PC Cluster の Usability 及び Performance 改善に関する研究報告

A Report on Studies related to Usability and Performance Improvement of PC clusters

幸谷智紀*, 越須賀正雄 *, 澤木 淳 *, 鈴木文明 *

Tomonori KOUYA*, Masao KOISUKA*, Jun SAWAKI* and Fumiaki SUZUKI*

Abstract: In this paper, we report our three studies in 2004 related to usability and performance improvement of PC clusters. Firstly, we describe our development of a Web interface using Perl CGI scripts, by which users of PC clusters obtain better code-writing and execution environment for their serial or parallel mutiple precision numerical computing. Secondly, the performance of BNCpack transplanted to Microsoft Windows operating system and the results of computation time estimation for product-type Krylov subspace methods are shown. Lastly, we report the studies on the performance improvement of our PC clusters by introducing Microsoft Windows to them.

1. 初めに

Microprocessor の発明以来，PC 向け CPU は一本調子で動作周波数を上げてきたが，1GHz に到達するその前後から急速にそのスピードを落としつつある．観測筋によれば，動作周波数を上げることはそれほど難しいことではないが，コンシューマ向けの PC では CPU が発する熱を処理するためのコストが問題になるとのことである．これだけが理由ではないにしろ，結果，Intel は予定していた 4GHz CPU の出荷を停止している [10]．

周波数を向上させる代わりに，性能アップを図るため登場したのが，AMD64, EM64T といった，現行の 32bit 環境との互換性を重視した 64bit 化技術であり，複数の CPU Core を一つのダイにまとめた Multi-Core 技術である．前者は一度に処理できるデータ長を増やして発行される命令を減らすことを目的とし，後者は一つのプログラムを実行した際に並列動作が可能な Multi-thread 化部分を同時に，そして Hyper-threading 技術のように一つしかない CPU 内のパイプラインの使用をめぐってぶつかる危険性なしで複数 Thread を実行させることを目的としている．どちらも動作周波数の向上が見込めなくても性能が向上するよう，並列性を高める技術であるが，ソフトウェア側でこれらの技術に対応した書き換えを行う必要がある．

具体的には，次のような対応が必要となる．64bit 化技術においては，今まで 32bit 長でのデータアクセスを基本としていたのを，64bit 長のアクセスを行うようにしなければならない．これはコンパイラ側の対応だけでかなりの部分が自動化できるが，場合によってはソースコードの修正が必要になることも考えられる．Multi-Core 技術においては，一つのプログラム内における処理を複数 Thread に分割し，なるべく Thread 同士が同期を取らずに済むように修正しなければ，Multi-Core CPU の性能を生かせないことになる．

従って，64bit 化にしろ Multi-Core にしろ，これらを搭載した CPU を使う限りは，パフォーマンスを向上させるのはソフトウェア側の並列化対応次第ということになる．2005 年はその端緒を開く年となる筈であり，そのためには今までに培われたソフトウェア資産や技法を纏め上げて次へのステップへの準備を終えておく必要がある．

我々の PC cluster(cs-pccluster[14], cs-pccluster2[15]) は複数のコモディティ PC をネットワークで接続し，ソフトウェアを組み込んで一つの並列マシンのように扱うことが出来るようにした手製のものである．それ故に，あまり使い勝手が良いものとは言えず，パフォーマンスも十分に発揮できていたとは言い難い．将来的にはこれらは全て 64bit Multi-Core の PC に置き換わることになるが，使いづらく，パフォーマンスが十分でないものになっては意味がない．従って，現状の PC cluster 環境を可能な限り良いものにするソフトウェアのサポートとベンチマークテストによる比較検討が不可欠となる．特に後者は将来における性能評価の指標となるため，データを蓄積しておくことが重要となる．

本稿では，本年度に実施された現状の PC cluster における使い勝手 (Usability) の向上と，パフォーマンス計測を行った結果について述べる．これらは卒業研究 (越須賀，澤木，鈴木) 及び指導教員 (幸谷) の研究活動として行われたものである．

最初に，使い勝手向上のための CGI インターフェース開発について述べる．通常，PC cluster で実行される科学技術計算プログラムはコンパイルされて実行ファイル形式に変換され，コマンドラインで実行されることが多い．しかし，現在の PC 環境は殆どが GUI であり，Windows におけるコマンドプロンプトの存在すら知らないユーザが増えた．そのため GUI で簡単に操作できる統合環境が望まれることになるが，単なる standalone なものでは管理上，複数ユーザへの対応が面倒なことになる．そこで PC cluster の 1 node を Web サーバとし，そこにログイン・簡易エディタ・実行結果出力部を併せ持った画面を表示する CGI を搭載することで，ネットワークを介して自在に並列計算が実行できる環境を提供することにした．

次に，Windows 環境における多倍長計算の性能比較と，それを利用した Krylov 部分空間法の計算時間予測を行った結果を述べる．前述したように，現行の PC 環境は殆どが GUI であり，その大半は Windows ファミリ (9x/ME/200x/XP) である．当大学に入学してくる学生は殆どが小中高校のどこかで PC 基礎教育を受けてきているか，自宅で PC を自在に使いこなしている世代であるが，彼ら/彼女らの多くは “Linux”という名前を聞いたことはあっても，UNIX という OS の存在は知らないのではないか．逆に，PC cluster は今も UNIX で構築されることが多く，CUI に不慣れなユーザに対しては，目的の科学技術計算を行う前に，UNIX の CUI 実習を受講してもらう必要が出てくる．従って，Windows を用いて PC cluster を

2005 年 2 月 28 日 受理

*理工学部 情報システム学科

構築できれば，それだけで使い勝手の向上になる．ここではWindows に移植された MPFR[2] のパフォーマンス計測と，それを用いた Windows 版 BNCpack に実装された積型 Krylov 部分空間法計算時間予測を行う．

最後に，実際に Windows を用いた PC cluster を構築し，その性能評価を行った結果を示す．通信速度の計測には Windows に移植した NetPIPE[9] を用い，並列計算のベンチマークには前述の Windows 版 BNCpack と，同じく Windows に移植した MPIBNCpack を用いる．この結果，少なくとも Vine Linux を用いた PC cluster よりも，Windows を用いた PC cluster の方が，通信のパフォーマンスは良く，並列化した並列多倍長数値積分や Krylov 部分空間法の計算速度の向上にも寄与することが確認された．

2. CGI を用いた PC Cluster 用 Web インターフェースの開発

情報システム学科 3 年生対象のネットワークシステム実験において，我々は「多倍長数値計算プログラミング」をテーマとして選択し，2004 年 4 月から 7 月までの 3ヶ月間，5 グループ 65 名に対して実施した．実習に用いたコンピュータ環境は cs-pccluster2[15] である．このクラスタの計算 node1 台につき 1 人を割り当て (10 台しかないので，1 グループを 2 分割して実施)，Vine Linux 2.6r3(途中から r4 へ update した) の GNOME GUI を用いてプログラミングを行い，ベンチマークと計算結果を取得する，というものである．

プログラミング能力は個人差があり，基本的な関数の使い方だけを提示して問題要求に沿うものをスクラッチから作成する，という方法を取っていては時間内に全ての学生の実習を終えることが出来ない．そのため，テキストエディタで打ち込んで，コマンドラインから GCC コンパイラを起動すればすぐに実行結果が得られるよう，ソースコードは予め提示しておくようにした．その分，レポート作成のためのデータ整理を Excel を使って行うようにし，アプリケーションスキルの向上に繋がるよう心掛けた．

結果，予想通り，プログラミングに関してはそれほど手間取らず，予定の実験時間を大幅に超過しても作業を終了できない学生は現れなかった．しかし，次の 2 点ではかなりの時間が取られる結果となった．

- UNIX のコマンドラインの操作 (GCC によるコンパイル，ファイルへのリダイレクト操作)
- ソースプログラムの打ち込みミスによるエラーの修正

前者に関しては，基本的なコンピュータリテラシ能力はある学生対象の実験であったため，慣れるまでは大変そうな様子が見えたが，慣れてしまえば一度実行したコマンドを何度も間違えるようなことはなかった．しかし後者については前者に比して多くの時間を費やした．これは，単純に学生のプログラミング能力の欠如に原因を求めることは出来ないものである．

今までの当学科におけるプログラミング教育は主として Microsoft Visual C++を使ってきたが，昨年からコマンドラインで使用するフリーのコンパイラに切り替えられた．しかし，世間では Windows におけるプログラム開発の多くは前者の統合開発環境 (IDE, Fig.1) を用いて行われるのが普通のようである．

この統合環境は，PC が非力であった頃は動作が快適であったとは言い難く，熟練者は自分の好みのテキストエディタとコマンドラインコンパイラを併用することが普通であった．今

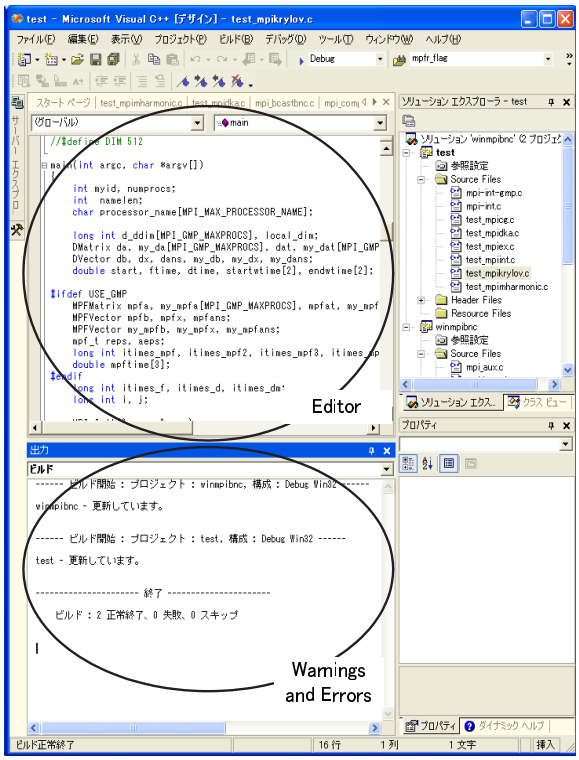

Fig. 1: 統合開発環境の例 (Visual Studio .NET)

でも UNIX でのプログラミングの多くはこのような形態で行われる．我々の実験はこのスタイルを踏襲している．

しかし，今回後述するように BNCpack/MPIBNCpack を Windows に移植するにあたって，久しぶりに Visual Studio .NET の IDE を使用したところ，エディタ機能の進化には正直言って驚かされた．関数の引数の順序を忘れてしまった時にはバルーンヘルプが手助けしてくれるし，単純なスペルミスは自動的に修正してくれる上，長いソースコードを中カッコ単位で表示・非表示できる機能もある等，構文エディタ機能の充実には目を見張るものがある．IDE から呼び出せる MSDN の膨大なドキュメント群は，Win32 API を使ったプログラミングでは必須である．一度，このような支援機能を体験してしまうと，シンプルなテキストエディタを用いたソースコード編集が苦痛になるのも仕方がない．少々動作が重くても，Windows 環境においては IDE の方を選択する人が多いのも納得できることである．

そのような現状で，十年一昔の開発環境を強いることについての是非は，さまざまな意見があろうが，少なくとも IDE に慣れた学生にとっては戸惑いの多いものであったことは容易に想像がつく．関数名を打ち間違える，と言った単純ミスも，リンクエラーと表示されるだけでどの行数で間違っているかを教えてくれないのである．プログラミングスキルの未熟な学生に対して，完全ではないにしろ，IDE に似た開発環境を UNIX においても提供することは，学習意欲を低減させないためにも考える必要がある．

2.1 CGI の構成

今回，我々は Web Browser を用いて簡易な擬似的統合環境を作成することにした．ネットワーク的には閉じた環境で実施される学生実験で使用すること，PC cluster を用いた並列計

算にも活用できるものにしたい，という点を考慮して，セキュリティについては甘くして，ごく簡単な認証機能だけを備えるようにした．また，我々は既に外部に公開するための多倍長計算用 CGI[3] も作成しており，そのノウハウも生かした．

CGI による Web インターフェースを提供することで，3 つの利点を持たせることが出来る．

まず，CGI を Perl script で作成することにより，CGI を実行できる Web サーバさえ動作すれば，Windows であれ，Linux であれ，多少の修正を加えることで，どの環境でも我々の CGI を実行できる，ということが挙げられる．

二つ目として，ユーザ側に Web Browser 以外の特別なソフトウェアが不要になる，ということが挙げられる．しかも Web Browser の操作は若年者ほど習熟しており，実行する際に特別な実習を施す必要はない．Web を介しているため，LAN 内であればどこからでも実行できるという強みもある．

三つ目として，Web 及び Cluster サーバにユーザ毎のシステムアカウントを追加する必要がないことが挙げられる．これは管理負担の軽減に繋がると共に，セキュリティについても好ましいものである．CGI の実行は明示的に変更されない限り，Web サーバ (Apache の場合は httpd デーモン) のユーザ権限で行われる．また，PC cluster 上で SPMD(Single-Program Multiple-Data) プログラムを使って MPI による並列計算をさせる場合は，全ての node マシンにシステムアカウントが必要となる．通常これは NIS(Network Information System) で集中管理されるものであるが，多人数のユーザに複数 node 上で自在に並列計算をさせるようなアカウントはなるべく発行したくない．よって PC cluster で並列計算が出来るシステムアカウントと，不特定多数が実行する CGI のアカウントは共用しないのが普通である．我々の CGI は，この原則を維持した上で，sudo コマンドによって両者の橋渡しをさせるようにしている．つまり，CGI がプログラムを実行する際には，並列計算可能なシステムアカウントに sudo して実行が行われることになる．

このような考えの元に，作成した CGI の構成を Fig.2 に示す．

ユーザが行う作業は 3 つ (ログイン，ソースコードのアップロード，ソースコードのコンパイルと実行) あるため，それぞれの作業毎に，それを請け負う CGI を作成した．これら 3 つの作業の流れを大まかに示すと次のようになる．

ログイン

(1) ユーザが ID とパスワード (pass) を Web サーバへ送る．

(2) login.cgi は，受け取った ID と，パスワードを crypt 関数 (Perl) に渡し，id.txt の結果と一致することを確認する．それができれば，edit.cgi を起動して Edit フォームを出力する．

ソースコードのアップロード

[1] Edit フォームにソースコードを書き込む，もしくは既存のファイルを upload する．

[2] Edit フォームに書き込まれたソースコードは edit.cgi に渡される．

[3] edit.cgi はソースコードを “ID.c”という名前で Web サーバのローカルディスクに保存する．

ソースコードのコンパイルと実行

{1} Edit フォーム内の実行ボタンを押す．

{2} exec.cgi はローカルディスクにある “ID.c”をコンパイルして実行ファイル “ID.exe”(UNIX の場合は拡張子なし) を生成する．エラーの場合はエラー結果のみを出力する.「多倍長利用」のチェックがあれば，適切なマクロを定義し，GMP, MPFR もリンクする．

{3} Cluster サーバは生成された実行ファイルを，Edit フォームに指定された PE 数 (1〜8PEs) で並列実行する．

{4} 標準出力及びエラー出力結果を exec.cgi に渡す．

{5} 結果を Web 画面 (一番下のフレーム) に表示する．

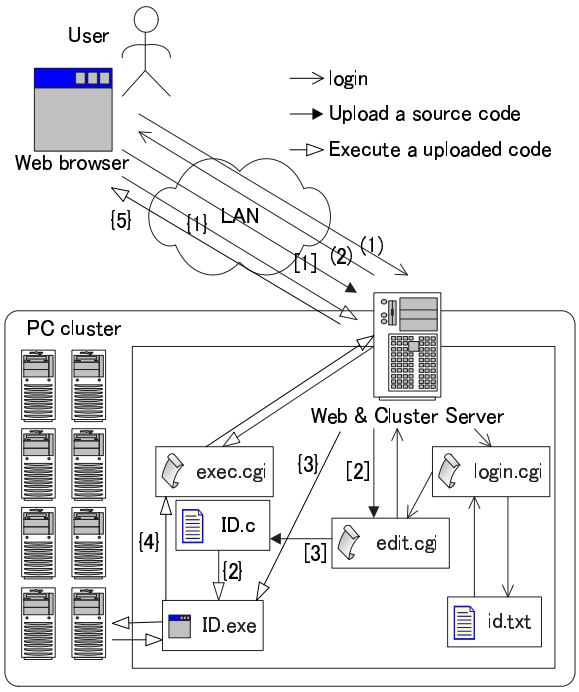

Fig. 2: PC クラスタ向け CGI の構成

実際にこの CGI を動作させ，8PE で並列計算した結果を Fig.3 に示す．一番上からログインのためのフレーム，Edit のためのフレーム，エラー・実行結果出力のためのフレームとなっており，先に示した Visual Studio .NET の IDE に似た機能配置になっている．ユーザは upload したソースコードを Edit フォームで自在に編集することが出来，エラーの結果も得ることが出来る．

しかし，まだ実際に学生に提供して実験を行っていないため，この CGI が真に初心者向けのものとなっているかどうかは検証できていない．また，PC cluster 向けの機能も不十分である．複数ユーザが複数プログラムの実行をすることも想定しておらず，バッチ処理機能や，タスクの kill 機能なども必要になるが，これらはまだ実装されていない．これらの実現は今後の課題として残っている．

3. Windows 環境における多倍長計算の性能比較と計算時間予測

我々は GMP[1] 及びその基本関数を利用した MPFR[2] をベースに，多倍長数値計算ライブラリ BNCpack[5] を開発してきた．GMP/MPFR は長らく UNIX 環境でのみ利用されてきたが，今年 (2004 年) になって Visual C++.NET と NASM を用いて，GMP 4.1.3 を Windows 環境に移植するためのソースキットが

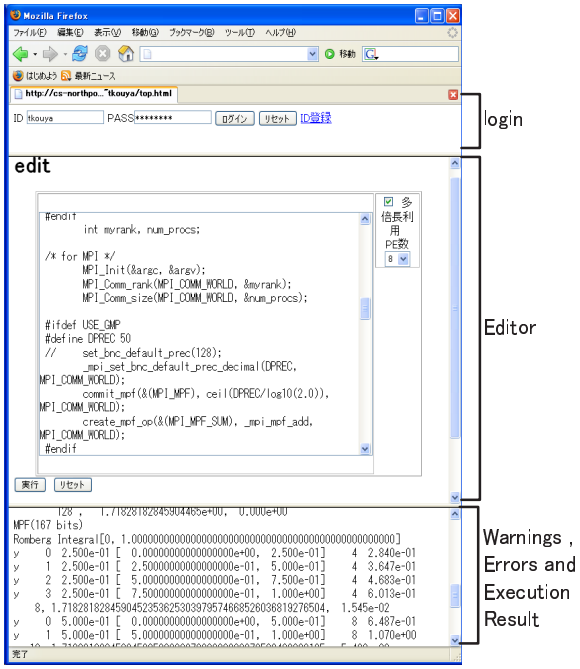

Fig. 3: 並列計算実行時の Web browser 画面

B.Gladman によって作成され公開された [4]．我々はこれに基づいて最新の GMP 4.1.4 を Windows に移植し*，それを利用して Windows 版 BNCpack を作成した．本節では，Windows 版の MPFR ライブラリの性能評価と，Windows 版 BNCpack を用いて Krylov 部分空間法の計算時間予測を行った結果について述べる．

3.1 Windows 版 MPFR 基本演算関数の性能評価

前述したように，GMP 及び MPFR は UNIX 版がオリジナルである．特別なオプションを必要としなければ，両者のインストールは./configure, make, make install と 3 つのコマンドで終了する．このうち最初の./configure 時にには，インストールされる環境に最も適したコードやマクロが選択され，結果，生成されたライブラリは最高のパフォーマンスが得られる仕組みになっている．

最近の Intel の CPU は，基本的には i80386 と同じ命令が実行可能だが，SIMD(Single-Instruction Multiple-Data) 命令が追加されつつある．これらは上位互換性はあるものの，下位互換性はない．従って，例えば MMX 命令は MMX Pentium 以降の CPU では実行可能だが，SSE(internet Streaming SIMD Extension) 命令は Pemtium III より前の CPU では実行できない，ということになる．

GMP の整数・有理数・浮動小数点数及び MPFR の関数は，どちらも GMP の自然数演算関数群 (mpn_*) をベースとして構築されたものである．この自然数演算関数においては，高速化のために CPU に依存する SIMD 命令を利用している．GMP と MPFR の高速性はこの部分の高速性によって維持されているものである．Gladman の移植作業の大半はこの CPU に依存した部分の NASM 対応作業に費やされており，Visual C++と NASM によってその部分は Pentium/Pentium II 用 (mpn_p0.lib), Pentium III 用 (mpn_p3.lib), Pentium IV 用 (mpn_p4.lib) にそれぞれ別ライブラリとして生成されるようになっている．

しかし，移植はなされたものの，その性能についての詳しい報告は今のところ見つけられない．そこで，我々は Pentium IV 2.8GHz マシンに, Windows 2000 Pro SP4 を導入し，更に比較対照のために同じマシンの Windows に UNIX コマンドライン環境を Cygwin[11] を用いて構築し，移植版の MPFR と Cygwin にインストールされたオリジナルの MPFR における，乗算 ($\sqrt{5}$ $\sqrt{3}$ の計算を繰り返して計測) と除算 ($\sqrt{5}$/ $\sqrt{3}$ の計算を繰り返して計測) のベンチマークテストを行ってみた．その結果を Fig.4 に示す．

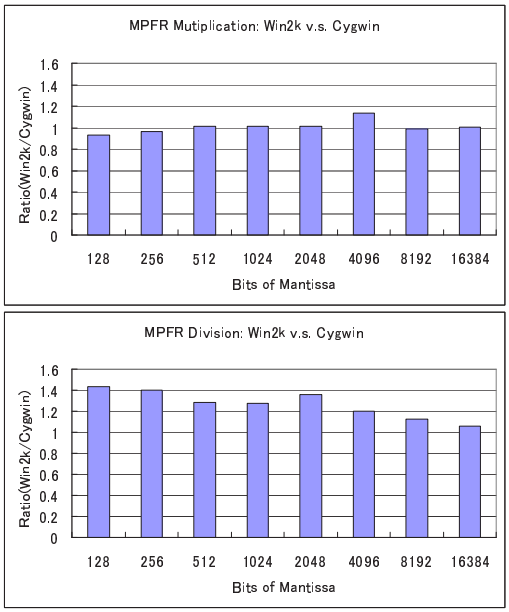

Fig. 4: Windows 版 MPFR の多倍長乗算 (上) と除算 (下) の性能比

乗算のパフォーマンスはオリジナルとほぼ変わりなく，除算については短い桁数では最大 1.4 倍悪化しているものの，桁数を長くするにつれてオリジナルとの差が縮まってくることが判明した．時間計測に用いた関数のソースは付録 1 に示した通り，Windows でも UNIX でも同じ関数が使用可能であるようにしてある．このベンチマークテストにおける時間差はこの関数の実装の差による可能性も排除できないが，後述するように Windows, Linux クラスタにおけるベンチマークテストの結果より (これも同じ関数を使って時間計測を行っている), 問題視するほど精度差があるとは思われない．

これによって，我々の BNCpack/MPIBNCpack を Windows 環境でも構築でき，Linux 環境とそれ程劣らない多倍長計算が可能になると予想される．

3.2 積型 Krylov 部分空間法の計算時間予測

前述した環境において，連立一次方程式

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

を解くための積型 Krylov 部分空間法 [16, 25] の反復一回あたりの時間予測を行う．これらのアルゴリズムは付録 2 にまとめて示した通り，全ての計算が行列，ベクトル単位で実行されるもので，大規模な連立一次方程式になれば並列計算の効果が発揮しやすいという利点がある一方，丸め誤差に対して極

*オリジナルのソースキットには MPFR ライブラリ作成時にリンクミスがあり，修正が必要である．

めて敏感で，計算桁数が少ないと収束しないことも多い．そのため，多倍長計算が必要なアルゴリズムであると言える．

ここで取り上げる 4 つのアルゴリズムの計算回数を Table 1 に示す．**(1)**〜**(4)** はそれぞれ

(1) ベクトル和
(2) ベクトルのスカラー倍
(3) 内積
(4) 行列・ベクトル積

である．この中で GPBiCG 法が他のアルゴリズムに比べて頭抜けて反復一回あたりの計算回数が多いことが分かる．しかし，低い精度でも安定して収束するという性質があり，反復回数は少なく済むことが知られている．従って，それぞれのアルゴリズムのトータルの計算時間については問題，特に行列 A の性質によって異なってくる．

Table 1: 積型 Krylov 部分空間法の反復一回あたりの演算回数

アルゴリズム	**(1)**	**(2)**	**(3)**	**(4)**
BiCG 法	2	5	5	2
CGS 法	6	5	2	2
BiCGSTAB 法	5	6	4	2
GPBiCG 法	15	13	7	2

これらのアルゴリズムの計算時間予測は，素朴に行うことにした．すなわち，**(1)**〜**(4)** の計算時間を予め別のプログラムを実行して求めておき，その結果に Table 1 の計算回数を乗じて予測値とする．本稿で縷々述べてきたように，近年の CPU は様々な高速化技法が使用され複雑化の一途を辿っており，動作周波数だけで計算時間を推し量るのは難しい．Single Core の CPU を搭載した単一 PC においても，Cache 容量，メモリ帯域，SIMD 命令，動的スケジューリングのサポート等の相違があれば計算時間は全く変わってくる．よって，計算時間の予測は，このような素朴かつ簡単な方法の方が精度は高いと予想される．この予想を以降でベンチマークテストの結果と比較して確認する．

3.3 時間予測結果

我々は，最初に **(1)**〜**(4)** の計算を二つの n 次元実ベクトル $\mathbf{c}, \mathbf{d}$ と行列 E を

$$
\begin{aligned}
\mathbf{c} &= \sqrt{5}[1\ 2\ \cdots\ n]^T \\
\mathbf{d} &= \sqrt{3}[n\ n-1\ \cdots\ 1]^T \\
E &= \sqrt{5}\begin{bmatrix} 1 & 2 & \cdots & n \\ 2 & 3 & \cdots & n+1 \\ \vdots & \vdots & & \vdots \\ n & n+1 & \cdots & 2n-1 \end{bmatrix}
\end{aligned}
$$

と定義し，これを使ってそれぞれ

(1) $\mathbf{c} + \mathbf{d}$
(2) $\sqrt{3}\mathbf{c}$
(3) $(\mathbf{c}, \mathbf{d})$
(4) $E\mathbf{d}$

という計算を行って予測に使用する基本演算時間を求めた．

時間予測を行った連立一次方程式 $A\mathbf{x} = \mathbf{b}$ は

$$
\begin{aligned}
A &= \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1/2 & 1/3 & \cdots & 1/(n+1) \\ \vdots & \vdots & & \vdots \\ 1/n & 1/(n+1) & \cdots & 1/(2n-1) \end{bmatrix} \\
\mathbf{x} &= [1\ 2\ \cdots\ n]^T \\
\mathbf{b} &= A\mathbf{x}
\end{aligned}
$$

である．この行列 A は Lotkin 行列と呼ばれる悪条件な非対称行列である．これは密行列であるが，成分の所々に 2 のマイナスべき乗，及びそれらの組み合わせで表現できるものがあるため，この行列とベクトルとの積は，**(4)** よりも幾分高速に実行される可能性がある．GMP, MPFR の浮動小数点演算関数は，前述したように自然数関数をベースに構築されおり，仮数部の下位にゼロが並んでいる場合はその分計算が軽くなるからである．

予測結果を Fig.5 に示す．これはそれぞれの桁数指定における，64, 128, 256 次元の問題における，反復一回分の予測時間と実際の計算時間との相対誤差を各アルゴリズム毎にグラフ化したものである．

全体として，時間予測の精度は全て 1 未満に収まっており，それなりに良い予測になっていることが分かる．しかし，予測とのズレが大きい所もある．

例えば，IEEE754 倍精度 (double) の結果は，特に 256 次元ではあまり良くない．また，10000 桁の多倍長計算を行った場合についても同様のことが言える．我々の予測手法では，PC 内部におけるメモリ転送のオーバーヘッドを無視しており，後者についてはそれの影響が高いものと思われるが，前者についてはデータ長がごく短いことを考えると，別の理由が影響しているのであろう．どちらにしろ，もっと次元数，桁数を小さい幅で変動させて，細かくベンチマークテストを実施する必要がある．その上で，この結果と後で述べる NetPIPE ベンチマークの結果を用いて並列 Krylov 部分空間法の時間予測を行うことが今後の課題として挙げられよう．

4. Windows MPI Cluster の構築及び性能評価

Windows が本格的に 32bit CPU に対応した Windows 95 の発売の前後から，Linux がそのコストパフォーマンスの良さを武器に台頭してきたため，両者はよくライバルとして比較される．一頃は後者の良さを強調するあまりに，Microsoft 社によって独占的に販売されている前者を貶すという向きも多かったように思われる．

しかし，Windows 2000 からは 16bit legacy 部分をほぼ脱ぎ捨て，Windows XP になってパフォーマンスの改善が行われ，Linux も Kernel は 2.6.x 系統が主流になって来ている．これだけ改善の歴史が長い両 OS に対しての評価はそれ程簡単なことではない．総合的には様々な部分の性能を評価する必要があろうし，個別のユーザにとっては自分が実施したい作業にふさわしいベンチマークテストを実施して判断すべきであろう．

我々は前述したように BNCpack を作成し，それをベースとして MPI を用いて並列化した MPIBNCpack を開発し利用している．従って，これらを高速に動作できる環境が望ましいことになる．これを調べるため，まず演算性能とネットワーク性能を調べることにする．

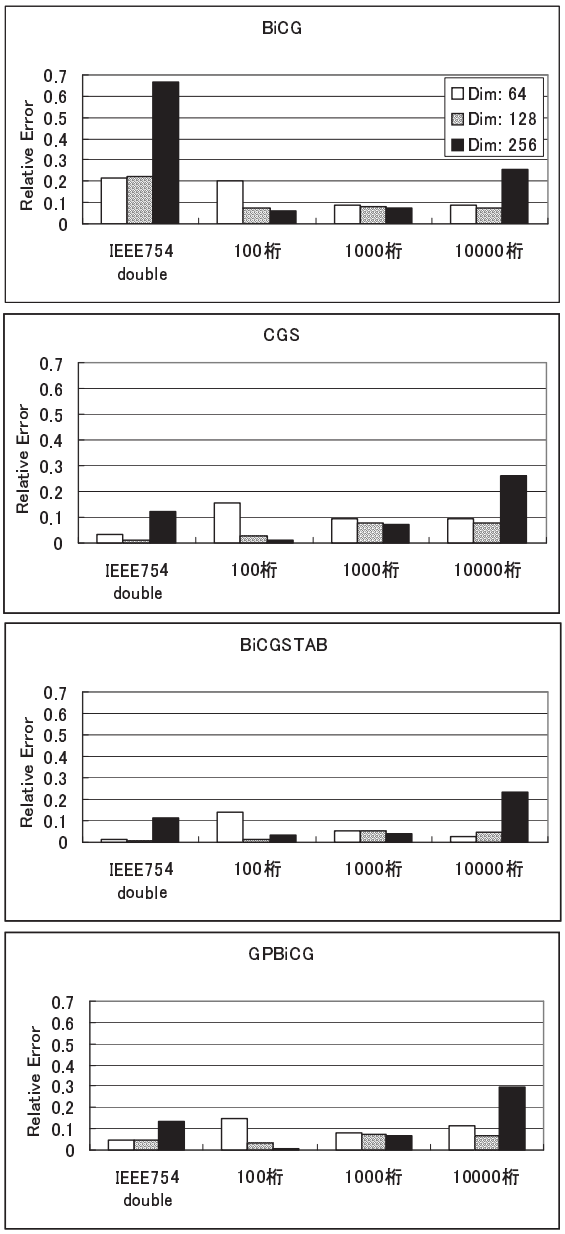

Fig. 5: Krylov 部分空間法の反復一回あたりの時間予測精度

MPFR の演算性能は Linux において最高の性能を発揮する．では Windows ではどうだろうか．先に示した Windows 版 MPFR の乗算・除算ベンチマークテストを Windows 2000 SP4(Win2k) と Windows XP SP1(WinXP) で実行してみることにする．同じ Intel 865PE chipset を搭載した同じメーカ (Gigabyte 社) の Mother Board を使った Pentium IV 2.8cGHz のマシンで実行した結果を Fig.6 に示す.

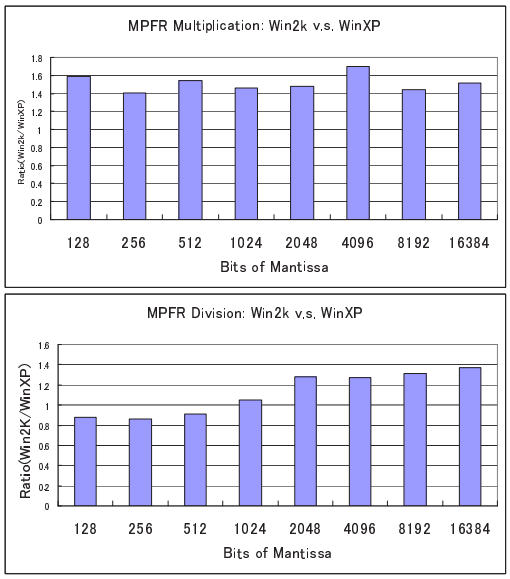

Fig. 6: Windows 2000 と XP における MPFR 乗算 (上), 除算 (下) 性能比

同じハードウェア上で実行したにも関わらず，乗算においては全体的に 1.4 倍の速度差があり，Windows XP の方が高速に実行できている．除算においても，短い桁数ではそれ程差がなく，むしろ Windows 2000 の方が良いのに対し，桁数を長くすると乗算と同様，1.4 倍近くまで XP の方が高速になる．これは同じ Visual Studio .NET と GMP 4.1.4 のバイナリを用いてコンパイルしたベンチマークテストプログラムの結果であるから，結論としては Windows XP においては少なくともメモリアクセスのパフォーマンスに関しては何らかの改善が図られている，と言える．

参考までに，Windows XP SP1 を用いた 1000BASE Ethernet(GbE) のネットワーク性能を Fig.7 に示しておく．これは我々が Windows に移植した NetPIPE[9] を用いて TCP 性能を計測したものである．cs-pccluster (Pentium III 1GHz/Celeron 1GHz) には，他社のチップを搭載した NIC (Network Interface Card) と比較した結果，最も性能の良かった Intel のチップを搭載した PCI NIC を使用し，cs-pccluster2 では Mother Board に搭載されている CSA 接続されたコネクタを用いている．同じ GbE とは言え，CPU/メモリ・I/O 帯域の違いによって，同じメーカのチップを用いていてもここまで差がつくのである．

Linux の方のネットワーク性能に関してはあまり良い結果を得られていない [12]．我々は昨年，Vine Linux を用いて構築した cs-pccluster2 の GbE 性能が，100BASE に比して 5〜7 倍程度に留まっていることを指摘した [24]．この理由は Linux のデバイスドライバにあるのか，Linux の構造そのものに起因するものなのかは現時点では不明である．今回，cs-pccluster2 に Windows を導入して，NetPIPE で TCP 性能を比較してみ

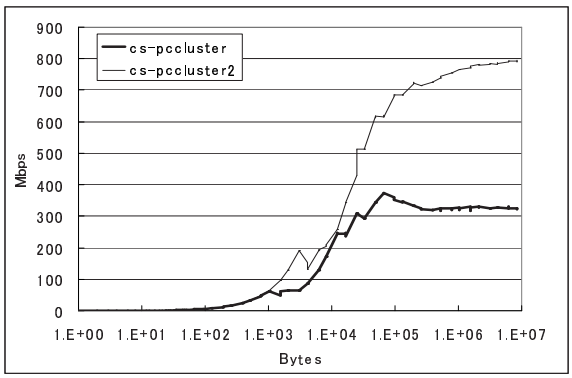

Fig. 7: cs-pccluster と cs-pccluster2 における TCP の通信性能

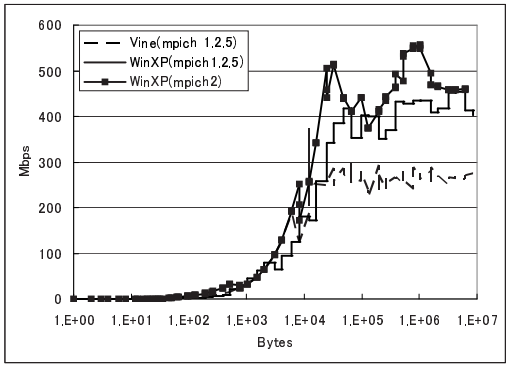

Fig. 9: Linux と Windows における MPICH の通信性能

たところ，最高性能では 200Mbps もの差がついていることが判明した (Fig.8).

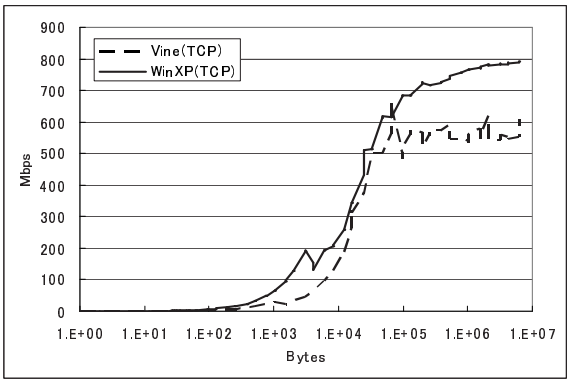

Fig. 8: Windows と Vine Linux の TCP の通信性能

我々が BNCpack の並列化に使用した mpich は TCP を用いて通信を行うようになっている．従って，この TCP 性能の差は MPI 性能にも現れてくる．

Windows 環境でも利用できる最新の MPICH2 Ver.1.0 は本年 (2004 年)11 月に Version 1.0 がリリースされているので，これも用いて従来我々が使用してきた MPICH 1.2.5(Windows, Vine Linux) との比較ベンチマークを行った．その結果を Fig.9 に示す．

これによって Windows 環境においては MPICH 1.2.5, MPICH2 共，Linux よりも最大 100Mbps 以上通信性能が良くなることが判明した．

従って，Windows 版 GMP/MPFR と MPICH2 を用いて MPIBNCpack を動作させると，短い桁数で並列度の低い計算を実行すれば多少パフォーマンスは落ちるが，並列度を上げ，長い桁数で計算を行えば，Vine Linux cluster で実行するより良いパフォーマンスが得られると期待される．実際，同じ cs-pccluster2 を Vine Linux cluster 環境 (以下, Linux 環境) と Windows cluster(以下，Windows 環境) に切り替え，それぞれの環境で補外法を用いた数値積分と，前述の Krylov 部分空間法 4 アルゴリズムの並列計算性能を比較したところ，予測通りの結果が得られた．本節では以降，これらを報告する．

4.1 補外法を用いた数値積分

我々は harmonic 数列を用いた補外法に基づく GBS アルゴリズムに対し，小規模 PC cluster 向けの改良を行ったものを提案した [23]．このアルゴリズムを用いて，Windows 環境と Linux 環境でベンチマークテストを行った結果を Fig.10 に示す．使用したのは Kahaner の 21 番目のテスト問題 (積分区間 $[0,1]$, 披積分関数 $f(x)= \cosh^{-2} 10(x-0.2)+\cosh^{-4} 100(x-0.4)+\cosh^{-6} 1000(x-0.6)$) であり，10 進 50 桁相当 (2 進 167bits) で計算を行った．

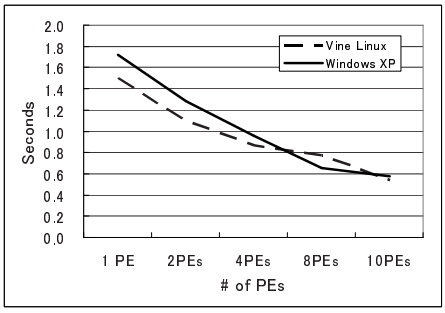

Fig. 10: 並列数値積分法の性能 (10 進 50 桁相当)

我々のアルゴリズムでは，並列度が上がると自動的に分割数も増加するため，全体の通信時間も増加する．従って，PE 数が増えるにつれて，通信性能の良い Windows 環境の方が有利になり，Linux 環境との差が近接してくることがわかる．

4.2 Krylov 部分空間法

Krylov 部分空間法は丸め誤差に対して非常に敏感で，計算精度を増やすと反復回数が大きく変化することが知られている．今回は長谷川 [25] が取り上げた非対称行列に対応した積型解法，BiCG, CGS, BiCGSTAB, GPBiCG 法を用いてベンチマークテストを行う．使用したのは 150×150 の Toeplitz 行列 ($\gamma = 1.7$) であり，真の解は $[0\ 1\ \cdots 149]^T$，停止条件は $\|\mathbf{r}_k\|_2 < 10^{-20}\|\mathbf{r}_0\|_2 + 10^{-50}$ とし，前処理は行っていない．ここでは 10 進 1000 桁相当 (2 進 3322bits) で計算した結果を示す (Fig.11(Linux 環境), Fig.12(Windows 環境)).

反復回数は BiCG 法が 134 回, CGS 法が 111 回, BiCGSTAB 法が 117 回, GPBiCG 法が 113 回であり，これにそれぞれの解法の反復 1 回あたりの計算時間が乗されて全体の計算時間が決定される．桁数が多いため，全体的にはかなり良好な並列

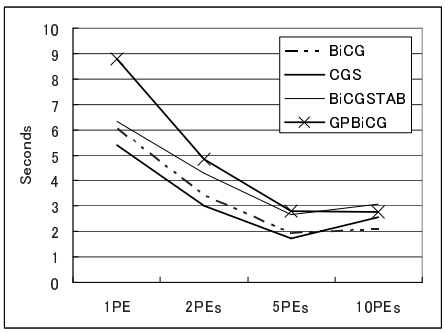

Fig. 11: Linux 環境における積型解法の性能 (10 進 1000 桁相当)

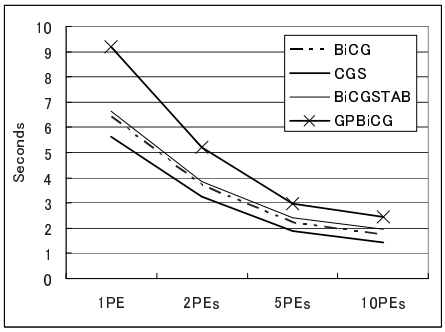

Fig. 12: Windows 環境における積型解法の性能

性能が得られているが，PE 数を 10 にすると，通信性能が勝る Windows 環境の方が，Linux 環境よりも全体の計算時間を減らしていることがわかる．

5. 結論と今後の課題

以上，3 つの研究についての報告を行った．それぞれ成果はあったが，緒に就いたばかりのもの，一応の結論は得たが未解決の部分を残すもの，次の研究課題を提示するもの等，今後に多くの課題を残すこととなった．最後にそれらを提示して本稿を終えたい．

5.1 CGI を用いた PC cluster 用 Web インターフェースの作成

基本的な機能を一通り実装することは出来たが，真に初心者にも使いやすいインターフェースになっているか，IDE に備えられている機能，並列計算に必要な機能を備えているか，という点についてはまだまだ不十分である．特に，IDE に必需品となっている help 機能，並列計算にとって必要なバッチ処理・途中停止機能は，実装はそれ程面倒ではないと予想されるので，なるべく早く提供できるようにしたい．その上で，実際にこれを使ってもらい，より良いものにしていきたいと考えている．

5.2 Windows 環境における多倍長計算の性能評価と Krylov 部分空間法の計算時間予測

Windows における MPFR の性能はそれなりに良いものであったが，まだ改善の余地がある．特に除算についてはアセンブラ部分に何がしかの問題がある可能性が高いので，調査する必要がある．

Krylov 部分空間法の計算時間予測も実用に供せる程度には精度が良いことが判明したが，より詳細な調査は必要である．その上で，NetPIPE による通信速度の実測値も活用して，並列計算時の時間予測が出来るかどうかを調査したいと考えている．

5.3 Windows MPI cluster の構築及び性能評価

前年度の紀要において報告したように，Vine Linux を用いた場合は GbE の計算速度向上率が予想以下であった．その原因については不明のままであるが，Windows を導入することによって，同じハードウェアでも性能向上が図れることが判明した．これによって，並列計算実行時において，数値積分，Krylov 部分空間法の実行時間を減らす効果があることも併せて判明し，我々の MPIBNCpack がより快適に動作する環境を構築できた．

これを土台として，最初に述べたように，2005 年度以降に本格導入される 64bit，Multi-Core CPU への対応を急ぎたいと考えている．その際には今回行ったベンチマークテスト結果よりどの程度の性能向上が行われたか，という点が焦点となるだろう．

謝辞

本研究は，2003 年度学内研究費の支援を受けて行われた．支援をして頂いた関係者に深く感謝する．また，PC の組み立てを行ってくれた幸谷ゼミ生 5 名 (李英理，米倉良平，大塚勇人，青木茂典，荒木聡介) の諸君にも感謝申し上げる．

参考文献

1) GNU MP, http://swox.com/gmp/
2) MPFR Project, http://www.mpfr.org/
3) Try MPFR!, http://www.jpsearch.net/try_mpfr.html
4) B.Gladman, GMP and MPFR for Win32, http://fp.gladman.plus.com/computing/gmp4win.htm
5) BNCpack マニュアル, http://na-inet.jp/na/bnc/bnc.pdf
6) MPIBNCpack マニュアル, http://na-inet.jp/na/bnc/mpibnc.pdf
7) MPICH, http://www-unix.mcs.anl.gov/mpi/mpich/
8) MPICH2, http://www-unix.mcs.anl.gov/mpi/mpich2/
9) NetPIPE, http://www.scl.ameslab.gov/netpipe/
10) PC Watch, http://pc.watch.impress.co.jp/docs/2004/1019/intel.htm
11) Cygwin Home, http://www.cygwin.com/
12) 幸谷智紀, PC Cluster の性能評価〜メモリ及びネットワーク帯域計測編〜, http://na-inet.jp/na/netpipebench.pdf
13) 幸谷智紀, 「A Tutorial of BNCpack & MPIBNCpack」, http://na-inet.jp/tutorial/
14) 幸谷智紀,「Vine Linux による PC cluster の構築」, http://na-inet.jp/na/bnc/mpipc.pdf
15) 幸谷智紀,「Vine Linux による PC cluster の構築 Version 2」, http://na-inet.jp/na/bnc/mpipc2.pdf
16) H. A. van der Vorst, Iterative Krylov Methods for Large Linear Systems, (Cambrige University Press, 2003).

17) 澤木敦, PC クラスタの実験用 CGI インターフェースの作成, 2004 年度卒業研究.
18) 鈴木文明, Windows 環境における多倍長計算のベンチマークテスト, 2004 年度卒業研究.
19) 越須賀正雄, Windows を用いた PC Cluster のベンチマーク, 2004 年度卒業研究.
20) 幸谷智紀, Windows を用いた PC Cluster における多倍長数値計算ライブラリ MPIBNCpack の性能評価, HPCS2005 ポスターセッション, 2005.
21) 幸谷智紀,「PC cluster を用いた多倍長数値計算ライブラリの並列分散化」, Linux Conference 2003.
22) 幸谷智紀,「PC cluster を用いた多倍長数値計算ライブラリの並列分散化」, FIT 2003.
23) 幸谷智紀・鈴木千里, 補外法を用いた並列多倍長数値積分法の実装と性能評価, SWoPP2004, 2004.
24) 幸谷智紀, 補外法を用いた並列任意精度 ODE Solver の実装と PC cluster における性能評価, 静岡理工科大学紀要 Vol.12(2004), pp.203-218.
25) 長谷川秀彦, 4 倍精度演算における積型反復解法の比較, LA 研究会, 2000.

付録 1　時間計測に使用した Windows 及び UNIX 対応関数

Windows 及び UNIX 環境下で共通に時間計測に使用した get_sec 関数を以下に示す．WIN32 が定義されている場合は, Windows 用にコンパイルされるものと判断する．

```
#include <stdio.h>
#include <time.h>

#ifdef WIN32
#include <windows.h>
#else
#include <sys/times.h>
#include <sys/time.h>
#endif

#define DIVTIMES 2

/* flag == 0: No print */
double get_sec(int flag)
{
#ifdef WIN32
  static int first = 1;
  static LARGE_INTEGER _tstart;
  static LARGE_INTEGER freq;

  if(first)
  {
    QueryPerformanceFrequency(&freq);
    first = 0;
  }
  QueryPerformanceCounter(&_tstart);
  return ((double)_tstart.QuadPart)/((double)fr
eq.QuadPart);
#else
  double ret;
  struct tms tmp;

#ifdef USE_CLOCK
  if(flag != 0)
    printf("Time  : %d / %d\n", (int)clock(),
 CLOCKS_PER_SEC);
  ret = (double)(clock()) / CLOCKS_PER_SEC;
#else
  times(&tmp);
  if(flag != 0)
  {
    printf("User Time  : %d / %d\n", tmp.tms_ut
ime, CLK_TCK);
    printf("System Time: %d / %d\n", tmp.tms_st
ime, CLK_TCK);
    printf("CUser Time  : %d / %d\n", tmp.tms_c
utime, CLK_TCK);
    printf("CSystem Time: %d / %d\n", tmp.tms_c
stime, CLK_TCK);
    printf("Ret(utime+stime)   : %g\n", (double
)(tmp.tms_utime + tmp.tms_stime) / CLK_TCK);
    printf("Ret(cutime+cstime) : %g\n", (double
)(tmp.tms_cutime + tmp.tms_cstime) / CLK_TCK);
  }
  ret = (double)(tmp.tms_utime + tmp.tms_stime)
/ CLK_TCK;
#endif
  return ret;
#endif
}
```

付録 2　ベンチマークに使用した積型 Krylov 部分空間法のアルゴリズム

以下，解くべき連立一次方程式を

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

とする．$A \in M_n(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^n$ は既知.

BiCG 法

$\mathbf{x}_0$: 初期値
$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)
$\widetilde{\mathbf{r}_0}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}_0}) \neq 0$ を満たす任意のベクトル．例えば $\widetilde{\mathbf{r}_0} = \mathbf{r}_0$.
K: 前処理行列 (前処理なしの場合は $K = I$)
for $i = 1, 2, \ldots$
　$K\mathbf{w}_{i-1} = \mathbf{r}_{i-1}$ を解いて $\mathbf{w}_{i-1}$ を求める．
　$K^T \widetilde{\mathbf{w}_{i-1}} = \widetilde{\mathbf{r}_{i-1}}$ を解いて $\widetilde{\mathbf{w}_{i-1}}$ を求める．
　$\rho_{i-1} = (\widetilde{\mathbf{w}_{i-1}}, \mathbf{w}_{i-1})$
　if $\rho_{i-1} = 0$ **then** 終了．
　if $i = 1$ **then**
　　$\mathbf{p}_1 = \mathbf{w}_0$
　　$\widetilde{\mathbf{p}_1} = \widetilde{\mathbf{w}_0}$
　else
　　$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
　　$\mathbf{p}_i = \mathbf{w}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1}$
　　$\widetilde{\mathbf{p}_i} = \widetilde{\mathbf{w}_i} + \beta_{i-1}\widetilde{\mathbf{p}_{i-1}}$
　end if
　$\mathbf{z}_i = A\mathbf{p}_i$

$\widetilde{\mathbf{z}}_i = A\widetilde{\mathbf{p}}_i$
$\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{p}}_i, \mathbf{z}_i)$
$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \mathbf{p}_i$
$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i \mathbf{z}_i$
$\widetilde{\mathbf{r}}_i = \widetilde{\mathbf{r}_{i-1}} - \alpha_i \widetilde{\mathbf{z}}_i$
収束判定
end for

CGS 法

$\mathbf{x}_0$: 初期値
$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)
$\widetilde{\mathbf{r}}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}}) \neq 0$ を満足する任意ベクトル．例えば $\widetilde{\mathbf{r}} = \mathbf{r}_0$.
K: 前処理行列 (前処理なしの場合は $K = I$)
for $i = 1, 2, ...$
$\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$
if $\rho_{i-1} = 0$ **then** 終了.
if $i = 1$ **then**
$\mathbf{u}_1 = \mathbf{r}_0$
$\mathbf{p}_1 = \mathbf{u}_1$
else
$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
$\mathbf{u}_i = \mathbf{r}_{i-1} + \beta_{i-1}\mathbf{q}_{i-1}$
$\mathbf{p}_i = \mathbf{u}_i + \beta_{i-1}(\mathbf{q}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1})$
end if
$K\widehat{\mathbf{p}} = \mathbf{p}_i$ を解いて $\widehat{\mathbf{p}}$ を求める.
$\widehat{\mathbf{v}} = A\widehat{\mathbf{p}_i}$
$\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{r}}, \widehat{\mathbf{v}})$
$\mathbf{q}_i = \mathbf{u}_i - \alpha_i \widehat{\mathbf{u}}$
$\widehat{\mathbf{u}}$ を $K\widehat{\mathbf{u}} = \mathbf{u}_i + \mathbf{q}_i$ を解いて求める.
$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \widehat{\mathbf{u}}$
$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i A\widehat{\mathbf{u}}$
収束判定
end for

BiCGSTAB 法

$\mathbf{x}_0$: 初期値
$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)
$\widetilde{\mathbf{r}}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}}) \neq 0$ を満足する任意ベクトル．例えば $\widetilde{\mathbf{r}} = \mathbf{r}_0$.
K: 前処理行列 (前処理なしの場合は $K = I$)
for $i = 1, 2, ...$
$\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$
if $\rho_{i-1} = 0$ **then** 終了.
if $i = 1$ **then**
$\mathbf{p}_1 = \mathbf{r}_0$
else
$\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
$\mathbf{p}_i = \mathbf{r}_i + \beta_{i-1}(_{i-1} - \omega_{i-1}\mathbf{v}_{i-1})$
end if
$\widehat{\mathbf{p}}$ を $K\widehat{\mathbf{p}} = \mathbf{p}_i$ を解いて求める.
$\widehat{\mathbf{v}_i} = A\widehat{\mathbf{p}}$
$\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{r}}, \mathbf{v}_i)$
$\mathbf{s} = \mathbf{r}_{i-1} - \alpha_i \mathbf{v}_i$
if $\|s\|$ が十分に小さい **then**
$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \widehat{\mathbf{p}}$
終了.
end if
$\widehat{\mathbf{s}}$ を $K\widehat{\mathbf{s}} = \mathbf{s}$ を解いて求める.
$\mathbf{t} = A\widehat{\mathbf{s}}$
$\omega_i = (\mathbf{t}, \mathbf{s})/(\mathbf{t}, \mathbf{t})$
$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \widehat{\mathbf{p}} + \omega_i \widehat{\mathbf{s}}$
収束判定
$\mathbf{r}_i = \mathbf{s} - \omega_i \mathbf{t}$
$\omega_i \neq 0$ であれば反復続行.
end for

GPBiCG 法

$\mathbf{x}_0$: 初期値
$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)
$\widetilde{\mathbf{r}}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}}) \neq 0$ を満足する任意ベクトル．例えば $\widetilde{\mathbf{r}} = \mathbf{r}_0$.
$\mathbf{u} = \mathbf{z} = 0$
for $i = 1, 2, ...$
$\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$
if $\rho_{i-1} = 0$ **then** 終了.
if $i = 1$ **then**
$\mathbf{p} = \mathbf{r}_0$
$\mathbf{q} = A\mathbf{p}$
$\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$
$\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i \mathbf{q}$
$\mathbf{v} = A\mathbf{t}$
$\mathbf{y} = \alpha_i \mathbf{q} - \mathbf{r}_{i-1}$
$\mu_2 = (\mathbf{v}, \mathbf{t})$
$\mu_5 = (\mathbf{v}, \mathbf{v})$
$\zeta = \mu_2/\mu_5$
$\eta = 0$
else
$\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\zeta)$
$\mathbf{w} = \mathbf{v} + \beta_{i-1}\mathbf{q}$
$\mathbf{p} = \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{p} - \mathbf{u})$
$\mathbf{q} = A\mathbf{p}$
$\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$
$\mathbf{s} = \mathbf{t} - \mathbf{r}_{i-1}$
$\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i \mathbf{q}$
$\mathbf{v} = A\mathbf{t}$
$\mathbf{y} = \mathbf{s} - \alpha_i(\mathbf{w} - \mathbf{q})$
$\mu_1 = (\mathbf{y}, \mathbf{y})$
$\mu_2 = (\mathbf{v}, \mathbf{t})$
$\mu_3 = (\mathbf{y}, \mathbf{t})$
$\mu_4 = (\mathbf{v}, \mathbf{y})$
$\mu_5 = (\mathbf{v}, \mathbf{v})$
$\tau = \mu_5\mu_1 - \overline{\mu_4}\mu_4$
$\zeta = (\mu_1\mu_2 - \mu_3\mu_4)/\tau$
$\eta = (\mu_5\mu_3 - \overline{\mu_4}\mu_2)/\tau$
end if
$\mathbf{u} = \zeta\mathbf{q} + \eta(\mathbf{s} + \beta_{i-1}\mathbf{u})$
$\mathbf{z} = \zeta\mathbf{r}_{i-1} + \zeta\mathbf{z} - \alpha_i\mathbf{u}$
$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p} + \mathbf{z}$
収束判定
$\mathbf{r}_i = \mathbf{t} - \eta\mathbf{y} - \zeta\mathbf{u}$
$\zeta \neq 0$ であれば反復続行.
end for