Vol.2018-HPC-167 No.27
2018/12/18

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

野村直也[†1]　中島研吾[†1]　河合直聡[†2]　藤井昭宏[†3]

概要：メニィコアプロセッサを結合したクラスタでは，メッセージパッシング（MPI）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されている．一方，大規模連立一次方程式を高速に解く解法として SA-AMG 法がある．SA-AMG 法は，与えられた問題行列から粗い問題を再帰的に生成し，それらを用いて様々な波長成分を効率よく減衰させることで，高い収束性やスケーラビリティを実現している．本研究では，SA-AMG 法に Hybrid 並列を適用し，大規模クラスタ環境上で実行時間や 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，Oakforest-PACS（JCAHPC），京（理化学研究所），ITO（九州大学）スーパーコンピュータシステムを用いて計測を行っており，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．

キーワード：Multigrid 法，Hybrid 並列

1. はじめに

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，工学における設計現場，気象予報，渋滞予測などの身近な場面で幅広く活用されている．このような問題の多くは最終的には連立一次方程式$\boldsymbol{Ax} = \boldsymbol{b}$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．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言われている．

一方，大規模連立一次方程式を高速に解く手法として Multigrid 法が提案されている．Multigrid 法は係数行列Aから階層的に粗いレベルの行列を生成し，各波長成分を効率よく減衰させることによって，高い収束性を実現している．Multigrid 法には様々な派生形が存在しており，本研究ではその中の Algebraic Multigrid（AMG）法を対象としている[1]．AMG 法はさらに，階層行列の生成方法により様々な手法が存在しており，本研究では Smoothed Aggregation に基づく AMG 法（SA-AMG）を用いている[2][3][4]．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．

本研究では，SA-AMG 法に Hybrid 並列を適用し，大規模クラスタ環境上で実行時間や 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．これにより，SA-AMG 法の高い収束性を，高並列環境下においても実現できると期待できる．数値実験では，Oakforest-PACS（JCAHPC），京（理化学研究所），ITO（九州大学）スーパーコンピュータシステムを用いて計測を行った．上記の計算機は，それぞれ Intel® Xeon Phi™ 7250（68 コア/CPU），SPARC64™ VIIIfx（8 コア/CPU），Intel® Xeon® Gold 6154（18 コア/CPU）によるメニーコアクラスタ型の計算機であり，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．

2. Hybrid 並列

2.1 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

†1 東京大学
The University of Tokyo
†2 理化学研究所　計算科学研究センター
RIKEN Center for Computational Science
†3 工学院大学
Kogakuin University

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 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 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 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．

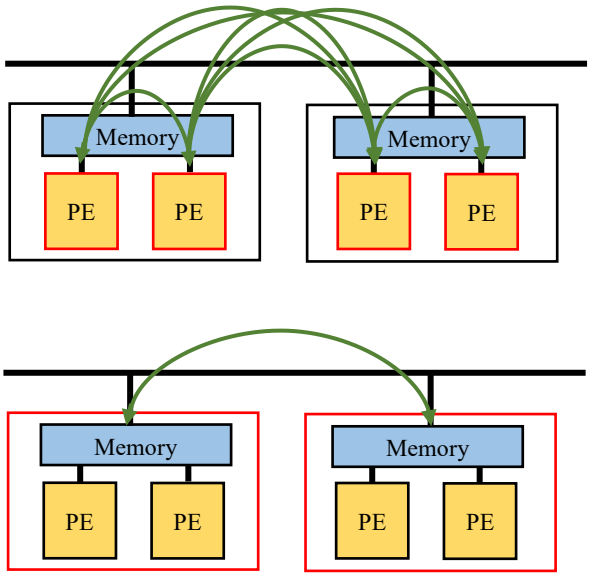


図 1　SMP クラスタ上における Flat MPI（上）と Hybrid 並列（下）の通信の例

2.2 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[5]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図 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．

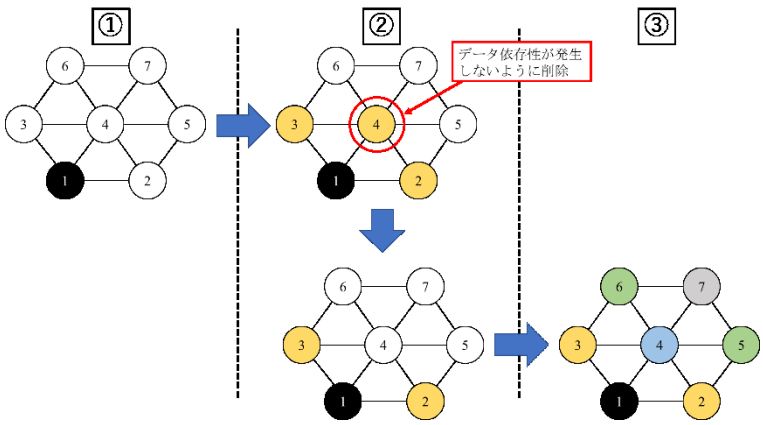


図 2　本研究で用いた Cuthill-Mckee 法

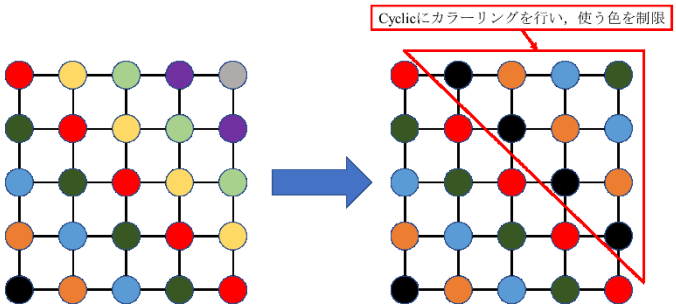


図 3　Cyclic マルチカラーリングの概要

Cyclic マルチカラーリングの詳細を図 3 に示す．図 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 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．

3. Multigrid 法

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．そこで，Multigrid 法ではこの性質を用い，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，高い収束性を実現している．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．本章ではまず，Multigrid 法の中の解法のひとつである AMG 法についての説明を行い，次に AMG 法の派生解法である SA-AMG 法についての説明を行う．

3.1 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．AMG 法ではまず，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用い，実際に問題を解く．

構築部の概要を図 4 に示す．構築部では細かいレベルの問題行列を基に，粗いレベルの問題行列や補間演算子である Prolongation（P）行列と Restriction（R）行列を階層的に生成する．AMG 法は階層型で，最上階に与えられた問題行列を用い，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．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）を行うことで作成している．

次に，解放部の構造を図 5 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つ．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様々な AMG 法が存在する[6]．本研究では，その中でも SA-AMG 法を対象とする．この手法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．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．

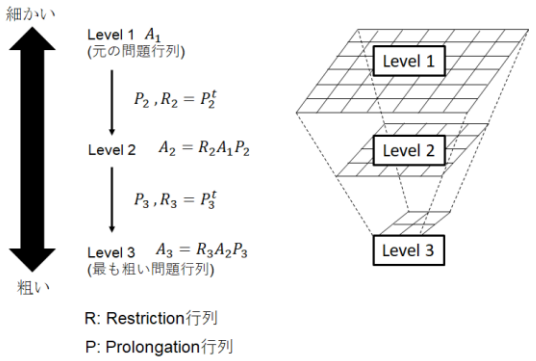


図 4　構築部の概要

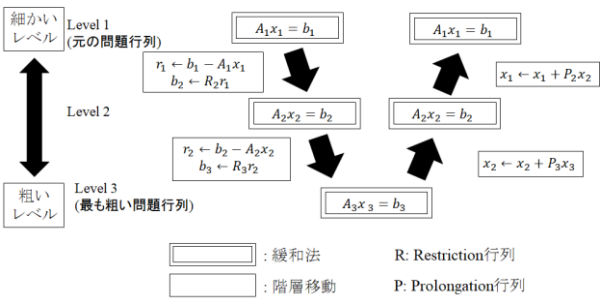


図 5　解法部の概要

Vol.2018-HPC-167 No.27
2018/12/18

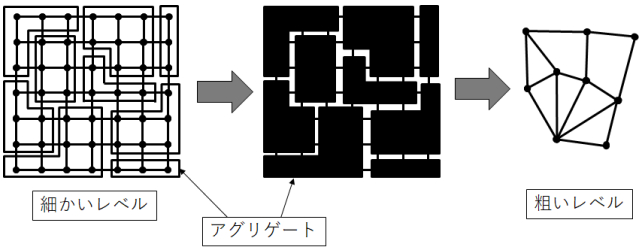


図 6　アグリゲート生成方法

3.2 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．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節点集合に分解する．アグリゲートは図 6 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．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，行列の階層構造を作成する．補間演算子である Prolongation 行列と Restriction 行列は，行列の階層構造を作成する際に用いられる．これらの行列の生成にニアカーネルベクトルを用いることで，収束性をさらに高められることが知られている[6][7]．本研究では，このことについて詳しい言及は行わない．

4. 実験結果

4.1 実験環境と問題サイズ

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），理化学研究所による京，そして九州大学情報基盤研究開発センターによる ITO の，3 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った．各計算機の構成を表 1 に示す．表 1 のように，OFP と京は 1 ノード内に 1CPU（OFP：Intel® Xeon Phi™ 7250（68 コア/CPU），京：SPARC64™ VIIIfx（8 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．

また本研究では，3 次元弾性体の問題を使用した．この問題は，ある物体に対して，力を加えたとき，どれだけ物体が変形するかを解く問題となっている．またこの問題は 3 次元なため，行列に落とし込む際には 1 節点あたりの行列要素が 3×3 のブロック行列となる．実験 2 で用いた弾性体の問題設定を図 9 に示す．この弾性体の問題は図 9 のように，均質な立方体の物体に対して，ある一部分に力を

表 1　スーパーコンピュータシステムの構成

		Oakforest-PACS	京	ITO
Node	CPU 数	1	1	2
	メモリ	16[GB](MCDRAM) (+96[GB] (DDR4))	16 [GB]	192[GB]
CPU		Intel® Xeon Phi™ 7250 (KNL)	SPARC64™ VIIIfx	Intel® Xeon® Gold 6154(Skylake-SP)
	コア数	68 [cores/CPU]	8 [cores/CPU]	18[core/CPU]
	Frequency	1.40 [GHz]	2.0 [GHz]	3.0[GHz]

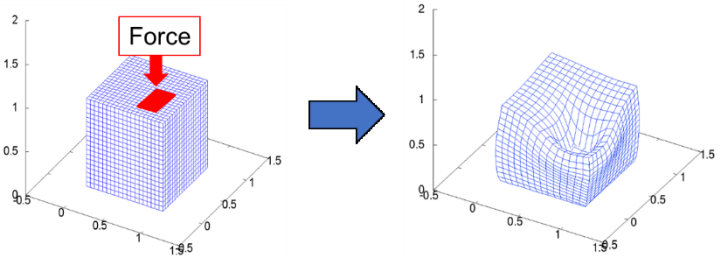


図 7　本研究で用いた 3 次元弾性体問題の問題設定

加えたとき，どのように変形するかを解く問題となっている．また，弾性体の問題において硬さを表すヤング率を，すべての部分で 1 に設定している．ヤング率は値が大きければ物体が硬いことを表している．用いた問題サイズについては，次章にて説明する．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．

本研究ではさらに、AMGS ライブラリ[8]を使用して実験を行っている．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 CG 法[9]を使用し，前処理として SA-AMG 法を適用している．SA-AMG 法における解法部の緩和法には，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．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．

AMGS ライブラリは，Fortran を用い作成を行っている．また，コンパイラは OFP 上では Intel®コンパイラ，京と 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，京と 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．

4.2 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 2 と表 3, および表 4 にそれぞれ示す． 表 2，表 3，表 4 のように，OFP では 1 ノード，京では 8 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．本実験では実験 1 と同様，使用コア数を等しくするため，使用ノード数を OFP では 1・8・64，京では 8・64・512 とそれぞれ設定した．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 2.2 章で説明した通り，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，対称ガウス・ザイデル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「解法部の結果のみ」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．

表 2 ノード，プロセス，スレッド数構成（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 プロセス当り 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 プロセス当り 32 スレッド起動
1n4p16t	〃 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 3 ノード，プロセス，スレッド数構成（京）

表記	詳細
フラット MPI	8 ノード使用，1 ノード内 8 プロセス起動
8n8p8t	8 ノード使用，1 ノード内 1 プロセス起動し， さらに 1 プロセス当り 8 スレッド起動
8n16p4t	〃 ，1 ノード内 2 プロセス起動し， さらに 1 プロセス当り 4 スレッド起動

表 4 ノード，プロセス，スレッド数構成（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 プロセス当り 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 プロセス当り 16 スレッド起動
2n8p8t	〃 4 プロセス起動し，〃 8 スレッド起動

4.3 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，4.2 節でも述べたように，2 つの実験を行った．

まず，実験 1 について，OFP 上における結果を表 5 と図 8, および図 9 に示す. まず，表 5 についての説明を行う．表 5 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 5 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 5 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, 3.2 章でも述べた通り, 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，本研

究では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 8 についての説明を行う．図 8 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 8 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 8 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 9 に，V-cycle の実行時間における内訳のグラフを示す．図 9 の縦軸および横軸は図 8 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 9 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．

表 5　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

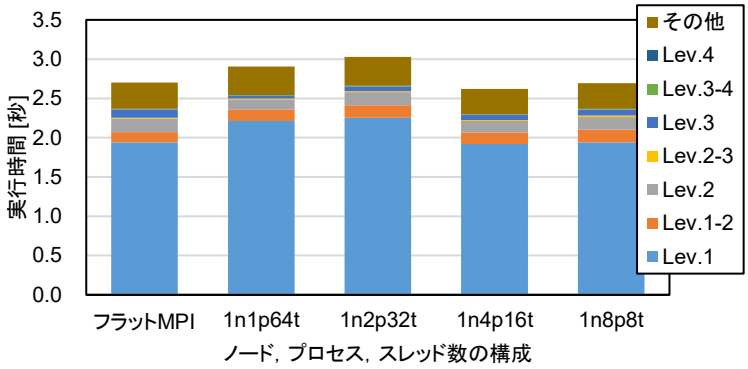


図 8　実行時間に占める V-cycle の割合（OFP）

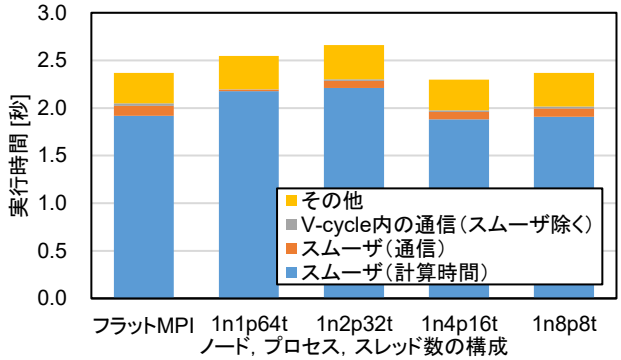


図 9　V-cycle 内の実行時間内訳（OFP）

次に，京においての計測結果を表 6 と図 10 および図 11 に示す．表やグラフの内容は，表 5，図 8，図 9 とそれぞれ同様である．まず，表 6 より，OFP とは違い，Hybrid 並列が遅いという結果となった．これについて図 10 より，レベル 1 の計算時間が要因であることがわかる．さらに図 11 より，Hybrid 並列におけるスムーザの計算時間が，Flat MPI と比べ遅いことがわかる．これらより，京において Hybrid 並列が Flat MPI よりも遅い要因として，Hybrid 並列におけるレベル 1 のスムーザの計算時間が遅いためであると考えられる．この要因として，Hybrid 並列時におけるスムーザの浮動小数点ロードキャッシュアクセス待ち，とりわけ L1 キャッシュにおけるミス率が大きいためであるとわかっている（詳細は付録 A.1，A.2 に示す）．これより，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．

次に，ITO における結果を表 7 と図 13 および図 14 に示す．それぞれの表やグラフの内容は OFP や京のものと同様である．まず，表 7 について述べる．表 7 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 12 に示す．図 12 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

表 6　収束までの実行時間，反復回数，カラーリング時間の結果（京）

	Flat MPI	1n1p64t	1n2p32t
実行時間[秒]	2.80	3.75	3.63
反復回数	26	26	26

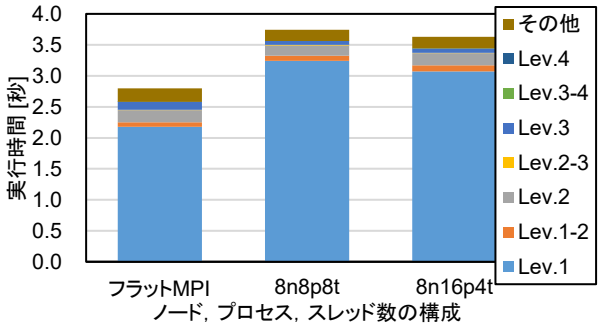


図 10　実行時間に占める V-cycle の割合（京）

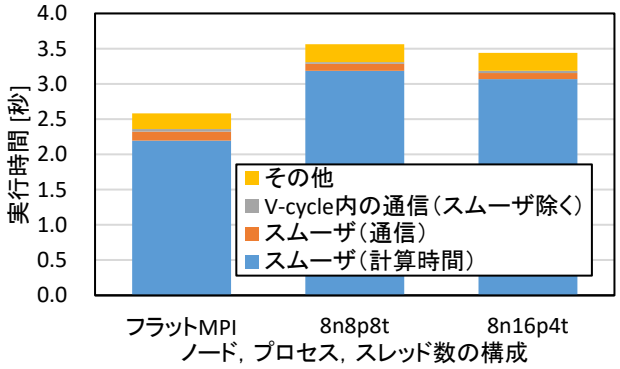


図 11　V-cycle 内の実行時間内訳（京）

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 7 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 13 と図 14 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 13 と図 14 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．

表 7　収束までの実行時間，反復回数，カラーリング時間の結果（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

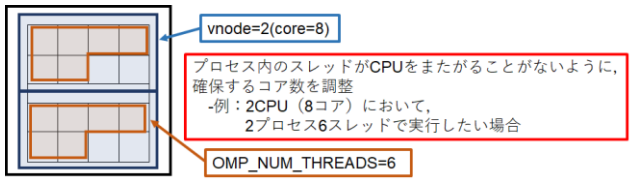


図 12　（fit）の具体例

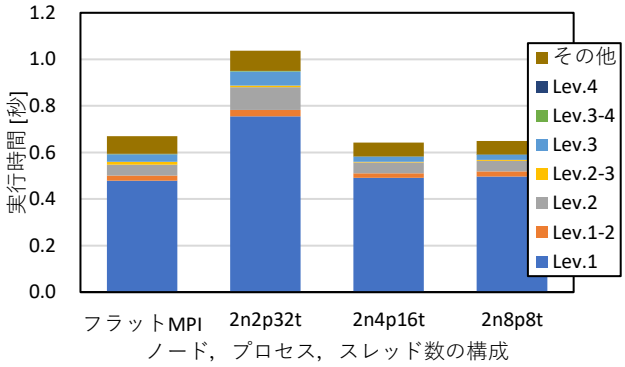


図 13　実行時間に占める V-cycle の割合（ITO）

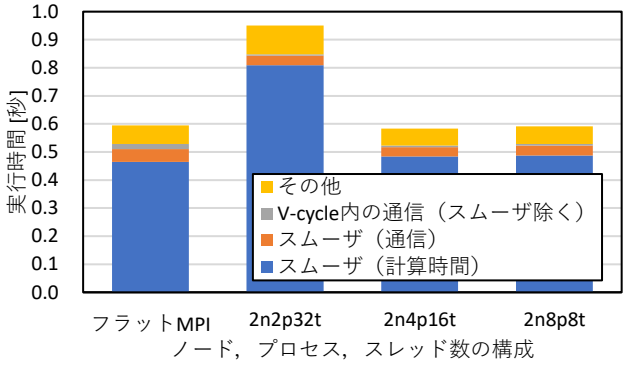


図 14　V-cycle 内の実行時間内訳（ITO）

次に，実験 2 の結果を示す．まず，OFP における結果を表 8 と表 9 に示す．表 8 と表 9 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 8 の括弧内は 1 反復あたりの実行時間を示している．表 8 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 9 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 ノードにおいて図 9 のように分析してみると，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（結果は付録 B.1 に記載）．次に，京における結果を表 10 と表 11 に示す．表 10 と表 11 は表 8 と表 9 と同様，それぞれ実行時間と反復回数を示す．表 10 と表 11 より, OFP と同様, 8 ノードにおける傾向が 512 ノード時においても表れていることがわかる. 最後に, ITO における結果を表 12 と表 13 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 や京と比べ，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 12 と表 13 より, OFP や京と同様, 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 8　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 9　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 10　京：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: 8	2.80 (0.108)	3.75 (0.144)	3.63 (0.140)
Node: 64	4.38 (0.125)	5.31 (0.166)	5.49 (0.161)
Node: 512	5.26 (0.135)	7.94 (0.199)	7.43 (0.177)

表 11　京：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: 8	26	26	26
Node: 64	35	32	34
Node: 512	39	40	42

表 12　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: 64	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)

表 13　ITO：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	37	35
Node: 64	35	31	31

5. おわりに

本研究では，SA-AMG 法に Hybrid 並列を適用し，Oakforest-PACS（JCAHPC），京（理化学研究所），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，問題サイズを 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．京においては，Hybrid 並列は Flat MPI よりも遅いという結果となった．これは，Hybrid 並列時におけるスムーザの計算部分が影響しており, 京における Hybrid 並列時のスムーザの計算部分の最適化に関しては, 今後の課題とする. また，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.

今後の課題として，まずは上記でも述べたように，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．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.

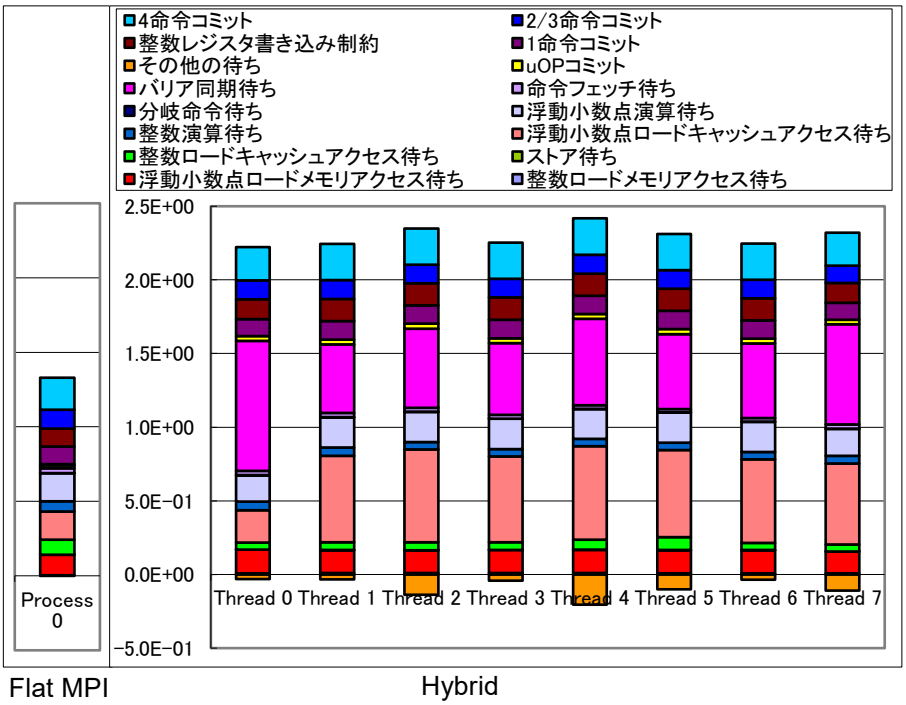
参考文献

[1] Pereira, F. H., Verardi, S. L. L., and Nabeta, S. I.: “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”, Numerische Mathematik 2001, vol 88, pp.559-579, 2001.
[3] Vanek, P., Mandel, J. and Brezina, M.: “Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems”, Computing, Vol.56, pp.179-196, 1998.
[4] Chan, T. F. and Vanek, P.: “Multilevel algebraic Elliptic Solvers”, UCLA Math, Dept. CAM Report, 1999.
[5] E. Cuthill and J. McKee, “Reducing the bandwidth of sparse symmetric matrices,” in Proc. of 24th national conference. ACM, 1969, pp. 157– 172.
[6] Brezina, M., Falgout, R., Maclachlan, S., Manteuffel, T., Mccormick, S. and Ruge, J.: Adaptive Smoothed Aggregation (αSA), SIAM J. Sci. Comput, Vol. 25, No.6, pp.1896-1920 (2004)
[7] Nomura, N., Nakajima, K. and Fujii, A.:“Robust SA-AMG Solver by Extraction of Near-Kernel Vectors”, SC17 Poster (2017).
[8] AMGS Library: http://hpcl.info.kogakuin.ac.jp/lab/software/amgs
[9] Hestenes, M. R. and Stiefel, E.: “Methods of Conjugate Gradients for Solving Linear Systems”, Journal of Research of the National Bureau of Standards, Vol. 49 , No.6, pp.409-436 (1952) .

付録

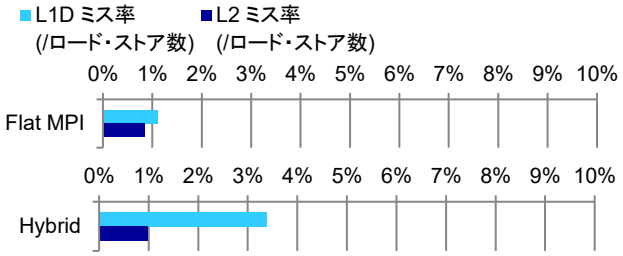
付録 A.1 京におけるスムーザ計算部分プロファイル結果

1 コア当たりの問題サイズを 15^3 に設定し，スムーザの計算部分のみをプロファイラにより分析した結果．Hybrid 並列において，赤橙色の要素「浮動小数点ロードキャッシュアクセス待ち」と紫色「バリア同期待ち」が大きいことがわかる．バリア同期待ちがキャッシュアクセス待ちによるものと考えると，Hybrid 並列が低速な理由が主に浮動小数点キャッシュアクセス待ちによるものと考えられる．



付録 A.2 キャッシュミス率の詳細

A1 において，さらにキャッシュミスに関しての詳細なプロファイル結果．この図より，Hybrid 並列は特に L1 に関するミス率が大きいことがわかる．



付録 B.1 実験 2 での OFP における 64 ノード時の V-cycle 内の実行時間内訳