計算工学講演会論文集 Vol. 26 (2021 年 5 月)
計算工学会

並列有限要素法プログラムへの可変前処理付きクリロフ部分空間法の実装と性能評価

Implementation of Variable Preconditioned Krylov Iterative Solver on an OSS Parallel FEM Program

奥田洋司 [1)] 櫛田慶幸 [2)]
Hiroshi Okuda and Noriyuki Kushida

[1)]博 (工) 東京大学 教授（〒 277-8563 千葉県柏市柏の葉 5-1-5, E-mail: okuda@k.u-tokyo.ac.jp)
[2)]博 (工) 東京大学 客員共同研究員（〒 277-8563 千葉県柏市柏の葉 5-1-5, E-mail: nkushi@multi.k.u-tokyo.ac.jp）

This paper presents the implementation and performance evaluation of the Self-Updating Preconditioned Generalized Minimum Residual Recursive (SUP-GMRESR) method in the parallel finite element method (FEM) program called FrontISTR. The SUP algorithm is an algorithm that updates preconditioning matrices so that the preconditioners strengthen their effect. The implementation in FrontISTR is discussed, and its performance is evaluated using realistic engineering problems. The results demonstrate that SUP-GMRESR achieves significantly faster convergence with fewer iterations compared to GMRES and GMRESR. In the best case scenario, the convergence speed is 4 times faster than the original GMRES. The computational time of SUP-GMRESR is also reduced as the additional computational effort with regard to the SUP algorithm is negligible, whereas the number of iterations to convergence is significantly fewer. As a result, SUP-GMRESR proves to be a viable solution for large-scale FEM simulations. The implemented code is available in the FrontISTR Gitlab repository, and the data used in the study can be obtained from the FrontISTR Commons Data Reservoir website.

Key Words : *Self-Updating Preconditioning, GMRESR, flexible Krylov method, parallel FEM, FrontISTR*

1. はじめに

数値的に偏微分方程式を解く場合、例えば有限要素法（FEM）では線形方程式ソルバーが計算時間の大半を占める。計算を高速化するために、多くの線形方程式解法アルゴリズムが開発されてきた。Krylov 型線形方程式ソルバーは、現在の高性能計算（HPC）分野におけるトレンドである分散メモリ型の計算機に適しているため主流となっている。Krylov 型線形方程式ソルバーは、収束を向上させ、計算性能を改善するために、しばしば前処理技術と共に用いられる。前処理技術は、元の線形方程式：

$$Ax = b, \tag{1}$$

を以下のように変換することでより良い収束を達成する [1]：

$$M_1^{-1}AM_2^{-1}(M_2x) = M^{-1}1b, \tag{2}$$

ここで、$M = M_1M_2$ は前処理行列を示す。変換後の行列 $M^{-1}1AM_2^{-1}$ は A よりも条件数が低くなることが期待される。実際のアルゴリズムでは、計算コストが許容できないほど大きくなるため、上記の変換は通常行わない。代わりに、Krylov 型ソルバーの各反復で前処理に関わるあらたな方程式、

$$Mz = p \tag{3}$$

が解かれる。ここで、p と z はそれぞれ元の系と変換された系における探索ベクトルである。

以上の数学的背景により、M は Krylov ソルバーの反復を通じて不変であることが期待される。いくつかの Krylov ソルバーアルゴリズムは、より良い収束を達成するために可変的な M を受け入れる。これらのアルゴリズムのほとんどは、比較的ゆるやかな収束基準を用いて方程式を反復的に解くものである [2] [3]。そのようなアルゴリズムの中でも、前処理行列を更新する方法と組み合わせることができる再帰的一般化最小残差法（GMRESR）は独創的な手法である [4]。Van der Vorst と Vuik は、Eirola と Nevanlinna のアルゴリズム [5] を前処理行列の更新に適用し（この組み合わせたアルゴリズムは GMRESR-EN と呼ばれる）、Kushida と Okuda は Broyden–Fletcher–Goldfarb–Shanno スキーム（BFGS）を適用した自己更新型前処理 SUP-GMRESR を開発し、元の GMRESR および GMRESR-EN よりも良い収束を示すことを確認した [6] [7]。

このような新規アルゴリズムは、非常に良い性能であることが示されている場合であったとしても、既存プログラムに取り入れられるまでには時間がかかることが多い。FrontISTR は、接触や材料非線形性などの複雑な現象を分析する能力と並列性能のために並列 FEM 構造解析プログラムとして好評を得ているうえ、オープンソースプロジェクトである。本研究では、SUP-GMRESR を FrontISTR に実装し、自由度が 600×10^6 に達する現実的な問題を用いてその性能を測定した。

2. アルゴリズム

SUP-GMRESR のアルゴリズムは先行研究で詳述されているため、本節ではアルゴリズムを掲載するに留める。下記に示すアルゴリズムは SUP-GMRESR の完全なアルゴリズムである。アルゴリズムは 2 つの主要な部分から構成されている：(1) GMRESR 部分と (2) 前処理行列を更新し、更新された前処理行列の系を解くためのメモリ制限型 BFGS（Limited memory BFGS: L-BFGS）部分である。行 1 から 31 までが GMRESR に相当する。この部分は、以下の行を除いて、元の GMRESR アルゴリズムとほぼ同等である。

- 10 行目は元々「Solve $Mz_1 = r$」となっているが、SUP-GMRESR は更新された M の系を解くために L-BFGS スキームを適用する手続きを呼び出す。
- L-BFGS を実行するために必要なデータを保存するため 21 行目から 24 行目が追加されている。より正確には、これらの行は、現在の反復と前の反復の解ベクトル x と勾配ベクトル r の差を保存する。

残る部分は「$M_{i_{tot}} z_1 = r$」を解く箇所である。ここで、$M_{i_{tot}}$ は更新された前処理行列である。行列を更新することは計算的にコストがかかるため、Kushida と Okuda は元の BFGS の代わりに L-BFGS を用いて計算リソースを減らした。L-BFGS の使用により、39 行目を実行できる任意のタイプの従来の前処理が利用可能である。言い換えれば、反復型前処理法や陰的な M であっても SUP-GMRESR と組み合わせることができる。$|\cdot|_2$ はベクトルの 2-ノルムである。いくつかのパラメータは利用者にによって決定される必要がある。すなわち、$m_{restart}$ と l はそれぞれ GMRESR アルゴリズムの Krylov 部分空間の最大次元と L-BFGS のベクトル s と y の数を定義する。前処理行列 M の定義もユーザーの決定のよるもので、最適な手法は問題依存である。本論文では、3 次元構造応力解析に効果的な前処理であり、FrontISTR で利用可能な 3×3 Block Symmetric Successive Over Relaxation 法（Block SSOR）および Block Incomplete Lower-Upper factorization (BILU) が使用されている。Block 化前処理の詳細は他の論文を参照 [8]。

3. FrontISTR への実装

(1) 設計指針

Algorithm 1 を見たときにアルゴリズムの主要部分が以下の演算で構成されていることがわかる；

- 行列ベクトル積
- ベクトル同士の加算・減算
- ベクトルの定数倍
- ベクトルの内積（自分自身との計算、つまりノルムの計算を含む）。

上記の演算はいわゆる BLAS でも提供されるほどの基本的な演算であり、FrontISTR でも該当する関数が用意されている。実際、共役勾配法などの FrontISTR に内蔵されている他の Krylov 部分空間法はそのような関数群で構成されている。SUP 法に特徴的な箇所として、23 行目および 24 行目のベクトルを保持する箇所が挙げられる。ここでは、最も古いベクトルを破棄し、新たにベクトルを追加する必要がある。最も簡単な実装は一つ若いインデックスのベクトルで古いベクトルを上書きするものであろうが、現在の計算機ではそのようなデータのコピーは非常にコストが高い。ここでは、FrontISTR のソルバーが Fortran90 で書かれていることを考慮し、データの保持に二次元配列を使い、現在インデックスを保持する別の小さな配列用意することで実装した。C 言語などではポインタを使った実装が簡便であろう。

Algorithm 1 Pseudocode of SUP-GMRESR

1: **procedure** GMRESR
2: Let x_0 be the initial guess, M be a conventional preconditioning matrix
3: $m_{restart}$ is the number of Krylov subspace,
4: l is the number of L–BFGS vectors to store
5: $c_m, u_m, z_m, p_m, (m = 1, \cdots, m_{restart})$, and $s_n, y_n\ (n = 1, \cdots, l)$ are sets of vectors
6: $i_{tot} \leftarrow 0, x \leftarrow x_0$
7: **while** true **do**
8: $r \leftarrow b - Ax$
9: **for** $k = 1, 2, \cdots, m_{restart}$ **do**
10: call L–BFGS–precond($z_1, r, M, \{y\}, \{s\}$)
11: $p_1 \leftarrow Az_1$
12: **for** $i = 1, 2, \cdots, k-1$ **do**
13: $\alpha \leftarrow c_i^T p_i$
14: $p_{i+1} \leftarrow p_i - \alpha c_i$
15: $z_{i+1} \leftarrow z_i - \alpha u_i$
16: **end for**
17: $c_k \leftarrow c_k / \|p_k\|_2$
18: $u_k \leftarrow u_k / \|p_k\|_2$
19: $x \leftarrow x + u_k c_k^T r$
20: $r \leftarrow r - c_k c_k^T r$
21: $s_{i_{tot}} \leftarrow u_k c_k^T r$
22: $y_{i_{tot}} \leftarrow c_k c_k^T r$
23: $\{s\} = \{s_{i_{tot}-l+1}, s_{i_{tot}-l+2}, \cdots, s_{i_{tot}}\}$
24: $\{y\} = \{y_{i_{tot}-l+1}, y_{i_{tot}-l+2}, \cdots, y_{i_{tot}}\}$
25: **if** $\|r\|$ is sufficiently small **then**
26: exit
27: **end if**
28: $i_{tot} \leftarrow i_{tot} + 1$
29: **end for**
30: **end while**
31: **end procedure**
32: **procedure** L–BFGS–PRECOND($t, u, M, \{y\}, \{s\}$)
33: $t \leftarrow u$
34: **for** $i = l, l-1, \cdots, 1$ **do**
35: $\rho_i \leftarrow \frac{1}{y_i^T s_i}$
36: $a_i \leftarrow \rho_i s_i^T t$
37: $t \leftarrow t - a_i y_i$
38: **end for**
39: $t \leftarrow M^{-1} t$
40: **for** $i = 1, 2, \cdots, l$ **do**
41: $c \leftarrow \rho_i y_i^T t$
42: $t \leftarrow t + (a_i - c)\, s_i$
43: **end for**
44: **end procedure**

(2) 具体的な手順

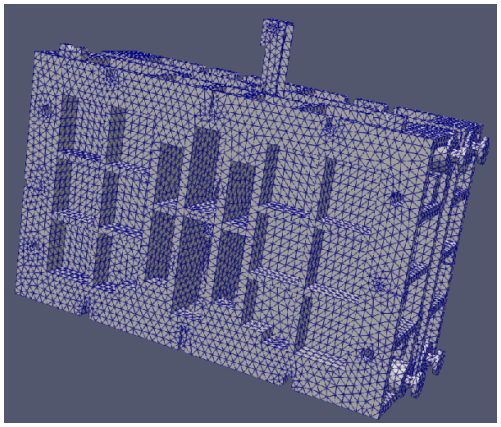
FrontISTR はオープンソースなプログラムであり、Gitlab でソースコードが公開されている。アーカイブとしてダウンロードすることも可能であるが、GIT を使うことでソースコードの管理・保存を容易にし、また、将来的に開発した機能を FrontISTR に提供することができる。`https://gitlab.com/FrontISTR-Commons/FrontISTR` にアクセスし、自分のアカウントにレポジトリを Fork することが推奨される。

FrontISTR は並列計算のフレームワークとして HECMW を採用している [9]。HECMW は共役勾配法や GMRES 法などの著名な Krylov 部分空間法を内蔵しており、新しく作成するソルバーもそれらを参考にして追加するのが簡便である。具体的には、`FrontISTR/hecmw1/src/solver/iterative` フォルダ以下にあるファイル群が相当する。例えば `hecmw_solver_CG.f90` は共役勾配法のプログラムが記述されている。`hecmw_solver_Iterative.f90` は設定ファイルをもとにソルバー呼び出すためのインターフェースであり、自分が作成したソルバーを登録する必要がある。FrontISTR の解析設定ファイルを読み込むための関数は別フォルダに格納されており、`FrontISTR/fistr1/src/common` にある、`fstr_ctrl_common.f90` が相当する。当該ファイルの `mlist` 変数に自分が開発したソルバー名を登録すると、それに応じて `hecmw_solver_Iterative.f90` の `METHOD` 変数に整数値が代入される。2024 年 4 月現在、`character(92) :: mlist = '1,2,3,4,101,CG,BiCGSTAB,GMRES,GPBiCG,GMRESR, GMRESREN,DIRECT,DIRECTmkl,DIRECTlag,MUMPS,MKL'` となっており、GMRESR の場合は `METHOD` に 5 が代入される。これは、`mlist` が数字を使ったソルバーの識別子、反復法ソルバーの英語名のリスト、直接法ソルバーの英語名のリストで構成されており、GMRESR は反復法ソルバーの 5 番目に登録されているためである。関連して、`integer(kind=kint) :: number_number = 5` は数字を使った識別子の数、`integer(kind=kint) :: indirect_number = 6` は反復法ソルバーの数であるので、新たにソルバーを追加した場合はこれらの定数値も適切に変更しなくてはならない。

以上、FrontISTR にソルバーを追加するに当たり必要であった情報を略述した。実際の作業には Fortran90、CMAKE および GIT のの知識も要求されるであろうが、インターネット上に記載されている解説で十分理解できる。

4. 性能評価

このセクションでは、GMRES、GMRESR、および SUP-GMRESR の性能について議論する。GMRES は FrontISTR に内蔵されたソルバーであり、これまでの研究により性能がよく評価されているため、よい比較対象となる。本研究では、FrontISTR によって用意された Mold(金型) モデルおよび、Lung モデルを例題とした。Figure 1 に Mold 問題のメッシュ図を、2 に Lung 問題のメッシュ図を示す。モデルは四面体二次要素を用いて



図–1 Mold モデルのメッシュ図

離散化された。本研究では二種類のコンピュータシステムを利用しており、Mold 問題の 1×10^6 有限要素節点, 55×10^6 有限要素節点、および Lung 問題 (2.3×10^6 有限要素節点) の比較的小規模な問題は以下の PC クラスタシステムを用いて計算した：

- 計 6 台の計算ノード。
- ノードあたり Intel Xeon Platinum 9242（2.3GHz、48 コア）x2。合計でノードあたり 96 コア。
- ノードあたり DDR4-2033 16GB モジュール x24（合計 384 GB）。
- InfiniBand EDR インターコネクト。

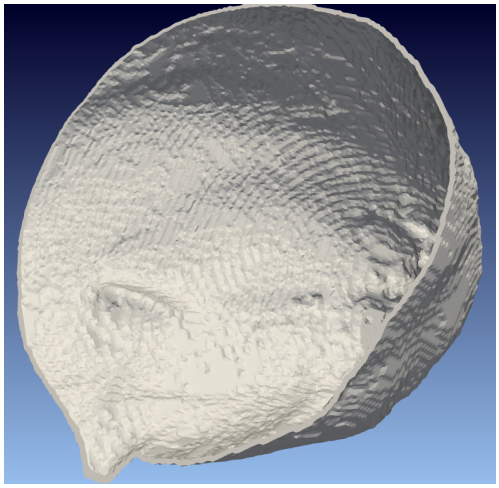
他方、最大規模の問題である 200×10^6 有限要素節点の問題は、東京大学に設置されている Wisteria-Odyssey 6,912 コアを用いて計算した。

すでに議論したとおり、$m_{restart}$ はアルゴリズムにおいて Krylov 部分空間の最大次元を定義する。Mold 問題では $m_{restart} = 20$、Lung 問題では $m_{restart} = 100$ とした。SUP-GMRESR の L-BFGS 部分に格納される y および s のベクトルの数はアルゴリズム内で l と定義されており、Mold 問題では $l = 1$、Lung 問題では $l = 1$ および $l = 5$ と設定した。Lung 問題は薄肉な形状をしており、反復法の収束性が悪くなる。そのため、収束を得るために Mold 問題よりも Krylov 部分空間の最大次元や Hessian 行列の近似度を高くする必要があった。

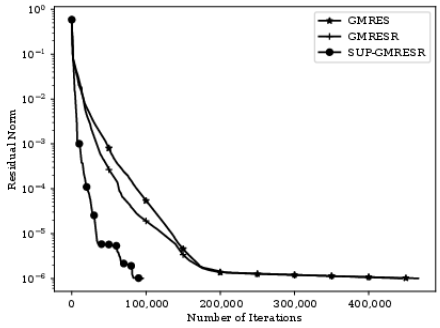
(1) Mold 問題

表 1 に、各ソルバーの収束に必要な反復回数と計算時間を記載している。表内の 11×10^6、55×10^6、および 200×10^6 は、それぞれ 11×10^6 問題、55×10^6 問題、および 200×10^6 問題を指す。GMRES と GMRESR は収束するが、SUP-GMRESR に比べて約 4 から 5 倍の反復回数を要する。計算時間もそれに応じて増加した。

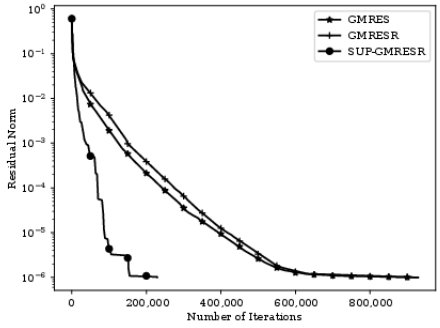
図 3 から図 5 は、それぞれ 11, 55 および 200×10^6 FE ノードを持つ Mold 問題における各ソルバーの収束履歴を示している。11×10^6 モデルは 1 つの計算ノード（96 コア）を使用して計算され、55×10^6 モデルは 4 つの計算ノード（480 コア）を使用して計算された。200×10^6 モデルは、より大きなメモリスペースが必要なため、東京大学に設置されている Wisteria-Odyssey スーパーコ



図–2 Lung モデルのメッシュ図



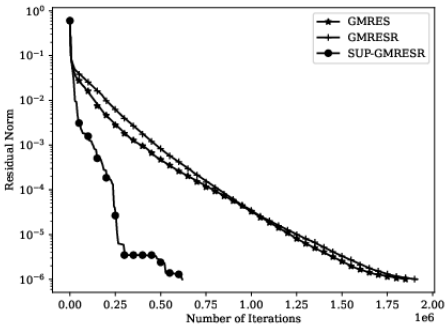
図–3 11×10^6 **問題の収束履歴**



図–4 55×10^6 **問題の収束履歴**

表–1 各ソルバーの収束に必要な反復回数と計算時間。$11{\times}10^6$**、**$55{\times}10^6$**、および** $200{\times}10^6$ **は問題を示している。GMRES、GMRESR、および SUP-GMRESR はソルバーのタイプを示している。相対計算時間は、各ケースでの SUP-GMRESR に対して正規化された各ソルバーの計算時間を示している。**

		Number of Iterations	Relative computation time (s)
11×10^6	GMRES	466,746	4.06
	GMRESR	444,779	4.44
	SUP-GMRESR	95,816	1.0
55×10^6	GMRES	929,377	3.29
	GMRESR	928,090	3.85
	SUP-GMRESR	229,140	1.0
200×10^6	GMRES	1,852,221	2.87
	GMRESR	1,914,339	2.98
	SUP-GMRESR	619,779	1.0

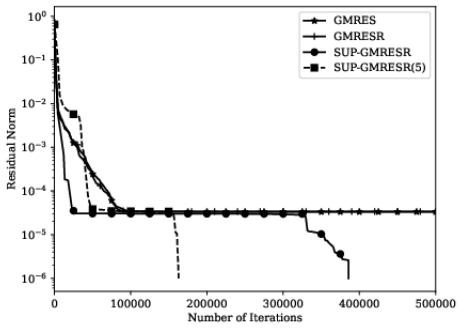


図–5 200×10^6 **問題の収束履歴**

ンピュータの 6,912 コアを使用して計算された。収束基準は $|r|_2/|b|_2 \leq 10^{-6}$ である。

(2) Lung 問題

図 6 に Lung 問題の収束履歴を示す。図中、SUP-GMRESR(5) は $l = 5$ の SUP-GMRESR を示す。図からわかるとおり、SUP-GMRESR および SUP-GMRESR(5) は収束したが、GMRES および GMRESR は 2.5×10^6 反復を経ても残差ノルムの減少が見られず収束の見込みがないため計算を打ち切った。収束したケースについて詳しく見ると、SUP-GMRESR が 385,930 反復、SUP-GMRESR(5) が 162,946 反復で収束している。計算時間は各々、1.06×10^5 秒、5.75×10^4 秒であった。Hessian



図–6 Lung 問題の収束履歴

行列の近似度を上げることは必ずしも収束性を高めるわけではないものの [7]、今回のケースでは収束までに必要な反復数を半減することができ、それにともなった計算時間もおおよそ半減させることができた。

5. まとめ

本研究では、自己更新型前処理付き GMRESR を並列有限要素法 OSS である FrontISTR に実装し、工学上実用的な問題を用いて性能を調査した。Mold 問題では、SUP-GMRESR は 95,816 回の反復で収束したのに対し、GMRES は 466,746 回の反復で収束したことから、SUP-GMRESR が GMRES や GMRESR に比べて約 4 倍の高速化を実現していることが観察された。さらに、600×10^6 DOF の大規模問題においても SUP-GMRESR の優れた収束が観察された。SUP-GMRESR は 619,779 回のステップで収束したのに対し、GMRES は 1,852,221 回のステップで収束した。Lung 問題では、SUP 法を適用しない GMRES および GMRESR では収束しなかったのにたいし、SUP 法を適用した場合収束が得られた。また、Hessian 行列の近似度をあげることで収束性が向上し、SUP-GMRESR が 385,930 反復で収束したのに対して SUP-GMRESR(5) が 162,946 反復で収束し、約 2 倍の高速化が達成された。

このように工学上実用的な問題において SUP-GMRESR が GMRES や GMRESR よりも良い収束性をしめしたが、本研究で使用された例は対称正定行列のみであり、非対称行列を使用した性能調査などさらなる研究が望まれる。

本研究で使用されたコードは、FrontISTR Gitlab リポジトリのメインブランチに統合され、自由にアクセスできる [10]。使用されたデータは、ユーザー登録を行うことで無償で入手可能な FrontISTR Commons Data Reservoir ウェブサイトから取得できる [11]。

参考文献

[1] Kushida, N.: Condition number estimation of preconditioned matrices, *PLOS ONE*, Vol.10, No.3, 0122331, 2015.

[2] Kushida, N.: Newton Raphson preconditioner for krylov type solvers on GPU devices, *SpringerPlus* vol.5, 788, 2016.

[3] Kushida, N., and Okuda, H.: Iterative approximation of preconditioning matrices through krylov-type solver iterations, *International Journal of Computational Methods*, Vol.18, No. 08, 2150027, 2021.

[4] Vorst, H.A.V., and Vuik, C.: GMRESR: a family of nested GMRES methods, *Numerical Linear Algebra with Applications*, Vol.1, No.4, pp.369–386, 1994.

[5] Eirola, T., and Nevanlinna, O.: Accelerating with rank-one updates. *Linear Algebra and its Applications*, Vol.121, pp.511–520, 1989.

[6] Kushida, N., and Okuda, H.: A new type of variable preconditioning for a general- ized minimum residual scheme, *International Journal of Computational Methods*, Vol.20 No.02, 2022.

[7] Kushida, N., Okuda, H.: A performance evaluation of a new flexible preconditioning method on a parallel finite element structure analysis program, FrontISTR, *Japan J. Indust. Appl. Math.*, vol.41, pp.723–737, 2024.

[8] Kushida, N., and Okuda, H.: Optimization of the parallel finite element method for the earth simulator, *Journal of Computational Science and Technology*, Vol.2, No.1, pp.81–91, 2008.

[9] Takahashi, Y., et al: Large-scale parallel computing performance of finite element analyses for reactor building incorporating soil-structure interaction, SMiRT-24, pp.20-25, 2017.

[10] FrontISTR: FrontISTR source code repository, *https://gitlab.com/ FrontISTR-Commons/FrontISTR/*, accessed 2024.

[11] FrontISTR: FrontISTR Data Reservoir, *https://gitlab.com/ FrontISTR-Commons/FrontISTR/*, accessed 2024.