The 2nd Int'l WORKSHOP ON POWER-AWARE REAL-TIME COMPUTING (PARC 2005)

September 22, 2005
Jersey City, New Jersey, USA

In conjunction with the EMSoft'05

Organizers

Program Chairs

Hakan Aydin	George Mason University, USA
Dakai Zhu	University of Texas at San Antonio, USA

Program Committee

Pedro Mejia Alvarez	CINVESTAV-INP, Mexico
Frank Bellosa	University of Karlsruhe, Germany
Luca Benini	University of Bologna, Italy
Bruce Childers	University of Pittsburgh, USA
Steve Goddard	University of Nebraska-Lincoln, USA
Rajesh Gupta	University of California at San Diego, USA
Sharon Hu	University of Notre Dame, USA
Mahmut T. Kandemir	Pennsylvania State Univesity , USA
Dong-In Kang	Information Sciences Institute, USA
Jihong Kim	Seoul National University, Korea
Ulrich Kremer	Rutgers University, USA
Giuseppe Lipari	SSSUP, Italy
David K. Lowenthal	University of Georgia, USA
Daniel Mossé	University of Pittsburgh, USA
Christian Poellabauer	University of Notre Dame, USA
Gang Quan	University of South Carolina, USA
Fan Zhang	Hong Kong Univ. of Science & Technology, HK
Yuanyuan Zhou	University of Illinois at Urbana-Champaign, USA

Program of PARC'05

September 22, 2005

Jersey City, New Jersey, USA

- **8:50 to 9:00 -- Opening Remarks by Program Chairs**

- **9:00 to 10:00 -- Keynote Speech**
  - Trevor Mudge (University of Michigan):
    Low-Power Robust Computing

- **10:00 to 10:30 -- Coffee Break**

- **10:30 to 12:00 -- Dynamic Voltage Scaling**
  - Ruibin Xu, Daniel Mosse and Rami Melhem (Univ. of Pittsburgh),
    Evaluating a DVS Scheme for Real-Time Embedded Systems

  - M. Angels Moncusi, Alex Arenas (Univ. Rovira i Virgili), Jesus Labarta (Univ. Politecnica de Catalunya),
    Moving Average Frequency Reduction for Low Power in Hard Real-Time Systems

  - Claudio Scordino (Univ. of Pisa) and Enrico Bini (Scuola Superiore Sant'Anna),
    Optimal Speed Assignment for Probabilistic Execution Times;

- **12:30 to 14:00 -- Lunch**

- **14:00 to 15:30 -- Global Power Management**
  - Hui Cheng and Steve Goddard (Univ. of Nebraska-Lincoln),
    Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation

  - Jian-Jia Chen and Tei-Wei Kuo (National Taiwan Univ.),
    Energy-Efficient Scheduling of Periodic Real-Time Tasks over Homogeneous Multiprocessors

  - Vibhore Vardhan, Daniel Grobe Sachs, Wanghong Yuan, Albert F. Harris, Sarita V. Adve, Douglas L. Jones, Robin H. Kravets, and Klara Nahrstedt (Univ. of Illinois at Urbana-Champaign)
    Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy

- **15:30 to 16:00 -- Coffee break**

- **16:00 to 17:00 -- Profiling and Instrumentation**
  - David Snowdon, Sergio Ruocco and Gernot Heiser (National ICT Australia and University of New South Wales), ***Power Management and Dynamic Voltage Scaling: Myths and Facts***

  - Graham R. Hellestrand, Mahdi Seddighnezhad and Jim Brogan (VaST Systems Technology Corporation) ***Profiles in Power: Optimizing Real-Time Systems for Performance and Power***

- **17:00 to 17:10 -- Closing Remarks**

Evaluating a DVS Scheme for Real-Time Embedded Systems *

Ruibin Xu, Daniel Mossé, Rami Melhem
Computer Science Department, University of Pittsburgh
{*xruibin,mosse,melhem*}*@cs.pitt.edu*

Abstract

Dynamic voltage scaling (DVS) has become a well-known and effective technique to exploit energy-performance trade-off in real-time embedded systems where energy imposes a major constraint. We focus on frame-based real-time systems that execute variable workloads with the goal of minimizing expected energy consumption in the system while still meeting the deadlines. In our separate publication, we proposed a new DVS scheme that incorporates the dynamic behavior of the tasks into the speed schedule and aims to minimize the expected energy consumption in the system. The new DVS scheme was derived based on the assumption of unrestricted continuous frequency. However, it remains unknown how the new DVS scheme performs in practical situations. In this paper, we first give a simple example through which we demonstrate the new DVS scheme and compare it with the existing DVS schemes. Then we present evaluation results to show that the new DVS scheme achieves significant energy savings over the existing schemes.

1 Introduction

Energy conservation is critically important for many real-time systems such as battery-operated embedded systems which have a restricted energy budget . Dynamic voltage scaling (DVS), which involves dynamically adjusting the voltage and frequency of the CPU, has become a well-known technique in power management for real-time embedded systems. Through DVS, quadratic energy savings can be achieved at the expense of just linear performance loss [14, 4]. Thus, the execution of tasks can be slowed down in order to save energy, as long as the deadline constraints are not violated. A natural problem that rises from this context is how to minimize the energy consumption in the system while still meeting the deadlines. The problem is often reduced to determining a task's speed (or equivalently, determining the amount of time allotted to a task) before it is scheduled to execute in the system.

The systems under our consideration are frame-based hard real-time embedded systems that execute variable workloads. The tasks in these systems exhibits dynamic behavior in the sense that they usually run for less than their worst-case execution times (WCET) and the execution time of the tasks is unpredictable before their execution. Therefore, the design goal of DVS schemes becomes *minimizing the expected (total) energy consumption* in the system.

In [12], we proposed a new DVS scheme that incorporates the dynamic behavior of the tasks into the speed schedule and aims to minimize the expected energy consumption in the system. To our knowledge, this is the first DVS scheme that is designed explicitly to Minimize the Expected Energy Consumption for frame-based real-time systems. Therefore, we call the new DVS scheme MEEC throughout this paper. The MEEC scheme was derived based on the assumption of unrestricted continuous frequency. We also extended it to take into consideration the practical issues, such as minimum and maximum frequency restriction, and provided solutions to the problems. However, it remains unknown how the MEEC scheme performs under all the practical considerations. In this paper, we first give a simple example through which we demonstrate the MEEC scheme and compare it with the existing DVS schemes. We show that failure to capture the dynamic behavior of the tasks by the existing DVS schemes and naive use of dynamic behavior information will lead to suboptimal power management. Then we present extensive evaluation results, both on synthetic and real-life workloads, to show that the MEEC scheme can achieve significant energy savings over the existing schemes.

This paper is organized in the following way. We first present the related work in Section 2. The system and task model are described in Section 3. We demonstrate the MEEC scheme and compare it with the existing schemes in Section 4. Evaluation results are presented in Section 5. We end the paper in Section 6 with concluding remarks.

*This work has been supported by NSF grant ANI-0125704 and ANI-0325353.

2 Related Work

Although much work has been done on exploring DVS in real-time environments, we will focus on the related work that takes into consideration actual (not worst-case) execution time of tasks. This is because real-time applications usually exhibit a large variation in actual execution times (e.g., [3] reported that the ratio of the worst-case execution time to the best-case execution time can be as high as 10 in typical applications; our measurements in [10] show that this ratio can be as high as 100), and thus the DVS schemes that use exclusively worst-case execution time lack the advantage of unused computation time. Besides frame-based real-time systems, we will also focus on the related work that applies to periodic real-time systems because frame-based real-time system is a special case of periodic real-time system.

DVS in real-time applications is categorized as *inter-task* or *intra-task* voltage scaling [5]. Inter-task schedules speed changes at each task boundary, while intra-task schedules speed changes within a single task. For inter-task voltage scaling, Mossé et al. [8] introduced the concept of *speculative speed reduction* and proposed three DVS schemes with different speed reduction aggressiveness for frame-based real-time systems. Aydin et al. [2] and Pillai et al. [9] independently proposed DVS schemes for achieving high energy savings for periodic real-time systems. They both precompute a static optimal schedule assuming that each task runs for WCEC and when a task runs for less than its WCEC, the scheduler uses the slack to create a new schedule for the remaining tasks. However, the exclusive use of static information in computing speed schedules by [8, 2, 9] leads to suboptimal power management for the system. The MEEC scheme makes use of both static and dynamic information to design the speed schedule. To be able to navigate the full spectrum of speculative speed reduction, in [2] system designers can set a parameter to control the degree of speed reduction aggressiveness. The MEEC scheme chooses the degree of speed reduction aggressiveness automatically, based the probability distribution of the workload of the tasks, to minimize the expected energy consumption.

For intra-task voltage scaling, Lorch et al. [6] have shown that if a task's computational requirement is only known probabilistically, there is no constant optimal speed for the task and the expected energy consumption is minimized by gradually increasing speed as the task progresses, which is an approach named as *Processor Acceleration to Conserve Energy* (PACE). Practical PACE (PPACE) [13] takes into consideration a number of practical issues and improves the performance of PACE. However, PACE and PPACE have only been studied for single task when considering hard real-time guarantee. In [7], PACE is used for soft real-time systems when the system has only one task but the maximum speed is used when the system has multiple tasks. In [12], we presented the theoretical results of using PACE for multiple tasks with a single hard deadline (frame length). We also show that a naive extension of PACE for multiple tasks is not recommended in Section 4.

AbouGhazaleh et al. [1] proposed a hybrid compiler-operating system intra-task DVS scheme for energy consumption of time-sensitive embedded applications. The MEEC scheme is implemented at the operating system level and assumes no access to application source codes.

3 Task and System Model

We consider a frame-based task model with N periodic tasks in the system, all ready at time zero. The task set is denoted by $T = \{T_1, T_2, \ldots, T_N\}$. Each task T_i ($1 \leq i \leq N$) is characterized by its worst-case execution cycles (WCEC) W_i and the probability density function of its execution cycles $P_i(x)$, which denotes the probability that task T_i executes for x ($1 \leq x \leq W_i$) cycles. Obviously, we have $\sum_{x=1}^{W_i} P_i(x) = 1$ and $P_i(W_i) \neq 0$. The average-case execution cycles (ACEC) of T_i can be computed as $\sum_{x=1}^{W_i} P_i(x)x$. All task periods are identical and all task deadlines are equal to their period. The common deadline/period (also known as frame length) is denoted by D. The execution of the frame is to be repeated and all tasks must be executed during each frame in the order of $T_1, T_2, \ldots, T_N$. Thus, the tasks can be treated as sequential sections of a single application. If the execution order of the tasks is flexible, the ordering strategies can be found in [12].

The tasks are to be executed on a variable voltage processor with the ability to dynamically adjust its frequency and voltage on application requests. Because processor is the major power consumer for many embedded systems, reducing processor energy consumption has a significant impact on the overall system energy consumption. In deriving the MEEC scheme [12], we assume that the processor frequency can be adjusted continuously from 0 to infinity. We also discuss the more realistic cases, such as the processor has minimum and maximum frequencies, in [12]. The processor power consumption when running at frequency f is $c_0 + c_1 f^\alpha$ (α is a constant that is at least 2) where c_0 and c_1 denote the power consumption of the processor when idle and the maximum dynamic power respectively. The dynamic power is determined by the processor operating frequency and the maximum dynamic power is the dynamic power consumed when the processor is operating at the maximum frequency.

4 The DVS Schemes

In this section, we give a simple example through which we demonstrate the MEEC scheme and compare it with the

Table 1. The parameters for the 3 tasks in the simple example

Task	W	P(x)	ACEC
T_1	2	$.9, .1$	1.1
T_2	4	$.9, 0, 0, .1$	1.3
T_3	2	$.5, .5$	1.5
$\hat{T}$	8	$0, 0, .405, .45, .045, .045, 0.05, .005$	3.9

existing schemes.

Example Suppose that there are 3 tasks in the frame-based real-time system with a frame length of 14 time units. The workload of a task in expressed in *super cycles*. A super cycle consists of a certain number of CPU cycles, which can be computed in order to keep the overhead of DVS low [1]. The tasks are required to be executed in the order of T_1, T_2, and T_3. The parameters for the 3 tasks are shown in Table 1. We also treat the three tasks as the three sequential sections of a single task $\hat{T}$ and its parameters are computed from those of the 3 tasks. $\hat{T}$ is used for the naive extension of PACE shown at the end of this section. For the processor, we suppose that $c_0 = 0$ and $c_1 = 1$. The maximum speed of the processor is 1 super cycle per time unit and the minimum speed of the processor is 0.

We start by reviewing the existing DVS schemes, which can be categorized into 3 schemes: proportional scheme, greedy scheme, and statistical scheme. The proportional scheme and greedy scheme only make use of WCEC and deadline information. The proportional scheme distributes the slack proportionally among all unexecuted tasks. Thus, in the example, the proportional scheme will start executing T_1 using speed $\frac{2+4+2}{14} = 0.5714$. The greedy scheme is more aggressive, because it gives all the slack to the next ready-to-run task. Therefore, the greedy scheme will start executing T_1 using speed $\frac{2}{14-(4+2)/1} = 0.25$. Note that the greedy scheme is using the lowest possible speed to execute the next task. The statistical scheme tries to take advantage of the average-case execution cycles (ACEC) of the tasks, to distribute the reclaimed slack, the natural slack, and the slack that would appear in the system if other tasks were to finish early. To guarantee that the deadline is not missed, the statistical scheme chooses the maximum of the speed obtained from the greedy scheme and the speed computed based the ACEC of the tasks. Thus, the statistical scheme will start executing T_1 using speed $max(0.25, \frac{1.1+1.3+1.5}{14}) = 0.2786$.

After a task finishes, the system reclaims the slack created by the task if it runs for less than its WCEC, and compute the speed of the next task recursively. This is also a common part of all dynamic-claiming DVS schemes, as follows. Let us see how they compute the speed for T_2 after T_1 finishes. Suppose that T_1 only runs for 1 super cycle. Then the time left for executing T_2 and T_3 in the proportional scheme is $14 - \frac{1}{0.5714} = 12.2499$, and speed $\frac{4+2}{12.2499} = 0.4898$ will be used to execute T_2. Similarly, the greedy scheme will use speed $\frac{4}{14-1/0.25-2/1} = 0.5$ to execute T_2, and the statistical scheme will use speed $max(\frac{4}{14-1/0.2786-2/1}, \frac{1.3+1.5}{14-1/0.2786}) = 0.4756$ to execute T_2.

Intuitively, when tasks tend to run close to their WCECs, the proportional scheme would perform well; when tasks tend to run much less than their WCECs, the greedy scheme would have good performance. The statistical scheme tries to strike a balance between proportional scheme and greedy scheme. However, none of them is optimal in terms of minimizing the expected energy consumption in the system.

The MEEC scheme incorporates the dynamic behavior of the tasks into the speed schedule. The dynamic behavior of the tasks is captured by the probability density function of the workload of the tasks, which is represented by histograms in practice. When using profiling to obtain WCEC and ACEC, the probability density function of the workload of the tasks can be also learned at the same time, only requiring certain amount of additional storage.

DVS Algorithm The MEEC scheme is divided into two phases: (a) the offline phase precomputes the speed schedule, which consists of the percentage factor β_i for each task T_i. The percentage factor β_i determines the speed to execute T_i: when T_i is ready to execute and the time left in the frame to execute $T_i, T_{i+1}, \ldots, T_N$ is d, then time $\beta_i d$ is allocated to execute T_i. The algorithm computes the percentage factors in the reverse order, that is, first compute β_N, then β_{N-1}, ..., and last β_1. The value of β_N is always 100% and the other percentage factors can be computed recursively and efficiently thanks to the nice property of energy function of the tasks. More details can be found in [12]; (b) the online phase is invoked before the execution of each task, obtaining the time left in the frame and computing the execution speed for the task: when starting executing task T_i and having time d left, set the speed to $\frac{W_i}{\beta_i d}$ (the actual speed value needs to be adjusted according to the available discrete speeds of the processor). Both phases are efficient: the offline phase runs in polynomial time and the online phase only takes constant time.

In the example, the percentage factors for T_1, T_2, T_3 can be computed to be equal to $39.38\%, 76.19\%, 100\%$, respectively. Thus, the MEEC scheme will use speed $\frac{2}{39.38\% \times 14} = 0.3628$ to execute T_1. If T_1 runs for 1 super cycle, then the time left for executing T_2 and T_3 is $14 - \frac{1}{0.3628} = 11.2737$. Then the MEEC scheme will use speed $\frac{4}{76.19\% \times 11.2737} = 0.4657$ to execute T_2. Table 3 shows the expected energy consumption per frame for all DVS schemes and the savings of the MEEC scheme over the other schemes. In [12], we prove that the MEEC scheme minimizes the expected energy consumption in the system under the assumption of

Table 2. The comparison of all DVS schemes for the simple example

Scheme	Expected energy consumption per frame	Saving
naive PACE	0.7953	23%
proportional	0.7733	21%
greedy	0.7388	17%
statistical	0.6771	10%
MEEC	0.6097	–

unrestricted continuous frequency.

Finally, we show through the simple example that a naive extension of PACE (or, naive PACE for short) cannot obtain energy savings over the DVS schemes that do not use intra-task voltage scaling. Since PACE has only been studied for a single task, the naive PACE treats all the tasks as a single super task and derives its parameters (WCEC and probability distribution of the workload) from those of the original tasks. For the example, the parameters for the super task $\hat{T}$ are shown in Table 1. For this super task, using PACE [6] will result in expected energy consumption per frame of 0.7953, which is the worst of all DVS schemes discussed so far. The reason why the naive PACE fails is that treating all tasks as a single super task results in loss of information (e.g., the naive PACE cannot determine when tasks terminate), losing the opportunity for dynamic slack reclamation. For instance, if task T_1 runs only for 1 super cycle, we can be sure that the rest of workload in the current frame is at most 6 super cycles. However, the naive PACE still assumes that the rest of workload is 7 super cycles in the worst case. In [12], we show that PACE must be used for executing individual tasks in order to obtain further energy savings over the DVS schemes that do not use intra-task voltage scaling.

5 Evaluation

The optimality of the MEEC scheme [12] only holds if we assume unrestricted continuous frequency which does not hold in practice. Therefore, we also discuss the issues that arise when our DVS scheme is used in practice and provide solutions to the problems in [12]. However, it remains unknown how the MEEC scheme performs under those practical considerations. To answer this question, we conducted extensive simulations for different power models and different workloads.

5.1 Power Models

We used two power models in our simulation. The first power model is a synthetic processor that strictly conforms to the $p(f) = f^3$ power-frequency relation and has 10 discrete frequencies ranging from 100MHz to 1000MHz with

Table 3. XScale speed settings and power consumptions

Speed (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8
Power (mW)	80	170	400	900	1600

100MHz step; its idle power is zero. The second power model is the Intel XScale (Table 3) [11]. For the idle power of Intel XScale, we assume that the CPU operates at the lowest frequency (i.e., 150 MHz) when idle. This is equivalent to say that the idle power is 80 mW. The power function for XScale used in deriving β_i's is

$$p(f) = 80 + 1520(\frac{f}{1000})^3 \quad (1)$$

where f is the frequency. Figure 1 shows that Equation (1) is a good approximation of the actual power function of Intel XScale.

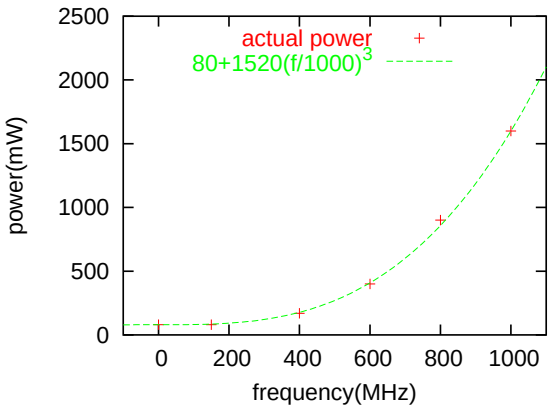
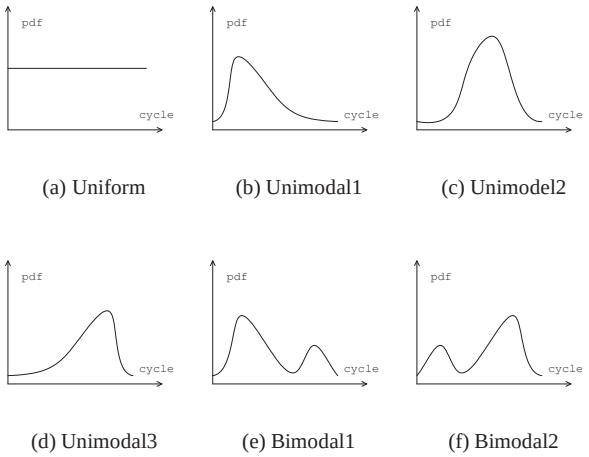


Figure 1. Approximate power function for Intel XScale

5.2 Synthetic Workloads

A frame-based real-time systems is characterized by the number of tasks, the WCEC of each task, the probability distribution of the workload of each task, the frame length. We simulated system that have 5, 10, 15, 20 tasks, respectively. We only show the results for the systems with 5 tasks because the results for systems with different number of tasks are similar. The WCEC of each task is randomly generated from 10,000,000 cycles to 1,000,000,000 cycles. The probability density function of each task's actual execution cycles is randomly chosen from 6 representative distributions shown in Figure 2. The bin width of the histograms denoting the probability density functions is 1,000,000 cycles. For each combination of the tasks, we computed the

worst-case finishing time (t) for a frame running at the highest speed. Then we varied the frame length from $1.2t$ to $4t$. For each simulated system (i.e., for each run with a set of tasks), we evaluated 8 DVS schemes: proportional without PACE (P), proportional with PACE [1] (PP), greedy without PACE (G), greedy with PACE (GP), statistical without PACE (S), statistical with PACE (SP), MEEC without PACE (M), MEEC with PACE (MP). For each experiment, we generated 100,000 frames and computed the average energy consumption per frame for each scheme. Under this experimental setup, we conducted over one million runs and averaged the results (which are shown here).



(a) Uniform (b) Unimodal1 (c) Unimodel2

(d) Unimodal3 (e) Bimodal1 (f) Bimodal2

Figure 2. Candidate probability density functions

For all the simulations using the synthetic CPU, the best scheme is always scheme M or MP, but scheme MP is only better than scheme M for 13.6% of the time with an average saving of 1.2% over scheme M. For all the simulations using XScale, the best scheme is always scheme M. Note that, in the simulations, we ignore the speed change overhead and online scheduling overhead, and thus we favor schemes using PACE. For the other schemes, scheme PP outperforms scheme P most of the time, but scheme G (S) outperforms scheme GP (SP) most of the time. The simulation results support our conjectures about using PACE in frame-based real-time systems in [12]. Therefore, PACE is not recommended in the MEEC scheme.

Next, we compare the MEEC scheme with other schemes, all without using PACE. Figure 3 shows the maximum and average energy savings of our scheme over other schemes for both the synthetic CPU and XScale. From the figure we can see that the MEEC scheme achieves an average of 20.45% (up to 33.45%) energy savings over the next best scheme (proportional) for the synthetic CPU, and an average of 6.52% (up to 20.85%) energy savings over the next best scheme (statistical) for XScale. The energy savings are significant. The two key factors that affect the energy savings are the minimum speed of the CPU and the number of speeds available from the CPU. In computing the speed schedules, the MEEC scheme assumes unrestricted continuous frequency. Because of the convexity of the power function, high speed is not usually obtained by the MEEC scheme. But low speed is desired because the MEEC scheme can navigate the full spectrum of available speeds and can find the best speed that minimizes the expected energy consumption. The importance of the number of speeds available from the CPU is obvious given that we need to convert the continuous speeds to discrete speeds. Therefore, because the minimum speed of the synthetic CPU is less than that of XScale and the number of speeds of the synthetic CPU is greater than that of XScale, the energy saving for XScale is less than the synthetic CPU.

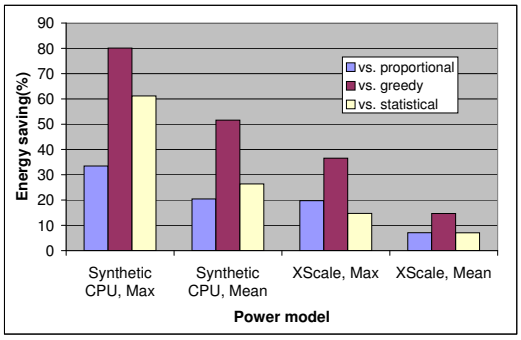


Figure 3. Energy savings of the MEEC scheme over other schemes for the synthetic workload

5.3 Automatic Target Recognition (ATR)

The ATR application[2] does pattern matching of targets in images. In ATR, the regions of interest (ROI) in one image are detected and each ROI is compared with all the templates. The number of target detections in each frame varies from 0 to 8 detections. Image processing time is proportional to the number of detections within an image.

In our system model, a front-end is responsible for collecting images and sending the images periodically to a back-end equipped with an Intel XScale CPU for target recognition. The back-end is required to finish processing all the images that it receives by the end of the period (frame) in order to process the next batch of images

[1] When the time allocated to execute a task is determined, use PACE technique to execute this task within the allocated time. The same holds for using PACE for other schemes.

[2] The original code and data for this application were provided by our industrial research partners

in a timely fashion. The period is 100 ms and the front-end sends 1 to 6 images to the back-end for one period.

Each task processes an image with 1 to 8 ROIs. We obtained the probability distribution of the workload of the task by profiling on a training image set, then precomputed the speed schedule (that is, computed the β_i values, see Section 4) for having $1, 2, 3, 4, 5, 6$ images to be processed in one period (frame), respectively. The six speed schedules are stored in the back-end. When a period begins, the back-end counts the number of images received and applies the corresponding speed schedule. Figure 4 shows the energy savings of the MEEC scheme over other schemes when the back-end has $1, 2, 3, 4, 5, 6$ images to process. From the figure we can see that the MEEC scheme can achieve an average of 11.04% energy savings (not counting the case for 1 image because all schemes achieve the same performance in this case) over the next best scheme (statistical).

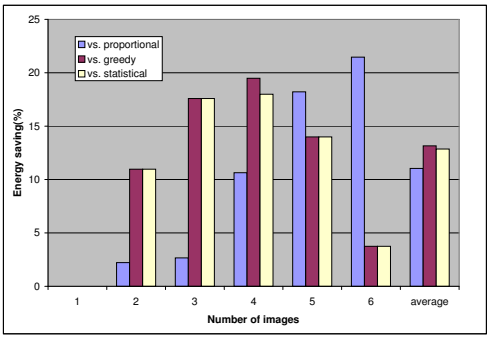


Figure 4. Energy savings of the MEEC scheme over other schemes for ATR

6 Conclusions

In this paper, we demonstrate and present extensive evaluations of the MEEC scheme proposed in [12]. We first review the existing DVS schemes and demonstrate the MEEC scheme through a simple example. Then we evaluate the existing DVS schemes and the MEEC scheme through different power models and different workloads. Evaluation results show that the MEEC scheme can achieve significant energy savings over the existing schemes. Another important conclusion from this work is the demonstration that using only static information or aggregating dynamic information, even with probabilistic techniques, will not produce as good results as when dynamic information for each task is considered separately.

Future work will investigate the case of the problem where different tasks have different deadlines.

References

[1] N. AbouGhazaleh, D. Mossé, B. Childers, R. Melhem, and Matthew Craven. Collaborative Operating System and Compiler Power Management for Real-Time Applications. In *RTAS*, May 2003.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *RTSS*, pages 95–105, December 2001.

[3] R. Ernst and W. Ye. Embedded Program Timing Analysis based on Path Clustering and Architecture Classification. In *ICCAD*, San Jose, CA, 1997.

[4] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *RTSS*, Madrid, Spain, December 1998.

[5] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *RTSS*, December 2002.

[6] J. Lorch and A. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *ACM SIGMETRICS*, June 2001.

[7] J. Lorch and A. Smith. Operating system modifications for task-based speed and voltage scheduling. In *MobiSys*, May 2003.

[8] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler-Assisted Dynamic Power-aware Scheduling for Real-Time Applications. In *COLP*, October 2000.

[9] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *SOSP*, pages 89–102, October 2001.

[10] C. Rusu, R. Xu, R. Melhem, and D. Mossé. Energy-Efficient Policies for Request-Driven Soft Real-Time Systems. In *ECRTS*, Catania, Italy, July 2004.

[11] Intel xscale microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[12] R. Xu, D. Mossé, and R. Melhem. Minimizing Expected Energy in Real-Time Embedded Systems. In *EMSOFT*, Jersey City, New Jersey, September 2005.

[13] R. Xu, C. Xi, R. Melhem, and D. Mossé. Practical PACE for Embedded Systems. In *EMSOFT*, Pisa, Italy, September 2004.

[14] F. Yao, A. Demers, and S.Shankar. A Scheduling Model for Reduced CPU Energy. In *FOCS*, pages 374–382, 1995.

Moving Average Frequency Reduction for Low Power in Hard Real Time Systems*

M.Angels Moncusí, Alex Arenas
{amoncusi,aarenas}@etse.urv.es
Dpt d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
Campus Sescelades, Av dels Països Catalans, 26
43007 Tarragona, Spain

Jesus Labarta
jesus@ac.upc.es
Dpt d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Jordi Girona, 1-3. D6 Campus Nord
08034 Barcelona, Spain

Abstract

We present a new policy to improve power saving in hard real time systems guarantying all tasks deadlines based on a moving average frequency reduction. Our study focus on the improvement obtained using this policy on the low power dual priority scheduling algorithm [3]. The resulting modified algorithm uses the total workload, the task execution history and the breakdown utilization to estimate the average minimum frequency of the processor to accomplish maximal energy reduction while meeting deadlines. The moving average strategy has been proposed to estimate the empirical execution time beyond the WCET, then updating the processor frequency for every task accordingly. We have performed extensive simulations that show a considerable enhancement in energy saving compared to original low power dual priority scheduling algorithm.

Keywords**:** Static priorities, power aware scheduling, Dynamic Voltage Scaling, Worst Case Execution Time.

1. Introduction

The energy consumption in portable and hard real time systems is a fundamental problem in the design of modern computational devices [1]. A lot of efforts have been made during the last decade to minimize this drawback, but the high performance of modern microprocessors and micro-controllers jointly with the increasing functionality of them obtained via software still require improvements in the power-efficiency context.

The dynamic power consumption in CMOS circuits is given by the equation $P \cong p_t\ C_L\ V_{DD}^2\ f$, where P is the power consumption, p_t is the probability of switching in power transition, C_L is the load capacitance, V_{DD} is the voltage supply and f is the operating clock frequency. Since the power has a quadratic dependency on the supply voltage, it is always energetically favorable scaling the voltage supply down. If the processor uses the voltage scaling technique to scale frequency, the relation between power and frequency is given by $P(f) \propto C_L\ V_{DD}^2\ f \approx k\ f^3$ [2-4]. The main techniques that take advantage of this non-linear dependence are: Clustering Voltage Scaling and Dynamic Voltage Scaling (CVS and DVS) [5-6] Its functioning is based on the reduction of the voltage supply along with the processor frequency, and have been successfully used in many applications.

In hard real time systems these techniques could affect adversely the system performance, because time restrictions are critical. Nevertheless, the DVS technique is used in hard real time systems via power aware scheduling algorithms that determine the operating frequency of the processor that guarantees all real-time constraints while minimizing the energy consumption. Generally speaking, the scheduling algorithms reduce the voltage supply along with the processor operating frequency whenever the full system performance is not necessary and the tasks deadlines are not going to be compromised. Basically, these power aware schedulers use the idle time intervals to slow down the processor, executing tasks at reduced operating frequency.

To calculate the abovementioned reduction of the operating frequency, there are mainly two different approaches: static and dynamic [4,6-10]. In the static approach, the frequency is calculated off-line, before runtime, for each task independently. Once the execution starts the frequency could be readjusted on-line depending on the dynamic slack that has been generated – i.e. part of the worst case execution time not consumed. In the dynamic approach, the operating frequency is calculated on-line, just before running each task, once the scheduler knows exactly what the history about the previous executed tasks have been and when the rest of tasks will arrive [4,8,10].

We will focus our attention on the dynamic approach. In this approach, whenever it exist more than one task in the system, the operating frequency reduction could be performed following at least three general policies:

- Executing ready tasks at the maximum processor operating frequency, and reducing the operating frequency only to execute the last task in the system. This conservative approach cannot use the idle time

* This research is supported by MCYT project number TIN2004-07739-C02-01

that could appear if the last task does not consume all its WCET because there is no task ready to be executed. The Cycle-conservative RT-DVS [8], the dynamic Reclaiming algorithm [10] and the Low Power Fixed Priority algorithm [11] uses a similar policy. See the sketch a) in Figure 1.

- Executing the first task at reduced speed and the following tasks at maximum operating frequency, this is a greedy approach. The first task executed uses the maximum possible idle time to reduce the clock operating frequency while the rest of tasks have to be executed necessarily at the maximum operating frequency. The advantage of this policy is that if a task does not consume all its WCET, the following task can use this time and then its operating frequency can be reduced. The Look-Ahead RT-DVS [8], the Aggressive Speed Reduction [10] and the Power Low Modified Scheduling Algorithm [3,12] uses a similar policy. See the sketch b) in Figure 1.
- Executing all ready tasks at some reduced operating frequencies whenever is possible. This scheme is similar to the static calculation of the operating frequency. Within this scheme all tasks execute at reduced operating frequency, trying to avoid any task execution at maximum operating frequency. In this case if the tasks finishes earlier, the slack generated can be used to reduce the operating frequency of the next task. See the sketch c) in Figure 1.

In this paper, we expose how to implement a dynamic approximation to the last policy in a dual priority scheme.

To motivate our work, based on the execution of tasks at a certain average frequency, let us compare the differences in energy saving obtained using the three aforementioned policies in a toy model. Let us assume a task set formed by two tasks, $task_1$ and $task_2$, with a period and deadline of 100 time units, and a WCET of 30 time units each. Their execution is sketched in Figure 1.

Depending on the different operating frequencies at which the tasks represented in Figure 1 are executed, the total energy consumption values are: a) E=3.52, b) E=3.52 and finally in c) E=2.16. Note that, in the latest policy, both tasks execute at a certain average operating frequency that shows to be clearly energetically favorable. This efficiency relies on the fact that the maximum operating frequency has been avoided, and because the relation between energy consumption and operating frequency is quadratic.

Note that the value for the average frequency used in Figure 1c corresponds exactly to the processor utilization percentage, in our case 60%, and this indicates a strong relationship between them. A similar approach considering the frequency reduction as a function of the processor utilization has been used in [9,10] for the Earliest Deadline First scheduling (EDF).

The rest of the paper is structured as follows: in section 2 we set the framework of the system, in section 3 we present a simple example of efficient reduction for Rate Monotonic scheduling. In section 4, we expose the modifications in the low power dual priority algorithm. In section 5, we compare the efficiency of the described policy with two energy aware scheduling algorithms. Finally in section 6 we present the conclusions of the current work.

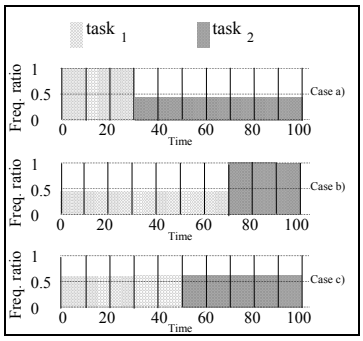


Figure 1: Three possible policies for tasks execution at different operating frequencies: a) $task_1$ is executed at the maximum operating frequency, and $task_2$ is executed at 0.42 of the maximum operating frequency; b) $task_1$ is executed at 0.42 and $task_2$ executes at the maximum operating frequency; c) the execution of both tasks is at 0.6 of the maximum operating frequency.

2. Framework

We consider task sets consisting on n independent periodic tasks, $\tau_1 \ldots \tau_n$, each task τ_i characterized by a 3-tuple (C_i,T_i,D_i), where C_i is the worst case execution time of τ_i. For each task instance the execution time varies from $0.1*C_i$ to C_i. T_i stands for its period (or minimum inter-arrival time), and D_i is its relative deadline. The tasks sets are scheduled using a fixed priority pre-emptive algorithm in a multi-operating frequency processor.

The computation time overhead for context switching and for the scheduler are assumed to be negligible. The extent to which these assumptions are realistic is discussed in the analysis of the algorithm given in [13] and it turns out to be practical if the switch is subsumed to the worst-case execution times of the different tasks. We also assume that the voltage scaling overhead is negligible; the safeness of the system under these conditions is proved on theorem 1 of the work by Shin and Choi [11]. When the processor is powered down we consider zero energy consumption. Note that we also are assuming that the energy consumption is minimized whenever the supply voltage is scaled down. A recent work of Miyoshi et al. [14] has pointed out that there exists some practical processors with energy-inefficient operating frequencies for which this hypothesis does not hold, in these cases the current approach should be correctly to avoid entering the range of non-operative frequencies.

The whole system will be characterized by the processor utilization (U) and the breakdown utilization (BU) [15,16]. The Processor Utilization (U) is defined as

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \quad (1)$$

and the BU is defined as the fraction of the utilization factor that marks the border for a system to be schedulable. To calculate BU, each execution time C_i in a given task set is multiplied by a constant scaling factor α, while the periods remain fixed. The task set is scaled to the point at which it is just schedulable, such that any increment of α would cause at least one task to miss its deadline. The utilization of the task set at that point, Σ_i ($\alpha C_i/T_i$) is the BU [16]. This scaling factor affects schedulability the same way the operating frequency reduction affects, then the BU should be considered as a lower bound to the static operating frequency reduction.

3. Average frequency reduction policy

Suppose that we have a system with only one task whose period and deadline is set to 100 time units and whose WCET is set to 60 time units. The processor utilization is 60%, the hyper-period is 100 and the Breakdown Utilization is 100 %, that is, we can scale the task set up to a real utilization of 100% (i.e. no idle time). In this simple situation, the execution frequency should be set exactly to 0.6 of the maximum processor frequency corresponding to 60(workload)/100(time units). Note that in this case the frequency reduction is optimal, i.e. a reduction below 0.6 makes the system not to meet the deadlines and a reduction over 0.6 will imply larger energy consumption.

Let us now consider an heterogeneous task set (see Table 1). The system is characterized by U=80% and BU=88%. If the scheduler extrapolates the average frequency policy presented before, the estimated frequency turns out to be 0.8. Using this frequency, the WCRTs (Worst Case Response Time) of tasks are 3.75, 26.25 and 63.75 respectively, and therefore $Task_3$ misses its deadline. This simple example shows that this frequency reduction is not feasible due to the real time constraints. A more accurate estimation of the average frequency in this case should take into account that the spanning time of tasks in a Fixed Priority system is constrained by the BU to 88%, then the appropriate frequency should be represented by the ratio U/BU. Using this ratio, the frequency reduction is 0.91 and the WCRTs of tasks are 3.34, 23.36 and 59.4 respectively, consequently all deadlines are meet.

	Period	Deadline	WCET	WCRT
$Task_1$	10	10	3	3
$Task_2$	40	40	12	18
$Task_3$	60	60	12	33

Table 1: Characteristics of the task set.

The determination of the average frequency reduction in a more general situation where there are N tasks in the system competing for real time execution is far more complicated. In this scenario, the tasks priorities as well as the constraints imposed by the deadlines increase the complexity of the optimization problem. An off-line optimization algorithm will not provide the correct values due to the dynamic interferences that take place on-line, and on the other hand an optimization algorithm on-line will require as much computational resources as the real time system itself.

Our idea in this general case is to use the ratio U/BU as an estimation of the average frequency. To show the reliability of this estimation, we analyze the behavior of Rate Monotonic scheduling using an average frequency reduction policy.

In Figure 2, we present the results of the average operating frequency for different Us and BUs. Symbols represent the empirical frequency we found via simulation (we simulate the execution of all tasks sets reducing progressively the processor operating frequency from the maximum to the minimum, to all tasks in each task set. We stop when one deadline is missed), while the lines represent the theoretical estimation of this frequency using the ratio U/BU.

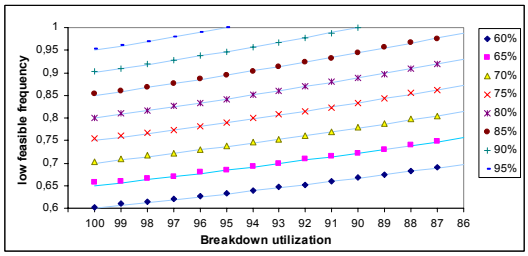


Figure 2: Lowest feasible frequency vs breakdown utilization for 4000 random task-sets with 8 independents task characterized by non-harmonics periods. Different symbols correspond to processor utilizations ranging from 60% to 95%. Lines represent the theoretical estimation based on the ration U/BU.

We also checked the effect of variability in number of periodic tasks, the harmonicity of the periods and the processor utilizations. In particular we have simulated a number of independent periodic tasks varying from 8 to 16 for both harmonic and non-harmonic periods. The results do not differ from those presented in Figure 2.

4. The Enhanced Power Low Dual Priority

The benefits in energy saving obtained in the Rate Monotonic Algorithm by applying the average operating frequency policy can be generalized to other fixed priority pre-emptive scheduling algorithms.

In particular, we are interested into extend these results to the Power Low Modified Dual Priority Scheduling algorithm (based on the Dual Priority scheduling [13]) because its adequacy to manage power saving in more

complex scenarios that could include aperiodic requests and because its performance has been contrasted against other fixed priority algorithms [11]. The original Power Low Dual Priority Scheduling algorithm (PLMDP) [12] guarantees to meet the temporal constraints and a significant energy consumption reduction.

Based on the results obtained for the RMA we propose the use of a balanced operating frequency reduction policy that gives to all ready tasks the opportunity to reduce its execution operating frequency. The balance is intended to provide an average operating frequency according to U/BU and it is controlled dynamically. Qualitatively the idea work as follows: If a task should execute at a certain operating frequency higher than the estimated average to meet its deadline, then the following tasks try to execute at an operating frequency lower than the estimated average to compensate the global effect in the system. Then our algorithm is designed to achieve an average operating frequency according to the ratio U/BU while meeting deadlines. BU is statically calculated off-line.

The PLMDP defines three levels of priorities that are organized as follows, the highest level, or upper run queue (URQ) is for tasks that can no longer be delayed by less priority tasks otherwise they cold miss their deadlines. The lowest level, or lower run queue (LRQ) occupied by those periodic tasks whose execution time can still be delayed without compromising their deadlines. At the beginning of each hyper-period the remaining processor utilization (W_{rem}) is set to the total workload of the task set = U.

The scheduling algorithm is driven by the following events:

1. Promotion time instant (Tp_i) [12] The moment at which the task is promoted from the LRQ to the URQ. At this moment the task can pre-empt a lower priority task currently in execution. At this time instant, W_{rem} is updated according to its real use, it is decremented by the consumed time of the pre-empted task.
2. Activation time (Ta_i). The task is queued in the LRQ sorted by its promotion time instant. At this moment this task can pre-empt a lower priority periodic task currently in execution, and W_{rem} is updated.
3. Task finalization time. At this time instant, W_{rem} is decremented by the consumed time plus the spare time of this task. After that, the highest priority task from the highest non-empty priority level (i.e. URQ or LRQ, in this order) is selected for execution.

In the new algorithm Enhanced Power Low Dual-Priority EPLDP, the processor operating frequency is individually calculated for each task, once the scheduler decides which task must be executed (Figure 3). The algorithm reduces the operating frequency at the maximum value between the frequency estimated by the original PLMDP, and the ratio between the processor utilization and the breakdown utilization: U_{rem}/BU. This operating frequency is the lowest frequency that assures that no deadline will be missed[1]. Before calculating the operating frequency U_{rem} is updated to the ratio between the workload that remains to be executed and the remaining time to arrive to the end of the hyperperiod.

$$U_{rem} = \left(\frac{W_{rem}}{hyperperiod - tc} \right) \quad (2)$$

Summarizing, the resulting algorithm (EPLDP) works as follows:

```
// Tp is the promoted time; Td is the deadline time; tc is the current time
// τi is the active task; τk is the next promoted task
// Urem is the remaining utilization = U – executed workload
L1  if not empty (URQ) then
L2      Active Task (τi)= URQ.head;
L3      if URQ.head.next = NIL
L4          Frequency = max( min(Tpk – tc, remaining_i) / min(Tpk, Tdi) – tc , Urem / BU )
L5      else
L6          Frequency = MAX_FREQ;
L7      endif
L8  else
L9      if not empty (LRQ) then
L10         Active task (τi)= LRQ.head;
L11         if Tpk < Tpi then
L12             Frequency = ( Urem / BU )
L13         else
L14             Frequency = max( min(Tpk – Tpi, remaining_i) / min(Tpk, Tdi) – tc , Urem / BU )
L15         endif
L16     else
L17         Set timer to (next Tai - wake up delay);
L18         Enter power-down mode;
L19     endif
L20     endif
L21 endif
L22 execute Active Task (τi) at calculated operating frequency;
```

Figure 3: **Enhanced Power Low Dual-Priority (EPLDP) Scheduling**.

1. If there is not any ready task in the system we set the timer to the next arriving task minus the wake up delay, and power down the processor.
2. If there is more than one task in the URQ then it must be executed at the maximum operating frequency.
3. If there is only one task in the URQ then it can be executed at low frequency:

$$Frequency = max\left(\frac{min(Tp_k - tc, remaining_i)}{min(Tp_k, Td_i) - tc}, \frac{U_{rem}}{BU} \right) \quad (3)$$

where Tp_k is the promotion time instant of any task in the system excluding the current executing task, $remaining_i$ is the non-executed worst execution time of the current task, Td_i is the deadline of the current

[1] The feasibility of our algorithm is achieved whenever the Dual Priority Algorithm is feasible [13] because our algorithm always uses the Dual Priority schedule as a lower bound for feasibility.

task, and finally *tc* is the current time. The desired operating frequency is based on the remaining workload, the remaining time to the hyper-period, and the BU.

4. If there is not any task in the URQ but there are some tasks in the LRQ then we can execute at the average operating frequency U_{rem}/BU.

At practice only certain discrete values of the frequency of the clock, and then speed, are attainable depending on the accuracy of the tuning, in this case the frequency selected should be a frequency equal or larger than the frequency obtained by the calculations to ensure time constraints.

The algorithm is designed to achieve an average operating frequency equal to the ratio U_{rem}/BU. This average is achieved whenever all tasks consume the 100% of its WCET. When the WCET is not totally consumed then this average is overestimated (in the next section we will discuss how to take advantage of this fact). The performance of the PLMDP is flexible to different WCET consumptions adapting is behavior when needed. At a certain critical value of the WCET consumption we expect PLMDP to overcome the performance of the EPLDP because this overestimation. In figure 4 we show the experimental critical curve for the WCET consumption delimiting the area of efficiency of both algorithms.

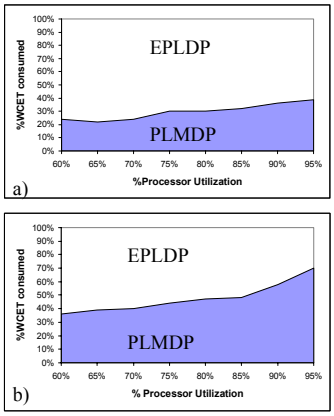


Figure 4: Critical line showing the transition performance between EPLDP and PLMDP. Above the line EPLDP is energetically favorable, below the line the PLMDP is energetically favorable. The line is obtained by simulation of 100 tasks sets (formed by 8 tasks each one) for each value of the processor utilization, varying the processor utilization in 5% each step. Harmonic periods from 1024 to 65536 are considered. The maximum task workload is set to 20%.

5. Moving average estimation of the empirical utilization of the processor

The WCET of a task depends on both the program flow and architectural factors like pipelines and caches. It must guarantee and not underestimate the real execution time, but often provides an overestimation of it. To reach maximum effectiveness of the use of the processor, the overestimation of the empirical execution time should be as small as possible. But note that as the processors have more complex features like for example out-of-order execution, the overestimation becomes usually large.

We want to point out that a dynamic estimation of the real utilization (EU) is possible by using the history of past executions (U_m) where m refers to different hyper-periods.

Here we present a moving average process that takes advantage of this information to determine the correct average frequency reduction that adapts to the real calculation consumption of tasks (U_i).

We modified our algorithm introducing a moving average of the utilization that reduces the overestimation of the WCET of tasks in the following manner (Figure 5):

1. Initially the estimated utilization (*EU*) is set to the total workload with the WCET provided by the application designers (U_0).
2. The hyper-period is executed using the current *EU*. At the same time the real workload of each task is updated after its real execution.
3. At the end of the hyper-period the *EU* is updated according to the available history ($\bar{U}$) and the recent hyper-period execution (U_i), using a moving average.

L1 $U_0 = \sum_{\forall \tau} C_i \qquad EU = U_0$

L2 i=1;

L3 while real_time_application_not_finished do

L3 execute hyper-period i with *EU* and update U_i;

L4 $\bar{U} = \frac{(\bar{U} * i) + U_i}{i+1} \qquad EU = U_i - \bar{U}$

L5 enddo

Figure 5: Moving average estimation of the empirical utilization of the processor.

In Figure 6 we present the evolution of the EPLDP using this estimation over different hyper-periods (EPLDP-m). The maximum utilization of the system is set to 80 % with a maximum workload for tasks of 20%. All tasks consumption is obtained from a Gaussian distribution with an average of 50% of its WCET and with a standard deviation of 10%. The results are obtained averaging over 100 different task sets. Note that a lower bound for the frequency reduction is provided by this estimation while the low power algorithm fixes the upper bound (Figure 3). The energy consumption obtained by EPLDP-m tends to be the same as the energy consumption obtained by the theoretical EPLDP-f, which is the EPLDP behavior assuming that the real utilization is known and fixed to 50% of the WCET (note that this information is unknown in real applications). The small divergence between EPLDP-m and EPLDP-f are consequence of the variations of the real use of the WCET that has been set to 10%.

It is important to note that this moving average strategy does not interfere with the hard real time because the

determination of the operating frequency is conservative with respect to deadlines i.e. we use the highest frequency between the calculated EU and the operating frequency calculated by the PLMDP algorithm. (see equation 3), to do not compromise any deadline.

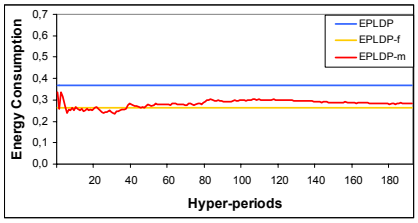


Figure 6: Evolution of EPLDP-m compared with EPLDP and with EPLDP-f.

The improvement in energy consumption provided by EPLDP-m is contrastable, and it should be more evident as the overestimation of the WCET is larger. In figure 7, we show this improvement as a function of the percentage of the WCET really consumed for a harmonic task sets (Figure 7a), and for a non-harmonic task sets (Figure 7b).

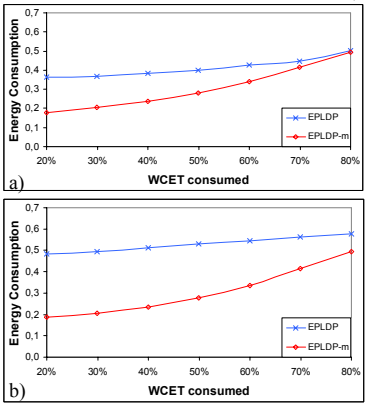


Figure 7: Performance of EPLDP and EPLDP-m with different percentage of WCET consumption. The total utilization of the system is U=80% and the maximum task workload is set to 20. The results are obtained averaging over 100 task sets. In a) the task sets are harmonic, and in b) there are non-harmonic task sets.

6. Results and discussion

In this section we test the energy saving efficiency of the proposed EPLDP-m algorithm versus the original Power Low Modified Dual Priority (PLMDP) [3,12] for real and synthetic task sets.

The first test (Figure 8) corresponds to the energy consumption for different fixed workloads of the system (U=60%, 75% and 95%) (Figure 8). We observe that the average energy consumption is energetically favorable to EPLDP. The average energy consumption is 59%, 61% and 68% for PLMDP and 17%, 28% and 49% for EPLDP-m (U=60%, 75% and 95% respectively) compared to the execution at maximum operating frequency.

We have also checked the dependence of energy consumption on the workload of the system (U). We calculate the average energy consumption of schedulable tasks sets composed by 100 synthetic task sets. The workloads range from 60% to 95%, in steps of 5%. The maximum task workload was fixed to 20%. There are 8 tasks in each task set. The periods range from 1024 to 65536 (harmonic task sets). (Figure 9). The experiment represents the results of the normalized average energy obtained with respect to the execution at maximum operating frequency. We run the simulation over 200 hyper-periods.

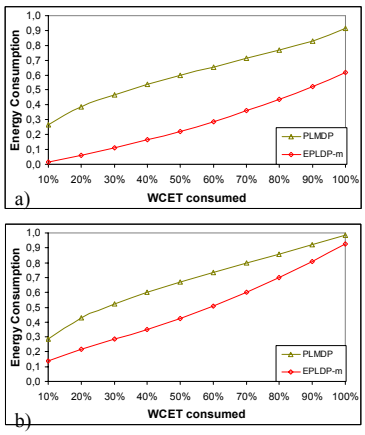


Figure 8: Normalized average energy consumption for different values of WCET consumption. We simulate 100 task sets, of 8 tasks each one, with a maximum task workload of 20%, in a) U=75%, and in b) U=95%.

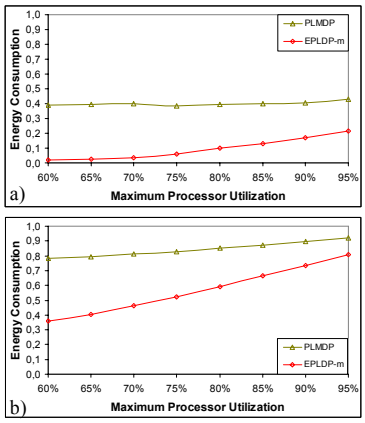


Figure 9 Normalized energy consumption versus processor utilization (U). Each dot corresponds to the average energy consumption of 100 different harmonic task sets. The consumed WCET is in a) 20% and in b) 90% .

To conclude the present analysis, we have also collected some real time applications: the avionics task set reported in [17], an Inertial Navigation System (INS) [18] and a Computerized Numerical Control Machine (CNC) [19]. The two first sets represent critical mission applications and the last one is an automatic control for specific machinery.

The results of energy consumption for each application varying the percentage of WCET consumed are drawn in

Figures 10 to 12. The average energy consumption referred to the execution at maximum frequency are 76% for PLMDP and 35% for the EPLDP-m in the avionics task set; 36% for PLMDP and 9% for the EPLDP-m in the INS task set; and 56% for PLMDP and 26% for the EPLDP-m in the CNC task set.

We have also performed simulations considering variations of the task sets specifications: doubling the number of tasks to 16 instead of 8, varying the maximum workload per task from 20% to 10%, and considering non-harmonic periods ranging from 1000 to 70000. The results obtained in these different experiments do not differ qualitatively from the results presented, although the precise values vary. In particular, non-harmonic periods introduce a shift on the energy consumption performance of all the algorithms we have studied. The main reason is that schedulability becomes more complex, and the breakdown utilization decreases.

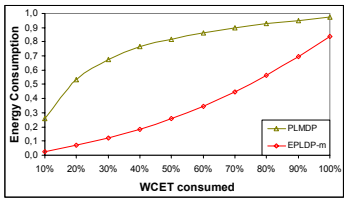


Figure 10:Comparison of the algorithms for the Avionics set [17]

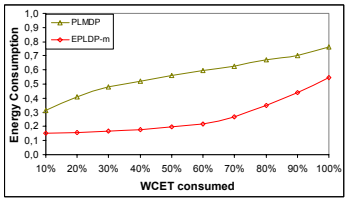


Figure 11: Comparison of the algorithms for the INS set [18]

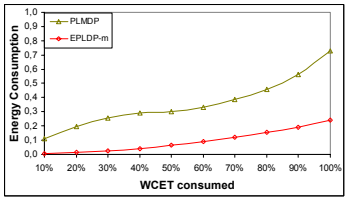


Figure 12:Comparison of the algorithms for the CNC set [19]

7. Conclusions

We have proposed a new version of the PLMDP algorithm that enhances energy saving based on the dynamic calculation of an average operating frequency EPLDP-m. The advantage of this energy reduction policy, consisting on giving the opportunity to the processor to reduce the operating frequency of every task, has been demonstrated to improve energy saving substantially without missing any deadline. The current performance could be extended to other dynamic power aware scheduling algorithms.

8. References

[1] A.P. Chandrakasan, S. Sheng and R. W. Brodersen, "Low-power CMOS digital design", *IEEE Journal of Solid-State circuits*, vol. 27, pp. 473-484, April 1992.

[2] C. M. Krishna, Y.H. Lee, "Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems", in *Real-Time Technology and Applications Symposium*, pp. 156-165, 2000.

[3] M.A. Moncusí, A. Arenas and J. Labarta, "A modified dual priority scheduling in hard real time systems to improve energy saving." in *Compilers and Operating systems for Low Power* pp 17-36. *Kluwer academic/Plenum publishers* 2003.

[4] S. Saewong and R. Rajkumar, "Practical voltage-scaling for fixed-priority RT-systems." *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp106-115, May 2003

[5] U. Kimiyoshi, M. Horowitz, "Clustered voltage scaling technique for low-power design", *in International Symposium on Low Power Electronics and Design*, pp. 3-8, 1995.

[6] J. Rabaey and M Pedram (Editors). "Low power design methodologies". *Kluwer Academic Publishers*, May 1996.

[7] S.T. Cheng, S.M. Chen and J.W. Hwang, "Low-power design for real-time systems", *Real-Time Systems*, 15, pp 131-148, 1998.

[8] P. Pillai and K.G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems" *18th ACM Symposium on Operating Systems Principles*, October 2001.

[9] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics" *13th Euromicro Conference on Real-Time Systems*, June 2001.

[10] H. Aydin, R. Melham, D. Mosse, P. Mejia-Alvarez. "Dynamic and Aggressive scheduling techniques for power-aware real-time systems." *Proc. Real-Time Systems Symposium*, pp. 95-105, 2001.

[11] Y. Shin and K. Choi, "Power conscious fixed priority scheduling in hard real-time systems" *DAC 99, ACM* 1-58113-7/99/06, 1999.

[12] M.A.Moncusi, A.Arenas, J.Labarta, "Improving energy saving in hard real time systems via a modified Dual Priority scheduling", *ACM SigArch Computer Architectur*e Newsletter, Vol 29, 19-24 (2001)

[13] R. Davis and A.J. Wellings, "Dual Priority scheduling", *Proceeding IEEE Real Time Systems Symposium*, pp. 100-109, December 1995.

[14] A. Miyoshi, C.Lefurgy, E.V. Hensbergen, R.Rajamony, and R. Rajkumar. "Critical power slope: Understanding the runtime effects of frequency scaling." *In Proceedings of the 16th Annual ACM International Conference on supercomputing*, June 2002.

[15] J. Lehoczky, L. Sha, and Y. Ding. "The rate monotonic scheduling algorithm: Exact characterization and average case behavior". In *Proceedings of IEEE Real-Time Systems Symposium*, pages 166–171. IEEE Computer Society Press, December 1989.

[16] Katcher, D.I.; Arakawa, H.; Strosnider, J.K.; "Engineering and analysis of fixed priority schedulers" *Software Engineering, IEEE Transactions on*, Volume: 19, Issue: 9, Sept. 1993 Pages: 920–934

[17] C. Locke, D. Vogel and T. Mesler, "Building a predictable avionics platform in Ada: a case study", *Proceedings IEEE Real-Time Systems symposium*, December 1991.

[18] A. Burns, K. Tindell and A. Wellings, "Effective analysis for engineering real-time fixed priority schedulers", *IEEE Transactions on Software Engineering*, 21, pp. 475-480, 1995.

[19] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi and H. Shin, "Visual assessment of a real-time system design: a case study on a CNC controller", *Proceedings IEEE Real-Time Systems symposium*, December 1996.

Optimal Speed Assignment for Probabilistic Execution Times

Claudio Scordino
University of Pisa, Italy
scordino@di.unipi.it

Enrico Bini
Scuola Superiore Sant'Anna, Italy
e.bini@sssup.it

Abstract

The problem of reducing energy consumption is dominating the design and the implementation of embedded real-time systems. For this reason, a new generation of processors allow to vary the voltage and the operating frequency to balance computational speed versus energy consumption. The policies that can exploit this feature are called Dynamic Voltage Scheduling (DVS).

In real-time systems, the DVS technique must also provide the worst-case computational requirement. However, it is well known that the probability of a task executing for the longest possible time is very low. Hence, DVS policies can exploit probabilistic information about the execution times of tasks to reduce the energy consumed by the processor.

In this paper we provide the foundations to integrate probabilistic timing analysis with energy minimization techniques, starting from the simple case of one task.

1 Introduction

The number of embedded systems operated by batteries is increasing in different application domains, from PDAs (*Personal Digital Assistants*) to autonomous robots, smart phones and sensor networks. Reducing the energy consumed by these systems has become a key design issue, as they can only operate on the limited battery supply. For this reason, a new generation of processors [9, 13, 19] allow to dynamically vary the voltage and the operating frequency to balance computational speed versus energy consumption.

In recent years, as the demand for computing resources has rapidly increased, even normal workstation PCs and servers face energy constraints. Not surprisingly, a significant portion of the consumed energy is due to the cooling devices, which may consume up to the 50% of the total energy [11]. In addition, researchers at IBM showed that average processor use of servers is between 10% and 50% of their peak capacity because the load depends on the time of the day or the day of the week [4]. This suggests that a striking energy reduction can be achieved by enriching DVS policies with a more detailed information on the required workload.

Recently, many DVS algorithms have been proposed in the literature. These algorithms can be divided in two classes: static and dynamic. Static techniques [21, 14, 17, 12, 3] are typically applied to periodic tasks, and make use of off-line parameters, such as periods and worst-case execution cycles (WCECs), to select the appropriate processor speed. Since the worst-case parameters may differ significantly from the actual values, these techniques save less energy than the dynamic ones.

On the other hand, much recent research has focused on dynamic techniques [14, 1, 22, 17, 16, 18], which take advantage of early job completion. Some studies have observed that the actual execution cycles of real-world embedded tasks may vary up to 80% with respect to their measured WCECs [20]. Thus, dynamic methods can exploit information about the run-time behaviour of tasks, which may be very far from the pessimistic assumptions required during the design of static techniques. Dynamic algorithms may take decisions — change the processor speed — at two different instants:

After task completion The algorithm does not make any assumption on task duration, and waits for task completion to know the exact execution time. Then, the processor speed is changed based on this information. The algorithms GRUB-PA [18], DVSST [16] and RTDVS-Cycle Conserving [14] belong to this category.

Before task completion The algorithm tries to foresee the duration of the current task instance, and takes the decision in advance, based on some task's characteristics such as the average execution time. This decision typically depends on the behaviour of previous instances of the task. Obviously, a right prediction allows to reduce considerably the energy consumption. However, when the predicted behaviour is distant from the reality, some undesired side effect, such as a deadline miss in soft real-time systems or an increase of the energy consumed, may occur. The algorithms RTDVS-Look Ahead [14], DVS-EDF [22] and DRA-Aggressive [1] are based on this mechanism.

The success of the second class of methods relies on predicting correctly the task behaviour. For this reason, the use

of a richer task information may enhance the effectiveness of the DVS. This increased information can be provided by the probability density function (p.d.f.) of the task execution time. Recently, the discipline of probabilistic timing analysis has significantly advanced [5, 7], and today there exist some tools which can provide the p.d.f. of task execution times [2].

An attempt to consider stochastic information in energy reduction problems has been done by Gruian [8]. However, it only addresses the case with no transition overheads and with a specific power function.

In this paper we integrate the concept of probabilistic execution time within the framework of energy minimization, providing the basis of a new challenging approach. We preliminary consider the case of only one task, since we believe that it can be extended to the general case of n tasks.

2 System model

2.1 Processor model

We assume that the processor has a continuous spectrum of operating modes. This means that the speed can continuously vary between zero and some speed upper bound. We know that in real-world architectures this hypothesis does not ever hold. However, many significant contributions in the literature [1, 14, 22] still assume a continuous speed because if the processor speed levels are very close each other then this approximation is very close to reality. Obviously, if the optimal speed is not available, it has to be approximated with the closest discrete level higher than the optimal one. In this case, there is an increase of energy consumption, called *energy quantization error*, that has been studied by Saewong and Rajkumar [17].

The processor model is formalized as follows:

- the speed α can vary within $[0, \alpha_{\max}]$, where $\alpha_{\max}$ is the maximum speed allowed by the processor;
- the power consumption at speed α is modelled by the function $p(\alpha)$. Typically, the power function $p(\alpha)$ is a polynomial [6]. However, due to the advances of semiconductor technology, it is expected that the power/speed relationship may change in the next future [8]. For this reason we model this relationship by a generic function $p(\alpha)$;
- the cost of mode switching is considered in terms of both time and energy. We call o the time overhead needed to switch between any two modes, and e the energy required. Notice that o and e do not depend on the modes before and after the switch.

2.2 Energy management scheme

We focus on the problem of reducing the energy consumed by a task τ on a variable speed processor. Some existing power-aware algorithms have been deployed starting from this simple scheme, since it constitutes a good starting point for more complex analysis [22].

The task τ has period and deadline both equal to T. The number of processor cycles required in the interval $[0, T]$ is modelled by a random variable whose p.d.f. is $f_C(c)$. The maximum possible number of cycles needed by τ is $C_{\max}$. Since the task is hard real-time, $C_{\max}$ cycles must be available in $[0, T]$.

If the number of required cycles in $[0, T]$ was known in advance, the best way to reduce energy consumption would be to keep a constant speed [10, 15]. In fact, the convexity of the power/speed curve implies that maintaining a constant speed α is better than switching between two different speeds. Unfortunately, this number of cycles is unknown in advance, hence we cannot determine the optimal speed α.

A common technique adopted in the literature [1, 22, 14] is based upon the idea of deferring some work, since we expect that the current instance of τ will request much less than its WCEC $C_{\max}$. This technique splits the task execution into two parts, as shown in Figure 1. In the first part, the task runs at a lower processor speed α_L in order to reduce the energy consumed in the average case. In the second part, instead, a higher processor speed α_H is used, so that we can provide up to $C_{\max}$ cycles even in the worst case. The idea is that, if a task tends to use much less than its WCEC, the second part, which consumes more power, may never be needed. When the worst-case condition occurs, instead, the speed increase guarantees the completion of all the deferred work within $[0, T]$.

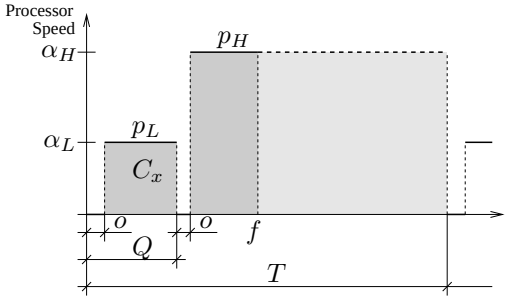


Figure 1. The energy management scheme.

The idea of deferring work has been widely used in the literature to create efficient power-aware algorithms. For instance, Pillai and Shin [14] proposed the RTDVS-Look Ahead algorithm, which tries to defer as much work as possible, and sets the operating frequency to the minimum value to ensure that all future deadlines will be met. This technique has also been used by Aydin et al. in the "aggressive" version of their DRA algorithm [1]. This algorithm speculatively assumes that current and future instances of the task will most probably present a computational demand lower than the worst case. Hence, it tries to reduce the speed of the running task by deferring all the work above a certain threshold, set according to the average workload. A similar

approach has been applied to EDF by Zhu and Mueller [22]. Each task's instance is divided into two portions. The objective is to provide the average number of cycles C_{avg} within the first portion. The second part at speed α_H ensures that the deadline is met even when the task requires C_{max} cycles.

Even if these techniques have been widely used in the literature, a probabilistic study of this model has not been proposed, yet. For instance, all previous algorithms set the speed α_H equal to the maximum possible value, even if this may not be optimal from the point of view of energy consumption. Even worse, some technique [22] is based on the intuitive idea that the optimal energy reduction is obtained by providing exactly the average execution cycles in the first part. In Section 3.1 we will prove that this intuition is not correct.

We decide to deeply study this model, extending it to the case in which probabilistic information about task execution time is known. Moreover, we use a general model for the processor, accounting for both the time and energy overheads of voltage transition.

3 Optimal speed assignment

Let α_L and α_H be the lower and the higher processor speeds, respectively. The period of the scheme is T. The number of processor cycles required by the task τ in each period is modelled by a random variable whose p.d.f. is $f_C(c)$, and the maximum number of cycles is C_{max}. This amount of cycles must be guaranteed in each period because the task is subject to a hard real-time constraint.

Our goal is to find the two speed levels α_L and α_H and the instant Q when to switch, in order to achieve the minimum energy consumption. Let C_x be the number of cycles provided while running at α_L, as shown in Figure 1. We can express (α_L, α_H) as a function of C_x and Q as follows

$$\alpha_L = \frac{C_x}{Q-o} \qquad \alpha_H = \frac{C_{\text{max}} - C_x}{T-Q-o}, \tag{1}$$

due to the constraint of providing C_{max} cycles within each period T.

Let also be c the number of cycles that actually occur and f the finishing time . We distinguish two different cases:

1. if $c \leq C_x$ then the task terminates before we could actually switch to α_H, and we expect $f \leq Q$;
2. otherwise, if $c > C_x$ then we need to run at speed α_H to provide the required cycles and we expect $f > Q + o$.

We consider the two cases separately. In order to have a more compact notation we set $p_H = p(\alpha_H)$ and $p_L = p(\alpha_L)$.

In the first case ($c \leq C_x$), the finishing time is

$$f = o + \frac{c}{\alpha_L}$$

and the energy consumed in one period T is

$$E = e + p_L\,(f - o) = e + \frac{p_L}{\alpha_L}\,c. \tag{2}$$

On the other hand, when $C_x < c \leq C_{\text{max}}$, we have

$$f = Q + o + \frac{c - C_x}{\alpha_H}$$

and the energy is

$$E = 2\,e + \frac{p_L}{\alpha_L}\,C_x + \frac{p_H}{\alpha_H}\,(c - C_x). \tag{3}$$

Equations (2) and (3) provide the energy E consumed when the number of cycles is c. Since the number of cycles is a random variable with p.d.f. $f_C(c)$, then the energy consumed is a random variable too. Our target then becomes to minimize the expectation E_{avg} of the random variable E. Let us compute this value.

$$\begin{aligned}
E_{\text{avg}} &= \int_0^{C_x} E\,f_C(c)\,\mathrm{d}c + \int_{C_x}^{C_{\text{max}}} E\,f_C(c)\,\mathrm{d}c \\
&= \int_0^{C_x} \left(e + \frac{p_L}{\alpha_L}\,c\right) f_C(c)\,\mathrm{d}c \\
&\quad + \int_{C_x}^{C_{\text{max}}} \left(2\,e + \frac{p_L}{\alpha_L}C_x + \frac{p_H}{\alpha_H}\,(c - C_x)\right) f_C(c)\,\mathrm{d}c \\
&= 2\,e + \frac{p_H}{\alpha_H}\,C_{\text{avg}} - \left(\frac{p_H}{\alpha_H} - \frac{p_L}{\alpha_L}\right) C_x \\
&\quad + \int_0^{C_x} \left(\left(\frac{p_L}{\alpha_L} - \frac{p_H}{\alpha_H}\right)(c - C_x) - e\right) f_C(c)\,\mathrm{d}c \\
&= e\,(2 - F_C(C_x)) + \frac{p_H}{\alpha_H}\,C_{\text{avg}} \\
&\quad - \left(\frac{p_H}{\alpha_H} - \frac{p_L}{\alpha_L}\right)(G_C(C_x) + C_x(1 - F_C(C_x)))
\end{aligned}$$

where

$$F_C(x) = \int_0^x f_C(c)\,\mathrm{d}c \qquad G_C(x) = \int_0^x c\,f_C(c)\,\mathrm{d}c.$$

For compactness we also set

$$\gamma(x) = G_C(x) + x(1 - F_C(x)), \tag{4}$$

so that the average energy consumed in a period is

$$E_{\text{avg}} = e\,(2 - F_C(C_x)) + \frac{p_H}{\alpha_H}(C_{\text{avg}} - \gamma(C_x)) + \frac{p_L}{\alpha_L}\,\gamma(C_x) \tag{5}$$

Notice that $G_C(C_{\text{max}})$ is equal to C_{avg}. For this reason we always have $0 \leq \gamma(x) \leq C_{\text{avg}}$ for all x.

Equation (5) is a new result in the literature because it expresses the average energy consumed as function of the probability density of the task execution cycles $f_C(c)$.

It is very insightful to plot the quantity E_{avg} on a plane (C_x, Q). Figure 2 shows the level curves of the quantity E_{avg} as function of C_x and of Q. In the plot we assumed an exponential p.d.f. with average value $C_{\text{avg}} = 0.2929$, the period T equal to 1 and the power function $p(\alpha) = k\,\alpha^3$.

As you can notice, the minimum at the center of the white

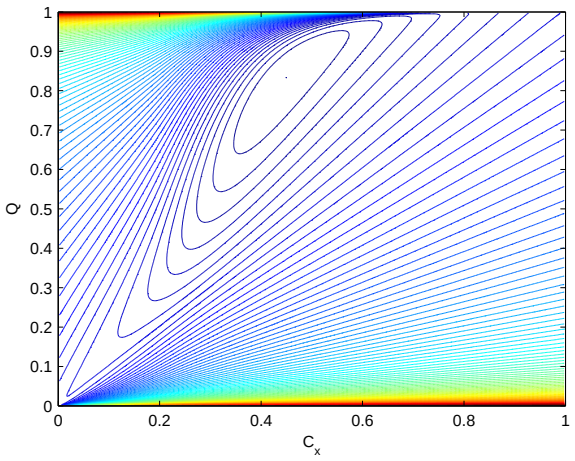


Figure 2. E_{avg} **for uniform execution times.**

region occurs for a value of C_x greater than C_{avg}. In order to find it analytically, we need to compute the partial derivative of E_{avg} with respect to the variables (C_x, Q). After opportune simplifications, we find that:

$$\frac{\partial E_{\text{avg}}}{\partial C_x} = -e\, f_C(C_x) - \left(p'_H - \frac{p_H}{\alpha_H}\right)\frac{C_{\text{avg}} - \gamma(C_x)}{C_{\max} - C_x} + \left(p'_L - \frac{p_L}{\alpha_L}\right)\frac{\gamma(C_x)}{C_x} - \left(\frac{p_H}{\alpha_H} - \frac{p_L}{\alpha_L}\right)\gamma'(C_x) \quad (6)$$

where p'_L and p'_H denote $p'(\alpha_L)$ and $p'(\alpha_H)$, respectively.

Now, we complete the analysis of the function E_{avg} by computing also $\frac{\partial E_{\text{avg}}}{\partial Q}$, which is

$$\frac{\partial E_{\text{avg}}}{\partial Q} = (p'_H \alpha_H - p_H)\frac{C_{\text{avg}} - \gamma(C_x)}{C_{\max} - C_x} - (p'_L \alpha_L - p_L)\frac{\gamma(C_x)}{C_x} \quad (7)$$

Equations (6) and (7) are the components of the gradient ∇E_{avg}. From functional analysis, we know that the minimum satisfies the condition $\nabla E_{\text{avg}} = 0$. Once the optimal (C_x, Q) is found, then the constraint $\alpha_H \leq \alpha_{\max}$ must be checked. In fact, if it is violated, it means that the global minimum would result in a too high value of α_H. It this case we know from the Kuhn-Tucker conditions that the minimum occurs when $\alpha_H = \alpha_{\max}$, which means that

$$\frac{C_{\max} - C_x}{T - Q - o} = \alpha_{\max} \;\Rightarrow\; \alpha_L = \frac{C_x\, \alpha_{\max}}{\alpha_{\max}(T - 2o) - C_{\max} + C_x} \quad (8)$$

From Eq. (5), substituting α_H with $\alpha_{\max}$ and α_L with the expression of Equation (8), we find E_{avg} as function of the unique variable C_x. The minimal energy solution is found by applying classical techniques of functional analysis of one-variable functions.

3.1 Polynomial power function

Once the main equations for the general case have been found, we show how they can be applied to find the optimal (C_x, Q) in some significant examples. Due to lack of space, we assume the time and energy overheads equal to zero (i.e. $o = 0$ and $e = 0$).

When considering continuous speed levels, a common assumption is that the relationship between the power consumption p and speed α is

$$p(\alpha) = k\, \alpha^n \quad (10)$$

for some k, n. The typical value of n is 3, however we keep the general form as long as the math is tractable.

In these hypothesis, the gradient can be greatly simplified. In order to find the point of minimal energy we have to set both the gradient components equal to zero. By setting $\nabla E_{\text{avg}} = 0$, we finally find that the pair (C_x, Q) minimizing the average energy E_{avg} must satisfy Equations (9) reported in Table 1. Due to lack of space we don't include all the calculations. For their importance we call the Equations (9) the **minimum stochastic energy equations**. Once we know n and the probability density $f_C(c)$, Equations (9) can be solved and produce the pair (C_x, Q) which minimizes the energy.

Uniform Density Let us now assume a uniform density between $C_{\min}$ and $C_{\max}$. It means that

$$f_C(c) = \begin{cases} \frac{1}{C_{\max} - C_{\min}} & \text{if } C_{\min} \leq c \leq C_{\max} \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

and also, when $C_{\min} \leq c \leq C_{\max}$,

$$F_C(c) = \frac{c - C_{\min}}{C_{\max} - C_{\min}} \qquad G_C(c) = \frac{c^2 - {C_{\min}}^2}{2(C_{\max} - C_{\min})}. \quad (12)$$

The function $\gamma(c)$, defined in Eq. (4), is

$$\gamma(c) = \frac{-c^2 + 2c\, C_{\max} - {C_{\min}}^2}{2(C_{\max} - C_{\min})} \quad (13)$$

and its derivative is

$$\gamma'(c) = \frac{C_{\max} - c}{C_{\max} - C_{\min}} \quad (14)$$

In this case the minimum energy can be simply found by properly substituting $\gamma(C_x)$ and $\gamma'(C_x)$ in the minimum stochastic energy equations (9).

To simplify and compact them, it is very convenient to normalize the cycles C_x and $C_{\min}$ with respect to $C_{\max}$. Hence, we set $x = \frac{C_x}{C_{\max}}$ and $a = \frac{C_{\min}}{C_{\max}}$. Due to lack of space here we do not report all the simplifications, which can be accomplished by any symbolic manipulation tool.

When $n = 2$ the optimal number of normalized cycles x, which provides the minimal energy, is

$$x_{\text{opt}} = \frac{1 + \sqrt{1 + 3\, a^2}}{3}. \quad (15)$$

$$\begin{cases} \dfrac{(n-1)\,\gamma(C_x) + C_x\gamma'(C_x)}{(n-1)\,(C_{\text{avg}} - \gamma(C_x)) + (C_{\text{max}} - C_x)\gamma'(C_x)} \left(\dfrac{C_{\text{max}}}{C_x} - 1\right)\left(\dfrac{C_{\text{avg}}}{\gamma(C_x)} - 1\right) = \dfrac{T}{Q} - 1 \\ \left(\dfrac{C_{\text{max}}}{C_x} - 1\right)^{n-1}\left(\dfrac{C_{\text{avg}}}{\gamma(C_x)} - 1\right) = \left(\dfrac{T}{Q} - 1\right)^{n} \end{cases} \tag{9}$$

Table 1. Minimum stochastic energy equations.

Instead, when $n = 3$, the solution is

$$x_{\text{opt}} = \frac{5 - \sqrt{5} + \sqrt{2}\sqrt{5(3-\sqrt{5}) - 8(1-\sqrt{5})\,a^2}}{8} \tag{16}$$

Interestingly, this result proves that the approach proposed by Zhu and Mueller [22] is sub-optimal, as stated by themselves. In fact, they suggested to set C_x equal to C_{avg}. From both Equations (15) and (16) we see that **the optimal value is always greater than C_{avg}** (see also Figure 3). Providing C_{avg} cycles at speed α_L would lead to increase

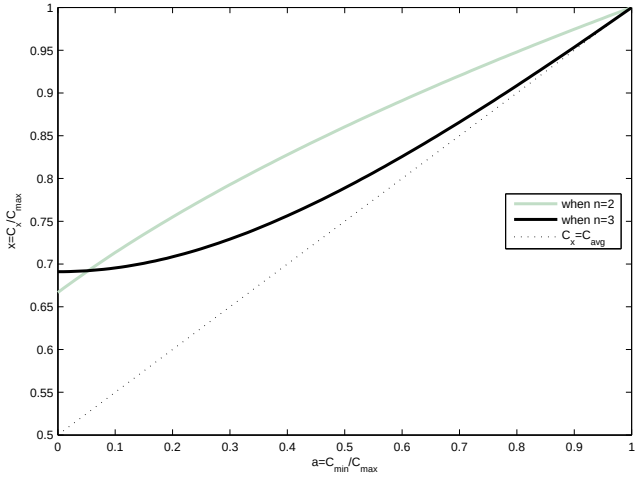


Figure 3. The optimal number of cycles.

the average energy consumed in a period.

Exponential Density The probability density considered previously is very simple and it allows to exactly find the pair (C_x, Q) which minimizes the average energy consumed. We consider now a more complex density $f_C(c)$ which better captures the characteristics of real execution times. Without loss of generality, we normalize the number of cycles toward C_{max} so that the possible values of cycles are in $[0, 1]$. As done before we set $a = \frac{C_{\text{min}}}{C_{\text{max}}}$.

We consider the following exponential p.d.f.:

$$f_C(c) = \begin{cases} \frac{1}{K} e^{\beta c}(1-c)(c-a) & \text{if } c \in [a, 1] \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where K is a proper constant such that $\int_a^1 f_C(c)\mathrm{d}c = 1$.

The presence of β allows to alter the symmetry of the density. In fact, for negative β the density shifts to the left, meaning that values close to C_{min} are more likely to happen. On the other hand, positive values of β means that execution cycles close to C_{max} occur more frequently. Figure 4 shows some possible functions.

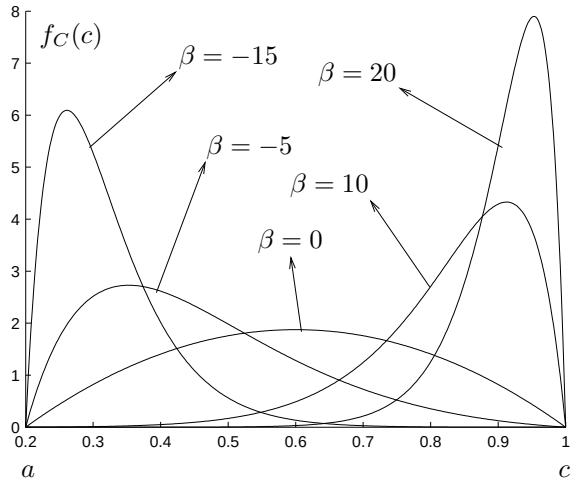


Figure 4. Exponential probability density functions.

For exponential densities, the minimal energy (C_x, Q) pair can only be found by numerical approximation. We investigated the effect of the p.d.f. asymmetry onto the solution. The result is quite interesting. In Figure 5 we plot the ratios $\frac{C_x}{C_{\text{avg}}}$ and $\frac{Q}{T}$, assuming $a = \frac{C_{\text{min}}}{C_{\text{max}}} = 0.2$. A first result, also noticed for uniform density, is that **the optimal C_x is always greater than C_{avg}**, differently than what suggested in a previous paper [22]. This fact is evidenced by the black curve which is always above 1. We also highlight that for big positive values of β (meaning that values close to C_{max} are more likely to occur), C_x tends to C_{avg}.

4 Conclusions and future work

Deferring the work is an effective technique already proposed in the literature to reduce the energy consumed by the processor. However, often this technique has been blindly applied, without a a systemic search of the minimal.

In this paper we have provided the foundations to integrate the probabilistic timing analysis with energy minimization techniques, starting from the simple case of one task. This problem has been studied using a general model

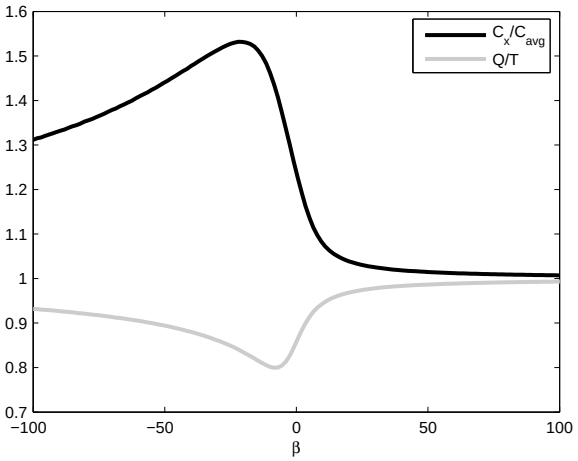


Figure 5. The optimal (C_x, Q) pair as function of the symmetry.

for the processor, taking into account both time and energy overheads. Thanks to this research, the design of effective energy minimization schemes using information about probabilistic execution times is now possible.

Finally, we refuted the idea that providing the average number of cycles at the lower speed is the best possible strategy.

References

[1] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, May 2004.

[2] G. Bernat, A. Colin, and S. M. Petters. pWCET: A tool for probabilistic worst-case execution time analysis of real-time systems. Ycs-2003-353, Computer Science dept., University of York, Feb. 2003.

[3] E. Bini, G. C. Buttazzo, and G. Lipari. Speed modulation in energy-aware real-time systems. In *Proc. of ECRTS*, Palma de Mallorca, Spain, July 2005.

[4] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. *The case for power management in web servers*. Kluwer Academic Publishers, 2002.

[5] A. Burns, G. Bernat, and I. Broster. A probabilistic framework for schedulability analysis. In *Proc. of EMSOFT*, Philadelphia, PA, Oct. 2003.

[6] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[7] A. Ermedahl, F. Stappert, and J. Engblom. Clustered calculation of worst-case execution times. In *Proc. of CASES*, San José, CA, Oct. 2003.

[8] F. Gruian. *Energy-Centric Scheduling for Real-Time Systems*. PhD thesis, Computer Science dept., Lund Institute of Technology, Lund, Sweden, Nov. 2002.

[9] Intel, `http://www.intel.com/design/intelxscale/`. *Intel XScale Technology*.

[10] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of the International Symposium on Low Power Electronics and Design*, Monterey, CA, Aug. 1998.

[11] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *IEEE Computer*, Dec. 2003.

[12] Y. Liu and A. K. Mok. An integrated approach for applying dynamic voltage scaling to hard real-time systems. In *Proc. of RTAS*, Washington DC, May 2003.

[13] Motorola, `http://e-www.motorola.com/webapp/sps/library/prod_lib.jsp`. *MPC5200: 32 bit Embedded Processor*.

[14] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of SOSP*, Banff, Canada, Oct. 2001.

[15] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proc. of MOBICOM*, Rome, Italy, July 2001.

[16] A. Qadi, S. Goddard, and S. Farritor. A dynamic voltage scaling algorithm for sporadic tasks. In *Proc. of RTSS*, Cancun, Mexico, 2003.

[17] S. Saewong and R. Rajkumar. Pratical voltage-scaling for fixed-priority RT-systems. In *Proc. of RTAS*, Washington, DC, May 2003.

[18] C. Scordino and G. Lipari. Using resource reservation techniques for power-aware scheduling. In *Proc. of EMSOFT*, Pisa, Italy, Sept. 2004.

[19] Transmeta, `http://www.transmeta.com/crusoe/`. *The Crusoe Processor*.

[20] J. Wegener and F. Mueller. A comparison of static analysis and evolutionary testing for the verification of timing constraints. In *Real-Time Systems*, Nov. 2001.

[21] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the* 36^{th} *Annual Symposium on Foundations of Computer Science*, Milwaukee, WI, Oct. 1995.

[22] Y. Zhu and F. Mueller. Feedback EDF scheduling exploiting dynamic voltage scaling. In *Proc. of RTAS*, Toronto, Canada, May 2004.

Integrated Device Scheduling and Processor Voltage Scaling for System-wide Energy Conservation

Hui Cheng and Steve Goddard
Department of Computer Science and Engineering
University of Nebraska — Lincoln
Lincoln, NE 68588-0115
{hcheng, goddard}@cse.unl.edu

Abstract

The challenge in conserving energy in embedded real-time systems is to reduce power consumption while preserving temporal correctness. Previous research has focused on power conservation for either the processor or I/O devices alone. The system-wide energy conservation has received little attention. In this paper, we analyze the problem of system-wide energy-efficient scheduling for hard real-time systems based on the preemptive periodic task model with non-preemptive shared resources. We propose an online system-wide energy-efficient scheduling algorithm System-wide Energy-Aware EDF (SYS-EDF), which integrates Dynamic Power Management (DPM) for I/O devices and Dynamic Voltage Scaling (DVS) for the processor. An evaluation of SYS-EDF shows that it yields significant energy savings with respect to DVS alone or DPM alone techniques.

1 Introduction

Embedded real-time systems often consist of a battery-operated microprocessor system with Input/Output (I/O) devices and a limited battery life. Energy conservation techniques are thus needed to extend their lifetimes. The need to prolong system lifetime has resulted in much work done in energy-efficient task scheduling for real-time systems.

In the last decade, much work has been done on processor-based power management techniques. Dynamic Voltage Scaling (DVS) is one of the most popular techniques to reduce the processor energy consumption. DVS-based real-time scheduling algorithms can effectively reduce the processor energy consumption by lowering the processor speed, while still guarantee that all jobs meet their deadlines. However, DVS-based algorithms reduce the dynamic power consumption of the processor at the cost of increased execution time, which in turn increases the I/O device standby energy consumption. It has been observed [4, 11] that aggressively lowering the processor speed may increase the overall system energy consumption rather than decreasing it.

The energy consumption of I/O devices can be reduced by shutting down devices under certain conditions. This method is commonly known as Dynamic Power Management (DPM). There have been some efforts [8, 9] in developing energy-efficient device scheduling algorithms that minimize the I/O device energy consumption for real-time systems. However, none of them considered the energy consumption of processors. As with the DVS alone scheduling algorithms, DPM alone cannot guarantee the overall system energy consumption is minimized.

In this paper, we analyze the problem of system-wide energy conservation for hard real-time systems based on the preemptive periodic task model with non-preemptive shared resources. Here we define the system-wide energy consumption as the sum of the processor energy consumption and the I/O device (including memory [1]) energy consumption. We propose an online system-wide energy-efficient scheduling algorithm, System-wide Energy-Aware EDF (SYS-EDF), which integrates device scheduling and processor voltage scaling to reduce the overall system energy consumption.

The rest of this paper is organized as follows. Section 2 discusses related work. The problem of energy-aware I/O device scheduling is analyzed in Section 3. Section 4 describes the SYS-EDF algorithm. Section 5 describes how we evaluated our system and presents the results. Section 6 presents our conclusions and describes future work.

2 Related Work

Compared to the research of processor-based energy conservation techniques or I/O-based energy conservation techniques, the research on system-wide energy conservation has received little attention. Only a few papers [4, 11] address this issue. In these papers, the negative effect of lowering processor speed is considered. Optimal slowdown factors of

[1] Some modern DRAM chips can be put in a *power down* state in which only the self-refresh circuitry is active to prevent data loss.

the processor speed to minimize the overall system energy consumption are computed and used as the lower-bound of the processor speed. They both achieve significant energy savings compared to DVS alone algorithms. Our work differs from the previous work in following aspects:

1. Our work supports periodic task sets with non-preemptive shared resources. In the previous work, all tasks were assumed to be fully preemptive. In practice, non-preemptive shared resources are pervasive in real-world applications. For example, a job that performs an uninterruptible I/O operation can block the execution of all jobs with higher priorities. Thus the time for the uninterruptible I/O operation needs to be treated as a non-preemptive resource access. Other resources besides I/O devices include critical sections of code, reader/writer buffers, etc.
2. Our work considers the problem of energy-efficient device scheduling and proposes a device scheduling algorithm, *i.e.*, Conservative Energy-Aware EDF (CEA-EDF). [4] and [11] made simplified assumption for the device scheduling. For example, [4] assumed that there is no delay for device state transition. Therefore, an aggressive device scheduling algorithm which turns off devices whenever they are not in use was implied in this work. However, this aggressive device scheduling is not applicable to hard real-time systems if devices that have non-zero transition delays are used. Similarly, [11] did not propose DPM for I/O devices.

The method proposed in this paper provides a energy-efficient device scheduling algorithm, CEA-EDF, for periodic task sets with non-preemptive shared resources. The optimal processor speed is then analyzed based on the proposed device scheduling algorithm. Finally, the SYS-EDF algorithm is proposed to reduce the overall system energy consumption by integrating CEA-EDF and the processor voltage scaling. To the best of our knowledge, no previous publication has addressed the same problem.

3. Energy-aware device scheduling

I/O devices usually have fewer power states than processors. Throughout this paper, we assume that a device has two states: *active* and *idle*. In a real-time system, in order to guarantee that jobs will meet their deadlines, a device cannot be made idle without knowing when it will be requested by a job, but, the precise time at which an application requests the operating system for a device is usually not known. Even without knowing the exact time at which requests are made, we can safely assume that devices are requested within the time of execution of the job making the request. Therefore, our method is based on inter-task device scheduling rather than intra-task scheduling. That is, the scheduler does not put devices in sleep while tasks that require them are being executed, even though there is no I/O requests at that time.

As discussed before, the energy-aware device scheduling algorithm needs to support the preemptive scheduling of periodic tasks with non-preemptive shared resources. However, the only known published energy-aware device scheduling algorithm for preemptive schedules, Maximum Device Overlap (MDO) [9], does not address the issue of resource blocking. As an offline method, it is difficult to integrate a resource accessing policy into MDO because it is hard to predict exact points that jobs access resources at the offline phase. It is possible that a seemingly feasible offline job schedule causes jobs to miss their deadlines at runtime.

An obvious online approach is to aggressively shut down devices whenever they are not needed; and start them as soon as they are needed, which is called the Aggressive Shut Down (ASD) algorithm [2]. Unfortunately, ASD cannot be directly applied to hard real-time systems, because the power consumption and the delay of the device state transition is usually too large to be neglected.

In our previous study [2], some online device scheduling algorithms that support preemptive schedules with shared resources are proposed. Among them, CEA-EDF can be used together with a DVS-based scheduler without any modification. As we will see shortly, CEA-EDF is independent of processor speed change, which makes it ideal for easy integration with DVS.

3.1 Device energy model

Associated with each device λ_i are the following parameters: the transition time from the *idle* state to the *active* state represented by $t_{wu}(\lambda_i)$; the transition time from the *active* state to the *idle* state represented by $t_{sd}(\lambda_i)$; the energy consumed per unit time in the *active* and *idle* states represented by $P_a(\lambda_i)$ and $P_i(\lambda_i)$ respectively; the energy consumed per transition from the *active* state to the *idle* state represented by $E_{sd}(\lambda_i)$; and the energy consumed per transition from the *idle* state to the *active* state represented by $E_{wu}(\lambda_i)$. We assume that for any device, the state switch can only be performed when the device is in a stable state, *i.e.,* the idle state or the active state. Therefore, the total energy consumed by a device λ_i is given by,

$$E_{\lambda_i} = P_a \times T_a(\lambda_i) + P_i \times T_i + E_{sd} \times N_{sd} + E_{wu} \times N_{sw} \quad (1)$$

where, T_a is the time that λ_i is in the *active* state; T_i is the time that λ_i is in the *idle* state; N_{sd} is the number of the transition of the device from active to idle; and N_{wu} is the number of the transition of the device from idle to active.

3.2. Energy-aware device scheduling

CEA-EDF is a simple, online energy-aware device scheduling algorithm for hard real-time systems. All devices that a job needs are active at or before the job is released. Thus devices are safely shut down without affecting the schedulability of tasks.

```
1  Device scheduling at time t:
2    If (t: instance when job J_{i,j} is completed)
3      If (∃λ_k, λ_k = active and T_req(λ_k, t) − t > BE(λ_k))
4        λ_k → idle;
5        Up(λ_k) ← T_req(λ_k, t) − t_wu(λ_k);
6      End
7    End
8    If (t: ∃λ_k, λ_k = idle and Up(λ_k) = t)
9      λ_k → active;
10     Up(λ_k) ← −1; // Clear the power up timer for λ_k
11   End
```

Figure 1. The CEA-EDF algorithm. $Up(\lambda_k)$ is the power up time set to λ_k, at when the device will be powered up.

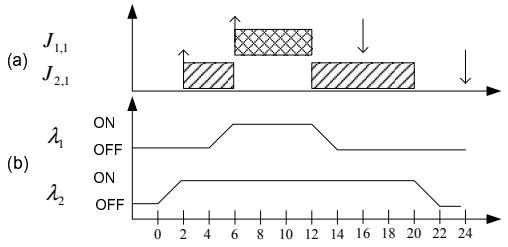


Figure 2. CEA-EDF scheduling example; (a) $J_{1,1}$ is released at 6 and uses device λ_1. $J_{2,1}$ is released at 2 and uses device λ_2. $J_{1,1}$ has a higher priority than $J_{2,1}$. (b) the device state transition with the CEA-EDF algorithm.

Because of the energy consumption associated with the device power state transition, it is not energy-efficient to frequently perform the power state transition. A *break-even time* is used to represent the minimum inactivity time required to compensate for the cost of entering and exiting the idle state. We let $BE(\lambda_i)$ denote the break-even time of device λ_i hereafter. The computation of $BE(\lambda_i)$ using our device energy model can be found in [2]. It is clear that if a device is idle for less than the break-even time, it is not worth performing the state switch. Therefore, CEA-EDF makes decisions of device state transition based on the break-even time rather than device state transition delay.

Next, we define the *next device request time* that is used in keeping track of the earliest time that a device is required.

Definition 3.1. *Next Device Request Time.* The next device request time is denoted by $T_{req}(\lambda_k, t)$ and is the earliest time that a device λ_k is requested by any uncompleted job. Since a job can only use a device after the job is released, the next device request time of a device λ_k is given by

$$T_{req}(\lambda_k, t) = Min(R(J_{i,j})) \tag{2}$$

where $J_{i,j}$ is any uncompleted job that requires device λ_k and $R(J_{i,j})$ is the release time of job $J_{i,j}$.

With CEA-EDF, a device λ_i is switched to the low power state at time t when $T_{req}(\lambda_i, t) - t > BE(\lambda_i)$. CEA-EDF sets a power up time, $Up(\lambda_i)$, for device λ_i when λ_i is switched to the idle state. For any idle device, it is switched back to the active state if the power up time $Up(\lambda_i)$ is equal to the current time t. The CEA-EDF scheduling algorithm then can be described as in Figure 1, and is invoked at *scheduling points* and when a power up time is reached. We define scheduling points as time instances at which jobs are released, completed, or exit critical sections. An example of CEA-EDF scheduling is illustrated in Figure 2.

4. System-wide energy-efficient scheduling

In this section, we first provide a power model for a typical DVS processor. Then we present a system-wide energy-efficient task scheduling algorithm, SYS-EDF, which integrates CEA-EDF and processor voltage scaling.

4.1 DVS processor energy model

In a CMOS circuit, the overall power consumption consists of dynamic power consumption and static power consumption. For a DVS processor, the dynamic power consumption can be given by,

$$P_{AC} = C_{eff} {V_{dd}}^2 f \tag{3}$$

where C_{eff} is the switched capacitance, V_{dd} is the supply voltage and f is the operating frequency. The relationship of f and V_{dd} is given by [6]

$$f = (L_d K_6)^{-1}((1 + K_1)V_{dd} + K_2 V_{bs} - V_{th1})^{\alpha} \tag{4}$$

where V_{bs} is the body bias voltage and $K_1, K_2, K_6, L_d, V_{th1}$ and α are technology constant parameters.

Several leakage sources contribute to the total static power consumption. According to [6], the leakage power dissipation is given by,

$$P_{DC} = L_g(V_{dd} I_{subn} + |V_{bs}| I_j) \tag{5}$$

where L_g is the number of devices in the circuit, I_{subn} is the subthreshold current, and I_j is the reverse bias junction current. The formal mathematical formulation and detailed explanations of related technical parameters can be found in [6]. The total power consumption of a processor is given by,

$$P_{cpu} = \begin{cases} P_{AC} + P_{DC} + P_{on} & \text{CPU is active} \\ 0 & \text{CPU is not active} \end{cases} \tag{6}$$

where P_{on} is an inherent power cost in keeping the processor on [3]. We assume that a processor does not consume energy when it is not in the active state.

Since the voltage transition delay of a processor is very short, we assume that the overhead incurred in changing the processor speed is negligible. The same assumption is made in previous works [7, 10, 11].

4.2. System-wide optimal processor speed

We let ν denote the normalized processor speed. That is, the ratio of the current processor speed to the maximal processor speed. As with previous work [7, 10, 11], we assume that the processor speed is approximately proportional to the current operating frequency f. Thus ν can be represented by f/f_{high}, where f_{high} is the maximum operating frequency. We assume that a DVS processor can provide m discrete operating frequency represented by $\{f_1, f_2, \ldots, f_m = f_{high}\}$.

Because of the standby energy dissipation of I/O devices, the lowest processor speed is not necessarily the most energy-efficient speed as assumed in previous DVS-alone scheduling algorithms. The leakage power dissipation of the processor and the standby energy dissipation of I/O devices increase with the extended task lifetime. Let $\Lambda(t)$ be the active device set that contains all devices that are in the active state at time t. Note that with the CEA-EDF device scheduling algorithm, devices not required by the current executing job may be kept in the active state to ensure the system schedulability. As shown in Figure 2, all devices in $\Lambda(t)$ are kept in the active state until the current job is completed.

Suppose that a processor can complete 1 unit workload in 1 unit time with the highest operating frequency, then the processor will take $1/\nu$ time units to complete 1 unit workload with a processor speed of ν. Next, we introduce *energy efficiency scale* to compare the overall system energy efficiency to complete 1 unit workload with different processor speeds. The energy efficiency scale is denoted by $ES(\nu, \Lambda)$ and is modelled by,

$$ES(\nu, \Lambda(t)) = \frac{P_{cpu}(\nu)}{\nu} + \frac{1}{\nu} \sum_{\lambda_k \in \Lambda(t)} (P_a(\lambda_k) - P_i(\lambda_k)) \quad (7)$$

where $P_{cpu}(\nu)$ is the processor energy consumption rate with a given processor speed ν, which can be acquired from the processor energy model presented in Section 4.1. $P_a(\lambda_k) - P_i(\lambda_k)$ is the difference of the energy consumption rate of device λ_k in the active state and the idle state. We use this difference rather than $P_a(\lambda_k)$ alone to evaluate how much energy can be saved by putting λ_k in the idle state.

The processor speed that can minimize the energy efficiency scale is the system-wide optimal processor speed. We let $\nu_{opt}(\Lambda(t))$ denote the optimal processor speed for a given active device set $\Lambda(t)$. Figure 3 shows the energy efficiency scale for three different active device sets. The CPU is based on Transmeta Crusoe processor with $70nm$ technology [2] [3, 6]. The technical parameters for these devices can be found in Table 1. It can be seen from Figure 3 that the energy efficiency scale varies with different active device sets, and so does the optimal processor speed. For example, the most energy efficient processor speed is 0.4 when only the Mobile RAM is in the active state, while the optimal processor speed is 0.9 when both the Mobile RAM and the MaxStream wireless module are in the active state.

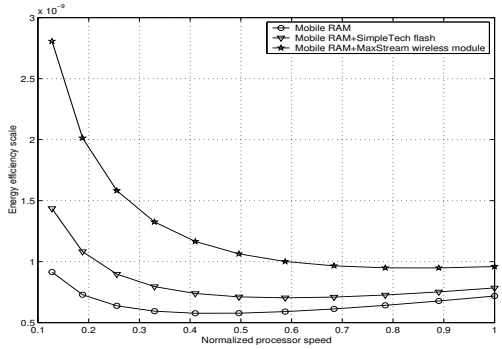


Figure 3. $ES(\nu, \Lambda(t))$ of different normalized processor speeds for three active device sets.

The system-wide optimal processor speed ν_{opt} is computed offline and retrieved at runtime. For a given active device set $\Lambda(t)$, $\nu_{opt}(\Lambda(t))$ can be acquired by computing $ES(\nu, \Lambda(t))$ for all possible ν values. The speed that minimizes $ES(\nu, \Lambda(t))$ is selected to be $\nu_{opt}(\Lambda(t))$. Since modern DVS processors provide finite discrete operating frequencies, this computation can be done in $O(m)$ time complexity for each given $\Lambda(t)$, where m is the number of operating frequencies that the processor can provide. Let K denote the number of devices in the system, then there are at most 2^K possible sets for $\Lambda(t)$. Therefore, the computational complexity of computing ν_{opt} for all possible $\Lambda(t)$ is $O(m \times 2^K)$. With all pre-computed $\nu_{opt}(\Lambda(t))$ stored in memory, retrieving ν_{opt} for any $\Lambda(t)$ at runtime can be done in $O(1)$ time.

4.3. SYS-EDF

The processor voltage scaling in SYS-EDF is based on the Dual Speed (DS) and the Dual Speed Dynamic Reclaiming (DSDR) algorithms proposed by Zhang *et.al,* [10]. The DS algorithm aims to minimize the dynamic energy consumption of the processor for real-time periodic tasks with non-preemptive blocking sections. The DSDR algorithm extends the DS algorithm by dynamically collecting unused run time for further slow down.

However, DS and DSDR considered only the dynamic energy dissipation of the processor. Based on the previous analysis, we develop the SYS-EDF algorithm, which improves DS and DSDR to reduce the overall system energy consumption. For the space limitation, we only discuss the basic improvement done to the DS algorithm in this paper. The basic idea is : the SYS-EDF algorithm keeps track of the active device set and computes the corresponding optimal processor speed. SYS-EDF uses the DS algorithm to

[2] This is a processor model based on the technology trends [6].

```
1  Initialize:
2    ν ← max(L, ν_opt(Λ(t))); END_H ← −1;
3    H and L are pre-computed processor speeds [10];
4  Scheduling at time t:
5    If (t: instance when job J_{i,j} is completed)
6      update Λ(t);
7      If (there is no pending job) ν ← 0;
8      Else ν ← max(ν, ν_opt(Λ(t)));
9      End
10   End
11   If (t: instance when job J_{i,j} is released)
12     update Λ(t);
13     ν ← max(ν, ν_opt(Λ(t)));
14     If (Prio(J_{i,j}) > Prio(J_curr) and J_{i,j} is blocked by J_curr)
15       ν ← max(H, ν_opt(Λ(t)));
16       End_H ← max(End_H, Deadline(J_curr));
17     End
18   End
19   If (t: instance when t = End_H)
20     End_H ← −1;
21     ν ← max(L, ν_opt(Λ(t)));
22   End
23 Schedule devices by CEA-EDF ;
24 Schedule jobs by EDF(SRP);
```

Figure 4. The simplified SYS-EDF algorithm.

adjust the processor speed with only one limitation: the processor speed is never set below the optimal processor speed. The improvement to the DSDR algorithm follows a similar approach, but uses a different dynamic reclaiming algorithm because more than two processor speeds are utilized in SYS-EDF.

The SYS-EDF algorithm is presented in Figure 4. With the proposed device scheduling algorithm, *i.e.,* CEA-EDF, the active device set changes only at the time instances when a job is completed or a new job is released (line 6,12). As with [10], a pre-computed high speed H and a pre-computed low speed L are used in SYS-EDF. Because of the space limitation, we do not present the computation of H and L in this paper. We refer the reader to [10] for the detailed explanation and computation. Since H, L and $\nu_{opt}(\Lambda(t))$ are pre-computed, the overhead of performing SYS-EDF is very low.

4.4. Schedulability

Theorem 4.1. *Suppose n periodic tasks are sorted by their periods. They are schedulable by SYS-EDF if*

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E(T_i)}{P(T_i)} + \frac{B(T_k)}{P(T_k)} \leq 1, \qquad (8)$$

where $E(T_k)$ and $P(T_k)$ are the execution time and period of task T_k respectively; and $B(T_k)$ is the maximal length that a job in T_k can be blocked.

Proof: The SYS-EDF algorithm consists of a energy-efficient device scheduling algorithm (CEA-EDF) and a processor voltage scaling algorithm. With the CEA-EDF algorithm, a device λ_k is guaranteed to be in the active state when any jobs requiring λ_k are released. Therefore, CEA-EDF does not affect the schedulability of any systems.

With the processor voltage scaling algorithm presented in Figure 4, the processor speed is set to the higher speed of the optimal processor speed and the speed when scheduled with the DS scheduling algorithm (line 2,8,15,21). In other words, the SYS-EDF algorithm keeps the processor at a speed no less than the speed when scheduled with the DS algorithm. Since Theorem 4.1 has been proved true for the DS scheduling algorithm in [10], Theorem 4.1 is also true for the SYS-EDF algorithm. □

Device	P_a (W)	P_i (W)	E_{wu}, E_{sd} (mJ) [3]
Realtek Ethernet Chip	0.187	0.085	1.25
MaxStream Wireless module	0.75	0.005	4
IBM Microdrive	1.3	0.1	6
Fujitsu MHL2300AT Hard disk	2.3	1.0	3
SimpleTech Flash Card	0.225	0.02	0.2
Mobile-RAM	0.075	0.00175	≈ 0

Table 1. Device Specifications.

5 Evaluation

This section presents evaluation results for the SYS-EDF algorithm. Section 5.1 describes the evaluation methodology used in this study. Section 5.2 describes the evaluation of SYS-EDF with various system utilizations.

5.1. Methodology

We evaluated the SYS-EDF algorithm using an event-driven simulator. This approach is consistent with evaluation approaches adopted by other researches for energy-aware scheduling [8, 4, 11].

The power requirements and state switching times for devices were obtained from data sheets provided by the manufacturer. The devices used in experiments are listed in Table 1. The DVS processor we simulated is based on Transmeta Crusoe processor with $70nm$ technology [3, 6]. We assume that the processor supports discrete voltage from $0.5V$ to $1.0V$ in steps of $0.05V$. The *normalized energy saving* is used to evaluate the energy savings of the algorithms. The normalized energy saving is the ratio of energy saving under a energy-conservation algorithm to the energy consumption when no energy-conservation technique is used, wherein all devices remain in the active state over the entire simulation.

In all experiments, we used randomly generated task sets to evaluate the performance of all algorithms. All task sets are pretested to satisfy the schedulability condition shown in Equation (8). Each generated task set contained $1 \sim 10$ tasks. Periods of tasks are chosen from $[100, 1000]$. Each

[3] Most vendors report only a single switching energy consumption. Thus we used this data for both E_{wu} and E_{sd}. The sources of these data can be found in [2].

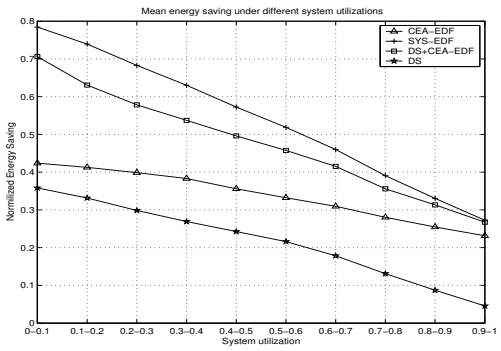


Figure 5. Mean normalized energy savings of different system utilization settings.

task in a task set required the RAM module and additional $0 \sim 2$ other devices from Table 1. Critical sections of all jobs were randomly generated. We repeated each experiment 500 times and present the mean value. During the whole experiment, we assume that the actual execution time of a task is equal to the WCET.

We did not measure scheduling overhead in a real system since all algorithms were evaluated with simulations. Instead, we compared the scheduling overhead of SYS-EDF with respect to EDF(SRP) in our simulations. We used *relative scheduling overhead* to evaluate the scheduling overhead of SYS-EDF. Let ρ denote the *relative scheduling overhead*, which is given by

$$\rho = \frac{\textit{scheduling overhead with SYS-EDF}}{\textit{scheduling overhead with EDF(SRP)}} - 1$$

The mean relative scheduling overhead of SYS-EDF is 3.2%, verifying that the overhead of SYS-EDF is low.

5.2. Average energy savings

To better evaluate the SYS-EDF algorithm, we compare SYS-EDF with three other algorithms for each simulation: (1) CEA-EDF is the algorithm that only performs DPM for devices; (2) DS is the DVS-alone algorithm proposed in [10], which considers only the dynamic energy conservation for processors; and (3) DS +CEA-EDF is the straightforward integration of (1) and (2), without considering the system-wide energy-efficient speed. Since [4] and [11] do not address the problem of resource blocking and the negative effect of device transition delays on system schedulability, we did not compare with them in this evaluation.

Figure 5 shows simulation results of the mean normalized energy saving for the SYS-EDF and other algorithms under different system utilizations. It can be seen that SYS-EDF saves more energy than the other algorithms. SYS-EDF can reduce the system energy consumption by up to 10% over DS +CEA-EDF. In most cases, as the system utilization increases, the normalized energy savings decreases. The rationale for this is that as tasks execute more, the amount of time devices can be kept in *idle* mode decreases and the processor voltage needs to be kept at a high value. As the system utilization approaches 100%, SYS-EDF, CEA-EDF and DVS+DPM perform comparable to each other, because there is not much space for processor energy saving and all of them merely perform DPM for devices.

6 Conclusion

This paper presents a system-wide energy-efficient scheduling algorithm, SYS-EDF, which supports the preemptive scheduling of periodic tasks with non-preemptive shared resources. SYS-EDF consists of a practical DPM algorithm for I/O devices and a corresponding processor voltage scheduling algorithm. The SYS-EDF algorithm provides remarkable power savings by wisely setting the processor speed to balance the energy consumption of all components in the system. The evaluation of SYS-EDF shows that it yields significant energy savings with respect to DVS alone or DPM alone techniques or the straightforward integration of DVS and DPM.

References

[1] Baker, T.P., "Stack-Based Scheduling of Real-Time Processes," *Real-Time Systems*, 3(1):67-99, March 1991.

[2] Cheng, H. and Goddard, S., "Online Energy-Aware I/O Device Scheduling for Hard Real-Time Systems with Shared Resources", Technical Report, 2005, `http://csce.unl.edu/~hcheng/TR_CEAEDF_EASD.pdf`

[3] Jejurikar, R., Pereira, C., Gupta, R., "Leakage aware dynamic voltage scaling for real-time embedded systems", *41st annual conference on Design automation*, 2004.

[4] Jejurikar, R., Gupta, R., "Dynamic Voltage Scaling for Systemwide Energy Minimization in Real-Time Embedded Systems", *Symp. on Low power electronics and design*, 2004.

[5] Liu and Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM*, 20(1), January, 1973.

[6] Martin, S., Flautner, K., Mudge, T., Blaauw, D., "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads", *IEEE/ACM intl. conf. on Computer-aided design*, 2002.

[7] Qadi, A., Goddard, S., Farritor, S., "A Dynamic Voltage Scaling Algorithm for Sporadic Tasks", *IEEE Real-Time Systems Symp.*, 2003.

[8] Swaminathan, V., Chakrabarty, K., and Iyengar, S.S., "Dynamic I/O Power Management for Hard Real-time Systems" *9th Intl. Symp. on Hardware/Software Codesign*, 2001.

[9] Swaminathan, V., and Chakrabarty, K., "Pruning-based, Energy-optimal, Deterministic I/O Device Scheduling for Hard Real-Time Systems", *ACM Transactions on Embedded Computing Systems*, 4(1):141-167, February 2005.

[10] Zhang, F., Chanson, S., "Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections", *IEEE Real-Time Systems Symp.*, 2002.

[11] Zhuo, J., Chakrabarti, C., "System-Level Energy-Efficient Dynamic Task Scheduling", *42nd annual conf. on Design automation*, 2004.

Energy-Efficient Scheduling of Periodic Real-Time Tasks over Homogeneous Multiprocessors

Jian-Jia Chen and Tei-Wei Kuo
Department of Computer Science and Information Engineering,
Graduate Institute of Networking and Multimedia,
National Taiwan University, Taipei, Taiwan, ROC.
Email:{r90079, ktw}@csie.ntu.edu.tw

Abstract

Different from many previous energy-efficient scheduling studies, this paper explores energy-efficient multiprocessor scheduling of periodic real-time tasks with different power consumption functions. When the goal is on the minimization of energy consumption, we propose a 1.412*-approximation algorithm in the derivation of a feasible schedule. A series of simulation experiments was done for the performance evaluation of the proposed algorithm.*

1 Introduction

With the advance technology of VLSI circuit designs, many modern processors, such as the Intel StrongARM SA1100 processor [19] and the Intel XScale [20], could now operate at various supply voltages and have different processor speeds. The power consumption of processors is usually a convex and increasing function of processor speeds, which is highly dependent on the hardware designs. The lower the speed, the less the power consumption is, where a lower processor speed usually means longer execution time for tasks.

In the past decades, energy-efficient task scheduling with various deadline constraints has received a lot of attention. Although many studies have been done for uniprocessor scheduling, such as [4, 6, 10, 11, 17, 25], not much work has been done for multiprocessor scheduling. As pointed out in [2], implementations of real-time systems with multiple processors could be often much more energy-efficient than those with a single processor, because of the convexity of power consumption functions. Due to the $\mathcal{NP}$-hardness of many multiprocessor energy-efficient scheduling problems, various heuristics were proposed in the derivation of schedules for different task models with an objective in the minimization of energy consumption, e.g., [1, 5, 8, 9, 12, 13, 18, 24, 26, 27]. In particular, several energy-efficient scheduling algorithms based on list heuristics were proposed [12, 13, 26]. Heuristic algorithms for periodic tasks in multiprocessor environments were proposed in [1, 5]. Zhu, et al. [27] explored on-line task scheduling with reclamation of slacks resulted from early completion of tasks during the run time. Mishra, et al. [18] explored energy-efficient scheduling issues with the considerations of the communication delay of tasks. In addition to the considerations of energy-efficient scheduling, Anderson and Sanjoy [2] explored the tradeoff between the total energy consumption of task executions and the number of required processors, where tasks in the proposed solutions run at the same speed. So far, not much work is done with approximation ratios in energy-efficient multiprocessor real-time scheduling. An example result is the approximation algorithms proposed for the scheduling of frame-based tasks in [8], where tasks share the same power consumption function, and [9], where tasks might have different power consumption functions. Energy-efficient multiprocessor scheduling of frame-based task sets was also explored in [24] for chip-multiprocessor (CMP) architectures, in which cores, i.e., processors, on a chip must share the same processor speed at any given time moment.

This paper considers energy-efficient scheduling of periodic real-time tasks over multiple processors. Different from previous energy-efficient scheduling studies, this research explores energy-efficient multiprocessor scheduling for periodic real-time tasks, in which each task might have different periods, initial arrival times, CPU execution cycles, and power consumption functions. The power consumption functions of tasks are modeled as $h \cdot s^{\alpha}$ [6, 14, 25], where α is a hardware-dependent factor, and h is a parameter related to the task under executions (Please see the discussions of power consumption functions in the next section). When the goal is on the minimization of energy consumption, we propose an approximation algorithm with an approximation ratio $\frac{(\alpha-1)^{\alpha-1}(2^{\alpha}-1)^{\alpha}}{\alpha^{\alpha}(2^{\alpha}-2)^{\alpha-1}}$, which is bounded by 1.412 since the value of α is at most 3 [6, 14, 25], in the

derivation of a feasible schedule. Simulation results show that our proposed algorithm not only guarantees the approximation factors but also derives solutions close to optimal solutions.

The rest of this paper is organized as follows: In Section 2, we define the system models and the multiprocessor energy-efficient scheduling problem. Section 3 presents an approximation algorithm for the multiprocessor energy-efficient scheduling problem. Section 4 presents evaluation results. Section 5 is the conclusion.

2 Models and Problem Definitions

2.1 Processor Models

We are interested in energy-efficient scheduling over homogeneous multiprocessors, where the power consumption function of each task remains the same for every processor. The power consumption function $P()$ in the dynamic voltage circuits is defined as a function of the adopted processor speed s [7, 23]:

$$P(s) = C_{ef} V_{dd}^2 s, \quad (1)$$

where $s = k\frac{(V_{dd}-V_t)^2}{V_{dd}}$, and C_{ef}, V_t, V_{dd}, and k denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively ($V_{dd} \geq V_t \geq 0$, $k > 0$, and $C_{ef} > 0$). The value of the effective switch capacitance is highly related to the software implementation and the execution path of a task (usually derived by profiling). Note that the power consumption function is a convex and increasing function of processor speeds. When V_t is 0, the power consumption function $P(s)$ could be rephrased as a cubic function of the processor speed s. As reported in the literature, e.g., [6, 14, 25], the power consumption function can be phrased as $h \cdot s^\alpha$, where α is a hardware-dependent factor, and h is a parameter related to the task under executions.

In this study, we assume that each processor could operate at any speed in $[0, \infty]$, and the speed of each processor could be adjusted independently from each another. We assume that the number of CPU cycles executed in a time interval is linearly proportional to the processor speed, and that the energy consumed for a processor in the execution of a task at the processor speed s for t time units is the multiplication of t and its corresponding power consumption $P(s)$ at the speed s. Let the amount of CPU cycles completed for a task running at a speed s for t time units be the multiplication of s and t. Suppose that the time and energy overheads required on speed/voltage switching be negligible.

2.2 Task Models

Tasks under discussions in this paper are periodic and independent in executions. A periodic task is an infinite sequence of task instances, referred to as *jobs*, where each job of a task comes in a regular period [15, 16]. Each task τ_i is associated with its initial arrival time (denoted by a_i), its execution CPU cycles (denoted by c_i), its period (denoted by p_i), and its power consumption function (denoted by $P_i()$). Note that c_i denotes the maximum number of CPU cycles required to complete the execution of any job of τ_i. The power consumption function $P_i()$ of each task τ_i is rephrased as a convex and increasing function of the processor speed s, i.e., $P_i(s) = h_i \cdot s^\alpha$, where α is a hardware-dependent constant between 2 and 3 [17, 21], and h_i is a positive parameter characterizing the average switch capacitance and the hardware factor. It is clear that each $P_i(s)$ is second-order differentiable. Given a set $\mathbf{T}$ of tasks, the *hyper-period* of $\mathbf{T}$, denoted by L, is defined as the least common multiple (LCM) of the periods of tasks in $\mathbf{T}$. Let the relative deadline of each task τ_i be equal to its period p_i in this paper. That is, the arrival time and deadline of the j-th job of task τ_i are $a_i + (j-1) \cdot p_i$ and $a_i + j \cdot p_i$, respectively.

2.3 Problem Definitions

A *schedule* of a task set $\mathbf{T}$ is a mapping of the executions of tasks in $\mathbf{T}$ to processors in the system with an assignment of a processor speed for each corresponding task execution, where the job arrivals of each task $\tau_i \in \mathbf{T}$ satisfy its timing constraints a_i and p_i. A schedule is *feasible* if no job misses its deadline, and all jobs of the same task execute on the same processor. The energy consumption of a schedule S, denoted as $\Phi(S)$, is the sum of the energy consumption of the executions of jobs in S. We are interested in real-time energy-efficient scheduling of independent tasks over multiple processors, where no task migration is allowed:

Definition 1 *The Minimization Problem of the Energy Consumption for Multiprocessor Scheduling*

Given a set $\mathbf{T}$ of independent tasks executing over M identical processors, the objective is to find a feasible schedule S for $\mathbf{T}$ in its hyper-period such that $\Phi(S)$ is minimized. □

Suppose that jobs of each task τ_i in a given schedule S execute at a speed s_i. $\Phi(S)$ is equal to $\sum_{\tau_i \in \mathbf{T}} \frac{L}{p_i} P_i(s_i) \frac{c_i}{s_i}$, where $\mathbf{T}$ is a given set of tasks under considerations, and L is the hyper-period of $\mathbf{T}$.

Theorem 1 *The Minimization Problem of the Energy Consumption for Multiprocessor Scheduling is $\mathcal{NP}$-hard.*

Proof. The correctness of this theorem follows from the fact that the corresponding problems, when $P_i(s) = s^3$, $a_i = 0$, and $p_i = D$, are $\mathcal{NP}$-hard (A similar argument to the proof in [8, Theorem 1]). □

With the $\mathcal{NP}$-hardness of the above problems, the objective of this research is to propose approximated solutions

with approximation bounds. Formally, a γ-approximation algorithm for the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling is an algorithm that derives a feasible schedule with an amount of energy consumption no more than γ times of an optimal solution (based on the definition of γ approximation in [22, §1]).

3 On the Minimization Problem of the Energy Consumption

In this section, we propose an approximation algorithm for the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling. If the number of tasks in $\mathbf{T}$ is no more than M, an optimal schedule would execute each task τ_i on a different processor at the speed c_i/p_i, for $i = 1, \ldots, |\mathbf{T}|$. For the rest of this section, we will focus our discussions on cases, where the number of tasks in $\mathbf{T}$ is more than M.

Let S be a feasible schedule of $\mathbf{T}$ for the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling. Let S_m denote the partial schedule of S on the m-th processor by removing the tasks running on the other processors, and T_m denote the set of tasks assigned to execute on the m-th processor. Note that $\cup_{m=1}^{M} T_m = \mathbf{T}$ and $T_m \cap T_n = \emptyset$ for any $m \neq n$. We claim that there must exist an optimal schedule S^* that satisfies the following two properties for any partial schedule S_m^* of S^*, where $1 \leq m \leq M$: (1) For every task τ_i in T_m^*, all jobs of τ_i execute at a common processor speed. (2) The total utilization tasks in S_m^*, which is defined as the sum of the utilization of each task (i.e., its execution time divided by its period) in S_m^*, is equal to 100%. This claim could be proved based on the convexity of power consumption functions by a similar argument to that for optimal energy-efficient scheduling in a uniprocessor system [4].

Let x_{im} be a binary variable to indicate whether τ_i is assigned to execute on the m-th processor, and t_i be a variable denoting the execution time of task τ_i. We can re-formulate the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling as a convex programming as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{\tau_i \in \mathbf{T}} E_i(t_i) \\ \text{subject to} & \sum_{\tau_i \in \mathbf{T}} x_{im} \cdot t_i/p_i = 1, \text{ for } \quad m = 1, \ldots, M \\ & t_i > 0, \quad \forall \tau_i \in \mathbf{T}, \\ & \sum_{m=1}^{M} x_{im} = 1, \quad \forall \tau_i \in \mathbf{T}, \text{ and} \\ & x_{im} \in \{0, 1\}, \quad \forall m = 1, \ldots, M, \text{ and } \tau_i \in \mathbf{T}, \end{array}$$

where $E_i(t_i)$ is defined as the *energy consumption* to execute all of the jobs of τ_i in the hyper-period L at the speed $\frac{c_i}{t_i}$, i.e., $E_i(t_i) = \frac{L}{p_i} P_i(\frac{c_i}{t_i}) t_i = \frac{L h_i}{p_i} \frac{c_i^{\alpha}}{t_i^{\alpha-1}}$. The reason why $t_i > 0$ comes from the assumption that the available speeds are continuous in $[0, \infty]$.

Our proposed algorithm for the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling consists of two phases: the relaxation phase and the rounding phase. In the relaxation phase, we relax the integral constraints on the variables x_{im} and derive an optimal solution for the relaxed problem (which is a lower bound on the energy consumption of an optimal schedule). In the rounding phase, we derive a feasible schedule based on the solution derived in the first phase.

3.1 Relaxation Phase

With the integral constraints on x_{im} being relaxed, we could first rewrite the above convex programming problem as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{\tau_i \in \mathbf{T}} E_i(t_i), \\ \text{subject to} & \sum_{\tau_i \in \mathbf{T}} t_i/p_i = M, \text{ and} \\ & 0 < t_i \leq p_i. \end{array} \tag{2}$$

An optimal solution for Equation (2) is a lower bound on the energy consumption for optimal schedules for $\mathbf{T}$ in the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling. Equation (2) can be resolved by applying the Karush-Kuhn-Tucker optimality condition in $O(|\mathbf{T}| \log |\mathbf{T}|)$. (Detail procedures to derive an optimal solution of Equation (2) can be found in [3, 4, 9].) Let $(t_1^*, t_2^*, \ldots, t_{|\mathbf{T}|}^*)$ be an optimal solution for Equation (2).

Lemma 1 *When $t_i^* < p_i$ and $t_j^* < p_j$, $p_i E_i'(t_i^*) = p_j E_j'(t_j^*)$, where $E_i'()$ and $E_j'()$ are the derivatives of $E_i()$ and $E_j()$, respectively.*

Proof. This Lemma is based on the Karush-Kuhn-Tucker condition for the optimal solution $(t_1^*, t_2^*, \ldots, t_{|\mathbf{T}|}^*)$, in which $E_i'(t_i^*) - \frac{\lambda}{p_i} = 0$, and $E_j'(t_j^*) - \frac{\lambda}{p_j} = 0$ for some constant λ when $t_i^* < p_i$ and $t_j^* < p_j$. □

3.2 Rounding Phase

Let the utilization $u_i^* = t_i^*/p_i$ of task τ_i in $\mathbf{T}$ derived in the first phase be called the *estimated utilization* of τ_i. In this phase, we derive a feasible schedule based on the estimated utilizations of the tasks derived in the first phase, i.e., $(u_1^*, u_2^*, \ldots, u_{|\mathbf{T}|}^*)$, by adopting the *Largest-Estimated-Utilization-First* strategy. The proposed algorithm is shown in Algorithm 1 and denoted as Algorithm LEUF:

Let T_m denote the set of the tasks assigned to execute on the m-th processor, which is an empty set initially. U_m denotes the *total estimated utilization* on the m-th processor, which is defined as the sum of the estimated utilizations of tasks in T_m. Tasks are considered to execute on a selected processor in a non-increasing order of their estimated utilizations. A task under consideration is assigned to execute on the m-th processor with the smallest total estimated utilization U_m (Tie-breaking is done by choosing the smallest index m). After all of the tasks in $\mathbf{T}$ are assigned to execute

on a specific processor, the utilization of τ_i is set as $\frac{u_i^*}{U_m}$ for every task τ_i in T_m. That is, the execution time of every job of task τ_i is set as $\frac{t_i^*}{U_m}$. The transformation of job execution times would result in a situation in which the total utilization of tasks assigned on a processor is exactly equal to 100%. The scheduling of tasks on each processor could be done successfully by the earliest-deadline-first scheduling algorithm because the earliest-deadline-first scheduling algorithm could always schedule periodic real-time independent tasks with a total utilization no more than one [15]. The time complexity of Algorithm LEUF is $O(|\mathbf{T}| \log |\mathbf{T}|)$. For the simplicity of representation, any schedule derived by Algorithm LEUF is denoted as S_{LEUF}.

Algorithm 1 : LEUF

Input: $(\mathbf{T}, M)$;
Output: A feasible schedule;
1: **if** $|\mathbf{T}| \leq M$ **then**
2: return the schedule by executing each task τ_i in $\mathbf{T}$ at the speed $\frac{c_i}{p_i}$ on the i-th processor;
3: let u_i^* be the estimated utilization for $\tau_i \in \mathbf{T}$;
4: sort $\mathbf{T}$ in a non-increasing order of their estimated utilizations;
5: $U_1 \leftarrow U_2 \leftarrow \cdots \leftarrow U_M \leftarrow 0$, and $T_1 \leftarrow T_2 \leftarrow \cdots \leftarrow T_M \leftarrow \emptyset$;
6: **for** $i = 1$ to $|\mathbf{T}|$ **do**
7: find the smallest U_m; (break ties by choosing the smallest index m)
8: $T_m \leftarrow T_m \cup \{\tau_i\}$ and $U_m \leftarrow U_m + u_i^*$;
9: **for** $m = 1$ to M **do**
10: **for each** task $\tau_i \in T_m$ **do**
11: $t_i' \leftarrow t_i^* \times \frac{1}{U_m}$;
12: return the schedule S_{LEUF} which executes task τ_i in T_m ($1 \leq m \leq M$) at the speed c_i/t_i' on the m-th processor in an earliest-deadline-first order;

3.3 Analysis of the Approximation Ratio

For notational brevity, let e_i^* be the *estimated energy consumption* of the jobs of task τ_i in the hyper-period, i.e., $e_i^* = E_i(t_i^*)$. Let $\mathbf{T}'$ be the subset of $\mathbf{T}$, where $\mathbf{T}'$ consists of tasks whose estimated utilizations are all strictly less than 1. That is, $\mathbf{T}' = \{\tau_i \mid t_i^*/p_i < 1, \forall \tau_i \in \mathbf{T}\}$. For notational brevity, let $\hat{\mathbf{T}}$ be $\mathbf{T} \setminus \mathbf{T}'$. Note that we only focus our discussions on the case that $\mathbf{T}'$ is not empty, since Algorithm LEUT guarantees to derive an optimal schedule for the other case.

Lemma 2 *For any two tasks $\tau_i, \tau_j \in \mathbf{T}'$, $\frac{e_i^*}{u_i^*} = \frac{e_j^*}{u_j^*}$.*

Proof. By the equality of $h_i \frac{L}{p_i} \frac{c_i^\alpha}{(t_i^*)^\alpha} \cdot p_i = \frac{L}{p_j} h_j \frac{c_j^\alpha}{(t_j^*)^\alpha} \cdot p_j$ in Lemma 1, we know that $\frac{u_i^*}{u_j^*} = \frac{e_i^*}{e_j^*}$. □

Lemma 3 *Suppose that U_{m^*} and $U_{\hat{m}}$ are the maximum and the minimum total utilizations, respectively, then $U_{\hat{m}} \leq 1 \leq U_{m^*} \leq 2U_{\hat{m}}$.*

Proof. By definition, we know that $U_{\hat{m}} \leq 1 \leq U_{m^*}$. If U_{m^*} is equal to 1, we know that $U_{\hat{m}}$ is also equal to 1 by applying the pigeon-hole principle. For the rest of this discussion, we only focus on the other case that U_{m^*} is greater than 1. Since the estimated utilization of a task is no greater than 1, T_{m^*} consists of at least two tasks. Let τ_v be the last one inserted into T_{m^*}. Since the tasks are assigned in a non-increasing order of their estimated utilization to execute on the processor whose current total estimated utilization is the smallest, we know $u_v^* \leq U_{m^*} - u_v^* \leq U_{\hat{m}}$. Therefore, we have $U_{m^*} \leq 2U_{\hat{m}}$. □

Lemma 4 *Suppose $f(x) = k \cdot (2x)^\alpha + (H - k)x^\alpha$ for a positive number H and a non-negative number k, where $0 \leq k \leq H$ and $2k \cdot x + (H - k) \cdot x = H$, then*

$$f(x) \leq \frac{(\alpha - 1)^{\alpha-1}(2^\alpha - 1)^\alpha}{\alpha^\alpha (2^\alpha - 2)^{\alpha-1}} H.$$

Proof. Since $2k \cdot x + (H - k) \cdot x = H$, we know $k = \frac{H - Hx}{x}$. Therefore,

$$f(x) \quad = H(x^{\alpha-1}(2^\alpha - 1) + x^\alpha(2 - 2^\alpha)),$$

and the derivative of $f(x)$ is

$$f'(x) = H((\alpha - 1)x^{\alpha-2}(2^\alpha - 1) + \alpha x^{\alpha-1}(2 - 2^\alpha)).$$

$f(x)$ is maximized at x^* when $f'(x^*) = 0$. By solving $f'(x^*) = 0$, we have $x^* = \frac{(\alpha-1)(2^\alpha-1)}{\alpha(2^\alpha-2)}$. As a result, we conclude that $f(x) \leq f(x^*) = \frac{(\alpha-1)^{\alpha-1}(2^\alpha-1)^\alpha}{\alpha^\alpha(2^\alpha-2)^{\alpha-1}} H$. □

Based on Lemmas 2, 4, and 3, the approximation ratio of the algorithm could be proved as follows:

Theorem 2 *Algorithm* LEUT *is a polynomial-time $\frac{(\alpha-1)^{\alpha-1}(2^\alpha-1)^\alpha}{\alpha^\alpha(2^\alpha-2)^{\alpha-1}}$-approximation algorithm for the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling.*

Proof. Let τ_r be a task in $\mathbf{T}'$. Based on Lemma 2 and the optimality of $\sum_{\tau_i \in \mathbf{T}} e_i^*$, we have $\Phi(S^*) \geq \sum_{\tau_i \in \mathbf{T}} e_i^* = \sum_{\tau_i \in \hat{\mathbf{T}}} e_i^* + e_r^*/u_r^* \sum_{\tau_i \in \mathbf{T}'} u_i^* = \sum_{\tau_i \in \hat{\mathbf{T}}} e_i^* + e_r^*/u_r^*(M - |\hat{\mathbf{T}}|)$, where S^* is an optimal schedule for $\mathbf{T}$.

Since u_i^* is equal to 1 for $1 \leq i \leq |\hat{\mathbf{T}}|$, the i-th processor is assigned only a task in S_{LEUF} . Based on Lemma 2, we have

$$\Phi(S_{\text{LEUF}}) = \sum_{\tau_i \in \hat{\mathbf{T}}} e_i^* + \sum_{m=|\hat{\mathbf{T}}|+1}^{M} \frac{e_r^*}{u_r^*} (U_m)^\alpha. \qquad (3)$$

The approximation ratio $\mathcal{A}$ of Algorithm LEUF is

$$\mathcal{A} = \frac{\Phi(S_{\text{LEUF}})}{\Phi(S^*)} \leq \frac{\sum_{m=|\hat{\mathbf{T}}|+1}^{M} (U_m)^\alpha}{M - |\hat{\mathbf{T}}|}. \qquad (4)$$

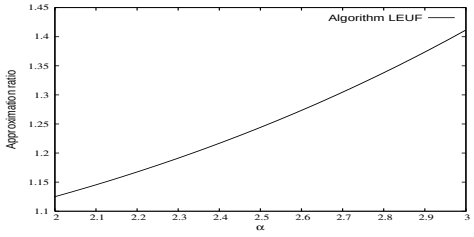


Figure 1. The approximation ratio of Algorithm LEUF for different values of α.

Based on Lemma 3, we have $2U_{\hat{m}} \geq U_{m^*} \geq U_m \geq U_{\hat{m}}$, for all $|\hat{\mathbf{T}}| < m \leq M$. Because of the convexity of the function U_m^α of U_m and the fact $2U_{\hat{m}} - U_m \geq 0$, we have

$$\sum_{m=|\hat{\mathbf{T}}|+1}^{M} U_m^\alpha \leq k \cdot (2U_{\hat{m}})^\alpha + (M - |\hat{\mathbf{T}}| - k)(U_{\hat{m}})^\alpha,$$

where $2k \cdot U_{\hat{m}} + (M - |\hat{\mathbf{T}}| - k)U_{\hat{m}} = (M - |\hat{\mathbf{T}}|)$. Let $f(x)$ be defined as $k \cdot (2x)^\alpha + (H - k)x^\alpha$ for a positive number H and a non-negative number k, where $k \leq H$ and $2k \cdot x + (H - k) \cdot x = H$. By Lemma 4, $f(x) \leq \frac{(\alpha-1)^{\alpha-1}(2^\alpha-1)^\alpha}{\alpha^\alpha(2^\alpha-2)^{\alpha-1}} H$ by solving $f'(x) = 0$. By setting H as $(M - |\hat{\mathbf{T}}|)$ and considering Equation (4), this theorem is proved. □

Corollary 1 *The approximation ratio of Algorithm* LEUF *is* 1.412.

Proof. The proof is done by setting α as 3. □

For different values of α, the approximation ratio of Algorithm LEUF is illustrated in Figure 1.

4 Performance Evaluation

In this section, we provide performance evaluation on the energy consumption of Algorithm LEUF. We also implemented algorithms in [1, 5], and revised the algorithm [8] by sorting tasks in a non-increasing order of c_i/p_i. However, the performace of these algorithms was much worse than Algorithm LEUF since they were proposed for tasks with the same power consumption function. Hence, another algorithm, denoted as Algorithm RAND, which is very similar to Algorithm LEUF, was simulated for comparison. The only difference between Algorithm RAND and Algorithm LEUF is that tasks are not sorted before the assignment procedure in Algorithm RAND.

4.1 Workload Parameters and Performance Metrics

Each periodic real-time task was generated based on three parameters: the number b_i of jobs within the time interval L, the required CPU cycles c_i, and the coefficient h_i of the power consumption function. The value of b_i was an integral variable uniformly distributed in the range of $[1, 16]$. c_i was an integral variable uniformly distributed in the range of $[1, 100]$, while h_i was uniformly distributed in the range of $[2, 10]$. The exponent of the power consumption functions of the processor speed s was set as 3, i.e., $P_i(s) = h_i s^3$, provided that the threshold voltage V_t is 0. To evaluate the effect of the exponent of the power consumption function, we also perform simulations by setting α as a random variable between 2.5 and 3 used for a set of tasks under simulations. The period of task τ_i was set as $\frac{L}{b_i}$.

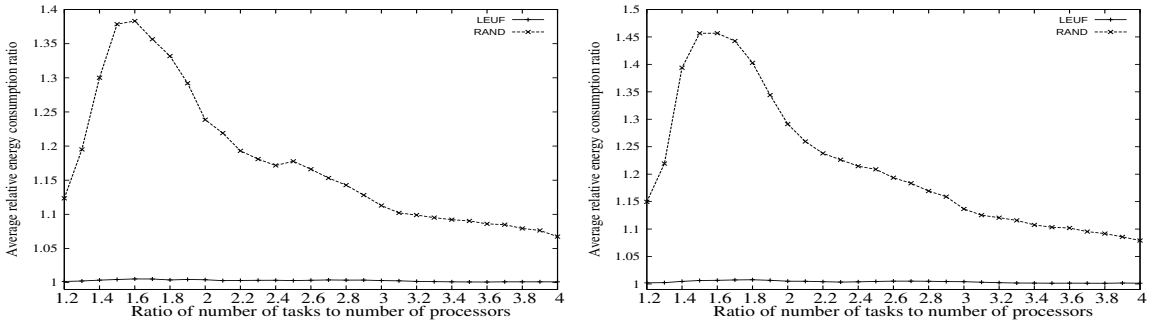
We simulated the algorithms for the effects on the ratio of the number of tasks to the number of processors. For a given ratio η of the number of tasks to the number of processors, the number of processors M was an integral random variable between 10 and 30, and the number of tasks was set as the floor of the multiplication of η and M, i.e., $\lfloor \eta \cdot M \rfloor$. The *relative energy consumption ratio* was adopted as the performance metric in our experiments. The relative energy consumption ratio for an input instance was defined as the energy consumption of the schedule derived by the algorithm to the optimal solution of Equation (2).

4.2 Experimental Results

For the Minimization Problem of the Energy Consumption for Multiprocessor Scheduling, Figures 2(a) and 2(b) present the average relative energy consumption ratios for the simulated algorithms when α is in the range of $[2.5, 3]$ and is 3, respectively. The performance of Algorithm LEUF was very close to that of the optimal solutions. The average relative energy consumption ratios for Algorithm LEUF were less than 1.01. The average relative energy consumption ratios for Algorithm RAND were less than 1.46. When the ratio of the number of tasks to the number of processors was small, both of Algorithm LEUF and Algorithm RAND might assign a task along with improper tasks on a processor. Such an assignment might result in a significant increase on the energy consumption of these tasks when the energy consumption for the other tasks were almost as the same as that in the optimal schedule. When the ratio of the number of tasks to the number of processors was small, in most cases, most processors were assigned with only one task, and the assignment was almost as the same as that of an optimal schedule. Therefore, the average energy consumption ratio was relatively small when the ratio of the number of tasks to the number of processors was less than 1.6.

5 Conclusion

In this paper, we explore approximation algorithms for energy-efficient scheduling of periodic real-time tasks over



(a) Average ratio when α is in the range of 2.5 and 3

(b) Average ratio when $\alpha = 3$

Figure 2. (a) and (b): average relative energy consumption ratios for different settings on α.

multiple processors, where the scheduling problem is $\mathcal{NP}$-hard. The task model explored in this work is more general than many previous studies in energy-efficient multiprocessor real-time scheduling, where tasks under considerations might have different periods, initial arrival times, CPU execution cycles, and power consumption functions. When the goal is on the minimization of energy consumption, we propose a 1.412-approximation algorithm in the derivation of a feasible schedule.

References

[1] T. A. AlEnawy and H. Aydin. Energy-aware task allocation for rate monotonic scheduling. In *Proceedings of the 11th IEEE Real-time and Embedded Technology and Applications Symposium (RTAS'05)*, pages 213–223, 2005.

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

[3] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Optimal reward-based scheduling for periodic real-time tasks. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 79–89, 1999.

[4] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proceedings of the IEEE EuroMicro Conference on Real-Time Systems*, pages 225–232, 2001.

[5] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of 17th International Parallel and Distributed Processing Symposium (IPDPS)*, pages 113 – 121, 2003.

[6] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *Proceedings of the 2004 Symposium on Foundations of Computer Science*, pages 520–529, 2004.

[7] A. Chandrakasan, S. Sheng, and R. Broderson. Lower-power CMOS digital design. *IEEE Journal of Solid-State Circuit*, 27(4):473–484, 1992.

[8] J.-J. Chen, H.-R. Hsu, K.-H. Chuang, C.-L. Yang, A.-C. Pang, and T.-W. Kuo. Multiprocessor energy-efficient scheduling with task migration considerations. In *EuroMicro Conference on Real-Time Systems (ECRTS'04)*, pages 101–108, 2004.

[9] J.-J. Chen and T.-W. Kuo. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *International Conference on Parallel Processing (ICPP)*, pages 13–20, 2005.

[10] J.-J. Chen, T.-W. Kuo, and H.-I. Lu. Power-saving scheduling for weakly dynamic voltage scaling devices. In *Workshop on Algorithms and Data Structures (WADS)*, pages 338–349, 2005.

[11] J.-J. Chen, T.-W. Kuo, and C.-L. Yang. Profit-driven uniprocessor scheduling with energy and timing constraints. In *ACM Symposium on Applied Computing*, pages 834–840, 2004.

[12] F. Gruian. System-level design methods for low-energy architectures containing variable voltage processors. In *Power-Aware Computing Systems*, pages 1–12, 2000.

[13] F. Gruian and K. Kuchcinski. Lenes: Task scheduling for low energy systems using variable supply voltage processors. In *Proceedings of Asia South Pacific Design Automation Conference*, pages 449–455, 2001.

[14] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 37–46. Society for Industrial and Applied Mathematics, 2003.

[15] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[16] J. W. Liu. *Real-Time Systems*. Prentice Hall, Englewood, Cliffs, NJ., 2000.

[17] P. Mejía-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2):284–306, 2004.

[18] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem. Energy aware scheduling for distributed real-time systems. In *International Parallel and Distributed Processing Symposium*, page 21, 2003.

[19] INTEL. Strong ARM SA-1100 Microprocessor Developer's Manual, 2003. INTEL.

[20] INTEL-XSCALE, 2003. http://developer.intel.com/design/xscale/.

[21] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proceedings of the 36th ACM/IEEE Conference on Design Automation Conference*, pages 134–139. ACM Press, 1999.

[22] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.

[24] 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 8th Conference of Design, Automation, and Test in Europe (DATE)*, pages 468–473, 2005.

[25] 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*, pages 374–382. IEEE, 1995.

[26] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Annual ACM IEEE Design Automation Conference*, pages 183–188, 2002.

[27] D. Zhu, R. Melhem, and B. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In *Proceedings of IEEE 22th Real-Time System Symposium*, pages 84–94, 2001.

Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy*

Vibhore Vardhan, Daniel Grobe Sachs, Wanghong Yuan, Albert F. Harris,
Sarita V. Adve, Douglas L. Jones, Robin H. Kravets, and Klara Nahrstedt
University of Illinois at Urbana-Champaign
grace@cs.uiuc.edu

Abstract

Energy efficiency has become a primary design criterion for mobile multimedia devices. Prior work has proposed saving energy through coordinated adaptation in multiple system layers, in response to changing application demands and system resources. The scope and frequency of adaptation pose a fundamental conflict in such systems. The Illinois GRACE project addresses this conflict through a hierarchical solution which combines (1) infrequent (expensive) global *adaptation that optimizes energy for all applications in the system and (2) frequent (cheap)* per-application *(or* per-app*) adaptation that optimizes for a single application at a time. This paper demonstrates the benefits of the hierarchical adaptation through a second-generation prototype, GRACE-2. Specifically, it shows that in a network bandwidth constrained environment, per-app application adaptation yields significant energy benefits over and above global adaptation.*

1 Introduction

Mobile devices primarily running soft real-time multimedia applications are becoming an increasingly important computing platform. Such systems are often limited by their battery life, and saving energy is a primary design goal. A widely used energy saving technique is to adapt the system in response to changing application demands and system resources. Researchers have proposed such adaptations in all layers of the system; e.g., hardware, application, operating system, and network. Recent work has demonstrated significant energy benefits in systems that employ coordinated multiple adaptive system layers or cross-layer adaptation [30, 31].

Such systems must employ intelligent control algorithms that determine when and what adaptations to invoke, to exploit the full potential of the underlying adaptations. These algorithms must balance the conflicting demands of adaptation scope and frequency. On one hand, an algorithm that considers all applications and adaptive system layers, referred to as global, is likely to save more energy than a more limited scope algorithm (e.g., considering only one application at a time). On the other hand, global algorithms are also likely to be more expensive since they must optimize across the cross-product of all configurations of all adaptive layers, considering the demands of all (possibly adaptive) applications on these configurations.

Previous cross-layer adaptation work, therefore, performs global adaptation relatively infrequently (e.g., when an application enters or leaves the system [30, 31]). This infrequent invocation in turn reduces the system's responsiveness to change, potentially sacrificing energy benefits. Other work performs adaptations more frequently, but assumes only one application in the system [25] or only a single adaptive layer [8].

To balance the conflict of frequency vs. scope, the Illinois GRACE project (Global Resource Adaptation through CoopEration) takes a *hierarchical approach* that invokes expensive global adaptation occasionally, and inexpensive limited-scope adaptations frequently [24, 30, 31]. GRACE uses three adaptation levels, exploiting the natural frame boundaries in periodic real-time multimedia applications (Figure 1 [24]). *Global* adaptation considers all applications and system layers together, but only occurs at large system changes (e.g., application entry or exit). *Per-application* adaptation (or *per-app*) considers one application at a time and is invoked every frame, adapting all system layers to that application's current demands. *Internal* adaptation adapts only a single system layer (possibly considering several applications) and may be invoked several times per application frame. All adaptation levels are tightly coupled by ensuring that the limited-scope adaptations respect the resource allocations made by global adaptation. The different adaptation levels may or may not consider the same adap-

*This work is supported in part by the National Science Foundation under Grant No. CCR-0205638 and a gift from Texas Instruments.

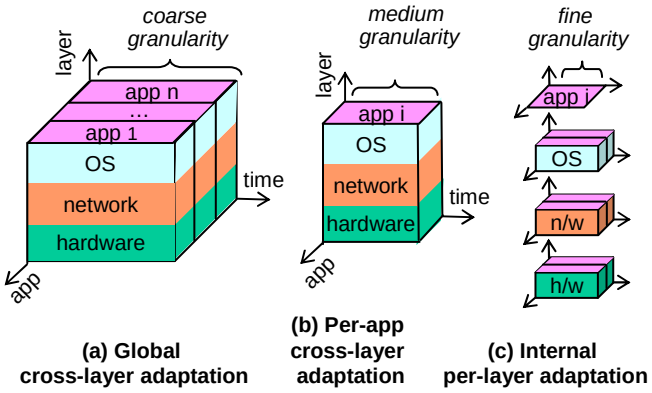


Figure 1. GRACE adaptation hierarchy. (We do not yet adapt the network.)

tations; they are distinguished by the granularity at which they consider an adaptation (e.g., both global and per-app levels may consider dynamic voltage and frequency scaling or DVFS for CPU adaptation).

We previously reported on the first GRACE prototype, GRACE-1, with adaptations in the CPU (DVFS), application (frame rate and dithering), and soft real-time scheduler (CPU time allocation) [30, 31]. GRACE-1's focus was on cross-layer *global* adaptation, for which it showed significant energy benefits. It reported a few experiments with hierarchical adaptation in the CPU and scheduler, but showed only modest benefits over global adaptation

This work focuses on the benefits of hierarchical adaptation in a mobile multimedia system, and reports results from the second generation prototype, GRACE-2. Our main contribution is to show that *per-app application adaptation provides significant benefits over and above global adaptation when network bandwidth is constrained*. These benefits occur with and without per-app CPU adaptation. Notably, the benefits with both per-app application and per-app CPU adaptation are often more than additive. In contrast, GRACE-1 neither provided per-app application adaptation nor implemented a network constraint, and is thus unable to obtain GRACE-2's benefits. Further, GRACE-1's hierarchical adaptation had to be redesigned to incorporate per-app application adaptation because it implicitly assumed a fixed application configuration between global adaptations.

GRACE-2 is implemented on a Pentium M based laptop running Linux 2.6.8-1. As illustrated in Table 1, GRACE-2 implements global adaptations in the CPU, application, and soft real-time scheduler; per-app adaptation in the CPU and application; and internal adaptation in the scheduler. It respects the constraints of CPU utilization and network bandwidth, while minimizing CPU and network transmission energy. All aspects of the system are fully implemented except for network communication. We report both the measured energy savings for the entire system and modeled energy

Objective: Minimize CPU and network transmission energy
Constraints: CPU time, network bandwidth

Layer	Adaptation	Hierarchy level		
		Global	Per-app	Int.
CPU	Dynamic voltage and frequency scaling (DVFS)	yes	yes	no
Application	Drop DCT and motion estimation computations based on adaptive thresholds	yes	yes	no
Scheduler	Change CPU time, network bandwidth budget	yes	no	yes

Table 1. Adaptations supported in GRACE-2

savings for just the CPU and network (we could not isolate the CPU energy through measurements).

We emphasize that the individual adaptations in GRACE-2 are not our focus, and have been previously proposed. Our focus is on their hierarchical control, and specifically on per-app application adaptation.

To our knowledge, this work is the first to demonstrate the benefits (energy savings) from per-app application adaptation over and above global adaptation. It is also the first to demonstrate significant benefits from hierarchical adaptation on a real multimedia system implementing multiple applications, adaptations, and constraints. Section 6 further discusses related work.

2 Layer Adaptations and Models

2.1 CPU

Adaptations: We study dynamic voltage and frequency scaling (DVFS). Our Pentium M CPU supports five frequencies {600, 800, 1000, 1200, 1300 MHz} and corresponding voltages {956, 1260, 1292, 1356, 1388 mV} [13].

To partially alleviate the limitations of the small number of discrete DVFS points supported, we emulate a continuous set of DVFS points as follows [14]. If we need to run at an unsupported frequency, f, we run at the supported frequency just below f (say f_l) for some number of cycles (say c_l) and the supported frequency just above f (say f_h) for the remaining cycles (say c_h). If c cycles need to be executed, then $c_l + c_h = c$ and $\frac{c}{f} = \frac{c_l}{f_l} + \frac{c_h}{f_h}$

Energy model: We report energy measurements from the actual system. However, we could not isolate the CPU energy from the rest of the measured system energy. To better understand the impact of our adaptations on the CPU

Bandwidth (Mbps)	2	5.5	11
Energy per byte (e^{-6} J)	4	2	.08

Table 2. Network bandwidth and energy/byte.

energy and to provide a CPU energy model to the adaptation control algorithms, we use the following: Energy = Power $\times$ Execution Time, where we approximate power at frequency f and voltage V by dynamic power $\propto V^2 \times f$.

We derive the proportionality constant using published numbers for the maximum Pentium M power. The above model does not incorporate leakage (static) power or the effect of application-specific clock gating (as is the case in much of the DVFS literature). These are difficult to incorporate analytically and do not affect the overall trends in the impact of per-app adaptation. This is substantiated by our measured (entire system) energy numbers which do include all effects.

It is noteworthy that CMOS technology is currently in the realm where frequency reductions result in sub-linear voltage reductions. Thus, while previously frequency reductions resulted in quadratic energy reductions (due to linear voltage reductions), this is no longer the case.

2.2 Network (non-adaptive)

We assume a non-adaptive (simulated) network layer with fixed available bandwidth. We model network transmission energy using a fixed energy/byte cost: Network Energy = EnergyPerByte $\times$ BytesTransmitted [4]. Table 2 summarizes energy per byte for different bandwidth values in an IEEE 802.11b wireless network, based on the energy consumption of a Cisco Aironet 350 series PC card [4].

We use different bandwidth values to model different constraints in the system. If the value selected is between two values in Table 2 (possible since not all the bandwidth of the channel is available to one node), we assume transmission cost of the higher bandwidth. We believe our network configurations represent reasonable scenarios seen in practice. Responding to variations in network bandwidth with an adaptive network layer is part of our ongoing work.

2.3 Applications

We consider periodic soft real-time applications or tasks. An application releases a job or a *frame* at the end of each period. We study workloads consisting of various combinations of speech and video encoders and decoders (Section 4). Our H.263 video encoder is adaptive while the other applications are non-adaptive.

Adaptations in the H.263 video encoder: We use the adaptations proposed in [25] (in the context of a system with a single application, and without global adaptation). Since these are not our focus, we only summarize them next and refer to [25] for details.

The adaptations trade off CPU computation (i.e., CPU energy) for the number of bytes transmitted (i.e., network transmission energy), to minimize the total CPU+network transmission energy. The appropriate trade off varies dynamically, depending on the video stream, the system load, and the ratio of network energy per byte to CPU energy per cycle (which depends on the chosen CPU frequency).

The adaptations work at the granularity of a single video frame. They enable dropping certain DCT (discrete cosine transform) computations and motion searches based on a threshold (set by the adaptation control algorithm) for the corresponding frame. The net effect is that, by changing the thresholds, the control algorithm can vary the bit rate and the computation cycles for a frame by about a factor of two. These adaptations can potentially reduce the PSNR (pseudo signal to noise ratio) of the stream, but this is compensated for by adjusting the quantizer step size. Thus, the adaptive encoder can be scaled between a highly compute-intensive but lower bit rate configuration to a less compute-intensive higher bit rate configuration, *without affecting the quality of the decoded video*.

We study four DCT and four motion-search thresholds, resulting in a configuration space of sixteen different encoder configurations.

Deadline misses and frame drops: A frame that does not complete computation or transmission of all its bytes by the end of the ensuing period is said to miss its deadline, with one exception. For video encoders, if a frame finishes its computation within 1ms of its period, we do not count it as a miss. We find these delays do not accumulate (the misses are not clustered). If the video encoder misses its deadline for one frame, the encoding/transmission for that frame continues in the next period, borrowing from the budget of the next frame. If it misses the deadline for two frames in a row, then the next frame is entirely dropped (i.e., incurs no computation or network transmission), enabling the encoder to catch up on its previous frame overruns. We have not (yet) modified the other applications to drop frames.

Since we use soft real-time applications, we assume that we may miss the deadline for or drop a total of up to 5% of all frames, without affecting quality. Although strictly speaking, missing a deadline by a small interval and dropping an entire frame have different effects on quality, we do not distinguish between the two and seek to limit both of these effects to a total of 5%.

2.4 OS Scheduler

We assume an earliest-deadline-first (EDF) soft real-time scheduler for CPU time and network bandwidth. The sched-

uler is responsible for enforcing budget allocations for both CPU time and network bandwidth. To reduce deadline misses due to imperfect predictions of resource demands, the scheduler performs an internal adaptation called budget sharing [2]. Briefly, this allows an application to reclaim unused budget from previous applications' underruns. The EDF CPU scheduler maintains a record of all unused budgets and their expiration times (i.e., the deadline for the job that released the budget). When an application is scheduled, the scheduler first tries to exhaust any unused budget before charging the elapsed cycles to the application. The unused budget can be given to an application only if the expiration time of the budget is less than the deadline for the application [2]. We similarly exploit network bandwidth sharing between applications. Unless stated otherwise, *budget sharing is used in all systems studied here.*

3 Adaptation Control Algorithms

3.1 Global Control

Overview: We use a global control algorithm similar to that in [31], but extended to incorporate a network bandwidth constraint. The algorithm is invoked on large changes in the system; e.g., when an application enters or exits. As input, the algorithm receives the resource requirements (CPU utilization, network bandwidth, CPU+network energy) for each combination of application and CPU configuration. The algorithm must then choose, for each application, the combination of the application and CPU configuration such that (i) the total CPU+network energy is minimized, and (ii) the resource requirements for all the applications (running with the chosen configurations) are met.

More formally, for application i, let Period_i be its period and C_i be a chosen CPU and application configuration combination. Let Energy_{i,C_i} be the energy consumed, Time_{i,C_i} be the CPU time taken, and Bytes_{i,C_i} be the network bytes required by a frame of application i with configuration C_i. Let there be a total of N_{apps} applications in the system and let B be the total network bandwidth (assumed to be fixed). Then the global algorithm must choose the CPU and application configuration C_i for each application i to:

$$\text{minimize} \quad \sum_{i=1}^{N_{apps}} \text{Energy}_{i,C_i}$$

subject to EDF scheduling and bandwidth constraints:

$$\sum_{i=1}^{N_{apps}} \frac{\text{Time}_{i,C_i}}{\text{Period}_i} \leq 1 \quad \text{and} \quad \sum_{i=1}^{N_{apps}} \frac{\text{Bytes}_{i,C_i}}{\text{Period}_i} \leq B$$

Solving the optimization: The above optimization problem is a multi-dimensional multiple-choice knapsack problem (MMKP) [17] and is known to be NP-hard. For the purpose of determining energy savings, we solve this problem using a brute force exhaustive search approach (with one modification below), to give global control the best showing. This approach is impractically expensive for a real system. When reporting the overhead for global, we use a more practical, but possibly sub-optimal heuristic approach based on Lagrangian techniques [17]. (We found the energy savings of both approaches to be comparable for the scenarios studied here.)

To reduce the complexity of both solution approaches, we choose the same frequency (CPU configuration) for all applications. We justify this heuristic by Jensen's inequality [15]: if the CPU energy per unit time is a convex function of frequency, then the best frequency setting is a single point for all applications (if the CPU does not support this single point, then a combination of adjacent supported frequencies is best). This optimization enables us to solve the MMKP problem separately for each supported frequency. We then pick the frequency that provides the minimum energy with the chosen application configurations at that frequency.

After the above process, it is possible that the chosen application configurations and frequency do not exhaust all the CPU utilization and network bandwidth. In that case, the leftover resources are divided among the applications in proportion to their current allocation. This leftover CPU utilization allows a further reduction in frequency. If the resulting frequency is not directly supported, the continuous DVFS emulation discussed in Section 2.1 is used.

Predicting resource requirements: The global algorithm requires predicted resource usage of a frame (Energy_{i,C_i}, Time_{i,C_i}, and Bytes_{i,C_i} in the optimization equations). These predictions must be representative of all frames until the next global adaptation is invoked. Following previous work on resource allocation and scheduling for soft real-time multimedia applications [3, 31], we use profiling of several frames to determine the resource usage. (In our experiments, since our streams are relatively short and since we would like to give global the best showing, we profiled the entire stream off-line.)

To reduce the amount of profiling, we leverage findings from [12]. Specifically, for our applications, the number of execution cycles for a given frame for a given application configuration is roughly independent of frequency; therefore, execution time scales roughly linearly with frequency.[1] Thus, by profiling each application configuration at a single CPU frequency, we are able to estimate the execution time (and the number of bytes) at all frequencies. These estimates also allow estimation of energy using the models in Section 2.

Since we assume a 5% deadline miss rate is acceptable,

[1] This is because these applications generally hit in the cache and do not see much memory stall time [12].

we use the execution time (and bytes) from the frame that falls in the 95th percentile of all profiled frames. For energy, we are concerned with minimization and not meeting a constraint. We therefore use the average time and bytes from the profiled frames as input to the energy models.

In practice, we expect to use on-line profiling of a few hundred frames [29]. For long streams, this poses a negligible overhead. In our experiments, since our streams are short and since we would like to give global the best showing, we profiled the entire stream off-line to determine the 95th percentile and average values.

3.2 Per-App Control

The per-app control algorithm (derived from [25]) is invoked at the start of a frame with the following inputs: (1) the resource allocation for the frame and (2) the resource requirements for the frame for each application configuration. The algorithm then simply chooses the application and CPU configuration combination that has the least energy, and whose CPU time and network bandwidth requirement is within its allocation. If such a combination is not found, then we use the application and CPU configuration of the last frame (likely leading to a deadline miss). The complexity of this algorithm is of the order of the product of the number of application and CPU configurations.

Predicting resource requirements: As for the global algorithm, estimating the execution cycles and bytes for a frame enables estimating all its resource requirements (execution time, bandwidth, and energy). Unlike global control, per-application control requires predicting resource usage for only the next frame.

For non-adaptive applications, we use a common history-based technique, where the average of the execution cycles and bytes in the last five frames is used to predict these quantities for the next frame. For the adaptive application, the history of the past frames may be for different application configurations, and cannot be used directly to predict the behavior of the next frame for yet other configurations. We therefore use an off-line profiling based prediction technique proposed by Sachs et al. as follows [25].

The technique generates an execution cycle predictor off-line by repeatedly encoding one or more sequences (for a fixed hardware frequency), randomly changing the encoder configuration at each frame. This off-line run generates several points for every pair of (previous, next) encoder configurations, mapping the number of cycles in the previous frame to the those in the next frame. The predictor is generated by fitting a function in the least-squared sense, for every pair of (previous, next) configurations. A byte count predictor is similarly generated. To avoid deadline misses, we conservatively add an adaptive leeway into the predicted values for both execution cycles and bytes. Improving the predictors for adaptive applications is part of our future work.

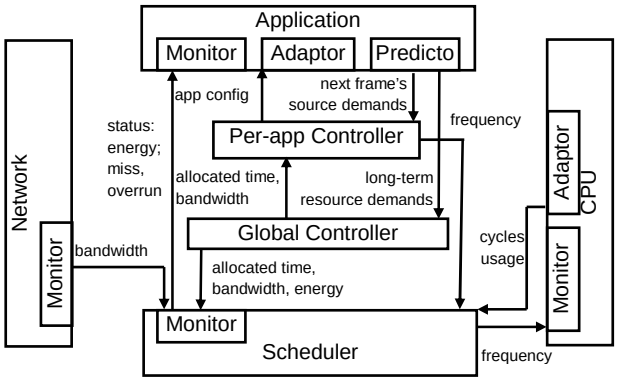


Figure 2. Integrated global and per-application control.

When the per-app adaptation is invoked, it determines the cycle count and byte count for each application configuration for the next frame by using the appropriate predictor, given the knowledge of the previous frame's application configuration, actual cycle count, and actual byte count.

3.3 Integrating Global and Per-App Control

A system that runs with only global control uses the frequency and application configurations as chosen by the global algorithm. In a system that additionally incorporates per-app control, the global algorithm's choice of configuration is only used to determine the resource allocation for each application. This resource allocation is fed as input to the per-app control algorithm. The latter then determines the appropriate configurations for the next frame based on its predictions of the resource usage of that frame and its allocation. Since the per-app controller makes a prediction only for the next frame, based on knowledge of all past frames, it is likely that its prediction is better than that of the global algorithm. Therefore, the per-app controller is likely to better utilize the resources that were allocated to its application by the global algorithm. Figure 2 summarizes the integrated system. As shown, the only interaction between the global and per-app controller is that the former gives the resource allocation to the latter.

4 Experimental Methodology

Implementation: We have implemented all aspects of the system studied except for the network communication (which is replaced with file I/O). Our implementation is on an IBM ThinkPad R40 laptop running the Linux kernel 2.6.8-1, and is described in detail in [27].

Energy measurement: We use an Agilent 66319D sampling power supply to measure the energy consumed by the

#	Applications	Inputs	Fps	Mbps
1	video (enc, enc)	foreman, buggy	30	2
2	video (enc, enc)	foreman, buggy	20	3.3
3	video (enc, dec)	carphone, paris	30	2
4	video (enc, dec) audio (enc, dec)	carphone, paris clinton, lpcqutfe	30 50	2.1
5	video (enc, dec, dec) audio (enc, dec, dec)	foreman, carphone, football female, clinton, male	30 50	6.7

Table 3. Workloads evaluated.

entire system. The measurements were done with the display brightness set to level 3 (0 is minimum). The wireless card was turned off, the laptop battery was removed, and the only applications running were from the experimental workload. All other parts of the system (e.g., hard drive) were on. The network energy used was calculated using the model in Section 2.2, and was added to the above measured energy to give the total system energy in Section 5.3.

Since we cannot isolate the CPU energy in our measurements and since the CPU and the network are the targets of our energy adaptations, our first set of results (Section 5.2) are based on modeled CPU (+network) energy, using the model in Section 2.1.

Workloads: We study various combinations of an H.263 video encoder and decoder and a speech encoder and decoder [28], representing workloads such as remote sensing and teleconferencing [27]. The video encoder is adaptive (Section 2.3) while the other applications are non-adaptive. We use standard video and audio input streams available on the Internet (QCIF size frames for the video encoder and CIF for the video decoder). To study the effect of different types of resource constraints (CPU load, network bandwidth), we run the workloads with different periods (frame rates) for the constituent applications and different values of the available network bandwidth. We studied 16 different workloads covering four resource constraint scenarios: unconstrained, only CPU constrained, only network constrained, and both CPU and network constrained [27]. For space, here we report detailed results from the (most significant) last two scenarios (i.e., those with a network constraint), and summarize the rest. We choose five representative workloads for the network constrained scenarios, summarized in Table 3. Each run includes between 150 to 500 frames for each application.

5 Results

5.1 Overheads

A detailed discussion of experiments showing the overheads of global and per-app adaptation appears in [27], here we briefly summarize our observations. As expected, global adaptation is significantly more expensive than per-app adaptation. For example, in one case, global adaptation took about 4% of a video encoder's average frame computation time, without including the overhead for on-line profiling for predicting the application's resource usage. In contrast, the per-app adaptation overhead was 8X lower. Further, as the number of possible adaptive layers, adaptive components within each layer, and the number of adaptive states within each component increases, the overhead of global will increase much faster than per-app.

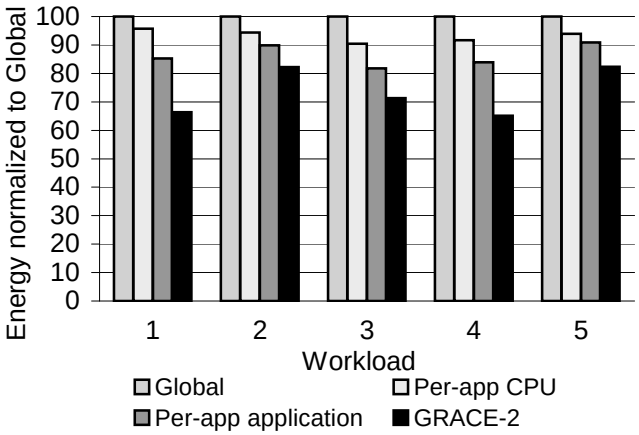


Figure 3. CPU+network energy benefits from per-app application adaptation. For each workload, the leftmost bar shows energy for a system with global adaptation in the CPU, application, and scheduler. The next three bars include this global adaptation as well as per-app CPU adaptation, per-app application adaptation, and both per-app CPU and per-app application adaptation (i.e., GRACE-2) respectively. The energy for each system is normalized to the system with only global adaptation (leftmost bar).

5.2 CPU and Network Energy Savings

Figure 3 illustrates the energy benefits in the CPU-network subsystem of per-app application adaptation. For each workload, the leftmost bar shows a system with global adaptation in the application, CPU, and scheduler. The next three bars shows systems that incorporate this global adaptation and additionally have per-app CPU adaptation (second bar), per-app application adaptation (third bar), and both per-app application and per-app CPU adaptation (the last bar, which represents GRACE-2). The energy of all systems is normalized to that consumed by the system with only global adaptation (the first bar).

Figure 3 shows that adding per-app application adaptation to a system with global adaptation can result in significant energy benefits. The benefits remain significant regardless of whether the base global system contains per-app CPU adaptation (second bar) or not. Relative to a system with only global adaptation, the energy savings from adding per-app application adaptation range from 9% to 18% with an average of 14% for the cases shown here. Relative to a system with both global and per-app CPU adaptation, the energy savings from adding per-app application adaptation range from 12% to 31% with an average of 21%.

It is noteworthy that adding only per-app CPU adaptation to global adaptation gives modest benefits.[2] In contrast, combining CPU and application adaptation at the per-app level gives more than additive benefits in some cases, resulting in quite significant overall savings of hierarchical adaptation relative to a system with only global adaptation.

Experiments from scenarios without a network constraint showed no benefit from per-app application adaptation and modest benefit (6% average) from per-app CPU adaptation, relative to global adaptation [27]. For reference, we also note that global adaptation gave significant benefits (41% average) relative to the non-adaptive system [27].

Analysis: A detailed analysis and supporting data to explain why per-app application adaptation shows significant benefits for the network constrained case and not for other cases appears in [27]. Briefly, for the specific case of our laptop based system, CPU energy dominates over the network energy. Therefore, the application configuration with the least computation is typically the most energy efficient. However, in a network bandwidth constrained scenario, there may be some frames for which this configuration produces too many bytes. Since the global-only system must pick a configuration safe for most frames, it cannot pick this configuration. GRACE-2's frame by frame adaptation, however, is able to pick this configuration for all the frames that produce bytes within the bandwidth constraint, thereby resulting in energy savings.

5.3 System-Wide Energy Savings

We next discuss (measured) system-wide energy savings of GRACE-2 over a system with only global adaptation.[3] Across all workloads in the scenarios with a network constraint reported in [27], we found that GRACE-2's per-app adaptation provides a system-wide energy benefit of 7% to 14% with an average of 10% (relative to only global adaptation). These savings are significant, considering that they are for the entire system including the display, disk, power-supply loss, and memory system; they are actual measured values; and they come from only adaptation of the CPU and application. (As reference, the one workload with multiple applications reported for GRACE-1 showed system-wide savings from hierarchical adaptation of only 3.8%, relative to global adaptation [30].)

[2] The benefits from CPU adaptation are modest relative to those seen for DVFS in much prior work due to the sub-linear relationship between frequency and voltage reductions in recent processors (Section 2.1).

[3] As explained, the network energy is realistically modeled, but is a very small part of the system energy [27].

5.4 Deadline Misses and Budget Sharing

The main benefit of budget sharing (i.e., the internal scheduler adaptation described in Section 2.4) is in reducing the number of deadline misses (including frame drops). Budget sharing has negligible ($< 1\%$) effect on energy. GRACE-2 shows acceptable deadline misses (within 5%) for each application in each scenario/workload studied. Without budget sharing, the deadline miss ratios are high (up to 23%) for several cases. Thus, budget sharing is effective and critical for our system.

6 Related Work

There has been a large amount of work on energy and bandwidth driven adaptations and resource allocation that is relevant to this work. This includes CPU adaptation with and without coordination with a real-time scheduler (e.g., [1, 8, 19, 20, 22, 26, 32]), adaptation of one or more applications with and without OS/middleware support (e.g., [6, 7, 9, 10, 16, 18, 21]), and single-layer or cross-layer adaptation or resource allocation with only global control supporting multiple applications (e.g., [11, 33, 23]) or only per-app control supporting a single application (e.g., [25]). The focus of this work, however, is on hierarchical adaptation control in a cross-layer adaptive system, and more specifically on fine-grained (per-app) application adaptation. None of the above systems exhibit this property.

The systems most closely related to the hierarchical adaptation of GRACE-2 are GRACE-1 [30, 31] which has already been discussed and Fugue [5]. Fugue proposed adaptation at multiple time scales for wireless video [5]. This is one of the key features of GRACE-2's hierarchical control. However, Fugue differs from GRACE-2 in the following important ways. First, it considers only one application running. Second, it is based on the insight that different types of adaptations work on different time scales; e.g., application quality control must occur at a coarser time scale than network transmission power control. GRACE-2's global and per-app controllers consider the same set of adaptations, but for different purposes – the former uses them for resource allocation among multiple applications while the latter does the actual adaptation. Incorporating adaptations that inherently work at different time scales can

be viewed as an orthogonal issue – our system incorporates these as well, but that is not the focus of this work.

7 Conclusions

The GRACE project balances the scope and frequency of energy saving adaptations in multiple layers through a hierarchical approach, where expensive and infrequent global adaptation allocates resources among applications based on long-term predictions, and inexpensive per-application control seeks to make the energy-optimal use of these resources through localized short-term predictions and cross-layer adaptations.

This paper presents results from the second generation prototype, GRACE-2. Specifically, it shows that per-app application adaptation provides significant benefits over and above global adaptation when the network bandwidth is constrained. These benefits are seen both with and without per-app CPU adaptation. For example, the energy savings in the CPU+network from adding per-app application adaptation to a system with global adaptation and per-app CPU adaptation were seen to be up to 31% (average 23%). Interestingly, when both per-app CPU and per-app application adaptation are added to a system with global adaptation, the combined benefits are more than additive.

To our knowledge, this work is the first to demonstrate the benefits from per-app application adaptation control over and above global control. It is also the first to demonstrate significant benefits from hierarchical adaptation on a real multimedia system implementing multiple applications, adaptations, and constraints. Given the low overhead of per-app control and the relatively low added system implementation complexity over a system with global control, the benefits achieved seem worthwhile to exploit.

Our ongoing work is incorporating an adaptive network layer that responds to variations in network bandwidth, and is also exploring other possible application adaptations including those that affect user perception.

References

[1] H. Aydin et al. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *RTSS*, 2001.

[2] M. Caccamo et al. Capacity sharing for overrun control. In *RTSS*, Dec. 2000.

[3] H. H. Chu and K. Nahrstedt. CPU service classes for multimedia applications. In *ICMCS*, 1999.

[4] Cisco Aironet 350 Series Client Adapters Datasheet. http://www.cisco.com/en/US/products/hw/wireless/ps4555/products_data_sheet09186a0080088828.html, 2004.

[5] M. Corner et al. Fugue: time scales of adaptation in mobile video. In *MMCN*, Jan. 2001.

[6] E. de Lara et al. HATS: hierarchical adaptive transmission scheduling for multi-application adaptation. In *MMCN*, 2002.

[7] C. Efstratiou et al. A platform supporting coordinated adaptation in mobile systems. In *WMCSA*, June 2003.

[8] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *OSDI*, Dec. 2002.

[9] J. Flinn et al. Reducing the energy usage of office applications. In *Proc. of Middleware*, Nov. 2001.

[10] J. Flinn and M. Satyanarayanan. PowerScope: A tool for proling the energy usage of mobile applications. In *WMCSA*, 1999.

[11] K. Gopalan and T. Chiueh. Multi-resource allocation and scheduling for periodic soft real-time applications. In *MMCN*, Jan. 2002.

[12] C. J. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[13] Intel Pentium M Processor Datasheet. http://www.intel.com/design/mobile/datashts/25261203.pdf, 2003.

[14] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *ISLPED*, 1998.

[15] S. Krantz, S. Kress, and R. Kress. *Jensen's Inequality*. Birkhauser, 1999.

[16] M. Mesarina and Y. Turner. Reduced energy decoding of MPEG streams. In *MMCN*, Jan. 2002.

[17] M. Moser et al. An algorithm for the multidimensional multiple-choice knapsack problem. In *IEICE Trans. Fundamentals of Electronics*, 1997.

[18] B. Noble et al. Agile application-aware adaptation for mobility. In *SOSP*, Dec. 1997.

[19] T. Pering et al. Voltage scheduling in the lpARM microprocessor system. In *ISLPED*, July 2000.

[20] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[21] C. Poellabauer et al. Cooperative run-time management of adaptive applications and distributed resources. In *Proc. 10th ACM Multimedia Conf.*, Dec. 2002.

[22] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *DAC*, 2001.

[23] C. Rusu et al. Maximizing the system value while satisfying time and energy constraints. In *RTSS*, Dec. 2002.

[24] Sachs et al. GRACE: A Cross-Layer Adaptation Framework for Saving Energy. *SIDEBAR in IEEE Computer*, dec 2003.

[25] D. Sachs et al. Adaptive video encoding to reduce energy on general-purpose processors. In *ICIP*, Sept. 2003.

[26] T. Simunic et al. Dynamic voltage scaling and power management for portable systems. In *DAC*, 2001.

[27] V. Vardhan et al. Integrating Fine-Grained Application Adaptation with Global Adaptation for Saving Energy (extended version). Technical report, UIUC, 2005. http://www.cs.uiuc.edu/grace/papers/perapp-tr.pdf.

[28] Xiph.org. Speex. http://www.speex.org/, 2003.

[29] W. Yuan. GRACE-OS: An Energy-Efficient Mobile Multimedia Operating System. PhD thesis, UIUC, 2004.

[30] W. Yuan et al. GRACE: Cross-Layer Adaptation for Multimedia Quality and Battery Energy. *IEEE Trans. Mobile Computing*. Accepted for publication.

[31] W. Yuan et al. Design and evaluation of cross-layer adaptation framework for mobile multimedia systems. In *MMCN*, 2003.

[32] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP*, 2003.

[33] H. Zeng et al. Ecosystem: Managing enegy as a first class operating system resource. In *ASPLOS-X*, 2002.

Power Management and Dynamic Voltage Scaling: Myths and Facts

David Snowdon, Sergio Ruocco and Gernot Heiser

National ICT Australia*
and
School of Computer Science and Engineering
University of NSW, Sydney 2052, Australia
Firstname.Lastname@nicta.com.au

Abstract

This paper investigates the validity of common approaches to power management based on dynamic voltage scaling (DVS). Using instrumented hardware and appropriate operating-system support, we account separately for energy consumed by the processor and the memory system. We find that memory often contributes significantly to overall power consumption, which leads to a much more complex relationship between energy consumption and core voltage and frequency than is frequently assumed. As a consequence, we find that the voltage and frequency setting that minimises energy consumption is dependent on system characteristics, and, more importantly, on the application-specific balance of memory and CPU activity. The optimal setting of core voltage and frequency therefore requires either a-priori analysis of the application or, where this is not feasible, power monitoring at run time.

1 Introduction

Dynamic voltage scaling (DVS) is a standard technique for managing the power consumption of a system [22]. It is based on the fact that the *dynamic* (switching) power P of CMOS circuits is strongly dependent on the core voltage V and the clock frequency f according to

$$P \propto fV^2. \tag{1}$$

Under the assumption that the number of clock cycles required for a computation is independent of the core frequency, the execution time is inversely proportional to the frequency. The total energy E for the computation is then proportional to the square of the voltage:

$$E \propto V^2. \tag{2}$$

Note that the total energy for a computation does in this simple model not depend on the frequency, but a reduced core voltage requires a reduction of the clock frequency and therefore implies a longer overall execution time.

The assumptions behind Eqn. 2 are highly dubious, as they ignore other system components, in particular the front-side bus and memory [2]. Those other components impact the execution time of a program, leading to a much more complex dependence on the processor frequency. Furthermore, those components themselves consume energy, and that energy consumption scales differently than the processor's. While memory power may be dominated by CPU power in high-end systems, this is not the case for embedded systems using low-power processors. Finally, Eqn. 1 is not even necessarily a good model of the power consumption of a modern processor, as for modern CMOS circuits the *static* energy consumption can no longer be ignored.

This paper presents a measurement-based examination of the effect of DVS on the energy required to execute applications on a modern embedded system. We independently measure processor and memory power consumption on a representative platform, and find that the behaviour is quite different from what is expected by the simple model. As a consequence, we find that more sophisticated methods are required in order to manage limited energy resources well.

2 Related Work

There exists a large body of work on both dynamic and static voltage scaling [3, 5, 6, 13, 14, 18, 22]. Many of the

*National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

ideas developed by Weiser et al. [22] and Govil et al. [5] form the basis of these algorithms: that the CPU idle (slack) time should be minimised by slowing the CPU core frequency. This reduces the DVS problem to estimating the idle time.

Other studies have examined frequency and voltage scaling in time-sensitive systems [11, 19, 20, 24]. One approach is to use timing information which is available in real-time systems. This can allow static schedules to be developed such that processor utilisation is maximised (all deadlines are only just met). Modifications can be made to the schedule on-line in order to make use of slack-time made available by processes which complete before their deadlines.

Weissel and Bellosa [23] measured the effect of frequency scaling on the performance and total power consumption of an XScale-based computer running several benchmarks. They examined the number of memory references and instructions executed at runtime in order to determine the memory dependence of an application, and thus estimate its response to a reduction in CPU frequency (a memory-bound application will be limited by memory speed rather than CPU speed). They determine what CPU core frequency will result in a 10% or less reduction in performance for the process. No voltage scaling was used in this work. Choi et al [10] refined this work to allow a dynamic rather than static tradeoff between power and performance reduction by characterising process memory and CPU usage at run-time.

There are several previous studies of the power consumed by real computers. The most relevant is that of Miyoshi et al [16] who examine a set of microbenchmarks running on two different platforms. They find that in some cases the lowest-performance setting may not give the lowest total energy. They also provide a methodology for choosing which settings shoud be ignored.

Fan et al. [2] used a modified simulator to estimate the power consumed by an XScale-based device with power-aware SDRAM. They observe that, owing to a system's static power consumption (particularly owing to DRAM), the energy reduction via frequency scaling can be outweighed by the energy resulting from a longer execution time. Their results indicate that an aggressive memory power-down policy such as that which they had previously developed [1] can reduce this effect.

Martin [14] studied the effect of frequency scaling on battery lifetime, developing a system for identifying the CPU frequency at which the most computation could be performed using a single battery charge.

Flinn et al. [4] conducted a similar study of the ItSY pocket computer, using external power management and off-line evaluation. Micro-benchmarks were used to study the effect of frequency scaling on the processor's performance and power consumption. Voltage scaling was not examined.

3 Benchmarks

A number of benchmarks were used to represent typical workloads for a variety of embedded system. The majority of these were taken from the MiBench [7] suite, along with four others, also representing typical embedded applications, described in previous work [21]. Each benchmark in this collection represents a fixed amount of "work" for the system, therefore the total energy for each benchmark is directly comparable.

MiBench is a suite developed by the academic community with the explicit aim of representing embedded workloads. The particular benchmarks used were selected based on their resource requirements: many of the MiBench tests require large input data which could not be accommodated on the RAM disk of our disk-less system. The future addition of network and disk support to PLEB 2 should allow the full suite to be executed. Furthermore, benchmarks which ran for less than four seconds were excluded to avoid measuring start-up and wind-down energy.

All output was discarded to avoid filesystem overheads and resource constraints.

4 Experimental Platform

The experiments were performed on PLEB 2 [21], a power-aware computer based on ARM XScale processor running a standard Linux OS, augmented with current sensors to measure the power consumption of the CPU core, RAM, and I/O devices.

4.1 Hardware

PLEB 2 is a single-board computer based on the Intel XScale PXA255 [9]. The PXA255 was chosen as being representative of high-performance, low-power CPUs designed for use in embedded systems. It consists of a 400MHz ARMv5TE-compatible core combined with a set of on-chip peripheral units including memory, interrupt, DMA and LCD controllers.

The computer consists of the CPU, SDRAM and flash memory. The SDRAM is implemented using two Micron MT48LC16M16A2 ICs [15], and the flash is implemented using two Intel TE28F320 ICs [8]. Three switching power supplies generate core, memory and IO power. A minimal set of peripherals (infra-red, USB, and serial port) are provided on-board. An 8-bit microcontroller performs a super-

visory role. The PXA255, flash, SDRAM and the power supply represent the core of a typical embedded system.

Linux 2.4.19, Linux 2.6.8, L4ka::Pistachio [12], and Iguana [17] have been adapted to run on PLEB 2 hardware.

4.2 Power management features

PLEB 2 supports a number of power management features. Frequency/voltage scaling and low-power modes are software-managed throttling mechanisms of interest.

The PXA255 supports the frequency scaling of three main clocks:

- the CPU core (*core*);
- the PXBus (*pxbus*): an internal bus that links the CPU core, DMA/Bridge, memory controller and LCD controller;
- the memory clock (*memclk*): drives the memory and LCD controller.

While the hardware which controls the frequency settings will allow a large number of combinations of core, pxbus and memclk frequencies, only a subset of these will allow the system to operate correctly (i.e. within the upper and lower frequency limits for all components in the system). All of the valid setpoints are shown in in Table 1.

The power-supply chip used in PLEB 2 (Epson S1F81100) supports voltage scaling. The core (CPU) voltage can be varied in 0.1V increments. This voltage is set to the appropriate value as given in the PXA255 developer's manual [9].

The PXA255, SDRAM and flash memory all support low-power states. In these states, the devices have a reduced functionality, but use significantly less power. For the PXA255, there are several modes: run/turbo, idle, 33MHz idle and sleep. Run/turbo are active modes where the CPU is running. Turbo mode is a mechanism for performing fast frequency changes by synchronously switching a clock divider. Idle mode stops the CPU core clock but does not halt its generation, avoiding loss of state and supporting a fast recovery to run mode. 33MHz idle and sleep mode are progressively deeper sleep states that require longer recovery times.

The Micron SDRAM also supports low-power modes. While not being accessed, it maintains an active standby mode which, according to the datasheet [15], consumes a maximum of 132mW per chip (although the typical idle current has been measured to be much lower on PLEB 2). If the chip is put into power-down mode (data is retained, but the chip must be refreshed periodically) it consumes a maximum of 6.6mW.

The Intel Flash chips have very effective automatic power management: according to the datasheet [8] they use less than 1mW unless being read/written, and even less when in one of the available power-down mode. Since the power consumption of flash is very small compared to CPU and memory we ignore it in our discussion.

4.3 Measurement system

The computing core of PLEB 2 is supplied by three power supplies. These are dedicated to the CPU core (nominally 1.5V), memory bus devices (3.3V), and IO devices (3.3V). Each of these power supplies is instrumented using a small series resistor and an amplifier. The analogue-to-digital converter within the on-board microcontroller reads the resulting signal. The voltage on the supplies is assumed to be sufficiently constant as to not require measurement. The power can then be calculated via $P = IV$.

Data collected is transferred from the microcontroller to the PXA255 via an I2C bus. Statistical sampling is used to associate the power readings with the running processes. For each sample, a series of interrupts are generated to record which process the sample should be attributed to, and to start the transfer of data. The resulting information is made available at user level at run-time.

The overhead associated with handling the measurements on the PXA255 will introduce an error to the measurement of time, cache misses, writebacks, etc. This is because the CPU will spend some time handling the interrupts, and because the events associated with the interrupt handlers. This overhead varies between 2% and 10% of execution time depending on the nature of the application. Because the sampling rate is independent of the processor speed the overheads will vary. The measured power is not affected by the measurement system because the power samples are always taken when running the real system.

Further information regarding the measurement system, its overheads and validation is given in a previous publication [21].

Cache miss and write-back numbers were determined using the PXA255's performance monitoring unit and appropriate OS support.

All experiments were run on Linux 2.4.19, modified to provide memory and CPU energy accounting through the `/proc` file system, from where it can be accessed by a modified `time()` function.

5 Methodology

A number of operating points were chosen. Each operating point defines a specific hardware configuration under

V_{core} (V)	f_{cpu} (MHz)	f_{pxbus} (MHz)	f_{mem} (MHz)
1.0	99.5	50.0	99.5
1.0	199.1	99.5	99.5
1.1	298.6	99.5	99.5
1.3	398.1	99.5	99.5
1.3	398.1	199.5	99.5

Table 1. Hardware configurations under test

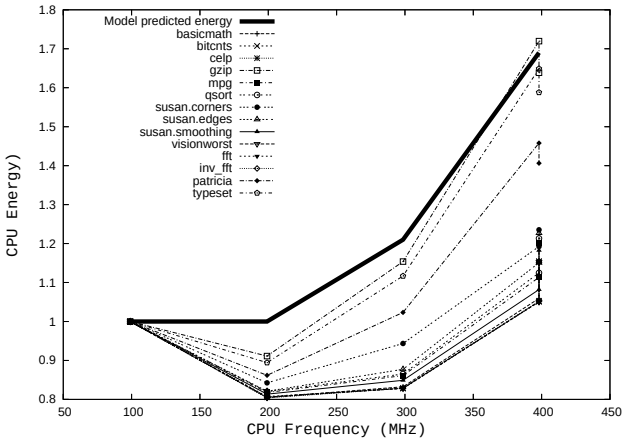


Figure 1. Normalised CPU energy

test. Five operating points (as shown in Table 1) were initially available as defined by the frequency-setting code in Linux — those where operating points where the memory frequency was 99.5MHz. By varying (and even overclocking) the memory frequency it would be possible obtain a larger number of operating points than those used.

The platform was configured according to each of the operating points and the benchmarks executed. The mean current was measured for the CPU and memory power supplies (since no devices are connected to the IO supply, that supply was deemed irrelevant) and recorded on the RAM disk. The results were later transferred to a PC for analysis. Each experiment was repeated 10 times and the results averaged, standard deviations were less than 1%.

6 Results

Fig. 1 shows the processor's energy consumption of the various benchmarks as a function of the core frequency, normalised to the energy consumption at the lowest clock rate (100MHz). There are two values at the highest frequency (400MHz) correspond with the two different processor bus frequencies used at that setting.

The bold line shows the prediction of Eqn. 2. Contrary to

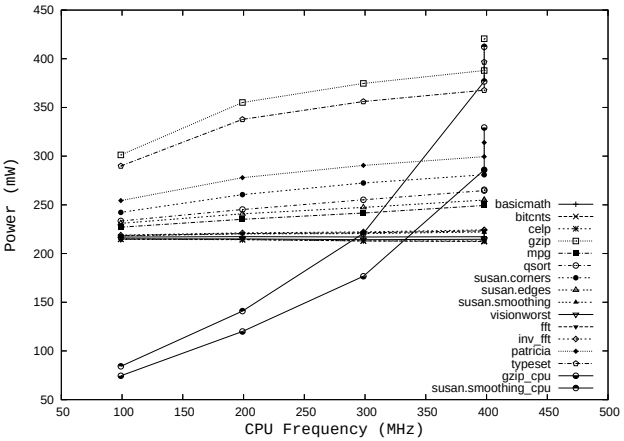


Figure 2. Memory power

naïve theory, we find that the energy is in fact quite dependent on the clock rate: Increasing the core frequency from 100MHz to 200MHz (at unchanged core voltage) results in a drop of total energy for all benchmarks. Further frequency increases lead to increases in energy, as they are accompanied by voltage scaling. The benchmarks consistently stay *below* the predictions of Eqn. 2 — in other words, Eqn. 2 *over-estimates* the benefit from DVS.

This effect can be explained with the processor's *static* power consumption, which is independent of the core frequency. As the runs with a faster clock require less total time, the total static energy consumption is less in those cases.

The two benchmarks whose energy, under frequency *and* voltage scaling (from 200 to 400MHz), grows steeper than the theory are `gzip` and `typeset`, which are both memory-limited while the others are CPU-limited. Memory is always clocked at the same rate, and therefore a memory-limited application's execution time benefits less from an increase in the core frequency than CPU-limited applications. Hence, for those benchmarks the influence of the static energy increases with increasing clock speed.

Memory behaves differently than the processor, and is best examined in the power, rather than the energy dimension. As Fig. 2 shows, most benchmarks consume very little memory power, and it is very weakly dependent on the core frequency. These benchmarks run essentially out of cache and cause very little memory traffic, so we see mainly the static energy of RAM, which is attributed to DRAM refreshes, leakage and powering input/output buffers. The memory-intensive benchmarks show a strong frequency dependence at lower clock rates, which then flattens out, a consequence of the saturation of the memory system in

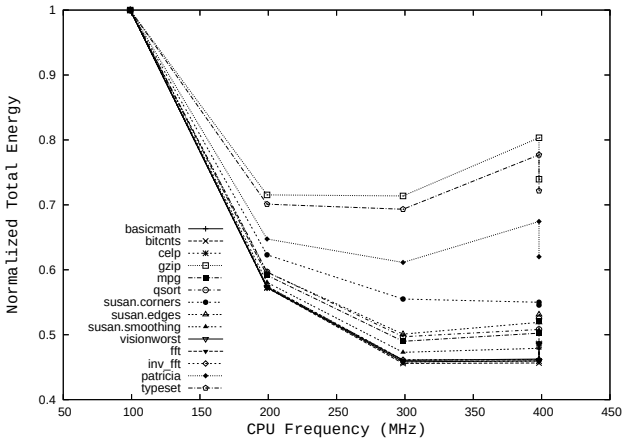


Figure 3. Normalised total energy

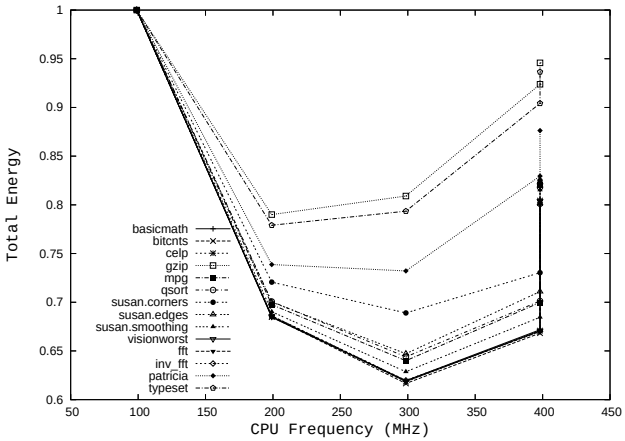


Figure 4. Normalised total energy padded to equal total time

those runs. Note also that the difference of the two data points at the highest frequency (corresponding to different bus frequencies, see Table 1) is highest for the memory-limited benchmarks.

The flat curves mean that memory power scales very little with core frequency. Translated into total energy (accounting for total execution time) this means that memory energy use is actually *lowest* at the *highest* clock rates.

Also shown in Fig. 2 (strongly rising curves) is the minimal and maximal CPU power consumed by the benchmarks. It can be seen that, compared to memory power, the range is relatively small, and except at the highest core frequency, CPU power is dominated by memory power.

This helps explain Fig. 3, which shows the total (CPU+memory) energy for the execution of the benchmarks. Inclusion of the memory energy leads to results which bear no similarity whatsoever with the model of Eqn. 2, and, contrary to folklore, shows that the highest clock rates actually *minimise* the energy requirements of the computations!

This result is somewhat misleading, however. The higher clock rates lead to faster completions of the runs. A more fair comparison of energy requirements needs to compare the energy used over the same total time period [2]. This means that the slack time remaining after an early completion results in an idle system, which still consumes power. We assume that the system, when idle, switches to the low-power idle mode from which it can be woken up quickly when an interrupt arrives (indicating a new computation task).

The result is shown in Fig. 4. We can clearly see that for all benchmarks the total energy is minimised at some intermediate frequency, neither the highest nor the lowest.

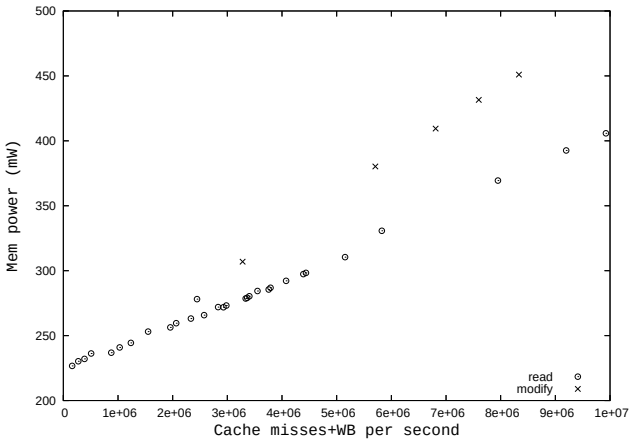


Figure 5. Memory power as a function of the cache miss rate

That frequency depends on the particular benchmark, it is lower for the memory-intensive than for the CPU-intensive benchmarks. The optimal frequency will obviously also depend on characteristics of the system, such as type and size of memory.

Weissel and Bellosa [23] model memory energy by using a performance counter to measure memory references, and assume that the energy cost of each memory reference is the same. Many systems (such as ours) do not provide such performance counters. One can attempt to approximate the number of memory references by the number of cache misses (for which the PXA255 has performance counters). Fig. 5 shows that this is inaccurate, as a single cache miss can produce either one or two memory references. For this figure we ran synthetic benchmarks which

in a tight loop contained load instructions (*read* case in the figure), or load instructions followed immediately by a store to the same memory location (*modify*). Varying numbers of nop instructions were inserted to vary the cache miss rate. We see that the modify case has a higher memory-energy cost per cache miss than the read, owing to the higher number of memory operations (write-back followed by a refill, compared to just the refill). A realistic load would lie somewhere in between those extremes, but it would be difficult to predict where. In addition, the presence of read and write buffers significantly complicates any modelling of memory traffic from cache miss rates.

7 Discussion

Our results show that traditional model of Eqn. 2 is not suitable for estimating the effect of DVS on modern processors, as it ignores the effect of static power, and grossly distorts reality. Furthermore, our measurements confirm that memory contributes significantly to the power consumption of embedded systems, and attempts to manage power without taking memory into account will likely lead to incorrect results.

Static power is also important for memory, and should be ideally be minimised by keeping as much RAM as possible in a low-power state. Furthermore, modelling dynamic memory power by measuring cache misses can produce misleading results, unless read and write misses can be measured separately (and even then it would be difficult to achieve good accuracy), owing to the complex memory-access patterns resulting from a processor which augments caches by read and write buffers.

Overall we find that the dependence of the energy cost of a computation on the processor core voltage and frequency is a complex function of system configuration and properties of the application, too complex to predict an energy-optimal operating point for DVS using simple models.

In some cases, the optimal operation may be determined by off-line measurements, but in general this is only possible if application loads are known well in advance. The only alternative is to determine the optimal voltage and frequency setting at run-time, based on the observation of the actual power consumption.

While we have shown how, with the help of some relatively simple instrumentation, such observation can be performed on off-the-shelf processors, this only provides the input data for successful power management. The required algorithms and policies remain the subject of future work.

References

[1] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for DRAM power management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2001.

[2] X. Fan, C. Ellis, and A. Lebeck. Synergy between power-aware memory and processor voltage scaling. In *Proceedings of the Proceedings of the Workshop on Power Aware Computer Systems (PACS)*. Springer-Verlag, December 2003.

[3] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic performance setting for dynamic voltage scaling. In *Mobile Computing and Networking*, pages 260–271, 2001.

[4] J. Flinn, K. Farkas, and J. Anderson. Power and energy characterization of the ItSY pocket computer (version 1.5). Technical Report TN-56, Western Research Laboratory, Compaq, 2000.

[5] K. Govil, E. Chan, and H. Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In *Mobile Computing and Networking*, pages 13–25, 1995.

[6] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 73–86, 2000.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[8] *Intel Advanced+ Boot Block Flash Memory (C3)*. http://www.intel.com/design/flcomp/products/b3_c3/techdocs.htm, May 2005.

[9] Intel PXA255 processor developer's manual. http://www.intel.com/design/pca/products/pxa255/techdocs.htm, 2005.

[10] R. S. Kihwan Choi and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, January 2005.

[11] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *Proceedings of the Symposium on Real-Time and Embedded Technology and Applications*, 2002.

[12] L4Ka Team. L4Ka::Pistachio kernel. http://l4ka.org/projects/pistachio/.

[13] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *SIGMETRICS/Performance*, pages 50–61, 2001.

[14] T. L. Martin. *Balancing Batteries, Power, and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2001.

[15] Micron Technology Inc. *Micron Synchronous DRAM Datasheet*, 2003.

[16] A. Miyoshi, C. Lefurgy, E. C. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the

runtime effects of frequency scaling. In *Proceedings of the 16th international conference on Supercomputing*, pages 35 – 44, 2002.

[17] National ICT Australia, http://ertos.nicta.com.au/Research/ESF/Iguana. *Iguana Embedded Operating System*.

[18] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the International Symposium on Low Power electronics and Design*, pages 76–81, 1998.

[19] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.

[20] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Proceedings of Design Automation Conference*, pages 438–443, 2001.

[21] D. C. Snowdon, S. M. Petters, and G. Heiser. Power measurement as the basis for power management. In *2005 WS Operat. System Platforms for Embedded Real-Time applications*, Palma, Mallorca, Jul 2005.

[22] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Operating Systems Design and Implementation*, pages 13–23, 1994.

[23] A. Weissel and F. Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the international conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES02)*, 2002.

[24] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Proceedings of the nineteenth ACM symposium on operating systems principles*, pages 149–163, Bolton Landing, NY, USA, 2003. ACM Press.

Profiles in Power: Optimizing Real-Time Systems for Power

Graham R. Hellestrand
g.hellestrand@vastsystems.com

Mahdi Seddighnezhad
m.seddighnezhad@vastsystems.com

James E. Brogan
j.brogan@vastsystems.com

VaST Systems Technology Corporation
1250 Oakmead Pkwy, Suite 310
Sunnyvale, CA 94085
+1-408-328-0949

ABSTRACT

High-performance, timing accurate models of complex systems (called Virtual System Prototypes (VSP)) enable the computation of relatively accurate power in terms of events that occur in the model. VSPs are the integrations of models of electronic hardware, communication and mechanical subsystems into systems that execute software accurately. Software has a first order impact on system performance and has, typically, the major effect on modern system optimization. The computation of relative power, although fundamental, is not useful by itself – doubling the *talk* time of a mobile phone is not useful if, concomitantly, the speed dwindles so that look-up functions take 20 seconds rather than the 2 seconds that competitors take. Power is an exemplar of the complex, *concept*-based functions, with many hardware, software and system parameters, that constitute optimization functions and will be treated in detail in this paper. The general form of a power computation function is given in the paper, as well as, a simple example of the implementation of a power calculator. The use of power, along with the other components of objective functions, such as speed (instructions per second), response latency and cost, must drive algorithm choice and software development in mobility and other power-performance sensitive applications. The use of VSPs is mandatory in specifying the hardware and software architectures of, and then building, complex optimal systems.

Categories and Subject Descriptors

C3 **[Special Purpose and Application Based Systems]** Real-time and embedded system; C4 **[Performance of Systems]** Measurement and modeling techniques; G3 **[Probability and Statistics]** Experimental design.

General Terms

Design, Experimentation, Measurement, Performance.

Keywords

Power measurement and analysis, quantitative systems architecture, empirical system design, event-based objective function, event data-driven optimization.

1. Background and Motivation

An empirical approach to composing optimal architectures for application specific embedded systems is relatively rare. The use of empiricism in developing optimal software is even rarer, and when used often primitive. The complexity of processor centric, electronic systems that control modern products (such as, cell phones, automobiles, communication base stations, consumer products) requires a systematic approach to developing software in order to deliver an optimal fit for an intended product. When a company's engineering process is being used as a competitive weapon, the luxury of optimality, especially wrt power and speed and response latency in mobile systems, becomes a necessity [1].

The bigger architecture picture is more complex. The intuitive optimization of systems – architecture, software design, hardware design, and interfaces – has largely been a by-product of hardware design. Since hardware designers have rarely understood, or had access to, the software that would run on their architectures, they produced conservative, often grossly over-engineered designs that were typically poor fits to a number of dimensions of the specification - especially in cost sensitive applications, where over-engineering is the antithesis of cost sensitivity [2].

The ability to support data-driven decision making early in the software development process is one of the underlying drivers of building models of systems that are timing accurate and high performance. Of course, this advantage also accrues to architecture development but that is another dialogue. From a purely software perspective, optimizing across the dimension of speed, response latency, cost (size) and power consumption is rarely done and, at a pre-silicon level, it is an undertaking only possible using high-performance, timing accurate models – called VSPs in this paper.

It is known that poor software and inefficient algorithms have a 1^{st} order effect on an embedded system's performance. This is difficult to reconcile with practice, when next-generation product planning has prime foci of processor microarchitecture and hardware (platform) design, regardless of the fact that iterative processor microarchitecture improvement typically yields a 2^{nd} or 3^{rd} order effect and hardware (platform) design has a 1^{st} order effect at the system level. The question is what happened to software? It is negligent not to employ empirical methods in the development of software in real-time, embedded systems.

For complex, multi-processor and networked real-time systems running full software loads (i) *advances in computer architecture and software have made it difficult or impossible to estimate or predict software's execution time* [8] and (ii) experience with proving the correctness of even a well constructed, small, single processor micro-kernel, L4, indicates

the overwhelming complexity of applying this technology to more complex systems [9]. This leaves schedulability analysis in the difficult position of having to use traces from simulation to determine whether a system will meets its overall real-time constraints. Deriving best-case, average and worst-case system performance from simulation using VSPs enables analyses of system variability leading to the identification of factors having the most significant impact on variability. These factors are prognostic as well as diagnostic and they can be used to drive the structural optimization of systems. On the deficit side, the number of simulation may require many experiments to be performed on various configurations of a system. However, here the design of experiment methodology [5] helps by providing a statistically valid mechanism for dramatically reducing the number of experiments needed to be performed.

Since VSPs are used to directly execute software, including hard real-time code, during development and debugging, trace information (streamed from non-perturbing probes inserted into the model) - including response latencies, power consumption, speed between markers, frequency of function calls, etc. - is produced alongside the usual debug data and hence is available to software and systems engineers as a normal part of the edit-compile-execute-debug software development cycle. This changes the perspective of where optimization should occur in the development cycle, and it is not as a post development exercise.

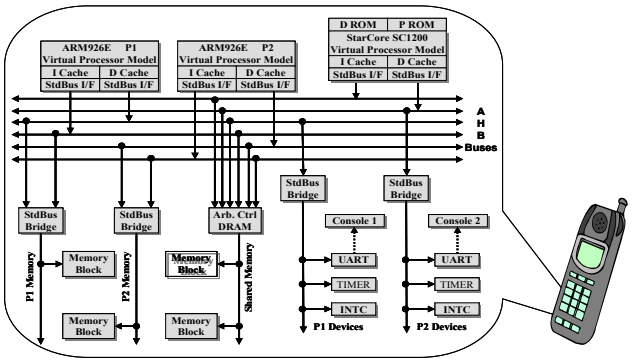


Figure 1: A Typical Virtual System Prototype for Mobility

Figure 1 shows the hardware platform component of a VSP, called a Virtual Prototype, used in the experiments in this paper.

2. Formulating Power

There are many ways of constructing objective functions including for power. The classical way is to track event frequencies and/or latencies and to construct the power function based on events that contribute significantly to the computation of power.

2.1 Event-Based Power Functions

In an event driven simulation environment, a general form of the power function can be expressed as a function whose parameters are functions each characterizing contributions to the objective function by one of the components constituting the system, viz. CPUs, buses, bus bridges, memories and peripheral devices. The parameter functions themselves have parameters that are functions of simulation event types sourced from the various event activities that occur in a VSP during simulation. In general, a power function will have the form shown in Equation 1 [3].

$$
\begin{aligned}
&Equation\,1:| \\
&F_{Power}(| f_{CPU}(\Theta_{cc=0..cn} \circ f_{CPU_{cc}}(\Theta_{CEvType=1..cet} \circ g_{CPU_{cc,CEvType}}(\Theta_{CEvCnt=scecn..tceen} \circ Event_{CPU_{cc,CEvType,CEvCnt}})), \\
&\quad f_{Bus}(\Theta_{bc=0..bcn} \circ f_{Bus_{bc}}(\Theta_{BEvType=1..bet} \circ g_{Bus_{bc,BEvType}}(\Theta_{BEvCnt=sbecn..tbecn} \circ Event_{Bus_{bc,BEvType,BEvCnt}})), \\
&\quad f_{BusBridge}(\Theta_{bbc=0..bbcn} \circ f_{BBus_{bbc}}(\Theta_{BBEvType=1..bbet} \circ g_{BBus_{bbc,BBEvType}}(\Theta_{BBEvCnt=sbbecn..tbbecn} \circ Event_{BBus_{bc,BBEvType,BBEvCnt}})), \\
&\quad f_{Mem}(\Theta_{mc=0..mcn} \circ f_{Mem_{mc}}(\Theta_{MEvType=1..met} \circ g_{Mem_{mc,MEvType}}(\Theta_{MEvCnt=smecn..tmecn} \circ Event_{Mem_{mc,MEvType,MEvCnt}})), \\
&\quad f_{Dev}(\Theta_{dc=0..cn} \circ f_{Dev_{dc}}(\Theta_{DEvType=1..\det} \circ g_{Dev_{dc,DEvType}}(\Theta_{DEvCnt=sdecn..tdecn} \circ Event_{Dev_{dc,DEvType,DEvCnt}})))
\end{aligned}
$$

$$
where\ \ f_{CPU_k}(\Theta_{EvType=1..et} \circ g_{CPUk,EvType}(\ldots)) = f_{CPU_k}(g_{CPU_k,1}(\ldots), g_{CPU_k,2}(\ldots)\ ,\ldots, g_{CPU_k,et}(\ldots))
$$

Table 1: Component Event Binding Table

Component Types Binding	Component Instance Binding	Component Event Binding
f_{CPU}	$f_{ARM1156T2F}$	$f_{SC1200_{TCM_{Write}}}$ $f_{SC1200_{Branch_{Taken}}}$ ..... $f_{SC1200_{LDD}}$
	f_{SC1200}	
	....	

A simple way to visualize and compute a power function is to build an interpretation table, as in Table 1. These tables are large and even though the *Event Bindings* are simple to implement, typically a pointer to a function and a history buffer of events, the extraction of appropriate data from register transfer (RT) models or representative samples of the silicon to put into the tables is not automatic and is difficult and time consuming.

3. Computing Power

We instrumented the VSP of Figure 1 and for the purposes of experiments for this paper put the 2nd ARM926 processor and the Starcore SC1200 processor in *Reset* – so they consumed no cycles and no power.

The basic function computed is Instant Power which calculates the total energy consumed over some period of time or some number of events (such as cycles).

The functions computed that are useful for optimization purposes are:

- Maximum power consumed, over a particular period (maximum of the instant powers)
- Average power consumed over the whole experiment.

A simplified function used to compute instant power per k-cycles is given in the Equation 2:

Similar functions occur for f_{Pipe}, f_{Cache}, f_{TLB}, f_{RegAcc}, f_{MemAcc}, $f_{PeriphAcc}$ and the weights for the constituent *accumulating* functions are given in Table 2, and the weights (W_i) for each of the classes of functions contributing to f_{Power} have been set to the constant function 1 in this study. In more complex studies, the *accumulating* function might be replaced with individual functions relevant to computing power in ways not considered for the simple examples of this paper. Such functions can include history and implementation dependent technology functions. Similarly, the weights (W_i) may be more complex functions – for example, the cache hit weights are functions of cache structure (size, wayness, policies).

4. Experimentation

The following is an outline of the experimental design process.

i. The goal of the 4 experiments reported here was to investigate the effects of various arrangements of cache, buses, memory hierarchy and algorithms on average power consumption and speed. The VSP used is that shown in Figure 1, but with only one ARM926E processor enabled. The target codes selected were MontaVista Linux v2.6, Viterbi and Sieve programs from the EEMBC [4] test suite, and a prime number program downloaded from the web [5]. Access to customer data was not possible for this study.

ii. To determine the goal, we specified, across the executed target codes:

- Power in terms of average power per instruction executed;
- Speed in terms of instructions executed per k-cycle (IPC_k);
- Cost – where cache size was used as a direct indicator of cost

The contributing factors (independent of target codes) to the computation of power were identified as events captured from the VSP. These events are delineated above in Equation 2 and Table 2. The computation of speed is a simpler function – the total number of instructions executed averaged across all cycles executed. This information is directly available from the simulation.

Table 2: Power: Function Types, Event & Weighting Functions

Function Types	Events	Weight Functions
Pipeline	ibase	6.0
Instruction Types	ijmp	2.0
	iexcept	2.0
	ictrl	0
	icoproc	12.0
	iundefs	0
	imemrd	0
	imemwt	0
	imemrw	0
	iarith	1.0
	iother	1.0
Caches (I&D)	Cache_lookup	$f_{i\text{-}dcache}$(size, ways)
	icache_hit	iCache-lookup + f_{icache}(line size, decode)
	icache_miss	Icache_lookup
	dcache_hit	Dcache_lookup + f_{dcache}(size, ways, line size,)
	dcache_miss	Dcache_lookup
	line_fill	0
TLB	tlb_miss	30.0
Register	regfile_access	1.0
Memory (incl. bus transactions)	membus_transaction	50.0
Periph Device (incl. bus transactions)	periphbus_reg_access	50.0

iii. In a simulation environment, all factors are effectively controllable. Therefore randomization of experiments will have no effect. However, sample size and selection – say the random selection of a number of the EEMBC [4] communications related programs – are indeed important parts of the experimental protocol. It is in this way that

$$
\begin{aligned}
&Equation\ 2: \\
&f_{Power} = .W_{Pipe} \times f_{Pipe} + W_{Instr} \times f_{Instr} + W_{Cache} \times f_{Cache} + W_{TLB} \times f_{TLB} + \\
&\quad W_{RegAcc} \times f_{RegAcc} + W_{MemAcc} \times f_{MemAcc} + W_{PeriphAcc} \times f_{PeriphAcc} \\
&where. \\
&f_{Instr} = .2 \times f_{Instr,jmp} + 2 \times f_{Instr,except} + 0 \times f_{Instr,ctrl} + 12 \times f_{Instr,coproc_{15}} + \\
&\quad 0 \times f_{Instr,LdSt} + f_{Instr,arith} + f_{Instr,other} \\
&and. \\
&f_{Instr,i} = .\sum (instructions\ of\ type_i\ in\ k-cycles)
\end{aligned}
$$

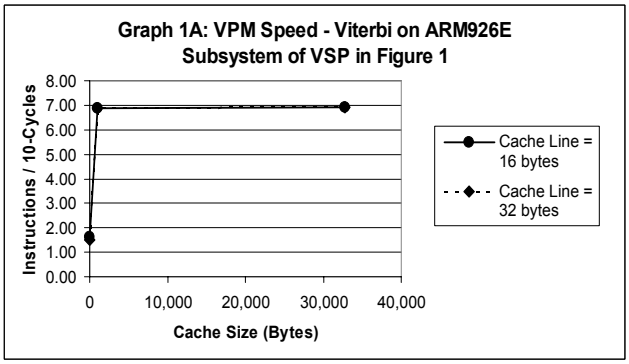
variability and variability optimization functions – such as minimization of variability – are addressed as part of the experimental procedure. In the latter characteristic, simulated systems and real systems are very similar.

iv. It then remains to determine which factors effect the power, speed and cost computations and what combination of factors produces optimal outcomes. In an industrial engineering set of experiments, we would want to determine whether the optimum we had achieve was local or whether a better result could be achieved and what factors can be adjusted to produce the better outcome.

Table 3: Factors Determining the Number of Experiments to be Peformed

Factors	Variants	Number of Variants	Number of Experiments
I-cache	Enabled, disabled	2	2
I-cache size	1k, 4k, 8k, 16k, 32k, 64k, 128k	7	7 * above = 14
I-cache Line Size	16B, 32B, 64B, 128B	4	4 * above = 56
D-cache	All variants – as for I cache	56	28 * above = 3,136
TLB	32, 64, 128 entries	3	3 * above = 9,408
I & D Bus Width	4B, 8B, 16B	3 * 3	9 * above = 84,672
I & D Memory	1st R/W = 4, 5, 6, 8 2nd R/W = 1, 2 (DDR, SDR)	2 * 4	8 * above = 677,376
Target code (programs)	Linux, Viterbi, Sieve	3	3 * above = 2,032,128
Event Weights in Table 2	Ibase, ijmp, …..	∞	∞
Etc.			

The design of experiments methodology relies on the ability to vary several variables in the system being observed in order to calculate the effects of the variables and the interactions between variables in terms of the objective functions. The prioritization of variables and interactions that cause the greatest effects gives us a handle by which to choose values of variables



that guarantee an optimal outcome. If there are no interaction effects between variables, the response of the objective function will be linear wrt the variables. Interaction effects produce higher-order polynomial responses.

Table 3 sets out the values of variables that can be set in experiments. It is impossible to perform but a small subset of the experiments in a reasonable amount of time given that simulation runs of 500 million cycles during a Linux boot might take a few minutes, in full data acquisition and profiling mode. Fortunately, nor is it necessary, the number of experiments can be reduced dramatically using fractional factorial designs in which the number of experiments is determined by the main effects and their interactions.

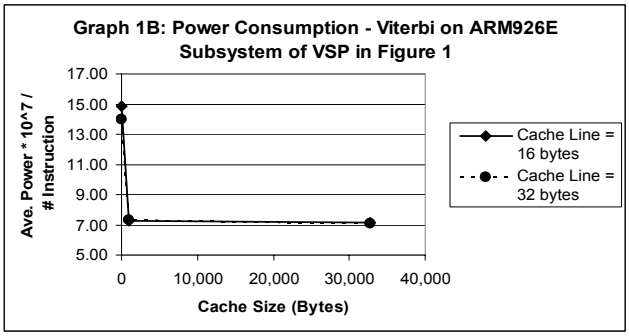
In our study, we ran exploratory experiments using Viterbi and Linux target code on many model variants and assessed the patterns of results in the light of analysis and expected behaviours. This preliminary investigation indicated that the important main effects were: I&D cache enabled/disabled; I&D cache size – 1k and 32k, cache line size – 16B & 32B, data rate of memory (DDR – double data rate, SDR – single data rate, and code), and target code. For simplicity here, we ignored interaction effects, even though to reach a global optimum they are likely to have an impact.

5. Experimental Results

We constructed 4 sets of experiments (58 in total) using various code running on the VaST ARM926E-based VSP subsystem with instruction and data buses bridged to a shared memory. The VSP subsystem was extensively parameterized and we used various configurations of cache and memory. For all experiments, the speed performance is calculated as instructions executed per 10 cycles (IPC_{10}) on the VSP (that is it is an index of VSP speed NOT processor speed) and power consumption is a relative measure of average power over all instructions executed.

5.1 Viterbi

The results from 7 Viterbi (calibration) experiments are expected, see Graphs 1A & 1B. Uncached performance is poor both in regard to power consumption and speed (IPC_{10}). With cache enabled, and even minimal cache (1,024 bytes) is sufficient, a good working set fit of Viterbi to cache was achieved. If the ARM926E was the selected controller



implementing an acoustic filter then a cache size of 1k bytes would be adequate. Since there is a better than 99.5% hit rate on the D-cache and I-cache, cache line size is immaterial as is bus width and memory type (either DDR or SDR). However, to minimize cost, SDR memory would be used instead of DDR.

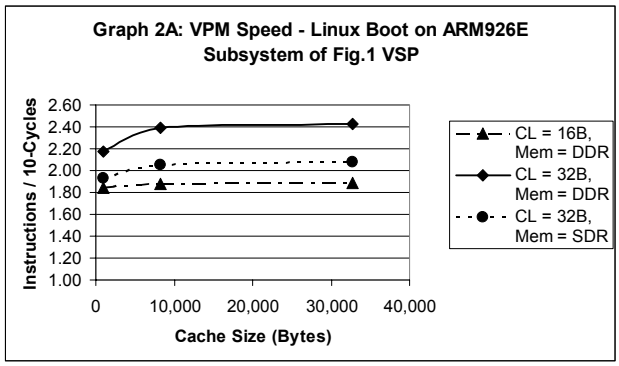
Optimizing Objective Functions*:*

Generalizing the results – for target code with a working set size that matches the cache size, cache size is the dominant determinant in optimizing speed and power consumption in the single processor VSP subsystem. When the optimum cache is

the smallest selectable, cost is also minimized with respect to this factor. Depending on the overall system objective function $f_{System}(Power, Speed, Cost)$ the selection of optimal sets of settings (the so-called optimal response contour) will be determined by the tradeoffs inherent in the objective function.

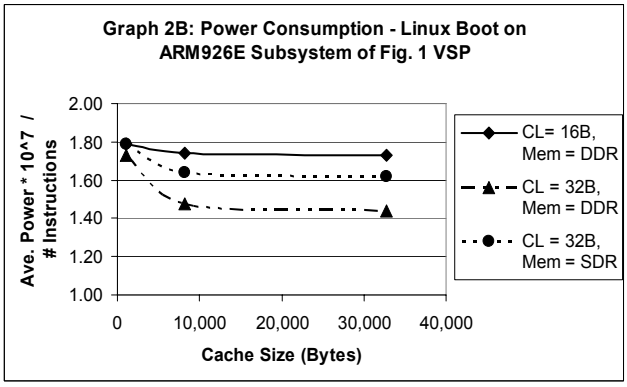
5.2 Linux Boot

The Speed (IPC_{10}) and relative Power Consumption of 9 structural variants of the experimental VSP were computed while booting Linux. The variants were selected from the full set of variants determined by - cache size: 1k, 8k, 32k; cache line: 16B, 32B; Mem configured as DDR (1st word delayed 5-cycles, 2nd word available per ½ cycle) and SDR (1st word delayed 5-cycles, 2nd word available per 1 cycle); bus data width



4bytes. The results are shown in the Graphs 2A & 2B.

The boot sequence of Linux spends more than 50% of its time executing with the ARM926E I&D caches disabled. Linux performs initialization of the cache after the Initial Program Load, kernel load and the device driver installations. Once the operating system has booted and the idle loop is executing, the behaviour of the ARM926E VSP is much the same as its

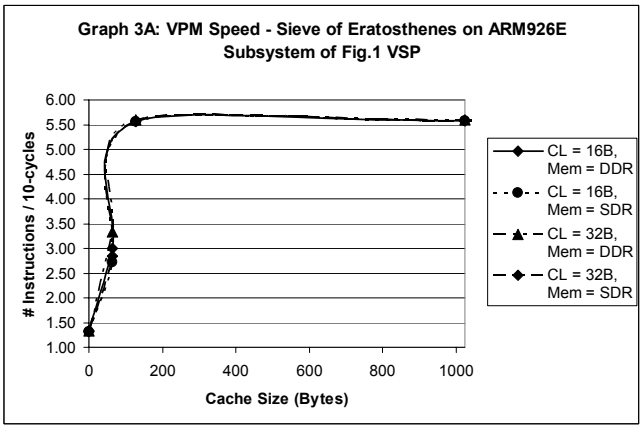


behaviour running Viterbi – that is the working-set size is compatible with any cache size. As is also expected, in an environment where the working set size of the target code greatly exceeds the cache size, the impact of the memory hierarchy on power and speed is considerable. For booting Linux, the settings of the ARM926E VSP subsystem: cache size (32 kbytes), cache line size (32bytes), and Memory (DDR) yield minimum power consumption and maximum VSP speed.

To mimimize cost, as well, a cache size (I&D) of 16 kbytes would proportionally reduce silicon cost by about 30% and adversely affect both power and speed by about 1%. To further optimize for cost, cache sizes of 8 kbytes will yield a further ~25% reduction in silicon with a worsening in power consumption and speed of 5%-10%..

5.3 Viterbi Executing on Linux

If the target code workload is Viterbi executing instead of the



Idle Loop of Linux then the analysis in Section 6.2 remains valid. This is far from a representative workload for a general purpose computer but it may easily be a representative of the constrained workloads on embedded processors – especially those executing real-time control code.

For real-time systems, a requirement is to demonstrate the meeting of service deadlines. A simple experiment to refute the hypothesis that the VSP will not meet the deadline, is to set worst case delays for appropriate peripheral devices attached to the VSP, then run the experiment. For the simple VSP used here, memory being set as DDR or SDR gives the flavour of the experiment.

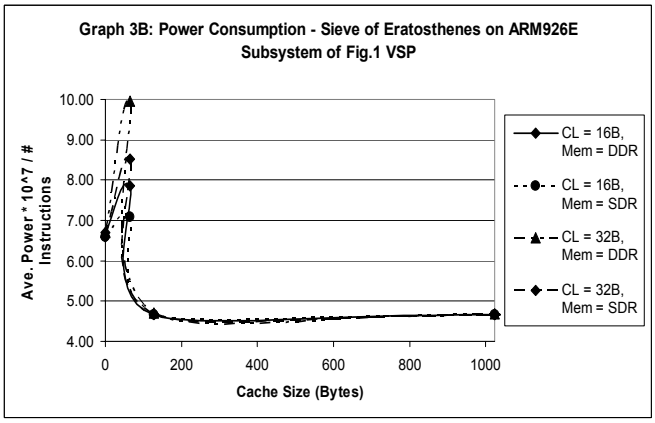
5.4 Alternate Memory Hierarchies

This investigation considered a pure embedded systems problem, that of finding the best tradeoff between speed, power consumption and silicon cost for a controller executing a limited amount of code – a prime number generator using the sieve of Eratosthenes algorithm [4]. This has the same outcome as the Viterbi experiment for cache sizes above 1 kbyte. However, we were interested in this experiment in determining the near minimum cache size that would still yield within 5% of optimum speed and power for the VSP .

In this experiment we considered I&D cache characteristics: size of 0B (uncached), 64B, 128B, 256B, 1 kB, 4 kB and 8kB; cache line size (16B, 32B), wayness (1, 2, 4), cache power weighting (3, 4, 5 – depending on size) and memory type (DDR, SDR). We varied the relative power consumption of the cache based on size. The results of the experiments are shown in Graphs 3A & 3B. The speed of the VSP followed expectations except that the transition between 64 bytes and 128 bytes was sharp and at 128B the VSP essentially achieved full speed. The power graph shows another picture. Uncached power consumed by the VSP was 20% - 35% less than the power consumed by 64 byte caches (variability was due to cache line size, wayness and memory type) and 200% higher than power consumed with 128 byte caches. What we are observing here is the step-function effect on power consumption of installing a cache in a

processor. For the sieve program, beyond 512 bytes, the power consumed was stable and about 20% higher than the minimum cache configuration at 128 bytes.

The effect on power consumption of installing a small cache in a processor to achieve a 4-fold increase in performance has a detrimental effect on power consumption due to the infrastructure required to support the cache. The cost of a cache is also high since the infrastructure consumes relatively large silicon real estate. These considerations led to an investigation



of alternative memory hierarchies that might achieve a better trade-off between speed, power and cost for a controller running a limited amount of code in an embedded application.

We varied the cache_hit/miss power weightings of the processor (see Table 2) to mimic the relative power consumed by a dedicated external buffer of 128 bytes (essentially a small, physically addressed, direct-mapped, on-chip cache external to the processor). This architecture is similar to the buffer organization found in processors like the Renesas SH2A [6] a processor popular in automotive control [1] where differences of cents in the price of a controller translate to several million dollars in large manufacturing runs. The results were that we could achieve a further ~40% power saving whilst maintaining near optimum speed. The cost of the chip is close to the non-cache cost. To prove that this is a global minimum requires more sophisticated statistical machinery (see [3]).

5.5 Algorithm Optimization

The final 10 experiments considered the effect of alternate algorithms on the problem of optimizing a VSP (software + hardware) for a particular (embedded) application. Since we had good empirical data already for the sieve prime number generator, we acquired from the web Kazmierczak's prime generation algorithm [7] and used the same experimental set-up as for the sieve experiments. The Kazmierczak algorithm required a small external buffer of 512 bytes to achieve maximum speed (IPC_{10}) ~40% higher than sieve and power consumption ~15% higher than sieve.

Clearly, algorithms have a 1st order effect on power, speed and cost – often say the dominant order effect! By just looking at or mathematically analyzing both the sieve and Kazmierczak algorithms, it is inconceivable that the optimal VSPs – that is software-hardware structure, as determined in this paper, would have been discovered.

6. Discussion and Conclusions

Empirical experimentation is a powerful mechanism with which to refute hypotheses that, when carefully constructed, drive the quantitative engineering process. To engage in this engineering process, prior to the existence of a physical realization, requires the existence and use of a model. If hypothesis building concerns speed, power consumption, reaction time, latency, meeting real-time schedules, etc. the model needs to be timing accurate (processor, buses, bus bridges and devices). If the extensive execution of software is an intrinsic part of the empirical experimentation, then the model needs to have high performance across all components. This paper assumes the existence of pre-silicon, high performance (20-100 MIPS), timing accurate virtual system prototypes.

Optimizing systems with complex objective functions is not intuitive. Complex tradeoffs between hardware structure and the software and algorithms that are executed on the hardware cannot be done by ratiocination or formal analysis alone, the acquisition of data as part of well-formed experiments refuting thoughtfully constructed hypotheses (ratiocination) enables decision making driven by results. Optimization comes from considering hardware and software together – not separately.

7. Acknowledgement

The capability underlying the empirical experimentation and data gathered and used to write this paper is due to the vision, effort and extra-ordinary dedication and execution of the high caliber VaST R&D team. Primus inter pares are Neville Clark, James Torossian, Foo Ngok Yong and Patricia Hughes. It is a pleasure to work with the whole VaST team.

8. References

[1] Winters, F.J., Mielenze, C. and Hellestrand, G.R. Design Process Changes Enabling Rapid Development. Proc. Convergence 2004 P-387, Oct 2004, 613-624, Society of Automotive Engineers, Warrendale, PA.

[2] Hellestrand, G.R. The Engineering of Supersystems. IEEE Computer, 38, 1(Jan 2005), 103-105.

[3] Hellestrand, G.R. Systems Architecture: The Empirical Way – Abstract Architectures to 'Optimal' Systems. ACM Conf. Proc. EmSoft2005, Sept 2005, Jersey City, NY.

[4] EEMBC: Embedded Microprocessor Benchmark Consortium. www.eembc.org

[5] Anthony, J. Design of Experiments. 2003, Butterworth-Heinemann, Oxford, UK.

[6] Renesas SH-2A, SH2A-FPU Software Manual, Rev 2.00, REJ09B0051-0200O, 13 Sept. 2004, Renesas Technology, Tokyo, Japan.

[7] Kazmierczak, M. Simple method of finding primes. 2002. http://www.mkaz.com/math/primes.html

[8] Lee, E. Absolutely Positively on Time: What Would It Take?. IEEE Computer, 38, 7(Jul 2005)

[9] Heiser, G. Private communication. Sept 2005. http://ertos.nicta.com.au/Research/L4/