On the Energy-Aware Partitioning of Real-Time Tasks on Homogeneous Multi-Processor Systems

Mario Bambagini[1], Juri Lelli[1], Giorgio Buttazzo[1] and Giuseppe Lipari[1,2]

[1]Scuola Superiore Sant'Anna, Pisa, Italy
[2]Ecole Normale Superieure de Cachan, France

***Abstract*—In high-performance computing systems, efficient energy management is a key feature for keeping energy bills low and avoiding thermal dissipation problems, as well as for controlling the application performance. This paper considers the problem of partitioning and scheduling a set of real-time tasks on a realistic hardware platform consisting of a number of homogeneous processors. Several well-known heuristics are compared to identify the approach that better reduces the overall energy consumption of the entire system. Despite the actual state of art, the approach which minimizes the number of active cores is the most energy efficient.**

I. Introduction

Energy saving has become an important issue in modern computing systems, spreading from embedded systems to high performance servers. Concerning the first scenario, in most cases, devices are battery operated, meaning that a lower energy consumption can lead to a longer lifetime or higher performance. Conversely, high-performance servers do not have explicit constraints on the supplied power, but an effective energy-aware policy may lead to cheaper energy bills and less strict thermal dissipation problems. This paper deals with energy saving in high-performance servers.

In CMOS technology, which is leading today's hardware circuits, the power consumption of a gate can be expressed as a function of the supply voltage V and the clock frequency f through the following Equation [1]:

$$P_{gate} = \alpha C_L V^2 f + \alpha V I_{short} + V I_{leak} \tag{1}$$

where C_L is the total capacitance driven by the gate, α is the gate activity factor (i.e., the probability of gate switching), I_{short} is the current flowing between the supply voltage and ground during gates switching, and I_{leak} is the leakage current. In particular, the three terms describe the dynamic, the short circuit, and the static power component dissipated in a gate, respectively.

Dynamic Voltage and Frequency Scaling (DVFS) techniques consist of reducing the voltage V and the frequency f in Equation (1), in order to slow down the processor and reduce the dynamic power. As a consequence, task execution times become longer. In this approach, the maximum usable frequency also depends on the voltage V according to Equation (2), where V_T is the *Threshold Voltage*, that is, the minimum voltage between gain and source able to create a channel from drain to source in a MOSFET transistor.

$$circuit\ delay = \frac{V}{(V - V_T)^2}. \tag{2}$$

Given such a limitation, not all the pairs (V,f) are usable.

Another well-known approach is the Dynamic Power Management (DPM), which consists in executing the workload at the maximum performance level and switching the processor off during idle intervals.

In actual operating systems, power management is implemented through the *Advanced Configuration and Power Interface* (ACPI) [2], whose specification defines a standard for devices configuration and monitoring. In particular, ACPI offers the operating system an easy and flexible interface to discover and configure the compliant devices. For instance, unused devices, including the entire system, can be switched to a low-power state. Concerning the processors, such a standard scales the CPU speed down according the system utilization, while using the idle state when there is no pending workload.

Since response time is an important parameter to minimize in data servers, real-time analysis can be exploited to formally guarantee whether the workload can or cannot be executed on the actual hardware without exceeding deadlines, preventing a quality of service degradation or unrecoverable faults. An additional benefit of using real-time scheduling techniques is the possibility of allocating a predefined fraction of CPU time to each task, so achieving a temporal isolation among them.

On multiprocessor systems, scheduling techniques can be roughly divided into three macro groups: partitioned, global, and hybrid. Partitioned techniques statically assigns each task to a specific CPU forbidding a task to migrate onto another processor even thought it is idle. This method allows designers to easily check the system feasibility but, in many cases, it leads to a waste of computational resources. Global approaches improve the system utilization by allowing task migration at any time in any processor, but are more difficult to analyze and may introduce significant run-time overhead. Hybrid scheduling approaches try to combine the two previous techniques to reduce their drawbacks and exploit their advantages.

Paper contribution. This paper considers the problem of partitioning and scheduling a set of real-time tasks on a realistic hardware platform consisting of a number of homogeneous

This work has been supported by the 7th Framework Programme JUNIPER (FP7-ICT-2011.4.4) project, founded by the European Community under grant agreement n. 318763.

processors. Several well-known heuristics are compared to identify the approach that better reduces the overall energy consumption of the entire system (including CPU, memory, I/O peripherals and other devices). Although many algorithms have been published on this topic, they are characterized by a significant complexity, which makes them not suitable for a real usage in high-performance systems. Conversely, most heuristic algorithms require a polynomial complexity in the worst case, resulting in very competitive performance even for reconfigurable systems, where the overhead due to reconfiguration has to be as small as possible.

A real platform (Dell PowerEdge R815 [3]), designed for high-performance computation, is taken into account in this work. An additional contribution of this paper with respect to the current state of art is that we consider the power consumption of the entire system, not only the dissipation due to the processor. This is particularly relevant, as considering only a single component may give misleading results that are not valid in a general case.

In summary, this paper aims at comparing the actual state of art from a pragmatical point of view, by considering a real platform rather than proposing new algorithms.

Paper organization. The rest of the paper is organized as follows: Section II introduces the system model taken into account in the analysis; Section III explains how the heuristics work; Section IV compares them on the system under analysis; Section V concludes the paper stating some final remarks and introducing a few open problems for a future work.

A. *Related work*

Energy saving issues in real-time embedded systems have been studied for decades, since Yao et al. [4] proposed an optimal off-line algorithm for single core systems.

Aydin and Yang [5] compared the behavior of four well-known heuristics (First-Fit, Next-Fit, Best-Fit, Worst-Fit) for homogeneous multicore systems and periodic independent tasks. Their work stated that Worst-Fit Decreasing (WFD), which aims at balancing the workload among the cores, is the most effective for reducing the energy consumption while considering cubic power functions.

Yang et al. [6] proposed an algorithm which partitions a set of frame-based tasks (with same period and deadline) using the Worst-First strategy and then scales speed in particular instant according to the task features. Although the algorithm is characterized by a good approximation factor with respect to the optimal scheduling, the authors made several non-realistic assumptions, such as continuous and infinitive frequency range ($s \in [0, \infty]$) and negligible consumption in idle state.

Kandhalu et al. [7] considered the issue of partitioning a set of periodic real-time tasks on multicore systems characterized from a single voltage island (all the processors share the same voltage and frequency). First of all, they proved the approximation upper bound for the classical Worst-First heuristic and then, they provided their own algorithm which overcomes several limitation of the state of art.

Huang et al. [8] faced the problem of partitioning a workload with precedence constraints on a multiprocessor system while considering communication costs between processors. Their solution relied on complete search and meta-heuristics.

Pagani and Chen [9] carried out an analysis that, independently from the task partitioning algorithm, found the ratio of the Single Frequency Approximation (SFA) scheme on multicore voltage islands with respect to the optimal solution. More precisely, SFA sets the frequency of the voltage island equal to the maximum utilization among the cores.

Petrucci et al. [10] considered the problem of partitioning a set of independent tasks on a heterogeneous systems. More precisely, they introduced a periodic partitioning algorithm (implemented as ILP problem) which migrates tasks among cores according to their actual phase (interval in which the code is mostly either CPU or I/O intensive).

II. System Model

This section presents the power model of the considered platform and introduces the workload characteristics.

A. *Power model*

We consider a multi-processor platform composed of m homogeneous processors ($\phi_j,\ j = 1, \ldots, m$) whose frequencies can be set independently from each others. The frequency range consists of k discrete elements $\{f_1, f_2, \ldots, f_k\}$, ordered by ascending values. In the following, the normalized speed s, defined as $s = f/f_k$, is used as a more convenient parameter ($s = 1$ denotes the maximum speed).

Scaling speed introduces an overhead proportional to the absolute difference between the new and the previous one, as the Phase-Locked Loop (PLL) which generates the clock signal must be tuned.

As required by the ACPI standard, each processor provides several low-power states characterized by different power consumption and different time overhead for entering and leaving such states. More generally, the system provides the following operative states:

- S0: The system is fully operative (both processors and memory);
- S1: Although caches are flushed and code execution is suspended, processors and memory are active;
- S2: Processors are switched off and the dirty cache is flushed to the memory. Other devices may be kept on;
- S3: Similar to S2 but more devices are put in sleep;
- S4 (hibernation): Data in memory is copied in to the hard drive and all the system is powered down;
- S5: The system is completely off except for the logic that allows the system to switch on. Putting the system in to S0 requires a complete boot and no data is retained.

Each processor can be put independently in S1, S2 and S3.

Intuitively, the deeper the low-power state the longer the required time to move in/out such a state. When the system is almost completely off (S5), the sleeping and waking up time is due to the shutting down and booting of the operating system, respectively. Moreover, the hibernation state requires writing/reading a large amount of data to/from the hard drive to properly restore the main memory content. The overhead related to S4 is around several seconds while S5 requires many seconds. States S2 and S3 require an overhead in the order of several milliseconds, while recovering from the first state (S1) requires only to fill the cache (comparable to the preemption overhead) [11], [12].

For the sake of simplicity, periods and computation times are assumed to be much longer than state transition overheads, which can then be discarded from the analysis.

The ACPI module is in charge of putting the processor in a predefined low-power state (statically selected in the BIOS) whenever there is no process running. Typically, the most used state is S3, as it provides a good trade-off between power consumption and time overhead for almost all the applications.

The energy consumption of the entire system in the interval $[t_1, t_2]$ can be expressed by the following equation:

$$E(t_2, t_1) = E_{CPU}(t_2, t_1) + E_{NO\_CPU}(t_2, t_1). \qquad (3)$$

where E_{CPU} denotes the energy dissipated by the processors and E_{NO_CPU} the energy dissipated by the remaining components, including the main memory (DDR), disks, network interfaces and other peripherals whose behaviors can be considered not directly affected by the running frequency of the processors. Although running tasks impact on such devices, for the sake of simplicity, we assume a constant average device dissipation, that is:

$$E_{NO\_CPU}(t_2, t_1) = (t_2 - t_1) \cdot P_{NO\_CPU}.$$

where P_{NO_CPU} is the power consumed by the devices. Since each processor can have a different state/speed at any time, the processor energy dissipation is the integral of power consumption in the interval $[t_1, t_2]$:

$$E_{CPU}(t_2, t_1) = \int_{t_1}^{t_2} P_{CPU}(t)dt.$$

Basically, the processor power consumption at time t is a function of the actual state of each processor: the low-power state in use if it is asleep, or the running speed if it is active. In other words, such a function depends on m variables:

$$P_{CPU}(t) = f(\phi_1(t), \phi_2(t), ..., \phi_m(t)),$$

where $\phi_j(t)$ denotes the state of processor ϕ_j at time t.

To characterize the power model of the processors, two possible methods are possible: extending the well-known Martin's equation [13] to consider m variables (one for each CPU) or creating a table with the consumption for all the possible configurations. In this paper we consider a variant of the second option which, given the consumption obtained by using the same speed for each processor, extrapolates an upper bound for all the other configurations.

B. Task model

The workload Γ consists of n fully-preemptive periodic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by a worst-case execution time (WCET) $C_i(s)$, which is a function of the speed, a relative deadline D_i and a period T_i.

The WCET of τ_i is computed as $C_i(s) = C_i^{fix} + C_i^{var}/s$, where C_i^{fix} and C_i^{var} denote the components which does not scale and scale with speed, respectively. More precisely, C_i^{var} models the code which executes CPU-intensive computation, while C_i^{fix} represents the code interacting with devices whose frequencies do not depend on CPU frequency (e.g. I/O operations and spin locks). Moreover, the parameter γ ($1 \geq \gamma \geq 0$) is introduced to specify the overall fraction of computation time that does not scale with the speed. Such a value is computed as the average speed-independent fraction: $\gamma = \frac{1}{n} \sum_{\tau_i} C_i^{fix}/C_i(s_m)$.

All parameters are assumed to be in $\mathbb{N}^+$. Each task τ_i generates an infinite sequence of jobs ($\tau_{i,k}, \ k = 1, 2...$), with the first job arriving at time zero ($a_{i,0} = 0$) and subsequent arrivals separated by T_i units of time ($a_{i,k+1} = a_{i,k} + T_i$).

In addition, the utilization term $U_i(s) = C_i(s)/T_i$ determines the processor bandwidth that task τ_i requires at speed s. Assuming to schedule the periodic workload by the EDF policy [14], the tasks assigned to the processor ϕ_j do not miss any deadline if and only if

$$U_{\phi_j} = \sum_{\tau_i \in \Gamma | \tau_i \rightarrow \phi_j} U_i(s_i) \leq 1. \qquad (4)$$

The overall task set is feasible if Equation 4 is satisfied for each processor.

Finally, we define the hyperperiod H as the time interval after which the schedule repeats itself. In the case of periodic tasks, such analysis horizon is computed as the least common multiple of periods, $H = lcm(T_1, T_2, ..., T_n)$.

III. Heuristics

This paper compares two heuristics: Worst-Fit Decreasing (WFD) and Best-Fit Decreasing (BFD).

First of all, both heuristics sort the task set by descending utilization: $\bar{\Gamma} = \{\tau_i \in \Gamma | u_{i-1} \geq u_i \geq u_{i+1}\}$. Then, starting from the first element in $\bar{\Gamma}$, WFD assigns each task to the processor with the highest unused utilization, while BFD chooses the one whose spare utilization fits better.

Let us consider three processors and five tasks with the following utilization values: $u_1 = 0.6$, $u_2 = 0.5$, $u_3 = 0.3$, $u_4 = 0.3$ and $u_5 = 0.1$. The WFD heuristic would start assigning each of the first three tasks to one free processor, τ_1 to ϕ_1, τ_2 to ϕ_2, and τ_3 to ϕ_3. Then, the spare utilization on each processors become 0.4, 0.5 and 0.7, respectively. Next, the fourth task is assigned to ϕ_3 which has the highest spare capacity, reducing it down to 0.4. Finally, τ_5, characterized by the lowest utilization, is assigned to ϕ_2. The final partitioning is shown in Figure 1.

Conversely, the BFD heuristic assigns τ_1 to ϕ_1 and then tries to allocate τ_2 to ϕ_1, but the residual utilization of ϕ_1 is not enough to accommodate τ_2, which is then allocated to ϕ_2. Unlike the previous case, τ_3 is allocated on ϕ_1 and all the remaining two tasks are assigned to the second core ϕ_2. The BFD result is reported in Figure 2.

In conclusion, WFD led to a task partitioning that left 0.4, 0.3 and 0.4 as spare capacity on the three cores, while BFD utilized entirely the second core and partially the first (90%), leaving the third processor off.

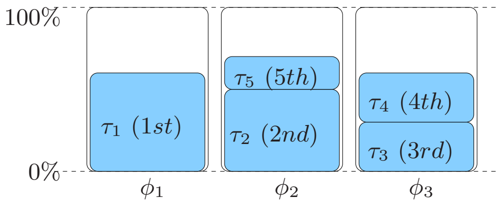

Fig. 1. Task partitioning obtained by the WFD heuristic.

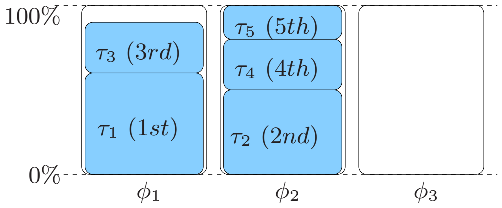

Fig. 2. Task partitioning obtained by the BFD heuristic.

As this simple example has shown, BFD aims at reducing the number of active cores, while WFD attempts to exploit all processors to reduce the overall working performance.

Once tasks have been partitioned, the remaining utilization on each core can be exploited to further reduce energy consumption [15]–[19].

In this paper, we consider two extreme approaches:

- DVFS: when the system starts, for each core, the slowest speed that guarantees the task set feasibility is set;
- DPM: the workload is executed at the maximum performance and then ACPI exploits low-power states when there are no pending tasks.

Since we deal with a discrete set of frequencies, if DVFS is not able to exploit the whole residual utilization, DPM is used in addition to take advantage of it.

IV. Experimental results

In this section, the power measurements related to the multicore platform is first reported to provide the consumption profile. Then, the two heuristics introduced in Section III are compared in terms of energy consumption.

A. Consumption profile

The considered platform, a Dell PowerEdge R815 rack server, is equipped with 48 homogeneous cores supporting the following frequency range $\{0.8, 1.0, 1.3, 1.5, 1.9\}$GHz which leads to the speed set: $\{0.42, 0.53, 0.68, 0.79, 1.0\}$. Each core can set its frequency independently of the others. Cores are divided in 8 clusters, each containing 6 cores and one eighth of the main memory (NUMA system). The platform runs the GNU/Linux kernel 3.10.0-rc3 [20].

The power measurements reported in the paper have been obtained by monitoring the absorbed power from the entire platform, including memory, I/O peripherals, and buses.

In the considered scenario (48 cores, 5 frequencies each and two low-power states), the configurations to be checked are 7^{48}. Since the number is extremely high, we measured the consumption for a set of key configurations and then the others are obtained by interpolation. More precisely, for each speed, we measured the consumption of setting all the active cores to the speed under analysis while varying the number of fully loaded cores (from 1 to 48). The workload consists of an endless execution of the EEMBC Coremark benchmark [21], which implements mathematical operations (CPU-intensive code) and keeps the processor always busy (no I/O phases). As a result, the execution time perfectly scales with the frequency. Concerning the unused processors, the two low-power states used by the ACPI driver have been considered. The first one is called IDLE and it is automatically inserted by the module when there are no pending tasks, while the OFF state is manually set by the user.

The first consideration related to the measurements reported in Figure 3 concerns the impact of the low-power state. More precisely, the OFF feature was supposed to guarantee the lowest consumption but it exploits S1 and introduces a significant overhead for updating the kernel data structures. On the other hand, the IDLE state exploits S3, guaranteeing lower consumption and shorter overhead.

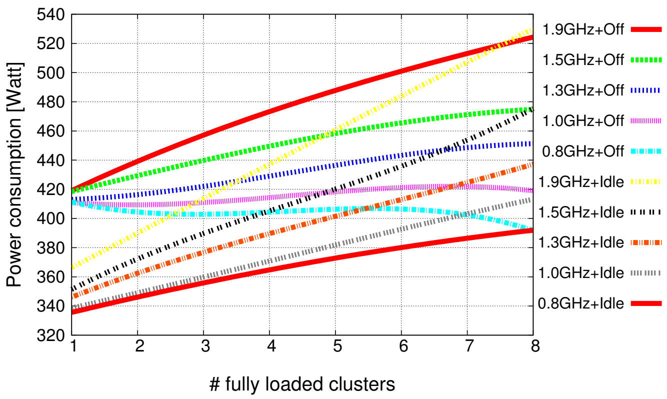

Fig. 3. Power consumption as a function of active clusters and low-power states.

Given such measurements, the configurations in which all the processors exploit the same frequency but only some of the cores in the cluster are active (e.g., 3 processors out of 6) are assumed to dissipate the interpolated value. We empirically tested such an assumption for several configuration and it turned out to be an acceptable approximation. When the processors exploit different frequencies, we consider the average consumption as the weighted average of the consumption assuming all the active processors running at the same frequency. Formally, it can be expressed as:

$$P_{CPU}(t) = \sum_{s_j} w_j \bar{f}(s_j),$$

where w_j is the actual number of CPU at speed s_j divided by the total number of active cores and $\bar{f}(s_j)$ represents the consumption obtained by setting all the active cores at speed s_j. For example, if there are two clusters at frequency $1.0GHz$ and one at $1.5GHz$, the consumption would be:

$$P_{CPU}(t) = \frac{12}{18}360 + \frac{6}{18}375 = 365W,$$

where 360 and 375 are the consumptions of having three clusters at 1.0 and 1.5 GHz, respectively.

It is worth noting that when the system keeps active only a single cluster, the power consumption can be varied from 335 to 365 (from 0.8GHz to 1.9GHz, respectively), meaning that speed scaling techniques can affect less than the 10%. Even when the spread between minimum and maximum consumption is the widest (all the processors are awake), the dynamic dissipation is only around 25%. In other words, the static dissipation is ascribed to dissipate at least the 75% of the overall power.

B. Performance evaluation

The following analysis compares the performance of the partitioning heuristics in terms of average power consumption for different task set parameters. The workload has been generated using the UUniFast [22] algorithm. Once the heuristics are executed, the average power consumption is computed analytically by assuming that all jobs are released at the beginning of the hyperperiod, meaning that each processor ϕ_j is active for $U_j \cdot H$ while it is off until next hyperperiod. This assumption is pessimistic because it leads to having the maximum number of active cores ($\forall \phi_j : U_j > 0$) for the longest time ($\min_{\phi_j} U_j$).

The first experiment considers the average power consumption obtained from the heuristics varying the number of tasks (from 25 to 300 with step 25) at three different utilizations: low ($U = 5.0$), medium ($U = 20$) and high ($U = 35.0$). The results are shown in Figure 4, Figure 5 and Figure 6, respectively. Note that, using the EDF policy, the utilization upper bound is equal to the number of processors (in our case, 48), but it is not easily achievable due to fragmentation.

The first significant result consists of showing the high average consumption obtained from WFD with respect to BFD. Such a result is mainly ascribable to the power model of the board, as it privileges approaches that switch off as many processor as possible rather than reducing their performance. Indeed, all the previous work assumed a cubic power function ($p(s) = \beta \cdot s^3$), which heavily reduces consumption when the speed is scaled down. Note that performance is not affected by the number of tasks, as WFD is slightly affected by fragmentation. Finally, note that applying either DPM or DVFS techniques produces the same result.

As already discussed, BFD fits well with this kind of architecture as it aims at compacting the workload on few cores and putting the others in low-power state. This approach drastically reduces the static dissipation, which accounts for at least 75% of the overall dissipation. According to the presented results, the higher the number of tasks, the lower the average power consumption. This can be easily explained by considering that in a larger task set tasks have smaller utilization, hence the heuristic can better compact them, reducing fragmentation. Concerning the strategies applied after task partitioning, the DVFS approach is more effective up to a certain point, after which DPM becomes more convenient. Such a turning point depends on the overall task utilization and the number of tasks. Basically, for a given utilization, small task sets create more significant fragmentation and such a spare capacity is better exploited by slowing the speed down. On the other hand, when the number of tasks increases, fragmentation reduces and the smaller slack time is optimized by the DPM approach. Generally, we can state that the higher the utilization, the higher the number of tasks that make the DPM strategy more effective. In the first case (Figure 4), the utilization is low ($U = 5.0$) and DPM is already more suitable. When the utilization is $U = 20.0$ (Figure 5), the turning point is at $n = 150$, while it reaches $n = 275$ for $U = 35.0$ (Figure 6).

For the sake of completeness, Figure 7 presents a different reading key, showing the average power consumption with 150 tasks for different utilizations ($U \in [5.0, 40.0]$ with step 2.5). As previously highlighted, the turning point is at $U = 20.0$ with 150 tasks. Moreover, Figure 7 shows what happens when computational times do not entirely scale with the frequency. More precisely, the higher the value of γ, the lower the average power consumption obtained by the DVFS approach. As already highlighted by Bambagini et al. [23], this is due to the fact that scaling speed with high γ is equivalent to schedule a task set with lower utilization than the original one, meaning that lower speeds can be effectively exploited while no deadline is missed.

V. Conclusions

This paper has presented a comparison between two opposite heuristics: Worst-Fit Decreasing and Best-Fit Decreasing. The first approach aims at exploiting all the available processors and reducing the overall performance while the second attempts to reduce the number of active processors by optimizing execution.

Our analysis showed that, even though the WFD heuristic is the actual state of art in the research literature, on a real platform it requires a higher average power than BFD approach. This result is a direct consequence of modern hardware which is characterized by high static dissipation whose impact is around the 75% of the overall consumption.

As future work, we would like to extend the presented analysis by further considering the fragmentation problem and addressing the delays related to the shared bus.

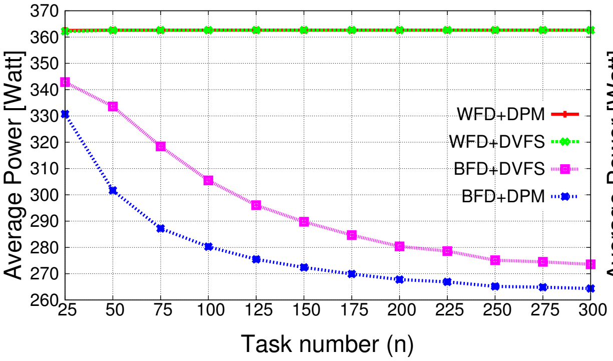

Fig. 4. Average power varying the task number at $U = 5.0$.

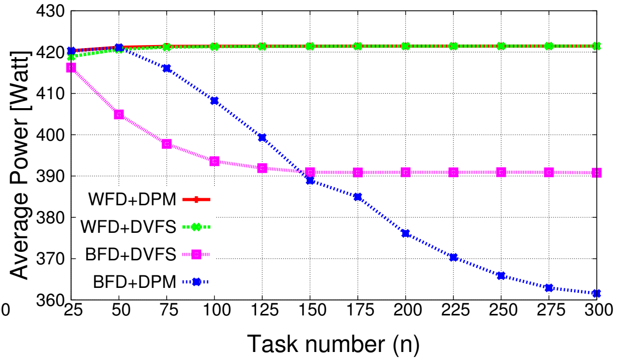

Fig. 5. Average power varying the task number at $U = 20.0$.

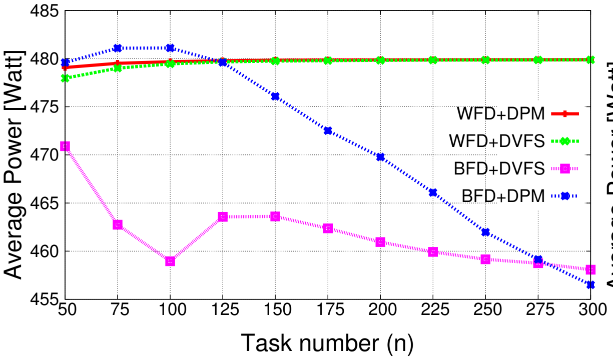

Fig. 6. Average power varying the task number at $U = 35.0$.

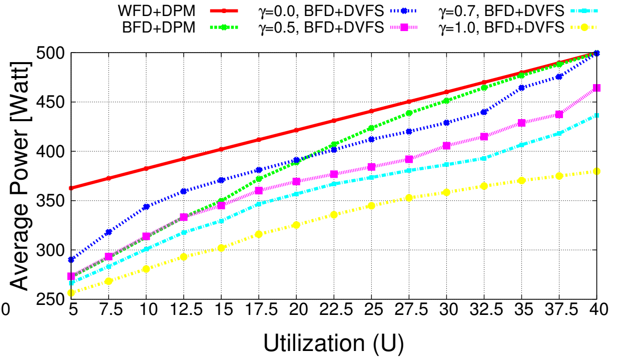

Fig. 7. Average power varying U and γ with $n = 150$.

References

[1] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low power cmos digital design," *IEEE Journal of Solid State Circuits*, vol. 27, 1995.

[2] "Acpi web site," http://www.acpi.info/.

[3] "Dell web site," http://www.dell.com/.

[4] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, ser. FOCS'95, 1995.

[5] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS)*, 2003.

[6] C.-Y. Yang, J.-J. Chen, and T.-W. Kuo, "An approximation algorithm for energy-efficient scheduling on a chip multiprocessor," in *Proceedings of the conference on Design, Automation and Test in Europe (DATE)*, 2005.

[7] A. Kandhalu, J. Kim, K. Lakshmanan, and R. R. Rajkumar, "Energy-aware partitioned fixed-priority scheduling for chip multi-processors," in *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2011.

[8] J. Huang, C. Buckl, A. Raabe, and A. Knoll, "Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems," in *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP)*, ser. PDP'11, 2011.

[9] P. Santiago and C. Jian-Jia, "Single frequency approximation scheme for energy efficiency on a multi-core voltage island," in *Proceedings of the 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2013.

[10] V. Petrucci, O. Loques, D. Mosse, R. Melhem, N. A. Gazala, and S. Gobriel, "Thread assignment optimization with real-time performance and memory bandwidth guarantees for energy-efficient heterogeneous multi-core systems," in *Proceedings of the 17th IEEE International Conference on Real-Time and Embedded Technology and Application Symposium (RTAS)*, 2012.

[11] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar, "Power and thermal management in the intel core duo processor," in *Intel technology Journal*, vol. 10, 2006.

[12] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "Power aware computing," 2002, ch. The case for power management in web servers.

[13] T. Martin and D. Siewiorek, "Non-ideal battery and main memory effects on cpu speed-setting for low power," *IEEE Transactions on VLSI Systems*, vol. 9, 2001.

[14] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, January 1973.

[15] M. Bambagini, F. Prosperi, M. Marinoni, and G. Buttazzo, "Energy management for tiny real-time kernels," in *Proceedings of the IEEE International Conference on Energy Aware Computing (ICEAC)*, 2011.

[16] M. Marinoni, M. Bambagini, F. Prosperi, F. Esposito, G. Franchino, L. Santinelli, and G. Buttazzo, "Platform-aware bandwidth-oriented energy management algorithm for real-time embedded systems," in *Proceedings of the IEEE International Conference on Emerging Technologies & Factory Automation (ETFA)*, 2011.

[17] M. Bambagini, M. Bertogna, M. Marinoni, and G. Buttazzo, "An energy-aware algorithm exploiting limited preemptive scheduling under fixed priorities," 2013.

[18] J.-J. Chen and T.-W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor," in *Proceedings of the ACM Conference on Language, Compilers, and Tool support for Embedded Systems*, ser. LCTES '06. ACM, 2006, pp. 153–162.

[19] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *Computers, IEEE Transactions on*, vol. 53, no. 5, pp. 584–600, 2004.

[20] "Gnu/linux kernel," https://www.kernel.org/.

[21] "Eembc coremark benchmark," http://www.eembc.org/coremark/.

[22] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, May 2005.

[23] M. Bambagini, G. Buttazzo, and M. Bertogna, "Energy-aware scheduling for tasks with mixed energy requirements," in *Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2013.