計算機アーキテクチャ 142−14
ハイパフォーマンス
コンピューティング 85−14
（2001. 3. 8）

メモリ使用量の少ない一般化共役残差法の提案

工藤 誠[†1] 黒田 久泰[†2]
片桐 孝洋[†3,†4] 金田 康正[†2]

大規模疎行列を係数行列に持つ連立一次方程式の解法には，Krylov 部分空間法が広く用いられている．その中でも，係数行列が非対称の場合の解法として，Arnoldi 原理に基づいた GCR(Generalized Conjugate Residual) 法が知られている．この GCR 法の実装に関してリスタート版の GCR や，eGCR(efficient GCR) などが提案されているが，これらの方法を計算量，メモリ使用量の両方の観点から考察し，これら既存の方法と比べて計算精度と計算量は同程度でメモリ使用量が少ない方法 (meGCR) と，計算精度は悪くなるが計算量とメモリ使用量が少ない方法 (uGCR) の 2 つの方法を提案する．さらに，これらの方法を逐次環境と並列環境で実装して収束性や実行時間を調べた．この結果，meGCR 法は収束性に問題がなく十分実用的であることがわかった．しかし，uGCR 法は精度の問題から収束性のよい問題にしか適用できないことがわかった．

A Proposal for GCR Methods with Less Memory Requirement

Makoto Kudoh,[†1] Hisayasu KURODA,[†2]
Takahiro KATAGIRI[†3,†4] and Yasumasa KANADA[†2]

For the solution of large sparse linear systems, Krylov subspace methods are widely used. Particularly for nonsymmetric linear systems, GCR(Generalized Conjugate Residual) method based on Arnoldi principle is well-known. Some implementations of GCR method such as restarted GCR, eGCR(efficient GCR) have been developed. Their computational complexity and required memory size are investigated, and two new methods, named meGCR and uGCR are proposed in this thesis. Compared with conventional methods, the meGCR method has the same computational error and computational complexity and small required memory size, and the uGCR method has the nature of large computational error, small computational complexity and required memory size. These conventional and proposed methods are implemented under the sequential and parallel environments. Their convergence behavior and execution time are investigated. It is shown that the meGCR method does not have computational error problem and it is practical enough. Moreover, it is proved that the uGCR method has serious computational error problem and it can be applied to limited problems with good convergence nature.

1. はじめに

係数行列が大規模疎行列の連立一次方程式 $Ax = b$ ($A \in \Re^{N \times N}$ $x, b \in \Re^N$) の解法には，Krylov 部分空間法が広く用いられている．GCR 法 (Generalized conjugate residual method) はその Krylov 部分空間法の一つであり，非対称の問題に適用可能で比較的広範囲な問題が解ける方法として知られている．また，近年効率的な解法として注目されている GMRESR 法[1] の一部としてもこの GCR 法が用いられており，GCR 法の重要性は高い．

さて，GCR 法は反復回数が増えるに従って計算量，メモリ使用量が線形的に増加するので，実用的にはリスタート版の GCR 法が多く用いられている．しかし，GCR 法には計算量，メモリ使用量が大きいという問題点があり，リスタート周期を大きくできない，大きな問題が解けない，などの不都合が生じる場合がある．

本報告では，memory efficient GCR(meGCR) 法と unrolled GCR(uGCR) 法という 2 つの GCR 法のアルゴリズムを提案する．meGCR 法は既存のアルゴリズムと比べて，計算量が同じで，メモリ使用量が約

†1 東京大学理学部情報科学科
Department of Information Science, the Faculty of Science, the University of Tokyo
†2 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
†3 東京大学大学院理学系研究科情報科学専攻
Department of Information Science, Graduate School of Science, the University of Tokyo
†4 日本学術振興会特別研究員
Research Fellow of the Japan Society for the Promotion of Science

半分になるという特徴を持つ．一方でuGCR法は，既存のアルゴリズムと比べて計算量，メモリ使用量とも少なくなる．

また，これらの提案するアルゴリズムと既存のアルゴリズムの逐次環境，並列環境での性能を比較する．

2. GCR法のアルゴリズム

GCR法はEisenstatら[2]によって提案された，Arnoldi原理に基づくKrylov部分空間法である．GCR法では，各反復においてArnoldi原理によりA^TA正規直交ベクトルを生成する．また，各反復で計算される残差は，同じくArnoldi原理に基づくKrylov部分空間法の一つであるGMRES法[3]の残差と理論的に同じである，という特徴を持つ．

2.1 リスタート版GCR法

GCR法では，各反復で正規直交ベクトルを計算するのに過去に計算したベクトルを使って計算しなければならず，反復回数の増加とともに計算量，メモリ使用量が増加する．このため，リスタート版のGCR法が実用的には広く使われている．[2][4] このアルゴリズムを図1に示す．以下で，このアルゴリズムをGCR(k)法と呼ぶ．

Select x_0, tol
$r_0 = b - Ax_0$
$p_0 = K^{-1}r_0$
for $n = 0, 1, \cdots k-1$ **until** $\|r_n\| \leq tol$ **do** :
begin
$\alpha_n = \frac{(Ap_n, r_n)}{(Ap_n, Ap_n)}$
$x_{n+1} = x_n + \alpha_n p_n$
$r_{n+1} = r_n - \alpha_n Ap_n$
$\beta_{n,i} = \frac{(Ap_i, AK^{-1}r_{n+1})}{(Ap_i, Ap_i)}, \quad i \leq n$
$p_{n+1} = K^{-1}r_{n+1} - \sum_{i=0}^{n} \beta_{n,i} p_i$
$Ap_{n+1} = AK^{-1}r_{n+1} - \sum_{i=0}^{n} \beta_{n,i} Ap_i$
end
$x_0 = x_k$
repeat

図1 restarted GCR法

2.2 Efficient GCR法

各反復で解のベクトルx_{n+1}を必要としないなら，このx_{n+1}をループの外に出すことができる．さらに式変形を行うと，各反復で計算量の大きいp_{n+1}, Ap_{n+1}の計算のうち，p_{n+1}の計算を省くことができる．このアルゴリズムはYang[5]によって提案され，Efficient GCR (eGCR)法と呼ばれる．eGCR法のアルゴリズムを図2に示す．以下では，リスタート周期kのeGCR法をeGCR(k)と呼ぶ．

このeGCR(k)では計算量$O(kN)$のp_{n+1}の計算を省くことができ，かなりの計算量を削減できるが，依然としてGCR(k)法と同じく$(N \times N)$の行列2つ分のメモリを必要とする．

Select x_0, tol
$r_0 = b - Ax_0$, $\hat{u}_0 = K^{-1}r_0$
$Ap_0 = AK^{-1}r_0$
for $n = 0, 1, \cdots, k-1$ **until** $\|r_n\| \leq tol$ **do** :
begin
$\alpha_n = \frac{(Ap_n, r_n)}{(Ap_n, Ap_n)}$
$r_{n+1} = r_n - \alpha_n Ap_n$
$\hat{u}_{n+1} = K^{-1}r_{n+1}$
$\beta_{n,i} = \frac{(Ap_i, AK^{-1}r_{n+1})}{(Ap_i, Ap_i)}, \quad i \leq n$
$Ap_{n+1} = AK^{-1}r_{n+1} - \sum_{i=0}^{n} \beta_{n,i} Ap_i$
end
$x_0 = x_0 + \hat{U}_{k-1} B_{k-1}^{-1} a_{k-1}$
where
$a_k = (\alpha_0, \alpha_1, \cdots, \alpha_k)^H$
$\hat{U}_k = (\hat{u}_0, \cdots, \hat{u}_k)$
$$B_k = \begin{pmatrix} 1 & \beta_{0,0} & \cdots & \beta_{k-1,0} \\ & 1 & \cdots & \beta_{k-1,1} \\ & & \ddots & \vdots \\ & & \cdots & \beta_{k-1,k-1} \\ & & & 1 \end{pmatrix}$$
repeat

図2 efficient GCR (eGCR)法

2.3 Memory efficient GCR法

ここでは提案手法の一つであるmemory efficient GCR法を説明する．eGCR(k)では，$\hat{u}_n$とAp_nの2つのベクトルの履歴を保存しておかなければいけないが，$\hat{u}_n$をAp_nで表すことにより，メモリ使用量を減らすことができる．このアルゴリズムを，図3に示す．このアルゴリズムを，memory efficient GCR (meGCR)法とし，リスタート周期kのmeGCR法を，以下ではmeGCR(k)法と呼ぶ．

meGCR(k)法はeGCR(k)法をもとにしているので，計算量はeGCR(k)法と同程度であり，GCR(k)法よりも少ない．さらに，メモリ使用量はGCR(k)法，eGCR(k)法の約半分しか必要でない．

2.4 Unrolled GCR法

meGCR(k)法において，Ap_1とAp_2は

$$\begin{aligned} & Ap_1 \\ & = AK^{-1}(r_0 - \alpha_0 Ap_0) - \beta_{0,0} Ap_0 \\ & = -\alpha_0 \left(AK^{-1}\right) r_0 + (1 - \beta_{0,0}) AK^{-1} r_0, \\ & Ap_2 \\ & = AK^{-1}(r_0 - \alpha_0 Ap_0 - \alpha_1 Ap_1) \\ & -\beta_{1,0} Ap_0 - \beta_{1,1} Ap_1 \\ & = -\alpha_0\alpha_1 \left(AK^{-1}\right)^3 r_0 \\ & + (-\alpha_0 - \alpha_1 (1 - \beta_{0,0}) \alpha_0 \beta_{1,1}) \left(AK^{-1}\right)^2 r_0 \\ & + (1 - \beta_{1,0} - \beta_{1,1} (1 - \beta_{0,0})) AK^{-1} r_0, \end{aligned}$$

Select x_0, tol

$r_0 = b - Ax_0$, $Ap_0 = AK^{-1}r_0$

for $n = 0, 1, \cdots, k-1$ **until** $\|r_n\| \leq tol$ **do** :

begin

$\alpha_n = \frac{(Ap_n, r_n)}{(Ap_n, Ap_n)}$

$r_{n+1} = r_n - \alpha_n Ap_n$

$\beta_{n,i} = \frac{(Ap_i, AK^{-1}r_{n+1})}{(Ap_i, Ap_i)}, \quad i \leq n$

$$Ap_{n+1} = AK^{-1}r_{n+1} - \sum_{i=0}^{n} \beta_{n,i} Ap_i$$

end

$x_0 = x_0 + K^{-1}D_{k-1}C_{k-1}B_{k-1}^{-1}a_{k-1}$

where

$a_k = (\alpha_0, \alpha_1, \cdots, \alpha_k)^H$

$$B_k = \begin{pmatrix} 1 & \beta_{0,0} & \beta_{1,0} & \cdots & \beta_{k-1,0} \\ & 1 & \beta_{1,1} & \cdots & \beta_{k-1,1} \\ & & \vdots & \ddots & \vdots \\ & & & 1 & \beta_{k-1,k-1} \\ & & & & 1 \end{pmatrix}$$

$$C_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ & -\alpha_0 & -\alpha_0 & -\alpha_0 & -\alpha_0 \\ & & -\alpha_1 & -\alpha_1 & -\alpha_1 \\ & & & \ddots & \vdots \\ & & & & -\alpha_{k-1} \end{pmatrix}$$

$$D_k = \begin{pmatrix} r0 & Ap_0 & Ap_1 & \cdots & Ap_{k-1} \end{pmatrix}$$

repeat

図 3 memory efficient GCR (meGCR) 法

と表すことができる．帰納的に，Ap_n は以下のように $\left(AK^{-1}\right)^i r_0$ の線形和で表すことができる．

$$Ap_n = s_{n,0}AK^{-1}r_0 + s_{n,1}\left(AK^{-1}\right)^2 r_0 + \cdots + s_{n,n}\left(AK^{-1}\right)^{n+1} r_0$$

また，r_n は以下のように表せる．

$$r_n = t_{n,0}r_0 + t_{n,1}AK^{-1}r_0 + \cdots + t_{n,n}\left(AK^{-1}\right)^n r_0$$

ここで，係数 $s_{i,j}$ と $t_{i,j}$ は，前の反復で計算した $s_{i,j}$ と $t_{i,j}$，そして $\left(\left(AK^{-1}\right)^i r_0, \left(AK^{-1}\right)^j r_0\right)$ を使って計算することができる．従って，反復計算の前に $\left(AK^{-1}\right)^n r_0$ と $\left(\left(AK^{-1}\right)^i r_0, \left(AK^{-1}\right)^j r_0\right)$ を計算しておけば，$s_{i,j}$ と $t_{i,j}$ をスカラー計算によって得ることができ，これによって計算量の削減ができる．また，$\left(\left(AK^{-1}\right)^i r_0, \left(AK^{-1}\right)^j r_0\right)$ の計算は密行列積 (BLAS3) の計算になるので，効率的に計算できる．メモリ使用量については，覚えておかなければならないベクトルは $\left(AK^{-1}\right)^i r_0$ の履歴のみなので，meGCR(k) 法とほぼ同程度で，既存のアルゴリズムの約半分である．しかし，$\left(AK^{-1}\right)^i r_0$ は i が大きくなるにつれて A の dominant な固有ベクトルの方向を向いてくるので，リスタート周期を大きくすると精度の低下が予想される．この方針で計算するアルゴリズムを unrolled GCR(uGCR) 法とし，リスタート周期 k の uGCR 法を以下では uGCR(k) 法と呼ぶ．uGCR(k) 法のアルゴリズムを図 4 に示す．

2.5 計算量，メモリ使用量の比較

GCR 法のそれぞれのアルゴリズムについて，計算量，メモリ使用量を比較する．計算量については，浮動小数点計算の数とともに計算のタイプも重要になるので，計算要素を表 1 のように分類する．ここで，N は問題サイズ，k はリスタート周期である．この分類を使って，1 リスタート周期 (k 反復) の計算量は，表 2 のようになる．

表 1 計算要素の分類

計算要素	k 反復目の計算量	説明
dmv	$2kn$	密行列-ベクトル積
smv	行列構造に依存	疎行列-ベクトル積
dp	$2n$	ベクトルの内積
daxpy	$2n$	$x = x + \alpha y$
prec	前処理に依存	前処理
bin	$\frac{1}{3}k(k-1)(k+1)$	B^{-1}
kmv	$2k^2$	$(k \times k)$ 密行列-ベクトル積
dmm	$2k^2 n$	密行列-密行列積

表 2 1 リスタート周期の計算量の比較

計算要素	GCR(k)	eGCR(k)	meGCR(k)	uGCR(k)
dmv	$3(k-1)$	$2k-1$	$2k-1$	0
dp	$2k$	$2k$	$2k$	0
daxpy	$2k-1$	k	k	0
smv	k	k	k	k
prec	k	k	$k+1$	$k+1$
bin	0	1	1	1
kmv	0	1	2	$4k$
dmm	0	0	0	1

また，メモリ使用量は表 3 のようになる．

表 3 メモリ使用量の比較

buffer size	GCR(k)	eGCR(k)	meGCR(k)	uGCR(k)
N	$2k+3$	$2k+3$	$k+3$	$k+2$
k^2	0	2	2	5

表 1-3 の計算量，メモリ使用量の比較から，以下ことがわかる．(1)meGCR(k) 法の計算量は GCR(k) 法よりも少なく，eGCR(k) 法と同程度である．(2)meGCR(k) 法のメモリ使用量は GCR(k) 法，eGCR(k) 法の約半分である．(3)uGCR(k) 法は，4 つのアルゴリズムの中でもっとも計算量が少なく，メモリ使用量は meGCR(k) 法とほぼ同じである．

Select x_0, tol

$r_0 = b - Ax_0$

for $i = 0, 1, \cdots k-1$

calculate $\left(AK^{-1}\right)^i r_0$

end

for $i = 0, 1, \cdots k-1, \quad j = 0, 1, \cdots k-1$

$u_{i,j} = \left(\left(AK^{-1}\right)^i r_0\ ,\ \left(AK^{-1}\right)^j r_0\right)$

end

$s_{0,0} = 1$

for $n = 0, 1, \cdots k-1$

$$(Ap_n, Ap_n) = (s_{n,0}, s_{n,1}, \cdots, s_{n,n}) \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n+1} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,n+1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n+1,1} & u_{n+1,2} & \cdots & u_{n+1,n+1} \end{pmatrix} \begin{pmatrix} s_{n,0} \\ s_{n,1} \\ \vdots \\ s_{n,n} \end{pmatrix}$$

$$(Ap_n, r_n) = (s_{n,0}, s_{n,1}, \cdots, s_{n,n}) \begin{pmatrix} u_{1,0} & u_{1,1} & \cdots & u_{1,n} \\ u_{2,0} & u_{2,1} & \cdots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n+1,0} & u_{n+1,1} & \cdots & u_{n+1,n} \end{pmatrix} \begin{pmatrix} u_{n,0} \\ u_{n,1} \\ \vdots \\ u_{n,n} \end{pmatrix}$$

$\alpha_n = \frac{(Ap_n, r_n)}{(Ap_n, Ap_n)}$

$t_{n+1,i} = t_{n,i} - \alpha_n s_{n,i-1}\ (0 \le i \le n) \quad t_{n+1,n+1} = -\alpha_n s_{n,n}$

$$\begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} s_{0,0} & 0 & 0 & \cdots & 0 \\ s_{1,0} & s_{1,1} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{n,0} & s_{n,1} & s_{n,2} & \cdots & s_{n,n} \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n+2} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,n+2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n+1,1} & u_{n+1,2} & \cdots & u_{n+1,n+2} \end{pmatrix} \begin{pmatrix} t_{n+1,0} \\ t_{n+1,1} \\ \vdots \\ t_{n+1,n+1} \end{pmatrix}$$

$\beta_{n,i} = \frac{\left(Ap_i, AK^{-1} r_{n+1}\right)}{(Ap_i, Ap_i)}$

$s_{n+1,i} = t_{n+1,i} + \sum_{j=0}^{i} \beta_{n,j} s_{j,i}\ (0 \le i \le n) \quad s_{n+1,n+1} = t_{n+1,n+1}$

end

$x_0 = x_0 + K^{-1} E_{k-1} F_{k-1} C_{k-1} B_{k-1}^{-1} a_{k-1}$

where $a_k = (\alpha_0, \alpha_1, \cdots, \alpha_k)^H$

$$B_k = \begin{pmatrix} 1 & \beta_{0,0} & \beta_{1,0} & \cdots & \beta_{k-1,0} \\ & 1 & \beta_{1,1} & \cdots & \beta_{k-1,1} \\ & & \vdots & \ddots & \vdots \\ & & & 1 & \beta_{k-1,k-1} \\ & & & & 1 \end{pmatrix} \quad C_k = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ & -\alpha_0 & -\alpha_0 & -\alpha_0 & -\alpha_0 \\ & & -\alpha_1 & -\alpha_1 & -\alpha_1 \\ & & & \ddots & \vdots \\ & & & & -\alpha_{k-1} \end{pmatrix}$$

$$E_k = \left(r_0, AK^{-1} r_0\ ,\ \left(AK^{-1}\right)^2 r_0\ ,\ \cdots\ ,\ \left(AK^{-1}\right)^k r_0\right) \quad F_k = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & s_{0,0} & s_{1,0} & \cdots & s_{k-1,0} \\ 0 & 0 & s_{1,1} & \cdots & s_{k-1,1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s_{k-1,k-1} \end{pmatrix}$$

repeat

図 4 unrolled GCR (uGCR) 法

3. 評価実験

ここでは，それぞれのアルゴリズムを HITACHI SR2201 上で比較実行した結果を示す．SR2201 の各 PE の理論ピーク性能は 300MFlops，メモリは 256MByte，各 PE は 3 次元ハイパークロスバー網でつながれており，通信速度は 300MBytes/s である．

本実験で使用した問題は以下の 3 つである．

- **Problem 1**: 係数行列 A は Toeplitz 行列

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 1 & 0 & \cdots & 0 \\ \gamma & 0 & 2 & 1 & \cdots & 0 \\ 0 & \gamma & 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \gamma & 0 & 2 \end{pmatrix}$$

右辺ベクトル b は $(1,1,\cdots,1)^T$，γ は 1.0 とした．

- **Problem 2**: 領域 $\Omega = [0,1] \times [0,1]$ における楕円型偏微分方程式の境界値問題（2 次元）

$$-u_{xx} - u_{yy} + Ru_x = g(x,y),$$
$$u(x,y)|\partial\Omega = 1 + xy,$$

について，5 点中心差分によって離散化した．R は 1.0 とし，右辺は厳密解が $u = 1 + xy$ となるように定めた．

- **Problem 3**: 領域 $\Omega = [0,1] \times [0,1] \times [0,1]$ における楕円型偏微分方程式の境界値問題（3 次元）

$$-u_{xx} - u_{yy} - u_{zz} + Ru_x = g(x,y,z),$$
$$u(x,y)|\partial\Omega = 0.0,$$

について，7 点中心差分によって離散化した．R は 1.0 とし，右辺は厳密解が $u = e^{xyz}\sin(\pi x) \times \sin(\pi y) \times \sin(\pi z)$ となるように定めた．

終了条件は，$\|r_k\|/\|r_0\| \leq 1.0 \times 10^{-12}$ とした．

3.1 meGCR 法の実験結果

meGCR(k) 法と，既存手法の GCR(k) 法，eGCR(k) 法を比較した結果を示す．リスタート周期 k はすべて 32 とした．まず最初に，逐次環境での実行結果を示す．前処理無しの結果は表 4，B-ILU(0) 前処理有りの結果は表 5 に示す．これらの表に示された計算要素は，表 1 のものと同じである．これらの結果から，eGCR(k) 法と meGCR(k) 法はほぼ同じ速さで，GCR(k) 法よりも速いことがわかる．またこの実行時間の違いは，ほとんど dmv(密行列-密行列積) の計算時間の違いであることがわかる．GCR(k) 法の dmv の計算時間と，eGCR(k) 法，meGCR(k) 法の dmv の計算時間の比は大体表 2 のとおり 3 : 2 である．他の計算要素と比べて dmv の計算には多くの時間がかかるので，この部分にかかる時間が少ない eGCR(k) 法と meGCR(k) 法は GCR(k) に比べて非常に速くなっている．

また，並列環境での結果を図 5 と図 6 に示す．これらの図には異なる PE 台数で測った実行時間が示されている．これらの結果から，3 つのアルゴリズムの速さの関係は，並列の環境でも逐次の環境と同じ傾向を示すことがわかる．

3.2 uGCR 法の実験結果

次に，uGCR(k) 法と他の 3 つの方法の比較結果を表 6 示す．計算誤差の問題から，uGCR(k) はリスタート周期を 8 よりも大きくすると収束が非常に悪くなるので，ここではすべてリスタート周期を 8 に固定した結果を表 6 に示す．リスタート周期が 8 では，Problem 2 の収束が非常に遅いので，Problem 2 の結果はこの表に入っていない．

前処理なしの場合では，uGCR(k) 法は他の 3 つよりも速いことがわかる．しかし，前処理をつけると他の手法と同程度か，それ以下の速さになっている．これは，前処理に時間が多くかかるために uGCR(K) 法の計算量が少ないという利点がほとんど影響していないためだと考えられる．

4. おわりに

実験結果から，提案手法の一つである meGCR(k) 法は，逐次環境においても並列環境においても既存の eGCR(k) 法と実行時間がほぼ同じであることがわかった．また，meGCR(k) 法のメモリ使用量は既存の方法の約半分なのでこの方法は計算量を増やさずにメモリ使用量を約半分にできることができ，十分実用的であることがわかった．もうひとつの提案手法である uGCR(k) 法は計算誤差のためにリスタート周期を大きくできないという問題点があるが，リスタート周期が小さい場合に比較すると，他の方法よりも高速であることがわかった．

参考文献

1) van der Vorst, H. A. and Vuik, C.: GMRESR: A family of nested GMRES methods, Technical ReportDUT-TWI-91-80, Delft University of Technology, Department of Technical Mathematics and Informatics, Delft, The Netherlands (1991).

2) Eisenstat, S. C., Elman, H. C. and Schultz, M. H.: Variational Iterative Methods for Nonsymmetric Systems of Linear Equations, *SIAM Journal on Numerical Analysis*, Vol. 20, No. 2, pp. 345–357 (1983).

3) Saad, Y. and Schultz, M. H.: GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.*, Vol. 7, pp. 856–869 (1986).

4) 藤野清次，張紹良: 反復法の数理，朝倉書店 (1996).

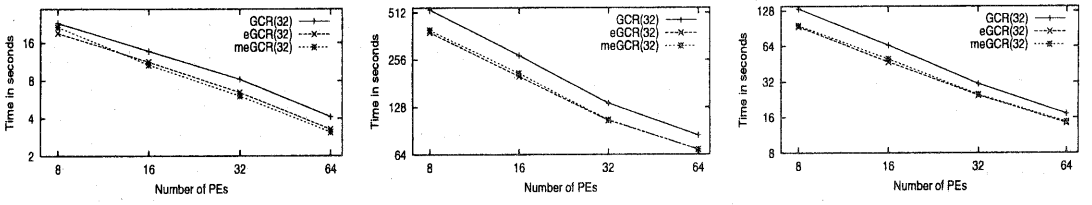
5) Yang, U. M.: A family of preconditioned iterative solvers for sparse linear systems, Technical Report 1904, Department of Computer Science, University of Illinois at Urbana-Champaign (1995).

表 4　実行時間 (秒) と反復回数: 逐次環境, 前処理なし

	Problem 1			Problem 2			Problem 3		
	GCR	eGCR	meGCR	GCR	eGCR	meGCR	GCR	eGCR	meGCR
iter.	46	46	46	20466	20466	20466	374	374	374
dmv	12.5	7.85	7.89	3410	2120	2130	23.6	14.4	15.1
smv	3.88	3.78	3.76	1030	1010	1010	9.82	9.53	9.97
dp	0.730	0.725	0.737	123	123	122	0.868	0.835	0.865
daxpy	1.24	0.542	0.546	219	97.6	95.9	1.53	0.709	0.695
bin	—	4.18×10^{-4}	4.43×10^{-4}	—	0.509	0.590	—	9.24×10^{-3}	1.05×10^{-2}
kmv	—	4.30×10^{-5}	2.29×10^{-4}	—	3.32×10^{-2}	1.83×10^{-2}	—	8.03×10^{-4}	3.31×10^{-3}
total time	22.8	18.3	17.9	4860	3440	3450	37.8	27.7	28.7

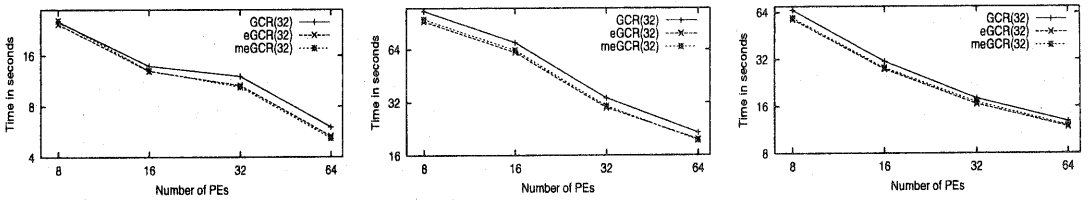
表 5　実行時間 (秒) と反復回数: 逐次環境, B-ILU(0) 前処理あり

	Problem 1			Problem 2			Problem 3		
	GCR	eGCR	meGCR	GCR	eGCR	meGCR	GCR	eGCR	meGCR
iter.	17	17	17	1893	1893	1893	85	85	85
dmv	5.31	3.39	3.29	317	200	200	5.57	3.32	3.50
smv	1.60	1.59	1.57	93.3	92.9	92.6	2.20	2.13	2.20
dp	0.282	0.282	0.286	11.5	11.5	11.4	0.205	0.203	0.203
daxpy	0.455	0.197	0.198	20.3	9.03	8.88	0.350	0.162	0.159
prec	8.30	9.00	9.20	492	494	508	11.4	11.9	11.8
bin	—	1.56×10^{-4}	1.61×10^{-4}	—	4.70×10^{-2}	5.93×10^{-2}	—	1.93×10^{-3}	2.22×10^{-3}
kmv	—	1.80×10^{-5}	8.60×10^{-5}	—	3.06×10^{-3}	1.70×10^{-2}	—	1.70×10^{-4}	7.10×10^{-4}
total time	21.2	19.8	20.1	938	812	825	21.9	19.9	20.1



Problem 1　Problem 2　Problem 3

図 5　並列環境, 前処理無しの実行時間 (秒)



Problem 1　Problem 2　Problem 3

図 6　並列環境, B-ILU(0) 前処理ありの実行時間 (秒)

表 6　uGCR(k) 法と他のアルゴリズムの比較:逐次環境での実行時間 (秒) と反復回数

		no preconditioning		B-ILU(0) preconditioning	
		iteration	time	iteration	time
Problem 1	GCR(k)	46	18.5	17	20.3
	eGCR(k)	46	15.4	17	18.2
	meGCR(k)	46	15.4	17	19.5
	uGCR(k)	55	13.5	25	26.7
Problem 3	GCR(k)	1096	64.8	150	30.6
	eGCR(k)	1096	53.6	150	29.6
	meGCR(k)	1096	55.7	150	31.7
	uGCR(k)	1053	43.0	150	30.1