Dynamic Analyses for Floating-Point Precision Tuning

Cindy Rubio-González
Department of Computer Science
University of California, Davis

Ecole Thématique Maths-Info-HPC – Lyon, France, May 2016
Floating-Point Precision Tuning

- Floating-point arithmetic used in variety of domains
- Reasoning about FP programs is difficult
  - Large variety of numerical problems
  - Most programmers are not experts in FP
- Common practice: use highest available precision
  - Disadvantage: more expensive!
- Goal: develop automated techniques to assist in tuning floating-point precision
Example: Mixed Precision

- Consider the problem of finding the arc length of the function

\[ g(x) = x + \sum_{0 \leq k \leq 5} 2^{-k} \sin(2^k x) \]

- Summing for \( x_k \in (0, \pi) \) into \( n \) subintervals

\[ \sum_{k=0}^{n-1} \sqrt{h^2 + (g(x_{k+1}) - g(x_k))^2} \quad \text{with} \quad h = \frac{\pi}{n} \quad \text{and} \quad x_k = k h \]

<table>
<thead>
<tr>
<th>Precision</th>
<th>Slowdown</th>
<th>Result</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>double-double</td>
<td>20X</td>
<td>5.795776322412856</td>
<td>✔</td>
</tr>
<tr>
<td>double</td>
<td>1X</td>
<td>5.795776322413031</td>
<td>✗</td>
</tr>
<tr>
<td>mixed precision</td>
<td>&lt; 2X</td>
<td>5.795776322412856</td>
<td>✔</td>
</tr>
</tbody>
</table>
Example: Mixed Precision

```c
long double g(long double x) {
    int k, n = 5;
    long double t1 = x;
    long double d1 = 1.0L;
    for(k = 1; k <= n; k++) {
        ...
    }
    return t1;
}

int main() {
    int i, n = 1000000;
    long double h, t1, t2, dpri;
    long double s1;
    ...
    for(i = 1; i <= n; i++) {
        t2 = g(i * h);
        s1 = s1 + sqrt(h*h + (t2 - t1)*(t2 - t1));
        t1 = t2;
    }
    // final answer stored in variable s1
    return 0;
}
```

Original Program
Example: Mixed Precision

long double g(long double x) {
  int k, n = 5;
  long double t1 = x;
  long double d1 = 1.0L;
  for(k = 1; k <= n; k++) {
    ...
  }
  return t1;
}

int main() {
  int i, n = 1000000;
  long double h, t1, t2, dp.pi;
  long double s1;
  ...
  for(i = 1; i <= n; i++) {
    t2 = g(i * h);
    s1 = s1 + sqrt(h*h + (t2 - t1)*(t2 - t1));
    t1 = t2;
  }
  // final answer stored in variable s1
  return 0;
}

double g(double x) {
  int k, n = 5;
  double t1 = x;
  float d1 = 1.0f;
  for(k = 1; k <= n; k++) {
    ...
  }
  return t1;
}

int main() {
  int i, n = 1000000;
  double h, t1, t2, dp.pi;
  long double s1;
  ...
  for(i = 1; i <= n; i++) {
    t2 = g(i * h);
    s1 = s1 + sqrtf(h*h + (t2 - t1)*(t2 - t1));
    t1 = t2;
  }
  // final answer stored in variable s1
  return 0;
}
PRECIMONIOUS
[SC’13]
**Precimonious**

“Parsimonious or Frugal with Precision”

Dynamic Analysis for Floating-Point Precision Tuning

- Annotated with error threshold
- If not available, random floating-point values

Less Precision

Speedup

Modified program in executable format
Challenges for Precision Tuning

• Searching efficiently over variable types and function implementations
  – Naïve approach $\rightarrow$ exponential time
    • 19,683 configurations for arc length program ($3^9$)
    • 11 hours 5 minutes
  – Global minimum vs. a local minimum

• Evaluating type configurations
  – Less precision $\rightarrow$ not necessarily faster
  – Based on run time, energy consumption, etc.

• Determining accuracy constraints
  – How accurate must the final result be?
  – What error threshold to use?
Searching: Delta Debugging

• Delta Debugging Search Algorithm [Zeller et al]
  – An approach to debugging
  – Isolates failures systematically
    • Failing test → Isolate the change(s) that introduced failure

• Main idea:
  – We can do better than making a change at the time
  – Start by dividing the change set in two equally sized subsets
  – Narrow the search to the subset that still causes the failure
  – Otherwise, increase the number of subsets

• Efficient search algorithm
  – Average time complexity: $O(n \log n)$
  – Worst case: $O(n^2)$
LCCSearch Algorithm

• Based on the Delta-Debugging Search Algorithm [Zeller et. Al]
• Our definition of a change
  – Lowering the precision of a floating-point variable in the program
    • Example: double x → float x
• Our success criteria
  – Resulting program produces an “accurate enough” answer
  – Resulting program is faster than the original program
• Main idea:
  – Start by associating each variable with set of types
    • Example: x → \{long double, double, float\}
  – Refine set until it contains only one type
• Find a local minimum
  – Lowering the precision of one more variable violates success criteria
Searching for Type Configuration

double precision

single precision
Searching for Type Configuration

- Double precision
  - Correct
- Single precision
  - Incorrect
  - Incorrect
Searching for Type Configuration

- double precision
  - ✔️
- single precision
  - ✘
Searching for Type Configuration

double precision

single precision

✔

✘

✔

✘

✘

✔
Searching for Type Configuration

double precision

single precision
Searching for Type Configuration

double precision

single precision
Searching for Type Configuration

double precision

Proposed configuration

Failed configurations

single precision
Applying Type Configurations

- Automatically generate program variants
  - Reflect type configurations produced by search algorithm

- Intermediate representation
  - LLVM IR

- Transformation rules for each LLVM instruction
  - alloca, load, store, fpext, fptrunc, fadd, fsub, etc.
  - Changes equivalent to modifying the program at the source level

- Able to run resulting modified program
Experimental Setup

• Benchmarks
  o 8 GSL programs
  o 2 NAS Parallel Benchmarks: \textit{ep} and \textit{cg}
  o 2 other numerical programs

• Test inputs
  o Inputs Class A for \textit{ep} and \textit{cg} programs
  o 1000 random floating-point inputs for the rest

• Error thresholds
  o Multiple error thresholds: $10^{-4}, 10^{-6}, 10^{-8},$ and $10^{-10}$
  o User can evaluate trade-off between accuracy and speedup
# Experimental Results

## Original Type Configuration

<table>
<thead>
<tr>
<th>Program</th>
<th>L</th>
<th>D</th>
<th>F</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gaussian</td>
<td>0</td>
<td>52</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>roots</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>polyroots</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rootnewt</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>0</td>
<td>31</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fft</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blas</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>EP</strong></td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>CG</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>arclength</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>simpsons</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

## Proposed Type Configuration

Error threshold: $10^{-4}$

<table>
<thead>
<tr>
<th>L</th>
<th>D</th>
<th>F</th>
<th>Calls</th>
<th># Config</th>
<th>mm:ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>130</td>
<td>37:11</td>
</tr>
<tr>
<td>0</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>201</td>
<td>16:12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>3</td>
<td>1:03</td>
</tr>
<tr>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>336</td>
<td>43:17</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>8</td>
<td>0</td>
<td>61</td>
<td>16:56</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
<td>22</td>
<td>0</td>
<td>325</td>
<td>28:14</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>3</td>
<td>1:16</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>17</td>
<td>0</td>
<td>3</td>
<td>1:06</td>
</tr>
<tr>
<td><strong>0</strong></td>
<td><strong>5</strong></td>
<td><strong>8</strong></td>
<td><strong>4</strong></td>
<td><strong>111</strong></td>
<td><strong>23:53</strong></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>30</td>
<td>3</td>
<td>44</td>
<td>0:57</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>33</td>
<td>0:40</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>0:07</td>
</tr>
</tbody>
</table>
Speedup for Error Threshold $10^{-4}$

Maximum speedup observed across all error thresholds: 41.7%
DEMO
BLAME ANALYSIS
[ICSE’16]
BLAME ANALYSIS

• Goal: alleviate scalability limitations of existing search-based FP precision tuning approaches
  – Reduce number of executions/transformations
  – Perform local, fine-grained isolated transformations

• Executes the program only once while performing shadow execution

• Focuses on accuracy, not performance

• Best results when used to prune the search space of PRECIMONIOUS
High-Level Approach

```c
int main() {
    double a = 1.84089642;
    double res, t1, t2, t3, t4;
    double r1, r2, r3;

    t1 = 4*a;
    t2 = mpow(a, 6, 2);
    t3 = mpow(a, 4, 3);
    t4 = mpow(a, 1, 4);

    // res = a^4 - 4a^3 + 6a^2 - 4a + 1
    r1 = t4 - t3;
    r2 = r1 + t2;
    r3 = r2 - t1;
    res = r3 + 1;

    printf("res = %.10f\n", res);
    return 0;
}
```

Two main components:

1. **Shadow execution** runs the program both in single and double precision
2. **Blame analysis** determines precision requirements for each program instruction
Shadow Execution

- Floating-point value associated with shadow value
- Shadow value defined as
  
  | Shadow Value | double | float |
  |
- Shadow execution computes on shadow values
- Maintains shadow memory and label map

\[
M : A \rightarrow S \\
LM : A \rightarrow L
\]

- \(A\): set of all memory addresses
- \(S\): set of all shadow values
- \(L\): set of all instruction labels
Shadow Execution in Action

\[ z = x - y; \quad // \text{label } l1 \]

FSubShadow(x, y, z, l1); // instrumentation

1. **SHADOW MEMORY**
   - Retrieve shadow values for operands
   - \( x_{\text{shadow}} \)
   - \( y_{\text{shadow}} \)

2. **SHADOW MEMORY**
   - \( x_d - y_d \)
   - \( x_s - y_s \)
   - Update shadow value for \&z
   - \( z_{\text{shadow}} \)

3. **LABEL MAP**
   - Instruction \( l1 \) last to compute value \( z \)
   - Update label for \&z

Instruction \( l1 \) last to compute value \( z \)
BLAME ANALYSIS - Local Precision

• Determines for each instruction $i$ and each precision $p$ the precision requirements for the operands so that $i$ has at least precision $p$

• We consider various precisions $p$
  – $f1, db_4, db_6, db_8, db_{10}, db$
  – Example: computing $db_8$ from $db$ value

$$
\begin{array}{cccccccccc}
0 & . & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 \\
\end{array}
$$

db

$$
\begin{array}{cccccccccc}
0 & . & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 \\
\end{array}
$$

db_8

8 significant digits
# Example – Local Precision

**Instruction:** \( z = x - y \)

**Precision:** \( \text{db}_8 \)

z’s \( \text{db}_8 \) value: \(-0.4999999887\)

z’s \( \text{db}_8 \) target value: \(-0.499999988\)

Assume: \( P = \{ \text{fl}, \text{db}_8, \text{db} \} \)

<table>
<thead>
<tr>
<th>Precision</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>(fl, fl)</td>
<td>6.8635854721</td>
<td>7.3635854721</td>
<td>-0.500000000000</td>
</tr>
<tr>
<td>(fl, ( \text{db}_8 ))</td>
<td>6.8635854721</td>
<td>7.3635856000</td>
<td>-0.5000001279</td>
</tr>
<tr>
<td>(fl, db)</td>
<td>6.8635854721</td>
<td>7.3635856800</td>
<td>-0.5000002079</td>
</tr>
<tr>
<td>(( \text{db}_8 ), fl)</td>
<td>6.8635856000</td>
<td>7.3635854721</td>
<td>-0.49999998721</td>
</tr>
<tr>
<td>(( \text{db}_8 ), ( \text{db}_8 ))</td>
<td>6.8635856000</td>
<td>7.3635856000</td>
<td>-0.500000000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(db, db)</td>
<td>6.8635856913</td>
<td>7.3635856800</td>
<td>-0.49999999887</td>
</tr>
</tbody>
</table>

Operators require precision \( (\text{db}, \text{db}) \) for result to be at least \( \text{db}_8 \)
BLAME ANALYSIS - Global Precision

Propagate precision requirements given target

Find dependencies, and choose precision requirements

Last, find variables that can be allocated in single precision
Experimental Evaluation

• Evaluation in different settings
  – **BLAME analysis** by itself
  – **BLAME analysis + Precimonious (B+P)**
  – Compared to **Precimonious (P)**

• Benchmarks
  – 2 NAS Parallel Benchmarks (ep and cg)
  – 8 GSL programs

• Test inputs
  – Inputs Class A for ep and cg programs
  – 1000 random floating-point inputs for the rest

• Error thresholds
  – Multiple error thresholds: $10^{-4}$, $10^{-6}$, $10^{-8}$, and $10^{-10}$
  – User can evaluate trade-off between accuracy and speedup
Analysis Performance (I)

- **BLAME ANALYSIS** introduces 50x slowdown
- B+P is faster than P in 31 out of 39 experiments

<table>
<thead>
<tr>
<th>Program</th>
<th>Speedup</th>
<th>Program</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>22.48x</td>
<td>sum</td>
<td>1.85x</td>
</tr>
<tr>
<td>gaussian</td>
<td>1.45x</td>
<td>fft</td>
<td>1.54x</td>
</tr>
<tr>
<td>roots</td>
<td>18.32x</td>
<td>blas</td>
<td>2.11x</td>
</tr>
<tr>
<td>polyroots</td>
<td>1.54x</td>
<td>ep</td>
<td>1.23x</td>
</tr>
<tr>
<td>rootnewt</td>
<td>38.42x</td>
<td>cg</td>
<td>0.99x</td>
</tr>
</tbody>
</table>

Combined analysis time is 9x faster on average, and up to 38x in comparison with PRECIMONIOUS alone.
Analysis Performance (II)

- B+P is slower in 8 out of 39 experiments
- Example: different search path makes P more expensive

![Graph showing time vs. error threshold for B+P and P]
Analysis Performance (III)

- B+P is slower in 8 out of 39 experiments
- Example: combined analysis more expensive than P
• **BLAME ANALYSIS** identifies at least 1 float variable in all 39 experiments

• Overall, **BLAME ANALYSIS** removes 40% of the variables from the search space (117 out of 293 variables), median 28%

• B+P and P agree on 28 out of 39 experiments

• B+P is slightly better in remaining 11 experiments
## Analysis Results (II)

<table>
<thead>
<tr>
<th>Program</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>gaussian</td>
<td>56</td>
<td>0</td>
</tr>
<tr>
<td>roots</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>polyroots</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>rootnewt</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>fft</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>blas</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>ep</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>cg</td>
<td>32</td>
<td>0</td>
</tr>
</tbody>
</table>

### Original Type Configuration

### Proposed Type Configurations

**Error threshold: 10^{-4}**

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>F</th>
<th>B</th>
<th>D</th>
<th>F</th>
<th>B+P</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>1</td>
<td>25</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gaussian</td>
<td>54</td>
<td>2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>roots</td>
<td>1</td>
<td>15</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>polyroots</td>
<td>10</td>
<td>21</td>
<td>10</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rootnewt</td>
<td>1</td>
<td>13</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>24</td>
<td>10</td>
<td>11</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fft</td>
<td>16</td>
<td>6</td>
<td>0</td>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blas</td>
<td>1</td>
<td>16</td>
<td>0</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ep</td>
<td>42</td>
<td>3</td>
<td>42</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cg</td>
<td>26</td>
<td>6</td>
<td>2</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- No configuration speeds up the program
- **BLAME ANALYSIS** finds good configuration
- B+P finds a better configuration

Many variables lowered to single precision
## Analysis Results (II)

### Original Type Configuration

<table>
<thead>
<tr>
<th>Program</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>gaussian</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>roots</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>polyroots</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>rootnewt</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>fft</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>blas</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>ep</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>cg</td>
<td>32</td>
<td>0</td>
</tr>
</tbody>
</table>

### Proposed Type Configurations

Error threshold: $10^{-4}$

<table>
<thead>
<tr>
<th>Program</th>
<th>B</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>gaussian</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>roots</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>polyroots</td>
<td>10</td>
<td>21</td>
<td>x</td>
</tr>
<tr>
<td>rootnewt</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>sum</td>
<td>11</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>fft</td>
<td>0</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>blas</td>
<td>0</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>ep</td>
<td>42</td>
<td>3</td>
<td>x</td>
</tr>
<tr>
<td>cg</td>
<td>2</td>
<td>30</td>
<td>2</td>
</tr>
</tbody>
</table>

### Program speedup up to 40%
Collaborators

University of California, Berkeley

Cuong Nguyen  
Diep Nguyen  
Ben Mehne  
James Demmel  
William Kahan  
Koushik Sen

Lawrence Berkeley National Lab

Costin Iancu  
David Bailey  
Wim Lavrijsen

Oracle

David Hough