PC Cluster 上における多倍長数値計算ライブラリ BNCpack の並列分散化 —MPIBNCpack について—

幸谷智紀
静岡理工科大学
tkouya@cs.sist.ac.jp

1 初めに

現在の数値計算は，PC や WS の CPU が直接処理することの出来る IEEE754 standard の定める浮動小数点数を用いることが主流である。多倍長計算はソフトウェアで実装する他なく，IEEE754 倍精度の計算であれば一つの機械語命令で済む処理でも，多数の命令を積み重ねて実現する必要が出てくる上，より多くのメモリ領域を確保しなければならない。一般に多倍長計算はより多くの計算時間を必要とする。

しかし，多倍長計算が実行可能な環境は増えており，一昔前のように全ての基本演算を自分で実装する必要はなくなってきている。PC の能力は格段に上がり，計算時間とメモリの問題も，満足出来るレベルに落ち着きつつある。このような状況下において，我々は今のところ最高に近い性能を有している GNU MP[6](以下，GMP と略記) を利用した数値計算ライブラリ BNCpack[2] を作成し公開した。この BNCpack を使ったプログラムの例については，Web ページ [2] 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，昨年の LC2002 の講演資料 [12] を参照されたい。

本稿では，BNCpack のさらなる高速化を図ったライブラリである，MPIBNCpack[23] について述べる。これは 1CPU 用に構築された BNCpack，そして並列分散処理に適した API である MPI の実装系である mpich[21] を用いて，並列化された数値積分，CG(Conjugate-Gradient) 法，DKA(Durand-Kerner-Aberth) 法をそれぞれ実装し，ライブラリとしてまとめたものである。ベンチマークテストを行ってこれらの性能評価を行い，その有効性を明らかにする。

2 多倍長数値計算の現状

可変長の整数型，浮動小数点数型をサポートしたライブラリ，もしくはソフトウェアを用いた計算を総称して，多倍長 (multiple precision, multiprecision) 計算と呼ぶ。科学技術シミュレーションには浮動小数点数が多用されるため，可変長の浮動小数点数のみを用いた計算をそのように呼ぶこともある。任意精度 (arbitrary precision) 計算という名称も用いられる。本稿では浮動小数点数のみを扱うことにする。

現在，多倍長計算を実行するためには，大きく分けて二つの方法がある。一つは，統合環境型の数式処理ソフトウェア (Mathematica[30], Maple[15], MuPAD[24]) もしくは数値計算ソフトウェ

ア (Matlab[17]) を使うことである。これらの有償の商品であるが，使いやすい UI を備え，グラフィックス機能や記号処理に優れており，洗練された研究用ツールとして人気がある。もう一つの方法は，C/C++/Fortran/Java などのプログラミング言語から多倍長計算ライブラリを使う方法である。前者に対して高速な演算が可能であるが，プログラミングのスキルが必要となり，使いこなすには習熟が必要である。多くは open source であり，自由に使用できる。現在では後述する GMP を利用したもの (CLN[3], MPFR[19]) や，Fortran で開発された MP をベースに発展してきたもの (MPFUN/ARPREC[1])，その他独自の実装 (FMLIB[5], MAPM[16], SN Library[26], mppack[20], Exflib[4]) がある。どちらにしても，演算そのものはソフトウェアで実装されており，固定長の IEEE754 倍精度浮動小数点数を用いた演算よりも速度は非常に遅くなる。

そのため，多倍長計算を多用した大規模な数値計算を実行するには，なるべく高速なライブラリを使用した上で，それらを並列に実行できることが望ましい。現在では，今井 [10, 11] や渡部 [28] らが FMLIB を用いて，長谷川 [7] が GMP を組み込んだ Fortran コンパイラを使用して，大規模な数値計算を行った結果を公表している。

3 GMP について

GMP は 1991 年から開発が進められてきた ANSI C とアセンブラで記述された多倍長計算ライブラリで，2003 年 5 月現在の最新バージョンは 4.1.2 である。最初は整数型 (mpz_t) と有理数型 (mpq_t) しか存在しなかったが，Version 2 になって 2 進浮動小数点型 (mpf_t) が追加された。Version 4 では，試験的ではあるが以前より要望の強かった C++クラスインターフェースが追加された。また，GMP とは別の Project で開発の進められてきた MPFR パッケージ [19] も含まれるようになっている。これは，GMP の mpf_t 型をベースにした mpfr_t 型を用い，IEEE754 standard と互換性を持つように改良された 2 進浮動小数点ライブラリである。丸めモードの変更が可能で，高速な初等関数も提供されている。これら mpf_t，mpfr_t 型は，変数ごとに仮数部の精度を指定できる上，精度の異なる変数同士の演算も可能である。多倍長計算よって，同じ計算を異なる精度の変数で行って精度を比較するといった使い方が出来る。現在の BNCpack はこれら浮動小数点数機能の上に成り立っている。図 1 にこれら二つのデータ型の構成を示す。なおこれは現行の Ver.4.1.2 に基づくもので，将来変更される可能性がある。

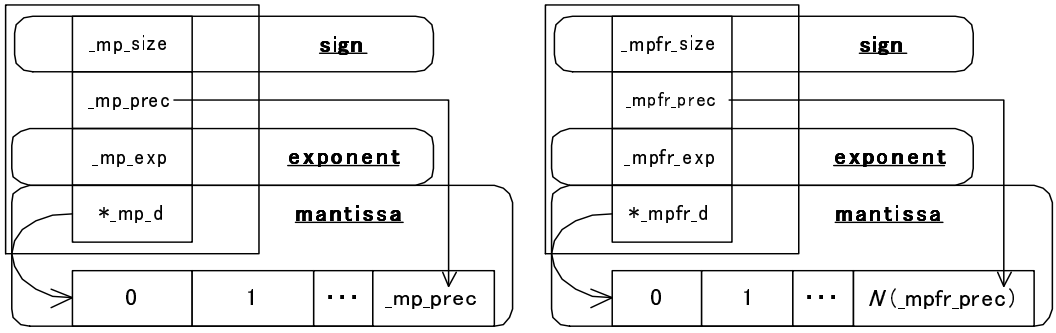

図 1: mpf_t 型と mpfr_t 型

どちらも拡張された符号部 (sign)・指数部 (exponent)・仮数部 (mantissa) を持った構造体として定義されている。精度の増減は，構造体内部のポインタ (_mp_d/_mpfr_d) が指定しているアドレス

位置に存在する仮数部の長さを調整することで行う。

BNCpack を作成するにあたり，この GMP を採用したのは，無償のオープンソースソフトウェアであることと，他のライブラリや数式処理ソフトウェアよりも高速な処理が可能である，という二つの理由からである。性能については MPFR Project の Web サイト [19] に提示されているので詳細はそちらを参照されたい。また，MPFR の実行例については，本家 Project[19] からリンクされている，著者が作成した Web ページ [27] をアクセスし，実際に数式を打ち込んで実行して得て頂きたい。

4 並列分散化の試み

多倍長計算はソフトウェアで実装されるため，CPU 内の演算ユニットで直接実行される 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[9] 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。また，分散処理用のライブラリには，国際的に広く認知されている MPI を実装した mpich[21] を使用することにした。

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。この流れを図 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[22] 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。なお，この環境の構築方法については著者の文書 [13] を参照されたい。

ハードウェア

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

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

ソフトウェア

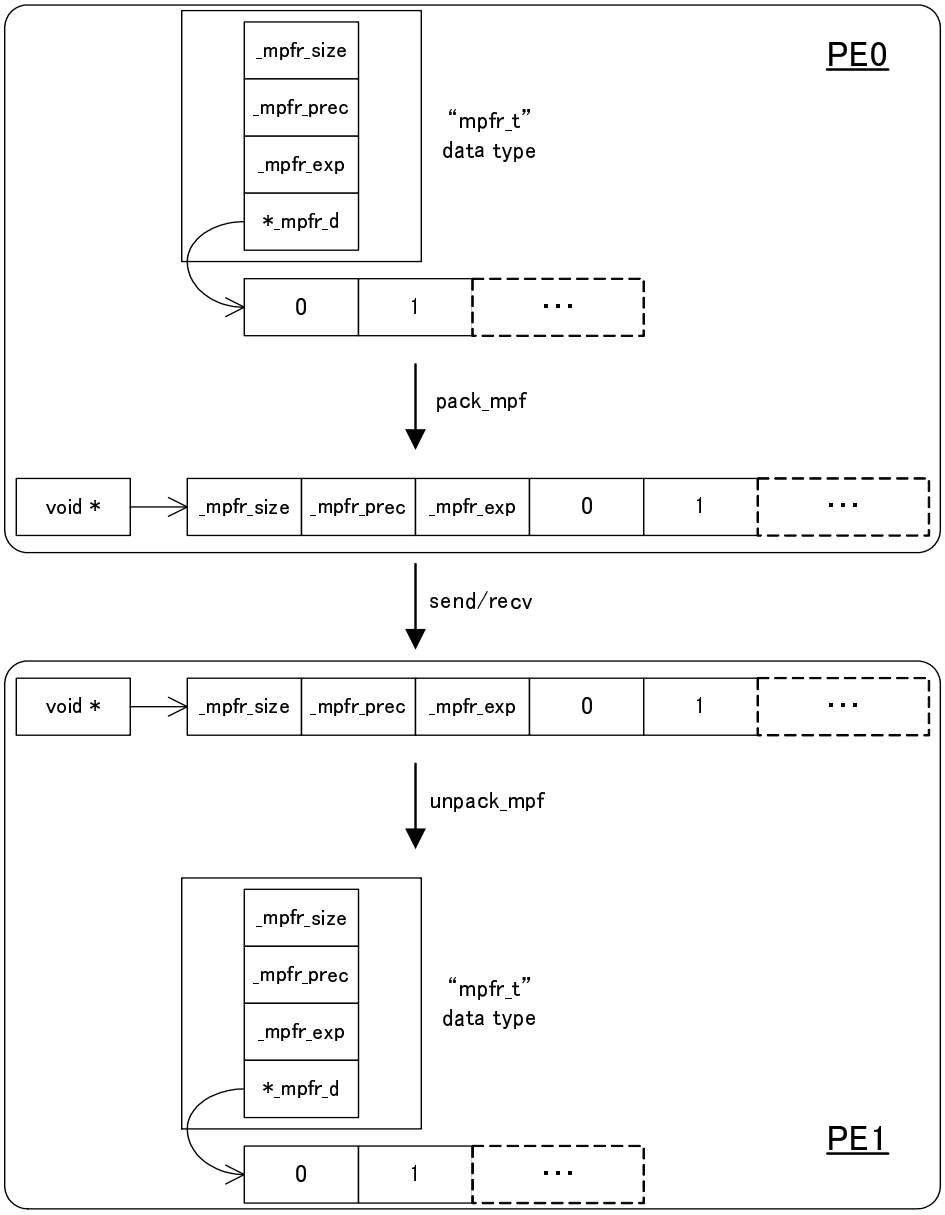

図 2: mpfr_t 型データの送受信方法

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図 3，図 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。

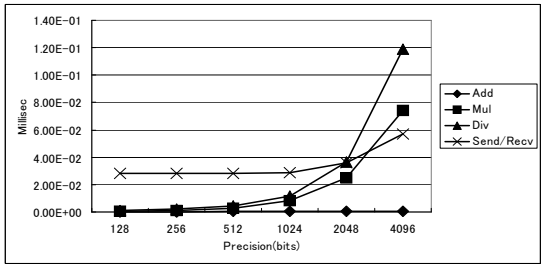

図 3: 四則演算 (和・積・商) と送受信のパフォーマンス

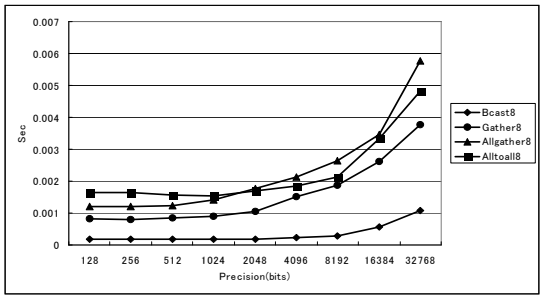

図 4: 集団通信のパフォーマンス

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，並列化の効果を確認しやすく，実装が容易な，数値積分，CG 法，DKA 法の 3 つのアルゴリズムを実装し，MPIBNCpack としてまとめ，公開した。このライブラリの構造を図 5 に示す。

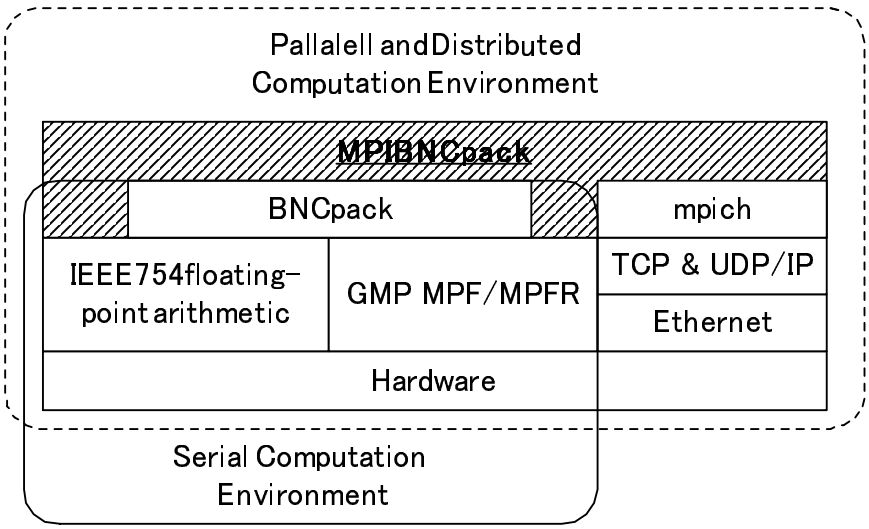

図 5: MPIBNCpack の構造

以下，実装したアルゴリズムのベンチマークテストの結果を示す。

4.1 数値積分

数値積分は MPI のテキストにも必ず取り上げられ，mpich にもサンプルプログラムとして cpi.c が付属している。これは中点則を用いたものであるが，mpfr_t 型を使って書き換えたものを実装し，ベンチマークテストを行った。但し，分割数は 16384 で固定し，桁数のみ変更して計算時間を計測してある。使用した数値積分は cpi.c で使用されている

$$\int_0^1 \frac{4}{t^2+1} dt = \pi \tag{1}$$

である。前述したように，これは MPI_Reduce を使用した実例となっている (図 6)。

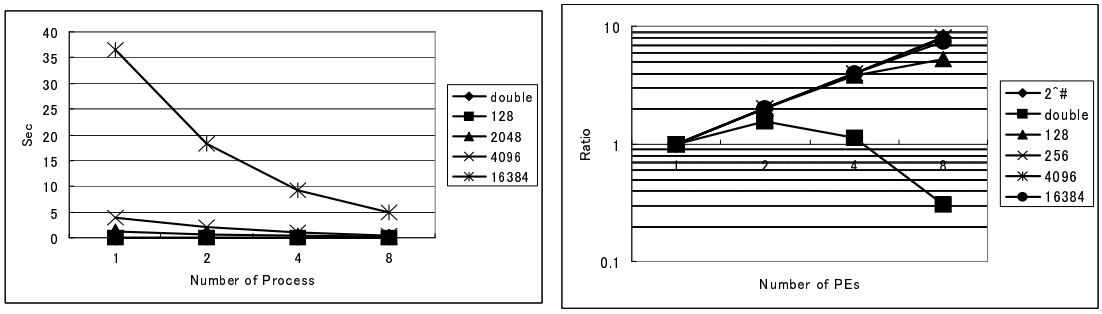

図 6: 数値積分の計算時間と並列化の効果

被積分関数に乗算，除算という，桁数の増大と共に急激に計算時間を要する演算を含んでいるため，桁数が多い時ほど並列分散処理の効果が大きくなっている。

この数値実験では分割数が固定であるため，精度に比して打切り誤差が相対的に大きすぎて，多倍長計算の意味が薄いものとなっている。一般に供される数値計算ライブラリとして考えるなら，不必要な桁数を確保しないような警告を発する機構が必要となろう。

4.2 CG 法

連立一次方程式の係数行列 A と解 **x** を

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

とする。定数ベクトル **b** は **b** := A**x** として与える。100 次元 (n=100) の時，この問題に対して CG 法を適用し，IEEE754 単精度 (float), 倍精度 (double)，128bits, 256bits, 512bits, 1024bits でそれぞれ計算し，残差 $\|\mathbf{r}_k\|_2 = \|\mathbf{b} - A\mathbf{x}_k\|_2$ をプロットすると図 7 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ことがわかる。

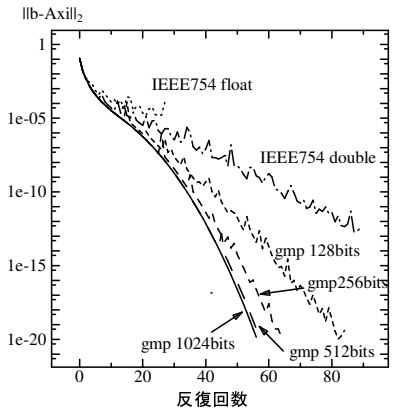

図 7: CG 法における精度変化の影響

前述の連立一次方程式 (2) を 1024 次元にし，CG 法を用いて計算する。CG 法に限らず，ベクトル同士の演算，行列・ベクトル演算が多用される Krylov 部分空間法は並列化による効果が大きいことが知られている。また，計算精度に敏感であり，精度を増やすことで反復回数が削減できることも図 7 に示した通りであり，多倍長計算の効果を示すには格好の例である。

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 (図 8)。

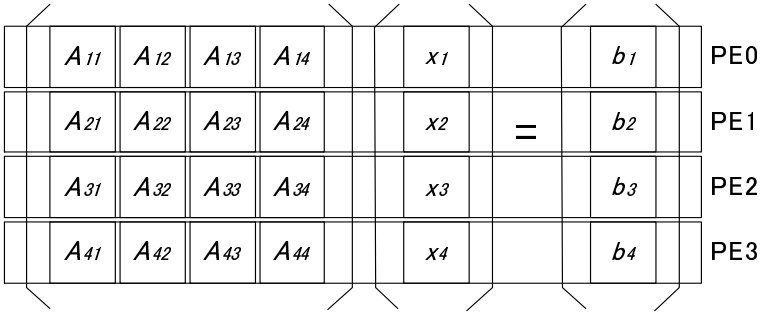

図 8: 行列とベクトルの分割方法

図 9 に示す通り，並列化の効果が大きいことが分かる。

4.3 DKA 法

数値計算の標準的なテキストでは，必ず「桁落ち」についての説明がある。これは，お互いに誤差を含んだ近接している浮動小数点数の加減算の結果，絶対値が極端に小さくなるとその分だけ相対誤差が増大する，という現象である。この桁落ちが計算途中に断続して発生し，計算結果の相対誤差がマシンイプシロンよりも極めて大きくなる問題を，悪条件 (ill-conditioned) と呼ぶ。例えば n 次元の連立一次方程式 $A\mathbf{x} = \mathbf{b}$ の係数行列 A の条件数 $\mathrm{cond}(A) = \|A\|\|A^{-1}\|$ が大きい時は，それに

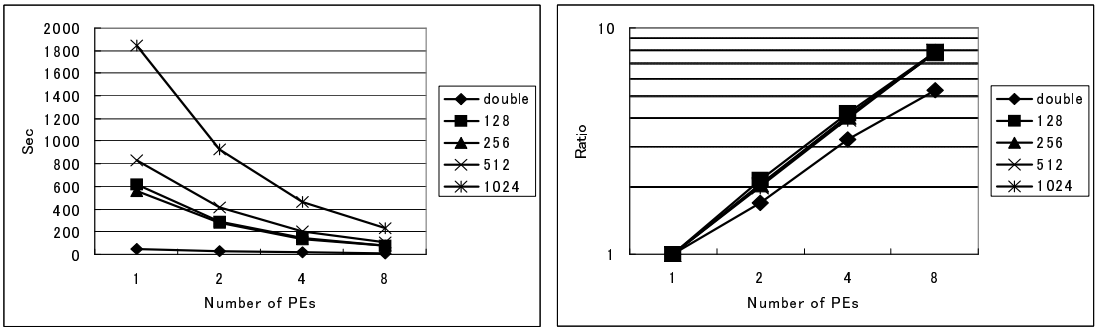

図 9: CG 法の計算時間と並列化の効果

比例して数値解 **x** に含まれる相対誤差が増大することが知られている。また，近接根を持つ代数方程式は悪条件な問題と言える。例えば Wilkinson[29] が例示した

$$\prod_{i=1}^{20}(x-i)=0 \tag{3}$$

がその例である。

DKA 法は実係数 n 次代数方程式

$$\sum_{i=0}^{n} a_i x^i = 0 \ (a_n \neq 0) \tag{4}$$

を，解と係数の関係を用いて n 次元 n 変数の非線型方程式に変換し，複素数演算を用いて各根の近似値 $z_l^{(k)}$ $(l = 1, 2, ..., n)$ を求める反復解法である。ここでは次の Jacobi の 2 次法を用いている。

$$z_l^{(k+1)} := z_l^{(k)} - \frac{\sum_{i=0}^{n} a_i (z_l^{(k)})^i}{a_n \prod_{j=1, j\neq l}^{n} (z_l^{(k)} - z_j^{(k)})} \tag{5}$$

古くからこの解法は並列計算向きであると言われているが，各 PE に反復式を分割して割り当てる方式では，反復一回毎に全ての近似解をやり取りする (MPI_Allgather に相当する通信) 必要が出てくるため，次数の低い場合は通信量の観点からはあまり効果が出ないのではないかと予想される。実際に，先の代数方程式 (3) を解いた結果を図 10 に示す。

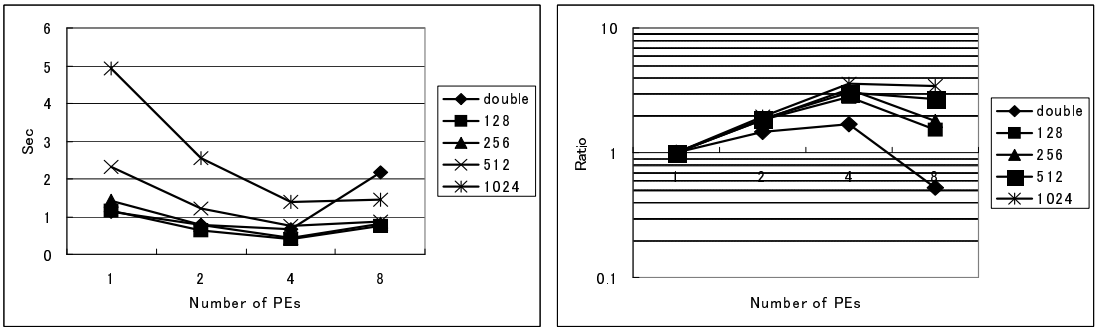

図 10: DKA 法の計算時間と並列化の効果

予想通り，4PEs 以上ではあまり並列化の効果が出ていないことが分かる。割り当て方式を見直すか，通信量を押さえるような反復式を考案する必要があろう。

5　今後の課題

以上述べてきたように，並列版多倍長数値計算ライブラリ MPIBNCpack は様々なオープンソースソフトウェアに支えられて構築されており，基本的な数値計算アルゴリズムが実装され，高速な処理が可能となっている。今後も他のソフトウェアの支えになるように更なる機能強化を行う必要がある。具体的には

- 関数の種類を増やしつつ，1CPU での処理を出来うる限り洗練させる
- ヘテロな環境にも適用できるような実装を行う

といったことが挙げられる。その上で，シームレスに超多倍長計算にも対応できる機能を付加していきたい。

参考文献

[1] Multiprecision Software Directory, http://crd.lbl.gov/~dhbailey/mpdist/

[2] BNCpack, http://na-inet.jp/na/bnc/

[3] Class Library for Numbers, http://www.ginac.de/CLN/

[4] Exflib, http://sumire.acs.i.kyoto-u.ac.jp/~fujiwara/exflib/

[5] FMLIB, http://myweb.lmu.edu/dmsmith/FMLIB.html

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

[7] 長谷川秀彦, クリロフ部分空間法の計算精度依存性, 日本応用数理学会講演予稿集, 2003.

[8] N.J.Higham, Accuracy and Stability of Numerical Algorithms(2nd ed.), SIAM, 2002.

[9] D.E.Knuth/中川圭介・訳,「準数値算法/算術演算」, サイエンス社,1986.

[10] 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.

[11] 今井仁司, 応用解析における多倍長計算, 数学 第 55 巻, 第 3 号, 2003.

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

[13] 幸谷智紀, Vine Linux による PC Cluster の構築,
http://na-inet.jp/na/mpipc.pdf

[14] LAPACK, http://www.netlib.org/lapack/

[15] Maple soft, http://www.maplesoft.com/

[16] MAPM, http://www.tc.umn.edu/ ringx004/mapm-main.html

[17] MathWorks, http://www.mathworks.com/

[18] MPFUN, http://www.netlib.org/mpfun/ tml

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

[20] MPPACK, http://phase.hpcc.jp/phase/mppack/

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

[22] MPIGMP Library, http://na-inet.jp/na/bnc/mpigmp.tar.gz

[23] MPIBNCpack, http://na-inet.jp/na/bnc/mpibnc.tar.gz

[24] MuPAD, http://www.mupad.de/

[25] M.L.Overton, Numerical Computation with IEEE754 floating point aritmetic, SIAM, 2001.

[26] SN Library, http://hp.vector.co.jp/authors/VA018507/

[27] Try MPFR!, http://www.jpsearch.net/try_mpfr.html

[28] 渡部善隆, 多倍長計算の最近の結果, 日本応用数理学会講演予稿集, 2003.

[29] J.H.Wilkinson, Rounding Errors in Algebraic Processes, Dover, 1991.

[30] Wolfram Research, http://www.wolfram.com/