SmartFloat: Numerical Error Estimator

Eva Darulová, Advisor: Viktor Kuncak

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Numerical Software Verification

Modern computing has adopted the floating point type as a default way to describe computations with real numbers. Thanks to dedicated hardware support, such computations are efficient on modern architectures. Reasoning about the resulting programs remains difficult because of the large gap between the finite representation and the ideal real mathematics.

Our library solution [1] provides two data types that provide a better understanding:

AffineFloat tracks a single computation in double precision and provides tight roundoff error bounds

SmartFloat tracks a range of floating-point values and computes besides the actual resulting interval also a guarateed upper bound on the roundoff error commited

This lets us determine, for example, that the second method on the right produces a more precise result.

```
> triangleTextbook(9.0, SmartFloat(4.8, 0.9),
   SmartFloat(4.8, 0.9))
[6.25, 8.62] +/- 1.10e-14

> triangleKahan(9.0, SmartFloat(4.8, 0.9),
   SmartFloat(4.8, 0.9))
[6.25, 8.62] +/- 3.11e-15
```

```
def triangleTextbook(a: SmartFloat,
      b: SmartFloat, c: SmartFloat): SmartFloat = {
  val s = (a + b + c)/2.0
  sqrt(s * (s - a) * (s - b) * (s - c))
}

def triangleKahan(a: SmartFloat,
      b: SmartFloat, c: SmartFloat): SmartFloat = {
  if(b < a) {
    val t = a
    if(c < b) { a = c; c = t }
    else {
      if(c < a) { a = b; b = c; c = t } else { a = b; b = t }
    }
  }
  else if(c < b) {
    val t = c; c = b;
    if(c < a) { b = a; a = t } else { b = t }
  }
  sqrt((a+(b+c)) * (c-(a-b)) * (c+(a-b))
                 * (a+(b-c))) / 4.0
}
```

Range Arithmetic

Traditionally: Intervals

Represent each variable as an interval $\bar{x} = [x_{lo}, x_{hi}]$ and perform each operation with directed rounding.

! Intervals suffer from the range-explosion problem, so that overapproximations become too pessimistic too quickly.

Our approach: Affine Arithmetic

Represent variables as affine forms [2]

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i \epsilon_i, \quad \epsilon_i \in [-1, 1]$$

whose represented range is

$$[\hat{x}] = [x_0 - rad(\hat{x}), x_0 + rad(\hat{x})], \qquad rad(\hat{x}) = \sum_{i=1}^{n} |x_i|$$

Unlike in interval arithmetic, correlations between variables are taken into account, and thus a much more precise analysis is possible.

The library uses two interpretations of affine arithmetic:

AffineFloat Each affine form representes *one* floating-point value and the deviations x_i the accumulated roundoff errors.

SmartFloat Each affine form representes a *range* of floating-point values. An additional affine form tracks the maximum roundoff error over this range.

Experiments

A comparison of our results obtained with AffineFloat (AF) to the results obtained with interval arithmetic (IA) on a set of scientific benchmarks [3, 4]:

Benchmark	rel. error AF	rel. error IA	IF/IA error
Scimark SOR 5 iter.	2.33e-14	4.87e-14	2.1
10 iter.	4.62e-13	3.21e-12	7.0
15 iter	8.85e-12	2.10e-10	23.7
20 iter	1.68e-10	1.38e-8	82.1
Nbody, 1s, h=0.01	1.58e-13	1.28e-13	0.8
1s, h=0.0156	1.04e-13	8.32e-14	0.8
5s, h=0.01	2.44e-10	7.17e-10	2.9
5s, h=0.015625	1.42e-10	4.67e-10	3.29
Spectral norm 2 iter	1.88e-15	7.13e-15	3.8
5 iter	4.93e-15	2.48e-14	5.0
10 iter	7.51e-15	5.62e-14	7.5
15 iter	1.01e-14	8.81e-14	8.7
20 iter	1.71e-14	1.19e-13	7.0

(For roundoff estimates over input ranges, like in the example above, intervals cannot be used at all.)

Integration

All that is needed to put our library into action is the replacement of Double types by SmartFloat or AffineFloat. The new types integrate seamlessly into Scala thanks to

- operator overloading
- special treatment of equals (==) for numeric types
- implicit conversions
- availability of the library functions

  log, expr, pow, cos, sin, tan acos, asin, atan, abs, max, min and the constants Pi and E.

Applications

Robustness A global flag is used to signal conditionals that cannot be unambiguously decided due to too large roundoff errors or input ranges.

Testing Using for example a binary search-like approach, we can use our library to generate a set of test interval inputs that will exercise all paths through a program, or refine paths based on roundoff errors.

User-defined Errors Apart from roundoff errors, SmartFloat can be used to propagate automatically user-added errors, e.g. approximations of errors of an iterative numerical method itself.

References

1 E. Darulova, V. Kuncak, On the Design and Implementation of SmartFloat and AffineFloat, EPFL-REPORT-164956, 2011.

2 L. H. de Figueiredo, J. Stolfi, Self-Validated Numerical Methods and Applications, Brazilian Mathematics Colloquium monograph, IMPA, 1997.

3 The Computer Language Benchmarks Game. http://shootout.alioth.debian.org/, Jan 2011.

4 R. Pozo and B. R. Miller. Java SciMark 2.0. http://math.nist.gov/scimark2/about.html, 2004.