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

Michael A. Laurenzano, Yunqi Zhang, Jiang Chen, Lingjia Tang, Jason Mars

International Symposium on Computer Architecture (ISCA)
June 20, 2016
Unit-level Power Gating
Unit-level Power Gating

- Core-level power gating based on usage
Unit-level Power Gating

- Core-level power gating based on usage
- On-core units consume a significant portion of core power budget

![Diagram of MLC, BPU, and VPU units]
Unit-level Power Gating

• Core-level power gating based on usage

• On-core units consume a significant portion of core power budget

• Cannot apply usage-based power gating to highly active units
Unit-level Power Gating

- Core-level power gating based on usage
- On-core units consume a significant portion of core power budget
- Cannot apply usage-based power gating to highly active units
Unit Criticality Across Phases

- Example — VPU criticality in gobmk
Unit Criticality Across Phases

- Example — VPU criticality in gobmk

![Graph showing vector operation intensity and VPU non-critical periods](Image)
Unit Criticality Across Phases

- Example — VPU criticality in gobmk

![Graph showing vector operation intensity over 200 million instructions](image)

- Vector Op Intensity
- VPU Non-critical

![Graph showing BPU and MLC activity](image)

- BPU
- MLC

Understanding and taking advantage of these opportunities as a dynamic property of the running application requires monitoring and analyzing application execution, for spilling/restoring or losing/reconstituting that state. Power gating too low but non-zero number of vector operations occur during various phases of execution, saving power without sacrificing performance.

During those phases, the large BPU (Small BPU) and a larger tournament local/global predictor (Large BPU). Unsurprisingly, using the large BPU instead of the MobileBench processor using a small local branch predictor (Small BPU).
Unit Criticality Across Phases

• Example — VPU criticality in gobmk

Unit criticality varies across applications and phases
Addressing Unit Non-criticality
Addressing Unit Non-criticality

- **Goal** — power gate units when they are non-critical to performant execution
Addressing Unit Non-criticality

• **Goal** — power gate units when they are non-critical to performant execution

• **Challenges**
Addressing Unit Non-criticality

- **Goal** — power gate units when they are non-critical to performant execution

- **Challenges**
  - **High activity** — continuously active units may not always be useful (e.g., middle level cache)
Addressing Unit Non-criticality

• **Goal** — power gate units when they are non-critical to performant execution

• **Challenges**
  
  • **High activity** — continuously active units may not always be useful (e.g., middle level cache)
  
  • **Unit state** — can be expensive to lose state (e.g., branch history)
Addressing Unit Non-criticality

- **Goal** — power gate units when they are non-critical to performant execution

- **Challenges**
  - **High activity** — continuously active units may not always be useful (e.g., middle level cache)
  - **Unit state** — can be expensive to lose state (e.g., branch history)
  - **Application behavior** — understanding degree and duration of non-criticality requires detailed application understanding
Addressing Unit Non-criticality

- **Goal** — power gate units when they are non-critical to performant execution

- **Challenges**
  - **High activity** — continuously active units may not always be useful (e.g., middle level cache)
  - **Unit state** — can be expensive to lose state (e.g., branch history)
  - **Application behavior** — understanding degree and duration of non-criticality requires detailed application understanding

**Hybrid architecture enables a solution to these challenges**
Hybrid Architecture Background
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
- Commercial examples
  - Transmeta Crusoe, Efficeon
  - NVIDIA project Denver, Parker

Diagram:
- Application Software
  - BT Software
    - Interpreter
    - Nucleus
    - Translator/Optimizer
  - Region Cache
    - Hardware
    - Hot code translations
Hybrid Architecture Background

- Software binary translation (BT)
  - Translations are the basic unit of execution
- Commercial examples
  - Transmeta Crusoe, Efficeon
  - NVIDIA project Denver, Parker

**BT can easily absorb the complexity of monitoring execution and determining non-criticality**
Hybrid Architecture Background

- Software binary translation (BT)
  - BT manages all software execution
  - Translations are the basic unit of execution
  - Commercial examples: Transmeta Crusoe, Efficeon
    - NVIDIA project Denver, Parker

BT can easily absorb the complexity of monitoring execution and determining non-criticality

BT can modify the instruction stream to create additional opportunities to exploit non-criticality
PowerChop Approach
PowerChop Approach

• Phase signatures collected continuously
PowerChop Approach

- Phase signatures collected continuously
- Profile new phases to characterize unit criticality and define power management settings
PowerChop Approach

• Phase signatures collected continuously

• Profile new phases to characterize unit criticality and define power management settings

• Apply power settings to recurring phases throughout execution
PowerChop Approach

- Phase signatures collected continuously
- Profile new phases to characterize unit criticality and define power management settings
- Apply power settings to recurring phases throughout execution
PowerChop Approach

• Phase signatures collected continuously

• Profile new phases to characterize unit criticality and define power management settings

• Apply power settings to recurring phases throughout execution
PowerChop Approach

- Phase signatures collected continuously
- Profile new phases to characterize unit criticality and define power management settings
- Apply power settings to recurring phases throughout execution
PowerChop Approach

- Phase signatures collected continuously
- Profile new phases to characterize unit criticality and define power management settings
- Apply power settings to recurring phases throughout execution
PowerChop Approach

• Phase signatures collected continuously

• Profile new phases to characterize unit criticality and define power management settings

• Apply power settings to recurring phases throughout execution
Phase Identification
Phase Identification

- Dynamic execution broken into windows of 1000 translations
Phase Identification

• Dynamic execution broken into windows of 1000 translations
Phase Identification

- Dynamic execution broken into windows of 1000 translations
- Hot translation buffer (HTB) — new unit to monitor execution and create phase signatures
Phase Signatures
Phase Signatures

- HTB counts translation executions within each window
Phase Signatures

- HTB counts translation executions within each window

<table>
<thead>
<tr>
<th>t3</th>
<th>t5</th>
<th>t8</th>
<th>t6</th>
<th>t4</th>
<th>t7</th>
<th>t5</th>
<th>...</th>
<th>t12</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>50</td>
<td>3</td>
<td>93</td>
<td>25</td>
<td>187</td>
<td>43</td>
<td>...</td>
<td>19</td>
</tr>
</tbody>
</table>

Translation count
Phase Signatures

- HTB counts translation executions within each window

<table>
<thead>
<tr>
<th>Translation</th>
<th>Execution Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>t3</td>
<td>49</td>
</tr>
<tr>
<td>t5</td>
<td>50</td>
</tr>
<tr>
<td>t8</td>
<td>3</td>
</tr>
<tr>
<td>t6</td>
<td>93</td>
</tr>
<tr>
<td>t4</td>
<td>25</td>
</tr>
<tr>
<td>t7</td>
<td>187</td>
</tr>
<tr>
<td>t5</td>
<td>43</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>t12</td>
<td>19</td>
</tr>
</tbody>
</table>

- Phase signature defined by hottest 4 translations in window
  - Example phase signature: \(<t3, t5, t6, t7>\)
Phase Signatures

- HTB counts translation executions within each window

<table>
<thead>
<tr>
<th>t3</th>
<th>t5</th>
<th>t8</th>
<th>t6</th>
<th>t4</th>
<th>t7</th>
<th>t5</th>
<th>...</th>
<th>t12</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>50</td>
<td>3</td>
<td>93</td>
<td>25</td>
<td>187</td>
<td>43</td>
<td>...</td>
<td>19</td>
</tr>
</tbody>
</table>

- Phase signature defined by hottest 4 translations in window
- Example phase signature: <t3, t5, t6, t7>
Phase Signatures

- HTB counts translation executions within each window

<table>
<thead>
<tr>
<th>t3</th>
<th>t5</th>
<th>t8</th>
<th>t6</th>
<th>t4</th>
<th>t7</th>
<th>t5</th>
<th>...</th>
<th>t12</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>50</td>
<td>3</td>
<td>93</td>
<td>25</td>
<td>187</td>
<td>43</td>
<td>...</td>
<td>19</td>
</tr>
</tbody>
</table>

- Phase signature defined by hottest 4 translations in window

- Example phase signature: \(<t3, t5, t6, t7>\)

Common translations between windows with same signature — 97%
Characterizing Unit Criticality
Characterizing Unit Criticality

- Criticality Decision Engine in software layer
Characterizing Unit Criticality

- **Criticality Decision Engine** in software layer
- New phase triggers profiling
Characterizing Unit Criticality

- **Criticality Decision Engine** in software layer
- New phase triggers profiling
- Characterize unit criticality during profiling
  - **VPU Criticality** — (\# SIMD) / (\# Insn)
  - **BPU Criticality** — tournament mispred. — local mispred.
  - **MLC Criticality** — (\# L2 hits) / (\# Insn)
Setting Power Management Policies
Setting Power Management Policies

- Use thresholds of unit criticality
Setting Power Management Policies

- Use thresholds of unit criticality
- Unit-specific settings
  - **VPU** — on vs. off
  - **BPU** — local vs. tournament
  - **MLC** — 1 vs. 4 vs. 8 ways
Setting Power Management Policies

- Use thresholds of unit criticality
- Unit-specific settings
  - **VPU** — on vs. off
  - **BPU** — local vs. tournament
  - **MLC** — 1 vs. 4 vs. 8 ways
- **Policy Vector Table (PVT)** — new unit to store and enact power management policies
Evaluation
Experimental Setup
Experimental Setup

- Simulation infrastructure
  - Gem5 for detailed performance simulation
  - McPAT (32nm) for power modeling
  - In-house binary translation implementation
Experimental Setup

- Simulation infrastructure
  - Gem5 for detailed performance simulation
  - McPAT (32nm) for power modeling
  - In-house binary translation implementation
- Platforms
  - Mobile (Cortex-A9) — MobileBench
  - Server (Nehalem) — SPEC CPU, PARSEC
Exploiting Non-criticality — Mobile

• % of cycles each spent power gated in PowerChop

![Bar chart showing % of application cycles with different power states for various applications]
Exploiting Non-criticality — Mobile

- % of cycles each spent power gated in PowerChop
Exploiting Non-criticality — Mobile

- % of cycles each spent power gated in PowerChop
Exploiting Non-criticality — Mobile

- % of cycles each spent power gated in PowerChop

![Bar chart showing percentage of cycles spent in different power states for various applications.]

- 25% of cycles spent power gated in MLC 8-way
- 91% of cycles spent in VPU On
- 40% of cycles spent in BPU Large
Exploiting Non-criticality — Server

- % of cycles each spent power gated in PowerChop
Exploiting Non-criticality — Server

- % of cycles each spent power gated in PowerChop

17% of cycles
Exploiting Non-criticality — Server

- % of cycles each spent power gated in PowerChop

17% of cycles

47% of cycles
Exploiting Non-criticality – Server

• % of cycles each spent power gated in PowerChop

17% of cycles
47% of cycles
11% of cycles
Core Power Reduction

- PowerChop hybrid vs. conventional hybrid
Core Power Reduction

- PowerChop hybrid vs. conventional hybrid

9% power reduction
Core Power Reduction

• PowerChop hybrid vs. conventional hybrid

9% power reduction
19%

SPEC-FP  server  SPEC-INT  PARSEC  mobile  MobileBench
Core Power Reduction

- PowerChop hybrid vs. conventional hybrid

9% power reduction
2% performance degradation
19%
Core Power Reduction

* PowerChop hybrid vs. conventional hybrid

- 9% power reduction
- 2% performance degradation
- 2 small units (< 0.02 mm²)

SPEC-FP  server  SPEC-INT  PARSEC  mobile  MobileBench
Conclusion
Conclusion

- Varying unit criticality presents a significant power reduction opportunity

- PowerChop for hybrid architectures
  - Software absorbs complexity of characterizing unit criticality and power management policies
  - Small hardware units (HTB, PVT) assist with phase attribution and power gating

- 19% power reduction on mobile, 9% on server
Backup Slides
Unit-level Power Gating

- On-core units consume a significant portion of power budget

Core-level power gating techniques have not translated to architectural units
Unit Non-criticality Across Phases

• Example — tournament vs. local branch predictor
Unit Non-criticality Across Phases

- Example — tournament vs. local branch predictor

![Graph showing IPC normalized vs. # of instructions executed for MobileBench msn with Small BPU, Large BPU Non-critical, and Large BPU.]
Unit Non-criticality Across Phases

- Example — tournament vs. local branch predictor
Unit Non-criticality Across Phases

- Example — tournament vs. local branch predictor

**Unit criticality varies across applications and phases**

- As shown in Figure 1, the intensity of vector operations and branch predictors over 13 million instructions of SPEC CPU2006.

- Modern branch predictors often leverage multiple branch predictors over 13 million instructions.

- Unsurprisingly, using the large BPU instead of the BPU presents the IPC over time for a web browser running on a MobileBench msn.

- Understanding and taking advantage of these opportunities by power gating may be useful for accurately predict branch outcomes using a tournament branch predictor.

- Neural, etc., predicting branch outcomes using a tournament branch predictor has low criticality, suggesting that there may be an opportunity to power gate parts of the BPU.

- Consequently, using the large BPU could be justified, the VPU could be put to use occasionally by application code, but be used during certain periods units can be critical for application execution. Moreover, consider Figures 2 and 3, where the criticality of the VPU varies over time. When these periods 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.
PowerChop Overview

• **Goal** — power gate units when they are non-critical for performance

• **Overview of approach**
  
  • Phase signatures collected continuously
  
  • Profile new phases to characterize unit criticality and define power management settings
  
  • Apply power settings to recurring phases throughout execution
PowerChop Overview

- **Goal** — power gate units when they are non-critical for performance

- Overview of approach
  - Phase signatures collected continuously
  - Profile new phases to characterize unit criticality and define power management settings
  - Apply power settings to recurring phases throughout execution
Unique Capabilities of Hybrid
Unique Capabilities of Hybrid

- BT layer can absorb the complexity needed to monitor application execution
Unique Capabilities of Hybrid

• BT layer can absorb the complexity needed to monitor application execution

• Tailor instruction stream to avoid non-critical units
Unique Capabilities of Hybrid

• BT layer can absorb the complexity needed to monitor application execution

• Tailor instruction stream to avoid non-critical units
  • In this work — avoid infrequent vector instructions
Applying Learned Policies

- Policies re-used throughout execution as phases recur
- **Policy vector table** — new unit to store and enact power management polices

<table>
<thead>
<tr>
<th>Phase Signature (128b)</th>
<th>Gating Policy (4b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t3, t4, t5, t12</td>
<td>V=1, B=0, M=01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>t3, t6, t7, t10</td>
<td>V=0, B=0, M=11</td>
</tr>
</tbody>
</table>

16 Entries
Phase Analysis Quality

- Comparison of 1000-translation windows assigned the same phase signature
Phase Analysis Quality

- Comparison of 1000-translation windows assigned the same phase signature

**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 execution. The higher the number of power gate switches needed, the higher the performance and energy penalty.

**Figure 10.** Unit activity on server processor design with POWERCHOP execution. 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.
Phase Analysis Quality

- Comparison of 1000-translation windows assigned the same phase signature

![Diagram showing translation 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.](image)

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 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.
Phase Analysis Quality

- Comparison of 1000-translation windows assigned the same phase signature

**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 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.
The Unit Criticality Problem

App. execution
The Unit Criticality Problem

Unit Criticality

App. execution
The Unit Criticality Problem

App. execution

Unit Criticality

Today’s
Power Dissipation
The Unit Criticality Problem

Unit Criticality

Today’s Power Dissipation

Ideal Power Dissipation

App. execution