混合精度优化

于恒彪
国防科技大学计算机学院

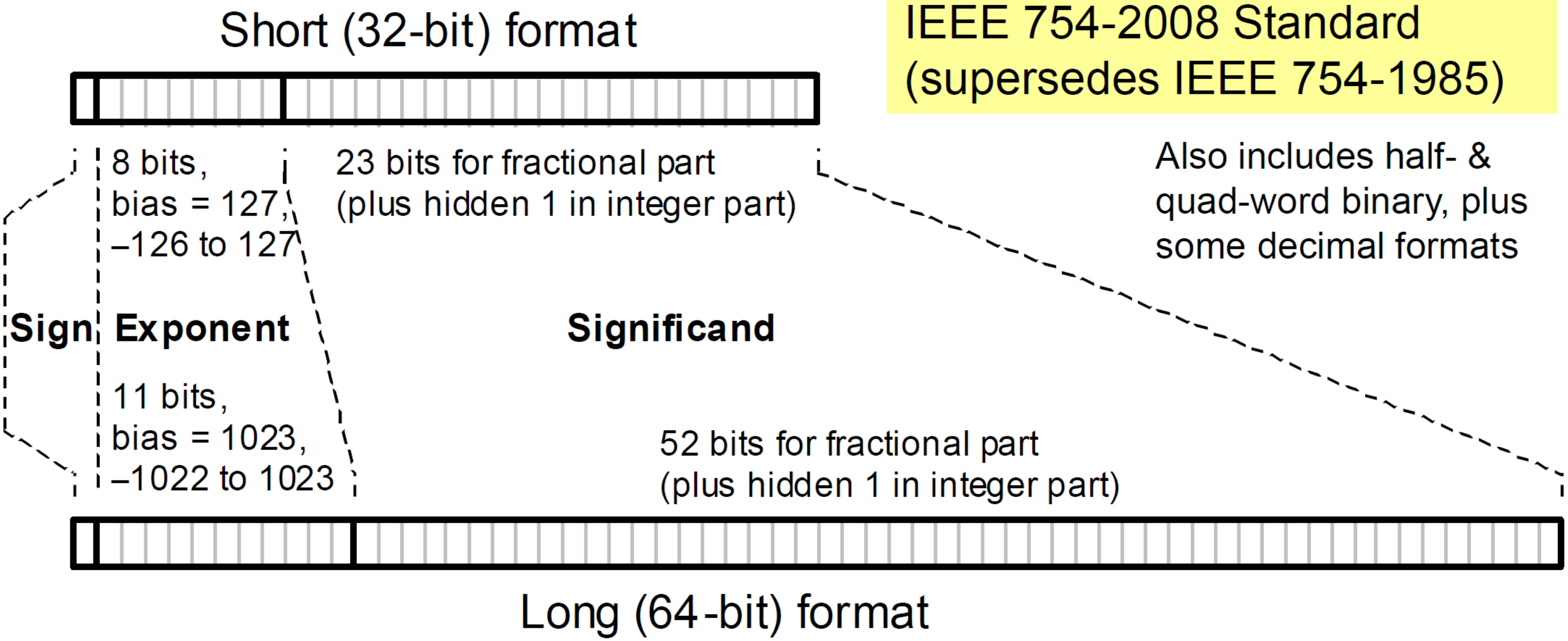
2025.07.12

一、混合精度优化概念

浮点数

- 浮点数是实数在计算机内表示的事实标准
- IEEE-754定义浮点数：$(-1)^S \times M \times 2^E$

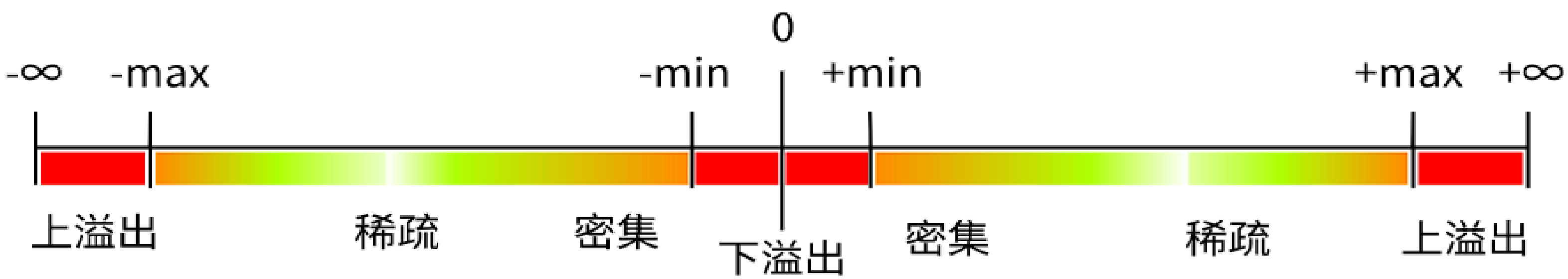
IEEE 754-2008 Standard
(supersedes IEEE 754-1985)



Also includes half- & quad-word binary, plus some decimal formats

浮点数

- 浮点数有限精度编码

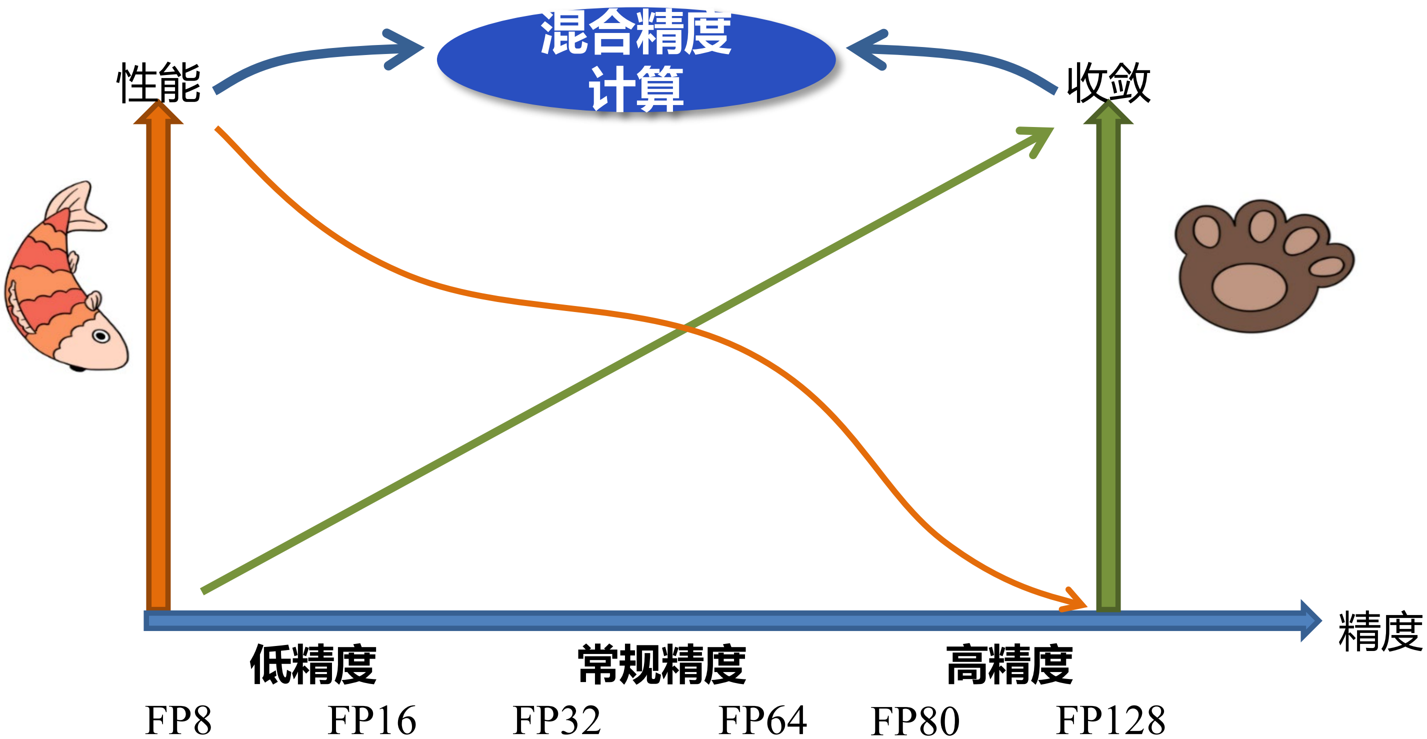


- 舍入误差不可避免：
  - fp32： 1.0E10+3.14=1.0E10
- 当前计算机软硬件支持各种不同精度
  - fp16、bf16、fp32、fp64、fp80、fp128、…

混合精度

- 不同精度的计算准确性不同
  - 舍入误差上限: fp32*2^{-23} / fp64*2^{-52}
- 不同精度的计算性能不同
  - 基础函数：sinf << sin
  - SIMD指令：单精度向量通路数是双精度的两倍
  - 运算指令：fdivs << fdivd
- 不同精度的存储、通信开销不同
  - Cache缓存数、网络传输数据量

混合精度优化



混合精度优化

程序中不同计算语句对于精度需求不一样

混合精度优化：在确保结果满足给定精度需求（RelErr<epsilon）下合理降低浮点操作精度来提升性能

- Algorithm-specific
  - **迭代求解器 Ax=B**
  - **精度补偿**
- Algorithm-independent
  - 静态分析
    - **面向浮点表达式**
    - **可扩展性差、近似保守**
  - 动态分析
    - **面向程序**
    - **可扩展性好**

Precision Tuning Example

```
long double fun(long double p) {
long double pi = acos(-1.0);
long double q = sin(pi * p);
     return q;
}

void simpsons() {
long double a, b;
long double h, s, x;
const long double fuzz = 1e-26;
const int n = 2000000;
…
L100:
    x = x + h;
    s = s + 4.0 * fun(x);
    x = x + h;
   if (x + fuzz >= b) goto L110;
    s = s + 2.0 * fun(x);
    goto L100;
L110:
    s = s + fun(x);
    //final answer:(long double)h *s/3.0
}
```

Original Program

Tuned Program

Error threshold 10^{-8}

Precision Tuning Example

```
long double fun(long double p) {
long double pi = acos(-1.0);
long double q = sin(pi * p);
    return q;
}

void simpsons() {
long double a, b;
long double h, s, x;
const long double fuzz = 1e-26;
const int n = 2000000;
…
L100:
    x = x
    s = s + 4.0 * fun(x);
    x = x + h;
   if (x + fuzz >= b) goto L110;
    s = s + 2.0 * fun(x);
    goto L100;
L110:
    s = s + fun(x);
    //final answer:(long double)h *s/3.0
}
```

```
long double fun(double p) {
double pi = acos(-1.0);
long double q = sinf(pi * p);
    return q;
}

void simpsons() {
float a, b;
double s, x; float h;
const long float fuzz = 1e-26;
const int n = 2000000;
    s = s + 4.0 * fun(x);
    x = x + h;
   if (x + fuzz >= b) goto L110;
    s = s + 2.0 * fun(x);
    goto L100;
L110:
    s = s + fun(x);
    //final answer:(long double)h *s/3.0
}
```

Tuned program runs 78.7% faster!

二、经典动态混合精度优化

基于搜索的混合精度优化

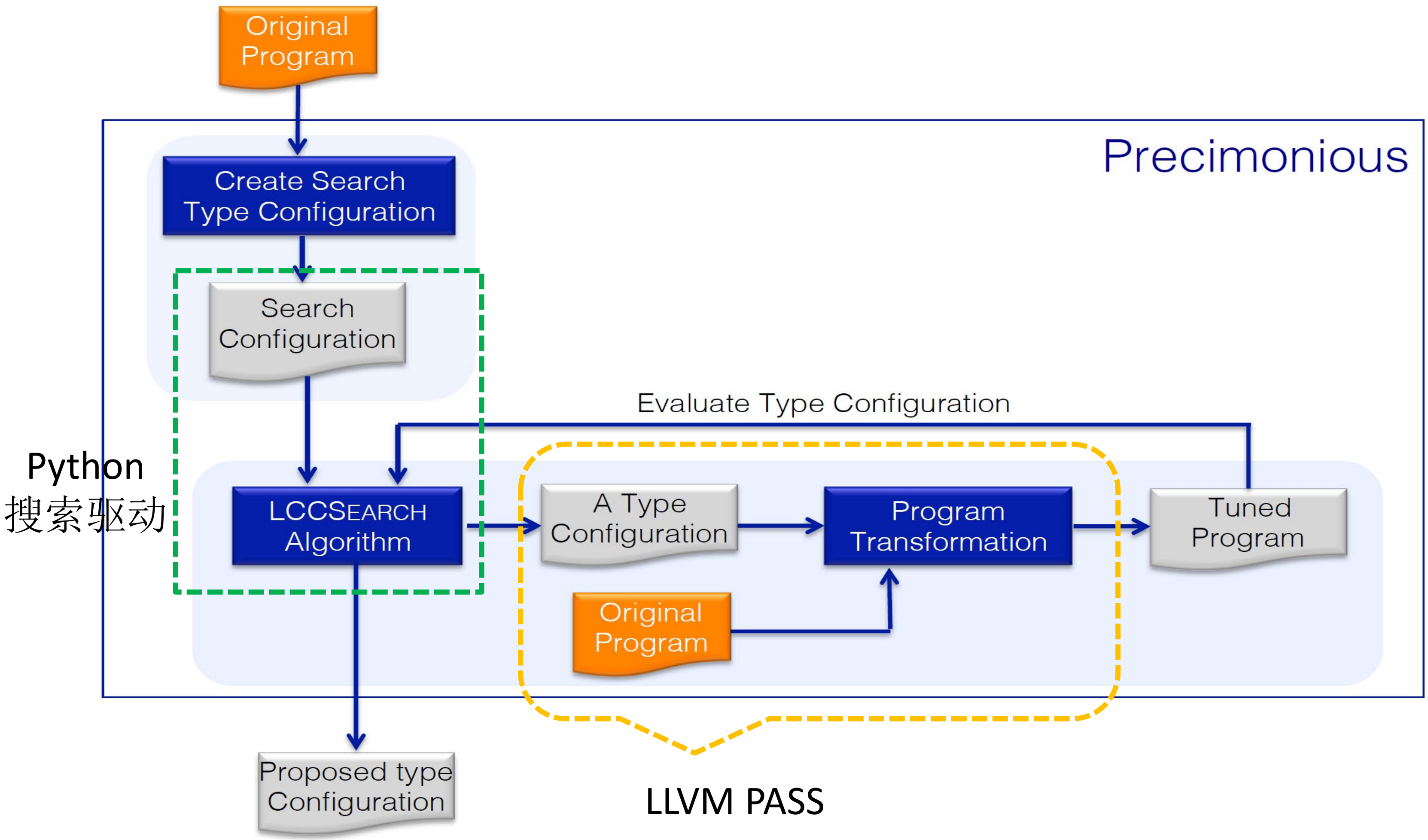
- 分析大规模浮点程序非常困难
  - 涉及大量数值问题：误差累积、传播等
  - 大多数程序员并非浮点理论专家

- 普遍做法: 统一使用高精度
  - 结果正确，但引入高昂的计算开销

- 自动动态浮点精度优化
  - 搜索确定浮点变量的类型 → 变量精度类型配置
  - 好的精度类型配置→结果准确性满足需求且性能提升

- 配置空间巨大
  - 假设存在fp32、fp64、fp128三种类型
  - 程序n个浮点变量，搜索空间3^n
  - 程序10个浮点变量，每次运行1s，搜完要16.4h

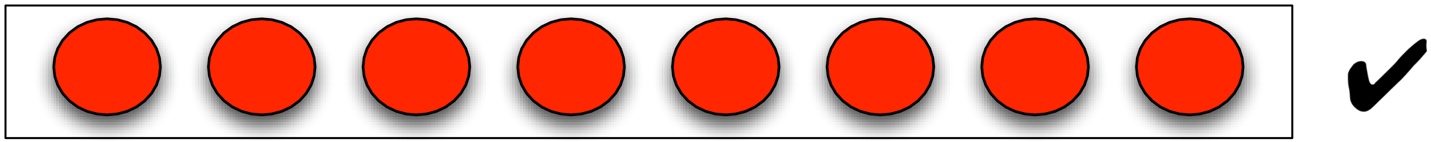
Precimonious[SC'13]



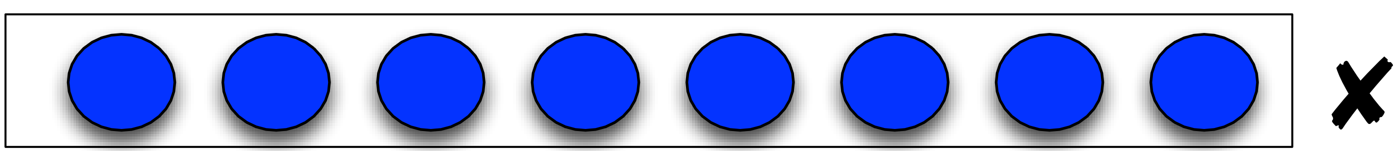
https://github.com/corvette-berkeley/precimonious

基于Delta-Debugging的配置搜索

double
precision

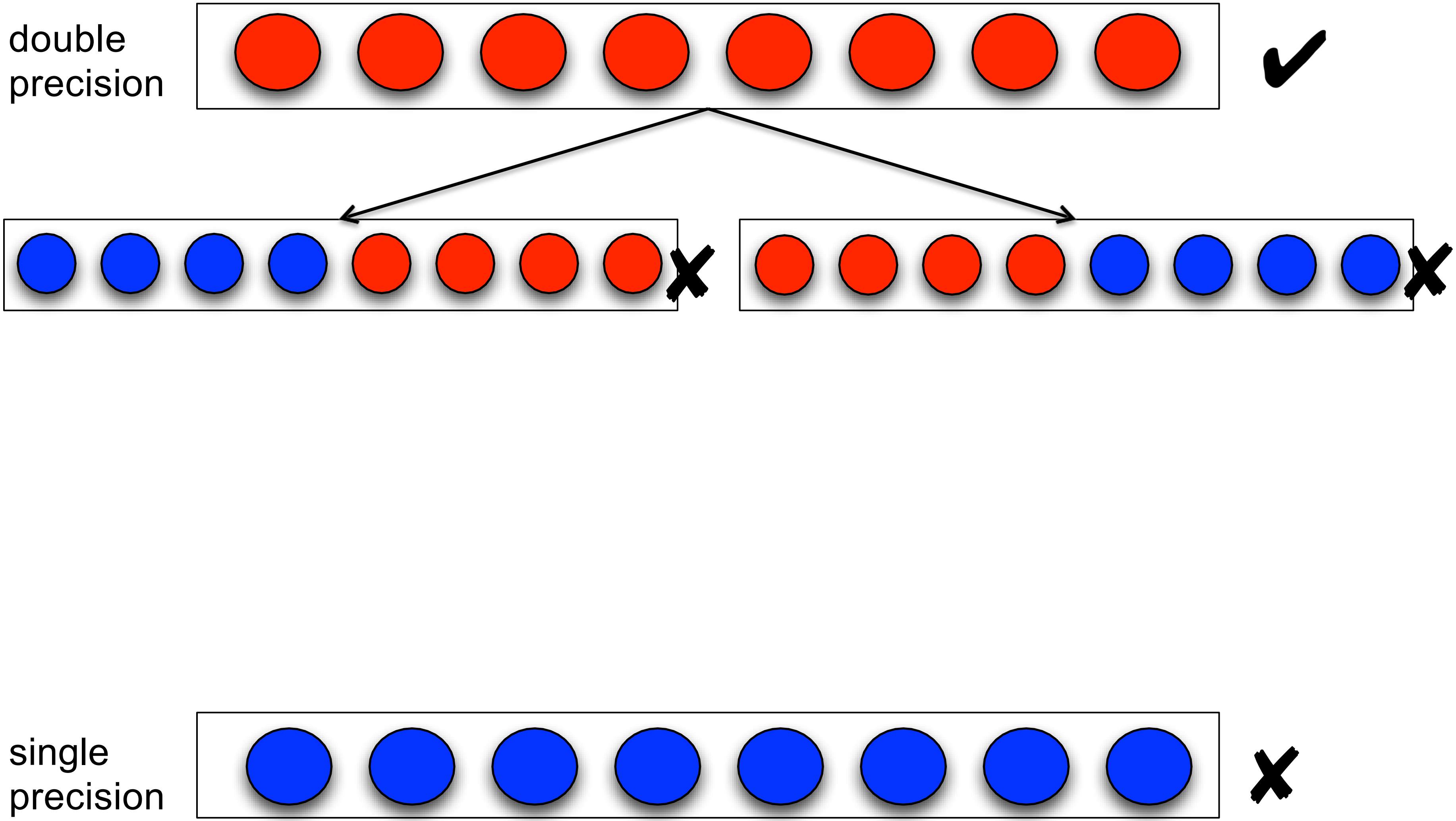


single
precision



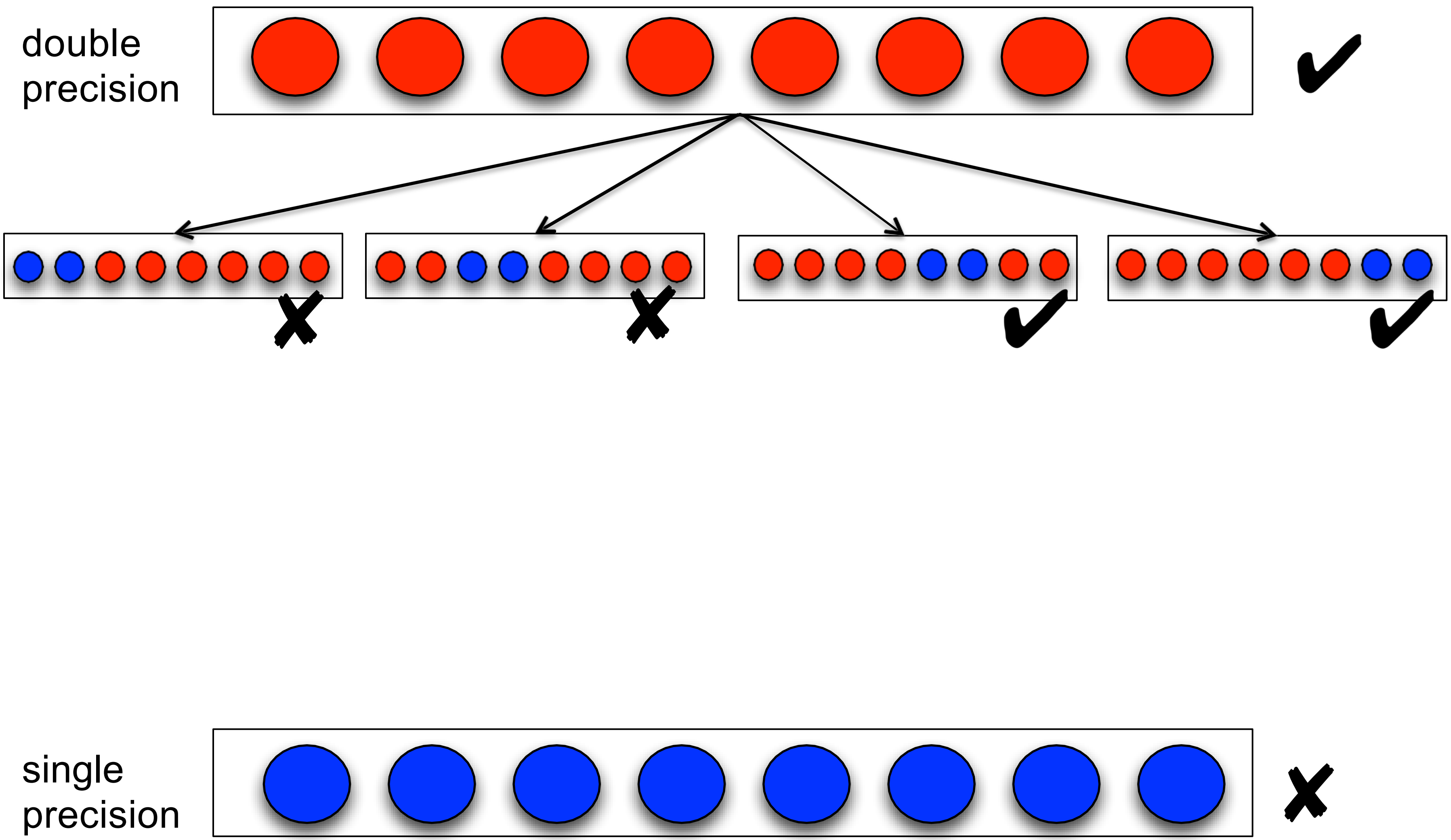
Precimonious[SC’13]

基于Delta-Debugging的配置搜索



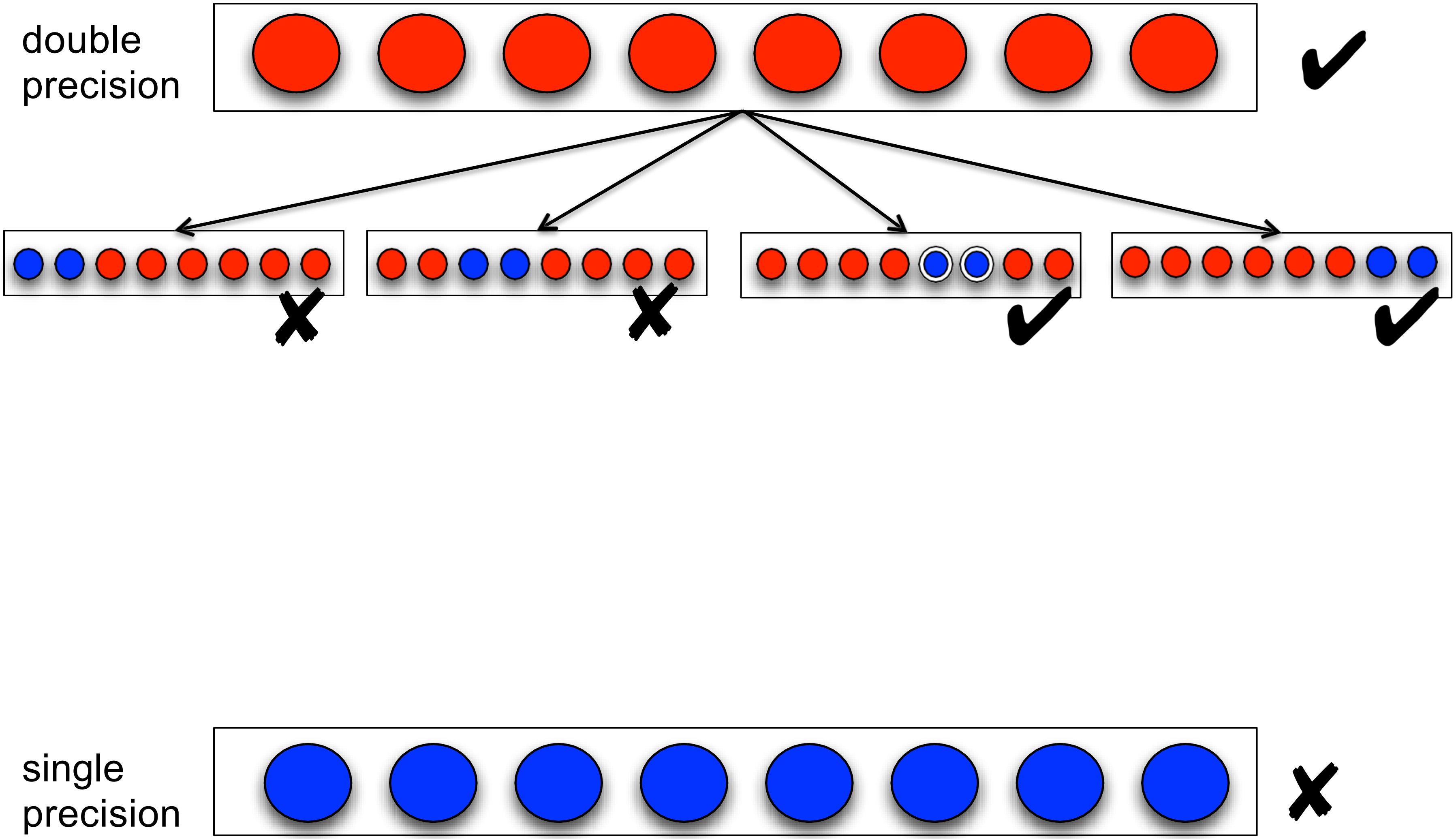
Precimonious[SC'13]

基于Delta-Debugging的配置搜索



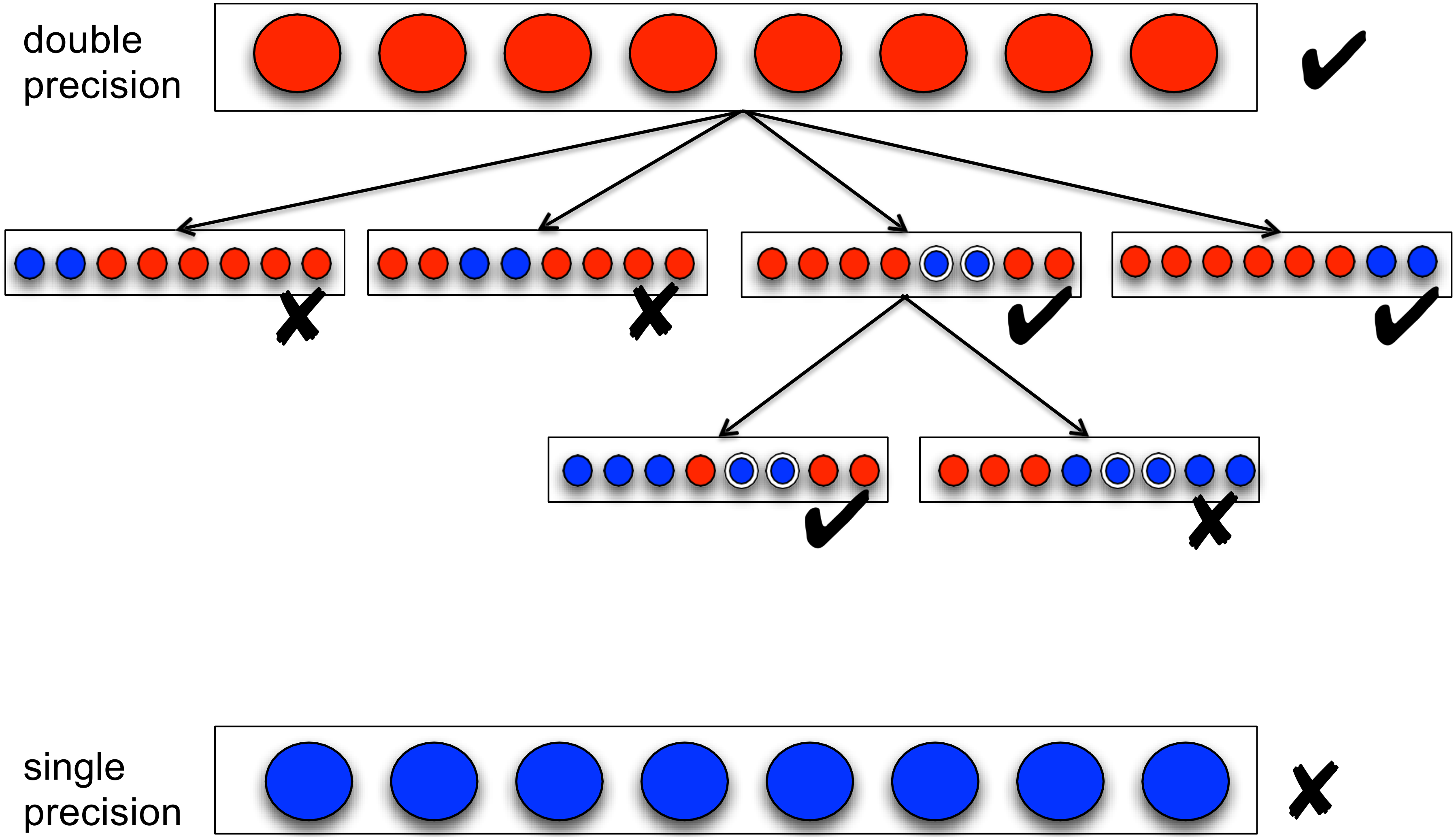
Precimonious[SC'13]

基于Delta-Debugging的配置搜索



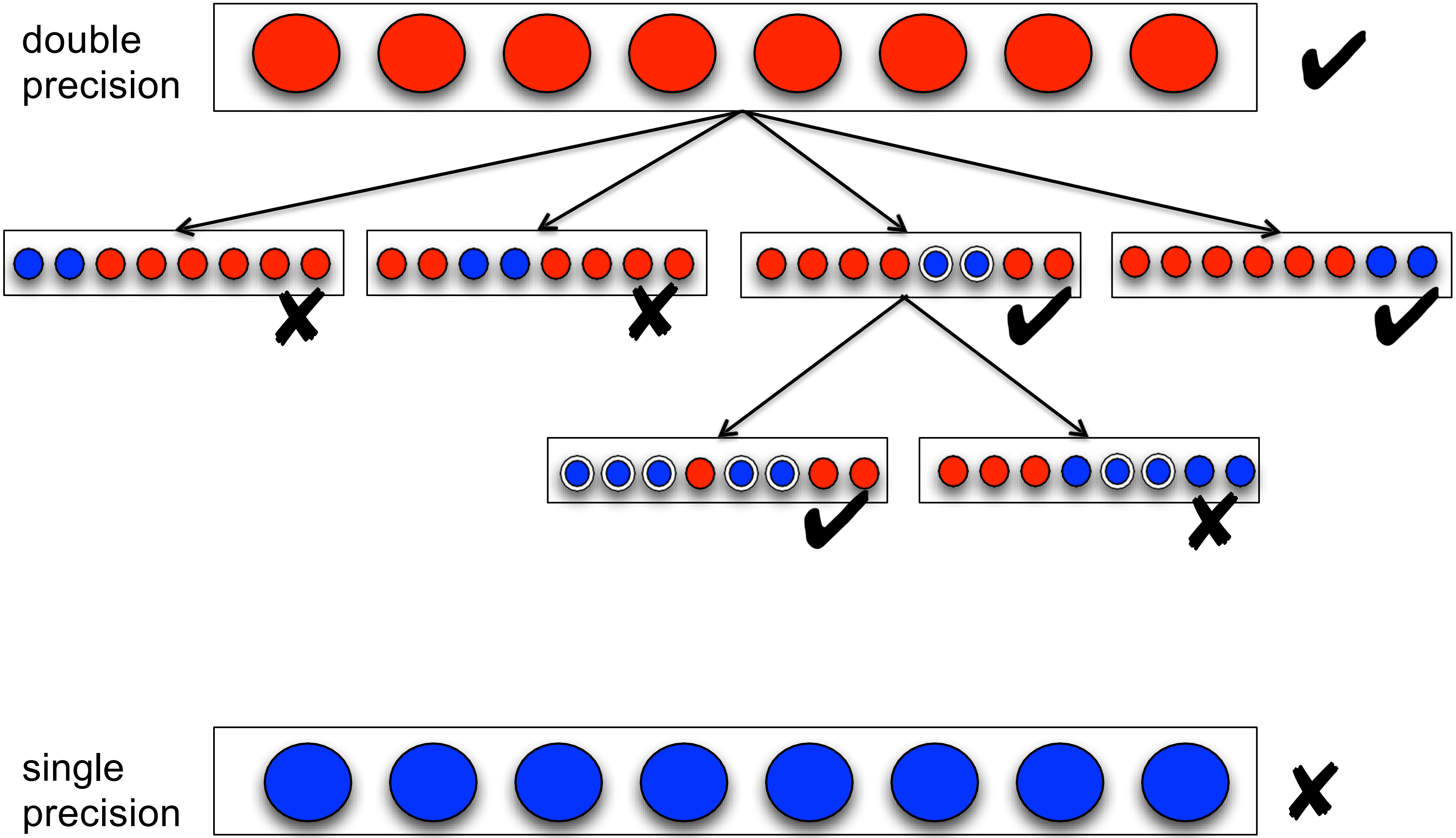
Precimonious[SC'13]

基于Delta-Debugging的配置搜索



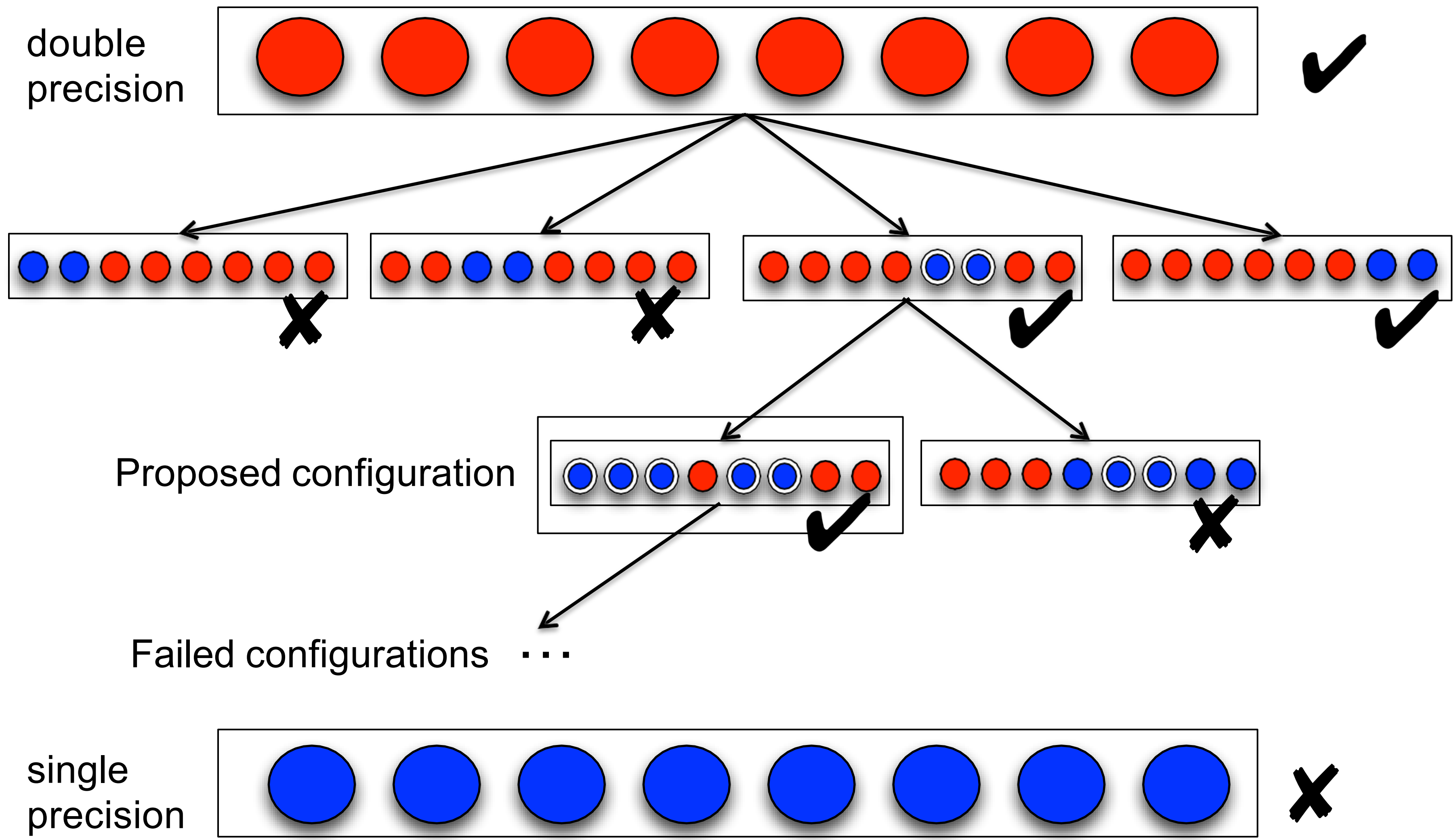
Precimonious[SC’13]

基于Delta-Debugging的配置搜索



Precimonious[SC’13]

基于Delta-Debugging的配置搜索



Precimonious[SC’13]

动态可扩展性？

- 搜索空间能否缩小?
  - 去除冗余变量：Blame Analysis[ICSE'16]
- 搜索算法能否改进?
  - 变量等价类划分：HiFPTuner[ISSTA'18]
  - 基于误差分析的搜索：Adapt[SC'18]
- 精度配置评价机制能否改进?
  - GPU程序性能模型：AMPT-GA[ISC'19]
  - 图神经网络模型：FPLEARNER[ICSE'24]

Blame[ICSE'16]:冗余变量删除

- BlameSet : 哪些变量降低精度对结果准确性不影响
  - Concrete + Shadow Execution
- Shadow Execution
  - 对浮点操作进行各种精度配置组合运算
  - 动态检查结果是否满足给定误差

r3=r2-t1 （oracle为全部高精度计算结果）

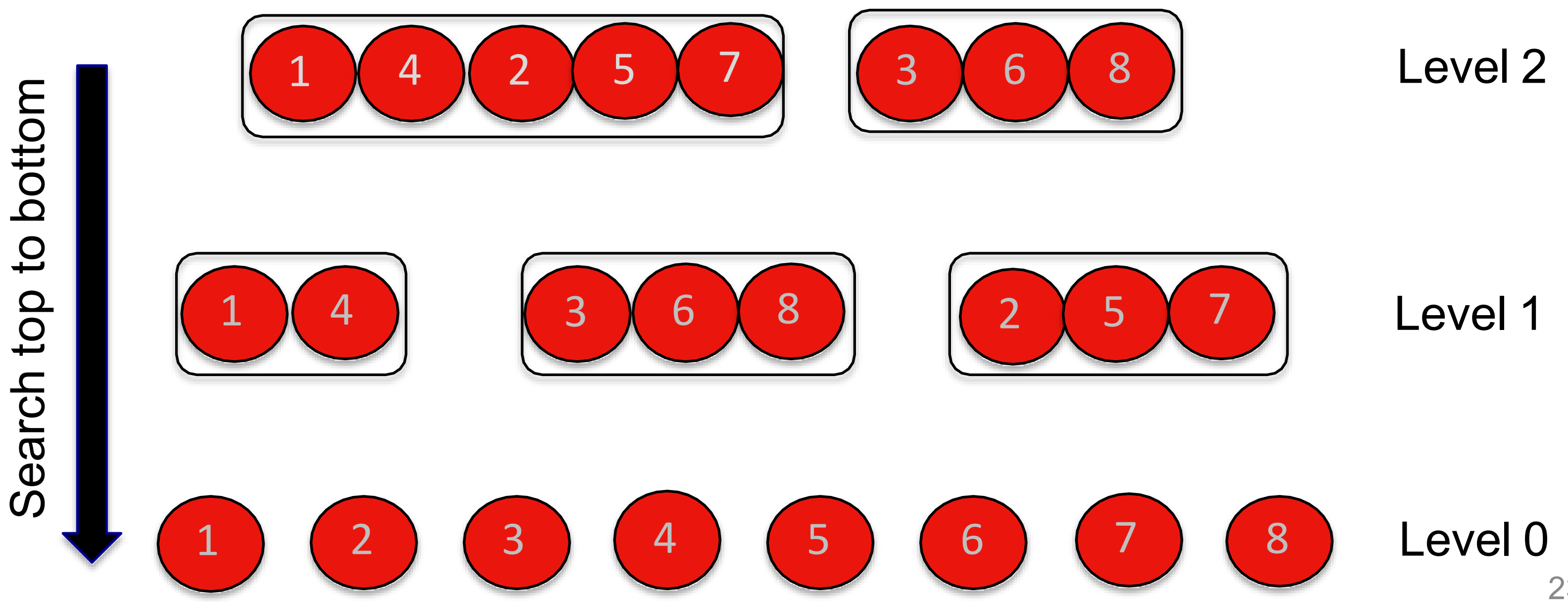
Prec	r2	t1	r3	S?
(fl,fl)	6.8635854721	7.3635854721	-0.5000000000	No
(fl,db$_8$)	6.8635854721	7.3635856000	-0.5000001279	No
(fl,db)	6.8635854721	7.3635856800	-0.5000002079	No
(db$_8$,fl)	6.8635856000	7.3635854721	-0.4999998721	No
(db$_8$,db$_8$)	6.8635856000	7.3635856000	-0.5000000000	No
...	...	...	...	...
(db,db)	6.8635856913	7.3635856800	-0.4999999887	Yes

Blame[ICSE'16]:冗余变量删除

- BlameSet计算:
  - 基于Merge的动态数据流分析
  - (fp32,fp32) ∪ (fp64,fp64) = (fp64,fp64)
- 搜索空间优化
  - 收集那些配置为fp32的变量作为BlameSet
  - 在搜索空间中去除掉BlameSet

HiFPTuner[ISSTA’18]:变量等价类划分

- 基于程序代码信息优化搜索：过多的type casting
- 搜索由黑盒转向白盒
  - 操作紧密关联的变量划分成一个等价类→ 赋予同一精度类型
  - 等价类个数少于变量个数→ 更快的搜索
  - 有效避免类型转换 → 更大的性能提升

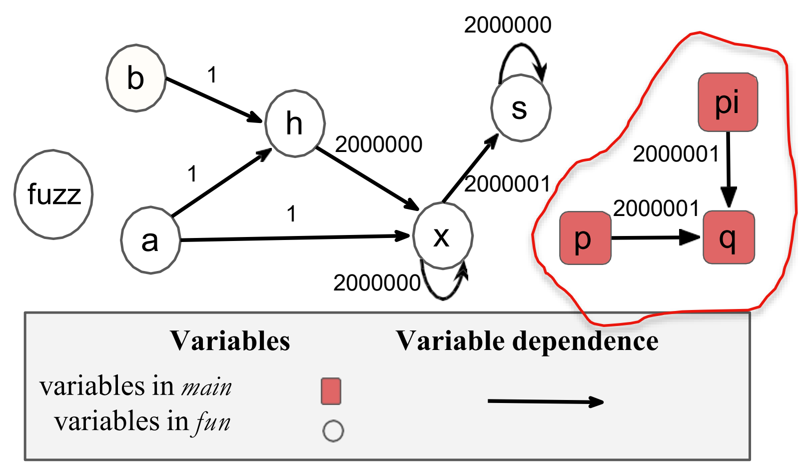


HiFPTuner[ISSTA’18]:变量等价类划分

```
long double fun(long double p) {
  long double pi =acos(-1.0);
  long double q = sin(pi * p);
  return q;
}

void simpsons() {
  long double a, b;
  // subinterval length, integral approximation, x
  long double h,s,x;
  const long double fuzz = 1e-26;
  const int n = 2000000;
  a = 0.0;
  b = 1.0;
  h = (b - a) / n;
  x = a;
  s = fun(x);
  L100:
    x = x + h;
    s = s + 4.0 * fun(x);
    x = x + h;
    if (x + fuzz >= b) goto L110;
    s = s + 2.0 * fun(x);
    goto L100;
  L110:
    s = s + fun(x);
    printf("%1.16Le\n", (long double)h * s / 3.0);
}
```

基于加权依赖图来对变量进行社区聚类



Adapt[SC'18]:误差分析引导搜索

- 泰勒展开式

$$
\begin{aligned}
y &= f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots, \\
&\approx f(a) + f'(a)(x-a).
\end{aligned}
$$

- 误差估计

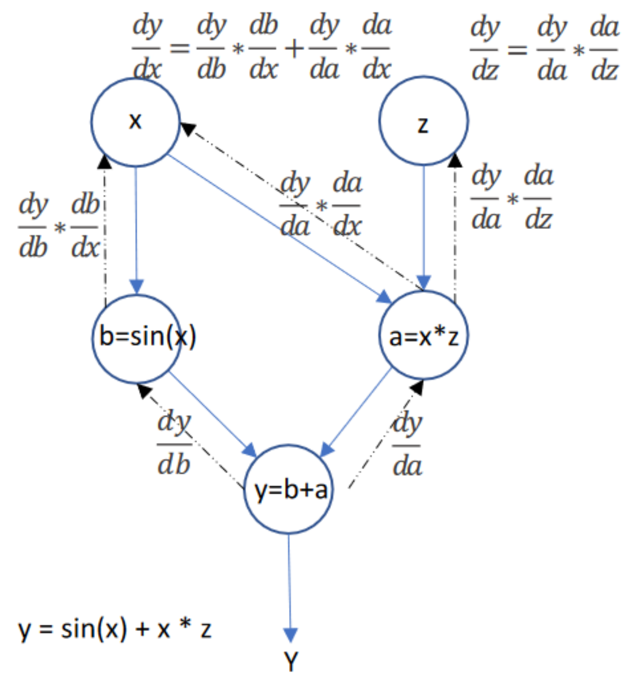
$$
\begin{aligned}
\Delta y &= |f(a+\Delta x) - f(a)|, \\
&\approx |f(a) + f'(a)(a+\Delta x - a) - f(a)|, \\
&= |f'(a)(\Delta x)|.
\end{aligned}
$$

- 精度降低带来的舍入误差评估

$$
\begin{aligned}
\Delta y_{x_i} &= f'(x_i)\Delta x_i, \\
&= f'(x_i)(x_i - x_i^{lower}).
\end{aligned}
$$

Adapt[SC'18]:误差分析引导搜索

- 链式微分



$$
\begin{aligned}
\Delta y_{x_i} &= f'(x_i)\Delta x_i, \\
&= f'(x_i)(x_i - x_i^{lower}).
\end{aligned}
$$

- 基于误差分析的贪心搜索
  - 基于引入误差值对变量排序
  - 在误差容忍范围内：优先降低误差引入值低的变量精度

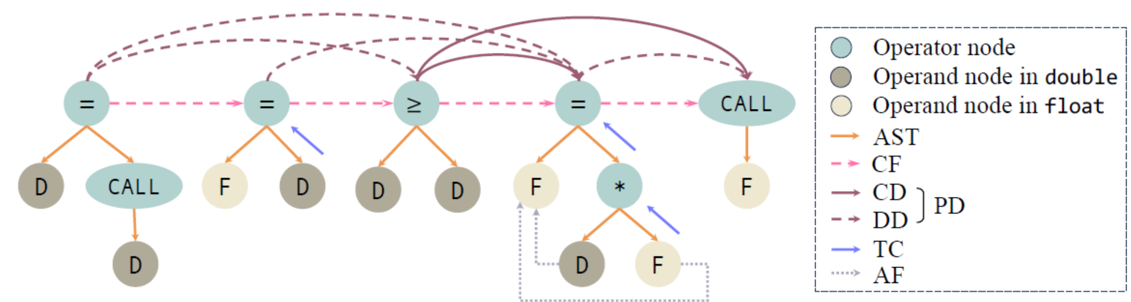
Adapt-GA[ISC'19]:性能模型

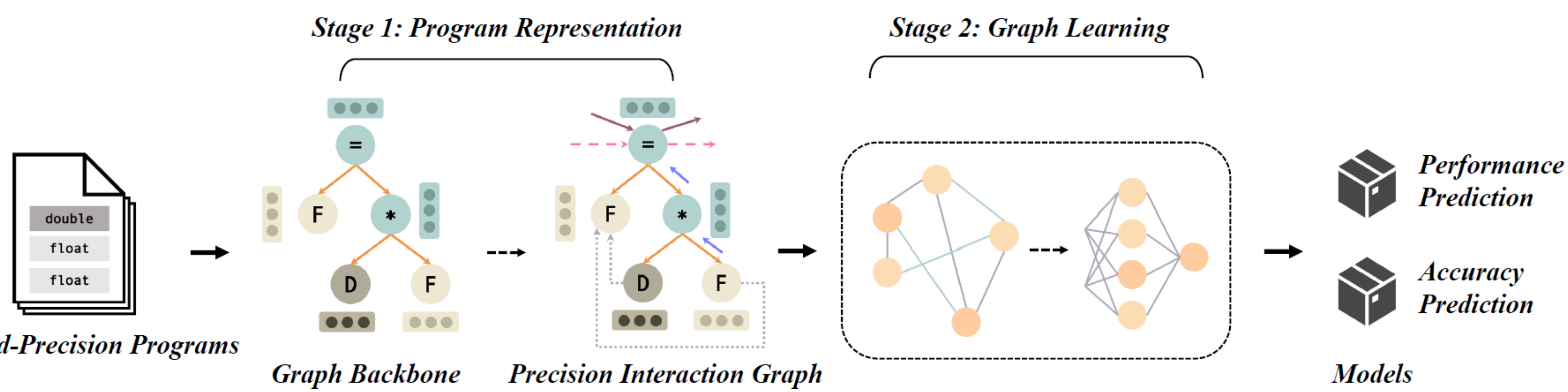
- 经典的自动混合精度优化
  - Search → Transform → Rerun
  - Rerun 开销巨大
- Adapt-GA
  - 面向GPU程序的混合精度优化
- 构建了性能模型
  - 基于依赖图构建
  - 建模主要浮点指令开销：casting、fadd、fmul等
  - 预测配置是否提升性能，避免过多Rerun

FPLearner[ICSE’24]:性能+精度模型

- 图神经网络(GNN)模型：预测配置是否满足精度/性能需求

```
void foo() {
  double a = sqrt(1.1);
  float b = 2.0;
  if (a >= 1.3952) {
    float c = a * b;
    update(c);
  }
}
```



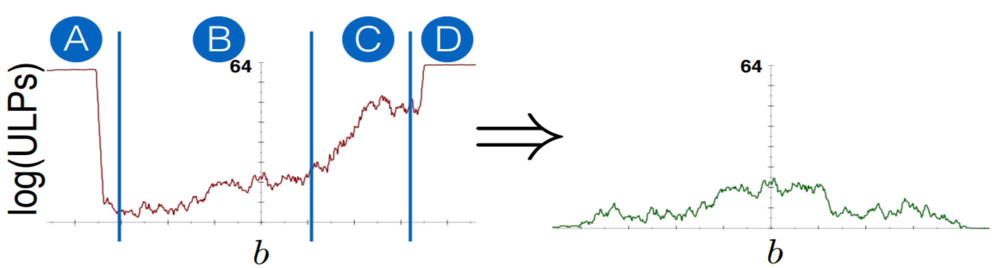


三、新的思路

结合表达式重写的精度优化

- 降低浮点表达式误差 Herbie[PLDI'15]
  - 基于采样的误差定位
  - 表达式重写+区间组合

$$\frac{-b+\sqrt{b^2-4ac}}{2a} \Longrightarrow \begin{cases} \frac{c}{b}-\frac{b}{a} & \text{if } b \in \text{A} \\ \frac{-b+\sqrt{b^2-4ac}}{2a} & \text{if } b \in \text{B} \\ \frac{2c}{-b-\sqrt{b^2-4ac}} & \text{if } b \in \text{C} \\ -\frac{c}{b} & \text{if } b \in \text{D} \end{cases}$$



结合表达式重写的精度优化

- 表达式重写+精度优化 Pherbie[arith’21]

$$\sqrt{\frac{e^{2x}-1}{e^{x}-1}}$$

(1) 在[-1,1]之间不准确

(2) x接近零时，除零导致NAN

(3) exp调用显著降低性能！

step1： 基于平方差重写 $\sqrt{e^{x}+1}$

step2： 基于0附近的泰勒展开重写 $\sqrt{2+x}$

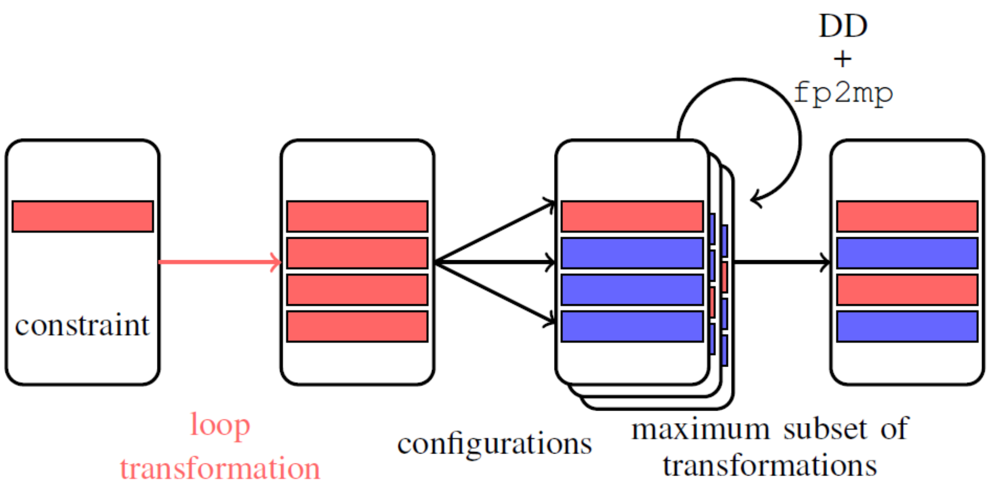
step3： 区间组合+精度优化

if $|x| \le 0.05$:
```
sqrt(2 + x)
```
else :
```
sqrt(expf(x) + 1.0f)
```

与$\sqrt{e^{x}+1}$精度相当，加速2倍

面向循环迭代空间的精度优化

- s1 and x 仍然需要long double
- 循环前段低精度，后段高精度
- DD为循环迭代空间合理分配精度



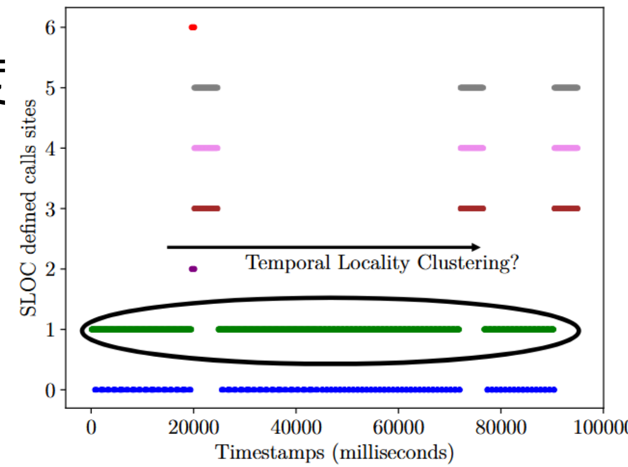

```c
long double fun(long double x) {
  long double pix = pi * x;
  long double result = sin(pix);
  return result;
}

int main( int argc, char **argv) {
  int l, i;
  const int n = 1000000;
  long double a = 0.0, b = 1.0, s1 = 0.0;
  long double h, x, tmp;
  h = (b - a) / (2.0 * n);
  x = a;
  s1 = fun(a);
  for(l = 0; l < n; l++) {
    x = x + h;
    s1 = s1 + 4.0 * fun(x);
    x = x + h;
    s1 = s1 + 2.0 * fun(x);
  }
  s1 = s1 + fun(b);
  tmp = h * pi / 3;
  s1 = s1 * tmp;
  printf("ans: %.15Le\n", s1);
  return 0;
}
```

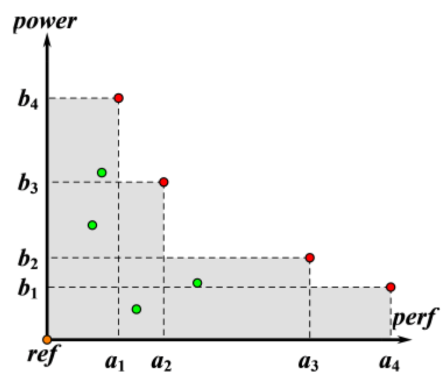
[Using loop transformation for precision tuning in iterative programs. arith'23]

OTHERS ...

- 结合程序运行时信息的精度优化PyFloT[SC’20]
  - 函数不同调用时机赋予不同精度
  - 基于运行时信息调用点聚类
  - 调用栈、时间局部性等信息



- 多目标搜索问题 [ISLPD’23]
  - 高性能+低功耗
  - 贝叶斯优化采样搜索



鲁棒性

- 混精优化后的程序是否满足给定精度需求
  - 基于给定/随机输入来运行优化后程序
  - 结果精度校验是否通过

  存在特定输入会使得混精优化程序运行结果不满足需求

- 提高混精优化后程序的鲁棒性
  - 满足输入域中大部分输入的结果精度需求
  - 基于高误差触发输入来运行优化后程序
  - 基于条件数（condition number）来引导产生高误差触发输入

Thank you!