Architecture Support for Disciplined Approximate Programming

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, Doug Burger
University of Washington, Microsoft Research

Presented by: Lucy Jiang, Cristina Garbacea
Outline

● Background

● Truffle: A Dual-Voltage Microarchitecture for Disciplined Approximation

● Evaluation

● Conclusion
Background

Goal

Reducing energy usage + Increasing energy efficiency

General Solution

Trading off quality of service to energy efficiency using approximate computation

Proposed Solution

Approximation-aware microarchitecture

Disciplined approximate programming models

Help programmers identify soft slice
Architecture Proposal

Allow a compiler to convey what can be approximated

ISA Extension

Processors that implement the ISA

To

Microarchitectural extensions

Rely on

A dual voltage supply for SRAM arrays and logic

HIGH Vdd

LOW Vdd
ISA Extensions for Approximation

- ISA supports both operations.
  - Precise: correct output guaranteed
  - Approximate: correct output expected

- Programming language assumption
  - Approximate doesn’t affect precise

- Precision of registers
  - Depending on the last instruction
  - Every instruction has an extra bit per operand to specify precision
Truffle: Dual-Voltage SRAM Structure
Truffle: Voltage Level Shifting and Multiplexing
Truffle Execution

Evaluating Truffle

Goals:

● Determine the energy savings brought by disciplined approximation

● Characterize where the energy goes

● Understand the QoS implications for software
Evaluation Setup

- Truffle is modeled at 65 nm technology node in the context of both in-order and out-of-order designs

- Evaluate each benchmark for two criteria: energy savings and sensitivity to error

- Statistics collected: variable, field, array accesses, basic blocks, arithmetic and logical operators

- Inject errors in the execution and measure the consequent degradation in output quality

Table 2. Microarchitectural parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>OOO Truffle</th>
<th>In-order Truffle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch/Decode Width</td>
<td>4/4</td>
<td>2/2</td>
</tr>
<tr>
<td>Issue/Commit Width</td>
<td>6/4</td>
<td>-/1</td>
</tr>
<tr>
<td>INT ALUs/FPUs</td>
<td>4/2</td>
<td>1/1</td>
</tr>
<tr>
<td>INT Mul/Div Units</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Approximate INT ALUs/FPUs</td>
<td>4/2</td>
<td>1/1</td>
</tr>
<tr>
<td>Approximate INT Mul/Div Units</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>INT/FP Issue Window Size</td>
<td>20/15</td>
<td>-</td>
</tr>
<tr>
<td>ROB Entries</td>
<td>80</td>
<td>-</td>
</tr>
<tr>
<td>INT/FP Architectural Registers</td>
<td>32/32</td>
<td>32/32</td>
</tr>
<tr>
<td>INT/FP Physical Registers</td>
<td>80/72</td>
<td>-</td>
</tr>
<tr>
<td>Load/Store Queue Size</td>
<td>32/32</td>
<td>-</td>
</tr>
<tr>
<td>L1 Cache Size</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>L1 Cache Size</td>
<td>64 Kbyte</td>
<td>16 Kbyte</td>
</tr>
<tr>
<td>Line Width/Associativity</td>
<td>32/2</td>
<td>16/4</td>
</tr>
<tr>
<td>DTLB</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>D Cache Size</td>
<td>64 Kbyte</td>
<td>32 Kbyte</td>
</tr>
<tr>
<td>Line Width/Associativity</td>
<td>16/2</td>
<td>16/4</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>Tournament</td>
<td>Tournament</td>
</tr>
</tbody>
</table>
Evaluation Setup

Benchmarks:

- 9 benchmark programs written in EnerJ:
  hand-annotated programs with approximate type qualifiers that distinguish their approximate parts
  (SciMark2, ZXing, jMonkeyEngine, ImageJ, 3D raytracer)

- How to quantify the loss in output quality caused by hardware approximation?
  Define an application specific quality-of-service metric to quantify the loss in output quality
  - RMSE
  - Proportion of incorrect intersection decisions
  - Proportion of unsuccessful decodings of a sample QR code image
Energy Savings

- $V_{ddH} = 1.5$ V
- $V_{ddL} : 50\%, 62.5\%, 75\%, 87.5\%$ of $V_{ddH}$
- Frequency = 1666 MHz

- in-order configuration: all voltage levels lead to energy savings (up to 43%)
- out-of-order configuration: energy savings when $V_{ddL}$ is less than 75\% of $V_{ddH}$

The impact of Truffle in in-order cores is much higher!

**Figure 5.** Percent energy reduction with unchecked OOO and in-order Truffle designs for various $V_{ddL}$ voltages.
Energy Breakdown Per Component

Out-Of-Order Truffle

In-Order Truffle

Imagefill: 42% (OOO) and 47% (In-Order) of energy is consumed in data movement/processing plane
Raytracer: 71% (OOO) and 50% (In-Order)
Energy Savings Potential

- reduce the voltage level of the instruction control plane to 1.2 V

\[ V_{ddL} = 0.75 \text{ V} \]

- the gap in energy savings potential between the OOO and in-order designs is significantly reduced.

Figure 8. Percent energy reduction potential for checked in-order and OOO Truffle designs with \( V_{dd}L = 0.75 \text{ V} \).
Error Propagation from Circuits to Applications

- Inject errors in each of the microarchitectural structures that support approximate behaviour
Conclusion

- **Disciplined approximate programming** is an effective and usable technique for trading off correctness guarantees with energy savings.

- **Dual-voltage microarchitectures** can provide both approximate and precise computation controlled at a fine-grain by the compiler; ISA relies on the compiler to eliminate the need for checking or recovery at run time.

- **Truffle** presents energy savings up to 43% under reasonable assumptions, with benchmarks exhibiting negligible degradation in output quality.
Discussion points

1. For approximation, should we rely more on hardware support or software support? Why?

2. Is Truffle practical enough to be implemented in real world applications?

3. What kind of applications are suitable for disciplined approximate programming using this microarchitecture?