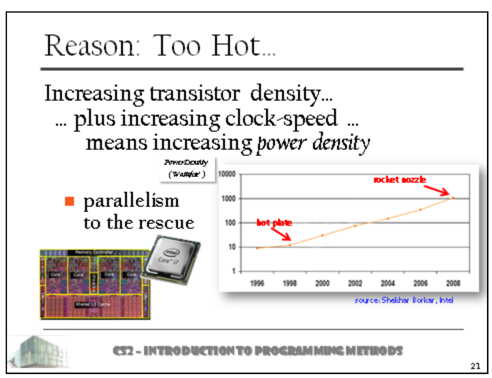
CS 2

Introduction to Programming Methods

Last Time

Concurrency and Parallelism

- synchronization of multiple threads



Today's Lecture

Numerics: when good math goes wrong

Computer programmers are fallible

- losing satellite due to public vs. private var.
  - self.setValue(.) bypassed; worked at the time…
  - but code changed, so value overwrote other data
- losing a Mars orbiter due to conversion
  - we have switched to the metric system, right?

Computers are perfect with numbers…

- right?

Integer Paradise

Dealing with bounded ints is great

- exact computations easy
  - as long as integers below a prescribed value (often, 32 bits)
    - 2 bits: 0 to 3
    - 8 bits: 0 to 255
    - n bits: 0 to 2^n -1
  - signed int
    - most commonly: **two's complement**
    - why?
      - » add/sub painless (bit overflow ignored)

$-x \equiv \bar{x}+1$

$$v = -d_{31}2^{31} + \sum_{n=0}^{30} d_n 2^n$$

Most significant bit

0	1	1	1	1	1	1	1	=	127
0	1	1	1	1	1	1	0	=	126
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	−1
1	1	1	1	1	1	1	0	=	−2
1	0	0	0	0	0	0	1	=	−127
1	0	0	0	0	0	0	0	=	−128

What About Reals?

Not so simple…

```
Python 2.5.1
Type "help", "copyright", "credits" or "license" for more information.
>>> 0.1
0.10000000000000001
>>>
```

- what's wrong?? [by the way, newer versions say: 0.1]

Tenths not very easy to represent in binary

- 0.1 = 0b0.00011001100110011001100110011001100110011001100110011010….

So, computations often not perfect

- and we can send people to the moon??

```
>>> sum = 0.0
>>> for i in range(10):
...     sum += 0.1
...
>>> sum
0.9999999999999999
```


Fixed-Point Representation

Approximate positive reals w/ powers of 2

- first, decide how many bits to use
  - 32 bits or 64 bits quite common
- then, decide where to put the "binary point"
  - for instance, point before last bit → quantum of 0.5

$$b_i \; b_{i-1} \; \cdots \; b_2 \; b_1 \; b_0 \,.\, b_{-1} \; b_{-2} \; b_{-3} \; \cdots \; b_{-j}$$

$$\sum_{k=-j}^{i} b_k \cdot 2^k$$

Signed reals?

- same two's complement idea
  - e.g., 32 bits for numbers between −1 and 1−2^-31^

$$v = -d_0 2^0 + \sum_{n=1}^{32} d_n 2^{-n}$$

Adaptive Representation?

Fixed-point is fast and simple

- but limited!
  - fixed accuracy, quite limited range
  - in fact, trade precision for range

Large range and high precision?

- would require lots of bits
- potentially wasteful

Floating-Point Representation

IEEE 754 definition

`s`	`exponent`	`mantissa`

Variants:

- single precision: 8 `exp` bits, 23 `mant.` bits
  - 32 bits total
- double precision: 11 `exp` bits, 52 `mant.` bits
  - 64 bits total
- extended precision: 15 `exp` bits, 63 `mant.` bits
  - mostly in Intel-compatible machines; stored in 80 bits
    - 1 bit wasted

Single Precision FP

For 32 bits

s	exponent	mantissa

- represents a value of $(-1)^s \, (1.m)_2 \, 2^{e-127}$
  - $1 \leq (1.m)_2 < 2 \qquad 0 < e < 255$
- example: 0xD1500000

1	1	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - value = $-1.625 \; 2^{35} = -5.5834e{+}10$

note: hexadecimal notation

x x x x : 0,1,2,…,8,9,A,B,C,D,E,F

Special Numbers

s	e	m	value
0	all zeros	all zeros	0
1	all zeros	all zeros	-0
0	all ones	all zeros	∞
1	all ones	all zeros	$-\infty$
0 or 1	all ones	non-zero	NaN
0 or 1	all zeros	non-zero	$(-1)^s \, (0.m)_2 \, 2^{-126}$

gradual underflow

Examples: $0\times\infty$, sqrt(-1), …

Check: max = 3.402823466e+38

min = 1.401298464e-45

Double-Precision FP

Same exact principles

- represents value of $\mathbf{(-1)^{s}(1.m)_2\ 2^{e-1023}}$

max = `1.7976931348623157e+308`

min = `5e-324`

Floating-Point Operations

Conceptually

- First, compute exact result
- Make it fit into desired precision
  - Possibly overflow if exponent too large
  - Possibly round to fit into `mantissa`

Example: $(-1)^{s_1}\ m_1\ 2^{e_1} \times (-1)^{s_2}\ m_2\ 2^{e_2}$

- exact result: $(-1)^{s}\ m\ 2^{e}$
  - `s = s1+s2, m = m1.m2, e = e1+e2`
  - $m \geq 2$? shift m right, increment e; e out of range? overflow
  - round m to fit `mantissa` precision

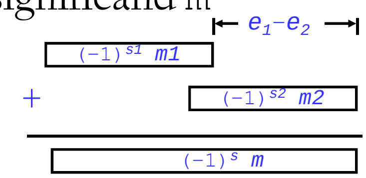
FP Addition

Operands: $(-1)^{s_1} m_1 2^{e_1}$ and $(-1)^{s_2} m_2 2^{e_2}$

- Assume $e_1 > e_2$

exact result: $(-1)^{s} m\ 2^{e}$

- exponent $e=e_1$; sign s; significand m
  - result of signed align & add



Normalization:

- If $m \geq 2$, shift m right, increment e
- if $m < 1$, shift m left k positions, decrement e by k
- overflow if e out of range, round m to fit precision

Math vs. Numerics

Properties of Addition

- commutative? YES
- associative? NO
  - overflow and rounding
- 0 is additive identity? YES
- always additive inverse ALMOST
  - except for ±infinity & NaNs
- $a \geq b \Rightarrow a+c \geq b+c$? ALMOST
  - except for ±infinity & NaNs

[Numerics in C]

C provides two levels

- `float` single precision
- `double` double precision

Casting between `int`, `float`, & `double`

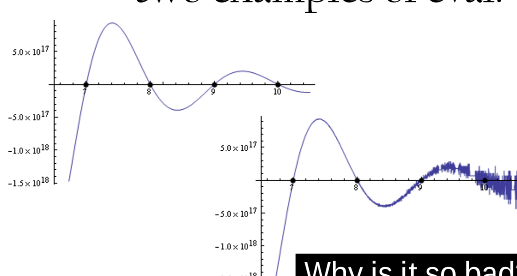
- `double` or `float` to (64 bit) `int`
  - truncates fractional part (rounding toward zero)
- `int` to `double`
  - exact conversion
- `int` to `float`
  - depends on rounding mode…

More Numerical Catastrophes…

Understanding FP helps with accuracy…

- p(.) polynomial having x=1, 2,…, 25 as roots
  - unique one of degree 25
- two examples of eval:

$$p(x) = \prod_{i=1}^{25} (x - i)$$



Why is it so bad?

$p(x) = -15511210043330985984000000 + 59190128811701203599360000\,x - 100480171548351161548800000\,x^2 + 102339530601744675672576000\,x^3 - 70874145319837672677196800\,x^4 + 35770355645907606826362624\,x^5 - 13746468217967926978680000\,x^6 + 4144457803247115877036800\,x^7 - 1001369304512841374110000\,x^8 + 196928100451110820242880\,x^9 - 31882014375298512782500\,x^{10} + 4284218746244111474800\,x^{11} - 480544558742733545125\,x1^{2} + 45145946926994481865\,x^{13} - 3557372853474553750\,x^{14} + 234961569422786050\,x^{15} - 12972753318542875\,x^{16} + 595667304367135\,x^{17} - 22563937825000\,x^{18} + 696829576300\,x^{19} - 17247104875\,x^{20} + 333685495\,x^{21} - 4858750\,x^{22} + 50050\,x^{23} - 325\,x^{24} + x^{25}$

Numerics Can Ruin a Game Too

Depth ac

How Numerics Can Kill

1996: Ariane 5 explodes

- problem quickly identified as numeric
  - but this part worked just fine on Ariane 4!
- Error was due to "*a data conversion from **64-bit floating point** to **16-bit signed integer** value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.*"
  - Ariane 5 was… faster
- $7.5B in development and launch lost