JAERI-Data/Code
97-056

非浸透格子ガスモデルによる二相流シミュレーションコードの開発並びに並列化

1998年1月

渡辺　正・海老原健一・蕪木英雄

日本原子力研究所
Japan Atomic Energy Research Institute

本レポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所研究情報部研究情報課（〒319-1195　茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-1195　茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, 319-1195, Japan.

編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

JAERI-Data/Code 97-056

非浸透格子ガスモデルによる二相流シミュレーションコードの開発並びに並列化

日本原子力研究所計算科学技術推進センター
渡辺　正・海老原健一・蕪木　英雄

(1997年12月2日受理)

非浸透格子ガスモデルによる二次元二相流シミュレーションコードを開発し、ベクトル並列型スーパーコンピュータＶＰＰ500並びにスカラー並列型計算サーバＡＰ3000において並列化を行った。並列処理にはＶＰＰ500ではＶＰＰ　ＦＯＲＴＲＡＮのユーザ指示行を使用し、ＡＰ3000ではＭＰＩライブラリを使用した。ＶＰＰ500 においては、プロセッサ間のデータ転送に関わる処理が高速に行われるため、計算処理の並列化のみにより実用上十分な並列化効率が得られた。一方、ＡＰ3000では計算処理の並列化と同時にデータ分割の必要性が示された。また、ＡＰ3000における並列処理にはＶＰＰ　ＦＯＲＴＲＡＮよりもＭＰＩを使用する方が効率的であることがわかった。

日本原子力研究所（東海駐在）：〒319－1195 茨城県那珂郡東海村白方白根２－４

JAERI-Data/Code 97-056

Development of a Two-phase Flow Simulation Code Using the Immiscible Lattice-gas Model and Its Parallelization

Tadashi WATANABE, Kenichi EBIHARA and Hideo KABURAKI

Center for Promotion of Computational Science and Engineering
(Tokai Site)
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

(Received December 2, 1997)

A two-dimensional two-phase flow simulation code using the immiscible lattice-gas model is developed and parallelized on the vector-parallel super computer VPP500 and on the scalar parallel server AP3000. The user directive of VPP FORTRAN and the MPI library are used for parallelization on VPP500 and on AP 3000, respectively. Reasonable efficiency of parallelization is obtained on VPP500 by procedure decomposition alone, since the band-width of node communication is very high. It is shown for AP3000 that not only procedure decomposition but also data decomposition is necessary because AP3000's bandwidth is much lower. MPI is found to be more effective than VPP FORTRAN for parallelization on AP3000.

Keywords: Immiscible Lattice-gas Model, Two-phase Flow, Parallelization, VPP500, VPP FORTRAN, AP3000, MPI

目　　次

Contents

1. はじめに

日本原子力研究所計算科学技術推進センターでは、原子力分野で見られる複雑現象として、代表的な伝熱・流動現象をとりあげ、その基礎過程を計算機シミュレーションを通して解明する研究を進めている。流動現象の解析は、通常、ナビエ－ストークス方程式に代表されるマクロな連続体の微分方程式を数値的に解くことによって行われている。しかしながら、混相流や多成分系の流動現象、あるいは複雑な流路内の流れなどの解析にあたっては、基礎方程式や数値解法に何らかの変更・仮定を施す必要がある [1]。一方、熱流動現象は、多数の流体の原子・分子、あるいは流体の微小な塊としての流体粒子（流体要素）による集団的な運動であると考え、粒子の運動を追跡しその統計的な平均値として流れ場の温度、速度などの流動状態を求める粒子法と呼ばれる数値解法がある。粒子法には、ミクロなレベルで原子・分子の運動を扱う直接シミュレーションモンテカルロ法 [2] や、分子動力学法 [3]、メソスコピックないしはマクロなレベルで流体粒子を扱う格子ガス系の手法 [4]、スムースパーティクル系の手法 [5] 等がある。これらの手法は、分子運動に起因する統計的な変動を平均的な流れ場と同時に求めたり、複雑な流路を模擬したり相界面の形状変化を記述するのに優れているが、逆に、変動の少ない定常的な流れ場を再現するためには統計平均数を多くする必要があり、一般に多数の粒子が必要となる。近年、計算機の進展にともなって、多数の粒子を現実的な計算時間で扱うことが可能となってきており、今後、粒子法の様々な流動現象への適用が期待されている [6]。

本報告書では、格子ガス系の手法の一つである非浸透 (immiscible) 格子ガスモデルによる二次元二相流シミュレーションコードの開発と並列化について記述する。シミュレーションコードは FORTRAN77 言語により作成し、原研東海研究所に導入されている富士通製ベクトル並列型スーパーコンピュータ VPP500 及びスカラー並列型計算サーバ AP3000 上で並列化を行った。並列処理には VPP500 においては VPP FORTRAN のユーザ指示行、AP3000 においては MPI ライブラリを使用した。ただし、AP3000 上でも VPP FORTRAN が使用できるため，AP3000 上における VPP FORTRAN と MPI との比較を行った。シミュレーションコードは、周期境界条件及び固体壁境界条件に対応しており、さらに、外力の効果を考慮することができる。ここでは、周期境界条件の計算領域において、外力の下で運動する気泡のシミュレーションを例にとり、並列化効率について検討する。

2. 非浸透格子ガスモデル

非浸透格子ガスモデルは、二種類の混合しない流体の挙動をシミュレーションするために、格子ガスセルラオートマトン法に基づいて提唱されたモデルである [7]。基礎となる格子ガスセルラオートマトン法は、Fig.2.1 に示すような離散的空間である三角格子上に、離散的な速度を持った粒子を配置し、ある定められた移動、衝突規則に従って、離散的な時間段階における粒子位置の発展を追うことによって流体現象をモデル化したものである。格子ガスセルラオートマトン法の詳細、粒子の衝突規則からの流体方程式の導出等は参考文献 [8] を参照のこと。ここでは、以下に非浸透格子ガスモデルの概略を説明する。

非浸透格子ガスモデルは、格子ガスセルラオートマトン法において二種類の粒子を使用することにより、混合しない二種類の流体の挙動を模擬できるようにしたものである。二次元の場合、物理空間は Fig.2.1 に示すような離散的な三角格子で表され、流体粒子が格子上に配置される。流体粒子は離散的な速度を持ち、計算の 1 タイムステップで 1 格子単位だけ移動することができる。すなわち、Fig.2.2 に示すように、粒子は隣の格子点の方向へ六通りの離散的な速度を持つことができる。静止粒子を考慮する場合は、速度ゼロの粒子も格子点上に存在できることにする。ある格子点上に複数の粒子がある場合、粒子はある衝突規則にしたがってその格子点上で配置を変化させる。粒子の衝突規則の代表的なものとしては、速度ゼロの粒子を含まず、格子点において粒子数と運動量を保存し、かつ等方的である衝突の最低限の可能性だけを考慮した FHP-I モデル [9]、速度ゼロの粒子を含みすべての可能性を考慮した FHP-III モデル [10]、その中間の FHP-II モデル [10] などが提案されている。この三種類の衝突規則を Fig.2.3 にまとめて示す [11]。Fig.2.3 において、N はその状態の粒子数、x と y は、それぞれ x 方向の運動量を 2 倍したもの、y 方向の運動量を $4/\sqrt{3}$ 倍したものである。衝突規則が適用できない格子点の状態は”／”、衝突規則を適用できるが考慮しない格子点の状態を” No ”、衝突規則を適用する格子点の状態を”○”で表してある。”○”を結ぶ”⟷”は、それらの状態間で遷移が起こることを表している。状態を表す pattern の欄の数字は、格子点を中心として、$\pi/3$ または $\pi/6$ 回転して得られる状態の数である。ここで、衝突によって遷移可能な状態数が 3 の場合、一つの状態は、1/2 の確率で他のどちらか一つの状態に必ず遷移する。状態数 2 の場合は一つの状態は、必ず他の状態へ遷移する。ここまでの粒子の衝突と移動に関する規則は格子ガスセルラオートマトン法のものと同一である。

非浸透格子ガスモデルでは、粒子に区別を付け、例えば赤と青の二種類の粒子を考える。それぞれの粒子には、格子ガスセルラオートマトン法における移動、衝突の他に、同種の粒子同士が引き合う効果を考慮する。この効果を表すために、色のフラックス（流束）とフィールド（場）という概念を導入する。まず、ある格子点、x、において、赤粒子の存在を

$$r(x) = \{r_i(x) \in \{0,1\}, i = 0,1,...,6\}, \tag{2.1}$$

により定義する。$r_i(x)$ は、速度 c_i の粒子が格子点上に存在するかどうかを 1 または 0 で表すブール変数であり、i は Fig.2.1 に示すような三角格子上の一つの格子点及びそこから隣の格子点へ向かう六方向を表すものとする。青粒子の存在も同様に定義する。赤と青の粒子は、同時にある格子点上に複数存在することが可能であるが、同じ速度を持つことはできないものとする。

これは、格子ガスセルラオートマトン法における排他原理と同じである。相分離は、粒子の移動が隣接格子の粒子配置に影響されることにより促進される。隣接格子の粒子配置の影響を表すために、色のフラックスを赤と青の粒子の運動量の差により

$$q[r(x), b(x)] = \sum_i c_i[r_i(x) - b_i(x)], \tag{2.2}$$

と定義する。また、局所の色のフィールドとして、隣接格子の赤と青の粒子数の差の総和をとったもの

$$f(x) = \sum_i c_i \sum_j [r_j(x + c_i) - b_j(x + c_i)], \tag{2.3}$$

を定義する。これらにより、色のフィールドに対して色のフラックスが行う仕事として

$$W(r, b) = -f \cdot q(r, b), \tag{2.4}$$

が得られる。格子点上の粒子配置は、この仕事量が最小になるように決定される。すなわち、フィールドの影響による粒子配置の変化、$r \to r'$、及び、$b \to b'$ は、

$$W(r', b') = minW(r, b), \tag{2.5}$$

を満足するように決定される。ここで、それぞれの色の粒子に対する質量保存

$$\sum_i r_i = \sum_i r'_i, \qquad \sum_i b_i = \sum_i b'_i, \tag{2.6}$$

および、全運動量保存

$$\sum_i c_i(r_i + b_i) = \sum_i c_i(r'_i + b'_i), \tag{2.7}$$

が満足される。シミュレーションにおいては、格子ガスセルオートマトンにおける粒子の移動、衝突の操作を行い運動量の配置を確定した後、色の配置を上述の方法にしたがって決定することになる。同種の色同士が集まる方向に移動するため二相の分離が表現され、分離した相の間には表面張力による力の釣合が成り立つことが知られている [7, 12]。

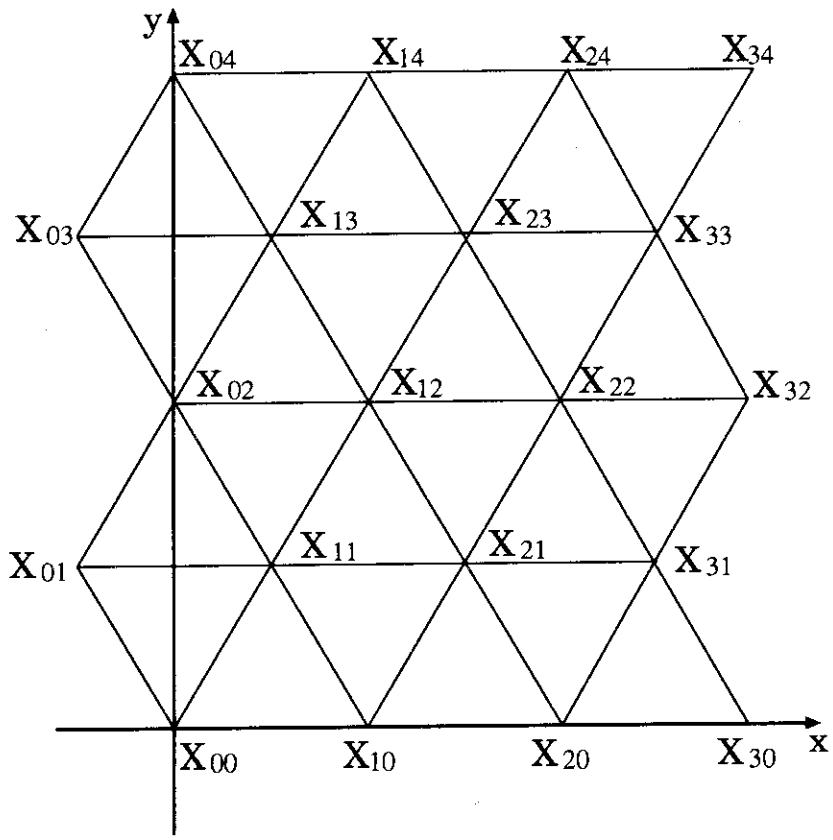

Fig. 2.1 Lattice structure

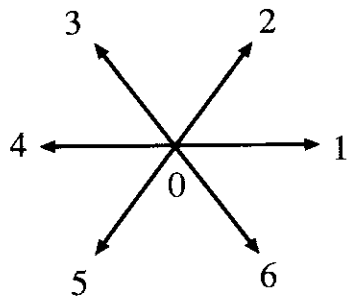

Fig. 2.2 Velocity direction at one site

N	x	y	Pattern	I	II	III	N	x	y	Pattern	I	II	III
0	0	0	1	No	No	No	7	0	0	1		No	No
1	0	0	1		No	No	6	0	0	1	No	No	No
1	2	0	6	No	No	No	6	2	0	6		No	No
2	0	0	3	○	○	○	5	0	0	3		No	○
2	2	0	6 6	 No	○ ○	○ ○	5	2	0	6 6	No 	No No	○ ○
2	3	2	6	No	No	No	5	3	2	6		No	No
3	0	0	3 2	 ○	○ ○	○ ○	4	0	0	3 2	No 	No ○	○ ○
3	2	0	6 6 6	 No No	No No No	○ ○ ○	4	2	0	6 6 6	No 	No No No	○ ○ ○
3	3	2	6		No	No	4	3	2	6	No	No	No
3	4	0	6	No	No	No	4	4	0	6		No	No

Fig. 2.3 Collision rules of FHP models

3. 二相流シミュレーションコード

ここでは、前章に記述した手法により二相流のシミュレーションを行うコードの概要について述べる。オリジナルコード作成においては、特にベクトル化、並列化を意識してはいない。すなわち、作業手順としては、既にワークステーション上などで作成され、利用されているシミュレーションコードの並列計算機あるいはワークステーションクラスターへの移植の場合と同様である。

3.1 概要

メインプログラムにおける処理の概略を以下に示す。

```
      call cdread
      call iniset
      call vramfi
      call addpar
      call rflin
      do 300 k=1,nit
      do 200 i=1,nst
      do 100 j=1,nis
      call simula
  100 continue
      call sample
  200 continue
      call propff
  300 continue
      call rflout
```

まず、cdread で格子数などの計算条件の入力を行い、iniset において配列や変数の初期設定を行う。vramfi は使用する乱数の初期化を行うものである。addpar では設定された格子上に粒子を配置する。ここでは、運動量、色に関して、任意の設定を行うことができる。rflin は、リスタート計算の際にファイルを読み込むものである。実際の非浸透格子ガスモデルによるシミュレーションは simula で実行する。格子ガスセルラオートマトン法では、モンテカルロ法や分子動力学法と同様に [13]、粒子情報の統計平均からマクロな物理量を算出する。このため、nis ステップごとの粒子のサンプリングを sample で行い、速度、密度などの物理量を nst ステップごとに propff において算出し、一つの流れ場とする。流れ場は、全部で nit 回計算する。rflout はリスタート計算用の出力を行うものである。

非浸透格子ガスモデルのシミュレーションを行なう simula では、以下の五つのステップを実行する。

```
call move
call cyclic
call wall
call collis
call force
```

move は粒子の移動、cyclic は計算領域の左右境界における境界条件の設定、wall は上下境界における境界条件の設定、collis は粒子の衝突及び色の配置、force は外力の効果を計算するものである。上下壁の間の水平層状流やポアズイユ流れ、あるいは上下壁に温度差をつけたレイリーベナール対流のシミュレーションを想定しているため、上下の境界条件を設定するルーチン名を wall、左右の境界条件を設定するルーチン名を cyclic としてあるが、固体壁条件、周期境界条件、いずれの設定も可能である。また、壁に平行な方向の運動量を与えることにより、キャビティ流れのシミュレーションを行うことも可能である。粒子の衝突規則としては、Fig.2.3 に示す FHP-I モデルから FHP-III までの三つのうちのいずれかを選択できる。外力の影響は、粒子速度の方向をある確率で力の向きの方向に強制的に向けることにより考慮する。ただし、格子点上で既にその向きの方向の粒子が存在する場合はこの操作は行わない。

3.2 上昇気泡のシミュレーション

シミュレーションコードの実行例として外力の影響のもとに上昇する気泡のシミュレーションを示す。二次元の計算領域は $x \times y = 256 \times 512$ の格子で表し、初期条件として 674160 個の粒子をランダムに配置する。粒子の衝突には、FHP-III モデルを用いる。FHP-III モデルでは、一つの格子は速度の異なる粒子を 7 つまで持つことができるため、無次元の粒子数密度は、$674160/256/512/7 = 0.73$ となる。すなわち、一つの格子点上には、平均として約 5 個の粒子が存在する。このうち、$(x, y) = (128, 205)$ の格子位置を中心として半径 102 格子長さの円形領域内の粒子は赤粒子とし、残りを青粒子とする。外力は赤粒子に対して上向きに 1/800、青粒子に対して下向きに 1/10000 とした。境界条件は、上下左右とも周期境界条件とした。計算領域は 32×64 の統計平均用のセルにわけ、連続する 100 計算ステップにおける粒子配置のセル平均により、マクロな流れ場の変数を求める。一回のシミュレーションではマクロな流れ場を 100 回計算するものとする。すなわち、100×100 の計算ステップを実行することにする。

Fig.3.1 ～ 3.4 に流れ場の過渡変化の様子を示す。流れ場の可視化は、汎用可視化ツール AVS [14] を用いたリアルタイムモニターシステム [15] により行った。リアルタイムモニターシステムは、スーパーコンピュータや計算サーバワークステーションで実行中のシミュレーションの経過を実時間で可視化し、モニターするシステムである。Fig.3.1 は，1 番目の流れ場（1 ～ 100 計算ステップの平均）、Fig.3.2 は 20 番目（1901 ～ 2000 計算ステップの平均）、Fig.3.3 は 33 番目 (3201 ～ 3300 計算ステップの平均）、Fig.3.4 は 50 番目 (4901 ～ 5000 計算ステップの平

均）の流れ場を示している。図はモノクロにして印刷したものであるが、赤青それぞれの領域を色分けで表し、白色の流線により流れの様子を示している。赤粒子により表される気泡が変形しながら上昇し、気泡両側に渦が発生する様子が見られる。ここでは、流れ場の流速を得るために統計平均を行っているため、気泡表面の形状はそれほど明確に示されてはいない。Fig.3.1 と 3.3 に対応した二相界面の形状を Fig.3.5 と 3.6 に示す。ここでは統計平均をとらずに、1 計算ステップ目と 3300 計算ステップ目の粒子配置をそのまま表示している。統計平均を行わない瞬時の流れ場においては、二相の界面形状が明瞭にとらえられることがわかる。

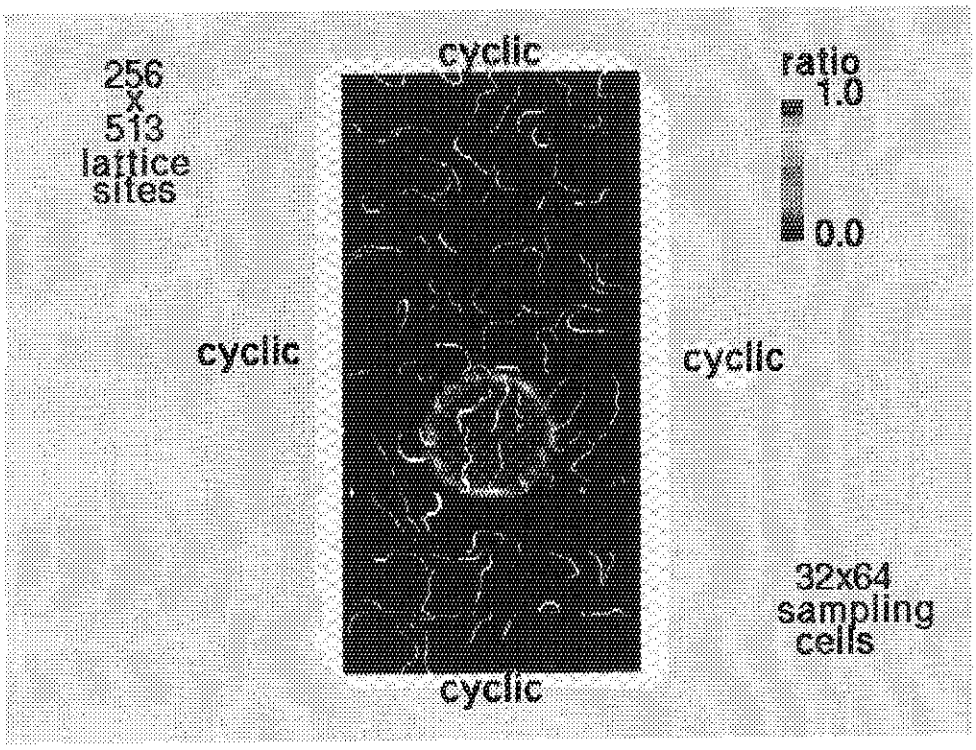

Fig. 3.1 Rising bubble simulation : 1st flow field

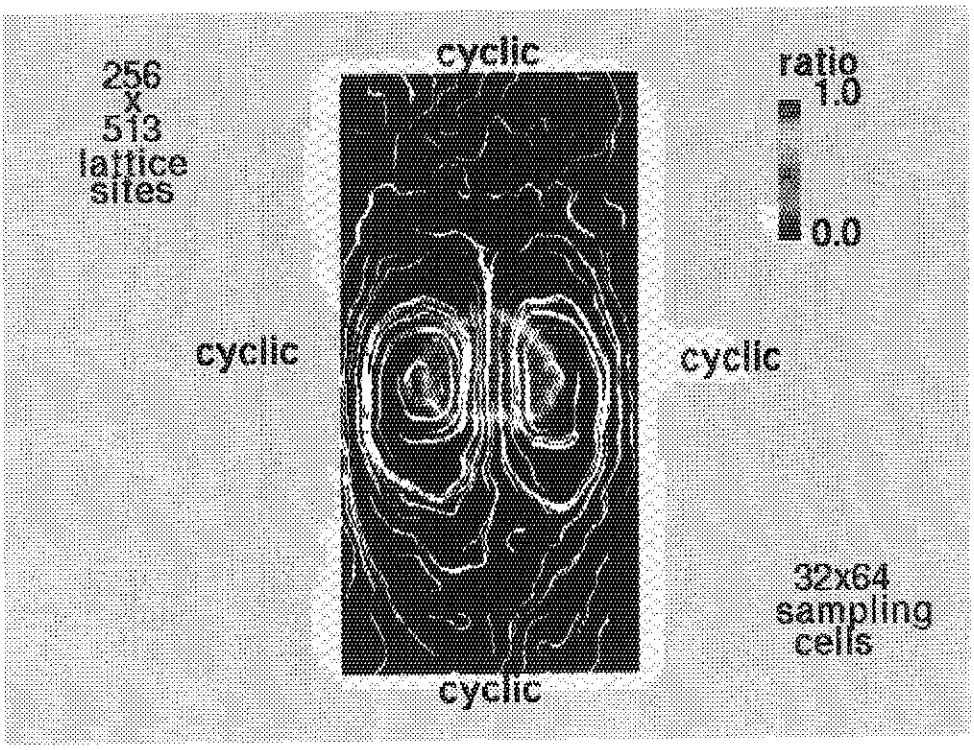

Fig. 3.2 Rising bubble simulation : 20th flow field

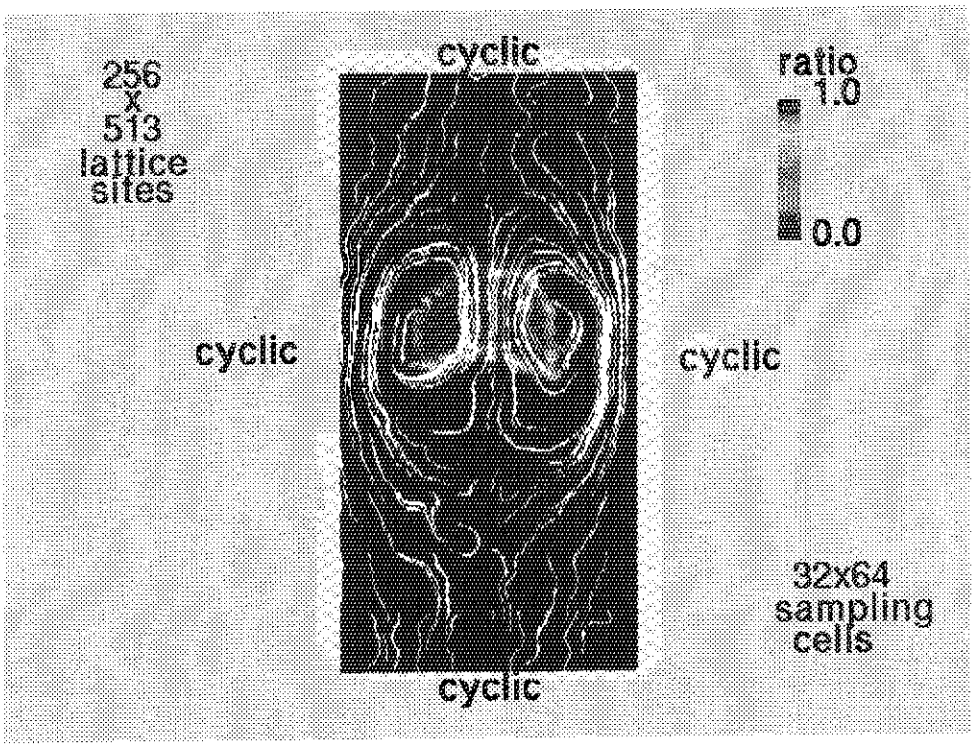

Fig. 3.3 Rising bubble simulation : 33rd flow field

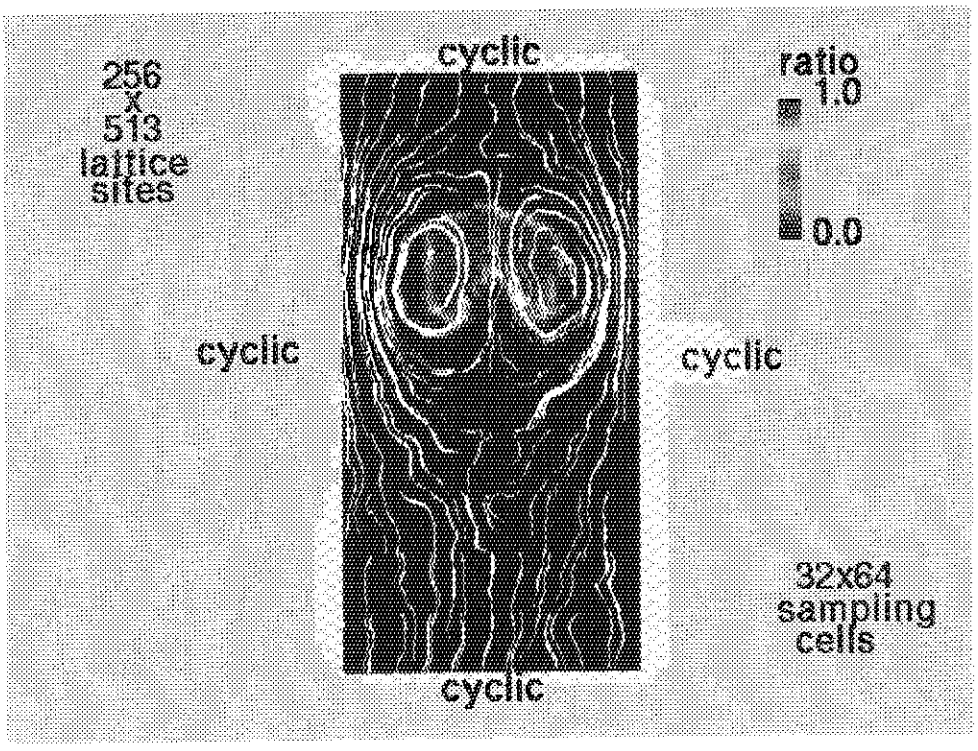

Fig. 3.4 Rising bubble simulation : 50th flow field

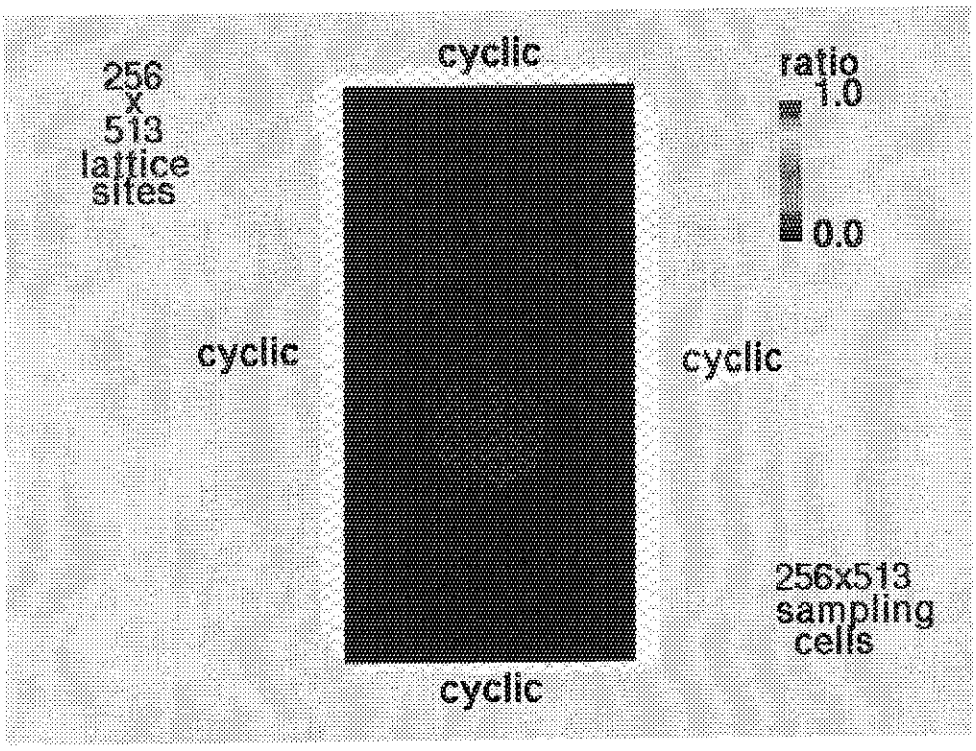

Fig. 3.5 Instant shape of a rising bubble at 1st time step

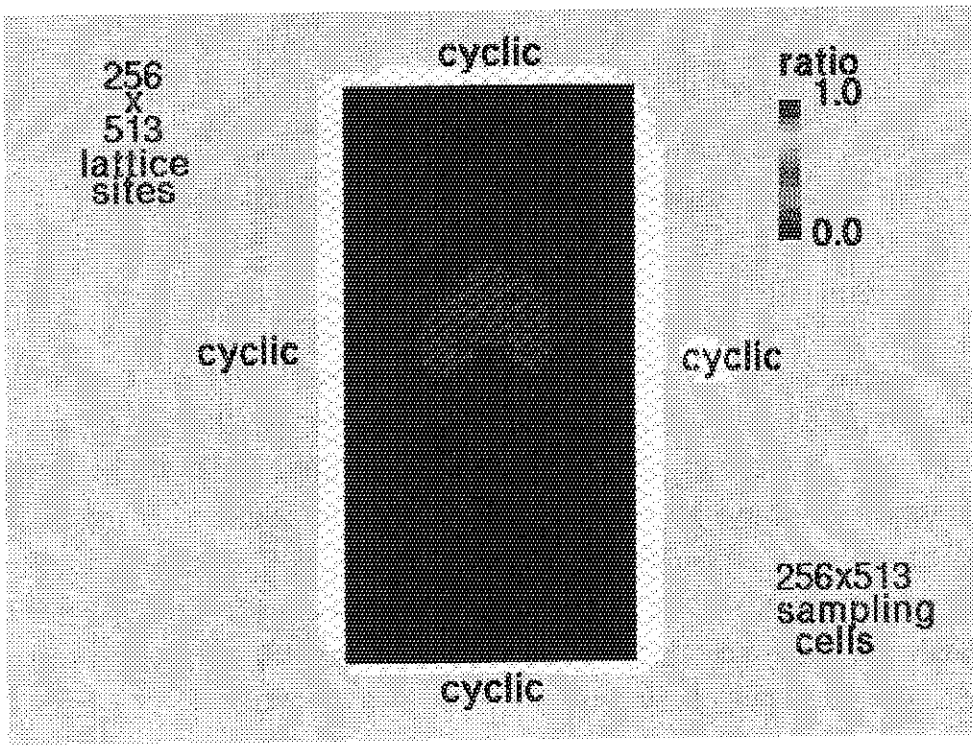

Fig. 3.6 Instant shape of a rising bubble at 3300th time step

4. VPP500 における VPP FORTRAN を用いた並列化

並列化を行うために、まず、VPP500 上でオリジナルコードを用いてシミュレーションを実行し、シミュレーションの各処理に要する計算時間を測定した。測定は、前章に示した例題のリスタート計算を用い、サブルーチン前後に経過時間測定用のルーチンをコールすることによって行った。以下に計測された経過時間を示す。プログラムのコンパイル時にベクトル標準セットのオプション (-UV) を用いた場合を左側に、スカラー標準セットのオプション (-UE) を用いた場合を比較のために右側に示す。

処理	ベクトル経過時間		スカラー経過時間	
	(秒)	(%)	(秒)	(%)
setup	1.38	0.01	1.64	0.01
rflin	0.51	0.00	0.82	0.00
move	1120.16	9.62	1327.51	6.74
cyclic	11.36	0.10	11.30	0.06
wall	73.40	0.63	1023.66	5.20
collis	4989.03	42.84	7899.83	40.13
force	378.05	3.25	4451.13	22.61
sample	5022.57	43.13	4918.35	24.99
propff	49.01	0.42	47.57	0.24
rflout	0.89	0.01	2.84	0.01
total	11646.36	100.00	19684.65	100.00

Table 4.1 Execution time on VPP500 with single processor

ここで、setup は、cdread、iniset、vramfi、addpar の初期化のための 4 つのサブルーチンをまとめたものである。各経過時間は、シミュレーション条件、統計平均のとり方などで多少変化する。また、同一の計算であっても、計算機システムの状態によって経過時間は変化する。なお、経過時間の割合の総和は端数の切捨てのため、正確に 100.00 にはなっていない。オリジナルコードでは、特にベクトル化の作業を行ってはいないため、ベクトル標準セットを用いた際のベクトル化率は 11.2% と小さいが、wall、force などのサブルーチンでは、スカラーとベクトルの経過時間に大きな差が見られる。これは、例えば上下の境界条件を設定する wall では、以下に示すように計算処理を領域の幅に対して行う部分が最も内側の do ループとなっている部分が多く (do 550 のループ)、この部分で高いベクトル化効率が得られたためと考えられる。

```
      do 220 j = 5,6
        do 550 iw=1,width
        ....
550     continue
220 continue
```

j は Fig.2.2 に示すような格子の方向であり、5、6 は下向きの方向を示している。すなわち、ここでは計算領域の下面において下向きの粒子があった場合の処理を、領域の幅方向に対し do 550 のループで行っている。なお、領域の幅 width の値は問題によって変わるが、典型的な値は 128 ～ 1024 程度である。一方、左右の境界条件を設定する cyclic では、同じようにある方向に対しての処理を領域の高さ方向に対して行うが、ここでは、以下に示すように格子の方向に対する処理が内側の do ループ (do 240 のループ) となっている。

```
      do 200 ih=2,height-1
      if(mod(ih,2).ne.0) then
        do 240 j = 1,3
        ....
240     continue
      ....
      else
      ....
      endif
200 continue
```

これは、Fig.2.1 に示すように左右境界では格子点が凹凸に並ぶため、その判定を格子点毎に if 文で行うためである。格子の各方向に対する処理 (ここでは do 240 のループ) は数箇所あり、この内側に高さ方向の処理 (ここでは do 200 のループ) を置くことは可能であるが、wall と異なりプログラムが読みにくくなるという欠点があり、プログラム作成時には考慮しにくい。また、境界条件などは、シミュレーションの種類によってはしばしば変更する必要があるため、経過時間の比率が特に高くない限り読みやすく作成することが重要である。サブルーチン wall も cyclic も特にベクトル化を意識しておらず、今後、修正、変更の可能性も多いためプログラムを作成した時点のままであり、Table 4.1 に示すような差が現れたものである。

ここでは、ベクトル標準セットを用いた経過時間を VPP500 における基準ケースと考える。測定から明らかなように、シミュレーションにおいて経過時間の多い部分は、move、collis、force、sample の各ルーチンである。そこで、これらのルーチンを集中的に並列化することを試みる。なお、本コードは二次元であり、ここで取り上げる例題に対して必要な記憶容量は、およそ 4.4MB であった。このため、計算処理の並列化のみを行うこととする。

4.1 移動 ： サブルーチン move

粒子の移動を計算する move ルーチンは、オリジナルコードでは以下のような処理を行う。

```
      do 19 i=1,maxlat
      gwk(i) = 0
   19 continue
      do 10 ih=1,height-1
      do 10 iw=1,width
      i = width*(ih-1)+iw+(ih-1)/2
      if( ior(grid(i),fix) - grid(i) .eq. 0 ) then
      gwk(i) = iand(grid(i), fixc)
      endif
   10 continue
      do 550 j=1,6
      do 550 ih=2,height-1
      do 550 iw=2,width-mod(ih,2)
      i = width*(ih-1)+iw+(ih-1)/2
      if( ior(grid(i),movef(j)) - grid(i) .eq. 0 ) then
      gwk(i+ptf(j)) =
     c     ior(gwk(i+ptf(j)),movef(j))
      gwk(i+ptf(j)) =
     c     ior(gwk(i+ptf(j)),iand(grid(i),movefc(j)))
      endif
  550 continue
```

ここで、grid(i) は格子点の粒子情報を持つ一次元配列で、gwk(i) はその作業用配列である。ptf(j) は、格子点の隣の六つの格子点の位置を示すものであり、fix は格子点の位置そのもの、すなわち静止状態を示した変数である。movef(j)、movefc(j) は、それぞれ、粒子の存在と色情報を持った配列である。本コードでは、粒子情報を整数配列で表し、粒子と色の有無を 1 または 0 によって表現している。maxlat は総格子数、height、width は、それぞれ、高さ方向の格子数に 1 を加えたもの、及び横方向の格子数である。

サブルーチン move では、ここに示したように、まず配列 grid(i) における静止粒子の有無を調べ、それを配列 gwk(i) に書き込む。つぎに、grid(i) の移動粒子を探し、移動する先の格子点の gwk(i) に書き込む。これら二つの作業（do 10 と do 550 のループ）は新しくゼロクリアされた変数 gwk(i) への書き込みであるため、以下に示すように容易に並列化できる。

```
    do 19 i=1,maxlat
    gwk(i) = 0
19 continue
!xocl spread do
    do 10 ih=1,height-1
    do 10 iw=1,width
    gwk(i) = ....
10 continue
!xocl end spread
!xocl spread do
    do 550 ih=2,height-1
    do 550 j=1,6
    do 550 iw=2,width-mod(ih,2)
    gwk(i+ptf(j)) = ....
550 continue
!xocl end spread sum(gwk)
```

なお、do 550 のループは，高さ方向のループ（ih=2,height-1）を外側とし、方向のループ（j=1,6）を内側にした。これにより計算領域が高さ方向に分割され、任意のプロセッサ数に対しておおむね均一な負荷の分散が期待される。

4.2 衝突 ： サブルーチン collis

粒子の衝突を計算する collis ルーチンは、衝突による位置の変化と、色のフィールドの作用による色の配置の変化を同時に計算する。collis ルーチンは、オリジナルコードでは以下のような構造となっている。

```
    do 10 i=1, maxlat
    grid(i) = 0
    igwk1(i,1) = 0
    igwk1(i,2) = 0
10 continue
```

```
    do 250 j=1,7
    do 250 ih=1,height-1
    do 250 iw=1,width
    i = width*(ih-1)+iw+(ih-1)/2
    if ( iand(gwk(i), colst(j)*7) .gt. 0 )then
    icolo = iand(gwk(i),colst(j)*7)/colst(j)
    igwk1(i,icolo) = igwk1(i,icolo)+1
    endif
250 continue
    do 900 ih=1,height-1
    do 900 iw=1,width
    i=width*(ih-1)+iw+(ih-1)/2
    .....
    call rbmax11(i,iw,ih,ip,1,ig)
    ....
900 continue
```

igwk1(i,j) は i 番目の格子点の j 番目の色の粒子数であり、以下の計算に使用するために、あらかじめ数えておくものである。colst(j) はある方向に対して何色の粒子が存在するかを示す変数で、本コードのデータ構造としては七種類の粒子まで考慮できるようになっているが、ここでは二種類の粒子を使用している。rbmax11 では、i 番目の格子点に対し、その二次元的な位置 (iw,ih) から、周囲の粒子情報を調べ、局所的な色のフィールドを計算し、最終的な粒子と色の配置を決定する。ip は格子点上の粒子の数、ig は衝突後に可能な粒子配置のパターンの数である。色のフィールドに対して色のフラックスの仕事が最小になる色の配置が複数ある場合は、乱数により一つを選択するものとする。格子点の粒子情報 grid(i) は rbmax11 の中で計算される。これら二つのループ（do 250 と do 900 のループ）を以下のように並列化する。

```
!xocl spread do
    do 250 ih=1,height-1
    do 250 j=1,7
    do 250 iw=1,width
    ....
250 continue
!xocl end spread sum(igwk1)
```

```
!xocl spread do
    do 900 ih=1,height-1
    do 900 iw=1,width
    ....
900 continue
!xocl end spread sum(grid)
```

なお、サブルーチン move と同様に do 250 のループにおいて、方向のループ (j=1,7) を内側へ移動した。

4.3 外力 ： サブルーチン force

外力の影響を考慮するサブルーチン force は、ある確率で粒子の向きを変更する。force ルーチンは、オリジナルコードでは以下のような構造となっている。

```
    do 900 ih=2,height-1
    do 900 iw=2,width-mod(ih,2)
    i=width*(ih-1)+iw+(ih-1)/2
    j=2
    jm=6
    if(inr(1).lt.nnb) goto 910
    if(iand(grid(i),colst(j)*7)/colst(j) .eq. 2) then
    if( (ior(grid(i),bitchk(jm)) - grid(i)) .ne. 0 ) then
    grid(i)=grid(i)-bitchk(j)+bitchk(jm)-colst(j)*2+colst(jm)*2
    endif
    endif
910 continue
    ....
    j=2
    jm=6
    if(inr(1).lt.nnb) goto 930
    if(iand(grid(i),colst(j)*7)/colst(j) .eq. 2) then
    if( (ior(grid(i),bitchk(jm)) - grid(i)) .ne. 0 ) then
    grid(i)=grid(i)-bitchk(j)+bitchk(jm)-colst(j)*2+colst(jm)*2
    endif
    endif
930 continue
    ....
900 continue
```

ここでは乱数を発生させ、それ（inr(1)）が、ある指定した値（nnb）に等しく、かつ、j方向に色番号2（青）の粒子があり、jm方向が空いている時、j方向の青粒子をjm方向へ移動させる。bitchk(j)、colst(j)は、それぞれ粒子の存在と色を表す変数である。この処理を色番号1（赤）の粒子に対しても行う。ここでは、計算領域全体で外力を考慮しているが、局所的に考慮することにより、系内に運動量を与えるポンプや撹拌器を表すこともできる。

このルーチンを並列化するために、粒子が移動した格子のデータを格納する作業用変数gwkk(i)を用意し、以下のように書き換える。

```
      do 19 i=1,maxlat
      gwkk(i) = 0
 19 continue
 !xocl spread do
      do 900 ih=2,height-1
      do 900 iw=2,width-mod(ih,2)
      i=width*(ih-1)+iw+(ih-1)/2
      j=2
      jm=6
      if(iand(grid(i),colst(j)*7)/colst(j) .eq. 2) then
      if( (ior(grid(i),bitchk(jm)) - grid(i)) .ne. 0 ) then
      gwkk(i)=grid(i)-bitchk(j)+bitchk(jm)-colst(j)*2+colst(jm)*2
      endif
      endif
      ....
      j=3
      jm=5
      if(iand(grid(i),colst(j)*7)/colst(j) .eq. 2) then
      if( (ior(grid(i),bitchk(jm)) - grid(i)) .ne. 0 ) then
      if(gwkk(i).eq.0) then
      gwk2=grid(i)
      else
      gwk2=gwkk(i)
      endif
      gwkk(i)=gwk2-bitchk(j)+bitchk(jm)-colst(j)*2+colst(jm)*2
      endif
      endif
      ....
 900 continue
```

```
!xocl end spread sum(gwkk)
    do 29 i=1,maxlat
    if(gwkk(i).ne.0) grid(i) = gwkk(i)
 29 continue
```

ここで、j=3 から jm=5 の方向への移動を行う際は、j=2 から jm=6 への移動粒子が無い場合には格子情報として grid(i) を用い、j=2 から jm=6 への移動粒子が有った場合は、既に grid(i) を変更した gwkk(i) を格子情報として用いる必要があるために、一旦 gwk2 という変数に格子情報を代入して用いている。

4.4 サンプリング ： サブルーチン sample

サブルーチン sample では、マクロな流れ場の密度や流速を求めるために、ある小領域における粒子の数と速度をサンプリングする。sample ルーチンは、オリジナルコードでは以下のような構造となっている。

```
    do 1000 iy=1,sdivy
    do 1000 ix=1,sdivx
    is = (iy-1)*sdivx + ix
    io = width*(iy-1)*(nscy-1) + nscx*(ix-1)
c     + int(real((iy-1)*(nscy-1))/2.) + 1
    do 500 k=1,nscy-1,2
    l = ( io + (k-1)*width + int(real(k-1)/2.) ) - 1
    do 100 m=1,7
    do 100 i=l+1,l+nscx
    if( ior(grid(i),bitchk(m))-grid(i) .eq. 0 ) then
    j = iand(grid(i), colst(m)*7) / colst(m)
    spar(j,is) = spar(j,is) + 1.
    svel(j,m,is) = svel(j,m,is) + 1.
    endif
100 continue
    .....
500 continue
    .....
1000 continue
```

sdivy、sdivx は、サンプリングを行う小領域の y 方向、及び x 方向の数である。また、nscx、nscy は、一つのサンプリングセル内の x 方向、及び y 方向の格子点の数である。本ルーチンで

は、is 番目のサンプリングセルにある色番号 j の粒子数を spar(j,is) に積算し、m 方向の運動量を svel(j,m,is) に積算している。ここでは、do 1000 のループを以下のように並列化する。

```
!xocl spread do
    do 1000 iy=1,sdivy
    do 1000 ix=1,sdivx
    .....
1000 continue
!xocl end spread
```

粒子数を集める変数 spar(j,is)、運動量を集める変数 svel(j,m,is) は、このサブルーチン内ではまとめることをせず、データは、それぞれのプロセッサ上に分散したままとする。これらのデータは、流れ場の変数の算出に使用されるため、最後にサブルーチン propff においてまとめることにする。

4.5 物理量の算出 ： サブルーチン propff

ここでは、サンプリングセル内の粒子数 spar(j,is) 及び運動量 svel(j,m,is) から、流れ場の密度及び流速を算出する。オリジナルコードでは、この処理は以下のようなものである。

```
    do 130 j = 1, maxcol
    do 130 i=1,maxcel
    den(j,i) = 0.
    veu(j,i) = 0.
    vev(j,i) = 0.
130 continue
    do 100 i=1,maxcel
    den(1,i) = fhp*spar(2,i)
    if(spar(2,i).gt.0.) then
    u = svel(2,1,i)* f1 + svel(2,2,i)* f2 + svel(2,3,i)*(-f2) +
*       svel(2,4,i)*(-f1) + svel(2,5,i)*(-f2) + svel(2,6,i)* f2
    v = svel(2,2,i)* f3 + svel(2,3,i)* f3 +
*       svel(2,5,i)*(-f3) + svel(2,6,i)*(-f3)
    veu(1,i) = u/spar(2,i)
    vev(1,i) = v/spar(2,i)
    endif
```

```
      tot = spar(1,i)+spar(2,i)
      den(2,i) = tot*fhp
      if(tot.gt.0.) then
      u = (svel(1,1,i)+svel(2,1,i))*f1 + (svel(1,2,i)+svel(2,2,i))*f2 +
     *    (svel(1,3,i)+svel(2,3,i))*(-f2) + (svel(1,4,i)+svel(2,4,i))*(-f1) +
     *    (svel(1,5,i)+svel(2,5,i))*(-f2) + (svel(1,6,i)+svel(2,6,i))*f2
      v = (svel(1,2,i)+svel(2,2,i))*f3 + (svel(1,3,i)+svel(2,3,i))*f3 +
     *    (svel(1,5,i)+svel(2,5,i))*(-f3) + (svel(1,6,i)+svel(2,6,i))*(-f3)
      veu(2,i) = u/tot
      vev(2,i) = v/tot
      endif
  100 continue
```

ここで、den(j,i)、veu(j,i)、vev(j,i) は、それぞれ流れ場の密度、x 方向速度、y 方向速度である。fhp、f1、f2、f3 は方向に対する重みの係数である。このサブルーチンの処理では、do 100 のループに対して並列化を行う。

```
  !xocl spread do
      do 100 i=1,maxcel
      .....
  100 continue
  !xocl end spread sum(den),sum(veu),sum(vev)
```

4.6 並列化効率

以上の並列化を行った後、vpp500 上でプロセッサ数を 1、4、16 とかえて経過時間を測定した結果を以下に示す。プログラムのコンパイルは、ベクトル並列標準セットのオプション (-UW) を使用して行った。

処理	1PE での経過時間		4PE での経過時間		16PE での経過時間	
	(秒)	(%)	(秒)	(%)	(秒)	(%)
setup	1.40	0.01	1.45	0.04	1.41	0.13
rflin	1.05	0.01	0.70	0.02	1.11	0.10
move	1121.45	9.29	340.75	9.84	138.80	13.11
cyclic	11.39	0.09	11.40	0.33	11.34	1.07
wall	73.47	0.61	73.44	2.12	73.37	6.93
collis	5450.90	45.14	1550.60	44.78	347.52	32.32
force	372.31	3.08	126.01	3.64	60.55	5.72
sample	5002.59	41.43	1279.94	36.97	343.33	32.42
propff	39.21	0.32	77.49	2.24	80.13	7.57
rflout	0.47	0.00	0.66	0.02	1.37	0.13
total	12074.24	100.00	3462.44	100.00	1058.93	100.00

Table 4.2 Execution time on VPP500

ここで、計算時間の割合（%）は端数の切捨てのため、総和は正確に 100.00 にはなっていない。表に示されるように、並列化を行ったサブルーチンでは、プロセッサ数を増やすことによって計算時間が短縮されており、トータルでは、4 プロセッサで 3.5 倍、16 プロセッサで 11.4 倍の高速化が実現されている。なお、ベクトル化率は，1 プロセッサで 10.4%、4 プロセッサで 11.5%、16 プロセッサで 15.0% であった。ここでは、データの配列が大きくないため並列処理だけを行いデータ分割は行っていないが、VPP500 では、プロセッサ間の通信速度が 400MB/s と高速であるため、計算処理だけを並列化することによって実用上十分な並列化性能が得られることがわかる。

5. AP3000 における MPI を用いた並列化

ここでは、並列型計算サーバ AP3000 上での並列化について検討する。シミュレーションコードにおいて並列化の対象とする部分は、VPP500 の場合と同様、move、collis、force、sample の四つのサブルーチンとする。

5.1 移動 ： サブルーチン move

ここでは、作業用配列 gwkk(i) を用意し、以下のように並列化を行う。

```
      do 10 ih=iam*nheight+1,(iam+1)*nheight
      do 10 iw=1,width
      gwkk(i) = ....
   10 continue
      do 552 ih=iam*nheight+1,(iam+1)*nheight
      if(ih.eq.1) goto 549
      do 550 j=1,6
      do 550 iw=2,width-mod(ih,2)
      gwkk(i+ptf(j)) = ....
  550 continue
  549 continue
  552 continue
      call mpi_allreduce(gwkk,gwk,maxlat,mpi_integer,mpi_sum,
     c     mpi_comm_world,ierr)
```

ここで、iam は計算を実行しているプロセッサの自分自身の番号（4 プロセッサなら 0 ～ 3）、nheight は（height-1）を並列計算に使用するプロセッサ数で割ったものである。すなわち、ここでは VPP500 の場合と同様に、高さ方向の領域分割を行っている。ただし、do 550 のループは、本来 ih が 2 から始まるループであるため、ih=1 の時はループをとばすように修正してある。VPP500 における end spread sum に対応して、ここでは、mpi_allreduce により gwkk(i) を gwk(i) に集めている。

5.2 衝突 ： サブルーチン collis

ここでは、作業用配列 igwk2(j,i) を用意することにより、j 番目の色の粒子数を数える作業を並列化し、igwk1(i,j) に代入した後、衝突の計算を VPP500 の場合と同様に並列化する。これは、並列に処理した配列を一つにまとめる際に、転送するデータが連続している方が容易なためである。

```
      do 10 i=1, maxlat
      grid(i) = 0
      igwk1(i,1) = 0
      igwk1(i,2) = 0
      igwk2(1,i) = 0
      igwk2(2,i) = 0
 10 continue
      do 250 ih=iam*nheight+1,(iam+1)*nheight
      do 250 j=1,7
      do 250 iw=1,width
      i = width*(ih-1)+iw+(ih-1)/2
      if ( iand(gwk(i), colst(j)*7) .ne. 0 )then
      icolo = iand(gwk(i),colst(j)*7)/colst(j)
      igwk2(icolo,i) = igwk2(icolo,i)+1
      endif
 250 continue
      call mpi_allgatherv(igwk2(1,ist(iam+1)+1),ien2(iam+1),
     c     mpi_integer,
     c     igwk2,ien2,ist2,mpi_integer,
     c     mpi_comm_world,ierr)
      do 100 i=1,maxlat
      igwk1(i,1)=igwk2(1,i)
      igwk1(i,2)=igwk2(2,i)
 100 continue
      do 900 ih=iam*nheight+1,(iam+1)*nheight
      do 900 iw=1,width
      ....
 900 continue
      call mpi_allgatherv(grid(ist(iam+1)+1),ien(iam+1),
     c     mpi_integer,
     c     grid,ien,ist,mpi_integer,
     c     mpi_comm_world,ierr)
```

ここでは、do 250 のループで粒子数を数えた後、mpi_allgatherv を使用して igwk2(j,i) のデータをまとめた後、do 100 のループで igwk1(i,j) に代入している。ist2、ien2 は、それぞれ転送するデータの二次元配列上での変移とデータ数であり、ist、ien は一次元配列上での変移である。do 900 のループでは、領域分割により並列化を行った後、同じく mpi_allgatherv により grid(i) のデータを作成している。

5.3 外力 ： サブルーチン force

サブルーチン force の並列化は、作業用配列 gwkk(i) をとおして行う。

```
      do 19 i=1,maxlat
      gwkk(i) = grid(i)
 19 continue
      do 900 ih=iam*nheight+1,(iam+1)*nheight
      if(ih.eq.1) go to 920
      do 910 iw=2,width-mod(ih,2)
      gwkk(i)=....
 910 continue
 920 continue
 900 continue
      call mpi_allgatherv(gwkk(ist(iam+1)+1),ien(iam+1),
     c     mpi_integer,
     c     grid,ien,ist,mpi_integer,
     c     mpi_comm_world,ierr)
```

ここで、ih=2 から始まる do 900 のループは、サブルーチン move と同様、ih=1 の場合にスキップするようにし、各プロセッサーにおけるステップ数を合わせている。

gwkk(i) は、mpi_allgatherv によりまとめている。

5.4 サンプリング ： サブルーチン sample

サブルーチン sample は VPP500 とほぼ同様に以下のように並列化を行うことができる。

```
      do 1000 iy=iam*nsdivy+1,(iam+1)*nsdivy
      do 1000 ix=1,sdivx
      spar(j,is) = spar(j,is) + 1.
      svel(j,m,is) = svel(j,m,is) + 1.
      ....
 1000 continue
```

ここで、nsdivy は sdivy を並列処理に使用するプロセッサ数で割ったものである。vpp500 の場合と同様、粒子数を集める変数 spar(j,is)、運動量を集める変数 svel(j,m,is) は、このサブルーチン内ではまとめることを行わず、データはそれぞれのプロセッサ上に分散したままとし、最後にサブルーチン propff においてまとめる。

5.5 物理量の算出 ： サブルーチン propff

ここでは、サンプリングセル内の粒子数 spar(j,is) 及び運動量 svel(j,m,is) から、流れ場の密度及び流速を算出する。このサブルーチンの処理では、do 100 のループに対して並列化を行う。並列化のために、密度、速度に対して dwn(j,i)， vwu(j,i)， vwv(j,i) という作業用変数を導入し、計算を行った後、den(j,i)、veu(j,i)、vev(j,i) に代入する。

```
      do 130 j = 1, maxcol
      do 130 i=1,maxcel
      den(j,i) = 0.
      veu(j,i) = 0.
      vev(j,i) = 0.
      dwn(j,i) = 0.
      vwu(j,i) = 0.
      vwv(j,i) = 0.
  130 continue
      do 100 iy=iam*nsdivy+1,(iam+1)*nsdivy
      do 100 ix=1,sdivx
      .....
  100 continue
      call mpi_allreduce(den,dwn,maxcel*2,
     c     mpi_real,mpi_sum,
     c     mpi_comm_world,ierr)
      call mpi_allreduce(veu,vwu,maxcel*2,
     c     mpi_real,mpi_sum,
     c     mpi_comm_world,ierr)
      call mpi_allreduce(vev,vwv,maxcel*2,
     c     mpi_real,mpi_sum,
     c     mpi_comm_world,ierr)
      do 135 j = 1, maxcol
      do 135 i=1,maxcel
      den(j,i) = dwn(j,i)
      veu(j,i) = vwu(j,i)
      vev(j,i) = vwv(j,i)
  135 continue
```

5.6 並列化効率

AP3000 上で MPI を用いて上述の並列化を行い、プロセッサ数を 1、2、4 と変化させて計算時間を測定した結果を以下に示す。すべてサブミットジョブとしてバッチ処理を行った結果である。

処理	1PE での経過時間		2PE での経過時間		4PE での経過時間	
	(秒)	(%)	(秒)	(%)	(秒)	(%)
setup	0.70	0.01	0.71	0.01	0.70	0.02
rflin	0.00	0.00	0.01	0.00	0.01	0.00
move	699.41	7.11	726.17	12.10	810.15	18.65
cyclic	5.87	0.06	6.08	0.10	7.02	0.16
wall	501.04	5.09	440.07	7.33	500.05	11.51
collis	3839.05	39.02	2304.54	38.40	1621.69	37.33
force	1863.13	18.94	1053.52	17.56	662.45	15.25
sample	2918.01	29.66	1456.73	24.28	728.54	16.77
propff	11.61	0.12	12.91	0.22	13.70	0.32
rflout	0.00	0.00	0.00	0.00	0.02	0.00
total	9838.82	100.00	6000.74	100.00	4344.33	100.00

Table 5.1 Execution time on AP3000 using MPI

まず、1PE において Table 4.2 と比べて、VPP500 よりも total の経過時間が少なくなっている。これは、VPP500 において経過時間が多い部分は、Table 4.1 からわかるようにベクトル化効率の小さい部分であり、この部分が AP3000 において高速に処理されているためである。これは、スカラー処理に関しては、VPP500 よりも AP3000 のほうが高速であるためと考えることができる。

Table 5.1 では、サブルーチン collis、force、sample でプロセッサ数の増加によりある程度の計算の高速化が達成されているが、move では逆に経過時間は増加している。これは、サブルーチン move では計算処理に対してデータ転送の割合が多いためと考えられる。他の三つのサブルーチンでは、一応プロセッサ数の増加とともに経過時間が減少しているというものの、その割合は、VPP500 の場合に比べ小さいものとなっている。トータルとして見ると 2 プロセッサで 1.6 倍、4 プロセッサで 2.3 倍の高速化に過ぎない。これは、AP3000 と VPP500 においてプロセッサ間のデータ転送に関わる処理速度が異なることに主に起因すると考えられる。データ転送速度は、AP3000 では 200MB/s であり、VPP500 の 1/2 である。このため、データ転送の割合の大きい部分では、計算処理の並列化だけでは、十分な並列化効率が期待できないということである。これは、転送速度がそれほど高速ではない計算サーバやワークステーションクラス

ターでは、深刻な問題となる。スーパーコンピュータ以外の分散メモリー型計算機システムで並列計算を行う場合、データ分割は必要不可欠と考えられる。

ここで、VPP500 で VPP FORTRAN を用いて並列化を行ったコードをそのまま AP3000 上で実行した結果を比較のために以下に示す。

処理	1PE での経過時間		2PE での経過時間		4PE での経過時間	
	(秒)	(%)	(秒)	(%)	(秒)	(%)
setup	0.00	0.00	0.00	0.00	0.01	0.00
rflin	10.41	0.10	24.12	0.34	26.00	0.31
move	773.60	7.76	936.30	13.09	1865.29	22.56
cyclic	2.98	0.03	7.93	0.11	7.99	0.10
wall	451.29	4.53	435.14	6.08	448.92	5.43
collis	3983.61	39.97	2929.57	40.96	3553.35	42.98
force	1767.80	17.74	1227.59	17.16	1368.91	16.56
sample	2965.46	29.75	1572.24	21.98	980.92	11.86
propff	11.54	0.12	19.36	0.27	16.64	0.20
rflout	0.03	0.00	0.01	0.00	0.00	0.00
total	9966.72	100.00	7152.26	100.00	8268.03	100.00

Table 5.2 Execution time on AP3000 using VPP FORTRAN

経過時間の増減の傾向は、MPI を用いた場合とほぼ同様であるが、サブルーチン collis、及び force で 4 プロセッサの場合に経過時間が増加しており、転送処理に必要な時間の割合が MPI を使用した場合よりも大きくなっていることがわかる。move における経過時間の増加とあわせて、全経過時間も 4 プロセッサでは増加している。2 プロセッサで 1.4 倍、4 プロセッサで 1.2 倍である。これは、AP3000 では、ほぼ同じ並列処理を行う際のデータ転送処理に関して、VPP FORTRAN のユーザ指示行の処理よりも MPI の処理の方が高速であることを示している。すなわち、AP3000 上で並列化を行う場合、MPI を使用した方が高い並列化効率が期待できるということがわかる。

ここで、VPP500 と AP3000 での転送に関わる処理の速度の簡単な比較を行っておく。比較を行うサンプル問題として、以下のような単純な代入処理を行うプログラムを VPP FORTRAN を用いて作成し、VPP500 と AP3000 で 4PE により実行する。

```
c!xocl spread do
    do 1 i=1,1000000
    j(i)=i
1   continue
c!xocl end spread
c!xocl end spread sum(j)
```

このプログラムに対して並列処理を行わないもの（逐次）、並列処理で spread do だけを行うもの（spread）、並列処理で spread do の後 sum をとるもの（spread-sum）、の三ケースを行い、経過時間を測定したものが以下の表である。VPP500 では、ベクトル標準セットのオプション (-UV、-UW) を用いている。

計算機	逐次	spread	spread-sum
	(秒)	(秒)	(秒)
VPP500	0.00250	0.00066	0.02158
AP3000	0.02142	0.12493	1.27296

Table 5.3 Execution time on VPP500 and AP3000 using VPP FORTRAN

VPP500 では、4PE での並列処理により経過時間が 1/4 近くまで減っており、並列処理を行うために必要な手続きに要する時間が十分小さいことがわかる。和をとる操作を行うと、データ量に応じて転送時間が必要となるため経過時間は増加している。AP3000 では、並列処理によって経過時間は増加しており、並列処理に必要な手続きに要する時間が比較的長いことがわかる。和をとる処理を行うと経過時間がさらに増大しており、これからも AP3000 においては転送処理に関わる時間が VPP500 に比べて無視できない程度であることがわかる。

参考までにAP3000においてMPIを使用し、同様のプログラムを以下に示すように作成し、4PEを用いて実行する。表に示したのは、逐次処理、並列処理(4nodes)、並列処理後総和(4nodes-reduce)の3ケースについての経過時間の測定値である。

```
c     do 1 i=250000*iam+1,250000*(iam+1)
      do 1 i=1,1000000
      j(i)=i
1     continue
c     call mpi_allreduce(j,jw,1000000,mpi_integer,mpi_sum,
c     c       mpi_comm_world,ierr)
```

計算機	逐次	4nodes	4nodes-reduce
	(秒)	(秒)	(秒)
AP3000	0.0300	0.0100	0.4600

Table 5.4 Execution time on AP3000 using MPI

まず、VPP FORTRANを用いた場合と比較して逐次処理が若干遅くなっているが、並列処理による高速化は達成されている。さらに、allreduceによりデータの転送を行うと経過時間は増加してしまうが、VPP FORTRANの場合よりも十分小さくなっている。このことから、AP3000においては転送も含めて並列処理に必要な手続きに要する時間は、MPIの方がVPP FORTRANよりも短いことが確認できる。また、VPP500の測定値と比べると、逐次処理は若干遅いものの、並列処理により高速化される点は定性的に同様である。ただし、データ転送を行うと大幅に経過時間が増えており、このことからも転送処理に必要な時間が比較的長いことがわかる。

これらのことから、AP3000において並列処理を行う際には、VPP FORTRANよりもMPIの方が効率的であり、さらに、データ分割等により転送処理を軽減することが望ましい、ということが確認できる。

6. まとめ

本報告書では、非混合格子ガスモデルによる二次元二相流シミュレーションコードの作成と並列化について記述した。非混合格子ガスモデルは、統計的によりマクロな流体挙動を計算するとともに、瞬時の二相界面形状をミクロ、あるいはメソスコピックなレベルで表現する計算手法である。作成した二次元コードはデータ容量が小さいため、データ分割を行うことをせずに、簡易的に計算処理のみを並列化し、並列化効率を検討した。並列化は東海研に導入されているベクトル並列型スーパーコンピュータ VPP500 及び並列型計算サーバ AP3000 上において行った。その結果、VPP500 においては、プロセッサ間のデータ転送に関わる処理速度が高速であるため、計算処理だけを並列化する簡易並列化のみで実用上十分な並列化効率が得られるのに対し、AP3000 では、転送処理がネックとなってプロセッサ数の増加に対してかえって経過時間が増加する部分が多くなることが明らかとなった。このことは、分散メモリー型の計算機システムにおいて、とりわけ、プロセッサ間のデータ転送処理の遅いシステムやワークステーションクラスターなどでは、データ容量の小さいプログラムであってもデータ分割が必要不可欠となることを示している。また、AP3000 上で VPP FORTRAN のユーザ指示行と MPI の比較を行ったところ、データ転送を含み並列処理に必要な作業に関して MPI の方が高速な処理が行えることが示された。したがって、VPP500 上のプログラムをそのまま AP3000 上で実行する場合を除いて、AP3000 において並列処理を行う場合は、MPI を使用する方が高い並列化効率が期待されるということがわかった。

なお、本報告書で作成した二相流シミュレーションコードは、(財) 電力中央研究所との共同研究「気液二相流のミクロ／マクロ・モデルによる解析に関する研究」において使用するたものものである。

謝　辞

本報告書をまとめるにあたり、有益な助言をいただいた計算科学技術推進センター並列処理支援技術開発グループ、渡部弘氏に感謝いたします。

6. まとめ

本報告書では、非混合格子ガスモデルによる二次元二相流シミュレーションコードの作成と並列化について記述した。非混合格子ガスモデルは、統計的によりマクロな流体挙動を計算するとともに、瞬時の二相界面形状をミクロ、あるいはメソスコピックなレベルで表現する計算手法である。作成した二次元コードはデータ容量が小さいため、データ分割を行うことをせずに、簡易的に計算処理のみを並列化し、並列化効率を検討した。並列化は東海研に導入されているベクトル並列型スーパーコンピュータ VPP500 及び並列型計算サーバ AP3000 上において行った。その結果、VPP500 においては、プロセッサ間のデータ転送に関わる処理速度が高速であるため、計算処理だけを並列化する簡易並列化のみで実用上十分な並列化効率が得られるのに対し、AP3000 では、転送処理がネックとなってプロセッサ数の増加に対してかえって経過時間が増加する部分が多くなることが明らかとなった。このことは、分散メモリー型の計算機システムにおいて、とりわけ、プロセッサ間のデータ転送処理の遅いシステムやワークステーションクラスターなどでは、データ容量の小さいプログラムであってもデータ分割が必要不可欠となることを示している。また、AP3000 上で VPP FORTRAN のユーザ指示行と MPI の比較を行ったところ、データ転送を含み並列処理に必要な作業に関して MPI の方が高速な処理が行えることが示された。したがって、VPP500 上のプログラムをそのまま AP3000 上で実行する場合を除いて、AP3000 において並列処理を行う場合は、MPI を使用する方が高い並列化効率が期待されるということがわかった。

なお、本報告書で作成した二相流シミュレーションコードは、(財) 電力中央研究所との共同研究「気液二相流のミクロ／マクロ・モデルによる解析に関する研究」において使用するたものものである。

謝　辞

本報告書をまとめるにあたり、有益な助言をいただいた計算科学技術推進センター並列処理支援技術開発グループ、渡部弘氏に感謝いたします。

参考文献

[1] 保原 充, 大宮司 久明 編：" 数値流体力学", 東京大学出版会, 東京 (1992) .

[2] G. A. Bird: "Molecular Gas Dynamics and the Direct Simulation of Gas Flows", (Clarendon, Oxford, 1994).

[3] 上田 顕：" コンピュータシミュレーション", 朝倉書店, 東京 (1990).

[4] D. H. Rothman and S. Zaleski: "Lattice-gas models of phase separation: interfaces, phase transitions, and multiphase flow", Rev. Mod. Phys., **66**, 1417(1994).

[5] 矢部 孝, 観山 正見, 椛島 成治："パソコンによるシミュレーション物理", 朝倉書店, 東京 (1992) .

[6] 矢部 孝, 川田 重夫, 福田 昌宏：" シミュレーション物理入門", 朝倉書店, 東京 (1989) .

[7] D. H. Rothman and J. M. Keller: "Immiscible cellular-automaton fluids," J. Stat. Phys., **52**, 1119(1988).

[8] 海老原 健一, 渡辺 正, 蕪木 英雄：" 格子ガスセルオートマトンの気液モデルによる相分離の研究 ", JAERI-Research 97-043(1997).

[9] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas automata for the Navier-Stokes equation," Phys. Rev. Lett., **56**, 1505(1986).

[10] U. Frisch, D. d'Humieres, B. Hasslacher, P. Lallemand, Y. Pomeau, and J. P. Rivet, "Lattice gas hydrodynamics in two and three dimensions," Complex Systems, **1**, 649(1987).

[11] D. d'Humieres and P. Lallemand, "Numerical simulations of hydrodynamics with lattice gas automata in two dimensions," Complex Systems, **1**, 599(1987).

[12] C. Appert, J. F. Olson, D. H. Rothman, and S. Zaleski, "Phase separation in a three-dimnsional, two-phase, hydrodynamic lattice gas," J. Stat. Phys., **81**, 181(1995): J. F. Olson and D. H. Rothman, "Three-dimensional immiscible lattice gas: application to sheared phase separation," J. Stat. Phys., **81**, 199(1995).

[13] 渡辺 正, 蕪木 英雄：" 粒子法による熱伝導 - 対流遷移の研究 ", JAERI-Research 96-046(1996).

[14] 加藤 克海, 渡辺 正, 蕪木 英雄： 私信

[15] 加藤 克海, 渡辺 正, 蕪木 英雄：" リアルタイムモニターシステムの開発 ", JAERI-Tech 96-044(1996).