Compiler-Directed Dynamic Voltage/Frequency Scheduling for Energy Reduction in Microprocessors*

Chung-Hsing Hsu and Ulrich Kremer
Department of Computer Science
Rutgers University
{chunghsu,uli}@cs.rutgers.edu

Michael Hsiao
Department of Electrical and Computer Engineering
Rutgers University
mhsiao@ece.rutgers.edu

DCS-TR-431

February 2001

Abstract

Dynamic voltage and frequency scaling of the CPU has been identified as one of the most effective ways to reduce energy consumption of a program. This paper discusses a compilation strategy that identifies scaling opportunities without significant overall performance penalty. Simulation results show CPU energy savings of 3.97%-23.75% for the SPECfp95 benchmark suite with a performance penalty of at most 2.53%.

1 Introduction

Modern architectures have a large gap between the speeds of the memory and the processor. Techniques exist to bridge this gap, including memory pipelines, cache hierarchies, and large register sets. Most of these architectural features exploit the fact that computations have temporal and/or spatial locality. However, many computations have limited locality, or even no locality at all. In addition, the degree of locality may be different for different program regions. Such computations may lead to a significant mismatch between the actual machine balance and computation balance, typically resulting in long stalls of the processor waiting for the memory subsystem to provide the data.

We will discuss the benefits of compile-time voltage and frequency scaling where the compiler identifies promising program regions for CPU voltage and CPU frequency scaling, and assigns clock frequencies and voltage levels for their execution. The goal is to provide similar overall performance while significantly reducing the power/energy dissipation of the processor. Opportunities for such an optimization are program regions where the CPU is mostly idle.

*This research was partially supported by NSF CAREER award CCR-9985050 and a Rutgers University ISC Pilot Project grant.

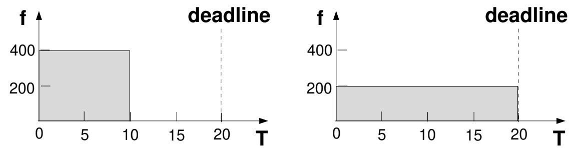

Figure 1: The essence of voltage scheduling: slack elimination

1.1 Background

The total execution time (T) and energy consumption (E) of a program can be estimated by

$$T \approx W \cdot \frac{1}{f} \quad \text{and} \quad E \approx C \cdot W \cdot V^2$$

where W is the total number of execution cycles, f is the clock frequency, C is the effective switching capacitance, and V is the supply voltage. C and W are assumed to be independent of frequency f. Since in dynamically voltage scaled (DVS) systems V varies approximately linearly with f ($V \propto f$), the performance-energy tradeoff of frequency scaling can be expressed as

$$T \propto \frac{1}{f} \quad \text{and} \quad E \propto f^2$$

Slowing down the clock frequency will result in energy savings at the cost of decreased performance.

Recent related work has been focused on determining *appropriate* clock frequencies with respect to a predefined *deadline*, using on-line and off-line algorithms [28, 8, 6, 12, 21]. The basic idea is to recognize and eliminate CPU slacks as illustrated in Figure 1. The schedule on the right meets the same deadline but consumes only 1/4 of the energy of the schedule on the left. This figure also illustrates why slowing down the CPU can save more energy than simply shutting it off.

1.2 Our Contributions

In previous work[11], we proposed a simple compile-time model to select the appropriate clock frequency at the cost of tunable performance penalty, and applied the model to a set of kernels to show its effectiveness. This paper extends our previous work to whole programs, and discusses the benefits of voltage and frequency scaling for programs in the SPECfp95 benchmark. Minimizing the power/energy dissipation of scientific computations leads to a reduction in heat dissipation and cooling requirements, which in turn reduces design, packaging, and operation costs of advanced architectures, including power bills for air conditioning of computing and data centers. While there is no opportunity of slowing down the entire programs without incurring a significant performance penalty, we discuss new techniques that identify certain profitable regions in the benchmarks and select appropriate slow-down factors using our model. Simulation results show that the CPU energy savings of 3.97%-23.75% are achieved with an overall performance penalty of at most 2.53%. A simple system energy model consisting of a CPU and memory has similar results. (3.93%-23.25% energy savings across the benchmark).

1.3 Why at the Compiler Level?

Our model is based on quantifying the *imbalance* and *overlap* between CPU and memory activities. In many cases, a compiler is able to not only predict but also *shape* these two factors in a program, giving compilers an advantage over operating systems and hardware techniques. Identifying program regions of large granularity, and assigning efficient voltage and frequency levels to such regions is crucial since the overhead for dynamic voltage and frequency can be significant. Investigating reshape transformation to enable or improve the opportunities for dynamic voltage and frequency scaling is beyond the scope of this paper.

The rest of the paper is organized as follows: Section 2 reviews the simple model proposed in [11], and how it is modified to be region-based. The details of the simulation settings, including the benchmark input data, the modeling system, and the energy models, are described in Section 3 together with the simulation results. Section 4 gives a brief summary of related work, and Section 5 concludes the paper.

2 Compiler-Directed Frequency Scaling

In [11], we proposed to divide the total execution cycles (W) of a program into three segments

$$W = W_c + W_m + W_b$$

with the right-hand side entities defined as follows: W_c, W_m, W_b denote the number of cycles in which the CPU is busy and the memory is idle, the CPU is idle and the memory is busy, and both the CPU and memory are busy, respectively. Using the decomposition, with the assumption that the program in any clock frequency behaves exactly the same for every program step as the program in the default frequency[1], the total execution cycles of the same program (W') when f is reduced by a factor of δ can be estimated as

$$W' = \delta * W_c + \max(W_m + W_b, \delta * W_b)$$

Enforcing

$$W_m + W_b \geq \delta * W_b \qquad (1)$$

we can then rewrite W' as

$$W' = (\delta - 1) * W_c + W \leq W + d \qquad (2)$$

where d is the performance penalty constraint. It is easy to see that W_c has a strong impact on the performance penalty of slowing down the CPU by δ. In addition, even when W_c is relatively small, δ is bounded by the degree of CPU slackness. We combine Conditions (1) and (2) in Condition (3):

$$\delta \leq 1 + \min(d/W_c, W_m/W_b) \qquad (3)$$

In order to avoid a potentially significant performance decrease due to the mismatch of memory and CPU cycle times [11], the model uses the following condition to reflect the clock skew effects during synchronization:

$$\text{memory latency } l \text{ is divisible by } \delta \qquad (4)$$

Finally, we restrict δ to be no less than 1.

$$1 \leq \delta \qquad (5)$$

Conditions (3)-(5) constitute the essence of δ-selection algorithm presented in [11].

Applying this δ-selection algorithm to the SPECfp95 benchmark programs with the performance penalty constraint $d/W = 1\%$ showed that W_c for each benchmark (14.78%-99.92%) is too large to allow a slow down of the whole program with negligible performance impact. A summary of the whole program results is shown in Table 1. For example, in `swim`, Condition (3) leads to

$$\delta \leq 1 + \min(\frac{1\%}{16.25\%}, \frac{68.95\%}{14.80\%}) = 1.06$$

[1] This may not be the case in practice, for instance due to out-of-order instruction execution.

benchmark	W_c/W	W_m/W	W_b/W
tomcatv	33.19%	54.80%	12.00%
swim	16.25%	68.95%	14.80%
su2cor	34.43%	54.19%	11.38%
hydro2d	14.78%	70.50%	14.72%
mgrid	30.54%	58.93%	10.54%
applu	27.09%	60.25%	12.66%
turb3d	53.75%	38.51%	7.74%
apsi	35.47%	54.07%	10.46%
fpppp	99.92%	0.08%	0.00%
wave5	54.57%	35.28%	10.15%

Table 1: The decomposition of the total execution cycles of `SPECfp95` benchmark suite.

(1) Identify a single program regions R: process the program in a top-down fashion based on its structure; if no profitable region can be found at the current level, refine regions and continue search at the refined level.

(2) Model expected performance
  (a) Determine $W_c^{R_i}$, $W_m^{R_i}$, and $W_b^{R_i}$
  (b) Compute slow-down factor δ_i using model discussed in Section 2.1

(3) Insert voltage/frequency setting instructions; adjust performance optimizations, if necessary

Figure 2: Outline of basic compilation strategy.

which prohibits a slow down factor δ of 5/4, 4/3, 2, etc (given memory latency of $l = 100$ cycles). Thus, instead of targeting the entire program, the compiler will apply the δ-selection algorithm to isolated regions of the program. The following section will describe this variant, including the new adjusted conditions for the region-based approach.

2.1 Region-based δ-Selection

Given a program region (R) and the overall performance penalty constraint d, Condition (3) is adjusted as

$$\delta \leq 1 + \min(d/W_c^R, W_m^R/W_b^R) \tag{6}$$

Consider the benchmark `swim` again. There is a region R in the program such that $W^R/W = 32.66\%$, $W_c^R/W^R = 2.99\%$, $W_m^R/W^R = 79.39\%$, and $W_b^R/W^R = 17.62\%$. As a result, we can derive

$$\delta \leq 1 + \min(\frac{1\%/32.66\%}{2.99\%}, \frac{79.39\%}{17.62\%}) = 2.02$$

and slow down the execution of the region by half. Simulation result show that this selection saves 23.2% CPU energy and only degrades the performance by 1.7%.

The algorithm can be extended to multiple regions as well. It can be expressed as a minization problem:

$$\text{minimize } E = \sum_i (\frac{1}{\delta_i})^2 \cdot \frac{W^{R_i}}{W} \tag{7}$$

such that

$$1 \leq \delta_i \leq 1 + W_m^{R_i}/W_b^{R_i},$$
$$\sum_i (\delta_i - 1) W_c^{R_i} \leq d, \quad and$$
$$l \equiv (0 \pmod{\delta_i})$$

where Expression (7) is a simple energy model that can be replaced with more accurate models, if necessary. For instance, a more accurate model may take into account the performance penalty due to voltage/frequency adjustments between regions. Note that the simulation results discussed in Section 3 consider these penalties.

2.2 Basic Compilation Strategy

The basic compilation strategy is shown in Figure 2. Program regions are evaluated in a top-down fashion based on the program structure, starting with the entire program as the single, outermost region. Regions are evaluated based on their expected benefits in terms of power/energy savings. For simplicity, our current approach selects only a single region.

The selected program region will be assigned a single voltage and frequency. Dynamic changes of voltage and frequency will occur only between the region and other portions of the program. The granularity of the region needs to be large enough to compensate for the overhead of voltage and frequency adjustments. Initially, we consider single or sequences of procedure calls, loop nests, and if-then-else constructs as candidate regions.

Different strategies can be used to determine W_c, W_m, and W_b. Static compile-time analysis, on- and offline performance monitoring, or a combination of both. For this paper, we assume that these values are available.

We illustrate our compilation strategy using benchmark `swim`, as shown in Figure 3. For simplicity, we only slow down *one* region if possible. Initially, four regions $R_1 - R_4$ are considered. Knowing the $W_c^{R_i}, W_m^{R_i}, W_b^{R_i}$ values of each region as shown in Table 2, we compute slow-down factor δ_i and potential energy savings (E), and select region R_4 to be slowed down. The next step is to insert voltage/frequency setting instructions at the entry and exit of the region. In our example, an instruction setting CPU speed by half is inserted right before the `IF` and an instruction resuming the full speed is in-

```
R1       CALL INITAL
      90 NCYCLE=NCYCLE+1
R2       CALL CALC1
R3       CALL CALC2
         IF (NCYCLE >= ITMAX) STOP
         IF (NCYCLE <= 1) THEN
            CALL CALC3Z
R4       ELSE
            CALL CALC3
         ENDIF
         GO TO 90
```

Figure 3: The outermost program structure of benchmark `swim` with marked candidate regions.

	R_1	R_2	R_3	R_4
W^R/W	3.50%	31.10%	32.51%	32.66%
W_c^R/W^R	76.82%	17.91%	21.31%	2.99%
W_m^R/W^R	17.65%	66.45%	66/53%	79.39%
W_b^R/W^R	5.53%	15.64%	12.16%	17.62%
δ_i	4/3	10/9	10/9	2
E	98.47%	94.09%	93.82%	75.51%

Table 2: The decomposition of total execution cycles for regions of benchmark `swim`.

serted right after the `ENDIF`.

3 Experiments

3.1 SPECfp95 Benchmarks

We used SPECfp95 for our experiments. To reduce simulation time, we took as input Burger's standard data sets (**std**) [3] that have as few instructions as possible while retain the behavior of the reference data sets.

3.2 Simulation Settings

All simulations are done through the SimpleScalar tool set [4], with memory hierarchy extensions [5]. SimpleScalar provides a cycle-accurate simulation environment for modern out-of-order superscalar processors with 5-stage pipelines and fairly accurate branch prediction mechanism. While the original version supports a multi-level non-blocking memory subsystem and captures limited memory bandwidth, the extensions model the limitedness of non-blocking caches through finite miss status holding registers (MSHRs) [14]. Bus contention and arbitration at all levels are also taken into account. What is not considered are multi-bank memory organization, page hits versus misses, precharging overhead, and refresh cycles. Figure 4 gives the simulation parameters used in the paper.

Simulation parameters	Value
frequency	1 GHz
fetch width	4 instructions/cycle
decode width	4 instructions/cycle
issue width	4 instructions/cycle, out-of-order
commit width	4 instructions/cycle
RUU size	64 instructions
LSQ size	32 instructions
FUs	4 intALUs, 1 intMULT, 4 fpALUs, 1 fpMULT, 2 memports
branch predictor	gshare, 17-bit wide history
L1 D-cache	32KB, 1024-set, direct-mapped, 32-byte blocks, LRU, 1-cycle hit, 8 MSHRs, 4 targets
L1 I-cache	as above
L1/L2 bus	256-bit wide, 1-cycle access, 1-cycle arbitration
L2 cache	512KB, 8192-set, direct-mapped, 64-byte blocks, LRU, 10-cycle hit, 8 MSHRs, 4 targets
L2/mem bus	128-bit wide, 4-cycle access, 1-cycle arbitration
memory	100-cycle hit, single bank
TLBs	128-entry, 4096-byte page
compiler	`gcc 2.7.2.3 -O3 -funroll-loops`

Figure 4: System simulation parameters.

We added a new instruction into SimpleScalar's ISA, which takes an explicit value for the new CPU frequency and implements the following 3-step semantics: (1) stop fetching new instructions and wait until CPU enters the *ready* state, i.e., the frequency scaling instruction is not speculative, the pipeline is drained, all functional units are idle, and all pending memory requests are satisfied, (2) wait a fixed amount of cycles to model the process of scaling up/down to the new frequency, and (3) resume the course using the new frequency. Each step has an associated performance penalty. In the simulation we set the step (2) cost as 10,000 cycles (10 μs for a 1GHz processor).

3.3 Analytical Energy Models

Due to the long simulation time, we use three simple analytical energy models to access the benefits gained from our compilation strategy. They model active CPU energy, total CPU energy, and total system energy. All these models are based on the same simple idea of associating with each cycle an energy cost.

Given a program in which region R is slowed down by δ, we introduce four "component" models:

$$E_1 = (W_c + W_b) - (1 - 1/\delta^2) \cdot (W_c^R + W_b^R)$$

$$
\begin{aligned}
E_2 &= \rho_i^{\texttt{cpu}} \cdot W_m \\
E_3 &= \rho_{m/c} \cdot (W_b + W_m)/l \\
E_4 &= \rho_i^{\texttt{mem}} \cdot W_c/r
\end{aligned}
$$

where E_1 models the total CPU energy usage when the CPU is active, E_2 models the total CPU energy usage when the CPU is idle, E_3 models the total memory energy usage when memory is active, and E_4 models the total memory energy usage when memory is idle. Parameter r is the refresh cycles, l is the memory latency, and $\rho_i^{\texttt{cpu}}, \rho_i^{\texttt{mem}}, \rho_{m/c}$ are particular ratios with respect to the energy cost of an active CPU cycle. In our evaluation, we set $\rho_i^{\texttt{puc}} = 30\%$, $\rho_{m/c} = 2$, $\rho_i^{\texttt{mem}} = 2$, and $r = 10{,}000$, considering memory consuming 2/3 of total energy.

3.4 Experimental Results

The profitable candidate regions of benchmarks are identified through simulating our basic compilation strategy by hand. The implementation of the proposed strategy is currently underway. For simplicity, we slowed down *one* region in each benchmark. Therefore, the experimental results are not necessarily optimal. The simulation results are shown in Table 3.

All benchmarks, except `fpppp`, can be slowed down to save 2.44%-16.09% of active CPU energy (E_1), 3.97%-23.75% of total CPU energy ($E_1 + E_2$), and 3.93%-23.25% of the overall system energy ($E_1+E_2+E_3+E_4$), at the performance penalty of 0.77%-2.53%. Benchmark `fpppp` cannot benefit from our compiler strategy since it is extremely CPU-bound.

The cost of scaling up/down the voltage and frequency is linearly proportional to the number of repetitions in **std** data sets. Given that benchmarks execute 0.73-15.62 billions of cycles and the repetitions is in the range of 2-62, this dynamic scaling cost turns out to be insignificant. Most of performance degradation comes from the impact of CPU slow-down to the program execution.

4 Related Work

There have been efforts in building up microprocessors capable of dynamic voltage and frequency scaling (DVS), such as Transmeta's Crusoe, Intel's SA-2, and [2, 23]. Besides saving power and energy at the expense of performance loss, the scaling overheads can take as long as 520μs ([2]) or 140μs ([23]), which suggests the coarse speed control may be appropriate.

For DVS-capable processors, the *voltage scheduling* problem involves deciding when to scale them up/down and by how much. New scheduling algorithms at the operating system level have been proposed, either task-based [29, 12, 10, 20, 17, 25, 27] or interval-based [28, 8, 22, 26]. Grunwald et al. [9] evaluated some of the interval-based schedulers through actual measurements and observed noticeable performance loss.

Most inter-task scheduling algorithms use worst-case assumption and thus fail to further exploit the CPU slackness from actual execution. As a result, intra-task scheduling algorithms [13, 19, 15, 24] are advocated. Both [19] and [24] use compile-time analysis to instrument the program to adapt to real execution behavior.

Other ways to optimize software for low power are possible. For example, Ghiasi et al. [7] used specified IPC rates to reconfigure processors, while Marculescu [18] determined scaling points at micro-architectural level. For further information, please refer to two excellent surveys [16, 1].

5 Conclusion and Future Work

Dynamic frequency and voltage scaling is an effective way to reduce power dissipation and energy consumption of memory-bound program regions. This paper discussed a simple performance model that allows the selection of efficient slow-down factors. Experiments based on the full SPECfp95 benchmark set and a simulator for an advanced superscalar architecture indicate the effectiveness of the model. The resulting CPU energy savings of our compilation strategy are in the range of 3.97%-23.75% with a performance slow-down of 0.77%-2.53%. The energy savings of the whole system case are similar.

We are currently investigating the impact of aggressive compiler optimizations for program performance on the slow-down opportunities, and possible program reshaping transformations to enable or enhance slow-down opportunities. More benchmarking will be done to further access the benefit of our compilation strategy across a range of applications. Finally, we want to point out that the slow-down opportunity is based on the program imbalance and therefore slowing down the memory subsystem to allow more energy savings in computation-bound program regions is also possible.

Acknowledgements

The authors wish to thank Professor Doug Burger from the University of Texas at Austin for providing the SimpleScalar tool set with his memory extensions.

References

[1] L. Benini and G. Micheli. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems*, 5:115–192, April 2000.

[2] T. Burd and R. Brodersen. Design issues for dynamic voltage scaling. In *Proceedings of 2000 International Symposium on Low Power Electronics and Design (ISLPED'00)*, July 2000.

benchmark	W^R/W	W_c^R/W^R	W_m^R/W^R	W_b^R/W^R	δ	T	E_1	E_1+E_2	$E_1+E_2+E_3+E_4$
tomcatv	39.58	1.76	81.40	16.84	2	101.99	86.82	76.25	76.75
swim	32.66	2.99	79.39	17.62	2	101.68	83.91	76.79	77.53
su2cor	21.20	17.16	67.57	15.27	5/4	100.77	94.10	92.06	92.22
hydro2d	31.56	9.57	74.61	15.82	4/3	101.47	87.95	84.61	85.11
mgrid	10.04	10.02	75.54	14.43	2	101.61	95.53	93.43	93.58
applu	23.52	7.29	76.28	16.42	4/3	101.82	93.96	90.43	90.68
turb3d	21.74	12.67	74.75	12.58	4/3	101.52	96.15	92.83	92.92
apsi	7.44	3.65	82.43	13.92	4	102.53	97.56	95.26	95.36
wave5	11.42	25.73	61.21	13.06	4/3	101.15	97.45	96.03	96.07

Table 3: Experimental results for SPECfp95. The values of slow-down factor δ are for the single selected region R. The four rightmost columns report the impact of the slowed down region R on the overall program performance in terms of relative execution time T and relative energy savings (original 100%).

[3] D. Burger. *Hardware Techniques to Improve the Performance of the Processor/Memory Interface*. PhD thesis, Computer Sciences Department, University of Wisconsin-Madison, 1998.

[4] D. Burger and T. Austin. The SimpleScalar tool set version 2.0. Technical Report 1342, Computer Science Department, University of Wisconsin, June 1997.

[5] D. Burger, A. Kägi, and M. Hrishikesh. Memory hierarchy extensions to SimpleScalar 3.0. Technical Report TR99-25, Department of Computer Science, University of Texas at Austin, April 1999.

[6] J. Chang and M. Pedram. Energy minimization using multiple supply voltages. In *International Symposium on Low Power Electronics and Design (ISLPED-96)*, pages 157–162, August 1996. published in IEEE Transaction on VLIS Systems 5(4): Dec 1997.

[7] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Workshop on Complexity Effective Design*, June 2000.

[8] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *the 1st ACM International Conference on Mobile Computing and Networking (MOBICOM-95)*, pages 13–25, November 1995.

[9] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI-2000)*, October 2000.

[10] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *Proceedings of the 35th ACM/IEEE Design Automation Conference(DAC'98)*, pages 176–181, June 1998.

[11] C.-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic frequency and voltage scheduling. In *Workshop on Power-Aware Computer Systems (PACS)*, November 2000.

[12] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design (ISLPED-98)*, pages 197–202, August 1998.

[13] C.M. Krishna and Y.-H. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proceedings of the 6th Real Time Technology and Applications Symposium (RTAS'00)*, May 2000.

[14] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *Proceedings of the 18th International Symposium on Computer Architecture (ISCA-81)*, pages 81–87, May 1981.

[15] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proceedings of the 37th Conference on Design Automation (DAC'00)*, pages 806–809, June 2000.

[16] J. Lorch and A. Smith. Software strategies for portable computer energy management. *IEEE Personal Communications Magazine*, 5(3), June 1998.

[17] A. Manzak and C. Chakrabarti. Variable voltage task scheduling for minimizing energy or minimizing power. In *Proceeding of the International Conference on Acoustics, Speech and Signal Processing*, June 2000.

[18] D. Marculescu. On the use of microarchitecture-driven dynamic voltage scaling. In *Workshop on Complexity-Effective Design*, June 2000.

[19] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compiler and Operating Systems for Low Power (COLP'00)*, October 2000.

[20] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. In *Proceedings of the 12th International Symposium on System Synthesis (ISSS'99)*, 1999.

[21] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of 1998 International Symposium on Low Power Electronics and Design (ISLPED'98)*, pages 76–81, August 1998.

[22] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lpARM microprocessor system. In *Proceedings of 2000 International Symposium on Low Power Electronics and Design (ISLPED'00)*, pages 96–101, July 2000.

[23] J. Pouwelse, K. Langendoen, and H. Sips. Voltage scaling on a low-power microprocessor. In *International Symposium on Mobile Multimedia Systems & Applications (MMSA'2000)*, November 2000.

[24] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. In *To appear in IEEE Design and Test of Computers*, March 2001.

[25] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'00)*, pages 365–368, November 2000.

[26] A. Sinha and A. Chandrakasan. Dynamic voltage scheduling using adpative filtering of workload traces. In *Proceedings of the 14th International Conference on VLSI Design*, January 2001.

[27] V. Swaminathan and K. Chakrabarty. Investigating the effect of voltage switching on low-energy task scheduling in hard real-time systems. In *Asia South Pacific Design Automation Conference (ASP-DAC'01)*, January/February 2001.

[28] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *the 1st Symposium on Operating Systems Design and Implementation (OSDI-94)*, pages 13–23, November 1994.

[29] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Annual Symposium on Foundations of Computer Science*, pages 374–382, October 1995.