

A Performance-Conserving Approach for Reducing Peak Power Consumption in Server Systems

Wes Felter, Karthick Rajamani, Tom Keller
IBM Austin Research Lab
11501 Burnet Road
Austin, TX 78758
{wmf,karthick,tkeller}@us.ibm.com

Cosmin Rusu
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260
rusu@cs.pitt.edu

ABSTRACT

The combination of increasing component power consumption, a desire for denser systems, and the required performance growth in the face of technology-scaling issues are posing enormous challenges for powering and cooling of server systems. The challenges are directly linked to the peak power consumption of servers.

Our solution, *Power Shifting*, reduces the peak power consumption of servers minimizing the impact on performance. We reduce peak power consumption by using workload-guided dynamic allocation of power among components incorporating real-time performance feedback, activity-related power estimation techniques, and performance-sensitive activity-regulation mechanisms to enforce power budgets.

We apply our techniques to a computer system with a single processor and memory. Power shifting adds a system power manager with a dynamic, global view of the system's power consumption to continuously re-budget the available power amongst the two components. Our contributions include:

- Demonstration of the greater effectiveness of dynamic power allocation over static budgeting,
- Evaluation of different power shifting policies,
- Analysis of system and workload factors critical to successful power shifting, and
- Proposal of performance-sensitive power budget enforcement mechanisms that ensure system reliability.

Categories and Subject Descriptors

C.4 [Performance of Systems]: Performance attributes, C.5.5 [Servers]

General Terms

Algorithms, Management, Measurement, Performance, Design.

Keywords

Power management, power modeling, processor simulation.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICS'05, June 20-22, Boston, MA, USA.

Copyright © 2005, ACM 1-59593-167-8/06/2005...\$5.00

1. INTRODUCTION

Chip-level power and thermal considerations threaten to halt the famous growth trend captured in Moore's Law. This lack of device scaling, combined with the push for increasing density both within chips (such as CMPs) and systems (such as dense server blades), have caused systems to reach the limits of existing power supply and cooling technologies. At many data centers a complete overhaul of cooling and power facilities would be necessary to accommodate the newest systems with the fastest processors.

Cooling and power supplies must be designed to accommodate computers' peak power consumption; thus reducing peak power consumption mitigates power and cooling limitations. Today, peak power consumption of computers is often estimated statically with spreadsheets – system designers estimate peak power consumption of individual components and add them up to arrive at the peak consumption of the system. This results in over-provisioning, since no system can simultaneously utilize all of its components at their peak levels. Such over-provisioning is inefficient because it ignores the variability of power consumption. Again, because current systems are operating close to the limit of supply and cooling resources, improving efficiency of their utilization is critical.

Power shifting is our solution to improve system efficiency by realizing better utilization of supply and cooling resources. In contrast to current approach of static component power allocations, power shifting dynamically distributes power among components using workload-sensitive policies. This system-level strategy allows power budgets to be reduced with minimal performance impact on many workloads, or alternately, allows a system to extract more performance from a particular power budget.

In recent years, a variety of circuit-level and microarchitectural techniques have been proposed to reduce power. However, there has been little focus on system-level power control. Exceptions are the ECOSystem [1,2] and Nemesis [3] operating system proposals that expose power as a first-class resource to be managed and scheduled by the operating system, but neither proposal allows the system power budget to be reduced. EPI Throttling [4] comes closest to our work – it focuses on peak power but using different mechanisms.

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

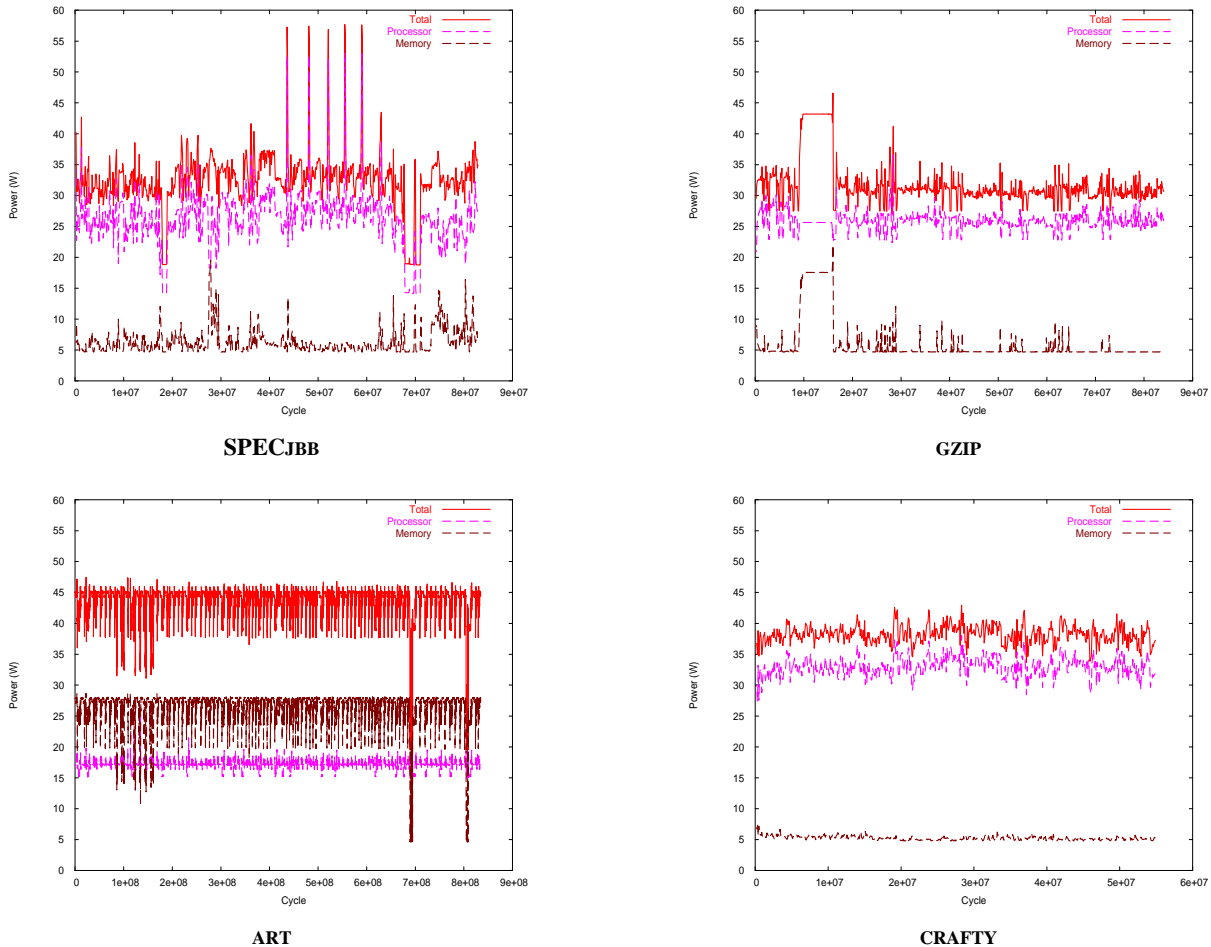


Figure 1. Simulated processor and memory power vs. time for different applications.

2. POWER SHIFTING

Power shifting dynamically divides a system’s power budget among its components in a way that attempts to maximize performance. Conceptually, power shifting determines which components need more power, and shifts power towards those components. This is accomplished by dynamically re-distributing the total system budget among the components based on the required activity at each component. Once each component has been assigned a power budget, some mechanism must enforce that budget - we use *threshold-based throttling* (described later) for this purpose. Power shifting maintains the invariant that the dynamic component budgets always add up to exactly the system power budget; allowing the sum of the dynamic component budgets to exceed the system budget could cause power supply failure and using less than the whole system budget wastes capacity, potentially decreasing performance.

2.1 Managing a Processor-Memory System

In this paper, we consider single-processor systems where we jointly manage the power of the processor and main memory. We focus on these two components as they are the most significant consumers of power in server computing systems, with memory often outstripping the processors in a fully-configured system[5]. Further, in most systems, processors and memory are already part of a single cooling domain and the same or closely located power

domains. So it does not require substantial design changes to implement power shifting for them.

We have examined a variety of policies for setting processor and memory power budgets. In our system, the power consumption of both the processor and memory is well correlated with its activity – in other words the components can trade off power and performance. The ideas we discuss and evaluate are extensible to any system where the components support this kind of power-performance tradeoff.

2.2 Workload Characteristics That Motivate Power Shifting

Power shifting is motivated by two key observations:

1. System and component activity (and consequently power consumption) varies significantly with workload.
2. Multiple components of the system are not simultaneously fully utilized.

Figure 1 shows the simulated processor and memory power consumption over time for a few example applications. There is significant variation between applications and variation over time within some applications. This means that a static estimation of component budget is less likely to match actual utilization than a run-time observation of actual activity/consumption.

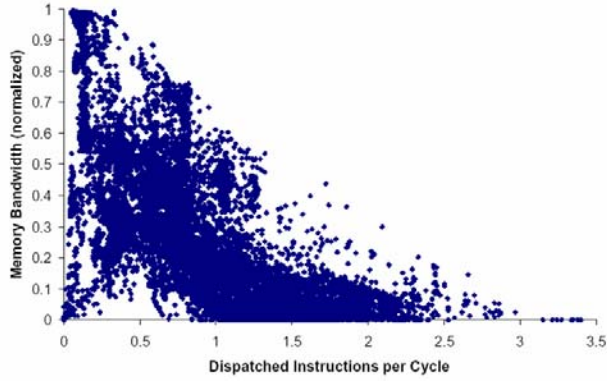


Figure 2: Component Utilization: Memory Bandwidth vs. Processor Dispatched Instructions/cycle

Figure 2 shows the correlation between processor and memory activity of all the workloads we studied (the data for these plots are from simulations described in more detail later); each point represents a 100K-cycle interval. Not surprisingly, there are no points with simultaneously high processor and memory utilization. **Figure 3** shows the processor and memory power consumption for those same intervals. Adding the peak processor power of 53W to the peak memory power of 29W produces a worst-case system power of 82W. However, the highest system power actually observed is much lower – only 58W. This leads us to the conclusion that one might be able to make use of a smaller power budget if the budget was dynamically partitioned amongst the components based on actual activity, rather than statically partitioned based on the expected maximum component-wise power consumption.

2.3 System Characteristics and Power Shifting

In this section, we discuss another important facet of power shifting – variation in power consumption with component activity. Knowing the exact relationship between the component activity and its power consumption is critical for predicting consumption and distributing the budget among components.

2.3.1 Activity-dependent power consumption

Figure 4 shows the correlation of processor power consumption with the number of dispatched instructions per cycle (plotted for 100K-cycle interval observations) for our entire evaluation

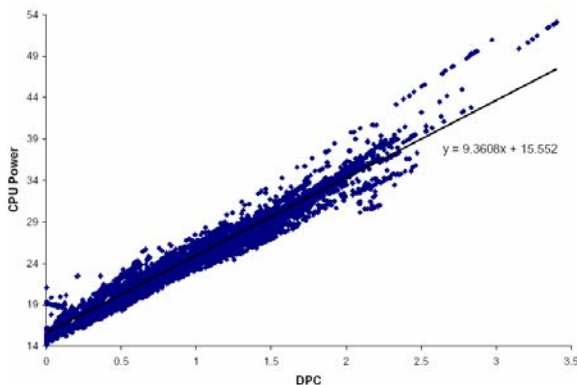


Figure 4. Processor Power (W) vs. Dispatched Instr./cycle

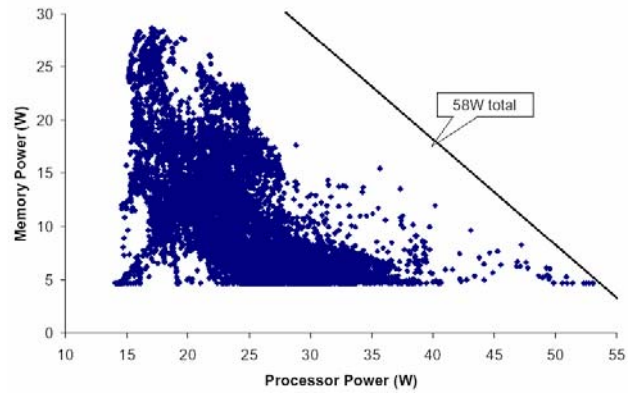


Figure 3: Component Power Consumption: Memory Power vs. Processor Power

workload set of 28 applications in our simulated system. Figure 5 shows the variation of memory power consumption with bandwidth. This data bolsters our intuition that it should be straightforward to estimate processor and memory power consumption from activity and to infer the thresholds for activity regulation in order to limit component power consumption.

It is possible that the models relating activity to power consumption are more complex for other components (or even other implementations of the same components). As long as the relationship is well-understood or can be characterized empirically our techniques would still be applicable. An important point to note is that, for effective power shifting, components need to show sizable variation in power consumption with activity.

2.3.2 Power Models and Real-Time Estimation of Power Consumption

2.3.2.1 Processor Power Model

We use a POWER4-like processor model provided by the Turandot [12] simulator. Added to the base POWER4 model is extensive clock-gating support as envisioned in future processors including support for fetch gating and issue-queue adaptation [6]. Our simulation environment and simulated system is described in more detail in section 3. The presence of extensive clock gating reduces unnecessary switching, leading to a significantly higher dependence of power consumption on useful activity in the processor. In a less aggressively power-conserving design one would expect significantly higher background power consumption

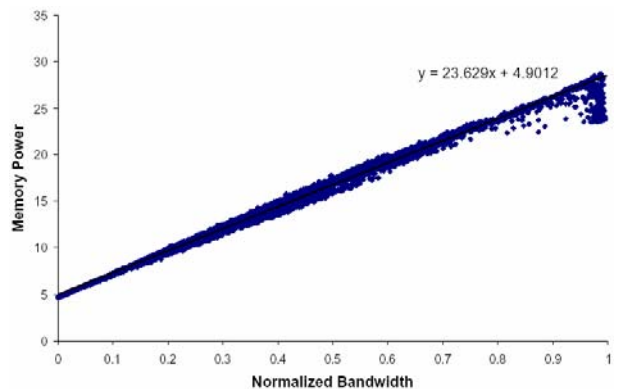


Figure 5. Memory Power (W) vs. Bandwidth

and a smaller slope for the activity-power curve. In this study, the real-time processor power estimation technique that is part of power shifting uses the linear regression of the data points in Figure 4 with an additional margin to cover the points above the regression.

2.3.2.2 Memory Power Model

DRAM power consumption is related to the nature of activity at the device and the nature of inactive modes used when the devices are idle. The exact physical layout of addresses across the ranks of DRAM devices and the DRAM page mode can significantly affect which devices are activated for a given request stream. Our memory simulator incorporates a detailed, highly parameterized model of the DRAM and memory controller.

Our memory system employs typical server-style memory interleaving (rank and bank interleaving) and page-mode (closed page-mode). We include support for the low-power *power-down* mode (CLK disable) to be utilized whenever a rank of devices is idle (there is a single DRAM cycle overhead for entering and exiting powerdown and it consumes about 90% less power than the conventional idle state). Utilizing this mode makes DRAM power consumption correlate well with real DRAM activity (reads and writes).

For the purpose of real-time estimation of main memory budget, we developed an approximate model for the power consumption. We ignore the variations in power consumption between read and write requests to simplify the model. With rank interleaving, requests to any consecutive stream of addresses would be distributed over all the ranks of devices. Given a certain request rate (aka bandwidth), one can estimate the average activity of each rank to be $1/(\text{number of ranks})$ of the given total activity. This can then be used to estimate the duty cycle for each rank. Given the parameters for the exact physical configuration of our system (number of ranks, DRAM timing and power/current data from datasheets) we then end up with a bandwidth-dependent power consumption estimate for the memory sub-system:

$$P_{mem} = \#ranks * \#devices_per_rank * V_{dram} * ((I_{active} - I_{idle}) * BW / PeakSystemBW + I_{idle}) + P_{otherchips}$$

Here, BW denotes the actual bandwidth and $PeakSystemBW$ denotes the maximum possible bandwidth with the specific DRAM speeds and organization (ignoring contention on shared buses). $P_{otherchips}$ is the additional power for registers and PLL on the memory DIMMs.

2.4 Power control via throttling

As power consumption correlates strongly with activity, we control power by limiting activity. Microprocessors employ a variety of activity regulation techniques that address thermal management by controlling power consumption – instruction (cache) decode throttling [7,8] in PowerPC, clock throttling in Pentium III, changing the *effective duty cycle* of the internal processor clocks [9] in Pentium4, and dynamic frequency and voltage scaling in PowerPC, Transmeta Crusoe, and Pentium M [10]. Brooks and Martonosi [11] provide a comparative study of many of these techniques for dynamic thermal management.

For power shifting, we use a throttling mechanism that places a limit on the number of operations performed by each component within a given interval. Once the number of operations performed

by the component reaches this threshold, no additional operations will be performed until the next interval. For the processor, we employ throttling at the instruction dispatch unit of the pipeline, placing a limit on the number of instructions dispatched within a specific interval. For the memory system, we limit the total number of memory requests serviced within an interval. These throttling schemes are quite similar to other throttling techniques previously proposed (and implemented).

With our throttling mechanisms, if N operations are permitted in a interval M , power consumption should be proportional to the ratio N/M – the average operation rate. Small values of M – 100s to 1000s of processor cycles – can provide fine-grained control while large values – 100Ks and beyond – give coarse-grained control. Power supplies have capacitance and thus an inherent time constant (usually in the millisecond range); what must be controlled is not instantaneous power but the average power over this time scale. Cooling systems have much larger time constants because of the thermal mass of heatsinks. Thus the throttling interval should be matched to the characteristics of the resource being controlled.

Thermal management solutions throttle only to avoid overload under exceptional circumstances and thus pay little attention to their performance impact. In contrast, power shifting uses throttling proactively to limit peak power and so we must consider its performance impact. Because our throttling mechanism places no limit on instantaneous performance, it allows undiminished performance as long as the total unconstrained activity in the interval is below the threshold (even if there are occasional bursts of activity), with any performance degradation happening only when the activity for the entire enforcement interval exceeds the threshold.

Like some other power management techniques, threshold-based throttling can cause high dI/dT i.e. rapid changes in current draw. We assume that the system has enough decoupling capacitance to handle high dI/dT . However, systems with inadequate decoupling can simply adjust the interval size. It is also possible to use multiple throttles with different rates and different intervals simultaneously to meet system requirements, although we do not consider it further.

2.5 Policies

All the policies presented here are interval-based, in that the activity monitoring, power estimation and allocation, and budget enforcement happen periodically at fixed intervals.

2.5.1 Proportional-by-last-interval (PLI) – the basic power shifting policy

In this policy, we estimate that the activities of the components in the next interval will be the same as in the current interval. The estimates are used to determine the required power allocation for each component for the next interval (discussed in section 2.3). Power allocations for the components are enforced by setting appropriate thresholds for the component-specific throttling techniques (discussed in section 2.4).

Power consumption of each component is broken down into an *active* (activity-dependent) and a *standby* portion. *Standby* power is consumed irrespective of the activity of the component. In this paper, we investigate re-distribution of only the *active* power because throttling only controls active power. Any allocation of

component power with our power shifting policies has to include at least the *standby* component.

Next we illustrate the working of the proportional power shifting policy. At the end of interval T_0 , the utilization of the processor and memory are observed as DPC_0 (average rate of dispatched instructions in interval T_0), and BW_0 (average memory request rate in interval T_0). Using the observed values as estimates of the component utilization for the next interval, T_1 , the component power requirements, $P_{cpu_est}(T_1)$ and $P_{mem_est}(T_1)$, are given as

$$P_{cpu_est}(T_1) = DPC_0 * C1 + P_{stby_cpu}$$

$$P_{mem_est}(T_1) = BW_0 * M1 + P_{stby_mem}$$

$C1$ and $M1$ are system-specific constants (see section 2.3.2).

For the activity-regulation techniques we use in this paper, the thresholds are the number of dispatched instructions (in the given interval) for the processor, D_{thres} ; and, the number of serviced requests (in the given interval) at the memory controller, M_{thres} . Given a total power budget, P_{budget} , the thresholds are then determined as

$$D_{thres} = DPC_0 / P_{est} * P_{dynamic} * Period(T_1)$$

$$M_{thres} = BW_0 / P_{est} * P_{dynamic} * Period(T_1)$$

where

$$P_{dynamic} = P_{budget} - P_{cpu_stby} - P_{mem_stby}$$

$$P_{est} = DPC_0 * C1 + BW_0 * M1$$

and the component-wise budgets enforced by these thresholds are

$$P_{cpu}(T_1) = D_{thres} / Period(T_1) * C1 + P_{stby_cpu}$$

$$P_{mem}(T_1) = M_{thres} / Period(T_1) * M1 + P_{stby_mem}$$

Any unused power from the previous interval is re-distributed based on how much of the allocated budget each of the components made use of in the last interval. Thus, any mismatch between the estimate-guided power allocation and actual consumption is handled naturally by the continuous monitoring and adaptation of the system.

2.5.1.1 Implications of the interval size

In the discussion above, the power estimation (and associated activity monitoring) and budget enforcement (and associated activity regulation) happen at the same periodicity – the size of the monitoring/enforcement interval. The interval size has implications for:

1. Estimation accuracy - Shorter monitoring periods imply better observation, hence, better estimation accuracy. History of past intervals is usually a better predictor over a shorter duration than longer ones.
2. Distortion of natural workload behavior - Periodic enforcement of the component-wise power budgets can cause a dampening of the natural activity of the workload. Shorter enforcement potentially imposes a stronger constraint on power consumption pattern than necessary. For some workloads, the additional dampening of activity is inconsequential to

performance. However, for most workloads, allowing temporary spikes in activity helps performance.

3. Implementation – From the implementation standpoint, large intervals can accommodate larger overheads for more software involvement and more complex estimation/prediction techniques, while smaller intervals require simpler schemes and/or hardware implementation.

2.5.2 Sliding Window

The sliding window policy extends our basic interval-based policy to utilize information from multiple consecutive intervals, thus averaging out short-term behavior. A smaller interval size is usually better for estimation accuracy and a larger interval size better for reducing the dampening of natural workload behavior. In our sliding window policy, we decouple the monitoring period from the enforcement period to exploit this dichotomy. A larger *window* encompasses multiple consecutive intervals. Instead of enforcing a budget for each interval, the sliding window policy effectively enforces an equivalent budget for each window. We use an effective enforcement window of 20 intervals by using the average consumption of the last 20 intervals as the estimated consumption for the next interval.

2.5.3 On-demand

The on-demand policy addresses the estimation inaccuracy that arises when the last interval is not a very good indicator of activity for the next interval. When the total system power is well below its budget, the on-demand policy does not enforce any component power budgets – avoiding unnecessary dampening of the natural activity of workloads from incorrect estimation of requirements for the next interval. When the system power approaches its budget, the on-demand policy behaves like the PLI policy. Since this does not proactively enforce power budgets at all times, it is possible that the consumption could exceed the desired budget for one interval. To ensure reliable operation, this requires the enforcement interval be small enough to meet the requirements of the power supply – for our results we confirm that there were no violations of the supply constraints.

2.5.4 Run-to-exhaustion (RTE)

The run-to-exhaustion policy is **not a power shifting policy**, but we include it to provide an *ideal* measure of the benefit of flexible distribution of system power among the components. It does not attempt to set component budgets at all – instead it allows the system to run normally while monitoring – on a cycle-by-cycle basis – the energy used within the interval. If any additional energy consumption would result in the average power for the interval exceeding the system power budget, the processor stops fetching for the remainder of the interval – in our modeled system there is no additional active energy consumption for the remainder of the interval for both the processor and memory.

RTE avoids any potential reduction in effectiveness from mispredictions of component utilizations (and consequent misallocations of power) and provides a comparison point for performance impact of constrained budgets. However, RTE is impractical as it relies on the exact knowledge of current system-wide energy consumption at each cycle to stop over-subscription.

2.5.5 Static Allocation

The conventional approach to power allocation is static. To create a good static allocation we look at the unconstrained component-wise power consumption of all the applications. The static

partitioning of budget is then based on the ratio of the average unconstrained component consumptions. For example, if the average unconstrained processor power consumption for all the applications is 60W and that for memory is 40W, then for a total budget of 50W the allocation for the processor would be $(50 \cdot 60 / 100 =) 30W$ and the allocation for memory would be $(50 \cdot 40 / 100 =) 20W$. Adherence to this budget is enforced by the same throttling mechanisms used by the dynamic power shifting techniques at the same enforcement intervals. Instead of throttling, it is possible (and likely more efficient) to statically reduce the processor's frequency and voltage to meet the budget, but for simplicity and consistency we do not consider it.

3. EVALUATION

3.1 Simulation Infrastructure

Turandot [12] is a trace-driven timing simulator for IBM POWER processor cores. It can be configured to support varying microarchitectures and incorporates different power models through PowerTimer [13,14]. Memsim is a simulator that models power and performance of the main memory (DRAM) subsystem [15]. It is a highly configurable simulator originally designed for modeling high-end server main memory system configurations with support for different memory interleaving, page modes, and power management policies. It uses the event-driven CSIM framework [16] and is driven by memory traces.

We extended Memsim to be driven by another simulator instead of a trace file. We integrated the Turandot/PowerTimer processor simulator with the Memsim memory simulator by replacing the fixed-latency memory model in Turandot with Memsim operating in this *component mode*. The integrated simulator reads instruction traces and generates timing and power information for both the processor and memory subsystems.

3.2 System Model

We model a system with a single 2.0 GHz PowerPC 970 processor [17] and 4GB of RAM. We modified PowerTimer's POWER4-like default power model to aggressively clock-gate many processor structures and also to support our throttling technique (discussed in 2.4) – cache sizes and other processor parameters are that of the PowerPC 970. We chose the most aggressive clock-gating style that can be realistically supported for each structure. For example, we assume that queues are only clocked when they are accessed. This is not based on any real processor but simulates one supporting state-of-the-art clock-gating techniques proposed in recent years.

The memory subsystem is 128-bit-wide 400 MHz registered ECC DDR SDRAM (PC3200) and consists of 4 ranks of 18 512Mb DRAM devices each. Cache lines are rank-interleaved to maximize parallelism. The memory controller has a 16-entry request queue and uses *queue-aware powerdown*: when a rank is idle and there are no requests for that rank in the memory controller's queue, it puts that rank into the *powerdown* state.

We do not model L2 cache power, internals and power of the U3 northbridge, power of synchronous buffer chips for DRAM, operating system code, and peripheral devices – we believe there will be no significant change to our results if these are modeled.

3.3 Workloads

In general, all of our workloads are 32-bit PowerPC instruction traces that contain only user instructions. We use

- 25 components of the SPEC CPU2000 [18] benchmark suite - traces collected on a IBM POWER system.
- SPECJbb2000 benchmark [19] – 1 warehouse using the IBM Java VM version 1.3.1 for 32-bit PowerPC and Linux 2.6 – trace collected on the Mambo full-system PowerPC simulator [20] during the steady state phase.
- STREAM memory bandwidth benchmark [21] – trace collected on Mambo.
- mload_rand - a microbenchmark that makes random memory accesses consuming very low processor power – trace collected on Mambo.

3.4 Results

3.4.1 Power Shifting versus Static Budgeting

Figure 6 shows the relative performance of the PLI approach (Section 2.5.1) compared to static budgeting for a budget of 40W. The y-axis shows the increase in execution time (slowdown) because of the budget enforcement. Power shifting, even with the simplest policy, gives much better performance than a static budgeting approach based on the average behavior across multiple applications.

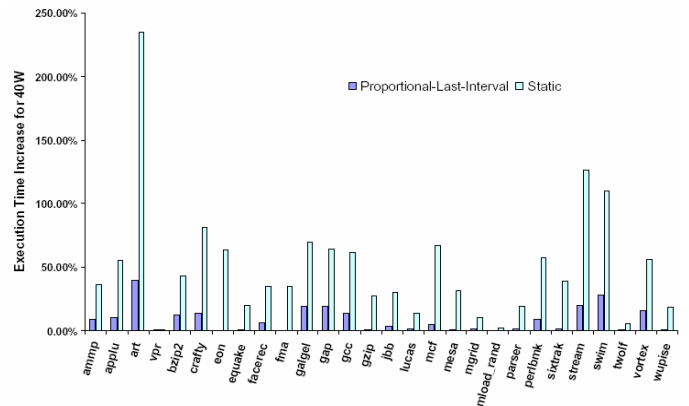


Figure 6: Dynamic Power Shifting (PLI) vs. Static Budgeting (40W) (shorter bars are better)

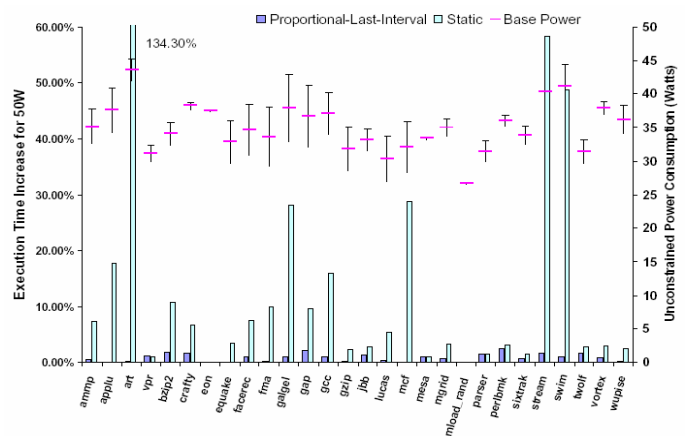


Figure 7: Dynamic Power Shifting (PLI) vs. Static Budgeting (50W) (shorter bars are better)

Figure 7 shows the results for a 50W budget and also includes the average unconstrained power consumption as dashes (with the error bars showing the standard deviation) for each application. It shows that even with a budget that is greater than the average

power consumption, static budgeting causes significant slowdown. The slowdown is greatest for applications whose behavior is very different from the average behavior – art, stream, swim, mcf, and galgel. Of these, the first three also have higher power consumption, but the slowdowns for the last two are primarily caused by the different nature of these applications from the rest. These are relatively high bandwidth applications – the static allocation results in low memory system performance that significantly inhibits the performance of these two workloads.

3.4.2 Policy comparison at 100K-cycle interval

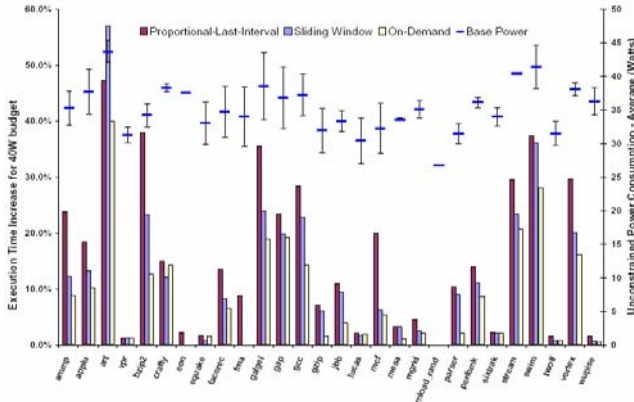


Figure 8: Performance of power shifting policies for 40W system budget (shorter bars are better)

Figure 8 shows the performance of the three power shifting policies described in section 2.5 using an interval of 100K processor cycles (50 us) with the system power budget set to 40W. Performance is measured as percentage increase in execution time over that of an unconstrained (base) system – higher bars correspond to lower performance. We chose 40W to highlight the performance differences between our policies as at 50W most workloads suffer very little slowdown. Figure 8 also contains the ‘Base Power’ data giving the average unconstrained power (measured at our enforcement criteria period of 1ms interval) with standard deviations.

3.4.2.1 Basic Power Shifting Performance

Applications with higher unconstrained power consumption – art, swim, and stream, all of whose average power consumption exceeds 40W - have larger slowdowns as they encounter more intervals that require throttling to enforce the power budget. The low average power applications – mload_rand, lucas, vpr, and twolf - have very little slowdown.

Among other applications, vortex, galgel, and gcc, have the highest slowdowns, eon and wupwise the smallest slowdowns, while applu, crafty, and gap are in between. In this range, the effectiveness of using the previous interval(s) as a predictor for the next interval is critical to the performance of power shifting. **Figure 10** shows how the unconstrained power consumption varies over time for these applications. The applications experiencing higher slowdowns all have larger temporal variability in their behavior, resulting in less accurate prediction. Unsurprisingly, the two applications with the lowest slowdowns – eon and wupwise – have very predictable behavior. The importance of predictable behavior is also reflected in the poorer performance for the relatively lower power applications ammp and bzip2 and to a lesser extent for mcf, parser, SPECJbb (jbb) and facerec; their high variability makes prediction more difficult.

3.4.2.2 Impact of Sliding Window and On Demand

Referring back to **Figure 8** one can see that the performance of *sliding window* and *on demand* are noticeably better in general, with *on demand* being the best policy for most applications. Both policies benefit applications with lower power consumption but with high temporal variability that make estimation more difficult for PLI – bzip2, mcf, galgel, ammp, fma, vortex, gcc, applu, facerec, and gap (stream is a relatively high power application that also sees significant benefit). Applications that see little improvement over the PLI power shifting approach are primarily those that see little slowdown even with PLI – vpr, mload_rand, sixtrak, lucas, twolf, and wupwise.

The value of the *on demand* policy is exemplified in the case of application phases with high temporal variability (not suitable for PLI) with variations over a longer window too (such that sliding window predictions are also not very accurate) but which have power consumption lower than the budget. These include high power applications like art and swim as well as mid-range applications like bzip2, gcc, parser, SpecJbb (jbb), galgel, and gzip. For intervals in such phases, *on demand* will not enforce throttling and so avoid a performance penalty while the *sliding window* policy will experience misprediction penalties.

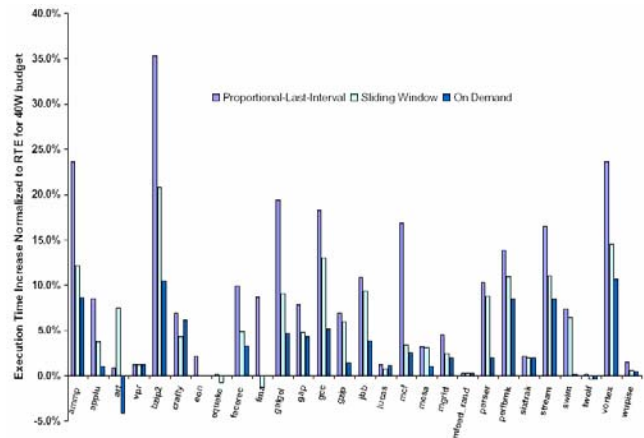


Figure 9: Performance of Power Shifting Policies Normalized to Run-To-Exhaustion (shorter bars are better)

Figure 9 shows the relative performance of the power shifting policies normalized to the performance of the Run-To-Exhaustion (RTE) ‘ideal’ execution. Since RTE never incurs any prediction errors, it provides, in some sense, a lower bound for performance

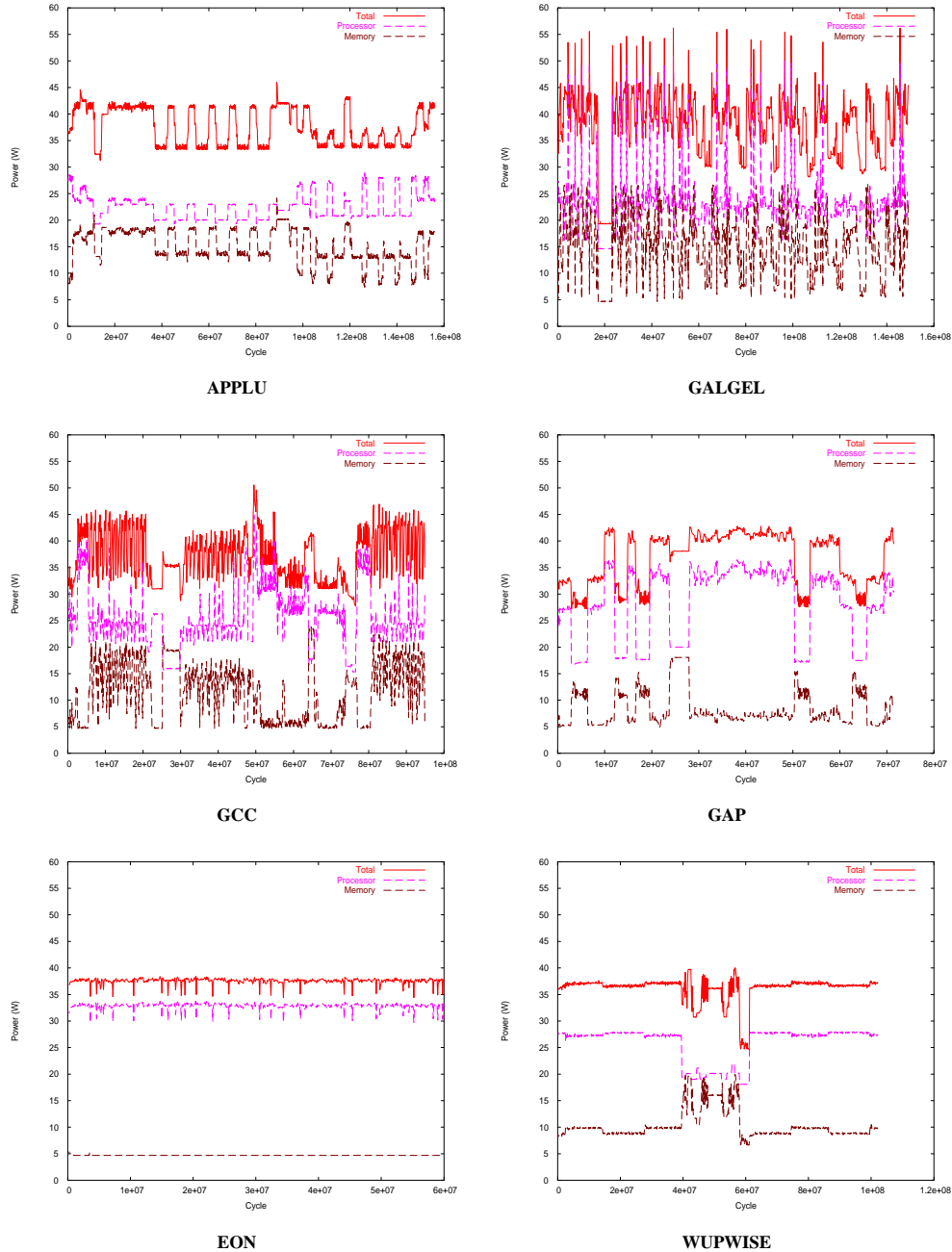


Figure 10: Variation in power consumption of selected applications (for 50us intervals).

degradation for a given budget. RTE shows the potential for improvement in the power shifting policies by eliminating prediction errors – normalizing to RTE attempts to neutralize the impact of budget limitations allowing us to focus on the limitations of the other policies. Performance of the *on demand* policy is quite close to that of RTE, with only a 3.2% average and 10.7% maximum increase in execution time.

For art, *on demand* actually provides better performance than RTE, showing that a requirement-aware power shifting approach can be more efficient in a heavily constrained system. While we expect such situations to be more common in systems with

multiple processors, its occurrence in a single-processor system is a surprise to us. To summarize, both *Sliding Window* and *On Demand* refinements improve PLI performance by either reducing the estimation inaccuracies or their impact on performance.

3.4.3 Impact of Interval Size

Section 2.5.1.1 discussed some of the implications of interval sizes on the power shifting approach. In this section, we present the performance results for the three power shifting policies at three different interval sizes – 10K cycles (5us), 100K cycles (50us), and 2M cycles (1ms) – in **Figure 11**, **Figure 12**, and **Figure 13** for PLI, *Sliding Window*, and *On demand*, respectively.

We observe that the interval sizes have some impact for all three policies, but the impact is both application and policy dependent.

For applications with highly variable behavior even at small intervals, but with relatively steady average power consumption, a small interval does not improve prediction while larger intervals will average out the variations and capture the stable behavior. This is illustrated by ammp. Smaller intervals are a better fit for applications which do not have large variations within small temporal neighborhoods, even if they have considerable variation overall e.g gap (Figure 10 shows gap’s power-time characteristics) and to a smaller extent mgrid and sixtrak. For bzip2, there appear to be two distinct phases, one that benefits from a smaller interval size (10K) and another from a larger size (2M), with the result that the intermediate 100K-cycle interval becomes the worst fit.

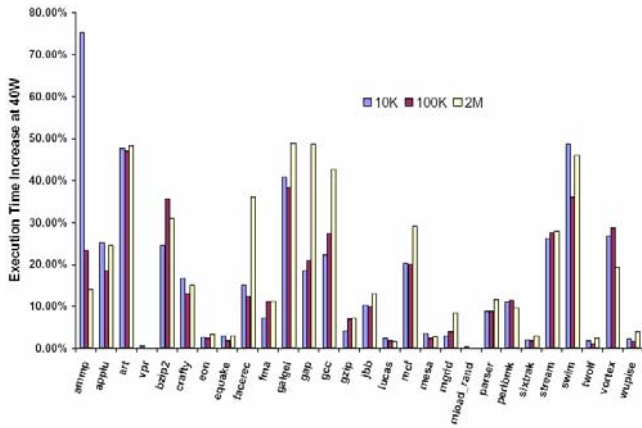


Figure 11: Performance of PLI for 10K (5us), 100K (50us), and 2M (1ms) intervals (shorter bars are better)

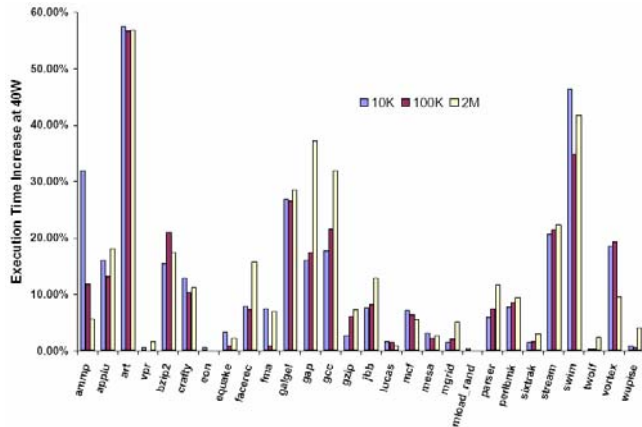


Figure 12: Performance of SW for 10K (5us), 100K (50us), and 2M (1ms) intervals (shorter bars are better)

For all the other applications, the best interval size is also dependent on the policy being employed. However, two high-level characteristics can be discerned:

1. In general, *On Demand* minimizes the disadvantage of larger intervals by reducing the impact of incorrect estimates. This results in better performance for bigger intervals. e.g. applications (with PLI or *Sliding Window*) that prefer smaller intervals but

with *on demand* prefer 100K: parser, stream, or applications that prefer 100K (with PLI or *Sliding Window*) that prefer 2M with *on demand*: swim, wupwise.

2. In general, *Sliding Window* reduces the tight enforcement problem of short intervals (since the effective enforcement interval is the larger size of window rather than the smaller size of the interval), allowing better performance for smaller intervals. e.g applications for which 100K is best (with PLI or *on demand*) prefer 10K with *Sliding Window*: SPECjbb (jbb), perlbnk.

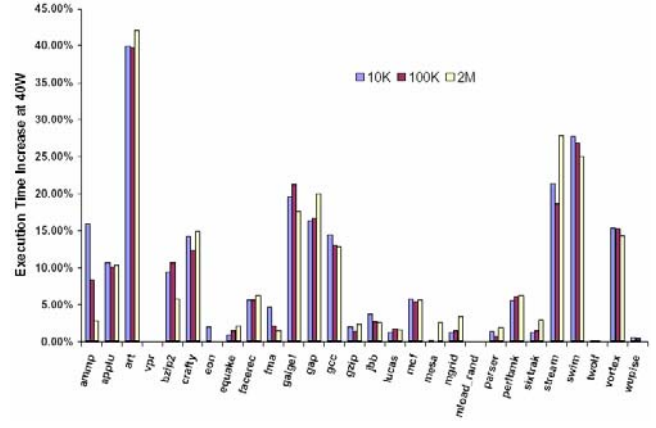


Figure 13: Performance of OD for 10K (5us), 100K (50us), and 2M (1ms) intervals (shorter bars are better)

These results show the performance impact of the interactions between application characteristics and interval. They also indicate the effectiveness of the refinements for mitigating the effects of any potential mismatch between a chosen interval size and application requirements. In addition, one could also use the insight gained to design an adaptive system that chooses interval size based on application-policy interactions.

4. DISCUSSION

We have not discussed whether power shifting should be implemented in software or hardware, and where that software or hardware should fit within the system; these issues merit significant further study.

We note that power shifting is particularly effective when either the CPU or memory is significantly less utilized than the other. There are a number of situations when more than one component in the system would be heavily utilized e.g. DMA activity when there is heavy CPU utilization by a different process, SMT processors when there is enough work for a second thread while the first keeps memory busy. In these situations, if the available power supply is inadequate to meet the demands of multiple components there is bound to be a loss in performance. This, however, is a consequence of the limited capacity of the supply relative to the workload’s needs and not of power shifting. Power shifting would still ensure reliable system operation by allocating the available power among the components in a manner that best matches the requirements at each component. A component that is more critically needed would get a larger allocation of power to better meet the execution requirements. Consequently power shifting would still provide higher performance than a workload-insensitive strategy.

In this work, we use throttling to enforce component power limits because it is simple to implement. Dynamic frequency and voltage

scaling (DVS) at the processor is an alternative that can be more efficient. Further, with increased scaling, leakage current is expected to dominate and DVS can be an effective mechanism to reduce leakage power. With this in view and with recent announcements of DVS support in server processors, we plan to incorporate DVS in place of or in addition to the throttling mechanism at the processor to further improve the effectiveness of power shifting.

5. CONCLUSIONS

We developed a novel, system-level power management technique, *power shifting*, for increasing performance under constrained power budgets. The technique employs activity monitoring and power estimation techniques to predict future power consumption, dynamic power allocation among the system components, and mechanisms to regulate component-level activity to enforce the desired power allocation. Our evaluations show that

- Dynamic power budgeting performs significantly better than static budgeting by adapting to application-specific and phase-specific power requirements.
- Inaccuracies in utilization and power requirement estimates can be effectively combated with the *sliding window* and *on demand* refinements to the basic *proportional-last-interval* policy.
- While the right interval for enforcing power budgets for minimal performance impact is application dependent, choosing the right policy can somewhat mitigate this effect.

We are currently investigating additional activity and power estimation techniques; extensions for managing a variety of systems – multiple cores and cache layers within a chip, hierarchical power shifting e.g. within a server blade chassis we can use power shifting to set budgets for individual blades and within each blade power shifting can allocate power to processors, memory, and disks; the impact of utilizing different power control mechanisms e.g. dynamic voltage/frequency scaling; and, combining real-time power measurement with power shifting to reduce power estimation errors that can arise from inadequate modeling or manufacturing and environmental variability in power consumption.

6. REFERENCES

- 1 Heng Zeng, Xiaobo Fan, Carla Ellis, Alvin Lebeck, and Amin Vahdat. ECOSystem: Managing energy as a first class operating system resource. In *Proceedings of 10th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, October, 2002.
- 2 Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy: A Unifying Abstraction for Expressing Energy Management Policies. In *Proceedings of the USENIX 2003 Annual Technical Conference*.
- 3 R. Neugebauer and D. McAuley. Energy is Just Another Resource: Energy Accounting and Energy Pricing in the Nemesis OS. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May, 2001.
- 4 Murali Annavaram, Ed Grochowski, John Shen. Mitigating Amdahl's Law Through EPI Throttling. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA-2005)*, June, 2005.
- 5 Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kister, and Tom W. Keller. Energy Management for Commercial Servers. In *IEEE Computer, Volume 36 (12)*, December, 2003.
- 6 Alper Buyuktosunoglu, Tejas Karkhanis, David H. Albonesi, and Pradip Bose. Energy Efficient Co-Adaptive Instruction Fetch and Issue. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-2003)*, June, 2003.
- 7 H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, J. Alvarez. Thermal Management System for High Performance PowerPC Microprocessors. *IEEE*, 1997.
- 8 IBM PowerPC 750FX RISC Microprocessor User's Manual version 1.01. February, 2003.
- 9 Thermal Monitoring and Protection, *IA-32 Intel Architecture Software Developer's Manual, System Programming Guide, Volume 3*, 2004.
- 10 Thermal Specifications and Design Considerations, *Intel Pentium M Processor on 90-nm Process with 2-MB L2 Cache, Datasheet*, October, 2004.
- 11 David Brooks and Margaret Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA-7)*, February, 2001.
- 12 Jaime H. Moreno, Mayan Moudgill. Turandot Users's Guide. *IBM Research Report RC 21968*, February 2001.
- 13 David Brooks, Pradip Bose, Viji Srinivasan, Michael Gschwind, Philip G. Emma, Michael G. Rosenfield. Microarchitectural-Level Power-Performance Analysis: The PowerTimer Approach. In *IBM Journal of Research and Development, Volume 47, No. 5/6*, Nov. 2003
- 14 Zhigang Hu, David Brooks, Viktor Zyuban and Pradip Bose. Microarchitecture-level power-performance simulators: Modeling, validation, and impact on design. *Tutorial at the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, December, 2003.
- 15 MEMSIM Users' Guide, *IBM Research Report RC23431*, October, 2004.
- 16 Mesquite Software, Inc. User's Guide - CSIM19 Simulation Engine.
- 17 Peter Sandon. PowerPC 970: First in a new family of 64-bit high performance PowerPC processors. presented at *Microprocessor Forum*, March 2002 (available at http://www-06.ibm.com/Hchips/techlib/techlib.nsf/products/PowerPC_970_and_970FX_Microprocessors).
- 18 Standard Performance Evaluation Corporation (SPEC). SPEC CPU2000. <http://www.specbench.org/cpu2000H>, October, 2004.
- 19 Standard Performance Evaluation Corporation (SPEC). SPEC JBB2000 (Java Business Benchmark). <http://www.specbench.org/jbb2000H>, October 2004.
- 20 Patrick Bhorer, Elmootazbellah Elnozahy, Ahmed Gheith, Charles Lefurgy, Tarun Nakra, Jim Peterson, Ram Rajamony, Ron Rockhold, Hazim Shafi, Rick Simpson, William Speight, Kartik Sudeep, Eric Van Hensbergen, and Lixin Zhang. Mambo – A Full System Simulator for the PowerPC Architecture. *ACM SIGMETRICS Performance Evaluation Review, Volume 31, Number 4*, March 2004.
- 21 John McCalpin. STREAM: Sustainable Memory Bandwidth in High Performance Computers. <http://www.cs.virginia.edu/stream/H>, October, 2004.