Vol.2021-HPC-181 No.6
2021/9/27

SA-AMG 法における軽量な粗格子集約手法と富岳上でのウィークスケーリング性能評価

藤井 昭宏[1,a)]　田中 輝雄[1]　中島 研吾[2]

概要：Smoothed aggregation に基づく代数的多重格子法の大規模環境，大規模問題への対応を検討した．まず，大規模なプロセス並列へ対応するため，並列アグリゲーションの手法を元に軽量な粗格子集約手法を提案した．また粗い階層の未知数の規模が 32 ビット整数型の範囲を超える問題に対しても対応できるようにするため，64 ビット整数型の利用をマルチレベル生成時の一部に限定する実装手法を示した．これらの手法を適用し，スーパーコンピューター富岳の 4 万 9 千ノード以上を利用し，未知数の個数が 10^{12} 以上の分散 CRS 形式の疎行列を入力とした不均質拡散係数を持つポアソン問題を解き，本手法による代数的多重格子法のウィークスケーリング性能を示す．

1. はじめに

代数的多重格子法 (SA-AMG) は，様々な不規則疎行列の大規模線形問題に適用されている反復解法である．この解法は問題行列から，特性が似ている小さな行列（粗いレベル）を生成し，解を補正することで効率良く収束させる．粗いレベルは再帰的に生成され，マルチレベルな解法となっている．一方，スーパーコンピュータ「富岳」などをはじめとして，数百万 CPU コアを使い大規模な問題が解ける環境も多くなってきている．このような環境で，代数的多重格子法を効率良く動作させせることは，最も粗いレベルで行列サイズがコア数以下になることが明らかであり，その計算の集約構造から一般的に難しい．また，スーパーコンピュータの性能の飛躍的な向上から，線形問題の未知数の個数も 32 ビット整数型の範囲を大きく超える問題でも適用可能になってきている．このような環境では，計算資源を投入すれば，それに応じてどこまでも大きいサイズの問題が解ける見込みがあることも実装としては重要になってくる．そこで本研究では，数百万コアを利用できる環境において，代数的多重格子法の粗いレベルの効率の良い扱い方と，粗いレベルの行列サイズ（行数）が 32 ビット整数型の範囲を超えても動作する実装手法を示す．

著者らはこれまでに，代数的多重格子法における粗格子の並列度を集約する手法として，グラフ分割ライブラリ ParMETIS[1] を用いた手法を提案 [2] してきた．しかし，プロセス数が数万を超える環境ではこのライブラリを用いた隣接プロセスグラフを分割するコストは非常に大きくなり，4.2 節の実験結果に示されるように ParMETIS の処理時間がマルチレベル生成部分の時間の 50%以上を占める場合があった．

そこで粗格子集約手法が粗格子での任意のプロセス領域のグループ分けに対応できることを示し，より計算コストの軽い粗格子領域のプロセスグループ分けの手法を並列アグリゲーション手法の CLJP[3] を基に提案する．また，本手法をスーパーコンピュータ「富岳」を用いて，最大約 5 万ノード，200 万コア以上を用いてウィークスケーリングで 10^{12} 程度の未知数の Darcy 流れに基づく拡散係数が不均質なポアソン問題を解き，有効性の検証を行った．このサイズの問題では，粗いレベルも 32 ビット整数型の範囲を超えるものになっている．

本報告における貢献は下記のようになる．

- 大規模なプロセス並列へ対応できる，軽量な粗格子集約手法の提案
- 粗い階層の未知数の規模が 32 ビット整数型の範囲を超える問題に対して，64 ビット整数型の利用はマルチレベル生成時の一部に限定した，使用メモリ量の増加を最小限に押さえた実装手法
- 上記の手法を適用し，スーパーコンピューター富岳の 4 万 9 千ノード以上を利用し，DOF が 10^{12} 程度の分散 CRS 形式 (Compressed Row Storage) の疎行列を入力とした不均質拡散係数を持つポアソン問題を解き，SA-AMG 法のウィークスケーリング性能を示し

[1] 工学院大学
Kogakuin University
[2] 東京大学
The University of Tokyo
[a)] fujii@cc.kogakuin.ac.jp

ⓒ 2021 Information Processing Society of Japan

たこと

2 章には本稿で対象とする代数的多重格子法を概説し，3 章に本研究での粗格子集約手法や，実装について記述する．最後に数値実験にて有効性を考察しまとめる．

2. 代数的多重格子法

本研究は SA-AMG 法 [4] の実装手法について考えている．ここでは簡単なアルゴリズムの概略と本研究で関連がある項目を中心に記述する．

2.1 アルゴリズム概略

SA-AMG 法は問題行列 A_1 が与えられると，そこからマルチレベルな行列の階層構造を Algorithm 1 に基づいて作成する．この図では，レベル数は L としている．まずはじめに，2 行目の $aggregate$ 関数にて未知数の全体集合を関連の強い背反な未知数集合に分割し，予備的な補間行列 $\tilde{P}$ を生成する．$\tilde{P}$ の行数は，未知数全体の個数に，列数は未知数全体を分割した未知数集合の個数にそれぞれ対応する．$\tilde{P}$ の各列は未知数集合 1 個に対応し，その集合に含まれる未知数に対応する要素にのみ 1 が入り，それ以外の未知数に対応する要素には 0 を入れる．$\tilde{P}$ をそのまま補間行列として利用することもできるが，問題行列の情報を使い，$\tilde{P}$ の各列の値を初期ベクトルとして，$A_1x = 0$ に対する減速ヤコビ法を適用する (3 行目の $smooth$ 関数) ことで，精度を上げた，補間行列 P を計算する．最後に P^TAP を計算することで，1 段粗いレベルの行列を計算する．これを繰り返し，未知数の個数がより小さくなった問題行列を生成し，問題行列も合わせて L 個の行列，$L-1$ 個の補間行列 P を生成する．

マルチレベルを生成した後，Algorithm 2 に記述されている V サイクルにより反復的に解を収束させる．本研究では，V-cycle 1 回の処理を CG 法の前処理として適用している．A_1, x_1, b_1 はそれぞれ，問題行列，初期解ベクトル，右辺ベクトルを表しており，$A_1x_1 = b_1$ を満たすように x_1 を繰り返し修正していくことになる．Algorithm 2 の最初の for 文は，ガウスザイデル法などの定常反復法を $smoother$ 関数により適用し，残差ベクトルを計算しそれを P^T により粗いレベルに写像し，そのレベルの右辺ベクトルとしてセットする．この処理を一番粗いレベルまで繰り返し，一番粗いレベルでは未知数の個数も十分小さいため，6 行目の $solve$ 関数では直接解法で解く．その粗いレベルの解を 1 段上の未知数の多いレベルに写像し，足し込むことで，解が補正される．これは 8 行目の処理に対応する．その後，定常反復解法を適用することで，解を修正し，そのレベルでの解により近づける．これを一番未知数の多いレベルまで繰り返すことで，解ベクトルがマルチレベルな構造を使って効率良く補正される．

Algorithm 1 Setup part, In: A_1, Out:$A_i, P_i, i = 2..L$

```
1: for i ← 2 to L do
2:     P̃_i ← aggregate(A_{i−1});
3:     P_i ← smooth(P̃_i);
4:     A_i ← P_i^T A_{i−1} P_i;
5: end for
```

Algorithm 2 V-cycle, In: A_1, x_1, b_1, Out:x

```
1: for i ← 1 to L − 1 do
2:     x_i ← smoother(A_i, x_i, b_i);
3:     b_{i+1} ← P_i^T (b_i − A_i x_i);
4:     x_{i+1} ← 0;
5: end for
6: x_L ← solve(A_L, x_L, b_L)
7: for i ← L − 1 to 1 do
8:     x_i ← x_i + P_{i+1} x_{i+1};
9:     x_i ← smoother(A_i, x_i, b_i);
10: end for
```

2.2 並列実装

Algorithm 1 のマルチレベルの行列生成部と Algorithm 2 の V-サイクルの処理があるが，$aggregate$ 関数以外はガウスザイデル系緩和法であったり，疎行列疎行列積であるため，疎行列ベクトル積と同様な方法で並列計算ができる．詳細は論文 [5] の通りに作成しているが，ここでは並列実装の方針を簡単に説明する．まず疎行列のデータ構造としては，ブロック行分割にし，各プロセスのランク番号順にブロック行を割り当てるものとしている．このように未知数がプロセス番号に 1 次元的に連続して並んでいると仮定することで，プロセスごとの担当未知数の個数だけを記録すれば，未知数番号からどのプロセスが保持しているか，わかることになる．また各プロセスごとに担当する未知数とその境界領域（のりしろ）の未知数を合わせてローカルな未知数番号で計算をすすめることで，全体での未知数番号を意識することなく疎行列ベクトル積などができるようになる，一方，のりしろ部分の未知数は担当するプロセスから更新された新しい値をとってくる必要があるため，どのプロセスの何番目の未知数かテーブルとして保持する必要がある．これを通信テーブルと呼び，疎行列ベクトル積を行う前に，隣接通信を行い，のりしろ領域にある未知数の値を更新する必要がある．この未知数の接続のされ方は行列に依って当然異なり，通信パターンは変わってくる．Algorithm 1 からもわかるように，各レベルごとの行列 A_i とレベル i からレベル $i-1$ へと未知数を写像する行列 P_i があり，それぞれ固有に通信テーブルを設定している．一般的に，行列が効率的に分散されている時は，のりしろにある未知数の個数が最小化されており，あるプロセスから見たときのりしろによりつながっている隣接プロセス数も小さくなる．数値実験では各レベルの疎行列の隣接プロセス数とレベル間の補間行列 P の隣接プロセス数が記載されている．

$aggregate$ 関数部分は未知数集合を背反する強連結している未知数集合に分けることになるが，各プロセスは領域分割により，自分の担当未知数を保持している．未知数集合をこの担当未知数内で作成することにして，領域境界には未知数集合を作らないことで各プロセスが担当領域を独立に分けることができ，並列に $aggregate$ 関数の処理ができる．このアグリゲート生成戦略を decoupled[6] と呼ぶ．実装が単純になる反面，高並列になった場合に，プロセス並列度より小さいサイズの粗いレベルが生成できないため，問題になることが知られている．一方，領域境界の上に未知数集合を作っていく，coupled 戦略の場合，プロセス並列度より小さく粗くすることができるため，高並列時の粗いレベルの安定性につながることが知られている [6]．しかし，いずれにしても超高並列時には最も粗いレベルに近いところでは，各プロセスが未知数を数個しか担当しなくなり，ほとんどが通信時間になる，という問題が出てくる．

3. 粗格子集約手法とマルチレベル生成戦略

3.1 粗格子集約手法

低並列時から超高並列時まで，また問題サイズも小さいものから大きいものまで対応するマルチレベル生成手法を考慮すると，粗いレベルでは適切に各プロセスの担当領域の大きさを確保しながら，並列度を下げる手法が重要となる．そのため，我々は粗格子集約手法 [2] を研究してきた．PETSc の SA-AMG ソルバや Adams らの研究 [7]，Lin らの研究 [8] でも粗いレベルの行列生成後，再分散することで，並列度の集約を行った性能が報告されているが，本研究では粗いレベルを生成してから再分散するのではなく，並列度が適切に調整された粗いレベルを一度に作成する手法となっている．具体的には，Algorithm 1 の行列 P を一度生成した後で，P の列番号を入れ替えをすることで，粗いレベルの行列の行の入れ替えができ，ブロック行分割にて各プロセスの担当行数を変化させることで，4 行目の三つの行列の積で適切な分散が実現された行列を生成する [2]．

細かいレベルの行列を $A = [a_{i,j}]$ と表記し，その行列から生成した粗いレベルの行列を $A_c = [ac_{i,j}]$ とする．$A_c = P^T AP$ により計算されるが，要素ごとに分けると $ac_{I,J} = \sum_{i,j} p_{j,I} * a_{i,j} * p_{i,J}$ を計算することになる．$[a_{i,j}]$ はブロック行分散されており，プロセス r が担当する行番号の範囲を $s_r..e_r$ とすると，式 (1) の内側の $\sum$ をプロセスごとに並列計算を行う．

$$ac_{I,J} = \sum_{0 \le r < procs} \sum_{s_r \le i \le e_r, j} p_{j,I} * a_{i,j} * p_{i,J} \quad (1)$$

粗いレベルの行列 A_c もブロック行分割をされており，上の式で計算した結果の $ac_{I,J}$ の行番号 I の値によって担当プロセス番号が決まり，そのプロセス上で外側の $\sum$ の和をとることにより行列 A_c を計算できる．行列 A_c の分散状況を調節することが粗格子集約に対応するが，行列 P の列（列番号）を入れ替えることが式 (1) の I を付け替えることに対応し，それにより任意の分散状況が実現できる．

次に粗格子集約をする場合の I の付け替えについて説明する．まず粗いレベルで 1 プロセスに担当させる隣接するプロセスグループを決める．その後，隣接する複数のプロセス領域の未知数集合の番号（行列 P の列番号）を連番になるように入れ替え，かつ，連番にした粗いレベルの行列のブロック行の担当をプロセスグループ内のどれかのプロセスに担当させる．

そのため，粗格子集約を行うには，隣接プロセスのプロセス集合を事前に作成する必要がある．ここでは，細かいレベルの未知数の隣接関係をつかい，各プロセスが生成した粗いレベルの未知数の個数を，頂点の重みとした隣接プロセスグラフを作成し，それをグルーピングすることで対応する．本論文では，グラフ分割ライブラリ ParMETIS による手法と，プロセスグラフ上での隣接集合を並列に作る（CLJP）に基づく手法について考える．

3.1.1 ParMETIS による手法

このグラフを使い，1 プロセスあたりの想定未知数の個数になるように分割数を定め，ParMETIS[1] ライブラリを利用し点の重みの合計が均等になるようにこのグラフの点集合の分割を行う．これにより結合すると 1 プロセスあたり適切なアグリゲートの個数になるプロセス集合を生成し，下記のことが実現できる．

- 粗いレベルになっても，平均的に 1 プロセスあたり一定個数以上の未知数を保持するように適切に並列度を調整する．すなわち，未知数を保持しないプロセスを必要に応じて作成する．担当未知数がないプロセスはそのレベルでは計算に参加しない．
- ParMETIS によりプロセス間隣接グラフを領域分割するが，それにより各領域内の点の重みの合計（粗いレベルの未知数の個数）を同程度になるようバランスさせながら，領域間で切断される枝の本数を最小化させている．つまり，粗いレベルにおけるプロセス間の隣接プロセス数が少なくなるようにしながら，合計の未知数のサイズは同程度になるように結合するプロセス集合を決めている．

一方で ParMETIS ライブラリは一般的に並列度（プロセス数）より節点数が多いグラフの分割が想定されており，各プロセスが 1 頂点となるような，並列度に対して極端に小さいグラフに対しては性能が出しにくく，ParMETIS ライブラリのコストが粗格子集約の中でボトルネックになりうる．

3.1.2 CLJP をベースにした手法

ParMETIS による手法より，コストの少ない手法として生成した隣接プロセスグラフ上で，CLJP アグリゲーション [3] を修正した手法を考える．CLJP アグリゲーションは，問題行列から粗いレベルの未知数集合を並列に作成す

るアルゴリズムである．

CLJP アグリゲーションを用いると，プロセス隣接グラフの中で，まず枝 3 本以上離れているノードの極大集合をグループの中心節点として使うノードとして求め，その点を中心に隣接している頂点をグループ化し，プロセス集合を生成する手法となる．「グループの中心節点として選択される」優先度を乱数を用いて全ての節点で同じ値がないように設定し，枝距離 2 の隣接節点の中で優先度が自分のノードが最も高い場合に，中心節点として選定されるアルゴリズムである．中心節点として設定されると，その枝距離 2 以内の隣接節点は候補点からは外し，枝距離 3 の節点でまだ中心節点の候補として残っている節点があれば，その優先度をインクリメントする．この処理を中心節点の候補がなくなるまで繰り返すことで，極大集合を作り，その後，その点を中心に周囲の節点をグループ化する手法である．

ただ，プロセスの隣接グラフの場合，次数が小さい点を中心にグループになるように，次数が小さい点の優先度を高くし，かつ，優先度を乱数で設定せず，同じがあっても良いものとした．枝距離 2 以内で優先度が最大値であれば，同じ優先度の節点が複数あっても中心節点として選択することにする．次にデータ構造も含めて記述する

プロセスグラフは分散して保持され，1 プロセス 1 頂点を保持するグラフ上で行なうことになり，その場合のグループごとの中心となる節点を選ぶアルゴリズムを Algorithm 3，中心となる節点情報からプロセスグループ番号を割り当てるアルゴリズムを Algorithm 4 として記述した．アルゴリズム中の $pri1[0:N]$ は，各プロセスが保持している配列で，$pri1[0]$ に自プロセスの優先度，$pri1[1:N]$ は隣接プロセスの優先度の情報を隣接通信により入れる配列である．N はこのプロセスの隣接プロセス数である．$pri2[0:N]$ は $pri1$ と同様のサイズの配列であり，枝距離 2 以内で最大の優先度を求めるために使用する．

Algorithm 3 は，各頂点の優先度を設定し，枝距離 2 本以内で隣接する頂点の中で最も優先度が高い頂点を中心点として選定するアルゴリズムである．本稿では，優先度は隣接プロセス数の逆数とし，隣接プロセス数が小さいノードの優先度を高めた設定とした（1 行目）．中心点として選択された節点は優先度を -2 と設定した (8 行目)．ここで選択された頂点から枝距離 2 以内でまだ選択されていない頂点はプロセスグループの中心となる候補から外し（10〜17 行目)，枝距離 3 の頂点の優先度はインクリメントすることでより多くの節点が選択できるようにしている（18〜21 行目)．中心となる点が決まったら，そこから枝距離 1 の点の優先度は -1 とし，枝距離 2 の点の優先度は 0 としており，優先度が正ではなくすることで候補点からは外している．このアルゴリズムの手順から，5 回隣接通信を行なうことで，中心点を選択するまでの処理を一通り行なうことができ，中心点として考慮すべき点がなくなるまでこの処理を繰り返す．このアルゴリズムを 81 プロセスで 9×9 の格子上に隣接関係で繋がっている時に，求まったグループの中心点を赤で塗ったものを図 1 に示す．この図の場合，節点は 25 個の点が選択され，そのための WHILE 文の繰り返しが 1 回のみ必要で，トータルとして隣接通信 5 回が必要なコストになっていた．隣接通信のコストが気にならない環境では，完全に並列に，かつ計算量を抑えて処理することができる．周辺の中で優先度が最大であれば，中心点として選択されるため，同一の優先度が周辺にありそれが最大値となっていると複数の隣接している点が中心点として選ばれることになる．

次に Algorithm 4 は，選択された節点集合から各プロセスグループを作るアルゴリズムである．$dom[0:N]$ は，プロセスグループの番号を入れる配列で，自分の番号は $dom[0]$ に入れる．$pri1$ の配列などと同様に隣接通信の対象となっていて，隣接プロセスのグループ番号も記録できる配列である．まず初めに，一度，グループの中心点として選択された節点集合の情報を $pri1all$ 配列に共有するため，MPI_Allgather 関数を処理する．$pri1all$ は全プロセスの優先度が共有されるが，-2 のプロセスが中心点として選択されたものになる．$pri1all$ の先頭から -2 があったらカウントアップしていき -2 と置き換えることで，そのプロセスのグループ番号が決められる（6 行目)．自プロセスが中心節点として選択されていた場合（$pri1[0] = -2$）は 10 行目でプロセスグループ番号を dom 配列に代入する．選択された節点が隣接プロセスとなっていれば（$pri1[0] = -1$)，17 行目で所属するグループ番号が設定される．19 行目で隣接プロセスとグループ番号を交換し，まだ決まっていない場合は，隣接するプロセスグループでより大きい通信テーブルでつながっているところのグループ番号とする（21 行目)．図 1 の節点集合に基づいて生成したプロセス集合をノードごとに色分けをすると図 2 のようになった．最初に 25 点選択されているので，全体を 25 個のグループに分ける形になっている．色は循環して割り当てているため，同じ色のプロセスグループがあるが、隣接していないグループは別のプロセスグループになっている．

ParMETIS による手法と比較して，下記のような状況となる．

- グループ分けの処理がほぼ完全に並列に動作し，隣接通信をせいぜい数十回と Allgather 通信 1 回のコストでプロセスグループの中心点が生成できる．
- CLJP による手法では，各ノードの重み情報を利用しておらず，結合されたプロセス領域全体の未知数集合同士で，均等にはならない．但し，並列度に対して未知数が少なすぎる時の領域集約を想定しているため，負荷の均等性はそれほど重要にはならない．
- ParMETIS の場合，プロセス集合のグループ数（粗い

レベルの並列度）を指定できていたが，本手法の場合，隣接プロセス領域をまとめてノード集合を分ける手法となっており，事前に粗いレベルの並列度は指定できない．

Algorithm 3 Parallel node selection on MPI process adjacency graph, In: N, $N_{max}, NEIBPE[1:N]$, Out: $pri1[0] == -2$ shows selected node as center of a process group. N is # of neighboring processes, N_{max} is max number of N among all processes.

1: $pri1[0] \leftarrow N_{max}/(N+1)$;
2: **while** some processes' pri[0] > 0 **do**
3: $pri1[1:N] \leftarrow halo_communication(pri1[0])$;
4: $pri2[0] \leftarrow max(pri1[0:N])$;
5: $pri2[1:N] \leftarrow halo_communication(pri2[0])$;
6: $max_pri_neib2 = max(pri2[0:N])$;
7: **if** $pri1[0] = max_pri_neib$ **then**
8: $pri1[0] \leftarrow -2$;
9: **end if**
10: $pri1[1:N] \leftarrow halo_communication(pri1[0])$;
11: **if** $(pri1[0] >= 0) \&\& (-2\ in\ pri1[1:N])$ **then**
12: $pri1[0] \leftarrow -1$;
13: **end if**
14: $pri1[1:N] \leftarrow halo_communication(pri1[0])$;
15: **if** $(pri1[0] > 0) \&\& (-1\ in\ pri1[1:N])$ **then**
16: $pri1[0] \leftarrow 0$;
17: **end if**
18: $pri1[1:N] \leftarrow halo_communication(pri1[0])$;
19: **if** $(pri1[0] > 0) \&\& (0\ in\ pri1[1:N])$ **then**
20: $pri1[0] \leftarrow pri1[0] +$ number of 0 in $pri1[1:N]$;
21: **end if**
22: **end while**

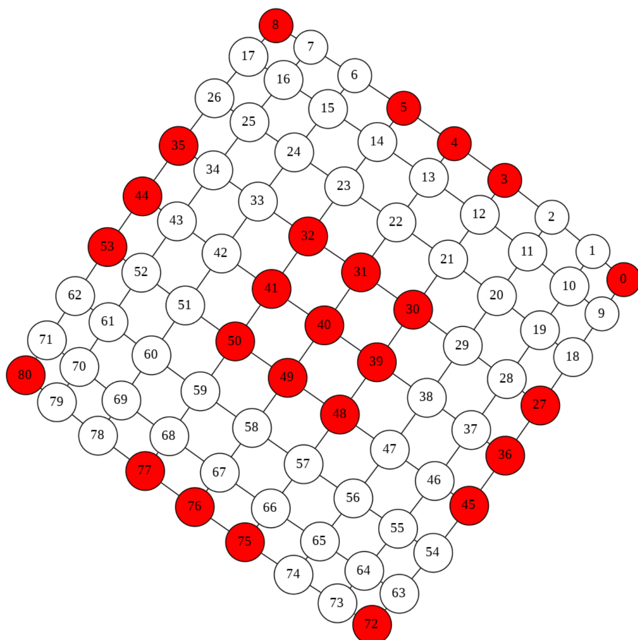

図 1　プロセスグループの中心節点
Fig. 1　Center nodes of process groups

Algorithm 4 Domain id determination with selected node set, In: $pri1[0]$, Out: $dom[0]$, which shows the process group id.

1: $pri1all[0:procs-1] \leftarrow all_gather(pri1[0])$;
2: $count = 0$;
3: **for** $i = 0$ to procs-1 **do**
4: **if** ($pri1all[i] = -2$) **then**
5: $count = count + 1$;
6: $pri1all[i]$=count;
7: **end if**
8: **end for**
9: **if** $(pri1[0] = -2)$ **then**
10: $dom[0] \leftarrow (pri1all[my_rank])$;
11: **else if** $(pri1[0] = -1)$ **then**
12: **for** $i = 0$ to N-1 **do**
13: **if** $(pri1all[NEIBPE[i]] > 0)$ **then**
14: break;
15: **end if**
16: **end for**
17: $dom[0] = pri1all[i]$;
18: **end if**
19: $dom[1:N] \leftarrow halo_communication(dom[0])$;
20: **if** ($pri1[0] = 0$) **then**
21: $dom[0] = dom[i]$ where $dom[i] > 0$ && large halo with $NEIBPE[i]$.
22: **end if**

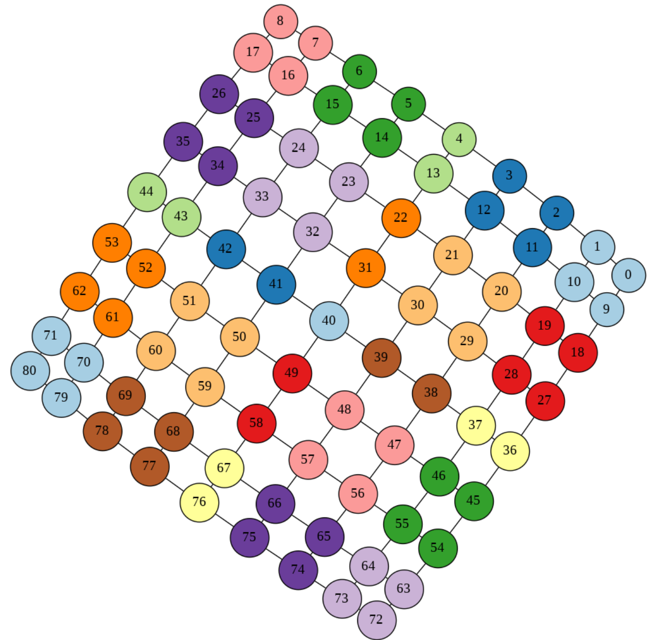

図 2　中心節点情報から生成したプロセスグループ
Fig. 2　process groups based on the center nodes

3.2　32 ビット整数型を超える大規模問題への対応

分散疎行列のデータ構造では，行列全体の行番号や列番号を表すグローバル番号は利用していない．そのため 1 プロセスの保持している行列が 32 ビットの表現範囲を越えなければ，並列度を上げることで 32 ビットの表現範囲を超えるサイズの行列も扱うことができる．

しかし，粗いレベルを生成するときに，式 (1) の行列 P

ⓒ 2021 Information Processing Society of Japan

表 1 スーパーコンピュータ 「不老」(type I subsystem) / スーパーコンピュータ 「富岳」1 ノード
Table 1 Supercomputer Flow(type I subsystem) / Supercomputer Fugaku 1 node

CPU name	A64FX (Arm v8.2-A + SVE)
number of cores/socket	48 + 2/4 for OS
number of sockets/node	1
frequency	2.2/2.0 GHz
memory	32GiB(HBM2)
Interconnect	TofuD
compiler	Fujitsu Fortran

の列番号 I はグローバル番号で保持し，各プロセスの担当範囲をグローバル番号で決めておくことで，行列 P の列番号 I から担当プロセスが決まり，粗いレベルの行列が計算できる．この行列 P のグローバル列番号に相当する値と，粗いレベルでの各プロセスの担当未知数番号の範囲を表す数値のみ 64 ビット整数型として保持することで，粗いレベルが 32 ビット整数型の範囲を超えてもほとんどメモリサイズを増やさず対応した．粗いレベルの行列が生成された後は，64 ビット整数型の情報は不要となるため，通常の 32 ビット整数型と 64 ビット浮動小数点数の値を使った分散疎行列のデータ構造でその粗いレベルも保持される．

4. 数値実験

4.1 問題設定と目的

ウィークスケーリングの環境において大規模問題における，粗格子集約手法の効果を調べることを目的として，4.2 節では従来の ParMETIS を用いた領域集約手法と本研究での手法との比較を行う．さらに 4.3 節ではスーパーコンピュータ「富岳」上で，本手法によるウィークスケーリング性能を分析する．

問題として，三次元立方体形状の有限体積法に基づくポアソン方程式 $\frac{\partial}{\partial x}(\lambda\frac{\partial T}{\partial x}) + \frac{\partial}{\partial y}(\lambda\frac{\partial T}{\partial y}) + \frac{\partial}{\partial z}(\lambda\frac{\partial T}{\partial z}) = f$ を対象とした．T が未知数であり，右辺の f は定数である．拡散係数 λ が不連続に変化する問題で，最大 10^5 程度から 10^{-5} 程度まで変化する問題である．ガウスザイデル前処理付き CG 法などでは収束しない問題となっている．

4.2 節では，スーパーコンピュータ「不老」(Type I サブシステム）[9] を 768 ノード利用し，なるべくプロセス数を多くした Flat-MPI 環境での実験を行った．4.3 節ではスーパーコンピュータ「富岳」を用いて１ノード 4 プロセス 12 スレッドで最大 49152 ノードを利用し自由度が最大 1.17×10^{12} の問題まで解いた．スーパーコンピュータ不老や富岳の 1 ノードの構成は表 1 のようになる．

4.2 粗格子集約手法間での比較

本節の実験では，スーパーコンピューター不老 Type1 サブシステムを利用し，1 ノードあたり，48 プロセスを起動し，フラット MPI で利用する形で実行した．プロセス集約の効果を出しやすくするため，1 プロセスあたりの担当未知数は最初からかなり少なめにし，高並列の環境で評価する．具体的には，1 プロセスあたり $10 \times 10 \times 10$ の未知数を割り当て，1 ノードあたり，4.8×10^4 個の未知数を割り当てた．768 ノード，36864 プロセスを使い，$240 \times 320 \times 480$ の 3 次元問題（3.68×10^7 の自由度）を解いた．

比較対象としては， 3.1.1 節で挙げた ParMETIS ライブラリによる粗格子集約と， 3.1.2 節で説明した CLJP アグリゲーションを修正した粗格子集約手法である．粗格子集約の設定としては，各プロセスが粗いレベルで平均的に 500 個の未知数を保持出来なさそうであれば，並列度を集約する．それ以外のソルバの設定は下記のようになる．レベル番号が参照されているが，レベル番号が１大きくなるごとに粗い，小さい行列サイズに対応し，レベル 1 が問題行列に対応する．

Solver AMG-CG method

Cycle V-cycle

Smoother at each lev. 1 iteration of Symmetric Gauss-Seidel method with acceleration coef. of 0.8. Unknown variables' dependency over process domain borders is ignored, and it becomes Jacobi type smoother over domain borders.

Coasest level LU factorization with 1 process

Strong connection threshold 0.01×0.8^l for level $l+1$

Convergence Criterion relative residual 2-norm of 1.0×10^{-11}

その結果，この図 3 のようになった．左側に反復解法部（Solve Part）の時間と右側にマルチレベル生成部（Setup part）の時間を棒グラフにしている．Metis と CLJP と書かれているがそれぞれ，ParMETIS による粗格子集約と CLJP を修正した粗格子集約をしたものを示している．Solve Part には棒グラフの上に数字が書かれているが，収束に要した反復回数である．これを見ると CLJP に基く粗格子集約手法の方が反復回数が少ないが反復解法部は時間が余計にかかっていることが分かる．

一方で，マルチレベル生成部を見ると，process aggregation の時間とそれ以外の時間に分けられている．process aggregation の時間は粗いレベルのプロセスのグループを作成する時間で ParMETIS ライブラリを呼び出す時間か，CLJP によるプロセスマッチングを生成する時間を表している．Metis の方ではマルチレベル生成部の半分以上の時間を使って粗いレベルのプロセスマッチングを計算しており，明らかに問題がある．一方 CLJP にすることでプロセスマッチングを作る時間はほとんど見えない程小さくなった．

表 2 と表 3 にそれぞれの粗格子集約手法を適用した時の，各レベルの行列の状況が整理されている．疎行列の情

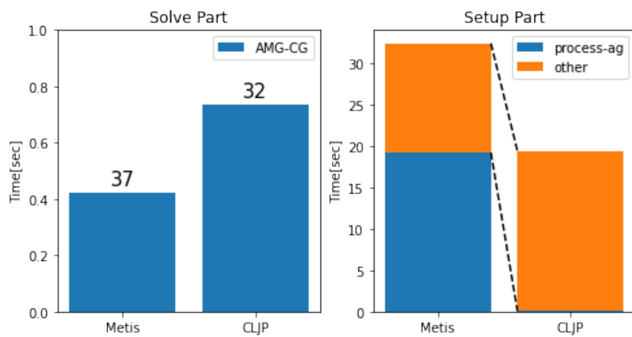

図 3 粗格子集約手法の比較

Fig. 3 Performance comparison of Coarse Grid aggregations

報，分散された疎行列の隣接プロセス数の情報，の二つのパートに分かれている．まず最初のパートは，DOF，# of P, ave. nonz, はそれぞれ行列の行数，分散されているプロセス数，各行あたりの平均非ゼロ要素数である．二つ目のパートには，各レベル，とレベル間の行列の隣接プロセスが最大何個あるかが書かれている．レベル 1 の max neib は 26 個の隣接プロセスを持っているが，これは三次元直方体形状に区切っているため，最大隣接 26 個のプロセス領域とデータのやりとりが発生していることを示す．bet levs はレベル間演算子の分散疎行列の隣接関係を表している．ここでレベル 2 に書かれている値はレベル 2，1 間の補間（縮約）行列の情報が書かれているため，レベル 1 の所は空白になっている．

これらの表の隣接プロセス数を見ても，ParMETIS により並列度を集約すると max neib がせいぜい 31 個に抑えられているのに対し，CLJP をベースにした手法で対応すると，隣接プロセス数が 100 個になるものも出てきていた．レベル間演算子についてはさらに，その傾向が強くなり，最大 600 個の隣接プロセスとのやりとりを必要としていた．このため，反復回数が CLJP は 32 回と少ないにもかかわらず，反復解法部の時間が倍近くかかってしまったと推測される．ParMETIS で粗格子集約をすると，集約後のグループ数も指定できるため，より適切に分散できる一方，超高並列になるとそのコストが許容できなくなるほど高くなることが分かる．今回の問題設定では，1 プロセスあたりの自由度がかなり小さく，極端に並列度が高い問題設定になっているため，行列生成部のコストが反復解法部の 20 倍以上となっていた．

4.3 スーパーコンピュータ富岳上での WeakScaling 評価

この節では，スーパーコンピュータ富岳を用いてウィークスケーリング性能の評価を行った．実行条件を表 4 に示す．最大約 5 万ノードを用いて実験を行うため，ノード当たりの 4 プロセスとして設定した．ノード当たり 32GB であるため，1 ノードあたりの行列サイズは 8GB 程度になるようなサイズとし評価を行った．

表 2 ParMETIS による CGA での各レベルの行列

Table 2 Matrix information at each level using ParMETIS CGA

Lev.	DOF	# of P	ave. nonz	max neib	bet levs
1	36864000	36864	26.83	26	
2	1910213	3821	85.79	31	105
3	81598	164	132.58	30	157
4	5139	11	158.37	9	60
5	314	1	118.15	0	10

表 3 CLJP をベースにした CGA での各レベルの行列

Table 3 Matrix information at each level using CLJP CGA

Lev.	DOF	# of P	ave. nonz	max neib	bet levs
1	36864000	36864	26.83	26	
2	1910213	24640	85.79	38	119
3	127696	14212	170.61	100	160
4	46220	105	732.89	45	598
5	1764	5	326.61	4	77
6	70	1	57.80	0	4

表 4 ウィークスケーリングテスト設定

Table 4 Weak scaling test setting

Domain size of a process	$288 \times 144 \times 144$
Process/thread allocation on 1 node	4 MPI processes 12 Open-MP threads
CRS matrix size of 1 node	7.8GB (27 unknowns per row)
Number of nodes	from 768 to 49152 nodes 768×2^i $(i = 0 \ldots 6)$
Maximum number of cores and processes	2.35×10^6 cores 1.9×10^5 processes
Largest domain size	$13824 \times 9216 \times 9216$ 1.17×10^{12}DOF

ソルバの設定は前節 4.2 とほぼ同じ AMG-CG 法である．変更したところは，実行が MPI-OpenMP ハイブリッドになったため，緩和法はマルチカラー対称ガウスザイデル法とし，また収束条件は大規模な問題でも収束するように緩和し 2 ノルム相対残差で 1.0×10^{-6} 以下となった時とした．

収束時の実行時間と AMG-CG 1 反復あたりの時間をプロットすると，図 4 と図 5 のようになった．図 4 は横軸に問題サイズを取り，収束に要する全体の時間が Total として，マルチレベル生成部の時間が Setup としてプロットされている．図内の数字は収束に要した反復回数である．反復回数も 30 回から 37 回と増大しているものの，問題サイズは最小サイズから最大サイズまで 64 倍されているのに実行時間は 1.31 倍程度ということで，良好なウィークスケーリング性能を示している．図 5 には横軸問題サイズを取り，解法部の 1 反復あたりの時間がプロットされているがほぼ一定であり，問題サイズ最小のところから最大問題サイズになると 1.07 倍の反復時間となっていた．

最後に最大問題サイズの時の分散疎行列の状態を表 5 に

ⓒ 2021 Information Processing Society of Japan

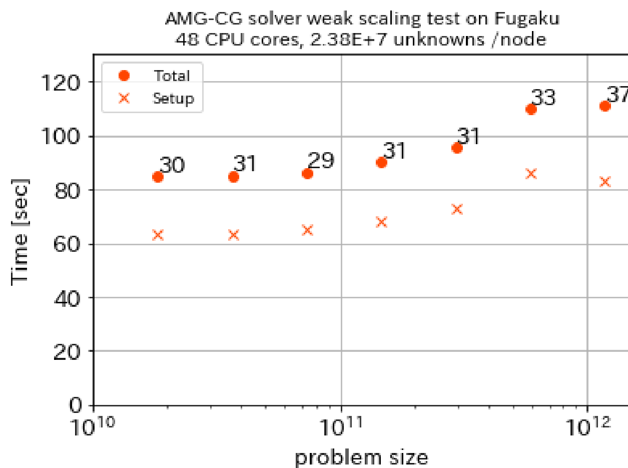

図 4 スーパーコンピュータ富岳上でのウィークスケーリング評価
Fig. 4 Weakscaling test on Fugaku

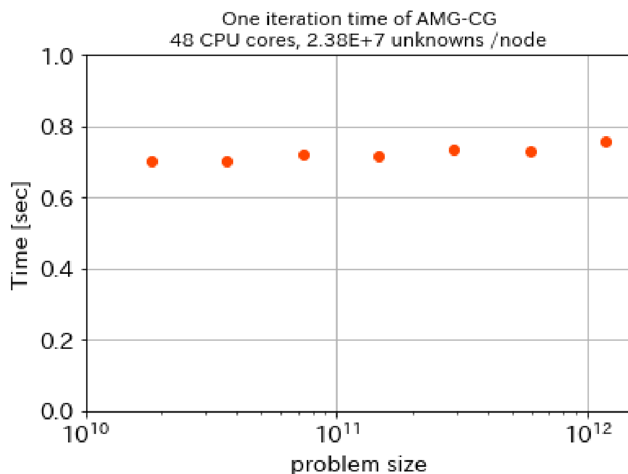

図 5 AMG-CG 反復解法部 1 反復の時間
Fig. 5 AMG-CG 1 iteration time

表 5 CLJP をベースにした CGA での各レベルの行列
Table 5 Matrix information at each level using CLJP CGA

Lev.	DOF	# of P	ave. nonz	max neib	bet levs
1	1174136684544	196608	26.99	26	
2	43201202689	196608	69.10	26	26
3	1218262155	196608	92.40	26	26
4	50184209	158400	115.88	38	117
5	2658219	123140	150.21	65	168
6	314456	1501	329.59	50	515
7	8223	37	194.09	26	232
8	257	1	103.45	1	36

示す．最大 8 レベル生成しており，1.17×10^{12} の未知数が 257 個の未知数の問題に集約されている．また，最大隣接プロセス数を見てもレベル内では，65 個程度に抑えられており，レベル間演算子では数百プロセスと連結されている部分もあるが，図 5 を見ても 1 反復あたりの時間はそれほど増大をしているわけでもなく，富岳の高い通信性能に下支えされており，今回の実験では大きなオーバーヘッドにはなっていなかった．

表 5 からレベル 2 の未知数のサイズが 4.32×10^{10} となっており，32 ビット整数型ではオーバーフローするが，32 ビット整数型で分散疎行列として表現できており，問題なく収束していることが確認できた．これにより，問題サイズがこれ以上大きくなっても，1 プロセスの保持する行列が 32 ビット整数型で表現できる範囲であれば，原理的にはソルバは適用可能であると想定される．

5. おわりに

本研究では，SA-AMG 法を対象に既存の並列アグリゲーションアルゴリズム（CLJP）をベースに，軽量な粗格子集約手法を提案し，グラフ分割ライブラリの ParMETIS による粗格子集約手法と比較を行った．ParMETIS による粗格子集約手法の方が粗いレベルの疎行列の隣接プロセス数や解法部の反復時間としてはより良い性質が出せていたが，ParMETIS のコストはプロセス数が 3 万プロセスを超えたグラフになってくると，マルチレベル生成部の半分以上を占めるようになり，それ以上のプロセス数の実行環境ではコストが大きくなりすぎる状況であった．本研究の CLJP をベースにした手法では，隣接通信を繰り返し粗格子集約をするが，そのコストはマルチレベル生成部全体からはほとんど見えない程度まで小さくできていた．

その粗格子集約手法を用いて，スーパーコンピュータ富岳上で，最大約 5 万ノード，約 20 万プロセスで，未知数の個数が 10^{12} 程度以上の分散疎行列による不均一拡散係数ポアソン問題に対しウィークスケーリングで性能評価を行った．ソルバの全体時間，解法部の 1 反復あたりの時間で評価したが，良好なウィークスケーリング性能が出せていた．一部 64 ビット整数型変数を利用することで大規模問題に対応しているが，32 ビットを超えるサイズの粗いレベルが必要な問題でも収束が確認できており，1 プロセスが扱うデータが 32 ビット整数型の範囲を超えない状況では，この先も並列度を増大させることでより大規模な問題へ適用可能であると想定される．

謝辞 本研究は学際大規模情報基盤共同利用・共同研究拠点，および，革新的ハイパフォーマンス・コンピューティング・インフラの支援による (課題番号: jh210026-NAH)．また HPCI システム利用研究課題 (課題番号：hp200299) を通じて，理化学研究所のスーパーコンピュータ「富岳」の計算資源の提供を受け，実施しました．

参考文献

[1] Karypis, G. and Kumar, V.: MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, http://www.cs.umn.edu/~metis (2009).

[2] Nomura, N., Fujii, A., Tanaka, T., Marques, O. and Nakajima, K.: Algebraic Multigrid Solver Using Coarse Grid Aggregation with Independent Aggregation, *2018 IEEE International Parallel and Distributed Processing*

posium Workshops (IPDPSW), pp. 1104–1112 (online), DOI: 10.1109/IPDPSW.2018.00170 (2018).

[3] Alber, D. M. and Olson, L. N.: Parallel coarse-grid selection, *Numerical Linear Algebra with Applications*, Vol. 14, No. 8, pp. 611–643 (online), DOI: https://doi.org/10.1002/nla.541 (2007).

[4] Vanek, P., Mandel, J. and Brezina, M.: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, *Computing*, Vol. 56, No. 3, pp. 179–196 (online), DOI: 10.1007/BF02238511 (1996).

[5] 藤井昭宏, 西田　晃, 小柳義夫：領域分割による並列 AMG アルゴリズム, 情報処理学会論文誌コンピューティングシステム（ACS）, Vol. 44, No. SIG06(ACS1), pp. 9–17 (2003).

[6] Tuminaro, R. and Tong, C.: Parallel Smoothed Aggregation Multigrid : Aggregation Strategies on Massively Parallel Machines, *Supercomputing, ACM/IEEE 2000 Conference*, pp. 5–5 (online), DOI: 10.1109/SC.2000.10008 (2000).

[7] Adams, M., Bayraktar, H., Keaveny, T. and Papadopoulos, P.: Ultrascalable Implicit Finite Element Analyses in Solid Mechanics with over a Half a Billion Degrees of Freedom, *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, pp. 34–34 (online), DOI: 10.1109/SC.2004.62 (2004).

[8] Lin, P. T.: Improving multigrid performance for unstructured mesh drift–diffusion simulations on 147,000 cores, *International Journal for Numerical Methods in Engineering*, Vol. 91, No. 9, pp. 971–989 (2012).

[9] Supercomputer Flow: Information Technology Center, Nagoya University, https://icts.nagoya-u.ac.jp/ja/sc/overview.html.

ⓒ 2021 Information Processing Society of Japan