Adaptive Cache Management for Low Power Embedded Systems

André Nácul* · Tony Givargis*

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[1,2], market pressures, and shortening of design cycles, have led to a new system-on-a-chip (SOC) platform based design methodology[3].

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[4]; dynamic voltage scaling[5]; and run-time cache reconfiguration (e.g., Motorola's M*CORE[6, 7]) have been commercially introduced. Dynamic reconfiguration of the platform provides an opportunity for operating system (OS) and/or application tasks to carry out strategic highlevel resource management and achieve energy savings.

In this work, we propose an online algorithm for dynamically adapting 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 cache configuration for the current task that is to be executed.

In our experiments, we consider the overhead of the OS scheduler, our online algorithm, as well as the cache reconfiguration time and energy penalties. Furthermore, we evaluate the quality of our technique by measuring the global

* Department of Information and Computer Science and members of the Center for Embedded Computer Systems, University of California, Irvine, CA 92697 USA.

power savings (i.e., considering processor, memories, and buses in addition to caches).

When invoked, our algorithm initially performs an incremental search of the cache configuration space and updates a set of pseudo-Pareto-optimal points for the current task to be executed. Subsequently, the pseudo-Pareto-optimal set is used to select a configuration meeting the task deadline while minimizing power consumption.

The remainder of this paper is organized as follows. In Section 2, we outline related work. In Section 3, we formulate the problem and state our assumptions. In Section 4, we introduce our online algorithm. In Section 5, we present our experiments. In Section 6, we give our concluding remarks.

2. Related Work

One similar approach that has gained popularity is dynamic voltage scaling (DVS), where one can save energy with minor performance degradation by reducing the operating supply voltage of the processor, or even of the whole system[8-10]. 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. Our approach is completely complementary to DVS. In our approach, we do not perform task scheduling. Instead, we assume an already scheduled task set. The premise of our work is to tune the cache down to the working set of each task that is to be executed, thus saving on cache power consumption. Our aim is not to disturb the task timings. Thus, the advantage is the possibility of combining a DVS scheduler with our approach for added benefit.

A great amount of previous work has shown that statically tuning the cache subsystem to the running task can result in significant energy savings[11,12]. 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. [7] have shown that the best cache configuration depends heavily on the particular running task. Likewise, Zhang et al.[13] 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.[14] 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.[15] 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 adaptivity at the task and OS levels.

3. Problem Description

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 nonpreemptive. One of the tasks that are 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).

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. There is also a power penalty associated to the cache reconfiguration that is also taken into account in the results. The power penalty is also due to writing dirty data back to memory and is a function of the current and the new configuration. Both time and power penalty for reconfiguration are different according to the cache configuration and the task that has completed. Because of the dynamic behavior of each different application, the amount of dirty data is not constant, and so the reconfiguration cost is dependable on both the task and the current cache configuration.

Fig. 1 depicts the runtime behavior proposed here.1) Here, task T_i runs with cache configuration C_i, while T_j with C_j. Between these two tasks, the scheduler T_s executes with the same cache configuration of the last running task. Our

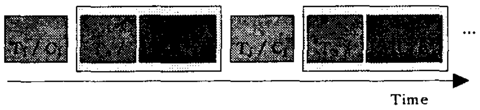

Fig. 1. Runtime behavior

1) The time line is not to scale.

algorithm runs as part of the scheduler task T_s. Note that the time penalty $P_T(C_i,C_j)$ of cache reconfiguration is also depicted in the figure.

We note that being able to select the next cache configuration without any prior knowledge of the currently running task is especially important, as we assume that the work load is dynamic and not necessarily known during design time. The main advantage of our algorithm is that it learns about task behavior under different cache configurations in a dynamic setting.

We assume that there is a way to measure the power consumption of each task just executed on the platform. We assume that this power consumption is inclusive of the power penalty for cache reconfiguration. Checking the power consumption can be accomplished by reading cache access counters and applying appropriate power models. Alternatively, a platform may provide direct measurements of the power consumption of its components.

We allow for some missed deadlines as the online algorithm is learning about the task behavior under different cache configuration. This is a reasonable assumption for soft real-time application where occasional lose of a deadline is not as critical as in a hard realtime application. Despite this timing relaxation, soft real time applications (e.g., multimedia, videoconferencing, etc.) are very common in the embedded system environment. In a strict deadlines scenario, there is little that can be done, since our algorithm is likely to miss some deadlines when learning about the task behavior. One alternative is to suppress the learning phase, providing the scheduler with offline data for each task. We leave this out of this work, since it is a simple case of the proposed approach.

4. Proposed Solution

4.1 Overview

Dynamic cache reconfiguration poses a trade-off between power and performance. Larger caches are supposed to reduce the number of misses, allowing a task to execute faster. On the other hand, the energy needed for a read or write in a bigger cache is larger than in a smaller cache, leading to a higher energy consumption scenario.

This is clearly a multi-objective function: we want to minimize power while still meeting a time constraint. In a multi-objective function, 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 performance criteria. These are the so-called Pareto-optimal solutions.

However computing the exact set of Paretooptimal configurations is a challenging problem, as the configuration space is likely to be large. Instead, we aim at computing an estimated Pareto-optimal set (i.e., a near-optimal set) which we refer to as the pseudo-Pareto-optimal cache configurations. We dynamically discover the pseudoParetooptimal cache configurations, for each task, which are used to determine the best cache configuration for low power.

Our online algorithm operates in two phases. The first phase of our online algorithm is designed to discover the performance of each task under different cache configurations. This is the Pareto-discovery phase. The second phase of our online algorithm is the cache configuration selector.

Fig. 2 depicts the parts of the task scheduler, which includes the two phases of our online algorithm. Note that the execution of the two phases of our online algorithm is interleaved and every time the scheduler is activated one iteration is completed for each of phase. We also point out that during phase I, the discovery phase, it is possible that some deadlines are missed. However, after this phase is over, the application behavior is known to the scheduler and no further deadlines are expected to be missed. 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

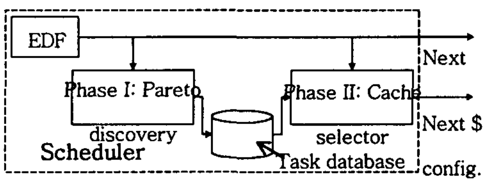

Fig. 2. The online algorithm

information about the pseudo-Pareto-optimal configurations known so far. After a finite number of iterations, the discovery process is considered finished and the database is stable. The configuration selection algorithm consults the database in order to pick the best cache configuration to be set for the current activation of the task.

The complete scheduler along with our online algorithm skeleton is given in Fig. 3.

```
SCHEDULER:
 Input: current task and config. ($T_i$, $C_i$)
 Output: next task and config. ($T_j$, $C_j$)

 // compute delta time and power
 dtime = time()  $T_i$.start_time
 dpower = (energy()  $T_i$.start_energy)/dtime

 // introduce new Pareto points
 is_pareto = true
 for each $p_k$ in $T_i$.P
  if( $p_k$.time < dtime && $p_k$.power < dpower )
   is_pareto = false
 if( is_pareto ) {
  $T_i$.P = Ti.P  { $C_i$ }
  for each $p_k$ in $T_i$.P
    if( $p_k$.time > dtime && $p_k$.power > dpower )
     $T_i$.P = $T_i$.P  { $p_k$ }
 }

 // perform standard scheduling
 $T_j$ = EDF()

 // explore or select
 if( need_to_explore($T_j$) )
   $C_j$ = discover_pareto($T_j$) // Section 4.2
 else
   $C_j$ = pick_best_config($T_j$) // Section 4.3

 // prepare for next execute
 $T_j$.start_time = time()
 $T_j$.start_energy = energy()
 return($T_j$, $C_j$)
```

Fig. 3. Scheduler skeleton

4.2 The Pareto Discovery Phase

The main objective of the Pareto discovery phase 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 pseudoParetooptimal set. Gradually, each of the cache size, line size, and associativity parameter 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 the configurations that are in the pseudo-Pareto-optimal set.

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.

4.3 Configuration Selection Phase

The configuration selection phase is based on the utilization rate of the processor. With smaller utilization rates, there is additional slack to change the current cache configuration to a lower energy at a higher execution time configuration.

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.

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

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.

$$target\_exec\_time_j = {exec\_time_j}/{util}$$

Given the target execution time, our online algorithm selects the pseudo-Pareto configuration that has a time less (but closest) to the target time.

Note that utilization rate of 1.0 means that there is no slack available, thus the fastest configuration must be used to meet the deadline. On the other extreme, a low utilization rate of 0.1 means that only 1/10 of the processing available is committed to task execution, and thus the system can shift to a much lower cache configuration, increasing the execution time and saving power.

5. Experiments

The target SOC platform used in our experiments is shown in Fig. 4. The Platune simulator was modified for our experiments [16]. Our platform included a MIPS processor with unified cache, main memory, and the associated buses. A timer peripheral was used by the scheduler for interval timing. Likewise, a hardware power monitor, based on models developed in Platune, was incorporated into the platform for task power measurements.

The simulated results for time and power consumption include all the penalties associated to cache reconfiguration, including the cache flush before each reconfiguration, and the cold misses that result from the fact that the cache is cleared before every task is scheduled. An operating system scheduler was also implemented, so that the scheduling overhead could be taken into account. Our online algorithm was incorporated into this scheduler.

For our platform, the reference configuration C_r was set to be an 8K byte, with a line size of 4 bytes, and a 2-way set associativity. The possible cache sizes ranged from 256 bytes to 8K bytes, in power of 2 increments. The possible line size ranged from 4 to 16 bytes in powers of 2 increments. Finally, the possible degree of associativity ranged from 1 to 4.

For our experiments, we used typical embedded system tasks that are part of the

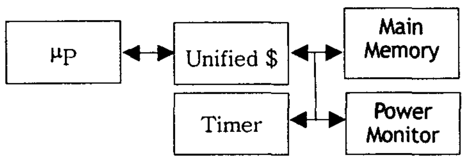

Fig. 4. Target SOC Platform

PowerStone benchmark applications [16]. These tasks included a Unix compression utility called *compress*, a CRC checksum algorithm called *crc*, an encryption algorithm called *des*, an engine controller called *engine*, an FIR filter called *fir*, a fax decoder called *g3fax*, a sorting algorithm called *ucbqsort*, an image rendering algorithm called *blit*, a POCSAG communication protocol called *pocsag*, and a JPEG decoder called *jpeg*.

We did three sets of experiments, each set corresponding to one of *high* processor utilization, *medium* processor utilization, and *low* processor utilization. In other words, we selected a mix of tasks, from the Powerstone set of tasks, along with appropriate deadlines to result in a processor utilization of 90%, 50%, and 20%. The task mix included tasks with different data work set size, therefore different cache requirements. One task with a large work set was selected, jpeg, together with medium and smaller tasks, g3fax, ucbqsort and blit.

Fig. 5 depicts our results for the three different processor utilization experiments. In the figure, the steady/dashed line gives the power consumption of the overall system if configured to execute with the reference cache configuration. The varying plot depicts power consumption of the system, as a function of time, as it adapts using our approach.

Note that during early stages, the power profile oscillates. This is due to the algorithm discovering the pseudo-Pareto-optimal points. During this time, we note that some deadlines are missed. For example, in the low processor utilization, the final power consumption (e.g., at 25 second marker) is higher than that corresponding to the earlier seen configurations (e.g., 5 second marker). However, that con-

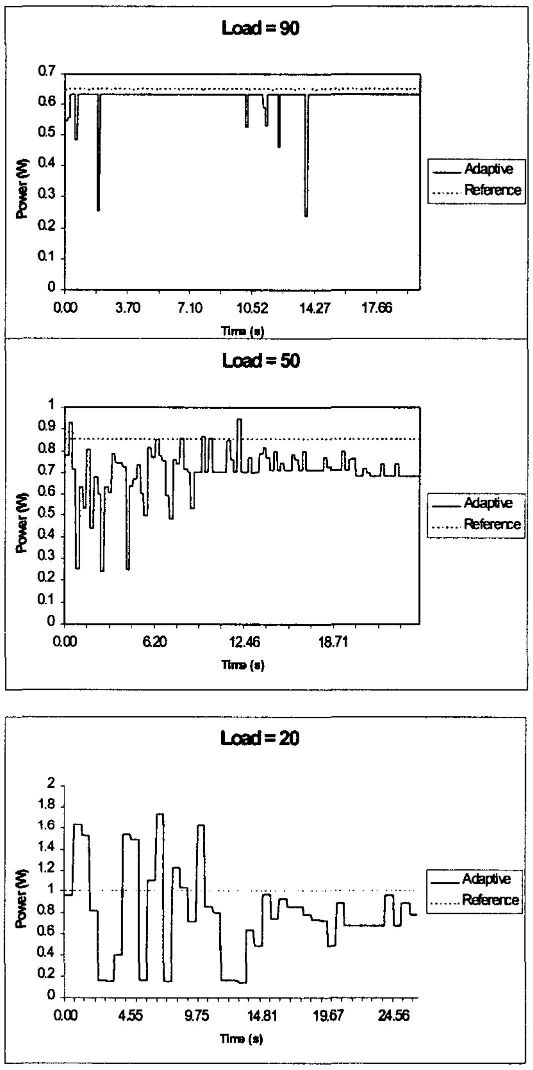

Fig. 5. Experimental Results

figuration is not used because of a deadline miss. Also, note that when the processor utilization is very high, most of the tasks need to run under the fastest configuration, which for us is the reference configuration, in order to meet the deadlines. In this case, there is as little as 1% opportunity for saving power. However, as the utilization goes down to medium and low, significant energy saving, namely 19% and 22% respectively, can be observed. We note that this is overall SOC platform power saving and not just the cache subsystem.

We also point out that the presented algorithm is generic enough to be used in SOC and non-SOC platforms. The results presented here are regarding the SOC platform previously discussed. In order to evaluate non-SOC platforms, one needs to adapt the power models of the simulator, especially those related to the cache-memory bus and the main memory. The overall savings might be different as well.

6. Conclusion

We have proposed an online algorithm for dynamically reconfiguring the cache subsystem of a system-on-a-chip (SOC) platform to meet timing requirements while minimizing power consumption. Our online algorithm gradually constructs a set of pseudo-Pareto-optimal cache configurations for each task, which it then uses to determine a low power operating point meeting timing requirements. We have evaluated the quality of our algorithm by considering the overall energy saving of an SOC platform, including the time and energy overhead of the scheduler and our online algorithm, as well as the cache reconfiguration penalties. Our results show savings of approximately 20% in overall system energy usage.

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, November 2000.

[10] A. Rae, S. Parameswaran. Voltage Reduction of Application-Specific Heterogeneous Multiprocessor Systems for Power Minimization. ASP-DAC, 2000, pp. 147-152.

[11] 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.

[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 and W. Najjar. Energy Benefits of a Configurable Line Size Cache for Embedded Systems. Proceedings of the IEEE Computer Society Annual Symposium on VLSI, 2003.

[15] W. Tang, A. Veidenbaum and R. Gupta. Architectural Adaptation for Power and Performance. Proceedings of the International Conference on Supercomputing, pp 145-154, 1999.

[16] 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.

[17] T. Givargis and 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.

André Nácul

• André Nácul is a PhD student at the University of California, Irvine, and is a member of the Center for Embedded Computer Systems at UC Irvine. He is a CAPES Foundation Fellow, and received a CALIT2 fellowship for the 2002/03 academic year. He received a B.S. in Computer Science from the Universidade Federal do Rio Grande do Sul, Brazil, in 1999, and a M.Sc. in Computer Science from the same institution in 2002. His research interests include resource management and operating systems for embedded computers.

Dr. Tony Givargis

• Dr. Tony Givargis received his B.S. and Ph.D. in Computer Science from the University of California, Riverside (1997 and 2001), where he was a GAANN (Graduate Assistance in the Area of National Need) Fellow as well as recipient of the department's best Thesis Award. He is currently an Assistance Professor in the Department of Computer Science at the University of California, Irvine. He is also a member of the Center for Embedded Computer Systems at University of California Irvine. Dr. Givargis is a co-author of the textbook "Embedded System Design" (J. Wiley and Sons 2002). His research focuses on issues related to the design of processor architectures (i.e., hardware platforms), Operating Systems, and user-level software, with respect to ultra low power design constraints.