Three Scalable Approaches to Improving Many-core Throughput for a Given Peak Power Budget

John Sartori and Rakesh Kumar
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

Abstract

Recently proposed techniques for peak power management [18] involve centralized decision-making and assume quick evaluation of the various power management states. These techniques suffer from two limitations. First, they do not prevent instantaneous power from exceeding the peak power budget, but instead trigger corrective action when the budget has been exceeded. Second, while these techniques may work for multi-core architectures (processors with small number of cores), they are not suitable for many-core architectures (processors with tens or possibly hundreds of cores on the same die) due to an exponential explosion in the number of global power management states.

In this paper, we look at three scalable techniques for peak power management for many-core architectures. The proposed techniques (mapping the power management problem to a knapsack problem, mapping it to a genetic search problem, and mapping it to a simple learning problem with confidence counters) prevent power from exceeding the peak power budget and enable the placement of several more cores on a die than what the power budget would normally allow. We show up to 47% (33% on average) improvements in throughput for a given power budget. Our techniques outperform the static oracle by 22%.

1 Introduction

While power has long been a well-studied problem, most dynamic power reduction techniques, e.g., V/f scaling, clock gating, etc., exploit slack in the execution behavior of programs to reduce average power. Peak power is often left untouched. This is in spite of the fact that peak power plays a large role in determining the characteristics and hence the cost of the power supply, thermal budgeting for the chip, as well as the reliability qualification of the processor [8].

While the inefficiencies due to the lack of peak power management are substantial even for uniprocessors [18], the extent of the problem becomes much worse for multi-core processors. This is because the number of additional cores that can be powered using the absolute gap between peak power and average power keeps increasing with the increasing number of cores on a processor die. In fact, this inefficiency increases linearly with the core scaling factor [18]. If the predictions of tens to hundreds of cores on future processors are correct, effective, efficient, and scalable peak power management will become a necessity.

Recently proposed techniques for peak power management [18, 33, 5] involve centralized decision-making and assume quick evaluation of the various power management states. While these policies work well for multi-core processors (with a relatively small number of cores), they may not be suitable for many-core architectures (processors with tens or possibly hundreds of cores on the same die) due to an exponential explosion in the number of global power management states. For example, an 8-core processor with four power states (three voltage states and one off state) has 4^8, i.e., over 65 thousand possible global states that must be evaluated. Similarly, an 80-core processor like Intel's recent announcement [17], with two power states (full-power and half-power) can have 2^{80} possibilities! Even though a large number of these states can trivially be ignored by most peak power management techniques, significant subsetting still needs to be done. In this paper, we we look at three practical, scalable approaches of identifying the power management states that maximize processor throughput for a given peak power budget.

The second limitation of previously proposed techniques [18, 33] is that while these techniques attempt to limit the global power consumption, power overshoots are still allowed (overshoots trigger corrective action). The techniques that we propose in this paper attempt to prevent the power of a multi-core processor from exceeding a threshold. The prevention is provided by dynamically selecting a subset of cores and scaling down their voltages. So, the processor effectively keeps switching between different peak power management states, each with a peak power consumption below a threshold. The switches are made between states based on appli-

cation execution characteristics to maximize the throughput of the processor.

2 Related Work

Power-related architectural optimizations have been the subject of much research. Most previous work in this area has focused on the use of techniques like gating [1, 28, 16, 11], voltage and frequency scaling [27, 14, 22, 2, 26, 32, 19, 29, 39], or heterogeneity [13, 23] to reduce processor power consumption. These approaches usually target average power dissipation for uniprocessors or chip multiprocessor architectures with a small number of cores. They do not address peak power management. Other power-related work has addressed current variability [20] and thermal issues [6, 35, 9] in modern microprocessors. Some of the proposed solutions could potentially be used for peak power management.

Annavaram, *et al.* [3, 15, 4], propose an approach to power management by controlling the energy consumed per instruction in response to the parallelism of a program. They also attempt to stabilize power consumption within a fixed budget. Their work differs from ours in that we focus on *scalable* policies for peak power management.

The work that is perhaps the closest to ours is that of Isci, *et al.* [18], who propose the use of a global power manager to limit the chip-level power consumption of a multi-core processor. While their methods address the gap between average power and peak power consumption in multi-core processors, they do not prevent power from exceeding a threshold (though one may imagine using their techniques to prevent power overshoots by triggering corrective action conservatively).

Another closely related work by Juang, *et al* [21], makes a case for distributed power management instead of local power management of individual cores. We focus on investigating policies for *efficient* peak power management.

There has been a lot of peak power management work done recently in the context of servers and datacenters [25, 10, 31]. Our work focuses on multi-core architectures.

There has also been work done in the context of high-level and low-level synthesis to maximize performance for a given peak power budget, as well as the dual problem of minimizing peak power for a given performance. Our techniques work at the system architecture-level and are, therefore, orthogonal.

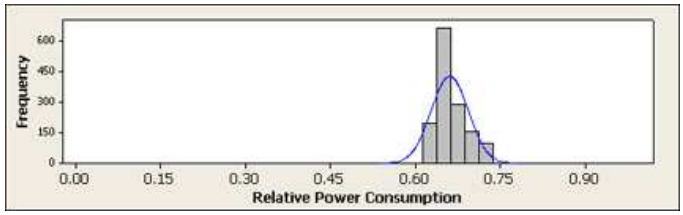

Figure 1. On average, a multicore processor consumes only a fraction of its maximum rated power.

To the best of our knowledge, this is the first work on mapping a multi-core system-level peak power management problem to the proposed algorithmic approaches.

3 Throughput Benefits of Peak Power Management

This section discusses our proactive approach to peak power management that enables the placement of several more cores on a processor die than what the power budget would allow for, thereby substantially increasing processor throughput. We also discuss scheduling policies that enhance the throughput even further.

3.1 Integrating More Cores than Allowed by the Power Budget

There is often a sizable gap between the average power and peak power of a multi-core processor. Figure 1 shows the distribution of power consumption for a 9-core chip multi-processor (CMP) as a percentage of peak power for a set of workloads that we studied (details in Section 5). On average, the processor consumes only 66% of its maximum rated power. However, the processor and the power supply must be designed to supply the peak power and rated to handle this load. In theory, we should be able to add approximately 50% more cores *running at full power* and still remain below the peak power, on average.

Now consider an architecture (Figure 2) with peak power management that has several more cores on the die than the baseline processor. The average power of the new architecture is just below the peak power of the baseline processor while a peak power management mechanism prevents the new architecture from exceeding the peak power of the baseline (even though the processor contains several more cores than the power budget would normally allow). The peak power management mechanism bounds the processor power by intelligently scaling down the power for a subset of

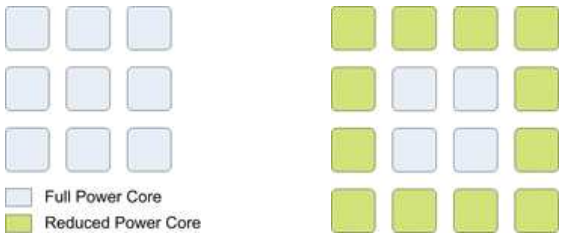

Figure 2. Power-Equivalent Processor Configurations.

cores. Power can be scaled down through the application of V/f scaling, clock gating, or power gating. The throughput of this architecture is higher than that of the baseline processor, due to the increased number of cores.

In this paper, we employ V/f scaling to limit the maximum power consumption on a core. Each core is assumed to be scalable independently. A centralized power manager is assumed like in [18].

3.2 Intelligent Core to Power State Mapping to Maximize Throughput

In this section, we describe the various baselines against which we compare our proposed approaches.

3.2.1 Static Mapping

We considered two static baselines – *random static* and *static oracle*. For a given processor configuration, the *random static* scheduler arbitrarily selects the cores that will be scaled, or equivalently, the cores that will receive full power.

Static oracular scheduling (similar to what was used in [24]) requires foreknowledge of the behavior of applications in various power states and employs a metric called weighted speedup (WS) [36]. WS expresses the throughput of an application running at full power relative to the throughput of the same application running in a reduced power state. A large WS value indicates that the performance of an application deteriorates rapidly as power is decreased. Conversely, an application with a WS value close to one can run in a reduced power state with very little performance degradation. The static oracular mapping is produced by assigning power states to processor cores such that WS is maximized. Those with the highest WS are assigned to a full power state, and those with lower WS values are allocated reduced power states.

We also evaluated a static configuration in which all cores are scaled to the lowest power state. This configuration – referred to as *all scaled* – maximizes core integration for a given power budget.

3.2.2 Dynamic Mapping

Static power management policies cannot react to the changing behavior of applications and therefore do not provide the most efficient processor power mappings.

We used the previously proposed MaxBIPS policy [18, 5, 33] as our centralized dynamic baseline. MaxBIPS, proposed by Isci *et al* [18], predicts the corresponding power and BIPS values for each possible power mode combination. Afterward, it chooses the combination with the highest throughput that satisfies the current power budget.

MaxBIPs was presented as a peak power management solution for a four-core multi-core processor and relies on a global arbiter/scheduler to make power management decisions for each core on the processor. While MaxBIPS works well for the four-core case [18, 33, 5], the technique may not be scalable for a large number of cores, as confirmed in Section 6. Secondly, while MaxBIPS attempts to limit the global power consumption, it allows power overshoots and triggers reactive correction for these overshoots.

The three dynamic techniques proposed in the next section prevent power overshoots and are targeted towards architectures with a large number of cores.

We also considered *sorted WS*. Under this policy, when power scheduling is triggered, each thread is evaluated in the available power states. The WS of each thread is calculated, and the threads are sorted in order of decreasing WS. Threads with high WS values are alotted full power while threads at the low end of the list are assigned low power states.

The benefit of *sorted WS* over *MaxBIPS* is that not all possible schedules are evaluated, and hence the time overhead of making a decision is relatively small. This benefit comes at the expense of making potentially worse decisions.

4 Three Scalable Techniques for Maximizing Throughput

Naïve methods of searching for an efficient global state (i.e., core to power state mapping for each core) have a time complexity of $O(n^2)$ or worse. These techniques rely on tactics such as exhaustive searching and sorting – tactics that will not scale well as the search space expands. We consider three canonical techniques that reduce the amount of time required to make a peak power management decision as well as the total number of decisions to be made during the decision space search. These

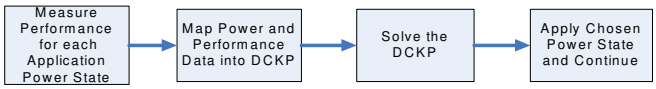

Figure 3. Knapsack-based arbitration reduces the search time for peak power management.

techniques directly promote the scalability and efficiency of our peak power management methods, making them suitable for many-core architectures.

4.1 Modeling Peak Power Management as a Disjunctively Constrained 0-1 Knapsack

The first technique is mapping the task of power management to an algorithm that runs in linear or sublinear time. One such algorithm that conforms nicely to the problem of peak power management is the disjunctively constrained 0-1 knapsack problem (DCKP) [30]. Figure 3 describes the process of power arbitration using the knapsack approach.

Knapsack problems operate on a set of items, each with fixed profits and weights. The objective of the knapsack problem is to fill a knapsack of limited capacity in such a way that the profit of the carried items is maximized. The DCKP is a special version of the knapsack problem in which the items are divided into classes. When filling the knapsack, one and only one item from each class must be selected.

The task of power state selection maps seamlessly onto the DCKP. The items to be chosen from are the applications that are currently running in the system. These items are naturally divided into classes based on the number of cores in the processor. Within each class, the items represent an application running on a core in each of the possible power states. Each item has a profit and weight represented respectively by the throughput and power consumption of the application for each power state. Finally, the capacity of the knapsack is characterized by the peak power of the processor. Thus, solving the DCKP corresponds to choosing the power state of each core to maximize the overall system performance while ensuring that the power consumption of the processor does not exceed the maximum budget. Due to the characteristics of the peak power management problem as mapped to the DCKP (specifically, the fact that the weights are non-negative integers) and the efficient partitioning techniques employed by the DCKP [30], solving the DCKP is an $O(n)$ problem in this case (though in a general, it is not).

Since the maximum power consumption of a core in a given power state is constant, the weight of each item is known a priori. However, the performance of an application changes, sometimes drastically, over the course of execution and therefore must be sampled prior to the selection of a new power state. Intuitively, more power states available to an application should result in a solution that is closer to optimal. However, as the number of power states increases, so does the evaluation time required to determine the profits and to a lesser degree the complexity of finding the optimal solution. Hence, the number of power states should be carefully chosen to balance this tradeoff and produce the highest throughput.

Note that the above algorithm runs in the power manager and requires no special hardware support (other than performance counters).

4.2 Mapping Peak Power Management to a Genetic Search Problem

An alternate way to increase the efficiency of making decisions about the best global state (that maximizes processor throughtput for a given power budget) is to treat the subset selection problem as a heuristic search problem. The goal of a heuristic search problem is to prune the search space based on certain properties of the search algorithm as well as the search space. Although the number of possible power mappings quickly escalates relative to the number of cores, a search technique can employ feedback gathered during the search to refine the search process and intelligently explore the search space.

One search method that has been applied to a wide range of problems is the genetic algorithm [12]. The genetic algorithm intelligently investigates the search space, selecting the candidate mappings that perform the best to be archetypes in the creation of new mappings. Mappings that perform poorly are replaced with the offspring from selected parent configurations. An aspect of randomness is also introduced into the reproduction process to more effectively cover the search space in the case that none of the members of the initial population closely match the optimal solution.

The evaluation process, depicted in figure 4, begins with the creation of random configurations to fill the initial population. These random configurations are designed to respect the power budget of the processor, since the number of processors at peak power remains constant across all mappings. After initialization of the population, each configuration is applied and evaluated with respect to a

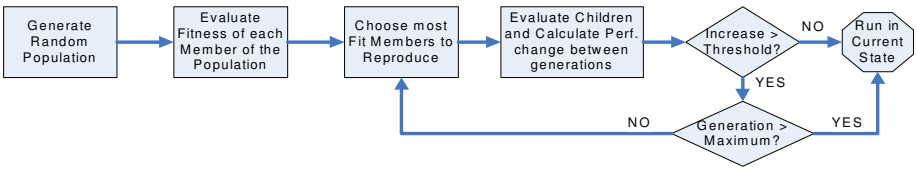

Figure 4. The genetic algorithm for peak power management traverses the search space intelligently.

fitness metric. In this case, the throughput of the processor in the current configuration determines the fitness of the power mapping. Once the evaluation of the initial population has been accomplished, the fittest members of the population are chosen to produce the next generation. Two parent configurations generate an offspring configuration through comparison of their mappings. If the parents agree on the power state of a core, then their union is passed on to the child. If the parent mappings do not agree, a decision bit is used to determine which parent trait will be passed on to the child. When the bit is set, a full power state is propagated. The bit is then toggled so that the next conflict will result in the propagation of a reduced power state. Since all mappings specify the same number of peak power cores and reduced power cores, the number of conflicts between two parents must be even. Thus, the decision bit upholds the guarantee that the resulting power state will respect the budget.

After reproduction, the newly created configurations replace the most unfit configurations, and hence the population is refined with each new generation. Evaluation and evolution continue until either the change in the best performance between successive generations falls below a threshold, indicating that the solution is close to the optimum, or some maximum number of generations is reached. At this time, the fittest configuration is selected as the new processor state and normal operation resumes.

The success of the genetic algorithm in locating a good power state may be affected by several factors. Typically, when genetic algorithms are employed, many random configurations are generated to comprise the initial population. Indeed, a larger initial population increases the likelihood that one or more of the configurations will closely resemble the optimum. However, consideration of a large initial population requires substantial time to evaluate the fitness of each member. Furthermore, since the initial members are randomly formed, they are just as likely to perform poorly as they are to perform well. Thus, evaluation of a large initial population may lead to the application of several inefficient power configurations. On the other hand, a large initial population increases the probability that some of the members are close to optimal. As such, the number of iterations required to arrive at an acceptable solution may decrease. Additionally, once the initial population has been evaluated, the focus of the genetic algorithm switches to the refinement of the existing solution. So, from this point on, the power mapping applied after each generation should only improve, even as evaluation continues.

Note that the genetic algorithm is simply a canonical example of a heauristic search technique that fits well to the fitness landscape of the peak power management problem. A complete exploration of all such approaches is beyond the scope of this work.

Note also that the above algorithm runs in the power manager and requires no special hardware support (other than performance counters).

4.3 Viewing Peak Power Management as a Simple Learning Problem with Confidence Counters

While performing an intelligent search can substantially reduce the arbitration overhead, the search space can be reduced significantly more if application characteristics are also considered while pruning the search space – specifically, if the processor knows for each application whether or not it requires full power to run without considerable performance degradation. Since, an application's behavior is not known by the processor before runtime, the processor must learn the needs of each application and choose an appropriate power state within these bounds. Figure 5 describes a learning-based approach to peak power management.

To implement learning, a set of counters – one for each core – is employed to remember how power was mapped to the core in the past. If the core consistently receives full power after each evaluation, the corresponding counter will increment until it saturates, indicating that the application on the core requires full power to run efficiently. Likewise, a core for which evaluation repeatedly selects a reduced power state will be tagged accordingly by a counter that decrements to saturation. Once an application

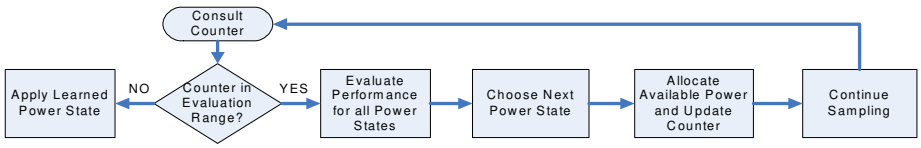

Figure 5. A learning-based approach reduces the search space for the peak power management decision by eliminating cores with static behavior from consideration.

to power state relationship has been learned, the association is remembered for subsequent evaluations. Reinitialization of the counter memory is performed periodically to allow for an unbiased evaluation of the processor power state even in the face of dynamically changing program behavior.

Counters for cores that tend to fluctuate between multiple power states remain close to the initial, mid-range value, indicating that evaluation of this core should continue in order to provide a good mapping. However, evaluation can cease for the cores that have already been associated with a particular power state. Consequently, performance losses during evaluation which result from a temporary suboptimal mapping are eliminated. Furthermore, the overhead associated with the evaluation process is significantly reduced, since decisions must be made for fewer cores. **Note that the above algorithm runs in the power manager and requires no special hardware support (other than performance counters)**.

5 Methodology

In this section we discuss the methodological details of this study – specifically the architectures that we studied, assumptions that we made about voltage/frequency scaling, our methodology for comparing processors with different numbers of cores, our workloads, and our simulation approach.

The processors evaluated in our study are chip multiprocessors (CMPs) with homogeneous cores. All cores are modeled with 65 nm process parameters. The frequency and supply voltage of each core are 3 GHz and 1.5 Volts, respectively, at full power. All cores are connected to the L2 cache banks through a matrix crossbar interconnect. To account for increased area due to additional cores and interconnections, the L2 cache size of an enhanced configuration is reduced by half with respect to the corresponding baseline configuration.

Power estimates reported by Wattch [7] were used to calculate the peak power consumption of each core in various power states. Wattch reports the maximum dynamic power consumption for a core at a given supply voltage. To calculate the total peak power for a core in a given power state, an assumption is made that the peak dynamic power consumption represents 75% of the total processor power. Thus, the dynamic power values from Wattch are scaled up to represent the dynamic and static contributions to peak power. Table 1 gives the peak power figures for several V/f scaling factors, assuming 65 nm process technology.

Scale	1.0	0.9	0.8	0.7	0.6	0.5
Power (W)	18.289	15.549	13.098	10.888	8.899	7.107

Table 1. Peak Power for V/f Scaling Factors.

The supply voltage and frequency of each core in our modeled CMPs can be controlled independently. When a core switches V/f domains, we do not assume an instantaneous change. Instead, we model a gradual transition from one V/f domain to another at a rate of 10 mV/μs. When a transition between V/f domains occurs, the cores are halted until the transition is complete for all cores. During this time, the processor is assumed to still consume power, but no performance gains are registered. Modeling a V/f transition in this manner represents a very conservative approach. Table 2 displays the penalty in cycles incurred when switching from a full power state to a reduced power state.

ScalingFactor	ΔV	Switching Time	Cycles @ 3 GHz
0.9	0.15	15 μs	45,000
0.8	0.30	30 μs	90,000
0.7	0.45	45 μs	135,000
0.6	0.60	60 μs	180,000
0.5	0.75	75 μs	225,000

Table 2. V/f Switching Penalties.

Workloads are constructed from a set of 16 SPEC2000 benchmarks. Table 3 lists the benchmarks used in workload construction along with fast-forward distances in billions of instructions.

ammp	mcf	bzip	crafty	eon	equake	galgel	mgrid
2.0	12.6	0.4	0.7	0.1	3.5	5.0	2.1
parser	swim	applu	art	twolf	vpr	vortex	wupwise
0.4	0.3	0.3	7.5	0.9	36.1	6.0	1.1

Table 3. Benchmarks and Fast-Forward Distances.

For the 16-core results that we present in this paper, we average the results over five kinds of workloads (each consisting of 16 threads). One

workload contains *all* benchmarks in equal proportions. Other workloads contain benchmarks with *high*, *low*, and *varied* sensitivity to V/f scaling. Finally, the *dynamic* workload contains benchmarks that exhibit more dynamic behavior than others.

Workloads are constructed using the sliding window approach [37]. The size of the window is equal to the number of cores in the processor.

Simulations are performed using SMTSIM [38] to simulate our various CMP configurations. Wattch is integrated into SMTSIM to gather power statistics. SMTSIM executes statically-linked Alpha binaries. After fast forwarding each benchmark for an appropriate time [34], all simulations run for 1 Billion cycles. The global power arbiter/scheduler is assumed to be implemented in software and the overheads of making power management decisions are modeled accordingly.

We performed our evaluations with respect to weighted speedup (WS) and aggregate throughput and found no significant difference in trends or analysis.

6 Analysis and Results

In this section, we demonstrate the performance benefits of peak power management for many-cores. First, we analyze the gains achieved by adding cores to the die and managing the power states of the cores using the baseline techniques. Next, we analyze the overheads of peak power management for the various techniques discussed in this paper and show that most strategies become ineffective for architectures with a large number of cores, due to their high timing overheads. Finally, we demonstrate how efficient policies can closely approximate the benefits of exhaustive centralized arbitration while substantially reducing the cost of decision-making.

6.1 Improving Throughput through Peak Power Management

Figure 6 shows throughputs for different baseline mapping policies. Results are shown relative to the average performance of a benchmark running on a reduced power core. As the results show, even static subsetting of cores can result in 16% throughput improvement when voltage is halved and by 5% when voltage is scaled by a factor of 0.8. Baseline corresponds to the 8-core processor, where all cores are running at full power. *Dynamic* corresponds to *sorted WS* and results in 18% throughput improvement over the baseline when voltage is halved.

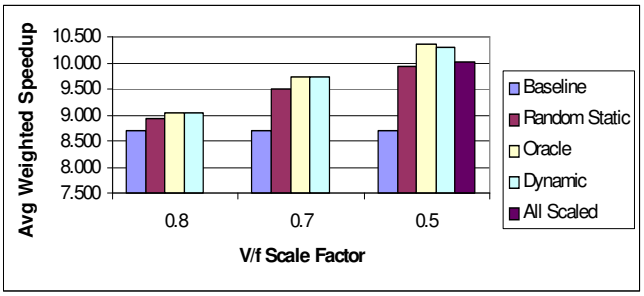

Figure 6. Performance Improvements due to Peak Power Management.

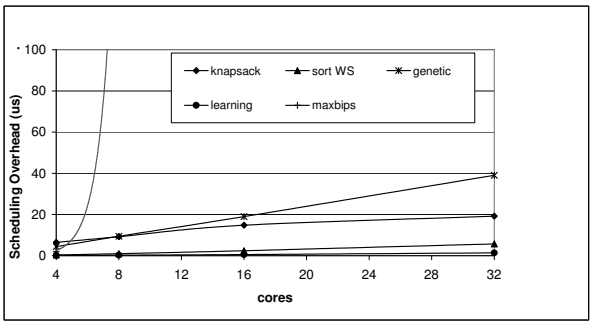

Figure 7. Timing Overhead of Making Peak Power Management Decisions for Various Policies

Sorted WS was chosen because *MaxBIPS* has prohibitively large overhead for processors with 8 or more cores (details below).

6.2 Overhead of Managing Peak Power for Many-cores

While the previous section shows the potential benefits of peak power management, the actual performance improvements depend on the overhead of making power scheduling decisions. Figure 7 shows the time overhead of making peak power management decisions in software for the various techniques for various numbers of cores. As the graph shows, MaxBIPS, which has been demonstrated as an effective global power management technique in [18, 5, 33] may be unsuitable, as is, for peak power management for many-core architectures (with unacceptably large overhead even for 8 cores) due to the exhaustive nature of its state space search. Overheads for *sorted WS* and *learning-based approach* are low for small and moderate number of cores because of significant reduction in the state space. The overheads increase linearly, however.

6.3 Performance of the Proposed Techniques

Figure 8 shows the performance of the proposed techniques for peak power management for a 16-

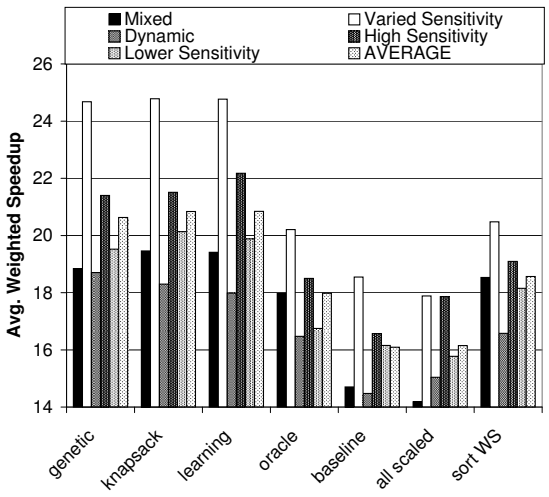

Figure 8. Performance of many-core power management policies against the baseline and the static oracle.

core CMP architecture along with the performance of the baseline processor and a static oracular scheduler for comparison. The graph also contains results for *sorted WS*. The graph does not contain results for *MaxBIPS*, as it has unacceptably high overhead for 16 cores.

As the results show, the proposed techniques significantly outperform the baseline as well as the static oracle. Consider the confidence counter-based learning-based approach, for example, in which cores are eliminated from the evaluation process as the power manager learns the optimal states for the cores. For the 16-core architecture that we studied, decisions only needed to be made for 4 or 5 cores, on average, after the first learning phase. This 71.67% reduction of cores in the decision corresponds to a 99.96% reduction of the search space, resulting in a substantial speedup in the arbitration process and marked gains in performance. On average, the learning-based approach results in a 33% performance improvement over the baseline processor and 22.1% over the static oracle. These improvements can primarily be attributed to the increased efficiency of the evaluation process in finding the optimal power mapping. Specifically, cores that have already learned their optimal power states are not subjected to a suboptimal state during evaluation. This benefit is evidenced by superior performance of the learning-based approach for the *high sensitivity* workload. When performance is sensitive to V/f scaling, temporarily suboptimal states encountered during evaluation can noticeably degrade performance.

Employing the genetic algorithm for intelligent exploration of the search space also results in reduced search time. Results show that nearly all searches terminate within two generations. Occasionally, the selection process extended for three generations. Thus, in the worst case for the 16-core model, the search space was reduced by 99.95%. On average, power management utilizing this technique generates performance gains of 30.7% over the baseline and 19.8% over the static oracle. The slight reduction in performance with respect to the best performing technique is likely due to the increased evaluational overhead of the genetic approach, which may require multiple iterations to converge on the final power mapping.

Modeling the search process as a knapsack problem results in a reduction in the computational complexity of the search algorithm, allowing for faster examination of the search space. On average, performance increased by 32.6% with respect to the baseline processor and by 21.7% over the static oracle. The knapsack approach exhibits good performance even for workloads with low performance contrast between different power states. Whereas a learning-based technique relies on performance differentiation between different power states to optimize the evaluation procedure, the knapsack approach considers the entire power budget and optimizes the power state partitioning, regardless of the variance in the the performance statistics. Note also that the knapsack approach examines thread performance in isolation (as opposed to search-based approaches that examine the overall performance of a workload). So, we expect the benefits of this approach to be reduced for highly bandwidth-sensitive workloads, where the system performance does not necessarily equal the sum of isolated performance values.

Note that the proposed approaches, even if they seem relatively complex (over random mapping or *sorted WS*, for example), outperform *sorted WS*, a simplistic dynamic mapping technique, by at least 12%, justifying the need for such approaches.

Note also that all the proposed techniques achieve roughly the same performance for our workloads. However, they still **differ in terms of the cost of making peak power management decisions (see Figure 7)**. We suspect that the learning-based approach will outperform other approaches for very large number of cores. The limitations of our current simulation infrastructure prevent us from doing that study.

7 Summary and Conclusions

This research focuses on maximizing many-core processor throughput for a given peak power budget. We demonstrated that managing peak power for many-core architectures (processors with tens or possibly hundreds of cores on the same die) may pose a significant challenge when using conventional, centralized techniques [18, 33, 5], especially if a large number of power domains are supported on the chip. The challenge stems from an exponential explosion in the global power management states. We proposed three scalable techniques for peak power management that are specifically tuned for many-core architectures. These techniques accelerate the decision process for peak power management by pruning the search space of global power management states. Over our set of diverse workloads, our enhanced architectures (using the proposed techniques) averaged 30% better performance than comparable CMPs with equivalent area and power budgets. Also, the proposed techniques performed, on average, at least 22% better than the peak power management policies designed for architectures with a small number of cores, even for 16 cores.

As the number of cores on a processor die continues to grow and as peak power management increasingly becomes a first order concern, the effectiveness of our techniques will only continue to increase.

References

[1] D. H. Albonesi. Selective cache-ways: On demand cache resource allocation. In *International Symposium on Microarchitecture*, Nov. 1999.

[2] D. H. Albonesi, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36(12):49–58, 2003.

[3] M. Annavaram, E. Grochowski, and J. Shen. Mitigating amdahl's law through epi throttling. *SIGARCH Comput. Archit. News*, 33(2):298–309, 2005.

[4] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *Proceedings of International Symposium on Computer Architecture*, 2005.

[5] R. Bergamaschi, G. Han, A. Buyuktosunoglu, H. Patel, I. Nair, G. Dittmann, G. Janssen, N. Dhanwada, Z. Hu, P. Bose, and J. Darringer. Exploring power management in multi-core systems. In *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pages 708–713, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[6] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *HPCA '01: Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001.

[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94, 2000.

[8] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[9] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. *SIGARCH Comput. Archit. News*, 34(2):78–88, 2006.

[10] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[11] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In *Proceedings of International Symposium on Computer Architecture*, 2001.

[12] J. Freeman. Artificial intelligence: Simulating a basic genetic algorithm. *The Mathematica Journal*, 3:52–56, 1993.

[13] S. Ghiasi and D. Grunwald. Aide de camp: Asymmetric dual core design for power and energy reduction. In *University of Colorado Technical Report CU-CS-964-03*, 2003.

[14] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low-power cpu. In *International Conference on Mobile Computing and Networking*, Nov. 1995.

[15] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. In *Proceedings of IEEE International Conference on Computer Design*, 2004.

[16] D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun. Confidence estimation for speculation control. In *ISCA '98: Proceedings of the 25th annual international symposium on Computer architecture*, pages 122–131, 1998.

[17] Intel Corp. *Intel's Teraflops Research Chip*.

[18] 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 *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[19] A. Iyer and D. Marculescu. Power-performance exaluation of globally-asynchronous, locally-synchronous processors. In *International Symposium on Computer Architecture*, 2001.

[20] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. In *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, 2003.

[21] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark. Coordinated, distributed, formal energy management of chip multiprocessors. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 127–130, New York, NY, USA, 2005. ACM.

[22] H. Kimm, S. Shin, and C. O. Sung. Evaluation of interval-based dynamic voltage scaling algorithms on mobile linux system. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, 2007.

[23] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Processor power reduction via single-isa heterogeneous multi-core architectures, 2003.

[24] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance. In *International Symposium on Computer Architecture*, June 2004.

[25] C. Lefurgy, X. Wang, and M. Ware. Power capping: a prelude to power shifting. *Cluster Computing*, 11(2):183–195, 2008.

[26] J. Li and J. Martinez. Power-performance implications of thread-level parallelism in chip multiprocessors. In *Proceedings of International Symposium on Performance Analysis of Systems and Software*, 2005.

[27] Y. Li, D. Brooks, Z. Hu, and K. Skadron. Performance, energy, and thermal considerations for smt and cmp architectures. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 71–82, Washington, DC, USA, 2005. IEEE Computer Society.

[28] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. *SIGARCH Comput. Archit. News*, 26(3):132–141, 1998.

[29] J. Oliver, R. Rao, P. Sultana, J. Crandall, E. Czernikowski, L. W. J. IV, D. Franklin, V. Akella, and F. T. Chong. Synchroscalar: A multiple clock domain, power-aware, tile-based embedded processor. *isca*, 00:150, 2004.

[30] D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83:394–410, 1995.

[31] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multi-level power management for the data center. *SIGOPS Oper. Syst. Rev.*, 42(2):48–59, 2008.

[32] G. Semeraro, G. Maklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *International Symposium on High-Performance Computer Architecture*, Feb. 2002.

[33] J. Sharkey, A. Buyuktosunoglu, and P. Bose. Evaluating design tradeoffs in on-chip power management for cmps. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*, pages 44–49, New York, NY, USA, 2007. ACM.

[34] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. In *IEEE Micro: Micro's Top Picks from Computer Architecture Conferences*, Dec. 2003.

[35] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, 2004.

[36] A. Snavely and D. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading architecture. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[37] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 234–244, 2000.

[38] D. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, Dec. 1996.

[39] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *SIGARCH Comput. Archit. News*, 32(5):248–259, 2004.