A Case for Guarded Power Gating for Multi-Core Processors

Niti Madan, Alper Buyuktosunoglu, Pradip Bose
IBM T.J. Watson Research Center
nmadan,alperb,pbose@us.ibm.com

Murali Annavaram
University of Southern California
annavara@usc.edu

Abstract

Dynamic power management has become an essential part of multi-core processors and associated systems. Dedicated controllers with embedded power management firmware are now an integral part of design in such multi-core server systems. Devising a robust power management policy that meets system-intended functionality across a diverse range of workloads remains a key challenge. One of the primary issues of concern in architecting a power management policy is that of performance degradation beyond a specified limit. A secondary issue is that of negative power savings. Guarding against such "holes" in the management policy is crucial in order to ensure successful deployment and use in real customer environments. It is also important to focus on developing new models and addressing the limitations of current modeling infrastructure, in analyzing alternate management policies during the design of modern multi-core systems. In this concept paper, we highlight the above specific challenges that are faced today by the server chip and system design industry in the area of power management.

1. Introduction

Dynamic power management has now become a primary focus in multi-core systems . Multi-core systems are commonly deployed in the data centers as servers, an area that is experiencing tremendous growth. Most of these servers are on average only 10-50% [2, 3] utilized and yet due to lack of energy proportionality they consume significant fraction of the peak power. Hence, these servers exhibit significant energy-inefficiency. US Environment Protection Agency has also voiced its concerns to the Congress about the growing energy-inefficieny of data centers[16]. Therefore, it has become increasingly important to improve the power efficiency of these server systems [12, 14]. To date dynamic voltage and frequency scaling (DVFS) continues to be one of the most successfully deployed power management techniques. However, the dynamic range of (voltage-frequency) operational points is getting smaller, with the supply voltage (V_{dd}) scaling closer towards threshold voltage. As the effectiveness of DVFS decreases, predictive power-gating is emerging as an increasingly important actuation knob in chip-level dynamic power management.

Power gating is a circuit-level technique that enables one to cut off the power supply to a logic macro. Power-gating can be applied either at the unit-level, such as ALUs, pipeline stages [5, 8, 18] or at the core-level [7]. It is implemented with the help of a sleep transistor ("switch") that is inserted as a series header or footer device in the V_{dd}-to-Ground circuit path that includes the targeted macro. With the help of microarchitectural predictive control, such gating is effected, when it is deemed that the macro is *likely* to be idle for a relatively long duration. Recently, Intel's Nehalem processor family [1] has made per-core power gating available as a power management facility within a multi-core processor chip setting.

While predictive power-gating is a promising control knob for power management, it suffers from some serious limitations. With frequent mis-predictions, it can lead to significant negative impact on power-performance, since there are overheads for switching on and off a gated macro. Depending on the size of the macro that is targeted for power gating, the overhead in terms of "wake-up" latency or the power cost for turning it back "on" from a "gated off" state may be quite significant. In prior work, Hu et al.[5] have discussed the overhead costs of the power gating process in some detail, and have described ways of quantifying the so-called "breakeven point" (BEP) in terms of circuit and technology parameters. Figure 1 illustrates the concept of BEP. As shown in the figure, BEP stands for the minimum number of consecutive processor execution cycles that the gated macro needs to remain in idle state (before being woken up back to active state), in order to ensure a net positive power savings. If the wake-up occurs before BEP, it results in negative power savings. Lungu et al. [8] propose the use of "guard" mechanisms to detect the onset of such "negative benefit" scenarios, so that the main power-gating algorithm can itself be disabled in time to avoid a net increase in system power. This work was however limited to

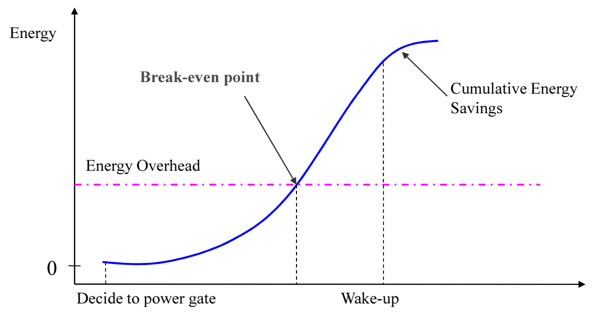

Figure 1. Break-even Point in Predictive Power Gating

unit-level power gating within a single core. We find that the per-core power-gating algorithms are also prone to similar vulnerabilities and would require guard-mechanisms as well to prevent such negative benefit scenarios.

We demonstrate that no matter how robust the baseline power gating algorithm is, it is still possible to fool the algorithm and suffer from either significant performance loss or increased core switching on/off activity that ends up dissipating more power or cause reliability failures. We advocate the need for guard mechanisms that can disable the power manager in the event of malicious power virus attack, an unpredictable workload behavior or workloads that exhibit periodic behavior that are in sync with the power management control time granularity. The power manager can be enabled back on when the system is in a stable state again. While we highlight the need for guarded power-gating, the concept of guarded power management is applicable to any dynamic power management policy. We hope this concept paper will motivate the research community to invent robust guard mechanisms that provide quality guarantees in dynamic power management. The paper is a refinement of preliminary ideas presented at a recent workshop [11].

In this paper, we also highlight the modeling challenges for power-management heuristics for many-core systems especially in the context of data center workloads. Queueing model based evaluation of data center workloads is gaining traction in the research community [4, 13]. We share our experience with such an analytical model and suggest approaches to improve its accuracy and real-workload representation.

The paper is organized as follows. We first describe our modeling framework and its limitations in Section 2 and then discuss the power-gating algorithms in Section 3 . We make a case for a guard mechanism in Section 4, and finally highlight the future research directions in Section 5.

2. Queueing Theory Based Analytical Model

Cycle-accurate full-system performance simulators do not scale well beyond a few tens of processor cores at best. As such, analytical models based on the theory of queueing systems, are a logical choice for developing a basic understanding of the fundamental tradeoffs in future, large-scale multi-core systems. Such a model can be easily applied to data center workloads that are represented by a continuous arrival stream of user tasks that wait in queue before being assigned to servers. An analytical model of this type is fast enough, that even when we study a many-core system with hundreds of cores, the speed of computation is quite manageable. Another issue with modeling of power management algorithms is the interaction with operating system (OS) scheduling policies. In a queueing model, we are able to easily add OS-appropriate time-slice support in order to enable more realistic power management scenarios. Our particular model is designed to support a generalized G/G/k queueing system. Figure 2 depicts the high-level overview of a simple queueing model, called Qute (Queueing based timing estimator) that has been implemented in C/C++ for the purposes of this study.

2.1. Model Overview

The model uses a centralized task arrival queue. The task arrival process can be modeled using any arrival distribution of choice: e.g. either the well-known Poisson process (with exponentially distributed inter-arrival time distances) or even the one derived empirically from a real, measured task arrival process at a server node. For the purposes of this paper, we assume a Poisson arrival process. Tasks are issued from the head of the central queue to waiting cores in a round robin fashion. Each core services an assigned task for a pre-determined time slice effectively modeling the OS time-quanta (which is a model parameter). If the task does not complete within that time slice then the core simply queues the task back to the tail of the centralized queue. A given task is removed from the queue once it is completed. Each task may require several time slices of processing depending upon its length. Task lengths (i.e. time durations) can either be picked from a user-specified probability distribution: e.g. Gaussian or we can use empirical data from real workload traces. There is also monitoring code in our model to keep track of the average number of utilized cores, the onset and duration of idle periods in each core and system utilization for simulating per-core power gating heuristic. Qute's task arrival queue effectively models the load balancer in a datacenter, where the load balancer receives a number of requests from clients which are in turn assigned to cores for service.

We use "Average Response Time" derived from our ana-

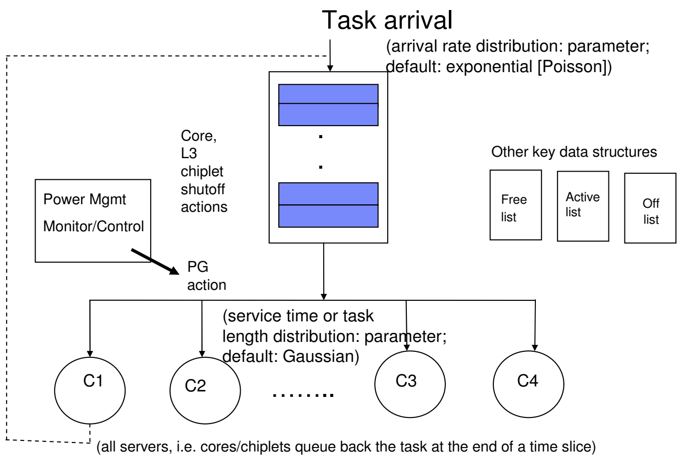
Task arrival
(arrival rate distribution: parameter; default: exponential [Poisson])
Core, L3 chiplet shutoff actions
Other key data structures
Free list
Active list
Off list
Power Mgmt
Monitor/Control
PG action
(service time or task length distribution: parameter; default: Gaussian)
C1
C2
C3
C4
(all servers, i.e. cores/chiplets queue back the task at the end of a time slice)

Figure 2. Overview of QUTE Framework

lytical model as the metric for evaluating performance. Our model computes the response time as the total time it takes from the task's arrival in the task queue to its completion. As far as the power model is concerned, we just keep track of the number of cores powered off as a fraction of the total number of cores, to estimate the power savings for any given power gating algorithm.

Table 1 shows the simulation parameters used for our experiments. All experiments run 1 million tasks which is long enough to reach a steady-state environment in our queueing model. We chose some of the parameters such as number of cores (N), mean inter-arrival time and mean task length such that our given queueing model is moderately utilized. The analytical system utilization is computed in the queueing theory using the following equation:

$$SystemUtilization(SU) = \lambda/(\mu * N) \qquad (1)$$

where λ is the mean arrival rate and μ is the mean task rate and N is the number of servers (i.e. processor cores). For our values of λ = 1/400μs , N = 256 and μ=1/50ms, the system utilization is 0.5. We can model different utilization levels in the system by varying the mean inter-arrival arrival time. Note that using this equation, it is possible to have utilization value higher than 1.0. Such a high value of utilization implies that all the cores are fully utilized and running tasks but there are several tasks queued up in the task queue.

We assume core wake-up latency (OnLat) = 1ms for most experiments unless specified. Core wake-up will require a combination of hardware/software enablements. It has been shown that core wake-up latency in linux can be as high as 100ms [7] as it also includes the time to deschedule all running processes, servicing pending interrupts and flushing caches. Therefore, in our sensitivity studies, we do consider the performance impact of a software enabled core wake-up.

Number of Tasks (M)	1000000
Number of Cores (N)	256
Mean Task Length	50ms
Mean Service time Distr	Gaussian
Mean Inter-Arrival time	400 μs
Mean Inter-Arrival Rate Distr	Exponential
Time Slice	10 ms
Wake-up latency (OnLat)	1 ms

Table 1. Experiment Parameters

2.2. Model Limitations

We next discuss the limitations of our statistical model and approaches to improve its fidelity, beyond the concept-building purpose of this particular paper.

- Real data center workload representation: While statistical workloads can be quite adequate for early stage analysis and design tradeoff studies, they may not be able to capture a real data center workload environment with full fidelity. This problem can be partly overcome by doing offline analysis of data center traces and using those empirical values in the model. We can augment this model by integrating it with real server utilization traces. We can reconstruct the measured arrival rate if we know the cpu utilization as shown in equation (1). By adopting this methodology, we can have a real data center utilization behavior. The mean task length can still be a statistical parameter in the model.

- Modeling fine-grain workload phases: The current analytical model assumes that a core is a black-box and simply executes the assigned task for a given time-slice. The core is not able to see fine-grain workload behavior such as cache misses, functional unit idle times etc. when it executes a task. Without this workload behavior view, we can not model within-core power management schemes such as DVFS or unit-level power-gating. One way to approach this is to assume that within a given time-slice, there are phase behavior distributions for unit idle times and cache miss rates. In [11], we modeled a statistical distribution for the length of time a core is in high power or low power state for studying unit-level power gating. We can improve this methodology further by incorporating each execution unit's idle-time histograms that can be obtained from cycle-accurate performance models. To model DVFS power states, we can integrate the model with a benchmark's DVFS state traces. These traces can be collected based on the methodology in [6, 15] where memory accesses per instruction completed (MPI) measurements are used to define state boundaries when a workload executes on a real system.

- Power model: We use a very simplistic power model, in making the basic arguments in this paper. In particular, the overhead power due to core wakeups can and should be estimated carefully, by taking into account the electrical power-on characteristics of the specific core-level power gating implementation.

3. Proposed Power Gating Heuristics

In this section we describe and evaluate the two baseline per-core power gating algorithms that we have explored.

3.1. Idleness-Triggered Per-core Power Gating (IdlePG)

In this heuristic, we monitor the idle duration in all cores. If a core has been idle for a pre-determined cycles or threshold C_T, we initiate the predictive gating off of this core. We assume that there is a global manager that wakes these cores up if it finds that there are tasks waiting to be executed. This is a very simple heuristic that does not suffer from much performance degradation as it quickly reacts to system load. If we choose a lower value of C_T, we get more power savings at the cost of increased performance degradation due to more frequent core wake-ups. The key parameters in this heuristic are the C_T that determines the power saving potential and *OnLat* that determines performance degradation (and consequently potential negative power savings) due to core wake-up latency.

Figures 3 and 4 show the sensitivity analysis results for this heuristic when we vary either the C_T or the OnLat parameter. The results are normalized to a baseline that

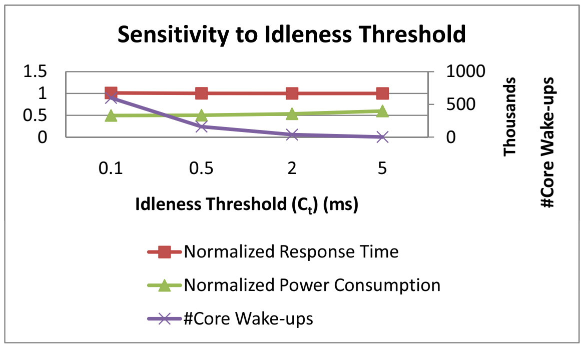

Figure 3. Idleness Threshold Sensitivity Analysis of IdlePG for OnLat = 1ms (Results Normalized to a Baseline With No Power Management)

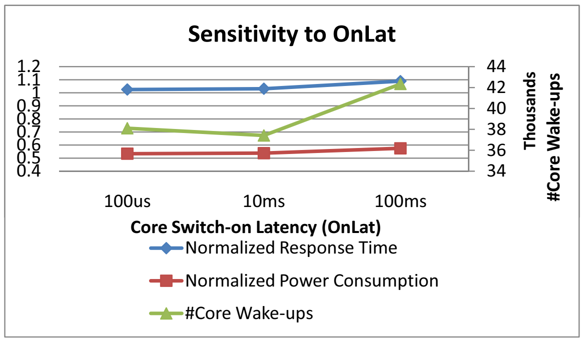

Figure 4. OnLat Sensitivity Analysis of IdlePG for C_T = 2ms (Results Normalized to a Baseline With No Power Management)

does not have any power-management policy. As expected, lower the idleness threshold, greater the power savings and higher the number of core wake-ups. For our chosen parameters, we find that the idleness threshold of 2ms yields high power savings at a moderate number of core wake-ups. The third set of experiments show sensitivity to OnLat if C_T is set to 2ms. We find that the performance degrades by 8.4% as OnLat increases to 100ms.

3.2. Utilization-based Per-core Power Gating(UtilPG)

In this heuristic, we sample system utilization every time window which is a parameter in our model ($UWIN$). We measure system utilization by using the following equation:

$$Utilization = (Number\_of\_busy\_cores + TaskQ\_entries)/Total\_cores \quad (2)$$

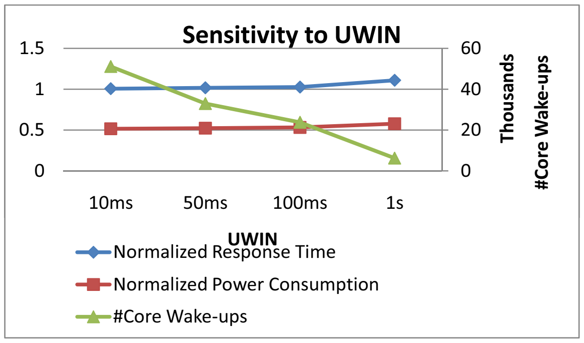

Figure 5. $UWIN$ Sensitivity Analysis of UtilPG for OnLat = 1ms (Results Normalized to a Baseline With No Power Management)

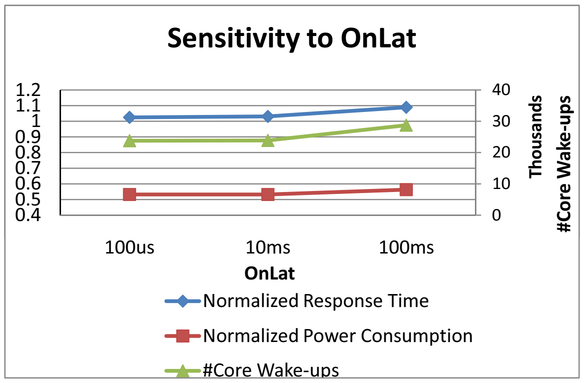

Figure 6. OnLat Sensitivity Analysis of UtilPG for UWIN = 100ms (Results Normalized to a Baseline With No Power Management)

We compute target number of cores that are required to be switched on in that interval as a function of utilization

$$(utilization + n) \times Total\_cores$$

We always provision for an extra n% cores to be on to handle unexpected changes in system load in that interval. If target number of cores is higher than the actual number of cores on then we wake up the required number of cores otherwise we switch the corresponding number of cores off. If the target is the same as the current number of awake cores, we remain in that state. This heuristic is more conservative in saving power as it always provisions for extra cores to be on in each time window. If the time-window $UWIN$ is very long then we may end up hurting performance if the system utilization changes in the middle of a time window.

Figures 5 and 6 show the sensitivity analysis results when $UWIN$ and OnLat are varied. We assume n = 5% in these experiments. We normalize these results to a baseline that does not implement power gating. We find that both the response time of the system and power savings are very sensitive to $UWIN$. If the utilization time window is very long where $UWIN$ = 1s, then the performance degradation can be as high as 11%. The number of core wake-ups is also dependent upon $UWIN$. If $UWIN$ is short, then there is more opportunity to switch cores on/off. Utilization window of 100ms represents an ideal point for these statistical distribution values that gives decent power savings with an acceptable number of core wake-ups. We find that very high core wake-up latencies can degrade performance by up to 9% as in the case of IdlePG heuristic.

PG Heuristic	Normalized Response Time	Normalized Power Consumption	Number of Core Wake-ups (M = 1 million tasks)
IdlePG (C_T = 2ms)	1.000	0.534	37760
UtilPG ($UWIN$ = 100ms)	1.026	0.532	23780

Table 2. Comparison of IdlePG and UtilPG for OnLat = 1ms (Results Normalized to a Baseline With No Power Management)

3.3. Discussion

Table 2 compares the two power gating heuristics normalized to a baseline that does not have any power gating. In general, IdlePG demonstrates better power-performance trade-offs when compared to UtilPG. But in terms of the overheads of core wake-ups, UtilPG has superior behavior. In terms of implementation complexity, UtilPG can be implemented in software, with only a small amount of hardware support in the form of providing runtime power monitoring facilities (e.g. [6]; see also the power proxy architecture description for POWER7[17]). However, such a centralized power management algorithm may be fundamentally limited by scalability (in terms of the number of processor cores, N). As N increases, the sense-and-actuate control loop tends to be limited by the bandwidth requirements of per-core and system-wide sensing of power and utilization metrics. Similarly, the verification complexity of such a global, multi/many-core power management protocol tends to blow up exponentially with N [9]. On the other hand, IdlePG relies on local utilization for predicting power gating, and such control can be implemented in hardware through a simple state machine controller per core. The power-performance benefits and the verification complexity, scale linearly with N in such a scenario in the ideal case. However, when it comes to per-core power gating, it is not possible to rely entirely on an autonomous, distributed hardware controller only. System software (i.e. the OS and hypervisor) must be involved in any decisions about turning off cores or powering up idle cores. Thus, even in the case of IdlePG, there will be a component of the power management algorithm that is ”global” and implemented in software. Overall, the engineering challenge is to find the right balance between hardware support and software control, that provides an acceptable degree of power-performance scalability, while keeping the verification complexity under practical limits. In practice, therefore, some kind of a hybrid power gating algorithm that uses elements of the UtilPG and IdlePG concepts may prove to be most beneficial as a baseline power manager. Later in this paper, we bring in another dimension of the problem: namely, robustness of the power gating algorithm, when it comes to natural or maliciously-induced workload variations. For the scenarios explored, we show that IdlePG is more robust than UtilPG. Of course, this paper represents an initial study of the fundamentals of core-level power gating; we do not claim to have covered the entire feasible design space of such algorithms.

4. A Case for Guard Mechanism

In this section, we examine the power gating heuristics described in the previous section and study their behavior towards certain workload conditions that make these heuristics vulnerable to power-overruns or severe performance degradation. There may be multiple ways to break these robust power gating algorithms but we focus on one obvious case where the workload task arrival pattern exhibits a periodic loop. This kind of workload behavior could either be deliberate in the form of power-virus attack or it could simply be the real nature of that workload. We expect that data center workloads typically have long periods of low utilization or a certain steady arrival rate followed by bursty high utilization periods. There may be a loop kind of a pattern even in data center workloads and even if that is the case, we believe that it may not be safe for servers to actuate their power-gating knobs. We will construct this scenario for each of the two baseline power gating algorithms and make a case for guard mechanism.

4.1. Vulnerability Example of Idle-triggered Power Gating Heuristic

Figure 7 shows how our baseline idle-triggered heuristic would react to a system load that is changing periodically. The task inter-arrival time is toggling between 200μs and 3000μs with a period of 50ms and our power gating algorithm's period is 2ms based on C_T. As shown in this plot, when the power gating algorithm reacts to the system load by switching on cores, the system utilization changes again due to change in mean inter-arrival time. This causes repeated switching on and off of cores that is highly undesirable and may cause negative power savings. Figure 8 shows the performance degradation caused by our power gating algorithm as a function of different core switch-on latencies. If we assume a rather aggressive value for the overhead (1ms) based on a hardware-centric approach to power gating then the performance impact is not much but the number of core wake-ups is 21X the baseline. However, if we assume a very high latency such as 100ms and if the software does the core wake-up then the performance degradation can be as high as 108%. Note that the number of wake-ups decreases as the wake-up latency increases. This happens because the cores are not idling for the duration they are experiencing waking up delays.

4.2. Vulnerability Example of Utilization-based Power Gating Heuristic

Figure 9 illustrates how the utilization-based power gating heuristic reacts to a periodically toggling system load. We consider an extreme case in this example where the toggling period of inter-arrival time is the same as our utilization time window ($UWIN$). When a new time window begins, the power gating manager decides to switch off on average 216 cores based on current low utilization level (5%) and inter-arrival time of 3000μs. Once the cores are switched off, the inter-arrival time decreases to 200μs and the utilization increases to almost 100%. This requires the cores to be woken up in the next time window. We again toggle and increase the mean arrival time to 3000μs causing the utilization to drop. This results in incorrect power gating decisions to take place at the start of each utilization time window. As expected in this scenario, there will be significant performance penalty as the number of cores available remain the same for the duration of $UWIN$. Figure 10 shows the normalized response times for different core switch-on (OnLat) latencies when compared to a baseline utilization-based heuristic that is experiencing steady arrival rate. We find that the utilization-based heuristic suffers from performance degradation as high as 100% for even a conservative OnLat value of 1ms. The degradation can be 208% for a very high OnLat latency.

The abovementioned case study demonstrates that the utilization-based heuristic is less robust when compared to the idlePG heuristic. To improve the robustness of the baseline UtilPG heuristic, we added a history-based enhancement to the heuristic. We observed that adding some level of utilization history does reduce the degree of performance degradation during unsafe workload conditions but it still does not eliminate the problem. For lack of space, we do not present the history-based enhancement ideas in full detail in this paper.

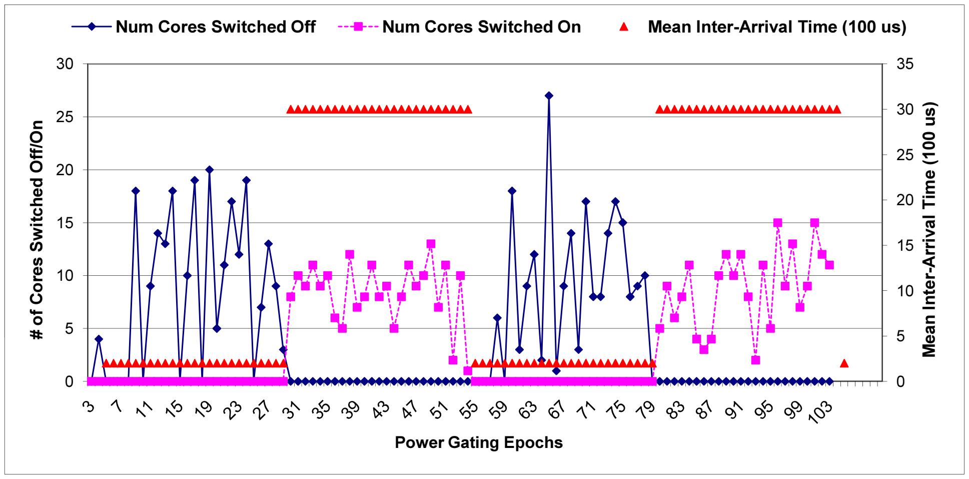

Figure 7. Impact of periodic arrival-rate on IdlePG Heuristic (C_T = 2ms)

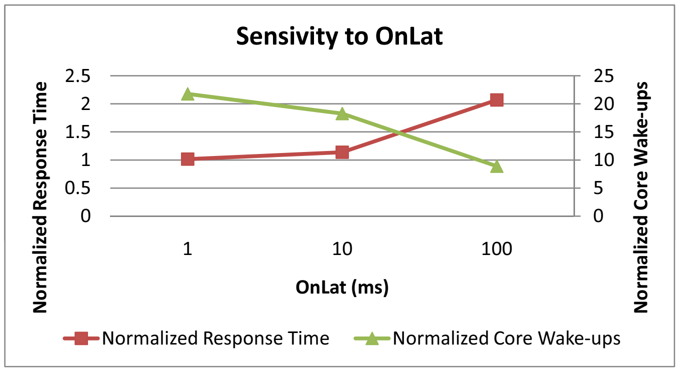

Figure 8. Performance Impact Normalized to IdlePG with Steady Mean Inter-arrival time

4.3. Guard Mechanism

We have discussed how both the baseline power gating heuristics (IdlePG and UtilPG) are vulnerable to holes triggered by spontaneous or malicious "corner-case" workload patterns. The net effect of such holes is to make the underlying machine susceptible to large, unpredictable performance degradation and/or negative power savings. We find that IdlePG is more robust than UtilPG, since the performance degrades more gracefully in the former case. There are two approaches to solve the problem of algorithm robustness in such a scenario.

The first approach is to try to improve and enhance the baseline algorithm to the extent that the number of potential "holes" is minimized, possibly to zero. The problem with this approach is that it is hard or impossible to anticipate all workload scenarios for a multi/many-core platform; hence, devising a provably robust algorithm that is guaranteed to work without manifesting any vulnerable "hole" is very difficult.

The second approach is to devise a "guard" mechanism that watches over the baseline algorithm and takes protective actions to prevent unwanted machine behavior. The easiest protective action would be to disable the baseline algorithm completely. This would, by construction, prevent any drastic loss of performance or an unbounded increase in power consumption. In this paper, we make a case for this latter class of "guard" mechanisms. We envisage such two-level, guarded management protocols to provide quality guarantees that a baseline algorithm cannot.

Apart from quality guarantees, a guard mechanism may help even in reducing the overall verification complexity of the power gating algorithm. A highly sophisticated baseline algorithm that anticipates corner case scenarios and provides built-in safeguards, will require very elaborate verification methods to make sure that all possible corner cases have been adequately covered. In contrast, having a guard mechanism obviates the need for a very sophisticated baseline management algorithm. The latter can, in fact, be a very simple one, that is designed without having to worry

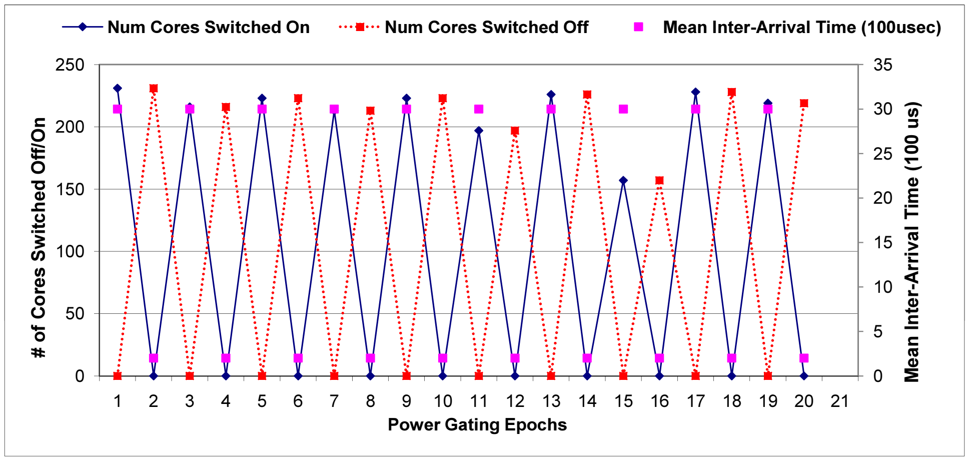

Figure 9. Impact of periodic arrival-rate on UtilPG Heuristic ($UWIN$ = 100ms)

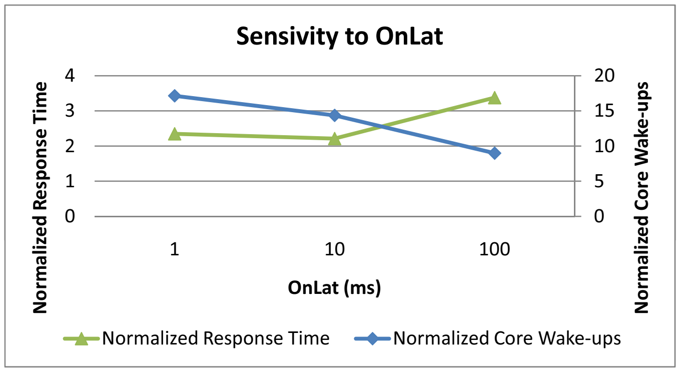

Figure 10. Performance Impact Normalized to UtilPG with Steady Mean Inter-arrival time

about rare corner case scenarios. The guard mechanism just looks for "problem" behaviors at the machine level, without having to worry about the inner details of how the baseline algorithm actually works. If the machine behavioral signatures trigger an alarm, the guard simply disables the baseline power manager. Thus designed, the guard mechanism is portable across alternate versions of the baseline management algorithm. Since both the baseline and the guard algorithms are relatively simple, the overall pre-silicon verification complexity is likely to be significantly *less* than that required for a very sophisticated, all-protective baseline management algorithm.

In recent prior work, Lungu et al. [9, 10] have trailblazed a visionary set of ideas about how to design microarchitectures and associated power management algorithms with verification cost in mind from the very outset. In particular, the work in [9] discusses the verification cost for multi-core power management (DVFS) algorithms. It is shown that centralized ("global") DVFS control algorithms increase exponentially with the number of cores, N, when it comes to verification cost. Also, such complexity explodes as the number of valid voltage-frequency steps increases. We would therefore expect that the verification complexity of multi-core power-gating algorithms is bounded to reasonable levels since there are only two power states: ON and OFF. Nonetheless, as already discussed in section 3.3, the challenge of verifying a power gating algorithm as it scales to increasing number of cores, is still of huge concern. Therefore, as indicated above, the promise of guarded, two-level management algorithms in curbing the verification complexity curve for multi- and many-core platforms, is an attractive proposition.

Finally, we would like to point towards recommended guard mechanism architectures. Figure 11 illustrates how we envisage the guard mechanism's embodiment. Guard mechanism should have monitors that sense negative power savings or performance degradation beyond a certain threshold. These monitors can be implemented at either the hardware or software level. If frequent violations are detected, then the guard mechanism should be able to disable the power manager. We may also need a monitor that is able to sense when to enable back the power manager if

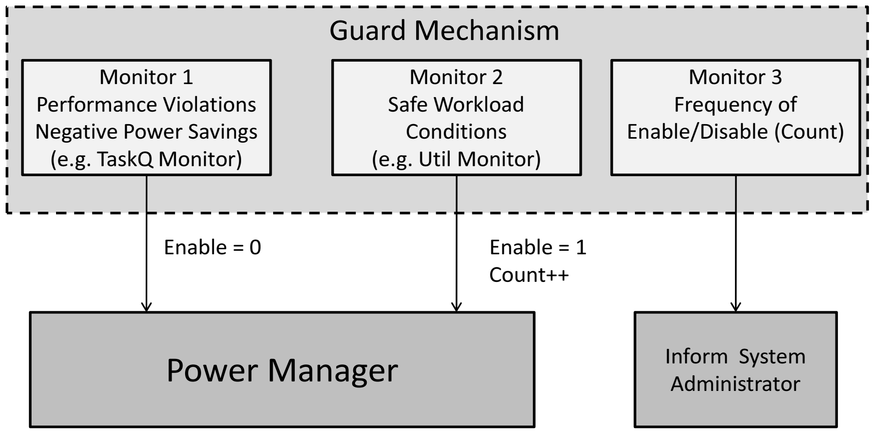

Figure 11. Guard Mechanism

the workload conditions change and become stable again. The other alternative is to support a very large time-out and re-enable back the power manager. If we are frequently enabling/disabling the power manager then the guard mechanism needs to inform the system administrator of a possible malicious power virus attack or unsafe workload conditions for power management. The sampling intervals of the various monitors also play a significant role in correct working of guard mechanism. We propose one sample guard mechanism that can help detect these violations especially in the utilization-based power gating heuristic. We propose having two monitors in this approach, a task-queue monitor and a utilization monitor. We monitor the task queue to see if there are enough tasks waiting for cores to be free while there are several cores that are switched-off. Under normal load circumstances, this scenario should not occur for a long duration. We monitor the task queue at a finer granularity than the utilization time window $UWIN$ and if the queue is filling up beyond a certain threshold for a long enough period, we disable the power manager. To enable back the power manager, we monitor the system utilization at the same granularity as $UWIN$. If the system utilization is in a steady state and not changing frequently, we enable the power manager.

5. Conclusions and Future Work

In this paper, we present a new challenge being faced in the systems industry. Even though many future processor and system products are expected to have some form of dynamic power management policy, there is no guarantee that the policy will work well when actually deployed in the field. We show how even the most robust baseline dynamic power gating heuristics can fail to provide power-performance guarantees.

We would like to encourage the academic research community to explore the following research directions in advancing the state of the art in power management:

- Continue to explore power management policies that especially scale well as the number of cores increase in future technology generations. We also need to make these power management policies more robust. It is imperative to consider the verification effort of these policies and focus on reducing the complexity of the proposed schemes.
- Architect efficient guard mechanisms that can guarantee power and performance bounds in the power management policy. The possible solution space for guard mechanisms can cover hardware, firmware, software or a combination. It is also worth exploring guard mechanisms that work without a priori knowledge of underlying power management policies. Researchers should consider designing “portable” guard mechanisms that work for any platform and any successive future generation of the same product.
- Develop better models and methodology to study power management policies. We need to improve the accuracy of analytical models.
- While this position paper advocates the need for guard mechanisms for power management, we believe that this problem is applicable to any dynamic management scheme. We need to eventually think about guarded computing.

Acknowledgements: The authors wish to thank Dr. Sriram Vajapeyam for developing the initial version of the Qute model that forms the basis of the analysis framework in this study. The work reported in this paper is supported in part by the National Science Foundation under Grant #0937060 to the Computing Research Association for the

CI Fellows project that supports the first author and Grant #0834798 that supports the last author.

References

[1] First the Tick, Now the Tock: Next Generation Intel Microarchitecture (Nehalem). Technical report, Intel Whitepaper, 2008.

[2] L. Barroso and U. Holzle. The Case for Energy-Proportional Computing. December 2007.

[3] X. Fan, W. Weber, and L. Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of 34th ACM/IEEE International Symposium on Computer Architecture (ISCA-34)*, June 2007.

[4] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. Optimal Power Allocation in Server Farms. In *Proceedings of SIGMETRICS*, June 2009.

[5] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural Techniques for Power Gating of Execution Units. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, August 2004.

[6] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *Proceedings of 39th ACM/IEEE International Symposium on Microarchitecture (MICRO-39)*, December 2006.

[7] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power Management of Datacenter Workloads Using Per-core Power Gating. *IEEE Computer Architecture Letters*, vol.8(2), July-December 2009.

[8] A. Lungu, P. Bose, A. Buyuktosunoglu, and D. Sorin. Dynamic Power Gating with Quality Guarantees. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, August 2009.

[9] A. Lungu, P. Bose, D. Sorin, S. German, and G. Janssen. Multicore Power Management: Ensuring Robustness via Early-Stage Formal Verification. In *Proceedings of 7th ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, July 2009.

[10] A. Lungu and D. Sorin. Verification-Aware Microprocessor Design. In *Proceedings of PACT*, September 2007.

[11] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram. Guarded Power Gating in a Multi-core Setting. *Presented at WEED-2 Workshop held in conjunction with ISCA-35*, June 2010.

[12] D. Meisner, B. Gold, and T. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of ASPLOS-XIV*, March 2009.

[13] D. Meisner and T. Wenisch. Stochastic Queuing Simulation for Data Center Workloads. In *Proceedings of EXERT Workshop held in conjuction with ASPLOS*, March 2010.

[14] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "Power" Struggles: Coordinated Multi-level Power Management for the Data Center. In *Proceedings of ASPLOS-XIII*, March 2008.

[15] R. Sarikaya, C. Isci, and A. Buyuktosunoglu. Runtime Workload Behavior Prediction Using Statistical Metric Modeling with Application to Dynamic Power Management. In *Proceedings of International Symposium on Workload Characterization (IISWC)*, December 2010.

[16] U.S. Environmental Protection Agency - Energy Star Program. *Report To Congress on Server and Data Center Energy Efficiency - Public Law 109-431*, 2007.

[17] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. C. Rubio, F. Rawson, and J. B. Carter. Architecting for Power Management: The IBM POWER7 Approach. In *Proceedings of 16th International Symposium on High Performance Computer Architecture (HPCA)*, January 2011.

[18] A. Youssef, M. Anis, and M. Elmasry. Dynamic Standby Leakage Prediction for Leakage-tolerant Microprocessor Functional Units. In *Proceedings of International Symposium on Microarchitecture (MICRO)*, December 2006.