Vol.2010-HPC-125 No.2
2010/6/17

自動チューニングインターフェース OpenATLib における 疎行列ベクトル積アルゴリズム

櫻井隆雄[†] 直野健[†] 片桐孝洋[††]
中島研吾[††] 黒田久泰[‡,††] 猪貝光祥[§]

自動チューニングの実装の再利用性を実現するため，筆者らは自動チューニングインターフェース OpenATLib を提案している．本稿では OpenATLib の提供する機能の 1 つである疎行列ベクトル積の備える 7 種のアルゴリズム(対称向け 3 種，非対称向け 4 種)について述べる．各アルゴリズムは単体実行向けや並列環境向けに最適化されており，与えられた行列や実行環境により最も高い性能が得られるアルゴリズムが自動的に選択される．T2K オープンスパコン（東大版）1 ノード（16 コア）上で様々なフロリダ大の行列を用いて各アルゴリズムを比較した．その結果，16 コア環境において対称向けアルゴリズムで最大 12.7 倍，非対称向けアルゴリズムで最大 2.5 倍の性能差があり，アルゴリズムの自動選択機能が有効であるとわかった．

Sparse Matrix-Vector Multiplication Algorithm for Auto-Tuning Interface “OpenATLib”

Takao Sakurai[†] Ken Naono[†] Takahiro Katagiri[††]
Kengo Nakajima[††] Hisayasu Kuroda[‡,††] Mitsuyoshi Igai[§]

We proposed Auto-tuning interface “OpenATLib” to realize reusability for implementation of Automatic Tuning facility (RIAT) on numerical libraries. In this paper, we developed 7 kinds of sparse matrix vector multiplication algorithms for OpenATLib (3 kinds for symmetric matrices and 4 kinds for unsymmetric matrices). By auto-tuning method, the best algorithm is selected for input matrices and execution environment. Performance evaluation of OpenATLib using several U. Florida matrices on T2K Open Supercomputer (U. Tokyo Combined Cluster) on 1 node (16 cores) indicated that the maximum speedup established 12.7x (for symmetric) and 2.5x (for unsymmetric).

1. はじめに

科学技術計算などに利用される行列計算ライブラリでは，入力パラメータの値の選択，疎行列ベクトル積演算や直交化演算といった重要な演算のアルゴリズムの選択により全体の演算時間が大幅に変化する．一方で，短い時間で演算できる良好なパラメータやアルゴリズムは入力行列や計算機環境によって大きく異なり，かつ事前予測が難しいものが存在する．そのため，行列計算ライブラリのユーザが良好な条件でライブラリを利用するのが困難となっていた．

これらの問題を解決する手法として，入力パラメータやアルゴリズム選択の支援をする自動チューニング方式（AT 方式）が注目されている．それらを備えた数値計算ライブラリが開発されてきた．例えば PHiPAC[1], ATLAS[2], FFTW[3], I-LIB[4], ABCLib[5], OSKI[6]などである．これらの方式により良好な条件でライブラリを使用するのが容易になってきた．

しかし，既存のライブラリでは対応していない特殊な行列を処理する場合などにおいて，行列計算ライブラリを自作する場合もある．このような自作のライブラリに対し，その作成者が AT 方式を適用する場合，プログラム記述のためにコード量が増加する場合がある．これらの問題のため，作成者が多数の AT 方式を自作ライブラリに適用するには非常に多くの労力を必要とする．そこで，これまで個別に提案されてきた自動チューニングの実装に関して再利用性（RIAT: Reusability for Implementation of Automatic Tuning facility）を高める手段が求められている．

この課題に対し，筆者らは AT 方式の共通部分を関数として提供する AT インターフェース OpenATLib を提案している．OpenATLib により少ない労力で AT 方式を自作ライブラリに組み込むことが可能となる．本稿では OpenATLib の β 版における主要機能の 1 つである疎行列ベクトル積の自動チューニング方式について述べ，それと同時に T2K オープンスパーコンピュータ上で性能を評価することでその有効性を検証する．

2. AT インターフェース OpenATLib の概要

筆者らは，RIAT の実現のため，Fortran で記述された疎行列反復法ライブラリ向け

† 日立製作所 中央研究所
Central Research Laboratory, Hitachi, Ltd.
†† 東京大学情報基盤センター スーパーコンピューティング研究部門
Supercomputing Research Division, Information Technology Center, The University of Tokyo
‡ 愛媛大学 大学院理工学研究科
Graduate School of Science and Engineering, Ehime University
§ 日立超 LSI システムズ
Hitachi ULSI Systems Co., Ltd.

の汎用 AT インターフェース OpenATLib を提案している[7,8]．OpenATLib は自動チューニング機能を備えた疎行列ベクトル積や直交化演算のサブルーチンを API (Application Programming Interface) の形式で提供する．これらは主にライブラリ作成者が自作ライブラリに AT 方式を実装するために利用することを想定している．OpenATLib が提供する AT 方式により疎行列反復解法ライブラリにおいて実行前に予測が困難なパラメータやアルゴリズムの最適値を実行中に探索し，1 回の演算で安定して高い演算性能を提供することが可能となる．

現在開発中の OpenATLib の β 版が備える機能は以下の 4 つである．これらの機能の命名規則は次のとおりである．まず，“OpenATI_”で機能名が始まる．続いて，“_”後の最初の 1 文字は演算精度（S:単精度，D:倍精度）であり，2 文字目と 3 文字目について，補助機能の場合は“AF”とし，演算機能の場合は，2 文字目は行列形状（S:対称，U:非対称），3 文字目は行列の格納形式（R:CRS，C:CCS）を表す．4 文字目と 5 文字目以降は機能名となる．

● リスタート周期最適化機能：**OpenATI_DAFRT**
Krylov 部分空間法において，相対残差の履歴から算出した停滞を判断する指標を用いてリスタート周期の最適値を実行時に判断する．

● 疎行列ベクトル積機能：**OpenATI_D{S|U}RMV**
複数用意された疎行列ベクトル積のアルゴリズムの中で，与えられた行列と実行する計算機環境において最も高い性能が得られる方式を判断する．**OpenATI_DSRMV** が対称行列向けの関数であり，**OpenATI_DURMV** が非対称行列向けの機能である．

● Gram-Schmidt 直交化機能：**OpenATI_DAFGS**
複数用意された Gram-Schmidt 直交化の実装方式の中で，与えられた行列と実行する計算機環境において最も高い性能が得られる方式を判断する．

● 数値計算ポリシーを反映するためのトランスレーター：
OpenATI_LINEARSOLVE | OpenATI_EIGENSOLVE
ユーザが要求する AT の方針（ポリシー）に基づいて演算を実行するために最適なライブラリの入力パラメータの値を判断する．

本稿ではこれらの機能の中で，疎行列ベクトル積機能である **OpenATI_D{S|U}RMV** に関して，実装されたアルゴリズムを述べるとともに，性能評価を実施する．

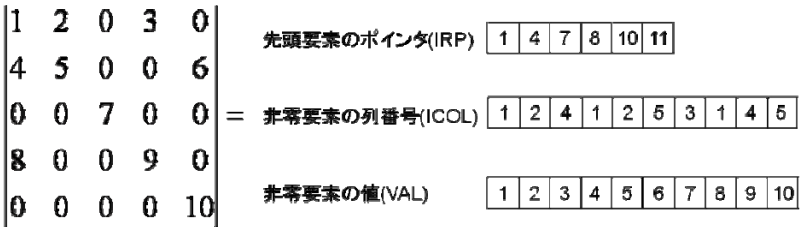

図 1　非対称行列における CRS (Compressed Row Storage)形式

3. 疎行列ベクトル積自動チューニング機能が備えるアルゴリズム

3.1 疎行列ベクトル積自動チューニング機能の概要

疎行列ベクトル積 y=Ax の自動チューニング機能 OpenATI_DSRMV は倍精度型実対称疎行列-ベクトル積, OpenATI_DURMV は倍精度型実非対称疎行列ベクトル積の最適なアルゴリズムを判定し実行する機能である．これらの機能は全て**図 1** に示すような CRS (Compressed Row Storage)形式に対応している．

疎行列ベクトル積は反復解法において多くの回数が実行され，その総演算時間は解法の中で大きな割合を占める．疎行列ベクトル積のアルゴリズムは様々なものが存在するが，実行する計算機環境や与えられた疎行列の非零構造などにより適切なアルゴリズムは変化する．そのため，一概にどのアルゴリズムが最適のものであるとは決め難い．そのため，実行時に計算機環境や入力行列に応じて最適なアルゴリズムを選択する方式が必要となる[9].

OpenATI_D{S|U}RMV は解法開始直後の最初の疎行列ベクトル積の実行時に各々のアルゴリズムを順番に実行し，演算時間が最も短かった方式で以後の疎行列ベクトル積を実行することで最適アルゴリズムの選択を実現している．また，要求メモリ量を節約するオプションも備えており，それが有効となっている場合はメモリを余分に必要とするアルゴリズムは選択対象としない機能も備えている．

以下に示すように OpenATI_DSRMV は 3 種，OpenATI_DURMV は 4 種のアルゴリズムを用意している．

OpenATI_DSRMV（対称疎行列用）
S1) 行分割方式
S2) 非零要素分割方式
S3) 非零要素分割方式＋演算量最適化リダクション並列

OpenATI_DURMV（非対称疎行列用）
U1) 行分割方式
U2) 非零要素分割方式
U3) Branchless Segmented Scan(BSS)方式
U4) Original Segmented Scan(OSS)方式

ライブラリ作成者は BLAS2（疎行列ベクトル積の標準ライブラリ）と同様に疎行列ベクトル積の実行タイミングで OpenATI_D{S|U}RMV を呼び出せばよい．
3.2～3.4 節で各アルゴリズムの詳細を述べる．

3.2 行分割方式と非零要素分割方式

本節では対称行列向けの OpenATI_DSRMV の S1，S2，非対称行列向けの OpenATI_DURMV の U1，U2 方式で実装されている行分割方式と非零要素分割方式について述べる．疎行列ベクトル積 y=Ax における疎行列 A が図 1 で示したような CRS 形式で格納されており，各行の先頭要素のポインタが IRP，各非零要素の列番号が ICOL，各非零要素の値が VAL という名の配列に格納されていたとすると，疎行列ベクトル積の単純な実装として以下のようなコードが考えられる．

```
DO I=1, N
    TMP = 0.0D0
    DO J=IRP(I), IRP(I+1)-1
        TMP = TMP + VAL(J)*X(ICOL(J))
    END DO
    Y(I) = TMP
END DO
```

このようなコードに対して並列化する際には，各スレッドが処理する行数が均等になるように次元数 N を基準とした分割を行う方式が広く使われている．この方式は，実装が容易であるが，**図 2(a)**のように行によって非零要素数の数に偏りがある場合，スレッド毎の負荷バランスが劣化する．このとき，他と比べて処理時間の長いスレッドが発生するため全体の処理時間が増加する．
そこで，処理時間を均一化するため，**図 2(b)**のように次元数 N ではなく各行の非零要素数 NZ を基準にし，各スレッドが処理する要素数を（総非零要素数）/（スレッド数）とする以下の方式を考案した．

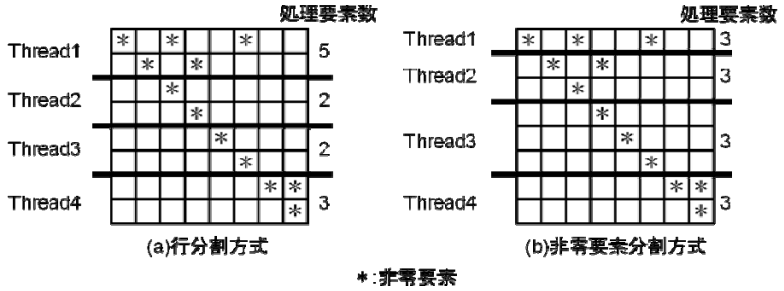

図 2　行分割方式と非零要素数分割方式

[Step 1]：各行の非零要素数を算出する．
[Step 2]：第 1 行は 1 番目のスレッドの担当とする．
[Step 3]：順に次の行の非零要素数を合計していく．
[Step 4]：合計した数が(総非零要素数)/(スレッド数)を超えたらそこまでをそのスレッドの処理担当とし，次の行を次のスレッドの処理担当の開始点とする．
[Step 5]：[Step 3]に戻る．

並列数が大きい場合は，非零要素分割方式により処理要素数が均等になり，演算性能が向上すると期待できる．一方，行ごとの非零要素数のばらつきが少ない行列や並列数が少ない場合は単純な実装の行分割方式が有効となる．

3.3 演算量最適化リダクション並列方式

この節では，OpenATI_DSRMV の S3 方式として実装されている演算量最適化リダクション並列方式について述べる．疎行列ベクトル積のリダクション並列では，メモリアクセスの競合を避けるために各スレッドで部分解を格納するための解ベクトルと同じサイズの配列（リダクションベクトル）を用意する．そして，全スレッドの処理が終了すると，**図 3** のようにリダクションベクトルに格納された部分解を合算し，解ベクトルを作成する．リダクション並列は，最後の合算の分の演算量が増加し，リダクションベクトルのために余分なメモリを必要とするが，メモリアクセスの競合が発生しなくなるため並列加速率が大幅に向上する．
このとき，一般的な実装では要素ごとに全てのスレッドのベクトルの要素を合算する．しかし，リダクションベクトルは各スレッドの担当する行において非零要素の入っている列は非零となっているが，他の列の値は 0 となっている．図 3 の場合，4 本のリダクションベクトルの半数以上の要素は 0 となっており，これらを全て加算する必要はない．

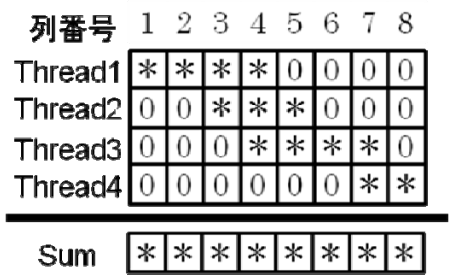

図 3　リダクションベクトルの合算

そこで，OpenATI_DSRMV では，リダクションベクトルの非零要素の配置が入力行列の非零要素の配置によって決定されることに着目し，演算量を最適化したリダクション並列方式を用いる．この方式では入力行列が与えられた際に解ベクトルの各要素において，どのスレッドのリダクションベクトルの要素が非零であるかをあらかじめ保持しておき，非零要素のリダクションベクトルのみを合算の対象とする．例えば，図 3 の第 5 列の場合，スレッド 2 とスレッド 3 のみが非零であるため，この 2 つのみを合算し，スレッド 1 とスレッド 4 は演算の合算の対象としない．

演算量最適化リダクション並列方式は疎な行列を高い並列数で処理する場合に特に有効に機能する．ただし，（解ベクトルの次元数）×（並列数）分の要素を記憶するため，メモリ量に余裕がある場合でないと使用できないという欠点がある．

3.4 Segmented Scan 方式，Branchless Segmented Scan 方式

本節では，OpenATI_DURMV の U3，U4 方式として実装した Segmented Scan 方式[10]と，それをスカラ型プロセッサ向けに改良した Branchless Segmented Scan 方式について述べる．

Segmented Scan 方式では入力行列の非零要素を行単位で分割するのではなく，一定の長さに等分割する．この等分割された非零要素の集合をセグメントベクトル（segment-vector）と呼び，これらのセグメントベクトルに対して行列ベクトル積を要素単位に実行する．さらに，セグメントベクトル内の行の境目を管理するために，各非零要素に対応する FLAG 配列を用意する．FLAG 配列は行の先頭の要素に対応する場合は True であり，それ以外の要素の場合は False となる．1 つのセグメントベクトルの演算では FLAG 配列が T の要素は次の行の先頭要素であるため，処理する行を切り替える．セグメントベクトル間の足しこみにおいては，先頭要素の FLAG 配列の値が False の場合は，前のセグメントベクトルの最後の行の値に加算する．

図 4 に次元数 8，非零要素 30 の行列に対し，長さ 6 のセグメントベクトルを 5 並列で実行する場合のセグメントベクトルと FLAG 配列の例を示す．この演算で 3 列目のセグメントベクトルを処理する場合，3 番目と 6 番目の要素の FLAG が True であるため，1 番目と 2 番目の要素の積算が 1 組目（入力行列の上では 4 行目），2 番目から 5 番目の要素の積算が 2 組目(5 行目)，6 番目の要素の積算が 3 組目の行（6 行目）の部分和を作成する．さらに，全てのセグメントベクトルの積算が終了した後の足しこみ演算において，1 番目の要素の FLAG は False，つまり行の先頭要素ではないため，2 列目のセグメントベクトルの最後の組に部分和を足しこみ，3 行目の総和を算出する．また 3 組目は 4 列目のセグメントベクトルの最初の組の部分和をもらって 5 行目の総和を算出する．

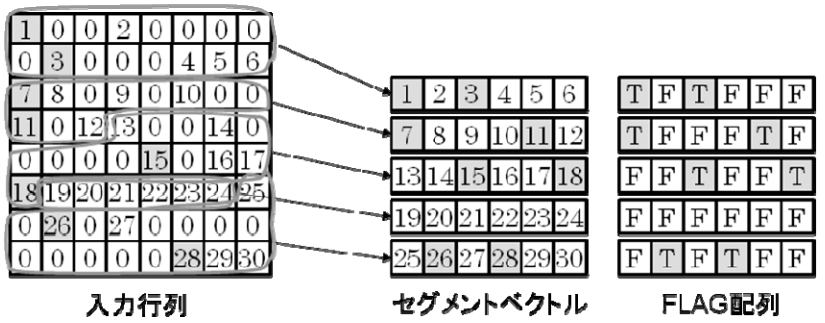

図 4　Segmented Scan 方式における各配列の値の例

このように，Segmented Scan 方式は各要素の処理において FLAG 配列の値により処理が変更されるため，最内側ループに分岐を伴う．しかし，行ごとの非零要素の配置に関わらず完全に各スレッドの負荷を等しく分散させられるため，ベクトル計算機に有効なアルゴリズムである．

しかし，Segmented Scan 方式をスカラ計算機で実行すると，最内側ループ内の分岐処理により，性能が大幅に低下する．そこで，FLAG 配列の代わりに MFLAG，JFSTART という 2 つの配列を用意する．MFLAG 配列は入力行列における各行の先頭要素と，セグメントベクトルに分割した際の先頭要素のポインタが格納される．そして，JFSTART 配列には MFLAG 配列においてセグメントベクトルの先頭要素のポインタが格納されている要素のポインタが入る．ここで，作成した JFSTAT 配列の各要素を外側ループの制御に使用し，MFLAG 配列の各要素を内側ループの制御に使用すると，分岐を行わずに Segmented Scan 方式と同じ処理が実行可能となる．つまり，CRS 形式の行の先頭要素のポインタ配列 IRP をセグメントごとに拡張したデータ構造となる．この方式を Branchless Segmented Scan 方式と名付ける．**図 5** に図 4 の入力行列と同じものに対する MFLAG 配列と JFSTART 配列の例を示す．

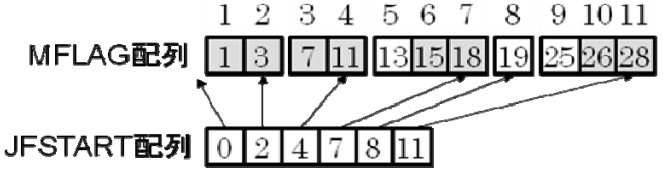

図 5　Branchless Segmented Scan 方式における各配列の値の例

Branchless Segmented Scan 方式はセグメントベクトルによる演算方式を継承しながら，最内側ループにある分岐を排除することで，スカラ型のマルチコア向け実装を改良した方式である．この方式は特定の行列に非零要素が集中している場合に非零要素分割方式よりさらに高い並列加速率が得られると期待できる．また，Segmented Scan 方式で必要な FLAG 配列の要素数は非零要素の個数と同じであるが，Branchless Segmented Scan 方式に必要な MFLAG 配列，JFSTART 配列の要素数は合わせてもセグメントベクトルの本数の 2 倍に解ベクトルの要素数を加えたものであり，メモリ使用量においても Branchless Segmented Scan 方式が有効となる．

ここまでに述べた方式を入力行列や計算機環境に合わせて適切に選択することで疎行列ベクトル積演算において高い性能が得られると期待できる．

4. 性能評価

4.1 性能評価環境とテスト行列

OpenATI_DSRMV，OpenATI_DURMV の有効性を確認するためにこれらの性能評価を実施した．本評価は T2K オープンスパコン（東大版）1 ノードを利用した．T2K オープンスパコン（東大版）1 ノードの仕様およびコンパイル環境を**表 1** に示した．

テスト行列はフロリダ大学の Sparse Matrix Collection[11]よりそれぞれ 5 個ずつ選出した．テスト用の対称行列を**表 2**，非対称行列を**表 3** に示した．表 2，表 3 では分野ごとに行列をまとめ，次元数の少ない順に並べた．

また，非対称行列については表 3 の行列に加えて，**図 6** のような 1 行だけ全ての要素が非零であり，他の行では対角要素のみが非零であるような特殊な行列を用意した．このような行列の場合，行単位で分割する非零要素分割方式より行内で分割が可能な Branchless Segmented Scan 方式が有効に働くと考えられる．

本性能評価では各アルゴリズムについて 1 回の前準備と 1000 回の疎行列ベクトル積を実行にかかった時間を 1000 で割り，それを 1 回の処理として GFLOPS 値を算出した．

表 1　T2K オープンスパコン（東大版）1 ノードの仕様およびコンパイル環境

CPU	Quad-Core AMD Opteron(tm) Processor 8356 2.3GHz 16core/1node
L2 サイズ	2MByte/4core
主記憶サイズ	32GByte（8GByte/1Socket）
OS	Red Hat Enterprise Linux 5
コンパイラ	Intel Fortran Compiler Professional Version 11.0
コンパイル オプション	–O3 –m64 –openmp –mcmodel=medium

表 2　テスト用対称行列

Matrix	N	NNZ	Field
gyro	17361	519260	model
t3dl	20360	265113	reduction
Si5H12	19896	379247	structural
dawson5	51537	531157	
Ga41As41H72	268096	9378286	

表 3　テスト用非対称行列

Matrix	N	NNZ	Field
memplus	17758	126150	electric circuit
poisson3Da	13514	352762	fluid dynamics
xenon1	48600	1181120	materials
epb1	14734	95053	thermal
epb3	84617	463625	

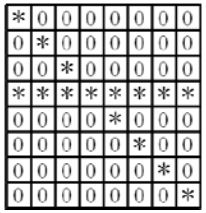

図 6　1 つの行に非零要素が集中した特殊な行列の例

4.2 対称疎行列ベクトル積の評価結果

最初に表 2 で示した 5 つ対称疎行列について疎行列ベクトル積の性能を評価した．

図 7 が SMP 並列数=1 の場合，**図 8** が並列数=16 の場合である．図 7，図 8 では横軸が疎行列，縦軸が性能(GFLOPS 値)である．

図 7 に示された結果では，リダクション並列を用いる S3 方式が最も高い性能が得られており，他の S1，S2 の 2 方式と比べて 1.5～1.6 倍程度の性能値となっていた．他の 2 方式を比較すると，行列の分割方式を決定するために余分な処理を必要としない S1 方式の方が平均して 10%性能が高かった．

一方，図 8 に示された並列数が 16 の場合では，S3 方式が他方式と比べて 4.8～12.8 倍と非常に高い性能を示していた．S1 と S2 方式を比較すると，行列分割方式を変更したことにより S2 方式の方が 15%程度性能向上しており，リダクション並列のためのメモリが用意できない場合は S2 方式を使う方が有利であった．

以上から，余分なメモリを使用しても高い性能が必要な場合は S3 を使うのが有効であると分かった．また，メモリに余裕がないときは，並列数が少ない場合は S1，並列数が多い場合は S2 を使うのが有効であるとわかった．

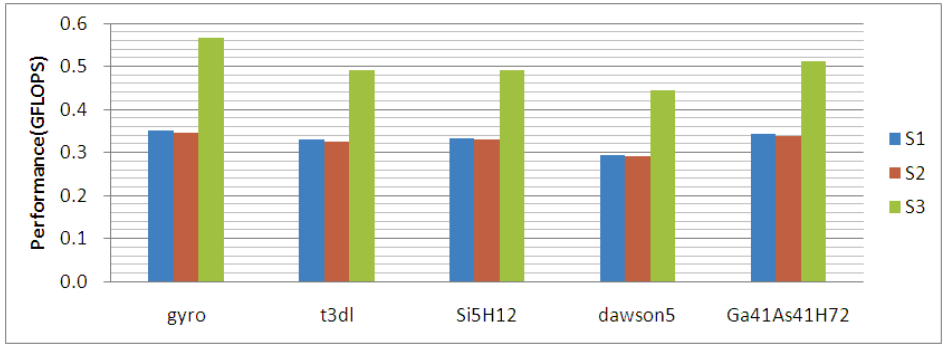

図 7　対称疎行列ベクトル積の性能(並列数=1)

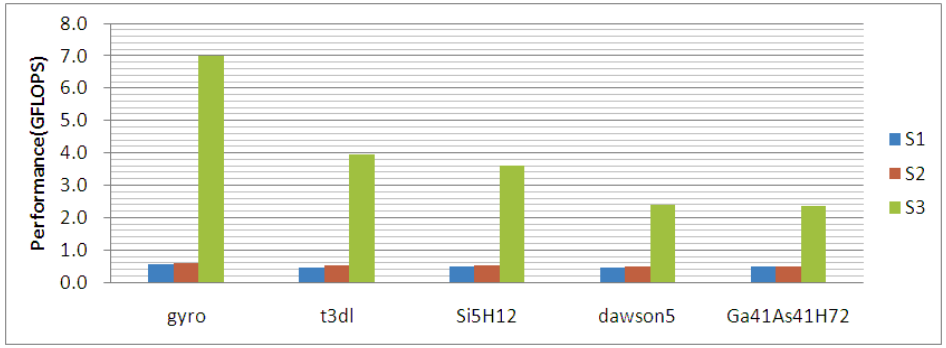

図 8　対称疎行列ベクトル積の性能(並列数=16)

4.3 非対称行列ベクトル積の評価結果

続いて，表 3 で示した 5 つの非対称行列について評価した．**図 9** が並列数 1 の場合，**図 10** が並列数 16 の場合の結果である．

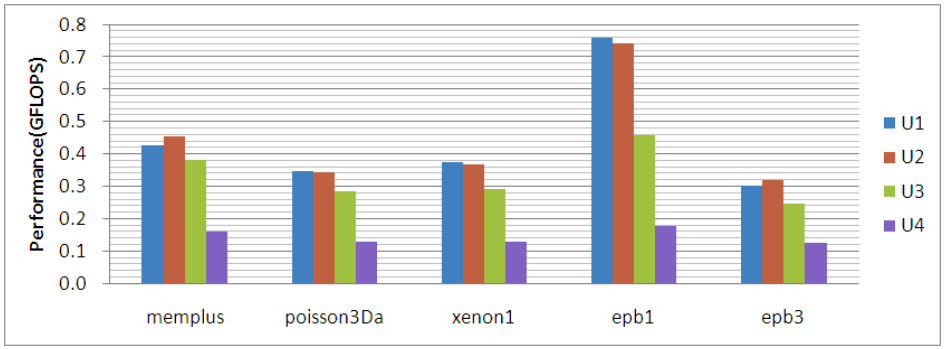

図 9　非対称疎行列ベクトル積の性能(並列数=1)

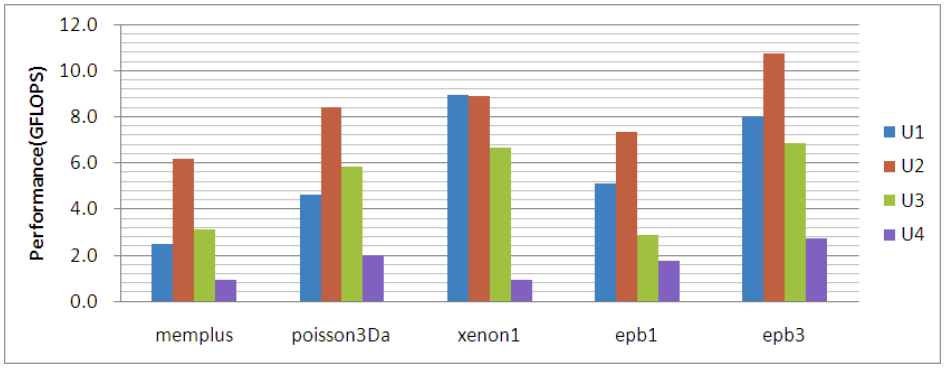

図 10　非対称疎行列ベクトル積の性能(並列数=16)

図 9 に示した並列数 1 の結果では，行列によって U1（行分割方式）が有効な場合と U2（非零要素分割方式）が有効な場合とに分かれていた．一方，図 10 に示した並列数 16 の結果では，4 つの行列で U2 方式が高い性能を得られており，最大の memplus の場合，他の方式に対して 2.5 倍の性能が得られていた．しかし，xenon1 のように，U1 方式が有効な行列もあり，このようなときに全ての方式を試して最も有効なアルゴリズムを探索する OpenATI_DURMV の機能が有効に働くと考えられる．

ここで，1 つの行に非零要素が集中した場合に有効として実装した U3 方式は，ベクトル計算機向けの U4 方式(オリジナルの Segmented Scan 方式)と比べて 2~7 倍性能が向上しており，並列数 16 の場合の memplus と poisson3Da で U1 方式より高性能となった．しかし，その場合も U2 方式よりも高い性能が得られておらず，最も有効な方式とはなっていなかった．

そこで，**図 11** に図 6 の例で示した 1 行だけ全ての要素が非零であり，他の行では対角要素のみが非零であるような特殊な行列で評価した結果を示した．このときの並列数は 16 である．図 11 では横軸が特殊な行列の次元数，縦軸が性能である．
図 11 の結果では，次元数が 200K の場合，U3 方式が他と U2 方式より 5.5 倍高い性能が得られていた．このように，非零要素が極端に 1 つの行に集中した場合は U3 方式が有効となる場合があるとわかった．しかし，現状では有効となる条件が極めて厳しいため，さらに高い性能が得られるように改良する必要があると確認できた．

5. おわりに

本稿では，自動チューニングインターフェース OpenATLib の 1 機能である，疎行列ベクトル積自動チューニング方式 OpenATI_D{S|U}RMV について述べた．

OpenATI_D{S|U}RMV は入力行列や計算機環境に合わせ，最も高い性能が得られるアルゴリズムを自動的に選択し，実行することを特徴とする．さらに，実装した 7 つのアルゴリズムについて述べた上で，フロリダ大の Sparse Matrix Collection から得た 5 個の行列を用いて性能評価を実施した．

評価の結果， OpenATI_D{S|U}RMV により最も有効なアルゴリズムは，単純なものと比べて対称行列の場合は最大で 12.8 倍，非対称行列の場合 2.5 倍の性能が得られていた．また，並列数，入力行列によって有効なアルゴリズムが異なることも確認でき，提案法の有効性が示された．

実装したアルゴリズムの中で，非零要素の数が各スレッドに等分割される非零要素分割方式とリダクション並列における集計演算において不要な 0 加算を省いた演算量最適化リダクション並列方式が特に有効に働いていた．しかし，1 つの行に非零要素が集中した場合に備えて用意した Branchless Segmented Scan が有効に働くことが少なかった．このアルゴリズムのさらなる改良が今後の課題となる．

なお，OpenATLib とそれを用いて開発された疎行列反復法ライブラリ Xabclib のソースコードは，PC クラスタコンソーシアム経由で配布されている．

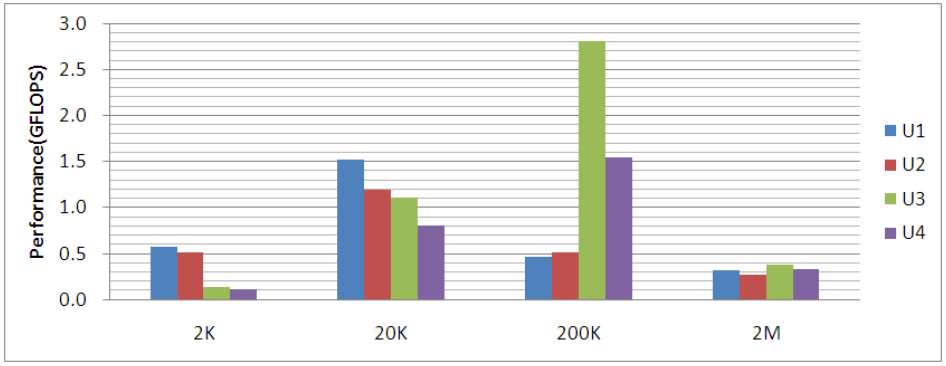

図 11　非零要素を 1 つの行に集中させた特殊な行列での性能(並列数=16)

謝辞　本研究は文部科学省「e−サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」，シームレス高生産・高性能プログラミング環境，の支援を受けている．

参考文献

1)　Bilmes, J., Asanovic, K., Chin, C.-W., and Demmel, J.W.: Optimizing Matrix Multiply Using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, Proceedings of International Conference on Supercomputing 97, pp.340-347 (1997).
2)　Whaley, R.C., Petitet, A., and Dongarra, J.J.: Automated Empirical Optimizations of Software and the ATLAS Project, Parallel Computing, 27, pp.3-35 (2001).
3)　Frigo, M.: A Fast Fourier Transform Compiler, Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation, Atlanta, Georgia, pp.169-180 (1999).
4)　片桐孝洋，黒田久泰，大澤清，工藤誠，金田康正: 自動チューニング機構が並列数値計算ソフトウエアに及ぼす効果，情報処理学会論文誌 : ハイパフォーマンスコンピューティングシステム， 42, SIG 12 (HPS 4), pp.60-76 (2001).
5)　Katagiri, T., Kise, K., Honda, H., and Yuba, T: ABCLib_DRSSED: A Parallel Eigensolver with an Auto-tuning Facility, Parallel Computing, 32, 3, pp.231-250 (2006).
6)　Vuduc, R., Demmel, J.W., and Yelick, K.: OSKI:A Library of Automatically Tuned Sparse Matrix

Vol.2010-HPC-125 No.2
2010/6/17

Kernels, Proceedings of SciDAC 2005, Journal of Physics: Conference Series (2005).
7) 片桐孝洋，櫻井隆雄，黒田久泰，直野健，中島研吾，OpenATLib:汎用的な自動チューニングインターフェースの設計と実装，情報処理学会研究報告:ハイパフォーマンスコンピューティング, 2009-HPC-121(3) (2009).
8) 櫻井隆雄，直野健，片桐孝洋，中島研吾，黒田久泰，OpenATLib を利用した疎行列ライブラリの開発と評価，情報処理学会研究報告:ハイパフォーマンスコンピューティング, 2009-HPC-121(17) (2009).
9) 工藤誠，黒田久泰，片桐孝洋，金田康正:並列疎行列ベクトル積における最適なアルゴリズム選択の効果，第 147 回アーキテクチャ研究会，情報処理学会研究報告 2002-ARC-147, pp.151-156 (2002).
10) Guy E. Blelloch, Michael A. Heroux, and Marco Zagha: Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors, Carnegie Mellon University, Pittsburgh, PA (1993).
11) T. A. Davis: Sparse Matrix collection, http://www.cise.ufl.edu/research/sparse/matrices/
12) Xabclib プロジェクトホームページ, http://www.abc-lib.org/Xabclib/index-j.html