Compiler-Directed Dynamic Voltage Scaling for Memory-Bound Applications

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

DCS-TR-498

August 2002

Abstract

This paper presents the design and implementation of a compiler algorithm that effectively reduces the energy usage of memory-bound applications via dynamic voltage scaling (DVS). The algorithm identifies program regions where the CPU can be slowed down with negligible performance penalty. It is implemented as a source-to-source level transformation using the SUIF2 compiler infrastructure. Physical measurements on a laptop with a 600 MHz - 1.2 GHz AMD Athlon 4 processor show that CPU energy savings in the range of 9.17% to 55.65% can be achieved with performance degradation in the range of 0.69% to 6.14% for the `SPECfp95` *benchmarks. On average, the energy and energy-delay product are reduced by 26.58% and 24.11%, respectively, at the cost of the performance slowdown of 3.26%. This paper also discusses a new methodology which attempts to approximate the minimum energy usage by any DVS algorithm. Our compiler-directed DVS algorithm is within 6% from the "optimal" case. To the best of our knowledge, this is one of the first work that evaluates DVS strategies by physical measurements.*

1. Introduction

Power dissipation has always been a crucial issue in the design of embedded, hand-held and portable devices, often more important than performance. More recently, power has become a first-class design constraint in the world of high-performance computing systems as well, such as workstations and server farms. Limiting power dissipation leads to longer battery lifetime, less cooling requirements, and higher system reliability. As a consequence, the costs of design, packaging, and operation, including power bills for air conditioning, are significantly reduced. For example, a server farm with 8,000 servers consumes 2 megawatts which account for 25% of operation cost [28].

Besides the efforts in designing lower-power circuitry, a common approach for power reduction is to put idle resources or devices into a hibernation state until they have to be accessed again. A complementary approach, called *dynamic voltage scaling* (DVS), reduces power even when these resources are still active. Dynamic voltage scaling exploits the fact that a major portion of power of CMOS circuitry scales quadratically with the supply voltage [7]. Lowering the voltage can therefore reduce power dissipation. However, lowering the voltage increases circuit delay. As a result, DVS reduces power at the expense of performance.

In this paper, we focus on reducing the energy consumption of non-interactive applications running on laptops with microprocessors that support DVS. Recent advances in fast, low-power microprocessors are increasing the number of potential applications for these computers. Non-interactive applications, such as movie playing, decompression, or encryption, are now commonly run on laptops. For most of these non-interactive applications, fast processors reduce device idle times, which in turn reduce the opportunities for energy savings through hibernation strategies. However, DVS technique may still be beneficial in such cases.

A key factor for the success of DVS in non-interactive applications is when the CPU performance is not on the critical path. This *program imbalance* provides great opportunites to apply DVS effectively. However, the transitions to and from the low performance levels can consume significant time and energy. For example, it takes about 20 microseconds for an Intel Xscale-based processor to switch frequencies [28]. For a high-performance processor, such a transition time can be translated into tens of thousands of instructions. The goal for a good DVS algorithm is to deter-

*This research was partially supported by NSF CAREER award CCR-9985050.

mine when to adjust the current performance level (*scaling points*) and to where (*scaling factors*) so that a considerable saving in energy can be achieved while still delivering the required performance.

In this paper we focus on applications where the memory performance is the bottleneck, not the CPU performance. Applications in this category include database applications [9], image processing algorithms [42], speech recognition systems [1], and scientific applications [17]. Due to the technology advances in both power and performance for laptops, we expect more and more such applications to be executed on laptops, making them de facto "mobile" workstations.

This paper presents the design and implementation of a compiler-directed DVS algorithm that effectively reduces the energy consumption of memory-bound applications. The idea is to identify those program regions for which the CPU is mostly idle due to memory stalls, and slow them down. On an architecture that allows the overlap of the CPU and memory activities, such slow-down will not result in serious performance degradation. Unlike many multimedia applications, the applications we tested do not have explicit deadlines. Therefore, we have chosen an "acceptable" overall performance penalty between 0% and 5% in our experiments. Note that 5% is an input parameter of the algorithm and can be changed. Overheads due to transitions between different frequency/voltage levels are also taken into account in our approach. The main contributions of this paper include

- the design and implementation of a compiler-directed DVS algorithm, and the evaluation of its effectiveness on a real laptop machine through physical measurements. To the best of our knowledge, this is one of the first implementation of a compiler-directed DVS algorithm.

- physical measurements showing that CPU energy savings in the range of 9.17% to 55.65% can be achieved with performance degradation in the range of 0.69% to 6.14% for the `SPECfp95` benchmarks on a state-of-the-art, high performance laptop system. On average, reductions of the energy consumption and energy-delay product of 26.58% and 24.11%, respectively, were observed, with a performance penalty of 3.26%.

- the development of a new methodology to approximate the lower bound in energy usage for any DVS algorithm, whether hardware, OS, or compiler directed. Using this methodology, we find that our compiler-directed DVS algorithm reduces energy consumption within 6% from the (approximated) "optimal".

The rest of the paper is organized as follows: Section 2 describes some of the previous work. Section 3 discusses in detail the design and implementation of our compiler-directed DVS algorithm. The experiment setup and results are discussed in Section 4, followed by conclusions and future work in Sections 5.

2. Related Work

Besides the continuous technology advances in the low-power high-performance circuit design, there has been a substantial increase in the amount of the research directed towards power-aware microarchitectures [5]. These power-aware resources provide multiple power modes (or configurations) with various power-performance trade-offs. The key is to intelligently manage the many power modes of the resources to tailor for the particular demand with the minimum amount of energy. Benini and Micheli provide a good tutorial survey for some of the issues [4].

Proposing a new design or management policy is one thing, evaluating its power reduction is another. It is difficult, or even impossible in some cases, to perform physical measurements for the evaluation. As a result, simulators are used instead, ranging from the low-level Hspice simulator to the architecture-level simulators such as Wattch [6], SimplePower [41], SoftWatt [15], and AccuPower [32]. Each simulator has an associated power model. Therefore, the accuracy of the evaluated results relies on the accuracy of the power model in a simulator.

There has been a few attempts in using the physical measurement for the evaluation. In the context of applying DVS for CPU power reduction, to the best of our knowledge, Pillai and Shin [31] was the first who evaluated their DVS algorithms, as a Linux kernel module, on a laptop with a 550 MHz AMD K6-2+ processor. Our work in this paper deals with the next generation of the AMD chip, and is one of the first working implementation of DVS algorithms applied to a high-performance laptop computer.

Existing DVS algorithms can be categorized in many ways. A DVS algorithm can be performed either off-line, on-line, or both, depending on when the scaling points and scaling factors are determined. A DVS algorithm can target general-purpose applications, time-critical applications such as real-time applications and multimedia applications, or interactive applications. These applications differ in how explicit the timing constraints are. A lot of work has been done on the off-line DVS algorithms for real-time operating systems, (e.g., [36, 13, 24, 33, 31, 39, 34]) and on-line DVS algorithms specialized for interactive and multimedia applications (e.g., [40, 12, 29, 14, 11, 23, 10, 37]). Our work in this paper proposes an off-line DVS algorithm for general-purpose memory-bound applications.

The closest work to ours are the algorithms based on checkpoints. In a checkpoint-based DVS algorithm, the scaling points are identified off-line and the scaling factors are calculated on-line [22, 27, 36, 2]. Many such algorithms place scaling points at selected branches to exploit the slacks due to run-time variations. Our algorithm identifies a *different* type of CPU slacks, namely the idle time due to memory stalls.

Our work adopts a table-driven approach similar to the one presented by Saputra et al. [34]. However, the table entries in our algorithm only store the performance information, while theirs store the energy information as well. We derive the energy information from an analytical model. The work in this paper is also significantly different from our previous work in [19, 18]. In the previous work, an analytical performance prediction model is used. As a result, the effectiveness of the DVS algorithm depends on how well the performance model predicts the target architecture. In this work we treat the target architecture as a black box. The performance information is derived through profiling.

Besides the cost of DVS transitions, several other system-level issues have been addressed in practical implementations of DVS algorithms. For example, the issues of total system power, nonideal battery capacity, and nonideal memory behavior are identified [3, 25]. Recently, Miyoshi et al. [26] found that some operating frequencies of a processor consume more energy than their higher frequencies. In addition, only a few operating frequencies are available in modern processors.

Finally, combining DVS and other power reduction techniques is under investigation. For example, Hughes et al. [21] proposed an algorithm to control both architectural adaptation and DVS for multimedia applications. Huang et al. [20] applied different low-power techniques to different sections of an application. Semeraro et al. divided a processor into four clock domains within which independent DVS can be performed [35].

3. The Algorithm

Our compiler-directed DVS algorithm looks into a set of program regions to select a region that can be slowed down to minimize the expected energy consumption while within the user-provided performance tolerance. DVS system calls are then inserted at the boundaries of the selected region. These system calls define the scaling points of program execution. The scaling factors are computed by the algorithm and passed as input parameters to DVS system calls. The performance penalty of DVS system calls are considered. It is proportional to the number of times the selected region is entered.

A program region in our algorithm is defined as a single entry and single exit program structure. It includes a loop nest, a procedure or function call, a procedure or function body, a sequence of statements, or even the entire program.

Phase 1: Instrumentation
Instrument program regions for their execution cycles and the number of times being visited.
Phase 2: Profiling
Collect region profile at various voltages and frequencies.
Phase 3: Region Selection
Enumerate all program regions R. Determine the appropriate voltage and frequency for each of them so that
- its expected performance penalty is no greater than the user's tolerance, and
- its expected energy usage is minimized.

Find the region R^{best} that has the lowest expected energy usage.
Phase 4: Code Generation
Insert DVS system calls at the boundaries of R^{best}.

Figure 1. The compiler algorithm to select a region for DVS.

This definition simplifies the calculation of the DVS overhead.

The prototype for our algorithm is profile-driven and implemented as part of the SUIF2 compiler infrastructure [38]. The prototype has four phases, as shown in Figure 1. The first phase instruments the original program at selected program locations. The instrumented code is then executed (the second phase), collecting information needed in the third phase. The third phase uses both the instrumented program and the profile information to select a single program region to be slowed down. In the final phase, the corresponding DVS system calls are inserted. Some of the details can be found in [16, 18].

3.1. Region Selection

Phase 3 in our prototype has the following goal in mind: find a region R with voltage V_R and frequency f_R for the following minimization problem.

minimize $P_R \cdot \mathcal{T}(R, V_R, f_R) - P_{max} \cdot \mathcal{T}(R, V_{max}, f_{max})$
subject to

$$\mathcal{T}(R, V_R, f_R) - \mathcal{T}(R, V_{max}, f_{max}) + 2 \cdot \mathcal{N}(R) \cdot T_{trans} \leq r \cdot T_{\mathrm{ORG}}$$

where $\mathcal{T}(R, V, f)$ stands for the execution time of region R at voltage V and frequency f, $\mathcal{N}(R)$ stands for the number of times region R is visited in the profile run, $P = cV^2 f$

stands for the power dissipation of the processor at voltage V and frequency f, r stands for the user-provided performance tolerance in terms of the relative performance, T_{trans} stands for the time in a single DVS transition, and T_{ORG} stands for the total execution time when the program is run at the maximum voltage V_{max} and frequency f_{max}.

The formulation simply says, if we slow down the region R with a lower voltage and/or frequency, we introduce the performance penalty due to longer execution time of the region $\mathcal{T}(R, V_R, f_R) - \mathcal{T}(R, V_{max}, f_{max})$, and due to the accumulated transition time $2 \cdot \mathcal{N}(R) \cdot T_{trans}$. We have to make sure that the overall performance penalty does not exceed the user's tolerance $r \cdot T_{\mathrm{ORG}}$. If a region satisfies the timing constraint, we evaluate its net energy savings $P_R \cdot \mathcal{T}(R, V_R, f_R) - P_{max} \cdot \mathcal{T}(R, V_{max}, f_{max})$. Phase 3 enumerates all candidate regions to find the region that has the largest net energy savings while satisfying the timing constraint.

This implementation is similar to the one in [18]. Both use analytical model to compute the expected CPU energy usage. However, the mechanism for calculating the expected performance penalty is new. Instead of using an analytical model to estimate $\mathcal{T}(R, V, f)$, this implementation adopts the table-driven approach similar to [34]. More specifically, for each region R and each valid (V, f) pair, the region is executed and its execution time is stored into the table. Note that the actual implementation does not construct the table $\mathcal{T}(R, V, f)$ on a per-region basis, but on a per-(V, f) basis. Only a subset of table entries are profiled. The rest are computed in an additional phase before Phase 3 (not shown in Figure 1). A similar technique is applied to compute the table $\mathcal{N}(R)$.

4. Experiments

4.1. Hardware Platform

The hardware platform is a Compaq (now Hewlett-Packard) Presario 715US laptop computer. It has a 1.2 GHz mobile AMD Athlon 4 processor, 200 MHz system bus, 256 MB PC-133 SDRAM, 20 GB hard disk, a combined CD-RW/DVD-ROM, a 4x AGP graphics card with 16 MB graphics RAM, a sound system, and a 14.1 inch LCD display. The processor is a 3-way superscalar out-of-order decode/execution-decoupled engine with dynamic branch prediction and speculative execution. It contains a 64KB instruction cache, a 64KB data cache, a full-speed on-die 256 KB level two exclusive cache, and a hardware data prefetching unit.

The processor supports DVS under software control by writing the desired voltage and frequency into a machine-specific register. For the Compaq Presario 715US laptop, the frequency of 300 MHz to 1200 MHz in 50 MHz increments is allowed, at the voltage ranging from 1.15 V to 1.45 V. However, not all the combinations of voltage and frequency result in stable and correct computation. We determined the valid pairs experimentally, using a set of CPU-intensive benchmarks to check the stability and correctness. We call these pairs *performance levels*. Table 1 gives the seven performance levels used in the experiments, along with their associated frequencies and voltages.

level	f (GHz)	V (volts)
1	0.6	1.15
2	0.7	1.20
3	0.8	1.25
4	0.9	1.30
5	1.0	1.35
6	1.1	1.40
7	1.2	1.45

Table 1. The seven performance levels used in the experiments.

4.2. Software Platform

The laptop was installed with the Linux 2.4.18 kernel. The choice of applications is the `SPEC95` floating-point benchmarks. They were chosen because they tend to be more memory-bound than the `SPEC95` integer benchmarks, and exercises the CPU more than the multimedia applications. The training input set comes from a reduced reference input set proposed by Burger [8]. In most benchmarks, the reduced reference input set uses the same data as the original one but decreases the number of control loop iterations. All the benchmarks were compiled with `g77 -O2`.

We have implemented DVS as a user-level system call. The input of a call is the desired frequency. The call will find the corresponding voltage according to Table 1, and write them to the machine-specific register that handles DVS. For our DVS algorithm, we also implemented another user-level system call. This system call reads out the current cycle counter value on a per-process basis. It is done through a performance counter interface written by Pettersson [30]. It basically serves as a high-resolution timer. Our DVS algorithm needs this to measure the elapsed time between two DVS system calls. Table 2 specifies the input parameters of our DVS algorithm for the experiments.

Finally, we want to mention that AMD implements an on-line DVS algorithm, called *PowerNow!*. On the Compaq Presario 715US laptop, it provides four performance levels. Unfortunately, we cannot compare it against our DVS algorithm since we do not have access to the Linux version of

parameter	value
$\mathcal{T}(R, V, f)$	profiled
$\mathcal{N}(R)$	profiled
$\{ (V, f) \}$	Table 1
T_{trans}	20 μs
r	5%

Table 2. The input parameters for our algorithm.

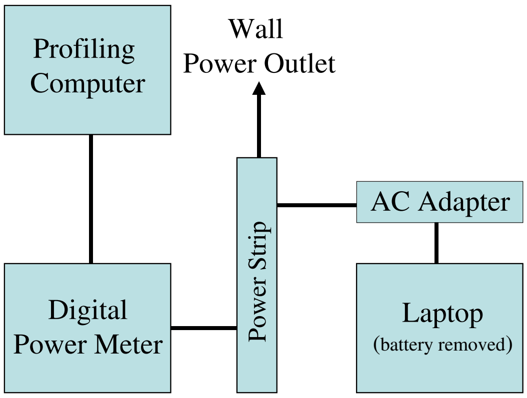

Figure 2. The experimental setup.

it.

4.3. Measurements and Metrics

We performed several experiments with our DVS algorithm and measured the actual energy consumption of the system through a digital power meter. The power meter, a Yokogawa WT110, collected instantaneous power measurements 3-4 times per second and sent these measurements to another computer, which stores them in a log for later use. All the programs were run to completion with the reference input, and executed in the range of 70-140 seconds. During the measurements, the battery was removed, and the external AC adapter was the only power source. The monitor was on, and the operating system was in console mode. Each benchmark was run four times and the measurement with the minimum execution time is reported. Figure 2 shows the setup.

We have observed that for all the benchmarks running at any performance level, the measured power is pretty stable. There are occasional increases in 1-3 watts due to disk accesses but very rare for our benchmarks. For example, Figure 3 shows the measured system power at every sample for a very memory-bound benchmark `swim` running at the peak performance level. The `SPECfp95` benchmarks were designed to be the compute-intensive applications that emphasize the performance of the processor and the main memory. They do not stress other computer components such as I/O (disk drives), networking, or graphics. However, it might be possible to configure a system in such a way that one or more of these components distort the energy picture of the benchmarks when comparing different DVS algorithms. To avoid such a situation, we present a methodology that separates out the overall system power dissipation into the part due to benchmark execution and the part due to these other components.

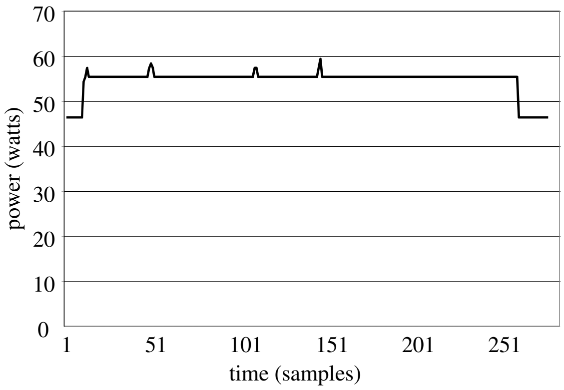

Figure 3. The results of power measurements for benchmark `swim` at the peak performance level.

The methodology divides the overall system power into two parts: the CPU power (P_{C}) and the non-CPU power (P_{NC}). The CPU power is totally dependent on the supply voltage and frequency of the processor ($P_{\text{C}} \propto V^2 f$). In contrast, the non-CPU power is not dependent on any of them. The methodology also defines the *base* system power (P^{base}) as the overall power when the system is not running any benchmark. It can also be the case when running a micro-benchmark that exercises the processor only. Based on these, we model the base system power using the following equation:

$$P^{base}(V, f) \quad = \quad c^{base} \cdot V^2 f + P_{\text{NC}}^{base} \qquad (1)$$

where c^{base} is a constant that represents the switched capacitance. The power dissipation of a program can then be determined by subtracting the non-CPU part of the base system power from the measured power when a program is executed (P^{sys}):

$$P^{prog} \quad = \quad P^{sys} - P_{\text{NC}}^{base}$$

In our experiments, P_{NC}^{base} = 20.93 watts is derived. It

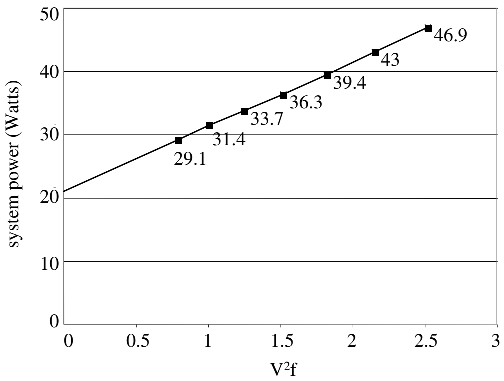

Figure 4. The linear regression on the power of the base system.

benchmark	t (%)	e^{prog} (%)	e^{sys} (%)
swim	101.33	44.35	65.62
tomcatv	102.70	46.30	67.50
hydro2d	103.48	64.32	79.41
su2cor	105.02	65.42	80.40
applu	102.23	84.34	90.99
mgrid	101.94	87.00	95.08
apsi	105.76	88.71	92.43
wave5	100.69	89.48	93.63
turb3d	106.14	90.83	96.40
average	103.26	73.42	84.61

Table 3. The power reduction and performance slowdown for the `SPECfp95` benchmarks using our DVS algorithm.

can be seen in Figure 4 that the base system power at various performance levels for the laptop we measured fits very well in the linear model (1). Note that these 20.93 watts include the power due to AC adapter and power strip, plus the power of some other components. On the other hand, program power only accounts for the power dissipation of the processor and main memory exercised by a program.

4.4. Experimental Results

The experimental results are shown in Table 3, with respect to the original non-DVS'ed programs. The *relative* execution time t and energy consumption e^{prog} are computed according to the following formula:

$$\begin{aligned} t &= T_{\mathrm{DVS}}/T_{\mathrm{ORG}} \\ e^{prog} &= T * (P^{prog}_{\mathrm{DVS}}/P^{prog}_{\mathrm{ORG}}) \end{aligned}$$

where T_{DVS} and P_{DVS} are the total execution time and program power for the DVS'ed benchmark, respectively; T_{ORG} and P_{ORG} are the total execution time and program power for the original program running at the highest performance level. It can be seen that the program energy saving of 9.17% to 55.65% can be achieved with performance degradation of 0.69% to 6.14%. On average, the energy and energy-delay product are saved 26.58% and 24.11%, respectively, at the cost of performance slowdown 3.26%.

We also present the relative overall system energy usage in column e^{sys}. Our DVS algorithm is able to bring down the overall system power by as much as 34.38%, with an average of 15.39%. System energy reduction reflects how much the battery life is prolonged. Assuming the energy of a battery is drawn at a constant rate, column e^{sys} says the battery life is lengthened by 18% (= 1 - 1/0.8461).

4.5. Profitability

A natural follow-up question is how good our algorithm is comparing to the other DVS algorithms. Instead of comparing them directly, we attempt to approximate the lower bound of energy consumption by any DVS algorithm. Our methodology formulates it as a minimization problem: given the set of measured execution time and system power $\{\ (T_i, P_i^{sys})\ \}$ at various performance level i, determine the execution time spent at each performance level δ_i such that

$$\text{(P)} \qquad \begin{aligned} &\text{minimize } \textstyle\sum_i P_i^{prog} \cdot \delta_i \\ &\text{subject to} \\ &\quad \textstyle\sum_i \delta_i \le (1+r) \cdot T_{\mathrm{ORG}} \\ &\quad \textstyle\sum_i \delta_i \cdot (T_i/T_{\mathrm{ORG}}) = T_{\mathrm{ORG}} \end{aligned}$$

The basic idea is to get the optimal time allocation for each performance level so that the total execution time does not violate the timing constraint (the first constraint), the required program workload is performed (the second constraint), and the program energy consumption is minimized.

Figure 5 compares the energy usage derived by our DVS algorithms and the (approximated) "optimal" results for `SPECfp95` benchmarks with $r = 5\%$. It can be seen that in many cases our algorithm has energy reduction very close to the "optimal" one, within 6%. In a few benchmarks, the "optimal" results have higher energy usage than those using our algorithm. The largest error occurs for benchmark `apsi`. The energy usage by the "optimal" algorithm is 6% more than that by our algorithm. We believe the main factor that contributes to the inaccuracy of this methodology is that both T_i and P_i^{sys} are experimentally derived, and therefore introduce measurement errors.

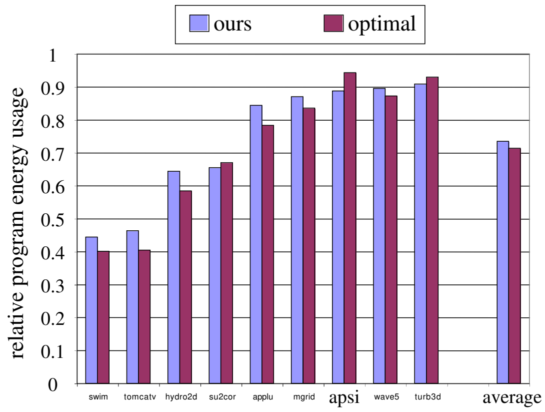

Figure 5. The comparison of the energy consumption derived by our algorithm and the "optimal" results.

program	performance level 1	2	3	4	5	6	7
swim	69%	31%					
tomcatv	83%		17%				
hydro2d			51%	49%			
su2cor				88%	12%		
applu					85%	15%	
mgrid					48%	52%	
apsi			20%				80%
wave5					17%	83%	
turb3d						75%	25%

Table 4. The percentage of overall execution time spent at each performance level for the optimal case.

4.6. Discussion

We have observed, in Table 4, two interesting phenomena about the solutions of the minimization problem (P) for the `SPECfp95` benchmarks. First of all, at most two performance levels are used in the optimal schedule. Secondly, the two levels are not necessarily the immediate neighbors to each other. In a way it seems to suggest that extending our algorithm to allow multiple regions to be slowed down at different speeds may not be more profitable. It also suggests that instead of executing one region at a slower rate and the rest at the default (highest) level, our DVS algorithm can be modified to try using other performance level as the default level during region selection.

We also performed experiments on another system, the Fujitsu LifeBook P2040 laptop. This laptop has a 300-800 MHz Transmeta Crusoe TM5800 microprocessor operated at 1.0-1.3 V. The system provides five performance levels.

spec	Compaq Presario 715US	Fujitsu LifeBook P2040
CPU	AMD mobile Athlon 4	Transmeta Crusoe TM5800
f	600-1200 MHz	300-800 MHz
V	1.15-1.45 V	1.0-1.3 V
levels	7	5
AC adapter	19 V, 3.16 A	16 V, 2.5 A
battery	14.8 V, 3.6 AHr	10.8 V, 1.9 AHr
P_{NC}^{base}	20.93 W	11.54 W
P^{base}	29.1-46.9 W	11.7-12.4 W
P^{sys}	33.56-57.33 W	12.96-15.90 W
P^{prog}	12.63-36.4 W	1.42-4.36 W
$P_{\mathrm{C}}^{base}/P^{base}$	28.08-55.37%	1.37-6.94%
P^{prog}/P^{sys}	37.63-63.49%	10.96-27.42%

Table 5. The DVS support and power breakdown of the two laptops we tested.

Table 5 compares the DVS support on both Compaq Presario 715US laptop and Fujitsu LifeBook P2040 laptop. It also shows the power breakdown for the CPU and the rest of the system. At the peak performance level, 55% of the total base system power can be attributed to the processor for the Presario laptop. In contrast, the processor of the LifeBook laptop only contributes 7% of the total system power. In addition, benchmark execution accounts for 64% and 27% of the total system power for the two laptops, respectively. Since the CPU power of the entire system on the Presario laptop only increases 8% when a benchmark is executed, it seems to suggest that a considerable amount of the CPU work is performed when the system is "idle". We suspect that it has something to do with the ACPI support in the Linux kernel on the laptop. This issue will be looked into in the future.

We applied the methodology for approximating the optimal case to the Fujitsu LifeBook laptop as well. Unfortunately, the results are disappointing. For example, benchmark `swim` can only get 5% of program energy reduction for 5% performance tolerance. In contrast, the Compaq Presario laptop reduces as much as 56%. Figure 6 shows the relative execution time of benchmark `swim` at various performance levels. It can be seen that the program execution time on the LifeBook laptop scales almost linearly with the operating frequency, while the execution time on the Presario laptop does not change much. It indicates the CPU performance is not the bottleneck for the Presario, but it is for the LifeBook. We do not know what causes this since we do not have access to the detailed information about the Crusoe processor.

We also compare the effectiveness of our compiler-

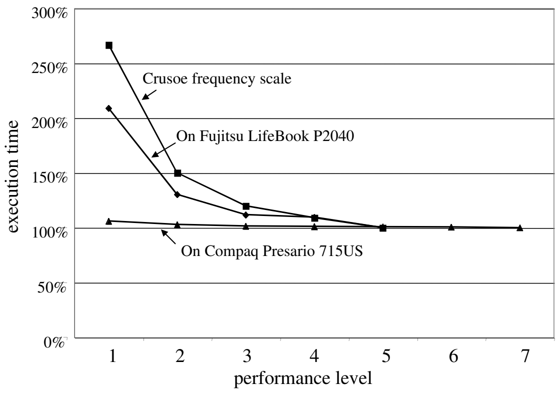

Figure 6. The relative execution time of benchmark `swim` at various performance levels.

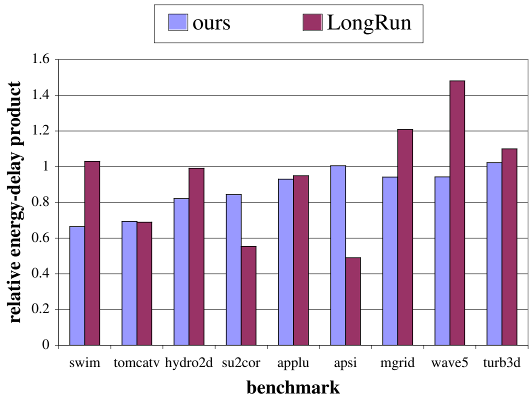

Figure 7. The relative energy-delay product of our compiler approach and the LongRun technology with respect to the Presario notebook without DVS. The energy is the entire system energy.

directed DVS algorithm on the Presario notebook with that of the Transmeta's LongRun technology on the LifeBook notebook. The results are shown in Figure 7. The baseline is the energy-delay product on the Presario notebook without DVS where energy accounts for the entire system energy. It can be seen that in most benchmarks, our compiler-directed approach has a similar energy-delay product value. The high-performance Presario notebook is comparable to the low-power LifeBook notebook in terms of energy efficiency.

Finally, we do not look into Intel's Xscale-based processors, which also support DVS. It is because they do not have the floating-point units while the benchmarks we tested are floating-point intensive.

5. Conclusions and Future Work

In this paper we have discussed a novel compiler algorithm that effectively utilizes dynamic voltage scaling to save energy. The algorithm picks a single region to be executed at a lower performance level without introducing serious performance degradation. A prototype implementation based on the SUIF2 compiler infrastructure was used to evaluate the algorithm on the `SPECfp95` benchmarks. Physical measurements showed significant energy savings in the range of 9.17% to 55.65% can be achieved with 0.69% to 6.14% for the `SPECfp95` benchmarks. On average, the energy and energy-delay product are saved 26.58% and 24.11%, respectively, at the cost of performance slowdown 3.26%. To the best of our knowledge, this work presents one of the first working implementation of DVS algorithms and one of the first to evaluate DVS algorithms through physical measurements.

We are currently setting up a testbed that will allow us to measure the CPU power dissipation directly at the chip level. It requires the laptop to be opened and instrumented. Such measurements will provide further validation of our power measurement strategy. For the Compaq Presario 715US laptop, we received help from Compaq to determine what needs to be done. For the Transmeta system (Fujitsu LifeBook P2040), we have not yet been able to get this information. We are planning to apply our compilation approach to other applications, including image processing, voice recognition, multi-media, and data base applications.

References

[1] K. Agaram, S. Keckler, and D. Burger. A characterization of speech recognition on modern computer systems. In *4th IEEE Workshop on Workload Characterization*, December 2001.

[2] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework. In *Proceedings of Design, Automation and Test in Europe Conference (DATE)*, March 2002.

[3] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of 9th ACM SIGOPS European Workshop*, September 2000.

[4] 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.

[5] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A.Buyuktosunoglu, J-D. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-aware microarchitecture: design and modeling challenges for the next generation microprocessors. *IEEE Micro*, 20(6):26–44, November/December 2000.

[6] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *27th International Symposium on Computer Architecture (ISCA)*, June 2000.

[7] T. Burd and R. Brodersen. Energy efficient CMOS microprocessor design. In *the 28th Hawaii International Conference on System Sciences (HICSS-95)*, January 1995.

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

[9] M. Clark, A. Durg, and K. Lienenbrugger. Characterization of tpc-h queries on amd athlon microprocessors. In *IEEE 4th Annual Workshop on Workload Characterization*, December 2001.

[10] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, July 2001.

[11] 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.

[12] 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.

[13] F. Gruian. Hard real-time scheduling for low-energy using stochastic data and DVS processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'01)*, August 2001.

[14] 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.

[15] S. Gurumurthi, A. Sivasubramaniam, M.J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L.K. John. Using complete machine simulation for software power estimation: The SoftWatt approach. In *Proceedings of the 8th International Symposium on High Performance Computer Architecture*, February 2002.

[16] C.-H. Hsu and U. Kremer. Compiler-directed dynamic voltage scaling based on program regions. Technical Report DCS-TR-461, Department of Computer Science, Rutgers University, November 2001.

[17] C.-H. Hsu and U. Kremer. Dynamic voltage and frequency scaling for scientific applications. In *Proceedings of the 14th annual workshop on Languages and Compilers for Parallel Computing (LCPC 2001)*, August 2001.

[18] C.-H. Hsu and U. Kremer. Single region vs. multiple regions: A comparison of different compiler-directed dynamic voltage scheduling approaches. In *Workshop on Power-Aware Computer Systems (PACS'02)*, 2002.

[19] C.-H. Hsu, U. Kremer, and M. Hsiao. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'01)*, August 2001.

[20] M. Huang, J. Renau, and J. Torrellas. Profile-based energy reduction in high-performance processors. In *4th Workshop on Feedback-Directed and Dynamic Optimization*, December 2001.

[21] C. Hughes, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proceedings of the 34th International Symposium on Microarchitecture (MICRO-34)*, December 2001.

[22] 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.

[23] J. Lorch and A. Smith. Improving dynamic voltage algorithms with PACE. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2001)*, June 2001.

[24] A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy. In

Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'01), August 2001.

[25] T. Martin and D. Siewiorek. Nonideal battery and main memory effects on cpu speed setting for low power. *IEEE Transactions on Very Large Scale Integration (VLSI) System*, 9(1):29–34, February 2001.

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

[27] 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.

[28] T. Mudge. Power: A first class design constraint for future architectures. *IEEE Computer*, 34(4):52–58, April 2001.

[29] 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.

[30] M. Pettersson. perfctr-2.3.7.

[31] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th Symposium on Operating Systems Principles*, October 2001.

[32] D. Ponomarev, G. Kucuk, and K. Ghose. AccuPower: An accurate power estimation tool for superscalar microprocessors. In *Proceedings of Design, Automation and Test in Europe Conference*, March 2002.

[33] J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'01)*, August 2001.

[34] H. Saputra, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, J. Hu, C.-H. Hsu, and U. Kremer. Energy-conscious compilation based on voltage scaling. In *ACM SIGPLAN Joint Conference on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems (LCTES/SCOPES'02)*, June 2002.

[35] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *8th International Symposium on High-Performance Computer Architecture*, February 2002.

[36] D. Shin and J. Kim. A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED'01)*, August 2001.

[37] P. Stanley-Marbell, M. Hsiao, and U. Kremer. A hardware architecture for dynamic performance and energy adaptation. In *Workshop on Power-Aware Computer Systems (PACS'02)*, 2002.

[38] SUIF. Stanford University Intermediate Format.

[39] 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.

[40] 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.

[41] W. Ye, N. Vijaykrishna, M. Kandemir, and M.J. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Design Automation Conference (DAC)*, June 2000.

[42] L. Zhang, J. Carter, W. Hsieh, and S. McKee. Memory system support for image processing. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, October 1999.