データ構造を考慮した実数型格子ガス法の並列計算の実装と評価

寺井優晃[†] 松澤照男[††]

実数型格子ガス法は非圧縮性流体解析の手法である．従来の粒子法と同様に微視的な粒子運動を用いて，巨視的な流体運動を格子点から得ることができる．その特徴から，アルゴリズム中に粒子と格子点がデータとして定義される．本稿では，格子点上でのみ粒子が衝突を起こすというスキームの抽象性を利用し，粒子法に特有の粒子データと格子点データの関連性を検討した．1つ目のデータ構造は粒子データを計算の主体ととらえ，粒子分割法を実装する．2つ目のデータ構造は格子点データを計算の主体ととらえ，領域分割法を実装する．キャビティを用いた計算の結果，格子点データをノード分散させた領域分割法を用いることによって，高い速度向上比が達成できた．これにより，これまで明確化されていない実数型格子ガス法に関する並列アルゴリズムとデータ構造の関係について示すことができた．

Data-structure Oriented Parallel Computation for Continuous-velocity Lattice-gas Model and Its Implementation and Evaluation

Masaaki Terai[†] and Teruo Matsuzawa[††]

The Continuous-velocity Lattice-gas model is an analysis approach to solve the incompressible flow. Just like the conventional particle method, this approach can obtain the macroscopic properties of fluid according to the motion of microscopic particles. This approach is featured with two physical objects as particle and lattice-point. In the present study, the authors investigated the relationship between the physical objects and the lattice-gas algorithm utilizing the abstraction of collision rule since particles collide only on lattice-points. Firstly, the data-structure with the particle as the primary calculation value was constructed with particle decomposite method. Secondly, the data-structure with the lattice-point as the primary calculation value was constructed with the domain decomposite method. As an experimental study, flow in cavty was simulated and high speed up ratio was obtained with domain decomposite method utilizing lattice data-structure. The relationship between the parallel-algorithm and data-structure in Continuous-velocity Lattice-gas model was shown.

1. はじめに

格子ガス法，格子ボルツマン法に代表される粒子法は，微視的な粒子運動を用いて，巨視的な流体運動を格子点から得る手法である．その特徴から，アルゴリズム中に粒子と格子点がデータとしてそれぞれ定義される．これまで，この手法に対する並列化についていくつか報告されている[1)~3)]．しかし，その粒子法に特有のデータ構造と並列アルゴリズムについての報告はされていない．一方，近年において粒子法の単純さを維持しつつ，従来の手法では困難な質量，運動量，エネルギー保存則をすべて満たすよう，現実的な流体運動の特徴を計算に反映させた，実数型格子ガス法[4)~7)]が提案された．しかし，実数型格子ガス法の並列化に関しては報告されていない．本稿では，実数型格子ガス法のアルゴリズムの特徴を考慮し，そのためのデータ構造と並列アルゴリズムを提案する．これらを用いて，各ノードへのデータの配置方法，およびデータの粒度が通信時間へ与える影響を調べ，実数型格子ガス法を用いた効率的な並列計算のためのデータ構造と並列アルゴリズムについて明らかにした．

粒子法は，粒子が並進と衝突を繰り返すことにより計算が進行する．連続体としての性質は，粒子の持つ物理量から求められる．また，粒子法は空間を離散化する．離散化によって格子が生成され，巨視的な物理量は格子点上でのみ観測されるという特徴を持つ．

† 北陸先端科学技術大学院大学情報科学研究科
School of Information Science, Japan Advanced Institute of Science and Technology (JAIST)
†† 北陸先端科学技術大学院大学情報科学センター
Center for Information Science, Japan Advanced Institute of Science and Technology (JAIST)

従来，数値流体に適用されてきた粒子法には，格子ガス法と格子ボルツマン法がある．しかし，現実の流体運動に比べると次の点で問題があった．まず格子ガス法は，並進の際の速度ベクトルに制限があり，FHPモデルなどでは隣接格子点にしか移動できない．そのため，非熱流体しか扱えない．また衝突ルールは有限であり，パターン化する際に繁雑になる．一方の格子ボルツマン法は，粒子数を実数で扱うことにより，衝突パターンの繁雑化をなくし，エネルギー保存則にも対応した．しかし，粒子の速度に制限があるため，外力などによる速度変化を微視的に適用することはできない．

これらの問題から，流体計算を行う必要条件である，質量，運動量，運動エネルギー保存則をすべて満たし，かつ外力の影響を微視的に考慮できる手法として，実数型格子ガス法が提案された[4)~7)]．実数型格子ガス法の特徴は，粒子の速度と座標を実数で表現できることである．この特徴から，粒子は速度ベクトルに応じて自由な方向への並進が可能となる．従来の手法にあった粒子の移動経路としての格子は必要なく，粒子を衝突するためのサイトとしての格子点だけを考慮する．そのため粒子間の衝突は近傍の格子点上で行われ，格子の形状に依存しない回転操作[6)] が適用可能である．

粒子法に共通した問題点は，多くの粒子および格子点を扱わなければ期待する流れ場を得られないという点である．高レイノルズ数ほど大規模な計算が必要になるが，現実として逐次計算機では記憶容量と計算能力が最大の障害となる．一方，これらの手法は計算方法が陽的である．また，衝突過程および並進過程の計算は格子点近傍で局所的に行われる．そのため各過程の計算を，並列化を用いることによって系全体で同時に計算することができ，実行時間の短縮が可能である．このような理由から，これまでの格子ガス法と格子ボルツマン法の並列計算では，共有メモリ型計算機および分散メモリ型計算機ともに良いパフォーマンスが得られている[1)~3)]．特に大規模計算のために広大なメモリ空間が必要な際には，分散メモリ型並列計算機が有効である．ただし，実数型格子ガス法に関しては比較的新しい手法であるため，並列化についての報告はされていない．

一方，並列化において，データ構造は各ノードへのデータの配置方法に影響を与え，さらに通信時間にも影響を与える．計算で扱うデータは，粒子の座標や速度を表す数値の羅列である．それら数値データとシミュレーションの物理的な性質を拘束するために，データ構造を用いる．しかし，これまでデータ構造が並列コードに用いられてきたにもかかわらず，粒子法に関する並列化とデータ構造の関係についての報告はされていない．よって，本稿で実数型格子ガス法を用いた効率的な並列計算のためのデータ構造と並列アルゴリズムを明らかにした．ここで得られたデータ構造とアルゴリズムの性質は，他の粒子法でも適用が可能であると期待する．

2. 実数型格子ガス法

2.1 概　　要

実数型格子ガス法の概要について述べる．実数型格子ガス法は，きわめて簡略化された多体系モデルを扱うことにより，微視的な立場から流れの性質を明らかにすることが可能な非圧縮性流体解析の手法である．計算は，並進過程と衝突過程を反復することにより進行する．粒子が持つ物理量は微視的なスケールの量であるから，連続体としての性質を持つ巨視的なスケールの物理量は，粗視化と呼ばれる平滑化処理によって得られる．なお以降の議論では，シミュレーションは2次元空間を対象としている．

2.2 衝 突 過 程

衝突過程は，粒子間の相互作用を考慮する過程である．この過程で，粒子は運動量と運動エネルギーを交換する．この際，質量，運動量，運動エネルギーは保存する．

式 (1) は，衝突過程で用いる式である．$\boldsymbol{v}_n$，$\boldsymbol{v}'_n$ を衝突前と衝突後の粒子の速度，$\boldsymbol{V}$ を格子点の粒子の平均速度とする．$\boldsymbol{\sigma}$ は回転行列であり，せん断粘性係数 ν が最も小さくなる条件[6)] を用いた．なお，粒子の質量は $m = 1$ に正規化した．

$$\boldsymbol{v}'_n = \boldsymbol{V} + \boldsymbol{\sigma}\,(\boldsymbol{v}_n - \boldsymbol{V}) \tag{1}$$

ここで粒子数密度 ρ の格子点における平均速度$\boldsymbol{V}$ は式 (2) によって与えられる[7)]．

$$\boldsymbol{V} = \frac{1}{\rho}\sum \boldsymbol{v}_n \ : \ (n = 1, \cdots, \rho) \tag{2}$$

レイノルズ数は，$\mathrm{Re} = L \cdot U/\nu$ で与えられる．ν は ρ と T の関数である．代表長さ L は1辺の格子点数に対応し，U は代表速度である．ここで，Re を大きくするには L と U を大きくし，ν を小さくする必要がある．ν を小さくするには，T を小さくするか，ρ を大きくする必要がある[5),6)]．

2.3 並 進 過 程

並進過程は，式 (3) に示されるように，各々の粒子の座標 $\boldsymbol{x}_n$ と速度 $\boldsymbol{v}_n$ に従い，新しい座標 $\boldsymbol{x}'_n$ を決定する過程である[6),7)]．並進過程では粒子の相互作用はいっさい考慮しない．

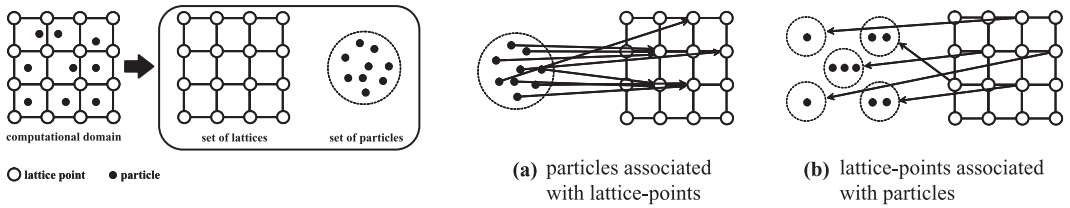


図 1 粒子と格子点の関連付け
Fig. 1 Association between particles and lattice-points.

$$\boldsymbol{x}'_n = \boldsymbol{x}_n + \boldsymbol{v}_n \quad (3)$$

速度 $\boldsymbol{v}_n$ は実数値であるため，並進後の座標 $\boldsymbol{x}'_n$ は実数値となり，粒子は格子点と格子点の間に配置される．次の衝突過程において，粒子は格子点上で衝突処理を行うため，この段階で規則[4),6)] に従って粒子を格子点上に移動させておく．

3. データ構造

実数型格子ガス法の特徴は，格子点は離散的に配置されるが，粒子は実数値を持って分布している．つまり，粒子は連続空間を並進するが，巨視的な物理量は配置された格子点でのみ観測することができる．この特徴から，計算をすすめるのに必要な物理量に関する情報を，粒子と格子点は互いに共有している．よって本稿では，粒子と格子点は，アルゴリズム中に直接定義される物理的な対象と位置づける．

3.1 データと物理量

粒子と格子点という 2 種類の物理的対象に含まれる情報を，実装ではデータとして記述する．ここで定義するデータに含まれる物理量を以下に示した．粒子データとは微視的な物理量を扱うデータ単位であり，格子点データとは巨視的な物理量を扱うデータ単位である．

- 粒子に固有の物理量 → 粒子データ
  - 座標 : $\boldsymbol{q}_P = (Real, Real)$
  - 速度 : $\boldsymbol{v}$
  - 質量 : m
- 格子点で観測される物理量 → 格子点データ
  - 座標 : $\boldsymbol{q}_L = (Integer, Integer)$
  - 平均速度 : $\boldsymbol{V}$
  - 粒子数密度 : ρ

3.2 データ間の関連付け

定義した 2 種類のデータを用いて，データ間の関連付けを明確にさせる．計算において，粒子と格子点が関連付けられるのは衝突過程と粗視化であるが，衝突過程はすべてのステップで実行される．よってここでは衝突過程を考慮した関連付けについて説明する．

関連付けには，粒子データから格子点データを関連付ける方法（**図 1** (a) 参照）と，格子点データから粒子データを関連付ける方法（図 1 (b) 参照）の 2 通りの方法がある．

図 1 (a) から説明する．この方法では，粒子データの座標 $\boldsymbol{q}_P$ に基づき，どの格子点上で衝突するかステップごとに判定する．つまり粒子データのリストを作成せずに，ステップごとの判定によって粒子データは格子点データに関連付けられる．格子点データは，ステップごとに生成される二次的な情報であり，格子点データのためのデータ構造は必要としない．粒子データをデータ構造に反映させたものを次に示した．

[データ構造（a)]

```
struct _dimension{
  double x;
  double y;
};
struct _particle{
  struct _dimension position;
  struct _dimension velocity;
};
```

まず，構造体 _dimension を定義する．次に，1 粒子についてのデータ（座標 *position*，速度 *velocity*）を定義する．定義した粒子データを構造体 _particle とする．粒子の質量は正規化したため，データ構造には m は実装していない．このデータ構造が，本計算手法にとって最小のデータの粒度になる．

実装では，この構造体から配列 *particle* を粒子数 N_P 個生成する．この配列を，各ノードに分散させたものを並列化手法 (a) とする．ノードへのデータ配置については 3.3 節で説明する．

次に，図 1 (b) について説明する．この方法では，粒子 $\boldsymbol{q}_P$ の近傍の格子点 $\boldsymbol{q}_L$ を判定し，格子点データの座標 $\boldsymbol{q}_L$ に基づき粒子データを振り分けておく．つまり，格子点上で衝突を考慮する粒子についてのリストを計算開始時に作成する．1 つの格子点はリストを 1 つ持つ．以上を，データ構造に反映させたものを次に

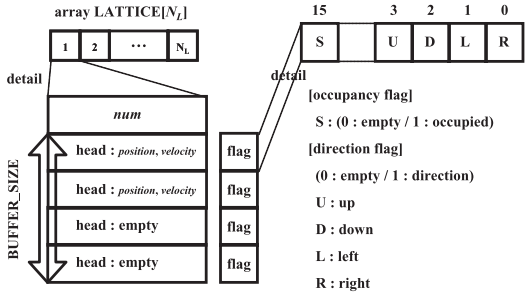


図 2 格子点データの構造
N_L は格子点数．$BUFFER\_SIZE$ は粒子データを格納するリストの大きさ．occupancy flag はリストの占有状態を示すフラグ．direction flag はノード間通信で用いるフラグ．

Fig. 2 Structure of lattice data.
N_L is number of lattice-points. $BUFFER\_SIZE$ is size of list in order to save particle data. Occupancy flag represents status of usage of list. Direction flag is used in comm.

示した．

[データ構造（b)]

```
struct _lattice{
  int num;
  struct _particle head[BUFFER_SIZE];
};
```

すでに定義してある構造体 _particle を用いて，1 格子点についての物理量を構造体 _lattice と定義する．粒子数密度は num に対応する．粒子データのリストは 1 次元配列 $head$ とする．3.1 節の定義では，平均速度$\boldsymbol{V}$ を用いたが，実際の計算では，式 (2) によって$\boldsymbol{V}$ を求めるための粒子データの配列 $head$ を構造体に格納する（**図 2** 参照）．これは，計算の進行に粒子データ単位が必要なためである．

本稿では，配列 $head$ は，計算開始時に静的に確保した．$BUFFER\_SIZE$ に関しては経験的に決定し，初期条件で与える ρ よりも十分に大きな値（$BUFFER\_SIZE = 100$）を用いた．

さらにここで粒子データの配列 $head$ とは別に，配列 $flag$ を導入した．$flag$ と $head$ は 1 対 1 で対応する．これにより，粒子の並進時に退避用の冗長領域を確保する必要がなくなり，メモリ資源の節約ができる．図 2 に示されるように，$flag$ の各ビットにフラグの意味を持たせる．フラグにはリスト中のブロックを占有しているか判断するための occupancy flag とノード間通信のために用いる direction flag が含まれる．フラグの使用に関しては 4 章で説明する．

実装では，この構造体から配列 $lattice$ を格子点数 N_L 個生成する．なお本稿では構造格子を用いたため，格子点データの座標 $\boldsymbol{q}_L$ は，配列 $lattice$ の添え字が

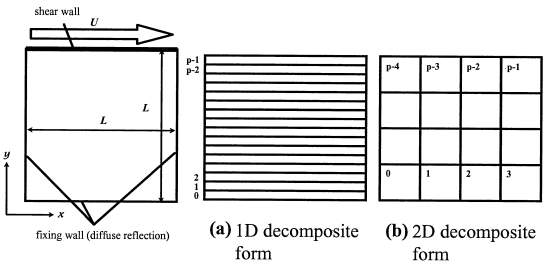


図 3 計算形状と領域分割形状
Fig. 3 Computation shape and domain decomposite form.

対応している．よって $\boldsymbol{q}_L$ は明示的には実装していない．この配列を，各ノードに分散させたものを並列化手法 (b) とした．データ配置については 3.3 節で説明する．

以上，データ間の関連付けをデータ構造を用いて示した．これらは並列化の際に，各ノードへのデータ配置の違いとして表れる．(a) の関連付けモデルの並列化は粒子分割法と同等になり，(b) の関連付けモデルの並列化は領域分割法と同等になる．

3.3 データの配置

粒子分割法では，配列 $particle$ を各ノードに対してブロック配置を行った．一方，領域分割法は，ドメイン形状によってデータの配置が異なる．本稿では長方形を用いる 1 次元分割（**図 3** (a) 参照）と，正方形を用いる 2 次元分割（図 3 (b) 参照）を実装した．1 次元分割ならば，配列 $lattice$ を各ノードに対してブロック配置をし，2 次元分割ならば，$N_L/\sqrt{p}$ 要素ごとにブロックサイクリック配置をした．p はノード数である．

4. 並列アルゴリズム

4.1 粒子分割法

ここでは，それぞれの並列化手法における，衝突過程と並進過程のアルゴリズムを示す．はじめに粒子分割法を用いた場合について示す．p はノード数とし，ノードあたりの粒子数を $N'_P = N_P/p$ とする．

なお，N_p は計算全体の粒子データの配列，$\boldsymbol{V}$ は格子点あたりの平均速度，$\boldsymbol{q}_p$ は粒子データ単位の座標，$\boldsymbol{q}_L$は格子点データ単位の座標 を示している（□は処理，■は繰返し処理，◇は判断を表す）．

[衝突過程]

□ $\boldsymbol{V}$ を格納するための配列を N_L 個確保する．
■ 粒子数 N'_p 回，以下の処理を繰り返す．
　□ 座標 $\boldsymbol{q}_P$ から衝突を考慮する座標 $\boldsymbol{q}_L$ を決定する．決定した座標に対応する配列要素を決定し，その要素に粒子データの速度 $\boldsymbol{v}$ を加

える．同時に，加えた粒子数も格子点ごとにカウントする．

□ 配列を全ノード間で重ねあわせる．

□ 得られた格子点ごとの速度の和とカウントした粒子数から，$\boldsymbol{V}$ を求める．

■ 粒子数 N_p' 回，以下の処理を繰り返す．

□ $\boldsymbol{V}$ を用いて，ノード内の全粒子に対して衝突後の速度 $\boldsymbol{v}'$ を設定する．

[並進過程]

■ 粒子数 N_P' 回，以下の処理を繰り返す．

□ 粒子の座標 $\boldsymbol{q}_P$ に $\boldsymbol{v}$ を加える．

□ 境界条件を考慮する．

4.2 領域分割法

次に領域分割法を用いた場合について示す．ノードあたりの格子点数を $N_L' = N_L/p$ とする．なお粒子を別の格子点へ移動する際は，$flag$ の操作も同時に行う．

[衝突過程]

■ 格子点数 N_L' 回，以下の処理を繰り返す．

◇ 格子点に関連付けられた粒子数が 2 個以上ならば，以下の処理を行う．

□ 格子点における $\boldsymbol{V}$ を求める．

□ 粒子に $\boldsymbol{v}'$ を設定する．

[並進過程]

■ 格子点数 N_L' 回，以下の処理を繰り返す．

■ 格子点に関連付けられた粒子について，以下の処理を繰り返す．

□ 粒子の座標 $\boldsymbol{q}_P$ に $\boldsymbol{v}$ を加える．

□ 境界条件を考慮する．

■ 格子点数 N_L' 回，以下の処理を繰り返す．

■ 格子点に関連付けられた粒子について，以下の処理を繰り返す．

◇ 自ノード内の格子点ならば次の処理を行う．

□ 粒子の新しい座標 $\boldsymbol{q}_P$ に基づき，適切な格子点データの $head$ に粒子をコピーする．コピーする $head$ の位置はリストの先頭から空き位置を occupancy flag を用いて線形探索する．その後にコピー元の粒子を削除する．

◇ 自ノード外の格子点ならば次の処理を行う．

□ 目標格子点のある方向を direction flag に設定する．

■ 以下の処理を，1 次元分割なら 2 方向の隣接ノードに対してそれぞれ p 回，2 次元分割ならば 8 方向の隣接ノードに対してそれぞれ $\sqrt{p}$ 回繰り返す．

□ direction flag の同じ粒子データを集め，まとめて送受信を行う．

□ 受け取った粒子が自ノード内の格子点宛てならば，適切な格子点に格納する．自ノード外の格子点宛てならば，新たに目標格子点のある方向を direction flag に設定する．

5. 結　　果

5.1 計算環境

分散メモリ型の並列計算機 CRAY T3E-1200E [8] 上に，C 言語を用いて実装した．コンパイルの最適化オプションには -O を用いた．また，ノード間通信には MPI を用いた．なお，T3E 特有の機構である Stream Buffer [9] は使用していない．

5.2 解析例

実数型格子ガス法の解析例は比較的少ない．計算結果の妥当性を示すために，キャビティ（図 3 参照）を用いた解析結果を**図 4** に示した．図 4 には，水平方向の速度成分 u，垂直方向の速度成分 v を示した．なお得られた結果は，各速度成分とも 0.5 で正規化して，直線の実線で結んである．いずれも Re=1000 であり，条件として代表長さ $L = 256$，粒子数密度 $\rho = 16$ を与えた．初期条件において，ρ は各格子点同じに設定し，系に与える温度 $T_0 = 1.5$，移動壁の速度 $U = 0.5$，壁境界の温度 $T_W = 1.5$ とした．境界条件は，cosine 法則に従う散乱反射の条件を適用した．なお粗視化は，30000 ステップ計算し，25000 ステップ以降について平滑化を行った．得られた結果を比較するために，差分法で求められた GHIA [10] の結果をプロットした．結果は定性的に再現できた．

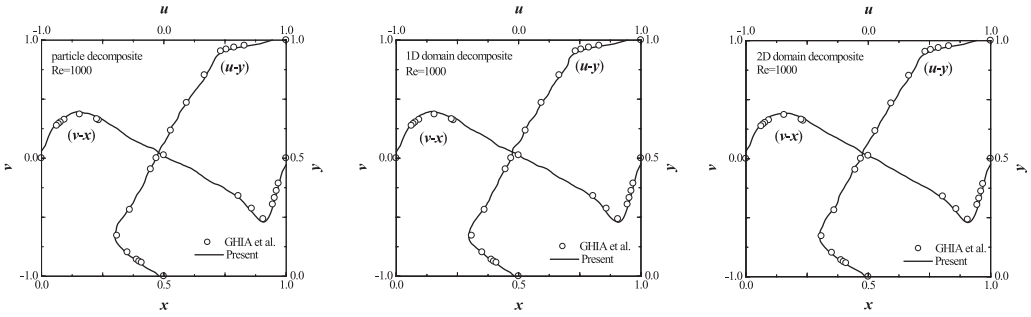
5.3 評価条件

実装コードのパフォーマンスの評価は，図 3 のキャビティについて，$L = 64, 128, 256$，$\rho = 4, 8, 16$ の組合せを用いて行った．他のパラメータに関しては 5.2 節に示したものと同じである．

ここで，1000 ステップの経過時間を $CPU\ time$ と定義した．ただし，実行には粗視化は含まれていない．また，通信時間には待ち時間が含まれている．なお，すべての評価において，逐次コードは用いていない．以上の条件でノード数 $p = 1, 4, 16, 64$ について計算を行った．

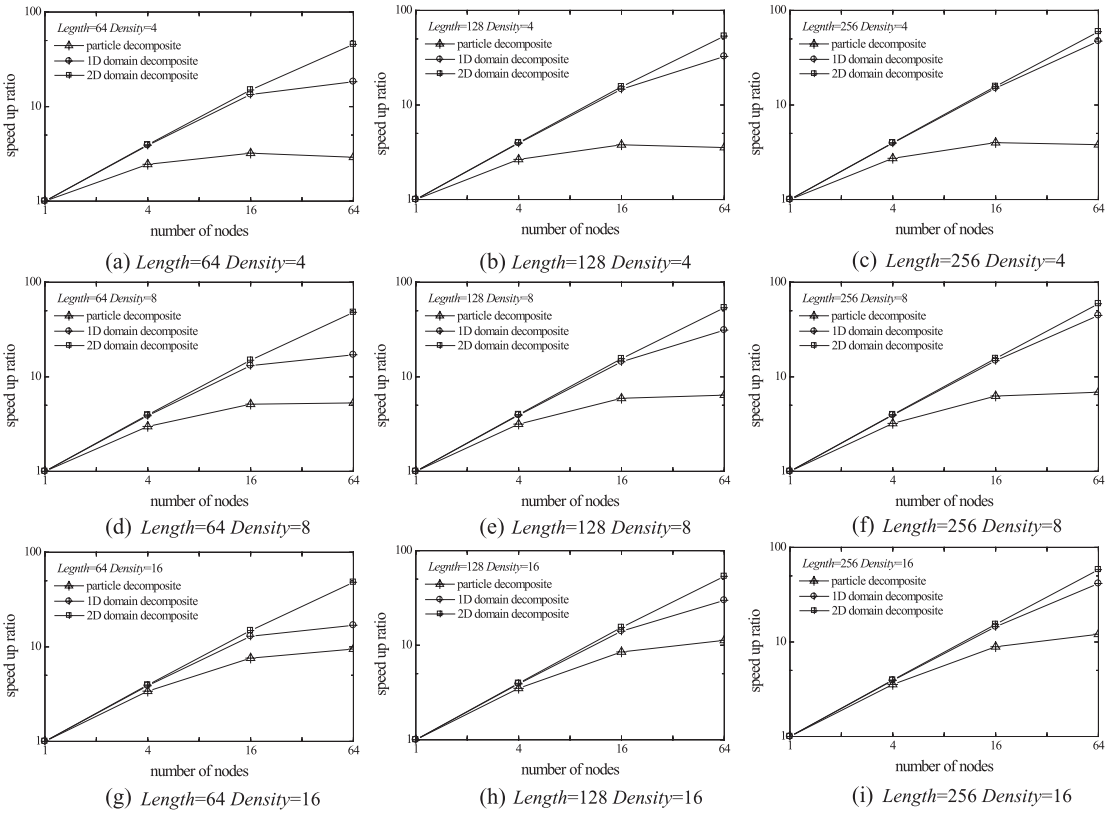
5.4 速度向上比

粒子分割法と領域分割法（1 次元分割および 2 次元分割）の速度向上比を**図 5** に示した．横軸はノード数，縦軸は速度向上比である．



(a) particle decomposite (b) 1D domain decomposite (c) 2D domain decomposite

図 4 Re=1000 の解析結果（64 ノード使用）
Fig. 4 Results of Re=1000 (apply 64 nodes).



(a) *Length*=64 *Density*=4 (b) *Length*=128 *Density*=4 (c) *Length*=256 *Density*=4
(d) *Length*=64 *Density*=8 (e) *Length*=128 *Density*=8 (f) *Length*=256 *Density*=8
(g) *Length*=64 *Density*=16 (h) *Length*=128 *Density*=16 (i) *Length*=256 *Density*=16

図 5 速度向上比
Length は代表長さ，*Density* は粒子数密度を表す．
Fig. 5 Speed up ratio.
Length represented length. *Density* represented density.

粒子分割法では，L，ρ のいずれの増加に対しても速度向上比は大きくなった．ただし，L よりも ρ の増加に対して，速度向上比は，より大きくなることが分かる．一方，図 5 (a)〜(d) において，16 ノードよりも 64 ノードの方が速度向上比は小さくなり，問題スケールに対するノード数の飽和が見られる．

1 次元分割を用いた領域分割法では，L と ρ のすべての条件において，ノード数の増加に対し，速度向上比は大きくなり，ノード数の飽和は見られない．また，速度向上比は，L の増加に対して大きくなったが，ρ

の増加に対しては若干小さくなる傾向を示した．

2 次元分割を用いた領域分割法でも，1 次元分割を用いた領域分割と同様に，L の増加に対して，速度向上比は大きくなったが，ρ の増加に対しては，速度向上比は若干小さくなる傾向を示した．なお，2 次元分割の領域分割法において，図 5 (c) は 59 倍，図 5 (f) と図 5 (i) は 58 倍であり，ρ に対する変化はほとんど見られなかった．また，ノード数の飽和もない．

すべての結果に共通して，粒子分割法よりも領域分割法の方が速度向上比が大きくなった．とくに 2 次元分割の領域分割法は，つねに 1 次元分割の領域分割法よりも速度向上比が大きく，ほぼ線形に高速化された．また，領域分割法に比べて，粒子分割法の方が，ノード数の飽和が早いことが分かる．粒子分割法の最大の速度向上比は $L = 256$，$\rho = 16$ の図 5 (i) における 12 倍であるのに対し，2 次元分割を用いた領域分割法では，$L = 256$，$\rho = 4$ の図 5 (c) における 59 倍が最大である．

5.5 実行時間の構成

1 ステップにおいて 1 粒子に要する処理の時間を式 (4) と定義した．p はノード数である．領域分割法では負荷分散を行っていないため，粒子数の偏りが考えられるが，ここでは 1 ノードが管理する粒子数は $L^2\rho/p$ と仮定する．条件より $step = 1000$ である．

$$update\ time = \frac{CPU\ time}{(L^2\rho/p) \times step} \qquad (4)$$

これを用いて**図 6** にノード数と各過程の処理時間の関係を示した．条件は，$L = 256$，$\rho = 16$ を用いた．実行時間は，粒子分割法および領域分割法のどちらも，衝突過程と並進過程の処理時間の和になっている．

図 6 から，粒子分割法における並進過程の処理時間は，ノード数に依存していないことが分かる．つまり粒子分割法の実行時間の増加は，衝突過程の処理時間の増加が原因であることが分かる．一方，領域分割法では，衝突過程の処理時間はノード数に関係なく一定であることが分かる．つまり領域分割法の実行時間の増加は，並進過程の処理時間の増加が原因であることが分かる．2 次元分割においても，実行時間の大半は並進過程の処理時間が原因であることが分かる．また，L と ρ を変化させたとしても本質的な傾向は同じであった．

5.6 計算時間と通信時間

計算時間と通信時間について，**図 7** に示した．計算時間 $calc$ は，実行時間 $CPU\ time$ と通信時間 $comm$ の差から求めた．なお，通信時間を測定するために設けた実行ステップおよび MPI 関数呼び出しの両方のオーバヘッドがあるため，通信が行われない 1 ノード使用時においても，計測される通信時間は厳密に 0[s] になっていない．

1 ノードについて，図 7 (a) の粒子分割法では，1000 ステップあたり 28[s] 消費しており，一方，図 7 (b) の 1 次元分割を用いた領域分割法では 0.08[s]，図 7 (c) の 2 次元分割を用いた領域分割法は 0.3[s] となっている．粒子分割法のオーバヘッドは，領域分割法に比べて大きいことが分かる．

図 7 (a) の粒子分割法では，ノード数の増加とともに，計算時間は一定の割合で減少していることが分かる．一方，通信時間は 4 ノードから 64 ノードに増やすと，92[s] から 233[s] まで増え，64 ノードでは通信時間が，実行時間の大半を占めていることが分かる．

図 7 (b) の 1 次元分割を用いた領域分割法でも同様に，計算時間は一定の割合で減少していることが分かる．通信時間は 4 ノードから 64 ノードに増やすと，1.5[s] から 29[s] に増えた．

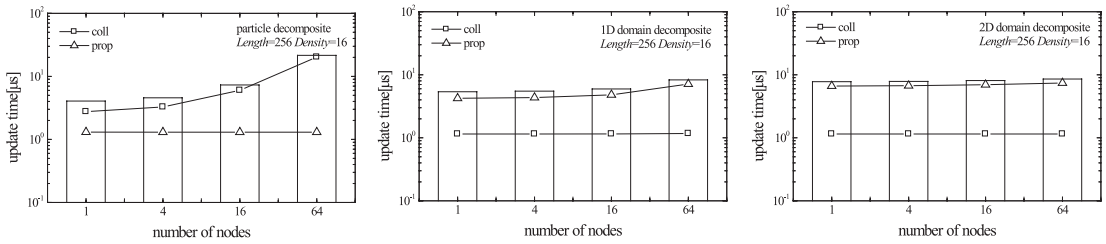
図 7 (c) の 2 次元分割を用いた領域分割法でも，計算時間の減少については，図 7 (a)，(b) ともに同じ傾向が得られた．一方，通信時間は 4 ノードから 64 ノードに増やすと，1.2[s] から 4.2[s] に増えた．通信時間の増加する割合は，図 7 (a)〜(c) の中で最も小さいことが分かる．

6. 考　　察

図 5 より，ノード数と問題スケールの関係を議論する．全体の傾向として，粒子分割法よりも領域分割法の方が明らかに速度向上比は大きく，また領域分割法の方が問題スケールに対するノード数の飽和は遅い．特に，1 次元分割よりも 2 次元分割の方が速度向上比が大きくなった．この結果から，2 次元分割を用いた領域分割法は大規模計算に適した方法である．

一方，粒子数密度 ρ の変化に対する速度向上比の変化は対照的である．粒子分割法では ρ が増加するにつれ，速度向上比は大きくなるが，領域分割法では，ρ の変化に対して，速度向上比は小さくなる傾向がある．

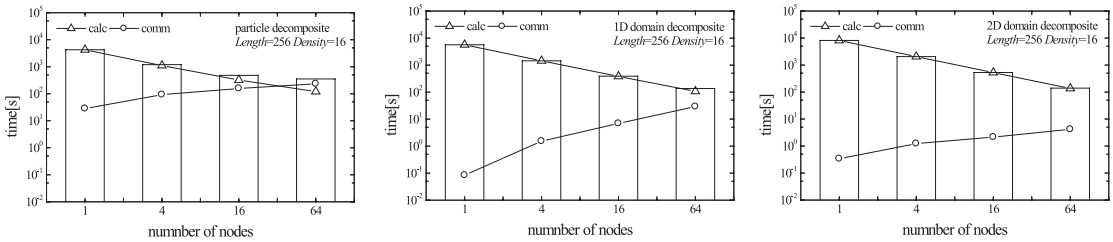
次に実行時間に占める衝突，並進過程の構成について議論する．図 6 より，粒子分割法では衝突過程，領域分割法では並進過程の処理に時間がかかっていることが分かる．これは，それぞれの並列アルゴリズムに固有の問題である．データ分割の方法が，一方の処理過程の高速化には有利に働くが，もう一方の処理過程には不利に働くことを示している．2 つの処理過程の高速化を同時に実現するデータ分割の難しさを示している．



(a) particle decomposite (b) 1D domain decomposite (c) 2D domain decomposite

図 6 1 ステップにおいて 1 粒子に費やされる各過程の処理時間（$L = 256$，$\rho = 16$）．
prop，coll はそれぞれ 1 ステップにおいて 1 粒子に費やされる並進過程と衝突過程の実行時間を表している．棒グラフは prop と coll の和を表している．

Fig. 6 CPU time per particle and timestep, in each process ($L = 256$, $\rho = 16$).
prop, coll was represented CPU time per particle and timestep, in propagation, collision process, respectively. Bar chart represented sum total of prop and coll.



(a) particle decomposite (b) 1D domain decomposite (c) 2D domain decomposite

図 7 計算時間と通信時間（$L = 256$，$\rho = 16$）．
calc，comm はそれぞれ計算時間と通信時間を示している．棒グラフは calc と comm の和を表している．

Fig. 7 Calculation time and communication time ($L = 256$, $\rho = 16$).
calc and comm represent calculation time and communication time, respectively. Bar chart represented sum of total of calc and comm.

また，ノード数の増加にともない，処理時間が増加している．これは，粒子分割法では衝突過程に通信時間が含まれ，領域分割法であると並進過程に通信時間が含まれていることが原因である．

次に，図 7 において示された結果から，実行時間に占める計算時間と通信時間の構成について議論する．計測された 1 ノード使用時の通信に要するオーバヘッドは，明らかに粒子分割法の方が大きい．アルゴリズムの構成から考えると，MPI 関数呼び出しの頻度は，粒子分割法よりも領域分割法の方が大きくなるのは自明である．しかし，領域分割法に用いられる 1 対 1 通信関数によるオーバヘッドは小さく，集団通信関数を用いた粒子分割法のオーバヘッドはきわめて大きいことに，並列化手法によって生じる通信時間の差の本質的な原因がある．

また，ノード数を 4 ノードから 64 ノードに増加させると，粒子分割法の通信時間は 2.5 倍，1 次元分割を用いた領域分割法では 20 倍に大きくなるのに対して，2 次元分割を用いた領域分割の通信時間は 3.5 倍に大きくなる．この点からも，2 次元分割の領域分割法は，通信に要するオーバヘッドが小さく，かつノード数に対する通信時間の増加も小さいことが分かる．これは，2 次元分割であると空間の細分化が進みにくく，分割方向に送信するデータの粒度が小さくなりにくいため，通信の回数も抑えられ，効率的な通信が行われているためと考える．

以上，アルゴリズムの構成から考察を述べたが，実数型格子ガス法の並列アルゴリズムは，分子動力学法で用いられるようなポテンシャルの近似[11)~13)]を用いない．そのため，粒子分割法と領域分割法の両方の解析結果に本質的な違いはない．結果が同じであれば，使用するノード数，計算規模にあわせた手法を用いるのが有効であると考える．特に大規模計算を行うのであれば，2 次元分割を用いた領域分割法は適していると考える．

ここで，粒子法に共通した性質と，実数型格子ガス

法に特有の性質を，データ構造とアルゴリズムの観点から考える．従来の粒子法と実数型格子ガス法に共通した点として，並進過程と衝突過程の分離がある．粒子の並進と，粒子間相互作用は同時には考慮しないという粒子法に特有のアルゴリズムの構成は共通している．また，最終的な物理量の算出は格子点から得られるため，粒子と格子点の関連付けも本質的には同じである．よって，それぞれの粒子法に用いられるデータ構造は，本稿で示した実数型格子ガス法のデータ構造を利用できると考える．ただし実数型格子ガス法は，従来の粒子法と異なり，格子点上の粒子数に制限がないため，一部の空間に粒子が偏る恐れがある．しかし今回の結果において，負荷分散された粒子分割法に比べて，負荷分散なしの領域分割法が安定して速度向上比を大きくできたのは，一部の空間に粒子がきわめて偏ってはいないためである．以上より，今回示したデータ構造を従来の粒子法に適用することは可能であると考え，類似した結果が得られることが期待される．

7. 結　　言

格子点上でのみ粒子が衝突を起こすというスキームの抽象性を利用し，データ構造を用いて，粒子データと格子点データの関連付けモデルを定義した．並列化においてはそれぞれのデータ構造から生成した配列を分割し，各ノードに分散させた．粒子データを分散させると粒子分割法になり，格子点データを分散させると領域分割法になった．粒子分割法を用いると負荷分散は行われるが，集団通信がともなうため大規模並列計算は難しい．一方，領域分割法を用いると，通信時間の増加を抑制し，実行時間の短縮が達成できた．2次元分割を用いた領域分割法はノード数が増加しても，高い速度向上比が維持できた．

謝辞 研究を進めるにあたり有益な資料の提供と助言をいただいた株式会社デンソーの今川洋造様，株式会社豊田自動織機の今村太郎様，本学博士後期課程の廣川雄一様に深く感謝いたします．

参 考 文 献

1) Bella, G., Filippone, S., Rossi, N. and Ubertini, S.: Using OpenMP on a Hydrodynamic Lattice-Boltzmann Code, *EWOMP2002* (2002).
2) Desplat, J.C., Pagonabarraga, I. and Bladon, P.: A parallel Lattice-Boltzmann code for complex fluids, *Computer Physics Communications*, Vol.134, pp.273–290 (2001).
3) Cappuccio, R., Cattaneo, G., Erbacci, G. and Jocher, U.: A parallel implementation of a cellular automata based model for coffee percolation, *Parallel Computing*, Vol.27, pp.685–717 (2001).
4) Malevanets, A. and Kapral, R.: Continuous-velocity lattice-gas model for fluid flow, *Europhys. Lett.*, Vol.44, No.5, pp.552–558 (1998).
5) Malevanets, A.: Statistical mechanics of hydrodynamics lattice gases, Thesis for the degree of Ph.D, Dept. of Chemistry, Univ. of Tronto (1997).
6) 橋本康弘，溝上伸也，陳　Yu，大橋弘忠：実数型格子ガス法を用いた3次元混相流解析，流体ミクロシミュレーションの大規模体系への適用研究，日本原子力学会報告書 (2000).
7) 井上康博，陳　Yu，大橋弘忠：実数格子ガス法による非ニュートン流体流れのシミュレーション，計算工学会講演会論文集，Vol.5, pp.827–830 (2000).
8) http://www.jaist.ac.jp/iscenter/mpc/t3e/
9) Anderson, E., Brooks, J. and Hewitt, T.: *The Benchmarker's Guide to Single-processor Optimization for CRAY T3E Systems*, Cray Research (1997).
10) Ghia, U., Ghia, K.N. and Shin, C.T.: High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method, *Journal of Computational Physics*, Vol.48, pp.387–411 (1982).
11) 清水大志，君塚　肇，蕪木英雄，荒川忠一：並列分子動力学ステンシルの開発，計算工学会論文集，Vol.4, No.22, pp.225–230 (2002).
12) 松原正純，板倉憲一，朴　泰祐：超並列計算機CP-PACSにおける大規模分子動力学法シミュレーション，情報処理学会論文誌，Vol.40, No.5, pp.2172–2181 (1999).
13) Beazley, D.M. and Lomdahl, P.S.: Message-passing multi-cell molecular dynamics on the Connection Machine Vol.5, *Parallel Computing*, Vol.20, pp.173–195 (1994).

(平成15年10月10日受付)
(平成16年 2 月 8 日採録)

寺井　優晃

1978年生．2001年岡山大学理学部卒業．2003年北陸先端科学技術大学院大学博士前期課程（情報科学）修了．現在同大学院博士後期課程（情報科学）在籍．超並列計算機上での数値流体解析に関する研究に従事．計算工学会会員．

松澤　照男（正会員）

1948 年生．1973 年信州大学大学院工学研究科修士課程修了．同年同大学医学部助手．1986 年沼津工業高等専門学校助教授．1991 年北陸先端科学技術大学院大学情報科学センター助教授．1995 年同教授．現在に至る．数値流体力学における並列計算の研究に従事．医学博士．日本機械学会，日本流体力学会各会員．