Microarchitecture-Directed Global Power Management for Multicore Processors

Gene Y. Wu, Yi Yuan, Dan Zhang
Department of Electrical and Computer Engineering
The University of Texas at Austin
{gwu,yyuan,dan.zhang}@mail.utexas.edu

1. INTRODUCTION

Power has become a first order concern in modern processor design. The past trend of technology scaling has provided a path to faster cycle times and increased on-chip transistor density. However, the increase in transistor density has also greatly increased power consumption. To make matters worse, smaller transistor feature sizes have caused leakage power, which was once a negligible component of power consumption, to become an increasing concern.

Power is obviously a concern in the embedded and mobile domain, where power consumption affects battery life. However, the power problem has now become a limiter in other areas as well. Increased power consumption introduces thermal dissipation challenges. It is often impractical, expensive, or even impossible to provide a cooling solution for chips running at full capacity for extended periods of time. Even when such cooling solutions are possible, it is often still desirable to reduce power consumption for monetary reasons. For example, data centers deploy processors on such a large scale that a small savings in per-chip power consumption can dramatically reduce operational costs.

With the emergence of multi-core technology, it has become necessary to manage power across multiple cores. Multi-core power management schemes offer the opportunity to dynamically adjust frequency and voltage on a per core-basis to achieve system level power reduction. Global power managers use feedback from the system to decide how to allocate power between the cores. A power manager's job is always to make a power vs. performance trade-off.

The goal of this study is to determine if microarchitectural statistics can be used effectively by a global power manager to make decisions about power allocation. We begin by observing that a fully utilized pipeline makes better use of the power it is allocated. The increased frequency allows a rarely-stalling pipeline to retire instructions faster in a given time period. Often times this means that the core is compute-bound. On the other hand, a pipeline will be poorly utilized while running a memory-bound application. The pipeline resources are potentially underutilized while the core waits for main memory. A faster clock is of little use to the core while the pipeline is stalled waiting for main memory.

A manager that is aware of which cores are currently memory bound can choose to allocate power to the better-utilized cores. These cores can make use of faster clock frequencies to retire instructions more quickly. This will improve the overall throughput of the system. The memory-bound cores will be set to lower power states. This allows power to be saved without significantly degrading performance.

Determining which cores are memory-bound is a challenge. However, we propose the use of microarchitectural statistics to allow the power manager to reason about which cores are the most memory-bound. Large numbers of L2 cache misses and reorder buffer (ROB) stalls are both symptoms of a core running a memory-bound application. Using these statistics as inputs to a global power manager allows it to determine which cores will make the best use of the additional power and which cores to reclaim power from if the chip power budget is not being met.

2. MOTIVATION AND BACKGROUND

It has been shown that global power management is a good approach for controlling the power of a multi-core processor. For a multi-core design, Sharkey et al. [9] show that trying to optimize power and performance needs to be done at the chip

level. Trying to optimize each core's power locally cannot provide as much performance per unit power.

Global power managers can try to make different power/performance trade-offs. Park et al. [8] propose a global power management scheme that makes a minimum performance degradation guarantee and then attempts to save as much power as possible. In such a scheme, keeping performance degradation within the guaranteed window is the most important concern. Isci et al. [4], on the other hand, make a guarantee about maximum power consumed. This scheme's first concern is the power budget. As long as the budget is met, the scheme will then try to optimize for maximum performance. These two examples show the different kinds of power/performance goals that a power manager can have.

Mutlu et al. [6] showed that long latency operations such as L2 cache misses will eventually fill the processor's instruction window and cause the pipeline to stall. Rather than use this observation to perform prefetching and enhance performance, we use this to find opportunities for saving power during execution.

3. MICROARCHITECTURE-DIRECTED POWER MANAGEMENT

Whereas previous global power management policies were focused on optimizing for the system's clock frequency or power consumption, our proposed policies focus on using the available power most efficiently -- in other words, striking the right balance between power and performance. We do this by examining the behavior of the programs running, in addition to the traditional way of targeting a given power budget.

3.1 Program behavior statistics

By characterizing the behavior of the active processes in the processor, we can base our power management policies on how well a particular program would respond to changes in the power state for the core on which it is running. In particular, we collect statistics that show the utilization rates for each core. If a program is making good use of the processor pipeline (i.e. few stalls and high utilization), then it would have high potential to reap significant benefits from an increased P-state, and likewise a significant performance degradation if the P-state is reduced. On the other hand, a program that is often stalling and waiting for memory operations to complete would not be able to utilize a higher P-state well, and would not significantly degrade in performance should the P-state be reduced.

Therefore, we chose to use in our power manager a metric of how memory-bound a program is. A memory-bound program is one that spends many processor cycles waiting for memory accesses, and thus has relatively low IPC. Such programs are bottlenecked by the memory subsystem, and thus would not respond well to changes in the processor P-state.

To determine how memory-bound a particular program (or phase of the program [10]) is, we identified and studied a few different metrics: L2 cache misses and reorder buffer (ROB) stalls. The intuition behind and how we use each of these metrics are discussed in the following subsections.

3.1.1 L2 cache misses

A direct measure of how many memory accesses a program makes is the number of misses in its last-level cache, which is the L2 cache in our processor model. All such misses generate at least one memory access (i.e. one to bring in the new line, and possibly another one to write back a dirty line that was replaced), which takes a few hundred processor cycles to complete in a modern-day 2-3GHz microprocessor.

If the memory access was for instructions, then the processor would have exhausted its supply of instructions, and would therefore have to insert bubbles into the pipeline until the memory request returns. During this time, the processor is not well-utilizing its execution resources.

On the other hand, if the memory access was for data, then an out-of-order processor can continue executing, assuming it can find additional instructions with no dependencies on the requested data, and as long as there is space in its ROB. However, the size of the ROB in a typical microprocessor would not be able to cover the entire memory latency [6]. Once the ROB is full, the entire processor pipeline is stalled and the processor must wait for the memory access to return. Should this happen very frequently, the program would not be able to efficiently use the processor, and would be considered memory-bound.

In order to provide a metric that is easy for the power manager to work with, we use a formula to convert the statistics we collect into a fraction in the range of 0 to 1, representing how memory-bound a particular program is. A higher number indicates the program is more memory-bound. The formula when characterizing the program using L2 cache misses is:

$$m = \frac{L2\ misses}{cycles}$$

3.1.2 Reorder buffer stalls

Another way to determine memory-boundedness is at the ROB itself, by measuring the number of cycles when it is full. The ROB can be filled up if the processor is waiting for some long-latency operation to complete. If an operation is outstanding for an extended time, it will eventually become the head of the ROB, waiting to retire, and it will block all subsequent entries from retiring. As more instructions come in, the ROB will become full and the processor would be unable to make further progress. Hence, the more cycles a processor spends with a full ROB, the more likely it is to be running a memory-bound program.

The formula to determine memory-boundedness is:

$$m = \frac{ROB\ stalled}{cycles}$$

3.2. Exploiting memory-boundedness using Amdahl's Law

In the previous section, we identified various indicators for memory-boundedness and proposed several equations for computing the memory-bound percentage of a program. The question is: how does one utilize this information to generate an efficient power management profile? To answer this question, we turn to Amdahl's Law, which is used to find the overall speedup or slowdown to a system when only one portion of the system is modified. In this scenario, the entire system includes the cores and system memory, and portion of the system that is modified is the clock speed of the cores. DVFS only modifies the frequency of the cores, whereas the latency of a system memory access stays the same. Therefore, if *m* is the portion of the computation that is memory-bound, we can write:

$$speedup = \frac{1}{m + \frac{freq_{current}}{freq_{next}}(1 - m)}$$

The above equation demonstrates that if the memory-boundedness of an application interval is close to 0, then the net speedup of the program is linear as frequency is changed. On the other hand, if the application interval is very memory-bound, then the net speedup of the program changes very little as frequency is changed. Thus, we can exploit this fact as follows: in the case of a memory-bound core, we can scale the frequency and voltage down and see negligible performance losses. In the case of a core that is compute-bound, we can attempt to scale the core up and benefit from near-optimal speedup.

3.3. The global power management framework

The power management policy is executed during a set phase in the overall global power manager algorithm. First, the global power manager allows the cores to run for one time interval. During this interval, each core collects the microarchitectural statistics previously described and measures power consumption. At the end of the interval, the power manager reads in each core's statistics and power measurement counters. At this point, the power manager policy is executed and the next DVFS state for each core is determined. The cores are then allowed to proceed. The power manager will wait until the end of the next time interval before taking any further action.

3.4. Proposed power management policies

We have devised several global power management policies that attempt to exploit the memory-boundedness property of an application. These policies are variants of the policies presented in [4], with the power manager attempting to maximize a parameter while operating at a provided power budget. However, our proposed policies optimize for performance efficiency instead of pure performance. We utilize our memory-boundedness equations and Amdahl's Law to improve performance efficiency. In addition, instead of attempting to use as much of the provided power budget as possible, our policies may choose to operate at a lower power level for that interval to save even more power.

3.4.1. Microarchitecture-Directed Power Management

This proposed scheme is not based on any of the schemes presented in [4] and is designed for maximum performance efficiency. First, the power manager checks to see if the cores over or under consumed power in the previous interval. Then it generates a power consumption estimate for each core in each of the available power states. Estimation is done by using the frequency and voltage of each of the core's available power states to scale the power measurement of the previous run interval. This allows the core to project how much power will be

saved or consumed as the core moves between the power states. The power manager also computes the memory-bound metric for each core based on the input statistics. A memory-bound threshold value, determined empirically, is used to determine which cores are highly memory-bound.

If the cores consumed less power than the budget in previous run interval, then the power manager searches the cores from least to most memory bound. As it searches the cores, those cores which are lower than the memory bound threshold (i.e. those cores not considered memory bound) are allowed to increase their power state by one for the next interval as long as the estimated additional power does not push the total power consumption above the budget. It is possible to be under consuming power while all cores are above the memory bound threshold. In this case, the manager assumes that no core would make efficient use of additional power and saves the power rather than allocating it.

If the cores consumed more power than the budget in the previous run interval, then the power manager begins searching by considering the cores that are above the memory bound threshold. The manager considers these cores from most to least memory bound. As the manager examines the cores, it drops them by one power-state and estimates how much power will be saved, if the core is not already in the lowest power state. Once the manager has iterated through all the cores above the memory bound threshold it checks if the projected power is below the power budget. If it is then the manager has finished setting each core's power state for the next run interval. If the projected power is still above the power budget, the power manager iterates again over the memory bound cores. This process repeats until the power budget is met or all cores above the memory bound threshold are in the lowest power state.

If the power manager finishes iterating over the cores above the memory bound threshold and the power budget has still not been met, then the power manager begins iterating over the cores below the threshold. The process for iterating over the cores below the threshold is the same as it was for the cores above the threshold. By searching the lowly memory bound cores second, we only lower the power of highly utilized cores when there are no other options to keep the chip within the power budget.

3.4.2. MaxEff

In addition to Microarchitecture-Directed Power Management, we have designed a power management policy that is based off the MaxBIPS algorithm described in [4]. The algorithm predicts the power and performance of each core at each DVFS frequency. Then, the algorithm uses a brute-force approach to test every possible core frequency combination that fits under the power budget. The combination with the highest net performance is chosen to be used for the following program execution interval.

However, several major changes were made. When generating the performance prediction for each core, Isci, et al. assumed that performance scales linearly with clock speed. Instead, we use Amdahl's Law to generate a more accurate prediction that takes into account the memory-boundedness of the application. In addition, instead of optimizing for performance, we optimize for an EDP (Energy-Delay-Product) metric that provides a better measurement for the performance efficiency. Finally, we observed that MaxBIPS sometimes exceeds the provided power budget due to inaccuracies in the power prediction algorithm. We alleviate this by dynamically changing the budget based on cumulative power consumption: the power budget will be lowered if previous intervals went over the target budget. Likewise, the power budget will be raised if previous intervals did not fully utilize the target budget.

4. EXPERIMENTAL METHODOLOGY

4.1 Simulation infrastructure

In order to evaluate our proposed multicore power management scheme, we looked for a simulator that could model a multicore processor system and be able to predict whole-system power. Furthermore, it would need to be able to take into account dynamically changing voltages and frequencies. Unfortunately, no existing simulator that's available for research could meet such criteria. Hence, we needed to develop our own simulation infrastructure.

We chose to modify the existing Wattch [1] power simulator, which is an expansion of SimpleScalar [2]. SimpleScalar is a software-based, cycle-accurate, microarchitectural-level simulator which includes a detailed out-of-order target processor model. The target model uses the DEC Alpha ISA and resembles a modern day single-core processor (see Table 1). The Wattch addition

Pipeline width	4
Branch predictor	2K-entry bimodal
Branch misp penalty	3 cycles
BTB	512-entry 4-way
ROB	16-entry
Load-store queue	8-entry
L1 I-cache	512-set, direct-mapped, 32-byte line, 1-cycle latency
L1 D-cache	128-set, 4-way, 32-byte line, 1-cycle latency
Unified L2 cache	1024-set, 4-way, 64-byte line, 6-cycle latency
DRAM	50ns, 64-bit
I-TLB	16-entry, 4-way
D-TLB	32-entry, 4-way
Functional units	4 int ALU, 1 int mult/div, 2 load-store, 4 FP ALU, 1 FP mult/div
CPU nominal frequency	3GHz
CPU nominal voltage	1.5V

Table 1. Target processor configuration

includes power estimation for this target processor model by modeling the energy usage of each individual microarchitectural structure, such as a cache, across each cycle.

To meet our simulation requirements, we added the ability to model multiple cores, have them synchronously run our global power management policy, and dynamically adjust the frequency and voltage of each core every interval. Because SimpleScalar only supports a single target process running on a single target processor, it would be very difficult to provide the full-system support needed for a multicore shared-memory target that supports multithreaded programs. Thus, we decided to simply run as many instances of SimpleScalar as cores in the system we are modeling, therefore restricting our evaluation to multi-programmed scenarios. To achieve this, we used a Python script as the driver for our simulator. It spawns a SimpleScalar process for each target core, controls their execution rate, and models our global power management policies. A diagram of this simulation infrastructure is shown in Figure 1.

In order to provide a synchronized point in target time at which the global power manager is activated, we use barrier synchronization for all the cores at the end of each interval when the power manager should run. Once all target cores (SimpleScalar processes) have reached the barrier, the global power manager performs its algorithm and provides the next power state to each core. Then, all cores execute until the end of the next interval. At the end of the interval, each core outputs the appropriate statistics collected during that interval, and waits for the power manager at the barrier again.

4.2 Benchmarks

For evaluating our power management schemes, we model a four-core system, where each core is as described in Table 1. All cores are assumed to be independent of one another and there are no shared components. This implies that the processor has per-core DVFS capabilities.

We picked a set of four benchmarks from the SPEC CPU2006 [11] suite -- two integer and two floating point workloads, which are described in Table 2. All four benchmarks are run with their reference inputs. For each benchmark, we will fast forward to the starting point of the most representative phase for it [7]. Each test runs until one core has reached 250 million instructions committed since the starting point. The length of these tests means that cache-warming effects can

Benchmark	Type	Description	Starting point (millions of instructions)
401.bzip2	int	compression	1528
436.cactusADM	FP	physics, general relativity	13068
450.soplex	FP	linear programming, optimization	1807
462.libquantum	int	physics, quantum computing	9974

Table 2. Benchmark description

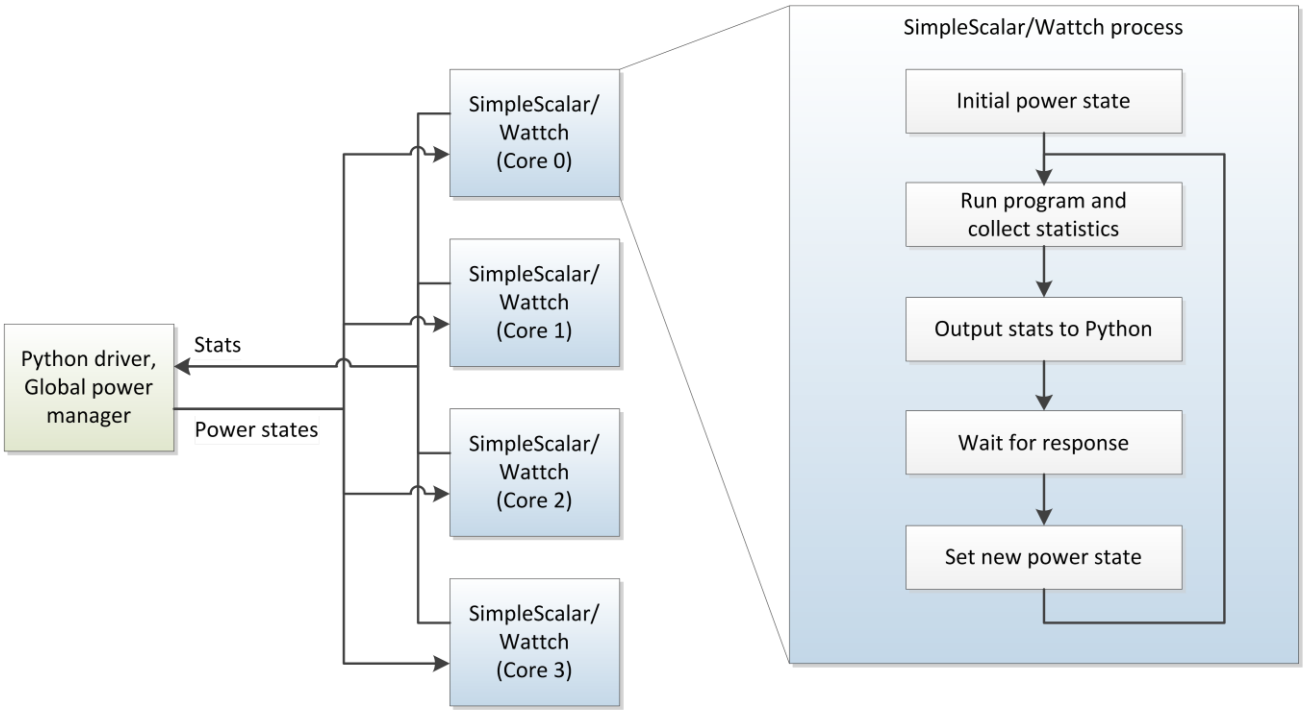

Figure 1. Simulation infrastructure

effectively be ignored.

For an objective and balanced evaluation, we did not pick any severely memory-bound or compute-bound programs. Also, because SimpleScalar models an Alpha machine, we needed the Alpha binary of any benchmark we wanted to run. We were able to acquire Alpha binaries for only a subset of the SPEC CPU benchmarks, which did not include 429.mcf – the most memory-bound of all SPEC benchmarks. This means we were not able to explore the most extreme instances of memory-bound programs. Characteristics of each benchmark are available in previous studies [3, 5].

5. RESULTS

This section compares our two proposed power management schemes to the MaxBIPS-based power management scheme described in [4]. We show results for Microarchitecture-Directed Power Management, using both ROB stall and L2 miss statistics, and the MaxEff scheme. All results are relative to a system with no power management with all cores running in the highest power state.

Figure 2 compares the proposed power management schemes in terms of power savings. Microarchitecture-Directed Power Management (MDPM) with the L2 miss metric saves the most power. Microarchitecture-Directed Power Management with the ROB stall metric saves the second most. MaxBIPS saves the least amount of power. The MaxBIPS scheme tries to stay as close to the power budget as possible, so it is natural that it saves the least amount of power. Recall that our schemes allow the cores to consume less than the power budget if there is low utilization across all cores.

Figure 3 compares the performance degradation of each scheme. Although Figure 2 showed that MDPM with the L2 stall metric was the best policy in terms of power savings, this policy does not do well in terms of performance degradation. It often underestimates the utilization of the cores, i.e. estimates the applications to be more memory-bound than they really are. This results in some cores receiving less power than they can take advantage of. Intuitively, instructions will only be blocked from committing when a memory operation waiting on an L2 miss reaches the head of the ROB. Furthermore, because of the out-of-order engine, it is not until the ROB becomes full that the pipeline stalls. In practice, not all L2 misses result in this behavior. This means that the L2 miss statistic is an overly conservative estimate of how memory-bound an application is.

MDPM with the ROB stall metric and MaxEff do better in terms or performance degradation.

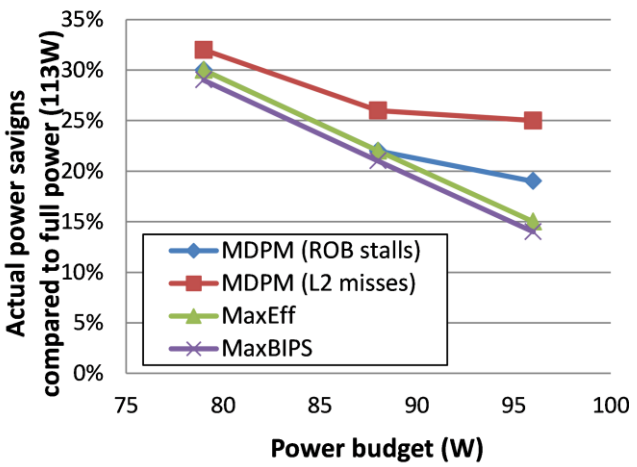

Figure 2. Power savings

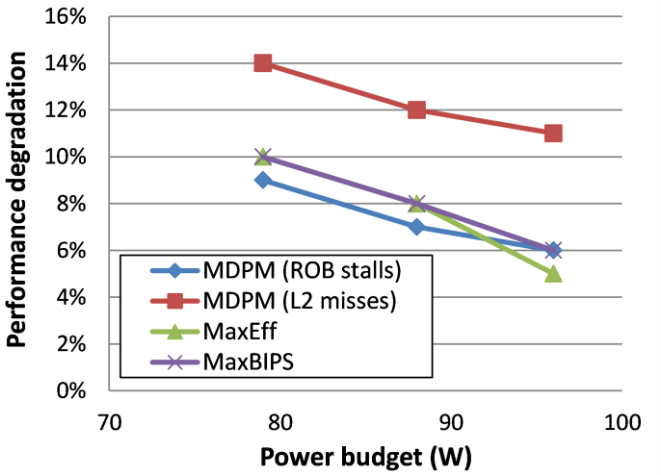

Figure 3. Performance degradation

MDPM with the ROB stall metric experiences the lowest performance degradation at power budgets of 79 and 88 watts, while MaxEff experiences the lowest performance degradation at a power budget of 96 Watts.

Figure 4 shows the power savings versus performance degradation tradeoffs between the various policies, where the lower left corner (high power savings with minimal performance loss) is optimal. MDPM with the L2 miss metric makes a trade-off that is much more skewed toward power savings. Both MDPM with the ROB stall metric and MaxEff make a more balanced tradeoff between power and performance, with MDPM having better overall results. To fairly compare the evaluated schemes, we propose a "performance efficiency" metric, defined as the power-savings to performance-lost ratio as shown in Figure 5. For the data points simulated, MDPM with the ROB stall metric has the best power-savings-to-performance-lost ratio at over 3% power savings per 1% performance degradation (over 3 on the "performance efficiency" scale). The results show that the number of L2 cache misses is not a good indicator towards memory-boundedness, performing worse than the baseline MaxBIPS implementation. MaxEff, the MaxBIPS variant that takes into account memory-boundedness, performs better than MaxBIPS in all cases but still performs worse than the MDPM implementation using the ROB stall metric.

5.1 MaxBIPS reference results

Compared to the results obtained by Isci, et al. [4] for their MaxBIPS scheme, our reference MaxBIPS implementation did not perform as well in our tests in terms of the power savings obtained for a given percentage of performance degradation. There could be several contributing factors.

First, Isci, et al. used four SPEC CPU 2000 benchmarks: 179.art, 181.mcf, 186.crafty, and 188.ammp. The benchmarks 179.art and 181.mcf are the two most memory-bound applications in the SPEC CPU 2000 suite, and 186.crafty and 188.ammp are two compute-bound applications. Therefore, because the authors used such extreme examples, the MaxBIPS scheme was easily able to trade off the power and performance of the memory-bound benchmarks and assign them to the compute-bound ones. Hence, they were able to observe very high power savings without sacrificing performance on the memory-bound programs. In contrast, we picked four moderate applications which better represent typical programs, and thus did not achieve the same evaluation results as those seen by Isci, et al.

Second, we did not use the same target microprocessor as that used by Isci, et al. While they simulated a 5-wide PowerPC processor, we used a 4-wide Alpha processor. Accordingly, they will have different power and performance behaviors.

Finally, while Isci, et al. used Turandot and PowerTimer to simulate and predict power, we relied on SimpleScalar and Wattch. Due to the inherent inaccuracies and differences between system-level power simulators, the results may differ even with identical implementations, benchmarks, and target systems.

6. CONCLUSIONS

In this paper, we exploit the characteristics of a memory-bounded program to develop high-efficiency global power management algorithms. We

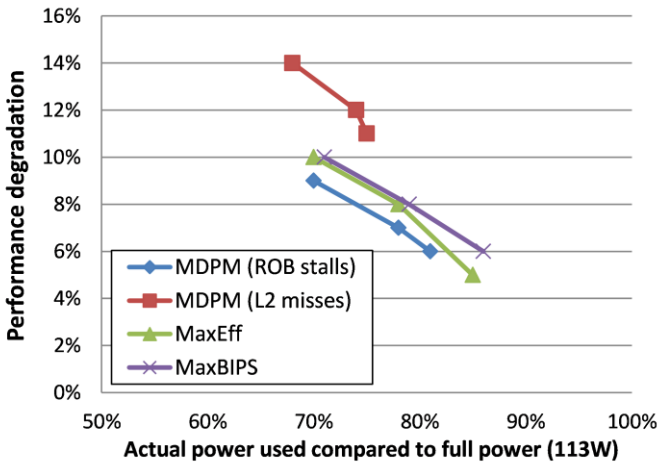

Figure 4. Tradeoff of power and performance

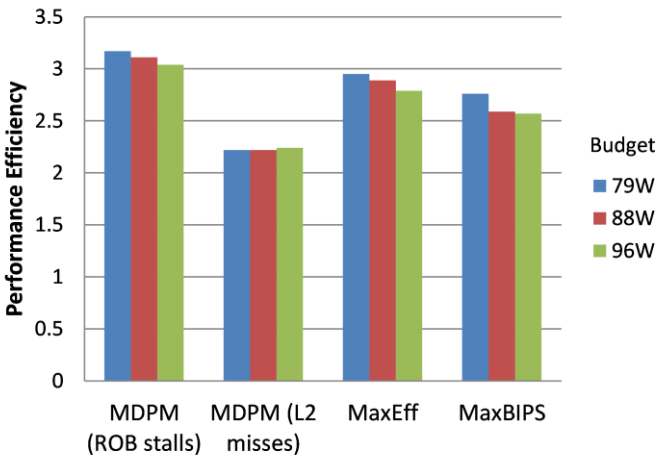

Figure 5. Performance efficiency

demonstrated that various microarchitectural statistics correlate strongly with memory-boundedness and developed algorithms to calculate memory-boundedness based on these metrics. Our experimental results demonstrate that MDPM with the ROB stall metric makes the most balanced power versus performance trade-off, performing up to 18% better than the baseline MaxBIPS policy proposed by Isci, et al [4].

REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the International Symposium on Computer Architecture (ISCA-29)*, pp.83-94, 2000.

[2] D. Burger and T. Austin. The SimpleScalar Tool Set, version 3.0. http://simplescalar.com/.

[3] K. Ganesan, D. Panwar, and L. K. John. Generation, Validation and Analysis of SPEC CPU2006 Simulation Points Based on Branch, Memory and TLB Characteristics. In *Proceedings of the SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, 2009.

[4] C. Isci, A. Buyuktosunoglu, C. Y. Cher, P. Bose, and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In *Proceedings of the 39th International Symposium on Microarchitecture (MICRO-39)*, pp.347-358, 2006.

[5] A. Jaleel. Memory Characterization of Workloads Using Instrumentation-Driven Simulation. 2007. http://www.glue.umd.edu/~ajaleel/workload/

[6] Onur Mutlu. Efficient Runahead Execution Processors. PhD Dissertation, HPS Technical Report, TR-HPS-2006-007, July 2006.

[7] A. Nair and L. K. John. Simulation Points for SPEC CPU 2006. In *Proceedings of the International Conference on Computer Design (ICCD)*, pp.397-403, 2008.

[8] S. Park, W. Jiang, Y. Zhou, and S. Adve. Managing Energy-Performance Tradeoffs for Multithreaded Applications on Multiprocessor Architectures. In *Proceedings of SIGMETRICS*, 2007.

[9] J. Sharkey, A. Buyuktosunoglu, P. Bose, Evaluating design tradeoffs in on-chip power management for CMPs. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2007.

[10] T. Sherwood, E. Perelman, and B. Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2001.

[11] SPEC CPU2006. http://www.spec.org/.