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

Chung-Hsing Hsu
CS Dept., Rutgers University
New Jersey, USA

Ulrich Kremer
CS Dept., Rutgers University
New Jersey, USA

Michael Hsiao
ECE Dept., Rutgers University
New Jersey, USA

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.

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.

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.

*Authors' email: {chunghsu,uli}@cs.rutgers.edu and mhsiao@ece.rutgers.edu.

ISLPED'01, August 6-7, 2001, Huntington Beach, California, USA.

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 trade-off of frequency scaling can be expressed as $T \propto \frac{1}{f}$ and $E \propto f^2$.

Recent related work has been focused on determining *appropriate* clock frequencies with respect to a predefined *deadline*, using on-line and off-line algorithms [23, 6, 5, 10, 16]. The basic idea is to recognize and eliminate CPU slacks.

1.2 Why at the Compiler Level?

Our model is based on quantifying the *imbalance* and *overlap* between CPU and memory activities. In many cases, compile-time analysis is able to determine these two factors and identify program regions where dynamic voltage and frequency scaling will be profitable. Since the runtime overhead of dynamic voltage/frequency adjustment can be significant, global program analyses are needed to detect program regions of sufficiently large granularity. Typically, hardware and operating system techniques are restricted to observe and evaluate program behavior within a restricted window of past program activity. In addition, compilers can reshape program behavior through local and global transformations, enabling additional program optimizations. The discussion of such "reshape" transformations is beyond the scope of this paper.

In a single user environment, the compiler generated voltage and frequency adjustment instructions are directly executed, assuming that the program is not preempted. In a multiprogramming environment where swapping may occur, the compiler can provide an application profile to the operating system that contains information about all requested frequency and voltage transitions during a program execution, and leave it to process scheduler to negotiate between the different process requirements.

1.3 Our Contributions

In previous work[8], 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

(1) Identify program regions R_i as candidates for dynamic voltage/frequency scaling.

(2) Model expected performance
  (a) Determine $W_c^{R_i}$, $W_m^{R_i}$, and $W_b^{R_i}$, for instance by using program execution traces.
  (b) Compute slow-down factor δ_i.

(3) Select single region with highest predicted benefit.

(4) Insert voltage/frequency setting instructions.

Figure 1: Outline of basic compilation strategy.

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

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.

While there is no opportunity of slowing down entire programs in the SPECfp95 benchmark 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).

The rest of the paper is organized as follows: Section 2 reviews the simple model proposed in [8], and how it is modified to be region-based. Section 3 discusses our experimental results, followed by a brief summary of related work and a conclusion in Sections 4 and 5, respectively.

2. COMPILER DIRECTED FREQUENCY SCALING

In [8], we proposed a way to determine the CPU slow-down factor δ with respect to performance penalty d, as follows:

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

$$\text{memory latency } l \text{ is divisible by } \delta \qquad (2)$$

where W_b, W_c, W_m represent the duration in which CPU activity (including L1 and L2 activity) is overlapped with memory access, is *not* overlapped with memory access, and is stalled due to memory access. Equation (2) takes into account the clock skews due to mismatch of memory and CPU speed.

Applying this δ-selection algorithm to the SPECfp95 benchmarks with the 1% performance penalty showed that slowing down the entire program is not profitable for any benchmark [9]. Thus, the compiler will apply the algorithm to program regions instead. For example, in benchmark `swim` there is a region R_4 such that our algorithm selects $\delta = 2$. And simulation result show that this slow-down saves 23.2% of total CPU energy and only degrades the performance by 1.7%.

Typically, a program consists of multiple program regions. The selection algorithm needs to be able to select a *subset* of them to slow down without introducing too much overhead due to switches between different voltages/frequencies. In addition, it needs to assign different slow-down factors to different selected regions so as to maximize the overall energy savings without violating the global performance penalty constraint. The first part will be sketched in the following section while the second section is beyond the scope of this paper and we refer the readers to [9] for further information.

Finally, we want to point out that various strategies can be used to determine W_c, W_m, and W_b, such as static compile-time analysis, on- and off-line performance monitoring, or a combination of both. We are currently evaluating different mechanisms to determine these values in terms of their cost/precision tradeoffs. For this paper, we assume a trace based approach with a trace generated by a characteristic program execution.

2.1 Basic Compilation Strategy

The basic compilation strategy is shown in Figure 1. 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.

We illustrate our compilation strategy using benchmark `swim`, as shown in Figure 2.1. Four regions $R_1 - R_4$ are considered. Based on the $W_c^{R_i}, W_m^{R_i}, W_b^{R_i}$ values of each region as shown in Table 2.1, 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 inserted right after the `ENDIF`.

3. EXPERIMENTS

Experiments were done through simulation, using the SimpleScalar tool set [4]. The simulator models an out-of-order superscalar processor and a memory subsystem that captures the impact of memory bandwidth and memory latency. Specifically, the memory system contains a 32KB L1 I-cache

Table 1: The decomposition of total execution cycles for regions of benchmark `swim`.

	R_1	R_2	R_3	R_4
W/W	3.50%	31.10%	32.51%	32.66%
W_c/W^R	76.82%	17.91%	21.31%	2.99%
W_m/W^R	17.65%	66.45%	66/53%	79.39%
W_b/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%

and D-cache and a 512KB L2 cache. All caches are direct-mapped, write-back, and can hold up to eight outstanding misses. The main memory is blocking, with 100-cycle access latency, and runs at 1/4 of the processor speed. Details can be found in [9].

The overheads of switching between voltages/frequencies are also modeled. A voltage/frequency setting instruction first stops fetching new instructions and drains the pipeline, followed by waiting until the desired frequency is generated, and then resume the execution. In our experiments, the period of scaling up/down to the new frequency is set to be a constant of 10,000 cycles (10 μs for a 1GHz processor) [1].

3.1 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 associating with each cycle an energy cost. Specifically, given a program in which region R is slowed down by δ, we introduce four "component" models:

$$\begin{aligned} E_1 &= (W_c + W_b) - (1 - 1/\delta^2) \cdot (W_c^R + W_b^R) \\ E_2 &= \rho_i^{\texttt{cpu}} \cdot W_m \\ E_3 &= \rho_{m/c} \cdot (W_b + W_m)/l \\ E_4 &= \rho_i^{\texttt{mem}} \cdot W_c/r \end{aligned}$$

where E_1 and E_2 model the CPU energy usage when it is active and idle, respectively; E_3 and E_4 model the memory energy usage when it is active and idle, respectively. 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.

3.2 Experimental Results

SPECfp95 are used for the experiments. To reduce simulation time while retain the behavior of the reference data input, we used Burger's standard data sets (**std**) [3]. And our compilation strategy was applied by hand to all the benchmarks. The implementation of the proposed strategy is currently underway. Since we slowed down at most *one* region in each benchmark, the experimental results are most likely suboptimal. The simulation results are shown in Table 3.2.

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.

[1]The scaling period for DVS-capable processors can take as long as 520μs ([2]) or 140μs ([18])

4. RELATED WORK

On the hardware side, there have been efforts in building up microprocessors capable of dynamic voltage and frequency scaling (DVS), such as Transmeta's Crusoe, Intel's Xscale, and [2, 18]. On the software side, new scheduling algorithms at the operating system level have been proposed, either task-based (for example, [24, 10, 7, 15, 13, 20, 22]) or interval-based (for example, [23, 6, 17, 21]). Early task-based scheduling algorithms focus on the CPU slackness between tasks. Recently, intra-task scheduling algorithms [11, 14, 12, 19] are advocated.

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 our approach. 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 on the slow-down opportunities, possible program reshaping transformations to enable or enhance slow-down opportunities, and extensions of our approach to deal with CPU bound computations with the opportunity of slowing down the memory subsystem. An implementation of the proposed strategy as part of the SUIF compiler infrastructure [1] is currently underway.

6. ACKNOWLEDGEMENTS

This research was partially supported by NSF CAREER award CCR-9985050 and a Rutgers University ISC Pilot Project grant. The authors would like to thank Doug Burger from UT Austin for providing the SimpleScalar tool set with his memory extensions.

7. REFERENCES

[1] National Compiler Infrastructure (NCI) project. www-suif.stanford.edu/suif/nci, 1998.

[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.

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

Table 2: 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%).

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

[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] 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, Aug. 1996. published in IEEE Transaction on VLIS Systems 5(4): Dec 1997.

[6] 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, Nov. 1995.

[7] 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.

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

[9] C.-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. Technical Report DCS-TR-431, Department of Computer Science, Rutgers University, Feb. 2001.

[10] 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, Aug. 1998.

[11] C. 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.

[12] 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.

[13] 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.

[14] 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)*, Oct. 2000.

[15] 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.

[16] 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, Aug. 1998.

[17] 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.

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

[19] 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*, Mar. 2001.

[20] 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, Nov. 2000.

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

[22] 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.

[23] 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, Nov. 1994.

[24] 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, Oct. 1995.