Understanding Idle Behavior and Power Gating Mechanisms in the Context of Modern Benchmarks on CPU-GPU Integrated Systems

Manish Arora[†‡] Srilatha Manne[†] Indrani Paul[†*] Nuwan Jayasena[†] Dean M. Tullsen[‡]

[†]Advanced Micro Devices, Inc. [‡]University of California, San Diego [*]Georgia Institute of Technology

***Abstract*—Overall energy consumption in modern computing systems is significantly impacted by idle power. Power gating, also known as C6, is an effective mechanism to reduce idle power. However, C6 entry incurs non-trivial overheads and can cause negative savings if the idle duration is short. As CPUs become tightly integrated with GPUs and other accelerators, the incidence of short duration idle events are becoming increasingly common. Even when idle durations are long, it may still not be beneficial to power gate because of the overheads of cache flushing, especially with FinFET transistors.**

This paper presents a comprehensive analysis of idleness behavior of modern CPU workloads, consisting of both consumer and CPU-GPU benchmarks. It proposes techniques to accurately predict idle durations and develops power gating mechanisms that account for dynamic variations in the break-even point caused by varying cache dirtiness. Accounting for variations in the break-even point is even more important for FinFET transistors. In systems with FinFET transistors, the proposed mechanisms provide average energy reduction exceeding 8% and up to 36% over three currently employed schemes.

I. Introduction

Dynamic and leakage power reduction is a first-order design goal of modern processor architects. Dynamic power is lowered by reducing the work required for accomplishing a task [48], or by utilizing dynamic voltage and frequency scaling (DVFS) to run the task at lower frequency and voltage [5]. Leakage power, however, is not related to processor activity but is more closely tied to the design choices required to achieve high frequency operation with smaller devices and lower maximum supply voltages [13]. Leakage power scales with supply voltage, but reducing the supply voltage only gets you so far; the only way to eliminate it completely is by removing or gating the power supply, referred to as *power gating*. It is also referred to as entering *core-C6* or *C6*.

Past work has addressed the problem of reducing leakage power during active times by power gating cores [29] [32] [39], caches [60], and components within the core [36] [41]. However, as [29] and [56] show, leakage power and global clock power is also a significant issue when the processor is idle or halted. In the AMD^{TM} 15h family of processors, for example, the core power drops significantly when the machine is halted because the pipeline is idle and the local clocks are gated. However, global clocks and leakage power from caches and logic still dissipate 35% of maximum power. The core power diminishes to near zero only when the module is power gated with the core and L2 cache entering C6 [29].

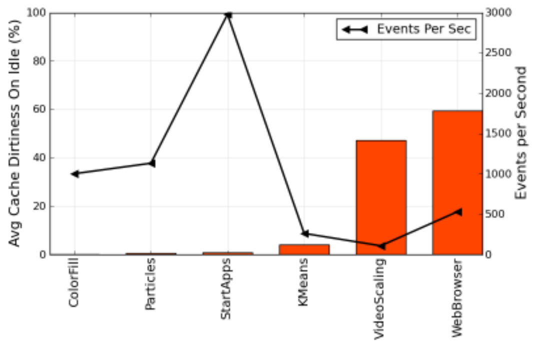
Srilatha Manne is currently employed at Cavium Inc.

Fully exploiting core idle time requires more than just power gating the core when it is not running any jobs. Rather, a large number of important consumer computing applications have significant idle time while the application is "running." Such idle time is characterized by many short idle events. Idleness while running applications happens for several reasons, such as the interactive or I/O-intensive nature of the applications or insufficient parallelism to keep all cores busy [31] [14]. The inclusion of general purpose programmable GPUs [32] [39] and hardware accelerators [18] [35] [20] [1] along with new architectural directions (e.g., addressing the dark silicon challenge [27]) and software directions [40] are expected to further increase the appearance of rapid idleness even in traditional compute intensive application domains.

To achieve energy proportional computing [10], the overall power of the system should be proportional to the work accomplished. This is only possible if the system dissipates little power when idle. However, there are three reasons why it may be difficult to reduce or eliminate idle power. First, transitioning between active and C6 requires time and energy to move state into and out of cores/caches and ramp voltage supply. Second, much of the idle time in modern applications is composed of short duration idle events, where it is not beneficial to enter C6. Third, the amount of state that needs to be moved varies – depending on workload and how long the core was active before becoming idle.

Existing power-gating techniques assume the existence of a *break-even point* – if the core will be idle longer than that break-even time, it should be gated, otherwise it should not. This work demonstrates that no such static point exists, and in fact the break-even point is dynamic, different for every idle event. This is because the cost of gating is heavily influenced by the time to flush the cache, which varies according to the number of dirty lines.

Fig. 1(a) shows the average number of idle events per second (line) and average percentage of the cache that is dirty when the core becomes idle (bar), across a subset of consumer and CPU-GPU applications discussed in Section II. For optimal power savings, the idle time of the application should be characterized by very few idle events. Unfortunately, most applications have a high frequency of idle events, ranging from an average of about 110 events per sec (or 9$m$sec between idle event starts) (*VideoScaling*) to 3,000 events per sec (or about 300 $u$sec between idle events) (*StartApps*). Also, applications differ significantly in the amount of cache that is dirty on idle, ranging from nearly 0% (*ColorFill*) to as much as 60% (*WebBrowser*). This makes it difficult to predict when

(a)


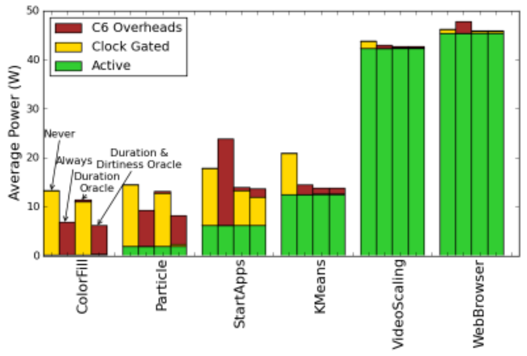
(b)


Fig. 1: **(a) Average number of idle events per sec (line) and average percentage of cache dirty on idle (bar). (b) Impact of idle management policies on average power (modeled) for 16nm FinFET.**

it is advantageous to enter the C6 state.

Fig. 1(b) shows the power associated with each workload using different idle power management schemes. All bars show the average power for the application for FinFET transistors at 16nm, broken down into power while active (*Active*) and power while idle. The idle power is further segmented into power when the core is idle but has not entered C6 (*Clock Gated*) and the power associated with the core entering and exiting C6 (*C6 Overheads*). Results are shown for never entering C6 (*Never*), always entering C6 (*Always*), a perfect duration predictor for entering C6 (*Duration Oracle*) that assumes an exact knowledge of idle durations but cache dirtiness as the average of actual dirtiness across all benchmarks, and a perfect duration and dirtiness predictor (*Duration & Dirtiness Oracle*) that has precise knowledge of both durations and dirtiness.

The results demonstrate that there can be significant benefit from entering C6. However, if not done carefully, C6 overheads may completely erase the gains from power gating (*StartApps, WebBrowser*) and we lose both power and performance [45] [12] [46] [55]. Also, even perfect knowledge of the durations of idle events is not enough (*ColorFill, Particles*), unless combined with the knowledge of cache dirtiness.

The era of increasing idleness in CPU cores presents a set of new challenges to the research and industrial design community. Traditional architectural research tends to focus on applications (SPEC® , PARSEC, etc.) with unrealistically few idle periods. These are crafted, in large part, to evaluate the active behavior of the CPU; in that context, CPU idleness is uninteresting. The modeling infrastructure often artificially removes I/O latencies, cannot typically account for accelerators such as GPUs, and simulates time frames too short to capture interesting idle behavior. This paper addresses these challenges, allowing us to accurately model idle behavior and its impact on power and performance. Specifically, this paper:

(1) Examines the idleness and cache dirtiness characteristics of current and emerging applications, especially in the realm of CPU/GPU heterogeneous applications;

(2) Clearly delineates the overheads associated with power gating and develops a power and performance model to characterize the costs and benefits of power gating;

(3) Evaluates power gating methodologies in the context of current technology as well as with future FinFET transistors, where the leakage and dynamic power trade-offs deviate significantly vs. existing process technologies; and

(4) Using the model, revisits state-of-the-art idle power management schemes and develops new algorithms that not only reduce energy, but also significantly reduce the incidence of large outliers.

The rest of the paper is organized as follows. Section II presents an idleness and cache dirtiness analysis of current consumer and emerging heterogeneous applications. Section III describes the process used for power gating cores, along with measurements of the time and power associated with each step in the process. In addition, it presents and verifies the modeling infrastructure developed for analyzing idle power management algorithms. Section IV describes existing idle power management algorithms, their key shortcomings, and describes our proposed enhancements. Section V presents results. Section VI describes related work, and Section VII concludes the paper.

II. Understanding Application Idle Behavior

Much of the research on power reduction and management has focused on power during active computation. However, recent work has shown that idle behavior and idle power management is as or more important in systems ranging from single processor clients [56] [29] [61] to large scale datacenters [10]. Managing idle power for cores is straightforward if the idleness duration and cache dirtiness behavior is predictable or if the idle periods are always long enough to compensate for the power and performance cost of power gating. However, neither of these conditions hold in many current and emerging applications. In this section, we present a set of important consumer and CPU-GPU benchmarks, and examine their behavior to determine the viability of different C6 strategies.

A. Applications

If we are going to model and evaluate techniques for idle-period power reduction, the first thing we need to do is identify workloads that have realistic idle behavior. For this study, we composed a suite of 20 benchmarks (see Table I) derived from (1) a classic consumer application benchmarking

TABLE I: Benchmark classification, active power (modeled), events per module, and power for different idle management policies (modeled) using 28nm planar technology. Oracle assumes perfect knowledge of both idle durations and dirtiness.

Benchmark	Description	Type	Active (W)	Events (per Sec)	Utilization (%)	Never (W)	Always (W)	Oracle (W)
ColorFill	Graphics color filling	Consumer graphics [6]	0.3	1001	0.4	27.1	10.7	10.0
TextureFill	Graphics texture filling	Consumer graphics [6]	0.5	1039	0.9	27.2	11.1	10.2
Heartwall	Medical imaging	Emerging CPU-GPU [16]	1.6	168	2.7	26.4	6.8	6.2
Particles	DirectX® GPU test	Consumer graphics [6]	3.2	1131	5.4	28.9	14.1	13.1
PicStore	Importing pictures from disk	Consumer storage [6]	10.1	819	16.9	32.4	18.5	17.9
StartApps	Starting of PC applications	Consumer storage [6]	10.1	2977	17.0	34.3	32.6	24.3
LavaMD	Molecular dynamics	Emerging CPU-GPU [16]	13.4	1239	22.5	33.6	24.8	18.4
StartGame	Storage test on a PC game	Consumer storage [6]	16.1	1611	27.1	36.5	29.1	25.7
KMeans	Clustering and data mining	Emerging CPU-GPU [16]	20.1	261	34.2	37.0	24.3	23.6
Pathfinder	Dynamic programming	Emerging CPU-GPU [16]	21.6	2266	36.6	38.5	39.1	27.0
Backprop	Pattern recognition	Emerging CPU-GPU [16]	23.0	5561	38.8	40.1	61.3	30.6
LUD	Dense linear algebra	Emerging CPU-GPU [16]	23.2	1041	39.4	39.0	32.4	27.3
DirectX9	DirectX 9 graphics test	Consumer graphics [6]	23.4	1988	39.8	40.8	39.2	32.8
Leukocyte	Medical imaging	Emerging CPU-GPU [16]	24.3	3821	41.2	40.6	51.7	31.2
NN	Data mining nearest neighbor	Emerging CPU-GPU [16]	27.6	3048	46.4	42.2	49.2	33.4
Gaussian	Dense linear algebra	Emerging CPU-GPU [16]	29.8	1146	50.7	42.7	39.2	33.2
TextEditor	Interactive text editing	Consumer interactive [6]	42.6	2281	60.1	54.8	59.1	50.8
WebDecrypt	Web browsing and data decryption	Consumer interactive [6]	61.8	416	87.0	65.3	64.7	62.9
VideoScaling	Video downscaling	Consumer media [6]	62.7	107	88.2	65.8	63.9	63.6
WebBrowser	Multi-tabbed browsing on Chrome	Consumer interactive [6]	67.4	530	94.9	68.9	70.2	68.2

suite (PCMark® [6]) popularly used in industry; and (2) heterogeneous CPU-GPU applications (Rodinia [16]). Many of these applications are common to the consumer domain such as graphics applications (*ColorFill, DirectX9* etc.), storage applications (*PicStore, StartApps* etc.), and interactive (*TextEditor, WebBrowser*). However, others are targeted at heterogeneous processors such as AMD APUs [56].

Heterogeneous processors with frameworks such as the heterogeneous system architecture (HSA) [1] offer opportunities for tightly-coupled interaction and easy compute offloading between the CPU, GPU, and other accelerators. Hence, we expect to see the CPU sharing computation tasks with other processors and accelerators in the future [35] [20], leading to finer-grained interaction between compute entities on the chip. This will produce more idle time and a higher frequency of idle events on the CPU.

The *Events* column in Table I shows the number of idle events per second per CPU module. There are a total of two CPU modules for the chip under study as described in Section III. An idle event is defined as the transition from active execution to idle. The *Utilization* column shows the proportion of active to total time for each application. The higher the utilization, the more active the application and higher the power attributed to active execution (*Active* column). The table is sorted by increasing utilization, and it ranges from less than 1% to greater than 94%.

The table also shows the total power associated with each application when C6 entry is never performed (*Never*), always performed (*Always*) and only performed when beneficial using precise knowledge of durations and cache dirtiness (*Oracle*). These values are derived using the modeling infrastructure described in Section III. In general, the lower the utilization, the more opportunity for idle power savings, and the more significant the relative power difference between never entering C6 and always or perfectly entering C6. Generalities, however, do not always hold. For instance, both *ColorFill* and *TextureFill* have very low utilization. However, even with a perfect predictor, the resulting power is still over 10W, with active state power contributing less than 1W in both cases. For *Heartwall*, however, the power can be brought down to less than 7W even without the Oracle predictor. Similarly, the higher the number of events, the more power but *Leukocyte* has almost twice the number of events and higher utilization than *DirectX9*, but still consumes less power with the Oracle predictor. The relationship between idle time, idle event frequency, and power savings is complicated, and the next section examines some of the characteristics that dictate when and by how much applications benefit from C6.

B. Idle Event Distribution

Table I shows the frequency of events and an estimate of idle time (100%-*Utilization*), but it does not provide any insights into how the events are distributed. For instance, *Backprop*, contains many idle events, and we can calculate the average duration, but the potential for gating will depend more on the distribution of idle event times than the average value. This section examines some of those distributions more closely.

Fig. 2 shows the distribution of idle events of our applications collected using the AMD A8-5600K APU and the Xperf tool set as described in Section III. The y-axis shows the cumulative distribution function (CDF) of idle events, as a function of the idle event duration, and the x-axis represents the event duration (in $u$sec). The figure shows several interesting trends. For example, a large number of applications have a majority of their events as short idle events of durations less than $100u$sec. Graphics applications such as *ColorFill* regularly interrupt the CPU at $1m$sec granularity because of the use of multimedia timers in Windows® to maintain system responsiveness. Lastly, we observe the OS timer tick inducing activity every $15.6m$sec for some of the applications.

Power gating is most effective with long idle events, or at least predictable-duration events. Examples of predictable

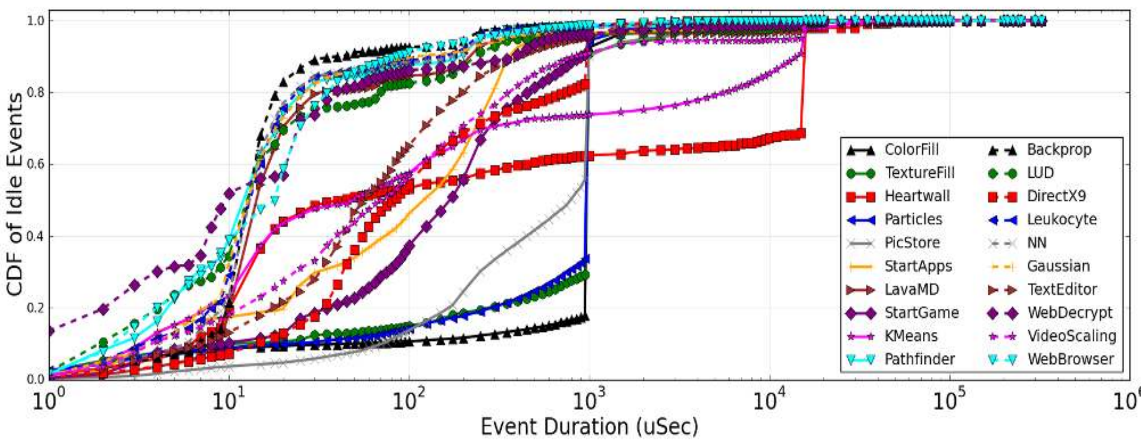


Fig. 2: **Cumulative distribution function (CDF) of idle events.**

events are where all idle events are of a fixed duration, or the idle events are bimodal, either very short or very long, or the short events come in bursts that can be predicted.

Some of the applications do fall into these predictable categories. A significant portion of the idle events in *Backprop* are events of duration less that $100u$sec. Hence filtering out the short duration events (e.g., by delaying entry into C6 until some time has passed), which represent about 90% of the total number of events, will result in good idle power savings. However, filtering out short duration events might be detrimental for other applications such as *ColorFill*, and *TextureFill*. These benchmarks do not have many short duration idle events, and most of the events and idle time are composed of $1m$sec events. By having a static algorithm that always delays entry into C6 in order to respond to the needs of *Backprop*, we will not be able to capture the full power savings possible for *ColorFill* and *TextureFill*.

Other applications are less predictable and thus more problematic. *StartApps*, which has low utilization and should be a good candidate for C6 entry, has many short duration idle events ($>$95% are $1m$sec or less). Therefore, there are very few long duration idle events, and it is difficult to identify those long duration events since the events in general are widely distributed with respect to their idle durations.

Another important factor that can significantly impact C6 decisions is the amount of cache dirtiness that is introduced by the active execution prior to the CPU becoming idle. Next we will examine active periods and cache dirtiness properties of different benchmarks.

C. Cache Dirtiness Characterization

Whenever the CPU becomes idle, dirty lines in the deactivated caches need to be flushed to either main memory or caches that are not power gated. Cache dirtiness depends on the benchmark, as well as the duration that the benchmark was active before the CPU became idle. This section examines active time periods and the induced dirtiness.

Fig. 3(a) shows the average durations of active events that are followed by long idle events. For this particular analysis, we refer to idle events great than $200u$sec as long, as they may potentially be C6 entry opportunities. Fig. 3(b) shows the growth in cache dirtiness as a function of the active time. Dirtiness was evaluated by collecting memory traces and doing cache simulation as explained in Section III.

Fig. 3(b) shows that even for the same active execution period, benchmarks differ in the cache dirtiness that they induce. *VideoScaling* and *WebBrowser* cause more than 50% of the cache to be dirty for their average active execution periods ($\sim 10m$sec). However, *StartGame* and *PicStore* exhibit a very low growth rate and final level of dirtiness. Although, *WebDecrypt* exhibits large active times, the maximum dirtiness it causes in the cache is $\sim$20%. Benchmarks also vary in the time when dirtiness stabilizes; however, all benchmarks exhibit cache dirtiness growth up to $10m$sec of active time. On average across all benchmarks, an active time lasts about $3m$sec and causes about $\sim$15% of the cache to be dirty.

To craft an effective power gating application, we must understand both application behavior and the power gating mechanism itself. The next section describes the latter.

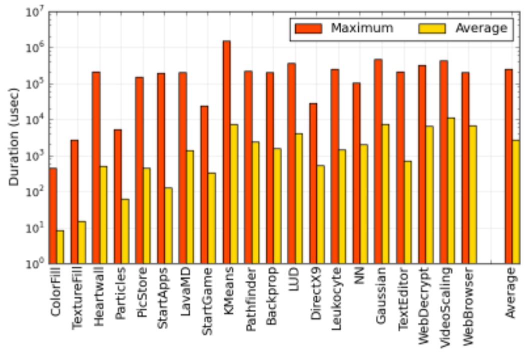
III. Modeling C6 Timing and Overheads

This section first describes the C6 process used in current hardware to establish the goals and challenges of different power gating control policies. Next, it explains our modeling infrastructure and presents results to validate our methodology. We use the AMD 2nd Generation A-Series APU (formerly codenamed "Trinity") A8-5600K with a 100W TDP running Windows 7 in performance mode at 3.6GHz (fixed P0 state when active) in our study. The A8-5600K contains two dual-core modules and a GPU core. Each module has two cores (for a total of four), and the cores in a single module share a front end, the floating-point unit, and a 2MB L2 cache. Each module can also be power gated, independent of the other module. A module is a candidate for power gating if both cores within the module are idle.

A. C6 Entry and Exit Process

Fig. 4 shows the details of the C6 entry and exit process. The top half of the figure shows the general steps that occur when a module is halted but does not enter the C6 sleep state. As soon as the halt occurs, the local clocks for the module are gated, resulting in some power savings. However, global clocks are still operational. Next, P-state changes are initialized to

(a)



(b)

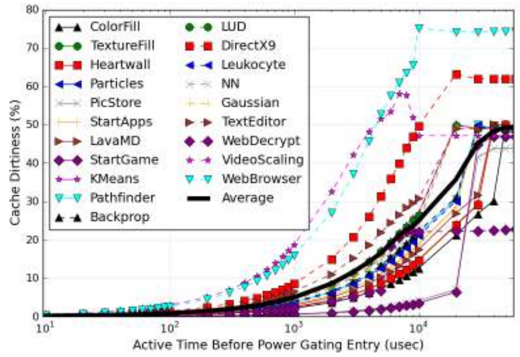


Fig. 3: **(a) Duration of active events preceding long idle events. (b) Growth of cache dirtiness with time.**

drop to lower active states in order to reduce voltage and hence reduce leakage power consumption. The P-State transition time is dependent on the absolute voltage change, the rate of voltage drop and minimum P-state residency requirements imposed by system management. The cores then continue to be in a clock gated state. Once work arrives, execution restarts with near negligible overheads.

In the bottom half of Fig. 4, we show the steps necessary for a core to enter C6 (i.e., be power-gated). Once idle, the core can choose to continue to remain in a clock gated state for some time, a state referred to as *PreFlush*. The *PreFlush* state is used by current generation AMD processors from the 15h family to filter out very short duration idle events [7] and is conceptually equivalent to the *IdlePG* heuristic proposed in [46]. Note that the *PreFlush* timer can be set to values less than the time it takes to waterfall P-state changes to the lowest active state.

As soon as the *PreFlush* timer expires, all modified data in the caches are flushed to main memory in the *CacheFlush* step. In other systems, the dirty data may be flushed to a non-gated cache (such as an L3 cache). This operation requires a variable amount of time depending on the number of dirty lines in the cache. The *PreFlush* and *CacheFlush* steps are both interruptible (i.e., they can be stopped once started).

After the caches are flushed, the core architectural state including machine state registers (MSRs) are copied into memory during *StateSave* and power is ramped down. The duration of this step depends on the voltage ramp down time and reliability limits imposed to reduce voltage swings. After saving the state, the core enters the *PowerGate* state. *PowerGate* is the lowest possible power state in which header or footer power transistors are switched off, effectively removing voltage from the core and its caches. The power consumed in C6 state is very low but non-zero [60]. The duration of time at which the module remains in this state is variable depending on the length of the idle period.

Once work is again available, the process to bring the core out of C6 is initiated. Depending on the source, the wake up interrupt propagates through the northbridge or southbridge controller to the system management unit and initiates a power ramp up. After power is ramped up, reset, fuse propagation, and micro-code patches are applied, followed by the *StateRestore* step to complete the C6 exit. *StateRestore* copies the saved system state from the back-up location in the main memory to the core and resumes normal operation.

Note that the time taken to propagate the interrupt, ramp up power, and perform *StateRestore*, manifests as delay and impacts the start of execution of the next available work (Task 1 execution is delayed in Fig. 4).

StateRestore restores only the architectural state of the core. Cold cache, TLB, and branch effects are an additional cost of a C6 event and can increase the duration of the post-C6 active event Task 1 as shown in Fig. 4. The combined effects of late start and cold miss penalties reduce the responsiveness of applications, and may manifest as actual delay in overall program run-time. Thus, the performance and energy cost of C6 exit is the sum of physical C6 overheads (fairly constant), cache flush time (dependent on the workload), and cold start effects (also dependent on the workload).

B. Simulation Model

Architectural simulation of benchmarks under study is not possible because of the unavailability of timing-accurate public domain simulation tools that can execute PCMark 7 under Windows 7 or realistically model the execution of CPU-GPU benchmarks on real hardware, including graphics drivers and OS activity. Thus, we developed a trace-driven simulator to estimate the impact of C6 management on performance and energy. The simulator replays traces of idle and active states that are captured using runs of these benchmarks on our target system hardware (the A8-5600K APU) and is able to quantify the effects of different C6 policies. The trace driven methodology we employ is similar to that used in [49] and [12]. This methodology allows us to combine the accuracy of real traces of complex application behavior collected on real hardware with the ability to manipulate the power gating policies implemented in the architecture. The duration and power cost of power state transitions is assessed based on measurements on our actual hardware as described below.

Trace Collection. We capture hardware traces using the Windows Xperf tool [4]. Xperf is an event-based performance analysis tool provided in the Windows Performance Toolkit. Xperf captures all idle and active transition events at one microsecond granularity for each core. We obtain Xperf traces at the core level and convert them to activity and idleness

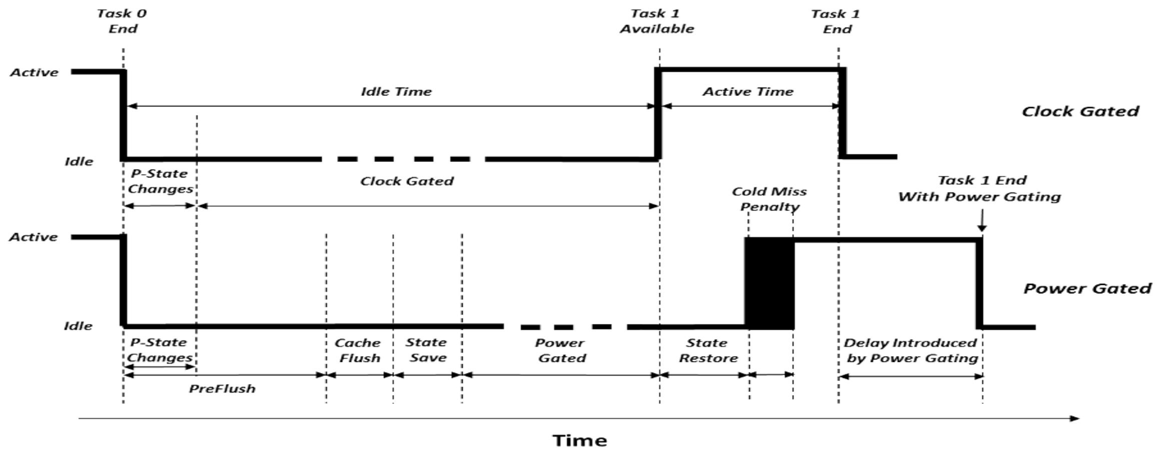


Fig. 4: **CPU behavior and steps in clock gated vs. power gated state.**

traces at the module level. If either of the cores in the module is active, the module is modeled as active.

To obtain estimates of cache dirtiness, we use the AMD SimNow$^{\text{TM}}$ [2] simulation infrastructure to model a Windows 7 based AMD A-Series APU system and collect traces of memory instructions. SimNow is a functionally accurate instruction level platform simulator, allowing us to run full system simulations to completion. However, SimNow is not timing accurate and hence its role is limited to only collecting memory traces in this work. After collecting memory traces, we use an in-house cache simulator to simulate a 2MB sized cache and generate average statistics on the expected dirtiness of the cache for a specified active time.

Micro-benchmarking. In order to determine latencies and power associated with different steps of the C6 process, we performe a micro-benchmark analysis on our system when connected to a power measurement infrastructure. We use the windows multimedia libraries to initiate interrupts every $1m$sec and recorded the change in power at a sample rate exceeding 1MHz for the package. We also vary the *PreFlush* latency and perform measurements at latencies of $0u$sec, $300u$sec and $800u$sec respectively. The results of our analysis are shown in Fig. 5. Note that the power shown in the figure is the combined power of the CPU and on-chip GPU. The figure shows a single timer interrupt window, but with three different values for the *PreFlush* latency.

Fig. 5 enables us to clearly visualize the time and power associated with the different C6 steps. We observe active execution that lasts until 125-$200u$sec followed by *P-State* changes or direct entry into power gated stated (for *PreFlush* $0u$sec). After *P-State* changes, we see the processor waiting in the *PreFlush* state followed by power gating entry on timer expiry around $400u$sec or $850u$sec. This methodology enables us to capture precise time periods of the various C6 steps. For example, *StateRestore* including power ramp-up takes about $80u$sec to complete and *P-state* changes last about $125u$sec. We also use this analysis to derive average power levels for different phases on the C6 process. The analysis gives us good insights into the C6 process. We see the relatively high inefficiencies of staying at clock gated state by comparing the power of active execution, *PreFlush* state, and power gated states. Note that the clock gated power is actually much closer

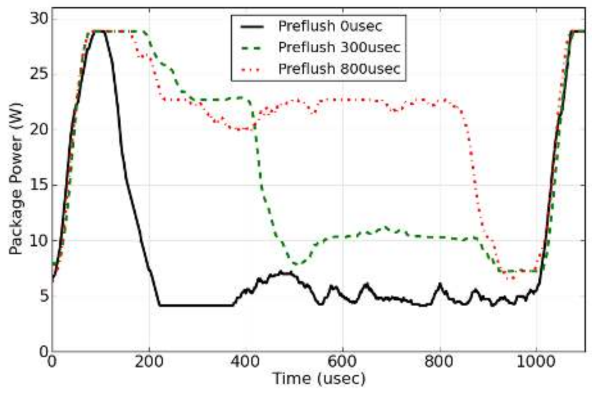
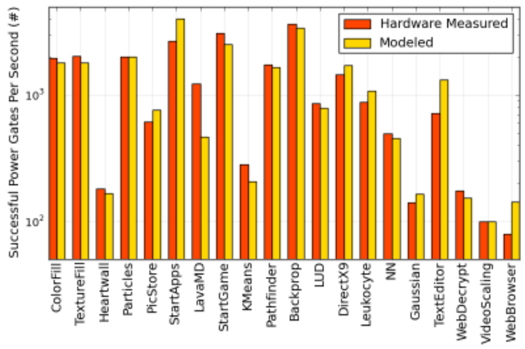


Fig. 5: **Measured package power ($m$sec interrupts).**

to active power than power-gated state.

To derive the remaining parameters, we use a combination of measurements and analytical models. For example, *CacheFlush* step latencies are modeled using an analytical analysis that calculates the time required to flush the cache by estimating the maximum time required to flush a completely dirty cache and estimated dirtiness based on the active time prior to becoming idle supplied by the active time trace. The maximum time to flush the cache is determined by the time required to flush a total of 32K cache lines (2MB cache with 64byte cache lines) to the main memory. Assuming reasonable values of dram and queuing latencies, we derived a time value in the range of 1-$1.5m$sec for the *CacheFlush* step for a fully dirty cache. We assume a value of $1m$sec as a lower bound for this step. This is consistent with values in [7]. We assume no extra power to flush the cache lines to memory beyond the power consumed by the processor in clock gated state. This gives a lower bound on the energy costs of power gating.

We derive active power associated with the different cores of the modules by running single and multi-threaded versions of compute intensive benchmarks. Cold miss time penalties are derived by building a regression model based on trace collection. We first collected active times with power gating off and then compared them with active times with power gating always on (*PreFlush* timer $0u$sec). We derived a single value of the active time overhead of cold miss penalties using this

(a) 

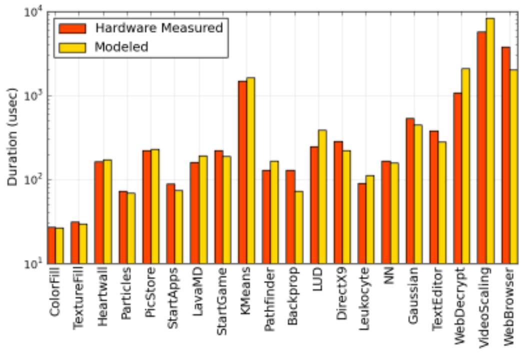
(b) 

Fig. 6: **Hardware measured vs. modeled at PreFlush 0$u$sec (a) Power gates per second. (b) Active times.**

approach. Although we model a single value for cold miss overhead, this value will also vary based on runtime behavior and cause even greater variance in the "break-even point." The analysis of this effect is reserved for future work. Lastly, the power consumed during the cold miss period is assumed to be the same as the active period.

Table II provides a detailed list of values of different parameters used in our model along with information as to how they were derived. Note that the module drops to the voltage level of P-State P5 to save leakage power during the *PreFlush* stage.

Lastly, we use process development kit (PDK) information and ITRS projections to derive dynamic and leakage scaling factors from planar 28nm to FinFET 16nm technologies. We are unable to publicly declare the values, but the important trends to note are the relative scaling values of dynamic and leakage power for FinFET transistors. While leakage power reduces significantly vs. planar transistors, dynamic power does not scale (or increases) because of increased capacitance of FinFET transistors. Hence the overheads of power gating that cause dynamic power consumption proportionally increase for FinFET transistors vs. planar technologies.

TABLE II: Parameters of the trace based simulation model.

Parameter	Value	Methodology
Active time	Variable	Captured using Xperf [4]
Idle time	Variable	Captured using Xperf [4]
P-State change time	125$u$sec	Measured in hardware
PreFlush time	Variable	Design parameter [7]
Max CacheFlush time	1$m$sec	Analytically modeled
CacheFlush time	Variable	Modeled with mem traces [2]
StateSave time	100$u$sec	Measured in hardware
StateRestore time	80$u$sec	Measured in hardware
Cold miss penalty	25$u$sec	Derived using traces
Single core dyn power	10.9W	Measured in hardware
Second core dyn power	6.1W	Measured in hardware
P5 Leakage power	8.5W	Measured in hardware
P0 Leakage power	14.3W	Scaled P5 leakage [29]
Constant dyn power	4.2W	Global clock estimates [29]
PreFlush power	12.7W	P5 leakage + constant dyn
CacheFlush power	12.7W	Same as PreFlush
Avg. StateSave power	8.8W	Measured in hardware
Avg. StateRestore power	8.8W	Measured in hardware
Leakage at C6	2.1W	5% of peak power [29]
FinFET scaling factors	-	Derived using PDK/ITRS

Model Verification. In order to validate our trace based simulation infrastructure, we captured active time traces with power gating turned off and then with PreFlush timer set to 0$u$sec in hardware. We use the former trace (with no power gating) to drive our simulations of the latter configuration, for each of our benchmarks. We then validate the simulation with two different measurements of the real system with power gating. The first is the frequency of gating events (measured with counters on the system under test) and the second is the average event active time gathered from the real trace. Our simulation of the latter, in particular, must account for cold start effects, idle events not power gated, etc. The results are shown in Fig. 6. For both the tests, we see a very good match between hardware based measurements and modeling. This confirms the validity of our timing and cold miss models.

IV. Towards Oracle C6 Entry

As described in Section III, whenever an entity idles there is an option to either stay in clock gated state or initiate entry into C6. This section first describes the state of the art algorithms from commercially available implementations today. Next, we propose algorithms that outperform state of the art implemented algorithms in energy efficiency.

A. Existing C6 Techniques

We examine three specific C6 entry algorithms, *fixed preflush filtering*, *autodemotion*, and *recent value prediction*.

Fixed Preflush Filtering (FPF). The FPF algorithm delays C6 entry for a fixed amount of time, referred to as the *PreFlush* latency, in order to filter out the high frequency, low duration idle events that can be detrimental to performance and energy. This scheme is used in production in the AMD 15h processor systems [7].

Autodemotion. The Autodemotion algorithm also tries to filter out short duration idle events. Autodemotion assumes that short duration idle events are clustered and come in *interrupt storms* [61]. Therefore, the approach taken in Autodemotion is to either enable or disable C6 entry based on the frequency of idle events. If the number is greater than or equal to a threshold of events/sec for the current time interval, C6 is disabled for the next time interval. Otherwise, C6 is enabled and the system immediately enters C6 after P-State changes. The Autodemotion scheme is implemented in the Intel SandyBridge family of processors [61].

Recent Value Prediction (RVP). The RVP algorithm predicts the duration of the current idle event by maintaining a history and taking the moving average of N previous idle events. Based on the predicted duration of averaged values, a decision can be made to enter into C6 or remain clock gated. The RVP algorithm is used as the idle duration prediction algorithm in the Linux idle governor [3] [53].

All of the above algorithms rely heavily on a threshold to choose good operating points – FPF on the preflush latency, Autodemotion on the number of events that constitute a storm, and RVP on the history length. We choose a value of $200u$sec and 5,000 events per second as thresholds for the FPF and Autodemotion schemes respectively. The selected values were found experimentally to be near optimal for our hardware and workloads. For RVP, we found that using the last 64 values to predict the idle duration provides the best prediction and energy efficiency.

Using our simulation infrastructure we performed an evaluation of the relative energy efficiency of all the existing schemes as compared to Oracle gating. Oracle gating is the best possible power gating scheme – it uses precise information on the length of the idle events and cache dirtiness to power gate (immediately) only when it saves power. To summarize our findings, we found that each of the three schemes fell short of Oracle gating by at least 6% for planar transistors and at least 11% for 16nm FinFET transistors. We will present detailed results in Section V. Out of the three schemes, RVP worked the best and Autodemotion worked slightly better than FPF. Next we identify the reasons for the relatively poor energy efficiency of existing schemes.

B. Key Shortcomings

Each of the three schemes suffers from one or more significant problems. Although simplistic and easy to implement in hardware, the FPF scheme divides short and long events with complete accuracy, but does so by taking a power hit on every long idle event (that should have been power-gated immediately). The Autodemotion scheme does a crude prediction (all events are short or all events are long), and suffers from limited accuracy as it performs an aggregate prediction to an unknown number of future events.

Amongst the three schemes, RVP performs the best as it attempts to use recent history to guess the current event duration. The relative success of RVP shows that idle events are correlated enough to allow us to do more accurate predictions. The success of autodemotion over FPF also implies some level of predictability. However, the fact the RVP falls short of the Oracle implies that there is potential for more accurate prediction than provided by this simple mechanism.

Another very significant contributor to the lack of efficiency in all of the three schemes is that all are based on the assumption of a static break-even point. As discussed in Section II, cache dirtiness is a function of the benchmark and active time prior to the idle event. Fig. 7 shows the variation in the break-even point for one module of our modeled system, at different cache dirtiness levels, for both planar 28nm and FinFET 16nm transistors. With no dirtiness in the cache, the break-even point is about $200u$sec for planar and about $425u$sec for FinFET transistors. Since FinFET transistors offer significantly lower leakage, it makes sense to start in clock gated state longer – hence the longer break-even point for FinFET transistors.

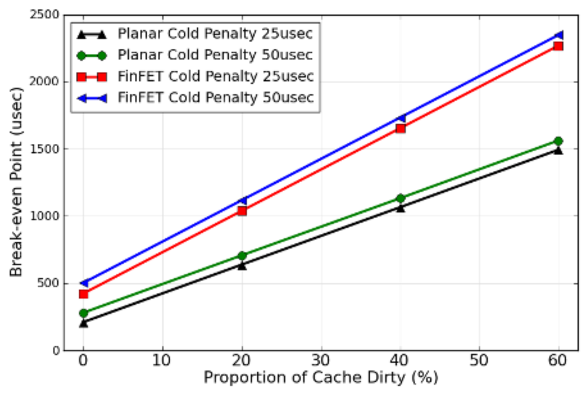


Fig. 7: **Variation in energy break-even point.**

Fig. 7 clearly demonstrates that the break-even point varies significantly with cache dirtiness. In fact, it varies over an extremely wide range of 5X to 7X. Hence, any scheme that only takes into account (predicted) event durations is still highly likely to make bad decisions even with perfect prediction.

Next we propose C6 entry algorithms that address the main shortcomings of existing techniques – lack of prediction accuracy and not incorporating cache dirtiness information.

C. Proposed Algorithm

We propose a scheme that predicts idle durations at a finer granularity (a new prediction for each event), using more careful analysis of past behavior. The proposed scheme is expected to be implemented in the system control unit and has to make sleep-state decisions once every few $u$sec – hence the computational complexity must remain low.

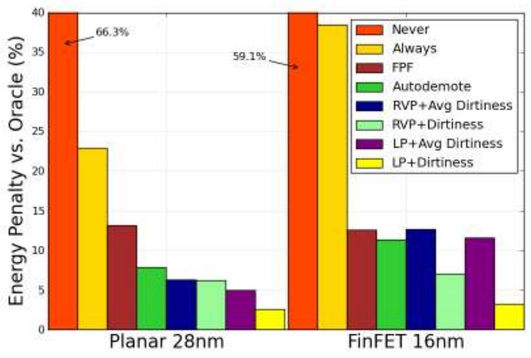
Linear Prediction (LP). The LP algorithm predicts the duration of the current idle event by maintaining a history and making a prediction by building linear prediction models [47] of previous idle event durations. Linear prediction estimates future events of a time series as linear functions of P prior observations as shown in equation 1.

$$\hat{Y}(t) = \sum_{i=1}^{P} a(i) \times Y(t-i) \quad (1)$$

In the above expression, a(i) represent the prediction coefficients and P dictates the number of prediction coefficients used. The error of the prediction, e(t), is:

$$e(t) = Y(t) - \hat{Y}(t) = Y(t) - \sum_{i=1}^{P} a(i) \times Y(t-i) \quad (2)$$

We use the Levinson-Durbin method [42] [47] to solve the above equation for the coefficients which minimize the mean square error of the prediction. Idle event duration history may be retained for N events (where N $\geq$ P) over which the prediction error is minimized for estimating the coefficients. This computation has a complexity of $NP + P^3$ multiply-accumulate (MAC) operations.

(a)


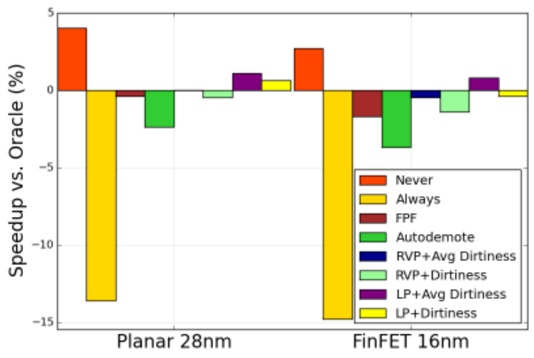
(b)


Fig. 8: **Summary of (a) Energy results. (b) Performance results.**

The selection of N and P values are design parameters. Larger values of these could potentially exploit long-term recurrence patterns at the expense of increased implementation costs. Larger values of N could also capture un-correlated data in the model and reduce performance. We performed a design space evaluation over the space of N and P and found that N=1024,P=1 yielded the best prediction accuracy.

Although the LP algorithm requires $NP + P^3$ MAC operations, or $N+1$ when P = 1, this only holds when we do a new calculation from scratch. If we reuse the prior calculation, we can do an incremental computation when we add one value to the history buffer, using two multiplications (one for the new value, one for the old value popped off the stack), one subtraction, and one addition. Hence for N=1024,P=1, every incremental calculation of coefficients can be calculated in less than 20 operations total. Since the prediction operations typically happen every few $u$sec, a load of 20 MAC operations represents a trivial load on system management controllers that run at speeds of hundreds of megahertz.

Mechanics. First, we use the LP scheme to predict the duration of each new idle event. Next, we obtain an estimate or exact calculation of cache dirtiness. The latter can be easily done with a counter that increments when a clean line transitions to dirty, and decrements when a line is written back. Then, the system management unit evaluates potential power savings by evaluating the dynamic break-even point and comparing it with the predicted duration. If gating is found to be beneficial based on power estimates, C6 is entered immediately, otherwise the module stays in clock gated state till the end of the idle event. The system management unit also tracks the accuracy of the prediction scheme at the end of the idle event once the actual idle duration is available. If there are large prediction errors consistently, the LP schemes default to the Autodemotion scheme. The modeled RVP scheme does the same.

V. Results

We implement *Never* power gating, *Always* power gating, the three baseline industry schemes, our proposed LP predictor using our modeling environment, and generate power and performance numbers for each application both for 28nm planar and 16nm FinFET technologies. The RVP and LP schemes were evaluated both in the context of a static break-even and a dynamic break-even calculation. For the former, they assume a break-even latency based on the C6 delays and costs associated with the fixed average measure (15%) of cache dirtiness. For the latter, a dynamic break-even is calculated based on counters that measure actual dirtiness. This could be a simple linear calculation based on Fig. 7.

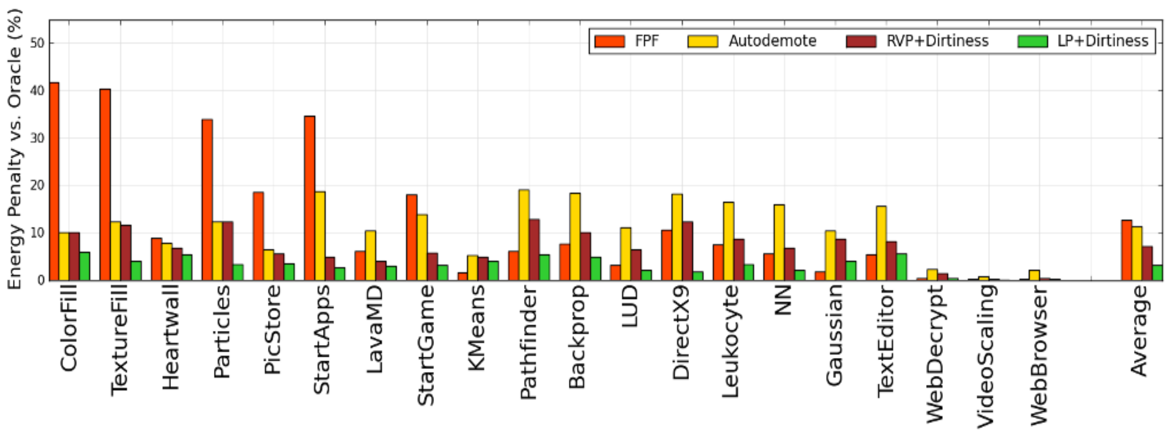
Fig. 8 (a) and (b) summarize the energy and performance findings over the complete set of benchmarks. All results are shown relative to the 28nm planar and 16nm FinFET specific Oracle prediction schemes with perfect knowledge of durations and dirtiness.

As seen in Fig. 8, both *Never* and *Always* have significant penalties vs Oracle gating. *Never* has a slight performance advantage of less than 5% but an energy disadvantage exceeding 50%. *Always* loses about 15% performance. The impact of always gating is worse for FinFET 16nm as the ratio of dynamic energy vs. leakage is high for FinFETs vs. planar, hence, the cumulative power gating entry and exit overheads become more significant for FinFET transistors.

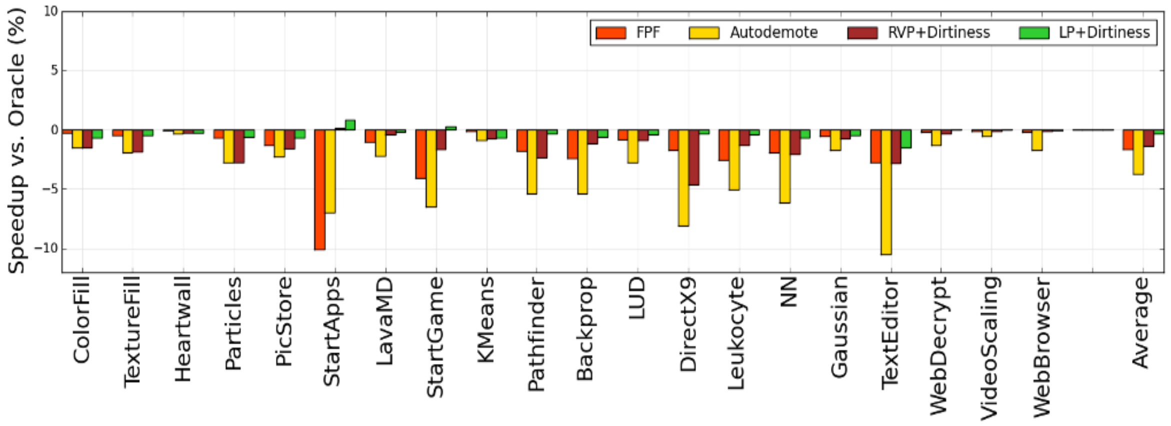
All of the baseline industry schemes (FPF, Autodemotion, and RVP) are more energy efficient than *Never* and *Always* gating. For planar transistors, FPF (13.2%) is much worse than Autodemotion (7.9%) as the leakage energy costs are higher in planar transistors. However, for FinFET transistors, FPF (12.6%) and Autodemotion (11.4%) are more similar.

The RVP+Avg Dirtiness and LP+Avg Dirtiness schemes only offer a very slight or no advantage over Autodemotion. This implies that any additional accuracy in the estimation of idle time is of little use given the inaccuracy of the break-even point. However, when combined with actual measures of dirtiness, RVP+Dirtiness and LP+Dirtiness schemes improve significantly. For FinFET transistors, RVP+Dirtiness improves to within 7.1% of Oracle, an improvement of more than 5% over RVP+Avg Dirtiness (12.7%).

LP+Dirtiness out performs every other scheme as it combines a very good predictor together with measures of dirtiness. To summarize the performance of LP+Dirtiness, it reaches within 3% of Oracle performance for both planar and FinFET transistors. This is nearly an 8% energy advantage over each of FPF, Autodemotion, and RVP+Avg Dirtiness for FinFET transistors. The benefits for planar transistors are more than 4% over all three baseline schemes. Performance of the LP+Dirtiness scheme reaches within +-1% of the Oracle scheme on average. These results clearly demonstrate the



(a)



(b)

Fig. 9: **(a) Energy for FinFET 16nm. (c) Performance for FinFET 16nm.**

benefit of having *both* a good predictor of idle duration and accurately computing the dynamic break-even point. Having one of the two is insufficient.

Fig. 9 (a) and (b) show the energy and performance comparisons between the FPF, Autodemotion, RVP+Dirtiness, and LP+Dirtiness schemes for the complete set of benchmarks. The benchmarks are shown in order of increasing processor utilization, with *ColorFill* being the lowest, and *WebBrowser* being the highest.

We notice some general trends in the energy results. FPF generally has higher energy than Autodemote when the utilization is low, but the trend reverses as utilization increases, indicating that Autodemote is not as effective at filtering short duration idle events with higher utilization benchmarks. As utilization increases, so does the average duration of active events and cache dirtiness; with long active events between idle events, even if the idle events are all short, they will never qualify as a "storm." Hence, Autodemote tends to power gate more often and lose efficiency.

Another interesting trend is that RVP+Dirtiness is worse than FPF on several benchmarks with higher utilization (e.g., *Pathfinder*, *Backprop*, *Gaussian* etc.). As utilization increases, with increasing active times, the duration of idle events decreases. FPF filters all short duration idle events regardless of utilization, and is thus less likely to "miss badly" on shorter idle events. RVP+Dirtiness may suffer from a lack of prediction accuracy in such cases and hence FPF outperforms it. With better prediction, the LP+Dirtiness scheme avoids this and outperforms FPF, Autodemote and RVP+Dirtiness in almost all cases.

Fig. 9 (b) shows performance results for the different benchmarks. In general, Autodemote has the worst performance amongst all schemes. It suffers penalties exceeding 5% on several benchmarks (*StartApps*, *DirectX9*, *TextEditor* etc.). This is consistent with the reasoning described above; Autodemote biases towards gating more often. *StartApps* shows an anomaly and loses about 10% performance on FPF even when the FPF scheme is conservative. The reason for this behavior lies in the event distributions for the *StartApps* benchmark. As shown in Fig. 2, *StartApps* has about 30% of its idle events distributed between 200-400$u$sec. Since the FPF scheme uses a threshold of 200$u$sec it wastes energy and yet does not filter out these medium sized events. Overall, the LP+Dirtiness scheme matches closest to Oracle performance.

In many cases, we care not only about performance, but also the distribution (e.g., the outliers can create long tail distributions, or worse, delay the completion of an entire parallel computation). The goal of any management scheme is to maximize the benefit as well as minimize any outliers.

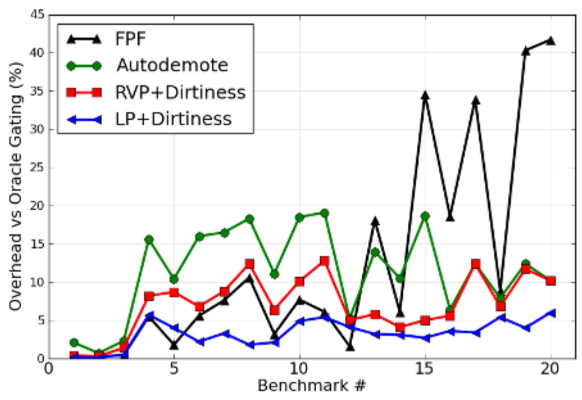


Fig. 10: **Energy efficiency of C6 management schemes for FinFET 16nm. Benchmarks are sorted by decreasing utilization.**

Fig. 10 shows the power penalty relative to Oracle for all four schemes sorted by decreasing utilization. The figure clearly shows that the LP+Dirtiness scheme not only gives the best average gain, but also significantly decreases variation, resulting in a more robust power management algorithm.

VI. Related Work

There have been numerous studies examining workload dependent power management strategies for processors [64] [17] [62] [58] [63] [54] [23], memory systems [21] [22] [25] [28], and disks [15] [34] [59]. Rather, the focus of this work is on idle power reduction. Idle power reduction could be achieved via (1) platform and component leakage power reduction [30] [38]; (2) coarse grain system power management and scheduling [50] [8] [51] [44]; and (3) fine-grained idle power management during active execution (e.g., [33] [3]). Our work falls in the last category. Active state, fine-grained idle power management work can be further classified in to four main techniques.

Unit level power management. Prior work proposes techniques to power-gate sub-components [26] [36], but they deal primarily with execution units and not full cores. We are primarily interested in core or module-level power gating.

Processor sleep-state selection. Processor sleep-state selection algorithms have been studied extensively in the past. Pallipadi *et al.* [57] introduced the use of moving averages as a mechanism to perform idle duration prediction. We compare against this baseline and demonstrate better prediction performance. Min *et al.* [53] propose a classification scheme for interrupts and develop different predictors for different interrupt sources (e.g., separate predictors for deterministic and irregular interrupts). Those techniques require information on interrupt sources from the OS and hence are useful for software-driven power gating. However, the focus of our work is hardware based power gating in the system management unit.

Diao *et al.* [24] propose a machine learning model to predict optimum state entry. Benini *et al.* [11] propose Markov model based modeling techniques to understand system idle and active state behavior. Chung *et al.* [19] develop a learning tree based idle duration prediction mechanism. While these sophisticated algorithms help improve prediction accuracy, they incur high computational costs. Our focus is on active workloads where state decisions need to be made every few $u$secs. Moreover, none of these schemes take cache dirtiness into account. We demonstrate that even Oracle prediction that does not use cache dirtiness information is insufficient.

Interrupt manipulation. Another approach to idleness management is to merge, coalesce, or align interrupts via scheduling and interrupt management. By suitably manipulating interrupts, a system can create longer or more predictable idle durations [52] [43] [37] [65]. Amur *et al.* [9] propose interrupt batching in the context of VM idle power management. Our proposed techniques are orthogonal to interrupt manipulation proposals and such manipulation techniques can augment the mechanism we propose.

Mitigating effects of gating. Lungu *et al.* [45] and Madan *et al.* [46] both demonstrate that it is difficult to develop a robust power-management scheme that works for all cases and highlight the need for guarded power gating, which disables power gating when it produces negative results. This is similar to the strategy used in the Intel Sandy Bridge processor [61] in which a high idle-to-active transition rate results in an auto-demotion mechanism that disables C6 entry until a more stable phase is reached.

Techniques proposed by Min *et al.* [53] account for the performance penalties of frequent mis-predictions of various target predictors. Amur *et al.* [9] develop a mechanism to account for and incorporate models of cache cold miss effects in making deep sleep entry decisions. Although these are viable mechanisms for facilitating better use of deep-sleep modes, this research is orthogonal to our work. All of these techniques can be used in conjunction with the proposed research.

VII. Conclusion

This work provides a comprehensive analysis of idle behavior in emerging applications on CPU-GPU systems. It shows that there is no fixed break-even point in core power gating because of the effects of cache dirtiness. It evaluates existing state-of-the-art systems, and demonstrates new techniques to predict when to enter the C6 power gated state. These schemes are shown to provide average energy reduction exceeding 8% over existing schemes, and sacrifice almost no performance.

Trademark Attribution

AMD, the AMD Arrow logo, AMD Opteron, AMD Radeon, SimNow and combinations thereof are trademarks of Advanced Micro Devices, Inc. SPEC is a registered trademark of Standard Performance Evaluation Corporation. Windows and DirectX are registered trademarks of Microsoft Corporation. PCMark is a registered trademark of Futuremark Corporation. Other names used herein are for identification purposes only and may be trademarks of their respective companies.

References

[1] http://www.hsafoundation.com.
[2] AMD SimNow simulator. http://developer.amd.com/tools-and-sdks/cpu-development/simnow-simulator/.
[3] The idle governor in linux kernel. http://www.kernel.org.
[4] Windows performance analysis. http://msdn.microsoft.com/en-us/performance/cc825801.

[5] Advanced configuration and power interface specification, revision 5.0, 2011. http://www.acpi.info.

[6] PCMark 7 PC performance testing white paper, 2011. http://www.futuremark.com/benchmarks/pcmark.

[7] Bios and kernel developer's guide (bkdg) for amd family 15h models 00h-0fh processors, Jan 2012.

[8] Y. Agarwal, S. Savage, and R. Gupta. Sleepserver: A software-only approach for reducing the energy consumption of pcs within enterprise environments. In *USENIX Annual Technical Conference*, 2010.

[9] H. Amur, R. Nathuji, M. Ghosh, K. Schwan, and H. H. S. Lee. Idlepower: Application-aware management of processor idle states. In *Workshop on Managed Many-Core Systems*, 2008.

[10] L. Barroso and U. Holzle. The case for energy-proportional computing. In *IEEE Computer*, 2007.

[11] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli. Policy optimization for dynamic power management. In *Transactions of Computer Aided Design of Integrated Circuit Systems*, 2006.

[12] W. L. Bircher and L. John. Predictive power management for multi-core processors. In *ISCA*, 2012.

[13] D. Blaauw, S. Martin, T. Mudge, and K. Flautner. Leakage current reduction in vlsi systems. In *Journal of Circuits, Systems, and Computers*, 2002.

[14] G. Blake, R. G. Dreslinski, T. Mudge, and K. Flautner. Evolution of thread-level parallelism in desktop applications. In *ISCA*, 2010.

[15] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *ICS*, 2003.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, 2009.

[17] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *ISLPED*, 2004.

[18] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai. Single-chip heterogeneous computing: Does the future include custom logic, fpgas, and gpgpus? In *MICRO*, 2010.

[19] E. Y. Chung, L. Benini, and G. De Micheli. Dynamic power management using adaptive learning tree. In *ICCAD*, 1999.

[20] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Architecture support for accelerator-rich cmps. In *DAC*, 2012.

[21] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Hardware and software techniques for controlling dram power modes. In *IEEE Transactions on Computers*, 2001.

[22] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: Active low-power modes for main memory. In *ASPLOS*, 2011.

[23] G. Dhiman, K. K. Pusukuri, and T. Rosing. Analysis of dynamic voltage scaling for system level energy management. In *Conference on Power Aware Computing and Systems*, 2008.

[24] Q. Diao and J. Song. Prediction of cpu idle-busy activity pattern. In *HPCA*, 2008.

[25] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini. Limiting the power consumption of main memory. In *ISCA*, 2007.

[26] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman. Managing static leakage energy in microprocessor functional units. In *MICRO*, 2002.

[27] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.

[28] X. Fan, C. S. Ellis, and A. R. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. In *Conference on Power Aware Computer Systems*, 2004.

[29] T. Fischer, S. Arekapudi, E. Busta, C. Dietz, M. Golden, S. Hilker, A. Horiuchi, K. Hurd, D. Johnson, H. McIntyre, S. Naffziger, J. Vinh, J. White, and K. Wilcox. Design solutions for the bulldozer 32nm soi 2-core processor module in an 8-core cpu. In *ISSCC*, 2011.

[30] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: Simple techniques for reducing leakage power. In *ISCA*, 2002.

[31] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge. Thread-level parallelism and interactive performance of desktop applications. In *ASPLOS*, 2000.

[32] D. Foley, M. Steinman, A. Branover, G. Smaus, A. Asaro, S. Punyamurtula, and L. Bajic. Amd's llano fusion apu. In *Hot Chips*, 2011.

[33] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *USENIX Technical Conference Proceedings*, 1995.

[34] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: Dynamic speed control for power management in server class disks. In *ISCA*, 2003.

[35] R. Hou, L. Zhang, M. C. Huang, K. Wang, H. Franke, Y. Ge, and X. Chang. Efficient data streaming with on-chip accelerators: Opportunities and challenges. In *HPCA*, 2011.

[36] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural techniques for power gating of execution units. In *ISLPED*, 2004.

[37] H. Huang, K. G. Shin, C. Lefurgy, and T. Keller. Improving energy efficiency by making dram less randomly accessed. In *ISLPED*, 2005.

[38] X. Huang, W. C. Lee, C. Kuo, D. Hisamoto, J. Kedzierski, E. Anderson, H. Takeuchi, Y. K. Choi, K. Asano, V. Subramanian, T. J. King, J. Bokor, and C. Hu. Sub 50nm finfet: Pmos. In *Electron Devices Meeting*, 1999.

[39] Intel. Energy-Efficient Platforms: Considerations for Application Software and Services. 2011.

[40] M. Kamruzzaman, S. Swanson, and D. M. Tullsen. Software data spreading: Leveraging distributed caches to improve single thread performance. In *PLDI*, 2010.

[41] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power management of datacenter workloads using per-core power gating. In *IEEE Compututer Architecture Letters*, 2009.

[42] N. Levinson. The wiener rms (room mean square) error criterion in filter design and prediction. In *Journal of Mathematics and Physics*, 1947.

[43] J. Liu and P. H. Chou. Optimizing mode transition sequences in idle intervals for component-level and system-level energy minimization. In *ICCAD*, 2004.

[44] Y. Liu, S. C. Draper, and N. S. Kim. Sleepscale: Runtime joint speed scaling and sleep states management for power efficient data centers. In *ISCA*, 2014.

[45] A. Lungu, P. Bose, A. Buyuktosunoglu, and D. J. Sorin. Dynamic power gating with quality guarantees. In *ISLPED*, 2009.

[46] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram. A case for guarded power gating for multi-core processors. In *HPCA*, 2011.

[47] J. Makhoul. Linear prediction: A tutorial review. In *Proceedings of the IEEE*, 1975.

[48] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy management. In *ISCA*, 1998.

[49] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: Eliminating server idle power. In *ASPLOS*, 2009.

[50] D. Meisner, B. T. Gold, and T. F. Wenisch. The powernap server architecture. *ACM Transactions on Computer Systems*, 2011.

[51] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *ISCA*, 2011.

[52] D. Meisner and T. F. Wenisch. Dreamweaver: Architectural support for deep sleep. In *ASPLOS*, 2012.

[53] A. W. Min, R. Wang, J. Tsai, M. A. Ergin, and T. Y. C. Tai. Improving energy efficiency for mobile platforms by exploiting low-power sleep states. In *Computing Frontiers*, 2012.

[54] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *ICS*, 2002.

[55] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the intel core duo processors. In *Intel Technology Journal*, 2006.

[56] S. Nussbaum. AMD Trinity APU. In *Hot Chips*, 2012.

[57] V. Pallipadi and A. Belay. Cpuidle do nothing efficiently. In *The Linux Symposium*, 2006.

[58] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, 2001.

[59] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *ICS*, 2004.

[60] A. Rogers, D. Kaplan, E. Quinnell, and B. Kwan. The core-C6 (CC6) sleep state of the AMD bobcat x86 microprocessor. In *ISLPED*, 2012.

[61] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. In *IEEE Micro*, 2012.

[62] R. Sarikaya, C. Isci, and A. Buyuktosunoglu. Runtime workload behavior prediction using statistical metric modeling with application to dynamic power management. In *IISWC*, 2010.

[63] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli. Dynamic voltage scaling and power management for portable systems. In *DAC*, 2001.

[64] J. L. W. Lloyd Bircher, M. Valluri and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *ISLPED*, 2005.

[65] R. Wang, J. Tsai, C. Maciocco, T. C. Tai, and J. Wu. Reducing power consumption for mobile platforms via adaptive traffic coalescing. In *IEEE Journal on Selected Areas in Communications*, 2011.