Detecting Inaccuracies in Floating-Point Programs

Yingfei Xiong, Peking University

Collaboration with

Daming Zou, Ran Wang, Zhendong Su, Xinrui He,

Lu Zhang, Gang Huang, Hong Mei

Floating-Point Inaccuracy

$$\underbrace{0.1 + 0.1 + 0.1 + \ldots\ldots + 0.1}_{10000 \text{ times}}$$

Code:

```
float x = 0, a = 0.1;
for(int i = 0; i < 10000; i++) {
    x = x + a;
}
printf("%.6f", x);
```

Output: 999.902893

Floating-Point Inaccuracy

$$\underbrace{0.1 + 0.1 + 0.1 + \ldots\ldots + 0.1}_{10000 \text{ times}}$$

Code:

```
double x = 0, a = 0.1;
for(int i = 0; i < 10000; i++) {
    x = x + a;
}
printf("%.6lf", x);
```

Output: 1000.000000

问题背景

- 浮点误差会导致严重的后果。
- 在海湾战争中，爱国者导弹由于累积的浮点误差拦截失败，造成了28人丧生。

Measure Errors

Absolute error:

$$Error_{abs} = \left|x_{ideal} - x_{fp}\right|$$

Relative error:

$$Error_{rel} = \left|\frac{x_{ideal} - x_{fp}}{x_{ideal}}\right|$$

$\boldsymbol{x_{ideal}}$?

Precision Tuning

Code:

```
float x = 0, a = 0.1;
for(int i = 0; i < 10000; i++) {
    x = x + a;
}
printf("%.6f", x);
```

$\Rightarrow$ $x_{orignal} = 999.902893$

$\Downarrow$ Raise precision

Code:

```
double x = 0, a = 0.1;
for(int i = 0; i < 10000; i++) {
    x = x + a;
}
printf("%.6lf", x);
```

$\Rightarrow$ $x_{high} = 1000.000000$

Measure Errors with Precision Tuning

Absolute error:

$$Error_{abs} = \left| x_{ideal} - x_{fp} \right| \approx \left| x_{high} - x_{original} \right|$$

Relative error:

$$Error_{rel} = \left| \frac{x_{ideal} - x_{fp}}{x_{ideal}} \right| \approx \left| \frac{x_{high} - x_{original}}{x_{high}} \right|$$

Precision-Unspecific Semantics

- Floating-point variables → Real numbers
- Floating-point operations → Real number operations
- Technique: precision tuning

On-the-fly detection of instability problems in floating-point program execution
A dynamic program analysis to find floating-point accuracy problems
Efficient search for inputs causing high floating-point errors
Trustworthy numerical computation in scala
Automatically improving accuracy for floating point expressions
A genetic algorithm for detecting significant floating-point inaccuracies

Assumption: High precision usually produce more accurate output

Is it true?

Example

A code piece simplified from `exp` function in the GNU C library:

```
double x = 3.7;
double n = 6755399441055744.0;
double y = (x + n) - n;
```

Answer from computer: `y = 4` ✓

Raise precision to long double:

Answer from computer: `y = 3.7002` ✗

Example

A code piece simplified from `exp` function in the GNU C library:

```
1:   double x = 3.7;
2:   double n = 6755399441055744.0;
3:   double y = (x + n) - n;
```

The goal of this code is to round x to the nearest integer.
n is a "magic number" specially designed for double precision. ($n = 1.5 \times 2^{52}$)

Example

A code piece simplified from `exp` function in the GNU C library:

```c
double x = 3.7;
double n = 6755399441055744.0;
double y = (x + n) - n;
```

A precision-specific operation

Precision-Specific Semantics

```
double x = 3.7;
double n = 6755399441055744.0;
double y = (x + n) - n;
```

- Interpret as “rounding x to the nearest integer”
- Problems
    - Any other form of precision-specific operations?
    - How to understand its intention?
    - How to compose `n` for different precisions?

Precision-Specific Semantics

```
double x = 3.7;
double n = 6755399441055744.0;
double y = (x + n) - n;
```

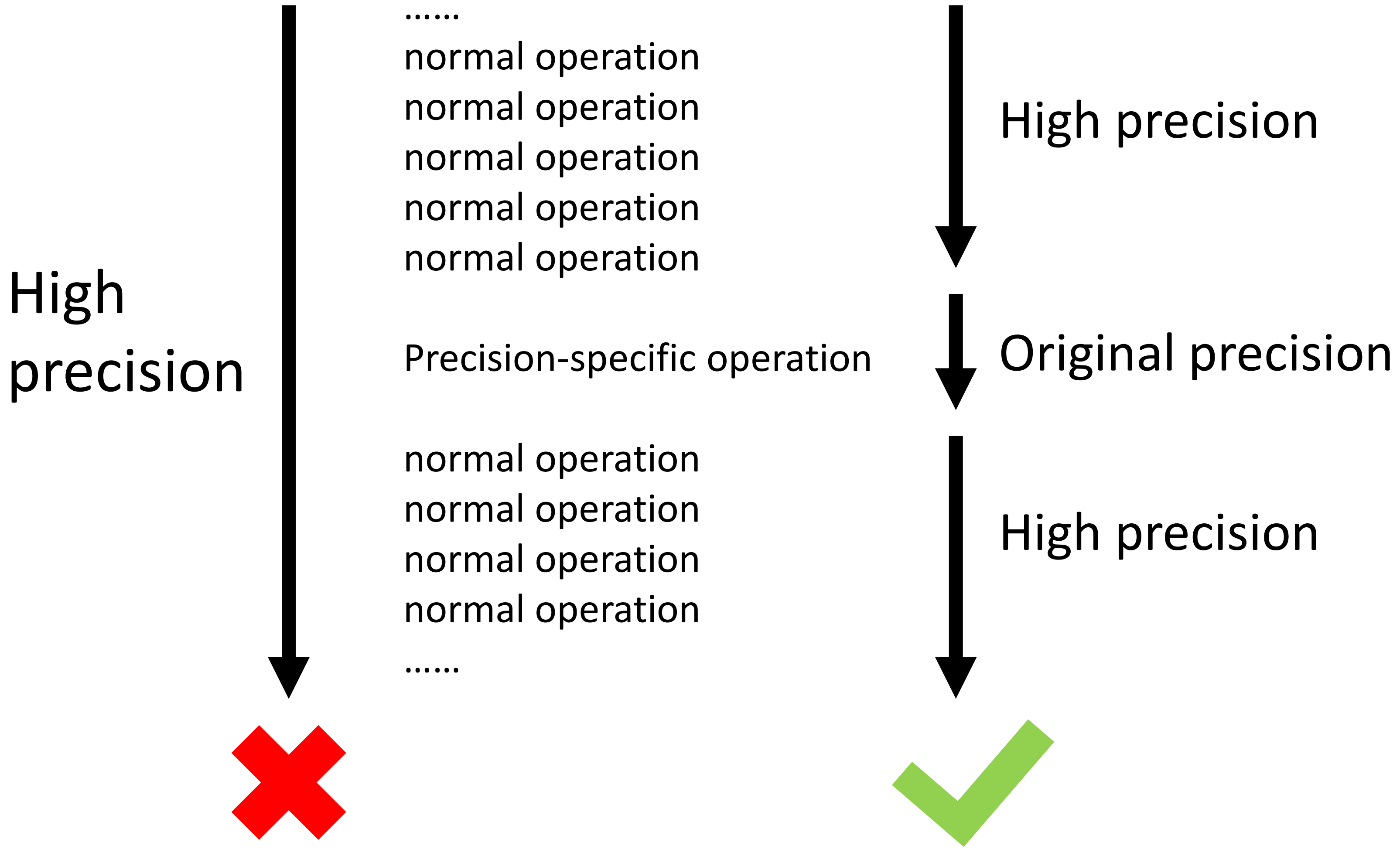
- Interpret as “rounding x to the nearest integer”
- Problems
  - Any other form of precision-specific operations?
  - How to understand its intention?
  - How to compose `n` for different precisions?
- Solution
  - A heuristic to detect precision-specific operations.
  - A fixing approach to enable precision tuning under the presence of precision-specific operations.

Heuristic

- An instruction is *possibly precision-specific* if it results in large error inflation in most executions.

- Intuition
  - Two sources of errors in computations
    - Accumulated Errors: usually small
    - Cancellation: normally occurs in a few executions

Fixing

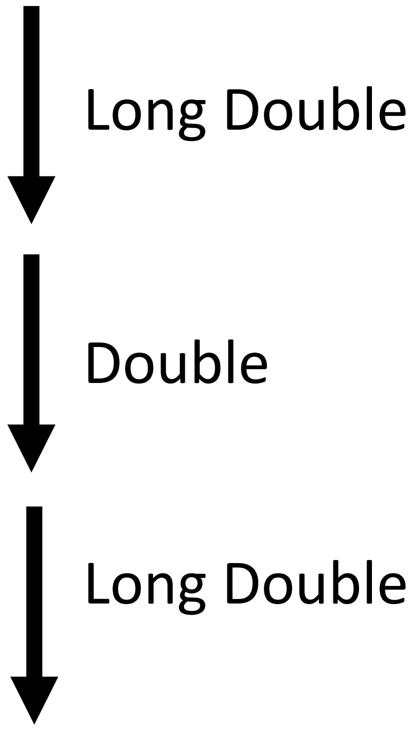

High precision
……
normal operation
normal operation
normal operation
normal operation
normal operation
Precision-specific operation
normal operation
normal operation
normal operation
normal operation
……
High precision
Original precision
High precision
✗
✓

Fixing Example

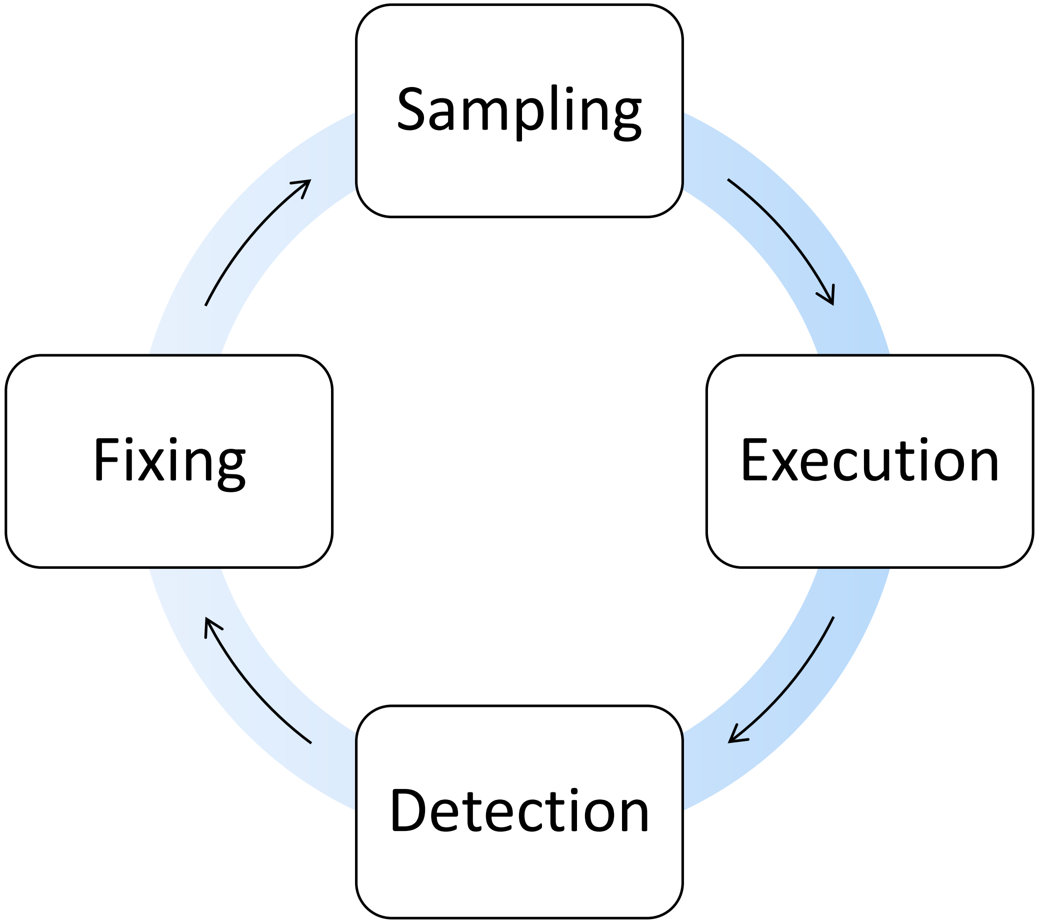
```
...
x = 3.7;
n = 6755399441055744.0;

temp = x + n;
y = temp - n;

...
```



Automatic Detection Overview

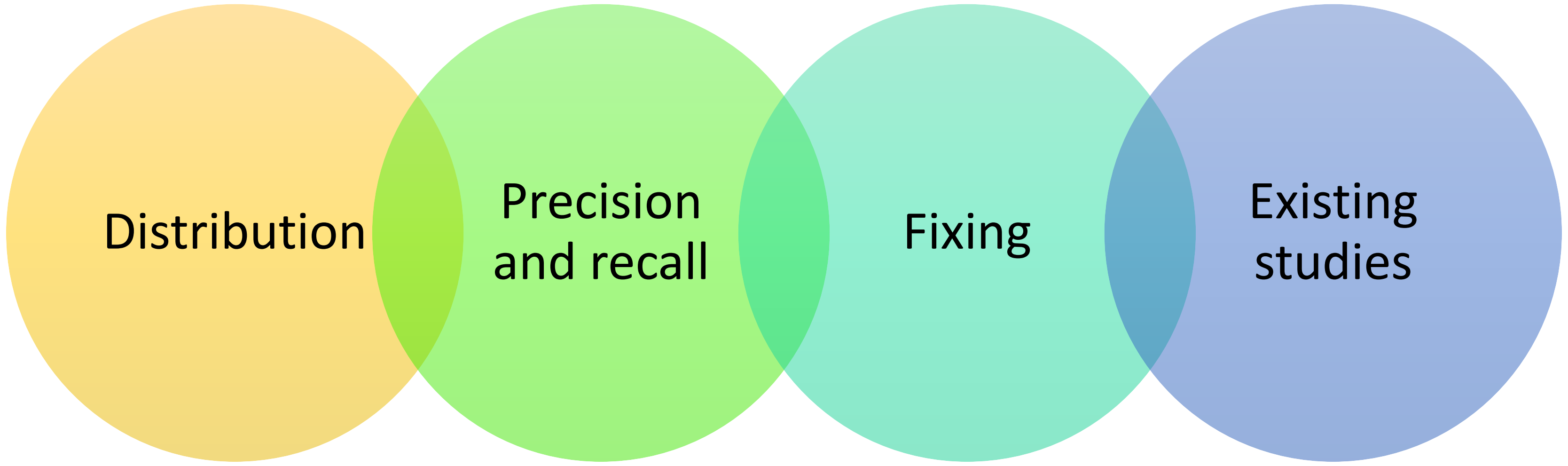


Evaluation

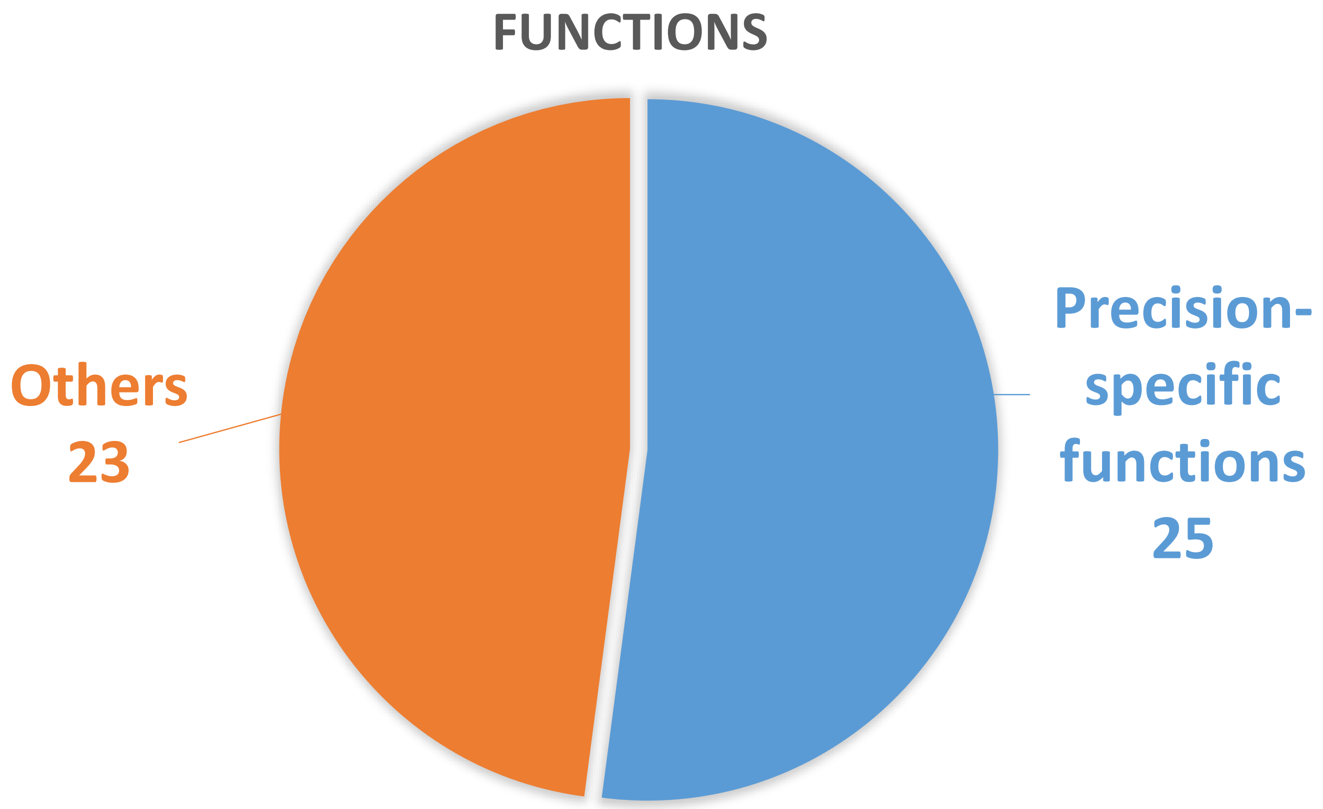
Subjects

48 double version math functions of the GNU C library (GLIBC).

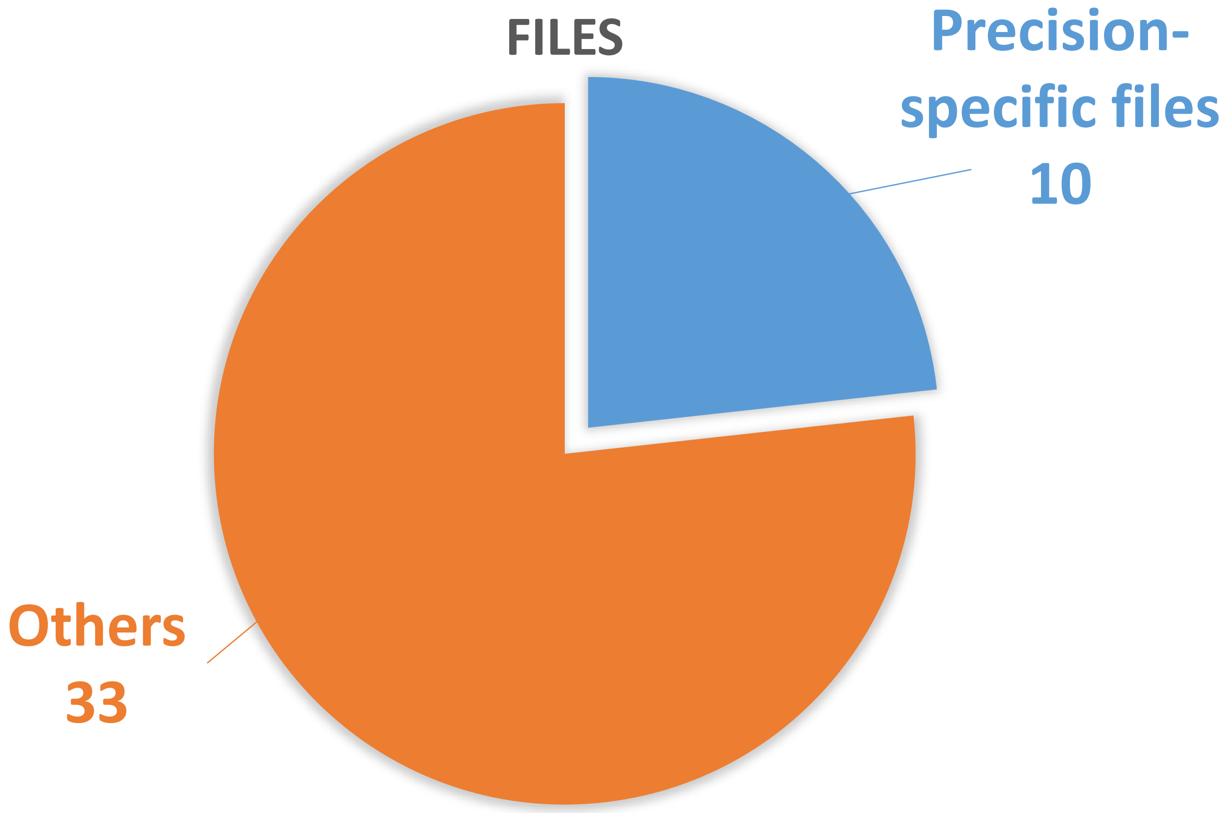
Research Questions



RQ1: Distribution of Precision-Specific Operations

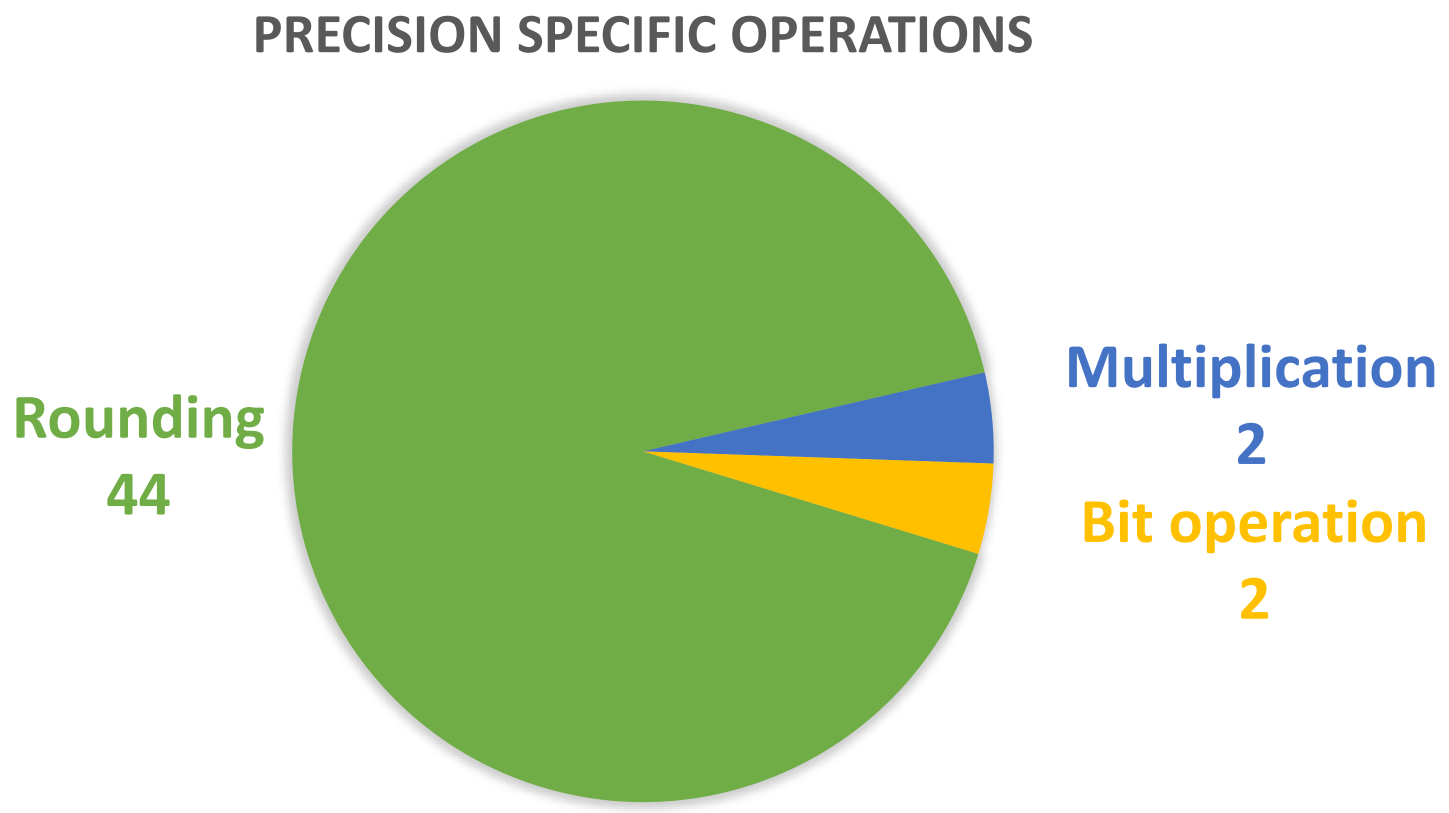


RQ1: Distribution of Precision-Specific Operations



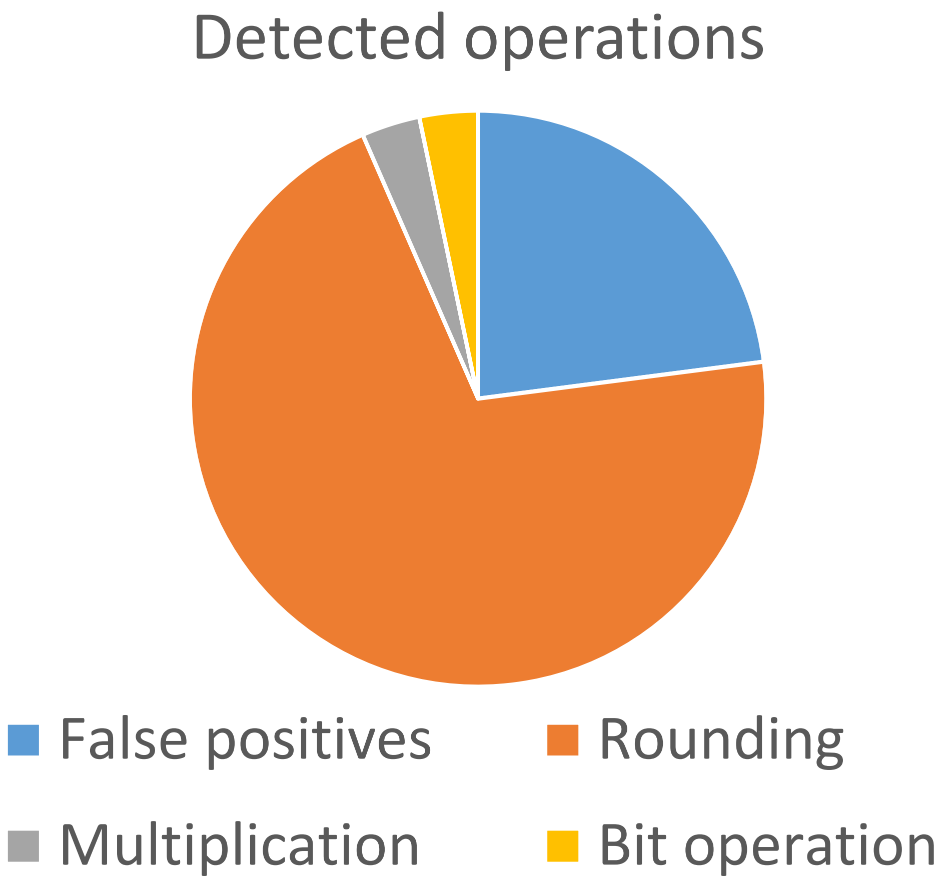
RQ1: Distribution of Precision-Specific Operations

In total, 48 precision-specific operations are detected.



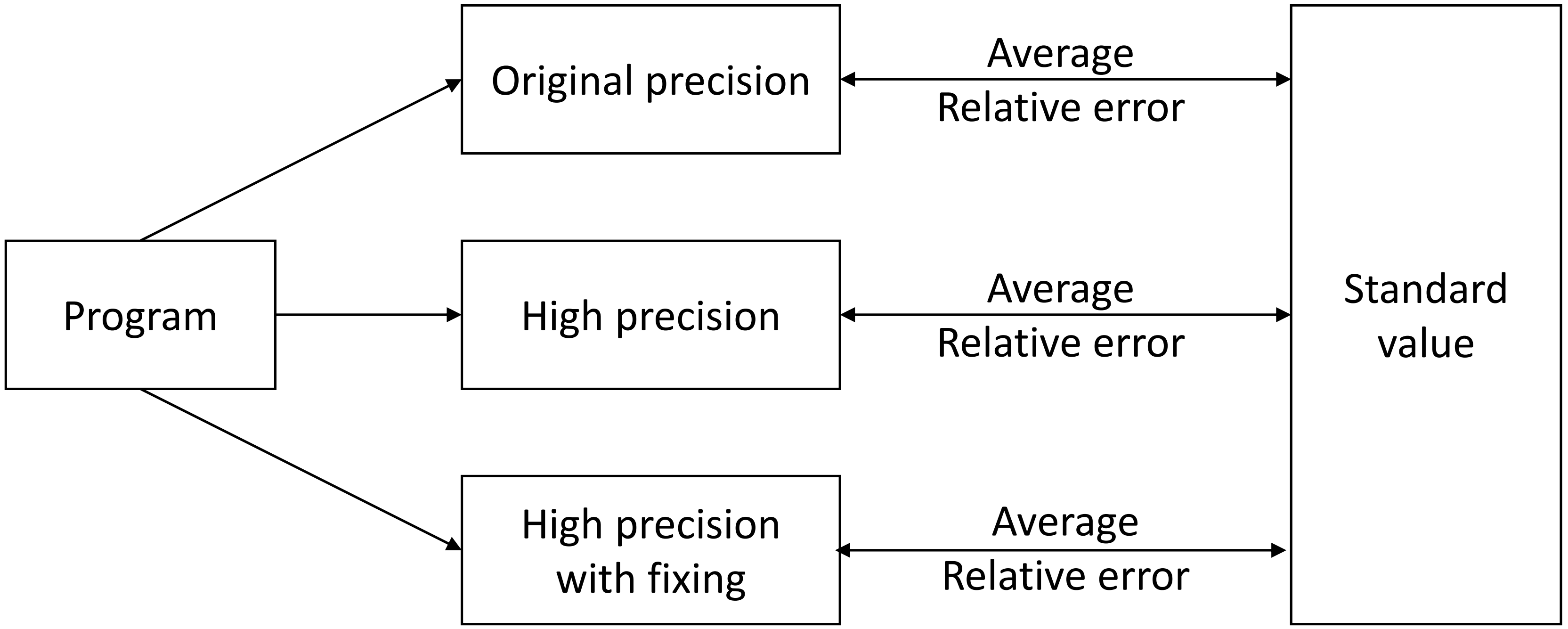
RQ2: Precision and Recall

- Precision: 77.5%
- Recall: 97.92%



RQ3: Fixing

- How effective is our fixing approach?
- We evaluate the average relative error to standard value.



RQ3: Automatic Fixing

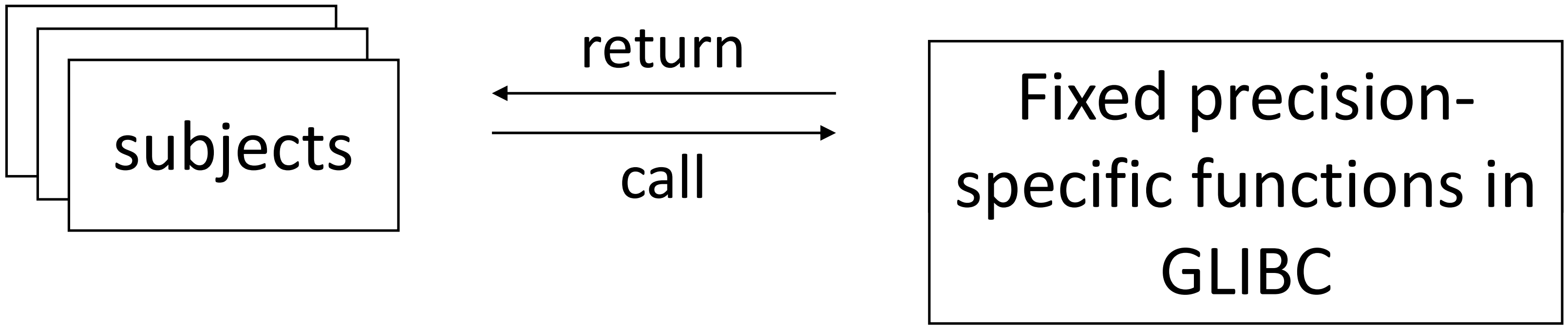
- Automatic fixing: fix detected instruction.
- The results from automatic fix are more accurate than the original precision and the high precision in the vast majority of cases.

Automatic Fixing ($E = 10^7$)			
$> O$	$< O$	$> H$	$< H$
20	1	12	2

O: original precision. H: high precision. >: better than. <: worse than

- Precision⬇ , error⬆
- Recall⬇ , error⬆

RQ4: Review Existing Study

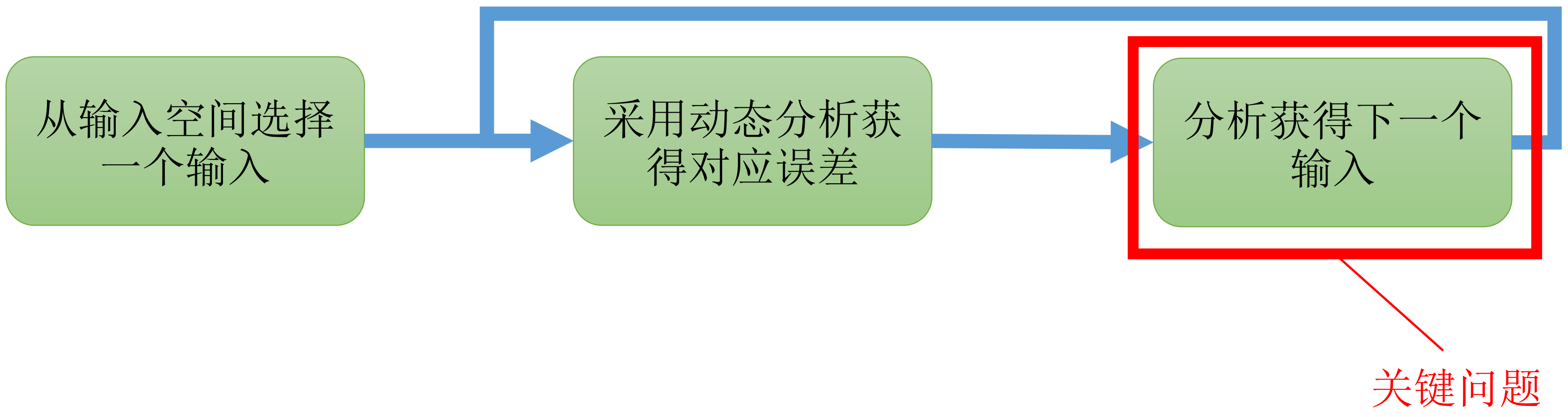


Function	Reported error	Actual error
exprel_2	2.85e+00	5.25e-12
synchrotron_1	5.35e-03	2.24e-13
synchrotron_2	3.67e-03	5.97e-15
equake (sin, cos)	Around 5.00e-01	--

小结与问题

- 修复精度特定运算之后，我们可以得到一次计算的误差
- 如何知道整个程序上是否存在较大误差？
  - 通常我们无法遍历整个程序的输入空间

基于搜索的误差查找



实证研究

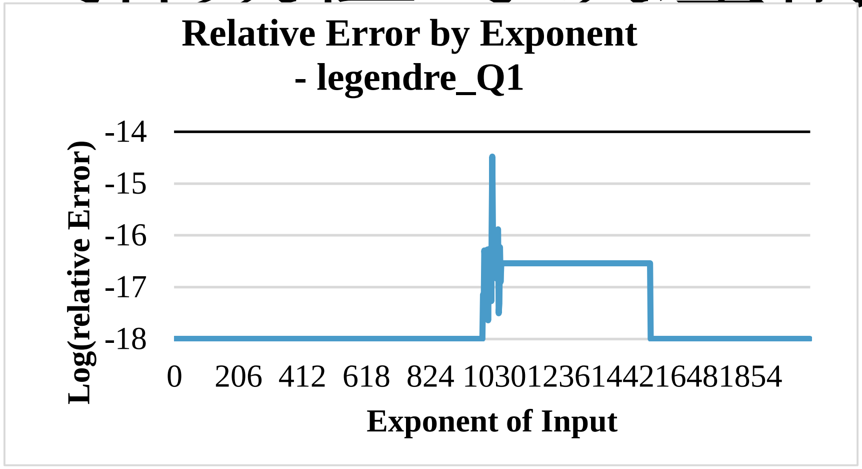
- 从GSL中选取4个函数，仅变化浮点数的尾数位或指数位，观察它们和内部误差之间的关系

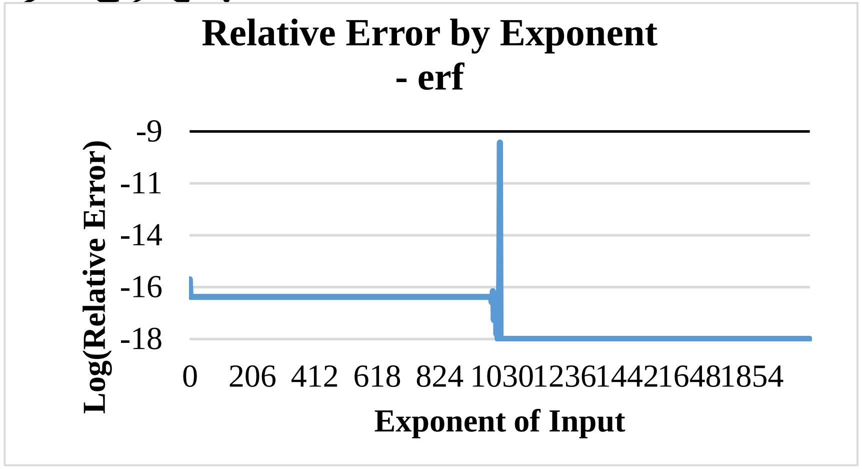
IEEE 754 Floating-Point Representation

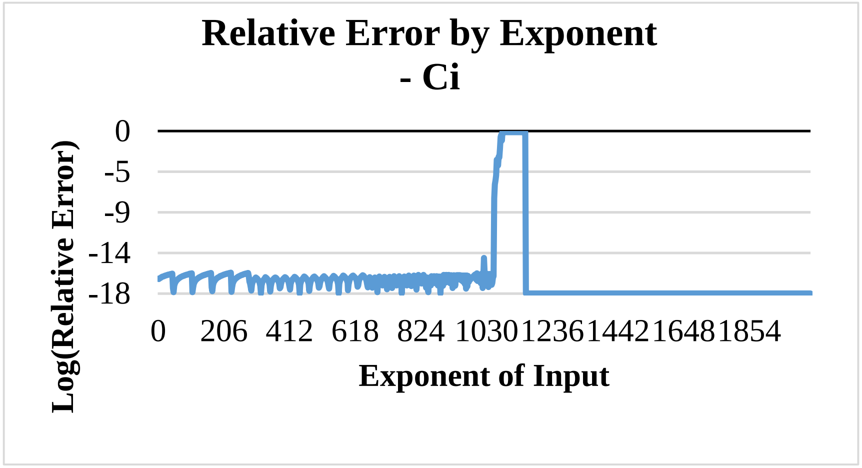
	Sign	Exponent	Significand
Single Precision	1	8	23
Double Precision	1	11	52

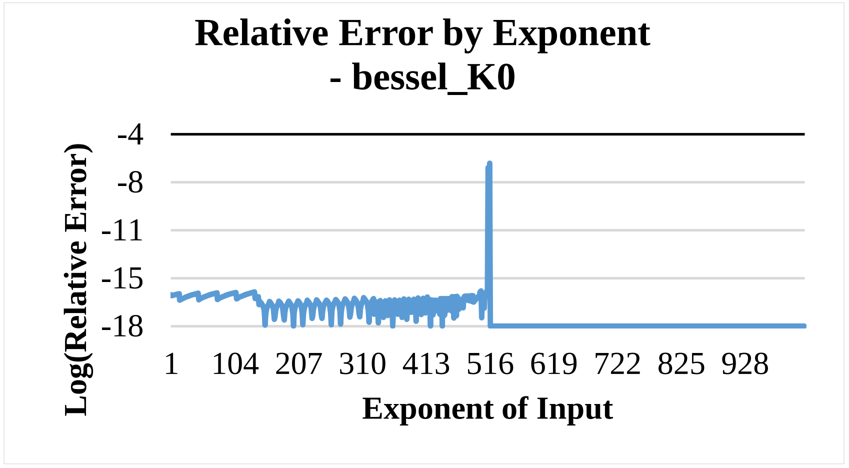
$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}{}^{\overbrace{-4}^{\text{exponent}}}$$

指数位与误差的关系


Relative Error by Exponent
- legendre_Q1
Log(relative Error)
-14
-15
-16
-17
-18
0
206
412
618
824
1030
1236
1442
1648
1854
Exponent of Input


Relative Error by Exponent
- erf
Log(Relative Error)
-9
-11
-14
-16
-18
0
206
412
618
824
1030
1236
1442
1648
1854
Exponent of Input


Relative Error by Exponent
- Ci
Log(Relative Error)
0
-5
-9
-14
-18
0
206
412
618
824
1030
1236
1442
1648
1854
Exponent of Input


Relative Error by Exponent
- bessel_K0
Log(Relative Error)
-4
-8
-11
-15
-18
1
104
207
310
413
516
619
722
825
928
Exponent of Input

主要发现 1

指数部分对误差的大小有重大影响。

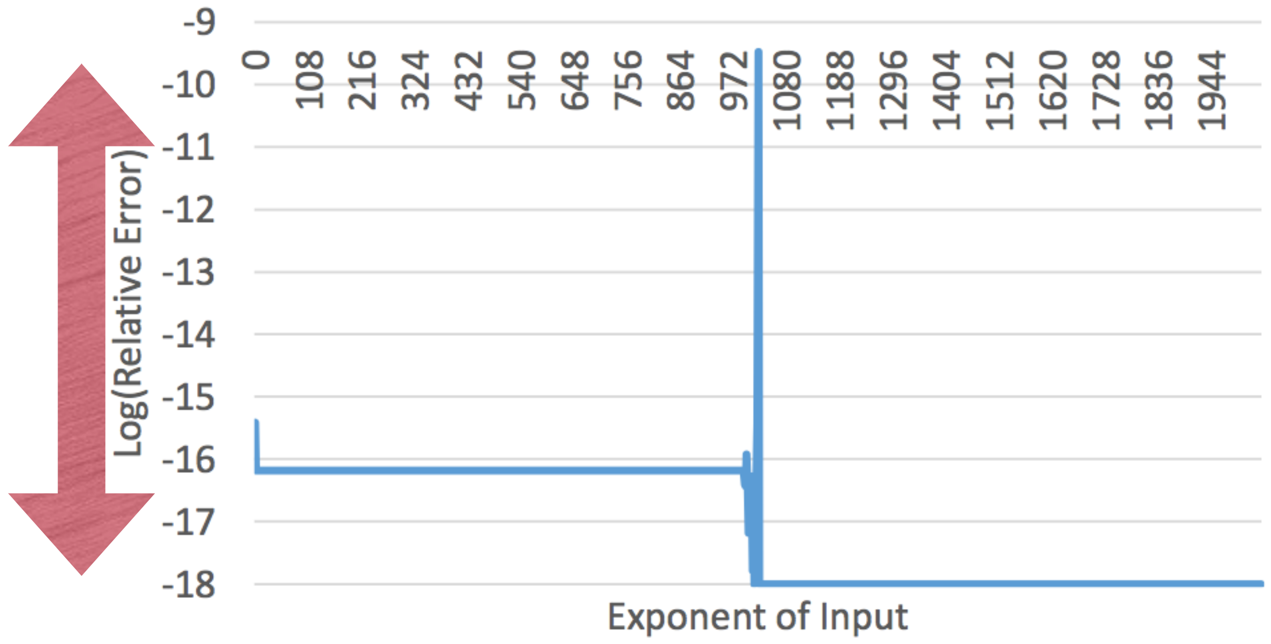


Fig. 1. erf at significand 0x34873b27b23c6

主要发现 2

能触发大误差的指数往往只存在于一个很小的范围。

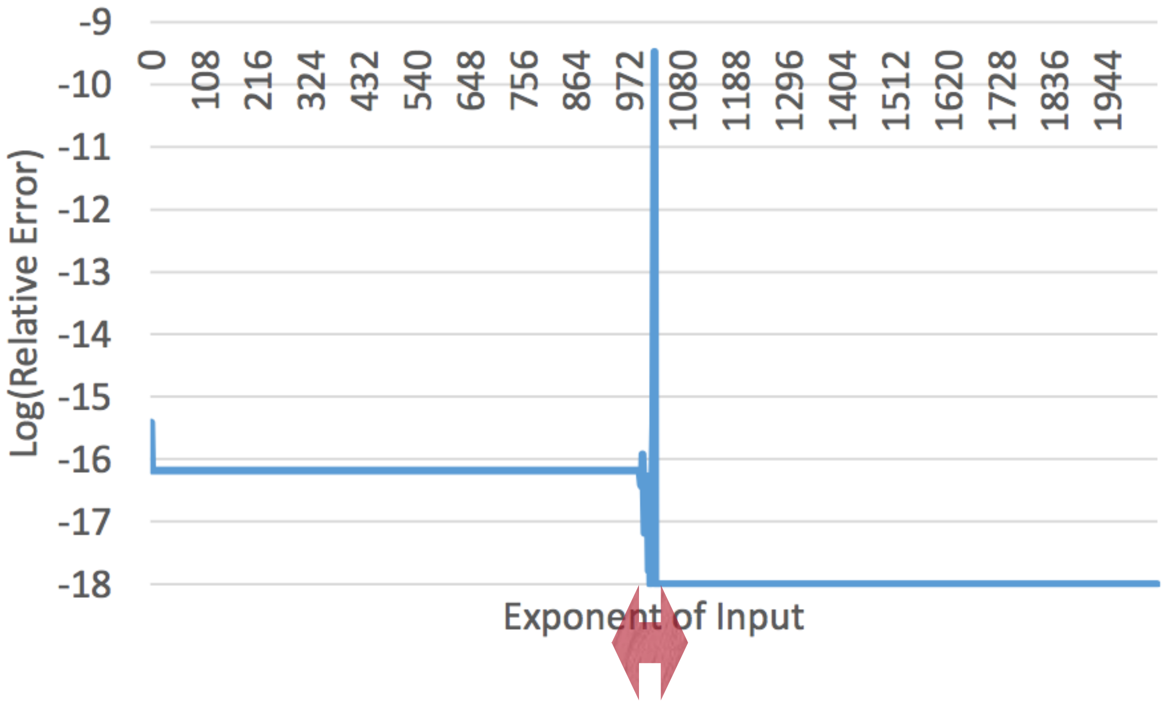


Fig. 1. erf at significand 0x34873b27b23c6

主要发现 3

在大误差的附近常常有高于平均误差的小波动。

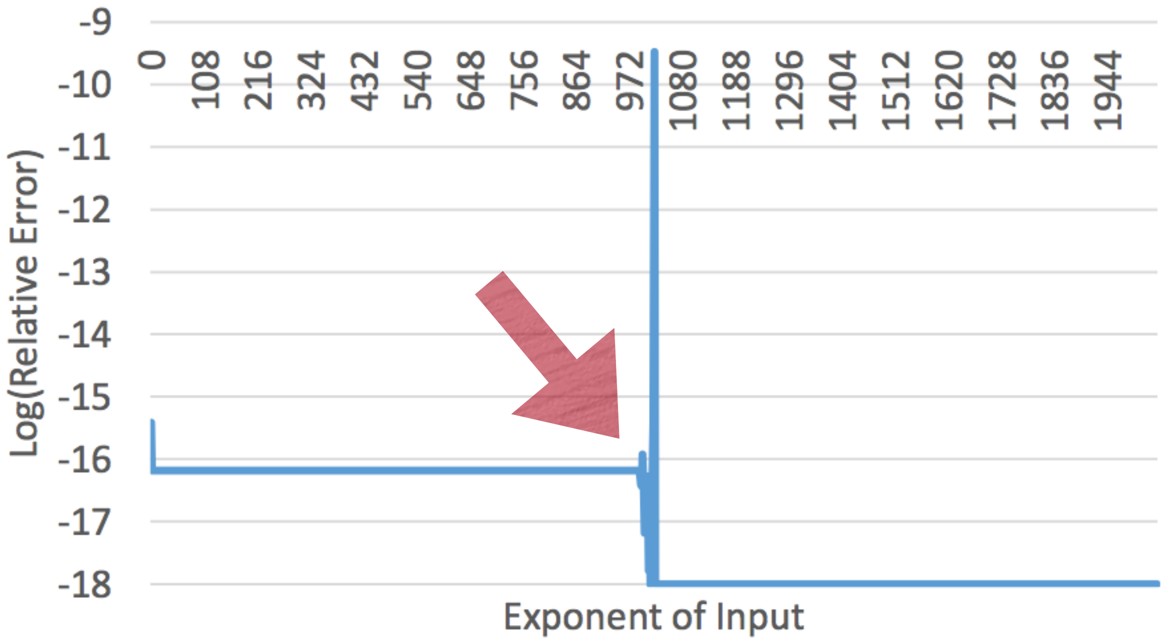


Fig. 1. erf at significand 0x34873b27b23c6

主要发现 4

大误差常常出现在浮点数中段位置。

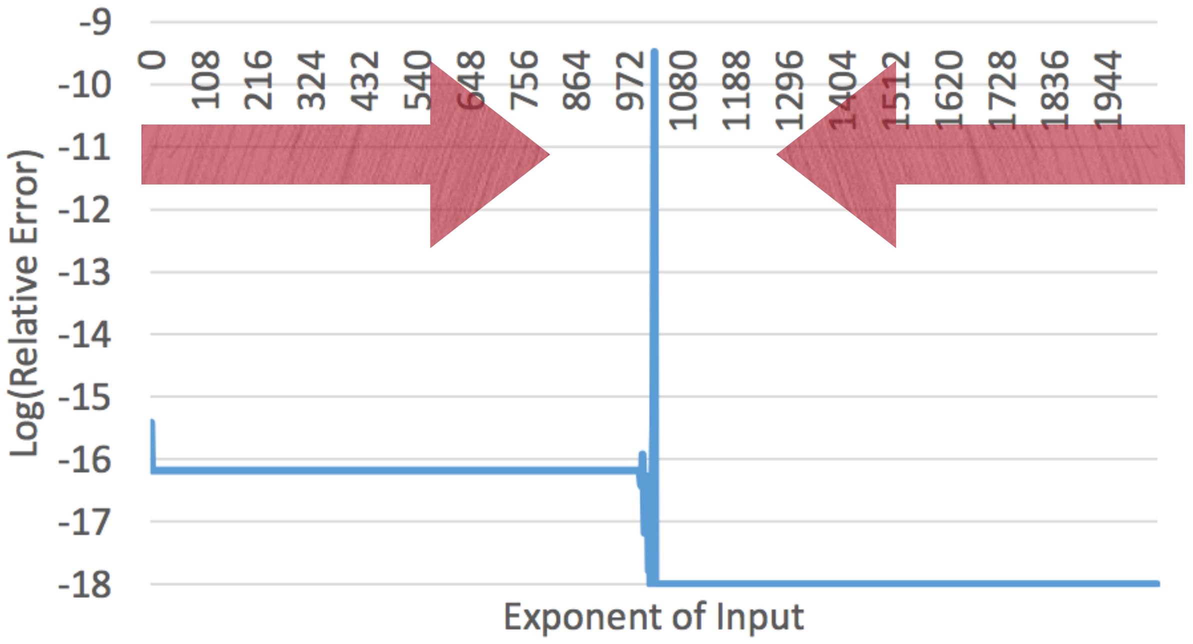


Fig. 1. erf at significand 0x34873b27b23c6

尾数位和浮点数之间的关系

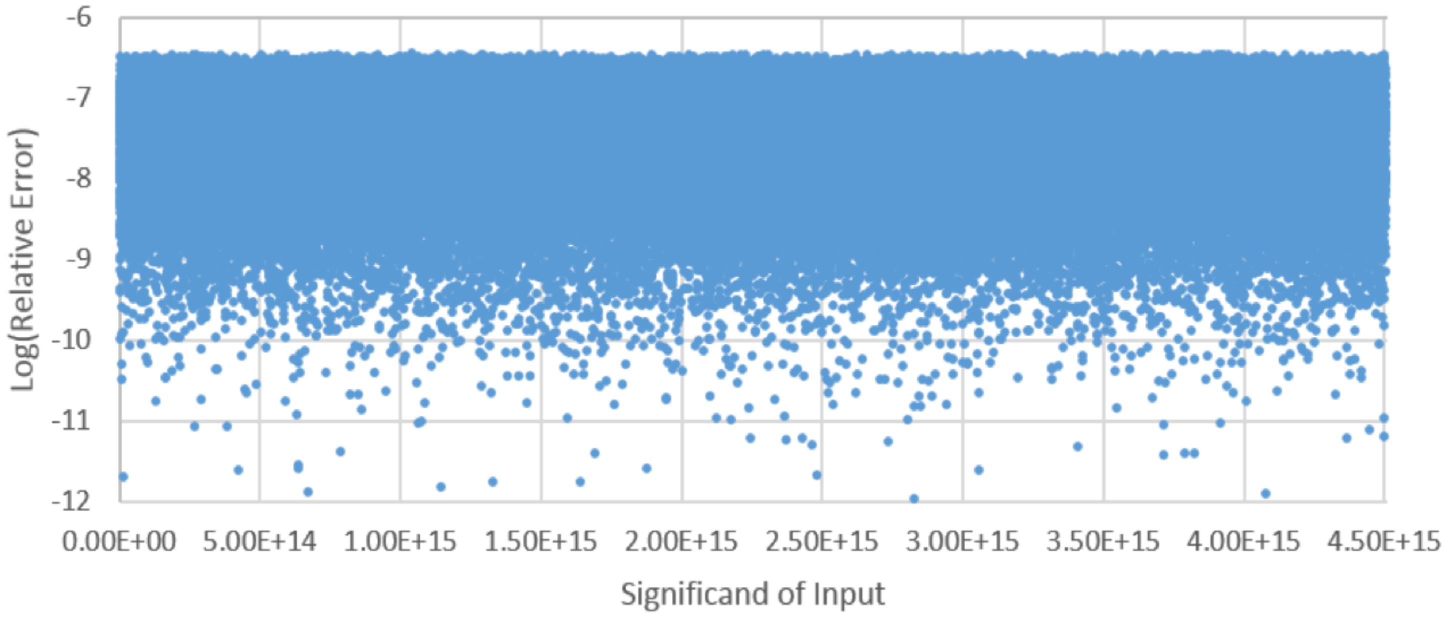


Fig. 2. erf at exponent 1023

主要发现 5

尾数位对浮点误差有显著影响。

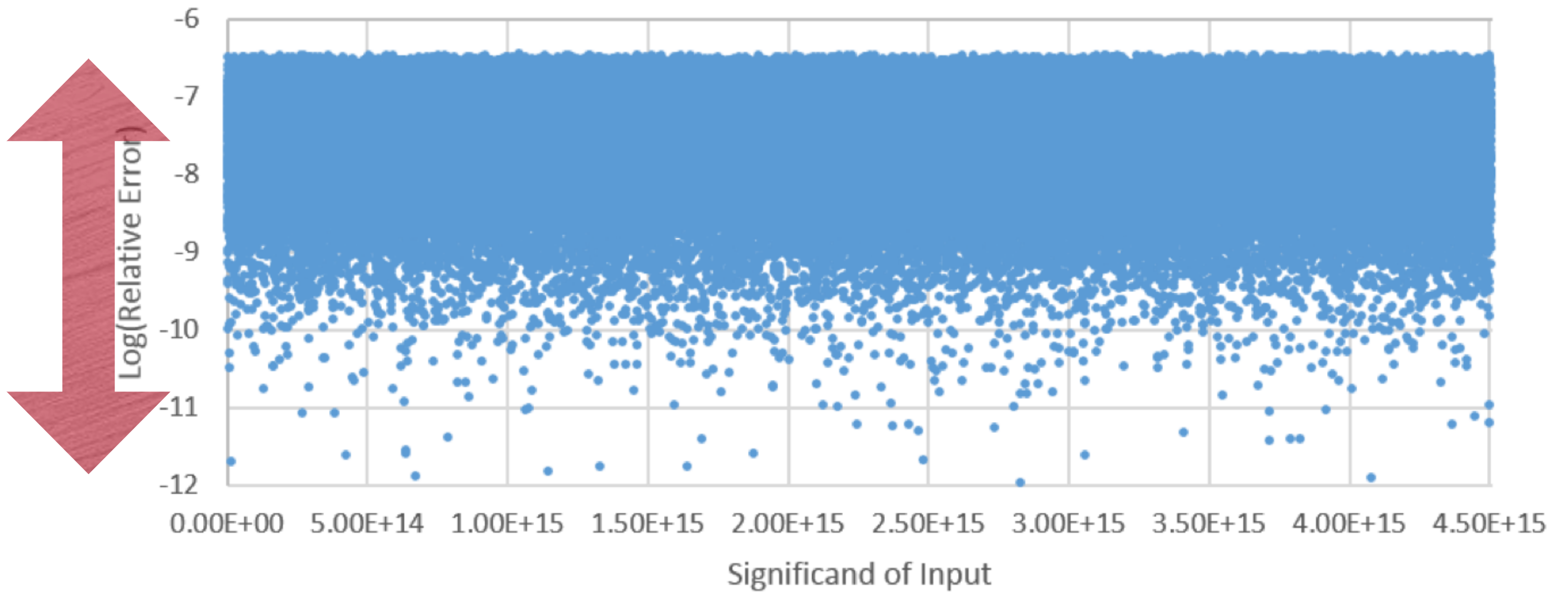


Fig. 2. erf at exponent 1023

主要发现 6

大量尾数都能触发大误差，并且在数轴上呈均匀分布。

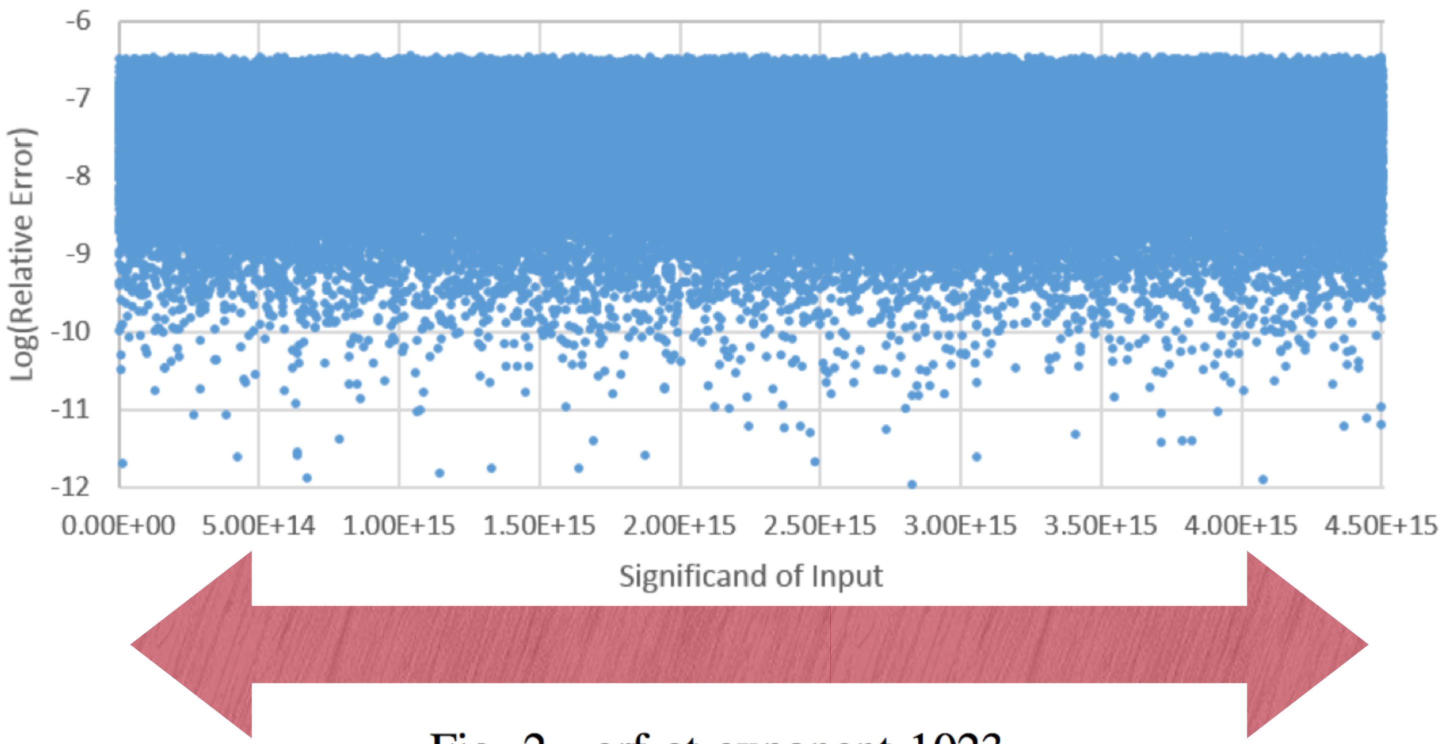


Fig. 2. erf at exponent 1023

局部敏感型遗传算法

- 根据以上特征，本研究设计了局部敏感型遗传算法。
- 对于指数位
  - 初始化时更多生成中位数附近。
  - 数值变异。
- 对于尾数位
  - 随机生成
  - 随机变异

实验评估

- 有效性检验：在6个经典浮点数程序上测试。
- 结果证明LSGA可以区分stable和unstable程序，初步证明了算法的可行性与有效性。

检验结果

	Newton	**Inv**	**Root**	**Poly**	**Exp**	**Cos**
Stable?	stable	stable	unstable	unstable	unstable	unstable
Max. Error Detected	2.8E-16	3.2E-16	8.1E+76	7.1E-11	1.0E+00	9.2E-01

实验评估

- 从GSL中选取154个函数。
- 与随机算法，标准遗传算法对比。

实验评估

检测到最大误差

Total	RAND	STD	LSGA	Tied
154	11 (7%)	24 (16%)	105 (68%)	14 (9%)

Sign Test 结果

	n_+	n_-	N	p
LSGA vs. RAND	127	12	139	< 4.14e-22
LSGA vs. STD	110	30	140	< 2.46e-11
STD vs. RAND	93	40	133	< 6.52e-06

实验评估

- 找到潜在内部精度问题的能力。
- 如何定义潜在问题？
  - 相对误差大于0.1%
  - 绝对误差大于预估绝对误差10倍以上

实验评估

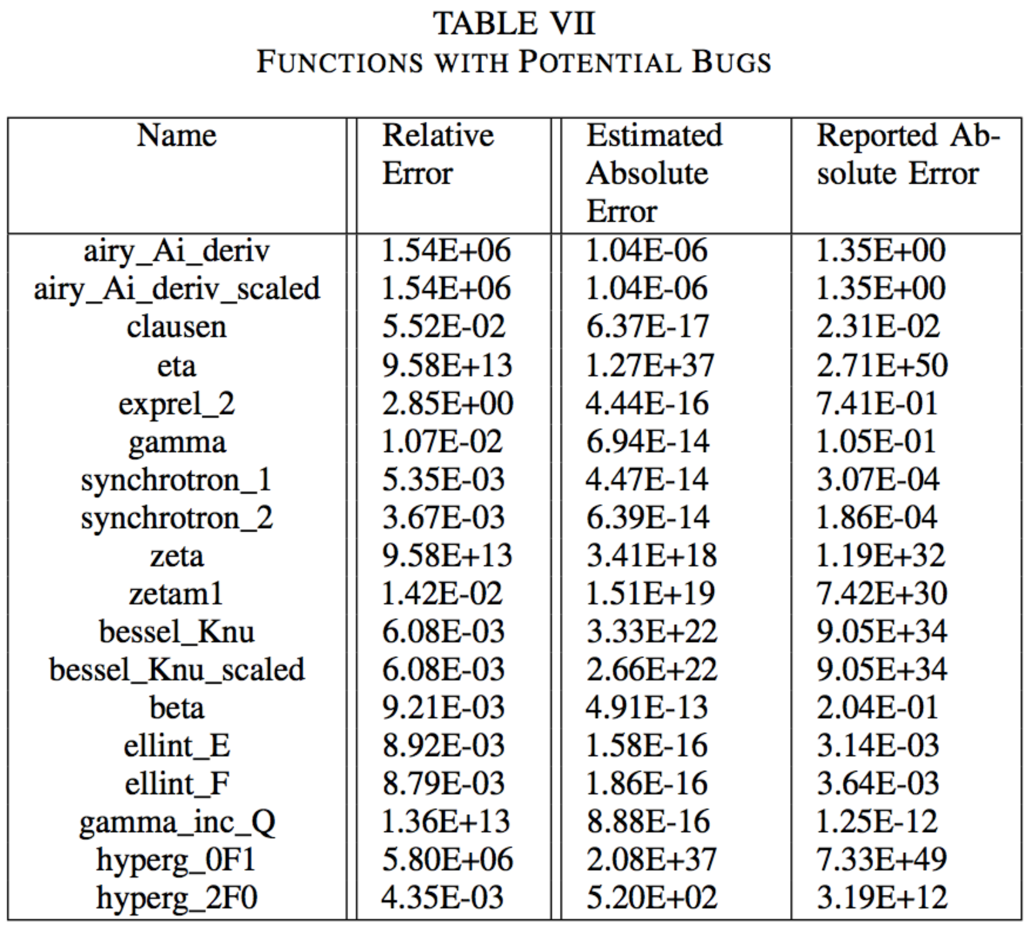


TABLE VII
FUNCTIONS WITH POTENTIAL BUGS

Name	Relative Error	Estimated Absolute Error	Reported Absolute Error
airy_Ai_deriv	1.54E+06	1.04E-06	1.35E+00
airy_Ai_deriv_scaled	1.54E+06	1.04E-06	1.35E+00
clausen	5.52E-02	6.37E-17	2.31E-02
eta	9.58E+13	1.27E+37	2.71E+50
exprel_2	2.85E+00	4.44E-16	7.41E-01
gamma	1.07E-02	6.94E-14	1.05E-01
synchrotron_1	5.35E-03	4.47E-14	3.07E-04
synchrotron_2	3.67E-03	6.39E-14	1.86E-04
zeta	9.58E+13	3.41E+18	1.19E+32
zetam1	1.42E-02	1.51E+19	7.42E+30
bessel_Knu	6.08E-03	3.33E+22	9.05E+34
bessel_Knu_scaled	6.08E-03	2.66E+22	9.05E+34
beta	9.21E-03	4.91E-13	2.04E-01
ellint_E	8.92E-03	1.58E-16	3.14E-03
ellint_F	8.79E-03	1.86E-16	3.64E-03
gamma_inc_Q	1.36E+13	8.88E-16	1.25E-12
hyperg_0F1	5.80E+06	2.08E+37	7.33E+49
hyperg_2F0	4.35E-03	5.20E+02	3.19E+12

小结

- 浮点数误差是软件开发的经典问题
- 本研究成果可以自动查找软件代码中的浮点误差问题
- 本研究成果避免了已有方法可能产生的错误结论
- 实验结果显示，我们设计的算法可以在实践中被广泛使用的库函数中找到潜在的精度问题

- 相关论文
  - Daming Zou, Ran Wang, Yingfei Xiong, Lu Zhang, Zhendong Su, Hong Mei. A Genetic Algorithm for Detecting Significant Floating-Point Inaccuracies. ICSE'15: 37th International Conference on Software Engineering, pages 529-539, May 2015
  - Ran Wang, Daming Zou, Xinrui He, Yingfei Xiong, Lu Zhang, Gang Huang. Detecting and Fixing Precision-Specific Operations for Measuring Floating-Point Errors. FSE'16: 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, November 2016.