HITACHI SR16000/M1 チューニング連載講座

６．数値計算ライブラリの利用

片桐　孝洋

東京大学情報基盤センター　准教授

１．はじめに

本稿では、HITACHI SR16000/M1（以降、SR16K と記載）で利用できる数値計算ライブラリについて紹介します。

SR16K では、数値計算ライブラリとして MATRIX/MPP、MATRIX/MPP/SSS、MSL2、BLAS、LAPACK、ScaLAPACK、Parallel NetCDF、FFTW、SuperLU、SuperLU_DIST、ESSL、Parallel ESSL、STL(Standard Template Library)、Boost C++ がインストールされています。

これらのライブラリを利用するには、コンパイラのオプションとして、以下を指定すればよいようになっています。

-L<ライブラリ検索パス名>

-l<ライブラリ名>

-l オプションは、プログラムファイル名の後ろに指定します。このオプションは左から順に処理されるので、内部で利用されている関数の呼び出し順を考慮し、ライブラリ名を指定する順序には注意して下さい。なお、センター提供の数値計算ライブラリのうち、MATRIX/MPP、MATRIX/MPP/SSS、MSL2、ESSL、Parallel ESSL の検索パスは、標準で設定されていますので、**-L** オプションは省略できます。

本稿では以上のライブラリのうち、BLAS、LAPACK、および ScaLAPACK の利用法を中心に、簡単な性能評価結果を報告します。

２．BLAS の性能

２．１ 概要

BLAS とは、Basic Linear Algebra Subprograms の略で、基本線形代数副プログラム集のことです。線形代数計算で用いられる、基本演算を標準化（Application Programming Interface (API)化）したものです。

BLAS は、**密行列用**の線形代数計算用の基本演算の副プログラムを指します。疎行列用の BLAS は、**スパース BLAS** というものがあります。しかし通常 BLAS というと、密行列の演算ルーチンであることに注意してください。

BLAS では以下のように分類し、サブルーチンの命名規則を統一しています。

1. 演算対象のベクトルや行列の型（整数型、実数型、複素型）
2. 行列形状（対称行列、三重対角行列）
3. データ格納形式（帯行列を二次元に圧縮）
4. 演算結果が何か（行列、ベクトル）

また、演算性能から、以下の３つに演算を分類しています。

1. **レベル１ BLAS**： ベクトルとベクトルの演算
2. **レベル２ BLAS**： 行列とベクトルの演算
3. **レベル３ BLAS**： 行列と行列の演算

２．２ 各 BLAS の性能

レベル１ BLAS の特性について解説します。

- ベクトル内積、ベクトル定数倍の加算、などです。
  - 例：$y \leftarrow \alpha\ x + y$
- データの読み出し回数、演算回数がほぼ同じです。
- データの再利用（キャッシュに乗ったデータの再利用によるデータアクセス時間の短縮）がほとんどできません。実装による性能向上が、あまり期待できません。ほとんど、計算機ハードウエアの演算性能となります。
- レベル１BLAS のみで演算を実装すると、演算が本来持っているデータ再利用性がなくなることがあります。
  - 例：行列-ベクトル積を、レベル１BLAS で実装するなど。

レベル２ BLAS の特性について解説します。

- 行列-ベクトル積などの演算です。
  - 例： $y \leftarrow \alpha\ A\ x + \beta\ y$
- 前進/後退代入演算、$T\ x = y$ （Tは三角行列）を xについて解く演算、を含みます。
- レベル１BLAS のみの実装よる、データ再利用性の喪失を回避する目的で提案されました。
- 行列とベクトルデータに対して、データの再利用性があります。データアクセス時間を、実装法により短縮可能です。実装法により性能向上がレベル１BLAS に比べしやすいですが、十分ではないです。

レベル３ BLAS の特性について解説します。

- 行列-行列積などの演算です。
  - 例：$C \leftarrow \alpha\ A\ B + \beta\ C$
- 共有記憶型の並列ベクトル計算機では、レベル２ BLAS でも性能向上が達成できませんでした。並列化により 1CPU 当たりのデータ量が減少するためです。より大規模な演算をとり扱わないと、再利用の効果が出ません。
- 行列-行列積では、行列データが $O(n^2)$に対し、演算は $O(n^3)$なので、データ再利用性が原理的に高いです。行列積は、アルゴリズムレベルでもブロック化でき、さらにデータの局所性を高めることができるので、最適化の効果が高いです。

各 BLAS の性能挙動は、図１のようになるのが普通です。

図１より、可能であればレベル３ BLAS3 演算を用いる演算にするほうが、理論ピーク性能に対する実効効率の観点から、高い性能を保つことが出来ます。

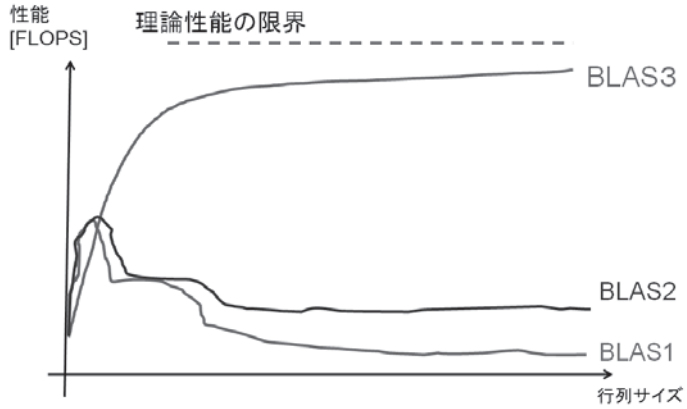

図 1　各 BLAS の性能挙動について

２．３ SR16K におけるレベル 3 BLAS の性能

２．３．１ 逐次性能

以降に、SR16K のレベル 3BLAS の性能を紹介します。

東大の SR16K では、高性能な BLAS を利用するためには、IBM 社の ESSL(Engineering and Scientific Subroutine Library)ライブラリが提供している BLAS ライブラリを利用します。

以下に、日立最適化 Fortran90 コンパイラから、ESSL の BLAS を利用する場合の利用例（逐次（スカラ）用）のコンパイル例を示します。

■ESSL ライブラリの BLAS を使う場合のコンパイル例(Fortran90 言語、逐次版)

```
f90 -o mat-mat-blas -i,L -opt=ss -noparallel mat-mat-blas.f -lessl
```

レベル 3 BLAS の呼び出しで行列 - 行列積演算の場合は、実行環境に依存せず以下のような実装になります。

■レベル 3 BLAS の呼び出し例(Fortran 言語)

```
double precision ALPHA,BETA
ALPHA=1.0d0
BETA=1.0d0
CALL DGEMM('N', 'N', n, n, n, ALPHA, A, n, B, n, BETA, C, n)
```

以上のレベル 3 BLAS の性能と、手書きした 3 重ループによる行列 - 行列積の性能の[GFLOPS]値を、N=100 から 3000 まで、100 刻みで調べたものを図 2 に示します。

なお、対象ルーチンは 10 回連続で呼び出し、1 回当たりの平均時間から GFLOPS 値を算出しています。したがって、配列データをキャッシュに載せた上での性能計測（**ホット・キャッシュ性能**）である点に注意してください。

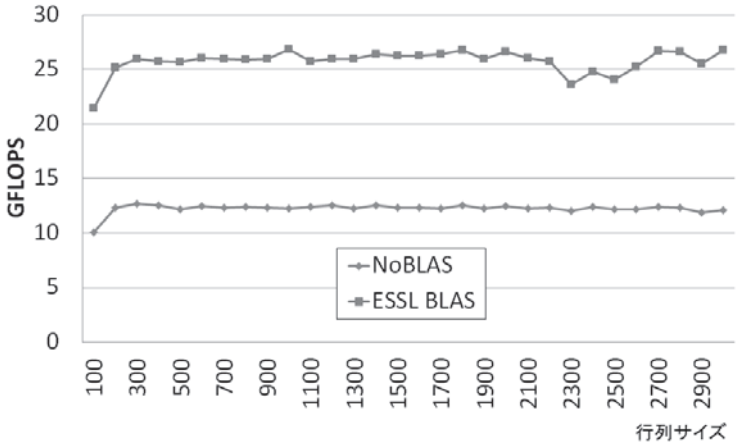

図 2　SR16K の ESSL の BLAS ライブラリの性能（逐次、1 コア）

SR16K の 1 ノードあたりのキャッシュ構成は、L1 が 32KB/コア、L2 が 256KB/コア、L3 が 32MB/8 コア共有、となっています[1]。たとえば、1000 次元の行列では、倍精度演算のとき、配列 A の容量は 8MB になります。したがって、配列 B、配列 C を考慮しても合計 32MB です。ですので、1000 次元までの行列は L3 キャッシュにのってしまいます。

図 2 からこの状況では、ESSL ライブラリを用いると、手書きのループに対して約 2.1 倍高速化されます[1]。また、SR16K の 1 コアの理論性能は 30.64 GFLOPS です。したがって、この実行条件での ESSL ライブラリの性能は、26.8/30.64 なので約 87.4%の実効効率（理論ピーク性能に対する効率）といえます。

２．３．２ スレッド性能

次に、スレッド並列時の BLAS 3 性能を紹介します。

以下に、日立最適化 Fortran90 コンパイラから、ESSL の BLAS を利用する場合の利用例（スレッド並列用の例）のコンパイル例を示します。

■ESSL ライブラリの BLAS を使う場合のコンパイル例(Fortran90 言語、スレッド並列版)

```
f90 -o mat-mat-blas -i,L -opt=ss -parallel mat-mat-blas.f -lesslsmp
```

BLAS ライブラリのスレッド数実行数を設定するためには、OpenMP のスレッド数の指定と同様にします。すなわち、環境変数 OMP_NUM_THREADS に実行したいスレッド数を代入して実行します。BLAS の呼び出しは、逐次コードから変更はありません。

日立コンパイラからの呼び出しのため、日立コンパイラによる自動並列化のスレッド数を 1 にする必要があります。もし、日立コンパイラのスレッド数が 1 でない場合や、特にスレッド数を指定しない場合は、日立コンパイラの生成するスレッド実行に加えて、このスレッドから派生した BLAS が、BLAS 内でさらにスレッド処理を行うために、実行性能が劇的に低下してしまいます。

[1]大規模サイズの実行では、手書き実装と ESSL の実行の性能差にかなり違いが出ることが予想されますので注意してください。

具体的には、8 スレッド実行の場合は、以下のようにします。

■日立コンパイラから ESSL の BLAS を使う場合のスレッド数指定の例

```
export HF_PRUNST_THREADNUM=1
export OMP_NUM_THREADS=8
./mat-mat-blas
```

SR16K では、1 ノードあたり 32 コアあります。したがって 1 ノード当たりの理論性能は 30.64*32=980.48 GFLOPS となります。

ESSL のスレッド並列化されたレベル 3 BLAS 行列 - 行列積の性能[GFLOPS]を、N=100 から 3000 まで、100 刻みで調べたものを図 3 に示します。

なお、対象ルーチンは 10 回連続で呼び出し、1 回当たりの平均時間から GFLOPS 値を算出しています。逐次と同じく、ホット・キャッシュ性能である点に注意してください。

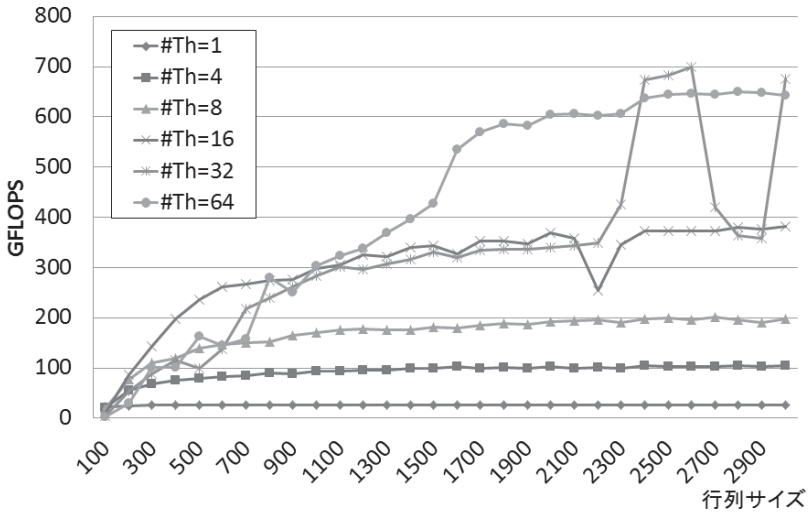

図 3 SR16K の ESSL ライブラリの BLAS の性能（スレッド並列版、1 スレッド〜64 スレッド)。64 スレッド実行は SMT 実行。1 コアあたり 2 スレッドが割り当てられる。

図 3 より、スレッド数が増加するにしたがい、飽和性能を得られるまでの行列サイズが増加します。たとえば、1 スレッド実行では N=100 でも飽和性能が得られていますが、64 スレッド実行になると、N=2500 ぐらいまでは飽和性能になりません。したがって、高スレッド数で実行する時には、高性能を得るために大きな問題サイズで BLAS を実行しないといけません。

図 3 では、32 スレッドの性能挙動が不安定です。この原因の一つは、スレッドの割り当てを OS 依存にしていることと、配列のローカルメモリ上への割り当てがデフォルトのままになっていることがあります。これらの NUMA 最適化は、一般に日立コンパイラのオプションで制御できます[1]。ESSL ライブラリは日立コンパイラでコンパイルされていないため、日立コンパイラのオプション指定を行うとスレッド並列化ができなくなります。AIX の制御コマンドで直接 NUMA 最適化を行えば、性能が安定化する可能性があります。

64 スレッド実行時は SMT 実行です。1 コア当たり 2 スレッドが割り当てられています。64 スレッド実行の時、約 650GFLOPS を達成しています。したがって理論ピーク性能に対する効率は 650/980.48 なので約 66.2%です。

逐次実行と同様に、大規模な問題サイズでの実行を行うと、逐次性能とスレット性能ともに改善がされる可能性があります。

２．４ C 言語からのレベル 3 BLAS の呼び出しについて

日立最適化 C コンパイラから、ESSL の BLAS を利用する場合の利用例（逐次実行）のコンパイル例を示します。

■ESSL ライブラリの BLAS を使う場合のコンパイル例(C 言語、逐次版)

```
cc -o mat-mat-blas -Os mat-mat-blas.c -lessl
```

■レベル 3 BLAS の呼び出し例(C 言語)

```
double ALPHA, BETA;
char TEX[1] = {'N'};
ALPHA=1.0;
BETA=1.0;
dgemm(&TEX, &TEX, &n, &n, &n, &ALPHA, A, &n,  B, &n, &BETA, C, &n);
```

以上から、C 言語からでも Fortran 言語に類似した方法でコンパイルと実行ができます。以降、Fortran 言語での実行例を紹介します。

３．LAPACK の性能

３．１ 概要

LAPACK は、密行列に対する連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンを集めたライブラリです[2]。

LAPACK は、主にレベル 3 BLAS を用いてアルゴリズムを構築したものです。スレッド並列化のみ対応しています。分散並列化（MPI 並列化）は、ライブラリ設計の原理上、対応しておりません。

スレッド並列化の方法は、レベル 3 BLAS のスレッド並列版を使うことで、スレッド並列化をしています。したがって、高スレッド実行時にレベル 3 BLAS の性能が劣化する場合は、LAPACK においてもスレッド並列化の性能が劣化します。

特に、16 スレッドを超えるスレッド実行で注意が必要です。この場合、分散並列版の ScaLAPACK を利用した方が、1 ノード内実行においても並列実行効率が高いことがあります。ですので、場合によりライブラリを使い分ける必要があります。

３．２ LAPACK の API の一例

LAPACK の命名規則は、以下のようになっています。

[2] http://www.netlib.org/lapack/

関数名：XYYZZZ

- X：データ型
  S:単精度、D：倍精度、C：複素、Z：倍精度複素
- YY：行列の型
  BD: 二重対角、DI：対角、GB：一般帯行列、GE：一般行列、HE:複素エルミート、HP：複素エルミート圧縮形式、SY：対称行列、など。
- ZZZ：計算の種類
  TRF:行列の分解、TRS：行列の分解を使う、CON：条件数の計算、RFS：計算解の誤差範囲を計算、TRI：三重対角行列の分解、EQU：スケーリングの計算、など。

LAPACK には多数の関数があります。ここでは、連立一次方程式の求解のための関数である DGESV を紹介します。

API 例：DGESV

- **DGESV(N, NRHS, A, LDA, IPIVOT, B, LDB, INFO)**
  - *A X = B* の解の行列 *X* を計算する。
  - *A * X = B*、ここで *A* は *N×N* 行列で、 *X* と *B* は *N×NRHS* 行列とする。
  - 行交換の部分枢軸選択付きの LU 分解 で *A* を *A = P * L * U* と分解する。ここで、*P* は交換行列、*L* は下三角行列、*U* は上三角行列である。
  - 分解された *A* は、連立一次方程式 *A * X = B* を解くのに使われる。
- 引数
  - N（入力）- INTEGER
    - 線形方程式の数。行列 *A* の次元数。 N >= 0。
  - NRHS（入力）- INTEGER
    - 右辺ベクトルの数。行列 *B* の次元数。 NRHS >= 0。
  - A（入力／出力）- DOUBLE PRECISION, DIMENSION(:,:)
    - 入力時は、N×N の行列 *A* の係数を入れる。
    - 出力時は、*A* から分解された行列 *L* と *U = P * L * U* を圧縮して出力する。*L* の対角要素は 1 であるので、収納されていない。
  - LDA（入力）- INTEGER
    - 配列 A の最初の次元の大きさ。 LDA >= max(1,N)。
  - IPIVOT（出力）- DOUBLE PRECISION, DIMENSION(:)
    - 交換行列 A を構成する枢軸のインデックス。 行列の i 行が IPIVOT(i) 行と交換されている。
  - B（入力／出力）- DOUBLE PRECISION, DIMENSION(:,:)
    - 入力時は、右辺ベクトルの N×NRHS 行列 B を入れる。
    - 出力時は、もし、INFO = 0 なら、N×NRHS 行列である解行列 *X* が戻る。
  - LDB（入力）- INTEGER
    - 配列 B の最初の次元の大きさ。 LDB >= max(1,N)。
  - INFO（出力）- INTEGER
    - = 0: 正常終了

- ＜0:　もし INFO = -i なら i-th 行の引数の値がおかしい。
- ＞0:　もし INFO = i なら U(i,i) が厳密に 0 である。分解は終わるが、U の分解は特異なため、解は計算されない。

３．３　LAPACK のコンパイルおよび利用法

LAPACK の DGESV ルーチンを、東京大学情報基盤センター中島研吾教授提供の粒子間熱伝導問題[3]に適用した時の性能を示します。

この問題では、ほぼ密行列となる連立一次方程式の解法が主演算となります。したがって、行列を密行列として扱い、LAPACK の DGESV ルーチンをコールして解を求めます。

以下に、日立最適化 Fortran90 コンパイラから、ESSL の LAPACK を利用する場合の利用例（スレッド並列化）のコンパイル例を示します。

■ESSL ライブラリの LAPACK を使う場合のコンパイル例(Fortran90 言語、スレッド並列版)

```
f90 -o testLAPACK -i,L -opt=ss -parallel testLAPACK.f -lesslsmp
```

DGESV の呼び出しは、以下になります。

■DGESV の呼び出し例(Fortran 言語)

```
call DGESV(N, INC, AMAT, N, PIV, RHS, N, INFO)
```

実行に際し、8 スレッド実行の場合は、以下のようにします。

■日立コンパイラから ESSL の LAPACK を使う場合のスレッド数指定の例

```
export MEMORY_AFFINITY=MCM
export HF_PRUNST_THREADNUM=1
export OMP_NUM_THREADS=8
./test
```

以上の MEMORY_AFFINITY=MCM は、NUMA 構成を考慮して、ローカルメモリに配列データを配置することで最適化を行うための指定です[1]。

３．４　DGESV の性能

表 1 に、SR16K の 1 ノードにおける、N=16,000 の正方密行列の ESSL の DGESV の実行性能を示します。

表 1 から、16 スレッドまでは良好な台数効果です。しかしながら 32 スレッドを超えると、台数効果が劣化します。この理由は、スレッド実行のオーバーヘッドが見えてくるからと推定されます。さらに大きな行列サイズでの実行をすることで、32 スレッドを超える実行での台数効果が改善できる可能性があります。

[3] http://nkl.cc.u-tokyo.ac.jp/09e/CE23/CE23.pdf

表 1 SR16K の 1 ノードの ESSL の DGESV 性能 (N=16,000)

スレッド数	1	4	8	16	32	64 (SMT)
[GFLOPS]	25.4	104.6	207.7	397.0	483.4	563.2
台数効果	1.00	4.1	8.1	15.6	19.0	22.1

表 1 の 1 スレッド実行のときの理論ピーク性能に対する効率は 25.4/30.64 なので約 82.8%です。

4. ScaLAPACK

4. 1 概要

ScaLAPACK[4]は、密行列に対する、連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンの並列化版を提供している数値計算ライブラリです。MPI での分散並列化に対応しています。

API は LAPACK に類似しています。しかし、LAPACK で提供されている関数すべてが ScaLAPACK で提供されているわけではありません。

ソフトウェア階層化がされています。ScaLAPACK は、BLAS を利用し、内部ルーチンは LAPACK、並列 API は BLACS で構築されています。したがって、BLAS、LAPACK、および BLACS の指定が、コンパイル時に必要になります。

分散メモリ並列計算機での行列（配列）のデータ分散方式に、2 次元ブロック・サイクリック分散方式を採用しています。ユーザは、このデータ分散方式を理解した上で、ScaLAPACK ルーチンをコールする前に、自ら分散データを用意する必要があります。

なお、データ分散方式など、並列処理の基礎知識を習得したい場合は、東京大学情報基盤センター主催の MPI 基礎講習会[5]の受講を勧めます。

4. 2 BLACS と PBLAS

ScaLAPACK を利用する際に重要となる BLACS と、並列版の BLAS である PBLAS の概要を説明します。

- BLACS
  - ScaLAPACK 中で使われる通信機能を関数化したもの。
  - 通信ライブラリは、MPI、PVM、各社が提供する通信ライブラリを想定し、ScaLAPACK 内でコード修正せずに使うことを目的とする
    - 通信ライブラリのラッパー的役割。ScaLAPACK 内で利用されている。
  - 現在 MPI がデファクトになったため、MPI で構築された BLACS のみ利用されている。
    - MPI をコンパイルできるコンパイラでコンパイルし、起動して利用する。
- PBLAS
  - BLACS を用いて BLAS と同等な機能を提供する関数群。並列版 BLAS。

4. 3 ScaLAPACK の API の一例

ScaLAPACK の命名規則については、原則、LAPACK の関数名の頭に“P”を付けたものになりま

[4] http://www.netlib.org/scalapack/
[5] http://www.cc.u-tokyo.ac.jp/support/kosyu/

す。そのほか、BLACS、PBLAS、データ分散を制御するための ScaLAPACK 用関数が用意されています。

例として、連立一次方程式の求解ルーチンである PDGESV を紹介します。

API 例：PDGESV

- **PDGESV**

  (N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO)

  - *sub(A) X = sub(B)*の解の行列 *X* を計算する
  - ここで sub(*A*)は *N*×*N* 行列を分散した A(IA:IA+N-１, JA:JA+N-１)の行列
  - *X* と *B* は *N*×*NRHS* 行列を分散した B(IB:IB+N-１, JB:JB+NRHS-１)の行列
  - 行交換の部分枢軸選択付きの LU 分解 で *sub(A)* を sub(*A*) = *P* * *L* * *U* と分解する。ここで、*P* は交換行列、*L* は下三角行列、*U*は上三角行列である。
  - 分解された *sub(A)* は、連立一次方程式 *sub(A)* * *X* = *sub(B)*を解くのに使われる。
  - N（大域入力） - INTEGER
    - 線形方程式の数。行列 A の次元数。 N >= 0。
  - NRHS（大域入力） - INTEGER
    - 右辺ベクトルの数。行列 B の次元数。 NRHS >= 0。
  - A（局所入力／出力） - DOUBLE PRECISION, DIMENSION(:,:)
    - 入力時は、N×N の行列 A の局所化された係数を
      配列 A(LLD_A, LOCc(JA+N-１))を入れる。
    - 出力時は、A から分解された行列 L と U = P*L*U を圧縮して出力する。
      L の対角要素は１であるので、収納されていない。
  - IA(大域入力) －INTEGER : *sub(A)*の最初の行のインデックス。
  - JA(大域入力) －INTEGER : *sub(A)*の最初の列のインデックス。
  - DESCA（大域かつ局所入力） - INTEGER
    - 分散された配列 A の記述子。
  - IPIVOT（局所出力） - DOUBLE PRECISION, DIMENSION(:)
    - 交換行列 *A* を構成する枢軸のインデックス。 行列の i 行が IPIVOT(i)行と交換されている。分散された配列(LOCr(M_A)+MB_A)として戻る。
  - B（局所入力／出力） - DOUBLE PRECISION, DIMENSION(:,:)
    - 入力時は、右辺ベクトルの N×NRHS の行列 B の分散されたものを(LLD_B, LOCc(JB+NRHS-１))に入れる。
    - 出力時は、もし、INFO = 0 なら、N×NRHS 行列である解行列 X が、行列 B と同様の分散された状態で戻る。
  - IB(大域入力) －INTEGER
    - *sub(B)*の最初の行のインデックス。
  - JB(大域入力) －INTEGER
    - *sub(B)*の最初の列のインデックス。
  - DESCB（大域かつ局所入力） - INTEGER
    - 分散された配列 B の記述子。

- INFO（大域出力）ーINTEGER
  - = 0: 正常終了
  - ＜ 0:
    - もし i 番目の要素が配列で、 その j 要素の値がおかしいなら、INFO = -(i*100+j)となる。
    - もし i 番目の要素がスカラで、かつ、その値がおかしいなら、INFO = -i となる。
    - ＞0: もし INFO = K のとき U(IA+K-１, JA+K-１）が厳密に 0 である。分解は完了するが、分解されたUは厳密に特異なので、解は計算できない。

４．４ ScaLAPACK のコンパイルおよび利用法

ScaLAPACK の PDGESV ルーチンを、LAPACK の時と同じ粒子間熱伝導問題に適用した場合の性能を評価します。

以下に、日立最適化 Fortran90 コンパイラから、Parallel ESSL の ScaLAPACK を利用する場合の利用例（MPI およびスレッド並列化実行（**ハイブリッド MPI 実行**））のコンパイル例を示します。

■ESSL ライブラリの ScaLAPACK を使う場合のコンパイル例(Fortran90 言語、スレッド並列版)

```
f90 -o testScaLAPACK -i,L -opt=ss -parallel -L/usr/local/lib testScaLAPACK.f
 -lblacssmp -lpesslsmp -lscalapack
```

なお、PDGESV の呼び出しは、以下になります。

■PDGESV の呼び出し例(Fortran 言語)

```
call  CALL PDGESV(NN, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO )
```

実行に際し、ScaLAPACK では、プロセスの 2 次元構成（**プロセッサー・グリッド**）の情報が必要になります。

プロセスとは別に、プロセスから呼ばれるスレッド数の指定が必要です。前述のとおり、ESSL ライブラリを使う場合は、OpenMP の環境変数 OMP_NUM_THREADS で指定します。

以上の前提から、SR16K では、

(全 MPI プロセス数×全スレッド数)/利用ノード数 <= 64　　…(1)

でなくてはいけません。

ノードあたりの MPI プロセス数が 32 以下の場合、SR16K では 1 コア当たり 2 プロセスを割り当て可能なため、（1）1 コア当たり 2 プロセス割り当てるのか、（2） 1 コア当たり 1 プロセス割り当てるのか、の選択があります。

このとき、（2）の割り当てを行うために、mpibind というコマンドが用意されています。

具体的には、4 ノードで、1 ノードあたりの MPI プロセス数が 32、スレッド実行が 1 のとき、

以下のようにジョブスクリプトに記載します。

```
■日立コンパイラから ESSL の ScaLAPACK を使う場合の指定の例
（ノードあたりの MPI プロセス数が 32 の場合）
#@$-q  debug
#@$-N  4
#@$-J  T32
export MEMORY_AFFINITY=MCM
export HF_PRUNST_THREADNUM=1
export OMP_NUM_THREADS=1
poe  mpibind  -thread  1  -hpc ./testScaLAPACK
```

また、上記と同様のノード数と MPI プロセス数とするとき、スレッド数を 2 にすると、1 コア当たり 1 プロセスと 1 スレッドの割り当てとなる SMT 実行になります。このときは、mpibind の指定は不要です。実行例は、以下になります。

```
■日立コンパイラから ESSL の ScaLAPACK を使う場合の指定の例
（ノードあたりの MPI プロセス数×スレッド数が 64 の場合）
#@$-q  debug
#@$-N  4
#@$-J  T32
export MEMORY_AFFINITY=MCM
export HF_PRUNST_THREADNUM=1
export OMP_NUM_THREADS=2
poe  ./testScaLAPACK
```

４．５ PDGESV の性能

ここでは、ノードあたりの問題サイズを N=16,000 に固定した場合における、PDGESV の実行例を示します。この方法での性能評価は、**弱スケーリング**と呼ばれます。

ノードあたりのスレッド数は、SR16K のノードあたりの制限である 64 以下を考慮したうえで、任意に設定できます。すべてのコアを使い切るためには、ノードあたりの MPI プロセス数×スレッド数が、32、もしくは 64（SMT 実行）実行となる必要があります。ここでは、ノードあたりの MPI プロセス数×スレッド数を 64 に限定します。

ScaLAPACK には、2 次元ブロック分割のデータ分散の単位であり、かつ、レベル 3 BLAS の実行性能に影響を及ぼす**ブロック幅**という性能パラメタがあります。

このブロック幅が小さすぎるとデータ分散にともなう演算負荷のバランスは良くなりますが、通信回数の増加とレベル 3 BLAS の性能が悪くなります。また、このブロック幅が大きすぎるとデータ分散にともなう演算負荷のバランスが悪くなりますが、通信回数の削減とレベル 3 BLAS の性能が良くなります。したがって、計算機アーキテクチャと並列実行数に応じて、最高性能になるようにブロック幅を調整する必要があります。

表 2 に、1 ノード、プロセッサ・グリッド 8×8、ノードあたりの MPI プロセス数×スレッド数=P64×T1 (SMT)に固定した時の、ブロック幅 (BL) を変化させたときの性能を示します。

表 2　SR16K の 1 ノードの Parallel ESSL の PDGESV 性能 (N=16,000)。
プロセッサ・グリッド：8×8。ノードあたりの MPI プロセス数×スレッド数=P64×T1 (SMT)

ブロック幅 (BL)	32	64	128	256
[GFLOPS]	293.2	377.9	392.0	308.6

表 2 から、BL=128 の時の性能が最も良いので、BL=128 に固定します。
表 3 に、以上の実行条件における、2 ノードと 4 ノードの実行性能[GFLOPS]を載せます。

表 3　SR16K の Parallel ESSL の PDGESV 性能 (BL=128)

(a) 2 ノードでの実行 (N=32,000)

P(ノードあたりの MPI プロセス数)×T(スレッド数) (プロセッサ・グリッド)	P64×T1 (8×16)	P32×T2 (8×8)	P16×T4 (4×8)	P8×T8 (4×4)	P4×T16 (2×4)	P2×T32 (2×2)	P1×T64 (1×2)
[GFLOPS]	902.2	712.9	833.4	880.5	788.5	947.7	680.3

(b) 4 ノードでの実行 (N=64,000)

P(ノードあたりの MPI プロセス数)×T(スレッド数) (プロセッサ・グリッド)	P64×T1 (16×16)	P32×T2 (8×16)	P16×T4 (8×8)	P8×T8 (4×8)	P4×T16 (4×4)	P2×T32 (2×4)	P1×T64 (2×2)
[GFLOPS]	1998.4	1200.0	1752.4	1830.1	2002.2	923.4	1658.0

表 3 から、プロセッサ・グリッドの構成とスレッド数の組合せに依存し、演算性能が変化します。したがって、これらはチューニング・パラメタとなります。
4 ノードのとき、P4×T16 (プロセッサ・グリッド：4×4) の実行形態の時が高速です。この時の理論ピーク性能に対する効率は 2002.2/(980.48*4)なので、約 51%になります。

５．その他のライブラリについて

SR16K では、BLAS、LAPCK、ScaLAPACK のほかに、以下のライブラリがプリインストールされています[6]。場合により、利用のご検討をお願いします。

- **MATRIX/MPP、MATRIX/MPP/SSS**
  - MATRIX/MPP は基本配列演算、連立 1 次方程式、逆行列、固有値・固有ベクトル、高速 Fourier 変換、擬似乱数等に関する副プログラムライブラリです。並列処理用インタ

[6] http://www.cc.u-tokyo.ac.jp/system/smp/smp-tebiki/chapter8.html#C8_1

フェースを用いることにより、データを各ノードに分散して配置、並列に実行することができます。

- MSL2
  - MSL2 は行列計算（連立 1 次方程式、逆行列、固有値・固有ベクトル等）、関数計算（非線形方程式、常微分方程式、数値積分等）、統計計算（分布関数、回帰分析、多変量解析等）に関する副プログラムライブラリです。
- Parallel NetCDF
  - アプリケーションに対し共通のデータアクセス方法を提供する I/O ライブラリ NetCDF(Network Common Data Form) ver1.1.1 がインストールされています。MPI 版のみの提供です。
- FFTW
  - FFTW(Fastest Fourier Transform in the West) は離散 Fourier 変換を計算するライブラリです。ver3.3 の MPI 版、要素並列（スレッド並列）版、スカラ（逐次）版がインストールされています。
- SuperLU、SuperLU_DIST
  - 疎行列の直接ソルバである SuperLU がインストールされています。ver4.2 の要素並列（スレッド並列）版及びスカラ（逐次）版、MPI 版の SuperLU_DIST (MPI 版、ver2.5) がインストールされています。
- C++ ライブラリ
  - C++ライブラリとして、日立最適化 C++ コンパイラにて標準ライブラリの一つである STL(Standard Template Library) を使用するためにはリンク時にオプションとして利用します。
  - Boost C++ ライブラリがインストールされています。リンク時にオプションとして指定します。本ライブラリは日立最適化 C++コンパイラには対応しておりません。IBM C++ コンパイラ、GNU C++ コンパイラをご利用ください。

６．おわりに

本稿では、SR16K における、数値計算ライブラリ BLAS、LAPACK、ScaLAPACK の利用法と、簡単な性能を紹介しました。

これらのライブラリが利用できる場合､ユーザ自ら作成したプログラムより 10 倍以上高速である可能性があります。各自のプログラムの利用状況を判断した上で、利用の検討をしていただければ幸甚です。

参 考 文 献

[1]片桐孝洋:HITACHI SR16000/M1 チューニング連載講座　2.単体ノード性能チューニング、スーパーコンピューティングニュース、東京大学情報基盤センター、Vol.14 No.1 (2012 年 1 月)