March Cabrelles, JL.; Sahuquillo Borrás, J.; Petit Martí, SV.; Hassan Mohamed, H.; Duato Marín, JF. (2011). A dynamic power-aware partitioner with task migration for multicore embedded systems. En Euro-Par 2011 Parallel Processing. Springer Verlag (Germany). 2011(6852):218-229. doi:10.1007/978-3-642-23400-2_21.

# A Dynamic Power-Aware Partitioner with Task Migration for Multicore Embedded Systems

José Luis March, Julio Sahuquillo, Salvador Petit, Houcine Hassan, and José Duato
jomarcab@gap.upv.es, {jsahuqui,spetit,husein,jduato}@disca.upv.es

Department of Computer Engineering (DISCA)
Universitat Politècnica de València, Spain

**Abstract.** Nowadays, a key design issue in embedded systems is how to reduce the power consumption, since batteries have a limited energy budget. For this purpose, several techniques such as Dynamic Voltage Scaling (DVS) or task migration can be used. DVS allows reducing power by selecting the optimal voltage supply, while task migration achieves this effect by balancing the workload among cores.
This paper first analyzes the impact on energy due to task migration in multicore embedded systems with DVS capability and using the well-known Worst Fit (WF) partitioning heuristic. To reduce overhead, migrations are only performed at the time that a task arrives to and/or leaves the system and, in such a case, only one migration is allowed.
The huge potential on energy saving due to task migration, leads us to propose a new dynamic partitioner, namely DP, that migrates tasks in a more efficient way than typical partitioners. Unlike WF, the proposed algorithm examines which is the optimal target core before allowing a migration. Experimental results show that DP can improve energy consumption in a factor up to 2.74 over the typical WF algorithm.

## 1 Introduction

Embedded systems is an important segment of the microprocessor market since they are becoming ubiquitous in our life. Systems like PDAs, smart phones, or automotive, provide an increasing number of functionalities such as voice communication, navigation, or gaming, so that computational power is becoming more important every day. However, increasing computational power impacts on battery lifetime, so how to improve power management is a major design concern.

To deal with both computational and power management requirements, many systems use multicore processors. These processors allow a better power management than complex monolithic processors for the same level of performance. Moreover, many manufacturers (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's Montecito [14] and IBM Power 5 [10]. Also, leading manufacturers of the embedded sector, like ARM, plan to include multithreading technology in next-generation processors [16].

A power management technique that is being implemented in most current microprocessors is Dynamic Voltage Scaling (DVS) [9]. This technique allows the system to

improve its energy consumption by reducing the frequency when the processor has a low level of activity (e.g., a mobile phone that is not actively used). In a multicore system, the DVS regulator can be shared among several cores, also referred to as *global*, or private to each 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, enables more energy savings since each core frequency can be properly tuned to its applications requirements but it is more expensive [15].

Energy consumption in systems with a global DVS regulator can be further improved by properly balancing the workload [7,13]. To this end, a partitioner module is in charge of distributing tasks according to a given algorithm (e.g., Worst Fit [1] or First Fit) that selects the target core to run the task. Unfortunately, the nature of some workload mixes prevents the partitioner from achieving a good balancing. To deal with this drawback some systems allow tasks to migrate (move their execution) from one core to another, which results in energy saving improvements.

This work presents a dynamic power-aware partitioner, namely DP, for a multicore multithreaded system that dynamically (at run-time) assigns tasks to cores and allows task migration to improve energy consumption. Our focus is on tasks presenting real-time constraints, that is, tasks must end their execution before a given deadline or run during several periods before leaving the system. The proposed partitioner readjusts possible dynamic imbalances (due to new arrivals or exits of tasks) by reallocating tasks among cores. In this way, the workload can be more fairly balanced, so system frequency -in many cases- can be reduced, thus enabling further energy consumption improvements. In addition, the number of migrations has been limited in order to reduce overhead.

Finally, as the aim of migration is to reduce imbalance, it makes sense to analyze the benefits of applying migration when the workload changes. Three cases have been analyzed: when a task arrives to the system, when a task leaves the system, and both cases together. Experimental results show that enabling migration *only on arrival* in the classical WF algorithm allows achieving energy improvements in a factor up to 2.18 with respect to the case where no migration is allowed, while in the proposed DP algorithm these improvements can be up to 2.74.

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

## 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*, that uses a private task queue for each processor. The former allows task migrations 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 theory algorithms such as EDF (Earliest Dead-

line First) or RMS (Rate Monotonic Scheduling). An example of global scheduling for sporadic tasks can be found in [11].

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 [19,2,1]. For instance, Wei et al. [19] reduce energy consumption by exploiting parallelism of multimedia tasks on a multicore platform combining DVS with switching-off cores. Aydin et al. [2] present a new algorithm that reserves a subset of processors for the execution of tasks with utilization not exceeding a threshold. Unlike our work, none of these techniques use task migration among cores.

Some proposals have been dealing with task migration. Brandenburg et al. [4] evaluate some scheduling algorithms (both global and partitioned) in terms of scalability, although no power consumption were investigated. In [21] Zheng divides tasks into fixed and migration tasks, allocating each of the latter to two cores, so they can migrate from one to another. Unlike our work, in this paper there is no consideration about dynamic workload changes (tasks arriving to and leaving the system), instead, all tasks are assumed to arrive at the same instant, so migrations can be scheduled off-line. Seo et al. [15] present a dynamic repartitioning algorithm with migrations to balance the workload and reduce consumption. In [5] Brião et al. analyze how soft tasks migration affects NoC-based MPSoCs in terms of deadline misses and energy consumption. These two latter works focus on non-threaded architectures.

Regarding the scheduler, in [8] El-Haj-Mahmoud et al. virtualize a simultaneous multithreaded (SMT) processor into multiple single-threaded superscalar processors with the aim of combining high performance with real-time formalism. In order to improve real-time tasks predictability, Cazorla et al. [6] devise an interaction technique between the Operating System (OP) and an SMT processor. Notice that these works do not tackle energy consumption.

## 3 System Model

Figure 1 shows a block diagram of the modeled system. When a task reaches the system, a partitioner module allocates it into a task queue associated to a core, which contains the tasks that are ready for execution in that core. These task queues are components of the power-aware scheduler that communicates with a DVS regulator, in charge of adjusting the working frequency of the cores in order to satisfy the workload requirements. To focus our research, experiments considered a two-core processor implementing three hardware threads each.

Processor cores implement the coarse-grain multithreading 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 the other threads access memory, 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 Task Real-time 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 reaching its deadline (hard real-time). The end of the period and the deadline of a task are considered 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 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 to model real systems mode changes.

Besides its period and deadline, a task is also characterized by its Worst Case Execution Time (WCET). This parameter is used to obtain the task utilization: $U = \frac{WCET}{Period}$. Different partitioning algorithms may use this value in the process of allocating incoming tasks to a core, guaranteeing schedulability.

### 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 [3], which priorizes the tasks with the closest deadlines. Thus, the three tasks with the closest deadlines will be always mapped into the three hardware threads implemented in each core.

The scheduler is also in charge of calculating the target speed of each core according to the tasks's requirements. In this sense, in order to minimize power consumption, each core will choose the minimum frequency that fulfills the temporal contraints of its task set. This information is sent to the DVS regulator that selects the maximum frequency/voltage level among the requested by the cores.

The target frequency is recalculated to check if it has to be updated, but only when the workload changes, that is, when a task arrives to or/and leaves the system. In the

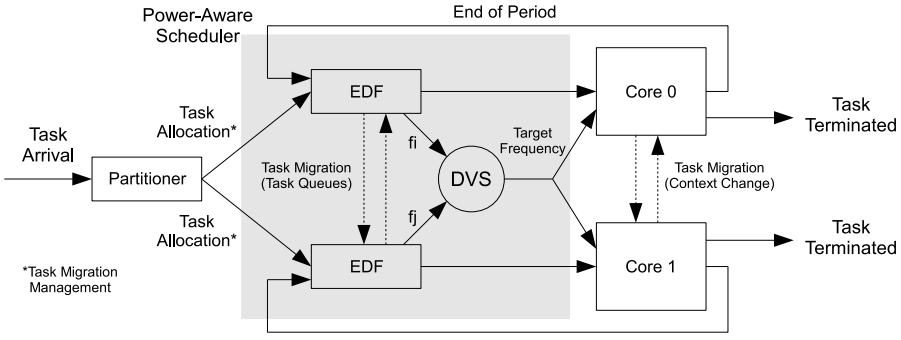



**Fig. 1.** Modeled system.

**Table 1.** Energy (E) used per frequency (F).

| F[MHz] | 500 | 400 | 300 | 200 | 100 |
|---|---|---|---|---|---|
| E[pJ/cycle] | 450 | 349.2 | 261.5 | 186.3 | 123.8 |

former case, a higher speed can be required because the workload increases. In the latter case, it could happen that a lower frequency could satisfy the deadline requirements of the remaining tasks.

Different speed values are considered for the power-aware scheduler, based on the frequency levels of a Pentium M [18] that are shown in Table 1. The 5L configuration allows the system to work at any of these five levels, whereas the 3L mode permits running tasks at the highest, the lowest and the intermediate (300 Mhz) frequency. Futhermore, the overhead of changing the frequency/voltage level has been modeled according to the voltage transition rate in the Pentium M processor, that is approximately $1mv/1\mu s$ [20].

## 4 Partitioning Heuristics with Task Migration

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 one of the best choices in order to balance the workload, thus improving energy savings [1]. WF balances workload by assigning the incoming task to the least loaded core. If more than one task arrives to the system at the same time, it arranges the incoming tasks by decreasing utilization order and assigns them to the cores beginning with the task with highest utilization. This algorithm was initially used in partitioned scheduling, thus, it does not support task migration among cores by design. Therefore, once WF has assigned 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).

### 4.1 Extending Worst Fit to Support Task Migration

Figure 2 shows an example of how task migration could improve workload balancing. 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 25% (i.e., its WCET occupies a quarter of its period), while the utilization of task 1 is around 33%. At point t2, task 2, whose utilization is around 66%, arrives to the system, and the WF algorithm assign it to core 0 (since it is the least loaded core). Consequently, the system would exhibit a high workload imbalance since the global utilization of core 0 and core 1 would be 91% and 33%, respectively. To solve this imbalance, task 0 can be migrated to core 1, so providing a better balance (66% in core 0 versus 58% in core 1).

The system can become unbalanced when the workload changes, that is, when a task arrives to or leaves the system. Thus, migration policies should apply in these points in order to be effective. This leads to three variants of the WF policy: $WF_{in}$, $WF_{out}$, and $WF_{in-out}$. $WF_{in}$ allows migration only when a new task arrives to the system, $WF_{out}$

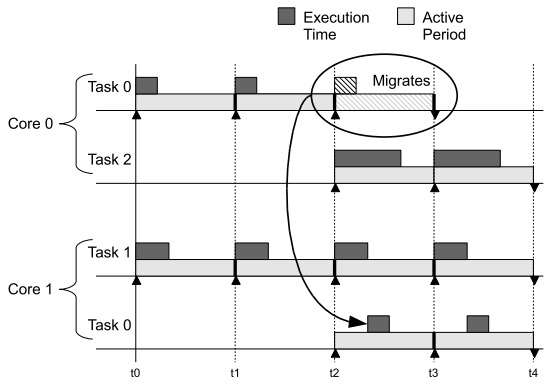



**Fig. 2.** Task periods and migrations.

when a task leaves the system, and $WF_{in-out}$ allows migration in both previous cases. To avoid performing too much migrations, which could lead to excessive overhead, we limit the number of migrations performed when a task arrives to or leaves the system to only one.

Figure 3 shows the Migration Attempt (MA) algorithm. This routine 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 (since there are two cores) to obtain a theoretical utilization value that represents the amount of work that should migrate to achieve a perfect balancing. Then, it searches the task in the most loaded core whose utilization is the closest to this one. Notice that it could happen that by migrating that task the workload balancing would not improve (e.g., consider a situation where only

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**

**Fig. 3.** Migration Attempt algorithm.

one task is assigned to the most loaded core). Therefore, the algorithm performs the migration only if it improves the workload balancing.

### 4.2 Dynamic Partitioner

This subsection presents the proposed Dynamic Partitioner (DP). As done by the WF algorithm, DP also arranges the tasks arriving to the system by decreasing utilization order. However, before assigning any incoming task to a given core, DP checks how the workload balancing would become if the incoming task were assigned to the first core. Then, it also calculates the effect of performing a migration attempt (as shown in Figure 3). These testings are performed for each core in the system. Finally, the core assignment that provides the best overall balance is applied. Two versions of DP are considered: $DP_{in}$ and $DP_{in-out}$. $DP_{in}$ refers to the described DP algorithm, where a migration can be performed only when a task arrives to the system, while $DP_{in-out}$ also performs a migration attempt when a task leaves the system.

Figure 4 depicts an example where the $DP_{in}$ heuristic improves the behavior of $WF_{in}$. The latter 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 balancing. Thus, the final imbalance becomes 20% (i.e., $90\% - 70\%$). In contrast, when $DP_{in}$ is applied, it also checks the result of allocating the new task to core 1 ($DP_{in}$ B arrow) and then considering one migration. In this case, the migration enables a better balance since both cores remain equally loaded with 80% of utilization, which will be the distribution selected by $DP_{in}$.

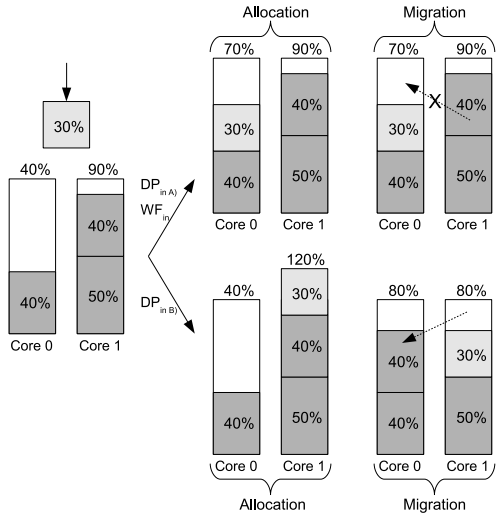



**Fig. 4.** $WF_{in}$ vs $DP_{in}$.

To sum up, the main difference between $WF_{in}$ and $DP_{in}$ is that the former selects only one core and performs a migration attempt, whereas the proposed heuristic checks different cores, and choses the best option in terms of workload balance.

## 5 Experimental Results

Experimental evaluation has been conducted by extending the Multi2Sim simulation framework [17], to model the system described in Section 3. As stated before, experiments considered a two-core processor implementing three hardware threads each. Internal core features have been modeled like an ARM11 MPCore based processor, but modified to work as a coarse-grain multithreaded processor with in-order execution, two-instruction issue width, and a 100-cycle memory latency.

Table 2 shows the benchmarks from the WCET analysis project [12] that were used to prepare real-time workload mixes. These mixes have been designed taking into account aspects such as task utilization, number of repetitions (task periodicity), and the sequence of active and inactive periods. 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 three and five voltage levels.

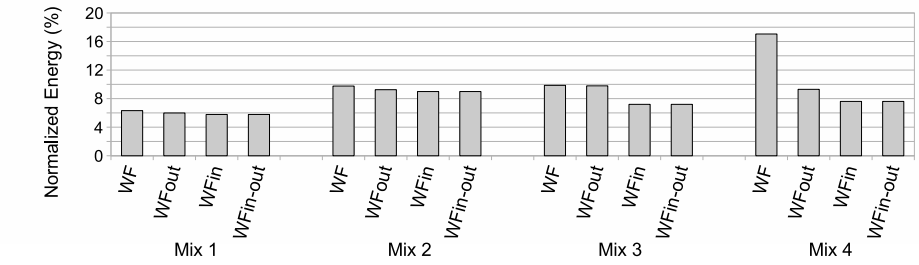
### 5.1 Impact of Applying Migrations at Different Points of Time

This section analyzes the best points of time to carry out migrations focusing on the standard WF algorithm (no migration is supported) and its variants supporting migration ($WF_{out}$, $WF_{in}$, $WF_{in-out}$). Figure 5 shows the relative energy consumption compared to the energy consumed by the system working always at the maximum speed for diverse benchmark mixes and DVS configurations.

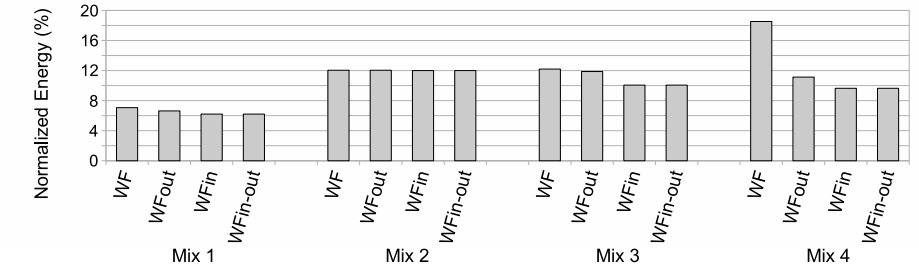
As observed, migration can provide huge energy savings with respect to no migration (WF) regardless when migration is applied. For instance, in the 5-level system with task migration mixes 3 and 4 improve their energy consumption in a factor up to 1.33

**Table 2.** Benchmark description.

| Name | Function Description | Name | Function Description |
|---|---|---|---|
| adpcm | Adaptive pulse code modulation algorithm | insertsort | Insertion sort on a reversed array of size 10 |
| bs | Binary search for a 15-element array | janne_complex | Nested loop program |
| bsort100 | Bubblesort program | jfdctint | Discrete-cosine transformation |
| cnt | Counts non-negative numbers in a matrix | lcdnum | Read ten values, output half to LCD |
| compress | Data compression program | lms | LMS adaptive signal enhancement |
| cover | Program for testing many paths | ludcmp | LU decomposition algorithm |
| crc | Cyclic redundancy check on 40-byte data | matmult | Matrix multiplication of two 20x20 matrices |
| duff | Copy 43-byte array | minver | Inversion of floating point matrix |
| edn | FIR filter calculations | ns | Search in a multi-dimensional array |
| expint | Series expansion for integral function | nsichneu | Simulate an extended Petri Net |
| fac | Factorial of a number | qsort-exam | Non-recursive version of quick sort algorithm |
| fdct | Fast Discrete Cosine Transform | qurt | Root computation of quadratic equations |
| fft1 | 1024-point Fast Fourier Transform | select | Nth largest number in a floating point array |
| fibcall | Simple iterative Fibonacci calculation | sqrt | Square root function |
| fir | Finite impulse response filter | statemate | Automatically generated code |




(a) 5L




(b) 3L

**Fig. 5.** Worst Fit variants comparison for different DVS levels.

and 2.18, respectively, when compared with their execution in the same system without migrations. This trend is also followed, although to a lesser extent, in the 3-level system.

Comparing the three WF versions with task migration, it can be observed that if migration can apply only each time a new task arrives instead of when a task terminates, then much higher energy savings can be achieved. The main reason is that the inter-arrival time standard deviation is higher than that of the inter-leaving time, since several tasks reach the system at the same time. Inter-arrival standard deviation values of the mixes are 3.87, 24.48, 43.98, and 14.65 Mcycles for mix 1, mix 2, mix 3, and mix 4, respectively. On the other hand, the inter-leaving time is, on average, 3.65, 22.50, 36.40, and 12.32 Mcycles. Finally, $WF_{in-out}$ offers scarce benefits over $WF_{in}$ since it only adds a low number of extra migrations.

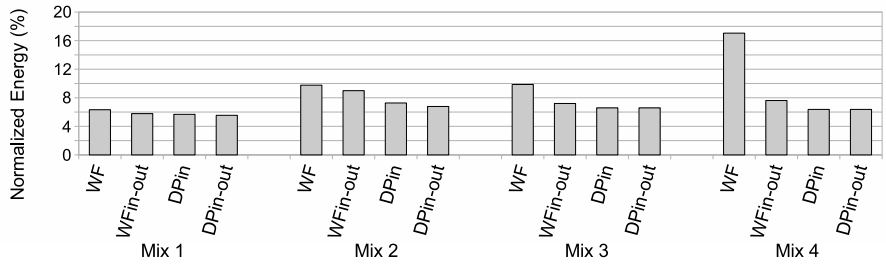
Notice that if the system implements more DVS frequency levels (5 levels in the figure), then more energy savings can be obtained since the system can select a frequency closer to the optimal estimated by the scheduler. However, despite this fact, an interesting observation is that energy benefits due to migration in the 3-level system can reach or even surpass the benefits of having the 5-level system without migrations. For example, the energy consumption of $WF_{out}$ for mix 4 in the 3-level system is around 11% of the consumption of the baseline, whereas the same value of WF in the 5-level system is 17%.

### 5.2 Comparing DP versus WF variants

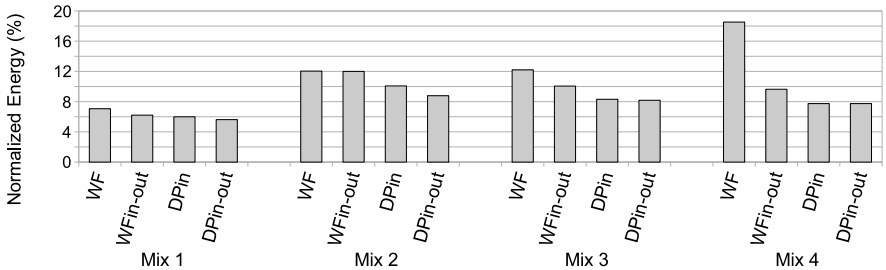
This section analyzes the energy improvements of two variants of the proposed DP algorithm ($DP_{in}$ and $DP_{in-out}$) over the WF algorithm. For comparison purposes the best variant of the WF ($WF_{in-out}$) with migration has been also included in the plots. Figure 6 shows the results.

Results show that, regardless the mix and system-level, both variants of DP always consume less power than $WF_{in-out}$. $DP_{in-out}$ achieves, for mixes 3 and 4, energy improvements over WF in a factor up to 2.74 and 1.56, respectively. Moreover, for mix 2, where $WF_{in-out}$ is only able to find scarce benefits over WF, the proposed DP improves the energy consumption of WF around 1.51.

For a better understanding of the algorithms behavior, we define the *migration rate* metric as the number of migrations performed by the algorithm divided by the number of times that the migration algorithm is executed. For instance, regarding the *in* variant of the WF and DP algorithms, the migration rates of $WF_{in}$ are 64%, 62%, 54%, 45% for mix 1, mix 2, mix 3, and mix 4, respectively; while for $DP_{in}$ the corresponding values are 64%, 76%, 68%, and 73%. This means that the proposal performs migrations in some cases where the WF is not able to find any candidate to migrate at all.




(a) 5L




(b) 3L

**Fig. 6.** WF versus DP for different DVS levels.

## 6 Conclusions

Workload balancing has been already proved to be an efficient power technique in multicore systems. Unfortunately, unexpected workload imbalances can rise at run-time provided that the workload is dynamically changing since new tasks arrive to or leave the system. To palliate this situation this paper has analyzed the impact on energy consumption of task migration combined with workload balancing.

To prevent excessive overhead, task migration has been strategically applied at three different execution times where the workload changes (at task arrival, at task termination, and in both cases). Results with respect to the WF algorithm showed that applying migration at arrival time can save results in a factor up to around 2.18. This results can be slightly improved if migration is also applied when tasks terminate.

Due to the potential of migration, this paper has proposed the DP algorithm, which achieves much better energy improvements than classical partitioning algorithms like WF. The proposal improves energy consumption in a factor of 1.51 in some workloads where WF with migrations provides scarce benefits, and energy can be improved in a factor up to 2.74 in the analyzed workloads.

Experimental results also showed that migration can provide energy consumption improvements with respect to a more complex system with a higher number of frequency/voltage levels. A final remark is that achieving a better workload balancing by allowing task migrations not only results in energy savings, but also allows a wider set of tasks to be scheduled.

## Acknowledgments

This work was supported by Spanish CICYT under Grant TIN2009-14475-C04-01, and by Consolider-Ingenio under Grant CSD2006-00046.

## References

1. AlEnawy, T.A., Aydin, H.: Energy-Aware Task Allocation for Rate Monotonic Scheduling. In: Proceedings of the 11th Real Time on Embedded Technology and Applications Symposium. pp. 213–223. IEEE Computer Society, San Francisco, CA, USA (7-10 March 2005)
2. Aydin, H., Yang, Q.: 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. p. 113. IEEE Computer Society, Nice, France (22-26 April 2003)
3. Baker, T.: An Analysis of EDF schedulability on a multiprocessor. IEEE Transactions on Parallel and Distributed Systems 16(8), 760–768 (2005)
4. Brandenburg, B.B., Calandrino, J.M., Anderson, J.H.: On the Scalability of Real-Time Scheduling Algorithms on Multicore Platforms: A Case Study. In: Proceedings of the 29th Real-Time Systems Symposium. pp. 157–169. IEEE Computer Society, Barcelona, Spain (30 November - 3 December 2008)
5. Brião, E., Barcelos, D., Wronski, F., Wagner, F.R.: Impact of Task Migration in NoC-based MPSoCs for Soft Real-time Applications. In: Proceedings of the International Conference on VLSI. pp. 296–299. IEEE Computer Society, Atlanta, GA, USA (15-17 October 2007)

6. Cazorla, F., Knijnenburg, P., Sakellariou, R., Fernández, E., Ramirez, A., Valero, M.: Predictable Performance in SMT Processors: Synergy between the OS and SMTs. IEEE Transactions on Computers 55(7), 785–799 (2006)
7. Donald, J., Martonosi, M.: Techniques for Multicore Thermal Management: Classification and New Exploration. In: Proceedings of the 33rd Annual International Symposium on Computer Architecture. pp. 78–88. IEEE Computer Society, Boston, MA, USA (17-21 June 2006)
8. El-Haj-Mahmoud, A., A.AL-Zawawi, Anantaraman, A., Rotenberg, E.: 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. pp. 213–224. ACM Press, San Francisco, CA, USA (24-27 September 2005)
9. Hung, C., Chen, J., Kuo, T.: 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. pp. 303–312. IEEE Computer Society, Rio de Janeiro, Brazil (5-8 December 2006)
10. Kalla, R., Sinharoy, B., Tendler, J.: IBM Power5 Chip: A Dual-Core Multithreaded Processor. IEEE Micro 24(2), 40–47 (2004)
11. Kato, S., Yamasaki, N.: Global EDF-based Scheduling with Efficient Priority Promotion. In: Proceedings of the 14th International Conference on Embedded and Real-Time Computing Systems and Applications. pp. 197–206. IEEE Computer Society, Kaohisung, Taiwan (25-27 August 2008)
12. Malardalen Real-Time Research Center, Vasteras, Sweden: WCET Analysis Project. WCET Benchmark Programs (2006), [Online]. Available: http://www.mrtc.mdh.se/projects/wcet/
13. March, J., Sahuquillo, J., Hassan, H., Petit, S., Duato, J.: A New Energy-Aware Dynamic Task Set Partitioning Algorithm for Soft and Hard Embedded Real-Time Systems. To be published on The Computer Journal (2011)
14. McNairy, C., Bhatia, R.: Montecito: A Dual-Core, Dual-Thread Itanium Processor. IEEE Micro 25(2), 10–20 (2005)
15. Seo, E., Jeong, J., Park, S., Lee, J.: Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors. IEEE Transactions on Parallel and Distributed Systems 19(11), 1540–1552 (2008)
16. Shah, A.: Arm plans to add multithreading to chip design. ITworld (2010), [Online]. Available: http://www.itworld.com/hardware/122383/arm-plans-add-multithreading-chip-design
17. Ubal, R., Sahuquillo, J., Petit, S., López, P.: Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. In: Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing. pp. 62–68. IEEE Computer Society, Gramado, RS, Brazil (24-27 October 2007)
18. Watanabe, R., Kondo, M., Imai, M., Nakamura, H., Nanya, T.: Task Scheduling under Performance Constraints for Reducing the Energy Consumption of the GALS Multi-Processor SoC. In: Proceedings of the Design Automation and Test in Europe. pp. 797–802. ACM, Nice, France (16-20 April 2007)
19. Wei, Y., Yang, C., Kuo, T., Hung, S.: Energy-Efficient Real-Time Scheduling of Multimedia Tasks on Multi-Core Processors. In: Proceedings of the 25th Symposium on Applied Computing. pp. 258–262. ACM, Sierre, Switzerland (22-26 March 2010)
20. Wu, Q., Martonosi, M., Clark, D.W., Reddi, V.J., Connors, D., Wu, Y., Lee, J., Brooks, D.: A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In: Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 271–282. IEEE Computer Society, Barcelona, Spain (12-16 November 2005)
21. Zheng, L.: 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. pp. 3055–3058. IEEE Computer Society, Shanghai, China (21-25 September 2007)