2008－HPC－118 (9)
2008／12／16

並列版PAGMEつきCG法の性能解析

馬場慎也[1] 南里豪志[2] 藤野清次[2] 染原一仁[3]
[1] 九州大学大学院システム情報科学府 [2] 九州大学情報基盤研究開発センター
[3] 日本電子株式会社

概要：実対称正定値行列の固有値に対する相加・相乗平均の関係に基づき，行列式の値を1に近づけて固有値を密集させるという前処理PAGMEが染原らによって提案された．この前処理では，逐次性の高い前進(後退)代入計算が並列化の容易な下(上)三角行列とベクトルの積の計算に置き換わるため，高い並列性能が得られると期待できる．そこで，本稿では，MPIによるプロセス並列化およびOpenMPによるスレッド並列化をPAGMEつきCG法に適用し，その並列性能の比較検証を行ったので結果を報告する．

Performance analysis of the CG method with parallelized PAGME (Preconditioning based on Arithmetic-Geometric Mean of Eigenvalues)

Shinya Baba[1] Takeshi Nanri[2] Seiji Fujino[2] Kazuhito Somehara[3]
[1] Graduate School of Information Science and Electrical Engineering, Kyushu University
[2] Reseach Institute for Information Technology, Kyushu University
[3] Japan Electron Optics Laboratory, Ltd.

Abstract: A preconditioning based on AGM (Arithmetic-Geometric Mean) of eigenvalues for symmetric positive definite matrix was newly proposed by Somehara *et al.* In this preconditioning, sequential forward (backward) substitution process is changed to mupltiplication of lower (upper) triangular matrix and vector due to preferable performance of parallelization. Therefore, we may expect more speedup on parallel computer with shared and distributed memory. In this article, we evaluate performance of the CG method with parallelized PAGME using MPI and OpenMP.

1 はじめに

大規模な実対称正定値行列 A を係数行列に持つ連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の解法として共役勾配(Conjugate Gradient, 以下CGと略す)法が一般に用いられる．CG法の収束性は係数行列の固有値の密集度に大きく左右されるため，前処理を併用して用いるのが一般的である．

最近，実対称正定値行列の固有値の相加・相乗平均の関係を利用し，行列式を1に近づけることにより固有値を密集させる前処理PAGME (Preconditioning based on Arithmetic-Geometric Mean of Eigenvaluesの略)が染原らにより提案された[4][5]．しかも，PAGMEは並列計算向きの設計思想を基に考案された．

一方，科学技術計算を高速で処理できる並列計算機が広く用いられるようになってきた．一般に，並列化手法にはMPI(Message Passing Interface)[1, 3]によるプロセス並列化とOpenMP[7]によるスレッド並列化，そしてそれらを組合わせたハイブリッド並列化がある．

本研究では，前処理PAGMEつきCG法のスレッド並列化，プロセス並列化を行い，数値実験を通してその性能評価を行う．本稿の構成は次の通りである．第2節で前処理PAGMEについて前処理行列，すなわち下(上)三角行列の作成方法について記述する．第3節で採用した並列化手法について説明する．第4節で数値実験結果と観察を示し，第5節でまとめと今後の展望について述べる．

2 前処理PAGMEについて

前処理PAGMEでは，並列化の容易な $V_L^T V_L \boldsymbol{r}_n$ の形の下(上)三角行列と残差ベクトルの積をどのように構成するかが重要な鍵になる．そこで，下三角行列の作り方について詳しく記述する．

2.1 下三角行列を用いた場合

対角スケーリングをした実対称正定値行列 $A = (a_{ij})$ に対する前処理を考える．ただし，行列 A では実対称正定値行列の性質から $1 > |a_{ij}|(i \neq j)$ が成立する．ここで，下三角行列 $V = (v_{ij})$ の各行の非対角項において非零とする要素を1つとし，その添字 (i, j) の集合を P とする．ここで下三角行列 V は，$(VAV^T) = 1$ を満たす．また，$i \leq j$ とする．$(VAV^T)_{ii}$ において

$$(VAV^T)_{ii} = v_{ij}^2 + v_{ii}^2 + 2a_{ij}v_{ij}v_{ii} = 1. \tag{1}$$

これを満たす v_{ii} は v_{ij} についての判別式より

$$\begin{aligned}
&(a_{ij}v_{ii})^2 - (v_{ii}^2 - 1) \geq 0 \\
&(v_{ii} - 1/\sqrt{1 - a_{ij}^2})(v_{ii} + 1/\sqrt{1 - a_{ij}^2}) \leq 0 \\
&-1/\sqrt{1 - a_{ij}^2} \leq v_{ij} \leq 1/\sqrt{1 - a_{ij}^2}.
\end{aligned} \tag{2}$$

固有値が密集するためには v_{ii} が最大でなければならない．よって，

$$v_{ij} = \begin{cases} 1/\sqrt{1 - a_{ij}^2}, & j = i \\ -a_{ij}/\sqrt{1 - a_{ij}^2}, & j \in P \\ 0, & j \neq i \wedge j \notin P \end{cases} \tag{3}$$

と導くことができる．ただし，$i = 1$ のときは，

$$v_{11} = 1 \tag{4}$$

とする．(i, j) の選択には次の二つの選択が考えられる．

- **選択 (a)**： $V_L(i-1)$: $j = i - 1$ とし，副対角項 a_{ii-1} を選択
- **選択 (b)**： $V_L(\max)$: 各行の非対角要素のうち，絶対値が最大 $\max_{1 \leq j \leq i-1} |a_{ij}|$ となるような a_{ij} を選択

選択 (a) をした場合は，前処理行列と残差ベクトルとの積 $\boldsymbol{y} = V_L\boldsymbol{r}$ と $V_L^T\boldsymbol{y}$ の計算中のループにおいてアンローリングが可能となるため，計算時間の増加を抑えられる．

選択 (b) をした場合は，上記の集合 P において $\prod_{k=1}^{N} v_{kk}$ の値が最大となるため，固有値が密集し，CG 法の収束性向上が期待できる．

2.2 上三角行列を用いた場合

前処理行列 V を上三角行列とした場合にも前節と同様に，$j \geq i$ として，

$$v_{ij} = \begin{cases} 1/\sqrt{1 - a_{ij}^2}, & j = i \\ -a_{ij}/\sqrt{1 - a_{ij}^2}, & j \in P \\ 0, & j \neq i \wedge j \notin P \end{cases} \tag{5}$$

が得られる．ただし，$i = N$ のときは，以下の式が得られる．

$$v_{NN} = 1 \tag{6}$$

また，行列 V を上三角行列とした場合にも前節と同様の集合 P の選択方法が考えられる．

- **選択 (c)**： $V_U(i+1)$: $j = i + 1$ とし，副対角項 a_{ii+1} を選択
- **選択 (d)**： $V_U(\max)$: 各行の非対角要素のうち，絶対値が最大 $\max_{i+1 \leq j \leq N} |a_{ij}|$ となるような a_{ij} を選択

2.3 前処理 PAGME 付き CG 法の算法

前処理行列に下三角行列 V_L を用いる場合について考える．前処理 PAGME 付き CG 法では，以下の方程式を CG 法で解くことになる．

$$(V_LAV_L^T)(V_L^{-T}\boldsymbol{x}) = V_L\boldsymbol{b}. \tag{7}$$

この式に対する CG 法の補助ベクトルを次式で定義する．

$$\tilde{\boldsymbol{r}}_n := V_L\boldsymbol{r}_n, \tilde{\boldsymbol{p}}_n := V_L^{-T}\boldsymbol{p}_n, \tilde{\boldsymbol{x}}_n := V_L^{-T}\boldsymbol{x}_n. \tag{8}$$

これらの式を使って，元の方程式 $A\boldsymbol{x} = \boldsymbol{b}$ に戻すと前処理 PAGME 付き CG 法の算法が得られる．以下に，前処理 PAGME 付き CG 法の算法を示す．前処理行列に上三角行列 V_U を用いる場合は，算法 1 中の V_L を V_U に置き換えればよい．

算法 1：前処理 PAGME 付き CG 法の算法

1. Let $\boldsymbol{x}_0$ be an arbitrary vector, and put $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0, \boldsymbol{p}_0 = V_L^TV_L\boldsymbol{r}_0$
2. For $n = 0, 1, \ldots,$ Do
3. $\alpha_n = \dfrac{(V_L^TV_L\boldsymbol{r}_n, \boldsymbol{r}_n)}{((V_L^TV_L\boldsymbol{r}_n, A\boldsymbol{p}_n)}$
4. $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha_n\boldsymbol{p}_n$
5. $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n - \alpha_nA\boldsymbol{p}_n$
6. If $\dfrac{||\boldsymbol{r}_{n+1}||_2}{||\boldsymbol{r}_0||_2} \leq \epsilon$ then stop
7. $\beta_n = \dfrac{(V_L^TV_L\boldsymbol{r}_{n+1}, \boldsymbol{r}_{n+1})}{(V_L^TV_L\boldsymbol{r}_n, \boldsymbol{r}_n)}$
8. $\boldsymbol{p}_{n+1} = V_L^TV_L\boldsymbol{r}_{n+1} + \beta_n\boldsymbol{p}_n$
9. End Do

3 並列化手法

並列化手法には共有メモリ型のスレッド並列化と分散メモリ型のプロセス並列化がある．本節では二つの並列化手法の違いについて説明し，本稿で用いた並列化手法を示す．

3.1 スレッド並列化とプロセス並列化

本研究では，並列化手法として，スレッド並列化とプロセス並列化を用いる．スレッド並列化ではOpenMPを，プロセス並列化ではMPIを使って並列化する．MPI版のPAGMEつきCG法の場合，反復内の各種計算でプロセス間通信を複数回必要とする．そのため，MPI版では並列台数の増加に従い，通信時間が増大し，OpenMP版の方が有効になる可能性が高い．

3.2 PAGMEつきCG法の並列化

PAGMEつきCG法に含まれる並列化にとって重要な計算を以下に示す．

1. ベクトルの内積
2. 疎行列・ベクトル積

各プロセス(またはスレッド)へのループの分割方法はブロック分割[2]を採用した．並列化の効果を高めるには，特に後者の疎行列・ベクトル積の計算が重要である．

3.2.1 ベクトルの内積の並列化

2つのベクトル $\boldsymbol{x} = (x_i)$, $\boldsymbol{y} = (y_i)$ の内積 $(\boldsymbol{x}, \boldsymbol{y})$ は

$$(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} x_i y_i \tag{9}$$

で定義され，内積 $(\boldsymbol{x}, \boldsymbol{y})$ は各プロセス(またはスレッド)の積和の小計を合計する必要がある．本研究では，プロセス並列化の場合はMPI_ALLREDUCE，スレッド並列化の場合はreductionを用いて内積を求めた．PAGME前処理つきCG法の反復中には，この内積計算が全部で3ヶ所所存在する．このため，MPI版では通信が3回必要となり，通信時間の増加を招くことが多い．

3.2.2 疎行列・ベクトル積の並列化

疎行列・ベクトル積 $\boldsymbol{y} = A\boldsymbol{p}$ を計算する場合，ベクトル $\boldsymbol{p}$ の値をすべて必要とする．スレッド並列化の場合はベクトル $\boldsymbol{p}$ の値を各スレッドで共有しているので問題無い．しかし，プロセス並列化の場合，各プロセスは $\boldsymbol{p}$ の値を部分的にしか持っていないため，通信用サブルーチンを用いてプロセス間で値を共有しなければならない．本研究では以下の通信方法を採用した．

1. MPI_ALLGATHER通通
2. MPI_ISEND, MPI_IRECV, MPI_WAITALL通信

ここで，行列の次元数を $ncol$，プロセス数を $nprocs$ とする．データの送信サイズは，次元数がプロセス数で割り切れる場合は $(ncol/nprocs)$，割り切れない場合は $(ncol + nprocs)/nprocs$ である．以下に，サブルーチンMPI_ISEND, MPI_IRECV, MPI_WAITALLを用いた通信の擬似アルゴリズムを示す．ただし，通信するベクトルを $\boldsymbol{p}$，送信サイズを $size$ とする．

擬似アルゴリズム：
MPI_ISEND, MPI_IRECV, MPI_WAITALLによる通信

```
count = 1
do i = 0, (プロセス数 − 1)
  if(i/ = プロセス番号) then
    call MPI_ISEND(p(担当計算範囲の先頭アドレス),
      size, ···, i, プロセス番号, ···, ireq(count), ···)
    count = count + 1
  end if
end do
do i = 0, (プロセス数 − 1)
  if(i/ = プロセス番号) then
    (ist : 受信データの先頭アドレス) = i ∗ size + 1
    if(i == (プロセス数 − 1)) then
      (iend : 受信データの後尾アドレス) = (行列の次元数)
    else
      (iend : 受信データの後尾アドレス) = ist + size − 1
    end if
    (recv_size : 受信サイズ) = iend − ist + 1
    call MPI_IRECV(p(ist), (recv_size), ···, i, i,
      ···, ireq(count), ···)
    count = count + 1
  end if
end do
call MPI_WAITALL(count − 1, ireq, ···)
```

4 数値実験

4.1 計算機環境と計算条件

数値実験は，九州大学情報基盤研究開発センターに設置された日立SR11000(以下，SRと略記する)で行った．OSはAIX 5L 5.3，CPUはIBM POWER5(クロック周波数: 1.9GHz)，ノード内コア数16，ネットワークは専用クロスバーネットワーク(ノードあたり4GB/s)である．プログラムはFortran90で実装した．計算はすべて倍精度浮動小数点演算で行った．コンパイラは日立最適化Fortran90を使用した．最適化オプションは，MPIでは-64 -Os -noparallel,

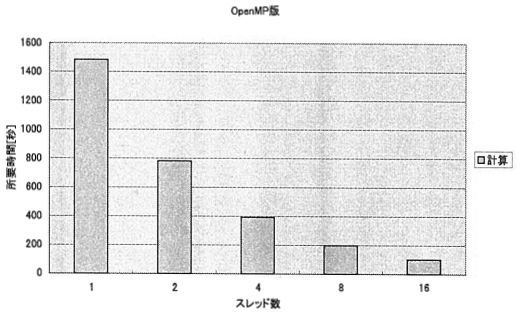
OpenMP では-64 -Os -omp を使用した．共通のオプションとして-noloopexpand -noprefetch を用いている．時間計測には，MPI では MPI_WTIME, OpenMP では system_clock を用いた．最大反復回数は行列の次元数と同じにした．収束判定値は，相対残差の 2 ノルム: $||\boldsymbol{r}_{n+1}||_2/||\boldsymbol{r}_0||_2 \leq 10^{-12}$ とした．プロセス数およびスレッド数は 1, 2, 4, 8, 16 の 5 通りとした．各実験は 3 回ずつ行いその平均値を表に示した．表 1 に，数値実験で用いた実対称行列 [6] の主な特徴を示す．ただし，表中の "$\max_{i \neq j} |a_{ij}|$" は対角スケーリングを適用した後の解くべき方程式の係数行列の要素の値である．

表 1: 実対称行列の主な特徴

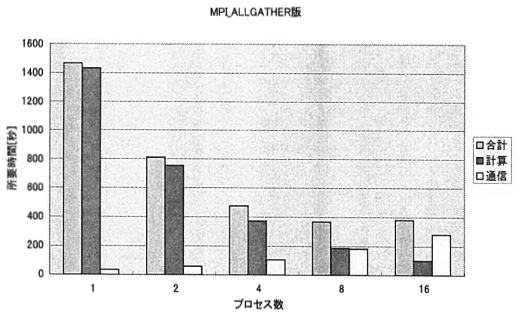
行列	次元数	非零要素数	解析	$\max_{i \neq j} \lvert a_{ij} \rvert$
nasasrb	54,870	2,677,324	構造 [6]	0.997
s3dkq4m2	90,449	4,427,725	同上	0.967
s3dkt3m2	90,449	3,686,223	同上	0.976
shipsec1	140,874	3,568,176	同上	0.990
ccs20	438,440	3,733,629	電磁界	0.993
ccs30	1,490,460	12,790,849	同上	0.993
ccs40	3,545,680	30,545,669	同上	0.993

4.2 実験結果

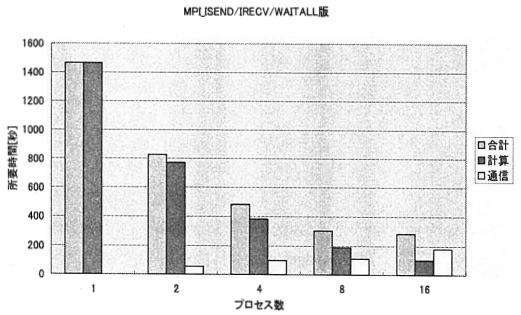
表 2-6 に並列化した PAGME つき CG 法 (V_L(max) を選択した) の各行列毎の反復回数と各種所要時間を示す．表 2 の"Th"はスレッド数を表す．"itr."は反復回数，"A$\boldsymbol{v}$"および"$V_L^T V_L \boldsymbol{r}_n$"は，行列・ベクトル積および下三角行列とベクトルの積の計算に要した各時間を表す．表 3-6 の"pro"はプロセス数，"計算時間"は計算部分に要した時間，"通信時間"は通信部分に要した時間，"合計時間"はそれらの合計時間である．また，表 4 の"計算"は計算部分，"通信"は通信部分に要した時間である．"他の計算"はベクトルの加減算，内積の計算に要した時間，"他の通信"は内積値の集計の通信時間である．また，図 1 に行列 ccs40 における (i)OpenMP 版，(ii)MPI_ALLGATHER 版および (iii)MPI_ISEND 版 (MPI_ISEND, MPI_IRECV, MPI_WAITALL を含む) の合計時間，計算時間，通信時間を示す．時間の単位はすべて秒とする．ただし，紙面の制約から，MPI_ISEND 版については，比較的小規模な構造解析の行列 (nasasrb, ⋯, shipsec1 まで) への適用結果は割愛した．



(a) OpenMP 版



(b) MPI_ALLGATHER 版



(c) MPI_ISEND 版

図 1: 行列 ccs40 に対する PAGME つき CG 法のスレッド (プロセス) 数ごとの合計時間および計算と通信時間の推移．

4.3 観察と考察

表 2-6, 図 1 に示した結果から次のことがわかる．

まず，OpenMP 版では，行列 ccs40 の場合，スレッド数の増加 (2Th, 4Th, 8Th, 16Th) に従い台数効果が 1.90 倍，

表 2: OpenMP 版 PAGME つき CG 法の反復回数と各種の計算に要した時間.

行列	Th	itr.	合計時間	Av	$V_L^T V_L r_n$	その他
nasasrb	1	7500	85.69	68.98	13.85	2.69
	2	7499	43.98	34.83	6.92	2.08
	4	7501	22.65	17.84	3.58	1.08
	8	7499	11.78	9.35	1.81	0.50
	16	7498	6.34	4.85	0.93	0.38
s3dkq4m2	1	9095	185.91	150.16	30.02	5.50
	2	9074	94.98	75.24	15.20	4.30
	4	9086	47.42	37.83	7.68	1.75
	8	9074	23.92	18.77	3.80	1.20
	16	9097	12.50	9.50	2.07	0.73
s3dkt3m2	1	9930	165.46	129.38	29.97	5.88
	2	9502	81.42	61.94	14.60	4.58
	4	9454	40.36	30.92	7.35	1.91
	8	9923	21.54	16.06	3.91	1.24
	16	9921	11.36	8.25	2.15	0.78
shipsec1	1	2479	80.95	66.64	11.85	2.31
	2	2477	42.09	33.94	6.07	1.94
	4	2477	21.13	17.20	2.94	0.84
	8	2479	10.70	9.00	1.17	0.41
	16	2476	5.54	4.41	0.75	0.28
ccs20	1	1976	94.32	59.61	28.60	5.93
	2	1976	49.68	30.09	14.49	4.94
	4	1975	25.10	15.23	7.26	2.46
	8	1975	12.49	7.43	3.62	1.31
	16	1975	6.18	3.65	1.89	0.50
ccs30	1	2874	474.33	300.81	142.51	30.45
	2	2874	248.93	152.02	71.94	24.47
	4	2874	124.64	76.29	35.91	11.99
	8	2873	62.96	38.29	17.97	6.30
	16	2873	32.51	19.71	9.13	3.24
ccs40	1	3759	1484.11	943.54	443.30	96.01
	2	3759	781.69	476.77	223.06	80.76
	4	3759	391.95	238.79	111.83	40.35
	8	3759	197.46	120.46	55.71	20.32
	16	3758	102.09	62.23	28.27	10.69

表 3: MPI_ALLGATHER 版 PAGME つき CG 法の反復回数と計算と通信に要した時間.

行列	pro	itr.	合計時間	計算時間	通信時間
nasasrb	1	7500	85.62	84.11	1.51
	2	7499	46.99	43.95	3.04
	4	7500	27.74	23.22	4.53
	8	7497	24.77	12.64	12.13
	16	7500	13.81	7.15	6.66
s3dkq4m2	1	9095	182.83	179.90	2.93
	2	9074	100.37	93.74	6.63
	4	8781	54.11	46.15	7.96
	8	9072	41.12	25.09	16.03
	16	9080	87.54	13.92	73.62
s3dkt3m2	1	9930	166.75	163.56	3.19
	2	9502	89.98	82.64	7.34
	4	9451	50.26	41.80	8.46
	8	9188	37.80	21.63	16.17
	16	9455	89.05	12.53	76.53
shipsec1	1	2479	81.41	80.06	1.36
	2	2477	45.51	41.58	3.92
	4	2477	25.13	21.07	4.05
	8	2479	16.36	11.09	5.27
	16	2478	26.60	6.14	20.46
ccs20	1	1976	93.67	91.16	2.51
	2	1976	51.23	47.81	3.43
	4	1975	30.38	23.67	6.70
	8	1975	22.32	11.83	10.50
	16	1975	31.79	5.82	25.97
ccs30	1	2874	468.54	457.02	11.53
	2	2874	255.37	239.05	16.32
	4	2873	148.25	118.87	29.39
	8	2873	112.27	59.75	52.52
	16	2873	126.34	31.88	94.46
ccs40	1	3759	1466.74	1432.55	34.19
	2	3759	810.38	752.16	58.21
	4	3759	476.31	372.39	103.92
	8	3758	368.15	186.38	181.77
	16	3758	380.99	99.27	281.72

3.79 倍, 7.52 倍, 14.54 倍と並列台数とほぼ同等の値になり, 並列化の効果がよく現れている.

次に, OpenMP 版と MPI_ISEND 版を比較する.

- 行列 ccs40 において, すべての並列台数で計算時間は MPI_ISEND 版の方が高速である.
- 疎行列・ベクトル積の計算時間は OpenMP 版の方が高速であるが, $V_L^T V_L r_n$ の計算や他の計算の計算時間は MPI_ISEND 版が高速である.
- しかし, 通信時間の増加のために合計時間に関しては OpenMP の方が有効である.

最後に, MPI_ALLGATHER 版と MPI_ISEND 版を比較する.

- 計算時間では, MPI_ALLGATHER 版の方が短い. しかし, 通信時間では, MPI_ISEND 版の方が短い.
- 特に, 16 並列時の通信時間は, MPI_ALLGATHER 版では 281.72 秒, MPI_ISEND 版では 180.77 秒と 100 秒近く MPI_ISEND 版の方が短い. そのため, 行列 ccs40 において, 2, 4 並列では MPI_ALLGATHER が有効であり, 8 並列以上では MPI_ISEND が有効である.
- プロセス数が多くなり, ノード内で実行する場合は MPI_ISEND の方が効果的である.
- 一方, MPI_ALLGATHER はプロセス数が小さい場合やノードを跨ぐ SMP クラスタでのハイブリッド並列化の場合に有効である.

以上の観察から, MPI による並列化では, より効率のよい通信方法を用いることにより, 通信時間を削減できる. しかし, SR 上での MPI 版と OpenMP 版の計算時間がほぼ同じであることから, SR 上のノード内での実行に関しては OpenMP の利用が効果的であることなどがわかった.

表 4: MPI_ALLGATHER 版 PAGME つき CG 法の反復回数と各種所要時間.

行列	pro	Av 合計	Av 計算	Av 通信	$V_L^T V_L \boldsymbol{r}_n$ 合計	$V_L^T V_L \boldsymbol{r}_n$ 計算	$V_L^T V_L \boldsymbol{r}_n$ 通信	他の計算	他の通信
	1	68.84	68.41	0.43	13.39	12.46	0.94	2.67	0.14
	2	35.77	35.02	0.75	8.29	6.28	2.01	1.63	0.29
nasasrb	4	19.53	18.26	1.27	6.20	3.21	2.99	0.69	0.27
	8	13.52	9.59	3.93	9.47	1.65	7.82	0.36	0.38
	16	7.03	5.05	1.98	4.94	0.83	4.11	0.20	0.58
	1	149.22	148.35	0.87	26.87	25.02	1.85	5.47	0.21
	2	77.02	74.96	2.06	16.97	12.72	4.24	4.10	0.33
s3dkq4m2	4	38.64	36.41	2.23	11.26	6.17	5.09	1.60	0.64
	8	24.06	19.09	4.97	13.29	3.22	10.07	0.77	0.98
	16	33.30	9.86	23.43	47.92	1.64	46.28	0.39	3.91
	1	129.84	128.88	0.95	29.59	27.57	2.02	5.96	0.22
	2	64.64	62.55	2.09	18.23	13.72	4.51	4.34	0.74
s3dkt3m2	4	33.70	31.34	2.36	12.09	6.71	5.38	1.65	0.73
	8	20.60	15.59	5.02	13.42	3.25	10.17	0.78	0.98
	16	32.67	8.39	24.29	50.21	1.67	48.55	0.40	3.69
	1	66.28	65.90	0.38	12.23	11.31	0.92	2.35	0.06
	2	34.39	32.99	1.40	7.69	5.78	1.91	1.94	0.62
shipsec1	4	17.48	16.48	0.99	5.05	2.91	2.14	0.80	0.92
	8	9.88	8.37	1.52	4.58	1.47	3.11	0.33	0.65
	16	10.73	4.32	6.40	13.80	0.74	13.06	0.16	0.99
	1	58.98	58.17	0.81	28.49	26.85	1.64	5.96	0.06
	2	30.54	29.44	1.10	15.65	13.42	2.23	4.78	0.10
ccs20	4	16.63	14.60	2.04	10.74	6.71	4.03	2.20	0.63
	8	9.84	7.19	2.64	8.94	3.36	5.58	1.10	2.27
	16	10.34	3.47	6.86	13.75	1.69	12.06	0.47	7.05
	1	297.65	293.93	3.72	139.65	131.96	7.68	30.60	0.12
	2	153.51	148.36	5.16	77.52	66.61	10.91	23.55	0.25
ccs30	4	83.31	74.43	8.88	51.65	33.08	18.57	10.83	1.94
	8	49.15	37.09	12.05	40.75	16.52	24.23	5.60	16.24
	16	39.16	19.92	19.24	43.93	8.38	35.55	3.03	39.67
	1	934.86	923.53	11.33	434.12	411.44	22.68	96.37	0.18
	2	484.56	465.87	18.69	245.27	206.04	39.23	79.03	0.30
ccs40	4	261.91	233.86	28.05	161.98	102.86	59.12	34.44	16.75
	8	158.33	116.50	41.83	131.12	51.50	79.63	17.13	60.31
	16	121.86	61.16	60.69	146.49	27.30	119.19	9.50	101.8

表 5: MPI_ISEND 版 PAGME つき CG 法の反復回数と所要時間.

行列	pro	itr.	合計時間	計算時間	通信時間
	1	1976	93.20	93.13	0.07
	2	1976	52.90	48.97	3.94
ccs20	4	1975	30.80	24.27	6.53
	8	1975	20.35	12.02	8.33
	16	1975	23.36	5.93	17.43
	1	2874	469.15	469.01	0.14
	2	2874	261.41	244.85	16.56
ccs30	4	2873	150.74	121.26	29.48
	8	2873	96.55	60.75	35.81
	16	2873	86.94	33.14	53.79
	1	3759	1466.19	1465.99	0.20
	2	3759	825.41	769.49	55.92
ccs40	4	3759	480.43	381.91	98.52
	8	3758	302.62	191.19	111.43
	16	3758	282.72	101.95	180.77

表 6: MPI_ISEND 版 PAGME つき CG 法の反復回数と各種所要時間.

行列	pro	Av 合計	Av 計算	Av 通信	$V_L^T V_L \boldsymbol{r}_n$ 合計	$V_L^T V_L \boldsymbol{r}_n$ 計算	$V_L^T V_L \boldsymbol{r}_n$ 通信	他の計算	他の通信
	1	60.47	60.47	0.00	26.60	26.59	0.01	5.91	0.06
	2	32.20	30.53	1.67	15.59	13.42	2.18	4.84	0.09
ccs20	4	17.16	15.12	2.05	10.40	6.69	3.70	2.29	0.78
	8	9.82	7.45	2.36	8.13	3.34	4.79	1.06	1.18
	16	8.66	3.61	5.05	11.99	1.68	10.31	0.46	2.06
	1	305.71	305.70	0.01	132.41	132.39	0.02	30.40	0.11
	2	159.53	154.46	5.07	77.90	66.61	11.29	23.24	0.20
ccs30	4	85.91	76.88	9.03	49.83	33.03	16.80	10.81	3.65
	8	49.36	38.35	11.01	37.02	16.46	20.55	5.38	4.24
	16	37.69	21.17	16.52	37.79	8.38	29.41	3.03	7.86
	1	958.71	958.69	0.01	410.47	410.44	0.03	95.66	0.16
	2	500.75	482.90	17.85	244.25	206.39	37.86	78.97	0.21
ccs40	4	270.29	243.15	27.14	162.50	102.74	59.76	34.80	11.62
	8	155.33	121.43	33.90	117.01	51.48	65.53	17.03	12.00
	16	114.98	64.20	50.78	132.16	26.98	105.18	9.46	24.81

5 まとめと今後の展望

本研究におけるこれまでの結果から, 計算機 SR 上での並列版 PAGME つき CG 法では, スレッド並列化の方が有効であった.

一方, 前処理 PAGME つき CG 法における MPI の効果的な利用法は, ノード内のみの実行ではなく, 複数ノードを使用した実行, あるいは OpenMP と組み合わせたハイブリッド並列化であると考えられる. そこで, 今後, より一層効率がよい通信方法の実装, および複数ノードを使用した実験を続行し, MPI 版とハイブリッド版の性能評価へと研究を繋げていきたい.

参考文献

[1] 青山幸也: 並列プログラミング入門 MPI 版, 理研スパコン・システム講習会配布テキスト, 2007.

[2] Lo, J.L., *et al.* : Tuning compiler optimizations for simultaneous multithreading, International Journal of Parallel Programming Vol.27, No.6, pp.477-503, 1999.

[3] Pacheco, P.: Parallel Programming with MPI, Morgan Kaufmann Publishers, 1997, 秋葉博訳: MPI 並列プログラミング, 培風館, 東京, 2001.

[4] 染原一仁, 藤野清次: 固有値の相加・相乗平均の関係を利用した前処理の提案. 日本応用数理学会 論文誌, Vol.18, No.4, 2008. (印刷中)

[5] Somehara, K., Fujino, S.: A Proposal of Preconditioning based on AGM (Arithmetic-Geometric Mean) of Eigenvalues and its Estimation, Abstract of PMAA'08, p.23, June, Neuchatel, Switzerland, 2008.

[6] University of Florida Sparse Matrix Collection: http://www.cise.ufl.edu/research/sparse/matrices/index.html

[7] 牛島 省: OpenMP による並列プログラミングと数値計算法, 丸善出版, 東京, 2006.