Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling

Akihiko Miyoshi[†]   Charles Lefurgy[‡]   Eric Van Hensbergen[‡]
Ram Rajamony[‡]   Raj Rajkumar[†]

[†]Real-Time and Multimedia Systems Lab
Dept. of Electrical and Computer Engineering
Carnegie Mellon University

[‡]Austin Research Laboratory
IBM

ABSTRACT

Energy efficiency is becoming an increasingly important feature for both mobile and high-performance server systems. Most processors designed today include power management features that provide processor operating points which can be used in power management algorithms. However, existing power management algorithms implicitly assume that lower performance points are more energy efficient than higher performance points. Our empirical observations indicate that for many systems, this assumption is not valid.

We introduce a new concept called *critical power slope* to explain and capture the power-performance characteristics of systems with power management features. We evaluate three systems - a clock throttled Pentium laptop, a frequency scaled PowerPC platform, and a voltage scaled system to demonstrate the benefits of our approach. Our evaluation is based on empirical measurements of the first two systems, and publicly available data for the third. Using critical power slope, we explain why on the Pentium-based system, it is energy efficient to run only at the highest frequency, while on the PowerPC-based system, it is energy efficient to run at the lowest frequency point. We confirm our results by measuring the behavior of a web serving benchmark. Furthermore, we extend the critical power slope concept to understand the benefits of voltage scaling when combined with frequency scaling. We show that in some cases, it may be energy efficient *not* to reduce voltage below a certain point.

Categories and Subject Descriptors

C.5.0 [**Computer System Implementation**]: General; C.4 [**Performance of Systems**]:

General Terms

Measurement, Performance, Design

Keywords

Energy aware computing

1. INTRODUCTION

Energy efficiency is becoming an increasingly important feature for both mobile systems [12] and server systems[2, 4]. Most processors designed today include power management features, such as frequency scaling [5] or dynamic frequency and voltage scaling [20]. These features provide a choice of processor *operating points*, which can be used in power management algorithms.

This paper presents a novel technique for determining the operating points of a processor that should *not* be considered by a power management algorithm. Specifically, the operating points identified by our technique will always lead to reduced energy efficiency. Knowing the domain of energy efficient operating points is important because existing power management algorithms [20, 9, 13, 18] implicitly assume that lower performance points are more energy efficient than higher performance points. As we show later, this assumption does not always hold, and there exist lower performance operating points that are not energy efficient.

Figures 1 and 2 illustrate the key concept of an energy-efficient operating point. Both figures depict the operation of a system when it executes the *same* load within the *same* period of time t at different operating points. In Figure 1, the system operates at a low performance–power mode for the entire duration and consumes energy $E_{active-low}$. In Figure 2, the system operates at a high performance–power mode and completes the job in a shorter amount of time. During the time that the processor is executing the workload, the system consumes energy $E_{active-high}$. After completing the job, the system goes into an idle state, where it consumes energy $E_{idle-high}$. The total energy consumed in this case is $E_{active-high} + E_{idle-high}$. For this system, the energy efficient operation point depends on the relative values of $E_{active-low}$ and $E_{active-high} + E_{idle-high}$.

One can look at this load as individual tasks in the context of real-time systems where t represents a period. In server systems, t corresponds to the time to respond to a request within the server. Customer Service Level Agreements or QoS requirements place a practical upper bound on t. Therefore it is appropriate to think of t as a soft-deadline in server systems as well[6, 13]. Since most power management techniques exploit the fact that the load im-
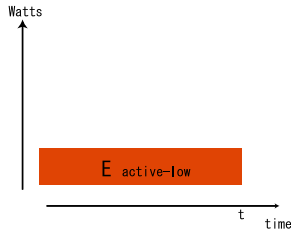
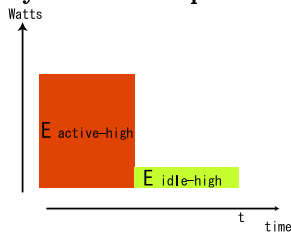**Figure 1: System at low performance point**

Figure 2: System at high performance point

posed on the system is low enough to run at a lower performance state without sacrificing the overall throughput, we make this assumption also.

Given a system with a set of operating points, the critical power slope can be used to ascertain which operating points must *not* be considered by a power management algorithm. Hence, our technique complements existing power management techniques, and can be used to determine an optimal frequency for a device with a fixed frequency microprocessor which is not able to scale the frequency and voltage dynamically.

Frequency scaling, clock throttling, and dynamic voltage scaling[1] are three processor power management techniques available on existing processors [7, 1, 10]. We evaluate three systems using the critical power slope methodology to determine the range of operating points that power management algorithms should consider. The first system is a Dell Inspiron 8000 laptop with a Pentium III processor whose performance could be clock throttled from 104 MHz to 850 MHz. The second system we consider is a single board computer with a PowerPC 405GP microprocessor that could be frequency scaled between 66 MHz and 266 MHz. The third system uses a dynamic voltage scaled StrongARM SA-1100 processor whose voltage and frequency can be varied from 59MHz/0.79V to 251MHz/1.65V. For the first two systems, we directly measured the base data necessary for calculating the critical power slope. We use data published elsewhere to determine the critical power slope for the third system. Our findings are counter–intuitive. On the Dell system, we found that the highest frequency point will *always* be the most energy efficient operating point. On the PowerPC system, we find that the lowest frequency point will be most energy efficient, presenting power management algorithms with the ability to trade off execution time for reduced energy usage.

---

[1]In this paper, we refer to the combination of voltage scaling and frequency scaling as *voltage scaling* and pure frequency scaling without modification to voltage, as *frequency scaling.*

On the DVS system, the published data leads us to conclude that operating the system in a voltage–scaled mode below 74 MHz will *not* be energy efficient from a system point of view.

Our findings are important because existing algorithms build upon processor power management techniques by *implicitly* assuming that operating at a lower performance point saves energy. Our findings show that this assumption is not always true. Furthermore, we present a technique that enables the inefficient operating points to be identified, enabling power management algorithms to correctly function by trading off performance for increased energy savings.

The rest of this paper is organized as follows. In section 3, we provide performance, power, and energy characteristics of a Pentium-based Linux system. We provide similar evaluation results for a PowerPC-based Linux system in section 4. Then in section 5, we provide a generalization of the two systems and introduce a concept we call *critical power slope*. Section 6 deals with performance and energy tradeoff issues in the context of voltage scaling systems. In section 7, we confirm our findings on a realistic workload. Finally, we discuss related work in section 8 and then conclude in section 9.

2. POWER MANAGEMENT

2.1 Frequency Scaling

Frequency scaling is a technique where the processor clock is reduced by some multiple of the maximum, permitting the processor to consume less power at the expense of reduced performance. Frequency scaled processors can have anywhere from two operating points to several operating points.

2.2 Clock Throttling

The Advanced Configuration and Power Interface specification(ACPI) [1] enables the use of *clock throttling* to dynamically modify the performance of an active processor. In contrast with frequency scaling where the frequency of the processor clock is actually modified, clock throttling keeps the clock running at the original frequency. However, the clock signal is *gated* or disabled for some number of cycles at regular intervals. This slows down the CPU since it receives fewer clock signals per unit of time.

2.3 Dynamic Voltage Scaling (DVS)

Dynamic voltage scaling (DVS) reduces the power consumed by a processor by lowering its operating voltage. A reduction in operating voltage generally also requires a proportional reduction in frequency [16, 17]. Voltage scaling is advantageous because the energy consumed by a processor is directly proportional to V^2, where V is the operating voltage. By varying the voltage along with the frequency, it is possible to obtain a quadratic reduction in power consumption. Transmeta's Crusoe processor[7] and Intel's Pentiums with SpeedStep[10] are examples of processors with DVS.

3. LINUX ON PENTIUM

As an example of a high performance system, we choose a laptop (Dell Inspiron 8000) with 850MHz Pentium III processor with 512MB of RAM running Linux-2.4.6-pre6. We used the clock throttling capabilities provided by ACPI to

state	clock unhalted (MHz)	slowdown
100.0 %	848.00	1.00
87.5 %	727.65	1.17
75.0 %	623.61	1.36
62.5 %	519.33	1.63
50.0 %	415.33	2.04
37.5 %	311.24	2.72
25.0 %	207.13	4.09
12.5 %	103.56	8.19

Table 1: Performance change due to clock throttling

change the processor performance states. On our Dell laptop, the processor runs at 8 different performance states. The eight states operate the processor at 100%, 87.5%, 75%, 62.5%, 50%, 37.5%, 25%, and 12.5% [2] of the maximum performance. When the system is idle, the CPU will go into sleep state and does not execute instructions.

3.1 Effects of Clock Throttling using ACPI

First, we evaluated the actual effect on the processor when throttling the clock. We observed the CPU slowdown by recording the *unhalted cycles* event on the Pentium performance counters. This returns the number of unhalted (ungated) cycles observed by the processor. As mentioned earlier, we are able to measure the actual clock cycles (original clock frequency) using the Pentium time stamp counter. By comparing the original clock frequency against the number of unhalted cycles, we get accurate information of the slowdown/speedup of the processor.

At 100% performance state, the clock is running at 848 MHz. When the performance state is set to 87.5%, the clock is unhalted 727.65 million times a second. This suggests that the processor slowed down by 1.17 times. The complete results for each performance state are shown in Table 1.

3.2 Pentium Micro Benchmark: Performance

To compare the actual performance with the clock slowdown observed in the previous section, we ran several benchmarks and measured the running times at different performance states. The micro benchmarks we used are:

- **Access to register:** Continuously copies the contents of a register to another register.

- **L1 cache (read):** Continuously reads bytes from a 1KB data structure that fits in the L1 cache.

- **L1 cache (write):** Continuously writes bytes into a 1KB data structure that fits in the L1 cache.

- **Access to memory (read):** Continuously reads bytes from a 32 MB memory region in a way so that each read causes a D-cache miss.

- **Access to memory (write):** Continuously writes a byte into a 32 MB memory region in a way so that each write causes a D-cache miss.

- **Disk read:** Continuously reads the contents of a new and different 1KB size file from disk.

---

[2]In our figures, we often denote the performance state with a integer number due to space limitations. (eg. 87.5 % is denoted as 87%)
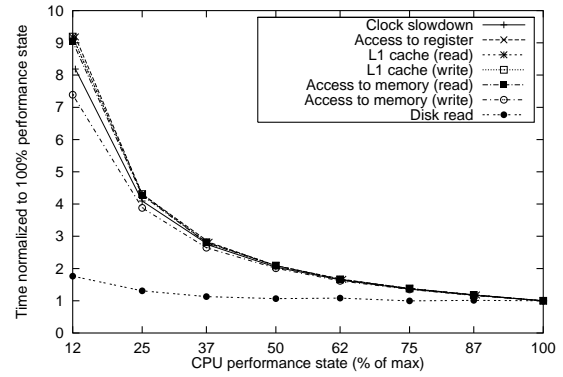
Figure 3: Micro benchmark performance (Pentium)

The results are shown in Figure 3. The CPU performance state is listed on the X axis, and the normalized slowdown of the benchmarks relative to the 100% performance state is shown on the Y axis. Clock slow down shows the *ideal* slowdown of the processor suggested by the measured clock frequencies reported in Table 1 (which is *clockspeed/clock unhalted*). We expected the benchmarks access to register, L1 cache read, and L1 cache write, to perform close to the ideal slowdown since they do not access memory. However, they are consistently above the ideal slowdown indicated by clock slowdown. For example at 12.5% performance state, all three benchmarks show a slowdown of 9.17 where as the clock slowdown is only 8.19. We expected a slow down of 8.19 for these benchmarks. Although we are not able to explain the cause of this phenomenon, we believe this is due to some architectural restriction on how the Pentium is implemented. (For example, an overhead incurred by phase lock loop). But aside from that, all three CPU intensive benchmarks show equal slowdown.

For access to memory (read), we start to see the mismatch between processor speed and memory speed showing an effect. At 12.5% performance state, the benchmark slowdown is slightly lower than 9.17. By lowering the performance of the processor, the *relative* memory speed increases. The processor wastes less cycles waiting for memory. Hence, one can say that the resources are utilized in a more efficient manner when the processor is running at a lower performance state. For access to memory (write) benchmark, we see a significant benefit compared to the ideal slow down. This is due to the asynchronous nature of writes using the write buffer. The processor need not stall waiting for the write buffer to empty. In the case of memory reads, the interaction between the memory and the processor is more synchronous. Hence, the performance benefits are not as large.

For the disk read benchmark, this trend becomes stronger. Even when the processor is throttled down to 12.5 % performance, the program slows down by less than 2 times. This indicates that the processor was underutilized when it was at a higher performance state, and there were opportunities to lower the processor speed without sacrificing user perceived performance. We expect the granularity of clock throttling to have an effect on how efficiently the memory or the disk is utilized. For example, if the clock is gated for 1K cycles every 8K cycles rather than every 1 cycle every 8 cycles,

CPU performance state	C1 (Watts)	C2 (Watts)
100.0 %	12.03	12.05
87.5 %	12.10	12.02
75.0 %	12.06	11.99
62.5 %	12.04	11.94
50.0 %	12.02	11.93
37.5 %	12.01	11.90
25.0 %	12.00	11.89
12.5 %	11.99	11.97

Table 2: Power usage in idle mode (Pentium)

the effect of clock throttling on performance should not be as strong. Another conclusion one can draw from these results is that for CPU intensive benchmarks, the runtime of the benchmark and the frequency of the processor has an approximate inverse relationship.

3.3 Pentium: Power Measurements

We determined the energy consumed by the system by measuring the drawn voltage and current. While we could directly measure voltages, we used a sense resistor in series with each system to measure the current. Signals from the sense resistors were filtered with a 10kHz low pass filter, gained to be within $\pm 10V$ using custom circuitry, and then passed to a PCI-6071E A-to-D board from National Instruments. Each channel was sampled at 1,000 times per second, and custom software was written to gather and process the data. We estimate the accuracy of our facility to be within +-2 %, with sense resistor accuracy and amplifier voltage offset being the dominant source of errors. For our Pentium-based laptop, we were only able to measure the *total* system power consumption. That is, wall power drawn by the laptop as a whole. In the numbers we report, power used by components such as CPU, memory, network, and disks are all included. The LCD was turned off.

Table 2 shows the system power consumption while the operating system is in idle mode. The Linux scheduler puts the processor into C1 or C2 sleep state defined by ACPI and waits until there is useful work to do. The idle system uses around 12 Watts in either sleep states. Idle power consumed in each performance state decreases as the performance is lowered but not significantly. Therefore, we can consider the idle state power to be a constant.

Next, we ran a simple benchmark (continuously incrementing a register) that exercises the CPU and changed the performance state dynamically from 100% to 12.5% while the program is running. Figure 4 shows power at the Y axis and the performance state at the X axis. We lowered the CPU performance state every 5 seconds, starting from 100%. Each second, 200 samples of the power used by the system is recorded. When the CPU is running at maximum performance, the system uses approximately 30 Watts. As the performance state is lowered, power decreases. At the lowest performance state (12.5%), the system uses around 16 Watts on average.

From these results, we see that clock throttling has the effect of bounding peak power consumption as well as average power consumption. We also see that power consumed by the system has a linear relationship with the performance state of the processor. As the performance of the processor is lowered, system power usage decreases linearly.
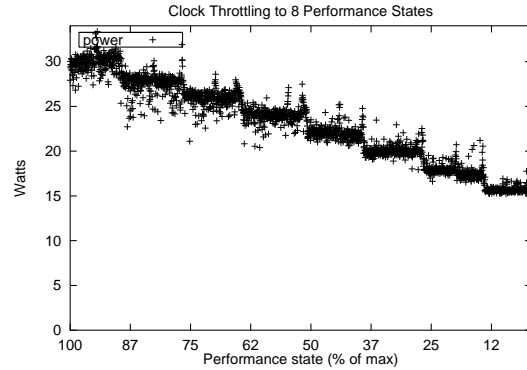

Figure 4: Power consumption at different performance states

3.4 Pentium Micro Benchmark: Energy consumption

To understand the effect of clock throttling on energy, we revisit the micro benchmarks used in section 3.2. As shown in Figures 1 and 2, we classify energy usage into two parts. Energy required to complete the benchmark: E_{active}, and energy required to be in idle state: E_{idle}. Our goal is to compare the energy used to execute the *same* load at different operating points during the *same* time interval. The time interval does not end at E_{active} because the server may not be simply turned off after responding to a request. It must be kept on since the arrival time of the next request is unknown.

The amount of energy consumed (E_{active}) while running the benchmark is measured. Simultaneously, the time to run the benchmark is also recorded. We compare this time against the time the benchmark takes at the lowest performance state. The difference becomes the *extra* time the system is in idle state. Since we know how much power is used in the idle state, we compute the amount of energy consumed while in this extra idle time (E_{idle}). Figures 5, 6, and 7 illustrates the results [3]. In these Figures, the amount of extra idle time (E_{idle}) is put on top of energy required to run the benchmark (E_{active}). The sum of these two numbers show us whether it is energy efficient to run them at a certain performance state.

To run L1 cache read benchmark (Figure 5) at the lowest performance state, it takes 174.3 seconds and uses 2591.5 Joules. At 100% performance state, the system uses 621.5 Joules to run the benchmark in 18.99 seconds. Then the system will be in extra idle state until 174.3 seconds. During this extra idle state, the system at 100% performance state consumes 1868.7 Joules. This means that the system consumes 2490.2 Joules (1868.7 + 621.5) at 100% to serve the same benchmark which is 101 Joules less than running the same benchmark at the lowest performance.

All our results show that the total system energy required to run the benchmark (E_{active}) decreases as the performance state is raised. This is because the benchmark takes shorter time to complete. But on the other hand, the extra en-

---

[3]We omit the results from other benchmarks due to space limitations but they show results consistent with our argument.
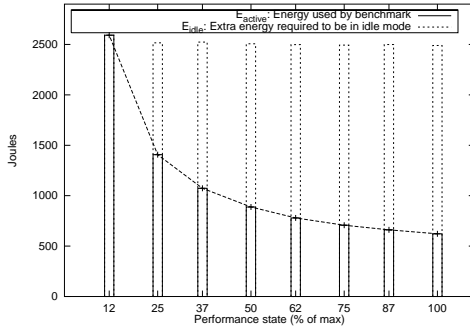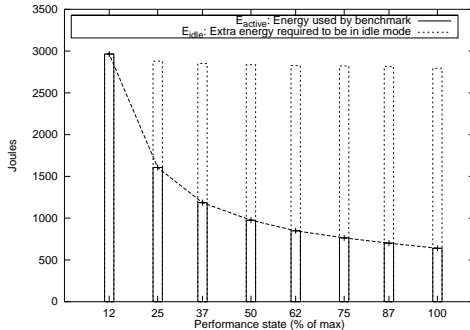
Figure 5: L1 cache (read)

Figure 6: Memory (read)

Figure 7: Disk read

Figure 8: Micro benchmark performance (PPC)

ergy required to be in the idle state after the benchmark completes (E_{idle}) increases. The sum of E_{active} and E_{idle} decreases slightly as the performance state is increased. This means that if we impose the same load on the system at different performance states, the system at higher performance state uses less energy.

For benchmarks that go to memory (writes) and disk, actual energy required to run the benchmark (E_{active}) does not decrease as much in high performance states compared to other benchmarks. This is because the memory or the disk becomes the bottleneck, and the runtime decreases less. From a performance perspective, the reduction of CPU stalls due to lower processor states lead to a smaller slowdown. At higher performance states, performance does not increase equally with the increased clock rate due to the processor waiting for memory and disks. From an energy perspective, at higher performance states, energy is spent for the CPU stalling while running the benchmark. Since it takes longer to complete the benchmark compared to CPU intensive ones, there is less time and energy spent in extra idle state. The two effects cancel each other out, and in our measurements, the overall results did not change. We expect this result may change if the power consumed while stalling for memory or disk was substantially different from the power required to be in idle state. In summary, when we consider the total system energy ($E_{active} + E_{idle}$), the benchmarks suggest that we should run this system at the highest performance state possible.

4. LINUX ON POWERPC

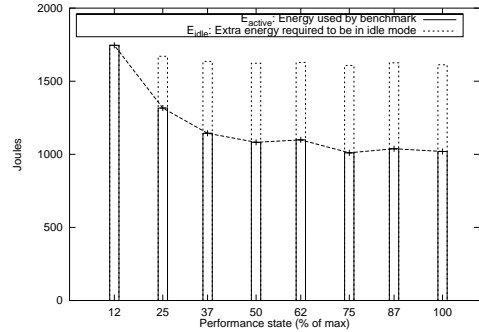Next, we analyze a different system, which is a single

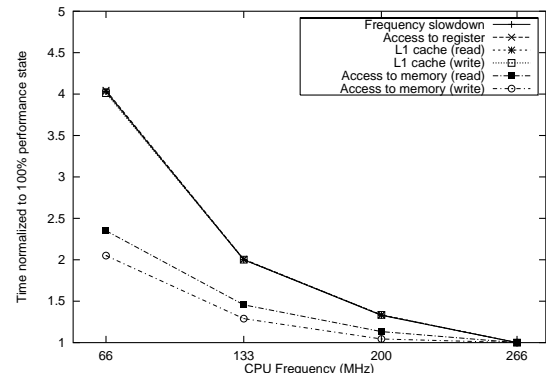board computer based on PowerPC 405GP microprocessor. The processor has 8KB of D-cache and 16KB of I-cache. The board has 32MB of memory and by modifying the strapping on the board, we can change the frequency of the processor and the processor local bus (PLB) that directly affects memory speed. We used Linux 2.4.0 as our operating system. On this system, we are able to change the actual clock frequency. For example, when the processor local bus (PLB) is set to 66MHz, the clock frequency of the processor can be set to 66, 133, 200, and 266 MHz. In this environment, we are able to measure the power usage of each of the separate components in the system. We report measurements of the memory and the CPU as well as the total system energy.

4.1 PowerPC Micro Benchmark: Performance

Figure 8 reports the performance of the same micro benchmarks used in Section 3.2. Since the board does not have a hard disk, we did not perform the disk read micro benchmark. Frequency slowdown indicates the slowdown of the clock relative to the maximum frequency of 266MHz. For the benchmark which accesses the register or the L1 cache (Access to register, L1 cache read, write), we see that the slowdown overlaps with the slowdown indicated by the clock frequency (Frequency slowdown). When the benchmark accesses memory, the slowdown becomes less than what the clock frequency indicates. This is due to the reduction of CPU stalls waiting for memory when CPU becomes slower as discussed previously.

Frequency	CPU (Watts)	Mem (Watts)	Total (Watts)
66 MHz	0.51	0.17	2.02
133 MHz	0.59	0.17	2.10
200 MHz	0.65	0.17	2.17
266 MHz	0.69	0.17	2.21

Table 3: Power usage in idle mode (PowerPC)

Frequency	CPU (Watts)	Mem (Watts)	Total (Watts)
66 MHz	0.74	0.17	2.27
133 MHz	1.09	0.18	2.63
200 MHz	1.36	0.18	2.89
266 MHz	1.58	0.18	3.13

Table 4: Active state power for PowerPC system

4.2 PowerPC: Power Measurements

Table 3 shows the system power consumption while the operating system is in idle mode. Unlike clock throttling on the Pentium system, we see that there is a difference in power usage while in idle mode at different frequencies. In the PowerPC implementation, the wait enable (WE) bit is set in the processor status register to prevent the processor from fetching instructions. Although the processor is not executing instructions, the clock is still operating at the specified frequency. The power consumption of the processor itself increases slightly as the frequency increases, but it is a small increase when compared to the total system power.

Table 4 shows the total active system power usage at each frequency when running the simple benchmark. As with the Pentium-based system, power increases as the frequency is raised.

4.3 PowerPC Micro Benchmark: Energy consumption

Figures 9, and 10 shows the energy usage of two micro benchmarks. (We omit the results of other benchmarks due to space limitations). As with the results shown in the previous section using the Pentium-based system, the energy to actually complete the benchmark is denoted as E_{active} and extra idle time as E_{idle}. Further, since we are able to measure the power usage of each component, we separate E_{active} into several components. CPU, SDRAM, and all other components of the board (which includes a PCI bridge and a network card).

In all our benchmarks we ran, our results showed that by lowering the frequency, the total energy used by the system including the extra idle time energy (E_{active} and E_{idle}) decreases. For example, when running the L1 cache (read) benchmark, the system consumes 162.07 Joules at 266 MHz, while it takes 150.53 Joules at 66MHz. Hence, on this system, by lowering the frequency to a point where the workloads can be adequately served without sacrificing latency, energy is saved. This result is opposite from the results we obtained from our Pentium-based system.

5. CHARACTERIZATION OF TWO SYSTEMS

There is a tradeoff one must make when looking at both performance and power. Should the processor run slower and reduce active state power, but take a longer time to
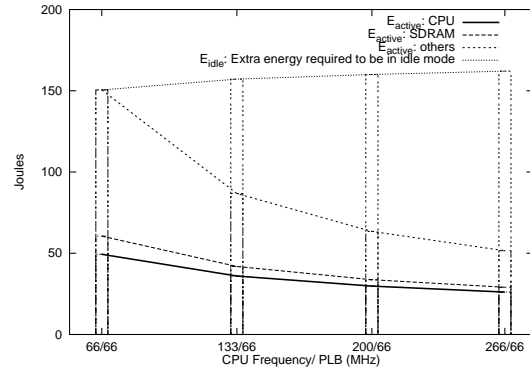

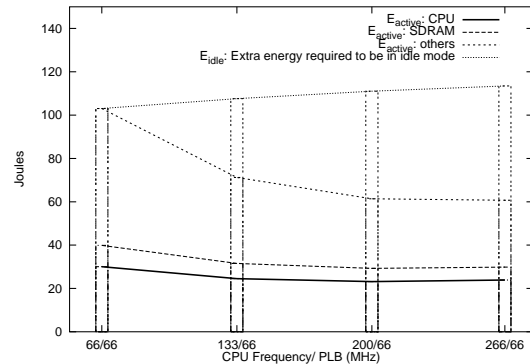**Figure 9: L1 cache (read): PowerPC**

Figure 10: Access to memory (write): PowerPC

complete a job which in turn reduces the time the system is in idle state, *or* should the processor run faster and move into the idle state more frequently? Our earlier measurements show the choice is unclear. It was energy efficient to run the same workload at a higher performance state on our Pentium-based system, and at a lower frequency on the PowerPC-based system. Here, we identify the qualities of the system one must understand to make this tradeoff decision.

Based on our measurement shown previously, we assume that the following characteristics should generally hold for systems with frequency scaling processors:

- Power usage of a system shows bimodal behavior. The system will either be in *active* state or *idle* state[2].

- Performance and frequency (processor performance state) has a linear relationship.

- Active state power increases linearly with frequency of the processor. We denote the active state power as a function of frequency: P_{freq}.

- Idle state power P_{idle} will be considered approximately constant for all frequencies.

Now, lets look at a CPU intensive workload **W**, and assume the lowest frequency that meets the deadline requirements of workload **W** is f_{min}. To keep our discussions simple, we assume that the utilization of the system at frequency f_{min} is 1, and **W** takes $T_{f_{min}}$ units of time to com-
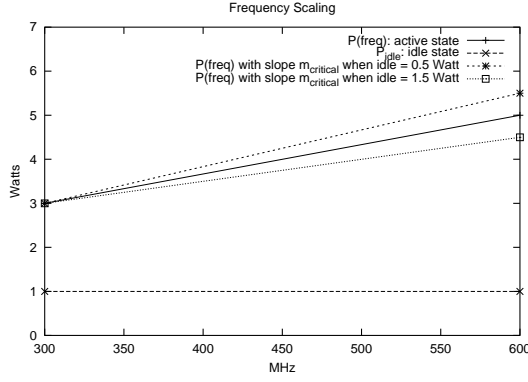
Figure 11: Frequency scaling system

plete. This implies the system is in active state (and consuming active state power $P_{f_{min}}$) for the duration of $T_{f_{min}}$. From this, we know that the energy $E_{f_{min}}$ consumed by the system while processing **W** is:

$$E_{f_{min}} = T_{f_{min}} \ P_{f_{min}}$$

At a higher frequency f, we look at a system with the same workload **W** imposed for the same duration of time $T_{f_{min}}$. At frequency f, the utilization of the system decreases to $\frac{f_{min}}{f}$ (since for CPU intensive workloads, we expect the performance of the system to be inversely proportional to the frequency). That means that the system is active $T_{f_{min}}$ $\frac{f_{min}}{f}$ units of time and idle $T_{f_{min}} \left(1 - \frac{f_{min}}{f}\right)$ units of time. We can thus compute the energy consumed while processing **W** at frequency f as:

$$E_f = (T_{f_{min}} \frac{f_{min}}{f})P_f + T_{f_{min}} (1 - \frac{f_{min}}{f})P_{idle}$$

5.1 Critical Power Slope

As we recall, a generic system would show power characteristics such as the one shown in Figure 11. Power increases linearly with the frequency of the processor, and constant at idle state. Both our Pentium and PowerPC systems share these properties. We denote the slope of the line representing active state power in Figure 11 as m. Then,

$$P_f = P_{f_{min}} + m(f - f_{min})$$

If we substitute P_f, E_f becomes

$$E_f = (T_{f_{min}} \frac{f_{min}}{f})[P_{f_{min}} + m(f - f_{min})] + (T_{f_{min}} - T_{f_{min}} \frac{f_{min}}{f})P_{idle}$$

Now we can calculate the energy required to complete **W** at various frequencies. By comparing the energy used in various frequencies, we would know if it is energy efficient to lower the frequency or not. Then, we see that the slope m determines the resulting decision, and there should be a slope where energy usage at all frequencies is equal. We will call this slope, *critical power slope* and denote as $m_{critical}$.

To derive this slope, we find slope m such that E_f is constant for all frequencies by solving for m when $E_{f_{min}} = E_f$. Then the critical power slope becomes:

$$m_{critical} = \frac{P_{f_{min}} - P_{idle}}{f_{min}}$$

From this equation, we see that the critical power slope is determined by the active state power $P_{f_{min}}$ and idle state power P_{idle}. These two attributes are influenced by various aspects such as configuration of the system. From a systems perspective, the implication $m_{critical}$ has is as follows:

- if actual slope m of the hardware used is less than the critical power slope $m_{critical}$, it is energy efficient to run the system at higher frequency. That is, it is energy efficient to minimize the time in active state and maximize idle time of system.

- if actual slope m of the hardware is greater than $m_{critical}$, the system should be run at slower frequency to save energy.

- the more optimized the idle state is, the more favorable it is to run at a higher frequency.

For the system depicted in Figure 11, the critical power slope when the idle state power is 1 Watt is:

$$\frac{5Watts - 1Watts}{600MHz - 300MHz} = .013$$

It turns out that the active state power line equals the critical power slope. This means that for CPU intensive workloads, one should end up with the same energy regardless of the frequency. If the idle state energy becomes smaller than 1 Watt, critical power slope will become higher than the active power line as shown in the Figure 11 which favors running the workload at a higher frequency. (Since idle state power is cheap, system should go into idle state as long as possible). On the other hand, when idle state power becomes higher than 1 Watt, critical power slope becomes smaller than the active power line.

If we look at the critical power slope for our Pentium-based system, it is:

$$\frac{15Watts - 12Watts}{848MHz * 12.5\%} = .028$$

Since the actual slope of a particular system can be calculated by,

$$m = \frac{P_f - P_{f_{min}}}{f - f_{min}}$$

the actual slope m for the Pentium-based system becomes

$$\frac{30Watts - 15Watts}{848MHz - (848MHz * 12.5\%)} = .020$$

which is smaller than the critical power slope. Hence, it is energy efficient to run at the highest performance state. For our PowerPC-based system, the critical power slope is:

$$\frac{2.27Watts - 2.02Watts}{66MHz} = .0038$$

where as the actual slope is:

$$\frac{3.13Watts - 2.27Watts}{266MHz - 66MHz} = .0043$$

Critical power slope is smaller then the actual slope on the PowerPC. This resulted in a tradeoff where it is energy efficient to reduce frequency which is the opposite from the Pentium case. Of course, the amount of reduction in frequency is subject to other constraints such as the desired performance, QoS level, or real-time requirements. Another observation we can make from this is that, if we are able to optimize the idle state power consumption of our PowerPC-based system to below 1.77 Watts (for example, going into deeper processor sleep modes, or shutting down parts of the

system like the network, or memory, which makes the critical slope smaller), then running at the highest frequency is more energy efficient than running at the lowest frequency.

As we can see, critical power slope captures the runtime tradeoff relationship between the active state energy and idle state and can be considered a metric which indicates the efficiency of idle state relative to active states.

6. CRITICAL POWER SLOPE IN SYSTEMS WITH VOLTAGE SCALING PROCESSORS

In the previous section, we assumed that the active power was linear with frequency. Critical power slope can be used to understand the runtime performance tradeoff of systems where this assumption does not hold. Voltage scaling processors show a nonlinear active power behavior as illustrated in Figure 12[4]. Unlike systems with frequency scaling processors, the combination of voltage scaling and frequency scaling yields a nonlinear savings of power (due to P $\propto V^2$ relationship). This means that the slope of the active state power is not constant, and it decreases as performance is lowered.

In the case where we assumed that the active power slope was linear, we only needed to look at one reference operating point at f_{min} to make the tradeoff decision. Since active power is linear with frequency, the relative relationship between active power slope and the critical power slope (whether one is larger than the other) is the same at all operating points.

For voltage scaling systems, we need to look at every operating point since the slopes are not constant. That is, for every operating point at frequency f_x, we look at the active power P_x, and derive the critical power slope for frequency f_x as:

$$m^{f_x}_{critical} = \frac{P_{f_x} - P_{idle}}{f_x}$$

Then, we compare it against the actual power slope m^{f_x} at operating point with frequency f_x. From this, we know that E_{f_x} which is energy consumed at frequency f_x has the following property to run some workload **W**:

- if $m^{f_x} < m^{f_x}_{critical}$, then $E_{f_x-\epsilon} > E_{f_x} > E_{f_x+\epsilon}$

- if $m^{f_x} > m^{f_x}_{critical}$, then $E_{f_x-\epsilon} < E_{f_x} < E_{f_x+\epsilon}$

where ϵ is a small number. If we look at a system with voltage scaling processors such as the one shown in Figure 12 and compare the critical power slope and the actual slope, we obtain the following results:

$m^{600MHz}_{critical} = .0083 \quad < \quad m^{600MHz} = .0240$
$m^{525MHz}_{critical} = .0061 \quad < \quad m^{525MHz} = .0187$
$m^{450MHz}_{critical} = .0040 \quad < \quad m^{450MHz} = .0107$
$m^{375MHz}_{critical} = .0027 \quad < \quad m^{375MHz} = .0040$
$m^{300MHz}_{critical} = .0023 \quad < \quad m^{300MHz} = .0027$
$m^{225MHz}_{critical} = .0022 \quad > \quad m^{225MHz} = .0007$
$m^{150MHz}_{critical} = .0030 \quad > \quad m^{150MHz} = .0004$
$m^{75MHz}_{critical} = .0056 \quad > \quad m^{75MHz} = .0003$

We see that at higher frequencies, $m^{f_x} > m^{f_x}_{critical}$ and hence $E_{f_x-\epsilon} < E_{f_x} < E_{f_x+\epsilon}$. That means it is energy efficient to lower the frequency. As we lower the frequency,

---
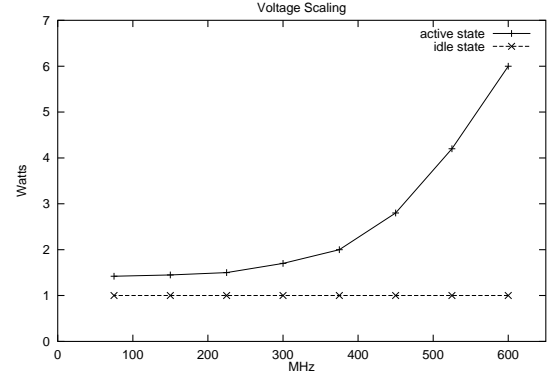
[4]Active state power based on Transmeta document[7]

Figure 12: Voltage scaling system

there will be a point where $m^{f_x}_{critical}$ becomes *larger* than m^{f_x}. This occurs somewhere between 225MHz and 300MHz. For the purpose of our discussion, lets assume that point was 290MHz. This means that it is no longer beneficial to further reduce voltage below a point at which it is capable of handling 290MHz, since $E_{290MHz} < E_{225MHz}$. Going into idle state becomes cheaper than running longer at lower performance below 290MHz. That is, the system would be energy efficient to run at 290MHz and go into idle state, rather than reducing voltage and running at 225MHz and going into idle state less frequently.

Further energy reduction after voltage scaling becomes ineffective may be achieved by performing clock throttling (and maintaining the voltage) below 290MHz **iff** the slope of active state power line using clock throttling is greater than the critical power slope. If the slope for active state power using clock throttling is smaller than the critical power slope, there is no incentive to reduce frequency lower than 290MHz. Techniques such as clock throttling can theoretically reduce active state power down to a level close to idle state power even after slope of active state power flattens out for voltage scaling. This means that the slope for clock throttling can potentially be steeper than the slope for active state power line using voltage scaling. This is indeed the case illustrated in [7].

Pouwelse [19] shows power measurement data very similar to Figure 12 for a system based on a StrongARM processor (SA-1100). Just like Figure 12, the active power slope for their system flattens out as voltage and frequency is lowered. Idle state power which is not strictly constant, shows a slight increase as the frequency is raised. Their experience with voltage scaling shows that the SA-1100 processor was able to operate from 59MHz at 0.79V up to 251MHz at 1.65V. We estimate from their data, that the critical power slope for their system becomes larger than the actual power slope at around 74MHz. From the figure provided, we approximate the critical power slope at 74MHz as:

$$m^{74MHz}_{critical} = \frac{121mW - 46mW}{74MHz} = 0.001$$

and the actual power slope as:

$$m^{74Hz} = \frac{121mW - 106mW}{74MHz - 59MHz} = 0.001$$

This means that, although their system can support voltage scaling from 251MHz to 59MHz, voltage scaling most
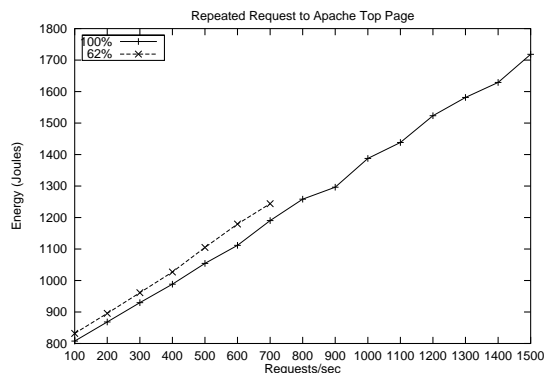
Figure 13: Repeated request to 3KB page

likely becomes inefficient starting at 74MHz. Thus, there is no incentive to operate at the range between 74MHz and 59MHz using voltage scaling.

Further, there is a chance that frequency scaling can provide more energy savings below 74MHz on the SA-1100-based system. The active power slope for frequency scaling when the voltage is kept constant at 1.5V was approximately 0.0029 which is greater than the critical power slope at 74MHz. We are uncertain at what voltage the system was able to handle 74MHz. But if the slope for that voltage is greater than the critical power slope at 74MHz, then the most energy efficient policy on the SA-1100-based system would be to employ voltage scaling until around 74MHz and switch to pure frequency scaling down to 59MHz.

Critical power slope can also be used to determine an optimal frequency for an embedded device with a fixed frequency microprocessor which is not able to scale the frequency/voltage dynamically. Designers can measure the active power slope, idle state power, and calculate the critical power slope. With this information, one can decide the optimal operating point for that particular device.

7. CRITICAL POWER SLOPE IN A REALISTIC WORKLOAD

To confirm our findings on a more realistic workload, we measure a static page request on a web server. Static page requests constitute an significant portion of real web server workloads. We used Apache-1.3.12-25 as our web server. The Pentium-based laptop was connected directly to a switch via IBM EtherJet 32bit cardbus 100Mb network card. The client is also directly connected to the switch with a 1 Gb connection. To generate requests, we used a publicly available web request generator called httperf[14].

By first observing the responses from our server, we concluded that our server at 100 % performance was able to handle up to 1500 requests/sec to requests to the top page of the server. The top page of the server was approximately 3KB in size, and in all our experiments, the page was cached in memory.

Figure 13 show results of energy measurements of the servers at two processor performance states, at 100% and 62.5% handling HTTP requests for 60 seconds. We measured the energy used by the server at different requests rates. At 100% performance state, the server handles up to

1500 requests/second. At 62%, the server is able to handle more than 700 requests/sec. We have confirmed this fact by looking at the number of requests handled in the 60 second period. Both the 62% and 100% performance state servers handled the same number of requests in that 60 second period, meaning that there was not a significant increase in response latency.

We see that the energy used by the servers increase linearly as the request rate is increased. Also, the result clearly indicates that on this particular Pentium-based system, it is *energy efficient* to run at a higher performance state, rather than to run them at a lower speed even though it may provide equal user perceived performance. With 100 requests/sec request rate, the server at 100% performance consumed 807.6 Joules. The server at 62% performance state used 832.6 Joules. A 25 Joule difference in 60 seconds. At 700 requests/sec, 100% server used 1190.4 Joules and the 62% server used 1243.9 Joules. A 53.5 Joule difference in 60 seconds. The same trend held true for other experiments with HTTP requests to pages of different sizes (eg. 1K and 10K) and different performance states (comparing them only if they can handle the same workload).

As shown in Section 5.1, critical power slope indicated that on our Pentium based system, it is energy efficient to run at the highest performance state. The results based on a realistic workload also confirm our findings.

8. RELATED WORK

Many newer processors have the capability to provide some power management features including Strong ARM SA-1100[19], and Transmeta's Crusoe[7] which provides frequency scaling and voltage scaling. Intel SpeedStep[10] technology offers two operating points each at different voltages. For example, 1GHz at 1.70V which consumes a maximum of 34.0 Watts, and 700MHz at 1.35V which consumes a maximum of 16.1 Watts. If we were to look only at the processor, the active power slope that the two operating points draw is much higher than the critical power slope for the processor. Hence, the runtime tradeoffs between active state power and idle state power strongly favors going into the lower operating point using SpeedStep whenever possible. It is unclear at this point how much limitation in energy savings there is by having only two operating points.

There has been several reports on the effects of frequency scaling. Farkas [5] quantified the energy consumption of a pocket computer running Java applications. Pouwelse [19] provides detailed analysis on the impact of dynamic voltage scaling looking at CPU and memory power consumptions.

There are various works on reducing system energy consumption. Power aware memory systems have been proposed in [11]. Dynamic voltage scaling (DVS) algorithms address the issue by reducing the voltage and the frequency of the CPU [20][9][15]. The basic approach is to reduce the performance of the processors by modifying the frequency and voltage while still providing adequate performance. Dynamic voltage scaling algorithms decide how much to scale the frequency/voltage and when. Often it predicts future requirements and decides the scaling factor by observing previous resource usage or utilization. PACE[13] provides performance equivalent optimization to DVS algorithms. We believe that our work is complementary or can be used to extend these works. For example, we capture the cost of the system being idle which is often neglected. We have shown

through actual measurement data, that idle cost can be very expensive, and hence, the basic premise of DVS algorithms that lower voltage/frequency uses less energy, may not always be true. Through the use of critical power slopes, we are able to inform these DVS algorithms the range of effective and ineffective operating points.

Compiler assistance [15] can be deployed in combination with DVS to provide system with useful hints to make informed decision. Real-Time DVS (RT-DVS) algorithms[18] incorporates real-time scheduling algorithms with DVS algorithms. This satisfies real-time guarantees as well as lowering the energy usage. Another approach is to adapt the application [8] to the environment. Application can be programmed in a way (by changing the semantics or fidelity) to conserve energy when required.

Burd[3] proposes a similar model from the point of CMOS microprocessor design in their overall power analysis methodology. Critical power slope agrees with their model, and we illustrate this using actual measurement data. Further, we used our model to define energy-inefficient operating points which can be used by the operating system.

9. CONCLUSION

Frequency scaling and voltage scaling have been proposed as promising techniques to reduce the energy usage by exploiting the fact that systems do not need to run at peak performance all the time. In this paper, we analyzed the runtime effects of frequency scaling on performance, power, and energy.

We used two very different platforms running Linux. One is a high performance PC based on a Pentium III 850MHz processor. The other platform is targeted as an embedded device with PowerPC 405GP processor. Our results show that on our Pentium-based system, it is energy efficient to run at the highest performance state, and on the PowerPC-based system, at a lower frequency when voltage is kept constant.

We explain the reason for the conflicting results by introducing a concept we call *critical power slope*. Critical power slope represents the runtime tradeoffs between active state power and idle state power. It tells us whether or not it is energy efficient to run at a higher performance state and complete the work and go into idle state. Hence, critical power slope can be used as a metric to understand how efficient the system is in regards to idle state relative to active state.

Further, we extend our analysis to voltage scaling systems. By looking at the critical power slope of several systems, we address the limitation of voltage scaling at lower performance states, and the basic premise that lower voltage/frequency uses less energy may not always be true. We argue for the possibility that a combination of voltage scaling and frequency scaling (while keeping the voltage constant) more energy efficient than pure voltage scaling based approaches.

10. ACKNOWLEDGEMENT

11. REFERENCES

[1] *Advanced Configuration and Power Interface Specification*, 2001. http://www.teleport.com/ acpi/spec.htm.

[2] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The case for power management in web servers. *Power Aware Computing*, 2002. Klewer Academic Publishers.

[3] T. D. Burd and R. W. Brodersen. Energy efficient CMOS microprocessor design. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, Jan. 1995.

[4] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[5] K. Farkas, J. Flinn, G. Back, D. Grunwald, and J.-A. Anderson. Quantifying the energy consumption of a pocket computer and a java virtual machine. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS)*, June 2000.

[6] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance-setting for dynamic voltage scaling. In *Proceedings of the International Conference on Mobile Computing and Networking (MOBICOM)*, 2001.

[7] M. Fleischmann. Dynamic Power Management for Crusoe Processors, Jan. 2001. http://www.transmeta.com/.

[8] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Dec. 1999.

[9] D. Grunwald, P. Levis, K. Farkas, C. Morrey, and M. Neufeld. Policies for dynamic clock scheduling. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, Oct. 2000.

[10] Intel. Mobile Intel Pentium III Processor in BGA2 and MicroPGA2 Packages, 2001. Order Number 283653-002.

[11] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis. Power aware page allocation. In *Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, Nov. 2000.

[12] J. Lorch and A. Smith. Energy consumption of Apple Macintosh computers. *IEEE Micro*, 18(6), Nov. 1998.

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

[14] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *Proceedings of the First Workshop on Internet Server Performance*, June 1998.

[15] D. Mosse, H. Aydin, B. Childers, and R. Melhem. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Proceedings of Workshop on Compiler and OS for Low Power (COLP)*, Oct. 2000.

[16] T. Mudge. Power: a first class design constraint. *Computer*, 34(4):52–57, Apr. 2001.

[17] T. Pering, T. Burd, and R. Brodersen. Dynamic voltage scaling and the design of a low-power microprocessor system. In *Power Driven Microarchitecture Workshop*, June 1998.

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

[19] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor . In *Proceedings of 7th International Conference on Mobile Computing and Networking (Mobicom)*, July 2001.

[20] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, Nov. 1994.