

疎行列-ベクトル積におけるブロック化 BSS 法と

高スレッド並列環境での性能評価

片桐 孝洋[†] 佐藤 雅彦[‡]

本論文では疎行列-ベクトル積において, Segmented Scan (SS) 法のマルチコア向き実装である BSS 法を改良し, 並列性を高め, キャッシュ親和性を高める新方式の Blocked BSS 法を提案する. 128 スレッド実行が可能な HITACHI SR16000/VL1 で性能評価を行った. 性能評価の結果, フロリダ行列では従来の単純な行分割方式に対し非零要素均等化方式が最大で 63.5% の速度向上が達成できた. また, ある特定の行が密となる人工行列では, 従来の BSS 法に対し提案する Blocked BSS 法は 48% の速度向上を達成できる場合があった.

A Blocked BSS Implementation for Sparse Matrix-vector Multiplication and Its Performance Evaluation on a High Thread Parallel Environment

TAKAHIRO KATAGIRI[†], MASAHICO SATO[‡]

In this paper, we propose “Blocked BSS” method, which gives us high parallelism and cache affinity property to BSS method, which is a multicore implementation based on Segmented Scan (SS) method. Performance evaluation with a highly threaded environment by using the HITACHI SR16000/VL1 indicates that: (1) maximum 63.5% speedup is established by using normalized non-zero method to the simple row-decomposition method with Florida matrix collection; (2) 48% speedup is established by using the proposed blocked BSS to the original BSS with an artificial matrix which is set to a particular dense row.

1. はじめに

数値計算ライブラリにおける自動チューニング (以降, AT) 技術は, 密行列や FFT など信号解析ライブラリで成功を収めてきた[1]-[3]. これは, 階層キャッシュに代表されるハードウェアの複雑化と, 数値計算ライブラリ特有の複雑なチューニング手法による開発コストの増大が背景にある. 一方, 疎行列ライブラリは, ハードウェアの複雑化のみならず, 入力データの特徴に見合う実装をしないと高性能が達成できない. そのため入力データの特徴を自動抽出し最適実装を AT する方式が, OSKI[4]を

中心に行われてきた.

近年, 計算機ハードウェアの複雑化はさらに進み, 数十コアに及ぶマルチコア化や多階層キャッシュ化が進んでいる. さらに, GPU (Graphics Processing Unit) に代表される演算アクセラレータとマルチコア CPU の混載型の計算機が登場し, 非均一 (ヘテロジニアス) 環境がハイエンドな計算機環境では普通となりつつある. 通信網も階層化され性能の静的見積もりが難しい. 今後, 耐故障性が重視されることを鑑みると, 実行時までジョブを割り当てた物理ノードの配置は不明となるだろう. 通信処理や計算処理の実行前最適化 (静的最適化) が困

[†] 東京大学 情報基盤センター

Information Technology Center, The University of Tokyo.

[‡] 自然科学研究機構 核融合科学研究所

National Institute for Fusion Science.

難になる。

このような状況の中、我々は疎行列ライブラリにおける実行時 AT の機能開発を目的に、Xablib[5]を開発してきた。特に実行時アルゴリズム(実装法)選択のため、入力データとノード内スレッド数の変化に応じた AT 機能を提供する。OpenMP で並列化しているため、ノード内の高スレッド実行時の AT 効果が重要となる。そこで本論文では、以下の2つの目的がある。

第一に、並列性とキャッシュ再利用性を高めた Blocked BSS (Branchless Segmented Scan) 法を提案する。BSS 法とは、Xablib プロジェクトで開発された Segmented Scan (SS) 法[6]による疎行列-ベクトル積(以降、SpMxV)演算の実装法である。マルチコア型 CPU 向きに高性能化した実装法である。

第二に、高スレッド実行で Xablib の AT 機能の効果を評価する。最大で 1 ノードあたり 128 スレッド実行が可能な HITACHI SR16000/VL1 (以降、SR16K) を用いて Xablib の性能評価を行う。

本論文の構成は以下のとおりである。第 2 章は、AT 機能付き疎行列ライブラリ Xablib と汎用的 AT インターフェース集 OpenATLib について説明する。第 3 章は、Blocked BSS 法の提案を行う。第 4 章は Xablib の SR16000/VL1 を用いた性能評価である。最後に、得られた知見のまとめを行う。

2. OpenATLib と Xablib

OpenATLib は、数値計算ライブラリで必要となる汎用的な AT 機能を API (Application Programming Interface) 化し、その参照実装の提供を目的に開発された[5]。現在、疎行列反復解法のうち、Krylov 部分空間法による解法に特化した AT 機能を実装している。現在、平成 20 年度に開発した OpenATLib α 版を基に、平成 21 年度に機能を高度化した OpenATLib β 版を公開している。 β 版は、以下の新規機能を追加している。

- SpMxV 関数: *OpenATLib_D(S/U)RMV*
 1. 非零要素均等化方式: 非零要素数を考慮し、並列実行時の計算負荷をバランス化させる方式(対称・非対称行列用の双方)
 2. 作業行列の零要素について、並列実行時の加算を省く方式(対称行列用のみ)
 3. Branchless Segmented Scan(BSS)方式(非対称行列用のみ)

- Gram-Schmidt 直交化関数: *OpenATLib_DAFGS*
 1. 古典 Gram-Schmidt (CGS)
 2. Daniel-Gragg-Kaufman-Stewart 型の Gram-Schmidt (DGKS)
 3. 修正 Gram-Schmidt (MGS)
 4. ブロック化古典 Gram-Schmidt (BCGS)
- 数値計算ポリシーを適用するためのメタ・インターフェース関数: *OpenATLib_LINEAROLVE* と *OpenATLib_EIGENSOLVE*

OpenATLib は、OpenMP を利用してスレッド並列化されている。

一方、Xablib は、数値計算ポリシーが実装された OpenATLib β 版を利用し、疎行列に対する対称標準固有値問題の解法であるリスタート付き Lanczos 法、および、連立一次方程式の解法である GMRES(m)法を実装して、数値計算ライブラリ化したものである。すなわち Xablib とは、OpenATLib を利用して開発された AT 機能付き数値計算ライブラリの一実装である。

3. Blocked BSS の提案

3.1 BSS 法のカーネル

OpenATLib β 版で新規開発した BSS 法[7]におけるメインカーネル(計算量が多い部分)は、図 1 となる。

```

<1> !$OMP PARALLEL DO PRIVATE(K,K1,S,I)
<2> DO J=1, JL
<3>   DO K=JFSTART(J-1)+1, JFSTART(J)
<4>     K1=K+1
<5>     S=0.0D0
<6>     DO I=MFLAG(K), MFLAG(K1)-1
<7>       S=VAL(I)*X(ICOL(I))+S
<8>     END DO
<9>     VALSS(K)=S
<10> END DO
<11> END DO
<12> !$OMP END DO PARALLEL

```

図 1 BSS 法のメインカーネル

図 1<2>において、変数 JL は SS 法で分割される疎行列の非零要素を収納する配列の分割数 (SS 法における並列度) を示している。非零要素数を NNZ とすると、NNZ/JL の切り上げ数となるベクトル長ごとに処理をする。

非零要素値が格納されている配列 VAL のインデックスの範囲が収納されている配列を MFLAG とする。図 1<3>の配列 JFSTART に、SS 法による分割後の第 J 行について、MFLAG のインデックスが収納されている。具体的には、JFSTART(J-1)+1 ~ JFSTART(J) に MFLAG のインデックス範囲が収納されている。

配列 VAL が、SS 法の分割において疎行列の行をまたぐように分割されている場合、その行またぎの切れ目ごとに MFLAG へインデックスが収納される。したがって、実際の疎行列の行をまたぐ数に依存し、JFSTART(J) の中身も変化する。

オリジナルの SS 法[6]では、疎行列の行またぎ情報はフラグ配列を用いる。かつ最内側ループで IF 文の記述で実装している。これが、スカラ計算機で性能劣化を引き起こす。そこで BSS 法では、フラグ配列と IF 文の削除を行った。その代わりに、JFSTART と MFLAG の配列を導入した。

3.2 Blocked BSS 法のカーネル

BSS 法のカーネルにおける主演算は図 1<7>の

$$S = \text{VAL}(I) * X(\text{ICOL}(I)) + S, \quad \dots (1)$$

である。式(1)は、スカラ S のループ伝搬フロー依存 (回帰演算) であるので、場合により並列実行を阻害する。

そこで、式 (1) の計算において、積の計算部分と、S への足しこみ部分が分離できる性質を利用し、並列性を高める (S をスカラ・エキスパンション) する方式を実装する。すなわち、

$$\text{VALS}(I) = \text{VAL}(I) * X(\text{ICOL}(I)) \quad \dots (2)$$

$$S = \text{VALS}(I) + S \quad \dots (3)$$

のように、数式とループとを分割することで、間接参照の式 (2) の並列性を増加させる。このことで、コンパイラによるソフトウェア・パイプライン適用の機会の拡大をねらう。この実装を図 2 に示す。

図 2 では、<7>行で S のスカラ・エキスパンション配列 VALS を導入している。さらにキャッシュの親和性を高めるため、図 2<6>-<9>の積の部分で昇順にインデックスを動かし VALS へ値を代入した後、

図 2<11>-<17>では逆に、降順にインデックスを動かす。このことで、配列 VALS に関するキャッシュヒット率の向上をねらう。

以上のように、(1)間接参照演算の並列性を高める目的で S をスカラ・エキスパンションする；(2)キャッシュ親和性を高めるため積部分で昇順、和部分で降順にインデックスを動かす；2 方式を特徴とする BSS 法の実装法を **Blocked BSS 法**と呼ぶ。

```
<1> !$OMP PARALLEL DO
  PRIVATE(S,K,I,K1,K2,IV,VALS,ISTART,IEND)
<2> DO J=1,JL
  ! ---- 積の計算部
<3> K1=JFSTART(J-1)+1; ISTART=MFLAG(K1);
<4> K2=JFSTART(J); IEND=MFLAG(K2+1)-1;
<5> IV=1
<6> DO I=ISTART, IEND
<7>   VALS(IV) = VAL(I)*X(ICOL(I))
<8>   IV = IV + 1
<9> END DO
  ! ---- 和の計算部
<10> IV = IV - 1
<11> DO K=K2, K1,-1
<12>   S=0.0D0
<13>   DO I=MFLAG(K+1)-1, MFLAG(K), -1
<14>     S=S+VALS(IV); IV=IV - 1;
<15>   END DO
<16>   VALSS(K)=S
<17> END DO
<19> END DO
<20> !$OMP END DO PARALLEL
```

図 2 Blocked BSS 法のメインカーネル

3.3 Blocked BSS 法の実装上の特徴

Blocked BSS には、以下の特徴がある。

1. キャッシュの大きさに合うように

配列 VALS の大きさを任意に設定可能

配列 VALS を大きくとりすぎるとキャッシュミスが増え、Blocked BSS 法の効果が無くなると思われる。しかし SS 法の特徴である、分割数 JL を疎行列形状に関係なく任意の数で設定できることを考慮すると、処理単位のベクトル長がキャッシュに収まる範囲で任意に設定できる。したがって、疎行列形状に関係なく、キャッシュに収まる大きさに配列 VALS の大きさを設定可能である。

2. 長い固定ベクトル長の維持が可能

一般に SpMxV のカーネルは、疎行列形状（各行における非零要素の数）に依存し最内側のループ長が定まり、任意の段数でアンローリング出来ない。ところが、Blocked BSS では、式 (2) の積部分（図 2<6>-<9>）は、疎行列形状によらず SS 法の分割によるベクトル長 NNZ/JL（固定長）となる§。

これは、たとえば一行当たりの非零要素数が 5 の場合はベクトル長が 5 に限定されるのに対し、JL が NNZ に比べ無視できるほど小さい場合、Blocked SS 法では NNZ のオーダの長さまで大幅に拡大できることを意味している。すなわち、アンローリング段数がどのような疎行列でも固定段数とれる。ベクトル計算機のようなベクトル長を長くすることで高性能化が達成できる環境に向けたカーネルになる。

4. 性能評価

4.1 計算機環境と対象ライブラリ

核融合科学研究所に設置されたプラズマシミュレータ HITACHI SR16000/VL1 を利用した。CPU は IBM POWER6 (5.0GHz), 64 コア/ノード（物理構成）, 128 スレッド/ノード（SMT 実行時）である。L1 データキャッシュは 64 Kbyte/コア, L2 データキャッシュは 4Mbyte/コア, および L3 データキャッシュは 32 MByte/2 コアである。メモリは、ノード当たり 1024GByte である。OS は AIX 5L v5.3 である。コンパイラは日立最適化 f90 V02-00/IB, コンパイラオプションは"-64 -opt=ss -omp" である。

Blocked BSS を実装した OpenATLib は β 版[8] である。なお、OpenATLib の疎行列圧縮形式は CRS (Compressed Row Storage) 形式である。

図 2<11>-<17>のループを、日立コンパイラが提供する最適化ディレクティブにより、4 段および 8 段のアンローリングを指定した場合の性能を評価する。以下に今回評価する SpMxV の実装をまとめる。

- U1: 行分割方式（従来のシンプルな実装）
- U2: 非零要素均等化方式（OpenATLib β 版の最適化方式）

- U3: BSS 法
- U4: ブロック化 BSS 法（アンローリング無）
- U5: ブロック化 BSS + アンローリング 4 段
- U6: ブロック化 BSS + アンローリング 8 段
- U7: オリジナル SS 法

4.2 テスト行列

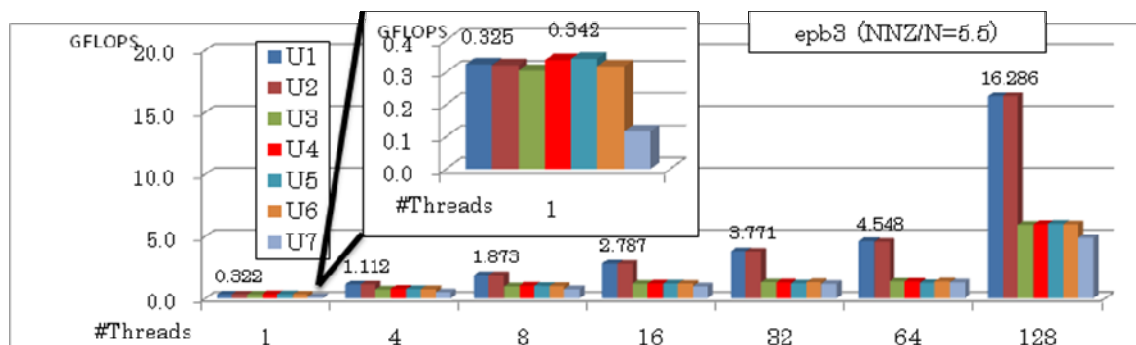
- University Florida Sparse Matrix Collection（以降、フロリダ行列）[9]
22 種の非対称行列を利用した。詳細を表 1 に示す。

表 1 フロリダ行列の詳細

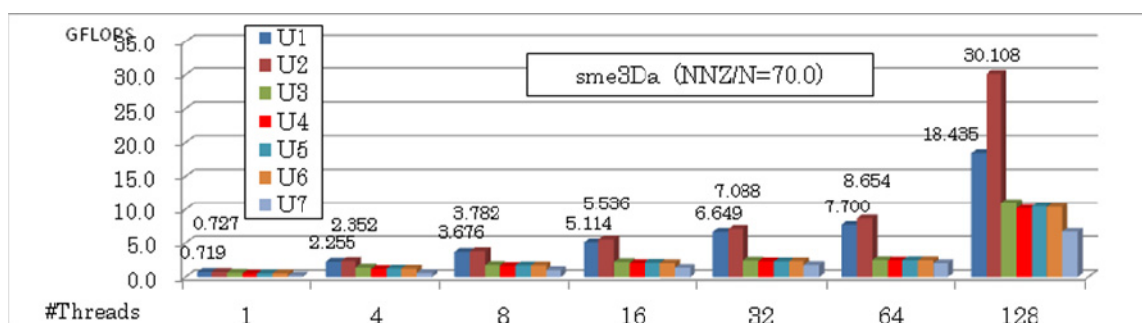
行列名	次元数	非零要素数	適用分野
chipcool0	20082	281150	2D/3D
chem_master1	40401	201201	
torso1	116158	8516500	
torso2	115067	1033473	
torso3	259156	4429042	
memplus	17758	126150	Electric circuit
ex19	12005	259879	Fluid Dynamics
poisson3Da	13514	352762	
poisson3Db	85623	2374949	
airfoil_2d	14214	259688	
viscoplastic2	32769	381326	Materials

行列名	次元数	非零要素数	適用分野
xenon1	48600	1181120	Materials
xenon2	157464	3866688	
wang3	26064	177168	Semiconductor device
wang4	26068	177196	
ecl32	51993	380415	
sme3Da	12504	874887	Structural
sme3Db	29067	2081063	
sme3Dc	42930	3148656	
epb1	14734	95053	Thermal
epb2	25228	175027	
epb3	84617	463625	

§過度のアンローリングでコード量が増えて L1 命令キャッシュから溢れ、性能が低下する場合がある。命令キャッシュ量を考慮したアンローリングが必要である。



(a) eps3 による実行結果 (一行当たりの非零要素数の平均 5.5)



(b) sme3Da による実行結果 (一行当たりの非零要素数の平均 70.0)

図 3 フロリダ行列による性能評価結果. 128 スレッド実行は SMT による実行.

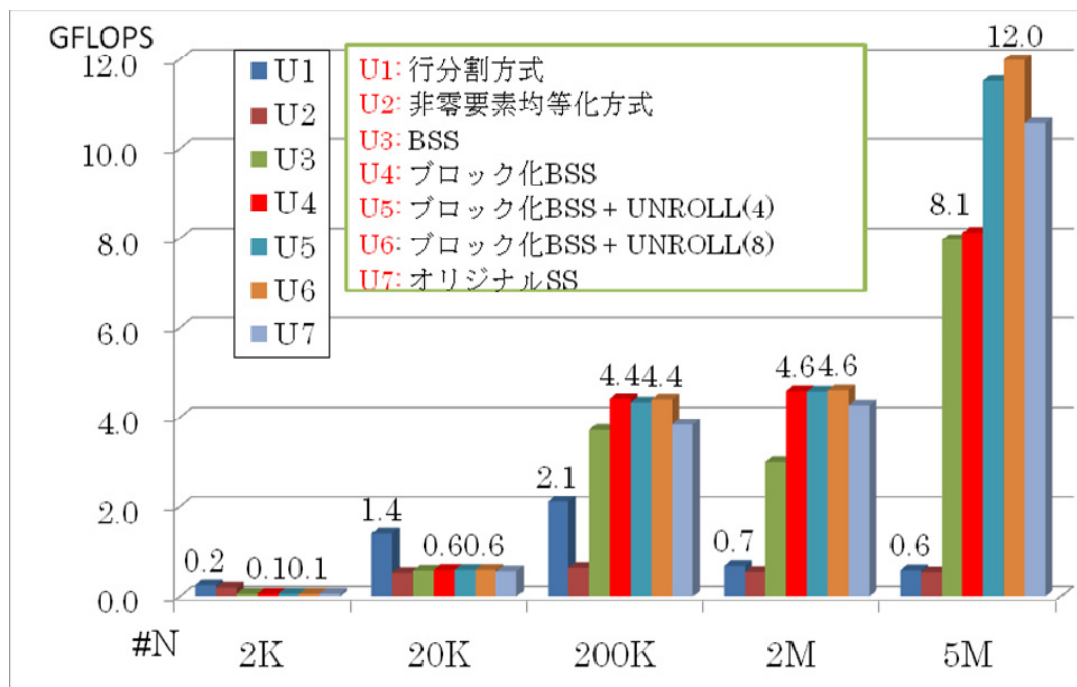
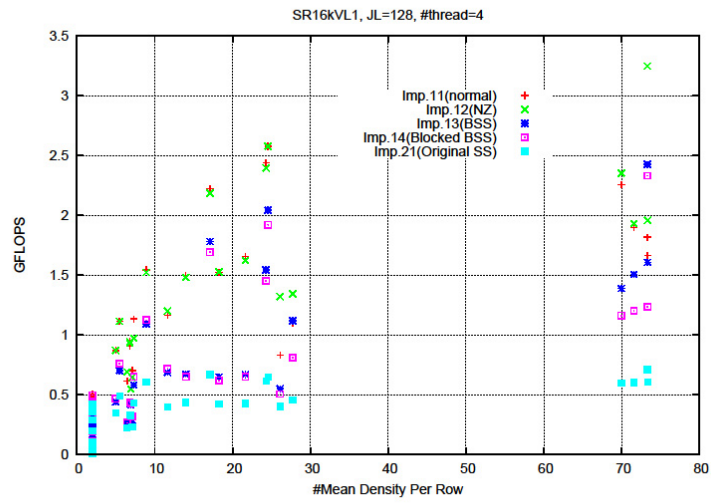
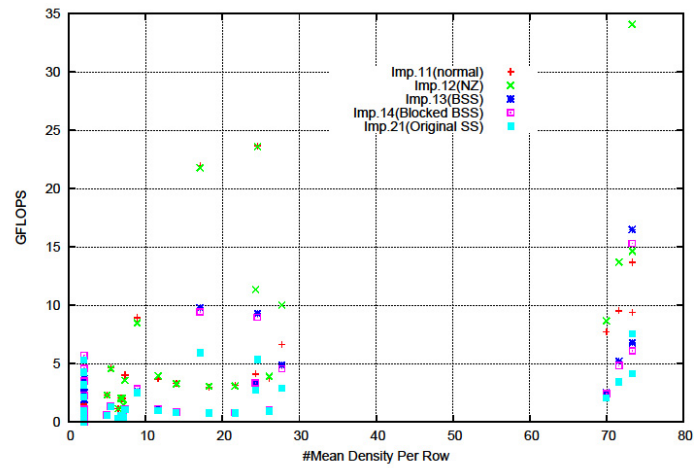


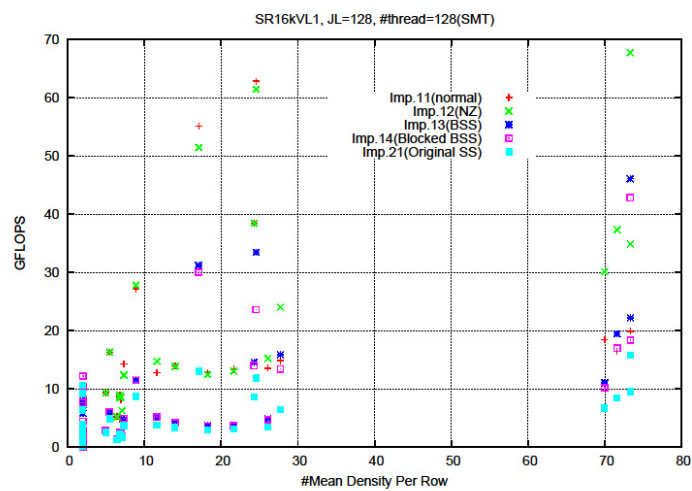
図 4 人工行列による性能評価結果. 128 スレッド (SMT) による実行



(a) 4 スレッド実行



(b) 64 スレッド実行



(c) 128 スレッド実行, SMT 実行

図 5 非対称行列のフロリダ行列 22 種を一行当たりの非零要素数の平均値で分類した図

表 1 のフロリダ行列は、今後の課題で反復解法の AT を評価するため、Xabclib β 版にて実行時間 1,000 秒以内 (4 ソケットの AMD Opteron プロセッサ 8356(2.3GHz)を用いた場合) で収束する行列から選んだ。

● 特定の 1 行が密な人工行列

SS 法は一部の行に非零要素が密集している場合でも並列性が抽出でき、従来法に対しメリットがある。そこで以下のような、第 $N/2$ 行のみ密行となっている人工行列で性能評価を行う。

$$A = (a_{i,j}), (i, j = 1, 2, \dots, N)$$

$$\begin{cases} \text{if } ((i = j) \text{ and } (i \neq N/2)) \text{ then } \#non-zero = 1. \\ \text{if } (i = N/2) \text{ then } \#non-zero = N. \end{cases}$$

非零要素値は乱数により生成。

4.3 Blocked BSS の効果

図 3 に 2 種のフロリダ行列による結果を示す。

図 3(a) では、1 コア実行時においても従来法である U1 より Blocked BSS が高速化されている。その効果は $0.342/0.325=5.2\%$ の速度向上である。しかしそれ以外の図 3(a) の行列では、SS 法全般は従来法より高速ではない。

図 3(b) では、図 3(a) と同様に SS 法全般は従来法より高速で無い。しかし、U2 の非零要素均等化法の速度向上が大きい。特に U1 と比べた場合、U2 は 128 スレッド実行時に $30.1/18.4=63.5\%$ の速度向上を達成した。これは、行列 sme3Da が一行当たりの非零要素数の平均が 70 と大きく、各行の非零要素数に偏りがあり、単純な行分割では高スレッド実行時に計算負荷の均等化ができないからと考える。

一方、図 3(a)(b) とともに、128 スレッド SMP 実行時の速度向上が、64 スレッド実行時に対し約 3.6 倍 (epb3 行列, U2) と、スーパーニアスピードアップを達成している。これは、データのメモリ読みだしの待ち時間が大きいことを意味している。SpMxV のような間接参照を必要とする演算では、SMT 機能が特に効果的であるといえる。

一方、図 4 の人工行列の結果は決定的である。行列サイズを増加させても、従来法の行分割による並列性を抽出する方法 (実装 U1 と U2) は全く並列性が抽出できない。N=5M のとき、U1 に対し BSS 法 (U3) は $8.1/0.6=13.5$ 倍も高速化される。さら

に従来の BSS 法に対し、提案する Blocked BSS 法に 8 段のアンローリングを掛けたもの (U6) は、 $12/8.1=48\%$ の速度向上を達成し、最高速となった。また従来法 (U1) に対する Blocked BSS (U6) の速度向上は $12/0.6=20$ 倍にも達し、決定的な性能差となる。

4.4 一行当たりの非零要素数の影響

図 5 は、非対称なフロリダ行列 22 種と人工行列について、一行当たりの非零要素数の平均を X 軸に取り、Y 軸に各実装の GFLOPS 値を取り分類した図である。

図 5(a)~(c) から、4 スレッド時にはあまり顕著ではないが、64 スレッド、128 スレッドと高スレッド環境になるにつれ、以下の特徴が現れる。

- 行当たりの非零要素数の平均が 10 個~30 個の行列では、行分割方式 (U1) が有効
- 行当たりの非零要素数の平均が 70 個以上の行列では、非零要素均等化方式 (U2) が有効

以上は、64 スレッド並列以上の高スレッド環境では一行当たりの非零要素数の平均が 70 個以上の行列で、単純な行分割では計算負荷の不均衡が生じやすく U2 方式が有効となることを意味している。

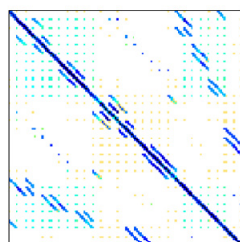
4.5 高性能を達成した行列の特徴

GFLOPS 値が大きい上位 2 行列は以下であった。

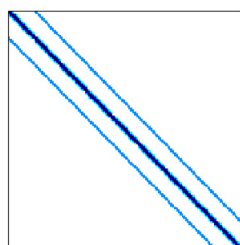
- 67.6GFLOPS (U2, 非零要素均等化方式)
 - 行列名: torso1
 - N: 116,158, NNZ: 8,516,500
 - 1 行当たり平均: 73.3 個
- 62.8GFLOPS (U1, 行分割方式)
 - 行列名: xenon2
 - N: 157,464, NNZ: 3,866,688
 - 1 行当たり平均: 24.5 個

右辺ベクトル x のサイズは、torso1 で約 907Kbyte, xenon2 は約 1.2MByte である。右辺ベクトル x の全ベクトル要素が L2 キャッシュ (4MByte) に載る。また、xenon2 のような対角行列では、参照される右辺ベクトルの要素が L1 キャッシュに載る確率が高く、高性能になることは予想される。面白いのは torso1 である。この行列は 1 行当たりの非零要素数に偏りがあり、非零要素均等化方式の効果が期待できる。

加えて、非零要素の配置は、非均一だが等間隔に配置されている。そのため、一度アクセスされた右辺 x の値が、次のアクセスで L1 キャッシュ上に残る可能性が高い。ゆえに、高性能が達成されたと考えられる。



(a) torso1



(b) xenon2

図 6 高性能行列の非零要素の分布

5. 関連研究

SpMxV で SS 法を用いる研究は GPU を中心に多く研究されている。たとえば、文献[10]があげられる。しかしながらマルチコア向けに SS 法を改良する研究は、Xabclib プロジェクト[7]が先駆けである。

CPU で SS 法を実装したものは文献[6]の Cray Y-MP C90 での実装までさかのぼる。近年の CPU では SS 法は実装評価がされていない。我々の実装 U7 (オリジナル SS 法) が、近年の CPU を用いた従来の SS 法の性能に相当する。

一方、近年では CPU ではなく GPU で SS 法を実装する研究が[11]でなされている。[11]では、CRS 形式での GPU 実装 (SS 法ではない) に対し、フロリダ行列を用いた性能評価で、最大で約 5GFLOPS を達成している。また SS 法による実装では、本論文で採用した偏った行列で、3.26GFLOPS を達成している。しかしながらいずれも、本評価で得られた性能値、67.6GFLOPS (U2, 非零要素均等化方式、行列名: torso1), および、12GFLOPS (偏った行列) に及ばない。

特定の行に非零要素が偏る場合、並列性の抽出に SS 法を使わず、行を分割し元の CRS 形式の最後の行にデータを追加することで、データ局所性と並列性を高める方式が[12]で提案されている。

6. おわりに

本論文では、疎行列・ベクトル積について Segmented Scan (SS) 法のマルチコア向き実装 BSS 法において、並列性を高め、キャッシュ親和性を高める方式の Blocked BSS 法を提案した。OpenATLib で提供する実装方式を、最大で 128 スレッド実行が可能な高スレッド環境 HITACHI SR16000/VL1 を用いて性能評価を行った。

性能評価の結果、128 スレッド実行時、フロリダ行列を用いた場合、従来の単純な行分割方式に対し、非零要素均等化方式を利用することで最大 63.5% の速度向上が達成できる場合があった。また、ある行が密行となり偏る行列では、従来の BSS 法に対し提案する Blocked BSS 法が 48% の速度向上を達成した。従来の単純な行分割方式に対する Blocked BSS 法の速度向上は 20 倍にも達する。

今後の課題として、NEC SX-9 のようなベクトル計算機で Blocked BSS を評価すること、および OpenATLib の AT 機構へ組み込むことがあげられる。また、MHD コードに組み込み、実用コードで性能評価をすることが最終的な目標である。

謝辞

日頃議論いただく Xabclib プロジェクトの諸氏、東京大学情報基盤センターの大島聡史助教、伊藤祥司特任准教授、黒田久泰准教授 (兼任、本務は愛媛大学)、中島研吾教授、および日立製作所中央研究所の櫻井隆雄氏、猪貝光祥氏、直野健博士に感謝いたします。本研究は、学際大規模情報基盤共同利用・共同研究拠点、公募型共同研究平成 22 年度採択課題、「大規模並列計算における陰的時間積分法を使用した MHD 非線形コードの高速化」による。

参考文献

- [1] Jeff Bilmes, Krste Asanovic, Chee-Whye Chin and Jim Demmel, "Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology", Proceedings of the 11th international conference on Supercomputing, 1997.

- [2] R. Clint Whaley, Antoine Petitet and Jack J. Dongarra, "Automated empirical optimizations of software and the ATLAS project", *Parallel Computing*, Vol. 27, Issues 1-2, pp. 3-35, January 2001.
- [3] Matteo Frigo, "A Fast Fourier Transform Compiler", *ACM SIGPLAN Notices*, Vol.34, Issue 5, pp. 169-180, May 1999.
- [4] Richard Vuduc, James W Demmel, and Katherine A Yelick, "OSKI: A Library of Automatically Tuned Sparse Matrix Kernels", *SciDAC 2005 Proceedings (Journal of Physics)*, San Francisco, CA, United States, June 26 - Jun 30, 2005.
- [5] 櫻井隆雄, 直野健, 片桐孝洋, 中島研吾, 黒田久泰, "OpenATLib : 数値計算ライブラリ向け自動チューニングインターフェース", *情報処理学会論文誌 : ACS*, Vol.3, No.2, pp.39-47, 2010.
- [6] Guy E. Blelloch, Michael A. Heroux, Marco Zagha, "Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors", *CMU-CS-93-173*, August 1993.
- [7] 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", *情報処理学会研究報告*, Vol.2010-HPC-125, No.2, 2010.
- [8] OpenATLib β 版, PC クラスタコンソーシアム, SCore Cluster System Version 7 download page, <http://www.pccluster.org/ja/score7.html>
- [9] The University of Florida Sparse Matrix Collection <http://www.cise.ufl.edu/research/sparse/matrices/>
- [10] Shubhabrata Sengupta, Mark Harris, Michael Garland, "Efficient Parallel Scan Algorithms for GPUs", *NVIDIA Technical Report NVR-2008-003*, Dec. 2008.
- [11] 大島聡史, 櫻井隆雄, 片桐孝洋, 中島研吾, 黒田久泰, 直野健, 猪貝光祥, 伊藤祥司, "Segmented Scan 法の CUDA 向け最適化実装", *情報処理学会研究報告*, Vol.2010-HPC-126, No.1, 2010.
- [12] 小川裕佳, 田邊昇, 高田雅美, 城和貴, "機能メモリと GPU の PCI express 接続によるヘテロ環境における超大規模疎行列ベクトル積の性能予測", *情報処理学会研究報告*, Vol.2010-HPC-126, No.20, 2010.