博士論文

ロバストな並列多重格子法のための効率的なニアカーネルベクトル抽出手法の研究

(The Study of Efficient Method for Extraction of Near-Kernel Vectors for Robust Parallel Multigrid Method)

Naoya Nomura

野村 直也

Department of Mathematical Informatics
Graduate School of Information Science and Technology
The University of Tokyo
Tokyo 113-8656, Japan

March 2020

Doctoral Dissertation
submitted to
Graduate School of Information Science and Technology, The University of Tokyo
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the field of MATHEMATICAL INFORMATICS

概要

大規模連立一次方程式に対する高速かつスケーラブルな解法として，Multigrid 法が広く用いられている．Multigrid 法は階層的に粗い問題を生成することで，様々な波長の誤差成分を効率よく減衰させ，高速で安定な収束やスケーラビリティを実現している．しかし，悪条件の問題においては収束性が悪化する傾向にあることが知られている．Multigrid 法は粗い問題の生成方法により，幾何的マルチグリッド (Geometric Multigrid, GMG) 法と，代数的マルチグリッド (Algebraic Multigrid, AMG) 法が存在する．その中でさらに本研究では，AMG 法の一種である SA-AMG（Smoothed Aggregation - Algebraic MG）法を用いている．著者らの先行研究などから，SA-AMG 法において粗い問題生成時に，問題に応じたニアカーネルベクトル $\boldsymbol{e}$（$A\boldsymbol{e} \approx 0$ となる非ゼロベクトル，0 固有値に近い固有ベクトルに対応）を，粗い係数行列生成に必要な補間演算子を作成する際に適切に用いることにより，より高速で安定な収束が実現できることがわかっている．そこで本研究では，悪条件問題においても高速かつ安定な収束の実現に向け，以下の 2 つについての研究を行った．

- ニアカーネルベクトル抽出手法の提案
- Hybrid 並列を適用することによる計算コストや通信時間の分析

ニアカーネルベクトル抽出手法について，本研究では新たに 3 つの抽出手法を提案する．まず，全レベル近似ニアカーネルベクトル抽出法を提案手法 1 として提案する．この手法では，全レベルの各係数行列に対してニアカーネルベクトルの抽出を行う．これにより，粗いレベルの減衰しにくい波長成分も効率よく減衰でき，高い収束性が得られると期待できる．数値実験により，適切な本数のニアカーネルベクトルを抽出することで有用となるが，そのために膨大な時間が必要であることがわかった．

そこで，予測手法付全レベル近似ニアカーネルベクトル抽出法を提案手法 2 として提案する．この手法では，ニアカーネルベクトルが十分に抽出されたかを，判定式を用いることで判定を行う．これにより，無駄なニアカーネルベクトルの抽出や判定処理を削減することができ，その結果検証時間を抑えることができた．さらに，適切なパラメタ設定により有用となることがわかった．

また，抽出のさらなる効率化のため，固有値計算法を用いた粗レベル補間近似ニアカーネルベクトル抽出法を，提案手法 3 として提案した．行列サイズが小さい粗いレベルの係数行列に対して固有値解析を実施することで，0 固有値に近い固有ベクトル，つまりニアカーネルベクトルを低コストで抽出できると考えられる．数値実験により，実際に従来手法と比べ抽出時間を抑えつつ，高い収束性を実現できることがわかった．

目次

第 1 章

序論

1.1 はじめに

1.1.1 研究背景

コンピュータによるシミュレーションに基づく計算科学は，有限要素法，差分法のような手法が広く用いられる．これらの手法では，最終的に連立一次方程式 $A\boldsymbol{x}=\boldsymbol{b}$ を解くことに帰着される．近年では，より複雑な問題を正確にシミュレートすることが求められており，それにより問題の大規模化が進んでいる．そのため，大規模連立一次方程式を高速かつ安定に解く手法の開発は急務となっている．

そこで，大規模連立一次方程式を高速に解く手法として Multigrid 法が使用されている．Multigrid 法は階層的に粗い問題を生成することで，様々な波長の誤差成分を効率よく減衰させ，高速で安定な収束やスケーラビリティを実現している．Multigrid 法は大きく分けて，以下の 2 つに分類される．

- 幾何的 Multigrid（GMG）法：空間離散化の情報に基づき階層行列生成
- 代数的 Multigrid（AMG）法 [2–4]：係数行列 A のみに基づき作成

本研究ではその中で，AMG 法を対象とする．AMG 法の中でも階層行列の生成方法により，さらに様々な派生解法が存在する [5–12]．本研究ではその中で特に，Smoothed Aggregation に基づく AMG（SA-AMG）法と呼ばれる手法を対象とした [13–16]．この手法は多くの問題に対して有用であることが知られており，広く用いられている解法となっている．

Multigrid 法は，与えられた問題行列から粗い問題を再帰的に生成する構築部と，それらを用いて解く求解部にわかれている．SA-AMG 法における構築部では，問題行列に基づくグラフ構造を作成し，それを基にアグリゲートと呼ばれる節点集合を作成する．これを再帰的に行うことで，次元数のより小さい複数の行列を作成する．求解部では，構築部で階層的に生成された行列を用い，緩和法を用いて問題行列を解く．このとき，Multigrid 法では主に，V-cycle と呼ばれる手法を用いて階層的に解いていく．

1.1.2 本研究の目的

本研究では，SA-AMG 法における高並列環境下における大規模問題に対する，高速かつ安定な収束性の実現を目的とする．そのために，以下の 2 つについての研究を行った．

- ニアカーネルベクトル抽出手法の提案
- Hybrid 並列を適用することによる計算コストや通信時間の分析

まず，ニアカーネルベクトル抽出手法の提案について述べる．SA-AMG 法は，収束しにくい誤差成分を粗いレベルで効率よく減衰させることで，高い収束性を実現している．ここで，収束しにくい成分とは，一般にニアカーネルベクトルと呼ばれ，問題行列 A との行列ベクトル積が 0 に近くなるような非ゼロベクトルをいう．またニアカーネルベクトルは，0 固有値に近い固有ベクトルに対応する．Gauss-Seidel 法のような通常の定常反復解法で解いた際，この成分が収束の停滞を引き起こす要因となることが知られている．SA-AMG 法ではこのような誤差成分を効率よく減衰させるため，構築部においてニアカーネルベクトルを用いて粗いレベルの行列を生成する．これにより，粗いレベルの行列に誤差成分が射影され，誤差成分を効率よく減衰させることが可能となる．その結果，残差を効率よく収束させることができる．

本研究では科学技術計算において広く用いられている，3 次元弾性体の問題を用いて実験を行っている．この問題では平行移動成分と回転成分がニアカーネルベクトルとして知られている．著者らによる先行研究により，SA-AMG 法においてこれらのニアカーネルベクトルを設定することにより，反復回数と実行時間双方で改善がみられることがわかっている．しかし，これは問題設定に依存したものであり，すべての問題において適用できるものではない．すべてのユーザーが，対象とする問題の物理的性質に基づいた，適切なニアカーネルベクトルの設定を行うことができるとは限らない．SA-AMG 法の高い収束性をより多くの問題に対して生かすために，係数行列 A から適切な数のニアカーネルベクトルを，代数的に効率よく抽出する手法の開発が急務である．

そこで著者らは，問題行列に応じた適切なニアカーネルベクトルの設定方法と，その効果についての研究を行ってきた．著者らの現在までの研究では，αSA 法 [17,18] と呼ばれる手法を基に，ニアカーネルベクトル抽出の効率化のため，V-cycle を用いてレベル 1（元の問題行列が置かれる最も細かいレベル）のみにおいて，問題行列からニアカーネルベクトルを複数本抽出する手法の提案，および有用性の検証を行った（詳細は 5.2 章にて述べる）．そして，その手法で抽出したニアカーネルベクトルを SA-AMG 法に適切な本数設定する（適切な本数の設定については，5.4.3 節にて実験結果を交え，詳細を述べる）ことで，平行移動成分と回転成分を設定した場合よりも，反復回数と実行時間双方で改善がみられることがわかった [19]．

[17,18] や [19] を含め，従来の関連研究では，階層の全レベルにおいて同一本数のニアカーネルベクトルを用いて設定を行っている（例えば，細かいレベルで 3 本設定した場合，粗いレベルでも 3 本用いて計算を行っている）．これは SA-AMG 法では通常，細かいレベルのニアカーネルベクトルをもとに，次の粗いレベルのニアカーネルベクトルを代数的に構築しているためである．しかし，細かいレベルから生成されたニアカーネルベクトルだけで

は，粗いレベルの行列におけるニアカーネルベクトル成分の減衰が不十分であり，高い収束性を生かしきれていないのではないかと考えた．また [19] において，ニアカーネルベクトルを「適切な」本数設定することができれば，改善がみられることがわかった．しかし [19] でさらに，多く設定することが必ずしも改善するとは限らないこともわかった．そのため，適切なニアカーネルベクトル設定本数の検証が必要であるが，従来では抽出本数の取りうるすべてのパターンを網羅する必要があり，検証に膨大な時間を必要としてしまうという問題があった．そのため，より高い実行効率を得るためには，適切な本数を抽出時点で判明させることが理想である．これらを考え，本研究では新たに 3 つのニアカーネルベクトルの抽出手法を提案した（概要を次節にて，詳細を 5 章にて述べる）．

次に，Hybrid 並列を適用することによる計算コストや通信時間の分析について述べる．近年では，より複雑な問題を正確にシミュレートすることが求められている．それにより問題の大規模化が進み，大規模問題に対する実行時間削減が大きな課題となっている．このような大規模問題を解く際には，スーパーコンピュータのような超並列計算機を用いることが必須となる．しかし，並列数が増加するにつれ通信コストが増大し，実行時間悪化につながる．近年の並列計算機は，メニィコアプロセッサを結合したクラスタ形式が採用されている計算機が主要形態のひとつとなっている．そのような環境では，メッセージパッシング（MPI, Message Passing Interface）と，スレッド並列（OpenMP）を併用した Hybrid 並列プログラミングモデルが適しているといわれる．Hybrid 並列の適用により，通信の効率化による通信コストの削減が期待できる．

1.1.3 ニアカーネルベクトル抽出と本研究での提案手法

本研究の最終目標は，SA-AMG 法の適用性（Applicability）の拡張である．目標実現のためには，適切な本数のニアカーネルベクトルを，効率よく抽出する手法の開発が急務である．そこで，本研究では 3 つのニアカーネルベクトル抽出手法を提案した．

まず提案手法 1 について述べる．上記で述べたように，従来手法（αSA 法 [17, 18]，著者らによる先行研究 [19]）では細かいレベルのニアカーネルベクトルをもとに，次の粗いレベルのニアカーネルベクトルを代数的に構築する．この場合，代数的に構築されたニアカーネルベクトルが粗いレベルの行列のニアカーネルベクトルに対応しているかは不明であり，関係が崩れている可能性がある．そこで本研究では [19] の手法を改良し，粗いレベルでニアカーネルベクトルを複数本抽出する手法の提案を行った．これにより，崩れていた各レベルにおけるニアカーネルベクトルの要件を維持することができるため，従来手法で不確実であった粗いレベルの行列における減衰しにくい成分の減衰を，効率よく行えるようになると考えられる．結果として，収束性改善につながることが期待できる．本手法により，抽出されたニアカーネルベクトルを適切な本数使用することで，反復回数と実行時間双方で有用となることがわかった．しかし，適切な本数の検証に膨大な時間がかかってしまうこともわかった．

そこで，提案手法 2 として，抽出時において適切な設定本数を予測する手法を追加した抽出手法の提案を行う．この手法では，ニアカーネルベクトルが十分に抽出されたかを，判定式を用いることで判定を行う．これにより，無駄なニアカーネルベクトルの抽出や判定処

理を削減することができ，検証時間の削減効果が期待できる．本手法により，適切なパラメタ設定を行うことで，抽出時間を抑えつつ，適切なニアカーネルベクトルを設定した場合と同等の収束性を得られることがわかった．

上記のニアカーネルベクトル抽出手法は，ニアカーネルベクトルの候補ベクトルの算出を 1 本ずつ行う．そのため，抽出本数によっては実際に解法を適用する時間と比べ，抽出コストが大きくかかってしまう場合がある．そこで，さらに効率的に適切なニアカーネルベクトルの抽出を行うために，提案手法 3 として，粗いレベルで固有値解析を実施し補間を行うことでニアカーネルベクトルを抽出する手法の提案を行う．ニアカーネルベクトルは 0 固有値に近い固有ベクトルに対応するが，最も細かいレベルの問題行列のような大規模問題に対し，直接固有値解析を実施すると膨大な時間が必要となる．本手法は効率的に抽出するため，粗いレベルにおいて 0 固有値に近い固有ベクトルを求め，補間演算子を用いて細かいレベルへの補間を行うことで，複数階層のニアカーネルベクトルの抽出を行う手法となっている．実験では，従来手法や 1 つめの抽出手法と反復回数や抽出時間を含めた実行時間で比較を行い，有用性の検証を行った．

その後，上記で述べた提案手法の有用性を他問題で検証するために，Texas A&M 大学の SuiteSparse Matrix Collection [20] より取得した問題に対して計測を行った．本手法により，抽出時間をさらに抑えつつ，1 つめの抽出手法とほぼ同等の高い収束性を得ることができた．

1.1.4 高並列環境下における実行時間効率化に向けた手法の適用

本研究では SA-AMG 法に OpenMP/MPI Hybrid 並列を適用した際の結果を分析する．実験では，JCAHPC による Oakforest-PACS [21]，そして九州大学情報基盤研究開発センターによる ITO [22] の，2 つのスーパーコンピュータシステム上において最大 64 ノードを用い，それぞれの環境において Hybrid 並列化による SA-AMG 法全体の実行時間や，通信時間の分析を行った．

上記に加え，本研究では高並列環境下における実行を考え，SA-AMG 法に Coarse Grid Aggregation （CGA）と呼ばれる手法を適用し，実験を行った結果も示す．並列時にアグリゲートを作成する際，各プロセスがそれぞれ独立にアグリゲート生成を行う（独立アグリゲート生成手法）と，最下層においてアグリゲート数がプロセス数以下にならず，高並列環境下において問題となる．そこで，各プロセス領域である程度粗くした後で，プロセス領域を集約することでこの問題に対処する CGA が提案された．数値実験では，高並列環境下において CGA を適用し，有用性の検証を行う．

1.2 本論文の構成

本論文は 7 つの章により構成されている．まず 2 章では，本研究において基礎となる連立一次方程式の解法である反復法について説明する．具体的には，まず定常反復法の概要を説明し，代表的な手法である Jacobi 法や Gauss-Seidel 法の手法の概要を説明する．その後，非定常反復法について説明し，代表的な手法である CG 法，さらに本研究で用いている前処

理付 CG 法についての手法の概要を説明する．

3 章および 4 章では，Multigrid 法とその派生形で本研究の対象となる手法である SA-AMG 法について述べる．3 章では Multigrid 法の概要を，Multigrid 法の基礎である 2 段グリッド法を交え概要を説明する．その後，Multigrid 法の派生解法である AMG 法についての説明を行う．4 章では，SA-AMG 法の概要を説明し，SA-AMG 法の内部で用いている CGA についての説明を行う．

5 章が本論文の中核となる．まずはじめに，ニアカーネルベクトルの概要を述べ，SA-AMG 法における位置づけ，およびニアカーネルベクトルを用いることによる収束性への影響および必要性を，実験を交え示す．その後，適切なニアカーネルベクトルを抽出する手法について説明する．まず関連研究および著者らによる先行研究による抽出手法 [19] について説明し，その後，本研究で提案するニアカーネルベクトル抽出手法について説明する．本研究では 3 つのニアカーネルベクトル抽出手法を提案しており，それぞれについて数値実験を交え，有用性の検証を行う．

6 章では，本手法を高並列環境下において実行することを考え，メニィコアクラスタ形式の計算機において，有効な並列化手法である Hybrid 並列を SA-AMG 法へ適用した際の結果を示す．まず Hybrid 並列の概要を述べ，Hybrid 並列を施す際に必要となるマルチカラーオーダリングの処理について説明する．その後，数値実験とそれにより得られた結果や知見を示す．

最後に第 7 章にて，本論文で示した研究成果を総括するとともに，研究成果により得られた新たな課題について述べる．

1.3 本研究で用いた問題行列

本研究では，3 次元弾性体問題を主に用いている（5.2.2 節と 5.2.1 節，および 5.7 節ではそれぞれ異なる問題を用いており，それらの問題行列の詳細に関しては，各節にて述べる）．本節では，3 次元弾性体問題の概要と，本研究で用いた問題設定を示す．

物体（弾性体）に外力が作用した際に，物体の変形に伴い，応力が発生する．弾性体問題は，外力と応力が釣り合うまで，どこまで弾性体が変形するかを求める問題である（厳密な内容については [23] 等にゆだねる）．そして本研究では，3 次元の弾性体問題に対し，立方体を単位要素とした有限要素法を適用することで離散化を行い，問題行列の生成を行っている．これにより，最終的に弾性体問題は連立一次方程式 $A\boldsymbol{u}=\boldsymbol{f}$ に帰着できる（ここで，$\boldsymbol{u}$ は物体の変位，$\boldsymbol{f}$ は物体に加えられた力を示す）．またこの問題は 3 次元なため，係数行列の生成時には 1 節点あたりの行列要素が 3×3 のブロック行列に表現される．上記でも述べた平行移動成分や回転成分は，物体の変形が起こらない，Zero Strain と呼ばれる成分となる．つまり，右辺ベクトル $\boldsymbol{f}$ が 0 となり，このような成分は弾性体問題においてニアカーネルベクトルとなる [24]．図 1.1 に，3 次元弾性体問題における平行移動成分と回転成分の例を示す．図 1.1 のように，平行移動や回転は物体の変形が伴わず，応力も働かない．そのため，3 次元弾性体問題ではこれらの成分は，ニアカーネルベクトルとされる．ここで，実験で用いた弾性体の問題設定を図 1.2 に示す．この弾性体の問題は図 1.2 のように，立方体の弾性体に対して，ある一部分に力を加えたとき，どのように変形するかを解く問題となって

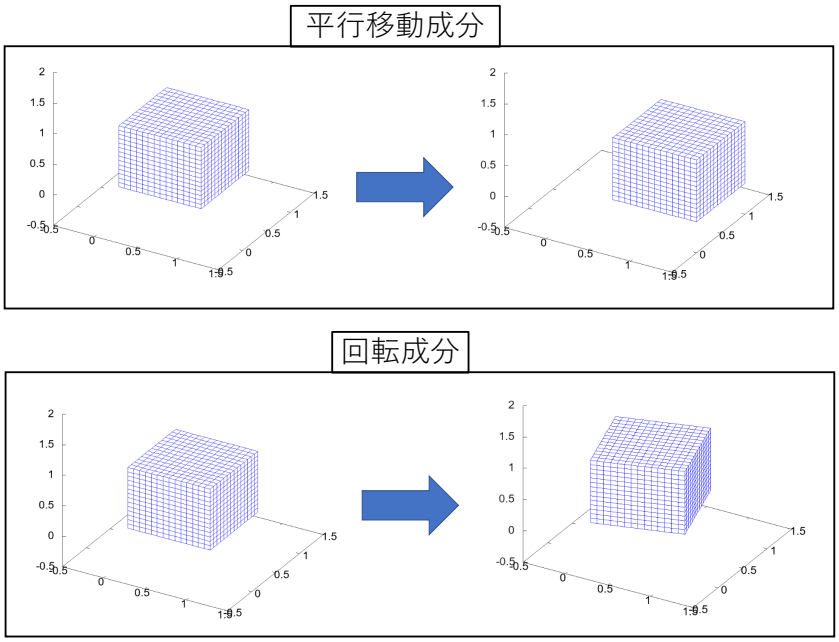


図 1.1: 3 次元弾性体問題における平行移動成分と回転成分の例

いる．弾性体問題では，ヤング率とポアソン比と呼ばれる，弾性体の応力とひずみに関するパラメータが存在する．これらのパラメータは，問題行列の条件数に影響する（例として，ポアソン比は 0.5 に近くなるほど悪条件となる）．本実験では，単純な問題において手法の分析を行うため，ヤング率をすべての部分で 1，ポアソン比を 0.3 と，比較的条件数が良い設定で実験を行った．

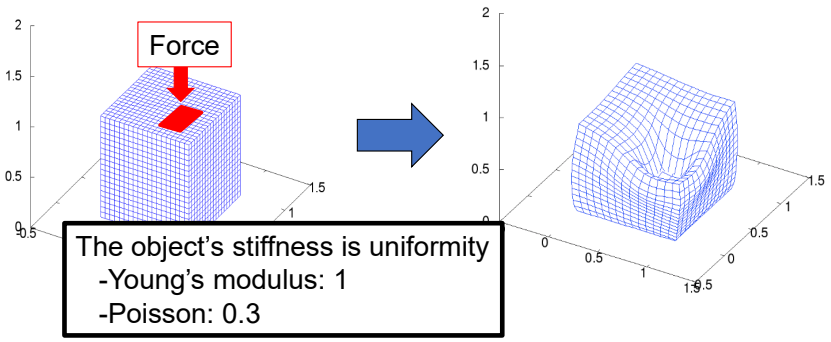


図 1.2: 数値実験で使用した 3 次元弾性体の問題設定

第 2 章

反復法

本節では，連立一次方程式

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.1}$$

を解く手法の一つである反復法について述べる（A は $n \times n$ $(n \in \mathbb{N})$ の正定値対称行列とし，以降本論文では特に断りがない限り，同様の設定を用いるものとする）．反復法は，適当な初期解 $\boldsymbol{x}_0$ から開始し，ある同一の操作を複数回反復して適用することで，近似解を得る手法である．反復法は，主に定常（Stationary）反復法と非定常（Nonstationary）反復法にわかれる．本節では，それぞれに関しての説明を行う．

2.1 定常反復法

定常反復法とは，解くべき方程式である式 (2.1) の解に収束する列 $\{\boldsymbol{x}_k\}_{k\in\mathrm{N}}$ を，

$$\boldsymbol{x}_{k+1} = M\boldsymbol{x}_k + N\boldsymbol{b} \tag{2.2}$$

の形の漸化式で生成する手法のことである．ここで，このような行列 M を反復行列と呼ぶ．本節では，定常反復法の代表的な手法である，Jacobi 法や Gauss-Seidel 法，さらにそれらに加速パラメータを追加した減速 Jacobi 法や SOR 法について述べる．

まず，Jacobi 法について述べる．連立一次方程式の係数行列 A を，上三角行列 U，下三角行列 L，対角行列 D に分解したと考える．つまり，

$$A = D + U + L$$

であると考える．このとき，連立一次方程式は，

$$(D + U + L)\boldsymbol{x} = \boldsymbol{b}$$

と表すことができ，変形すると，

$$\boldsymbol{x} = D^{-1}\{\boldsymbol{b} - (L + U)\boldsymbol{x}\}$$

が得られる．このとき，初期値を $\boldsymbol{x_0}$，k 回目の反復で得られた近似解を $\boldsymbol{x_k}$ とすると，Jacobi 法は以下のように表される．

$$\boldsymbol{x}_{k+1} = D^{-1}\{\boldsymbol{b} - (L+U)\boldsymbol{x}_k\}$$

変形すると，

$$\boldsymbol{x}_{k+1} = D^{-1}\boldsymbol{b} - D^{-1}(L+U)\boldsymbol{x}_k \tag{2.3}$$

$M = -D^{-1}(L+U)$，$N = D^{-1}$ とすれば，上記で述べた漸化式の形に一致する．Jacobi 法は，係数行列 A の対角要素の絶対値が，非対角要素の絶対値の総和より大きい場合，つまり

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|$$

が満たされていれば，十分な回数反復を行うことで厳密解に近くなることが知られている（このとき,（狭義）対角優位であるという）[25].

次に，Gauss-Seidel 法について述べる．Gauss-Seidel 法は以下のように表される．

$$\boldsymbol{x}_{k+1} = -L\boldsymbol{x}_{k+1} - U\boldsymbol{x}_k + D^{-1}\boldsymbol{b}$$

Jacobi 法では，次の反復の近似解 $\boldsymbol{x}_{k+1}$ の計算の際に，前回の反復の近似解 $\boldsymbol{x}_{k+1}$ を用いることで更新を行う．一方 Gauss-Seidel 法では，$\boldsymbol{x}_{k+1}$ の値を計算するために，すでに同じ反復で更新された近似解を用いる．これにより，一般に Gauss-Seidel 法は Jacobi 法と比べ収束性が良い [25]．Gauss-Seidel 法も同様に，以下のように変形できる．

$$\boldsymbol{x}_{k+1} = (I + D^{-1}L)^{-1}(-D^{-1}U)\boldsymbol{x}_k + (D+L)^{-1}\boldsymbol{b}$$

$M = (I + D^{-1}L)^{-1}(-D^{-1}U)$，$N = (D+L)^{-1}$ とすれば，漸化式の形となる．

実際にこれらの手法を計算機上で用いる際には，各反復で計算された値をそのまま用いるのではなく，各反復における補正量 $\boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ に，加速係数 ω（$1 < \omega$）を乗じて，収束性を改善する手法が用いられることが多い．つまり，次の反復への近似解 $\boldsymbol{x}_{k+1}$ を求める際に，

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \omega(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$$

として，解を修正する．Jacobi 法に加速係数 ω を導入した手法は，重み付き（または減速）Jacobi 法と呼ばれ，漸化式は以下のようになる．

$$\boldsymbol{x}_{k+1} = \{(1-\omega)I - \omega D^{-1}(L+U)\}\boldsymbol{x}_k + \omega D^{-1}\boldsymbol{b}$$

また，同様に Gauss-Seidel 法に加速係数 ω を導入した手法を SOR（Successive Over Relaxation method）法と呼び，漸化式は以下のようになる．

$$\boldsymbol{x}_{k+1} = (I + \omega D^{-1}L)^{-1}\{(1-\omega)I - \omega D^{-1}U\}\boldsymbol{x}_k + \omega(D + \omega L)^{-1}\boldsymbol{b}$$

定常反復法においては，減衰しにくい成分，つまり，式 (2.2) で示した定常反復法の反復行列 M に対し，

$$M\boldsymbol{e} \approx \boldsymbol{e} \tag{2.4}$$

となる成分 $\boldsymbol{e}$ が存在する．このような成分は一般に，代数的に滑らかな成分と呼ばれる [25]．代数的に滑らかな成分は，係数行列 A との行列ベクトル積が小さいという特徴がある．実際に，適当な条件下の下でこれを容易に示すことができる．例として，定常反復法として減速 Jacobi 法を用いるとする．このとき，反復行列 M は，

$$M = I - \omega D^{-1}A$$

であった．これを式 (2.4) に代入して整理すると，

$$\omega D^{-1}A\boldsymbol{e} \approx 0$$

が得られる．係数行列の対角要素 D が十分な大きさであれば，これより，

$$A\boldsymbol{e} \approx 0 \tag{2.5}$$

が導かれる．このように，代数的に滑らかな成分は，係数行列 A との行列ベクトル積が小さくなることがわかる．

2.2 非定常反復法

非定常反復法とは，次の反復への更新解 $\boldsymbol{x}_{k+1}$ への漸化式が，$\boldsymbol{x}_k$ に関して非線形かつ非定常である反復解法のことをいう．代表的な手法として，CG（Conjugate Gradient，共役勾配）法や CG 法の派生解法である BiCG（双共役勾配）法，また GMRES（Generalized Minimal Residual，一般化最小残差）法がある．本研究では特に CG 法を用いているため，以下より CG 法について述べる．

解くべき連立一次方程式である式 (2.1) に対し，方程式の厳密解を $\boldsymbol{x^*} = A^{-1}b$ とする．係数行列 A は対称正定値行列という設定を用いると，解くべき連立一次方程式は，以下を最小にする x を求めることと同義である．

$$(x - x^*)^T A(x - x^*) \tag{2.6}$$

式 (2.6) を変形すると，以下のようになる．

$$\begin{aligned}(x - x^*)^T A(x - x^*) &= (x, Ax) - (x^*, Ax) - (x, Ax^*) + (x^*, Ax^*) \\ &= (x, Ax) - 2(x, Ax^*) + (x^*, Ax^*) \\ &= (x, Ax) - 2(x, b) + (x^*, b)\end{aligned} \tag{2.7}$$

ここで，$(*, *)$ はベクトル同士の内積を表す．(x^*, b) は定数であるため，式 (2.7) は最終的

に以下のような 2 次関数，

$$Q(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x}, A\boldsymbol{x}) - (\boldsymbol{b}, \boldsymbol{x}) + C \tag{2.8}$$

の最小点を求めることと同義となる [25]．ここで，定数である (x^*, b) を C と置いた．したがって，関数値 $Q(\boldsymbol{x})$ を減少させるようなベクトル $\boldsymbol{x}$ の列 $\boldsymbol{x}_0, \boldsymbol{x}_1, ...$ を生成すれば，真の解 $\boldsymbol{x}^*$ の近似解が得られると期待できる．ここで，近似解のベクトルの列の生成方法について，適当な $\boldsymbol{x}_0$ から始めて，各反復 $k = 0, 1, ...$ に対し適当に定めた探索方向 $\boldsymbol{p}_k$ において，最小となるステップサイズ α_k を定める．その後，次の反復の更新解 $\boldsymbol{x}_{k+1}$ を，

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \tag{2.9}$$

と更新することを考える．これを，逐次最小化法と呼ぶ．ここで，近似解 $\boldsymbol{x_k}$ の残差 $\boldsymbol{r_k}$ を，

$$\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$$

とすると，式 (2.8) と式 (2.9) より，

$$Q(\boldsymbol{x}_k + \alpha \boldsymbol{p}_k) = Q(\boldsymbol{x}) - \alpha_k(\boldsymbol{r}_k, \boldsymbol{p}_k) + \frac{\alpha_k^2}{2}(\boldsymbol{p}_k, A\boldsymbol{p}_k) \tag{2.10}$$

が得られる．α について微分したもの（$dQ/d\alpha$）に対し，$dQ/d\alpha = 0$ として整理すると，

$$\alpha_k = \frac{(\boldsymbol{r}_k, \boldsymbol{p}_k)}{(\boldsymbol{p}_k, A\boldsymbol{p}_k)}$$

となり，このように α を設定することで．式 (2.10) を最小化する．ここまでで，逐次最小化法における α の取るべき値が求まった，次に，探索方向 $\boldsymbol{p}$ に関して述べる．探索方向として最も単純な定め方は，$Q(\boldsymbol{x})$ の最小化が目的であったため，$Q(\boldsymbol{x})$ の最急降下方向 $\boldsymbol{s}_k$，つまり，

$$\boldsymbol{s}_k = -\nabla Q(\boldsymbol{x}_k) = \boldsymbol{r}_k$$

とすることである．ここで，∇Q は，Q の勾配を表す．このような逐次最小化法を最急降下法と呼ぶ．最急降下法は単純な手法であるが，係数行列 A が悪条件である場合，収束性が悪化する．具体的には，係数行列 A の最大固有値を λ_{max}，最小固有値を λ_{mix} とすると，

$$Q(\boldsymbol{x}_{k+1}) \leqq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^2 Q(\boldsymbol{x}_k)$$

となることが知られている（Kantorovich の不等式）[26, 27]．

そこで，残差をより効率的に収束させることができる手法として，CG（Conjugate Gradient，共役勾配）法が提案された．CG 法では，探索方向 p を A 共役にとる，つまり，k 回反復したと仮定すると，

$$(p_i, Ap_j) = 0 \qquad (0 \leqq i < j \leqq k)$$

となるように p をとることを考える．そして，$Q(\boldsymbol{x})$ の最急降下方向を基にして，修正方向を決定することを考え．

$$p_k = r_k - \sum_{j=0}^{k-1} \frac{(r_k, Ap_j)}{(p_j, Ap_j)} \tag{2.11}$$

と定める．このように定めることで，A 共役であり，他の探索方向とは独立な，新たな探索方向のベクトルを生成することができる．一般に，ベクトル y に行列 A をかけてできるベクトル群 $A, A\boldsymbol{y}, A^2\boldsymbol{y}...$ による空間

$$\mathcal{K}_k(A, \boldsymbol{y}) = \mathrm{span}(\boldsymbol{y}, A\boldsymbol{y}, ..., A^{k-1}\boldsymbol{y})$$

を，Krylov 部分空間と呼ぶ．式 (2.11) のように生成された探索方向 $\boldsymbol{p}_k$ は，初期残差 $\boldsymbol{r}_0$ から生成される Krylov 部分空間と一致することが知られている．つまり，

$$\mathrm{span}(\boldsymbol{p}_0, \boldsymbol{p}_1, ..., \boldsymbol{p}_k) = \mathrm{span}(\boldsymbol{r}_0, \boldsymbol{r}_1, ..., \boldsymbol{r}_k) = \mathcal{K}_{k+1}(A, \boldsymbol{r_0})$$

となる．このように，Krylov 部分空間 $\mathcal{K}_k$ 上で探索することにより解を得る反復解法を，一般に Krylov 部分空間法と呼ぶ．CG 法は Krylov 部分空間法のひとつである．

ここで，CG 法の収束性に着目する．2 ノルムに関する条件数を，

$$\kappa = \frac{\lambda_{max}}{\lambda_{min}}$$

とすると，以下のように導かれることが知られている．

$$Q(\boldsymbol{x}_k) \leqq Q(\boldsymbol{x}_0) \cdot 4 \left(\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1} \right)^{2k} \tag{2.12}$$

CG 法は理論上，有限回の反復回数で厳密解に到達するという，大きな特徴を持っている．つまり，高々 n 回の反復により，厳密解に収束する．ただし，計算機上で実際に計算を行う際には，丸め誤差が発生するため，必ずしも n 回の反復で収束するとは限らない．また，最急降下法と比べ収束性は改善しているが，式 (2.12) からもわかるように，悪条件の問題に対しては収束性が悪化する傾向にある．

CG 法の計算手順をまとめたものを Algorithm 1 に示す．Algorithm 1 のように，CG 法を実際に用いる際には，近似解がある程度厳密解に近づいたら，収束したとして反復を終了する．本研究では，残差 $\boldsymbol{r}_k$ と右辺ベクトル $\boldsymbol{b}$ の 2 ノルムの比 $\dfrac{||\boldsymbol{r}_k||}{||b||}$ が 10^{-7} 以下となったときに収束したと判定し，反復を終了している．

Algorithm 1 CG 法のアルゴリズム

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
$\boldsymbol{p}_0 = \boldsymbol{r}_0$
for $k = 1, ...$ **do**
  $\alpha_k = \dfrac{\boldsymbol{r_k}^T \boldsymbol{r_k}}{\boldsymbol{p_k^T} A \boldsymbol{p_k}}$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
  if convergent **then**
    break
  end if
  $\beta_k = \dfrac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$
  $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
end for

2.3 前処理付 CG 法

前章でも述べたように，CG 法の収束性は条件数に強く依存している．そこで CG 法の収束性向上のため，適切な正則な前処理行列 C を用意し，式 (2.1) を，

$$(C^{-1}AC^{-T})(C^T\boldsymbol{x}) = (C^{-1}\boldsymbol{b})$$

と変形し，これに対して CG 法を適用する．言い換えれば，$\tilde{A} = (C^{-1}AC^{-T})$，$\tilde{\boldsymbol{x}} = (C^T\boldsymbol{x})$，$\tilde{\boldsymbol{b}} = (C^{-1}\boldsymbol{b})$ とし，

$$\tilde{A}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$$

に対して CG 法による求解を行うことを考える．このとき，$(C^{-1}AC^{-T})$ の条件数が 1 に近い，つまり，

$$C^TC \approx A$$

となるような C を用意することができれば，$\tilde{A}$ が単位行列に近づき，より効率的に近似解を求められる可能性がある．

CG 法に前処理行列を導入することを考える．$\tilde{\boldsymbol{r_k}} = C^{-1}\boldsymbol{r}_k$，$\tilde{\boldsymbol{p_k}} = C^T\boldsymbol{p}_k$ とおくと，CG 法内で行われている計算

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k\boldsymbol{p}_k, \quad \boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_k A\boldsymbol{p}_k, \quad \boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k\boldsymbol{p}_{k-1}$$

は，

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \tilde{\alpha}_k\boldsymbol{p}_k, \quad \boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \tilde{\alpha}_k\tilde{A}\boldsymbol{p}_k, \quad \boldsymbol{p}_k = \hat{\boldsymbol{r}}_k + \tilde{\beta}_k\boldsymbol{p}_{k-1}$$

のように置き換えることができる．また，$\hat{\boldsymbol{r}} = (C^T C)^{-1}\boldsymbol{r}$ とすると，$\tilde{\alpha}$ や $\tilde{\beta}$ は，

$$\tilde{\alpha_k} = \frac{(\tilde{\boldsymbol{r}_k}^T \tilde{\boldsymbol{r}_k})}{(\tilde{\boldsymbol{p}_k}^T \tilde{A} \tilde{\boldsymbol{p}_k})} = \frac{((C^{-1}\boldsymbol{r}_k)^T C^{-1}\boldsymbol{r}_k)}{((C^T\boldsymbol{p}_k)^T (C^{-1}AC^{-T}) C^T\boldsymbol{p}_k)} = \frac{(\boldsymbol{r}_k^T \hat{\boldsymbol{r}_k})}{(\boldsymbol{p}_k^T A \boldsymbol{p}_k)}$$

$$\tilde{\beta_k} = \frac{\tilde{\boldsymbol{r}}_{k+1}^T \tilde{\boldsymbol{r}}_{k+1}}{\tilde{\boldsymbol{r}_k}^T \tilde{\boldsymbol{r}_k}} = \frac{(C^{-1}\boldsymbol{r}_{k+1})^T C^{-1}\boldsymbol{r}_{k+1}}{(C^{-1}\boldsymbol{r}_k)^T C^{-1}\boldsymbol{r}_k} = \frac{\boldsymbol{r}_{k+1}^T \hat{\boldsymbol{r}}_{k+1}}{\boldsymbol{r}_k^T \hat{\boldsymbol{r}_k}}$$

となる．

これまでの計算手順をまとめたものを Algorithm 2 に示す．ただし，$M = C^T C$ とする．このように，前処理行列を用いて条件数を改善し，より効率的に近似解を求める CG 法を，一般に前処理付き CG（Preconditioned Conjugate Gradient, PCG）法と呼ぶ．

前処理行列としては，究極的には係数行列の逆行列，つまり $M^{-1} = A^{-1}$ と設定し，直接 $\boldsymbol{z} = M^{-1}\boldsymbol{r}$ を計算することが理想である．しかし，逆行列の計算はコストが大きい，そもそも逆行列が計算できれば連立一次方程式が求解できているため，実際の計算には用いない．そのため，通常近似逆行列を計算し，それを前処理行列として用いる．ごく簡単な例としては，係数行列の対角要素でできた行列 D の逆行列を用いる，つまり $M^{-1} = D^{-1}$ と設定する対角スケーリングと呼ばれる手法がある．対角スケーリングは単純であるが，悪条件の問題に対しては効果的ではない場合があることが知られている．また他にも，Jacobi 法や Gauss-Seidel 法，不完全 LU 分解（Incomplete LU Factorization）などの手法を用いて，$Mz = r$ を近似的に解くことが一般的である．

以上のように，前処理行列の計算手法は収束性改善のために重要であり，多くの研究がおこなわれている．本研究では，次章より説明を行う Multigrid 法を用いて，$Mz = r$ を近似的に解いている．

Algorithm 2 前処理付 CG 法のアルゴリズム

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
$\boldsymbol{p}_0 = \boldsymbol{r}_0$
for $k = 1, ...$ **do**
  Solve $M\boldsymbol{z}_{k-1} = \boldsymbol{r}_{k-1}$
  $\rho_{k-1} = \boldsymbol{r}_{k-1}\boldsymbol{z}_{k-1}$
  if $k == 1$ **then**
    $\boldsymbol{p}_1 = \boldsymbol{z}_0$
  else
    $\beta_{k-1} = \rho_{k-1}/\rho_{k-2}$
    $\boldsymbol{p}_k = \boldsymbol{p}_{k-1} + \beta_{k-1}\boldsymbol{z}_{k-1}$
  end if
  $\alpha_k = \dfrac{\rho_{k-1}}{\boldsymbol{p}_k^T A\boldsymbol{p}_k}$
  $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k\boldsymbol{p}_k$
  $\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_k A\boldsymbol{p}_k$
  if convergent **then**
    break
  end if
end for

第 3 章

Multigrid 法

本章では，Multigrid 法および派生解法の代数的マルチグリッド（Algebraic Multigrid）法について述べる．まず，Multigrid 法の簡単な例として 2 レベルのみの Multigrid 法（以降，2 段グリッド法と呼ぶ）をモデル問題を交え説明する．その後，マルチレベルな解法である Multigrid 法について説明し，最後に AMG 法について説明する．

3.1 Multigrid 法の概要

有限要素法のような格子を用いて離散化した問題に対し，Gauss-Seidel 法のような定常反復解法を適用すると，メッシュサイズと同じサイズの誤差が早く減衰し（空間的高周波成分），長い波長の誤差は減衰しにくい（空間的低周波成分）ことが知られている [25, 28]．そこで，Multigrid 法ではこの性質に着目し，粗い格子を階層的に生成し，それらを用いることで，低周波成分を効率よく減衰させ，高速で安定な収束やスケーラビリティを実現している．Multigrid 法の大きな特徴として、収束性が問題サイズによらないスケーラブル性があり，大規模問題に対して非常に有効な解法となっている．Multigrid 法は粗い格子の生成方法により，さらに空間離散化の情報に基づき粗い格子を生成する幾何的マルチグリッド (Geometric Multigrid, GMG) 法と，一般の連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の係数行列 A の情報のみに基づき粗い格子を生成する，代数的考察に基づいた代数的マルチグリッド (Algebraic Multigrid, AMG) 法が存在する．GMG 法は粗い格子の生成に空間離散化の情報が必要となる．一方，AMG 法は係数行列 A のみでよく，不規則メッシュ状の問題やメッシュそのものが存在しない問題まで適用できるため，一般に AMG 法のほうが対象とする問題の適用可能範囲が広く，よく用いられている．

3.2 2 段マルチグリッド法

まず，マルチグリッド法の出発点である 2 段グリッド法について説明する [25, 28]．解くべき方程式である式 (2.1) に対して，Jacobi 法や gauss-Seidel 法を数回適用して近似解 $\tilde{\boldsymbol{x}}$ が得られたとする．このとき，真の解 $\boldsymbol{x}^*$ と近似解との誤差 $\tilde{\boldsymbol{o}}$ と，近似解の残差 $\tilde{\boldsymbol{r}}$ は以下のよ

うに表すことができる．

$$A\tilde{\boldsymbol{o}} = A(\boldsymbol{x}^* - \tilde{\boldsymbol{x}}) = A\boldsymbol{x} - A\tilde{\boldsymbol{x}} = b - A\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{r}}$$

$\boldsymbol{x}^* = \tilde{\boldsymbol{x}} + \tilde{\boldsymbol{o}}$ であるため，$\tilde{\boldsymbol{o}}$ を求めることができれば，近似解 $\tilde{\boldsymbol{x}}$ を $\tilde{\boldsymbol{o}}$ で修正することにより，真の解 $\boldsymbol{x}^*$ を得ることができる．よって，$\tilde{\boldsymbol{o}}$ を効率的に求める方法を考える．

Jacobi 法や Gauss-Seidel 法のような定常反復法を用いることで，高周波成分は効率よく減衰させることが可能である．そこで，これらの解法を適用した際の誤差 $\tilde{\boldsymbol{o}}$ は，高周波成分が十分減衰したものとする．このとき，$\tilde{\boldsymbol{o}}$ は減衰しにくい低周波成分が主となる．つまり，波数が小さい誤差成分が優位である．そこで，この残りの低周波成分の減衰のため，刻み幅を粗くした粗い格子空間上に誤差成分を射影し，ここで得られた解を用いて，誤差 $\tilde{\boldsymbol{o}}$ を補正することを考える．このように補正することで，波数が小さい誤差成分を効率よく減衰させることが可能となり，誤差 $\tilde{\boldsymbol{o}}$ を大幅に減衰させることができると期待できる．

2 段マルチグリッド法の具体例を，モデル問題を用いて説明する．モデル問題として，ここでは単位正方形領域 $\Omega = [0,1] \times [0,1] = \{(x,y) | 0 \leq x \leq 1, 0 \leq y \leq 1\}$ における Dirichlet 問題を考える．つまり、Ω の境界 Γ 上で定義された関数 $g(x,y)$ が与えられたとき，Γ の内部で Laplace 方程式

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

を満たし，境界 Γ 上で Dirichlet 境界条件

$$u(x,y) = g(x,y) \qquad (x,y \in \Gamma)$$

を満たす関数 $u(x,y)$ を求める問題を考える．これに対し，$u(ih,jh)$ に対する近似解を $u_{i,j}$ とし，刻み幅 $h = 1/N$ で中心差分を適用すると，偏導関数を

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}, \quad \frac{\partial^2 u}{\partial y^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}$$

のように近似することができる．これにより，$u_{i,j} (1 \leq i,j \leq N-1)$ に関する連立一次方程式

$$4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = 0$$

が得られる．ただし，境界条件より，

$$u_{0,j} = g(0,jh), \quad u_{N,j} = g(1,jh), \quad u_{i,0} = g(ih,0), \quad u_{i,N} = g(ih,1)$$

となる．これは，$\boldsymbol{u}_h = (u^h_{i,j} | 1 \leq i,j \leq N-1)$ を未知ベクトルとする連立一次方程式 $A_h \boldsymbol{u}_h = \boldsymbol{b}_h$ とみることができる．ここで，I_N を N 次の単位行列，B_N を N 次の以下のよう

な行列

$$B = \begin{bmatrix} 0 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 0 \end{bmatrix}$$

と定義すると，テンソル積 $\otimes$ を用いることで，

$$A^h = \frac{1}{h^2}(4I_{N-1} \otimes I_{N-1} - B_{N-1} \otimes I_{N-1} - I_{N-1} \otimes B_{N-1})$$

と表現できる．また，

$$\begin{aligned} \boldsymbol{b}_h = \frac{1}{h^2} \Bigg\{ & \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \otimes \begin{bmatrix} g(0,h) \\ g(0,2h) \\ \vdots \\ g(0,(N-1)h) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \otimes \begin{bmatrix} g(1,h) \\ g(1,2h) \\ \vdots \\ g(1,(N-1)h) \end{bmatrix} \\ & + \begin{bmatrix} g(h,0) \\ g(2h,0) \\ \vdots \\ g((N-1)h,0) \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} g(h,1) \\ g(2h,1) \\ \vdots \\ g((N-1)h,1) \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \Bigg\} \end{aligned} \tag{3.1}$$

となる．以上のモデル問題を用いて、細かい格子から粗い格子を作成することを考える．この例では単純に，刻み幅 h を $2h$ の格子に粗くすることを考える．このような例では，一般に近傍の 9 点に対して図 3.1 のように重みをつけ，

$$u_{i,j}^{2h} = \frac{1}{4}u_{2i,2j}^h + \frac{1}{8}\sum_{p=\pm1} u_{2i+p,2j}^h + \frac{1}{8}\sum_{p=\pm1} u_{2i,2j+q}^h + \frac{1}{16}\sum_{p,q=\pm1} u_{2i+p,2j}^h$$

とすることが多い．このように，細かい格子から粗い格子を作成する手順を，粗い格子への制限（Restriction）と呼ぶ．逆に，粗い格子から細かい格子へ戻す際には，一般に

$$\begin{aligned} u_{2i,2j}^h &= u_{i,j}^{2h}, \\ u_{2i+1,2j}^h &= \frac{1}{2}\left(u_{ij}^{2h} + u_{i+1,j}^{2h}\right), \\ u_{2i,2j+1}^h &= \frac{1}{2}\left(u_{ij}^{2h} + u_{i,j+1}^{2h}\right), \\ u_{2i+1,2j+1}^h &= \frac{1}{4}\left(u_{ij}^{2h} + u_{i+1,j}^{2h} + u_{i,j+1}^{2h} + u_{i+1,j+1}^{2h}\right) \end{aligned}$$

にように行われることが多い．このような手順を延長（Prolongation）と呼ぶ．より一般には，制限においては横長の Restriction 行列 R を定め $\boldsymbol{u}_{2h} = R_h\boldsymbol{u}_h$ とし，延長においては縦長の Prolongation 行列を定め $\boldsymbol{u}_h = P_h\boldsymbol{u}_{2h}$ とする．

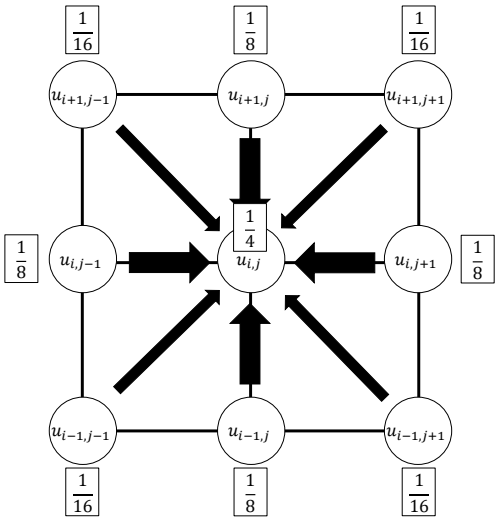


図 3.1: モデル問題における制約演算の例 [25, 28]

以上をまとめたものを Algorithm 3 に示す．Algorithm 3 内において $\boldsymbol{x}_1$ のように添え字に 1 が示されているものは，最上層におけるベクトルおよび行列を示し，$\boldsymbol{x}_2$ は最下層におけるものを示している．最初と最後において反復法を適用しているが，これは波数が大きい誤差成分を減衰させることを目的としたものである．このような動作をスムージングと呼び，スムージングを行う反復法をスムーザと呼ぶ．マルチグリッド法では，最初に行われるスムージングを Pre-smoothing，後に行われるスムージングを Post-smoothing と呼ぶ．Pre-smoothing は初期の誤差ベクトルに含まれる波数が大きい誤差成分を減衰させるために行い，Post-smoothing は粗いレベルから細かいレベルへの写像の際に混入した誤差成分を減衰させるために行われる．また，最下層において方程式 $A_2\boldsymbol{x}_2 = \boldsymbol{b}_2$ を解いているが，最下層では格子数が少なくなるため，直接法を用いて厳密解をそのまま求めることが多い．

Algorithm 3 2 段マルチグリッド法

Pre-smoothing: $\tilde{\boldsymbol{x}}_1 \leftarrow S(\boldsymbol{x}_1, \boldsymbol{b}_1)$
Coarse grid correction:
$\boldsymbol{r}_1 \leftarrow \boldsymbol{b_1} - A_1\ \tilde{\boldsymbol{x}}_1$
$\boldsymbol{b}_2 \leftarrow R_1\ \boldsymbol{r}_1$
$A_2\boldsymbol{x}_2 = \boldsymbol{b}_2$ を $\boldsymbol{x}_2$ について解く
$\boldsymbol{x}_1 \leftarrow \boldsymbol{x_1} + P_1\ \boldsymbol{x}_2$
Post-smoothing: $\boldsymbol{x}_1 \leftarrow S(\boldsymbol{x}_1, \boldsymbol{b}_1)$

$S(\boldsymbol{x}, \boldsymbol{b})$: $A\boldsymbol{x}= \boldsymbol{b}$ に対する反復法の適用（スムーザ）

3.3 Multigrid 法

本節では，より実用的である Multigrid 法について述べる．2 段グリッド法では，用いる格子が 2 段のみとなるため，格子数が大きい場合，粗いレベルにおける直接法による求解には依然として格子数が十分に小さくならず，膨大な時間が必要となる可能性がある．さらに 2 段のみでは，より波数の小さい誤差成分の減衰を十分に行えない可能性がある．そこで，2 段グリッド法における $A_2\boldsymbol{x_2} = \boldsymbol{b_2}$ を解く際に，再帰的に 2 段グリッド法を適用することを考える．これを Multigrid 法と呼ぶ．Algorithm 4 に Multigrid 法の計算方法のひとつである V-cycle についての概要を示す．$\boldsymbol{x}_l$ のように添え字に l がついているものは，レベル l におけるベクトルや行列を意味している．階層移動（Coarse grid correction の箇所）では，構築部で作成された補間演算子の Prolongation 行列と Restriction 行列を使う．階層を下りる際には，現階層の残差を計算し，横長の Restriction 行列と行列ベクトル積を行うことで短いベクトルを生成し，ひとつ下の階層で利用する．階層を上る際は，現階層の解と縦長の Prolongation 行列との行列ベクトル積を行うことで長いベクトルを生成し，ひとつ上の階層の補正解として利用する．Multigrid 法を解法として用いる際には，Algorithm 4 の計算手順をマルチグリッド法の 1 反復とし，これを期待する精度の近似解が得られるまで繰り返す．図 3.2 に，以上の Multigrid 法における V-cycle の流れをまとめたものを図示する．この図のように，複数の階層を行き来する様子が V 字を連想させるため，V-cycle と呼ばれている．

Algorithm 4 Multigrid 法（V-cycle）

Pre-smoothing: $\tilde{\boldsymbol{x}}_l \leftarrow S_l(\boldsymbol{x}_l, \boldsymbol{b}_l)$
Coarse grid correction:
$\boldsymbol{r}_l \leftarrow \boldsymbol{b}_l - A_l\, \tilde{\boldsymbol{x}}_l$
$\boldsymbol{b}_{l+1} \leftarrow R_l\, \boldsymbol{r}_l$
if $l+1 = L$ **then**
　$A_L\, \boldsymbol{x}_L = \boldsymbol{b}_L$ を直接法（例；LU 分解）で解く．
else
　$l+1$ において $\boldsymbol{x}_{l+1} = 0$ とし Pre-smoothing から計算．
end if
$\boldsymbol{x}_l \leftarrow \boldsymbol{x}_l + P_l\, \boldsymbol{x}_{l+1}$
Post-smoothing: $\boldsymbol{x}_l \leftarrow S_l(\boldsymbol{x}_l, \boldsymbol{b}_l)$

$S(\boldsymbol{x}, \boldsymbol{b})$: $A\boldsymbol{x} = \boldsymbol{b}$ に対するスムーザの適用
$l = 1, 2, ..., L$: レベル

Algorithm 4 では V-cycle の例を示したが，再帰の方法により，他に W-cycle や FMG(Full Multigrid V-cycle) scheme と呼ばれるものが存在する．それぞれの手法についての例を示したものを図 3.3 に示す．図 3.3 からわかるように，W-cycle や FMG scheme は V-cycle と比べ強力な手法ではあるが，計算コストが増大する．本研究では，単純で分析が W-cycle や FMG scheme よりも比較的容易な V-cycle を用いている．

以上のように，Multigrid 法は始めに階層構造を生成し，その後 V-cycle 等を用いて与え

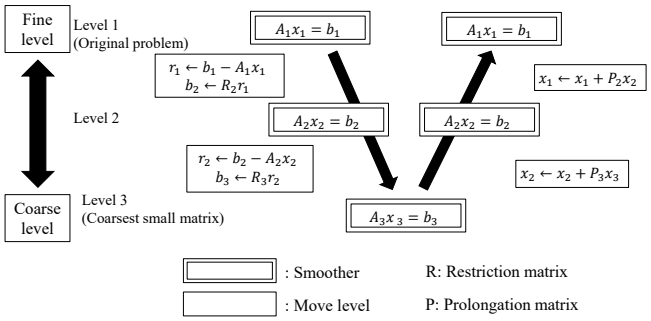


図 3.2: V-cycle の流れ

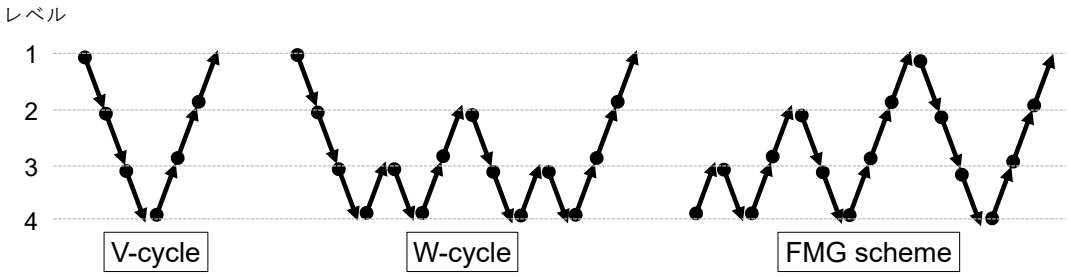


図 3.3: Multigrid 法における様々な計算方法（V-cycle，W-cycle，FMG scheme）

られた連立一次方程式を解く．階層構造を作成する処理を構築部（Setup part），V-cycle を用いて実際に連立一次方程式を解く処理を求解部（Solution part）と呼ぶ．上記ではモデル問題を対象に，粗い格子を生成する方法を示したが，3.1 節でも述べたように，Multigrid 法は粗い格子の生成方法により GMG 法と AMG 法が存在する．一般に AMG 法のほうが対象とする問題の適用可能範囲が広く，そのため本研究では，AMG 法を対象とし研究を行っている．AMG 法の中でも，粗い格子の生成方法により，さらに様々な解法が存在する [5–12]．本研究では，その中で特に SA-AMG 法に着目して研究を行っている [14–16]．次節より，AMG 法についてより詳細な説明を行い，4 章にて SA-AMG 法についての説明を行う．

3.4 前処理としての Multigrid 法

Multigrid 法は，低周波な誤差成分を効率よく減衰させることができる有用な解法であり，その性質を利用し CG 法をはじめとした Krylov 部分空間法の前処理として利用されることが多い．本研究においても，2.3 章において示した前処理付 CG 法の前処理として，Multigrdi 法を適用している．

前処理の目的は，条件数の改善，つまり最大固有値と最小固有値の比を 1 に近づけることであった．そのため，前処理付 CG 法の収束性能は，前処理の性能によって大きく左右される．本章では収束性能の分析のため，Multigrid 前処理の前処理行列を構築し固有値解析を行うことで，Multigrid 前処理を適用することによる固有値の変化を分析する方法について述べる [29, 30]．

簡単な例として，2 段グリッド法において，緩和法に減速 Jacobi 法を 1 回適用することを考える．また，初期ベクトルは 0 ベクトルを用いるとする．減速 Jacobi 法は 2.1 章でも述べたように，

$$\tilde{\boldsymbol{x}} = L\boldsymbol{x} + N\boldsymbol{b}$$

のような形で表すことができる．また，Algorithm 3 における最下層においての「$A_2\boldsymbol{x}_2 = \boldsymbol{b}_2$ を解く」箇所においては，直接法で解いたと仮定する。つまり，最下層では，

$$\boldsymbol{x}_2 = A_2^{-1}\boldsymbol{b}_2$$

とする．これらの仮定を用いて，2 段グリッド法の処理をまとめると，図 3.4 のように導くことができる．ここで，2.1 章の Algorithm 2 からわかるように，前処理では $M\boldsymbol{z} = \boldsymbol{r}$ を解いている．言い換えると，$\boldsymbol{z} = M^{-1}\boldsymbol{r}$ を解いていることと同義である．これと図 3.4 にて最終的に得られた式を組み合わせると，以下を得ることができる．

$$M^{-1} = L\{(I - P_2A_2^{-1}R_2A_1)N + P_2A_2^{-1}R_2\} + N \tag{3.2}$$

2.1 章で述べたように，前処理付 CG 法では，$(C^{-1}AC^{-T})$ のような形で前処理行列を考える．そのため，実際に前処理適用後の固有値分布を調べたい場合は，式 (3.2) で求めた M^{-1} に対し，Cholesky 分解を適用して，

$$M^{-1} = C^{-1}C^{-T}$$

のように分解を行う．その後，$(C^{-1}AC^{-T})$ を求めてから固有値計算法を適用することで，固有値分布の分析を行う．

Multigrid 法を前処理とした CG 法における収束性のスケーラブル性については，単純な問題設定で限られた条件の下において数学的に示されている（例えば，[31] の Lemma 7.2.2 においては，1 次元 Poisson 問題に対し，2 レベルのグリッドにおいて最下層で補正解を厳密に解くという条件の下で示されている）．しかし，他の条件では数学的解析が困難であり，数値計算結果のみによる裏付けとなっている．

上記で説明した前処理行列構築手法を用いて，Multigrid 法を前処理として適用することによる固有値分布の変化を，テスト問題を用いて示す．図 3.5 に，2 次元熱拡散問題に対して固有値分布の分析を行った結果を示す．このテスト問題は等方な問題であり，問題サイズを 40^2 に設定した，単純な問題設定となっている．図 3.5 から明らかであるように，Multigrid 前処理を適用することで，すべての固有値が 1 に近づき，条件数が改善されていることがわかる．ちなみに，対称 Gauss-Seidel 前処理付 CG 法と，対称 Gauss-Seidel 法をスムーザとして用いた SA-AMG 前処理付 CG 法における収束までの反復回数を比較すると，前者は 15 反復であるのに対し、後者は 5 反復という結果となった．このように，Multigrid 法を用いることによる固有値分布の改善で，実際に収束性の改善効果もあることもわかる．

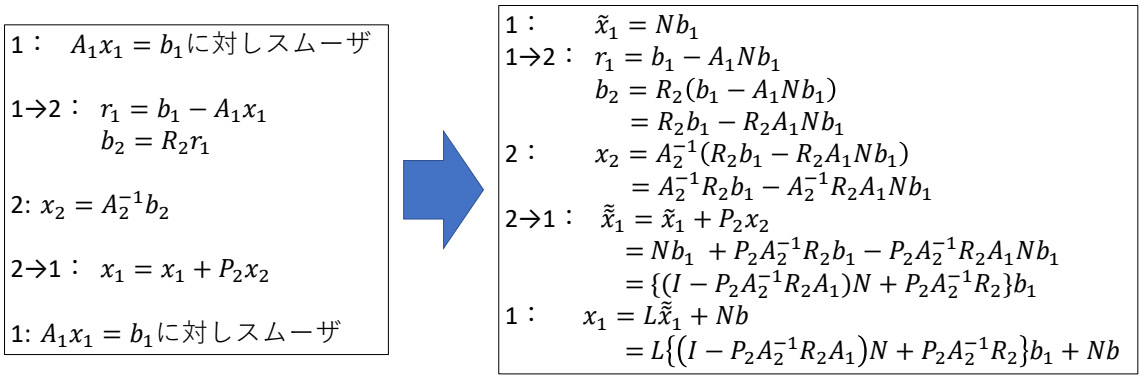


図 3.4: Multigrid 法の前処理行列の構築

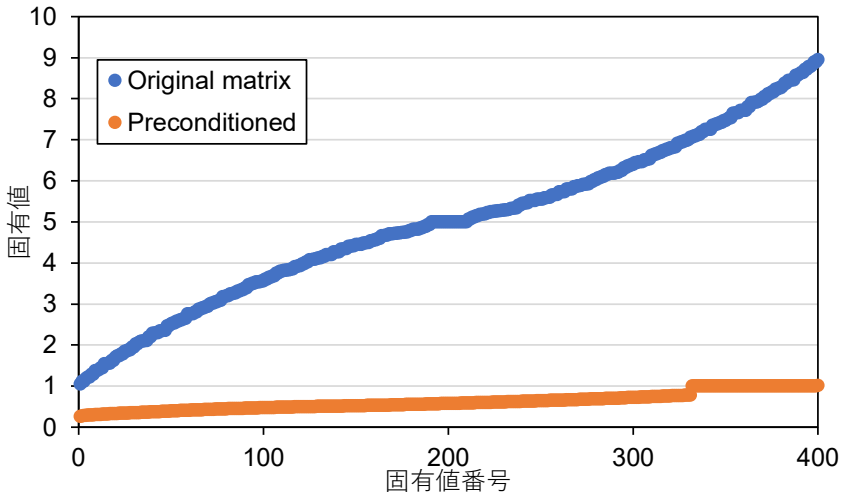


図 3.5: 元の係数行列（Original matrix）と前処理適用後の行列（Preconditioned）の固有値分布（条件数：Original matrix ⋯ 8.57，Preconditioned ⋯ 3.84）

3.5 代数的マルチグリッド（Algebraic Multigrid, AMG) 法

AMG 法は上記でも述べた通り，一般の連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ の係数行列 A の情報のみに基づき粗い格子を生成する Multigrid 法である．以下より，AMG 法についての説明を行う．

3.3 節における Multigrid 法では，モデル問題における空間の格子情報を用いて階層構造の作成を行った．これは，メッシュサイズと同じサイズの誤差が早く減衰し（空間的高周波成分），長い波長の誤差は減衰しにくい（空間的低周波成分）という性質を用いて，様々な誤差成分を減衰させるためであった．AMG 法では，そのような格子情報が存在しないという状況下において，高周波成分や低周波成分がどのような成分であるかを考える．3.3 節において述べたように，定常反復法を複数回適用することで，高周波な誤差成分は効率よく減衰し，低周波な誤差成分が残るという現象がある．これを基に，2.1 節で述べた代数的に

滑らかな成分を，低周波成分ととらえることとする．そして，代数的に滑らかな成分を用いて，粗い格子を決定することを考える．

代数的に滑らかな成分は，より具体的には，

$$(A\boldsymbol{e}, D^{-1}A\boldsymbol{e}) << (\boldsymbol{e}, A\boldsymbol{e}) \tag{3.3}$$

のような関係が成り立つベクトル $\boldsymbol{e}$ であることが知られている．これを変形すると，

$$\begin{aligned}(A\boldsymbol{e}, \boldsymbol{e}) &= (D^{-1/2}A^{1/2}A^{1/2}\boldsymbol{e}, D^{1/2}\boldsymbol{e}) = ((D^{-1}A)^{1/2}A^{1/2}\boldsymbol{e}, D^{1/2}\boldsymbol{e}) \\ &\leq ((D^{-1}A)^{1/2}A^{1/2}\boldsymbol{e}, (D^{-1}A)^{1/2}A^{1/2}\boldsymbol{e})^{1/2}(D^{1/2}\boldsymbol{e}, D^{1/2}\boldsymbol{e})^{1/2} \\ &= (A\boldsymbol{e}, D^{-1}A\boldsymbol{e})^{1/2}(\boldsymbol{e}, D\boldsymbol{e})^{1/2}\end{aligned}$$

が得られる（Cauchy-Schwarz の不等式）．式 (3.3) より，

$$(\boldsymbol{e}, A\boldsymbol{e}) << (\boldsymbol{e}, D\boldsymbol{e}) \tag{3.4}$$

であることがわかる．ここで，行列やベクトルの成分で考える．a_{ij} を係数行列 A の i 行 j 列目，$\boldsymbol{e}_i$ をベクトル $\boldsymbol{e}$ の i 番目の成分とすると，式 (3.4) の左辺は以下のように変形できる．

$$\begin{aligned}(\boldsymbol{e}, A\boldsymbol{e}) = \sum_i \sum_j a_{ij}e_i e_j &= \sum_i \sum_j \frac{1}{2}\left\{(-a_{ij})(e_i - e_j)^2 + a_{ij}e_i^2 + a_{ij}e_j^2\right\} \\ &= \sum_i \sum_{j\neq i} \frac{1}{2}(-a_{ij})(e_i - e_j)^2 + \sum_i (\sum_j a_{ij})e_i^2\end{aligned} \tag{3.5}$$

ここで，3.2 章にて示したモデル問題で考えると，$j \neq i$ のとき $a_{ij} \leq 0$ であり，かつ式 (3.5) の最後の式の第 2 項は非負である．その上で，式 (3.4) より，

$$\sum_j \frac{1}{2}|a_{ij}|(e_i - e_j)^2 << a_{ii}e_i^2$$

となり，最終的に以下を得ることができる．

$$\sum_j \frac{|a_{ij}|}{a_{ii}}\left(\frac{e_i - e_j}{e_i}\right)^2 << 1 \tag{3.6}$$

式 (3.6) の左辺は非負であるため，$|a_{ij}|$ は十分大きい値である場合，$e_i \approx e_j$ が成り立つこととなる．ここで，i 行目の要素において，値が比較的大きい非対角要素の列番号の集合を S_i とする．つまり，以下のような行番号の集合を定義する．

$$S_i = \{j | j \neq i, |a_{ij}| \geq \theta \max_{k\neq i}|a_{ik}|\} \tag{3.7}$$

式 (3.6) より，各 $j \in S_i$ に対して，$e_i \approx e_j$ が成り立つこともわかる．これより，代数的に

滑らかな成分は S_i の方向に緩やかに変化することがわかる．

次に，粗い格子の決定方法について考える．AMG 法には，様々な派生解法が存在する [32,33]．以下より，AMG 法の基本となる classical interpolation と呼ばれる手法の中の，Brandt らによる手法 [34–36] について説明する．粗い格子に対応するものとして，格子の節点番号の集合 $V = \{1, 2, ...\}$ の中における適当な望ましい性質を持つ部分集合を，粗い格子の節点番号の集合と考える．つまり，V における適当な部分集合 C を決定し，粗い格子（Coarse Grid）とする．逆に，粗い格子に選ばれなかった節点，つまり C の補集合 $F = V \backslash C$ を細かい格子（Fine Grid）とする．この仮定のもと，階層構造の作成に必要な補間行列である Prolongation 行列 P（$P : \mathbb{R}^{|C|} \to \mathbb{R}^{|C \cup F|}$，$|C|$ は集合 C の要素数）をうまく定め，代数的に滑らかな成分を C 上で近似できるように作成することを考える．ここで，補間行列 P を，

$$P_{ik} = \begin{cases} 1.0 & (i \in C, i = k) \\ w_{ik} & (i \in F, k \in C) \end{cases}$$

の形で定めるとする．ただし，w_{ik} は適当な重みである．また，近似解 u に含まれる代数的に滑らかな誤差 e を減衰させることを考えるとき，つまり適当な近似解 v を選んで新しい近似解 $\tilde{u} = u + Pv$ に含まれる誤差 $\tilde{e} = e + Pv$ を 0 にするためには，$e \in \mathrm{Im}P$ となる P を定める必要がある（ImP は P の像空間を示す）．これを踏まえ，補間行列 P の要素である重み w を決定する．こ k で，新たな集合 $C_i = C \cap S_i$，$F_i = F \cap S_i$ を定義する．重ねて，i 行目における非ゼロ要素の列番号の集合を $N_i = \{j | j \neq i, a_{ij} \neq 0\}$ とし，その中でも値が比較的小さい列番号の集合を $W_i = N_i \setminus S_i$ と定義する．式 (2.5) より，以下のように表すことができる．

$$a_{ii}e_i \approx -\sum_{j \in C_i} a_{ij}e_j - \sum_{k \in F_i} a_{ik}e_k - \sum_{l \in W_i} a_{il}e_l$$

W_i に含まれる要素は小さい値であることから，さらに以下のような近似式が得られる．

$$\left(a_{ii}e_i + \sum_{l \in W_i} a_{il}e_l \right) \approx -\sum_{j \in C_i} a_{ij}e_j - \sum_{k \in F_i} a_{ik}e_k \tag{3.8}$$

ここで，$e_k \in F_i$ は i 番目の要素に強く影響を与えているが，C に含まれる変数でないため，そのままでは補間行列 P に使用できない．そこで，$k(\in F_i)$ 行目の係数の大きさで重みづけを行い，平均をとることで，以下のように近似を行う．

$$e_k \approx \frac{\sum_{j \in C_i} a_{kj}e_j}{\sum_{j \in C_i} a_{kj}} \tag{3.9}$$

ここで，$e \in \mathrm{Im}P$，言い換えると $\boldsymbol{v} = \boldsymbol{e}|_C$ に対して $\boldsymbol{e} \approx P\boldsymbol{v}$ となる P を定めることが目的であった．これを加味しながら，式 (3.8) と式 (3.9) を用いて変形すると，$e_i \approx \sum_{j \in C_i} w_{ij}e_j$

となり，目的の形となる．ただし，

$$w_{ij} = -\frac{a_{ij} + \sum_{k \in F_i} \left(\frac{a_{ik} a_{kj}}{\sum_{m \in C_i} a_{km}} \right)}{a_{ii} + \sum_{l \in W_i} a_{il}}$$

である．なお，$\sum_{j \in C_i} a_{kj} \approx 0$ とならないように,「$i, k \in F$ に対し，$k \in S_i$ ならば $C_i \cap C_k \neq \phi$」（ただし，ϕ は空集合を表す）と選ぶ必要がある．このように，集合 C と F の分離に関する制約が高く，集合の選択により収束性能に大きく影響を及ぼすことが知られている．

粗いレベルの問題を作成する際，3.2 章においては格子の情報を用いた．AMG 法では粗いレベルの行列を $\bar{A}$ とすると，$\bar{A} = RAP, R = P^T$ とするのが普通である．これは，新しい近似解 $\tilde{u} = u + Pv$ の残差が P の像空間と直交するようにするためである．このとき，A が正定値対称であれば，これは新しい近似解に含まれる誤差 $\tilde{e} = e + Pv$ を最小にする，つまり，

$$||\boldsymbol{e} + P\boldsymbol{v}||_A = \min_{w} ||\boldsymbol{e} + P\boldsymbol{w}||_A \tag{3.10}$$

となることが変分原理により確かめることができる [25,28]．ただし，$||w||_A = (Aw, w)^{1/2}$ である．

以上の AMG 法における階層構造の生成の流れ（構築部）の概要を図 3.6 に示す．上記で述べたように，構築部では細かいレベルの問題行列を基に，未知数間のグラフ構造を作り，次のレベルに残す未知数を選択する．そして，粗いレベルの問題行列や，階層移動の際に必要となる補間演算子 Prolongation（P）行列と Restriction（R）行列を生成する．これを再帰的に行うことで，階層的に行列を生成する．図 3.6 のように，1 レベル目は与えられた問題行列が置かれ，階層が下がるにつれて問題行列より小規模な行列を生成する．階層数は問題サイズによって可変となる．また，粗いレベルの問題行列は，横長の Restriction 行列と縦長の Prolongation 行列，さらに現階層の問題行列とで行列行列積（RAP）を行うことで作成する．

補間演算子から行列の階層構造が生成されるため，Multigrid 法では補間演算子の生成方法は重要となる．この補間演算子の生成手法により様々な AMG 法が存在する [5–12]．その中で，本研究では SA-AMG 法と呼ばれる手法を対象としている．この手法では，問題行列のみから未知数間の依存関係を定義する．そして，依存関係のある未知数同士で集合を作り，その集合内で重みづけをして補間演算子を生成する．その後，生成された補間演算子を基に，集約を行う．SA-AMG 法はさまざまな分野で利用されており，AMG 法の代表的な手法のひとつとなっている．

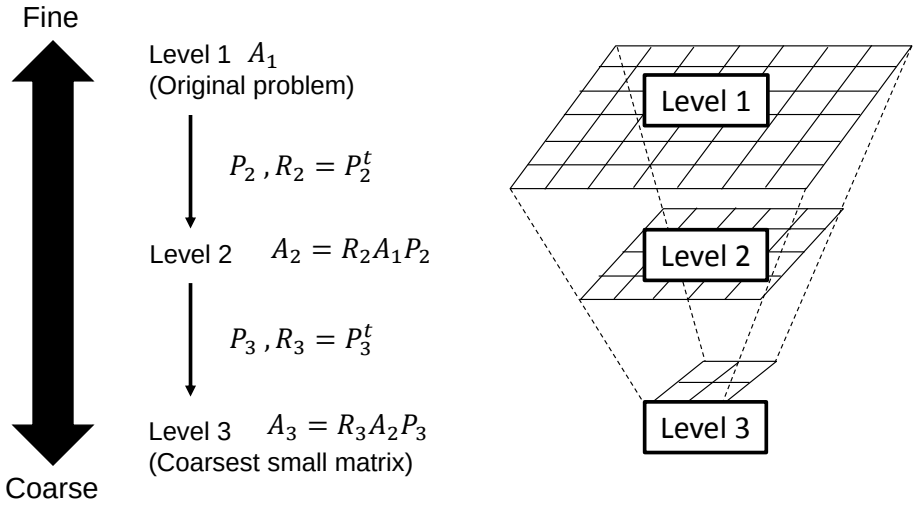

Fine
Level 1 A_1
(Original problem)
P_2 , $R_2 = P_2^t$
Level 2 $A_2 = R_2 A_1 P_2$
P_3 , $R_3 = P_3^t$
Level 3 $A_3 = R_3 A_2 P_3$
(Coarsest small matrix)
Coarse
Level 1
Level 2
Level 3
R: Restriction matrix
P: Prolongation matrix

図 3.6: 構築部の概要

第 4 章

SA-AMG 法

本章では，AMG 法の派生解法のひとつであり，本研究の対象である SA-AMG 法について説明する．まず，SA-AMG 法での階層行列や補間演算子の作成方法を説明し，その後本研究で用いている CGA（Coarse Grid Aggregation）と呼ばれる，並列実行時における階層行列生成手法について説明する．

4.1 SA-AMG 法の概略

3 章でも述べたように，SA-AMG（Smoothed Aggregation - AMG）法は，Multigrid 法の派生解法のひとつである．SA-AMG 法では，問題行列に基づく節点と辺で構成されたグラフ構造を用いて粗い問題を作成していく．ここで，問題行列の各行が節点に対応し，非ゼロ要素が辺に対応している．

3.5 節で述べたように，式 (3.7) が成り立つような節点集合，つまり非対角要素が比較的大きい要素は，代数的に滑らかな成分となる．一般に，式 (3.7) のような関係がある節点同士は，強連結である（Strong connection または Strongly coupled）と呼ばれる．式 (3.6) より，強連結な関係にある節点同士では，代数的に滑らかな誤差成分の変化が緩いという特徴があった．そこで SA-AMG 法では，粗い問題を作成する際に，節点全体をアグリゲートと呼ばれる強連結同士の節点集合に分解する．アグリゲートは図 4.1 のように，次の粗いレベルで 1 つの節点に対応し，グラフ構造である節点を中心に近くの節点をまとめた節点集合と定義される．そのため，アグリゲート数と次の粗いレベルの節点数は同じとなる．実際にアグリゲートを生成する際は，以下の式を用いて生成を行う．

$$N_i(\epsilon) = \{j : |A_{ij}| \geq \epsilon \sqrt{|A_{ii}||A_{jj}|}\} \cup \{i\} \tag{4.1}$$

ここで，N_i は i 番目のアグリゲート，A_{ij} は行列 A の i 行 j 番目の要素を示す．この式に示す通り，i 番目のアグリゲートは i 番目の要素に対応する節点かつ，絶対値が対角成分に対し比較的大きい非対角成分の要素に対応する節点で構成される．アグリゲート生成の全体の流れを Algorithm 5 に示す．まず Algorithm 5 の Step:1 では，式 (4.1) を用いて，アグリゲートの生成を行う処理となっている．ここで，アグリゲート生成では，補間の関係上すべての

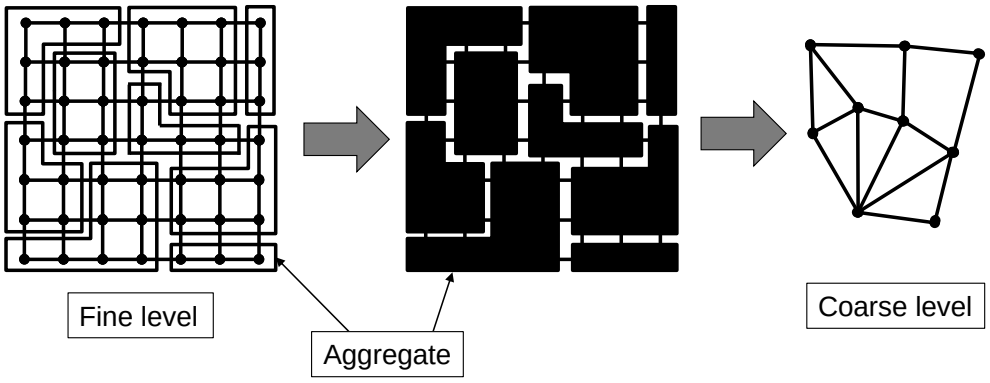


図 4.1: アグリゲート生成の概要

要素がどこかしらのアグリゲートに属していなければいけないという制約がある．そのため，任意の節点がどこか 1 つのアグリゲートに属するように，アグリゲートを生成する．Step:2 ではそのために，すでに生成されたアグリゲートと残った要素とで再度式 (4.1) を適用し，成立した場合アグリゲートに追加する．Step:3 では，Step:2 を経てもなお残る節点に対し，その節点をアグリゲートとするように処理を行う．このように，すべての節点がいずれかのアグリゲートに属するように，アグリゲートの生成を行う．その後，作成されたアグリゲート内の未知数に重み付けをすることで補間演算子である Prolongation 行列と Restriction 行列を計算し，行列の階層構造を作成する．最も単純な例として，以下のような生成方法があげられる．

$$\tilde{P}_{ij} = \begin{cases} 1 & (i \in C_j) \\ 0 & (otherwise) \end{cases} \tag{4.2}$$

生成された行列 $\tilde{P}$ をそのまま補間演算子として用いてもよいが，SA-AMG 法ではさらに収束性向上のため，$\tilde{P}$ に緩和法を 1 回適用する．一般に，減速 Jacobi 法が用いられることが多く，その場合，

$$P = (I - \omega D^{-1} A_l)\tilde{P}$$

のようにし，P を補間演算子として用いる．以上の補間演算子生成方法は対称な行列のみを想定している．非対称な行列においては，M. Sala らによる手法 [37] が提案されている．本研究ではこのことについては詳しい言及はせず，対称行列のみを用いる．

4.2 並列時におけるアグリゲート生成手法

本研究では並列 SA-AMG 法を対象としているが，アグリゲート生成は各節点で依存関係のある処理を行うため，並列実行時には工夫が必要となる．ここでは，プロセス並列を施すことを前提として考える．プロセス並列はメモリ分散型の並列化手法であるため，一般に，問題行列を分割し，計算単位であるプロセスにそれぞれの領域を分配してから計算する．領域分割された問題行列の領域上でアグリゲートを生成する手法は，独立アグリゲート生成手

Algorithm 5 SA-AMG 法におけるアグリゲート生成の手順

Step 1:
全節点の集合 $V = 1, 2, ...$ から，$N_i(\epsilon) \in V$ を計算.
その後，$j \leftarrow j+1, C_j \leftarrow N_i(\epsilon), V \leftarrow V \setminus C_j$ とし，新たな C_j が作成できなくなるまで繰り返す.

Step 2:
C_k のコピー $\tilde{C}_k$ を作成.（$\tilde{C}_k = C_k, k = 1, ..., j$）
もし $N_i(\epsilon) \cap \tilde{C}_k \neq \phi$ となる $i \in V$ が存在するなら，$C_k \leftarrow C_k \cup \{i\}$ とし，条件を満たす i が存在しなくなるまで繰り返す.

Step 3:
もし $i \in V$ が存在するなら，$j \leftarrow j+1, C_j = V \cap N_i(\epsilon)$ とし，$V = \phi$ となるまで繰り返す.

法と共有アグリゲート生成手法の 2 種類に分類される．独立アグリゲート生成手法と共有アグリゲート生成手法を図 4.2 に示す．4 つに区切られた正方形は各プロセス領域を示し，灰色の部分は領域境界であることを示す．独立アグリゲート生成手法とは，アグリゲートが各領域内に収まるように各領域独立にアグリゲートを生成する手法である．また，共有アグリゲート生成手法とは，アグリゲートが領域境界を跨いで生成される手法である．共有アグリゲート生成手法では，領域境界を跨いでアグリゲートを生成する際に，徐々に形状が崩れていくため，粗いグリッドが複雑な構造となる．一方，独立アグリゲート生成手法は，領域境界に沿ってアグリゲートを生成するため，共有アグリゲート生成手法よりも，粗いグリッドが領域分割の形状にそって生成される．

ただし，独立アグリゲート生成手法は，担当領域内に少なくとも 1 つの未知数要素を持たなければならない制約があるため，最も粗いレベルにおいて最低でも領域数の未知数要素が存在する必要がある．高並列時には，最も粗いレベルの未知数要素数をプロセス並列数以上とするため，十分に粗くできない場合が出てくる．また，最も粗いレベルの未知数の増加に伴い，求解部の処理時間が増加すると考えられる．共有アグリゲート生成手法は，まず境界付近からアグリゲートを生成し，その後に各領域内でアグリゲートを生成する．領域境界を跨いでアグリゲートが生成されるため，粗いレベルになるとアグリゲートを担当しない領域が出現することがある．この手法では，領域境界を跨いでアグリゲートが生成されるため，異方性のある問題に対してもよりよい収束性能を得ることができる [38].

以下より，具体的な実装方法について述べる．並列 SA-AMG 法では，各レベルにおいての緩和法による隣接プロセスへの通信と，レベル間においての他領域に跨った Prolongation 行列と Restriction 行列による通信が発生する．この通信は節点間の接続関係に基づいて行われる．

本研究は節点間の接続関係に基づく領域分割をしている．たとえば，図 4.3 はシンプルな 2 次元格子構造の領域分割例である．3 つの領域は異なるメモリ空間に配置される．これに対し緩和法を適用する場合，領域境界において節点の情報を通信により相互にやり取りする必要がある．通信テーブルは各領域に，SEND テーブルと RECV テーブルを持たせている．SEND テーブルは隣接領域の計算で参照される節点番号の集合であり，RECV テーブルは自領域の計算で参照する隣接領域の節点番号の集合である．

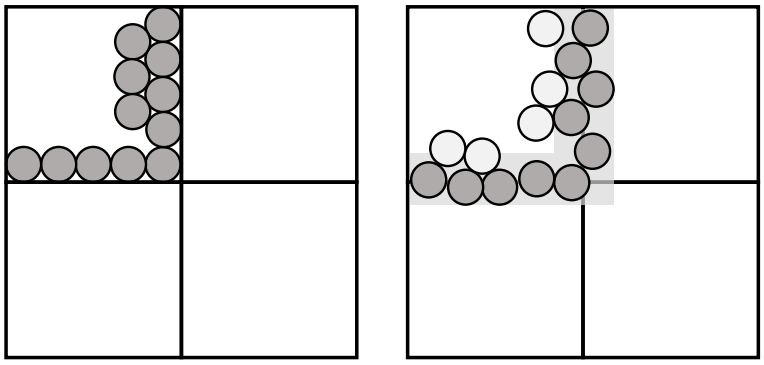


図 4.2: 独立アグリゲート生成手法と共有アグリゲート生成手法（左：独立アグリゲート生成手法，右：共有アグリゲート生成手法）

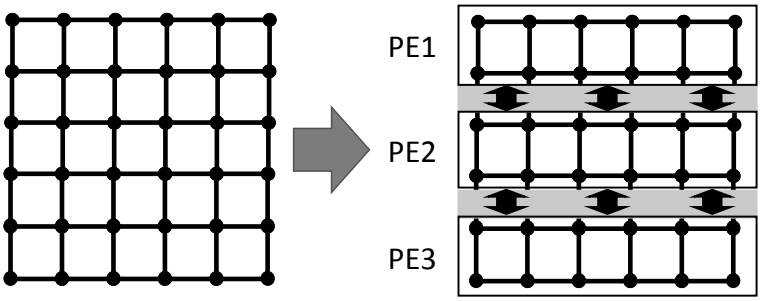


図 4.3: 2 次元格子問題と領域分割による通信（左：領域分割前，右：領域分割後）

また，階層移動の Restriction や Prolongation を行う場合，緩和法とは異なる領域間通信が必要となる．領域分割による並列 SA-AMG 法は，領域境界を跨がるような形でアグリゲートが存在する可能性がある．この場合，アグリゲートは跨っているどちらかの領域に所属することになり，領域境界では所属に応じて通信を行う．図 4.4 に Restriction と Prolongation の通信例を示す．簡略化のため，緩和法による通信を省略し，レベル間通信のみとしている．PE1 が Restriction を行う際，アグリゲートの一部の節点が PE2 の領域にあるが，Restriction はアグリゲートを用いて複数の節点を次の粗いレベルのひとつの節点にまとめる処理なので，PE2 から必要となる節点を受信する．また，Prolongation はひとつの節点を元のアグリゲートの節点に分散させる処理なので，PE2 へ必要となる節点を分散させる．

4.3 CGA（Coarse Grid Aggregation）

4.3.1 CGA の概要

本節では，独立アグリゲート生成手法を用いた SA-AMG 法において，粗いレベルの領域を適宜集約していく CGA と呼ばれる手法について述べる [1]（この手法は，以降の数値実験においても用いている）．共有アグリゲート生成手法と比べ，独立アグリゲート生成手法では，領域境界を跨がずに自領域内でアグリゲートの生成を行うため，プロセス間での節点

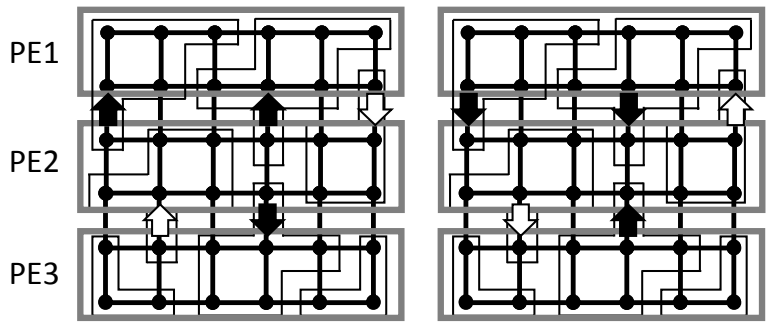
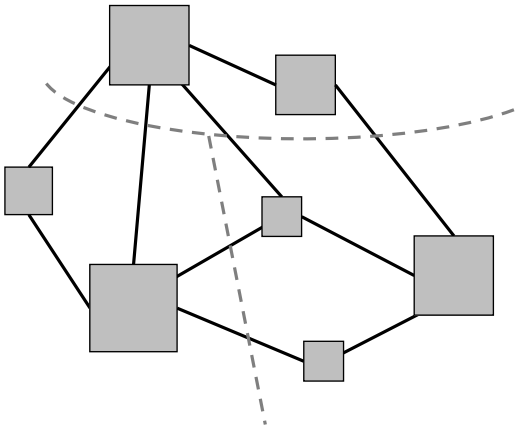


図 4.4: Restriction（左）と Prolongation（右）の通信



領域集約手法のアルゴリズム

1. 分割されたアグリゲートを示すグラフ構造を作成
2. ParMETISライブラリを利用して，プロセスの組み合わせを決定し，領域を集約する
3. ParMETISによる組み合わせにより集約されたプロセスは，粗いレベルのプロセスとして使用される

図 4.5: 領域集約手法の概要

情報の通信は行われない．そのため高並列環境下において，最下層での節点数がプロセス数以下にならず十分粗くならない．これにより，最下層の処理時間が増加する可能性がある．しかし CGA は，自プロセスが担当していたアグリゲートを，ほかのプロセスへ担当を変更する．これにより，領域を跨いでアグリゲートを作成することができるため，高並列環境下においても十分な粗さにすることができる．

図 4.5 に CGA の概要を示す．領域集約は，細かいレベルで複数プロセスの担当領域を組み合わせ，粗いレベルではその中の最もランク番号の低いプロセスに担当させることで実装する．そうすることで，アグリゲート生成後に複数領域の組み合わせを決定し，領域の集約を行い粗いレベルの並列度を落とすことができる．

現状では，複数プロセスの組み合わせは，各プロセスの平均アグリゲート数がなるべく一定数を保つことができるように実装を行っている．そのために，アグリゲートの総数，隣接プロセスの総数とランク番号の隣接情報を使用して，プロセス間のデータ分散状況を示す重みつきグラフを作成し，ParMETIS ライブラリを使用してプロセスの組み合わせを決定している．

Algorithm 6 に，SA-AMG 法のアルゴリズムに領域集約手法の手順を加えたアルゴリズムを示し，各手続きについて説明する．Algorithm 6 の (1) と (5)，および (6) はそれぞれ，

上記で説明したアグリゲート生成，生成したアグリゲートを基に補間演算子 P の生成，次の粗いレベルの行列生成となっている．(2) から (4) はそのうえで，CGA による領域集約のための手順を示したものとなっている．それぞれについては，以下に示すとおりとなっている．

（2）粗いレベルの並列度の決定

次の粗いレベルの並列度を決定する．Algorithm 7 に粗いレベルの並列度決定方法を示す．細かいレベルの各プロセスの平均アグリゲート数が，領域集約の閾値以下のときに粗いレベルで領域集約を行い，閾値よりも大きいときは領域集約を行わない（*coarser_lev_parallelism*=-1）．並列度の決定方法は，1 プロセスあたりの平均アグリゲート数が一定数（*Proc_aggre_threshold*）以上になるように，次レベルの並列数を決定している．

（3）領域の組み合わせの決定

（2）で求めた粗いレベルの並列数より，プロセスの組み合わせを決定し，プロセスの集約を行う．各プロセスの平均アグリゲート数が一定となるように，アグリゲートの総数，隣接プロセスの総数とランク番号の隣接情報を使用して重みつきグラフを作成する．そして ParMETIS ライブラリを使用して，グラフを分割することにより領域の組み合わせが決定され，領域の集約が行われる．ここで，各プロセスの平均アグリゲート数を一定数保ち，隣接プロセス数が最小となるように ParMETIS ライブラリを利用する．

プロセス間でアグリゲートを集約するときは，ランク番号が最小であるプロセスがすべてのアグリゲートを引き継ぐ．また集約する際，アグリゲートはランク番号順に順次割り当てる．

（4）アグリゲートテーブル等の調節

（3）の結果により，プロセスランク番号順になるようにアグリゲート番号の付け替えを行い，各プロセスに配分する．

Algorithm 6 CGA を加えた SA-AMG 法のアルゴリズム

for $l = 1$ to L **do**
  $\tilde{P}_l = aggregation(A_l)$ ⋯ (1)
  !- Coarse Grid Aggregation -!
  call decide_parallelism() ⋯ (2)
  call decide_process_combination() ⋯ (3)
  call set_GIN_aggre_and_aggre_tbl() ⋯ (4)
  !————————————!
  $P_l = smooth(\tilde{P}_l)$ ⋯ (5)
  $A_{l+1} = R_l A_l P_l$ ⋯ (6)
end for

Algorithm 7 粗いレベルの並列度決定方法

if $(All\_aggre\_size/Proc\_num) > Proc\_aggre\_threshold$ **then**
  $coarser\_lev\_parallelism = -1$
else
  $coarser\_lev\_parallelism = All\_aggre\_size/Proc\_aggre\_threshold + 1$
end if

All_aggre_size：アグリゲートの総数
$Proc\_num$：実行プロセス数
$coarser\_lev\_parallelism$：粗いレベルの並列数
$Proc\_aggre\_threshold$：領域集約の閾値のパラメタ

4.3.2 実験環境と数値実験 [1]

本節では，CGA について行った実験について述べる． 実験環境には，東京大学が運用している Fujitsu PRIMEHPC FX10（Oakleaf-FX）スーパーコンピュータシステム（以降，FX10）を使用して実験を行った．FX10 は，1 ノードに 1 個の SPARC64IXfx プロセッサ（16cores, 1,848GHz）を採用している．数値実験では，最大 1440 ノードを使用し計測を行った．並列化手法については，プロセス並列とスレッド並列を組み合わせた Hybrid 並列を使用した．FX10 の各ノードは 16 コア搭載されているため，各ノードにおける MPI プロセス数と各プロセスのスレッド数の積が 16 となるように設定されている．実験 1 での Hybrid 並列の実装に関しては，OpenMP/MPI を用いて実装を行っており，1 ノード内に 1 つのプロセスと 16 のスレッドを起動し，並列実行している．計算対象となる問題は，立方体領域における上下，左右，斜めの 27 点参照となる 3 次元 Poisson 方程式の，Darcy の流れ問題から導出される，拡散係数が不連続に変化する問題を対象とした [39]．問題サイズは 300×300×300 の問題として計測を行った．問題行列の各プロセスへの分割には ParMETIS を使用し，領域間の通信量が最小となるように領域分割を行っている [40]．ParMETIS は，グラフ分割や疎行列のオーダリングのための様々なアルゴリズムをもつライブラリである．

まず，CGA の閾値の変化による実行時間への影響を示したグラフを図 4.6 に示す．ここで，閾値とは，Algorithm 7 での $Proc\_aggre\_threshold$ のことを示し，図 4.6 ではパラメータ C として示している．図 4.6 より，768 ノードまでは C を 100 と設定したときの結果が最もよいことがわかる．しかし，1440 では C を 500 と設定したときが最もよくなることもわかる．これは，本実験はウィークスケーリングであり，使用ノード数増加とともに問題サイズが大きくなる．さらに閾値を小さく設定すると，レベル数が増加する傾向があるため，C を 1000 と設定したときに，レベル数増加に伴う計算コストの増加が大きくなってしまったためであると考えられる．

次に，GGA と同様の並列時におけるアグリゲート生成手法である共有アグリゲート生成手法と比較を行った際の結果を図 4.7 と表 4.1 に示す．図 4.7 より，CGA を用いた手法のほうが共有アグリゲートに比べ，すべての並列度において有用であることがわかる．これは，表 4.1 より，CGA の反復回数が共有アグリゲートに比べ低く抑えられているからであることがわかる．これより，CGA によるアグリゲート生成手法は，共有アグリゲート生成

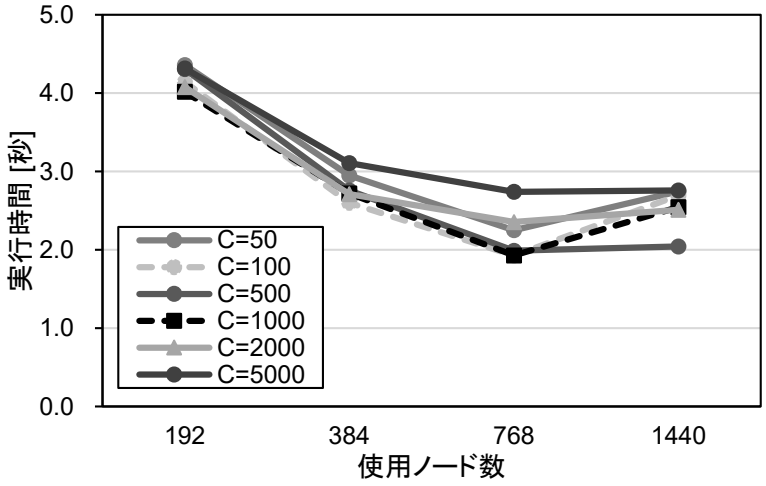


図 4.6: 閾値の変化による実行時間の推移（C：CGA のパラメータの値）[1]

表 4.1: CGA と共有アグリゲート生成手法との反復回数の比較 [1]

ノード数	共有アグリゲート	集約あり（C=1000）
192	19	16
384	21	17
768	20	16
1440	26	15

手法と比べ，反復回数と実行時間双方で有用であることがわかった．

4.3.3 関連研究

関連研究としては，中島らにより，幾何的多重格子法において粗いレベルの問題領域を組み合わせることも含めて，最適なパラメタ設定についての研究が行われている [41]．また，H.Sundar らにより，未知数の個数が多い（細かい）レベルでは，幾何的多重格子法を利用し，粗いレベルでは代数的多重格子法を適用することで，粗いレベルの 1 行あたりの非ゼロ要素数の増加を抑え，ストロングスケーリングの性能を向上させる手法を提案していた [42]．さらに，M.Adams らにより，実装手法は明確ではないが，問題行列を一部持つプロセスは，1 プロセスあたりの問題行列が一定サイズ以上になるように行列を再分散していたが，高並列時におけるストロングスケーリングの性能評価は行われていない [43]． 並列時におけるアグリゲート生成手法として, 他にグラフ理論で用いられている最大独立点集合という点集合を用いて粗いレベルの問題を作成する手法もある [44]．最大独立点集合は，各節点が隣接していないような最大の点集合のことである．この手法は並列化が容易であり，並列環境下における SA-AMG 法のアグリゲート生成に用いられることが多い [45]．また，より単純な方法として，Handshaking による Graph Matching がある [46]．この手法ではまず，各節点の強連結な辺を計算し，その後お互いに強連結と判断した節点同士を節点集合とする手法で

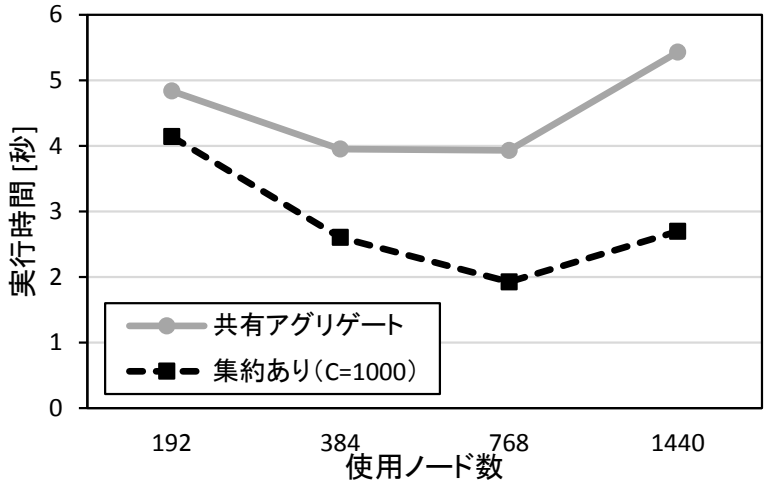


図 4.7: CGA と共有アグリゲート生成手法との比較（共有アグリゲート：共有アグリゲート生成手法，集約あり（C=1000）：CGA のパラメータを 1000 に設定）[1]

ある．この手法は AMG 法のライブラリである AmgX [47] に用いられている．

第 5 章

ニアカーネルベクトル抽出のための新手法の提案

本章は本研究の中核となるニアカーネルベクトル抽出手法の提案を行い，有用性を示す．本研究の手法は連立一次方程式 $A\boldsymbol{x} = \boldsymbol{b}$ を解く前段階において，あらかじめニアカーネルベクトルを抽出することで，実際に連立一次方程式を解く際に高い収束性を実現することを目的としている．より具体的に説明すると，3.3 節において，Multigrid 法は構築部と求解部に分かれると述べたが，本研究の提案手法を用いる際には、その前段階として，ニアカーネルベクトルを抽出する処理（以下より，本論文中では抽出部と表記する）が行われる．つまり，以下のような一連の流れにおいて，連立一次方程式を解く．

1. 抽出部：ニアカーネルベクトルを与えられた問題行列から抽出する
2. 構築部：抽出されたニアカーネルベクトルを用いて，後述の 5.1.2 節で述べる方法で補間演算子と粗いレベルを生成
3. 求解部：構築部で生成された階層行列を用いて，連立一次方程式を解く

このように，本研究で提案する手法を用いる際には，ニアカーネルベクトルを抽出する抽出部と，抽出されたベクトルを用いて連立一次方程式を解いていく構築部と求解部とで大まかにわかれ，提案手法は下線で示した抽出部に着目した手法となる．本章ではまず，ニアカーネルベクトルの概要を述べ，ニアカーネルベクトルを SA-AMG 法に用いることによる有用性を，数値実験を交え述べる．その後，著者らによる先行研究と本研究で提案するニアカーネルベクトル抽出手法について述べる．本研究では 3 つの抽出手法を提案しており，以下にそれぞれの概要を示す．

- 提案手法 1：各階層において V-cycle を適用し，各階層それぞれのニアカーネルベクトルの抽出を行う手法
- 提案手法 2：適切な抽出本数を抽出の時点で予測する手法を導入した抽出手法
- 提案手法 3：任意の粗いレベルにおいて固有値計算法を適用，補間を行うことでニアカーネルベクトルの抽出を行う手法

その後，他問題における有用性の検証を行うため，Texas A&M 大学の SuiteSparse Matrix Collection [20] から取得した行列を用い，実験を行った結果を示す．

5.1 ニアカーネルベクトルとは

5.1.1 ニアカーネルベクトルの概要

3.5 節において，式 (2.3) より求まる，定常反復法の適用で残差が停滞するような収束しにくい成分 $\boldsymbol{e}$ は，$A\boldsymbol{e} \approx 0$ となることを示した．この成分は係数行列 A との積が 0 に近くなることから，別の呼び方としてニアカーネルベクトル（Near-kernel vector），または Near-kernel error や Near-null vector などとも呼ばれる．AMG 法では，補間演算子である Prolongation 行列（Restriction 行列）を適切に選ぶことで，ニアカーネル成分を効率よく減衰できる．具体的には，3.5 節にて述べたように，適当な近似解 v を選んで新しい近似解 $\tilde{\boldsymbol{u}} = \boldsymbol{u} + P\boldsymbol{v}$ に含まれる誤差 $\tilde{\boldsymbol{e}} = \boldsymbol{e} + P\boldsymbol{v}$ を 0 にすることを考えた際，適当な条件下において，式 3.10 のような最小化（$||\boldsymbol{e} + P\boldsymbol{v}||_A = \min_w ||\boldsymbol{e} + P\boldsymbol{w}||_A$）が行われるのであった [25, 28]．このように，適切なニアカーネルベクトルを用いることで，減衰しにくい成分（つまりニアカーネルベクトル成分）を効率よく減衰させることができ，高い収束が実現可能となる．ここで，適切な Prolongation 行列を設定するためには $\tilde{\boldsymbol{e}} = \boldsymbol{e} + P\boldsymbol{v}$ から，$\boldsymbol{e} \in \mathrm{Im}P$ となることが必要である [25, 28]．式 (4.2) において簡単な補間演算子生成の例を示したが，この設定はポアソン方程式のような一部の単純な問題に対してのみ有効である．そこで著者らの研究から，SA-AMG 法ではニアカーネルベクトルを用いて，Prolongation 行列の作成を行うことが有効となることがわかっている（このニアカーネルベクトルを用いた補間演算子生成に関しては，5.2.1 節において数値実験を交え示す）．これにより，Prolongation 行列の各行ベクトルの線形結合によりニアカーネルベクトルを表現することが可能となり，$\boldsymbol{e} \in \mathrm{Im}P$ となることが期待できる．以上より Multigrid 法の特徴である高い収束性の実現のため，実際に SA-AMG 法を適用する際には，ニアカーネルベクトルの生成および設定方法の確立が必要不可欠となる．

問題の性質からニアカーネルベクトルが特定できる場合もある [24]．例えば，本研究で用いている 3 次元弾性体の問題では，平行移動成分と回転成分がニアカーネルベクトルとして知られている．弾性体問題は 1.3 節でも述べたように，物体に力が加えられたときの，物体の変形をシミュレートする問題である．平行移動や回転は物体の変形が伴わないため，物体自体に力が加わらない，つまり連立一次方程式 $A\boldsymbol{u}=\boldsymbol{f}$ に対し，u にこれらの成分を用いることで，右辺ベクトル $\boldsymbol{f}$ が 0 となることが知られている．弾性体の問題を対象としている場合，これらを SA-AMG 法に用いることで，収束性が高まる．具体的には，平行移動成分（$(1,0,0)(0,1,0)(0,0,1)$）と，回転成分（$(0,-z,y),(z,0,-x),(-y,x,0)$）がニアカーネルベクトルとなる．これらについては，5.2.1 節にて，数値実験による結果を示す．

5.1.2 SA-AMG 法への補間演算子生成方法

ニアカーネルベクトルの補間演算子への設定方法の概要を Algorithm 8 に示す [14–16]．本節では，実際にニアカーネルベクトルを使用した場合の処理の流れを示す．レベル l にお

ける補間演算子の生成は，Algorithm 8 のように示す流れで作成を行う．この処理をレベル $l+1$ 以降においても同様に行い，階層毎に補間演算子 $P_{l+1}, P_{l+2}, ...$ を生成する．

以上の補間演算子作成の流れを，図 5.1 に図示する．図 5.1 は 2 本のニアカーネルベクトルを用いて，問題 A から 2 つのアグリゲートが作成されたときの，Prolongation 行列の作成方法を図示している．図 5.1 のように，まずアグリゲートの節点番号に対応したニアカーネルベクトルの列要素を，一次行列 S に入力する（a）．その後，S に対し QR 分解を行い，算出された行列 Q を仮の補間演算子 $\tilde{P}_l$ に入力する（b）．同時に算出された行列 R は，アグリゲート情報を基に，次レベルのニアカーネルベクトル V_{l+1} の構築に用いられる（c）．最後に，仮の補間演算子 $\tilde{P}_l$ に緩和法を適用する（d）．以上の操作を行うことで，補間演算子 P_l や次レベルのニアカーネルベクトル V_{l+1} が生成される．次レベルにおけるニアカーネルベクトル V_{l+1} は要素番号と対応させるために，1 節点が $N_v \times N_v$ のブロック行列として扱うこととなる．例えば，図 5.1 の例では Level$l+1$ において，1 要素が 2×2 のブロック行列として扱う必要がある．図 5.1 は 2 本のニアカーネルベクトルを用いた例だが，本研究の SA-AMG 法では，より多くのニアカーネルベクトルを設定することができる．その場合，Step.1 における補間演算子の候補行列 $\hat{P}$ の行列サイズが大きくなり，結果として計算量が増加する．そのため，ニアカーネルベクトルを複数本設定することで反復回数が減少するが，1 反復あたりの計算時間が増加するといった，トレードオフが発生すると考えられる．

Algorithm 8 補間演算子生成方法

Step.1 Decomposition and Restriction

ニアカーネルベクトル群 V_l から，生成されたアグリゲート情報を基に，一次行列 $S_1, S_2, ...(S \in \mathbb{R}^{N_l \times N^v})$ を作成する（4.1 節でも述べたように，アグリゲート数は次の節点数 N_{l+1} と同一であることに注意）

for $i = 1$ to N_{l+1} **do**
  $kernel\_num = 1$ to N^v
    $S_i^{j,kernel\_num} \leftarrow v_l^{j,kernel\_num} \mid_{j \in C_i}$,
  end for
end for

Step.2 QR decomposition

S に対して QR 分解を行い，直交行列 $Q \in \mathbb{R}^{N_l \times N^v}$ と上三角行列 $R \in \mathbb{R}^{N^v \times N^v}$ を生成する．

for $i = 1$ to N_{l+1} **do**
  $\tilde{S}_i \rightarrow Q_i R_i$
end for

その後，Q を基に仮の補間演算子 $\tilde{P}_l \in \mathbb{R}^{N_l \times (N^v N_{l+1})}$ を生成する．

for $i = 1$ to N_{l+1} **do**
  $\tilde{P}_l^{\cdot,(i-1)N^v} \leftarrow Q_i$
end for

次のレベルのニアカーネルベクトル V_{l+1} を，R 行列を基に作成する．$\tilde{P}$ の作成時に使用したアグリゲート情報から，行列 R は次の粗いレベルの節点と対応できる．行列 R を次レベルの節点情報を基に組み合わせ，次レベルのニアカーネルベクトルを作成する．

for $i = 1$ to N_{l+1} **do**
  $v_{l+1}^{(i-1)N^v,\cdot} \leftarrow R_i$
end for

Step.3 Final smoothing

$\tilde{P}$ をこのまま Prolongation 行列として用いてもよいが，最後に要素間の係数を滑らかにするため，スムーザを適用する．本研究では減速ヤコビ法を適用しており，以下のように表記される．

$P_l \leftarrow (I - \omega D^{-1} A_l)\tilde{P}_l$

C_i: i 番目のアグリゲートに含まれる要素集合
N_l: レベル l における未知数個数
N^v: ニアカーネルベクトルの設定本数
$a_l^{i,j}$: レベル l におけるベクトル a の i 行 j 列目の要素
（行要素すべてを参照する場合は $a_l^{i,\cdot}$，列要素は $a_l^{\cdot,j}$ と表記）
$P_l = \{p_l^{\cdot,1}, p_l^{\cdot,2}, ...(p \in \mathbb{R}^{N_l})\}$: レベル l における Prolongation 行列
$V_l = \{v_l^{\cdot,1}, v_l^{\cdot,2}, ...(v \in \mathbb{R}^{N_l})\}$: レベル l におけるニアカーネルベクトル群の行列
D: A の対角要素
ω: 減速係数

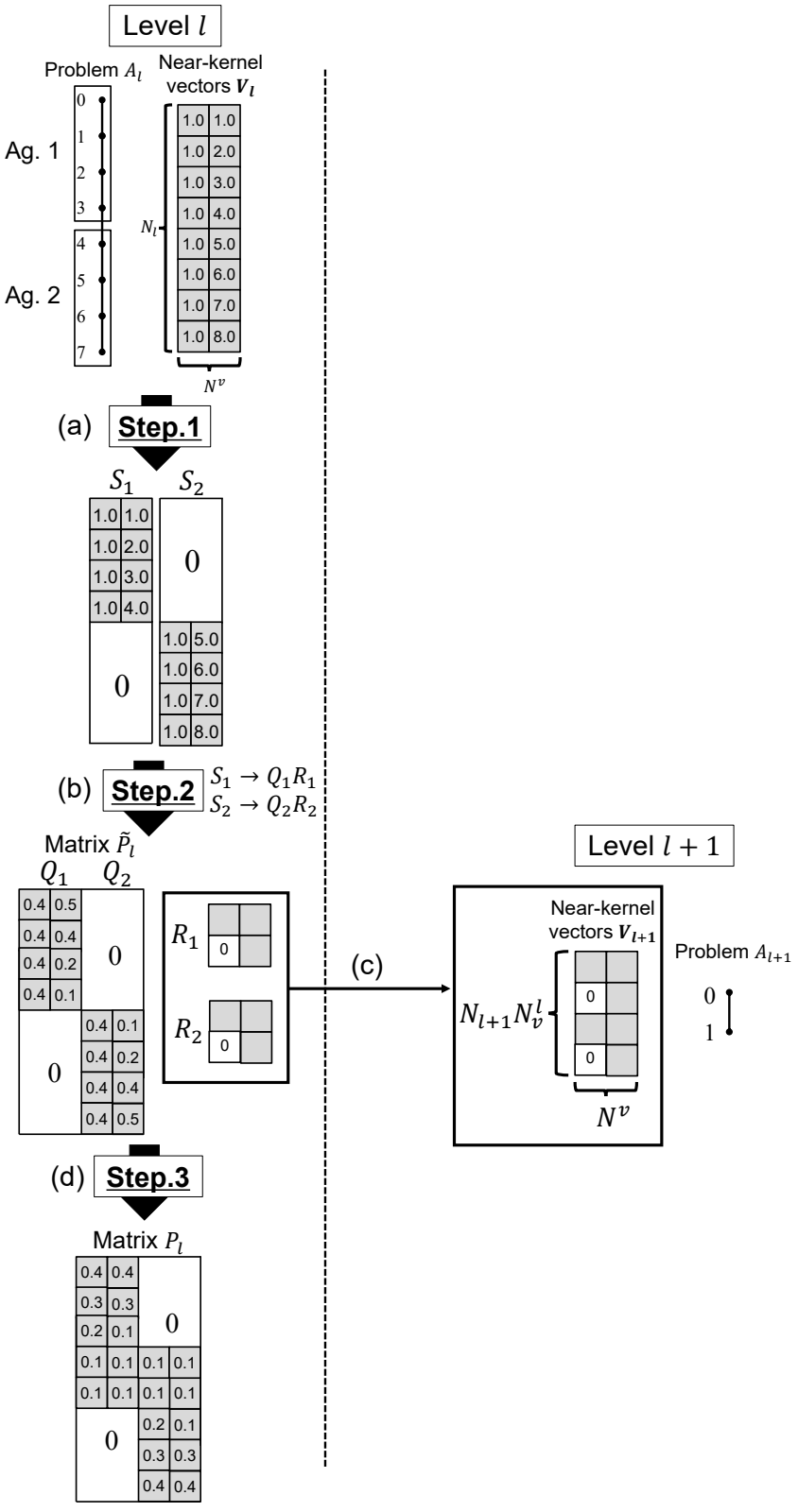


図 5.1: 補間演算子（Prolongation 行列）の生成方法

5.2 著者らによる先行研究におけるニアカーネルベクトル抽出手法

ニアカーネルベクトル抽出手法の先行研究として αSA 法 [17,18] があるが，この手法では全階層の行列に対するニアカーネルベクトルを 1 本のベクトルで作成する（詳しくは 5.9 節にて述べる）．しかし著者らは，全階層を 1 本のベクトルのみに基づいて表現することは難しく．さらにニアカーネルベクトルを用いることによる SA-AMG 法の収束性への影響について，最も細かいレベルの行列が，最も影響を与えるのではないかと考えた．そこで，抽出時間の削減も含め，ニアカーネルベクトル抽出の効率化を図るため，著者らによる先行研究として，αSA 法を基に，V-cycle を用いレベル 1 のみにおいてニアカーネルベクトルの抽出を行う手法を提案し，収束性や実行時間に関する計測および評価を行った [19]．Algorithm 9 に [19] における抽出手法の概要を示す．この手法では Algorithm 9 のように，まず初期ベクトル $\boldsymbol{x}$ を乱数にて初期化を行い（4 行目），$A\boldsymbol{x}$=0 を対象に V-cycle を μ 回繰り返す（5 行目）．この μ はユーザー側が与えるパラメータである．その後，ニアカーネルベクトル候補群である行列 B に，V-cycle により出てきたベクトル $\boldsymbol{x}$ を入力する（6 行目）．その後 αSA 法と同様に，抽出されたニアカーネルベクトル群の行列 B を使用して，Algorithm 8 で示した補間演算子 P の生成，および粗いレベルの行列生成を再度全レベルで行い，階層行列を再度作り変える（7 行目）．これを抽出したい本数分繰り返す（3 行目から 8 行目）．最後に，ニアカーネルベクトル候補群である行列 B を，ニアカーネルベクトルとして出力する．このようにして，レベル 1 の与えられた問題行列 A に対して，ニアカーネルベクトルの複数本抽出を実現している．本手法では最初に行列 B を与えることとなっているが，これはニアカーネルベクトルとして考えられるベクトルを入力する（例えば，3 次元弾性体問題では平行移動や回転成分を行列 B に与えている）．図 5.2 に Algorithm 9 の流れを図示する．この図の各段は Algorithm 9 の for ループの各 n で行う処理を示す．この図のように，V-cycle を用いて残った成分 $\boldsymbol{e}$ を集めた行列 B を用いて新しい階層行列と補間演算子を生成（図中では A', P', R' などと表記）し，新たな抽出を行う．

Algorithm 9 の 5 行目において $A\boldsymbol{x} = 0$ に対し V-cycle を適用しているが，$A\boldsymbol{x} = 0$ を V-cycle で近似的に解くことにより，V-cycle で減衰されない成分が残ることが期待できる．つまり，収束しにくいニアカーネルベクトル成分 $\boldsymbol{e}$ に対し，前処理行列を M とすると

$$M\boldsymbol{e} \approx \boldsymbol{e}$$

となる成分が残ると考えられる．これを SA-AMG 法にニアカーネルベクトルとして追加設定し，それらを基に階層行列を再生成し，さらに再度 V-cycle で近似的に解く．これにより，SA-AMG 法における設定したニアカーネルベクトルの成分が効率よく減衰される性質により，設定されたニアカーネルベクトルの成分が除かれた，新たなニアカーネルベクトルが抽出されると期待できる（誤差の影響で，厳密には独立とならない）．つまり，ニアカーネルベクトル全体のなす空間を E，n 本の抽出されたニアカーネルベクトル成分でなす空間を $\tilde{E}_n$ とし，n 本目に抽出されたニアカーネルベクトルを e_n とすると，$n+1$ 本目に抽出されるニアカーネルベクトル $\boldsymbol{e}_{n+1}$ は

$$\boldsymbol{e}_{n+1} \in E \backslash \tilde{E}_n$$

となるいずれかのニアカーネルベクトル成分が抽出できることが期待できる．これを実現するために，5 行目にて抽出されたニアカーネルベクトルを用いて，階層行列の再構成を行っている．本研究ではニアカーネルベクトルが抽出されるたびに，余分な成分を除去するため QR 分解を施し，抽出された各ベクトルが一次独立となるような処理を行っている．これにより，本手法を用いることで，適切なニアカーネルベクトルを効率よく複数本抽出できると予想される．この手法では，最初にニアカーネルベクトルを設定する必要があり，このベクトルをもとに独立なニアカーネルベクトルを抽出していくこととなる．

上記のようにニアカーネルベクトルを設定する手法はいくつか存在する．本研究では [19] の手法を基に新たなニアカーネルベクトル抽出手法を提案し，計測および有用性の評価を行った．次章より，本研究で用いたニアカーネルベクトル抽出手法について述べる．

Algorithm 9 著者らの先行研究 [19] におけるニアカーネルベクトル抽出手法

$Given : B$
for $n = 1$ to $extract\_number$ **do**
  $\tilde{\boldsymbol{e}} \leftarrow Random()$
  $e \leftarrow V\_cycle^{\mu}(A\tilde{\boldsymbol{e}} = 0)$
  $B \leftarrow [B, \boldsymbol{e}]$
  $Multilevel\_creation(B)$
end for
Output B matrix as near-kernel vectors

$extract\_number$：抽出したい本数
A：与えられた問題行列 A
$V\_cycle^{\mu}(A\boldsymbol{e} = 0)$：$A\boldsymbol{e} = 0$ を対象に V-cycle を μ 回適用
B：ニアカーネルベクトル候補群の行列
$[B, \boldsymbol{e}]$：行列 B の最終列へのベクトル $\boldsymbol{e}$ の追加
$Multilevel\_creation(B)$:行列 B を基に，補間演算子 $P_1, P_2, ...$，および階層行列 $A_1, A_2, ...$ を再生成

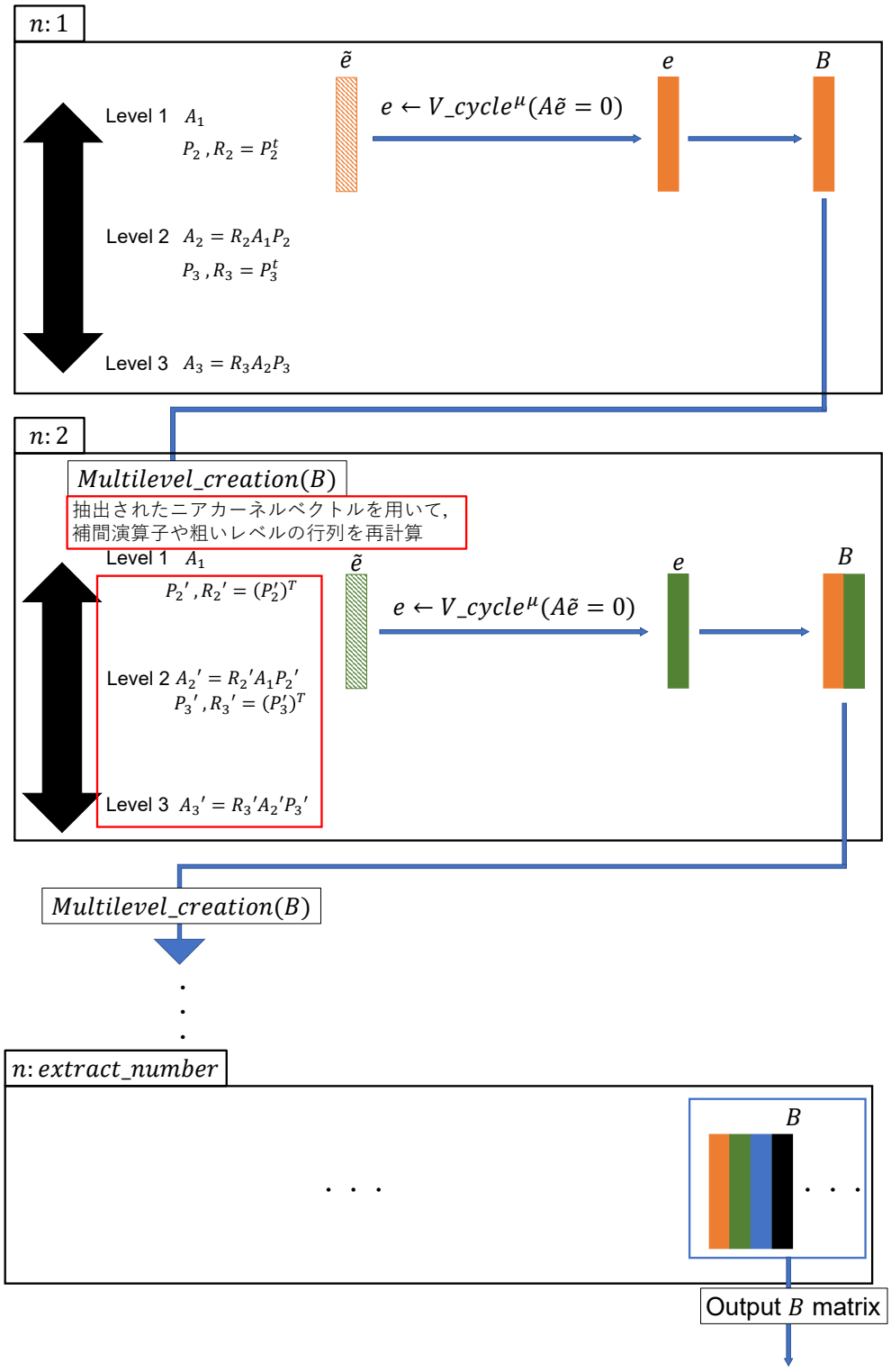


図 5.2: 先行研究 [19] におけるニアカーネルベクトル抽出手法の概要（n : Algorithm 9 における for ループの変数）

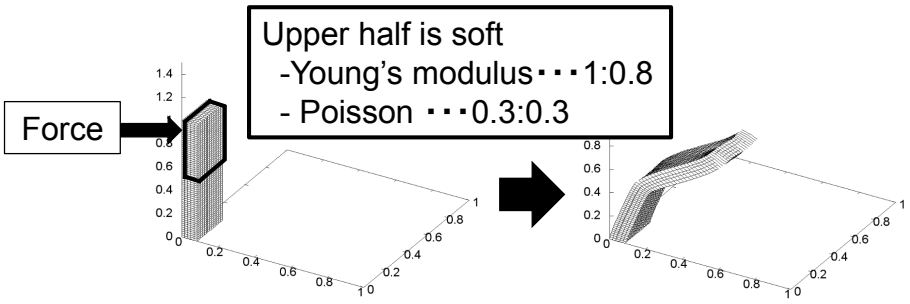


図 5.3: 3 次元弾性体問題を用いた事前実験（5.2.1 節）における問題設定（問題サイズ：1 プロセスあたり $6 \times 15 \times 60$）

表 5.1: 3 次元弾性体問題を用いた事前実験（5.2.1 節）における比較対象

Candidate	The number of near-kernel vectors
Case 1	1 (Constant vector in which all elements are 1)
Case 2	3（Parallel translation in each axis (X, Y, Z)）
Case 3	6（Case 2 + rotation in each axis (X, Y, Z)）

5.2.1 事前実験：3 次元弾性体問題におけるニアカーネルベクトル設定による収束性検証

図 5.3 と表 5.1，および図 5.4 に，SA-AMG 法においてニアカーネルベクトルを複数本設定することによる反復回数の変化を示すための，事前実験の問題設定と比較対象，およびその結果を示す．本実験では，東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営する，最先端共同 HPC 基盤施設（JCAHPC：Joint Center for Advanced High Performance Computing）による Oakforest-PACS スーパーコンピュータシステム [21] において実験を行った．図 5.3 にこの実験で対象とした問題を示す．この問題は 3 次元弾性体問題であるが，1.3 節で述べたように，1.3 節のものとは異なる問題設定を用いている．具体的には，上半分のヤング率を 0.8，下半分を 1 に設定し不均質性を持たせることで，より悪条件な問題となるような設定を行っている．反復の終了条件は相対残差が 1.0×10^{-7} となったときとした．また，問題サイズについては，1 プロセスあたり $6 \times 15 \times 60$ としたウィークスケーリングで計測を行った．さらに表 5.1 において，事前実験における比較対象を示す．表 5.1 に示すように，事前実験ではニアカーネルベクトルの設定本数により，3 種類の比較対象を用意し実験を行った．図 5.4 に，収束までに必要とした反復回数のグラフを示す．図 5.4 より，ニアカーネルベクトルを複数本設定することで，収束性の改善がみられることがわかる．これは，適切なニアカーネルベクトルが設定できているため，このような傾向がみられたと考えられる．本研究では数値実験において，ニアカーネルベクトルを問題行列から抽出することにより，これらのベクトルよりもさらに反復回数の改善が可能となる性質のよいニアカーネルベクトルが設定できるかの検証も行う．

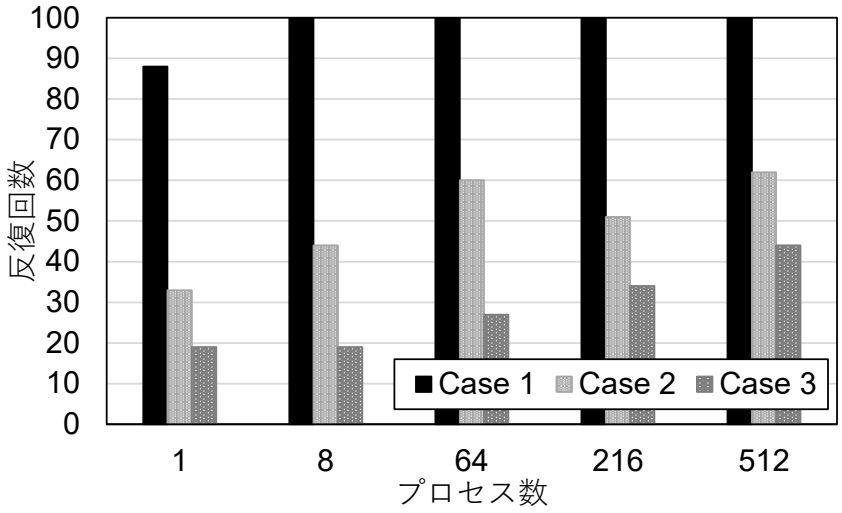


図 5.4: ニアカーネルベクトル設定による SA-AMG 法の反復回数の変化（事前実験（5.2.1 節）の結果，凡例の詳細は表 5.1 に記載）

5.2.2 事前実験：2 次元定常熱伝導問題に対するニアカーネルベクトル設定による収束性検証

前節において，3 次元弾性体問題においてニアカーネルベクトルの複数本設定による，収束性における有用性を検証した．そこで，3 次元弾性体問題以外の問題に対しても，ニアカーネルベクト設定の有用性を示す．本章では，2 次元定常熱伝導問題

$$\frac{\partial}{\partial x}\left(\lambda_{x,y}\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda_{x,y}\frac{\partial u}{\partial y}\right)+\dot{F}(x,y)=0$$

（ただし，本実験では 4 面における領域境界 Γ 上で Dirichlet 境界条件（$\Delta u(x,y)=0,(x,y\in\Gamma)$）を付している）に対し，ニアカーネルベクトルを複数本設定した際の有用性を示す．本実験で使用した問題の概要を図 5.5 に示す．図 5.5 のように，本実験の問題では 4 つの領域に分割し，それぞれの領域で異なるパラメータ設定を行った．まず，要素数を $X_1=X_3=32, X_2=X_4=128, Y=160$ と設定した．また，領域 1 においての発熱量 $\dot{F}$ を，節点座標 (x,y) に依存するように設定し（本実験では $Q=1.0$ に設定），その他の領域では $\dot{F}=0.0$ に設定した．さらに熱伝導率を，領域 1 や領域 2 および領域 4 では $\lambda_1=\lambda_2=\lambda_4=1.0$ と設定し，領域 3 における熱伝導率 λ_3 を，他領域と同等の $\lambda_3=1.0$ と不均質性を持たせた $\lambda_3=0.067$ の 2 種類の設定において実験を行った．

本実験の結果を以下に示す．本実験では 5.2.1 節と同様，Oakforest-PACS スーパーコンピュータシステムにおいて，1 コアのみを用いて実験を行った．また本実験では比較対象として，ポアソン方程式や拡散方程式のニアカーネルベクトルとして知られている定数ベクトル $(1,1,...)^T$ を，ニアカーネルベクトルとして設定した際の結果を 1p として記載している．また,「レベル 1 のみ」は 5.2 節において示した，著者らによる先行研究の抽出手法を用いた時の結果を,「粗いレベルで抽出」は以降の 5.4 節にて説明を行う提案手法 1 を用いた時の結

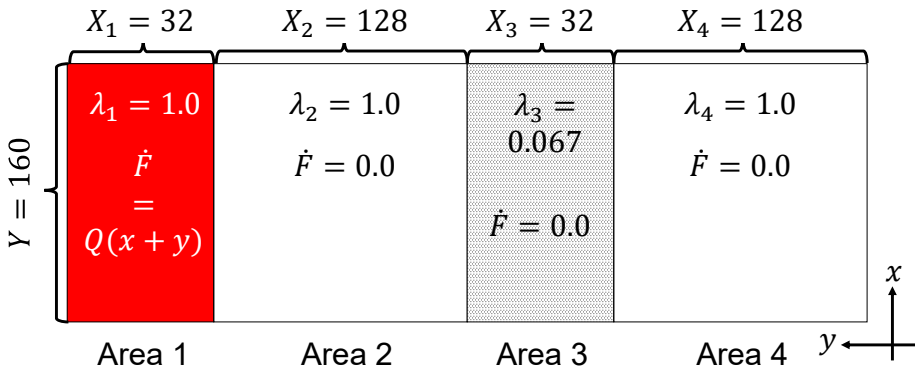


図 5.5: 本研究で用いた 2 次元定常熱伝導問題の問題設定

表 5.2: 事前実験 5.2.2 節の結果：2 次元定常熱伝導問題に対するニアカーネルベクトル設定本数による反復回数の変化（均質性問題，$\lambda_3 = 1.0$，1p：従来用いていたすべての要素が 1 の定数ベクトル，1p+1,1p+2,...：抽出したニアカーネルベクトルの本数，レベル 1 のみ：著者らによる先行研究の抽出手法，粗いレベルで抽出：提案手法 1）

	ニアカーネルベクトル設定本数				
	1p	1p+1	1p+2	1p+3	1p+4
レベル 1 のみ	5	4	4	3	3
粗いレベルで抽出:+1	—	4	4	3	3
粗いレベルで抽出:+5	—	4	4	3	3

果となっている．さらに，1p+1，1p+2... は，抽出したニアカーネルベクトルの本数を示している．均質性問題（$\lambda_3 = 1.0$）における結果を表 5.2，不均質性問題（$\lambda_3 = 0.035$）における結果を表 5.3 に示す．不均質性が強い問題では条件数が大きくなり，それに伴い収束性も悪化する．実際，表 5.3 の 1p では収束性の悪化が見受けられる．しかし，ニアカーネルベクトルを適切に設定することで，高い収束性を発揮していることがわかる．さらに，ニアカーネルベクトルの設定本数を増加させることで，さらに収束性の改善がみられることもわかる．以上より，ニアカーネルベクトルを適切に設定することで収束性が改善し，特に不均質性が強い，条件数が大きい問題に対して高い効果を発揮することがわかる．

表 5.3: 事前実験 5.2.2 節の結果：2 次元定常熱伝導問題に対するニアカーネルベクトル設定本数による反復回数の変化（不均質性問題，$\lambda_3 = 0.067$，表の項目は表 5.2 と同様）

	ニアカーネルベクトル設定本数				
	1p	1p+1	1p+2	1p+3	1p+4
レベル 1 のみ	81	4	4	3	3
粗いレベルで抽出:+1	—	4	4	3	3
粗いレベルで抽出:+5	—	4	4	3	3

5.3 提案手法の学術的な意義

本研究の提案手法を説明するにあたり，提案手法の学術的な位置づけを説明する．そのために，本節においてまず本論文で述べてきたことを，歴史的な側面と数学的な観点からまとめる．

SA-AMG 法の基となる AMG 法が最初に提案されたのは，1985 年に Brandt A. らによる classical AMG 法である [35]．この時点において，幾何的な Multigrid 法における収束しにくい長波長成分を代数的な側面で考えるため，$M\boldsymbol{e} \approx \boldsymbol{e}$ つまり $A\boldsymbol{e} \approx 0$ となる成分 $\boldsymbol{e}$（この時点では smooth error と呼ばれていた）を定義し，これを減衰させるため 3.5 節で説明したような処理を行った．その後，1995 年に Vanek P. らにより，AMG 法の派生解法である SA-AMG 法が提案された [2,3]．これらの手法では，補間演算子の生成の際には，式 (4.2) で示したような単純なものを使用していた．また [14] では，補間演算子の生成に，問題行列からわかる Zero energy function （ニアカーネルベクトルと同様の意味）を用いていた．SA-AMG 法の分野において，ニアカーネルベクトルという言葉が初めて取り入れられたのは，2004 年に Brezina M. らにより提案された αSA 法である [17,18]．従来では問題行列からわかるニアカーネルベクトルを用いていたが，本論文の手法では SA-AMG 法適用前にニアカーネルベクトルを抽出することで，様々な問題に対し SA-AMG 法を有効に機能させることを目的とした．その後，様々な抽出手法が提案されたが，それらの手法では最終的には最上層のレベルのみにおけるニアカーネルベクトルとして抽出する手法である（これらの手法に関しては後述の 5.9 章にて説明を行う）．これらをまとめたものを図 5.6 に示す．図 5.6 に示すように，従来手法（αSA 法 [17,18]，Bootstrap AMG 法 [48,49]，著者らによる先行研究 [19]）では，最上層のみに対してニアカーネルベクトルの設定を行っていた．しかしマルチレベルによる収束しにくい誤差成分の減衰がうまく行われているかは不明であり，GMG 法の観点から考えると，真のマルチレベルな解法になっていないのではないかと考えられる．

このことを数学的観点から説明する．まず，GMG 法の観点から考える．減速 Jacobi 法のような定常反復法は，格子サイズと同じ波長の波数成分が減衰しやすい．GMG 法はこの特徴を利用し，離散点数を 1/2，1/4，... とすることで，各波数成分を減衰させるのであった．言い換えると，減速ヤコビ法において，どの離散点数 n に対しても「中周波から高周波が一様に減衰する」ため,「離散点数 n における低周波成分を，異なる離散点数（$n' = n/2$ 等）における中周波から高周波成分に読み替えて減衰する」ことを目的としている．そのため，減衰しにくい低周波な成分を減衰することを考える必要がある．ここで，収束しにくい波数成分を $\boldsymbol{e}$ とし，各レベルでそれらがなす空間を E_l とする．また前処理行列を M とすると，

$$M\boldsymbol{e} \approx \boldsymbol{e} \qquad (e \in E_1)$$

のような形で低周波成分を表記できるのであった（2.1 節より）．そのうえで GMG 法の V-cycle を考えると，GMG 法では

$$E_1 \supset PE_2 \supset ...$$

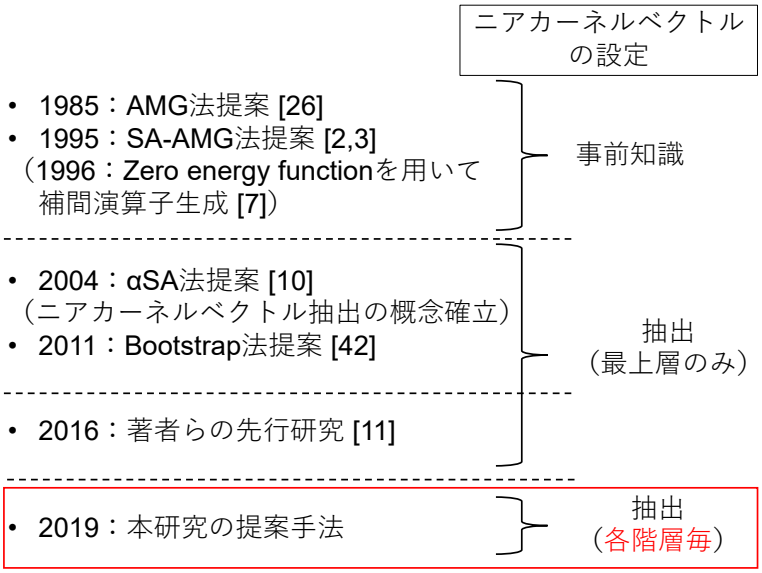


図 5.6: 関連研究まとめ

のようになる（ただし，P_2^1 はレベル 2 から 1 へ補間を行う Prolongation 行列）．つまり，レベル 1 から 2 において，差分集合 $E_1 \backslash PE_2$ が，レベル 2 において減衰させる対象である「レベル 1 では減衰しにくい成分」となる．以上のように，Multigrid 法は各レベルでの E の差を利用し，各レベルで異なる波数成分を減衰させるところにある．しかし，これを実現するには，3.5 節や 5.1.1 節で述べたように，

$$E \subseteq \mathrm{Im}P$$

となることが必要である．

以上で述べてきたことをまとめる．ここで，各レベルにおける係数行列を $A_{l+1} = R_l^{l+1} A_l P_{l+1}^l$ とし，レベル間の補間演算子が $P_{l+1}^l, R_l^{l+1} = (P_{l+1}^l)^T$ となっているとする．このとき，

- 各レベルの担当部分空間 E_l に有意な差が存在：$E_l \backslash P_{l+1}^l E_{l+1} \neq \phi$
- レベル間の補間における有意な情報伝達：$E_l \subseteq \mathrm{Im}P_{l+1}^l$

となれば，Multigrid 法の目的である各レベルにおける波数成分の減衰が効率的に行われる．以上を示した図を図 5.7 に示す．この図における E_l は上記で示した減衰しにくい成分を示し，G_l はそれ以外の減衰しやすい中波長から長波長成分を示す．このように Multigrid 法では，各レベルにおける異なる離散点数による波数成分の減衰を利用し，効率よく長波長成分を減衰させることを目的とする．

ここで，これを SA-AMG 法上においても同様に考える．GMG 法では，各レベルは異なる離散点数による空間離散化に対応しており，A_l や E_l は異なる離散点数における同一の存在として解釈できた．そして補間演算子 P に対し，適切な設定を施すことでそれらの要件が実現できた．しかし，SA-AMG 法においては（AMG 法全般に言えることだが）これら

の空間的な解釈を用いることができず，代数的な観点から A_l や P_l の構築を考えなければならない．そこで SA-AMG 法では，代数的な考えにより構成された節点集合（アグリゲート，Aggregate）と，E_l の要素を近似的に満たす

$$B_l \approx E_l$$

となる部分空間 B_l を用いることで，A_l や P_l の構築を行う．ここで，$B_l \approx E_l$ としたのは，図 5.7 に示した GMG 法の例を実現するためには,「逐一固有値問題を解く」といった処理が必要となり，実用上あまりにもコストが大きいからである．そこで SA-AMG 法では，問題設定から予想できるニアカーネルベクトルを設定するなどし，最上層における近似的なニアカーネルベクトル部分空間 B_1 の設定を行っていた．また，先行研究におけるニアカーネルベクトル抽出手法においても同様である．以上のようにマルチレベルを構成することを考える場合，マルチレベルを規定するのは，アグリゲートと B_1 のみである．この状況においては，Multigrid 法において必要な条件であった $E_l \backslash P_{l+1}^l \neq \phi$，言い換えると

$$E_1 \approx B_1 \subseteq \mathrm{Im}P_l^1 \qquad (P_l^1 := P_2^1 P_3^2 ... P_l^{l-1})$$

となることが必要である．しかし，SA-AMG 法では代数的に各レベルの補間演算子 P を生成するため，これを担保しないことが考えられる．このことを，図 5.8 にて説明する．図 5.8 のように，粗いレベルの行列において，補間演算子生成に必要な行列 B は QR 分解を用いて生成を行うが，この行列が各レベルの行列に対するニアカーネルベクトル成分に近い成分となっているかは不明である．これにより，粗いレベルによる収束しにくい誤差成分の減衰がうまくいかず，効率よく減衰できない可能性がある．この状況を図 5.9 に示す．図 5.9 に示すように，上記のような状況下では，補間演算子による波数成分減衰の情報がうまく伝わらない．以上をまとめると，以下のようになる．

- 原則として，真のニアカーネルベクトル E_l はわからない（計算コストが膨大であり，非現実的）ので，その近似空間 B_l を用いる．有意なマルチレベルを構成するためには必須である．

- $P_l^1 := P_2^1 P_3^2 ... P_l^{l-1}$ とすると，Multigrid 法の観点から考えると，$E_1 \approx B_1 \subseteq \mathrm{Im}P_l^1$ であり，かつ $B_1 \backslash P_l^1 E_l \neq \phi$ であることが必須である．しかし，SA-AMG 法においては代数的に各レベルの補間演算子 P が生成されるため，これが失われていると考えられる．

そこで，本研究の提案手法では，各レベルにおいてニアカーネルベクトルの抽出を行うことで，担保できていなかった各レベルにおける補間演算子の構成要件である $E_l \backslash P_{l+1}^l \neq \phi$ を担保することが目的である．これを図 5.10 を用いて説明する．図 5.10 では図 5.8 に加え，粗いレベルにおいて抽出したニアカーネルベクトルを使用している．これにより，Multigrid 法における数学的な要請を，従来手法に比べよく実現できると考えられる．これにより，本研究の提案手法を用いることで，補間演算子による波数成分減衰の情報がうまく伝わるようになり，従来手法と比べより効率的にニアカーネルベクトル成分を減衰できることが期待

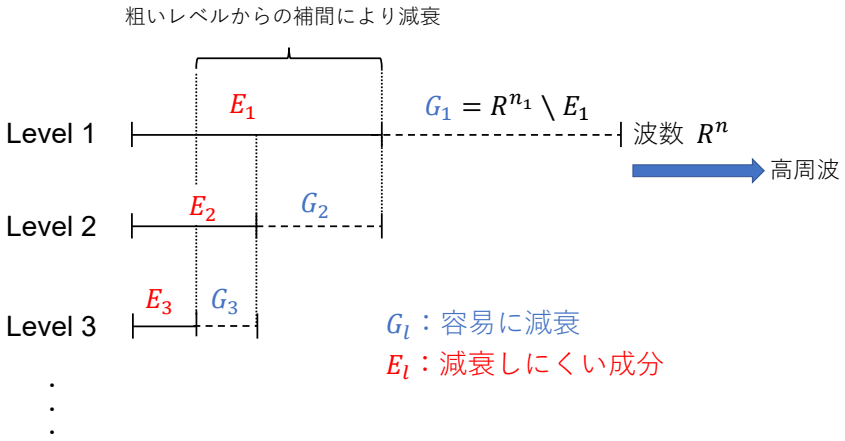


図 5.7: GMG 法における各レベルでの理論的な波数成分減衰のモデル

できる．以下の節より，本研究で提案するニアカーネルベクトル抽出手法についての説明を行う．

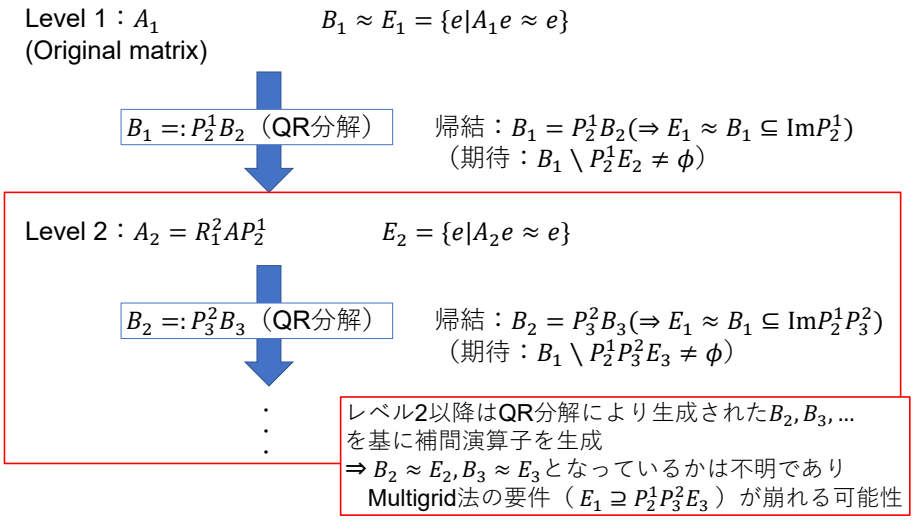


図 5.8: SA-AMG 法のニアカーネルベクトルに基づく粗いレベルの行列生成

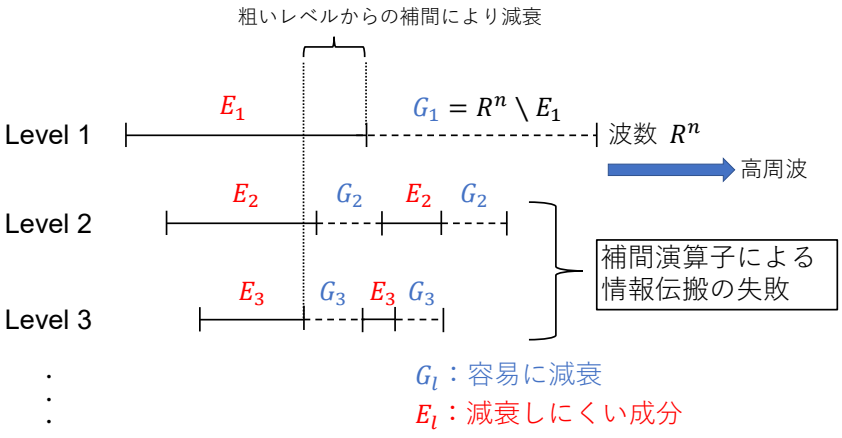


図 5.9: 実際の SA-AMG 法の運用において予想される各レベルでの波数成分減衰のモデル

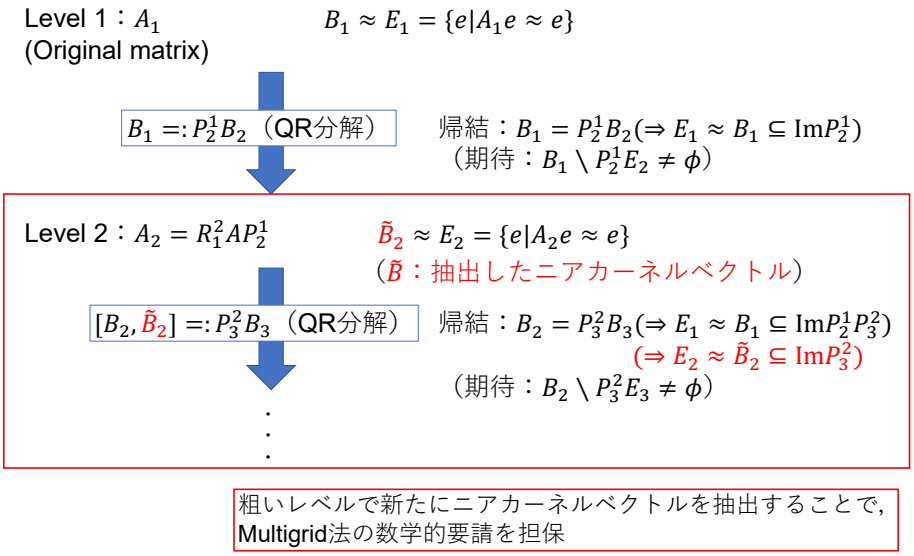


図 5.10: 提案手法を用いた SA-AMG 法の粗いレベル行列生成（$[B_2, \tilde{B}_2]$ は Algorithm 9 の時と同様，行列 B_2 に $\tilde{B}_2$ の要素を最後列に追加することを示す）

5.4 全レベル近似ニアカーネルベクトル抽出法（提案手法 1）

本章では，新たなニアカーネルベクトル抽出手法の提案を行う．まず，提案手法の概要を説明し，数値実験を交え有用性の検証を行う．

5.4.1 手法の概要

この節では，本研究で提案するニアカーネルベクトルを問題行列から抽出する手法について説明する．

[17,18] や [19] の手法では，最終的にレベル 1 としてのニアカーネルベクトルのみを出力している．しかし，粗いレベルのニアカーネルベクトルは，細かいレベルのニアカーネルベクトルだけでは不十分であり，粗いレベルにおいても抽出や追加に設定を行えるようにするべきではないかと考えた．図 5.1 で示したように，粗いレベルのニアカーネルベクトルは細かいレベルのニアカーネルベクトルを基に生成する．しかし，粗いレベルの行列は細かいレベルの行列とは異なるため，このように生成されたニアカーネルベクトルが，粗いレベルのニアカーネルベクトルとして適切であるかは不明である．ここで，レベル l における定常反復法で減衰しにくい成分を E_l とし，その近似を B_l とする．また，レベル m から n へ補間を行う補間演算子を P_m^n とする．上記で述べたことは、つまり 5.3 節でも述べたように，Multigrid 法の数学的な理論より，

$$E_1 \approx B_1 \subseteq \mathrm{Im} P_l^1 \qquad (P_l^1 := P_2^1 P_3^2 ... P_l^{l-1})$$

となることが望ましいが，粗いレベルの行列や補間演算子を代数的に構成しているため，この関係が崩れている可能性がある．そこで，粗いレベルにおいてニアカーネルベクトルを個別に追加設定することで，計算コストの増大を低く抑えつつ，崩れていた上記の数学的関係を補修し，細かいレベルで行えなかったニアカーネルベクトル成分の減衰を，効率よく行うことができると期待できる．そこで本研究では [19] の手法を基に，粗いレベルにおいてもニアカーネルベクトルを抽出および SA-AMG 法にて追加的に設定する手法の提案，および収束性や実行時間の評価を行った [50,51]．Algorithm 10 に具体的な流れを示す．Algorithm 10 の赤字箇所は，Algorithm 9 から追加された箇所である．Algorithm 10 からわかるように，本手法は Algorithm 9 を基に，粗いレベルにおいてもニアカーネルベクトルを複数本抽出できるように改良を加えた手法となっている．本手法ではニアカーネルベクトル候補群である行列 B を各階層で用意しており，最終的に各階層ごとのニアカーネルベクトルがそれぞれ出力される．これらのニアカーネルベクトルを SA-AMG 法に用いる際には，まずは先行研究による手法と同様に，最も細かいレベルにおいて抽出されたニアカーネルベクトルを設定し，次の粗いレベルの行列とニアカーネルベクトルを生成する．そして，粗いレベルでも同様に，細かいレベルのニアカーネルベクトルを基に，さらに粗いレベルを生成する．本手法ではこの際，細かいレベルをもとに生成されたニアカーネルベクトルに追加して，抽出されたニアカーネルベクトルを用いる．この操作を最大レベル数-1 まで行う．図 5.11 に上記で説明した Algorithm 10 の流れを図示する．図 5.11 の各段は抽出対象とするレベルを示す．図 5.11 のように，提案手法 1 では先行研究を基に，各レベルにおいて抽出処理を行い，各

Algorithm 10 提案手法 1 のニアカーネルベクトル抽出提案手法

$Given : B_1$
for $level = 1$ to $max\_level - 1$ **do**
  for $n = 1$ to $extract\_number$ **do**
    $\tilde{\boldsymbol{e}}_{level} \leftarrow Random()$
    $\boldsymbol{e}_{level} \leftarrow V\_cycle^{\mu}(A_{level}\tilde{\boldsymbol{e}}_{level} = 0)$
    $B_{level} \leftarrow [B_{level}, \boldsymbol{e}_{level}]$
    $Multilevel\_creation(B_{level})$
  end for
end for
Output $B_1, B_2, ...$ matrices as near-kernel vectors

$Random()$：乱数生成
max_level：階層の最大レベル数
$extract\_number$：抽出したい本数
A_{level}：レベル $level$ における問題行列 A
$V\_cycle^{\mu}(A\boldsymbol{e} = 0)$：$A\boldsymbol{e} = 0$ を対象に V-cycle を μ 回適用
B_{level}：レベル $level$ におけるニアカーネルベクトル候補群の行列
$[B, \boldsymbol{e}]$：行列 B の最終列へのベクトル $\boldsymbol{e}$ の追加
$Multilevel\_creation(B_l)$：行列 B を基に，補間演算子 $P_l, P_{l+1}, ...$，および階層行列 $A_l, A_{l+1}, ...$ を再生成

レベルそれぞれのニアカーネルベクトルを抽出する．図 5.11 の粗いレベルにおける抽出において，レベル 1 の行列を使わないため，$Multilevel\_creation$ において無駄な処理が行われてしまうように見える．しかし，本研究の実装では粗いレベルのみにおいて階層行列の再構築を行い，無駄な処理は行わないようにしている．以上のように本手法では，[19] で用いた手法に加え粗いレベルのニアカーネルベクトルも考慮しているため，粗いレベルの行列が全体の収束性に大きな影響を与えている場合，[19] よりもさらに収束性が改善することが見込める．

外部から入力するパラメータとして，$extract\_number$ と μ が存在する．$extract\_number$ は抽出したい本数であり，著者らによる研究 [19] から，適切な本数を設定することが必要であることが考えられる．数値実験において，抽出本数による反復回数や抽出時間への影響の分析を行う．また μ は，V-cycle を繰り返す回数であり，この値により収束性が変化すると考えられるが，本事項については今後の課題とし，本研究では μ を 20 に設定し，計測を行っている．

ここで，ここまでの手法に関してまとめた表を表 5.4 に示す．本研究での提案手法であることを強調するために，表 5.4 の本研究の箇所を赤字で示している．表 5.4 からわかるように，本研究の手法における最も大きな点としては，各階層でニアカーネルベクトルを抽出，および設定本数の変更が可能となったことである．次節の数値実験において，粗いレベルにおけるニアカーネルベクトル抽出本数の変更による，収束性や実行時間への影響についても示す．

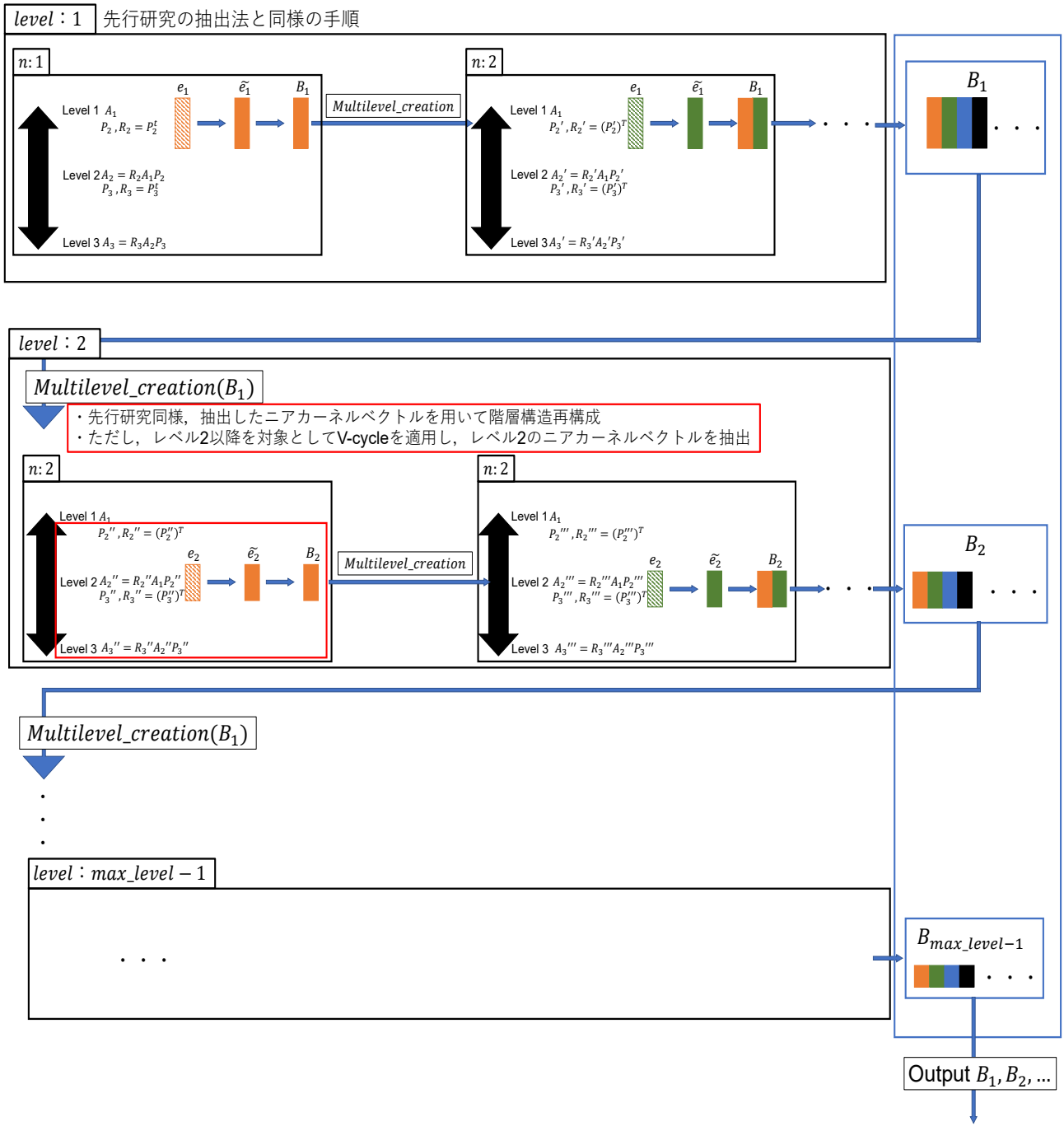


図 5.11: 提案手法 1 のニアカーネルベクトル抽出手法の概要

表 5.4: 各論文における手法の比較と実験環境設定

	[Brezina et al., 2004] [17] （αSA）	[Nomura et al., 2016] [19]	Algorithm 10 （提案手法 1）
各レベルの設定本数	一定	一定	変更
抽出方法	全階層を 1 本の ベクトルに基づき表現	レベル 1 のみ抽出	各階層で抽出
設定本数	最大 6	最大 10	最大 10（レベル 1） 最大 15（レベル 2） 最大 20（レベル 3）
プロセス数	4	512	512
問題サイズ	200,000	5,120,000	5,120,000

5.4.2 実験環境と実験内容

本研究では 5.2.1 節と同様，東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営する，最先端共同 HPC 基盤施設（JCAHPC：Joint Center for Advanced High Performance Computing）による，Oakforest-PACS スーパーコンピュータシステム [21] を使用し数値実験を行った．Oakforest-PACS は，1 ノードに 1 個の Intel(R) Xeon Phi(TM) プロセッサ（68cores，1.4GHz）と，MCDRAM（16GB，400GB/s）と DDR4（16GB×6，1 メモリ当り 19.2GB/s）メモリを搭載している（本研究では，MCDRAM のみ用いる設定を行い，数値実験を行った）．また，Oakforest-PACS では Intel®Omni-Path（12.5GB/s）により，各計算ノードを接続している．

数値実験では最大 8 ノード使用し，計測を行った．また並列化手法については，1 コアに 1 プロセス起動するフラット MPI を使用した．さらに本研究では，1.3 節に示した 3 次元弾性体の問題を使用した．問題サイズについては，1 プロセスあたり 15×15×15 のウィークスケーリング（Weak Scaling）方式による計測を行った．ウィークスケーリングは，1 プロセスが担当する問題サイズが一定になるように問題サイズを設定するようにしたものである．そのため，プロセス数が増加するほど全体の問題サイズが増加する．また，問題行列の各プロセスへの分割は，各軸方向へ問題を均等に分割し，それにより作成された小行列を各プロセスへ分配する単純な方法で分割を行った．

求解部では CG 法 [52] を使用し，前処理として SA-AMG 法を適用している．求解部で用いる V-cycle の各レベルの緩和法として，対称 Gauss-Seidel 法を 2 回適用する．ただし，領域境界では，依存関係を無視している．また節点数が 100 以下になったとき，最も粗いレベルとし，LU 分解を行い，解を求めている．また，反復の終了条件は相対残差が 1.0×10^{-7} とし，反復回数の上限を 500 回とした．求解部の CG 法は，Fortran を用い作成を行っている．また，コンパイラは Intel®コンパイラ（mpiifort）を使用した．

5.4.3 数値実験と結果

本節では，提案手法 1（Algorithm 10）の実験結果を示す．数値実験による結果を図 5.12 と図 5.13 に示す．図 5.12 と図 5.13 はそれぞれ 1 並列，512 並列時の，ニアカーネルベクト

ルの設定本数による反復回数および実行時間の変化を示している．これらのグラフでは，比較対象として 3p と 6p を設定した際の結果も記載している．まず反復回数に着目すると，本研究の提案手法（粗いレベルで抽出）を用いることで，反復回数の改善がみられることがわかる．これは，粗いレベルで追加的にニアカーネルベクトルを設定することで，粗いレベルの行列においても効率的に誤差成分の減衰が行え，全体の反復回数が削減されたものと考えられる．次に全体の実行時間に着目しても，1 並列においては提案手法を用いることで，実行時間が改善されることがわかる（「レベル 1 のみ」の最良値である 3p+3 の場合と，粗いレベルで抽出の最良値である「粗いレベルで抽出：+5」の 3p+3 を比較すると，「粗いレベルで抽出」を用いることで約 10%の削減効果）．しかし，1 並列における 3p+4 以降，および 512 並列では，6p の実行時間と比べ悪化していることがわかる．これは，ニアカーネルベクトル本数の増加に伴い，構築部における粗いレベルの行列生成にかかる時間が増加し，結果として収束性改善の効果以上に計算時間が悪化したためであると考えられる．実際，破線で示した求解部のみの実行時間を見ると，「粗いレベルで抽出」が「レベル 1 のみ」よりも良い結果が得られることがわかる．これは，収束性改善効果が表れたものである．このように，抽出したニアカーネルベクトルを追加で設定することによる，収束性改善と計算コスト増加のトレードオフが発生することがわかる．

ここで，プロセス数を変化させることによる反復回数および実行時間を図 5.14 と図 5.15，および図 5.16 に示す．グラフ内の要素である「従来最良」は，「レベル 1 のみ」において，「複数階層最良」は「粗いレベルで抽出」において最良のものを設定した際の値となっている（例として図 5.15 の 1 プロセスにおいて図 5.12 と対応させると，「従来最良」は「レベル 1 のみ」の 3p+3 の値を示し，「複数階層最良」では「粗いレベルで抽出:+5」の 3p+3 の値を示している）．さらに「従来最良」および「複数階層最良」について，図 5.14 は反復回数，図 5.15 は全体の実行時間，図 5.16 は求解部のみの実行時間にそれぞれ着目したときに，最良となる設定を行った場合の結果である（そのため，図 5.14，図 5.15，図 5.16 の「従来最良」と「複数階層最良」では，ニアカーネルベクトル設定本数がそれぞれ異なる）．図 5.14 より，粗いレベルでニアカーネルベクトルを抽出することにより，著者らによる先行研究での抽出手法（Algorithm 9）と比べ収束性の改善がみられることがわかる（複数階層最良を用いることで，512 並列において「6p」と比べ約半分，「従来最良」と比べても約 30%の削減）．これより，粗いレベルでニアカーネルベクトルを適切な本数抽出および設定を行うことで，高並列・大規模問題においてもニアカーネルベクトル成分を効率よく減衰させることができ，結果として収束性の改善がみられることがわかった．次に図 5.15 に着目すると，216 並列までは複数階層最良が最も良い結果が得られていたが，512 並列時では 6p と比べ悪化していることがわかる．しかし，図 5.16 に着目すると，複数階層最良が全体的に最も良い結果を示すこともわかる．これは上記の図 5.13 についての説明でも述べたように，ニアカーネルベクトルの設定本数を増やすことによる構築部の計算コスト増大によるものである．さらに，図 5.15 および図 5.16 において用いたニアカーネルベクトルの本数構成を，表 5.5 と表 5.6 にそれぞれ示す．表 5.5 と表 5.6 における複数階層最良において，「3p+3：+4」のように表記されているが，これはレベル 1 において 3 本抽出，粗いレベルにおいて 4 本ずつ抽出したことを示す．表 5.5 より，ニアカーネルベクトルを少数設定することで，構築部における計算コストが抑えられ，全体の実行時間が改善される傾向があることがわかる．また，

表 5.5: 図 5.15 におけるニアカーネルベクトル本数の構成

	1	8	64
従来最良	3p+3	3p+3	3p+3
複数階層最良	3p+3：+4	3p+3：+3	3p+1：+3
	216	512	—
従来最良	3p+1	3p+5	—
複数階層最良	3p+1：+3	3p+1：+2	—

表 5.6: 図 5.16 におけるニアカーネルベクトル本数の構成

	1	8	64
従来最良	3p+6	3p+5	3p+5
複数階層最良	3p+6：+5	3p+6：+5	3p+7：+5
	216	512	—
従来最良	3p+4	3p+5	—
複数階層最良	3p+6：+5	3p+7：+5	—

表 5.6 より，ニアカーネルベクトルを多数設定することで収束性が改善し，結果として求解部の実行時間の改善につながることがわかる．

上記より，ニアカーネルベクトルを適切な本数設定することで，提案手法 1 は有用となることがわかる．しかし，適切な本数の検証には，提案手法 1 ではすべてのニアカーネルベクトルのパターンを試す必要がある（例えば図 5.12 の場合，1 レベル目のニアカーネルベクトルを 7 本，2 レベル目以降は 1 本と 5 本，および抽出なしの 3 通り存在し，本実験においても合計 15 通り存在する）．ニアカーネルベクトルの取りうる本数は最大で行列サイズ分あり，すべてのパターンを網羅することは不可能である．実際に，図 5.14 と図 5.15，および図 5.16 における，「複数階層最良」の設定本数の検証にかかった時間を図 5.17 に示す．抽出適用時間総計の割合が解法適用時間総計に比べ大きいが，これは主に 2 つ要因がある．ひとつは，ニアカーネルベクトルの本数増加に伴い，階層行列の再構成（Algorithm 10 における $Multilevel\_creation(B_{level})$）のコストが増大してしまうためである．もうひとつは，細かいレベルのニアカーネルベクトルが変更された場合，粗いレベルの行列がすべて変化してしまうため，その都度粗いレベルの行列に対して，再抽出を行う必要があるためである．このグラフより，検証にかかる時間は実際の実行時間と比べ約 400〜500 倍となることがわかる．現状では，本手法を実際の問題に適用する場合に，検証に膨大な時間が必要となり，実用上の観点から問題がある．次章ではこの問題の解決に向け，適切なニアカーネルベクトルの本数を予測する方法を組み込んだ手法をさらに提案し，数値実験による検証を行う．

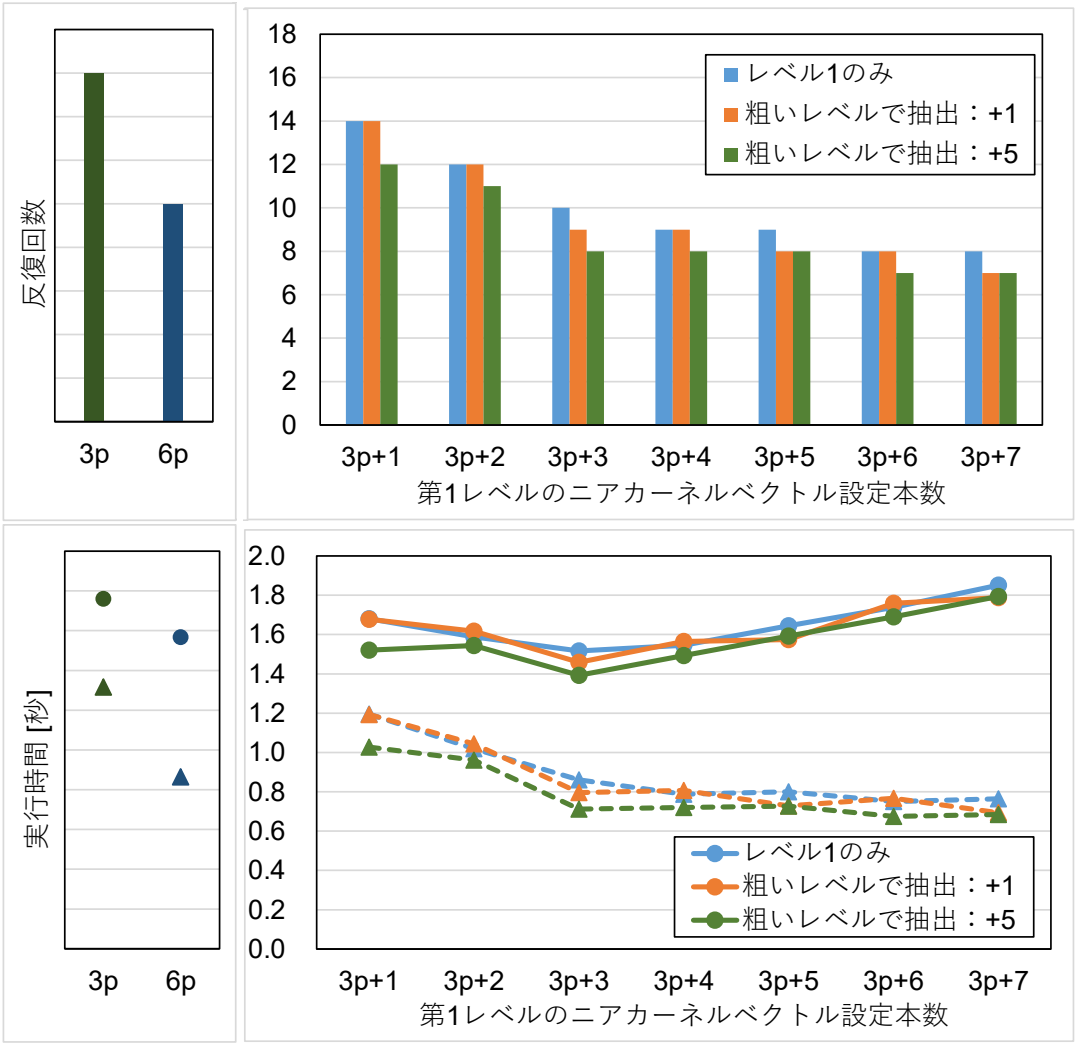


図 5.12: ニアカーネルベクトルの設定本数による反復回数（上）と実行時間（下）の変化：1 並列（下図における実線（丸マーカー）は全体（構築部＋求解部），破線（三角マーカー）は求解部のみ，レベル 1 のみ：著者らによる先行研究の抽出手法（Algorithm 9），粗いレベルで抽出：提案手法 1（+1，+5 はそれぞれ粗いレベルで 1 本，5 本抽出））

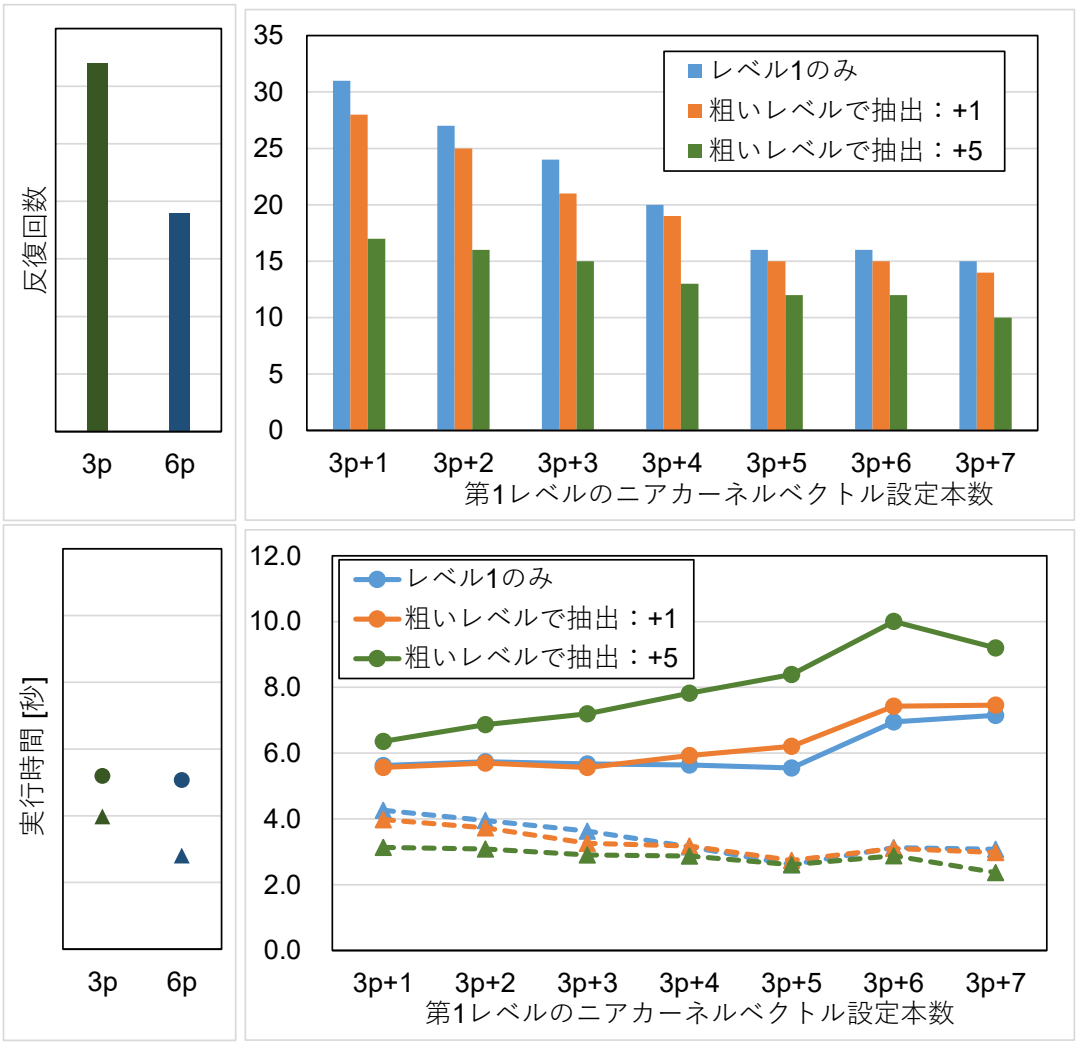


図 5.13: ニアカーネルベクトルの設定本数による反復回数（上）と実行時間（下）の変化：512 並列（グラフ要素は図 5.12 と同様）

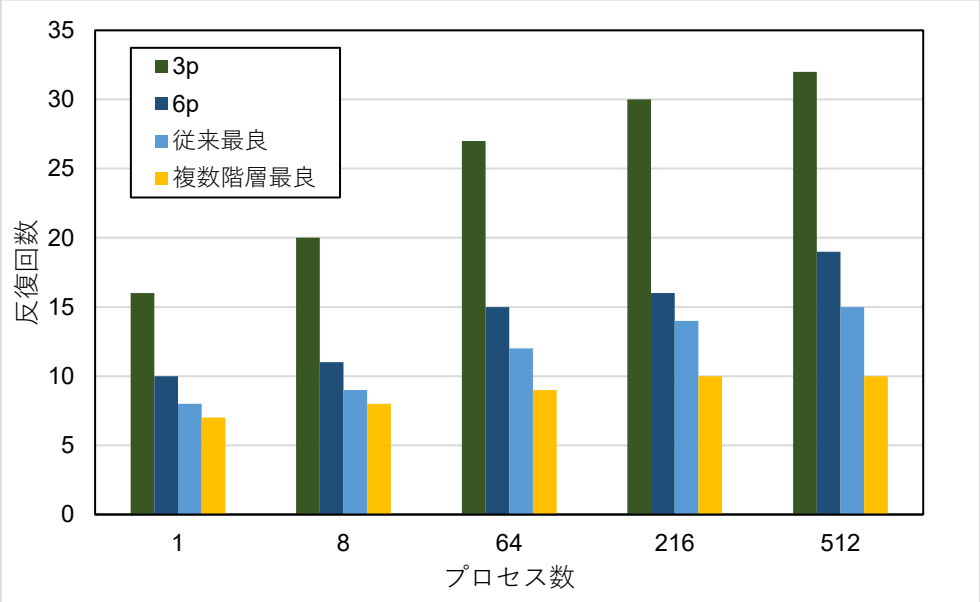


図 5.14: プロセス数変化による反復回数（3p，6p：平行移動成分（3p）と平行移動+回転成分（6p）を設定，従来最良，複数階層最良：それぞれ「レベル 1 のみ」と「粗いレベルで抽出」における反復回数に着目したときの最良本数設定時の結果）

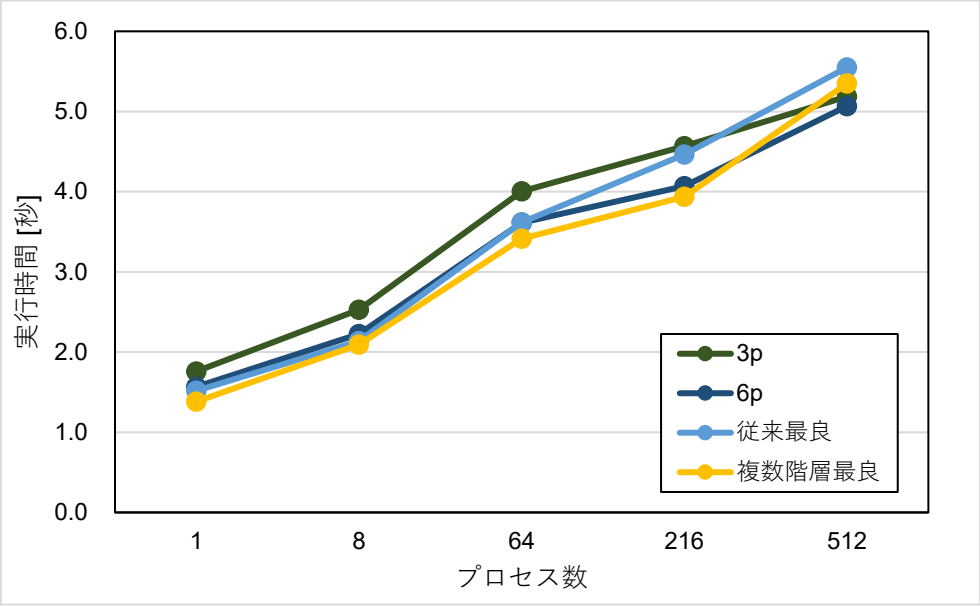


図 5.15: プロセス数変化による実行時間（グラフ要素は図 5.14 と同様）

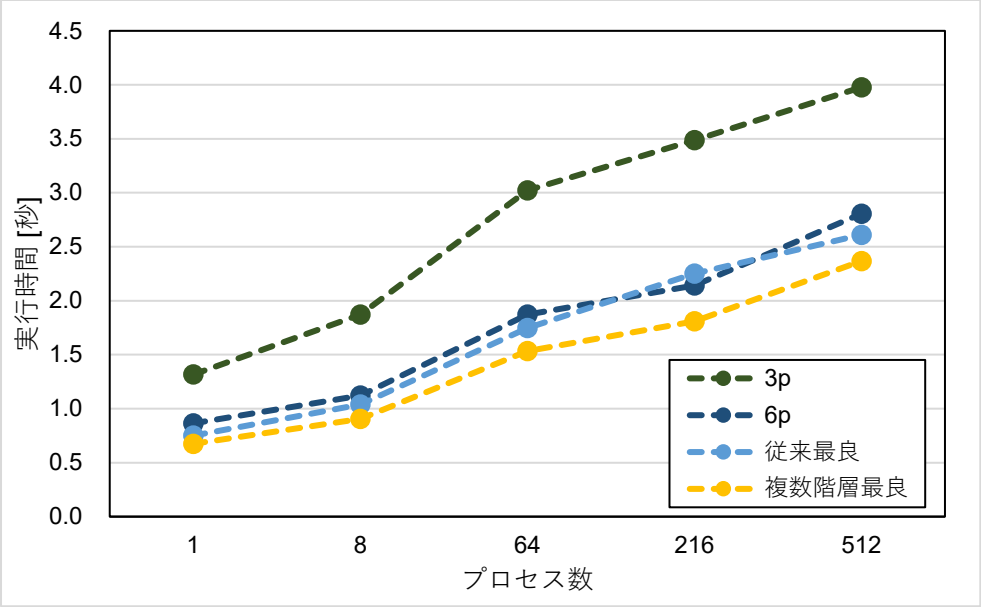


図 5.16: プロセス数変化による求解部のみの実行時間（グラフ要素は図 5.14 と同様）

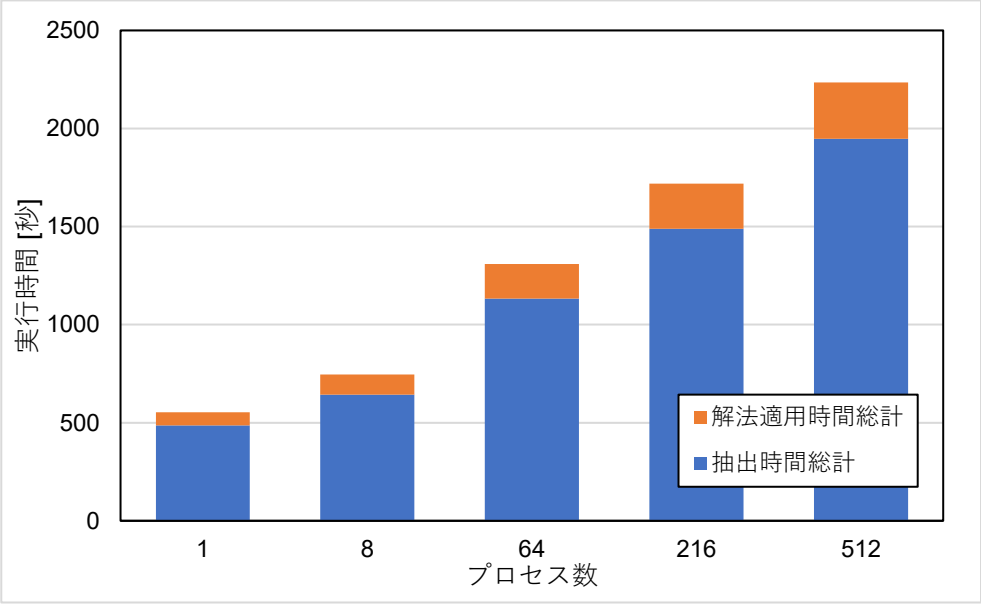


図 5.17: 適切なニアカーネルベクトル設定本数検証にかかる時間（解法適用時間総計：連立一次方程式を実際に解いて反復回数・実行時間の計測を行ったときの総時間，抽出時間総計：ニアカーネルベクトルの抽出にかかった時間の総計）

5.5 予測手法付全レベル近似ニアカーネルベクトル抽出法（提案手法 2）

本節ではさらに，ニアカーネルベクトル設定本数（Algorithm 9 や Algorithm 10 における $extract\_number$）の適切な設定本数を予測する手法を検討する．

5.5.1 手法の概要

前章の実験結果や著者らによる先行研究 [19] により，著者らによる先行研究においての抽出手法（Algorithm 9）や提案手法 1（Algorithm 10）の手法において，ニアカーネルベクトルを適切に設定することで，収束性や実行時間において有用となる．しかし，適切なニアカーネルベクトル本数の検証には，抽出本数の取りうるすべてのパターンを網羅する必要がある．5.4.3 節より，現状では検証に膨大な時間を必要としてしまい，実用面からみて大きな問題となる．そこで本研究では，[17,18] にて用いられていた判定基準値を Algorithm 10 の手法に組み込み，ニアカーネルベクトルの抽出を行った [53]．具体的な計算式は以下の通りとなる．

$$\{(A_{level}\boldsymbol{e}_{level}, \boldsymbol{e}_{level})/(A_{level}\tilde{\boldsymbol{e}}_{level}, \tilde{\boldsymbol{e}}_{level})\}^{1/\mu}$$

ここで，$(\cdot, \cdot)$ の形で表されている式は，ベクトル同士の内積を示す．上記の式において，$\tilde{\boldsymbol{e}}$ は抽出時の V-cycle を適用する前，$\boldsymbol{e}$ は抽出のために V-cycle を μ 回適用した後のベクトルを示す．また，$level$ は level 番目の階層における行列やベクトルを示す．5.2 節で述べたように，Algorithm 9 のように V-cycle を適用することで，$M\boldsymbol{e} \approx \boldsymbol{e}$ となる成分が残り，これをニアカーネルベクトルとして階層行列の再構成を行い再び V-cycle の適用を行うことで，前回抽出されたニアカーネルベクトル成分 $\boldsymbol{e}$ が効率よく減衰し，収束性が改善する．この性質を利用し，十分なニアカーネルベクトルが抽出された，つまり停滞を引き起こすようなニアカーネルベクトル成分が十分抽出され，抽出時における $A\boldsymbol{e} = 0$ の求解による残差が早く減衰した場合，ニアカーネルベクトルの抽出を終了する．上記の判定基準値において，n 本目に抽出したニアカーネルベクトル $\boldsymbol{e}$ による行列ベクトル積 $A\boldsymbol{e}$ が，$n-1$ 本目 $\tilde{\boldsymbol{e}}$ による行列ベクトル積 $A\tilde{\boldsymbol{e}}$ よりも十分 0 に近くなれば，十分にニアカーネルベクトルが抽出されたと見なすことができ，その場合式の結果が 0 に近くなる．つまり，判定基準値の結果が十分小さい値を示した際に，これ以上のニアカーネルベクトルの追加による，収束性改善の効果が見込めないと判断できる．この際ニアカーネルベクトルの抽出を終了することで，過剰なニアカーネルベクトルの抽出を抑えられると期待できる．図 5.18 に，Algorithm 9 適用時の，行列 A と V-cycle 反復中における抽出中のニアカーネルベクトル $\boldsymbol{e}$（Algorithm 9 内での $V\_cycle^{\mu}(A\boldsymbol{e} = 0)$ は V-cycle を μ 回（本研究では 20 に設定）反復させるという意味であり，図 5.18 での反復回数はこの V-cycle 反復の 1 反復ごとの結果を意味する）との行列ベクトル積のノルム $||A\boldsymbol{e}||$ を示す．図 5.18 からわかるように，ニアカーネルベクトルの設定本数を増やすことによりニアカーネルベクトル成分が効率よく減衰され，残差のノルム $||A\boldsymbol{e}||$ が 0 に近くなっていることがわかる．このように，十分なニアカーネルベクトルが抽出された場合，抽出時の残差ノルムが 0 に近くなることが予想できるため，その時点で抽出を止めることがよいと考えられる．

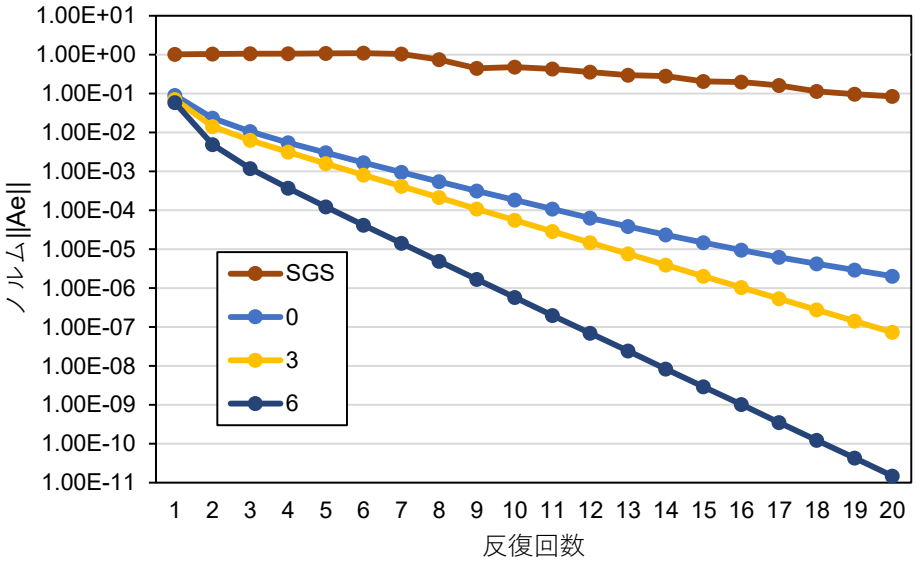


図 5.18: 抽出時における各反復での抽出するベクトル e と行列 A との積のノルム $||Ae||$ の結果（SGS : $A\boldsymbol{x} = 0$ を解くように対称 Gauss-Seidel 法を適用，0，3，6 : 先行研究（Algorithm 9）において，ニアカーネルベクトルをそれぞれ設定しない，3・6 本抽出したときに，次のニアカーネルベクトル候補となる 1・4・7 本目をそれぞれ抽出しようとしたときの結果）

Algorithm 11 に，予測手法を用いたニアカーネルベクトル抽出手法（提案手法 2）の概要を示す．Algorithm 11 の青字で示した箇所は Algorithm 10（提案手法 1）からの追加箇所である．Algorithm 11 において新たな値 ϵ がある．これは，ユーザー側が与えるパラメータとなっており，この値よりも判定基準値が小さい場合，そのレベルにおける抽出は十分であると判断し，次のレベルの抽出に移る．次節で，Algorithm 11 の手法による有用性を数値実験により示す．

5.5.2 実験内容

本節では，予測手法付きニアカーネルベクトル抽出手法（提案手法 2）における数値実験の結果を示す．実験環境は 5.4.2 節と同様であるため，割愛する．以下より，数値実験の内容とその結果を示す．

本実験では，提案手法 2 の有用性検証に向けた数値実験の内容について述べる．この実験ではまず予備実験として，反復回数と予測式の関係を，問題サイズ 10×10×10 の小規模な問題上において示す．次に，ウィークスケーリング方式による実験で，提案手法 2 の有用性を示す．ここで Algorithm 11 および 5.5 節で述べたように，提案手法 2 では閾値 ϵ を設定する必要がある．本研究では閾値設定の際，各レベルで異なる値を設定し，計測を行った．これについての具体的な設定数値を表 5.7 および表 5.8 に示す．表 5.7 および表 5.8 のように，本研究では階層行列の最大レベル数に応じて，閾値 ϵ の設定を変更している（本実験ではウィークスケーリング方式による計測であるため，プロセス数の増加とともにレベル数が増加する．今回の場合，1 並列および 8 並列では 3 レベル，64 並列および 216 並列では 4

Algorithm 11 予測手法付きニアカーネルベクトル抽出手法

Given : B_1
for $level = 1$ to $max\_level - 1$ **do**
  for $n = 1$ to $extract\_number$ **do**
    $\tilde{\boldsymbol{e}}_{level} \leftarrow Random()$
    $\boldsymbol{e}_{level} \leftarrow V\_cycle^{\mu}(A_{level}\tilde{\boldsymbol{e}}_{level} = 0)$
    if $(A_{level}\boldsymbol{e}_{level}, \boldsymbol{e}_{level})/$
    $(A_{level}\tilde{\boldsymbol{e}}_{level}, \tilde{\boldsymbol{e}}_{level})\}^{1/\mu} < \epsilon_{level}$ **then** break
    $B_{level} \leftarrow [B_{level}, \boldsymbol{e}_{level}]$
    $Multilevel\_creation(B_{level})$
  end for
end for
Output $B_1, B_2, ...$ matrix as near-kernel vectors

$Random()$：乱数生成
max_level：階層の最大レベル数
$extract\_number$：抽出したい本数
A_{level}：レベル $level$ における問題行列 A
$V\_cycle^{\mu}(A\boldsymbol{e} = 0)$：$A\boldsymbol{e} = 0$ を対象に V-cycle を μ 回適用
B_{level}：レベル $level$ におけるニアカーネルベクトル候補群の行列
$[B,\boldsymbol{e}]$：行列 B の最終列へのベクトル $\boldsymbol{e}$ の追加
$Multilevel\_creation(B_l)$：行列 B を基に，補間演算子 $P_l, P_{l+1}, ...$，および
階層行列 $A_l, A_{l+1}, ...$ を再生成
ϵ_{level}：判定基準値の閾値

レベル，512 並列は 5 レベルとなる）．さらに，レベル数が下がるごとに，閾値 ϵ が低くなるように設定を行っている．著者らによる研究から，細かいレベルは問題サイズが大きいため，ニアカーネルベクトル設定本数の増加により，計算コストが大きく増加し，結果として全体実行時間へ大きく影響してしまうこともわかっている．これを緩和するため表 5.7 および表 5.8 のように，本実験では粗いレベルにおいて，閾値 ϵ を低く設定することでニアカーネルベクトル抽出本数を増やし，ニアカーネルベクトル成分の減衰をより厳密に行うようにした．また，閾値 ϵ の値を高く設定することで，ニアカーネルベクトルの抽出本数が少なくなる．これにより，計算コストが減少するが，収束性が悪化することが考えられる．これを検証するために，表 5.7 は閾値 ϵ を低く設定，表 5.8 は高く設定しており，それぞれの閾値 ϵ による検証実験も行った．ここまで閾値 ϵ について述べてきたが，表 5.7 および表 5.8 のような設定は，以上のような経験則に基づいており，すべての問題で同じ設定が適用できるかは不明である．そのため，閾値 ϵ の設定手法の確立は今後の課題とする．

5.5.3 数値実験と結果

本節では，提案手法 2（Algorithm 11）における結果を示す．まず，予備実験における結果を図 5.19 に示す.「予測式」については，3p+n 本目を抽出した際に計算された時の予測

表 5.7: 各レベルにおける ϵ の値（全体実行時間優先）

	ϵ の値			
最大レベル数	Level 1	Level 2	Level 3	Level 4
3	0.250	0.156	—	—
4	0.400	0.250	0.156	—
5	0.640	0.400	0.250	0.156

表 5.8: 各レベルにおける ϵ の値（収束性優先）

	ϵ の値			
最大レベル数	Level 1	Level 2	Level 3	Level 4
3	0.100	0.070	—	—
4	0.143	0.100	0.070	—
5	0.204	0.143	0.100	0.070

式の結果を示している（例として 3p+1 における値は，1 本目を抽出した際に算出された値となっており，本実験では約 0.22 を示した）．図 5.19 より，3p+3 から 3p+4 にかけて，反復回数の改善が効率よく行われていないことがわかる．これより，ニアカーネルベクトルの設定本数増加により計算コストが増大することを考慮すると，本実験の問題設定では 3p+3 が適切な本数設定である．それと同時に，3p+3 から 3p+4 にかけて予測式の結果が大きく下がっていることがわかる．これらより，本実験の問題設定では，閾値 ϵ を 0.1 に設定することで，予測式による適切な本数の設定が可能となる．これは実際にすべてのパターンを計測したため判明した事実であり，5.5.2 節でも述べたように，閾値 ϵ の設定は今後の課題とし，次の実験では表 5.7 および表 5.8 の値を使用した．

次の実験では，実際に提案手法 2 を用いたウィークスケーリングにおける結果を示す．この実験による実行時間を図 5.20 および図 5.21 に示す．グラフの要素である「提案手法 1」は，提案手法 1 において図 5.12 や図 5.13 のように，1 レベル目を 7 本，2 レベル目以降を 5 本抽出するまでのすべて組み合わせを試し，実行時間においてすべての組み合わせにおける最小，最大，平均を算出したものとなる（つまり，（最小）は提案手法 2 における理想の結果であり，提案手法 2 を用いることで低コストで（最小）に近づくことが目的となる）．図 5.20 より，全体の実行時間を考え，構築部の実行時間を抑えるために閾値を高く設定した場合，全体の実行時間は従来（最小）とほぼ同等となることがわかる．しかし求解部の実行時間に着目すると，提案手法は従来（最小）と比べ悪化していることがわかる．また図 5.21 より，求解部の実行時間を考え，逆に閾値を低く設定した場合，全体の実行時間は提案手法 1 と比べ悪化しているが，求解部の実行時間は従来（最小）と同等か，さらに改善することがわかる．これらは，閾値を高く設定した場合，ニアカーネルベクトルの抽出本数を抑えるため，構築部の実行時間が改善するが，収束性が悪化し求解部の実行時間が増大するためである．逆に，低く設定するとその逆の効果が得られることがわかる．実際，表 5.9 からもその事実は明らかである．表 5.9 は提案手法 2 をそれぞれの閾値設定で実行した場合の反復回数を示している．この表より，上記で述べたように，閾値を高く設定した表 5.7 では反復回

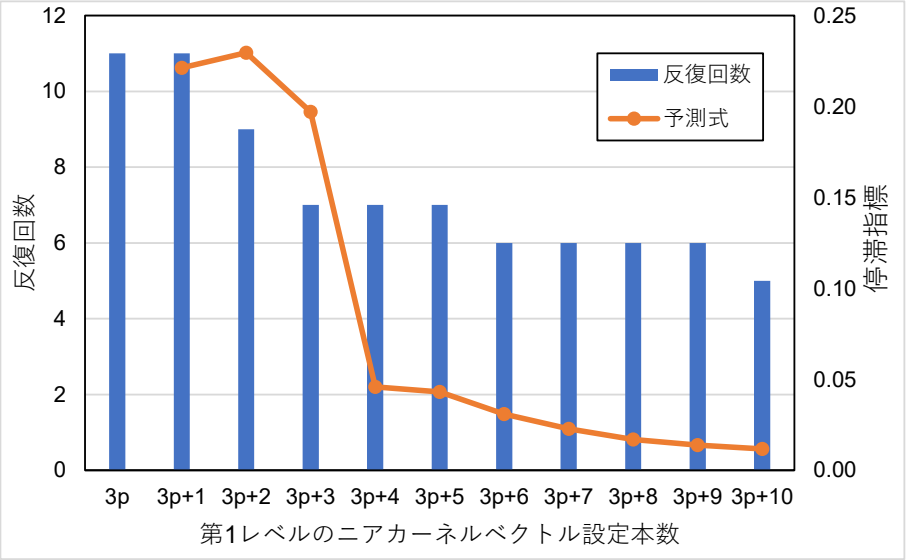


図 5.19: 予備実験：予測式と反復回数の結果（問題サイズ：10×10×10，1 並列）

数が多く，閾値を低く設定した表 5.8 は反復回数が少ないことがわかる．ここで，提案手法 2（閾値 ϵ 設定：表 5.8）における結果が従来（最小）よりも改善していることがわかる．これは，提案手法 1 の結果では 1 レベル目を 7 本，2 レベル目以降を 5 本抽出するまでの組み合わせまでのみしか検証していないが，提案手法 2 により，より多くのニアカーネルベクトルを使用するという判断をしたためである．このように提案手法 2 を用いることで，パターンが膨大であり通常では検証できない組み合わせにおいても，低コストで判断できることがわかる．ここで，前実験と同様に，適切なニアカーネルベクトルの設定本数検証の時間を図 5.22 に示す．図 5.22 より，提案手法 2 を用いることで，従来問題となっていたニアカーネルベクトルの本数設定に対する検証時間が約 90%減と，大きく削減されていることがわかる．これより提案手法を用いることで，実際に運用する際にも低コストで容易に運用可能となると考えられる．

以上より，本論文の提案手法 2 では，用途に応じて閾値 ϵ を適切に設定することで，従来手法と比べ低コストで改善がみられることがわかった．より詳細な内容としては，与えられた問題行列が解法適用時に毎回異なる場合には，表 5.7 のように閾値 ϵ を高く設定する，対して反復回数を多く必要とする悪条件の問題や，元の問題設定は変えずに右辺ベクトル $\boldsymbol{b}$ の設定を変更する（この場合，構築部での粗いレベルの問題作成を 1 回行えば，そのまますべての計算に同じ階層行列を使用できる）場合には，表 5.8 のように低く設定することで，本手法が高い効果を発揮できると考えられる．

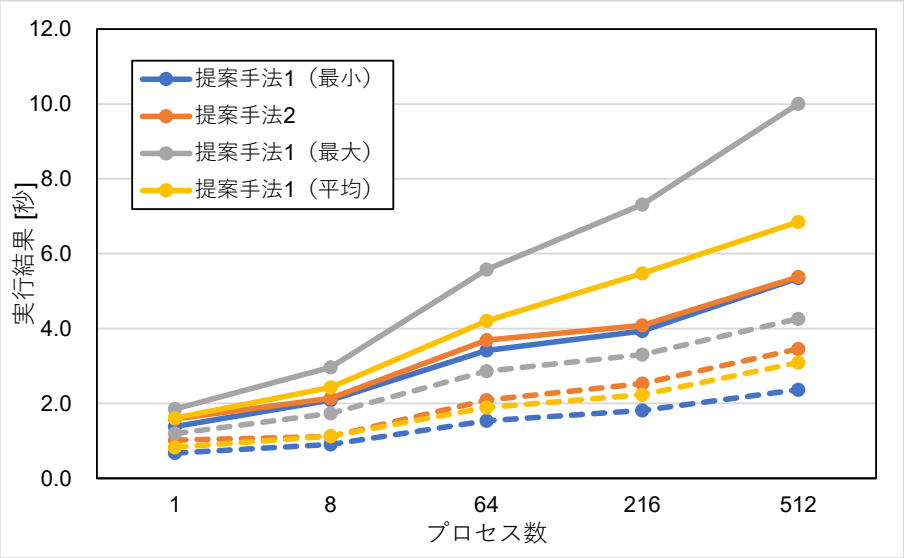


図 5.20: 提案手法 2（Algorithm 11）を用いた際の実行時間の結果（表 5.7 の閾値 ϵ を使用. 実線は全体の実行時間，構築部＋求解部，破線は求解部のみの時間を示す，提案手法 1：提案手法 1 を用いて全パターン網羅したときの最小・最大・平均の実行時間，提案手法 2：提案手法 2 を使用）

表 5.9: 提案手法 2（Algorithm 11）による反復回数

	1	8	64	216	512
提案手法 1（最小）	7	8	9	10	10
提案手法 2（閾値 ϵ 設定：表 5.7）	12	11	17	20	24
提案手法 2（閾値 ϵ 設定：表 5.8）	8	10	7	7	8

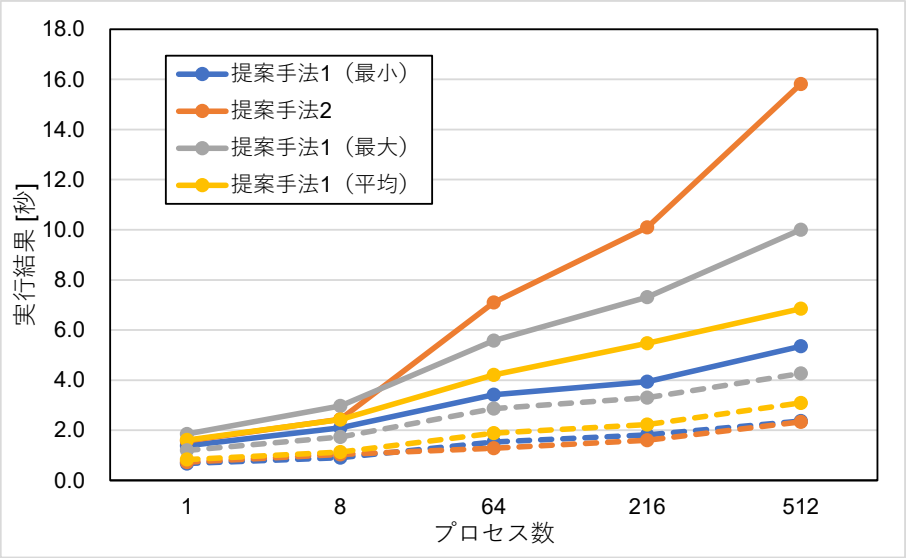


図 5.21: 提案手法 2（Algorithm 11）を用いた際の実行時間の結果（表 5.8 の閾値 ϵ を使用．グラフの要素は図 5.20 と同様）

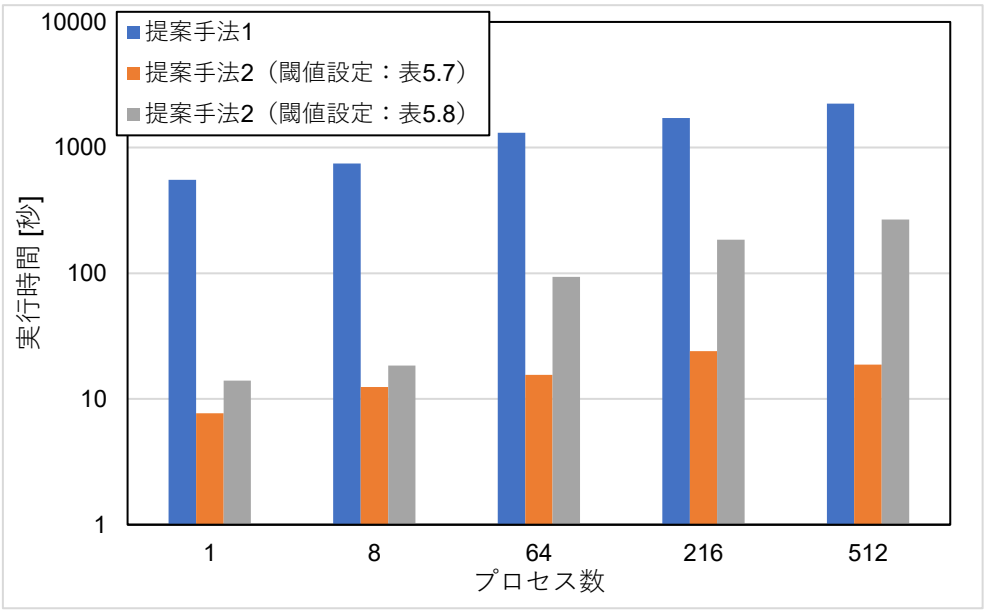


図 5.22: 提案手法 2（Algorithm 11）による適切なニアカーネルベクトル設定本数検証にかかる時間（提案手法 1：提案手法 1 にて全パターン網羅したときの実行時間，提案手法 2：提案手法 2 を使用）

5.6 固有値計算法を用いた粗レベル補間近似ニアカーネルベクトル抽出法（提案手法 3）

本節では，本研究で提案する 2 つめの抽出手法である，粗いレベルにおける固有値計算法を用いたニアカーネルベクトル抽出手法について説明する．

5.6.1 手法の概要

5.4 節や 5.5 節にて述べた手法では，V-cycle を用いてニアカーネルベクトルを 1 本づつ抽出する．そのため，抽出本数により抽出時間が増加してしまう．これにより，ニアカーネルベクトル抽出コストが，実際に解法を適用し連立一次方程式を解くコストと比べ，大きくなってしまう問題がある．

そこで，Bootstrap AMG 法 [48,49] と提案手法 1 を基に，任意の粗いレベルにおいて固有値解析を実施し補間を行い，ニアカーネルベクトルを複数階層において抽出，および設定する手法を提案し評価を行った [54]．概要を Algorithm 12 および Algorithm 13 に示す．まず，Algorithm 12 について説明する．Algorithm 12 は，レベル 1 のみを対象にニアカーネルベクトルを抽出する手法である．この手法ではまず，任意の最大レベル数以下の数 $\hat{L}$ を入力する（1 行目）．そして，そのレベルにおいて，固有値解析を実施する（4 行目）．その後，補間演算子 P 行列とスムーザを用いて，細かいレベルに移動する．粗いレベルにおいての抽出を行いたい場合は，Algorithm 12 に加え，Algorithm 13 の処理を行う．Algorithm 13 は，Algorithm 10（提案手法 1）に基づいており，粗いレベルにおけるニアカーネルベクトルの抽出を行う際には，まず Algorithm 12 において抽出されたニアカーネルベクトルを基に階層行列の再作成を行い，階層行列の更新を行う．次に粗いレベルのニアカーネルベクトルを算出するのだが，本手法では提案手法 1 とは異なり，補間演算子 R 行列のみを用いて行列行列積を行い，最終的に算出された各レベルの行列群 $\hat{W}_2, ..., \hat{W}_{\hat{L}}$ をニアカーネルベクトルとして出力する．

ここで，実際に提案手法 2（Algorithm 11）の抽出および検証時間と，収束までにかかった計算時間を比較したグラフを図 5.23 に示す．図 5.23 からわかるように，抽出および検証時間は従来よりも大幅に改善したものの，解法を適用して連立一次方程式を解く時間と比較すると，まだ抽出コストが大きいことがわかる．そこで上記で説明したように，粗いレベルにおいて固有値解析を実施することで，低コストで複数本のニアカーネルベクトルを抽出できると期待できる．また，粗いレベルにおいてもニアカーネルベクトルの抽出を行うことで，さらに高い収束性能を発揮できると考えられる．

5.6.2 実験内容

本実験では，1.3 節にて示した 3 次元弾性体問題と，5.2.2 節で用いた 2 次元定常熱伝導問の 2 つの問題に対し，それぞれ実験を行った結果を示す．

3 次元弾性体問題に対する実験では，1 プロセスと 64 プロセスの，2 つの環境下においてそれぞれ計測を行った．Algorithm 12 において，抽出の際に固有値解析を実施する必要があるが，今回の実験では，1 プロセス時においては Lapack ライブラリの固有値計算法の

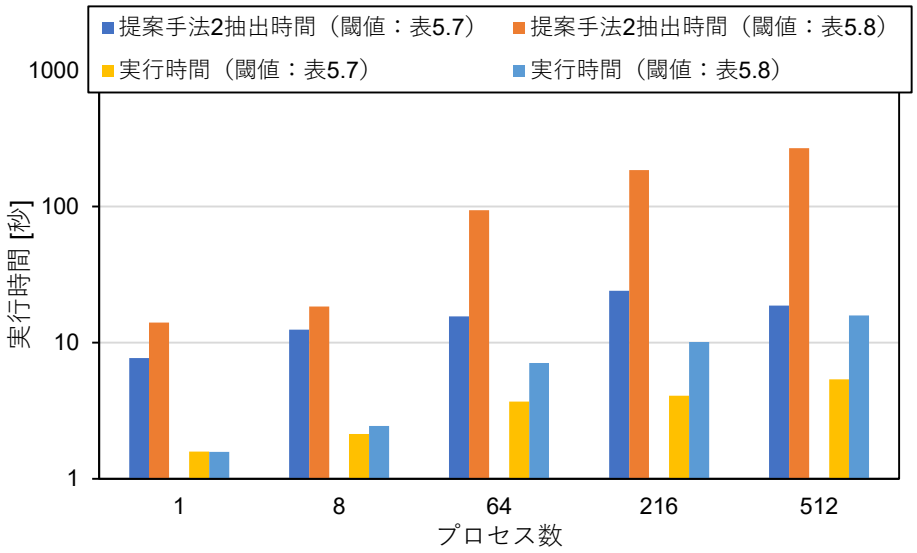


図 5.23: 5.5 節の実験における提案手法 2 の抽出および検証時間（提案手法 2 抽出時間）と SA-AMG 前処理付 CG 法の解法適用時間（実行時間）

関数（dsyev）[55] を用いた．また，64 プロセス時では，Scalapack ライブラリの固有値計算法の関数（pdsyev）[56] を適用した場合と，正定値対称な疎行列向けの固有値計算法である Lanczos 法を 300 反復適用した場合の 3 つにおいて，それぞれ実験結果を示す．また，2 次元定常熱伝導問題における実験では，1 プロセスのみ用いて計測を行い，Algorithm 12 の固有値計算法は Lapack を用いた．

5.6.3 数値実験と結果

本実験では 1.3 節で示した 3 次元弾性体問題に加え，5.2.2 節で用いた 2 次元定常熱伝導問題に適用したときの実験結果をそれぞれ示す．本実験における比較対象を表 5.10 に示す．本実験では比較のため，著者らの研究で従来用いていた抽出手法（Algorithm 9）と，提案手法 1（Algorithm 10）を適用した際の結果も示す．また，本実験では第 1 レベルのニアカーネルベクトルに関しては，表 5.11 に示すような本数の抽出および設定をそれぞれ行った．表 5.11 の 3p と 6p は弾性体問題の問題設定から予想されるニアカーネルベクトルである．また，第 1 レベルにて抽出したニアカーネルベクトルを用いる際には，3p に追加する形で設定を行っている（3p+1，3p+2，...）．「提案手法 1」（Algorithm 10）と「提案手法 3（粗いレベルあり）」（Algorithm 13）に関しては，粗いレベルでのニアカーネルベクトル設定本数によりさらに性能が変化する．そのため，実験結果を示す際には，粗いレベルで最良の本数を設定したときの結果を示している．

まず，1 並列時の結果を図 5.24 から図 5.26 に示す．3 つの図はそれぞれ各手法を用いた時の抽出時間，反復回数，構築部と求解部の実行時間を合計した全体実行時間と求解部のみの実行時間をそれぞれ示している．まず，図 5.24 より，従来手法である Algorithm 9 や Algorithm 10 では，抽出時間が本数とともに大きく増加してしまっていることがわかる．し

Algorithm 12 ニアカーネルベクトル抽出手法：提案手法 3（レベル 1 のみ）

Given : $\hat{L}$
for $l = \hat{L}$ to 1 **do**
  if $l == \hat{L}$ **then**
    $W_L = \{w_l^\kappa | A_l w_l^\kappa = \lambda_l^\kappa w_l^\kappa, \kappa = 1, ..., k_e\}$
  else
    $w_l^\kappa = P_{l+1}^l w_{l+1}^\kappa, \lambda_l^\kappa = \lambda_{l+1}^\kappa, \kappa = 1, ..., k_e$
    for $\kappa = 1$ to k_e **do**
      Relax on $A_l w_l^\kappa = 0$
    end for
  end if
end for
Output W_1 matrix as near-kernel vectors

$\hat{L}$：固有値解析の対象とする最大レベル以下の任意のレベル
A_l：レベル l における問題行列 A
P_{l+1}^l：レベル $l+1$ から l へ補間を行う Prolongation 行列

かし，提案手法 3 である Algorithm 12 を用いることで，抽出時間を少なく抑えることができ，3p+7 において約 90%程度の改善効果がみられることが分かった．表 5.12 に各レベルにおける未知数個数を示す．表 5.12 からわかるように，提案手法 3（Algorithm 12）において実際に固有値解析を実施しているレベル 2 では，本実験の問題設定においては未知数個数が 125 程度と小さくなる．そのため，抽出時間を低く抑えることが可能となる．また，図 5.25 と図 5.26 より，提案手法 3（Algorithm 12）は，著者らによる先行研究での抽出手法や提案手法 1 と比べ，ほぼ同等の収束性能や実行時間削減効果を示していることがわかる．これより，1 並列においては，今回の手法である Algorithm 12 や Algorithm 13 を用いることで，著者らによる先行研究での抽出手法や提案手法 1 と比べ低コストで比較的性質のよいニアカーネルベクトルの抽出が行えることがわかった．

次に，64 並列における結果を示す．5.6.2 節でも述べたように，本実験では Scalapack の関数（pdsyev）と Lanczos 法の 2 種類を適用したときの結果を示す．本実験では，抽出時間と収束性のみに着目する．実験結果を図 5.27 と図 5.28 に示す．図 5.27 と図 5.28 はそれぞれ抽出時間と反復回数を，各手法において実行した際の結果を示している．まず図 5.27 より，Scalapack を用いた手法は，著者らによる先行研究での抽出手法や提案手法 1 と比べ大きく抽出時間がかかってしまっていることがわかる．これは，本実験はウィークスケーリングなため，2 レベル目においても未知数個数が十分な粗さを確保できていないことに加え，Scalapack では密行列を対象としており，疎行列から密行列への変換および固有値解析自体に大きなオーバーヘッドがかかってしまったためである．一方，Lanczos 法による手法では，著者らによる先行研究での抽出手法や提案手法 1 と比べても抽出時間が低く抑えられていることがわかる．これは，Lanczos 法は疎行列向け固有値計算法であり，Scalapack を用いた場合と比べ低コストで抽出できたためである．次に，図 5.28 に着目すると，Lanczos 法を

Algorithm 13 提案手法 3 における粗いレベルのニアカーネルベクトル設定方法

$Given : \hat{L}$
$Given : W_1, ..., W_{\hat{L}}$
for $l = 1$ to $\hat{L}$ **do**
  $Multilevel\_creation(W_l)$
  $\hat{W}_{l+1} = R_l^{l+1} W_l$
end for
Output $\hat{W}_2, ..., \hat{W}_{\hat{L}}$ matrices as near-kernel vectors

$\hat{L}$：固有値解析の対象とする最大レベル以下の任意のレベル
L：最大レベル数
A_l：レベル l における問題行列 A
R_l^{l+1}：レベル l から $l+1$ へ縮約を行う Restriction 行列
$Multilevel\_creation(W)$：行列 W を基に，補間演算子 $P_l, P_{l+1}, ...$，および階層行列 $A_l, A_{l+1}, ...$ を再生成

表 5.10: 実験 1 における比較対象

表記	内容
著者らの先行研究	著者らによる先行研究の抽出手法（Algorithm 9）[19] を使用
提案手法 1	提案手法 1（Algorithm 10）を使用
提案手法 3（粗いレベルなし）	提案手法 3（Algorithm 12）を使用
提案手法 3（粗いレベルあり）	提案手法 3（Algorithm 12）＋ 粗いレベルでニアカーネルベクトルを設定 （Algorithm 13）

使用した手法では若干悪化しているものの，どの手法もほぼ同様の収束性能を示すことがわかった．本研究で用いた Lanczos 法は 300 反復適用した際の近似的な結果であり，Scalapack は QR 法を用いた厳密な固有値と固有ベクトルの解である．そのため，Lanczos 法を用いた手法にみられた収束性の悪化に関しては，Restart 付きの Lanczos 法などを用い解の精度を高めることで，さらに改善すると考えられる．

ここで，提案手法 3 を用いることによる抽出時間の有用性について，提案手法 1 や提案手法 2 と比較を行い示す．各提案手法の抽出時間を図 5.29 に示す．図 5.29 より，提案手法 3 を用いることで，提案手法 1 より約 99%，提案手法 2（閾値設定 : 表 5.7）より約 75%，提案手法 2（閾値設定：表 5.8）より約 90%のニアカーネルベクトル抽出コストの削減効果が得られることがわかった．以上より，提案手法 3 である Algorithm 12 や Algorithm 13 を用いることで，抽出手法を抑えつつ，著者らによる先行研究での抽出手法や提案手法 1 と同等の高い収束性能を発揮できることがわかった．また，今回は Algorithm 12 の固有値計算法と適用する箇所において，Lapack の固有値計算関数と Lanczos 法のみを用いたが，その他の手法を適用することによる結果の分析も必要であると考えられる．次の実験では，他問題への有用性の検証として，2 次元定常熱伝導問題における実験結果を示す．

次に，実験 2 の結果を示す．本実験は 2 次元定常熱伝導問題における実験結果を示す．本

表 5.11: 第 1 レベルのニアカーネルベクトル抽出本数

表記	内容
3p	平行移動成分 (X, Y, Z) (本数: 3)
6p	3p (本数: 3) + 回転成分 (X, Y, Z) (本数: 3)
3p+1,3p+2,...	3p (本数: 3) + レベル 1 で抽出された ニアカーネルベクトル (最大本数: 7)

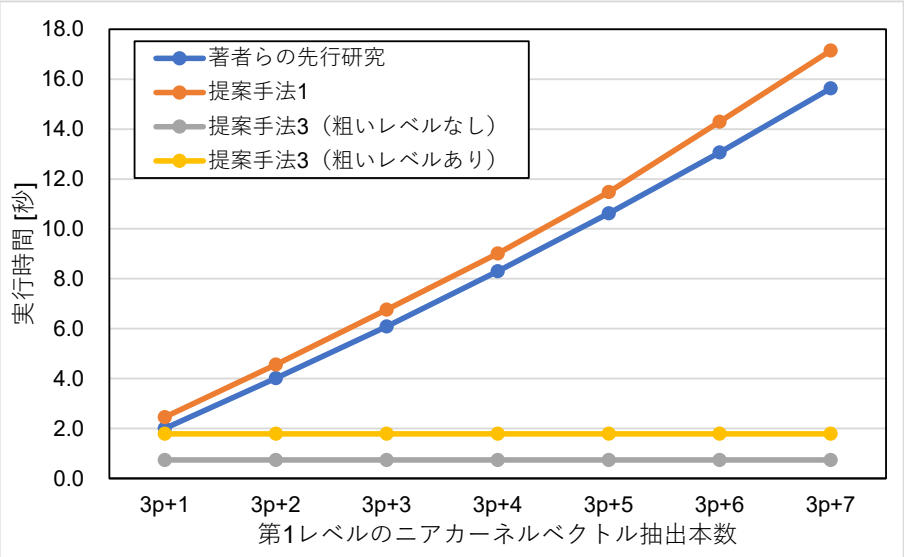


図 5.24: 実験 1：1 並列時の抽出時間（凡例の詳細は表 5.10 に記載）

実験では，1 コアのみを用いて実験を行った．また本実験では比較対象として，ポアソン方程式や拡散方程式のニアカーネルベクトルとして知られている定数ベクトル $(1, 1, ...)^T$ を，ニアカーネルベクトルとして設定した際の結果を 1p として記載している．結果を表 5.13 と表 5.14 に示す．表 5.13 はそれぞれの手法を用いた際の反復回数を示している．本実験では，提案手法 3（Algorithm 12）の手法における固有値解析を実施する対象のレベルに関して，レベル 3 において適用した結果も示している．不均質性が強い問題では条件数が大きくなり，それに伴い収束性も悪化する（これは 5.2.2 節にて示した）．しかし，いずれの手法においても，ニアカーネルベクトルを適切に設定することで，高い収束性を発揮していることがわかる．以上より，ニアカーネルベクトルを適切に設定することで収束性が改善し，

表 5.12: 実験 1：1 並列時の各レベルの未知数個数

Level	# of DOF
1	3375
2	125
3	6

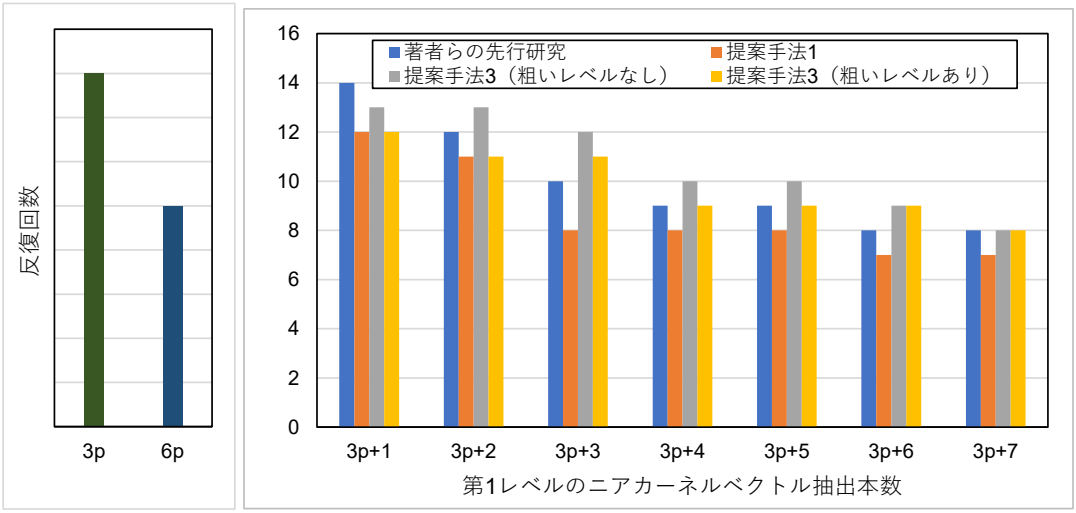


図 5.25: 実験 1：1 並列時の反復回数（内容は図 5.24 と同様）

特に不均質性が強い，条件数が大きい問題に対して高い効果を発揮することがわかる．次に表 5.14 に着目する．表 5.14 は，それぞれのニアカーネルベクトル抽出手法において，ニアカーネルベクトル抽出にかかった時間を示している．表 5.14 より，本実験では提案手法 3（Algorithm 12）による抽出時間が，著者らによる先行研究での抽出手法や提案手法 1 と比べ遅くなってしまっていることがわかる（最大 74 倍）．ここで，表 5.15 に着目する．表 5.15 は，各レベルにおける未知数個数を示している．表 5.15 と表 5.12 のレベル 2 を比較するとわかるように，本実験の問題設定では，レベル 2 においても未知数個数が多い．そのため，固有値解析の計算時間を多く必要としたため，提案手法 3（Algorithm 12）の抽出時間が悪化している．しかし，提案手法 3（Algorithm 12）における固有値解析を実施するレベルを 1 つ下げ，レベル 3 において固有値解析を実施することで，抽出時間を約 99%削減できることがわかる．一方，表 5.13 より，反復回数が若干悪化していることもわかる．これは，レベルを 1 段下げたことで，問題行列の表現できる空間が小さくなり，それに伴い固有値解析による固有ベクトルの張る空間が小さくなったことで，与えられた問題行列における低周波なニアカーネルベクトル成分をとらえることができなくなったためであると考えられる．このように，提案手法 3（Algorithm 12）において固有値解析を実施するレベルによる，抽出時間と反復回数はトレードオフの関係があり，さらに分析する必要があると考えられる．

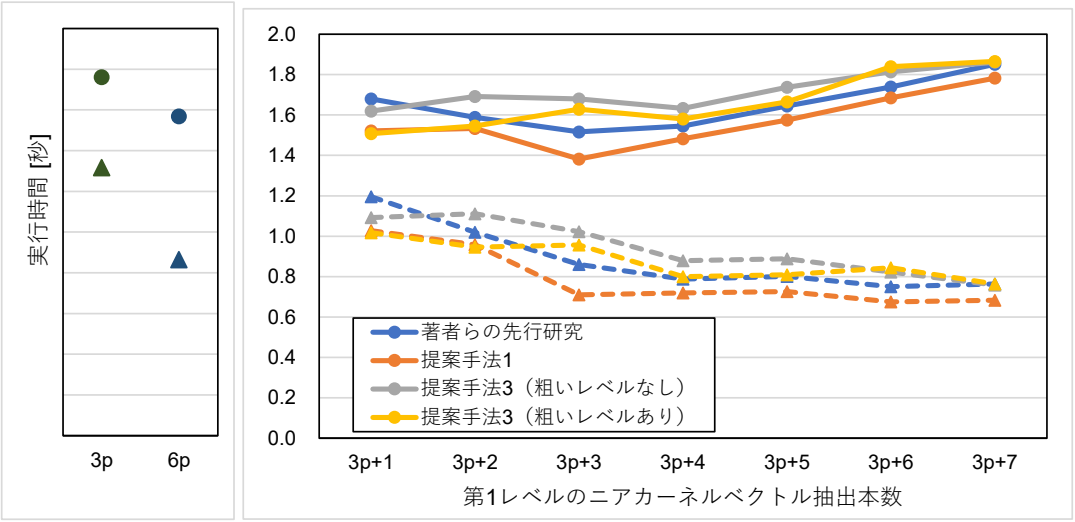


図 5.26: 実験 1：1 並列時の全体実行時間（下図における実線（丸マーカ）は全体（構築部+求解部），破線（三角マーカ）は求解部のみ，凡例の詳細は表 5.10 に記載）

表 5.13: 2 次元定常熱伝導問題：反復回数

	ニアカーネルベクトル設定本数				
	1p	1p+1	1p+2	1p+3	1p+4
著者らの先行研究	81	4	4	3	3
提案手法 1	—	4	4	3	3
提案手法 3（粗いレベルなし）	—	6	4	4	3
提案手法 3（粗いレベルあり）	—	6	4	4	3
提案手法 3（粗いレベルなし）（レベル 3 で固有値解法適用）	—	6	5	4	3

表 5.14: 2 次元定常熱伝導問題：抽出時間 [秒]

	ニアカーネルベクトル設定本数			
	1p+1	1p+2	1p+3	1p+4
著者らの先行研究	5.20	11.14	18.17	26.61
提案手法 1	19.72	30.10	43.24	57.58
提案手法 3（粗いレベルなし）	379.68	379.68	379.68	379.68
提案手法 3（粗いレベルあり）	385.36	385.36	385.36	385.36
提案手法 3（粗いレベルなし）（レベル 3 で固有値解法適用）	2.33	2.33	2.33	2.33

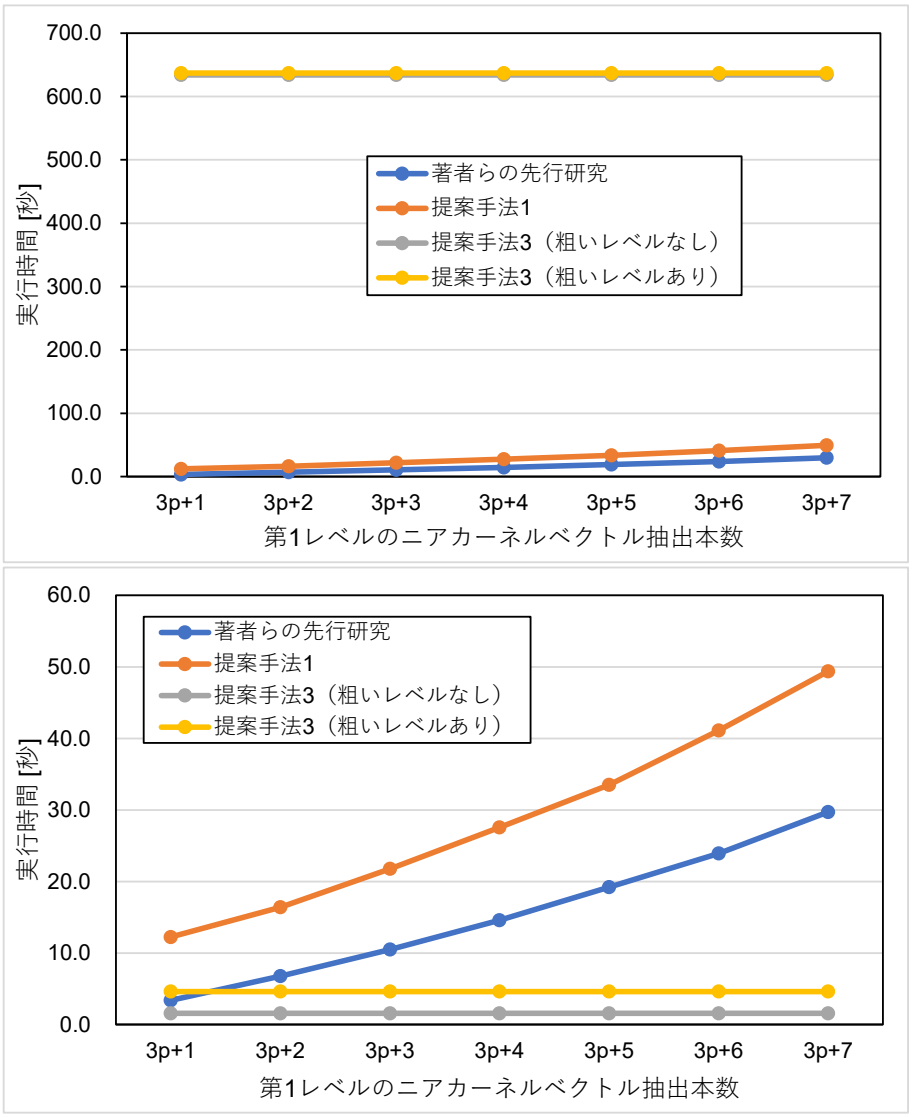


図 5.27: 実験 1：64 並列時の抽出時間（上：Scalapack の関数を使用，下：Lanczos 法を使用，凡例の詳細は表 5.10 に記載）

表 5.15: 実験 2：1 並列時の各レベルの未知数個数

Level	# of DOF
1	51200
2	8550
3	884
4	131
5	21

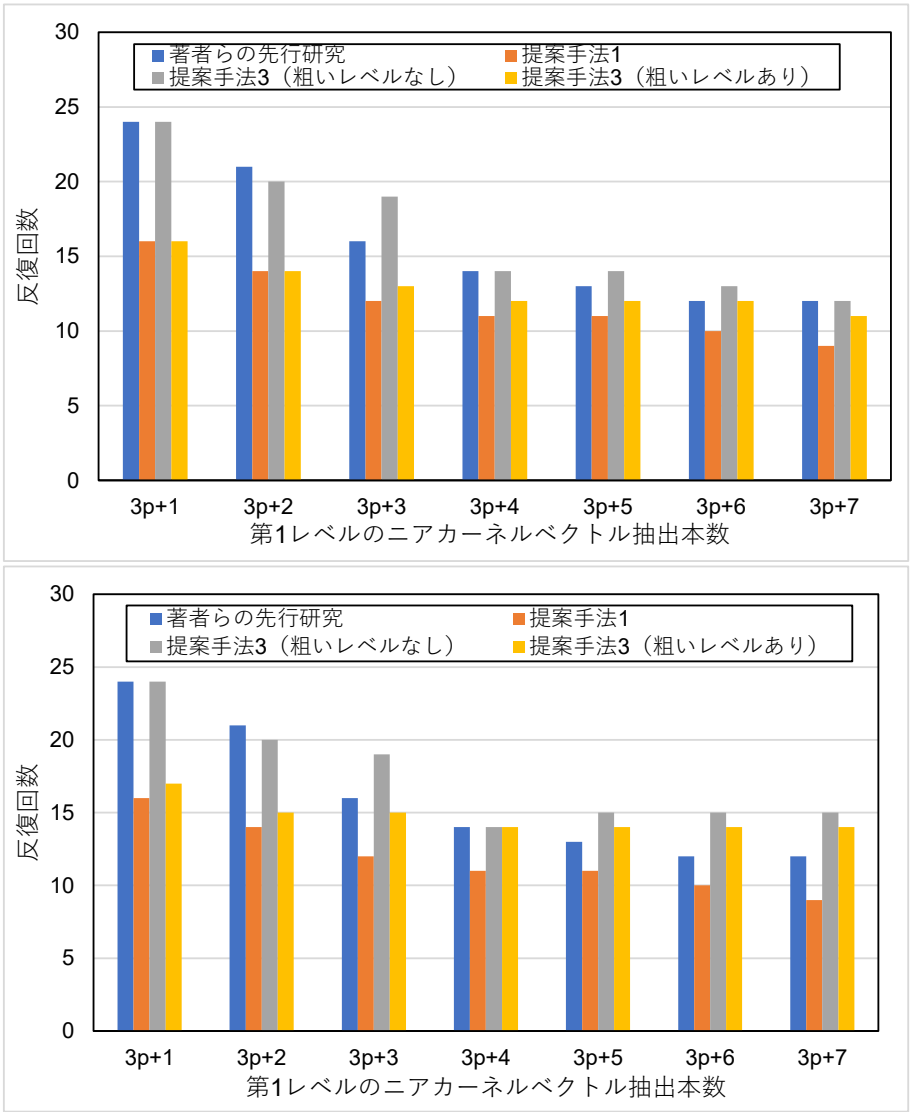


図 5.28: 実験 1：64 並列時の反復回数（上：Scalapack の関数を使用，下：Lanczos 法を使用，凡例の詳細は表 5.10 に記載）

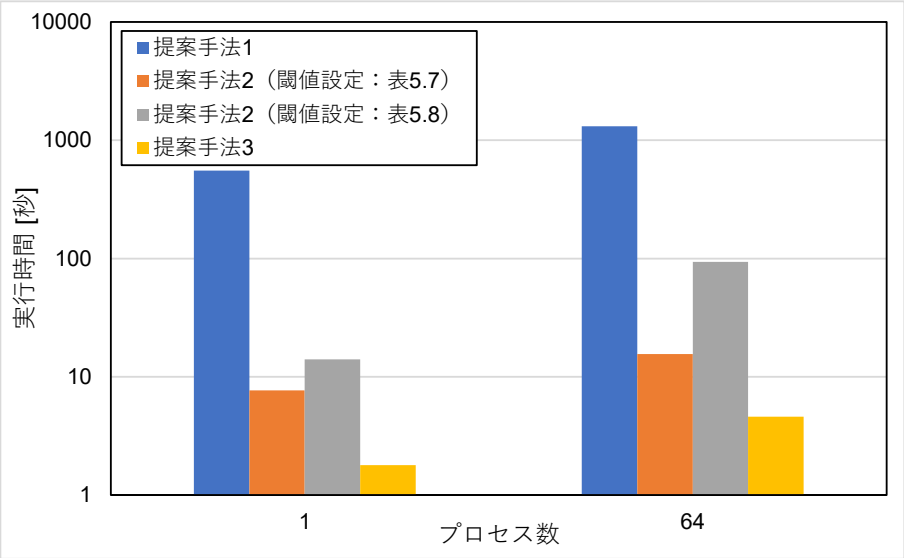


図 5.29: 本研究における各提案手法の抽出時間比較（提案手法 3 の 64 プロセスは内部で用いる固有値解析手法に Lanczos 法を用いた際の結果）

5.7 様々な問題への適用

ここまで，3 次元弾性体問題と 2 次元定常熱伝導問題を対象に実験を行ってきた．本章では，他問題に対する有用性の検証のため，Texas A&M 大学の SuiteSparse Matrix Collection [20] から取得した行列を用い，合計 7 個の問題で計測した．使用した問題を表 5.16，使用した問題行列の形状を図 5.30 に示す．表 5.16 は，行数の小さい順に問題を並べている．本実験では，構造・流体解析問題を中心に，様々な大きさや条件数の持つ問題行列を使用した．また，実験環境は 5.4 節と 5.6 節と同様のものを用いた．また，すべて逐次で実行した際の結果となっている．

本実験により得られた反復回数，構築部と求解部双方を足した全体の実行時間，求解部のみの実行時間，さらに抽出時間をそれぞれ表 5.17 から表 5.21 に示す．それぞれの表における要素において，SGS は対称 Gauss-Seidel 法を前処理に用いた CG 法である．また，他は SA-AMG 法を前処理とした CG 法において，抽出なしはニアカーネルベクトルとしてすべての要素が 1 の定数ベクトルを使用した際の結果，提案手法 1 は Algorithm 10，提案手法 3 は Algorithm 12 と Algorithm 13 の手法を用いてニアカーネルベクトルの抽出を行った時の，最良の結果となっている．まず，表 5.17 において，提案手法を用いることで，他のニアカーネルベクトルを抽出しない手法と比べ改善されていることがわかる．これは，ニアカーネルベクトル抽出による収束性削減の効果によるものである．次に，表 5.18 に着目すると，提案手法 3 が他の手法と比べ比較的良い結果を示すことがわかる．これは，表 5.19 や表 5.21 より，収束性改善に伴う解法適用時間の削減も要因のひとつであるが，とりわけ提案手法 3 の抽出時間の改善効果が大きく，これが表れたものであると考えられる．ここで，表 5.19 と表 5.20 に着目すると，問題サイズが比較的小さい問題においては，提案手法の実行時間に優位性があまり見られないことがわかる．これは，1 反復の計算量が小さいために，収束性改善の効果よりも，ニアカーネルベクトル設定に伴う計算コストのほうが大きくなってしまったためであると考えられる．例外として，ex10 は提案手法が有用であるが，これは反復回数が 85%から 90%と大きな削減効果を示しており，結果として計算コスト増加よりも改善効果が大きくなったためであると考えられる．また，比較的大きい問題においては，提案手法が優れており，特に af_shell1 においては，SGS を前処理とした単純な CG 法では収束しなかったが，提案手法を用いることで，大きく反復回数を削減することに成功している．また，今回の 2 つの提案手法で比較を行うと，基本的に提案手法 1 のほうが提案手法 3 よりも反復回数や実行時間の面で有用であることがわかる．しかし抽出時間に着目すると，表 5.21 からわかるように，提案手法 3 のほうが低く抑えられることがわかる．例外として af_shell1 は提案手法 3 のほうが抽出時間が遅くなっているが，これは af_shell1 ではレベル 2 においても十分に粗くならなかったため，固有値解析の実施に時間がかかってしまったためである．これは，固有値解析を実施するレベル（Algorithm 12 の $\hat{L}$）を今回は 2 に設定したが，これを 3 以降にする，または別の固有値計算法を用いるなどの対策をする必要がある．提案手法 3 の収束性は，固有値解析を実施したレベルにおける固有値分布に影響されていると考えられる．そこで以下より，固有値分布による分析を行った結果を示す．

ここで，提案手法 3 についての分析を行うため，問題サイズが最も小さい ex10 と 2 番目に小さい bcsstk28 において，1 レベルめと 2 レベルめの固有値分布を図 5.31 と図 5.33 に示

表 5.16: 使用した問題リスト

問題名	# of rows	# of nnz	kind	条件数
ex10	2,410	54,840	Computational Fluid Dynamics Problem	9.10e+11
bcsstk28	4,410	219,024	Structural Problem	9.45e+08
s1rmq4m1	5,489	262,411	Structural Problem	1.81e+06
fv2	9,801	87,025	2D/3D Problem	8.81e+00
Pres_Poisson	14,822	715,804	Computational Fluid Dynamics Problem	2.04e+06
s3dkq4m2	90,449	4,427,725	Structural Problem	-
af_shell1	504,855	17,562,051	Structural Problem Sequence	-

表 5.17: 各問題における各手法による反復回数（SGS：対称 Gauss-Seidel 前処理付 CG 法，抽出なし：ニアカーネルベクトルとしてすべての要素が 1 の定数ベクトルを使用，提案手法 1：Algorithm 10 を用いたときの最良値，提案手法 3：Algorithm 12+Algorithm 13 を用いたときの最良値）

問題名	SGS	抽出なし	提案手法 1	提案手法 3
ex10	369	63	36	48
bcsstk28	1,543	613	355	475
s1rmq4m1	282	120	29	72
fv2	16	5	3	3
Pres_Poisson	276	34	25	30
s3dkq4m2	3574	660	275	429
af_shell1	（収束せず）	418	86	157

す．提案手法 3 は，粗いレベルで固有値計算法を用いているため，粗いレベルの固有値分布により，抽出するニアカーネルベクトルの性質に影響することが考えられる．反復回数に着目すると，ex10 においては提案手法 1 が 36 回，提案手法 3 が 48 回と，提案手法 3 は提案手法 1 とほぼ同等の収束性を示すことがわかった．しかし，bcsstk28 においては提案手法 1 が 355 回，提案手法 3 が 475 回と，提案手法 3 の収束性が悪いことがわかる．そのうえで ex10 の固有値分布を示した図 5.31 に着目すると，2 レベルめでは，1 レベルめ最小固有値を完全には表現できていないが，ある程度近似できていることがわかる．しかし，bcsstk28 の固有値分布を示した図 5.33 に着目すると，2 レベルめの固有値は 1 レベルめの固有値をよく表現できていないことがわかる．これより，提案手法 3 は粗いレベルの固有値分布により，抽出するニアカーネルベクトルの性質に影響することがわかった．また，図 5.32 や図 5.34 より，性質のよいニアカーネルベクトルを設定することで，前処理後の固有値分布も改善することもわかる（特に図 5.34 でこの傾向がよく見られる）．今後の課題として，他の問題に対しても同じような分析を行い，同様の傾向がみられるかを検証する必要がある．

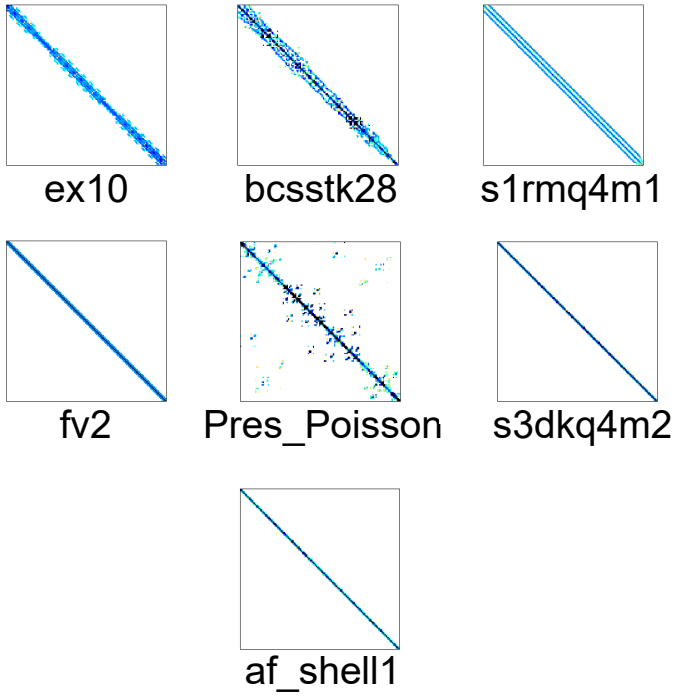


図 5.30: 使用した問題行列の形状図

表 5.18: 各問題における各手法による総実行時間（抽出部+構築部+求解部）

問題名	SGS	抽出なし	提案手法 1	提案手法 3
ex10	5.53e-01	3.23e-01	1.12e+00	2.98e-01
bcsstk28	7.63e+00	9.77e+00	8.76e+00	7.93e+00
s1rmq4m1	1.76e+00	2.56e+00	5.30e+00	1.94e+00
fv2	4.41e-02	1.02e-01	4.56e+00	8.87e-01
Pres_Poisson	4.36e+00	1.99e+00	1.21e+01	2.30e+00
s3dkq4m2	3.69e+02	2.41e+02	1.93e+02	1.82e+02
af_shell1	-	5.73e+02	4.36e+02	1.57e+03

表 5.19: 各問題における各手法による解法適用実行時間（構築部+求解部）

問題名	SGS	抽出なし	提案手法 1	提案手法 3
ex10	5.53e-01	3.23e-01	2.10e-01	2.54e-01
bcsstk28	7.63e+00	9.77e+00	5.92e+00	7.88e+00
s1rmq4m1	1.76e+00	2.56e+00	9.25e-01	1.87e+00
fv2	4.41e-02	1.02e-01	2.54e-01	2.52e-01
Pres_Poisson	4.36e+00	1.99e+00	1.84e+00	2.12e+00
s3dkq4m2	3.69e+02	2.41e+02	1.15e+02	1.76e+02
af_shell1	-	5.73e+02	1.49e+02	2.62e+02

表 5.20: 各問題における各手法による求解部実行時間

問題名	SGS	抽出なし	提案手法 1	提案手法 3
ex10	5.53e-01	3.02e-01	1.78e-01	2.29e-01
bcsstk28	7.63e+00	9.71e+00	5.78e+00	7.75e+00
s1rmq4m1	1.76e+00	2.46e+00	6.79e-01	1.65e+00
fv2	4.41e-02	5.50e-02	4.61e-02	4.56e-02
Pres_Poisson	4.36e+00	1.78e+00	1.40e+00	1.69e+00
s3dkq4m2	3.69e+02	2.392+02	1.10e+02	1.71e+02
af_shell1	-	5.68e+02	1.34e+02	2.46e+02

表 5.21: 各問題における各手法による抽出時間

問題名	提案手法 1	提案手法 3
ex10	9.08e-01	4.44e-02
bcsstk28	2.85e+00	5.17e-02
s1rmq4m1	4.37e+00	6.67e-02
fv2	4.31e+00	6.36e-01
Pres_Poisson	1.02e+01	1.72e-01
s3dkq4m2	7.80e+01	6.16e+00
af_shell1	2.87e+02	1.31e+03

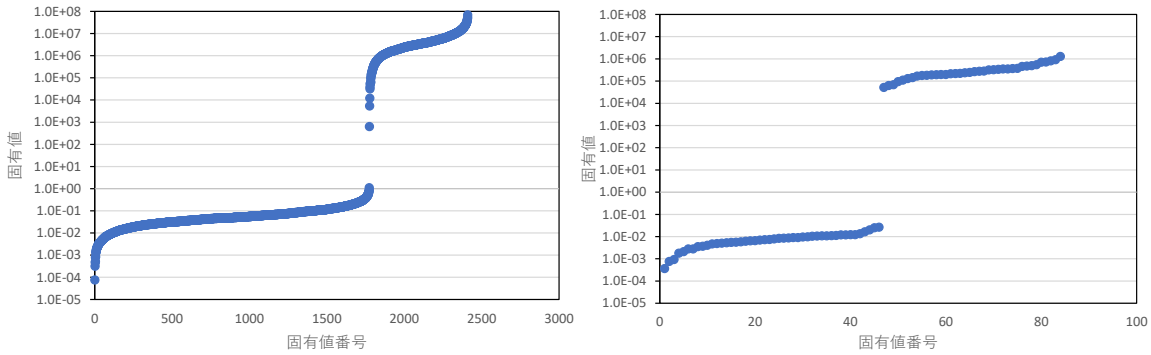


図 5.31: ex10 における各レベルの固有値分布（左：レベル 1，右：レベル 2）

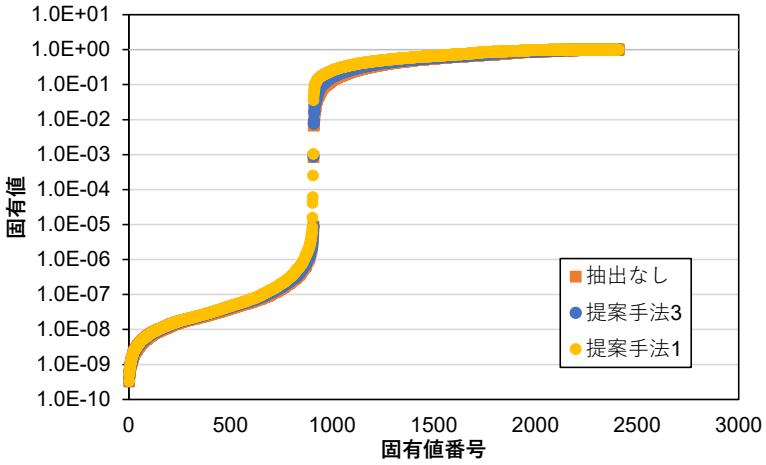


図 5.32: ex10 における前処理適用後の行列の固有値分布（抽出なし：ニアカーネルベクトルの抽出を行わない SA-AMG 法，提案手法 3：提案手法 3 を使用しニアカーネルベクトルの抽出を行った SA-AMG 法，提案手法 1：提案手法 1 を使用）

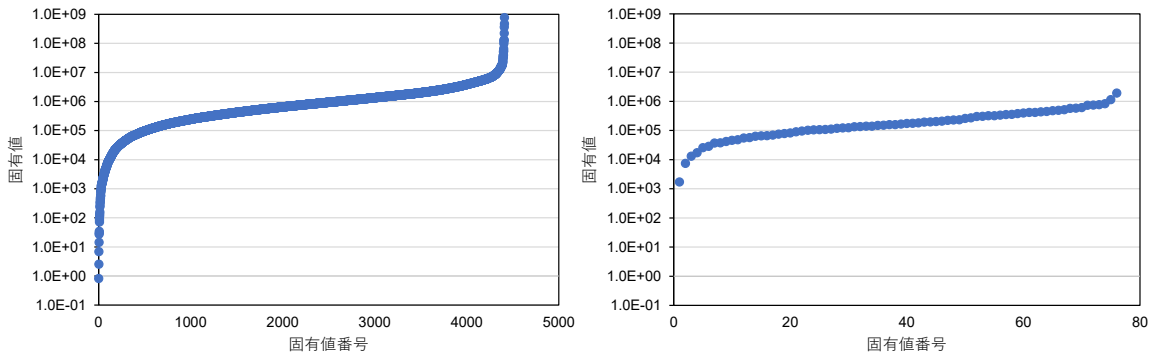


図 5.33: bcsstk28 における各レベルの固有値分布（左：レベル 1，右：レベル 2）

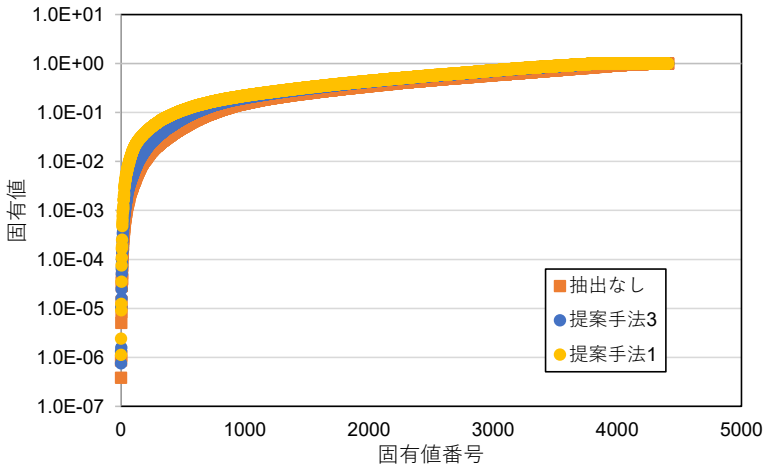


図 5.34: bcsstk28 における前処理適用後の行列の固有値分布（抽出なし：ニアカーネルベクトルの抽出を行わない SA-AMG 法，提案手法 3：提案手法 3 を使用しニアカーネルベクトルの抽出を行った SA-AMG 法，提案手法 1：提案手法 1 を使用）

5.8 抽出したニアカーネルベクトルの妥当性検証

ここまでで，ニアカーネルベクトルの新たな抽出手法の提案を行い，有用性について実験を交え示してきた．本節では，本研究において提案した抽出手法により抽出されたニアカーネルベクトルが，真のニアカーネルベクトルを適切に表現できているか，および収束性への影響の分析を行った結果を示す．本検証を行うにあたり，以下のような 2 つの実験を行った（それぞれニアカーネルベクトル検証実験 1，2 とする）．

1. cos 類似度による類似度計算
2. 抽出されたニアカーネルベクトルに含まれるエラー成分の算出

まず，実験 1 である cos 類似度による類似度計算実験の内容を説明する（cos 類似度は，あるベクトル $\boldsymbol{a}$ と $\boldsymbol{b}$ に対し，$cos\theta = (a,b)/||a||||b||$ により算出される値を示す）．ここで，以下より行う実験ではすべて，与えられた問題行列から算出された固有ベクトルを真のニアカーネルベクトルとし，実験を行っている（固有ベクトル算出には Lapack ライブラリの dsyev 関数 [55] を使用，固有ベクトルは対応する固有値の小さい順に並ぶ）．手順は以下の通りである．

1. 問題行列から固有ベクトルを固有値計算ライブラリにより算出
2. 抽出したベクトルと算出された固有ベクトルそれぞれにおいて，すべての取りうるパターンに対して cos 類似度を計算する．
3. cos 類似度が最も 1 に近いものを算出

上記の手順により，小さい固有値に対応する固有ベクトルに対して，cos 類似度が最も 1 に近くなれば，ニアカーネルベクトル成分を適切に表現できているとみなすことができる．つまり，ニアカーネルベクトル抽出手法により，理想的なベクトルが抽出できているとみることができる．

以下より，本実験による実験結果を示す．実験環境は 5.4 節や 5.6 節と同様のものを用いた．また，1.3 節に示した 3 次元弾性体の問題に対し，逐次環境で実験を行っている．はじめに，提案手法 1 において上記の実験を行った際の結果を表 5.22 および図 5.35 に示す．まず表 5.22 の項目「最類似固有ベクトル番号」に着目すると，比較的小さい番号を示していることがわかる．これより，抽出したニアカーネルベクトルが 0 固有値に近い固有ベクトル，つまりニアカーネルベクトル成分を適切に表現できていると考えることができる．また，反復回数および図 5.35 にも着目すると，cos 類似度が 1 に近い値を示した場合において，より収束性の改善がみられることもわかった．例として，表 5.22 の 3p+2 や 3p+3 は，他と比べ cos 類似度が 1 に近く，同様に図 5.35 の 3p+2 や 3p+3 に着目すると，3p+1 や 3p+2 からの収束性が大きく改善していることがわかる．以上のように，抽出したニアカーネルベクトルと固有ベクトルの cos 類似度と収束性改善には，ある程度の相関があることがわかった．また，cos 類似度が 0.45 程度であっても，ある程度の収束性改善効果がみられることもわかった．しかし，表 5.22 の項目「最類似固有ベクトル番号」において，同じ数値が算出

表 5.22: ニアカーネルベクトル検証実験 1 結果：提案手法 1（最類似固有ベクトル：最も cos 類似度が 1 に近かった固有ベクトルの番号（固有ベクトルは対応する固有値の小さい順に並ぶ，cos 類似度：算出された cos 類似度，反復回数：抽出されたニアカーネルベクトルを SA-AMG 前処理付 CG 法に用いた際の収束まで要した反復回数））

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
最類似固有ベクトル番号	6	3	1	7	7	8	1
cos 類似度	0.23	0.82	0.92	0.15	0.41	0.46	0.64
反復回数	14	12	10	9	9	8	8

表 5.23: ニアカーネルベクトル検証実験 1 結果：提案手法 3）

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
最類似固有ベクトル番号	1	2	4	10	5	8	6
cos 類似度	0.99	0.98	0.99	0.09	0.92	0.86	0.95
反復回数	13	13	12	10	10	9	8

されている箇所が見受けられる．これは，同じようなニアカーネルベクトルが（今回の実験による数字上では）抽出されてしまっていることが考えられる．これに関して，抽出したニアカーネルベクトル成分の詳細な分析を行う必要がある．

次に，提案手法 3 において本実験を行った結果を示す．実験結果を表 5.23 および図 5.36 に示す．まず表 5.22 に着目すると，提案手法 1 における実験と同様に，「最類似固有ベクトル番号」が比較的小さい値を示しており，また，cos 類似度に関しても 1 に近い成分が多く，ニアカーネルベクトル成分をよく表現できていることがわかる．しかし，表 5.23 の反復回数および図 5.36 に着目すると，提案手法 1 における実験においてみられた「cos 類似度」と反復回数における相関が，みられないこともわかった．例として，3p+3 において cos 類似度が 0.99 となっており，反復回数が 3p+2 から 3p+3 にかけて 13 回から 12 回となっている．対して，3p+4 においては cos 類似度が 0.09 と悪化しているが，反復回数は 3p+3 から 3p+4 にかけて 12 回から 10 回と，反復回数削減効果が高いことがわかる．これに関しては，抽出したニアカーネルベクトルに含まれる成分のなかで，単一の成分だけでなく，含まれる様々な成分がそれぞれ効率的に作用したためであると考えられる．このことを含めニアカーネルベクトルの検証を行うために，次の実験では，抽出したニアカーネルベクトルが表現する空間に着目した検証実験を行った結果を示す．

次に，実験 2 の抽出されたニアカーネルベクトルに含まれるエラー成分の算出についての説明を行う．式 (3.10)（$||\boldsymbol{e}+P\boldsymbol{v}||_A = \min_w||\boldsymbol{e}+P\boldsymbol{w}||_A$）より，真のニアカーネルベクトル（$\boldsymbol{v}_1, \boldsymbol{v}_2, ...$ とする）と抽出されたニアカーネルベクトル（$\boldsymbol{e}_1, \boldsymbol{e}_2, ...$ とする）は，必ずしも $\boldsymbol{v}_1 \approx \boldsymbol{e}_1$,$\boldsymbol{v}_2 \approx \boldsymbol{e}_2$ のように一致することは必要ではなく，

$$\boldsymbol{e}_1 \in \mathrm{span}(\boldsymbol{v}_1, \boldsymbol{v}_2, ...) \tag{5.1}$$

のように，真のニアカーネルベクトルが張る空間に抽出されたニアカーネルベクトルが所属すればよいと予想できる．式 (5.1) の仮定を基にすると，抽出されたニアカーネルベクト

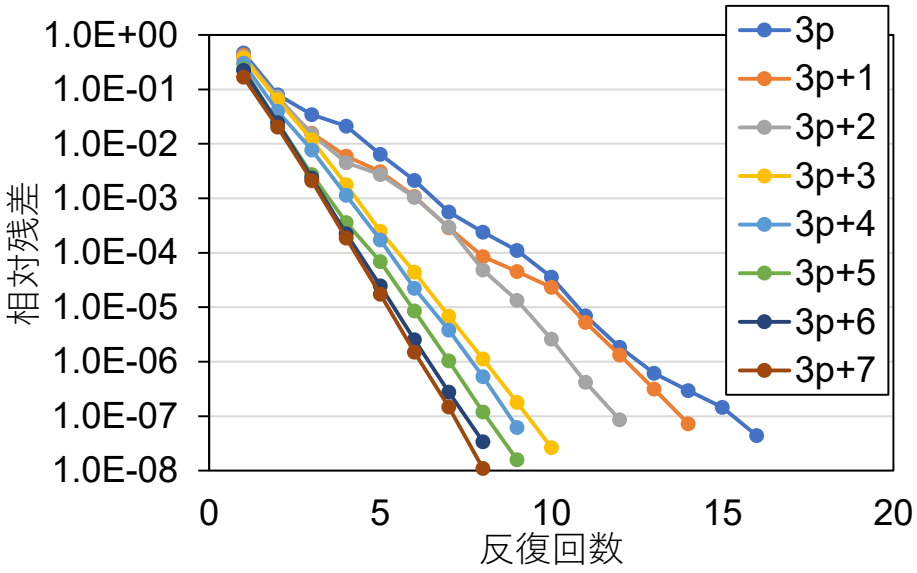


図 5.35: 抽出したニアカーネルベクトルを用いた際の相対残差履歴：提案手法 1

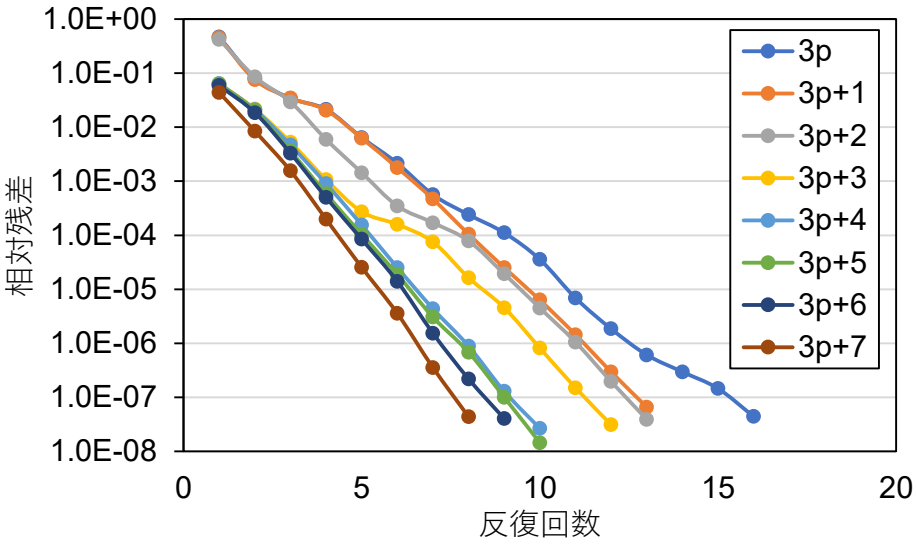


図 5.36: 抽出したニアカーネルベクトルを用いた際の相対残差履歴：提案手法 3

ルは，

$$\begin{aligned} \boldsymbol{e}_1 &= c_1^1\boldsymbol{v}_1 + c_2^1\boldsymbol{v}_2 + ... + c_k^1\boldsymbol{v}_k + \boldsymbol{r}_1 \\ \boldsymbol{e}_2 &= c_1^2\boldsymbol{v}_1 + c_2^2\boldsymbol{v}_2 + ... + c_k^2\boldsymbol{v}_k + \boldsymbol{r}_2 \end{aligned} \tag{5.2}$$

（ただし，$k, c_1^1, c_1^2, ...$ は適切な変数，$\boldsymbol{r}_1$ はベクトル $\boldsymbol{v}_k$ までの線形結合で表せなかった残りの成分）のように，線形結合で表すことができる．ここで，真のニアカーネルベクトル $\boldsymbol{v}$ はそれぞれ独立かつ正規化されていると仮定すると，変数 c は，

$$\begin{aligned} c_1^1 &= \boldsymbol{e}_1\boldsymbol{v}_1 \\ c_2^1 &= \boldsymbol{e}_1\boldsymbol{v}_2 \end{aligned} \tag{5.3}$$

のように算出可能である．

式 (5.2) 内のベクトル $\boldsymbol{r}_1$ は，$\boldsymbol{e}_1 \in \mathrm{span}(\boldsymbol{v}_1, \boldsymbol{v}_2, ...)$ とならなかった余分な成分であり，抽出されたニアカーネルベクトル内に存在する適切でない成分である．そこで，本実験ではベクトル $\boldsymbol{r}_1$ のノルムを算出することで，抽出されたベクトルが適切にニアカーネルベクトル成分を表現できているかの検証を行った．具体的には，式 (5.2) と式 (5.3) より，

$$\boldsymbol{r}_1 = \boldsymbol{e}_1 - (c_1^1\boldsymbol{v}_1 + c_2^1\boldsymbol{v}_2 + ... + c_k^1\boldsymbol{v}_k) \tag{5.4}$$

であることがわかる．本実験では，式 5.4 の 2 ノルムをとり，抽出されたニアカーネルベクトル内に存在する適切でない成分 $\boldsymbol{r}_1$ の大きさを算出することで，妥当性の検証を行った（以下，$\boldsymbol{r}_1$ のノルムをエラーノルムとする）．ここで，変数 k は式 (5.1) における空間の次元数を示しており，k によって比較対象である真のニアカーネルベクトルのなす空間が決定する．そのため，本実験の結果は変数 k により変化することが考えられるが，本研究ではこのことについての分析は行わず，k を 20 と 30 で設定し実験を行った．

以下より，本実験による実験結果を示す．まず，提案手法 1 における実験結果を示す．表 5.24 に本実験の結果を示す．表 5.24 と図 5.35 に着目すると，エラーノルムが少ないほど，収束性の改善効果が大きく見えることがわかる．例えば 3p+3 から 3p+4 においては，エラーノルムが k が 20 と 30 の場合双方とも大きく増加しており，それに伴い収束性の改善も悪化していることがわかる．これより本実験においては，式 (5.4) により計算されたエラーノルムと収束性に，相関があることがわかった．

次に，提案手法 3 における実験結果を示す．表 5.25 に本実験の結果を示す．表 5.25 と図 5.36 から，提案手法 1 での検証実験にみられたような強い相関性はみられないという結果となった．エラーノルムは k の値により変化することから，そこで，様々な k により実験を行い，有意な特徴がみられるかの実験を行った．実験結果を表 5.26 に示す．表 5.26 より，本実験においては k を 10 に設定した場合に，比較的収束性とエラーノルムの相関性がみられることがわかった．しかしニアカーネルベクトルが張る空間と収束性への影響についての分析は不十分であり，今後の課題としてそれらの具体的な相関について，さらなる分析が必要であると考えられる．また，変数 k の設定，および本実験では 3 次元弾性体問題のみ扱ったが，他の問題に適用した際の分析なども必要であると考えられる．

表 5.24: ニアカーネルベクトル検証実験 2 結果：提案手法 1（抽出されたニアカーネルベクトルに含まれるエラー成分の算出）

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
エラーノルム（$k = 20$）	2.77e-02	2.78e-02	2.21e-02	9.57e-02	1.27e-01	1.27e-01	2.59e-01
エラーノルム（$k = 30$）	1.76e-02	2.39e-02	1.50e-02	7.20e-02	7.50e-02	9.82e-02	1.45e-01
反復回数	14	12	10	9	9	8	8

表 5.25: ニアカーネルベクトル検証実験 2 結果：提案手法 3

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
エラーノルム（$k = 20$）	3.73e-02	3.83e-02	2.59e-02	6.80e-02	7.25e-02	9.19e-02	9.16e-02
エラーノルム（$k = 30$）	2.77e-02	2.92e-02	2.47e-02	6.59e-02	5.10e-02	6.88e-02	5.79e-02
反復回数	13	13	12	10	10	9	8

表 5.26: ニアカーネルベクトル検証実験 2 結果（k の設定による実験結果）：提案手法 3

	3p+1	3p+2	3p+3	3p+4	3p+5	3p+6	3p+7
$k = 10$	4.22e-02	4.26e-02	3.36e-02	9.65e-02	1.72e-01	3.27e-01	1.96e-01
$k = 20$	3.73e-02	3.83e-02	2.59e-02	6.80e-02	7.25e-02	9.19e-02	9.16e-02
$k = 30$	2.77e-02	2.92e-02	2.47e-02	6.59e-02	5.10e-02	6.88e-02	5.79e-02
$k = 100$	1.58e-02	1.66e-02	1.24e-02	3.95e-02	3.15e-02	3.17e-02	3.63e-02

5.9 関連研究

SA-AMG 法において，どのようなニアカーネルベクトルを設定するかを決めることは，SA-AMG 法において重要なこととなる．通常，このニアカーネルベクトルの設定に関しては，問題の性質から予想されるベクトルを用いることで収束性の改善を図ることが多い [24]．[24] では，薄板弾性体問題 [57] において，SA-AMG 法の適用とその結果が報告されている．[24] における研究では，回転成分 $(0,-z,y),(z,0,-x),(-y,x,0)$（$(x,y,z)$ は節点要素の座標）がニアカーネルベクトルとして用いられている．また，[58] において 3 次元弾性体問題に対し，*Hypre* と呼ばれるライブラリ内の SA-AMG 法を用いて，5.4.3 節の 3p と 6p のように問題設定からわかるニアカーネルベクトルを設定したときの結果と有用性が示されている（この研究での弾性体問題の設定は，厳密には本研究と異なる．具体的には，3d elasticity problem with intersecting slide surfaces と呼ばれる問題を使用し，ヤング率・ポアソン比に関しては明記されていない）．[58] では，問題サイズを 66,404 に設定したものに対し，ニアカーネルベクトルを 3 本（3p と同等）と 6 本（6p と同等）を用いており，それぞれ反復回数に関しては 51 回と 48 回という結果が報告されている．5.4.3 節の結果と比較すると，本研究の問題設定は比較的容易なため，本研究の弾性体問題を解いた際の反復回数は [58] と比べ少ない．そのため，今後の課題として，[58] で示されているようなより悪条件な問題に対して適用することを考えている．

ここまでで，関連研究において，問題設定から予想されるニアカーネルベクトルを設定することによる有用性の報告例を示した，しかし，このようなベクトルのみで，十分なニアカーネルベクトルが設定できているとは限らない．また，これは問題設定に依存したものであり，すべての問題において，対象とする問題の物理的性質に基づき適切なニアカーネルベクトルを予測できるとは限らない．そこで，ニアカーネルベクトルを問題行列から抽出する手法が提案されてきた．この手法により，上記のようなニアカーネルベクトル設定の際に起こる問題を解消することができると考えられる．

ニアカーネルベクトルを抽出する手法に関しての関連研究はいくつか存在する．まず，M. Brezina らにより提案された αSA 法である [17,18]．提案手法 1 や提案手法 2 は，各レベルでそれぞれニアカーネルベクトルの抽出を行ったが，αSA 法では 1 本のベクトルに基づき全階層における行列を網羅するように，ニアカーネルベクトルを抽出する．αSA 法の概要を Algorithm 14 に示す．Algorithm 14 内の $Multilevel\_creation(B_l)$ は，ニアカーネルベクトル群の行列 B_l を基に，レベル l 以下の粗いレベルにおいて，5.1.1 節の Algorithm 8 で述べた補間演算子 $P_l, P_{l+1}, ...$ の生成，および生成された補間演算子に基づき階層行列 $A_l, A_{l+1}, ...$ の再生成を行う処理とする．このように，1 本のベクトルを基に各階層で抽出計算を行い，最終的に粗いレベルで算出されたベクトルを補完することで，ニアカーネルベクトルの抽出を行う，

また，Brandt A. らにより提案された Bootstrap AMG 法と呼ばれる手法がある [48,49]．この手法ではまず，Multigrid 法の構築部の処理により生成された階層行列において，最下層の行列に対し固有値解析を実施し，固有値と固有ベクトルを算出する．その後，補間演算子である Prolongation 行列を用いて，算出された最下層での固有ベクトルを細かいレベルに移動し，ニアカーネルベクトルとして出力する．Bootstrap AMG 法の概要を Algorithm 15

Algorithm 14 αSA 法

Given : B_1
Select : x_1
for $n = 1$ to $extract\_number$ **do**
  for $l = 1$ to $L - 1$ **do**
    $\tilde{x}_l \leftarrow V\_cycle^{\mu}(A_l x_l = 0)$
    $B_l \leftarrow [B_l, \tilde{x}_l]$
    $B_{l+1} \leftarrow P_l^T B_l$
    $x_{l+1} \leftarrow$ Last col. of B_{l+1}
    $Multilevel\_creation(B_{l+1})$
  end for
  for $l = L$ to 2 **do**
    $x_{l-1} \leftarrow P_{l-1} x_l$
  end for
end for
Output x_1 as near-kernel vector

$extract\_number$：抽出したい本数
A_{level}：レベル $level$ における問題行列 A
$V\_cycle^{\mu}(Ax = 0)$：$Ax = 0$ を対象に V-cycle を μ 回適用
B_{level}：レベル $level$ におけるニアカーネルベクトル候補群の行列
$[B,\boldsymbol{x}]$：行列 B の最終列へのベクトル x の追加
$Multilevel\_creation(B_l)$：行列 B を基に，補間演算子 $P_l, P_{l+1}, ...$，および
階層行列 $A_l, A_{l+1}, ...$ を再生成

に示す．Bootstrap AMG 法ではまず，最下層 L において，固有値解析を実施する（3 行目）．その後，補間演算子 P やスムーザを適用し（5〜9 行目），最上層であるレベル 1 のニアカーネルベクトルとして出力する．提案手法 3 では任意の粗いレベルで固有値解析を実施し，その後補間を行い各階層で算出されたベクトルを，各階層のニアカーネルベクトルとして用いた．Bootstrap AMG 法ではこのように，最下層で固有値解析を実施し最上層まで補間を行い，最上層のニアカーネルベクトルとしてのみの抽出を行う．

そのほかにも，M.Brezina らにより提案された Spectral AMG 法と呼ばれる手法がある [59]．この手法では，まず有限要素法の単位要素から，その要素ごとにおける小行列を生成する．その後，生成された複数の小行列に対し，それぞれ 0 固有値に近い固有ベクトルを求める．最後に，以上のように各要素にて計算されたベクトルを，それぞれの要素番号に対応するように組み合わせ，ニアカーネルベクトルとする手法である．また，J. Xu らにより提案された Subspace correction がある [60,61]．この手法では，与えられた問題行列から部分行列を生成し，それぞれの部分行列上で問題を解く．また，固有値と固有ベクトルを近似的に効率よく解くことに着目すると，A. Stathopoulos らによる Deflation に基づき固有値と固有ベクトルを解く手法 [62] がある．この手法では，Lanzcos 法のように CG 法の手順を基に，反復内で小行列に縮約を行いそれに対して固有値計算法を用いることで，効率的に算

Algorithm 15 Bootstrap AMG

for $l = L$ to 1 **do**
  if $l == L$ **then**
    $W_L = \{w_l^\kappa | A_l w_l^\kappa = \lambda_l^\kappa T_l w_l^\kappa, \kappa = 1, ..., k_e\}$
  else
    $w_l^\kappa = P_{l+1}^l w_{l+1}^\kappa, \lambda_l^\kappa = \lambda_{l+1}^\kappa, \kappa = 1, ..., k_e$
    for $\kappa = 1$ to k_e **do**
      $Relax\ on\ (A_l - \lambda_l^\kappa T_l) w_l^\kappa = 0$
      $\lambda_l^\kappa = \langle A_l w_l^\kappa, w_l^\kappa \rangle_2 / \langle T_l w_l^\kappa, w_l^\kappa \rangle_2$
    end for
  end if
end for
Output W_1 matrix as near-kernel vectors

L：最大レベル数
A_l：レベル l における問題行列 A
P_{l+1}^l：レベル $l+1$ から l へ補間を行う Prolongation 行列

出している．

より広い補間演算子生成という観点から考えると，M. W. Gee らにより提案された Basis function shifting algorithm に基づいた補間演算子生成手法がある [63]．異方性のある問題では，補間演算子の非ゼロ要素が増大する傾向にあり，それにより粗いレベルの行列の行列サイズが十分に粗くならない場合がある．そこで，この手法では，補間演算子生成の際に非ゼロ要素のパターンを制御することで，粗いレベルの非ゼロ要素数を抑える．これにより，異方性のある問題においても，粗いレベルの行列の行列サイズを十分に粗くしつつ，高い収束性を実現したことが報告されている．

また，様々な問題への拡張として，非線形問題向けに SA-AMG 法を拡張し，有用性の検証を行った研究もある [64]．この研究において，ニアカーネルベクトルの設定に αSA 法を用いることで，収束性や実行時間において有用であることが報告されており，本研究において提案した手法を用いることで，非線形問題に対してさらに高い収束性を実現できることが期待できる．

第 6 章

Hybrid 並列

5 章における実験では，最大 64 ノードを用い，最大問題サイズ 648,000（$15^3 \times 3 \times 64$）において計測を行った．しかし近年では，大規模問題を解くことが求められている．5 章にて用いた OFP スーパーコンピュータシステムでは，最大 8208 ノード搭載されており，メモリ容量（MCDRAM:16GB，DDR4:96GB）を加味すると最大 420 億程度（$120 \times 120 \times 120 \times 3 \times 8208$）の問題を解くことが可能である．このような大規模問題を解く際には，このようにより高並列な環境下において計算することが必要となる．しかし並列度が増加するにつれ，計算時間よりも通信時間のコストが多くを占めるようになる．そのため，通信コストの削減は大規模並列環境下での実行を考えるうえで，大きな課題となる．そこで本章では，高並列環境下における通信コスト削減ため，メニーコアクラスタで有用とされる OpenMP/MPI Hybrid 並列を SA-AMG 法前処理付 CG 法に適用し，得られた結果について述べる．

6.1 Hybrid 並列の概要

Hybrid 並列とは，MPI を代表とするプロセス並列と，OpenMP を代表とするスレッド並列を併用した並列プログラミングモデルである．

まず，プロセス並列について説明する．プロセス並列とは，複数のプロセスを起動し，並列化を行う手法である．プロセス間はメモリが非共有であり，データを共有したい場合，通信が必要となる．プロセス並列の実装には，MPI と呼ばれる，プロセッサ間の通信を行うための標準化された規格が広く用いられている．並列計算を MPI のみで行うプログラミングモデルは，一般的に Flat MPI と呼ばれている．Flat MPI は実装の容易さからよく用いられている並列化手法であり，Flat MPI と Hybrid 並列の優劣についてはしばしば議論されてきた．本研究においても，Hybrid 並列と Flat MPI との比較を行い，Hybrid 並列を適用することによる実行時間や並列化効率への影響の分析を行っている．

次に，スレッド並列について説明する．スレッド並列とは，共有メモリ型並列計算機上において，複数のスレッドを起動し並列化を行う手法である．スレッド間はメモリが共有されているため，通信は不要である．スレッド並列を行う際には，OpenMP と呼ばれる標準化された規格が広く用いられている．

Hybrid 並列は，上記で説明したプロセス並列とスレッド並列を併用した並列プログラミ

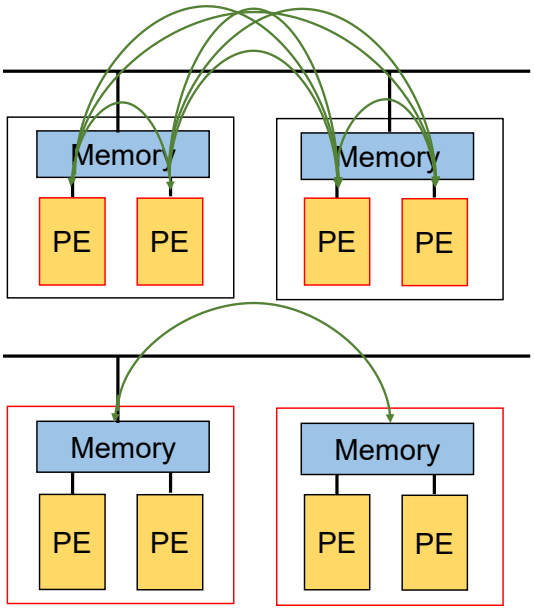


図 6.1: Hybrid 並列の概要

ングモデルである．近年のスーパーコンピュータでは，メニーコアプロセッサに対しネットワークを介して複数個数結合した，メニーコアクラスタと呼ばれる構成が主要形態のひとつとなっている．Flat MPI の場合，コア数分の通信が発生してしまう．そのため，高並列時において通信が実行時間のボトルネックとなってしまうといった問題があった．そこで，メニーコアクラスタにおいて，ノード間の通信を MPI で行い，ノード内では OpenMP で並列計算を行うように Hybrid 並列を適用する．これにより，通信をノード間のみに抑えることができるため，通信オーバーヘッドを抑えることができると期待される．図 6.1 に，上記で述べた Flat MPI と Hybrid 並列の通信に関する例を示す．図 6.1 の例では，1 ノード内に 1CPU 搭載され，その中に 2 コアあるメニーコアクラスタを想定している．また，図 6.1 内の上図は Flat MPI，下図は Hybrid 並列の場合を示している．この図のように，Flat MPI ではコア間においてもプロセス間のデータ共有の際には通信が必要となり，Hybrid 並列と比べ通信回数が多くなる．一方，Hybrid 並列では，通信がノード間のみでしか発生せず，Flat MPI と比べ，通信の回数を抑えることができる．このように，Hybrid 並列を行うことで，高並列環境下で問題となる通信を削減でき，並列化効率の向上が期待できる．

6.2 マルチカラーオーダリング

Hybrid 並列を実際に適用する際には，スレッド並列化部分において，データの依存性に気を付けなければならない．そこで本研究では，マルチカラーオーダリングを行い，データ依存性の問題への対処を行った．マルチカラーオーダリングは，依存性を持たない要素群に対し同じ色に色付けを行い，その色分けに従って要素番号を並び替える手法である．これにより，色内で依存性を持つことなく，同時に独立に並列処理を行うことができる．マルチカラー

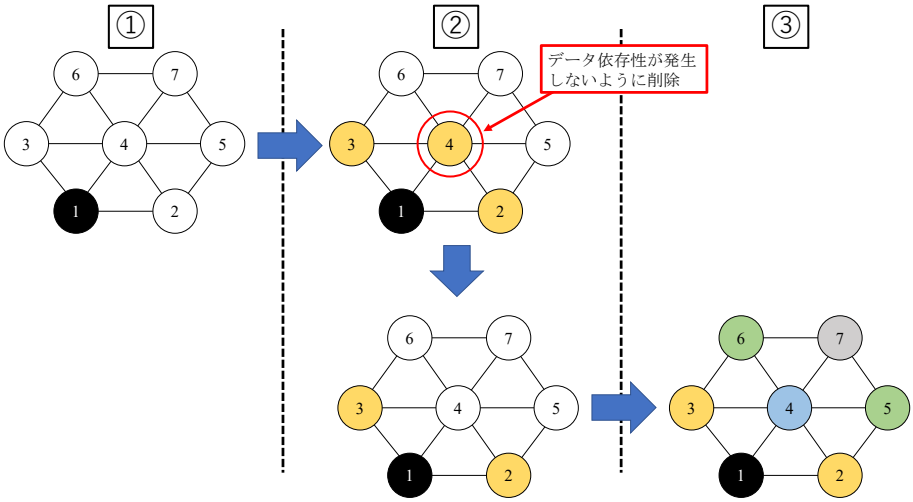


図 6.2: 修正 Cuthill-Mckee 法の概要

オーダリングに関しては多くの手法が提案されている．本研究では，その中でも広く知られている Cuthill-Mckee 法 [65] をもとに，並列計算向けにさらに修正された Cuthill-Mckee 法（以下，修正 CM 法と呼ぶ）を用いた．修正 CM 法のアルゴリズムを以下に，修正 CM 法のアルゴリズムを示した図を Fugire 6.2 に示す．

1. 各要素に隣接する要素数を「次数」とし，最小次数の要素をレベル 1 の要素とする

2. レベル $k-1$ の要素に隣接する要素をレベル k とする．同じレベルに属する要素はデータ依存性が発生しないように，隣接している要素同士が同じレベルに入る場合は一方を除外する．

3. すべての点要素にレベル付けがされるまで，k を 1 つずつ増やして 2. を繰り返す．すべての要素がレベル付けされたら，レベルの番号に再番号付けを行う．

以上のようにマルチカラーオーダリングを行うことで，同じ色内では未知数間の依存関係が発生しないように，色分けを行うことができる．

現状では，本手法をそのまま適用すると，未知数個数に応じて色数が多くなる．マルチカラーオーダリングの情報を用いて並列計算を行う場合，色ごとに並列計算を行うため，なるべく色数は少ないほうが望ましい．そこで本研究ではさらに，Cyclic マルチカラーリングの適用も併せて行った．Cyclic マルチカラーリングの詳細を図 6.3 に示す．図 6.3 は，5 色に色数を抑えるように Cyclic マルチカラーリングを適用した際の，カラーリングの状態を示した図となっている．図 6.3 のように，5 色と制限した場合，6 色目以降は再び 1 色目から色付けを行うようにカラーリングを行っている．このようにカラーリングを行うことで，不必要に色数が増加することを抑えることができる．

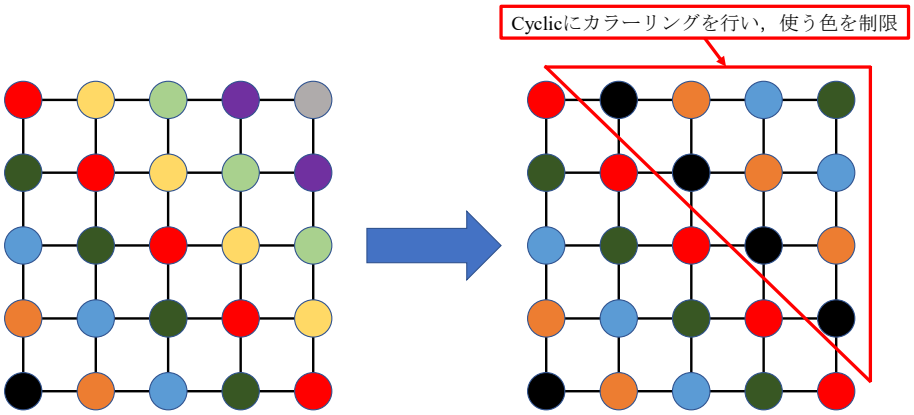


図 6.3: Cyclic マルチカラーリングの概要

6.3 数値実験と評価

6.3.1 実験環境と問題設定

本研究では，東京大学情報基盤センターと筑波大学計算科学研究センターが共同運営する，最先端共同 HPC 基盤施設（JCAHPC : Joint Center for Advanced High Performance Computing）による，Oakforest-PACS（以下，OFP とする）[21]，そして九州大学情報基盤研究開発センターによる ITO [22] の，2 つのスーパーコンピュータシステムを使用し数値実験を行い，Hybrid 並列適用による効果の分析を行った [66,67]．各計算機の構成を表 6.1 に示す．表 6.1 のように，OFP は 1 ノード内に 1CPU（OFP : Intel®Xeon Phi^{TM}7250（68 コア/CPU））, ITO は 2CPU（Intel®Xeon®Gold 6154（18 コア/CPU）×2）搭載されている構成となっている．また本実験では，1.3 節にて示した 3 次元弾性体の問題を使用した．用いた問題サイズについては，次章にて説明する．5.4.2 節と同様に，問題行列の各プロセスへの分割は，各軸方向へ問題を均等に分割し，それにより作成された小行列を各プロセスへ分配する単純な方法で分割を行った．数値実験において使用した並列度に関しても，次節にて説明を行う．また本実験では 5.4.2 節と同様，求解部では CG 法を使用し，前処理として SA-AMG 法を適用している．また，反復の終了条件は相対残差が 1.0×10^{-7} とし，反復回数が 500 回となったときに，収束しなかったとした．

コンパイラは OFP 上では Intel®コンパイラ，ITO では富士通社製コンパイラを用いている．コンパイラオプションは，OFP 上では高速化のための最適化オプションである ”-O3 ” と “ -xMIC-AVX512 ”，OpenMP を用いるためのオプションである ”-qopenmp ” を使用した．また，ITO では高速化オプションである ”-Kfast ”，OpenMP を用いるためのオプションである ”-Kopenmp ” を使用した．さらに，OFP においては，実行時において最適化のために，環境変数に対し “ I_MPI_PIN_DOMAIN=auto ”，“ I_MPI_PIN_PROCESSOR_EXCLUDE_LIST=0,1,68,69,136,137,204,205 ” のような設定を施し，数値実験を行った．さらに，ITO に関しては NUMA 構成となっているため，ITO における結果はすべてファーストタッチを施したものを掲載している．

表 6.1: 本実験で使用したスーパーコンピュータシステムの構成）

		Oakforest-PACS	ITO
Node	# of CPUs	1	2
	Memory	16[GB](MCDRAM) (+96[GB](DDR4))	192[GB]
CPU	Processor name	Intel®Xeon Phi™ 7250 (KNL)	Intel®Xeon®Gold 6154 (Skylake-SP)
	# of cores	68[cores/CPU]	18[cores/CPU]
	Frequency	1.40[GHz]	3.0[GHz]

6.3.2 数値実験の内容

本実験では，2 つの数値実験を行った．行った実験の概要を以下に示す．

- 実験 1：問題サイズを 60^3 に固定
- 実験 2：ウィークスケーリング（1 ノードあたりの問題サイズ 15^3）

まず，実験 1 について説明する．実験 1 では，問題サイズを 60^3 に設定し，小規模・低並列な環境における実行時間の計測を行った．起動ノード数，プロセス数およびスレッド数に関しては，各環境においてそれぞれ搭載コア数が異なるため，起動コア数が等しくなるようにそれぞれ設定を行っている．詳細を表 6.2，および表 6.3 にそれぞれ示す．表 6.2，表 6.3 のように，OFP では 1 ノード，ITO では 2 ノード使用し，総使用コア数が同じとなるように（本実験では 64 コア），起動プロセス数とスレッド数の構成を行った．

次に，実験 2 について説明する．実験 2 では，1 ノードあたりの問題サイズを 15^3 と固定したウィークスケーリングによる実験を行った．本実験では使用ノード数を OFP では 1・8・64 に設定し，ITO に関しては，使用ノード制限のため，2・16・64 の計測を行った．

また，本研究では 6.2 節で説明した通り，Cyclic マルチカラーリングにより，色数の制限を行っている．この時の色数に関して，レベル 1 においては，本実験では規則メッシュを用いており，最小色数が 8 と予測可能である．そのため，レベル 1 では 8 色に固定し，Cyclic マルチカラーリングを適用した．2 レベル目以降に関しては，最小色数が予測不可能なため，Greedy な方法を用いて Cyclic にカラーリングを行った．また，本実験ではカラーリングに要した時間に関しては，特に言及を行わない．さらに，本実験では Flat MPI に対してはマルチカラーリングを行わず，対称 Gauss-Seidel 法をそのまま適用した．

さらに，本研究の数値実験は，すべて「求解部の結果のみ」を載せている．構築部での処理は，各要素で依存関係が発生しているため，スレッド並列化は困難である．これに関しては，様々な研究がなされており，構築部の Hybrid 並列化は今後の課題とする．

6.3.3 実験結果

本節では，数値実験による結果を示す．本実験では，6.3.2 節でも述べたように，2 つの実験を行った．

表 6.2: ノード，プロセス，スレッド数の構成（OFP）

表記	詳細
フラット MPI	1 ノード使用，1 ノード内 64 プロセス起動
1n1p64t	1 ノード使用，1 ノード内 1 プロセス起動し， さらに 1 プロセスあたり 64 スレッド起動
1n2p32t	〃，1 ノード内 2 プロセス起動し， さらに 1 プロセスあたり 32 スレッド起動
1n6p16t	〃 4 プロセス起動し，〃 16 スレッド起動
1n8p8t	〃 8 プロセス起動し，〃 8 スレッド起動

表 6.3: ノード，プロセス，スレッド数の構成（ITO）

表記	詳細
フラット MPI	2 ノード使用，1 ノード内 32 プロセス起動
2n2p32t	2 ノード使用，1 ノード内 1 プロセス起動し， さらに 1 プロセスあたり 32 スレッド起動
2n4p16t	〃，1 ノード内 2 プロセス起動し， さらに 1 プロセスあたり 16 スレッド起動
2n8p8t	〃 4 プロセス起動し，〃 8 スレッド起動

まず，実験 1 について，OFP 上における結果を表 6.4 と図 6.4，および図 6.5 に示す．まず，表 6.4 についての説明を行う．表 6.4 では，収束までに要した実行時間，反復回数を示している．表 6.4 より，Hybrid 並列において 1n4p16t や 1n8p8t が良い結果を示していることがわかる．これより，OFP 上において Hybrid 並列を行う際には，スレッド数を 16 程度に抑えたほうがよいことがわかる．また，Flat MPI と 1n4p16t の実行時間がほぼ同等であることがわかる．これは，本実験では低並列な環境であり，通信の影響がまだ小さいためであると考えられる．また，表 6.4 から構成の変更により反復回数が変化していることがわかる．これはカラーリングによる影響ではなく，前処理として用いている SA-AMG 法の構築部によるものである．SA-AMG 法の構築部では，問題行列から生成されたグラフ構造を用いてアグリゲートを生成し，次のレベルの問題行列を生成している．このアグリゲート生成の際，4.3 節でも述べたように，一旦各プロセスの領域間の接続は無視して，アグリゲートの生成を行っている．そのため，プロセス並列により，階層行列が変化し，結果として反復回数に影響を及ぼすことがある．実際，1・2・4・8 プロセスそれぞれにおいて，スレッドを起動せず実行した際の反復回数と，1n1p64t・1n2p32t・1n4p16t・1n8p8t の反復回数が一致することを確認した（この実行結果については，本論文には記載しない）．次に，図 6.4 についての説明を行う．図 6.4 は，SA-AMG 前処理付 CG 法の実行時間に占める V-cycle の実行時間を，各レベルで示した図となっている．図 6.4 の横軸はノード，プロセス，スレッド数を示し，縦軸は実行時間を示している．また，グラフの要素である Lev.1 等は各レベル，Lev.1-2 は階層移動に要した実行時間を示している．さらに，その他は前処理である V-cycle を除いた，CG 法の計算・通信時間の総和を示している．図 6.4 より，V-cycle の時間が全体の実行時間の約 9 割を占め，特にレベル 1 に占める割合が大きいことがわかった．これ

表 6.4: 収束までの実行時間，反復回数の結果（OFP）

	Flat MPI	1n1p64t	1n2p32t	1n4p16t	1n8p8t
実行時間 [秒]	2.70	2.90	3.03	2.62	2.70
反復回数	26	30	30	26	26

は，レベル 1 の問題サイズが大きく，スムーザにおける計算や通信によるコストが最も大きくなったからであると考えられる．ここで，V-cycle についてさらに分析を行う．図 6.5 に，V-cycle の実行時間における内訳のグラフを示す．図 6.5 の縦軸および横軸は図 6.4 と同じである．グラフの青色の要素はスムーザにおける通信を除いた計算時間，オレンジ色の要素は通信のみを示す．また，灰色の要素は V-cycle 内において，スムーザによる通信を除いた通信時間（主に階層移動の際に発生），黄色はそれ以外の，階層移動や最下層の LU 分解により発生した計算時間の総和を示す．図 6.5 より，本実験の環境ではスムーザによる計算時間が V-cycle のほとんどを占めており，全体の実行時間から見ても，多くを占めていることがわかった．例えば，フラット MPI の場合においては，スムーザの計算時間は V-cycle 内で 81%，全体と比較しても 71%を占めることがわかった．また Hybrid 並列においても，スムーザの計算時間はどの構成においても，V-cycle 内で 80%以上，全体でも 70%以上を占めていることがわかった．また通信時間に占める割合は，本実験では少ないこともわかった．これは，この実験は最大で 1 ノード 64 プロセスのみしか扱わない，低並列な環境であるためと考えられる．

次に，ITO における結果を表 6.5 と図 6.6 および図 6.7 に示す．それぞれの表やグラフの内容は OFP のものと同様である．まず，表 6.5 について述べる．表 6.5 の（fit）という項目は，プロセス内のスレッドが，CPU をまたがないようにリソース確保を調整したときの結果となっている．ITO は 1 ノード内に 2CPU 搭載されている構成となっている．そのため，通常通りリソース確保を行うと，プロセス内のスレッドが CPU をまたいでしまい，性能に影響を及ぼすことが考えられる．そこで，（fit）では 1 つの CPU 内でプロセスとスレッドの配置が完結するように，リソース確保を行ったときの結果を示している．具体例を図 6.8 に示す．図 6.8 の例では，リソース確保の時点では，vnode（仮想ノードの意味．ITO では基本的に仮想ノードでリソースの確保を行う．基本的に起動プロセス数と等価）を 2 と設定し，core（確保する CPU コアの設定．基本的に起動スレッド数と等価）を 8 と設定する．その後，実行時に環境変数 OMP_NUM_THREADS により起動スレッド数を調整することで，1 つの CPU 内にプロセスとスレッドの配置を完結させつつ，起動スレッド数を調整することが可能となる．これにより，実際に表 6.5 から，通常通りリソース確保をする方法よりも，良い性能を得られることがわかる．また，Flat MPI と（fit）における 2n4p16t，2n8p8t の実行時間がほぼ同等であることがわかる．これより，OFP と同様，Hybrid 並列を行う際には，起動スレッド数を，1CPU 内に収まる 16 程度に抑えたほうがよいことがわかる．次に，図 6.6 と図 6.7 に着目する．これ以降の ITO の結果は，すべて（fit）の結果を掲載する．図 6.6 と図 6.7 より，基本的な傾向は OFP と同様であることがわかる．また OFP 同様，本実験では並列度が小さいため，通信による実行時間への影響が小さいこともわかる．

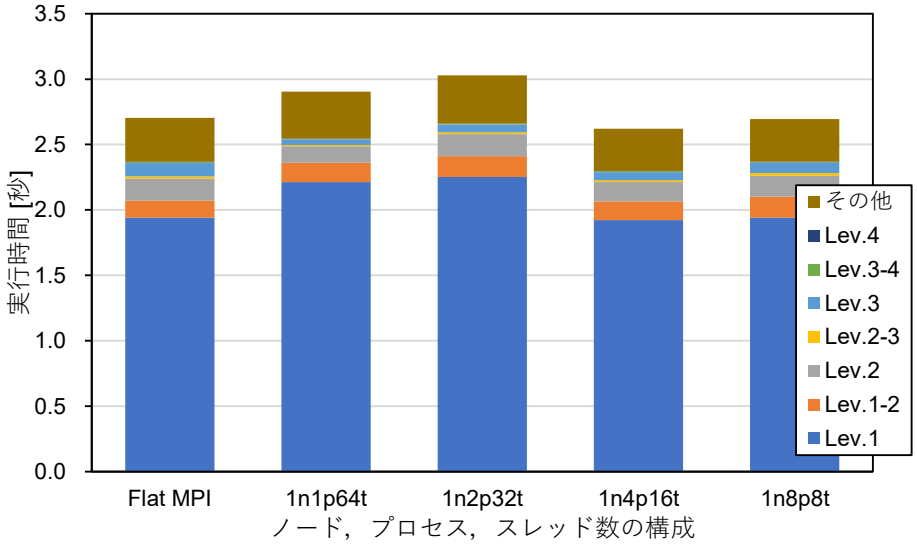


図 6.4: 実行時間に占める V-cycle の割合（OFP）

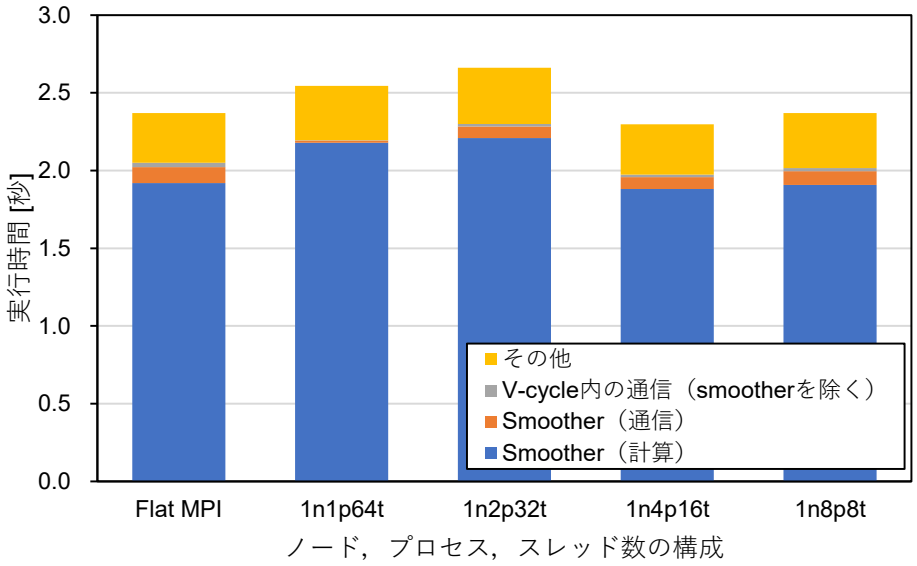


図 6.5: V-cycle 内の実行時間内訳（OFP）

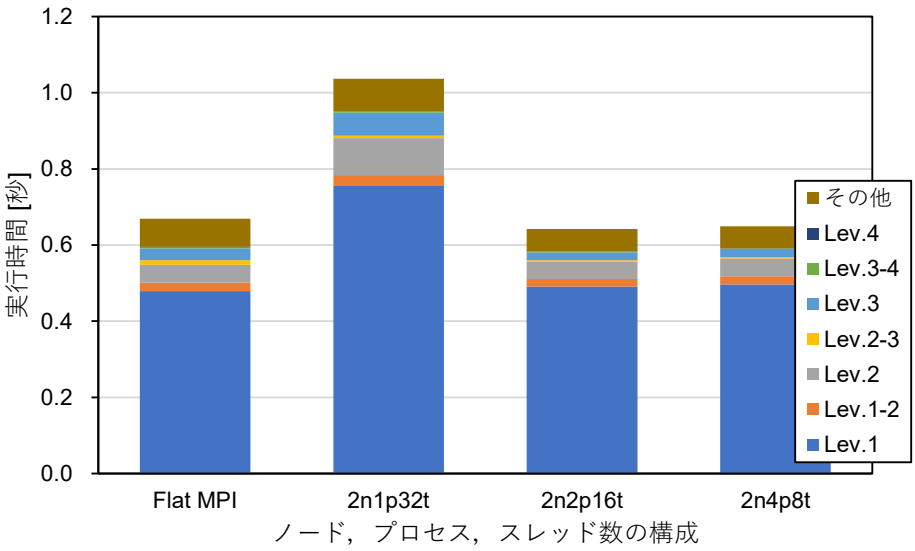


図 6.6: 実行時間に占める V-cycle の割合（ITO）

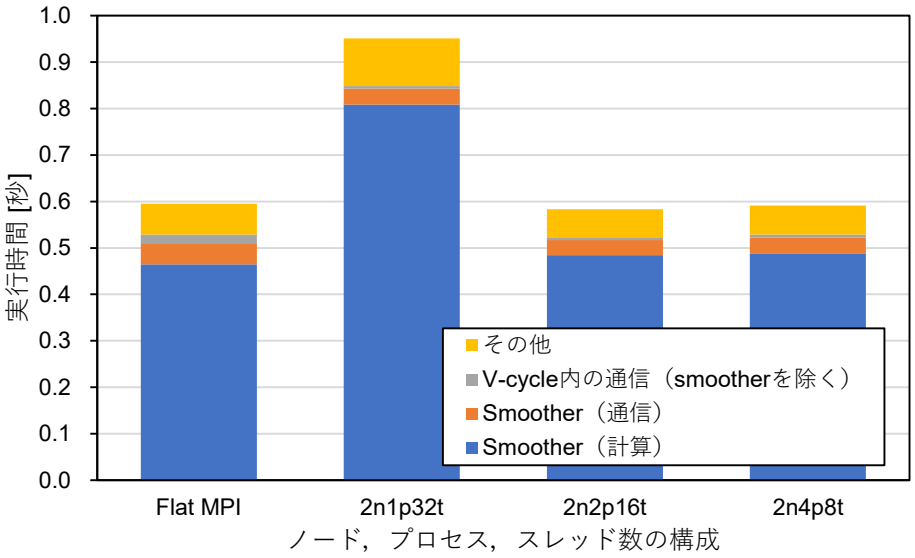


図 6.7: V-cycle 内の実行時間内訳（ITO）

表 6.5: 収束までの実行時間，反復回数の結果（ITO）

	Flat MPI	2n2p32t	2n4p16t	2n8p8t
実行時間 [秒]	0.67	0.86	0.72	0.71
反復回数	26	30	26	26
	—	2n2p32t(fit)	2n4p16t(fit)	2n8p8t(fit)
実行時間 [秒]	—	0.88	0.65	0.65
反復回数	—	30	26	26

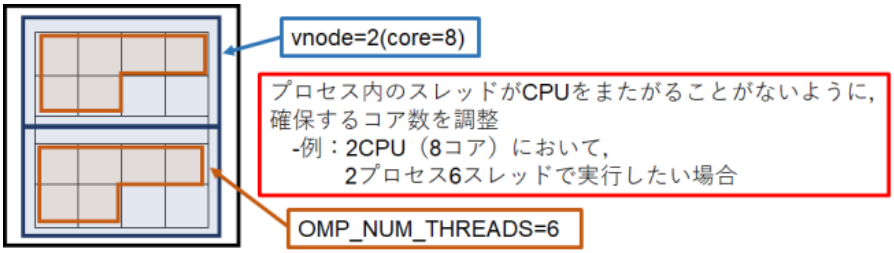


図 6.8: (fit) の詳細

次に，実験 2 の結果を示す．まず，OFP における結果を表 6.6 と表 6.7 に示す．表 6.6 と表 6.7 はそれぞれ OFP 上におけるウィークスケーリングによる実行時間と反復回数の結果を示している．また，表 6.6 の括弧内は 1 反復あたりの実行時間を示している．表 6.6 より，本実験では 1 ノードの傾向がそのまま 64 ノードでも見られることがわかる．また，64 ノードの時の ”Thread:64 ” の実行時間がその他と比べ遅くなっているが，これは表 6.7 より，主に反復回数の影響であることもわかる．これらより，本実験での並列度においても，通信の影響があまり表れず，1 ノードの傾向がそのまま 64 ノードでも表れていることがわかる．実際に，64 ノードにおいて表 6.7 のように分析した結果を図 6.9 に示す．図 6.9 より，通信時間が 1 ノード時よりも増加しているものの，全体の実行時間と比べてまだ小さいことがわかる．

次に，ITO における結果を表 6.8 と表 6.9 に示す．ITO の結果においては，ノード資源利用制限のため，OFP と比べ，最大並列時における総使用コア数が少なくなっている．表 6.8 と表 6.9 より，OFP と同様，1 ノード時と同様の傾向が，64 ノード時にみられることがわかる．以上の結果より，本実験における並列度においては，通信の影響がまだ少なく，今後の課題として，より高並列な環境で計測することがあげられる．

表 6.6: OFP：ウィークスケーリングにおける実行時間 [秒]（括弧内は 1 反復あたりの時間）

	Flat MPI	Thread:64	Thread:16
Node:1	2.70(0.104)	2.90(0.097)	2.62(0.101)
Node:8	4.47(0.128)	3.98(0.117)	4.23(0.124)
Node:64	6.13(0.153)	6.85(0.152)	6.06(0.148)

表 6.7: OFP：ウィークスケーリングにおける反復回数

	Flat MPI	Thread:64	Thread:16
Node:1	26	30	26
Node:8	35	34	34
Node:64	40	45	41

表 6.8: ITO：ウィークスケーリングにおける実行時間 [秒]（括弧内は 1 反復あたりの時間）

	Flat MPI	Thread:32	Thread:16
Node:1	0.67(0.026)	1.04(0.035)	0.64(0.025)
Node:8	1.18(0.034)	1.75(0.047)	1.17(0.033)
Node:64	1.78(0.051)	2.21(0.071)	1.38(0.045)

6.4 関連研究

6.2 節において，Hybrid 並列を適用する際にデータ依存性の対処のための手法として，マルチカラーオーダリングを紹介した．データ依存性のある計算においての並列化計算手法の関連研究として，Asynchronous iterative method [68] がある．この手法は，データ依存性のある計算を，依存性のあるまま計算する手法である．このままでは，収束性に大きな影響を与えることがわかっているが，Multigrid 法のスムーザとして用いる際には，あまり影響がないともいわれている [69]．また，計算領域を通信と計算部分の領域にわけ，オーバーラップすることでデータ依存性を回避する手法 [70] や，計算する領域を少しずつ広げることで，データ依存性の問題を回避しつつ並列計算する手法 [71]，さらに領域境界を階層的に計算し問題を回避する HID（Hierarchical Interface Decomposition）[72] [73, 74] と呼ばれる手法も提案されている．

一方，Multigrid 法の Hybrid 並列化による性能を，性能モデルを作成して Hybrid 並列の恩恵を得られるかを判断する研究もある [75, 76]．また，上記以外にも様々な Hybrid 並列化に伴う依存関係に関する手法 [77–81] や，Multigrid 法向けの GPU を使用した Hybrid 並列手法に関する研究 [82–92] がなされている．

表 6.9: ITO：ウィークスケーリングにおける反復回数

	Flat MPI	Thread:32	Thread:16
Node:1	26	30	26
Node:8	35	37	35
Node:64	35	31	31

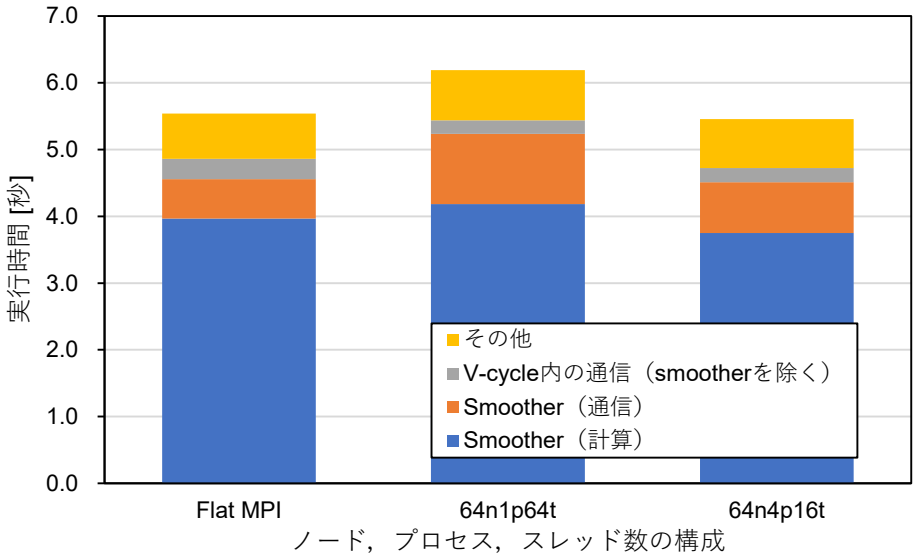


図 6.9: 64 ノード時の V-cycle 内の実行時間内訳（OFP）

第 7 章

結論

7.1 まとめ

本研究では SA-AMG 法の収束性安定化のため，2 つのニアカーネルベクトル抽出手法を提案した．1 つめのニアカーネルベクトル抽出手法として，粗いレベルでニアカーネルベクトルを複数本抽出し，抽出時において適切な設定本数を予測する手法を提案し，従来手法と比較を行い有用性を検証した．2 つめのニアカーネルベクトル抽出手法として，粗いレベルで固有値解析を実施し補間を行うことでニアカーネルベクトルを抽出する手法の提案を行った．本手法の数値実験では，従来手法や 1 つめの抽出手法と反復回数や抽出時間を含めた実行時間で比較を行い，有用性の検証を行った．

提案手法 1 では，ニアカーネルベクトルの設定本数の変化による収束性や実行時間の変化を示した．この実験より，ニアカーネルベクトルを複数本設定することで，反復回数が改善することが分かった．また実行時間に関しても，ニアカーネルベクトルを複数本設定することで，反復回数の改善の影響で求解部の実行時間が改善することも分かった．しかし，階層行列を作成する構築部においては，ニアカーネルベクトルの本数を増やすことで，階層行列生成の計算コストが増加してしまい，結果として構築部と求解部の実行時間の合計が悪化してしまうこともみられた．これらより，ニアカーネルベクトルを複数本設定する際には，適切な本数設定することが必要であることがわかった．適切な本数の設定には，この手法ではニアカーネルベクトルの本数の組み合わせを全パターン網羅する必要があるが，この検証には膨大な時間がかかってしまうという問題があった．

そこで提案手法 2 では，粗いレベルでニアカーネルベクトルを複数本抽出し，抽出時において適切な設定本数を予測する手法の提案を行った．この手法では，パラメータとして閾値 ϵ を設定する必要がある．この実験より，この値を適切に高く設定することで，構築部と求解部の合計時間を前実験の最良実行時間とほぼ同等に，またこの値を低く設定することで，求解部のみであるが実行時間をほぼ同等かさらに良い結果を示した．また検証時間についても，従来膨大な時間を必要としていたが，本手法によりコストを大きく抑えることに成功した．以上より，本論文の提案手法では，与えられた問題行列が解法適用時に毎回異なる場合には，閾値 ϵ を高く設定する．対して，反復回数を多く必要とする悪条件の問題や，元の問題設定は変えずに右辺ベクトルが毎回異なる場合には低く設定するように，目的に応じ

て適切なパラメータを設定することで，本手法が高い効果を発揮できると考えられる．

上記で述べたニアカーネルベクトル抽出手法では，V-cycle を用いてニアカーネルベクトルを 1 本づつ抽出するため，抽出本数により抽出時間が増加してしまう．これにより，ニアカーネルベクトル抽出コストが，実際に解法を適用し問題を解くコストと比べ，大きくなってしまう問題がある．そこで提案手法 3 として，抽出のさらなる効率化のため，粗いレベルで固有値解析を実施し補間を行うことでニアカーネルベクトルを抽出する手法の提案を行った．そして，抽出されたニアカーネルベクトルを用いることによる反復回数や実行時間，および抽出時間への影響の分析を行った．本研究の手法を用いることで，従来手法と比べ抽出時間を抑えつつ，従来手法と同等の高い収束性を実現できることがわかった．本研究の手法では，抽出時に固有値解析を実施する必要がある．本研究では Lapack ライブラリの固有値計算関数（dsyev），並列時に Scalapack ライブラリの固有値計算関数（pdsyev）と正定値対称疎行列向け固有値計算法である Lanczos 法を用いて実験を行ったが，各手法により抽出時間や抽出されたニアカーネルベクトルの性質に影響があることもわかった．そのため，他の固有値計算法を用いることによる影響の分析も必要であると考えられる．

本研究ではさらに，高並列環境下における実行を考え，粗いレベルでプロセス領域の集約を行う CGA についての実験を行った．本実験では，最大 1440 ノード使用したウィークスケーリングにおいて，CGA にて用いられている閾値を変化させたときの実行時間への影響，および関連手法である共有アグリゲート生成手法と比較を行い，有用性の検証を行った．数値実験により，適切な閾値を設定することで，CGA は高並列環境下においても有用であることがわかった．

上記に加え，本研究ではメニィコアクラスタ形式の計算機において適しているとされる Hybrid 並列を，SA-AMG 法に適用した際の結果を分析した．本研究では，OpenMP と MPI による OpenMP/MPI Hybrid 並列を適用し，実験を行った．実験では，Oakforest-PACS（JCAHPC），ITO（九州大学）スーパーコンピュータシステム上で実行時間や Flat MPI との比較による並列化効率への影響の分析を行った．数値実験では，問題サイズ 60^3 に設定した小規模低並列環境下においての実験と，ウィークスケーリング（問題サイズを 1 プロセスあたり 15^3 に設定）においての，2 つの実験を行った．1 つめの実験より，Oakforest-PACS と ITO において，Hybrid 並列時にスレッド数を適切に設定することが必要であることがわかった．また，本実験の環境では低並列なため通信の影響があまり表れず，Hybrid 並列と Flat MPI で実行時間がほぼ同等であることがわかった．さらに，2 つめの実験より，本実験の環境ではまだ通信の影響が少なく，1 ノード時と同様の傾向が並列度を増加させてもみられることがわかった．

以上をまとめる．まず，5 章で示した本研究において提案したニアカーネルベクトル抽出手法のまとめを表 7.1 に示す．表 7.1 のように，本研究において提案したニアカーネルベクトル抽出手法を用いることで，従来手法と比べ高い収束性とそれによる実行時間改善効果がみられるがわかった．しかし，パラメタ設定方法など様々な課題があり，これらは今後の課題とする．

また，本研究では高並列環境下に向けた手法として，CGA と Hybrid 並列の 2 つについて実験を行った．以下に本研究で得られた知見や課題をまとめる．

表 7.1: 本研究のニアカーネルベクトル抽出提案手法まとめ（5 章）

	提案手法（5.4 節）	提案手法 2（5.5 節）	提案手法（5.6 節）
利点	・適切な本数抽出で，収束性と実行時間改善	・検証時間削減 ・提案手法 1 と同等の高収束性実現	・抽出時間をさらに改善 ・提案手法 2 と同等の高収束性実現
課題	・適切な抽出本数の検証に膨大な時間	・閾値設定が必要 ・抽出時間が問題を解く時間と比べ大	・用いる固有値問題やパラメタ設定によるさらなる分析が必要

CGA（Coarse Grid Aggregation）: 4.3 節

- 適切な閾値を設定することで，高並列環境下においても有用
- どの閾値設定が適切かは環境依存（閾値設定による実行時間予測モデルの作成等，適切な閾値設定方法の確立と分析が必要）

Hybrid 並列：6 章

- 環境に応じて起動スレッド数を適切に設定することが必要
- 本研究の環境設定では，通信の影響があまり表れず，Hybrid 並列と Flat MPI で実行時間がほぼ同等（より高並列な環境での分析が必要）

7.2 今後の課題

今後の課題について，まずニアカーネルベクトル抽出手法に着目して述べる．まず提案手法 2 において，ニアカーネルベクトルの本数予測で用いている閾値 ϵ の設定方法の確立，または新たな予測手法の提案が必要であると考えられる．本研究では閾値 ϵ は利用者側が決定することとなっているが，この値により提案手法を用いることによる収束性や実行時間が大きく影響すると考えられる．そのため，予測手法に用いている閾値 ϵ の設定方法の確立，あるいは閾値 ϵ を必要としない新たな予測手法の確立が必要となる．

次に提案手法 2 において，本研究の手法に用いる固有値計算法に関する分析を行うことが考えられる．実験により、抽出時間や抽出されたニアカーネルベクトルの性質が、用いた固有値計算法に影響されることがわかった．例として，今回は並列時において Lanczos 法を用いたが，派生解法である Thick-restart Lanczos 法 [93] などの解法を用いることが考えられる．その上で，他の問題に対して適用し，有用性の検証を行うことが考えられる．

そのうえで，新たなニアカーネルベクトル抽出手法の提案を行うことが必要である．今回は問題行列から，V-cycle や固有値解法を用いて抽出を行った．これらとは別のアプローチとして，Aggregate の生成に着目しながら抽出を行うような方向性も考えられる．本研究では単純な方法で Aggregate の生成を行ったが，Aggregate の生成により，ニアカーネルベクトル抽出にも影響すると考えられる．これより，ニアカーネルベクトル抽出に適した

Aggregate の生成方法が存在すると考えられ，これらの関係性の分析，および新たなニアカーネルベクトル抽出手法が提案できると考えられる．

Hybrid 並列においては，まずは京におけるスムーザの計算部分に関する分析および最適化を行うことがあげられる．実験より，京においてキャッシュミスにより計算時間が低下する傾向が顕著に見られた．そのため，京における Hybrid 並列時には，ELL 形式の実装などの，キャッシュ利用に関する最適化が必要であると考えられる．これにより，本研究で用いていない他環境においても，Hybrid 並列における通信効率化の長所が生かせると期待できる，さらにその上で，より高並列な環境において，Hybrid 並列による効果を分析することが必要である．本研究ではウィークスケーリングのみの実験であったが，さらに問題サイズを大きくする，ストロングスケーリングにおいて計測するなどを行い，高並列環境下における通信の影響を調査する必要がある．

謝辞

本論文は筆者が東京大学大学院情報理工学系研究科数理情報学専攻博士後期課程に在籍中の研究成果をまとめたものである．

本論文を執筆するにあたり，多くの方々からのご指導やご助言，ご協力をいただいたことに対して，ここに深く感謝の意を表する．

東京大学情報基盤センターの中島研吾教授には，指導教官として本研究の実施の機会を与えていただき，その遂行にあたって終始，ご指導ご鞭撻をいただいた．また研究内容のみならず，研究の方向性や事務手続き書類の作成に至るまで，数多くのご指導をいただいた．ここに感謝の意を表する．

東京大学情報理工学系研究科の松尾宇泰教授ならびに長尾大道准教授，東京大学新領域科学研究科の奥田洋司教授，ならびに北海道大学情報基盤センターの岩下武史教授には副査としてご助言をいただくとともに，本論文の細部にわたりご指導をいただいた．ここに感謝の意を表する．

東京大学情報理工学系研究科の田中健一郎准教授ならびに佐藤峻助教，ならびに法政大学情報科学部の相島健助准教授には，研究室内輪講において研究に関するさまざまなご意見をいただいた．ここに感謝の意を表する．

工学院大学情報学部の田中輝雄教授ならびに藤井昭宏准教授には，修士課程より本研究を遂行するにあたり多くのご指導やご助言をいただいた．ここに感謝の意を表する．

理化学研究所の松葉浩也氏ならびに河合直聡氏には，理化学研究所インターンシップの受け入れ，および研究内容に関するご指導ご助言をいただいた．ここに感謝の意を表する．

最後に，支えてくださった家族や友人に深く感謝する．

参考文献

[1] N. Nomura, A. Fujii, T. Tanaka, O. Marques, and K. Nakajima. Algebraic multigrid solver using coarse grid aggregation with independent aggregation. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1104–1112. IEEE, 2018.

[2] S. Míka and P. Vaněk. Acceleration of convergence of a two-level algebraic algorithm by aggregation in smoothing process. *Applications of Mathematics*, 37(5):343–356, 1992.

[3] P. Vaněk. Fast multigrid solver. *Applications of Mathematics*, 40(1):1–20, 1995.

[4] F. H. Pereira, S. L. L. Verardi, and S. I. Nabeta. A fast algebraic multigrid preconditioned conjugate gradient solver. *Applied mathematics and computation*, 179(1):344–351, 2006.

[5] T. F. Chan and P. Vaněk. Multilevel algebraic elliptic solvers. In *International Conference on High-Performance Computing and Networking*, pages 999–1014. Springer, 1999.

[6] V. E. Henson and U. M. Yang. Boomeramg: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002.

[7] M. F Adams. Algebraic multigrid methods for constrained linear systems with applications to contact problems in solid mechanics. *Numerical linear algebra with applications*, 11(2-3):141–153, 2004.

[8] S. MacLachlan, T. Manteuffel, and S. McCormick. Adaptive reduction-based amg. *Numerical Linear Algebra with Applications*, 13(8):599–620, 2006.

[9] L. N. Olson, J. B. Schroder, and R. S. Tuminaro. A general interpolation strategy for algebraic multigrid using energy minimization. *SIAM Journal on Scientific Computing*, 33(2):966–991, 2011.

[10] P. D'Ambra and P. S. Vassilevski. Adaptive amg with coarsening based on compatible weighted matching. *Computing and Visualization in Science*, 16(2):59–76, 2013.

[11] A. Franceschini, V. A. P. Magri, G. Mazzucco, N. Spiezia, and C. Janna. A robust adaptive algebraic multigrid linear solver for structural mechanics. *Computer Methods in Applied Mechanics and Engineering*, 352:389–416, 2019.

[12] Victor A. Paludetto M., A. Franceschini, and C. Janna. A novel algebraic multigrid approach based on adaptive smoothing and prolongation for ill-conditioned systems. *SIAM Journal on Scientific Computing*, 41(1):A190–A219, 2019.

[13] R. Blaheta. A multilevel method with correction by aggregation for solving discrete elliptic problems. *Aplikace matematiky*, 31(5):365–378, 1986.

[14] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[15] P. Vaněk, M. Brezina, and J. Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88(3):559–579, 2001.

[16] A. Fujii, A. Nishida, and Y. Oyanagi. Evaluation of parallel aggregate creation orders: Smoothed aggregation algebraic multigrid method. In *High Performance Computational Science and Engineering*, pages 99–122. Springer, 2005.

[17] M. Brezina, R Falgout, S. MacLachlan, T Manteuffel, S McCormick, and J Ruge. Adaptive smoothed aggregation (αSA). *SIAM Journal on Scientific Computing*, 25(6):1896–1920, 2004.

[18] M. Brezina, T. Manteuffel, S. MCormick, J. Ruge, and G. Sanders. Towards adaptive smoothed aggregation (α sa) for nonsymmetric problems. *SIAM Journal on Scientific Computing*, 32(1):14–39, 2010.

[19] N. Nomura, A. Fujii, T. Tanaka, K. Nakajima, and O. Marques. Performance analysis of SA-AMG method by setting extracted near-kernel vectors. In *International Conference on Vector and Parallel Processing*, pages 52–63. Springer, 2016.

[20] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[21] Joint Center for Advanced High Performance Computing (JCAHPC). `https://ofp-www.jcahpc.jp/`, (accessed: Dec. 6, 2019).

[22] 九州大学情報基盤研究開発センター. `https://www.cc.kyushu-u.ac.jp/scp/system/ITO/`, (accessed: Dec. 6, 2019).

[23] 奥田洋司. 並列有限要素解析 [II]. クラスタコンピューティング, 2008.

[24] R. Tamstorf, T. Jones, and S. F. McCormick. Smoothed aggregation multigrid for cloth simulation. *ACM Transactions on Graphics (TOG)*, 34(6):245, 2015.

[25] 杉原正顯，室田一雄．線形計算の数理. 岩波書店, 2009.

[26] L. V. Kantorovich. Functional analysis and applied mathematics. *Uspekhi Matematicheskikh Nauk*, 3(6):89–185, 1948.

[27] W. Greub and W. Rheinboldt. On a generalization of an inequality of lv kantorovich. *Proceedings of the American Mathematical Society*, 10(3):407–415, 1959.

[28] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial*, volume 72. Siam, 2000.

[29] 建部修見，小柳義夫．マルチグリッド前処理付き共役勾配法. 情報処理学会研究報告ハイパフォーマンスコンピューティング *(HPC)*, 1992(66 (1992-HPC-042)):9–16, 1992.

[30] O. Tatebe. The multigrid preconditioned conjugate gradient method. In *6th Copper Mountain Conference on Multigrid Methods, Copper Mountain*. Front Range Scientific Computations, Inc., 1993.

[31] 建部修見. *MGCG METHOD: A ROBUST AND HIGHLY PARALLEL ITERATIVE METHOD MGCG* 法*:* ロバストで高効率な並列解法. PhD thesis, The University of Tokyo, 1996.

[32] K. Stüben and U. Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In *Multigrid methods*, pages 1–176. Springer, 1982.

[33] W. Hackbusch. *Multi-grid methods and applications*, volume 4. Springer Science & Business Media, 2013.

[34] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S McCormick, and J. Ruge. Adaptive algebraic multigrid. *SIAM Journal on Scientific Computing*, 27(4):1261–1286, 2006.

[35] A. Brandt, S. McCoruick, and J. Huge. Algebraic multigrid (amg) for sparse matrix equations. *Sparsity and its Applications*, 257, 1985.

[36] A. Brandt. Algebraic multigrid theory: The symmetric case. *Applied mathematics and computation*, 19(1-4):23–56, 1986.

[37] M. Sala and R. S. Tuminaro. A new petrov–galerkin smoothed aggregation preconditioner for nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 31(1):143–166, 2008.

[38] R. S. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In *SC'00: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*, pages 5–5. IEEE, 2000.

[39] C. V. Deutsch and A. G. Journel. Gslib geostatistical software library and user ' s guide, 1998.

[40] G. Karypis. Metis and parmetis. *Encyclopedia of parallel computing*, pages 1117–1124, 2011.

[41] K. Nakajima. Automatic tuning of parallel multigrid solvers using openmp/mpi hybrid parallel programming models. In *International Conference on High Performance Computing for Computational Science*, pages 435–450. Springer, 2012.

[42] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler. Parallel geometric-algebraic multigrid on unstructured forests of octrees. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 43. IEEE Computer Society Press, 2012.

[43] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos. Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 34. IEEE Computer Society, 2004.

[44] H. D. Sterck, U. M. Yang, and J. J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):1019–1039, 2006.

[45] J. Brannick, Y. Chen, X. Hu, and L. Zikatanov. Parallel unsmoothed aggregation algebraic multigrid algorithms on gpus. In *Numerical Solution of Partial Differential Equations: Theory, Algorithms, and Their Applications*, pages 81–102. Springer, 2013.

[46] J. Cohen and P. Castonguay. Efficient graph matching and coloring on the gpu. In *GPU Technology Conference*, pages 1–10, 2012.

[47] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, and R. Strzodka. Amgx: A library for gpu accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 37(5):S602–S626, 2015.

[48] A. Brandt, J Brannick, K. Kahl, and I. Livshits. Bootstrap amg. *SIAM Journal on Scientific Computing*, 33(2):612–632, 2011.

[49] P. D'Ambra, L. Cutillo, and P. S. Vassilevski. Bootstrap amg for spectral clustering. *Computational and Mathematical Methods*, 1(2):e1020, 2019.

[50] 野村直也，中島研吾，藤井昭宏，田中輝雄．高並列環境下に向けた粗いレベルでニアカーネルベクトルを抽出および追加的に設定する手法の性能評価．研究報告ハイパフォーマンスコンピューティング *(SWoPP2017)*, 2017(31):1–8, 2017.

[51] N. Nomura, A. Fujii, and K. Nakajima. Robust SA-AMG solver by extraction of near-kernel vectors. In *2017 The International Conference for High Performance Computing, Networking, Storage and Analysis (SC17)*. IEEE, 2017.

[52] M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

[53] N. Nomura, A. Fujii, and K. Nakajima. The study of the automatic extraction method of near-kernel vector for high scalable and stable SA-AMG method. *IPSJ Transactions on Computing Systems*, 12(3):1–18, 2019.

[54] 野村直也，中島研吾，藤井昭宏．SA-AMG 法における収束性安定化のための効率的なニアカーネルベクトル抽出手法に向けた研究. 研究報告ハイパフォーマンスコンピューティング *(SWoPP2019)*, 2019(20):1–10, 2019.

[55] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' guide*, volume 9. SIAM, 1999.

[56] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, and A. Petitet. *ScaLAPACK users' guide*, volume 4. SIAM, 1997.

[57] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1998.

[58] T. T. Charles. Numerical solution of linear elasticity problems with intersecting slide surface constraints. *LLNL Technical Report, LLNL-JC-148529*, 2002.

[59] M. Brezina and P. S. Vassilevski. Smoothed aggregation spectral element agglomeration amg: Sa-ρamge. In *International Conference on Large-Scale Scientific Computing*, pages 3–15. Springer, 2011.

[60] J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM review*, 34(4):581–613, 1992.

[61] J. Wu and H. Zheng. Parallel subspace correction methods for nearly singular systems. *Journal of Computational and Applied Mathematics*, 271:180 – 194, 2014.

[62] A. Stathopoulos and K. Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM Journal on Scientific Computing*, 32(1):439–462, 2010.

[63] M. W. Gee, J. J. Hu, and R. S. Tuminaro. A new smoothed aggregation multigrid method for anisotropic problems. *Numerical Linear Algebra with Applications*, 16(1):19–37, 2009.

[64] M. W. Gee and R. S. Tuminaro. Nonlinear algebraic multigrid for constrained solid mechanics problems using trilinos. Technical report, Sandia National Laboratories, 2006.

[65] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172. ACM, 1969.

[66] 野村直也，中島研吾，藤井昭宏．oakforest-pacs 上における SA-AMG 法の hybrid 並列化に関する分析．研究報告ハイパフォーマンスコンピューティング *(SWoPP2018)*, 2018(34):1–8, 2018.

[67] 野村直也，中島研吾，河合直聡，藤井昭宏．大規模クラスタ環境上における SA-AMG 法の hybrid 並列化に関する分析．研究報告ハイパフォーマンスコンピューティング *(HPC167)*, 2018(27):1–9, 2018.

[68] G. M. Baudet. Asynchronous iterative methods for multiprocessors. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1976.

[69] H. Anzt, S. Tomov, M. Gates, J. Dongarra, and V. Heuveline. Block-asynchronous multigrid smoothers for gpu-accelerated systems. *Procedia Computer Science*, 9:7–16, 2012.

[70] T. H. Kaiser and S. B. Baden. Overlapping communication and computation with openmp and mpi. *Scientific Programming*, 9(2, 3):73–81, 2001.

[71] B. Dolwithayakul, C. Chantrapornchai, and N. Chumchob. An efficient asynchronous approach for gauss-seidel iterative solver for fdm/fem equations on multi-core processors. In *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*, pages 357–361. IEEE, 2012.

[72] P. Hénon and Y. Saad. A parallel multistage ILU factorization based on a hierarchical graph decomposition. *SIAM Journal on Scientific Computing*, 28(6):2266–2293, 2006.

[73] 中島研吾. 階層型領域分割による並列多重格子法. 「ハイパフォーマンスコンピューティングと計算科学シンポジウム *HPCS2008*」 論文集, 2008.

[74] K. Nakajima. Parallel iterative solvers for ill-conditioned problems with heterogeneous material properties. In *ICCS*, pages 1635–1645, 2016.

[75] H. Gahvari, W. Gropp, K. E. Jordan, M. Schulz, and U. M. Yang. Modeling the performance of an algebraic multigrid cycle using hybrid MPI/OpenMP. In *2012 41st International Conference on Parallel Processing*, pages 128–137. IEEE, 2012.

[76] H. Gahvari, W. Gropp, K. Jordan, and U. M. Yang. Applying performance modeling to guide hybrid MPI/OpenMP use and communication/computation tradeoff in algebraic multigrid. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2014.

[77] D. Wallin, H. Löf, E. Hagersten, and S. Holmgren. Multigrid and gauss-seidel smoothers revisited: parallelization on chip multiprocessors. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 145–155, 2006.

[78] V. Marjanović, J. Labarta, E. Ayguadé, and M. Valero. Overlapping communication and computation by using a hybrid mpi/smpss approach. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 5–16, 2010.

[79] B. Dolwithayakul, C. Chantrapornchai, and N. Chumchob. Gpu-based total variation image restoration using sliding window gauss-seidel algorithm. In *2011 International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS)*, pages 1–6. IEEE, 2011.

[80] Y. Sato, T. Hino, and K. Ohashi. Parallelization of an unstructured navier–stokes solver using a multi-color ordering method for openmp. *Computers & Fluids*, 88:496–509, 2013.

[81] M. Kawai, A. Ida, and K. Nakajima. Hierarchical parallelization of multi-coloring algorithms for block ic preconditioners. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 138–145. IEEE, 2017.

[82] A. Krechel and K. Stüben. Parallel algebraic multigrid based on subdomain blocking. *Parallel Computing*, 27(8):1009–1031, 2001.

[83] M. Adams, M. Brezina, J. Hu, and R. Tuminaro. Parallel multigrid smoothing: polynomial versus gauss–seidel. *Journal of Computational Physics*, 188(2):593–610, 2003.

[84] U. M. Yang. Parallel algebraic multigrid methods—high performance preconditioners. In *Numerical solution of partial differential equations on parallel computers*, pages 209–236. Springer, 2006.

[85] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. Multigrid smoothers for ultraparallel computing. *SIAM Journal on Scientific Computing*, 33(5):2864–2887, 2011.

[86] A. H. Baker, R. D. Falgout, H. Gahvari, T. Gamblin, W. Gropp, T. V. Kolev, K. E. Jordan, M. Schulz, and U. M. Yang. Preparing algebraic multigrid for exascale. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2012.

[87] L. Wang, X. Hu, J. Cohen, and J. Xu. A parallel auxiliary grid algebraic multigrid method for graphic processing units. *SIAM Journal on Scientific Computing*, 35(3):C263–C283, 2013.

[88] X. Yue, S. Shu, and C. Feng. Ua-amg methods for 2-d 1-t radiation diffusion equations and their cpu-gpu implementations. In *2013 21st International Conference on Nuclear Engineering*. American Society of Mechanical Engineers Digital Collection, 2013.

[89] S. C. Khelifi, N. Méchitoua, F. Hülsemann, and F. Magoulès. A hybrid multigrid method for convection–diffusion problems. *Journal of Computational and Applied Mathematics*, 259:711–719, 2014.

[90] R. Gandham, K. Esler, and Y. Zhang. A gpu accelerated aggregation algebraic multigrid method. *Computers & Mathematics with Applications*, 68(10):1151–1160, 2014.

[91] B. Gmeiner, U. Rude, H. Stengel, C. Waluga, and B. Wohlmuth. Performance and scalability of hierarchical hybrid multigrid solvers for stokes systems. *SIAM Journal on Scientific Computing*, 37(2):C143–C168, 2015.

[92] F. Wan, Y. Yin, and S. Zhang. 3d parallel multigrid methods for real-time fluid simulation. *3D Research*, 9(1):8, 2018.

[93] K. Wu and H. Simon. Thick-restart lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.