Vol.2021-HPC-182 No.9
2021/12/7

近似逆行列前処理における前処理行列生成部の簡略化とスレッド並列化

鈴木 謙吾[1,a)]　深谷 猛[2]　岩下 武史[2]

概要：近似逆行列前処理（AINV 法）は，反復法を用いた連立一次方程式の求解における前処理手法として広く知られている．特に，GPU を用いる実装では並列性の観点から AINV 法は有用であると言える．しかし，他の前処理手法と比較し前処理行列生成に多くの時間を要し，前処理行列生成部分がソルバ全体の性能向上の妨げとなる場合がある．そこで本研究では，この前処理行列生成部分に着目し，AINV 法の簡略化とその並列化による高速化手法を提案する．SuiteSparse Matrix Collection から取得した行列に対し，提案手法では前処理行列生成部分の高速化が実現されることを示した．また，提案手法により生成した近似逆行列を CG 法の前処理行列とした場合について，反復部の計算時間を既存手法と比較し，評価した．

1. はじめに

本研究では，疎な実対称行列 $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ を係数とする連立一次方程式

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

に対する Krylov 部分空間法による求解を扱う．GPU が並列性の高い計算に対し高速に動作することから，近年では，様々な実問題における数値計算に対し GPU の利用が進み，Krylov 部分空間法による式 (1) の数値求解においても GPU が活用されている．Krylov 部分空間法は，行列ベクトル積やベクトルの演算を主とし実行されるため GPU の活用に適している．しかし，前処理手法を併用することが一般的であり，この前処理手法に対しても高い並列性が要求される．

一般に，並列性の高い前処理手法として，係数行列の逆行列 $\boldsymbol{A}^{-1}$ の近似を陽的に構成し前処理行列として用いる手法が知られている．このような手法では，ILU 分解（または，IC 分解）により $\boldsymbol{A}$ を三角行列の積 $\boldsymbol{LU}$ で近似する手法と異なり，逆行列に相当する行列が直接得られるため，前処理過程の計算が並列性の高い行列ベクトル積により実行できるという特徴がある．ILU 分解に基づく手法の多くは，前処理過程の計算として前進・後退代入計算を行う必要があり，並列性の確保に難点があるため，それらの手法と比較し，逆行列の近似に基づく手法は並列性の観点から GPU での実装に適していると考えられる．

[1] 北海道大学大学院情報科学院
[2] 北海道大学情報基盤センター
[a)] kengo.suzuki@eis.hokudai.ac.jp

しかし，逆行列の近似を陽的に構成する手法では，他の ILU 分解に基づく手法など比較し，前処理行列の生成に多くの時間を必要とする傾向にあり，ソルバ全体（本研究では，前処理行列生成部分と Krylov 部分空間法の求解部分の合計）の実行時間に占める前処理行列の生成時間の割合が大きい問題では，特に，前処理行列の生成部分の高速化が重要となる．

そこで本研究では，逆行列の近似を陽的に構成する前処理手法のうち，近似逆行列前処理（AINV 法）[1] を対象に，前処理行列生成部分の簡略化とその並列化による高速化手法を提案する．簡略化手法は AINV 法における近似逆行列生成時に $\boldsymbol{A}$-直交化を部分的に省略するという考え方に基づいており，近似逆行列生成部分の高速化が期待できる．並列化手法では，係数行列のリオーダリングを行うことで，簡略化手法での近似逆行列生成手順をスレッド並列化する．従来の AINV 法では実行手順の依存関係が複雑で並列化は容易ではないが，提案する簡略化手法を用いた場合，係数行列の非零パターンにより依存関係が決定されることとなり，係数行列のリオーダリングによって適当な非零パターンを創出することでスレッド並列化が可能となる．

本研究では，SuiteSparse Matrix Collection[2] から取得した行列を用い数値実験を行った．AINV 法と二つの提案手法それぞれでの前処理行列生成時間の評価を行い，提案手法により前処理行列生成部分の高速化が実現されることを示した．また，提案手法では，$\boldsymbol{A}$-直交化を部分的に省略したことにより，収束性に対する前処理の効果が低減すると考えられるため，本研究では，共役勾配法（CG 法）[3] の前処理として用いた場合について，提案手法の前処理と

しての性能を検証した．

本論文の構成は以下の通りである．まず，2 章で AINV 法について説明を行う．次に，3 章，4 章で提案手法である，AINV 法の簡略化手法とその並列化手法について説明する．5 章で，数値実験による結果について議論し，最後に，6 章でまとめを行う．

2. 近似逆行列前処理

2.1 前処理

Krylov 部分空間法の収束性は係数行列の固有値分布や条件数などの影響を受けることが知られ，それらの改善のために前処理手法が用いられる．一般的な前処理手法では，ある前処理行列 $\boldsymbol{M}_1$，$\boldsymbol{M}_2$ を用い，式 (1) を

$$(\boldsymbol{M}_2\boldsymbol{A}\boldsymbol{M}_1)(\boldsymbol{M}_1^{-1}\boldsymbol{x}) = (\boldsymbol{M}_2\boldsymbol{b}) \tag{2}$$

と変形し，$\boldsymbol{M}_2\boldsymbol{A}\boldsymbol{M}_1$ が単位行列に近づくことを期待する．$\boldsymbol{M}_2\boldsymbol{A}\boldsymbol{M}_1$ が単位行列であれば，$\boldsymbol{A}\boldsymbol{M}_1\boldsymbol{M}_2$ も単位行列となるため，一般に，$(\boldsymbol{M}_1\boldsymbol{M}_2)^{-1}$ が $\boldsymbol{A}$ の近似となるか，$\boldsymbol{M}_1\boldsymbol{M}_2$ が $\boldsymbol{A}^{-1}$ の近似となるような $\boldsymbol{M}_1$，$\boldsymbol{M}_2$ が用いられる．また，係数行列の対称性を維持するためには，$\boldsymbol{M}_2 = \boldsymbol{M}_1^T$ である必要がある．

2.2 A-直交化に基づく逆行列の計算

対称行列 $\boldsymbol{A}$ の逆行列 $\boldsymbol{A}^{-1}$ の計算手法として，ベクトルの $\boldsymbol{A}$-直交化に基づく手法が知られている．この手法は，互いに $\boldsymbol{A}$-直交なベクトルからなる行列 $\boldsymbol{Z} = [\boldsymbol{z}_1 \dots \boldsymbol{z}_n]$ により，$\boldsymbol{A}$ が

$$\boldsymbol{Z}^T\boldsymbol{A}\boldsymbol{Z} = \boldsymbol{D} = \mathrm{diag}(d_1, \dots, d_n) \tag{3}$$

のように対角され，$\boldsymbol{A}^{-1}$ が

$$\boldsymbol{A}^{-1} = \boldsymbol{Z}\boldsymbol{D}^{-1}\boldsymbol{Z}^T \tag{4}$$

により表現できるという事実に基づいている．具体的な実行手順は Algorithm 1 で与えられ，$\boldsymbol{Z}$ の初期値である標準基底 $(= [\boldsymbol{e}_1, \dots, \boldsymbol{e}_n])$ の各ベクトルが互いに $\boldsymbol{A}$-直交となるように順に更新していくことで，$\boldsymbol{Z}$，$\boldsymbol{D}$ が得られる．なお，この手法では $\boldsymbol{Z}$ の初期値として標準基底を採用しているため，Algorithm 1 の実行後に得られる $\boldsymbol{Z}$ は上三角行列となる．

Algorithm 1 A-直交化に基づく逆行列の計算

```
1: for i = 1, ··· , n do
2:    z_i = e_i
3:    for j = 1, ··· , i − 1 do
4:       p_j = a_i^T z_j          // a_i は A の i 番目の列
5:       z_i = z_i − (p_j / d_j) z_j
6:    end for
7:    d_i = a_i^T z_i
8: end for
```

2.3 A-直交化に基づく近似逆行列の計算

一般に，疎行列 $\boldsymbol{A}$ に対する逆行列 $\boldsymbol{A}^{-1}$ は密行列となるため，2.2 章の手法の実行には多くの時間やメモリ容量を要する．しかし，この手法を前処理行列の生成のために用いる場合，前処理行列は $\boldsymbol{A}^{-1}$ の近似であれば良く，$\boldsymbol{Z}$，$\boldsymbol{D}$ を厳密に計算する必要はない．そこで，$\boldsymbol{z}_i$ の更新時に，各要素の絶対値に対する閾値 δ を用いて要素の切り捨てを行い，$\boldsymbol{Z}$，$\boldsymbol{D}$ を近似することで計算量を削減する．$\bar{\boldsymbol{Z}}$，$\bar{\boldsymbol{D}}$ をそれぞれ，$\boldsymbol{Z}$，$\boldsymbol{D}$ の近似とすれば，$\boldsymbol{A}$ の近似逆行列 $\bar{\boldsymbol{A}}^{-1}$ は

$$\bar{\boldsymbol{A}}^{-1} = \bar{\boldsymbol{Z}}\bar{\boldsymbol{D}}^{-1}\bar{\boldsymbol{Z}}^T \approx \boldsymbol{A}^{-1} \tag{5}$$

となる．本稿では，この手法により生成される近似逆行列を前処理行列とする前処理手法を近似逆行列前処理（AINV 法）と呼ぶ．AINV 法における近似逆行列の生成は Algorithm 1 に閾値による要素の切り捨てを追加した Algorithm 2 により実行される．

Algorithm 2 AINV 法における近似逆行列の生成

```
1: for i = 1, ··· , n do
2:    z_i = e_i
3:    for j = 1, ··· , i − 1 do
4:       p_j = a_i^T z_j
5:       z_i = z_i − (p_j / d_j) z_j
6:       for k = 1, ··· , j do
7:          if |z_ki| ≤ δ then        // z_ki は z_i の k 番目の要素
8:             z_ki = 0
9:          end if
10:      end for
11:   end for
12:   d_i = a_i^T z_i
13: end for
```

2.4 前処理付き共役勾配法

本研究では，式 (1) の求解に対する Krylov 部分空間法として，$\boldsymbol{A}$ が対称な場合に最も一般的な解法である CG 法を用いる．なお，本来 CG 法は正定値な $\boldsymbol{A}$ を対象とした手法だが，$\boldsymbol{A}$ が正定値でなくとも有効となる場合があることが知られている．

前処理を併用した（式 (2) に対する）CG 法のアルゴリズムが Algorithm 3 であり，前処理過程として，$\boldsymbol{M}_1\boldsymbol{M}_2\boldsymbol{r}$ の演算が実行される．前処理として AINV 法を用いる場合この演算は行列ベクトル積となるため，

$$\boldsymbol{M}_1\boldsymbol{M}_2 = \bar{\boldsymbol{Z}}\bar{\boldsymbol{D}}^{-1}\bar{\boldsymbol{Z}}^T \tag{6}$$

とすれば良く，$\boldsymbol{M}_1\boldsymbol{M}_2$ が

$$\begin{aligned} \boldsymbol{M}_1 &= \bar{\boldsymbol{Z}}\sqrt{\bar{\boldsymbol{D}}^{-1}} \\ \boldsymbol{M}_2 &= \sqrt{\bar{\boldsymbol{D}}^{-1}}\bar{\boldsymbol{Z}}^T \end{aligned} \tag{7}$$

のように分解できる必要はない．しかし，CG 法の収束性

の改善には式 (7) の分解が可能（つまり，$\bar{\boldsymbol{D}}^{-1}$ の全要素の値が正）であることが望ましい．Algorithm 1 を厳密に計算する場合，$\boldsymbol{A}$ が正定値であれば $\boldsymbol{D}^{-1}$ の全要素が正となるため，Algorithm 2 においても $\boldsymbol{A}$ が正定値であることが重要だと考えられる．そこで本研究では，「加速係数」ξ を用い，$\boldsymbol{A}$ の対角成分のみを定数倍した行列 $\boldsymbol{A} + (\xi - 1)\mathrm{diag}(\boldsymbol{A})$（$\mathrm{diag}(\boldsymbol{A})$ は $\boldsymbol{A}$ の対角要素のみを抽出した行列）を近似逆行列生成時に限り使用するという手法を用いる．この操作により，式 (7) の分解が可能となることが期待される．

Algorithm 3 前処理付き共役勾配法

1: Compute $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$, $\boldsymbol{p_0} = \boldsymbol{M}_1\boldsymbol{M}_2\boldsymbol{r}_0$, *and* $\boldsymbol{z}_0 = \boldsymbol{p}_0$
2: **for** $l = 0, 1, \ldots,$ until $\|\boldsymbol{r}_l\|_2 / \|\boldsymbol{b}\|_2 \leq \epsilon$ is satisfied **do**
3: $\quad \alpha_l = (\boldsymbol{r}_l, \boldsymbol{r}_l)/(\boldsymbol{A}\boldsymbol{p}_l, \boldsymbol{p}_l)$
4: $\quad \boldsymbol{x}_{l+1} = \boldsymbol{x}_l + \alpha_l \boldsymbol{p}_l$
5: $\quad \boldsymbol{r}_{l+1} = \boldsymbol{r}_l - \alpha_l \boldsymbol{A}\boldsymbol{p}_l$
6: $\quad \boldsymbol{z}_{l+1} = \boldsymbol{M}_1\boldsymbol{M}_2\boldsymbol{r}_{l+1}$
7: $\quad \beta_l = (\boldsymbol{r}_{l+1}, \boldsymbol{r}_{l+1})/(\boldsymbol{r}_l, \boldsymbol{r}_l)$
8: $\quad \boldsymbol{p}_{l+1} = \boldsymbol{r}_{l+1} + \beta_l \boldsymbol{p}_l$
9: **end for**

3. 近似逆行列前処理の簡略化手法

3.1 近似逆行列前処理の実装

$\boldsymbol{A}$ は疎行列であり，$\boldsymbol{z}_i$ の更新時には要素の切り捨てを行うため，Algorithm 2 中のベクトルは全て疎ベクトルとして扱うことができる．つまり，以下の点を考慮することで Algorithm 2 の計算量を削減し効率的な実行が可能である．

① $\boldsymbol{Z}$ の各ベクトルの非零要素の値と行番号のみを保持する格納形式を用い，4，5 行目をそれぞれ，疎ベクトル同士の内積，減算を行う実装が可能．

② 疎ベクトル同士の内積の値は 0 となることが多いため，事前に $\boldsymbol{z}_j$ が $\boldsymbol{a}_i$ と同じ行番号に非零要素を持つかどうかの判定を行い，$p_j = 0$ となることが分かる j に対しては 4，5 行目の計算を省略可能．

しかし一般に，②における疎ベクトル同士が同じ行番号に非零要素を持つかどうかの判定は容易ではなく，効率的な実装には，別途 $\boldsymbol{Z}$ を行方向に格納した配列等が必要となる．加えて，更新により $\boldsymbol{z}_i$ の要素が増加することへの対応や，複数の j に対して判定を行うために，ソートや重複削除が必要となるなどの問題点があり，AINV 法の高速実行には，②の $p_j = 0$ となる j の判定の効率化が要求される．

3.2 近似逆行列前処理の簡略化

そこで本研究では，$\boldsymbol{z}_j$ に非零要素の増加が起きていないと仮定することで，②の $p_j = 0$ となる j の判定を簡略化する手法（以下，簡略化手法と呼ぶ）を提案する．簡略化手法では，$\boldsymbol{z}_j$ に非零要素の増加が起きていないという仮定により，$\boldsymbol{z}_j$ は初期値である $\boldsymbol{e}_j$ とみなされる．したがって，$p_j = 0$ となる j の判定として，$\boldsymbol{a}_i$ と $\boldsymbol{e}_j$ が同じ行に非零要素を持つかどうかの判定を行うことになり，結果として，$\boldsymbol{a}_i$ の非零要素の行番号に対応する j を選択することになる．簡略化手法の実行手順は Algorithm 4 のようになり，$\boldsymbol{a}_i$ と $\boldsymbol{z}_j$ の内積の計算部分に $a_{ji} \neq 0$ かどうかの条件が追加される．

簡略化手法では，②の判定時に問題であった $\boldsymbol{z}_i$ の要素の増加への対応が省略され，ソート，重複削除の必要もなくなるため，高速化が期待できる．一方で，$\boldsymbol{Z}$ の各ベクトルの $\boldsymbol{A}$-直交化が部分的に省略されるため，前処理行列としての収束性改善への効果が低下することが予想される．つまり，前処理行列生成部分で削減される時間が，CG 法の求解部分で増加する時間を上回れば，全体として簡略化手法による高速化が期待できる．

Algorithm 4 簡略化手法

1: **for** $i = 1, \cdots, n$ **do**
2: $\quad \boldsymbol{z}_i = \boldsymbol{e}_i$
3: $\quad$ **for** $j = 1, \cdots, i-1$ **do**
4: $\quad\quad$ **if** $a_{ji} \neq 0$ **then** $\quad$ // a_{ji} は $\boldsymbol{a}_i$ の j 番目の要素
5: $\quad\quad\quad p_j = \boldsymbol{a}_i^T \boldsymbol{z}_j$
6: $\quad\quad\quad \boldsymbol{z}_i = \boldsymbol{z}_i - \frac{p_j}{d_j}\boldsymbol{z}_j$
7: $\quad\quad$ **end if**
8: $\quad\quad$ **for** $k = 1, \cdots, j$ **do**
9: $\quad\quad\quad$ **if** $|z_{ki}| \leq \delta$ **then**
10: $\quad\quad\quad\quad z_{ki} = 0$
11: $\quad\quad\quad$ **end if**
12: $\quad\quad$ **end for**
13: $\quad$ **end for**
14: $\quad d_i = \boldsymbol{a}_i^T \boldsymbol{z}_i$
15: **end for**

4. 近似逆行列前処理のスレッド並列化手法

本研究では，3 章にて説明した AINV 法の簡略化手法の提案に加え，簡略化手法における前処理行列生成部分に対するスレッド並列化手法（以下，並列化手法と呼ぶ）を提案する．式 (1) に対し，代数的マルチカラーオーダリング法（AMC 法）[4] を用いることで，簡略化手法における前処理行列生成のスレッド並列化を実現する．

4.1 代数的マルチカラーオーダリング法

4.1.1 オーダリング

オーダリング手法とは，係数行列の並び替えにより，前処理行列の生成や前処理過程の実行に適した非零パターンを得る手法である．まず，ある置換行列 $\boldsymbol{P}$ を用い式 (1) を

$$(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^T)(\boldsymbol{P}\boldsymbol{x}) = (\boldsymbol{P}\boldsymbol{b}) \tag{8}$$

と変形する．なお，$\boldsymbol{P}$ は直交行列であり $\boldsymbol{P}^T = \boldsymbol{P}^{-1}$ である．次に，新たな変数

$$\boldsymbol{A}' := \boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^T, \quad \boldsymbol{x}' := \boldsymbol{P}\boldsymbol{x}, \quad \boldsymbol{b}' := \boldsymbol{P}\boldsymbol{b} \tag{9}$$

を定義すれば，式 (8) は

$$\boldsymbol{A}'\boldsymbol{x}' = \boldsymbol{b}' \tag{10}$$

と表現できる．ここで，$\boldsymbol{A}'$ に何らかの意味で前処理に適した非零パターンが得られれば，効率的な前処理の実行が期待できる．

オーダリング手法を用いる場合，変形後の式 (10) を求解の対象とする．しかしこの時，式 (1) を直接求解する場合と比べ，IC 分解に基づく前処理を用いる場合に代表されるように，前処理付き反復法の収束性が変化することがあるため，留意が必要である [5]．

4.1.2 代数的マルチカラーオーダリング

AMC 法では，$\boldsymbol{A}$ の非零パターンに基づく未知数 $\boldsymbol{x}$ のカラーリングにより $\boldsymbol{P}$ が構成され，$\boldsymbol{A}'$ は次の条件を満たす．

条件

$\boldsymbol{A}'$ は，全ての対角ブロックが対角行列からなるブロック分割をもち，その次数は各色に属す未知数の数とそれぞれ等しい．

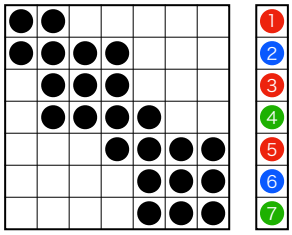
具体的には，各未知数を頂点，$\boldsymbol{A}$ を隣接行列とみなし，$\boldsymbol{A}$ において隣接のない未知数を同じ色でカラーリングし，得られたカラーリング後の未知数を基に，$\boldsymbol{x}'$ では同じ色の未知数がまとまって並ぶように $\boldsymbol{P}$ を構成する．本研究では，カラーリング手法として貪欲法（一つの色で可能な限り多くの未知数を色付けしていく手法）を用い，各色ごとに未知数がオーダリング前の行番号において昇順に並ぶように $\boldsymbol{P}$ を構成する．

4.2 前処理行列生成部分のスレッド並列化

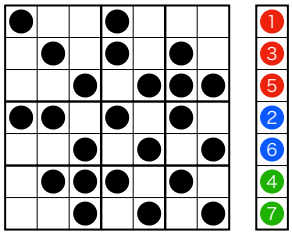
式 (1) に対し，AMC 法を適用することで得られる係数行列 $\boldsymbol{A}'(=(a'_{ij})_{n\times n,(1\le i,j\le n)})$ は 4.1.2 章に示す条件を満たすため，対角行列からなる対角ブロックを持つ．つまり，c 番目の色に属する n_c 個の未知数の番号の集合を $\mathbb{I}^{(c)} = \{c', c'+1, \ldots, c'+n_c-1\}$ とすれば，任意の $i', j' \in \mathbb{I}^{(c)}$ $(i' \neq j')$ に対し $a'_{i'j'} = 0$ が成り立つ．この時，Algorithm 4 では，簡略化手法において追加した 4 行目の条件判定を行うことにより，$\boldsymbol{z}_{i'}$ と $\boldsymbol{z}_{j'}$ の更新（$i = i', j'$ での計算）は互いに影響を及ぼさず，それぞれ並列に実行可能である．したがって，AMC 法の適用により，Algorithm 4 は色ごとに $i = c', c'+1, \ldots, c'+n_c-1$ に対する $\boldsymbol{z}_i$ の更新を全て並列に実行可能となる．

図 1 に，あるサンプル行列に対する AMC 法でのオーダリング結果の例を示す．この $\boldsymbol{A}'$ の非零パターンの例では，並び替え後の未知数の番号に対し，$\mathbb{I}^{(1)} = \{1,2,3\}$, $\mathbb{I}^{(2)} = \{4,5\}$, $\mathbb{I}^{(3)} = \{6,7\}$ であり，Algorithm 4 の $i = 1,2,3$，$i = 4,5$，$i = 6,7$ のそれぞれにおいて並列実行が可能である．

近似逆行列を前処理行列として用いる Krylov 部分空間法では，前処理過程が行列ベクトル積で与えられるため，



(a) $\boldsymbol{A}$ の非零パターンと未知数のカラーリング例



(b) $\boldsymbol{A}'$ の非零パターンと未知数の並び替え

図 1: 代数的マルチカラーオーダリング法によるオーダリングの例．●が非零要素を表す．

$\boldsymbol{A}'$ に対する近似逆行列

$$\bar{\boldsymbol{A}'}^{-1} = \bar{\boldsymbol{Z}}'\bar{\boldsymbol{D}}'^{-1}\bar{\boldsymbol{Z}}'^{T} \tag{11}$$

を生成した後，再度並び替えを行っても，（丸め誤差の影響の変化を除き）Krylov 部分空間法の収束性は変化しない．つまり，新たな置換行列 $\boldsymbol{Q}$ を用い，式 (10) を

$$(\boldsymbol{Q}\boldsymbol{A}'\boldsymbol{Q}^T)(\boldsymbol{Q}\boldsymbol{x}') = (\boldsymbol{Q}\boldsymbol{b}') \tag{12}$$

と変形しても良い．この時，新たな係数行列 $\boldsymbol{Q}\boldsymbol{A}'\boldsymbol{Q}^T$ の逆行列は

$$(\boldsymbol{Q}\boldsymbol{A}'\boldsymbol{Q}^T)^{-1} \approx \boldsymbol{Q}\bar{\boldsymbol{A}'}^{-1}\boldsymbol{Q}^T \tag{13}$$

と近似できる．

事前調査により，行列格納形式として Sliced ELLPACK (SELL-C, C は slice サイズ) 形式 [6] を用い，GPU 上で疎行列ベクトル積（SpMV）を行う場合，$\boldsymbol{A}$ に対する SpMV と比較し，$\boldsymbol{A}'$ に対する SpMV は計算時間が増加する傾向にあることが分かった．そこで本研究では，$\boldsymbol{Q}$ として $\boldsymbol{P}^T$ を用い，式 (12) を式 (1) で表現し直し，式 (1) を求解の対象とする．この時，$\boldsymbol{A}^{-1}$ に関して，式 (9)，(11)，(13) から

$$\begin{aligned}\boldsymbol{A}^{-1} &= (\boldsymbol{P}^T\boldsymbol{A}'\boldsymbol{P})^{-1} \\ &\approx \boldsymbol{P}^T\bar{\boldsymbol{A}'}^{-1}\boldsymbol{P} \\ &= \boldsymbol{P}^T\bar{\boldsymbol{Z}}'\bar{\boldsymbol{D}}'^{-1}\bar{\boldsymbol{Z}}'^{T}\boldsymbol{P}\end{aligned} \tag{14}$$

が成り立つため，

$$\bar{\boldsymbol{Z}} = \boldsymbol{P}^T\bar{\boldsymbol{Z}}' \tag{15}$$

$$\bar{\boldsymbol{D}}^{-1} = \bar{\boldsymbol{D}}'^{-1} \tag{16}$$

とすることで，$\boldsymbol{A}$ に対する近似逆行列 $\bar{\boldsymbol{Z}}\bar{\boldsymbol{D}}^{-1}\bar{\boldsymbol{Z}}^T$ が得られる．

表 1: 実験に用いた計算ノード（Fujitsu CX2570 M5）の仕様

CPU	Intel Xeon Gold 6230 (Cascade Lake) × 2 論理コア数：40
メモリ	DDR4-2933 (32GB×12 = 384GB)
コンパイラ オプション	gcc (Red Hat 4.8.5-36) -O3 -fopenmp
GPU	NVIDIA Tesla V100
コンパイラ オプション	nvcc (V10.1.243) -lcublas -Xcompiler -fopenmp -lgomp

5. 数値実験

5.1 実験環境と問題設定

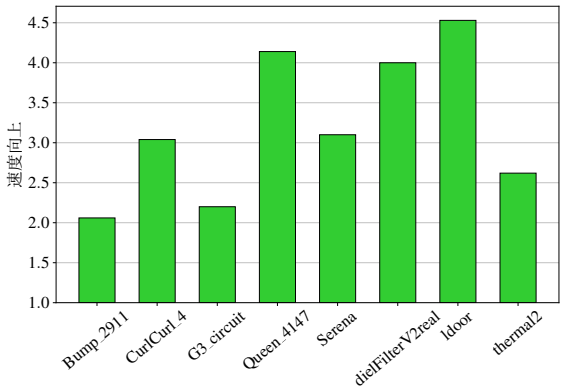
本研究では，北海道大学情報基盤センターの計算ノード Fujitsu CX2570 M5（CPU：Intel Xeon Cascade Lake, GPU：NVIDIA Tesla V100）を用い数値実験を行った．ノードの仕様や，コンパイラの情報などは表 1 に示す通りである．プログラムは C と CUDA により記述し，倍精度浮動小数点演算を用いた．また，CPU 実行部分のスレッド並列化には OpenMP を用い，GPU 上での SpMV には行列格納形式として SELL-32 形式を用いた実装を行った．なお，その他のベクトル演算には cuBLAS の関数を用いた．

求解対象には, 表 2 に示す SuiteSparse Matrix Collection から取得した行列を係数行列とし，全要素が 1 のベクトルを右辺ベクトルとする連立一次方程式を用いた．Krylov 部分空間法には前処理付き CG 法を用い，CG 法の反復計算部分は GPU 上で実行した．前処理行列生成部分は CPU 上で実行することとし，AINV 法，簡略化手法は逐次実行，並列化手法は 40 スレッドを使用し実行した．また，CG 法の初期近似解はゼロベクトルとし，収束判定には相対残差ノルム（$\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2 / \|\boldsymbol{b}\|_2$）を用い，相対残差ノルムが 10^{-8} より小さくなることを収束の条件とした．

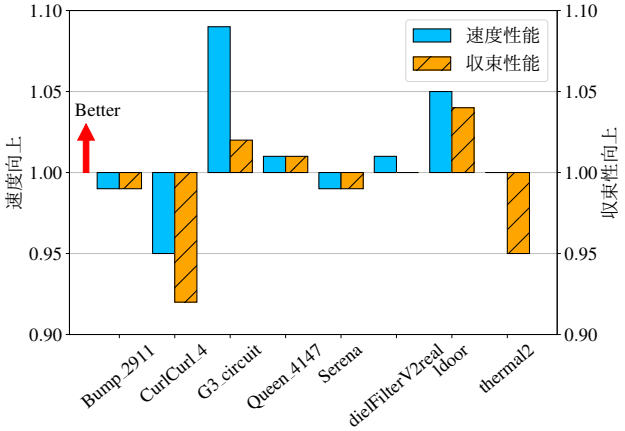
AINV 法及び，簡略化手法，並列化手法における $\boldsymbol{z}_i$ 更新時の閾値 δ は，0.08，0.09，0.1，0.11，0.12 とし，それぞれの値に対し実験を行った．また，「加速係数」ξ には 1.0, 1.1，1.2，1.3 から従来の AINV 法の結果が最善となる値を選択した．各行列に対する「加速係数」は表 2 に示す通りである．

5.2 実験結果

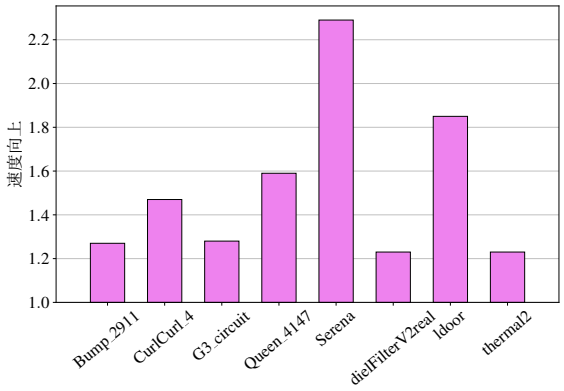
表 3–5 に，各テスト行列に対する，AINV 法，簡略化手法，並列化手法のそれぞれでの前処理生成部分の実行時間（表 3），CG 法の前処理として用いた場合の反復回数（表 4），計算時間（表 5）を示す．以下，表 3–5 の結果を基に，5.2.1 章，5.2.2 章において，簡略化手法と並列化手法のそれぞれについて性能評価を行う．閾値 δ の変化に対する各手法の傾向は同様であるため，以下の比較では，実



(a) 前処理行列生成部分の実行時間比較



(b) CG 法の前処理とした場合の反復回数と計算時間の比較



(c) 前処理行列生成時間と CG 法の計算時間の合計の比較

図 2: $\delta = 0.1$ とした場合の AINV 法と簡略化手法の比較

用において一般的な設定である $\delta = 0.1$ での結果を扱うが，他の δ の値に対しても同様のことが言える．

5.2.1 簡略化手法の性能評価

図 2(a) に，閾値 $\delta = 1$ での AINV 法と簡略化手法の前処理行列生成部分に対する実行時間比を示す．全てのテスト行列に対し，簡略化手法は AINV 法と比較し 2 倍を超える速度向上を達成している．特に，Queen_4147, dielFilterV2real，ldoor に対する速度向上は著しく，4 倍以上の速度向上を実現した．

図 2(b) に，AINV 法と簡略化手法のそれぞれを CG 法

表 2: 実験に用いたテスト行列の性質．SuiteSparse Matrix Collection から取得した．

行列	次元数	非零要素数	非零要素数/次元数	問題の分野	加速係数
Bump_2911	2,911,419	127,729,899	43.87	2D/3D Problem	1.2
CurlCurl_4	2,380,515	26,515,867	11.14	Model Reduction Problem	1.2
G3_circuit	1,585,478	7,660,826	4.83	Circuit Simulation Problem	1.0
Queen_4147	4,147,110	316,548,962	76.33	2D/3D Problem	1.3
Serena	1,391,349	64,131,971	46.09	Structural Problem	1.2
dielFilterV2real	1,157,456	48,538,952	41.94	Electromagnetics Problem	1.2
ldoor	952,203	42,493,817	44.63	Structural Problem	1.3
thermal2	1,228,045	8,580,313	6.99	Thermal Problem	1.0

表 3: 提案手法と既存手法での近似逆行列生成時間 [s] の比較．a) が AINV 法，b) が簡略化手法，c) が並列化手法を表す．

行列		閾値 δ				
		0.08	0.09	0.1	0.11	0.12
Bump_2911	a)	9.74	8.73	7.96	7.32	6.88
	b)	4.41	4.30	3.85	3.82	3.66
	c)	0.81	0.78	0.79	0.84	0.78
CurlCurl_4	a)	3.99	3.47	2.99	2.57	2.42
	b)	1.13	1.03	0.98	0.88	0.86
	c)	1.01	0.78	0.70	0.58	0.49
G3_circuit	a)	1.06	0.86	0.74	0.69	0.65
	b)	0.37	0.34	0.33	0.34	0.33
	c)	0.29	0.23	0.22	0.20	0.19
Queen_4147	a)	56.55	50.32	47.93	45.01	41.72
	b)	12.83	12.21	11.55	10.62	10.16
	c)	2.28	1.96	1.87	1.80	1.75
Serena	a)	8.84	7.71	6.95	6.16	5.74
	b)	2.80	2.52	2.23	2.19	2.10
	c)	0.49	0.39	0.38	0.38	0.40
dielFilterV2real	a)	10.70	9.48	8.62	7.98	7.26
	b)	2.29	2.18	2.15	2.00	1.92
	c)	0.70	0.61	0.56	0.55	0.52
ldoor	a)	10.31	9.06	7.72	7.15	6.25
	b)	2.17	1.95	1.70	1.57	1.52
	c)	0.65	0.49	0.40	0.33	0.31
thermal2	a)	1.47	1.38	1.22	1.19	1.10
	b)	0.50	0.50	0.46	0.43	0.44
	c)	0.26	0.20	0.17	0.15	0.15

表 4: 提案手法と既存手法を CG 法の前処理として用いた場合の反復回数の比較．a) が AINV 法，b) が提案手法 1，c) が提案手法 2 を表す．

行列		閾値 δ				
		0.08	0.09	0.1	0.11	0.12
Bump_2911	a)	2690	2834	2921	3001	3031
	b)	2685	2858	2944	3003	3031
	c)	3094	3193	3284	3386	3461
CurlCurl_4	a)	1244	1330	1331	1469	2239
	b)	1352	1365	1434	1521	1571
	c)	1545	1555	1528	1628	1670
G3_circuit	a)	1050	1303	1486	1480	1455
	b)	1258	1423	1446	1478	1489
	c)	1714	1744	1759	1765	1765
Queen_4147	a)	6626	6785	6879	6807	6935
	b)	6548	6701	6769	6716	6811
	c)	6332	6593	7060	7586	9144
Serena	a)	517	534	556	578	599
	b)	517	537	558	581	604
	c)	500	533	549	565	1181
dielFilterV2real	a)	11443	11801	12013	12515	13230
	b)	11588	11961	11980	12921	13258
	c)	13766	14443	14581	14978	15633
ldoor	a)	3426	3532	3682	3769	3833
	b)	3443	3354	3521	3559	3642
	c)	3329	3559	3678	3732	3909
thermal2	a)	2351	2505	2616	2716	2829
	b)	2587	2691	2746	2832	2920
	c)	2648	2701	2758	2789	2774

の前処理として用いた場合の反復回数と計算時間の比を示す．G3_circuit や dielFilterV2real などに対しては，簡略化手法により収束性がより改善し，CG 法の計算時間においても短縮されている．一方，CurlCurl_4，thermal2 などでは，AINV 法と比較し CG 法の収束性が悪化し，計算時間に関しても増加した．しかし，最も計算時間が増加した CurlCurl_4 においても，その増加率は 5%程度であった．

提案手法により CG 法の計算時間が増加する場合においても，前処理行列生成部分で削減される時間が，CG 法において増加する計算時間を上回れば，提案手法によるソルバ全体の高速化が達成される．そこで，対象とする問題により，前処理生成部分にかかる時間（T_{pre}）と，CG 法での求解にかかる時間（T_{cg}）の比が異なることを考慮し，T_{pre} と T_{cg} の比に基づくソルバ全体の計算時間に対する提案手法の評価式を導出する．提案手法による，前処理行列生成部分の速度向上の逆数を α，CG 法の反復計算部分の速度向上の逆数を β で表せば，

$$\alpha T_{pre} + \beta T_{cg} < T_{pre} + T_{cg} \qquad (17)$$

表 5: 提案手法と既存手法を CG 法の前処理として用いた場合の反復部分の計算時間比較．a) が AINV 法，b) が提案手法 1，c) が提案手法 2 を表す．

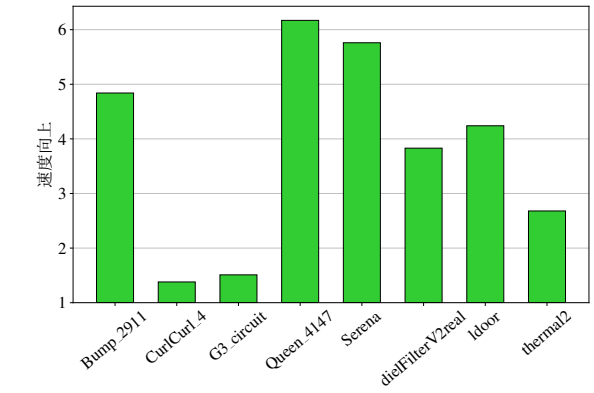
行列		閾値 δ				
		0.08	0.09	0.1	0.11	0.12
Bump_2911	a)	10.16	10.50	10.67	10.83	10.84
	b)	10.20	10.58	10.77	10.85	10.86
	c)	13.98	14.11	14.27	14.45	14.63
CurlCurl_4	a)	2.78	2.84	2.76	2.98	4.30
	b)	2.91	2.85	2.91	2.99	3.03
	c)	3.72	3.68	3.65	3.70	3.70
G3_circuit	a)	1.48	1.64	1.78	1.70	1.65
	b)	1.55	1.63	1.63	1.64	1.61
	c)	2.20	2.19	2.17	2.14	2.19
Queen_4147	a)	50.94	51.41	51.65	50.56	51.09
	b)	50.08	50.61	50.71	49.87	50.06
	c)	59.65	60.90	64.11	68.10	80.60
Serena	a)	1.37	1.39	1.41	1.44	1.47
	b)	1.37	1.38	1.41	1.44	1.47
	c)	1.85	1.92	1.90	1.91	3.24
dielFilterV2real	a)	25.98	26.49	26.62	27.38	28.65
	b)	26.15	26.57	26.30	28.13	28.61
	c)	35.95	37.19	36.87	37.37	38.44
ldoor	a)	5.88	5.90	6.01	6.01	5.98
	b)	6.17	5.57	5.71	5.65	5.69
	c)	6.58	6.73	6.75	6.71	6.95
thermal2	a)	2.67	2.74	2.77	2.77	2.81
	b)	2.76	2.78	2.77	2.78	2.82
	c)	3.24	3.24	3.19	3.26	3.16

が成立するとき，提案手法によりプログラム全体の実行時間が短縮されると考えられる．つまり，T_{pre} と T_{cg} の比として γ を考え，$T_{pre} = \gamma T_{cg}$ とすれば，次の不等式

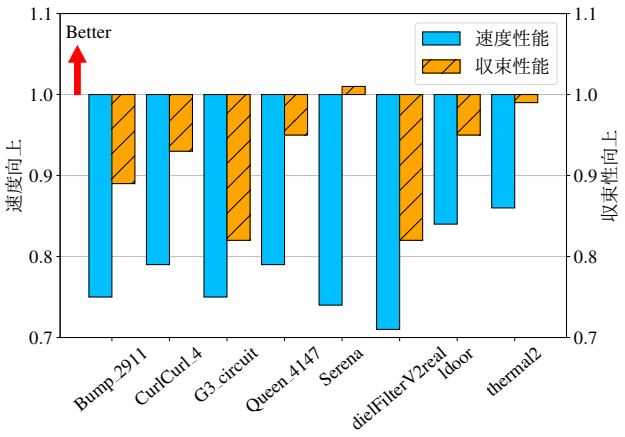
$$\begin{aligned} \alpha T_{pre} + \beta\gamma T_{pre} &< T_{pre} + \gamma T_{pre} \\ \alpha + \beta\gamma &< 1 + \gamma \\ \gamma &< \frac{1-\alpha}{\beta - 1} \end{aligned} \tag{18}$$

を満たす問題に対し，提案手法は有効だと言える．$\delta = 1$ での実験結果のうち，最も簡略化手法の効果が小さかった場合として，$\alpha = 1/2$，$\beta = 1/0.95$ とすれば，式 (18) から $\gamma < 9.5$ となる．つまり，少なく見積もった場合でも，既存の AINV 法において T_{cg} が T_{pre} の 9.5 倍程度までの問題に対し，簡略化手法は有効であると考えられる．

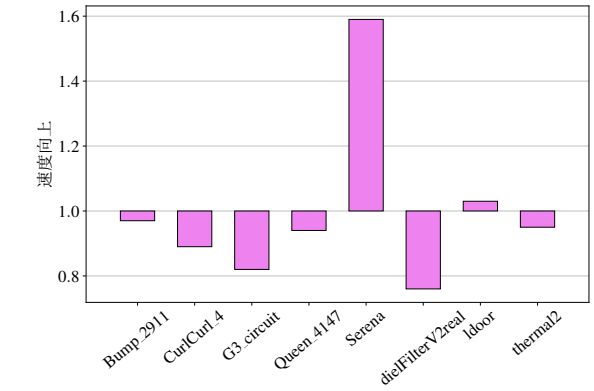
本数値実験における問題設定では，全ての行列に対し $\gamma < 9.5$ が成立しているため，簡略化手法の適用により，従来の AINV 法を用いる場合と比較し，ソルバ全体の実行時間においても短縮すると予想できる．実際に，ソルバ全体の実行時間に対し AINV 法と簡略化手法の比較を行ったものが図 2(c) であり，簡略化手法により全ての行列に対しソルバ全体の速度性能が向上していることが確認できる．



(a) 前処理行列生成部分の実行時間比較



(b) CG 法の前処理とした場合の反復回数と計算時間の比較



(c) 前処理行列生成時間と CG 法の計算時間の合計の比較

図 3: $\delta = 0.1$ とした場合の簡略化手法と並列化手法の比較

5.2.2 並列化手法の性能評価

次に，並列化手法の評価を行う．前項の AINV 法と簡略化手法の比較と同様に，閾値 $\delta = 1$ での結果を基に評価する．

図 3(a) に，簡略化手法と並列化手法の前処理行列生成部分に対する実行時間比を示す．前処理行列生成部分に関しては，並列化手法において全ての行列に対し速度向上が確認できる．Queen_4147, Serena に対する実験において性能向上が著しく，6 倍程度の速度向上を実現した．並列化手法による AINV 法からの速度向上は図 2(a) と図 3(a) の値の

積に等しく，全ての行列に対し，少なくとも 3 倍を超える速度向上が見られる．特に性能向上の著しい Queen_4147 では 25 倍を超える速度向上を達成した．

図 3(b) に，簡略化手法と並列化手法を CG 法の前処理として用いた場合の CG 法の反復回数比と計算時間比を示す．CG 法の前処理として用いた場合の反復回数，計算時間に関しては，並列化手法を用いたことで，ほぼ全ての行列に対し性能悪化が確認された．これは，リオーダリングにより，逆行列に対するより粗い近似が行われ前処理行列の収束性改善への効果が低下したことと，前処理過程の行列ベクトル積におけるベクトルへのメモリアクセスパターンが悪化したことが主な原因と考えられる．

並列化手法においても，式 (18) による評価を考える．並列化手法では，CurlCurl_4，G3_circuit のように，前処理行列生成部分の速度向上が小さい行列と（$\alpha \approx 1/1.5$），Queen_4147，Serena のように速度向上が大きい行列（$\alpha \approx 1/6.0$）がある．CG 法の収束までの計算時間に関しては平均として 1.3 倍程度増加するため（$\beta = 1.3$），$\alpha = 1/1.5$ の場合 $\gamma < 1.1$，$\alpha = 1/6.0$ の場合 $\gamma < 2.8$ となる．つまり，簡略手法を用いた場合の実行時間において，並列化の効果の小さいもので T_{cg} が T_{pre} の 1.1 倍程度，並列化効果の大きいもので T_{cg} が T_{pre} の 2.8 倍程度までの問題に対し，並列化手法は有効であると考えられる．

図 3(c) にソルバ全体の実行時間に対する簡略化手法と並列化手法の比較を示す．簡略化手法との比較においては，Serena が式 (18) に示す γ の条件を満たす行列であり，実際に，ソルバ全体の実行時間において，並列化手法による簡略化手法からの速度向上が見られる．また，本数値実験の問題設定では，並列化手法による CG 法の計算時間の悪化が比較的に小さく抑えられた ldoor に対しても，1.03 倍程度と僅かではあるがソルバ全体の速度性能が向上した．この二つの行列に対しては，AINV 法を用いた場合との比較では，並列化手法の適用により Serena では 3.6 倍，ldoor では 1.9 倍程度のソルバ全体の速度向上を実現した．なお，残り六つの行列に対しては簡略化手法からの性能低下が見られるが，AINV 法との比較では，dielFilterV2real を除く五つの行列に対して速度向上を達成している．

6. おわりに

本研究では，AINV 法の前処理行列生成部分を高速化する手法として，AINV 法の近似逆行列生成の手順を簡略化する手法（簡略化手法）と，係数行列のリオーダリングにより簡略化手法をスレッド並列化する手法（並列化手法）を提案した．CPU として Intel Xeon (Cascade lake)，GPU として NVIDIA Tesla V100 を搭載するノードを使用し，前処理行列生成部分は CPU で実行し，CG 法の反復計算部分は GPU で実行するという条件で，SuiteSparse Matrix Collection から取得した八つのテスト行列を用い数値実験を行った．

まず，前処理行列生成部分の実行時間について，簡略化手法は従来の AINV 法から 2.0–4.5 倍程度，並列化手法は簡略化手法から 1.5–6.0 倍程度の高速化をそれぞれ実現した．

次に，提案手法を CG 法の前処理として用いた場合の性能について，簡略化手法では AINV 法と比較し，四つの行列に対し CG 法の計算時間にも速度向上が見られ，計算時間が増加した残りの四つの行列に対してもその増加率は 5%程度に抑えられた．一方，並列化手法では，係数行列のリオーダリングの影響により収束性や行列ベクトル積演算でのメモリアクセスパターンが悪化し，簡略化手法と比較し CG 法の計算時間が 30%程度増加することが分かった．

最後に，対象とする問題により，前処理生成部分にかかる時間（T_{pre}）と，CG 法での求解にかかる時間（T_{cg}）の比（$\gamma := T_{cg}/T_{pre}$）が異なることから，γ に基づく提案手法の性能評価を行った．実験結果から，簡略化手法では AINV 法を用いた場合に $\gamma < 9.5$，並列化手法では簡略化手法を用いた場合に $\gamma < 1.1$–2.8 程度である問題に対し有効であると考えられる．なお，本研究における問題設定では，ソルバ全体の速度性能において，簡略化手法は八つ全てのテスト行列に対し AINV 法からの性能向上を達成し，並列化手法は二つのテスト行列に対し簡略化手法からの性能向上を実現した．

今後は，CG 法の収束性が悪化した並列化手法に対し，他のオーダリング手法適用するなどのアプローチにより性能の改善を試みたい．また，提案手法を行列が非対称な場合にも拡張し，Bi-CGSTAB 法などの非対称行列に対する Krylov 部分空間法との併用とその性能評価を行いたい．

参考文献

[1] Benzi, M., Meyer, C. D. and Tůma, M.: A Sparse Approximate Inverse Preconditioner for the Conjugate Gradient Method, *SIAM Journal on Scientific Computing*, Vol. 17, No. 5, pp. 1135–1149 (1996).
[2] Davis, T. A. and Hu, Y.: The University of Florida sparse matrix collection, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 38, No. 1, pp. 1–25 (2011).
[3] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2nd edition (2003).
[4] Iwashita, T. and Shimasaki, M.: Algebraic multicolor ordering for parallelized ICCG solver in finite-element analyses, *IEEE Transactions on Magnetics*, Vol. 38, No. 2, pp. 429–432 (2002).
[5] Iwashita, T., Li, S. and Fukaya, T.: Hierarchical block multi-color ordering: A new parallel ordering method for vectorization and parallelization of the sparse triangular solver in the ICCG method, *CCF Transactions on High Performance Computing*, Vol. 2, No. 2, pp. 84–97 (2020).
[6] Monakov, A., Lokhmotov, A. and Avetisyan, A.: Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures, *High Performance Embedded Architectures and Compilers*, Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 111–125 (2010).