User-Driven Frequency Scaling

Arindam Mallik, *Student Member, IEEE,* Bin Lin, Gokhan Memik, *Member, IEEE,* Peter Dinda, *Member, IEEE,* and Robert P. Dick, *Member, IEEE*
Department of Electrical Engineering and Computer Science, Northwestern University

Abstract— **We propose and evaluate User-Driven Frequency Scaling (UDFS) for improved power management on processors that support Dynamic Voltage and Frequency Scaling (DVFS), e.g, those used in current laptop and desktop computers. UDFS dynamically adapts CPU frequency to the individual user and the workload through a simple user feedback mechanism, unlike currently-used DVFS methods which rely only on CPU utilization. Our UDFS algorithms dramatically reduce typical operating frequencies while maintaining performance at satisfactory levels for each user. We evaluated our techniques through user studies conducted on a Pentium M laptop running Windows applications. The UDFS scheme reduces measured system power by 22.1%, averaged across all our users and applications, compared to the Windows XP DVFS scheme.**

I. Introduction

Dynamic Voltage and Frequency Scaling (DVFS) is one of the most commonly used power reduction techniques in high-performance processors. DVFS 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 integrates OS-level control, such control is pessimistic about the user. Indeed, it ignores the user, assuming that CPU utilization is a sufficient proxy. A high CPU utilization leads to a high frequency and high voltage, regardless of the user's satisfaction or expectation of performance.

In response to this observation, on which we elaborate in Section II-A, we introduce User-Driven Frequency Scaling (UDFS). This technique uses direct user feedback to drive an online control algorithm that determines the processor frequency (Section II-B). We describe and evaluate two different frequency control algorithms. Previous work [5], [7] has shown that there is variation among users with respect to the satisfactory performance level for a given workload. We exploit this variation to dynamically customize frequency control policies to the user. Unlike previous work, on which we elaborate in Section IV, our approach employs direct feedback from the user during ordinary use of the machine.

We evaluate our techniques through user studies conducted on a modern Pentium M laptop running Windows applications. Our studies, described in detail in Section III, include both single task and multitasking scenarios. The UDFS scheme reduces measured system power by 22.1%, averaged across all our users and applications, compared to the Windows XP DVFS scheme.

Manuscript submitted: 24 Aug. 2006. Manuscript accepted: 8 Oct. 2006. Final manuscript received: 15 Oct. 2006. This work is in part supported by Department of Energy Award DE-FG02-05ER25691 and NSF Grants IIS-0613568, CNS-0551639, CNS-0347941, CCF-0541337, IIS-0536994, CCF-0444405, ANI-0093221, ANI-0301108, and EIA-0224449.

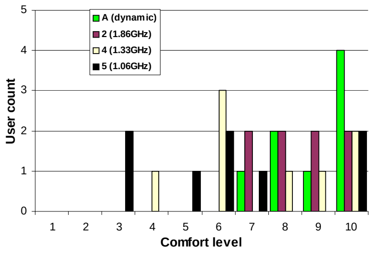
II. User-Driven Frequency Scaling

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 III.

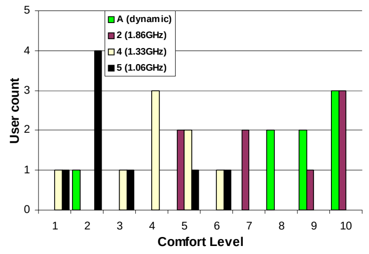
A. Pessimism About The User

Current software that drives DVFS does not consider 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, and power consumption up along with it. CPU-intensive applications also immediately cause an almost instant increase in operating frequency and voltage.

In both cases, the CPU utilization is functioning as a proxy for user comfort. Is it a good proxy? To find out, we conducted a randomized user study of eight users, comparing four processor frequency strategies including dynamic, static low frequency (1.06 GHz), static medium frequency (1.33 GHz), and static high frequency (1.86 GHz). The dynamic strategy is the default DVFS used in Windows XP Professional. Note that the maximum processor 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 carried out three different tasks with the following durations - (a) PowerPoint (4 minutes in total, 1 minute per strategy); (b) Shockwave (80 seconds in total, 20 seconds per strategy); (c) FIFA (4 minutes in total, 1 minute per strategy).



(a)



(b)

Fig. 1. User comfort for (a) Shockwave; (b) FIFA game.

Users verbally ranked their experiences after each task/strategy pair on a scale of 1 (discomforted) to 10 (very comfortable). 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 for the Shockwave animation and the FIFA game (powerpoint is omitted). 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. Not surprisingly, user comfort with any given frequency is highly dependent on the application, but, much less obviously, *there is considerable variation among users in the frequency that is acceptable for any given application.* It is this variation that we seek to exploit. In addition, the comfort levels for the dynamic frequency is practically indistinguishable from the static high frequency, which uses a lower frequency than the dynamic strategy.

B. Technique

Our implementation of user-driven frequency scaling consists of client software that runs as a Windows toolbar task as well as software that implements CPU frequency changes and data recording. The client is a modified version of an earlier tool used to understand user comfort with resource borrowing [5] and implement user-driven scheduling [7]. In the client, the user can express discomfort at any time by pressing the F11 key. These events drive the UDFS algorithm which then uses the Windows API to control CPU frequency. We monitor the CPU frequency using Windows Performance Count and Log [10]. We next describe the UDFS algorithms and strategies.

1) Expectations: 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. Second, even when a user is working with an application, the behavior of the application and the expected performance varies over time. Applications go through phases, each with potentially different computational requirements. Finally, 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 user's needs.

2) UDFS1 Algorithm: UDFS1 is an adaptive algorithm that can be viewed as an extension/variant of the TCP congestion control algorithm [11], [2]. UDFS1 has two state variables: f, the current control value (CPU frequency) and f_t (the current threshold). Adaptation is controlled by three constant parameters: ρ, the rate of decrease, $\alpha = f(\rho)$, the slow start speed, and $\beta = g(\rho)$, the additive decrease speed. Like TCP, UDFS1 operates in three modes, as described below.

- Slow Start (Exponential Decrease): If $f > f_t$, we decrease f exponentially with time (e.g., $f \propto 2^{\alpha t}$).
- User event avoidance (Additive Decrease): If no user feedback is received and $f \leq f_t$, f decreases linearly with time, $f \propto \beta t$.
- User event (Multiplicative Increase): When the user expresses discomfort at level f we immediately set $f_t = f_{t-1}$ and set f to the initial (highest) frequency.

This behavior is virtually identical to that of TCP Reno, except for the more aggressive setting of the threshold. Additionally, 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 and T_{AVI} is the target mean inter-arrival time between user events, as currently preset by us. γ controls the sensitivity to the feedback.

We set our constant parameters ($T_{AVI} = 120, \alpha = 1.5, \beta = 0.8, \gamma = 1.5$) based on the experience of two of the authors using the system. These parameter values were subsequently validated via user studies (Section III). Ideally, we would empirically evaluate the sensitivity of UDFS1 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 excessively 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 validate them by evaluating the whole system with the choices. We observed 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. Otherwise, it slowly decreases.

3) 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 f_i to f_{i+1}. The default interval is 10 seconds for all levels. If the user is irritated at control level f_i, we reset the frequency level to f_{i-1} and we update all of our intervals via:

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

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 III.

III. Evaluation

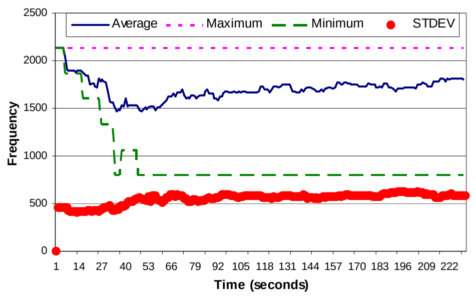
UDFS employs user feedback to customize processor frequency to the individual user. The amount of feedback from the user is reasonable, and declines quickly over time as an application or set of applications is used.

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. We ran a study with 20 users. The user study took around 45 minutes for each user. First, users fill out a questionnaire stating level of experience in different O/S and applications. It was followed by a brief period of acclimation to the performance of our machine. Each user was asked perform the following tasks for UDFS1: Microsoft PowerPoint plus music (4 minutes); 3D Shockwave animation (4 minutes);and FIFA game (8 minutes). The user repeated the same set of tasks for UDFS2.

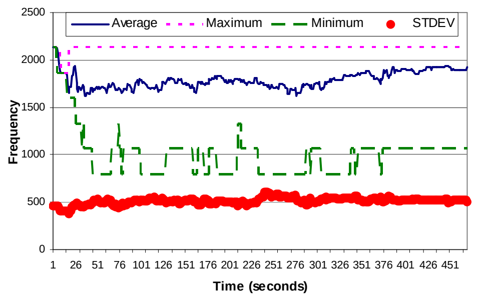
Figure 2 illustrates the performance of the UDFS2 algorithm in our study (UDFS1 and PowerPoint are omitted for space constraints, but are similar). Each graph shows, as a function of time, the minimum, average, maximum, and standard deviation of user-driven CPU frequency, aggregated over our 20 users. Notice that there is large variation in acceptable frequency among the users for the animation and game. For both algorithms it is very rare to see the processor run at the maximum CPU frequency. Even the most sophisticated users were comfortable with running the tasks with lower frequencies than those selected by the dynamic Windows DVFS scheme.

A. CPU Dynamic Power Improvement

We used the system described in Section II-B, recording frequency over time. We then combine this frequency information



(a) UDFS2 - Shockwave



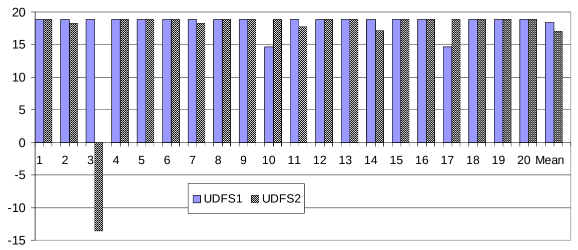
(b) UDFS2 - FIFA Game

Fig. 2. Frequency vs. time, UDFS2, aggregated, 20 users.

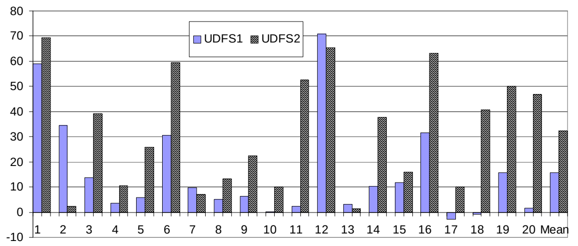
to derive CPU power savings for UDFS. For reference, we used the nominal core voltage given in the datasheet [6] at different operating frequencies. The dynamic power consumption of a processor is directly related to frequency and supply voltage and can be expressed using the formula $P_{dyn} = V^2CF$, which states that dynamic power is equal to the product of voltage squared, capacitance, and frequency.

Figure 3 presents both individual user results and average results for UDFS1 and UDFS2 for three different applications. The vertical axis show the percentage improvement for power over the Windows native DVFS scheme. For the Shockwave animation, we see mixed responses from the users, although on average UDFS1 and UDFS2 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. 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.

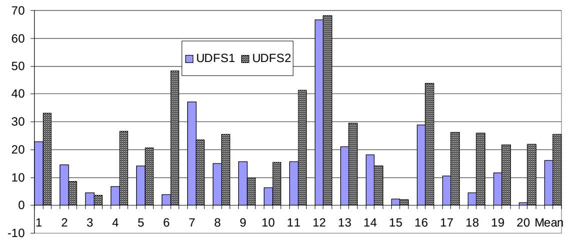
The FIFA game also exhibits considerable variation among users. Using conventional DVFS, the system always runs at the highest frequency. The UDFS schemes try to throttle down the frequency over the time. They therefore reduce the power consumption even in the worst case (0.9% and 2.1% for UDFS1 and UDFS2, respectively) while achieving better improvements, on average (16.1% and 25.5%, respectively). For PowerPoint, UDFS1 and UDFS2 reduce power consumption by an average of 18.4% and 17.0%, respectively. On average, the power consumption can be reduced by 24.9% over existing



(a) PowerPoint Music



(b) 3D Shockwave Animation



(c) FIFA Game

Fig. 3. UDFS power improvement over Windows DVFS.

DVFS schemes for all three applications using the UDFS2 algorithm.

B. System Power Measurement

To further measure the impact of our techniques, we replayed 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. Note that during the measurements, we have turned off the display of the laptop to make our readings closer to the CPU power consumption. Ideally, we would have preferred to measure CPU power directly for comparison with results of the previous section, but we do not have the surface mount rework equipment needed to do so. For the Shockwave animation, UDFS1 and UDFS2 reduce the power consumption by 17.2% and 33.6%, respectively. In the FIFA game, UDFS1 and UDFS2 save 15.5% and 29.5% of the power consumption, respectively. On average, the power consumption of the overall system can be reduced by 22.1% for all three applications across all the users.

We have analyzed the experimental results further to investigate whether the UDFS schemes statistically reduce the power consumption. We applied the student t-test on the power readings observed during the simulations. For both UDFS1 and UDFS2, the student t-test revealed that the mean of the power consumption is reduced with over 0.999 confidence interval for all the studied applications.

IV. Related work

Dynamic voltage and frequency scaling (DVFS) is an effective technique for microprocessor energy and power control [3]. Other DVFS algorithms use task information, such as measured response times in interactive applications [8], [12] as a proxy for the user. In Vertigo [4] the authors proposed a latency-based voltage scaling technique. Unlike Vertigo, we monitor the *user* instead of the application. 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. To the best of our knowledge, the UDFS work is the first to employ direct user feedback instead of a proxy for the user.

V. Conclusion

We have identified user pessimism as a key factor holding back effective power management for processors with support for DVFS. In response, we have developed and evaluated User-Driven Frequency Scaling (UDFS). UDFS techniques dramatically reduce CPU power consumption in comparison with existing DVFS techniques. Extensive user studies show that UDFS reduces the system power by 22.1% on average compared to the Microsoft Windows XP DVFS scheme. More detailed results can be found in our technical report [9].

References

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

[2] 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).

[3] Brock, B., and Rajamani, K. Dynamic Power Management for Embedded Systems. In *Proceedings of the IEEE SOC Conference* (2003).

[4] Flautner, K., and Mudge, T. Vertigo: Automatic Performance-setting for Linux. *SIGOPS Oper. Syst. Rev. 36*, SI (2002), 105–116. http://doi.acm.org/10.1145/844128.844139.

[5] 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).

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

[7] Lin, B., and Dinda, P. Putting the User in Direct Control of CPU Scheduling. Tech. Rep. NWU-EECS-06-07, Department of Electrical Engineering and Computer Science, Northwestern University, August 2006.

[8] Lorch, J. R., and Smith, A. J. 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). citeseer.ist.psu.edu/lorch03using.html.

[9] Mallik, A., Lin, B., Dinda, P., Memik, G., and Dick, R. Process and User Driven Dynamic Voltage and Frequency Scaling. Tech. Rep. NWU-EECS-06-11, Department of Electrical Engineering and Computer Science, Northwestern University, August 2006.

[10] MICROSOFT CORPORATION. Performance Logs and Alerts Overview. http://www.microsoft.com/windows2000/en/advanced/help/.

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

[12] 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* (2005).