Dynamic Analyses for Floating-Point Precision Tuning

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

ICERM Workshop, May 7th 2020

Supported by DE-SC0008699, DE-SC0010200, HR0011-12-2-0016, and ASPIRE Lab industrial sponsors and affiliates Intel, Google, Hewlett-Packard, Huawei, LGE, NVIDIA, Oracle, and Samsung
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
  - PRECIMONIOUS
  - BLAME ANALYSIS
Example: Arc Length

- 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

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

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

Program

def g(x):
    k, n = 5
    t1 = x
    d1 = 1.0L
    for k = 1; k <= n; k++
        ...
    return t1

int main() {
    int i, n = 1000000
    h, t1, t2, dpdi
    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
}
Example: Arc Length

```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, dppi;
    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;
}
```

```c
int main() {
    int i, n = 1000000;
    double h, t1, t2, dppi;
    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;
}
```

Original Program

```c
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, dppi;
    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;
}
```

Mixed Precision Program
Dynamic Analysis for Floating-Point Precision Tuning

Annotated with error threshold

Precimionious

"Parsimonious or Frugal with Precision"

SOURCE CODE

TEST INPUTS

Modified program in executable format

Less Precision

Speedup

TYPE CONFIGURATION

MODIFIED PROGRAM
Challenges for Precision Tuning

• Searching efficiently over variable types and function implementations
  – Naïve approach → 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 → 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?

Automated

Specified by the user
Search 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 a 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

single precision
Searching for Type Configuration

double precision

single precision

X

X

X

X
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

Proposed configuration

Failed configurations

…
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
Implementation Details

Original Program → LLVM Bitcode (Clang)

Create Search Type Configuration → LLVM Passes (C++)

Search Configuration → JSON text file

Search Algorithm → A Type Configuration

Python

Program Transformation → Tuned Program

Original Program → LLVM Passes (C++)

Proposed Type Configuration → JSON text file
Experimental Setup

• Benchmarks
  o 8 GSL programs
  o 2 NAS Parallel Benchmarks: *ep* and *cg*
  o 2 other numerical programs

• Test inputs
  o Inputs Class A for *ep* and *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>
</tbody>
</table>

### Proposed Type Configuration

<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>
</tbody>
</table>

### GSL and NAS

- **GSL**
- **NAS**

**Error threshold**: $10^{-4}$
Speedup for Error Threshold $10^{-4}$

Maximum speedup observed across all error thresholds: 41.7%
Summary so far

• Devised a dynamic analysis for tuning the precision of floating-point programs

• Implemented in publicly available tool named Precimonious
  https://github.com/ucd-plse/precimonious

• Initial evaluation on 12 programs shows encouraging speedups of up to 41%
Limitations

• Type configurations rely on program inputs tested
  – No guarantees if worse conditioned input
  – Additional experiments to assess inputs used in evaluation

• Getting trapped in local minimum

• Analysis scalability
  – Approach does not scale to tune long-running applications
  – Need to reduce search space, and reduce number of runs
  – Rest of this talk: Blame Analysis

• Analysis effectiveness
  – Exploit relationships among variables
  – See our more recent work on HiFPTuner [ISSTA’18]
    https://github.com/ucd-plse/HiFPTuner
BLAME ANALYSIS

**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 on performance
- Best results observed when used to prune the search space of PRECIMONIOUS
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;
}
Shadow Execution

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

### Shadow Memory

- $M: A \rightarrow S$
  - $A$: set of all memory addresses
  - $S$: set of all shadow values

### Label Map

- $LM: A \rightarrow L$
  - $L$: set of all instruction labels
Shadow Execution in Action

\[ z = x - y; \quad // \text{label l1} \]
FSubShadow(x, y, z, l1); \quad // instrument
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 $p$
  - $f1, \text{db}_4, \text{db}_6, \text{db}_8, \text{db}_{10}, \text{db}$
  - Example: computing $\text{db}_8$ from $\text{db}$ value

\[
\begin{array}{cccccccccccc}
0 & . & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
0 & . & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 3 \\
\end{array}
\]

8 significant digits
Example – Local Precision

Instruction: \( z = x - y \)

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

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

z’s \( \text{db}_8 \) target value: \(-0.49999998\)

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>-0.5000000000</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, ( \text{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>-0.4999998721</td>
</tr>
<tr>
<td>(( \text{db}_8 ), ( \text{db}_8 ))</td>
<td>6.8635856000</td>
<td>7.3635856000</td>
<td>-0.5000000000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(( \text{db} ), ( \text{db} ))</td>
<td>6.8635856913</td>
<td>7.3635856800</td>
<td>-0.4999999887</td>
</tr>
</tbody>
</table>

Operands 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

• Same inputs and error thresholds as Precimonious
Analysis Performance

- **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>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>22.48x</td>
</tr>
<tr>
<td>gaussian</td>
<td>1.45x</td>
</tr>
<tr>
<td>roots</td>
<td>18.32x</td>
</tr>
<tr>
<td>polyroots</td>
<td>1.54x</td>
</tr>
<tr>
<td>rootnewt</td>
<td>38.42x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>1.85x</td>
</tr>
<tr>
<td>fft</td>
<td>1.54x</td>
</tr>
<tr>
<td>blas</td>
<td>2.11x</td>
</tr>
<tr>
<td>ep</td>
<td>1.23x</td>
</tr>
<tr>
<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 Results (I)

- **BLAME ANALYSIS** identifies at least 1 float variable in each of the 39 experiments

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

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

- B+P is slightly better in remaining 11 experiments
### 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>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>

#### Proposed Type Configurations

Error threshold: $10^{-4}$

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

**GSL**

- Many variables lowered to single precision

**NAS**

- No configuration speeds up the program
- BLAME ANALYSIS finds good configuration
- B+P finds a better configuration
## 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>gamma</td>
<td></td>
<td></td>
</tr>
<tr>
<td>roots</td>
<td></td>
<td></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}**

**B**

<table>
<thead>
<tr>
<th>Program</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>bessel</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>polyroots</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>rootnewt</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>sum</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>fft</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>blas</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>ep</td>
<td>42</td>
<td>3</td>
</tr>
<tr>
<td>cg</td>
<td>26</td>
<td>6</td>
</tr>
</tbody>
</table>

**B+P**

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

**P**

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

**P does not find a configuration**

Program speedup up to 40%
Limitations

- **BLAME ANALYSIS** does not guarantee accurate results for all possible inputs
- **BLAME ANALYSIS** does not take performance into consideration
- Program transformations still limited to changing variable types
Summary

- **PRECIMONIOUS**: configurations lead to speedup, but requires running program numerous times during search
- **BLAME ANALYSIS**: successful at lowering precision, but does not guarantee speedup, single run of the program while performing shadow execution
- Largest impact when combining the analyses
  - Combined analysis time is 9x faster on average, and up to 38x in comparison with PRECIMONIOUS alone
  - Type configurations lead to speedup of up to 40%
- The dynamic analyses presented today represent a step towards more scalable FP precision tuning
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