12-NA02

高精度行列 - 行列積アルゴリズムにおける並列化手法の開発

片桐孝洋（東京大学）

概要　行列-行列積に代表される基本線形計算を集約したライブラリ BLAS（Basic Linear Algebra Subprograms）は、多くの線形計算で必須の処理である。しかし、従来の数値計算ライブラリは、演算速度は考慮しているが演算精度の考慮が不十分である。本研究では、大石グループで開発された高精度行列-行列演算に 2 種のスレッド並列化と、この 2 種のスレッド並列化に対応できる自動チューニング（AT）機能の実装を行ってきた。本報告では中間報告からの進展として、計算途中で現れる行列の特性に着目し、疎行列演算化を行う実装を提案する。また MPI 化について検討し、高精度行列-行列積における MPI 化の問題となる要因の特定と、その解決法についての検討を行った。

1. 研究の目的と意義

行列-行列積に代表される基本線形計算を集約したライブラリ BLAS（Basic Linear Algebra Subprograms）は、多くの線形計算で必須の処理である。一般に BLAS を含む従来の数値計算ライブラリでは演算速度は考慮しているが、計算結果の正確性の考慮が不十分なことが多い。解の精度保証が重要な課題となっている。一方で、BLAS を用いた汎用的な数値計算ライブラリ（例：LAPACK）において、精度保証をする研究や実装は多くない。

BLAS を精度保証する研究が、早稲田大学の大石教授のグループにより進められている。本研究は大石グループで開発された高精度行列-行列演算（尾崎の方法[1]）を基本とし、その並列化を中心する以下の研究開発を行うことを目的とする。

i. 「疎行列 - 密行列積」、もしくは、「疎行列 - 疎行列」の実装方式の開発
ii. スレッド並列化手法の開発
iii. 分散メモリ型計算機による並列化手法の開発

2. 当拠点公募型共同研究として実施した意義

(1) 共同研究を実施した大学名と研究体制

- 東京大学(代表者：片桐孝洋)
- 芝浦工業大学（副代表者：尾崎克久）
- 東京女子大学（荻田武史）
- 早稲田大学（大石進一）

(2) 共同研究分野

- 精度保証計算、高精度計算
- 高性能計算（HPC）
- 並列処理
- マルチコア・メニーコア最適化
- ソフトウェア自動チューニング（AT）

(3) 当公募型共同研究ならではという事項など

本研究は、精度保証計算を専門とする数値解析分野の数理科学者と、並列化および高性能計算を専門とする高性能計算分野の計算機科学者との協業で成り立つ。従来は、このような境界的な研究テーマに関して、数値解析分野の研究者が並列化を行うことが多かった。しかし近年の計算機アーキテクチャの複雑化により、数値解析分野の研究者による高効率な並列化がますます困難になっている。

一方で、超並列実行により解くべき問題サイズ（行列の次元）は大規模化している。連立一次方程式では、数百万次元の密行列がベンチマークレベルでは解かれているが、このような大規模問題における実問題の解の品質について検証が十分になされていない。近い将来、現状の倍精度演算では要求精度を満たさなくなる可能性は否定できない。これらの要請に答えられるのは、計算機科学分野の研究者では不可能である。数値解析分野の研究者の知識を必要とする。

以上から、高速実行・超並列化技術の確立と、解の精度保証（高精度化）の 2 点が、大規模数値計算には必要であると結論付けられる。本研究は、

適用対象が行列 - 行列積に限定はされてはいるが、一部の要請に答えることが可能である。この点が、当公募型共同研究ならではの特徴である。

3. 研究成果の詳細と当初計画の達成状況

(1) 研究成果の詳細について

概要

本研究で採用する高精度な行列-行列積アルゴリズムは、尾崎らが開発したアルゴリズム[1](以降、**尾崎の方法**）を用いる。尾崎の方法は、入力行列に対して以下の無誤差変換を行う：

I)行列 A と行列 B を下記のように分解する（インデックスが若いほうが高いビットを持つようにする）

$$A = A^{(1)} + A^{(2)} + A^{(3)} + \ldots + A^{(p)}$$

$$B = B^{(1)} + B^{(2)} + B^{(3)} + \ldots + B^{(q)}$$

II)行列積 AB を以下のように計算する（ pq 個の行列積となる）

$$AB = (A^{(1)} + A^{(2)} + \ldots + A^{(p)})(B^{(1)} + B^{(2)} + \ldots + B^{(q)})$$

$$= (A^{(1)} + A^{(2)} + \ldots + A^{(p)})(B^{(1)} + B^{(2)} + \ldots + B^{(q)})$$

$$= A^{(1)} B^{(1)} + A^{(1)} B^{(2)} + A^{(2)} B^{(1)} + \ldots + A^{(p)} B^{(q)} \quad \ldots(1)$$

処理 I）の分解の仕方を工夫することで、処理 II)における行列積を無誤差の演算にすることができる。行列サイズが大きくなるに従い、ほとんどの演算時間は行列積処理部分となる。

また、分解数である p、q を限定すれば、部分的な多倍長化演算と同様の効果を得ることができる。したがって、高精度化を図ることが可能である。

処理 I)の分解の過程で、入力行列の要素値のレンジに依存し疎行列が生成される。このため密行列を入力としても、分解の過程で疎度が大きくなる場合は、密行列から疎行列化したほうがよい。すなわち、「疎行列 - 密行列積」、および、「疎行列 - 疎行列積」の演算に切り替えるほうが、全体の計算量（実行時間）の縮減が期待できる。

以下の前提を置く。

- LU 分解や QR 分解の精度保証への適用については、荻田が開発したアルゴリズムを利用する。
- 既存の数値計算ライブラリへの適用は LAPACK などのフリーウェアを利用する。
- 「疎行列 - 密行列積ルーチン」ではないが、類似の機能である「疎行列 - ベクトル積」の並列化に関し、代表者の片桐が行った高スレッド並列化の研究がある。この研究では、自動チューニング機能付き数値計算ライブラリ Xabclib へ開発ルーチンを実装した。開発コードの実用ライブラリへの展開技術も有している。

以上の経験を基に、高精度行列-行列積アルゴリズムの開発を行う。数値計算ライブラリ全体でみると、単に実行速度のチューニングだけではなく、演算精度を考慮したチューニングも可能となる点にも新規性確保が期待される。

尾崎の高精度行列 - 行列積アルゴリズム

尾崎の方法では、図１が主演算となす。

```
Function F = EFT_Mul(A, B)

  [A, n_A] := Split_A; [B, n_B] := Split_B; k := 1;

  for i =1: n_A

    for j =1: n_B

      F{ k } := A{ i } * B{ j };    k := k + 1;
  end;  end;  end
```

図１

$$AB = \sum_{k=1}^{n_A \cdot n_B} F^{(k)}$$

の変形部分

ここで図１の F の和には、faithful と呼ばれる結果を得るアルゴリズムを用いている。そのため演算結果は、ほぼ最良になることが知られている。

図２に行列 A の分解部分の詳細を載せる。

図１、図２で用いられている表記法について、以下にまとめる。

- 表記の説明

fl（・）：　最近点への丸めで計算

F：浮動小数点数の集合

A : サイズが m 行 n 列の行列

B : サイズが n 行 p 列の行列

u : the unit round off

2^{-24}: IEEE 754 binary32 (single precision)

2^{-53} : IEEE 754 binary64 (double precision)

```
Function [D, n_A] = Split_A(A)

    n_A := 0; n := size(A, 2)

    while (norm(A, 1)~=0)

        n_A := n_A + 1;

        μ := max(abs(A), [ ], 2);

        τ := 2.^ceil ((log2(u^-1) + log2(n +1))/2);

        t := 2.^ceil((log2(μ)* τ;

        δ_A^(nA) := repmat(t, 1, q);

        A{n_A} := fl((A + δ_A^(nA)) - δ_A^(nA));

        A := fl(A - A{n_A});

    end;  end
```

図２　行列Aの分解部分の詳細

疎行列演算化の実装提案

尾崎の方法では、行列 A、B を高精度計算する際、その行列要素に応じて行列を分解する（図 2 の処理を行う）ことはすでに述べた。この図 2 の演算の際に、行列要素値のばらつきが大きいとき、分解の最終段階においては、行列が疎行列ーすなわち零要素の多い行列ーとなる。

一方、主演算は図 1 の処理であり、通常は BLAS を用いた密行列の行列-行列積(dgemm)で実装される。つまり、疎行列になる状況でも、疎行列の特徴を利用して演算量を減らす工夫がされていない。

以上の理由から、零要素の数がある値以上になるとき、図 1 の主演算を、密行列積から疎行列積に切り替える実装手法が有効になることが予想される。

以上の疎行列演算化に対して、実装を考慮し、いくつか検討すべき課題がある。それを以下に記載する。

- 入力行列は密行列のため、疎行列演算化のために、疎行列用のデータ形式（疎行列データ形式）に、実行時に変換する必要がある。この変換時間も、演算時間に含まれる。
- 疎行列演算の実装方法は複数存在する。ここでは簡単のため、A を疎行列とし、B については、実際は疎行列でも密行列として取り扱う。このことで、疎行列-ベクトル積(Sparse Matrix-Vector multiplication, SpMV)に帰着でき、代表者の先行研究の知見が活用できる。
- SpMV の実装方式については多種存在する。疎行列データ形式の種類に加え、SpMV 実装方式、およびスレッド並列化の組合せがある。これらを考慮する必要がある。

本研究では、以下の実装方式を選択した。その理由も記載する。

- **疎行列データ形式**

  CRS(Compressed Row Storage)形式を採用する。この理由は、高精度行列-行列積で現れる疎行列については、入力行列の非ゼロ要素の分布について何の仮定も置くことが出来ないからである。この結果、疎行列はランダムスパース行列であると仮定する。ランダムスパース行列の場合、実装により高速化が期待できる ELL 形式などの疎行列データ構造を採用しても、CRS 形式よりも高速化する見込みがない。また、COO(Coordinate)形式を採用すると SpMV に間接参照が 2 回生じ、CRS 形式による SpMV 間接参照 1 回に対して、高速化できる見込みがあまりない。このことから、CRS 形式は性能的にみて悪くない形式といえる。

- **SpMV の実装方式**

  (1) CRS ループにおける最外ループのスレッド並列化する方法；(2) SpMV は逐次処理にするが、高精度行列-行列積のループにおけるスレッド並列性を利用する方法；(3) 複数ベク

トルを同時に SpMV する方法で最外ループをスレッド並列化する方法；の３通りを採用する。この理由は、(1)は従来よく使われる並列処理の形式であること、(2) は (1) に対しメモリを必要とするが、(1) に対し効率のよい並列化が期待できること、および、(3) は (1) (2) と演算式が異なるように実装できるため、逐次最適化の効果がより高くなることが期待できること、からである。

疎行列演算の切り替えのための前処理

提案実装方式では、事前に行列 A のゼロ要素の値を調べる。この調査は、図 2 の行列分解の処理中に実装可能である。この時、以下の２つを調べる：(1) 全体の要素数に対する値 0 の要素の割合（疎度[%]）；(2) 零行列かどうか；

以上の情報をフラグ配列 iAsp[]に保存する。このとき、調べた疎度があらかじめ指定した値以上である場合は“1”をセットする。また、零行列である場合は“2”をセットする。

SpMV の実装方法の詳細

- **単純な外側ループのスレッド並列化(SpMV(内部並列))**

この実装における、密行列演算と疎行列演算の切り替え部分は、以下のコードになる。

```
for (i=0;i<Ak;i++) {
  if (iAsp[i] == 0) {
    for (j=0;j<Bk;j++) {
      dgemm の呼び出し;
    }
  } else {
  if (iAsp[i] != 2) {
    // Set value to Sparse Matrix
    SetMat(val, irp, icol, A+i*m*n, m, n);
    for (j=0;j<Bk;j++) {
      for (ii=0; ii<p; ii++) {
        // Set RHS vector x
        SetVec(x, B+j*n*p+ii, n, p);
       // call SpMV main routine
        spmv1(y, val, irp, icol, x, m);
        //Store the vector y
     StoreVec(C+m*p*(j+Bk*i)+ii*p, y, m, p);
      }
    } }
}
```

SpMV 部分は以下のコードになる。

```
#pragma omp parallel for private(s,j_ptr, i)
for (i=0; i<n; i++) {
  s = 0.0;
  for (j_ptr=irp[i]; j_ptr<irp[i+1]; j_ptr++)
{
    s = s + val[j_ptr] * x[icol[j_ptr]];
  }
  y[i] = s;
}
```

- **問題特有のループ並列化を利用するスレッド並列化 (SpMV (外部並列))**

この実装では、SpMV 部分は逐次のものを使う。この SpMV コードは、上記の spmv1 で OpenMP 記述を無くしたものと等価なため説明を省略する。演算の概要を以下に載せる。

```
for (i=0;i<Ak;i++) {
  if (iAsp[i] == 0) {
    for (j=0;j<Bk;j++) {
          dgemm の呼び出し;
    }
  } else {
    if (iAsp[i] != 2) {
      // Set value to Sparse Matrix
      SetMat(val, irp, icol, A+i*m*n, m, n);
      for (j=0;j<Bk;j++) {
#pragma omp parallel for
        for (ii=0; ii<p; ii++) {
          // Set RHS vector x
          SetVec(&x[ii*p], B+j*n*p+ii, n, p);
          // call SpMV main routine
          spmv2(&y[ii*p], val, irp, icol,
              &x[ii*p], m);
```

```
        //Store the vector y
        StoreVec(C+m*p*(j+Bk*i)+ii*p,
           &y[ii*p], m, p);
      }
    } } }
```

- **複数ベクトルを同時に SpMV して最外ループをスレッド並列化（SpMV（複数右辺同時））**

この実装における、密行列演算と疎行列演算の切り替え部分は、以下のコードになる。

```
for (i=0;i<Ak;i++) {
  if (iAsp[i] == 0) {
    for (j=0;j<Bk;j++) {
          dgemm の呼び出し;
    }
  } else {
    if (iAsp[i] != 2) {
      SetMat(val, irp, icol, A+i*m*n, m, n);
      for (j=0;j<Bk;j++) {
       spmvM(y, p, val, irp, icol, B+j*n*p, m);
#pragma omp parallel for private(jj)
       for (ii=0; ii<m; ii++) {
         for (jj=0; jj<p; jj++) {
           C[m*p*(j+Bk*i)+ii*p+jj] =
               y[ii+jj*p];
         }
       }
    } } } }
```

以上の場合、m 本の複数右辺を同時に SpMV するルーチンの spmvM は、以下の実装となる。

```
#pragma omp parallel for private(j_ptr, i, ib, iloc)
for (i=0; i<n; i++) {
  for (ib=0; ib<mb; ib++) {
    y[i*mb+ib] = 0.0;
  }
  for (j_ptr=irp[i]; j_ptr<irp[i+1]; j_ptr++)
{
    iloc = icol[j_ptr]*mb;
    for (ib=0; ib<mb; ib++) {
      y[i*mb+ib] = y[i*mb+ib] +
         val[j_ptr] * x[iloc+ib];
    } } }
```

試験行列

試験行列は以下である.

- *A, B*の要素を pow(10,rand()%Φ) で生成

Φの値が大きいと、行列要素の値の分散が大きくなる。この場合、尾崎の方法による行列分割の回数が増え、行列 - 行列積の演算回数も増える。総合的な演算量（演算時間）が増加するので、並列処理の効果に影響する。

ここでは、さらに以下の条件をつける。

- 行列 *A* について、ほとんどすべての行列要素を Φ=1 で生成し、全体要素の 1%ほどの要素について、乱数で位置を決め、行列要素を Φ=200 で生成する。

すなわち行列 A の分解が進むにつれ、全体の約 99%程度が零要素となる疎行列が生成されると期待される例である。

性能評価環境

以下の表１に示すハードウェアの１ノードを利用し、性能評価を行う。

表１　性能評価に用いたハードウェアの概要

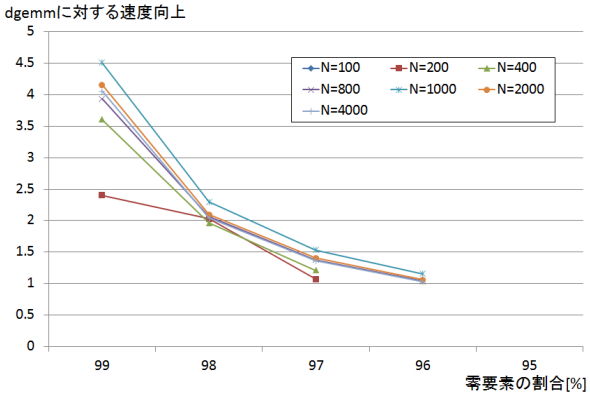
(A) FX10

計算機名	ハードウェア		
	総合ノード数	*総 FLOPS*	
富士通 PRIME HPC FX10 (FX10)	4800	1.135 PFLOPS	
	理論ピーク (ノード)	*コア数 (ノード)*	*主メモリ (ノード)*
	236.5 Gflops	16	32 GB
	アーキテクチャ (コア)	*周波数 (コア)*	*理論ピーク (コア)*
	SPARC64 IXfx	1.848 GHz	14.78 Gflops
	ソフトウェア		
	コンパイラ	*オプション*	
	富士通 C コンパイラ	-Kfast, -openmp	

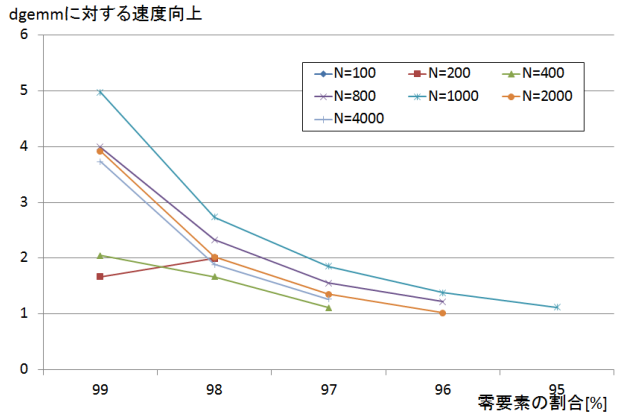
計算機名	ハードウェア	
	総合ノード数	*総 FLOPS*
	Version 1.2.1	
	BLAS ライブラリ	
	Scientific Subroutine Library II (SSL II) (富士通が最適化済み)	

疎行列切り替えのための「疎度」の見積もり

ここでは、FX10 において富士通社が最適化した BLAS に対して SpMV 演算のほうが高速となる疎度について調査した。その結果を図 3 に示す。



(a) 1 スレッドの場合



(b) 16 スレッドの場合

図 3 FX10 における dgemm より高速となる SpMV 演算における疎度

図 3 (a) (b) より、零要素の占める割合が約 97%以上ないと、FX10 では疎行列化の効果がない。また、N=100 以下の行列では効果がない。1 回の行列−行列積演算あたり、最大で約 5 倍の速度向上が期待できる。特に、N=1000 の時に効果が大きい。ここで得られた知見は、AT 方式として実装することで、疎行列化の高精度行列 - 行列積のライブラリに実装可能である。

なおこの評価では、密行列から疎行列へ変換する時間は入っていない。したがって、この速度向上率は上限値といえる。

高精度行列−行列積処理への疎行列化の適用

行列サイズを N=1000 に固定し、疎行列への切り替えの疎度を 97%に設定したうえで、性能評価を行った。

この試験行列では、行列 A の分解後の行列は 29 個になった。また、零要素が占める割合が 97%以上の行列は 20 個（零行列を除く。零行列は 2 個。）であった。図 4 に、密行列演算のみによる高精度行列−行列積(dgemm)の実行時間に対する速度向上率を載せる。

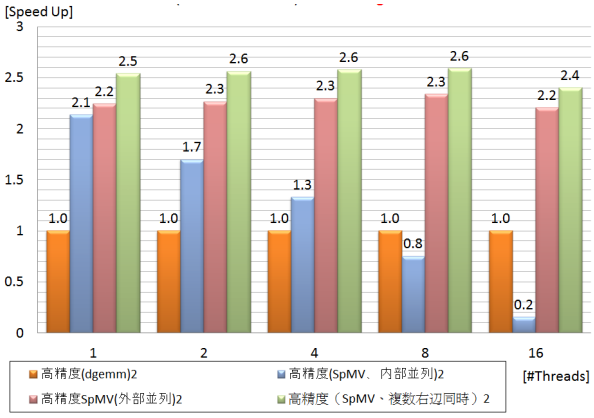


図 4 疎行列化の効果（主演算部分）

図 4 から疎行列化を行うことで、疎行列データ形式へのデータ変換時間を含めた速度向上が、約 2.1 倍〜約 2.6 倍期待できることがわかる。

スレッド数が増えるにつれ、単純なスレッド並列化である SpMV（内部並列）は 8 スレッドを超えると、dgemm 呼び出しのみによる高精度行列−行列積に対して速度向上が期待できない。この理由は、単純な並列化では疎行列の行単位の並列性を利用するが、ランダムスパース行列の場合、行あたりの非ゼロ要素数のばらつきが多いと想定され、負荷バランスが悪くなることがあげられる。一方、SpMV（外部並列）はベクトル数の並列性（この場合は 1000 個）を利用するため、各疎行列演算についての演算数が一定となる。そのため、負荷バランスは SpMV（内部並列）に比べて良い。一方、

SpMV(複数右辺同時)が最も性能が良いが、これは複数右辺を同時に SpMV すると、CRS 形式による間接参照が最内ループ内で反復ごとに生じないことによる。このため、逐次最適化の効果がより良いと予想される。

以上から高精度行列-行列積の疎行列化においては、SpMV（複数右辺同時）の実装を採用するのが良いと結論付けられる。

参考文献

[1] K. Ozaki, T. Ogita, S. Oishi, S. M. Rump: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numerical Algorithms, Springer, 2011.
(DOI: 10.1007/s11075-011-9478-1)

(2) 当初計画の達成状況について

本提案では研究進捗を以下の３フェーズに分け、研究の意義を達成することを目的にしていた。

- フェーズ１（2012 年 4 月～7 月）:「疎行列 - 密行列積」の逐次およびスレッド並列化の実装を行う。
- フェーズ２（2012 年 7 月～12 月）: フェーズ１の結果を基に、「疎行列 - 疎行列積」の逐次およびスレッド並列化の実装を行う。
- フェーズ３(2013 年 1 月～3 月):フェーズ１、２での結果を基に、性能パラメタを抽出する。得られた性能パラメタをもちいてチューニング効果の検証をする。その情報をもとに、自動性能チューニング方式を研究開発し予備評価を行う。

  MPI による分散メモリ計算機での並列化に向け、アルゴリズム設計、および、高ノード数（10,000 コア超実行）程度での通信時間評価などプロトタイピングを行う。

本研究においては、フェーズ１、２の目的を達成できた。フェーズ３においては、疎度の導入による AT 化の目途が立っているため、この部分は完了している。

一方、平成２４年度ではプロトタイピングまで至らなかったが、MPI 化の検討を行い、分散並列化の目途を付けることができた。なお MPI 化については、拠点終了後も独自の共同研究として進めていく。

4. 今後の展望

- **疎行列化の展望**

  本研究により、疎行列化のプロトタイピングを行い、疎行列化が有効であることを示した。今後の展望として、複数右辺同時の SpMV を、汎用ライブラリとして AT 機能を含めて実装し、本研究のアプリケーションに適用することがあげられる。

  密行列のみを用いている従来の高精度行列-行列積ではメモリを多く消費する。そのため、全ての行列を疎行列として扱い、状況に応じて、密行列化をすることで、メモリ量の削減と高速化の双方を達成する方式の研究も興味深い。このとき、密行列化の変換コストを考慮した実装方式を新規開発することが、技術的な挑戦事項になるだろう。

- **MPI 化（分散メモリ並列化）の展望**

  本報告では紙面の都合から説明できなかったが、MPI 化の検討を行った。その要点を以下に記載する。

  密行列-行列演算部分を、ScaLAPACK に置き換えることで、分散並列化するアプローチがある。この方法では、分散並列化の実装はライブラリを使うので容易である。しかし、小さい行列サイズによるブロックサイクリック-サイクリック分散による通信量の増加が懸念される。

  一方、本問題においては、行列-行列レベルで分散並列化せず（すなわち、行列要素を分散せず、行列全体の単位で分散する)、行列 A、B の分解行列レベルで分散並列化する方法がより自然な並列化となる。この方法の欠点は、行列 C を得るときの、「高精度行列和」の呼び出し部分に並列性が存在せず、かつ、特定の

1 プロセスに分散データを集積するため（たとえば、MPI_GatherV 関数を使う集団通信が必要)、通信時間がプロセス数を増やすと増加し台数効果を制限する。そこで本研究で検討した設計では、高精度和の内部演算レベルの並列性を利用し、並列リダクションレベルの並列性ーすなわち、P プロセスで Log(P)の並列性を抽出する方法ーが理論的に実現可能であることを発見した。残念ながらこの方法のプロトタイピングは課題終了時には未完である。しかし、H25 年度以降、独自の共同研究として継続する予定である。

5. 研究成果リスト

(1) 学術論文（投稿中のものは「投稿中」と明記）

1. K. Ozaki, T. Ogita, S. Oishi, S. M. Rump: Generalization of Error-Free Transformation for Matrix Multiplication and its Application, Nonlinear Theory and its Applications, IEICE, Vol. 4:1 (2013), pp. 2-11.
2. K.Ozaki, T.Ogita, S.Oishi, S.M.Rump: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numerical Algorithms, Vol.59:1 (2012), pp. 95-118.

(2) 国際会議プロシーディングス

3. K.Ozaki, T.Ogita: Application of Error-Free Transformation for Matrix Multiplication, Fifth Conference on Numerical Analysis and Applications, Lozenetz, Bulgaria (2012/06/15-20), pp. 34.
4. K.Ozaki, T.Ogita, S.Oishi: Memory Reduced Implementation of Error- Free Transformation of Matrix Multiplication and its Performance, 2012 International Symposium on Nonlinear Theory and its Applications (NOLTA2012), Palma de Majorca (2012/10/22-26), pp.877-880.
5. K.Ozaki, T.Ogita: Performance Comparison of Accurate Matrix Multiplication, 15th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN 2012), the Russian Academy of Sciences, Novosibirsk, Russia (2012/09/23-29), pp.129-130.

(3) 国際会議発表:なし

(4) 国内会議発表

6. 片桐孝洋：ppOpen-AT：ポストペタスケール時代の数値シミュレーション基盤ソフトウェア ppOpen-HPC のための自動チューニング基盤、京都大学数理解析研究所研究集会「科学技術計算における理論と応用の新展開」、京都大学数理解析研究所講究録 1791、(2012)、pp.107-111.
7. 片桐孝洋、尾崎克久、荻田武史、大石進一：高精度行列 - 行列積アルゴリズムのスレッド並列化と ABCLibScript への機能実装、第 133 回 HPC 研究会、情報処理学会研究報告 HPC-133 (2012/03/26-27).
8. 尾崎克久、荻田武史：BLAS を用いた高精度な行列積アルゴリズムの使用メモリ量の削減とその性能について、京都大学数理解析研究所講究録、1791、(2012)、pp. 66-75.
9. 片桐孝洋、尾崎克久、荻田武史、大石進一：高精度行列-行列積アルゴリズムの疎行列演算化による高速化、2013 年並列／分散／協調処理に関する『北九州』サマー・ワークショップ (SWoPP 北九州 2013)、日本応用数理学会「行列・固有値問題の解法とその応用」研究部会、(2013) (発表予定)

(5) その他（特許，プレス発表，著書等）

- ppOpen-AT ver.0.1 版公開
  - http://ppopenhpc.cc.u-tokyo.ac.jp/
  - 尾崎の方法の AT が実現可能な言語 ABCLibScript の後続プロジェクトによるコードの整備と公開。