湖南大學
HUNAN UNIVERSITY

Scalable Detection of Floating-point Errors via Adaptive Parallel Subdomain Search

Zuoyan Zhang, Shihan Yuan, Hongru Yang, Jie Zhao
Hunan University

JinChen Xu*
Information Engineering University

July 19, 2025

Outline

Floating-point errors

Some inputs may trigger significant floating-point errors, considering:

$$f(x) = \frac{tanx - sinx}{x^3} \tag{1}$$

When x limits to zero, the result equals to 0.5, the equation as showns below:

$$\lim_{x \to 0} f(x) = 0.5 \tag{2}$$

Considering the function in C++ language

```
double f(double x) {
 double num = tan(x) - sin(x);
 double den = x * x * x;
 return num / den;
}
```

The result of `f(1e-7)` in `double` precision (64-bit) is **0.5029258124322410**, while the result with 128-bit precision is **0.5000000000000012**

Floating-point errors

The root cause is:

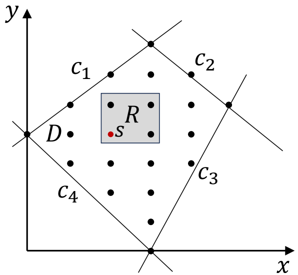
- Finite precision bits cannot represent all real numbers exactly

And the rounding errors can be amplified by floating-point operations

Large errors may lead to catastrophic software failures:

1. Missile yaw *[skeel' 92]*
2. Stock trading disorder *[Quinn' 83]*
3. Rocket launch failure *[Lions' 96]*

`Detect floating-point errors is a crucial work!`



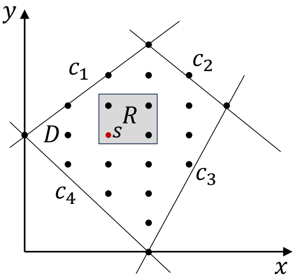
Guided search

Guided search process

- Search space $D = \{(x, y) : c_1 \wedge c_2 \wedge c_3 \wedge c_4\}$
- c_1, c_2, c_3 and c_4 are constraints of the two variables x and y
- The points within R are input values that may trigger significant errors
- s is the input that triggers the maximum error

Guided search goal is to find the point of s

What's the difficulties of guided search?



Guided search

Difficulties

1. D may be complex and large, especially multi-dimension space
2. s and R could both be many

Existing search method

Herbie[1]

- Using Random Search (RS)

S3fp[2]

- Using Binary Guided Random Testing (BGRT)

HSED[3]

- Using Hierarchical Search (HS)

EIFFEL[4]

- Using Polynomial Extrapolation Search (PES)

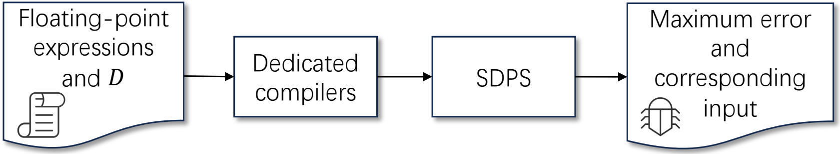
Limitations

1. Poor scalability for multi-parameter functions
2. Limited exploitation of floating-point characteristics
3. Insufficient parallelization

[1] **https://github.com/herbie-fp/herbie**.
[2] **https://github.com/soarlab/S3FP**.
[3] **https://github.com/zuoyanzhang/HSED**.
[4] **https://github.com/zuoyanzhang/Maxfpeed**.

Our method

We present the FPEPD[5] to solve those problems

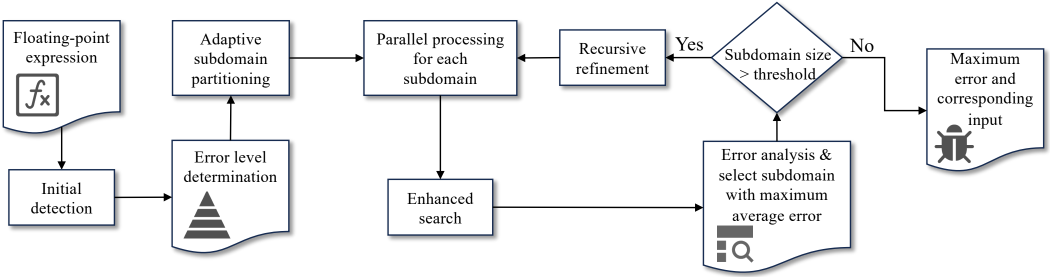


Dedicated compiler module generates the code of high-precision in order to obtain the oracle value

SDPS is the core of the FPEPD

[5] **https://github.com/zuoyanzhang/FPEPD**.

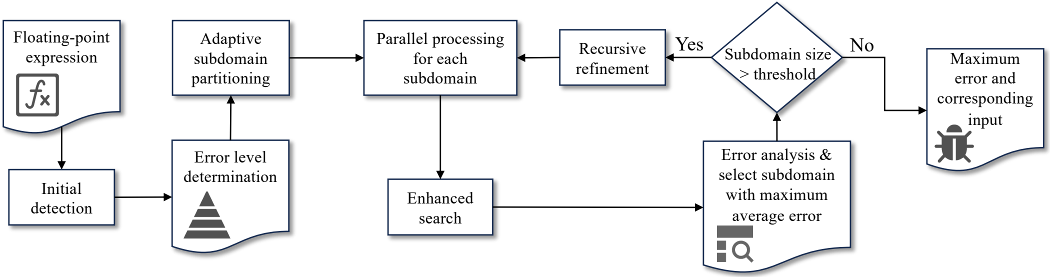
The framework of SDPS



SDPS's key innovations

1. Multi-level error classification
2. Specialized point generation strategies
3. Automatic strategy adaptation

The framework of SDPS

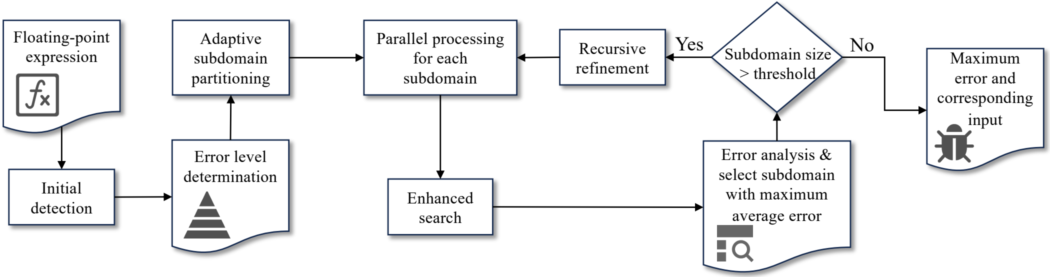


Error classification and sampling strategy

- Five-level error tiers (ε_{min}, ε_s, ε_m, ε_l, ε_c) assigned after a quick detect; larger detected error $\rightarrow$ higher tier
- Per-tier rules: higher tier $\rightarrow$ more aggressive sampling (more initial points, more hotspots kept, more local variants)

Five-level classification steers adaptive sampling, focusing effort on high-error regions while keeping overall cost low

The framework of SDPS

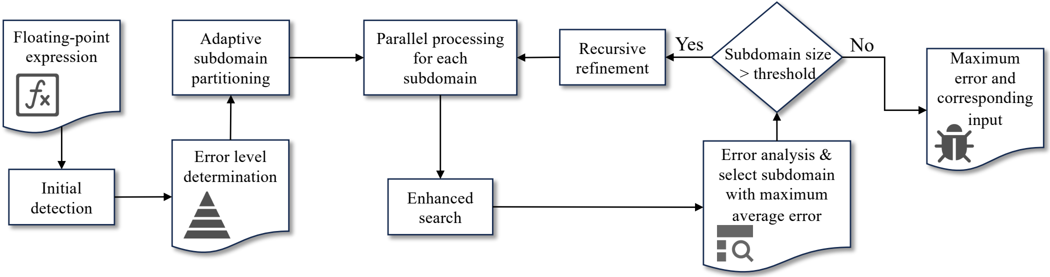


Point generation strategies

- **Gradient search**: follow local error slope with adaptive step to climb toward hot spots
- **Special-value probes**: hit powers-of-two, interval edges, and `nextafter` neighbors to stress corner cases
- **Mantissa tweaking**: keep exponent fixed, flip low-order bits to expose reprentation-specific errors

These three tactics jointly maximize search coverage while zeroing in on the worst errors

The framework of SDPS



Parallel implementation

- Sub-domains are independent, hence the search is embarrassingly parallel
- A dynamic thread pool spreads them across all CPU cores, keeping every core busy

Experiment environment and benchmarks

We implemented FPEPD using about **5,000** lines of code in C++ language

Experiment environment

- macOS 14.5 (BuildVersion 23F79) with an Apple M3 Pro chip
- 11 CPU cores (5 performance and 6 efficiency cores) and 14 GPU cores
- 18 GB memory
- Compiled with `clang++ 16.0.0` using the "`-lm -lmpfr`" compilation options

Benchmarks

- We selected **59** benchmarks from FPBench[6]
- Encompassing all single-, double-, and triple-parameter expressions available in the FPBench
- **32** univariate and **27** multivariate expressions

[6] **https://fpbench.org/**.

General applicability

Bencharmak converge

- **Herbie and EIFFEL**: Support multi-variable detection, but limited by accuracy or high overhead
- **S3fp**: Frequent timeouts (27 failures)
- **HSED**: Only work for single-variable cases

FPEPD advantages

- Supports both single and multi-variable expressions
- Detects larger maximum errors in most cases
- Lower computational overhead

	no.	FPBench	D	Herbie	S3fp	HSED	EIFFEL	FPEPD	Time
single-variate	1	sqroot	[0,1]	✓	✓	✓	✓	3.67E-16	0.08
	2	sqrt_add	[1,1000]	✓	N/R	✓	✓	2.61E-16	0.06
	3	exp1x	[0.01,0.5]	✓	✓	✓	✓	1.14E-14	0.71
	4	exp1x_log	[0.01,0.5]	✓	N/R	✓	✓	2.55E-16	0.37
	5	NMSEexample37	[0.01,100]	✓	N/R	✓	✓	1.11E-14	0.86
	6	NMSEproblem336	[0.01,100]	✓	N/R	✓	✓	8.12E-14	1.20
	7	NMSEexample39	[0.01,100]	✓	✓	✓	✓	7.23E-12	0.32
	8	NMSEproblem344	[0.01,100]	✓	N/R	✓	✓	7.98E-15	0.33
	9	NMSEsection311	[0.01,100]	✓	N/R	✓	✓	1.12E-14	0.99
	10	NMSEproblem345	[0.01,100]	✓	✓	✓	✓	1.05E-11	0.48
	11	NMSEproblem337	[0.01,100]	✓	N/R	✓	✓	1.65E-12	1.03
	12	verhulst	[0.1,0.3]	✓	✓	✓	✓	2.00E-16	0.04
	13	predatorPrey	[0.1,0.3]	✓	✓	✓	✓	2.74E-16	0.05
	14	logexp	[0.01,8]	✓	N/R	✓	✓	8.72E-06	1.60
	15	sine	[-π/2,π/2]	✓	✓	✓	✓	3.03E-16	0.12
	16	carbonGas	[0.1,0.5]	✓	✓	✓	✓	2.81E-16	0.07
	17	NMSEproblem341	[0.01,100]	✓	✓	✓	✓	1.15E-09	0.64
	18	NMSEexample38	[0.01,100]	✓	✓	✓	✓	7.54E-04	1.31
	19	NMSEproblem334	[0.01,100]	✓	N/R	✓	✓	5.33E-14	0.21
	20	NMSEproblem333	[0.01,100]	✓	✓	✓	✓	1.59E-12	0.19
	21	NMSEproblem331	[0.01,100]	✓	✓	✓	✓	1.52E-14	0.13
	22	NMSEexample36	[0.01,100]	✓	N/R	✓	✓	5.15E-14	0.17
	23	NMSEexample35	[0.01,100]	✓	N/R	✓	✓	9.66E-15	0.79
	24	NMSEexample34	[0.01,100]	✓	N/R	✓	✓	6.19E-09	0.63
	25	NMSEexample31	[0,100]	✓	N/R	✓	✓	3.48E-14	0.13
	26	test05_nonlin1_r4	[1.00001,2]	✓	✓	✓	✓	1.80E-16	0.04
	27	test05_nonlin1_test2	[1.00001,2]	✓	✓	✓	✓	1.57E-16	0.03
	28	intro-example-mixed	[1,999]	✓	✓	✓	✓	1.64E-16	0.04
	29	sineOrder3	[-2,2]	✓	✓	✓	✓	2.04E-16	0.05
	30	bsplines3	[0,1]	✓	✓	✓	✓	2.26E-16	0.04
	31	NMSEexample310	[0.001,1]	✓	N/R	✓	✓	1.14E-13	1.14
	32	NMSEproblem343	[0.001,1]	✓	N/R	✓	✓	7.99E-14	0.84
double-variate	33	nonlin2	[1.001,2][1.001, 2]	✓	✓	✗	✓	1.78E-05	0.33
	34	hypot	[1,100][1,100]	✓	N/R	✗	✓	1.59E-16	0.09
	35	x_by_xy	[1,4][1,4]	✓	✓	✗	✓	2.15E-16	0.07
	36	NMSEproblem335	[0.01,100][0.01,100]	✓	N/R	✗	✓	1.96E-03	1.47
	37	NMSEproblem332	[0.01,100][0.01,100]	✓	N/R	✗	✓	1.22E-02	4.60
	38	floudas3	[0,2][0,3]	✓	✓	✗	✓	3.17E-16	0.11
	39	himmibeau	[-5,5][-5,5]	✓	✓	✗	✓	9.74E-06	0.61
	40	carthesianToPolar	[1,100] [1,100]	✓	N/R	✗	✓	2.69E-16	0.44
	41	NMSEexample33	[0.01,100][0.01,100]	✓	N/R	✗	✓	1.71E-03	1.41
	42	i4	[0.1,10][-5,5]	✓	N/R	✗	✓	1.61E-16	0.08
	43	i6	[0.1,10][-5,5]	✓	N/R	✗	✓	1.86E-03	1.39
	44	test03_nonlin2	[0,1][-1,-0.1]	✓	✓	✗	✓	2.59E-16	0.07
	45	polarToCarthesian,x	[1,10][0,360]	✓	✓	✗	✓	3.43E-02	1.38
	46	polarToCarthesian,y	[1,10][0,360]	✓	✓	✗	✓	2.50E-01	1.40
	47	NMSEproblem346	[0.01,100][0.01,100]	✓	N/R	✗	✓	2.03E-12	5.04
	48	complex_square_root	[0.01,100][0.01,100]	✓	N/R	✗	✓	1.84E-16	0.15
	49	complex_sine_cosine	[0.01,100][0.01,100]	✓	N/R	✗	✓	8.05E-15	2.99
	50	jetEngine	[-5,5][-20,5]	✓	✓	✗	✓	2.14E-02	2.10
	51	rump's example	[0.01,100][0.01,100]	✓	✓	✗	✓	5.44E-03	1.43
triple-variate	52	doppler1	[-100,100][20,20000][-30,50]	✓	✓	✗	✓	5.64E-16	0.10
	53	sum	[1,2][1,2][1,2]	✓	N/R	✗	✓	1.97E-16	0.06
	54	rigidBody1	[-15,15][-15,15][-15,15]	✓	✓	✗	✓	9.73E-12	0.18
	55	rigidBody2	[-15,15][-15,15][-15,15]	✓	✓	✗	✓	1.77E-10	0.28
	56	turbine1	[-4.5,-0.3][-2.5,0.9][3.8,7.8]	✓	✓	✗	✓	4.67E-16	0.09
	57	turbine2	[-4.5,-0.3][-2.5,0.9][3.8,7.8]	✓	✓	✗	✓	5.03E-11	0.21
	58	turbine3	[-4.5,-0.3][-2.5,0.9][3.8,7.8]	✓	✓	✗	✓	4.71E-12	0.27
	59	test01_sum3	[1,2][1,2][1,2]	✓	N/R	✗	✓	1.97E-16	0.06

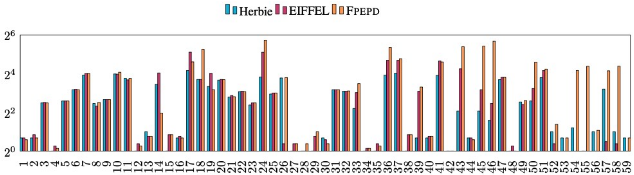
Effectiveness

Average bit error (higher is better)

- **FPEPD** 11.9/15.1* Herbie 6.1/4.9* EIFFEL 7.8/7.0*
  *(mutli-variable subset)

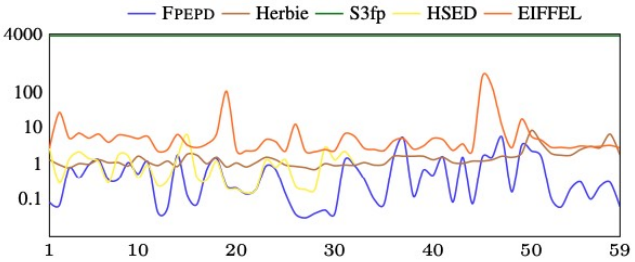
Benchmarks where FPEPD detects the largest error

- 40/59 vs Herbie
- 28/59 vs EIFFEL
- 54/59 vs S3fp
- 23/32 vs HSED



	no.	FPBench	HSED	S3fp	FPEPD
single-variate	1	sqroot	3.29E-16	3.13E-16	3.67E-16
	2	sqrt_add	2.69E-16	N/R	2.61E-16
	3	exp1x	1.10E-14	1.10E-16	1.14E-14
	4	exp1x_log	2.45E-16	N/R	2.55E-16
	5	NMSEexample37	8.10E-15	N/R	1.11E-14
	6	NMSEproblem336	6.73E-14	N/R	8.12E-14
	7	NMSEexample39	5.59E-12	2.07E-12	7.23E-12
	8	NMSEproblem344	5.97E-15	N/R	7.98E-15
	9	NMSEsection311	1.07E-14	N/R	1.12E-14
	10	NMSEproblem345	7.79E-12	3.05E-16	1.05E-11
	11	NMSEproblem337	1.52E-12	N/R	1.65E-12
	12	verhulst	1.66E-16	2.10E-16	2.00E-16
	13	predatorPrey	2.69E-16	2.15E-16	2.74E-16
	14	logexp	4.99E-13	N/R	8.72E-06
	15	sine	2.79E-16	3.03E-16	3.03E-16
	16	carbonGas	2.98E-16	3.41E-16	2.81E-16
	17	NMSEproblem341	3.48E-03	1.10E-16	1.15E-09
	18	NMSEexample38	5.91E-09	6.59E-14	7.54E-04
	19	NMSEproblem334	5.72E-14	N/R	5.33E-14
	20	NMSEproblem333	1.54E-12	2.88E-14	1.59E-12
	21	NMSEproblem331	1.67E-14	6.49E-15	1.52E-14
	22	NMSEexample36	4.60E-14	N/R	5.15E-14
	23	NMSEexample35	9.57E-15	N/R	9.66E-15
	24	NMSEexample34	3.58E-03	N/R	6.19E-09
	25	NMSEexample31	3.27E-14	N/R	3.48E-14
	26	test05_nonlin1_r4	1.66E-16	8.30E-17	1.80E-16
	27	test05_nonlin1_test2	1.67E-16	8.30E-17	1.57E-16
	28	intro-example-mixed	1.66E-16	1.09E-16	1.64E-16
	29	sineOrder3	3.00E-16	2.80E-16	2.04E-16
	30	bsplines3	2.20E-16	1.66E-16	2.26E-16
	31	NMSEexample310	1.11E-13	N/R	1.14E-13
	32	NMSEproblem343	4.57E-14	N/R	7.99E-14
double-variate	33	nonlin2	✗	2.73E-14	1.78E-05
	34	hypot	✗	N/R	1.59E-16
	35	x_by_xy	✗	1.11E-16	2.15E-16
	36	NMSEproblem335	✗	N/R	1.96E-03
	37	NMSEproblem332	✗	N/R	1.22E-02
	38	floudas3	✗	1.11E-16	3.17E-16
	39	himmibeau	✗	4.41E-16	9.74E-06
	40	carthesianToPolar_theta	✗	N/R	2.69E-16
	41	NMSEexample33	✗	N/R	1.71E-03
	42	i4	✗	N/R	1.61E-16
	43	i6	✗	N/R	1.86E-03
	44	test03_nonlin2	✗	1.11E-16	2.59E-16
	45	polarToCarthesian,x	✗	1.11E-16	3.43E-02
	46	polarToCarthesian,y	✗	1.11E-16	2.50E-01
	47	NMSEproblem346	✗	N/R	2.03E-12
	48	complex_square_root	✗	N/R	1.84E-16
	49	complex_sine_cosine	✗	N/R	8.05E-15
	50	jetEngine	✗	8.32E-14	2.14E-02
	51	rump's example, with pow	✗	5.89E-16	5.44E-03
	52	doppler1	✗	3.37E-16	5.64E-16
triple-variate	53	sum	✗	N/R	1.97E-16
	54	rigidBody1	✗	3.66E-14	9.73E-12
	55	rigidBody2	✗	1.70E-10	1.77E-10
	56	turbine1	✗	6.98E-14	4.67E-16
	57	turbine2	✗	3.39E-16	5.03E-11
	58	turbine3	✗	1.87E-13	4.71E-12
	59	test01_sum3	✗	N/R	1.97E-16

Comparison of overhead



Achieves remarkable speedup over existing tools

- **2.1x faster** than Herbie
- **17.7x faster** than EIFFEL
- **5169x faster** than S3fp

Even surpasses HSED on single-parameter cases with **2.3x** improvement
Maintains superior detection accuracy alongside high efficiency

Conclusion

FPEPD: fist scalable, FP-aware search that locates maximum errors in both single- and multi-parameter functions

Key innovations

- Multi-level error-classfication –> adaptive sampling
- Three FP-specific point strategies: **gradient**, **special-value**, **mantissa-pattern**
- Efficient parallel implementation
- Automatic strategy adaptation

Takeaways and future work

Takeaways

- FP representation matters ->representation-aware search outperforms blind methods
- Adaptive partition + parallelism delivers both accuracy and speed
- FPEPD is ready for real-world numerical code auditing

Future work

- Replace MPFR library with error-free transformations + randow execution for faster high-precision refs
- Extend to loops, conditionals; support full programs, not just expressions

THANK YOU!