

計算機の仕組み

最低限の構成要素

CPU	命令を読み込み、それにしたがって計算して、結果をはき出す。
記憶装置	プログラムとデータを記憶する。レジスタ、メモリ、ハードディスクなど。
クロック	メトロノームみたいなもの。この振動にあわせて計算する。
入力	キーボード、マウスなど。
出力	画面など。

CPU の仕組み

プログラムカウンタ：命令の位置を記録
デコーダ：命令を解読して演算器に指示を出す
演算器 (ALU)：計算する。 $+$ $-$ $\times$ $\div$ $\cdot$ $\cdot$ $\cdot$ $\cdot$
レジスタ：小さな記憶装置

計算の流れ

1. プログラムカウンタの位置の命令を読む
2. デコーダが解析して演算器に指示を出す
3. 例えば足し算なら、
 1. 引数 1 が記録されている記憶装置上の場所から数を演算装置に読み込む
 2. 引数 2 の場所から読み込む
 3. 2 つの数を足して記憶装置上の指定した場所書きこむ
4. プログラムカウンタをひとつ進める

CPU の種類によるが、基本はこの一連の流れを 1 クロックの間に行う。

最近の CPU は数個並列してこれをやるのもっと早い。

スパコン (ベクトル計算機) や GPU は何百と同時並列で計算を行う。

クロック

水晶かなんかの振動子が入っている。

CPU はこの振動にあわせて計算するので、一秒間に何回命令を実行できるかの目安になる。

PC の CPU はだいたい 3GHz くらいで、このとき 1 クロックの間に光が進む距離は 0.1m だから配線はある程度短くないといけない。

記憶装置の種類

知ってると思うけどメモリの単位は bit 数で数える。1bit には 0, 1 のどちらかが記録できる。だから 10bit なら 1024 通りの数が記録できる。

レジスタ ~1kB (1B = 1byte = 8bit)

キャッシュ ~1MB

メモリ ~1GB

HDD, SSD ~1TB

上に行くほど CPU に近く高速

データの型

整数などの変数は、次の bit 数だけ連続したメモリを消費する。

4 (bit) char 英字、数字、記号を表すのに使う

32 int 整数を表すのに使う

32 float 小数を表すのに使う

64 double 小数を表すのに使う。float より精度が必要なときはこっち

配列の場合はさらに要素数だけ並んだメモリが消費される

計算速度

FLOPS

一秒間に何回小数点付きの 7 桁くらいの数同士の演算が出来るか。

人間	0.1FLOPS
PC の CPU	50GFLOPS
GPU	1TFLOPS
東大のスパコン	100TFLOPS
京	10PFLOPS

アルゴリズムと計算量

アルゴリズムとは計算手順のこと。いくら早いコンピュータでも計算手順が悪いと計算に時間がかかる。アルゴリズムに対する計算速度を評価する指標が計算量である。

ランダウの記法 $O(n)$ をつかって表記して、入力に対してどれだけの計算が必要かを表す。

ふつう定数とか主でない項は省く ($3x^2+2x+1 \rightarrow O(x^2)$)

例

- ・ n 個の数値から最大の数を求める
入力のサイズ: n に比例
アルゴリズム: 最初から一個ずつ見て行って、今までの最大値よりも大きい数を見つけたら、最大値をその数に更新する
計算量: $O(n)$
- ・ $n \times n$ の二次元行列 A, B の足し算
入力のサイズ: n^2 に比例
アルゴリズム: 全ての要素を足すだけ
計算量: $O(n^2)$
- ・ $n \times n$ の二次元行列 A, B の掛け算
入力のサイズ: n^2 に比例
アルゴリズム: $(AB)_{ij} = \sum a_{ik} b_{kj}$
計算量: $O(n^3)$
- ・ 入力された n 個の文字を並び替えてできる全ての単語を調べる
入力のサイズ: n に比例
アルゴリズム: ひたすら並び替える
計算量: $O(n!)$
- ・ x^k を計算
入力のサイズ: 2 つの数
アルゴリズム 1: k 回掛け算
計算量: $O(k)$
アルゴリズム 2: x^2 を計算 $\rightarrow x^4$ を計算 $\rightarrow x^8$ を計算 $\dots$
計算量: $O(\log(k))$

浮動小数点数と計算誤差

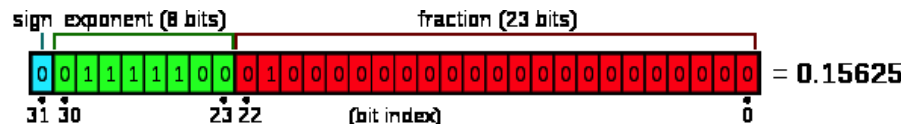
浮動小数点数の表現

指数表記で記録されている。つまり、

$$\pm B \times 2^{(A-E)}$$

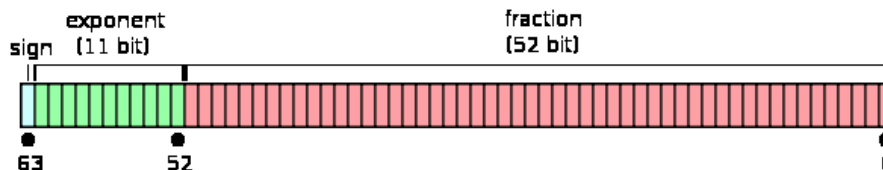
最初の 1bit は符号を、次の数 bit がべき乗数 A を、残りの bit が基本となる数 B を表す。

float:



・・・ B の有効数字は 10 進数で 7 桁くらい、A の範囲は -127~+127

double:



・・・ B の有効数字は 10 進数で 15 桁くらい、A の範囲は -1023~+1023

丸め誤差

$$1.000001 \times 10^0 - 1.000000 \times 10^0 = 1.000000 \times 10^{-6} \leftarrow \text{有効数字が 7 桁から 1 桁に！}$$

$$1.000000 \times 10^0 + 1.234567 \times 10^{-4} = 1.000123 \times 10^0 \leftarrow 4567 \text{ は消滅}$$

打ち切り誤差

数値計算は近似計算なので、どれだけ細かく計算するか、で精度が決まる。

たとえば展開級数の何次の項まで見るか、など。

ある程度以上細かくすると丸め誤差が大きくなるのでトレードオフになる。

差分法の基礎

常微分方程式の差分解法(1 次)

計算したい量 : $x(t)$

方程式 : $dx/dt = ax + b$

↓微分の定義

$$\lim \{(x(t+\Delta t) - x(t))/\Delta t\} = ax(t) + b$$

↓差分化

$$x(t+\Delta t) \sim x(t) + (ax(t) + b)\Delta t$$

時刻 t での x の値がわかっていれば $t+\Delta t$ での x が近似計算できる。 Δt を小さくするほど精度は上がる。ところで $x(t+\Delta t)$ を $x(t)$ まわりでテイラー展開すると、 $O((\Delta t)^2)$ の誤差があることがわかる。たとえば一秒後の x を求めたいとしたら、 $1/\Delta t$ 回この処理を繰り返すので、誤差は拡大して $O(\Delta t)$ 程度になる。このとき、「この方法の精度は 1 次」という。

偏微分方程式の差分解法(1 次)

計算したい量 : $f(x, t)$

方程式 : $\partial f/\partial t = a (\partial f/\partial x) + bf + c$

↓偏微分の定義

$$\lim \{(f(x, t+\Delta t) - f(x, t))/\Delta t\} = a \lim \{(f(x+\Delta x, t) - f(x, t))/\Delta x\} + b f(x, t) + c$$

↓差分化

$$f(x, t+\Delta t) = f(x, t) + [a (f(x+\Delta x, t) - f(x, t)) / \Delta x + b f(x, t) + c]\Delta x$$

つまり、時刻 t での $\forall x$ $f(x, t)$ がわかっていれば、 $t+\Delta t$ での f が近似計算できる。これも 1 次の精度である。

二次精度の差分

精度を上げることを考えてみよう。

上の $\partial f/\partial x$ について、テイラー展開により、

$$f(x+\Delta x, t) = f(x, t) + (\partial f/\partial x)(x, t) \Delta x + (1/2) (\partial^2 f/\partial x^2)(x, t) (\Delta x)^2 + O((\Delta x)^3)$$

$$f(x-\Delta x, t) = f(x, t) - (\partial f/\partial x)(x, t) \Delta x + (1/2) (\partial^2 f/\partial x^2)(x, t) (\Delta x)^2 + O((\Delta x)^3)$$

↓

$$f(x+\Delta x, t) - f(x-\Delta x, t) = 2 (\partial f/\partial x)(x, t) \Delta x + O((\Delta x)^3)$$

↓

$$(\partial f/\partial x)(x, t) = (f(x+\Delta x, t) - f(x-\Delta x, t)) / (2\Delta x) + O((\Delta x)^2)$$

これを先ほどの方程式の右辺で使ってあげると、精度は二次になる。この方法を「中心差分」という。

初期値問題... $f(0)$ が与えられた時 $f(T)$ を求めるような問題

境界値問題...境界条件が与えられた時、それを満たす f を求めるような問題