Learning and Leveraging the Relationship between Architecture-Level Measurements and Individual User Satisfaction

Alex Shye Berkin Ozisikyilmaz Arindam Mallik Gokhan Memik
Peter A. Dinda Robert P. Dick Alok N. Choudhary
Dept. Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

Abstract

The ultimate goal of computer design is to satisfy the end-user. In particular computing domains, such as interactive applications, there exists a variation in user expectations and user satisfaction relative to the performance of existing computer systems. In this work, we leverage this variation to develop more efficient architectures that are customized to end-users. We first investigate the relationship between microarchitectural parameters and user satisfaction. Specifically, we analyze the relationship between hardware performance counter (HPC) readings and individual satisfaction levels reported by users for representative applications. Our results show that the satisfaction of the user is strongly correlated to the performance of the underlying hardware. More importantly, the results show that user satisfaction is highly user-dependent. To take advantage of these observations, we develop a framework called Individualized Dynamic Voltage and Frequency Scaling (iDVFS). We study a group of users to characterize the relationship between the HPCs and individual user satisfaction levels. Based on this analysis, we use artificial neural networks to model the function from HPCs to user satisfaction for individual users. This model is then used online to predict user satisfaction and set the frequency level accordingly. A second set of user studies demonstrates that iDVFS reduces the CPU power consumption by over 25% in representative applications as compared to the Windows XP DVFS algorithm.

1. Introduction

Any architectural optimization (performance, power, reliability, security, etc.) ultimately aims at satisfying the end-user. However, understanding the happiness of the user during the run of an application is complicated. Although it may be possible to query the user frequently, such explicit interaction will annoy most users. Therefore, it would be beneficial to estimate user satisfaction using implicit metrics. Traditionally, computer architects have used implicit metrics such as instructions retired per second (IPS), processor frequency, or the instructions per cycle (IPC) as optimization objectives. The assumption behind these metrics is that they relate in a simple way to the satisfaction of the user. When two systems are compared, it is assumed, for example, that the system providing a higher IPS will result in higher user satisfaction. For some application domains, this assumption is generally correct. For example, the execution time of a long running batch application is largely determined by the IPS of the processor. Hence, increasing IPS will result in an increase in user satisfaction. However, in this paper we show that the relationship between hardware performance and user satisfaction is complex for interactive applications and an increase in a metric like IPS does not necessarily result in an increase in user satisfaction. More importantly, we show that the relationship between hardware performance and user satisfaction is highly user-dependent. Hence, we explore the feasibility of estimating individual user satisfaction from hardware metrics, develop accurate non-linear models to do so, and use these models for run-time power management.

Driving architectural decisions from estimates of user satisfaction has several advantages. First, user satisfaction is highly user-dependent. This observation is not surprising. For example, an expert gamer will likely demand considerably more computational power than a novice user. In addition, each user has a certain "taste"; for example, some users prefer to prolong battery life, while others prefer higher performance. If we know the individual user's satisfaction with minimal perturbation of program execution, we will be able to provide a better experience for the user. Second, when a system optimizes for user satisfaction, it will automatically customize for each application. Specifically, a system that knows the user's satisfaction with a given application will provide the necessary performance to the user. For interactive applications, this may result in significant advantages such as power savings or increased lifetime reliability. For example, one of our target applications exhibits no observable change in performance when the frequency of the processor is set to its lowest level. In this case, our system drastically reduces the power consumption compared to traditional approaches without sacrificing user satisfaction.

Ultimately, our goal is to map microarchitectural information to user satisfaction. Such a map can then be used to understand how changes in microarchitectural metrics affect user satisfaction. Modern microprocessors contain integrated hardware performance counters (HPCs) that count architectural events (e.g., cache misses) as well as a variety of events related to memory and operating system behavior [1-3]. In this work, we aim at finding a mapping from the HPC readings to user satisfaction. We first show that there is a strong correlation between the HPCs and user satisfaction. However, the relationship between the two is often non-linear and user-dependent.

A good estimate of user satisfaction derived from microarchitectural metrics can be used to minimize power consumption while keeping users satisfied. Although utilizing user satisfaction in making architectural decisions can be employed in many scenarios, in this work, we focus on dynamic voltage and frequency scaling (DVFS) [8], which is one of the most commonly used power reduction techniques in modern processors. DVFS make decisions online to change microprocessor frequency and voltage according to processing needs. Existing DVFS techniques in high-performance processors select an operating point (CPU frequency and voltage) based on the utilization of the processor. Like many other architectural optimizations, DVFS is pessimistic about user satisfaction and assumes that the maximum processor frequency is necessary for every process that has a high CPU utilization. We show that incorporating user satisfaction into the decision making process can improve the power reduction yielded by DVFS. Specifically, our contributions in this work follow:

- We unveil a strong relationship between HPCs and user satisfaction for interactive applications;
- We show that this relationship is often non-linear, complex, and highly user-dependent;
- We show that individual user satisfaction can be accurately predicted using neural network models;
- We design *Individualized Dynamic Voltage and Frequency Scaling* (*i*DVFS), which employs user satisfaction prediction in making decisions about the frequency of the processor; and
- We implement and evaluate *i*DVFS on Windows with user studies that show it reduces power consumption compared to Window DVFS.

The paper is organized as follows. In Section 2, we give an introduction to hardware counters. Section 3 describes our user study process. Section 4 presents results showing the relationship between user satisfaction and hardware counters. The predictive user-aware power management scheme is described in Section 5. Section 6 presents the results obtained from user studies. We compare our work with previous studies in Section 7. Section 8 summarizes our contributions.

2. Hardware Performance Counters

Modern microprocessors include integrated hardware performance counters (HPC) for non-intrusive monitoring of a variety of processor and memory system events [1-3]. HPCs provide low-overhead access to a wealth of detailed performance information related to CPU's functional units, caches, main memory, etc. Even though this information is generally statistical in nature, it does provide a window into certain behaviors that are otherwise impractical to observe. For instance, these events include various counts of instructions, cache activity, branch mispredictions, memory coherence operations, and functional unit usage. Several tools and microprocessors have extended this functionality beyond simple event counting. For example, Intel's Itanium processors [2, 18] have features that allow monitoring specific events based on an instruction or data address range, a specific instruction opcode, or execution at specific privilege levels.

Current microprocessors support a limited number of HPCs. For example, the IA-64 architectures only support counting four events at a time [2]. In our experiments, we use the Pentium M processor which only supports two counters at a time. As a result, it is not possible to collect all hardware information simultaneously. One workaround is to time multiplex sets of counters and then scale the values appropriately. Azimi, Stum, and Wisniewski [7] show that time multiplexing up to 10 sets of counters provides statistically significant counter values. Despite this limitation, the low-overhead access to low-level architectural information provided by HPCs is very useful and is often leveraged in run-time profiling and optimization systems [6, 22].

We use WinPAPI, the Windows variant of PAPI [9], to access the HPCs present in the processor. In our study we concentrate on the nine specific performance metrics listed in Table 1. These counters are manually selected as a representative set of the HPCs available on the Pentium M. The choice of using only nine counters is due to a WinPAPI limitation. We collect counter values every 100 ms. WinPAPI automatically time multiplexes and scales the nine event counters.

Table 1. HPCs used in experiments.

PAPI counter	Description
PAPI_TOT_INS	Instructions issued
PAPI_RES_STL	Cycles stalled on any resource
PAPI_TOT_CYC	Total cycles
PAPI_L2_TCM	Level 2 cache misses
PAPI_BTAC_M	Branch target address cache misses
PAPI_BR_MSP	Conditional branches mispredicted
PAPI_HW_INT	Hardware interrupts
PAPI_L1_DCA	Level 1 data cache accesses
PAPI_L1_ICA	Level 1 instruction cache accesses

3. User Study Setup

To explore the relationships between different microarchitectural parameters and user satisfaction, we conduct two sets of studies with 20 users. Our experiments are 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 laptop is tethered to the power outlet during all experiments. 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. For both user studies, we experiment with three types of applications: a 3D Shockwave animation, a Java game, and high-quality video playback. The details of these applications follow:

- Shockwave: 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 is stored locally. Shockwave options are configured so that rendering is done entirely in software on the CPU.
- Java Game: Playing a Java based First Person Shooter (FPS). The users have to move a tank and destroy different targets to complete a mission. The game is CPU-intensive.
- Video: Watching a DVD quality video using Windows Media Player. The video uses high bandwidth MPEG-4 encoding.

Since we target the CPU in this paper, we picked three applications with varying CPU requirements: the Shockwave animation is very CPU-intensive, the Video places a relatively low load on the CPU, and the Java game falls between these extremes.

Our user studies are double-blind, randomized, and intervention-based. We developed a user pool by advertising our studies within Northwestern University. While many of the participants were CS, CE, or EE graduate students, our users included inexperienced computer users as well.

4. User Satisfaction and Hardware Counters

The primary objective of our first user study is to explore the correlation between HPCs and user satisfaction. The monitored hardware counters are listed in Table 1. In this first set of experiments, the users are asked to carry out the three application tasks as described in Section 3. During execution, we randomly change the frequency and ask the users to verbally rank their experience on a scale of 1 (discomfort) to 10 (very comfortable). Users typically provided a satisfaction rating within 5-10 seconds. These satisfaction levels are then recorded along with the HPC readings and analyzed as described in the next section. Then we compute the maximum, minimum, average, range, and the standard deviation of the counter values for up to 5 seconds within the given interval. The end result is a vector of 45 metrics for each satisfaction level reported by the user. Note that since we have performed the user studies with 20 users and three applications, we collected 360 user satisfaction levels.

We then find the correlation of the 45 metrics to the user satisfaction rating by using the formula:

$$r_{x,y} = \frac{N\sum xy - (\sum x)(\sum y)}{\sqrt{[N\sum x^2 - (\sum x)^2][N\sum y^2 - (\sum y)^2]}}$$

Pearson's Product Moment Correlation Coefficient (r) is commonly used to find correlation among two data series (x and y) and results in a value between −1 and 1. If the correlation is negative, the series have negative relationship; if it's positive, the relationship is positive. The closer the coefficient is to either −1 or 1, the stronger the correlation between the variables. Thus, the magnitude of these correlations allows us to compare the relative value of each independent variable in the predicting the dependent variable.

The correlation factors for each of the 45 parameters and the user rating are presented in Appendix A. In summary, we observe a strong correlation between the hardware metrics and user satisfaction rating: there are 21 parameters that correlate with the user satisfaction rating by a factor above 0.7 (all these 21 parameters have a factor ranging between 0.7 and 0.8) and there are 35 parameters with factors exceeding 0.5. On one hand, this result is intuitive; it is easy to believe that metrics representing processor performance relate to user satisfaction. On the other hand, observing the link between such a high-level quantity as measured user satisfaction and such low-level metrics as level 2 cache misses is intriguing.

We classify the metrics (and their correlations with user satisfaction) based on their statistical nature (mean, maximum, minimum, standard deviation, and range). The mean and standard deviation of the hardware counter values have the highest correlation with user satisfaction rating. A t-test analysis shows with over 85% confidence that mean and standard deviation both have higher r values when compared to the minimum, maximum, and range of the HPC values.

We analyze the correlations between the satisfaction results and user. Note that the r value cannot be used for this purpose, as the user numbers are not independent. Instead, we repeatedly fit neural networks to the data collected for each application, attempting to learn the overall mapping from HPCs to user satisfaction. As the inputs to the neural network, we use the HPC statistics along with a user identification for each set of statistics. The output is the self-reported user satisfaction rating. In each fitting, we begin with a three-layer neural network model using 50 neurons in the hidden layer (neural networks are described in more detail in Section 5.1.1). After each model is trained, we perform a sensitivity analysis to find the effect of each input on the output. Sensitivity analysis consists of making changes at each of the inputs of the neural network and observing the corresponding effect on the output. The sensitivity to an input parameter is measured on a 0 to 1 scale, called the relative importance factor, with higher values indicating higher sensitivity. By performing sensitivity analysis, we can find the input parameters that are most important in determining an output parameter, i.e., user satisfaction. During this process, *we consistently find that the user number input has by far the highest relative importance factor.* Averaging across all of our application tasks, the relative importance factor of the user number is 0.56 (more than twice as high as the second factor). This strongly demonstrates that the user is the most important factor in determining the rating.

Finally, to understand the nature of the relationship between the HPCs and the user satisfaction, we analyze the trends for different functions for user satisfaction as provided by the user at each of the processor frequencies.

Table 2. User trend categorization, the number of users in each category for each application.

Applications	Constant	Linear	Step	Staircase	Other
Java Game	0	4	8	4	4
Shockwave	1	5	7	6	1
Video	18	0	0	0	2

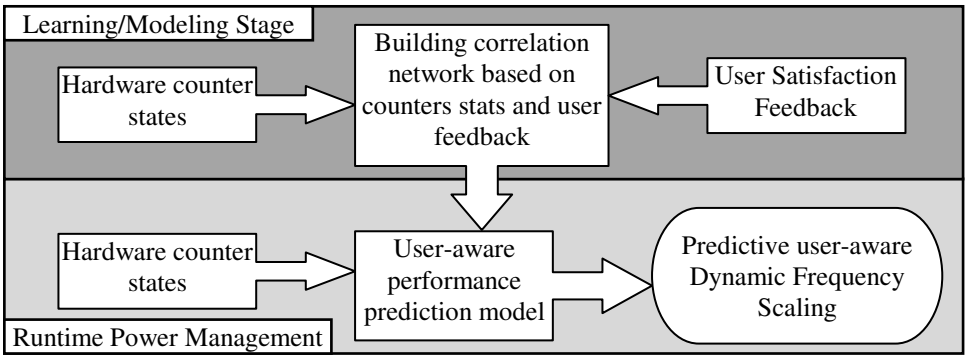


Figure 1. Framework of the predictive user-aware power management.

Table 2 summarizes the trends observed among different users for our three applications. The first row shows the trend curves when we plot user satisfaction against the different frequencies (along x-axis). Most of the trends can be placed in four major categories:

- *Constant* – User satisfaction remains unchanged with frequency. As a result, it is not affected by frequency setting.
- *Linear* – User satisfaction increases linearly with processor frequency.
- *Step* – User satisfaction is the same for a few high frequencies but then plummets suddenly for the remaining lower ones.
- *Staircase* – User satisfaction takes on discrete values that monotonically increase with increasing frequency.

User satisfaction functions that do not match any of the above categories are labeled *Other*. Usually, this is due to user feedback which provides a non-monotonic function

These results reveal several important trends. First, user satisfaction is often non-linearly related to processor frequency. The majority of users provide functions that are categorized as *Constant*, *Step*, or *Staircase*. Note that although *Constant* is a linear function, it does not follow the regular assumption that an increase in a given metric results in an increase in user satisfaction. Second, user satisfaction is application-dependent. For example, for the Video application, almost all of the users report a *Constant* function. On the other hand, the trends for the Java game are distributed among various categories. Finally, user satisfaction is user-dependent. For example, in both the Java game, and the Shockwave animation, users specify utility functions that span multiple categories. This shows that different users have significantly different expectations for the system.

As we will discuss in the next section, these observations have an important effect on the modeling technique we use for learning and predicting user satisfaction.

Overall, this motivational study indicates that

- Hardware counter have a strong correlation with user satisfaction;
- The individual user is the most important factor in determining user satisfaction;
- The relation between hardware performance and user satisfaction is often non-linear; and
- User satisfaction is both application dependent and user dependent.

Based on these observations, we design, implement, and evaluate a DVFS scheme that is based on individual user preferences.

5. Predictive User-Aware Power Management

Based on the initial user study results presented in Section 4, we develop a power management scheme that sets the frequency of the processor based on estimates of user satisfaction. This section presents this predictive user-aware power management scheme, called *Individualized Dynamic Frequency and Voltage Scaling* (*i*DVFS). To implement *i*DVFS, we have built a system that is capable of predicting

a user's satisfaction based on interaction with the system. The framework can be divided into two main stages as depicted in Figure 1:

Learning Stage – The system is initially trained based on reported user satisfaction levels and HPC statistics as described in Section 4. Machine learning models, specifically artificial neural networks, are trained offline to learn the function from HPC values to user satisfaction.

Runtime Power Management – Before execution, the learned model is loaded by the system. During run time, the HPC values are sampled, entered into the predictive model, and then the predicted user satisfaction is used to dynamically set the processor frequency.

5.1 Learning Stage

In its learning stage, our algorithm builds a predictive model based on individual user preferences. The model estimates user satisfaction from the HPCs. In this stage, the user is asked to give feedback (user satisfaction level) while the processor is set to run at different frequency levels. The nature of this training stage is similar to the user study described in Sections 3 and 4. Note that the user study and its survey are repeated for each application. While a user study runs, the nine performance counters are collected and the 45 statistical metrics computed from them are extracted. The combination of these values and the user feedback are used to build the model that will later be used online.

5.1.1 Predictive Model Building

The learning stage helps us gather data that associates an individual user's satisfaction with different hardware performance counter readings and statistics. These instances are then used to build a predictive model that estimates the satisfaction of a particular user from the HPCs. We use neural networks to learn this model. We have also experimented with regression models and decision trees, but the neural networks provided the highest accuracy.

An artificial neural network (NN) is an interconnected group of artificial neurons that uses a mathematical or computational model for information processing based on a connectionist approach to computation. A NN maps a set of p input variables $x_1,\ldots,x_p$ to a set of q response variables $y_1,\ldots,y_q$. It works by simulating a large number of interconnected simple analog processing units that resemble abstract versions of a neuron. Each processing unit (or neuron) computes a weighted sum of its input variables. The weighted sum is then passed through the sigmoid function to produce the units output. We use a three-layer NN model with one input layer, one hidden layer, and one output layer. The well-known Backpropagation algorithm is used to train the neural network from instance data. In the Backpropagation algorithm, the weights between the neurons begin as random values. During the learning phase, training inputs are provided to the NN and the associated output errors are used to adjust neuron weight functions to reduce error.

Our experiments represent a very interesting case for machine learning. Typically, machine learning algorithms are extensively trained using very large data sets (e.g., thousands of labeled training inputs). We would like to use NNs for their ability to learn complex non-linear functions, but do not have a very large data set. For each application-user pair, we only have six training inputs; one for each processor frequency. A training input consists of a set of HPC statistics and a user-provided satisfaction label. When we first began building NN models with all 45 inputs (9 HPC counters with 5 statistics each), we noticed that our models were overly conservative, only predicting satisfaction ratings within a narrow band of values. We used two training enhancements to permit the construction of accurate NN models. First, we simplified the NN by limiting the number of inputs. Large NNs require large amounts of training data to sufficiently learn the weights between neurons. To simplify the NN, we used the two counters that had the highest correlation, specifically PAPI_BTAC_M-avg and PAPI_TOT_CYC-avg (as shown in Appendix A). Second, we repeatedly created and trained multiple NNs, each beginning with different random weights. After 30 seconds of repeated trainings, we used the most accurate NN model. These two design decisions were important in allowing us to build accurate NN models.

5.2 Counter-Based Frequency Control Algorithm

*i*DVFS uses NN models to determine the frequency level. The decision is governed by the following variables: f, the current CPU frequency; μ_{US}, the user satisfaction prediction for the last 500 ms of execution as predicted by the NN model; ρ, the satisfaction tradeoff threshold; α_f, a per-frequency threshold for limiting the decrease of frequency from the current f; M, the maximum user comfort level; and T_i, the time period for re-initialization.

*i*DVFS employs a greedy approach to determine the operating frequency. At each interval, if μ_{US} is within $\alpha_f \rho$ of M, *i*DVFS predicts that the frequency is in a satisfactory state. If μ_{US}-1, the previously predicted user comfort, is also within $\alpha_f \rho$ of M, the system determines that it may be good to decrease the processor frequency; if not, then the system maintains the current frequency. If μ_{US} is not within $\alpha_f \rho$ of M, then the system determines that the current performance is not satisfactory and increases the operating frequency. *i*DVFS uses the α_f thresholds as a hysteresis mechanism to eliminate the ping-pong effect between two states. If the processor rapidly switches between two states N times in a short time interval, the appropriate α_f threshold is decreased to make it harder to decrease to the lower frequency level. This feature of the algorithm ensures that *i*DVFS can adjust to a set of operating conditions very different from those present at initialization but at a rate that is maximally bounded by T_i. The constant parameters (ρ = .15, N = 3, T_i = 20 seconds) were set based on the experience of the authors using the system. α_f thresholds are initialized to one for each of the frequency level and is decremented by 0.1 at each frequency boost.

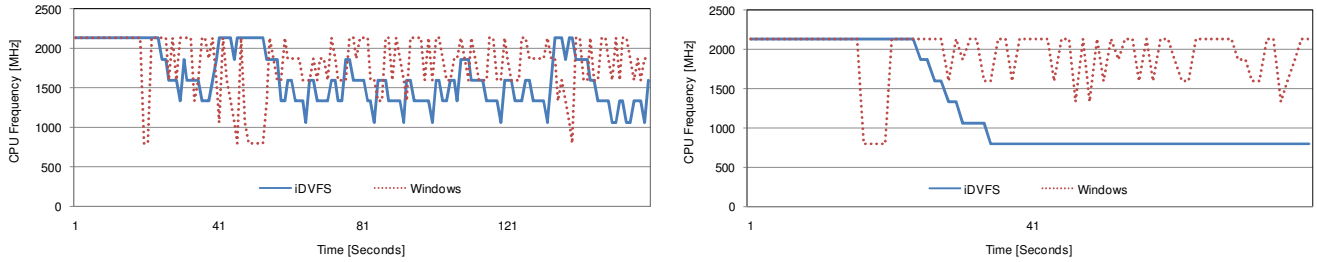


Figure 2. Frequency traces using *i*DVFS and Windows XP DVFS schemes for (a) Java Game and (b) Video.

Ideally, we would like to empirically evaluate the sensitivity of *i*DVFS performance to the selected parameters. However, it is important to note that any such study would require having real users in the loop, and thus would be slow. Testing four values of four parameters on 20 users would require 256 days (based on 20 users/day and 25 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 illustrates the performance of the *i*DVFS algorithm for two of the three applications in our study. Each graph shows, as a function of time, the CPU frequency for a randomly-selected user when playing the Java Game and watching the Video. First, note that the frequency transitions in the two example traces differ greatly from the decisions that Windows XP DVFS makes. The reason is that Windows XP DVFS alters frequency based upon CPU utilization while *i*DVFS alters frequency based upon predicted user satisfaction. *i*DVFS reduces the frequency significantly in the Video application. In this case, the user has indicated high satisfaction with all levels of performance. As shown in Table 2, the Video has the least variation in user satisfaction values at lower frequencies. As a result the *i*DVFS algorithm can reduce CPU frequency without affecting user satisfaction. In both cases, the frequency level follows the satisfaction levels reported by the user and minimizes power consumption with little impact on satisfaction. These traces show that *i*DVFS can successfully adjust the clock frequency throttle according to the user satisfaction derived from the HPCs. For a highly compute-intensive application (such as the Java Game), the reduction in the frequency is minimal because any change in frequency causes a significant reduction in user-perceived performance. For other applications (such as the Video), frequency can be drastically reduced without affecting user satisfaction.

5.3 Implementation, Integration, and Limitations

Currently, we have not integrated *i*DVFS with the operating system (OS). Instead, we have implemented client software that runs as a Windows toolbar task, and manually activate *i*DVFS for our user studies. The client is implemented in a manner that is similar to profile-directed optimization. An initial calibration stage is used for building a model that is used to predict user satisfaction during run time. The current implementation requires direct user feedback in a calibration stage for each user and each application. While this may be cumbersome, there are two points we would like to make. First, we believe that the current system is practical for some users (e.g., heavy gamers will not mind a few minutes of calibration). Second, we argue that explicit user feedback is a viable option. Future work in limiting the feedback and learning effectively from explicit/implicit mechanisms will allow such schemes to be deployed widely.

*i*DVFS has a few limitations that will be eliminated once it is integrated into the OS. First, we provide the client software with per-user, per-application neural network models tailored to the application we are about to invoke. Second, *i*DVFS is currently only intended for interactive applications. The OS has knowledge of users, as well as active applications, and could automatically load the appropriate prediction models for interactive applications during context switches.

WinPAPI only supports system-wide HPC sampling; this includes other programs, background processes, and kernel execution. For our work, we run a single workload on the machine at a time; hence HPC samples correlate to the workload directly. Ideally, the HPC interface would include thread-specific information as well as distinguish between user level and kernel level applications. Other HPC interfaces (i.e., perfmon2 for Linux [4]) also include this support.

The performance of *i*DVFS is largely dependent upon good user input. While this may be a limitation for a current user and application, the user is free to provide new ratings and recalibrate *i*DVFS if the resulting control mechanism causes dissatisfaction.

6. Experimental Results

In this section, we evaluate the predictive user-aware power management scheme with a user study, as described in Section 3. We compare *i*DVFS with the native Windows XP DVFS scheme and report reductions in CPU dynamic power, as well as changes in measured user satisfaction. This is followed by a trade-off analysis between user satisfaction and system power reduction. We report the effect of *i*DVFS on the power consumption and user satisfaction.

We compare *i*DVFS to Windows Adaptive DVFS, which determines the frequency largely based on CPU usage level. A burst of computation due to, for example, a mouse or keyboard event brings utilization quickly up to 100% and drives frequency, voltage, power consumption, and temperature up along with it. CPU-intensive applications cause an almost instant increase in operating frequency and voltage regardless of whether this change will impact user satisfaction. Windows XP DVFS uses six of the frequency states in the Enhanced Intel Speedstep technology, as mentioned in Section 3. Performance requirements are determined using heuristics based on metrics "such as processor utilization, current battery level, use of processor idle states, and inrush current events" [11]. In the Windows native adaptive DVFS scheme, decisions are made according to the algorithm described in Figure 3. We note that this is our best interpretation of the DVFS algorithm described in [11].

```
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
```

Figure 3. Windows XP DVFS Algorithm

6.1 Analysis of User Satisfaction and Power Measurements

To analyze the effect of *i*DVFS on system power consumption, we perform a second set of user studies in which the users are asked to carry out the tasks described in Section 3. This time, the durations of the applications are increased: the Java Game is executed for 2.5 minutes; Shockwave and Video are executed for 1.5 minutes each. The user is asked to execute the application twice, once for Windows XP DVFS and once for *i*DVFS, which loads the individual neural network model for the user/application before the start of the execution. Once the execution completes, the users are asked to rate their satisfaction with each of the systems on a scale of 1 (very dissatisfied) to 5 (very satisfied).

During these experiments, we log the frequency over time. We use these frequency logs to derive CPU power savings for *i*DVFS compared to the default Windows XP DVFS strategy. We have also measured the online power consumption of the entire system, and provide a detailed discussion and analysis of trade-offs between power consumption and user satisfaction.

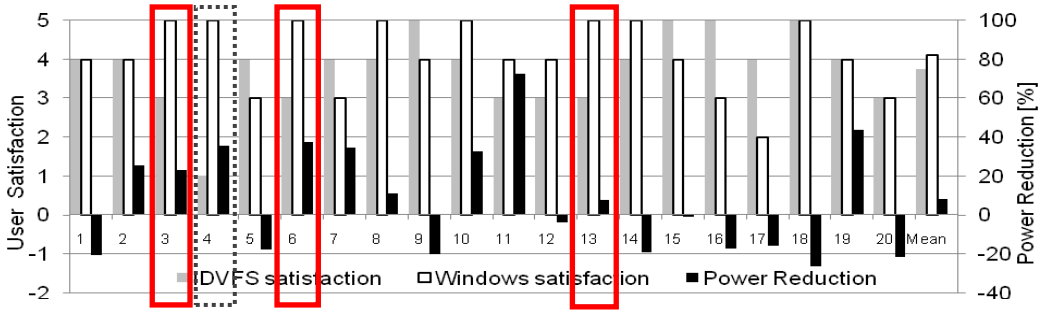
6.1.1 Dynamic Power Consumption and User Satisfaction

The dynamic power consumption of a processor is directly related to frequency and supply voltage and can be expressed using the formula $P = V^2CF$, 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 [16], we calculated the relative dynamic power consumption of the processor. Figure 4 presents the CPU dynamic power reduction achieved by the *i*DVFS algorithm compared to the Windows XP DVFS algorithm for the individual users for each application. It also presents their reported satisfaction levels. To understand the figure, consider a group of three bars for a particular user. The first two bars represent the satisfaction levels for the users for the *i*DVFS (gray) and Windows (white) schemes, respectively. The third bar (black) shows the power saved by *i*DVFS for that application compared to the Windows XP DVFS scheme (for which the scale is on the right of the figure).

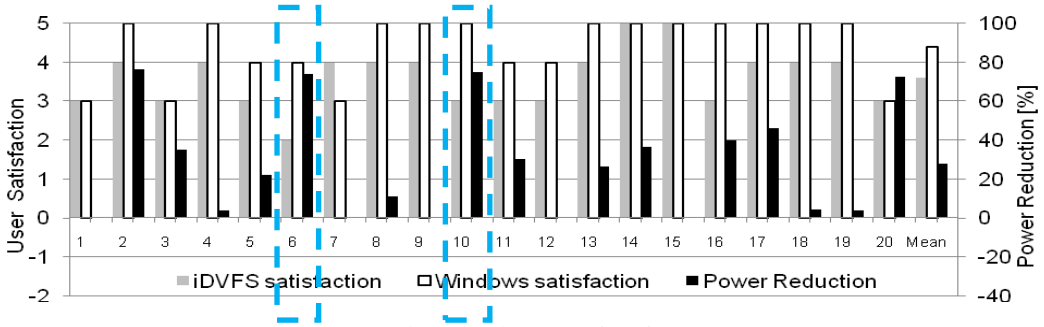
On average, our scheme reduces the power consumption by 8.0% (Java Game), 27.9% (Shockwave), and 45.4% (Video) compared to the Windows XP DVFS scheme. A one-sample t-test of the *i*DVFS power savings shows that for Shockwave and Video, *i*DVFS decreases dynamic power with over 95% confidence. For the Java game, there are no statistically-significant power savings. Correspondingly, the average user satisfaction level is reduced by 8.5% (Java Game), 17.0% (Shockwave), and remains the same for Video. A two-sample paired t-test comparing the user satisfaction ratings from *i*DVFS and Windows XP DVFS indicates that for Java and Video, there is no statistical difference in user satisfaction when using *i*DVFS. For Shockwave, we reduce user satisfaction with over 95% confidence

The combined results show that for Java, *i*DVFS is no different than Windows XP DVFS, for Shockwave, *i*DVFS trades off a decrease in user satisfaction for a decrease in power consumption, and for the Video, *i*DVFS significantly decreases power consumption while maintaining user satisfaction.

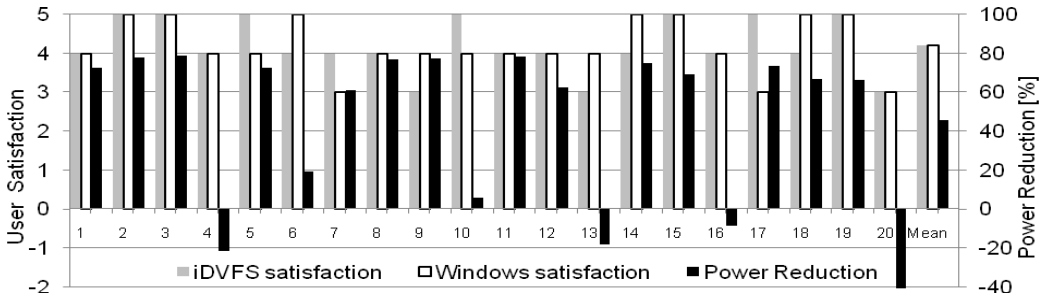
An analysis of the results quickly reveals that the average satisfaction levels are strongly influenced by a few exceptional cases. We have analyzed the cases where there is a difference of more than 1 step between the user ratings. Among these, we found six cases that require special attention. For the Java Game, the training inputs of Users 3, 6, and 13 (solid rectangles in Figure 4) significantly mismatched the performance levels of the processor. Specifically, these users have given their highest ratings to one of the lowest frequency levels. As a result, *i*DVFS performs as the user asks and reduces the frequency, causing dissatisfaction to the user. The cause of dissatisfaction for User 4 (dotted rectangle in Figure 4) was different. Our neural network for that user did not match the training ratings and thus the user was dissatisfied. Similarly, for the Shockwave application, Users 6 and 10 (dashed



(a) Java Game.



(b) Shockwave animation.



(c) Video.

Figure 4. User satisfaction and dynamic power reduction for *i*DVFS compared to the Windows XP DVFS scheme. In the graphs, the individual users are plotted on the horizontal axis. The left vertical axis reflects the reported satisfaction for *i*DVFS and Windows XP DVFS, and the right vertical axis report the percentage reduction in dynamic power of *i*DVFS compared to Windows XP DVFS.

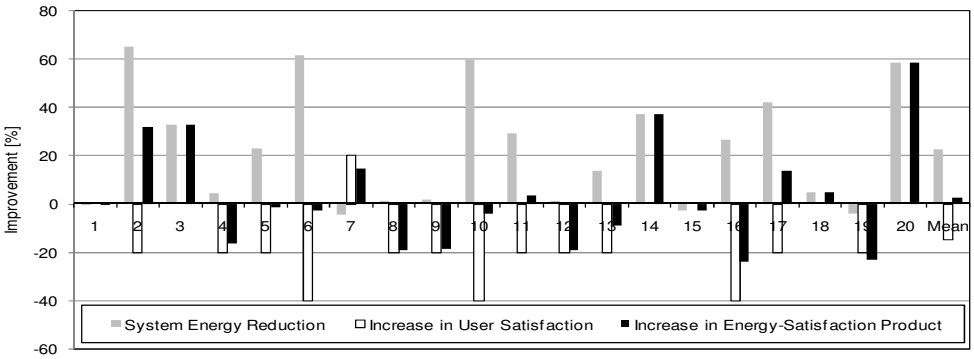


Figure 5. Improvement in energy consumption, user satisfaction, and energy-satisfaction product for the Shockwave application.

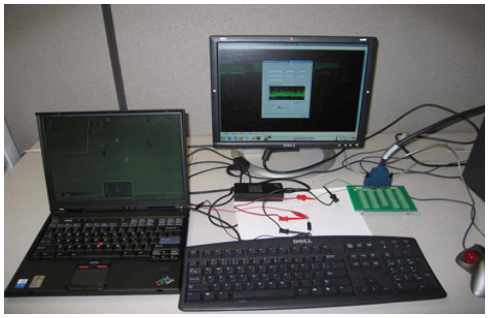


Figure 6. System Power Measurement Setup.

rectangle in Figure 4) provided a roughly constant user satisfaction across the various frequencies. During the user study, however, these Shockwave users highlighted their dissatisfaction when they were able to compare the performance of *i*DVFS to the Windows scheme, which keeps the processor at the highest frequency at all times

It is important to note that such exceptional cases are rare; only 10% of the cases (6 out of 60) fall into this category. Such exceptional cases can be easily captured during a learning phase and eliminated by forcing the user to retake the survey and re-train the model, i.e., training can be repeated until successful. In addition, any dissatisfied user can retrain until a satisfactory performance level is reached. However, our results reveal that such cases will be rare.[1]

User 16's results are likely to be caused by noise and provide a good example of the intricacies of dealing with real users. This user rated *i*DVFS two steps lower than the Windows scheme for Shockwave. At the same time, he/she rated *i*DVFS two grades *higher* for the Java Game application even though *i*DVFS used a lower frequency throughout execution.

Overall, these initial results provide strong evidence that a highly-effective individualized power management system can be developed. Specifically, the results from our user study reveal that

- There exist applications (e.g., Video), for which providing customized performance can result in significant power savings without impacting user satisfaction;
- There exist applications (e.g., Shockwave), for which the users can trade off satisfaction level with power savings. In fact, in the next section, we provide an analysis of such trade-offs; and
- There exist applications (e.g., Java Game), for which traditional metrics in determining the satisfaction is good and *i*DVFS will provide the same performance level and user satisfaction.

[1] We also analyzed the performance of *i*DVFS without considering these extreme cases. Overall, *i*DVFS reduces power consumption by 5.2% (Java Game), 24.0% (Shockwave), and 45.4% (Video). User satisfaction levels were increased by 4.8% (Java Game), reduced by 13.9% (Shockwave), and remained identical for Video (where there are no exceptional cases).

6.1.2 Total System Power and Energy-Satisfaction Trade Off

In the previous section, we have presented experimental results indicating the user satisfaction and the power consumption for three applications. For two applications (Video and the Java Game), we concluded that the *i*DVFS users are at least as satisfied as Windows XP DVFS users. However, for the Shockwave application, we observed that although the power consumption is reduced, this is achieved at the cost of a statistically significant reduction in average user satisfaction. Therefore, a designer needs to be able to evaluate the success of the overall system. To analyze this trade-off, we developed a new metric called the *energy-satisfaction product (ESP)* that works in a similar fashion to popular metrics such as energy-delay product. Specifically, for any system, the ESP per user/application can be found by multiplying the energy consumption with the reported satisfaction level of the user.

Clearly, to make a fair comparison using the ESP metric, we have to collect the total system energy consumption during the run of the application. To extract these values, we replay the traces from the user studies of the previous section. The laptop is connected to a National Instruments 6034E data acquisition board attached to the PCI bus of a host workstation running Windows (and the target applications), which permits us to measure the power consumption of the entire laptop (including other power consuming components such as memory, screen, hard disk, etc.). The sampling rate is set to 10 Hz. Figure 6 illustrates the experimental setup used to measure the system power.

Once the system energy measurements are collected (for both Windows XP DVFS and *i*DVFS), we find the ESP for each user by multiplying their reported satisfaction levels and the total system energy consumption. The results of this analysis are presented in Figure 5. In this figure, we present the reduction in system energy consumption, increase in user satisfaction, and change in ESP for each user. Hence, the higher numbers correspond to improvement in each metric, whereas negative numbers mean that the Windows XP DVFS scheme performed better. Although the ESP improvement varies from user to user, we see that *i*DVFS improves the ESP product by 2.7%, averaged over all users. As a result, we can conclude that Windows XP DVFS and *i*DVFS provide comparable ESP levels for this particular application. In other words, the reduction in user satisfaction is offset at a significant benefit in terms of power savings.

7. Related Work

Dynamic voltage and frequency scaling (DVFS) is an effective technique for microprocessor energy and power control for most modern processors [8, 16]. Energy efficiency has traditionally been a major concern for mobile computers. Fei, Zhong and Ya [14] propose 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 [10], but these tend to provide power improvements only for memory-bound applications. Wu et al. [26] present a design framework for a run-time DVFS optimizer in a general dynamic compilation system. The Razor [13] architecture dynamically finds the minimal reliable voltage level. Dhar, Maksimovic, and Kranzen [12] propose an adaptive voltage scaling technique that uses a closed-loop controller targeted towards standard-cell ASICs. Intel Foxton technology [19] provides a mechanism for select Intel Itanium 2 processors to adjust core frequency during operation to boost application performance. To the best of our knowledge, none of the previous DVFS techniques consider the user satisfaction prediction.

Other DVFS algorithms use task information, such as measured response times in interactive applications [21, 23, 28] as a proxy for the user. Vertigo [15] monitors application messages and can be used to perform the optimizations implemented in our study (although to the best of our knowledge this has not been studied). However, compared to Vertigo, our approach provides a metric/framework that is much easier to use. Xu, Ross, and Melhem propose novel schemes [27] 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. Gupta, Lin, and Dinda [17], and Lin and Dinda [20] demonstrate a high variation in user tolerance for performance in the scheduling context, variation that we believe holds for power management as well.

Mallik et al. [23, 24] show that it is possible to utilize user feedback to control a power management scheme, i.e., allow the user to control the performance of the processor directly. However, their system requires constant feedback from the user. Our scheme correlates user satisfaction with low level microarchitectural metrics. In addition, we use a learning mechanism to eliminate user feedback to make long-term feedback unnecessary. Anand, Nightingale, and Flinn [5] discuss 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.

Sasaki et al. [25] propose a novel DVFS method based on statistical analysis of performance counters. However, their technique needs compiler support to insert code for performance prediction. Furthermore, their technique does not consider user satisfaction while setting the frequency. The primary contribution of our work is to establish the correlation between hardware counters and user satisfaction and utilize this correlation to develop a user-aware DVFS technique.

8. Conclusion

Through extensive user studies, we have demonstrated that there is a strong, albeit usually nonlinear, link between low-level microarchitectural performance metrics, as measured by hardware performance counters (i.e., "close to the metal" numbers), and user satisfaction (i.e., "close to the flesh" numbers) for interactive applications. More importantly, we show that the link is highly user-dependent. This variation in user satisfaction indicates potential for optimization. Using neural networks, we learn per-user per-application functions (which might be called "metal to flesh functions") that map from the hardware performance counters to individual user satisfaction levels. This result in a computer system that can uses small amounts of *explicit* user feedback, and then *implicitly* learns from the feedback to make online predictions of user satisfaction. We demonstrate the utility of this implicit feedback by employing it in a user-aware DVFS algorithm. Experimental results, and analysis of user studies, show that there are interactive applications for which knowledge of user satisfaction permits power consumption savings. Others present an interesting trade-off between user satisfaction and power savings. Overall, our system reduces the power consumption of Windows XP DVFS by over 25%, while only affecting user satisfaction in one application.

9. Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. We are also very grateful to the many users who volunteered their time for the user studies in this paper. 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.

10. References

[1]. *BIOS and Kernel Developer's Guide for AMD Athlon64 and AMD Opteron Processors*. 2006, AMD.

[2]. *Intel Itanium 2 Processor Reference Manual: For Software Development and Optimization*. 2004, Intel Corporation.

[3]. *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide*. 2007, Intel Corporation.

[4]. *perfmon2: the hardware-based performance monitoring interface for Linux:* http://perfmon2.sourceforge.net/.

[5]. Anand, M., E. Nightingale, and J. Flinn, *Self-tuning Wireless Network Power Management*, in *The Ninth Annual International Conference on Mobile Computing and Networking (MobiCom'03)*. 2003: San Diego, California, USA.

[6]. Anderson, J.M., L.M. Berc, J. Dean, S. Ghemawat, M.R. Henzinger, S.-T.A. Leung, R.L. Sites, M.T. Vandevoorde, C.A. Waldspurger, and W.E. Weihl, *Continuous Profiling: Where Have All the Cycles Gone?* 1997, Digital Equipment Corporation Systems Research Center.

[7]. Azimi, R., M. Stumm, and R.W. Wisniewski, *Online Performance Analysis by Statistical Sampling of*

Microprocessor Performance Counters. International Conference on Supercomputing, 2005.

[8]. Brock, B. and K. Rajamani. *Dynamic Power Management for Embedded Systems*. in *Proceedings of the IEEE SOC Conference*. 2003. Portland, Oregon, USA.

[9]. Browne, S., J. Dongarra, N. Garner, G. Ho, and P. Mucci, *A Portable Programming Interface for Performance Evaluation on Modern Processors.* The International Journal of High Performance Computing Applications, 2000. **14**(3): p. 189-204.

[10]. Choi, K., R. Soma, and M. Pedram, *Dynamic Voltage and Frequency Scaling based on Workload Decomposition.* Proceedings of The 2004 International Symposium on Low Power Electronics and Design (ISLPED '04), 2004: p. 174-179.

[11]. Corporation, M., *Windows Native Processor Performance Control*, in *Windows Platform Design Notes*. 2002, Microsoft Corporation.

[12]. Dhar, S., D. Maksimovic, and B. Kranzen, *ClosedLoop Adaptive Voltage Scaling Controller For Standard Cell ASICs.* Proceedings of The International Symposium on Low Power Electronics and Design (ISLPED) 2005: p. 251-254.

[13]. Ernst, D., N.S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge, *Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation.* ACM/IEEE International Symposium on Microarchitecture (MICRO), 2003.

[14]. Fei, Y., L. Zhong, and N.K. Jha, *An Energy-aware Framework for Coordinated Dynamic Software Management in Mobile Computers.* IEEE/ACM Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2004.

[15]. Flautner, K. and T. Mudge, *Vertigo: Automatic Performance-Setting for Linux.* Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), 2002.

[16]. Gochman, S. and R. Ronen, *The Intel Pentium M Processor: Microarchitecture and Performance.* Intel Technology Journal, 2003.

[17]. Gupta, A., B. Lin, and P.A. Dinda, *Measuring and Understanding User Comfort with Resource Borrowing.* Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004), 2004.

[18]. Intel, I., *Intel Itanium 2 Processor at 1.0 GHz and 900 MHz Datasheet*. July 2002.

[19]. John Wei. *Foxton Technology Pushes Processor Frequency, Application Performance*.

[20]. Lin, B. and P. Dinda, *Putting the user in direct Control of CPU Scheduling.* The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2006.

[21]. Lorch, J. and A. Smith, *Using User Interface Event Information in Dynamic Voltage Scaling Algorithms.* Technical Report UCB/CSD-02-1190, Computer Science Division, EECS, University of California at Berkeley, August 2002., 2002.

[22]. Lu, J., H. Chen, P.-C. Yew, and W.-C. Hsu, *Design and Implementation of a Lightweight Dynamic Optimization System.* Journal of Instruction-Level Parallelism, 2004. **6**.

[23]. Mallik, A., J. Cosgrove, R.P. Dick, G. Memik, and P. Dinda. *PICSEL: Measuring User-Perceived Performance to Control Dynamic Frequency Scaling*. in *the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-2008)* 2008.

[24]. Mallik, A., B. Lin, G. Memik, P. Dinda, and R.P. Dick, *User-Driven Frequency Scaling.* IEEE Computer Architecture Letters, 2006. **5**(2): p. 16.

[25]. Sasaki, H., Y. Ikeda, M. Kondo, and H. Nakamura. *An intra-task DVFS technique based on statistical analysis of hardware events* in *Proceedings of the 4th international conference on Computing frontiers* 2007

[26]. Wu, Q., V. Reddi, Y. Wu, J. Lee, D. Connors, D. Brooks, M. Martonosi, and D.W. Clark, *Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance.* 38th International Symposium on Microarchitecture (MICRO-38), 2005.

[27]. Xu, R., D. Moss, and R. Melhem, *Minimizing Expected Energy in Real-time Embedded Systems.* Proceedings of the 5th ACM international conference on Embedded software(EMSOFT), 2005: p. 251-254.

[28]. Yan, L., L. Zhong, and N.K. Jha, *User-perceived Latency based Dynamic Voltage Scaling for Interactive Applications.* Proceedings of ACM/IEEE Design Automation Conference (DAC '05), 2005.

Appendix A

Table 3 presents the correlation between 45 metrics based on hardware counter readings. Please see Section 4 on details of the calculation of these correlation factors.

Table 3. Correlation between the hardware performance counters and user satisfaction

Performance Metrics	Correlation	Performance Metrics	Correlation	Performance Metrics	Correlation
PAPI_BTAC_M-avg	0.771	PAPI_RES_STL-max	0.738	PAPI_TOT_INS-range	0.625
PAPI_L1_ICA-avg	0.770	PAPI_BTAC_M-max	0.733	PAPI_TOT_INS-min	0.603
PAPI_L1_ICA-stdev	0.770	PAPI_TOT_INS-max	0.729	PAPI_L1_DCA-min	0.528
PAPI_BTAC_M-stdev	0.770	PAPI_L2_TCM-avg	0.722	PAPI_L2_TCM-max	0.525
PAPI_L1_DCA-stdev	0.768	PAPI_L1_DCA-range	0.721	PAPI_BR_MSP-min	0.503
PAPI_TOT_INS-avg	0.768	PAPI_L2_TCM-stdev	0.709	PAPI_L2_TCM-range	0.497
PAPI_TOT_CYC-avg	0.767	PAPI_RES_STL-min	0.694	PAPI_L2_TCM-min	0.495
PAPI_L1_DCA-max	0.767	PAPI_TOT_CYC-min	0.689	PAPI_BR_MSP-max	0.379
PAPI_TOT_CYC-stdev	0.767	PAPI_RES_STL-range	0.684	PAPI_BR_MSP-range	0.360
PAPI_TOT_INS-stdev	0.766	PAPI_L1_ICA-min	0.682	PAPI_BTAC_M-min	0.289
PAPI_L1_DCA-avg	0.766	PAPI_L1_ICA-range	0.675	PAPI_HW_INT-max	0.131
PAPI_RES_STL-avg	0.761	PAPI_BR_MSP-avg	0.662	PAPI_HW_INT-range	0.119
PAPI_RES_STL-stdev	0.761	PAPI_BTAC_M-range	0.653	PAPI_HW_INT-min	0.112
PAPI_TOT_CYC-max	0.756	PAPI_TOT_CYC-range	0.644	PAPI_HW_INT-stdev	0.094
PAPI_L1_ICA-max	0.749	PAPI_BR_MSP-stdev	0.638	PAPI_HW_INT-avg	0.048