EECS 573 Presentation

Architecture Support for Disciplined Approximate Programming

Du Lyu Xiao Wu

11/16/2015
Outline

- Motivation
- Viability
- Compiler and Language Support
- Hardware Support
- Evaluation
- Conclusion
Motivation

Energy consumption is the first-class concern in today's computer system design.

Typical data center operating cost profile

- Power & cooling cost: 31%
- Servers: 57%
- Networking equipment: 8%
- Other Cost: 4%
Viability

• There many areas in which perfect correctness is not required such as computer vision, machine learning, information retrieval and so on.

• Floating-point computation, for example, is by nature imprecise. Therefore many FP-heavy applications have inherent tolerance to error.
Compiler and Language Support

• Need to decompose programs into precise and approximate parts safely. The author uses HW-SW-co-design method and leaves this concern to compiler and language support.

• Enerj is a language that helps us interleave precise and approximate operations statically.
Enerj

```java
@Approx float[] nums;
⋮
@Approx float total = 0.0f;
for (@Precise int i = 0; 
i < nums.length;
   ++i)
   total += nums[i];
return total / nums.length;
```

Declared as approximate data storage

Approximate operations
Compiler and Language Support

```java
@Approx float[] nums;
...
@Approx float total = 0.0f;
for (@Precise int i = 0;
    i < nums.length;
    ++i)
    total += nums[i];
return total / nums.length;
```

Enerj code
Advantages of Compiler-directed approximation

- Finish safety checks at compile time, no need to do expensive checks at run time.

- Reduce overhead in performance, energy and design complexity.
Hardware support for disciplined approximate programming

- ISA extension for approximation
- Design choices
- Truffle: A dual-voltage microarchitecture
Hardware support for disciplined approximate programming

- ISA extension for approximation
- Design choices
- Truffle: A dual-voltage microarchitecture
Approximation-aware ISA need:

Approximate operations:

\[
\begin{align*}
\text{ALU} & \quad + \quad \div \quad \& \\
& \quad - \quad \times \quad | 
\end{align*}
\]

Instruction is controllable individually since we want approximation to be interleaved with precise computation.

```java
@Approx float[] nums;
⋮
@Approx float total = 0.0f;
for (@Precise int i = 0;
    i < nums.length;
    ++i)
    total += nums[i];
return total / nums.length;
```

- want loop increment to be precise
- want body of the loop to be approx
Approximation-aware ISA need:

Approximate storage:

- registers
- caches
- main memory

Data should be able to transition between approximate and precise storage. The ISA must permits programs to use precise data approximately and vice-versa for full flexibility.

The paper focus on the core
Approximation-aware ISA need:

Precise instructions always carry traditional semantic guarantees.

Approximation must be confined to predictable areas.
Proposed approximation-aware ISA extension

\[ + \div \times \mid \& \]

operations

\[
\begin{align*}
\text{ADD.a} & \quad \text{Sub.a} \\
\text{MUL.a} & \quad \text{DIVF.a} \\
\text{AND.a} & \quad \text{OR.a} \\
\text{CMPLE.a} & \quad \text{XNOR.a} \\
& \quad \ldots
\end{align*}
\]
Approximate operations

ADDer1r1r2r3:

some value

writes the sum of r1 and r2 to r3

• Informally, this value is expected to approximate the sum of r1 and r2. Actual error patterns depend on microarchitecture, voltage, variation and...

• Need to confine the approximation to avoid certain undefined behaviors.
Confine Approximate operations

No other register is modified

No floating point division exception is raised

Can not jump to an arbitrary address...
Complete proposed approximation-aware ISA extension

**ALU**

- Operations
  - ADD.a
  - Sub.a
  - MUL.a
  - DIVF.a
  - AND.a
  - OR.a
  - CMPLE.a
  - XNOR.a
  - ...

- Storage
  - LDx.a
  - STx.a
  - LDF.a
  - STF.a
Approximate registers

Each register in the architecture is in either precise mode or approximate mode.

- **approximate mode**
- **precise mode**

Reads from registers in approximate mode may return any value.
Approximate registers: set precision mode

The precision of a register is implicitly set by the precision of the last instruction that wrote to it.

ADD r5 r6 r7
Approximate registers: set precision mode

The precision of a register is implicitly set by the precision of the last instruction that wrote to it.

```
ADD.a r5 r6 r7
```
Approximate registers: operand access

The precision of operand access must be declared explicitly. Every instruction that takes register operands is extended to include an extra bit per operand specifying its precision.

ADD r5 r6.a r7
Approximate caching

Data enters cache with precision of the access and set the precision mode of that line.

Uses the simple cache approximation policy that a line’s precision is set by misses and writes but is not affected by read hits.
What the compiler should do…

The ISA does not enforce strict precision correspondence to avoid the need for precision state checks and simplify implementation complexity.

\[
\text{ADD r5 r6.a r7} \checkmark
\]

The compiler should help ensure the precision consistency.

After a precise store to a line, only precise loads may be issued to it until next store to it.
Hardware support for disciplined approximate programming

- ISA extension for approximation
- Design choices
- Truffle: A dual-voltage microarchitecture
Design choices

• Dual-voltage pipeline vs single-low-voltage pipeline with error correction for critical structures

• Unchecked Dual-voltage design vs checked design

• Selecting VDDL: Fine-grained voltage adjustment
Hardware support for disciplined approximate programming

- ISA extension for approximation
- Design choices
- Truffle: A dual-voltage microarchitecture
Truffle: A Dual-Voltage Microarchitecture

```java
@Approx float[] nums;
⋮
@Approx float total = 0.0f;
for (@Precise int i = 0;
    i < nums.length;
    ++i)
    total += nums[i];
return total / nums.length;
```

Enerj code
Dual-voltage pipeline

Control plane

Data movement & processing plane
Dual-voltage pipeline

- Register File
- Integer FU
- FP FU
- Data Cache
Dual-voltage pipeline

Register File

Integer FU

Integer FU

FP FU

FP FU

Data Cache

switch
(dynamic)

replicate
(static)

switch
(dynamic)
Dual-voltage pipeline functional units: shadow structures

Only one structure is active at a time.
Dual-voltage pipeline functional units: shadow structures

- Issue width not changed (scheduler is unaware of shadowing)
- Inactive unit is power-gated
- No voltage change latency
Dual-voltage pipeline registers and caches

row selection + data (write) + precision

DV-SRAM subarray
Detailed Dual-voltage SRAM design
Voltage level-shifter and mux circuits
Evaluation

Simulated **EnerJ** programs
Precision-annotated Java [PLDI’11]
Scientific kernels, mobile app, game engine, imaging, raytracer

Modified **McPAT** models for OoO (Alpha 21264) and in-order cores
[Li, Ahn, Strong, Brockman, Tullsen, Jouppi; MICRO’09]
65 nm process, 1666 MHz, 1.5 V nominal ($V_{DDH}$)
4-wide (OoO) and 2-wide (in-order)
Includes overhead of additional muxing, shadow FUs, etc.

Extended **CACTI** for DV-SRAM structures
[Muralimanohar, Balasubramonian, and Jouppi; MICRO’07]
64 KB (OoO) and 32 KB (in-order) L1 cache
Line size: 16 bytes
Includes precision column overhead
Energy savings on in-order core

7–24% energy saved on average. Raytracer saves 14–43% energy
Energy savings on OoO core

Energy savings up to 17%. Efficiency loss up to 5% in the worst case
OoO with low-power control plane

VDDL = 0.75 V and VDDH = 1.2 V (nominal 1.5 V)
Application variation

Error resilience varies across applications
Conclusion

• **Approximation-aware ISA**
  Tightly coupled with language-level precision information

• **Dual-voltage microarchitecture**
  Data plane can run at lower voltage
  Low-complexity design relying on compiler support

• **Significant energy savings**
  Up to 43% vs. a baseline in-order core
Thank you! Questions?
Discussion

• How to compare this HW-SW-co-design to other methods, like Razor?
• Is it worthwhile to do instruction-wise voltage scaling?
• Any robustness concern in this design? Does the compiler-based statical check really free the ISA with any runtime check?
• Can this approach really help with modern computer system design? Is its application area too limited?