Architecture Support for Disciplined Approximate Programming

Hadi Esmaeilzadeh  
Adrian Sampson  
Luis Ceze  
Doug Burger  

University of Washington  
Microsoft Research  

ASPLOS 2012
mobile devices
battery usage

data centers
power & cooling costs

dark silicon
utilization wall
Disciplined approximate programming
The EnerJ programming language

Precise

References
Jump targets
JPEG header

Approximate
Pixel data
Neuron weights
Audio samples
Video frames

safely interleaves approximate and precise operation
Errors  Energy

Errors  Energy
Perfect correctness is not required

computer vision

machine learning

augmented reality

sensory data

scientific computing

information retrieval

physical simulation

games
Disciplined approximate programming
The EnerJ programming language

```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;
```
Disciplined approximate programming
The EnerJ programming language

```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;
```

approximate data storage
Disciplined approximate programming
The EnerJ programming language

```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;
```

approximate operations
Hardware support for disciplined approximate programming

```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;
```
Hardware support for disciplined approximate programming

**Compiler-directed approximation**

- Safety checks at compile time
- No expensive checks at run time
- Simplify hardware implementation
Hardware support for disciplined approximate programming

Approximation-aware ISA
Dual-voltage microarchitecture
Energy savings results
Hardware support for disciplined approximate programming

Approximation-aware ISA

Dual-voltage microarchitecture

Energy savings results
Approximation-aware languages need:

Approximate operations

Approximate data

Fine-grained interleaving
Approximation-aware languages need:

Approximate operations per instruction

Approximate data per cache line

ALU + - × | &

registers caches main memory
Traditional, precise semantics

ADD r1 r2 r3:

writes some value the sum of r1 and r2 to r3
Approximate semantics

ADD r1 r2 r3:

writes the sum of r1 and r2 to r3

Informally: r3 gets something that approximates the sum r1 + r2.

Actual error pattern depends on microarchitecture, voltage, process, variation, …
Undefined behavior

ADD r1 r2 r3:
Approximate semantics

ADD r1 r2 r3:

writes some value
writes the sum of r1 and r2 to r3

Informally: r3 gets something that approximates the sum r1 + r2.

No other register is modified.
No floating point division exception is raised.
Does not jump to an arbitrary address.
No missiles are launched.
An ISA extension with approximate semantics

**operations**

- ADD.a
- MUL.a
- CMPLE.a
- AND.a
- XNOR.a
- SRL.a
- ADDF.a
- DIVF.a

**storage**

- LDL.a
- STF.a
- STL.a
- LDF.a
- ...
Dual-voltage pipeline
Dual-voltage pipeline

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

- Register File
- Integer FU
- FP FU
- Data Cache
- switch (dynamic)
- replicate (static)
- switch (dynamic)
Dual-voltage functional units: shadow structures

One structure is active at a time.
Dual-voltage functional units: shadow structures

Issue width not changed  
(scheduler is unaware of shadowing)

Inactive unit is power-gated

No voltage change latency
Approximate storage: register modes

precise mode

approximate mode

Reads from registers in approximate mode may return any value.
Approximate storage: register modes

ADD r1 r2 r3
Approximate storage: register modes

The destination register’s mode is set to match the writing instruction.

```
ADD.a r1 r2 r3
```
Approximate storage: register modes

<table>
<thead>
<tr>
<th>r1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2</td>
</tr>
<tr>
<td>r3</td>
</tr>
<tr>
<td>r4</td>
</tr>
<tr>
<td>r5</td>
</tr>
<tr>
<td>r6</td>
</tr>
<tr>
<td>r7</td>
</tr>
<tr>
<td>r8</td>
</tr>
</tbody>
</table>

ADD r2 r3.a r4

Register operands must be marked with the register’s mode. (Otherwise, read garbage.)
Registers and caches: dual-voltage SRAMs

row selection + data (write) + precision (for sense amplifiers and precharge)

DV-SRAM subarray
Registers and caches: dual-voltage SRAMs

Mixture of precise and approximate data

Instruction stream gives access levels (compiler-specified)
Approximate storage: caches

Data enters cache with precision of the access.

Compiler: consistently treat data as approximate or precise. (Otherwise, read garbage.)
Also in the paper

Approximate main memory

Detailed DV-SRAM design

Voltage level-shifter and mux circuits

Replicated pipeline registers

Broadcast network details
Hardware support for disciplined approximate programming

Approximation-aware ISA

Dual-voltage microarchitecture

Energy savings results
Energy savings results

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

Raytracer saves 14–43% energy

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
Application accuracy trade-off

Application-specific output quality metrics
Error resilience varies across applications
Hardware support for disciplined approximate programming

```java
int p = 5;
@Approx int a = 7;
for (int x = 0..) {
    a += func(2);
    @Approx int z;
    z = p * 2;
    p += 4;
}

a /= 9;
func2(p);
a += func(2);
@Approx int y;
z = p * 22 + z;
p += 10;
```
Hardware support for disciplined approximate programming

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
Future work on **disciplined approximate programming**

- Approximate accelerators
- Precision-aware programmer tools
- Non-voltage approximation techniques