第2章　IEEE754 standardと多倍長浮動小数点数

2.1　PCの構成

現在標準的に使用されているPCは，以下の主要デバイスによって構成されている。

CPU (Central Processing Unit) ••• コンピュータにおける命令，演算の全てを実行する半導体チップ

RAM (Ramdom Access Memory) ••• メインメモリ (main memory) とも呼ばれる，実行されるプログラムやそこで使用されるデータを置いておくためのメモリ

I/O (Input/Output device) ••• コンピュータ外部からの入力 (input) や外部への出力 (output) を行うためのデバイス。キーボード，マウスは input device, ハードディスク (外部記憶)，ディスプレイ，プリンタは output device である

NIC (Network Interface Card) ••• ネットワークに接続するためのデバイス。Ethernetが一般的であるが，さらに高速なMilinet(?) のようなネットワークも高性能なPC cluster専用機には使用されることがある

これらのデバイスは図2.1のように接続され，PCを構成している。

本書では以下の実機を使用してプログラムの実行を行っている。先頭の太字部分は本書全体で使用する略称である。

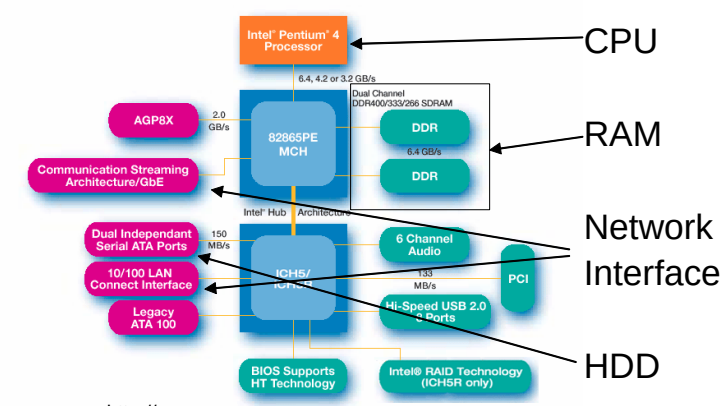
x86_32 アーキテクチャ

Pentium4 ••• Intel Pentium IV 2.8cGHz (11 nodes, 11PEs, Vine Linux 3.2)

Xeon ••• Intel Xeon 3.0GHz[1](8 nodes, 16PEs, Redhat 8)

x86_64 アーキテクチャ

[1]名古屋大学三井研所有



http://support.intel.com/products/i/chipsets/865pe/865pe_schematic.gif

図 2.1: PC の構成 (Pentium 4+65 chipset)

PentiumD ・・・ Intel Pentium D 820 (2.8GHz, 4 nodes, 8PEs, Fedora Core 5 x86_64)

Athlon64X2 ・・・ AMD Athlon64 X2 3800+ 2.0GHz (Fedora Core 4 x86_64)

2.2 IEEE754 倍精度

本書で扱う数値計算と呼ばれる演算処理主体のプログラムは，これらのデバイスを図 2.2 のように使用しながら実行される。

入力 入力データはキーボード，あるいは他のデバイスから入力されることが多い。人間が認識しやすい 10 進表現で入力され，コンピュータ内部に取り込まれる際に 2 進表現に変換される。浮動小数点数として表現される場合は，後述する IEEE754 単精度，倍精度，拡張倍精度の形式に変換される。

演算 変換された 2 進表現データはメインメモリ (RAM) に格納され，必要に応じて CPU から呼び出されて，CPU 内部のレジスタ (register) を介して処理 (演算) される。近年の CPU には RAM とレジスタの間には RAM より高速に読み出せるキャッシュメモリ (Cache memory) が搭載されているのが普通である。同じ RAM のデータを繰り返し呼び出す際には，実際にはこの Cache からデータを呼び出していることも多い。

出力 処理されたデータは外部記憶 (HDD, 光ドライブ等) にファイルの形で書き出されて保存され，必要に応じてディスプレイに表示される。人間が直接文字

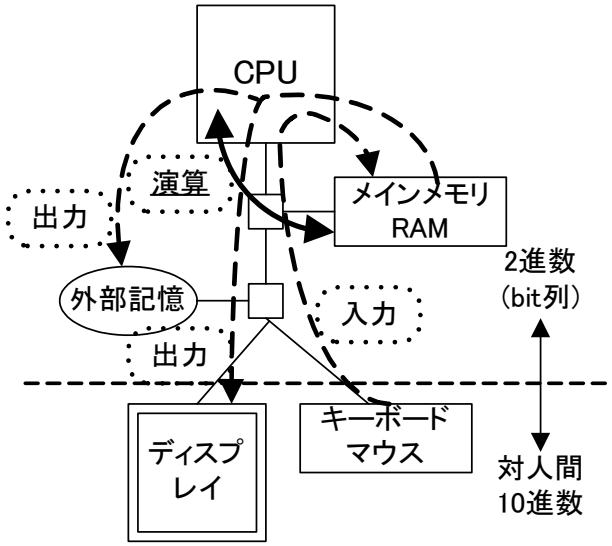


図 2.2: コンピュータ内部の演算処理の流れ

列として読み取る目的のために，この時点で 10 進表現に変換されることも多い。

実数データは全て，有限桁の浮動小数点数として表現しなければならないため，1/3 のような循環小数となる有理数や，$\sqrt{2}, \pi$ のような無理数は性格に表現することは出来ず，必ず丸め誤差 (rounding error) が混入する。

現在標準的に使用されている 2 進浮動小数点形式は，IEEE754-1985 規格で定められているものである。これを図 2.3 に示す。

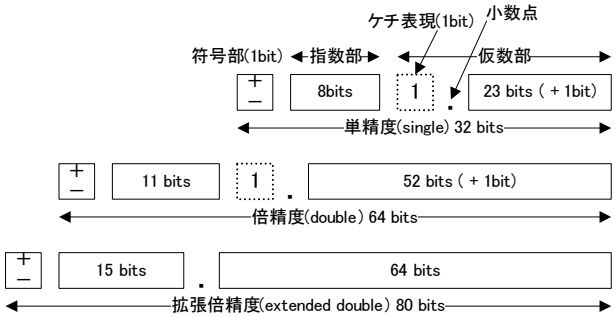


図 2.3: IEEE754 standard floangint-point number

FORTRAN77 や C/C++のようなプログラミング言語では，普通，次の変数型がこの IEEE754 単精度 (single precision), 倍精度 (double precision) に相当する。

言語名	単精度	倍精度
FORTRAN77	`REAL, REAL*4`	`REAL*8, DOUBLE PRECISION`
C/C++	`float`	`double`

単精度，倍精度，共に 2 進数なので 10 進表現に変換した際の精度桁数が分かりづらいので，ここできちんと計算しておこう。

単精度は仮数部 (mantissa) がケチ表現部を含めて 24bit あり，末尾桁は 2^{-23} である，同様に，倍精度は 53bit あり，末尾桁は 2^{-52} である。従って，この仮数部の bit 数以上の桁数が必要になる実数はそれぞれ必ずこの bit 数に丸められるので，単純に 0 捨 1 入したとすれば[2]，単精度の場合は最大で $1/2 \times 2^{-23} = 2^{-24} \approx 6.0 \times 10^{-8}$, 倍精度の場合は $2^{-53} \approx 1.1 \times 10^{-16}$ となる。よって 10 進数に換算すると，それぞれ約 7 桁，約 16 桁の精度を持つ，ということになる。

RAM 容量が GB(Giga Byte) 単位になり，IEEE754 浮動小数点数が CPU 内部のハードウェアユニットで直接処理されるようになった現在ではもっぱら倍精度が使用されるので，本書で使用する C/C++言語の場合，`double` 型は 10 進約 16 桁の精度を持つと考えてよい。

2.3　多倍長浮動小数点数: GMP と MPFR

「多倍長計算」あるいは「multiple precision」で Web を検索してみると，さまざまなサイトでさまざまなソフトウェアが存在していることがわかる。但し，そのうちの多くは円周率 π の計算を扱ったものである。後述するが，このような計算は興味深いものであるが，我々が目指す「実用的な精度の」多倍長計算とは縁遠いものである。

IEEE754 倍精度では十分な精度を得られない悪条件 (ill-conditioned) 問題や，丸め誤差に敏感なアルゴリズムの研究を行うには，より精度の高い多倍長浮動小数点数 (多倍長 FP 数) が必要になる。我々はこのような目的にのみ，多倍長 FP 演算を適用することを考えていく。

CPU 内部のハードウェア演算ユニットで全ての演算をこなしてくれる IEEE754 浮動小数点数 IEEE754 FP 数) とは異なり，仮数部が可変になる多倍長の浮動小数点演算は全てソフトウェアで構築する必要がある。しかし作るのはそれ程難しくはないものの,「実用的な」速度にチューンすることが難しいことが知られている。

多倍長 FP 数・演算の実現方法は大きく分けて二通りある。

一つは，現状の IEEE754 FP 数を繋げて桁を長くする方法である。この代表として ARPREC[8](C++/Fortran90 で使用できるライブラリ) がある。

[2]実際の丸め方は少し異なる。

もう一つは多倍長「自然数」演算を積み上げて多倍長 FP 演算を実装する方法である。この代表として，GNU MP(GMP)[7] とそのファミリーを形成するライブラリ群がある。

我々が主として使用するのはこの GMP 及びそのファミリーである MPFR である。これは GMP の自然数 (mpn_*) ライブラリを土台にした多倍長 FP 数ライブラリで，初等関数や特殊関数など数値計算に必要な機能が豊富な，IEEE754 互換の丸めモードも備えた多倍長 FP ライブラリである。そのソフトウェア構造を図 2.4 に示す。

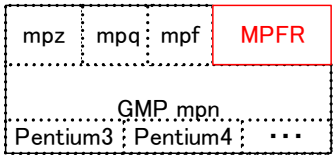


図 2.4: GMP と MPFR の階層構造

MPFR が提供する多倍長 FP 数は構造体になっている。その構成を図 2.5 に示す。

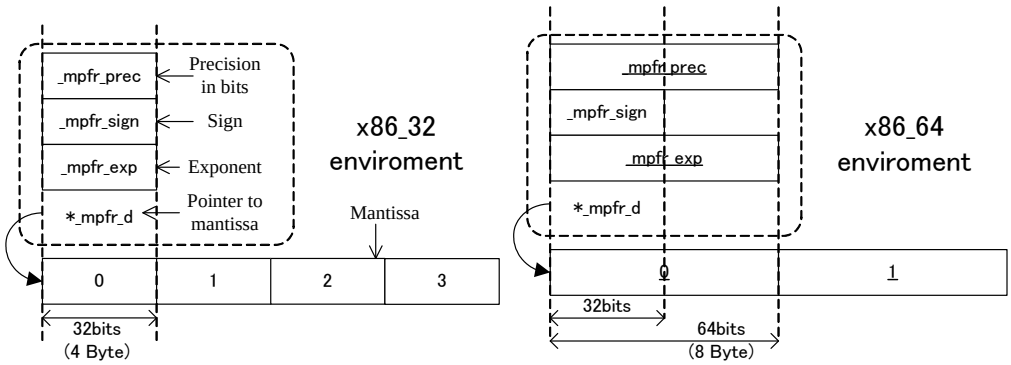


図 2.5: MPFR の構造体 (左:32bit 環境, 右: 64bit 環境)

構造体を構成する各変数の機能は次のようになる。

_mpfr_prec ・・・ 仮数部の bit 数

_mpfr_sign ・・・ 符号 ± と仮数部における非ゼロ要素数

_mpfr_exp ・・・ 指数部

_mpfr_d ・・・ 仮数部へのポインタ

一見して分かるのは，仮数部が構造体とは分離して存在していることである。これは仮数部の長さをプログラム実行中にも変更可能なようにとの配慮から来ているが，並列計算の際にはデータ転送時において若干面倒なことになってしまう。

なお，Pentium D や Athelon64 以降の x86_64 CPU が提供するのような 64bit 環境においては図 2.5 の左に示すように，一部の変数が 64bit サイズになる。これによってメモリアクセス回数を減らす効果が期待できる。

2.4　MPFR の性能と現実的な多倍長精度の範囲

MPFR はベースとなる自然数演算を GMP に頼っているため，その演算性能は完全に GMP に依存する。この部分は CPU アーキテクチャごとに最適化されており，Intel CPU でも徐々に追加されてきた SIMD 命令を使用してその CPU ごとに最も高速な演算・データ転送が可能なようにチューンされている。それでも，乗算や除算といった重い演算は，仮数部の長さ N に応じた 4 つのアルゴリズムを用いて実装されており，速度の大半はこれによって決まる。具体的に言うと，乗算の場合は 4 つのアルゴリズムを実装している。

Basecase 乗算 (筆算と同じ) ･･･ $O(N^2)$

Karatsuba 乗算 ･･･ $O(N^{1.585})$

Toom-Cook 3way 乗算 ･･･ $O(N^{1.465})$

FFT 乗算 (Option) ･･･ $O(N^{1.333\sim1.4})$

π を何万桁も計算する時には必ず FFT(Fast Fourier Transform) による乗算アルゴリズムが取り上げられるが，GMP の開発者は，FFP が利いてくるのは少なくても約 10000bits 以上 (by GMP マニュアルより) であると判断しているようで，コンパイル時に明示的に指定しない限り，これは使用されず，残り 3 つのアルゴリズムだけで計算が行われる。除算のアルゴリズムも乗算に準して決定されるため，速度の Order は同程度になる。

従って，MPFR の乗算･除算は仮数部の長さの 1.4〜1.6 乗に比例してそのパフォーマンスが決定されることになる。これを実際に計測した結果を図 2.6 に示す。

このように，MPFR の演算は，GMP のチューンによってかなり高速ではあるが，実用を考えると，特に大規模な線型計算が伴うような数値計算では，1 演算が 1 ミリ秒 (milli-sec) を越えるようでは不都合が生じる。この場合，行列積や連立一次方程式の直接法のように,，次元数の 3 乗の演算量が必要になるから，これが 1 秒程

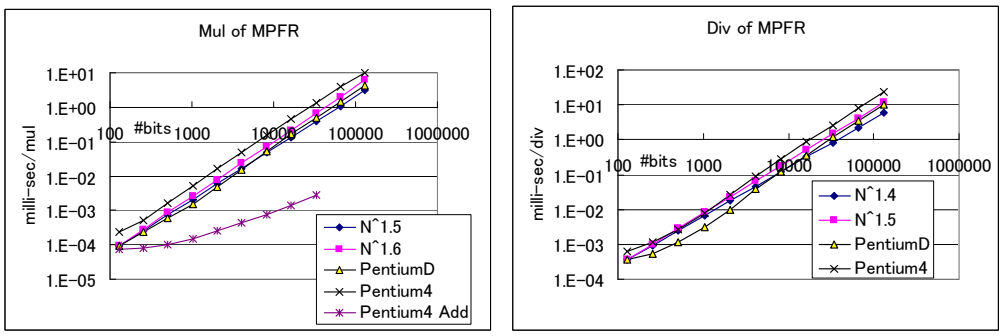


図 2.6: MPFR の性能

度で終了しないことには，これを繰り返し実行する非線型問題や常・偏微分方程式を実用的な速度で解く事は難しいことになるからである。

現状では，下記に示すとおり，Pentium D で 32768bits の仮数部を持つ多倍長 FP 演算が全て 1 ミリ秒程度以下で実行できる。

	Pentium D 820		Pentium 4 2.8GHz	
# of bits	MUL	DIV	MUL	DIV
32768	0.491	1.16	1.33	2.65
65536	1.48	3.5	3.84	7.97

従って,「現実的な多倍長精度」は 30000bits 以下と考えるのが妥当であろう (図 2.7)。

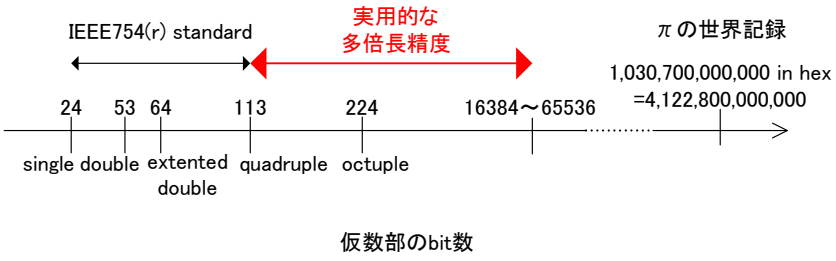


図 2.7: 現実的な多倍長精度の範囲

では最後に，多倍長 FP 演算は，IEEE754 倍精度演算に比べてどのぐらい「遅い」のかを計測してみよう。

今回は行列積 (第 9 章参照) を計算し，FLOPS(乗算回数/sec) を計測してその差を比較していることにする。但し，MPFR/GMP が CPU ごとのチューンを行った

ライブラリであることを勘案すれば，IEEE754 倍精度演算も同様のチューンが行われていないと不公平である。そこで今回は ATLAS[1] の BLAS3(付録参照) 関数 (DGEMM) を使用して行列積の計算を行うことにした。MPFR の仮数部は 128bit(10 進約 38 桁) である。その結果を図 2.8 に示す。

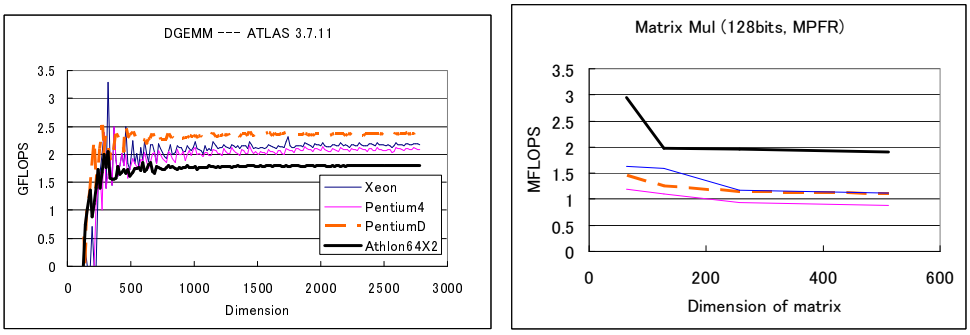


図 2.8: IEEE754 倍精度と MPFR の性能差

128bit 計算でも IEEE754 double の 1/1000 以下であることが分かる。最も，IEEE754 倍精度といえども，Cache メモリのヒット率を全く勘案せずにプログラムを組むと，この差はもっと縮まってしまうのが普通である。