

Proactive Peak Power Management for Many-core Architectures

John Sartori and Rakesh Kumar

Coordinated Science Laboratory

1308 West Main St

Urbana, IL 61801

Abstract

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. However, 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.

This paper proposes proactive peak power management policies that attempt to prevent the power of a processor from exceeding a certain threshold. The threshold is chosen to be close to the peak power of the processor, thereby minimizing the inefficiency due to the growing gap between average power and peak power of a processor, especially a multi-core processor [14]. We demonstrate that proactive peak power management can enable the placement of several more cores on a die than the power budget would allow. This can result in significant (up to 47%, 33% on average) improvements in throughput for a given power budget.

We also show that proactive peak power management does not have to be centralized and heavyweight and can be applied even to many-core architectures (processors with a large number of cores). We investigate a number of efficient, decentralization techniques – e.g., mapping the proactive peak power management problem to a disjunctively constrained 0-1 knapsack problem, using machine learning and classical search/optimization approaches to reduce the decision space, and using distributed control algorithms for decentralized decision making. Several of these techniques can be used even to reduce average power through traditional dynamic and global power management.

1 Introduction

Power is known to be a zero to first order design constraint [3] for microprocessors. A considerable amount of research effort has been devoted to the reduction of power consumption of processors as well as reducing the adverse effects of power on processor performance, cost, and reliability [14, 30, 31, 7]. Special attention has been devoted to reactive dynamic power management techniques like voltage/frequency scaling [9, 16], throttling [3], and clock gating [2, 19, 10] that utilize program performance information to lower the power consumption of a processor.

One major limitation of most dynamic power reduction techniques applied conventionally is that they exploit slack in the execution behavior of programs to reduce average power. Peak power is often left untouched. However, peak power plays a large role in determining the characteristics and hence the cost of the power supply, thermal budgeting for the chip [4, 24], as well as the reliability qualification of the processor [6].

Inability to do peak power management often results in underutilization of available processor power. For example, power supply and delivery circuitry are over-designed for a worst case that rarely occurs in normal operation. This unduly increases the system cost. Similarly, a power supply that is designed for the peak power requirement will exhibit diminished efficiency when supplying loads that are substantially less than the peak power [11]. Hence, additional power will be dissipated due

⁰University of Illinois at Urbana-Champaign Center for Reliable and High-Performance Computing Technical Report number CRHC-07-04

to these inefficiencies. Similarly, a large gap between peak power and average power can result in large current swings and manifests itself as the dI/dt problem [15, 22]. It also results in thermal inefficiencies (see Section 3.2).

While the inefficiencies are substantial even for uniprocessors [14], the extent of the problem gets much worse as multi-core processors become increasingly pervasive. The gap between peak power and average power keeps increasing with the increasing number of cores on a processor die – in fact, the gap is multiplied by the core scaling factor [14]. If one believes the predictions of tens to hundreds of cores on future processors [1], effective, efficient, and scalable peak power management will become a necessity.

This paper proposes proactive peak power management policies that attempt to prevent the power of a processor from exceeding a certain threshold. The threshold is chosen to be close to the peak power of the processor, thereby minimizing the inefficiency due to the growing gap between the average power and peak power of a multi-core processor. Proactivity allows the extra power represented by the gap between average and peak power to be put to use in increasing a processor's throughput, thereby maximizing the performance to power ratio of the multi-core processor. This may be accomplished in two ways – either by reduction of the peak power for the same number of cores or by increasing the number of cores while keeping the peak power requirement the same. In our experimentation, we consider the latter case, but both cases are essentially equivalent.

Although our proposed architecture increases core integration relative to the baseline configuration, the peak power of the processor remains the same. This is accomplished by presenting each core with voltage and frequency scaling capabilities and introducing arbitration that proactively controls the power states of the cores to ensure that the peak power of the chip never exceeds a fixed budget. Under the proactive power management scheme, some cores will operate at full power and some will be scaled such that the peak power of the enhanced processor remains unchanged.

Since a centralized global decision-making entity [14] will not scale well as the number of cores continues to increase (e.g. as we move from multi-cores to many-cores), we investigate a number of efficient and scalable decentralization strategies for proactive peak power management in many-core architectures. First, we employ intelligent search routines, machine learning, and remodeling of the arbitration decision (to a disjunctively-constrained 0-1 knapsack problem) to speed up the decision making process. Secondly, we propose the coupling of these efficient methods with decentralized power management techniques (e.g., using distributed control algorithms like gradient ascent) that will scale to a processor of arbitrary size. Several of these techniques can be used even to reduce average power through traditional dynamic and global power management.

This paper makes the following contributions.

- We introduce a proactive approach to peak power management and demonstrate the benefits of such an approach. Proactivity can be used to provide guarantees regarding peak power consumption. In turn, these guarantees can be exploited to allow full utilization of available power in a processor.
- Through the application of proactive peak power management, we demonstrate the ability to add more cores to the die than would be allowed in a traditional sense, generating throughput gains of up to 47% and 33% in the average case.
- We extend power management techniques to the realm of many-core processors, addressing issues such as scalability and efficiency. We adapt a number of efficient and decentralized power management techniques which facilitate scala-

bility to many-core architectures. Our application of optimization techniques such as machine learning, gradient ascent, and evolutionary algorithms represents a novel approach to power management.

- Our research motivates the use of proactivity in other areas of computer architecture and demonstrates some of the additional benefits that result from proactive peak power management. Namely, we demonstrate increased utilization of processor power, reduction in power and area overheads for power supply and decoupling capacitance, decreased current variability, and increased efficiency and performance.

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 gating techniques [2, 19, 10], voltage and frequency scaling [9, 16], or heterogeneity [17] to reduce processor power consumption. While these approaches to power management decrease the average power consumption of a processor, they do not address the problem of peak power management. Our approach to power management differs from these methodologies in that we address the problem proactively, providing guarantees on the power consumption of a processor to allow for full and efficient utilization of available power.

Other power-related work has addressed current variability [15] and thermal issues [4, 24] in modern microprocessors. Previous techniques to deal with these problems have been reactive in nature. These approaches do not provide guarantees. In contrast, the guarantees provided by our proactive methodology allow for increased efficiency in the management of these factors.

One approach to power management proposed by Annavaram *et al.* [3], attempts to stabilize power consumption within a fixed budget by controlling the energy consumed per instruction in response to the parallelism of a program. This work differs from ours in that we provide guarantees by strictly enforcing a fixed power budget through proactive means.

Perhaps the closest work to ours is that of Isci, *et al.* [14], who propose the use of a global power manager to limit the chip-level power consumption of a processor. While their methods address the gap between average power and peak power consumption in multi-core processors, they do not provide guarantees. Their global power manager reacts to power overshoots by selectively lowering the power of certain cores. Another key aspect of our work is the extension of power management techniques to the realm of many-core architectures. Instead of a centralized arbiter that makes decisions globally for a small number of cores, we institute decentralized approaches that allow our techniques to scale for application in many-core processors. We also propose efficient techniques that reduce the costs associated with evaluation and decision making, enhancing the scalability and maximizing the performance of our power management strategy. Note that several of these techniques can be used even to enhance the effectiveness of the power management scheme proposed by Isci, *et al.*

3 Proactive vs Reactive Power Management

In this section, we discuss how providing power *guarantees* with existing reactive approaches to power management is possible only for *unacceptable* performance/area overheads. We also discuss other benefits of proactive peak power management.

3.1 Providing Power Guarantees

One might think that power guarantees can be provided even reactively by detecting imminent current overdraw (using a voltage/current sensor) and then reacting by throttling the processor [26]. However, a current sensor that is employed to constantly monitor the current drawn by a processor must be able to report an impending power emergency to the processor in a timely manner, such that the maximum power constraint will not be violated, resulting in damage to the system. This requirement implies that the speed of the processor is limited by the speed of the sensor. However, the fastest current sensors operate at frequencies well below those of present day microprocessors [12]. Thus, the shortcomings of current sensor technology would place a burdensome limitation on the operating frequency of the processor, making current sensor based reactive peak power management unfeasible.

Similar limitations exist for voltage sensor based reactive peak power management. For example, the fastest voltage sensor that we could find in literature [20] multiplexes two 500 MHz sample streams to provide an effective sampling rate of 1 GHz. The sensor has 8-bit resolution and a conversion latency of 8 cycles. The circuit of a 3GHz processor, therefore, would have to be able to withstand 24 cycles of overdraw before the condition would be detected by the sensor. Alternatively, the processor could be slowed down to a frequency of 125 MHz and react to the power emergency after one cycle of overdraw. Regardless of the approach, reactive methods are unable to provide guarantees for peak power management in a timely manner.

Two trends make the situation even worse. With each successive technology generation, supply voltage tends to decrease while processor power increases. Hence, resolution of voltage sensors will need to improve to provide the precision needed for any reactive technique to be even considered. Unfortunately, increasing the resolution of the voltage sensor means that the latency of the conversion will increase. Secondly, as the number of cores increases rapidly on processor dies, the gap between peak and average power increases, resulting in an increase in the extent of possible overdraw. This will reduce the sustainability of overdraw using circuit techniques like decoupling capacitance, etc., and will decrease the effectiveness of reactive techniques.

A proactive peak power management approach rigorously enforces a tight bound on the peak power consumption which allows for operation close to or at the maximum. Thus, proactive management enables full utilization of the processor's resources and consequently performs much better than a necessarily overly conservative, reactive technique. In this paper, we use proactivity to improve the throughput of a multi-core processor by putting more cores on a processor die than what the power budget would traditionally allow.

3.2 Current Variation, Thermal Guarantees, and Decoupling Capacitance

The ability of proactive peak power management techniques to prevent the current from exceeding a threshold that is close to the peak current drawn by the processor has several other significant implications.

One implication is increased control over current variation in processors. Aggressive techniques for power reduction in today's microprocessors cause variations in current draw that may destabilize the processor's supply voltage. This issue is commonly referred to as the dI/dt problem [15]. Proactive peak power management aims to increase power utilization and

decrease variations in power dissipation, thus reducing the frequency of voltage emergencies which result from large current swings.

Similarly, proactivity can be used to provide thermal guarantees. With knowledge of the heating and cooling rates of processor regions, proactivity could be employed to manage access to active regions in a way that guarantees a bound on local temperature. Also, peak die temperature, which is immediately related to power density, can be controlled directly through proactive power management. Note that sensor-based reactive approaches have been used for thermal management, but the effectiveness of the reactive techniques is limited by the technology and response time of the sensors [24]. The best sensors for this application have a precision of $\pm 2^\circ\text{C}$ and a sampling period of $10\ \mu\text{s}$. To improve the precision to $\pm 1^\circ\text{C}$, a moving average filter with a 10 sample window can be used [24], but this slows down the sensing even more. Even with a single sample, the sampling rate of the sensor is unacceptable for providing a thermal guarantee. Another problem with sensor precision arises from sensor placement issues. Hot spots are critical locations on a chip that heat up faster than other areas of the chip and can potentially cause timing errors and physical damage. To accurately measure the temperature of a hot spot, a sensor should be placed in direct contact with the region. However, due to layout constraints, this is not always possible. As the sensor moves away from the hot spot, the precision of the temperature measurement degrades, causing the effectiveness of a reactive technique to diminish as well. Because of the issues with sensor technology, reactive techniques, though effective in reducing average temperature, are unsuitable for providing thermal *guarantees*.

An ancillary advantage of proactive peak power management is processor area savings due to a reduction in the amount of decoupling capacitance that is required. Decoupling capacitors are used to prevent hazardous voltage drops on the supply lines that result from current overdraws and the dI/dt problem. The area overhead for these capacitors is quite substantial – 15 to 30% of chip area to sustain transient noises of 10 to 15% of the supply voltage [32]. Since reactive power management techniques cannot provide guarantees on the peak power consumption, reactive designs must provision for prohibitively large amounts of decoupling capacitance to handle the increased overdraw that is possible when more cores are added to the processor for the same peak power rating. With proactive power management, however, the peak processor power is guaranteed to be bounded, resulting in no need to increase decoupling capacitance.

4 Improving Multi-core Throughput through Proactivity

This section discusses how proactive peak power management enables putting 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.

4.1 Integrating Cores to Bridge the Average/Peak Power Gap

There is often a sizeable 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 7.2). On average, the processor consumes only 66% of its maximum rated power. However, the processor and the power supply still must be designed to supply the peak power and rated to handle this load. In

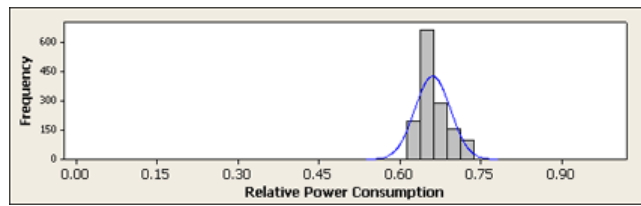


Figure 1. Relative Power Consumption in a Multi-core Processor.

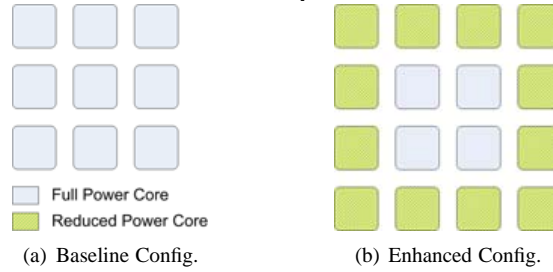


Figure 2. Power Equivalent Processor Configurations.

theory, we should be able to add approximately 50% more cores *running at full power* and still remain below the peak power, on average.

We propose an architecture with *proactive 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 proactive peak power management mechanism *guarantees* that the peak power (of the baseline) will never be exceeded (even though the processor contains several more cores than the power budget would normally allow). The proactivity mechanism provides this guarantee 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.

For example, figure 2 shows a 9-core baseline processor with all cores running at full power and a 16-core processor with a proactive peak power management mechanism that runs 4 cores at full power and 12 cores at reduced power (through the use of V/f scaling). The aggregate peak power requirement of both processors is the same. Nevertheless, the throughput of the 16-core processor can be up to 78% higher (assuming linear dependence with the number of cores on the die). The observed performance gains, of course, will be somewhat less than this optimal value due to overhead costs of our power management techniques, and since throughput actually does decrease with V/f scaling in most cases.

Note that the exact mechanism for proactive arbitration may involve either using a scheduler (hardware or OS-based), a microcode implementation, a hardware implementation of a state machine, or a token management logic that assigns power tokens to individual cores. All of our subsequent discussion is valid for all of these mechanisms, even though the results are shown assuming an arbitration overhead comparable to that of an OS-level scheduler.

4.2 Intelligent Core to Power State Mapping to Maximize Throughput

While the availability of more cores should be sufficient for guaranteeing increased throughput for proactively managed architectures, performance can be further maximized by choosing the power state of a core intelligently, based on application characteristics [18]. We devote the next several sections to discussing techniques that map applications to power states.

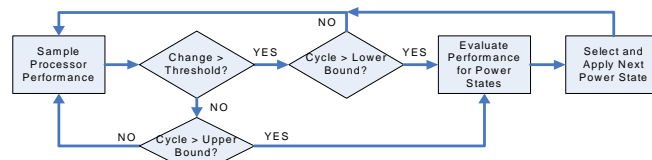


Figure 3. Trigger-based Dynamic Scheduling.

4.2.1 Static Mapping

In our research, we considered various static policies including *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. This type of static scheduler can easily be implemented in hardware.

Static oracular scheduling [18] requires foreknowledge of the behavior of applications in various power states and as such is not implementable. Still, this arbitration scheme is evaluated to provide an upper bound on the performance of static approaches. To devise an oracular power mapping, a metric called weighted speedup (WS) is employed. 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.

Dynamic policies are also compared against a static configuration in which all cores are scaled to the lowest power state. This configurations maximizes core integration for a given power budget.

4.2.2 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. Dynamic policies, on the other hand, gather statistics during program execution and use the gathered data to select the most efficient mapping for each application in the current program phase. A simple dynamic strategy samples throughput at a regular rate, uses throughput to calculate WS, and determines the next processor power state based on WS as described above.

A more sophisticated dynamic policy, depicted in Figure 3, attempts to determine exactly when the power needs of the processor change and only performs evaluation and arbitration at these times.

Although these policies react to performance changes, the bound on peak power is guaranteed proactively by only allocating full power to a subset of cores. The policies are employed to select this subset and consequently only affect performance. Policies have no effect of the peak power guarantee, which is always strictly enforced.

For trigger-based dynamic scheduling, each time a new performance sample is taken, the percentage change is calculated between the current throughput and the throughput measured after the most recent application of a new power state. If the change indicates that the performance has decreased by more than a specified threshold, the power scheduling routine is triggered. This occurrence suggests that some applications have entered a new phase of execution and a new power mapping should be determined to match the new requirements of these applications. A lower bound on the number of cycles be-

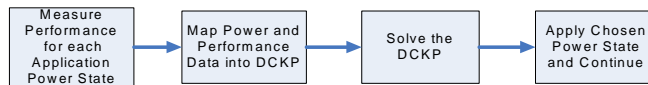


Figure 4. Knapsack Modeled Power Management.

tween evaluations is enforced to prevent evaluation from being triggered while the processor settles into its new power state. Evaluation may also be triggered if the number of cycles that have elapsed since the last evaluation surpasses an upper bound.

4.2.3 Mapping Considerations for Many-cores

The above scheduling policies rely on each core’s performance and power information being sent to a central decision-making node (or to the central arbitration logic) followed by an exhaustive search performed on the possible core to power state mappings to identify a good mapping. Clearly, these mechanisms and policies will not scale for processors with tens or hundreds of cores on the processor die (many-core processors). The interconnection network required to support such an architecture is not scalable if a hardware scheduler or a hardware arbitration mechanism is assumed. Similarly, the inefficient search methods employed when finding a new power state will have trouble as the number of cores and possible power states to choose from increase. The next two sections address these problems and present techniques for scalable, effective, and efficient peak power management in many-core architectures.

5 Efficient Proactivity for Many-cores

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. An 8-core processor with four power states (three voltage states and one off state), for example, has 4^8 , i.e., over 65 thousand possible global states. Similarly, an 80-core processor like Intel’s recent announcement [13] with two power states (full power and half-power) can have over 1.2×10^{24} possibilities! Any naive method of arbitration will clearly be infeasible for such processors.

We discuss below some efficient techniques that reduce the amount of time required to make a decision as well as the number of decisions executed during search. These techniques directly promote the scalability and efficiency of our proactive peak power management methods. **Note that the discussed techniques can be used even for reactive power management.**

5.1 Modeling Proactivity as a Disjunctively Constrained 0-1 Knapsack

As discussed above, algorithms with superlinear time complexity will become unmanageable as the search space for the power management problem continues to grow, either due to more cores on the die, more power states per core, or both. One way to combat this problem is to map 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 (DCKP) [21]. Figure 4 describes the process of power arbitration using the knapsack approach.

Knapsack problems operate on a set of items, each with a fixed profit and weight. 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

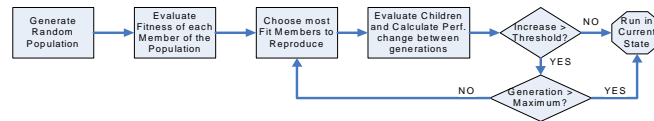


Figure 5. Genetic Algorithm for Power Management.

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.

Notice that 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.

5.2 Treating Proactivity as Classical Search Problem

An alternate way to increase the efficiency of making close-to-optimal decisions about the best global state with significantly reduced complexity/overhead is to treat the decisions as classical search/optimization problems. The goal of a classical search/optimization problem is to prune the search space based on certain properties of the search algorithm as well as the search space.

One intelligent search method that has been applied to a wide range of optimization problems is the genetic algorithm [8]. 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 5, begins with the creation of random configurations to fill the initial population. These random configurations must be designed to respect the power budget of the processor. After initialization of the population, each configuration is applied and evaluated with respect to a fitness metric. In this case, the throughput of the processor in the current configuration determines the fitness of the power mapping. Once 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 this 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 the number of conflicts between two parents must be even, the decision bit serves the function of guaranteeing 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

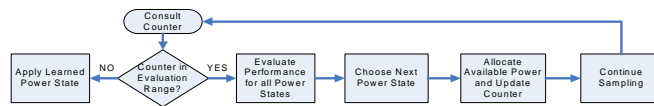


Figure 6. Power Management using Machine Learning.

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.

5.3 Using Machine Learning Approaches to Reduce the Proactivity Overhead

While performing an intelligent search can substantially reduce the arbitration overhead, the search space can be reduced significantly more if application characteristics are also used 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 6 describes an approach to peak power management that employs machine learning.

To implement machine learning, a set of counters – one for each core – is created 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 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 temporarily poor mapping are eliminated. Furthermore, the overhead associated with the evaluation process is significantly reduced, since decisions must be made for fewer cores.

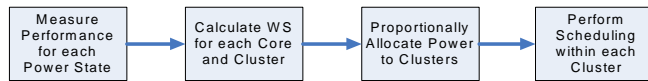


Figure 7. Hierarchical Scheduling.

The application of the above three techniques significantly reduces (results in Section 8.2) the overhead of proactive peak power management for many-core architectures. The benefit is largely in terms of the reduced overhead for making a decision about core to power state mapping. Note that these techniques can be applied to reactive power management as well without much modification. In fact several of these techniques can be applied to several other areas of computer architecture as well and are the focus of authors' current and future work.

6 Distributed Proactivity for Many-cores

While several of the techniques discussed above reduce the number of evaluations to be performed for every core to power state assignment, they continue to rely on a global arbiter/scheduler to make power management decisions for each core on the processor. This is not scalable, however, for a large number of cores, as discussed in Section 4.2.3. 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.

In this section, we discuss decentralized arbitration policies that can reduce the cost of finding a proficient mapping of processor power. These decentralized methods primarily follow the divide and conquer paradigm by breaking down a large and complex global decision problem into smaller, manageable local decisions. In the extreme case, global arbitration can be replaced by a completely distributed control algorithm that ensures satisfaction of the peak power guarantee while allowing power management decisions to be made at the core level rather than the processor level.

6.1 Hierarchical Scheduling

One technique that addresses the need for distributed scheduling is the 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. 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.

Hierarchical scheduling divides the processor into several clusters of cores and performs 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

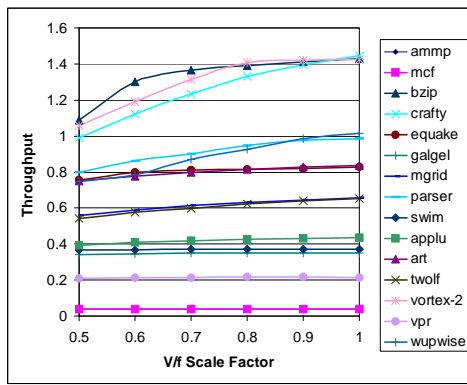


Figure 8. Performance at Various Scaling Factors.

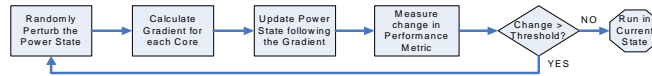


Figure 9. Gradient Ascent Power Management.

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 removed by allocating extra power to clusters in order of decreasing WS.

The algorithmic flow of hierarchical scheduling is depicted in Figure 7. Note that hierarchical scheduling may result in the decisions that are globally non-optimal.

6.2 Localized Scheduling

Hierarchical scheduling reduces the cost of arbitration by making decisions hierarchically. 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* [29].

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 the rates. Figure 8 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 be 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 9 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.

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. Surely, 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.

7 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.

7.1 Hardware/Technology Assumptions

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 [5] were used to calculate the peak power consumption of each core in various power states. 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.

$X \times V_{dd}$	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 scaling to another at a rate of 10 mV/ μ s. 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.

$X \times V_{dd}$	ΔV	Switching Time	Cycles @ 3 GHz
0.9	0.15	15 μ s	45,000
0.8	0.30	30 μ s	90,000
0.7	0.45	45 μ s	135,000
0.6	0.60	60 μ s	180,000
0.5	0.75	75 μ s	225,000

Table 3. V/f Switching Penalties.

The overhead due to V/f switching, even conservatively modeled, imposes insignificant degradation on performance. With our trigger-based sampling approach, the steady phase between successive V/f domain switches is on the order of tens of milliseconds, whereas the switching penalty for the maximum differential voltage swing is less than 100 μ s. In fact, the steady phase between switching is usually much greater, on the order of hundreds of milliseconds, since evaluation is only triggered when necessary to react to changes in application behavior. Sampling phases, which measure the performance of an application in various power states, are also on the order of milliseconds. With efficient sampling techniques, two V/f switches are required at most to characterize the performance in each possible next power state and to switch to the optimal state. In the worst case, the overhead introduced by V/f switching is less than 0.5%.

Simulation of a V/f scalable processor required some modifications to the simulation infrastructure. The baseline architecture was altered to support different supply voltage and frequency parameters for each core. These parameters are linked for each core so that changing the power state of the core affects both parameters proportionally. To model the effects of frequency scaling, pipeline processing is not executed on every cycle for all cores. Instead, pipeline stages only execute periodically, at a fraction of the full power frequency.

7.2 Workload Construction

For all simulations, each core runs a single thread for the entire simulation time. Workloads are constructed from a set of 16 SPEC2000 benchmarks. Half of the selected benchmarks are floating point applications, and half are integer applications. Table 4 lists the benchmarks used in workload construction along with fast forward distances in billions of instructions.

ampp	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.

For simulations of multi-core processors in which the number of cores is less than the number of benchmarks, workloads are constructed using the sliding window approach [28, 25]. The size of the window is equal to the number of cores in the processor. Workloads are created as the fixed-size window slides over the array of benchmarks in a circular fashion, producing a total of 16 different workloads for each simulation.

When the number of cores is greater than or equal to the number of benchmarks, a different approach is applied to workload construction to avoid testing multiple copies of the same workload repeatedly. Five workload classes were created for many-core simulations to test various aspects of our power management policies.

The first type of workload is mixed. The mixed workload contains all 16 benchmarks in equal proportions. This type of workload tests the general performance of a policy for a wide variety of applications. The second type of workload includes benchmarks that exhibit the highest sensitivity to V/f scaling, while the third workload category includes benchmarks with reduced sensitivity to V/f scaling. The fourth type of workload includes benchmarks with varying degrees of sensitivity to V/f scaling. These workloads test how effectively our power management policies determine the optimal power allocation for each application and how well our efficient, decentralized techniques perform at reducing the search overheads during subsequent evaluations. Finally, the fifth workload contains benchmarks that exhibit more dynamic behavior than others, as illustrated by figure 11. This workload class distinguishes a policy’s ability to react to the changing power requirements of applications during different phases of execution.

To choose the benchmarks for each type of workload, simulations were run for individual applications in order to gauge the dynamic behavior of the application and the level of performance variation exhibited by each benchmark in response to V/f scaling. The simulation results led to the creation of the workloads found in table 5. For each workload, four benchmarks that embody the characteristics of the desired test are selected and replicated sufficiently to create a workload that specifies a single application for each core. The mixed workload, of course, still contains all 16 of the original benchmarks. The diverse set of workload classes tests the performance of our techniques in general and also singles out key features implemented into our policies for evaluation and analysis.

Workload	Type	Benchmarks
I	mixed	all benchmarks
II	high sensitivity	crafty, galgel, bzip, wupwise
III	lower sensitivity	vpr, twolf, parser, vortex
IV	varied sensitivity	crafty, wupwise, vortex, bzip
V	dynamic	bzip, parser, applu, wupwise

Table 5. Workloads for Many-core Simulations.

7.3 Simulation Approach

Simulations are performed using SMTSIM [27] to simulate our various CMP configurations. Watch is integrated into SMTSIM to gather power statistics. SMTSIM executes statically-linked Alpha binaries.

Simpoints [23] are used to locate threads in each benchmark that best characterize the dynamic behavior of the application. Fast forward distances, generated with the Simpoint tool, are displayed in table 4. After fast forwarding, all simulations run for 1 Billion cycles.

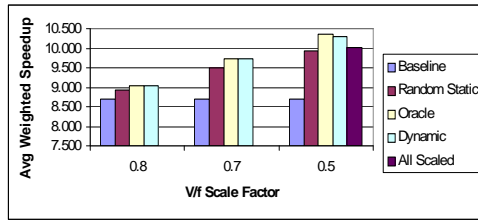


Figure 10. Performance Improvements due to Proactivity.

7.4 Evaluation Metrics

We use weighted speedup [25] to evaluate the performance of our policies. For the purpose of evaluation, weighted speedup measures a thread’s throughput relative to its throughput when running alone on a core scaled to half voltage and frequency. This metric reveals the fact that our policies aim not only to increase throughput but to decrease latency as well.

For completeness reasons, we also performed our evaluations with respect to aggregate throughput and found no significant difference in trends or analysis.

8 Analysis and Results

In this section, we demonstrate the performance benefits of our proactive approach to power management. First, we analyze the gains achieved by adding cores to the die and efficiently managing the power states of the cores. Next, we analyze how proactive decisions can be made faster through efficient investigation of the search space. Finally, we demonstrate how efficient search, combined with decentralized arbitration, can closely approximate centralized arbitration while substantially reducing the cost of decision-making.

8.1 Improving Throughput through Proactivity

In situations where a resource is limited by a fixed bound, proactivity can be used to control and maximize the utilization of the resource. In the case of peak power management, we proactively utilize available processor power to the fullest extent in order to increase performance. Table 6 describes four different processor configurations which have nearly the same peak power requirements but differ in the number of cores.

$X \times V_{dd}$	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 6. Processor Power Configurations.

For each of the configurations described in table 6, intelligent proactivity is used to match applications to power states in a way that minimizes performance degradation. A number of static and dynamic policies are described in section 4. Figure 10 shows how the average weighted speedup from each of these policies compares to that of the baseline processor.

Figure 10 shows that both dynamic and static power management techniques result in performance gains. In general, a dynamic policy should perform better than a static policy, because dynamic policies have the ability to recognize and respond to the changing power requirements of applications. However, there are several factors that may influence the effectiveness

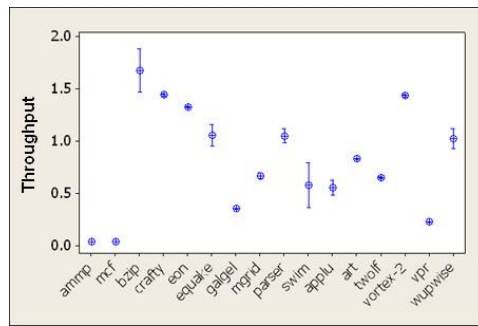


Figure 11. Throughput Variation for Benchmarks.

of dynamic policies when compared to static policies. First of all, for a dynamic policy to be effective, applications must exhibit dynamic behavior. Applied to power management, this means there must be phases of the program with varying power requirements. One example is an application that has both memory intensive and computation intensive phases. This quality may be evidenced by how the throughput of the application changes during execution. To investigate the amount of dynamic behavior present in our workloads, simulations were run for each of the benchmarks, and throughput was sampled periodically. Figure 11 shows the mean throughput recorded and displays a 95% confidence interval around the mean. The width of the confidence interval gives an idea of how much the throughput of the benchmark varies during runtime.

Another factor that influences the effectiveness of dynamic techniques is the sensitivity of performance to V/f scaling. If performance is insensitive to V/f scaling, then the amount of power allocated to a core does not matter. The performance of the application on the core will remain roughly the same no matter what power state the core is in. This fact is utilized when discerning how to map power to cores. However, the performance for a workload with many insensitive benchmarks will be nearly the same for static and dynamic policies. Figure 8 gives an idea of how sensitive each benchmark is to V/f scaling. The steeper a benchmark's performance curve is, the more sensitive the application is to V/f scaling.

The final two factors that influence the efficacy of dynamic policies are the number of cores in the processor and the penalties associated with providing the dynamic scheduling. Just as the gap between average and worst case power scales with the number of cores, so does the gap between optimal and observed performance. If an application experiences suboptimal behavior as a result of being statically mapped to an inefficient power state, the resulting performance of the processor will reflect this fact. As the number of cores increases, the small differences between optimal and observed performance multiply and become substantial. Thus, dynamic policies are necessary for peak power management in many-core architectures.

However, the implementation of dynamic power management should not result in penalties to performance which negate the positive effects of having proactivity. Our simulation methods conservatively model the penalties associated with dynamic power management to ensure that the costs of proactivity are accounted for.

8.1.1 Reactive Power Management

Reactive techniques cannot be used to provide peak power guarantees without imposing significant restrictions on processor speed. The factor by which the frequency of a processor must be scaled down to guarantee a peak power bound depends on how much time it takes to detect and react to an indicator of current overdraw. Given the limitations of present day sensor technologies, the performance degradation that must be introduced to provide guarantees reactively makes reactive

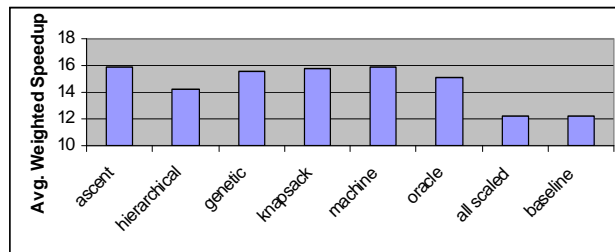


Figure 12. Performance for Many-Core Processors.

peak power management impractical. Table 7 relates the performance penalties of reactive methods relative to the baseline processor model.

Sensor Type	Scaled Freq.	Avg. Perf.	% Reduction
current	32 MHz	0.301	93.7%
voltage	125 MHz	1.121	76.5%

Table 7. Performance Degradation for Reactive Power Guarantees.

Note that these results apply particularly to peak power *guarantees*. Actual overheads might be lower, since on-chip capacitors would somewhat reduce the sensitivity of the die to instantaneous current changes. However, even if the decoupling capacitance of the chip is increased to tolerate more overdraw, no guarantees can be made unless the duration of the overdraw is guaranteed to be shorter than the time it takes to drain the capacitors. Assuming that the duration of overdraw could be limited to a single cycle, over 100 μF would be needed to sustain the additional load that could be generated by a 16-core enhanced processor. This capacitance – over 4 times the amount provisioned for by current processors – would have overly prohibitive cost and area implications.

8.2 Efficient Proactivity through Machine Learning, Search, and Modeling

The previous section demonstrated the performance advantages of our proactive approach to peak power management. However, the arbitration methods employed were inefficient and unsuitable for use in many-core processors. Throughout the remainder of this section, we demonstrate the benefits of efficient and distributed power management techniques.

Figure 12 shows the performance of the various techniques for a 16-core CMP architecture along with the performance of the baseline processor and a static oracular scheduler for comparison.

As the results show, the various proposed techniques significantly outperform the baseline as well as the static oracle. A comparison of figures 10 and 12 demonstrates that the competitiveness of static configurations such as “all scaled” decreases as core integration increases.

The use of efficient proactive techniques greatly reduces the search space of possible power configurations, thereby reducing the overhead and enhancing the scalability of our power management techniques. In one case scaling up to a 32-core processor resulted in improvements of 67% over a power equivalent baseline model.

Consider the machine learning approach, for example, in which cores are eliminated from the evaluation process as the power manager learns the optimal states for the cores. For a 16-core architecture that we studied, decisions only need 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 use of machine learning results in a 33% performance improvement over the baseline processor and gains of 22.1% over the static oracle. These improvements can primarily be attributed to the increased efficiency of the evaluation process. Specifically, cores that have already learned their optimal power state will no longer be temporarily subjected to a suboptimal state during evaluation.

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. Although the evolution-based search has the highest overhead in terms of evaluation time, this technique exhibits the best performance for the dynamic workload. The elements of randomness and refinement in the search routine allow the power manager to adapt to the changing power requirements of applications.

Modeling the search process as a knapsack problem generates a reduction in the computational complexity of the search algorithm, allowing for faster searching. While this aspect of the knapsack-based approach facilitates scalability, the approach in general performs well at the task of power management. On average, performance was increased by 32.6% with respect to the baseline processor and 21.7% over the static oracle. The knapsack approach exhibits the best performance of any policy on the workload containing benchmarks with low sensitivity to V/f scaling. The low contrast between application performance in different power states for this workload requires finer examination that may not be afforded by other approaches. The characteristics of this workload may also reduce the effectiveness of techniques such as machine learning and gradient ascent which inherently rely on performance differentiation between different power states. The knapsack approach, on the other hand, considers the processor as a whole and finds the best way to allocate available power, regardless of the variance in the the performance statistics.

8.3 Distributed Proactivity

While the techniques discussed above reduce the overhead of arbitration, decisions are still made centrally. Figure 12 shows the performance of our distributed power management techniques.

We observe that distributed techniques further enhance the efficiency of proactive peak power management for many-core processors. 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%. As the number of cores in the processor continues to grow, the complexity benefit of hierarchical scheduling will continue to grow. For the 16-core processor that we studied, the use of hierarchical power management generated average improvements of 18.7% and 7.8%, respectively, over the baseline and static oracular models. Two factors explain why these returns are somewhat less than the best observed results. First, hierarchical scheduling produces a power mapping that is locally optimal, but may be globally suboptimal. Second, in order to distinguish the effects due to hierarchical scheduling alone, the original, inefficient search methods were used within each cluster for decision-making. The performance of hierarchical scheduling can potentially be improved by coupling the decentralized arbitration methodology with efficient search methods for allocating power within the clusters.

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.

As the number of cores on the die continues to increase, the marginal benefits of our proactive power management techniques increase as well. For example, for one 32-core case, we observed improvements of up to 67% over a power equivalent baseline model.

9 Summary and Conclusions

This research introduces a novel, proactive approach to peak power management. We have demonstrated that multi-core processors that employ our proactive techniques garner significant advantages in performance and efficiency over corresponding baseline models with equivalent power and area constraints. These advantages result from the ability of a proactive power manager to provide guarantees that afford full and efficient utilization of a processor's power budget. We also propose several intelligent and decentralized power management techniques that provide increased performance and facilitate the scalability of our proactive methodology to many-core architectures.

Over our entire set of diverse workloads, our enhanced architectures averaged 30% better performance than comparable CMPs with equivalent area and power budgets. For our best arbitration policies, gains of up to 47% were observed. As the number of cores on a processor die rapidly increases, and the difference between the peak power and average power of a processor continues to grow, the effectiveness of our techniques will only continue to increase.

Acknowledgments

The research was funded in part by a gift from Intel.

References

- [1] International Technology Roadmap for Semiconductors 2005, <http://public.itrs.net>.
- [2] D. H. Albonesi. Selective cache-ways: On demand cache resource allocation. In *International Symposium on Microarchitecture*, Nov. 1999.
- [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] 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.
- [5] 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.
- [6] 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.

- [7] A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi, and P. Bose. Energy efficient co-adaptive instruction fetch and issue. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 147–156, 2003.
- [8] J. Freeman. Artificial intelligence: Simulating a basic genetic algorithm. *The Mathematica Journal*, 3:52–56, 1993.
- [9] 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.
- [10] 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.
- [11] N. Horowitz. *Power Supplies: A Hidden Opportunity for Energy Savings*. NDRC, 2002.
- [12] J. P. Hurst and A. D. Singh. A differential built-in current sensor design for high speed iddq testing. In *VLSID '95: Proceedings of the 8th International Conference on VLSI Design*, page 419, Washington, DC, USA, 1995. IEEE Computer Society.
- [13] Intel Corp. *Intel's Teraflops Research Chip*.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. Processor power reduction via single-isa heterogeneous multi-core architectures, 2003.
- [18] 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 *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*, 2004.
- [19] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. *SIGARCH Comput. Archit. News*, 26(3):132–141, 1998.
- [20] National Semiconductor. *ADC081000 High Performance, Low Power 8-Bit, 1 GSPS A/D Converter*.
- [21] D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83:394–410, 1995.
- [22] M. D. Powell and T. N. Vijaykumar. Pipeline damping: a microarchitectural technique to reduce inductive noise in supply voltage. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 72–83, 2003.
- [23] 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.
- [24] 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.
- [25] A. Snaveley 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.
- [26] D. Snowdon, S. Petters, and G. Heiser. Power measurement as the basis for power management, 2005.
- [27] D. M. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, 1996.
- [28] D. M. Tullsen, S. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22th Annual International Symposium on Computer Architecture*, pages 392–403, 1995.
- [29] T. Weyrauch, M. A. Vorontsov, T. G. Bifano, and M. K. Giles. Adaptive optics system with micromachined mirror array and stochastic gradient descent controller. In *Proc. SPIE Vol. 4124, p. 178-188, High-Resolution Wavefront Control: Methods, Devices, and Applications II*, John D. Gonglewski; Mikhail A. Vorontsov; Mark T. Gruneisen; Eds., Nov. 2000.
- [30] S. Yaldiz, A. Demir, S. Tasiran, P. Ienne, and Y. Leblebici. Characterizing and exploiting task-load variability and correlation for energy management in multi-core systems. *IEEE*, pages 135–140, 2005.
- [31] Y. Zhang, X. S. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 183–188, 2002.
- [32] S. Zhao, K. Roy, and C.-K. Koh. Decoupling capacitance allocation for power supply noise suppression. In *ISPD '01: Proceedings of the 2001 international symposium on Physical design*, pages 66–71, 2001.