JAERI-Data/Code
2000-024

格子ガス気液モデルの シミュレーションコードの並列化

2000年3月

川井　渉*・海老原健・久米悦雄・渡辺　正

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、お申し越し下さい。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行っております。

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

JAERI－Data/Code 2000－024

格子ガス気液モデルのシミュレーションコードの並列化

日本原子力研究所計算科学技術推進センター
川井 渉*・海老原 健一・久米 悦雄・渡辺 正

(2000年2月25日 受理)

MPI (Message Passing Interface) ライブラリを用いて，格子ガス気液モデルによる流体現象のシミュレーションコードの並列化を行った。並列化によって，より大規模なシミュレーションを行うことが可能となった。また，分散メモリ型ベクトル並列計算機 VPP500，分散メモリ型スカラ並列計算機 AP3000，ワークステーションクラスタにおいて実行した結果，実行時間がほぼプロセッサ台数に比例して減少した。

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
*富士通株式会社

JAERI－Data/Code 2000－024

Parallelization of Simulation Code for Liquid-gas Model of Lattice-gas Fluid

Wataru KAWAI*, Kenichi EBIHARA, Etsuo KUME and Tadashi WATANABE

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

(Received February 25, 2000)

A simulation code for hydrodynamical phenomena which is based on the liquid-gas model of lattice-gas fluid is parallelized by using MPI (Message Passing Interface) library. The parallelized code can be applied to the larger size of the simulations than the non-parallelized code. The calculation times of the parallelized code on VPP500(Vector-Parallel super computer with dispersed memory units), AP3000(Scalar-parallel server with dispersed memory units), and a workstation cluster decreased in inverse proportion to the number of processors.

Keywords: Liquid-gas Model of Lattice-gas Fluid, Parallelization, Vectorization, MPI, VPP500, AP3000, Workstation Cluster

*FUJITSU,Ltd

目　　次

Contents

1. はじめに

本レポートでは，高速化及び計算の大規模化を目的とした，格子ガス気液モデルに基づく流体現象のシミュレーションコードの並列化について記述する。

次章で述べるように，格子ガス気液モデルは，流体現象を粒子の集まりとして計算する格子ガスに，ある距離はなれた粒子同士の相互作用（長距離相互作用）を導入することによって，一成分系での相分離を伴う流体現象をシミュレーションするモデルである。このコードで，より大きな系におけるシミュレーションを高速に行うことは，他の流体現象を計算するコードと同様に，より詳細な現象を観察するために必要である。

しかし，現在の並列化されていないコードでは，富士通製分散メモリ型ベクトル並列計算機 VPP500/42（以下，VPP500）の場合，使用可能なメモリサイズが 200MB 弱であるため約 700x700 程度，分散メモリ型スカラ並列計算機 AP3000/24 （以下，AP3000）の場合，使用可能なメモリサイズは約 2000MB 弱であるため約 3000x3000 程度の格子サイズを確保できるだけである。よって，より大きなサイズのシミュレーションをするために並列化を行った。

このコードは，系のサイズに比例して使用メモリが増加するため，各プロセスが分割された領域について計算する方法が効率的である。よって，今回の並列化では，計算領域を各プロセスに分割する領域分割による並列化法を採用した。

尚，以下において並列化する前のコードをシングルコードと記述する。

2. コード概要

本コードは，２次元格子ガス気液モデルを用いて，相分離を伴う一成分流体のシミュレーションを行う [2]。

まず，格子ガス気液モデルについて記述する。格子ガスは，Fig.2.1 に示したような三角形格子で離散化された空間に，格子点間を結ぶリンク（Fig.2.2）を速度として持つ粒子を分布させ，個々の粒子の伝播，衝突の繰り返しによって，その粒子分布を離散時間で時間発展させ，流体現象をシミュレーションするモデルである。よって，格子ガスは，流体を粒子の集まりとしてシミュレーションするため，連続体の方程式を数値的に解く場合に較べ，二相流のような流体を容易に扱うことができる。

当初，格子ガスとして，正方格子を用いた HPP（Hardy, Pomeau, Pazzis）モデル [3] が使われていたが，このモデルは，連続極限において等方的な Navier-Stokes 方程式を得ることができず，また衝突規則に非物理的な保存則が含まれていた。これを解消するために，1986 年に Frisch, Hasslacher, Pomeau により，三角格子を用いた FHP モデルが提案された [4]。FHP モデルでは，三角格子にすることにより，Navier-Stokes 方程式を等方的にするために必要な自由度を得て，連続極限での Navier-Stokes 方程式の等方性を回復し，かつ粒子同士の衝突則を増やすことにより非物理的な保存則を減少させた。これにより，格子ガスモデルが，流体のシミュレーションに広く使われるようになった。FHP モデルは，その衝突則によって，以下の三種類に大きく分けることができる [5]。基本的な２体衝突，３体衝突のみを含んだ FHP1 モデル，FHP1 モデルに格子点上に停止している 速度 0 の粒子を含めた FHP2 モデル，FHP2 に衝突に直接的には関与しない粒子を含め，さらに各衝突則のデュアル（粒子の有無を反転させること）を取った衝突則を含めた FHP3 モデルである。

さらに，この格子ガスに，Fig.2.3 に示すようなある距離離れた格子点の粒子同士を引きつける相互作用（長距離相互作用）を付加することにより，一成分系での相分離を可能としたモデルが，格子ガス気液モデルである。このモデルは，1990 年に Appert と Zaleski によって提案されたモデルであり [6]，連続極限において Van der Waals の気液理論に類似した状態方程式を持つ。つまり，格子ガス気液モデルの状態方程式では，長距離相互作用の距離が Van der Waals の気液理論での温度のような働きをし，ある粒子密度の系では，この距離が変化することにより，不安定化し相分離が起こる。

次に，格子ガス気液モデルのシミュレーションコードの概要について記述する。上記のように格子ガスモデルは，流体をシミュレーションするモデルであるので，格子のサイズ，粒子分布密度をシミュレーションのパラメータとして変化させることができる。格子サイズは，二次元であることから x 方向，y 方向について与え，粒子密度は，格子点に配置される粒子の数に上限があるため，その上限に対する各格子点の粒子数の比（換算粒子密度）によって与える。よって，換算粒子密度は，0 ～ 1 の間の数値となる。また，シミュレーションに用いる衝突則として，３つの代表的な FHP モデル，FHP1，FHP2，FHP3 のいずれかを選択することも可能であ

る。さらに，長距離相互作用の距離もパラメータとして変更可能である。シミュレーションを実行する場合，基本的には周期的境界条件が選択されるが，いくつかの境界を壁として導入することにより，シミュレーション領域を制限することも可能である。そして，このような境界のいずれかを選択し，粒子に規則的に運動量を与えることにより，一成分系における相分離を伴った流体現象のシミュレーションを行うことができる。コードの処理は，Fig.2.4 に示すように，大きく 2 つのフェーズに分かれている。フェーズ 1 では，上記のパラメータにしたがって粒子を格子上に分布し，系全体で均一になるように粒子を拡散する。フェーズ 2 では，フェーズ 1 で拡散された粒子分布を用いシミュレーションを行う。

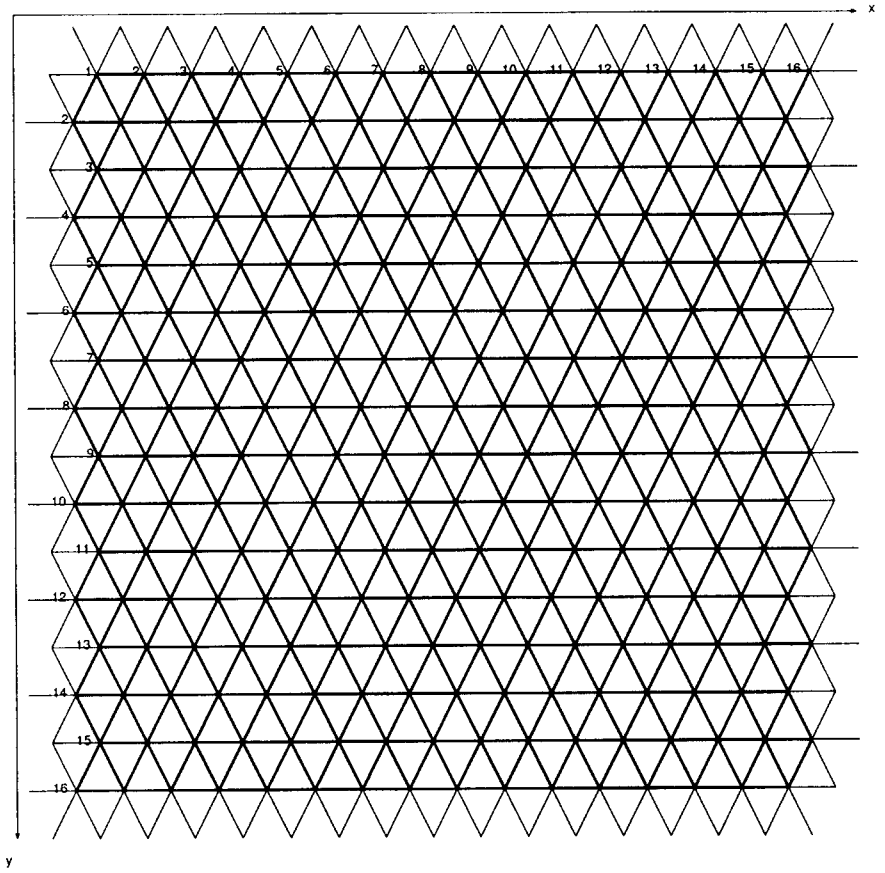

Fig. 2.1 Example of the 2-dimensional triangular lattice(16x16)

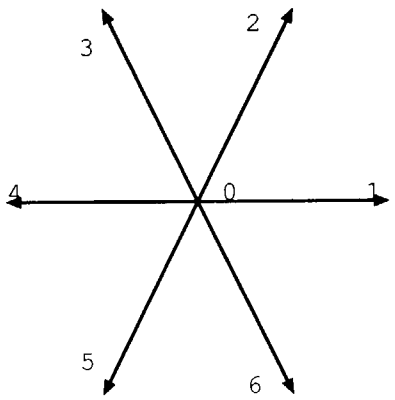

Fig. 2.2 Velocity directions

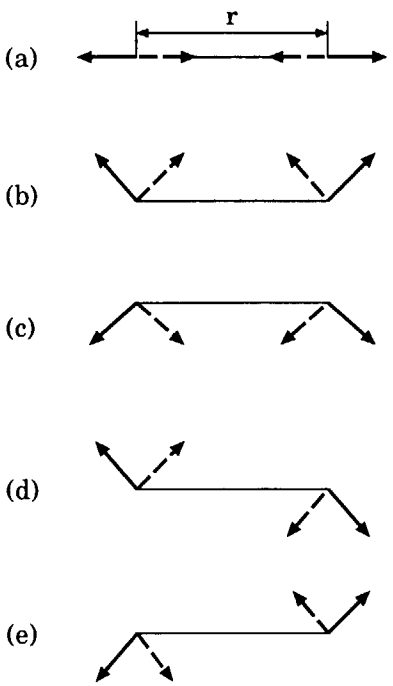

Fig. 2.3 Long-range interaction

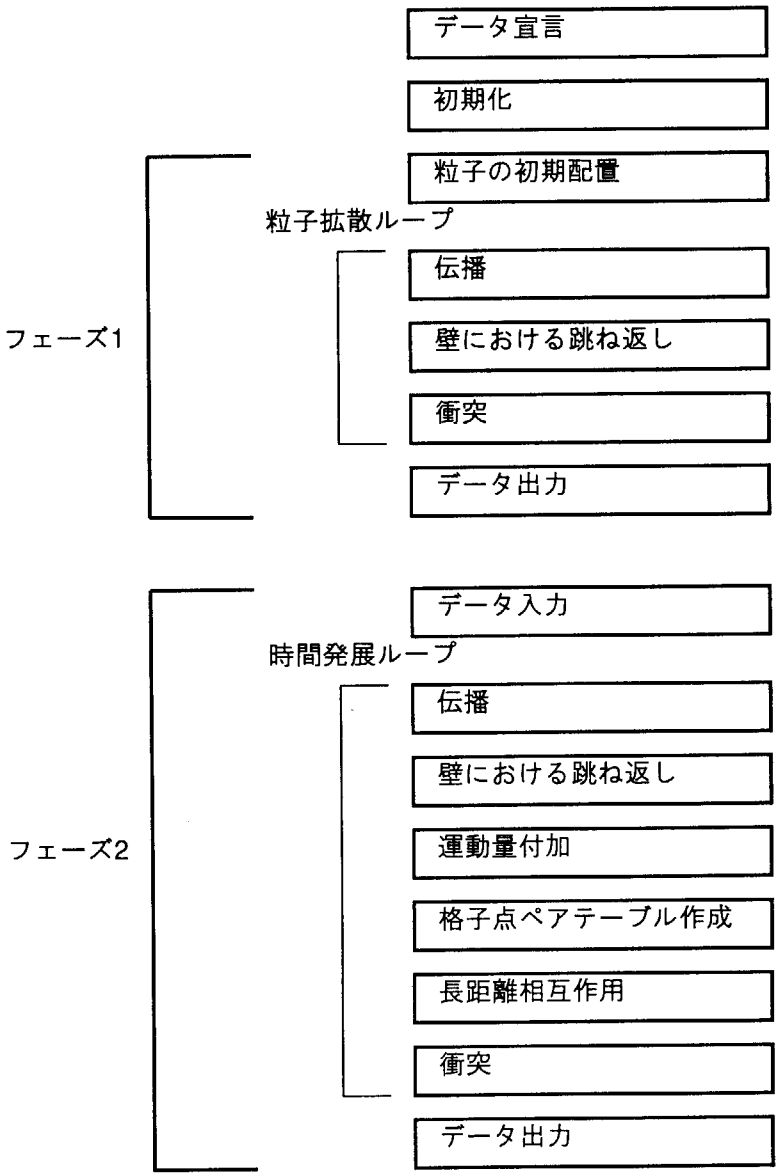

Fig. 2.4 Flowchart of the simulation code

3. コーディングスタイルの変更

並列化は，シングルコードを参照して行ったが，作業の効率化やメンテナンス性のためにいくつかの部分を変更した。変更点を以下に示す。

・ 配列 state の削除によるメモリサイズ節約

シングルコードでは，格子点毎の粒子の状態を保持する配列として，メインルーチンでは state，a_state，b_state が，サブルーチンでは before と after が用いられていた。

この内，配列 state は b_state や a_state のデータが，一時的に代入されるだけであったため削除した。これにより，配列 state で費やされていたメモリサイズを節約することができた。また，a_state と after，b_state と before はそれぞれ同一のメモリ領域で使用されていたため，a_state と after は配列 astate に，b_state と before は配列 bstate に統一した。

・ 初期化サブルーチン等の非採用

シングルコードでは，配列の初期化のみを行うサブルーチン initdat2 と配列から配列へのデータコピーのみを行うサブルーチン copydata が存在していた。これらの呼び出し処理を省くためにサブルーチン化をやめ，メインルーチンにインライン展開した。

・ 引数渡しのとり止めと COMMON 変数化

シングルコードでは，各サブルーチンに引数渡しの方法でデータを渡していた。引数渡しされているデータは，親ルーチンと子ルーチンの間でメモリ領域やデータの意味が同一であったが，サブルーチン内で名前が変更されていた。このようなコーディングスタイルにおいてデバッグ等を行う場合，親ルーチンと子ルーチンの間等で配列名や内容を対応させる作業が発生する。そこで，引数渡しを行った方が良い場合を除いて，これらを COMMON 変数化して名前を同一にし，引数渡しをとり止めることにした。この結果，各サブルーチンの名前が同一になり，コードが扱いやすくなった。

・ 粒子配置データの管理方法の変更とこれに伴うメモリサイズの節約

格子点毎の粒子の状態は，各方向（6 または 7 方向）に存在していれば 1，存在していなければ 0 のフラグを立てて管理される。よって，各方向で 1 か 0 の値しかとらない。この値をメモリサイズを節約できるように管理する方法として，ブール値を 10 進数として管理する方法に変更した。つまり，粒子配置を 6 ビットまたは 7 ビットのブール値の 2 進数と考え，これを 10 進数へ変換して管理する。これによって，系全体の粒子の状態を格納する配列のサイズは，全格子点数×（6 または 7）であったのが，全格子点数のみのサイズになった。壁配置データの管理方法もこれと同様に変更した。

4. 並列化方針

本コードにおいて宣言されている大部分の配列は，全格子点の数が基本サイズであり，各格子点の属性データとして使用される。また Fig.2.4 の各処理部の主なループは，全格子点数で回転するループで構成される。この内，伝播部は，各格子点とその隣の格子点のデータを必要とし，衝突部と運動量付加部，壁における跳ね返し部は各格子点のみのデータを必要とする。つまり，各格子点のみか，各格子点とその隣の格子点のデータを使用した処理がループ回転数分繰り返される構成になっている。また，長距離相互作用部と格子点ペアテーブル作成部は，長距離相互作用の距離分だけ離れた格子点を参照する必要がある。

並列化では 2 次元格子空間を Fig.4.1 のように分割する。Fig.4.1 では Fig.2.1 の格子空間を 4 つの領域に分割した例を示している。更に，隣の分割領域とのデータ授受を行うために，Fig.4.2のように，各分割領域に長距離相互作用分の領域を付加する（図では長距離相互作用の距離 r=3 の場合を示してある）。この領域は，隣同士の分割領域との共有領域である。各プロセッサの処理では，Fig.4.2 の全領域から共有領域を除いた領域に対するループの回転を行うことになる。以下，Fig.4.2 の分割した 2 次元格子空間を分割領域，分割領域の点線部を袖部，実線部を主領域部と呼ぶ。また図中において rank とは，n 個のプロセスにより並列実行する場合，各プロセスを識別するために割り当てられる番号であり，0 ～ (n-1) の番号が付けられる。

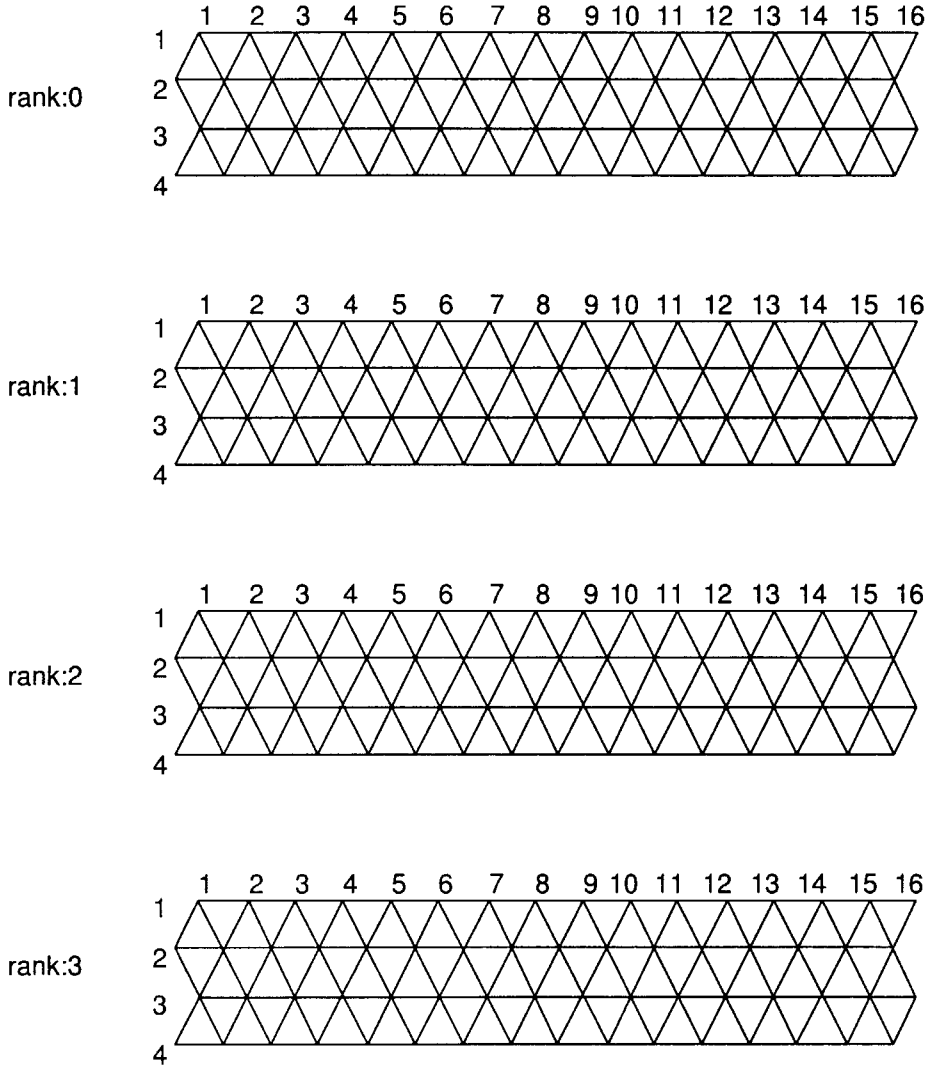

Fig. 4.1 Division of the 2-dimensional triangular lattice

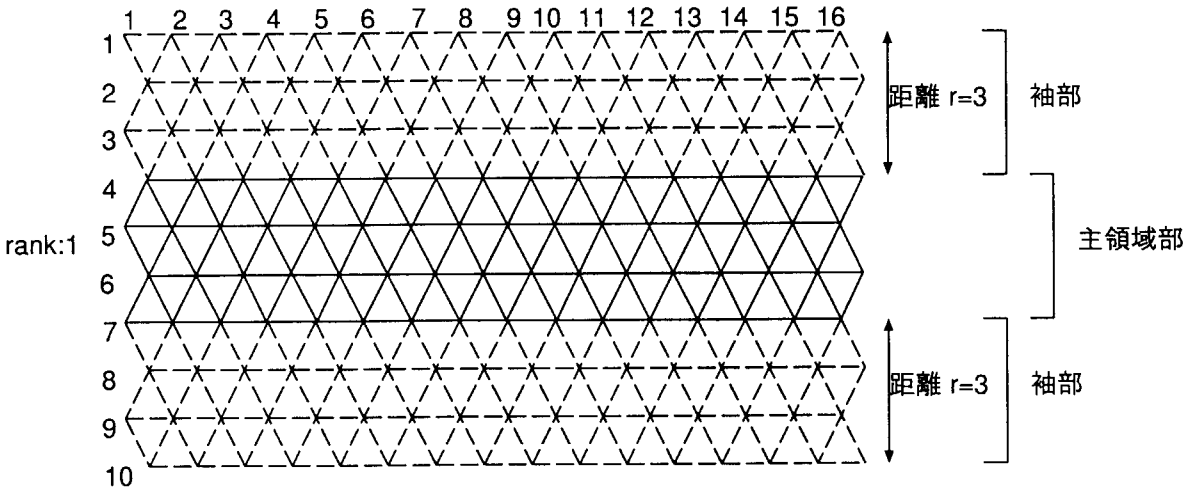

Fig. 4.2 Divided region with overlap areas

5. 並列化

ここでは，Fig.2.4 の処理フローの順に，各ルーチンの並列化について記述する。並列化の際に使用した並列化命令は，MPI（Message Passing Interface）ライブラリである [7] [8]。MPI ライブラリは，現在各ベンダーの計算機上で用意されているライブラリであるため，これを用いて並列化することにより，将来の移植作業における負荷が軽減される（今回の作業で使用した MPI ライブラリの関数を付録 1 に示す）。

5.1 データ宣言部

データ宣言部の主な変更点は，データの COMMON 変数化，並列実行時パラメータの追加，データの分割である。シングルコードのデータ宣言部を Fig.5.1 に，並列化コードのデータ宣言部を Fig.5.2 に示す。

・シングルコード

Fig.5.1 において，2 次元格子空間の x 方向のサイズ xs，y 方向のサイズ ys，各格子点が持つ方向の数 0 ～ nsite-1 によって宣言される。FHP1 モデルのシミュレーションでは 10 行目において nsite=6 と指定され，FHP2 と FHP3 モデルのシミュレーションでは nsite=7 と指定される。他の大部分は作業配列である。

・並列化コード

シングルコードでは，引数渡しの方法で各サブルーチンへデータを渡していたが，COMMON 変数化を行い，各サブルーチンにおいて，同じ名前の配列名や変数名を用いることにした。そして，並列化コードのデータ宣言部をインクルードファイルに格納し，これを各サブルーチンでインクルードして利用するようにした。このインクルードファイル INC_DATA を Fig.5.3 に示す。

また各 FHP モデルにおける方向の扱い方を変更することにした。FHP1 モデルのシミュレーションでは，10 行目において nsite=6，istart=1 と指定し 1 ～ 6 の範囲を用い，FHP2 と FHP3 モデルのシミュレーションでは，nsite=6，istart=0 と指定し 0 ～ 6 の範囲を用いることにした。この結果，Fig.2.2 に示す方向と同様の考え方でコードを利用できる。

19 ～ 34 行目が並列実行のためのパラメータである。21 ～ 22 行目は，1 プロセスのみの実行時に有効化する（iwidth=0）。25 行目と 28 行目は，長距離相互作用の距離 irr=0，かつ複数プロセス実行時に有効化する（iwidth=1）。27 ～ 28 行目は，irr ≠ 0，かつ複数プロセス実行時に有効化する（iwidth=irr）。

32 行目では元の 2 次元格子空間の y 方向のサイズ（iyss）をプロセス数（ipe）で割り，分割領域の主領域部の y 方向のサイズ（iys）を設定する。33 行目の numsite_para は分割領域すべての格子点数であり，34 行目の numsite は分割領域の主領域部の格子点数である。

36 ～ 48 行目までがコードで用いる主な配列の宣言であり，分割領域の格子点数 numsite_para を基本サイズにして宣言し，50 ～ 51 行目においてこれらを COMMON 変数化した。53 ～ 54 行目は乱数発生のための種 iseed と乱数を格納する配列 rand の宣言であり，56 行目でデータ転送時に用いるバッファ配列 itrans_array と共に COMMON 変数化した。58 ～ 62 行目，64 ～ 66 行目，68 ～ 70 行目 も各配列の宣言方法は同様である。72 行目と 74 行目で宣言する変数は各サブルーチンでローカルに使用する変数であるため，COMMON 変数化は行っていない。

MPI ライブラリを用いて並列化する場合，プロセス数や各プロセスを識別するランクを多用する。これらも各ルーチンで使用するためにはデータ宣言が必要であるため，MPI ライブラリ用のインクルードファイルを作成した。このファイル INC_MPI を Fig.5.4 に示す。NPROCS は実行時に決まるプロセス数，MYRANK はランク ID，ipe はコンパイル前に指定するプロセス数であり，インクルードファイル mpif.h は MPI ライブラリを使用するルーチンでは必ず必要である。4 行目の配列 status は MPI ライブラリを使用する際に多頻度に使用される変数である。

5.2 初期化部

5.2.1 メインルーチンについて

・シングルコード

Fig.5.5 において，2 行目の br は，サブルーチン lrange （格子点ペアテーブル作成部） を最初に 1 回だけ呼ぶためのフラグである。4 ～ 13 行目ではシミュレーションパラメータの出力を行う。17 行目で呼んでいるサブルーチン mkclt3 では，格子点毎に各方向に位置する格子点 ID を求め，配列 site に格納する。20 ～ 33 行目は，シミュレーションパラメータで指定された optw によって 2 次元格子空間の境界に壁を設定する。optw=0 では壁を設定しない。32 ～ 36 行目は start という変数に 1 か 0 を代入する。FHP1 モデルでは停止粒子（速度 0）は考慮しないため，start に 1 を代入する。一方，FHP2 モデルか FHP3 モデルでは停止粒子も考慮するため，start に 0 を代入する。39 ～ 42 行目は，配列 state，acumtr，acumlg，acumbo の 0 への初期化である。

・並列化コード

Fig.5.6 において，1 ～ 3 行目は MPI ライブラリの初期化部である。5 行目はシングルコードと同様にサブルーチン lrange を最初の 1 回だけ呼ぶためのフラグである。7 ～ 18 行目では，これもシングルコードと同様にシミュレーションパラメータの出力を行う。並列化コードの場合はランク 0 のみで出力させた。24 ～ 25 行目では，サブルーチン mkhtable を呼び出し，ブール値を変換した 10 進数と方向を与えた場合にその方向のビット情報を取得できるテーブル tab と tabw を作成する。テーブル tab は各格子点の粒子の状態を保持する配列 astate と bstate 用で，テーブル tabw は各格子点の壁情報を保持する配列 iwall 用である。

27 ～ 29 行目では 3 つの配列の 0 番目に 0 を代入して初期化する。分割領域中に添字 0 の格

子点は存在していないが，サブルーチン lgcol 内において，便宜上添字 0 の配列を必要とするためにこの処理を行う。

36 ～ 56 行目で呼ばれるサブルーチン mkclt3_1， mkclt3_2 は格子点毎に各方向に位置する格子点 ID を求め，配列 isite に格納する。 63 ～ 89 行目で呼ばれるサブルーチン mkwalla1, mkwalla2， mkwallb， mkwallc1， mkwallc2 は領域の境界部に壁を設定する。 91 ～ 97 行目は，シングルコードと同様に配列 acumtr， acumlg， acumbo を 0 で初期化する。

5.2.2 サブルーチンについて

ここでは，初期化部で使用するサブルーチンや関数について述べる。今回追加したサブルーチンは mkhtable である。また，シングルコードのサブルーチン mkclt3 を 並列化コードでは mkclt3_1 と mkclt3_2 に， mkwalla を mkwalla1 と mkwalla2 に， mkwallc を mkwallc1 と mkwallc2 にそれぞれ変更した。

(1) サブルーチン mkhtable

本ルーチンでは，ある格子点上の粒子配置または壁配置を 10 進数で表した値とその格子点のある方向を与えた時，その格子点の同じ方向における粒子の有無または壁の有無を返すテーブルを作成する。作成するテーブルは 2 つあり， tab と tabw である。 tab は格子点毎の粒子の状態が 10 進数として格納されている配列 astate と bstate 用であり， tabw は 壁情報（ -1 または 0 ）が 10 進数として格納されている配列 iwall 用である。つまり，それぞれのテーブルの使用方法は，

例： tab(bstate(a),3) → 格子点 a の方向 3 に粒子が存在する (1) か否か (0) を返す。

例： tabw(iwall(b),2) → 格子点 b の方向 2 に壁が存在する (-1) か否か (0) を返す。

となる。

(2) サブルーチン mkclt3

本ルーチンでは， 2 次元格子空間上の左上の格子点から右下の格子点へ向けて 1,2,3,4,.... のように連番で ID が付加されているとし，配列の引数に格子点 ID と粒子の方向を与えると，その方向の格子点 ID が参照できるように配列を定義する。シングルコードの mkclt3 を Fig.5.7 に示す。

・シングルコード - サブルーチン mkclt3

Fig.5.7 において， 10 ～ 11 行目は 2 次元格子空間の x 方向と y 方向のサイズのループであり，この 2 重ループで全格子点数のループになる。 16 行目は，各格子点の方向数のループであり，ループ内では方向毎に if 文で分岐している。 12 行目の s は，各格子点に付けられている ID である。 13 行目の itmp は格子点 ID を x 方向のサイズで割った余りであり，これによって格子点 s が x 方向の何処にあるかがわかる。 14 行目の jtmp は，格子点の y 方向の値が偶数か奇数かを判別できる値である。偶数か奇数かによって，横に凹 (Fig.5.8) の線上に位置する格子点か，または横に凸 (Fig.5.9) の線上に位置する格子点かを見極める。これは，凹か凸によって各方向（隣の格子点）の格子点 ID を求める方法が異なるためである。これらの s,

itmp, jtmp, x 方向のサイズ（xx），y 方向のサイズ（yy）を用いて，配列 tsite に各格子点 ID の各方向の格子点 ID を定義する。

各格子点において，方向 0 の格子点はその格子点自身であるため，配列 tsite には s を代入する。方向 1 の格子点はその格子点の右隣の格子点であるため s+1 を代入するが，2 次元格子空間の右境界の格子点の方向 1 の格子点は 2 次元格子空間が周期的であることから，s-xx+1（反対側の格子点）を代入する。方向 4 の格子点は方向 1 の格子点を考える場合の逆になる。

・並列化コード - サブルーチン mkclt3_1，mkclt3_2

シングルコードにおいては，2 次元格子空間を分割する必要がないため，ID の付け方は一通りあれば良く，Fig.2.1のように横に凹から始まる 2 次元領域として固定できた。しかし，領域を分割する場合は，分割数や袖部のサイズによって，以下のことを考慮する必要がある。

a. 分割領域の主領域部について

並列化コードでは，すべての分割領域の主領域部を結合した時，元の 2 次元格子空間となる必要がある。よって，複数プロセス実行時のランク 0 に担当させる分割領域の主領域部は，横に凹の場合から始まる領域にしなければならない。

b. 分割領域の袖部について

各分割領域に袖部が存在する場合は，袖部の y 方向のサイズ（長距離相互作用の距離）によって，分割領域の始まりが横に凹の場合も横に凸の場合もある。つまり，主領域部が横に凹で始まり，袖部の y 方向のサイズが偶数の場合は分割領域が横に凹から始まり，奇数の場合は分割領域が横に凸から始まる。よって，並列化コードでは，横に凸から始まる格子点 ID の定義方法が必要になる。

c. 分割領域の主領域部の y 方向のサイズが奇数の場合について

分割領域の主領域部の y 方向のサイズが奇数になる場合は，主領域部の形状が各プロセスによって異なる。奇数プロセス毎にまたは偶数プロセス毎に同一の形状になる。

d. 配列に格納する格子点 ID について

シングルコードにおいては，元の 2 次元格子空間の左上から右下へ向けて連番で ID が付加されている。並列化コードでこれと同様に連番で ID を付けると，例えばランク 1 ではランク 0 での分割領域の最終格子点 ID を知る必要があるため，プロセス間に依存関係が生じてしまう。しかし，本コードでは，必ずしも格子点 ID をすべて連番で持たせ，これらの順番を保持する必要はない。そこで，分割領域に対してはプロセス毎独立に，ID を付加することにした。

以上の 4 点を考慮し，サブルーチン mkclt3 の代わりに，並列化コードでは，サブルーチン mkclt3_1，mkclt3_2 の 2 つを作成し，メインルーチンにおいてこれらを呼び出す部分を Fig.5.6 の 36 ～ 56 行目のようにした。並列化コードの mkclt3_1 を Fig.5.10 に，mkclt3_2 を Fig.5.11 に示す。

mkclt3_1 は，シングルコードの mkclt3 と ID の付加方法は同じである。ただし，分割領域が対象となるため，y 方向のサイズを Fig.5.10の 12 行目において，分割領域の y 方向のサ

イズに変更した。また，並列実行時の分割領域には必ず袖部が存在するため，元の 2 次元格子空間のような y 方向についての周期的な格子点の参照は必要ない。周期的な格子点参照は，袖部のデータ転送によってなされる。よって，本ルーチンの終りの部分（116 ～ 125 行目）において，分割領域の上境界部の格子点については方向 2 と 3 に，下境界部の格子点については方向 5 と 6 に，格子点 ID ではなくフラグ -1 を代入した。

mkclt3_2 は，分割領域の始まりが横に凸の場合に使用する。

(3) サブルーチン mkwalla

本ルーチンは，2 次元格子空間の全境界に壁を設定する（正方形型 Fig.5.12）。これと同様の壁を分割領域で実現するには，Fig.5.13 のようにすればよい。これは，最初のランク（ランク 0）が担当する主領域部では，上境界部，左境界部，右境界部に壁を設定し，最後のランク（ランク max）では，下境界部，左境界部，右境界部に，その他のランクでは，左境界部と右境界部に設定することになる。並列化コードでは正方形型の壁を設定するサブルーチンとして，mkwalla1 と mkwalla2 を作成した。mkwalla1 は分割領域の主領域部の y 方向のサイズが偶数で横に凹から始まる場合であり，mkwalla2 は y 方向のサイズが奇数で横に凸から始まる場合かつ奇数プロセスの場合である。これらの 2 つのサブルーチンの区別を Fig.5.6 の 63 ～ 89 行目のようにメインルーチンの呼び出し部分で行った。これにおいて，64 ～ 72 行目が正方形型の場合に実行される部分である。シングルコードの mkwalla を Fig.5.14 に，並列化コードの mkwalla1 を Fig.5.15 に，mkwalla2 を Fig.5.16 に示す。

・シングルコード - サブルーチン mkwalla

本ルーチンにおいて，壁情報を設定する（格子点毎に壁を設定する方向に -1 を代入する）場合も，対象とする格子点 ID を識別する考え方は mkclt3 と同様である。正方形型の壁を作る場合は 2 次元格子空間の境界格子点のみを対象にすればよい。Fig.5.14 において，10 行目は格子点数のループであるが，ループ中では if 文により境界格子点のみを対象にしている。上境界部の格子点では方向 2 と 3 に -1 を設定し，下境界部の格子点では方向 5 と 6 に -1 を設定し，右境界部の格子点では方向 1 または方向 1 と 2 と 6 に，左境界部では方向 4 または方向 4 と 3 と 5 に -1 を設定している。

・並列化コード - サブルーチン mkwalla1，mkwalla2

並列化コードにおける分割領域に正方形型の壁を設定する場合は，既述のように，ランク 0 とランク max とその他のランクで設定する境界が異なる。よって，Fig.5.15 の 16 行目，60 行目，104 行目で if 文によりランクの場合分けをした。ランク 0 の場合においては，分割領域の格子点数のループではなく，上境界の格子点数のループと右境界部の格子点数のループ，左境界部の格子点数のループを作成した。ランク max の場合においては，右境界部の格子点数のループと左境界部の格子点数のループ，下境界部の格子点数のループを作成した。その他のランクの場合においては，右境界部の格子点数のループと左境界部の格子点数のループのみを作成した。mkwalla1 は，分割領域の主領域部の y 方向のサイズが偶数で横に凹から始まる場合に使用する。

mkwalla2 は，主領域部の y 方向のサイズが奇数でかつ奇数プロセスの場合に使用し，横に

凸から始まる場合の壁を設定する。横に凹から始まる場合（mkwalla1）と異なるのは，分割領域の右境界部と左境界部において横に凸部と凹部が反対になることである。また，mkwalla2 は奇数プロセスのみで実行されるため，ランク 0 用の処理を省いている。

(4) サブルーチン mkwallb

本ルーチンは，2 次元格子空間の上下境界に壁を設定する（上下型 Fig.5.17）。これと同様の壁を分割領域で実現するには，Fig.5.18 のようにすればよい。これは，ランク 0 が担当する主領域部の上境界部と，ランク max が担当する主領域部の下境界部への設定になる。上下型の壁を設定する上で，凹で始まるか凸で始まるかは関係ない。ここで使用するサブルーチンは mkwallb のみである。シングルコードの mkwallb を Fig.5.19 に，並列化コードの mkwallb を Fig.5.20に示す。

・シングルコード - サブルーチン mkwallb

本ルーチンは，全格子点数のループの中で，上境界部と下境界部を if 文で判定し，上境界部では方向 2 と 3 に，下境界部では方向 5 と 6 に -1 を設定している。

・並列化コード - サブルーチン mkwallb

並列化コードにおける分割領域に上下型の壁を設定する場合は，既述のように，ランク 0 とランク max だけで設定すればよい。よって，Fig.5.20 の 16 行目と 27 行目において if 文によりランクの場合分けをした。ランク 0 の場合においては，上境界部の格子点のループを作成し，ランク max の場合においては，下境界部の格子点のループを作成して壁を設定するようにした。

(5) サブルーチン mkwallc

本ルーチンは，2 次元格子空間の左右の境界に壁を設定する（左右型 Fig.5.21）。これと同様の壁を分割領域で実現するためには，Fig.5.22 のようにすればよい。つまり，すべてのランクで主領域部の左境界部と右境界部に壁を設定すればよい。そこで，並列化コードでは左右型の壁を設定するサブルーチンとして，mkwallc1 と mkwallc2 を作成した。シングルコードの mkwallc を Fig.5.23 に，並列化コードの mkwallc1 を Fig.5.24 に，mkwallc2 を Fig.5.25 に示す。

・シングルコード - サブルーチン mkwallc

本ルーチンも全格子点数のループの中で，右境界部と左境界部を if 文で判定し，右境界部が横に凹の境界の場合は方向 1 のみに，横に凸の境界の場合は方向 1 と 2 と 6 に -1 を設定している。また，左境界部が横に凹の境界の場合は方向 4 のみに，横に凸の境界の場合は方向 4 と 3 と 5 に -1 を設定している。

・並列化コード - サブルーチン mkwallc1，mkwallc2

mkwallc1 は，分割領域の主領域部の y 方向のサイズが偶数で横に凹から始まる場合に使用する。Fig.5.24 においては，右境界部の格子点のみを対象とするループと左境界部の格子点のみを対象とするループを作成した。それぞれの壁の設定の方法はシングルコードと同様である。

mkwallc2 は，分割領域の主領域部の y 方向のサイズが奇数で横に凸から始まる場合において使用する。mkwallc1 と異なるのは，右境界部の格子点において方向 1 にのみ -1 を設定していたのが，方向 1 と 2 と 6 に必要になり，方向 1 と 2 と 6 に設定していたのが，方向 1 のみに必要になる。また左境界部の格子点において方向 4 にのみ -1 を設定していたのが，方向 4 と 3 と 5 に必要になり，方向 4 と 3 と 5 に設定していたのが，方向 4 のみに必要になる。

(6) ユーザ関数 irehash

本ルーチンは，並列化にともなって新たに加えられたサブルーチンであり，壁設定を行う各ルーチンにおいて，格子点毎の壁配置を一次的に格納する配列 la を 10 進数に変更するルーチンである。並列化コードの各壁設定ルーチンの最終部分で使用される。本ルーチンを Fig.5.26 に示す。

本ルーチンは，ビット情報を仮引数で配列で受ける。FHP1 モデルのシミュレーションで格子点自身（方向 0）は考慮しないため，FHP2 モデルと FHP3 モデルのシミュレーションと区別することにした。Fig.5.26 において 6 行目が FHP2 と FHP3 モデルのシミュレーション向けであり，8 行目が FHP1 モデルのシミュレーション向けである。FHP1 モデルでは最大 6 ビットまで考慮すればよい。

5.3 粒子の初期配置部 (フェーズ 1)

この部分は，各格子点に対して粒子を初期配置し，フェーズ 1 のみで行われる。

5.3.1 メインルーチンについて

・シングルコード

Fig.5.27 において，3 行目でサブルーチン sist を呼び粒子の初期配置を行う。4 ～ 6 行目は sist 中のエラー処理でエラーを発見した場合の出力である。9 ～ 11 行目は x 方向と y 方向のモーメントと方向毎の粒子総数，格子点上の粒子の総数を求めて出力する。これにより，元の 2 次元格子空間上における初期配置状態を知ることができる。14 ～ 15 行目の n は出力ファイルに結合させる機番である。

・並列化コード

Fig.5.28 において，3 行目でいくつかのサブルーチンで使われる乱数発生の種の初期化を行う。シングルコードでは，富士通科学計算サブルーチンライブラリ ssl2 で用意されている ranu2 を用いて乱数を発生させていた。この乱数発生ルーチン ranu2 は，初期種を与えるとそれに対する乱数を発生し，同時に次の乱数を発生する時に用いる種も求める。つまり，コード中において 1 つ初期種を決めると，その後に発生する乱数列と種の列が決まる。並列実行時にはプロセス間で同一の乱数を発生させない方が適切であるので，ここでプロセス毎に異なる初期種を設定することにした。

10 ～ 13 行目ではシングルコードと同様に，粒子の初期配置を行う sist を呼び出し，エラー

があった場合エラー出力も行うようにした。並列化コードでは引数渡しでサブルーチンを呼ばないことにした。粒子の初期配置結果は，配列 bstate に格納する。

15 ～ 20 行目では x 方向と y 方向のモーメントと方向毎の粒子総数，格子点上の粒子の総数を求めて出力する。これにより，各プロセスにおける分割領域上の初期配置状態を知ることができる。並列化コードでは countPQ の呼び出しのみでは各値がプロセス毎に求まってしまうため，これらの値をプロセス間で総和する必要がある。この総和を行うサブルーチンを countPQ_sum として作成した。

22 行目の n は，出力ファイルに結合させる機番である。10+MYRANK より各ランクの出力ファイルに結合させる機番をプロセス毎に設定する。

5.3.2 サブルーチンについて

(1) サブルーチン sist

本ルーチンでは，2 次元格子空間上に粒子を配置する。配置する粒子数は，（格子点数×格子点毎の方向× den）である。den は，シミュレーションパラメータで指定される粒子数密度である。シングルコードでは，シミュレーションする格子空間すべてを対象にして配置していたのに対し，並列化コードでは，各プロセスの分割領域の主領域部を対象として配置することになる。

シングルコードでは，2 次元格子空間の x 方向のサイズと y 方向のサイズをそれぞれシミュレーションパラメータで指定される opt の値で割り，格子上に箱を設ける。そして，箱内の 1 つの格子点の全方向に粒子を配置したら次の箱に移動し，また 1 つの格子点の全方向に粒子を配置する。そしてこれを繰り返す。この際，配置した粒子数をカウントし，これが（格子点数×格子点毎の方向× den）になったら，繰り返しを終了する。

分割領域の主領域部では y 方向のサイズがシングルコードと異なるため，opt が同値だと 1 つの主領域部にシングルコードと同じ数の箱が設けられてしまう。そこで，主領域部に設けられる箱の数の全プロセスの総数が，シングルコードでの箱の数と同一になるように，主領域部に対する opt を設定するサブルーチン site_particle_check を新たに作成した。

並列化コードにおいて，このサブルーチンを利用して粒子を配置しても，厳密には，配置される格子点がシングルコードと並列化コードで異なることになるが，フェーズ 1 のシミュレーションの目的は，フェーズ 2 でのシミュレーションに対する初期状態を準備することであり，また，2 次元格子空間上の粒子を均一に拡散させることが目的であるため，このサブルーチンで配置される格子点の位置の違いはシミュレーション上特に問題にはならない。シングルコードの sist を Fig.5.29 に，並列化コードの sist を Fig.5.30 に示す。

・シングルコード - サブルーチン sist

26 行目で出力する値の一つが配置する粒子数である。32 ～ 61 行目が粒子配置の繰り返し処理であり，37 ～ 38 行目のループが箱の数のループである。45 ～ 50 行目において格子点に粒子を配置（配列 state に 1 を代入）し，配置粒子数（ii）をカウントする。67 ～ 77 はエラー処理部である。

・並列化コード - サブルーチン sist

23 行目において，主領域部に配置する粒子数を求める。27 行目で，主領域部向けの opt を求めるため，サブルーチン site_particle_check を呼び出す。33 ～ 75 行目が粒子配置の繰り返し処理であり，38 ～ 39 行目のループが箱の数のループである。55 ～ 62 行目において格子点に粒子を配置し，配置粒子数をカウントする。シングルコードでは，壁が存在しない格子点にのみ粒子を設定してカウントしていた。しかし，（格子点×格子点毎の方向× den）で決められる粒子配置数には壁が考慮されていないため，壁が存在しない場合に設定可能な粒子数より粒子配置数が大きくなる場合がある。このような場合，シングルコードではエラー出力を行っていたが，並列化コードでは壁の存在する格子点についてもカウントのみを行うようにし，エラー出力を行わないように変更した。この部分が，59 ～ 61 行目に相当する。

(2) サブルーチン site_particle_check

本ルーチンを Fig.5.31 に示す。10 行目の x 方向のサイズ ix を iopt で割った ix_part は，x 方向の箱の数である。x 方向は並列化における分割軸ではないため，シングルコードと同様である。14 行目において iopt をプロセス数で割り主領域部毎の y 方向の opt 値 iopt_part を求める。これで iy_part2 （=iys） を割り主領域部における y 方向の箱の数 iy_part を求める。また，24 行目では x 方向の opt 値を isetx に，25 行目では y 方向の opt 値を isety に求める。これらの値を親ルーチンである sist に返す。

(3) サブルーチン countPQ

本ルーチンは，格子上の x 方向と y 方向のモーメントと方向毎の粒子総数，格子上の粒子総数を求める。シングルコードの countPQ を Fig.5.32 に，並列化コードの countPQ を Fig.5.33 に示す。

・シングルコード - サブルーチン countPQ

21 ～ 26 行目において方向毎に粒子数をカウントする。31 ～ 37 行目において x 方向と y 方向のモーメント（それぞれ momx， momy）と格子上の粒子総数（nump）を求める。

・並列化コード - サブルーチン countPQ

12 ～ 18 行目で方向毎に粒子数をカウントする。ループ範囲は分割領域の主領域部である。23 ～ 40 行目において x 方向と y 方向のモーメントと格子上の粒子総数を求める。並列化コードでは，格子上の粒子総数 numpar を求める場合， FHP1 モデルのシミュレーションの場合と，FHP2， FHP3 モデルのシミュレーションの場合に，30 行目の if 文によって分けた。

(4) サブルーチン countPQ_sum

本ルーチンでは，並列化コード countPQ によって求められた，モーメントと各粒子総数について，プロセス間の総和を行う。本ルーチンを Fig.5.34 に示す。

15 行目では，分割領域上の粒子総数 numpar についてプロセス間の総和を行う。総和後の値は， snumpar に求められる。これが元の 2 次元領域上の総数に相当する。18 行目では， x 方

向のモーメント momx についてプロセス間の総和を行う。総和後の値は， smomx に代入される。21 行目では， y 方向のモーメント momy についてプロセス間の総和を行う。総和後の値は，smomy に代入される。24 行目では，配列 p_num についてプロセス間の総和を行う。総和後の値は，配列 sp_num に求められる。

以上，すべての総和された値は，標準出力へ出力するのみであるため， MPI ライブラリ mpi_reduce を使用して，ランク 0 のみに値が求まるようにした。

5.4 伝播部 (フェーズ 1)

この部分は， 2 次元格子上に存在する粒子の伝播を行う。粒子の伝播は Fig.5.35 のように，ある格子点のある方向に粒子が存在していたら同方向の隣の格子点の同じ方向に粒子を伝播する。フェーズ 2 の伝播部もこの部分と同様である。

5.4.1 メインルーチンについて

・シングルコード

Fig.5.36 において 2 行目でサブルーチン trans を呼び，粒子の伝播を行う。伝播前の粒子の状態は配列 state に，伝播後の粒子の状態は配列 a_state に格納される。 4 ～ 8 行目はデバッグモードでの実行時に trans の結果をチェックするために x 方向と y 方向のモーメントと方向毎の粒子総数，格子上の粒子総数を求めて出力する。

・並列化コード

Fig.5.37 において 4 行目はサブルーチン mpi_trans により配列 bstate の内容を各分割領域の袖部へデータ転送する。 11 ～ 14 行目では，粒子の伝播を行う trans を呼び出し，エラーがあった場合にエラー出力も行うようにした。並列化コードでは引数渡しでサブルーチンを呼ばないようにした。伝播前の粒子の状態は配列 bstate に，伝播後の粒子の状態は配列 astate に格納する。 16 ～ 23 行目はデバッグモードで実行される部分であり， x 方向と y 方向のモーメントと方向毎の粒子総数，格子上の粒子総数を求めて出力する部分である。

5.4.2 サブルーチンについて

(1) サブルーチン mpi_trans

本ルーチンでは分割領域の袖部へのデータ転送を行う。使用した MPI ライブラリは mpi_sendrecv である。これを使用すると，一回の命令で，片方の袖部でのデータの送受信ができる。よって，両袖部への転送を行うにはこれを二回行えばよい。本サブルーチンを Fig.5.38 に示す。

これにおいて，仮引数で受ける配列 transfer は転送対象配列であり，変数 ioverlap は配列 transfer の転送対象要素数である。 ioverlap は，分割領域の袖部の要素数であり両袖と

もに同じ数である。

10 ～ 17 行目では，各プロセスにおいて送信先プロセスのランクを設定する。10 行目では各プロセスより 1 つ下のランクを，11 行目では 1 つ上のランクを設定する。12 ～ 14 行目では，1 つ下のランクが 0 未満になった場合にランク max を代入する。これは 2 次元格子が周期的であるためである。また同じ理由から，15 ～ 17 行目でも 1 つ上のランクが max を越えた場合にランク 0 を代入する。20 ～ 27 行目では，各プロセスにおいて送信元プロセスのランクを設定する。20 行目では各プロセスより 1 つ下のランクを，21 行目では 1 つ上のランクを設定する。22 ～ 27 行目では，12 ～ 17 行目と同じ理由で，ランクが 0 未満になった場合はランク max を，ランクが max を越えた場合にはランク 0 を代入する。

30 ～ 54 行目が分割領域の下の袖部とのデータ送受信を，57 ～ 81 行目が上の袖部とのデータの送受信を行う。mpi_sendrecv を用いる場合，先に設定した送信先のランクと送信元のランクの他に，送受信バッファと送受信要素数が必要になる。送受信バッファは，配列 itrans_array(転送対象要素数,2) を用いることにした。送信バッファはこの配列の 2 次元目の値を 1，受信バッファは配列の 2 次元目の値を 2 にして使用した。また送受信要素数は，送信バッファの準備の段階である transfer から itrans_array への代入処理において実際の要素数をカウントした値 num_data を用いた。42 ～ 46 行目により，アドレス itrans_array(1,1) から num_data 個の要素をランク isup へ送信し，アドレス itrans_array(1,2) へ num_data 個の要素をランク irdown から受信する。また，70 ～ 75 行目により，itrans_array(1,1) から num_data 個の要素をランク isdown へ送信し，num_data 個の要素を itrans_array(1,2) へランク irup から受信する。50 ～ 54 行目，77 ～ 81 行目は，受信した要素を元の配列 transfer へ代入する。

(2) サブルーチン trans

本ルーチンでは，Fig.5.35 のように粒子の伝播を行う。

並列化の方針で各プロセスにおける計算担当部は分割領域の主領域部であり，各サブルーチンでのループ回転範囲も主領域部の格子点数にすることにした。しかし，分割領域の主領域部と袖部の境界部の格子点において，主領域部へ伝播させるべき粒子が袖部に存在している場合，ある格子点からその格子点の隣の格子点へ粒子を伝播する方法では，ループ回転範囲に袖部も含む必要がある。そこで，並列化コードでは袖部をループ範囲としないように，ある格子点へその格子点の隣の格子点から粒子を伝播する方法へ変更した。シングルコードの本ルーチンを Fig.5.39 に，並列化コードの本ルーチンを Fig.5.40 に示す。

・シングルコード - サブルーチン trans

Fig.5.39 において，17 ～ 24 行目が粒子の伝播を行う部分である。ある格子点のある方向に粒子が存在している場合，配列 tsite からその方向の格子点 ID を取得し，粒子を取得した ID を持つ格子点の同じ方向に伝播させる。これは，ある格子点から隣の格子点へ粒子を伝播する方法である。26 ～ 29 行目は，伝播に関係の無い格子点自身（方向 0）の状態のコピーである。31 ～ 38 行目は，エラー処理である。

・並列化コード - サブルーチン trans

Fig.5.40 において，15 ～ 25 行目が粒子の伝播を行う部分である。ここにおいて do 変数 i はある格子点での方向，io は i の逆方向である。まず，ID k1 を持つ格子点の io 方向にある隣の格子点の i 方向に粒子が存在するか否かを判定し，存在していれば ID が k1 の格子点の i 方向に粒子を伝播させる。これは，ある格子点へその隣の格子点から粒子を伝播させる方法である。27 ～ 31 行目は，伝播に関係の無い格子点自身（方向 0）の状態のコピーである。格子点自身の処理は，FHP1 モデルのシミュレーションを行う場合は不要であるため，27 行目の if 文でその場合は行わないようにした。33 行目では，サブルーチン rehash を呼び出し，格子点毎の粒子配置が格納されている配列 la を 10 進数変換して配列 astate へ代入する。

35 ～ 47 行目はエラー処理である。この処理は，デバッグモードでの実行時のみに行わせる。そこで，35 行目において if 文でデバッグモードか否かを判定している。FHP1 モデルのシミュレーションの場合にも対応できるように，Fig.2.2と同様の考え方で istart ～ nsite の範囲で回転させるようにした。

(3) サブルーチン rehash

本ルーチンは，並列化によって新たに加えられたサブルーチンであり，配列 la に一次的に格納されている格子点毎の粒子配置を 10 進数に変換し，配列 astate や bstate に格納する。並列化コードの本ルーチンを Fig.5.41 に示す。

FHP2 モデルと FHP3 モデルのシミュレーションの場合に 10 進数への変換は 7 ビットまで考慮するのに対し，FHP1 モデルのシミュレーションの場合は，格子点自身（方向 0）の状態がないため 6 ビットまで考慮すればよい。変換方法は，サブルーチン irehash の場合と同様である。

5.5 壁における跳ね返し部 (フェーズ 1)

この部分では，trans において 2 次元格子空間の境界部に設定した壁に粒子が伝播した場合に，その粒子を逆方向へ跳ね返す処理を行う。よって，壁を設定した場合のみに実行される部分である。跳ね返しの処理は，1 つの格子点で閉じているため，格子点毎に独立な処理になる。フェーズ 2 の壁における跳ね返し部もこの部分と同様である。

5.5.1 メインルーチンについて

・シングルコード

Fig.5.42 において 2 ～ 3 行目はサブルーチン chwall を呼び出し，壁における格子点上の粒子をチェックし，壁の存在する方向に粒子が存在する場合に同じ格子点上の逆方向へ向け直す。trans の結果が a_state に得られるので，ここでは，跳ね返し前の粒子の状態が配列 a_state に，跳ね返し後の粒子の状態が配列 state に格納される。5 ～ 9 行目はデバッグモードでの実行時に chwall の結果をチェックするために x 方向と y 方向のモーメントと方向毎の粒子総

数，格子上の粒子総数を求めて出力する。11 行目では astate の内容を a_state にコピーする。

フェーズ 1 において，粒子の伝播部と衝突部は必ず実行される部分であるが，chwall 部は実行されない場合がある。よって，粒子の伝播部では処理前の状態を state，処理後の状態を a_state，衝突部では処理前の状態を a_state，処理後の状態を state として，粒子の伝播部と衝突部で配列を循環させている。そこで，壁における粒子の跳ね返し部では次の衝突部のために結果を a_state に代入する必要がある。

・並列化コード

Fig.5.43 において 7 ～ 10 行目では壁での粒子の跳ね返しを行う chwall を呼び出し，エラーがあった場合はエラー出力を行う。trans の結果が astate に得られているので，ここでは，跳ね返し前の粒子の状態は astate に，跳ね返し後の粒子の状態は bstate に格納する。11 ～ 13 行目は bstate の内容を astate にコピーする。これは，chwall が実行されない場合もあるためである。15 ～ 22 行目は，粒子数やモーメントを求める処理である。

5.5.2 サブルーチンについて

(1) サブルーチン chwall

本ルーチンについては，シングルコードと並列化コードではアルゴリズムは同一であり，異なるのは，元の 2 次元格子空間の全格子点数のループを分割領域の全格子点数のループへ変更したことである。エラー処理については，trans と同様の変更を行った。シングルコードの本ルーチンを Fig.5.44 に，並列化コードの本ルーチンを Fig.5.45 に示す。

5.6 衝突部 (フェーズ 1)

この部分は，FHP の衝突規則を用い，粒子同士の衝突処理を行う。衝突処理も 1 つの格子点で閉じた処理であるため，格子点毎に独立した処理である。フェーズ 2 の衝突部もこの部分と同様である。

5.6.1 メインルーチンについて

・シングルコード

Fig.5.46 において 1 ～ 13 行目はシミュレーションパラメータで指定される model の値によって，サブルーチン fhp1 か fhp2，または fhp3 が呼ばれ，衝突処理が行われる。衝突前の粒子の状態は a_state に，衝突後の粒子の状態は state に格納する。15 ～ 19 行目は x 方向と y 方向のモーメント等を求める処理である。

・並列化コード

Fig.5.47 において，1～7 行目では，シミュレーションパラメータで指定する model の値により，サブルーチン fhp1 か fhp2，または fhp3 を呼び出し，粒子の衝突処理を行う。エラーがあった場合はエラー出力を行う。衝突前の粒子の状態は astate に，衝突後の粒子の状態は bstate に格納する。

5.6.2 サブルーチンについて

(1) サブルーチン fhp1，fhp2，fhp3

サブルーチン fhp1，fhp2，fhp3 はそれぞれ，FHP1，FHP2，FHP3 モデルの衝突処理を行う。これらのサブルーチンにおける並列化のための主な変更は，格子点数のループ分割のみであり，ループ中のアルゴリズムは，シングルコードと同様である。また，エラー処理部があるが，これも trans や chwall と同様にデバッグモードでの実行時だけ有効化できるように if 文で囲んだ。

これらのルーチンはステップ数が多いため，変更部分のみについて，サブルーチン fhp1 については，シングルコードを Fig.5.48，並列化コードを Fig.5.49 に，サブルーチン fhp2 については，シングルコードを Fig.5.50，並列化コードを Fig.5.51 に，サブルーチン fhp3 については，シングルコードを Fig.5.52，並列化コードを Fig.5.53 にそれぞれ示す。

5.7 データ出力部 (フェーズ 1)

5.7.1 メインルーチンについて

・シングルコード

フェーズ 1 の結果の出力を行うため，サブルーチン outfiles を呼ぶ（Fig.5.54）。

・並列化コード

シングルコードと同様にフェーズ 1 の結果の出力を行うため，サブルーチン outfiles を呼ぶ（Fig.5.55）。ここでは引数渡しで配列 bstate と出力ファイルに結合させる機番 n を渡す。

5.7.2 サブルーチンについて

(1) サブルーチン outfiles

本ルーチンでは，計算結果の出力を行う。シングルコードでは，2 次元格子空間すべての各格子点のデータを出力していたのに対し，並列化コードでは，分割領域の主領域部の各格子点のデータをプロセス毎に出力させることにした。よって，並列化コードでは出力ファイルの個数がプロセス数になる。シングルコードの本ルーチンを Fig.5.56 に，並列化コードの本ルーチンを Fig.5.57 に示す。

・シングルコード - サブルーチン outfiles

Fig.5.56 において 12 行目が格子点数のループであり，このループ中で出力が行われる。出力内容は，18 ～ 28 行目に示すデータであるが，FHP1 モデルのシミュレーションでは，格子点自身（方向 0）については考慮しないため，これとは別に，31 ～ 40 行目で出力を行う。13 ～ 14 行目で求められる i，j は，格子点 ID k における x 方向と y 方向の値である。これらを用いて 15 行目において 2 次元格子空間の範囲内か否かを判定し出力を行う。

・並列化コード - サブルーチン outfiles

並列化コードにおいても処理構成は，シングルコードと同様である。16 ～ 26 行目が FHP2 と FHP3 モデルのシミュレーションのデータ，28 ～ 37 行目が fhp1 モデルのシミュレーションのデータである。同じように，11 ～ 12 行目において分割領域の格子点 ID k1 における x 方向と y 方向の値を求め，13 行目の if 文において主領域部か否かを判定して出力を行う。

ここで，15 行目と 28 行目の j の出力が，シングルコードと異なる。ここでの j は元の 2 次元格子空間上の値でなければならないが，並列化コードでは，各プロセスが分割領域を担当するため，単純に j を出力することはできない。そこで，j-iwidth+(iys*myrank) を出力することにした。各分割領域には袖部が含まれるため，そのサイズ iwidth を引く。更に iys*myrank を足せば，シミュレーションする全 2 次元格子空間における y 方向の値が決まる。43 行目では，乱数の種を出力する。ここで出力された種は，フェーズ 2 での初期種として使用する。これは並列化コードで新たに設けた機能である。

5.8 データ入力部 (フェーズ 2)

5.8.1 メインルーチンについて

・シングルコード

Fig.5.58 において，2 行目のサブルーチン readdata で，フェーズ 1 の結果を読み込み，格子点毎の粒子の状態は配列 state に格納する。3 行目と 5 行目によってフェーズ 2 の出力ファイルに結合させる機番 n を決定する。

・並列化コード

Fig.5.59 において，2 行目は，フェーズ 1 でプロセス毎に出力されたファイルに結合させる機番を決める。3 行目ではサブルーチン readdata を呼び出しフェーズ 1 の結果を読み込む。格子点毎の粒子の状態は配列 bstate に格納する。4 行目ではフェーズ 2 の結果を出力するファイルに結合させる機番を決める。並列化コードではプロセス毎にファイルが必要になるので，10+nprocs+myrank で n を求めれば，各プロセスで機番が重複せず，上書きを回避することができる。

5.8.2 サブルーチンについて

(1) サブルーチン readdata

本ルーチンでは，フェーズ 1 での計算結果を読み込む。シングルコードでは，2 次元格子空間の各格子点のデータを読み込んでいたのに対し，並列化では，各々のプロセスがデータを読み込み，各配列の主領域部の範囲に格納しなければならない。シングルコードの本ルーチンを Fig.5.60 に，並列化コードの本ルーチンを Fig.5.61 に示す。

・シングルコード - サブルーチン readdata

Fig.5.60 において 22 行目と 59 行目が格子点数のループであり，これらのループ中で入力が行われるが，一旦，一次変数に読み込んでから，コード内で使用する各配列に代入している。代入される各配列は，outfiles で出力するデータと同一である。23 行目での読み込みが FHP1 モデルのシミュレーションの場合で，60 行目が FHP2 と FHP3 モデルのシミュレーションの場合である。

・並列化コード - サブルーチン readdata

一旦出力したデータを再度読み込む場合は，出力した場合と同様のコーディングで読み込みを行えばよい。よって，並列化コードにおける本ルーチンは，並列化コードの outfiles のコーディングを流用した。54 行目では，フェーズ 1 で出力された乱数の種を読み込む。

5.9 粒子への運動量付加部 (フェーズ 2)

この部分では，2 次元格子空間上の粒子に対し運動量を与える。空間の上境界部の粒子に運動量を与える場合，下境界部の粒子に運動量を与える場合，空間を y 方向で 2 分割し上半分の領域と下半分の領域に分け，それぞれの領域で逆方向に運動量を与える場合の 3 通りについて可能である。

5.9.1 メインルーチンについて

・シングルコード

Fig.5.62 において 6 行目では粒子に運動量を与えるサブルーチン mvwall を呼び出す。運動量を与える前の粒子の状態は b_state に，運動量を与えた後の粒子の状態は a_state に格納する。8 ～ 10 行目は mvwall でエラーが発生した場合にエラー出力を行う部分である。11 行目においては a_state から state に内容をコピーする。13 ～ 21 行目では格子点毎かつ方向毎について運動量を与えた粒子をカウントする。

・並列化コード

Fig.5.63 において 7 行目では粒子に運動量を与えるサブルーチン mvwall を呼ぶ。運動量を与える前の粒子の状態は astate に，運動量を与えた後の粒子の状態は bstate に格納する。11 ～ 13 行目において bstate から astate へ内容のコピーを行う。14 ～ 18 行目では格子点毎かつ方向毎に運動量を与えられた粒子数をカウントする。

5.9.2 サブルーチンについて

(1) サブルーチン mvwall

本ルーチンでは，シミュレーションパラメータで指定するフラグによって，元の2次元格子空間の上境界部の粒子または下境界部の粒子，または空間の上半分と下半分の粒子に運動量を与える。並列化コードにおいてシングルコードと比較した場合の大きな変更点は，上半分と下半分に分ける場合の2次元領域の中央の決め方である。

シングルコードで対象とする2次元格子空間の y 方向の中央を求めるには，y 方向を単純に 1/2 すればよい。しかし，並列化コードでは，2次元格子空間が分割されているため，元の2次元格子空間の中央を求める処理を追加した。シングルコードの本ルーチンを Fig.5.64 に，並列化コードの本ルーチンを Fig.5.65 に示す。

・シングルコード - サブルーチン mvwall

本ルーチンは，シミュレーションする全2次元格子空間の上境界部の粒子に運動量を与える場合と下境界部の粒子に運動量を与える場合，上半分と下半分に分けて運動量を与える場合に if 文で場合分けが行われている。上境界部の粒子に運動量を与える場合は，上境界部の x 方向のサイズが処理ループの回転数になり，下境界部の粒子に運動量を与える場合は，下境界部の x 方向のサイズが処理ループの回転数になる。更に，上半分と下半分に分けて運動量を与える場合は，上半分の格子点数のループと下半分の格子点数のループで構成される。

・並列化コード - サブルーチン mvwall

シングルコードの場合と同様に，上境界部の粒子に運動量を与える場合，下境界部の粒子に運動量を与える場合，上半分と下半分に分けて運動量を与える場合に if 文で場合分けを行う。ただし，並列化コードにおいては，分割領域が対象になるため，上境界部の粒子に運動量を与える場合はランク 0 が担当する分割領域のみで，下境界部の粒子に運動量を与える場合はランク max が担当する分割領域のみを対象にすればよい。また，上半分と下半分の粒子に運動量を与える場合はすべての分割領域が対象になるが，中央が存在する分割領域を識別する必要がある。

Fig.5.65 の 1 行目で元の2次元格子空間の中央の y 方向の値を求め，8 ～ 11 行目でランク 0 から加算された y 方向のサイズがそれと同じになった場合，そのランク（idiv_point(1)）の分割領域の y 方向の値（idiv_point(2)）を元の2次元格子空間の中央とした。上境界部と下境界部の粒子に運動量を与える if 文内では，ランク 0 とランク max を判定する if 文を追加した。また，上半分と下半分の粒子に運動量を与える if 文内では，中央が存在するランク idiv_point(1) より小さいランクを判定する if 文，中央が存在するランクを判定する if 文，idiv_point(1) より大きいランクを判定する if 文を追加した。

5.10 格子点ペアテーブルの作成部 (フェーズ2)

この部分では，長距離相互作用の処理のため，相互作用する格子点のペアテーブルを作成する。

5.10.1　メインルーチンについて

・シングルコード

Fig.5.66 において，12 行目の lrange で，ある格子点と距離 r 離れた格子点同士のペア配列を作成する。ペア情報は配列 range に格納される。

・並列化コード

Fig.5.67 において，17 行目で，格子点毎のペアを作成するための lrange を呼ぶ。lrange ではある格子点と irr （並列化コードでは距離 r を irr とした）離れた格子点との間の壁の有無を判定するために，壁情報が必要になる。そこで，lrange を呼ぶ直前の 16 行目において袖部に対し壁情報のデータ転送を行うようにした。

5.10.2　サブルーチンについて

(1) サブルーチン lrange

本ルーチンでは，粒子間の長距離相互作用処理のために，ある格子点に対して距離 irr 分離れた格子点を検索し，格子点と相手格子点の間に壁が無い場合はペアを作成する。これは，実行時に最初の 1 回だけ呼ばれ，それ以降では同一のペアを用いて処理が行われる。本ルーチンにおける主な変更は，Fig.5.68 の do 15 ループの分割であり，他はシングルコードと同様の処理内容である。並列化コードでは，ペア情報は配列 itrange に格納される。シングルコードの本ルーチンを Fig.5.68 に，並列化コードの本ルーチンを Fig.5.69に示す。

5.11　長距離相互作用部 (フェーズ 2)

この部分では，サブルーチン lrange で求められた格子点同士のペア情報を用いて，それら 2 つの格子点における粒子間の相互作用を行う。

5.11.1　メインルーチンについて

・シングルコード

Fig.5.70 において 4 ～ 5 行目では，長距離相互作用を行うサブルーチン lgcol を呼ぶ。作用前の粒子の状態は b_state に，作用後の粒子の状態は a_state に格納する。2 行目において state から b_state に内容をコピーする。6 ～ 8 行目は lgcol でエラーが発生した場合にエラー出力を行う部分である。9 行目では a_state から state に内容をコピーする。11 ～ 19 行目では格子点毎かつ方向毎に相互作用した粒子をカウントする。

・並列化コード

Fig.5.71 において 12 行目では lgcol を呼ぶ。lgcol では格子点同士のペア情報を用い，

それらの格子点に存在する粒子の相互作用を行う。その際袖部データの参照も行われる可能性があるため，予め袖部を最新のデータにしておく必要がある。作用前の粒子の状態は astate に，作用後の粒子の状態は bstate に格納する。16 ～ 18 行目では，配列の循環に合わせるため，bstate から astate へ内容のコピーを行う。19 ～ 23 行目では格子点毎かつ方向毎に相互作用の対象になった粒子数のカウントを行う。

5.11.2 サブルーチンについて

(1) サブルーチン lgcol

このサブルーチンでは，lrange で作成された格子点同士のペア情報を用いて，それら格子点における粒子と粒子の長距離相互作用を行う。本ルーチンにおける主な変更点は，2 次元格子空間の全格子点数のループ分割であり，ループ中の処理は，シングルコードと同様である。本ルーチンの変更部分のみについて，シングルコードを Fig.5.72 に，並列化コードを Fig.5.73 に示す。

5.12 データ出力部 (フェーズ 2)

この部分は，フェーズ 2 の結果を出力する。フェーズ 1 の 出力部と異なり，シミュレーションパラメータで指定される frate （並列化コードでは ifrate）を用い，時間発展ループの回転数が frate 毎に出力を行う。また，シミュレーションパラメータで指定される trate （並列化コードでは itrate）を用い，trate 毎に全系の圧力を求める。

5.12.1 メインルーチンについて

・シングルコード

Fig.5.74 において 2 行目ではフェーズ 2 の結果を出力するためにサブルーチン outfiles を呼ぶ。outfiles はループ回転数 s が frate の倍数の時に呼ばれる。4 行目の n はファイルと結合する機番であるが，frate によって複数回に渡って結果を出力する場合があるため，1 回出力を行った後は，n+1 を行い，ファイルの機番を変更している。11 ～ 17 行目ではループ回転数 s が trate で割り切れる時に全系の圧力としての値 pp を求める。22 ～ 24 行目では同じく trate で割り切れる時に配列 acumtr，acumlg，acumbo を 0 で初期化する。

・並列化コード

Fig.5.75 において 7 行目ではフェーズ 2 の結果を出力するため，サブルーチン outfiles を呼ぶ。この outfiles はループ回転数が ifrate で割り切れた時に呼ばれる。よって，複数回呼ばれる可能性がある。そこで，8 行目の n によって，再度 outfiles が呼ばれた時に出力ファイルに結合させる機番を前の出力ファイルを上書きしないように変更している。

13 ～ 18 行目はループ回転数が itrate で割り切れる時 pp を求めるが，ここでの pp はプロ

セス毎の pp にすぎないので，プロセス間で pp を総和する必要がある。これを行うためにサブルーチン pp_sum を作成し 19 行目で呼ぶようにした。24 ～ 30 行目も同じく itrate で割り切れる時に配列 acumtr， acumlg， acumbo を 0 で初期化する。

5.12.2　サブルーチンについて

(1) サブルーチン pp_sum

本ルーチンは，シングルコードで求められる pp の値についてプロセス間で総和を行うルーチンである。サブルーチン pp_sum を Fig.5.76 に示す。8 ～ 9 行目において pp がプロセス間で総和され結果が spp に求められる。この spp は標準出力へ出力するのみであるため，MPI ライブラリは mpi_reduce を用いた。

5.13　最終処理部

この部分は，シミュレーション結果として，x 方向と y 方向のモーメントと方向毎の粒子総数，全格子点の粒子総数を求め，標準出力先とファイルへの出力を行う。シングルコードの最終処理部を Fig.5.77 に，並列化コードを Fig.5.78 に示す。

・シングルコード

Fig.5.77 において 3 行目でサブルーチン countPQ を呼び出し，各方向のモーメントと方向毎の粒子総数，全格子点の粒子総数を求める。これを 4 ～ 7 行目において，機番 99 と結合されるファイルと標準出力先へ出力する。9 ～ 26 行目でも，シミュレーションパラメータを機番 99 と結合されるファイルと標準出力先へ出力する。

・並列化コード

Fig.5.78 において 5 ～ 6 行目で countPQ と countPQ_sum を呼び出し，各方向のモーメントと方向毎の粒子総数，全格子点の粒子総数を求める。これを 7 ～ 12 行目において機番 99 と結合されるファイルと標準出力へ出力する。出力はランク 0 のみで行った。14 ～ 33 行目でも，ランク 0 においてシミュレーションパラメータを同様なファイルと標準出力へ出力した。

```
 1       integer xs,ys,opt,step,frate,trate,rr,model,steps,phase
 2       integer optw,optwo,mvpos,mvdirc,debug
 3       real den,mvrate,rvel
 4
 5
 6       parameter(xs=300,ys=300,step=1200,steps=1,phase=2)
 7       parameter(den=0.1,rr=0,opt=4)
 8       parameter(frate=1200,trate=300)
 9       parameter(optw=2,mvrate=0.1,mvpos=1,mvdirc=1)
10       parameter(nsite=7,model=3,debug=0)
11       parameter(rvel=0.0)
12
13       integer site(1:xs*ys,0:nsite-1)
14       integer wall(1:xs*ys,0:nsite-1)
15       integer state(1:xs*ys,0:nsite-1)
16       integer range(1:xs*ys,0:nsite-1)
17       integer acumtr(1:xs*ys,0:nsite-1)
18       integer acumlg(1:xs*ys,0:nsite-1)
19       integer acumbo(1:xs*ys,0:nsite-1)
20       integer b_state(1:xs*ys,0:nsite-1)
21       integer a_state(1:xs*ys,0:nsite-1)
22       integer iacum(1:xs*ys,0:nsite-1)
23       integer iacum2(1:xs*ys,0:nsite-1)
24       integer iacum3(1:xs*ys,0:nsite-1)
25
26       integer iduma(1:xs*ys)
27       integer idumb(1:xs*ys)
28
29       integer p_num(0:nsite-1)
30
31
32       real pp
33       integer kx,ky,km,kr,ku,kl,kd
34
35       integer numsite,ierr,br
35
36       integer n,itmp,s,i,j,k,vel
37       integer start,tdirc,dirc
38       integer numpar,momx,momy
```

Fig. 5.1 Variable definition of the non-parallelized code

```
 1       program LGM_MAIN
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
```

Fig. 5.2 Variable definition of the parallelized code

```
 1       integer*4 istep,isteps,iphase,ioptw,ifrate,itrate,irr
 2       integer*4 imvpos,imvdirc,ioptwo,idebug,iloop
 3       real*4    mvrate,rvel
 4       integer*4 ixs,iyss,istart,nsite,model
 5       integer*4 iopt,iwidth,iys,numsite_para,numsite
 6       real*4    den
 7
 8       parameter(ixs=300,iyss=300,istep=1200,isteps=1,iphase=2)
 9
10       parameter(istart=0,nsite=6,model=3)
11       parameter(ioptw=2)
12
13       parameter(den=0.1,irr=0,iopt=4)
14       parameter(ifrate=1200,itrate=300)
15
16       parameter(mvrate=0.1,imvpos=1,imvdirc=1)
17       parameter(idebug=0)
18
19 C***** For single **************
20 C--- NPROCS = 1 ---
21       parameter(iwidth=0)
22       integer*4 itrans_array(ixs*1,2)
23 C***** For parallel ************
24 C--- NPROCS > 1 .and. irr = 0 ---
25 c     parameter(iwidth=1)
26 C--- NPROCS > 1 .and. irr > 0 ---
27 c     parameter(iwidth=irr)
28 c     integer*4 itrans_array(ixs*iwidth,2)
29 C*******************************
30
31
32       parameter(iys=iyss/ipe)
33       parameter(numsite_para=ixs*iys+ixs*iwidth*2)
34       parameter(numsite=ixs*iys)
35
36       integer*4 isite(1:numsite_para,istart:nsite)
37       integer*4 tab(0:127,istart:nsite)
38       integer*4 tabw(0:127,istart:nsite)
39       integer*4 la(1:numsite_para,istart:nsite)
40       integer*4 iwall(0:numsite_para)
41       integer*4 bstate(0:numsite_para)
42       integer*4 astate(0:numsite_para)
43       integer*4 iacum(1:numsite_para,istart:nsite)
44       integer*4 iduma(1:numsite_para)
45       integer*4 acumtr(1:numsite_para,istart:nsite)
46       integer*4 acumlg(1:numsite_para,istart:nsite)
47       integer*4 acumbo(1:numsite_para,istart:nsite)
48       integer*4 itrange(1:numsite_para,istart:nsite)
```

Fig. 5.3 INC_DATA file (1/2)

```
49
50        common/original1/isite,tab,tabw,la,iwall,bstate,astate
51        common/original2/iacum,iduma,acumtr,acumlg,acumbo,itrange
52
53        integer*4 iseed
54        real*4 rand(1)
55
56        common/parallel/iseed,itrans_array,rand
57
58        integer*4 p_num(istart:nsite),momx,momy,numpar
59        integer*4 sp_num(istart:nsite),smomx,smomy,snumpar
60
61        common/debug1/p_num,momx,momy,numpar
62        common/debug2/sp_num,smomx,smomy,snumpar
63
64        real*4 pp,spp
65
66        common/total_p/pp,spp
67
68        integer*4 ierr
69
70        common/ERR/ierr
71
72        integer*4 ic(istart:nsite)
73
74        integer*4 k1,k2,k3,k4,k5
```

Fig. 5.3 INC_DATA file (2/2)

```
1 *include mpif.h
2         integer*4 NPROCS,MYRANK,ipe
3         common/PARA_PE/NPROCS,MYRANK
4         integer*4 status(MPI_STATUS_SIZE)
5         parameter(ipe=4)
```

Fig. 5.4 INC_MPI file

```
      numsite=xs*ys
      br=0

      write(6,*) "xs    = ",xs
      write(6,*) "ys    = ",ys
      write(6,*) "nsite= ",nsite
      write(6,*) "den   = ",den
      write(6,*) "opt   = ",opt
      write(6,*) "step = ",step
      write(6,*) "frate = ",frate
      write(6,*) "trate = ",trate
      write(6,*) "rr    = ",rr
      write(6,*) "model= ",model


c     /***** making hexagonal lattice *****/
      call mkclt3(xs,ys,site)

c     /***** making hexagonal lattice *****/
      if(optw.ne.0) then
         if(optw.eq.1) then
            call mkwalla(xs,ys,wall)
         else if(optw.eq.2) then
            call mkwallb(xs,ys,wall)
         else if(optw.eq.3) then
            call mkwallc(xs,ys,wall)
         endif
      else
         call initdat2(xs,ys,wall,0)
      endif

      if(model.eq.1) then
         start=1
      else if((model.eq.2).or.(model.eq.3)) then
         start=0
      endif

c     /***** initializing state *****/
      call initdat2(xs,ys,state,0)
      call initdat2(xs,ys,acumtr,0)
      call initdat2(xs,ys,acumlg,0)
      call initdat2(xs,ys,acumbo,0)
```

Fig. 5.5 Initialization part in the main routine of the non-parallelized code

```
 1       call mpi_init(ierr)
 2       call mpi_comm_size(mpi_comm_world,NPROCS,ierr)
 3       call mpi_comm_rank(mpi_comm_world,MYRANK,ierr)
 4
 5       ibr=0
 6
 7       if(MYRANK.eq.0) then
 8         write(6,*) 'ixs    = ',ixs
 9         write(6,*) 'iys    = ',iys
10         write(6,*) 'nsite  = ',nsite-istart+1
11         write(6,*) 'den    = ',den
12         write(6,*) 'iopt   = ',iopt
13         write(6,*) 'istep  = ',istep
14         write(6,*) 'ifrate = ',ifrate
15         write(6,*) 'itrate = ',itrate
16         write(6,*) 'irr    = ',irr
17         write(6,*) 'model  = ',model
18       endif
19 C***************************************
20 C
21 C      CALL MKHTABLE
22 C
23 C***************************************
24       call mkhtable(tab,1)
25       call mkhtable(tabw,2)
26
27       astate(0)=0
28       bstate(0)=0
29       iwall(0)=0
30
31 C***************************************
32 C
33 C      CALL MKCLT
34 C
35 C***************************************
36       if(mod(iys,2).eq.0) then
37         if(mod(iwidth,2).eq.0) then
38           call mkclt3_1
39         else
40           call mkclt3_2
41         endif
42       else
43         if(mod(iwidth,2).eq.0) then
44           if(mod(MYRANK,2).eq.0) then
45             call mkclt3_1
46           else
47             call mkclt3_2
48           endif
49         else
50           if(mod(MYRANK,2).eq.0) then
```

Fig. 5.6 Initialization part in the main routine of the parallelized code (1/2)

```
51              call mkclt3_2
52            else
53              call mkclt3_1
54            endif
55          endif
56        endif
57
58 C**************************************
59 C
60 C     CALL MKWALL
61 C
62 C**************************************
63        if(ioptw.eq.1) then
64          if(mod(iys,2).eq.0) then
65            call mkwalla1              ← 主領域部の始まりが横に凹
66          else
67            if(mod(MYRANK,2).eq.0) then
68              call mkwalla1
69            else
70              call mkwalla2            ← 主領域部の始まりが横に凸
71            endif
72          endif
73        elseif(ioptw.eq.2) then
74          call mkwallb
75        elseif(ioptw.eq.3) then
76          if(mod(iys,2).eq.0) then
77            call mkwallc1
78          else
79            if(mod(MYRANK,2).eq.0) then
80              call mkwallc1            ← 主領域部の始まりが横に凹
81            else
82              call mkwallc2            ← 主領域部の始まりが横に凸
83            endif
84          endif
85        else
86          do k2=1,numsite_para
87            iwall(k2)=0
88          enddo
89        endif
90
91        do k1=istart,nsite
92        do k2=1,numsite_para
93          acumtr(k2,k1)=0
94          acumlg(k2,k1)=0
95          acumbo(k2,k1)=0
96        enddo
97        enddo
```

Fig. 5.6 Initialization part in the main routine of the parallelized code (2/2)

```
 1       subroutine mkclt3(xx,yy,tsite)
 2
 3       integer xx,yy
 4       integer tsite(1:xx*yy,0:6)
 5
 6       integer i,j,k,s,itmp,jtmp
 7
 8       call initdat2(xx,yy,tsite,0)
 9
10       do 10 j=1,yy
11          do 10 i=1,xx
12             s=xx*(j-1)+i
13             itmp=mod(s,xx)
14             jtmp=mod(j,2)
15
16             do 20 k=0,6
17                if(k.eq.0) then    ← 方向０の場合
18                   tsite(s,k)=s
19
20                else if(k.eq.1) then    ← 方向１の場合
21                   if(itmp.eq.0) then
22                      tsite(s,k)=s-xx+1
23                   else
24                      tsite(s,k)=s+1
25                   endif
26
27                else if(k.eq.2) then    ← 方向２の場合
28                   if((s.ge.1).and.(s.le.xx-1)) then
29                      tsite(s,k)=s+xx*(yy-1)+1
30                   else if(s.eq.xx) then
31                      tsite(s,k)=xx*(yy-1)+1
32                   else if(jtmp.eq.0) then
33                      tsite(s,k)=s-xx
34                   else if(itmp.eq.0.and.jtmp.eq.1) then
35                      tsite(s,k)=s-2*xx+1
36                   else if(itmp.ne.0.and.jtmp.eq.1) then
37                      tsite(s,k)=s-xx+1
38                   endif
39
40                else if(k.eq.3) then    ← 方向３の場合
41                   if((s.ge.1).and.(s.le.xx)) then
42                      tsite(s,k)=s+xx*(yy-1)
43                   else if((jtmp.eq.0).and.(itmp.eq.1)) then
44                      tsite(s,k)=s-1
45                   else if((jtmp.eq.0).and.(itmp.ne.1)) then
46                      tsite(s,k)=s-xx-1
47                   else if(jtmp.eq.1) then
48                      tsite(s,k)=s-xx
49                   endif
50
```

Fig. 5.7 Subroutine mkclt3 of the non-parallelized code (1/2)

```
51                else if(k.eq.4) then    ← 方向４の場合
52                   if(itmp.eq.1.or.s.eq.1) then
53                      tsite(s,k)=s+xx-1
54                   else
55                      tsite(s,k)=s-1
56                   endif
57
58                else if(k.eq.5) then    ← 方向５の場合
59                   if((s.ge.xx*(yy-1)+2)
60     +                .and.(s.le.xx*yy)) then
61                        tsite(s,k)=s-xx*(yy-1)-1
62                   else if(s.eq.xx*(yy-1)+1) then
63                        tsite(s,k)=xx
64                   else if((jtmp.eq.0).and.(itmp.eq.1)) then
65                        tsite(s,k)=s+2*xx-1
66                   else if((jtmp.eq.0).and.(itmp.ne.1)) then
67                        tsite(s,k)=s+xx-1
68                   else if(jtmp.eq.1) then
69                        tsite(s,k)=s+xx
70                   endif
71
72                else if(k.eq.6) then    ← 方向６の場合
73                   if((s.ge.xx*(yy-1)+1)
74     +                .and.(s.le.xx*yy)) then
75                      tsite(s,k)=s-xx*(yy-1)
76                   else if((jtmp.eq.1).and.(itmp.eq.0)) then
77                      tsite(s,k)=s+1
78                   else if((jtmp.eq.1).and.(itmp.ne.0)) then
79                      tsite(s,k)=s+xx+1
80                   else if(jtmp.eq.0) then
81                      tsite(s,k)=s+xx
82                   endif
83
84                endif
85 20           continue
86 10     continue
87        return
88        end
```

Fig. 5.7 Subroutine mkclt3 of the non-parallelized code (2/2)

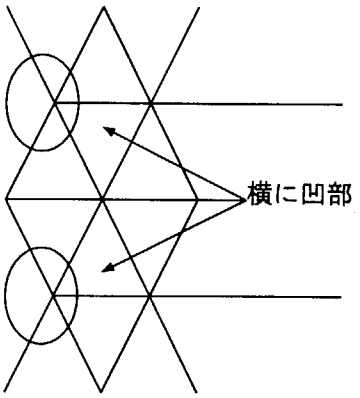

Fig. 5.8 Concave side of the 2-dimensional lattice

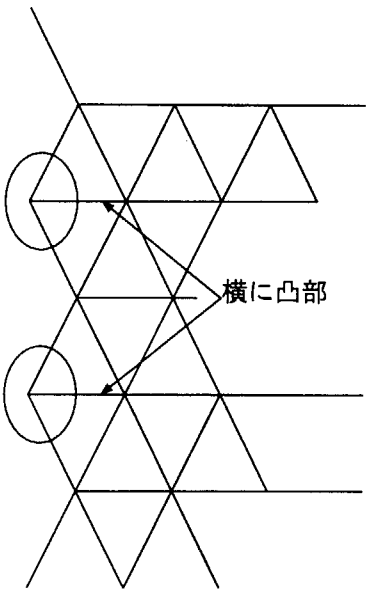

Fig. 5.9 Convex side of the 2-dimensional lattice

```
 1       subroutine mkclt3_1
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         isite(k2,k1)=0
 9       enddo
10       enddo
11
12       iyy=iys+iwidth*2
13
14       do 10 k1=1,iyy
15         do 20 k2=1,ixs
16           is=ixs*(k1-1)+k2
17           itmp=mod(is,ixs)
18           jtmp=mod(k1,2)
19
20           do 30 k3=istart,nsite
21             if(k3.eq.0) then    ← 方向０の場合
22               isite(is,k3)=is
23
24             elseif(k3.eq.1) then    ← 方向１の場合
25               if(itmp.eq.0) then
26                 isite(is,k3)=is-ixs+1
27               else
28                 isite(is,k3)=is+1
29               endif
30
31             elseif(k3.eq.2) then    ← 方向２の場合
32               if( (is.ge.1).and.(is.le.ixs-1) ) then
33                 isite(is,k3)=is+ixs*(iyy-1)+1
34               elseif(is.eq.ixs) then
35                 isite(is,k3)=ixs*(iyy-1)+1
36               elseif(itmp.eq.0.and.jtmp.eq.0) then
37                 isite(is,k3)=is-ixs
38               elseif(itmp.eq.0.and.jtmp.ne.0) then
39                 isite(is,k3)=is-2*ixs+1
40               elseif(itmp.ne.0.and.jtmp.eq.0) then
41                 isite(is,k3)=is-ixs
42               elseif(itmp.ne.0.and.jtmp.ne.0) then
43                 isite(is,k3)=is-ixs+1
44               elseif(jtmp.eq.0) then
45                 isite(is,k3)=is-ixs
46               elseif(jtmp.ne.0) then
47                 isite(is,k3)=is-ixs+1
48               endif
49
```

Fig. 5.10 Subroutine mkclt3_1 of the parallelized code (1/3)

```
50              elseif(k3.eq.3) then   ← 方向３の場合
51                if( (is.ge.1).and.(is.le.ixs) ) then
52                  isite(is,k3)=is+ixs*(iyy-1)
53                elseif(itmp.eq.1.and.jtmp.eq.0) then
54                  isite(is,k3)=is-1
55                elseif(itmp.eq.1.and.jtmp.ne.0) then
56                  isite(is,k3)=is-ixs
57                elseif(itmp.ne.1.and.jtmp.eq.0) then
58                  isite(is,k3)=is-ixs-1
59                elseif(itmp.ne.1.and.jtmp.ne.0) then
60                  isite(is,k3)=is-ixs
61                elseif(jtmp.eq.0) then
62                  isite(is,k3)=is-ixs-1
63                elseif(jtmp.ne.0) then
64                  isite(is,k3)=is-ixs
65                endif
66
67              elseif(k3.eq.4) then   ← 方向４の場合
68                if(itmp.eq.1.or.is.eq.1) then
69                  isite(is,k3)=is+ixs-1
70                else
71                  isite(is,k3)=is-1
72                endif
73
74              elseif(k3.eq.5) then   ← 方向５の場合
75                if(is.eq.ixs*(iyy-1)+1) then
76                  isite(is,k3)=ixs
77                elseif( (is.ge.ixs*(iyy-1)+2).and.(is.le.ixs*iyy) ) then
78                  isite(is,k3)=is-ixs*(iyy-1)-1
79                elseif(itmp.eq.1.and.jtmp.eq.0) then
80                  isite(is,k3)=is+2*ixs-1
81                elseif(itmp.eq.1.and.jtmp.ne.0) then
82                  isite(is,k3)=is+ixs
83                elseif(itmp.ne.1.and.jtmp.eq.0) then
84                  isite(is,k3)=is+ixs-1
85                elseif(itmp.ne.1.and.jtmp.ne.0) then
86                  isite(is,k3)=is+ixs
87                elseif(jtmp.eq.0) then
88                  isite(is,k3)=is+ixs-1
89                elseif(jtmp.ne.0) then
90                  isite(is,k3)=is+ixs
91                endif
92
```

Fig. 5.10 Subroutine mkclt3_2 of the parallelized code (2/3)

```
93              elseif(k3.eq.6) then   ← 方向６の場合
94                if( (is.ge.ixs*(iyy-1)+1).and.(is.le.ixs*iyy) ) then
95                  isite(is,k3)=is-ixs*(iyy-1)
96                elseif(itmp.eq.0.and.jtmp.eq.0) then
97                  isite(is,k3)=is+ixs
98                elseif(itmp.eq.0.and.jtmp.ne.0) then
99                  isite(is,k3)=is+1
100               elseif(itmp.ne.0.and.jtmp.eq.0) then
101                 isite(is,k3)=is+ixs
102               elseif(itmp.ne.0.and.jtmp.ne.0) then
103                 isite(is,k3)=is+ixs+1
104               elseif(jtmp.eq.0) then
105                 isite(is,k3)=is+ixs
106               elseif(jtmp.ne.0) then
107                 isite(is,k3)=is+ixs+1
108               endif
109
110             endif
111     30     continue
112
113     20   continue
114     10 continue
115
116        if(NPROCS.ne.1) then
117          do k1=1,ixs
118            isite(k1,2)=-1
119            isite(k1,3)=-1
120          enddo
121          do k1=numsite_para-ixs+1,numsite_para
122            isite(k1,5)=-1
123            isite(k1,6)=-1
124          enddo
125        endif
126
127        return
128        end
```

Fig. 5.10 Subroutine mkclt3_1 of the parallelized code (3/3)

```
 1       subroutine mkclt3_2
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         isite(k2,k1)=0
 9       enddo
10       enddo
11
12       iyy=iys+iwidth*2
13
14       do 10 k1=1,iyy
15         do 20 k2=1,ixs
16           is=ixs*(k1-1)+k2
17           itmp=mod(is,ixs)
18           jtmp=mod(k1,2)
19
20           do 30 k3=istart,nsite
21             if(k3.eq.0) then    ← 方向０の場合
22               isite(is,k3)=is
23
24             elseif(k3.eq.1) then    ← 方向１の場合
25               if(itmp.eq.0) then
26                 isite(is,k3)=is-ixs+1
27               else
28                 isite(is,k3)=is+1
29               endif
30
31             elseif(k3.eq.2) then    ← 方向２の場合
32               if( (is.ge.1).and.(is.le.ixs) ) then
33                 isite(is,k3)=is+ixs*(iyy-1)
34               elseif(itmp.eq.0.and.jtmp.eq.0) then
35                 isite(is,k3)=is-2*ixs+1
36               elseif(itmp.eq.0.and.jtmp.ne.0) then
37                 isite(is,k3)=is-ixs
38               elseif(itmp.ne.0.and.jtmp.eq.0) then
39                 isite(is,k3)=is-ixs+1
40               elseif(itmp.ne.0.and.jtmp.ne.0) then
41                 isite(is,k3)=is-ixs
42               elseif(jtmp.eq.0) then
43                 isite(is,k3)=is-ixs+1
44               elseif(jtmp.ne.0) then
45                 isite(is,k3)=is-ixs
46               endif
47
```

Fig. 5.11 Subroutine mkclt3_2 of the parallelized code (1/3)

```
48              elseif(k3.eq.3) then    ← 方向３の場合
49                if(is.eq.1) then
50                  isite(is,k3)=is+ixs*iyy-1
51                elseif( (is.ge.2).and.(is.le.ixs) ) then
52                  isite(is,k3)=is+ixs*(iyy-1)-1
53                elseif(itmp.eq.1.and.jtmp.eq.0) then
54                  isite(is,k3)=is-ixs
55                elseif(itmp.eq.1.and.jtmp.ne.0) then
56                  isite(is,k3)=is-1
57                elseif(itmp.ne.1.and.jtmp.eq.0) then
58                  isite(is,k3)=is-ixs
59                elseif(itmp.ne.1.and.jtmp.ne.0) then
60                  isite(is,k3)=is-ixs-1
61                elseif(jtmp.eq.0) then
62                  isite(is,k3)=is-ixs
63                elseif(jtmp.ne.0) then
64                  isite(is,k3)=is-ixs-1
65                endif
66
67              elseif(k3.eq.4) then    ← 方向４の場合
68                if(itmp.eq.1.or.is.eq.1) then
69                  isite(is,k3)=is+ixs-1
70                else
71                  isite(is,k3)=is-1
72                endif
73
74              elseif(k3.eq.5) then    ← 方向５の場合
75                if( (is.ge.ixs*(iyy-1)+1).and.(is.le.ixs*iyy) ) then
76                  isite(is,k3)=is-ixs*(iyy-1)
77                elseif(itmp.eq.1.and.jtmp.eq.0) then
78                  isite(is,k3)=is+ixs
79                elseif(itmp.eq.1.and.jtmp.ne.0) then
80                  isite(is,k3)=is+2*ixs-1
81                elseif(itmp.ne.1.and.jtmp.eq.0) then
82                  isite(is,k3)=is+ixs
83                elseif(itmp.ne.1.and.jtmp.ne.0) then
84                  isite(is,k3)=is+ixs-1
85                elseif(jtmp.eq.0) then
86                  isite(is,k3)=is+ixs
87                elseif(jtmp.ne.0) then
88                  isite(is,k3)=is+ixs-1
89                endif
90
```

Fig. 5.11 Subroutine mkclt3_2 of the parallelized code (2/3)

```
91              elseif(k3.eq.6) then   ← 方向６の場合
92                if( (is.ge.ixs*(iyy-1)+1).and.(is.le.ixs*iyy-1) ) then
93                  isite(is,k3)=is-ixs*(iyy-1)+1
94                elseif(is.eq.ixs*iyy) then
95                  isite(is,k3)=is-ixs*iyy+1
96                elseif(itmp.eq.0.and.jtmp.eq.0) then
97                  isite(is,k3)=is+1
98                elseif(itmp.eq.0.and.jtmp.ne.0) then
99                  isite(is,k3)=is+ixs
100               elseif(itmp.ne.0.and.jtmp.eq.0) then
101                 isite(is,k3)=is+ixs+1
102               elseif(itmp.ne.0.and.jtmp.ne.0) then
103                 isite(is,k3)=is+ixs
104               elseif(jtmp.eq.0) then
105                 isite(is,k3)=is+ixs+1
106               elseif(jtmp.ne.0) then
107                 isite(is,k3)=is+ixs
108               endif
109
110             endif
111     30    continue
112
113     20  continue
114     10 continue
115
116       if(NPROCS.ne.1) then
117         do k1=1,ixs
118           isite(k1,2)=-1
119           isite(k1,3)=-1
120         enddo
121         do k1=numsite_para-ixs+1,numsite_para
122           isite(k1,5)=-1
123           isite(k1,6)=-1
124         enddo
125       endif
126
127       return
128       end
```

Fig. 5.11 Subroutine mkclt3_2 of the parallelized code (3/3)

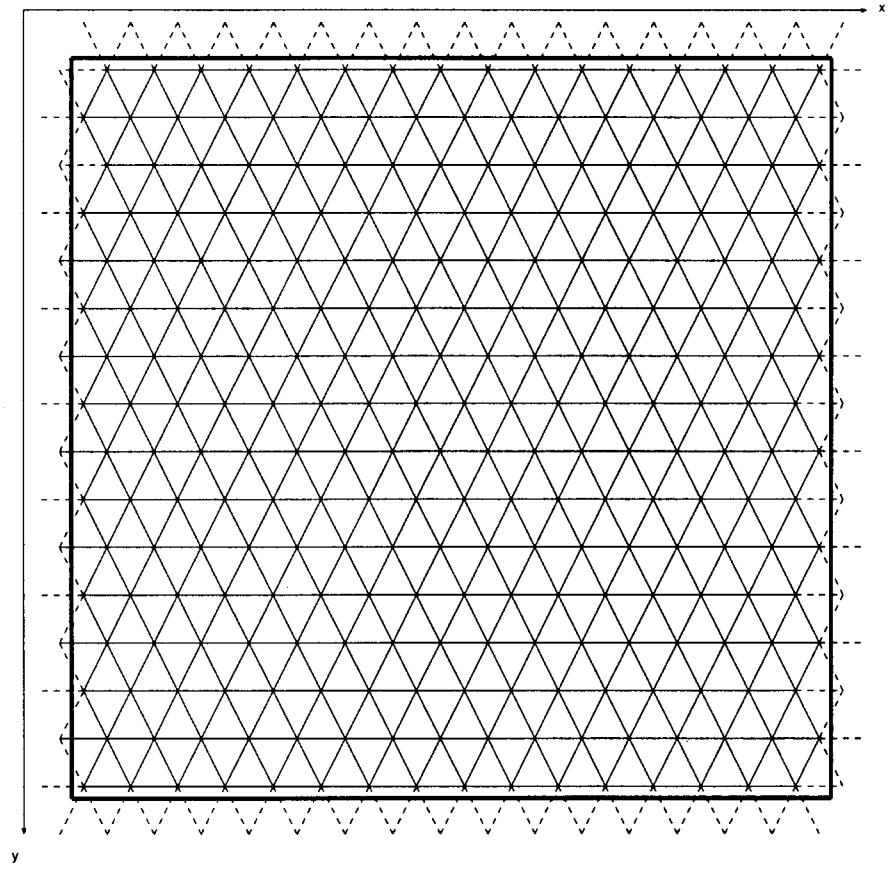

Fig. 5.12 Square wall of the non-parallelized code

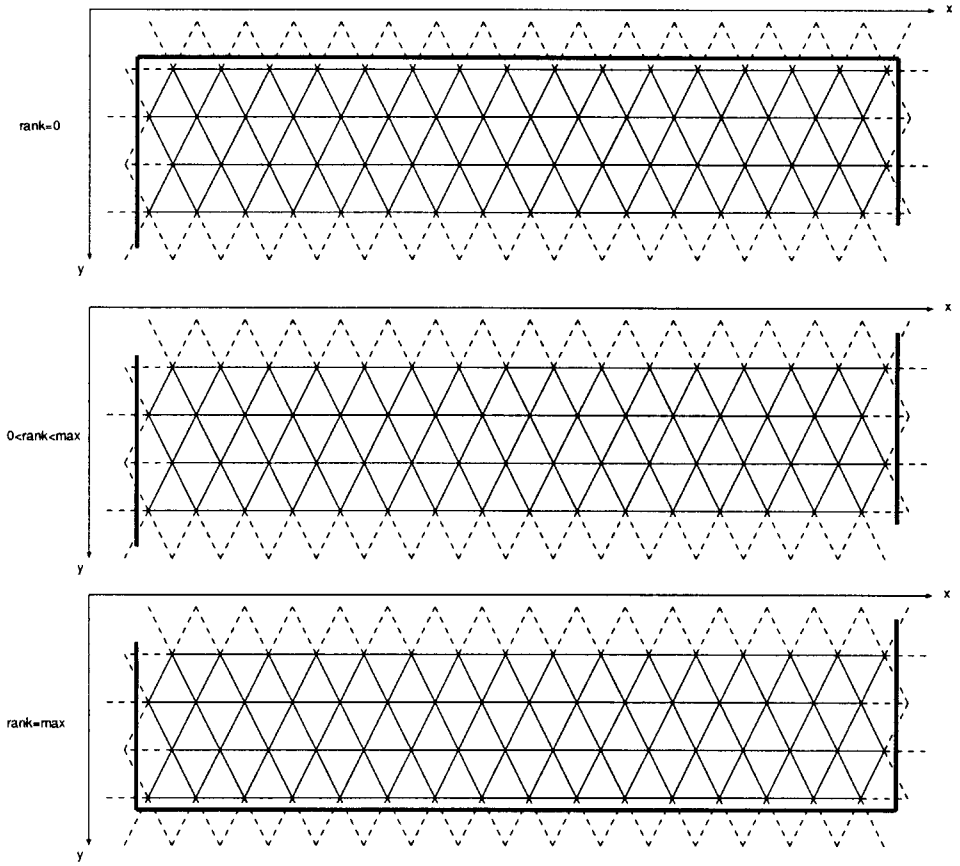

Fig. 5.13 Square wall of the parallelized code

```
 1       subroutine mkwalla(xx,yy,twall)
 2
 3       integer xx,yy,twall(xx*yy,0:6)
 4       integer k,nsite,itmp,jtmp
 5
 6       nsite=xx*yy
 7
 8       call initdat2(xx,yy,twall,0)
 9
10       do 10 k=1,nsite
11          itmp=mod(itox(xx,yy,k),xx)
12          jtmp=mod(itoy(xx,yy,k),2)
13          if(k.eq.1) then    ← 左上の格子点
14             twall(k,2)=-1
15             twall(k,3)=-1
16             twall(k,4)=-1
17          else if(k.ge.2.and.k.le.xx-1) then    ← 上境界部
18             twall(k,2)=-1
19             twall(k,3)=-1
20          else if(k.eq.xx) then    ← 右上の格子点
21             twall(k,1)=-1
22             twall(k,2)=-1
23             twall(k,3)=-1
24             twall(k,6)=-1
25          else if(itmp.eq.1.and.jtmp.eq.0.and.k.le.xx*(yy-1)) then
26             twall(k,3)=-1                    ← 左境界部，横に凸部
27             twall(k,4)=-1
28             twall(k,5)=-1
29          else if(itmp.eq.1.and.jtmp.eq.1.and.k.le.xx*(yy-1)) then
30             twall(k,4)=-1                    ← 左境界部，横に凹部
31          else if(itmp.eq.0.and.jtmp.eq.0.and.k.le.xx*(yy-1)) then
32             twall(k,1)=-1                    ← 右境界部，横に凹部
33          else if(itmp.eq.0.and.jtmp.eq.1.and.k.le.xx*(yy-1)) then
34             twall(k,1)=-1                    ← 右境界部，横に凸部
35             twall(k,2)=-1
36             twall(k,6)=-1
37          else if(k.eq.xx*(yy-1)+1) then    ← 左下の格子点
38             twall(k,3)=-1
39             twall(k,4)=-1
40             twall(k,5)=-1
41             twall(k,6)=-1
42          else if(k.ge.xx*(yy-1)+2.and.k.le.xx*yy-1) then    ← 下境界部
43             twall(k,5)=-1
44             twall(k,6)=-1
45          else if(k.eq.xx*yy) then    ← 右下の格子点
46             twall(k,1)=-1
47             twall(k,5)=-1
48             twall(k,6)=-1
49          endif
50 10    continue
51
52       return
53       end
```

Fig. 5.14 Subroutine mkwalla of the non-parallelized code

```
 1       subroutine mkwalla1
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9       enddo
10       enddo
11       do k1=1,numsite_para
12         iwall(k1)=0
13       enddo
14
15 CCC============================== rank:0
16       if(MYRANK.eq.0) then
17
18 CCC--- Top  (ランク０：上境界部)
19         do k1=ixs*iwidth+1,ixs*iwidth+ixs
20           if(k1.eq.ixs*iwidth+1) then
21             la(k1,4)=-1
22           endif
23           if(k1.eq.ixs*iwidth+ixs) then
24             la(k1,1)=-1
25             la(k1,6)=-1
26           endif
27           la(k1,2)=-1
28           la(k1,3)=-1
29         enddo
30
31 CCC--- Right  (ランク０：右境界部)
32         num_iys=0
33         do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
34           num_iys=num_iys+1
35           if(mod(num_iys,2).ne.0) then
36             la(k1,1)=-1
37             la(k1,2)=-1
38             la(k1,6)=-1
39           else
40             la(k1,1)=-1
41           endif
42         enddo
```

Fig. 5.15　Subroutine mkwalla1 of the parallelized code (1/4)

```
43
44 CCC--- Left （ランク 0 ：左境界部）
45          num_iys=0
46          do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
47            num_iys=num_iys+1
48            if(mod(num_iys,2).ne.0) then
49              la(k1,4)=-1
50            else
51              la(k1,3)=-1
52              la(k1,4)=-1
53              la(k1,5)=-1
54            endif
55          enddo
56
57        endif
58
59 CCC============================ rank:NPROCS-1
60        if(MYRANK.eq.NPROCS-1) then
61
62 CCC--- Right （ランク max：右境界部）
63          num_iys=0
64          do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
65            num_iys=num_iys+1
66            if(mod(num_iys,2).ne.0) then
67              la(k1,1)=-1
68              la(k1,2)=-1
69              la(k1,6)=-1
70            else
71              la(k1,1)=-1
72            endif
73          enddo
74
75 CCC--- Left （ランク max：左境界部）
76          num_iys=0
77          do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
78            num_iys=num_iys+1
79            if(mod(num_iys,2).ne.0) then
80              la(k1,4)=-1
81            else
82              la(k1,3)=-1
83              la(k1,5)=-1
84              la(k1,4)=-1
85            endif
86          enddo
87
```

Fig. 5.15 Subroutine mkwalla1 of the parallelized code (2/4)

```
88 CCC--- Bottom （ランク max：下境界部）
89          do k1=numsite_para-ixs*(iwidth+1)+1,numsite_para-ixs*iwidth
90            if(k1.eq.numsite_para-ixs*(iwidth+1)+1) then
91              la(k1,3)=-1
92              la(k1,4)=-1
93            endif
94            if(k1.eq.numsite_para-ixs*iwidth) then
95              la(k1,1)=-1
96            endif
97            la(k1,5)=-1
98            la(k1,6)=-1
99          enddo
100
101       endif
102
103 CCC============================== rank:1 - NPROCS-2
104       if(MYRANK.gt.0.and.MYRANK.lt.NPROCS-1) then
105
106 CCC--- Right （0＜ランク＜max：右境界部）
107         num_iys=0
108         do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
109           num_iys=num_iys+1
110           if(mod(num_iys,2).ne.0) then
111             la(k1,1)=-1
112             la(k1,2)=-1
113             la(k1,6)=-1
114           else
115             la(k1,1)=-1
116           endif
117         enddo
118
119 CCC--- Left （0＜ランク＜max：左境界部）
120         num_iys=0
121         do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
122           num_iys=num_iys+1
123           if(mod(num_iys,2).ne.0) then
124             la(k1,4)=-1
125           else
126             la(k1,4)=-1
127             la(k1,3)=-1
128             la(k1,5)=-1
129           endif
130         enddo
```

Fig. 5.15 Subroutine mkwalla1 of the parallelized code (3/4)

```
131
132        endif
133
134        do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
135          do k2=istart,nsite
136            ic(k2)=-la(k1,k2)
137          enddo
138          iwall(k1)=irehash(ic,istart,nsite)
139     40 continue
140
141        return
142        end
```

Fig. 5.15 Subroutine mkwalla1 of the parallelized code (4/4)

```
 1       subroutine mkwalla2
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9       enddo
10       enddo
11       do k1=1,numsite_para
12         iwall(k1)=0
13       enddo
14
15 CCC============================== rank:NPROCS-1
16       if(MYRANK.eq.NPROCS-1) then
17
18 CCC--- Right （ランク max：右境界部）
19         num_iys=0
20         do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
21           num_iys=num_iys+1
22           if(mod(num_iys,2).ne.0) then
23             la(k1,1)=-1
24           else
25             la(k1,1)=-1
26             la(k1,2)=-1
27             la(k1,6)=-1
28           endif
29         enddo
30
31 CCC--- Left （ランク max：左境界部）
32         num_iys=0
33         do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
34           num_iys=num_iys+1
35           if(mod(num_iys,2).ne.0) then
36             la(k1,3)=-1
37             la(k1,4)=-1
38             la(k1,5)=-1
39           else
40             la(k1,4)=-1
41           endif
42         enddo
43
```

Fig. 5.16 Subroutine mkwalla2 of the parallelized code (1/3)

```
44 CCC--- Bottom （ランク max：下境界部）
45          do k1=numsite_para-ixs*(iwidth+1)+1,numsite_para-ixs*iwidth
46            if(k1.eq.numsite_para-ixs*(iwidth+1)+1) then
47              la(k1,3)=-1
48              la(k1,4)=-1
49            endif
50            if(k1.eq.numsite_para-ixs*iwidth) then
51              la(k1,1)=-1
52            endif
53            la(k1,5)=-1
54            la(k1,6)=-1
55          enddo
56
57        endif
58
59 CCC============================== rank:1 - NPROCS-2
60        if(MYRANK.gt.0.and.MYRANK.lt.NPROCS-1) then
61
62 CCC--- Right （0＜ランク＜max：右境界部）
63          num_iys=0
64          do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
65            num_iys=num_iys+1
66            if(mod(num_iys,2).ne.0) then
67              la(k1,1)=-1
68            else
69              la(k1,1)=-1
70              la(k1,2)=-1
71              la(k1,6)=-1
72            endif
73          enddo
74
75 CCC--- Left （0＜ランク＜max：左境界部）
76          num_iys=0
77          do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
78            num_iys=num_iys+1
79            if(mod(num_iys,2).ne.0) then
80              la(k1,3)=-1
81              la(k1,4)=-1
82              la(k1,5)=-1
83            else
84              la(k1,4)=-1
85            endif
86          enddo
87
88        endif
```

Fig. 5.16 Subroutine mkwalla2 of the parallelized code (2/3)

```
89
90        do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
91          do k2=istart,nsite
92            ic(k2)=-la(k1,k2)
93          enddo
94          iwall(k1)=irehash(ic,istart,nsite)
95     40 continue
96
97        return
98        end
```

Fig. 5.16 Subroutine `mkwalla2` of the parallelized code (3/3)

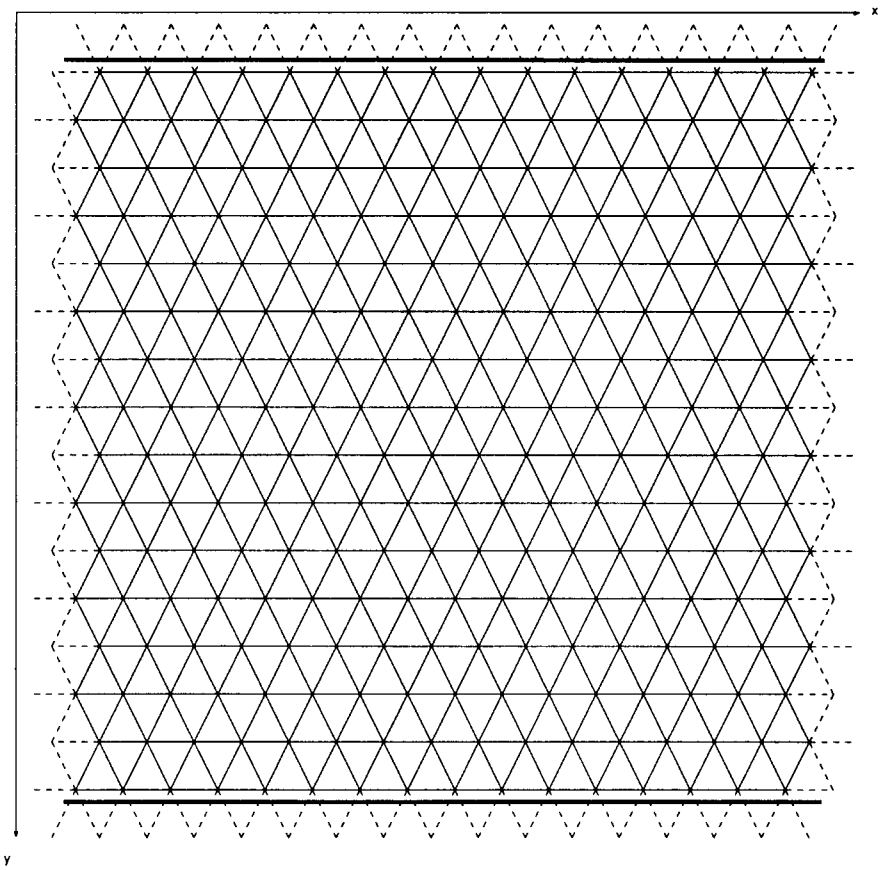

Fig. 5.17 Horizontal parallel wall of the non-parallelized code

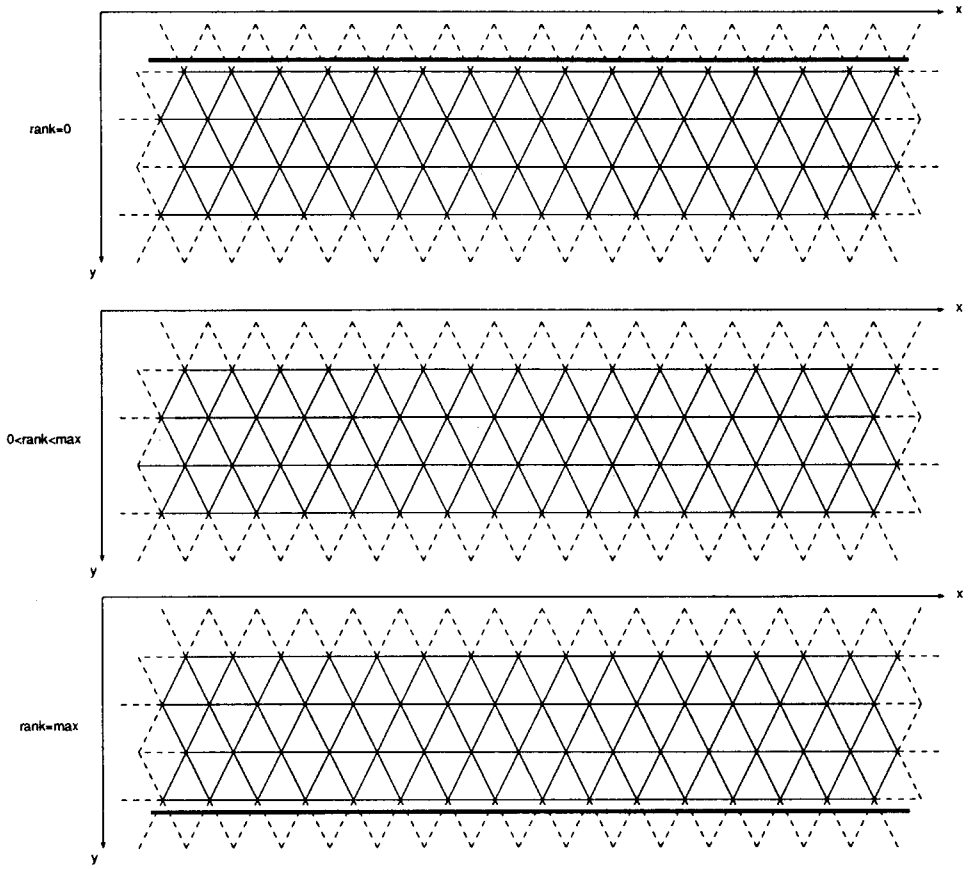

Fig. 5.18 Horizontal parallel wall of the parallelized code

```
 1        subroutine mkwallb(xx,yy,twall)
 2
 3        integer xx,yy,twall(1:xx*yy,0:6)
 4        integer k,nsite
 5
 6        nsite=xx*yy
 7
 8        call initdat2(xx,yy,twall,0)
 9
10        do 10 k=1,nsite
11           if(k.ge.1.and.k.le.xx) then  ← 上境界部
12              twall(k,2)=-1
13              twall(k,3)=-1
14           else if(k.ge.xx*(yy-1)+1.and.k.le.xx*yy) then   ← 下境界部
15              twall(k,5)=-1
16              twall(k,6)=-1
17           endif
18 10     continue
19
20        return
21        end
```

Fig. 5.19 Subroutine mkwallb of the non-parallelized code

```
 1       subroutine mkwallb
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9       enddo
10       enddo
11       do k1=1,numsite_para
12         iwall(k1)=0
13       enddo
14
15 CCC============================== rank:0
16       if(MYRANK.eq.0) then
17
18 CCC--- Top （ランク０：上境界部）
19         do k1=ixs*iwidth+1,ixs*iwidth+ixs
20           la(k1,2)=-1
21           la(k1,3)=-1
22         enddo
23
24       endif
25
26 CCC============================== rank:NPROCS-1
27       if(MYRANK.eq.NPROCS-1) then
28
29 CCC--- Bottom （ランク max：下境界部）
30         do k1=numsite_para-ixs*(iwidth+1)+1,numsite_para-ixs*iwidth
31           la(k1,5)=-1
32           la(k1,6)=-1
33         enddo
34
35       endif
36
37       do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
38         do k2=istart,nsite
39           ic(k2)=-la(k1,k2)
40         enddo
41         iwall(k1)=irehash(ic,istart,nsite)
42    40 continue
43
44       return
45       end
```

Fig. 5.20 Subroutine mkwallb of the parallelized code

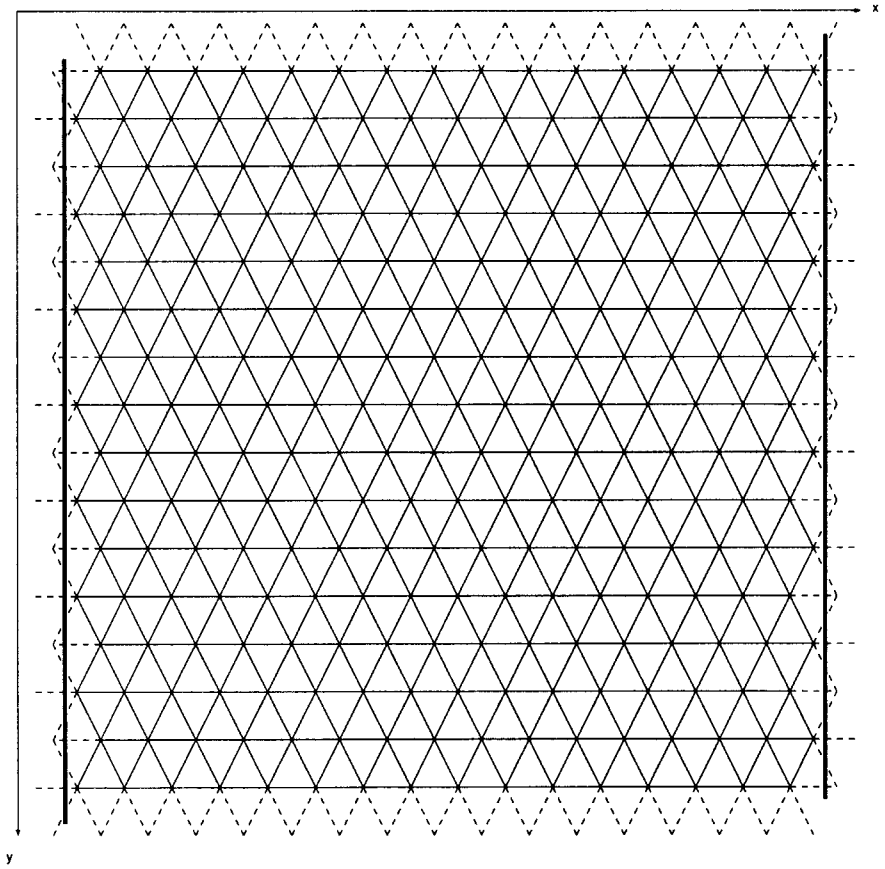

Fig. 5.21 Vertical parallel wall of the non-parallelized code

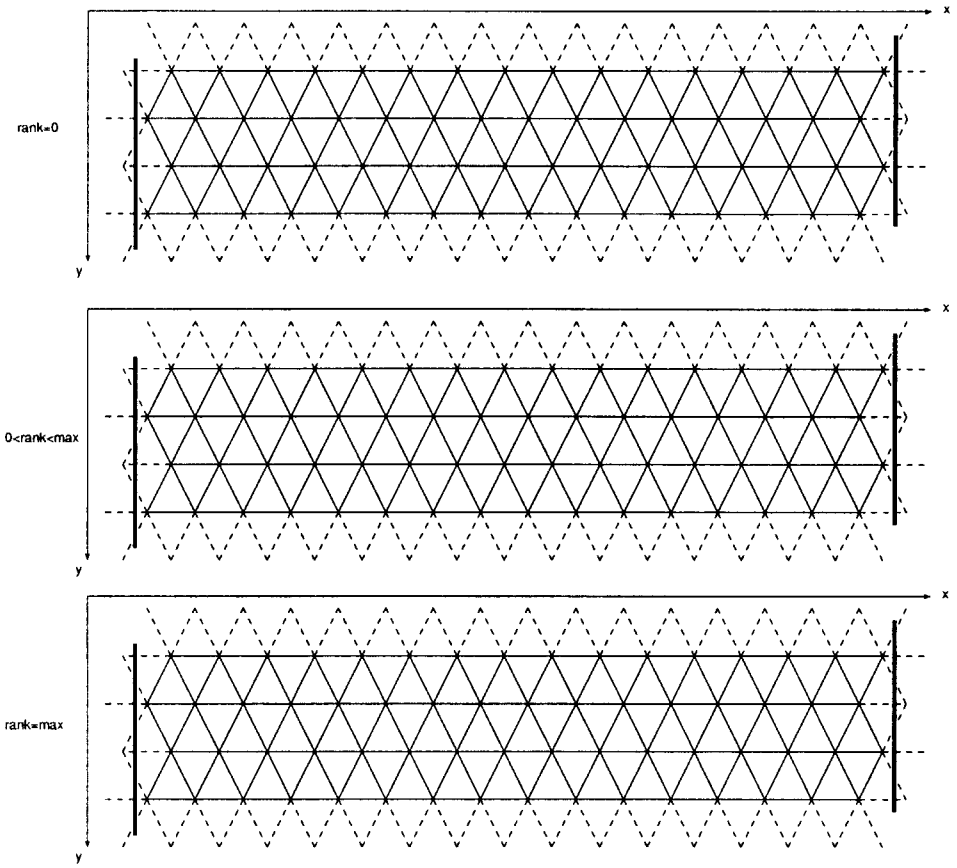

Fig. 5.22 Vertical parallel wall of the parallelized code

```
 1        subroutine mkwallc(xx,yy,twall)
 2
 3        integer xx,yy,twall(xx*yy,0:6)
 4        integer k,nsite,itmp,jtmp,ix
 5
 6        nsite=xx*yy
 7
 8        call initdat2(xx,yy,twall,0)
 9
10        do 10 k=1,nsite
11           ix=itox(xx,yy,k)
12           jtmp=mod(itoy(xx,yy,k),2)
13           if(ix.eq.1) then          ← 左境界部
14              if(jtmp.eq.1) then        ← 横に凹部
15                 twall(k,4)=-1
16              endif
17              if(jtmp.eq.0) then        ← 横に凸部
18                 twall(k,3)=-1
19                 twall(k,4)=-1
20                 twall(k,5)=-1
21              endif
22           endif
23           if(ix.eq.xx) then         ← 左境界部
24              if(jtmp.eq.1) then        ← 横に凸部
25                 twall(k,1)=-1
26                 twall(k,2)=-1
27                 twall(k,6)=-1
28              endif
29              if(jtmp.eq.0) then        ← 横に凹部
30                 twall(k,1)=-1
31              endif
32           endif
33 10     continue
34
35        return
36        end
```

Fig. 5.23 Subroutine mkwallc of the non-parallelized code

```
 1       subroutine mkwallc1
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9       enddo
10       enddo
11       do k2=1,numsite_para
12         iwall(k2)=0
13       enddo
14
15 CCC--- Right （全ランク：右境界部）
16       num_iys=0
17       do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
18         num_iys=num_iys+1
19         if(mod(num_iys,2).ne.0) then
20           la(k1,1)=-1
21           la(k1,2)=-1
22           la(k1,6)=-1
23         else
24           la(k1,1)=-1
25         endif
26       enddo
27
28 CCC--- Left （全ランク：左境界部）
29       num_iys=0
30       do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
31         num_iys=num_iys+1
32         if(mod(num_iys,2).ne.0) then
33           la(k1,4)=-1
34         else
35           la(k1,3)=-1
36           la(k1,4)=-1
37           la(k1,5)=-1
38         endif
39       enddo
40
41       do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
42         do k2=istart,nsite
43           ic(k2)=-la(k1,k2)
44         enddo
45         iwall(k1)=irehash(ic,istart,nsite)
46    40 continue
47
48       return
49       end
```

Fig. 5.24　Subroutine mkwallc1 of the parallelized code

```
 1       subroutine mkwallc2
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9       enddo
10       enddo
11       do k1=1,numsite_para
12         iwall(k1)=0
13       enddo
14
15 CCC--- Right （全ランク：右境界部）
16       num_iys=0
17       do k1=ixs*iwidth+ixs,numsite_para-ixs*iwidth,ixs
18         num_iys=num_iys+1
19         if(mod(num_iys,2).ne.0) then
20           la(k1,1)=-1
21         else
22           la(k1,1)=-1
23           la(k1,2)=-1
24           la(k1,6)=-1
25         endif
26       enddo
27
28 CCC--- Left （全ランク：左境界部）
29       num_iys=0
30       do k1=ixs*iwidth+1,numsite_para-ixs*(iwidth+1)+1,ixs
31         num_iys=num_iys+1
32         if(mod(num_iys,2).ne.0) then
33           la(k1,3)=-1
34           la(k1,4)=-1
35           la(k1,5)=-1
36         else
37           la(k1,4)=-1
38         endif
39       enddo
40
41       do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
42         do k2=istart,nsite
43           ic(k2)=-la(k1,k2)
44         enddo
45         iwall(k1)=irehash(ic,istart,nsite)
46    40 continue
47
48       return
49       end
```

Fig. 5.25　Subroutine mkwallc2 of the parallelized code

```
 1      function irehash(ic,istart,nsite)
 2
 3      integer*4 ic(istart:nsite)
 4
 5      if(istart.eq.0) then
 6        irehash=64*ic(0)+32*ic(1)+16*ic(2)+8*ic(3)+4*ic(4)+2*ic(5)+ic(6)
 7      elseif(istart.eq.1) then
 8        irehash=32*ic(1)+16*ic(2)+8*ic(3)+4*ic(4)+2*ic(5)+ic(6)
 9      endif
10
11      return
12      end
```

Fig. 5.26 User's function **irehash** of the parallelized code

```
 1       if(phase.eq.1) then
 2 c        /***** distributing particles on the lattice *************/
 3          call sist(xs,ys,state,wall,den,opt,model,ierr)
 4          if(ierr.eq.-1) then
 5             write(6,*) "Error in sist"
 6          endif
 7
 8 c        /**** count Physical Quantity ***************************/
 9           call countPQ(xs,ys,state,numpar,momx,momy,p_num,model)
10           write(6,4) numpar,momx,momy
11           write(6,*) (p_num(i),i=start,6)
12
13 c        /***** output initial state ******************************/
14           n=10
15           n=n+1
```

Fig. 5.27 Particle distribution part in the main routine of the non-parallelized code (phase1)

```
 1       if(iphase.eq.1) then
 2
 3       iseed=MYRANK*1
 4
 5 C*************************************************
 6 C
 7 C     CALL SIST (make bstate)
 8 C
 9 C*************************************************
10       call sist
11           if(ierr.eq.-1) then
12             write(6,*) 'Error in sist on rank:',MYRANK
13           endif
14
15       call countPQ(bstate)
16       call countPQ_sum
17       if(MYRANK.eq.0) then
18         write(6,4) snumpar,smomx,smomy
19         write(6,*) (sp_num(i),i=istart,nsite)
20       endif
21
22       n=10+MYRANK
```

Fig. 5.28 Particle distribution part in the main routine of the parallelized code (phase1)

```
 1       subroutine sist(xx,yy,state,twall,den,opt2,model,ierr)
 2
 3       integer xx,yy,ierr
 4       integer model
 5       integer itmp1,start,itmp
 6       integer state(xx*yy,0:6),twall(xx*yy,0:6)
 7       integer i,j,si,sj,k,s,nsite,ix,iy
 8       real total,den
 9       integer opt2
10
11       data xxx/0/
12
13       nsite=xx*yy
14
15       call initdat2(xx,yy,state,0)
16
17       if(model.eq.1) then
18          start=1
19          total=6.0
20       else if(model.ne.1) then
21          start=0
22          total=7.0
23       endif
24
25
26       write(6,*) "#paticle ",int(real(xx*yy)*den*total)
27      +," rate ",opt2
28
29       ii=0
30       sj=0
31       si=0
32 64    continue
33       if(ii.ge.int(real(xx*yy)*den*total)) then
34          goto 63
35       endif
36
```

Fig. 5.29 Subroutine sist of the non-parallelized code (1/2)

```
      do 62 j=1,yy,yy/opt2
      do 62 i=1,xx,xx/opt2
         k=xx*(j+sj-1)+i+si
         if(ii.ge.int(real(xx*yy)*den*total)) then
            goto 63
         endif
         itmp1=twall(k,0)+twall(k,1)+twall(k,2)+twall(k,3)
     +        +twall(k,4)+twall(k,5)+twall(k,6)
         do 50 s=start,6,1
            if(state(k,s).eq.0.and.itmp1.eq.0) then
               state(k,s)=1
               ii=ii+1
            endif
50 50       continue
         if(k.gt.xx*yy) then
            write(6,*) "Error",k,"(",i,",",j,")",si,sj,ii
         endif
62 62    continue

      si=si+1
      if(si.eq.xx/opt2) then
         sj=sj+1
         si=0
      endif
      goto 64

63 63    continue


66 c     /***** checking state *****/
      ierr=0
      do 3000 k=1,nsite
         ix=itox(xx,yy,k)
         iy=itoy(xx,yy,k)
         do 3002 i=0,6
            if(state(k,i)*twall(k,i).ne.0) then
               write(6,*) k,i,ix,iy,state(k,i)
               ierr=-1
            endif
76 3002     continue
77 3000  continue

      return
      end
```

Fig. 5.29 Subroutine sist of the non-parallelized code (2/2)

```
 1       subroutine sist
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       real*4 total
 7
 8       do k1=istart,nsite
 9       do k2=1,numsite_para
10         la(k2,k1)=0
11       enddo
12       enddo
13       do k1=1,numsite_para
14         bstate(k1)=0
15       enddo
16
17       if(model.ne.1) then
18         total=7.0
19       else
20         total=6.0
21       endif
22
23       imaxp=int(real(numsite)*den*total)
24
25       write(6,*) "#particle ",imaxp," rate ",iopt,' on rank:',MYRANK
26
27       call site_particle_check(ixs,iys,iyss,iopt,iwidth,ix_part,isetx,
28      &iy_part,isety)
29
30       ii=0
31       js=0
32       is=0
33    64 continue
34       if(ii.ge.imaxp) then
35         go to 63
36       endif
37
```

Fig. 5.30 Subroutine sist of the parallelized code (1/3)

```
38       do 60 j=iwidth+1,iwidth+(iy_part*isety),(iy_part*isety)/isety
39       do 61 i=1,ix_part*isetx,(ix_part*isetx)/isetx
40         k=ixs*(j+js-1)+i+is
41         if(ii.ge.imaxp) then
42           go to 63
43         endif
44         if(istart.eq.0) then
45         itmp1=tabw(iwall(k),0)+tabw(iwall(k),1)
46      &        +tabw(iwall(k),2)+tabw(iwall(k),3)
47      &        +tabw(iwall(k),4)+tabw(iwall(k),5)
48      &        +tabw(iwall(k),6)
49         else
50         itmp1=tabw(iwall(k),1)
51      &        +tabw(iwall(k),2)+tabw(iwall(k),3)
52      &        +tabw(iwall(k),4)+tabw(iwall(k),5)
53      &        +tabw(iwall(k),6)
54         endif
55         do 50 iis=istart,nsite,1
56           if(la(k,iis).eq.0.and.itmp1.eq.0) then
57             la(k,iis)=1
58             ii=ii+1
59           elseif(la(k,iis).eq.0.and.itmp1.ne.0) then
60             ii=ii+1
61           endif
62    50   continue
63         if(k.lt.ixs*iwidth+1.or.k.gt.ixs*(iys+iwidth)) then
64           write(6,*) 'Error',k,'(',i,',',j,')',is,js,ii,' on rank:',
65       &MYRANK
66         endif
67    61 continue
68    60 continue
69
70       is=is+1
71       if(is.eq.ixs/iopt) then
72         js=js+1
73         is=0
74       endif
75       go to 64
76
77    63 continue
```

Fig. 5.30 Subroutine sist of the parallelized code (2/3)

```
78
79        call rehash(la,bstate)
80
81        if(idebug.eq.1) then
82        ierr=0
83        do 3100 k1=istart,nsite
84        do 3000 k2=ixs*iwidth+1,ixs*iwidth+numsite
85          if(tab(bstate(k2),k1)*tabw(iwall(k2),k1).ne.0) then
86            ix=itox(ixs,iys,k2)
87            iy=itoy(ixs,iys,k2)
88            write(6,*) k2,k1,ix,iy,tab(bstate(k2),k1),' on rank:',MYRANK
89            ierr=-1
90          endif
91  3000  continue
92  3100  continue
93        endif
94
95        return
96        end
```

Fig. 5.30 Subroutine sist of the parallelized code (3/3)

```
 1       subroutine site_particle_check(ixs,iys,iyss,iopt,iwidth,ix_part,
 2      &isetx,iy_part,isety)
 3
 4       integer*4 ixs,iys,iyss,iopt,iwidth,ix_part,isetx,iy_part,isety
 5
 6 *include INC_MPI
 7
 8       integer*4 iy_part2,iopt_part
 9
10       ix_part=ixs/iopt
11       iy_part2=iys
12
13       if(mod(iopt,ipe).eq.0) then
14         iopt_part=iopt/ipe
15       else
16         write(6,*) 'PARALLEL mod(iopt,ipe) error in site_particle_ch
17      &eck on rank:',MYRANK
18         stop
19       endif
20
21       iy_part=iy_part2/iopt_part
22
23       if(mod(ixs,iopt).eq.0.and.mod(iy_part2,iopt_part).eq.0) then
24         isetx=iopt
25         isety=iopt_part
26       else
27         write(6,*) 'PARALLEL mod(ixs,iopt) or mod(iy_part2,iopt_part)
28      &error in site_particle_check on rank:',MYRANK
29         stop
30       endif
31
32       return
33       end
```

Fig. 5.31 Subroutine site_particle_check of the parallelized code

```
 1        subroutine countPQ(xx,yy,state,nump,momx,momy,num,model)
 2        integer xx,yy
 3        integer nump,momx,momy,model
 4        integer state(1:xx*yy,0:6)
 5        integer num(0:6)
 6
 7        integer k,i,nsite,start
 8
 9        nsite=xx*yy
10
11        if(model.eq.1) then
12           start=1
13        else if(model.ne.1) then
14           start=0
15        endif
16
17        do 10 i=0,6
18           num(i)=0
19  10    continue
20
21        do 30 i=0,6
22           do 30 k=1,nsite
23           if(state(k,i).eq.1) then
24              num(i)=num(i)+1
25           endif
26  30    continue
27
28        nump=0
29        momx=0
30        momy=0
31        do 40 k=1,nsite
32           momx=momx+2*state(k,1)+state(k,2)+state(k,6)
33       +             -2*state(k,4)-state(k,3)-state(k,5)
34           momy=momy+state(k,2)+state(k,3)-state(k,5)-state(k,6)
35           nump=nump+state(k,1)+state(k,2)+state(k,3)+state(k,4)
36       +             +state(k,5)+state(k,6)+state(k,0)
37  40    continue
38
39        return
40        end
```

Fig. 5.32 Subroutine countPQ of the non-parallelized code

```
      subroutine countPQ(istate)

*include INC_MPI
*include INC_DATA

      integer*4 istate(0:numsite_para)

      do 10 k1=istart,nsite
        p_num(k1)=0
   10 continue

      do 30 k1=istart,nsite
        do 31 k2=ixs*iwidth+1,ixs*iwidth+numsite
          if(tab(istate(k2),k1).eq.1) then
            p_num(k1)=p_num(k1)+1
          endif
   31   continue
   30 continue

      numpar=0
      momx=0
      momy=0
      do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
      momx=momx+2*tab(istate(k1),1)+tab(istate(k1),2)
     &         +tab(istate(k1),6)
     &         -2*tab(istate(k1),4)-tab(istate(k1),3)
     &         -tab(istate(k1),5)
      momy=momy+tab(istate(k1),2)+tab(istate(k1),3)-tab(istate(k1),5)
     &         -tab(istate(k1),6)
      if(istart.eq.0) then
      numpar=numpar+tab(istate(k1),1)+tab(istate(k1),2)
     &             +tab(istate(k1),3)+tab(istate(k1),4)
     &             +tab(istate(k1),5)+tab(istate(k1),6)
     &             +tab(istate(k1),0)
      else
      numpar=numpar+tab(istate(k1),1)+tab(istate(k1),2)
     &             +tab(istate(k1),3)+tab(istate(k1),4)
     &             +tab(istate(k1),5)+tab(istate(k1),6)
      endif
   40 continue

      return
      end
```

Fig. 5.33 Subroutine countPQ of the parallelized code

```
 1       subroutine countPQ_sum
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       snumpar=0
 7       smomx=0
 8       smomy=0
 9       do k1=istart,nsite
10         sp_num(k1)=0
11       enddo
12
13       call mpi_barrier(mpi_comm_world,ierr)
14
15       call mpi_reduce(numpar,  snumpar,  1,
16      &mpi_integer,mpi_sum,0,mpi_comm_world,ierr)
17
18       call mpi_reduce(momx,    smomx,    1,
19      &mpi_integer,mpi_sum,0,mpi_comm_world,ierr)
20
21       call mpi_reduce(momy,    smomy,    1,
22      &mpi_integer,mpi_sum,0,mpi_comm_world,ierr)
23
24       call mpi_reduce(p_num(istart),sp_num(istart),(nsite-istart+1),
25      &mpi_integer,mpi_sum,0,mpi_comm_world,ierr)
26
27       call mpi_barrier(mpi_comm_world,ierr)
28
29       return
30       end
```

Fig. 5.34 Subroutine countPQ_sum of the parallelized code

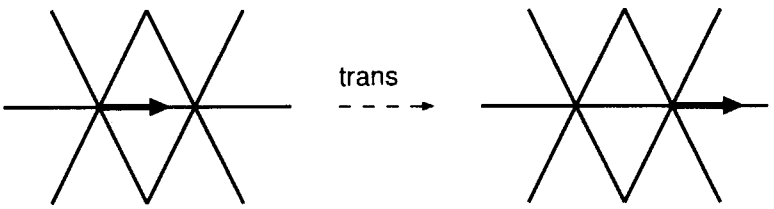

Fig. 5.35 Particle propagation

```
1
2               call trans(xs,ys,site,state,a_state,label,a_label,ierr)
3
4               if(debug.eq.1) then
5               call countPQ(xs,ys,a_state,numpar,momx,momy,p_num,model)
6               write(6,*) "trans. ",numpar," (",momx,",",momy,") "
7               write(6,*) (p_num(i),i=start,6)
8               endif
```

Fig. 5.36 Particle propagation part in the main routine of the non-parallelized code (phase1)

```
 1 C*********************
 2 C  data state transfer
 3 C*********************
 4       call mpi_trans(bstate,ixs*iwidth)
 5
 6 C*****************************************
 7 C
 8 C     CALL TRANS (bstate -> astate)
 9 C
10 C*****************************************
11       call trans
12           if(ierr.eq.-1) then
13             write(6,*) 'Error in trans on rank:',MYRANK
14           endif
15
16           if(idebug.eq.1) then
17             call countPQ(astate)
18             call countPQ_sum
19             if(MYRANK.eq.0) then
20               write(6,*) 'trans. ',snumpar,' (',smomx,',',smomy,') '
21               write(6,*) (sp_num(i),i=istart,nsite)
22             endif
23           endif
```

Fig. 5.37 Particle propagation part in the main routine of the parallelized code (phase1)

```
 1       subroutine mpi_trans(transfer,ioverlap)
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 transfer(0:numsite_para)
 7       integer*4 ioverlap
 8
 9 CCC--- Send_rank
10       isup=MYRANK-1
11       isdown=MYRANK+1
12       if(isup.lt.0) then
13         isup=isup+NPROCS
14       endif
15       if(isdown.gt.NPROCS-1) then
16         isdown=isdown-NPROCS
17       endif
18
19 CCC--- Receive rank
20       irup=MYRANK-1
21       irdown=MYRANK+1
22       if(irup.lt.0) then
23         irup=irup+NPROCS
24       endif
25       if(irdown.gt.NPROCS-1) then
26         irdown=irdown-NPROCS
27       endif
28
29 CCC--- transfer of data from (Top of down_rank) to (Bottom of up_rank)
30       do k1=1,2
31       do k2=1,ioverlap
32         itrans_array(k2,k1)=0
33       enddo
34       enddo
35       num_data=0
36       do k1=ixs*iwidth+1,ixs*iwidth+ixs*iwidth
37         num_data=num_data+1
38         itrans_array(num_data,1)=transfer(k1)
39       enddo
40
41       call mpi_barrier(mpi_comm_world,ierr)
42       call mpi_sendrecv(itrans_array(1,1),num_data,mpi_integer,
43      &                  isup,              1,
44      &                  itrans_array(1,2),num_data,mpi_integer,
45      &                  irdown,            1,
46      &                  mpi_comm_world,status,ierr)
47       call mpi_barrier(mpi_comm_world,ierr)
```

Fig. 5.38 Subroutine mpi_trans of the parallelized code (1/2)

```
48
49       num_data=0
50       do k1=(ixs*(iys+iwidth*2))-(ixs*iwidth)+1,(ixs*(iys+iwidth*2))
51      &-(ixs*iwidth)+ixs*iwidth
52         num_data=num_data+1
53         transfer(k1)=itrans_array(num_data,2)
54       enddo
55
56 CCC--- transfer of data from (Bottom of up_rank) to (Top of down_rank)
57       do k1=1,2
58       do k2=1,ioverlap
59         itrans_array(k2,k1)=0
60       enddo
61       enddo
62       num_data=0
63       do k1=(ixs*(iys+iwidth*2))-2*(ixs*iwidth)+1,(ixs*(iys+iwidth*2))
64      &-2*(ixs*iwidth)+ixs*iwidth
65         num_data=num_data+1
66         itrans_array(num_data,1)=transfer(k1)
67       enddo
68
69       call mpi_barrier(mpi_comm_world,ierr)
70       call mpi_sendrecv(itrans_array(1,1),num_data,mpi_integer,
71      &                  isdown,           1,
72      &                  itrans_array(1,2),num_data,mpi_integer,
73      &                  irup,             1,
74      &                  mpi_comm_world,status,ierr)
75       call mpi_barrier(mpi_comm_world,ierr)
76
77       num_data=0
78       do k1=1,ixs*iwidth
79         num_data=num_data+1
80         transfer(k1)=itrans_array(num_data,2)
81       enddo
82
83       return
84       end
```

Fig. 5.38 Subroutine mpi_trans of the parallelized code (2/2)

```
 1        subroutine trans(xx,yy,tsite,before,after,blabel,alabel,ierr)
 2
 3        integer xx,yy,ierr
 4        integer tsite(1:xx*yy,0:6)
 5        integer before(1:xx*yy,0:6),after(1:xx*yy,0:6)
 6        integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
 7
 8        integer i,j,k,nsite
 9        integer ib_tmp
10
11        nsite=xx*yy
12        ierr=0
13
14        call initdat2(xx,yy,after,0)
15        call initdat2(xx,yy,alabel,-2)
16
17        do 30 i=1,6
18        do 30 k=1,nsite
19              if(before(k,i).eq.1) then
20                 ib_tmp=tsite(k,i)
21                 after(ib_tmp,i)=before(k,i)
22                 alabel(ib_tmp,i)=blabel(k,i)
23              endif
24 30     continue
25
26        do 15 j=1,nsite
27           after(j,0)=before(j,0)
28           alabel(j,0)=blabel(j,0)
29 15     continue
30
31        do 85 i=0,6
32        do 85 j=1,nsite
33           if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
34              write(6,*) "After translation"
35              write(6,521) after(j,i),j,i
36              ierr=-1
37           endif
38 85     continue
39
40 521    format(1h ,"Error state= ",i10," at ",i6," of site",i4)
41
42        return
43        end
```

Fig. 5.39 Subroutine trans of the non-parallelized code

```
 1       subroutine trans
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9       enddo
10       enddo
11       do k1=1,numsite_para
12         astate(k1)=0
13       enddo
14
15       do 30 i=1,6
16         io=i+3
17         if(io.gt.6) then
18           io=io-6
19         endif
20         do 20 k1=ixs*iwidth+1,ixs*iwidth+numsite
21           if(tab(bstate(isite(k1,io)),i).eq.1) then
22             la(k1,i)=1
23           endif
24    20   continue
25    30 continue
26
27       if(istart.eq.0) then
28       do 10 k1=ixs*iwidth+1,ixs*iwidth+numsite
29         la(k1,0)=tab(bstate(k1),0)
30    10 continue
31       endif
32
33       call rehash(la,astate)
34
35       if(idebug.eq.1) then
36       ierr=0
37       do 85 k1=istart,nsite
38       do 86 k2=ixs*iwidth+1,ixs*iwidth+numsite
39         if( (tab(astate(k2),k1).lt.0).or.(tab(astate(k2),k1).gt.1) )
40      &then
41           write(6,*) 'After translation on rank:',MYRANK
42           write(6,521) tab(astate(k2),k1),k2,k1
43           ierr=-1
44         endif
45    86 continue
46    85 continue
47       endif
48
49   521 format(1h ,'Error state= ',i10,' at',i6,' of site',i4)
50
51       return
52       end
```

Fig. 5.40 Subroutine trans of the parallelized code

```
 1       subroutine rehash(idata,odata)
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 idata(1:numsite_para,istart:nsite)
 7       integer*4 odata(0:numsite_para)
 8
 9       if(istart.eq.0) then
10       do 10 k1=ixs*iwidth+1,ixs*iwidth+numsite
11         odata(k1)=0
12         odata(k1)=64*idata(k1,0)+32*idata(k1,1)+16*idata(k1,2)
13      &          +8*idata(k1,3) +4*idata(k1,4) +2*idata(k1,5)
14      &          +idata(k1,6)
15    10 continue
16       else
17       do 11 k1=ixs*iwidth+1,ixs*iwidth+numsite
18         odata(k1)=0
19         odata(k1)=32*idata(k1,1)+16*idata(k1,2)+8*idata(k1,3)
20      &          +4*idata(k1,4)+2*idata(k1,5)+idata(k1,6)
21    11 continue
22       endif
23
24       return
25       end
```

Fig. 5.41 Subroutine **rehash** of the parallelized code

```
 1               if(optw.ne.0) then
 2                 call chwall(xs,ys,wall,a_state,state,iacum
 3       +         ,a_label,label,ierr)
 4
 5                 if(debug.eq.1) then
 6                 call countPQ(xs,ys,state,numpar,momx,momy,p_num,model)
 7                 write(6,*) "chwall ",numpar," (",momx,",",momy,") "
 8                 write(6,*) (p_num(i),i=start,6)
 9                 endif
10
11                 call copydata(xs,ys,state,a_state)
12                 call copydata(xs,ys,label,a_label)
13               endif
```

Fig. 5.42 Bounce-back wall part in the main routine of the non-parallelized code (phase1)

```
 1        if(ioptw.ne.0) then
 2  C**************************************
 3  C
 4  C     CALL CHWALL (astate -> bstate -> astate)
 5  C
 6  C**************************************
 7           call chwall
 8               if(ierr.eq.-1) then
 9                 write(6,*) 'Error in chwall on rank:',MYRANK
10               endif
11           do k1=1,numsite_para
12             astate(k1)=bstate(k1)
13           enddo
14
15               if(idebug.eq.1) then
16                 call countPQ(astate)
17                 call countPQ_sum
18                 if(MYRANK.eq.0) then
19                   write(6,*) 'chwall ',snumpar,' (',smomx,',',smomy,')
20                   write(6,*) (sp_num(i),i=istart,nsite)
21                 endif
22               endif
23        endif
```

Fig. 5.43 Bounce-back wall part in the main routine of the parallelized code (phase1)

```
 1       subroutine chwall(xx,yy,twall,before,after,acum,blabel,alabel
 2      +,ierr)
 3
 4       integer xx,yy,ierr
 5       integer twall(1:xx*yy,0:6)
 6       integer before(1:xx*yy,0:6),after(1:xx*yy,0:6)
 7       integer acum(1:xx*yy,0:6)
 8       integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
 9
10       integer i,j,k,nsite,io,itmp1,itmp2,ii1,ii2,ii
11
12       nsite=xx*yy
13       ierr=0
14
15       call copydata(xx,yy,before,after)
16       call copydata(xx,yy,blabel,alabel)
17
18 c     /*** normal wall (no slip wall) ***/
19
20       do 30 ii=1,3
21          i=ii
22          io=i+3
23          if(io.gt.6) then
24              io=io-6
25          endif
26          do 40 k=1,nsite
27             itmp1=twall(k,i)*before(k,i)
28             itmp2=twall(k,io)*before(k,io)
29             if(itmp1.eq.-1) then
30                after(k,i)=0
31                after(k,io)=1
32                acum(k,io)=acum(k,io)+1
33                alabel(k,i)=blabel(k,io)
34                alabel(k,io)=blabel(k,i)
35              else if(itmp2.eq.-1) then
36                after(k,i)=1
37                after(k,io)=0
38                acum(k,i)=acum(k,i)+1
39                alabel(k,i)=blabel(k,io)
40                alabel(k,io)=blabel(k,i)
46             endif
47 40        continue
48 30     continue
```

Fig. 5.44 Subroutine chwall of the non-parallelized code (1/2)

```
49
50 c      /* checking */
51        do 86 i=0,6
52        do 86 j=1,nsite
53           if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
54              write(6,*) "After cheking wall"
55              write(6,521) after(j,i),j,i
56              ierr=-1
57           endif
58           itmp1=twall(j,i)*after(j,i)
59           if(itmp1.lt.0) then
60              write(6,*) "After cheking wall"
61              write(6,521) after(j,i),j,i
62              ierr=-1
63           endif
64 86     continue
65
66 521    format(1h ,"Error state= ",i10," at ",i6," of site",i4)
67
68        return
69        end
```

Fig. 5.44 Subroutine chwall of the non-parallelized code (2/2)

```
 1       subroutine chwall
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         la(k2,k1)=0
 9         la(k2,k1)=tab(astate(k2),k1)
10       enddo
11       enddo
12
13       do 30 ii=1,3
14         i=ii
15         io=i+3
16         if(io.gt.6) then
17           io=io-6
18         endif
19         do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite
20           itmp1=tabw(iwall(k1),i)*tab(astate(k1),i)
21           itmp2=tabw(iwall(k1),io)*tab(astate(k1),io)
22           if(itmp1.eq.-1) then
23             la(k1,i)=0
24             la(k1,io)=1
25             iacum(k1,io)=iacum(k1,io)+1
26           elseif(itmp2.eq.-1) then
27             la(k1,i)=1
28             la(k1,io)=0
29             iacum(k1,i)=iacum(k1,i)+1
30           endif
31    40   continue
32    30 continue
33
34       call rehash(la,bstate)
35
```

Fig. 5.45 Subroutine chwall of the parallelized code (1/2)

```
      if(idebug.eq.1) then
      ierr=0
      do 85 k1=istart,nsite
      do 86 k2=ixs*iwidth+1,ixs*iwidth+numsite
        if( (tab(bstate(k2),k1).lt.0).or.(tab(bstate(k2),k1).gt.1) )
     &then
          write(6,*) 'After checking wall-1 on rank:',MYRANK
          write(6,521) tab(bstate(k2),k1),k2,k1
          ierr=-1
        endif
        if( tabw(iwall(k2),k1)*tab(bstate(k2),k1).lt.0 ) then
          write(6,*) 'After checking wall-2 on rank:',MYRANK
          write(6,521) tab(bstate(k2),k1),k2,k1
          ierr=-1
        endif
 86   continue
 85   continue
      endif

521   format(1h ,'Error state = ',i10,' at',i6,' of site',i4)

      return
      end
```

Fig. 5.45 Subroutine chwall of the parallelized code (2/2)

```
            if(model.eq.1) then
               call FHP1(xs,ys,a_state,state,wall,iduma
     +                   ,a_label,label,ierr)

            else if(model.eq.2) then
               call FHP2(xs,ys,a_state,state,wall,iduma
     +                   ,a_label,label,ierr)

            else if(model.eq.3) then
               call FHP3(xs,ys,a_state,state,wall,iduma,idumb
     +                   ,a_label,label,nfcol,ierr)

            endif

            if(debug.eq.1) then
            call countPQ(xs,ys,state,numpar,momx,momy,p_num,model)
            write(6,*) "FHP",model,"  ",numpar,"(",momx,",",momy,") "
            write(6,*)(p_num(i),i=start,6)
            endif
59        continue
```

Fig. 5.46 Particle collision part in the main routine of the non-parallelized code (phase1)

```
      if(model.eq.1) then
        call fhp1
      elseif(model.eq.2) then
        call fhp2
      elseif(model.eq.3) then
        call fhp3
      endif
            if(ierr.eq.-1) then
              write(6,*) 'Error in fhp on rank:',MYRANK
            endif
            if(idebug.eq.1) then
              call countPQ(bstate)
              call countPQ_sum
              if(MYRANK.eq.0) then
                write(6,*) 'FHP ',snumpar,' (',smomx,',',smomy,') '
                write(6,*) (sp_num(i),i=istart,nsite)
              endif
            endif

 59 continue
```

Fig. 5.47 Particle collision part in the main routine of the parallelized code (phase1)

```
 1       subroutine FHP1(xx,yy,before,after,twall,aa,blabel,alabel,ierr)
 2
 3       integer xx,yy,ierr
 4       integer twall(1:xx*yy,0:6)
 5       integer before(1:xx*yy,0:6),after(1:xx*yy,0:6),aa(1:xx*yy)
 6       integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
 7
 8       integer i,k,nsite,itmp
 9       integer j0,j1,j2,j3,j4,j5,j6
10       integer c0,c1,c2,c3,c4,c5,c6
11       real    rand(1)
12       integer col21,col22,col3
13       integer a_col21a,a_col22a
14       integer a_col21b,a_col22b
15
16
17       common/RANSU/iseed
18       data xxx/0/
19
20       nsite=xx*yy
21       ierr=0
22
23       call copydata(xx,yy,before,after)
24       call copydata(xx,yy,blabel,alabel)
25
26       do 15 k=1,nsite                              ← 分割対象
27 c        call ranu2(xxx,rand,1,icon)
28          call ranu2(iseed,rand,1,icon)
29          if(rand(1).gt.0.5) then
30             aa(k)=1
31          else
32             aa(k)=0
33          endif
34 15    continue
35
36       do 20 i=1,3
37          do 30 k=1,nsite                           ← 分割対象
38             j0=i
39             j1=i+1
40             j2=i+2
41             j3=i+3
42             j4=i+4
43             j5=i+5
44             j6=i+6
45             if(j4.gt.6) then
46                j4=j4-6
47             endif
48             if(j5.gt.6) then
49                j5=j5-6
50             endif
51             if(j6.gt.6) then
52                j6=j6-6
53             endif
```

Fig. 5.48 Subroutine fhp1 of the non-parallelized code (1/2)

```
54 c            /* two bodies collision */
55              c0=before(k,j0)
56              c1=before(k,j1)
57              c2=before(k,j2)
58              c3=before(k,j3)
59              c4=before(k,j4)
60              c5=before(k,j5)
61              c6=before(k,j6)
62
63                 .
64                 .
65                 .
66                 .
67                 .
68                 .
69                 .
70
71 30          continue
72 20       continue
73
74 c        /* checking */
75          do 125 i=0,6
76          do 125 j=1,nsite
77             if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
78                write(6,*) "After collision"
79                write(6,521) after(j,i),j,i
80                ierr=-1
81             endif
82 125      continue
83
84 521      format(1h ,"Error state= ",i10," at ",i6," of site",i4)
85
86
87          return
88          end
```

Fig. 5.48 Subroutine fhp1 of the non-parallelized code (2/2)

```
 1       subroutine fhp1
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer itmp
 7       integer j0,j1,j2,j3,j4,j5,j6
 8       integer c0,c1,c2,c3,c4,c5,c6
 9       integer col21,col22,col3
10       integer a_col21a,a_col22a
11       integer a_col21b,a_col22b
12
13       do k1=istart,nsite
14       do k2=1,numsite_para
15         la(k2,k1)=0
16         la(k2,k1)=tab(astate(k2),k1)
17       enddo
18       enddo
19
20       do 15 k1=ixs*iwidth+1,ixs*iwidth+numsite     ← 分割後
21         call ranu2(iseed,rand,1,icon)
22         if(rand(1).gt.0.5) then
23           iduma(k1)=1
24         else
25           iduma(k1)=0
26         endif
27    15 continue
28
29       do 20 i=1,3
30
31           j0=i
32           j1=i+1
33           j2=i+2
34           j3=i+3
35           j4=i+4
36           j5=i+5
37           j6=i+6
38           if(j4.gt.6) then
39             j4=j4-6
40           endif
41           if(j5.gt.6) then
42             j5=j5-6
43           endif
44           if(j6.gt.6) then
45             j6=j6-6
46           endif
47
48         do 30 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
```

Fig. 5.49 Subroutine fhp1 of the parallelized code (1/2)

```
49
50            c0=tab(astate(k1),j0)
51            c1=tab(astate(k1),j1)
52            c2=tab(astate(k1),j2)
53            c3=tab(astate(k1),j3)
54            c4=tab(astate(k1),j4)
55            c5=tab(astate(k1),j5)
56            c6=tab(astate(k1),j6)
57
58                  .
59                  .
60                  .
61                  .
62                  .
63                  .
64                  .
65
66     30   continue
67     20 continue
68
69        call rehash(la,bstate)
70
71        if(idebug.eq.1) then
72        ierr=0
73        do 125 k1=istart,nsite
74        do 126 k2=ixs*iwidth+1,ixs*iwidth+numsite
75         if((tab(bstate(k2),k1).lt.0).or.(tab(bstate(k2),k1).gt.1)) then
76           write(6,*) 'After collision on rank:',MYRANK
77           write(6,521) tab(bstate(k2),k1),k2,k1
78           ierr=-1
79         endif
80    126 continue
81    125 continue
82        endif
83
84    521 format(1h ,"Error state= ",i10," at ",i6," of site",i4)
85
86        return
87        end
```

Fig. 5.49 Subroutine fhp1 of the parallelized code (2/2)

```
 1       subroutine FHP2(xx,yy,before,after,twall,a,blabel,alabel,ierr)
 2
 3       integer xx,yy,ierr
 4       integer a(1:xx*yy)
 5       integer twall(1:xx*yy,0:6)
 6       integer before(1:xx*yy,0:6),after(1:xx*yy,0:6)
 7       integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
 8
 9       integer i,j,k,nsite,itmp
10       integer j0,ja1,ja2,ja3,ja4,ja5,ja6,itmp2
11       integer jb1,jb2,jb3,jb4,jb5,jb6,jb7
12       integer c0,c1,c2,c3,c4,c5,c6,c7
13       real    rand(1)
14       integer col21,col22,col3
15       integer col2r,col2m
16       integer a_col21a,a_col22a
17       integer a_col21b,a_col22b
18
19       common/RANSU/iseed
20       data xxx/0/
21
22       nsite=xx*yy
23       ierr=0
24
25       call copydata(xx,yy,before,after)
26       call copydata(xx,yy,blabel,alabel)
27
28       do 25 j=1,nsite                          ← 分割対象
29 c        call ranu2(xxx,rand,1,icon)
30          call ranu2(iseed,rand,1,icon)
31          if(rand(1).gt.0.5) then
32             a(j)=1
33          else
34             a(j)=0
35          endif
36 25    continue
37
```

Fig. 5.50 Subroutine fhp2 of the non-parallelized code (1/4)

```
38       j0=0
39       do 50 i=1,3
40          do 60 k=1,nsite                          ← 分割対象
41             jb1=i
42             jb2=i+1
43             jb3=i+2
44             jb4=i+3
45             jb5=i+4
46             jb6=i+5
47             jb7=i+6
48             if(jb4.gt.6) then
49                jb4=jb4-6
50             endif
51             if(jb5.gt.6) then
52                jb5=jb5-6
53             endif
54             if(jb6.gt.6) then
55                jb6=jb6-6
56             endif
57             if(jb7.gt.6) then
58                jb7=jb7-6
59             endif
60             c1=before(k,jb1)
61             c2=before(k,jb2)
62             c3=before(k,jb3)
63             c4=before(k,jb4)
64             c5=before(k,jb5)
65             c6=before(k,jb6)
66             c7=before(k,jb7)
67
68                .
69                .
70                .
71                .
72                .
73                .
74
75 60       continue
76 50    continue
77
```

Fig. 5.50 Subroutine fhp2 of the non-parallelized code (2/4)

```
78        j0=0
79        do 30 i=1,6
80           do 40 k=1,nsite                          ← 分割対象
81              ja1=i
82              ja2=i+1
83              ja3=i+2
84              ja4=i+3
85              ja5=i+4
86              ja6=i+5
87              if(ja2.gt.6) then
88                 ja2=ja2-6
89              endif
90              if(ja3.gt.6) then
91                 ja3=ja3-6
92              endif
93              if(ja4.gt.6) then
94                 ja4=ja4-6
95              endif
96              if(ja5.gt.6) then
97                 ja5=ja5-6
98              endif
99              if(ja6.gt.6) then
100                ja6=ja6-6
101             endif
102             c0=before(k,j0)
103             c1=before(k,ja1)
104             c2=before(k,ja2)
105             c3=before(k,ja3)
106             c4=before(k,ja4)
107             c5=before(k,ja5)
108             c6=before(k,ja6)
109
110                .
111                .
112                .
113                .
114                .
115                .
116                .
117                .
118
119
120 40         continue
121 30      continue
122
```

Fig. 5.50 Subroutine fhp2 of the non-parallelized code (3/4)

```
123 c     /* checking */
124       do 125 i=0,6
125       do 125 j=1,nsite
126          if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
127             write(6,*) "After collision"
128             write(6,521) after(j,i),j,i
129             ierr=-1
130          endif
131 125    continue
132
133 521    format(1h ,"Error state= ",i10," at ",i6," of site",i4)
134
135       return
136       end
```

Fig. 5.50 Subroutine fhp2 of the non-parallelized code (4/4)

```
 1       subroutine fhp2
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 j0,ja1,ja2,ja3,ja4,ja5,ja6,itmp2
 7       integer*4 jb1,jb2,jb3,jb4,jb5,jb6,jb7
 8       integer*4 c0,c1,c2,c3,c4,c5,c6,c7
 9       integer*4 col21,col22,col3
10       integer*4 col2r,col2m
11       integer*4 a_col21a,a_col22a
12       integer*4 a_col21b,a_col22b
13       integer*4 i,itmp
14
15       do k1=istart,nsite
16       do k2=1,numsite_para
17         la(k2,k1)=0
18         la(k2,k1)=tab(astate(k2),k1)
19       enddo
20       enddo
21
22       do 25 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
23         call ranu2(iseed,rand,1,icon)
24         if(rand(1).gt.0.5) then
25           iduma(k1)=1
26         else
27           iduma(k1)=0
28         endif
29    25 continue
30
```

Fig. 5.51 Subroutine fhp2 of the parallelized code (1/4)

```
31        j0=0
32        do 50 i=1,3
33
34            jb1=i
35            jb2=i+1
36            jb3=i+2
37            jb4=i+3
38            jb5=i+4
39            jb6=i+5
40            jb7=i+6
41            if(jb4.gt.6) then
42              jb4=jb4-6
43            endif
44            if(jb5.gt.6) then
45              jb5=jb5-6
46            endif
47            if(jb6.gt.6) then
48              jb6=jb6-6
49            endif
50            if(jb7.gt.6) then
51              jb7=jb7-6
52            endif
53
54          do 60 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
55
56            c1=tab(astate(k1),jb1)
57            c2=tab(astate(k1),jb2)
58            c3=tab(astate(k1),jb3)
59            c4=tab(astate(k1),jb4)
60            c5=tab(astate(k1),jb5)
61            c6=tab(astate(k1),jb6)
62            c7=tab(astate(k1),jb7)
63
64                  .
65                  .
66                  .
67                  .
68                  .
69                  .
70
71      60    continue
72      50 continue
73
```

Fig. 5.51 Subroutine fhp2 of the parallelized code (2/4)

```
74        j0=0
75        do 30 i=1,6
76            ja1=i
77            ja2=i+1
78            ja3=i+2
79            ja4=i+3
80            ja5=i+4
81            ja6=i+5
82            if(ja2.gt.6) then
83              ja2=ja2-6
84            endif
85            if(ja3.gt.6) then
86              ja3=ja3-6
87            endif
88            if(ja4.gt.6) then
89              ja4=ja4-6
90            endif
91            if(ja5.gt.6) then
92              ja5=ja5-6
93            endif
94            if(ja6.gt.6) then
95              ja6=ja6-6
96            endif
97
98          do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
99
100           c0=tab(astate(k1),j0)
101           c1=tab(astate(k1),ja1)
102           c2=tab(astate(k1),ja2)
103           c3=tab(astate(k1),ja3)
104           c4=tab(astate(k1),ja4)
105           c5=tab(astate(k1),ja5)
106           c6=tab(astate(k1),ja6)
107
108                .
109                .
110                .
111                .
112                .
113                .
114                .
115                .
116
117
118     40    continue
119     30  continue
```

Fig. 5.51 Subroutine fhp2 of the parallelized code (3/4)

```
120
121       call rehash(la,bstate)
122
123       if(idebug.eq.1) then
124       ierr=0
125       do 125 k1=istart,nsite
126       do 126 k2=ixs*iwidth+1,ixs*iwidth+numsite
127       if((tab(bstate(k2),k1).lt.0).or.(tab(bstate(k2),k1).gt.1)) then
128        write(6,*) 'After collision on rank:',MYRANK
129        write(6,521) tab(bstate(k2),k1),k2,k1
130        ierr=-1
131       endif
132   126 continue
133   125 continue
134       endif
135
136   521 format(1h ,"Error state= ",i10," at ",i6," of site",i4)
137
138       return
139       end
```

Fig. 5.51 Subroutine fhp2 of the parallelized code (4/4)

```
 1       subroutine FHP3(xx,yy,before,after,twall,a,b,blabel,alabel,ncol
 2      +,ierr)
 3
 4       integer xx,yy,ierr
 5       integer a(1:xx*yy),b(1:xx*yy)
 6       integer twall(1:xx*yy,0:6)
 7       integer before(1:xx*yy,0:6),after(1:xx*yy,0:6)
 8       integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
 9
10       real ncol(1:xx*yy)
11
12       integer i,j,k,nsite,itmp,itmp1,itmp2,itmp3,itmp4
13       integer j0,j1,j2,j3,j4,j5,j6,j7
14       integer c0,c1,c2,c3,c4,c5,c6,c7
15       common/RANSU/iseed
16
17       real    rand(1)
18       integer col21,col22,col3
19       integer col21d,col22d
20       integer col2ra,col2ma
21       integer col2mb
22       integer col2r1b,col2r2b
23       integer col2mc
24       integer col2r1c,col2r2c
25       integer a_col21a,a_col22a
26       integer a_col21da,a_col22da
27       integer a_col21b,a_col22b
28       integer a_col21db,a_col22db
29
30       real*8 tg1,tg2,tga,tgb
31
32       data xxx/0/
33
34 c     CALL gettod(tga)
35       nsite=xx*yy
36       ierr=0
37
38       call copydata(xx,yy,before,after)
39       call copydata(xx,yy,blabel,alabel)
40
41       do 20 j=1,nsite                    ← 分割対象
42 c        call ranu2(xxx,rand,1,icon)
43          call ranu2(iseed,rand,1,icon)
44          if(rand(1).gt.0.5) then
45             a(j)=1
46             b(j)=1
47          else
48             a(j)=0
49             b(j)=0
50          endif
51 20    continue
```

Fig. 5.52 Subroutine fhp3 of the non-parallelized code (1/4)

```
52
53 c      CALL gettod(tg1)
54
55        j0=0
56        do 30 i=1,3
57           j1=i
58           j2=i+1
59           j3=i+2
60           j4=i+3
61           j5=i+4
62           j6=i+5
63           j7=i+6
64           if(j5.gt.6) then
65              j5=j5-6
66           endif
67           if(j6.gt.6) then
68              j6=j6-6
69           endif
70           if(j7.gt.6) then
71              j7=j7-6
72           endif
73           do 40 k=1,nsite                    ← 分割対象
74
75                .
76                .
77                .
78                .
79                .
80                .
81
82 40        continue
83 30     continue
```

Fig. 5.52 Subroutine fhp3 of the non-parallelized code (2/4)

```
84
85 c      CALL gettod(tg2)
86 c      write(6,*) "  ",(tg1-tga)*1.0D-6
87 c      write(6,*) "  ",(tg2-tg1)*1.0D-6
88 c      write(6,*) "  stotal",(tg2-tga)*1.0D-6
89
90        j0=0
91        do 50 i=1,6
92           j1=i
93           j2=i+1
94           j3=i+2
95           j4=i+3
96           j5=i+4
97           j6=i+5
98           j7=i+6
99           if(j2.gt.6) then
100             j2=j2-6
101          endif
102          if(j3.gt.6) then
103             j3=j3-6
104          endif
105          if(j4.gt.6) then
106             j4=j4-6
107          endif
108          if(j5.gt.6) then
109             j5=j5-6
110          endif
111          if(j6.gt.6) then
112             j6=j6-6
113          endif
114          if(j7.gt.6) then
115             j7=j7-6
116          endif
117          do 60 k=1,nsite              ← 分割対象
118
119                .
120                .
121                .
122                .
123                .
124                .
125
126 60        continue
127 50     continue
```

Fig. 5.52 Subroutine fhp3 of the non-parallelized code (3/4)

```
128
129
130 c     CALL gettod(tgb)
131 c     write(6,*) "  stotal ",(tgb-tg2)*1.0D-6
132 c     write(6,*) "   total ",(tgb-tga)*1.0D-6
133
134 c     /* checking */
135       do 125 i=0,6
136       do 125 j=1,nsite
137          if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
138             write(6,*) "After collision"
139             write(6,521) after(j,i),i,j
140             ierr=-1
141          endif
142 125   continue
143       do 126 i=0,6
144       do 126 j=1,nsite
145          if(after(j,i)*twall(j,i).lt.0) then
146             write(6,*) "After collision (wall)"
147             write(6,521) after(j,i),i,j
148             ierr=-1
149          endif
150 126   continue
151
152 521   format(1h ,"Error state= ",i3," at ",i6," of site",i10)
153
154
155       return
156       end
```

Fig. 5.52 Subroutine fhp3 of the non-parallelized code (4/4)

```
 1       subroutine fhp3
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 c0,c1,c2,c3,c4,c5,c6,c7
 7       integer*4 col21,col22,col3
 8       integer*4 col21d,col22d
 9       integer*4 col2ra,col2ma
10       integer*4 col2mb
11       integer*4 col2r1b,col2r2b
12       integer*4 col2mc
13       integer*4 col2r1c,col2r2c
14       integer*4 a_col21a,a_col22a
15       integer*4 a_col21da,a_col22da
16       integer*4 a_col21b,a_col22b
17       integer*4 a_col21db,a_col22db
18
19       integer*4 icon,j0,i,j1,j2,j3,j4,j5,j6,j7,itmp
20       integer*4 itmp1,itmp2,itmp3,itmp4
21
22       do k1=istart,nsite
23       do k2=1,numsite_para
24         la(k2,k1)=0
25         la(k2,k1)=tab(astate(k2),k1)
26       enddo
27       enddo
28
29       do k1=1,numsite_para
30         iduma(k1)=0
31       enddo
32
33       do 20 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
34         call ranu2(iseed,rand,1,icon)
35         if(rand(1).gt.0.5) then
36           iduma(k1)=1
37         else
38           iduma(k1)=0
39         endif
40    20 continue
41
```

Fig. 5.53 Subroutine fhp3 of the parallelized code (1/4)

```
42        j0=0
43
44        do 30 i=1,3
45
46          j1=i
47          j2=i+1
48          j3=i+2
49          j4=i+3
50          j5=i+4
51          j6=i+5
52          j7=i+6
53          if(j5.gt.6) then
54            j5=j5-6
55          endif
56          if(j6.gt.6) then
57            j6=j6-6
58          endif
59          if(j7.gt.6) then
60            j7=j7-6
61          endif
62
63          do 40 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
64
65                .
66                .
67                .
68                .
69                .
70                .
71
72    40    continue
73
74    30 continue
```

Fig. 5.53 Subroutine **fhp3** of the parallelized code (2/4)

```
75
76        j0=0
77
78        do 50 i=1,6
79
80          j1=i
81          j2=i+1
82          j3=i+2
83          j4=i+3
84          j5=i+4
85          j6=i+5
86          j7=i+6
87          if(j2.gt.6) then
88            j2=j2-6
89          endif
90          if(j3.gt.6) then
91            j3=j3-6
92          endif
93          if(j4.gt.6) then
94            j4=j4-6
95          endif
96          if(j5.gt.6) then
97            j5=j5-6
98          endif
99          if(j6.gt.6) then
100           j6=j6-6
101         endif
102         if(j7.gt.6) then
103           j7=j7-6
104         endif
105
106         do 60 k1=ixs*iwidth+1,ixs*iwidth+numsite ← 分割後
107
108                 .
109                 .
110                 .
111                 .
112                 .
113                 .
114
115     60    continue
116
117     50 continue
```

Fig. 5.53 Subroutine fhp3 of the parallelized code (3/4)

```
118
119       call rehash(la,bstate)
120
121       if(idebug.eq.1) then
122       ierr=0
123       do 125 k1=istart,nsite
124       do 126 k2=ixs*iwidth+1,ixs*iwidth+numsite
125         if( (tab(bstate(k2),k1).lt.0).or.(tab(bstate(k2),k1).gt.1) )
126      &then
127           write(6,*) 'After collision on rank:',MYRANK
128           write(6,521) tab(bstate(k2),k1),k1,k2
129           ierr=-1
130         endif
131   126 continue
132   125 continue
133
134       do 127 k1=istart,nsite
135       do 128 k2=ixs*iwidth+1,ixs*iwidth+numsite
136         if( tab(bstate(k2),k1)*tabw(iwall(k2),k1).lt.0 ) then
137           write(6,*) 'After collision on rank:',MYRANK
138           write(6,521) tab(bstate(k2),k1),k1,k2
139           ierr=-1
140         endif
141   128 continue
142   127 continue
143       endif
144
145   521 format(1h ,'Error state = ',i3,' at ',i6,' of site',i10)
146
147       return
148       end
```

Fig. 5.53 Subroutine **fhp3** of the parallelized code (4/4)

```
1           call outfiles(xs,ys,n,state,acumtr,acumlg,acumbo,model)
2           write(6,*) "---------------------- write file ",n
```

Fig. 5.54 Data output part in the main routine of the non-parallelized code (phase1)

```
1       call outfiles(bstate,n)
```

Fig. 5.55 Data output part in the main routine of the parallelized code (phase1)

```
 1       subroutine outfiles(xx,yy,nn,istate,atr,alg,abo,model)
 2
 3       integer xx,yy,nn,model
 4       integer istate(1:xx*yy,0:6)
 5       integer atr(1:xx*yy,0:6)
 6       integer abo(1:xx*yy,0:6)
 7       integer alg(1:xx*yy,0:6)
 8       integer i,j,k,nsite
 9
10       nsite=xx*yy
11       open(unit=nn)
12       do 55 k=1,nsite
13          i=itox(xx,yy,k)
14          j=itoy(xx,yy,k)
15          if(i.le.xx.and.j.le.yy) then
16             if(model.ne.1) then
17                write(nn,500) i,j,
18      +            istate(k,0),istate(k,1),
19      +            istate(k,2),istate(k,3),
20      +            istate(k,4),istate(k,5),
21      +            istate(k,6),
22      +            atr(k,0),atr(k,1),atr(k,2),
23      +            atr(k,3),atr(k,4),atr(k,5),
24      +            atr(k,6),
25      +            alg(k,1),alg(k,2),alg(k,3),
26      +            alg(k,4),alg(k,5),alg(k,6),
27      +            abo(k,1),abo(k,2),abo(k,3),
28      +            abo(k,4),abo(k,5),abo(k,6)
29             else
30                write(nn,510) i,j,
31      +            istate(k,1),istate(k,2),
32      +            istate(k,3),istate(k,4),
33      +            istate(k,5),istate(k,6),
34      +            atr(k,0),atr(k,1),atr(k,2),
35      +            atr(k,3),atr(k,4),atr(k,5),
36      +            atr(k,6),
37      +            alg(k,1),alg(k,2),alg(k,3),
38      +            alg(k,4),alg(k,5),alg(k,6),
39      +            abo(k,1),abo(k,2),abo(k,3),
40      +            abo(k,4),abo(k,5),abo(k,6)
41             endif
42          endif
43 55    continue
44       close(nn)
45
           .
           .
           .
56
57       return
58       end
```

Fig. 5.56 Subroutine outfiles of the non-parallelized code

```
 1       subroutine outfiles(istate,nn)
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 istate(0:numsite_para)
 7       integer*4 i,j
 8
 9       open(unit=nn)
10       do 55 k1=1,ixs*(iys+iwidth*2)
11         i=itox(ixs,iys,k1)
12         j=itoy(ixs,iys,k1)
13         if(i.le.ixs.and.(j.ge.iwidth+1.and.j.le.iwidth+iys)) then
14          if(model.ne.1) then
15           write(nn,500) i,j-iwidth+(iys*MYRANK),
16      +             tab(istate(k1),0),tab(istate(k1),1),
17      +             tab(istate(k1),2),tab(istate(k1),3),
18      +             tab(istate(k1),4),tab(istate(k1),5),
19      +             tab(istate(k1),6),
20      +             acumtr(k1,0),acumtr(k1,1),acumtr(k1,2),
21      +             acumtr(k1,3),acumtr(k1,4),acumtr(k1,5),
22      +             acumtr(k1,6),
23      +             acumlg(k1,1),acumlg(k1,2),acumlg(k1,3),
24      +             acumlg(k1,4),acumlg(k1,5),acumlg(k1,6),
25      +             acumbo(k1,1),acumbo(k1,2),acumbo(k1,3),
26      +             acumbo(k1,4),acumbo(k1,5),acumbo(k1,6)
27           else
28            write(nn,510) i,j-iwidth+(iys*MYRANK),
29      +             tab(istate(k1),1),tab(istate(k1),2),
30      +             tab(istate(k1),3),tab(istate(k1),4),
31      +             tab(istate(k1),5),tab(istate(k1),6),
32      +             acumtr(k1,1),acumtr(k1,2),acumtr(k1,3),
33      +             acumtr(k1,4),acumtr(k1,5),acumtr(k1,6),
34      +             acumlg(k1,1),acumlg(k1,2),acumlg(k1,3),
35      +             acumlg(k1,4),acumlg(k1,5),acumlg(k1,6),
36      +             acumbo(k1,1),acumbo(k1,2),acumbo(k1,3),
37      +             acumbo(k1,4),acumbo(k1,5),acumbo(k1,6)
38           endif
39         endif
40    55 continue
41       close(nn)
42
43       write(60+MYRANK,*) iseed
44
           .
           .
           .
55
56       return
57       end
```

Fig. 5.57 Subroutine outfiles of the parallelized code

```
1       else if(phase.ne.1) then
2           call readdata(xs,ys,model,state,acumtr,acumlg,acumbo)
3            n=11
4       endif
5       n=n+1
```

Fig. 5.58 Data input part in the main routine of the non-parallelized code (phase2)

```
1       elseif(iphase.ne.1) then
2         n=10+MYRANK
3         call readdata(bstate,n)
4         n=10+NPROCS+MYRANK
5       endif
```

Fig. 5.59 Data input part in the main routine of the parallelized code (phase2)

```
 1       subroutine readdata(xx,yy,model,istate,iacumtr,iacumlg,iacumbo)
 2       integer xx,yy,model
 3       integer istate(1:xx*yy,0:6)
 4       integer iacumtr(1:xx*yy,0:6)
 5       integer iacumlg(1:xx*yy,0:6)
 6       integer iacumbo(1:xx*yy,0:6)
 7
 8       integer ix,iy,site,v0,v1,v2,v3,v4,v5,v6
 9       integer itr0,itr1,itr2,itr3,itr4,itr5,itr6
10       integer ilg1,ilg2,ilg3,ilg4,ilg5,ilg6
11       integer ibo1,ibo2,ibo3,ibo4,ibo5,ibo6
12
13 c     ##### read state data #####
14       open(UNIT=9
15 c    +,FILE='/dg04/ufs01/j4888/F/src/CA/work/CLASS/state.tmp'
16 c    +,FILE='/dg04/ufs04/j4888/F/src/CA/work/CLASS/state.tmp'
17 c    +,FILE='/wka2/j4888/work/CLASS/state.tmp'
18      +,FILE='/wka4/j4888/work/CLASS/state.tmp'
19      +,STATUS='old')
20       if(model.eq.1) then
21 *VOCL LOOP,SCALAR
22          do 10 i=1,xx*yy
23             read(9,410) ix,iy,v1,v2,v3,v4,v5,v6
24     +,itr1,itr2,itr3,itr4,itr5,itr6
25     +,ilg1,ilg2,ilg3,ilg4,ilg5,ilg6
26     +,ibo1,ibo2,ibo3,ibo4,ibo5,ibo6
27             site=xx*(iy-1)+ix
28             istate(site,0) = 0
29             istate(site,1) = v1
30             istate(site,2) = v2
31             istate(site,3) = v3
32             istate(site,4) = v4
33             istate(site,5) = v5
34             istate(site,6) = v6
35             iacumtr(site,0) = 0
36             iacumtr(site,1) = itr1
37             iacumtr(site,2) = itr2
38             iacumtr(site,3) = itr3
39             iacumtr(site,4) = itr4
40             iacumtr(site,5) = itr5
41             iacumtr(site,6) = itr6
42             iacumlg(site,0) = 0
43             iacumlg(site,1) = ilg1
44             iacumlg(site,2) = ilg2
45             iacumlg(site,3) = ilg3
46             iacumlg(site,4) = ilg4
47             iacumlg(site,5) = ilg5
48             iacumlg(site,6) = ilg6
49             iacumbo(site,0) = 0
50             iacumbo(site,1) = ibo1
51             iacumbo(site,2) = ibo2
```

Fig. 5.60 Subroutine readdata of the non-parallelized code (1/2)

```
52              iacumbo(site,3) = ibo3
53              iacumbo(site,4) = ibo4
54              iacumbo(site,5) = ibo5
55              iacumbo(site,6) = ibo6
56 10         continue
57        else if(model.ne.1) then
58 *VOCL LOOP,SCALAR
59           do 20 i=1,xx*yy
60              read(9,400) ix,iy,v0,v1,v2,v3,v4,v5,v6
61       +,itr0,itr1,itr2,itr3,itr4,itr5,itr6
62       +,ilg1,ilg2,ilg3,ilg4,ilg5,ilg6
63       +,ibo1,ibo2,ibo3,ibo4,ibo5,ibo6
64              site=xx*(iy-1)+ix
65              istate(site,0) = v0
66              istate(site,1) = v1
67              istate(site,2) = v2
68              istate(site,3) = v3
69              istate(site,4) = v4
70              istate(site,5) = v5
71              istate(site,6) = v6
72              iacumtr(site,0) = itr0
73              iacumtr(site,1) = itr1
74              iacumtr(site,2) = itr2
75              iacumtr(site,3) = itr3
76              iacumtr(site,4) = itr4
77              iacumtr(site,5) = itr5
78              iacumtr(site,6) = itr6
79              iacumlg(site,0) = 0
80              iacumlg(site,1) = ilg1
81              iacumlg(site,2) = ilg2
82              iacumlg(site,3) = ilg3
83              iacumlg(site,4) = ilg4
84              iacumlg(site,5) = ilg5
85              iacumlg(site,6) = ilg6
86              iacumbo(site,0) = 0
87              iacumbo(site,1) = ibo1
88              iacumbo(site,2) = ibo2
89              iacumbo(site,3) = ibo3
90              iacumbo(site,4) = ibo4
91              iacumbo(site,5) = ibo5
92              iacumbo(site,6) = ibo6
93 20         continue
94         endif
95         close(9)
96
97 400     format(I4,I6,I4,6I3,19i6)
98 410     format(I4,I6,I4,5I3,18i6)
99
100         return
101         end
```

Fig. 5.60 Subroutine readdata of the non-parallelized code (2/2)

```
 1       subroutine readdata(istate,nn)
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 istate(0:numsite_para)
 7       integer*4 ir_tmp
 8
 9       do k1=istart,nsite
10       do k2=1,numsite_para
11         la(k2,k1)=0
12       enddo
13       enddo
14       do k1=1,numsite_para
15         istate(k1)=0
16       enddo
17
18       open(unit=nn)
19       do 55 k1=1,ixs*(iys+iwidth*2)
20         i=itox(ixs,iys,k1)
21         j=itoy(ixs,iys,k1)
22         if(i.le.ixs.and.(j.ge.iwidth+1.and.j.le.iwidth+iys)) then
23          if(model.ne.1) then
24           read(nn,500) i,ir_tmp,
25      +             la(k1,0),la(k1,1),
26      +             la(k1,2),la(k1,3),
27      +             la(k1,4),la(k1,5),
28      +             la(k1,6),
29      +             acumtr(k1,0),acumtr(k1,1),acumtr(k1,2),
30      +             acumtr(k1,3),acumtr(k1,4),acumtr(k1,5),
31      +             acumtr(k1,6),
32      +             acumlg(k1,1),acumlg(k1,2),acumlg(k1,3),
33      +             acumlg(k1,4),acumlg(k1,5),acumlg(k1,6),
34      +             acumbo(k1,1),acumbo(k1,2),acumbo(k1,3),
35      +             acumbo(k1,4),acumbo(k1,5),acumbo(k1,6)
36                    j=ir_tmp+iwidth-(iys*MYRANK)
37           else
38            read(nn,510) i,ir_tmp,
39      +             la(k1,1),la(k1,2),
40      +             la(k1,3),la(k1,4),
41      +             la(k1,5),la(k1,6),
42      +             acumtr(k1,1),acumtr(k1,2),acumtr(k1,3),
43      +             acumtr(k1,4),acumtr(k1,5),acumtr(k1,6),
44      +             acumlg(k1,1),acumlg(k1,2),acumlg(k1,3),
45      +             acumlg(k1,4),acumlg(k1,5),acumlg(k1,6),
46      +             acumbo(k1,1),acumbo(k1,2),acumbo(k1,3),
47      +             acumbo(k1,4),acumbo(k1,5),acumbo(k1,6)
48                    j=ir_tmp+iwidth-(iys*MYRANK)
49           endif
50          endif
51    55 continue
```

Fig. 5.61 Subroutine readdata of the parallelized code (1/2)

```
52        close(nn)
53
54        read(60+MYRANK,*) iseed
55
56        call rehash(la,istate)
57
58    500 format(i4,2x,i4,2x
59       +,i2,1x,i2,1x,i2,1x,i2,1x,i2,1x,i2,1x,i2,1x,i2,1x
60       +,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x
61       +,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x
62       +,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5)
63    510 format(i4,2x,i4,2x
64       +,i2,1x,i2,1x,i2,1x,i2,1x,i2,1x,i2,1x
65       +,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x
66       +,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x
67       +,i5,1x,i5,1x,i5,1x,i5,1x,i5,1x,i5)
68
69        return
70        end
```

Fig. 5.61 Subroutine readdata of the parallelized code (2/2)

```
 1  c         /***** moving wall ******************/
 2            if(mvrate.ne.0.0) then
 3               call initdat2(xs,ys,iacum,0)
 4               call copydata(xs,ys,state,b_state)
 5               call copydata(xs,ys,label,b_label)
 6                call mvwall(xs,ys,b_state,a_state,iacum,mvrate
 7         +                  ,b_label,a_label,mvpos,mvdirc,ierr)
 8               if(ierr.eq.-1) then
 9                  write(6,*) "Error in mvwall"
10               endif
11               call copydata(xs,ys,a_state,state)
12               call copydata(xs,ys,a_label,label)
13               do 92 j=1,numsite
14                  acumbo(j,0)=acumbo(j,0)+iacum(j,0)
15                  acumbo(j,1)=acumbo(j,1)+iacum(j,1)
16                  acumbo(j,2)=acumbo(j,2)+iacum(j,2)
17                  acumbo(j,3)=acumbo(j,3)+iacum(j,3)
18                  acumbo(j,4)=acumbo(j,4)+iacum(j,4)
19                  acumbo(j,5)=acumbo(j,5)+iacum(j,5)
20                  acumbo(j,6)=acumbo(j,6)+iacum(j,6)
21  92           continue
22
23  c            /* count Physical Quantity */
24               if(debug.eq.1) then
25               call countPQ(xs,ys,state,numpar,momx,momy,p_num,model)
26               write(6,*) "mvwall ",numpar," (",momx,",",momy,") "
27               write(6,*) (p_num(i),i=start,6)
28               endif
29            endif
```

Fig. 5.62 Part for adding momentum to particles in the main routine of the non-parallelized code (phase2)

```
      if(mvrate.ne.0.0) then
C**************************************
C
C     CALL MVWALL ((astate -> bstate) -> astate)
C
C**************************************
        call mvwall
           if(ierr.eq.-1) then
             write(6,*) 'Error in mvwall on rank:',MYRANK
           endif
        do k1=1,numsite_para
          astate(k1)=bstate(k1)
        enddo
        do k1=istart,nsite
        do k2=ixs*iwidth+1,ixs*iwidth+numsite
          acumbo(k2,k1)=acumbo(k2,k1)+iacum(k2,k1)
        enddo
        enddo
           if(idebug.eq.1) then
             call countPQ(astate)
             call countPQ_sum
             if(MYRANK.eq.0) then
               write(6,*) 'mvwall ',snumpar,' (',smomx,',',smomy,') '
               write(6,*) (sp_num(i),i=istart,nsite)
             endif
           endif
      endif
```

Fig. 5.63 Part for adding momentum to particles in the main routine of the parallelized code (phase2)

```
 1       subroutine mvwall(xx,yy,before,after,acum,rrate,blabel,alabel
 2      +,wall,opt,dirc,ierr,debug)
 3
 4       integer xx,yy,opt,dirc,ierr,debug
 5       integer before(1:xx*yy,0:6),after(1:xx*yy,0:6)
 6       integer acum(1:xx*yy,0:6),wall(1:xx*yy,0:6)
 7       integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
 8
 9       real rrate
10       common/RANSU/iseed
11       real rand(1)
12
13       integer i,j,k,nsite
14       integer i1,i2,i3,i4,i5,i6
15
16       nsite=xx*yy
17       ierr=0
18
19       call copydata(xx,yy,before,after)
20       call copydata(xx,yy,blabel,alabel)
21       call initdat2(xx,yy,acum,0)
22
23       if(dirc.eq.1) then
24          i1=1
25          i2=2
26          i3=3
27          i4=4
28          i5=5
29          i6=6
30       else
31          i1=4
32          i2=3
33          i3=2
34          i4=1
35          i5=6
36          i6=5
37       endif
38
39 c     /*** (adding momenta on the ceiling wall) ***/
40       if(opt.eq.1) then
41          do 40 k=1,xx                                    ← 上境界部
42             if(before(k,i4).eq.1.and.before(k,i1).eq.0
43      +         .and.wall(k,i1).eq.0) then
44                call ranu2(xxx,rand,1,icon)
45                if(rand(1).le.rrate) then
46                   after(k,i4)=0
47                   after(k,i1)=1
48                   acum(k,i1)=acum(k,i1)+2
49                   alabel(k,i4)=-2
50                   alabel(k,i1)=blabel(k,i4)
51                endif
52             endif
```

Fig. 5.64 Subroutine mvwall of the non-parallelized code (1/3)

```
53              if(before(k,i5).eq.1.and.before(k,i6).eq.0
54        +        .and.wall(k,i6).eq.0) then
55                 call ranu2(xxx,rand,1,icon)
56                 if(rand(1).le.rrate) then
57                    after(k,i5)=0
58                    after(k,i6)=1
59                    acum(k,i1)=acum(k,i1)+1
60                    alabel(k,i5)=-2
61                    alabel(k,i6)=blabel(k,i5)
62                 endif
63              endif
64 40        continue
65 c       /*** (adding momenta on the floor wall) ***/
66         else if(opt.eq.2) then                          ← 下境界部
67            do 50 k=xx*(yy-1)+1,xx*yy
68               if(before(k,i4).eq.1.and.before(k,i1).eq.0
69        +        .and.wall(k,i1).eq.0) then
70                 call ranu2(xxx,rand,1,icon)
71                 if(rand(1).le.rrate) then
72                    after(k,i4)=0
73                    after(k,i1)=1
74                    acum(k,i1)=acum(k,i1)+2
75                    alabel(k,i4)=-2
76                    alabel(k,i1)=blabel(k,i4)
77                 endif
78              endif
79              if(before(k,i3).eq.1.and.before(k,i2).eq.0
80        +        .and.wall(k,i2).eq.0) then
81                 call ranu2(xxx,rand,1,icon)
82                 if(rand(1).le.rrate) then
83                    after(k,i3)=0
84                    after(k,i2)=1
85                    acum(k,i1)=acum(k,i1)+1
86                    alabel(k,i3)=-2
87                    alabel(k,i2)=blabel(k,i3)
88                 endif
89              endif
90 50        continue
91 c       /*** (shear flow) ***/
92         else if(opt.eq.3) then                          ← 上下２分割
93            do 70 k=1,xx*(yy/2)                          ← 上半分のループ
94               if(before(k,0).eq.1.and.before(k,i1).eq.0
95        +        .and.wall(k,i1).eq.0) then
96                 call ranu2(xxx,rand,1,icon)
97                 if(rand(1).le.rrate) then
98                    after(k,0)=0
99                    after(k,i1)=1
100                   acum(k,i1)=acum(k,i1)+1
101                   alabel(k,0)=-2
102                   alabel(k,i1)=blabel(k,0)
103                endif
104             endif
105 70        continue
```

Fig. 5.64 Subroutine mvwall of the non-parallelized code (2/3)

```
106           do 75 k=xx*(yy/2)+1,xx*yy                    ← 下半分のループ
107              if(before(k,0).eq.1.and.before(k,i4).eq.0
108       +         .and.wall(k,i4).eq.0) then
109                 call ranu2(xxx,rand,1,icon)
110                 if(rand(1).le.rrate) then
111                    after(k,0)=0
112                    after(k,i4)=1
113                    acum(k,i4)=acum(k,i4)+1
114                    alabel(k,0)=-2
115                    alabel(k,i4)=blabel(k,0)
116                 endif
117              endif
118 75        continue
119        endif
120
121        if(debug.eq.1) then
122           do 87 i=0,6
123           do 87 j=1,nsite
124              if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
125                 write(6,*) "After moving wall process"
126                 write(6,521) after(j,i),j,i
127                 ierr=-1
128              endif
129 87        continue
130        endif
131
132 521    format(1h ,"Error state= ",i10," at ",i6," of site",i4)
133
134
135        return
136        end
```

Fig. 5.64 Subroutine mvwall of the non-parallelized code (3/3)

```
 1       subroutine mvwall
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       integer*4 idiv_point(2)
 7
 8       idiv=(iys*NPROCS)/2
 9       iyss_total=0
10       do iirank=0,ipe-1
11         iys_total=0
12         do k=1,iys
13           iys_total=iys_total+1
14           iyss_total=iyss_total+1
15           if(iyss_total.eq.idiv) then
16             idiv_point(1)=iirank
17             idiv_point(2)=iys_total
18           else
19           endif
20         enddo
21       enddo
22
23       do ig=istart,nsite
24       do igg=1,numsite_para
25         la(igg,ig)=0
26         la(igg,ig)=tab(bstate(igg),ig)
27         iacum(igg,ig)=0
28       enddo
29       enddo
30
31       if(imvdirc.eq.1) then
32          i1=1
33          i2=2
34          i3=3
35          i4=4
36          i5=5
37          i6=6
38       else
39          i1=4
40          i2=3
41          i3=2
42          i4=1
43          i5=6
44          i6=5
45       endif
46
```

Fig. 5.65 Subroutine mvwall of the parallelized code (1/4)

```
47        if(imvpos.eq.1) then
48          if(MYRANK.eq.0) then                    ← ランク０：上境界部
49            do 10 k=ixs*iwidth+1,ixs*iwidth+ixs
50              if(tab(bstate(k),i4).eq.1.and.tab(bstate(k),i1).eq.0
51       &         .and.tabw(iwall(k),i1).eq.0) then
52                call ranu2(iseed,rand,1,icon)
53                if(rand(1).le.mvrate) then
54                  la(k,i4)=0
55                  la(k,i1)=1
56                  iacum(k,i1)=iacum(k,i1)+2
57                endif
58              endif
59              if(tab(bstate(k),i5).eq.1.and.tab(bstate(k),i6).eq.0
60       &         .and.tabw(iwall(k),i6).eq.0) then
61                call ranu2(iseed,rand,1,icon)
62                if(rand(1).le.mvrate) then
63                  la(k,i5)=0
64                  la(k,i6)=1
65                  iacum(k,i1)=iacum(k,i1)+1
66                endif
67              endif
68     10     continue
69          endif
70        elseif(imvpos.eq.2) then
71          if(MYRANK.eq.NPROCS-1) then          ← ランク max：下境界部
72            do 20 k=numsite_para-ixs*(iwidth+1)+1,numsite_para-ixs*iwidth
73              if(tab(bstate(k),i4).eq.1.and.tab(bstate(k),i1).eq.0
74       &         .and.tabw(iwall(k),i1).eq.0) then
75                call ranu2(iseed,rand(1),1,icon)
76                if(rand(1).le.mvrate) then
77                  la(k,i4)=0
78                  la(k,i1)=1
79                  iacum(k,i1)=iacum(k,i1)+2
80                endif
81              endif
82              if(tab(bstate(k),i3).eq.1.and.tab(bstate(k),i2).eq.0
83       &         .and.tabw(iwall(k),i2).eq.0) then
84                call ranu2(iseed,rand(1),1,icon)
85                if(rand(1).le.mvrate) then
86                  la(k,i3)=0
87                  la(k,i2)=1
88                  iacum(k,i1)=iacum(k,i1)+1
89                endif
90              endif
91     20     continue
92          endif
```

Fig. 5.65 Subroutine mvwall of the parallelized code (2/4)

```
93        elseif(imvpos.eq.3) then                              ← 上下２分割
94          if(MYRANK.le.idiv_point(1)-1) then                  ← 中央ランクより
下位
95            do 30 k=ixs*iwidth+1,ixs*iwidth+numsite
96              if(tab(bstate(k),0).eq.1.and.tab(bstate(k),i1).eq.0
97       &          .and.tabw(iwall(k),i1).eq.0) then
98                call ranu2(iseed,rand(1),1,icon)
99                if(rand(1).le.mvrate) then
100                 la(k,0)=0
101                 la(k,i1)=1
102                 iacum(k,i1)=iacum(k,i1)+1
103               endif
104             endif
105   30      continue
106         elseif(MYRANK.eq.idiv_point(1)) then                  ← 中央ランク
107           do 31 k=ixs*iwidth+1,ixs*iwidth+ixs*idiv_point(2)
108             if(tab(bstate(k),0).eq.1.and.tab(bstate(k),i1).eq.0
109      &          .and.tabw(iwall(k),i1).eq.0) then
110               call ranu2(iseed,rand(1),1,icon)
111               if(rand(1).le.mvrate) then
112                 la(k,0)=0
113                 la(k,i1)=1
114                 iacum(k,i1)=iacum(k,i1)+1
115               endif
116             endif
117   31      continue
118           do 32 k=ixs*iwidth+ixs*idiv_point(2)+1,ixs*iwidth+numsite
119             if(tab(bstate(k),0).eq.1.and.tab(bstate(k),i4).eq.0
120      &          .and.tabw(iwall(k),i4).eq.0) then
121               call ranu2(iseed,rand(1),1,icon)
122               if(rand(1).le.mvrate) then
123                 la(k,0)=0
124                 la(k,i4)=1
125                 iacum(k,i4)=iacum(k,i4)+1
126               endif
127             endif
128   32      continue
```

Fig. 5.65 Subroutine mvwall of the parallelized code (3/4)

```
129         elseif(MYRANK.ge.idiv_point(1)+1) then        ← 中央ランクより
上位
130           do 33 k=ixs*iwidth+1,ixs*iwidth+numsite
131             if(tab(bstate(k),0).eq.1.and.tab(bstate(k),i4).eq.0
132      &          .and.tabw(iwall(k),i4).eq.0) then
133               call ranu2(iseed,rand(1),1,icon)
134               if(rand(1).le.mvrate) then
135                 la(k,0)=0
136                 la(k,i4)=1
137                 iacum(k,i4)=iacum(k,i4)+1
138               endif
139             endif
140    33     continue
141         endif
142       endif
143
144       call rehash(la,astate)
145
146       if(idebug.eq.1) then
147       ierr=0
148       do 87 i=istart,nsite
149       do 88 j=ixs*iwidth+1,ixs*iwidth+numsite
150         if((tab(astate(j),i).lt.0).or.(tab(astate(j),i).gt.1)) then
151           write(6,*) 'After moving wall process'
152           write(6,521) tab(astate(j),i),j,i
153           ierr=-1
154         endif
155    88 continue
156    87 continue
157       endif
158
159   521 Format(1h ,'Error state= ',i10,' at ',i6,' of site',i4)
160
161       return
162       end
```

Fig. 5.65 Subroutine mvwall of the parallelized code (4/4)

```
 1           if(rr.gt.0.and.s.gt.0) then
 2              dirc=mod(s-1,3)
 3              if(dirc.eq.0) then
 4                 dirc=1
 5              else if(dirc.eq.1) then
 6                 dirc=6
 7              else if(dirc.eq.2) then
 8                 dirc=2
 9              endif
10
11              if(br.ne.rr) then
12                 call lrange(xs,ys,rr,site,wall,range)
13                 br=rr
14              else
15                 br=rr
16              endif
```

Fig. 5.66 Part for making the pair-table between lattice nodes in the main routine of the non-parallelized code (phase2)

```
 1          if(irr.gt.0.and.iloop.gt.0) then
 2            idirc=mod(iloop-1,3)
 3            if(idirc.eq.0) then
 4              idirc=1
 5            elseif(idirc.eq.1) then
 6              idirc=6
 7            elseif(idirc.eq.2) then
 8              idirc=2
 9            endif
10 C*************************************
11 C
12 C     CALL LRANGE
13 C
14 C*************************************
15            if(ibr.ne.irr) then
16              call mpi_trans(iwall,ixs*iwidth)
17              call lrange
18              ibr=irr
19            else
20              ibr=irr
21            endif
```

Fig. 5.67 Part for making the pair-table between lattice nodes in the main routine of the parallelized code (phase2)

```
 1       subroutine lrange(xx,yy,r,tsite,twall,trange)
 2
 3       integer xx,yy
 4       integer r,dirc
 5       integer tsite(1:xx*yy,0:6),twall(1:xx*yy,0:6)
 6       integer trange(1:xx*yy,0:6)
 7
 8       integer tpsiteb,tpsitea
 9       integer nsite,itmp
10       integer j,s
11
12       nsite=xx*yy
13
14       call initdat2(xx,yy,trange,0)
15
16 *VOCL LOOP,SCALAR
17       do 10 dirc=1,6
18 *VOCL LOOP,SCALAR
19       do 15 j=1,nsite        ← 分割対象
20          tpsiteb=j
21 *VOCL LOOP,SCALAR
22             do 50 s=1,r
23                itmp=twall(tpsiteb,0)+twall(tpsiteb,dirc)
24                if(itmp.ne.0) then
25                   trange(j,dirc)=0
26                   goto 15
27                endif
28                tpsitea=tsite(tpsiteb,dirc)
29                tpsiteb=tpsitea
30                trange(j,dirc)=tpsiteb
31 50          continue
32 15    continue
33 10    continue
34
35       return
36       end
```

Fig. 5.68 Subroutine lrange of the non-parallelized code

```
 1       subroutine lrange
 2
 3 *include INC_MPI
 4 *include INC_DATA
 5
 6       do k1=istart,nsite
 7       do k2=1,numsite_para
 8         itrange(k2,k1)=0
 9       enddo
10       enddo
11
12       do 10 idirc=1,6
13       do 15 k1=ixs*iwidth+1,ixs*iwidth+numsite  ← 分割後
14         itpsiteb=k1
15         do 20 is=1,irr
16           itmp=tabw(iwall(itpsiteb),0)+tabw(iwall(itpsiteb),idirc)
17           if(itmp.ne.0) then
18             itrange(k1,idirc)=0
19             go to 15
20           endif
21           itpsitea =isite(itpsiteb,idirc)
22           itpsiteb=itpsitea
23           itrange(k1,idirc)=itpsiteb
24    20    continue
25    15 continue
26    10 continue
27
28       return
29       end
```

Fig. 5.69 Subroutine lrange of the parallelized code

```
 1              call initdat2(xs,ys,iacum,0)
 2              call copydata(xs,ys,state,b_state)
 3              call copydata(xs,ys,label,b_label)
 4              call lgcol(xs,ys,rr,range,b_state,a_state,dirc
 5       +      ,iacum,b_label,a_label,nlcol,ierr)
 6              if(ierr.eq.-1) then
 7                  write(6,*) "Error in lgcol "
 8              endif
 9              call copydata(xs,ys,a_state,state)
10              call copydata(xs,ys,a_label,label)
11              do 150 j=1,numsite
12                 acumlg(j,0)=acumlg(j,0)+iacum(j,0)
13                 acumlg(j,1)=acumlg(j,1)+iacum(j,1)
14                 acumlg(j,2)=acumlg(j,2)+iacum(j,2)
15                 acumlg(j,3)=acumlg(j,3)+iacum(j,3)
16                 acumlg(j,4)=acumlg(j,4)+iacum(j,4)
17                 acumlg(j,5)=acumlg(j,5)+iacum(j,5)
18                 acumlg(j,6)=acumlg(j,6)+iacum(j,6)
19  150         continue
20
21  c           /* count Physical Quantity */
22              if(debug.eq.1) then
23              call countPQ(xs,ys,state,numpar,momx,momy,p_num,model)
24              tdirc=dirc+3
25              if(tdirc.gt.6) then
26                 tdirc=tdirc-6
27              endif
28              write(6,*) "lgm (",dirc,",",tdirc,") "
29       +                 ,numpar," (",momx,",",momy,") "
30              write(6,*) (p_num(i),i=start,6)
31              endif
32
33          endif
```

Fig. 5.70 Long-range interaction part in the main routine of the non-parallelized code (phase2)

```
 1         do k1=istart,nsite
 2         do k2=ixs*iwidth+1,ixs*iwidth+numsite
 3           iacum(k2,k1)=0
 4         enddo
 5         enddo
 6 C**************************************
 7 C
 8 C     CALL LGCOL ((astate -> bstate) -> astate)
 9 C
10 C**************************************
11         call mpi_trans(astate,ixs*iwidth)
12         call lgcol(idirc)
13             if(ierr.eq.-1) then
14               write(6,*) 'Error in lgcol on rank:',MYRANK
15             endif
16         do k1=1,numsite_para
17           astate(k1)=bstate(k1)
18         enddo
19         do k1=istart,nsite
20         do k2=ixs*iwidth+1,ixs*iwidth+numsite
21           acumlg(k2,k1)=acumlg(k2,k1)+iacum(k2,k1)
22         enddo
23         enddo
24            if(idebug.eq.1) then
25              call countPQ(astate)
26              call countPQ_sum
27              itdirc=idirc+3
28              if(itdirc.gt.6) then
29                itdirc=itdirc-6
30              endif
31              if(MYRANK.eq.0) then
32                write(6,*) 'lgm (',idirc,',',itdirc,') ',snumpar,' (',
33     &                           smomx,',',smomy,') '
34                write(6,*) (sp_num(i),i=istart,nsite)
35              endif
36            endif
37       endif
```

Fig. 5.71 Long-range interaction part in the main routine of the parallelized code (phase2)

```
 1       subroutine lgcol(xx,yy,r,trange,before,after,dirc,acum
 2      +,blabel,alabel,ncol,ierr)
 3
 4       integer xx,yy,r,ierr
 5       integer dirc,pdirc,odirc
 6       integer acum(1:xx*yy,0:6)
 7       integer trange(1:xx*yy,0:6)
 8       integer before(1:xx*yy,0:6),after(1:xx*yy,0:6)
 9       integer blabel(1:xx*yy,0:6),alabel(1:xx*yy,0:6)
10
11       real    ncol(1:xx*yy)
12
13       integer pcola,pcolb,pcolc,pcold,pcole
14       integer ocola,ocolb,ocolc,ocold,ocole
15       integer ja1,ja2,ja3,ja4
16       integer jb1,jb2,jb3,jb4
17       integer jc1,jc2,jc3,jc4
18       integer jd1,jd2,jd3,jd4
19       integer je1,je2,je3,je4
20       integer ca1,ca2,ca3,ca4,ca5,ca6
21       integer cb1,cb2,cb3,cb4,cb5,cb6
22       integer cc1,cc2,cc3,cc4,cc5,cc6
23       integer cd1,cd2,cd3,cd4,cd5,cd6
24       integer ce1,ce2,ce3,ce4,ce5,ce6
25       integer nsite
26       integer ii,i,k,s,jj,itmp
27
28 c     REAL*8 tg1,tg2,tga,tgb
29
30 c     CALL gettod(tga)
31
32       nsite=xx*yy
33       ierr=0
34
35       call copydata(xx,yy,before,after)
36       call copydata(xx,yy,blabel,alabel)
37       call initdat2(xx,yy,acum,0)
38
39 c     CALL gettod(tg1)
40 c     write(6,*) "     ",(tg1-tga)*1.0D-6
41
```

Fig. 5.72 Subroutine lgcol of the non-parallelized code (1/2)

```
42        do 60 ii=1,2
43           if(ii.eq.1) then
44              i=dirc
45           else if(ii.eq.2) then
46              i=dirc+3
47              if(i.gt.6) then
48                 i=i-6
49              endif
50           endif
51           pdirc=i
52           odirc=i+3
53           if(odirc.gt.6) then
54              odirc=odirc-6
55           endif
56
57           do 20 k=1,nsite      ← 分割対象
58
59                 .
60                 .
61                 .
62                 .
63                 .
64                 .
65
66
67              endif
68 20        continue
69 60     continue
70
71 c      CALL gettod(tgb)
72 c      write(6,*) "    ",(tgb-tg1)*1.0D-6
73 c      write(6,*) "stotal",(tgb-tga)*1.0D-6
74
75        do 145 i=0,6
76        do 145 j=1,nsite
77           if((after(j,i).lt.0).or.(after(j,i).gt.1)) then
78              write(6,*) "After liquid_gas"
79              write(6,521) after(j,i),j,i
80              ierr=-1
81           endif
82 145    continue
83
84 521    format(1h ,"Error state= ",i10," at ",i6," of site",i4)
85
86        return
87        end
```

Fig. 5.72 Subroutine lgcol of the non-parallelized code (2/2)

```
 1       subroutine lgcol(idirc)
 2
 3       integer*4 idirc
 4
 5 *include INC_MPI
 6 *include INC_DATA
 7
 8       integer*4 ipdirc,iodirc
 9       integer*4 pcola,pcolb,pcolc,pcold,pcole
10       integer*4 ocola,ocolb,ocolc,ocold,ocole
11       integer*4 ja1,ja2,ja3,ja4
12       integer*4 jb1,jb2,jb3,jb4
13       integer*4 jc1,jc2,jc3,jc4
14       integer*4 jd1,jd2,jd3,jd4
15       integer*4 je1,je2,je3,je4
16       integer*4 ca1,ca2,ca3,ca4,ca5,ca6
17       integer*4 cb1,cb2,cb3,cb4,cb5,cb6
18       integer*4 cc1,cc2,cc3,cc4,cc5,cc6
19       integer*4 cd1,cd2,cd3,cd4,cd5,cd6
20       integer*4 ce1,ce2,ce3,ce4,ce5,ce6
21       integer*4 ii,i,s,jj,itmp
22
23       do k1=istart,nsite
24       do k2=1,numsite_para
25         la(k2,k1)=0
26         la(k2,k1)=tab(astate(k2),k1)
27       enddo
28       enddo
29
30       do 60 ii=1,2
31
32         if(ii.eq.1) then
33           i=idirc
34         elseif(ii.eq.2) then
35           i=idirc+3
36           if(i.gt.6) then
37             i=i-6
38           endif
39         endif
40
```

Fig. 5.73 Subroutine lgcol of the parallelized code (1/2)

```
41          ipdirc=i
42          iodirc=i+3
43          if(iodirc.gt.6) then
44            iodirc=iodirc-6
45          endif
46
47          do 20 k1=ixs*iwidth+1,ixs*iwidth+numsite  ← 分割後
48
49                  .
50                  .
51                  .
52                  .
53                  .
54                  .
55
56
57     20   continue
58     60 continue
59
60        call rehash(la,bstate)
61
62        if(idebug.eq.1) then
63        ierr=0
64          do 145 k1=istart,nsite
65          do 146 k2=ixs*iwidth+1,ixs*iwidth+numsite
66            if((tab(bstate(k2),k1).lt.0).or.(tab(bstate(k2),k1).gt.1))
67       &then
68                write(6,*) 'After liquid_gas'
69                write(6,521) tab(bstate(k2),k1),k2,k1
70                ierr=-1
71              endif
72     146   continue
73     145   continue
74        endif
75
76     521 format(1h ,'Error state= ',i10,' at ',i6,' of site',i4)
77
78        return
79        end
```

Fig. 5.73 Subroutine lgcol of the parallelized code (2/2)

```
 1           if(mod(s,frate).eq.0) then
 2              call outfiles(xs,ys,n,state,acumtr,acumlg,acumbo,model)
 3              write(6,*) "---------------------- write file ",n
 4              n=n+1
 5           endif
 6  700      format(i4,i4,i6,f10.3,f10.3)
 7
 8  c        /***** output total pressure ***************/
 9           if(mod(s,trate).eq.0.and.s.ge.0) then
10              pp=0.0
11              do 180 k=1,numsite
12                  pp=pp+real(acumtr(k,1)+acumtr(k,2)+acumtr(k,3)
13       +                 +acumtr(k,4)+acumtr(k,5)+acumtr(k,6))
14       +           +0.5*real(acumlg(k,1)+acumlg(k,2)+acumlg(k,3)
15       +                    +acumlg(k,4)+acumlg(k,5)+acumlg(k,6))
16  180         continue
17              pp=0.5*pp/real(trate*xs*ys)
18              write(87,*) s,pp
19           endif
20
21           if(mod(s,trate).eq.0.and.s.ge.0) then
22              call initdat2(xs,ys,acumtr,0)
23              call initdat2(xs,ys,acumlg,0)
24              call initdat2(xs,ys,acumbo,0)
25           endif
26
27  60     continue
```

Fig. 5.74 Data output part in the main routine of the non-parallelized code (phase2)

```
 1          if(mod(iloop,ifrate).eq.0) then
 2  C****************************************
 3  C
 4  C     CALL OUTFILES
 5  C
 6  C****************************************
 7            call outfiles(bstate,n)
 8            n=n+NPROCS+MYRANK
 9          endif
10
11          if(mod(iloop,itrate).eq.0.and.iloop.ge.0) then
12            pp=0.0
13            do 180 k1=ixs*iwidth+1,ixs*iwidth+numsite
14              pp=pp+real(acumtr(k1,1)+acumtr(k1,2)+acumtr(k1,3)
15       &            +acumtr(k1,4)+acumtr(k1,5)+acumtr(k1,6))
16       &        +0.5*real(acumlg(k1,1)+acumlg(k1,2)+acumlg(k1,3)
17       &        +acumlg(k1,4)+acumlg(k1,5)+acumlg(k1,6))
18     180      continue
19              call pp_sum
20              if(MYRANK.eq.0) then
21                spp=0.5*spp/real(itrate*ixs*iyss)
22                write(87,*) iloop,spp
23              endif
24              do k1=istart,nsite
25              do k2=1,numsite_para
26                acumtr(k2,k1)=0
27                acumlg(k2,k1)=0
28                acumbo(k2,k1)=0
29              enddo
30              enddo
31            endif
32
33       60 continue
```

Fig. 5.75 Data output part in the main routine of the parallelized code (phase2)

```
 1        subroutine pp_sum
 2
 3  *include INC_MPI
 4  *include INC_DATA
 5
 6        call mpi_barrier(mpi_comm_world,ierr)
 7
 8        call mpi_reduce(pp, spp, 1, mpi_real,
 9       &mpi_sum,0,mpi_comm_world,ierr)
10
11        call mpi_barrier(mpi_comm_world,ierr)
12
13        return
14        end
```

Fig. 5.76 Subroutine pp_sum of the parallelized code

```
 1 c      /**** count Physical Quantity ****/
 2        write(6,*) " last step ",s
 3        call countPQ(xs,ys,state,numpar,momx,momy,p_num,model)
 4        write(99,*) "last  ",numpar," (",momx,",",momy,") "
 5       +,(p_num(i),i=start,6)
 6        write(6,*) "last  ",numpar," (",momx,",",momy,") "
 7       +,(p_num(i),i=start,6)
 8
 9        write(99,*) "x ",xs
10        write(99,*) "y ",ys
11        write(99,*) "dd ",den
12        write(99,*) "opt ",opt
13        write(99,*) "step ",step
14        write(99,*) "frate ",frate
15        write(99,*) "trate ",trate
16        write(99,*) "rr ",rr
17        write(99,*) "model ",model
18
19        write(6,*) "x ",xs
20        write(6,*) "y ",ys
21        write(6,*) "dd ",den
22        write(6,*) "opt ",opt
23        write(6,*) "step ",step
24        write(6,*) "frate ",frate
25        write(6,*) "trate ",trate
26        write(6,*) "rr ",rr
27
28 4       format(1h ,"init ",i7," (",i7,",",i7,") ")
29
30        stop
31        end
```

Fig. 5.77 Final process part in the main routine of the non-parallelized code

```
 1       if(MYRANK.eq.0) then
 2         write(6,*) 'last step ',iloop
 3       endif
 4
 5       call countPQ(bstate)
 6       call countPQ_sum
 7       if(MYRANK.eq.0) then
 8         write(99,*) 'last  ',snumpar,' (',smomx,',',smomy,') ',
 9      &              (sp_num(i),i=istart,nsite)
10         write(6,*)  'last  ',snumpar,' (',smomx,',',smomy,') ',
11      &              (sp_num(i),i=istart,nsite)
12       endif
13
14       if(MYRANK.eq.0) then
15         write(99,*) 'ixs ',ixs
16         write(99,*) 'iys ',iys
17         write(99,*) 'den ',den
18         write(99,*) 'iopt ',iopt
19         write(99,*) 'istep ',istep
20         write(99,*) 'ifrate ',ifrate
21         write(99,*) 'itrate ',itrate
22         write(99,*) 'irr ',irr
23         write(99,*) 'model ',model
24
25         write(6,*) 'ixs ',ixs
26         write(6,*) 'iys ',iys
27         write(6,*) 'den ',den
28         write(6,*) 'iopt ',iopt
29         write(6,*) 'istep ',istep
30         write(6,*) 'ifrate ',ifrate
31         write(6,*) 'itrate ',itrate
32         write(6,*) 'irr ',irr
33       endif
34
35     4 format(1h ,"init ",i7," (",i7,",",i7,") ")
36
37       call mpi_barrier(mpi_comm_world,ierr)
38       call mpi_finalize(ierr)
39
40       stop
41       end
```

Fig. 5.78 Final process part in the main routine of the parallelized code

6. ベクトル化

今回，並列化後にベクトル化も行った。ベクトル化作業では，最初に動的挙動解析ツール ANALYZER を用い，並列化コードの各ルーチンのベクトル化状況とコスト分布を確認した。ANALYZER は，VPP500 システムの一部である GSP 上で動作するツールであり，解析結果として，コード全体に対する各サブルーチンのコストや各サブルーチンに存在する各ループのコストを出力する。ANALYZER が動作する計算機は VPP500 ではなく GSP であるが，ベクトル化状況やコスト分布を把握する上で，VPP500 と GSP は大差はない。

並列化（ベクトル化前）コードの ANALYZER の結果を Table 6.1 に示す。これにおいて，コストが高い（Table 6.1の v-cost 欄）サブルーチンは上から 3 つあげると，fhp3，mvwall，lgcol である。しかし，fhp3 と lgcol はベクトル化効果（Table 6.1の v-effect 欄）として約 36 ～ 55 倍出ている。これらの他のサブルーチンは同じく 約 40 ～ 55 倍出ているか，または実行時に一回しか実行されないサブルーチンであり，チューニング対象から外すことにした。よって，倍率の低いサブルーチン mvwall のみをベクトル化の対象とした。

ベクトル化は，ベクトルコンパイラの自動ベクトル化機能によって行われるが，これによるベクトル化の対象は DO ループ（多重ループの場合はその最内ループ）である。Fig.5.65 においてベクトル化の対象となるのは，49 ～ 68 行目，72 ～ 91 行目，95 ～ 105 行目，107 ～ 117 行目，118 ～ 128 行目，130 ～ 140 行目の各 DO ループであるが，コンパイラによってベクトル化されていない。これは，ループ中において乱数発生ルーチン ranu2 の呼び出しを行っていることが原因である。

ranu2 は，実引数に必要な乱数の個数を指定すると，その個数の乱数を求める。ベクトル化対象のループ中からループの直前へ ranu2 の呼び出しを移し，直後のループで使用される個数分の乱数生成を行うと共に，直後のループのベクトル化を行った。この方法は，上であげた DO ループすべてに対し同様に適用できる。この結果，使用する乱数の順序がベクトル化前の乱数と異なるがシミュレーション上は問題無い。

・ Fig.5.65 の 50 ～ 58 行目の if ブロックのベクトル化

Fig.6.1 において 2 ～ 7 行目で ranu2 に渡す乱数の個数をカウントする。この部分のループ回転数と if 文の条件文は元のループのそれと同様である。8 行目で ranu2 を呼び出しカウント数 icount1 を実引数として渡す。icount1 個の乱数は，icount1 個の要素を格納できる配列に求まる。そこで，新たに配列 rand_tmp を用意した。ただし，この配列によるメモリサイズ増加を避けるため，他のルーチンで使用されていて，かつ本ルーチンで更新しても影響のない配列とメモリ共有させることにした。メモリ共有される配列は iduma を選び，配列 rand_tmp をこれと同じサイズで宣言し equivalence 宣言によってメモリ共有を行った。宣言サイズは，iduma と同じであり本ルーチンで乱数を格納する上で全く問題無いサイズである。10 ～ 20 行目は，元のループと構成は同様である。異なるのは，ranu2 を呼び出す代わりに，配列 rand_tmp の

内容を，添字をインクリメントしながら参照するのみである。

・Fig.5.65 の 59 ～ 67 行目の if ブロックのベクトル化

この部分も方法は，50 ～ 58 行目の if ブロックのベクトル化と同様である。Fig.6.1 の 23 ～ 28 行目において ranu2 に渡す乱数の個数をカウントし，これを 29 行目において ranu2 に渡す。使用する配列も同様に rand_tmp である。31 ～ 41 行目は 10 ～ 20 行目と同様である。

以上のベクトル化と同様のベクトル化を他のすべてのループに施した。ベクトル化後の ANALYZER の結果を Table 6.2 に示す。サブルーチン mvwall はベクトル化効果として 40.3 ～ 54.6 倍の効果を得ることができた。

Table 6.1 Distribution of computational cost before vectorization

```
vectorize - total list ------------------------------------------------------
name      ex-count   v-cost    %   s-cost    %  v-leng  v-rate    v-effect
FHP3          1200 .9598E10 53.3 .3947E12 61.0   73637    99.7  36.0- 47.0
MVWALL        1200 .3188E10 17.7 .1873E11  2.9   55408    84.8   5.8-  6.0
LGCOL         1200 .2387E10 13.2 .1117E12 17.3   67501   100.0  40.3- 54.6
LGM_MAIN         1 .1342E10  7.4 .6281E11  9.7   79410   100.0  40.3- 54.6
CHWALL        1200 .5033E09  2.8 .2355E11  3.6   69232   100.0  40.3- 54.6
REHASH        6001 .4040E09  2.2 .1890E11  2.9   45001   100.0  40.3- 54.6
TRANS         1200 .3585E09  2.0 .1677E11  2.6   75001   100.0  40.3- 54.6
LRANGE           1 .2026E09  1.1 .2069E09  0.0      17     2.1   1.0-  1.0
COUNTPQ          1  8614645  0.0 15694583  0.0   65456    46.1   1.8-  1.8
READDATA         1  6493885  0.0 11250052  0.0   73637    43.2   1.7-  1.7
                          .
                          .
                          .
                          .
```

Table 6.2 Distribution of computational cost after vectorization

```
vectorize - total list ------------------------------------------------------
name      ex-count   v-cost    %   s-cost    %  v-leng  v-rate    v-effect
FHP3          1200 .9587E10 62.7 .3942E12 60.6   73637    99.7  36.0- 47.0
LGCOL         1200 .2386E10 15.6 .1117E12 17.2   67501   100.0  40.3- 54.6
LGM_MAIN         1 .1342E10  8.8 .6281E11  9.7   79410   100.0  40.3- 54.6
CHWALL        1200 .5032E09  3.3 .2355E11  3.6   69232   100.0  40.3- 54.6
MVWALL        1200 .4690E09  3.1 .2194E11  3.4   56269   100.0  40.3- 54.6
REHASH        6001 .4040E09  2.6 .1890E11  2.9   45001   100.0  40.3- 54.6
TRANS         1200 .3591E09  2.3 .1680E11  2.6   75001   100.0  40.3- 54.6
LRANGE           1 .2026E09  1.3 .2069E09  0.0      17     2.1   1.0-  1.0
COUNTPQ          1  8614645  0.1 15694583  0.0   65456    46.1   1.8-  1.8
READDATA         1  6493885  0.0 11250052  0.0   73637    43.2   1.7-  1.7
                          .
                          .
                          .
                          .
```

```
 1                icount1=0
 2                do 200 k1=ixs*iwidth+1,ixs*iwidth+ixs
 3                  if(tab(astate(k1),i4).eq.1.and.tab(astate(k1),i1).eq.0
 4           &         .and.tabw(iwall(k1),i1).eq.0) then
 5                    icount1=icount1+1
 6                  endif
 7         200      continue
 8                  call ranu2(iseed,rand_tmp(1),icount1,icon)
 9                  icount1=0
10                  do 300 k1=ixs*iwidth+1,ixs*iwidth+ixs
11                    if(tab(astate(k1),i4).eq.1.and.tab(astate(k1),i1).eq.0
12           &           .and.tabw(iwall(k1),i1).eq.0) then
13                      icount1=icount1+1
14                      if(rand_tmp(icount1).le.mvrate) then
15                        la(k1,i4)=0
16                        la(k1,i1)=1
17                        iacum(k1,i1)=iacum(k1,i1)+2
18                      endif
19                    endif
20         300      continue
21       C---------
22                  icount1=0
23                  do 201 k1=ixs*iwidth+1,ixs*iwidth+ixs
24                    if(tab(astate(k1),i5).eq.1.and.tab(astate(k1),i6).eq.0
25           &           .and.tabw(iwall(k1),i6).eq.0) then
26                      icount1=icount1+1
27                    endif
28         201      continue
29                  call ranu2(iseed,rand_tmp(1),icount1,icon)
30                  icount1=0
31                  do 301 k1=ixs*iwidth+1,ixs*iwidth+ixs
32                    if(tab(astate(k1),i5).eq.1.and.tab(astate(k1),i6).eq.0
33           &           .and.tabw(iwall(k1),i6).eq.0) then
34                      icount1=icount1+1
35                      if(rand_tmp(icount1).le.mvrate) then
36                        la(k1,i5)=0
37                        la(k1,i6)=1
38                        iacum(k1,i1)=iacum(k1,i1)+1
39                      endif
40                    endif
41         301      continue
```

Fig. 6.1 Partially vectorized subroutine mvwall

7. 並列化の評価

ここでは，並列化を行ったコードについて，実行結果の検証及び，計算規模と実行時間の評価を行う。

7.1 実行結果の検証

並列化コードについて，VPP500上でシングルコードと並列化コードの実行結果の比較を行った。比較は，シミュレーションパラメータを様々に変更しながらロードモジュールを作成し，それぞれの実行結果を用いて行った。ただし，本コードでは乱数が用いられているため，並列実行時にはシングルコードの乱数とは異なることから実行結果も異なる。よって，結果が乱数の違いのみによるものかどうかを把握することが必要である。

シングルコードと並列化コードを比較して，乱数の発生方法が異なるのはサブルーチン mvwall である。これは，ベクトル化によって乱数発生順序が変更された部分である。そこで，サブルーチン mvwall をベクトル化前のものに戻して乱数の発生方法を同一にし，実行結果の比較を行った。

その結果，シングルコードと並列化コードの 1 プロセスのみの実行結果は完全一致した。一方，並列化コードの並列実行では，プロセス毎に異なる乱数を発生させていることから，シングルコードとは結果は一致しない。そこで，結果の違いが乱数の影響のみであるか否かを確認するために，シングルコードで発生させた乱数をファイルに保存し，これを並列実行で用いる方法で結果の確認を行った。その結果，実行結果は完全一致した。よって，シングルコードと並列化コードの結果の違いは，乱数のみが原因であることが判明し，並列実行結果は妥当であることを確認した。同時に，mvwall を除いた部分の実行結果は一致することを確認した。

また，ベクトル化後の mvwall を使用すると，乱数の発生方法がシングルコードと異なるため，並列化コードの 1 プロセスのみを用いた実行も，シングルコードと異なる。しかし，実行結果には特別な違いが見られなかった。

7.2 計算規模

今回の並列化作業において，本コードのメモリ使用サイズ／1PE を削減するために，データ分割を採用した並列化を行った。これにより，コードで使用可能なメモリサイズが，並列実行時に使用する PE 台数の約全メモリサイズまで使用可能になる。しかし，実際にコードをコンパイルして作成したロードモジュールには，コードで使用するデータ（配列や変数）以外のメモリ（ユーザからは不明である）がいくらか付加されるため，ジョブ投入の際に使用可能とされているメモリ量をすべてコード中のデータで使用することはできない。そこで，実際にコードで使用する配列等のサイズをどの位まで指定すれば実行不可能になるか判断するため，実際にコンパイ

ルして作成したロードモジュール自体のメモリ使用量とこのロードモジュールをジョブ投入した場合の実行可否を VPP500 と AP3000 で調査した。メモリ使用量については，それぞれの計算機上に用意されている size コマンドで調査し，実行可否は実際にジョブ投入を行って調査した。この結果，VPP500 上ではロードモジュール自体のメモリ使用量が 185MB/1PE まで，AP3000 上では 1844MB/1PE までで限界であった。

この結果を基にして，本コードの計算規模を調査すると，シングルコードでは，VPP500 で 2 次元領域サイズが 694×694 までで限界であったのが，並列化コードの 1PE で 812×812 まで，4PE で 1620×1620 まで，16PE では 3232×3232 までのサイズを用いることが可能になった。これより，ほぼ PE 台数倍のサイズが使用可能になったことがわかる。また，シングルコードと並列化コードの 1PE のサイズが異なるのは，今回の高速化をする上で必要のない配列等を削除したためである。一方，AP3000 では並列化コードの 1PE で 2948×2948 まで，2PE で 4168×4168 まで，4PE では 5894×5894 までのサイズを用いることが可能になった。

これらは，分割領域の袖部が 1 の場合の結果であるが，袖部が 1 より大きい場合も同様に，VPP500 では 185MB/1PE，AP3000 では 1844MB/1PE を基準にして size コマンドで指定可能な空間サイズを知ることができる。

7.3 実行時間

並列化コードについて，シングルコードからの性能向上比を調査するため，実行時間測定を行った。測定は，次のシミュレーションパラメータと測定方法で行った。

(1) シミュレーションパラメータ

・固定のパラメータ

```
steps=1200, frate=1200, trate=300, ixs=300, iyss=300, model=3, den=0.1,
ioptw=2(上下に平行の壁),
imvpos=1, imvdirc=1, mvrate=0.1
```

・変更するパラメータ

irr=(0 or 9 or 15) － 計 3 通り。

ここでの変数の意味は，以下の通りである。

- steps: ステップ数
- frate: 結果を出力するステップ数
- trate: 時間平均をとるステップ数
- ixs: x 方向の格子サイズ

- iyss: y 方向の格子サイズ
- model: 使用するモデルの選択
- den: 系の粒子密度
- ioptw: 使用する壁の種類
- imvpos: 運動量を付加する位置
- imdirc: 運動量を付加する方向
- mvrate: 運動量を付加する割合
- irr: 長距離相互作用の距離

(2) 測定方法

フェーズ 1 のデータ宣言部，初期化部，粒子の初期配置部は，1 回のシミュレーションで，最初に一度呼ばれるだけであり，その他のサブルーチンは フェーズ 2 でも用いられているので，測定は，フェーズ 2 に対してのみ行った。つまり，フェーズ 2 部分の時間発展ループとこのループ中で呼ばれる各サブルーチン（mpi_trans, trans, chwall, mvwall, lrange, lgcol, fhp3）について，ループ 1 回毎に各実行時間を測定し，ループ回転数 300 回毎にそれらの平均を取った。各実行時間には，実時間を取得できるルーチンを用いて該当部分を挟み，その差を採用した。Fig.7.1 に VPP500 上で実時間を取得できるサービスルーチン gettod を用いた時の実行時間取得の例を示す。AP3000 上には，サービスルーチン gettod は用意されていないため，C 言語により gettod 関数 [9] を作成し，これを用いた。これを Fig.7.2 に示す。また，DIGITAL Personal Workstation 上での実行では，計算機システムに用意されている実時間を取得できるサブルーチンライブラリ secnds [10] を用いた。

・実行プロセス数

すべての計算機において，シングルコードは 1 プロセスのみの実行を，並列化コードは 1 ～ 4 プロセスの各実行を行った。以下の Table では，各々の実行において，600 ステップ後に 300 ステップでステップ平均を取った値を表示してある。また，複数プロセスの場合は，rank 0 の時間を表示した。なお Table の値の単位は秒である。

(1) VPP500

irr=0 の時の実行時間計測結果を Table 7.1 に，irr=9 の時の実行時間計測結果を Table 7.2 に，irr=15 の時の実行時間計測結果を Table 7.3 に示す。

これらによると，データ転送サブルーチン mpi_trans を除く各サブルーチンは，ほぼ実行時に使用したプロセス数倍の効率を得ている。一部でプロセス数倍ではない場合が見られるが，測定値が実時間であることからこの誤差であると考えられる。

(2) AP3000

irr=0 の時の実行時間計測結果を Table 7.4 に，irr=9 の時の実行時間計測結果を Table 7.5 に，irr=15 の時の実行時間計測結果を Table 7.6 に示す。

これらによると，AP3000 でも，VPP500 と同様にほぼ実行時に使用したプロセス数倍の効率を得ていることが分かる。

(3) DIGITAL Personal Workstation 600au （ワークステーションクラスタ：WSC)

irr=0 の時の実行時間計測結果を Table 7.7 に，irr=9 の時の実行時間計測結果を Table 7.8 に，irr=15 の時の実行時間計測結果を Table 7.9 に示す。

これらによると，並列実行においては，プロセス数倍以上の倍率を得ている場合も見られる。これらは，CPU 自体の性能が高いことと，メモリアクセス速度（特にキャッシュの効率的利用）が原因であると考えれる。

Table 7.1 Comparison of the calculation time for the different number of processors on VPP500 (irr=0)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	9.5e-02	0.18	2.7e-03	0.0	0.0	0.76	1.06
1PE	3.4e-04	1.7e-02	3.2e-02	1.1e-02	0.0	0.0	0.61	0.69
2PE	1.8e-03	8.6e-03	1.6e-02	5.7e-03	0.0	0.0	0.31	0.35
3PE	2.5e-03	5.7e-03	1.0e-02	3.9e-03	0.0	0.0	0.20	0.23
4PE	2.5e-03	4.3e-03	8.2e-03	3.0e-03	0.0	0.0	0.15	0.17

Table 7.2 Comparison of the calculation time for the different number of processors on VPP500 (irr=9)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	9.5e-02	0.17	2.9e-03	0.0	0.19	0.77	1.26
1PE	6.7e-04	1.8e-02	3.4e-02	1.2e-02	0.0	0.13	0.63	0.85
2PE	1.2e-02	9.1e-03	1.7e-02	6.9e-03	0.0	6.6e-02	0.31	0.43
3PE	2.0e-02	5.8e-03	1.1e-02	4.7e-03	0.0	4.2e-02	0.20	0.29
4PE	2.2e-02	4.3e-03	8.8e-03	3.5e-03	0.0	2.9e-02	0.15	0.22

Table 7.3 Comparison of the calculation time for the different number of processors on VPP500 (irr=15)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	9.5e-02	0.18	2.9e-03	0.0	0.19	0.77	1.26
1PE	6.7e-04	1.9e-02	3.6e-02	1.3e-02	0.0	0.14	0.64	0.87
2PE	2.3e-03	9.9e-03	1.9e-02	8.1e-03	0.0	7.3e-02	0.32	0.44
3PE	2.2e-02	6.2e-03	1.3e-02	5.3e-03	0.0	4.6e-02	0.21	0.31
4PE	1.5e-02	4.6e-03	1.0e-02	4.5e-03	0.0	3.4e-02	0.15	0.23

Table 7.4 Comparison of the calculation time for the different number of processors on AP3000(irr=0)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	0.11	0.15	6.6e-02	0.0	0.0	1.78	2.67
1PE	3.3e-04	5.8e-02	5.5e-02	5.3e-02	0.0	0.0	1.09	1.37
2PE	1.5e-02	2.2e-02	2.3e-02	2.5e-02	0.0	0.0	0.53	0.68
3PE	1.5e-02	1.4e-02	1.3e-02	1.3e-02	0.0	0.0	0.35	0.44
4PE	1.9e-02	9.9e-03	1.1e-02	5.9e-03	0.0	0.0	0.26	0.33

Table 7.5 Comparison of the calculation time for the different number of processors on AP3000(irr=9)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	0.10	0.15	6.8e-02	0.0	0.56	1.78	3.37
1PE	7.1e-04	5.9e-02	5.5e-02	5.2e-02	0.0	0.27	1.08	1.69
2PE	2.1e-02	2.7e-02	2.6e-02	2.3e-02	0.0	0.13	0.54	0.85
3PE	2.8e-02	1.7e-02	1.6e-02	1.1e-02	0.0	9.2e-02	0.36	0.57
4PE	2.3e-02	1.1e-02	1.1e-02	1.0e-02	0.0	6.6e-02	0.26	0.42

Table 7.6 Comparison of the calculation time for the different number of processors on AP3000(irr=15)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	9.7e-02	0.15	6.6e-02	0.0	0.56	1.78	3.37
1PE	7.0e-04	5.8e-02	5.2e-02	5.2e-02	0.0	0.29	1.08	1.70
2PE	3.6e-02	2.7e-02	2.7e-02	2.3e-02	0.0	0.13	0.54	0.88
3PE	2.7e-02	1.8e-02	1.7e-02	1.3e-02	0.0	9.5e-02	0.36	0.58
4PE	2.9e-02	1.1e-02	1.2e-02	1.2e-02	0.0	6.8e-02	0.26	0.44

Table 7.7 Comparison of the calculation time for the different number of processors on WSC(irr=0)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	9.1e-02	0.12	6.5e-02	0.0	0.0	0.66	1.45
1PE	2.0e-04	5.3e-02	4.0e-02	4.5e-02	0.0	0.0	0.45	0.67
2PE	2.1e-03	2.2e-02	1.6e-02	1.9e-02	0.0	0.0	0.22	0.31
3PE	5.6e-03	1.2e-02	9.6e-03	1.1e-02	0.0	0.0	0.14	0.20
4PE	1.7e-02	9.2e-03	6.7e-03	7.0e-03	0.0	0.0	0.10	0.16

Table 7.8 Comparison of the calculation time for the different number of processors on WSC(irr=9)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	8.2e-02	0.12	6.5e-02	0.0	0.16	0.65	1.73
1PE	3.1e-04	5.3e-02	4.0e-02	4.5e-02	0.0	9.1e-02	0.44	0.79
2PE	1.4e-02	2.3e-02	1.9e-02	2.3e-02	0.0	4.4e-02	0.21	0.40
3PE	1.3e-02	1.6e-02	1.2e-02	1.3e-02	0.0	3.0e-02	0.14	0.27
4PE	1.2e-02	1.0e-02	7.8e-03	8.9e-03	0.0	2.1e-02	0.10	0.19

Table 7.9 Comparison of the calculation time for the different number of processors on WSC(irr=15)

	mpi_trans	trans	chwall	mvwall	lrange	lgcol	fhp3	loop
single	0.0	7.6e-02	0.12	6.5e-02	0.0	0.16	0.64	1.72
1PE	5.2e-04	5.2e-02	4.0e-02	4.5e-02	0.0	9.0e-02	0.44	0.79
2PE	1.9e-02	2.5e-02	2.0e-02	2.4e-02	0.0	4.5e-02	0.21	0.41
3PE	2.2e-02	1.6e-02	1.1e-02	1.2e-02	0.0	3.0e-02	0.14	0.28
4PE	7.4e-02	1.0e-02	9.6e-03	1.3e-02	0.0	2.2e-02	0.10	0.26

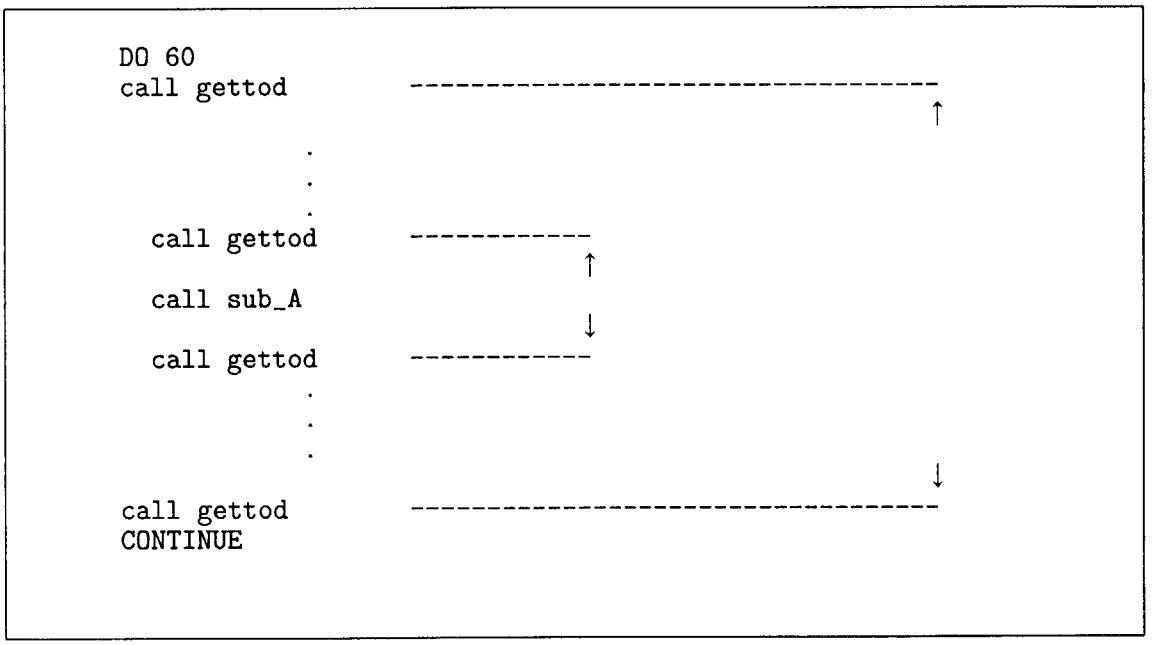

Fig. 7.1 Process of obtaining the calculation time on VPP500

```
#include <sys/types.h>
#include <sys/times.h>

void gettod_(d0)
double *d0;
{
 *d0 = gethrtime();
 return;
}
```

Fig. 7.2 Function gettod for obtaining the calculation time on AP3000

8. おわりに

今回，格子ガス気液モデルのシミュレーションコードの並列化を行った。並列化は，より大きな2次元格子空間を用いたシミュレーションに対応できるように，メモリ分散による並列化を行い，これにともなって処理分割も行った。

これにより，2次元格子空間のサイズは，VPP500では，シングルコードにおいて694 x 694までが限界であったのに対し，最大で3232 x 3232（16PE使用時）までのサイズを用いることが可能になった。AP3000では，2948 x 2948までが限界であったのに対し，最大で5849 x 5849（4PE使用時）までのサイズを用いることが可能になった。このような大規模サイズを用いることができるようになったのは，各データの依存関係が比較的簡潔であったために，大部分のデータを容易に分割可能であったことによる。更に，データ分割にともない処理分割も行われるので，主計算部の大部分のループを分割することができ，計算時間についても，VPP500とAP3000において，ほぼプロセス数倍の効率を得ることができた。

また，本コードは，MPIライブラリを用いて並列化を行ったため，各計算機への移植性のよいコードとなった。

謝　　辞

本報告書をまとめるにあたり，有益な助言を頂きました計算科学技術推進センター並列処理基本システム開発グループ 樋口健二氏に感謝致します。また，本報告書をまとめる機会を下さいました計算科学技術推進センター長 秋元 正幸 氏，計算科学技術推進センター管理課長 藤井 実氏に感謝致します。

参考文献

[1] 渡辺正，海老原健一，加藤克海；ワークステーションクラスターによる格子ガス二相流シミュレーションコードの並列計算，JAERI-Data/Code 99-029，1999年5月.

[2] 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年7月.

[3] J. Hardy and Y. Pomeau, *J. Math. Phys.* **13**, 1042 (1972); J. Hardy, Y. Pomeau and O. de Pazzis, *J. Math. Phys.* **14**, 1746 (1973); *Phys. Rev.* **A13**, 1949 (1976).

[4] U. Frisch, B. Hasslacher and Y. Pomeau, *Phys. Rev. Let.* **56**, 1505 (1986).

[5] U. Frisch, D. d'Humières, B. Hasslacher, P. Lallemand, Y. Pomeau and J. Rivet, *Complex Systems* **1**, 649 (1987).

[6] C. Appert and S. Zaleski, *Phys. Rev. Lett.* **64**, 1 (1990). 97-056，1998年1月.

[7] MPI:A Message-Passing Interface Standard, Message Passing Interface Forum, May 10, 1996(http://www.mpi-forum.org).

[8] UXP/V MPI使用手引書 V10用，富士通株式会社，1996年12月.

[9] AP3000システム利用手引，日本原子力研究所 計算科学技術推進センター 情報システム管理課，1997年4月.

[10] DEC Fortran90 リファレンスマニュアル，日本ディジタルイクイップメント株式会社（現COMPAQ社），1995年7月.

付録1 今回の並列化で使用したMPIライブラリ

・MPIで用いる言葉の説明

コミュニケータ　：通信空間を規定し通信操作の適切なスコープを定めるもの.

グループ　：順序付けされたプロセスの集合.

ランク　：n個のプロセスから成るグループ内における，各プロセスに0～(n-1)のプロセスを識別するために割り当てられた番号.

・mpi_init(ierr)

機能　：MPI環境の初期化処理を行う.

ierr　：エラーコード

・mpi_comm_size(comm, size, ierr)

機能　：コミュニケータの中のプロセス数を返す.

comm　：コミュニケータ

size　：コミュニケータのグループ内のプロセス数

ierr　：エラーコード

・mpi_comm_rank(comm, rank, ierr)

機能　：呼び出しプロセスのコミュニケータ内のランクを返す.

comm　：コミュニケータ

rank　：コミュニケータのグループ内の呼び出しプロセスのランク

ierr　：エラーコード

・mpi_comm_finalize(ierr)

機能　：MPI環境の終了処理を行う.

ierr　　：エラーコード

・mpi_barrier(comm, ierr)

機能　　：グループ内のすべてのプロセスによって呼び出されるまで，呼び出し元のプロセスをブロックする．同期をとる．

comm　　：コミュニケータ

ierr　　：エラーコード

・mpi_sendrecv(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status, ierr)

機能　　：１対１（1プロセス対1プロセス）のデータの送受信を行う．

sendbuf　　：送信バッファの開始アドレス

sendcount　　：送信バッファ内の要素数

sendtype　　：送信バッファのデータ型

dest　　：送信先のランク

sendtag　　：送信タグ

recvbuf　　：受信バッファの開始アドレス

recvcount　　：受信バッファ内の要素数

recvtype　　：受信バッファのデータ型

source　　：送信元のランク

recvtag　　：受信タグ

comm　　：コミュニケータ

status　　：ステータスオブジェクト

ierr　　：エラーコード

・mpi_reduce(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)

機能　　：グループ内の各プロセスのデータに対し演算を行い，結果をルートのプロセスに返す．

sendbuf	：送信バッファの開始アドレス
recvbuf	：受信バッファの開始アドレス
count	：送信バッファ内の要素数
datatype	：送信バッファのデータ型
op	：演算
root	：ルートプロセスのランク
comm	：コミュニケータ
ierr	：エラーコード

This is a blank page.

国際単位系 (SI) と換算表

表 1　SI 基本単位および補助単位

量	名称	記号
長さ	メートル	m
質量	キログラム	kg
時間	秒	s
電流	アンペア	A
熱力学温度	ケルビン	K
物質量	モル	mol
光度	カンデラ	cd
平面角	ラジアン	rad
立体角	ステラジアン	sr

表 2　SI と併用される単位

名称	記号
分，時，日	min，h，d
度，分，秒	°，′，″
リットル	l，L
トン	t
電子ボルト	eV
原子質量単位	u

1 eV = 1.60218 × 10^{-19} J

1 u = 1.66054 × 10^{-27} kg

表 5　SI 接頭語

倍数	接頭語	記号
10^{18}	エクサ	E
10^{15}	ペタ	P
10^{12}	テラ	T
10^{9}	ギガ	G
10^{6}	メガ	M
10^{3}	キロ	k
10^{2}	ヘクト	h
10^{1}	デカ	da
10^{-1}	デシ	d
10^{-2}	センチ	c
10^{-3}	ミリ	m
10^{-6}	マイクロ	μ
10^{-9}	ナノ	n
10^{-12}	ピコ	p
10^{-15}	フェムト	f
10^{-18}	アト	a

表 3　固有の名称をもつ SI 組立単位

量	名称	記号	他の SI 単位による表現
周波数	ヘルツ	Hz	s^{-1}
力	ニュートン	N	m·kg/s^2
圧力，応力	パスカル	Pa	N/m^2
エネルギー，仕事，熱量	ジュール	J	N·m
工率，放射束	ワット	W	J/s
電気量，電荷	クーロン	C	A·s
電位，電圧，起電力	ボルト	V	W/A
静電容量	ファラド	F	C/V
電気抵抗	オーム	Ω	V/A
コンダクタンス	ジーメンス	S	A/V
磁束	ウェーバ	Wb	V·s
磁束密度	テスラ	T	Wb/m^2
インダクタンス	ヘンリー	H	Wb/A
セルシウス温度	セルシウス度	℃	
光束	ルーメン	lm	cd·sr
照度	ルクス	lx	lm/m^2
放射能	ベクレル	Bq	s^{-1}
吸収線量	グレイ	Gy	J/kg
線量当量	シーベルト	Sv	J/kg

表 4　SI と共に暫定的に維持される単位

名称	記号
オングストローム	Å
バーン	b
バール	bar
ガル	Gal
キュリー	Ci
レントゲン	R
ラド	rad
レム	rem

1 Å = 0.1 nm = 10^{-10} m

1 b = 100 fm^2 = 10^{-28} m^2

1 bar = 0.1 MPa = 10^5 Pa

1 Gal = 1 cm/s^2 = 10^{-2} m/s^2

1 Ci = 3.7 × 10^{10} Bq

1 R = 2.58 × 10^{-4} C/kg

1 rad = 1 cGy = 10^{-2} Gy

1 rem = 1 cSv = 10^{-2} Sv

（注）

1. 表 1 - 5 は「国際単位系」第 5 版，国際度量衡局 1985 年刊行による。ただし，1 eV および 1 u の値は CODATA の 1986 年推奨値によった。
2. 表 4 には海里，ノット，アール，ヘクタールも含まれているが日常の単位なのでここでは省略した。
3. bar は，JIS では流体の圧力を表わす場合に限り表 2 のカテゴリーに分類されている。
4. EC 閣僚理事会指令では bar，barn および「血圧の単位」mmHg を表 2 のカテゴリーに入れている。

換　算　表

力	N(=10^5 dyn)	kgf	lbf
	1	0.101972	0.224809
	9.80665	1	2.20462
	4.44822	0.453592	1

粘　度　1 Pa·s(N·s/m^2) = 10 P(ポアズ)(g/(cm·s))

動粘度　1 m^2/s = 10^4 St(ストークス)(cm^2/s)

圧力	MPa(=10 bar)	kgf/cm^2	atm	mmHg(Torr)	lbf/in^2(psi)
	1	10.1972	9.86923	7.50062 × 10^3	145.038
	0.0980665	1	0.967841	735.559	14.2233
	0.101325	1.03323	1	760	14.6959
	1.33322 × 10^{-4}	1.35951 × 10^{-3}	1.31579 × 10^{-3}	1	1.93368 × 10^{-2}
	6.89476 × 10^{-3}	7.03070 × 10^{-2}	6.80460 × 10^{-2}	51.7149	1

エネルギー・仕事・熱量	J(=10^7 erg)	kgf·m	kW·h	cal(計量法)	Btu	ft·lbf	eV
	1	0.101972	2.77778 × 10^{-7}	0.238889	9.47813 × 10^{-4}	0.737562	6.24150 × 10^{18}
	9.80665	1	2.72407 × 10^{-6}	2.34270	9.29487 × 10^{-3}	7.23301	6.12082 × 10^{19}
	3.6 × 10^6	3.67098 × 10^5	1	8.59999 × 10^5	3412.13	2.65522 × 10^6	2.24694 × 10^{25}
	4.18605	0.426858	1.16279 × 10^{-6}	1	3.96759 × 10^{-3}	3.08747	2.61272 × 10^{19}
	1055.06	107.586	2.93072 × 10^{-4}	252.042	1	778.172	6.58515 × 10^{21}
	1.35582	0.138255	3.76616 × 10^{-7}	0.323890	1.28506 × 10^{-3}	1	8.46233 × 10^{18}
	1.60218 × 10^{-19}	1.63377 × 10^{-20}	4.45050 × 10^{-26}	3.82743 × 10^{-20}	1.51857 × 10^{-22}	1.18171 × 10^{-19}	1

1 cal = 4.18605 J(計量法)

= 4.184 J　(熱化学)

= 4.1855 J　(15 °C)

= 4.1869 J(国際蒸気表)

仕事率　1 PS(仏馬力)

= 75 kgf·m/s

= 735.4 W

放射能	Bq	Ci
	1	2.70270 × 10^{-11}
	3.7 × 10^{10}	1

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58 × 10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

(86 年 12 月 1 日現在)

格子ガス気液モデルのシミュレーションコードの並列化