並列分散環境における AMR のマイグレーション

Data migration for AMR on distributed parallel environment

○ 小野 謙二　理研　埼玉県和光市広沢 2-1　E-mail : keno@riken.jp
山崎 昇　みずほ情報総研　東京都千代田区神田錦町 2-3　E-mail : noboru.yamazaki@mizuho-ir.co.jp
Kenji ONO　RIKEN　2-1 Hirosawa, Wako, Saitama
Noboru YAMAZAKI　Mizuho IR　2-3 Kanda-Nishiki-Cho, Chiyoda-ku, Tokyo

An incompressible flow solver was developed on pFTT data structure that allows to independently add or delete leaf cells in the tree data management. The flow solver is constructed by HSMAC algorithm to avoid extra communications between nodes during the pressure iteration, which is a considerable part in the flow calculation. A load balance between nodes is maintained by equally division of an array list that consists collected leaf cells from a whole computational domain. In this paper, a novel algorithm to migrate leaf cells using a communication matrix was presented and was evaluated on a parallel environment of 128 cores. The measured performance results showed that the speedup ratio is about 55 times and the fraction that can be made parallel is 98.953%.

1. はじめに

直交格子を用いたシミュレーションの利点は，計算格子作成の容易性・ロバスト性・迅速性，格子の均一性・直交性による高精度の計算結果などが挙げられる[1]．この特徴は，設計適用計算から現象解析にわたる広範囲の目的に適っている．加えて，計算スキームの高次精度化や物理モデルの導入の点でも，非構造格子や BFC 格子に比べると比較的容易である．また，等間隔の場合には座標変換を用いないため，計算記憶容量を大幅に低減することができ，PC でもかなり大規模な計算が可能になる点も見逃せない．その一方で，任意の形状をいかに精度良く近似して境界条件として取り込むかが計算技術開発の重要な視点になっている．

直交 8 分木格子[2]は，シミュレーション対象となる現象を精度良く計算するために必要な最大の解像度と計算空間スケールの比が大きな場合に有効で，解像度を局所的に変えることにより格子分布の最適化を図ることができる強力な手法である．しかしながら，細分化のレベルが異なる界面において格子幅が急に変化するため，高次精度のスキームが導入しにくい点，並列化した場合の計算処理の複雑さと並列性能の点で解決すべき問題点がある．

8 分木法は AMR(Adaptive Mesh Refinement)[3]と併用して用いられることも多い．この AMR は，計算した物理量やその勾配などの値がある閾値を超えたら格子を細分化し，より細かな格子によって精度の良い結果を得る方法である．AMR を用いると計算結果に応じて格子分布が変化するため，分散並列環境での並列計算の場合，ロードバランスが大きく変化する可能性がある．各ノード間のロードバランスが大きく異なると，並列性能が向上しない問題点が生じる．

本論文では，分散並列環境における直交 8 分木格子を用いた AMR 並列処理に焦点を絞り，負荷分散を最適化する方法の実装と性能評価について報告する．

2. 解法アルゴリズムとデータ構造

2.1 HSMAC 法

計算対象として，三次元非定常非圧縮性流体を考え，無次元化された連続の式 (1) と Navier-Stokes 方程式 (2) を解く．

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial}{\partial x_j}\left(u_i u_j\right) = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j^2} \tag{2}$$

計算方法としては，8 分木格子を有限体積法により離散化，HSMAC 法[4]を用いた．対流項スキームは，現状では一次風上スキームである．

$$u_i^{n+1,m} = u_i^n + \Delta t\left\{\frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j^2} - \frac{\partial\left(u_i u_j\right)}{\partial x_j} - \frac{\partial p}{\partial x_i}\right\}^n \tag{3}$$

$$\left.\begin{aligned} p^{n+1,m+1} &= p^{n+1,m} + \delta p^{n+1,m} \\ \delta p^{n+1,m} &= -\left(\frac{D}{\partial D/\partial p}\right)^{n+1,m} \\ D &= \frac{\partial u_i}{\partial x_i} \\ \frac{\partial D}{\partial p} &= 2\Delta t\left(\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}\right) = \frac{6\Delta t}{h^2} \end{aligned}\right\} \tag{4}$$

$$\left.\begin{aligned} u_i^{n+1,m+1}\Big|_{l+1/2} &= \left[u_i^{n+1,m} + \frac{\Delta t}{h}\delta p^{n+1,m}\right]_{l+1/2} \\ u_i^{n+1,m+1}\Big|_{l-1/2} &= \left[u_i^{n+1,m} - \frac{\Delta t}{h}\delta p^{n+1,m}\right]_{l-1/2} \end{aligned}\right\} \tag{5}$$

ここで，$u_i^{n+1,m}$ における $n+1$ はタイムレベル，m は反復回数を示す．また，δ, h, ω はそれぞれ変化量，格子幅，加速（緩和）係数を示す．また，インデクス l は x, y, z 方向のセルインデクスで，ハーフインデクスはスタガード位置を表す．HSMAC 法に従い，式 (3) により，速度の $n+1$ タイムレベルの予測値を求めた後，式 (4) のニュートン反復により，圧力の変化量 δp と p^{n+1} を求める．その後，式 (5) により着目するセルの面上に配置された速度成分を圧力勾配により修正し，$u_i^{n+1,m+1}$ を得る．収束判定として，非圧縮条件 (1) を直接用いて評価し，収束するまで，$m = 1, 2, ...$ と反復する．HSMAC 法のアルゴリズムは，式 (4) の反復過程において δp の境界条件を必要としないため，収束は遅くなる一方で，並列処理時のノード間の通信が発生しない利点がある．

2.2 データ構造

8 分木格子のデータ構造として，メモリ使用量が少なく，セルの追加削除処理が並列処理に適した FTT[5]データ構造に対し，著者らは隣接セル探索をより高速化したデータ構造 pFTT を提案し，その有効性を示してきた[6]．pFTT のデータ構造を次に示す．

```
struct oct {
  struct cell* cellPtr; // 親セルへのポインタ
  struct cell cell[8];  // 子セルの実体
  struct cell* nghbr[6];// 隣接セルへのポインタ
  short ped[4];     // Pedigree 情報 + padding
}
struct cell {
  int   pos;          // oct 内の自セルの位置
  float phi[m];       // 物理変数とワーク用の変数
  struct oct* child; // 子 oct へのポインタ
  struct oct* myOct; // 自 oct へのポインタ
}
-------------------------------------------
```

構造体 oct は cell の実体を 8 つもち，cell は物理量とワーク用の変数を m 個保持する．ここで，cell から oct へはポインタを介して参照が行われる．cell を細分化する場合には，oct を一つインスタンスし，そのアドレスを child に保持する．逆に，セルを削除する場合には，child に NULL を代入し，対象の oct を削除するだけでよい．この処理は，他の oct/cell には全く影響を与えず局所的であるため，並列処理が可能である．

pFTT は，計算領域全体におけるセルの位置を Pedigree index によるビット列操作で高速に処理する．あるセルの両隣は Pedigree index に ±1 することにより直ちにその位置がわかるため，探索コストが抑えられる．

3. 並列処理

3.1 領域分割

図 1 に 4 分木の場合の領域分割の方法を示す．この図では全計算領域を PE0~PE6 の 7 領域に分割している．計算領域は Z-ordering により Depth First Traversal(DFT) でリスト化する．この方法では，一次元のリストの要素数をノード数（領域数）で等分割することにより，データの局所性を保った上で各領域が担当する計算量を均一化できる．注意する点として，AMR 処理における Join が同一領域内で処理できるようにする．つまり，Oct が持つすべての Cell が LeafCell のときには Oct 内で領域境界を作らない．**図 1** の例において，負荷均等化の点からは 21 と 22 の LeafCell の間に領域境界を設けるのがよいが，それでは 19~22 の LeafCell で Join が起きたときに，同一領域内で処理できなくなる．この場合，領域境界を 22 と 23 の LeafCell の間とする．

並列計算の場合には，各領域内部点の計算に隣接領域の情報が必要になるため，ガイドセルを設けて値を保持する．**図 2** には PE5 の計算に必要なガイドセルを示す．スキームのステンシルは各軸方向に 3 セルを想定しているのでガイドセルは 1 層分としているが，ステンシルが 5 点の場合には 2 層分必要となる．**図 1** と**図 2** から，PE5 のガイドセルは PE2,PE3,PE4,PE6 から集められることがわかる．ある領域の計算に必要なガイドセルが属する領域は，格子の細分化の状況によって異なる．したがって，効率的なガイドセルの構築方法を考える必要がある．

3.2 ガイドセルの構築の方針

各領域が保持する木構造は計算内部セルとガイドセルから構成され，AMR の Split&Join 処理で木構造は変化する．ガイドセルは以下の手順で決定する．

- 各領域内が担当する LeafCell に隣接する Cell がガイドセル候補として仮決めし，これを隣接領域へ送信する．
- この候補から隣接領域は、自領域が持つ木構造からガイドセルを確定し，元の領域へ送信する．
- 元の領域側でガイドセルの木構造を確定し，ガイドセルの値をコピーする．

なお，自領域がガイドセルとして隣接領域の Cell を必要とするならば，当然，隣接領域は自領域の Cell をガイドセルとして必要するので，隣接領域同士はお互いを認識することが可能である．したがって，上記の隣接領域間の通信は，それぞれの領域が局所的にに決定した隣接領域との送受信で閉じることができる．

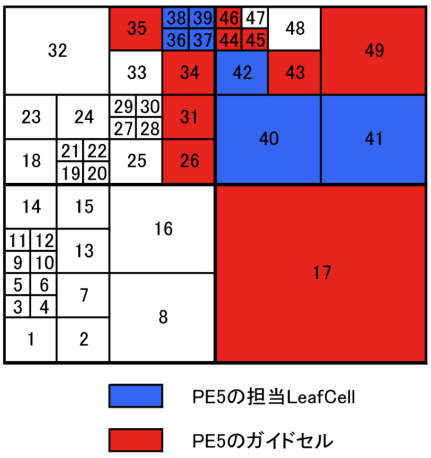

Fig. 2: PE5 の計算に必要なガイドセル．

3.3 ガイドセルの木構造の構築

ガイドセルが確定するので，このガイドセルに対する木構造を構築する．各領域の木構造は自領域で必要な部分のみを保持する．したがって、ガイドセルの木構造の構築処理で通信は発生しない．すでに自領域が保持していた木構造を基に，Split&Join に伴う木構造の追加削除を行う．この処理のため，Cell が持つフラグの 1bit をその Cell が必要な木構造の一部か否かのフラグ（木構造判定フラグ）として用いる．

3.4 ガイドセルの実体の通信

ガイドセルの実体の通信は，ガイドセルの通信箇所で通信する物理量の数に応じたバッファを用意し，データを送受信する．通信されるデータはパックされる．その通信量はガイドセル送信リストやガイドセル受信リスト，および物理量の数で決まる．一つのガイドセルの通信箇所においては，隣接領域とそれぞれ 1 回の送受信を行うのみである．

3.5 Split における通信

AMR における Split&Join は，各領域内部で局所的に処理できることが望ましい．領域分割の際に述べたとおり，Join の可能性がある LeafCell は同じ領域内に分割されるようにしているのはこのためである．Join の処理は隣接 Cell との格子のレベル差が 1 以下となるように制御する．自領域の木構造はガイドセルの設定時に既に確定しているので，隣接 Cell がガイドセルの場合でも，この Join の処理は通信なしで可能である．

Join と Split の処理においては，計算精度の点から Split 処理を優先している．このため，Split により隣接 Cell と格子のレベル差が 1 より大きくなるならば，隣接セルを Split する．すなわち，ある Cell で生じた Split は場合によっては，その周りの Cell へ伝播する．この伝播がガイドセルに達したとき，もはや自領域内で Split 処理は閉じることができず，通信して Split が伝播したことを隣接領域へ知らせる必要が生じる．また，隣接領域で Split が生じたことにより，自領域が持つガイドセル送受信リスト，および領域境界の LeafCell の Pedigree Index に変更が生じる可能性がある．

4. Migration

4.1 Migration の発生と処理手順

Migration は次の場合に生じる．

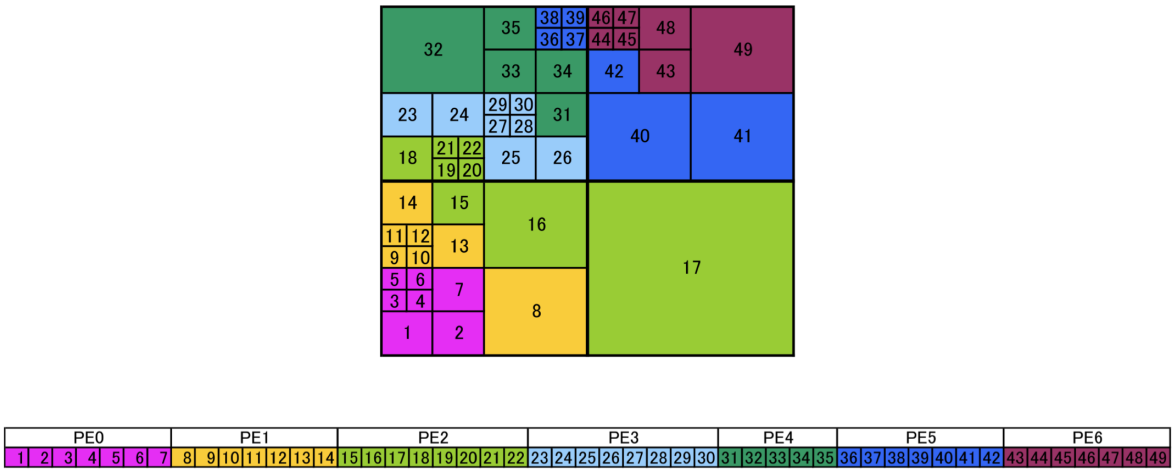

Fig. 1: 4 分木における領域分割．7 領域の分割で，色は各領域に対応する．

- Join 処理のルール
  AMR における Split&Join を各領域内部で局所的に処理するため，Join 処理を実行する場合「Oct が持つすべての Cell が LeafCell であるときにはこの Oct 内で領域境界は作らない」という条件を満足する必要がある．例えば，**図 2** において 44~47 が Join すると，42~48 が全て leafCell になるが，42 のみ PE5 で残りの LeafCell は PE6 に属するため，前述の条件を満足しなくなる．したがって，44~47 の Join はできない．この可能性を事前に検査するための付加的な処理が必要になる．

- 領域間のロードバランスが崩れた場合
  Split&Join が繰り返され，各領域が担当する LeafCell 数に偏りが発生すると，負荷を均一に分散する必要が生じる．

なお，Migration を行わない場合でも，Split&Join がおきれば，ガイドセルは設定しなおす．自己の担当 LeafCell のうちガイドセルに面した Cell が Split&Join している可能性やガイドセルが Split&Join している可能性があるためである．

Migration は，次のように 2 段階の通信により行う．

(1) 各領域が担当している LeafCell 数と領域境界における分断の可否フラグをマスターノードを通じて gather&scatter する．

(2) 送信された LeafCell 数を元に，各領域が Migration 後の領域境界を定める．

(3) 各領域は，領域境界の pedigree index と担当 LeafCell 数をマスターノードを通じて gather&scatter する．

(4) 各領域は，送信された LeafCell 数から、自領域がどの領域とどれだけデータ交換をする必要があるかを判断する．

(5) データ交換が必要な領域間で交換すべき LeafCell 数は決定しているので，LeafCell の pedigree index を送受信する．

(6) 受信した pedigree index から，ガイドセルの木構造の構築と同様に，必要となる LeafCell の木構造を追加構築する．

(7) 必要となる領域へデータを送受信してデータを交換する．

図 3 は，初期分布から Split&Join が生じ，Migration が発生する状況を示している．PE2 と PE3 では Split が生じ，LeafCell 数が増加する．また，PE5, PE6 では Join が生じ，LeafCell 数は減少する．上記の手順に従って，ロードバランスを均一化するように LeafCell の担当領域を決定し，必要に応じてデータの移動を行う．LeafCell の Migration プロセスを**図 4** に示す．結果として，PE2→PE3(5)，PE3→PE4(8)，PE4→PE5(5)，PE5→PE6(3) の Migration が発生する．括弧内は移動する LeafCell 数である．

4.2 領域番号の付け替え

前節の Migration 処理によって移動すべき LeafCell が確定するが，ある領域の過半数のデータを移動することになると，通信量が大幅に増加し並列性能の低下が問題となる．例えば，前述の**図 3** における Split&Join の結果，**図 4** に示したような通信が発生する．このとき，PE3 は自領域の全ての LeafCell を PE4 へ通信しなければならない．Migration 後の PE4 を PE3 と読み替えれば，この部分の通信は行わずにすむ．つまり，領域番号の付け替えが領域間のデータ転送量を低減できる．具体的には，MPI の通信空間における各領域のランク番号とは別に管理用の領域番号を保持し，この領域番号と MPI 通信空間のランク番号の関係を制御することにより，Migration 時の通信を最適化する．このため，領域番号からランク番号を参照するためのリストと，その逆にランク番号から領域番号を参照するためのリストの双方（ともに大きさは分割ノード数）をすべての領域が保持する．領域番号の付け替えを処理を以下に示す．

まず，Migration によるデータ交換に対して**図 5** に示す通信マトリックスを考える．通信マトリックスは行方向に Migration 前のマッピングリスト，つまり領域番号 (Domain number) とランク番号の関係，列方向に Migration 後のマッピングリストを示す．通信マトリックスの対角要素は，Migration 前後で同じ領域に属するので通信の必要がない．一方，i 行 j 列の非対角要素は，Migration 前に PE-i に属していたが，Migration により PE-j に属するようになった LeafCell 数を示す．この非対角要素は通信が必要である．なお，**図 5** には領域番号と MPI 空間のランク番号を ⇔ で対応付けして示している．Migration 前後で，この対応関係を確定するのが本処理の目的である．

初期状態として領域番号とランク番号は一致しているとする．本処理では通信量を最適にすればよいので，通

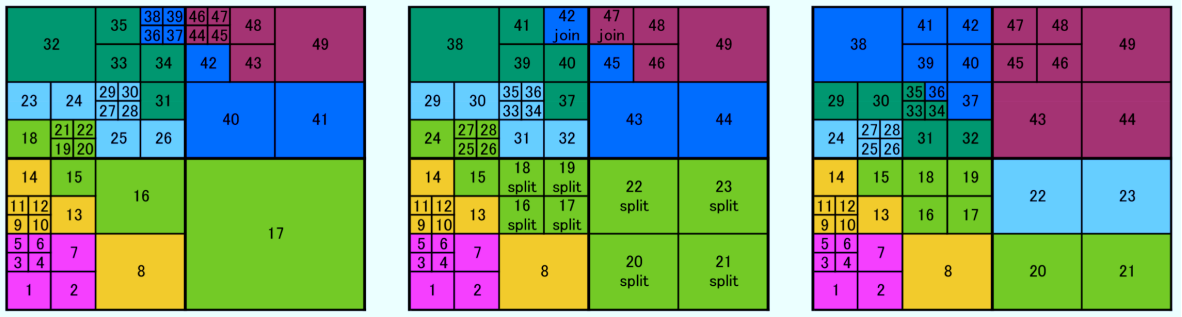

Fig. 3: Split&Join による木構造の変化．左から，初期，Split&Join の発生，Migration 処理後の状態を示す．

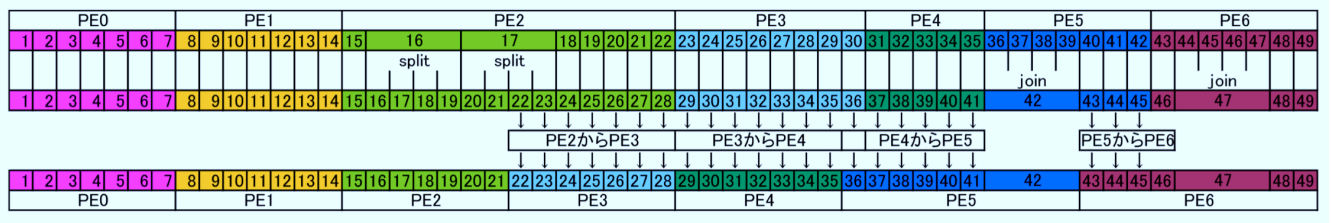

Fig. 4: Migration 処理のプロセス．

信マトリックスの非対角要素の大きいものを極力対角位置へ移動し，Migration 後の領域番号とランク番号の対応付けを変更する．

(1) 第 i 行の処理に対し，i 行 j 列 $(j \geq i)$ または j 行 i 列 $(j \geq i)$ のうち最大値を i 行 i 列に持ってくるように i 列と j 列の交換および i 行と j 行の交換を行う．このとき j 行 j 列の値の方が検索された最大値よりも大きな場合，行または列の交換で通信量が増えてしまうため交換しない．

(2) 行または列の交換に際して，領域番号とランク番号の対応付けも同時に交換する．

(3) この操作を通信マトリックスのランク分繰り返す．

通信マトリックスの初期状態を**図 5** に示す．2 行 3 列の要素 5 は PE2→PE3(5) を示す．i=0,1,2 はそれぞれ対角要素が最大なので行列の交換は起きない．しかし，i=3 において 3 行 3 列は 0 であるが，3 行 4 列が 8 なので第 3 列と第 4 列を交換する．このとき Migration 後の領域番号も 3 番目と 4 番目を交換する．

同様にして，第 4 列と第 5 列の交換，第 5 列と第 6 列の交換，第 5 行と第 6 行の交換の結果，マッピングリストが得られた．

最終的に，Migration 後の領域番号とランク番号を見て，PE4⇒PE3，PE5⇒PE4，PE3⇒PE5 のように領域番号を付け替える．これにより，通信量は**図 4** から**図 6** のように減少する．

この領域番号付け替えの処理は，Migration 時にマスターノードに各ノードから LeafCell 数が gather されたタイミングでマスターノードだけがこの処理を行い，得られた領域番号とランク番号の対応付けを全ノードに送信する．

		0	1	2	3	4	5	6
		0	1	2	**4**	**3**	5	6
0	0	7	0	0	0	0	0	0
1	1	0	7	0	0	0	0	0
2	2	0	0	9	0	**5**	0	0
3	3	0	0	0	**8**	0	0	0
4	4	0	0	0	0	0	5	0
5	5	0	0	0	0	0	1	3
6	6	0	0	0	0	0	0	4

		0	1	2	3	4	5	6
		0	1	2	4	5	**3**	**6**
0	0	7	0	0	0	0	0	0
1	1	0	7	0	0	0	0	0
2	2	0	0	9	0	0	0	5
3	3	0	0	0	8	0	0	0
4	4	0	0	0	0	5	0	0
5	6	0	0	0	0	0	**4**	0
6	5	0	0	0	0	**1**	**3**	0

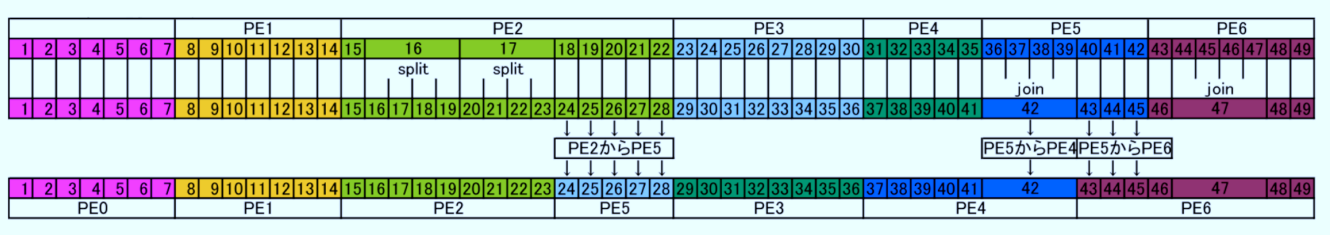

Fig. 6: 領域番号の付け替えによる Migration 時の転送量の削減.

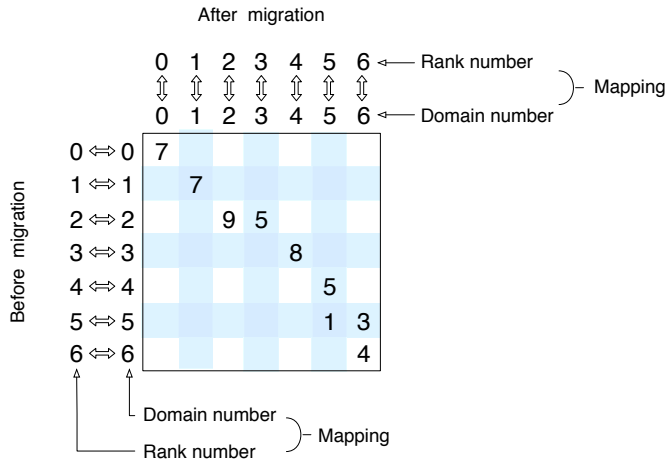

Fig. 5: 領域番号の付け替えに用いる通信マトリックス.

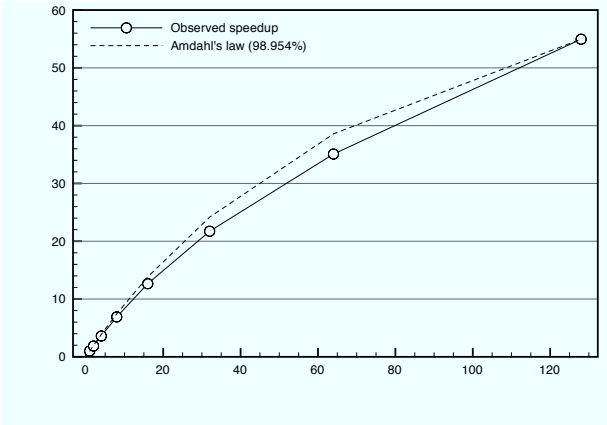

Fig. 7: QUEST システムにおける並列性能. 破線は, 並列可能部分が 98.954%である場合の Amdahl 則によるスピードアップを示す.

5. 計算結果とコスト分析

5.1 評価方法

以上のアルゴリズムに基づき, 並列 AMR のプログラムを開発し, $Re = 10^3$ の三次元直方体キャビティの問題を解いた[7]. 評価に利用した計算機は, 理化学研究所の QUEST システム (Intel Core2Duo@1.5GHz, 1024Nodes/2048Cores, GE) を用い, 128Core までの測定を実施した. 計算は t=12(15600step) まで実施した. 比較的滑らかな物理量分布をもつ非圧縮性流体の計算においては, 数ステップ毎の Split&Join 処理で効率が良いことが確認されているため, 10 ステップ毎に Split&Join を行った[8]. 計算結果は Guermond ら[7]の実験結果と良い一致が確認された.

5.2 並列性能

図 7 に 128Core までの実行時の性能を示す. ここで, Observed Speedup S_o は逐次実行時間を並列実行時間 (Elapse time) で割った値である. Observed Speedup は全ての並列化阻害要因を含んでいるので Amdahl 則による Speedup S_a よりも常に小さい値になる[9]. 128Core 測定時の S_o から計算した並列化率は 98.954%である.

5.3 計算コスト分析

図 8 に計算時間中で 60%以上を占める圧力反復部の計時データを示す. Pressure, PrsEvolute などの圧力の反復計算自体はほぼ一定であるが, 同期や通信, ガイドセルに関する部分のコストが並列数の増加とともに指数的に増えている.

Split&Join については, **図 9** に示すように, 処理その

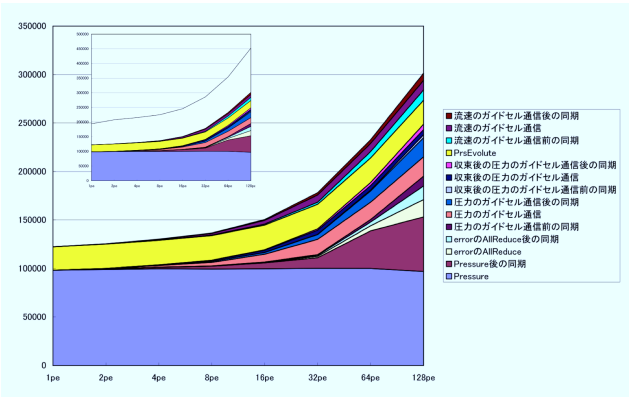

Fig. 8: 計算時間に占める各部分の割合とその並列数に対する変化. 小さな図の中の実線は全計算時間を示す.

ものの処理時間が PE 数の増加とともに増えている．Join に関する処理は各領域で局所的に処理するので領域数を増やしても計算時間は変わらないと考えられるが，若干ながら増加している．Join 後には Join したセルが他の領域のガイドセルになっていた場合に当該セルが Join したことを通信しているが，このガイドセル通信および通信後の同期が領域数の増大に伴い顕著なる．ただし，現状ではこの Split&Join に要する時間は全体の処理時間からみればさほど大きくはない．

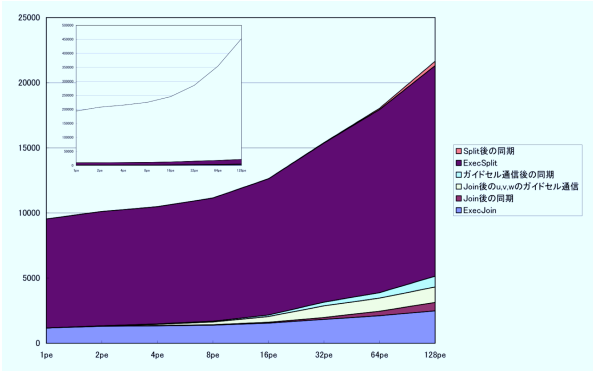

Fig. 9: Split&Join のコスト比較．

Migration のタイミング結果を**図 10** に示す．通信を含む Migration の処理そのものの処理時間は領域数の増加とともに増えてはいるが，それほど大きなコストではない．それよりも，ガイドセル作成に要する時間が領域数に対して指数関数的に増大する傾向が見られる点が問題である．ガイドセルの作成は，隣接領域とのガイドセル候補の通信のやりとりなどが含まれ，領域数の増大に伴い隣接となる領域も増えることが処理時間の増大につながっているものと考えられる．ただし，この Migration も現状の 10step に 1 回の処理では，全体の処理時間に比してさほど大きな割合にはなっていない．

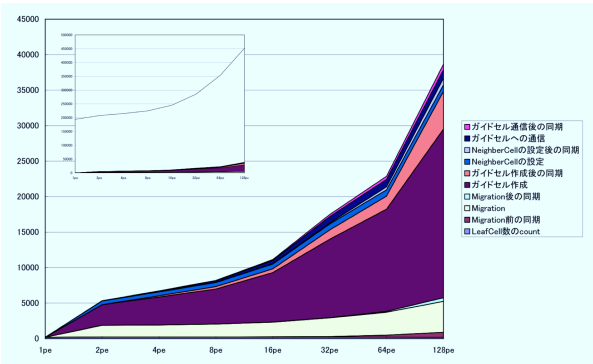

Fig. 10: Migration のコスト比較．

主要なコスト分析結果は，以下の通りである．

- 時間刻み Δt や圧力反復誤差の AllReduce 処理に処理時間がかかっている．特に，AllReduce は 128PE では 1PE の総計算時間の 16%程度になる．
- ガイドセル通信についても相応の処理時間がかかっている．計算本体でのガイドセルの通信では，128PE では 1PE の総計算時間の 47%になる．
- Split&Join や Migration について，その処理時間は全体の処理時間からみればそれほど大きくはない．ただし，Migration 後に行なわれるガイドセルの再設定に要する時間は PE 数の増大とともに指数関数的に増大する恐れがある．このガイドセルの再設定とそれに伴うガイドセルの通信に要する処理時間は，128PE では 1PE の総計算時間の 5%程度である．
- 検討が必要な項目として、Pressure 後の同期に異常に処理時間を要していることである．Pressure 後の同期は，LeafCell 数が同じならば PE の計算負荷が同じという仮定にたてば，ここに処理時間の差は生じないはずである．実装の問題点がある可能性がある．同様の状況は，その絶対的な処理時間が大きくないが各所で見受けられる．

6. まとめ

pFTT データ構造を用いて，三次元非定常非圧縮性流体を HSMAC 法を用いて解く並列 AMR コードを開発し，並列処理時のデータマイグレーション機構アルゴリズムの開発と実装を行い，その並列性能を評価した．マイグレーションのコストは相対的に低く，狙い通りの機能を果たしている．一方で，基本となるソルバー部分で同期とガイドセルに関する処理で改善の余地がある．本開発コードの 128 並列時のスピードアップは 55 倍，並列化率は 98.95%程度であった．

謝辞

本研究は，文部科学省 最先端・高性能汎用スーパーコンピュータの開発利用「次世代生命体統合シミュレーションソフトウェアの研究開発」の支援を受け実施された．

参考文献

1. 小林俊雄（編）. CFD ハンドブック, 第 10.4 章. 丸善, 東京, 2003.
2. M.J. Berger. Data structures for adaptive grid generation. *SIAM J. Sci. Stat. Comput.*, Vol. 7, No. 3, pp. 904–916, 1986.
3. M.J. Berger. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, Vol. 82, pp. 64–84, 1989.
4. C.W. Hirt, B.D. Nichols, and N.C. Romero. SOLA - a numerical solution algorithm for transient fluid flows. *Los Alamos Scientific Laboratory Report*, No. LA-5852, 1975.
5. A.M. Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. *Journal of Computational Physics*, Vol. 143, No. 3, pp. 519–543, 1998.
6. Masako Iwata, Kenji Ono, and Tsuyoshi Tamaki. Simulations with adaptive mesh refinement using pFTT data structure. In *The 10th ISGG Conference on Numerical Grid Generation*, Sep. 2007. Refereed.
7. J.-L. Guermond, C. Migeon, G. Pineau, and L. Quartapelle. Start-up flows in a three-dimensional rectangular driven cavity of aspect ratio 1:1:2 at Re = 1000. *Journal of Fluid Mechanics*, Vol. 450, pp. 169–199, 2002.
8. 岩田正子, 小野謙二, 玉木剛. pFTT データ構造によるオクツリー解適合格子を用いたシミュレーション. 第 20 回 数値流体力学シンポジウム講演論文集, No. E2-1, 2006.
9. Stefan Goedecker and Adolfy Hoisie. *Performance Optimization of Numerically Intensive Codes*. SIAM, 2001.