計算工学講演会論文集　Vol.28 (2023年5月)　　計算工学会

並列有限要素構造解析コードADVENTURE_Solidの富岳向けチューニング

Tuning Up Parallel Finite Element Structural Analysis Code ADVENTURE_Solid for Supercomputer Fugaku

宮村倫司[1)]，小池邦昭[2)]，吉村忍[3)]
Tomoshi Miyamura, Kuniaki Koike, and Shinobu Yoshimura

1) 日本大学工学部情報工学科（〒963-8642 郡山市田村町徳定字中河原１番地)
2)（株）先端力学シミュレーション研究所（〒112-0002文京区小石川5-5-5 プライム茗荷谷ビル5F）
3) 東京大学大学院工学系研究科システム創成学専攻（〒113-8656 文京区本郷7-3-1）

ADVENTURE_Solid is a parallel finite element structural analysis code that utilizes the hierarchical domain decomposition method. The code incorporates the balancing domain decomposition (BDD) method as a parallel linear solver. Supercomputer Fugaku is a massively parallel supercomputer consisting of approximately one hundred and sixty thousand computation nodes. In this study, an enhanced parallel implementation of the BDD method in ADVENTURE_Solid is presented for optimal performance on Supercomputer Fugaku.

Key Words : *Parallel finite element analysis, ADVENTURE_Solid, Domain decomposition, Supercomputer Fugaku, Sparse direct solver*

1.　はじめに

ADVENTURE_Solidは階層型領域分割法をベースとした陰的な有限要素法に基づく並列構造解析コードであり，オープンソースのCAEシステムであるADVENTUREシステムに含まれる解析モジュールのひとつである[1][2]．一方，スーパーコンピュータ「富岳」[3]は，48コアのCPUを持つ約16万個の計算ノードを6次元メッシュ／トーラスネット―ワークであるTofuD インターコネクトで結合した構成のスーパーコンピュータ（スパコン）である．多数の計算ノードを持つ富岳では，数千から数万個の計算ノードを用いた計算ジョブを日常的に実行することが可能であるため，その計算資源を活用できるようなソフトウェアの開発が求められる．

本研究では，最初に従来のADVENTURE_Solid（ver. 2）を富岳で実行した場合の問題点について分析する．次に，数万個の計算ノードを用いても十分な並列化効率が得られるような実装方法を提案する．それに基づいて実装したADVENTURE_Solid（ver. 3）の計算性能を「富岳」上で測定し，その結果を報告する．

2.　BDD法

Mandel[4]により提案されたバランシング領域分割（Balancing Domain Decomposition; BDD）法は，部分構造型領域分割法に基づく，線形問題に対する前処理付きの反復解法である．ここではその定式化の概要を示す．BDD法の計算過程では元の線形問題よりも小規模ないくつかの線形問題を解く必要がある．実装方法を検討するためにこれらを整理する．

(1) BDD法の定式化

有限要素法に基づき離散化された静的線形構造解析における釣り合い式は，次のような連立一次方程式となる．

$$\mathbf{K}\mathbf{u} = \mathbf{p} \tag{1}$$

ここに，$\mathbf{K}$ は剛性行列，$\mathbf{u}$ は節点変位ベクトル，$\mathbf{p}$ は節点荷重ベクトルである．

ここでは，領域分割法の中でも部分領域内部の自由度を消去する部分構造型領域分割法を用いる．まず，解析領域をN個のオーバーラップのない部分領域に分割する．部分領域 k に対する剛性行列を $\mathbf{K}^{(k)}$，節点変位を $\mathbf{u}^{(k)}$，節点荷重を $\mathbf{p}^{(k)}$ としたとき，これらの行列，ベクトルを次式のように部分領域内部の自由度と部分領域間境界の自由度に分割する．

$$\mathbf{K}^{(k)} = \begin{bmatrix} \mathbf{K}_{\mathrm{II}}^{(k)} & \mathbf{K}_{\mathrm{IB}}^{(k)} \\ \mathbf{K}_{\mathrm{BI}}^{(k)} & \mathbf{K}_{\mathrm{BB}}^{(k)} \end{bmatrix}, \quad \mathbf{u}^{(k)} = \begin{bmatrix} \mathbf{u}_{\mathrm{I}}^{(k)} \\ \mathbf{u}_{\mathrm{B}}^{(k)} \end{bmatrix}, \quad \mathbf{p}^{(k)} = \begin{bmatrix} \mathbf{p}_{\mathrm{I}}^{(k)} \\ \mathbf{p}_{\mathrm{B}}^{(k)} \end{bmatrix} \tag{2}$$

ここに，添字Iは部分領域内部の自由度を，添字Bは部分領域間境界上の自由度を表す．領域間境界全体の自由度数を n_{B} とする．また，部分領域 k に関係する領域間境界自由度の自由度数を $n_{\mathrm{B}}^{(k)}$ とする．

部分領域の内部の自由度に関する釣り合いは次式のように各部分領域において独立に成立する．

$$\mathbf{K}_{\mathrm{II}}^{(k)}\mathbf{u}_{\mathrm{I}}^{(k)}+\mathbf{K}_{\mathrm{IB}}^{(k)}\mathbf{u}_{\mathrm{B}}^{(k)}=\mathbf{p}_{\mathrm{I}}^{(k)} \tag{3}$$

一方，部分領域間境界自由度の釣り合いは次式で表される．

$$\sum_{k=1}^{N}\mathbf{N}_{\mathrm{B}}^{(k)}\mathbf{K}_{\mathrm{BI}}^{(k)}\mathbf{u}_{\mathrm{I}}^{(k)}+\sum_{k=1}^{N}\mathbf{N}_{\mathrm{B}}^{(k)}\mathbf{K}_{\mathrm{BB}}^{(k)}\mathbf{u}_{\mathrm{B}}^{(k)}=\sum_{k=1}^{N}\mathbf{N}_{\mathrm{B}}^{(k)}\mathbf{p}_{\mathrm{B}}^{(k)} \tag{4}$$

ここに，$\mathbf{N}_{\mathrm{B}}^{(k)}$ は部分領域 k に関する領域間境界自由度を全ての部分領域に対する領域間境界自由度に変換するブーリアン行列である．式(3)は次のように内部節点変位について解くことができる．

$$\mathbf{u}_{\mathrm{I}}^{(k)}=\left(\mathbf{K}_{\mathrm{II}}^{(k)}\right)^{-1}\left(\mathbf{p}_{\mathrm{I}}^{(k)}-\mathbf{K}_{\mathrm{IB}}^{(k)}\mathbf{u}_{\mathrm{B}}^{(k)}\right) \tag{5}$$

式(5)を式(4)に代入すると次式のようにまとめられる．

$$\mathbf{S}\bar{\mathbf{u}}_{\mathrm{B}}=\bar{\mathbf{p}}_{\mathrm{B}} \tag{6}$$

ここに，$\mathbf{S}$ はSchur complement，$\bar{\mathbf{u}}_{\mathrm{B}}$ は部分領域間境界上の全ての節点に対する節点変位ベクトル，$\bar{\mathbf{p}}_{\mathrm{B}}$ は静的縮約された部分領域間境界上の節点荷重ベクトルである．行列 $\mathbf{S}$ は $n_{\mathrm{B}}\times n_{\mathrm{B}}$ 型行列となる．部分領域 k に対するローカルSchur complementを $\mathbf{S}^{(k)}$ とすると，全体のSchur complementである $\mathbf{S}$ は次式となる．

$$\mathbf{S}=\sum_{k=1}^{N}\mathbf{N}_{\mathrm{B}}^{(k)}\mathbf{S}^{(k)}\mathbf{N}_{\mathrm{B}}^{(k)\mathrm{T}} \tag{7}$$

ここに，

$$\mathbf{S}^{(k)}=\mathbf{K}_{\mathrm{BB}}^{(k)}-\mathbf{K}_{\mathrm{BI}}^{(k)}\mathbf{K}_{\mathrm{II}}^{(k)-1}\mathbf{K}_{\mathrm{IB}}^{(k)} \tag{8}$$

である．

式(6)は前処理付き共役勾配法で解くものとする．Mandel[4]が提案したバランシング領域分割法（Balancing Domain Decomposition Method, BDD法）はこの問題に対する効果的な前処理手法であり，ADVENTURE_Solidにも実装されている．その前処理行列は次式となる．

$$\begin{aligned}\mathbf{M}_{\mathrm{BDD}}^{-1}=&\left(\mathbf{I}-\mathbf{R}\mathbf{K}_{\mathrm{C}}^{-1}\mathbf{R}^{\mathrm{T}}\mathbf{S}\right)\mathbf{M}_{\mathrm{NN}}^{-1}\left(\mathbf{I}-\mathbf{S}\mathbf{R}\mathbf{K}_{\mathrm{C}}^{-1}\mathbf{R}^{\mathrm{T}}\right)\\&+\mathbf{R}\mathbf{K}_{\mathrm{C}}^{-1}\mathbf{R}^{\mathrm{T}}\end{aligned} \tag{9}$$

ここに，$\mathbf{M}_{\mathrm{NN}}^{-1}$ はNeumann-Neumann前処理のための前処理行列である．$\mathbf{R}$ と $\mathbf{R}^{\mathrm{T}}$ はそれぞれマルチグリッド法prolongation行列とrestriction行列に相当する．固体，構造解析では行列 $\mathbf{R}$ は $n_{\mathrm{B}}\times 6N$ 型行列となる．$\mathbf{K}_{\mathrm{C}}$ は次式によって計算される．

$$\mathbf{K}_{\mathrm{C}}=\mathbf{R}^{\mathrm{T}}\mathbf{S}\mathbf{R} \tag{10}$$

行列 $\mathbf{K}_{\mathrm{C}}$ はCoarse grid（コース）行列と呼ばれ，剛性行列 $\mathbf{K}$ を部分領域の剛体運動を利用して低次元で近似した行列となる．$\mathbf{K}_{\mathrm{C}}$ を係数行列とした線形問題をコース（グリッド）問題と呼ぶ．コース問題を解いた結果を用いてCG法の収束性を高める処理は，前処理の一種でありコースグリッド修正と呼ばれる．初期残差に対してコースグリッド修正をするとそれ以降はコース空間の成分が取り除かれるため，式(9)の第1項の最初のコースグリッド修正と最後の項を省略できる．このとき，前処理行列は，

$$\mathbf{M}_{\mathrm{BDD}}^{-1}=\left(\mathbf{I}-\mathbf{R}\mathbf{K}_{\mathrm{C}}^{-1}\mathbf{R}^{\mathrm{T}}\mathbf{S}\right)\mathbf{M}_{\mathrm{NN}}^{-1} \tag{11}$$

となる．

Neumann-Neumann前処理を表す $\mathbf{M}_{\mathrm{NN}}^{-1}$ は，

$$\mathbf{M}_{\mathrm{NN}}^{-1}=\sum_{k=1}^{N}\mathbf{N}_{\mathrm{B}}^{(k)}\mathbf{D}^{(k)}\left(\mathbf{S}^{(k)}\right)^{+}\mathbf{D}^{(k)\mathrm{T}}\mathbf{N}_{\mathrm{B}}^{(k)\mathrm{T}} \tag{12}$$

と表される．ここに，$\mathbf{D}^{(k)}$ は領域間境界自由度を共有する部分領域の数の逆数を対角成分とした対角行列であり，重み行列と呼ばれる．$\left(\mathbf{S}^{(k)}\right)^{+}$ はローカルSchur complementの一般逆行列を表す．荻野[2]は一般逆行列を使う代わりに次式のようにペナルティ項を加えて正則化し，逆行列を使う方法を提案した．

$$\left(\mathbf{S}^{(k)}\right)^{+}\approx\left(\mathbf{S}^{(k)}+\max\left(diag\left(\mathbf{K}^{(k)}\right)\right)\mathbf{I}\right)^{-1} \tag{13}$$

(2) 解くべき線形問題の整理

BDD法の計算過程で解く必要がある線形問題を整理する．以下では，一般的なベクトルを $\mathbf{a}$，$\mathbf{b}$ あるいはそれらに添字を加えた記号で表す．

CG法の計算過程における残差ベクトルの計算，式(10)のコース行列の作成，式(11)のNeumann-Neumann前処理の後に $\mathbf{S}$ との積の計算がある．そのために，次式のようなローカルSchur complement $\mathbf{S}^{(k)}$ とベクトルの積の計算が必要である．

$$\mathbf{b}_{\mathrm{B}}^{(k)}=\mathbf{S}^{(k)}\mathbf{a}_{\mathrm{B}}^{(k)} \tag{14}$$

$\mathbf{S}^{(k)}$ を陽に作る代わりに次式のように本来のディリクレ境界に加えて部分領域境界自由度もディリクレ境界として与えた問題（ディリクレ問題）を解く．ディリクレ境界の処理は，ディリクレ境界自由度を除く方法ではなく，次式のようにその自由度に関する係数行列の対角項に1を入れる方法で処理する．

$$\begin{bmatrix}\mathbf{K}_{\mathrm{II}}^{(k)} & \mathbf{O}\\ \mathbf{O} & \mathbf{I}_{\mathrm{BB}}\end{bmatrix}\begin{bmatrix}\mathbf{u}_{\mathrm{I}}^{(k)}\\ \mathbf{u}_{\mathrm{B}}^{(k)}\end{bmatrix}=\begin{bmatrix}-\mathbf{K}_{\mathrm{IB}}^{(k)}\mathbf{b}_{\mathrm{B}}^{(k)}\\ \mathbf{a}_{\mathrm{B}}^{(k)}\end{bmatrix} \tag{15}$$

ここに，$\mathbf{I}_{\mathrm{BB}}$ は単位行列を表す．左辺の係数行列の逆行列を陽に求めるのではなく，線形ソルバーを使う．次に，$\mathbf{b}_{\mathrm{B}}^{(k)}$ を次のように求める．

$$\mathbf{b}_{\mathrm{B}}^{(k)}=\begin{bmatrix}\mathbf{K}_{\mathrm{BI}}^{(k)} & \mathbf{K}_{\mathrm{BB}}^{(k)}\end{bmatrix}\begin{bmatrix}\mathbf{u}_{\mathrm{I}}^{(k)}\\ \mathbf{u}_{\mathrm{B}}^{(k)}\end{bmatrix} \tag{16}$$

一方，Neumann-Neumann前処理では，次式のように式

(13)の一般逆行列を近似した行列とベクトルの積の計算が必要である．

$$\mathbf{b}_{\mathrm{B}}^{(k)} = \left(\mathbf{S}^{(k)} + \max\left(diag\left(\mathbf{K}^{(k)}\right)\right)\mathbf{I}\right)^{-1}\mathbf{a}_{\mathrm{B}}^{(k)} \qquad (17)$$

ここでも $\mathbf{S}^{(k)}$ を陽に作る代わりに次の方程式（ノイマン問題）を線形ソルバーにより解く．

$$\begin{bmatrix} \mathbf{K}_{\mathrm{II}}^{(k)} & \mathbf{K}_{\mathrm{IB}}^{(k)} \\ \mathbf{K}_{\mathrm{BI}}^{(k)} & \mathbf{K}_{\mathrm{BB}}^{(k)} + \max\left(\mathrm{diag}\left(\mathbf{K}^{(k)}\right)\right)\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{a}_{\mathrm{I}}^{(k)} \\ \mathbf{a}_{\mathrm{B}}^{(k)} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b}_{\mathrm{B}}^{(k)} \end{bmatrix} \qquad (18)$$

最後に，コースグリッド修正に含まれる $\mathbf{K}_{\mathrm{C}}^{-1}$ とベクトルの積についても，逆行列を陽に作る代わりに線形ソルバーを用いて計算する．すなわち，

$$\mathbf{b} = \mathbf{K}_{\mathrm{C}}^{-1}\mathbf{a} \qquad (19)$$

を求めるために，

$$\mathbf{K}_{\mathrm{C}}\mathbf{b} = \mathbf{a} \qquad (20)$$

を線形ソルバーにより解く．

以上のように，BDD法の実装では式(15)，(18)，(20)の求解の実装方法がポイントとなる．式(15)，(18)は部分領域単位の局所的な計算であるため，ローカル問題と呼ぶ．式(20)の計算はコース問題と呼ぶ．

3. ADVENTURE_Solid におけるバランシング領域分割法の従来の実装方法

(1) 階層型領域分割法

ADVENTURE_Solid は Yagawa and Shioya[5]が提案した階層型領域分割法をベースとして並列実装をしている．図-1 のようにメッシュはまず Part と呼ばれる領域に分割され，各 Part がさらに部分領域に分割される．階層型領域分割はドメインデコンポーザーADVENTURE_Metis で行う．その中で，Part への分割には ParMetis[6]が，Part の部分領域への分割には Metis[7]が使われている．

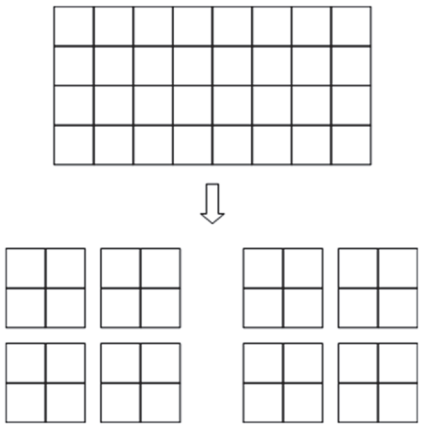

図-1 階層型領域分割

(2) 従来の実装方法

非公開のADVENTURE_Solid FS版では式(15)，(18)のローカル問題は直接法の一種であるスカイライン法のシリアル処理版で解いている．一方，式(20)のコース問題はスパース直接法ソルバーMUMPS[8]で解いている．ADVENTURE_SolidはMPIにより並列化されており，1個のPartを1個のMPIプロセスに割り当てている．マルチコアCPUを使う場合には，1個のMPIプロセスの中でOpenMPによるスレッド並列化を行っており，1個のPartの中にある複数の部分領域に対する式(15)，(18)のローカル問題をコア数と同数のスレッドに割り当てることで並列に計算する．この並列化では理想的なスケーラビリティが得られている．一方，コース問題に使うMUMPSもMPI並列化されているが，超並列計算機ではPart数と同数のMPIプロセス数は非常に大きいため，Partの情報をMUMPSに適した数のMPIプロセスに集約し，少ないMPIプロセス数で並列計算を行っている．また，MUMPSで使われるBLASをスレッド並列版とすることで複数のコアを使ったマルチスレッド並列化ができる．

(3) 従来版による原子力発電所モデルの解析

富岳に実装したADVENTURE_Solid FS版の従来版により，四面体二次要素による福島第1原子力発電所1号機モデルの静的解析を富岳により行う．このモデルの自由度数は1,497,322,665（約15億）である．なお，富岳の1個の計算ノードは4個のCMGと呼ばれる単位により構成され，48個のコアとメインメモリの関係は均質ではない．ここでは1個のCMGに1個のMPIプロセスを割り当てて計算する．したがって，1個のMPIプロセスには12個のコアが割り当てられる．

表-1に5種類の領域分割のケースを示す．ここではCase AとBに対して従来版により解析を行った結果を示す．表-2にそれぞれのケースにおけるコース問題の規模とローカル問題の規模を示す．コース問題の規模は全部分領域数×6（剛体モードの数）となる．ローカル問題の規模は，全自由度数/全部分領域数で概算したものである．実際には部分領域間境界上の自由度が重複することや，式(15)のディリクレ問題よりも式(18)のノイマン問題の方が係数行列の非零成分少ないという差異はあるものの，計算のオーダーの見積りにはなっている．富岳上の従来版により計算性能に対するパラメータスタディをした結果，従来版ではCase Aの領域分割によって最も速い計算速度が得られている．

図-2にCG法の1反復ステップ（1CG）の計算におけるコース問題の計算時間とその他の計算時間の割合を示す．Case Aのローカル問題の数はCase Bの10倍以上で，ローカル問題の規模は1/10以下となっている．ローカル問題の求解に用いているスカイライン法は自由度に対してスケーラブルな解法ではないため，全てのローカル問題に対する計算時間はCase Aの方が圧倒的に短い．しかし，コース問題の規模が大きいため，MUMPSによるコース問題の求

解の時間が全体の半分近くを占めている．MUMPSに割り当てるMPIプロセス数は96～192程度であり，全体の6144個のMPIプロセスの中のほんの一部に過ぎないため，MUMPSの計算の間はほとんどのMPIプロセスは何もしていないことになる．一方，Case Bでは1個のPartの部分領域数を1個に減らすことでコース問題の規模を小さくしている．コース問題とローカル問題の規模は同程度である．ただし，コース問題はMUMPSで，ローカル問題はスカイラインソルバーで解いているため，計算時間の大半はローカル問題の求解時間になっている．全体の計算速度はCase Aよりも大幅に遅く，収束まで計算していない．

プロセスとスレッドの割り当て，および，1CGおよび全体の計算時間とCG法の反復回数は後の表-3, 4にそれぞれ改良版の情報と共に示す．CG法の収束判定では，後の改良版による解析も含めて相対残差の閾値を1.0×10^{-3}としている．

4. ローカル問題にもMUMPSを使った実装

(1) 実装

3(3)節の数値実験の結果から，部分領域数を調整することで，コース問題とローカル問題の規模を同程度にすることができ，多くの計算ノードを有効活用できる可能があることがわかった．しかし，その結果，ローカル問題の規模が大きくなり，スカイライン法を使うと計算時間が長くなる．そこで，ローカル問題にもMUMPSを使えるように解析コードを改良する．

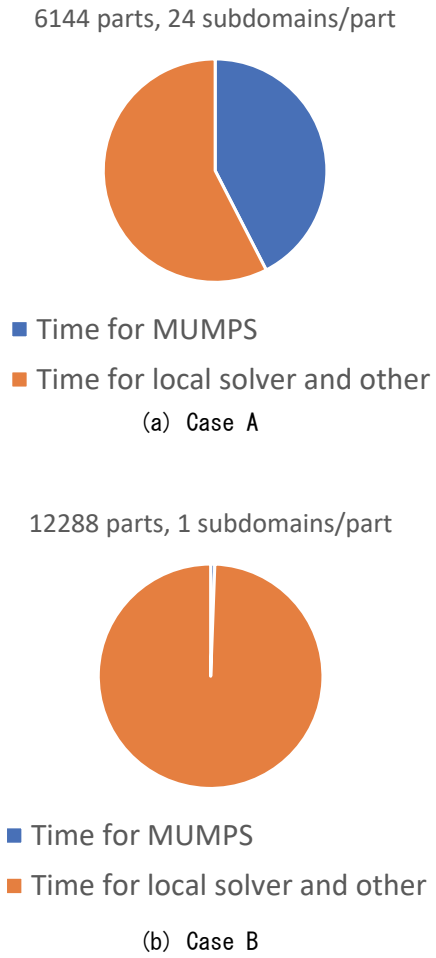

図-2 1CG の計算におけるコース問題の計算時間とその他の計算時間の割合（従来版による）

表-1 領域分割

ケース	Part数	1Partあたりの部分領域数	全部分領域数
Case A	6,144	24	147,456
Case B	12,288	1	12,288
Case C	6144	4	24576
Case D	12,288	2	24576
Case E	24000	1	24000

表-2 コース問題とローカル問題の規模

ケース	コース問題の規模	ローカル問題の規模
Case A	約88万自由度	約1万自由度
Case B	約7.3万自由度	約12.2万自由度
Case C	約15万自由度	約6.1万自由度
Case D	約15万自由度	約6.1万自由度
Case E	約14万自由度	約6.3万自由度

1個の部分領域に関するローカル問題をMPIにより分散並列化されているMUMPSで解く場合，1個の部分領域に複数のMPIプロセス（以下，プロセス）を割り当てる必要がある．従来のADVENTURE_Solidでは，図-3のように部分領域を管理するPartに1個のプロセスを割り当てていたため，プロセスの割り当て方法を見直す必要がある．

MPIでは集団的操作の対象となるプロセスの集まりをコミュニケータとして定義する．図-4に改良したADVENTURE_Solid（改良版）におけるMPIのコミュニケータの配置を示す．従来の実装ではPart数と同数のプロセスで構成されるMaster Groupのみであったのに対し，今回の実装ではWorker Groupを加える．MPI_COMM_WORLDは全プロセスを表すコミュニケータである．PLUコースグリッドMUMPSコミュニケータは，コースグリッド問題の求解に使うMUMPS用である．MPC MUMPSコミュニケータは，MPCを含む問題において拘束条件に関する射影を行うMUMPS用のコミュニケータである．このコミュニケータは複数個配置することもある．以上のコミュニケータはMaster Groupの中に置く．MUMPSローカルコミュニケータは，ローカル問題の求解に使うMUMPS用であり，Partの数と同じ数になるように生成する．コミュニケータ

に2個以上のプロセスが含まれる場合には，Master Groupのプロセスだけでは足らないので, Worker Groupのプロセスを使う．1個のPartに複数の部分領域がある場合には，MUMPSのオブジェクトは部分領域毎に生成するが，計算は順番に行うものとしてコミュニケータは同じもの（各Partに一つ配置）を使う．

(2) 計算性能評価

表-1のCase C, D, Eについて，改良版で解析を行う．表-3に各ケースのプロセスとスレッドの割り当てを，表-4に計算時間とCG法の反復回数を示す．改良版ではローカル問題に対して多くのMPIプロセスが必要なこと，また，スレッド並列が部分領域毎の処理に対するものではなく，スレッド並列版のBLASに対するものであるため，並列性能が低いことから, 1個の計算ノードに8個のMPIプロセスを割り当てている．Case C, D, Eの全部分領域数はほぼ同じであり，違いは1個のPartあたりの部分領域数である．

Case CとDについて，Case C のCG法の反復数がCase Dと同じであると仮定した上で並列化効率を求めると，66.4%となり良好な結果となっている．1CGループの計算時間で比較すると，Case Eではさらに高速になっている．しかし，Case Eでは収束までのCG法の反復回数が多く，全計算時間は従来版の最速値であるCase Aの計算時間よりも遅くなっている．BDD法の性質により，部分領域数が少なくなるとコースグリッド修正の性能が低下して反復回数が増大する．それだけでなく，1個のPartあたりの部分領域数が1個の場合，収束性がより低下する現象がみられる．部分領域数を少し変えたケースでも同様の傾向となった．Partあたりの部分領域数が1の場合，領域分割はParMetisのみで行うことになるので，ParMetisによる領域分割の品質が，Metisによるものと比べてやや劣っている可能性が考えられる．

5. おわりに

本研究では，階層型領域分割法に基づく並列有限要素構造解析コードADVENTURE_Solid (ver. 2) を富岳で実行した場合の問題点について分析した．次に，数万個の計算ノードを用いても十分な並列化効率が得られるように，ローカル問題をスパース直接法ソルバーMUMPSで解くような実装を行い（ADVENTURE_Solid（ver. 3））, その計算性能を評価した．

謝辞：本研究は，文部科学省「富岳」成果創出加速プログラム「スーパーシミュレーションとAIを連携活用した実機クリーンエネルギーシステムのデジタルツインの構築と活用」（JPMXP1020200303）の一環として実施されたものです．また, 本研究の一部は, 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（課題番号：hp210175, hp220169）．

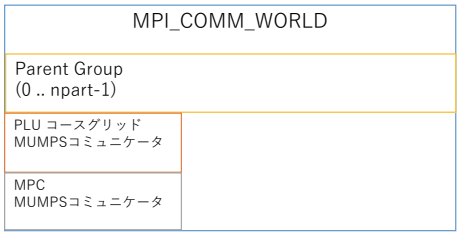

図-3 従来版におけるコミュニケータの配置

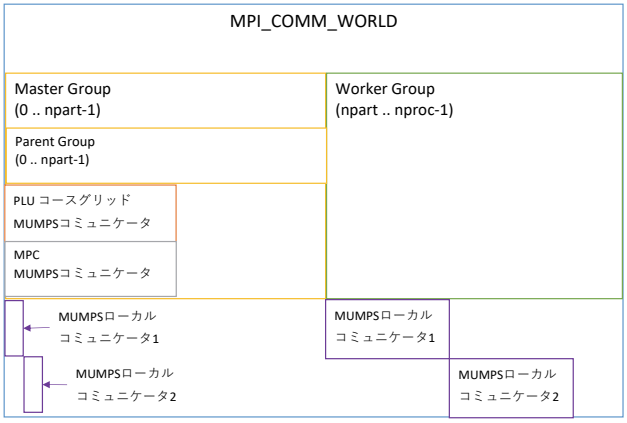

図-4 改良版におけるコミュニケータの配置

表-3 MPIプロセスとスレッドの割り当て（Case A, Bは従来版，Case C, D, Eは改良版）

ケース	ノード数	MPIプロセス数	部分領域あたりのプロセス数	コース問題用プロセス数	プロセスあたりのスレッド数
Case A	1,536	6,144	1	192	12
Case B	3,072	12,288	1	16	12
Case C	3,072	24,576	4	24	6
Case D	6,144	49,152	4	24	6
Case E	12,000	96,000	4	24	6

表-4 計算時間とCG法の反復数

ケース	1CGの計算時間（s）	CG法の反復回数	全計算時間（s）
Case A	0.55	678	461
Case B	3.72	—	—
Case C	0.86	1,260	1,193
Case D	0.53	1,024	594
Case E	0.38	1,321	524

参考文献

[1] Home page of ADVENTURE Project, https://adventure.sys.t.u-tokyo.ac.jp
[2] 荻野正雄，塩谷隆二，金山寛，田上大助，吉村忍：バランシング領域分割法による並列弾性有限要素解析,

日本機械学会論文集 A 編, Vol.69, No.685, pp.1360-1367, 2003.

[3] Web site of Fugaku, https://www.r-ccs.riken.jp/en/fugaku/

[4] Mandel, J: Balancing domain decomposition, *Communications on Num. Methods in Eng*. Vol. 9, pp. 233-241, 1993.

[5] Yagawa, G., Shioya, R.: Parallel finite elements on a massively parallel computer with domain decomposition, *Comput. Syst. Engng*., Vol. 4, pp. 495–503, 1993.

[6] Karypis, G, Kumar, V: Parallel Multilevel K-Way Partitioning Scheme for Irregular Graphs, *Tech. Rep., Dept. of Comp. Sci. Univ. of Minnesota*, TR 96-036, 1996.

[7] Karypis. G, Kumar, V: Multilevel K-Way Partitioning Scheme for Irregular Graphs, *Tech. Rep., Dept. of Comp. Sci. Univ. of Minnesota*, TR 95-064, 1995.

[8] Home page of MUMPS: a parallel sparse direct solver, https://mumps-solver.org/