Approximate Computing on Approximate Data

Martin Rinard

MIT CSAIL
Loop Perforation

Problem

Program Takes Too Long To Run
(Or Consumes Too Much Energy)
Solution
[Misailovic et. al. ICSE 2010, Sidiroglou et. al. FSE 2011]

Profile program
Find loops that take most time
Perforate the loops
  • Don’t execute all loop iterations
  • Instead, skip some iterations

```c
for (i = 0; i < n; i++) { ... }
```
Profile program
Find loops that take most time
Perforate the loops
  • Don’t execute all loop iterations
  • Instead, skip some iterations

\[\text{for } (i = 0; i < n; i++) \{ \ldots \}\]

\[\text{for } (i = 0; i < n; i += 2) \{ \ldots \}\]
**Solution**

[Misailovic et. al. ICSE 2010, Sidiroglou et. al. FSE 2011]

Profile program

Find loops that take most time

Perforate the loops

- Don’t execute all loop iterations
- Instead, skip some iterations

```c
for (i = 0; i < n; i++) { ... }
```

```c
for (i = 0; i < n; i += 4) { ... }
```
Common Reaction

OK, I agree program should run faster
But you can’t do this
Because you will get wrong result!
Our Response

OK, I agree program should run faster

But you can’t do this

Because you will get wrong result!

We absolutely can do this
You won’t get the wrong result
You may get a different result
Key Points

Need to find right loops to perforate
Testing can identify viable loops
Converging loops
Distance metrics for heuristic searches
When you find right loops to perforate, can get
Significant time/energy savings (6X)
Acceptable inaccuracy
What This Talk Is About

Approximation to Reduce Energy Consumption

Approximate Data

Phillip Stanley-Marbell

Approximate Computation

Sasa Misailovic
OLED Displays Are Increasingly Popular

Smart Watches with OLED Displays:

- LG
- Samsung
- Apple

Tablets with OLED Displays:
OLED Displays Consume Lots of Power

ARM Cortex-M0 Processor: 0.3-18 mWatts
Intel Atom 21000 6 Watts
120x120 OLED Display: 8.5 Watts
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Power (mW)</th>
<th>Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>7328.02</td>
<td></td>
</tr>
<tr>
<td>Shape transform 1</td>
<td>5927.78</td>
<td>19.1%</td>
</tr>
<tr>
<td>Shape transform 2</td>
<td>8514.26</td>
<td>-16.2%</td>
</tr>
<tr>
<td>Shape transform 3</td>
<td>6753.95</td>
<td>7.8%</td>
</tr>
</tbody>
</table>
Original
1647.3 mW
(0.0% savings)

Color Transform
1319.48 mW
(19.9% savings)

Color + 0.88x area
1123.46 mW
(31.8% savings)

Color + 0.72x area
930.72 mW
(43.5% savings)

Original
7716.36 mW
(0.0% savings)

Color Transform
7616.04 mW
(1.3% savings)

Color + 0.88x area
7924.70 mW
(-2.7% savings)

Color + 1.4x area
6358.28 mW
(17.6% savings)
On OLED displays, blue uses almost twice power of green.

Numbers From On-Board Current Measurements
Wrote device drivers to measure current driving display
Characterized power for all colors
Color Transforms

Characterize Tradeoff Curve:
Start with an image and a distance in an underlying color space
Find a new image within that distance that minimizes power

Power model used in minimization formulation
(calibrated with measurement data)

\[
p(v) = \sum_{c \in \{r, g, b\}} \sum_{i=1}^{N} \frac{1}{2} \alpha_c v_c[i]^2 + \beta_c v_c[i]
\]

Closed-form formula for color transformation
(power minimization under a distance constraint)

\[
u_c[i] = \frac{\lambda v_c[i] - \beta_c}{\lambda + \alpha_c}
\]

Where
\[
\begin{align*}
u_c[i] & \quad \text{Transformed pixel value for } i \text{ th pixel on } c \text{ th channel} \\
v_c[i] & \quad \text{Original value of } i \text{ th pixel on } c \text{ th channel} \\
\alpha & \quad \text{Parameter from } i \text{ th model on } c \text{ th channel} \\
\beta & \quad \text{Parameter from power model} \\
\lambda & \quad \text{Power-vs-distance tradeoff parameter}
\end{align*}
\]
Key Question

Relationship between $\lambda$ and human perception
Use Amazon Mechanical Turk

Discretize $\lambda$
Generate $\lambda$ variants of pictures
Mechanical Turk workers rate pictures (0-3)
370 People, 2636 picture ratings

Result: $f$
perceived goodness = $f(\lambda)$
choose $\lambda$ such that $f(\lambda) = 2$, use that $\lambda$
For $\lambda = 3.28$, CIELAB space, get 33%-50% power savings.
Cairo is a standard, widely-used graphics API
Firefox, Graphviz, Poppler, …
CrayonGen intercepts Cairo calls, generates Crayon IR
Capture layering information
Shape optimization for drawing operations
Closed-form color transformation
Crayon implementation is about 12K LOC
Cairo-specific code is about 232 LOC
int main (int argc, char * argv[]) {
    cairo_surface_t *    surface;
    cairo_t *            cr;

    surface = cairo_image_surface_create(CAIRO_FORMAT_RGB24, 96, 96);
    cr       = cairo_create(surface);

cairo_set_line_width(cr, DRAWING_LINE_WIDTH);
cairo_scale(cr, 96, 96);

draw_mit_logo(cr,
    /* mtidot_red   */ MIT_LOGO_MITLIGHTGRAY_R,
    /* mtidot_green */ MIT_LOGO_MITLIGHTGRAY_G,
    /* mtidot_blue  */ MIT_LOGO_MITLIGHTGRAY_B,
    /* istem_red    */ 1.0,
    /* istem_green  */ 1.0,
    /* istem_blue   */ 1.0);

cairo_surface_flush(surface);
cairo_surface_write_to_png (surface, "mitlogo.png");
cairo_destroy(cr);

cairo_surface_destroy(surface);

    return 0;
}

void draw_mit_logo(cairo_t *cr, double mtidot_red, double mtidot_green, double mtidot_blue, double istem_red, double istem_green, double istem_blue)
{
    /* M: */
    cairo_set_source_rgb(cr, mtidot_red, mtidot_green, mtidot_blue);
    cairo_rectangle(cr,
        /*    x    */ 0.0,
        /*    y    */ 0.0,
        /*  width  */ MIT_LOGO_BAR_WIDTH,
        /*  height */ MIT_LOGO_LETTER_HEIGHT);
    cairo_fill(cr);
    cairo_rectangle(cr,
        /*    x    */ MIT_LOGO_BAR_WIDTH+MIT_LOGO_BAR_SPACING,
        /*    y    */ 0.0,
        /*  width  */ MIT_LOGO_BAR_WIDTH,
        /*  height */ 3*MIT_LOGO_BAR_WIDTH);
    cairo_fill(cr);
    cairo_rectangle(cr,
        /*    x    */ 2*(MIT_LOGO_BAR_WIDTH+MIT_LOGO_BAR_SPACING),
        /*    y    */ 0.0,
        /*  width  */ MIT_LOGO_BAR_WIDTH,
        /*  height */ MIT_LOGO_LETTER_HEIGHT);
    cairo_fill(cr);
    ...
Sensor Power Consumption

- Processor: 0.309 mW
- Gyroscope: 18.300 mW
- Humidity: 0.540 mW
- Magnetometer: 0.408 mW
- IR/Color: 0.528 mW
- Accelerometer: 0.312 mW
- Pressure: 0.010 mW
- Bluetooth LE: 0.436 mW
Reduced Power Sensor Operation

L3G4200D Manual: Run sensor at 1.8V-3.6V

Us: Why would we do this?
   We can save lots of power if reduce V!

Power = $C_{L3G4200D} V^2$
Sensor Still Works (Mostly)!

L3G4200D Gyro Sensor
And We Save Power

L3G4200D Gyro Sensor

![Graph showing dynamic power savings vs. errors per 10^3 readings. The graph includes data points and error bars for mean and standard deviation.]
Moving Sensor Data To Processor Costs Energy

Every 1→ 0 or 0 → 1 transition costs energy!
Reducing Transitions

Let’s say we decide to tolerate \( m \) error. Why not just make \( \log_2(m) \) bits all 0 or all 1?
Encoding Algorithm Overview

**Problem:** Given input $s=64$, find an encoded value $t$, such that $|s-t| \leq m$, for $m = 16$, and for which $\#_\delta(t) < \#_\delta(s)$

**Phase 1:** Identify transition positions and cumulative run counts

- **Moving left from LSB**, transition position set is {5, 6}
- Run of 0s at bit position 5 will have contribution 63 if flipped
- Run of 1s at bit position 6 will have contribution 64 if flipped

**Phase 2:** Find runs at that can be flipped in opposite directions

1. Start from first transition seen, **moving right from MSB**
2. Can complement removing transition at position 6 (with contribution -64) with lower-order run of 0s (contribution +63)
   - Resulting deviation is 1
   - Number of transitions is reduced from 2 to 1
Great, But How Does it Affect Real End-to-End Applications?

<table>
<thead>
<tr>
<th>Tolerable Deviation</th>
<th>Image A</th>
<th>OCR Text</th>
<th>Transition Reduction</th>
<th>Image B</th>
<th>OCR Text</th>
<th>Transition Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>EXIT</td>
<td>“EXIT”</td>
<td>0% ↓</td>
<td>EXIT</td>
<td>“EXIT”</td>
<td>0% ↓</td>
</tr>
<tr>
<td>4%</td>
<td>EXIT</td>
<td>“EXIT”</td>
<td>58% ↓</td>
<td>EXIT</td>
<td>“EXIT”</td>
<td>49% ↓</td>
</tr>
<tr>
<td>10%</td>
<td>EXIT</td>
<td>“EXIT”</td>
<td>72% ↓</td>
<td>EXIT</td>
<td>“LTXIT”</td>
<td>61% ↓</td>
</tr>
<tr>
<td>20%</td>
<td>EXIT</td>
<td>“EXIT”</td>
<td>75% ↓</td>
<td>EXIT</td>
<td>“”</td>
<td>73% ↓</td>
</tr>
</tbody>
</table>
Great, But How Does it Affect Real End-to-End Applications?
Great, But How Does it Affect Real End-to-End Applications?

Without Transition Reduction

- Maximal activity axis
- Low-pass filter
- Extremal-value marking

Reports 19 steps

With Transitions Reduced By 54%

- Maximal activity axis
- Low-pass filter
- Extremal-value marking

Reports 20 steps
Approximate Computation

Chisel: Reliability- and Accuracy-Aware Optimization of Approximate Computational Kernels

Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, Martin Rinard
(OOPSLA 2014 Best Paper Award)
Image Scaling
Image Scaling: Interpolation Function

\[ f(\begin{array}{c}
    \text{Original Image 1} \\
    \text{Original Image 2}
\end{array}) = \text{Scaled Image} \]
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src),
        y = src_location_y(dst_y, src);
    int up    = src[y - 1][x],
        down   = src[y + 1][x],
        left   = src[y][x - 1],
        right  = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
Approximate Hardware Model

Approximate Units (ALUs and Main/Cache Memories)
• May produce incorrect results
• Hardware specification contains savings and reliability
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src),
    y = src_location_y(dst_y, src);

    int up    = src[y - 1][x],
    down    = src[y + 1][x],
    left    = src[y][x - 1],
    right   = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
int interpolation(int@ dst_x, int@ dst_y, int@ src[][[]])
{
    int@ x = src_location_x(dst_x, src),
            y = src_location_y(dst_y, src);

    int@ up   = src[y -. 1][x],
            down = src[y +. 1][x],
            left  = src[y][x -. 1],
            right = src[y][x +. 1];

    int@ val = up +. down +. left +. right;

    return 0.25 * . val;
}
Function and Program Accuracy

Probability $p$ with which interpolation kernel produces a correct pixel.
Function’s and Program’s Accuracy

Produce a correct pixel with probability $> 0.99$
```c
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src),
        y = src_location_y(dst_y, src);

    int up    = src[y - 1][x],
    down   = src[y + 1][x],
    left   = src[y][x - 1],
    right  = src[y][x + 1];

    int val = up +. down +. left +. right;

    return 0.25 *. val;
}
```
int interpolation(int dst_x, int dst_y, int@ src[][]) {
    int x = src_location_x(dst_x, src),
    y = src_location_y(dst_y, src);

    int up    = src[y - 1][x],
    down    = src[y + 1][x],
    left    = src[y][x - 1],
    right   = src[y][x + 1];

    int@ val = up + down + left + right;

    return 0.25 * val;
}
Produce a correct pixel with probability at least \(0.99\)

```c
int interpolation(int dst_x, int dst_y, int src[][[]]) {
    int x = src_location_x(dst_x, src), y = src_location_y(dst_y, src);
    int up = src[y-1][x], down = src[y+1][x], left = src[y][x-1], right = src[y][x+1];
    int val = up + down + left + right;
    return 0.25 * val;
}
```

How to find approximate function with maximum energy savings?
Chisel

Automates placement of
• approximate arithmetic operations
• variables in approximate memory

Approximate function:
  Maximizes energy savings
  Satisfies accuracy specifications
# Accuracy Specification

<table>
<thead>
<tr>
<th>Reliability</th>
<th>Function computes result correctly with probability &gt; 0.99</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute Error</td>
<td>Absolute error of function’s result &lt; 2.0</td>
</tr>
<tr>
<td>Reliability and Absolute Error</td>
<td>Absolute error of function’s result &lt; 2.0 with probability &gt; 0.99</td>
</tr>
</tbody>
</table>
Reliability Specification

Reliability degradation

\[ \text{int} < 0.99 \times R(\Delta x = 0, \Delta y = 0, \Delta \text{src} = 0) > \]

interpolation(int x, int y, int src[][]) ;

The function computes result correctly with probability at least 0.99
Reliability Specification

Reliability degradation

Parameter Reliability

\[
\text{int} < 0.99 \times R(\Delta x = 0, \Delta y = 0, \Delta src = 0) >
\]

interpolation(int x, int y, int src[][][]);

Probability that the parameters have correct values before function starts executing

(facilitates function composition)
Reliability Specification

Reliability degradation

\[ \text{interpolation}(\text{int } x, \text{int } y, \text{int } \text{src}[]): ]

Parameter Reliability

\[ \text{int} < 0.99 \times R(\Delta x = 0, \Delta y = 0, \Delta \text{src} = 0) > \]

\[ \bullet \text{ Reliability factor: } R(\Delta v_1 \leq d_1, \ldots, \Delta v_n \leq d_n) \]

\[ \Delta v \equiv |v_{\text{exact}} - v_{\text{approx}}| \]

Numerical bound
Function Optimization Problem

Find Function Configuration \( q \):

\[
\text{max } \text{EnergySavings} (q) \\
\text{s. t. } \text{Reliability} (q) \geq \text{ReliabilityBound} \\
\text{AbsoluteError} (q) \leq \text{ErrorBound}
\]
Function Configuration

Binary vector $\mathbf{q} = (q_1, q_2, \ldots, q_n)$

Variable Declarations:
- $q_i$ - if 1, variable is stored in approximate memory
  if 0, variable is stored in exact memory

Arithmetic Operations:
- $q_i$ - if 1, the operation is approximate,
  if 0, the operation is exact
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src);
    int y = src_location_y(dst_y, src);

    int up    = src[y - 1][x];
    int down  = src[y + 1][x];
    int left  = src[y][x - 1];
    int right = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
Function Configuration

```c
int interpolation(int q_{\text{dx}} dst_x, int q_{\text{dy}} dst_y, int q_{\text{src}} src[][[]]) {
    int q_x x = src_location_x(dst_x, src);
    int q_y y = src_location_y(dst_y, src);

    int q_{\text{up}} up = src[y - 1][x];
    int q_{\text{down}} down = src[y + 1][x];
    int q_{\text{left}} left = src[y][x - 1];
    int q_{\text{right}} right = src[y][x + 1];

    int q_{\text{val}} val = up + down + left + right;
    return 0.25 * val;
}
```
Each assignment of vector $q$ denotes a different approximate function
Reliability Analysis

- Efficiently represent reliability of all approximate versions of the function

- Construct constraints that describe those approximate functions that satisfy specification
Approximate hardware specification:

- Reliability of arithmetic operations: $r_{op} \in (0, 1]$
- Reliability of memory reads and writes: $r_{rd}, r_{wr} \in (0, 1]$

Analysis:

- Sound static analysis, operates backward
- Constructs symbolic expressions that characterize reliability of traces
Reliability Analysis

Statement

```
return val * 0.25;
```

Exact Statement

val and * exact

Approximate Statement

val and * approximate

Read val

Multiply

Return result
Reliability Analysis

Statement

```
return val * 0.25;
```

Exact Statement

Approximate Statements

<table>
<thead>
<tr>
<th>Statement reliability</th>
<th>1.0</th>
</tr>
</thead>
</table>

| `val and *` | exact |

| `val and *` | approximate |

| `val` | approximate |

| `*` | approximate |

| `r_{rd}` | `r_{times}` |

| `r_{rd}` | `r_{times}` |
Reliability Analysis

Statement

\[ \text{return } \text{val} \times 0.25; \]

Encode approximation choice:

- Variable declaration: \( \text{int}_{val} \text{val}; \)
- Multiplication: \( \text{val} \times 0.25; \)
Reliability Analysis

Statement

```
return val * 0.25;
```

Reliability Expression

\[(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot R(\Delta{val} = 0)\]

Encode approximation choice:

- Variable declaration: \[\text{int}_{q_{val}} \text{val;}\]
- Multiplication: \[\text{val}^{q_{*}} 0.25;\]
Reliability Analysis

Statement

```javascript
return val * 0.25;
```

Reliability Expression

\[
(r_{rd})^{q_{val}} \cdot (r_{times})^{q*} \cdot R(\Delta\text{val} = 0)
\]

Reliability of reading val from either exact or approximate memory:

\[
(r_{rd})^0 = 1.0 \quad (r_{rd})^1 = r_{rd}
\]
Reliability Analysis

Statement

\[ \text{return val * 0.25;} \]

Reliability Expression

\[ (r_{rd})^{q_{val}} \cdot (r_{times})^{q_\ast} \cdot R(\Delta\text{val} = 0) \]

Reliability of either exact or approximate multiplication
Reliability Analysis

Statement

return val * 0.25;

Reliability Expression

\[(\frac{r_{rd}}{q_{val}})^{q_{val}} \cdot (\frac{r_{times}}{q_{*}})^{q_{*}} \cdot R(\Delta val = 0)\]

Probability that previous statements computed val correctly
Interpolation Function

```plaintext
int interpolation(int q_{dstx} dst_x, int q_{dsty} dst_y, int q_{src} src[][[]])
{
    return val * q_{src} * 0.25;
}
```

\[
(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot R(\Delta val = 0)
\]
Interpolation Function

```c
int interpolation(int dstdx dst_x, int dsty dst_y, int src[][][])
{

    int val = up + q1 + q2 + q3;
    return val * 0.25;
}
```
Reliability Expression

Function’s Reliability Expression:

\[ r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \cdot R \left( P_{param} \right) \]

- Probability operations executed reliably (for all approximate versions of the function)
- Probability parameters have correct values at function start
Reliability Constraint

Relate developer’s specification and analysis result:

\[ r_{spec} \cdot R(P_{spec}) \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \cdot R(P_{param}) \]
Reliability Constraint

\[ r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \]

and

\[ R(P_{spec}) \leq R(P_{param}) \]

Can Immediately Solve
Reliability Constraint

\[ r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \]

and

\[ R(P_{spec}) \leq R(P_{param}) \]

\[ P_{spec} \Rightarrow P_{param} \]
Reliability Constraint

\[ r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \]

Denotes approximate function versions that satisfy the developer’s specification
Reliability Constraint

for the optimization problem

\[
\log(r_{spec}) \leq q_1 \cdot \log(r_1) + q_2 \cdot \log(r_2) + \cdots + q_n \cdot \log(r_n)
\]

Denotes approximate function versions that satisfy the developer’s specification
# Reliability and Control Flow

<table>
<thead>
<tr>
<th>Conditionals</th>
<th>Constraints for each program path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis removes redundant constraints</td>
</tr>
<tr>
<td></td>
<td>(most constraints can be removed - OOPSLA ’13)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bounded Loops</th>
<th>Statically known loop bound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis unrolls loop</td>
</tr>
</tbody>
</table>

| Optimization Granularity | Optimize blocks of code instead of individual instructions |
Reliability and Function Calls

Choose between alternative implementations:

\[
\begin{align*}
\text{int } & <1.00*R(\Delta x = 0)> \ f(\text{float } x) \\
\text{int } & <0.99*R(\Delta x = 0)> \ f'(\text{float } x)
\end{align*}
\]

- Reliability degradation: \(0.99^{q_f}\)
- Enables composition of approximate components
Function Optimization Problem

Find Function Configuration \( q \):

\[
\text{max} \quad \text{EnergySavings} \ (q) \\
\text{Reliability} \ (q) \geq \text{ReliabilityBound} \\
\text{AbsoluteError} \ (q) \leq \text{ErrorBound}
\]
Function Optimization Problem

Find Function Configuration $q$:

\[
\begin{align*}
\text{max} & \quad \text{EnergySavings} \ (q) \\
\text{Reliability} \ (q) & \geq \text{ReliabilityBound} \\
\text{AbsoluteError} \ (q) & \leq \text{ErrorBound}
\end{align*}
\]
Energy Savings Analysis

Profile information:
• Collects traces from running representative inputs

Analysis:
• Estimates savings for instructions and variables from traces

\[
\begin{align*}
\text{instruction:} & \quad q_\ell \cdot \text{Count}_\ell \cdot \text{Saving}_{ALU} \\
\text{variable:} & \quad q_m \cdot \text{Size}_m \cdot \text{Saving}_{MEM}
\end{align*}
\]
Energy Savings Analysis

Profile information:
• Collects traces from running representative inputs

Analysis:
• Estimates savings for instructions and variables from traces

Approximate hardware specification:
• Relative savings for operations and memories
• Percentage of system energy that ALU and memory consume

\[
\begin{align*}
    c_{ALU} \sum_{\ell \in \text{Instr}} q_{\ell} \cdot \text{Count}_{\ell} \cdot \text{Saving}_{\text{ALU}} + c_{MEM} \sum_{m \in \text{Var}} q_{m} \cdot \text{Size}_{m} \cdot \text{Saving}_{\text{MEM}}
\end{align*}
\]
Function Optimization Problem

Find Function Configuration $q$:

$\max \text{ EnergySavings (} q \text{)}$

Reliability $(q) \geq \text{ ReliabilityBound}$

AbsoluteError $(q) \leq \text{ ErrorBound}$
Find Function Configuration $q$:

\[
\text{max } \text{EnergySavings} (q) \geq \text{ReliabilityBound}
\]
\[
\text{AbsoluteError} (q) \leq \text{ErrorBound}
\]

Solve using off-the-shelf solvers (we use Gurobi)
Putting the Pieces Together for a Unified Optimization Framework

• Approximate Sensors, Approximate Data
• Approximate Communication
• Approximate Memory, Computation
• Approximate Output
More Research

Approximation for Time/Energy Savings

• **Outlier Detection/Correction** [MIT TR 2014]
• Skipping Tasks [ICS 2006]
• Early Barrier Termination [OOPSLA 2007]
• Loop Perforation [ICSE 2010, FSE 2011, PLDI 2012]
• Dynamic Knobs [ASPLOS 2011]
• Synchronization Elimination [SPLASH 2012, HOTPAR 2013]
• Approximate Parallelization [ACM TOCS 2013]
• Accuracy/Reliability Guarantees [SAS 2011, PLDI 2012]
• Optimal Approximate Map/Fold Programs [POPL 2012]
Even More Research

Resilience Techniques to Obtain Immortal Programs

- Identification/hardening critical data/computation [ISSTA 2010]
- Surviving out of bounds accesses [ACSAC 2004, OSDI 2004]
- Repairing corrupt data structures [OOPSLA 2003, ICSE 2005]
- Eliminating memory leaks [ISMM 2007]
- Escaping infinite loops [ECOOP 2011, OOPSLA 2012]
- Recovering from null pointer/divide by zero errors [PLDI 2014]

Finding Security Vulnerabilities

- At API boundaries [ICSE 2009]
- Integer, buffer overflows [ASPLOS 2015]

Eliminating Security Vulnerabilities

- Input Rectification/Filtering [ICSE 2012, POPL 2014]
- Automatic Code Transfer [PLDI 2015]