O problema da precisão numérica em computação

A representação de números fracionários por computadores digitais pode levar a problemas de precisão numérica. Sabemos que um número na base 10 (decimal) é representado como:

$23457 = 2 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 7 \times 10^0$

Analogamente, um número binário (base 2) pode ser representado como:

$110101 = 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 53$

O bit mais a direita é o menos significativo e portanto o seu valor é $1 \times 2^0 = 1$
O segundo bit a partir da direita tem valor de $0 \times 2^1 = 0$
O terceiro bit a partir da direita tem valor de $1 \times 2^2 = 4$
O quarto bit a partir da direita tem valor de $0 \times 2^3 = 0$
O quinto bit a partir da esquerda tem valor de $1 \times 2^4 = 16$
Por fim, o bit mais a esquerda tem valor de $1 \times 2^5 = 32$
Somando tudo temos: 1 + 4 + 16 + 32 = 5 + 48 = 53.

Essa é a regra para convertermos um número binário para sua notação decimal.

Veremos agora o processo inverso: como converter um número decimal para binário. O processo é simples. Começamos dividindo o número decimal por 2:

53 / 2 = 26 e sobra resto **1** → esse 1 será nosso bit mais a direita (menos significativo no binário)

Continuamos o processo até que a divisão por 2 não seja mais possível:

26 / 2 = 13 e sobra resto **0** → esse 0 será nosso segundo bit mais a direita no binário

13 / 2 = 6 e sobra resto **1** → esse 1 será nosso terceiro bit mais a direita no binário

6 / 2 = 3 e sobra resto **0** → esse 0 será nosso quarto bit mais a direita no binário

3 / 2 = 1 e sobra resto **1** → esse 1 será nosso quinto bit mais a direita no binário

1 / 2 = 0 e sobra resto **1** → esse 1 será o nosso último bi (mais a esquerda)

Note que de agora em diante não precisamos continuar com o processo pois

0 / 2 = 0 e sobra 0
0 / 2 = 0 e sobra 0

ou seja a esquerda do sexto bit teremos apenas zeros, e como no sistema decimal, zeros a esquerda não possuem valor algum. Portanto, 53 em decimal equivale a 110101 em binário.

Com números fracionários, a ideia é similar:

Na base 10:
$456{,}78 = 4 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 7 \times 10^{-1} + 8 \times 10^{-2}$

Na base 2:
$101{,}101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-2} = 4 + 1 + 0.5 + 0.125 = 5{,}625$

Na base 10, dividir por 10 significa deslocar a vírgula uma casa para a esquerda e multiplicar por 10 significa deslocar a vírgula para direita, acrescentando ou removendo zeros quando for o caso.

2317 / 10 = 231,7
231,7 / 10 = 23,17
23,17 / 10 = 2,317

Com números binários a ideia é a mesma. Mover a vírgula para a esquerda significa divisão por 2 e mover a vírgula para direita significa multiplicação por 2

28 = 11100
14 = 1110
7 = 111
3,5 = 11,1
1,75 = 1,11

Para armazenar números reais em computadores, é preciso representá-los na base 2 (binário)

No caso de números inteiros, a conversão é direta

25 = 11001 pois

25 / 2 resulta em 12 com resto 1
12 / 2 resulta em 6 com resto 0
6 / 2 resulta em 3 com resto 0
3 / 2 resulta em 1 com resto 1
1 / 2 resulta em 0 com resto 1

No caso de números reais, o processo é similar

5,625

Primeiramente, devemos dividir o número em 2 partes: parte inteira e parte fracionária
A conversão é feita independentemente para cada parte. Assim, primeiro devemos converter o número 5

5 / 2 = 2 com resto 1
2 / 2 = 1 com resto 0
1 / 2 = 1 com resto 1

Então, temos que 5 = 101

Em seguida iremos trabalhar com a parte fracionária: 0,625. Nesse caso ao invés de dividir por 2, iremos multiplicar por 2 a parte fracionária e tomar a parte inteira do resultado (a esquerda da vírgula), repetindo o processo até que não se tenha mais casas decimais depois da vírgula.

0,625 x 2 = 1,25 → 1 (primeira casa fracionária)
0,25 x 2 = 0,5 → 0 (segunda casa)

0,5 x 2 = 1,0 → 1 (terceira casa)

Assim, temos que 0,625 = 0,101 e portanto 5,625 = 101,101

Porém, em alguns casos, alguns problemas podem surgir. Por exemplo, suponha que desejamos armazenar num computador o número 0,8 na base 10. Para isso, devemos proceder da forma descrita anteriormente.

0,8 x 2 = 1,6 → 1
0,6 x 2 = 1,2 → 1
0,2 x 2 = 0,4 → 0
0,4 x 2 = 0,8 → 0
0,8 x 2 = 1,6 → 1
0,6 x 2 = 1,2 → 1
0,2 x 2 = 0,4 → 0
0,4 x 2 = 0,8 → 0
0,8 x 2 = 1,6 → 1
0,6 x 2 = 1,2 → 1
0,2 x 2 = 0,4 → 0
0,4 x 2 = 0,8 → 0
…

Portanto, 0,8 = 0,11001100110011001100….

Infinitas casas decimais. Porém, como na prática temos um número finito de bits, deve-se truncar o número para uma quantidade finita. Isso implica numa aproximação. Por exemplo, qual é o erro cometido ao se representar 0,8 como 0,11001100?

½ + ¼ + 1/32 + 1/64 = 51/64 = 0.796875, o que implica num erro de 0,003125.

O problema pode ser amplificado ao se realizar operações matemáticas com esse valor (é como se o erro fosse sendo propagado nos cálculos). Existem outros valores para que isso ocorre: 0,2, 0,4, 0,6

Ex: Forneça as representações computacionais na base 2 (binário) para os seguintes números. Quais são os erros cometidos se considerarmos apenas 8 bits para a parte fracionária?

a) 11,6
b) 27,4
c) 53,6
d) 31,2

“The real voyage of discovery consists not in seeking new landscapes, but in having new eyes.”
(Marcel Proust)