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

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の性能は行列の構造や計算機の性質に大きく依存するため，これらの性質に合わせた高速化手法で計算することが重要である．本論文では筆者らの作成した複数の高速化手法を含んだ並列疎行列ベクトル積のルーチンを紹介する．また行列や計算機の性質に応じて最適な高速化手法を選択した場合の性能を，従来法である妥当な固定手法を用いた場合の性能と比較する．4 機種の並列計算機上での実験では，最適なアルゴリズム選択により平均して 1.29 倍の速度向上が得られた．

The Effect of Optimal Algorithm Selection of Parallel Sparse Matrix-Vector Multiplication

Makoto Kudoh,[†] Hisayasu KURODA,[††]
Takahiro KATAGIRI[†††] and Yasumasa KANADA[††]

The computational performance of parallel sparse matrix-vector multiplication (SpMxV) depends highly on the non-zero structure of the target matrix and the nature of machine's architecture. Therefore it is important to select the best optimization method according to these characteristics. In this paper, our parallel SpMxV routine which includes several optimization algorithms is described, and the performance with optimal algorithm is compared to that with uniform default algorithm on 4 kinds of parallel machines. The average speed-up of 1.29 is obtained.

1. はじめに

疎行列ベクトル積 (SpMxV) は，科学技術計算を始めとする多くのアプリケーションにおいて重要な計算カーネルである．最近では並列計算機の普及に伴って，並列 SpMxV が重要視されてきている．今までに多くの並列 SpMxV の高速化アルゴリズムが提案されてきたが[1)~7)]，これらの効果は対象の行列の非零構造や計算機のアーキテクチャに大きく依存し，最適な高速化アルゴリズムは行列や計算機の性質によって異なる．この理由から，高速化アルゴリズムをこれらの性質に合わせて選択することが重要であると我々は考える．従来の手法では，すべての行列と計算機に対して固定した妥当なアルゴリズムを適用する方法が多い．また最適なアルゴリズムを選択したとしても，並列環境において十分なものではない．本稿の目的は，並列 SpMxV において行列や計算機の性質に合わせた高速化アルゴリズムを用いることの効果を示すことである．

SpMxV では，右辺ベクトルに対して不規則なデータアクセスパターンになるために一般的に密行列ベクトル積よりもパフォーマンスが悪い．そのためデータアクセスパターンを最適化して SpMxV の計算カーネルを高速化することが重要である．一方並列環境では，右辺ベクトルのデータをプロセッサ間で通信する必要があり，低速なネットワークで結ばれた環境ではこの通信時間が無視できない．したがって，ローカルな計算と通信との両方を最適化することが重要である．

我々は計算部分と通信部分において様々な高速化アルゴリズムを用いて実装されている並列 SpMxV ルーチンを作成した．このルーチンでは，それぞれのアルゴリズムの時間を実行時に計測して最適なアルゴリズムを選択する，という手法でパフォーマンスの向上を狙った．最適なアルゴリズムは行列の特性によって変化するので，実行時にアルゴリズムを選択する機構はハイパフォーマンスを達成するために不可欠である．本稿では我々のルーチンの性能と従来手法の固定アルゴリズムを用いた場合の性能を比較することで，我々

† 東京大学大学院情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, the University of Tokyo
†† 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
Computer Centre Division, Information Technology Center, the University of Tokyo
††† 科学技術振興事業団, さきがけ研究 21
PRESTO, Japan Science and Technology Corporation (JST)

のアプローチの効果を示す．ただし，本報告の目的は最適化の可能性を示すことにあるので，アルゴリズム選択の時間については考慮していない．

1.1 関連研究

現在，SpMxV の高速化手法にはライブラリ型とコンパイラ型の 2 種類の取り組み方がある．ライブラリ型は SpMxV ルーチンをユーザが使うメソッドとして提供するものである．このアプローチには，PSPARSLIB[8)]，PETSc[9)]，ILIB[10)] などが知られている．PSPARSLIB と PETSc は高速化の手法を実装しているが，これらの高速化はすべての行列や計算機に対して固定されているか，またはユーザが指定するようになっている．ILIB ではどの高速化手法を使うかを実行時に選択するようになっているが，SpMxV の高速化手法に関する部分の実装は少ない．

コンパイラ型は，対象の行列に最適化された SpMxV のプログラムコードを実行前に生成するものであり，SPARSITY[1)] や *sparse compiler*[5)] などがある．このアプローチでは行列や計算機の性質に合わせた最適化を行うことができるが，並列環境で通信の最適化を行う機構がない．

本稿で示す研究はライブラリ型の基本研究であるといえる．

2. 実装したアルゴリズム

行列ベクトル積 $y = Ax (A \in \Re^{n\times n}, x \in \Re^n, y \in \Re^n)$ を考える．ここで A は疎行列，x, y は密なベクトルである．並列環境では，次の 2 つの処理が必要となる．

1 ベクトル x のデータをプロセッサ間で通信する．
2 各プロセッサで行列ベクトル積を計算する．

本章では，我々が実装したアルゴリズムについて説明する．

2.1 データ構造

疎行列を格納するデータ構造については，いくつかの方法が提案されているが，我々は行圧縮形式 (Compressed Sparse Row Format) を採用した (図 1)．また行列データの分散方法は行ブロック分散 (図 2) とした．

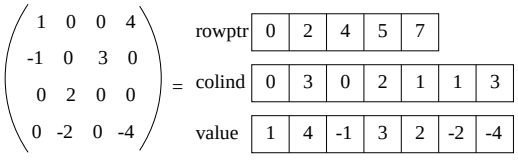

図 1 行圧縮形式
Fig. 1 The compressed sparse row format

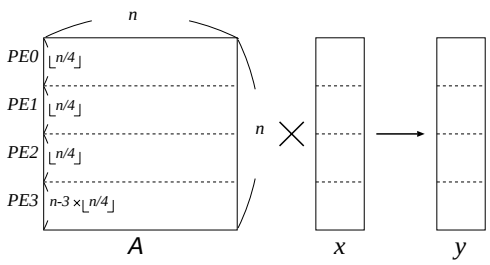

図 2 行ブロック分散
Fig. 2 Row block distribution

2.2 プロセッサ内計算のアルゴリズム

2.2.1 Register Blocking

疎行列の中の 2×2 や 3×3 の小さな密ブロックを見つけ出して，それをブロック化して計算することにより，レジスタ使用を最適化し，ロード命令を減少させることができる．この方法は Register Blocking と呼ばれる (図 3)[1)~3),6),7)]．疎行列はこのブロッキングによりブロック化されたものと，ブロック化されなかった余りの行列との 2 つに分割される．

$$A = A_{rblock} + A_{remain}. \quad (1)$$

我々のルーチンでは，次の 15 種類のブロックサイズの Register Blocking を用意した．

1×2, 1×3, 1×4, 2×1, 2×2, 2×3, 2×4,
3×1, 3×2, 3×3, 3×4, 4×1, 4×2, 4×3, 4×4

以降では，‘Register blocking $n \times m$’ のことを ‘R$n \times m$’ と表記する．

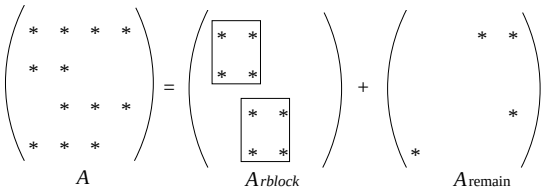

図 3 レジスタブロッキングの例 (2×2)
Fig. 3 An example for the register blocking (2×2)

2.2.2 Diagonal Blocking

疎行列には，対角要素周辺に非零要素が密集しているものが多い．これらの行列の SpMxV を高速にするために，対角要素周辺をブロック化し，その部分が密の帯行列として扱うことで高速化できる．この方法を Diagonal Blocking と呼ぶ．Diagonal Blocking では，ブロック内にある零要素も非零要素として計算するので，浮動小数点計算が増える可能性がある．また我々のルーチンでは，Diagonal Blocking の残りの行列に対して Register Blocking を適用した．この方法により，行列は次の 3 つに分割される．

$$A = A_{dblock} + A_{rblock} + A_{remain} \quad (2)$$

我々のルーチンでは，次の 8 種類のブロックサイズを用意した．

3, 5, 7, 9, 11, 13, 15, 17, 19

ここでブロックサイズは，帯行列の帯幅のことである．以降では，ブロックサイズ n の Diagonal Blocking を ‘D n’ と表記する．

2.2.3 Unrolling

我々のルーチンには，アンローリングした計算カーネルも実装されている．この実装では，最内ループのみがアンローリングされている．本稿で実装したアンローリング段数は，1,2,3,4,5,6,7,8 の 8 種類である．以降では，n 段アンロールのことを ‘U n’ と表記する．

2.3 通信アルゴリズム

並列環境では，SpMxV 計算は右辺ベクトルのデータをプロセッサ間で通信しなくてはならない．この通信時間はプロセッサ数が増えるに従って大きくなる．したがって，通信時間を減らす工夫をすることは並列 SpMxV を高速化するのに重要である．我々のルーチンでは，通信ライブラリに MPI を使用し，次の 3 種類のアルゴリズムを実装した．

2.3.1 Allgather 通信

各プロセッサが他の全てのプロセッサに対して自分が担当する全ベクトルデータを送信する，ナイーブな通信アルゴリズムである．この処理は MPI_Allgather を使って実現される．しかしこの方法では，通信するデータ量は非常に大きくなる．

2.3.2 範囲限定通信

この通信アルゴリズムでは，各プロセッサは通信が必要な最小の連続した範囲のブロックのみを通信する．通信が必要でないプロセッサ間では，通信が全く行われない．しかし通信データの量は，多くの行列で最小とはならない．

2.3.3 最小量通信

この通信アルゴリズムでは，通信が必要な要素のみを通信する．そのため通信データ量は最小である．ただしこの通信を一回の Message Passing で実現するためには，通信前と通信後に通信バッファに詰め込んだり (pack)，元の並びに戻したり (unpack) する作業が必要となる．この処理は重たい処理ではないが，通信量が小さい行列の場合にはこれがオーバーヘッドになる場合がある．

3. 最適アルゴリズムの選択方法

我々のルーチンでは，前章で述べた計算と通信の複数のアルゴリズムを実装しており，この中で最適なものを実行時に選択する．この章では，対象の行列と計算機に最適なアルゴリズムを見つける方法について説明する．

3.1 アルゴリズムセット

我々のルーチンに実装されたプロセッサ内計算アルゴリズムを表 1 に示す．これらのうちの一つが選択される．各プロセッサごとに最適な計算アルゴリズムが異なる場合があり得るので，プロセッサごとに異なる計算アルゴリズムを選択することがある．

表 1 計算アルゴリズム
Table 1 The local computation algorithms

Algorithm	Type
Unrolling	U1,U2,U3,U4,U5,U6,U7,U8
Register blocking	R1×2,R1×3,R1×4, R2×1,R2×2,R2×3,R2×4, R3×1,R3×2,R3×3,R3×4, R4×1,R4×2,R4×3,R4×4
Diagonal blocking	D3,D5,D7,D9,D11,D13,D15,D17,D19

表 2 通信アルゴリズム
MPI_Send,MPI_Recv はブロッキング通信
MPI_Isend,MPI_Irecv は非ブロッキング通信である
Table 2 The communication algorithms
MPI_Send and MPI_Recv are blocking communication, MPI_Isend and MPI_Irecv are non-blocking communication

comm param	Explanation
Allgather	Allgather 通信
SendRecv-range	MPI_Send と MPI_Recv を使う範囲限定通信
IsendIrecv-range	MPI_Isend と MPI_Irecv を使う範囲限定通信 MPI_Isend MPI_Irecv order の順
IrecvIsend-range	MPI_Isend と MPI_Irecv を使う範囲限定通信 MPI_Irecv MPI_Isend の順
SendRecv-min	MPI_Send と MPI_Recv を使う最小量通信
IsendIrecv-min	MPI_Isend と MPI_Irecv を使う最小量通信 MPI_Isend MPI_Irecv の順
IrecvIsend-min	MPI_Isend と MPI_Irecv を使う最小量通信 MPI_Irecv MPI_Isend の順

また実装された通信アルゴリズムは表 2 に示されている．これらの中の最適な一つが通信アルゴリズムとして選択される．

3.2 選択方法

最適アルゴリズムの選択には，2 つの段階がある．最初に通信アルゴリズムを選択し，次に計算アルゴリズムを選択する．通信アルゴリズムの選択では，表 1 の全アルゴリズムの時間を計測し，最も速いものを選択する．次に計算アルゴリズムを選択するが，計算の時間のみを計測して最適なアルゴリズムを選択したのでは，通信と組み合わせた時に多くの場合それが最適ではない．したがって，前に得られた最適な通信アルゴリズムと組み合わせた時間を計算アルゴリズムの時間として計測する．この方法で表 1 のすべての候補の時間を計測し，最速なものを選択する．

なお本稿の目的は，並列 SpMxV が最適なアルゴリズムを選択することによりどれだけ高速化され得るか，を示すことにあるのでアルゴリズムの選択にかかる時間については考慮していない．

4. 性能評価

前章で説明した我々のルーチンを使い，最適なアル

表 3　実験機種
Table 3　Machine's Architectures

Name	Processor	PE	Network
	Compiler Name	Compiler version	Compiler option
PC-cluster	PentiumIII 800 MHz	8	100 base-T Ethernet
	gcc	2.95.2	-O3
COMPAQ AlphaServer GS80	Alpha 21264 731 MHz	8	SMP
	Compaq C	6.3-027	-fast
SGI 2100	MIPS R12000 350 MHz	8	DSM
	MIPSpro C	7.30	-64 -O3
HITACHI HA8000	Itanium 733 MHz	8	SMP
	Intel Itanium Compiler	5.0.1	-O3

表 4　実験行列
Table 4　The characteristics of test matrices

No.	Name	Explanation	Dimension	Non-zeros
1	crystk03	FEM crystal vibration	24696	1751178
2	venkat01	Unstructured 2D euler solver	62424	1717792
3	nasasrb	Shuttle rocket booster	54870	2677324
4	pwtk	Pressurized wind tunnel	217918	11634424
5	gearbox	Aircraft flap actuator	153746	9080404

ゴリズムを選択した場合の性能と，あらかじめ決めておいた固定アルゴリズムを用いた場合の性能とを比較した．

4.1　実験環境

本実験では，性能やアーキテクチャが異なる 4 機種の並列計算機上で実験を行った．これら計算機の特徴を表 3 に示す．

4.2　テスト行列

実験で使用した行列は，すべて Tim Davis's matrix collection[11] から集めたものである．これらの特徴を表 4 に示す．

4.3　固定アルゴリズム

本実験では最適なアルゴリズムの場合と，すべての環境と行列に対してデフォルトの固定アルゴリズムの場合の性能を比較する．比較に用いた 4 つの固定アルゴリズムを表 5 に示す．これらの固定アルゴリズムの組み合わせは多くの SpMxV ルーチンの実装でよく出てくるようなものにした．計算部分では 'U1'(no unrolling) または 'R2×2'．'U1' は高速化をしていないナイーブな実装であり，'R2×2' は多くの環境や行列に対してよい性能が期待されるものである．また，通信アルゴリズムは 'Allgather' または 'IrecvIsend-min' としている．'Allgather' は簡単だがとても遅い実装，'IrecvIsend-min' はほとんどの場合高速なことが期待されるアルゴリズムである．計算と通信がそれぞれ 2 通りずつあり，それらの組合せによって 4 通りの固定アルゴリズムとなっている．

4.4　実験結果

実験の結果を表 6 から表 9 に示す．この表の中で，'No.' は表 4 に示された行列番号を表す．'local' の列

表 5　固定のデフォルトアルゴリズム
Table 5　Default fixed algorithms

No.	local computation	communication
1	U1	Allgather
2	U1	IrecvIsend-min
3	R2×2	Allgatehr
4	R2×2	IrecvIsend-min

は計算のアルゴリズムに何が選択されたかを表す．並列数が 8 なのでこの列にはそれぞれ 8 つの要素が書かれている．列 'comm' は選択された通信アルゴリズムを表す．列 'time default' は固定アルゴリズムを使用した場合の時間を表す．単位はミリ秒である．括弧の中の 2 つの数字は，それぞれ計算時間と通信時間である．列 'time opt' は，最適なアルゴリズムを選択した場合の時間を表す．単位はミリ秒で，括弧の中は 'time default' と同じである．列 'speed-up' は，最適なアルゴリズムを使用した場合の速度が固定アルゴリズムの場合と比べてどれだけ高速になったかの比率を表す．

スピードアップの平均を表 10 にまとめた．この結果から，最適なアルゴリズムを選択した場合には，固定アルゴリズムを用いた場合に対して 1.29 倍から 4.56 倍高速になることがわかった．

これらの結果から，計算機のアーキテクチャや対象の行列によって最適なアルゴリズムは大きく異なることがわかる．Alpha マシンでは計算アルゴリズムに Unrolling がしばしば用いられている．一方，他の計算機では Register Blocking が多く選択されている．また計算アルゴリズムは行列の性質によっても最適なアルゴリズムが大きく異なることがわかる．最適な通信アルゴリズムに関しては，Allgather か範囲限定法か最小量通信か，といったアルゴリズムのタイプは行列に依存し，MPI の関数のタイプは計算機に依存していることがわかる．

表 10　スピードアップのまとめ
Table 10　Summary of speed-ups

Machine	1	2	3	4
PC-cluster	8.58	1.40	8.23	1.08
COMPAQ	3.66	1.54	3.12	1.48
SGI	3.63	1.59	3.16	1.40
HITACHI	2.39	1.71	1.86	1.20
average	4.56	1.56	4.09	1.29

5. おわりに

我々はローカルな計算部分と通信部分に複数の高速化アルゴリズムを持つ並列疎行列ベクトル積 (SpMxV) ルーチンを作成した．このルーチンでは，各アルゴリズムの性能を実行時にすべて計測することで行列や計

表 6 PC-cluster での結果
Table 6 The result of PC-cluster

No.	local	comm	time default	time opt	speed-up
			62 (12,52)		6.28
1	R2×4 R3×3 R3×3 R3×2	IsendIrecv-range	15 (12,3)	9	1.54
	D11 R1×3 R3×3 R3×3		58 (8,52)	(7,2)	5.87
			11 (8,3)		1.13
			184 (11,176)		10.7
2	R1×4 R1×4 R1×4 R1×4	IrecvIsend-min	21 (14,8)	17	1.22
	R1×4 R1×4 R1×4 R1×4		183 (10,175)	(9,9)	10.6
			20 (12,8)		1.17
			164 (16,150)		9.78
3	R2×3 R3×3 R3×4 R3×3	IrecvIsend-range	23 (20,3)	16	1.38
	U6 R2×2 R3×4 R2×3		158 (11,150)	(15,2)	9.44
			17 (14,3)		1.05
			589 (80,520)		9.23
4	R3×3 R3×3 R3×2 R3×3	SendRecv-min	98 (82,20)	63	1.55
	R3×2 R3×3 R3×3 R3×3		548 (48,509)	(48,16)	8.60
			68 (52,20)		1.08
			418 (66,363)		6.80
5	R2×3 R3×3 D15 R3×3	IsendIrecv-min	78 (67,12)	61	1.27
	R2×3 R3×3 R3×3 R3×3		401 (46,364)	(50,12)	6.53
			59 (48,12)		0.973

表 7 Compaq での結果
Table 7 The result of Compaq

No.	local	comm	time default	time opt	speed-up
			8 (3,5)		3.94
1	U1 U1 U1 U1	SendRecv-min	3 (2,1)	2	1.58
	U1 U1 U1 U1		7 (3,4)	(1,0)	3.45
			3 (2,1)		1.66
			20 (6,14)		6.44
2	U1 U1 U1 U1	SendRecv-min	6 (3,3)	3	1.94
	U1 U1 U1 U1		18 (5,13)	(2,1)	5.84
			6 (4,2)		2.04
			23 (12,11)		4.34
3	U1 U1 U1 U1	SendRecv-range	8 (6,2)	5	1.67
	U1 U1 U1 U1		16 (6,10)	(4,1)	3.08
			8 (6,2)		1.52
			150 (103,51)		1.59
4	D5 U3 U3 U1	SendRecv-min	104 (95,9)	94	1.10
	U3 U3 U3 U1		132 (85,50)	(89,5)	1.40
			86 (78,10)		0.917
			117 (82,38)		1.98
5	D5 R3x3 R3x3 R3x3	SendRecv-min	82 (76,7)	58	1.40
	R3x3 R3x3 R3x3 R3x3		107 (72,37)	(55,3)	1.82
			73 (66,7)		1.24

表 8 SGI での結果
Table 8 The result of SGI

No.	local	comm	time default	time opt	speed-up
			13 (4,8)		5.36
1	R3x3 R3x3 R3x3 R3x3	IrecvIsend-range	4 (3,1)	2	1.89
	R3x3 R3x3 R3x3 R3x3		10 (2,8)	(1,0)	4.44
			4 (2,1)		1.69
			27 (7,20)		4.48
2	R1x4 R1x4 R1x4 R1x4	SendRecv-min	7 (4,3)	6	1.30
	R1x4 R1x4 R1x4 R1x4		26 (6,20)	(3,2)	4.29
			7 (4,3)		1.30
			32 (14,18)		3.78
3	R2x2 R3x3 R2x2 R2x2	IsendIrecv-range	11 (9,2)	8	1.37
	R2x2 U4 R3x3 R4x3		25 (9,16)	(7,0)	3.02
			9 (7,2)		1.16
			204 (149,57)		2.42
4	R3x3 R3x3 R4x4 R2x3	SendRecv-min	160 (148,12)	84	1.90
	R3x3 R3x3 R3x3 R3x3		189 (133,57)	(79,6)	2.24
			138 (126,12)		1.64
			119 (78,43)		2.09
5	D5 R3x3 R3x3 R3x3	SendRecv-min	83 (76,7)	56	1.47
	R3x3 D7 D9 R3x3		102 (61,42)	(52,4)	1.80
			67 (60,6)		1.18

表 9　HITACHI HA8000 での結果
Table 9　The result of HITACHI HA8000

No.	local	comm	time default	time opt	speed-up
1	D13 R3x3 R3x3 R3x3 D5 R3x3 R3x3 R3x3	SendRecv-range	11 (6,4) 7 (5,1) 7 (3,4) 4 (2,1)	3 (2,1)	3.64 2.38 2.63 1.50
2	R2x4 R4x1 R4x4 R3x2 R1x4 R1x4 R3x2 R3x4	SendRecv-min	14 (7,7) 9 (6,3) 12 (5,7) 7 (4,3)	6 (4,2)	2.26 1.55 1.99 1.19
3	R3x2 R4x2 R3x3 D7 R3x3 R3x3 R3x3 R3x3	SendRecv-range	21 (14,7) 15 (13,2) 15 (8,7) 9 (7,2)	7 (7,1)	2.68 1.92 1.88 1.19
4	D5 D5 R4x4 R4x4 R4x4 D5 R4x4 R4x4	SendRecv-min	105 (76,31) 83 (75,8) 86 (58,31) 65 (57,9)	63 (61,4)	1.66 1.32 1.36 1.03
5	R3x3 R3x3 R3x3 R3x3 R3x3 R4x4 R3x4 R4x3	SendRecv-min	82 (62,21) 67 (61,6) 68 (48,23) 53 (47,6)	48 (44,4)	1.69 1.38 1.40 1.08

算機に対して最適なアルゴリズムを選択する．この最適なアルゴリズムを用いた場合の性能を，妥当な固定アルゴリズムを用いた場合と比較した．実験の結果，最適なアルゴリズムの選択により平均して 1.29 倍から 4.56 倍の速度向上が得られた．この結果から，行列や計算機によらず一様にアルゴリズムを固定して計算をするのは高性能を達成するのに十分ではなく，行列や計算機の性質に合わせてアルゴリズムを変えることが重要であるということが示された．したがって我々は，高速な並列 SpMxV のルーチンを作成するには，最適なアルゴリズムを動的に選択する機構を取り入れるべきである，と主張する．

今後の課題は，最適なアルゴリズムを軽いオーバーヘッドで見つけ出す手法を構築することである．この機能は高速な並列疎行列ベクトル積ルーチンを作成するために重要である．

参 考 文 献

1) Im, E.: *Optimizing the Performance of Sparse Matrix - Vector Multiplication*, PhD Thesis, University of California at Berkeley, May 2000 (2000).
2) Toledo, S.: Improving the memory-system performance of sparse-matrix vector multiplication, *IBM Journal of Research and Development*, Vol. 41, No. 6, pp. 711–725 (1997).
3) Agarwal, R. C., Gustavson, F. G. and Zubair, M.: A high performance algorithm using preprocessing for the sparse matrix-vector multiplication, *Proceedings of Supercomputing '92, IEEE Computer Society Press*, pp. 32–41 (1992).
4) Kotlyar, V., Pingali, K. and Stodghill, P. V.: Compiling Parallel Code for Sparse Matrix Applications, *Proceedings of ACM/IEEE SC '97 Conference* (1997).
5) Bik, A. and Wijshoff, H.: Automatic data structure selection and transformation for sparse matrix computations, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 2, pp. 109–126 (1996).
6) Geus, S. R. R.: Towards a fast parallel sparse matrix-vector multiplication, *Parallel Computing: Fundamentals & Applications, Proceedings of the International Conference ParCo'99, 17-20 August 1999, Delft, The Netherlands*, pp. 308–315 (2000).
7) Pinar, A. and Heath, M. T.: Improving Performance of Sparse Matrix-vector Multiplication, *Proc. Supercomputing 99* (1999).
8) Saad, Y. and Lo, G.: PSPARSLIB Users Manual: A Portable Library of parallel Sparse Iterative Solvers (1996).
9) Balay, S., Gropp, W. D., McInnes, L. C. and Smith, B. F.: PETSc 2.0 Users Manual, Technical Report ANL-95/11 - Revision 2.0.29, Argonne National Laboratory (2000).
10) Katagiri, T., Kuroda, H., Ohsawa, K., Kudoh, M. and Kanada, Y.: Impact of Auto-tuning Facilities for Parallel Numerical Library, *IPSJ Transactions on High Performance Computing Systems*, Vol. 42, No. SIG 12, pp. 60–76 (2001).
11) Davis, T.: University of Florida Sparse Matrix Collection (1997). T. Davis. University of Florida Sparse Matrix Collection. NA Digest, vol. 97., Jun 1997.
http://www.cise.ufl.edu/~davis/sparse/.