Dynamic Voltage and Cache Reconfiguration for Low Power

Andre Costi Nacul and Tony Givargis
Department of Computer Science
University of California, Irvine
Center for Embedded Computer Systems
{nacul, givargis}@ics.uci.edu

Technical Report #03-34
November 7, 2003.

Abstract

Given a set of real-time tasks scheduled using the earliest deadline first (EDF) algorithm, we discuss two techniques for reducing power consumption while meeting all timing requirements. Specifically, we proposed a combined Dynamic Voltage Scaling (DVS) and Dynamic Cache Reconfiguration (DCR) technique for low power embedded systems. Toward this goal, we first analyze the potential power savings achievable by each technique (DVS or DCR) alone, then, we present an online algorithm that combines both techniques, reducing the power consumption even more. Our online algorithm gradually constructs a set of pseudo-Pareto-optimal system configurations for each task, which it then uses to determine a low power operating point meeting timing requirements. We evaluate the possible savings and observe that they are highly correlated with the specific timing requirements of the task. We also show that the combination of voltage and cache reconfiguration provides the best overall power savings, as much as 28% when considering the total platform power consumption.

Table of Contents

1. Introduction

Minimizing energy consumption of electronic devices has become a first class system design concern [1], especially, in the areas of embedded and portable devices, since such devices draw their current from batteries that place a limited amount of energy at the system's disposal. On the other hand, in recent years, increased application demand for functionality [2][3], market pressures, and shortening of design cycles, have led to a new system-on-a-chip (SOC) platform based design methodology [4].

A platform is a computing system composed of artifacts such as general-purpose processors, hierarchy of caches, on-chip main memory, I/O peripherals, co-processors, and possibly FPGA fabric for post-fabrication customizations. These platforms are generally targeted toward a large number of applications from a specific domain (e.g., networking or multimedia). To address the need for energy efficiency, the artifacts within these SOC platforms are often designed to be dynamically configurable. Features such as processor and memory power modes [5]; dynamic voltage scaling [6]; and run-time cache reconfiguration (e.g., Motorola's M*CORE [7][8]) have been commercially introduced. Dynamic reconfiguration of the platform provides an opportunity for operating system (OS) and/or application tasks to carry out strategic high-level resource management for low power.

Dynamic Voltage Scaling (DVS) has been extensively studied and shown to be a very efficient power saving technique [9][10][11]. More recently, some authors have focused their attention on cache reconfiguration, since a good tuning of the memory hierarchy also yields significant power savings [12][13][14][15][16][17]. Although DVS and *Dynamic Cache Reconfiguration* (DCR) are orthogonal solutions and could be applied to the same system individually in an optimal way, their combined effect on power consumption might not be simply a linear combination of the two optimal solutions. Both DVS and DCR rely on the same principle to save power: the processor/cache does not need to run at full speed/capacity at every moment in order to meet a task's deadline. We commonly refer to this excess processor speed or cache capacity as *slack*. In general, significant energy savings is achievable when reducing the slack.

In this work, we propose a combined DVS and DCR online algorithm that dynamically adapts the processor speed (i.e., voltage) and the cache subsystem to the workload requirements for the purposes of saving energy. The workload is considered to be a set of tasks with real-time deadlines. Our online algorithm is invoked as part of the OS scheduler, which performs standard earliest deadline first (EDF) task scheduling first. Then, our online algorithm, determines an ideal voltage/cache configuration for the current executing task.

This paper is organized as follows. In Section 2, we present some related previous work. In Section 3, we formalize the problem and introduce our proposed solution. In Section 4, we describe our experimental setup. In Section 5, we present our simulation results. In section 6, we introduce the algorithm for learning the configurations while the system is running. In Section 7, we state our concluding remarks.

2. Related Work

Dynamic Voltage Scaling (DVS) is an approach for power reduction that has gained much attention in the recent years. With DVS, one can save energy with minor performance degradation by reducing the operating supply voltage of the processor, or even of the whole system [9][10]. The premise of all DVS techniques is to achieve a steady/even processor speed while meeting all tasks deadlines. This is often accomplished by appropriately scheduling tasks and selecting voltage settings that eliminate the slack [11][18][19]. Some of these approaches consider an ideal processor with infinite voltage settings and no reconfiguration penalty while others focus on exact DVS behavior of commercial processors.

A great amount of previous work has also shown that statically tuning the cache subsystem to the running task can result in significant energy savings [12][13]. For example, Motorola's recent version of an M*CORE processor IC has a configurable 4 way set associative unified cache, in which each way can be disabled, or used for instructions, data, or both. Malik et al. [8] have shown that the best cache configuration depends heavily on the particular running task. Likewise, Zhang et al. [14][15] analysis shows that having a dynamically configurable line size architecture can have a significant (up to 50%) energy saving potential in embedded systems.

Tang et al. [16] have proposed an architectural scheme for dynamic cache line sizing. Their approach is to introduce a hardware unit along with a memory and cache protocol for fine grained tuning of the line size. In contrast, our approach is a software technique that allows the OS to take charge of cache reconfiguration, taking into account a dynamic workload and application requirements.

In a similar effort, Dropsho et al. [17] have considered disabling cache ways (i.e., associativity) dynamically to achieve low power. They propose cache architectures intended for dynamic reconfiguration. Further, they provide a hardware solution for adaptivity. As with the previous technique, our approach is a software technique performing the resource management at the task and OS levels.

Fan et. al. [20] also analyze the interactions between processor and memories, showing that there is a benefit in combining a memory-aware system with a DVS-enabled processor. However, their solution involves only shutting down parts of the main memory, while we try to combine the cache subsystem and the processor speed.

3. Problem Formulation

3.1. Overview

Our problem formulation is as follows. The system is composed of N tasks, T_1, T_2 … T_n. Each task T_i has a deadline D_i and a period P_i. To generalize the solution, a non periodic or sporadic task T_i is assumed to have $P_i = 0$. Tasks are *non-preemptive*. One of the tasks that is running on the platform is the scheduler T_s. Scheduler task T_s has no deadline and no period, and is activated every time a task finishes execution to perform the context switching. As stated previously, the scheduler selects the next task T_j to be executed based on EDF. Then, our online algorithm, running as part of the scheduler, selects an appropriate cache configuration that maintains the timing of the task T_j while saving as much energy as possible.

The platform's cache subsystem is assumed to have a finite number of possible configurations C_1, C_2 … C_n. Each configuration C_i will be different than any other configuration C_j by at least one of the configurable parameters: *cache size*, *line size* or *cache associativity*. Among all valid configurations, one of them is the so-called reference configuration C_r. The reference configuration is assumed to be the default system configuration, or the configuration to be used if dynamic cache reconfiguration is not used. For schedulability testing, we assume that the worse case execution time of each task under the reference configuration is known ahead of time (e.g., obtained via offline simulation).

The voltage of the platform can also be set to one of a finite set of voltages V_1, V_2 … V_n. A reduction in voltage directly affects the operating frequency of the system as well.

We assume a time penalty for cache reconfiguration. This penalty is for writing dirty data back to memory. The time penalty is captured by a function $P_T(C_i,C_j)$ of the current configuration C_i and the new configuration C_j. This function can be either hard coded statically, or learned by our online algorithm during run time.

As with time, there is also a power penalty associated with cache reconfiguration that is taken into account in our reported results. The power penalty is partially due to writing dirty data back to memory and is a function of the current and the new configuration as well as the last task that executed.

In a similar fashion, we assume a constant time penalty for selecting a new processor operating point (i.e., voltage/speed).

3.2. Proposed Solution

Any feasible solution in this context must address a multi-objective problem: minimize power while still meeting task deadlines. In a multi-objective problem, it is usually the case that one specific solution is good for one objective, but not so good for the other ones. In the universe of different configurations, we can identify some configurations that are better than all the other ones for at least one of the objectives. These are the so-called Pareto-optimal solutions.

Assuming the exact set of Pareto-optimal voltage and cache configurations for each task are known, our online algorithm, after performing EDF scheduling, picks the Pareto-optimal configuration that best fills the slack given the next task to be executed as shown in Algorithm 1.

Algorithm 1: Schedule

```
Input: $T_1$, $T_2$ … $T_n$
Output: $T_{next}$
Output: V, C  // voltage and cache points
$T_{next}$ := EDF($T_1$, $T_2$ … $T_n$);
slack := calculate_utilization(); // see Eq. 1
target_time := $T_{next}$.fastest/slack; // see Eq. 2
// assume P's are sorted w.r.t. execution time
for P ∈ $T_{next}$.Pareto-optimal-set do
  if P.time > target_time then
     break;
  <V, C> := P.<V, C>
```

The challenge, thus, is to compute the voltage and cache Pareto-optimal configurations for each task. Computing the Pareto-optimal set for the voltage parameter is trivial, since all the voltage configurations are part of the Pareto-optimal set. However, that is not the case for the cache parameters, and extensive simulations are needed in order to compute the exact Pareto-optimal set in this case.

For practical reasons, we have considered computing the Pareto-optimal sets online. However, due to computation overhead, it is not feasible to compute the exact Pareto-optimal sets. Instead, an approximation of the Pareto-optimal set is sufficient. In section 6, we present our online algorithm to compute the approximate Pareto-optimal sets.

Given the Pareto-optimal sets (or an approximation in the online case), the system can trade-off power consumption with execution time by selecting the configuration that is best suited to fill the excess processing time or cache capacity (i.e., slack). The configuration selection is based on the utilization rate of the processor. The utilization rate of the processor is calculated every time a task finishes execution, or whenever a task is added or removed to and from the system. At any moment, given that the tasks are sorted according to EDF, the utilization rate can be calculated as follows.

$$util = \max_{\forall i}\left(\frac{\sum_{j=1}^{i} exec\_time_j}{deadline_i - current\_time} \right) \qquad \text{Eq. (1)}$$

For the utilization calculation, the best case execution time (but not necessarily most energy efficient) of each task is used. Given this utilization rate, we calculate the target execution time for the next task T_j as shown below.

$$target\_exec\_time_j = {exec\_time_j} \big/ {util} \qquad \text{Eq. (2)}$$

Given the target execution time, the scheduler is able to select the Pareto-optimal configuration that has a time less, but closest to the target time.

4. Experimental Setup

In order to evaluate the effects and benefits of our online algorithm, we have performed several simulations. In our simulations, we have considered different task timings and have experimented with only DVS, only DCR, and the combination of DVS and DCR.

Our simulations were performed on a target platform that is composed of a MIPS processor, unified L1 reconfigurable cache, on-chip memory, and the associated busses between the cache and the processor, as well as cache and on-chip memory, as depicted in Figure 1. In addition, our platform includes a hardware power monitor for real-time power measurements.

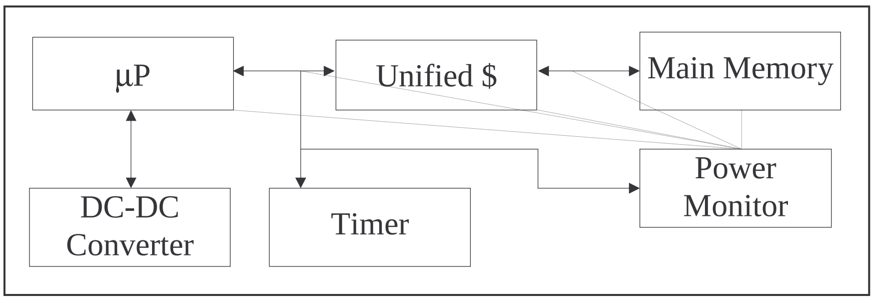

Figure 1 - The Target Platform

Task running on our platform are able to dynamically modify the cache configuration through the use of a dedicated register. Similarly, task running on our platform can dynamically scale the voltage via the DC/DC converter. The unified cache can be configured to accommodate different cache sizes, line sizes, and degrees of associativity. Cache size ranges from 1K to 32K, in powers of two intervals. Line size can be set to values between 4 and 64, also in power of 2 increments. Finally, the degrees of associativity are 1 (direct mapped cache), 2, 4 and 8. The total number of possible cache configurations is the cross-product of these parameters, resulting in 820 different valid platform configurations. The DVS sub-system of our platform is modeled after the Intel XScale commercial processor [21]. Specifically, there are 7 possible voltage/frequency combinations that range from a fast, 400 MHz, working at 1.3 V, down to a slower 100 MHz clock operating at 1V.

In our experiments, we ran the task-set under different system scenarios, as listed below:

(A) Largest cache (largest size, line, and associativity) configuration and maximum voltage. This is the platform configuration for highest performance.

(B) Typical cache configuration and static voltage. Typical cache configuration is derived from an offline analysis of the benchmarks, leading to a cache configuration that would perform reasonably well (in terms of power and performance) over a large set of the benchmarks executed.

- (B1) running at 400Mhz
- (B2) running at 330Mhz
- (B3) running at 300Mhz

(C) Large cache configuration and fixed voltage. The processor voltage is set offline to a fixed voltage, and no dynamic adjustments are made.

- (C1) running at 300Mhz
- (C2) running at 266Mhz

(D) Large cache configuration and DVS. In this configuration, we can benefit from DVS in a system that has a general, high performance cache configuration.

(E) Typical cache configuration and DVS. In this configuration, we can benefit from DVS in a system that has a tuned cache configuration.

(F) DCR and fixed voltage. In this case, the voltage is fixed, and the cache configuration is modified at run-time, according to the task that is running;

- (F1) running at 400Mhz
- (F2) running at 330Mhz

(F3) running at 300Mhz

(G) DCR and DVS. The combination of DVS and DCR potentially maximizes the power savings and yield a better usage of the available slack.

The selected applications represent a mix of large, medium, and small embedded applications drawn from the PowerStone [8], MediaBench [23], and MiBench [22] application sets. Specifically, we selected *v42* (modem protocol), *g3fax* (fax encoding), *crc* (checksum calculation) and *compress* (the Unix compression utility). The task deadlines were set so that different amount of slack were available to be used by our online algorithm.

Our reported time and power results include all the penalties associated with having DVS and cache reconfiguration. In the DVS case, the time penalty is considered to be the time needed by the PLL clock circuit to stabilize. The power penalty is due to the power dissipation in the DC-DC converter. The cache reconfiguration penalty includes the effects of the cache flush before each reconfiguration, and the cold misses that result from the fact that the cache is cleared before every task is executed.

For this work, we assumed that tasks are non-preemptive and that they execute to completion before yielding the processor back to the scheduler. Although it is possible to apply dynamic cache reconfiguration to a preemptive system, the recurring penalties of flushing the cache, associated to the extra cold misses that occur when a task is replaced in the processor would probably reduce the overall savings.

5. Combining DVS and DCR

In our first set of experiments we attempted to estimate the possible savings by the combination of DVS and DCR. In order to get a more accurate estimate of the largest savings possible, the operating system was fed with the actual Pareto-optimal sets. We call this an Oracle solution, since it knows the behavior of the applications beforehand.

We observed that the combination of DVS and DCR had a potential for larger savings than either of the two techniques alone. However, the effective energy savings was highly dependable on the deadlines (and so on the slack available), as expected. Table 1 summarizes the results for the different scenarios and platform configurations.

Configuration	D=19.0ms	D=20.3ms	D=22.5ms
(A) 32K,64,8,400Mhz	2.26W 13.4ms	2.29W 13.4ms	2.33W 13.4ms
(C1) 32K,64,8,300Mhz	1.14W 18ms	1.16W 18ms	1.19W 18ms
(C2) 32K,64,8,266Mhz	n/a	0.99W 20.2ms	1.02W 20.2ms
(D) 32K,64,8,DVS	1.04W 18.9ms	1.04W 19.6ms	0.93W 21.9ms
(B1) 16K,16,2,400Mhz	0.65W 16.3ms	0.67W 16.3ms	0.69W 16.3ms
(B2) 16K,16,2,330Mhz	n/a	0.51W 19.7ms	0.54W 19.7ms
(B3) 16K,16,2,300Mhz	n/a	n/a	0.34W 21.7ms
(E) 16K,16,2,DVS	0.53W 18.5ms	0.50W 19.2ms	0.44W 22.3ms
(F1) Dynamic,400Mhz	0.55W 18.7ms	0.52W 20.2ms	0.47W 21.7ms
(F2) Dynamic,330Mhz	0.56W 18.4ms	0.51W 20.2ms	0.45W 22.4ms
(F3) Dynamic,300Mhz	n/a	n/a	0.33W 22.3ms
(G) Dynamic,DVS	0.52W 18.9ms	0.45W 20.1ms	0.32W 22.4ms
n/a = not possible to meet time constrains with the respective configuration			

Table 1. Summary of experimental results.

Based on Table 1, we conclude that a DVS-only system performs slightly better than a DCR-only system. For example, the results of configuration E (DVS only) and F2 (DCR only) are very close, with a small advantage in terms of power to E. However, the difference is almost irrelevant. DVS and DCR perform very similarly in all scenarios. At the same time, it is possible to notice that the combination of DVS and DCR (experiment G) always has better overall savings than either of the techniques alone.

Based on Table 1, we observe that the additional savings provided by the combination of DVS and DCR increases with larger slacks. In the 19 ms deadline scenario, there is almost no gain in adding DCR to the system (i.e., changing from E to G). However, in the 20.3 ms and 22.5 ms, there is an extra saving of 10% and 27%, respectively.

Table 1 also shows that combining DVS and DCR allows a better usage of the slack. For example, in the 20.3 ms scenario, DVS only (experiment E) can slowdown execution to 19.2 ms, leaving 1.1 ms unused. When DCR is combined to DVS (experiment G), the execution time is stretched to 20.1 ms, almost fulfilling the available processing time. DCR only (experiment F2) has an execution time that is even closer to completely using the slack. Overall, experiment G is consistently closer to using all the available slack than D or E.

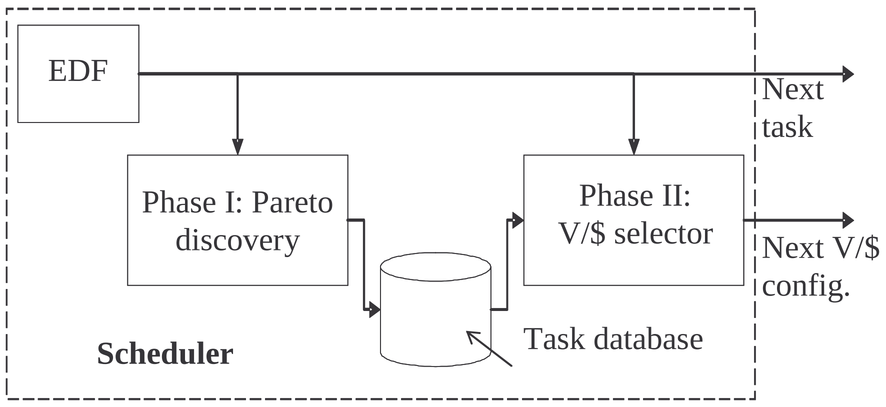

Figure 2 - The Online Algorithm

6. Online Reconfiguration

As an alternative to pre-computing the exact Pareto-optimal sets, this section introduces an online algorithm for calculating an approximation of the Pareto-optimal sets.

This online algorithm uses the same scheduling algorithm that was discussed in Section 3. Here, the OS scheduler additionally interleaves the configuration selection with the configuration discovery algorithm, as depicted in Figure 2. After each invocation of the scheduler, a new point may be added to the Pareto-optimal sets.

In our algorithm, a common task configuration database is shared between discovery and selection. The database is build incrementally by the Pareto discovery algorithm, and is used to keep information about the Pareto-optimal configurations known so far. After a finite number of iterations, the discovery process is considered finished and the database is stable.

6.1. Discovery Algorithm

The main objective of the Pareto discovery algorithm is to converge on to a reasonable approximation of the actual Pareto-optimal set for each task.

The discovery procedure starts with the reference configuration as the only member of the Pareto-optimal sets. Gradually, each of the cache size, line size, and associativity parameters are varied, individually (i.e., one change per scheduler invocation) in a greedy search process. Specifically, in a first stage, starting from the reference configuration, the cache size parameter is changed until all possible settings have been explored, or the task timings are affected beyond certain threshold. Then, in a similar fashion, during second and third stages, the cache line and associativity parameters are varied for all the configurations in the Pareto database at the time.

A new point p_i is introduced into the pseudo-Pareto-optimal set P if it has a better time or power measure than every other point $p_j \in P$. The newly added point p_i will invalidate any existing point $p_j \in P$ if p_j has an inferior time and power measures than p_i. Invalidated points are removed from the set P. The discovery algorithm is merged with the scheduler, resulting Algorithm 2.

Algorithm 2: Discover and Schedule

```
Input: $T_{curr}$      // current task
Input: $T_1$, $T_2$ … $T_n$
Output: $T_{next}$     // next task
Output $V$, $C$      // voltage and cache
// compute delta time and power
dtime := time() – $T_{curr}$.start_time
dpower := (energy() – $T_{curr}$.start_energy)/dtime
// introduce new Pareto points
is_pareto := true
for $P \in T_{curr}$.Pareto-optimal-set do
  if P.time < dtime && P.power < dpower then
    is_pareto := false
if is_pareto then
  $T_{curr}$.P := Ti.P $\cup$ { $C_i$ }
  for $P \in T_{curr}$.Pareto-optimal-set do
    if( P.time > dtime && P.power > dpower )
      $T_i$.P := $T_i$.P – { P }
// perform standard scheduling
$T_{next}$ := EDF($T_1$, $T_2$ … $T_n$);
// explore or select
if need_to_explore($T_{next}$) then
   C,V := discover_pareto($T_{next}$)
else
   C,V := pick_best_config($T_{next}$) // see Alg. 1
// prepare for next execute
$T_{next}$.start_time := time()
$T_{next}$.start_energy := energy()
return($T_{next}$, V, C)
```

The complexity of the discovery algorithm can be analyzed for each operation. Checking if a new point is a Pareto-optimal point is $O(log\ N)$, where N is the total number of Pareto configurations in the set, since the Pareto set is kept in a binary heap data structure. Whenever a point is added to the set, it demands that a filtering process is performed, to eliminate the configurations that are invalidated by the newly added point. This operation is $O(N)$, since one new point might invalidate all the other points in the set if it has a better execution time and power consumption than all the other points in the set.

6.2. Simulation Results

The power consumption results for the online approach are depicted in Figure 3, Figure 4, and Figure 5. For comparison purposes, all the figures include the plots for the online system as well as the Oracle system (see Section 5) and the reference configuration.

As expected, the online performance is slightly worse than the Oracle DCR+DVS implementation. The worst increase happened in the case when the deadline is 22.5 ms, where the power consumption increased by as much as 20%. Despite the slight overall increase in power, savings are still higher when DVS and DCR are combined when compared to either technique alone.

The online discovery behavior can also be seen in the plots shown in Figure 3, Figure 4, and Figure 5. Initially, the power consumption oscillates quickly, as the system discover new Pareto-optimal points. As the discovery converges, the power profile stabilizes.

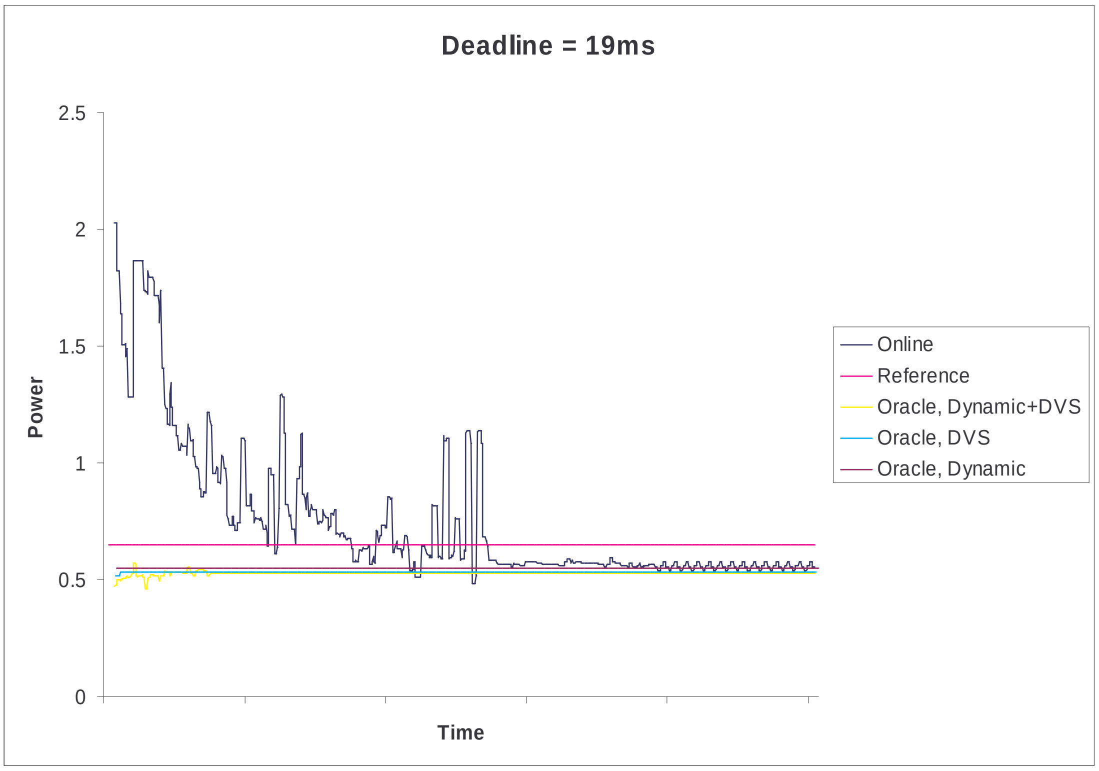

Figure 3 - Online Behavior, Deadline = 19ms

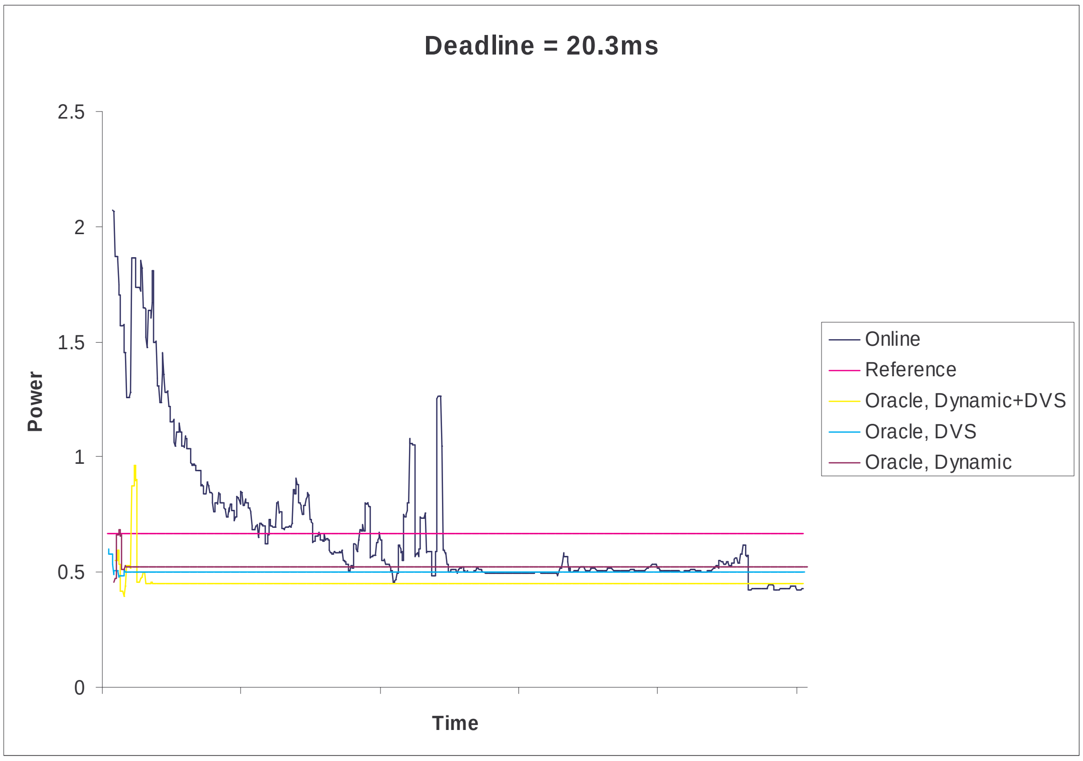

Figure 4 - Online Behavior, Deadline = 20.3ms

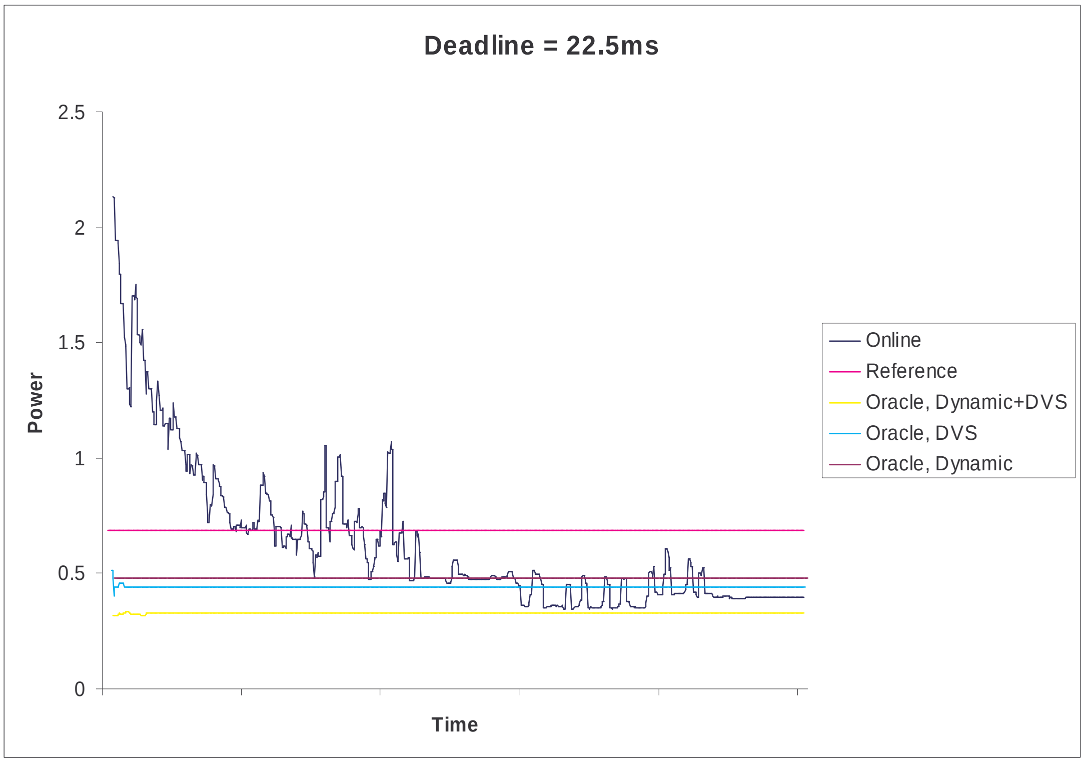

Figure 5 - Online Behavior, Deadline = 22.5ms

Furthermore, while the system is testing new cache configurations, some deadlines are eventually lost. In the tighter execution, with deadline set to 19 ms, 10% of the deadlines are lost during discovery. On the other hand, when deadline is 22.5 ms, only 4% of the deadlines are lost during the discovery of the pseudo-Pareto-optimal configurations. Clearly, the online approach is not suitable for real-time applications with strict deadlines. In the hard real-time instances, the static approach to discovering Pareto-optimal points should be utilized

As a final remark, we observed that the discovery process requires to analyze about 60-70 platform configurations in order to converge. This is less than 10% of the 820 possible configurations when cache and voltage are combined.

7. Conclusions

We have discussed the combination of Dynamic Voltage Scaling (DVS) and Dynamic Cache Reconfiguration (DCR) for power reduction is embedded systems. We have analyzed the power savings achievable using these techniques alone or in a combined way. We have presented an online algorithm for dynamically selecting the best cache and voltage configuration using the Pareto-optimal sets. We have also presented an online algorithm for discovery of the Pareto-optimal sets. Our results show that the combination of DVS and DCR can achieve 27% more savings when compared to DVS alone.

Acknowledgments

This work was supported in part by a National Science Foundation Award (#0205712) and by a CAPES Foundation, Brazil scholarship (#1054015).

References

[1] T. Mudge. Power: A First Class Architectural Design Constraint. IEEE Computer, vol. 34, no. 4, pp. 52-57, 2001.
[2] International Technology Roadmap for Semiconductors (ITRS), 2001.
[3] D. Wingard, R. Fordham, J. Ready, F. Romeo, A. de Oliveira. Embedded system design: the real story, in Proceedings of the Design Automation Conference, 2001.
[4] F. Vahid, T. Givargis. The case for a configure-and-execute paradigm, in the Proceedings of the International Workshop on Hardware/Software Codesign, 1999.
[5] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, M.J. Irwin. Hardware and Software Techniques for Controlling DRAM Power Modes. IEEE Transactions on Computers, vol. 50, no. 11, pp. 1154-1173, 2001.
[6] L. Geppert, T.S. Perry. Transmeta's magic show. IEEE Spectrum, vol. 37, no. 5, pp. 26-33, May 2000.
[7] Motorola M*CORE Product Page. http://www.motorola.com.
[8] A. Malik, B. Moyer, D. Cermak. A Lower Power Unified Cache Architecture Providing Power and Performance Flexibility. International Symposium on Low Power Electronics and Design, June 2000.
[9] T.D. Burd, T.A. Pering, A.J. Stratakos, R.W. Brodersen. A Dynamic Voltage Scaled Microprocessor System. IEEE International Solid-State Circuits Conference, Nov. 2000.
[10] T. Pering, T. Burd, R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. International Symposium on Low Power Electronics and Design. Aug. 1998.
[11] A. Rae, S. Parameswaran. Voltage Reduction of Application-Specific Heterogeneous Multiprocessor Systems for Power Minimization. ASP-DAC, 2000, pp. 147-152.
[12] P. Petrov, A. Orailoglu. Towards Effective Embedded Processors in Codesigns: Customizable Partitioned Caches. International Workshop on Hardware/Software Codesign, 2001.
[13] C. Su, A.M. Despain. Cache Design Trade-offs for Power and Performance Optimization: A Case Study. International Symposium on Low Power Electronics and Design, 1995.
[14] C. Zhang, F. Vahid, W. Najjar. A Highly Configurable Cache Architecture for Embedded Systems. In Proceedings of International Symposium on Computer Architecture. 2003.
[15] C. Zhang, F. Vahid. Cache Configuration Exploration on Prototyping Platforms. In Proceedings of International Workshop on Rapid Systems Prototyping. Jun. 2003.
[16] W. Tang, A. Veidenbaum and R. Gupta. Architectural Adaptation for Power and Performance. Proceedings of the International Conference on Supercomputing, pp 145-154, 1999.
[17] S. Dropsho, et al. Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power. Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2002.
[18] P. Pillai, K. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), October, 2001
[19] I. Hong, M. Potkonjak, M. Srivastava. On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor. In Proceedings of International Conference on Computer Aided Design, 1998.
[20] X. Fan, C Ellis, A. Lebeck. The Synergy between Power-aware Memory Systems and Processor Voltage Scaling. Technical Report CS-2002-12, Department of Computer Science, Duke University, 2002.
[21] Intel Corporation. Intel PXA255 Processor Developer’s Manual. Mar. 2003.
[22] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, R. Brown. MiBench: A free, commercially representative embedded benchmark suite. IEEE 4th Annual Workshop on Workload Characterization. Dec. 2001.
[23] C. Lee, M. Potkonjak, W. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In Proceedings of Annual IEEE/ACM International Symposium on Microarchitecture, Dec. 1997, pp. 330 -335.

[24] T. Givargis, F. Vahid. Platune: A Tuning Framework for System-on-a-chip Platforms. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, v21, n11, pp 1317-1327, 2002.