Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

GPUにおける格納形式自動選択による疎行列ベクトル積の高速化

久保田　悠司[†1]　　高　橋　大　介[†1]

近年，科学技術計算の分野で GPGPU が注目されている．科学技術計算では，特に疎行列ベクトル積を用いることが多いため，疎行列ベクトル積の高速化が重要である．疎行列には多くの格納形式があるが，疎行列によって最適な格納形式は異なる．そこで，本研究では与えられた疎行列によって最適な格納形式に変換してから，疎行列ベクトル積を行うことで高速化を図る．まず予備実験として，いくつかの疎行列の格納形式について，疎行列ベクトル積を実装し，実行速度を測定した．その後，予備実験の結果をもとに自動選択するためのパラメータを決定し，自動選択アルゴリズムを実装する．また，実装したアルゴリズムを，反復法による連立一次方程式の求解を用いて評価した．その結果，多くの疎行列において最適な格納形式を選択し高速化することに成功した．

Optimization of Sparse Matrix-Vector Multiplication by Auto Selecting Storage Schemes on GPU

Yuji Kubota[†1] and Daisuke Takahashi[†1]

Sparse matrix vector multiplication is one of the most often used functions in scientific and engineering computing. The storage schemes for sparse matrices have been proposed, however, each sparse matrices have an optimal storage scheme, In this paper, we propose an auto-tuning algorithm of sparse matrix vector multiplication by selecting storage schemes automatically on GPU, We evaluated our algorithm using Conjugate Gradient solver. As a result, we found that our algorithm was effective in many sparse matrices.

†1 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

1. はじめに

近年，GPU（Graphics Processing Unit）をより汎用的な計算に用いる GPGPU（General-Purpose computation on GPU）が注目されている．GPU は元々グラフィック演算を専門とするアクセラレータであるが，GPU の高い演算性能をグラフィック演算だけでなく，より汎用的で自由度の高い計算に用いるのが GPGPU である．

GPU は複数のコアを搭載しており，単一のコアでは複雑な命令は実行できないが，単純な命令を実行できるコアを多く搭載することで演算性能を高めている．また，GPU はそれぞれのコアで，同一の命令を行うスレッドを同時に複数個実行する事ができる SIMT（Single Instruction Multiple Thread）アーキテクチャである．そのため，同一の命令を何度も繰り返し実行するような演算や，N 体問題や行列積などのメモリアクセス量に対して演算量の多い計算に適している．これらの計算は GPU 上で高い演算性能を出すことができる．

一方で，科学技術計算に用いられる連立一次方程式では，疎行列を対象とするものも多い．この場合，反復法により解を求めることになり，疎行列ベクトル積が何度も呼び出されることになるため，この疎行列ベクトル積の高速化が求められる．疎行列とは 0 を多く含んだ行列であり，この零要素を省いて格納することで効率的に計算を行うことができる．この格納の形式には様々なものが提案されているが[1)]，対象となる疎行列によって各格納形式での疎行列ベクトル積の性能に差が生じる．疎行列ベクトル積は演算量よりもメモリアクセス量が多い演算であるため，GPU の性能を発揮させるためには最適な格納形式を選択することが重要である．

そこで本研究では，入力として与えられた疎行列によって，最適な格納形式を自動選択することで疎行列ベクトル積を高速化する．まず予備評価としてさまざまな格納形式での疎行列ベクトル積を実装し，評価を行う．その後，予備評価の結果を用いて最適な格納形式の選択のために必要なパラメータを，非零要素数や非零要素のパターンなどの疎行列の性質から求め，自動選択アルゴリズムを実装し，評価を行う．評価としては，反復法を用いた連立一次方程式の求解アルゴリズムに自動選択アルゴリズムを適用し，性能を測定する．

2. 関連研究

関連研究としては，まず梶山らによる行列計算ライブラリインターフェース SILC（Simple Interface for Library Collections）[2)] が挙げられる．この梶山らの研究では，CPU 上で格納形式を変換してから計算を行うことで高速化を行っている．

Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

また，Cevahir らの研究[3] では，GPU クラスタを用いて共役勾配法を行う際の行列ベクトル積の最適化を行っている．共役勾配法を解く前に，複数の格納形式の行列ベクトル積を数回実行し，最適な格納形式を選択する．疎行列ベクトル積を数回実行するための時間は共役勾配法を解く時間に比べ短いことから，最適化した後に計算することにより全体の計算時間は短くなる．本研究の目的は，様々な種類の格納形式の中から入力として与えられた疎行列に最適な形式を自動的に選択することであり，その点でこの Cevahir らの研究とは異なる．また，共役勾配法では前処理によって反復回数が減少するものがある．そのような疎行列では，共役勾配法全体の実行時間に対し，疎行列ベクトル積を複数回行うオーバヘッドの占める割合が大きくなってしまう．本研究で提案するアルゴリズムでは，前処理によって反復回数が減少した疎行列に対しても高い性能を出すことが可能であると考えられる．

3. 疎行列ベクトル積

ここでは，本研究で用いた疎行列の格納形式と，GPU を用いたそれぞれの格納形式での疎行列ベクトル積の実装について述べる．実装は NVIDIA 社が提供している開発環境 CUDA（Compute Unified Device Architecture）[4] を用いた．また，以下の小節では式（1）で示す行列 A（行列サイズ $n = 4$，非零要素数 $nz = 8$）を例として説明する．

$$A = \begin{bmatrix} 4 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 2 & 0 & 6 & 3 \\ 0 & 1 & 0 & 5 \end{bmatrix} \quad (1)$$

3.1 CRS 形式

CRS（Compressed Row Storage）形式とは，疎行列を行方向に走査し，零要素を省いた格納形式である．この CRS 形式は格納が容易なため，最も使われている形式の 1 つである．**図 1** に，行列 A を CRS 形式で格納した場合の例を示す．図 1 中の配列 val は非零要素の値，配列 $colind$ は非零要素の列番号を格納する配列である．また配列 $rowptr$ は，val と $colind$ における各行の先頭の非零要素の添字を表している．

次に CUDA を用いた CRS 形式での疎行列ベクトル積の実装について説明する．CRS 形式での疎行列ベクトル積の実装の方法としては，行分割方式や非零要素分割方式などがあるが[5]，ここでは単純な行分割方式について説明する．単純な行分割方式では，疎行列を行毎に区切り，それぞれを 1 つのスレッドが計算する．**図 2** に，各スレッドが実行する擬似コードを示す．図 2 の配列 x はベクトルの値が格納された配列，配列 y は出力ベクトルの配列，変数 id はスレッド番号を表している．

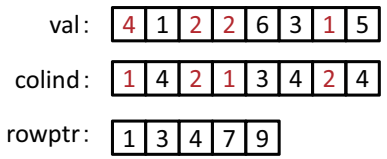

図 1 行列 A の CRS 形式での格納

```
out = 0
for i = rowptr[id] − 1 to rowptr[id + 1] − 1 do
  out = out + val[i] × x[colind[i] − 1]
end for
y[id] = out
```

図 2 CRS 形式での行列ベクトル積のカーネルコード

また，本研究では CRS 形式の疎行列ベクトル積として，NVIDIA 社が提供しているライブラリ CUSPARSE[6] を用いる．これは，本研究で実装した CRS 形式の疎行列ベクトル積よりも，CUSPARSE での CRS 形式の疎行列ベクトル積が非常に高速であったためである．この CUSPARSE では，CRS 形式での疎行列ベクトル積や行列積，いくつかの格納形式の変換などが実装されている．

3.2 JDS 形式

JDS（Jagged Diagonal Storage）形式では，前述した CRS 形式よりも複雑な操作が必要になる．**図 3** に，行列 A を JDS 形式で格納した例を示す．まず，行列 A を行方向に詰め（図 3 左図），次に 1 行あたりの非零要素が多い順に並べ替える（図 3 中央図）．その後，図 3 右図のように，列方向に走査し 4 つの配列と 1 つの変数に格納する．

次に JDS 形式での疎行列ベクトル積の実装について説明する．JDS 形式では，1 個のスレッドが行列ベクトル積の出力ベクトルの 1 要素を計算する．**図 4** に，各スレッドが実行するカーネルの擬似コードを示す．

まず，配列 ptr と変数 max から，計算に必要な配列 val と $colind$ の範囲を求める．その後，それぞれのスレッドで計算する行に含まれる非零要素とそれに対応するベクトルとの積の和を求める．最後に，配列 $perm$ を参照し出力ベクトルの対応するアドレスに格納する．

この JDS 形式では配列 val と配列 $colind$ にアクセスする際に，連続する番号のスレッド

Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

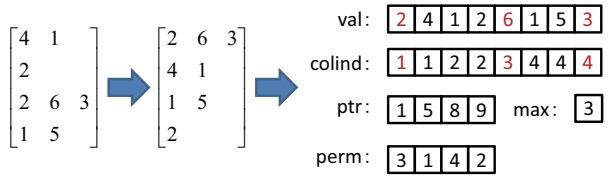

図 3 行列 A の JDS 形式での格納

```
out = 0
for i = 0 to max do
  if ptr[i] − 1 + id < ptr[i + 1] − 1 then
    ind = ptr[i] − 1 + id
    col = colind[ind] − 1
    out = out + val[ind] × x[col]
  else
    break
  end if
end for
y[perm[id] − 1] = out
```

図 4 JDS 形式での行列ベクトル積のカーネルコード

が連続するメモリ領域にアクセスしている．GPU では，グローバルメモリへのアクセスはある一定のバイト数毎に行われている．そのため，1 回のメモリアクセスで複数のスレッドが同時に参照することで，メモリアクセスの回数を削減できる．

また，1 行あたりの非零要素数が大きく異なる場合，スレッド毎に実行速度が異なってしまう．疎行列ベクトル積全体の実行時間は，実行時間が最も遅いスレッドに律速されるため，JDS 形式では十分な性能を発揮できない場合が存在する．

3.3 ELL 形式

ELL（ELLPACK/ITPACK）形式[7] は，疎行列を $n \times k$ の密行列に格納する．ここで n は元の疎行列の一辺のサイズ，k は疎行列 1 行あたりの非零要素数の最大値を表している．また，1 行あたりの非零要素数が k に満たない場合は，0 をパディングする．**図 5** に ELL 形式での格納の例を示す．また，配列 $colind$ では，0 ではなく 1 をパディングする．これは，疎行列ベクトル積を求める時に，$colind[ind] - 1$ という計算を行い，必要となるベクトルの値を参照するためである．

次に，ELL 形式での疎行列ベクトル積の実装について説明する．ELL 形式では，JDS 形式と同様に 1 個のスレッドが出力ベクトルの 1 要素を計算する．**図 6** にカーネルの擬似コードを示す．

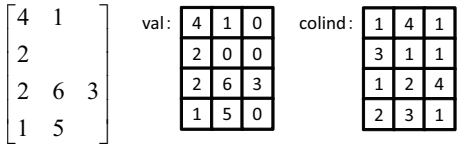

図 5 行列 A の ELL 形式での格納

```
out = 0
for i = 0 to max do
  ind = n × i + id
  col = colind[ind] − 1
  out = out + val[ind] × x[col]
end for
y[id] = out
```

図 6 ELL 形式での行列ベクトル積のカーネルコード

ELL 形式では JDS 形式と比べると分岐命令が減っていることがわかる．GPU では，同時に実行されるスレッド内で分岐の方向が異なる場合，両方の分岐処理を行うため時間を要する．これをウォープダイバージェントという．このウォープダイバージェントが発生する可能性があるため，分岐命令は GPU において性能の低下の原因となる場合がある．また，ELL 形式ではすべてのスレッドで計算する要素数が等しくなっている．これらのことから，ELL 形式は JDS 形式よりも GPU に適していると考えられる．しかし，1 行あたりの非零要素数に大きなばらつきがある場合，無駄な計算が多くなってしまうという問題点がある．

3.4 SS 形 式

SS（Segmented Scan）形式[8] では，疎行列をある一定の長さのセグメントベクトルに分割して格納する．この SS 形式を CUDA 向けに最適化したものが，大島らのインデックスを用いた SS 形式である[9]．以降インデックスを用いた SS 形式を単に SS 形式と呼ぶ．これまで述べてきた格納形式では，1 行あたりの非零要素数に大きな差があると，性能が低い場合がある．しかし，SS 形式では 1 行あたりの非零要素数に大きな差がある場合でも十分な性能を発揮することができる．**図 7** に SS 形式での格納の例を示す．この例では，セグメントベクトル長が 4 となっている．SS 形式では，非零要素数を列方向に走査し，セグメントベクトルに格納していく．また，配列 $index$ は元の疎行列の行頭要素からの番号，もしく

Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

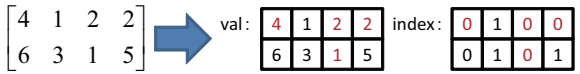

図 7 行列 A の SS 形式での格納

```
counter = tnum/2
while counter > 0 do
  if (index[id] >= counter)&&(index[id] < counter × 2) then
    sval[id − counter] = sval[id − counter] + sval[id]
    sval[id] = 0.0
  end if
  __syncthreads()
  counter = counter/2
end while
```

図 8 SS 形式でのインデックスを用いた加算の擬似コード[9)]

は，セグメントベクトルの先頭要素からの番号を格納する．

次に，SS 形式での疎行列ベクトル積について述べる．SS 形式では，1 つのスレッドブロックが SS 形式に変換された後の 1 行を計算する．まず，各スレッドが非零要素とそれに対応するベクトルの要素との積を求め，シェアードメモリに格納する．その後各スレッドブロック毎にインデックスを用いて加算を行う．この加算のコードを**図 8** に示す．

図 8 中の $tnum$ は 1 ブロックあたりのスレッド数（ここでは 2 のべき乗としている），配列 $sval$ はシェアードメモリ上の配列サイズ $tnum$ である配列を表している．すべてのスレッドブロックで加算が終了した後，インデックス番号 0 に格納されている計算結果を出力ベクトルの対応する要素に書き込む．ただし，SS 形式では元の疎行列の 1 行が複数の行にまたがって格納されている場合があるため，同時に書き込みが起こらないように注意する．

4. 予備評価

4.1 評価環境

表 1 に，評価環境を示す．疎行列は，University of Florida Sparse Matrix Collection[10)] から入手した 30 個の疎行列を用いた．評価に用いた疎行列の一部を**表 2** に示す．表 2 のばらつきとは，1 行あたりの非零要素数のばらつきを表しており，1 行あたりの非零要素数の最大値を max，1 行あたりの非零要素数の平均を ave とすると，max/ave で求められる．この値が大きいと，1 行あたりの非零要素数に差が大きく，1 に近いほど差が少ない．また，

表 1 評価環境

CPU	AMD Opteron 6134 × 2
メインメモリ	4GB
GPU	NVIDIA Tesla C2050
ビデオメモリ	3GB
理論ピーク性能	515GFLOPS（倍精度）
開発環境	CUDA SDK 3.2 RC2
コンパイラ	nvcc （-O3 -arch=sm_20） version 3.2

表 2 予備評価に用いた疎行列の一部

疎行列の名前	一辺の長さ	非零要素数	ばらつき	非零要素率（%）
s3dkt3m2	90449	3686223	1.031	0.04506
af_shell9	504855	17588845	1.148	0.00690
tmt_sym	726713	5080961	1.287	0.00096
nd24k	72000	28715634	1.304	0.55393
ldoor	952203	42493817	1.725	0.00469
m_t1	97578	9753570	2.371	0.10244
pwtk	217918	11524432	3.404	0.02427
F1	343791	26837113	5.572	0.02271
bmw3_2	227362	11288630	6.767	0.02184
Ga3As3H12	61349	5970947	16.665	0.15865

非零要素率とは，行列全体に対する非零要素の占める割合を表している．

評価では，GPU 上のメモリの確保やデータの転送時間は含めず，カーネルの実行時間のみを測定した．この理由としては，反復法などの疎行列ベクトル積を複数回用いるアプリケーションを想定しているためである．また，本研究の評価はすべて倍精度演算を用い，FLOPS を算出し比較する．この FLOPS は，疎行列の非零要素数を nz，計算に要した実行時間を $time$ とすると，$nz \times 2/time$ で求めることができる．

4.2 各格納形式による疎行列ベクトル積の評価

予備評価として，30 個の疎行列全てにおいて各格納形式による疎行列ベクトル積の性能を測定した．ただし，いくつかの疎行列については，GPU でのメモリ不足により ELL 形式での測定ができなかった．**図 9** に表 2 で挙げた疎行列での評価結果を示す．

評価結果より，疎行列によって最適な格納形式が異なっていることがわかった．主に，非零要素のばらつきが大きい時では CUSPARSE（CRS 形式）が高速で，ばらつきが小さい時には ELL 形式が高速であった．これは，ELL 形式ではばらつきが大きい疎行列を格納するとき，0 でパディングする部分が多くなってしまうため，無駄な計算が多くなることが原

Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

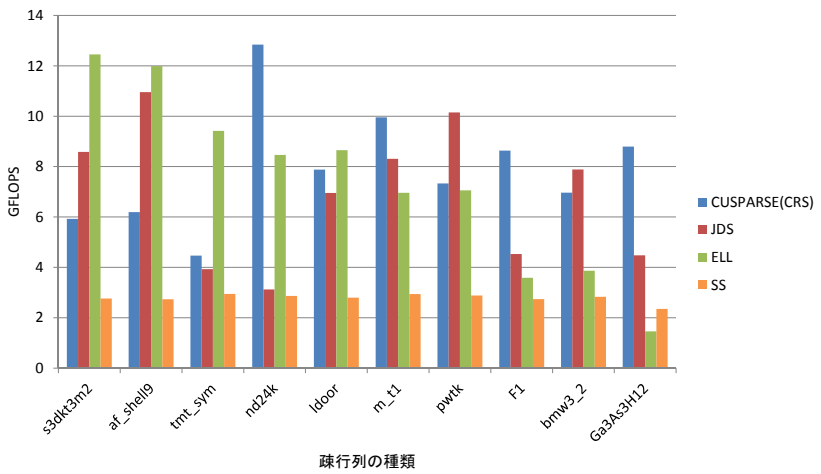

図 9 各格納形式での疎行列ベクトル積の実行速度

因であると考えられる．また，SS 形式では疎行列による性能の差は小さかったが，全体的に低い性能であった．この原因としては，FMA（Fused Multiply Add）命令を用いていないからであると考えられる．FMA 命令は，積和演算 $a \times b + c$ を 1 命令で実行可能であるため，高速な計算が可能となっている．しかし，SS 形式では一度すべての非零要素に対し，乗算を行った後にインデックスを用いた加算を行っている．そのため，FMA 命令が用いられず，他の格納形式と比べ性能が低下していると考えられる．

5. 自動選択のためのパラメータ決定

予備評価の結果から，格納形式の自動選択を行うためのパラメータを決定する．予備評価の結果をまとめると，図 9 と表 2 からもわかるように，各格納形式での疎行列ベクトル積の実行速度の違いは主に非零要素のばらつきにあるのではないかと考えられる．そこで，本研究では自動選択のためのパラメータとして，まず非零要素のばらつきを用いる．予備評価の結果を解析したところ，非零要素のばらつきが 2 以下の時は ELL 形式，8 以上の時は CUSPARSE（CRS 形式），その間の時は JDS 形式が高速であるということがわかった．しかし，CUSPARSE は本研究で実装した CRS 形式での疎行列ベクトル積よりも高速であることから，他の格納形式での疎行列ベクトル積の実装より CUSPARSE の実装が優れている可能性がある．そのため，CUSPARSE（CRS 形式）での疎行列ベクトル積が高速である場合においても，その疎行列における最適な格納形式が CRS 形式でない場合もあると考えられる．この点については，今後各格納形式での疎行列ベクトル積の実装を最適化し，検討していきたい．

表 3 CRS 形式から各格納形式への変換に要する時間と疎行列ベクトル積（SpMV）に要する時間（sec）

	JDS		ELL		SS	
疎行列の名前	SpMV	変換	SpMV	変換	SpMV	変換
s3dkt3m2	0.000859	0.049256	0.000592	0.012649	0.002668	0.029562
af_shell9	0.003211	0.267710	0.002934	0.059335	0.012878	0.137890
tmt_sym	0.002585	0.280375	0.001079	0.009658	0.003448	0.018470
nd24k	0.018401	0.170978	0.006787	0.093085	0.020029	0.242276
ldoor	0.012218	0.540457	0.009824	0.157335	0.030399	0.344673
m_t1	0.002347	0.087594	0.002803	0.038037	0.006634	0.082242
pwtk	0.002271	0.154539	0.003267	0.048457	0.007996	0.094341
F1	0.011855	0.356287	0.014963	0.139352	0.019605	0.224287
bmw3_2	0.002863	0.194000	0.005839	0.061181	0.007971	0.092064
Ga3As3H12	0.002666	0.260357	0.008171	0.058018	0.005075	0.049721

また，いくつかの疎行列についてはこれに当てはまらないものがある．そこで，今度は疎行列の非零要素率に着目する．予備評価の結果と非零要素率から，前述の条件に合わない疎行列の多くにおいて，非零要素率が他のものよりも大きくなっていることがわかった．以上のことを考慮し，自動選択のパラメータとしては非零要素のばらつきと非零要素率を用いる．自動選択の条件を以下に示す．また，以下の条件で最適な格納形式を選択できた疎行列は，予備評価として用いた 30 個の疎行列のうち 28 個である．

$$\begin{cases} \text{ELL 形式} & (\text{ばらつき} < 2.0 \text{ かつ 非零要素率} < 0.048) \\ \text{CUSPARSE（CRS 形式）} & (\text{ばらつき} > 8.0 \text{ または 非零要素率} \geq 0.048) \\ \text{JDS 形式} & (\text{上記以外}) \end{cases} \quad (2)$$

また，本研究では最もよく使われていると考えられる CRS 形式の疎行列を入力とし，そこからパラメータによって最適な格納形式へ変換することで高速化することを目的としている．そのため，最適な格納形式の選択には，変換に要する時間も考慮する必要があると考えられる．**表 3** に，CRS 形式から各格納形式への変換に要する時間と各格納形式での疎行列ベクトル積に要する時間を示す．

格納形式の変換には疎行列ベクトル積よりも多くの時間を要する．しかし，本研究では反

Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

復法などの疎行列ベクトル積を何度も呼び出す場合を想定している．そのため，最適な格納形式を選択し疎行列ベクトル積を高速化することで，変換に要する時間よりも高速化による実行時間の軽減が大きくなる．よって，今回は格納形式の変換に要する時間は自動選択のためのパラメータには考慮しない．

6. 評価実験

6.1 連立一次方程式の求解を用いた評価

実装した自動選択アルゴリズムを反復法を用いた連立一次方程式 $Ax = b$ の求解を用いて評価する．行列 A は CRS 形式の疎行列とし，ベクトル x は求める解ベクトル，ベクトル b は解ベクトルの要素がすべて 1 となるように求めたベクトルを用いる．また，反復法のアルゴリズムは様々なものがあるが，今回は Bi-CGSTAB 法を用い，前処理は行わない．反復回数は 100 回と 500 回の 2 通り行った．評価環境は予備評価と同様である．また，疎行列は予備評価で用いた 30 個の疎行列を用いた．

この評価では，まず CRS 形式の疎行列を入力とし，前節で決定したパラメータに従い最適な格納形式に変換する．その後反復法を用いて解ベクトルを求める．時間の測定は，最適な格納形式の選択の時間や，CRS 形式から各格納形式への変換の時間も含めた全体の実行時間を測定する．

評価は，CRS 形式のまま CUSPARSE を用いて反復法を実行した場合と CRS 形式からそれぞれの格納形式へ変換した場合，本研究で実装した自動選択に基づいて格納形式を変換した場合に加え，文献 3) を参考に実装したアルゴリズム（exam）を測定した．このアルゴリズムでは，反復法を行う前にそれぞれの格納形式で疎行列ベクトル積の性能を測定し，それに基づき最適な格納形式を選択し，反復法を行うといったものである．また，CRS 形式以外の格納形式で疎行列ベクトル積の性能を測定した後は，GPU のメモリ容量の制約により，一旦その格納形式のデータを破棄してから，他の格納形式での性能を測定する．その後，測定した結果に基づき，最適な格納形式に再び変換してから，反復法を行う．

評価として用いた 30 個の疎行列のうち表 2 に示した疎行列について，反復回数が 100 回の時の結果を**図 10**，反復回数が 500 回の時の結果を**図 11** に示す．この評価結果は，CRS 形式のまま CUSPARSE を用いて実行した場合に対する，CRS 形式から各格納形式へ変換し実行した場合の速度向上率を表している．

この結果より，ほとんどの疎行列で最適な格納形式を選択し，CRS 形式のまま CUSPARSE を用いて実行するのと同等，またはそれ以上の高速化ができたことがわかる．また，反復回数が 100 回の時と比べ 500 回の時では，自動選択した場合と反復法の前に疎行列ベクトル積を数回実行した場合との差が小さくなっていることがわかる．これは，反復回数が多くなるに従って，反復法の前に疎行列ベクトル積を数回実行する時間の割合が，全体の実行時間に比べ小さくなるためである．反復回数が少ない時ではこの時間の割合が多くなり，最適な格納形式を選択出来ていても低い性能になってしまう場合があった．これに対し，本研究の自動選択では反復回数が少ない時でも，最適な格納形式を選択し，高速化することができた．

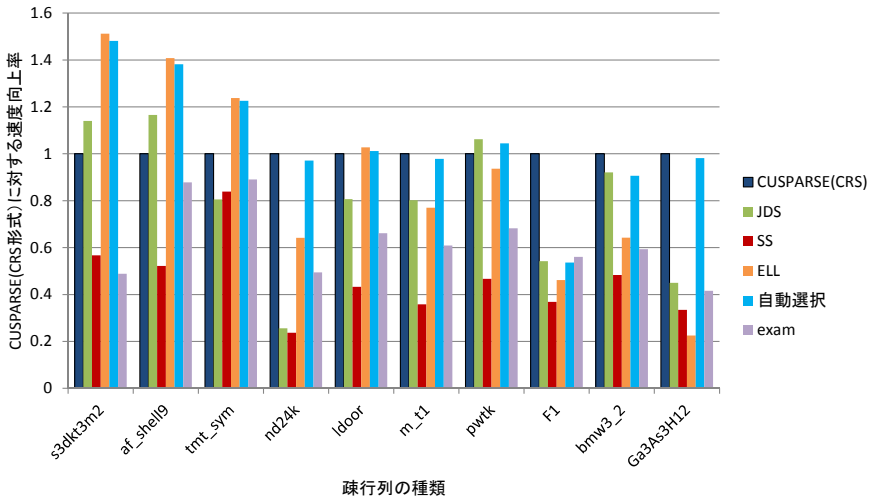

図 10 連立一次方程式を用いた評価結果（反復回数 100 回）

6.2 考　　察

連立一次方程式を用いた評価では，今回評価として用いた 30 個の疎行列のうち 28 個の疎行列において最適な格納形式を選択し，高速化することができた．しかし，いくつかの疎行列では自動選択で選択された格納形式よりも，高速であった格納形式があったため，本研究で用いた自動選択のパラメータでは不十分であるということがわかる．そのため，正しく自動選択するためには，より正確な条件が必要である．また，自動選択のためのオーバヘッドはほとんどなかった．これは，本研究で用いた自動選択のパラメータである非零要素のばらつきと非零要素率は，求めるために要する時間が少ないためである．さらに，非零要素のばらつきを求めるために必要な 1 行あたりの非零要素数の最大値 max は，変換した後の格

Vol.2010-ARC-192 No.19
Vol.2010-HPC-128 No.19
2010/12/17

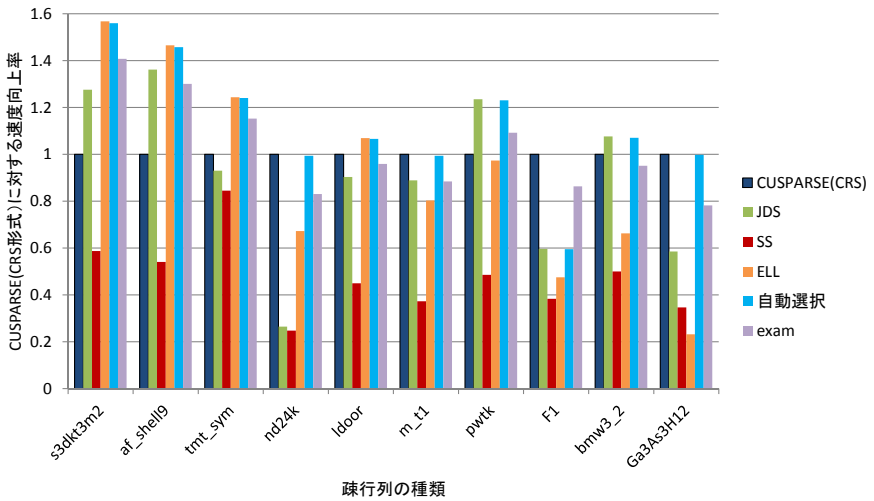

図 11 連立一次方程式を用いた評価結果（反復回数 500 回）

納形式にも用いられるため，この max を求める計算時間も無駄にならなかった．

7. ま と め

本研究では，疎行列の格納形式を自動選択することで疎行列ベクトル積の高速化を行った．まず，予備評価として４種類の格納形式での疎行列ベクトル積を実装し，性能を評価した．その結果，非零要素のばらつきと非零要素率によって各格納形式での疎行列ベクトル積の性能に違いが生じることがわかった．この予備評価の結果を用いて，最適な格納形式を自動選択するためのアルゴリズムを実装し，反復法を用いた連立一次方程式の求解を用いて評価を行った．評価の結果，今回用いた疎行列のうち多くの疎行列において最適な格納形式を選択し，CRS 形式のまま CUSPARSE を用いて実行する場合と同等，またはそれよりも高速であることを示した．また，特に反復回数が少ない時でも本研究で提案した自動選択が有効であることを示した．

今後の課題としては，対応できる格納形式の追加や格納形式変換アルゴリズムや各格納形式での疎行列ベクトル積の高速化，より多くの疎行列での評価などが挙げられる．本研究では４種類の格納形式について評価を行ったが，他にも SKYLINE 形式[1] や複数の格納形式を組み合わせたハイブリッド形式[11] などがあるため，それらの格納形式についても評価を行いたい．また，本研究ではメモリアライメントやブロックあたりのスレッド数などの最適化については行っていないため，それらを考慮することでさらなる高速化が可能であると考えられる．最後に，より多くの疎行列での評価が挙げられる．今回用いた疎行列だけでなく，より多くの疎行列について評価を行い，より正確な自動選択のためのパラメータを決定していきたい．

謝辞 本研究の一部は，文部科学省科学研究費補助金「新学術領域研究」（課題番号 22104003）による．

参 考 文 献

1) Saad, Y.: SPARSKIT: a basic tool kit for sparse matrix computations - Version 2 (1994).
2) 梶山民人，額田彰，須田礼仁，長谷川秀彦，西田晃：SILC: 行列計算ライブラリの利用を簡単化するフレームワーク，第 10 回日本計算工学会講演会論文集， Vol.10, pp. 239–242 (2005).
3) Cevahir, A., Nukada, A. and Matsuoka, S.: CG on GPU-enhanced Clusters, 情報処理学会研究報告， Vol.2009-HPC-123, No. 15, pp.1–8 (2009).
4) NVIDIA: NVIDIA GPU Computing Developer Home Page, `http://developer.nvidia.com/object/gpucomputing.html`.
5) 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, pp.1–8 (2010).
6) NVIDIA: CUSPARSE User Guide, `http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUSPARSE_Library.pdf`.
7) Kincaid, D.R., Oppe, T.C. and Young, D.M.: ITPACKV 2D User's Guide, Technical report, University of Texas (1989).
8) Blelloch, G.E., Heroux, M.A. and Zagha, M.: Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors, Technical report, Carnegie Mellon University (1993).
9) 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, pp.1–7 (2010).
10) T.A.Davis: University of Florida Sparse Matrix Collection, `http://www.cise.ufl.edu/research/sparse/matrices/`.
11) Bell, N. and Garland, M.: Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation (2008).