CS 6210 Fall 2016

Bei Wang

Lecture 4
Floating Point Systems Continued

Take home message

1. **Floating point rounding**
2. **Rounding unit**
3. 64 bit word: double precision (IEEE standard word)
4. **Exact rounding**
5. **Guard digits**
6. General floating point systems
7. Spacing of floating point numbers
8. **Cancellation error**
9. **Good coding practice in floating point arithmetic**

The almost complete story on FP rounding

Decimal FP rounding (Integer rounding), Binary FP rounding
Possibly all you have to know about FP rounding
ON THE WHITEBOARD

Guard Digit

ON THE WHITEBOARD

Rounding Unit

ON THE WHITEBOARD

FP addition and multiplication

Rules, Fixings and Properties
ON THE WHITEBOARD

Good (bad) coding practices

For FP arithmetic

http://www.codeproject.com/Articles/29637/Five-Tips-for-Floating-Point-Programming

Rule 1: Don’t test for equality

```
double x;
double y;
...
if (x == y) {...}
```

```
double x = 10;
double y = sqrt(x);
y *= y;
if (x == y)
    cout << "Square root is exact\n";
else
    cout << x-y << "\n";
```

```
-1.778636e-015
```

```
double tolerance = ...
if (fabs(x - y) < tolerance) {...}
```

Equality test likely to fail: FP numbers may not match in their last bits...

Solution: set a tolerance...

Rule 2: FP has finite ranges

```
float f = 16777216;
cout << f << " " << f+1 << "\n";
```

```
x = 9007199254740992; // 2^53
cout << ((x+1) - x) << "\n";
```

```
x = 1.0;
y = x + 0.5*DBL_EPSILON;
if (x == y)
    cout << "Sorry!\n";
```

DBL_EPSILON: smallest positive number e s.t. 1+e!=1.

Here f = f+1 Because float (or double) has no precision left to represent f+1.

Solution: be aware of data range and be ready to handle exception

Rule 3: Use Logarithms to avoid overflow and underflow

How do you evaluate 200!/(190!10!)?

Solution: consider log(xy) = logx + log y, then log(n!) = log(n) + log(n-1) + ... log(1)

```
x = exp( logFactorial(200)
       - logFactorial(190)
       - logFactorial(10) );
```

```
double logFactorial(int n)
{
    double sum = 0.0;
    for (int i = 2; i <= n; ++i)
        sum += log((double)i);
    return sum;
}
```

Rule 4: Numerical operations do not always return numbers

```
x = DBL_MAX;
cout << 2*x << "\n";
```

DBL_MAX: largest number to be represented by a double in C++

What to do when NaN occurs?

Solution: exception handling.

Rule 5: SW specific rounding rules

MATLAB: y = round(x): nearest integer; when tie, away from zero

MATLAB: y=convergent(x): nearest integer; when tie, nearest even number

Mathematica: y = Round[x]: nearest integer; when tie, nearest even number

Python: round(x,n): round to the nearest; when tie, round away from zero

```
>>> round(2.675, 2)
2.67
```

Why not 2.68?

```
2.67499999999999982236431605997495353221893310546875
```

Decimal to binary FP approximation with exact value closer to 2.67.

https://docs.python.org/2/tutorial/floatingpoint.html

For your procrastination reading list

1. More details: http://www.codeproject.com/Articles/25294/Avoiding-Overflow-Underflow-and-Loss-of-Precision
2. William D. (Bill) Young Lecture slides http://www.cs.utexas.edu/~byoung/cs429/slides4-fp.pdf

Extra Notes

WHITEBOARD SUMMARY

The almost complete story on FP rounding

Decimal FP rounding (Integer rounding), Binary FP rounding

Possibly all you have to know about FP rounding

WHITEBOARD SUMMARY

IEEE Standard for FP arithmetic (IEEE 754-1985, newer 2008)

Five rounding rules (1-3 directed rounding, 4-5 round to nearest)

1. Round towards 0 “truncation”
2. Round towards +infinity “round up”
3. Round towards -infinity “round down”
4. Round to nearest, ties away from zero (typical for decimal FP)
5. **Round to nearest, ties to even** (default for binary FP, recommended for decimal FP): Round to the nearest value, if number falls **exactly** midway, it is rounded to the nearest value with an even (e.g. **zero for binary FP**) least significant bit.

Rule 5 prevents upwards or downwards bias in summing many #s that look like x.5, since round up and round down happen 50% of the time.

Example 1. Decimal FP round to integer

Rule	2.3	2.7	2.5	-2.3	-2.7	-2.5	3.5	-3.5
to 0	2	2	2	-2	-2	-2	3	-3
to +infinity	3	3	3	-2	-2	-2	4	-3
to -infinity	2	2	2	-3	-3	-3	3	-4
nearest, tie away from 0	2	3	3	-2	-3	-3	4	-4
nearest, tie to even	2	3	2	-2	-3	-2	4	-4

Example 2 decimal FP to nearest 100th (RULE 5)

Decimal FP	Rounded Value	Rule
1.2349999	**1.23**	**round down**
1.2350001	**1.24**	**round up**
1.2350000	**1.24**	**tie, round to even**
1.2450000	**1.24**	**tie, round to even**

Example 3. Binary FP round to nearest ¼ (2 bits fraction) RULE 5

value	Binary representation	Rounded value in binary	Rounded value in decimal	Action
2 + 3/32	10.00011	10.00	2	round down
2 + 3/16	10.00110	10.01	2+1/4	round up
2+7/8	10.11100	11.00	3	tie to even (up)
2+5/8	10.10100	10.10	2+1/2	tie to even (down)

Guard Digit

WHITEBOARD SUMMARY

Example: precision $p = 3$

Exact result: $10.1 - 9.93 = 0.17$

Computing without guard digit

$x = 10.1 = 1.01 \times 10^1$, $y = 0.99 \times 10^1$

$x - y = 0.02 \times 10^1$ wrong!

Computing with 1 guard digit

$x = 1.010 \times 10^1$, $y = 0.993 \times 10^1$

$x - y = 0.017 \times 10^1$ exact!

Theorems

Theorem 1: Using FP format with parameters b (base) and p (precision), counting differences using p digits, the relative error of the result can be as large as b-1.

Theorem 2: x and y are FP with base b and precision p, if subtraction is done with p+1 digits (i.e. 1 guard digit), then the relative error is upper bounded by 2 times machine precision.

http://www.sci.utah.edu/~beiwang/teaching/cs6210/Goldberg.pdf

Rounding Unit

WHITEBOARD SUMMARY

Rounding Unit (a.k.a. machine precision)

It is the upper bound on relative error due to rounding in FP arithmetic

Base: b

precision: p (number of digits of the significand, including the leading implicit bit)

rounding unit: $e = \frac{1}{2} b^{1-p}$

Example 1:

$b = 2$, $p = 53$ (52 fraction bit, plus 1 implicit bit)

$e = \frac{1}{2} 2^{-52} = 2^{-53} \approx 1.11 \times e^{-16}$

Example 2 (consider single precision)

NOTE: some authors refer to machine precision differently (differ by order of 2)

Deriving the Rounding Unit

Hint: using two FP numbers that are close to each other, with the same exponent e, they have a spacing of $b^{e-(p-1)}$.

Example:

$b = 2$, $p = 1$, two adjacent FP numbers are 1×2^e and 0×2^e: spacing $= 2^e$

$b = 2$, $p = 2$, 1.0×2^e, 1.1×2^e: spacing $= 2^{e-1}$

General FP System

WHITEBOARD SUMMARY

General FP System IEEE Standard **(b, p, L, U)**

b: base; p: precision; L <= e <= U

Consider b = 2, common FP systems:

	name	exponent bit	fraction bit	L	U	exponent bias
binary16	half precision	5	11	-14	15	2^4-1 = 15
binary32	single	8	24	-126	127	2^7-1=127
binary64	double	11	53	-1022	1023	2^10-1=1023
binary128	quadruple	15	113	-16382	16383	2^14-1=16383

FP representation

S: sign bit

E = (exp bit - bias)

M = 1.xx...x (p bits including p-1 bits of fraction)

FP representation: $(-1)^S \times M \times 2^E$

The largest number precisely represented by a Standard Word

S = -1, E = 2013, M=1.11...1 (total 53 bits)

$(1+½+¼+....1/(2^{52})) \times 2^{1023} = (2 - 2^{-52}) \times 2^{1023} \approx 2^{1024} \approx 10^{308}$

The smallest

$(1+0) \times 2^{(-1022)} \approx 2.2 \times 10^{(-308)}$

Credits

Special thanks to all the people who made and released these awesome resources for free:

- ✘ Presentation template by SlidesCarnival
- ✘ Photographs by Unsplash