IdlePower: Application-Aware Management of Processor Idle States

Hrishikesh Amur Ripal Nathuji Mrinmoy Ghosh

Karsten Schwan Hsien-Hsin S. Lee

Center for Experimental Research in Computer Systems
Georgia Institute of Technology
Atlanta, Georgia 30332.
{*hamur3, schwan*}*@cc.gatech.edu* {*rnathuji, mrinmoy, leehs*}*@ece.gatech.edu*

ABSTRACT

Power has become the first class design constraint in modern processor design. To reduce the power density caused by aggressive, speculative execution seen in previous processor generations, computer architects have turned to a multicore design strategy with each core substantially simplified. Additionally, different power-saving features have been proposed and integrated into each core to adapt dynamic execution scenarios. Due in part to the independent nature of these cores, the power management has also become more flexible to further reduce the overall power consumption. With careful adaptation schemes, the system can save power by entering different idle states dynamically with minimal performance impact. Given the simultaneous emergence of virtualization technologies, the question, then, is how to effectively leverage these idle states in the context of multiple virtual machines (VMs) executing on multicore parts. Towards this end, we develop the IdlePower approach to managing idle states in virtualized systems. Our approach combines a novel batching algorithm that creates improved opportunities to enter deep idle states by removing unnecessary system wakeups depending upon monitored behavior of workloads. IdlePower also provides application awareness in another fashion by not only entering deep idle states based upon transition latencies, but also factoring in the performance degradation that can occur due to secondary effects such as data loss in cache structures. We extend the use of Bloom filters with IdlePower to detect application characteristics for dynamically predicting whether deep idle states are worthwhile based upon possible performance implications. Overall, IdlePower is shown to improve residencies in the deepest C3 idle state by up to 10%, and to avoid performance degradations in workloads of up to 26%.

1. INTRODUCTION

To continue on the trajectory of Moore's Law without aggravating design and verification complexity, processor architects turned to multicore design to improve performance by exploiting thread level parallelism. On the other hand, as power becomes the first-class design constraint in processors, the design of each core in a multicore has been substantially simplified. Various power-saving features, seen in a modern processor to effectively perform online power management, have become indispensable to maximize the computational efficiency.

Research in this domain, focused on mobile and embedded systems, considers limitations in battery capacities and demands for longer device lifetimes [11, 28, 33, 34]. In high performance and enterprise systems, power management has been driven by two significant issues. First, there are limitations in the power delivery to and cooling capabilities of large scale machines. For example, provisioning 60 Amps per rack can become a bottleneck for high density configurations enabled by small form factor blade systems. Second, there are financial implications of power and cooling in these environments. A facility consisting of 30,000 square feet and consuming 10MW, for instance, requires an accompanying cooling system that costs from $2-$5 million [20], and the yearly cost of running this cooling infrastructure can reach up to $4-$8 million [27].

The trends described above dictate the active management of power hungry components such as processors in end platforms. Nonetheless, such management must take into account another facet of modern systems, which is that they routinely use virtualization supported by hardware [24] and/or software solutions (e.g.,Xen [5] or VMware [30]) in order to attain basic benefits like fault [5] and performance isolation [16] and resource consolidation via seamless VM migration across different physical machines [8]. It is inevitable that power management solutions must operate in the context of system virtualization.

This paper considers power management for multicore processors in virtualized system platforms. We build on conventional power management strategies designed to use processor support for dynamic voltage and frequency scaling (DVFS) during active execution. These different voltage/frequency modes, known as ACPI defined P-states [13], balance performance and power of a part while a workload is executing. The novel element considered in this paper is that due to increased pressure for efficient idle modes, modern processors also incorporate additional sleep, or ACPI C-states, which can be used during extended idle periods. Specifically, given the availability of multiple, deep C-states in modern platforms, we are concerned with idle power management leading to our technique – *IdlePower*, where our technical contribution is to evaluate the ability to attain significant power savings by aggressively utilizing C-state switching during short idle periods of workload execution, possibly coupled with DVFS methods.

IdlePower provides an approach and an architecture for idle power management, including a novel algorithm that batches timing interrupts to increase deep C-state residency as an increasing number of VMs are aggregated onto a multicore part. The batching scheme leverages the observation that most commodity VMs require timing interrupts from

the hardware to keep their clocks synchronized. The IdlePower batching algorithm detects idle periods in the execution of the applications inside the VMs and then batches timing interrupts during these periods to extend the deep C-state residency of the processor. While this part of the algorithm is straightforward, an interesting outcome of analyzing IdlePower's aggressive C-State usage relates to the manner in which deep C-states are implemented. A particular issue here is that deep C-states power down the L2 cache, with resulting potential impacts on application performance. Specifically, since deeper C-states turn off first the L1 and then the L2 caches [14], the cache contents are lost each time the state is entered. While the effect of reloading the cache is not felt for extended idle periods, doing so during shorter transient idle periods observed during the execution of a workload can cause significant performance hits. Therefore, for workloads that suffer significant performance hits due to cache flushing, it is desirable to temper the aggressiveness of the C-state switching algorithm. Toward this end and to support dynamic C-state management based upon workload specific characteristics, the IdlePower architecture is extended with a novel hardware extension. Namely, using Bloom filters [6], IdlePower obtains a signature of the cache and observes the rate of change of cache contents during the execution of a workload. This information is then used to directly scale the aggression of the IdlePower C-state management algorithm.

Experimental evaluations were made in two stages. Measurements for the batching algorithm and the degradation of performance due to cache flushing were conducted on an Intel Core Duo machine with 1Gb of memory, which was running the KVM hypervisor [1]. Measurements for the dynamic assessment of a workload using Bloom Filters were made using Simics [4], a full system emulator to emulate an x86-based virtual machine with a modified *g-cache* module to implement the Bloom Filter Signatures infrastructure.

In summary, this paper evaluates the tradeoffs and effects of C-state management on multicore architectures in virtualized systems. Contributions include:

- The IdlePower batching algorithm that consolidates timing interrupts and increases the deep C-state residency of the processor as shown with experimental results attained on modern multicore platforms.
- An experimental analysis of the effects of cache flushing due to aggressive C-state switching, using synthetic workloads.
- A hardware extension for IdlePower consisting of Bloom filters to obtain signatures of cache contents for determining the suitability of an application for aggressive C-state switching. Simulation results demonstrate the effectiveness of the approach.

Experimental evaluations of the IdlePower batching algorithm illustrate improvements in C3 state residency of up to 10% for web server workloads running in commodity VMs. The resulting exploitation of small idle periods, however, can cause the interesting performance issues highlighted above. In our analysis, for instance, we find that certain workloads suffer up to 26% performance degradation due to loss of cache contents during execution. Using Bloom filters, we then show that workloads that are unduly affected by deep C-states can be identified dynamically. The outcome is an argument for creating IdlePower C-state policies that use application-awareness to scale their aggressiveness based on feedback from these Bloom filters.

The remainder of the paper is organized as follows. Section 2 discusses related work, and is followed by Section 3 that motivates the problem of timing interrupts and their effects on deep C-state residency. Section 4 discusses the issues that arise due to aggressive switching. Section 5 discusses the architecture of our system and is followed by experimental analysis in Section 6. Finally, we conclude in Section 7.

2. RELATED WORK

Since the dynamic power consumption of a CPU is proportional to the product of frequency and voltage squared [21], dynamic voltage and frequency scaling (DVFS) can be effective in reducing power consumption during program execution [31, 26]. When DVFS is used during memory bound phases of workloads, power savings can be achieved with minimal impact to workload performance [18, 15]. In other cases, increased lengths of active periods due to frequency scaling can affect the ability to utilize processor and device sleep states. Recent work has shown that platform level tradeoffs such as underlying power management schemes or idle power management mechanisms can create a dynamic tradeoff between use of frequency scaling versus idle state [19, 10, 22]. This paper considers idle power management, to explore how recent advances in providing deep C-states for mobile and server processors can further extend processor manageability.

A substantial amount of research for exploiting idle periods is concerned with communication devices. The benefits of a separate low power channel [29] to determine when to turn off these devices has been investigated. Kravets and Krishnan [17] modify the 802.11 protocol at the client and base station to develop a collaborative approach to putting a wireless device to sleep. Mechanisms for energy-aware resource usage based upon novel I/O interfaces have also been proposed [32]. In comparison, IdlePower focuses on performing aggressive C-state management of processors during short bursts of idle periods.

At the system level, Chase *et al.* [7] discuss how to turn servers on and off based on demand in datacenters. Methods for performing server management in an application-aware manner in virtualized systems have also been introduced [23]. With respect to toggling processors between on and off states, Diao *et al.* [9] use learning methods to predict CPU idle patterns across multiple cores and switch into deep C-states based on these predictions. Pallipadi *et al.* [25] use a simpler technique, where they use the knowledge of the next timer deadline in the scheduler to match with C-state latencies and transition to the deepest possible one. However, both of these fail to address the associated issues of lost cache state and the resulting performance implications from switching to deeper C-states. IdlePower addresses these issues.

3. TECHNICAL BACKGROUND

Since average system workloads rarely saturate available hardware due to both the nature of traffic as well as overprovisioning for peak workloads, strategies for idle power management become increasingly important. Recent strate-

gies to manage power have been to consolidate all VMs running workloads onto a small number of physical machines. However, for web-based workloads and for streaming applications, there also exist opportunities to exploit the smaller idle periods that occur during the execution of the workload, particularly in lieu of the fact that industry is ever-shortening the transition times between different processor states.

Short idle times can be exploited through the use of ACPI-defined C-states[13]. There are a series of states from C_0 to C_n. When the processor is active, it runs in C_0, but when it is idle, it can switch to progressively deeper C-states, which are characterized by greater transition latency and lower CPU power consumption. The processor remains in such idle states until a break event causes it to return to C_0. Deeper states are able to save more power by using methods that include turning off the caches and disabling the internal Phase Locked Loops [14].

Past work has based decisions concerning C-state transitions on transition latency. An issue with this simple approach is that most current operating systems operate in a 'tickful' manner. As a result, when multiple idling VMs run on the same processor, these VMs depend on a periodic interrupt or "tick" from the underlying hardware to keep up their clocks. The unfortunate outcome is that the processor is unable to transition to a power-saving deep C-state.

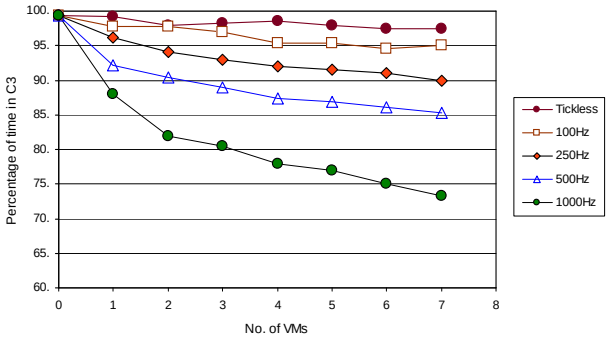


Figure 1: C_3 residency with idling VMs.

Figure 1 shows the effects of running multiple VMs on C_3 residency, measured using a utility for Linux called PowerTop [3]. Higher clock frequencies of the VMs cause the processor to be woken up more frequently and the result is a decrease in C_3 residency. The prevalence of commodity tickful systems means that this cannot easily be dismissed, and techniques are needed to reduce the large number of unnecessary wakeups. Also we measure C_3 residency in this set of experiments since we observed the residencies in all except C_3 (deepest) and C_0(active) to be negligible(less than 3%). The active power state for this set of experiments was set to the lowest performance state, to factor out the effects of P-state changes on power consumption.

To deal with tickful OSs running on C-state-capable platforms, IdlePower uses the technique of batching interrupts to decrease the number of processor wakeups and increase the residency times in deeper sleep states. Specifically, its batching algorithm first detects when a particular running VM is simply idling without performing useful work. It then marks this VM as eligible for interrupt batching. Such batching

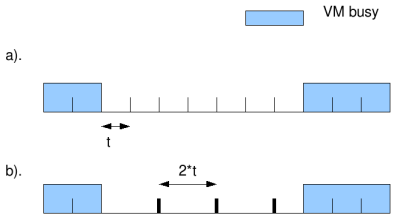


Figure 2: Batching of VM time-keeping interrupts.

essentially accumulates the clock interrupts that have to be sent to a VM and then dispatches them together. For example, in Figure 2, the activity of the VM is defined to be busy in the ticks indicated by the shaded rectangles, and batching is disabled during these periods. If the batching factor decided by the algorithm is n, which is equal to 2 in this case, and for the VM time period t, the algorithm sets a timer for $t * n$. After this period, n interrupts are sent in a single batch. In this fashion, a VM clock depending on an interrupt to increment a timer value is updated properly.

An innovative element of the IdlePower batching algorithm is its observation of the execution patterns of VMs to decide when batching can be enabled for each VM. Toward this end, each timeslice for a VM is classified as busy or idle, using a heuristic based on the time spent on the CPU by the VM during any given timeslice. A history of this data is maintained and is used to enable and disable batching.

As expected, batching decreases the number of processor wakeups and increases deep sleep state residency. Figure 3 compares the change in residency of C_3 state for clock frequencies of 500Hz, 1000Hz and 1000Hz with batching enabled, called 1000Hz-batch. As evident from the figure, with a batching factor of 2, the C_3 state residency for 1000Hz-batch almost matches that of 500Hz for a small number of VMs and is close to 15% more than the 1000Hz case for a higher number of VMs.

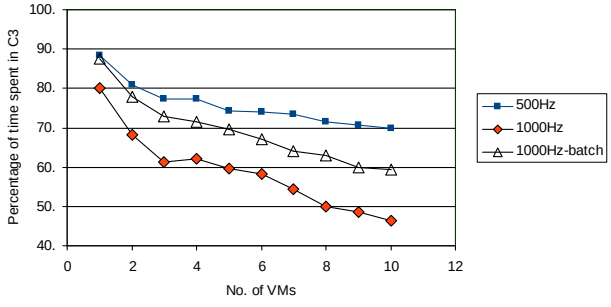
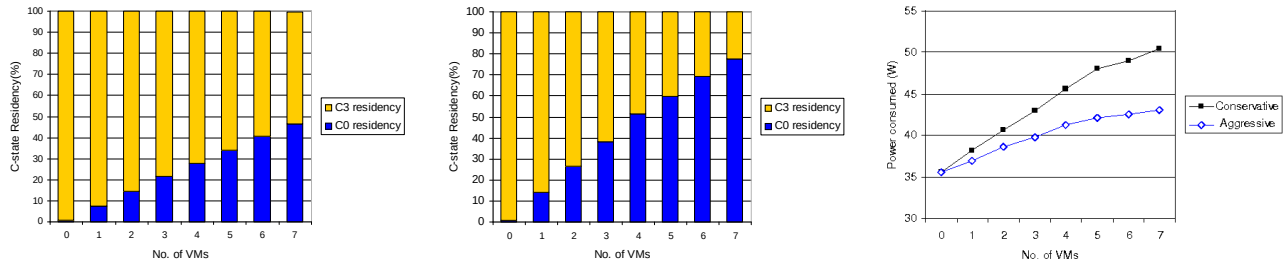


Figure 3: Increase in C_3 residency due to batching. A batching factor of 2 used which improves the residency for 1000Hz.

In the next section, we highlight some of the issues that need to be addressed because of batching and idle power management in short idle periods in general.



(a) Conservative P-State Governor (b) Aggressive P-State Governor (c) Power Consumption

Figure 4: Race to idle not always beneficial

4. IMPLICATIONS OF SHORT IDLE PERIODS

4.1 Interplay with P-state Algorithms

The batching approach to idle power management, characterized by attempts to increase deep C-state residency times, works well when VMs are idle for extended periods of time. It is not clear, however, whether such an approach can successfully reduce power consumption by leveraging the smaller idle periods found in modern service-based applications, such as web server and multimedia workloads. An important reason for this is the interplay between batching and active power management strategies like DVFS algorithms. For instance, there has been prior work regarding the tradeoffs between running at high-active-power (and hence performance) states to quickly reach an idle period vs. carrying out required work by running at low-active-power states [19, 10, 22]. The appropriateness of each of these strategies depends on the workload and on the power management capabilities of the hardware and may be difficult to determine, but it is clear that a simplistic race-to-idle strategy will not work with all workloads. Figure 4, for instance, compares the use of aggressive vs. conservative P-state governors with an aggressive C-state algorithm for the compression utility *tar* workload. The aggressive P-state governor always causes the processor to run in the P-state characterized by lowest frequency and voltage, while the conservative governor always opts for the highest performance P-state. Figure 4(a) shows the case with the conservative P-state governor and Figure 4(b) the aggressive case. It can be seen that although the proportion of time spent in C_3 state is much higher in the conservative case this does not translate to lower power consumption as shown in Figure 4(c). These results highlight how IdlePower should carefully leverage C-states in the context of P-states. In view of the considerable amount of past work that has already developed techniques for active and idle power management [19, 10, 22], in the remainder of this paper we focus on the previously unconsidered implications of deep C-states on caches.

4.2 Performance Implications of Cache Flushing

The important question a C-state switching algorithm must answer is what C-state to transition to at a given point of time when the processor becomes idle. Recent approaches to answering this question involve either predicting CPU execution patterns [9] or using advance information about timers [25] to match with the expected hardware dependent transition latency and thus choose the deepest C-state whose transition latency is within the estimated idle time. An important factor neglected by such methods is the effect on caches. One of the techniques used by deep C-states to save power is to turn off cache levels progressively, which results in loss of cache state. This can lead to significant performance degradation, which may not be justified by the power savings obtained. In fact, for our workloads, performance degradation of up to 26% was seen. IdlePower addresses this issue by using Bloom filters to dynamically assess the suitability of aggressive C-state switching.

5. IDLEPOWER ARCHITECTURE

5.1 Counting Bloom Filters

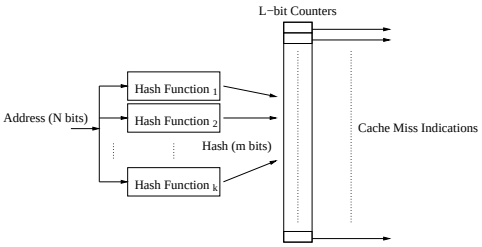


Figure 5: Counting Bloom Filters.

The original purpose of Bloom filters as proposed by Burton Bloom in [6] was to build memory efficient database applications. We have taken this idea, and extended it as a hardware component to develop a Counting Bloom filter (CBF) for monitoring workload behavior with IdlePower. The overall structure of our Counting Bloom filter is shown in Figure 5. The CBF utilizes a series of observed addresses to update a set of counters which make up a Bloom filter bit vector. An address of N bits is hashed to k hash values using k different hash functions. The output of each hash function is an m-bit index value that addresses a Bloom filter bit vector of 2^m elements. Naturally m is much smaller than N. Initially the entire Bloom Filter bit vector is set to zero. Only the bits which are accessed by the hash values generated are modified.

Based upon updates to the bit vector over time, the CBF dynamically captures information regarding addresses that have been observed. To query on a given address, the N-bit

address is again hashed using the k hash functions and the bit values are read from the locations indexed by the m-bit hash values. If any of the bit values are 0, it is sufficient to conclude that the address has definitely not been observed and this is called a true miss [12]. The Counting Bloom filter cannot guarantee that in the case of each of the bit values being 1 however, that the address has definitely been observed. This would be called a false hit.

The major drawback of the original Bloom filter is that the filter can be polluted rapidly and filled up with 1's as it does not have deletion capability. Thus, the Counting Bloom filter (CBF) was proposed to allow deletion of entries from the filter. The CBF reduces the number of false hits by introducing counters instead of a bit vector. In the CBF, when a new address is observed for addition to the Bloom filter, each m-bit hash index addresses to a specific counter in an L-bit counter array,[1] whose counter is then incremented by one. Similarly, when an address is observed for deletion from the Bloom filter the counter is decremented by one. If more than one hash index addresses to the same location for a given address, the counter is incremented or decremented only once. If the counter is zero, it is a true miss. Otherwise, the outcome is not precise.

From the description of the CBF, we can see that it is a simple, low overhead data structure that acts as a signature of addresses present in a cache via the bit vector contents. IdlePower uses observations regarding the rate of change of this signature as a means of characterizing executing VMs. In particular, we use the Counting Bloom filter to obtain an estimate of the change in cache footprint between two points in time, t_1 and t_2. The hardware stores the bit vector state from the last time it was accessed by the software (e.g. the hypervisor) at time t_1. At a subsequent t_2, when it is accessed again, hardware computes the bitwise XOR of the current and stored bit vectors, and counts the number of positive entries. This metric gives us an idea of temporal locality of the benchmark. A high value of the metric suggests significant changes in cache contents between successive points in time and therefore low temporal locality. A low value suggests minimal changes in cache content and thus high temporal locality. There is a corner case where there may be minimal accesses to the L2 cache, or the workload may have a minimal L2 cache footprint that may cause a low value of the metric since the number of cache lines changes will also be low. These factors can be easily accounted for by also requiring that the number of L2 accesses observed be greater than some threshold. The number of L2 accesses can be obtained by using standard processor performance counters.

5.2 System Architecture

We have presented a definition of our Counting Bloom filter, as well as a metric that can be calculated in hardware to estimate changes in cache footprint. Our goal is to provide this knowledge to the hypervisor, where it can be exported to privileged management domains that are responsible for effectively determining whether utilizing deeper C-states may impact VM performance. To accomplish this, we provide an interface where the hypervisor can access the CBF and obtain the current metric value, which is calculated based upon the current bit vector values and the state capture from a previous access as described earlier. The resulting architecture of the IdlePower system is shown in Figure 6.

[1] L must be wide enough to prevent saturations.

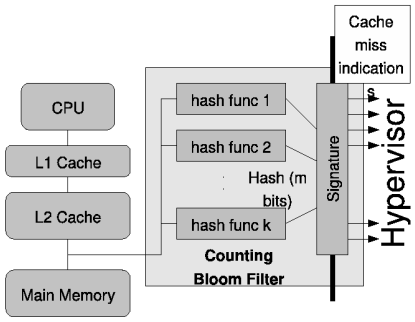


Figure 6: Proposed System Architecture.

Since we're mainly concerned with L2 footprint information, the Bloom filter is attached to that layer of the memory hierarchy. The hypervisor accesses the Bloom filter during periodic intervals while the VM is executing on the processing core. Over time this allows policies that drive IdlePower to learn about application behavior. As we will show in Section 6, the stream of monitored data can successfully differentiate workloads based upon their cache usage behavior. By allowing the hypervisor to obtain these trends at runtime, the Bloom filter component of IdlePower enables application-aware management of C-states.

6. EXPERIMENTAL EVALUATION

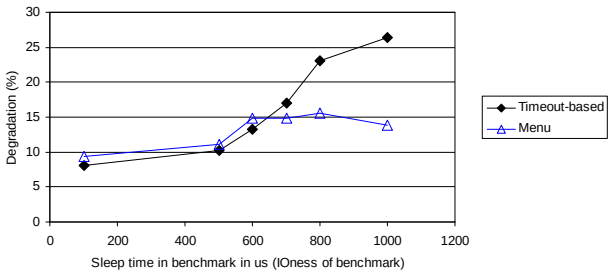


Figure 7: Degradation of performance for *cache-fit* measured against average length of idle periods.

6.1 Cache Implications

Our experiments to measure the effect of deep C-states on performance degradation were run on an Intel Core Duo processor at 2.16GHz. The size of the L2 cache used in the experiments was set to 2MB. To test the effects of aggressive C-state switching algorithms we synthesized benchmarks *cache-fit* and *cache-insensitive* which represented extreme cases for the effects of cache flushing. The *cache-insensitive* benchmark was characterized by a total lack of

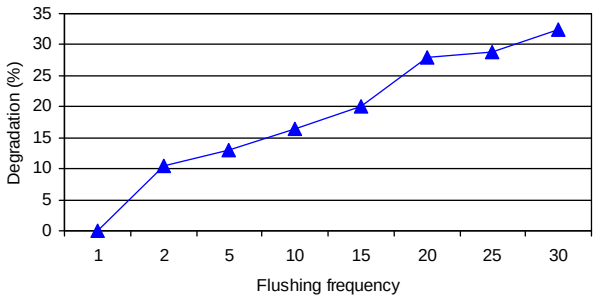


Figure 8: Degradation of performance for *cache-fit* measured against frequency of occurrence of idle periods.

temporal or spatial locality of memory accesses and therefore showed negligible degradation due to the cache flushing caused by C-state switching. However the *cache-fit* benchmark which contained accesses, which showed a high degree of both spatial and temporal locality, showed significant degradation.

The structure of the benchmarks consisted of iterative cycles of intense memory activity followed by a sleep period. The relative lengths of the two sections are configurable. For our experiments, the minimum length of the sleep period was fixed such that this value was greater that the expected latency for transitioning to state C_3 on our hardware.

These benchmarks were tested with the default C-state algorithm in Linux version 2.6.22 [2] which is timeout-based and progressively steps to deeper C-states and the menu governor from the cpuidle framework [25]. The menu governor uses information available in the Linux scheduler about the next timer deadline to match with a list of latencies for each C-state and chooses the deepest possible C-state to transition to.

Figure 7 shows the performance of the two conventional algorithms for the *cache-fit* benchmark when the average length of the periodic sleep time was varied. Since the length of the sleep periods ensured that C_3 state was entered during each idle period, the L2 cache was also flushed. The resulting degradation was measured by running the benchmark first with C_2 and C_3 states disabled in BIOS and then with the deep states enabled. The frequency of entering C_3 state (and therefore flushing the L2 cache) could be set by varying the length of the memory-bound section. The degradation of performance of the algorithms when the frequency of the idle periods was varied is shown in Figure 8.

Both figures indicate that for certain workloads, the degradation due to actions of workload-agnostic algorithms is considerable. Therefore it is imperative that we identify workloads that are susceptible to performance degradation due to aggressive idle power management. In the next two sections we first describe our simulation setup and the show how Counting Bloom filters were used to identify these kinds of workloads by observing estimates of L2 cache misses during the execution of the workloads.

6.2 Simulation Environment

We use Simics [4], a full system emulator that can run unmodified production software like commodity operating systems, to emulate an x86-based virtual machine. The Simics *g-cache* module has been modified to implement the Bloom Filter Signatures infrastructure.

The hardware-software interface in this infrastructure has been implemented using Simics *magic* instructions. The Linux 2.6.22 kernel has been modified to incorporate calls to the special *magic* instruction periodically. When a *magic* instruction is executed by the Simics simulator, it passes the control to the Simics handler. This *magic* handler has been modified to pass bloom filter signatures. A kernel module reads these signatures and makes these available to userspace. The signatures can then be used by userspace C-state algorithms to scale their aggressiveness.

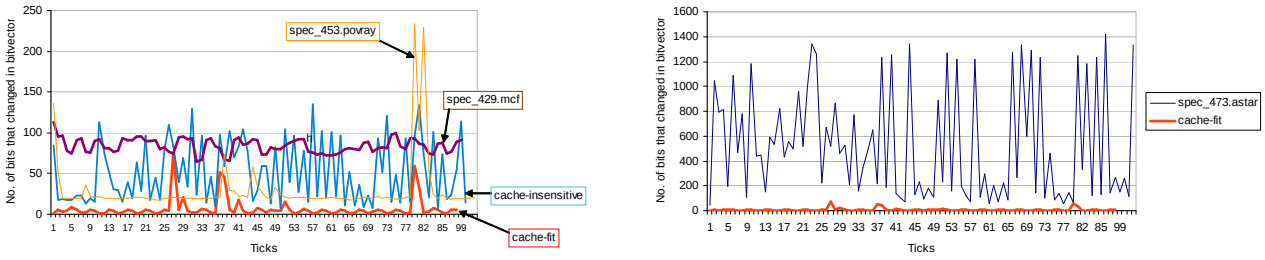
6.3 Counting Bloom filter

Along with *cache-fit* and *cache-insensitive*, we also choose three benchmarks from the SPEC2006 suite. The benchmarks chosen were a ray tracer - *453.povray*, a program to solve a scheduling problem - *429.mcf* and an A^* search algorithm - *473.astar*. The *453.povray* benchmark is known to be CPU-intensive while the *473.astar* is extremely memory-intensive. The *429.mcf* benchmark represents an intermediate case.

Figure 9(a) shows the differences in the values representing estimates for cache misses for our benchmarks. It can be seen that there exists a suitable gradient for different kinds of workloads for the power management algorithms to grade the suitability of a workload for aggressive C-state management. We use a separate graph in Figure 9(b) to show the large difference between the values for *473.astar* and *cache-fit* which indicate that the former is an excellent candidate for aggressive idle power management while the latter would suffer high performance degradation as shown already. These results show that IdlePower is justified in using this parameter as a criterion for suitability of aggressive C-state switching for a given workload.

7. CONCLUSIONS

This paper introduces the IdlePower methods for integrating application-awareness to idle state management algorithms. Idle power management has been used in the sense of consolidating all workloads on fewer machines first in non-virtualized and then increasingly in virtualized environments. However, we argue that advances in idle power management techniques in hardware such as low power deep C-states and low transition latencies have made it possible to use aggressive C-state algorithms during short idle periods that occur during execution. We introduce a novel batching algorithm that batches timing interrupts sent to virtual machines running workloads in order to increase the average length of idle periods and hence the deep C-state residency. We show an average increase of 10% in C_3 state residency. However, we also recognize that aggressive idle state management has a number of implications. The most important one is the degradation of application performance due to the loss of cache state, especially the L2 cache, which results from entering deep power-conserving C-states. This means that aggressive algorithms cannot be used in an application-agnostic manner. Conventional C-state switching algorithms were observed to cause as much as 26% degradation in performance for certain workloads. This information regarding the suitability of the application for such power management measures is contained in the usage statistics of the cache. We



(a) (b)

Figure 9: Estimation of the number of cache misses for different workloads

tap into this information by using a simple Counting Bloom filter which stores a signature of the cache contents. By observing the rate of change of this signature and making this information available to the power management algorithms, we introduce application-awareness to these algorithms.

As future work we plan to utilize the IdlePower strategies developed in this paper along with our VirtualPower architecture developed in prior work [23] to build a comprehensive management system for multicore chips. This includes investigating how to account for and possibly overcome dependencies between core sleep states. In the context of multipackage server platforms, IdlePower can be extended to intelligently allocation VMs onto packages so as to maximize utilization of deep C-states. Finally, we plan to extend the use of Bloom filters as part of these allocation strategies to effectively consolidate VMs while minimizing performance impacts due to cache contention between workloads.

8. REFERENCES

[1] Kvm. http://kvm.qumranet.com/kvmwiki.

[2] The linux kernel archives. http://kernel.org.

[3] Powertop. http://www.lesswatts.org/projects/powertop/.

[4] Virtutech simics. http://www.simics.net.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[6] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. In *Communications of the ACM*, 1970.

[7] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, 2001.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[9] Q. Diao and J. Song. Prediction of cpu idle-busy activity pattern. In *Proceedings of High Performance Computer Architecture (HPCA)*, February 2008.

[10] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In *Proceedings of the Workshop on Power-Aware Computer Systems (PACS)*, December 2003.

[11] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, December 1999.

[12] M. Ghosh, E. Ozer, S. Biles, and H. Lee. Efficient system-on-chip energy management with a segmented bloom filter. In *Proceedings of the 19th International Conference on Architecture of Computing Systems*, pages 283–297, March 2006.

[13] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification. http://www.acpi.info, September 2004.

[14] J. Horrigan, N. Thangavelu, G. Vargese, and B. Holscher. Cache flushing, us patent 465216, intel corporation, December 2003.

[15] 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 the 39th International Symposium on Microarchitecture (MICRO-39)*, December 2006.

[16] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2007.

[17] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. In *Proceedings of the Fourth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pages 263–277, October 1998.

[18] H. Li, C. Cher, T. Vijaykumar, and K. Roy. Vsv: L2-miss-driven variable supply-voltage scaling for low power. In *Proceedings of the IEEE International Symposium on Microarchitecture (MICRO-36)*, 2003.

[19] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proceedings of the 16th Annual ACM International Conference on Supercomputing*, 2002.

[20] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling cool: Temperature-aware workload placement in data centers. In *Proceedings of the USENIX Annual Technical Conference*, June 2005.

[21] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34(4), April 2001.

[22] R. Nathuji, K. O'Hara, K. Schwan, and T. Balch. Compatpm: Enabling energy efficient multimedia workloads for distributed mobile platforms. In *Proceedings of the ACM Multimedia Computing and Networking Conference (MMCN)*, 2007.

[23] R. Nathuji and K. Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, October 2007.

[24] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. In *Intel Technology Journal (http://www.intel.com/technology/itj/2006/v10i3/)*, August 2006.

[25] V. Pallipadi, S. Li, and A. Belay. cpuidle: Do nothing, efficiently... In *Proceedings of the Linux Symposium*, June 2006.

[26] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-based dynamic frequency scaling for memory-bound real-time applications. In *Proceedings of the 11th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, March 2005.

[27] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006.

[28] B. Seshasayee, R. Nathuji, and K. Schwan. Energy-aware mobile service overlays: Cooperative dynamic power management in distributed mobile systems. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2007.

[29] E. Shih, P. Bahl, and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pages 160–171, September 2002.

[30] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, 2001.

[31] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 13–23, November 1994.

[32] A. Weissel, B. Beutel, and F. Bellosa. Cooperative i/o-a novel io semantics for energy-aware applications. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[33] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[34] H. Zeng, C. Ellis, A. Lebeck, and A. Vahdat. Currentcy: A unifying abstraction for expressing energy management policies. In *Proceedings of the USENIX Annual Technical Conference*, June 2003.