Approximating to the Last Bit

Thierry Moreau, Adrian Sampson, Luis Ceze
{moreau, luisceze}@cs.washington.edu, asampson@cornell.edu

WAX 2016
co-located with ASPLOS 2016
April 3rd 2016
What this Talk is About

How many bits in a program are really that important?

1 - AXE: Quality Tuning Framework

2 - PERFECT Benchmark Study
Precision Tuning

More precision means larger memory footprint, more data movement, more energy used in computation
Precision Tuning

More precision means larger memory footprint, more data movement, more energy used in computation
Precision Tuning

More precision means larger memory footprint, more data movement, more energy used in computation
AXE Precision Tuning Framework

Goal: Maximize Bit-Savings given a Quality Target
AXE Precision Tuning Framework

Built on top of **ACCEPT**, the approximate C/C++ compiler

http://accept.rocks
AXE Precision Tuning Framework

Default (no bit-savings)

instruction 0
instruction 1
instruction 2
...
instruction n-1
instruction n

bit-savings

quality

bad  OK
AXE Precision Tuning Framework

Coarse-Grained Precision Reduction

instruction 0
instruction 1
instruction 2
...
instruction n-1
instruction n

bit-savings

quality

bad ↔ OK
AXE Precision Tuning Framework

Fine-Grained Precision Reduction

instruction 0
instruction 1
instruction 2
...
instruction n-1
instruction n

bit-savings

quality

bad ← OK
## PERFECT Benchmark Suite

<table>
<thead>
<tr>
<th>Application Domain</th>
<th>Kernels</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERFECT Application 1</td>
<td>Discrete Wavelet</td>
<td>[120dB to 10dB] (0.0001% to 31.6% MSE)</td>
</tr>
<tr>
<td></td>
<td>2D Convolution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Histogram Equalization</td>
<td></td>
</tr>
<tr>
<td>Space Time Adaptive Processing</td>
<td>Outer Product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>System Solve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inner Product</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpolation 1</td>
<td></td>
</tr>
<tr>
<td>Synthetic Aperture Radar</td>
<td>Interpolation 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Back Projection</td>
<td></td>
</tr>
<tr>
<td>Wide Area Motion Imaging</td>
<td>Debayer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image Registration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change Detection</td>
<td></td>
</tr>
<tr>
<td>Required Kernels</td>
<td>FFT 1D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FFT 2D</td>
<td></td>
</tr>
</tbody>
</table>
1 - PERFECT Dynamic Instruction Mix

- load/store: 27%
- int arith: 4%
- fp arith: 31%
- int arith: 25%
- control: 11%
- math: 1%

Safe to approximate
Precise
1 - PERFECT Dynamic Instruction Mix

Long latency ops are all safe to approximate

- Safe to approximate: 31%
- Precise: 1%

Legend:
- Green: Safe to approximate
- Gray: Precise
1 - PERFECT Dynamic Instruction Mix

Memory ops are mostly safe to approximate (mostly data vs. pointers)

- load/store 27%
- Safe to approximate
- Precise
1 - PERFECT Dynamic Instruction Mix

Control and address computation must remain precise.
2 - Bit-Savings over Approximate Instructions

Approximate  High Quality

Bit-Savings

- 83% at 10 dB
- 74% at 20 dB
- 57% at 40 dB
- 48% at 60 dB
- 40% at 80 dB
- 32% at 100 dB
- 26% at 120 dB

Average SNR (dB)
2 - Bit-Savings over Approximate Instructions

Bit-Savings

Average SNR (dB)

PERFECT Manual
0.001% MSE

0%
20%
40%
60%
80%
100%

10
20
40
60
80
100
120

83%
74%
57%
48%
40%
32%
26%
2 - Bit-Savings over Approximate Instructions

Approximate Computing
10% MSE

PERFECT Manual
0.001% MSE
Future Architectural Challenges

Mechanisms to translate bit-savings into energy savings?

New data types/representations?

ISA extensions?
Thank You!

Approximating to the Last Bit

Thierry Moreau, Luis Ceze, Adrian Sampson
{moreau, luisceze}@cs.washington.edu, asampson@cornell.edu

WAX 2016
co-located with ASPLOS 2016
April 3rd 2016
Backup Slides
Bit Savings

Explore the opportunity for precision reduction in a *hardware-agnostic* way

\[
\text{BitSavings} = \sum_{insn_{\text{static}}} \frac{(\text{precision}_{\text{ref}} - \text{precision}_{\text{approx}})}{\text{precision}_{\text{ref}}} \times \frac{\text{execs}}{\text{execs}_{\text{total}}}
\]
Framework Overview

Built on top of ACCEPT, the approximate C/C++ compiler

[http://accept.rocks]
void conv2d (pix *in, pix *out, flt *filter)
{
    for (row) {
        for (col) {
            flt sum = 0
            int dstPos = …
            for (row_offset) {
                for (col_offset) {
                    int srcPos = …
                    int fltPos = …
                    sum += in[srcPos] * filter[fltPos]
                }
            }
            out[dstPos] = sum / normFactor
        }
    }
}
void conv2d (APPROX pix *in, APPROX pix *out, APPROX flt *filter) {
    for (row) {
        for (col) {
            APPROX flt sum = 0
            int dstPos = ...
            for (row_offset) {
                for (col_offset) {
                    int srcPos = ...
                    int fltPos = ...
                    sum += in[srcPos] * filter[fltPos]
                }
            }
            out[dstPos] = sum / normFactor
        }
    }
}
Program Annotation

Takeways:
Annotating \textbf{data} is intuitive (~10 mins to annotate a kernel)
Variables used to index arrays cannot be safely approximated

tips on annotating programs faster

```c
typedef float flt
typedef int pix
```

```c
typedef APPROX float flt
typedef APPROX int pix
```
void conv2d (APPROX pix *in, APPROX pix *out, APPROX flt *filter)
{
    for (row) {
        for (col) {
            APPROX flt sum = 0
            int dstPos = ...
            for (row_offset) {
                for (col_offset) {
                    int srcPos = ...
                    int fltPos = ...
                    sum += in[srcPos] * filter[fltPos]
                }
            }
            out[dstPos] = sum / normFactor
        }
    }
}
Approximate Binary

Program Inputs & Quality Metrics

Annotated Program

ACCEPT static analysis

ILPC*

ACCEPT error injection & instrumentation

Approximate Binary

Execution & Quality Assessment

Quality Results & Bit Savings

Each instruction in the ILCP acts as a quality knob that the autotuner can use to maximize bit-savings

Instruction-Level Precision Configuration (ILPC)

conv2d:13:7:load:Int32
conv2d:13:10:load:Float
conv2d:13:11:fmul:Float
conv2d:13:12:fadd:Float
conv2d:15:1:fdiv:Float
conv2d:15:7:store:Int32

Error Injection

Instruction-Level Precision Configuration

(ACCEPT)
The programmer provides a quality assessment script to evaluate quality on the program output.
Greedy iterative algorithm: reduces precision requirement of the instruction that impacts quality the least

Finds solution in $O(m^2n)$ worst case where $m$ is the number of static safe-to-approximate instructions and $n$ are the levels of precision for all instructions
Autotuner

The autotuner greedily maximizes bit-savings as the quality target is lowered.

SNR (dB - higher is better)
Precision “Guarantees”

Currently **empirically derived** and **input dependent**

**Future work** would extend on the current infrastructure to assimilate data dependence information in order to derive **formal error guarantees**
1 - PERFECT Dynamic Instruction Mix

![Bar chart showing the distribution of different instruction types across various categories.](chart.png)
## PERFECT Benchmark Suite

<table>
<thead>
<tr>
<th>Application Domain</th>
<th>Kernels</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERFECT Application 1</td>
<td>Discrete Wavelet</td>
<td>SNR [120dB to 10dB]</td>
</tr>
<tr>
<td></td>
<td>2D Convolution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Histogram Equalization</td>
<td></td>
</tr>
<tr>
<td>Space Time Adaptive Processing</td>
<td>Outer Product</td>
<td>10 \log_{10} \left( \frac{\sum_{k=1}^{N}</td>
</tr>
<tr>
<td></td>
<td>System Solve</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Inner Product</td>
<td></td>
</tr>
<tr>
<td>Synthetic Aperture Radar</td>
<td>Interpolation 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpolation 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Back Projection</td>
<td></td>
</tr>
<tr>
<td>Wide Area Motion Imaging</td>
<td>Debayer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image Registration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Change Detection</td>
<td></td>
</tr>
<tr>
<td>Required Kernels</td>
<td>FFT 1D</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FFT 2D</td>
<td></td>
</tr>
</tbody>
</table>
2 - Bit-Savings over Approximate Instructions

Aggregate Bit Savings

2dconv | dwt | histeq | outer | systemsolve | inner | interp1 | interp2 | bp | debayer | lucaskanade | changedet | fft1d | fft2d | AVERAGE
You don’t need a lot of bits to obtain an acceptable output!
Architectural Target

Core Energy Breakdown

overheads
compute
compute

General Purpose CPU
Vector Processor*
Accelerators

specialization

the smaller the overheads, the larger the potential gains

* [Quora, Venkataramani et al., MICRO2013]
Precision Scaling

Mechanisms for precision scalability:

• Fine-grained ALU power gating*
• Bit-sliced ALU units
• Lossy Compression

* [Quora, Venkataramani et al., MICRO2013]