2003－HPC－95　(1)
2003／8／4

CG法の最短計算時間を探索する試み

An experiment to seek the shortest computational time for CG methods with multiple precision floating-point arithmetic

幸谷智紀
Tomonori Kouya
静岡理工科大学
Shizuoka Institute of Science and Technology, Fukuroi, Shizuoka 437-8555 Japan
tkouya@cs.sist.ac.jp

概要

CG 法は

- 内積計算と行列・ベクトル積の計算が多く，並列化が容易である
- 桁数の少ない浮動小数点数を用いると，丸め誤差の影響が強く現れる

という特性を持っている。従って，並列分散化した多倍長計算を行えば，反復回数も計算時間も低減させることができる筈である。本稿では比較的良条件の例題に対して多倍長計算を用いた CG 法を実行して最短計算時間を求め，IEEE754 倍精度での最短計算時間との比率がどの程度になるかを探求した結果について報告する。

1　初めに

n 次正定値実対称行列 A を係数行列として持つ連立一次方程式

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

の解 $\mathbf{x} \in \mathbb{R}^n$ を求める数値解法として Conjugate-Gradient 法 (以下 CG 法と略記) がある。そのアルゴリズムを以下に示す。

1. 初期値 $\mathbf{x}_0$ を決める。
2. $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$，$\mathbf{p}_0 := \mathbf{r}_0$ とする。
3. $k = 0, 1, 2, \cdots$ に対して以下を計算する。
   (a) $\alpha_k := \dfrac{(\mathbf{r}_k, \mathbf{p}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)}$
   (b) $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
   (c) $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k A\mathbf{p}_k$ (又は $:= \mathbf{b} - A\mathbf{x}_{k+1}$)
   (d) $\beta_k := \dfrac{\|\mathbf{r}_{k+1}\|_2^2}{\|\mathbf{r}_k\|_2^2}$
   (e) 収束判定
   (f) $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

CG 法の特性として，(a) 内積計算と行列・ベクトル積の計算が多く並列化が容易であり，計算時間を減らすことができることが挙げられる。しかし反面，理論的には n 回の反復で収束する筈であるが，(b) 桁数の少ない浮動小数点数を用いると丸め誤差の影響が強く現れ，収束を遅くしたり，収束しない場合があることも知られている。

従って，(b) の欠点を補うために多倍長計算を行い，更に (a) の特徴を生かしてその並列化を行えば，反復回数も計算時間も低減させることができる筈である。もちろん，ソフトウェアで多倍長計算を行うことで，演算ごとの計算時間は増大するからおのずと限界はある。では，近年定着しつつある並列分散環境である PC cluster 上において，多倍長計算を用いた CG 法はどの程度まで高速化できるのだろうか？

本稿では，特定の問題にターゲットを絞り，mpich[6] と GMP およびその上で構築した多倍長数値計算ライブラリ BNCpack[1] を使用したベンチマークテストを行うことで，一定の指標を与えることを目的とする。

2　例題の収束特性

本稿で用いる例題を次のように設定する。係数行列 A と解 $\mathbf{x}$ は

$$A = \begin{bmatrix} n & n-1 & \cdots & 1 \\ n-1 & n-1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ n-1 \end{bmatrix} \tag{2}$$

とする。定数ベクトル $\mathbf{b}$ は $\mathbf{b} := A\mathbf{x}$ として与える。

次元数を 64, 128, 256, 512 とし，それぞれに対して CG 法を適用し，IEEE754 倍精度 (double)，128bits, 256bits, 512bits, 1024bits でそれぞれ計算し，その数値特性を見ることにする。この計算は BNCpack に実装した CG 法を用い，多倍長計算には GMP の MPFR パッケージの関数群を用いて実行した。

残差 $||\mathbf{r}_k||_2 = ||\mathbf{b} - A\mathbf{x}_k||_2$ をプロットすると Fig.1 のようになる。

通常は IEEE754 倍精度での数値実験結果のみを使用することが多いが，こうして多倍長計算も行ってみると，如何に CG 法が丸め誤差の影響に敏感であるかが明確となる。512bit 計算でも僅かだが残差のノルムが上昇しており，完全に単調収束させるにはそれ以上の精度が必要である。同時に，精度を増やしてもある限度以上には反復回数を低減させる事はできない。従って，最小の計算時間は，多倍長計算の計算時間と反復回数の最適化を図ったところで得られることになる。

3　BNCpack とその並列分散化

BNCpack は，Basic Numerical Computation PACKage の略称である。この名から分かる通り数値計算の標準的なテキストに書いてある程度のアルゴリズムを実行できる環境を作りたい，というのが本来の目的であった。GMP が Version 2 の時にそれを採用して多倍長化し，原型となる基本線型計算の関数群を作り上げた。幸い GMP の上位互換性に助けられ，大幅な仕様変更なしで現在の GMP でも利用可能である。

GMP が Version 3 までは ANSI C のインターフェースしか提供していなかったこともあって，使用言語は全て C(not C++) とした。

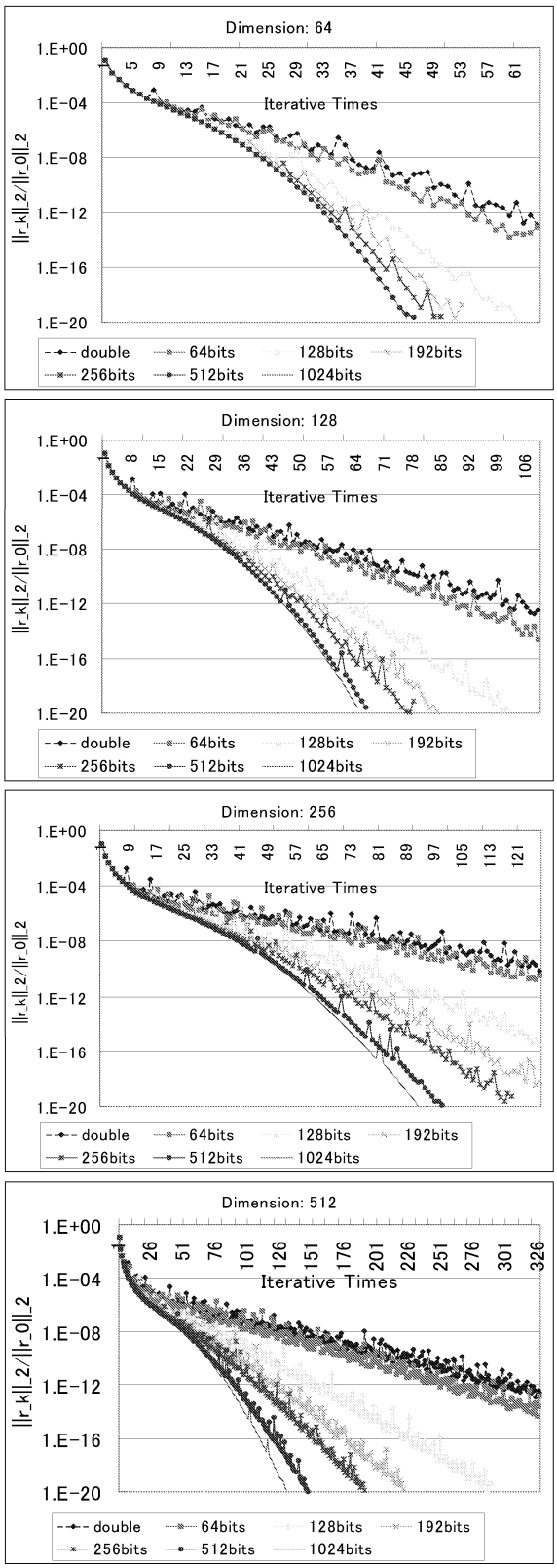

Fig. 1: Effect of round-off errors in CG methods

現在提供されているのは次の関数群である。

1. 初等関数
2. 複素数演算
3. 基本線型計算 (ベクトル，正方密行列の和・差・内積 (積)・スカラー倍など)
4. 連立一次方程式 (直接法，反復法，CG 法)
5. 行列の固有値・固有ベクトル計算 (現状ではべき乗法・逆べき乗法のみ)
6. 非線型方程式 (Newton 法，準 Newton 法, Regula-Falsi 法)
7. 代数方程式 (DKA 法)
8. 数値積分 (台形則，Romberg 積分)

なるべく関数の引数を簡素にするため，よく使用すると思われるデータ型は構造体を用いて typedef してある。現在の所，複素数 (MPFCmplx), 実係数多項式 (MPFPoly), 配列 (MPFArray, CMPFArray), スタック (MPFStack), 実ベクトル (MPFVector), 実密行列 (MPFMatrix) のデータ型が利用出来る。これらは全て，デフォルトの桁数とは別に設定が可能である。例えば実行列型は

```
typedef struct{
    unsigned long int prec;
    mpf_t *element;
    long int row_dim, col_dim;
} mpfmatrix;

typedef mpfmatrix *MPFMatrix;
```

と宣言してある。このうち prec は行列要素全体の精度 (bit 数) を保持している。よって，BNCpack で独自に宣言した変数においても，GMP の mpf_t 型と同様，一つのプログラム中に異なる桁数のデータが混在できるようになっている。

BNCpack を使ったプログラムの例については，Web ページ [1] で配布しているソースコード群を参照されたい。また行列の積を計算するプログラムの実行時間 (積を計算する部分のみ) を Mathematica と比較した結果は [8] を参照されたい。当然だが，Mathematica よりも高速な処理が可能となっている。

3.1 並列分散化の試み

多倍長計算は IEEE754 浮動小数点計算と比べて多くの計算時間を要する。そのため，高速化を図る必要があるが，1CPU 上での逐次処理には限界がある。GMP は Knuth[5] が論述してるアルゴリズムを忠実に実装したものであるが，これらの劇的な改良は現時点では困難と思われる。そこで数値計算全体を高速化するために並列分散化を行うことを考えた。現在，一般的かつ安価な並列分散処理を行うハードウェア構成として，複数の PC を Ethernet(10/100/1000BASE) で接続した PC cluster が普及している。よって，この PC cluster 上で高速化を図ることにした。

問題は多倍長浮動小数点数をどのように並列分散処理に使用するかである。数値計算に必要とされる桁数はそれほど多くなく，IEEE754 倍精度の 2 倍の精度，4 倍の精度, 8 倍, 16 倍 ··· といった程度で十分だという反応をあちこちで聞いた。また，PC cluster で用いる Ethernet では，1 フレームのサイズ制限が 1500bytes となっている。これに収まる 2 進浮動小数点数の有効桁数は 10 進数換算で 3000 桁を超えるものとなる。これだけの桁数を必要とするケースはかなり少ないと予想される。更に，通信量は少なく押さえる必要があり，1 フレームに収まるサイズの浮動小数点数は分割せずに処理した方が良いと考えられる。こうすれば実装も容易になる。

以上の観点から，実装する並列分散プログラムでの多倍長浮動小数点数は分割せず，GMP の mpf_t 型もしくは mpft_t 型をそのまま使用し，各 PE(Processor Element) 間のやりとりの際にもこれらのデータ型をそのままバッファに pack して使うようにした。この流れを Fig.2 に示す。例えば PE0 から mpfr_t 型のデータを送信し，PE1 でそれを受信する場合は，PE0 は送信前にデータを void 型ポインタで指定されたバッファにパック (pack_mpf 関数を使用) し，PE1 は受信したバッファからデータをアンパック (unpack_mpf 関数を使用) する。この一連の操作を行うための関数群は MPIGMP Library[7] として公開済みなので，詳細はそちらを参照されたい。

開発環境は，以下に示す PC cluster(9PEs) とソフトウェアを用いている。以降示すベンチマークテストも全て以下の環境下で行っている。

ハードウェア

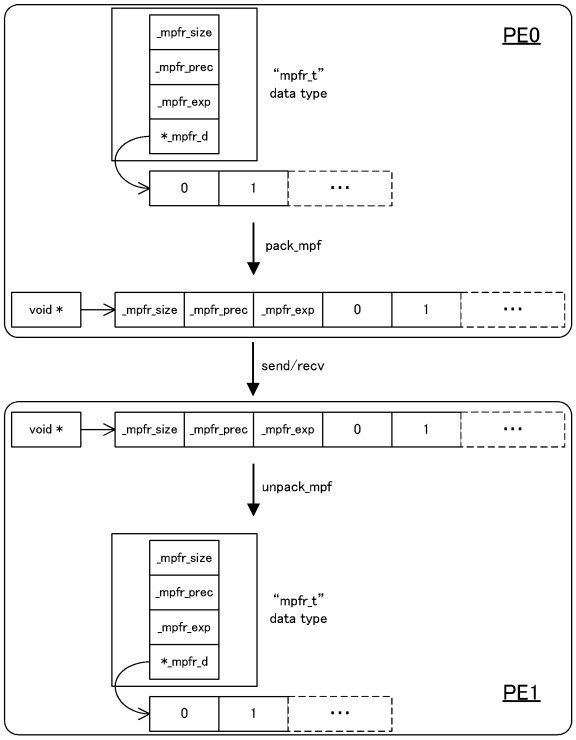

Fig. 2: Sending and Receiving one mpfr_t datatype

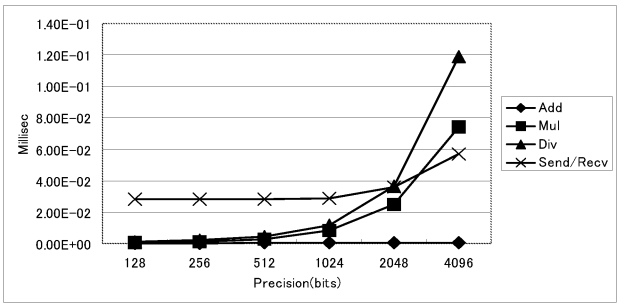

Fig. 3: Performance of Arithmetics and Send/Recv

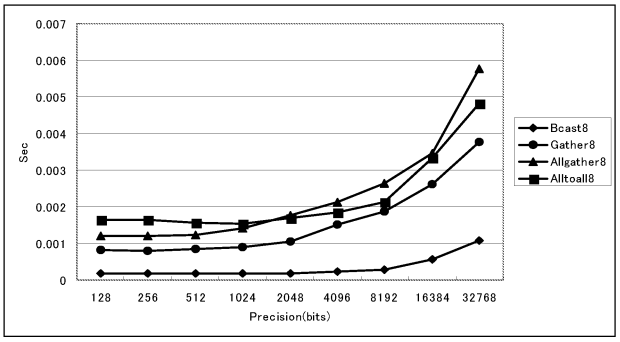

Fig. 4: Collective communications

CPU Intel Pentium III 1GHz/Celeron 1GHz 合計 9 台

Ethernet 100BASE-TX
+ 24port Switch(NFS/mpich 共用)

ソフトウェア

OS Vine Linux 2.6r1

MPI mpich 1.2.5-1 (ch_p4, rsh 使用)

C compiler gcc 3.2.2

GMP GMP 1.4.2 (mpfr_t 型を使用)

前述した多倍長計算の実装方針に基づき，以上の PC cluster 環境において，四則演算と基本通信 (MPI_Send/Recv)，及び集団通信 (MPI_Bcast/Gather/Allgather/Alltoall) のベンチマークテストを行った結果を Fig.3，Fig.4 にそれぞれ示す。集団通信は全て 8PE を用い，通信する際に必要となる memcpy の処理にかかる時間も含め，MPI_Wtime 関数を用いて時間計測を行った。

この結果，実際の数値計算アルゴリズムにおいても良好な結果が得られると判断した。

CG 法の並列化に際しては行列 A，定数ベクトル $\mathbf{b}$ 及び作業用のベクトルを各 PE 毎に列ブロック単位で分割して割り当て，内積計算は MPI_Allreduce を，行列とベクトルの積は MPI_Allgather を使って実装している。Fig.5 に示す通り，並列化の効果が大きいことが分かる。

4 ベンチマークテスト

以上の並列分散処理環境下において，ベンチマークテストを行った結果を以下の表に示す。なお，1CPU での数値実験とは異なり，ベクトルと行列を分割して扱っているために，反復回数では，1CPU でのそれと数回程度のずれが生じている。IEEE754 倍精度 (double) での計算時間 (秒数) の最小値と，多倍長計算での計算時間の最小値にそれぞれ下線を引いておく。

Dimension: 64

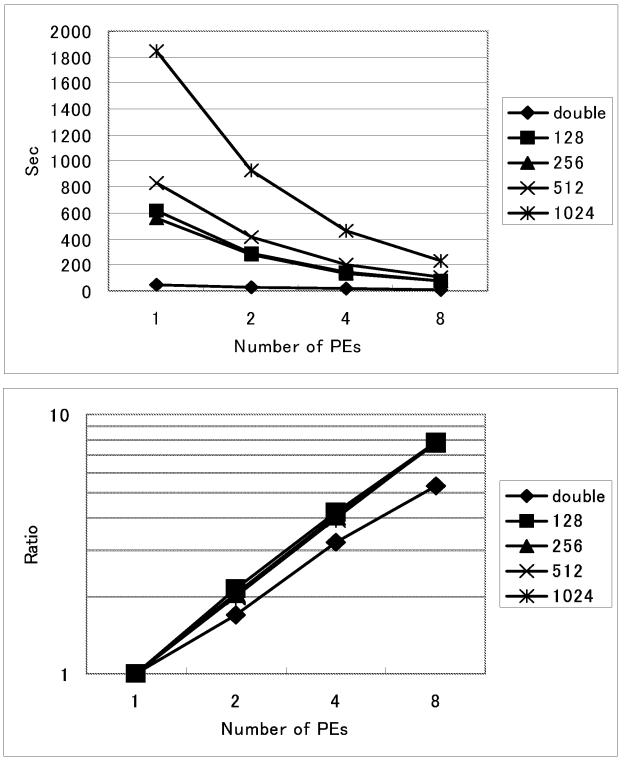

Fig. 5: Computational time and parallelization ratio of CGs

	double	64	128	192	256	512
1PE	0.02	0.178	0.164	0.197	0.225	0.456
2PE	0.036	0.112	0.101	0.117	0.133	0.253
4PE	0.051	0.085	0.073	0.081	0.091	0.154
8PE	0.219	0.138	0.127	0.129	0.138	0.291

Dimension: 128

bits	double	64	128	192	256	512
1PE	0.110	1.125	0.995	1.103	1.223	2.381
2PE	0.083	0.621	0.536	0.593	0.655	1.240
4PE	0.089	0.352	0.306	0.337	0.371	0.675
8PE	0.382	0.393	0.315	0.332	0.334	0.510

Dimension: 256

bits	double	64	128	192	256	512
1PE	1.05	6.90	5.54	6.00	6.59	12.46
2PE	0.48	3.66	2.90	3.10	3.39	6.34
4PE	0.31	1.97	1.51	1.64	1.78	3.27
8PE	0.53	1.36	1.06	1.10	1.17	1.94

Dimension: 512

bits	double	64	128	192	256	512
1PE	7.27	44.50	32.90	33.66	36.54	66.18
2PE	2.28	22.08	16.83	17.06	18.58	
4PE	1.56	11.29	8.64	8.72	9.46	
8PE	1.46	6.52	4.89	4.92		

このうち，256 次元と 512 次元の場合は，実験で使用した PC cluster の PE 数が足りず，並列化の効果を最大限利用した結果とはなっていないことが分かる。特に後者では倍精度・多倍長計算の両方で更に計算時間を低減できる可能性が高い。

これをまとめて，各次元数における倍精度と多倍長計算での計算時間の最小値，およびその比率 (多倍長/倍精度) を求めると

次元数	64	128	256	512
倍精度 (PE 数)	0.02(1)	0.083(2)	0.31(4)	1.46(4)
多倍長 (bit 数, PE 数)	0.073 (128,4)	0.306 (128,4)	1.06 (128,8)	4.89 (128,8)
比率	**3.67**	**3.71**	**3.44**	**3.35**

となる。これによって，例題 (2) の場合は，すべての次元数において，多倍長計算を用いても倍精度の 3 倍強の計算時間で済むことが分かる。

5 今後の課題

今回は比較的良条件の一例題に対して，極めて小規模の PC cluster 上で実験した結果を示すに留まった。しかし，内積及びベクトル・行列積で使用する集団通信に要する時間を計測することで，もっと大きな PE 数を用いた場合での実行時間の予測はある程度可能であると思われるので，今後はこの辺を改善したい。

参考文献

[1] BNCpack, http://member.nifty.ne.jp/tkouya/na/bnc/

[2] GNU MP, http://swox.com/gmp/

[3] The MPFR Library, http://www.mpfr.org/

[4] 今井仁司,「偏微分方程式の無限精度数値シミュレーションに関する研究」科研費補助金報告書, 2001.

[5] D.E.Knuth, The Art of Computer Programming
Vol.2, 1981.

[6] mpich,
http://www-unix.mcs.anl.gov/mpi/mpich/

[7] MPIGMP Library,
http://member.nifty.ne.jp/tkouya/na/bnc/mpigmp.tar.gz

[8] 幸谷智紀, 多倍長計算についての考察 —GNU MP
と BNCpack—, Linux Conference 2002, 2002.