Disciplined Approximate Computing: From Language to Hardware, and Beyond

Luis Ceze

University of Washington

joint work with Adrian Sampson, Hadi Esmaelizadeh, Mike Ringenburg, Renee StAmant, Ben Ransford, Andre Baixo, Thierry Moreau, Dan Grossman, Mark Oskin (UW), Karin Strauss, Doug Burger, Todd Mytkowicz and Kathryn McKinley (Microsoft Research).
These applications consume a lot (most?)

Often input data is inexact by nature (from sensors)

They have multiple acceptable outputs
These applications consume a lot (most?)

Often input data is inexact by nature (from sensors)

They have multiple acceptable outputs

They do not require “perfect execution”
Notions of “approximation” have been around for a long time...

Floating point
Lossy compression
Iterative algorithms

...
Notions of “approximation” have been around for a long time...

Floating point
Lossy compression
Iterative algorithms

...
Notions of “approximation” have been around for a long time...

So what is “Approximate computing” then?

Floating point
Lossy compression
Iterative algorithms

...
So what is “Approximate computing” then?

Notions of “approximation” have been around for a long time...

Floating point
Lossy compression
Iterative algorithms
...
Notions of “approximation” have been around for a long time...

So what is “Approximate computing” then?

- Floating point
- Lossy compression
- Iterative algorithms

...
Notions of “approximation” have been around for a long time...

So what is “Approximate computing” then?

Floating point
Lossy compression
Iterative algorithms
...

Sources of systematic accuracy loss:

• Unsound code transformations, \( \sim2X \)
• Unreliable, probabilistic hardware (near/sub-threshold, etc.), \( \sim5X \)
• Fundamentally different, inherently inaccurate execution models, “closer to physics” (e.g., neural networks, analog computing), \( \sim10-100X \)
But approximation needs to be done carefully... or...
“Disciplined” approximate programming

Precise

- references
- jump targets
- JPEG header

Approximate

- pixel data
- neuron weights
- audio samples
- video frames
"Disciplined" approximate programming

- Programmer has direct control of approximate/precise and the flow
- System is free to approximate as long as rules are obeyed
Disciplined Approximate Programming (EnerJ, EnerC,...)

```c
int p = 5;
@Approx int a = 7;
for (int x = 0..) {
    a += func(2);
    @Approx int z;
    z = p * 2;
    p += 4;
}
a /= 9;
p += 10;
socket.send(z);
write(file, z);
```
Disciplined Approximate Programming (EnerJ, EnerC,...)

```java
int p = 5;
@Approx int a = 7;
for (int x = 0..) {
    a += func(2);
    @Approx int z;
    z = p * 2;
    p += 4;
}
```
```java
a /= 9;
p += 10;
socket.send(z);
write(file, z);
```
Disciplined Approximate Programming (EnerJ, EnerC,...)

Relaxed Algorithms

Aggressive Compilation

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

a /= 9;
p += 10;
socket.send(z);
write(file, z);
Disciplined Approximate Programming (EnerJ, EnerC,...)

Relaxed Algorithms

Aggressive Compilation

Approximate Data Storage

```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;
p += 10;
socket.send(z);
write(file, z);
```
Disciplined Approximate Programming (EnerJ, EnerC,...)

- Relaxed Algorithms
- Aggressive Compilation
- Approximate Data Storage
- Variable-Accuracy ISA

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

a /= 9;
p += 10;
socket.send(z);
write(file, z);
```
Disciplined Approximate Programming (EnerJ, EnerC,...)

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

a /= 9;
p += 10;
socket.send(z);
write(file, z);
```
Disciplined Approximate Programming (EnerJ, EnerC,...)

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

a /= 9;
p += 10;
socket.send(z);
write(file, z);
Disciplined Approximate Programming (EnerJ, EnerC,...)

- Relaxed Algorithms
- Aggressive Compilation
- Approximate Data Storage
- Variable-Accuracy ISA
- Approximate Logic/Circuits

Variable-quality wireless communication

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

a /= 9;
p += 10;
socket.send(z);
write(file, z);
```
Disciplined Approximate Programming (EnerJ, EnerC,...)

- Relaxed Algorithms
- Aggressive Compilation
- Approximate Data Storage
- Variable-Accuracy ISA
- Approximate Logic/Circuits

```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;
p += 10;
socket.send(z);
write(file, z);
```
Goal: support a wide range of approximation techniques with a **single unified abstraction**.
The plan for this talk

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
</tr>
<tr>
<td>Compiler</td>
</tr>
<tr>
<td>Architecture</td>
</tr>
<tr>
<td>Circuits</td>
</tr>
</tbody>
</table>
The plan for the rest of this talk

- Application
  - EnerJ
  - QoR

- Compiler
  - Truffle
  - NPU

- Circuits

- Type system for where-to-approximate
- ISA w/ variable accuracy

- Quality of results
- Neural networks as accelerators

- Approximate Storage
- Approximate Wireless
- Prototype, etc
EnerJ: Approximate Data Types for Safe and General Low-Power Computation, PLDI 2011
EnerJ

Separate critical and non-critical program components. *Analyzable statically.*
EnerJ: Separate critical and non-critical program components. Analyzable *statically*.

```java
int a = ...;
int p = ...;
```
Separate critical and non-critical program components. *Analyzable statically.*

```java
@Approx int a = ...;
@Precise int p = ...;
```
Separate critical and non-critical program components. *Analyzable statically.*

```java
@Approximate int a = ...;
@Precise int p = ...;

p = a;  // ✗
\[x\]
a = p;  // ✓
```
Operator overloading for approximate operations:

Endorsement of approximate values:

Dealing with implicit flows in control:
• Operator overloading for approximate operations: \( p + p; \ p + a; \ a + a; \)

• Endorsement of approximate values:

• Dealing with implicit flows in control:
Operator overloading for approximate operations: \( p + p;\ p + a;\ a + a; \)

Endorsement of approximate values:

\[ \checkmark \ p = \text{endorse}(a); \]

Dealing with implicit flows in control:
@Approx int a = ...;
@Precise int p = ...;

- Operator overloading for approximate operations: \( p + p; \ p + a; \ a + a; \)

- Endorsement of approximate values:
  \[ \checkmark \ p = \text{endorse}(a); \]

- Dealing with implicit flows in control:
  \[ \times \text{if} (a == 10) \]
  \[ \quad p = 2; \]
  \[ \} \]
@Approx int a = ...;
@Precise int p = ...;

- Operator overloading for approximate operations: \( p + p; \ p + a; \ a + a; \)

- Endorsement of approximate values:
  \( \checkmark \ p = \text{endorse}(a); \)

- Dealing with implicit flows in control:
  \( \checkmark \text{if} \ (\text{endorse}(a == 10)) \{ \p = 2; \} \)
language for where-to-approximate

Application

EnerJ  Monitoring

quality evaluation
Application

EnerJ  Monitoring

language for where-to-approximate

quality evaluation
How good is my final output?

• **Quality-of-Result (QoR)**
• Application dependent
  – e.g., % of bad pixels, deviation from expected value, % of poorly classified images, car crashes, etc…
Specifying and checking QoR
Specifying and checking QoR

```plaintext
res = computeSomething();
assert diff(res, res') < 0.1;
```

precise version of the result
Verifying quality expressions

Expressing and Verifying Probabilistic Assertions, PLDI’14
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled
precise
re-execution
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled
precise
re-execution
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled
precise
re-execution
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled
precise
re-execution

\[-\varepsilon\ ?\]
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled
precise
re-execution

\[ \epsilon \lesssim \epsilon \]

\[ \epsilon \lesssim \epsilon \]

\[ \epsilon \lesssim \epsilon \]
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled
precise
re-execution

\[ < \varepsilon \]

\[ < \varepsilon \]
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

<table>
<thead>
<tr>
<th>Sampled precision re-execution</th>
<th>Simple verification functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

<table>
<thead>
<tr>
<th>Sampled precise re-execution</th>
<th>Simple verification functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Sampled" /> <img src="image" alt="precise" /> <img src="image" alt="re-execution" /></td>
<td><img src="image" alt="Simple" /> <img src="image" alt="verification functions" /></td>
</tr>
</tbody>
</table>

\[ - \varepsilon? \]

\[ - \varepsilon? \]
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

- Sampled
  - precise
  - re-execution

- Simple verification functions

- Fuzzy Memoization
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

Sampled precise re-execution

Simple verification functions

Fuzzy Memoization
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

- Sampled
  - precise
    - re-execution

- Simple
  - verification functions

- Fuzzy Memoization
Online QoR monitoring

Can *react* – recompute or reduce approximation
But needs to be cheap!

<table>
<thead>
<tr>
<th>Sampled precise re-execution</th>
<th>Simple verification functions</th>
<th>Fuzzy Memoization</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Sampled precise re-execution" /></td>
<td><img src="image" alt="Simple verification functions" /></td>
<td><img src="image" alt="Fuzzy Memoization" /></td>
</tr>
</tbody>
</table>
What about *actual* approximate execution?

- Application
  - EnerJ
  - Monitoring
  - Compiler
    - Truffle

language for where-to-approximate

ISA w/ variable accuracy

quality evaluation
What about actual approximate execution?

- Truffle
- ISA w/ variable accuracy
  - language for where-to-approximate
- Compiler
  - EnerJ
  - Monitoring
- Application

quality evaluation
Hardware support for disciplined approximate execution

```c
int p = 5;
@Approx int a = 7;
for (int x = 0; x < 4; x++) {
    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;
```

Architecture Support for Disciplined Approximate Programming, ASPLOS 2012
@Approx float[] nums;
:
@Approx float total = 0.0f;
for (@Precise int i = 0;
    i < nums.length;
    ++i)
    total += nums[i];
return total / nums.length;
@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
@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
Relaxing the hardware-software interface

- EnerJ
- Compiler
- ISA
- Architecture
- Circuits
Approximation-aware ISA

ld    0x04 r1
ld    0x08 r2
add   r1   r2   r3
st    0x0c r3
Approximation-aware ISA

```
ld    0x04 r1
ld    0x08 r2
add.a r1    r2    r3
st.a    0x0c r3
```
Approximation-aware ISA

ld 0x04 r1
ld 0x08 r2
add.a r1 r2 r3
st.a 0x0c r3

operations

storage

ALU

registers
caches
main memory
Dual-voltage pipeline

- Fetch
- Decode
- Reg Read
- Execute
- Memory
- Write

- Branch Predictor
- Instruction Cache
- ITLB

- Fetch
- Decode
- Register File
- Integer FU
- FP FU
- Integer FU
- DTLB
- Data Cache
- Register File

replicated functional units

dual-voltage SRAM arrays
Dual-voltage pipeline

replicated functional units

replicated functional units

dual-voltage SRAM arrays

7–24% energy saved on average

(fft, game engines, ray tracing, QR code readers, etc)

(scope: processor + memory)
Dual-voltage pipeline

7–24% energy saved on average

(fft, game engines, raytracing, QR code readers, etc)

(not good... :(

(though better implementations likely)

(scope: processor + memory)
Amdahl’s law... damn!
Amdahl’s law... damn!

• Benefit limited to what can be approximated
• Instruction control can not be approximated
How can we get rid of exact instruction bookkeeping?
How can we get rid of exact instruction bookkeeping?

If behavior is approximate, why program it precisely?
Compiler

EnerJ  Monitoring

Truffle  NPU

language for where-to-approximate

traditional architecture

dynamic quality evaluation

neural networks as accelerators
Compiler

EnerJ | Monitoring

Truffle | NPU

Application

dynamic quality evaluation
neural networks as accelerators

language for where-to-approximate
traditional architecture
Why Neural Networks as Approximate Accelerators?

Neural Acceleration of General-Purpose Approximate Programs, MICRO 2012
General-Purpose Code Acceleration with Limited-Precision Analog Computation, ISCA 2014
Why Neural Networks as Approximate Accelerators?

Very efficient hardware implementations!

Trainable to mimic many computations!

Recall is imprecise.

Fault tolerant

[Temam, ISCA 2012]

Neural Acceleration of General-Purpose Approximate Programs, MICRO 2012
General-Purpose Code Acceleration with Limited-Precision Analog Computation, ISCA 2014
Neural acceleration

Program
Neural acceleration

Find an approximate program component
Neural acceleration

Find an approximate program component
Neural acceleration

Find an approximate program component

Compile the program and train a neural network
Neural acceleration

Find an approximate program component

Compile the program and train a neural network

Execute on a fast Neural Processing Unit (NPU)
An example: Sobel filter

```cpp
#pragma once

@approx float grad(approx float[3][3] p) {
    ...
}

void edgeDetection(aImage &src,
                   aImage &dst) {
    for (int y = ...) {
        for (int x = ...) {
            dst[x][y] =
                grad(window(src, x, y));
        }
    }
}

@approx float dst[][];
```
An example: Sobel filter

edgeDetection()

```c
@approx float grad(approx float[3][3] p) {
    ...
}
```

```c
void edgeDetection(aImage &src,
                   aImage &dst) {
    for (int y = ...) {
        for (int x = ...) {
            dst[x][y] =
                grad(window(src, x, y));
        }
    }
}
```

```c
@approx float dst[][];
```
Empirically selecting target code

Program
Empirically selecting target code

Program → √ → Accelerated Program

Program → ✗ → Accelerated Program
Empirically selecting target code

Each region of code leads to a different NN configuration.
Neural Processing Unit

Core

- enq.d → input
- deq.d ← output
- enq.c → configuration
- deq.c ←

NPU
A digital NPU

Bus Scheduler

scheduling

Processing Engines

input

output
A digital NPU

Bus Scheduler

input

output

scheduling

Processing

Many other implementations
- FPGAs
- Analog
- Hybrid HW/SW
- SW only? on GPUs?
...
How do the NNs look like in practice?

- **edge detection**
  - 88 static instructions (56% of dynamic instructions)
  - 18 neurons

- **triangle intersection**
  - 1,079 static x86-64 instructions (97% of dynamic instructions)
  - 60 neurons
    - 2 hidden layers
How do the NNs look like in practice?

- **edge detection**
  - 88 static instructions
  - 56% of dynamic instructions

- **triangle intersection**
  - 1,079 static x86-64 instructions
  - 97% of dynamic instructions

- Neural network with 18 neurons
  - Output: $o_j = \text{sigmoid}\left(\sum_i w_{ji} x_{ji}\right)$

- Neural network with 60 neurons
  - 2 hidden layers
Summary of results

2.3x average speedup
Ranges from 0.8x to 11.1x

3.0x average energy reduction for digital, ~10x for analog
All benchmarks benefit

Quality loss below 10% in all cases
Based on application-specific quality metrics

Just one possible design. Many others possible. Analog is where the big gains are likely (~10x+).
Key here is algorithmic transformation that enables new more efficient execution models.
language for where-to-approximate

traditional architecture

Application

EnerJ    Monitoring

Compiler

Truffle    NPU

Circuits

Approximate Storage

Approximate Wireless

Prototype
dynamic quality evaluation

neural networks as accelerators
Approximate Storage

Approximate Wireless

Prototype
22.5 GB Available
34.6 GB Used

- Music: 15.9 GB
- Photos & Camera: 11.3 GB
- TripAdvisor: 549 MB
- Stay.com: 535 MB
- Keynote: 389 MB
Approximate mass storage with Flash and PCM

Approximate Storage in Solid State Memories [MICRO’13]
Approximate mass storage with Flash and PCM

Approximate Storage in Solid State Memories [MICRO’13]

Cells wear out over time
Approximate mass storage with Flash and PCM

Approximate Storage in Solid State Memories [MICRO’13]

Cells **wear out** over time

**Multi-level** cells are slow or unreliable
Approximate mass storage with Flash and PCM

Approximate Storage in Solid State Memories [MICRO’13]

Cells wear out over time

Use worn-out memory for approximate data instead of throwing it away.

Multi-level cells are slow or unreliable
Approximate mass storage with Flash and PCM

Approximate Storage in Solid State Memories [MICRO’13]

Cells **wear out** over time

Use **worn-out** memory for **approximate** data instead of throwing it away.

**Multi-level** cells are slow or unreliable

Trade off **accuracy** for performance/density in **multi-level cell** accesses.
Precise Multi-level Cells
Approximate Multi-level Cells
Typical Trade-off in Multi-Level Cells

Fast   ←   Dense
Adding a New Trade-Off Axis

Fast  ⇌  Dense

Accurate

- Fast vs. Dense
- Accurate vs. Dense
- Accurate vs. Fast
Approximate Wireless Communication
Approximate Wireless Communication

precise
Approximate Wireless Communication
Approximate Wireless Communication
Approximate Wireless Communication

< 3% of bits are bad!
Approximate Wireless Communication

Configurable-quality wireless protocol. Quality automatically set by the data type.

< 3% of bits are bad!
Neural Acceleration on a programmable SoC
Showing End-to-End benefit

Mobile Vision/Augmented Reality
Linux on Zynq SoC (ARM CPU + FPGA)

Neural Accelerator + Compiler Support + Approx. FPGA

Measure Energy Savings
Measure Speedup
Evaluate User Experience
Evaluate Programmer Effort
How will approximate computing fail?

- Applications can’t take advantage of approximation opportunities
- Programmers aren’t able to write/debug/test approximate code
- Quality assurance problems
- Marketing reasons: “buy my flaky system!”
How will approximate computing fail?

• Applications can’t take advantage of approximation opportunities
• Programmers aren’t able to write/debug/test approximate code
• Quality assurance problems
• Marketing reasons: “buy my flaky system!”
How will approximate computing fail?

• Applications can’t take advantage of approximation opportunities
• Programmers aren’t able to write/debug/test approximate code
• Quality assurance problems
• Marketing reasons: “buy my flaky system!”

Good luck debugging that...

Oh god, this sounds awful .. and it has enough potential to actually make things worse, if hardware vendors end up shoving it down our throats by force (i.e. people recalculating things over and over again, just to be safe; non-standard/"unofficial" hardware that tries to work around the limitations in embedded devices, and other things like that).
How will approximate computing fail?

• Applications can’t take advantage of approximation opportunities
• Programmers aren’t able to write/debug/test approximate code
• Quality assurance problems
• Marketing reasons: “buy my flaky system!”

Oh god, this sounds awful .. and it has enough potential to actually make things worse, if hardware vendors end up shoving it down our throats by force (i.e. people recalculating things over and over again, just to be safe; non-standard/"unofficial" hardware that tries to work around the limitations in embedded devices, and other things like that).

There was a somewhat related post to this a few weeks ago here.

The basic idea was simple: if hardware suffers more transient failures as it gets smaller, why not allow software to detect erroneous computations and re-execute them? This idea seemed promising until John realized THAT IT WAS THE WORST IDEA EVER. Modern software barely works when the hardware is correct, so relying on software to correct hardware errors is like asking Godzilla to prevent Mega-Godzilla from terrorizing Japan.
Other ongoing effort

- **Understanding** specialization vs. approximation benefits
- **Compiler-only** approximation w/ unsound transformations
- **HCI aspects**: how do measure user satisfaction? do incentives matter in choosing quality?
- **Language support** for QoR (quality of results, probabilistic assertions)
- **Tools** to help programmers w/ porting, testing and debugging
- Exploring uses in **energy-harvesting**-based devices
- **approxbensch.org**
Conclusion

We need to exploit application properties and co-design hardware-software for better efficiency.

Getting closer to physics might lead to very big efficiency gains.

Our goal is to exploit approximate computing across the system. (compute, storage, communication)

Key aspect is co-designing programming model with approximation techniques: disciplined approximate programming.

Early results encouraging. Approximate computing can potentially save our bacon in a post-Dennard era and be in the survival kit for dark silicon.
Thanks!

Luis Ceze
University of Washington
luisceze@cs.washington.edu

NSF
Microsoft®
Qualcomm
C-FAR
Saimmipa