IMPROVING PROGRAMS THROUGH SOURCE CODE TRANSFORMATIONS

Dissertation Proposal
Jonathan Dorn
April 19, 2016
Beyond Functional Correctness

• Software development involves \textit{tradeoffs}.
  • “Fast, good, or cheap. Pick any two.”
  • Time vs. memory.
  • Maintainability vs. efficiency.
  • ...


float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalves = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = *( long * ) &y;  // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 );     // what the ?
    y = *( float * ) &i;
    y = y * ( threehalves - ( x2 * y * y ) );  // 1st iteration
    // y = y * ( threehalves - ( x2 * y * y ) );  // 2nd iteration, this can be removed

#ifdef Q3_VM
#ifdef __linux__
    assert( !isnan(y) ); // bk010122 - FPE?
#endif
#endif

    return y;
}
Non-Functional Properties

• “How good” instead of “what” [Paech 2004].
  • “More” or “less;” “higher” or “lower.”

• Difficult to reason about (e.g., security).

• Characterize implementations by how much of a property they possess.
Non-Functional Tradeoffs
Non-Functional Tradeoffs
Quantifying Non-Functionality

• Different metrics for different properties.
  • **Image quality**: RGB distance (e.g., $L^2$), SSIM, EMD.
  • **Runtime**: seconds, speedup/slowdown.
  • **Energy efficiency**: joules, watts.
  • **Maintainability**: bug fix time, defect density.
  • **Correctness**: % error, accuracy, precision, PSNR.
Local Changes

- *Small* changes can have *large* effects.
  - E.g., `bubble_sort(a)` → `quick_sort(a)`.
- Option of *fine-grained* control.
- Program lines, statements, AST nodes.
Proposal Thesis

By applying *local software transformations*, we can select better tradeoffs between *non-functional properties* of existing software artifacts.
The rest of this proposal

- Overview of the proposed research thrusts
  - Visual error and runtime performance
  - Energy usage
  - Coding style
- Proposed research timeline
- Conclusion
The rest of this proposal

• Overview of the proposed research thrusts
  • Visual error and runtime performance
  • Energy usage
  • Coding style
• Proposed research timeline
• Conclusion
Computer Generated Imagery

• 11% of all tickets in 2015 went to computer animated movies.*
  • Does not include live movies with CGI.
• Video games topped $90B in 2015.**

* http://www.boxofficemojo.com/
** http://www.gamesindustry.biz/articles/2015-04-22-gaming-will-hit-usd91-5-billion-this-year-newzoo
Visual Error and Runtime Performance

Original Program

Desired Program

Existing Technique
Hypothesis

• We can apply local changes to the abstract syntax tree of a graphics program to produce an approximation that is:
  • Visually faithful to the original and
  • Efficient to compute.

• Evaluate both image quality and runtime.
Simple Example

```
return (color)
-
floor
+
floor
  p.x
  p.x
  0.5
return (color)
```

floor - p.x

```
(18)
```

floor
Simple Example

\[
\text{return (color)} - \\text{floor} + p.x - 0.5
\]
Simple Example

```
return (color)
- floor + p.x
  p.x 0.5
```

Before

After
Approach

• Transformation: Replace node N with N’.
• Determine replacements offline (manual).
• Genetic search to select nodes to replace.
  • Use image quality as fitness function.
Experimental Setup

• Benchmarks chosen from previous work.
• Record *runtime* and *image quality*.
• Three data points for each benchmark:
  1. Original program.
  2. Baseline “slower but less error” approach.
  3. Best transformed variant from our search.
Runtime Results

Baseline | Our Approach

circles1 | wood | brick | noise2 | circles2 | perlin

Normalized Runtime

0 | 5 | 10 | 15 | 20

step | ridges | pulse | noise1 | checker | wood | brick | noise2 | circles2 | perlin
Image Quality Results

![Image Quality Results Graph]

- Baseline
- Our Approach
Summary

• We can apply local changes to produce programs that:
  • Are significantly *faster* than the baseline approach,
  • Have *less error* than the original program, and
  • Often have *less error* than the baseline.
Outline

• Overview of the proposed research thrusts
  • Visual error and runtime performance
  • Energy usage
  • Coding style
• Proposed research timeline
• Conclusion
Data Center Energy Use

Percentages of US electricity use in a given year

- 2000: 0.8%
- 2005: 1.5%
- 2010: 2.0%

Electricity Use (billion kWh/year)

Genetic Optimization Algorithm

- Local changes to assembly code.
- Tradeoff between reduced energy and relaxed semantics.
  - Validated with test suite.
Genetic Optimization Algorithm

- Local changes to assembly code.
- Tradeoff between reduced energy and relaxed semantics.
- Validated with test suite.
Scaling to Larger Programs

![Graph]

- **Energy Reduction (%)**
  - Y-axis
- **Lines of Code (Assembly)**
  - X-axis
- The graph shows the relationship between energy reduction and lines of code for larger programs.
Hypothesis

By directing the genetic search more effectively and reducing the search space, we can achieve larger energy optimizations faster.

Evaluate both magnitude of optimization and search time.
Intuition

• Optimizations on different paths through the program are likely to be independent.
  • **Combine** optimizations from separate searches.

• Optimizations on frequently executed paths are likely to have larger impact.
  • **Profile** the program to target hot paths.
Algorithm Overview

**INPUT**

**EVALUATE FITNESS**

Eject

Report

**OUTPUT**

**MUTATE**
Optimizing Two Workloads
Option 1: Share Variants During Search
Option 2: Combine Best After Search
Experimental Setup

Benchmarks

• Collect HPC and data center benchmarks.
• Collect multiple workloads for each benchmark.

Baseline: GOA search

1. Only one workload.
2. All workloads in single fitness function.
Metrics for Energy Optimization

- **Energy** measured at the wall.
- **Wall time** before best variant.
  - Latest best variant if combining after search.
- **Fitness evaluations** before best variant.
- Success if searching separately produces larger energy reduction across all workloads.
Preliminary Results

Energy Improvement

- **set 0**
- **set 1**
- **combined**
- **GOA**

Energy Improvement values:
- set 0: 0%
- set 1: 39%
- combined: 40%
- GOA: 10%
Outline

• Overview of the proposed research thrusts
  • Visual error and runtime performance
  • Energy usage
  • Coding style

• Proposed research timeline

• Conclusion
Programmer Time

- Programmer salaries in the U.S. exceed $100B.

- Programmers spend much more time reading code than writing it.

Stylish Code

• *Broad consensus* for standardized coding style.

• *Persistent disagreement* on specifics.
  • E.g., tabs vs. spaces.

• “Every major open source project has its own style guide.” – Google’s style guide.
Beacons

• Indicate likely structure or functionality.
• Semantic or syntactic.
• May vary
  • Between programmers,
  • And over time.

```c
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        ...
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
        ...
    }
}
Beacons

• Indicate likely structure or functionality.

• Semantic or syntactic.

• May vary
  • Between programmers
  • And over time.

```c
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

Possible sort implementation?
Beacons

• Indicate likely structure or functionality.

• Semantic or syntactic.

• May vary
  • Between programmers,
  • And over time.

```c
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        ...
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
        ...
    }
}
```

End of scope.
Classification of Coding Style

- Typographic and Structural [Oman 1988].
  - Typographic: whitespace, line length, identifier length, layout.
  - Structural: modularity, level of nesting, control and information flow.
Classification of Coding Style

• Typographic and Structural [Oman 1988].

  • Typographic: whitespace, line length, identifier length, layout.

  • Structural: modularity, level of nesting, control and information flow.
Hypothesis

• We can apply local changes to the typographic elements of source code to
  • Match a programmer’s expected style and
  • *Improve* their understanding of the code.

• Evaluate time and accuracy on tests of understanding.
Modeling Typographic Style

• *N*-gram language model.
  • Uses previous *n*-1 tokens to predict next token.
  • Learn probabilities from existing code.
  • NATURALIZE framework [Allamanis 2014].
  • Can predict or suggest whitespace.
Similarity of Typographic Style

• Measure similarity of $N$-gram models.
  • $N$-gram models are probability distributions.

• Measure similarity of style-checker rules.
  • Allamanis et al. generate rules from $n$-gram models.
Experimental Setup

Benchmarks
1. Reformat the same code in different ways.
2. Collect similar code from different authors (e.g., textbook examples).

Participants
- Undergraduate student volunteers from upper level electives.
- Amazon Mechanical Turk workers who pass a screening test.
Human Study

1. Identify written style.
   - Participants write code to accomplish simple tasks.
   - E.g., check that a list is sorted.

2. Perform maintenance tasks.
   - Participants answer questions about code examples.
   - E.g., what is the value of x on line 5?
Metrics for Program Understanding

• Collect *similarity* between code participants wrote and the code samples.
• Collect *time* and *accuracy* in answering questions.
• Measure correlation between similarity and time and between similarity and accuracy.
Outline

• Overview of the proposed research thrusts
  • Visual error and runtime performance
  • Energy usage
  • Coding style

• Proposed research timeline

• Conclusion
Conclusion

Enable better tradeoffs between non-functional properties through local software transformations.

2. Energy usage.
3. Coding style.
BACKUP
The Rendering Equation (simplified)

\[ \int P I(p) \cos(p) k(p, w) \, dp \]
### Graphics Primitives

- **Substitutions for built-in expressions.**

- **Assuming locally uniform lighting and Gaussian pixels.**

<table>
<thead>
<tr>
<th>$f(x)$</th>
<th>$\hat{f}(x,w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>$x^2$</td>
<td>$x^2 + w^2$</td>
</tr>
<tr>
<td>$\text{fract}_1(x)$</td>
<td>$\frac{1}{2} - \sum_{n=1}^{\infty} \frac{\sin(2\pi nx)}{\pi n} e^{-2\pi^2 n^2}$</td>
</tr>
<tr>
<td>$\text{fract}_2(x)$</td>
<td>$\frac{1}{2w} \left( \text{fract}^2 \left( x + \frac{w}{2} \right) + \left[ x + \frac{w}{2} \right] \right) - \text{fract}^2 \left( x - \frac{w}{2} \right) - \left[ x - \frac{w}{2} \right]$</td>
</tr>
<tr>
<td>$\text{fract}_3(x)$</td>
<td>$\frac{1}{12w^2} (f'(x-w) + f'(x+w) - 2f'(x))$</td>
</tr>
</tbody>
</table>

\[ f'(t) = 3t^2 + 2\text{fract}^3(t) - 3\text{fract}^2(t) + \text{fract}(t) - t \]

- **$|x|$**
  - $x \text{erf} \frac{x}{w\sqrt{2}} + w \sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2w^2}}$
  - $x - \text{fract}(x)$
  - $\text{floor}(x,w) + 1$
  - $\cos x e^{-\frac{x^2}{2}}$

- **$\text{saturate}(x)$**
  - $\frac{1}{2} \left( x \text{erf} \frac{x}{w\sqrt{2}} - (x-1) \text{erf} \frac{x-1}{w\sqrt{2}} + w \sqrt{\frac{2}{\pi}} \left( e^{-\frac{x^2}{2w^2}} - e^{-\frac{(x-1)^2}{2w^2}} \right) + 1 \right)$

- **$\sin x$**
  - $\sin x e^{-\frac{x^2}{2}}$

- **$\text{step}(a,x)$**
  - $\frac{1}{2} \left( 1 + \text{erf} \frac{x-a}{w\sqrt{2}} \right)$

- **$\text{trunc}(x)$**
  - $\text{floor}(x,w) - \text{step}(x,w) + 1$
Correctness of Composition

• If “A” and “B” are constants.
• If “op” is addition or subtraction.
• If “op” is multiplication and “A” and “B” are multiplicatively separable.
# Graphics Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>step</td>
<td>1</td>
<td>Black and white plane</td>
</tr>
<tr>
<td>ridges</td>
<td>1</td>
<td>fract(x)</td>
</tr>
<tr>
<td>pulse</td>
<td>2</td>
<td>Black and white stripes</td>
</tr>
<tr>
<td>noise1</td>
<td>3</td>
<td>Super-imposed noise</td>
</tr>
<tr>
<td>checker</td>
<td>4</td>
<td>Checkerboard</td>
</tr>
<tr>
<td>circles1</td>
<td>5</td>
<td>Tiled circles</td>
</tr>
<tr>
<td>wood</td>
<td>18</td>
<td>Wood grain</td>
</tr>
<tr>
<td>brick</td>
<td>26</td>
<td>Brick wall</td>
</tr>
<tr>
<td>noise2</td>
<td>28</td>
<td>Color-mapped noise</td>
</tr>
<tr>
<td>circles2</td>
<td>74</td>
<td>Overlapping circles</td>
</tr>
<tr>
<td>perlin</td>
<td>244</td>
<td>Improved Perlin noise</td>
</tr>
</tbody>
</table>
Results: Brick and Wood

<table>
<thead>
<tr>
<th>Target Image</th>
<th>Original Program</th>
<th>Baseline Approach</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Target Image]</td>
<td>![Original Program]</td>
<td>![Baseline Approach]</td>
<td>![Our Approach]</td>
</tr>
</tbody>
</table>

![Target Image] | ![Original Program] | ![Baseline Approach] | ![Our Approach] |
Results: More Complex Procedures

Target Image | Original Program | Baseline Approach | Our Approach
Execution Profiles

• Dynamic profiles:
  • Run workload, sample program counter periodically, estimate probability distribution.

• Static profiles:
  • Estimate distribution directly from code.

• Select lines to mutate from distribution.
Updating Profiles

• Some mutations may change the profile.
  • E.g., remove a hot path.
• Can we update the profile without starting from scratch?
Does Coding Style Matter?

IF $A > B$ THEN
  $S := 1$
ELSE IF $A = B$ THEN
  IF $C > D$ THEN
    $S := 2$
  ELSE
    $S := 3$
  ELSE IF $C > D$ THEN
    $S := 4$
  ELSE IF $C = D$ THEN
    $S := 5$
  ELSE
    $S := 6$

Does Coding Style Matter?

Reproduced from P. W. Oman and C. R. Cook.
**N-gram Models for Coding Style**

```c
void main ( void ) {
    printf ( "hello world\n" ) ;
}
```

\[
P(\langle \text{indent}_2 \rangle | \langle \text{space}_1 \rangle \{ \langle \text{newline} \rangle \})
\]
Evaluation: Human Study 0

**Approach A**

- Present participants with two examples of same code (or similar implementation) in different styles.
- Participants rate the similarity of the two examples.

**Approach B**

- Present participants with three examples of code in different styles.
- Participants select whether the second or third is most similar to the first.
Evaluation: Human Study 0

• Compare similarity model’s rating against human perceptions of similarity.
  • Two examples of same code (or similar implementations) in different typographical styles.
  • Participants rate how similar the examples are.

• Measure human-human and tool-human correlations.