社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－177 (33)
2008－HPC－114 (33)
2008／3／6

行列のブロック・サイクリック分割に基づく IDR(s) 法の並列性能評価

藤野清次　草場健一郎†　尾上勇介†
(九州大学情報基盤研究開発センター，　九州大学大学院システム情報科学府†)

既報において，Sonneveld と van Gijzen により提案された IDR(s) 法では，右前処理が効率面から最適の選択で，他の BiCG 法系統の反復法に比べて，収束性が優れていることを明らかにした．本研究では，IDR(s) 法の並列性能について報告する．まず，IDR 定理と IDR(s) 法そして拡張 IDR 定理の概要を記述し，その算法を紹介する．次に，行列・ベクトルの積の計算に対する，行列のブロック分割と同ブロック・サイクリック分割について記述する．数値実験を通して，IDR(s) 法のブロック分割とブロック・サイクリック分割のときの IDR(s) 法の並列性能評価を行う．

Evaluation of parallel performance of IDR(s) method based on block-cyclic decomposition for matrix

Seiji FUJINO　Ken'ichiro KUSABA†　Yusuke ONOUE†
(Research Institute for Information Technology, Kyushu University
Graduate School of Information Science and Electrical Engineering, Kyushu University†)

Our previous paper made clear that the right preconditioning is the most suitable preconditioning for IDR(s) method, and IDR(s) method with the right preconditioning converges much faster than various BiCG-type of iterative methods for many realistic problems. In this article, effectiveness of parallel implementation of IDR(s) method by means of the block and block-cyclic decompositions for matrix-vector multiplication is examined. Through some numerical experiments, validity of IDR(s) method will be shown in view of parallelization.

1. はじめに

非対称行列を係数行列に持つ連立一次方程式に対して，非対称行列用の反復法 BiCG 法系統の反復法，GMRES 法系統の反復法そして GCR 法系統の反復法など様々な反復法が提案されきた．本稿では，Sonneveld と van Gijzen により提案された IDR(s) 法 [3] を OpenMP [1] [5] を用いて並列化を行ったので，(発表は逐次版の結果 [2] も合わせて) 結果を紹介する．IDR とは，Induced Dimension Reduction の略であり，数学的帰納法による次元縮小法と呼ばれる．並列化の手法は，反復法の中で，最も計算量が多い行列・ベクトルの積の計算に目標を絞り，行列に対して，2 つの分割法，すなわち，ブロック分割とブロック・サイクリック分割を行った．

本稿の構成は次のとおりである．第 2 節で，IDR 定理と IDR(s) 法そして拡張 IDR 定理の概要を記述し，その算法を紹介する．第 3 節で，ブロック分割について記述する．次の第 4 節で，ブロック・サイクリック分割について記述する．第 5 節で，実装について簡単に触れる．そして，第 6 節で，数値実験を通して，IDR(s) 法のブロック分割とブロック・サイクリック分割のときの並列性能評価を行う．第 7 節で，まとめと今後の課題を述べる．

2. IDR 定理と IDR(s) 法

解くべき線形方程式を

$$A\boldsymbol{x} = \boldsymbol{b}. \tag{1}$$

とする．$A = (a_{ij})$ は $N \times N$ の係数行列，$\boldsymbol{x} \in R^N$ は解ベクトル，$\boldsymbol{b} \in R^N$ は右辺ベクトルとする．非対称行列に対する反復法として，IDR(s) 法 [3] が提案された．IDR(s) 法の算法の元となる IDR 定理は以下の通りである．

定理 1： IDR 定理
係数行列 $A \in \mathrm{R}^{N\times N}$，任意のベクトル $\boldsymbol{v}_0 \in \mathrm{R}^N$，空間 $\mathcal{G}_j$ を完全 Krylov 部分空間 $K_N(A, \boldsymbol{v}_0)$ とする．空間 S を $S \in \mathrm{R}^N$ とし，$S \cap \mathcal{G}_0 \notin A$ を満たすとする．さらに，一連の空間 $\mathcal{G}_j (j = 1, 2, \cdots)$ を以下のように定義する．

$$\mathcal{G} := (I - \omega_j A)(\mathcal{G}_{j-1} \cap S). \tag{2}$$

ω_j は非零のスカラー値である．このとき，次の関係が成り立つ．

(i) $\mathcal{G}_j \subseteq \mathcal{G}_{j-1}$ for all $j > 0$
(ii) $\mathcal{G}_j = \mathbf{0}$ for some $j \leq N$

IDR(s) 法の算法を以下に示す．ϵ は収束判定用の微小な値である．

1. Let $\boldsymbol{x}_0$ be an initial guess; and put $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
2. For $n = 0, \cdots, s-1$ Do
3. $\boldsymbol{v}_n = A\boldsymbol{r}_n,\ \omega_n = \frac{(\boldsymbol{v}_n, \boldsymbol{r}_n)}{(\boldsymbol{v}_n, \boldsymbol{v}_n)}$
4. $\boldsymbol{q}_n = \omega_n \boldsymbol{r}_n,\ \boldsymbol{e}_n = -\omega_n * \boldsymbol{v}_n$
5. $\boldsymbol{r}_n = \boldsymbol{r}_n + \boldsymbol{e}_n,\ \boldsymbol{x}_n = \boldsymbol{x}_n + \boldsymbol{q}_n$
6. End Do
7. $E_s = (\boldsymbol{e}_{s-1} \cdots \boldsymbol{e}_0)$
8. $Q_s = (\boldsymbol{q}_{s-1} \cdots \boldsymbol{q}_0)$
9. Do $n = s, s+1, \cdots$
10. Solve $\boldsymbol{c}_n$ from $P^T E_n \boldsymbol{c}_n = P^T \boldsymbol{r}_n$
11. $\boldsymbol{v}_n = \boldsymbol{r}_n - E_n \boldsymbol{c}_n$
12. If mod$(n, s+1) = s$ then
13. $\boldsymbol{t}_n = A\boldsymbol{v}_n,\ \omega = \frac{(\boldsymbol{t}_n, \boldsymbol{v}_n)}{(\boldsymbol{t}_n, \boldsymbol{t}_n)}$
14. $\boldsymbol{e}_n = -E_n \boldsymbol{c}_n - \omega_n \boldsymbol{t}_n,$
15. $\boldsymbol{q}_n = -Q_n \boldsymbol{c}_n + \omega_n \boldsymbol{v}_n$
16. Else
17. $\boldsymbol{q}_n = -Q_n \boldsymbol{c}_n + \omega_n \boldsymbol{v}_n,\ \boldsymbol{e}_n = -A\boldsymbol{q}_n$
18. End If
19. $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \boldsymbol{e}_n,\ \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{q}_n$
20. if $||\boldsymbol{r}_{n+1}||_2 / ||\boldsymbol{r}_0||_2 \leq \epsilon$ then stop
21. $E_n = (\boldsymbol{e}_{n-1} \cdots \boldsymbol{e}_{n-s})$
22. $Q_n = (\boldsymbol{q}_{n-1} \cdots \boldsymbol{q}_{n-s})$
23. End Do.

さらに，次に示す拡張 IDR 定理が成り立つ．

定理 2:　拡張 IDR 定理

$$0 \leq d_j - d_{j+1} \leq d_{j-1} - d_j \leq s.$$

次に，残差ベクトル $\boldsymbol{r}_{m+1}$ の構成法とその線形結合について記す．

- 残差ベクトル $\boldsymbol{r}_{m+1}$ の構成法

  IDR(s) 法の一連の空間 $\mathcal{G}_j$ に $s+1$ 個の残差ベクトル $\boldsymbol{r}_{m+1}$ が属するように構成する．また，$\boldsymbol{r}_{m+1} \in \mathcal{G}_j$ であるとき，空間 $\mathcal{G}_j$ において IDR(s) 定理が成り立つように，残差ベクトル $\boldsymbol{r}_{m+1}$ は次式で計算される．

$$\boldsymbol{r}_{m+1} \quad = \quad (I - \omega_j A)\boldsymbol{v},\ \boldsymbol{v} \in \mathcal{G}_{j-1} \cap \mathrm{Null}(P^T). \tag{3}$$

- 残差ベクトル $\boldsymbol{r}_{m+1}$ の線形結合

  $\boldsymbol{v} \in \mathcal{G}_{j-1}$ より，ベクトル$\boldsymbol{v}$は空間 $\mathcal{G}_{j-1}$ に属する残差ベクトルの線形結合で表される．$\boldsymbol{r}_{m-i} \in \mathcal{G}_{j-1} (i = 0, 1, \ldots, s)$ とし，残差ベクトルの差分を $\boldsymbol{e}_\ell := \boldsymbol{r}_\ell - \boldsymbol{r}_{\ell-1}$ と定義すると，$\boldsymbol{v}$ は，

$$\boldsymbol{v} = \boldsymbol{r}_m - \sum_{\ell=1}^{s} c_{m-\ell} \boldsymbol{e}_{m-\ell} \tag{4}$$

  と表せる．$\boldsymbol{v} \in \mathrm{Null}(P^T)$ より $P^T\boldsymbol{v} = 0$ が成り立つため，係数群 $c_{m-\ell} (\ell = 1, \ldots, s)$ を未知数とする $s \times s$ の連立一次方程式の求解から，(4) 式中の係数群 $c_{m-\ell}$ が定まり，ベクトル $\boldsymbol{v}$ が求まる．

IDR(s) 法の算法では，反復 1 回毎に行列 A とベクトル $\boldsymbol{v}$ の積 $A\boldsymbol{v} = \boldsymbol{y}$ を計算する．A の非零要素を CRS (Compressed Row Storage) 方式で保持するとき，行列・ベクトルの積の計算を Fortran90 で表すと，以下の実装のように書ける．ここで，“ncol” は次元数，“rowptr”，“colind”，“val” は各々 A の各行の先頭ポインタ用配列，各要素の列インデックス配列，非零要素の値とし，“y(i)” はベクトル $\boldsymbol{y}$ の i 番目の要素，“v(i)” はベクトル $\boldsymbol{v}$ の i 番目の要素を表す．

実装

```
1  do i = 1, ncol
2     y(i)=0.0d0
3     do j = rowptr(i),rowptr(i+1)-1
4           y(i) = y(i) + val(j) * v(colind(j))
5     end do
6  end do
```

行列・ベクトルの積の計算の並列化のとき，行列の各スレッドへの分割法には以下の方法がある．

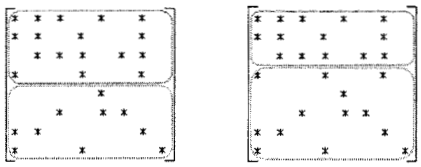
1. ブロック分割
2. ブロック・サイクリック分割
3. ハイブリッド分割: 上記の第 1 項と第 2 項を組み合わせたハイブリッド (今回は割愛)

3. 行列のブロック分割

ブロック分割とは，各スレッドに対してある部分領域を担当させる方式で，次の 2 通りがある．

- **分割 (a-1)** ··· 各ブロックの行数が均等になるようなブロック分割．
- **分割 (a-2)** ··· 各ブロックの非零要素数が均等になるようなブロック分割．

2 つのブロックに分割し，各ブロックをスレッド Th0，Th1 に割り当てるとすると，分割 (a-1)，(a-2) は下図のようになる．



分割 (a-1) のとき　　分割 (a-2) のとき

図 1：ブロック分割の一例

上の分割 (a-1) では，各ブロックの非零要素数，行列・ベクトル積の計算で，各スレッドで 4 行目の計算の実行回数は

- 上ブロック ····· 16 個，Th0 ····· 16 回
- 下ブロック ····· 10 個，Th1 ····· 10 回

となり，各スレッドに対する負荷分散は均等でない．したがって，非零分布が均等でない行列には，分割 (a-1) は不適である．

分割 (a-2) では，各ブロックの行数，行列・ベクトル積の計算の中で，各スレッドが 3 行目の rowptr 配列にアクセスする回数，外側ループ回数は，

- 上ブロック ·· 3 行，Th0 ·· 3 回，Th0 ·· 3 回
- 下ブロック ·· 5 行，Th1 ·· 5 回，Th1 ·· 5 回

となるため，スレッドに対する負荷分散が均等でない．非零分布が均等でない行列では，分割 (a-2) のように各ブロックの行数が不均等になる．

4. 行列のブロック・サイクリック分割

この手法は，何行かまとめて 1 つのブロックとし，ブロック単位でサイクリック分割する手法である．具体的には，スレッド数を nTh とするとき，

1. k ブロックに分割．
2. i 番目のスレッドに，nTh*(j-1) + i (i=1, ···, nTh，j=1, ···, $\frac{k}{nTh}$) 番目のブロックを割り当てる．

一般に，k が大きくなるほど負荷の均等性は向上するが，分割の手間が増える．例えば，次の行列では，nTh=2，k=2 のときのブロック・サイクリック分割は次のように表せる．

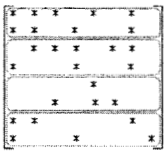


図 2：ブロック・サイクリック分割の一例

このとき，各ブロックの非零要素数，行列・ベクトル積の計算で，各スレッドで 4 行目の計算回数は，

- 赤色のブロック ···· 12 個，Th0 ···· 12 回
- 青色のブロック ···· 14 個，Th1 ···· 14 回

となり，ブロック分割に比べて負荷は均等になる．

図 3 にブロック分割とブロック・サイクリック分割を用いたときの各スレッドにおける経過時間の概念図を示す．ブロック分割ではスレッド Th1 が経過時間 t4 で終了し Th0 が経過時間 t1 で終了する．この経過時間 t4 から経過時間 t1 までの間，スレッド Th1 は待ち状態となる．一方，ブロック・サイクリック分割では，負荷の均等性が向上しており，スレッド Th0 での経過時間が t2 となり，Th1 では経過時間 t3 となる．これにより，ブロック分割と比較すると，ブロック・サイクリック分割の方が，全体の経過時間が短縮される．

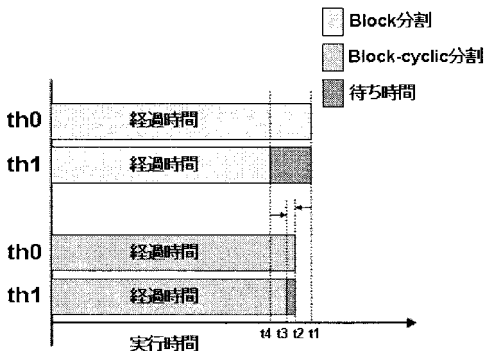
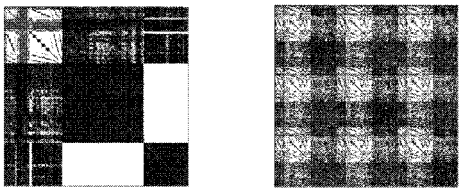


図 3：各スレッドにおける経過時間の概念図

実際に Florida 大学疎行列データベース [4] の行列 Poisson3db (次元数=85623，非零要素数: nnz=2374949)，Sme3da (次元数=12504，非零要素数: nnz=874887) において，ブロック・サイクリック分割を行ったとき (nTh=2) の各スレッドの非零要素数を示す．本研究では，スレッド数に 1，2，4，8，16 としたので，分割数は 2 のべき乗とした．図 4 は，行列 Poisson3db，Sme3da の非零要素の分布を表す．図中の赤色は正の要素，青色は負の要素を表す．



(a)Poisson3db　　(b)Sme3da

図 4：行列 Poisson3db，Sme3da の非零要素分布

表 1：行列 Poisson3db のときの，スレッド毎の非零要素数とその比

分割数 k	Th0 の nnz	Th1 の nnz	Th0/Th1
2	1,397,841	977,108	1.4306
8	1,409,776	965,173	1.4606
32	1,197,199	1,177,750	1.0165
128	1,193,120	1,181,829	1.0096
256	1,191,977	1,182,972	1.0076
512	1,191,919	1,183,030	1.0075
1024	1,187,942	1,187,007	1.0008

表 2：行列 Sme3da のときの，スレッド毎の非零要素数とその比

分割数 k	Th0 の NNZ	Th1 の NNZ	Th0/Th1
2	446,213	428,674	1.0409
8	443,288	431,599	1.0271
32	440,161	434,726	1.0125
128	440,397	434,490	1.0136
256	437,984	436,903	1.0025
512	437,514	437,373	1.0003
1024	436,680	438,207	1.0035

表 1-2 から，分割数 k が多くなる程ほど，負荷の均等性が向上 (行列 Poisson3db：分割数 k=2 → 8 では悪化し，行列 Sme3da：分割数 k=32 → 128 で悪化) していることが分かる．ここで，分割数 k=2 のときがブロック分割と同じなので，負荷の均等性は，ブロック・サイクリック分割の方がよい．

5. 実装について

ブロック・サイクリック分割では与えられたパラメータでブロック数が決定される．ブロック分割のときと同様に，各ブロックの上端の行番号を配列 start(i) に格納する．1 行目で，パラメータ k の値からブロック数を計算する．各ブロックの上端の行番号を格納する配列 start(i) の大きさは i=1，...，nTh*k となる．

```
 1  numb1 = k      ⇐ ブロック数
 2  start(1) = 1
 3  tmpi1 = ncol/(numb1)
 4  tmpi2 = mod(ncol,(numb1))
 5  do i=1,tmpi2
 6    start(i+1) = start(i)+tmpi1+1
 7  end do
 8  do i=tmpi2+1,numb1
 9    start(i+1) = start(i)+tmpi1
10  end do
```

図 5：ブロック・サイクリック分割の実装

```
1 !$omp parallel do private(i,l,tmp,temp)
2 do i=1,nthread
3   do l=1,k
4     tmp = (l-1)*nthread+i  ⇐ スレッドのブロック番号
5     do k=start(tmp),start(tmp+1)-1
6       temp=0.0d0
7       do j = rowptr(k),rowptr(k+1)-1
8         temp = temp + val(j) * p(colind(j))
9       end do
10      y(k) = temp
11    end do
12  end do
13 end do
```

図 6：行列・ベクトル積の計算の実装

6. 数値実験

計算機環境と計算条件は以下の通りである。計算機は，Hitachi SR11000 モデル J1 (CPU: POWER5，クロック周波数: 1.9GHz，メモリ: 128Gbytes，OS:AIX5.3) を使用した．コンパイラは最適化 Fortran90 を用い，逐次における最適化オプションは-64 -Oss -noparallel，並列化における最適化オプションは-64 -Oss -omp を使用した．計算はすべて倍精度浮動小数点演算で行った．時間の計測は，逐次計算，並列計算ともに計測関数 xclock を用いた．収束判定は残差 2 ノルムの比：$||\boldsymbol{r}_{k+1}||_2/||\boldsymbol{r}_0||_2 \leq 10^{-12}$ とした．初期近似解 $\boldsymbol{x}_0$ はすべて 0.0 とした．最大反復回数は 10000 回とした．対角項はすべて 1 に正規化して IDR(s) 法を適用した．IDR(s) 法のパラメータ s の値は 1 から 10 まで 10 通り調べた．スレッド数 Th は 1，2，4，8，16 と変えた．テスト行列の特徴を表 3 に示す．

表 3：テスト行列の特徴

行列	次元数	非零要素数	平均非零要素数
Sme3da	12,504	874,887	69.97
Sme3db	29,067	2,081,063	71.6
Sme3dc	42,930	3,148,656	73.34
Poisson3da	13,514	352,762	26.1
Poisson3db	85,623	2,374,949	27.74
Memplus	17,758	126,150	7.1
Wasewaka	19,060	24,377,548	1279.0
Language	399,130	1,216,334	3.05

表 4-表 11 に 8 個の行列に対する性能評価を示す．各表において，IDR(s) 法の結果は最適なパラメータ s の場合である．表中で，“分割”は，ブロック (block) 分割とブロック・サイクリック (block-cyclic) 分割である．時間の単位は秒である．“TRR”とは，近似解 $\boldsymbol{x}_n$ に対する真の相対残差 (True Relative Residual) の $||\boldsymbol{b}-A\boldsymbol{x}_n||_2/||\boldsymbol{r}_0||_2$ の常用対数: $\log_{10}$ の値を意味する．複数スレッドの場合，ブロック・サイクリック分割のとき，分割数は 512 に今回は固定して実験を行った．“台数効果”は，スレッド数が 1 のときの経過時間を 1 としたときの比である．スレッド 16 の場合，ブロック分割のときの，台数効果の最小 3.48 倍 (行列 Memplus)，最大 10.71 倍 (行列 Sme3da) であった．一方，ブロック・サイクリック分割のときの，台数効果の最小 4.58 倍，最大 13.49 倍 (行列 Sme3da) であった．真の相対残差 TRR については，行列 Language のときが異常に低かった．

表 4：行列 Sme3da に対する IDR(s=9) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	2173	8.54	1.00	−11.36
	2	2	2465	5.02	1.70	−10.61
	4	4	2113	2.44	3.50	−10.33
	8	8	2467	1.63	5.24	−11.16
	16	16	2325	1.10	7.76	−10.42
block-cyclic	1	1	2173	8.54	1.00	−11.36
	2	512	2190	4.76	1.79	−10.46
	4	512	2254	2.61	3.27	−10.67
	8	512	2200	1.38	6.19	−10.86
	16	512	2361	0.95	8.99	−10.12

表 5：行列 Sme3db に対する IDR(s=9) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	2876	29.26	1.00	−10.46
	2	2	3614	19.11	1.53	−10.54
	4	4	3219	9.54	3.07	−10.14
	8	8	3647	6.00	4.88	−9.75
	16	16	2709	3.14	9.32	−10.16
block-cyclic	1	1	2876	29.26	1.00	−10.46
	2	512	2804	15.16	1.93	−10.26
	4	512	2959	8.38	3.49	−9.92
	8	512	3020	4.56	6.42	−9.80
	16	512	2876	2.74	10.68	−10.85

表 6：行列 Sme3dc に対する IDR(s=9) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	4007	83.25	1.00	−9.49
	2	2	4410	36.01	2.31	−9.56
	4	4	4461	20.40	4.08	−10.53
	8	8	4616	11.82	7.04	−9.89
	16	16	4326	7.77	10.71	−10.78
block-cyclic	1	1	4007	83.25	1.00	−9.49
	2	512	4410	37.14	2.24	−9.56
	4	512	4461	19.21	4.33	−10.53
	8	512	4103	9.31	8.94	−10.25
	16	512	4326	6.17	13.49	−10.78

表 7：行列 Poisson3da に対する IDR(s=6) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	234	0.51	1.00	−11.93
	2	2	231	0.29	1.76	−12.13
	4	4	238	0.18	2.83	−12.37
	8	8	234	0.13	3.92	−12.29
	16	16	232	0.08	6.38	−12.01
block-cyclic	1	1	234	0.51	1.00	−11.93
	2	512	237	0.30	1.70	−12.56
	4	512	238	0.16	3.19	−12.37
	8	512	230	0.10	5.10	−12.14
	16	512	238	0.07	7.29	−12.29

表 8：行列 Poisson3db に対する IDR(s=4) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	534	8.48	1.00	−12.15
	2	2	539	4.53	1.87	−12.09
	4	4	543	2.85	2.98	−12.03
	8	8	545	2.11	4.02	−12.24
	16	16	540	1.31	6.47	−12.07
block-cyclic	1	1	534	8.48	1.00	−12.15
	2	512	539	4.15	2.04	−12.09
	4	512	543	2.26	3.75	−12.03
	8	512	545	1.30	6.52	−12.24
	16	512	540	0.85	9.98	−12.07

表 9：行列 Memplus に対する IDR(s=2) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	867	0.87	1.00	−12.08
	2	2	885	0.59	1.47	−12.19
	4	4	999	0.47	1.85	−12.13
	8	8	909	0.30	2.90	−12.12
	16	16	924	0.25	3.48	−12.03
block-cyclic	1	1	867	0.87	1.00	−12.08
	2	512	885	0.56	1.55	−12.19
	4	512	999	0.40	2.18	−12.13
	8	512	957	0.25	3.48	−12.25
	16	512	924	0.19	4.58	−12.03

表 10：Wasewaka に対する IDR(s=10) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	1949	90.21	1.00	−9.79
	2	2	2391	107.57	0.84	−10.08
	4	4	2058	89.32	1.01	−8.55
	8	8	2115	56.11	1.61	−10.20
	16	16	2177	28.88	3.12	−11.61
block-cyclic	1	1	1949	90.21	1.00	−9.79
	2	512	2391	56.26	1.60	−10.08
	4	512	2101	25.42	3.55	−9.58
	8	512	2332	14.05	6.42	−11.83
	16	512	2301	8.42	10.71	−10.07

表 11：Language に対する IDR(s=3) 法の性能

分割	Th	分割数	反復回数	経過時間	台数効果	TRR
block	1	1	46	1.17	1.00	−9.68
	2	2	46	0.66	1.77	−9.79
	4	4	48	0.45	2.60	−11.50
	8	8	44	0.28	4.18	−7.82
	16	16	47	0.21	5.57	−12.61
block-cyclic	1	1	46	1.17	1.00	−9.68
	2	512	44	0.61	1.92	−9.80
	4	512	45	0.36	3.25	−9.58
	8	512	44	0.25	4.68	−7.82
	16	512	44	0.18	6.50	−9.43

7. まとめと今後の課題

IDR(s) 法の並列評価を行った．ブロック・サイクリック分割は，行列の非零要素の分布が不均等の場合，ブロック分割より効率的な分割法である．今後は，ブロック分割とブロック・サイクリック分割のハイブリッド型分割のときの性能を調べていく予定である．

表 12 に，ブロック分割とブロック・サイクリック分割，両者のハイブリッド型分割のときの，予想される主な特徴を示す．

表 12：3 つの分割の予想される主な特徴

分割	分割の手間	負荷の均等性
block	少ない	悪い
block-cyclic	多い k →大きい程増大	良い k →大きい程向上
hybrid	増大	向上

表 12 から，ブロック分割は非零分布が均等な行列に向く．また，ブロック・サイクリック分割は，非零分布が不均等な行列に向く．両者のハイブリッド型分割は，適当な指標の設定により，分割の手間を抑えて，負荷の均等性を実現できることができると思われる．

参考文献

[1] 南里豪志，渡部善隆: OpenMP 入門，九州大学情報基盤センター広報誌別刷，Vol.2，No.1，2002.

[2] 尾上勇介，藤野清次：前処理つき IDR(s) 法における前進後退代入計算の可変的演算量，応用数学合同研究集会報告集，龍谷大学瀬田キャンパス，12 月，pp.254-259，2007.

[3] P. Sonneveld and M. B. van Gijzen: "IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems", TR07-07, Depart. of Math. Ana., Delft, The Netherlands, 2007.

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

[5] 牛島省: OpenMP による並列プログラミングと数値計算法，丸善，2006.