数値計算と速度の話

～ スーパーコンピュータはどれくらい速いか ～

渡部 善隆 †

計算機センターで仕事をしていて何よりも幸運なのは、最新の計算機がまわりにたくさんあり、(利用者の邪魔にならない範囲ですが) 比較的自由にいじれることです。研究や仕事の都合で複数の計算機やプログラミング言語を取り扱っていると、面白そうですがちゃんと実験したことがない疑問がいくつか浮上しました。

フランス革命のスローガンは「自由」「平等」「博愛」でしたが、数値計算の世界では「速い」「でかい」「正しい」が正義の旗印です。本稿では以下にあげる 5 つの実験を通し、数値計算における「速度」を主に考えてみたいと思います。

- 単精度演算と倍精度演算はどれくらい差があるか。
- ベクトル計算機は他の計算機に比べて桁違いに速いか。
- 自作のサブルーチンはプロにかなうか。
- 数学関数は信用できるか。
- C と Fortran はどっちが速いか。

なお、特定の分野にしか通用しない用語には極力 (よけいな) 解説を加えましたが、一応、自分で Fortran プログラムを一度くらい組んだことがある人を読者の対象にさせていただきました。

32bit 対 64bit

実数型宣言に必要な記憶領域

計算機では、情報はすべて 2 進法に基づいて処理されます。つまり 0 と 1 の世界です。2 進数字 1 個を 1**bit** と呼びます。数字や文字はこの bit を使って表現されます。

0 1 ← bit

bit が集まって構成される情報量の単位を **byte** と呼びます。通常は 8bit を単位として 1byte といいます。

10010011 01001011 ← byte

さて、Fortran プログラムで倍精度実数 A を一つ宣言したとします。この時、A には 8byte の記憶領域が必要になります。よって、double presicion (または REAL*8) で宣言した A のため、計算機はプログラム実行時に 64bit の記憶領域を確保します。

対して単精度実数は、倍精度の半分の 4byte の記憶領域が必要です。従って、REAL で宣言した実数は 32bit の領域を確保します。

数値計算で通常いちばん記憶領域を食う配列についても同じです。

```
REAL A(1001,1001)             ! 4byte の配列宣言
DOUBLE PRECISION B(1001,1001) ! 8byte の配列宣言
```

上のプログラムで、単精度実数行列 A が確保する領域は、宣言した配列の要素数分です。よって、$1001 \times 1001 \times 4$byte で、約 3.82MB[1] の記憶領域が必要です。

一方、同じサイズの宣言でも、倍精度実数行列 B はちょうど A の倍の約 7.65MB の記憶領域が必要になります。

バンクコンフリクトの恐怖

先ほどの配列宣言は REAL A(1001,1001) となっています。なぜきりのよい REAL A(1000,1000) とか、REAL A(1024,1024) とか書かないのだろうと疑問に思う方もいらっしゃると思います。

† 九州大学大型計算機センター・研究開発部

[1] MB は MegaByte の略。1MB は 1024^2byte です。

その訳は、現在のスーパーコンピュータの主力機である**ベクトル計算機**の性質からきています。

実は、ベクトル計算機で配列宣言が 128 の整数倍や、一般に $2n$ とか $4n$ (n は自然数) などで定義された場合、主記憶装置内で配列への参照要求がかち合って、実行性能が著しく低下することがあります。この現象を**バンクコンフリクト** (bank conflict) といいます。なお、バンクコンフリクトについての詳しい解説は [1] を御覧下さい。バンクコンフリクトは、ベクトル計算機特有の現象で、それ以外の計算機では発生しません。

そのような訳で、ベクトル計算機で配列を宣言する場合は、多少の記憶領域の無駄は覚悟の上、配列次元を $2n+1$ や $4n+1$ などと宣言した方が、実行スピードの改善になります[2]。

浮動小数点の形式

単精度型は、記憶領域が倍精度型の半分ですむ反面、精度が倍精度型の半分しかありません。

宣言された実数は、計算機内で浮動小数点 (floating point number) として扱われます。浮動小数点表現において、精度に関係する部分は「仮数部」と呼ばれるところです。例えば、汎用機 M-1800/20 の Fortran システムでは、単精度型で 24bit、倍精度型で 56bit が仮数部にあてられます[3]。 M-1800/20 の浮動小数点表現は IBM 形式[4] といわれるもので、4 つの bit を単位とした 16 進数で表現されています。従って、仮数部は単精度で 16 進 6 桁、倍精度で 16 進 14 桁の精度を持っています[5]。

これを 10 進数表現に直してみましょう。単精度の場合 $16^6 = 10^x$、倍精度の場合 $16^{14} = 10^x$ をみたす x をそれぞれ求めれば、大体の精度がわかります。

また、ワークステーション標準の浮動小数点表現に IEEE 形式[6] というものがあります。この形式では仮数部は単精度で 2 進 23 桁、倍精度で 2 進 52 桁の精度を持っています。

[2] 実際は第一添字だけ修正すれば大丈夫と思います。また、バンクコンフリクトを起こす条件は、各ベクトル計算機のハード構成により微妙に異なります。

[3] 残りの 8bit は符号と指数部の表現に使われます。

[4] IBM は International Business Machines の略。ご存知世界屈指のコンピュータ・メーカー。

[5] もちろん、演算を繰り返す度に誤差がたまっていきます

[6] IEEE は Institute of Electrical and Electronics Engineers の略。アメリカ電気電子技術者協会。「アイトリプルイー」と読むらしいのですが、聞く方は「アイトルプリー」と聞こえます。浮動小数点の形式には、他にも CRAY 形式などがあります。

ごちゃごちゃしてきたので、表 1 にまとめます。

表 1: 浮動小数点の精度

	IBM 形式	IEEE 形式
単精度型	16 進 6 桁 (7.22 桁)	2 進 23 桁 (6.92 桁)
倍精度型	16 進 14 桁 (16.86 桁)	2 進 52 桁 (15.65 桁)

( ) 内は 10 進に換算した近似値

IEEE 形式の場合、仮数部分の精度が IBM 形式より少し劣る代わりに、指数部分に力を注いでいます。そのため、IBM 形式よりも絶対値の意味で「でかい」数が処理できます。

浮動小数点形式についての解説は、数値計算の本 (例えば [2] や [3]) を読むと、たいてい最初の章に書いてあります。また [4] では、IEEE 形式と IBM 形式の変換方法についての有意義な解説がされていますので、興味のある方はお読み下さい。

あなたならどうする

さて、表 1 でわかるように、単精度型が倍精度型の “半分” の精度ということは、演算の過程で致命的な誤差が計算結果に混入する危険が倍精度型の “倍” あるということにほかなりません。

大規模な数値計算が研究の成果を大きく左右するような場合、「記憶領域をとるか、精度をとるか」で悩むこともしばしばです。

ただし、倍精度やさらに倍の精度を持つ 4 倍精度で計算したからといって、計算結果を全面的に信用していいかといえば、必ずしもそうはいきません。浮動小数点演算があくまでも無限の近似にすぎないという事実から、情報の損失は必ず起きます。「計算の品質」については [5], [6] を御覧下さい。

数値計算を単精度でやるべきか倍精度でやるべきかの選択は、前書きで述べた数値計算の正義のスローガン「速い」「でかい」「正しい」全てに関わる重要な問題です。

そこで、次の実験をします。

実験 I

倍精度 (64bit) データは単精度 (32bit) データの倍の記憶領域が必要。では、倍精度での演算時間は単精度での演算時間の倍かかるのか？

4 倍精度も気になるところですが、4 倍精度型は Fortran の正式な規格ではありません。また、テス

トに用いたサブルーチンが 4 倍精度型に対応していなかったため、今回は行いませんでした[7]。

サブルーチンの性能比較

実験は、九州大学大型計算機センターにある汎用機 FUJITSU M-1800/20 と、ベクトル計算機 FUJITSU VP2600/10 を用いて行いました。まず、多くの数値計算で多用される行列積と連立 1 次方程式の解を求めるサブルーチンにターゲットを絞り、単精度と倍精度で処理速度を調べることにします。

通常、プログラム全体を通じて、これらの行列演算は大量のコストがかかるところです。環境は表 2 にまとめました。

VMGGM, DVMGGM は行列の積を求めるサブルーチン、 VLAX, DVLAX は密行列の連立 1 次方程式を解くサブルーチンです。先頭に “D” がついた方が倍精度用です。

これらのサブルーチンは、富士通株式会社提供の科学計算用サブルーチンパッケージ SSL II, SSL II/VP の拡張機能の一部で、計算機の能力を最大限に引き出すように「ギンギン」にチューニングされています[8]。引数や解法の解説は [8] を御覧下さい。

配列はバンクコンフリクトを避けるため、1025×1025 で宣言します。行列の大きさを n として $n = 16, 32, 64, 128, 256, 512, 1024$ に対し、行列とベクトルを

$$\begin{cases} A = B = (b_{ij}) \\ b_{ij} = \sqrt{\dfrac{2}{n+1}} \sin \dfrac{\pi ij}{n+1} \\ f = (f_i) \\ f_i = \displaystyle\sum_{j=1}^{n} b_{ij} \end{cases} \tag{1}$$

で定義した後、単精度、倍精度の各サブールーチンを用いて行列積

$$C = A \times B$$

に要する時間、および方程式

$$Ax = f$$

の解 $x = (x_i)$ を得るまでの時間を計測します。計測は各 Fortran システムが装備している CLOCK サブルーチン ([9]) を用い[9]、次のようにして実行時間を調べます。

```
CALL CLOCK(T1,2,2)
CALL DVMGGM(A,MAXN,B,MAXN,C,MAXN,N,N,N,ICON)
CALL CLOCK(T2,2,2)
CPU=(T1-T2)*1.0D-3
```

サブルーチンを呼び出すのに要する時間は無視します。また、 CLOCK サブルーチンの計測結果は信用することにします。

実験結果

それでは結果を見ましょう。測定は 1995 年 6 月 29 日に行いました。

まずは汎用機での実行時間です。表 3 を御覧下さい。

表 3: M-1800/20 での測定結果

	行列の積		連立 1 次方程式の解法	
次数	単精度	倍精度	単精度	倍精度
16	0.293	0.303	0.244	0.244
32	1.786	1.732	1.109	1.022
64	13.199	12.742	6.314	5.879
128	114.875	118.569	41.415	41.126
256	892.322	936.736	304.125	314.801
512	6214.375	6800.281	2350.711	2409.872
1024	49992.702	54451.673	18847.299	19295.309

(単位: ミリ秒)

だいたい同じですが、次数を大きくすると単精度演算の方が若干速くなっています。 $n = 1024$ での行列積では、単精度は倍精度の約 1.1 倍速いという結果を得ました。

続いては、全く同じプログラムをベクトル計算機で測定した結果です。

なんと、ベクトル計算機では倍精度演算の方が速いという結果が得られました。もちろん、 CLOCK 関数の誤差も考える必要がありますが[10] $n = 1024$ のとき、倍精度は単精度に対して連立 1 次方程式の

[7] 新しい JIS 規格である Fortran 90 では、必要な精度と範囲を自由に宣言することが可能になりました ([7])。ただしその精度に対応した関数やサブルーチンは自分で作ることになります。現在多くの Fortran システムに実現されている 4 倍精度型が、サブルーチンライブラリや組み込み関数も含めて今後もサポートされるかどうかは、よくわかりません。

[8] 一部またはほとんどがアセンブラで書かれているのではと思いますが、ソースコードは企業秘密で非公開です。

[9] 実行時間計測関数は、計算機やコンパイラによってまちまちです。出来れば統一して欲しいものです。

[10] 一回の実行が極めて短いジョブを調べるには、例えば同じジョブを 10000 回繰り返した結果を 10000 で割って平均値をとるという方法がありますが、 FORTRAN77 EX の CLOCK 関数は精度がよさそうなので、信用します。

表 2: 実験 I の環境

計算機	M-1800/20	VP2600/10
OS コンパイラ ライブラリ	OS IV/MSP V10L10 FORTRAN77 EX V12L10 SSL II V12L10	OS IV/MSP V10L10 FORTRAN77 EX/VP V12L10 SSL II/VP V12L10
サブルーチン	VMGGM, DVMGGM (行列積) VLAX, DVLAX (連立 1 次方程式)	

表 4: VP2600/10 での測定結果

	行列の積		連立 1 次方程式の解法	
次数	単精度	倍精度	単精度	倍精度
16	0.104	0.102	0.181	0.202
32	0.280	0.230	0.400	0.418
64	0.952	0.672	0.968	0.972
128	3.810	2.354	2.828	2.671
256	17.815	10.005	9.681	8.602
512	111.110	60.410	39.927	34.228
1024	874.635	455.867	215.531	187.933

(単位: ミリ秒)

解法で約 1.2 倍、行列積で約 1.9 倍の性能を出しています。

全体の実行時間を比較すると？

表 3, 4 は SSL II のサブルーチンライブラリの実行時間に限った実験でした。もしかしたらライブラリに細工がしてあり、倍精度のときはうんと頑張るのかも知れません。そこで今度は、 Fortran プログラムを自分で組み。翻訳、結合・編集、実行全体に要する時間で比較することにします。

プログラムは以下の要領で作成しました。

- 配列は 4001×4000 で宣言。
- 次元数 n を 4000 として行列 A とベクトル f を (1) で定義。このように作ると、 $Ax = f$ の解は正確に $x = (1, 1, \cdots, 1)$ になることが知られている。
- 連立 1 次方程式 $Ax = b$ を Gauss の消去法で解くサブルーチン MYLU を作成。なお、 MYLU は特別にベクトル計算機を意識していない。
- MYLU を用いて解 $x = (x_i)$ を求める。また誤差として、各要素の絶対値での最大誤差を

$$\max_{1 \le i \le n} |x_i - 1|$$

で特定する。

- 以上のプログラムを単精度用と倍精度用で作成し、実行時間を比較した。なお、実行時間は計算機システムが記録するログによった。

MYLU の中身は後ほど紹介します。単精度用と倍精度用のプログラムの違いは、実数宣言と組み込み関数の名前だけで、アルゴリズムは全く同じです。計算機の環境は表 2 と同じです。また、翻訳オプションとして、`OPT(E)` を指定しました。

実験は 1995 年 7 月 7 日に行いました。表 5 が実行結果です。

表 5: Fortran プログラムの計算時間比較

	M-1800/20		VP2600/10	
種別	単精度	倍精度	単精度	倍精度
翻訳	0.45	0.45	0.62	0.62
結合・編集	0.08	0.08	0.36	0.35
実行	14669	15362	188	84
合計	14670	15363	189	85
最大誤差	1.02E+2	2.10E-8	7.46E+1	1.27E-8

(単位: 秒)

汎用機では、やはり単精度計算の方がわずかに速いという結果がでました。ただし比率は 1.05 倍程度なので、互角の戦いといえます。

片やベクトル計算機では、倍精度の 85 秒に対して、単精度はその 2.2 倍の 189 秒も処理に時間がかかってしまいました。

それにもまして注目していただきたいのは、「最大誤差」の欄です。 M-1800/20 の単精度の誤差が “1.02E+2” という意味は、真の解は全て 1 であるべきなのに、数値結果のどれかの絶対値が 100 を越える値になってしまったということです。 VP2600/10 でも、誤差は 70 以上にふくれあがっています。これでは、ほとんど使いものにならないと言っていいでしょう。対して倍精度演算では、誤差でかなり

汚されてはいますが、7 桁くらいは一致しているので、状況によっては使える余地がありそうです。

つまり、VP2600/10 で 4000 元ほどの大規模計算を行うこのテストプログラムでは、単精度演算は倍精度と比べて実行時間が 2 倍以上かかる上に、数値結果が使いものにならないという恐ろしい結果が出ました。こうなると、領域の節約という利点は何の役にも立たなくなります。

ベクトル計算機に代表されるスーパーコンピュータ[11]を用いての数値計算は、大規模な配列を DO ループで “ぶんまわす” のがほとんどです。計算規模が大きいということは、演算量も多いということであり、演算量が多いということは、それにともなう誤差の増大が避けられないということです。誤差が増大するということは、計算結果に全く意味がなくなる危険性が大きくなるということです。

従って、この実験での結論は次のようになります。

結論 I

大規模な数値計算をベクトル計算機で実行する場合、計算結果の品質を保つため、出来るだけ倍精度演算でやりましょう。

FACOM じゃないの？

汎用機 M-1800/20 もベクトル計算機 VP2600/10 も、商品名は “FUJITSU” です。両方とも、伝統ある「M シリーズ」「VP シリーズ」の一商品ですが、この代になって商品名が “FUJITSU” に代わりました。前の機種までの商品名は “FACOM” です。

FACOM は Factory Automation COMputer の略で、長年富士通コンピュータのブランドとして使われてきたのですが、何故か変わってしまいました。理由は調べてないのでよくわかりませんが、一つの説として [10] の指摘を紹介しておきます。

> 富士通のコンピュータは世界中に広く輸出されているが、その商品名はファコム FACOM という。富士通のコンピュータといった意味だ。しかし、これも英語圏の人は顔をしかめる。字を見た時には気づかないが、耳で聞くと「ファック・オン Fuck on」に近い感じがするからである。意味は「犯してやれ」とでもなるだろうか。ひどく下品に響く。人前では口にしにくい名前だ。
>
> 呉 智英『言葉につける薬』

[11]「スーパーコンピュータ」とは、一般に「その時代の汎用の計算機と比較して格段に高速の計算機」のことをいいます。

FORTRAN じゃないの？

Fortran という名称は Formula Translation の略です。日本語にすると「数式翻訳」といった意味になります。

Fortran は代々 “FORTRAN” と、全て大文字で書くのが普通でした。しかし、新しい Fortran 90 から、頭の F だけ大文字で、あとは小文字で書くようになりました。現在の規格は Fortran 90 ですので、言語としての名前も “FORTRAN” ではなく、“Fortran” と書くのが正式のようです。

そういう訳で、コンパイラの名前が大文字となっているものは、例えば “FORTRAN77 EX” などと書きますが、それ以外は “Fortran” で統一することにします。

FUJITSU M-1800/20

ベクトル計算機の威力

速いぞベクトル計算機

実験 I の結果をみると、ベクトル計算機の処理がいかに高速かよくわかります。倍精度演算で比較すると、ベクトル計算機は汎用機に対して、おおよそ

DVLAX で　103 倍
DVMGGM で　120 倍
テストプログラムで　181 倍

のスピードを示しています。

もっとも、ベクトル計算機はその名前の示す通り、もともとこのようなベクトル・行列演算を高速に行うために設計された計算機であって、逆にこれくらいの性能が出ないと、その存在理由[12]が失われてしまいます。

ベクトル処理による高速化のからくりを知りたい方は、ぜひ [11] を御覧下さい。

では、みなさんがよくお使いになっているパーソナルコンピュータやワークステーションと比べて、ベクトル計算機がどれくらいの性能をもっているのか、実験しましょう。

不公正にも、ベクトル計算機がもっとも得意な土俵で対決させます。その代わりに SSL II サブルーチンは使わずに、同一の Fortran プログラムで勝負します。

実験 II

パーソナルコンピュータやワークステーションは、行列積演算で汎用機やベクトル計算機と渡りあうことができるか？

その前に、いくつか用語の説明をします。

FLOPS という指標

FLOPS とは Floating Point Operations per Second の略です。読んでわかるとおり、1 秒間に得られる浮動小数点演算の結果の数を表します。FLOPS を単位として MFLOPS, GFLOPS など、メガ・ギガの値がよく使われ、計算機の演算性能を表す代表的な指標の一つになっています[13]。

収束判定などを用いたりして、実行してみなければ演算回数がわからないプログラムでなければ、浮動小数点演算 (四則演算) を何回やるか数えることができます。すると、実行時間を計る関数を使い、FLOPS 値が計算可能です。

```
      do 10 i=1,n
      do 10 j=1,n
      do 10 k=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

上の行列積を求める DO ループの演算数も、簡単に数えることができます。各変数 i, j, k 毎に、加法と乗法を一回づつ行っているので、$2 \times n \times n \times n = 2n^3$ 回の演算数です。

このループの前後に実行時間を計測する関数を埋め込んでおけば、一秒間あたり何回の浮動小数点演算を行ったかの FLOPS 値を

$$\mathrm{FLOPS} = \frac{2n^3}{\text{時間(秒)}}$$

で求めることができます。

行列積の型

先ほどの行列積のプログラムにおいて、i, j, k の順序を変えてもアルゴリズムとしては等価です。

行列積のプログラムとしては、c の初期化が必要ですが、ここでは考えないことにします。

制御変数のとり方は 6 通りあります。DO ループに登場する順番に応じて以下のように名前をつけます。

IJK 型

```
      do 10 i=1,n
      do 10 j=1,n
      do 10 k=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

JIK 型

```
      do 10 j=1,n
      do 10 i=1,n
      do 10 k=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

[12] 自分をひとかどの人物に見せたがる人は、気取って “raison d’hatetre”(レゾンデートル) などといいます。

[13] 絶対的な性能を意味するものではありませんが、有力な性能の指標です。

KIJ 型

```
      do 10 k=1,n
      do 10 i=1,n
      do 10 j=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

KJI 型

```
      do 10 k=1,n
      do 10 j=1,n
      do 10 i=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

IKJ 型

```
      do 10 i=1,n
      do 10 k=1,n
      do 10 j=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

JKI 型

```
      do 10 j=1,n
      do 10 k=1,n
      do 10 i=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
```

さて、結果としては等価なこれらの型ですが、いざ翻訳・実行すると、実行時間にかなりの差があることが知られています。

Fortran は主記憶へのアクセスを列優先に行うため、行を優先してアクセスするプログラムの場合、主記憶へのアクセスが不連続になり、処理速度が低下することがその理由です。なお、詳しい解説が [1] にありますので、参照下さい。

実験 II は、実験 I で用いた汎用機 M-1800/20 と、ベクトル計算機 VP2600/10 に加え、パーソナルコンピュータとして DEC 社の AlphaStation 200[14] を、ワークステーションとして Sun Microsystems 社の SPARCstation 10 をテストに使いました。環境は表 6 にまとめました。

最適化オプション

表 6の「オプション」とは、測定の際に指定した最適化のオプションです。

最適化 (optimization) とは、プログラムを可能な限り高速に実行できるような命令列及びデータ域を生成することです。つまり、「頑張って速く動かせ！」というコンパイラへの命令です。最適化機能は、各計算機やコンパイラによって異なりますが、基本的にはコンパイラが論理的に同値な範囲でプログラムを書き換えることをいいます。

おおよそ、オブジェクトモジュールの生成段階で、次のことが起きます。

削除：
実行しない文や、同じ結果となるところを消してしまう。

移動：
ループ内にあるが、同じ結果になることがわかっているものをループの外に放り出す。

変更：
演算式の計算順序やデータを変える。

複写：
ループ内の繰り返し範囲を何重かに複写して繰り返し回数を減らす。有名な「ループアンローリング」のこと。

展開：
組み込み関数などのコードを、引用場所に組み込む。

実行：
翻訳時に実行結果がわかるものは値を確定させ、実行させない。

その他：
ハードウェアにあわせてもっと頑張る。

論理的に同値とはいえ、プログラムを書き換えるのですから、ひょっとしたら利用者の予期しない結果となる可能性もあります。普通、何もオプションを指定しなければ、最適化は最低限の機能に押えられます。しかし、利用者の責任で積極的に最適化をおこなうことで、格段の実行性能が得られるプログラムも数多く見られます。

詳しい最適化の機能や、副作用の注意などは、お使いのコンパイラによって違いますので、各マニュアルをお読み下さい。

なお、 M-1800/20 と VP2600/10 の最適化オプションはそれぞれ `OPT(E)` を指定しました。

行列の定義

配列は VP2600/10 のバンクコンフリクトを避けるため、 1025×1025 で宣言します。行列は 1024×1024 のフランク行列

$$\begin{cases} A = B = (b_{ij}) \\ b_{ij} = n + 1 - \max(i, j) \end{cases} \tag{2}$$

[14] 取り扱い説明書に「パソコン」と書いてありました。

図 1: 行列積の処理性能 (MFLOPS)

[1] では、各型についてベクトルアルゴリズムの観点から次のような考察を加えています。

- JIK 型はベクトルアルゴリズムとしては不適切。
- IJK 型はまあまあ。
- KJI 型の方が KIJ 型よりもバンクコンフリクトの観点から見て良い。
- IKJ 型は Fortran の環境では勧められない。
- JKI 型が主記憶のアクセスの立場からみて一番いい。

さらに、証拠として、VP2600/10 の先祖である VP-200 を用いた数値が掲載されています。それによれば、JKI 型が最も速く、IJK 型もそこそこの性能を出していることが報告されています。

この本が出たのが 1988 年です。おそらくこの本を読んだ富士通のコンパイラ部隊が奮起されたのでしょう。VP-200 の後継の VP2600/10 では、どの型に対してもほとんど同じ性能を出すことに成功しています。これはベクトル化コンパイラ FORTRAN77 EX/VP が賢くなったためで、DO ループの繰り返し回数が翻訳時に判っているものは、VP2600/10 のハードウェアを最も効率的に使用で

図 2: スカラー演算での行列積の処理時間 (秒)

りも性能が劣ることがわかりました。

みなさんの書かれるプログラムは、行列の積だけでもなければ、ベクトル化コンパイラがギブアップするほどのスカラー専用プログラムでもないと思います。つまり、桁違いの性能を示す図 1 と、汎用機並のお粗末さを見せる図 2 の中間に位置するはずです。

自動ベクトルコンパイラがいくら賢くても、その解釈にはどうしても限界があります[17]。従って、ベクトル計算機の性能を最大に引き出すためには、[11] などの参考文献をしっかり読んだ上、大いにプログラムを改良する必要があります。

そんなわけで、あたりまえの結論を得ました。

結論 II

もしベクトル計算機の性能を十分に引き出すプログラムを組めば、桁違いの性能が得られる。

将来、上の「ベクトル計算機」の部分は「ベクトル並列計算機」になっているはずです。

そうはいいますが、じっくり腰を据えてプログラミングの勉強をするには、なにかと雑用の多い世の中です[18]。

てっとり早く実行性能を上げるにはどうすればよいか。一番簡単な方法は、black box になるのは覚悟の上で、専門家が注意深くプログラムしたサブルーチン・ライブラリを利用することです。センターでは SSL II/VP, NUMPAC/VP がそれにあたります。

[17] ましてや将来のスーパーコンピュータの有力な候補である並列計算機の性能を十分に引き出す「自動並列コンパイラ」には、まだ決定版がありません。

[18] 特に九州大学は移転や再開発計画にともなって、みなさんお忙しそうです。

ベンチマークの話

計算機の性能を評価する問題をベンチマーク (benchmark) と呼びます。普通は、ある目的に沿って作られた性能評価プログラムを各計算機上で実行し、速度や効率を測定することを指します。

「計算機の性能」といっても、様々なことが考えられます。数値計算の分野ならば、浮動小数点演算の計算速度に関心が集まりますし、記号処理の世界では再帰呼び出しやリスト処理の性能が大事です。また、データベースの世界では、効率的なディスクへのアクセスが要求されます。

記号処理やデータベース、画像処理の分野などでは、目的が分散しているため、計算機にプラスして、その上にのっかっているソフトの頭の良さが大きく問われます。そのため、定量的な性能評価となるベンチマークプログラムを作るのは大変な作業です。

一方で、スーパーコンピュータのベンチマークに限ってしまえば、「でかい Fortran や C プログラムが、ある程度の精度を持ってバシバシ実行できる。」ことが絶対の正義ですので、例えば、先ほどやった「行列積演算」という切り口を提示すれば、一応定量的で公正なベンチマークプログラムの１つと強弁することが出来ます[19]。

数値計算の分野では、その道の専門家が作成した性能評価プログラムがたくさんあります[20]。

ある機関でスーパーコンピュータを調達しようとする場合は、その機関の実情にあったベンチマークプログラムを用意することが一般的です。例えば、大型計算機センターなどでは、利用者の応用プログラムがどのくらいの性能で実行できるかが、大きなポイントです。

このような性能評価は、コンピュータの分野に競争があるからこそ必要になるものです。一定の枠や予算が限られている中で少しでも優秀なものが欲しい場合は、なにがしかの「客観的」選別基準が必要です。

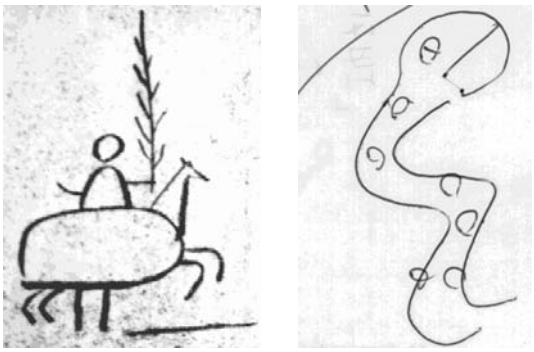
かと言って、これは計算機など特定の目的のために作られたものに限ったことです。計算機は「速く」なければ役に立ちません。役に立たないものは捨てられることになります。そういう意味で、大学入試や国家公務員試験は、一応定量的で公正な「客観的」選別基準ですが、これを「ベンチマークプログラム」と言うのは、ちょいと差し障りがあるでしょう。

[19]ただし、行列積だけ反則のようにものすごく速くて、割り算やファイルアクセスなどはたいしたことないという計算機があるという噂を聞きます。

[20]有名なところで Dhrystone, Whetstone, Livermore Fortran Kernels, NAS Kernel, Linpack, SPEC, PERFECT Benchmark などです。

芸術のベンチマーク

計算機のベンチマークにおける「客観的」基準という話が出ました。ではここで問題です。下の絵はどちらが芸術的でしょうか？



極めて抽象的な画風？

左の絵は、スペインに出張した人の見せてくれた写真にありました。なんでも、バルセロナ市内の壁に描かれているデザインの一部で、どうやら作者はパブロ・ピカソさんだということです。

対して右は、自称画家兼詩人を名乗るある助教授が、多少酔っぱらって描いた意欲的な作品です。

FUJITSU VP2600/10

連立 1 次方程式の性能比較

連立 1 次方程式

自然界に起こる現象を定式化し、数値計算によってそれを処理しようとすると、多くの問題が最終的に**連立 1 次方程式** (linear equation) に帰着されます。これは、数学の言葉で書かれた非線形微分 (積分) 方程式を離散化・線形化する過程から導かれるものです。

解くべき連立 1 次方程式を行列形式で

$$Ax = b$$

と書きます。差分法や有限要素法などの手法を用いている場合、行列 A は成分の大部分が 0 となる疎行列 (sparse matrix) で、しかも 0 でない成分が対角線の近くにかたまって分布している帯行列 (band matrix) となることがほとんどです[21]。

$$\begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 5 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 6 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 7 \end{bmatrix}$$

9×9 の実 3 項行列の例

どうやって解く？

連立 1 次方程式の解法は、大きく分けて直接解法と反復解法とがあります。直接解法は、A を有限回いじくり回し、b の値を使うことで x を求める方法です。行列が正則で、丸め誤差の影響を無視すれば必ず解が得られます。

対して反復解法は、あるべき解に収束していく近似解の系列を作り出していく方法です。反復解法の利点は、A をいじくる必要がなく、大型な疎行列の場合は、少ない記憶領域で解を得ることができることです。ただし、A の性質によっては収束の速さが異なり、場合によっては収束しないことがあります。

直接解法の利点は、有限回の操作で解を得られることと、アルゴリズムがすっきりしていて、行列を“ぶんまわす” ベクトル計算機によくマッチして、一般に処理が速いことです。ただし、行列の変形がともなうため、その分の記憶領域が必要です。

Gauss の消去法

記憶領域をたくさん確保できるスーパーコンピュータの世界では、直接解法で解を求めるサブルーチンがよく使われます。直接解法で最も代表的な手法は**Gauss の消去法** (Gaussian elimination) と呼ばれるものです。

連立 1 次方程式の解法には、直接解法に Gauss-Jordan 法、反復解法に Gauss-Seidel 法などがあり、“Gauss” さんがあちこちに出没してとても紛らわしい世界です。ちなみに「掃き出し法」とは Gauss-Jordan 法のことです。 Gauss-Jordan 法は Gauss の消去法の変形ですが、計算量が Gauss の消去法に比べて 1.5 倍もかかるため、実際はあまり使われません。早くも混乱してきた人は、是非 [13] をお読み下さい。

通常、 Gauss の消去法では、行列 A を下三角行列 L と上三角行列 U に分解する LU 分解 (LU decomposition) を行ないます。そして、分解された行列 L と U を用いて解 x を求めます[22]。

さて、 A が帯行列の場合、 LU 分解の過程で必要な領域は対角線近くの領域に限られ、その他は 0 のままで済むことがわかります。従って、添字に注意すれば、領域節約型のプログラムを組むことも可能です。また、行列が対称の場合や「正定値」といわれるとても良い性質を持つ場合は、さらに領域の節約・高速化が図れます。以上、詳しい手法の解説は [2] を御覧下さい。

Gauss の消去法にかかるコスト

以下は、 A の成分が 0 でない密行列 (dense matrix) で考えます。疎行列・対称行列・帯行列などに対する解法は全て密行列の変形であり、密行列を考えることが基本となります。

n 元連立 1 次方程式を Gauss の消去法で解く場合の演算量は、次のようになることがよく知られています。

[21] なぜそうなるかは、プログラムを組めばすぐにわかります。

[22] この過程を前進消去 (代入)、後退代入と呼びます。

Gauss の消去法の演算量は、十分大きな n に対し $3n^3/2$ となる。しかもそのほとんどは LU 分解に要するコストである。

上から二つのことが分かります。

- A はそのままで、b を何度も変えて方程式を解く場合、一度 LU 分解しておけば、効率良く解が得られる。
- 連立 1 次方程式のプログラムの前後に時間計測関数を置くことで、おおよその FLOPS 値が計算できる。

ここまで準備したところで、次の興味深い実験をします。

実験 III

ベクトル計算機で Gauss の消去法を用いて連立 1 次方程式を解く時、アルゴリズム通り組んだプログラムと、その道のプロが作ったプログラムとでは、速度・精度にどれくらいの違いが出るか？

その前に、すみませんが、用語の説明をさせていただきます。

ピボット選択

Gauss の消去法の過程では、変形前の A の対角成分に対応する要素での割り算が必要です。この部分をピボット (pivot) と呼びます。さて、割り算ですので、もしピボットが 0 だったり絶対値が極めて小さい場合は、実行不可能になるか、解の精度が著しく悪くなることが知られています[23]。

その場合、列方向に代わりのピボットに相応しい数を探し、もしあったなら、対応する行を入れ換えるという操作を行うことで、その事態を回避することが出来ます。この操作を「ピボットの部分選択 (partial pivoting)」と呼びます[24]。

また、行だけでなく、列の入れ換えもあわせて行う「ピボットの完全選択 (complete pivoting)」もありますが、コストが莫大なことと、実用的にはピボットの部分選択でほとんど問題ないことから、滅多に見ません。

[23]特に非線形の問題を無理矢理線形化して解く場合は、行列の性質が悪くなるので、注意が必要です。

[24]「部分的ピボッティング」とも呼びます。また、本によっては「枢軸対策」という、三国同盟の向こうをはるようないかめしい呼び方もしています。

ピボット選択は、Gauss の消去法のプログラミングをする場合、是非とも組み込むべき手法です。計算コストは LU 分解に比べて無視できるほどですし、様々な行列に対して安定した精度を得られます。また、比較するプログラムがすべてピボット選択を行っていますので、対抗上、組み込まざるを得ません。

数値計算の本のアルゴリズムを見ながら Gauss の消去法のサブルーチンを作ってみました。

```
      SUBROUTINE MYLU(A,B,N,NX,P,VW)
      DOUBLE PRECISION A(NX,1),B(1),VW(1),D
      INTEGER N,NX,I,J,K,L,P(1)
!---------- PARTIAL PIVOTING ------------------
      DO 10 K=1,N
      P(K)=K
 10   CONTINUE
      DO 20 K=1,N
      D=0.0D0
      DO 30 I=k,N
      IF(ABS(A(P(I),K)).GE.D) THEN
      D=ABS(A(P(I),K))
      L=I
      END IF
 30   CONTINUE
      IF(K.NE.L) THEN
      J=P(K)
      P(K)=P(L)
      P(L)=J
      END IF
!---------- LU DECOMPOSITION ------------------
      A(P(K),K)=1.0D0/A(P(K),K)
      DO 40 I=K+1,N
      A(P(I),K)=A(P(I),K)*A(P(K),K)
      DO 50 J=K+1,N
      A(P(I),J)=A(P(I),J)-A(P(I),K)*A(P(K),J)
 50   CONTINUE
 40   CONTINUE
 20   CONTINUE
!---------- FORWARD ELIMINATION ---------------
      DO 60 I=1,N
      VW(I)=B(P(I))
      DO 70 J=1,I-1
      VW(I)=VW(I)-A(P(I),J)*VW(J)
 70   CONTINUE
 60   CONTINUE
!---------- BACKWARD SUBSTITUTION -------------
      DO 80 I=N,1,-1
      B(I)=VW(I)
      DO 90 J=I+1,N
      B(I)=B(I)-A(P(I),J)*B(J)
 90   CONTINUE
      B(I)=B(I)*A(P(I),I)
 80   CONTINUE
      RETURN
      END
```

サブルーチンの名前は、安直に MYLU としておきます。行列が正則でない場合や、ピボットが異常に小さくなってしまった場合の処理がないので、まだ修正の必要がありますが、一応、正しい結果を得ることが期待されるプログラムです。

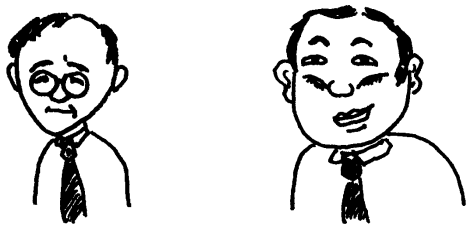
プロが作ったサブルーチン

続いて、数値計算のプロフェッショナルが作った Gauss の消去法のサブルーチン群の入場です。なおソースプログラムは、残念ながら著作権者の許可をもらっていないのでお見せ出来ません。

表 8: 実験 III に使用するサブルーチン

サブルーチン名	登録パッケージ名
DLAX	SSL II/VP
DVLAX	SSL II/VP(拡張機能版)
LEQLUW	NUMPAC/VP
DGEFA & DGESL	Linpack

すべて倍精度 (64bit) 用のサブルーチンです。 Linpack は LU 分解用のサブルーチン DGEFA と分解された行列を用いて解を決定するサブルーチン DGESL に分かれており、その合計で時間を計測します。



数値計算のプロ (想像図)[25]

測定機種は、ベクトル計算機 VP2600/10 です。コンパイラ等の環境は実験 I, II と同様です。 DLAX, DVLAX, LEQLUW は、既に登録されているロードモジュールを呼び出して利用します。また、 DGEFA & DGESL と MYLU は、サブルーチンのソースプログラムをメインプログラムに続けて記述し、翻訳・実行させました。全て最適化オプションは `OPT(E)` です。

Linpack について

Linpack はアメリカ Argonne 国立研究所の Dongarra さんの研究グループによって開発されたサブルーチン・ライブラリです。

[25]左は研究一筋のタイプ。ただし飲み屋で豹変する場合があります。右も... 研究一筋の人ですが、こちらは飲み屋でもあまり変わりません。あくまでも想像図です。

今回テストに用いた DGEFA と DGESL は、内部の DO ループの中で BLAS と呼ばれる線形計算プログラムのいくつかを CALL しています。

で、 DO ループ内でサブルーチンを呼ぶという行為は、現在の自動ベクトル化コンパイラにとって実に困ったことでして、大いに最適化の妨げになり、結果的に性能が極端に低下します。詳しくは [14] を御覧下さい。

つまり、 DGEFA & DGESL は Fortran ソースを単純に翻訳しても、そのままではベクトル計算機の性能がうまく引き出せないことになります。そのことを分った上で、実験に参加させます。

測定プログラム

各サブルーチンの精度も併せて測定したいので、数学的に exact な解がわかるものを [15] から選びました。次元は n とします。

方程式 I

方程式 I は、行列とベクトルを

$$\begin{cases} A = (a_{ij}) \\ a_{ij} = a_{ji} = n + 1 - i \quad (\text{if } \ i \geq j) \\ b = (b_i) \\ b_i = \displaystyle\sum_{k=1}^{n-i+1} \frac{n-k+1}{n} \end{cases} \tag{3}$$

で定義し、 $Ax = b$ を各サブルーチンで解きます。解は正確に $x = (1/n, 1/n, \cdots, 1/n)$ となります。

方程式 II

方程式 II は、 (1) と同じです。もう一度書くと

$$\begin{cases} A = (a_{ij}) \\ a_{ij} = \sqrt{\dfrac{2}{n+1}} \sin \dfrac{\pi ij}{n+1} \quad (i, j = 1, 2, \cdots, n) \\ b = (f_i) \\ b_i = \displaystyle\sum_{j=1}^{n} b_{ij} \quad (i = 1, 2, \cdots, n) \end{cases}$$

で定義した $Ax = b$ を解きます。解は正確に $x = (1, 1, \cdots, 1)$ となります。

時間は CLOCK 関数をサブルーチンの前後に埋め込んで測定しました。また誤差を error とすると、真の値を $\hat{x}_i$ に対し相対誤差の最大値を

図 4: Gauss の消去法の処理能力 (MFLOPS)

VP2600/10 のピーク性能は 4.9GFLOPS です。DVLAX は、その性能に迫る値をはじき出しています。これはかなり驚異的な数値で、ハードウェアを知りつくした優秀なスタッフが “ゴリゴリ” とチューニングした結果だと思われます。

DVLAX は別としても、 DLAX や LEQLUW は純粋に Fortran で書かれたサブルーチンです。また DLAX のソースプログラムは、他の計算機機へ移植しないという前提のもとで公開されています。ということは、 Fortran サブルーチン MYLU をベクトル計算機向きに書き直す余地がかなりあるということです。またうまく書き直すことが出来たなら、 4 倍近い性能が出ることが、データからわかります

誤差は？

次は、気になる精度です。表 9, 10を御覧下さい。

やはり誤差は n が大きくなるに従って増大しますが、基本的なアルゴリズムが同じなので、極端な差は出ていません。

なお、相対誤差の計算も倍精度で行っています。大規模な行列演算では、いくら専門のライブラリを用いたとしても、行列の種類によってはかなりの誤差が累積することがわかります。

SSL II/VP には、得られた解を反復修正して改良するサブルーチン DLAXR を装備しています。例えば方程式 II くらい誤差がある場合は、 (問題にもよ

表 9: Gauss の消去法における誤差 (方程式 I)

サブルーチン	$n = 1000$	$n = 7000$
DLAX	0.282053E-09	0.167452E-07
DVLAX	0.282053E-09	0.167452E-07
LEQLUW	0.224743E-09	0.829077E-07
MYLU	0.598924E-09	0.217968E-07
DGEFA&DGESL	0.212306E-09	0.571381E-07

表 10: Gauss の消去法における誤差 (方程式 II)

サブルーチン	$n = 1000$	$n = 7000$
DLAX	0.361267E-12	0.101252E-11
DVLAX	0.357311E-12	0.110334E-11
LEQLUW	0.158526E-12	0.871081E-11
MYLU	0.116365E-12	0.502862E-12
DGEFA&DGESL	0.129854E-12	0.625944E-11

りますが) 精度の改良を積極的におこなうべきだと思われます。

精度があまり変わらない以上、高速なライブラリを使った方が大いに得策です。従って、この実験での結論は次のようになります。

結論 III

ベクトル演算を意識しないプログラムと、専用のライブラリでは、数倍の性能差が出ます。従って、ベクトル計算機の性能を上手に引き出すには、専用のライブラリを積極的に利用しましょう。

Carl Friedrich Gauss

この章では “Gauss” という名前がむやみに出ましたので、ついでに Gauss さんの経歴を詳解します。

Carl Friedrich Gauss さんはドイツ Braunschweig という街で 1777 年 4 月 30 日に生まれました。亡くなったのは 1855 年 2 月 23 日です。

Gauss さんは、幼い頃から異常な数学的才能を示し、「複素係数をもつ方程式は必ず複素数の根を有する」という有名な代数学の基本定理を 22 才のときに証明して学位をもらいました。

整数論、非 Euclid 幾何学、超幾何級数、複素関数論、楕円関数論ですぐれた業績をあげ、最小 2 乗法、曲面論、ポテンシャル論の創始者でもあります。19 世紀前半最大の数学者として純粋数学にも応用数学にも大きな足跡を残しました。

こんな顔をしています。

Carl Friedrich Gauss さん

また、後年は天文学、測地学、電磁気学の分野でも不朽の貢献をしましたが、生涯整数論を熱愛していて、「数学が科学の女王ならば、数論は数学の女王である」と言っていたそうです。

ざっと現在に残る Gauss がつくものをあげると、

- 磁束密度の単位である Gauss
- Gauss 計
- Gauss 像
- Gauss の公式 (曲面論)
- Gauss の超幾何微分方程式
- Gauss の定理 (ベクトル解析学)
- Gauss 波
- Gauss 分布
- Gauss 平面

などあちこちの分野で見ることができます。

Braunschweig 大学の近くには、Gauss さんの人類への偉大な貢献を讃えた巨大な銅像がそびえています[27]。

[27] 以上、『岩波数学辞典』『平凡社世界大百科事典』によりました

数学関数の精度

数学関数

この章では、速度の話はお休みにして、数学関数について少し考えます。 Fortran や C, C++ など、科学技術計算への応用に (も) 使われる言語は、日本工業規格 (JIS) の定める規格において、 sin, cos, log などの基本的な数学関数を備えることが要求されています。

ただし、数学関数がどのようなアルゴリズムで組まれるかはメーカーまかせで、「単精度で 7 桁、倍精度で 15 桁は正しい値を返さない場合は不良品と見倣す。」などといった精度に関する規定は定められていません。

つまり、利用者にとって数学関数は完全な black box であって、「とりあえずそれっぽい結果みたいだから、まあ信用しておこうか」といった対応がほとんどだろうと思います[28]

ところが、 1985 年に出版された本を読むと、次のような恐ろしい記述があります。

> 基本的なソフトであるコンパイラについて言えば、比較的最近まで「数値計算に関係する部分は新入社員が研修の一つとして作ればよい」というくらいに考えていたわけでもなかろうが、わが国最大の計算機メーカの最大機種の FORTRAN で CABS(複素数の絶対値) を呼んだら偏角が出てきたとか、その他の計算機でも倍精度にしても割算の結果が単精度しか出ないとか、組み込み関数の精度が目茶苦茶であるとか、とにかくいろいろ面白い話題の種は尽きなかった (現在ではもうそんなことはない — と信じたい)。
>
> 伊理正夫, 藤野和建 『数値計算の常識』

「わが国最大の計算機メーカ」が何処かを追及するのはさておき、現在日本でスーパーコンピュータを製造している会社は 3 社あります。その一つの富士通株式会社を例にとると、 Fortran コンパイラの使用手引書 [9] において、数学関数の精度をきちんと示しています。このマニュアルより、代表的な関数について精度を引用します。

[28] 信用できない人は、自分で関数を作ることになります。しかし、何も考えずに Tailor 展開を近似すれば全て大丈夫かといえば、そうでもありません。慎重な丸め誤差対策が必要です。

ただし精度の比較は、単精度は倍精度と、倍精度は 4 倍精度と比較した結果だそうです。また、引数の範囲や誤差ノルムの設定が各関数でまちまちなので、大体の目安として御覧下さい。

関数名	単精度	倍精度
sqrt	4.16E-7	9.95D-17
exp	4.56E-7	1.79D-16
log	4.68E-7	2.33D-16
sin	3.28E-7	6.94D-17
cos	4.18E-7	9.70D-17
tan	8.17E-7	1.98D-15
arctan	7.01E-7	2.12D-16

FORTRAN77 EX 組み込み関数の精度

π の計算

疑うわけではないですが、 FORTRAN77 EX の組み込み関数 `ATAN` を使って、円周率 π がちゃんと求まるかどうかのテストをします。

[16] によると、 π と逆正接関数 arctan を結びつける関係式は 249 個知られています。その中の幾つかの公式を使って、 π を生成します。公式は次の 6 つです[29]。

$$
\begin{aligned}
\pi = &\ 4\,(\arctan\frac{1}{2} + \arctan\frac{1}{3}) && \text{Euler}\\
= &\ 4\,(2\arctan\frac{1}{3} + \arctan\frac{1}{7}) && \text{Clausen}\\
= &\ 4\,(4\arctan\frac{1}{5} - \arctan\frac{1}{239}) && \text{Machin}\\
= &\ 4\,(4\arctan\frac{1}{7} + 2\arctan\frac{3}{79}) && \text{Euler-Vega}\\
= &\ 4\,(\arctan\frac{1}{2} + \arctan\frac{1}{5} + \arctan\frac{1}{8}) && \text{Dase}\\
= &\ 4\,(4\arctan\frac{1}{5} - \arctan\frac{1}{70} + \arctan\frac{1}{99}) && \text{Rutherford}
\end{aligned}
$$

右に書いてあるのが公式の名前です。テストは、汎用機 M-1800/20 を用いました。表 11 が実行結果です。

どの公式を使っても、単精度で 6 桁、倍精度で 15 桁は正しい値を返しました。 M-1800/20 の浮動小数点は IBM 形式なので、表 1 と見比べても、これはまずまずの結果と言えるでしょう。

しかしながら、上の公式から組み込み関数を用いて数値的に値を引っ張ってくれば、必ずわずかな

[29] 最も有名なものに $\pi = 4\arctan(1)$ がありますが、つまらないのでやりません。

図 5: 公式 A の誤差

誤差が n の order で増加していくことがよくわかります。

arctan の有限和

続いては、先ほどの π の計算ではなかなかの技を見せた arctan に関する公式です。

公式 II

$$\sum_{r=1}^{n} \arctan \frac{1}{r^2+r+1} = \arctan \frac{n}{n+2}$$

これも、両辺を計算する必要があります。そこで、演算数が圧倒的に少ない (割り算 1 回と組み込み関数 1 回) 右辺を 4 倍精度で求め、真の値だと勝手に仮定した上で、倍精度演算で求めた左辺との相対誤差を $n = 2^1, 2^2, \cdots, 2^{30}$ について評価しました。図 6 は n を横軸にとった相対誤差の分布です。

これも n に比例して誤差がどんどん増えていきます。

1 の n 乗根

図 8: 公式 D の誤差

御覧のように、 NUMPAC の組み込み関数 DCOTHP を利用した方が、通常の COT 関数を使うよりも 2 桁ほど精度が良いことがわかります。ただし、実行時間は NUMPAC の方が約 2 倍かかります。

COS の公式

これまでの公式では、徹底的に和や積をとっても、それなりに踏んばっていましたが、最後の公式の結果は、芳しくありません。

公式を用いたこれらの実験は、どれもが極端に多い演算数を要求していました。結局、数値結果の精度を保つためには、数学的に同値でしかも少ない演算で済む式があるなら、出来るだけそっちに置き換えた方がよいということです。

これはプログラム全体にもいえることで、対称性がわかっている場合は、無用な計算をしないように工夫するとか、数式処理で出てきた多項式演算をそのまま C や Fortran に流用する時には、出来るだけ共通項でくくるなどの地味な努力が、計算の信頼度を高めることになります。

図 9: 公式 E の誤差

図 9 では、誤差グラフは直線にならず、“にょろにょろ” とした線を描きながら増加していきます。そして $n = 2^{30} = 1073741824$ では、最大誤差が 30 にもなってしまいました。ということは、左辺の絶対値が 15 程度にまで膨らんでしまったわけです。

もっとも、関数の和を 10 億回以上もやる数値計算がそんなにたくさんあるとも思えませんが...

結論

数学関数のテストは、全て M-1800/20 での実験結果です。この実験は、数学関数の精度と浮動小数点形式、アンダーフローが起きた場合の処理形態などで結果が違ってくると考えられます。IBM 形式と IEEE 形式の比較や、他のプログラミング言語での調査、また、実行時間を測るベンチマークなど、面白そうなデータが出そうですが、今回はこれくらいでやめることにします。

では、結論を。

結論 IV

組み込み関数をハードに足し合わせたり、徹底的にかけ合わせたりするのは、やっぱり良くない。

「1 つの数を上手に計算するためには、その数を親しみをもって知らなくてはならない[32]」

[32] π の計算で有名な David Chudnovsky さんの言葉。

C と Fortran

プログラミング言語

現在、数値計算の世界で用いる言語は、 Fortran, C, C++, Pascal などです。アセンブラでゴリゴリと書く方もいますが、一部の計算機センターが計算機を独占していた時代はとっくの昔に去り、計算機資源が研究室や自宅や電車の中や旅先の温泉宿[33]まで浸透しているご時世なので、移植性に優れ[34]、かつ多くの人が使っている言語を少なくとも一つ知っていないと、悲しい思いをします。

私の知る限り、数値計算でのプログラム言語で現在もっとも使われているのは C と Fortran です。この二つの言語は敵同士のような扱いを受けることがよくありますが、それはプログラム言語に “美” を見い出そうとする人の理想に燃える言説であったりして[35]、プログラム言語をアルゴリズムから数値データに変換する「道具」だと割り切って用いる人たちは、二つの言語のそれぞれの長所を生かして上手に使い分けています。

数値計算の分野でも近年 C が使われはじめましたが、やはり Fortran でプログラムを組む人が多数です。 [7] の文章も自信に満ちています。

> 30 年たってみて、 Fortran は、大半の計算機において使用できるただ一つのプログラム言語であるとは、とてもいえない存在である。その間、新しい言語が次々に開発されてきた。特定の種類の応用に適合した言語は、 Fortran よりも優先して採用された。 Fortran の優越はつねに、数値計算、科学、工学、技術の応用分野にあった。これらの分野では、顕著な競争相手はいまだにない。
>
> M. Metcalf, J.Reid 『詳解 Fortran 90』

流体のシミュレーションなどの大規模な数値計算をスーパーコンピュータで行なったりする研究者が Fortran を使うわけは、単に現在のスーパーコンピュータで Fortran が幅をきかせているためであって、もっと大規模計算に適した優れた言語が登場すれば、さっさと乗り換えてしまうでしょう[36]。

とはいっても、 Fortran が科学技術分野の主力言語であるという現実は、とりあえず今世紀中は変わらないでしょう。その理由は

- C のベクトル化技術がまだまだなこと。
- C の数値ライブラリの充実がまだまだなこと。
- Fortran 90 と、次の Fortran 95 が頑張りそうなこと。
- 研究室に Fortran プログラムの膨大な蓄積があること。

といったところです。

Fortran 90 では、新しく配列演算やポインター、利用者定義の構造データ型や動的記憶割付けなど、 C を大いに真似た機能を追加しています (cf.[7])。更にプログラムの記述も、昔のカード時代の呪縛を脱し、自由な形式で書けるようになりました。

どっちが速い？

C と Fortran の得意技を考えると、何となく

- Fortran は数値計算が得意
- C は文字列処理やシステムコールが得意

といったイメージがあります。多分その通りでしょう。しかし、役割が違うといっても、同じ計算機に言語が二つのっていると、やっぱり調べたくなります。そこで、数値計算の分野で対決させることにしました。

実験 IV

Fortran と C はどっちが速い？

二つの簡単なプログラムを作り、実行時間を調べることにします。なお、最適化のコンパイラオプションの指定によって実行時間に差が出ます。そこで、 VP2600/10 についてはスカラー処理ユニットを使っての実行結果と、最適化オプションを指定し

[33] モデム内蔵のノートパソコンを常に持ち歩く人が増えました。中洲には、 ISDN 経由でインターネットに接続できるスナックがあるそうです。そのうち「スナック××に今すぐ来い！」などといったボスからの呼び出しメールが来るかも知れません。恐いですね〜

[34] どの計算機に持っていってもすぐに動く。

[35] ある国際学会で、ドイツの数値解析の研究者が、うるんだ目をしながら Pascal の美しさを讃え、片や憎悪に満ちた口調で C や Fortran をけなしている講演を聞いたことがあります。肝心の講演内容は忘れてしまいました。

[36] もちろん、長年使えば愛着は生じます。しかしプログラム言語では、あまり「こだわり」を持たないでドライに考えていた方が得策でしょう。

図 10: C と Fortran の実行時間比較 (標準)

なお VP2600/10 の結果は、スカラー処理ユニットを用いた結果です。標準モードでの比較では、SPARCstation で C が勝利しました。汎用機、ベク

図 11: C と Fortran の実行時間比較 (最適化)

最適化を施した結果では、ベクトル計算機と SPARCstation は引き分けといっていいでしょう。汎用機と AlphaStation では Fortran の勝ちです。

ここで、最適化オプションを指定した場合の、VP2600/10 と他機種との速度比較をしましょう。

Fortran プログラムで VP2600/10 は、

M-1800/20 の	35 倍
AlphaStation の	48 倍
SPARCstation の	286 倍

の性能を出しています。

行列演算で勝負だ

続いては、 Fortran がもっとも得意 (そう) な行列演算で実験します。まず、 Fortran のプログラムを用意します。時間計測などの余計なものを除くと、だいたい右上のようなプログラムです。

中身は以下の計算をしています。10 のループで (2) のフランク行列を定義します。20 は行列 c の初期化です。30 は

$$C = A \times B$$

を JKI 型で行います。40 は

$$C - A \times B + A^T \times B^T$$

を計算します。

```
      parameter(maxn=401,n=maxn-1)
      real*8 a(maxn,maxn),b(maxn,maxn),
     &       c(maxn,maxn)
      integer i,j,k
      do 10 j=1,n
      do 10 i=1,n
      a(i,j)=DBLE(n+1-max(i,j))
      b(i,j)=a(i,j)
 10   continue
      do 20 j=1,n
      do 20 i=1,n
      c(i,j)=0.0D0
 20   continue
      do 30 j=1,n
      do 30 k=1,n
      do 30 i=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
 30   continue
      do 40 j=1,n
      do 40 k=1,n
      do 40 i=1,n
      c(i,j)=c(i,j)-a(i,k)*b(k,j)+a(k,i)*b(j,k)
 40   continue
      END
```

この Fortran プログラムを真似て、 2 次元配列を使った C プログラムを作ります。

```
#include <stdio.h>
#define maxn  401
#define    n  maxn-1
main()
{
 double a[maxn][maxn],b[maxn][maxn],
        c[maxn][maxn];
    int i,j,k,z;
      for(j=0; j<n; j++)  {
          for(i=0; i<n; i++)  {
            z = (i>j) ? (i+1):(j+1) ;
            a[i][j]=(double)(n+1-z);
            b[i][j]=a[i][j];
          }
       }
      for(j=0; j<n; j++)  {
          for(i=0; i<n; i++)  {
           c[i][j]=0.0;
          }
       }
      for(j=0; j<n; j++)  {
          for(k=0; k<n; k++)  {
              for(i=0; i<n; i++)  {
               c[i][j] += a[i][k]*b[k][j];
              }
          }
      }
      for(j=0; j<n; j++)  {
          for(k=0; k<n; k++)  {
              for(i=0; i<n; i++)  {
               c[i][j] += -a[i][k]*b[k][j]
                          +a[k][i]*b[j][k] ;
              }
           }
        }
}
```

さて、 C の本を読むと、 Fortran が 2 次元配列を

図 13: C と Fortran の実行時間比較 (最適化)

次は最適化オプションを付加した図 13 を御覧下さい。最適化しても、 Fortran の優位は変わりませんが、 1 次元版との差は、それほど決定的とはいえません。

2 次元版 C プログラムに対する各計算機の性能はまちまちです。 M-1800/20 と SPARCstation10/40 では 1 次元版に 3 倍以上の差をつけられましたが、 AlphaStation では 2 倍以内に迫り、 VP2600/10 では 2 次元版の C が 1 次元版を逆転しました。

スカラー処理では 2 次元版は 1 次元版に 3 倍以上の差をつけられていたことから、ベクトル化がうま

く行われたことがわかります。

最後に、最適化オプションを指定した場合の、VP2600/10 と他機種との速度比較をしましょう。

Fortran プログラムで VP2600/10 は、

M-1800/20 の	74 倍
AlphaStation の	174 倍
SPARCstation の	419 倍

の性能を出しています。

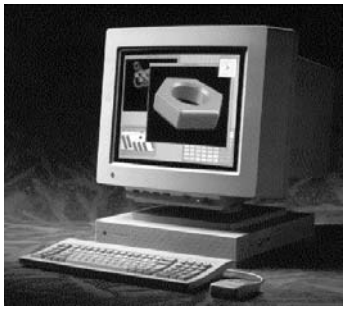
結論

コンパイラの技術は、日々進化しています。これからもどんどん高機能・高品質な製品が世に出ることでしょう。SPARCstation のコンパイラのように、Fortran と C が全く互角の性能を出す計算機・プログラム言語もあります。しかし、テストの結果から一応の結論を出しておきましょう。

結論 V

数値計算を高速にやりたいなら、今のところ Fortran でプログラミングした方がいい。

Fortran から C への書き換えは、f2c などを用いて案外簡単に出来るので、気が変わって C へ乗り換えることは容易です。



SPARCstation

おわりに

現在のベクトル計算機のコンパイラは大変優秀で、そこそこのベクトル化はやってくれます。しかし、スーパーコンピュータの性能を引き出すためには、ある程度自分で勉強し、スーパーコンピュータ向きのプログラムを書く必要があります。

この「ある程度」が曲者でして、もしかしたらサブルーチンを入れ換えるだけで十分かも知れませんし、どんなに頑張っても結局ほとんど性能を上げることができないかも知れません。

記事の中では、スーパーコンピュータ向きの具体的なプログラムの修正方法 (チューニングといいます) は述べませんでした。既に詳しい解説記事や書籍が多数存在するからです。計算機科学や情報処理の世界では、高速な計算機のためのプログラミング技術は重要な分野であり、書店や図書館に行けば必ず見つかります。もし、お手元に Fortran や C の文法書しかお持ちでない方は、是非ともプログラミングの本をお手元に置かれることをお勧めします。

これからやってくる (であろう)「ベクトル並列計算機」の時代は、恐ろしいことに、利用者が並列化およびベクトル化プログラミング技術を知っているのと知らないのとで決定的な差がつくことになりそうです。

つまり、思いっきり具体的に言うと、時間方向まで入った 3 次元 Navier-Stokes 方程式の近似解を有限要素法や差分法で求めるような、ギガバイト・テラバイト単位の大規模数値計算が必要な方にとって、論文完成のためには、否応なしに並列プログラミングの勉強をしなくてはならなくなる時がやってくるかも知れません。

ただし、この頃には優れた解説書や、そこそこの性能を出す並列プログラム変換ツールが出回っているのでは、と私は勝手に期待しています。

illustration : Hidaki Naoko and 周

参考文献

[1] 津田 孝夫 : 数値処理プログラミング, 岩波講座 ソフトウェア科学 9, 岩波書店 (1988).

[2] 森 正武 : 数値解析法, 朝倉現代物理学講座 7, 朝倉書店 (1984).

[3] 名取 亮 : 線形計算, すうがくぶっくす 12, 朝倉書店 (1993).

[4] 佐藤 周行 : コンピュータネットワークのせいで不便になること, 九州大学大型計算機センター広報, Vol.27, No.2, 93–110 (1994).

[5] 精度保証付き数値計算とその応用, 情報処理 Vol.31, Np.9 (1990).

[6] 渡部 善隆 : 数値計算と誤差の話, 九州大学大型計算機センター広報, Vol.27, No.5, 556–580 (1994).

[7] M. Metcalf, J. Reid (西村 恕彦, 和田 英穂, 西村 和夫, 高田 正之 訳) : 詳解 Fortran90, bit 別冊, 共立出版 (1993).

[8] SSL II 拡張機能使用手引書 (科学用サブルーチンライブラリ), 99SP–4070-2, 富士通株式会社 (1991).

[9] OS IV/MSP FORTRAN77 EX 使用手引書 V12 用, 79SP-5031，富士通株式会社 (1991).

[10] 呉 智英 : 言葉につける薬, 双葉社 (1994).

[11] 島崎 眞昭 : スーパーコンピュータとプログラミング, 計算機科学／ソフトウェア技術講座 9, 共立出版 (1989).

[12] OS IV/MSP FORTRAN77 EX/VP 使用手引書 V12 用, 79SP-5041，富士通株式会社 (1991).

[13] 伊理 正夫, 藤野 和建 : 数値計算の常識, 共立出版 (1985).

[14] 島崎 眞昭 : 数値計算におけるベンチマーク, 情報処理, Vol.31, No.3, pp.313–320 (1990).

[15] Gregory, R. T. and Karney, D. L. : *A collection of matrices for testing computational algorithms*, John Wiley & Sons, New York (1969).

[16] 金田 康正 : π のはなし, 東京書籍 (1991).

[17] 小松 勇作, 梶原 壤二 : 詳解 関数論演習, 共立出版 (1983).

[18] 岩波数学公式 II, 岩波書店 (1957).

[19] 二宮 市三 : π と e の代わりに 2 を使おう, 名古屋大学大型計算機センターニュース, Vol.24, No.1, 34–42 (1993).

1981 年頃のセンター端末室の風景