JAERI-Data/Code
96-013

2次元Lattice Boltzmannコードの並列計算

1996年3月

鈴木惣一朗*・蕪木英雄・横川三津夫

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，日本原子力研究所技術情報部情報資料課（〒319-11 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（〒319-11 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 Information Division, Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

© Japan Atomic Energy Research Institute, 1996

編集兼発行　日本原子力研究所
印　　刷　㈱原子力資料サービス

JAERI－Data/Code 96－013

2次元 Lattice Boltzmann コードの並列計算

日本原子力研究所計算科学技術推進センター
鈴木惣一朗*・蕪木　英雄・横川三津夫

（1996 年2月9日受理）

Lattice Boltzmann 法を用いた2次元流体シミュレーション・コードをベクトル並列計算機富士通VPP500，およびスカラ並列計算機Intel Paragon XP/S上で開発した。ベクトル並列化には1次元領域分割を，スカラ並列化には2次元領域分割を用いた。並列化効率は，ベクトル並列（1152×1152グリッド，16プロセッサ）で95.1%，スカラ並列(800×800グリッド，100プロセッサ）で88.6%となり，高い効率が得られた。同数のプロセッサでの計算速度は，ベクトル並列がスカラ並列の約100倍あり，パフォーマンス・モデルを用いた予測では，この値が100プロセッサ程度まで保たれる。ベクトル並列計算で，プロセッサあたりの計算領域をメモリの上限にとりプロセッサ数に比例して全計算領域を増加させた場合，プロセッサを数 100台に増加しても計算時間は数パーセントした増えないことが分かった。したがって，大規模領域あるいは細密メッシュでの数値実験という立場から見ると，一般にプロセッサ間通信にコストがかかると言われる1次元領域分割を用いたベクトル並列コードでも，充分実用的である。

日本原子力研究所：〒153　東京都目黒区中目黒2-2-54
＊ 業務協力員，㈱情報数理研究所　技術部第四課

JAERI−Data/Code 96−013

Parallelization of 2-D Lattice Boltzmann Codes

Soichiro SUZUKI* ,Hideo KABURAKI and Mitsuo YOKOKAWA

Center for Promotion of Computational Science and Engineering
Japan Atomic Energy Research Institute
Nakameguro, Meguro-ku, Tokyo

(Received February 9, 1996)

Lattice Boltzmann (LB) codes to simulate two dimensional fluid flow are developed on vector parallel computer Fujitsu VPP500 and scalar parallel computer Intel Paragon XP/S. While a 2-D domain decomposition method is used for the scalar parallel LB code, a 1-D domain decomposition method is used for the vector parallel LB code to be vectorized along with the axis perpendicular to the direction of the decomposition. High parallel efficiency of 95.1% by the vector parallel calculation on 16 processors with 1152×1152 grid and 88.6% by the scalar parallel calculation on 100 processors with 800×800 grid are obtained. The performance models are developed to analyze the performance of the LB codes. It is shown by our performance models that the execution speed of the vector parallel code is about one hundred times faster than that of the scalar parallel code with the same number of processors up to 100 processors. We also analyze the scalability in keeping the available memory size of one processor element at maximum. Our performance model predicts that the execution time of the vector parallel code increases about 3% on 500 processors. Although the 1-D domain decomposition method has in general a drawback in the inter-processor communication, the vector parallel LB code is still suitable for the large scale and/or high resolution simulations.

Keywords: Lattice Gas, Lattice Boltzmann eq., Parallel Calculation, Vector Parallel Calculation, VPP500, Paragon, Scalability Analysis, Large Scale Simulation, High Resolution Simulation

---

* on loan to Information and Mathematical Science Laboratory, Inc.

目　　次

Contents

1. 序論

Lattice gas 法は離散空間上の仮想粒子の運動をセルオートマトンによりシミュレートするものである。また、Lattice Bolzmann 法は lattice gas の仮想粒子に対する Boltzmann 方程式 (lattice Boltzmann 方程式) を解くものである。どちらも適当な条件下で Navier-Stokes 方程式に従う速度場が得られる。これらの方法は流体の新しい数値計算法として注目され、通常の流れの他に乱流、多孔質媒体中の流れ、化学反応を伴なう流れ、電磁流体などへの応用が試みられている[1)]。Lattice gas 法は、変数が boolean (粒子がある or ない) であるためメモリ効率などの利点があるが、計算により得られる速度場などに粒子性に起因する大きなノイズがあり、空間・時間的な粗視化 (スムージング) を行なう必要がある。一方、lattice Bolzmann 法は Lattice gas と違い実数変数である分布関数を扱うため、計算結果にスムージングを行なう必要はない。

いずれの方法も離散点における計算には空間的に強い独立性があり、並列計算が可能である。特に実数演算が中心となる lattice Boltzmann 法は、ベクトル並列計算機による高速計算が期待される。今までスカラ並列計算機での計算報告はあるが[2)]、ベクトル並列計算機でのものはない。今回我々は、ベクトル並列計算機上での lattice Boltzmann コードを開発して、その大規模高速計算の可能性について調べた。ベクトル並列コードの他に、スカラ並列計算機上での並列コードを作成して性能評価の比較を行なったので報告する。ベクトル並列コードは富士通 VPP500 上で、スカラ並列コードは Intel Paragon XP/S 上で開発した。

また、高レイノルズ数流れ ($R = 10^4$) の計算で、lattice Boltzmann コードと差分法を用いたものとの計算結果の比較を行なったので、あわせて報告する。

2. Lattice Boltzmann 法

2.1 Lattice Boltzmann 方程式

ここでは、コードの基礎となる lattice Boltzmann 方程式について説明する。Fig.2.1 のように 2 次元 3 角メッシュを考える。粒子は各ノードに存在でき、単位時間に最近接ノードにのみ移動できるものとする。しがたって、粒子速度は 7 つのベクトル $\{\boldsymbol{c}_i, i = 0 \sim 6\}$ ($i = 0$ は静止粒子) のいずれかを持つ。時刻 t にメッシュ上のノード $\boldsymbol{x}$ に存在する速度 $\boldsymbol{c}_i$ の粒子数を分布関数 $N_i(\boldsymbol{x}, t)$ で表すと、巨視的な粒子密度 ρ、速度 $\boldsymbol{u}$ は、N に対する平均値として以下のように定義できる。

$$\rho(\boldsymbol{x}, t) = \sum_{i=0}^{6} N_i(\boldsymbol{x}, t) \tag{1}$$

$$\boldsymbol{J}(\boldsymbol{x}, t) = \rho \boldsymbol{u}(\boldsymbol{x}, t) = \sum_{i=0}^{6} \boldsymbol{c}_i N_i(\boldsymbol{x}, t) \tag{2}$$

分布関数 N の時間発展として、我々は以下の lattice Boltzmann 方程式を用いた[3)]。

$$N_i(\boldsymbol{x} + \boldsymbol{c}_i, t + 1) - N_i(\boldsymbol{x}, t) = \sum_{j=0}^{6} A_{i,j} \left(N_j(\boldsymbol{x}, t) - N_j^{eq}(\boldsymbol{x}, t) \right) \tag{3}$$

1. 序論

Lattice gas 法は離散空間上の仮想粒子の運動をセルオートマトンによりシミュレートするものである。また、Lattice Bolzmann 法は lattice gas の仮想粒子に対する Boltzmann 方程式 (lattice Boltzmann 方程式) を解くものである。どちらも適当な条件下で Navier-Stokes 方程式に従う速度場が得られる。これらの方法は流体の新しい数値計算法として注目され、通常の流れの他に乱流、多孔質媒体中の流れ、化学反応を伴なう流れ、電磁流体などへの応用が試みられている[1)]。Lattice gas 法は、変数が boolean (粒子がある or ない) であるためメモリ効率などの利点があるが、計算により得られる速度場などに粒子性に起因する大きなノイズがあり、空間・時間的な粗視化 (スムージング) を行なう必要がある。一方、lattice Bolzmann 法は Lattice gas と違い実数変数である分布関数を扱うため、計算結果にスムージングを行なう必要はない。

いずれの方法も離散点における計算には空間的に強い独立性があり、並列計算が可能である。特に実数演算が中心となる lattice Boltzmann 法は、ベクトル並列計算機による高速計算が期待される。今までスカラ並列計算機での計算報告はあるが[2)]、ベクトル並列計算機でのものはない。今回我々は、ベクトル並列計算機上での lattice Boltzmann コードを開発して、その大規模高速計算の可能性について調べた。ベクトル並列コードの他に、スカラ並列計算機上での並列コードを作成して性能評価の比較を行なったので報告する。ベクトル並列コードは富士通 VPP500 上で、スカラ並列コードは Intel Paragon XP/S 上で開発した。

また、高レイノルズ数流れ ($R = 10^4$) の計算で、lattice Boltzmann コードと差分法を用いたものとの計算結果の比較を行なったので、あわせて報告する。

2. Lattice Boltzmann 法

2.1 Lattice Boltzmann 方程式

ここでは、コードの基礎となる lattice Boltzmann 方程式について説明する。Fig.2.1 のように 2 次元 3 角メッシュを考える。粒子は各ノードに存在でき、単位時間に最近接ノードにのみ移動できるものとする。しがたって、粒子速度は 7 つのベクトル $\{\boldsymbol{c}_i, i = 0 \sim 6\}$ ($i = 0$ は静止粒子) のいずれかを持つ。時刻 t にメッシュ上のノード $\boldsymbol{x}$ に存在する速度 $\boldsymbol{c}_i$ の粒子数を分布関数 $N_i(\boldsymbol{x}, t)$ で表すと、巨視的な粒子密度 ρ、速度 $\boldsymbol{u}$ は、N に対する平均値として以下のように定義できる。

$$\rho(\boldsymbol{x}, t) = \sum_{i=0}^{6} N_i(\boldsymbol{x}, t) \tag{1}$$

$$\boldsymbol{J}(\boldsymbol{x}, t) = \rho \boldsymbol{u}(\boldsymbol{x}, t) = \sum_{i=0}^{6} \boldsymbol{c}_i N_i(\boldsymbol{x}, t) \tag{2}$$

分布関数 N の時間発展として、我々は以下の lattice Boltzmann 方程式を用いた[3)]。

$$N_i(\boldsymbol{x} + \boldsymbol{c}_i, t + 1) - N_i(\boldsymbol{x}, t) = \sum_{j=0}^{6} A_{i,j} \left(N_j(\boldsymbol{x}, t) - N_j^{eq}(\boldsymbol{x}, t) \right) \tag{3}$$

左辺は粒子の最近接ノードへの移動を表わす。右辺は BGK タイプの衝突項で、(局所) 平衡分布 N^{eq} への緩和を表わしている[4]。

衝突行列 $A_{i,j}$ は、物理的要請から以下のような制限が加わる[3)4)]。微視的な時間反転の対称性より、対称行列となる。三角格子の空間対称性より $A_{i,j}$ は、ベクトル $\boldsymbol{c}_i$ と $\boldsymbol{c}_j$ の間の角度のみに依存する。密度・運動量の保存より、それらに対応する 3 つの固有値は 0 になる。これらの制限により、$A_{i,j}$ は 3 つのパラメータ $(\lambda_s, \lambda_b, \lambda_g)$ のみで特徴づけられ、これらは 0 でない 4 つ (λ_s は 2 重に縮退) の固有値である。また

$$-2 < \lambda_s, \lambda_b, \lambda_g < 0 \tag{4}$$

の条件で、(3) 式は線形安定である。

平衡分布関数 N^{eq} として以下のものを用いた。

$$\begin{aligned} N_0^{eq} &= \frac{1}{7}\rho - \frac{1}{c^2}\rho u^2 \\ N_{i\neq 0}^{eq} &= \frac{1}{7}\rho + \frac{1}{3c^2}\rho \boldsymbol{c}\cdot\boldsymbol{u} \\ &\quad + \frac{2}{3c^4}\rho\left(c_{i\alpha}c_{i\beta} - \frac{c^2}{2}\delta_{\alpha\beta}\right)u_\alpha u_\beta + \frac{1}{6c^2}\rho u^2 \end{aligned} \tag{5}$$

ここで、ギリシャ文字の添字は空間成分 x または y で、例えば $c_{i\alpha}$ はベクトル $\boldsymbol{c}_i$ の α 成分を表わす。また、アインシュタインの総和規則をとるものとする。

2.2 Navier-Stokes 方程式との関係

Lattice Boltzmann 方程式 (3) から Navier-Stokes 方程式を導くには通常の Botlzmann 方程式と同様に Chapman-Enskog 展開を用いる[5)6)]。平衡分布関数として (5) 式をとると、速度 $\boldsymbol{u}$ の小さい極限で、連続の式と Navier-Stokes 方程式が得られる。

$$\partial_t\rho + \boldsymbol{\nabla}\cdot\boldsymbol{J} = 0 \tag{6}$$

$$\partial_t J_\alpha + \partial_\beta J_\alpha u_\beta = -\partial_\alpha p + \zeta\partial_\alpha\boldsymbol{\nabla}\cdot\boldsymbol{J} \tag{7}$$

$$+ \nu\partial_\beta\left(\partial_\alpha J_\beta + \partial_\beta J_\alpha - (\boldsymbol{\nabla}\cdot\boldsymbol{J})\delta_{\alpha\beta}\right) \tag{8}$$

ここで、∂_t は時間について ∂_α は空間の α 成分についての偏微分を表わす。

圧力 p、動粘性率 ν、体積粘性率 ζ と lattice Boltzmann 方程式のパラメータとの関係は以下の通りである。

$$p = \frac{3}{7}c^2\rho \tag{9}$$

$$\nu = \frac{\tau c^2}{8}\left(\frac{-2}{\lambda_s} - 1\right) \tag{10}$$

$$\zeta = \frac{\tau c^2}{28}\left(\frac{-2}{\lambda_b} - 1\right) \tag{11}$$

ここで、$c = |\boldsymbol{c}_{i\neq 0}|$。また、時間ステップ幅を 1 から τ に改めた*。音速 c_s は

*これは、(3) 式の左辺で、$\boldsymbol{c}_i \to \boldsymbol{c}_i\tau$、$1 \to \tau$ と置き換えたことに相当する。

$$c_s = c\sqrt{\frac{3}{7}} \tag{12}$$

で与えられる。

次に レイノルズ数について調べるために、いくつかのパラメータを定義する。

L　系を特徴づける長さ
N　長さ L に含まれるノード数
U　特徴的な速度
M　マッハ数 $\left(=\sqrt{\frac{7}{3}}N\tau U/L\right)$

すると、レイノルズ数 R は以下のように表わすことができる。

$$R = \frac{UL}{\nu} = 8\sqrt{\frac{3}{7}}MN\left(\frac{-2}{\lambda_s} - 1\right)^{-1} \tag{13}$$

上式より、高いレイノルズ数の流れを実現するには、

- 速度を上げる。($M \rightarrow$ 大)
- 粘性率を下げる。($\lambda_s \rightarrow -2$)
- 細かいグリッドを用いる。($N \rightarrow$ 大)

とすればよい。しかし、Navier-Stokes 方程式を Lattice Boltzmann 方程式から導くのに、マッハ数 M が小さいという条件を用いているため、流体の速度はあまり大きくとれない。また、粘性率を小さくするために $\lambda_s \rightarrow -2$ とすることは安定性の限界に近づけることになり、この場合も Navier-Stokes 方程式からのずれが生じることが知られている[7]。したがって lattice Boltzmann 法を用いて高レイノルズ数流れを数値計算するには、より細かなグリッドを使用しなければならず、大容量のメモリを備えた計算機が要求される。

分散メモリ型の並列計算機上の lattice Boltzmann コードには、単に高速計算を目指すだけでなく、大容量メモリをいかした高レイノルズ数流れのシミュレーションが可能になるという利点がある。

3. 差分法との比較

我々は、今回作成した lattice Boltzmann コードと差分コードとの計算結果の比較を行なった。比較には、富士通 VPP500 上での 1 プロセッサでの計算結果をもとにした。コードの並列化とは直接は関係のない話題なので、比較の結果についてこの章で報告する。

マッハ数が低く非圧縮性が満される条件のもとで、lattice Boltzmann コードと非圧縮差分コードとの比較を行なった。差分コードには、アルゴリズムに HSMAC スキーム[8] および移流項に QUICK スキーム[8]を用いたものを使用した。

計算領域は 2 次元周期境界条件の正方形領域 ($0 \leq x, y < 1$) とし、差分コードでは lattice Boltzmann コードと同数のノードをもった長方形グリッドを用いた。また、初期条件は以下のも

$$c_s = c\sqrt{\frac{3}{7}} \tag{12}$$

で与えられる。

次に レイノルズ数について調べるために、いくつかのパラメータを定義する。

L　系を特徴づける長さ
N　長さ L に含まれるノード数
U　特徴的な速度
M　マッハ数 $\left(=\sqrt{\frac{7}{3}}N\tau U/L\right)$

すると、レイノルズ数 R は以下のように表わすことができる。

$$R = \frac{UL}{\nu} = 8\sqrt{\frac{3}{7}}MN\left(\frac{-2}{\lambda_s} - 1\right)^{-1} \tag{13}$$

上式より、高いレイノルズ数の流れを実現するには、

- 速度を上げる。($M \rightarrow$ 大)
- 粘性率を下げる。($\lambda_s \rightarrow -2$)
- 細かいグリッドを用いる。($N \rightarrow$ 大)

とすればよい。しかし、Navier-Stokes 方程式を Lattice Boltzmann 方程式から導くのに、マッハ数 M が小さいという条件を用いているため、流体の速度はあまり大きくとれない。また、粘性率を小さくするために $\lambda_s \rightarrow -2$ とすることは安定性の限界に近づけることになり、この場合も Navier-Stokes 方程式からのずれが生じることが知られている[7]。したがって lattice Boltzmann 法を用いて高レイノルズ数流れを数値計算するには、より細かなグリッドを使用しなければならず、大容量のメモリを備えた計算機が要求される。

分散メモリ型の並列計算機上の lattice Boltzmann コードには、単に高速計算を目指すだけでなく、大容量メモリをいかした高レイノルズ数流れのシミュレーションが可能になるという利点がある。

3. 差分法との比較

我々は、今回作成した lattice Boltzmann コードと差分コードとの計算結果の比較を行なった。比較には、富士通 VPP500 上での 1 プロセッサでの計算結果をもとにした。コードの並列化とは直接は関係のない話題なので、比較の結果についてこの章で報告する。

マッハ数が低く非圧縮性が満される条件のもとで、lattice Boltzmann コードと非圧縮差分コードとの比較を行なった。差分コードには、アルゴリズムに HSMAC スキーム[8] および移流項に QUICK スキーム[8]を用いたものを使用した。

計算領域は 2 次元周期境界条件の正方形領域 ($0 \leq x, y < 1$) とし、差分コードでは lattice Boltzmann コードと同数のノードをもった長方形グリッドを用いた。また、初期条件は以下のも

のを使用した[3]。

$$\rho = 1.0 \tag{14}$$

$$u_x = \begin{cases} \tanh((y-0.25)/\varepsilon) & \text{for } y \leq 0.5 \\ \tanh((0.75-y)/\varepsilon) & \text{for } y > 0.5 \end{cases} \tag{15}$$

$$u_y = \delta \sin(2\pi x) \tag{16}$$

Fig.3.1 のように、x 方向に ± 1 の流れがあり、厚さ ε の領域をはさんで速度が逆転している。速度逆転領域での激しいシェアのため $u_y = 0$ の解は物理的に不安定であり、その不安定性の種として y 方向に振幅 δ のサイン型の摂動を加えてある。

レイノルズ数 R の決定に必要な系を特徴づける長さ L としては、摂動の波長 $(=1)$ を用いた。いずれの比較計算でも $R = 10^4$、$\delta = 1/20$ とした。Lattice Boltzmann コードのマッハ数は 0.01 とし、非圧縮の条件が満されているかどうかは、計算前後での密度を比較することにより確認している。

Fig.3.2、Fig.3.3 は、それぞれ差分コード、lattice Boltzmann コードでの $\varepsilon = 1/30$、$t = 1.6$ での渦度の等値線図である。グリッドはともに 128×148 のものを用いた。Lattice Boltzmann コードでは渦の中心付近に歯車のような構造が見られるが、差分コードの結果にはそのような構造は無い。Fig 3.4 にグリッドを 2 倍にした 256×296 での差分コードの結果を示す。この図には、はっきりとした構造が現われている。したがって、この例題計算の条件では、lattice Boltzmann コードが差分コードより高精度に計算できていることが分かる。

次に、$\varepsilon = 0$ での計算を行なった。この条件は、速度 u_x が逆転する領域の厚さが 0 であり、y 方向に隣合うノードでの速度が ± 1 となり、数値解析的により厳しい初期条件である。Fig.3.5、Fig.3.6 に、それぞれ $t = 0.6$ での差分コード、lattice Boltzmann コードでの結果を示す。グリッドはともに 128×148 である。前例の計算と違い、定性的にもかなり異なった結果が得られた。差分コードの結果は、解として安定に収束したが、lattice Boltzmann では見られない 2 次的な渦が生じている。ε が有限の場合の計算結果と比較することにより、lattice Boltzmann コードで得られた結果のほうが物理的に正しい解をとらえていると思われる。Fig.3.7、Fig.3.8 はグリッドを 2 倍の 256×296 にした同条件での計算結果である。この場合でも、2 つのコードの計算結果の定性的な相違は残っている。

以上 2 つの例ではあるが、高レイノルズ数流れの計算において、lattice Boltzmann コードが同条件の差分コードより高精度に計算できたことを示した。このことがどの程度一般的に言えるかは、今後の検討課題である。

4. 並列化

並列化の方法としては、計算領域を分割して各プロセッサに割当てる領域分割法を用いた。コード上では lattice Boltzmann 方程式 (3) を以下の 2 式に変形して、数値的に解いた。

$$N_i^{temp}(\boldsymbol{x},t) = N_i(\boldsymbol{x},t) + \sum_{j=0}^{6} A_{i,j}\left(N_j(\boldsymbol{x},t) - N_j^{eq}(\boldsymbol{x},t)\right) \tag{17}$$

$$N_i(\boldsymbol{x},t+1) = N_i^{temp}(\boldsymbol{x}-\mathbf{c}_i,t) \tag{18}$$

(18) 式は粒子の移動による分布関数の変化を表すが、隣接する分割領域の境界ノードからの粒子の流入が常に存在する。したがって (18) 式を解く前に、境界ノードのデータを隣接するプロセッサ間の通信により交換する必要がある。このプロセッサ間通信にかかわる部分以外では、式 (17)、(18) ともに完全にベクトルおよび並列計算が可能である。

ベクトル並列計算機とスカラ並列計算機では、その特性を活かすために、別々の分割法を採用した。

- ベクトル並列 (Fig.4.1)

  ベクトル長を長くするために、y 方向に 1 次元領域分割をし、x 方向にベクトル化した。プロセッサ間通信は各領域の境界で行なわれる。プロセッサ数を増やしても境界の長さは変化しないため、プロセッサ当りの通信量はプロセッサの数には依存しない。

- スカラ並列 (Fig.4.2)

  2 次元領域分割を用いた。プロセッサ間通信は各領域の境界で行なわれる。全計算領域の長さを N、使用するプロセッサの数を P とすると、プロセッサ間通信のデータ量は $4 \times \frac{N}{\sqrt{P}}$ に比例する。したがって、プロセッサ数が増加するとともに通信にかかるコストが削減がされる。

各コードの実行効率を評価するために、以下の 2 つのパラメータを定義する。P を計算に用いたプロセッサ数、T_p をプロセッサ P 個を用いた計算の実行時間とする。

スピードアップ

$$S_p = \frac{T_1}{T_p} \tag{19}$$

プロセッサ数を増加させた時、何倍速く計算できたかの目安。理想的な場合は P (プロセッサ数に比例して速度が増加) になる。

並列化効率

$$E_p = \frac{S_P}{P} = \frac{T_1}{PT_p} \tag{20}$$

スピードアップをプロセッサ数で規格化したもの。理想的な場合は 1 になる。

5. ベクトル並列計算

富士通 VPP500 を用いて、16 プロセッサまでの計算を行なった。

5.1 計算領域固定での計算

2 つのグリッド 640×640、1152×1152 に対して, プロセッサ数を増加させながら、実行時間の測定を行なった。以下の数値は 10,000 時間ステップでの実行時間†である。

Table 5.1 に計算時間を、Table 5.2 に並列化効率を示す。Fig 5.1 がスピードアップのグラフである。プロセッサ数の増加とともに、実行時間全体に対するプロセッサ間通信にかかわる時間の割合が増え、並列化の効果が減少している。16 プロセッサでの実行では並列化効率は、640×640 のケースで 88.8 %、1152×1152 のケースで 95.1 % に低下している。

これらの測定結果を考えるために、我々は以下のパフォーマンス・モデルをたてた。グリッドを $N \times N$ とする。全実行時間は、各プロセッサ領域の境界間のデータ通信にかかる時間 T_{com} とそれ以外の主要計算部 T_{cal} の和に分けることができる。主要計算部では計算は完全に並列に実行され、計算時間は各プロセッサ領域に含まれるノード数に比例すると考えられる。

$$T_{cal} \propto N \times \frac{N}{P} = \frac{N^2}{P} \tag{21}$$

また、通信にかかる時間は、通信データ量に比例する部分とデータ量に依存しないオーバヘッドとの和として

$$T_{com} \propto \alpha + \beta N \tag{22}$$

とした。したがって、全計算時間は、

$$T_p = \left(\gamma \frac{N^2}{P} + \alpha + \beta N \right) \times \text{時間ステップ数} \tag{23}$$

となる。パラメータ (α, β, γ) は実測値をもとに以下の値とした。

$$\begin{aligned} \alpha &= 4.07 \times 10^{-4} \\ \beta &= 1.67 \times 10^{-7} \\ \gamma &= 1.23 \times 10^{-7} \end{aligned} \tag{24}$$

Fig.5.2 は実測された実行時間と我々のモデルとの比較である。このグラフから分かる通り、2 つのグリッドでの実測値に完全にフィットさせることは困難であった。しかし、コードの定性的な性能特性、定量的な性質のオーダーを調べるために、このパフォーマンス・モデルは有用である。実測値とモデルの不一致の原因は、モデル式 (23) で省略したベクトル・レジスタ長などの影響の他に、プロセッサ間通信にかかる時間が他のユーザのプロセスなどの測定時の計算機の状況にかなり依存すること、また T_{cal} と T_{com} の桁数にひらきがあることなどが考えられる。

†以下、計算実行時間とは、Paragon 上のものも含めて、計算結果の出力などのディスクへのアクセス時間は差し引いたものである。

5.2 プロセッサあたりのメモリ量固定での計算

プロセッサあたりのメモリ量を固定したまま、プロセッサ数 P を増加させた場合を考える。$\tilde{N}$ を固定して、グリッドを $\tilde{N}\sqrt{P} \times \tilde{N}\sqrt{P}$ と大きくする。プロセッサあたりの計算に必要なメモリ量は、

$$\tilde{N}\sqrt{P} \times \frac{\tilde{N}\sqrt{P}}{P} = \tilde{N}^2 \tag{25}$$

に比例するのでプロセッサ数に依存しない。

大容量メモリという分散型並列計算機の特徴を利用した大規模計算を考える場合、このような条件でのパフォーマンスの測定は重要である。今回の測定では、$\tilde{N}$ を 1020 として測定を行なった。この値は、倍精度計算の場合にプロセッサあたりの使用可能メモリ (200 MB) からとれる最大値に近いものである。

Table 5.3 に 10,000 時間ステップでの実行時間を示す。プロセッサ数の増加とともに、実行時間も増加していることが分る。パフォーマンス・モデル (23) 式に、$N = \tilde{N}\sqrt{P}$ を代入すると

$$T_p = \left(\gamma\tilde{N}^2 + \alpha + \beta\tilde{N}\sqrt{P}\right) \times \text{時間ステップ数} \tag{26}$$

となり、$\sqrt{P}$ に比例して通信時間が増えることが予想できる。上式と Table 5.3 を重ねたものを Fig.5.3 に示す。実測値とパフォーマンス・モデルより求めた値はあまりよくフィットしていないが、オーダーの目安として、同様のグラフでプロセッサ数を 500 台まで外挿したものを Fig.5.4 に示す。このグラフでは、縦軸は 1 プロセッサでの実行時間に対する比としている。このグラフより、プロセッサあたりのメモリーを全て使用するような大規模計算を行なう場合に、プロセッサを数 100 台まで増やしても、実行時間の増加は数パーセントのオーダーにとどまることが予測できる。

6. スカラ並列計算

Intel Paragon を用いて、100 プロセッサまでの計算を行なった。

6.1 計算領域固定での計算

2 つのグリッド 640×640、800×800 に対して、プロセッサ数を増加させながら、実行時間の測定を行なった。

Table 6.1 に計算時間を、Table 6.2 に並列化効率を示す。Fig. 6.1 がスピードアップのグラフである[†]。並列化法として 2 次元領域分割を用いたため、100 プロセッサでも 82.8 %、88.6 % という高い並列化効率が得られた。

ベクトル並列の場合と同様に、パフォーマンス・モデルをたてた。$N \times N$ グリッドを考える。主計算部分はベクトル並列と同様に各プロセッサ領域に含まれるノード数に比例し

[†]スカラ並列コードは、並列化ライブラリとしてメッセージ・パッシング型の NX ライブラリを用いたため、1 プロセッサでの実行はできない。このため、$T_1 = T_4/4$ としてスピードアップ、並列化効率を求めた。

5.2 プロセッサあたりのメモリ量固定での計算

プロセッサあたりのメモリ量を固定したまま、プロセッサ数 P を増加させた場合を考える。$\tilde{N}$ を固定して、グリッドを $\tilde{N}\sqrt{P} \times \tilde{N}\sqrt{P}$ と大きくする。プロセッサあたりの計算に必要なメモリ量は、

$$\tilde{N}\sqrt{P} \times \frac{\tilde{N}\sqrt{P}}{P} = \tilde{N}^2 \tag{25}$$

に比例するのでプロセッサ数に依存しない。

大容量メモリという分散型並列計算機の特徴を利用した大規模計算を考える場合、このような条件でのパフォーマンスの測定は重要である。今回の測定では、$\tilde{N}$ を 1020 として測定を行なった。この値は、倍精度計算の場合にプロセッサあたりの使用可能メモリ (200 MB) からとれる最大値に近いものである。

Table 5.3 に 10,000 時間ステップでの実行時間を示す。プロセッサ数の増加とともに、実行時間も増加していることが分る。パフォーマンス・モデル (23) 式に、$N = \tilde{N}\sqrt{P}$ を代入すると

$$T_p = \left(\gamma\tilde{N}^2 + \alpha + \beta\tilde{N}\sqrt{P}\right) \times \text{時間ステップ数} \tag{26}$$

となり、$\sqrt{P}$ に比例して通信時間が増えることが予想できる。上式と Table 5.3 を重ねたものを Fig.5.3 に示す。実測値とパフォーマンス・モデルより求めた値はあまりよくフィットしていないが、オーダーの目安として、同様のグラフでプロセッサ数を 500 台まで外挿したものを Fig.5.4 に示す。このグラフでは、縦軸は 1 プロセッサでの実行時間に対する比としている。このグラフより、プロセッサあたりのメモリーを全て使用するような大規模計算を行なう場合に、プロセッサを数 100 台まで増やしても、実行時間の増加は数パーセントのオーダーにとどまることが予測できる。

6. スカラ並列計算

Intel Paragon を用いて、100 プロセッサまでの計算を行なった。

6.1 計算領域固定での計算

2 つのグリッド 640×640、800×800 に対して、プロセッサ数を増加させながら、実行時間の測定を行なった。

Table 6.1 に計算時間を、Table 6.2 に並列化効率を示す。Fig. 6.1 がスピードアップのグラフである[‡]。並列化法として 2 次元領域分割を用いたため、100 プロセッサでも 82.8 %、88.6 % という高い並列化効率が得られた。

ベクトル並列の場合と同様に、パフォーマンス・モデルをたてた。$N \times N$ グリッドを考える。主計算部分はベクトル並列と同様に各プロセッサ領域に含まれるノード数に比例し

[‡]スカラ並列コードは、並列化ライブラリとしてメッセージ・パッシング型の NX ライブラリを用いたため、1 プロセッサでの実行はできない。このため、$T_1 = T_4/4$ としてスピードアップ、並列化効率を求めた。

$$T_{cal} \propto \frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} = \frac{N^2}{P} \tag{27}$$

となる。また、プロセッサ境界は $\frac{N}{\sqrt{P}}$ に比例するため、通信時間は、

$$T_{com} \propto \alpha + \beta \frac{N}{\sqrt{P}} \tag{28}$$

となる。したがって、全計算時間は、

$$T_p = \left(\gamma \frac{N^2}{P} + \alpha + \beta \frac{N}{\sqrt{P}} \right) \times \text{時間ステップ数} \tag{29}$$

となる。パラメータ (α, β, γ) は実測値をもとに以下の値とした。

$$\begin{aligned} \alpha &= 6.15 \times 10^{-3} \\ \beta &= 1.41 \times 10^{-4} \\ \gamma &= 1.75 \times 10^{-5} \end{aligned} \tag{30}$$

Fig.6.2 は実測された実行時間と我々のモデルとの比較である。

6.2 プロセッサあたりのメモリ量固定での計算

プロセッサ数 P を増加させるとともに、グリッドも $\tilde{N}\sqrt{P} \times \tilde{N}\sqrt{P}$ と大きくする場合の実行時間を測定した。$\tilde{N}$ としては、倍精度計算の場合にプロセッサあたりの使用可能メモリ (24 MB) からとれる最大値に近い 400 としてた。1,000 時間ステップの実行を行なった。

Table 6.3 に測定結果を示す。数値に変動はあるが、プロセッサ数に対する増加傾向は見られない。パフォーマンス・モデル (29) 式に、$N = \tilde{N}\sqrt{P}$ を代入すると

$$T_p = \left(\gamma \tilde{N}^2 + \alpha + \beta \tilde{N} \right) \times \text{時間ステップ数} \tag{31}$$

となり、計算時間がプロセッサ数に依存しないことが分かる。Table 6.3 の測定値の変動の大きさは 0.3 % と小さく、2 次元領域分割の利点を良く表している。

7. ベクトル並列とスカラ並列の比較

ベクトル並列コードとスカラ並列コードの比較を行なうために、以下のような規格化された実行時間 $\tilde{T}_p$ を定義する。

$$\tilde{T}_p = \frac{T_p}{\text{ノード数} \times \text{時間ステップ数}} \tag{32}$$

これは、1 ノードを更新するのに必要な時間である。

グリッド 640×640 と 1152×1152 におけるベクトル並列計算 (VPP500)、グリッド 640×640 と 800×800 におけるスカラ並列計算 (Paragon) での実行時間を規格化したものを Fig.7.1 に示す。このグラフより、同プロセッサ数での実行では、ベクトル並列コードがスカラ並列コードよ

$$T_{cal} \propto \frac{N}{\sqrt{P}} \times \frac{N}{\sqrt{P}} = \frac{N^2}{P} \tag{27}$$

となる。また、プロセッサ境界は $\frac{N}{\sqrt{P}}$ に比例するため、通信時間は、

$$T_{com} \propto \alpha + \beta \frac{N}{\sqrt{P}} \tag{28}$$

となる。したがって、全計算時間は、

$$T_p = \left(\gamma \frac{N^2}{P} + \alpha + \beta \frac{N}{\sqrt{P}} \right) \times \text{時間ステップ数} \tag{29}$$

となる。パラメータ (α, β, γ) は実測値をもとに以下の値とした。

$$\begin{aligned} \alpha &= 6.15 \times 10^{-3} \\ \beta &= 1.41 \times 10^{-4} \\ \gamma &= 1.75 \times 10^{-5} \end{aligned} \tag{30}$$

Fig.6.2 は実測された実行時間と我々のモデルとの比較である。

6.2 プロセッサあたりのメモリ量固定での計算

プロセッサ数 P を増加させるとともに、グリッドも $\tilde{N}\sqrt{P} \times \tilde{N}\sqrt{P}$ と大きくする場合の実行時間を測定した。$\tilde{N}$ としては、倍精度計算の場合にプロセッサあたりの使用可能メモリ (24 MB) からとれる最大値に近い 400 としてた。1,000 時間ステップの実行を行なった。

Table 6.3 に測定結果を示す。数値に変動はあるが、プロセッサ数に対する増加傾向は見られない。パフォーマンス・モデル (29) 式に、$N = \tilde{N}\sqrt{P}$ を代入すると

$$T_p = \left(\gamma \tilde{N}^2 + \alpha + \beta \tilde{N} \right) \times \text{時間ステップ数} \tag{31}$$

となり、計算時間がプロセッサ数に依存しないことが分かる。Table 6.3 の測定値の変動の大きさは 0.3 % と小さく、2 次元領域分割の利点を良く表している。

7. ベクトル並列とスカラ並列の比較

ベクトル並列コードとスカラ並列コードの比較を行なうために、以下のような規格化された実行時間 $\tilde{T}_p$ を定義する。

$$\tilde{T}_p = \frac{T_p}{\text{ノード数} \times \text{時間ステップ数}} \tag{32}$$

これは、1 ノードを更新するのに必要な時間である。

グリッド 640×640 と 1152×1152 におけるベクトル並列計算 (VPP500)、グリッド 640×640 と 800×800 におけるスカラ並列計算 (Paragon) での実行時間を規格化したものを Fig.7.1 に示す。このグラフより、同プロセッサ数での実行では、ベクトル並列コードがスカラ並列コードよ

り約 100 倍高速なことが分かる。単一プロセッサでのピーク性能は、VPP500 が 1.6 GFLOPS、Paragon XP/S が 75 MFLOPS であり、その比 21 より大きな性能差が得られた。それぞれのパフォーマンス・モデルでの、主要計算部 (プロセッサ間通信以外の部分) の比較をすると、式 (24)、(30) より、

$$\frac{\gamma(\mathrm{VPP})}{\gamma(\mathrm{Paragon})} = 1.4 \times 10^{-2} \tag{33}$$

となり、全体の速度比とほぼ同じになる。また、ともに 16 プロセッサまで範囲で並列化効率が 90 % 以上でていることから、2 つのコードの性能の比は、プロセッサ間通信にかかる時間ではなく、主計算部の計算速度に多く依存することが分かる。スカラ並列コードの主計算部をさらにチューニングすることにより、2 つのコードの同数プロセッサでの速度比を 1 プロセッサのピーク性能比により近づけることが可能であると思われる。

Fig.7.2 に、ベクトル並列計算による 1152×1152 グリッドとスカラ並列計算による 800×800 グリッドの結果、および、それぞれのパフォーマンス・モデルから得られる値をグラフにした。このグラフでは横軸のプロセッサ数を 1,000 まで外挿して、現在では不可能なプロセッサ数[§]での計算性能の予想ができるようにした。VPP500 上で 1152×1152 グリッドのベクトル並列計算は、プロセッサ数 50 あたりから速度向上の飽和が見られる。しかし、Paragon 上での 100 プロセッサのスカラ並列計算との速度比は依然として 100 倍程度ある。一方、Paragon 上での 800×800 グリッド のスカラ並列計算は、プロセッサ数 1,000 でもあまり速度向上の飽和は見られず、2 次元領域分割の特性がよく現れている。

8. 結論

Lattice Boltzmann 法の並列化では計算全体に対するプロセッサ間通信に関わる部分の割合が小さいため、1 次元領域分割を用いたベクトル並列コード、2 次元領域分割を用いたスカラ並列コードともに高い並列化効率が得られた。

プロセッサ間通信以外の部分は、並列化だけでなく完全にベクトル化が可能である。そのため、ベクトル並列コードは同プロセッサ数のスカラ並列コードに比べて約 100 倍の速度比が得られた。1 プロセッサでのピーク性能の比と比較することにより、スカラ並列コードは主要計算部をチューニングすることにより計算速度の向上が計れる可能性がある。

パフォーマンス・モデルによると、ベクトル並列コードの速度向上は、1152×1152 グリッド の場合プロセッサ 50 台あたりから飽和が予測される。しかし、このような計算領域を固定してプロセッサに分割するという条件での性能評価は、大規模領域あるいは細密メッシュでの数値実験という立場から見るとあまり現実的ではない。このような立場では、5.2 章、6.2 章で述べたように、プロセッサごとの使用メモリを最大量に固定し、プロセッサ数の増加に比例して計算領域も増やす条件での計算性能がより重要である。5.2 章において、VPP500 のプロセッサが数百台になって

[§]使用した VPP500 の総プロセッサ数は 42 台、Paragon XP/S は 256 台である。このうち現在一般ユーザが並列計算で使用できる上限は、それぞれ 16 台と 128 台である。

り約 100 倍高速なことが分かる。単一プロセッサでのピーク性能は、VPP500 が 1.6 GFLOPS、Paragon XP/S が 75 MFLOPS であり、その比 21 より大きな性能差が得られた。それぞれのパフォーマンス・モデルでの、主要計算部 (プロセッサ間通信以外の部分) の比較をすると、式 (24)、(30) より、

$$\frac{\gamma(\mathrm{VPP})}{\gamma(\mathrm{Paragon})} = 1.4 \times 10^{-2} \tag{33}$$

となり、全体の速度比とほぼ同じになる。また、ともに 16 プロセッサまで範囲で並列化効率が 90 % 以上でていることから、2 つのコードの性能の比は、プロセッサ間通信にかかる時間ではなく、主計算部の計算速度に多く依存することが分かる。スカラ並列コードの主計算部をさらにチューニングすることにより、2 つのコードの同数プロセッサでの速度比を 1 プロセッサのピーク性能比により近づけることが可能であると思われる。

Fig.7.2 に、ベクトル並列計算による 1152×1152 グリッドとスカラ並列計算による 800×800 グリッドの結果、および、それぞれのパフォーマンス・モデルから得られる値をグラフにした。このグラフでは横軸のプロセッサ数を 1,000 まで外挿して、現在では不可能なプロセッサ数[§]での計算性能の予想ができるようにした。VPP500 上で 1152×1152 グリッドのベクトル並列計算は、プロセッサ数 50 あたりから速度向上の飽和が見られる。しかし、Paragon 上での 100 プロセッサのスカラ並列計算との速度比は依然として 100 倍程度ある。一方、Paragon 上での 800×800 グリッド のスカラ並列計算は、プロセッサ数 1,000 でもあまり速度向上の飽和は見られず、2 次元領域分割の特性がよく現れている。

8. 結論

Lattice Boltzmann 法の並列化では計算全体に対するプロセッサ間通信に関わる部分の割合が小さいため、1 次元領域分割を用いたベクトル並列コード、2 次元領域分割を用いたスカラ並列コードともに高い並列化効率が得られた。

プロセッサ間通信以外の部分は、並列化だけでなく完全にベクトル化が可能である。そのため、ベクトル並列コードは同プロセッサ数のスカラ並列コードに比べて約 100 倍の速度比が得られた。1プロセッサでのピーク性能の比と比較することにより、スカラ並列コードは主要計算部をチューニングすることにより計算速度の向上が計れる可能性がある。

パフォーマンス・モデルによると、ベクトル並列コードの速度向上は、1152×1152 グリッド の場合プロセッサ 50 台あたりから飽和が予測される。しかし、このような計算領域を固定してプロセッサに分割するという条件での性能評価は、大規模領域あるいは細密メッシュでの数値実験という立場から見るとあまり現実的ではない。このような立場では、5.2 章、6.2 章で述べたように、プロセッサごとの使用メモリを最大量に固定し、プロセッサ数の増加に比例して計算領域も増やす条件での計算性能がより重要である。5.2 章において、VPP500 のプロセッサが数百台になって

[§]使用した VPP500 の総プロセッサ数は 42 台、Paragon XP/S は 256 台である。このうち現在一般ユーザが並列計算で使用できる上限は、それぞれ 16 台と 128 台である。

も計算時間の増加は数パーセントと予測された。したがって、この意味において、1 次元領域分割を用いたベクトル並列コードはプロセッサが数百台となっても充分実用的である。

3 章で行なった比較計算の範囲では、lattice Boltzmann コードは同数のグリッドの差分コードより高精度に計算できた。これにより、lattice Boltzmann 法を用いた高レイノルズ数流れのシミュレーションの可能性を示すことができた。

謝 辞

(株) 情報数理研究所の飯塚幹夫氏には、非圧縮差分コードを提供していただいたことを感謝します。

参 考 文 献

1) Lattice gas 法、lattice Boltzmann 法の論文集および雑誌の特集号は、
Doolen, G. D. eds : "Lattice Gas Methods for Partial Differential Equations", Addison-Wesley, New York, (1989).
Physica D, **47**, no 1-2, (1991).
J. Stat. Phys., **68**, no 3/4, (1992).

2) Punzo, G. Massaioli, F. and Succi, S. : Comput. Phys., **8**, 706 (1994).

3) McNamara, R. and Alder, B. : private communication.

4) Higuera, F. J., Succi, S. and Benzi, R. : Europhys. Lett., **9**, 345 (1989).

5) Frisch, U., d'Humieres, D., Hasslacher, B., Lallemand, P., Pomeau, Y. and River, J. P. : Complex Systems, **1**, 649 (1987).

6) Wolfram, S. : J. Stat. Phys., **45**, 471 (1986).

7) Benzi, R., Succi, S. and Vergassola, M. : Phys. Rep., **222**, 145 (1992).

8) 保原充, 大宮司久明 編: "数値流体力学", 東京大学出版会, (1992).

も計算時間の増加は数パーセントと予測された。したがって、この意味において、1 次元領域分割を用いたベクトル並列コードはプロセッサが数百台となっても充分実用的である。

3 章で行なった比較計算の範囲では、lattice Boltzmann コードは同数のグリッドの差分コードより高精度に計算できた。これにより、lattice Boltzmann 法を用いた高レイノルズ数流れのシミュレーションの可能性を示すことができた。

謝 辞

(株) 情報数理研究所の飯塚幹夫氏には、非圧縮差分コードを提供していただいたことを感謝します。

参 考 文 献

1) Lattice gas 法、lattice Boltzmann 法の論文集および雑誌の特集号は、
Doolen, G. D. eds : "Lattice Gas Methods for Partial Differential Equations", Addison-Wesley, New York, (1989).
Physica D, **47**, no 1-2, (1991).
J. Stat. Phys., **68**, no 3/4, (1992).

2) Punzo, G. Massaioli, F. and Succi, S. : Comput. Phys., **8**, 706 (1994).

3) McNamara, R. and Alder, B. : private communication.

4) Higuera, F. J., Succi, S. and Benzi, R. : Europhys. Lett., **9**, 345 (1989).

5) Frisch, U., d'Humieres, D., Hasslacher, B., Lallemand, P., Pomeau, Y. and River, J. P. : Complex Systems, **1**, 649 (1987).

6) Wolfram, S. : J. Stat. Phys., **45**, 471 (1986).

7) Benzi, R., Succi, S. and Vergassola, M. : Phys. Rep., **222**, 145 (1992).

8) 保原充, 大宮司久明 編: "数値流体力学", 東京大学出版会, (1992).

も計算時間の増加は数パーセントと予測された。したがって、この意味において、1 次元領域分割を用いたベクトル並列コードはプロセッサが数百台となっても充分実用的である。

3 章で行なった比較計算の範囲では、lattice Boltzmann コードは同数のグリッドの差分コードより高精度に計算できた。これにより、lattice Boltzmann 法を用いた高レイノルズ数流れのシミュレーションの可能性を示すことができた。

謝 辞

(株) 情報数理研究所の飯塚幹夫氏には、非圧縮差分コードを提供していただいたことを感謝します。

参 考 文 献

1) Lattice gas 法、lattice Boltzmann 法の論文集および雑誌の特集号は、
Doolen, G. D. eds : "Lattice Gas Methods for Partial Differential Equations", Addison-Wesley, New York, (1989).
Physica D, **47**, no 1-2, (1991).
J. Stat. Phys., **68**, no 3/4, (1992).

2) Punzo, G. Massaioli, F. and Succi, S. : Comput. Phys., **8**, 706 (1994).

3) McNamara, R. and Alder, B. : private communication.

4) Higuera, F. J., Succi, S. and Benzi, R. : Europhys. Lett., **9**, 345 (1989).

5) Frisch, U., d'Humieres, D., Hasslacher, B., Lallemand, P., Pomeau, Y. and River, J. P. : Complex Systems, **1**, 649 (1987).

6) Wolfram, S. : J. Stat. Phys., **45**, 471 (1986).

7) Benzi, R., Succi, S. and Vergassola, M. : Phys. Rep., **222**, 145 (1992).

8) 保原充, 大宮司久明 編: "数値流体力学", 東京大学出版会, (1992).

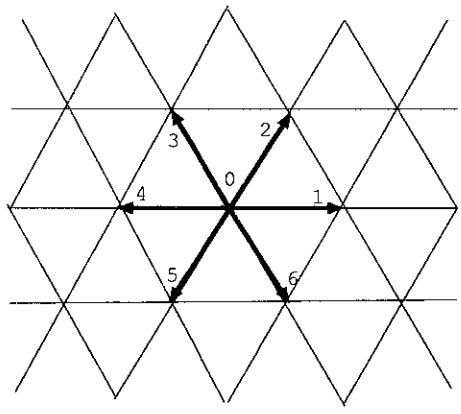

Fig.2.1 2–D triangular lattice and seven velocity vectors.

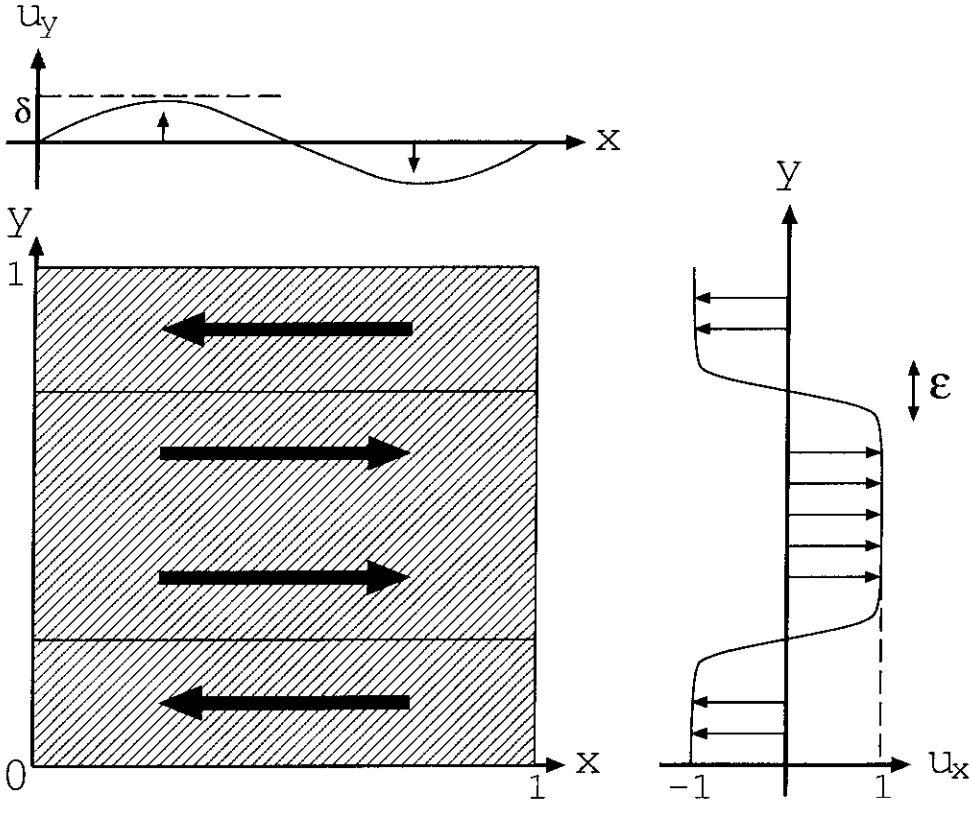

Fig.3.1 The initial conditions for a comparision of a lattice Boltzmann code and a finite difference code.

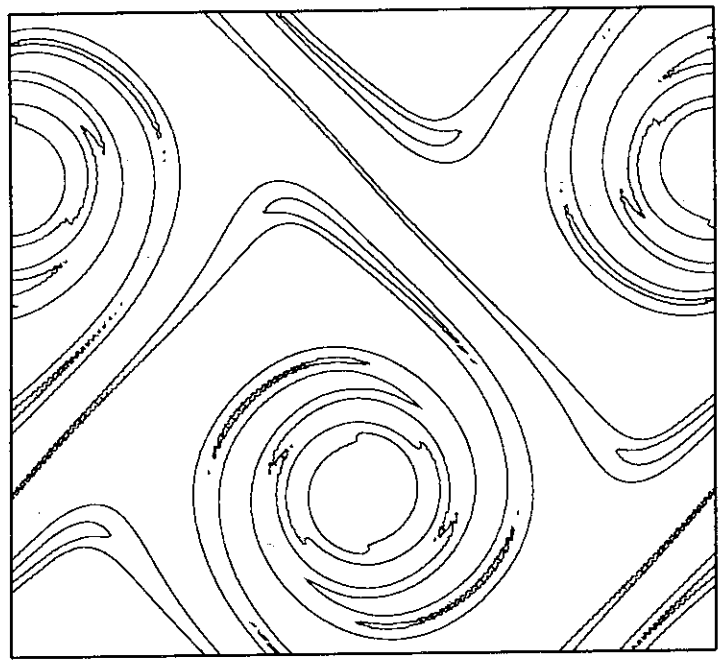

Fig.3.2 Vorticity contours produced by (128 × 148) lattice Boltzmann system. $R = 10^4$, $\varepsilon = 1/30$, $\delta = 1/20$, $t = 1.6$.

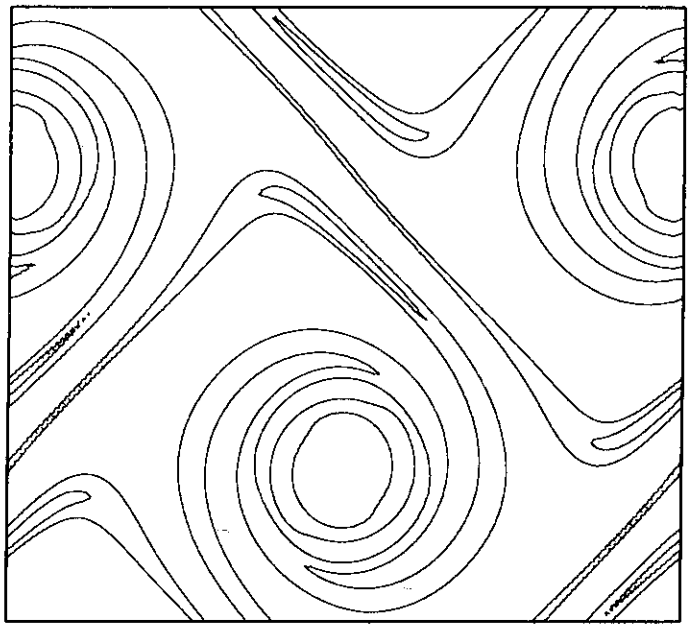

Fig.3.3 Vorticity contours produced by (128 × 148) finite difference grid under the same conditions as Fig.3.3. $R = 10^4$, $\varepsilon = 1/30$, $\delta = 1/20$, $t = 1.6$.

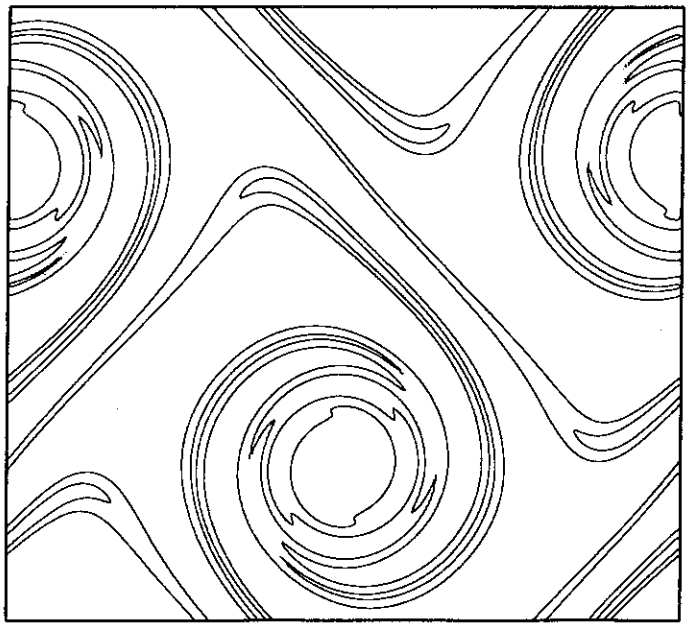

Fig.3.4 Vorticity contours produced by (256 × 296) finite difference grid under the same conditions as Fig.3.3. $R = 10^4$, $\varepsilon = 1/30$, $\delta = 1/20$, $t = 1.6$.

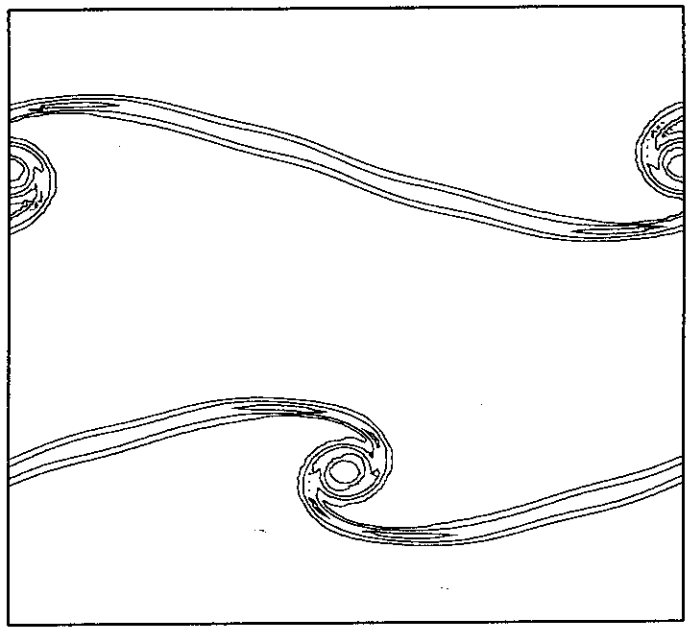

Fig.3.5 Vorticity contours produced by (128 × 148) lattice Boltzmann system. $R = 10^4$, $\varepsilon = 0$, $\delta = 1/20$, $t = 0.6$.

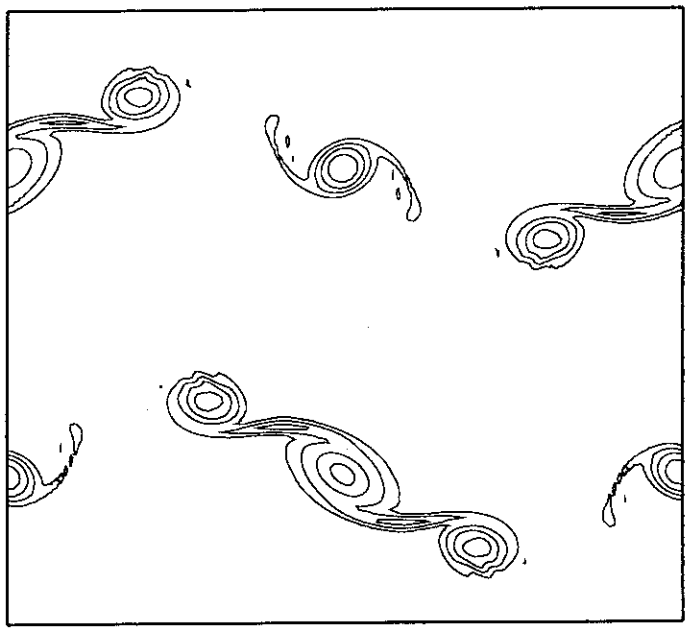

Fig.3.6 Vorticity contours produced by (128 × 148) finite difference grid under the same conditions as Fig.3.5. $R = 10^4$, $\varepsilon = 0$, $\delta = 1/20$, $t = 0.6$.

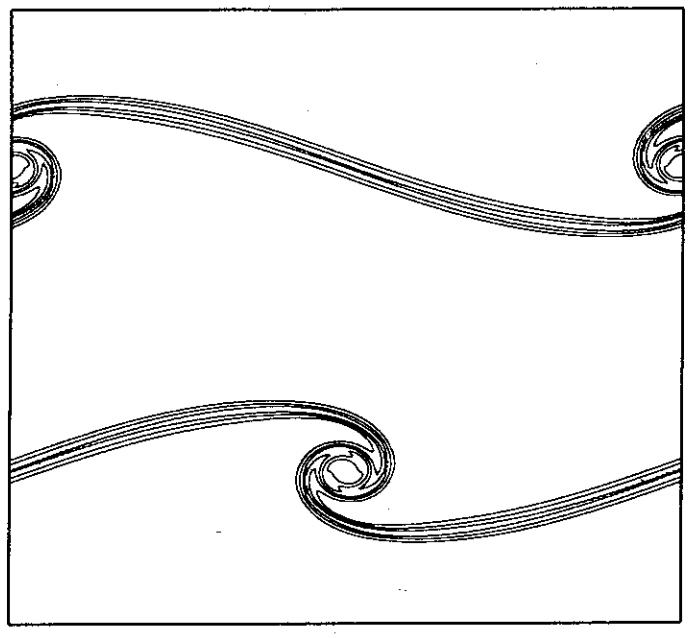

Fig.3.7 Vorticity contours produced by (256×296) lattice Boltzmann system under the same conditions as Fig.3.5. $R = 10^4$, $\varepsilon = 0$, $\delta = 1/20$, $t = 0.6$.

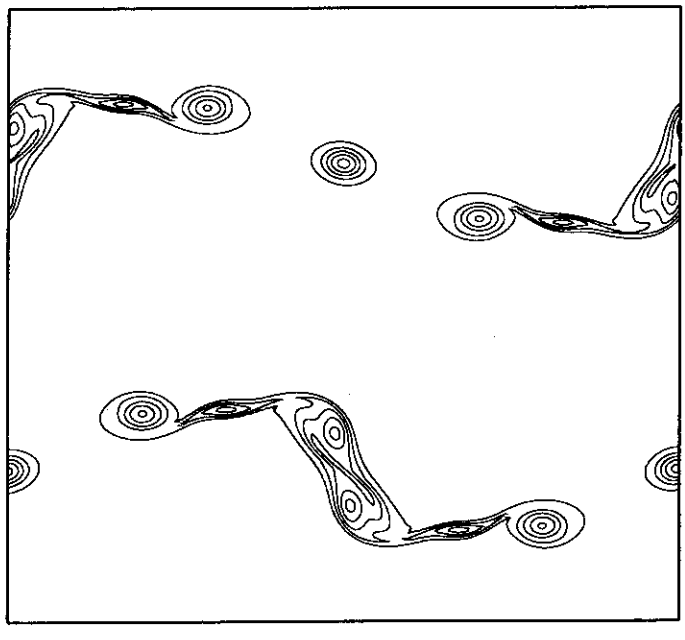

Fig.3.8 Vorticity contours produced by (256×296) finite difference grid under the same conditions as Fig.3.5. $R = 10^4$, $\varepsilon = 0$, $\delta = 1/20$, $t = 0.6$.

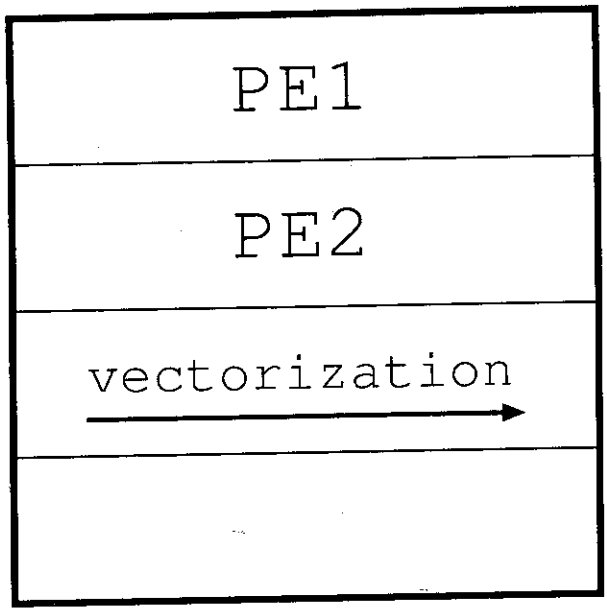

Fig.4.1 1–D domain decomposition for a vector parallel code.

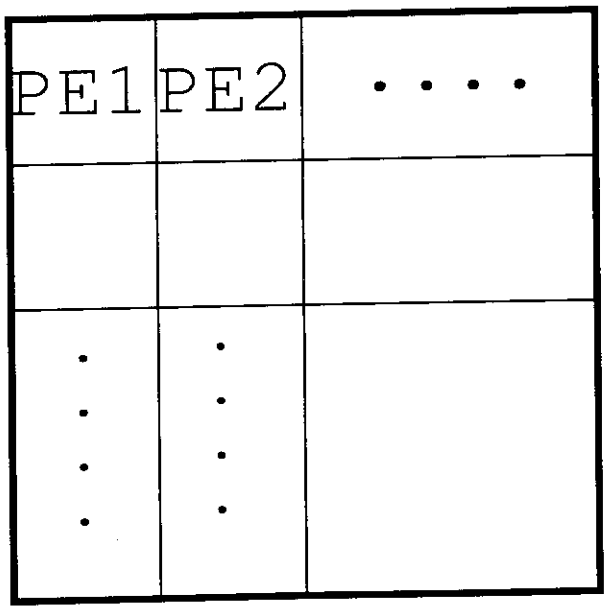

Fig.4.2 2–D domain decomposition for a scalar parallel code.

Table 5.1 Execution times(sec) on VPP500 for (640 × 640) grid and (1152 × 1152) grid.

PE	1	2	4	8	16
640 × 640	552.479	278.195	141.447	73.932	38.904
1152 × 1152	1639.370	821.706	413.553	208.923	107.715

Table 5.2 Parallel efficiency on VPP500 for (640 × 640) grid and (1152 × 1152) grid.

PE	2	4	8	16
640 × 640	0.993	0.976	0.934	0.888
1152 × 1152	0.998	0.991	0.981	0.951

Table 5.3 Execution times(sec) on VPP500 for $(1020\sqrt{P} \times 1020\sqrt{P})$ grid.

PE	1	4	9	16
$N = 1020\sqrt{P}$	1020	2040	3060	4080
Time	1285.515	1286.377	1288.327	1290.438

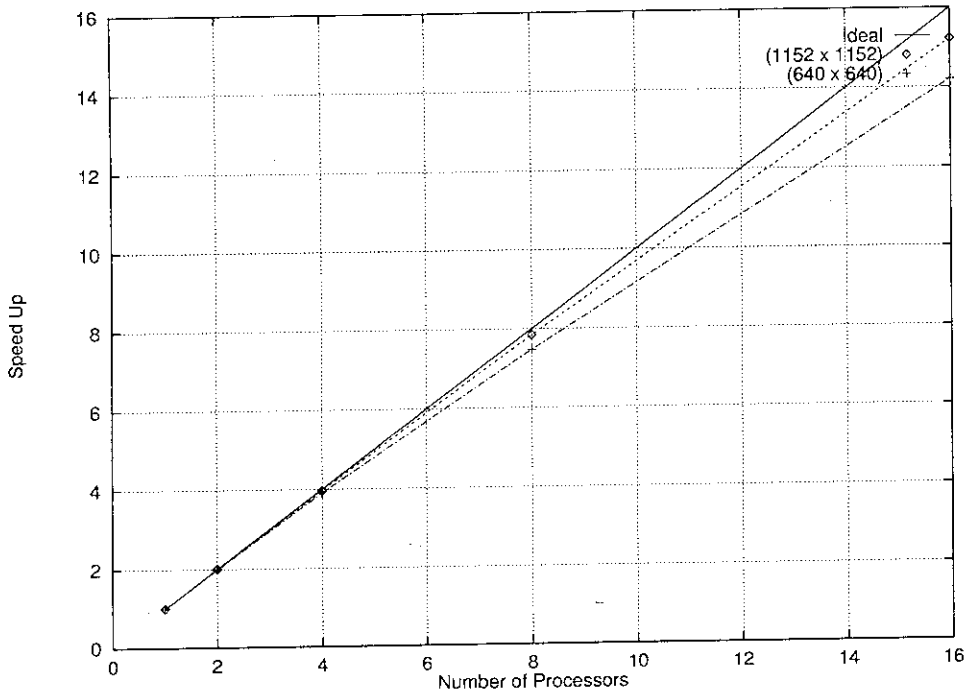

Fig.5.1 Speed up on VPP500 for (640×640) grid and (1152×1152) grid.

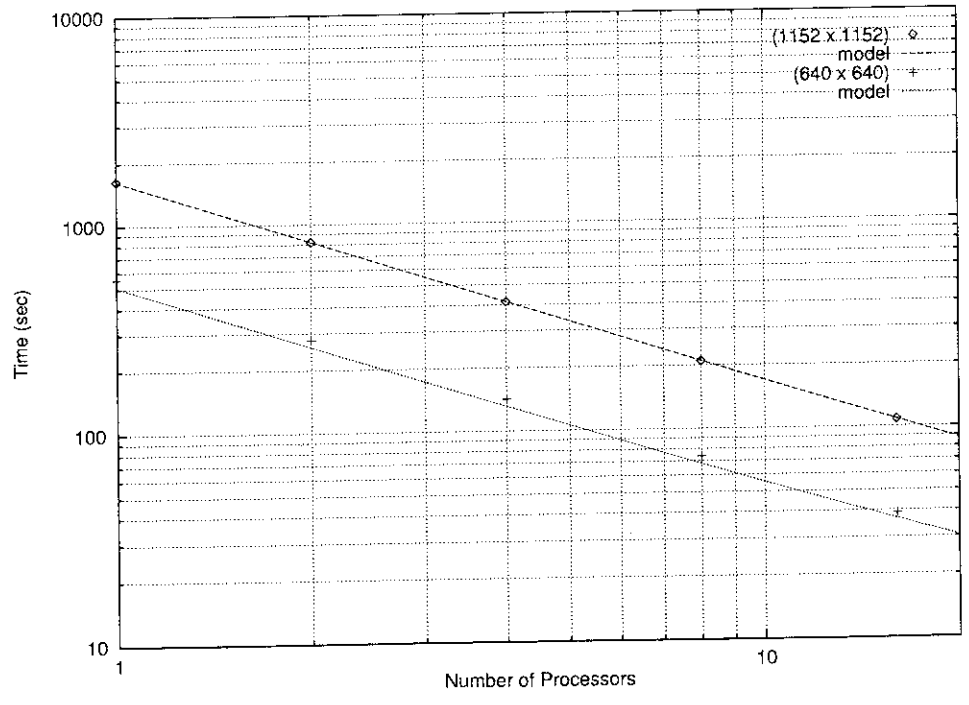

Fig.5.2 Log-log plot of execution times and model estimated values as a function of the number of processors on VPP500 for (640×640) grid and (1152×1152) grid.

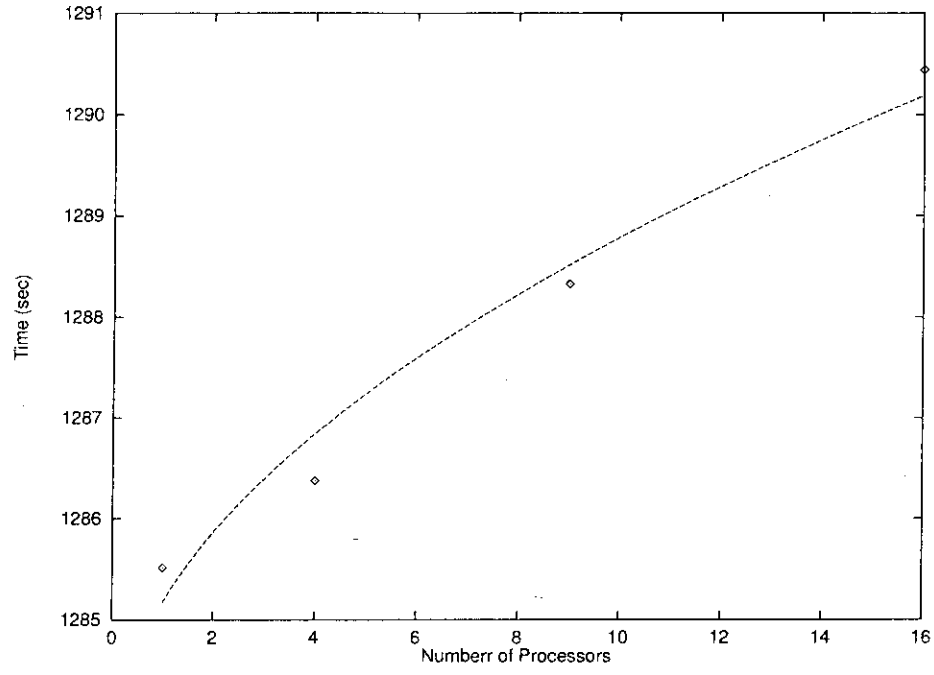

Fig.5.3 Execution times and model estimated values as a function of the mumber of processors on VPP500 for $(1020\sqrt{P} \times 1020\sqrt{P})$ grid.

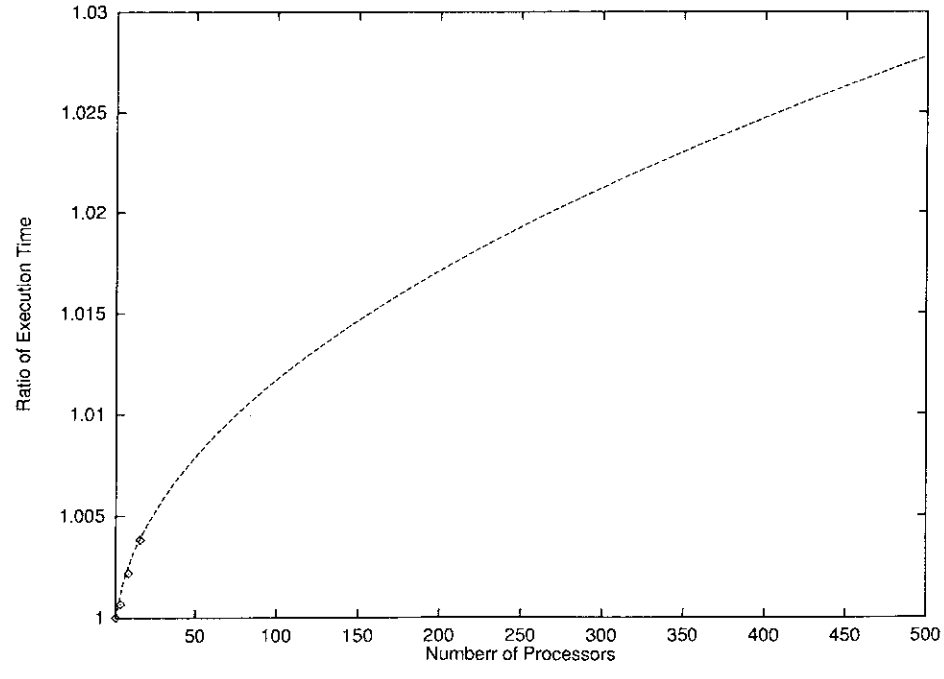

Fig.5.4 Model estimation of T_p/T_1 on VPP500 for $(1020\sqrt{P} \times 1020\sqrt{P})$ grid.

Table 6.1 Execution times(sec) on Paragon for (640 × 640) grid and (800 × 800) grid.

PE	4	16	64	100
640 × 640	1841.023	476.850	126.426	88.968
800 × 800	2681.041	685.776	182.030	121.037

Table 6.2 Parallel efficiency on Paragon for (640 × 640) grid and (800 × 800) grid.

PE	4	16	64	100
640 × 640	1.0	0.965	0.910	0.828
800 × 800	1.0	0.977	0.921	0.886

Table 6.3 Execution times(sec) on Paragon for $(400\sqrt{P} \times 400\sqrt{P})$ grid.

PE	4	16	64	100
$N = 4000\sqrt{P}$	800	1600	3200	40000
Time	2681.133	2682.000	2689.657	2685.258

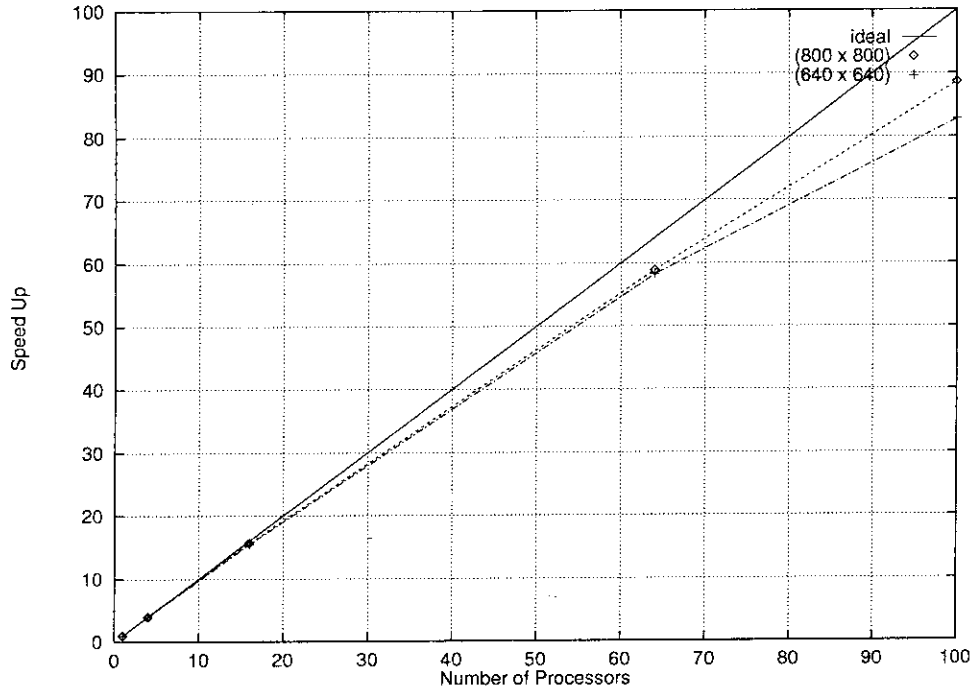

Fig.6.1 Speed up on Paragon for (640 × 640) grid and (800 × 800) grid.

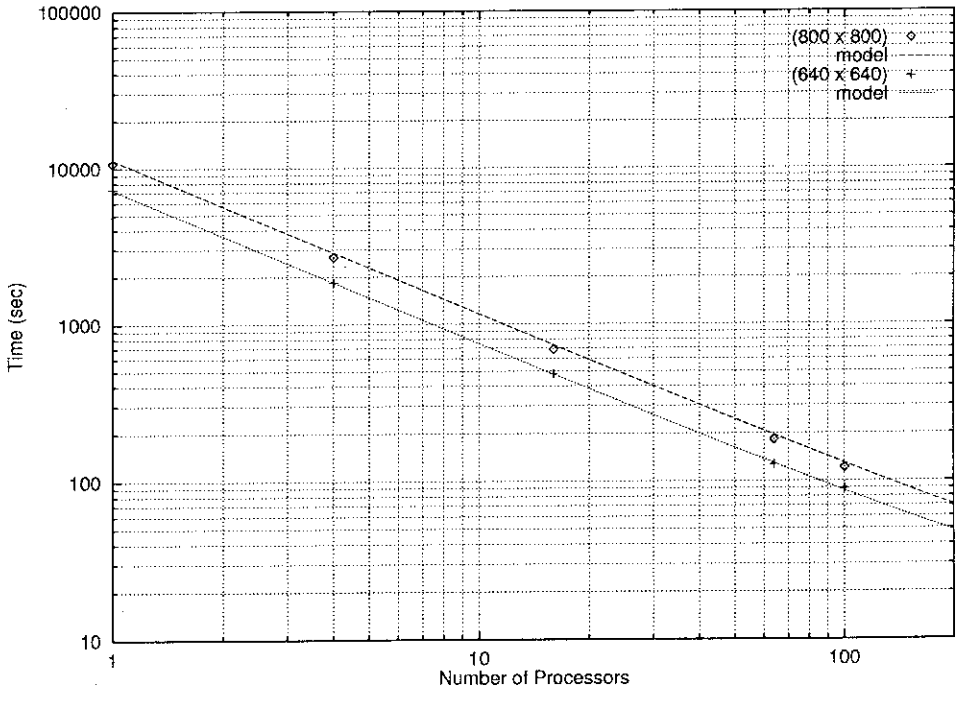

Fig. 6.2 Log-log plot of execution times and model estimated values as a function of the number of processors on Paragon for (640 × 640) grid and (800 × 800) grid.

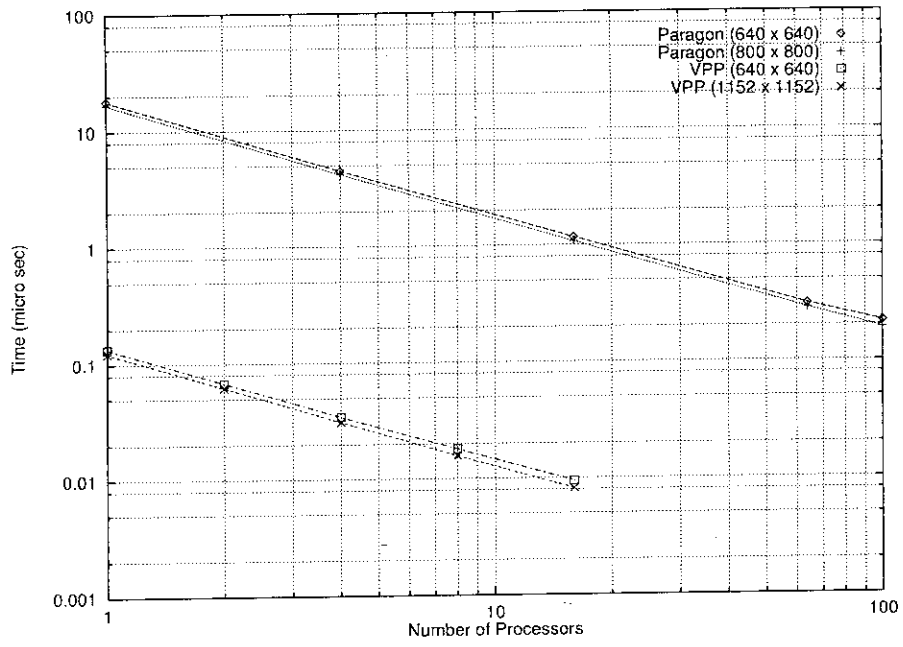

Fig.7.1 Log-log plot of normarized execution times as a function of the number of processors.

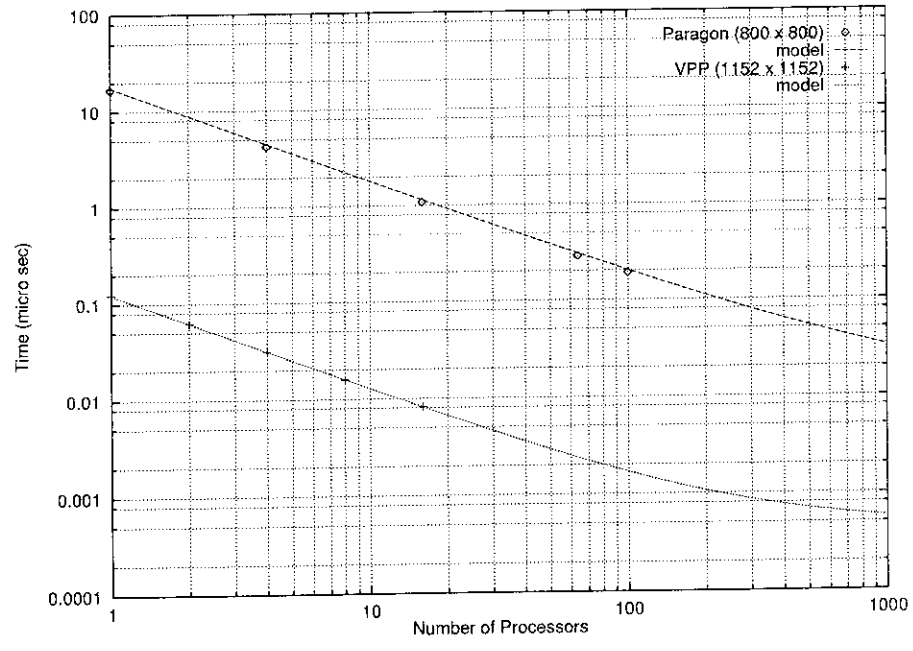

Fig.7.2 Log-log plot of normarized execution times and model estimated values as a function of the number of processors. VPP500 (1152×1152) grid and Paragon (800×800) grid.

付録 2 次元 lattice Boltzmann コード *LB 2* 使用法

VPP500 (ベクトル並列コード)

1. パラメータの設定

   ファイル `param.inc` で以下の定数を設定する。

(Nx, Ny)	全グリッド・サイズ
NP	プロセッサ数

2. コンパイル・リンク

   `make` とするだけで、実行ファイル `lb2` が生成される。

3. 入力ファイル

   以下の 4 行からなるファイルを用意する。

   入力フラグ (I), 出力フラグ (I)
   マッハ数 (R*8)
   レイノルズ数 (R*8)
   時間ステップ数 (I) (入力フラグ=0) または、時間 (R*8) (入力フラグ ≠0)

   出力フラグは 0 の場合、3 角格子上での (u_x, u_y) を出力する。それ以外では、渦度を計算しグラフ表示プログラム gnuplot 用の出力ファイルを出力する。

```
1 1
0.01
1.0D4
1.0
```

   上の例では、マッは数 0.01、レイノルズ数 10^4 の流れの計算を時刻 1.0 まで行ない、gnuplot 用の出力ファイルを生成する。

4. 実行方法

   入力ファイルは標準入力から読み、出力ファイルは器番 10 のファイルに書かれる。また、計算時の情報が標準出力に出力される。以下に、実行スクリプトの例を示す。

```
#!/bin/csh -f
#@$-lPv 4
cd LB2

setenv fu10bf 1024
setenv fu10 ~/vfl/outfile

./lb2  < infile > log
```

   この例では、ファイル類がディレクトリ `LB2` にあるとして、入力ファイル `infile` に対して、4 プロセッサでの実行を行なう。計算結果は、vfl 上の `outfile` に、計算情報はファイル `log` に出力される。

実行させるには、

qsub -q “ジョブクラス名” “スクリプト・ファイル名”

とすればよい。

5. gnuplot による計算結果の表示

出力オプション $\neq 0$ とした計算結果は gnuplot によりグラフ表示できる。出力ファイルを gnuplot のインストールされたワークステーションにコピーし以下のスクリプトを実行すると、3 章で示したのと同様なグラフを見ることができる。

```
#!/usr/local/bin/gnuplot
#    ↑ ↑ ↑ ↑ ↑
#  gnuplot のあるパスに設定して下さい
#
set cont
set nosurf
set view 0.,0.
set data style line
set cntrp level 10
set size 0.721,1.0
set noxtics
set noytics
#set cntrparam levels incremental -90.,20.,110.
#
splot 'outfile'
#      ↑ ↑ ↑ ↑
#      データ・ファイルの名前
#
pause -1 "<Hit return to quit>"
```

Paragon (スカラ並列コード)

1. コンパイル・リンク

make とするだけで、実行ファイル lb2 が生成される。

2. 入力ファイル

以下の 6 行からなるのファイルを用意する。

入力フラグ (I)

x 方向プロセッサ数 (I), y 方向プロセッサ数 (I)

x 方向グリッド数 (I), y 方向グリッド数 (I)

マッハ数 (R*8)

レイノルズ数 (R*8)

時間ステップ数 (I) (入力フラグ=0) または、時間 (R*8) (入力フラグ $\neq$0)

出力は 3 角格子上の (u_x, u_y) のみである。

```
0
4 4
```

```
400 400
0.01
1.0D4
1000
```

上の例では、マッハ数 0.01、レイノルズ数 10^4、(400×400) のグリッドに対して、プロセッサ 4×4 台を用いて 1,000 時間ステップの計算を行なう。

3. 実行方法

入力ファイルは標準入力から読み、出力はファイル `outfile` に書かれる。また、計算時の情報が標準出力に出力される。以下に、実行スクリプトを例を示す。

```
cd LB2
./lb2 -plk -sz 16 < infile  >log
```

この例では、ファイル類がディレクトリ `LB2` にあるとして、入力ファイル `infile` に対して 16 プロセッサでの計算をする。計算結果はファイル `outfile` に、計算情報はファイル `log` に出力される。

実行させるには、

`qsub -q` “ジョブクラス名” “スクリプト・ファイル名”

とすればよい。