RC23425 Revised (W0507-140) July 19, 2005
Computer Science

IBM Research Report

Scheduling Processor Voltage and Frequency in Server and Cluster Systems

Ramakrishna Kotla
Department of Computer Science
University of Texas at Austin
Austin, TX

Soraya Ghiasi, Tom Keller, Freeman Rawson
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Scheduling Processor Voltage and Frequency in Server and Cluster Systems

Ramakrishna Kotla,
Department of Computer Science
University of Texas at Austin
kotla@cs.utexas.edu

Soraya Ghiasi, Tom Keller and Freeman Rawson
IBM Austin Research Laboratory
{sghiasi,tkeller,frawson}@us.ibm.com

Abstract

Modern server farm and cluster sites consume large quantities of energy both to power and cool the machines in the site. At the same time, less power supply redundancy is offered and power companies and government officials are requesting power consumption be reduced during certain time periods. These trends lead to the requirement of responding to rapid reductions in the maximum power the site may consume. Each possible solution must respond to the new power budget before a cascading failure occurs. Available techniques include powering down some nodes or slowing all nodes in a system uniformly. This work instead examines the feasibility of slowing nodes non-uniformly in response to their performance demands. This approach provides an opportunity to reduce the performance loss caused by a reduction in the power budget.

This paper uses the execution characteristics of the work currently running on each processor of the system or cluster to predict the performance of the work at the available frequency settings. The scheduling mechanism selects the lowest frequency for the processor that provides essentially all of the available performance of the work. It ensures that the selected frequency fits within the available global power budget and, if not, reduces it so that it does. The paper demonstrates the approach using a simple, synthetic benchmark and then validates it using additional, real-world applications.

1. Introduction

System-level and processor power is now a first-order design constraint across all classes of computing devices. While the primary problem for embedded and laptop computers is battery life and, thus, total energy consumption over time, the most important problem for servers and server clusters is maximum power [1]. Server computing environments have limitations on their internal power-delivery and cooling systems as well as installation limits on the total power and cooling available in the external environment. Prior work has led to the development of processor frequency and voltage scaling as a way to reduce processor power, which is often the single most important contributor to system-level power consumption. In the past, dynamic voltage and frequency scaling have found their primary application in embedded and laptop machines. This paper considers how to use them to control maximum power dissipation in SMP servers and server clusters.

Earlier work [2] demonstrates that workloads vary in their level of memory intensity, both between different workloads and within a workload's execution lifetime. Given that secondary cache and memory speed are unaffected by processor frequency scaling, memory-intensive workloads exhibit performance saturation at a characteristic frequency related to their level of memory intensity. Raising the frequency above the saturation point yields no further benefit in performance. The previous work considered only a single application at a time and used post-processing to determine the most appropriate frequency for each job.

This research considers how to identify and use an appropriate frequency, and corresponding voltage, for the aggregate workload on each processor. There are four main advantages to scheduling frequencies and voltages rather than work. First, it avoids the overhead of moving work from one processor to another. Second, it overcomes the difficulty generally experienced in cluster environments that work migration is difficult or impossible. Third, it allows systems vendors to implement necessary power control mechanisms without requiring changes to the operating system's scheduler. This is crucial in many environments since the vendor either does not have the source for the operating system or encounters difficulties and delays in getting the necessary changes accepted by a larger community. Finally, scheduling frequencies and voltages to processors as proposed in this paper does not depend on a detailed analysis of program phases and execution sequences. Instead, it relies on data from the performance counters on all processors in the system or cluster. While this sacrifices some accuracy, its simplicity makes it attractive, especially when the primary goal is to ensure that power remains under some maximum.

The research presented here makes a number of contributions beyond the prior work in the area.

- It uses the known phenomena of workload diversity and performance saturation with the predictive performance model of [2] to determine an appropriate frequency setting for each processor based on observed behavior.

- It applies frequency and voltage scheduling, as opposed to work scheduling, to a new domain, servers and server clusters. Previously, frequency and voltage scheduling was used primarily on mobile and client systems.
- It targets multi-programmed, multi-tasking systems.
- It ensures that the power remains below some global maximum value and responds to changes in the power limit by adjusting the frequencies and voltages appropriately.
- The techniques developed can be implemented in a number of different ways and in different portions of the hardware/software stack. In particular, it does not require changes to the operating system and its scheduling code.
- The results apply to server clusters as well as SMP systems.

The results reported here represent a prototype implementation and its initial evaluation rather than a definitive study of the underlying ideas and techniques.

2. Motivation

In order to better illustrate the ideas contained in this paper, a motivating example is introduced and used in the paper. The underlying details of the example are presented here. The system contains of four 140W CPUs, which consume 75% of the total system power. The entire system, including CPUs, memory, fans, etc., consumes 746W. Each power supply is only capable of providing 480W. There are different failure modes which can require a reduction in total system power, including site air conditioning failures, requests from outside parties to cap power consumption, and failure of a power supply. The example focuses on the last possibility, but the mechanism developed in this paper works in other situations as well.

When a power supply fails, the system must quickly respond to the failure of the power supply or a cascade failure can occur where the second power supply will also fail. At time T_0, the power supply fails and by time $T_0+\Delta T$, the system must be under the new power limit or the second power supply will fail. Time ΔT is a characteristic of the power supply and can be quantified. The problem then becomes one of developing a mechanism which can bring the system under the new power limit in less than time ΔT with a minimal loss in workload performance.

3. Related Work

The approach used in this work draws heavily from two distinct areas of prior research – dynamic frequency and voltage scaling as well as heterogeneous cores. Underlying details and additional prior work can be found in [2].

3.1. Dynamic Frequency and Voltage Scaling

Transmeta's LongRun [7] and Intel's Demand Based Switching [8] respond to changes in demand, but do so using a very simple model. In both schemes, an increase in CPU utilization leads to an increase in frequency and voltage while a decrease in utilization leads to a corresponding decrease. Neither one makes any use of information about how efficiently the workload uses the processor or about its memory behavior. Instead, they rely on simple metrics like the number of non-halted cycles in an interval of time.

Flautner and Mudge [3] explored the use of dynamic frequency and voltage scaling in the Linux operating system with a focus on average power and total energy consumption. They examined laptop applications and the interaction between the system and the user to determine the slack due to processor over-provisioning. They used frequency and voltage scaling to reduce power while consuming the slack by running the computation slower. Their Vertigo system dynamically uses multiple performance-setting algorithms to reduce energy.

Elnozahy, et al. [15] extended the ideas found in [3] to the domain of web server farms. They explore the use of DVS to respond to changes in server demands. They also examine the use of request batching to gain larger reductions in power during periods of low demand. The two techniques compliment each other, but neither provides a means to address peak power

This work differs by responding to easily observed changes in memory subsystem demands. It scales frequency and voltage in response to changes in available power and observed changes in memory behavior. Frequency and voltage scaling are performed only when the memory subsystem indicates there are a large number of memory stalls in the current phase, or the system is over its power limit due to a reduction in available power.

3.2. Heterogeneous Processors

The scheduling scheme described in this paper creates an environment in which an SMP server has heterogeneous processors since they differ in frequency and voltage. Prior work on single-ISA, heterogeneous processors falls into two distinct categories. The first uses a processor family which may be run at the same frequency, while the second category uses a processor family which cannot be run at the same frequency.

Single frequency heterogeneous processors have been studied by Kumar, et al. ([4], [5], [6]). Their work uses different generations of the Alpha processor family scaled into the same technology generation and run at the same frequency. The goal of the work is to minimize energy consumption while maintaining performance. The authors use a variety of metrics to identify which jobs should be assigned to which core, with all cores running simultaneously. Trial-and-error testing is used to identify the best-suited core. In contrast, this paper predicts performance to find the appropriate core.

Ghiasi and Grunwald ([9], [10], [11]) explored single-ISA, heterogeneous cores of different frequencies and different microarchitectures for controlling the thermal characteristics of a system. Applications run simultaneously on multiple cores and an operating system component monitors and directs applications to the appropriate job queues.

This work uses a single generation in IBM's PowerPC processor family, but cores are run at different frequencies. It also differs from prior work by using a commercial product and direct evaluation of techniques, rather than relying on simulation.

3.3. Hardware Approaches

Stanley-Marbell, et al, [12], propose a hardware mechanism that does frequency selection based on predicted performance loss. It makes use of the memory behavior of the workload to determine when the processor can run more slowly due to a heavy use of the cache and memory subsystem. It differs by virtue of its focus on microprocessor changes for the uniprocessing environment. The details of the performance model are also different since [12] works from processor/memory overlap values while this paper uses access counts.

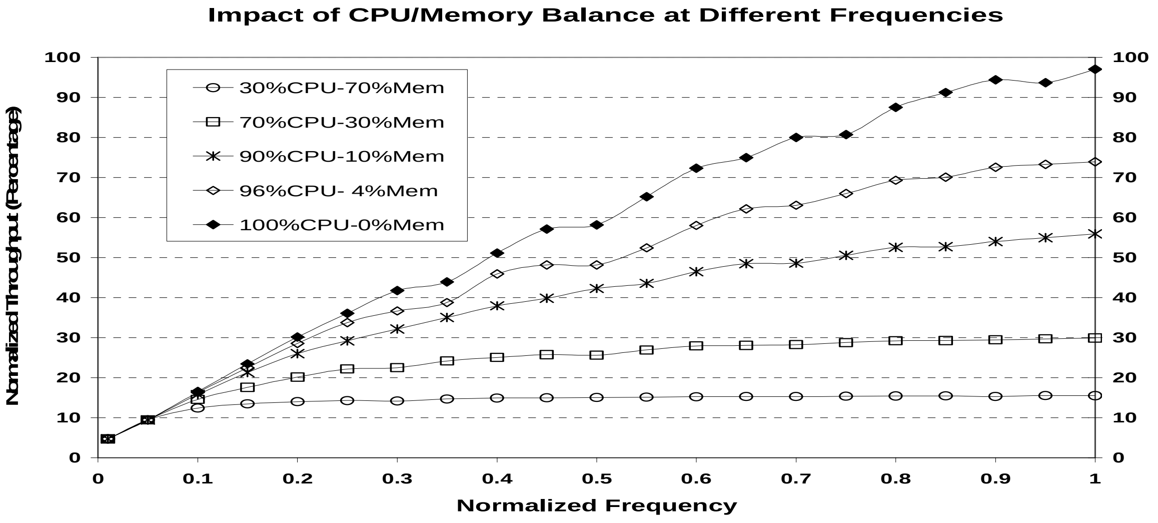

Figure 1: Performance saturation (Kotla, et. al [2])

4. Scheduling Mechanism

On a system or cluster whose processors support dynamic frequency and voltage scaling, the environment needs a mechanism to determine what frequency and voltage to assign to each processor. This section presents a methodology for predicting the performance impact of different frequency settings, given counts of the cache and memory accesses, and then using the predictions to guide the assignment of frequencies and voltages in order to meet power constraints and reduce average power dissipation.

4.1. Performance Saturation

Workloads often cannot make use of all of the available frequency due to the latencies associated with cache and memory accesses. This phenomenon is referred to here as *performance saturation*. Many programs obtain a limited benefit from increasing processor frequency due to the slow speed of memory relative to the processor. At some point, the speed of a program making memory references is limited by the speed of the memory. The ratio of memory-intensive to CPU-intensive work in a workload determines the saturation point as illustrated for a simple program by Figure 1.

Figure 1 illustrates that performance saturation allows the frequency to be reduced without a noticeable impact on application performance. Even when the power constraint is severe enough to require some performance penalty, it is generally possible to take advantage of performance saturation to minimize the overall performance penalty of the power-management action.

4.2. Workload Diversity and Phases

Workload diversity and the existence of different phases within a single workload are well-known and oft-used phenomena. Given an initial assignment of work to processors and strong or complete affinity of work to its originally assigned processors, it is reasonable to suppose systems and clusters often exhibit an overall diversity of behavior. The work running on some processors is more memory-intensive than the work running on other processors.

Unless the system explicitly load balances the memory intensity across the processors, it is likely that the system shows different cache and memory access rates on different processors. In clusters, where load balancing is difficult and expensive, if possible at all, diversity is even more common and more likely to persist over time. The tendency to assign work in a cluster by tiers where some machines run the web server, some the processing logic and some the database accentuates the level of diversity and stabilizes the phenomenon over time. This approach uses the aggregate behavior of all applications on a given processor, but workload diversity is still observed.

4.3. Predicting Performance at Frequency *f*

Since scaling-enabled processors typically offer only a fixed, small set of operating frequencies [13], predicting the performance impact on the current workload on a processor, in terms of its effect on the observed instructions per cycle (IPC), can be done by calculating a projected IPC at each available frequency. Existing processor hardware, such as that found in the IBM Power4, has performance counters that a scheduling mechanism may use to gather the number of accesses to each level of the memory hierarchy in an interval of time. To do the necessary IPC projection, the performance model used here and in [2] breaks the IPC into frequency-dependent and frequency-independent components.

In the following, α is the IPC of a perfect machine with infinite L1 caches and no stalls. α is a constant that takes into account both the instruction-level parallelism of the workload and the processor resources available to extract it. Each N_x is a count of the number of occurrences of a particular type of cache or memory reference, as provided by the performance counters, and each T_x is the time consumed by each reference. T_x is pre-determined for the particular processor by measurement of memory latencies and is assumed constant for simplicity[1].

[1] In reality, this is not true and is a source of error, but in practice it does yield a reasonable approximation for the purpose of frequency and voltage scheduling. Two different possible approaches have been investigated, but neither is used here. The first approach, described in [2], involves taking measurements at two separate frequencies. The second approach instead uses both best and worst case latencies to provide best and worst case bounds on performance at each frequency[17].

$$IPC \equiv \frac{Instructions}{Cycles} = \frac{Instr}{C_{stall} + C_{inst}}$$

$$= \frac{1}{\frac{1}{\alpha} + \frac{C_{other\_stalls}}{Instr} + \frac{1}{Instr}(N_{L2}T_{L2\_stalls} + N_{L3}T_{L3\_stalls} + N_{mem}T_{mem\_stalls}) * f}$$

At any given frequency, this equation can be used to predict the IPC at another frequency given the number of misses at the various levels in the memory hierarchy as well as the actual time it takes to service a miss. This provides a mechanism for identifying the performance loss of a processor. As expected, the more memory-intensive a workload is, as indicated by the high memory subsystem references, the more feasible it is to lower the frequency (and voltage) to save power without impacting the performance.

The scheduling algorithm uses the predicted IPC and the frequency used in the prediction to calculate the performance impact of running at that frequency. The following equations calculate the performance difference, *PerfLoss(f, g)* between the workload at the current frequency *g* and at the target frequency *f*. Values of *PerfLoss(f, g)* greater than 0 indicate a performance gain while those less than 0 show a performance loss.

$$Perf(f) = IPC(f) * f$$

$$PerfLoss(f, g) = \frac{Perf(f) - Perf(g)}{Perf(f)}$$

4.4. Applying Power Limits to Frequencies

To schedule frequencies and voltages based on a power constraint, the scheduler must first convert frequency and voltage values to corresponding power values. At each available frequency, the minimum voltage necessary to reliably drive that frequency is selected.

The equation $P = CV_{dd}^2 f + \beta V_{dd}^2$ gives the power as a function of frequency and voltage. C is the capacitance, and β is process- and temperature-dependent. The first term is the active power while the second is the static power which is due primarily to leakage. A typical computational approach is to calculate in advance the maximum power associated with each available frequency setting using the minimum acceptable voltage. This calculation ignores clock gating, but it provides an upper bound on power. When selecting a frequency for a particular maximum power, the system can then just select the highest frequency that yields a power value less than the maximum. Since the scheduler is scheduling voltages and frequencies across a set of processors, the power must represent the aggregate processor power consumption of the entire system.

5. The Scheduling Procedure

Given the tools presented in the previous section, a scheduler may calculate the frequency setting for each processor that yields the lowest power under the constraints of the maximum total power and the bound on the performance loss. If both constraints cannot be met, the scheduler must meet the power constraint while coming as close to the performance constraint as possible. For simplicity, the scheduling mechanism described here operates in two passes although it is possible to implement in a single pass scheduler.

Figure 2 shows the overall structure of the frequency and voltage scheduling procedure. It provides a simplified view showing only a single processor of an SMP or a single node of a cluster even though the algorithm schedules frequency and voltage across all processors and nodes. However, the power limit is a global one. The system uses power status and measurement data to determine the value of the limit and to monitor compliance with it.

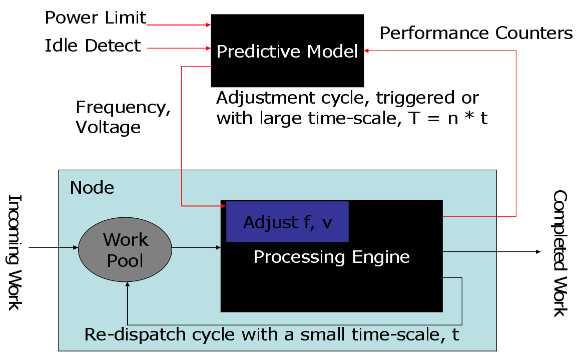

Figure 2: Scheduling structure.

Three possible triggers for changing frequency and voltage are considered here. First, the global power limit may change, due, for example, to the loss or the restoration of a power supply in the system. Second, there is a periodic readjustment of the voltage and frequency based on the expiration of a timer. Although the period of the timer, T, is a parameter to the mechanism, it is, for convenience, selected to be a multiple of the dispatch period, t, of the nodes. Generally, the multiplier, n, is fairly large to help stabilize the scheduler and amortize the overhead of both the inter-processor communication required and the frequency and voltage changes.

Finally, there are processors that idle by running a tight, CPU-intensive, loop rather than by halting. The Power4+ used in this research is such a processor. The scheduling mechanism runs the CPU-intensive idle loop at the highest frequency allowed by the power constraints. To avoid this problem, the scheduler needs input from the firmware or operating system indicating the processor is idle. On receiving this signal, it ignores the predictor and sets the frequency and voltage to their minimum values. When a processor exits from the idle loop, the idle detection mechanism sends the scheduler another signal indicating that the processor is no longer idle and that normal operation should resume. If the processor idles by halting and has a performance counter that tracks the number of halted cycles, then there is no need for the idle indicator.

The acceptable performance loss, ε, is also a parameter to the algorithm. When overall power is not tightly constrained, ε bounds the performance loss of the workload on each processor. Due to the relatively small number of possible frequency settings, it is not always possible to achieve very small values of ε, and its value must be greater than the minimum performance step caused by a change in frequency and voltage.

The mechanism described here is applicable to systems with a small, fixed set of available frequencies. This approach may prove computationally undesirable in systems with many frequencies or systems that do continuous frequency scaling. The computational limitations can be overcoming by extending the mechanism to include the identification of an "ideal frequency" at which to run the workload on a processor. The ideal frequency, f_{ideal}, is the frequency at which little or no performance is lost. The calculation of f_{ideal} is based on the two limiting cases, CPU-intensive and memory-intensive, from the IPC prediction equation discussed above.

$f_{ideal} = f_{max}$ if IPC > 1;

otherwise $f_{ideal} =$

$$\frac{Instructions * Perf(t, f_{\max}) * (1-\varepsilon)}{\alpha * Instructions - \alpha * T_{mem\_all} * Perf(t, f_{\max}) * (1-\varepsilon)}$$

where $T_{mem\_all} = N_{L2}T_{L2} + N_{L3}T_{L3} + N_{mem}T_{mem}$ and ε is a small constant used to indicate how much performance loss will be tolerated. This extension is not discussed further here. The fixed frequency set implementation provides a simpler illustration of the procedure used to schedule frequencies and voltages.

In both the fixed frequency set and non-fixed frequency set cases, the value of ε is used to bound the amount of performance lost by reducing the frequency. The goal of the scheduler is to have $PerfLoss(f_{max}, f) < \varepsilon$, where f_{max} is the nominal maximum frequency. For each possible frequency setting, the algorithm calculates the predicted IPC of the aggregate workload on that processor at that frequency. It then computes the performance loss versus the performance at f_{max} and chooses the smallest value that still has a performance loss less than ε.

It is important to note that ε may not limit the performance loss to less than ε. The ε limit is applied to a predicted performance which may be incorrect. Similarly, the use of aggregate performance counter data on each processor may mask the presence of a high CPU-intensity application among many memory-intensive applications. A reduced frequency in such a case will produce a larger performance loss than predicted. Phase transition can also cause a larger performance loss than predicted, particularly when the new phase requires a higher frequency due to more CPU-intensive instructions.

The data used in calculating the performance loss is aggregated across all programs running on a particular processor. The scheduler does not explicitly take multiprogramming or program phase transitions into account, but does work in their presence. This simplification comes at some cost in accuracy. The use of power measurement to monitor the total power consumption ensures that the system stays below the absolute limit. If necessary, the global limit may contain a margin of safety that forces a downward adjustment of frequency and voltage before any hardware-related, critical power limits are reached.

```
Let F = $f_0, f_1, \ldots, f_{max}$ be the possible
processor frequencies in ascending order

(1) for n in Nodes
      for p in Procs(n)
        for all $f_i$ in F
          calculate IPC($f_i$), PerfLoss($f_{max}$, $f_i$)
        $f_{n,p}$ = min $f_i$ such that $PerfLoss(f_{max}, f_i) < \varepsilon$

(2) while $\Sigma P_{n,p} > P_{max}$
      select n, p with smallest $PerfLoss(f_{max}, f_{less})$
      $f_{n,p} = f_{less}$

(3) for n in Nodes
      for p in Procs(n)
        $v_{n,p}$ = MinVoltage($f_{n,p}$)
```

Figure 3: Frequency and voltage scheduling algorithm.

Figure 3 shows the frequency and voltage scheduling algorithm. Step 1 iterates through all of the processors of the system and calculates the predicted IPC values, using the equation of Section 1.1, based on the target frequency settings and the observed values of the performance counters. The frequency selected is the lowest frequency that is predicted to keep the performance loss within ε. Although one more commonly thinks of downward adjustments in frequency, step 2 may, in fact, adjust it upward if a higher value is required to meet the performance loss criterion. At step 2 the algorithm adjusts the frequencies downward, if necessary, until the power constraint is met, selecting downward adjustments that have the least impact on performance. In step 2, for each n and p, the value of f_{less} is the next lower frequency in F below the $f_{n,p}$. Finally, at step 3, the algorithm relies on a table look-up to determine the lowest voltage setting allowed for the selected frequency of each processor. It may be the case that the voltage table is different for each processor if there is significant process variation among them.

The frequency and voltage scheduler reacts to the observed behavior of the workloads assigned by the operating system or cluster management software to each processor. On standard SMP operating systems, the kernel does some form of load balancing. Clusters also typically try to balance the load as well through clever initial assignments of work to nodes. However, there is nothing in the frequency and voltage scheduler that attempts to balance the system. It only attempts to minimize total power within the constraints of the limitation on maximum power and a bound on the performance lost.

Using the example system introduced in Section 2, the following is a sample calculation. The available frequency settings are: 1.0GHz, 0.9GHz, 0.8GHz, 0.7GHz, and 0.6GHz. At time T_0, a power supply fails. The system identifies and schedules frequencies for each processor. The system had a 294W total power constraint on its processors. The ε-contrained (Step 1) and actual frequency (Step 2) vectors for the system were found to be [1.0GHz, 0.7GHz, 0.8GHz, 0.8GHz] and [0.9GHz, 0.6GHz, 0.7GHz, 0.7GHz] respectively. The actual frequency vector has corresponding power and performance loss vectors of [109W, 48W, 66W, 66W] and [3.5%, 8%, 7%, 10%].

During the time between T_0 and T_1, the aggregate characteristics of the jobs running on Processor 0 change, becoming more memory intensive. At time T1, the calculations are performed again. Processor 0 has a new ε-constrained frequency of 0.6GHz, resulting in a new ε-contrained frequency vector of [0.6GHz, 0.7GHz, 0.8GHz, 0.8GHz]. It is now possible to schedule all processors at their ε-contrained frequencies, resulting in a power vector of [48W, 66W, 84W, 84W] for a total power consumption of 282W. In this case, the performance loss vector is now [ε%, ε%, ε%, ε%] because all aggregate workloads are scheduled appropriately. It is important to note that while the aggregate workload on a given processor does not suffer a performance loss, individual jobs may. This is a limitation of this approach, but in systems where job migration is difficult, this may be acceptable.

6. Prototype Implementation

To evaluate the frequency and voltage scheduler proposed in the previous section, the authors implemented a prototype version on an IBM pSeries system running Linux. The prototype runs on a single SMP. The development of a prototype for the cluster environment remains as future work.

The frequency and voltage scheduler (**fvsst**) prototype relies on an approximation of frequency scaling and cannot actually scale voltages. The underlying hardware provides mechanisms for throttling the pipeline use by interspersing the dispatch, fetch or commit cycles with dead cycles. Fetch throttling is used to mimic the effects of frequency scaling. Throttling can be used to cover the entire range from 0% to 100% frequency. This work assumes throttling yields the same power and performance results that using different frequencies for the processors would but ignores the settling time. Although not completely accurate, microbenchmarks indicate that this is a reasonable first approximation for the hardware used in the experimental studies.

The Power4+ processor used in the current generation of pSeries machines provides performance counters for cache and memory accesses. The prototype is a privileged user-level daemon process implemented as a single-threaded program. It relies on pre-existing kernel support to read the performance counters on all of the processors and to throttle them when necessary. This implementation is an initial prototype, and Section 9 discusses some possible enhancements.

The program collects the performance-counter data periodically and, after some number of collection cycles or when given a signal with a new frequency limit, executes the scheduling calculation and throttles the processors accordingly. The program generates both scheduling and performance counter data logs that provide performance and frequency information for monitoring and data analysis. Due to the limitations of the hardware, the program does not do any voltage calculations or detailed power computations. However, the data collected is sufficient for post-processing to determine the amount of power that would have been saved.

The Linux scheduler limits the choice of values for t, the dispatch cycle and the interval between readings of the performance counters. Values for t of less than 10 milliseconds interfere with the time quantum used in the operating system and result in inaccurate data collection. The values of T, the interval between scheduling calculations, are generally 10 times of those of t. Although it is feasible to change the frequency and voltage more often, it is not necessary to do so when considering aggregate behavior only.

7. Evaluation Methodology

This section describes the experimental platform, the metrics used to evaluate the benchmark results, and the benchmarks themselves.

7.1. Experimental Platform

The experiments described in this paper were performed on an IBM PowerPC-based pSeries P630 [14] system consisting of 4 1GHz Power4+ cores operating at a core voltage of 1.3 volts. Each core has a private 32 KB L1 instruction cache and a 64 KB L1 data cache. Two adjacent cores share a unified 1.44 MB data cache as well as a 32 MB L3. The machine has 4 GB of main memory. Using experimentation, it was determined that the nominal latency to various levels of the memory hierarchy are as follows: 4 to 5 processor cycles to the L1 caches, 15 cycles to the L2 cache, 113 cycles to the L3 cache, and 393 cycles to memory. These values are used by the scheduling implementation. The experimental platform runs SUSE Linux with a 2.6.5 kernel with modifications to support CPU throttling. An earlier prototype was developed on an experimental platform running Gentoo Linux with a 2.6.7 kernel with similar modifications to support CPU throttling.

The Power4+ cores have the property that they idle hot. The observed IPC of the idle loop is quite high, generally around 1.3, based on measurements taken for this paper. None of the idle-detection techniques discussed above are currently implemented.

Frequency (MHz)	Power (Watts)	Frequency (MHz)	Power (Watts)
250	9	650	57
300	13	700	66
350	18	750	75
400	22	800	84
450	28	850	95
500	35	900	109
550	41	950	123
600	48	1000	140

Table 1:Frequencies available for scheduling

Power estimates for each available frequency in the system are generated by the Lava power estimation tool developed by Devgan[16]. Lava is a circuit-level tool used to determine the shape of the power versus voltage and frequency curves for a particular technology. The peak power for frequencies considered by the scheduler is listed in . Power estimates for frequencies below 500 MHz are likely to be inaccurate, but extremely low frequencies were included in the experiment to find out what range of frequencies is truly required. In the event of a power supply failure with limited redundancy, extremely low frequencies may be required.

7.2. Metrics

The **fvsst** prototype must be evaluated by a variety of metrics. The predictor component must provide accurate predictions. **fvsst**, as a whole, must not impose a significant performance impact on the system. In power-constrained systems, it is also important to study the impact on power and performance.

7.3. Benchmarks

This work is a preliminary study of prediction to guide frequency and voltage scheduling. Evaluation has been done with the synthetic benchmark used in [2], which allows one to measure the performance variability of a program with an adjustable ratio of CPU-intensive to memory-intensive operations. The synthetic benchmark is a single-threaded program that accepts parameters that determine the ratio of memory-intensive to CPU-intensive work as well as the length of phases. It currently supports two (2) phases, but each phase may be of a different length and different memory-to-CPU intensity. It is constructed so that a miss in the L1 is highly likely to result in a memory access due to the large memory footprint. The program reports its performance in terms of throughput from its phases.

Although the synthetic benchmark provides an excellent tool for studying scheduler behavior, it does not truly mimic a real world application. In light of this, four additional benchmarks have been selected for study. These benchmarks are gzip, gap, and mcf from SPECCPU2000, and health from Olden. gzip and gap are CPU-intensive applications while mcf and health are memory-intensive applications.

8. Experimental Results

This section gives the results of the experiments run to evaluate the frequency and voltage scheduler using the prototype implementation. All results were run with T of 100 ms and t of 10 ms. When results are reported for only a single benchmark, the benchmark was run on CPU 3, and the remaining CPUs ran a "hot" idle. Results for only the CPU with a benchmark of interest assigned to it are shown here.

8.1. Predictor Accuracy and Prototype Overhead

The predictor used by **fvvst** must provide accurate predictions for this approach to be feasible. At the same time, the overhead of using **fvsst** must be low enough that it does not produce a significant negative impact on overall system performance. The error in the predictor is shown in Table 2. The final column of the table (CPU3*) is the deviation when the initialization and exit phases of the synthetic benchmark are eliminated from consideration. These results indicate that the predictor works well when the program is not in an initialization or termination phase. The remaining error is partially attributable to a bias in the predictor. The predictor currently does not account for non-memory stalls and uses constant memory latencies. These shortcomings are acceptable given the need for rapid response to power supply failure.

CPU intensity %	**IPC deviation**				
	CPU0	**CPU1**	**CPU2**	**CPU3**	**CPU3***
100	0.009	0.008	0.01	0.3	0.1
75	0.009	0.007	0.01	0.2	0.05
50	0.008	0.007	0.01	0.2	0.05
25	0.008	0.007	0.009	0.2	0.04

Table 2: Predictor Error. CPU3* is the error excluding initialization and termination phases.

Figure 4 shows the performance impact of running **fvsst** on the reported throughput of the synthetic benchmark is small. The performance degradation is more noticeable with the more CPU-intensive settings, but it is still no more than 3%. The **fvsst** implementation has not been optimized, and the performance impact should be lower when it is. The performance degradation reported here includes both the overhead of **fvsst** and the performance lost due to mispredictions.

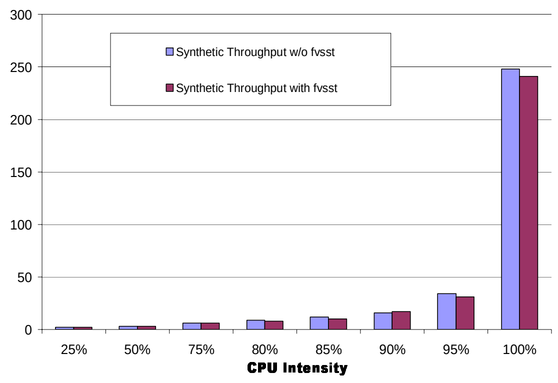

Figure 4: Performance Impact of fvsst scheduler on the synthetic benchmark.

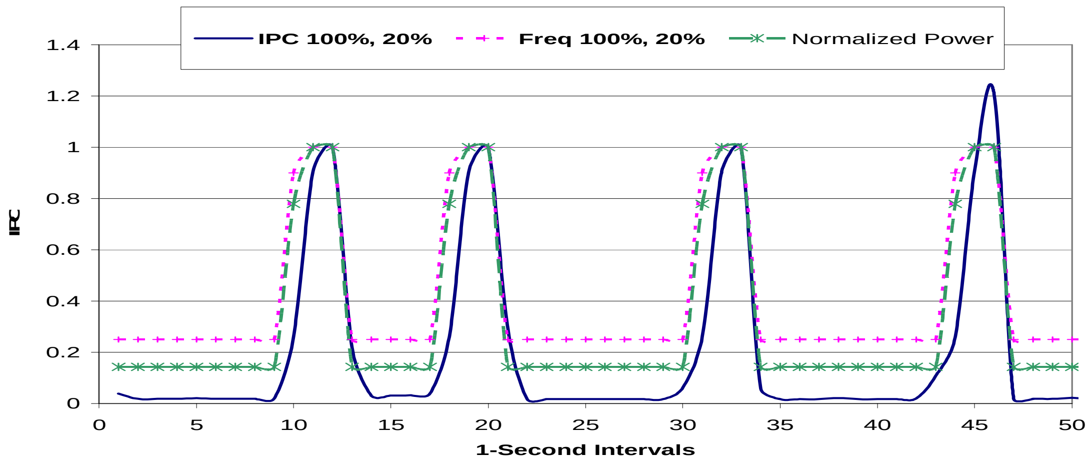

Figure 5: fvsst response to phase behavior

8.2. Phase Behavior

Figure 5 illustrates that **fvsst** accommodates and responds to phase changes. The frequency tracks closely with changes in the measured IPC. The settings of T and t are small enough to detect phase behavior because it occurs over a time-scale longer than 100 ms. The settings for T and t studied here obscure smaller phases and do not take advantage of them to reduce power. Figure 5 illustrates that IPC and the desired frequency trend together. Additionally, the power consumption of the system tracks the changes in frequency.

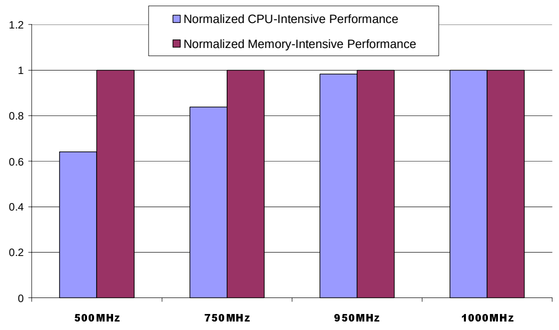

Figure 6: Performance impact of power limits

8.3. Response to Changes in Power Limits

One of the key motivations for doing power management in servers is to keep the system below a maximum power constraint. Due to environmental changes or failures of power supplies, the amount of power available may drop during system operation. To avoid failures, the system must be able to adapt to these changes. shows the performance impact of the scheduler under various power limits on the synthetic benchmark for its two phases, CPU-intensive at 100% intensity, and memory-intensive at 20% intensity. The results are for the system configured to use only a single processor. The performance values are normalized to their values at full power. For the memory-intensive phases, there is no degradation, while, for the CPU-intensive phases, the degradation is slightly less than one-to-one with the degradation in frequency since there are some memory-related stalls even in the CPU-intensive phase of the program.

Figure 7 illustrates the impact of power constraints more directly on a configuration consisting of a 100% CPU intensive phase and a 75% CPU intensive phase. When the CPU is able to run at full power, and thus full frequency, the high CPU-intensity phases can be accommodated. When the power limit is dropped to 75 Watts (a maximum frequency of 750 MHz), the high CPU-intensity phases can no longer be scheduled without suffering a performance loss. After an additional drop in power, to a limit of 35 Watts and frequency of 500 MHz, neither phase can be scheduled without performance loss. Both phases are now scheduled at the power-constrained frequency

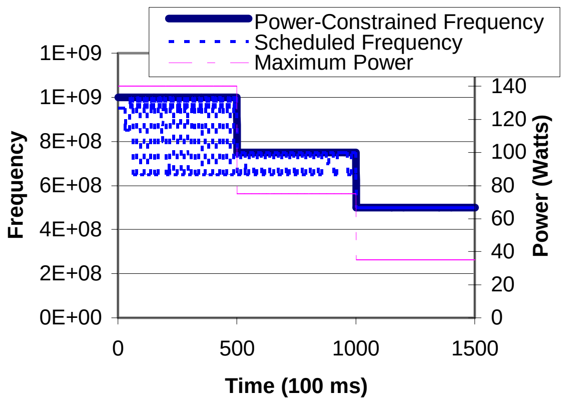

Figure 7: Power-constraints and responsiveness

8.4. Other Applications

Table 3 contains the power and performance results for the additional benchmarks studied. Performance loss is normalized against unconstrained performance and smaller numbers indicate larger performance losses. The CPU-intensive applications, gzip and gap, suffer noticeable, but sub-linear performance degradations as the power constraint is tightened. The memory-intensive applications, mcf and health, illustrate that it is possible to run certain applications under tighter power constraints without suffering a performance loss. In this case, neither benchmark lost performance at a power constraint of 75W. Both show significant performance loss at a budget of 35W because both possess large phases which would need to be scheduled at 600MHz (48W), exceeding the power constraint.

	gzip	**gap**	**mcf**	**health**
Perf @ 140W	1	1	1	1
Perf @ 75W	.79	0.8	.99	1
Perf @ 35W	.52	0.54	.81	.72
Energy @ 140W	.94	0.88	.43	.43
Energy @ 75W	.68	0.67	.43	.43
Energy @ 35W	.47	0.47	.31	.35

Table 3: Performance and power under constraint

The energy reduction provided by **fvsst** is significant even in the case where the processor may consume full power if it requires the corresponding frequency. CPU-intensive applications use less energy than the application running on a system which does not respond to changes in frequency needs. Even CPU-intensive applications have phases which are unable to utilize all of the resources of the pipeline. The energy reduction for memory-intensive applications is even larger. Even while providing full performance, the CPU consumes only 43% of the energy of a non-**fvsst** enabled system.

Figure 8 shows the frequency distribution of the benchmarks at different power-constrained frequencies. The frequency distribution at 1000Mhz indicates the desired frequencies in an unconstrained system. For the CPU-intensive applications gzip and gap, the effect of a power-constrained frequency cap is obvious at 750MHz. The two applications initially divided time primarily between 1000MHz and 950MHz. Once the frequency is limited to 750MHz, the two CPU intensive applications must run at the fastest frequency available. This limitation is reflected in the 500MHz maximum frequency case as well. The memory-intensive applications mcf and health do not show such behavior until the frequency limit drops to 500MHz. Both mcf and health have very similar performance characteristics and frequency distributions at 1000MHz and 750MHz. In particular, both applications spend the majority of their execution times executing with a frequency of 650MHz. Small differences in the frequency distribution do occur. When the frequency is limited to 750MHz, the work that was originally done at frequencies faster than 750MHz is now executed at 750MHz instead. This shift in execution frequency had little effect on the performance of mcf and health.

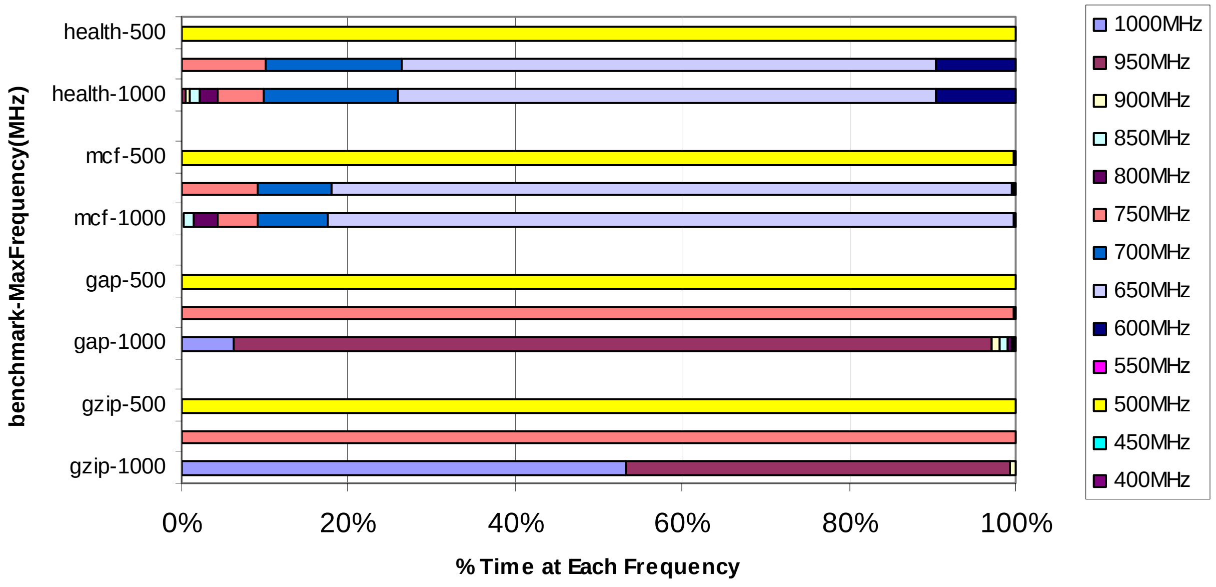

Figure 8: Percentage of time at each frequency

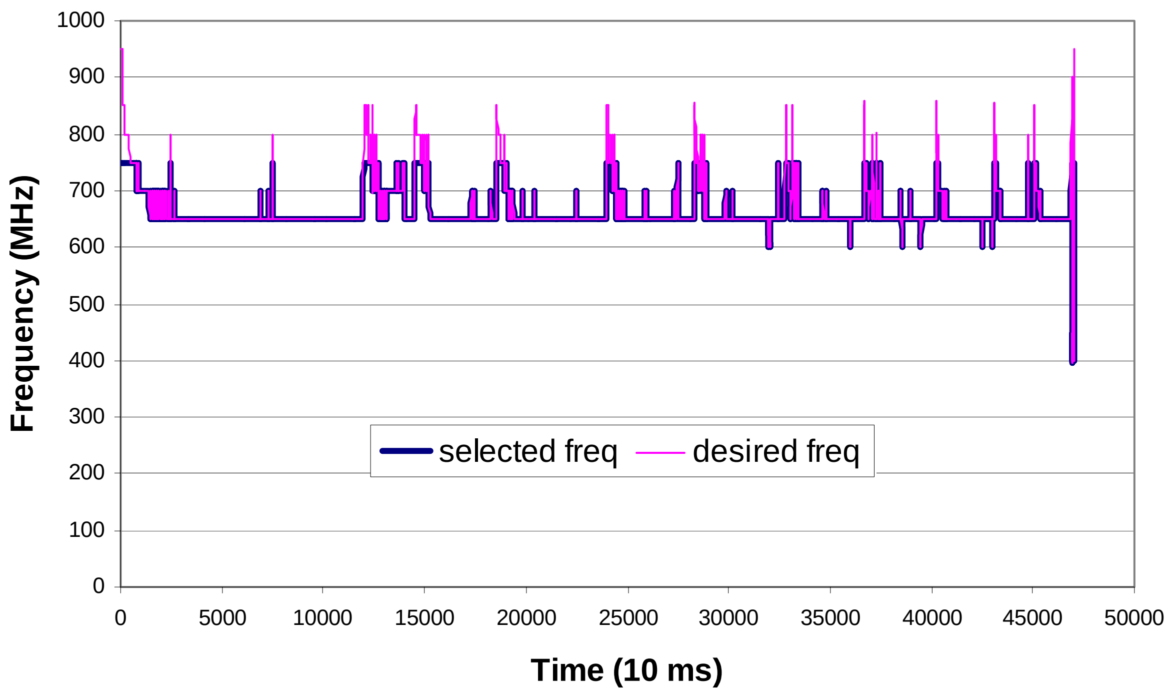

Figure 9: Actual and desired frequencies for gap at 750MHz (power limit of 75W)

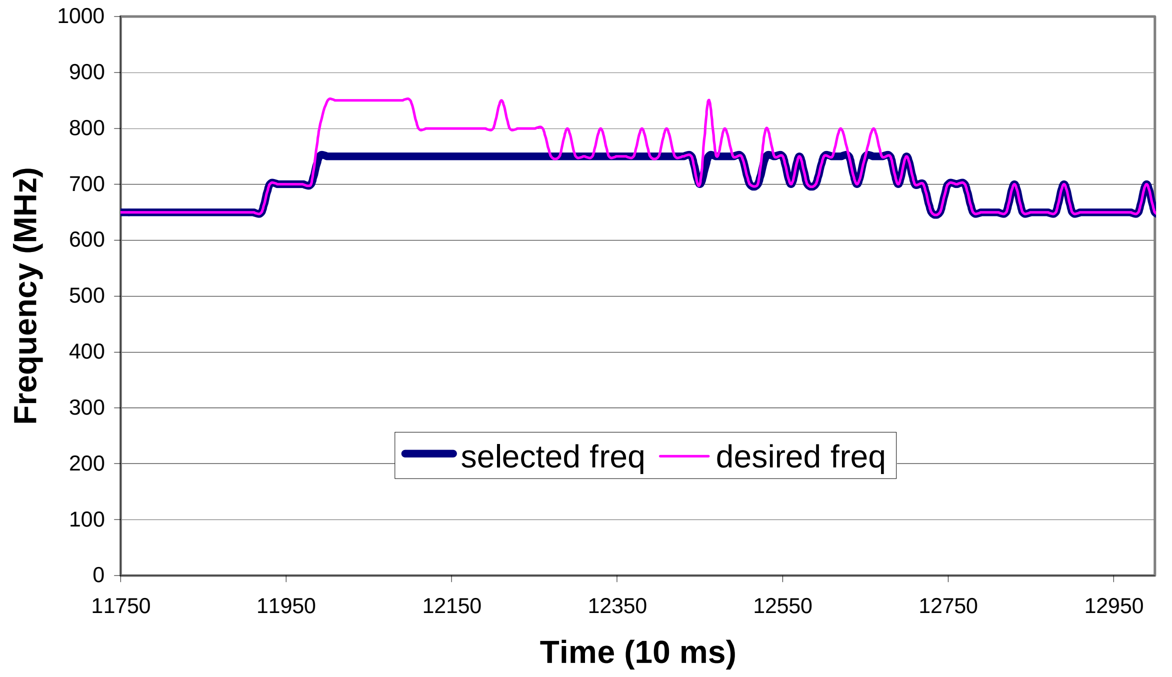

Figure 10: Magnification of time slice from Figure 9

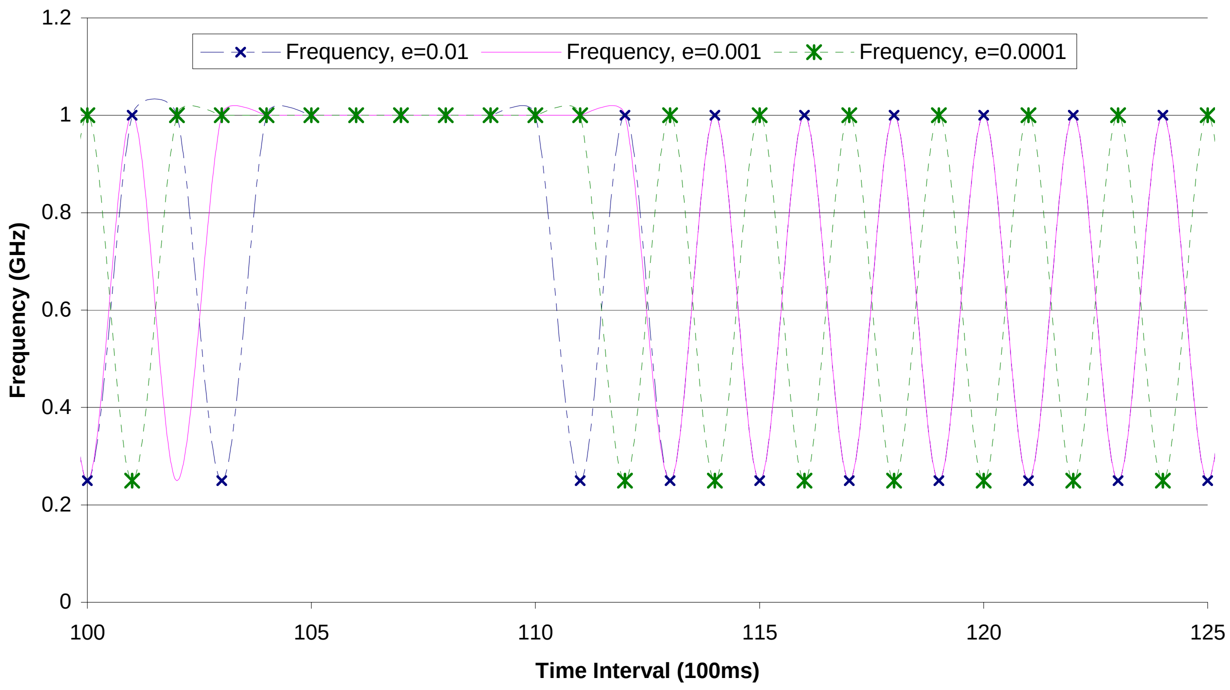

Figure 11: ε impact on frequency prediction

Figure 9. The application displays distinct phases. These phases have different frequency requirements. The actual and desired frequencies track closely, but

Figure 9. It illustrates how closely the frequencies track as well as a flattening effect at 750MHz when

Figure 11 illustrates the impact of different values of ε, the performance loss constraint on the predicted frequency for gzip. **fvsst** has similar, although not identical behavior for different values of ε. A value of 0.0001 provides the tightest performance bound, but other options perform comparably. A small degree of workload shifting occurs under larger ε values, but the general characteristics remain the same. Smaller values of ε tend to run at higher frequencies for longer periods.

9. Conclusions

This study demonstrates the value of scheduling frequencies and voltages rather than using a fixed set of frequencies and voltages and moving work to the processors with the appropriate performance characteristics. A scheme that schedules frequencies and voltages is applicable to environments such as clusters where migrating work is difficult. It also offers the advantage that one can implement it outside the operating system. Since operating system

The actual frequency used and desired frequency for gap at 75W (750 MHz) are shown in because of the frequency limit of 750MHz, gap cannot run at any frequencies higher than 750MHz. Figure 10 is an enlargement of a section of gap spends more time at 750MHz than it did previously.

Finally,

schedulers are, at best, difficult to change and, in many cases, impossible to alter, this is a tremendous advantage to system developers. The use of predictors makes it easier to manage the system and allows the system to take advantage of the natural diversity in workload to reduce power consumption at a minimal loss in performance.

Although this paper represents a start, this research is by no means complete. The mechanism presented here can be viewed as using the concept of ε-diminishing returns. The idea is that the scheduler considers all of the available frequency settings, determines the performance loss at each one and then picks the smallest one that comes within ε of the current performance. Rather than calculating the performance loss at each available frequency, the scheduler could instead calculate f_{ideal}. The idea is that the scheduler treats frequencies continuously rather than discretely and scales to the frequency determined by ε.

The paper reports on a preliminary implementation and a limited evaluation. Much more prototyping and measurement remain to be done. Currently, the implementation of the scheduler is as a single-threaded program using the kernel to collect the performance counter data. A better one would use multiple threads, two per processor. One thread on each processor collects the performance counter data from the counters at user level while the other one controls the throttling or frequency and voltage scaling for it. The prototype lacks an idle detection mechanism. For reasons of predictability, the code normally runs at the maximum round-robin priority. The Linux scheduler elongates the dispatch quanta for such threads, which is a source of overhead and inaccuracy when the applications also use the maximum priority.

10. Acknowledgment

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCH3039004.

11. References

[1] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Mike Kistler and Tom W. Keller, "Energy Management for Commercial Servers", Computer, volume 36, number 12, December, 2004, pages 39-48.

[2] R. Kotla, A. Devgan, S. Ghiasi, T. Keller and F. Rawson, "Characterizing the Impact of Different Memory-Intensity Levels", IEEE 7th Annual Workshop on Workload Characterization (WWC-7), October, 2004.

[3] K. Flautner and T. Mudge, "Vertigo: Automatic performance-setting for Linux", Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02), December, 2002, pages 105-116.

[4] Rakesh Kumar, Keith Farkas, Norman P. Jouppi, Partha Ranganathan, and Dean M. Tullsen, "A Multi-Core Approach to Addressing the Energy-Complexity Problem in Microprocessors", Workshop on Complexity-Effective Design, 2003.

[5] Rakesh Kumar, Keith Farkas, Norman P. Jouppi, Partha Ranganathan, and Dean M. Tullsen, "Single-IA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction", Proceedings of the 36th International Symposium on Microarchitecture, December, 2003.

[6] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, Keith I. Farkas, "Single – ISA Heterogeneneous Multi-Core Architectures for Multithreaded Workload Performance", Proceedings of the 31st International Symposium on Computer Architecture, June, 2004.

[7] Transmeta Corporation, "Transmeta LongRun Dynamic Power/Thermal Management", http://www.transmeta.com/crusoe/longrun.html.

[8] Deva Bodas, "New Server Power-Management Technologies Address Power and Cooling Challenges", Technology@Intel, http://www.intel.com/update/contents/sv09031.htm.

[9] S. Ghiasi and D. Grunwald, "Aide de Camp: Asymmetric Dual Core Design for Power and Energy Reduction", Technical Report CU-CS-964-03, Department of Computer Science, University of Colorado, Boulder, May, 2003.

[10] S. Ghiasi and D. Grunwald, "Thermal Management with Asymmetric Dual Core Designs", Technical Report CU-CS-965-03, Department of Computer Science, University of Colorado, Boulder, May, September, 2003.

[11] S. Ghiasi, "Aide de Camp: Asymmetric Multi-Core Design for Dynamic Thermal Management", Ph. D. thesis, Department of Computer Science, University of Colorado, Boulder, July, 2004.

[12] P. Stanley-Marbell, M. Hsiao and U.Kremer, "A Hardware Architecture for Dynamic Performance and Energy Adaptation", Power-Aware Computer Systems, Lecture Notes in Computer Science 2325, Springer Verlag, 2002.

[13] Intel Corporation, "Intel Pentium M: Enhanced SpeedStep Technology", http://developer.intel.com.

[14] Guilian Anselmi, Derrick Daines, Stephen Lutz, Marcelo Okano, Wolfgang Seiwald, Dave Williams and Scott Vetter, pSeries 630 Models 6C4 and 6E4 Technical Overview and Introduction, IBM Corporation, December, 2003.

[15] E.N. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers", Proceedings of the 4th Annual Usenix Symposium on Internet Technologies and Systems, Usenix Association, 2003.

[16] A. Devgan, "LAVA: Leakage Avoidance and Analysis", IBM User's Guide, 2004.

[17] R. Kotla, S. Ghiasi, T. Keller, and F. Rawson, "Scheduling for Heterogeneous Processors in Server Systems", Proceedings of Computing Frontiers 2005, 2005, To appear.