Vol.2012-HPC-136 No.18
2012/10/4

MPACK:高精度 BLAS, LAPACK の概要と性能評価

中田 真秀[1,a)]

概要：MPACK は、高精度線形代数演算ライブラリである。BLAS, LAPACK をベースとしていて、行列-行列積、ベクトルの内積演算等の処理を行う MBLAS と MBLAS をビルディングブロックとして固有値を求めたり、線形方程式を解いたりする MLAPACK から成る。特徴は (i) 様々な高精度演算ライブラリをサポートし、非常に高精度だが低速、四倍精度程度だがより高速な演算などに対応する。(ii) 高精度演算を C++のクラスで行い、従来の単精度、倍精度型のような形でのプログラミングが可能となっている、(iii) 様々な環境 (Linux, Windows, Mac) などをサポートする、(iv) 高精度線形代数演算ライブラリ MPACK として (http://mplapack.sourceforge.net/) でオープンソースソフトウェアとして公開されている、などである。執筆時点で最新のバージョンは 2012/6/16 にリリースされた 0.7.0 である。今回、ナイーブに OpenMP でマルチスレッド化した行列-行列積の性能評価を、Intel Xeon E7-8870 (Westmere EX) 2.4GHz 40 コアマシン、AMD Opteron 6174 (Magny Cours) 2.2GHz 48 コアマシンで行った。前者はコア数に概ねスケールする結果が得られたが、後者ではコア数に比例しない結果が得られた。

1. はじめに

エクサスケールコンピューティングに向け、高精度演算が注目されている。なぜならば、倍精度では解けない数値的に難しい問題を解くことや、数値計算を安定させたい場合や、10^{20} 回程度の圧倒的な回数の演算が行われるため、誤差の蓄積による結果の信頼性が問題となることなどがあるからである。

高精度化すべき演算にも様々なものがあるが、その中で特に線形代数演算の高精度化が重要であると考え、それに焦点をしぼった。その理由は重要な問題の多くが線形代数の問題として定式化できるということによる。また、線形代数の高精度化はどうすればよいかという問いにも様々な答えがあると考えられるが、私は、デファクトスタンダードのライブラリである BLAS [1]、LAPACK [2] を元に高精度版の MPACK [3] というライブラリを構築、ということを提案する。

本稿では、BLAS, LAPACK の紹介、浮動小数点数と多倍長精度演算ライブラリ、MPACK の紹介、MPACK の品質保証方法、 MPACK の作成方法、入手, インストール及び使い方、行列-行列積の性能評価、まとめについて述べる。

[1] 理化学研究所 情報基盤センター
RIKEN ACCC, Hirosawa 2-1, Wako, Saitama 351–0198, Japan
[a)] maho@riken.jp

2. BLAS, LAPACK の紹介

BLAS (Basic Linear Algebra Subprograms) [1], LAPACK (Linear Algebra PACKage) [2] は広く使われている標準的な線形代数演算パッケージであり、MPACK はこのソースを元にして開発した。さて、BLAS とは FORTRAN77 で書かれた線形代数演算ライブラリ、ルーチン群で基本的なベクトルおよび行列演算を行う為の標準的なビルディングブロックとなっている。これには Level 1, Level 2, Level 3 があり、それぞれベクトル-ベクトル演算、ベクトル-行列演算、行列-行列演算を行う様々なルーチンが定義されている。BLAS それ自身が API を定義し参照実装を提供するため、さらに高速な外部ライブラリの存在、プログラムの構造化、プログラムの品質が高いことなども重要である。

次に LAPACK であるが、これは FORTRAN 90 (バージョン 3.1 以前は FORTRAN 77) で書かれ、BLAS をビルディングブロックとして使いつつ、連立一次方程式、最小二乗法、固有値問題、特異値問題などを解いたり、行列の分解 (LU, Cholesky, QR, SVD, Schur, 一般化 Schur 分解) を行うルーチン群である。その他にも様々な種類のルーチンがあり、2010 年 11 月 14 日にリリースされた LAPACK 3.3 にはなんと 1657 ものルーチンがある。そして LAPACK の一部のルーチンについては高速な BLAS を利用すると高速化される。例えば高速な行列-行列積の `dgemm` を使うとコレスキー分解や LU 分解も高速化される。より詳しい情

© 2012 Information Processing Society of Japan

報は netlib.org から参照実装、今後の方針、議事録、実装の詳細、評価、 論文などが入手できるので参照された。

3. 浮動小数点数と多倍長精度演算ライブラリについて

コンピューター上で実数を扱うのは難しいため、IEEE 754 規格 [4] に基づき、次のような浮動小数点フォーマットと、その上の算術 (浮動小数点演算: 加減乗除や誤差丸めなど) がよく使われる。

$$(-1)^s \times 2^e \times m, \tag{1}$$

ここで s は符号 (0 または 1), e は指数部, そして仮数部 m は次のような文字列で表されるような数で,

$$m = d_0.d_1d_2\cdots d_{p-1}, \tag{2}$$

d_i は 0 または 1(従って $0 \le m < 2$ となる)、 そして p は有効桁である。IEEE 754 の binary64 (倍精度) では e は $-1022 \le e \le +1023$ であり p は 53 なので, 10 進 16 桁の精度を持つ ($15.954 \simeq 53\log_{10}2$)。ただ、どのように演算を定義しても一般には誤差が生じる。これが誤った結果や、数値的な不安定性を引き起こしたりする。 解決方法には e および特に p の範囲を大きくすることで精度を高める計算手法である、多倍長精度計算がよく使われる。

MPACK の高精度化には多倍長精度計算を行うことにした。また多倍長精度の数同士の演算はライブラリに任せることにした。

4. MPACK: 高精度汎用線形代数演算ライブラリ

MPACK は MBLAS および MLAPACK という二つのパッケージからなる。MBLAS は BLAS を、 MLAPACK は LAPACK をそれぞれ高精度化したものである。 あらゆる高精度に対するニーズに応えるため柔軟な作りを心がけている。このセクションでは、MPACK の特徴、ルーチン命名規則、関数コールの仕方、サポートされている関数、多倍長精度演算ライブラリについて、作成方法、品質保証方法を述べる。

4.1 MPACK の特徴

(1) BLAS、LAPACK ベースの高精度ライブラリを構築。参照実装 (API) の提供が第一目標。
(2) これまでの資産活用を容易にする。
(3) バージョン 0.7.0 (2012/8/20); 100 個の MLAPACK ルーチンが完成、全ての MBLAS が完成。
(4) マルチプラットフォーム:Linux/Mac/Windows/FreeBSD など。
(5) 可搬性を高めるため、6 種の高精度演算手法に対応した (GMP/MPFR/DD/QD/__float128/double)。
精度をコンピュータ資源の許す限り上げられるようにするため、まず GMP[5] をサポートし、より厳密な計算を行えるように MPFR(任意精度+丸めモード)[6] および MPC(MPFR ベースの複素数演算ライブラリ)[7] をサポートした。精度と速度の妥協点も様々に用意した。 QD ライブラリ [8] は倍精度演算のみで実装された倍々精度、8 倍精度演算を行う。IEEE 754 の binary128 の演算を行う為に__float128 (gcc のみ) や、MPACK の品質向上のため、または倍精度と同じ結果を出すためなどの目的で IEEE 754 の倍精度である double もサポートした。
(6) C++で書き直したため、よりプログラミングしやすく, 可読性が高くなった。
高精度演算型は C++のクラスとして扱うため、倍精度型である double とほぼ同じように扱ったプログラミングができるようになった。高速化したい場合は参照実装を高速化すればよい。これは BLAS, LAPACK で行われているようなアプローチと同じものになるようにした。
(7) 2-条項 BSD ライセンス:商用利用、改変、再配付自由。線形代数演算ライブラリにはソースコードが開示されてないものがあるが、このような基本的なライブラリにはソースコードの開示は重要である。
(8) オープンな開発体制によりコミュニティの成果を取り込みやすくしている。

4.2 ルーチンの命名規則

BLAS, LAPACK では (実) 単精度, 倍精度で計算するルーチンに於いてルーチン名に “s”, “d” を接頭辞としてつけ, 単精度複素数, 倍精度複素数については “c”, “z” をつけている。また, C++は関数名として大文字, 小文字を区別するが, FORTRAN は関数, サブルーチン名では大文字小文字を区別しない。これらを踏まえ, C++でのルーチン名の接頭辞で実数, 複素数を表すには大文字でそれぞれ “R”, “C” とし、あとは小文字とした。従って例えば

- daxpy, zaxpy → Raxpy, Caxpy
- dgemm, zgemm → Rgemm, Cgemm
- dsterf, dsyev → Rsterf, Rsyev
- dzabs1, dzasum → RCabs1, RCasum

のようにした。

4.3 関数のコールの仕方

呼び出しにおける BLAS, LAPACK との違いは call by value および call by reference の違いである。つまり、BLAS, LAPACK を C から使うようにするには以下のように参照として渡さねばならないが、

```
dgemm_f77("N", "N", &n, &n, &n, &One...
dgetri_f77(&n, A, &n, ipiv, work, &lwork...
```

© 2012 Information Processing Society of Japan

表 1 LEVEL1 MBLAS

Crotg	Cscal	Rrotg	Rrot	Rrotm	CRrot
Cswap	Rswap	CRscal	Rscal	Ccopy	Rcopy
Caxpy	Raxpy	Rdot	Cdotc	Cdotu	RCnrm2
Rnrm2	Rasum	iCasum	iRamax	RCabs1	Mlsame
Mxerbla					

表 2 LEVEL2 MBLAS

Cgemv	Rgemv	Cgbmv	Rgbmv	Chemv	Chbmv
Chpmv	Rsymv	Rsbmv	Ctrmv	Cgemv	Rgemv
Cgbmv	Rgemv	Chemv	Chbmv	Chpmv	Rsymv
Rsbmv	Rspmv	Ctrmv	Rtrmv	Ctbmv	Ctpmv
Rtpmv	Ctrsv	Rtrsv	Ctbsv	Rtbsv	Ctpsv
Rger	Cgeru	Cgerc	Cher	Chpr	Cher2
Chpr2	Rsyr	Rspr	Rsyr2	Rspr2	

表 3 LEVEL3 MBLAS

Cgemm	Rgemm	Csymm	Rsymm	Chemm	Csyrk
Rsyrk	Cherk	Csyr2k	Rsyr2k	Cher2k	Ctrmm
Rtrmm	Ctrsm	Rtrsm			

MPACK の場合は, 適切な箇所では, 以下のように値を渡すようにしている。

```
Rgemm("n", "n", n, n, n, alpha, A...
Rgetrf(n, n, A, n, ipiv, &info);
Rgetri(n, A, n, ipiv, work, lwork, &info);
Rsyev("V", "U", n, A, n, w, work, &lwork...
```

4.4 サポートする関数

表 1, 2, 3 に MBLAS でサポートされているルーチンを挙げ、表 4 に MLAPACK でサポートされているルーチンを挙げた。今後 MLAPACK は LAPACK のルーチンのうちなるべく多くを実装したいと考えている。

4.5 多倍長精度演算ライブラリについて

MPACK ではいくつも多倍長精度演算ライブラリをサポートする. そのデフォルトでの精度および型宣言を表 5 にまとめた. これらは混在も可能である.

4.6 MPACK の作成方法:主に FORTRAN から C++ への変換について

MPACK は C++で書くことにした. 高級言語的な書き方をすることで初めて, GMP, MPFR, QD, DD, __float128, double と 6 つの高精度計算手法に対応することができた. さらに新たな多倍長精度演算ライブラリなどにも迅速に対応できるようになっている. 品質保証の観点から BLAS, LAPACK になるべく近いプログラムになるようにした. まず, MBLAS は手変換であったが, MLAPACK は 600 を越えるルーチンがあるため FORTRAN から C++への機械的

表 4 MLAPACK (100 ルーチン)

Mutils	Rlamch	Rlae2	Rlaev2	Claev2	Rlassq
Classq	Rlanst	Clanht	Rlansy	Clansy	Clanhe
Rlapy2	Rlarfg	Rlapy3	Rladiv	Cladiv	Clarfg
Rlartg	Clartg	Rlaset	Claset	Rlasr	Clasr
Rpotf2	Clacgv	Cpotf2	Rlascl	Clascl	Rlasrt
Rsytd2	Chetd2	Rsteqr	Csteqr	Rsterf	Rlarf
Clarf	Rorg2l	Cung2l	Rorg2r	Cung2r	Rlarft
Clarft	Rlarfb	Clarfb	Rorgqr	Cungqr	Rorgql
Cungql	Rlatrd	Clatrd	Rsytrd	Chetrd	Rorgtr
Cungtr	Rsyev	Cheev	Rpotrf	Cpotrf	Clacrm
Rtrti2	Ctrti2	Rtrtri	Ctrtri	Rgetf2	Cgetf2
Rlaswp	Claswp	Rgetrf	Cgetrf	Rgetri	Cgetri
Rgetrs	Cgetrs	Rgesv	Cgesv	Rtrtrs	Ctrtrs
Rlasyf	Clasyf	Clahef	Clacrt	Claesy	Crot
Cspmv	Cspr	Csymv	Csyr	iCmax1	RCsum1
Rpotrs	Rposv	Rgeequ	Rlatrs	Rlange	Rgecon
Rlauu2	Rlauum	Rpotri	Rpocon		

表 5 多倍長精度演算ライブラリの精度と MPACK での型. 括弧内はデフォルト.

ライブラリ	10 進精度	実数型	複素数型
GMP	任意 (154)	mpf_class	mpc_class
QD	64	qd_real	qd_complex
DD	32	dd_real	dd_complex
MPFR	任意 (154)	mpreal	mpcomplex
__float128	32	__float128	complex<__float128>
double	16	double	complex<double>

© 2012 Information Processing Society of Japan

な変換を F2C というプログラムを使って行った. ただ, そのままだと可読性に欠けるため, F2C にも手を加え, そして, 変換後の C のソースに sed を通し, さらに手直した上で, コンパイルが通るようにした. 最大の問題は, C++と FORTRAN の配列とループの取扱いの違いであった. 単純なものは単に 1 を引く程度で良いが、複雑なものについては自明ではない。バグ回避と効率を上げるために機械的に書き換える次のような手法を使った。

- 配列については, FORTRAN と C++では, 0 から始まるか, 1 から始まるかの違いがあるが, これは
  `FORTRAN : A(i) , C++: A[i-1]`
  のように置き換えた.
- 2 次元配列 (行列) の違いについては, FORTRAN の一次元配列をそのまま C++に置き換えた. つまり
  `FORTRAN : A(i,j) , C++: A[(i-1) + (j-1) * lda]`
  のように置き換えた。 説明を省くが, Column-major や leading dimension の考え方が出てくることに注意.
- ループについては, これは FORTRAN でも C++でも同じとした.
  `FORTRAN : DO I = 1, N , C++: for(i = 1; i <=N; i++)`
  これで機械的に変換するだけで正しいコードとなる. つまり, ループ内での配列の要素の整合性がとれる. 例えば,

```
DO I = 1, N
    DAXPY (N-I, A(I,J+1), B(I), 1,
                Y(MAX(N-I,I)), 1)
END DO
```

は

```
for (i = 1; i <= N; i ++) {
    Raxpy (n-i, A [(i - 1) + j * lda],
      B[i-1], 1, Y [max(n-i, i)-1], 1);
}
```

のように変換した。ここのループを C/C++らしく 0 から始めようとすると、簡単ではなくなってしまう。

4.7 MPACK の品質保証方法;BLAS, LAPACK と比較

本来, 線形代数演算ライブラリは正確さが命で、計算速度は二の次である. だが正確さとは何か、と考え始めると本質的に検証が困難なことがわかる. ここではある一定の意味で正確ということで, 品質保証とした. 具体的には BLAS, LAPACK の正確さを前提として, MPACK は基本的には 2 つ (i) MPACK の MPFR 版に BLAS, LAPACK に様々な値を代入してそれらを比較してよい一致を得る, (ii) MPACK の MPFR 版と GMP 版, QD 版, DD 版, __float128 版、double 版と比較してよい一致を得る, が満たされたとき, 品質保証されたとした. (M)BLAS は基本的に代数演算で構成されているため、このような方法でバグはほとんどとれる。ただし (M)LAPACK には、収束の概念が入る。つまり反復解法で、例えば誤差がある小さな数以下ならば収束した、という判定を行う。このようなときは実際には今回提示した方法は完全ではない。解の精度が低くなる場合などが考えられる。しかし今のところ大きく問題はなさそうである。

5. 入手, インストールおよび使い方

5.1 入手方法

MPACK は http://mplapack.sourceforge.net/から入手, インストールして利用できる. いつでも最新バージョンを入手して欲しい. 原稿執筆現在, 最新バージョンは 2012/6/16 にリリースされた 0.7.0 である.

5.2 インストール

ここでは, Linux, Mac でのインストール方法を述べる. Windows については Linux でクロス開発環境を作りビルドする必要があるので手順が長くなりすぎるため割愛した。Linux で必要なのは gcc, g++バージョン 4.2 以上推奨である。通常これらは apt-get (Ubuntu, Debian) や yum (CentOS, Fedora, RedHat) でインストールできる。Mac で必要なのは Xcode という開発ツールである. さて, それらの用意ができたら以下の手順でビルド, インストールができる。デフォルトでは GMP 版のみが構築され、/usr/local/以下にインストールされる。

```
$ tar xvfz mpack-0.7.0.tar.gz
$ cd mpack-0.7.0
$ ./configure
$ make install
```

展開した `mpack-0.7.0/examples` の下には, Rgemm, 固有値, 逆行列, 条件数近似, ヒルベルト行列をそれぞれ求めるサンプルがあるので, 参考にして欲しい.

また、他の高精度演算ライブラリを用いた MPACK を生成するには、`./configure` を行うときにコマンドラインに適当なものを代入する。例えば`--enable-mpfr=yes` を渡すと MPFR 版、`--enable-qd=yes` を渡すと QD/DD 版、`--enable-__float128=yes` を渡すと__float128 版、`--enable-double=yes` を渡すと double 版が生成される。最後に`--enable-debug=yes` を渡すと section 4.7 で示した品質保証用のプログラムが生成される。これに yes を渡した場合は FORTRAN コンパイラが必要となる。すべての高精度演算ライブラリについて MPACK を構築しそれらを実際にチェックするには

```
$ tar xvfz mpack-0.7.0.tar.gz
$ cd mpack-0.7.0
$ ./configure --enable-debug=yes --enable-mpfr=yes \
   --enable-qd=yes --enable-__float128=yes \
   --enable-double=yes
```

© 2012 Information Processing Society of Japan

```
$ make -j20
$ make check
$ make install
```

などとするとよい。ただし、make や make check には膨大な時間がかかり、環境によってはコンパイラの不具合などによってエラーがおこることがある。

5.3 サンプル:ヒルベルト行列の逆行列を求めてみる

ヒルベルト行列 H は $n \times n$ 行列で, 各要素 H_{ij} が

$$H_{ij} = \frac{1}{i+j-1} \ (1 \le i, j \le n)$$

のように定義される。ヒルベルト行列は n が大きくなると条件数が指数関数的に大きくなるのが特徴で、 たとえば $n = 12$ のとき 1.7132×10^{16} となり, $n \ge 12$ では倍精度の範囲では逆行列を求めるのが困難である. さて, この逆行列を MPACK を用いて求めてみよう。図 1 のプログラムをエディタなどを使って入力し、 “hilbert_mpfr.cpp” としてセーブする:

入力, セーブが終わったら次のようにコンパイルする。

```
$ g++ -fopenmp -I/usr/local/include/mpack \
 -I/usr/local/include hilbert_mpfr.cpp \
-o hilbert_mpfr -L/usr/local/lib -lmpfrcxx \
-lmlapack_mpfr -lmblas_mpfr -lmpfr \
-lmpc -lgmp
```

これで

```
$ ./hilbert_mpfr
```

とすれば走るはずだ. この hilbert_mpfr はヒルベルト行列の逆行列を求め, さらにこれらの二つの行列の積と単位行列との 1-ノルムによる誤差も計算する. デフォルトでは MPFR では約 154 桁の精度をもち, $n = 50$ の場合の誤差は

```
1norm(I-A*invA)=1.8784847910273908e-73
```

となった.

5.4 精度を実行時に変化させる

MPACK では環境変数を設定することで, 精度を実行時に変化させることができる. MPFR 版の場合は, MPACK_MPFR_PRECISION という環境変数に仮数部 の長さをビット単位で入れることで実行時に精度を変更できる. 精度を約 308 桁として hilbert_mpfr を走らせるには

```
$ MPACK_MPFR_PRECISION=1024;
$ export MPACK_MPFR_PRECISION ; ./hilbert_mpfr
```

とすればよい. こうすることで誤差は $n = 50$ の場合

```
1norm(I-A*invA)=1.9318639065194500e-226
```

となり, 153 桁精度が良くなった.

図 1 MPACK サンプル:“hilbert_mpfr.cpp”

```
#include <mblas_mpfr.h>
#include <mlapack_mpfr.h>
void inv_hilbert_matrix(int n) {
    mpackint lwork, info;
    mpreal *A = new mpreal[n * n];
    mpreal *Aorg = new mpreal[n * n];
    mpreal *C = new mpreal[n * n];
    mpackint *ipiv = new mpackint[n];
    mpreal One = 1.0, Zero = 0.0, mtmp;
//setting A matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            mtmp = (i + 1) + (j + 1) - 1;
            A[i + j * n] = One / mtmp;
            Aorg[i + j * n] = One / mtmp;
        }
    }
//work space query
    lwork = -1; mpreal *work = new mpreal[1];
    Rgetri(n, A, n, ipiv, work, lwork, &info);
    lwork = int (double (work[0]));
    delete[]work;
    work = new mpreal [std::max(1, (int) lwork)];
//inverse matrix
    Rgetrf(n, n, A, n, ipiv, &info);
    Rgetri(n, A, n, ipiv, work, lwork, &info);
    One = 1.0, Zero = 0.0;
    Rgemm("N", "N", n, n, n, One, Aorg,
          n, A, n, Zero, C, n);
    printf("1norm(I-A*invA)=");
    mtmp = 0.0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (mtmp < abs(C[i + j * n]
                 - ((i == j) ? 1.0 : 0.0)))
                mtmp = abs(C[i + j * n]
                 - ((i == j) ? 1.0 : 0.0));
        }
    }
    printf("%.16e\n", double (mtmp));
    delete[]A; delete[]Aorg;
    delete[]C; delete[]ipiv;
    delete[]work;
}
int main() {
    for (int n = 1; n <= 50; n++) {
        printf("# inversion of Hilbert"
               " matrix of order n=%d\n", n);
        inv_hilbert_matrix(n);
    }
}
```

© 2012 Information Processing Society of Japan

6. 行列-行列積の性能評価

メニイコアの Graphics Processing Unit (GPU) が安価かつ高速なプロセッサとして注目されているが、高精度計算を GPU で行う場合、例えば GMP ライブラリのポーティング、チューニングなどを行わねばならないが、これらの作業は自明ではない。近年、GPU の高性能化を受け、CPU もマルチコア化への模索が始まっている。この場合もチューニングは自明ではないため、ナイーヴな OpenMP でのスレッド並列化によるパフォーマンスの性能評価は非常に重要である。

また、高精度計算はメモリバンド幅の大きさより、演算処理速度が重要なファクターとなる。今回は `Rgemm` について性能評価を行った。`Rgemm` は `dgemm` の高精度版で、行列 A, B, C, スカラー α, β について

$$C \to \alpha AB + \beta C$$

を行うルーチンである。今回は A, B, C について正方行列とし、A, B, C, α, β はランダムな数で埋め、さまざまなサイズでの計測を行った。OpenMP での実装は、リファレンス実装に対して、単純に行または列でスレッドを生成するようにした。ブロッキング等は行っていない。

マシンは Intel Xeon E7-8870 (Westmere EX) 2.4GHz 40 コア (=10 × 4:理論性能値 384GFlops) マシンおよび AMD Opteron 6174 (Magny Cours) 2.2GHz 48 コア (=12 × 4: 理論性能値 422GFlops) のマシンで、Rgemm を GMP 版、double-double 版の二つの精度で測定した。GMP は任意精度だが 10 進 154 桁程度の有効桁を持たせ、double-double 版は 10 進約 32 桁の有効桁がある。コンパイラは Intel Composer 12.1.4 を用いた。ここで 2, 4, 8, 16, 24, 32, 40 スレッドを立ち上げた場合について測定したものについて、図 2, 図 3, 図 4, 図 5 に示す。Westmere EX の GMP 版 (図 2) は最大で約 250MFlops 程度でていた。ただし、32 スレッドで計算させた場合とと 40 スレッドで計算さてた場合を比較すると、行列のサイズが 800 程度までは 32 スレッドのほうが高速であった。その後 40 スレッドの場合はかなり性能が不安定となった。また、行列のサイズによってパフォーマンスがのこぎり状に変化することがあった。これは倍精度でもメモリアクセスがある程度無い場合はパフォーマンスが落ちるということで発生するが、高精度版でも発生することがわかった。Westmere EX の double-double 版 (図 3) についても GMP 版とほぼ同様な傾向が見られ、パフォーマンスは最大 7GFlops 程度出ることがわかった。ただ、プロセッサの最適化が進めば、20GFlops 程度まで出ると考えられる (Westmere EX では FMA 等のサポートが無いため、加算、積算にほぼ 20 オペレーションかかるため)。

次に Magny Cours の性能を見てみる。GMP 版 (図 4) で

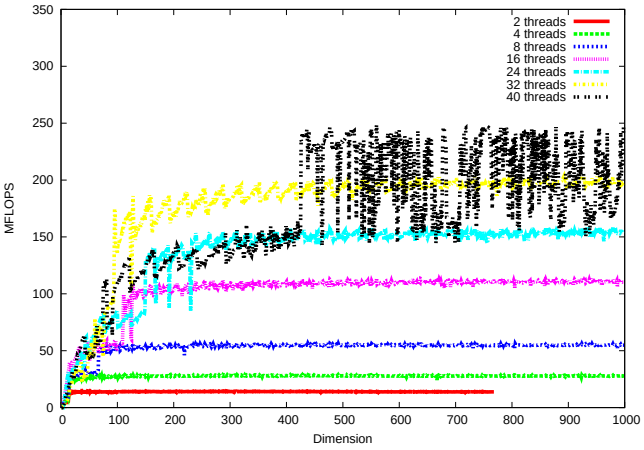

図 2 Rgemm GMP performance on Intel Xeon E7-8870 (Westmere EX) 2.4GHz 40 cores (= 10 × 4)

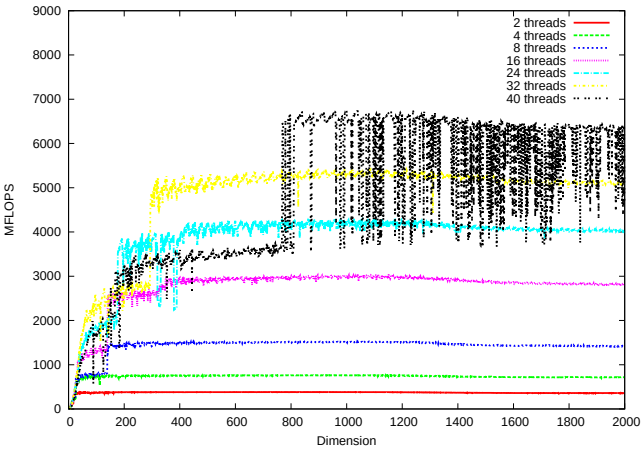

図 3 Rgemm DD performance on Intel Xeon E7-8870 (Westmere EX) 2.4GHz 40 cores (= 10 × 4)

は、のこぎり状のパフォーマンス変化が見られた。これは Westmere EX の場合と同じであったが、24 スレッドを超えた場合には行列のサイズが大きくなった場合に激しい性能の劣化が見られた。最終的には行列の大きさが増えると 16 スレッドで動かした場合の結果とほぼ一致した。最大で 200MFlops 程度得られた。これは Westmer EX よりは悪いパフォーマンスであった。次に double-double 版を見てみよう (図 5)。パフォーマンスの差は見えるが、定性的には GMP 版とほぼ同じ挙動を示した。ピークでは 3.5GFlops であるが、これは Westmere EX のほぼ半分の性能である。

7. まとめ

高精度線形代数演算ライブラリである MPACK を紹介した。MPACK は BLAS, LAPACK をベースとしていて、行列-行列積、ベクトルの内積演算等の処理を行う MBLAS と MBLAS をビルディングブロックとして固有値を求めたり、線形方程式を解いたりする MLAPACK から成る。入手方法、インストール方法、および簡単なプログラムの提示を行った。また、計算性能の測定が重要なので行列-行

© 2012 Information Processing Society of Japan

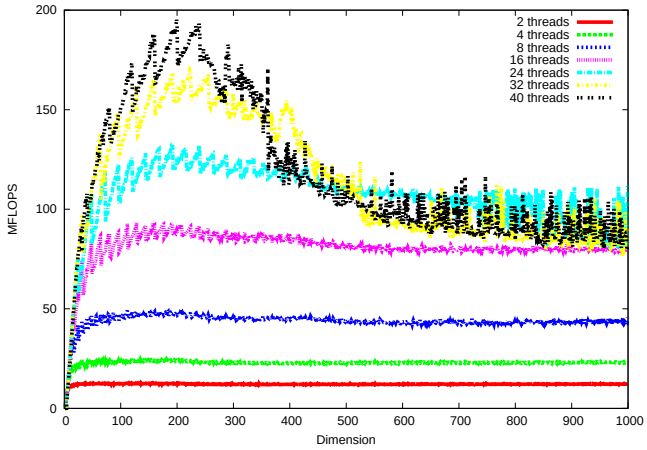

図 4 Rgemm GMP performance on Opteron 6174 (Magny Cours) 2.2GHz 48 cores (= 12 × 4)

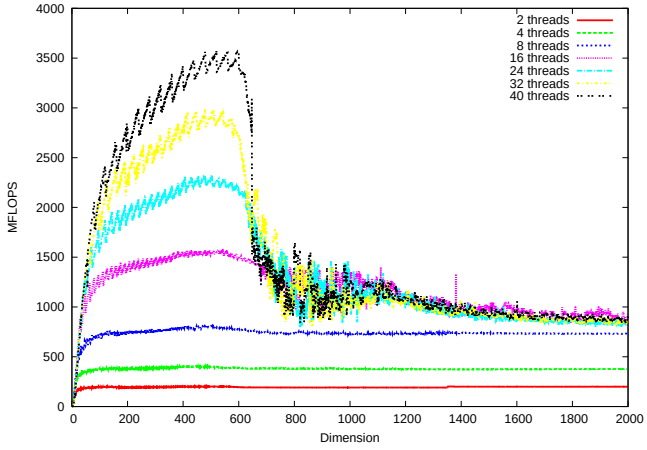

図 5 Rgemm DD performance on Opteron 6174 (Magny Cours) 2.2GHz 48 cores (12 × 4)

列積の性能評価をメニイコアマシン上で行った。今回、ナイーブに OpenMP でマルチスレッド化した行列-行列積の性能評価をメニイコアマシンである、Intel Xeon E7-8870 (Westmere EX) 2.4GHz 40 コアマシン、AMD Opteron 6174 (Magny Cours) 2.2GHz 48 コアマシンで行った。前者では性能がかなりよくスケールしたが、それでも行列のサイズにより性能が大きく変化する場合が見られた。後者では行列のサイズが小さめのところでピークが見られ、その後性能が単調に劣化し、最終的には 16 コア時の程度の性能に収束した。今後は様々な方法で性能の改善を試みたい。

謝辞 メニイコアマシンたちへのアクセスを提供していただいた、中央大学の藤澤克樹先生、および理化学研究所の今村俊幸先生に深く感謝する。

参考文献

[1] Lawson, C. L. Hanson, R. J., Kincaid D., and Krogh F. T, Basic Linear Algebra Subprograms for FORTRAN usage, ACM Transactions on Mathematical Software, 5 308 (1979).

[2] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D., “LAPACK Users’ Guide Third Edition”, SIAM, Philadelphia, 1999.

[3] Nakata, M., “The MPACK (MBLAS/MLAPACK) a multiple precision arithmetic version of BLAS and LAPACK”, http://mplapack.sourceforge.net/, 2008-2012.

[4] IEEE Standard for Binary Floating-point Arithmetic, ANSI/IEEE Standard 754-2008, IEEE, 2008.

[5] Granlund, T., and the GMP development team, “GNU MP: The GNU Multiple Precision Arithmetic Library”, Version 5.0.5, 2012.

[6] Fousse, L., Hanort, G., Lefèvre, V., Pélissier, P., Zimmerman, P., “MPFR: A multiple-precision binary floating-point library with correct rounding”, ACM Transactions on Mathematical Software, 33, 13, (2007).

[7] Enge, A., Gastineau, M., Théveny, P., Zimmermann, P., “mpc — A library for multiprecision complex arithmetic with exact rounding”, INRIA, 1.0, (2012), http://mpc.multiprecision.org/.

[8] Hida, Y. Li, S. X., Baily H. D.. : “Quad-Double Arithmetic: Algorithms, Implementation, and Application”, Technical Report LBNL-46996, Lawrence Berkeley National Laboratory, 2000.

© 2012 Information Processing Society of Japan