PICSEL: Measuring User-Perceived Performance to Control Dynamic Frequency Scaling

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

Masters of Science
in
COMPUTER ENGINEERING

By
Jack Cosgrove

April 2008

ABSTRACT

The ultimate goal of a computer system is to satisfy its users. The success of architectural or system-level optimizations depends largely on having accurate metrics for user satisfaction. I propose to derive such metrics from information that is “close to flesh” and apparent to the user rather than from information that is “close to metal” and hidden from the user. Arindam Mallik, a graduate student in the Electrical Engineering and Computer Science department at Northwestern University, and I describe and evaluate PICSEL, a dynamic voltage and frequency scaling (DVFS) technique that uses measurements of variations in the rate of change of a computer’s video output to estimate user-perceived performance. Adaptive algorithms, one conservative and one aggressive, use these estimates to dramatically reduce operating frequencies and voltages for graphically-intensive applications while maintaining performance at a satisfactory level for the user. I explore the best method to measure video output. Arindam and I evaluate PICSEL through user studies conducted on a Pentium M laptop running Windows XP. Experiments performed with 20 users executing three applications indicate that the measured laptop power can be reduced by up to 12.1%, averaged across all users and applications, compared to the default Windows

XP DVFS policy. User studies revealed that the difference in overall user satisfaction between the more aggressive version of PICSEL and Windows DVFS were statistically insignificant, whereas the conservative version of PICSEL actually improved user satisfaction when compared to Windows DVFS.

ACKNOWLEDGMENTS

I would like to thank my advisor Prof. Robert Dick for his insight, enthusiasm, and patience, and without whom my research would not have been possible. I would also like to thank Arindam Mallik for being a top-flight collaborator, also without whom this research would not have been possible. I would additionally like to thank Profs. Gokhan Memik and Peter Dinda for their invaluable collaboration and support. Many thanks also to Bin Lin who did early work in user-driven frequency scaling, and to the participants in the user study. I would also wish to extend my gratitude to the reviewers and attendees of ASPLOS 2008 for their valuable feedback.

Much love and thanks to my family and friends who have brought me to this point in my life.

Lastly, I would like to thank Profs. Dick, Memik, and Dinda for administering my thesis examination.

This research is supported in part by DOE Awards DE-FG02-05ER25691 and DE-AC05-00OR22725 (via ORNL), NSF Awards CNS-0720691, CNS-0721978, CNS-0715612, IIS-0613568, CNS-0551639, CNS-0347941, CCF-0541337, IIS-0536994, CCF-0444405, 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.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Existing architectures and systems software typically optimize for user satisfaction by employing metrics based largely on instruction throughput (e.g., instructions-per-second). These metrics are used because they are easy to access, easy to compare across platforms, and are believed to reflect user demands for performance at a very low level. However, I will show that low-level information is not as good a proxy for user satisfaction with performance as is high-level information actually observed or perceived by the user. I focus on interactive applications and show that it is possible to infer information about user-perceived performance by measuring changes in video output. This provides better information about the performance level necessary to maintain user satisfaction. Arindam Mallik and I demonstrate the utility of this information in on-line power management.

Processor frequency has a strong effect on power consumption and temperature, directly and also indirectly through the need for higher voltages at higher frequencies. Dynamic Voltage and Frequency Scaling (DVFS) is one of the most commonly used power reduction techniques in modern processors. DVFS varies the frequency and voltage of a microprocessor at runtime to trade off power consumption and processor performance. Specifically, existing DVFS techniques in high-performance processors select an operating point (CPU frequency and voltage) based on the utilization of the processor and other information available to the Operating System (OS) kernel. This approach is often pessimistic regarding user satisfaction, setting the processor frequency higher than necessary to ensure user satisfaction with performance. A high level of CPU utilization or a burst of certain OS events leads directly to a high frequency (and high voltage), regardless of the user's satisfaction with performance. This can produce unnecessary increases in frequency, voltage, power consumption, and temperature.

In response to this observation, Arindam and I have developed a new power management technique that relies upon a more accurate proxy for user performance needs than CPU- or OS-level events, but that is still inexpensive to measure. We estimate user satisfaction with processor performance using information that is "close to flesh" and apparent to the user rather than information that is "close to metal" and hidden from the user. Interface devices are the logical locations for these measurements since they sit between computation and user perception. Video output is particularly useful because it is the user's primary source of information regarding the performance of the computer.

We must note that a user satisfaction-aware optimization policy does not need an absolute metric for user-perceived performance to make decisions. The policy will only make decisions for the architecture on which it is implemented. Using this idea in the context of DVFS, we can compare the displayed performance of applications that change the display at lower frequencies to the displayed performance at the processor's highest available frequency. If the two frequencies result in identical sequences and timing of frames on the screen, then we can safely conclude that these two processor states have the same displayed performance. This maximum frequency satisfies user demands for displayed performance as well as the architecture can, marking a basis for satisfying the user that is fixed for the architecture. Hence, initializing a DVFS policy at the maximum frequency "seeds" the policy with a meaningful level of displayed performance.

To evaluate this idea, we have developed a new power management framework called **PICSEL** (**P**erception-**I**nformed **C**PU performance **S**caling to **E**xtend battery **L**ife) that monitors the rate of change of pixel intensities in the display. An algorithm controlling the processor's operating frequency then makes decisions based upon these rates of change. The algorithm is tested with two configurations: conservative PICSEL (cPICSEL) and aggressive PICSEL (aPICSEL) (Section 3.2). We focus on the DVFS technique implemented by a commercial OS and show that runtime estimation of user-perceived performance using pixel intensities can enhance the effectiveness of the power management scheme. We also show that this approach can result in optimizations that are not possible otherwise.

CHAPTER 2

USER-PERCEIVED PERFORMANCE

The motivation for including user-perceived performance in any objective function is clear: the ultimate goal of computer systems design is to satisfy the user. However, the difficulty in optimizing directly for user-perceived performance is in finding a corresponding metric that can be efficiently measured at runtime. For interactive applications, the events occurring on the input/output devices are good candidates for measuring user satisfaction with performance. However, input events are rare compared to output events. Therefore, considering output to the user is preferable for estimating the performance experienced by the user. Of all the types of output supplied to the user, graphics are used in the highest proportion of applications. Therefore, we considered exploiting properties of the display to estimate user-perceived performance.

Given an application that only changes the display, it is plausible that the frame sequence and frame rate are indications of the user-perceived performance of that application. For example, if there are two architectural alternatives that result in identical frame sequences and frame rates, we can reasonably say that these architectures provide the same user-perceived performance. And even if there is some degradation in video output, this degradation may be offset by reductions in power consumption that ultimately lead to the same user-perceived performance.

Trading off video output for power consumption requires knowledge of when the video output has degraded beyond a level acceptable to the user. Ghinea and Thomas [2005] have done a perceptual study showing that varying both the color depth and the frame rate has a significant effect on user satisfaction with performance. However, extracting the exact frame rate and color depth information would require changes in the application or OS. Hence, we decided to employ a metric that is independent of the application and easily measurable: the rate of pixel intensity change over time. This captures the combination of color depth and frame rate.

I built the experimental framework for testing this metric, and Arindam performed an experiment to understand the relationships between instruction throughput, rate of pixel change, and user-perceived performance. In these experiments, we measured the number of instructions-per-second (IPS) on a 2.13 GHz Intel Pentium

M-based laptop (please see Section 4.1 for further details on the experimental study environment) for three applications: a 3D Shockwave animation, a DVD quality video, and a 3D video game. We also measured the changes in intensity in the red, green, and blue channels of some of the pixels being used to display these applications using the method described in Section 3.1, and averaged these changes together for each time instance to obtain the **Average Pixel Change (APC)**. The procedure to calculate APC is presented in Table 1. We repeated these measurements at all six available processor operating frequencies.

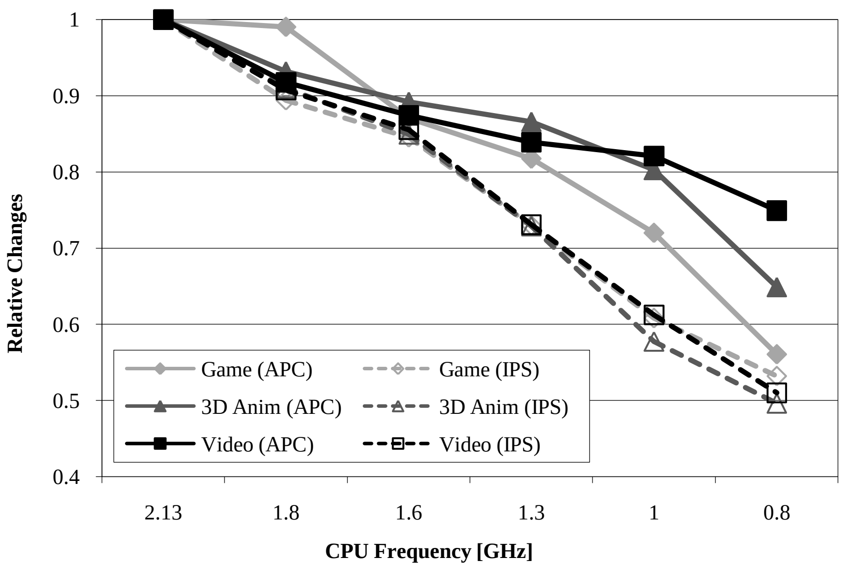


FIGURE 1. IPS AND APC CURVE

TABLE 1. USER-PERCEIVED PERFORMANCE METRICS

Metric	Measurement Procedure
Average Pixel Change (APC)	• Capture the Pixel intensities of the RGB channels of all the pixels in a memory buffer • Calculate the relative changes for all the sampled pixels • The mean of relative changes is the APC
Rate of Average Pixel Change (APR)	$(APC_{T_i} – APC_{T_{i-1}})/(T_i - T_{i-1})$

Figure 1 illustrates the results of this experiment, with the solid lines representing the APC curve and dotted lines representing the IPS curve. As depicted in the figure, the IPS of the system is closely related to the operating frequency of the CPU and is fairly uniform across the three applications. APC is also dependent on the operating frequency, but this dependence is influenced by the application more than IPS is influenced. For the Shockwave application, the effect on APC due to frequency throttling is below 10% for the highest three frequencies.

The Video application shows similar properties. For this task, we could simply set the frequency statically to a lower value without causing noticeable change in the APC. For the game application, the highest two frequency states can sustain the APC value within 10% of its maximum value. However, the lower frequency states cause the APC value to drop suddenly. Most importantly, we see a significant difference between the reduction in IPS and APC. In other words, these results show that the instruction throughput and user-perceived performance are not linearly related. We observe that the APC value of a system can quantify user perceived performance and can be used as a metric for a power management scheme that implements DVFS based on user-perceived performance.

The primary metric we use for user-perceived performance is APC normalized to the total number of pixels in the display. As shown in Figure 1, we observe considerable variation in the APC values across different applications as well as different frequency states. On the other hand, it is also possible that the reduction in the frequency may result in discontinuities in the video output. Previous researchers [Gulliver and Ghinea 2007] have found that jitter and latency are the main sources of user discontent in networked multimedia applications. For example, consider an application that starts skipping frames when the computational power is reduced. In such a case, the APC may not be affected significantly: in a sequence of frames, even if some of the intermediate frames are skipped, the pixel difference between the first and the last frames does not change.

To capture the occurrences of such discontinuities, we record the **Rate of Average Pixel Change (APR)** normalized over the number of pixels. In other words, we calculate the difference between the APC values measured at each time instance. This is the derivative of the APC. Figure 2 illustrates the APR trends observed in three applications used in this paper. When there are glitches during display, the APR value tends to increase rapidly. This is true for applications where video glitches are observed at lower frequencies, namely the Video and 3D Shockwave animation. For other applications (such as the game), we simply observe an overall slowdown and APR values drop in proportion to APC levels. Such applications reduce game jitter at the price of reducing the frame rate. As a result, for this particular application we actually observe a reduction in APR value at lower frequencies as the game's average frame rate is reduced.

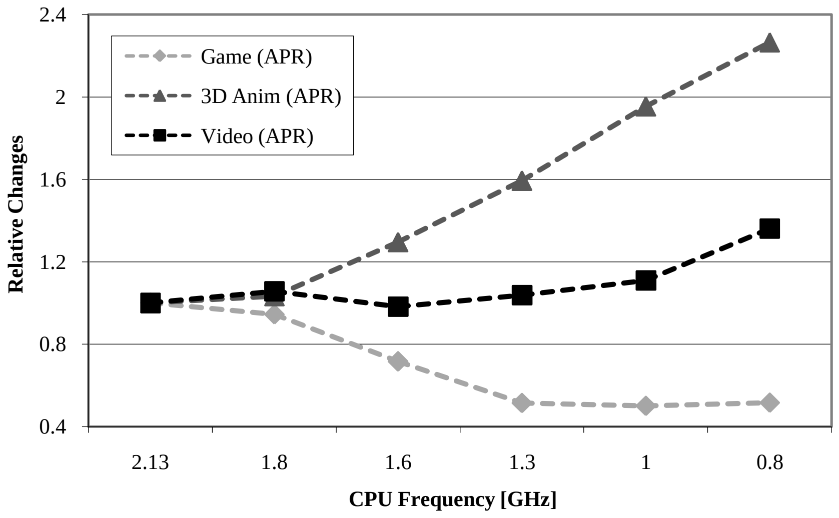


FIGURE 2. APR CURVES FOR THE THREE APPLICATIONS

APR reveals even more pronounced differential behavior across applications than APC. This behavior can permit a DVFS algorithm to differentiate between two applications with similar computational loads and to assign them to different operating frequencies, one potentially lower than would have otherwise been assigned by existing pessimistic DVFS schemes.

CHAPTER 3

PICSEL FRAMEWORK

User-perceived performance-based frequency scaling has two components. First, we have to measure the rate of change in the pixels displayed on the screen. This measurement tool is described in the next section and was built by me. Then, we have to make a throttling decision based on these measurements. The algorithm making this decision is described in Section 3.2 and was designed by Arindam Mallik. In Section 3.3, we describe how PICSEL interacts with the system.

SECTION 3.1

PICSEL DISPLAY ACCESS

There are several methods for accessing the content of a computer display owing to the many steps involved in generating this content. Although more complex schemes are possible, the organization of a generic graphics pipeline in a contemporary computer is shown in Figure 3.

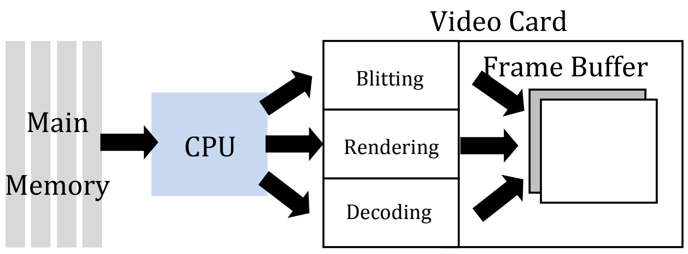


FIGURE 3. GRAPHICS PIPELINE IN A MODERN PC

Application content is read and produced by the CPU, which determines what action should be taken by the video card. The video card then performs operations on the data stream sent by the CPU. The most common operations are blitting, rendering, and decoding. Blitting is a method to erase and redraw sections of a bitmapped image faster than a pixel-by-pixel scan. Rendering uses highly parallel floating-point processors to transform three-dimensional primitives into two-dimensional images. Some video cards also decode compression techniques such as MPEG-2. Each of these different methods may use separate portions of video memory that are invisible to each other until pieced together on the frame buffer by a process called composition. The frame buffer consists of at least two video memory buffers each as large as the monitor screen.

The use of both the processor and the graphics hardware offers two implementation models for PICSEL. One model stores the screenshots in main memory and performs the computations on the CPU. This has the advantage of faster CPU speeds and the disadvantage of transferring more data through the bus that lies between the CPU and graphics hardware. This bus is a major bottleneck in CPU-GPU communication. The other model stores the screenshots in video memory and performs the computations on the GPU. This has the advantage of transferring less data over the bus and the disadvantage of using the slower and increasingly energy-hungry GPU to perform the computation. We compared a main memory/CPU implementation against a video memory/GPU implementation, as well as a hybrid approach, and found that for most screen sizes the CPU implementation was the most efficient. We used this main memory/CPU implementation in our user studies with PICSEL.

SECTION 3.1.1

CPU IMPLEMENTATION ON TARGET HARDWARE

Our target hardware was an IBM Thinkpad T43p with a 2.13 GHz Pentium M-770 CPU, an ATI Mobility FireGL V3200 GPU with PCI-Express, and 1 GB memory running Microsoft Windows XP Professional SP2.

CPU-based PICSEL gathers screen information using the Windows GDI screenshot method, which is simple to implement and can blit any region of the screen to main memory. However, screen content may be missing from sections of the blitted region if those sections were drawn elsewhere in video memory by a rendering or decoding operation. We set our applications to perform rendering and decoding in software in order to capture these operations with a GDI screenshot. This also places the computational load for those operations on the CPU, thus making them subject to CPU frequency scaling. Ideally we would like to consider all the pixels present in the display while calculating the APC. Furthermore, the rate of APC calculation should be same as the rate of frame change in the system. However, both of these constraints introduce heavy computational overhead on the system used for the user studies. Therefore, it was necessary to reduce the size of the captured screen area the CPU load of the capturing process. We decided to limit the overhead to less than 2% CPU utilization. The final captured area is 64×51 pixels, or a scaling down of each dimension of a 1280 by 1024 screen by a factor of 20. This area contained 3276 pixels and was fixed at the center of the screen. We chose to capture a contiguous rectangle of pixels rather than a disjoint grid of pixels because capturing the disjoint grid proved to be much more computationally intensive than capturing the contiguous rectangle, holding the number of pixels constant. Because blitting transfers contiguous blocks of data by design, fewer transfers are necessary to capture an area covered by a fraction of the blocks rather than a screen-size grid covered by all of the blocks.

The sampling frequency for calculating APC was chosen to be 10 Hz, the highest frequency with which our framework did not exceed the 2% CPU utilization threshold for the captured pixel area. There is a computational tradeoff between sampling frequency and capture area. Arindam and I found an acceptable tradeoff through initial testing on the game, video, and Shockwave applications. As we will show in Section 4, these sampling parameters do not prevent PICSEL from capturing the user-perceived performance for our target applications. Nevertheless, it is possible that some applications will not use our capture area; hence it may be desirable to overcome these limitations for other application domains.

After a section of the screen has been captured, it is stored to a memory buffer. This buffer is compared to another buffer containing the previous screen capture, and the intensity differences for the red, green, and blue channels are calculated. Only two buffers are necessary, with each buffer toggling between old and new screen captures. All of the magnitude differences are averaged to obtain the APC.

It is important to understand that this method does not capture each frame and that there are unaccounted-for frames between the two frames used to calculate the intensity difference. This introduces noise into the APC metric. However, since we also measure the APR, we can detect trends in pixel intensity that would otherwise be obscured by the noise. This permits a sampling frequency below the frame rate of the screen.

SECTION 3.1.2

CPU IMPLEMENTATION ON CONTEMPORARY HARDWARE

We also tested CPU-based PICSEL on a Shuttle xPC with a 1.86 GHz Core 2 Duo E6300 CPU, a NVIDIA GeForce 8600GT GPU with PCI-Express, and 1 GB memory running Microsoft Windows XP Professional SP2. This system allowed us to capture an area of 1024 by 1024 pixels at a sampling frequency of 40 Hz while staying under the 2% utilization ceiling. This capture area and sampling frequency are close to actual screen sizes and display refresh rates. These results show that a version of CPU-based PICSEL that captures the entire screen every frame is practical with contemporary hardware.

SECTION 3.1.3

GPU IMPLEMENTATION ON CONTEMPORARY HARDWARE

The raw video output that is sampled by PICSEL resides in video memory, and an implementation that was able to run on the GPU would preclude transferring the sampled output to main memory. This would remove the main bottleneck on performance. With this in mind, I rebuilt PICSEL within NVIDIA's CUDA framework for programming on GPUs. Starting from the "postProcessGL" example in the CUDA SDK [2008] and also using code from the "reduction" example, the GPU-based version of PICSEL stores the captured screen areas in video memory and also performs the computation on these screenshots using the GPU, requiring minimal data transfer back to main memory. GPU-based PICSEL does not use the Windows GDI screenshot and can only capture the contents of one OpenGL application window. This method of capture is acceptable for performance measurements but is not practical for end-use since only one window of the screen is captured.

GPU-based PICSEL renders the screen to a pixel buffer object (PBO). Subsequent PBOs are then compared, with the absolute RGB intensity differences between all of the pixels stored in a buffer in video memory. This halves the amount of memory needed, since the four channels in each pixel (ARGB) require four bytes while the sum of absolute RGB intensity differences can be stored in two bytes. These per-pixel sums are then summed together for the entire capture area. This process is called reduction because it reduces many values into a single value. The reduction portion of PICSEL display access is not efficient on the GPU, and as will be seen, its overhead exceeds the improvements resulting from efficient video memory transfer and packing.

SECTION 3.1.4

HYBRID IMPLEMENTATION ON CONTEMPORARY HARDWARE

The first part of GPU-based PICSEL calculates the per-pixel sum of absolute differences on the GPU. This sum can be expressed using only two bytes, which is half the amount of data needed to express the ARGB components of each pixel. This halves the amount of data that needs to be transferred from video memory to main memory. To exploit this improvement without incurring the cost of GPU-based reduction, we implemented a third hybrid CPU-GPU-based version of PICSEL that calculated the per-pixel sum of absolute intensity differences between frames on the GPU and the sum of these differences on the CPU. This method proved to have an overhead comparable to CPU-based PICSEL and outperformed it at a screen size of 1024 by 1024 pixels.

SECTION 3.1.5

PICSEL BENCHMARKS ON CONTEMPORARY HARDWARE

In order to capture the screen for the GPU-based and hybrid versions of PICSEL, it was necessary to have a simple OpenGL application running in the background. This application consisted of the GLUT teapot rotating in space and placed a negligible load on both the CPU and GPU. All timing information for the benchmarks was obtained from clock cycle counters embedded in the code of the PICSEL versions. Each counter captures all of the elapsed time since the last counter, meaning all other processes running on the machine as scheduled by the OS were included in each measurement. However, we minimized the number of outside processes and any error introduced by this timing technique is uniform across all three PICSEL versions. The sampling frequency for all PICSEL versions was held constant at 10 Hz and the CPU frequency was also held constant at 1.86 GHz.

The timing information shown in Figure 4 is the actual amount of time that elapsed starting from the screen capture and ending with the output of the final summed value for APC for that sample.

The power consumption shown in Figure 5 was measured using a current clamp around one wire of the power supply cable to the computer case, meaning the power consumption of the CPU and GPU were both captured but the power consumption of the peripheral devices with independent power supplies was excluded.

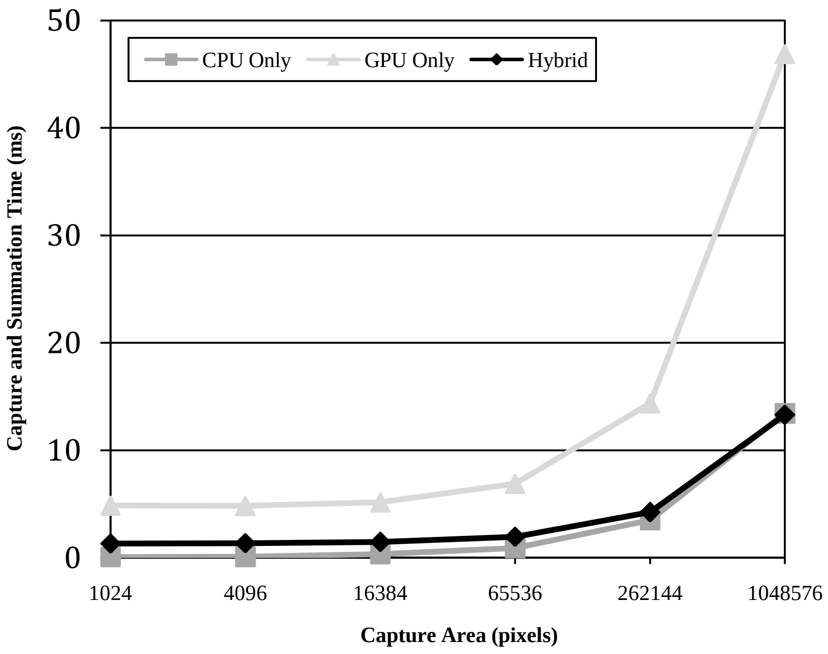


FIGURE 4. SAMPLE TIME FOR PICSEL VERSIONS AS CAPTURE AREA INCREASES

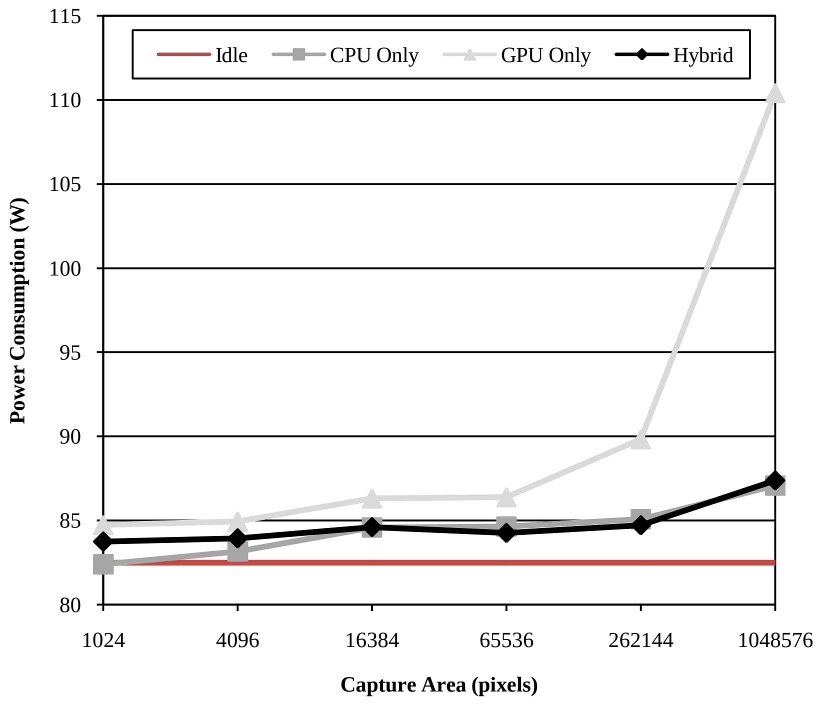


FIGURE 5. POWER CONSUMPTION FOR PICSEL VERSIONS AS CAPTURE AREA INCREASES

These results suggest that CPU-based PICSEL can be efficiently implemented on contemporary processors with high sampling rates and large capture areas. They also show that GPU-based PICSEL is possible, although a CPU-based implementation is generally more efficient on contemporary hardware.

SECTION 3.2

PICSEL ALGORITHM

PICSEL sets the frequency level based on three state variables: f, the current CPU frequency; μ_{APC}, APC in the last time interval; and μ_{APR}, APR in the last time interval. Pixel data are measured at fixed sampling frequency and stored to a file by a background process. Adaptation is controlled by three constant parameters: ρ, the APC change threshold; γ, the APR change threshold; and α, the threshold difficulty level corresponding to each frequency state.

PICSEL can either be in the initialization or the control state. The idea in the initialization stage is to capture information about the APC and APR values observed at the highest frequency. These values will be used as a base case for comparison during the control stage to make throttling decisions. Therefore, during initialization, the CPU frequency is set at the highest value f_{max} for a time interval T_{init}. The APC and APR values of the system over the time interval T_{init} are obtained from the background process and initialized as APC_{init} and APR_{init}. PICSEL then enters the control state in which, at the end of each time interval T_i, the APC and APR of the system over the last interval are obtained from the background process. PICSEL then makes a decision as follows:

```
IF $(APC_{init} - \mu_{APC}) < \rho \times (1-\alpha) \times APC_{init}$
 OR $|APR_{init} - \mu_{APR}| < \gamma \times (1-\alpha) \times APR_{init}$
    Reduce $f$ by one level
    Reset $\alpha$ of the last level to 0
ELSE
    Increase $f$ by one level
    Increment $\alpha$
```

The main idea in this pseudocode is to compare the last observed APC and APR against the APC and APR captured when the processor is executing at the highest frequency. Then, based on the threshold factors defined by ρ, γ, and α, we may conclude that the user-perceived performance is unchanged and try to reduce the frequency and power consumption. Otherwise, out-of-bound values of μ_{APC} and μ_{APR} suggest that user-perceived performance has suffered in the last interval due to low CPU frequency and it is increased accordingly to improve the user-perceived performance.

Factor α helps the algorithm to learn from those times when it decreased the frequency too much. If PICSEL had to increase the processor frequency several times to restore user-perceived performance, α makes it harder to subsequently decrease the frequency. Following every third (n=3) update to α, PICSEL reenters the initialization state. This feature of the algorithm permits PICSEL to gradually adjust to a new set of operating conditions. The constant parameters (T_i = 7 seconds, T_{init} = 10 seconds) were set based on the experience of the authors using the system. α is initialized to zero for each of the frequency levels and is incremented by 0.1 for each frequency boost. We used two variations of the PICSEL algorithm by fixing the $\rho = 0.05$, $\gamma = 0.15$ and $\rho = 0.10$, $\gamma = 0.30$, which correspond to **conservative PICSEL (cPICSEL)** and **aggressive PICSEL (aPICSEL)**, respectively.

Ideally, we would like to empirically evaluate the sensitivity of PICSEL 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 three values of five parameters on 20 users would require 243 days (based on 20 users per day and 25 minutes per 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.

SECTION 3.3

CURRENT IMPLEMENTATION AND INTEGRATION

For our user studies, we disabled the default DVFS policy and give control of the processor frequency to PICSEL. Once PICSEL is active, it executes client software that runs as a Windows toolbar task as well as an API that controls CPU frequency based on user perceived performance. In the client, we log the APC and APR at the background. The API uses these values to control CPU frequency. It is this implementation that we evaluate in the next section.

In its current implementation, PICSEL has some limitations that would require integration with the OS to overcome. One limitation is that PICSEL always controls the processor frequency while it is running. PICSEL should control the processor frequency only if the system is executing an application that modifies the display. This could be accomplished by sampling the video output as usual without making frequency decisions. If the APC/ APR values cross a threshold, PICSEL frequency control is activated. If the APC/APR values drop below a threshold, PICSEL gives the processor frequency control back to the OS. This means that although PICSEL will always monitor the video output, it will only assume control of the processor frequency when appropriate. Another limitation is that PICSEL cannot detect the computational needs of processes without video output, such as code compilation and computational analysis. This limitation is partially overcome if another process with video output is running at the same time as the process without video output. In this case, the frequency will be kept high if a background job takes CPU resources away from the process with video output. In this way, the process with video output is acting as a “canary in a coal mine” whose performance degradation is apparent to PICSEL.

We must note that running background jobs does not cause any problem for PICSEL. In fact, one of our applications targeted in the next section includes a non-interactive background job to demonstrate that our concept is applicable in such cases. If there is a CPU-intensive background job, a reduction in the frequency causes a significant reduction in the APC (even if the interactive application itself is not computationally intensive). Therefore, PICSEL will keep the frequency high. If, on the other hand, the background job is not CPU-intensive, the frequency can be safely reduced, which is exactly the action taken by PICSEL.

CHAPTER 4

EVALUATION

We now evaluate cPICSEL and aPICSEL. We compare against the native Windows XP DVFS scheme, displaying reductions in power consumption and temperature. In Section 4.4, we also present user satisfaction results.

Our evaluations are based on user studies conducted by both Arindam Mallik and me. These are described in Section 4.1. We trace the user's activity on the system during the use of the applications and monitor the responses of Windows DVFS, cPICSEL, and aPICSEL. For studies involving PICSEL, the cPICSEL and aPICSEL algorithms are used online to control the clock frequency in response to APC and APR values. In the rest of this section, we first describe a user study of PICSEL that provides both independent results and traces for later use. Next, we present dynamic CPU power consumption estimates, system power measurements, and temperature measurements.

PICSEL estimates user-perceived performance via APC and APR values and customizes 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 frame buffer readings and the corresponding calculations for measuring user-perceived performance are infrequent, and impose less than 2% computational overhead. PICSEL performs APC and APR readings during user studies, hence all the results presented for PICSEL (including power and user satisfaction) include this overhead and its potential impact on user satisfaction.

SECTION 4.1

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. The Pentium M uses the second generation of Intel's SpeedStep technology, in which six CPU frequency-voltage operating points are available.

Our base case for comparison, the Windows XP Adaptive scheme, is Microsoft's adaptive DVFS scheme for portables/laptops. Adaptive DVFS uses all of the frequency states in the Intel Speedstep technology. Performance needs are measured from heuristics "such as processor utilization, current battery level, use of processor idle states, and inrush current events" [Microsoft Corporation 2003]. In our experimental setup, we ran the computer off AC power, the processor was always active, and we ran trials close enough together to prevent hard disk timeout in order to minimize inrush current events. This leaves processor utilization as the main input to the adaptive DVFS, which makes decisions according to the following algorithm. The algorithm is evaluated whenever the system is in the idle loop, or every 300 ms if running processes prevent the system from entering the idle loop.

```
IF 150 ms have passed since the last frequency state adjustment
    AND Performance has increased by 20% since the last evaluation
        Increase $f$ by one level within the next 10 ms
IF 500 ms have passed since the last frequency state adjustment
    AND Performance has decreased by 30% since the last evaluation
    AND A decrease of frequency state by one operating point will remain above
    50% of the maximum frequency state
        Decrease $f$ by one level within the next 10 ms
```

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

- Watching a 3D Shockwave animation using the Microsoft Internet Explorer web browser. 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 sports game. The game was stored locally. There were no constraints on user gameplay.
- Watching an HD quality movie trailer in Windows Media Player (WMP) while decoding another MPEG movie clip in the

background. Both clips were stored locally and decoding was done in software on the CPU.

We conducted a study with twenty users to evaluate PICSEL. We developed a user pool by advertising our studies within Northwestern University. Some participating users were computer science, computer engineering, or electrical engineering students and others were less experienced with computer use. The studies were double-blind and randomized (i.e., the order of DVFS techniques during the tests were randomized to eliminate any bias that might potentially be introduced by the order of techniques). The studies included intervention by proctors between trials. Each user evaluation lasted about thirty minutes, and consisted of the user doing the following:

1. Filling out a questionnaire that asked the user to rate his or her level of experience in the use of PCs, Windows XP, DVD video, 3D animation, and FIFA 2005 from among the following set: "Power User", "Typical User", or "Beginner".

2. Listening to an explanation of how to play FIFA 2005 and how to rate his or her satisfaction with each application instance.

3. Watching the 3D Shockwave animation three times using cPICSEL, aPICSEL, and Windows DVFS (2 minutes each).

4. Playing FIFA 2005 three times using cPICSEL, aPICSEL, and Windows DVFS (3.5 minutes each).

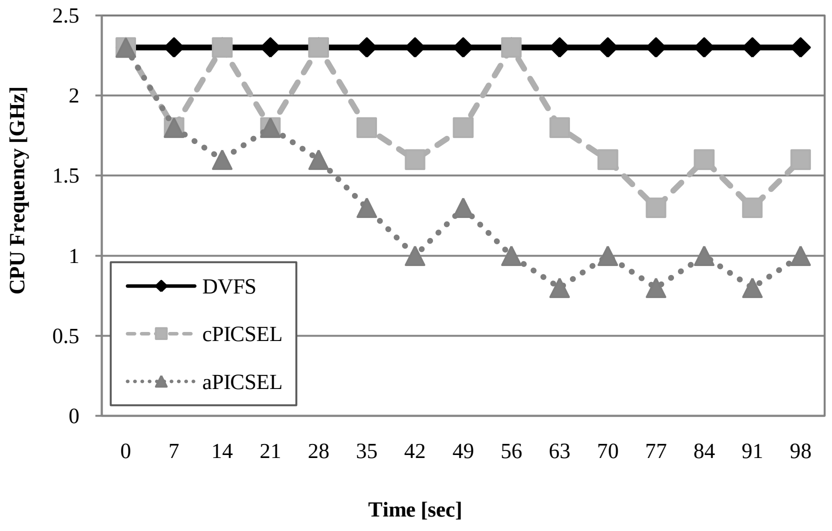
5. Watching the movie trailer three times using cPICSEL, aPICSEL, and Windows DVFS (2 minutes each).

After each application, the users were instructed to assign one of five levels of satisfaction to their experiences with the system performance for each instance of an application. The users were not asked to rank the instances against each other.

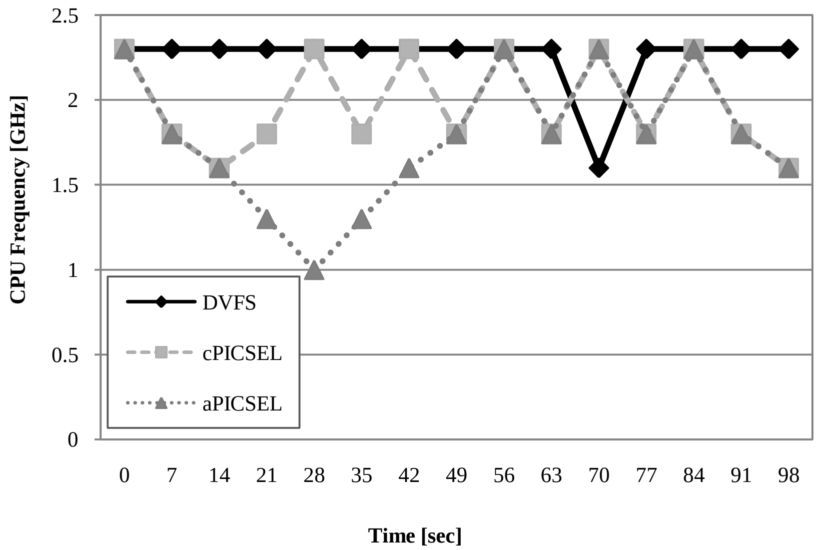
SECTION 4.2

FREQUENCY RESULTS

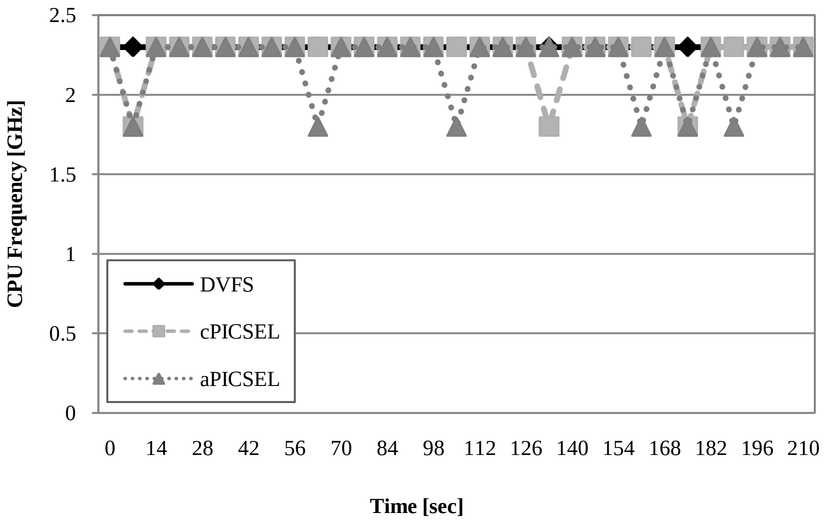
Figure 6 illustrates the performance of the two algorithms for three applications in our study. Each graph shows the CPU frequency for a randomly-selected user as a function of time. Notice that in all the applications both versions of PICSEL generally reduced processor frequency more than the Windows DVFS policy. The amount of frequency reduction varies across applications. PICSEL is most effective for the 3D animation application. As illustrated in Figure 2, this has the least variation in APC and APR values at lower frequencies. As a result, PICSEL greatly reduced the CPU frequency without affecting user-perceived performance. Similar results were observed for the video application. For the game, we observe less processor throttling. This is also expected because the APC values in Figure 1 degrade very quickly for the game and PICSEL can reduce the frequency in few cases. These results show that PICSEL reduces frequency compared to Windows DVFS while maintaining user satisfaction. In Section 4.4, we also analyze user satisfaction with the default Windows DVFS and PICSEL algorithms and show that the user satisfaction is not adversely affected for any of our target applications.



(A) 3D SHOCKWAVE ANIMATION



(B) VIDEO



(C) FIFA GAME

FIGURE 6. FREQUENCY VS. TIME FOR THREE USER TRIALS

SECTION 4.3

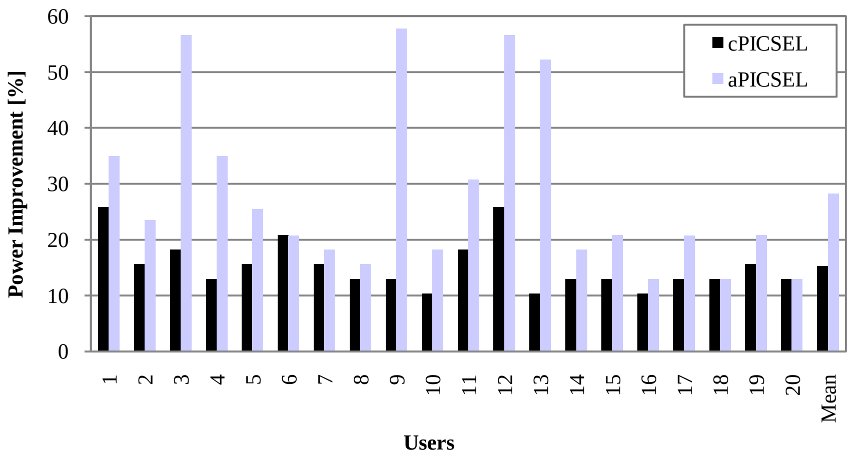
POWER MEASUREMENTS

To analyze the effect of cPICSEL and aPICSEL on the power consumption of the system, we logged the frequency over time during the user studies described in the previous section. We then combine this frequency information with the offline profile to derive power savings for cPICSEL, aPICSEL, and the default Windows XP DVFS policy. In Section 4.3.1 we present the CPU dynamic power savings and in Section 4.3.2 we present the total system power savings. Section 4.3.3 presents the changes in the operating temperatures.

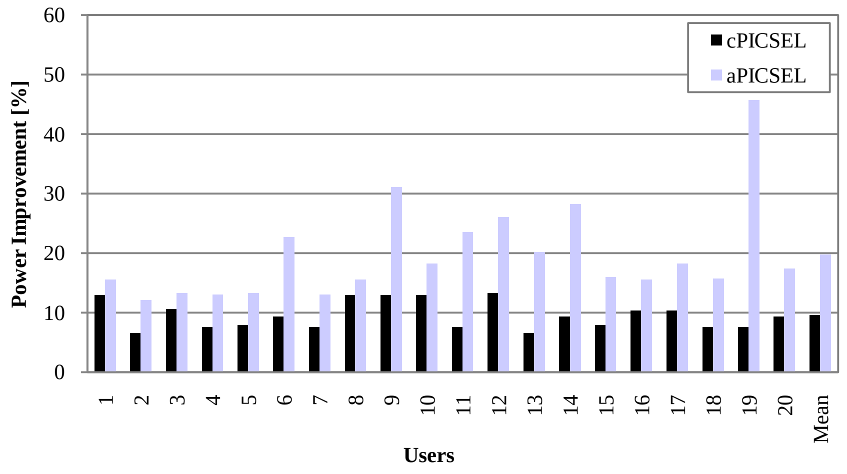
SECTION 4.3.1

CPU DYNAMIC POWER REDUCTION

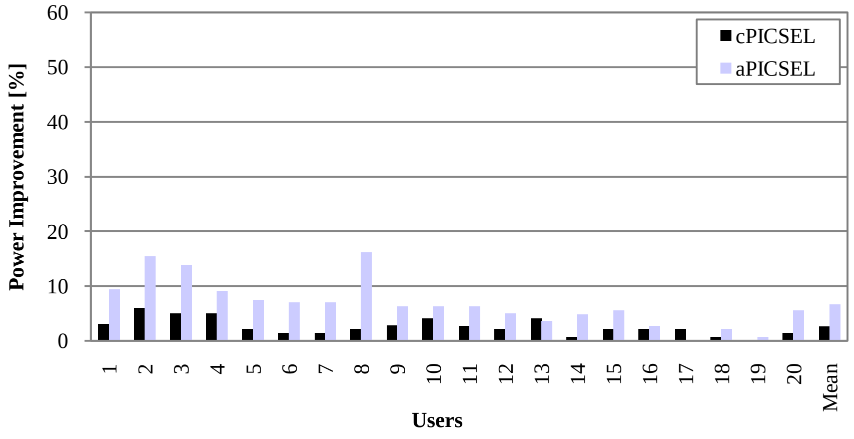
The dynamic power consumption of a processor is directly related to its frequency and supply voltage and can be expressed using the formula $P \propto V^2CF$, which states that power is equal to the product of voltage squared, capacitance, and frequency. By using the frequency traces and the nominal voltage levels on our target processor [Gochman et al. 2003], we calculated the relative dynamic power consumption. Figure 7 presents the CPU dynamic power reduction achieved by the PICSEL algorithms (cPICSEL and aPICSEL) for individual users. The rightmost bars correspond to the savings averaged across users.



(A) 3D SHOCKWAVE ANIMATION



(B) VIDEO



(C) FIFA GAME

FIGURE 7. CPU DYNAMIC POWER REDUCTION WITH CPICSEL AND APICSEL OVER WINDOWS DVFS

For the 3D Shockwave animation, we see mixed responses from the users, although on average PICSEL reduces power by 21.8%. On average, cPICSEL and aPICSEL independently reduce the power consumption by 15.3% and 28.2%, respectively. aPICSEL reduces processor frequency more because it is less likely to identify changes in video output that annoy the user. The results show a considerable variation among different users. This can be explained by the fact that the control agent for APC calculation considers a sampling window of 64×51 pixels at the center of the display window. The relative position of the shockwave player while the user watches the 3D animation plays a role in the calculation of APC and APR. It subsequently affects the decision taken by the PICSEL algorithm. Nevertheless, as we will show in Section 4.4, such variations do not have an impact on user satisfaction.

For the Video application, cPICSEL and aPICSEL reduce power consumption by averages of 9.6% and 19.7%, respectively. This suggests that the Video application is less conducive to frequency throttling than the Shockwave application. User 19 is the only exception where aPICSEL results in a power savings of 45.8%, greater than those for the Shockwave application. For the FIFA game, the average power improvements of 2.6% for cPICSEL and 6.7% for aPICSEL were lower than Video and Shockwave applications, suggesting that the FIFA game was the least conducive to frequency throttling. Note that PICSEL does not reduce the frequency for all the users while they play the FIFA game. For example, cPICSEL does not reduce the frequency for user 19. Similarly, aPICSEL does not reduce the frequency for user 17. This is understandable since the game application has the most steeply-sloped APC curve (Figure 1), meaning a change in frequency will have a larger effect on the game's displayed output than on the displayed output of the other applications.

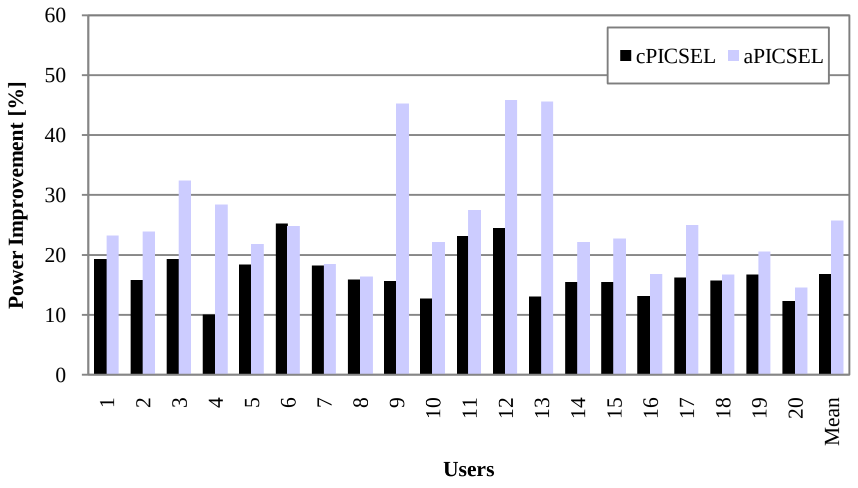
For all three applications, we see that in all cases cPICSEL and aPICSEL lead to power savings compared to Windows DVFS. Averaged over three applications and 20 users, aPICSEL reduces the dynamic power consumption by 18.2%. cPICSEL results in a 9.1% power reduction.

SECTION 4.3.2

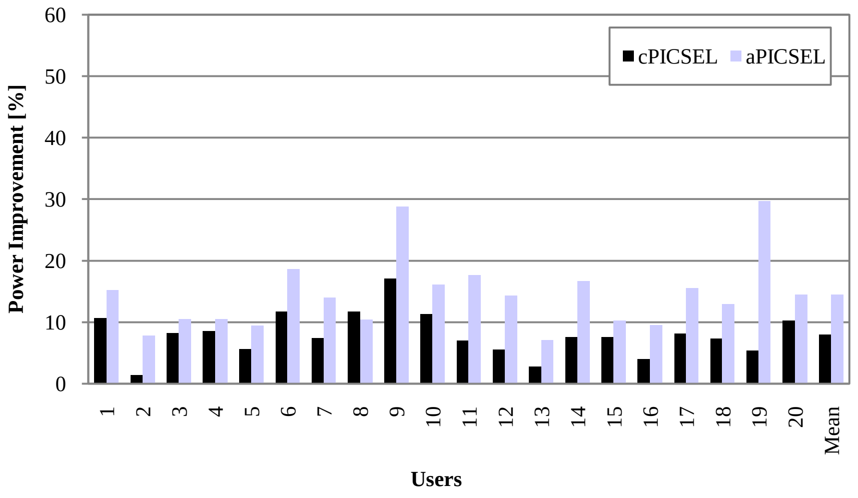
SYSTEM POWER MEASUREMENT

To further measure the impact of our techniques, we replayed the traces from the user studies described in Section 4.3.1 on the laptop. The laptop was connected to a National Instruments 6034E data acquisition board attached to the PCI bus of a host workstation, which permitted us to measure the power consumption of the entire laptop (including other memory, screen, hard disk, etc.). The sampling rate was set to 10 Hz. Each of the user studies was replayed five times to average out variation across trials.

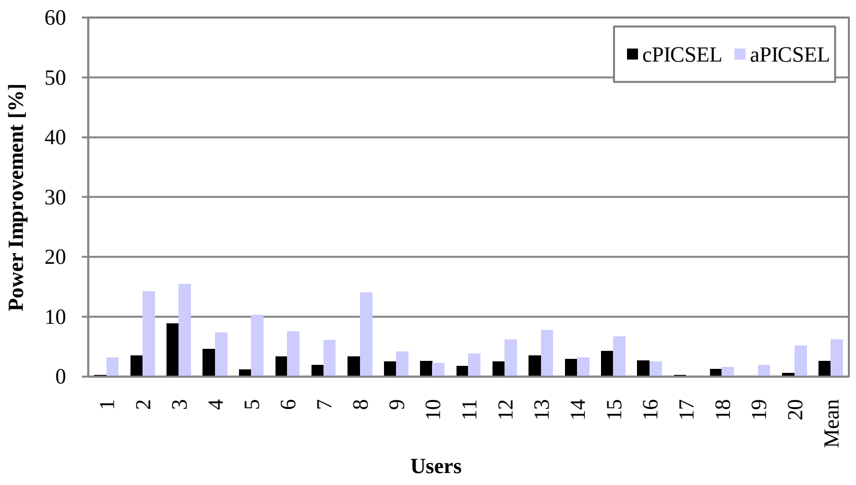
Figure 8 presents the system-level power savings of cPICSEL and aPICSEL relative to Windows DVFS. In general, the reduction in system-level power consumption is similar to the estimated processor dynamic power savings. cPICSEL and aPICSEL reduce power consumption by 16.8% and 25.7% on average for the 3D Shockwave animation, by 8.0% and 14.5% on average for the Video application, and by 2.6% and 6.2% on average for the FIFA game, respectively. On average, aPICSEL reduces system-level power consumption by 12.1%, aggregated over 20 users and three applications. cPICSEL reduces the system-level power consumption by 7.1%.



(A) 3D SHOCKWAVE ANIMATION



(B) VIDEO



(C) FIFA GAME

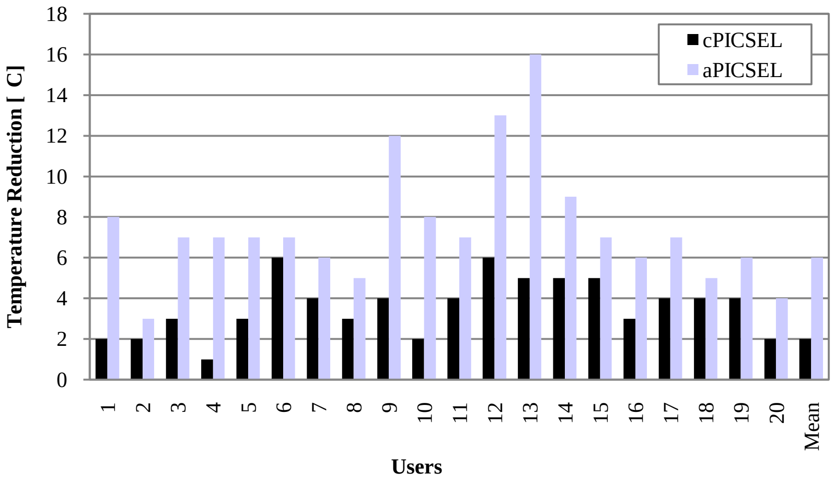
FIGURE 8. SYSTEM POWER REDUCTION WITH CPICSEL AND APICSEL COMPARED TO WINDOWS DVFS

We must note that the dynamic CPU power savings presented in the previous section and the system-level power savings presented in this section cannot be directly compared because the previous section reports the dynamic power consumption of the CPU. This section, on the other hand, reports the measured power consumption of the laptop (which includes leakage power of the CPU as well as all the power consumption of other components in the laptop including memory, screen, hard disk, etc.). However, some conclusions can be drawn from the data in both sections. Applications that result in high CPU dynamic power consumption tend to also observe high system power savings. Clearly, part of the system power reduction comes from the decrease in the CPU dynamic power consumption. Leakage is also reduced due to the decrease in voltage and the decrease in temperature resulting from reduced dynamic power consumption.

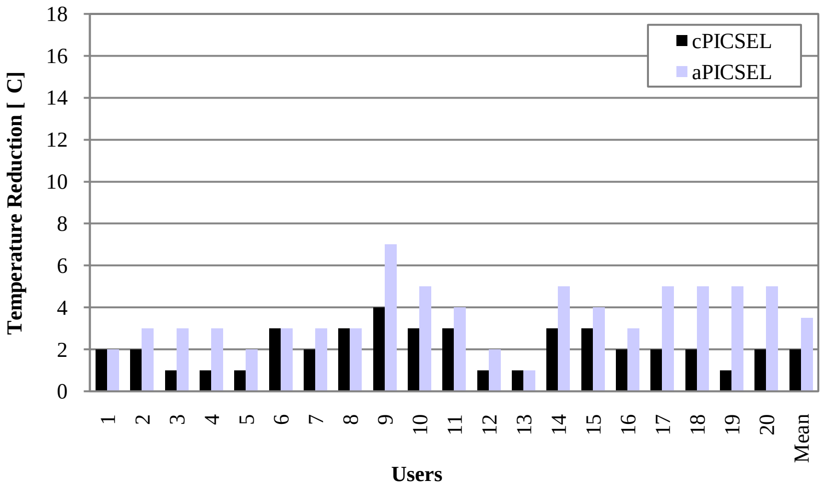
SECTION 4.3.3

CHANGES IN PEAK TEMPERATURE

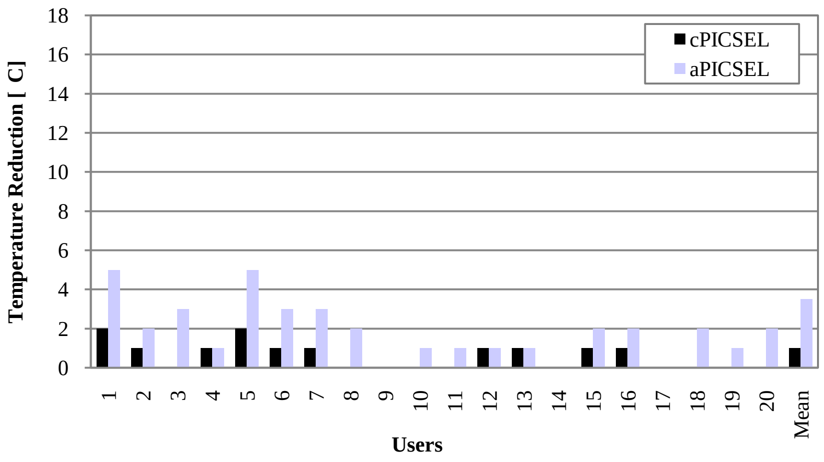
We used CPUCool [Podien 2007], a Windows-based tool that logs temperatures at processor cores, to measure CPU temperature in the system. Figure 9 shows the reductions in peak temperatures of the system when using the cPICSEL and aPICSEL schemes. In all cases, the cPICSEL and aPICSEL schemes lower the temperature compared to the Windows native DVFS scheme due to the power reductions we have reported in the previous sections. The maximum temperature reduction of 16°C is seen in the case of the aPICSEL scheme used for the Shockwave application. On average, for all three applications, cPICSEL and aPICSEL reduce the peak temperature of the system by 1.7°C and 4.3°C, respectively, aggregated over all 20 users.



(A) 3D SHOCKWAVE ANIMATION



(B) VIDEO



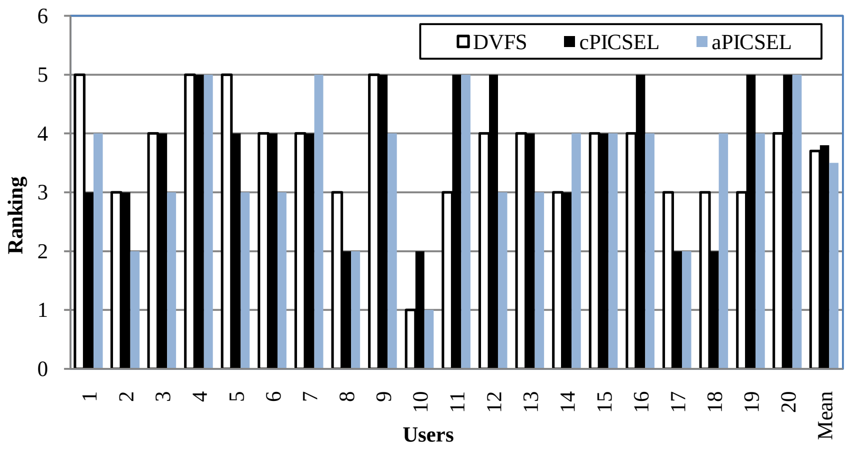
(C) FIFA GAME

FIGURE 9. PEAK TEMPERATURE REDUCTION

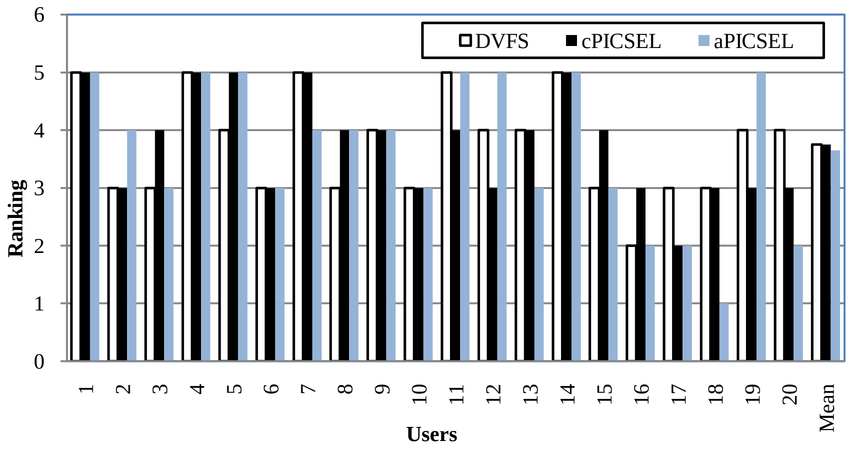
SECTION 4.4

USER SATISFACTION

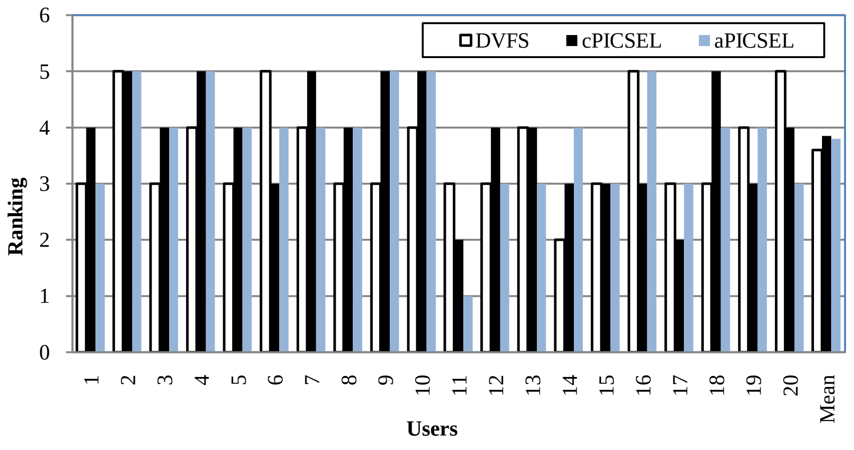
We now discuss the satisfaction levels with the Windows DVFS and PICSEL algorithms for three applications as reported by individual users. During the user study, each participant was asked to give a satisfaction level from 1 to 5 (5 being the most satisfactory performance) for each application. Figure 10 illustrates the ranks awarded by each user. Compared to Windows DVFS, cPICSEL results in slightly better satisfaction levels for all three applications aggregated over 20 users. The student t-test analysis of the results reveals that the difference is not due to chance with 90% confidence. aPICSEL and Windows DVFS provide the same satisfaction (a Student's t-test analysis identifies the two means to be identical with over 99% confidence). On average, aPICSEL is rated highest for the game application (3.8) where it results in the least power reduction. For the Shockwave application, maximum power reduction for the aPICSEL scheme caused it to have the lowest average user satisfaction score (3.5).



(A) 3D SHOCKWAVE ANIMATION



(B) VIDEO



(C) FIFA GAME

FIGURE 10.USER RANKING DISTRIBUTION

We noticed cPICSEL was ranked higher than Windows DVFS although Windows DVFS runs the system at higher frequencies. The only time Windows DVFS will throttle the frequency below what CPU utilization would prescribe is in the case of the temperature crossing a thermal trip point [Microsoft Corporation 2003]. This led to the hypothesis that user dissatisfaction caused by thermal emergencies was the reason for the decreased user satisfaction with Windows DVFS. We ran an experiment in which FIFA 2005 was played under Windows DVFS until the user observed several distinct processor frequency reductions triggered by thermal emergencies. The results of this experiment are shown in Figure 11.

This figure shows processor temperature and frequency when FIFA 2005 is played until it triggers a thermal emergency (about 16 minutes after starting the game). At that point, the processor itself reduces its own frequency to the lowest value possible. This causes a perceivable slowdown in game play and lower instruction throughput. Windows DVFS continues to operate even though it has been overridden by the processor, and lowers its frequency to match the lower instruction throughput. Soon after the processor returns frequency control to Windows DVFS, the frequency is again set to the highest available frequency on the processor. This causes the temperature to rise again quickly, leading to consecutive thermal emergencies.

Both cPICSEL and aPICSEL reduced the occurrence of thermal emergencies, with a total of 51 and 52 thermal emergencies during the game across all users, as compared to a total of 59 thermal emergencies under Windows DVFS. As a result, for processor-intensive applications, PICSEL may deliver better user-perceived performance by reducing the probability of thermal emergencies. The satisfaction results also support this claim: aPICSEL provides the highest satisfaction for the game on average, because for this highly compute-intensive application, aPICSEL allows the greatest reduction in temperature, and resulting thermal emergencies.

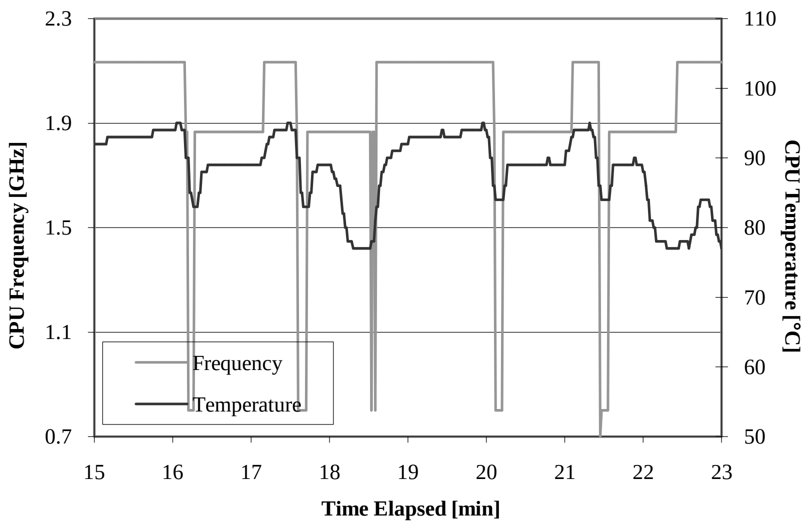


FIGURE 11. THERMAL EMERGENCIES UNDER WINDOWS DVFS

SECTION 4.5

RELATED WORK

Dynamic voltage and frequency scaling (DVFS) is an effective technique for microprocessor energy and power control for most modern processors [Brock and Rajamani 2003, Gochman et al. 2003]. Energy efficiency has been a major concern for mobile computers. Gurun and Krintz [2006] have proposed a new model for estimating energy consumption using hardware and software counters. Fei et al. [2004] 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 [Choi et al. 2004], but these tend to provide power improvements only for memory-bound applications. Wu et al. [2005] presented a design framework of a run-time DVFS optimizer in a general dynamic compilation system. The Razor [Ernst et al. 2003] architecture dynamically finds the minimal reliable voltage level. Dhar et al. [2002] proposed adaptive voltage scaling that uses a closed-loop controller targeted towards standard-cell ASICs. Intel Foxton technology [Wei 2007] provides a mechanism for certain Intel Itanium 2 processors to adjust core frequency during operation to boost application performance. However, unlike PICSEL it does not perform dynamic voltage setting. To the best of our knowledge, no previous DVFS techniques consider user-perceived performance.

Other DVFS algorithms use task information, such as response times in interactive applications [Lorch and Smith 2003, Yan et al. 2005] as a proxy for the user. Vertigo [Flautner and Mudge 2002] monitors application messages and could be used to perform the optimizations implemented in our study. However, compared to Vertigo, our approach is simpler to implement and achieves comparable power savings. Xu et al. [2005] proposed novel schemes to minimize energy consumption in certain real-time embedded systems. However, they try to adapt to the variability of the workload rather than to the users. Gupta et al. [2004] studied user satisfaction with resource borrowing and noted a high variation in user tolerance for any given level of system resources in desktop computing applications. Lin and Dinda [2006] developed a CPU scheduling system that used direct user feedback to exploit this variation. Mallik et al. [2006] showed that this variation also exists for power management, and presented a successful power management approach based on direct user feedback.

Ranganathan et al. [2006] explored using OS-level knowledge about screen content to reduce the power consumption of the screen itself, however no work has been done using knowledge of screen content to control the voltage and frequency of a processor. Gurun and Krintz [2005] looked at OS-level knowledge of user-generated events to control a DVFS scheme but did not use knowledge of screen content. Our work, instead, uses detailed screen information to control the CPU's voltage and frequency levels.

A study of user perception of audio and video quality found that the loss of video frames decreases user satisfaction [Wijesekera 1999]. Frame rate also has a significant effect on user satisfaction, with satisfaction increasing logarithmically with the number of frames displayed per second [Claypool et al., 2006]. Finally, Gulliver and Ghinea [2007] found that both video delay and jitter cause a significant reduction in users' perception of the quality of a video. However, none of these results were used to control processor resources.

CHAPTER 5

CONCLUSION

Any architectural optimization ultimately aims to satisfy the user. Its success or failure rests on the accuracy of its performance metrics as proxies for user satisfaction. In this work, we argue that rather than using metrics that are “close to metal”, architectures should optimize for metrics that are “close to flesh”. To evaluate such an approach, we have developed a new power management technique: **PICSEL** (**P**erception-**I**nformed **C**PU performance **S**caling to **E**xtend battery **L**ife). This technique reduces CPU power consumption in comparison with existing DVFS techniques. User studies show that our technique reduces system-level power consumption of our target laptop on average by 7.1% for a conservative approach (cPICSEL) and 12.1% for the aggressive version (aPICSEL) compared to the Windows XP DVFS scheme. Furthermore, CPU temperatures can be markedly decreased through the use of our techniques. User studies also revealed that the difference in overall user satisfaction between the more aggressive version of PICSEL and Windows DVFS were statistically insignificant, whereas the conservative version of PICSEL improved the users' overall satisfaction when compared to Windows DVFS.

REFERENCES

BROCK, B. AND RAJAMANI, K. 2003. Dynamic power management for embedded systems. In *Proc. of the IEEE SOC Conf. (SOC'03).*

CHOI, K., SOMA, R., AND PEDRAM, M. 2004. Dynamic voltage and frequency scaling based on workload decomposition. In *Proc. of the 2004 Int. Symp. on Low Power Electronics and Design. (ISPLED'04),* 174-179.

CLAYPOOL, M., CLAYPOOL, K., AND DAMAA, F. 2006. The effects of frame rate and resolution on users playing first-person shooter games. In *Proc. of ACM/SPIE Multimedia Computing and Networking (MMCN'06),*

DHAR, S., MAKSIMOVIC, D., AND KRANZEN, B. 2002. Closed-loop adaptive voltage scaling controller for standard cell asics. In *Proc. of the 2005 Int. Symp. on Low Power Electronics and Design (ISPLED'05),* 103-107.

ERNST, D., KIM, N. S., DAS, S., PANT, S., RAO, R., PHAM, T., ZIESLER, C., BLAAUW, D., AUSTIN, T., FLAUTNER, K., AND MUDGE, T. 2003. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proc. of the 36th ACM/IEEE Int. Symp. on Microarchitecture (MICRO'03),* 7-18.

FEI, Y., ZHONG, L., AND JHA, N. K. 2004. An energy-aware framework for coordinated dynamic software management in mobile computers. In *Proc. of the IEEE Computer Society's Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04),* 306-317.

FLAUTNER, K. AND MUDGE, T. 2002. Vertigo: automatic performance-setting for Linux. *ACM SIGOPS Operating Systems Review 36, SI (Winter 2002)*, 105-116.

GHINEA, G. AND THOMAS, J. P. 2005. Quality of perception: user quality of service in multimedia presentations. *IEEE T. Multimedia 7, 4 (Aug. 2005)*, 786-789.

GOCHMAN, S. RONEN, R., ANATI, I., BERKOVITS, A., KURTS, T., NAVEH, A., SAEED, A., SPERBER, Z., AND VALENTINE, R. C. 2003. The Intel

Pentium M processor: Microarchitecture and Performance. *Intel Technology J. 7, 2 (May 2003)*, 21-36.

GULLIVER, S.R. AND GHINEA, G. 2007. The perceptual and attentive impact of delay and jitter in multimedia delivery. *IEEE T. Broadcast 53, 2 (June 2007)*, 449-458.

GUPTA, A., LIN, B., AND DINDA, P. A. 2004. Measuring and understanding user comfort with resource borrowing. In *Proc. of the 13th IEEE Int. Symp. on High Performance Distributed Computing (HPDC'04),* 214-224.

GURUN, S. AND KRINTZ, C. 2005. AutoDVS: an automatic, general-purpose, dynamic clock scheduling system for hand-held devices. In *Proc. of the 5th ACM Int. Conf. on Embedded Software (EMSOFT'05)*, 218-226.

GURUN, S. AND KRINTZ, C. 2006. A run-time, feedback-based energy estimation model for embedded devices. In *Proc. of the Int. Conf. on Hardware/Software Codesign and System Synthesis. (CODES+ISSS'06).*

LIN, B. AND DINDA, P. A. 2006. Towards scheduling virtual machines based on direct user input. In *Proc. of the 1st Int. Workshop on Virtualization Technology in Distributed Computing (VTDC'06)*. See also technical report NWU-EECS-06-07, Northwestern University, EECS.

LORCH, J. AND SMITH, A. 2003. Using user interface event information in dynamic voltage scaling algorithms. In *Proc. of the IEEE Computer Society's Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03),* 46-55.

MALLIK, A., COSGROVE, J., MEMIK, G., DICK, R. P, AND DINDA, P. A. 2008. PICSEL: measuring user-perceived performance to control dynamic frequency scaling. In *Proc. of the 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, (ASPLOS'08).* 70-79.

MALLIK, A., LIN, B. MEMIK, G., DINDA, P. A., AND DICK, R. P. 2006. User-driven frequency scaling. *IEEE Computer Architecture Letters 5, 2 (July 2006)*, 16. A summary of this work also appeared in *ACM SIGMETRICS 2007*.

MICROSOFT CORPORATION. 2003. Windows native processor performance control. *Windows Platform Design Notes (May 2003)*. Retrieved from http://www.microsoft.com/whdc/system/pnppwr/powermgmt/ProcPerfCtrl.mspx.

NVIDIA CORPORATION. 2008. CUDA SDK version 1.1 for Windows XP. Retrieved from http://www.nvidia.com/object/cuda_get.html.

PODIEN, W. CPUCool. Retrieved from http://www.cpu-cool.de/index.html.

RANGANATHAN, P., GEELHOED, E., MANAHAN, M, AND NICHOLAS, K. 2006. Energy-aware user interfaces and energy-adaptive displays. *Computer 39, 3 (March 2006),* 31-38.

WEI, J. Foxton technology pushes processor frequency, application performance. *Technology@Intel Mag. (July 2007)*. Retrieved from http://www.intel.com/technology/magazine/computing/foxton-technology-0905.htm.

WIJESEKERA, D., SRIVASTAVA, J., NERODE, A., FORRSTI, M. 1999. Experimental evaluation of loss perception in continuous media. Multimedia *Systems 7, 6 (Nov. 1999),* 486-499.

WU, Q., MARTONOSI, M., CLARK, D. W., REDDI, V. J., CONNORS, D., WU, Y., LEE, J., AND BROOKS, D. 2005. Dynamic compilation framework for controlling microprocessor energy and performance. In *Proc. of the 38th IEEE/ACM Int. Symp. on Microarchitecture (MICRO'05)*, 271-282.

XU, R., MOSS, D., AND MELHEM, R. 2005. Minimizing expected energy in real-time embedded systems. In *Proc. of the 5th ACM Int. Conf. on Embedded Software (EMSOFT'05)*, 251-254.

YAN, L., ZHONG, L., AND JHA, N. K. 2005. User-perceived latency-based dynamic voltage scaling for interactive applications. In *Proc. of ACM/IEEE Design Automation Conf. (DAC'05),* 624-627.