Vol.2013-HPC-139 No.17
2013/5/30

# Smoothed Aggregation に基づく AMG 法における 分散アグリゲートの集約による通信の最適化

野村卓矢[†1]　藤井昭宏[†1]　田中輝雄[†1]

Smoothed Aggregation に基づく Algebraic Multigrid（SA-AMG）法を分散メモリ型でプロセス並列をするために，問題行列を領域分割法により各プロセスに分割している．このとき，プロセス数を増加させると，粗いレベルでは自プロセスの担当節点数が減り，通信を必要とする隣接プロセスの外部節点数が増える．これは粗いレベルにおいて，演算時間に対する相対的な通信時間の増加をもたらし，より高並列な環境では，さらにこの比率が広がると考えられる．そこで，本研究は問題行列生成部において，複数のプロセスに分散しているアグリゲートを粗いレベルでひとつのプロセスに集約し，通信時間をなくす手法を提案する．そして，数値実験を行い，CPU クラスタ環境で最大 100 万次元の 3 次元拡散方程式を解き，分析を行う．また，処理時間が分散アグリゲートを集約しない従来手法と比べ最大 71% 削減されたことを確認した．

## 1. はじめに

計算機の数値計算の応用において，大規模な連立 1 次方程式$Ax = b$はさまざまな場面で現れ，高速に解くことが求められている．大規模な連立 1 次方程式を解く解法のひとつに代数的マルチグリッド法（以下，AMG 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 法のなかで有効な解法のひとつに，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．

今後の計算環境は，マルチコアやメニーコアなどの高い並列性が求められる環境へ移行していくため，高い並列環境においても高い計算効率を実現する必要がある．本研究は，SA-AMG 法を分散メモリ型でプロセス並列する[5]．そして，問題行列を領域分割法により，各プロセスに分割している．このとき，プロセス数を増加させると，粗いレベルでは自プロセスの担当節点数が減り，通信を必要とする隣接プロセスの外部節点数が増える．これは粗いレベルにおいて，演算時間に対する相対的な通信時間の増加をもたらし，より高並列な環境では，さらにこの比率が広がると考えられる[5]．

SA-AMG 法は，問題行列生成部（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．

[6]の論文では，本稿で対象とする SA-AMG 法をシミュレーションへ適用している．このとき，1 プロセッサあたり 500 個の未知数を維持するように再分割している．しかし，問題の再分割は，計算コストがかかり，かつ，レベル間の通信テーブルなどの整合性をとるために冗長な計算と通信が発生する．そこで，本研究は，構築部においてアグリゲートが生成された段階での再配置を考えた．そして，粗いレベルでアグリゲートをひとつのプロセスに集約し，通信時間をなくす手法を提案する．数値実験では，AMGS ライブラリ[7]を使用し，FX10 スーパーコンピュータシステム（東京大学）[8]上で最大 100 万次元の 3 次元拡散方程式を解く．そして，従来手法と提案手法を比較し，処理時間の構成を分析する．

## 2. AMG 法

### 2.1 AMG 法概要

AMG 法は大規模な非構造格子の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，階層的に生成された行列を作る．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．

解法部（V-cycle）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．AMG 法は階層型で，最上層に与えられた問題行列が設置され，階層が下がるに連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誤差を計算し，横長の Restriction 行列と行列ベクトル積を行うことで

†1 工学院大学
kogakuin University

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.

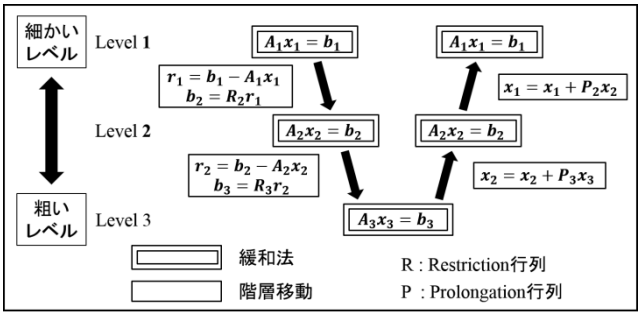


図 1　解法部（V-cycle）

補間演算子を決めることで階層構造の生成や各レベル間の移動が実現される．この補間演算子の生成手法により，さまざまな AMG 法の解法が存在する[9]．本研究は 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 Smoothed Aggregation に基づく 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．アグリゲート生成のルールは，任意の節点がどこかひとつのアグリゲートに属することである．

領域分割による並列 SA-AMG 法では, 分割された各領域において並列にアグリゲートを生成することになる．並列にアグリゲートを生成する手法には 2 つある．ひとつ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独立アグリゲート生成手法，もうひとつはアグリゲートが領域境界を跨いで生成される共有アグリゲート生成手法である．本研究では共有アグリゲート生成手法を使用している．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，その後各領域内でアグリゲートを生成する．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[10].

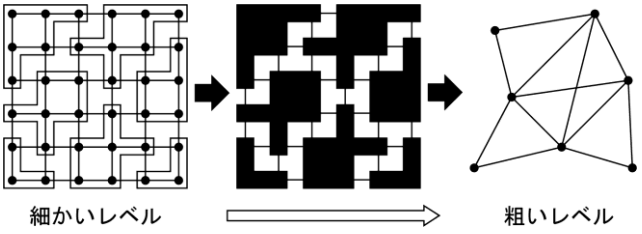


図 2　アグリゲートの生成

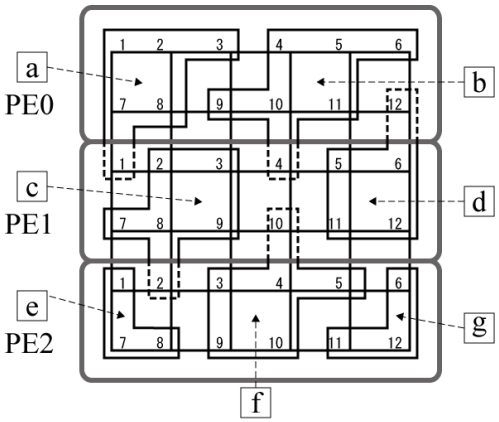


図 3　分散アグリゲートの例

本研究は，節点間の接続関係を考慮して領域分割を行なっており，それぞれのアグリゲートは各領域に分散している．また，領域境界を跨ぐアグリゲートは，どちらかの領域に属する．各プロセスに分散しているアグリゲートには種類が 2 つあり，内部アグリゲートと外部アグリゲートがある．内部アグリゲートとは自領域が担当するアグリゲートである．外部アグリゲートとは他領域が担当するアグリゲートであり，自領域へ領域を跨ぐアグリゲートである．

図 3 は，分散アグリゲートの一例である．分散アグリゲートとは，図 3 のように複数のプロセスに分散しているアグリゲートを指す．それぞれのアグリゲートには a から f の名称が付けられている．また，2 次元格子上の数字は各節点の番号であり，領域分割されている領域ごとに 1 から番号が振り分けられる．例として図 3 の PE0 について説明する．なお，PE（Processing Element）は，プロセスを指す．PE0 の領域上のアグリゲート（a，b）は内部アグリゲートであり，アグリゲート（d）は外部アグリゲートである．

### 2.3 構築部

図 4 は SA-AMG 法の構築部のアルゴリズムを示す. 構築部では，主に図 4 の（1）から（4）の手続きを行う．図 4 において，*LEVEL* は最大レベル数，*lev* は現階層のレベル数，*clev* は現階層の次のレベル数をあらわす．そして，構築部の（1）から（4）の手続きを説明する．

/*構築部*/
/*Input : $A_1$, $x_1$, $b_1$*/
/*$A_1$から$A_2, \ldots,$ $A_{LEVEL}$と*/
/*$P_1$から$P_2, \ldots,$ $P_{LEVEL}$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)
/*次レベルの問題行列生成*/
$A_{clev}\ =\ R_{clev}A_{lev}P_{clev}$ ・・・(4)
$end\ do$

図 4　SA-AMG 法の構築部のアルゴリズム

(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 0.05 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，フィルタリングした後の行列$\tilde{A}_{lev}$から補間行列$\tilde{P}_{clev}$を生成する．$\tilde{P}_{clev}$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を示す．

(3)　アグリゲートの重みづけ
アグリゲートの節点に適切に重みづけを行う．行列$\tilde{P}_{clev}$をそのまま補間演算子として使ってもよいが，収束性を高めるために緩和法を 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る場合が多い．

(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．

以上の（1）から（4）の手続きを繰り返し行うことで，図 5 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）の間に各プロセスに分散しているアグリゲートを集約する手続きを加えている．

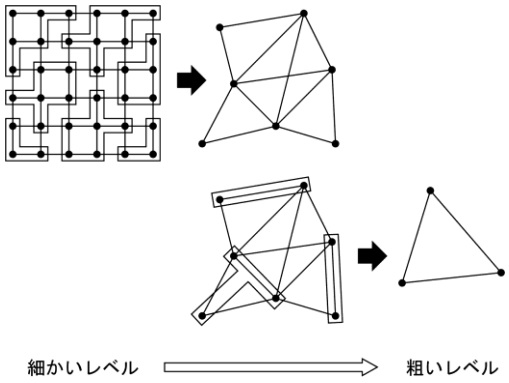


図 5　階層構造の生成

### 2.4 領域分割法

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 6 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 テーブルをもたせている．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 7 は Restriction と Prolongation の通信例を示す．最も上の領域が Restriction を行う際，アグリゲートの一部の節点が中間の領域にある．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，他領域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，他領域へ必要となる節点を分散させる．

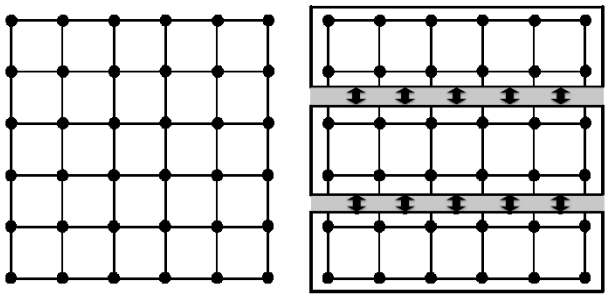

図 6　2 次元格子構造と領域分割による通信

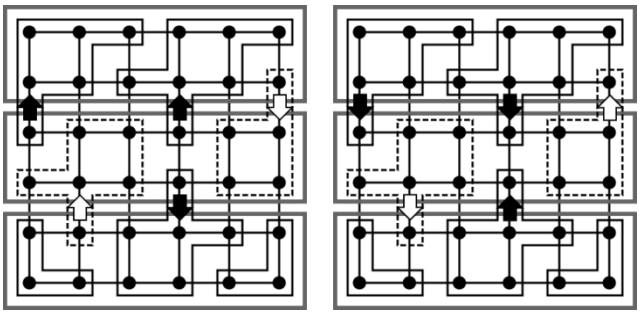

図 7　Restriction と Prolongation の通信

## 3. 分散アグリゲートのデータ構造

表 1　分散アグリゲートのデータ構造

| | |
|---|---|
| aggregate_size | アグリゲート数 |
| aggregate（：） | 各アグリゲートがもつ節点番号 |
| neibPE_size | 隣接 PE 数 |
| neibPE（：） | 隣接 PE 番号 |
| aggregate_table_size（:） | 各隣接 PE の外部アグリゲート数 |
| aggregate_table（：，：） | 各隣接 PE の各外部アグリゲート情報 |

表 2　例）PE0 の分散アグリゲートのデータ値

| | |
|---|---|
| aggregate_size | 3 |
| aggregate（1） | 1，2，3，7，8 *1 |
| aggregate（2） | 4，5，6，9，10，11 |
| aggregate（3） | 12 |
| neibPE_size | 1 |
| neibPE（1） | 1 |
| aggregate_table_size（1） | 1 |
| aggregate_table（1*2，1） | 3*2，3*1 |

*1 ローカルアグリゲート番号
*2 グローバルアグリゲート番号

2.2 節の図 3 を用いて説明をする．図 3 の例では問題を 3 つの領域にブロック行分割している．問題の節点はすべてで 36 個あり，1PE あたり 12 個の節点をもつ．そのため，PE ごとに 1 から 12 まで順に節点に番号が振り分けられる．表 1 は，それぞれの PE がもつ分散アグリゲートのデータ構造の説明を示す．分散アグリゲートはそれぞれに番号が振られており，2 種類の番号がある．ひとつはローカルアグリゲート番号で, PE の領域上のアグリゲートについて独立に振り分けられる番号である．もうひとつはグローバルアグリゲート番号で，すべての PE 上のアグリゲートについて共有に振り分けられる番号である．表 2 は図 3 の PE0 の分散アグリゲートのデータ値の例を示し，以下に各データ値の説明を示す．

(1) アグリゲート数（aggregate_size）: 3
　PE0 の領域上に内部アグリゲートと外部アグリゲートが合計で 3 つ存在することを示す
(2) 各アグリゲートがもつ節点番号（aggregate（：））は表 2 のように値が記録されている．そして，PE0 上の 1 から 12 の各節点が，どのアグリゲートに属するかを示す．
(3) 隣接 PE 数（neibPE）: 1
　PE0 に隣接している PE がひとつあることを示す
(4) 隣接 PE 番号（neibPE（：））: {1}
　PE0 が PE1 と隣接することを示す
(5) 各隣接 PE の外部アグリゲート数（aggregate_table_size（：））: {1}
　隣接 PE1 から PE0 への外部アグリゲートがひとつあることを示す．また，PE0 上のアグリゲート数（aggregate_size）の値は 3 なので，内部アグリゲートが 2 つあることがわかる．
(6) 各隣接 PE の各外部アグリゲート情報（aggregate_table（：，：））: {3,4}
　隣接 PE1 から PE0 への外部アグリゲートがひとつ存在し, その外部アグリゲートのグローバルアグリゲート番号が 3 で, ローカルアグリゲート番号が 3 であることを示す

## 4. 分散アグリゲートの集約方法

本研究では，分散アグリゲートの集約方法について 2 つの案を考えた．従来手法は，図 8 のように最も細かいレベルで領域分割を行い, 下のレベルではそれに従って分割し，並列化する．

案 1 は，図 9 のように粗いレベルの問題生成後に各領域に分散している問題をひとつの領域に集約し，緩和法適用後に再度各領域に分散を行う．

案 2 は，アグリゲートの概念を拡張する．自領域に存在しないアグリゲートも内部アグリゲートとし，また，自領域へ領域境界を跨がないアグリゲートも外部アグリゲートとする．以上のように双方の概念を拡張することにより，アグリゲート生成時にデータ構造の値の変更を行う．それにより, 図 10 のように細かいレベルから直接粗いレベルへ問題をひとつの領域に集約することができるようになる．

案 1 は案 2 と比べ階層移動の時間以外に粗いレベルの問題をひとつの領域に集約し，再度各領域に分散する時間が生じる．そのため，案 2 より処理時間が多くなると考えられるので，本研究では案 2 を実装し，評価を行った．

本研究で行う分散アグリゲートのデータ構造の値の変更では，分散アグリゲートを集約しない領域において，各領域の内部アグリゲートを外部アグリゲートとし，分散アグリゲートを集約する領域の内部アグリゲートとして登録する．

緩和法の領域間通信において，案 2 は従来手法と比べ分散アグリゲートの集約を適用した階層の緩和法の領域間通信がなくなる．また，階層移動の領域間通信において，従来手法は，すべてのレベルで領域分割による並列化をしており，各領域独立に Restriction と Prolongation を行う．案 2 の Restriction では，分散アグリゲートを集約する領域に他領域のすべての節点を集約する通信をする必要がある．また，Prolongation では，分散アグリゲートを集約している領域から元々他領域に存在した節点を各隣接領域へ通信する必要がある．そのため，案 2 の Restriction と Prolongation の通信は，ひとつの領域とその他の多くの領域と通信が発生するので，従来手法に比べ通信時間は増加すると考えられる．

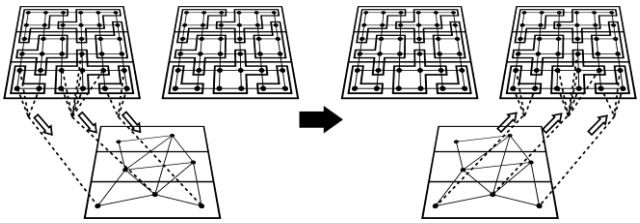

図 8　従来手法

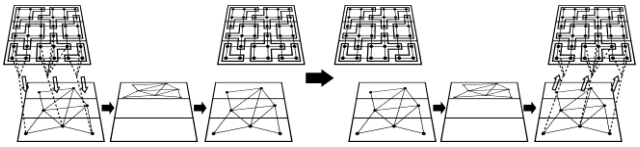

図 9　分散アグリゲートの集約（案 1）

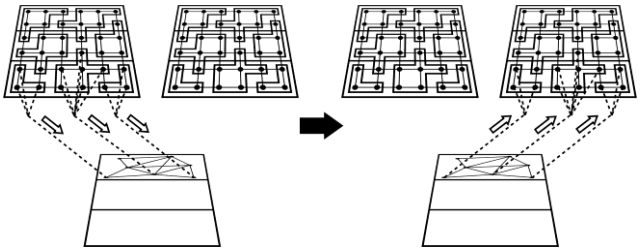

図 10　分散アグリゲートの集約（案 2）

表 3　実行環境

| | | |
|---|---|---|
| マシン名 | | FX10（Oakleaf-FX） |
| ノード | CPU チップ数 | 1CPU チップ |
| | メモリ容量 | 32GB |
| CPU | 名称 | SPARC64™ IXfx |
| | コア数 | 16 コア/CPU チップ |
| | 動作周波数 | 1.848GHz |
| 倍精度演算性能（理論性能） | 1 ノード性能 | 236.5GFLOPS |

## 5. 数値実験と考察

### 5.1 実行環境と計算対象

本研究は，FX10 スーパーコンピュータシステム（東京大学）を使用し，数値実験を行った．表 3 に FX10 の構成を示す. 数値実験は, 12 ノード, 192 コアを使用し, FlatMPI で実験を行った．ノード，コア間の領域間通信は MPI を用いて 192 プロセス立ち上げ，1 コアあたり 1 プロセスを担当している．

計算対象は，27 点参照の格子構造をもつ 3 次元拡散方程式の等方性の問題である．等方性とは，物体の物理的性質が方向によって異ならない性質である．問題サイズは立方体領域が $10^3$ から $100^3$ の 10 個を扱う．階層構造の生成には SA-AMG 法を用いた．領域分割法を用いるため，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ため，問題サイズごとに階層構造のレベル数は異なる．また，収束条件として 2 ノルム相対残差を用い，$1.0^{-7}$ を下回れば解けたものとした．

グラフ分割には，ParMetis[11]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なっている. ParMetis は，グラフ分割や疎行列のオーダリングのためのさまざま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形方程式を AMG 法で解くライブラリである. 解法部では, BiCGSTAB 法[12]を使用し，前処理として AMG 法を適用している．AMG 法では，各レベルでガウス・ザイデル法 1 回を各プロセスの領域内で行い，領域境界の節点についてもガウス・ザイデル法を行う．ただし，領域境界を跨ぐ節点においては，更新前の値を使用しているため収束性は低下する．節点数が 100 以下になったとき最も粗いレベルとし，対称ガウス・ザイデル法を 30 回反復させる．

表 4 は，従来手法と提案手法で分散アグリゲートを集約する適用階数を変化させた 3 パターンについての説明を示す．また，これら 4 つの手法について，解法部の収束時間の比較を行う．

表 4　比較対象

| | | |
|---|---|---|
| 従来手法 | no | 最も細かいレベルで領域分割を行い，下のレベルでは，それに従って分割する |
| 提案手法 | apply_1 | 最も粗いレベルにおいて，分散アグリゲートを集約する |
| | apply_2 | 最も粗いレベルのひとつ上のレベルから分散アグリゲートを集約する |
| | apply_3 | 最も粗いレベルの 2 つ上のレベルから分散アグリゲートを集約する |

### 5.2 数値実験

図 11 は各手法の解法部における収束時間の比較結果である．すべての問題サイズで apply_1 と apply_2 は，no と比べ，ともによい結果を示した．棒グラフ（no，apply_1，apply_2，apply_3）がすべて揃っている問題サイズ $30^3$ のとき，no に対し apply_1 は約 55%，apply_2 は約 58%，apply_3 は約 33%の削減を確認した．ある一定の大きさ以上の問題が設置されるレベルにおいて，提案手法を適用して 1 プロセスで実行すると，領域分割法を用いて並列実行するよりも計算時間が掛かる．そのため，提案手法を適用するレベル数には注意が必要である．

問題サイズが $10^3$ と $20^3$ のとき，apply_2 と apply_3 の棒グラフが不足している．これは，全体の節点数が 100 以下になったとき最も粗いレベルとしており，問題サイズごとに階層構造のレベル数は異なるからである．分散アグリゲートを集約するには，階層構造が最低 2 レベル必要となる．その理由は，アグリゲート生成のタイミングが，細かいレベルからひとつ下の粗いレベルの問題生成時であり，アグリゲートを生成した直後に分散アグリゲートを集約するからである．そのため，問題サイズが $10^3$ のときは，階層構造が 2 レベルなので，最も粗いレベルのみ提案手法を適用している．同様に，問題サイズが $20^3$ のときは，階層構造が 3 レベルなので，最も粗いレベルと最も粗いレベルのひとつ上のレベルから提案手法を適用している。

no は，通常であれば問題サイズが増えるにつれて，収束時間が増加すると考えられる．しかし，実験結果からは収束時間に周期が見られた．これは，最も粗いレベルの節点数が関係していると考えられる．数値実験ではプロセスを 192 個立ち上げており，最も粗いレベルでは全体の節点数が 100 以下となるため，多くのプロセスが待機状態となる．1 プロセスあたりの節点数が最低 1 個と仮定すると，節点数分の緩和法の領域間通信が行われる．また，最も粗いレベルでは，問題が収束すると考えられるまで緩和法を 30 回繰り返し適用するので，多くの領域間通信が発生する．そのため，最も粗いレベルの節点数が 100 に近い値の問題のほうが通信時間のオーバーヘッドが大きくなり，収束時間が増加したと考えられる．以上の理由により，図 11 の apply_3 は，問題サイズが $100^3$ のときに no よりも遅くなると考えられる．

図 12 は，問題サイズ $90^3$ について各手法の収束時間の構成を示す．また，表 5 で図 12 の収束時間の構成の説明を示す．このとき，各手法の V-cycle のレベル数は 5 レベルである．以下に収束時間の構成についての結果と考察を示す．

(1) Lv.5_smooth_comm

no の収束時間のうち，最も大きな割合を占めているのが，最も粗いレベルの緩和法の通信時間（Lv.5_smooth_comm）である．

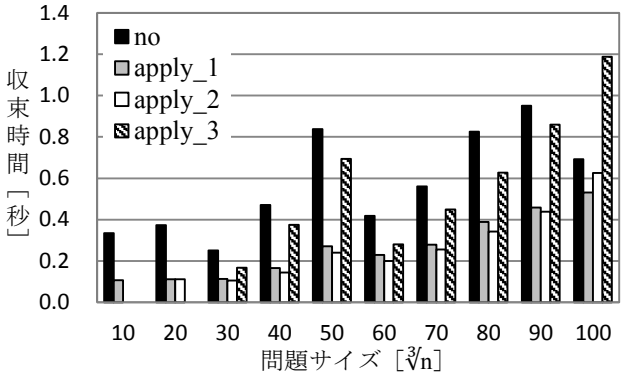


図 11　解法部の収束時間

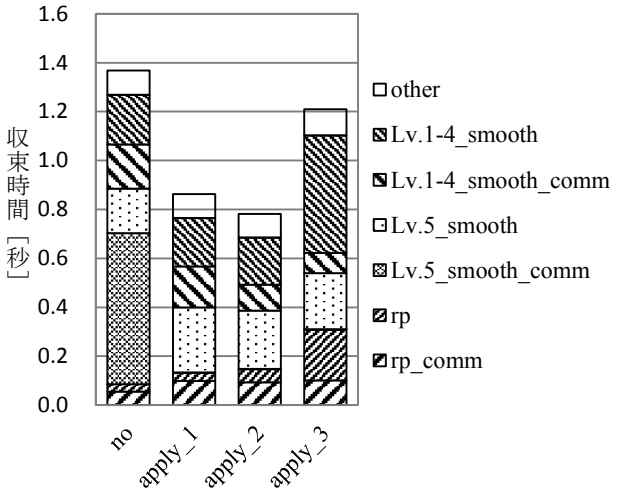


図 12　問題サイズ $90^3$ の収束時間の構成

表 5　図 12 の構成の説明

| | |
|---|---|
| other | 他の計算時間 |
| Lv.1-4_smooth | 最も粗いレベル以外の緩和法の計算時間 |
| Lv.1-4_smooth_comm | 最も粗いレベル以外の緩和法の通信時間 |
| Lv.5_smooth | 最も粗いレベルの緩和法の計算時間 |
| Lv.5_smooth_comm | 最も粗いレベルの緩和法の通信時間 |
| rp | 階層移動（Restriction，Prolongation）の計算時間 |
| rp_comm | 階層移動（Restriction，Prolongation）の通信時間 |

提案手法を適用することにより，行列が 1 つの領域に集約され，緩和法の計算を 1 プロセスで行えるので，通信部分がなくなる．

(2) Lv.5_smooth

提案手法を適用することにより，行列を 1 プロセスで処理するため，並列して実行している no よりも緩和法の計算時間は増加する．

(3) Lv.1-4_smooth_comm

提案手法の適用階数が増えるにつれて，行列が 1 つの領域に集約される階層が増える．そのため，緩和法の計算を 1 プロセスで行える階層が増えるので，緩和

法の通信時間は減少傾向となる．

(4) Lv.1-4_smooth
提案手法の適用階数が増えるにつれて，行列を 1 プロセスで計算する階層が増えるため，緩和法の計算時間は増加傾向となる．

(5) rp_comm
提案手法を適用することにより，階層移動の Restriction と Prolongation の通信では，ひとつの領域とその他多くの領域と通信するため，階層移動の通信時間は増加する．

(6) rp
提案手法の適用階数が増えるにつれて，階層移動の計算を 1 プロセスで行う階層が増えるため，階層移動の計算時間は増加傾向となる．

(7) other
V-cycle 中の階層移動（Restriciton）をするための残差計算と，AMG 法を前処理とした BiCGSTAB 法の V-cycle 以外の計算時間であり，提案手法を適用しても影響はあまりないので，計算時間はほぼ一定となる．

## 6. おわりに

本研究は，構築部においてアグリゲートが生成された段階での再配置を考えた．そして，粗いレベルでアグリゲートをひとつのプロセスに集約し，通信時間をなくす手法を提案し，実装した．階層構造すべてのレベルで並列化している従来手法（no）と最も粗いレベルのみ提案手法を適用した（apply_1），ひとつ上のレベルから提案手法を適用した（apply_2），2 つ上のレベルから提案手法を適用した（apply_3）の 3 パターンを比較し，評価を行った．数値実験では，12 ノード，192 コアを使用して FlatMPI で実験を行った．すべての問題サイズの中で，分散アグリゲートの集約を適用する階層構造のレベルは，ほとんどの問題サイズで apply_2 が有効であった．しかし，問題サイズによっては，apply_1 が有効であった．数値実験の結果より，apply_2 が，no と比べ，最大約 71%の収束時間の削減を確認した．また，問題サイズ $10^3$ から $100^3$ において平均で約 55%の収束時間の削減を確認した．

問題サイズ $90^3$ の収束時間は，逐次では 35.71 秒，FlatMPI による 192 並列では 0.95 秒であり，その比は約 38 倍であった．また，提案手法(apply_2)を並列化へ適用することにより，適用前と比べ約 2.2 倍となり，これは逐次と比べ約 81 倍であった．

今後の研究では，細かいレベルと粗いレベルの途中階層において，分散しているアグリゲートをひとつの領域に集約するのではなく，2 つ以上の領域に部分的に集約を行う．この処理により問題行列が小さくなるにつれて並列数を調節することにより各レベルの通信の最適化を行う．

**謝辞** 本研究の一部は，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」による．

## 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 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] Adams, M.~F., et al.: Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom, ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing (2004).
[7] AMGS ライブラリ：
http://hpcl.info.kogakuin.ac.jp/olab/software
[8] 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/
[9] 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).
[10] R. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines, In J. Donnelley, editor, SupperComputing (2000).
[11] ParMETIS：
http://glaros.dtc.umn.edu/gkhome/metis/parmetis/download
[12] 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).