Instability Problem

```
float x, z;
x = input();
z = x*x*x*x–4*x*x*x
+6*x*x–4*x+1;
if (z > 0.5)
    printf (“hit”);
else
    printf (“miss”);
```

x: 1.84089642

$z = (x-1)^4$

actual execution	ideal execution
z: 0.4999...	z: 0.50..01129
miss	hit

Unstable Execution

actual execution
(w/ limited precision)

ideal execution
(w/ infinite precision)

discrete differences

- Control flow differences (predicate outcome)
  - if (z > 0.5) … else …
- Array index differences (type cast)
  - k = (int) f(x); z = A[k];

Possible Solutions:

```
double x, z;
x=input();
z=x*x*x*x–4*x*x*x+
   6*x*x–4*x+1;
if (z>0.5)
  printf("hit");
else
  printf("miss");
```

```
double x, z;
x=input();
z=(x-1) * (x-1) *
   (x-1) * (x-1);
if (z>0.5)
  printf("hit");
else
  printf("miss");
```

x: 1.840896415…

actual execution *differs* ideal execution

(a)

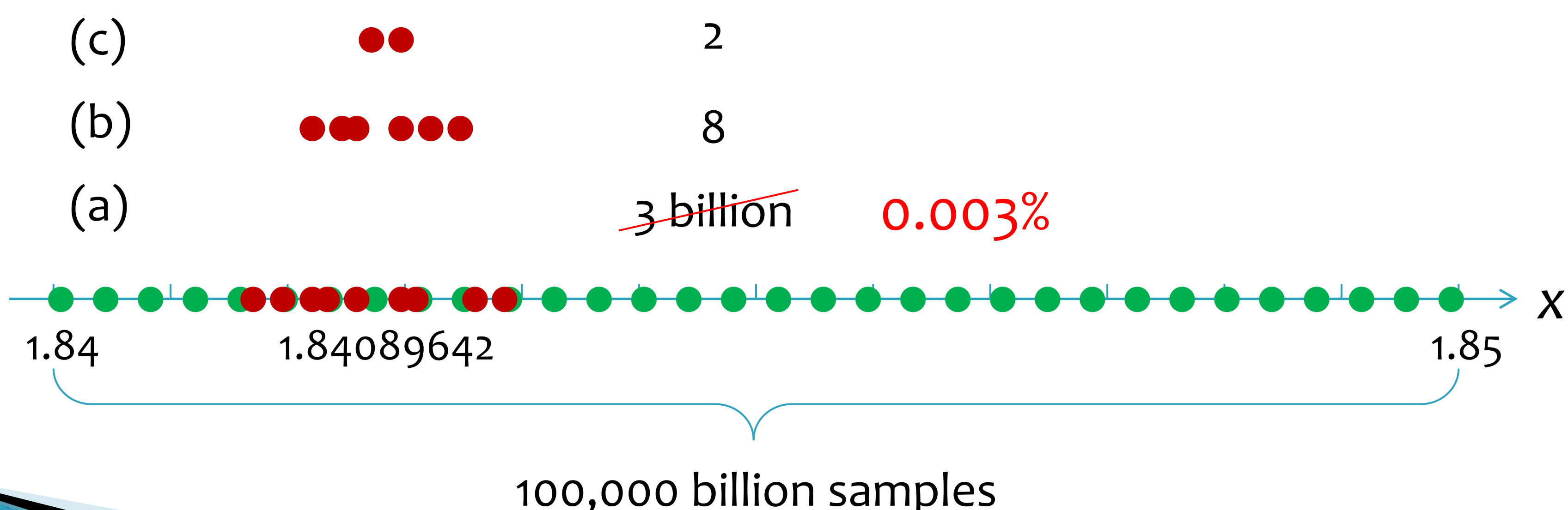
```
float x, z;
x = input();
z = x*x*x*x–
4*x*x*x +6*x*x–
4*x+1;
if (z > 0.5)
    printf (“hit”);
else
    printf (“miss”);
```

(b)

```
double x, z;
x = input();
z = x*x*x*x–
4*x*x*x +6*x*x–
4*x+1;
if (z > 0.5)
    printf (“hit”);
else
    printf (“miss”);
```

(c)

```
double x, z;
x = input();
z = (x–1)*(x-1)*
    (x-1)*(x-1);
if (z > 0.5)
    printf (“hit”);
else
    printf (“miss”);
```



Observations

- The instability problem cannot be completely evaded.
- A FP program only suffers from the instability problem for a very small input range.
- For a particular input, we can evade the problem by using high precision.

Our idea

- Using lightweight runtime predictor to predict if an execution is stable. If not, we switch to the high precision on demand.
  - This is different from handling traditional functional bugs.

Our approach

- Execute the program in normal precision;
- Monitor the growth of the *relative error* at runtime.

$$\hat{x} \quad = \quad x \quad + \quad \widehat{\Delta_x}$$

$$0.9997 = 0.999700009822 + (-0.000000009822)$$

ideal value actual value error

- Def.1: The *relative error* of a variable x, denoted by Δ_x, is computed as $\left|\widehat{\Delta_x}/x\right|$.

PURDUE UNIVERSITY

Δ_x: relative error of x, i.e. $\left|\frac{\widehat{\Delta_x}}{x}\right|$

$\boxed{z_1} = x*x*x*x - 4*x*x*x;$

- $\boxed{z_2} = z_1 + 6*x*x;$
- $\boxed{z_3} = z_2 - 4*x;$
- $\boxed{z_4} = z_3 + 1;$

$\boxed{z_5} = z_4 - 0.5;$

if $(\boxed{z_5} > 0)$ ...

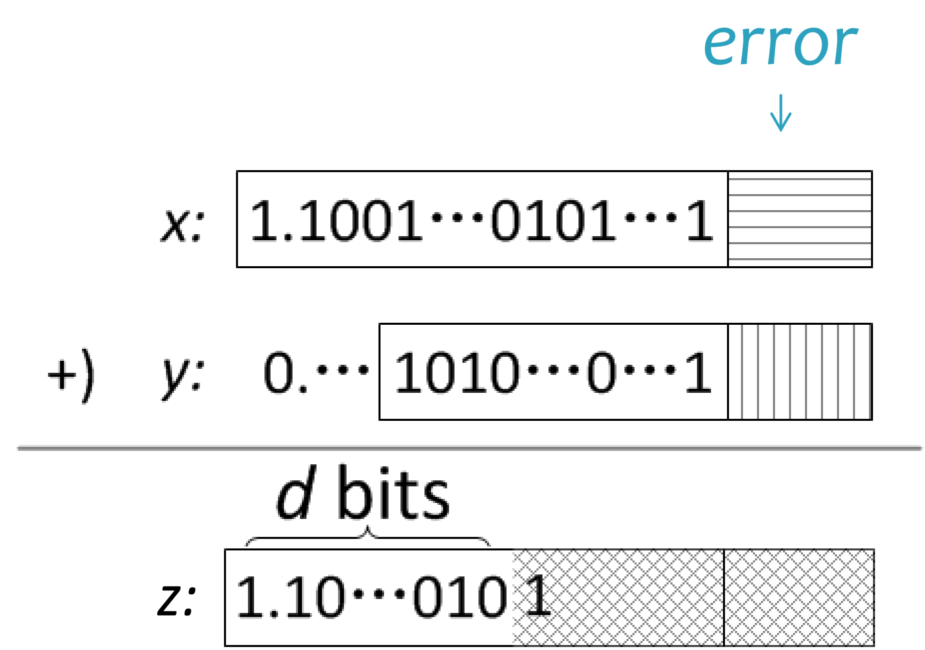
$\boxed{z_5}$ Δ_x is large

$\boxed{z_1}$ Δ_x is small

- Report unstable if it may lead to discrete differences.

Relative Error Inflation in Addition/Subtraction

e_x: the exponent of x



- The relative error may likely become 2^d times larger than the relative errors of the operands.

$$\boxed{d = \max(e_x, e_y) - e_z}$$

$\boxed{z_1} = x*x*x*x - 4*x*x*x;$

- $\boxed{z_2} = z_1 + 6*x*x;$
- $\boxed{z_3} = z_2 - 4*x;$
- $\boxed{z_4} = z_3 + 1;$

$\boxed{z_5} = \boxed{z_4} - \boxed{0.5};$

$z_4 = 0.4999\ldots,$

$z_5 = -0.0000019..$

d=17

$\text{if } (\boxed{z_5} > 0) \ldots$

- Our approach is to detect and propagate the relative error inflation.

Propagation Rules

Case 1: Both of the operands are tagged red.

- $\boxed{z} = \boxed{x} + \boxed{y}$;

PURDUE
UNIVERSITY

e_x: the exponent of x

Propagation Rules (cont.)

Case 3: Only one of the operands is tagged red.

- $z = \boxed{x} + \boxed{y}$;

$\boxed{}$? $\boxed{}$

$\boxed{z}$ if $(e_x - e_y > \tau_s)$;

$\boxed{z}$ otherwise.

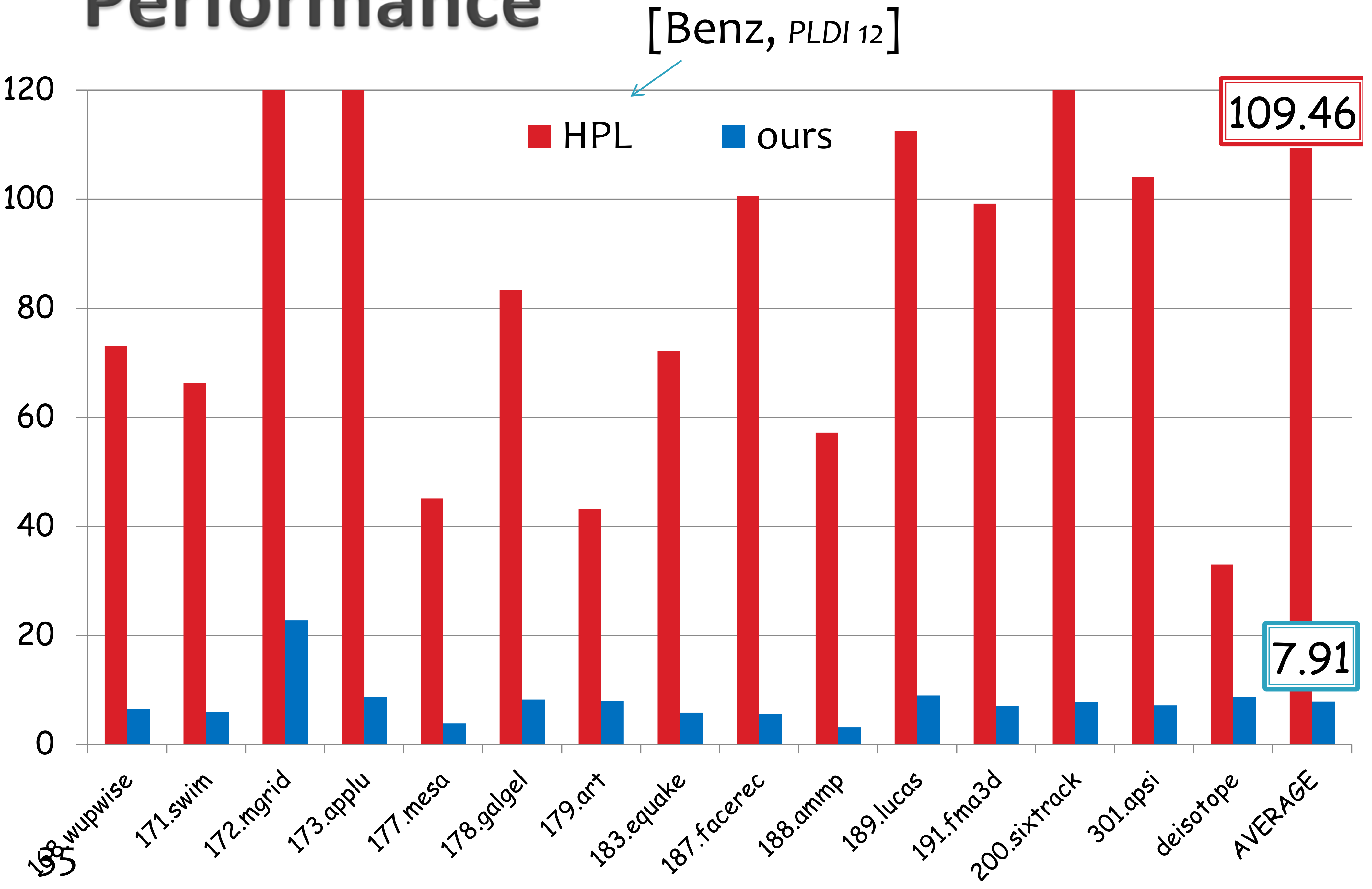
$\tau_s = 4$

3 $\boxed{z_6} = \boxed{z_5} - \boxed{90}$; z_5=1E-10

4 **if** ($\boxed{z_6}$ > 0) …

PURDUE UNIVERSITY

Performance



Effectiveness

approach	# of cases	%	detected range
	1E+14		[0.5900, 0.6000]
HPL	849	8.49E-10%	[0.596265750063108, 0.596265750064926]
ours (τ_C=36)	59457611	5.95E-5%	[0.596265**720335802**, 0.596265**779793411**]
ours (τ_C=40)	2716295	2.72E-6%	[0.596265**748204800**, 0.596265**751922657**]
ours (τ_C=44)	232165	2.32E-7%	[0.596265**749946110**, 0.596265**750181511**]
ours (τ_C=48)	12613	1.26E-8%	[0.59626575**0056901**, 0.59626575**0066600**]
ours (τ_C=52)	296	2.96E-10%	[0.596265750063**257**, 0.596265750065**373**] †

187.facerec

PURDUE UNIVERSITY