2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

PowerChop: Identifying and Managing Non-critical Units in Hybrid Processor Architectures

Michael A. Laurenzano, Yunqi Zhang, Jiang Chen*, Lingjia Tang and Jason Mars
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor
{mlaurenz, yunqi, jiangc, lingjia, profmars}@umich.edu

***Abstract*—On-core microarchitectural structures consume significant portions of a processor's power budget. However, depending on application characteristics, those structures do not always provide (much) performance benefit. While timeout-based power gating techniques have been leveraged for under-utilized cores and inactive functional units, these techniques have not directly translated to high-activity units such as vector processing units, complex branch predictors, and caches. The performance benefit provided by these units does not necessarily correspond with unit activity, but instead is a function of application characteristics.**

This work introduces POWERCHOP, a novel technique that leverages the unique capabilities of HW/SW co-designed hybrid processors to enact unit-level power management at the application phase level. POWERCHOP adds two small additional hardware units to facilitate phase identification and triggering different power states, enabling the software layer to cheaply track, predict and take advantage of varying *unit criticality* across application phases by powering gating units that are not needed for performant execution. Through detailed experimentation, we find that POWERCHOP significantly decreases power consumption, reducing the leakage power of a hybrid server processor by 9% on average (up to 33%) and a hybrid mobile processor by 19% (up to 40%) while introducing just 2% slowdown.

I. INTRODUCTION

A power saving technique that can prove critical for energy-efficient processor design is *unit-level power gating*. Unit-level power gating is a mechanism for dramatically reducing static power consumption by cutting the supply voltage to a circuit block within a core, and can be applied at various granularities [1]–[4]. Although timeout-based power gating techniques have been shown to be effective for underutilized whole cores and inactive functional units, these techniques have not translated to large, stateful, performance-critical units such as the vector processing unit (VPU), middle-level cache (MLC), and branch prediction unit (BPU). The challenges in power gating this class of units include:

1) **Unit Criticality –** depending on the characteristics of the executing application code, these units can be critical for performance. Performance can be significantly hindered if the unit is gated off at a time when it could help application performance.
2) **Statefulness –** these units may contain state that must be managed if power gated to the point of retention loss (e.g., the register file in a VPU or branch history in a BPU). The saving, restoring, and management of state when power gating these units can introduce significant performance overheads if gated on and off at high frequency.
3) **High Activity –** these units are often active, regardless of the performance benefit they provide. For example, whether a memory operation results in an MLC hit or miss, the MLC handles the memory operation. This property thwarts conventional timeout-based approaches to power gating, as these units are not subject to lengthy periods of idleness [5]. The nearly continuous activity within such units also makes it challenging to design decision mechanisms to determine when the unit is needed again for high performance.
4) **Application Behavior –** detecting whether a unit may become more or less critical and understanding the duration of criticality is needed to identify when gate a unit on and off. This requires a mechanism to monitor and analyze application behavior dynamically, and hardware-only techniques to perform this introspection may introduce significant additional complexity.

The key underlying insight of this work is that the hardware/software co-design of hybrid processors is uniquely capable of addressing these challenges. We broadly define a *hybrid architecture* as one that leverages hardware/software co-design to couple a software binary translation subsystem with the architectural design of the processor. There has been a resurgence of interest in this class of designs with the recent release of NVIDIA's Project Denver [6], [7]. In the design of Project Denver and its predecessors [8], [9], the software component takes the form of a binary translation (BT) and optimization subsystem sitting below the ISA interface, and is integral to the specification of the microarchitectural design.

The software component of a hybrid processor provides a mechanism that can be leveraged to facilitate the monitoring of execution and to infer properties about the characteristics of the executing workload that indicate the criticality of underlying microarchitectural units. We define *unit criticality* as the performance benefit a unit provides for the executing application code. In addition to analyzing the executing instruction stream, the BT capability of the software system can enable the tailoring of the instruction stream to steer execution away from non-critical units. For example, infre-

*Jiang Chen is currently a software engineer at Google Inc., Mountain View, CA, USA.

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.22

quently executed vector instructions can be transformed into equivalent non-vector instructions to facilitate power gating the VPU during a phase of low VPU criticality. Because the software subsystem can absorb the complexity required to monitor execution and make changes to the instruction stream to enable intelligent power gating, an approach that leverages this subsystem can to avoid the addition of a potentially prohibitive amount of hardware complexity.

In this work, we present POWERCHOP, a HW/SW co-designed approach that enables sophisticated unit-level power management decisions based on continuous monitoring of unit criticality. The cornerstone of POWERCHOP's design is a novel mechanism for continuous unit criticality analysis and code attribution. Throughout execution, phase signatures composed of code region identifiers are collected and monitored. Dynamic profiles of unit criticality are attributed to phases and stored via two small hardware structures. The software subsystem leverages these structures to dynamically detect phase edges and recall unit criticality profiles of previously-seen phases, using that information to configure power gating at the unit level across application phases. The specific contributions of this work are:

- We introduce POWERCHOP, a technique for identifying and managing non-critical microarchitectural units on HW/SW co-designed hybrid processor architectures, power gating units when they do not provide a performance benefit.
- We describe an approach for application phase identification and unit criticality attribution on hybrid processor architectures that leverages two small additional hardware units.
- We design phase triggered power gating policies for three large, stateful, performance-critical architectural units – the VPU, BPU, and MLC.
- We perform a thorough evaluation of POWERCHOP for each of these three units, as well as for the entire POWERCHOP system for server and mobile processor designs across a spectrum of workload classes that include SPEC CPU2006, PARSEC, and MobileBench.

We find that POWERCHOP significantly reduces power consumption, lowering the leakage power draw of the server processor by 9% on average (up to 33%) and the mobile processor by 19% (up to 40%), introducing an average of 2% slowdown.

II. BACKGROUND

This section provides the foundational background on HW/SW co-designed hybrid processors and unit-level power gating necessary to understand the remainder of this paper.

A. Hybrid Processor Architectures

Commercial implementations of hardware/software co-designed *hybrid* processor architectures include NVIDIA's Project Denver [6], [7], as well as the Efficeon and Crusoe processors from Transmeta [8], [9]. Common to the design of these architectures is the presence of a binary translation (BT) software layer sitting atop the hardware. The BT layer runs all system and application software, translating from code supplied to the guest ISA[1] – the ISA exposed to system and application software – to a proprietary host ISA implemented in hardware.

The BT subsystem is central to a hybrid architecture design, and in this work the BT is designed to resemble the Transmeta BT [8]. The BT consists of three principle components – the interpreter, the translator and the nucleus. The interpreter decodes and executes guest instructions sequentially while collecting statistics about execution and branch behavior. When a particular region of guest code has reached a certain hotness threshold, the interpreter yields to the translator. The translator produces a highly-optimized version of the guest code region for the host ISA. This optimized region of host-ISA code, called a *translation*, is then inserted into a software structure called the region cache. Subsequent executions of the code region can thus occur from the translation in the region cache, without incurring additional interpretation and translation costs. The nucleus is responsible for handling interrupts and exceptions at both the host ISA level and in the microarchitecture, for example when recovering from mis-speculated load/store reorderings. Further details on the BT subsystem of hybrid processor architectures can be found in prior work [8]–[10].

B. Unit Level Power Gating

Power gating is a technique that reduces the power consumed by a circuit block by cutting its supply voltage. This technique can be applied at a range of granularities, including at the core-level [11]–[14], and for large units within the core [2], [15]. Given a logical circuit block, a sleep transistor is used to control the supply voltage to the block. When a sleep signal is asserted to the sleep transistor, the unit is said to be *gated off*, causing it to lose its state and functionality. While gated off, the unit has a minimal amount of static leakage and switching (dynamic) power. When the sleep signal is deasserted by restoring its voltage, the unit is said to be *gated on*, allowing the unit to function as normal. As opposed to *clock gating*, which reduces dynamic power, power gating reduces both static and dynamic power. However, power gating incurs overheads in terms of both time and power to wait for the sleep signal to be distributed through the sleep transistor and to drive V_{dd} when power is restored to the unit. Detailed discussions of these overheads and their implications for power gating units can be found in prior work [2].

III. OPPORTUNITIES AND CHALLENGES

There are a number of performance critical units that consume a significant fraction of the power budget of the core. However, the performance benefit they provide varies across applications and across execution phases within an application. This non-uniformity in the criticality of units

[1]The guest ISA of Project Denver is ARMv8, and the guest ISA of the Transmeta designs is x86.

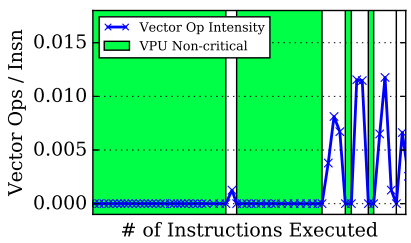


Figure 1. Vector operation intensity over 200 thousand instructions of `gobmk`; VPU criticality varies across execution

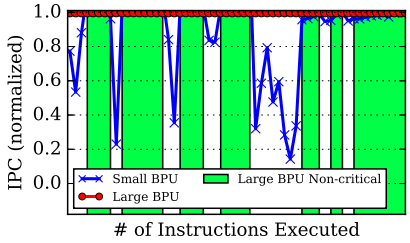


Figure 2. Small (local) vs. large (tournament) branch predictors over 13 million instructions of MobileBench `msn`

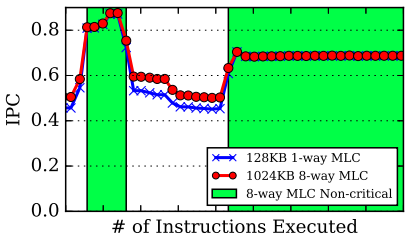


Figure 3. 128KB 1-way vs. 1024KB 8-way L2 cache performance over 120 million instructions of `GemsFDTD`

provides power gating opportunities when the performance benefit of keeping them powered on is marginal.

A. Variable Unit Criticality

Figure 1 depicts the intensity of vector operations over 200 thousand instructions of `gobmk` from SPEC CPU2006. As shown in Figure 1, the intensity of vector operations and the criticality of the VPU vary over time. When these periods of low-criticality are long and the overhead of powering the unit on and off can be justified, the VPU could be power gated during periods of low criticality to reduce power consumption, with a minimal impact on performance. It is important to note that the low-criticality periods include times when instructions hitting the VPU are scarce but non-zero, a behavior that is difficult to take advantage of using conventional approaches based on timeouts [2].

Modern branch predictors often leverage multiple branch prediction approaches (local, global, hybrid, adaptive, agree, neural, etc.), predicting branch outcomes using a tournament of conventional approaches. The rationale behind tournament branch predictors is that each of the small predictors may be useful for accurately predict branch outcomes among a subset of applications or phases, however accurately predicting branches across *all* applications and phases may need to consider multiple such predictors. Figure 2 presents the IPC over time for a web browser running on a mobile processor using a small local branch predictor (Small BPU) and a larger tournament local/global predictor (Large BPU). Unsurprisingly, using the large BPU instead of the small BPU improves IPC overall. However, the performance benefit provided by the large BPU is negligible during many phases of execution. During those phases, the large branch predictor has low criticality, suggesting that there may be an opportunity to power gate parts of the BPU during various phases of execution, saving power without sacrificing performance.

Figure 3 illustrates the opportunity for power gating parts of the MLC, showing the IPC of a server processor running `GemsFDTD` with either a 128KB 1-way MLC or a 1024KB 8-way MLC. When the working set fits into the full 8-way MLC but not into the 1-way MLC, having the full MLC can provide significant performance benefit. However, when the working set is small enough to fit in L1 or is too large to fit in the MLC (e.g., streaming from memory), the benefit of having the full MLC diminishes. In such situations, parts of the MLC could be power gated without significantly impacting performance.

B. Challenges of Unit-level Management

Taking advantage of these opportunities by power gating at the unit level is challenging. Firstly, as shown in Figures 1-3, during certain periods units can be critical for application performance. Gating off these units requires a careful understanding of application behavior, as an incorrect gating decision may cause significant performance degradations.

Secondly, units can exhibit activity even while they are non-critical for performant execution. The VPU may be put to use occasionally by application code, but be used infrequently enough that the VPU lacks criticality (i.e., a low but non-zero number of vector operations occur during execution). Moreover, consider Figures 2 and 3, where the large BPU and MLC, respectively, exhibit low performance criticality during certain phases of execution, but are nevertheless continuously active throughout all of execution. Recent work has demonstrated that these high levels of activity are the common case for the BPU and MLC, with branches accounting for around 1 in 7 instructions in mobile workloads, while MLC accesses occur around 1 in 125 instructions [5]. Thus, it is difficult to adopt a strategy based on timeouts that would be able to identify and take advantage of periods of low-criticality among these highly active units.

Thirdly, these units can contain architectural or microarchitectural state. The VPU has a register file, the MLC may have dirty lines, and the BPU has a branch target buffer (BTB) and other types of branch history. Power gating too frequently can introduce significant performance overheads for spilling/restoring or losing/reconstituting that state.

Understanding and taking advantage of these opportunities as a dynamic property of the running application requires monitoring and analyzing application execution, which may be complex and costly to implement in hardware. However, as we show in the next section, hybrid processors are uniquely suited to address these challenges, as much of the complexity needed to solve this problem can be absorbed by the software layer included in hybrid designs.

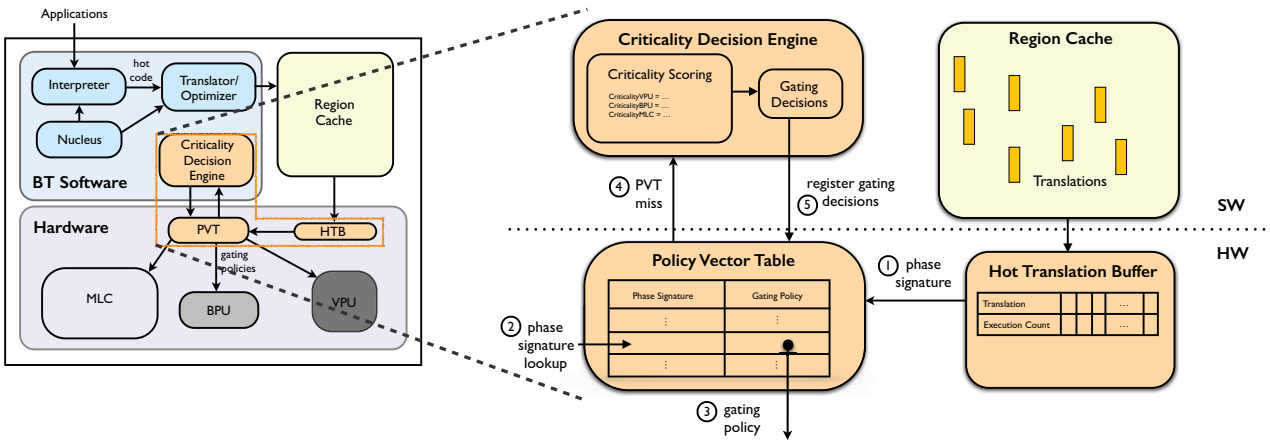


Figure 4. Overview of PowerChop, illustrating how it fits into a conventional hybrid processor architecture (left) and a detailed view of how its components interact (right).

IV. System Design

This work introduces PowerChop, a novel approach for dynamically identifying and taking advantage of non-critical units in hybrid processor architectures.

A. Overview

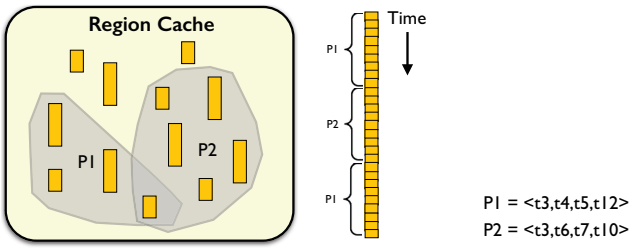
Motivating the design of PowerChop is to develop a system that can identify and manage low-criticality units, reducing power consumption by providing a mechanism that dynamically characterizes application execution to obtain unit criticality metrics at the granularity of application phases. Figure 4 shows an overview of the design of PowerChop. On the left side of the figure, we show PowerChop in the context of a conventional hybrid processor design. PowerChop's design spans both the hardware and software subsystems. The *policy vector table* (PVT) and *hot translation buffer* (HTB) are small hardware structures added by PowerChop that enable low-level continuous phase edge identification and attribution of unit criticality to phases, while characterizing unit criticality and making power management decisions is hoisted into the software subsystem via the Criticality Decision Engine (CDE). The right of Figure 4 shows a detailed view of how these components are used within PowerChop.

Application execution on the hybrid processor occurs in the region cache, which consists of a collection of short traces of dynamic code sequences called translations [8], [9]. PowerChop uses the translation abstraction as a key primitive for phase-based unit criticality analysis and power management. The runtime operation of PowerChop can be characterized as follows:

1. Throughout execution, translation execution counts are maintained by the HTB, which are used to form phase signatures. The HTB reports phase signatures to the PVT for the most recent window of executed translations. The phase signatures function as unique identifiers for the executing application phases.
2. Each dynamically detected phase signature results in a PVT lookup. The PVT is a simple hardware structure that maintains a record of recently executed phase signatures and their corresponding power gating policies that have been defined by the CDE.
3. If a PVT lookup results in a hit, the associated gating decisions are applied to the relevant units.
4. If a PVT lookup results in a miss, the CDE is invoked to handle the miss.
5. When a PVT miss is compulsory, unit criticality for the phase is characterized by the CDE and a power gating policy is assigned. The CDE then registers the phase signature and gating policy with the PVT. Upon a capacity miss, the phase signature and management policy are fetched from memory bu the CDE and placed into the PVT. Evicted entries from the PVT are then stored in memory by the CDE.

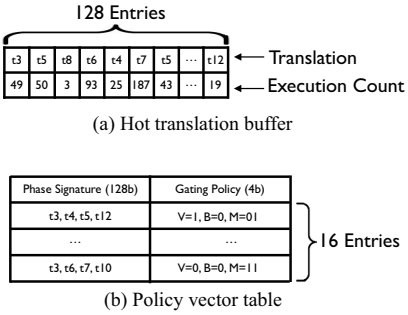
PowerChop's design takes advantages of the relevant strengths of the hardware and software components of the hybrid processor architecture. Hardware continuously monitors the currently executing phases (via the HTB) and triggers power management directives at phase change boundaries (via the PVT), while software characterizes new phases, analyzing unit criticality and configuring the power gating policies at phase edges (via the CDE). Using this design, PowerChop addresses the key challenges that need to be solved to enact unit-level power gating decisions:

1) **Unit Criticality –** by analyzing the code and making power gating decisions on phase edges, PowerChop determines unit criticality, gating off units that are non-critical for performant execution.
2) **Statefulness –** PowerChop can leverage the software runtime to flexibly control the granularity of managing units, minimizing the performance overhead of saving/restoring or losing state.



(a) Active code regions (b) Execution trace (c) Phase signatures

Figure 5. Phase identification in POWERCHOP



(a) Hot translation buffer

(b) Policy vector table

Figure 6. POWERCHOP hardware structures

3) **High Activity –** based on measurements of unit criticality, POWERCHOP can power down high activity units when they are non-critical.
4) **Application Behavior –** POWERCHOP leverages the software component to facilitate application characterization with minimal additional hardware complexity.

In the following subsections, we provide more details on the hardware and software components of POWERCHOP.

B. Hardware Support

Two small hardware structures called the hot translation buffer (HTB) and policy vector table (PVT) are used by POWERCHOP to facilitate identifying phases, attributing unit criticality to phases, and enacting power gating decisions.

1) Phase Identification: Figure 5 illustrates the intuition of the phase recognition mechanism in POWERCHOP. As translations are executed by the processor out of the region cache during execution, these translations correspond to active code regions, denoted P1 and P2 in Figure 5(a). We define an *execution window* as a period of execution, measured in the number of dynamically executed translations. For example, an execution window of 100 would correspond to 100 consecutively executed translations. Figure 5(b) shows POWERCHOP's view of the dynamic sequence of execution windows that correspond to P1 and P2, showing three consecutive execution windows – P1, P2, then P1. To uniquely identify phases, POWERCHOP builds a *phase signature* using the hottest N translations from each execution window. Figure 5(c) shows an example of the phase signatures that identify P1 and P2 ($N = 4$ in the example).

Care must be taken in choosing the phase signature length N and the execution window size. If the phase signature length is too long, the phase signature may contain insignificant traces that are unlikely to recur. If the trace signature length is too short, distinct phases may not be treated as distinct. Similarly, the window size can impact the quality of the phase recognition approach. Larger window sizes may miss short phases, while short window sizes may result in phases dominated by short-lived, transient behavior, potentially causing frequent power gating policy changes. To arrive at these parameter settings in designing POWERCHOP we performed a sensitivity analysis, finding that using a trace signature length of 4 and a window size of 1000 translations proves effective across a wide range of workloads.

2) Hot Translation Buffer: To facilitate continuous phase signature collection and identification, POWERCHOP introduces a simple hardware structure called the *hot translation buffer* (HTB), illustrated in Figure 6(a). The HTB is a fully associative hardware buffer that tracks translations as they execute along with the number of dynamic instructions executed on each translation.

The program counter (PC) of a translation head uniquely identifies translations in our BT. We use the lower 32-bits of the instruction at each translation head's PC as a unique ID for each translation (the region cache is typically far smaller than 32-bits, guaranteeing that these 32-bits are unique). Unique translation IDs are denoted in the figure as $t1$, $t2$, etc. in Figure 6. Tracking head PCs is facilitated by the introduction of a new bit into the instruction format of the host ISA to indicate whether the instruction is a translation head, as well as a performance counter to track the number of instructions seen between translation heads. The translation and execution counts within the HTB are updated as a side effect of translation head execution, occurring off the critical path of execution.

As each translation is executed, if it is already present in the HTB, its associated dynamic instruction count is incremented by the number of instructions executed since the previous translation. Otherwise, the new translation is added to the HTB and its dynamic instruction count is initialized to the value in the counter. If the number of unique translations in the current execution window exceeds the size of the HTB, it is simply ignored. Throughout this work, we use a HTB size of 128 for a window size of 1000 translations to track phases. As a result, the HTB holds a record of the dynamic instruction counts for each unique translation executed in the current execution window. At the end of each execution window, the HTB initiates a PVT lookup and the HTB is flushed for the next execution window.

3) Policy Vector Table: The *policy vector table* (PVT) is a small structure containing the recent history of uniquely executed phases. The PVT functions as a fully-associative cache, maintaining a record of recently executed phase signatures and for each signature, a corresponding power gating policy. The gating policy takes the form of a bit

Algorithm 1 Criticality Decision Engine (CDE)

```
while CDE is invoked do
    if is new phase then
        collect performance statistics;
        if profiling is complete then
            register to PVT;
        else
            insufficient information, keep collecting;
        end if
    else
        if old phase being profiled then
            collect performance statistics;
            if profiling is complete then
                register to PVT;
            else
                insufficient information. keep collecting;
            end if
        else
            // is an old phase and has been profiled
            re-register to PVT;
        end if
    end if
end while
```

vector that defines the power gating state for each of the logical units controlled by POWERCHOP.

Figure 6(b) illustrates the design of the PVT for the three unit types POWERCHOP currently supports: the vector processing unit (VPU), branch prediction unit (BPU) and middle-level cache (MLC). In the figure, we show 2 phase signatures, each with their associated policy for the VPU (V), BPU (B) and MLC (M). The policies for the VPU and BPU are bimodal, with `1` representing the gated-on state and `0` representing the gated-off state. The MLC uses a finer-grain policy, having 3 power gating states that allow it to be configured to have all ways gated on, half the ways gated on, or a single way gated on. Thus, the power gating policy for the MLC uses two bits. The number of states for each unit can be increased by increasing the number of bits used in the PVT to represent the power states of that unit. The PVT holds 16 entries for recent phases that have been executed. As new phases are registered (written into the PVT) by the Criticality Decision Engine, stale phase signatures are evicted using an approximate LRU replacement policy.

4) Hardware Costs: The PVT in our design uses 16 entries, totalling 264 bytes (each entry has 4× 32-bit PCs plus 4-bits for power states). The HTB is 128 entries and 1 KB storage (32-bits per translation ID and 32-bits per execution counter). Using cacti simulations [16], we find that the power (0.027 W) and area (0.008 mm^2) needed for the HTB are small, particularly in comparison to the power and area budgets typical of modern processor designs.

C. Software Subsystem Support

The Criticality Decision Engine is responsible for characterizing unit criticality and determining the power gating policy for each phase signature.

1) Criticality Decision Engine: The CDE is implemented as an addition to the BT subsystem of the hybrid processor. Model specific registers (MSRs) are used as the primary interface between the PVT and the CDE. Algorithm 1 presents a summary of the core functionality of the CDE. When invoked via a PVT miss, the CDE performs one of three actions:

- **New Phase –** if the CDE finds that a new phase has been detected (i.e., the phase has never been seen before), it records the phase as being in *profiling mode*. Profiling information is then collected for the next execution window from hardware performance monitors. If the information gathered from one execution window is sufficient for unit criticality analysis, the phase and the power gating policy are then registered to the PVT immediately. Otherwise, the phase is not registered and will result in subsequent invocations of profiling the next time the phase is executed. Details of gating policies and examples of both of these situations are presented shortly in Section IV-C2.
- **Continued Phase Profiling –** when the CDE finds a phase signature that is in profiling mode, it will gather performance counter information and either remain in profiling mode if more profiling is needed, or register the policy with the PVT when enough profiling information has been collected.
- **Evicted Phase –** if the CDE finds a phase signature that is already characterized but was previously evicted from the PVT, the CDE re-registers that phase signature and its gating policy with the PVT. An approximate LRU eviction policy selects the phase that is evicted from the PVT to make room for the current phase.

2) Criticality Scoring and Policies: The CDE characterizes the unit criticality for a phase based on the information gathered during a profiling window. This section describes the approaches used to characterizing criticality for the three power-hungry unit types supported by POWERCHOP – the VPU, BPU and MLC.

VPU. POWERCHOP uses the ratio of SIMD instructions committed during a phase $\mathrm{Phase}_{\mathrm{SIMD}}$ to the number of total instructions committed during the phase $\mathrm{Phase}_{\mathrm{TotInsn}}$, to assess the criticality of VPU during a phase $\mathrm{Criticality}_{\mathrm{VPU}}$. These profiles are collected during a single profiling window. When profiling is complete, POWERCHOP assigns the gate-off policy to the VPU if $\mathrm{Criticality}_{\mathrm{VPU}}$ fails to exceed a threshold $\mathrm{Threshold}_{\mathrm{VPU}}$. When the VPU is gated off, any instructions bound for the VPU (e.g., SSE and AVX instructions in x86, or NEON instructions in ARM) are emulated using scalar operations emitted along alternate code paths in the region cache's translations.

BPU. For the BPU, POWERCHOP uses two profiling windows to assess the criticality of a large tournament branch predictor relative to a small local predictor. $\mathrm{MisPred}_{\mathrm{Large}}$ is the misprediction rate for the large predictor during a first profiling window, and $\mathrm{MisPred}_{\mathrm{Small}}$ is the misprediction rate for the small predictor from a second profiling window. POWERCHOP uses the difference between these two misprediction rates as the criticality of the large predictor $\mathrm{Criticality}_{\mathrm{BPU}}$, assigning the gate-off policy to the BPU if $\mathrm{Criticality}_{\mathrm{BPU}}$ fails to exceed a threshold $\mathrm{Threshold}_{\mathrm{BPU}}$.

MLC. For the MLC, POWERCHOP assesses unit criticality by profiling a single window to measure the number of L2 cache hits and the number of total instructions executed during the window, $\mathrm{Phase}_{\mathrm{L2Hit}}$ and $\mathrm{Phase}_{\mathrm{TotInsn}}$, respectively. The criticality of the MLC $\mathrm{Criticality}_{\mathrm{MLC}}$ is defined as the ratio of these two values. POWERCHOP keeps either 1 way, half the ways, or all ways of the MLC in an active state, allowing the MLC to service requests at all times while allowing for significant reductions in power consumption. This design uses two thresholds to assign gating policies, leaving all ways active if $\mathrm{Criticality}_{\mathrm{MLC}}$ exceeds a threshold $\mathrm{Threshold}_{\mathrm{MLC1}}$, leaving 1 way active if $\mathrm{Criticality}_{\mathrm{MLC}}$ does not exceed a second threshold $\mathrm{Threshold}_{\mathrm{MLC2}}$, and leaving half the ways active otherwise.

3) Software Costs: Because the BT subsystem already provides support for region cache, translations and interrupts, much of the software complexity in POWERCHOP is absorbed by the existing BT subsystem. The most significant additional source of overhead over the conventional BT are additional interrupts triggered by PVT misses. Experiments show that an average 0.017% of translations across the SPEC CPU2006 benchmarks cause PVT misses, resulting in less than 0.5% additional performance overhead on average.

D. Unit-level Power Gating

Power gating is implemented by adding a header or footer transistor to the block that is to be power gated. A sleep signal is applied to the header/footer sleep transistor, which cuts the supply voltage to the block. Even when a unit is gated, its supply voltage is non-zero. We therefore assume that the leakage power of a gated unit is reduced to 5% of its non-gated leakage power. To incorporate the energy overhead E_{Overhead} of asserting and deasserting the sleep signal to the header/footer transistor, we use the model proposed by Hu et. al. [2] and summarized by Equation 1.

$$E_{Overhead} = 2\frac{W_H}{\alpha}E^S_{cyc} \quad (1)$$

Determining E_{Overhead} for a unit thus requires determining three parameters — the average switching energy of the unit for a single cycle E^S_{cyc}, the ratio of the area of the sleep transistor to the unit W_H and the average switching factor for the unit α. We find E^S_{cyc} for a unit from a McPAT [17] estimate of that unit's peak dynamic power. For W_H, estimates in the literature range between 0.05 to 0.20 [2], [18]–[20]; for the purpose of modeling $E_{Overhead}$ we assume a value of 0.20, which results in the largest energy overhead from this range of estimates. For the switching factor α, we use a value of 0.05.

Power gating also incurs a performance cost, as the affected unit will be idle while the sleep signal is distributed through the sleep transistor and while V_{dd} is restored to the unit. To model this performance impact, we assume that all application execution is paused while the unit is being gated on or off. We apply a 50 cycles penalty when gating the MLC, 30 cycles when gating the VPU, and 20 cycles for gating the BPU.

Finally, microarchitectural units may have state that cannot be retained when the unit is gated off. For instance, the MLC and BPU have microarchitectural state that includes cache lines and branch history, respectively, while a VPU may contain architecturally-visible registers in a register file. In this work, we assume that most microarchitectural state is lost when a unit is power gated. The exception to this is dirty lines in the MLC, which must be written back to last level cache when the MLC is gated off. Moreover, we assume that the VPU contains a register file that is explicitly saved and restored when the VPU undergoes gating policy transitions, applying a 500 cycle penalty when the VPU is gated on or off. In the case of lost microarchitectural state, the relevant microarchitectural structures must be re-warmed as application execution continues. In the case of writing back dirty MLC lines and saving/restoring VPU registers, we assume that application execution is halted whilst those operations occur. In all cases, we measure the impact of these overheads via detailed architectural simulation.

V. EVALUATION

We next evaluate POWERCHOP, using detailed simulation to observe its impact on performance and power consumption across server and mobile processor designs.

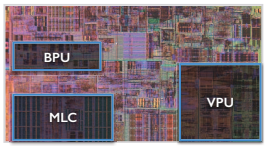
A. Methodology

Applications and Software Stack. We evaluate POWERCHOP on a range of applications from SPEC CPU2006 [21], PARSEC [22] and the Realistic General Web Browsing (R-GWB) benchmarks from MobileBench [23]. PARSEC and SPEC are used with the server processor, while MobileBench is used with the mobile processor. Our server configuration runs on Linux kernel version 3.2. MobileBench R-GWB is a set of web browsing benchmarks, and our experiments run each benchmark inside the web browser on the full Android software stack and Android browser.

Simulation Environment. Power is modeled for a 32nm technology node using McPAT [17]. We use detailed architectural simulation in gem5 [24] to evaluate POWERCHOP, using SimPoint [25] to select simulation regions. The key overheads that are addressed in simulation to fully account for the impact of POWERCHOP include the overhead of application idle time while power gating takes effect and the impact of dealing with unit state, both discussed in detail in Section IV-D, as well as the overhead of running without the benefit of the units gated off by POWERCHOP.

Architecture. Our evaluation covers two processor designs points, reflecting server and mobile configurations as shown in Figure 7. The key characteristics of these architectures, the units managed by POWERCHOP within the processor, and additional summary information about the power gating operations of the processors are summarized in Table I.

Criticality Thresholds. $\mathrm{Threshold}_{\mathrm{VPU}}$, $\mathrm{Threshold}_{\mathrm{BPU}}$ and $\mathrm{Threshold}_{\mathrm{MLC1}}$ are all set to 0.001 in this work, while $\mathrm{Threshold}_{\mathrm{MLC2}}$ is set to 0.0001. We have found



(a) Server core - Intel Nehalem

(b) Mobile core - ARM Cortex-A9

Figure 7. Server/mobile core diagrams, highlighting the key units used by POWERCHOP

Table I
SUMMARY OF ARCHITECTURAL DESIGN POINTS USED IN THE EVALUATION

		Server Processor Configuration	Mobile Processor Configuration
	Applications	SPEC CPU2006 [21], PARSEC [22]	MobileBench [23]
MLC	**Baseline**	1024KB, 8-way	2048KB, 8-way
	Area	35% of core	60% of core
	Gated Off	512KB 4-way or 128KB 1-way	1024KB 4-way or 256KB 1-way
	State	WB dirty lines, lose clean lines, rewarm	WB dirty lines, lose clean lines, rewarm
	Overheads	50 cycles/switch + WB + rewarm	50 cycles/switch + WB + rewarm
VPU	**Baseline**	4-wide SIMD	2-wide SIMD
	Area	20% of core	18% of core
	Gated Off	unit off, ops emulated by BT	unit off, ops emulated by BT
	State	save/restore register file to memory	save/restore register file to memory
	Overheads	30 cycles/switch + 500 cycle save/restore	30 cycles/switch + 500 cycle save/restore
BPU	**Baseline**	loc/glob tourney, 4K-ent BTB, 16K-ent chooser	loc/glob tourney, 2K ent BTB, 8K-ent chooser
	Area	4% of core	3% of core
	Gated Off	local only, 1K-entry BTB	local only, 512-entry BTB
	State	lose global, chooser and BTB state, rewarm	lose global, chooser and BTB state, rewarm
	Overheads	20 cycles/switch + rewarm	20 cycles/switch + rewarm

these thresholds to work well to enable significant power draw reductions while minimizing the performance impact of critical units being gated off. Alternative policies to this are also possible, such as more aggressive policies using higher thresholds that target energy minimization.

B. Phase Identification

POWERCHOP leverages its online phase recognition capability (Section IV-B1) to identify execution phases during which units have low performance criticality in order to make power gating decisions. The quality of the online phase recognition in capturing application execution phases that have similar properties (executing the same code and exhibiting similar unit performance criticality) is crucial for POWERCHOP's effectiveness at saving power while maintaining high performance.

We evaluate the quality of the phase detection by comparing the code executed across recurrences of each phase. During the application run, a phase signature is generated every 1000 translations. To compare how well the phases detected by POWERCHOP capture what code is being executed, we compare the *translation vectors* between 1000-translation execution windows that are identified by POWERCHOP as being part of the same phase. To generate this comparison, we take the Manhattan distance of each pair of translation vectors in the application that have identical signatures, then compute the average Manhattan distance of all such pairs across application execution. A perfect phase analysis approach would therefore have an average Manhattan distance of 0, indicating that the exact same 1000 translations are executed across all windows recognized by POWERCHOP as the same phase, while a worst-case approach would have a distance of 1000. As illustrated in Figure 8, our approach effectively identifies phases that are executing identical or similar code: the phases characterized by POWERCHOP as having the same phase signatures execute highly overlapping sets of translations. The average Manhattan distance across applications is just 2.8% (28 out of 1000 translations), and never exceeds 6.8%.

C. Per-unit Analysis

We now evaluate POWERCHOP's effectiveness in gating the VPU, BPU and MLC each in isolation, where one unit is managed while the others are gated on throughout execution.

Unit Activity. Figures 9 and 10 show the percentage of cycles POWERCHOP is able to power gate each of the three units for the mobile processor design and the server processor design, respectively. Overall, POWERCHOP gates off units a significant fraction of execution. The VPU is gated off around 90% of the time for almost all SPEC-INT benchmarks on the server processor and for all of the applications on the mobile processor. Surprisingly, the VPU is also shut off for significant fractions of some SPEC-FP and PARSEC applications, discussed in further detail in Section V-E. The VPU is gated off above 90% of the time for `namd` and `dedup`, and 20% of the time for `soplex` and `sphinx`.

For the MLC, POWERCHOP also way-gates the cache a significant amount of the time. POWERCHOP configures the MLC as 1-way for over 40% of the cycles on several SPEC and PARSEC applications such as `gems`, `milc`, `gcc`, `libquantum` and `streamcluster`. For the MobileBench applications on the mobile processor, the MLC is gated off in some fashion an average of nearly 20% of the time across all applications. The large BPU is often found to be necessary by POWERCHOP for the SPEC and PARSEC benchmarks on the server processor, though there are notable exceptions where the BPU is gated for significant fractions of execution for applications such as `lbm` and `hmmer`. However, for the MobileBench applications on the mobile processor, the BPU is gated off a substantial fraction of the time, an average of 40% across applications.

Policy Change Frequency. Figure 11 presents the average number of times the policies enacted by POWERCHOP result in changes to the power gating state of units throughout

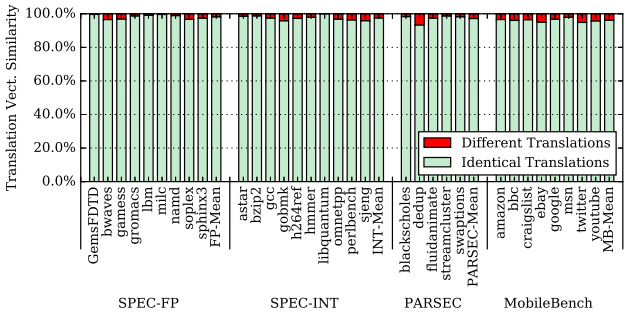


Figure 8. Code similarity between different execution windows characterized by POWERCHOP as having the same phase signature. On average, 97.8% of translations are identical, demonstrating the effectiveness of the phase identification approach

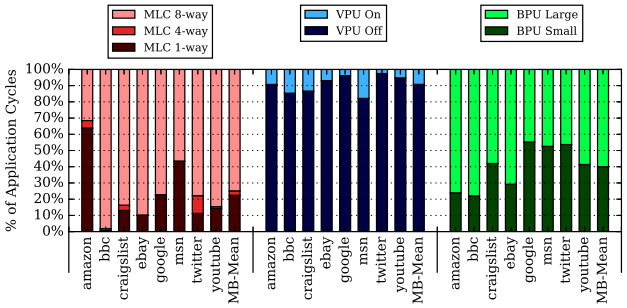


Figure 9. Unit activity on the mobile processor design with POWERCHOP

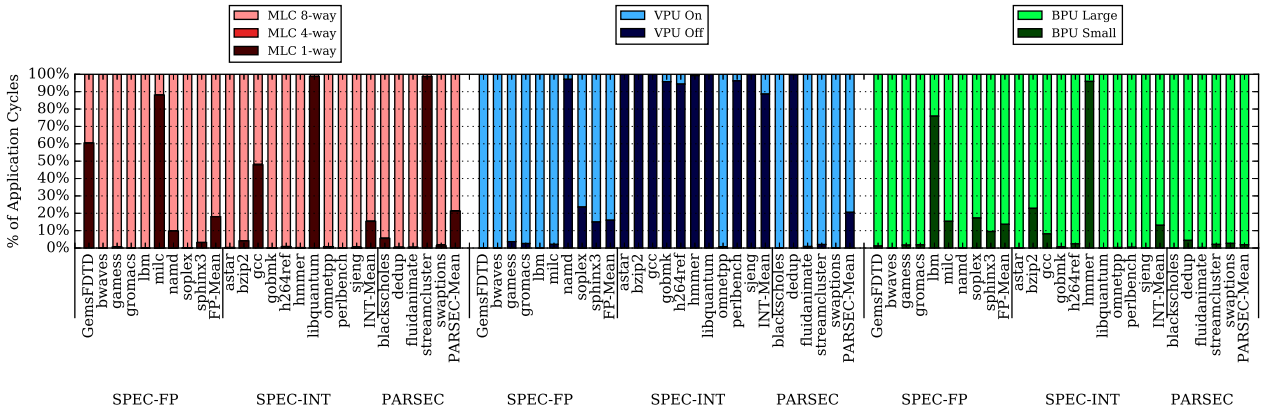


Figure 10. Unit activity on server processor design with POWERCHOP

execution. The higher the number of power gate switches needed, the higher the performance and energy penalty. We find that on average POWERCHOP changes the BPU policy an average of less than 50 times per million cycles, the VPU less than 10 times per million cycles, and the MLC less than 5 times per million cycles. Note that POWERCHOP gates off units for a high percentage of time while also maintaining a reasonably low number of unit state changes, which helps minimize any resultant performance and power overheads. The quantitative impact of POWERCHOP on performance and power is examined in further detail later next.

D. Multi-unit Management

The following experiments evaluate the impact of POWERCHOP when applied simultaneously to all three units.

Performance. Figure 12 presents the performance of a full-powered configuration (MLC, VPU and BPU are at their highest-power states for the entire execution), a POWERCHOP-managed configuration (POWERCHOP chooses when to power gate all three units) and a minimally-powered configuration (MLC, VPU and BPU are in their lowest-power states for the entire execution). As shown in the figure, the minimally-powered configuration loses substantial performance compared to a fully-powered core, around 84% on average. On the other hand, POWERCHOP loses very little performance when compared to the full-powered core, averaging only 2.2% across all applications. By exploiting opportunities to gate units when they are not performance-critical, POWERCHOP achieves nearly all of the performance of a core that is always fully-powered while significantly improving the power consumption.

Power and Energy. Figure 13 presents the total core power reduction and energy reduction when POWERCHOP manages the MLC, VPU and BPU simultaneously. Overall, POWERCHOP reduces total core power consumption – including both leakage and dynamic power – by 10% for SPEC-INT, 6% for SPEC-FP, 8% for PARSEC and 19% for MobileBench. POWERCHOP achieves significant total power reduction for a large set of applications; for 13 out of 29 applications studied, POWERCHOP achieves above 10% core power reduction. For benchmarks such as `lbm`, `milc` and `amazon`, it achieves larger reductions of up to 40% of total core power consumption.

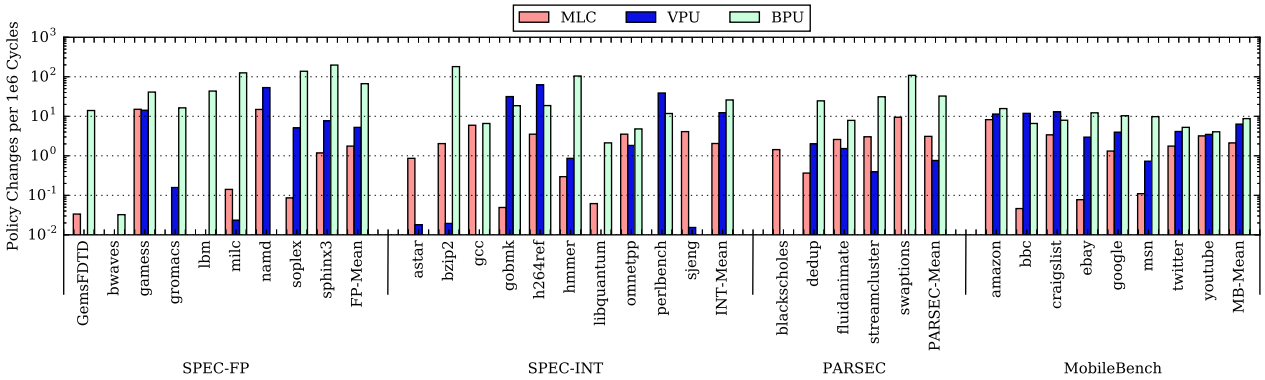


Figure 11. Frequency of unit state changes resulting from POWERCHOP enacting power gating policies

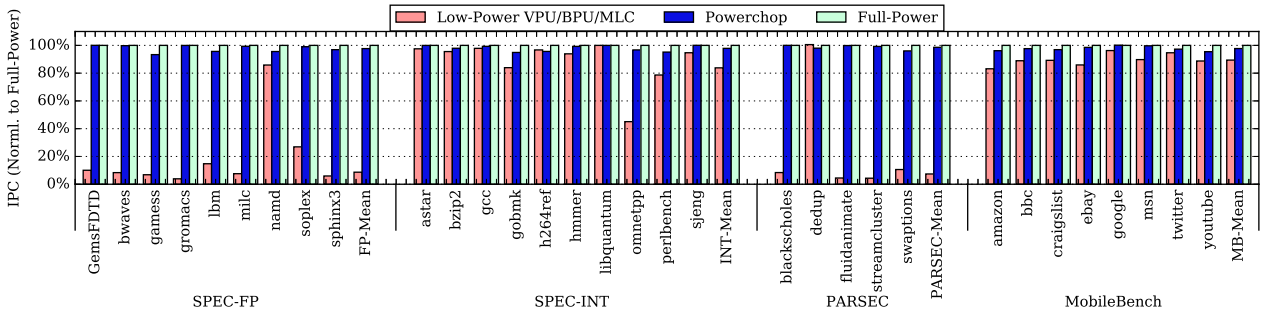


Figure 12. Application performance with POWERCHOP compared a full-power approach that keeps the VPU, BPU and MLC gated on throughout execution and a low-power approach that keeps the units in their lowest-power state throughout execution

Energy reductions are slightly smaller than power reduction since POWERCHOP allows for minor performance degradations (below 2.2% on average). POWERCHOP achieves up to a 37% reduction on total energy. For 10 out of 29 applications, it achieves more than 10% of total energy reduction. On average, the energy reduction is 9% across all 29 applications in our study.

Leakage Power. Leakage power is an important part of power consumption, and is of particular importance as process technologies shrink and leakage maintains or increases its share of the power budget. Figure 14 shows the reduction in core leakage power when POWERCHOP manages power gating for the VPU, BPU and MLC. POWERCHOP achieves significant leakage power reductions for most the applications. For 10 out of 29 applications, POWERCHOP achieves around a 20% reduction in leakage power; and for an additional 12 out of the 29 applications, POWERCHOP achieves significantly higher than 20% (up to 52%) leakage reductions. On average, POWERCHOP achieves a 10% leakage power reduction for SPEC-FP and a 12% reduction for PARSEC. Its effectiveness is higher for SPEC-INT and MobileBench, averaging a 23% reduction for SPEC-INT and 32% for MobileBench. Moreover, these power reductions come at a modest performance degradation of just 2.2%.

E. Comparison to HW-Only Timeouts

An approach that has been proposed for both cores and logical units [2], [11]–[15] is to power gate after a period of unit idleness. For unit-level power gating, prior work focuses on functional units like the VPU, which is the most promising unit for timeout-based approaches among the three units in our study. Here we evaluate POWERCHOP's VPU gating against a time-out based approach.

Figure 15 illustrates the prevalence vector operations in every 1000-instruction execution shard within running applications. As shown in the figure, for several applications certain phases of execution only contain a small number of vector operations ($0 < V \leq 4$). Because POWERCHOP leverages the binary translation subsystem to avoid vector operations when those operations are infrequent and the performance-criticality is low, it can exploit these opportunities to create larger execution windows that make power gating the VPU worthwhile.

The key factor in a timeout approach is choosing the *timeout period* – the number of idle cycles after which the unit is power gated. To carefully devise a well-performing timeout approach, we ran a spectrum of timeout periods from 100 to 100K cycles. From among these we chose a 20K cycle timeout, as this is the timeout period that saves the most power while incurring less than 5% worst-case

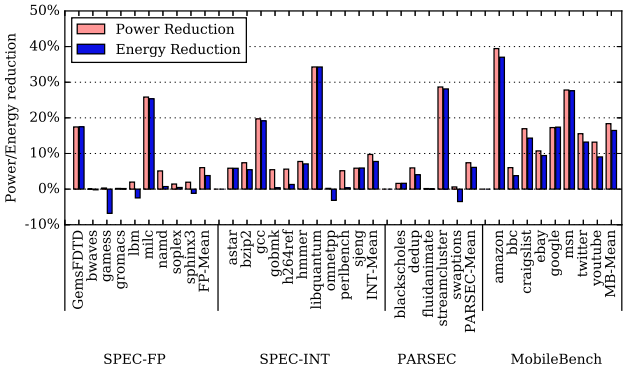


Figure 13. Total core power and energy reduction with PowerChop

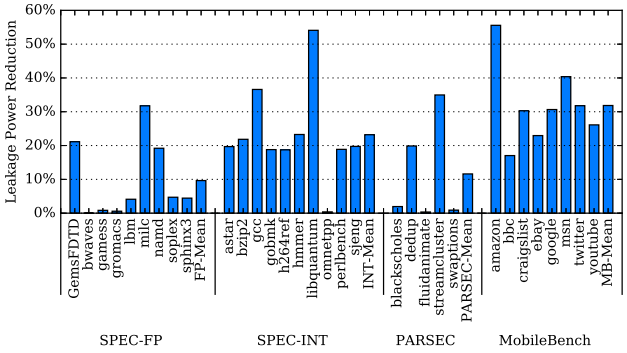


Figure 14. Leakage power reduction with PowerChop

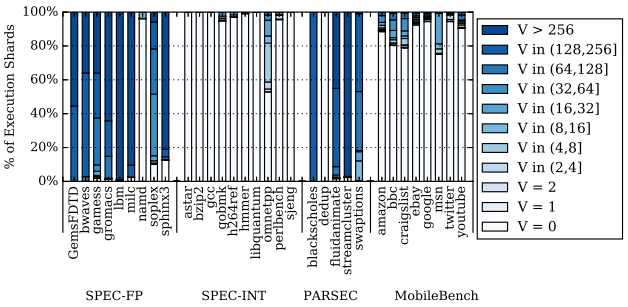


Figure 15. Vector operation prevalence (V) among execution shards

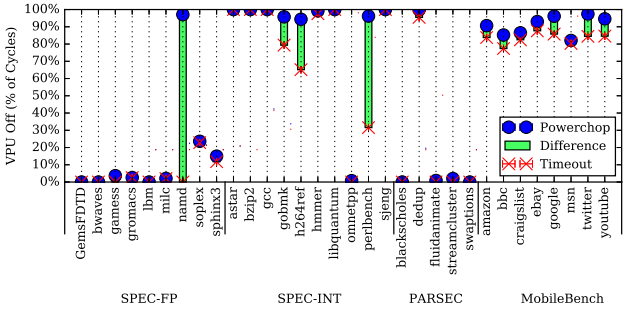


Figure 16. VPU gating activity for PowerChop vs. timeout

application performance degradation (the comparable level of performance degradation to PowerChop).

Figure 16 shows the percentage of cycles the VPU is kept idle during PowerChop-managed runs in comparison to timeout. PowerChop gates the VPU off at least as much as the timeout approach across all applications. In a few cases, including `namd`, `perlbench` and `h264`, PowerChop shows immense benefits over timeout. For example, PowerChop keeps the VPU gated off during nearly all of `namd`'s execution while timeout keeps the VPU gated on for the nearly the entirety of execution. This occurs because `namd` has occasional phases of small number of vector operations. These small numbers of VPU operations are nearly uniformly distributed throughout execution, which prevents the timeout approach from gating off the unit. PowerChop, on the other hand, is able to quickly identify that the VPU is not performance-critical during these phases and gate the unit off throughout most of execution.

Timeout based approaches are ill-suited for the MLC and BPU due to the highly active nature of those units. The difficulty in applying timeouts to these units is that the BPU and MLC are unlikely to be inactive for long periods, regardless of whether they are providing a substantial performance benefit to the application, and thus unit inactivity is of limited use for triggering a timeout. Prior work has pointed out that branches account for 1 out of every 20 instructions executed in SPEC, and for a range of cache configurations, MLC accesses occur 1 out of every 100 to 200 instructions executed [5]. Additionally, for units like the VPU, the decision mechanism for gating the unit back on is clear – gate it back on when the unit is needed (e.g., the VPU is needed to execute a vector operation). It is unclear how a timeout approach can easily derive such a decision mechanism for highly active units such as the BPU or MLC.

VI. Related Work

This work most closely ties into three research areas: power gating, hybrid processor architecture design and phase analysis. Prior work has shown that core power gating can be controlled at very coarse granularity by software or the operating system [11]. Conversely, unit-level (or smaller) power gating [12] has been shown to be possible using hardware-only timeout approaches [2] for a certain class of units that are 1) subject to prolonged periods of inactivity and 2) stateless. PowerChop overcomes the first limitation by leveraging *unit criticality* rather than unit inactivity to make gating decisions and the second limitation by enacting gating decisions at a coarser granularity, allowing it to amortize the larger gate switching overheads that may accompany saving and restoring architectural state in the gated unit.

Others have proposed techniques to reduce cache energy when cache ways or lines are not effectively utilized [26], [27]. Flautner et. al. [27] propose the drowsy cache, a per-line leakage power reduction technique that puts cache lines

into a low-voltage drowsy state when they are idle. This approach differs from ours in two key respects: (1) the drowsy cache requires that state be preserved, which limits how low the voltage can be cut and how much leakage power can be saved [28] and (2) drowsy cache will leave cache lines active while they are highly active, being accessed frequently but doing little to help the performance of the application (e.g., when the working set size is larger than the cache).

A key enabling technology for POWERCHOP is the hybrid processor architecture, and the binary translation mechanism [8], [29]–[31] it encompasses. Our work is complementary to recent work that focus on some of the advantages offered by the unique capabilities of hybrid architecture designs to improve performance [9], [10], [32]–[35].

A number of prior works have shown that code-based phase classification has numerous advantages [36] and that using basic block vectors (BBVs) and its variants are effective [25], [37]. There are several differences between our phase recognition approach and BBV. Firstly, POWERCHOP takes advantage of the binary translation system and uses translations (traces) from the BT instead of basic blocks as the basic unit for phase recognition. Secondly, instead of keeping track of a full record of frequencies of all basic blocks, POWERCHOP identifies only the hottest translations. Since translations are traces that already embed path information, keeping track of only a few hot translations (4 in this work) is sufficient for us to capture phases accurately. More importantly, it also facilitates a low-overhead hardware implementation. Prior work, for example, requires full comparisons between vectors of basic block frequencies for phase matching and identification [38].

Dhodapkar et.al [39] propose a phase transition detection technique based on working set signatures. Their technique identifies phases that are multiple orders of magnitude longer than those detected by POWERCHOP. The finer-grain stable phases identified by POWERCHOP allows it to exploit more potential gating opportunities and provides faster reactions (up to a 3 orders of magnitude difference). This is particularly important due to the large performance cost associated with wrong gating-off decisions.

VII. CONCLUSION

This work introduces POWERCHOP, a novel approach to unit-level power gating for HW/SW co-designed hybrid processor architectures. Our approach identifies *unit criticality* characteristics and uses those characteristics to enact power gating decisions on non-critical units. We demonstrate the effectiveness of POWERCHOP using three power hungry units: the vector processing unit (VPU), branch prediction unit (BPU), and middle-level cache (MLC). POWERCHOP dramatically reduces power consumption while retaining application performance. When applying POWERCHOP to the VPU, BPU, and MLC simultaneously we observe leakage power reductions of up to 33% (9% on average) on a server processor across a wide range of workloads and 40% (19% on average) on a mobile processor with an average performance degradation of just 2%.

ACKNOWLEDGEMENTS

We would like to thank our anonymous reviewers for their comments. This work was sponsored by National Science Foundation grants CCF-1302682 and CCF-1553485.

REFERENCES

[1] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2000.

[2] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[3] A. Youssef, M. Anis, and M. Elmasry, "Dynamic Standby Prediction for Leakage Tolerant Microprocessor Functional Units," in *International Symposium on Microarchitecture (MICRO)*, 2006.

[4] S. Roy, N. Ranganathan, and S. Katkoori, "A Framework for Power-Gating Functional Units in Embedded Microprocessors," *Transactions on Very Large Scale Integration Systems (T-VLSI)*, 2009.

[5] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in *High Performance Computer Architecture (HPCA)*, 2013.

[6] "NVIDIA's first CPU is a winner," http://www.linleygroup.com/mpr/article.php?id=11262, [Online; accessed 14-April-2016].

[7] "Project denver-based soc due in 2015," http://techreport.com/news/24530/, [Online; accessed 14-April-2016].

[8] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson, "The transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges," in *Code Generation and Optimization (CGO)*, 2003.

[9] N. Neelakantam, D. R. Ditzel, and C. Zilles, "A real system evaluation of hardware atomicity for software speculation," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2010.

[10] J. Mars and N. Kumar, "Blockchop: Dynamic squash elimination for hybrid processor architecture," in *International Symposium on Computer Architecture (ISCA)*, 2012.

[11] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *Computer Architecture Letters (CAL)*, 2009.

[12] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "Guarded power gating in a multi-core setting," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[13] K. Jeong, A. B. Kahng, S. Kang, T. S. Rosing, and R. Strong, "MAPG: Memory access power gating," in *Design, Automation and Test in Europe (DATE)*, 2012.

[14] A. B. Kahng, S. Kang, T. Rosing, and R. Strong, "TAP: token-based adaptive power gating," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2012.

[15] S. Dropsho, V. Kursun, D. H. Albonesi, S. Dwarkadas, and E. G. Friedman, "Managing static leakage energy in microprocessor functional units," in *International Symposium on Microarchitecture (MICRO)*, 2002.

[16] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model." in *Technical Report 2001/2, Compaq Computer Corporation*, 2001.

[17] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. IEEE, 2009, pp. 469–480.

[18] C. Long and L. He, "Distributed sleep transistor network for power reduction," *Transactions on Very Large Scale Integration Systems (T-VLSI)*, 2004.

[19] H. Singh, K. Agarwal, D. Sylvester, and K. J. Nowka, "Enhanced leakage reduction techniques using intermediate strength power gating," *Transactions on Very Large Scale Integration Systems (T-VLSI)*, 2007.

[20] H. Jiang, M. Marek-Sadowska, and S. R. Nassif, "Benefits and costs of power-gating technique," in *International Conference on Computer Design (ICCD)*, 2005.

[21] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Computer Architecture News*, 2006.

[22] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 2008, pp. 72–81.

[23] D. Pandiyan, S.-Y. Lee, and C.-J. Wu, "Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite–mobilebench."

[24] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, 2011.

[25] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Architectural Support for Programming Languages and Pperating Systems*, 2002.

[26] D. H. Albonesi, "Selective cache ways: On-demand cache resource allocation," in *International Symposium on Microarchitecture (MICRO)*, 1999.

[27] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *International Symposium on Computer Architecture (ISCA)*, 2002.

[28] K. Agarwal, K. Nowka, H. Deogun, and D. Sylvester, "Power gating with multiple sleep modes," in *Proceedings of the 7th International Symposium on Quality Electronic Design*. IEEE Computer Society, 2006, pp. 633–637.

[29] V. Bala, E. Duesterwald, and S. Banerjia, "Dynamo: a transparent dynamic optimization system," in *ACM SIGPLAN Notices*, vol. 35, no. 5. ACM, 2000, pp. 1–12.

[30] D. J. Hiniker, K. Hazelwood, and M. D. Smith, "Improving region selection in dynamic optimization systems," in *38th Annual International Symposium on Microarchitecture*, Barcelona, Spain, November 2005, pp. 141–154.

[31] K. Hazelwood and M. D. Smith, "Generational cache management of code traces in dynamic optimization systems," in *36th Annual International Symposium on Microarchitecture*, San Diego, CA, December 2003, pp. 169–179.

[32] D. S. McFarlin, C. Tucker, and C. Zilles, "Discerning the dominant out-of-order performance advantage: is it speculation or dynamism?" in *Architectural Support for Programming Languages and Operating Systems*, 2013.

[33] D. S. McFarlin and C. Zilles, "Branch vanguard: decomposing branch functionality into prediction and resolution instructions," in *International Symposium on Computer Architecture (ISCA)*, 2015.

[34] M. Lupon, E. Gibert, G. Magklis, S. Samudrala, R. Martínez, K. Stavrou, and D. R. Ditzel, "Speculative hardware/software co-designed floating-point multiply-add fusion," *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014.

[35] D. S. McFarlin and C. Zilles, "Bungee jumps: accelerating indirect branches through hw/sw co-design," in *International Symposium on Microarchitecture (MICRO)*, 2015.

[36] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," in *International Symposium on Computer Architecture (ISCA)*, 2003.

[37] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *Parallel Architectures and Compilation Techniques (PACT)*, 2001.

[38] S. Padmanabha, A. Lukefahr, R. Das, and S. Mahlke, "Trace based phase prediction for tightly-coupled heterogeneous cores," in *International Symposium on Microarchitecture (MICRO)*, 2013.

[39] A. S. Dhodapkar and J. E. Smith, "Managing multi-configuration hardware via dynamic working set analysis," in *International Symposium on Computer Architecture (ISCA)*, 2002.