Distributed Peak Power Management for Many-core Architectures

John Sartori and Rakesh Kumar
Coordinated Science Laboratory
1308 West Main St
Urbana, IL 61801

Abstract

Recently proposed techniques for peak power management [5] 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 a hierarchical and a gradient ascent-based technique for decentralized peak power management for many-core architectures. The proposed techniques 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 [3].

While the inefficiencies due to the lack of peak power management are substantial even for uniprocessors [5], 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 [5]. 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 [5] involve centralized decision-making and assume quick evaluation of the various power management states. For example, Isci, *et al* [5], propose techniques that assume a global power manager that records the performance and power information corresponding to all the cores on the die and then instantly searches through the decision space. 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 [4], with two power states (full-power and half-power) can have over 1.2×10^{24} possibilities! While one can use intelligent search/optimization techniques to prune the search space, an alternate approach is to look for solutions that are not centralized. This paper presents two such solutions.

The second limitation of previously proposed techniques 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 guarantee that the power of a multi-core processor does not exceed a threshold. The guarantee 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 applications' execution characteristics to maximize the throughput of the processor.

2 Throughput Benefits of Peak Power Management

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 4). 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

[0]Power overshoots can probably be prevented by triggering corrective action conservatively

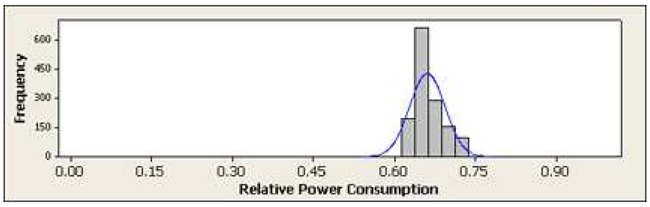

Figure 1. On average, a multicore processor consumes only a fraction of its maximum rated power.

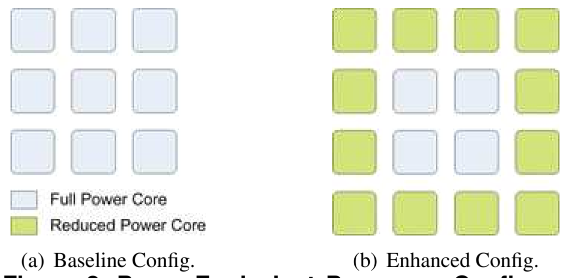

Figure 2. Power-Equivalent Processor Configurations.

power by intelligently scaling down the power for a subset of 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 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.

2.1 Intelligent Core to Power State Mapping to Maximize Throughput

While the availability of more cores should be sufficient to guarantee increased throughput for an architecture that supports peak power management, performance can be further maximized by choosing the power state of each core intelligently, based on application characteristics. In this section, we describe the various baselines that we constructed to compare against our proposed approaches.

2.1.1 Static Mapping

We considered two static baselines – *random static* and *static oracle* – that prevent power overshoots and, therefore, provide the throughput benefits mentioned previously.

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 [6] requires foreknowledge of the behavior of applications in various power states and employs a metric called weighted speedup (WS) [8]. 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 sorting the applications in a workload with respect to WS. 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.

2.1.2 Dynamic Mapping

We used the previously proposed MaxBIPS policy [5] as our centralized dynamic baseline.

MaxBIPS, proposed by Isci *et al* [5], aims to optimize system throughput by predicting and choosing in the power scheduling stage the power mode combination that should maximize the throughput. MaxBIPS policy predicts the corresponding power and BIPS values for each possible 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. As confirmed in Section 5), the technique is not scalable for a large number of cores. Secondly, while MaxBIPS attempts to limit the global power consumption, it allows power overshoots and triggers reactive correction for these overshoots.

The two dynamic techniques proposed in the next section do not allow power overshoots and are targeted towards architectures with a large number of cores.

3 Non-centralized Peak Power Management

In a many-core architecture, the responsibility of power management must be shifted away from a central arbiter and distributed to multiple locations around the processor.

3.1 A Hierarchical Approach

Consider a processor with 64 cores in which half of the cores run at full power and the other half run at reduced power to meet the peak power budget. The task of choosing the optimal power state boils down to choosing the 32 cores that can best utilize the full power state, or alternatively, choosing the 32 cores that suffer the least from running in a reduced power state. Essentially, the size of the search space contains C(64,32) configurations, where C(x,y) denotes the number of y-combinations from an x-set.

Now, consider the same processor, divided into 4 clusters. For each cluster, the search space contains C(16,8) configurations when power is divided evenly between the clusters. Thus, the search space for the entire processor consists of 4 parallel decisions between C(16,8) states – 14 orders of magnitude fewer than the number required for a global decision. When cluster-level power distribution is included, the number of states will be reduced even further. Clearly, hierarchical scheduling demonstrates considerable potential for reducing the search space for power management.

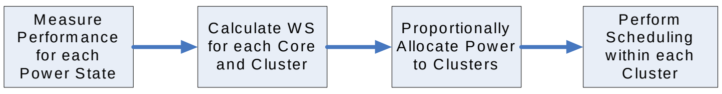
Measure Performance for each Power State
Calculate WS for each Core and Cluster
Proportionally Allocate Power to Clusters
Perform Scheduling within each Cluster

Figure 3. Hierarchical scheduling performs power management within clusters to reduce arbitration overheads.

Hierarchical Power Management involves dividing the processor into several clusters of cores and performing local scheduling within each cluster. The next level up in the hierarchy provides power management between clusters. Power negotiation between clusters is based on the aggregate weighted speedup (WS) metric.

Power is allocated to each cluster proportionally, based on WS. During allocation, the cluster-level arbiter ensures that no cluster receives more power than it can possibly use or less power than it needs to run each core at the most reduced power state. When concessions need to be made to satisfy the peak power guarantee, a deficit is cleared by skimming power from clusters in the order of increasing WS, and a surplus is distributed by allocating extra power to clusters in order of decreasing WS. In this respect, each cluster can be thought of as owning some "power tokens". The amount of power that a cluster can consume is determined by the number of tokens present in the cluster. The tokens can flow through the system. The amount of power that a cluster can consume can vary with time, depending on application characteristics.

As an example, if the weighted speedup values for the 4 clusters of a 16 core processor described above are 5,4,3, and 2 respectively, and if the peak power budget of the chip is 70W, the clusters receive 25W, 20W, 15W, and 10W respectively. Within each cluster, local scheduling performs dynamic subsetting of cores with high power states such that cluster-level instantaneous peak power budget is respected. If the WS values change over time, the cluster-level peak power budget is adjusted accordingly.

The algorithmic flow of hierarchical scheduling is depicted in figure 3. Note that hierarchical scheduling may result in the decisions that are globally suboptimal. The degree of sub-optimality is determined by the effectiveness of moving the power tokens around from one cluster to another.

In this paper, we consider a hierarchical scheduler that is based on hybrid a global trigger – i.e., re-allocation and local scheduling is triggered when the change in aggregate weighted speedup of the chip exceeds a threshold. For re-allocation, as described above, power is allocated to clusters based on their contribution to the aggregate weighted speedup. Local scheduling for all the clusters is done in parallel and involves calculating the WS for each core, sorting the WS values, and allocating power in order of decreasing WS.

3.2 A Gradient Ascent-based Approach

Hierarchical scheduling reduces the cost of power management decisions. However, the cost of making such decisions is truly minimized only when power state determination is performed locally by each core rather than by a high-level arbiter that must consider the needs of many cores. One way to shift the responsibility of power management to the core level is to introduce a distributed control algorithm such as *gradient ascent* [1].

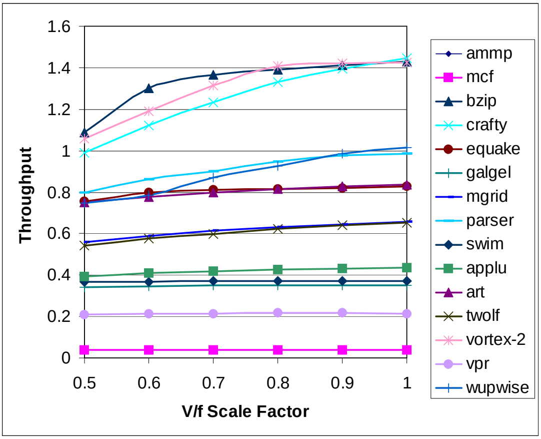

Figure 4. Different applications exhibit varying performance gradients with respect to power scaling.

Gradient ascent is motivated by the fact that different applications have different rates of performance decrease when voltage/frequency is scaled down and makes power allocation decisions based on these rates. Figure 4 shows the performance gradients for some SPEC benchmarks. For some applications, the gradient of performance with respect to power is steep. Gradient ascent determines that these applications should be allocated full power. Other applications, however, have flat gradients and therefore should run in the lowest possible power state for the most efficient power mapping. Since all the information needed to choose an efficient power state is present at the core level, the decision-making process can by executed locally, thereby eliminating the global arbiter and creating a scalable power management policy, independent of the number of cores on the processor. The algorithm in figure 5 describes the gradient ascent approach to peak power management.

To provide a global peak power guarantee, a single power balance counter is used to track the number of requests to ascend and descend one power level. If the power balance is not zero after each core asserts its desire, either some cores must accept a lower power state or some cores must assume a higher power state to keep the power requirement of the processor constant. A natural tradeoff exists here between the locality and scalability of decision-making and the amount of global information used, which in turn affects the optimality of the solution.

Ideally, the cores which evidence the steepest gradients should receive extra power first to resolve a positive power balance. Similarly, the cores with the flattest gradients should be the first to give up power when concessions must be made to satisfy the guarantee. However, these optimizations require more global information in the form of a weighted speedup value from each core. Alternatively, the cores that are forced to change power states may be selected randomly or based on a weighted speedup threshold. The former techniques result in a solution that is closer to optimal whereas the latter techniques favor decentralization of the decision-making process.

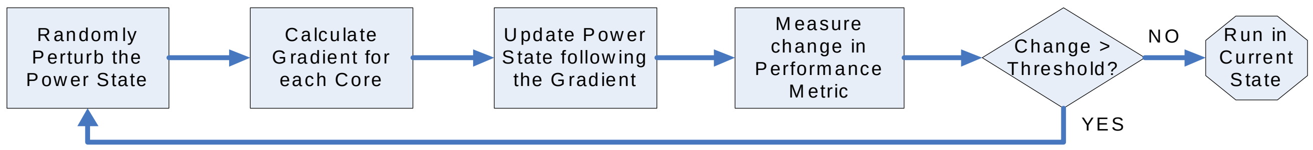
Randomly Perturb the Power State
Calculate Gradient for each Core
Update Power State following the Gradient
Measure change in Performance Metric
Change > Threshold?
NO
Run in Current State
YES

Figure 5. The gradient ascent approach performs power management locally at each core rather than relying on a global arbiter.

A common difficulty with gradient ascent is the presence of local maxima on the path of ascent. Since the gradient in any direction is negative at a maximum, the control variable will inevitably return to the maximum, unless the perturbation of the state is large enough to escape the realm of the maximum. Interestingly, the gradient ascent approach to peak power management is not impeded by the presence of any local maxima. This is because the performance function is guaranteed to be strictly monotonic. The performance of an application will never increase as frequency is decreased and can never decrease as frequency is increased. Thus, no local extrema exist for the performance function, and local determination of the gradient always results in a step towards optimal performance.

3.3 Decentralized MaxBIPS

Finally, for completeness, we also consider a decentralized version of MaxBIPS. The decentralized version of MaxBIPS is same as hierarchical scheduling where the local scheduling policy is MaxBIPS. Determination of when to do mapping is still based on a hybrid global trigger.

Note that decentralized MaxBIPS will not prevent power overshoots because of the reactive nature of MaxBIPS.

4 Methodology

Table 1 presents individual core specifications for the baseline processor.

Core Component	Baseline Value
Fetch Width	2
Issue Width	2
I/D Cache	32 KB, 4 way, 1 cycle latency
ITLB/DTLB	48/128 entries
MSHR	16 entries
L2 Cache	shared, 4 way, 12 cycle latency, 8 banks 4/8 MB for 8/16-core baseline

Table 1. Core Specifications.

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. Consequently, enhanced configurations that are compared against an 8-core baseline are modeled with a 2 MB L2 cache. As the core count is scaled up to allow analysis of our power management techniques in many-core architectures, the size of the L2 cache is scaled linearly with the number of cores.

Power estimates reported by Wattch [2] 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 2 gives the peak power figures for several V/f scaling factors, assuming 65 nm process technology. At most, a core is allowed to scale down to half of the original voltage and frequency. Thus, we consider voltages from 1.5 to 0.75 Volts.

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

Table 2. Peak Power Consumption 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 3 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 3. V/f Switching Penalties.

Workloads are constructed from a set of 16 SPEC2000 benchmarks. Table 4 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 4. Simulation 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). The *all* workload contains all 16 benchmarks in equal proportions. The *high sensitivity* workload includes benchmarks that exhibit the highest sensitivity to V/f scaling. The *low sensitivity* workload includes benchmarks with low sensitivity to V/f scaling. The *mixed sensitivity* workload includes benchmarks with varying degrees of sensitivity to V/f scaling. Finally, the *dynamic* workload contains benchmarks that exhibit more dynamic behavior than others.

Simulations are performed using SMTSIM [9] 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 [7], all simulations run for 1 Billion cycles.

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

5 Analysis and Results

In this section, we demonstrate the performance benefits of peak power management for many-cores.

5.1 Improving Throughput through Peak Power Management

$X \times V_{dd}$	Core Count	Full Power Cores	Reduced Power Cores	Peak Power (W)
1.0	8	8	0	146.0 W
0.8	9	5	4	143.8 W
0.7	10	5	5	145.9 W
0.5	11	6	5	145.3 W

Table 5. Alternative processor configurations with the same peak power bound.

To quantify the throughput benefits of peak power management, we consider an 8-core baseline processor, all cores running at full power, and compare it against peak power-equivalent processors with larger number of cores, where some cores are running at reduced power. Table 5 describes four different processor configurations which have nearly the same peak power requirements but differ in the number of cores. Figure 6 shows the corresponding throughputs for different static 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.

5.2 Overhead of Managing Peak Power for Many-cores

The previous section shows the potential benefits of static peak power management. Performance improvements for dynamic peak power management depend on the overhead of making dynamic power scheduling decisions. Figure 7 shows the overhead of making peak power management decisions for the various techniques for various number of cores. MaxBIPS, which has been shown as an effective

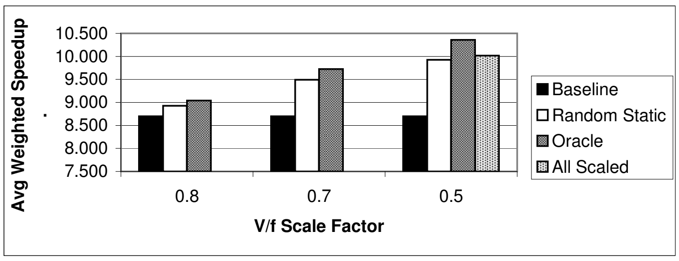

Figure 6. Performance comparison against the baseline processor for various peak power management techniques.

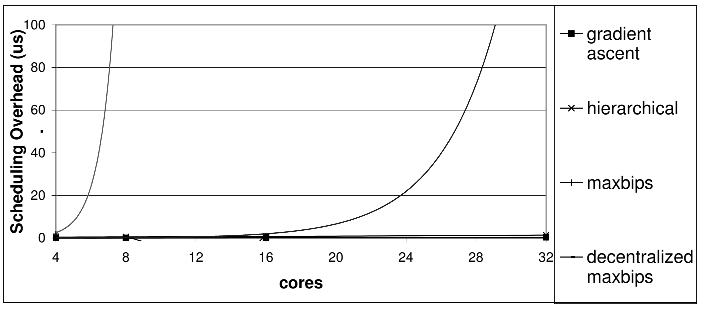

Figure 7. Timing Overhead of Making Dynamic Peak Power Management Decisions

global power management technique in [5], is unsuitable for peak power management for many-core architectures (with unacceptably large overheads even for 8 cores) due to the exhaustive nature of its state space search. Even the decentralized version of *MaxBIPS* has unacceptable (exponential) overheads for more than 16 cores due to the exhaustive nature of its local search. The decentralized policies (*gradient ascent* and *hierarchical*) have the least overhead. In fact, the overhead of *gradient ascent* shows little change with increasing number of cores till at least 1000 cores and is, therefore, the most suitable for many-core peak power management. Overheads for other techniques increase linearly and may be prohibitive for a large number of cores.

5.3 Power Management using Non-centralized Techniques

The previous section demonstrated that naive, centralized peak power management policies may be inadequate for peak power management for chips with a large number of cores (more than 16 or 32). There are certain policies that are inadequate even for 8 core processors. In this section, we consider a chip with a small number (16) of cores and compare the various strategies in terms of throughput improvement when their timing overheads are still acceptable. Figure 8 shows the results. The graph contains results for all policies except *MaxBIPS* (which has unacceptably high overhead for 16 cores). Decentralized MaxBIPS results are included as the timing overhead is still manageable for 16 cores.

Results show that the proposed distributed techniques fare quite well at peak power management even for 16-core processors (for which the naive centralized techniques have acceptable timing overheads) due to their ability to re-

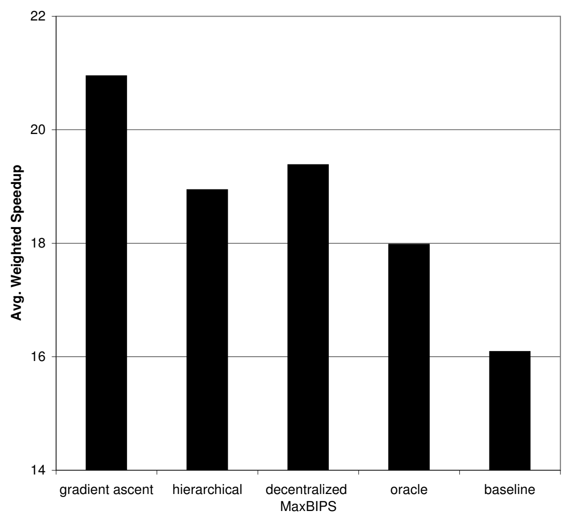

Figure 8. Performance of many-core power management policies against the baseline and the static oracle.

duce the state search space. With the hierarchical scheduling approach, for example, dividing the arbitration decision between four clusters in a 16-core processor reduces the complexity of the task by 99.8%. The use of hierarchical power management generated average improvements of 18.7% and 7.8%, respectively, over the baseline and static oracular models.

Note that hierarchical scheduling produces a power mapping that is potentially globally and also locally suboptimal. This is why worse performance is observed for 16 cores compared to decentralized MaxBIPS which is locally optimal. However, as Figure 7 shows, the overhead of decentralized maxBIPS becomes prohibitive for higher number of cores, so hierarchical scheduling will easily win out.

The gradient ascent approach to peak power management produced the best results of any policy, demonstrating a 33.3% improvement over the baseline processor and a 22.5% increase over the static oracle. While most other policies were designed with two possible power states for each core, the gradient ascent algorithm was given more freedom in the application of V/f scaling. Because the task of arbitration is distributed to each core, this additional control does not represent a substantial increase in the search space for power configurations. In fact, the complexity of the algorithm will remain the same for an arbitrary number of possible power states. However, since our modified gradient ascent approach only shifts scale factors in discrete quanta, more power states will result in longer time to convergence when the optimal global mapping is far from the initial state of the processor. Overall, decentralized arbitration and flexibility in V/f scaling make the gradient ascent approach particularly effective in choosing the optimal power allocation for each core.

6 Summary and Conclusions

This research focuses on peak power management for many-core architectures. We demonstrated that there is a signficant throughput advantage to doing peak power management for many-core architectures where power is prevented from exceeding the peak power budget. We proceeded to show that managing peak power for many-core architectures (processors with tens or possibly hundreds of cores on the same die) poses a significant challenge when using naive centralized techniques. Finally, we proposed two non-centralized peak power management techniques that are specifically tuned for many-core architectures.

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 policies that we devised specifically for many-core architectures performed, on average, at least 22% better than our policies for multi-core architectures, even for 16 cores. As the number of cores on a processor die rapidly increases, the effectiveness of our techniques will only continue to increase.

References

[1] P. Bartlett, E. Hazan, and A. Rakhlin. Adaptive online gradient descent. Technical Report UCB/EECS-2007-82, EECS Department, University of California, Berkeley, Jun 2007.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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