Vol.2015-HPC-149 No.8
2015/6/26

SA-AMG 法における粗格子領域集約と性能分析

野村直也[†1]　野村卓矢[†1]　藤井昭宏[†1]　田中輝雄[†1]

大規模な非構造格子問題を高速に解く線形解法のひとつに，代数的マルチグリッド法（AMG 法）がある．AMG 法は問題行列から次元数の異なる複数の行列を生成して解く手法である．しかし，高並列時では粗いレベルでそのまま処理を実行することは性能劣化につながる．そこで本研究では，SA-AMG 法に対して粗いレベルの領域集約（Coarse Grid Aggregation Method，CGA）により，並列度を適宜落とす手法について評価を行う．評価の内容としては，CGA に用いられる領域集約の閾値のパラメタである C の値の最適値を求めるために，1 回の V-cycle のメッセージ数と，1 反復あたりの実行時間を比較し分析を行い，本実験では C が 1000 のときに最適となることが確認された．

1. はじめに

数値計算の応用において，大規模な連立一次方程式 $Ax=b$ は様々な場面で現れ，高速に解くことが求められている．大規模な連立一次方程式を解く解法のひとつに AMG（Algebraic Multigrid）法[1]がある．AMG 法は問題行列から，次元数の異なる複数の行列を生成して解く手法である．AMG 法のなかの解法のひとつに，Smoothed Aggregation に基づく AMG 法（以下，SA-AMG 法）[2][3][4]があり，本研究ではこの SA-AMG 法を対象とする．

SA-AMG 法は，問題生成部（以下，構築部）と反復解法部（以下，解法部）から構成される．構築部はアグリゲートと呼ばれる節点集合を作成し，作成したアグリゲートを用い問題行列を代数的に粗くする．これを再帰的に行うことで，次元数の異なる複数の行列を作成する．解法部は，構築部で階層的に生成された行列に対し，Jacobi 法などの緩和法を用いて問題行列を解く．このとき，問題行列など大規模な行列が設置される階層（レベル）を細かいレベル，問題行列から生成された小規模な行列が設置される階層（レベル）を粗いレベルと呼ぶ．

現在の計算機環境は，マルチコアやメニーコアといった並列性の高い環境が多く，今後はより高い並列性が求められることが考えられる．そのため，高並列環境下においても高い計算効率を実現する必要があると考えられる．本研究は SA-AMG 法を，OpenMP と MPI を組み合わせた，Hybrid 並列で並列実行する[5]．そして，問題行列を領域分割法により，各プロセスに分割する．ただし，粗いレベルでは行列の非ゼロ要素数が増え，サイズも小さくなるため，高並列時に処理をそのまま実行することは性能劣化につながる[5]．そのため，粗いレベルで適切に並列度を集約させる手法が重要となる．

そこで我々は，高並列時に問題となる粗いレベルの計算は，粗いレベルの領域集約（Coarse Grid Aggregation Method：CGA）により並列度を適宜落とす手法[6]を研究してきた．CGA は，構築部においてアグリゲートが生成された段階で，ParMETIS ライブラリ[7]を用いて領域の組み合わせを決定し，領域集約を行う．CGA により，粗いレベルの生成時においての行列の再分配などの必要はなくなり，負荷が小さくなると考えられる．

CGA では，1 プロセスが担当するアグリゲートの個数の閾値を設定し，閾値より個数が少なくなれば領域集約を行う．この閾値の設定により，集約の度合いが調節でき，性能にも影響を及ぼすことが考えられる．

そこで本研究では，この領域集約の閾値の最適値を求めるために，閾値を変化させることによる性能への影響の分析を行った．領域集約では，メッセージ数を減らせるメリットが大きいものと考えられる．そのため，最適値を推定するために，メッセージ数のモデルを作成し，実行時間との関係性を評価し，CGA の有用性を分析した．

2. AMG 法

この節では，AMG 法についての概略を説明する．AMG 法では，与えられた問題行列から階層的に小規模な問題行列を作成していく．2.1 節では，AMG 法についての説明を行い，2.2 節で SA-AMG 法の並列アルゴリズムを示す．

2.1 AMG 法

AMG 法は，大規模な非構造格子による線形問題を高速に解く数値解法のひとつである．まず，与えられた問題行列を複数段階に分けて小規模な行列を生成し，これらを用いて問題行列を解く．AMG 法は大きく分けて構築部と解法部の 2 つの処理からなる．構築部は問題行列から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，粗いレベルの行列を生成する．これを再帰的に行うことで，階層的な行列構造を作る．一方，解法部は構築部で生成された行列を使い，実際に問題を解く．

†1 工学院大学
Kogakuin University

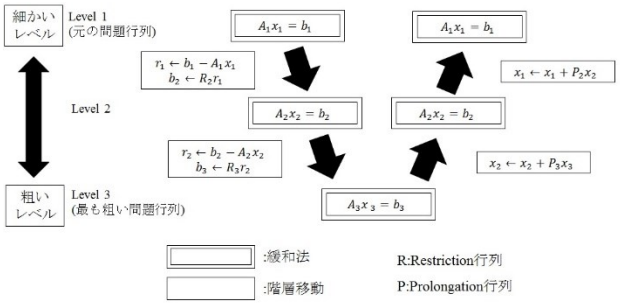

図 1　解法部（V-cycle）

解法部の構造を図 1 に示す．解法部は，主に行列ベクトル積と緩和法から成り立っている．この図では，最上層に与えられた問題行列が設置され，階層が下がるにつれて，問題行列より小規模な行列が設置される．階層数は問題サイズによって可変となる．階層移動では，構築部で用意した補間演算子の 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．複数の階層を行き来する様子が V 字を連想させるため，このような解法部を V-cycle と呼ぶ．

補間演算子から，行列の階層構造が生成されるため，この補間演算子の生成手法により，さまざまな AMG 法が存在する[8]．本研究では，SA-AMG 法を対象とし，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．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．

2.2 SA-AMG 法

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考える．問題行列の各行と未知数は節点と対応し，非ゼロ要素は辺と対応している．

節点全体をアグリゲートと呼ばれる節点集合に分解する．アグリゲートは図 2 のように次の粗いレベルでひとつの節点に対応し，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．任意の節点がどこかひとつのアグリゲートに属するように，アグリゲートが生成される．このアグリゲート内の未知数に重み付けをすることで，補間演算子を計算し，行列の構造を作成する．

2.2.1 構築部

図 2 に SA-AMG 法の構築部のアルゴリズム[2][3][4]を示す．構築部では，主に図 2 の（1）から（4）の手続きを行う．図 2 において，LEVEL は最大レベル数，*lev* は現階層のレベル番号，*clev* は現階層より次の粗いレベル数を表す．図 2 の，構築部の(1)から(4)の手続きを説明する．

(1)　フィルタリング

問題行列A_{lev}が与えられると対角要素の値に対し，設定した閾値より割合の小さな非ゼロ要素を取り除いた行列$\tilde{A}_{lev}$を生成する．本稿の数値実験は閾値を 0.02 としている．フィルタリング後の行列$\tilde{A}_{lev}$の各行は節点を示し，その行の非ゼロ要素は各節点との接続関係を示す．

(2)　アグリゲート生成

グラフ上で隣接する未知数の集合（アグリゲート）を作り，行列$\tilde{A}_{lev}$が示す補間行列$\tilde{P}_{clev}$を生成する．ここで，*clev* は次のレベル番号を表している．また$\tilde{P}_{clev}$は縦長の行列であり，各行は節点を示し，各列はアグリゲートに対応し，そのアグリゲートに含まれる節点のみ 1 が要素として入っている．それ以外の要素には 0 が入っている．

(3)　アグリゲートの重み付け

アグリゲートの節点に適切に重み付けを行う．行列$\tilde{P}_{clev}$をそのまま補間演算子として用いてもよいが，収束性を高めるため緩和法を 1 回適用し，行列P_{clev}を生成する．緩和法は減速ヤコビ法を用いた．

(4)　次レベルの行列生成

次レベルの行列を生成するために行列積を行う．問題行列A_{lev}と(3)で求めた補間行列P_{clev}，行列P_{clev}を転置した行列R_{clev}について行列積$R_{clev}A_{lev}P_{clev}$を行い，次レベルの行列A_{clev}を生成する．つまり，P_{clev}とR_{clev}はそれぞれ *lev* と *clev* 間のレベル間演算子となる．

/*構築部*/
Input: A_1, x_1, b_1
/*A_1からA_2,…, A_{LEVEL}と*/
/*P_1からP_2,…, P_{LEVEL}が生成される*/
do lev = 1 *to* (*LEVEL* - 1)
　clev = *lev* + 1
　/*フィルタリング*/
　$\tilde{A}_{lev} = Filter(A_{lev})$　　・・・(1)
　/*アグリゲート生成*/
　$\tilde{P}_{clev} = aggregation(\tilde{A}_{lev})$　　・・・(2)
　/*アグリゲートの重み付け*/
　$P_{clev} = smooth(\tilde{P}_{clev})$　　・・・(3)
　/*次レベルの問題生成*/
　$A_{clev} = R_{clev}A_{lev}P_{clev}$　　・・・(4)
end do

図 2　SA-AMG 法の構築部のアルゴリズム

以上の(1)から(4)の手続きを繰り返し行うことで，図 3 のように複数レベルの小規模な問題を作り階層構造を生成する．本研究は構築部で，手続きの(2)と(3)の間に粗いレベルのプロセス領域を適宜集約していく手続きを加えており，これについては 3 章で記述する．

2.2.2 アグリゲート生成手法

領域分割されたグラフ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 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．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，解法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[8][9].

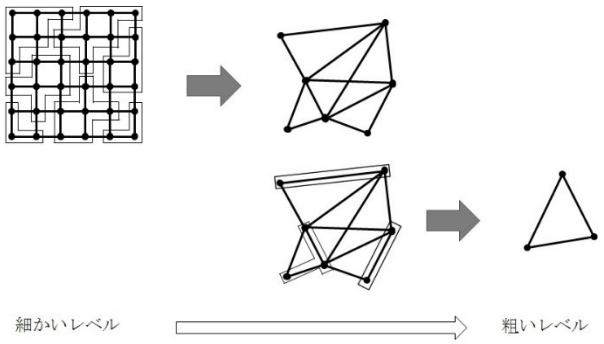

図 3　階層構造の生成

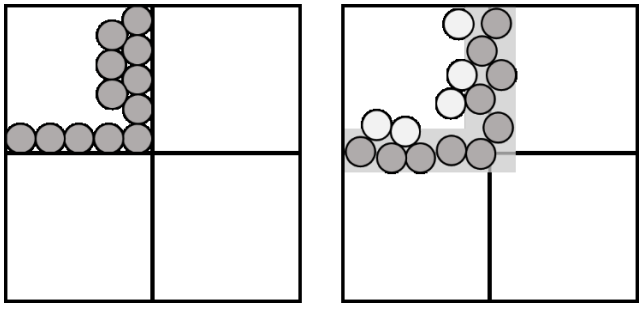

図 4　独立アグリゲート手法と共有アグリゲート手法

2.2.3 AMG 法における領域分割法による並列化手法

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 5 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．

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，アグリゲートは跨が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 6 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．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．

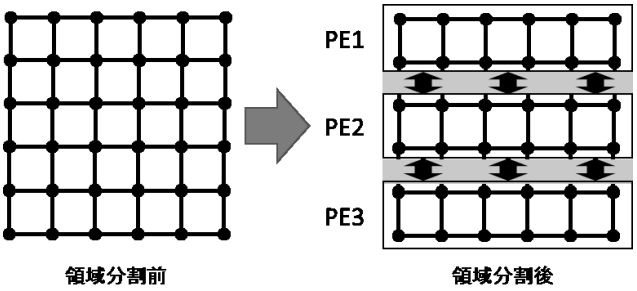

図 5　2 次元格子問題と領域分割による通信

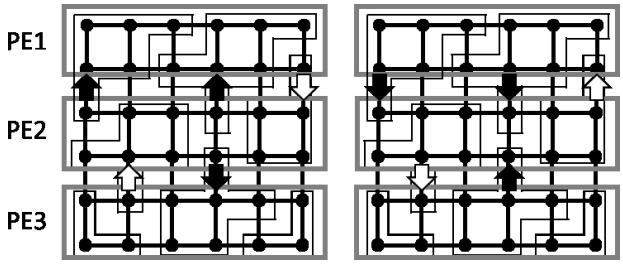

図 6　Restriction（左）と Prolongation（右）の通信

3. 領域集約手法

3.1 領域集約手法の概要

3 章では，独立アグリゲート生成手法を用いた SA-AMG 法において，粗いレベルのプロセス領域を集約していく手法について述べる．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情報の通信は行われない．しかし，領域集約手法を用いる場合には，プロセス領域が狭く，領域境界がアグリゲート作成をさまたげることがない．そのため，独立アグリゲート生成手法の SA-AMG 法に領域集約手法を適用したものも，共有アグリゲート生成手法と同様に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と考えられる．

図 7 に領域集約手法の概要を示す．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．

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一定数を保つことができるように，アグリゲートの総数，隣接プロセスの情報を使用して，プロセス間のデータ分散状況を示す重み付きグラフを作成し，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．

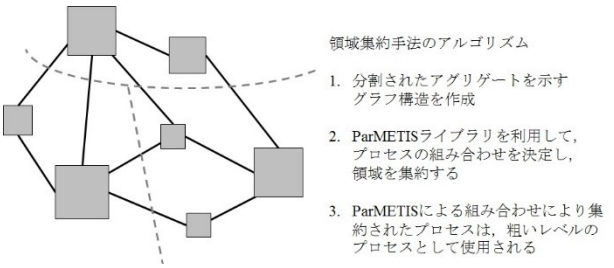

図 7　領域集約手法の概要

3.2 領域集約手法のアルゴリズム

図 8 に，図 2 で示した 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．

(3)　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．図 9 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）．並列数の決定方法は，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．

/*構築部*/
Input: A_1, x_1, b_1
/*A_1からA_2,…, A_{LEVEL}と*/
/*P_1からP_2,…, P_{LEVEL}が生成される*/
do lev = 1 *to* (*LEVEL* - 1)
　clev = *lev* + 1
　/*フィルタリング*/
　$\tilde{A}_{lev} = Filter(A_{lev})$　・・・(1)
　/*アグリゲート生成*/
　$\tilde{P}_{clev} = aggregation(\tilde{A}_{lev})$　・・・(2)
　/*領域集約手法のルーチン*/
　/*粗いレベルの並列数の決定*/
　call decide_parallelism()　・・・(3)
　/*領域の組み合わせの決定*/
　call decide_process_combination()　・・・(4)
　/*アグリゲートテーブル等の調節*/
　call set_GIN_aggre_and_aggre_tbl()　・・・(5)
　/*アグリゲートの重み付け*/
　$P_{clev} = smooth(\tilde{P}_{clev})$　・・・(6)
　/*次レベルの問題生成*/
　$A_{clev} = R_{clev}A_{lev}P_{clev}$　・・・(7)
end do

図 8　領域集約手法を加えた SA-AMG 法のアルゴリズム

```
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

図 9　粗いレベルの並列度決定方法

(4)　領域の組み合わせの決定

(3)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付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，アグリゲート番号はランク番号順に順次割り当てる．

(5)　アグリゲート番号の調節

(4)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．

4. メッセージ数モデルによる性能予測

領域集約手法を用いた SA-AMG 法では,領域集約の閾値を変更することにより性能が変化する．そこで，最適な領域集約のパラメタを推定するために，通信を行うメッセージ数のモデルを用い性能予測を行う．領域集約では，メッセージ数が減らせるメリットと，粗いレベルの並列度が下がるデメリットがある．粗いレベルでは，問題サイズが小さくメッセージ数を減らせる効果のほうが大きいと考えられる．そのため，性能への影響が大きいと考え，集約パラメタ，問題行列のサイズ，平均隣接プロセス数を入力すると，V-cycle の 1 回あたりの通信メッセージ数を予測するモデルを作成した．アグリゲート生成手法には，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,そして CGA を用いた独立アグリゲート生成手法の 3 つがある．共有アグリゲート生成手法に関しては，未知数個数が各プロセスで異なってしまうため，モデル化が難しい．そのため，今回はモデル化を行っていない．今回モデル化を行う手法は以下の 4 つである．以下の 4 つの手法は，独立アグリゲート生成手法においての，領域集約について実装を変えた手法となっている．

1. 集約をしない手法
2. 最も粗いレベルのみ 1 プロセスに集約する手法
3. 集約の際にレベル間通信を 1 度の通信で行う手法（現状）
4. 集約の際の通信を 2 回に分けて行う手法（独立アグリゲート最適化）

以上の 4 つの手法を比較する．1・2 の手法は従来手法となっており，3 は我々が研究している手法となっている．また，4 については有用な手法であると予想されるが，今回は実装を行っていない手法である．評価対象は，解法部の V-cycle を 1 回適用したときの通信メッセージ数である．

V-cycle は各レベルで緩和法を適用するが，ヤコビ法 1 回のような形を想定し，疎行列ベクトル積 1 回と想定する．もし領域集約を行わない場合，最下層は 1 つ上のレベルと同程度の並列性が残っているため，最下層では 30 回程度反復解法を回すことを想定した．

単純化するため，粗いレベルの行列の性質については，以下のように想定する．

- 1 レベル粗くなるたびに未知数の個数は R 倍されて減っていく．（R≈1/30）
- 1 レベル粗くなるたびに隣接プロセスの個数はα倍で増えていく．（α ≈ 1.5）

また，メッセージ数モデルに必要なパラメタを以下のように定義する．

- B：隣接しているプロセス数の平均値
- L：最も粗いレベルのレベル数
- P：全体プロセス数
- P_k：レベル $k+1$ での未知数をもつプロセス数
- N：問題の未知数の個数
- C：領域集約の閾値のパラメタ
- X_k：レベル $k+1$ にする際に集約するプロセス領域数

R, α, B, L, P, N, Cのパラメタはユーザー側が入力する値であり，P_k，X_kは計算に使用する値である．また，X_k はレベルを $k+1$ にする際に 1 プロセスに集約するプロセス領域数であり，P_kはレベル $k+1$ での未知数をもつプロセス数であるため，それぞれ以下のように計算できる．

$$X_k = min(P_{k-1}, \lceil P_{k-1}C/(R^kN) \rceil),\ P_k = P_{k-1}/X_k$$

X_kは十分に未知数の個数があり，集約されないときは 1 となる．R^kNはレベル k のときの未知数の個数を表している．そのため，$(R^kN)/P_{k-1}$を計算することにより，1 プロセスあたりのアグリゲート数が求められ，その逆数をCに

表 1　レベル数 3 の V-cycle での各レベルの通信回数

レベル	集約なし	最下レベルのみ集約	集約あり（現状）	集約あり（独立アグリゲート最適化）
1	B	B	B	B
1-2 間	B	B	$min(X_1^{2/3}B + X_1 - 1, P_0)$	$min(B + X_1 - 1, P_0)$
2	$min(\alpha B, P)$	$min(\alpha B, P)$	$min(\alpha B, P_1)$	$min(\alpha B, P_1)$
2-3 間	$min(\alpha B, P)$	P	$\min(X_2^{2/3} min(\alpha B, P_1) + X_2 - 1, P_1)$	$min(\alpha B, P_1) + X_2 - 1$
3	$30\times \alpha^2 B$	-	-	-
3-2 間	$min(\alpha B, P)$	P	$\min(X_2^{2/3} min(\alpha B, P_1) + X_2 - 1, P_1)$	$min(\alpha B, P_1) + X_2 - 1$
2	$min(\alpha B, P)$	$min(\alpha B, P)$	$min(\alpha B, P_1)$	$min(\alpha B, P_1)$
2-1 間	B	B	$min(X_1^{2/3}B + X_1 - 1, P_0)$	$min(B + X_1 - 1, P_0)$
1	B	B	B	B

かけることにより，集約すべき領域数が求まる．P_{k-1}との最小値をとっているが，これは集約する領域数上限が上のレベルでのプロセス数のためである．また X_kが 1 以上の整数となるように切り上げを行っている．X_kは集約するプロセス数のため，次のプロセス数は$P_k = P_{k-1}/X_k$で求められる．P_kの計算の際に値が小数となる場合があるが，最終的にP_kが 1 となる．また，このメッセージ数モデルでは，緩和法が 1 回の隣接通信で行えるような場合を想定している．

表 1 にレベル数が 3 のときの各レベル，各レベル間での隣接通信のメッセージ数を示す．

現状では，粗いレベルへの移動の際に，1 度の通信で 1 プロセスへ集約される．1 プロセスに集約されるデータは，X プロセスとそれに隣接するプロセスが保持している．X プロセスの隣接プロセス数は 3 次元の問題を想定し，1 プロセスの隣接プロセス数と比較して$X^{2/3}$倍に増えるとした．また X -1 プロセスからも送られてくるため，その合計（$X_1^{2/3}B + X_1 - 1$（1-2 間））を出している．

一方，集約あり（独立アグリゲート最適化）では，集約の際の通信を 2 回に分けて行う．この手法では，粗いレベルに移動する際に，細かいレベルで一度通信をした後でプロセス集約を行う．このようにすることで，$B + X_1 - 1$（1-2 間）で計算できることになる．

図 10 に，モデル上で C を 50 から 5000 まで変化させたときのメッセージ数の推移を示す．今回のモデルでは，全体のプロセス数 P を 1000，問題サイズ N を 27000000（300×300×300），隣接プロセス数の倍率α を 1.5，粗いレベルになるときの未知数の縮小率 Rを1/20，最初の隣接プロセス数 B を 27 とした．また，領域集約をしないときは，独立アグリゲートでは一定以上粗くできなくなるため，レベル数が 4 のときに最下層となるとした．

図 10 より，モデル上では領域集約を行うことで，メッセージ数が大きく抑えられることがわかる．また独立アグリゲート生成手法による集約の最適化を行うことで，さらにメッセージ数が削減でき，性能が向上するものと予想できるが，実装に関しては今後の課題である．

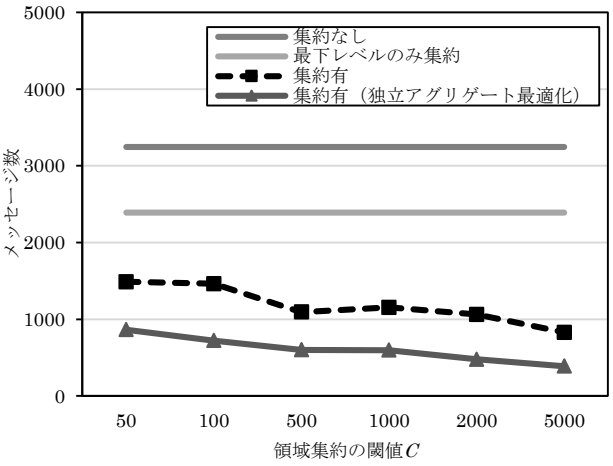

図 10　メッセージ数モデル

次章より数値実験と評価の章に入るが，メッセージ数モデルと実測データとの比較を行い，メッセージ数モデルについての評価を行う．その際，メッセージ数モデルを実際の環境に合わせるために，モデルのパラメタを変更している．また集約有では，領域集約を行うため，隣接プロセス数の増加の割合があまり大きくならない．そのため，パラメタαは集約有りとなしで，異なる値を設定した．

5. 数値実験と評価

5.1 実験環境

本研究では，東京大学の FX10 スーパーコンピュータシステム（Oakleaf-FX）[10]を使用し，数値実験を行った．FX10 の構成を表 2 に示す．FX10 では 6 次元メッシュ/トーラス構成を採用しており，隣接通信を用いた通信パターンの最適化を容易にするために，ユーザー側が計算ノードの 3 次元メッシュ/トーラスネットワークのサイズを指定するこ

とができる．そこで本研究では，表 3 のようにトポロジーのサイズを指定した．

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．本研究での 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．また Hybrid 並列の領域間通信は，MPI を用いて通信を行い，領域内では OpenMP を用いて並列実行を行っている．

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[11]を対象とした．問題サイズは 300×300×300 の問題として計測を行った．階層構造の生成には SA-AMG 法を用いた．領域分割法を用いているため，MPI のプロセス番号ごとに次元数の異なる複数の行列をもつ．SA-AMG 法は一定の節点数以下になるまで階層構造を生成する．また，反復の終了条件は相対残差が 1.0×10^{-7} とした．

問題行列の各プロセスへの分割には ParMETIS[7]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．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．

数値実験では AMGS ライブラリ[12]を使用している．AMGS ライブラリは，大規模な疎行列係数の線型方程式を AMG 法で解くライブラリである．解法部では BiCGSTAB 法[13]を使用し，前処理として AMG 法を適用している．AMG 法における解法部の緩和法には，マルチカラー・対称ガウス・ザイデル法を適用する．そして，解法部の V-cycle の各レベルで緩和法を 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．本実験では，高並列時に粗いレベルの影響が大きくなるストロング・スケーリングにて，本実装の効率や有用性について評価をする．

表 2　FX10 の構成

ノード	CPU 数	1
	メモリ	32GB
CPU	SPARC64™IXfx	
	コア数	16 Core / CPU
	動作周波数	1.848GHz
コンパイラ	富士通社製コンパイラ	

表 3　トポロジーのサイズ指定

ノード数	トポロジーサイズ（X×Y×Z）
192	4×6×8
384	8×6×8
768	8×12×8
1440	10×12×12

5.2 数値実験と評価

本研究では 2 つの実験を行った．実験 1 として，領域集約の閾値を変化させることによる，実行時間の変化を分析する実験を行った. 実験 1 では, 領域集約の閾値を 50, 100, 500，1000，2000，5000 として実験を行った．実験 2 として，共有アグリゲート生成手法と，領域集約手法を用いた独立アグリゲート生成手法を比較し，領域集約手法の有用性を検証する実験を行った．実験 2 での領域集約手法を用いた手法では，実験 1 で得られた最適な領域集約の閾値を用いて計測を行った．

実験 1 の結果を図 11 に示す．グラフの横軸は使用ノード数となっており，縦軸は実行時間となっている．縦軸は実行時間となっているため，下にあるほど性能が良いということを表している. 図 11 より, 768 ノードを用いた時に，領域集約の閾値を 1000 とすると最速となることがわかる．

ここで，メッセージ数モデルとの比較を行う．図 12 にメッセージ数モデルの図を，図 13 に実際に計測されたメッセージ数を，図 14 に 1 反復あたりの実行時間を示す．図 12・図 13・図 14 では，図 11 で得られた，全体的に最も性能が良くなったプロセス数である 768 プロセスで比較を行っている．また図 12 と図 13 の横軸は，領域集約の閾値の値である C の値であり，縦軸はメッセージ数となっているため，下にあるほどメッセージ数が少ないということを示している. 図 12 のモデルは, V-cycle が 1 回まわった時の，MPI メッセージ数をモデル化している．また，本実験では対称ガウス・ザイデル法を各レベルで 2 回適用しているため，各レベルでの通信回数の値を 4 倍している．図 13 は，実際に計測された V-cycle が 1 回まわった時のメッセージ数のグラフである．集約有と最下レベルのみ集約に関しては，最下層で LU 分解を用いているが，集約なしでは，最下層で緩和法を 30 回適用している．図 14 のグラフでは，横軸が C の値であり，縦軸が実行時間となっており，下にあるほど実行時間が少なく，性能が良いということを示している．また図 14 の実測時間では，メッセージ数モデルとの比較を行うために，AMG-BiCGstab 法 1 反復の時間をグラフの数値として用いている．図 12 と図 13 より，メッセージ数モデルと，実際に計測されたメッセージ数を見てみると，集約を行うことによりメッセージ数が抑えられることが双方のグラフで分かる．また，C の値を 5000 とした時にメッセージ数が最も少なくなることも双方のグラフから言え，メッセージ数モデルのグラフでの予測はおおむね正

しいといえる．ここで，図 12 及び図 13 と図 14 での比較を行う．図 12 と図 13 で見てみると，集約なしが最も性能が悪くなると予想できる．しかし，図 14 では，最下レベルのみ集約が最も性能が悪くなっている．これは，最下レベルのみ集約では，最下レベルにおいての LU 分解の時間が，大きくかかってしまったためと考えられる．最下レベルのみ集約では，最下レベルで集約するまでは，要素数は並列度以下にならず，集約した後で要素数が十分に小さくならなかったため，時間がかかってしまったと考えられる．図 14 より，AMG-BiCGstab 法 1 反復で実行時間を見ると，C を 1000 とした時に最も性能が良いことがわかる．また図 13 を見てみると，C を 1000 としたとき以降，メッセージ数があまり変動していないことがわかる．これより，C を 1000 以降としたときには，並列性が下がることの影響により，性能が悪くなってしまったと考えられる．

次に実験 2 の結果を図 15 に示す．図 15 のグラフでは横軸が使用ノード数，縦軸が実行時間となっており，下にあるほど性能が良いということを表している．またグラフ内の要素の Coupled は，共有アグリゲート生成手法を表し，CGA+Independent は領域集約手法を用いた独立アグリゲート生成手法を表している．実験 2 での領域集約の閾値は，実験 1 で得られた最適な値である 1000 を使用した．図 15 より，どの並列度を用いても，CGA+Independent は Coupled よりも高速であることがわかる．これより，領域集約手法を用いた独立アグリゲート生成手法の有用性が確認された．

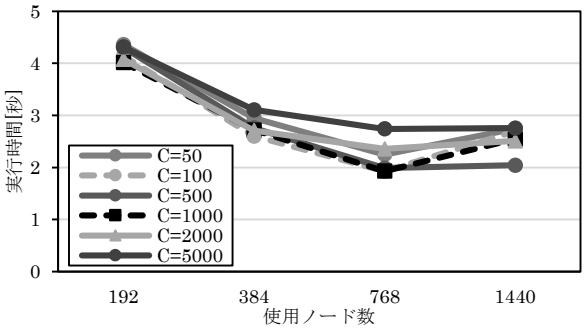

図 11　領域集約の閾値の変化による実行時間の推移

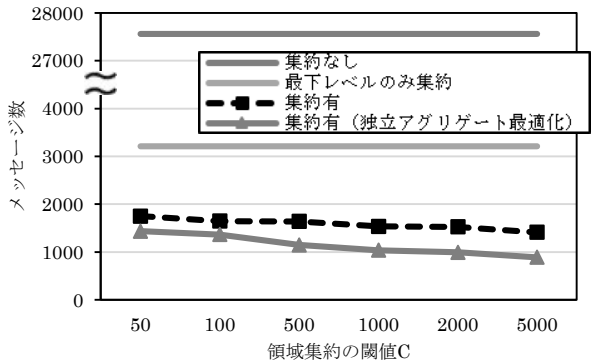

図 12　メッセージ数モデル（768 ノード）

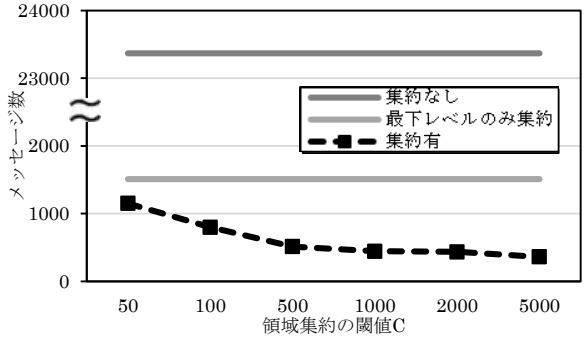

図 13　実測メッセージ数（768 ノード）

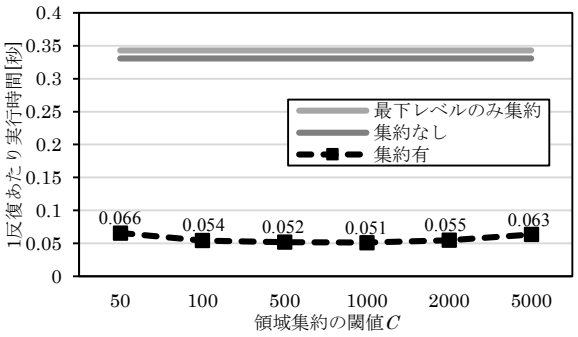

図 14　AMG-BiCGstab 法 1 反復の実行時間（768 ノード）

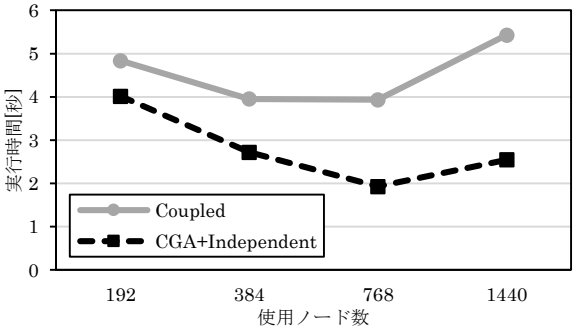

図 15　各手法の実行時間の比較

6. おわりに

本研究では，独立アグリゲート生成手法を用いた SA-AMG 法において，高並列時に問題となる粗いレベルの計算を，粗いレベルの領域集約により並列度を適宜集約していく手法について評価を行った．この評価には，2 つの実験を行った．1 つめは，領域集約手法の閾値のパラメタを変更することによる実行時間への影響の分析である．2 つめは，領域集約を用いない手法と領域集約を用いた手法を比較する実験である．

実験 1 では，領域集約の閾値を 6 つ（閾値：50，100，500，1000，2000，5000）用意し，閾値を変化させることによる性能評価を行った．この実験では，使用ノード数を 768，領域集約の閾値を 1000 としたときに最も良い性能が出ることがわかった．また，この実験ではメッセージ数モデル

を用い，実測した 4 つの手法との比較も行った．この実験より，モデル上で予測されたように，実際の計算でも領域集約を行うことにより，メッセージ数が抑えられることがわかり，メッセージ数の多い CGA なしの独立アグリゲート生成手法は実行時間が遅くなることを確認した．次に実験 2 では，共有アグリゲート生成手法と，領域集約手法を用いた独立アグリゲート生成手法を比較した. この実験で，問題により性能に違いがあるものの，領域集約手法用いた独立アグリゲート生成手法が最速となることがわかった．

今後の課題として，領域集約においての独立アグリゲート生成手法を前提とした，通信の実装の最適化があげられる．今回の実験では，集約の際のレベル間通信を 1 度の通信で行っているが，この方法ではメッセージ数が大きくなり, 性能が出なくなってしまうことが考えられる. そこで，集約の際の通信を 2 回に分けて行うことにより，メッセージ数が減り，性能が出るものと予想される．また，性能モデルの改良も必要である．性能モデルは，最適な領域集約の閾値を推定するために必要であるが，今回の推定はメッセージ数のみ着目したモデルであり，集約の結果，並列度が下がる影響を考慮できていない．そこで，性能モデルの改良を行うことで，最適な領域集約手法が推定できるものと考えられる．

謝辞 本研究の一部は，JSPS 科研費 15K15998 と JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」の助成を受けたものです.

参考文献

1) F. H. Pereira, S. L. L. Verardi, and S. I. Nabeta, A fast algebraic multigrid preconditioned conjugate gradient solver, Applied Mathematics and Computation 179, pp.344-351 (2006).
2) Vanek, P., Brezina, M. and Mandel, J., Convergence of Algebraic Multigrid Based on Smoothed Aggregation, Computing, Vol.56, pp.179-196 (1998).
3) Vanek, P., Mandel, J. and Brezina, M., Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems, Technical Report UCD-CCM-036 (1995).
4) Chan, T. F. and Vanek, P., Multilevel algebraic Elliptic Solvers, UCLA Math, Dept. CAM Report (1999).
5) 藤井昭宏，西田晃，小柳義夫，領域分割による並列 AMG 法アルゴリズム，コンピューティングシステム，Vol.44，No.SIG6 (ACS1)，pp.9-17 (2003).
6) Akihiro Fujii, Takuya Nomura, Teruo Tanaka and Osni Marques, AMGS: Algebraic Multigrid Solver with Coarse Grid Aggregation, Annual Meeting on Advanced Computing System and Infrastructure (ACSI2015).
7) ParMETIS, http://glaros.dtc.umn.edu/gkhome/metis/parmetis/download.
8) 藤井昭宏，小柳義夫，科学技術シミュレーションにて多用される代数的多重格子法の評価，シミュレーション　第 28 巻第 4 号，pp.9-14 (2009).
9) R. Tuminaro and C. Tong, Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines, In J. Donnelley, editor, Supercomputing (2000).
10) 東京大学情報基盤センタースーパーコンピューティング部門，FX10 スーパーコンピュータシステム (Oakleaf-fx), http://www.cc.u-tokyo.ac.jp/system/fx10/.
11) Deutsch, C.V., Journel, A.G., GSLIB Geostatistical Software Library and User's Guide, Second Edition, Oxford University Press (1998).
12) AMGS ライブラリ, http://hpcl.info.kogakuin.ac.jp/lab/software/amgs.
13) H. A. van der Vorst, Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems, SIAM Journal on Scientific and Statistical Computing, vol.13, no.2, pp.631-644 (1992).