Precimonious
Tuning Assistant for Floating-Point Precision

Ignacio Laguna, Harshitha Menon, Tristan Vanderbruggen
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Ganesh Gopalakrishnan
University of Utah

Cindy Rubio-González
University of California at Davis

http://fpanalysisstools.org/

This work was supported by through the X-Stack program funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under collaborative agreement SC0008699, NSF grant 1750983, and a gift from Oracle.
Floating-Point Precision Tuning

• Floating-point (FP) arithmetic used in variety of domains

• Reasoning about FP programs is difficult
  o Large variety of numerical problems
  o Most programmers are not experts in FP

• Common practice: use highest available precision
  o Disadvantage: more expensive!

• Goal: automated technique to assist in tuning floating-point precision
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

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

<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>
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, dp;  // dp stands for double precision
    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;
}
```

Mixed Precision Program
Precimonious

“Parsimonious or Frugal with Precision”

Dynamic Analysis for Floating-Point Precision Tuning

Annotated with error threshold

Modified program in executable format

http://fpanalysistools.org/
Challenges for Precision Tuning

- Searching efficiently over variable types and function implementations
  - Naïve approach -> exponential time
  - 19,683 configurations for arclength program \(3^9\)
  - 11 hours 5 minutes
  - Global minimum vs. Local minimum

- Evaluating type configurations
  - Less precision not necessarily faster
  - Based on runtime, energy consumption, etc.

- Determining accuracy constraints
  - How accurate must the final result be?
  - What error threshold to use?
Precimonious Search Algorithm

● Based on Delta Debugging Algorithm (TSE’02)

● Our definition of a change
  ○ Lowering the precision of a floating-point variable in the program
    ▪ Example: double $x$ -> float $x$

● Main idea
  ○ We can do better than making a change at the time
  ○ Start by dividing the change set into two equally sized subsets
  ○ Narrow the search to the subset that satisfies the success criteria
  ○ Otherwise, increase the number of subsets

● Our success criteria
  ○ Resulting program produces an answer within the given error threshold
  ○ Resulting program is faster than original program

● Find local minimum
  ○ Lowering the precision of any one more variable violates the success criteria
Searching for Type Configuration

double precision

single precision
Searching for Type Configuration

double precision

single precision

http://fpanalysisertools.org/
Searching for Type Configuration

double precision

single precision

http://fpanalyistools.org/
Searching for Type Configuration

double precision

single precision

X

http://fpanalysistools.org/
Searching for Type Configuration

double precision

single precision

http://fpanalysis tools.org/
Searching for Type Configuration

double precision

single precision

http://fpanalysistools.org/
Searching for Type Configuration

double precision

Proposed configuration

Failed configurations

single precision

http://fpanalyistools.org/
Applying Type Configuration

- Automatically generate program variants
  - Reflect type configurations produced by the algorithm
- Intermediate representation
  - LLVM IR
- Transformation rules for each LLVM instruction
  - alloca, load, store, fadd, fsub, fpext, fpintrunc, etc.
  - Changes equivalent to modifying the program at the source level
  - Clang plugin to provide modified source code (not discussed today)
- Able to run resulting modified program
  - Evaluate type configuration: accuracy & performance
Limitations

● Type configurations rely on inputs tested
  ○ No guarantees if worse conditioned input
  ○ Could be combined with input generation tools (e.g., S3FP)

● Getting trapped in local minimum

● Analysis scalability
  ○ Approach does not scale well for long-running applications
  ○ Need to reduce search space and reduce number of runs
  ○ Check out our follow up work on Blame Analysis (ICSE’16)

● Analysis effectiveness
  ○ Approach does not exploit relationship among variables
  ○ Check out our follow up work on HiFPTuner (ISSTA’18)
Source code available:
https://github.com/corvette/precimonious

Questions?
Exercises
Exercises with Precimionious

1. Run Precimionious on sample program funarc
2. Run Precimionious on sample program simpsons

Directory Structure

/Module-Precimionious
   |--./exercise-1
   |   |--./exercise-2
Exercise 1
Step 1: Build Precimonious

- Open setup.sh file
- Precimonious uses LLVM and is built using scons
- Execute:
  - $ ./setup.sh

Success building and running tests

http://fpanalysistools.org/
Step 2: Annotate Program (already done)

- Execute:
  - `$ cd exercise-1`
  - `$ ls`

- Open `funarc.c` file

  **Accuracy logging & checking**

  ```c
  threshold = result* pow(10, epsilon);
  // cov_spec_log("spec.cov", threshold, 1, result);
  cov_log("result", "log.cov", 1, result);
  cov_check("log.cov", "spec.cov", 1);
  
  FILE* file = fopen("score.cov", "w");
  fprintf(file, "%ld\n", diff);
  fclose(file);
  
  /***** END PRECIMONIOUS ACCURACY and PERFORMANCE LOGGING ******/
  ```

  **Performance logging**

The program we will tune:

- `Makefile`
- `funarc.c`
- `reference`
- `run-dependencies.sh`
- `spec.cov`
- `run-analysis.sh`
- `run-config.sh`
Step 3: Compile Program with Clang

- Execute:
  - $ make clean
  - $ make

- Creates LLVM bitcode file and optimized executable for later use
Step 4: Run Analysis on Program

- Execute:
  - $ ./run-analysis.sh funarc

Sample output:

Type changes are listed for each explored configuration

Suggested type configuration

http://fpanalysistools.org/
Step 4: Run Analysis – Configuration File

- Open config_funarc.json
- Original type configuration
Step 4: Run Analysis – Search File

- Open search_funarc.json
- Search space file
- To exclude functions edit exclude.txt
- To exclude variables edit exclude_local.txt
- Or you can directly edit search file prior to analysis
Step 4: Run Analysis – Output Files

- Execute:
  - $ cd results
  - $ ls

http://fpanalysisistools.org/
Step 4: Run Analysis – Output Files

- Open dd2_valid_funarc.bc.json: suggested configuration file in JSON format
- Open dd2_diff_funarc.bc.json: summary of type changes

```plaintext
localVar: d1 at fun longdouble -> float
localVar: s1 at main longdouble -> double
localVar: t1 at main longdouble -> double
localVar: t2 at main longdouble -> double
localVar: h at main longdouble -> double
localVar: dpdi at main longdouble -> float
call: acos at mainacos -> acosf
call: sqrt at mainsqrt -> sqrtf
```
**Step 5: Apply Result Configuration & Compare Performance**

- **Execute:**
  - $./run-config.sh funarc

- **Execute:**
  - $ time ./original_funarc.out
  - $ time ./tuned_funarc.out

Run the following to compare performance:

time ./original_funarc.out
time ./tuned_funarc.out

http://fpanalysistools.org/
Exercise 2
Exercise 2: Run Precimonious on simpsons program

- Open exercise-2/simpsons.c to see annotated program
- Execute:
  - cd ../exercise-2
  - make clean
  - make
  - ./run-analysis.sh simpsons
  - ./run-config.sh simpsons
- Open results/dd2_valid_simpsons.bc.json to see configuration in JSON format
- Open results/dd2_diff_simpsons.bc.json to see difference between original program and proposed configuration
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

http://fpanalysistools.org/
Source code available:
https://github.com/corvette/precimonious

Questions?