User- and Process-Driven Dynamic Voltage and Frequency Scaling

Bin Lin Arindam Mallik Peter Dinda Gokhan Memik Robert Dick
{b-lin,arindam,pdinda,g-memik,dickrp}@northwestern.edu
Department of EECS, Northwestern University

Abstract

We describe and evaluate two new, independently-applicable power reduction techniques for power management on processors that support dynamic voltage and frequency scaling (DVFS): user-driven frequency scaling (UDFS) and process-driven voltage scaling (PDVS). In PDVS, a CPU-customized profile is derived offline that encodes the minimum voltage needed to achieve stability at each combination of CPU frequency and temperature. On a typical processor, PDVS reduces the voltage below the worst-case minimum operating voltages given in datasheets. UDFS, on the other hand, dynamically adapts CPU frequency to the individual user and the workload through direct user feedback. Our UDFS algorithms dramatically reduce typical operating frequencies and voltages while maintaining performance at a satisfactory level for each user. We evaluate our techniques independently and together through user studies conducted on a Pentium M laptop running Windows applications. We measure the overall system power and temperature reduction achieved by our methods. Combining PDVS and the best UDFS scheme reduces measured system power by 49.9% (27.8% PDVS, 22.1% UDFS), averaged across all our users and applications, compared to Windows XP DVFS. The average temperature of the CPU is decreased by $13.2°C$*. User trace-driven simulation to evaluate the CPU only indicates average CPU dynamic power savings of 57.3% (32.4% PDVS, 24.9% UDFS), with a maximum reduction of 83.4%. In a multitasking environment, the same UDFS+PDVS technique reduces the CPU dynamic power by 75.7% on average.*

1 Introduction

Dynamic Voltage and Frequency Scaling (DVFS) is one of the most commonly used power reduction techniques in high-performance processors and is an important OS power management tool. DVFS is generally implemented in the kernel and it varies the frequency and voltage of a microprocessor in real-time according to processing needs. Although there are different versions of DVFS, at its core DVFS adapts power consumption and performance to the current workload of the CPU. Specifically, existing DVFS techniques in high-performance processors select an operating point (CPU frequency and voltage) based on the utilization of the processor. While this approach can integrate information available to the OS kernel, such control is pessimistic.

Existing DVFS techniques are pessimistic about the user. They assume that CPU utilization or the OS events prompting it are sufficient proxies. A high CPU utilization simply leads to a high frequency and high voltage, regardless of the user's satisfaction or expectation of performance.

Existing DVFS techniques are often pessimistic about the CPU. They assume worst-case manufacturing process variation and operating temperature by basing their policies on loose worst-case bounds given by the processor manufacturer. A voltage level for each frequency is set such that even the slowest shipped processor of a given generation will be stable at the highest specified temperature.

In response to these observations, on which we elaborate in Sections 2.1 and 3.1, we have developed two new power management techniques that can be readily employed independently or together. In particular, we introduce the following techniques.

User-Driven Frequency Scaling (UDFS) uses direct user feedback to drive an online control algorithm that determines the processor frequency (Section 2.2). Processor frequency has strong effects on power consumption and temperature, both directly and also indirectly through the need for higher voltages at higher frequencies. The choice of frequency is directly visible to the user as it determines observed performance. There is considerable variation among users with respect to the satisfactory performance level for a given workload mix. UDFS exploits this variation to customize frequency control policies dynamically to the *individual* user. Unlike previous work (Section 5), we employ direct feedback from the user during ordinary use of the machine.

Process-Driven Voltage Scaling (PDVS) creates a custom mapping from frequency and temperature to the minimum voltage needed for CPU stability (Section 3.2), taking advantage of process variation. This mapping is then used online to choose the operating voltage by taking into account the current operating temperature and frequency. Researchers have shown that process variation causes IC speed to vary up to 30% [2]. Hence, using a single supply voltage setting does not exploit the variation in timing present among processors. Although some processors customize voltage–frequency mappings based on process variation, none adjust voltage as a function of temperature. PDVS does both. We take advantage of the variation among ICs via a customization process that determines the slack of the *individual*

This work is in part supported by DOE Awards DE-FG02-05ER25691 and DE-AC05-00OR22725 (via ORNL), NSF Awards CNS-0720691, CNS-0721978, CNS-0715612, CNS-0551639, CNS-0347941, CCF-0541337, CCF-0444405, CCF-0747201, IIS-0536994, IIS-0613568, ANI-0093221, ANI-0301108, and EIA-0224449, by SRC award 2007-HJ-1593, by Wissner-Slivka Chair funds, and by gifts from Symantec, Dell, and VMware.

processor, as well as its dependence on operating temperature. This offline measurement is used online to dynamically set voltage based on frequency and temperature.

We evaluate our techniques independently and together through user studies conducted on a Pentium M laptop running Windows applications. Our studies, described in detail in Section 4, include both single task and multitasking scenarios. We measure the overall system power and temperature reduction achieved by our methods. Combining PDVS and the best UDFS scheme reduces measured system power by 49.9% (27.8% PDVS, 22.1% UDFS), averaged across all our users and applications, compared to the Windows XP DVFS scheme. The average temperature of the CPU is decreased by $13.2°$C on average. Using user trace-driven simulation to evaluate the CPU in isolation, we find average CPU dynamic power savings of 57.3% (32.4% PDVS, 24.9% UDFS), with a maximum reduction of 83.4%. In a multitasking environment, the same UDFS+PDVS technique reduces the CPU dynamic power by 75.7% on average.

1.1 Experimental setup

Our experiments were done using an IBM Thinkpad T43p with a 2.13 GHz Pentium M-770 CPU and 1 GB memory running Microsoft Windows XP Professional SP2. Although eight different frequency levels can be set on the Pentium M-770 processor, only six can be used due to limitations in the SpeedStep technology.

In all of our studies, we make use of three application tasks, some of which are CPU intensive and some of which frequently block while waiting for user input:

- Creating a presentation using Microsoft PowerPoint 2003 while listening to background music using Windows Media Player 10. The user duplicates a presentation consisting of complex diagrams involving drawing and labeling, starting from a hardcopy of a sample presentation.
- Watching a 3D Shockwave animation using the Microsoft Internet Explorer web browser. The user watches the animation and is encouraged to press the number keys to change the camera's viewpoint. The animation was stored locally. Shockwave options were configured so that rendering was done entirely in software on the CPU.
- Playing the FIFA 2005 Soccer game. FIFA 2005 is a popular and widely-used First Person Shooter game. There were no constraints on user gameplay.

In the following sections, we describe the exact durations of these tasks for each user study and additional tasks the user was asked to undertake. In general, our user studies are double-blind, randomized, and intervention-based. The default Windows DVFS scheme is used as the control. We developed a user pool by advertising our studies within Northwestern University. We selected a random group of users from among those who responded to our advertisement. While many of the selected users were CS, CE, or EE graduate students, our users included staff members and undergraduates from the humanities. Each user was paid $15 for participating. Our studies ranged from number of users $n = 8$ to $n = 20$, as described in the material below.

2 User-Driven Frequency Scaling (UDFS)

Current DVFS techniques are pessimistic about the user, which leads them to often use higher frequencies than necessary for satisfactory performance. In this section, we elaborate on this pessimism and then explain our response to it: user-driven frequency scaling (UDFS). Evaluations of UDFS algorithms are given in Section 4.

2.1 Pessimism about the user

Current software that drives DVFS is pessimistic about the individual user's reaction to the slowdown that may occur when CPU frequency is reduced. Typically, the frequency is tightly tied to CPU usage. A burst of computation due to, for example, a mouse or keyboard event brings utilization quickly up to 100% and drives frequency, voltage, temperature, and power consumption up along with it. CPU-intensive applications also cause an almost instant increase in operating frequency and voltage.

In both cases, the CPU utilization (or OS events that drive it) is functioning as a proxy for user comfort. Is it a good proxy? To find out, we conducted a small ($n = 8$) randomized user study, comparing four processor frequency strategies including dynamic, static low frequency (1.06 GHz), static medium frequency (1.33 GHz), as well as static high frequency (1.86 GHz). The dynamic strategy is the default DVFS policy used in Windows XP Professional. Note that the processor maximum frequency is 2.13 GHz. We allowed the users to acclimate to the full speed performance of the machine and its applications for 4 minutes and then carry out the tasks described in Section 1.1, with the following durations:

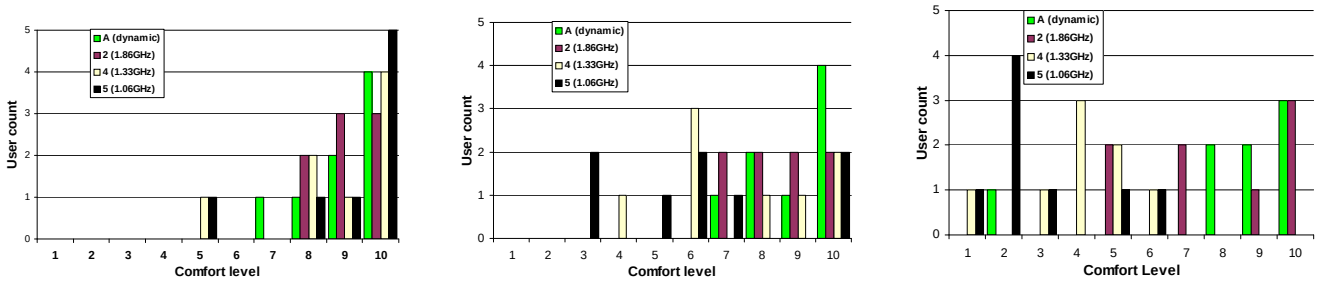
- PowerPoint (4 minutes in total, 1 minute per strategy)
- Shockwave (80 seconds in total, 20 seconds per strategy)
- FIFA (4 minutes in total, 1 minute per strategy)

Users verbally ranked their comfort levels after each task / strategy pair on a scale of 1 (discomfort) to 10 (very comfortable). Note that for each application and user, strategies were tested in random order.

Figure 1 illustrates the results of the study in the form of overlapped histograms of the participants' reported comfort level for each of four strategies. Consider Figure 1(a), which shows results for the PowerPoint task. The horizontal axis displays the range of comfort levels allowed in the study and the vertical axis displays the count of the number of times that level was reported. The other graphs are similar.

User comfort with any given strategy is highly dependent on the application. For PowerPoint, the strategies result in indistinguishable satisfactions levels. For this task, we could simply set the frequency statically to a very low value and never change it, presumably saving power. For animation, a higher static level is preferred but the medium and high frequencies are statistically indistinguishable from the dynamic strategy despite not using as high a frequency. For the game, the high static setting is needed to match the satisfaction level of the dynamic strategy. However, that setting does not use the highest possible frequency, which was used by the dynamic strategy throughout the experiment.

Comfort with a given strategy is strongly user-dependent, i.e.,



(a) Microsoft Powerpoint (b) 3D Shockwave Animation (c) FIFA Game.

Figure 1. User pessimism.

it is important to note that for any particular strategy, there is considerable spread in the reported comfort levels. In addition to the power-specific results just described, we note that Gupta et al. [14] and Lin et al. [20] have also demonstrated a high variation in user tolerance for performance in other contexts. Our dynamic policy automatically adapts to different users and applications. Hence, it can reduce power consumption while still achieving high user satisfaction.

2.2 Technique

To implement user-driven frequency scaling, we have built a system that consists of client software that runs as a Windows toolbar task as well as software that implements CPU frequency and temperature monitoring. In the client, the user can express discomfort at any time by pressing the F11 key (the use of other keys or controls can be configured). These events drive the UDFS algorithm. The algorithm in turn uses the Windows API to control CPU frequency. We monitor the CPU frequency using Windows Performance Counter and Log [26] and temperature using CPUCool [39].

It is important to note that a simple strategy that selects a static frequency for an application (and/or for a user) is inadequate for three reasons. First, each user will be satisfied with a different level of performance for each application. Finding these levels statically would be extremely time consuming. Second, typical users multitask. Capturing the effects of multiple applications would necessitate examining the power set of the application set for each individual user, resulting in a combinatoric explosion in the offline work to be done. Finally, even when a user is working with a single application, the behavior of the application and the expected performance varies over time. Applications go through phases, each with potentially different computational requirements. In addition, the user's expected performance is also likely to change over time as the user's priorities shift. For these reasons, a frequency scaling algorithm should dynamically adjust to the individual user's needs.

Responding to these observations, we designed algorithms that employ user experience feedback indicated via button presses.

2.2.1 UDFS1 Algorithm

UDFS1 borrows from the adaptive algorithm in our user-driven CPU scheduling work [21] and can be viewed as extensions/variants of the TCP congestion control control algorithm. The TCP congestion control algorithm [33, 38, 3, 10] is designed to adapt the send rate dynamically to the available bandwidth in the path. A congestion event corresponds to a user button press, send rate corresponds (inversely) to CPU frequency, and TCP acknowledgments correspond to the passage of time.

UDFS1 has two state variables: r, the current control value (CPU frequency, the smaller the value, the higher the frequency.) and r_t (the current threshold, integer value). Adaptation is controlled by three constant parameters: ρ, the rate of increase, $\alpha = f(\rho)$, the slow start speed, and $\beta = g(\rho)$, the additive increase speed. Like TCP, UDFS1 operates in three modes, as described below.

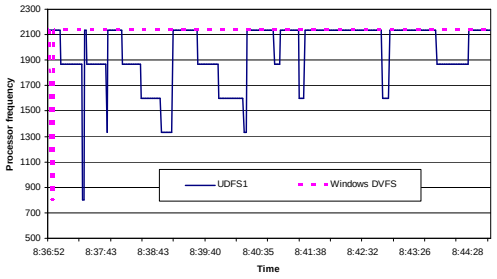
- Slow Start (Exponential Increase): If $r < r_t$, we increase r exponentially fast with time (e.g., $r \propto 2^{\alpha t}$). Note that frequency settings for most processors are quantized and thus the actual frequency changes abruptly upon crossing quantization levels.
- User event avoidance (Additive Increase): If no user feedback is received and $r \geq r_t$, r increases linearly with time, $r \propto \beta t$.
- User event (Multiplicative Decrease): When the user expresses discomfort at level r we immediately set $r_t = r - 1$ and set r to the initial (highest) frequency.

This behavior is virtually identical to that of TCP Reno, except for the more aggressive setting of the threshold.

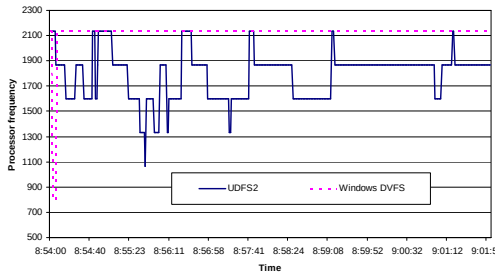
Unlike TCP Reno, we also control ρ, the key parameter that controls the rate of exponential and linear increase from button press to button press. In particular, for every user event, we update ρ as follows:

$$\rho_{i+1} = \rho_i \left(1 - \gamma \times \frac{T_i - T_{AVI}}{T_{AVI}}\right)$$

where T_i is the latest inter-arrival time between user events. T_{AVI} is the target mean inter-arrival time between user events, as currently preset by us. γ controls the sensitivity to the feedback.



(a) UDFS1 scheme



(b) UDFS2 scheme

Figure 2. The frequency for UDFS schemes during FIFA game for a representative user.

We set our constant parameters ($T_{AVI} = 120, \alpha = 1, \beta = 1, \gamma = 0.8$) based on the experience of two of the authors using the system. These parameters were subsequently used when conducting a user study to evaluate the system (Section 4). Ideally, we would empirically evaluate the sensitivity of UDFS1 (and UDFS2) performance to these parameters. However, it is important to note that any such study would require having real users in the loop, and thus would be quite slow. Testing five values of each parameter on 20 users would require 312 days (based on 8 users/ day and 45 minutes/user). For this reason, we decided to choose the parameters based on qualitative evaluation by the authors and then "close the loop" by evaluating the whole system with the choices.

Figure 2(a) illustrates the execution of the UDFS1 and Windows DVFS algorithms for a typical user during the FIFA game task. Note that Windows DVFS causes the system to run at the highest frequency during the whole execution period except the first few seconds. On the other hand, the UDFS1 scheme causes the processor frequency to increase only when the user expresses discomfort (by pressing F11). Otherwise, it slowly decreases.

2.2.2 UDFS2 Algorithm

UDFS2 tries to find the lowest frequency at which the user feels comfortable and then stabilize there. For each frequency level possible in the processor, we assign an interval t_i, the time for the algorithm to stay at that level. If no user feedback is received during the interval, the algorithm reduces the frequency from r_i to r_{i+1}, i.e., it reduces the frequency by one level. The default interval is 10 seconds for all levels. If the user is irritated at control level r_i, we update all of our intervals and the current frequency level as follows:

$$\begin{aligned} t_{i-1} &= \alpha t_{i-1} \\ t_k &= \beta t_k, \forall k : k \neq i-1 \\ i &= \min(i-1, 0) \end{aligned}$$

Here $\alpha > 1$ is the rate of interval increase and $\beta < 1$ is rate of interval decrease. In our study, $\alpha = 2.5$ and $\beta = 0.8$. This strategy is motivated by the conjecture that the user was comfortable with the previous level and the algorithm should spend more time at that level. Again, because users would have to be in the inner loop of any sensitivity study, we have chosen the parameters qualitatively and evaluated the whole system using that choice, as described in Section 4.

Figure 2(b) illustrates the execution of the algorithm for a representative user in the FIFA game task. Note that UDFS2 settles to a frequency of approximately 1.86 GHz, after which little interaction is needed.

3 Process-Driven Voltage Scaling (PDVS)

Current DVFS techniques are pessimistic about the processor, which leads them to often use higher voltages than necessary for stable operation, especially when they have low temperatures. We elaborate on this pessimism and then explain our response to it, process-driven voltage scaling (PDVS). PDVS is evaluated in Section 4.

3.1 Pessimism about the CPU

The *minimum stable voltage* of a CPU is the supply voltage that guarantees correct execution for given process variation and environmental conditions. It is mainly determined by the critical path delay of a circuit. This delay consists of two components: transistor gate delay and wire delay. Gate delay is inversely related to the operating voltages used in the critical paths of the circuit. Furthermore, temperature affects the delay. Charge carrier mobility decreases with increasing temperature. Although partially offset by decreased threshold voltages, in current technologies this reduction in carrier mobility causes circuits to slow down with increasing temperature. Wire delay is also temperature-dependent and increases under higher current/temperature conditions. The maximum operating frequency (F_{max}) varies in direct proportion to the sustained voltage level in the critical timing paths, and inversely with temperature-dependent RC delay [37].

In addition to the operating conditions, which dynamically change, process variation has an important impact on the minimum voltage sufficient for stable operation. Even in identical environments, a variation in timing slack is observed among the manufactured processors of the same family. As a result, each processor reacts differently to changes. For example, although two processors can run safely at 2.8 GHz at the default supply voltage, it is conceivable that these minimum supply voltages will differ. Customizing voltage choices for individual processors adapts to, and exploits, these variations. Despite these known effects of process variation and temperature on minimum stable

Operating Freq. (MHz)	Nominal Voltage (v)	Stable V_{dd} (V) at temp ranges (°C)			
		52–57	62–67	72–77	82–87
800	0.988	0.736	0.736	0.736	0.736
1,060	1.068	0.780	0.780	0.780	0.780
1,200	1.100	0.796	0.796	0.796	0.796
1,330	1.132	0.844	0.844	0.860	0.876
1,460	1.180	0.876	0.892	0.908	0.924
1,600	1.260	0.908	0.924	0.924	0.924
1,860	1.324	1.004	1.004	1.020	1.020
2,130	1.404	1.084	1.100	1.116	1.116

Figure 3. Minimum stable V_{dd} for different operating frequencies and temperatures.

voltage, DVFS ignores them: for a given frequency, traditional DVFS schemes use a single voltage level for all the processors within a family at all times.

The dynamic power consumption of a processor is directly related to frequency and supply voltage and can be expressed using the formula $P \propto V^2CF$, which states that power is proportional to the product of voltage squared, capacitance, and frequency. In addition to its direct impact on the power consumption, reliable operation at increased frequency demands increased supply voltage, thereby having an indirect impact on power consumption. Generally, if the frequency is reduced, a lower voltage is safe.

As processors, memories, and application-specific integrated circuits (ASICs) are pushed to higher performance levels and higher transistor densities, processor thermal management is quickly becoming a first-order design concern. The maximum operating temperature of an Intel Pentium Mobile processor has been specified as $100°$C [16, 17]. As a general rule of thumb, the operating temperature of a processor can vary from $50°$C to $90°$C during normal operation. Thus, there is a large difference between normal and worst-case temperatures.

We performed an experiment that reveals the relationship between operating frequency and minimum stable voltage of the processor at different temperature ranges. We used Notebook Hardware Control (NHC) [18] to set a particular V_{dd} value for each operating frequency supported by the processor. When a new voltage value is set, NHC runs an extensive CPU stability check. Upon failure, the system stops responding and computer needs to be rebooted. We execute a program that causes high CPU utilization and raises the temperature of the processor. When the temperature reaches a desired range, we perform the CPU stability check for a particular frequency at a user-defined voltage value.

Figure 3 shows the results of this study for the machine described in Section 1.1. For reference, we also show the nominal core voltage given in the datasheet [17]. Note that the nominal voltage is the voltage used by all the DVFS schemes by default. The results reveal that, even at the highest operating temperature, the minimum stable voltage is far smaller than the nominal voltage. The results also show that at lower operating frequencies, the effect of temperature on minimum stable voltage is not pronounced. However, temperature change has a significant impact on minimum stable voltage at higher frequencies. In particular, at 1.46 GHz, the core voltage value can vary by 5.6% for a temperature change of $30°$C. This would reduce dynamic power consumption by 11.4%.

As the results shown in Figure 3 illustrate, there is an opportunity for power reduction if we exploit the relationship between frequency, temperature, and the minimum stable voltage. The nominal supply voltage specified in the processor datasheet has a large safety margin over the minimum stable voltages. This is not surprising: worst-case assumptions were unnecessarily made at a number of design stages, e.g., about temperature. Conventional DVFS schemes are therefore pessimistic about particular *individual* CPUs, often choosing higher voltages than are needed to operate safely. They also neglect the effect of temperature, losing the opportunity to save further power.

3.2 Technique

We have developed a methodology for exploiting the process variation described in Section 3.1 that can be used to make *any* voltage and frequency scaling algorithm adapt to individual CPUs and their temperature, thereby permitting a reduction in power consumption.

Our technique uses offline profiling of the processor to find the minimum stable voltages for different combinations of temperature and frequency. Online temperature and frequency monitoring is then used to set the voltage according to the profile. The offline profiling is virtually identical to that of Section 3.1 and needs to be done only once. Currently, it is implemented as a watchdog timer-driven script on a modified Knoppix Live CD that writes the profile to a USB flash drive. To apply our scheme, the temperature is read from the online sensors that exist in the processor. The frequency, on the other hand, is determined by the dynamic frequency scaling algorithm in use. By setting the voltage based on the processor temperature, frequency, and profile, we adapt to the operating environment. While the frequency can be readily determined (or controlled), temperature changes dynamically. Hence, the algorithm has built-in filtering and headroom to account for this fact. Our algorithm behaves conservatively and sets the voltage such that even if there is a change of $5°$C in temperature before the next reading (one Hertz rate), the processor will continue working correctly.

A reader may at this point be concerned that our reduction of the timing safety margin from datasheet norms might increase the frequency of timing errors. However, PDVS carefully determines the voltage required for reliable operation for each processor; that is, it finds the *individual* processor's safety margin. Moreover, it decreases the operating temperature of the processor, which reduces the rates of lifetime failure processes. If characteristics of processors change as a result of wear, PDVS can adapt by infrequently, e.g., every six months, repeating the offline characterization process. To determine processor reliability when using reduced operating voltage, we ran demanding programs test the stability of different processor components, e.g., the ALU, at lower voltages. We have set the processor to work at modified supply voltages as indicated in Figure 3. The system remained stable for approximately two months, at which point we terminated testing. Although observing the stable operation of one machine does not prove reliability, it is strong evidence.

4 Evaluation

We now evaluate UDFS and PDVS in isolation and together. We compare against the native Windows XP DVFS scheme, displaying reductions in power and temperature.

Our evaluations are based on user studies, as described in Section 1.1 and elaborated upon here. For studies not involving UDFS, we trace the user's activity on the system as he uses the applications and monitor the selections DVFS makes in response. For studies involving UDFS, the UDFS algorithm is used online to control the clock frequency in response to user button presses. We begin by describing a user study of UDFS that provides both independent results and traces for later use. Next, we consider PDVS as applied to the Windows DVFS algorithm. We then consider UDFS with and without PDVS, comparing to Windows DVFS. Here, we examine both dynamic CPU power (using simulation driven from the user traces) and system power measurement (again for a system driven from the user traces). In measurement, we consider not only power consumption, but also CPU temperature. Finally, we discuss a range of other aspects of the evaluation of the system.

The following claims are supported by our results:

- UDFS effectively employs user feedback to customize processor frequency to the individual user. This typically leads to significant power savings compared to existing dynamic frequency schemes that rely only on CPU utilization as feedback. The amount of feedback from the user is infrequent, and declines quickly over time as an application or set of applications is used.
- PDVS can be easily incorporated into any existing DVFS scheme, such as the default Windows scheme, and leads to dramatic reductions in power use by lowering voltage levels while maintaining processor stability.
- In most of the cases, the effects of PDVS and UDFS are synergistic: the power reduction of UDFS+PDVS is more than the sum of its parts.
- Multitasking increases the effectiveness of UDFS+PDVS.
- Together and separately, PDVS and UDFS typically decrease CPU temperature, often by large amounts, increasing both reliability and longevity. In addition, the effects of PDVS and UDFS on temperature are synergistic.

There are limitations to using summary statistics to compare results with $n = 20$. Although we have done statistical tests[1] to support our comparisons, we also try to provide *unsummarized* data to the largest extent possible given space limitations so that the readers can use their own judgment. As can be seen from the paper, the differences are very large and thus unlikely to be due to chance.

4.1 UDFS

To evaluate the UDFS schemes, we ran a study with 20 users. Experiments were conducted as described in Section 1.1. Each user spent 45 minutes to

[1] Although we have also done unpaired t-tests, we generally base our comparisons on Chebyshev bounds. Chebyshev bounds allow us to avoid assumptions about the distribution of the data and are looser bounds, hence making our results stronger.

Application	Power Reduction (%) over Max Freq.	
	DVFS	DVFS+PDVS
PowerPoint + Music	83.08	90.67
3D Shockwave Animation	3.19	40.67
FIFA Game	1.69	39.69

Figure 4. Power reduction for Windows DVFS and DVFS+PDVS

1. Fill out a questionnaire stating level of experience in the use of PCs, Windows, Microsoft PowerPoint, music, 3D animation video, and FIFA 2005 from among the following set: "Power User", "Typical User", or "Beginner" (2 minutes);
2. Read a one page handout (2 minutes);
3. Acclimatize to the performance of our machine by using the above applications (5 minutes);
4. Perform the following tasks for UDFS1: Microsoft PowerPoint plus music (4 minutes); 3D Shockwave animation (4 minutes); FIFA game (8 minutes); and
5. Perform the same set of tasks for UDFS2.

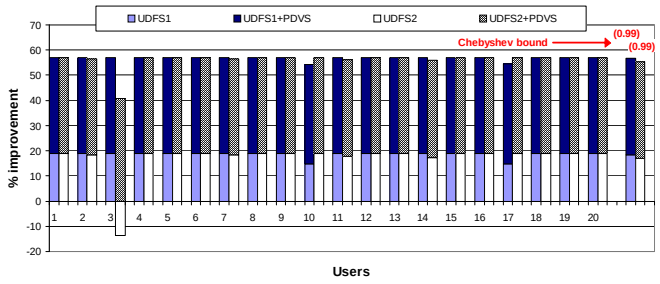
Each user was instructed to press the F11 key upon discomfort with application performance. We recorded each such event as well as the CPU frequency over time.

As one might expect, the average frequency at which users are comfortable is higher for the Shockwave animation and the FIFA game. However, there is a large variation in acceptable frequency among the users for the animation and game. Generally, UDFS2 achieves a lower average frequency than UDFS1. For both algorithms it is very rare to see the processor run at the maximum CPU frequency for these applications. Even the most sophisticated users were comfortable with running the tasks with lower frequencies than those selected by the dynamic Windows DVFS scheme. Sections 4.3 and 4.4 give detailed, per-user results for UDFS (and UDFS+PDVS).

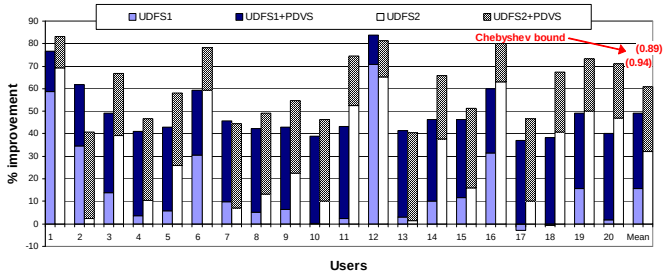
4.2 PDVS

Using the experimental setup described in Section 1.1, we evaluate the effects of PDVS on the default Windows XP DVFS scheme. In particular, we run the DVFS scheme, recording frequency, then determine the power saving possible by setting voltages according to PDVS instead of using the nominal voltages of DVFS.

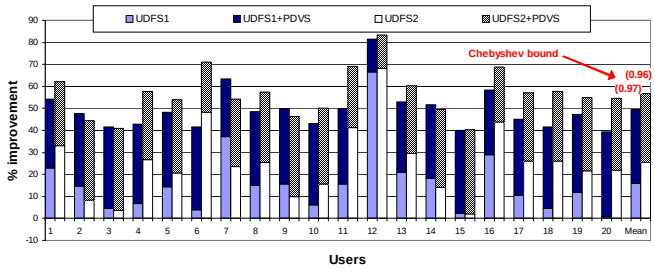
Figure 4 illustrates the average results, comparing stock Windows DVFS and our DVFS+PDVS scheme. The baseline case in this experiment is running the system with the highest possible CPU frequency and its corresponding nominal voltage. The maximum power savings due to dynamic frequency scaling with nominal voltages are observed for PowerPoint. For this application, the system ran at the lowest clock frequency most of the time, resulting in a reduction of 83.1% for the native DVFS scheme. DVFS+PDVS reduces the power consumption by 90.7%. For PowerPoint, adding PDVS to DVFS only reduces power slightly.



(a) Microsoft PowerPoint



(b) 3D Shockwave animation



(c) FIFA game

Figure 5. Comparison of UDFS algorithms, UDFS+PDVS, and Windows XP DVFS (CPU Dynamic Power). Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown.

For the Shockwave animation and the FIFA game, the power reductions due to dynamic frequency scaling are negligible because the Windows DVFS scheme runs the processor at the highest frequency most of the time. DVFS+PDVS, however, improves the energy consumption of the system by approximately 40%, compared to the baseline. These results clearly demonstrate the benefits of process-driven voltage scaling.

4.3 UDFS+PDVS (CPU dynamic power, trace-driven simulation)

To integrate UDFS and PDVS, we used the system described in Section 2.2, recording frequency over time. We then combine this frequency information with the offline profile and techniques described in Sections 3.1 and 3.2 to derive CPU power savings for UDFS with nominal voltages, UDFS+PDVS, and the default Windows XP DVFS strategy. We calculate the power consumption of the processor. We have also measured online the power consumption of the overall system, as described in Section 4.4.

We conducted a user study ($n = 20$) with exactly the same structure presented in Section 2.2, except that Windows XP DVFS was also considered. Figure 5 presents both individual user results and average results for UDFS1, UDFS1+PDVS, UDFS2, and UDFS2+PDVS. In each case, power savings over the default Windows DVFS approach are reported. To interpret the figure, first choose an application. Next, note the last two bars on the corresponding graph. These indicate the average performance of UDFS1 and UDFS2, meaning the percentage reduction in power use compared to Windows DVFS. Each bar is broken into two components: the performance of the UDFS algorithm without PDVS is the lower component and the improvement in performance of the algorithm combined with PDVS is the upper component. The remaining bars on the graph have identical semantics, but represent user-specific information.

For PowerPoint, UDFS1+PDVS and UDFS2+PDVS reduce power consumption by an average of 56%. The standalone UDFS algorithms reduce it by an average of 17–19%. User 3 with UDFS2 is anomalous. This user pressed the feedback button several times and as a result spent most of the time at high frequencies.

For the Shockwave animation, we see much more mixed responses from the users, although on average we reduce power by 55.1%. On average, UDFS1 and UDFS2 independently reduce the power consumption by 15.6% and 32.2%, respectively. UDFS2 performs better for this application because the users can be satisfied by ramping up to a higher frequency rather than the maximum frequency supported by the processor. Note that UDFS1 immediately moves to the maximum frequency on a button press. User 17 with UDFS1 is anomalous. This user wanted the system to perform better than the hardware permitted and thus pressed the button virtually continuously even when it was running at the highest frequency. Adding PDVS lowers average power consumption even more significantly. On average, the power is reduced by 49.2% (UDFS1+PDVS) and 61.0% (UDFS2+PDVS) in the combined scheme.

There is also considerable variation among users for the FIFA game. Using conventional DVFS, the system always runs at the highest frequency. The UDFS schemes try to throttle down the frequency over the time. They therefore reduce the power consumption even in the worst case (0.9% and 2.1% for UDFS1 and UDFS2, respectively) while achieving better improvement, on average (16.1% and 25.5%, respectively). Adding PDVS improves the average power savings to 49.5% and 56.7% for UDFS1 and UDFS2, respectively.

For the Shockwave animation and the FIFA game, we see a large variation among users, but in all cases the combination of PDVS and UDFS leads to power savings over Windows DVFS. On average, in the best case, the power consumption can be reduced by 57.3% over existing DVFS schemes for all three applications. This improvement is achieved by combining the UDFS2 (24.9%) and PDVS (32.4%) schemes.

UDFS and PDVS are synergistic. The UDFS algorithms let us dramatically decrease the average frequency, and PDVS's benefits increase as the frequency is lowered. At higher frequencies,

the relative change from the nominal voltage to the minimum stable voltage is lower than that at lower frequencies. In other words, the power gain from shifting to the minimum stable voltage is higher at the lower frequencies. However, at higher frequencies, PDVS also gains from the variation in minimum stable voltage based on temperature as shown in Figure 3. These two different advantages of the PDVS result in power improvements at at a wide range of frequencies.

UDFS+PDVS mean results have statistical significance even with weak bounds. Figure 5 shows mean improvements across our 20 users. Normality assumptions hold neither for the distribution of individual user improvements nor for the error distribution of the mean. Instead, to discard the null hypothesis, that our mean improvements for UDFS+PDVS are not different from zero, we have computed the p value for discarding the null hypothesis using Chebyshev bounds, which are looser but rely on no such assumptions. As can be seen from the figure, $1 - p$ is quite high, indicating that it is extremely unlikely that our mean improvements are due to chance. We use Chebyshev bounds similarly for other results.

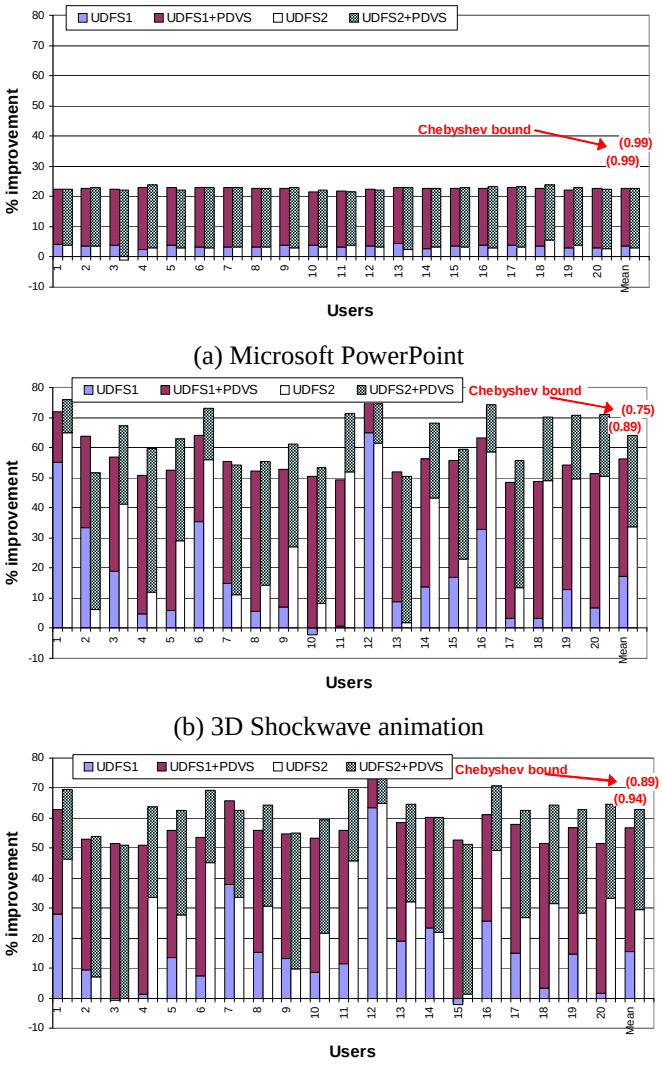
User self-reported level of experience correlates with power improvement. For example, for FIFA, experienced users expect faster response from the system causing the system to run at higher frequencies, resulting in smaller power improvements. Our interpretation is that familiarity increases both expectations and the rate of user feedback to the control agent, making annoyance with reduced performance more probable and thus leading to higher frequencies when using the UDFS algorithms.

4.4 UDFS+PDVS (System power and temperature measurement)

To further measure the impact of our techniques, we replay the traces from the user study of the previous section on our laptop. The laptop is connected to a National Instruments 6034E data acquisition board attached to the PCI bus of a host workstation running Linux, which permits us to measure the power consumption of the entire laptop. The sampling rate is 10 Hz. During the measurements, we have turned off the display of the laptop to make our readings more comparable to the CPU power consumption results of the previous section. Ideally, we would have preferred to measure CPU power directly for one-to-one comparison with results of the previous section, but we do not have the surface mount rework equipment needed to do so.

Power Figure 6 presents results for UDFS1, UDFS1+PDVS, UDFS2, and UDFS2+PDVS, showing the power savings over the default Windows DVFS approach. The Chebyshev bounds indicate that the mean improvements are extremely unlikely to have occured by chance.

For PowerPoint, UDFS1+PDVS and UDFS2+PDVS reduce power consumption by averages of 22.6% and 22.7%, respectively. For the Shockwave animation, although we see much more variation, UDFS1 and UDFS2 reduce the power consumption by 17.2% and 33.6%, respectively. Using UDFS together with PDVS lowers average power consumption by 38.8% and 30.4% with UDFS1 and UDFS2, respectively. The FIFA game



(a) Microsoft PowerPoint

(b) 3D Shockwave animation

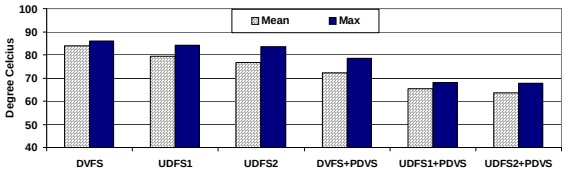
(c) FIFA Game

Figure 6. Comparison of UDFS algorithms, UDFS+PDVS, and Windows XP DVFS (measured system power with display off). Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown.

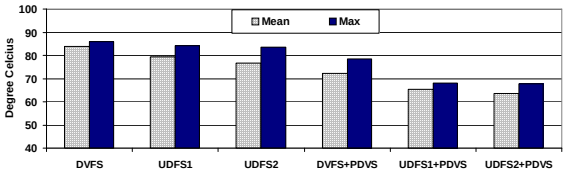
also shows considerable variation among users. On average, we save 15.5% and 29.5% of the power consumption for UDFS1 and UDFS2, respectively. Adding PDVS improves the average power savings to 56.8% and 62.9% over Windows DVFS with UDFS1 and UDFS2, respectively.

On average, the power consumption of the overall system can be reduced by 49.9% for all three applications. This improvement is achieved by combining the UDFS2 scheme (22.1%) and PDVS scheme (27.8%).

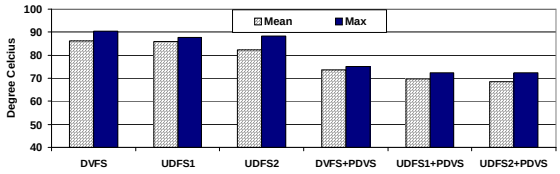
The results presented in the previous section, and in this section, cannot be directly compared because the previous section reports the simulated power consumption of the CPU and this section reports the measured power consumption of the laptop.



(a) Microsoft Powerpoint



(b) 3D Shockwave animation



(c) FIFA game

Figure 7. Mean and peak temperature measurement.

However, some conclusions can be drawn from the data in both sections. For applications like PowerPoint, where the CPU consumes only a small fraction of the system power, the benefit on system power is low. On the other hand, for the applications that originally result in high CPU power consumption, the system power savings can be substantial due to the reduction in dynamic power as well as the operating temperatures and consequently leakage power.

Temperature We used CPUCool [39] to measure CPU temperature in the system. Figure 7 shows the mean and peak temperatures of the system when using the different combinations of DVFS, PDVS, and UDFS schemes. The values reported for UDFS and UDFS+PDVS are the averages over 20 users.

In all cases, the UDFS1 and UDFS2 schemes lower the temperature compared to the Windows native DVFS scheme due to the power reductions we have reported in the previous sections. The maximum UDFS temperature reduction is seen in the case of the UDFS2 scheme used for the Shockwave application ($7.0°$C). On average, for all 3 applications, the UDFS1 and UDFS2 schemes reduce the mean temperature of the system by $1.8°$C and $3.8°$C, respectively. Similarly, PDVS reduces the mean system temperature by $8.7°$C on average for the three applications. The best improvement is observed for the FIFA game, where temperature decreases by $12.6°$C.

The combination of PDVS and UDFS is again synergistic, leading to even greater temperature reductions than PDVS or UDFS, alone. For the Shockwave application, UDFS2+PDVS reduces the mean temperature by $19.3°$C. The average temperature reductions in all three applications by the UDFS1+PDVS and UDFS2+PDVS schemes are $12.7°$C and $13.7°$C, respectively. Our $13.2°$C claim averages these two.

Algorithms	PowerPoint 4 min	3D animation 4 min	FIFA Game 4 min	FIFA Game 4 min
UDFS1	0.35	11.85	5.10	3.42
UDFS2	0.60	14.25	6.50	3.82

Figure 8. Average number of user events.

4.5 Discussion

We now discuss the degree of user interaction needed to make UDFS work, the CPU reliability and longevity benefits of our techniques, and the effects of multitasking.

User interaction While PDVS can be employed without user interaction, UDFS requires occasional feedback from the user. Minimizing the required rate of feedback button presses while maintaining effective control is a central challenge. Our current UDFS algorithms perform reasonably well in this respect, but could be improved. Figure 8 presents the average number of annoyance button presses over a 4 minute period for both versions of UDFS algorithms in our 20 user study. Generally, UDFS2 requires more frequent button presses than UDFS1, because a single press only increments the frequency. The trade-off is that UDFS1 generally spends more time at the maximum frequency and thus is more power hungry. On average, a user pressed a button every 8 minutes for PowerPoint, every 18 seconds for the Shockwave animation, and every 50 seconds for the FIFA game. During the course of the study, for the 3D animation, there were some extreme cases in which the user kept pressing the button even when the processor was running at the highest frequency. This can be explained by the user's dissatisfaction with the original quality of the video or the maximum performance available from the CPU, over which we had no control. If we omit the three most extreme cases from both maximum and minimum number of annoyances, on average a user presses the annoyance button once every 30 seconds for the Shockwave application.

We also note that the system adapts to users quickly, leading to a reduced rate of button presses. In the Figure 8, we show both the first and second 4 minute interval for the FIFA game. The number of presses in the second interval is much smaller than the first. Our interpretation is that once a stable frequency has been determined by the UDFS scheme, it can remain at that frequency for a long time, without requiring further user interaction.

Figure 9 records the average number of voltage transitions for the six different schemes used in our study. A voltage transition is caused either due to a button press or a significant change in operating temperature. For the PowerPoint application, we observe a reduction in the number of transitions because the spikes observed for DVFS do not occur for UDFS1 and UDFS2. On the other hand, the 3D animation and FIFA Game applications have more voltage transitions than observed with Windows native DVFS, because they aim to reduce power by adjusting throttle and, in effect, voltage. In contrast, conventional DVFS keeps

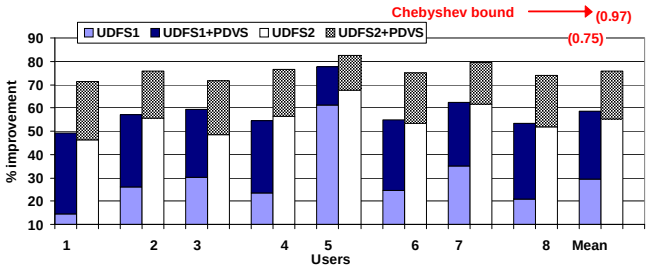


Figure 10. Power improvement in the multitasking environment. Chebyshev bound-based $(1-p)$ values for difference of means from zero are also shown.

the system at the highest frequency during the entire interval. The increase in the number of transitions for the PDVS schemes implemented on top of UDFS are caused by the extra voltage transitions due to changing temperature at a given frequency level.

Although we have not measured the "placebo effect" of providing a way to express irritation with computer performance in this system, we have done so in similar earlier work related to CPU resource management [14], where we found it to be small. We did not measure the degree to which having to provide input itself irritates the user, but we expect that the amount of such direct input will decrease over time, especially using biometric feedback, which we comment on in the related work.

Reliability and longevity In addition to its direct impact on power consumption, our techniques may ultimately improve the lifetime reliability of a system. Earlier research [32] showed that the effect of operating temperature on integrated circuit's mean time to failure (MTTF) is exponential. As we show in Section 4.4, our schemes can reduce the operating temperature by $13.2°$C on average, thereby potentially reducing the rate of failure due to temperature-dependent processes such as electromigration.

Traditionally, the required supply voltage of a processor is reported at the maximum operating temperature of the system. Therefore, at temperatures below the maximum rated temperature, timing slack exists. As long as the current temperature is below the highest rated operating temperature, the operating voltage can be reduced below the rated operating voltage without reducing reliability below that of the same processor operating at the rated voltage and at the maximum temperature.

Multitasking A natural question to ask is whether the extremely simple "press the button" user feedback mechanism we use in UDFS is sufficient for describing user preferences in a multitasking environment. To see the effect of UDFS in a multitasking environment, we conducted a small study ($n = 8$) similar to that of Section 4.1. Instead of several consecutive tasks, the user was asked to watch a 3D animation using Microsoft Internet Explorer while listening to MP3 music using Windows Media Player in compact mode with visualization.

Figure 10 shows the measured system power improvements compared to Windows DVFS. On average, the power consumption of the overall system is reduced by 29.5% and 55.1% for UDFS1 and UDFS2, respectively. Adding PDVS improves the average power savings to 58.6% and 75.7% for UDFS1 and UDFS2, respectively. Although these results are preliminary, combined with the results from the combined PowerPoint+MP3 task described in Section 4.1, they suggest that the simple feedback mechanism is sufficient in a multitasking environment. It is clearly a better proxy of the user's satisfaction than the CPU utilization of the combined task pool.

5 Related work

Dynamic voltage and frequency scaling (DVFS) is an effective technique for microprocessor energy and power control for most modern processors [13, 4]. Energy efficiency has been a major concern for mobile computers. Fei et al. [11] proposed an energy aware dynamic software management framework that improves battery utilization for mobile computers. However, this technique is only applicable to highly adaptive mobile applications. Researchers have proposed algorithms based on workload decomposition [6], but these tend to provide power improvements only for memory-bound applications. Wu et al. [40] presented a design framework of a run-time DVFS optimizer in a general dynamic compilation system. The Razor [9] architecture dynamically finds the minimal reliable voltage level. Dhar et al. [8] proposed adaptive voltage scaling that uses a closed-loop controller targeted towards standard-cell ASICs. These schemes are similar to the PDVS scheme. However, our approach is completely operating system controlled and does not require any architectural modifications and therefore incurs no hardware overhead. Intel Foxton technology [19] provides a mechanism for select Intel Itanium 2 processors to adjust core frequency during operation to boost application performance. However, unlike PDVS it does not perform any dynamic voltage setting.

Other DVFS algorithms use task information, such as measuring response times in interactive applications [24, 42] as a proxy for the user. Unlike our work, which monitors the *user*, Vertigo [12] monitors the application and PICSEL [25] monitors the changes on the display. Xu et al. proposed novel schemes [41] minimizing energy consumption in real-time embedded systems that execute variable workloads. However, they try to adapt to the variability of the workload rather than to the users. AutoDVS [15] is a dynamic voltage scaling (DVS) system for handheld devices. They used user activity as an indicator to detect computationally intensive CPU intervals and use that to drive DVS. In contrast, UDFS uses user activity to directly control the frequency of the system. Ranga et al. proposed energy-aware user interfaces [27] based on usage scenarios, but they concentrated on the display rather than the CPU.

Gupta et al. [14] demonstrated a high variation in user tolerance for different CPU, memory, and disk resource levels. Anand et al. [1] discussed the concept of a control parameter that could be used by the user. However, they focus on the wireless networking domain, not the CPU. Second, they do not propose or evaluate a user interface or direct user feedback. The UDFS component of our work is significantly different as compared to these

Applications	DVFS	DVFS+PDVS	UDFS1	UDFS1+PDVS	UDFS2	UDFS2+PDVS
PowerPoint+Music	11.00	11.00	4.40	4.65	6.55	6.50
3D Animation	3.00	4.00	10.30	11.50	16.3	17.55
FIFA Game	6.00	6.00	18.06	18.05	28.85	29.30

Figure 9. Number of voltage transitions

approaches as it employs direct user feedback instead of a proxy for the user. Theocharous, et al. [35] carefully investigated the use of machine learning techniques to customize power management to users. Shye et al. [29] study the relationship between user satisfaction and hardware performance counters. They also present an algorithm that aims at learning user satisfaction with different frequencies. However, their approach is not dynamic. In addition, they do not evaluate their scheme in combination with other schemes (such as the PDVS scheme we analyze in this work). The same group has also proposed biometric approaches for garnering user satisfaction, which can reduce or eliminate user interaction with the power management system [30].

Dynamic thermal management is an important issue for modern microprocessors due to the expense of cooling solutions. Previous work has discussed microarchitectural modeling and optimization based on temperature [7, 28, 31]. Liu and Svensson made a trade-off between speed and supply voltage [23]. Brooks and Martonosi [5] proposed dynamic thermal management for high-performance processors. Transmeta's Crusoe [36] and Intel's Pentium-M [13] are notable commercial products that uses innovative dynamic thermal management. Leveraging core-to-core process variation is studied in detail by Teodorescu and Torrellas [34]. To the best of our knowledge, however, the PDVS component of our work is the first to consider exploiting process variation via per-CPU customization using profiling [22]. In addition, it is the first scheme to consider temperature in voltage level decisions.

6 Conclusion

We have identified processor and user pessimism as key factors holding back effective power management for processors with support for DVFS. In response, we have developed and evaluated the following new, process- and user-adaptive DVFS techniques: process-driven voltage scaling (PDVS) and user-driven frequency scaling (UDFS). These techniques dramatically reduce CPU power consumption in comparison with existing DVFS techniques. Extensive user studies show that we can reduce power on average by over 50% for single task and over 75% for multitasking workloads compared to the Microsoft Windows XP DVFS scheme. Furthermore, CPU temperatures can be markedly decreased through the use of our techniques. PDVS can be readily used along with any existing frequency scaling approach. UDFS requires that user feedback be used to direct processor voltage and frequency control. PDVS and UDFS are synergistic. UDFS leads to lower average frequencies and PDVS allows great decreases in voltage at low frequencies. We are currently exploring machine learning techniques to develop UDFS algorithms that require even less input from the user.

References

[1] ANAND, M., NIGHTINGALE, E., AND FLINN, J. Self-tuning Wireless Network Power Management. In *The Ninth Annual International Conference on Mobile Computing and Networking (MobiCom'03)* (2003).

[2] BORKAR, S., KARNIK, T., NARENDRA, S., TSCHANZ, J., KESHAVARZI, A., AND DE, V. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)* (2003).

[3] BRAKMO, L. S., O'MALLEY, S. W., AND PETERSON, L. L. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of the Conference on Communications Architectures, Protocols and Applications* (1994), pp. 24–35.

[4] BROCK, B., AND RAJAMANI, K. Dynamic Power Management for Embedded Systems. In *Proceedings of the IEEE SOC Conference* (2003).

[5] BROOKS, D., AND MARTONOSI, M. Adaptive Thermal Management for High-Performance Microprocessors. In *Workshop on Complexity Effective Design* (2000).

[6] CHOI, K., SOMA, R., AND PEDRAM, M. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In *Proceedings of The 2004 International Symposium on Low Power Electronics and Design (ISLPED '04)* (2004), ACM Press, pp. 174–179.

[7] COHEN, A., FINKELSTEIN, F., MENDELSON, A., RONEN, R., AND RUDOY, D. On Estimating Optimal Performance of CPU Dynamic Thermal Management. *IEEE Computer Architecture Letters 2*, 1 (2003), 6.

[8] DHAR, S., MAKSIMOVIC, D., AND KRANZEN, B. ClosedLoop Adaptive Voltage Scaling Controller For Standard Cell ASICs. In *Proceedings of The International Symposium on Low Power Electronics and Design (ISLPED)* (2005), pp. 251–254.

[9] ERNST, D., KIM, N. S., DAS, S., PANT, S., PHAM, T., RAO, R., ZIESLER, C., BLAAUW, D., AUSTIN, T., AND MUDGE, T. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *ACM/IEEE International Symposium on Microarchitecture (MICRO)* (2003).

[10] FALL, K., AND FLOYD, S. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM Computer Communication Review 26*, 3 (1996), 5–21.

[11] FEI, Y., ZHONG, L., AND JHA, N. K. An Energy-aware Framework for Coordinated Dynamic Software Management in Mobile Computers. In *IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (2004).

[12] FLAUTNER, K., AND MUDGE, T. Vertigo: Automatic Performance-Setting for Linux. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)* (December 2002).

[13] GOCHMAN, S., AND RONEN, R. The Intel Pentium M Processor: Microarchitecture and Performance. In *Intel Technology Journal* (2003).

[14] GUPTA, A., LIN, B., AND DINDA, P. A. Measuring and Understanding User Comfort with Resource Borrowing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)* (June 2004).

[15] GURUN, S., AND KRINTZ, C. AutoDVS: an Automatic, General-purpose, Dynamic Clock Scheduling System for Hand-held Devices. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software* (2005), pp. 218–226.

[16] INTEL CORPORATION. Intel Pentium M Datasheet. `http://developer.intel.com/design/mobile/pentium-m/documentation.htm`.

[17] INTEL CORPORATION. Intel Pentium M Processor Thermal Management. `http://www.intel.com/support/processors/mobile/pm/sb/CS-007971.htm`.

[18] JAIDER, M. Notebook Hardware Control Personal Edition. `http://www.pbus-167.com/chc.htm/`.

[19] JOHN WEI. Foxton Technology Pushes Processor Frequency, Application Performance. `http://http://www.intel.com/technology/magazine/computing/foxton-technology-0905.htm`.

[20] LIN, B. *Human-driven Optimization*. PhD thesis, Department of Electrical Engineering and Computer Science, Northwestern University, July 2007. Available as Technical Report NWU-EECS-07-04.

[21] LIN, B., AND DINDA, P. User-driven scheduling of interactive virtual machines. In *Proceedings of the Fifth International Workshop on Grid Computing* (November 2004).

[22] LIN, B., MALLIK, A., DINDA, P., MEMIK, G., AND DICK, R. Power reduction through measurement and modeling of users and cpus: Summary. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)* (June 2007). Extended version appears as Northwestern EECS technical report NWU-EECS-06-11.

[23] LIU, D., AND SVENSSON, C. Trading Speed for Low Power by Choice of Supply and Threshold Voltages. In *IEEE J. Solid-State Circuits* (1993), vol. 28, pp. 10–17.

[24] LORCH, J., AND SMITH, A. Using User Interface Event Information in Dynamic Voltage Scaling Algorithms. In *Technical Report UCB/CSD-02-1190, Computer Science Division, EECS, University of California at Berkeley, August 2002.* (2002).

[25] MALLIK, A., COSGROVE, J., DICK, R., MEMIK, G., AND DINDA, P. PICSEL: Measuring User-Perceived Performance to Control Dynamic Frequency Scaling. In *Proceedings of the ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2008).

[26] MICROSOFT CORPORATION. Performance Logs and Alerts overview. `http://www.microsoft.com/windows2000/en/advanced/help/`.

[27] RANGANATHAN, P., GEELHOED, E., MANAHAN, M., AND NICHOLAS, K. Energy-Aware User Interfaces and Energy-Adaptive Displays. *Computer 39*, 3 (2006), 31–38.

[28] ROHOU, E., AND SMITH, M. Dynamically Managing Processor Temperature and Power. In *2nd Workshop on Feedback Directed Optimization* (Nov 1999).

[29] SHYE, A., OZISIKYILMAZ, B., MALLIK, A., MEMIK, G., DINDA, P., DICK, R., AND CHOUDHARY, A. Learning and Leveraging the Relationship between Architecture-Level Measurements and Individual User Satisfaction. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (2008).

[30] SHYE, A., PAN, Y., SCHOLBROCK, B., MILLER, J. S., MEMIK, G., DINDA, P., AND DICK, R. Power to the people: Leveraging human physiological traits to control microprocessor frequency. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (November 2008).

[31] SKADRON, K., STAN, M. R., SANKARANARAYANAN, K., HUANG, W., VELUSAMY, S., AND TARJAN, D. Temperature-aware Microarchitecture: Modeling and Implementation. *ACM Trans. Archit. Code Optim. 1*, 1 (2004), 94–125.

[32] SRINIVASAN, J., ADVE, S. V., BOSE, P., AND RIVERS, J. A. The Case for Lifetime Reliability-Aware Microprocessors. In *The International Symposium on Computer Architecture(ISCA)* (2004).

[33] STEVENS, W. TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms. In *Internet RFC 2001* (1997).

[34] TEODORESCU, R., AND TORRELLAS, J. Variation-aware application scheduling and power management for chip multiprocessors. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (June 2008).

[35] THEOCHAROUS, G., MANNOR, S., SHAH, N., GANDHI, P., KVETON, B., SIDDIQI, S., AND YU, C.-H. Machine learning for adaptive power management. *Intel Technology Journal 10*, 4 (November 2006).

[36] TRANSMETA CORPORATION. The Technology Behind the Crusoe Processor, (2000).

[37] WAIZMAN, A., AND CHUNG, C. Resonant free Power Network Design using Extended Adaptive Voltage Positioning (EAVP) Methodology. *IEEE Transactions on Advanced Packaging 24*, 3 (August 2001), 236–244.

[38] WANG, Z., AND CROWCROFT, J. Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm. In *ACM Computer Communications Review* (1992).

[39] WOLFRAM PODIEN. CPUCool. `http://www.cpufsb.de/CPUCOOL.HTM`.

[40] WU, Q., REDDI, V., WU, Y., LEE, J., CONNORS, D., BROOKS, D., MARTONOSI, M., AND CLARK, D. W. Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *38th International Symposium on Microarchitecture (MICRO-38)* (2005).

[41] XU, R., MOSS, D., AND MELHEM, R. Minimizing Expected Energy in Real-time Embedded Systems. In *Proceedings of the 5th ACM international conference on Embedded software(EMSOFT)* (2005), pp. 251–254.

[42] YAN, L., ZHONG, L., AND JHA, N. K. User-perceived Latency based Dynamic Voltage Scaling for Interactive Applications. In *Proceedings of ACM/IEEE Design Automation Conference (DAC '05)* (2005).