RC23747 (W0510-087) October 12, 2005
Computer Science

IBM Research Report

Evaluation of a Dynamic Processor Overclocking Implementation for Power-Constrained Systems

Juan Rubio, Karthick Rajamani, Freeman Rawson,
Heather Hanson, Soraya Ghiasi, Tom Keller
IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Evaluation of a Dynamic Processor Overclocking Implementation for Power-Constrained Systems

Juan Rubio, Karthick Rajamani, Freeman Rawson, Heather Hanson,
Soraya Ghiasi, Tom Keller
Power Aware Systems Group
IBM Research, Austin, TX 78758
{rubioj,karthick,frawson,hansonh,ghiasi,tkeller}@us.ibm.com

Abstract

Power and thermal constraints are proving to be limiting factors for computer system performance. Current approaches set the nominal frequency of a system to meet the timing, power, and cooling limits of the system, while running a worst-case workload. This limits the system to operate at this frequency even for workloads that have lower power and cooling requirements. As a consequence real workloads pay a performance cost for operating at a design point defined by the worst-case workloads.

We propose *Dynamic Processor Overclocking* (DPO) as a way of reducing the performance impact of the chosen power and thermal design point on workloads that have sufficient slack in their power and cooling requirements. Dynamic Voltage and Frequency Scaling (DVFS) has been widely studied in recent years, while solutions exploiting it have focused on saving energy, DPO uses this technique to obtain increased performance. To do so, the DPO module monitors the workload behavior and adjusts the frequency of the system at run-time to improve performance while maintaining the power below the designed limit.

The paper presents a prototype DPO implementation and evaluates its benefits using the SPEC CPU2000 benchmarks. We identify the workload characteristics that affect the speedups obtained from overclocking as well as how the power constraints affect the performance improvement we obtain from DPO.

1 Introduction

The last twenty years have seen a steady increase in the performance of computer systems at a rate unmatched by any other industry. But the increasing capacity and density of components in systems has resulted in a steady growth in power supply and cooling requirements. This growth in power and cooling requirements has now reached a point where power and thermal constraints are barriers to the continuation of cost-effective growth in computing performance.

1.1 Power Constraints Dictate Performance

A computer system or a component may be *power-limited* by:

- Power delivery: a limit in the amount of power that can be supplied to a module or system. This limitation can occur because of factors such as a limit in the capacity of the wires that feed current to the module, or

the maximum current a specific voltage regulator design can deliver.

- Power dissipation: a limit in the rate at which power (in the form of heat) can be removed from a system. This limitation is directly connected with the characteristics of the cooling system such as the thermal resistance and capacity of the heat sink and airflow through the system.

Both of these limitations can result from a combination of factors including physical and cost constraints.

In many systems, the primary consumer of power and generator of heat is the processor. Peak processor power consumption and heat are related to peak switching activity. So the conventional approach for ensuring a working system is to design the power and cooling systems for specific capacities tied to an estimate of the switching activity for the chip operating at a target frequency and associated voltage. Alternatively, for a system with specified power supply and cooling capabilities, the conventional approach is to fix the frequency of the chip and the voltage needed to support it at a value low enough to limit the switching power and, consequently, the total power to the specified capabilities. The chosen frequency of operation that matches the supply and cooling capabilities then dictates the performance of the system.

1.2 Fixing Frequency to Meet Power-Constraints Impedes Performance

Using a fixed operating frequency to obtain a known power consumption assumes that the operating frequency solely determines the switching activity and consequently the power consumption. However, the switching power is actually dependent on the both the frequency of the circuit operation and a highly workload-dependent switching factor: $P \propto Capacitance * V^2 * frequency * switching\ factor$. Applications that cause intense processor core activity either from a high degree of instruction-level parallelism (e.g., high instructions retired per cycle) or higher speculative execution tend to have higher core power consumption than those that do not. At the chip-level, high level of on-chip cache activity can also increase power consumption. As a consequence it is impossible to adopt a workload-agnostic approach for determining the power consumption even at a fixed frequency. The current way of meeting power constraints under variable switching activity and power consumption for different workloads is to have the chip vendors test their chips at a desired operating frequency for a special (set of) worst-case workload(s) which impose switching factors that they expect no practical workload can match, appropriately de-rate the power consumption for realistic workloads and provide the de-rated number to system designers building the cooling and power supply sub-systems. Most workloads still fall short of the vendor-specified (derated) power consumption numbers but have to pay the price in performance by using the

fixed frequency and resulting performance level that are guaranteed to be safe by the vendor specifications.

1.3 Dynamic Processor Overclocking to Boost Performance

In this paper, we present an adaptive approach for meeting power constraints that avoids the performance impact seen by real workloads when forced to operate at a frequency determined by worst-case workloads. Our scheme uses the voltage and frequency scaling capabilities of processors and the systems supporting them to boost application performance through *Dynamic Processor Overclocking* (DPO) whenever permitted by the power constraints. Essentially, we dynamically trade off frequency with workload-dependent switching activity to ensure that the power consumption is just within its limits.

Dynamic frequency and voltage scaling (DFVS) technology was initially adopted in processors designed for embedded devices such as IBM PowerPC 405LP [1] and Intel PXA (Xscale) processors [2]. The technology was used for energy savings in battery-operated environments. Even when variations of the technology moved to larger system environments as with the Intel PentiumM [3] for laptops and the IBM PowerPC 970[4, 5] and newer Intel Xeon processors in desktop and server systems, current systems exploit DVS *only to save power* when the system utilization falls below pre-specified thresholds. Examples include Windows XP's built-in power-saving modes exploiting the Pentium-M's DFVS capability, the PowerTune technology in the Apple G5 system exploiting the PowerPC970's frequency-scaling abilities, and Intel's Demand-Based Switching (DBS) for systems using some of Intel's newer Xeon processors. When DFVS is used just for power savings, the static margins for setting the maximum operating frequency have to ensure that the power constraints will be met irrespective of workloads. Thus, DVFS saves power by lowering the frequency and voltage when the machine is largely idle, but it does not attempt to do anything to increase the upper bound on frequency imposed by the design.

In contrast, our proposal utilizes DFVS to do two things.

- Meet system power constraints - We lower frequency when the workload-dependent switching activity at the operating frequency causes too high a power consumption relative to the supply and cooling capabilities.

- Increase performance for real workloads - We increase frequency whenever the workload-dependent switching activity and associated power consumption is sufficiently low relative to system capabilities.

The newly announced Intel Montecito [6, 7] processor adopts a dynamic approach to processor clocking somewhat similar to the one we describe, but it is managed by an on-chip embedded microcontroller, Foxton. In

contrast, we rely on features commonly found in a wide range of modern processors. Our work also provides a general methodology that can be used by a variety of implementations to improve performance while remaining under a power constraint. Section 3.2 provides more details on our implementation and differences with the Foxton approach.

The major contributions of this paper are

- the introduction of the notion of Dynamic Processor Overclocking as way of extracting additional performance while keeping the power within the design limits of the system
- the evaluation of the benefits of DPO on hardware that implements DFVS
- a study of the relationship between the workload and the system characteristics which determine how much the workload benefits from DPO.

The rest of the paper is organized as follows. The following section gives a broad overview of the related work. Section 3 provides a broader description of our DPO proposal. Section 4 describes our experimental platform, which is based around the Intel Pentium M processor, and our evaluation methodology for analyzing DPO's performance impact. In Section 5, we present the results of our evaluation, and we summarize our conclusions in Section 6.

2 Related Work

2.1 Performance-aware Processor Activity Scaling

There have been a number of efforts over the years examining the implementation and effectiveness of dynamic voltage and frequency scaling (DVFS) for saving power in embedded and laptop systems [8, 9, 10, 11, 12, 13]. Performance-oriented explorations include attempts to quantify and/or reduce the performance loss encountered when using DVFS to save energy and prolong battery life. In contrast, our work targets performance increases from the use of DVFS in a power-constrained environment.

Annavaram, et al., use energy per instruction to guide processor configuration in a four-processor system [14]. It focuses on exploiting thread-level parallelism in a workload to choose using multiple low-frequency cores instead of a single high-frequency core for a more efficient and potentially higher performance execution. Although this work does focus on performance improvement, it is applicable only to situations in which the degree of thread-level parallelism varies and the number of available processors can be dynamically varied.

There are a number of schemes that use feedback control to select DVFS settings. For example, Uht and Vaccaro's TEAPC uses a feedback control system to adapt to changes in system temperature by decreasing or increasing the frequency of the processor clock but makes no attempt to exceed the nominal frequency in situations where the temperature remains below their threshold under full CPU load [15]. An earlier work along somewhat similar lines but done for power control is that of Minerick, et al [16].

Felter, et al, examine the possibility of maximizing performance in the presence of reduced, fixed power budgets by dynamically changing the allocation of power between the processor and memory [17]. For example, during memory-intensive phases, more power can be allocated to the memory subsystem and less to the processor.

Recently, there has been more attention to the problems of power management for server and high-performance computing systems. For example, the work of Kappiah, et al, attempts to save power in high-performance computing environments by using DVFS to reduce the voltage and frequency on nodes that have less computation to do [18]. Femal and Freeh apply a similar scheme to the data center environment [19].

2.2 Power Modeling with Performance Counters

Although processor performance counters are designed to monitor performance, they also provide an indication of processor and system activity, useful indicators for indirectly inferring or predicting power. One of the earliest studies of using performance event counters as a proxy for power monitoring is [20], in which Bellosa, et al. develop an energy model using event counters for the XScale processor.

Bircher, et al. investigate the correlation of several performance monitoring counters with power for a Pentium4 system in [21]. They create an accurate model of average power with two event counters: fetched uops and microcode ROM-delivered uops.

Isci and Martonosi also monitor power and collect performance counter data on a Pentium 4 system, developing a per-component power model for the processor core. They also report similarity matrices, demonstrating power signatures that indicate distinct phases in the benchmark power behavior [22]. Contreras and Martonosi construct, using regression techniques, power models for the Intel XScale architecture, which like our Pentium M platform, has a limited number of performance counters [23].

Lee and Skadron develop a performance-counter-based temperature model which can model system temperature while an application is running [24].

2.3 Performance-Aware DFVS with Performance Counters

Power and temperature models based on performance counters can be used to evaluate management decisions and choose appropriate frequency and voltage settings for current conditions. For example, Weissel and Bellosa develop an energy-efficient operating system scheduler known as Process Cruise Control that adapts to changes in the composition of the instruction mix [25] as recorded by processor event counters. However, their goal is to save as much energy as possible within the constraint of a particular, allowable performance loss.

Kotla, et al., instead use dynamic program classification and scheduling techniques to adjust the frequency and voltage of a system to minimize the power consumption for existing performance levels [26]. Ghiasi, et. al., introduce an operating system scheduler that dynamically places tasks on the processor that most closely matches the application's ideal frequency setting [27] in a multi-processor system with heterogeneous-frequency processors. Both rely on performance counter-based prediction to guide decision making. Neither attempts to provide an explicit model of the power consumption or to ensure that any particular power constraint is met. Instead, they assume that the power associated with a particular voltage and frequency has a fixed, constant value. In contrast, we have a dynamic power model based on the performance counters, and exploit the variability in power consumption at the same frequency from different workload activities enabling the choice of a higher frequency than would be otherwise possible.

3 Dynamic Processor Overclocking

Although manufacturing technology and circuit design considerations both limit processor clock speed, increasingly the first-order limitation for a particular processor family on a specific platform is power.

System vendors set the nominal clock frequency of the processor in a system by using a worst-case test of the machine's power, even thought there are many programs that can run at higher clock frequencies without crossing this power limit. This slack in the power consumption for many workloads creates an opportunity to extract more performance for these workloads by dynamically increasing the processor clock beyond the nominal value dictated by the worst-case test. One can also take advantage of a power-adaptive processor clocking mechanism in other scenarios. In a data-center power management situation, one can envision placing power caps on different systems. A dynamic processor clocking system could be used to adapt to the set power limit while obtaining the maximum performance under that limit. The solution we have developed is equally adept at fitting those needs.

3.1 Definition

Dynamic Processor Overclocking (DPO) increases the processor frequency to values that are within its range of correct execution but which would exceed the power limitation imposed on the processor by the system environment, if not carefully controlled. It does so dynamically using the standard mechanisms for dynamic voltage and frequency slewing that are increasingly provided by the power management features of modern microprocessors, and it increases the clock speed only when the power consumed by running the current workload would remain below the defined power limit. For many programs and workload mixes, this means that the processor often runs at a higher than nominal frequency since the executing work requires less power than the worst-case workload used to set the nominal frequency.

Figure 1 illustrates the behavior of a system using dynamic processor overclocking with two possible frequency settings, f_1 and f_2, such that $f_1 > f_2$. The top two curves show that the power constraint is exceeded during portions of the system timeline when the processor uses only f_1 but never when it uses only f_2. The third graph depicts the behavior when the processor dynamically chooses between f_1 and f_2 by picking the higher value of f_1 whenever the power is low enough and f_2 otherwise.

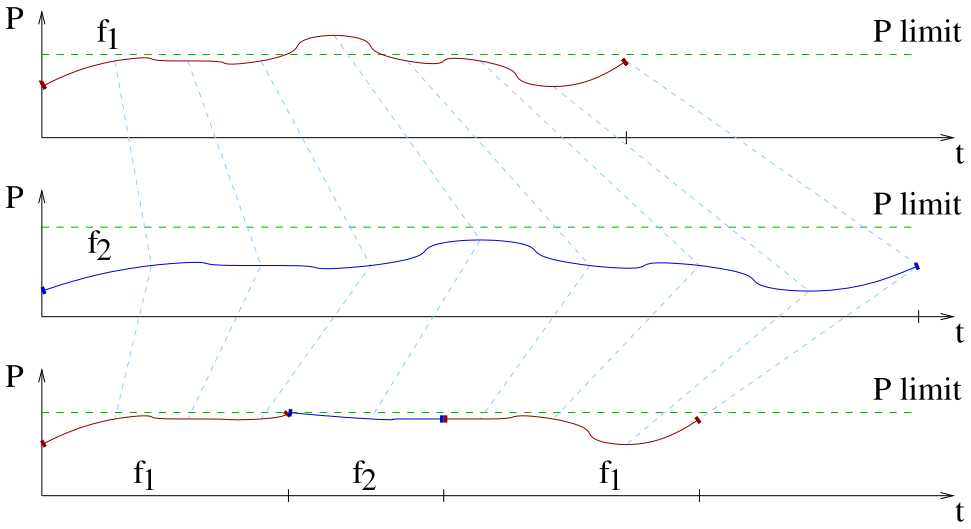


Figure 1: Example of Dynamic Processor Overclocking. All traces show the same number of instructions. The top plot is for a high frequency f_1, the second for a lower frequency f_2, and the bottom plot for a run using DPO.

.

3.2 Implementation

For a DPO implementation, it is important to correctly determine when it is safe to overclock the processor. This requires some idea of the current power consumption – either a good real-time power feedback or real-time access to good proxies for power. One would also need to have the ability to make a good estimate of the new power consumption when a frequency change is to be made or alternatively a very fast and fine-grain adjustment mechanism that one can use in a *test-and-set* fashion. Furthermore, a DPO implementation may be either in-band or out-of-band. An in-band implementation requires software running on the main processor(s) to implement the DPO algorithms. This software can potentially compete with the current workload for CPU cycles and additional resources. An out-of-band implementation could be either completely in hardware or with additional firmware support. Montecito's Foxton microcontroller is an example of an out-of-band hardware implementation for dynamic frequency adaptation [7] that operates using the *test-and-set* approach mentioned earlier.

In this paper, we present an in-band DPO solution prototype. In contrast to the *test-and-set* approach it uses microarchitecture performance counters and an activity-based power model [28] to estimate both the current power consumption and also predict the new power consumption if a frequency change were to be made. As we use the now conventional DVS mechanisms instead of a specially architected, fine-grain slewing mechanisms we have to couple it with good estimation models to avoid overshoots when the frequency is changed. The power model we use is tuned to our specific processor and cooling conditions – to avoid processor manufacturing variations and long-term temperature dependent effects – and can estimate the power consumption under different voltages and frequencies [1]. The model is formulated as a group of piece-wise linear equations, which are easily implemented in the form of lookup tables. We program and read the processor performance counters to obtain this information. In a real-world implementation one would need a dedicated copy of the same counter for effective deployment (conventional counters are for performance analysis capabilities and can be hijacked by tools providing such services).

Our DPO software is implemented in the form of a service that continually monitors the performance counters and estimates the power consumed while the main workload is running. This service is invoked approximately every 10 ms and was designed to be brief and robust to avoid negatively impacting the main workload (we target 150 samples per second for the results discussed in the paper). On each invocation, the DPO service reads the performance counters that are needed by the power model. Since these counters are read at a given frequency

[1] A document that presents a detailed description and evaluation of the power model is being prepared for a separate submission.

setting, a separate invocation of the model is needed to estimate the effect that frequency has over the measured performance counters before we can estimate the power at those frequencies. Rate of some architectural events would scale linearly with frequency ($\frac{f'}{f} \times Counter(f)$) while some are unchanged by frequency scaling [28]. So we not only use a different set of coefficients in our piece-wise model for a frequency change but also used an estimated value for the model inputs as well. All terms in the power model have positive coefficients. Thus, to produce a conservative power estimate, we follow a scaling of the form:

$$Counter(f') = \begin{cases} Counter(f), & f' \leq f \\ \frac{f'}{f} \times Counter(f), & f' \geq f \end{cases} \quad (1)$$

Once we obtain the estimated counters at each frequency, the DPO algorithm determines the highest frequency setting that can be used without breaking the power limit. For the results discussed in this paper, the equations employ a single counter-based model using the Instruction Decoded counts. Once we decide to change a frequency we can do so immediately or implement a sliding window that reduces the occurrence of unnecessary and potentially harmful transitions for workloads that exhibit a bursty behavior. For the results discussed, we make down-scaling decisions immediately and delay up-scaling decisions to confirm that the opportunity for boosting exists across multiple samples. Section 5.3 briefly discusses the need for this approach.

4 Evaluation Methodology

4.1 Processor and system board

To study the value of DPO, this paper uses a real machine rather than simulation. The specific platform studied here is a Pentium M-based system running Windows XP. The 90 nm Pentium M processor "Dothan" has a 32 KB primary instruction cache, 32 KB primary data cache, and a 2MB, 8-way unified secondary cache [3]. The processor has programmable performance counters to track events such as the number of instructions decoded or memory references during program execution.

Dynamic voltage and frequency scaling (DVFS) allows the system to operate throughout a range of power and performance levels. The processor supports dynamic voltage and frequency scaling from 600 MHz to 2.0 GHz, with its corresponding supply voltage in the range of 0.988 V to 1.34 V. The driver we developed can adjust the processor frequency within its full operating range in steps of 100 MHz, and results in processor stalls of up to 10 μs. The processor chip is paired with an Intel 855GME chipset and 512 MB of DDR SDRAM memory on a

Radisys uni-processor motherboard [29].

4.2 Power Measurement

We modified the hardware to allow us to measure and record the power consumption of the processor. We used on-board sense resistors placed between each voltage regulator module (VRM) and the processor and placed data acquisition probes to monitor processor supply voltage and current levels. Figure 2 shows the system under test and the ribbon cables that connect the probe points to the data acquisition system. We collect and analyze power data on a separate computer to avoid interference with the workloads executing on the system under test. A National Instruments data acquisition system samples current and voltage values and interfaces with a system that executes a custom LabView program to capture a voltage trace of each probe.

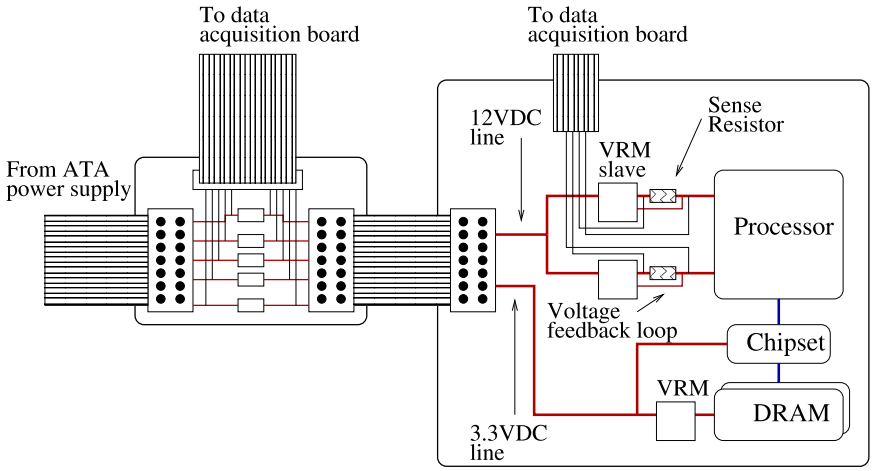


Figure 2: Experimental platform: system under test with sense resistors and data acquisition probes to measure processor power.

4.3 Performance Measurement

In addition to application execution time, we monitor performance events throughout execution of each benchmark. We developed light-weight software to collect performance data. This software configures the performance counters sets the processor's frequency and voltage and then spawns the benchmark under test. During benchmark execution, the software reads performance counter values with the RDPMC instruction at approximately 10 ms intervals[2] and then generates an output file of timestamps and counter values.

The Pentium M performance registers hold two performance counter values. Some counters use additional bits to specify variations of the data collected, such as including or excluding prefetched lines in a cache access

[2]This value is limited by the resolution of the timer mechanisms in Windows XP

count [30].

4.4 Workloads

For this dynamic processor overclocking study, we characterized the full suite of SPEC CPU2000 benchmarks, both integer and floating-point applications [31]. We used the "reference" input set with the `runspec` script supplied by the SPEC organization to run each benchmark in the suite with appropriate options and input files. Some programs, such as `gzip`, run multiple times with different input files. In those cases, we follow the SPEC standard of summing the individual run times to produce a total result for the benchmark. The study uses highly optimized binaries compiled for execution on Windows XP with Intel compilers.

4.5 Methodology

The experiments reported here use a frequency range between 1.8 GHz and 2.0 GHz. This range provides a very noticeable power difference as well as a measurable effect on the performance of programs that are sensitive only to the clock speed of the processor. But it is narrow enough to represent a realistic overclocking across a range of processors and processor families.

Since the machine used in the experiments is based on commercially available parts, we have direct access to information about its design limits. However, for the purposes of some of the experiments that we describe, we determined a power limit using a worst-case methodology. We assume that 1.8 GHz is the nominal frequency and that 1.9 and 2 GHz are overclocking frequencies. A highly optimized version of the Linpack benchmark run at 1.8 GHz provides a value for the assumed processor power limit. The peak power from the worst-case Linpack test of 16.5 W is the power limit used in much of our analysis of dynamic processor overclocking although we report some results with higher and lower power limits.

For each run, the instrumentation gathers 100 processor voltage and current samples per second. Our DPO implementation adjusts the frequency to the desired value, configures the performance counters, and creates a process that will run the benchmark with the highest user-level priority. It is also responsible for gathering performance samples of instructions executed and elapsed cycles every 10 ms. Further, it is also designed to raise the voltage of a general purpose I/O pin (*marker*) to indicate the power instrumentation that a managed application is running. The voltage of the marker pin is lowered once the application signals its completion.

5 Results and Discussion

This section presents the results for our DPO implementation discussed in Section 3. The high-level results are summarized first. Sections 5.2 and 5.3 present additional material regarding workload characteristics and implementation issues that impact the benefits obtained from DPO.

5.1 Results for DPO

Our experiments are done with a nominal (default) operating point of 1.8 GHz. As indicated above, the default power limit set for the nominal operating point is 16.5W. DPO is allowed to exploit higher level operating points of 1900MHz and 2.0 GHz while attempting to meet the 16.5W power limit. Speedups relative to performance at 1.8 GHz and the maximum obtainable at 2.0 GHz serve as the performance comparisons. Behavior with respect to the power limit is measured as a percentage of execution time for which the power signal filtered at 10Hz exceeds the given power limit. Power excursions at this frequency are relatively critical for supply designs.

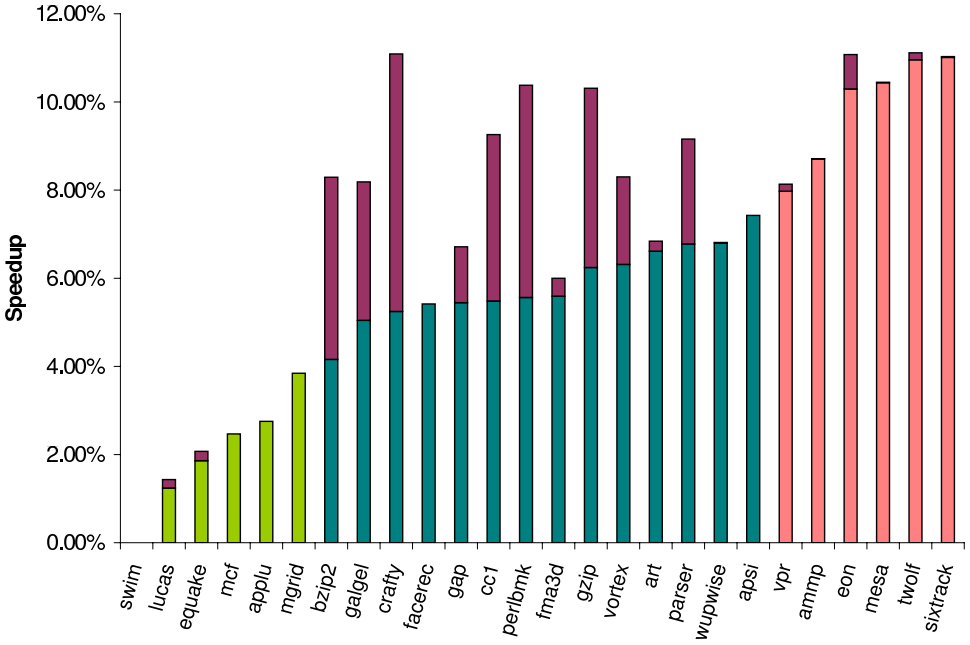


Figure 3: Speedup for SPEC CPU 2000 applications over 1.8GHz from Dynamic Processor Overclocking with a 16.5 W power limit.

Figure 3 shows the speed-up achieved by using DPO with a 16.5 W power limit in the form of a stacked bar chart. In each stacked-bar, the lower bar represents the speedup with DPO and the upper bar the additional speedup that can be obtained at 2.0 GHz if there was no power limit (essentially an upper bound on speedup for that application). The applications are sorted left-to-right in the order of increasing speedup from DPO. The

first point to note is there are many applications that obtain a significant boost in performance from DPO. This validates the motivation behind DPO that dynamic adaptation of operating points to workload-specific power consumption can have significant performance benefits over a static fixed operating point solution (which is 1.8 GHz for these experiments). The next point to note is that there is significant variation among the benefits seen by the applications. For discussion purposes, we break the applications into three groups by the range of speedups:

- Below 4% – `swim`, `lucas`, `equake`, `mcf`, `applu`, and `mgrid`
- Between 4% and 8% – `bzip2`, `galgel`, `crafty`, `facerec`, `gap`, `cc1`, `perlbmk`, `fma3d`, `gzip`, `vortex`, `art`, `parser`, `wupwise`, and `apsi`.
- Above 8% – `vpr`, `ammp`, `eon`, `mesa`, `twolf`, and `sixtrack`.

Note that for applications in the lowermost speedup group there is little additional performance improvement even at 2.0 GHz with unlimited power. Same is true for the highest performance group. There are two sub-groups in the middle performance group. `bzip2`, `galgel`, `crafty`, `gap`, `cc1`, `perlbmk`, `gzip`, `vortex` and `parser` can all benefit from higher frequency and unlimited power but cannot reach that speedup with DPO. The rest of the applications in the middle performance group cannot get any additional speedup even at 2.0 GHz.

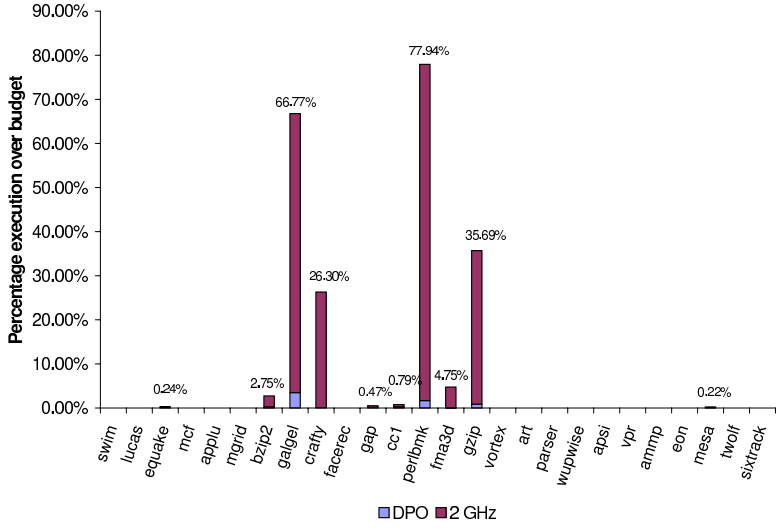


Figure 4: Elimination of Overshoots at 16.5W Budget

The other aspect of DPO besides adaptive performance improvement is adherence to the power limit. Figure 4 shows the extent of overshoot above the 16.5W limit. Again the graphs shows stacked bars. The lower bar

represents the overshoots for the DPO implementation and the upper bar represents the additional overshoot if operating always at 2.0 GHz. Overshoots with DPO are zero for most applications. The very high fraction of time in overshoots for 2.0 GHz shows how inconceivable it is to operate the system for this collection of applications at a fixed frequency of 2.0 GHz.

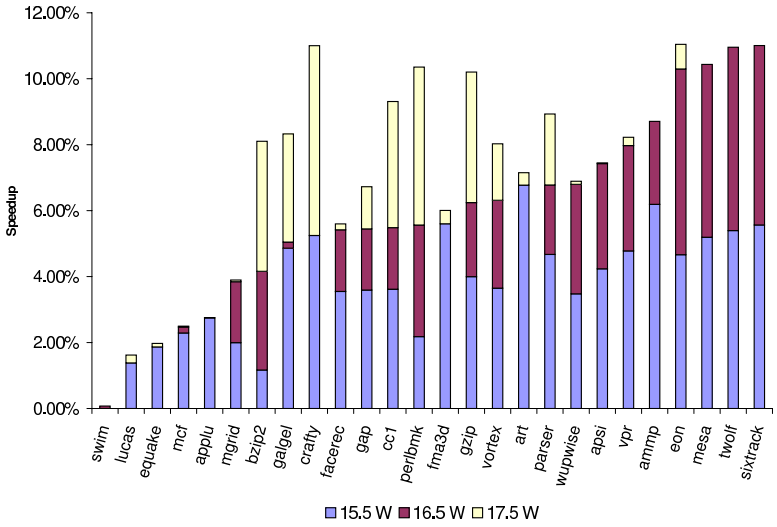


Figure 5: Adapting to Power Limit with Dynamic Processor Overclocking

An additional point to note is that the applications with the longer duration overshoots with 2.0 GHz – `galgel`, `crafty`, `perlbmk` and `gzip` are also in that sub-group of the medium speedup applications that have big gaps between DPO speedups and the upper bounds for speedups at 2.0 GHz. Figure 5 provides an explanation for this behavior. It shows the speedups with DPO for three different power limits – 15.5W, 16.5W, and 17.5W. Again it is a stacked-bar chart with each stack showing the additional speedup over the previous limit.

The medium speedup applications that could not reach their potential with DPO in Figure 3 all reach their potential when the power limit is increased to 17.5W. These same applications also exhibit significant overshoots at 2.0 GHz over DPO. Thus, DPO gives up performance on these applications to meet the power limit. But, if for some reason, the power limits can be raised it can take full advantage of the increased power to obtain maximum performance for these applications.

In the following discussion, we will refer to this subset of medium speedup applications as *power-limited* applications. Such a labeling of these applications may seem power-limit-specific. For example, if our power limit for the nominal operating point were 15.5W, all the applications in the highest speedup group, from `vpr` on the left to `sixtrack` on the right, would also be “power-limited”. However, at the same power-limit, our

medium performance "power-limited" sub-group is more constrained by the limit than the higher performance group.

In the next section, we look at some of the reasons for this behavior and, in general, the variable speedups from DPO for the different applications.

5.2 Application Characteristics that Impact DPO Benefits

To investigate the variable performance of DPO for different applications, we collect more detailed activity data for the the applications using processor performance counters. The counters can be programmed to collect counts of different microarchitecture events. For our discussion here, we present the data collected from seven performance counters in Table 1. The data is presented in the form of rates for each of these events, with applications in each column sorted in descending order by the event rates shown in that column.

Table 1: Relative ranking of the SPEC CPU 2000 benchmarks (sorted in descending rank) in terms of architecture performance events.

DCU_M_O per cycle		RESOURCE STALLS per cycle		INST RETIRED per cycle		INST DECODED per cycle		L2_LD demand per cycle		L2_RQSTS per cycle		MEM_RQSTS per cycle	
art	2.29	mcf	0.876	sixtrack	1.41	crafty	1.67	art	0.091	art	0.057	swim	0.012
mcf	1.17	swim	0.845	crafty	1.35	gzip	1.49	gzip	0.044	gzip	0.031	equake	0.011
swim	1.12	applu	0.795	perlbmk	1.33	sixtrack	1.44	galgel	0.039	twolf	0.025	lucas	0.011
fma3d	1.1	art	0.792	mesa	1.28	perlbmk	1.41	mcf	0.036	galgel	0.023	mgrid	0.011
equake	0.89	equake	0.788	vortex	1.21	mesa	1.33	twolf	0.031	vpr	0.020	mcf	0.001
apsi	0.84	lucas	0.703	wupwise	1.19	parser	1.32	equake	0.026	mcf	0.019	applu	0.009
applu	0.55	ammp	0.677	gap	1.12	bzip2	1.27	swim	0.025	cc1	0.015	fma3d	0.008
ammp	0.52	apsi	0.671	galgel	1.12	twolf	1.25	mgrid	0.023	swim	0.015	art	0.007
mgrid	0.47	fma3d	0.634	gzip	1.1	vortex	1.23	vpr	0.022	vortex	0.014	wupwise	0.006
vpr	0.46	mgrid	0.604	eon	1.06	wupwise	1.23	parser	0.019	parser	0.013	facerec	0.006
galgel	0.38	facerec	0.518	parser	1.01	gap	1.2	cc1	0.019	ammp	0.013	gap	0.005
wupwise	0.3	sixtrack	0.48	mgrid	0.97	eon	1.17	ammp	0.017	apsi	0.013	apsi	0.004
vortex	0.28	wupwise	0.463	bzip2	0.95	galgel	1.13	vortex	0.014	mgrid	0.012	vpr	0.003
gzip	0.27	galgel	0.455	cc1	0.91	vpr	1.13	apsi	0.014	equake	0.011	galgel	0.003
bzip2	0.27	gap	0.443	twolf	0.9	cc1	1.09	wupwise	0.013	bzip2	0.011	ammp	0.003
cc1	0.27	vpr	0.44	facerec	0.9	mgrid	0.97	facerec	0.011	crafty	0.010	bzip2	0.003
lucas	0.21	twolf	0.422	vpr	0.75	facerec	0.9	bzip2	0.011	lucas	0.009	vortex	0.003
parser	0.19	parser	0.338	ammp	0.74	ammp	0.79	fma3d	0.010	applu	0.007	parser	0.002
facerec	0.19	vortex	0.336	apsi	0.69	apsi	0.7	lucas	0.01	facerec	0.006	cc1	0.002
twolf	0.18	bzip2	0.311	lucas	0.58	lucas	0.58	applu	0.01	fma3d	0.006	mesa	0.001
gap	0.09	eon	0.309	fma3d	0.55	fma3d	0.57	gap	0.005	wupwise	0.005	gzip	0.001
perlbmk	0.04	mesa	0.277	applu	0.44	equake	0.44	crafty	0.005	perlbmk	0.005	perlbmk	0.001
crafty	0.03	gzip	0.271	equake	0.42	applu	0.44	perlbmk	0.004	gap	0.004	sixtrack	0.000
sixtrack	0.01	cc1	0.266	art	0.36	art	0.38	mesa	0.001	mesa	0.002	crafty	0.000
mesa	0.01	crafty	0.134	swim	0.18	mcf	0.24	sixtrack	0.001	sixtrack	0.001	twolf	0.000
eon	0	perlbmk	0.097	mcf	0.16	swim	0.18	eon	0.000	eon	0.001	eon	0.000

The first two columns DCU Miss Outstanding and Resource Stalls represent incidence of stalls in the memory hierarchy and processor pipeline. In general, the higher these values the less effective the application is in making use of a higher frequency. It is also potentially less power consumptive as it cannot make effective use of the memory hierarchy or the pipeline. Instruction Decoded/cycle and Instruction Retired/cycle represent measures of pipeline throughput – the former accounting for speculative activity and consequently a truer measure of power consumption and the latter of actual completed activity. The last three counters are indicative of memory hierarchy dependence. Relative L2 Requests activity is used below to explain relative power differences from on-chip cache activity. L2 Demand Loads and Memory (DRAM) Requests are used to identify dependence on lower speed memory hierarchies and consequently lower effectiveness in exploiting higher frequencies.

From the table it can be seen that the applications in the lowest speedup group – `swim`, `lucas`, `equake`, `mcf`, `applu`, and `mgrid` – share the characteristics of relatively high Resource Stalls per cycle and/or DCU Miss Outstanding per cycle. They have relatively high values in the Memory Requests per cycle column indicating that the DCU stalls are from waiting for DRAM, which does not scale with processor frequency rather than just a high fraction of L2 requests, whose access speeds scale with processor frequency. These stalls effectively restrict the applications from exploiting higher frequencies for performance gain. This applies to both DPO and a fixed 2.0 GHz chip. This result is consistent with those from an earlier study of memory hierarchy usage for SPEC benchmarks by Kotla et al. [32].

The non power-limited applications in the medium speedup group – `facerec`, `gap`, `fma3d`, `wupwise`, `apsi` and `art` – also share some of these characteristics but are less extreme in their stall rates allowing for higher speedups. For some of them, memory hierarchy dependence is stronger on the L2 than on DRAM. For instance, `art` occurs higher in the L2 Requests column than in the Memory Requests column. This makes it possible for it to take advantage of an increase in frequency as the on-chip L2 access also speeds up.

The most power-limited applications – `crafty`, `gzip`, `perlbmk`, `bzip2`, and `galgel` – display different characteristics . They have relatively high Instructions Decoded per cycle and/or L2 Requests per cycle. That is, they have a high activity rate in the core or in the L2, accounting for their higher power consumption at the same frequency. In the table one can see them near the top of these two columns. Also they are relatively low in the stall cycle columns indicating that if run at higher frequency (i.e. if power was not limited), they would see higher speedups.

The high speedup group applications share the characteristics of relatively low dependence on the slower

memory hierarchy levels. They are towards the bottom of the DCU Stalls and Memory Requests per cycle columns. At the same time they are not pushing against the power limits by being at the top of both the Instruction Decoded per cycle and L2 Requests per cycle columns. For instance, `sixtrack` is quite noticeable at the top of the Instructions Decoded column, but it is towards the bottom of the L2 Requests per cycle column.

To summarize the variable benefits for applications with DPO, applications that scale best have low memory hierarchy dependency and resource stalls. Applications that see little benefit are the ones that encounter many memory stalls due to heavy usage of slower levels of the memory hierarchy or significant numbers of processor resource stalls. High-intensity use of the on-chip memory hierarchy and processor pipeline, even if not resulting in stalling, can also result in the application encountering a power-constraint bottleneck, preventing it from exploiting higher frequencies with DPO. More details about the impact of application characteristics can be found in our technical report that investigates the maximum benefits obtainable from DPO based on application analysis on the same platform [33].

In the following section, we look at characteristics of key components of the DPO implementation and how they could affect its benefits.

5.3 Considerations for a good DPO Implementation

5.3.1 Impact of Monitoring Facilities Characteristics

A key aspect of the DPO implementation is the ability to periodically sample the architectural events. This is required to estimate the impact of a change in frequency on the power consumption. As this input is used to control/regulate the power consumption, it has to be done at a rate more than an order of magnitude higher than the power consumption signal of interest. We use the Windows timer facilities to schedule the monitoring and control events in our in-band control system. Figure 6[a] shows the desired interval time versus the most common bin of actual interval time obtained when we are running `galgel`. It appears to show a relatively adequate timer mechanism for our purposes at the 150 samples per second rate used for the results presented here. However, Figure 6[b] reveals a troublesome detail. It shows the histogram for the monitoring intervals during the run, timed using the processor TSC counter. A significant fraction of the samples are at much longer intervals than at the desired rate. This means we can miss the opportunity to boost performance or take corrective action by lowering the frequency because we detect the change in workload behavior too late.

The latter appeared to be the case with `galgel`, which is one of the highest power consumptive applications in the set, and leading to a high fraction of overshoots. (The results presented earlier with low overshoots are

with corrective action, described below). When we discovered the overshoots, we considered two changes to our basic algorithm.

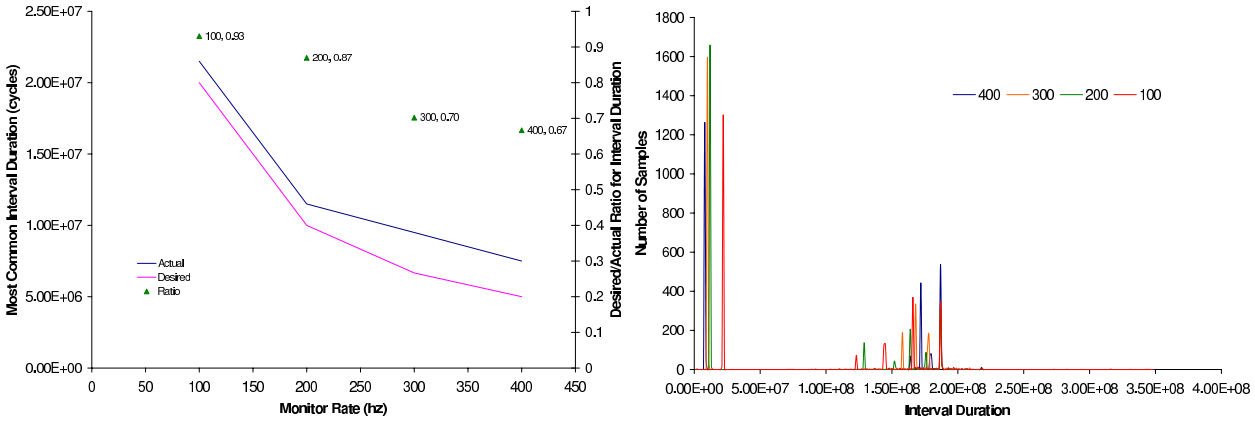
- (a) boost frequency only when our model indicated that favorable conditions for doing so for a duration significantly longer than one interval
- (b) boost the frequency only when favorable conditions for doing so exist for multiple number of samples

If the samples were uniformly distributed, we should see the same impact from both alternatives if the number of samples used for (b) matched an equivalent boost-delay duration for (a). Figure 7 shows the power-performance trade-offs among the three alternatives: single interval (6.7ms at 150 samples per second) boost window, 67ms boost window, and an equivalent 10 sample boost window for two of the power-limited applications - `galgel` and `perlbmk` (there was no noticeable overshoot for the other applications even with the single interval model). We see that the higher-samples-based option does a much better job at avoiding the overshoots - most clearly with `galgel`. Investigating the cause for this led us to discover the Windows timer behavior presented earlier. The longer duration samples hide the more recent increasing activity information by combining it with past low activity phase. Based on such a sample, the implementation expects a low activity phase and boosts the frequency, not realizing that the more recent high activity phase (and high power consumption phase) has begun. Increasing the boost-decision window to 67 ms (from 6.7ms) does not overcome the averaging out behavior by the long samples. However, waiting out the long samples by using a sample-count-based boost window of 10 samples dramatically reduces the amount of overshoot. This is the approach used for all our results discussed earlier.

While we tackled this infrastructure limitation by a short-term solution of an algorithm change, we did so at the cost of the opportunity to boost performance for other applications not similarly power-constrained. A better long-term solution would be to fix the underlying cause of the timer variability. Alternatively, we could adopt an out-of-band implementation for our control algorithms.

5.3.2 Impact of Power Model on DPO Performance

Another key component of our DPO implementation is the activity-based power prediction model we use. While our single-counter based model is fast and appears very effective for our purposes from the results, there is still room for improvement. Figure 8 shows some of the issues related to the model. It depicts a short portion of a `galgel` run showing the measured power, estimated power, the instructions decoded rate (counter used for the power model) and the operating frequency controlled by DPO.



(a) Actual versus Desired Interval Durations

(b) Histogram of Interval Durations

Figure 6: Windows Timer Behavior

The first observation is the gap between estimated power (green) and measured power (red). For galgel the measured power almost always rides on top of the estimated power. Since our model is based on the averaged behavior of a collection of power traces and galgel is at the high-power end for such traces, this is to be expected. While we use a 'safety margin' of 0.5 Watts over and above our estimated power to make our decisions, the gap in the figure can be seen to be greater than this. A model that bridges this gap would be useful both for avoiding power excursions as well as increasing performance (when the gap is the other way). The second observation is that there are a couple of places in the trace, where the extent of a change in measured power is not suitably tracked by the estimated power. The first point occurs just before 579 seconds and the second between 580 and 581. While at the first point, the model underestimates the impact of the change at the second point it appears to react too soon to the applied frequency change unlike the measured power (one could adjust for the latter more effectively in the model if detected reliably). A thorough evaluation of the impact of the model on DPO is beyond the scope of this paper and not discussed because of the extent of effectiveness of our current models. However, we have identified potential shortcomings and are currently investigating approaches to improve the fit of the basic models and also incorporate real-time power feed-back for online model refinements.

6 Conclusions

We propose the use of *Dynamic Processor Overclocking* for boosting application performance in power-constrained systems. Dynamic Processor Overclocking (DPO) uses the DFVS capabilities of modern processors to obtain increased performance by increasing the processor frequency when it is possible to do while

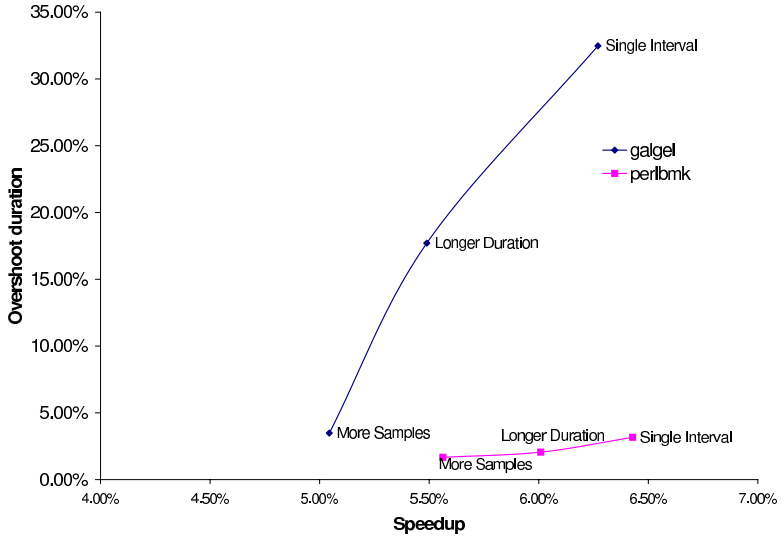


Figure 7: Algorithm Alternatives to Combat Overshoot from Timer Behavior

still meeting the power constraint. In contrast to existing power management solutions where DFVS is used solely for energy savings, we provide a DFVS-based solution focused on performance improvement in power-constrained environments. We designed a prototype implementation for DPO using facilities common in recent systems. We use the implementation to evaluate the benefits of DPO. We evaluated the benefits of DPO with the SPEC CPU2000 benchmarks on a Pentium M-based platform for a frequency range of frequency of 11.1%, from 1.8 GHz to 2 GHz. The applications exhibited a wide range of performance improvements right up to 11%. With analysis using the processor performance counters, we identify the application characteristics that determine the extent of performance improvement. We also show the relationship between the level of power constraint and the performance benefit. We quantify the value of DPO in alleviating the performance limitations imposed by power constraints and our analysis provides an approach for estimating the benefits of this solution for new workloads. Lastly, we identify the implementation issues that affect DPO performance and suggest potential remedies.

References

[1] International Business Machines Corporation, “PowerPC 405LP Embedded Processor Product Brief.” www.ibm.com, January 2003.

[2] Intel Corporation, “Intel XScale microarchitecture technical summary.”

[3] Intel, “Pentium M processor on 90 nm process with 2-MB L2 cache datasheet.” http://www.intel.com/design/mobile/datashts/302189.htm, Jan. 2005.

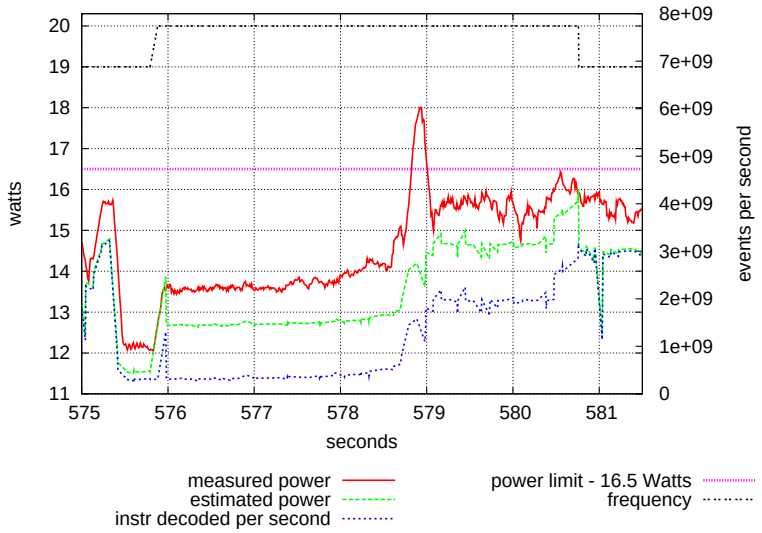


Figure 8: Estimated Power Compared to Measured Power for `galgel`

[4] N. J. Rohrer, et al, "PowerPC 970 in 130nm and 90nm technologies," in *IEEE International Solid-States Circuits Conference 2004*, February 2004.

[5] C. Lichtenau, et al, "PowerTune: Advanced frequency and power scaling on 64b PowerPC microprocessor," in *IEEE International Solid-States Circuits Conference 2004*, February 2004.

[6] C. McNairy and R. Bhatia, "Montecito: A dual-core dual-thread itanium processor," in *IEEE International Solid-States Circuits Conference 2005*, March-April 2005.

[7] C. Poirier, R. McGowen, C. Bostak, and S. Naffzigr, "Power and temperature control on a 90nm itanium-family processor," in *IEEE International Solid-States Circuits Conference 2005*, March-April 2005.

[8] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU," in *Proceedings of the ACM International Conference on Mobile Computing and Networking*, pp. 13–25, November 1995.

[9] T. Pering, T. Burd, and R. Brodersen, "Dynamic voltage scaling and the design of a low-power microprocessor system," in *Power Driven Microarchitecture Workshop, attached to ISCA98*, June 1998.

[10] D. Grunwald, P. Levis, K. Farkas, C. Morrey, and M. Neufeld, "Policies for dynamic clock scheduling," in *Proceedings of the Symposium on Operating System Design and Implementation*, October 2000.

[11] J. Lorch and A. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Proceedings of the ACM SIGMETRICS 2001 Conference*, June 2001.

[12] K. Flautner and T. Mudge, "Vertigo: Automatic Performance-Setting for Linux," in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 105–116, December 2002.

[13] B. Brock and K. Rajamani, "Dynamic power management for embedded systems." IEEE International SOC Conference, September 2003.

[14] M. Annavaram, E. Grochowski, and J. Shen, "Mitigating Amdahl's law through EPI throttling," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA-32)*, 2005.

[15] A. K. Uht and R. J. Vaccaro, "TEAPC: Adaptive computing and underclocking in a real PC," in *Proceedings of the 1st Watson $P = ac^2$ Conference*, Oct. 2004.

[16] R. J. Minerick, V. W. Freeh, and P. M. Kogge, "Dynamic power management using feedback," in *Proceedings of Workshop on Compilers and Operating Systems for Low Power*, pp. 6–1–6–10, September 2002.

[17] W. Felter, K. Rajamani, C. Rusu, and T. Keller, "A performance-conserving approach for reducing peak power consumption in server systems," in *Proceedings of the 19th ACM International Conference on Supercomputing*, June 2005.

[18] N. Kappiah, V. W. Freeh, D. K. Lowenthal, and F. Pan, "Exploiting slack time in power-aware, high-performance programs," in *IEEE/ACM Supercomputing 2005 (SC—05)*, November 2005.

[19] M. E. Femal and V. W. Freeh, "Boosting data center performance through non-uniform power allocation," in *Proceedings of the Second International Conference on Autonomic Computing (ICAC)*, pp. 250–262, June 2005.

[20] F. Bellosa, "The case for event-driven energy accounting," Tech. Rep. TR-I4-01-07, Friedrich-Alexander-Universitat Erlangen-Nurnberg, 2001.

[21] W. L. Bircher, M. Valluri, J. Law, and L. K. John, "Runtime identification of microprocessor energy saving opportunitites," in *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2005.

[22] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *36th Annual ACM/IEEE International Symposium on Microarchitecture*, December 2003.

[23] G. Contreras and M. Martonosi, "Power prediction of intel xscale processors using performance monitoring unit events," in *2005 International Symposium on Low Power Electronics and Design*, August 2005.

[24] K. S. Kyeong-Jae Lee, "Using performance counters for runtime temperature sensing in high-performance processors," in *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, April 2005.

[25] A. Weissel and F. Bellosa, "Process cruise control: Event-driven clock scaling for dynamic power management," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002)*, pp. 238–246, October 2002.

[26] R. Kotla, S. Ghiasi, T. W. Keller, and F. L. Rawson, "Scheduling processor voltage and frequency in server and cluster systems," in *Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, April 2005.

[27] S. Ghiasi, T. W. Keller, and F. L. Rawson, "Scheduling for heterogeneous processors in server systems," in *Proceedings of the International Conference on Computing Frontiers (CF 2005)*, May 2005.

[28] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, "Activity-based models for online power and performance estimation enabling dynamic power management," Tech. Rep. RC, IBM Research, Jan. 2006.

[29] Radisys Corporation, "Endura LS855 Product Data Sheet." http://www.radisys.com/oem_products/ds-page.cfm?productdatasheetsid=1158, Oct. 10 2004.

[30] Intel Corporation, "IA-32 intel architecture software developer's manuals, volume 3." Document Number: 25366816.

[31] Standard Performance Evaluation Corporation, “SPEC CPU2000 v1.2,” Jan. 2 2002. http://www.spec.org/cpu2000/.

[32] R. Kotla, A. Devgan, S. Ghiasi, T. W. Keller, and F. L. Rawson, “Characterizing the impact of different memory intensity levels,” in *Proceedings of the Seventh Annual IEEE International Workshop on Workload Characterization (WWC-7)*, October 2004.

[33] J. Rubio, K. Rajamani, F. Rawson, H. Hanson, S. Ghiasi, and T. Keller, “Dynamic processor overclocking for improving performance of power-constrained systems,” Tech. Rep. RC23666, IBM Research, July 14 2005.