GridRPC/MPI ハイブリッドによる修正多重リスタート付き Arnoldi 法

木原崇智[†] 小瀧義久[†]
多田野寛人[††] 櫻井鉄也[†,††]

本論文では，大規模非 Hermite 疎行列を持つ標準固有値問題の並列解法について述べる．大規模固有値問題の反復解法の 1 つとしてリスタート付き Arnoldi 法がある．この方法はリスタート周期の選び方によって，解への収束性が大きく異なるが，収束性の良いリスタート周期をあらかじめ与えることは困難である．多重リスタート付き Arnoldi 法は，複数のリスタート周期の Arnoldi 法を並列に実行し，残差ノルムが最小となる近似ベクトルを次のリスタートの初期ベクトルとする．しかしながら，この方法では残差ノルムが停滞する可能性がある．本論文ではこの問題の回避法を提案する．さらに，GridRPC システムである Ninf-G と MPI のハイブリッド環境で実装し数値実験を行った．

Modified Multiple Explicitly Restarted Arnoldi Method with Hybrid GridRPC/MPI Implementation

Takanori Kihara,[†] Yoshihisa Kodaki,[†] Hiroto Tadano[††] and Tetsuya Sakurai[†,††]

In this paper, we consider a parallel method for standard eigenvalue problems with large sparse non-Hermitian matrices. The explicitly restarted Arnoldi method is one of iterative solution technique for large-scale standard eigenvalue problems. The convergence of this method depends on a restart period. However, it is difficult to specify a restart period which has good convergences. The modified multiple explicitly restarted Arnoldi method is based on a multiple use of explicitly restarted Arnoldi method in order to improve the convergence. However, the residual norm of this method may stagnate. We propose a modified method to avoid this drawback and have implemented the modified method with hybrid of GridRPC and MPI. Some numerical examples illustrate the performance of the presented method.

1. はじめに

大規模非 Hermite 疎行列 $A \in \mathcal{C}^{n\times n}$ を持つ標準固有値問題

$$A\boldsymbol{u} = \lambda\boldsymbol{u}$$

の代表的な解法に，Krylov 部分空間法の一種である Arnoldi 法[1),12)] がある．同法には，精度や収束性向上のために多数の変種[11),13)] が存在するが，その中で基本的なものにリスタート付き Arnoldi 法（ERAM：Explicitly Restarted Arnoldi Method）[6),10)] がある．ERAM はリスタート周期によって解への収束性が大きく異なり，計算時間に大きな影響がある．しかしながら，計算前に収束性の良いリスタート周期を特定することは困難である．

ERAM の収束性を向上させる方法として，異なるリスタート周期の ERAM プロセスを同時に複数個実行する多重リスタート付き Arnoldi 法（MERAM：Multiple ERAM）[2),3)] がある．MERAM は各 ERAM プロセスが相互にデータをやりとりすることで収束性を向上させる．MERAM には同期型と非同期型があり，非同期型 MERAM は同期型 MERAM より計算時間が短い傾向にある．しかしながら，非同期型 MERAM には精度改善が停滞し，解が求められない場合がある．本論文では，この問題点について示し，その回避法として修正非同期型 MERAM を提案し性能を評価する．

本論文では，MERAM を GridRPC（RPC：Remote Procedure Call）システムの一種である Ninf-G[8)] と MPI（Message Passing Interface）のハイブリッド環

† 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
†† 科学技術振興機構戦略的創造研究推進事業
Core Research for Evolutional Science and Technology, Japan Science and Technology Agency

境で実装し，いくつかのテスト行列で性能を評価した．Grid コンピューティングとは，遠隔地に分散したヘテロな計算機資源をネットワークで結び，1 つの統合的な計算資源として利用する技術である．この技術により既存の計算機資源で大規模な並列計算が可能となるため，現在 Grid 環境での並列プログラミングを支援するための GridRPC の研究がさかんに行われている．

本論文では，クライアント・サーバ型の GridRPC システムである Ninf-G[8] を用いて，MERAM の各 ERAM を並列化した．MERAM はアルゴリズムの性質上，クライアント・サーバ型に適している．また，Ninf-G は 1 度サーバ側に送ったデータをサーバ側で保持しておくことが可能であり，MERAM の場合初めに行列データをサーバ側に送ると，2 回目以降のリスタート計算時には改めて行列データを送る必要がない．したがって，効率的な並列計算が実現できる．特に，行列のサイズが巨大な場合，この機能がなければ実行時間の大部分をデータの転送時間が占めることとなるため，この機能が大きな意味を持つ．文献 2) では MERAM を NetSolve[7] で実装しているが，リスタートごとに行列データを送り直しているため，実験で用いた行列のサイズは小さいにもかかわらずデータ転送時間がボトルネックになると考えられる．

さらに，各 ERAM プロセスの計算時間は，その大部分を行列ベクトル積が占める．したがって，本論文では，各 ERAM プロセスの行列ベクトル積を MPI で並列化することによりさらなる高速化を図った．

本論文の構成は次のようになっている．まず，2 章ではリスタート付き Arnoldi 法（ERAM）について説明する．次に 3 章では，同時に複数の ERAM を実行する多重リスタート付き Arnoldi 法（MERAM）について説明し，その問題点を示す．そして，その問題を回避するための方法を示す．4 章では，4 つの数値例を用いて MERAM の性能を評価し，提案法の有効性について検証する．最後に，5 章で結論と今後の課題を述べる．また，本論文を通して内積は $(\boldsymbol{f},\boldsymbol{g})\equiv \boldsymbol{f}^{\mathrm{H}}\boldsymbol{g}$ で定義する．

2. リスタート付き Arnoldi 法：ERAM

2.1 Arnoldi 法

基本となる Arnoldi 法（BA：Basic Arnoldi）について説明する．行列 A と非零ベクトル $\boldsymbol{v}$ によって生成される m 次元の Krylov 部分空間は

$$\mathcal{K}_m = \mathrm{Span}(\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{m-1}\boldsymbol{v}) \qquad (1)$$

で表される（以下，Krylov 部分空間 $\mathcal{K}_m$ の次元数 m をリスタート周期と呼ぶ）．まず Krylov 部分空間 $\mathcal{K}_m$ の正規直交基底 $\{\boldsymbol{w}_i\}_1^m$ を計算する．ここで，$W_m = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m)$ とおくと

$$AW_m = W_m H_m + h_{m+1,m}\boldsymbol{w}_{m+1}\boldsymbol{e}_m^{\mathrm{T}},$$
$$H_m = W_m^{\mathrm{H}} A W_m \qquad (2)$$

が成り立つ[1),10)]．ただし，$H_m(= h_{i,j})$ は $m\times m$ 上 Hessenberg 行列，$h_{m+1,m}$ はスカラー値を表し，$\boldsymbol{e}_m$ は第 m 成分のみが 1 の m 次元ベクトルである．次に，行列 H_m の固有対（固有値：$\lambda_i^{(m)}$，固有ベクトル：$\boldsymbol{y}_i^{(m)}$）を求める．ここで得られた固有値 $\lambda_i^{(m)}$ が，行列 A の近似固有値（絶対値最大の固有値から順に m 個）であり，Ritz 値と呼ばれる．また，

$$\boldsymbol{u}_i = W_m \boldsymbol{y}_i, \quad (i = 1, 2, \ldots, m) \qquad (3)$$

により得られるベクトル $\boldsymbol{u}_i^{(m)}$ が，$\lambda_i^{(m)}$ に対応する行列 A の m 本の近似固有ベクトルとなる．$\boldsymbol{u}_i^{(m)}$ は Ritz ベクトルと呼ばれる．行列 A の求めるべき固有対（Ritz pair）の数が s（$s < m$）組の場合，行列 H_m の固有対を絶対値最大の固有値から順に s 組求めればよい．以下に Arnoldi 法（BA）のアルゴリズムを示す．ここで ρ は残差ノルムであり，$\boldsymbol{r}$ は ρ で構成したベクトルである．また，$\Lambda_s^{(m)}$，$U_s^{(m)}$ は $\Lambda_s^{(m)} = (\lambda_1^{(m)}, \ldots, \lambda_s^{(m)})$，$U_s^{(m)} = (\boldsymbol{u}_1^{(m)}, \ldots, \boldsymbol{u}_s^{(m)})$ である．

Basic Arnoldi Algorithm : BA
(Input : A, s, m, $\boldsymbol{v}$, Output : $\boldsymbol{r}_s$, $\Lambda_s^{(m)}$, $U_s^{(m)}$)

1. $\boldsymbol{w}_1 = \boldsymbol{v}/\|\boldsymbol{v}\|_2$
2. **For** $j = 1, 2, \ldots, m$ **do**:
   $\hat{\boldsymbol{w}}_j = A\boldsymbol{w}_j$,
   For $i = 1, 2, \ldots, j$ **do**:
   $h_{i,j} = (\boldsymbol{w}_i, \hat{\boldsymbol{w}}_j)$,
   $\hat{\boldsymbol{w}}_j = \hat{\boldsymbol{w}}_j - h_{i,j}\boldsymbol{w}_i$,
   End
   $h_{j+1,j} = \|\hat{\boldsymbol{w}}_j\|_2$,
   $\boldsymbol{w}_{j+1} = \hat{\boldsymbol{w}}_j / h_{j+1,j}$,
   End
3. Compute the eigenpairs of H_m and select the s desired ones.
4. Compute the s associate Ritz vectors $\hat{\boldsymbol{u}}_i^{(m)} = W_m \boldsymbol{y}_i^{(m)}$, $(i = 1, 2, \ldots, s)$.
5. Normalize the s Ritz vectors $\boldsymbol{u}_i^{(m)} = \hat{\boldsymbol{u}}_i^{(m)} / \|\hat{\boldsymbol{u}}_i^{(m)}\|_2$, $(i = 1, 2, \ldots, s)$.
6. Compute $\boldsymbol{r}_s = (\rho_1, \ldots, \rho_s)$ with $\rho_i = \|(A - \lambda_i^{(m)} I)\boldsymbol{u}_i^{(m)}\|_2$.

2.2 ERAM への拡張

2.1 節で説明した BA にリスタート処理を加えた，

リスタート付き Arnoldi 法（ERAM）について説明する．BA で得られた Ritz pair が十分な精度でない場合，初期ベクトル $\boldsymbol{v}$ を，BA で得られた Ritz ベクトル $U_m = (\boldsymbol{u}_1^{(m)}, \ldots, \boldsymbol{u}_s^{(m)})$ を用いて

$$\boldsymbol{v} = \alpha_1 \boldsymbol{u}_1^{(m)} + \cdots + \alpha_s \boldsymbol{u}_s^{(m)}, (\forall \{\alpha_i\}_{i=1}^s \in \boldsymbol{C}) \tag{4}$$

と定め BA をリスタートさせる．この処理を残差ノルムベクトル $\boldsymbol{r}_s$ の最大ノルムが収束判定値（tol）以下になるまで繰り返す．式 (4) での $\{\alpha_i\}_{i=1}^s$ の選び方については文献 10) で詳しく述べられている．以下に ERAM のアルゴリズムを示す．

ERAM Algorithm :
(Input : A, s, m, $\boldsymbol{v}$, Output : $\boldsymbol{r}_s$, $\Lambda_s^{(m)}$, $U_s^{(m)}$)

1. **Start**. Choose a parameter m and an initial vector $\boldsymbol{v}$.
2. **Iterate**. Compute a **BA** step.
3. **Restart**. If $\|\boldsymbol{r}_s\|_\infty > tol$ then update the initial vector $\boldsymbol{v}$ by (4) and goto 2.

3. 多重リスタート付き Arnoldi 法：MERAM

MERAM は，異なるリスタート周期の複数の ERAM プロセスを並列に実行し，相互にデータをやりとりをすることで収束性を向上させる解法である．MERAM には同期型と非同期型がある．MERAM のアルゴリズムを以下に示す．$\boldsymbol{r}_s^i = (\rho_1^i, \ldots, \rho_s^i)$ は i 番目の ERAM プロセスの残差ノルムベクトルであり，ρ_j^i は i 番目の ERAM プロセスの j 番目の Ritz pair の残差ノルムである．また，$M = (m_1, \ldots, m_l)$ は l 個の ERAM プロセスのそれぞれ異なるリスタート周期である．

MERAM Algorithm :
(Input : A, s, M, $\boldsymbol{v}$, Output : $\boldsymbol{r}_s$, $\Lambda_s^{(m)}$, $U_s^{(m)}$)

1. **Start**. Choose an initial vector $\boldsymbol{v}$ and a set of subspace sizes $M = (m_1, \ldots m_l)$.
2. **Iterate**. **For** $i = 1, \ldots, l$ **do** in parallel:
   (a) **Receiv_Initial_Data**
   (b) Compute a **BA** step.
   (c) If $\|\boldsymbol{r}_s^i\|_\infty \leq tol$ then stop all processes.
   (d) **Send_Eigen_Info**
3. **Restart**. Update the initial vector $\boldsymbol{v}$ by (5) and goto 2.

3.1 同期型 MERAM

上記アルゴリズム Step1 はクライアント側で行う．l 個の ERAM プロセスのそれぞれ異なるリスタート周期 $M = (m_1, \ldots m_l)$ を定め，初期ベクトル $\boldsymbol{v}$ とともにサーバ側へ送る．

Step2 はサーバ側で行う．Step2.(a) では，Step1 で与えたリスタート周期，初期ベクトルで l 個の ERAM プロセスをそれぞれ並列に実行する．いずれかの ERAM プロセスから得られた Ritz pair の残差ノルムが収束判定値（tol）以下であった場合，すべてのプロセスを終了させる．一方，Step2.(c) を満たさなかった場合は，得られた Ritz ベクトルと残差ノルムデータをクライアントに返す（Step2.(d)）．

Step3 はクライアント側で行う．すべての ERAM プロセスから結果が得られたら，その結果を用いて初期ベクトル $\boldsymbol{v}$ を更新し，新たな初期ベクトルをサーバ側に送り，各 ERAM プロセスをリスタートさせる．

ここで，Step3 における初期ベクトル $\boldsymbol{v}$ の更新方法について述べる．まず，$\rho_j^p \leq \rho_j^q$ のとき，$\boldsymbol{u}_j^{(m_p)}$ は $\boldsymbol{u}_j^{(m_q)}$ よりも “better” であると定義する．そして，各 ERAM プロセスから s 組の Ritz pair の “best” の Ritz ベクトル $U_s^{\text{best}} = (\boldsymbol{u}_1^{\text{best}}, \ldots, \boldsymbol{u}_s^{\text{best}})$ をそれぞれ選ぶ．この U_s^{best} を用いて，Step3 での新たな初期ベクトル $\boldsymbol{v}$ を

$$\boldsymbol{v} = \alpha_1 \boldsymbol{u}_1^{\text{best}} + \cdots + \alpha_s \boldsymbol{u}_s^{\text{best}}, (\forall \{\alpha_i\}_{i=1}^s \in \boldsymbol{C}) \tag{5}$$

と定める．式 (5) で $\{\alpha_i\}_{i=1}^s$ の定め方については文献 10) を参照されたい．

3.2 同期型 MERAM の問題点

並列に実行する各 ERAM プロセスの計算時間は，リスタート周期や計算機性能によって大きく異なる．しかしながら，同期型 MERAM では Step3 においてすべてのプロセスを同期させなければならず，各 ERAM プロセスをそれぞれ異なる計算機で並列処理させた場合，一番遅いプロセスの結果を他の全プロセスが待たなければならないという欠点がある．この問題を回避する方法として，非同期型 MERAM がある．

3.3 非同期型 MERAM

Step3 において，i 番目の ERAM プロセスから結果の Ritz ベクトル $(\boldsymbol{u}_1^{(m_i)}, \ldots, \boldsymbol{u}_s^{(m_i)})$，残差ノルムが得られるたびに，その時点でクライアントが得ていた “best” な Ritz ベクトル $(\boldsymbol{u}_1^{\text{best}}, \ldots, \boldsymbol{u}_s^{\text{best}})$ と $(\boldsymbol{u}_1^{(m_i)}, \ldots, \boldsymbol{u}_s^{(m_i)})$ を比べる．そしてより残差ノルムが小さいほうの Ritz ベクトルを新たな $(\boldsymbol{u}_1^{\text{best}}, \ldots, \boldsymbol{u}_s^{\text{best}})$ として更新し，式 (5) より新たな

初期ベクトル $\boldsymbol{v}$ を定める．ここで得られた初期ベクトル $\boldsymbol{v}$ を i 番目の ERAM プロセスへ与え，リスタートさせる．上記操作を Step2.(c) が満たされるまで繰り返す．

この方法では同期型 MERAM で見られた Step3 での同期がなくなり，サーバ側計算機がアイドル状態になるのを避けることが可能となる．

3.4 非同期型 MERAM の問題点

非同期型 MERAM は計算過程で最も残差ノルムが小さい Ritz ベクトル $U_s^{\rm best} = (\boldsymbol{u}_1^{\rm best}, \ldots, \boldsymbol{u}_s^{\rm best})$ を用いて，次のリスタートの初期ベクトル $\boldsymbol{v}$ を決定する．この性質上，すべての ERAM プロセスの結果の Ritz ベクトルで $U_s^{\rm best}$ が更新されなかった場合，再び同じ初期ベクトル $\boldsymbol{v}$ で各 ERAM が実行される．したがって，同じ計算の繰返しとなり，それ以上精度が更新されず無限ループに陥る．以下，このことを精度改善の停滞と呼ぶ．

3.5 修正非同期型 MERAM

3.4 節で示した精度改善の停滞を以下の方法で回避する．すべての ERAM プロセスの結果の Ritz ベクトル $\{(\boldsymbol{u}_j^{(m_i)}, \ldots, \boldsymbol{u}_j^{(m_i)}),\ (i = 1, \ldots, l)\}_{j=1}^{s}$ のすべての残差ノルムが，クライアントが保持していた $U_s^{\rm best}$ の残差ノルムより大きいと確認された時点で $\{(\boldsymbol{u}_j^{(m_i)}, \ldots, \boldsymbol{u}_j^{(m_i)}),\ (i = 1, \ldots, l)\}_{j=1}^{s}$ の中から最も残差ノルムが小さい Ritz ベクトル $(\boldsymbol{u}_1^{\rm best}, \ldots, \boldsymbol{u}_s^{\rm best})$ を選び，式 (5) より新たな初期ベクトルを定める．この処理により，残差ノルムは一時的に悪化するが精度改善の停滞は避けられる．以下，同処理を加えた非同期型 MERAM を修正非同期型 MERAM と呼ぶ．

精度改善の停滞が生じたことは，以下の方法で判断する．

(a) 各 ERAM プロセスは初期値 **false** の **flag** を持つ．
(b) i 番目の ERAM プロセスから得られた Ritz ベクトル $(\boldsymbol{u}_1^{(m_i)}, \ldots, \boldsymbol{u}_s^{(m_i)})$ で，クライアントが保持していた Ritz ベクトル $U_s^{\rm best}$ が更新されなかった場合，i 番目の ERAM プロセスの **flag** を **true** にする．
(c) i 番目の ERAM プロセスから得られた Ritz ベクトル $(\boldsymbol{u}_1^{(m_i)}, \ldots, \boldsymbol{u}_s^{(m_i)})$ で，クライアントが保持していた Ritz ベクトル $U_s^{\rm best}$ が更新された場合，すべての ERAM プロセスの **flag** を **false** にする．
(d) すべての ERAM の **flag** が **true** になったら，精度改善の停滞が生じたと見なす．

4. 数値実験

4.1 実験環境

MERAM を GridRPC システム Ninf-G と，MPI のハイブリッド環境で実装した（**図 1**）．複数の ERAM プロセスを並列に実行する部分は Ninf-G（バージョン 4.2.1）を用い，各 ERAM 内の行列ベクトル積は MPI（GridMPI：バージョン 0.11）を用い並列化した．

ERAM の収束判定値 *tol* は 1.0×10^{-6} とした．また，MERAM の挙動の解析を容易にするため求める固有対は 1 組（$s = 1$）とした．また，式 (5) における $\{\alpha_i\}_{i=1}^{s}$ はすべてスカラー値 1.0 とした．

表 1 に MERAM の各 ERAM のリスタート周期を示す．表 1 のリスタート周期は，ほぼ 10 から 50 の間で等間隔になるように決定した．また実験結果の計算時間には，Ninf-G における初期化時間，Handle の作成時間，Handle の破棄時間，プロセスの終了時間は含めないものとした．

実験は産業技術総合研究所の F-32 クラスタ上で行った．F-32 クラスタは，2 つの Intel 社製 Xeon プロセッサ（3.06 GHz）を有する Linux Network 社製 Evolocity2e サーバ 268 台がギガビットイーサネットでつな

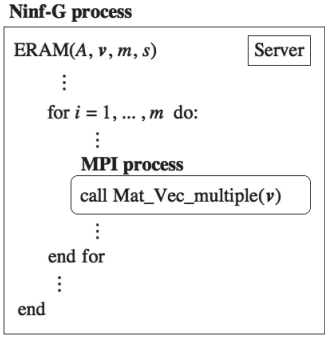

図 1 サーバ側における MERAM の Ninf-G/MPI による実装
Fig. 1 Ninf-G/MPI hybrid implemenation of MERAM in servers.

表 1 MERAM のリスタート周期
Table 1 Restart Periods of MERAM.

#ERAMs	リスタート周期
2	20, 40
3	10, 30, 50
4	12, 24, 36, 48
5	10, 20, 30, 40, 50
6	10, 18, 26, 34, 42, 50
7	12, 18, 24, 30, 36, 42, 48
8	12, 17, 22, 27, 32, 37, 42, 47
9	10, 15, 20, 25, 30, 35, 40, 45, 50
10	12, 16, 20, 24, 28, 32, 36, 40, 44, 48

表 2 テスト行列の次元数と非零要素数
Table 2 Number of dimensions and non-zero elements of test matrices.

行列	次元数	非零要素数
PDE810000	810,000	4,046,400
PDE262144	262,144	1,308,672
RAN12000	12,000	4,800,000
RAN60000	60,000	15,000,000

がれ，全体で 3.3TFLOPS の総演算性能と 1.1 TB のメインメモリ，99 TB のストレージを持つ．本論文では，クライアント・サーバとも F-32 クラスタを利用した．したがって，クライアント・サーバ間の通信遅延時間は実際の Grid 環境での実装に比べ短くなっている．しかしながら，各 ERAM プロセスの粒度が大きいために，通信遅延時間の影響は比較的小さいと考えられる．

4.2 テスト行列

数値実験では，Matrix Market [5] で提供されている行列生成プログラム：MATPDE を用いて生成した行列 PDE810000，行列生成プログラム：MATRAN を用いて生成した 2 つの行列 RAN12000，RAN60000，および Phase [9] で提供されている行列生成プログラムを用いて生成した PDE262144 をテスト行列として用いた．

PDE810000 は 2 次元偏微分方程式を 5 点中心差分で離散化して得られた行列であり，PDE262144 は 3 次元偏微分方程式を 7 点中心差分で離散化して得られた行列である．MERAM の提案者である Emad らの論文[3] で用いられている行列で最も大規模な行列は，この MATPDE を用いて生成された行列（次元数 490,000，非零要素数 2,447,200）であった．本論文ではよりサイズの大きい行列で実験するため PDE810000 を生成し実験した．また，2 次元偏微分方程式と 3 次元偏微分方程式の対比のため PDE262144 を用いた．

一方，RAN12000，RAN60000 は LAPACK [4] の乱数発生関数 DLARAN を用いて作成した行列である．これらの行列は，特定の応用分野で現れる行列ではないが，次元数，非零要素数を任意で決められるためテスト行列に適している．PDE810000，PDE262144 が非常にスパースな行列であるため，その対比として RAN12000，RAN60000 は非零要素数の割合が比較的大きいものにした．各行列の次元数，非零要素数を **表 2** に示す．

これら 4 つの行列に同期型 MERAM，非同期型 MERAM，および修正非同期型 MERAM を適用した．

4.3 数 値 例

実験 1：実験 1 では PDE810000 を用いた．**図 2** は PDE810000 に ERAM を適用した場合のリスタート周期に対する計算時間の結果である．図 2 より，リスタート周期によって計算時間が大きく異なることが分かる．また，リスタート周期と計算時間の相関関係は見られない．したがって，ERAM を実行する前に計算時間の短いリスタート周期を推定することは困難である．

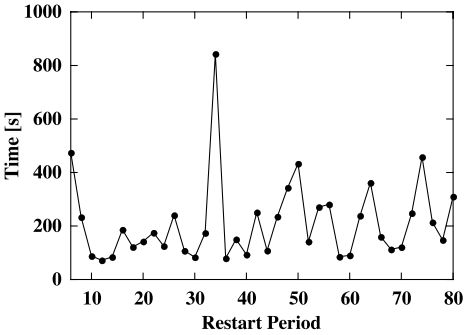

図 2 ERAM のリスタート周期に対する計算時間（PDE810000）
Fig. 2 Computation time for each restart period of ERAM (PDE810000).

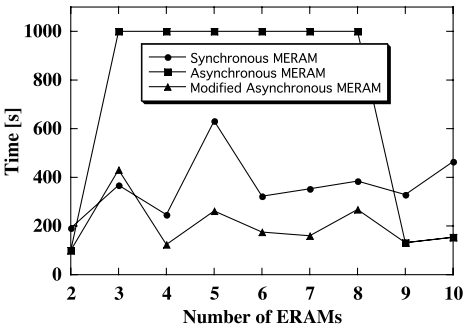

図 3 ERAM の数に対する計算時間（PDE810000）
Fig. 3 Computation times for number of ERAM processes (PDE810000).

図 3 は PDE810000 に同期型 MERAM，非同期型 MERAM，および修正非同期型 MERAM を適用した場合の計算時間の結果である．ただし，計算時間の上限は 1,000 秒に設定した．また横軸は並列に実行する ERAM の数を示している．図 3 から，同期型 MERAM では並列に実行する ERAM の数によらず解に収束しているが，非同期型 MERAM では解に収束しない場合が確認できる．ただし，解に収束する場合には非同期型 MERAM の計算時間は同期型 MERAM の計算時間より短い．また，修正非同期型 MERAM では非同期 MERAM では解に収束しない場合も収束し，計算時間も同期型 MERAM に比べ速いことが確認できる．しかしながら，PDE810000 は ERAM でも比較的簡単に解が求まり，MERAM を適用する有

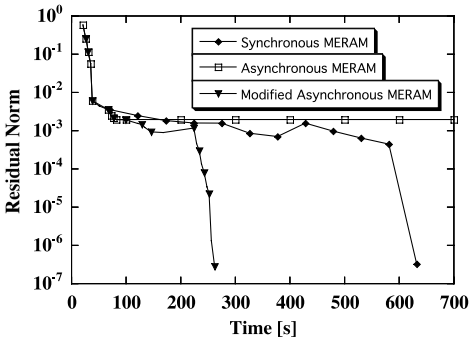

図 4 残差ノルムの収束過程（PDE810000）
Fig. 4 Convergence behaviors of residual norm (PDE810000).

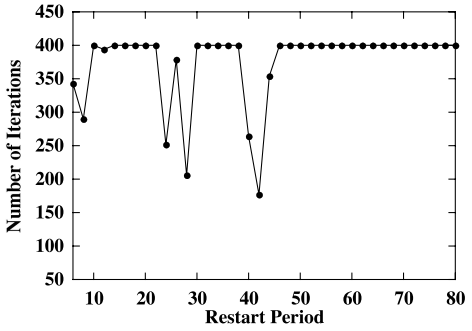

図 5 ERAM のリスタート周期に対する反復回数（PDE262144）
Fig. 5 Number of iteration for each restart period of ERAM (PDE262144).

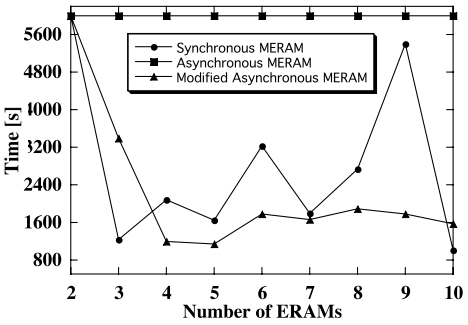

図 6 ERAM の数に対する計算時間（PDE262144）
Fig. 6 Computation times for number of ERAM processes (PDE262144).

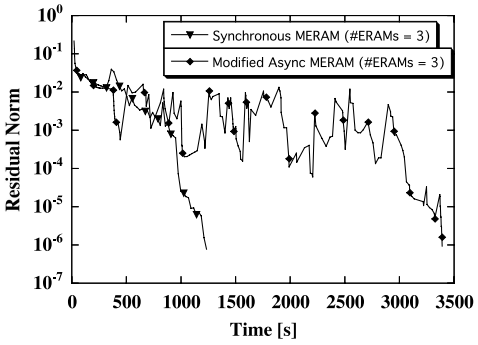

図 7 残差ノルムの収束過程（PDE262144，ERAM 数=3）
Fig. 7 Convergence behaviors of residual norm (PDE262144, #ERAM process=3).

用性は見られなかった．

図 4 は PDE810000 に並列に実行する ERAM の数が 5 つの場合の同期型 MERAM，非同期型 MERAM，および修正非同期型 MERAM を適用した場合の残差ノルムの収束過程である．図 4 より，非同期型 MERAM では 100 秒付近から残差ノルムの停滞が生じていることが分かる．同期型 MERAM と非同期型 MERAM を比べると，230 秒付近までは残差ノルムがほぼ一致しているが，その後修正非同期型 MERAM は急激に残差ノルムが減少した．

実験 2：実験 2 では PDE262144 を用いた．**図 5** は PDE262144 を ERAM に適用した場合のリスタート周期に対する反復回数の結果である．最大反復回数は 400 回に設定した．図 5 より，一部のリスタート周期では解に収束するが，ほとんどのリスタート周期で解に収束しないことが確認できる．

図 6 は PDE262144 に同期型 MERAM，非同期型 MERAM，および修正非同期型 MERAM を適用した場合の計算時間の結果である．ただし，計算時間の上限は 6,000 秒に設定した．また横軸は並列に実行する ERAM の数を示している．図 6 より，非同期型 MERAM では解に収束しないが，同期型 MERAM，修正非同期型 MERAM では並列に実行する ERAM の数が 3 つ以上の場合に解に収束したことが確認できる．また計算時間は，おおむね同期型 MERAM より修正非同期型 MERAM の方が短かった．

ここで，並列に実行する ERAM の数が 3 の場合に，修正非同期型 MERAM より同期型 MERAM の方が計算時間が短かったこと，および並列に実行する ERAM の数が 9 の場合に同期型 MERAM の計算時間が大きく悪化したことに着目し，上記 2 つの場合の同期型 MERAM と修正非同期型 MERAM の残差ノルムの収束過程をそれぞれ**図 7**，**図 8** に示す．図 7 より，同期型 MERAM は 1,000 秒付近で急速に解に収束していったのに対し，修正非同期型 MERAM は 3,000 秒付近まで残差ノルムの増減を繰り返し収束を遅くさせていた．また図 8 より，修正非同期型 MERAM では 1,500 秒付近で急速に解に収束しているのに対し，同期型 MERAM では 3,500 秒付近まで残差ノルムがほとんど変化せず収束を遅くさせていたことが確認できる．**図 9** は PDE262144 に並列に実行する ERAM

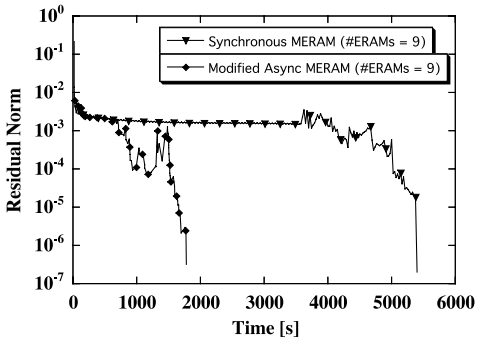

図 8 残差ノルムの収束過程（PDE262144，ERAM 数=9）
Fig. 8 Convergence behaviors of residual norm (PDE262144, #ERAM process=9).

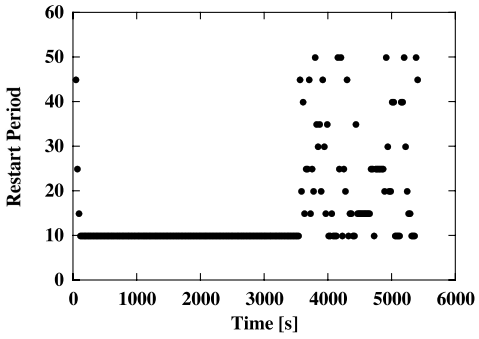

図 9 同期型 MERAM において採用されたリスタート周期（PDE262144）
Fig. 9 Adopted restart period in synchronous MERAM (PDE262144).

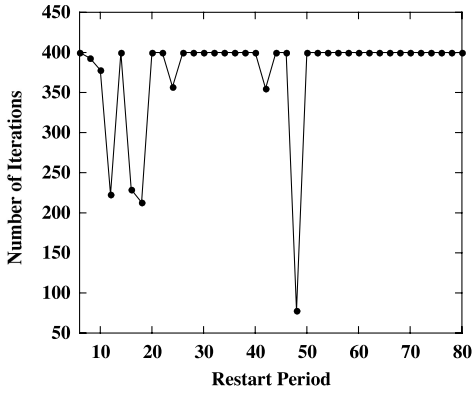

図 10 ERAM のリスタート周期に対する反復回数（RAN12000）
Fig. 10 Number of iteration for each restart period of ERAM (RAN12000).

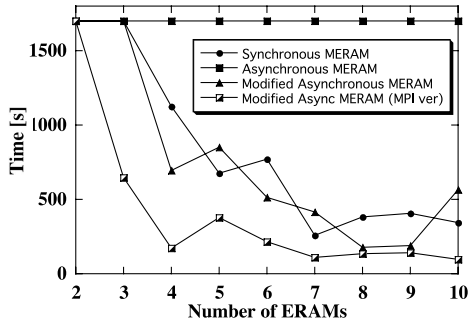

図 11 ERAM の数に対する計算時間（RAN12000）
Fig. 11 Computation times for number of ERAM processes (RAN12000).

の数が 9 の同期型 MERAM を適用した場合，同期時にどのリスタート周期が最も残差ノルムが小さかったかを示している．図 9 より，図 8 で見られた同期型 MERAM の残差ノルムの停滞は，同期型 MERAM の中でリスタート周期が 10 の ERAM 以外が機能しておらず，リスタート周期が 10 の ERAM を 1 つ実行している場合と変わらなかったためだと考えられる．

実験 3：実験 3 では RAN12000 を用いた．**図 10** は RAN12000 に ERAM を適用した場合のリスタート周期に対する反復回数の結果である．ただし，最大反復回数は 400 回に設定した．図 10 より，一部のリスタート周期では解に収束するが，ほとんどのリスタート周期で解に収束しないことが確認できる．

図 11 は RAN12000 に同期型 MERAM，非同期型 MERAM，修正非同期型 MERAM，および修正非同期型 MERAM を適用し，1 つの ERAM につき 15 CPU で MPI で並列化した場合の計算時間である．ただし，計算時間の上限は 1,700 秒に設定した．また横軸は並列に実行する ERAM の数を示している．図 11 より，非同期型 MERAM では解に収束しないが，同期型 MERAM，修正非同期型 MERAM では並列に実行する ERAM の数が 4 以上の場合に解に収束したことが確認できる．また，同期型 MERAM と修正非同期型 MERAM の実行時間はおおむね同程度であった．修正非同期型 MERAM を 1 つの ERAM につき 15 CPU で MPI で並列化した場合は，最大で 6 倍程度のスピードアップが実現できたが，大きな台数効果は得られなかった．

ここで図 11 より，修正非同期型 MERAM において並列に実行する ERAM の数が 9 の場合に比べ，10 の場合に計算時間が約 3 倍になったことに着目し，両者の場合の残差ノルムの収束過程を**図 12** に示す．図 12 より，並列に実行する ERAM の数が 9 の場合は 200 秒付近で解に収束したのに対し，10 の場合には 450 秒付近まで残差ノルムが増減を繰り返し収束を遅らせていたことが分かる．

表 3 は，図 10 で解に収束しなかったリスタート周期の中から，6 から 50 の間でほぼ等間隔になるよう

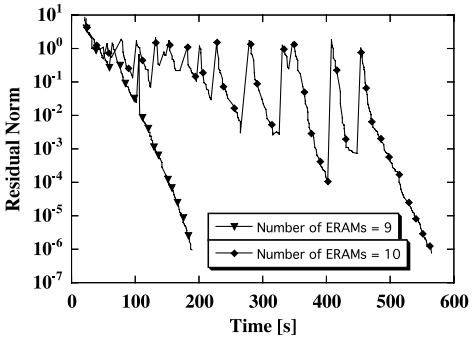

図 12 残差ノルムの収束過程（RAN12000）
Fig. 12 Convergence behaviors of residual norm (RAN12000).

表 3 実験 2 における MERAM のリスタート周期
Table 3 Restart periods of MERAM in example2.

#ERAMs	リスタート周期
2	20, 40
3	14, 30, 50
4	14, 24, 36, 46
5	6, 20, 30, 40, 50
6	6, 14, 20, 28, 38, 46
7	6, 14, 24, 30, 36, 44, 50
8	6, 14, 22, 28, 30, 34, 40, 46
9	6, 14, 22, 28, 32, 36, 40, 44, 50
10	6, 14, 20, 26, 30, 34, 38, 40, 44, 46

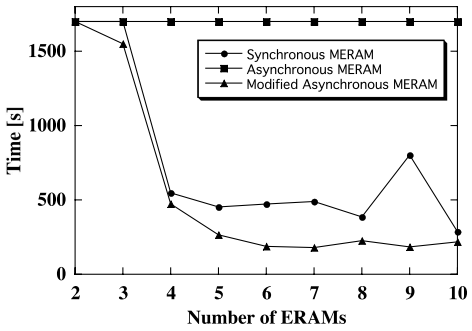

図 13 ERAM の数に対する計算時間（RAN12000）
Fig. 13 Computation times for number of ERAM processes (RAN12000).

に選んだ MERAM のリスタート周期表である．**図 13** は，RAN12000 に表 3 で示したリスタート周期を持つ同期型 MERAM，非同期型 MERAM，および修正非同期型 MERAM を適用した場合の計算時間の結果である．図 13 より ERAM では解に収束しなかったリスタート周期でも，同期型 MERAM，修正非同期型 MERAM では解に収束することが確認できる．また計算時間は同期型 MERAM より修正非同期型 MERAM のほうが短かった．

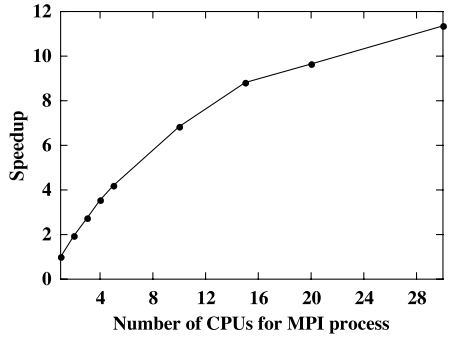

図 14 ERAM における行列ベクトル積部分の並列化の効果（RAN60000）
Fig. 14 Effectiveness of matrix-vector multiply parallelization of ERAM (RAN60000).

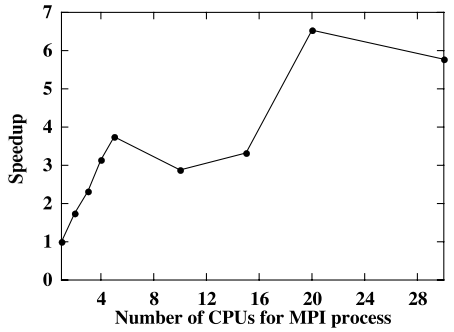

図 15 MERAM における行列ベクトル積部分の並列化の効果（RAN60000）
Fig. 15 Effectiveness of matrix-vector multiply parallelization of MERAM (RAN60000).

実験 4：実験 4 では RAN60000 を用いた．行列ベクトル積を MPI で並列化した場合，1 回あたりの ERAM（リスタート周期 30）は **図 14** に示される性能向上を得た．図 14 より，行列ベクトル積を MPI で並列化することで，ERAM の計算時間は最大で 10 倍以上高速になったことが確認できる．一方，行列ベクトル積を MPI で並列化した場合，リスタート周期（10，20，30，40，50）の修正非同期型 MERAM は**図 15** で示される性能向上を得た．修正非同期型 MERAM の場合では最大で約 7 倍のスピードアップが確認できた．また，行列ベクトル積並列化の CPU 数が 10 台，15 台の場合にスピードアップが落ちている原因は，行列ベクトル積を並列化し高速化したことでリスタート周期が大きい ERAM の結果がクライアント側に返る頻度が増したが，それが逆に精度改善の悪化を招き全体の反復回数を増加させたためである．したがって，大きいリスタート周期の ERAM が効果的に働く行列に対しては，行列ベクトル積の並列化により台数効果以上のスピードアップが期待できる．また，より大規模

な行列を適用することで行列ベクトル積の並列効果はさらに大きくなると考えられる．

ERAM では解くことができない問題を MERAM では解くことが可能な場合があり，本実験で MERAM の有効性が確認できた．また，行列が大規模な場合，行列データの通信時間がボトルネックとなるが，Ninf-G は一度送ったデータをサーバ側で保持することができるため，2 回目以降のリスタート計算時には初期ベクトル $\boldsymbol{v}$ のみをサーバ側に送ればよく，通信コストの面で有利である．しかしながら，並列に実行する ERAM プロセスの数が増えると，それに応じて初めに行列データを送る部分のデータ転送時間が増加し，計算開始が遅くなる．本実験では，クライアント，サーバとも同一クラスタ内で実験したのでこの問題は確認できなかったが，実際の Grid 環境で実験する場合には，初めのデータ転送時間が全体の実行時間に大きな影響を及ぼすと考えられる．したがって，実験環境に応じて並列に実行させる ERAM プロセス数を考慮する必要があると予測できる．

5. おわりに

本論文では同期型 MERAM，非同期型 MERAM の性能を評価した．また，非同期型 MERAM では精度改善が停滞する場合があるという問題を示し，その回避法として修正非同期型 MERAM を提案した．同期型 MERAM は安定して解が求まる反面，収束が遅い，非同期型 MERAM は収束が速い反面，解が求まらない場合が多いという特徴があった．一方，修正非同期型 MERAM は同期型 MERAM と同程度に安定して解が求まり，非同期型 MERAM と同等の収束スピードが確認できた．ERAM でも比較的容易に解ける問題に対しては MERAM の有効性は見られなかったが，ERAM では解くことが困難な問題に対しては，同期型 MERAM，修正非同期型 MERAM は有効であった．

本論文では，クライアント，サーバともに同一クラスタ上で実験したが，GridRPC を用いて実装したため，実際の Grid 環境上での実験も可能である．Grid 環境での大規模並列演算は今後期待される技術であり，Grid 環境での MERAM の性能評価を今後の課題とする．

また本論文では，MERAM の挙動解析を容易にするため求める固有対の数を 1 組（$s = 1$）とした．$s > 1$ とした場合，$U_s^{\mathrm{best}} = (\boldsymbol{u}_1^{\mathrm{best}}, \ldots, \boldsymbol{u}_s^{\mathrm{best}})$ 内で更新される Ritz ベクトルと更新されない Ritz ベクトルが入り乱れ，残差ノルムの挙動がより複雑になると考えられる．したがって，精度が完全に停滞しない段階で U_s^{best} のとり方に工夫を加えることで収束が加速する可能性があると考えられる．この，$s > 1$ の場合の修正非同期型 MERAM の改良を今後の課題とする．さらに MERAM のアルゴリズムは，計算機の故障などにより 1 つ以上の ERAM プロセスから結果が返ってこない場合，そのプロセスを無視して計算を進めても大きな影響はないと考えらえる．したがって，MERAM の耐故障性の検証も今後の課題としてあげられる．

謝辞 本研究は（独）科学技術振興機構が行う戦略的創造事業（CREST）の研究領域「シミュレーション技術の革新と実用化基盤の構築」の研究プロジェクトの支援による．

参考文献

1) Arnoldi, W.E.: The Principle of Minimized Iteration in the Solution of Matrix Eigenvalue Problems, *Quart. Appl. Math.*, Vol.9, pp.17–29 (1951).
2) Emad, N., Shahzadeh Fazeli, S.A. and Dongarra, J.: An Asynchronous Algorithm on NetSolve Global Computing System, *Future Genetarion Computing Systems*, Vol.22, No.3, pp.279–290 (2004).
3) Emad, N., Petiton, S. and Edjlali, G.: Multiple Explicitly Restarted Arnoldi Method for Solving Large Eigenproblems, *SIAM J. Sci. Comput.*, Vol.27, pp.253–277 (2005).
4) LAPACK: Linear Algebra PACKage. http://www.netlib.org/lapack/
5) Matrix Market. http://math.nist.gov/MatrixMarket/
6) Morgan, R.B.: On restarting the Arnoldi method for large nonsymmetric eigenvalue problems, *Math. Comput.*, Vol.65, No.215. pp.1213–1230 (1996).
7) NetSolve. http://icl.cs.utk.edu/netsolve/
8) Ninf: A Global Computing Infrastructure. http://ninf.apgrid.org/
9) Phase: Parallel and High Performance Applicational Software Exchange. http://phase.hpcc.jp/
10) Saad, Y.: Variations on Arnoldi's Method for Computing Eigenelements of Large Unsymmetric Matrices, *Linear Algebra Applications*, Vol.34, pp.269–295 (1980).
11) Sorensen, D.C.: Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations, *Parallel Numerical Algorithms*, Keyes, D.E., Sameh, A. and Venkatakrishnan, V. (Eds.), pp.119–166, Dordrecht. Kluwer (1997).

12) Stewart, G.W.: A KrylovSchur Algorithm for Large Eigenproblems, *SIAM J. Matrix Anal. Appl.*, Vol.23, No.3, pp.601–614 (2001).
13) Wu, K. and Simon, H.: Thick Restart Lanczos method for Symmetric Eigenvalue Problems, *SIAM J. Matrix Anal. Appl.*, Vol.22, No.2, pp.602–616 (2000).

(平成 18 年 10 月 13 日受付)
(平成 19 年 1 月 16 日採録)

木原 崇智

2006 年筑波大学情報学類卒業．現在，筑波大学大学院博士前期課程システム情報工学研究科在学中．主に大規模固有値問題のグリッド環境向き並列解法の研究に従事．日本応用数理学会学生会員．

小瀧 義久

2005 年筑波大学情報学類卒業．現在，筑波大学大学院博士前期課程システム情報工学研究科在学中．主に大規模固有値問題のグリッド環境向き並列解法の研究に従事．日本応用数理学会学生会員．

多田野寛人

2006 年筑波大学大学院博士課程システム情報工学研究科修了．博士（工学）．現在，独立行政法人 科学技術振興機構 CREST 研究員．研究分野は大規模線形計算，主に連立一次方程式の反復解法と固有値問題の並列解法の研究に従事．日本応用数理学会会員．

櫻井 鉄也（正会員）

1986 年名古屋大学大学院工学研究科博士課程前期課程修了．同年名古屋大学工学部助手．1993 年筑波大学電子・情報工学系講師．1996 年同大学助教授．現在，筑波大学大学院システム情報工学研究科教授．東京大学，慶應義塾大学非常勤講師．独立行政法人産業技術総合研究所客員研究員．博士（工学）．大規模固有値問題の並列解法，方程式の反復解法と精度保証，数理ソフトウエアの利用支援環境の研究に従事．1996 年日本応用数理学会論文賞受賞．日本応用数理学会，日本数学会各会員．