{st}$ Tertile)
- 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. ISSN: 1532-0626. Wiley-Blackwell. To Be Published in 2012. (JCR $2^{nd}$ Tertile)

- J. L. March, J. Sahuquillo, S. Petit, H. Hassan and J. Duato. "A Dynamic Power-Aware Partitioner with Task Migration for Multicore Embedded Systems". Proceedings of the $17^{th}$ International European Conference on Parallel and Distributed Computing. Pages 218-229. ISBN: 978-3-642-23399-9. Springer-Verlag. Bordeaux, France. 29 August - 2 September 2011. (CORE A)

- 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". Proceedings of the $10^{th}$ International Conference on Algorithms and Architectures for Parallel Processing. 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. 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. Fiuggi, Italy. To Be Published in 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. Elche, Spain. To be Published in 2012.

# References


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

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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*, 54(8):1282–1294, 2011.

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

[10] *Intel Atom Processor Microarchitecture*. INTEL Corp., Santa Clara, CA, USA, 2012. [Online]. Available: www.intel.com.

[11] *Marvell ARMADA$^{TM}$ 628*. Marvell Semiconductor, Inc., Santa Clara, CA, USA, 2012. [Online]. Available: http://www.marvell.com/company/press_kit/assets/Marvell_ARMADA_628_Release_FINAL3.pdf.

[12] 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.

[13] 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.

[14] 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.

[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] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] G. Buttazzo, E. Bini, and Yifan Wu. Partitioning Real-Time Applications Over Multicore Reservations. *IEEE Transactions on Industrial Informatics*, 7(2):302–315, 2011.

[24] *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.

[25] 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.

[26] 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.

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