Artifact Scorecard

Ran Wang, Daming Zou, Xinrui He, Yingfei Xiong, Lu Zhang, Gang Huang
Key Laboratory of High Confidence Software Technologies (Peking University), MoE
Institute of Software, EECS, Peking University, Beijing, 100871, China
{wangrancs,zoudm,hexinrui,xiongyf,zhanglucs,hg}@pku.edu.cn

1. INSIGHTFUL

Many research efforts have been devoted into determining [2, 9, 1], verifying [8, 6], and optimizing [5, 7] the accuracy of floating-point programs. To implement these approaches, we need to know the specification of a program. Typically, these approaches take a *precision-unspecific semantics* to interpret the program to get the specification: floating-point variables are interpreted as real numbers, floating-point operations are interpreted as real arithmetic operations.

However, a program usually contains *precision-specific operations*, such direct interpretation may lead to wrong results. Precision-specific operations are operations that are designed to work on a specific precision. For example, let us consider the following program.

```
1:  double round(double x) {
2:    double n = 6755399441055744.0;
3:    return (x + n) - n;
4:  }
```

The goal of the program is to round `x` to an integer and return the result. The constant `n` is a "magic number" working only for double-precision floating-point numbers, which precisely rounded off the decimal digits of `x`. However, if we directly interpret the program as real computations, the whole procedure would be interpreted as computing $x+n-n$, which is mathematically equivalent to x. When existing approaches follow such a specification, they would produce undesirable results.

In the paper we propose a lightweight approach for detecting precision-specific operations. The evaluation shows that our detection approach has high precision and recall. We also found false positives and potential false positives in existing results. We further propose a fixing approach for supporting precision-specific operations in precision tuning. Precision tuning is a commonly used technique for getting the accurate results. The evaluation shows that our fixing leads to more accurate results in the presence of precision-specific operatinos.

2. USEFUL

Our artifact is a replication package for our experiments. This artifact serves several important purposes. First, it helps reproduce our experiments and cross-validate our results. Second, the code can be modified to be applied on other subjects, for further understanding of the problem on a broader scope. Third, the artifact presents the precision-specific operations found in the C math library, and other tools and approaches may directly use this information to improve their performance. Fourth, researchers may further integrate our code to detect more precision-specific operations in more programs, improving existing tools and approaches in a broader scope. Fifth, the code for our fixing approach can be integrated into those approaches that rely on precision tuning, leading to more accurate results in precision tuning.

3. USABLE

The artifact consists of tutorials, a configured virtual machine image, all experimental scripts and data, and a list of genuine precision-specific operations we found. The tutorials elaborate on subjects selection procedure, how to set up the configuration for the experiment, subjects instrumentation, how to rerun experiment, and other details of the scripts and data. A virtual machine image helps researchers to escape from setting up the configuration. Experimental scripts can be directly executed on the virtual machine. The artifact is published in MIT license (with dependencies in their original licenses) and can be downloaded[1].

4. REFERENCES

[1] F. Benz, A. Hildebrandt, and S. Hack. A dynamic program analysis to find floating-point accuracy problems. In *Proc. PLDI*, pages 453–462, 2012.

[2] E. Darulova and V. Kuncak. Sound compilation of reals. In *POPL*, pages 235–248, 2014.

[3] E. Goubault and S. Putot. Static analysis of numerical algorithms. In *Proc. SAS*, pages 18–34, 2006.

[4] P. Panchekha, A. Sanchez-Stern, J. R. Wilcox, and Z. Tatlock. Automatically improving accuracy for floating point expressions. In *PLDI*, pages 1–11, 2015.

[5] S. Putot, E. Goubault, and M. Martel. Static analysis-based validation of floating-point computations. In *Numerical software with result verification*, pages 306–313. 2004.

[6] D. Zou, R. Wang, Y. Xiong, L. Zhang, Z. Su, and H. Mei. A genetic algorithm for detecting significant floating-point inaccuracies. In *ICSE*, pages 529–539, 2015.

[1] https://github.com/floatfeather/PSO/tree/master/artifact

Detecting and Fixing Precision-Specific Operations for Measuring Floating-Point Errors

Ran Wang, Daming Zou, Xinrui He, Yingfei Xiong, Lu Zhang, Gang Huang
Key Laboratory of High Confidence Software Technologies (Peking University), MoE
Institute of Software, EECS, Peking University, Beijing, 100871, China
{wangrancs,zoudm,hexinrui,xiongyf,zhanglucs,hg}@pku.edu.cn

1. MATERIALS

The artifact is replication package for our experiment, consisting of the following materials.

1. A tutorial on subject selection, experiment configuration and how to run experimental scripts.
2. A virtual machine image with all tools and subjects installed.
3. Open source tools the experiment depends on, such as FPDebug, MPFR.
4. Experimental scripts that instrument subjects, detect and fix precision-specific operations in subjects and analyze the results.
5. Experimental data like instrumented subjects, discovered precision-specific operations.

The artifact is published under MIT license (the open source tools and subjects are published under their original licenses) and can be downloaded[1].

2. TUTORIAL

The tutorial provides step by step guidance for reproducing our experiments. The tutorial covers two scenario: the virtual machine and manual installation on a new machine. The former is much simpler and is recommended. The later involves the installation and configuration of several tools, and the change of scripts, and is only recommended if you plan to modify our code for new tools and experiments.

3. VIRTUAL MACHINE IMAGE

The virtual machine has tools installed (FPDebug, GMP, MPFR) and configured. The two versions of instrumented GLIBC are also installed. The scripts and data are also placed in the virtual machine so that reruning the experiment only needs a few commands, which are described in the tutorial.

4. TOOLS

Our experiments depend on several open source tools. We use FPDebug [1] to tune precision. FPDebug further relies on a modified MPFR, a C library for multiple-precision floating-point computations with correct rounding, for more accurate computations. LLVM and CLANG are used to instrument the subjects. All these tools are installed on the virtual machine except LLVM and CLANG.

[1] https://github.com/floatfeather/PSO/tree/master/artifact

5. SCRIPTS

We provide scripts to run the experiment, including instrumentation, the detection and fixing approach, and analysis for the results. Here we briefly introduce how to replicate our experiment in the virtual machine.

The experiments are implemented as two steps. The first is to detect and fix precision-specific operations. The second is to analyze the results. Invoking the following command would perform the first step.

```
cd /home/artifact/work/exe-art
./run_2.sh
```

The second step is implemented as four different scripts. The first script analyzes the precision and recall of the detection approach, and can be invoked using the following commands.

```
cd /home/artifact/work/exe-art
g++ ErrorProcess.cpp -o ErrorProcess
./ErrorProcess
```

The rest three scripts analyze respectively for the three types of fixing in our experiment, and can be invoked by the following commands.

```
cd /home/artifact/work/base-art
./run_auto.sh
./run_manual.sh
./run_fixlast.sh
```

The results will be stored in `result_auto_E.csv`, `result_manual.csv` and `result_fixlast.csv`. `E` stands for the parameter `E` used in our detection approach.

6. DATA

The data include subject functions and other data needed for the experiments, such as accurate outputs for each subject function. The data also include a list of precision-specific operations found in the double version of the GNU C math library. The turotial gives a more detailed description of the data.

7. REFERENCES

[1] F. Benz, A. Hildebrandt, and S. Hack. A dynamic program analysis to find floating-point accuracy problems. In *Proc. PLDI*, pages 453–462, 2012.