JAERI-Data/Code
97-052

原子力コードのVPP500における ベクトル化、並列化及び移植（並列化編） — 平成8年度作業報告書 —

1997年12月

渡辺秀雄[*]・川井　渉[*]・根本俊行[*]・川崎信夫[※]・田辺豪信[※]
鈴木信太郎[*]・原田裕夫・庄司　誠・久米悦雄・藤井　実

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（〒319-11 茨城県那珂郡東海村）あて，お申し越しください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11 茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

This report is issued irregularly.
Inquiries about availability of the reports should be addressed to Research Information Division, Department of Intellectual Resources, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

JAERI-Data/Code 97-052

原子力コードのＶＰＰ500におけるベクトル化、並列化及び移植（並列化編）
－平成８年度作業報告書－

日本原子力研究所計算科学技術推進センター
渡辺　秀雄* ・川井　　渉* ・根本　俊行* ・川崎　信夫※
田辺　豪信※ ・鈴木信太郎* ・原田　裕夫　・庄司　　誠
久米　悦雄　・藤井　　実

（1997年11月28日受理）

本報告書は、平成８年度に計算科学技術推進センター情報システム管理課で行った原子力コードのＶＰＰ500における高速化及び移植作業のうち並列化作業部分について記述したものである。

原子力コードのＶＰＰ500における高速化及び移植作業は、平成８年度に11件行われた。これらの作業内容は、同種の作業を行うユーザに有益な情報を提供することを意図して、「並列化編」、「ベクトル化編」、「移植編」の３冊にまとめられている。

本報告書の「並列化編」では、２次元相対論的電磁粒子コードEM2D、円筒座標系直接数値シミュレーションコードCYLDNS、ダイヤモンド型結晶を扱う分子動力学コードDGRに対して行ったベクトル並列化作業について記述されている。別冊の「ベクトル化編」では、中性子・光子輸送計算コード DORT-TORT、気体流動解析コード FLOWGR、相対論的ボルツマン・ウェーリング・ウーレンベック法によるシミュレーションコード RBUU に対して行ったベクトル化作業について記述されている。また、別冊の「移植編」では、軽水炉安全解析コード RELAP5/MOD3.2及び RELAP5/MOD 3.2.1.2、原子核データ処理システム NJOY、２次元多群ディスクリート・オーディネーツ輸送コード TWOTRAN-Ⅱに対して行った移植作業と汎用図形処理解析システム IPLOT に対して行った UNIX OS への移行調査作業について記述されている。

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　※　外来研究員：富士通株式会社

JAERI-Data/Code 97-052

Vectorization, Parallelization and Porting of Nuclear Codes on the VPP500 System (Parallelization) - Progress Report Fiscal 1996 -

Hideo WATANABE*, Wataru KAWAI*, Toshiyuki NEMOTO*,
Nobuo KAWASAKI※, Hidenobu TANABE※, Shintaro SUZUKI*,
Hiroo HARADA, Makoto SHOJI, Etsuo KUME and Minoru FUJII

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

(Received November 28, 1997)

Several computer codes in the nuclear field have been vectorized, parallelized and transported on the FUJITSU VPP500 system at Center for Promotion of Computational Science and Engineering in Japan Atomic Energy Research Institute. These results are reported in 3 parts, i.e., the vectorization part, the parallelization part and the porting part. In this report, we describe the parallelization.

In this parallelization part, the parallelization of 2-Dimensional relativistic electromagnetic particle code EM2D, Cylindrical Direct Numerical Simulation code CYLDNS and molecular dynamics code for simulating radiation damages in diamond crystals DGR are described. In the vectorization part, the vectorization of two and three dimensional discrete ordinates simulation code DORT-TORT, gas dynamics analysis code FLOWGR and relativistic Boltzmann-Uehling-Uhlenbeck simulation code RBUU are described. And then, in the porting part, the porting of reactor safety analysis code RELAP5/MOD3.2 and RELAP5/MOD3.2.1.2, nuclear data processing system NJOY and 2-D multigroup discrete ordinate transport code TWOTRAN-II are described. And also, a survey for the porting of command-driven interactive data analysis plotting program IPLOT are described.

Keywords: EM2D, CYLDNS, DGR, Parallelization, Vectorization, VPP500, Nuclear Codes

---

※ On leave from FUJITSU, Ltd

* FUJITSU, Ltd

目　　次

Contents

1. はじめに

計算科学技術推進センター情報システム管理課では，原子力コードをセンターが保有する各種のスーパーコンピュータ上に整備し，それぞれのスーパーコンピュータに最適な高速化を施す作業を実施している．この作業は，コンピュータ資源の効率的利用を図るとともにユーザの計算待ち時間の短縮を通じてユーザの仕事の効率化を図っている．

原子力コードのVPP500における高速化及び移植作業は，平成8年度に11件行われた．これらの作業内容は，同種の作業を行うユーザに有益な情報を提供することを意図して，「並列化編」，「ベクトル化編」，「移植編」の3冊にまとめられている．

本報告書の「並列化編」では，2次元相対論的電磁粒子コードEM2D，円筒座標系直接数値シミュレーションコードCYLDNS，ダイヤモンド型結晶を扱う分子動力学コードDGRに対して行ったベクトル並列化作業について記述されている．別冊の「ベクトル化編」では，気体流動解析コードFLOWGR，相対論的ボルツマン・ウェーリング・ウーレンベック法によるシミュレーションコードRBUU，中性子・光子輸送計算コードDORT-TORTに対して行ったベクトル化作業について記述されている．また，別冊の「移植編」では，原子核データ処理システムNJOY，2次元多群ディスクリート・オーディネーツ輸送コードTWOTRAN-II，軽水炉安全解析コードRELAP5/MOD3.2及び，RELAP5/MOD3.2.1.2についてのVPP500への移植作業と汎用図形処理解析システムIPLOTのUNIX OSへの移行調査作業について記述されている．

第2章「EM2Dコードのベクトル並列化」では，2次元相対論的電磁粒子コードEM2Dに対して行った高速化作業について述べる．本コードは，元々ベクトル化がなされていたが，一部ベクトル実行できない箇所と処理効率の点で問題となる箇所が存在していたため，この部分にベクトルチューニングを施し，更なる高速化を行なった．また，並列化を施すことで，1PE当りのデータ量を減らし，大規模な計算に対応できるようにした．この結果，16PEを使用したベクトル並列実行では，オリジナル版のベクトル実行に比較して，約5.4倍の速度性能の向上が得られた．

第3章「CYLDNSコードのベクトル並列化」では，円筒座標系直接数値シミュレーションコードCYLDNSに対して行った高速化作業について述べる．本コードについては，ILUCGS法による行列ソルバのベクトル化が不十分であったことから，この部分に対して，ベクトル化及び，並列化を施した．この結果，16PEを使用したベクトル並列実行では，オリジナル版のベクトル実行に比較して，約12.5倍の速度性能の向上が得られた．

第4章「DGRのベクトル並列化」では，ダイヤモンド型結晶を扱う分子動力学コードDGRに対して行った高速化作業について述べる．本コードでは，ベクトル化版コードの一部修正と並列化を施した．並列化においては，原子間の相互作用計算を行うに際して，単純に原子ループを分割しただけでは依存関係が問題となってしまうため，各物理時間ステップで行う処理を更に分割し，時間差を設けて計算させるようにした．この結果，16PEを使用したベクトル並列実行では，オリジナル版のベクトル実行に比較して約12倍の速度性能の向上が得られた．

尚，本報告書の第2章及び第3章の作業は渡辺が，第4章の作業は川井が担当した．

2. EM2D コードのベクトル並列化

2.1 はじめに

本作業では EM2D コードの 富士通(株)製分散メモリ型ベクトル並列計算機 VPP500/42(以下 VPP500) 向け並列化作業を行った. 本並列化作業における並列化の方針は次の 2 つである.

- 1 プロセッサあたりのデータ量を減らして大規模な計算に対応できるようにするため, 本コードが取り扱う粒子と空間の 2 種類のデータのそれぞれを複数のプロセッサに分担して保持させる.
- 粒子データを格納する配列は単純に分割して各プロセッサに担当させるのではなく, 各粒子の空間上の位置に注目して分割する. つまり, あるプロセッサに割り当てられた空間上に存在する粒子に関するデータをそのプロセッサに担当させる. 粒子データを単純に分割した場合, 粒子と空間の相互作用の計算時にプロセッサ間のデータ転送が大量に発生してしまうが, このようにすることで粒子と空間の相互作用の計算時のデータ転送量を非常に小さくすることができる. ただし, プロセッサが担当する空間外に移動してしまった粒子に関するデータを適当なプロセッサに転送する処理が必要となる.

以下, 本作業の内容について示す.

2.2 並列化

VPP500 は分散メモリ型計算機であるため, VPP500 向け並列化を行うに当たってはデータをどのように各プロセッサに分担させるか (以下, データの分割) が非常に重要な問題となる. データ分割の方針が決定されると, 各プロセッサは自分に割り当てられたデータに関する計算を行えばよいので, 処理の分割についての基本的な方針も決定されることになる. つまり処理の分割がデータの分割方法に依存することになるため, 不適当なデータの分割を行うと各プロセッサに割り当てられる処理量が均一とならず, 並列化効率に悪影響を与えてしまう. また, 自分の処理に必要なデータを他のプロセッサが保持している場合, 処理の前にプロセッサ間のデータ通信を行い, あらかじめ必要なデータを自分が保持するようにしなければならない. あるいは, あるプロセッサでの処理の終了後, 計算結果を他のプロセッサに転送する処理が必要となることもある. このプロセッサ間のデータ転送は並列化効率に悪影響を及ぼすので, データ転送量をできるだけ少なくすることを考慮して, データの分割の方針を決定する必要がある. 本並列化作業におけるデータ分割の方針を決定する際にも, 処理の均一化とデータ転送量を小さく抑えることを考慮した.

以下, 本コードの計算内容について簡単に示した後, データ分割の方針, 処理分割の方針, 及びその他の本コードの変更に関する方針について述べる.

2.2.1 コード概要

本コードは基本的に粒子コードであると考えられる. ただし, 粒子間の相互作用によって次ステップの粒子の分布が決定されるのではなく, 粒子と空間が相互に影響を及ぼし合うことによっ

て次ステップにおける粒子の分布が計算される．このように本コードは主に粒子に関するデータと空間に関するデータの２種類のデータを持っている．空間は２次元平面であり，x 軸方向と y 軸方向の両方に対して周期境界条件が課せられている．粒子間の相互作用は存在しない．

また，本コードの計算内容は大きく次の４種類に分けることができる．

粒子が空間に与える影響の計算

粒子データを参照して実空間（２次元平面）の各格子点に与える影響を計算する．各粒子はそれぞれ４つの格子点 (kpx,kpy)，(kpx+1,kpy)，(kpx,kpy+1)，(kpx+1,kpy+1) に影響を与える．ここで kpx と kpy の値は，粒子 i の位置 x(i) と y(i) から式 (2.1) のように計算される．

```
kpx = x(j)*odlt_xg + 0.500001d0
kpy = y(j)*odlt_yg + 0.500001d0
```
(2.1)

空間のみに関する計算

２次元平面の各格子点上の各種物理量の計算を行う．なお，実空間とフーリエ空間の２種類が使用されており，両者間の変換に２次元（逆）FFT 計算が用いられる．

空間が粒子に与える影響の計算

空間データを参照して粒子毎の各種物理量を計算する．各粒子はそれぞれ実空間上の４つの格子点 (kpx,kpy)，(kpx+1,kpy)，(kpx,kpy+1)，(kpx+1,kpy+1) からの影響を受ける．ここで kpx と kpy の計算方法は，粒子が空間に与える影響の計算の場合と同様である．

粒子のみに関する計算

空間データを参照することなく粒子毎に各種物理量の計算を行う．粒子間の相互作用は存在しない．

2.2.2 データ分割の方針

本並列化では大規模計算への対応を考慮して，粒子に関するデータ及び空間に関するデータの全てを分割の対象とすることにした．以下，それぞれのデータの分割方法について述べる．

2.2.2.1 空間データの分割

本コードでは空間に関するデータは２次元配列に格納される．本並列化ではこれらの配列を次の３種類に分けて考えることにした．それぞれの配列の概要及びそれぞれの分割方法を次に示す．

実空間上のデータを格納する配列（大きさ：(0:Nx_p+1,0:Ny_p+1)）

配列の２次元目（以下，y 軸方向）をデータ分割の対象とし，全プロセッサに均等に保持させることにした．なお，この配列は空間と粒子との間の影響を計算する際に使用される配列なので「右袖」が必要となる（「2.2.2.2 粒子データの分割」参照）．

実空間上のデータを格納する配列（大きさ：(Nx_p,Ny_p)）

上の配列の場合と同様に，y 軸方向をデータ分割の対象とし，全プロセッサに均等に保持させることにした．ただし，この配列と上の配列との間にはデータのやりとりが発生するので，データ通信が発生しないように上の配列と同じ箇所で分割されるようにしておく．具体的には，分割時に指定するインデックスの範囲を 1 から始めるのではなく，上の配列の場合と同じ 0 から始めるようにしておく（Fig. 2.1 参照）．

フーリエ空間上のデータを格納する配列

実空間上のデータを格納する配列の場合と異なり，x 軸方向をデータ分割の対象とした．このように互いに異なる分割方法を採用した理由については「2.2.3.5 2次元 FFT 計算の並列化方法」で述べる．

2.2.2.2 粒子データの分割

空間データの場合と同様に粒子データを単純に各プロセッサで均等分割しておくと，空間・粒子間の相互作用の計算の際に他プロセッサの保持する空間データあるいは粒子データの参照が必要となり，プロセッサ間のデータ転送が大量に発生してしまう可能性がある．このデータ転送量を小さく抑えるため，それぞれの粒子の位置から計算される変数 kpy の値に注目して粒子データの分割を行うことにした．

実空間のデータを格納する配列は y 軸方向を分割の対象として各プロセッサに均等に割り当てられる．あるプロセッサが y=y1 から y=y2 の範囲を割り当てられているとすると，そのプロセッサには kpy の値が y1 と y2 の範囲となる粒子のデータを担当させることにする．その結果，粒子と空間の相互作用の計算をプロセッサ間のデータ通信なしに行うことが可能となる．ただし，この粒子と空間の相互作用の計算は空間上の 4 点（(kpx,kpy)，(kpx+1,kpy)，(kpx,kpy+1)，(kpx+1,kpy+1)）が参照，定義されることになるため，kpy=y2 となる粒子については正しい計算を行うことができない．そこで，空間データに「右袖」をつけておくことにした．粒子が空間から受ける影響を計算する際には，あらかじめ右隣のプロセッサから「右袖」の部分に空間データの一部を転送しておくようにすることで，kpy=y2 となる粒子についても正しい空間データの参照が可能となる．また，空間が粒子から受ける影響を計算する際にも，計算後に「右袖」部分を右隣のプロセッサに転送し，そのプロセッサが持つ空間データの左端に重ね合わせることで正しい計算ができることになる．

なお，粒子データは十分な大きさを確保した重複ローカル配列に格納することにする．また，そのプロセッサが担当している粒子数を重複ローカル変数に格納するものとする．

2.2.3 処理分割の方針

VPP500 における並列ジョブの動作は冗長実行部分と並列計算部分に分けられる．一般に冗長実行部分では全プロセッサが同じ挙動をするため，並列実行による速度向上が得られない．ただし，全プロセッサが同じ挙動を示すのはこの部分で参照されるデータが全プロセッサで同じ値を持っているためであり，意識的にプロセッサ毎に異なる値を与えておけば個々のプロセッサがそれぞれ異なる動作をするようになる．本並列化作業では，粒子ループに対してこれを利用して

並列計算を行わせることにした.

本コードで行われる4種類の計算部分の基本的な処理分割の方針を次に示す. また, その他の特別な方法を用いて並列化を行う部分について, その並列化方法を示す.

2.2.3.1 粒子に関する計算部分

並列計算のための特別な変更は行わず冗長実行による並列実行を利用する.

2.2.3.2 空間に関する計算部分

各プロセッサが自分の保持する配列の領域だけを計算の対象とするようにDOループの処理を分割する.

2.2.3.3 空間が粒子に与える影響の計算部分

この処理は粒子ループで構成される. したがって粒子に関する計算部分の並列化方法と同様に, 特に並列化のための指示行などを指定せず, 冗長実行による並列化を利用する. ただし, 空間データを格納する配列の「右袖」部分を参照することがあるので, あらかじめ "overlapfix" 文で「右袖」部分にデータを転送しておく.

2.2.3.4 粒子が空間に与える影響の計算部分

この処理も粒子ループで構成されるため, 特別な指示行は指定せず, 冗長実行による並列化を利用する. ただし, 空間データを格納する配列の「右袖」部分に値が定義されることがあるため, あらかじめ「右袖」部分をゼロクリアしておく. また, 計算の終了後, 「右袖」に格納された値を隣のプロセッサに転送して, そのプロセッサが持つ空間データの左端に重ね合わせる処理を行う. 以下, この処理の概要を示す.

まず準備として Fig. 2.2 に示す処理を行い, 各プロセッサが担当する空間の範囲を重複ローカル変数 jmin と jmax に格納しておく. ここで, 空間を割り当てられていないプロセッサには jmin に -1 が格納される. 次に, 粒子が空間に与える影響の計算部分を Fig. 2.3 のように変更する. 図中の配列 Bz は空間データを格納する配列であり, 他の手続き中で既に袖部分のゼロクリアが行われている. do 1300 ループ中で粒子が空間に与える影響の計算が行われた後, do 3000 ループで右袖部分を右隣のプロセッサに転送する. このループ中の IF 文は, 空間データを割り当てられており, かつ空間の右端を担当していないプロセッサだけに転送処理を行わせるためのものである. do 3010 ループ中で実際の転送が行われるが, この時 mp+1 は右隣のプロセッサを, jmax+1 は右袖のインデックスを示している. 最後に do 3100 ループで左隣のプロセッサから転送されたデータを自分が担当する空間の左端に加える処理を行う. ループ中の IF 文は, 左隣のプロセッサからデータを転送されたプロセッサだけを動作させるためのものである (jmin.lt.0 は空間データを割り当てられていないプロセッサを示し, jmin.eq.0 は全空間の左端を担当しているプロセッサを示す. これらのプロセッサには do 3000 ループによってデータが転送されることはない).

2.2.3.5 2次元 FFT 計算の並列化方法

2次元 FFT 計算は x 軸方向の1次元 FFT 計算と y 軸方向の1次元 FFT 計算から構成される．x 軸方向の1次元 FFT 計算の場合には y 軸方向に並列性があり，この方向でベクトル計算を行う．したがって x 軸方向の1次元 FFT 計算を行う際には，空間データを y 軸方向を分割の対象としてデータ分割しておけば，ベクトル長は減少するものの複数のプロセッサで処理を分担することが可能となる．それに対して y 軸方向の1次元 FFT 計算の場合，x 軸方向に並列性があるため，x 軸方向を分割の対象としてデータ分割しておくことにより，ベクトル長は減少するものの複数のプロセッサで処理を分担させることが可能となる．以上のように2次元 FFT 計算では，その計算の途中でデータ分割の分割軸を変更する必要があり，プロセッサ間のデータ転送が発生してしまう．このデータ転送の回数を少なくするため，実空間とフーリエ空間とで配列の分割軸を異なるようにしておき，かつ FFT 計算の順序を変更して次の手順で計算を行うことにした．

正変換

1. y 軸方向を分割の対象とする配列（実空間上のデータを格納する配列）に対して x 軸方向の1次元 FFT 計算を行う．
2. プロセッサ間のデータ転送を行い，分割軸を変更する．具体的には，y 軸方向を分割の対象とする配列から x 軸方向を分割の対象とする配列（フーリエ空間上のデータを格納する配列）にデータをコピーする．
3. x 軸方向を分割の対象とする配列に対して y 軸方向の1次元 FFT 計算を行う．

逆変換

1. x 軸方向を分割の対象とする配列（フーリエ空間上のデータを格納する配列）に対して y 軸方向の1次元逆 FFT 計算を行う．
2. プロセッサ間のデータ転送を行い，分割軸を変更する．具体的には，x 軸方向を分割の対象とする配列から y 軸方向を分割の対象とする配列（実空間上のデータを格納する配列）にデータをコピーする．
3. y 軸方向を分割の対象とする配列に対して x 軸方向の1次元逆 FFT 計算を行う．

2.2.3.6 総和計算

本コードでは全粒子の運動エネルギーの和を計算する処理が行われる．粒子に関するデータは複数のプロセッサで分担して保持することにしたため，この処理は次のように並列化することにした．

まずそれぞれのプロセッサ毎に総和計算を行う．この処理は，個々のプロセッサが自分の担当する粒子データのみを参照すればよいのでプロセッサ間のデータ通信をすることなく並列実行することができる．次にグローバル SUM 関数を用いて，プロセッサ毎に求めた値をプロセッサ間で総和する．総和した結果は自動的に全プロセッサに転送されるので，グローバル SUM を実行した結果，全プロセッサが全粒子のエネルギーの和を得ることができる（Fig. 2.4 参照）．

2.2.3.7 空間境界部分の補正

本コードが取り扱う2次元平面には x 軸方向及び y 軸方向のそれぞれに周期境界条件が適用されており，空間に関する処理の中で反対側の境界部分を参照して境界部分を補正する計算が行われている．この計算の対象となる配列（実空間上のデータを格納する配列）は y 軸方向が分割の対象となっているので，x 軸方向の境界の補正計算は特に問題なく並列計算することができる．しかし y 軸方向の場合は，両端の境界部分を一般にそれぞれ異なるプロセッサが保持しているのでプロセッサ間のデータ転送が必要となる．また，すべてのプロセッサが計算を行うわけではないので，特定のプロセッサだけに処理を行わせるようにした．

Fig. 2.5 にこの処理の並列化例を示す．do 1500 ループでは x 軸方向の境界の補正計算を行っているが，空間データを格納する配列 Bz は y 軸方向が分割の対象となっているため，データ転送の必要はない．それに対して do 1600 ループでは y 軸方向の境界の補正計算を行っている．図中の IF 文によって境界部分を担当するプロセッサだけを動作させ，反対側の境界データを取り寄せて自分の持つデータに加える処理を行う．

2.2.4 その他

以下，プログラムの変更に関するその他の方針について示す．

2.2.4.1 プロセッサ間の粒子データの交換

全ての粒子について空間から受ける力を計算した後，各粒子の座標が更新される．その結果それまで担当していたプロセッサの領域外に移動してしまう粒子については，その粒子に関する全てのデータを新たに担当するべきプロセッサに転送する必要がある．本作業では2つの新しいサブルーチンを作成してこの処理を行わせることにした．実際の転送処理は新サブルーチン move で行い，新サブルーチン move_init は move で参照される変数，配列の初期化を行う．これら新規に作成したサブルーチンの内容を Fig. 2.6 と Fig. 2.7 に示す．

2.2.4.2 配列宣言方法の変更

オリジナルのプログラムでは，MAIN プログラム中で配列 a を宣言しておき，手続き間の引数の結合を利用して，この配列 a の領域を複数の領域に分割してそれぞれを別の配列名で利用していた．この方法は配列の大きさを動的に決めることができる長所を持つ．ところが VPP FORTRAN77EX/VPP コンパイラは，このように動的に大きさが決まる配列のそれぞれを分割して複数のプロセッサに保持させることができない仕様となっている．

そこで，それぞれの配列を独立した配列として宣言しておくことにした．この時それぞれの配列の大きさを定数で与えておく必要があるため，空間に関するデータの場合にはパラメータ Nx_p と Ny_p を，粒子に関するデータの場合には N_p_p を定義して利用することにした．これら3つのパラメータは並列化にともなって新たに導入するものである．

また，プログラムの変更を容易にするため，これらの配列はすべてコモン配列で置き換えることにした．

2.2.4.3 ロードファイルとリスタートファイルの分割

本プログラムは load.dat ファイル（以下，ロードファイル）またはリスタートファイルを入力とし，計算終了時にリスタートファイルを出力する．これらのファイルには個々の粒子に関するデータが含まれているので，ファイルの読み込み中または読み込み後にプロセッサ間のデータ転送を行って，個々の粒子データを適当なプロセッサに分配する必要がある．この処理には比較的大きな作業領域と複雑な処理が必要となる．そこで，あらかじめ粒子データを複数のファイルに分割しておき，並列化後のプログラムでは各プロセッサが自分に割り当てられたファイルを読み込めば済むようにした．その結果，特別な作業領域も必要なく，かつ簡単な処理でファイルの入出力を行うことができるようになった．また，その他のデータ（空間に関するデータ等）については，全プロセッサが共通にアクセスする１つのファイルに格納することにした．したがって，ロードファイルとリスタートファイルはそれぞれ（プロセッサ台数＋１）個のファイルに分割される．

2.3 ベクトル化

本コードには，回帰参照が原因でベクトル化が行われない箇所と，ベクトル化は行われるもののメモリアクセス方法に問題があって処理効率がそれほどよくないと思われる箇所の２点の問題点が含まれていた．以下，これらの問題点を改善するための方針について示す．また，本コードの並列化に伴いベクトル長が減少してしまう箇所に対する対処方法について述べる．

2.3.1 回帰参照の回避

空間が粒子から受ける影響を計算する部分は，Fig. 2.8 のようなループで構成される．図中の DO 1400 ループは粒子ループであり，DO 変数 j は個々の粒子に割り当てられた粒子の番号に相当する．ループ中では，それぞれの粒子について kpx と kpy の値を計算し，空間に関するデータを格納する配列 rho の４箇所（kpx,kpy），（kpx+1,kpy），（kpx,kpy+1），（kpx+1,kpy+1）に値を加える処理を行う．このような DO ループでは複数の粒子が空間上の同じ箇所に影響を与える可能性があり，その場合回帰参照が発生してしまうことになるためベクトル計算を行うことができない．

このような DO ループをベクトル計算する方法として次の２通りが考えられる．ひとつは，全ての空間上の格子点それぞれについてたかだか１つの粒子のみが影響を与えることがあらかじめわかっている場合に，DO 文の直前に最適化制御行 “*vocl loop,novrec” を挿入しておく方法である．つまり，回帰参照の可能性が全くないことがわかっていれば，それをコンパイラに知らせてやることでベクトル計算が行われるようになる．もうひとつの方法は，複数の空間を用意しておき個々の粒子がそれぞれ異なる空間に対して値を定義するようにする方法である．最後に全ての空間の要素毎の和を計算すれば正しい結果が得られる．このようにすると同一の空間上の１点に対して複数の粒子が影響を与えることがなくなるためベクトル計算を行うことが可能となる．ただし，粒子の個数だけ空間（作業配列）を用意しておくのは計算機のメモリを大量に消費してしまうことになるため，通常は十分な個数の空間を用意しておきそれらの空間を繰り返し使用する Fig. 2.9 のようなプログラムとしておく．この場合，最も内側の DO ループ中では粒子と

空間との間に1対1の関係があるので回帰参照の可能性はなく，このループ中の計算はベクトル実行することができる．本コードのベクトル化には，この方法を用いることにした．

2.3.2 メモリアクセス方法の改善

本コードには空間の境界部分の計算やx軸方向の1次元FFT計算の場合など，配列の2次元目でベクトル化されている箇所がある．このような箇所ではメモリのアクセスが一定の間隔（配列の1次元目の大きさ）で飛び飛びとなっている．

ベクトル計算機では，メモリ領域を連続にアクセスできる場合が最も効率がよい．次いで一定の間隔で飛び飛びにアクセスする場合，リスト配列などを用いたランダムなアクセスの順となっている．この一定の間隔で飛び飛びにアクセスする場合には，その間隔の大きさが奇数であるのが望ましく，偶数（特に2の巾乗）の場合にはメモリアクセスの効率が悪くなってしまう．

本コードの場合空間に関するデータを格納する配列の1次元目の大きさは2の巾乗（Nx）か，それに2を加えた（0:Nx+1）偶数値であったので，1を加えて奇数値にしておくことにした（Nx+1 または 0:Nx+1+1）．その結果アクセスされることのない無駄な領域がメモリ上に生じることになるが，メモリアクセスの向上に伴いベクトル実行の効率が向上する．

2.3.3 並列化に伴うベクトル長減少の回避

「2.2並列化」で述べたように，フーリエ空間上のデータを格納する配列についてはx軸方向（2次元配列の1次元目）を分割の対象にすることにした．その結果 Fig. 2.10 に示すようなDOループは並列化によってベクトル長が短くなってしまう．これを避けるため，DO文を入れ換えておくことにした．ただし，その結果として配列の2次元目でベクトル化が行われるようになるため，ベクトル処理効率を考慮して配列の添字を入れ換えておくことにした（Fig. 2.11 参照）．

2.4 計算結果の評価

並列化後の本コードによる計算結果が妥当なものであるかについてはユーザにお願いして確認していただいた．その過程でいくつかの問題点が見つかり，その修正を行った．その結果，オリジナルの本コードを他計算機上で実行した時の結果と大差ない計算結果が得られ，妥当なものであるとの評価を得た．

2.5 効果

以下，本作業によって得られた効果について述べる．まず，本作業で行ったベクトル化方法のひとつにパラメータによって処理効率が変化するものがあるため，最も適当なパラメータの値について考察する．次に本作業の結果，並列実行によってどの程度の効果が得られるかについて述べる．

2.5.1 パラメータ jjjskp の値について

「2.3.1 回帰参照の回避」で述べたベクトル化方法は，パラメータ jjjskp の値によって処理効率が変化する．このパラメータの値が大きい程良好な処理効率が得られるが，非常に大きな作業領域が必要となってしまうので，jjjskp には十分な処理効率が得られる最小の値を指定するのが望ましい．そこで，jjjskp の値を変化させた時にプログラム全体の計算時間がどの程度となるかを調査した．調査結果を Table 2.1に示す．また，この結果をグラフにしたものを Fig. 2.12 に示す．この結果から，jjjskp の値は 32 程度であるのが最も望ましく，あるいは 16 程度でも比較的良好なベクトル処理効率が得られることがわかった．

なおこの調査には，プロセッサ台数を 1 とした（MPE=1）並列化版ソースプログラムを用い，このソースプログラムに対して非並列コンパイル（-Wx オプションなし）を行って作成したロードモジュールを使用した．また，入力データには Fig. 2.13 に示すものを使用した．

2.5.2 並列化の効果

並列化による効果を調べるため，並列化版の本コードを 4 台のプロセッサで実行した場合と 16 台で実行した場合の計算時間を測定した．この時，ベクトル処理効率に影響するパラメータ jjjskp には 16 を指定し，入力データには Fig. 2.13 に示したものを利用した．また，並列実行時の計算時間が不安定だったため，それぞれ 5 回ずつ測定を行った．測定結果と並列化効率などを計算した結果を Table 2.2 に示す．ここで，並列化による速度向上と並列化効率は式 (2.2) を用い，1 プロセッサで実行した時の経過時間に Table 2.1 の jjjskp=16 の測定結果を使用して計算した．

$$\begin{aligned} \text{並列化による速度向上} &= \frac{\text{1プロセッサで実行した時の経過時間}}{\text{並列実行時の経過時間}} \\ \text{並列化効率（\%）} &= \frac{\text{並列化による速度向上}}{\text{プロセッサ台数}} \times 100 \end{aligned} \tag{2.2}$$

この表に示したようにプログラム全体の実行時間からは良好な並列化効率を得ることができなかった．これは計算時間の割にファイル入出力に要する時間が比較的大きいためではないかと思われたため，このファイル入出力に要する時間を削除した場合の並列化効率も求めてみることにした．具体的には，MAIN プログラム中の時間発展ループ DO 1000 の前後にサービスサブルーチン GETTOD の呼び出しを挿入して DO 1000 ループ中の処理に要する経過時間を測定した．また，サブルーチン diagout と snapout 中の WRITE 文を全て削除して DO 1000 ループ中では全くファイル入出力が行われないようにした．測定結果を Table 2.3 に示す．

2.6 まとめ

本作業の結果，4 プロセッサで 73%，16 プロセッサで 34% 程度の並列化効率を得た．これは，期待していたよりも小さめの値である．このように小さめの並列化効率しか得られない原因としては，並列計算の粒度が小さい割に全プロセッサによる同期が頻繁に行われること，及び FFT 計算で行われるデータ転送量が比較的多いことなどが考えられる．

一方，主な配列のほとんどを分割したことで，これまで以上に大規模な計算を行うことが可能となった．

Table 2.1 Execution time on each value of parameter `jjjskp`

jjjskp	1	4	8	16	32	64
real	489.97s	424.33s	318.54s	267.55s	228.74s	251.90s
user	463.63s	400.76s	286.38s	233.55s	209.01s	204.48s
sys	1.31s	1.26s	1.25s	1.25s	1.29s	1.38s
vu-user	103.84s	342.37s	227.67s	175.56s	150.97s	145.85s

Table 2.2 Execution time of parallel executions

(a) 4 プロセッサでの結果

	1 回目	2 回目	3 回目	4 回目	5 回目
real	162.47s	147.34s	153.51s	162.52s	227.34s
user	490.04s	473.36s	494.62s	507.53s	691.37s
sys	17.79s	17.80s	17.72s	17.69s	17.70s
vu-user	201.49s	201.49s	201.48s	201.49s	201.49s
速度向上	1.65	1.82	1.74	1.65	1.18
並列効率	41.3%	45.5%	43.5%	41.3%	29.5%

(b) 16 プロセッサでの結果

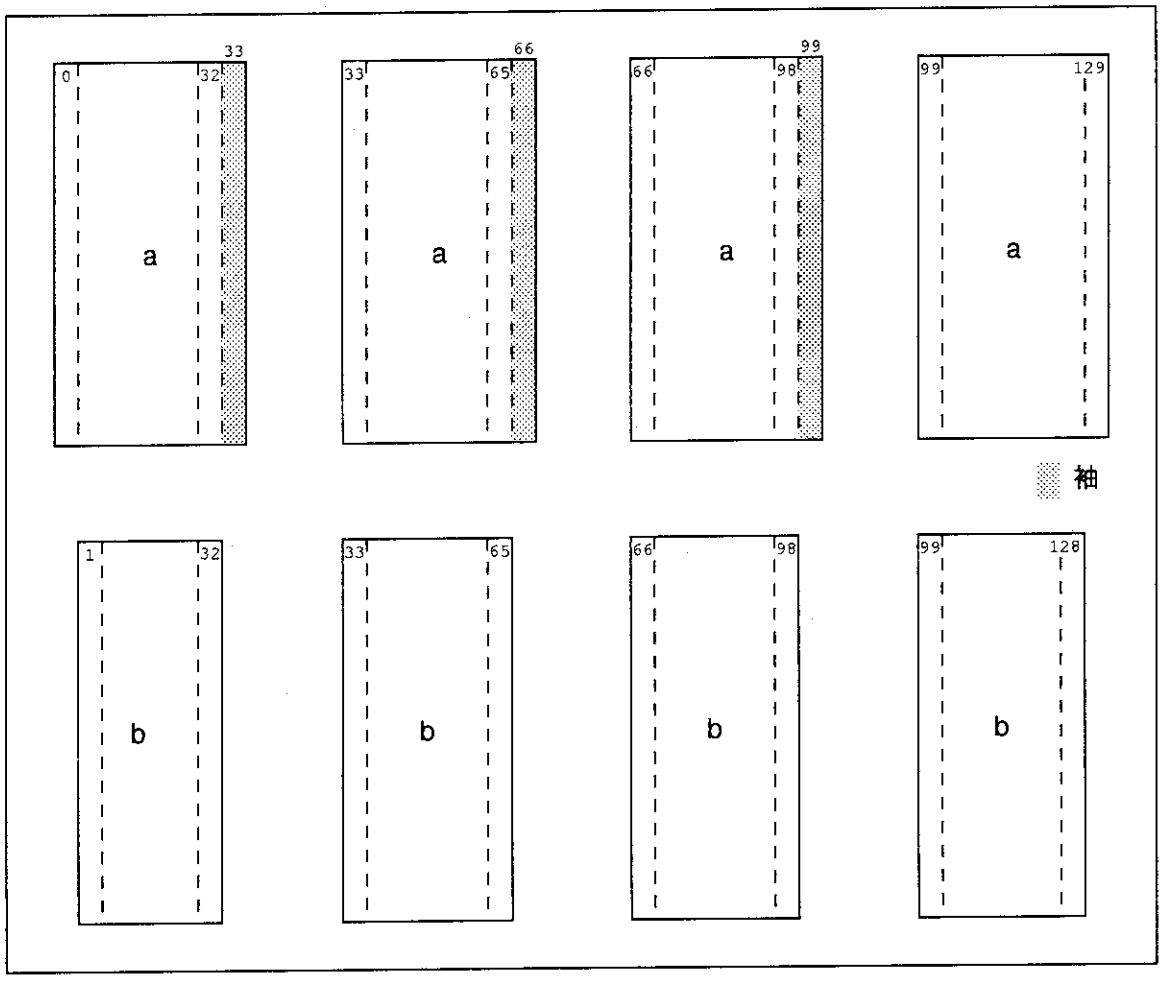
	1 回目	2 回目	3 回目	4 回目	5 回目
real	130.38s	136.38s	141.68s	134.43s	129.13s
user	1572.89s	1588.64s	1655.02s	1600.39s	1583.78s
sys	24.28s	24.19s	24.27s	24.12s	24.30s
vu-user	333.88s	333.88s	333.88s	333.88s	333.88s
速度向上	2.05	1.96	1.89	1.99	2.07
並列効率	12.8%	12.3%	11.8%	12.4%	12.9%

Table 2.3 Speed up and effect (without file I/O time)

プロセッサ台数	1	4	16
測定結果（秒）	225.37	77.21	41.26
速度向上	—	2.92	5.46
並列化効率	—	73.0%	34.1%

```
      parameter(MPE=4,Nx_p=...,Ny_p=128)
!xocl processor pe(MPE)
!xocl index partition ipy=(pe,index=0:Ny_p+1,part=band)
!xocl index partition ipyo=ipy(overlap=(0,1))
      dimension a(0:Nx_p+1,0:Ny_p+1)
      dimension b(  Nx_p   ,   Ny_p   )
!xocl local  a(:,/ipyo)
!xocl local  b(:,/ipy )
```

(a) プログラム



(b) 分割の様子

Fig. 2.1 Example of data division for real space

```
      PARAMETER(MPE=4)
!xocl processor pe(MPE)
      PARAMETER(Nx_p=512,Ny_p=128)
!xocl subprocessor spe=pe
!xocl index partition ipy=(spe,index=0:Ny_p+1,part=band)
        :
      jmin = -1
!xocl spread do /ipy
      do 100 j = 0, Ny+1
         if ( jmin.lt.0 ) jmin = j
  100 continue
!xocl end spread
      if ( jmin.lt.0 ) then
         jmax = jmin - 1
      else
         jmax = j - 1
      endif
        :
```

Fig. 2.2 Part of new subroutine **olap_init**

```
      PARAMETER(MPE=4)
!xocl processor pe(MPE)
      PARAMETER(Nx_p=512,Ny_p=128)
!xocl subprocessor spe=pe
      dimension Bz(0:Nx_p+1+1,0:Ny_p+1)
!xocl index partition ipy=(spe,index=0:Ny_p+1,part=band)
!xocl index partition ipyo=ipy(overlap=(0,1))
!xocl local  Bz  (:,/ipyo)
      dimension wk10  (0:Nx_p+1,3,MPE)
!xocl index partition ipmpe=(spe,index=1:MPE,part=band)
!xocl local  wk10  (:,:,/ipmpe)
!xocl global wk10_g
        :
      do 1300 is =  1,N_sp
         coef = q(is)*qn(is)
         do 1400 j = l_st(is), l_ed(is)
            kpx = x(j)*odlt_xg + 0.500001d0
            kpy = y(j)*odlt_yg + 0.500001d0
            chrg1 = (dlt_xg-(x(j)-xg(kpx)))*(dlt_yg-(y(j)-yg(kpy)))
            chrg2 = (x(j)-xg(kpx))*(dlt_yg-(y(j)-yg(kpy)))
            chrg3 = (dlt_xg-(x(j)-xg(kpx)))*(y(j)-yg(kpy))
            chrg4 = (x(j)-xg(kpx))*(y(j)-yg(kpy))
c
            Bz(kpx  ,kpy  ) = Bz(kpx  ,kpy  ) + chrg1*coef
            Bz(kpx+1,kpy  ) = Bz(kpx+1,kpy  ) + chrg2*coef
            Bz(kpx  ,kpy+1) = Bz(kpx  ,kpy+1) + chrg3*coef
            Bz(kpx+1,kpy+1) = Bz(kpx+1,kpy+1) + chrg4*coef
 1400    continue
 1300 continue
!xocl spread do
      do 3000 mp = 1, MPE
         if ( jmin.ge.0 .and. jmax.lt.Ny+1 ) then
!xocl spread move
            do 3010 i = 0, Nx+1
               wk10_g(i,1,mp+1) = Bz(i,jmax+1)
 3010       continue
!xocl end spread (mw1)
!xocl movewait (mw1)
         endif
 3000 continue
!xocl end spread
c
!xocl spread do
      do 3100 mp = 1, MPE
         if ( jmin.gt.0 ) then
            do 3110 i = 0, Nx+1
               Bz(i,jmin) = Bz(i,jmin) + wk10(i,1,mp)
 3110       continue
         endif
 3100 continue
!xocl end spread
        :
```

Fig. 2.3 Part of parallelized subroutine charge

```
        :
!xocl spread do
      do 4000 mp = 1, MPE
         do 4100  is = 1,N_sp
            energy_k(is) = 0.d0
            do 4110  j =  l_st(is),l_ed(is)
               p2 = 0.25d0 * ( P(j,1)**2 + P(j,2)**2 + P(j,3)**2
     \            + Pn(j,1)**2 + Pn(j,2)**2 + Pn(j,3)**2
     \            + 2.d0*P(j,1)*Pn(j,1) + 2.d0*P(j,2)*Pn(j,2)
     \            + 2.d0*P(j,3)*Pn(j,3) )
               energy_k(is) = energy_k(is) +
     \                dsqrt( 1.d0 +  p2 /(p_mass(is)*c)**2 )
 4110       continue
 4100    continue
 4000    continue
!xocl end spread sum(energy_k)
      do 4200 is = 1, N_sp
         energy_k(is) = energy_k(is) - N_p(is)
         energy_k(is) = p_mass(is) * c * c * energy_k(is)
         energy_k(is) = energy_k(is)/Np_eon
         energy_k_t  =  energy_k_t + energy_k(is)
 4200 continue
        :
```

Fig. 2.4 Part of parallelized subroutine `kinetic_e`

```
      PARAMETER(MPE=4)
!xocl processor pe(MPE)
      PARAMETER(Nx_p=512,Ny_p=128)
!xocl subprocessor spe=pe
      dimension Bz(0:Nx_p+1+1,0:Ny_p+1)
!xocl index partition ipy=(spe,index=0:Ny_p+1,part=band)
!xocl index partition ipyo=ipy(overlap=(0,1))
!xocl local  Bz  (:,/ipyo)
      dimension wk11(0:Nx_p+1,3)
        :
!xocl spread do /ipy
      do 1500 j = 0, Ny+1
         Bz(1 ,j) = Bz(1 ,j) + Bz(Nx+1,j)
         Bz(Nx,j) = Bz(Nx,j) + Bz(   0,j)
 1500 continue
!xocl end spread
c
!xocl spread do /ipy
      do 1600 j = 1, Ny, Ny-1
c
         if ( j.eq.1 ) then
!xocl spread move
            do 1610 i = 0, Nx+1
               wk11(i,1) = Bz_g(i,Ny+1)
 1610       continue
!xocl end spread (mw2)
!xocl movewait (mw2)
            do 1620 i = 0, Nx+1
               Bz(i, 1) = Bz(i, 1) + wk11(i,1)
 1620       continue
         endif
c
         if ( j.eq.Ny ) then
!xocl spread move
            do 1630 i = 0, Nx+1
               wk11(i,1) = Bz_g(i,   0)
 1630       continue
!xocl end spread (mw3)
!xocl movewait (mw3)
            do 1640 i = 0, Nx+1
               Bz(i,Ny) = Bz(i,Ny) + wk11(i,1)
 1640       continue
         endif
c
 1600 continue
!xocl end spread
        :
```

Fig. 2.5 Part of parallelized subroutine charge

```
      subroutine move_init
c
      implicit real*8 (a-h,o-z)
c
      logical*1 system_l_2pi
      common /input$/ c, dlt_t, system_lx, system_ly,
     \                Nx, Ny, No_ion, No_eon, n_time,
     \                int_snap,system_l_2pi
c
      PARAMETER(MPE=4)
!xocl processor pe(MPE)
      PARAMETER(Nx_p=512,Ny_p=128)
c
!xocl subprocessor spe=pe
!xocl index partition ipy=(spe,index=0:Ny_p+1,part=band)
c     include '../include/MOVE'
      common / move1 / myid, jrange_min(MPE), jrange_max(MPE)
c
        ! 自プロセッサ番号の取得
!xocl spread do
      do 1000 mp = 1, MPE
         myid = mp
 1000 continue
!xocl end spread
c
c---
c
        ! 自分が担当する y 軸方向の空間の範囲の計算.
        ! 空間が割り当てられていないプロセッサには,
        ! do j=jmin,jmax のような DO ループを処理さ
        ! せないように jmin > jmax としておく.
      jmin = -1
!xocl spread do /ipy
      do 2000 j = 0, Ny
         if ( jmin.lt.0 ) jmin = j
 2000 continue
!xocl end spread
      if ( jmin.lt.0 ) then
         jmax = jmin - 1
      else
         jmax = j - 1
      endif
c
        ! 各プロセッサが担当する範囲の情報を全ての
        ! プロセッサで共有する.
      do 2100 mp = 1, MPE
         jrange_min(mp) = 0
         jrange_max(mp) = 0
 2100 continue
c
!xocl spread do
      do 2200 mp = 1, MPE
         jrange_min(mp) = jmin
         jrange_max(mp) = jmax
 2200 continue
!xocl end spread sum(jrange_min), sum(jrange_max)
c
      end
```

Fig. 2.6 Contents of new subroutine move_init

```
      subroutine move(iflag)
c
      include '../include/define.f'
      include '../include/prtcl.f'
      include '../include/cnst.f'
c
      include '../include/PARAM'
      include '../include/IP'
      include '../include/ARRAY'
c     include '../include/MOVE'
      common / move1 / myid, jrange_min(MPE), jrange_max(MPE)
c
      dimension kind(N_p_p), list(N_p_p)
      dimension sbuf1(NBUFSZ), sbuf2(NBUFSZ), sbuf3(NBUFSZ),
     $          sbuf4(NBUFSZ), sbuf5(NBUFSZ), sbuf6(NBUFSZ),
     $          isbuf(NBUFSZ)
      dimension rbuf1(NBUFSZ,MPE), rbuf1_g(NBUFSZ,MPE),
     $          rbuf2(NBUFSZ,MPE), rbuf2_g(NBUFSZ,MPE),
     $          rbuf3(NBUFSZ,MPE), rbuf3_g(NBUFSZ,MPE),
     $          rbuf4(NBUFSZ,MPE), rbuf4_g(NBUFSZ,MPE),
     $          rbuf5(NBUFSZ,MPE), rbuf5_g(NBUFSZ,MPE),
     $          rbuf6(NBUFSZ,MPE), rbuf6_g(NBUFSZ,MPE),
     $          irbuf(NBUFSZ,MPE), irbuf_g(NBUFSZ,MPE)
      equivalence (rbuf1,rbuf1_g), (rbuf2,rbuf2_g), (rbuf3,rbuf3_g),
     $            (rbuf4,rbuf4_g), (rbuf5,rbuf5_g), (rbuf6,rbuf6_g),
     $            (irbuf,irbuf_g)
      dimension irecv_total(MPE), irecv_total_g(MPE)
      equivalence (irecv_total,irecv_total_g)
c
!xocl local  rbuf1  (:,/ipmpe), rbuf2  (:,/ipmpe), rbuf3  (:,/ipmpe)
!xocl local  rbuf4  (:,/ipmpe), rbuf5  (:,/ipmpe), rbuf6  (:,/ipmpe)
!xocl local  irbuf  (:,/ipmpe)
!xocl global rbuf1_g          , rbuf2_g          , rbuf3_g
!xocl global rbuf4_g          , rbuf5_g          , rbuf6_g
!xocl global irbuf_g
!xocl local  irecv_total  (/ipmpe)
!xocl global irecv_total_g
c
c-----------------------------------------------------------------------
c
c-----------------------------------------------------------------------
c
      iflag = 0
c
        ! itoward は転送先プロセッサ番号. まず自プロセッサ番号を
        ! 格納しておく.
      itoward = myid
c
c---
c
        ! 各粒子毎に粒子の種類を保存しておく.
      j = 0
      do 1000 is = 1, N_sp
         do 1010 i = l_st(is), l_ed(is)
            j = j + 1
            kind(j) = is
 1010    continue
 1000 continue
```

Fig. 2.7 Contents of new subroutine move (1/6)

```
c
c-----------------------------------------------------------------------
c
c-----------------------------------------------------------------------
c
        ! 転送先プロセッサを順に変えながら以下の処理を行う.
      do 2000 ir = 1, MPE-1
c
c---
c
        ! 転送先プロセッサの計算.  itoward の値は +1 されるが,
        ! MPE を越える場合には 1 がセットされる.
         itoward = mod(itoward, MPE) + 1
c
c---
c
        ! プロセッサ itoward に転送するべき粒子のインデックスを
        ! 配列 list に格納する. また, 転送すべき粒子数を isend_total
        ! に格納する.
         j = 0
         do 2100 i = 1, N_p_t
            kpy = y(i) * odlt_yg + 0.500001d0
            if ( kpy.ge.jrange_min(itoward) .and.
     $           kpy.le.jrange_max(itoward) ) then
               j = j + 1
               list(j) = i
            endif
 2100    continue
c
         isend_total = j
c
c---
c
        ! プロセッサ間で isend_total の最大値 isend_total_max を
        ! 見つける.
         isend_total_max = 0
!xocl spread do
         do 2200 mp = 1, MPE
            isend_total_max = max(isend_total_max, isend_total)
 2200    continue
!xocl end spread max(isend_total_max)
c
c---
c
        ! isend_total_max が 0 なら転送の必要なし.
         if ( isend_total_max.eq.0 ) goto 2000
c
c---
c
        ! これから転送する粒子数を転送先のプロセッサに通知する.
!xocl spread do
         do 2110 mp = 1, MPE
            irecv_total_g(itoward) = isend_total
 2110    continue
!xocl end spread
c
c---
c
```

Fig. 2.7 Contents of new subroutine move (2/6)

```
        ! icapacity は受けとることができる粒子数. これから転送
        ! する粒子数も考慮して計算する.
         icapacity = N_p_p - ( N_p_t - isend_total )
c
         if ( irecv_total(myid).gt.icapacity ) then
            ifail = 1
         else
            ifail = 0
         endif
c
c---
c
        ! 転送されてくる粒子を全て受けとることができないプロセッ
        ! サが 1 台でもあれば全てのプロセッサは処理を中断する.
        ! これまでに転送した粒子データの整理を行った後, 呼び出し
        ! 元ルーチンに復帰する.
         ifail_t = 0
!xocl spread do
         do 2150 mp = 1, MPE
            ifail_t = ifail_t + ifail
 2150    continue
!xocl end spread sum(ifail_t)
c
         if ( ifail_t.gt.0 ) then
            write(*,*)
            write(*,*) 'particle data overflow'
            write(*,*)
!xocl spread do
            do 2160 mp = 1, MPE
               if ( ifail.ne.0 ) then
                  write(*,*)
     \               'pe-id, N_p_p, N_p_t, isend_total, irecv_total = ',
     \                mp   , N_p_p, N_p_t, isend_total, irecv_total(mp)
               endif
 2160       continue
!xocl end spread
            iflag = 1
            goto 2010
         endif
c
c---
c
         igap_min = 1
         igap_max = 0
         igaps = 0
         last = N_p_t
c
        ! 転送用バッファの大きさに応じて転送処理を繰り返す.
         do 2300 iioffset = 0, isend_total_max-1, NBUFSZ
c
c---
c
        ! isend は, これから転送する粒子数.
            isend = max(0,min(isend_total-iioffset,NBUFSZ))
c
c---
c
```

Fig. 2.7 Contents of new subroutine **move** (3/6)

```
        ! 転送用バッファに粒子データを格納する.
            do 2310 j = 1, isend
               i = list(iioffset+j)
               sbuf1(j) = x(i)
               sbuf2(j) = y(i)
               sbuf3(j) = Pn(i,1)
               sbuf4(j) = Pn(i,2)
               sbuf5(j) = Pn(i,3)
               sbuf6(j) = gamma(i)
               isbuf(j) = kind(i)
 2310       continue
c
            igap_max = igap_max + isend
            igaps = igaps + isend
c
c---
c
        ! データ転送を行う. 各プロセッサはデータを送り出すと
        ! 同時に他のプロセッサからデータを受け取る.
!xocl spread do
            do 2315 mp = 1, MPE
!xocl spread move
               do 2320 i = 1, isend
                  rbuf1_g(i,itoward) = sbuf1(i)
                  rbuf2_g(i,itoward) = sbuf2(i)
                  rbuf3_g(i,itoward) = sbuf3(i)
                  rbuf4_g(i,itoward) = sbuf4(i)
                  rbuf5_g(i,itoward) = sbuf5(i)
                  rbuf6_g(i,itoward) = sbuf6(i)
                  irbuf_g(i,itoward) = isbuf(i)
 2320          continue
!xocl end spread (mw1)
!xocl movewait (mw1)
 2315       continue
!xocl end spread
c
c---
c
        ! 受け取ったデータ数の計算.
            irecv = max(0,min(irecv_total(myid)-iioffset,NBUFSZ))
c
c---
c
        ! 受けとったデータを, データを送り出したことによって
        ! 生じた隙間に格納する.
            insert = min(irecv, igaps)
c
*vocl loop,novrec
            do 2330 j = 1, insert
               i = list(igap_min+j-1)
               x(i)     = rbuf1(j,myid)
               y(i)     = rbuf2(j,myid)
               Pn(i,1)  = rbuf3(j,myid)
               Pn(i,2)  = rbuf4(j,myid)
               Pn(i,3)  = rbuf5(j,myid)
               gamma(i) = rbuf6(j,myid)
               kind(i)  = irbuf(j,myid)
 2330       continue
c
```

Fig. 2.7 Contents of new subroutine **move** (4/6)

```
            igap_min = igap_min + insert
            igaps = igaps - insert
c
c---
c
        ! 隙間に格納できないデータは，末尾に格納する.
            do 2340 j = insert+1, irecv
               last = last + 1
               i = last
               x(i)     = rbuf1(j,myid)
               y(i)     = rbuf2(j,myid)
               Pn(i,1)  = rbuf3(j,myid)
               Pn(i,2)  = rbuf4(j,myid)
               Pn(i,3)  = rbuf5(j,myid)
               gamma(i) = rbuf6(j,myid)
               kind(i)  = irbuf(j,myid)
 2340       continue
c
 2300    continue
c
c---
c
        ! データを送り出すことによって生じた隙間が残っていれば
        ! その隙間を詰める.
         do 2400 iii = 1, igaps
            ii = igap_min+iii-1
            i0 = list(ii)+1
            if ( iii.lt.igaps ) then
               i1 = list(ii+1)-1
            else
               i1 = last
            endif
c
            do 2410 i = i0, i1
               x(i-iii)     = x(i)
               y(i-iii)     = y(i)
               Pn(i-iii,1)  = Pn(i,1)
               Pn(i-iii,2)  = Pn(i,2)
               Pn(i-iii,3)  = Pn(i,3)
               gamma(i-iii) = gamma(i)
               kind(i-iii)  = kind(i)
 2410       continue
 2400    continue
c
c---
c
        ! 各プロセッサが担当する粒子数の更新.
         N_p_t = N_p_t - isend_total + irecv_total(myid)
c
 2000 continue
        ! 全データ転送処理の終了
 2010 continue
c
c-----------------------------------------------------------------------
c
c-----------------------------------------------------------------------
c
```

Fig. 2.7 Contents of new subroutine **move** (5/6)

```
        ! 粒子の種類をキーとして昇順ソートを行い，粒子の種類毎に
        ! データを並び替える.
      do 3000 j = 1, N_p_t-1
         if ( j.eq.1 .or. kind(j).ne.kind(j-1) ) then
            jm = j
            km = kind(j)
            do 3100 i = j+1, N_p_t
               if ( km.gt.kind(i) ) then
                  jm = i
                  km = kind(i)
               endif
 3100       continue
            if ( jm.ne.j ) then
               rtmp1 = x(jm)
               rtmp2 = y(jm)
               rtmp3 = Pn(jm,1)
               rtmp4 = Pn(jm,2)
               rtmp5 = Pn(jm,3)
               rtmp6 = gamma(jm)
               itmp  = kind(jm)
c
               x(jm)     = x(j)
               y(jm)     = y(j)
               Pn(jm,1)  = Pn(j,1)
               Pn(jm,2)  = Pn(j,2)
               Pn(jm,3)  = Pn(j,3)
               gamma(jm) = gamma(j)
               kind(jm)  = kind(j)
c
               x(j)     = rtmp1
               y(j)     = rtmp2
               Pn(j,1)  = rtmp3
               Pn(j,2)  = rtmp4
               Pn(j,3)  = rtmp5
               gamma(j) = rtmp6
               kind(j)  = itmp
            endif
         endif
 3000 continue
c
c-----------------------------------------------------------------------
c
c-----------------------------------------------------------------------
c
        ! 粒子の種類毎の範囲 l_st, l_ed を計算し直す.
      do 4000 is = 1, N_sp
         l_st(is) = 1
         l_ed(is) = 0
 4000 continue
c
      is = 1
      do 4100 i = 1, N_p_t
         if ( kind(i).ne.is ) then
            l_ed(is  ) = i - 1
            is = kind(i)
            l_st(is  ) = i
         endif
 4100 continue
      l_ed(is) = N_p_t
c
      end
```

Fig. 2.7 Contents of new subroutine move (6/6)

```
v         do 1300 is =  1,N_sp
v            coef = q(is)*qn(is)
m            do 1400 j = l_st(is), l_ed(is)
v               kpx = x(j)*odlt_xg + 0.500001d0
v               kpy = y(j)*odlt_yg + 0.500001d0
v               chrg1 = (dlt_xg-(x(j)-xg(kpx)))*(dlt_yg-(y(j)-yg(kpy)))
v               chrg2 = (x(j)-xg(kpx))*(dlt_yg-(y(j)-yg(kpy)))
v               chrg3 = (dlt_xg-(x(j)-xg(kpx)))*(y(j)-yg(kpy))
v               chrg4 = (x(j)-xg(kpx))*(y(j)-yg(kpy))
   c
m               rho(kpx  ,kpy  ) = rho(kpx  ,kpy  ) + chrg1*coef
m               rho(kpx+1,kpy  ) = rho(kpx+1,kpy  ) + chrg2*coef
m               rho(kpx  ,kpy+1) = rho(kpx  ,kpy+1) + chrg3*coef
m               rho(kpx+1,kpy+1) = rho(kpx+1,kpy+1) + chrg4*coef
v   1400     continue
v   1300  continue
```

Fig. 2.8 Part of original subroutine **charge**

```
          parameter(jjjskp=16)
          common /wwwwww/ wrho( ... , ... ,jjjskp)
            :
s         do 1010 jj=1,jjjskp
s2        do 1010 j=0,Ny+1
v2        do 1010 i=0,Nx+1
v2           wrho(i,j,jj)=0.d0
v2   1010 continue
s         do 1300 is =  1,N_sp
v            coef = q(is)*qn(is)
m               jjjend=l_ed(is)-l_st(is)+1
s            do 1400 jjj=1,jjjend,jjjskp
   *vocl loop,repeat(jjjskp)
v            do 1400 jj=1,min(jjjend-jjj+1,jjjskp)
v               j=jjj+jj-1
v               kpx = x(j)*odlt_xg + 0.500001d0
v               kpy = y(j)*odlt_yg + 0.500001d0
v               chrg1 = (dlt_xg-(x(j)-xg(kpx)))*(dlt_yg-(y(j)-yg(kpy)))
v               chrg2 = (x(j)-xg(kpx))*(dlt_yg-(y(j)-yg(kpy)))
v               chrg3 = (dlt_xg-(x(j)-xg(kpx)))*(y(j)-yg(kpy))
v               chrg4 = (x(j)-xg(kpx))*(y(j)-yg(kpy))
   c
v               wrho(kpx  ,kpy  ,jj) = wrho(kpx  ,kpy  ,jj) + chrg1*coef
v               wrho(kpx+1,kpy  ,jj) = wrho(kpx+1,kpy  ,jj) + chrg2*coef
v               wrho(kpx  ,kpy+1,jj) = wrho(kpx  ,kpy+1,jj) + chrg3*coef
v               wrho(kpx+1,kpy+1,jj) = wrho(kpx+1,kpy+1,jj) + chrg4*coef
v   1400     continue
v   1300  continue
s2        do 1310 jj=1,jjjskp
s2        do 1310 j=0,Ny+1
v2        do 1310 i=0,Nx+1
v2           rho(i,j)=rho(i,j)+wrho(i,j,jj)
v2   1310 continue
```

Fig. 2.9 Part of vectorized subroutine **charge**

```
         dimension Ex(0:Nx+1,0:Ny+1),Ey(0:Nx+1,0:Ny+1), ...
           :
s2       do 1000 j = 1, Ny
v2          do 1100 i = 1, Nx                          ←処理分割の対象
v2             Ex(i,j) = 0.5d0*( Akxro(i,j) + Akxroo(i,j) )
v2             Ey(i,j) = 0.5d0*( Akyro(i,j) + Akyroo(i,j) )
v2             Ez(i,j) = 0.5d0*( Akzro(i,j) + Akzroo(i,j) )
v2             wk1(i,j) = 0.5d0*( Akxio(i,j) + Akxioo(i,j) )
v2             wk2(i,j) = 0.5d0*( Akyio(i,j) + Akyioo(i,j) )
v2             wk3(i,j) = 0.5d0*( Akzio(i,j) + Akzioo(i,j) )
v2  1100    continue
s2  1000 continue
```

Fig. 2.10 Part of original subroutine EB_field

```
          dimension Ex(0:Ny+1,0:Nx+1),Ey(0:Ny+1,0:Nx+1), ...
            :
s2        do 1000 i = 1, Nx                              ←処理分割の対象
v2           do 1100 j = 1, Ny
v2              Ex(j,i) = 0.5d0*( Akxro(j,i) + Akxroo(j,i) )
v2              Ey(j,i) = 0.5d0*( Akyro(j,i) + Akyroo(j,i) )
v2              Ez(j,i) = 0.5d0*( Akzro(j,i) + Akzroo(j,i) )
v2              wk1(j,i) = 0.5d0*( Akxio(j,i) + Akxioo(j,i) )
v2              wk2(j,i) = 0.5d0*( Akyio(j,i) + Akyioo(j,i) )
v2              wk3(j,i) = 0.5d0*( Akzio(j,i) + Akzioo(j,i) )
v2  1100     continue
s2  1000  continue
```

Fig. 2.11 Part of vectorized subroutine EB_field

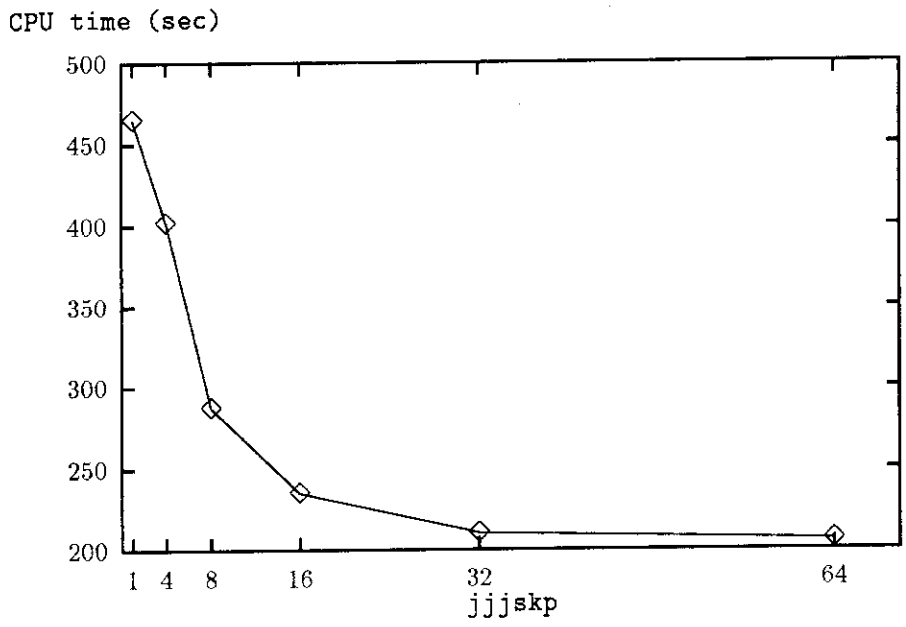


Fig. 2.12 Relation between parameter jjjskp and cpu time (user + sys)

```
&option
 dlt_t=0.001d0,n_time=250,load_type=2,rstrt=.f.
&end
&geom
 c=10.d0,system_l_2pi=.f.,
 Nx=512,Ny=128,system_lx=256.d0,system_ly=64.d0
&end
&diag
 int_snap=250, Nx_d = 128, Ny_d = 32
&end
&ions
 No_ion=0
&end
&eons
 p_mass_e=1.d0, q_e=-1.d0,
 M_e=2, No_eon=1, ney_func=5,
 ney0=1.d0, ney1=1.d0, ney2=1.d0, ney3=5.d0,
 Te_func=0, Te0=2.d0, Np_e=5
&end
&wave
 rLw = 5.d0, x0 = 10.d0, x1 = 35.d0, y1 = 32.d0,
 rLx = 15.d0, rLy = 8.d0, E0 = 113.73d0
&end
 M_e=2, No_eon=1, ney_func=5,
 ney0=1.d0, ney1=8.d0, ney2=4.d-1, ney3=10.d0,
```

Fig. 2.13 Input data

3. CYLDNS コードのベクトル並列化

3.1 概要

本作業では円筒座標系直接数値シミュレーションコード CYLDNS の富士通(株)製分散メモリ型ベクトル並列計算機 VPP500/42(以下 VPP500) 向き並列化作業を行った. 本コードは, 時間・空間発展する円管内の乱流挙動を直接的に数値解析で求めるコードである.

本作業では, まず ILUCGS 法 [1] による行列ソルバのベクトル化が不十分で計算コストが集中していることがわかっていたことから, このルーチンのベクトル化を行い, 次に領域分割と呼ばれる方法を用いて並列化を行った.

3.2 ベクトル化

本作業では, サブルーチン ILUCGS のみをベクトル化の対象とした. ベクトル化前のこのサブルーチンの内容を Fig. 3.1 に示す.

この図からわかるように, このサブルーチンにはいくつかベクトル計算できないループが含まれている. まず, これらをベクトル実行できるように変更し, また, ベクトル処理効率の向上を考慮して複数の DO ループを一つにまとめる作業を行った. ベクトル化後のこのサブルーチンの内容を Fig. 3.2 に示す. また, ベクトル化後のサブルーチンで参照される変数, 配列を定義するために作成したサブルーチン MKLIST の内容を Fig. 3.3 に示す.

以下, 作業の内容について述べる.

3.2.1 最適化制御行の利用

ここでは, DO 10 ループのベクトル化方法について述べる. このループ中の処理が完全にベクトル化されていない理由は, 変数 N の値によっては Fig. 3.1 の (a) と (b) が重複した領域に値を定義してしまう可能性があるためである.

このループのベクトル化方法には次の 2 通りが考えられる.

- DO 10 ループを 2 つのループに分割し, (a) と (b) を別々のループで処理する.
- 実際には, 変数 N には必ず正数が格納されており, (a) と (b) が重複した領域に値を定義することはない. そこで, 適当な指示行を挿入して, 重複しないことをコンパイラに通知する.

本作業では, 後者の方法を採ることに決め, ループの直前に次の指示行 (最適化制御行) を挿入した.

```
*VOCL LOOP,N.GT.0
```

3.2.2 Hyper Plane 法の利用

ここでは，DO 15，DO 20，DO 50，DO 60，DO 90，DO 100，DO 150，及び DO 160 の 8 個のループのベクトル化方法について述べる．

まず，DO 15，DO 20，DO 50，DO 90，及び DO 150 について示す．これらのループがベクトル化されていないのは，I 番目の要素を計算する時に，同じループ中で以前に計算した結果（1 番目から I-1 番目までの要素の値）を参照していることが原因である．例えば DO 15 ループの場合，DDD(I-1)，DDD(I-M1)，DDD(I-M2) を参照して DDD(I) の計算を行っている．このような処理は回帰計算と呼ばれ，ベクトル計算することはできない．そこで，Hyper Plane 法 [1] によって計算の順序を変更することで回帰参照を回避し，ベクトル計算を可能にすることにした．

本サブルーチンは，二次元平面（メッシュ数： M2×N/M2）に対して 7 点差分を適用して得られた連立一次方程式を解く処理を行っており，元々二次元配列に格納していた値を一次元配列にコピーして使用している．M1=M2-1 を考慮して上記の 4 つの配列要素の位置関係を元々の二次元平面上で考えると，Fig. 3.4 のようになる．また，DDD(I-M1) と掛け合わされる係数 AAA(I,2) には，Fig. 3.4 (a) に示す位置関係にある時，常に 0.0 が格納されている．

したがって，Fig. 3.5 に示すように計算すれば，個々の線分上の要素の計算はベクトル実行することが可能となる．この時，DDD(I) の計算時に参照される DDD(I-1) と DDD(I-M2) の値は直前の DO ループ中で既に計算されており，回帰参照は発生しない．一方，DDD(I-M1) については，Fig. 3.4 (a) の位置関係にある時 DDD(I) と同一の線分上に存在するため回帰参照となってしまう．しかし，この時 DDD(I-M1) と掛け合わされる係数の値が 0.0 であることから，強引にベクトル計算しても正しい結果を得ることができる．

その他の DO 60，DO 100，DO 160 についても，逆順に計算が行われているだけで，同様なことがいえる．DO 60 ループの場合を例に示すと，RRR(I)，RRR(I+1)，RRR(I+M1)，及び RRR(I+M2) の二次元平面における位置関係は Fig. 3.6 のようになり，Fig. 3.6 (a) の場合，RRR(I+M1) に掛け合わされる係数 AAA(I,6) の値は常に 0.0 である．したがって，計算する線分の順序を逆順とすれば，DO 15 ループなどの場合と同様な方法でベクトル化することができる．

実際のベクトル化は次のように行った．まず，サブルーチン MKLIST を新規に作成し，Hyper Plane 法に従って計算するためのリストを作成することにした．このリストの作成は一度だけでよいので，MAIN プログラムから一度だけ呼び出されるようにした．作成したリストは，コモン HYPER を介して ILUBCG サブルーチンに渡される．このコモン HYPER のメンバについて Table 3.1 に示す．

次に，サブルーチン MKLIST で計算したリストに従って計算が行われるように，サブルーチン ILUCGS 中の DO ループを変更した．また，個々のループの直前に，ベクトル実行してもよいことを示す*最適化制御行「*VOCL LOOP,NOVREC(...)」（... は配列名）を挿入した．

* 実際には，指定された配列の定義・引用関係に，回帰がないことを示す．

3.2.3 DO ループの融合

既にベクトル化された複数のループを一つのループにまとめておくと，メモリアクセスの効率などが改善され，ベクトル処理効率が若干向上することがある．本サブルーチンの場合，DO 30 ループと DO 40 ループ，及び 120 ループと DO 130 ループをそれぞれ一つのループにまとめることが可能である．そこで，これらのループをそれぞれ一つのループにまとめた．

3.3 並列化

3.3.1 並列化の方針

本コードの並列化では，「領域分割」と呼ばれる手法を用いることにした．「領域分割」とは，解析空間をプロセッサ台数個に分割し，個々の領域をそれぞれのプロセッサに担当させる手法である．各プロセッサの担当する領域の大きさを均等にしておくことで，各プロセッサが負担するデータ量及び処理量をほぼ均等に分散させることができる．この解析空間の分割方法には様々な方法が考えられるが，本コードの並列化では，分割軸を一つだけ選んで分割する方法を採ることにする．その理由は，単純な分割方法を選んでおく方が並列化作業が容易になること，及びその他の分割方法を採用する必要性が特にないためである．以下，どの軸を分割軸とするかについて述べる．

本コードの特筆すべき処理として，I 軸方向の一次元 FFT 計算を行っていること，及び，I の値ごとに（各 J-K 平面ごとに）独立な連立一次方程式を作成し，これを ILUCGS 法で解いていることが挙げられる．前者の計算部分では K 軸方向と J 軸方向のそれぞれに並列性があるが，K 軸方向はすでにベクトル計算に用いられているため並列計算には J 軸方向の並列性を利用することにする．一方，後者の計算部分では I 軸方向に並列性がある．このように，両者に適した分割軸が異なるため，両者の計算の間にデータの分割軸を変更するための処理（以下，「転置」）が必要となる．

この処理は，実際には分割軸の異なる配列にデータをコピーすることで実現する．したがって，プロセッサ間のデータ転送が伴うので，転置の回数はできるだけ少ない方が効率がよい．本コードの場合，1 時間ステップ当たり数回の FFT 計算が行われるのに対して，ILUCGS 法による計算部分は 1 回しか実行されない．そこで，通常は FFT 計算に都合のよい J 軸方向を分割しておき，ILUCGS 法による計算部分の直前，直後で転置を行うことにする．

なお，本コードでは，個々のプロセッサが自分に割り当てられた領域を越えて，両隣のプロセッサが担当する領域（分割面の近傍）のデータを参照することがある．この参照が正常に行われるように，このような参照の対象となるデータを「袖」付きで分割しておくことにする．「袖」とは，分割面に冗長な領域を確保したものであり，この領域に，両隣のプロセッサから分割面近傍のデータを転送しておく（袖転送）ことで，本来の担当領域を越えるデータの参照が可能となる．

3.3.2 分割指定子の定義

分割指定子は，データと処理の分割で利用し，個々のプロセッサがどの領域を担当するかを定めたものである．本コードの並列化では，Fig. 3.7 に示す 6 種類の分割指定子を定義した．

図の 1 行目で定義される分割指定子 ipj は，-3 から NJG の範囲を NPE 台のプロセッサに均等に分担させるためのもので，処理とデータを J 軸方向に分割する場合に利用する．この分割指定子を使用すると，例えば NPE=4，NJG=67 の場合，各プロセッサが担当する範囲は Table 3.2 のようになる．

また，3 行目で定義される ipi は，0 から NIG-4 の範囲を NPE 台のプロセッサに「循環」的に担当させるために使用する．例えば NPE=4，NIG=67 の場合，各プロセッサには Table 3.3 のように割り当てが行われる．この分割指定子は，I 軸方向の分割で用いる．循環分割を適用した理由については，「3.3.5.4 循環分割の利用」で述べる．

その他の分割指定子は，データを袖付きで分割するために使用する．例えば NPE=4，NJG=67 の場合，ipj11 を使用すると各プロセッサには Table 3.4 のようにデータが割り当てられる．この表の括弧内に示した要素が袖に相当する．この袖部分は，両隣のプロセッサの担当する領域と重複しており，袖転送によって両隣のプロセッサからデータをコピーしてくることが可能である．その結果，各プロセッサは，自身が担当する領域を越えてデータ参照できるようになる．

3.3.3 データの分割

本コードの並列化では，主に解析空間全体のデータを格納し，非常に大きなメモリ領域を必要とする配列のみを分割の対象とすることにした．これは，小さな配列を分割しても 1 プロセッサあたりのメモリ量の削減にはそれほど効果がないこと，及び，分割するデータの個数を少なくしておくことで並列化作業を容易とするためである．ただし，ファイル入出力の対象となるような配列の中には，分割した方がかえって並列化作業が容易となるものがあり，このような配列に対しては特に分割の対象とすることにした．

次に，分割の対象とする配列について，プログラム中でどのように参照されているかを調べて，袖が必要かどうかを決定した．

また，データ転送（袖転送も含む）の対象となる配列，及びファイル入出力の対象となる配列について，EQUIVALENCE によってグローバル配列との結合を行った．データ転送の対象となる配列は，VPP FORTRAN77EX/VPP の仕様によりグローバル配列と結合しておく必要があるためであり，ファイル入出力の対象となる配列については，グローバル配列名を用いて入出力を行うためである．

3.3.4 処理の分割

基本的な処理分割の方法は，各プロセッサが自分に割り当てられた領域のデータのみをアクセスするように DO ループの処理を分割することである．例えば，J 軸方向を分割の対象としている場合，J 方向の DO ループが処理分割の対象となる．この DO 文の直前に最適化制御行 `!xocl spread do /ipj` を挿入しておくと，分割指定子 ipj に従って J のループが分割され，各プロセッサは自分に割り当てられた範囲だけを処理するようになる．

なお，spread do による並列計算の開始と終了には，若干のオーバヘッド時間が必要となる．このオーバヘッド時間をできるだけ小さく抑えるため，可能な限りループの入れ換えを行って，分割の対象としたループが他のループの外側に位置するようにした．

以下，特別な方法を用いて行った処理分割について示す．

3.3.4.1 境界計算部分の処理分割

本コードの並列化では領域分割という方法を用いているため，境界計算を行っている箇所では，境界を担当しているプロセッサだけに処理を行わせる工夫が必要となる．以下，境界計算部分の分割方法について Fig. 3.8 を例に述べる．

この例でアクセスされる配列 P は，J 軸方向が分割の対象となっている．したがって，I 軸方向の両端の境界計算（DO 77）と K 軸方向の場合（DO 7777）については，特別な並列化は必要ない（Fig. 3.9 参照）．

それに対して，J 軸方向の両端の境界計算（DO 777）の場合には，まず両端を担当するプロセッサだけに計算を行わせる工夫が必要となる．そこで，まず DO 777 の外側に J 軸方向の DO ループを挿入し，かつ IF 文を用いて，J=1 または J=JG を担当するプロセッサだけに処理を行わせるようにした．

さらに，この例では J=1 の要素を J=0 の要素に，J=JG の要素を J=JG+1 の要素にコピーする処理を行っている．J=1 と J=0，または J=JG と J=JG+1 の要素を同一のプロセッサが担当しているとは限らないため，正常に動作しない場合が考えられる．そこで，Fig. 3.9 に示すように，プロセッサ間のデータ転送を利用する方法を用いた．

3.3.4.2 ブロードキャストの利用

並列化前のサブルーチン TAIRYU の内容を Fig. 3.10 に示す．このサブルーチンでアクセスされる配列 U は J 軸方向が分割された配列であり，かつ J=JG-1 の要素だけが参照されているので，境界計算部分の並列化と同様に特定のプロセッサだけに処理を行わせる必要がある．ところが，ここで値が定義される配列 ATAI は重複ローカル配列であり，処理を行ったプロセッサだけが正しい計算結果を持つことになってしまう．そこで，処理を行ったプロセッサから，他のプロセッサが保持する重複ローカル配列 ATAI の領域に対してデータ転送を行うことにした．ここで利用したのが，次の指示行である．

!xocl broadcast（配列名）（フラグ変数名）

この指示行は，フラグの値が真であるプロセッサから，その他のプロセッサに対して，指定したデータの内容をコピーすることを示す．

並列化後の TAIRYU の内容を Fig. 3.11 に示す．

3.3.5 その他

以下，プロセッサ間のデータ転送に関するもの，及びその他の作業内容について示す．

3.3.5.1 転置

分割軸の変更が必要となるのは，サブルーチン SPERMG 中である．並列化後のこのサブルーチンの概要を Fig. 3.12 に示す．

このサブルーチンには，I 軸方向を分割した作業配列 WK1 と WK2 が宣言されている．これらの配列に J 軸方向を分割した配列から必要なデータをコピーした後，ILUCGS 計算部分を処理して，結果を J 軸方向を分割した配列に戻す処理が行われる．

3.3.5.2 袖転送

袖転送は，最適化制御行 !xocl overlapfix が挿入された位置で，指定された配列を対象として実行される．本作業では，まず袖付き分割を行った配列についてプログラム中での定義・引用関係を調査し，Fig. 3.13 に示す図を作成した．次に，この図を基に袖転送を行う箇所を決定し，プログラム中に上記の最適化制御行を挿入した．

3.3.5.3 上位展開

VPP FORTRAN77/VPP の仕様では，最適化制御行 !xocl spread do で分割された DO ループ中から他の手続きを呼び出し，下位の手続き中で分割された配列をアクセスすることを許していない．Table 3.5 に示す手続き間に，この仕様に抵触する部分があったため，これらを変更する作業を行った．

例として，サブルーチン SPERMG と FFTS1 の場合について示す．これらを単純に並列化したものを Fig. 3.14 に示す．この図において，FFTS1 は上位の SPERMG 中の spread do で分割されたループから呼び出されており，また FFTS1 でアクセスされる配列の CP と CP2 は分割された配列である．このため上記の仕様に抵触し，コンパイルは正常に終了するものの，実行時に誤った処理を行ってしまう．そこで，下位の手続き FFTS1 の処理を上位の手続き中に記述して，FFTS1 を呼び出さないようにした．正しく動作する並列化版を Fig. 3.15 に示す．

3.3.5.4 循環分割の利用

ILUCGS 法による計算部分では，I の値と ILUCGS の反復回数との間に Fig. 3.16 に示すような関係がある．したがって，I 軸方向を単純に分割した場合に各プロセッサの処理量が均一とならず，並列化効率に悪影響を与えてしまう．そこで，I 軸方向の分割に循環分割を利用して，並列化効率の改善を図ることにした．

3.3.6 ファイル入出力の高速化

本コードは，元々テキスト形式のリスタートデータを入力として動作し，同様にテキスト形式のリスタートデータを出力して終了するようになっていた．このリスタートデータは比較的規模が大きく，並列化後の計算時間に大きく影響してしまうことが考えられたため，リスタートデータをバイナリ形式に変更し，かつ一括して入出力するようにした．このようにすることで，リスタートデータ自体の大きさを縮小し，また内部形式 - 10 進形式の変換に要する時間を節約し，さらに一括して処理することで入出力に要するオーバヘッド時間の短縮を図った．

3.4 計算結果の評価

本作業の結果，並列化前後の計算結果に若干の違いが見られた．以下では，まず計算結果に影響を与えた要因について述べる．また，これらの要因を取り除いて計算結果の評価を行い，本作業に問題がなかったことを確認したことについて述べる．

3.4.1 計算結果に影響する要因

本コードの場合，次の 2 つの要因が考えられる．

VPP500 の個々のプロセッサは，ベクトル演算を行うベクトルユニットとスカラ演算を行うスカラユニットを持つ．通常，ベクトルユニットでは浮動小数点演算時の端数が「切捨て」で処理されるのに対して，スカラユニットでは「四捨五入」で処理される．このため，ベクトル化作業によって，それまでスカラ計算で処理されていた箇所をベクトル計算できるようにすると，計算結果に微妙な影響が現われる．

また，コンパイラによる最適化方法の中には演算結果に微妙に影響を与えてしまうものが含まれている．本作業に伴うソースプログラムの変更によってコンパイラによる最適化方法が影響を受けた結果，計算結果に違いが現われたことが考えられる．

3.4.2 確認方法

前節で述べた計算結果に影響を与える要因は，次のようにすることで取り除くことができる．

- 端数の処理方法の違いによる影響は，サービスサブルーチン CHROUND を利用してスカラユニットの処理方法を「切捨て」に変更することで取り除くことができる．
- コンパイラの最適化による影響は，演算結果に影響する最適化を抑止するためのコンパイルオプション `-Os,-E` を利用することで取り除くことができる．

そこで，作業前のソースプログラムと作業後のソースプログラムの両者に対して上記の処置を施して計算結果を比較したところ，完全に一致する結果が得られた．よって，本作業には特に問題点がなかったものと判断した．

なお，計算結果の評価の後，サービスサブルーチン CHROUND の呼び出しとコンパイラオプション `-Os,-E` は削除した．

3.5 効果

作業前，ベクトル化後，及び並列化後のソースプログラムを用いて実行した時の計算時間を Table 3.6 に示す．計測には `timex` コマンドを使用した．この表から，ベクトル化によって 2.12 倍程度に高速化されていることがわかる（`user` 時間 + `sys` 時間による比較）．また，ベクトル化後の計算時間を基準として，4 プロセッサで並列実行した時の速度向上は 2.83 倍程度（並列化効率 70.7%），16 プロセッサでは 5.91 倍程度（並列化効率 36.9%）となった．ここで，並列化効率は式 (3.1) を用いて計算した．

$$\text{並列化効率}\ (\%) = \frac{\text{実時間の比較による速度向上}}{\text{プロセッサ台数}} \times 100 \tag{3.1}$$

上記の結果では良好な並列化効率が得られていないが，その原因として次のようなことが考えられる．本作業で用いたデータは，計算の規模が比較的小さい（`64x64x64`）．したがって，計算の粒度も非常に小さく，プロセッサ間の同期などによるオーバヘッド時間が相対的に大きくなってしまっていることが考えられる．また，本作業ではファイル入出力をバイナリ化すること

で，ファイル入出力に要する処理時間の短縮を図ったが，それでも計算時間と比較してファイル入出力に要する時間が大きく，並列化効率に悪い影響を与えていると思われる．

3.6 まとめ

本作業では，CYLDNS コードに対して Hyper Plane 法を利用したサブルーチン ILUCGS のベクトル化，及び VPP500 向け並列化を行った．ベクトル化については，ILUCGS の計算コストが元々大きかったこともあって，コード全体の計算時間を 1/2 程度に短縮することができた．並列化については，用いたデータの計算の規模が十分に大きくなかったため，良好な並列化効率は得られなかった．しかし，データ分割によって，1 プロセッサでは実行できない大規模な計算が可能となった．

Table 3.1 New common block "HYPER"

No.	名前	型	要素数	備考
1	LPMAX	i*4	—	線分の総数
2	LDX	i*4	≥ JG+KG	各線分上に存在する点が，IDX のどの範囲に格納されているかを示す
3	IDX	i*4	≥ JG*KG	リスト配列

Table 3.2 Charge for each processor in the case NPE=4, NJG=67

プロセッサ番号	担当する範囲
1	-3 ～ 14
2	15 ～ 32
3	33 ～ 50
4	51 ～ 67

Table 3.3 Charge for each processor when dividing cyclically

プロセッサ番号	担当する範囲
1	0, 4, 8, ..., 56, 60
2	1, 5, 9, ..., 57, 61
3	2, 6, 10, ..., 58, 62
4	3, 7, 11, ..., 59, 63

Table 3.4 Charge for each processor when dividing with overlap

プロセッサ番号	担当する範囲
1	-3 ～ 14, (15)
2	(14), 15 ～ 32, (33)
3	(32), 33 ～ 50, (51)
4	(50), 51 ～ 67

Table 3.5 Subroutines to be folded into the above

上位のルーチン	下位のルーチン
SPERMG	FFTS1
	FFTSV1
SPET4S	FFTSH
SPEU4S	FFTSVH
SPEV4S	
SPEW4S	

Table 3.6 Execution time of the original, the vectorized and the parallelized code

	real	user	sys	vu-user
オリジナル	1624.48s	1436.41s	10.14s	101.83s
ベクトル化後	788.45s	671.85s	9.95s	558.31s
並列化後(4PE)	278.83s	1071.94s	2.15s	661.65s
並列化後(16PE)	133.41s	1422.33s	2.35s	807.86s

```
          SUBROUTINE ILUCGS(N,N1,N2,M1,M2,EPS,ITR,IER,S)
      ************************************************************************
      *  INCOMPLETE LU DECOMPOSITION CONJUGATED GRADIENT SQUARED METHOD FOR  *
      *      FINITE DIFFERENCE METHOD.                                       *
      *  COPYRIGHT    T. OGUNI     JUNE 30 1989    VERSION 1.0               *
      *  MODIFIED   S.SATAKE       OCT  15 1993    VERSION 1.0               *
      ************************************************************************
           IMPLICIT REAL*8(A-H,O-Z)
          COMMON /AAMAT/AAA(-63:4160,7)
          COMMON /BMAT/BBB(4096),XXX(-63:4160),DDD(-63:4160)
         &,WWW(-63:4160),PPP(-63:4160),QQQ(-63:4160),RRR(-63:4160),RO(4096)
         &,EEE(4096),HHH(4096)
      C
      C
           TH = 1.0D0
           IF (S .GT. 0.0 .AND. S .LT. 1.0) THEN
            TH = S
            S = 1.0D0
           ENDIF
m          DO 10 I=1-M2,0
v           DDD(I) = 0.0D0
s4          XXX(I) = 0.0D0                          -+
s4          PPP(I) = 0.0D0                           |
s4          QQQ(I) = 0.0D0                           | (a)
s4          RRR(I) = 0.0D0                           |
s4          WWW(I) = 0.0D0                          -+
s4          WWW(I+N+M2) = 0.0D0                     -+
s4          PPP(I+N+M2) = 0.0D0                      |
s4          QQQ(I+N+M2) = 0.0D0                      | (b)
s4          RRR(I+N+M2) = 0.0D0                      |
s4       10 XXX(I+N+M2) = 0.0D0                     -+
v          DO 11 I=1,N
v        11 DDD(I) = 0.0D0
      C  INCOMPLETE CHOLESKY DECOMPOSITION
           IF (S .NE. 0.0) THEN
m           DO 15 I=1,N
m4         SS=S*AAA(I,4)-AAA(I,3)*(AAA(I-1,5)+(AAA(I-1,6)+AAA(I-1,7))*TH)
          &*DDD(I-1)-AAA(I,2)*(AAA(I-M1,6)+(AAA(I-M1,5)+AAA(I-M1,7))*TH)
          &*DDD(I-M1)-AAA(I,1)*(AAA(I-M2,7)+(AAA(I-M2,6)+AAA(I-M2,5))*TH)
          &*DDD(I-M2)
m4       15   DDD(I) = 1.0D0 / SS
           ELSE
s2          DO 20 I=1,N
m2         SS =AAA(I,4)-AAA(I,3)*AAA(I-1,5)*DDD(I-1)-AAA(I,2)*AAA(I-M1,6)
```

Fig. 3.1 Subroutine ILUCGS (original, 1/3)

```
        &*DDD(I-M1)-AAA(I,1)*AAA(I-M2,7)*DDD(I-M2)
m2     20   DDD(I) = 1.0D0 / SS
          ENDIF
v         DO 30 I=1,N
v      30 QQQ(I)=AAA(I,1)*XXX(I-M2)+AAA(I,2)*XXX(I-M1)+AAA(I,3)*XXX(I-1)
        &+AAA(I,4)*XXX(I)+AAA(I,5)*XXX(I+1)+AAA(I,6)*XXX(I+M1)
        &+AAA(I,7)*XXX(I+M2)
v         DO 40 I=1,N
v      40  RRR(I) = BBB(I) - QQQ(I)
s3        DO 50 I=1,N
s3     50  RRR(I) = DDD(I)*(RRR(I)-AAA(I,3)*RRR(I-1)-AAA(I,2)
        &*RRR(I-M1)-AAA(I,1)*RRR(I-M2))
s3        DO 60 I=N,1,-1
s3     60  RRR(I) = RRR(I)-DDD(I)*(AAA(I,5)*RRR(I+1)+AAA(I,6)
        &*RRR(I+M1)+AAA(I,7)*RRR(I+M2))
   C      CAUTION !!
          C1 = 0.0D0
v         DO 70 I=1,N
v          R0(I) = RRR(I)
v          PPP(I) = RRR(I)
v          EEE(I) = RRR(I)
v      70  C1 = C1 + RRR(I) * RRR(I)
   C   ITERATION PHASE
          DO 200 K=1,ITR
v          DO 80 I=1,N
v      80 QQQ(I)=AAA(I,1)*PPP(I-M2)+AAA(I,2)*PPP(I-M1)+AAA(I,3)*PPP(I-1)
        &+AAA(I,4)*PPP(I)+AAA(I,5)*PPP(I+1)+AAA(I,6)*PPP(I+M1)
        &+AAA(I,7)*PPP(I+M2)
s3         DO 90 I=1,N
s3     90 QQQ(I)=DDD(I)*(QQQ(I)-AAA(I,3)*QQQ(I-1)-AAA(I,2)*QQQ(I-M1)
        &-AAA(I,1)*QQQ(I-M2))
s3         DO 100 I=N,1,-1
s3    100 QQQ(I)=QQQ(I)-DDD(I)*(AAA(I,5)*QQQ(I+1)+AAA(I,6)*QQQ(I+M1)
        &+AAA(I,7)*QQQ(I+M2))
           C2 = 0.0D0
v          DO 110 I=1,N
v     110  C2=C2+QQQ(I)*R0(I)
   C      CAUTION !!
          ALPHA = C1 / C2
          C3 = 0.0D0
          X1 = 0.0D0
          X2 = 0.0D0
v         DO 120 I=1,N
v          HHH(I) = EEE(I) - ALPHA * QQQ(I)
```

Fig. 3.1 Subroutine ILUCGS (original, 2/3)

```
v     120  CONTINUE
v          DO 130 I=1,N
v     130   WWW(I) = EEE(I) + HHH(I)
v          DO 140 I=1,N
v     140 QQQ(I)=AAA(I,1)*WWW(I-M2)+AAA(I,2)*WWW(I-M1)+AAA(I,3)*WWW(I-1)
         &          +AAA(I,4)*WWW(I)+AAA(I,5)*WWW(I+1)+AAA(I,6)*WWW(I+M1)
         &          +AAA(I,7)*WWW(I+M2)
s3         DO 150 I=1,N
s3    150 QQQ(I)=DDD(I)*(QQQ(I)-AAA(I,3)*QQQ(I-1)-AAA(I,2)*QQQ(I-M1)
         &-AAA(I,1)*QQQ(I-M2))
s3         DO 160 I=N,1,-1
s3    160 QQQ(I)=QQQ(I)-DDD(I)*(AAA(I,5)*QQQ(I+1)+AAA(I,6)*QQQ(I+M1)
         &+AAA(I,7)*QQQ(I+M2))
v          DO 170 I=1,N
v           Y = XXX(I)
v           RRR(I)=RRR(I)-ALPHA*QQQ(I)
v           XXX(I)=XXX(I)+ALPHA*WWW(I)
v           C3 = C3 + RRR(I)*RO(I)
v           X1 = X1 + Y*Y
v     170   X2 = X2 + (XXX(I) - Y)**2
           IF (X1 .NE. 0.0) THEN
            RES = DSQRT(X2 / X1)
            IF (RES .LE. EPS) THEN
             ITR = K
             IER = 0
             EPS = RES
             IF (TH .NE. 1.0D0) S = TH
             RETURN
            ENDIF
           ENDIF
           BETA = C3 / C1
           C1 = C3
v          DO 180 I=1,N
v         EEE(I)=RRR(I)+BETA*HHH(I)
v         PPP(I)=EEE(I)+BETA*(HHH(I)+BETA*PPP(I))
v     180 CONTINUE
   C
     200 CONTINUE
         IER = 1
         WRITE(*,*) '(SUBR. ILUCGS) NO CONVERGENCE. '
         EPS = RES
         IF (TH .NE. 1.0D0) S = TH
         RETURN
         END
```

Fig. 3.1 Subroutine ILUCGS (original, 3/3)

```
      SUBROUTINE ILUCGS(N,N1,N2,M1,M2,EPS,ITR,IER,S)
*************************************************************************
*  INCOMPLETE LU DECOMPOSITION CONJUGATED GRADIENT SQUARED METHOD FOR  *
*      FINITE DIFFERENCE METHOD.                                       *
*  COPYRIGHT    T. OGUNI     JUNE 30 1989    VERSION 1.0               *
*  MODIFIED   S.SATAKE       OCT  15 1993    VERSION 1.0               *
*************************************************************************
       IMPLICIT REAL*8(A-H,O-Z)
      COMMON /AAMAT/AAA(-63:4160,7)
      COMMON /BMAT/BBB(4096),XXX(-63:4160),DDD(-63:4160)
     &,WWW(-63:4160),PPP(-63:4160),QQQ(-63:4160),RRR(-63:4160),R0(4096)
     &,EEE(4096),HHH(4096)
CV    DIMENSION LDX(1:LPMAX+1), IDX(1:N)
CV      (WHERE KG=M2, JG=N/M2, LPMAX=KG+JG-1, N=KG*JG)
      COMMON /HYPER/ LPMAX,LDX(128),IDX(4096)
C
C
      TH = 1.0D0
      IF (S .GT. 0.0 .AND. S .LT. 1.0) THEN
       TH = S
       S = 1.0D0
      ENDIF
*VOCL LOOP,N.GT.0
v       DO 10 I=1-M2,0
v        DDD(I) = 0.0D0
v        XXX(I) = 0.0D0
v        PPP(I) = 0.0D0
v        QQQ(I) = 0.0D0
v        RRR(I) = 0.0D0
v        WWW(I) = 0.0D0
v        WWW(I+N+M2) = 0.0D0
v        PPP(I+N+M2) = 0.0D0
v        QQQ(I+N+M2) = 0.0D0
v        RRR(I+N+M2) = 0.0D0
v    10  XXX(I+N+M2) = 0.0D0
v       DO 11 I=1,N
v    11  DDD(I) = 0.0D0
C  INCOMPLETE CHOLESKY DECOMPOSITION
      IF (S .NE. 0.0) THEN
CV      DO 15 I=1,N
s       DO 15 LP=1,LPMAX
*VOCL LOOP,NOVREC(DDD)
v       DO 15 L=LDX(LP)+1,LDX(LP+1)
v         I=IDX(L)
```

Fig. 3.2 Subroutine ILUCGS (vectorized, 1/4)

```
v          SS=S*AAA(I,4)-AAA(I,3)*(AAA(I-1,5)+(AAA(I-1,6)+AAA(I-1,7))*TH)
          &*DDD(I-1)-AAA(I,2)*(AAA(I-M1,6)+(AAA(I-M1,5)+AAA(I-M1,7))*TH)
          &*DDD(I-M1)-AAA(I,1)*(AAA(I-M2,7)+(AAA(I-M2,6)+AAA(I-M2,5))*TH)
          &*DDD(I-M2)
v       15   DDD(I) = 1.0D0 / SS
           ELSE
CV            DO 20 I=1,N
s           DO 20 LP=1,LPMAX
*VOCL LOOP,NOVREC(DDD)
v           DO 20 L=LDX(LP)+1,LDX(LP+1)
v             I=IDX(L)
v           SS =AAA(I,4)-AAA(I,3)*AAA(I-1,5)*DDD(I-1)-AAA(I,2)*AAA(I-M1,6)
          &*DDD(I-M1)-AAA(I,1)*AAA(I-M2,7)*DDD(I-M2)
v       20   DDD(I) = 1.0D0 / SS
           ENDIF
v          DO 30 I=1,N
CV      30 QQQ(I)=AAA(I,1)*XXX(I-M2)+AAA(I,2)*XXX(I-M1)+AAA(I,3)*XXX(I-1)
v          QQQ(I)=AAA(I,1)*XXX(I-M2)+AAA(I,2)*XXX(I-M1)+AAA(I,3)*XXX(I-1)
          &+AAA(I,4)*XXX(I)+AAA(I,5)*XXX(I+1)+AAA(I,6)*XXX(I+M1)
          &+AAA(I,7)*XXX(I+M2)
CV            DO 40 I=1,N
v       40  RRR(I) = BBB(I) - QQQ(I)
v       30 CONTINUE
CV            DO 50 I=1,N
s           DO 50 LP=1,LPMAX
*VOCL LOOP,NOVREC(RRR)
v           DO 50 L=LDX(LP)+1,LDX(LP+1)
v             I=IDX(L)
v       50  RRR(I) = DDD(I)*(RRR(I)-AAA(I,3)*RRR(I-1)-AAA(I,2)
          &*RRR(I-M1)-AAA(I,1)*RRR(I-M2))
CV            DO 60 I=N,1,-1
s           DO 60 LP=LPMAX,1,-1
*VOCL LOOP,NOVREC(RRR)
v           DO 60 L=LDX(LP)+1,LDX(LP+1)
v             I=IDX(L)
v       60  RRR(I) = RRR(I)-DDD(I)*(AAA(I,5)*RRR(I+1)+AAA(I,6)
          &*RRR(I+M1)+AAA(I,7)*RRR(I+M2))
C       CAUTION !!
            C1 = 0.0D0
v           DO 70 I=1,N
v            R0(I) = RRR(I)
v            PPP(I) = RRR(I)
v            EEE(I) = RRR(I)
v       70  C1 = C1 + RRR(I) * RRR(I)
```

Fig. 3.2 Subroutine ILUCGS (vectorized, 2/4)

```
      C  ITERATION PHASE
            DO 200 K=1,ITR
v            DO 80 I=1,N
v        80 QQQ(I)=AAA(I,1)*PPP(I-M2)+AAA(I,2)*PPP(I-M1)+AAA(I,3)*PPP(I-1)
           &+AAA(I,4)*PPP(I)+AAA(I,5)*PPP(I+1)+AAA(I,6)*PPP(I+M1)
           &+AAA(I,7)*PPP(I+M2)
      CV         DO 90 I=1,N
s             DO 90 LP=1,LPMAX
      *VOCL LOOP,NOVREC(QQQ)
v             DO 90 L=LDX(LP)+1,LDX(LP+1)
v               I=IDX(L)
v        90 QQQ(I)=DDD(I)*(QQQ(I)-AAA(I,3)*QQQ(I-1)-AAA(I,2)*QQQ(I-M1)
           &-AAA(I,1)*QQQ(I-M2))
      CV         DO 100 I=N,1,-1
s             DO 100 LP=LPMAX,1,-1
      *VOCL LOOP,NOVREC(QQQ)
v             DO 100 L=LDX(LP)+1,LDX(LP+1)
v               I=IDX(L)
v       100 QQQ(I)=QQQ(I)-DDD(I)*(AAA(I,5)*QQQ(I+1)+AAA(I,6)*QQQ(I+M1)
           &+AAA(I,7)*QQQ(I+M2))
              C2 = 0.0D0
v             DO 110 I=1,N
v       110   C2=C2+QQQ(I)*R0(I)
      C       CAUTION !!
              ALPHA = C1 / C2
              C3 = 0.0D0
              X1 = 0.0D0
              X2 = 0.0D0
v             DO 120 I=1,N
v              HHH(I) = EEE(I) - ALPHA * QQQ(I)
      CV  120  CONTINUE
      CV         DO 130 I=1,N
v       130    WWW(I) = EEE(I) + HHH(I)
v       120  CONTINUE
v             DO 140 I=1,N
v       140 QQQ(I)=AAA(I,1)*WWW(I-M2)+AAA(I,2)*WWW(I-M1)+AAA(I,3)*WWW(I-1)
           &          +AAA(I,4)*WWW(I)+AAA(I,5)*WWW(I+1)+AAA(I,6)*WWW(I+M1)
           &          +AAA(I,7)*WWW(I+M2)
      CV         DO 150 I=1,N
s             DO 150 LP=1,LPMAX
      *VOCL LOOP,NOVREC(QQQ)
v             DO 150 L=LDX(LP)+1,LDX(LP+1)
v               I=IDX(L)
v       150 QQQ(I)=DDD(I)*(QQQ(I)-AAA(I,3)*QQQ(I-1)-AAA(I,2)*QQQ(I-M1)
```

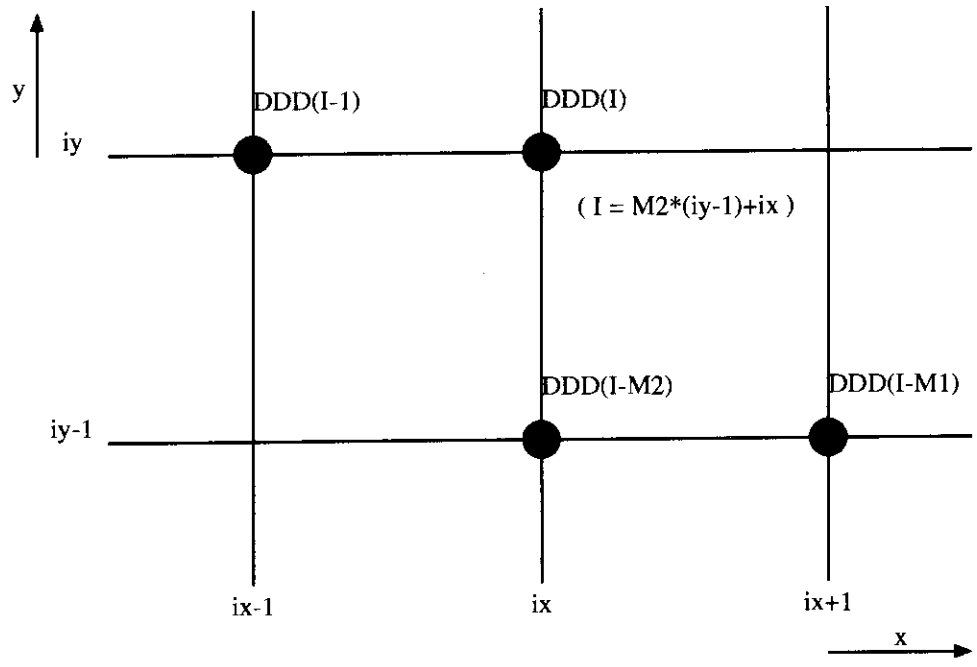
Fig. 3.2 Subroutine ILUCGS (vectorized, 3/4)

```
     &-AAA(I,1)*QQQ(I-M2))
CV        DO 160 I=N,1,-1
S        DO 160 LP=LPMAX,1,-1
*VOCL LOOP,NOVREC(QQQ)
v        DO 160 L=LDX(LP)+1,LDX(LP+1)
v          I=IDX(L)
v   160 QQQ(I)=QQQ(I)-DDD(I)*(AAA(I,5)*QQQ(I+1)+AAA(I,6)*QQQ(I+M1)
     &+AAA(I,7)*QQQ(I+M2))
v        DO 170 I=1,N
v         Y = XXX(I)
v         RRR(I)=RRR(I)-ALPHA*QQQ(I)
v         XXX(I)=XXX(I)+ALPHA*WWW(I)
v         C3 = C3 + RRR(I)*RO(I)
v         X1 = X1 + Y*Y
v   170   X2 = X2 + (XXX(I) - Y)**2
         IF (X1 .NE. 0.0) THEN
          RES = DSQRT(X2 / X1)
          IF (RES .LE. EPS) THEN
           ITR = K
           IER = 0
           EPS = RES
           IF (TH .NE. 1.0D0) S = TH
           RETURN
          ENDIF
         ENDIF
         BETA = C3 / C1
         C1 = C3
v        DO 180 I=1,N
v       EEE(I)=RRR(I)+BETA*HHH(I)
v       PPP(I)=EEE(I)+BETA*(HHH(I)+BETA*PPP(I))
v   180 CONTINUE
C
  200 CONTINUE
      IER = 1
      WRITE(*,*) '(SUBR. ILUCGS) NO CONVERGENCE. '
      EPS = RES
      IF (TH .NE. 1.0D0) S = TH
      RETURN
      END
```

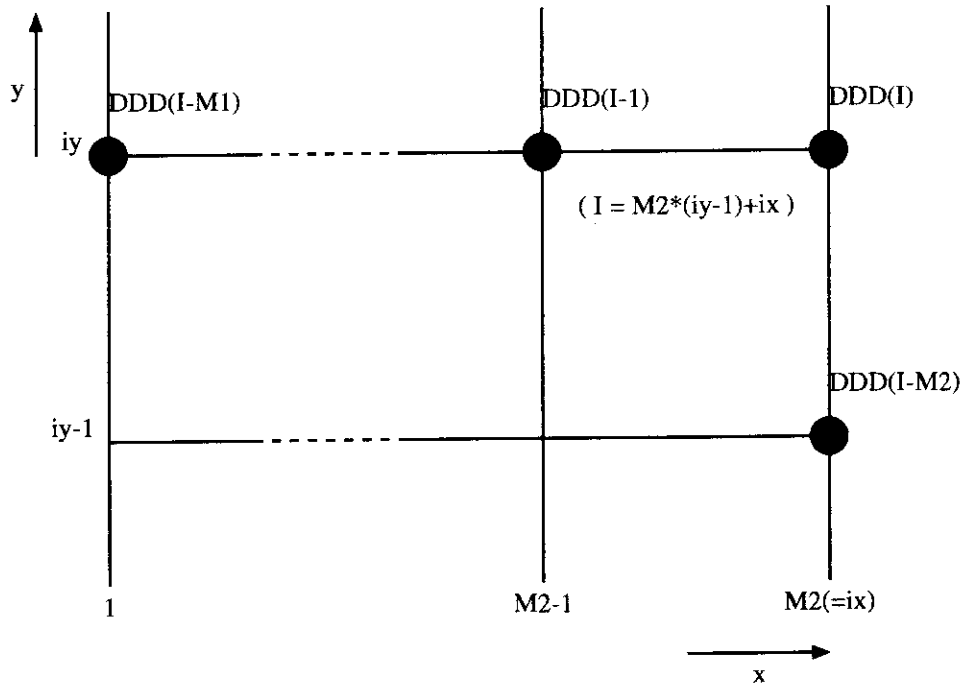
Fig. 3.2 Subroutine ILUCGS (vectorized, 4/4)

```
      SUBROUTINE MKLIST
      IMPLICIT REAL*8(A-H,O-Z)
      PARAMETER(NIG=67,NJG=67,NKG=67)
      COMMON /NEW/R(0:NJG),RR(0:NJG),DR(-2:NJG),DRR(-2:NJG),RM(0:NJG)
     &       ,DT,RE,PPE,PR,ALFA,CS,RLUX,PERR
     &       ,DSI,DZ,/GRID/IG,JG,KG,KAPPER
C     DIMENSION LDX(1:LPMAX+1), IDX(1:N)
C     (WHERE LPMAX=KG+JG-1,N=KG*JG)
      COMMON /HYPER/ LPMAX,LDX(128),IDX(4096)
C
      LPMAX=KG+JG-1
C
      II=0
s     DO 1000 LP=1,LPMAX
s        LDX(LP)=II
s        DO 1100 K=1,KG
v           DO 1200 J=1,JG
v              IF ( K+J.EQ.LP+1 ) THEN
v                 II=II+1
v                 IDX(II)=K+KG*(J-1)
v              ENDIF
v 1200      CONTINUE
s 1100    CONTINUE
s 1000 CONTINUE
C
      LDX(LPMAX+1)=KG*JG
C
      RETURN
      END
```

Fig. 3.3 New subroutine MKLIST

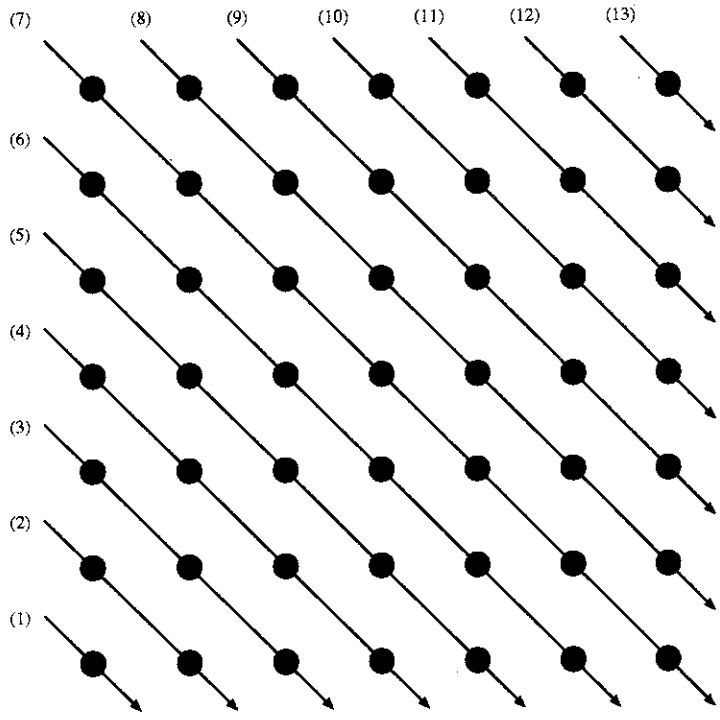


(a) ix ≠ M2 の場合



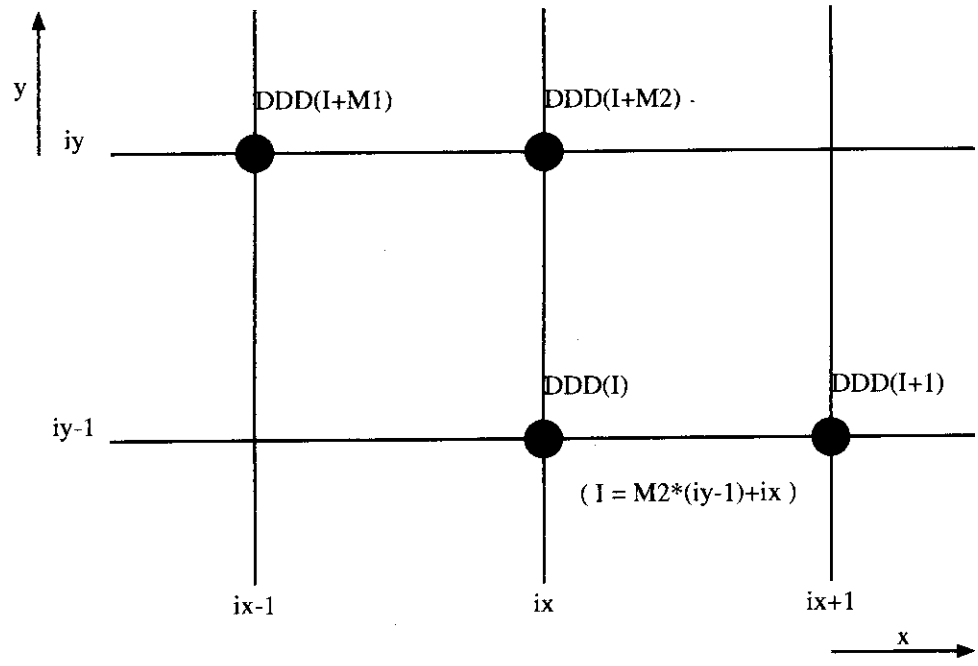
(b) ix = M2 の場合

Fig. 3.4 Relation between array elements in two dimensions

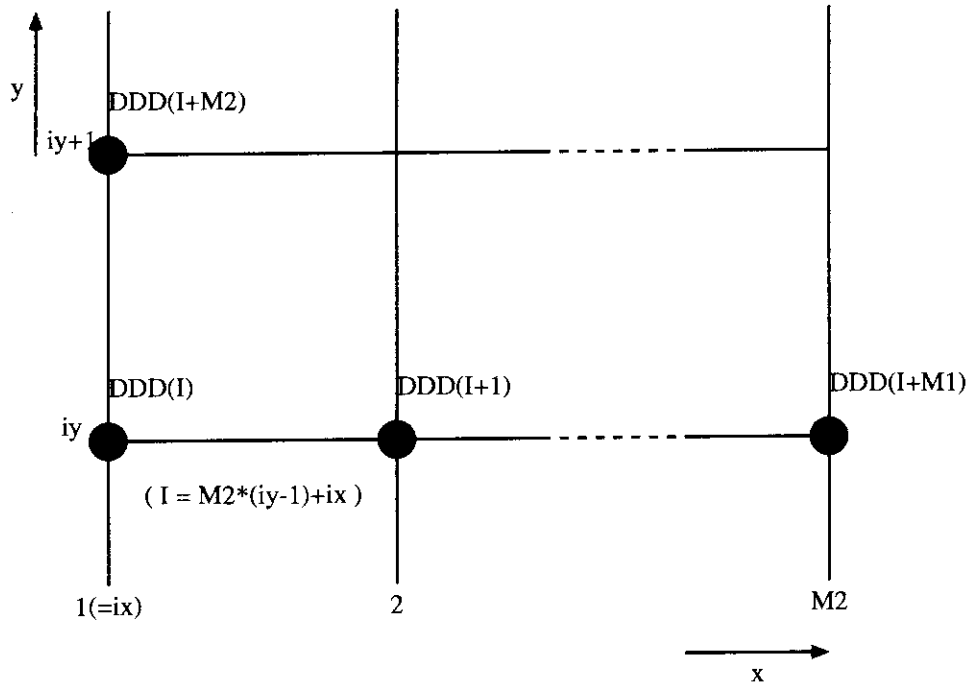



```
      DO 15 LP= 線分(1), 線分(2), ...
      DO 15 I=  (線分 LP 上の点)
      SS=S*A(I,4)- ...
   15 DDD(I) = 1.0D0 / SS
```

Fig. 3.5 Calculating order by Hyper Plane method



(a) ix ≠ 1 の場合



(b) ix = 1 の場合

Fig. 3.6 Relation between array elements in two dimensions (for backward solution)

```
!xocl index partition ipj=(spe,index=-3:NJG,part=band)
!xocl index partition ipi=(spe,index=0:NIG-4,part=cyclic)
C
!xocl index partition ipj11=ipj(overlap=(1,1))
!xocl index partition ipj01=ipj(overlap=(0,1))
!xocl index partition ipj10=ipj(overlap=(1,0))
!xocl index partition ipj21=ipj(overlap=(2,1))
C
```

Fig. 3.7 Definition of partition specification names (contents of the include file 'IP')

```
      DO 77 J=1,JG
      DO 77 K=1,KG
      P(0,J,K)=P(IG,J,K)
   77 P(IG+1,J,K)=P(1,J,K)
      DO 777 I=1,IG
      DO 777 K=1,KG
      P(I,JG+1,K)=P(I,JG,K)
  777 P(I,0,K)=P(I,1,K)
      DO 7777 I=1,IG
      DO 7777 J=1,JG
      P(I,J,KG+1)=P(I,J,KG)
 7777 P(I,J,0)=P(I,J,1)
```

Fig. 3.8 Example of border calculation (subroutine **SMATJG**)

```
!xocl spread do /ipj
      DO 77 J=1,JG
      DO 77 K=1,KG
      P(0,J,K)=P(IG,J,K)
   77 P(IG+1,J,K)=P(1,J,K)
!xocl end spread
CP       DO 777 I=1,IG
CP       DO 777 K=1,KG
CP       P(I,JG+1,K)=P(I,JG,K)
CP  777 P(I,0,K)=P(I,1,K)
!xocl spread do /ipj
      DO 777 J=1,JG
         IF ( J.EQ.1 ) THEN
!xocl spread move
            DO 7771 K=1,KG
            DO 7771 I=1,IG
               P_G(I,J-1,K)=P(I,J,K)
 7771       CONTINUE
!xocl end spread (mw5)
!xocl movewait (mw5)
         ENDIF
         IF ( J.EQ.JG ) THEN
!xocl spread move
            DO 7772 K=1,KG
            DO 7772 I=1,IG
               P_G(I,J+1,K)=P(I,J,K)
 7772       CONTINUE
!xocl end spread (mw6)
!xocl movewait (mw6)
         ENDIF
 777  CONTINUE
!xocl end spread
CP       DO 7777 I=1,IG
!xocl spread do /ipj
      DO 7777 J=1,JG
      DO 7777 I=1,IG
      P(I,J,KG+1)=P(I,J,KG)
 7777 P(I,J,0)=P(I,J,1)
!xocl end spread
```

Fig. 3.9 Example of parallelization for border calculation (subroutine SMATJG)

```
      DO 9 K=1,KG
      ATAI(K)=0.0D0
   9  CONTINUE
      J=JG-1
C
      DO 900 K=1,KG
      DOOP=0.0D0
      DO 919 I=1,IG
      DOOP=DOOP+1.0D0
      ATAI(K)=ATAI(K)+U(I,J,K ,2)
  919 CONTINUE
      ATAI(K)=ATAI(K)/DOOP
      WRITE(6,'(''k ATAI'',i3,e15.6)')K,ATAI(K)
  900 CONTINUE
```

Fig. 3.10 Example of BROADCAST (before parallelization, subroutine TAIRYU)

```
      LOGICAL FLAG
C
      FLAG = .FALSE.
!xocl spread do /ipj
      DO 101 J=JG-1,JG-1
         FLAG = .TRUE.
C
      DO 9 K=1,KG
      ATAI(K)=0.0D0
   9  CONTINUE
CP      J=JG-1
C
      DO 900 K=1,KG
      DOOP=0.0D0
      DO 919 I=1,IG
      DOOP=DOOP+1.0D0
      ATAI(K)=ATAI(K)+U(I,J,K ,2)
  919 CONTINUE
      ATAI(K)=ATAI(K)/DOOP
      WRITE(6,'(''k ATAI'',i3,e15.6)')K,ATAI(K)
  900 CONTINUE
C
 101  CONTINUE
!xocl end spread
!xocl broadcast (ATAI) (FLAG)
```

Fig. 3.11 Example of BROADCAST (subroutine TAIRYU)

```
        :
                          ! I 軸方向を分割した作業配列の宣言
      DIMENSION WK1_G(0:NIG-4,NJG-3,NKG-3),
     $          WK1  (0:NIG-4,NJG-3,NKG-3),
     $          WK2_G(0:NIG-4,NJG-3,NKG-3),
     $          WK2  (0:NIG-4,NJG-3,NKG-3)
      EQUIVALENCE (WK1_G,WK1),(WK2_G,WK2)
!xocl global WK1_G          ,WK2_G
!xocl local  WK1  (/ipi,:,:),WK2  (/ipi,:,:)
        :
C
                          ! J 軸方向を分割した配列から I 軸方向を分割し
                          ! た配列にデータを転送する
!xocl spread move /ipj
      DO 102 J=1,JG
         DO 101 K=1,KG
            DO 103 I=0,IG-1
               WK1_G(I,J,K)=CQ (I,J,K)
               WK2_G(I,J,K)=CQ2(I,J,K)
 103        CONTINUE
 101     CONTINUE
 102  CONTINUE
!xocl end spread (mw1)
!xocl movewait (mw1)
        :
                          ! ILUCGS 計算
!xocl spread do /ipi
      DO 915 MU=0,IG-1
C
      CALL GATRIJ(MU)
C
      DO 215 I=M,N+M2
  215  XXX(I)=0.0D0
C*********************REAL PART CALC
C      WRITE(6,*) 'ILUCGS. REAL part start'
      DO 2200 J=1,JG
      DO 2200 K=1,KG
      JX=KG*(J-1)
      II=K+JX
C      WRITE(6,'('' MU J K CQ'',3I5,E15.6)')MU,J,K,CQ(MU,J,K)
CP 2200 BBB(II)=CQ(MU,J,K)
 2200 BBB(II)=WK1(MU,J,K)
C      WRITE(6,*) ' bbb=cq end'
      ITR=30000
C      EPS=1.0D-40
      EPS=1.0D-6
      S=0.950D0
      CALL ILUCGS(N,L,M,M1,M2,EPS,ITR,IER,S)
C      WRITE(6,*) 'EXAMPLE OF ILUCGS. REAL',NU, ITR, EPS, S
C      WRITE(6,*) 'ILUCGS. REAL part end'
C
      DO 3300 J=1,JG
      DO 3300 K=1,KG
      JX=KG*(J-1)
      II=K+JX
CP 3300 CP(MU,J,K)=XXX(II)
 3300 WK1(MU,J,K)=XXX(II)
```

Fig. 3.12 Contents of parallelized subroutine SPERMG (1/2)

```
C
CP      IF(MU.EQ.IG/2) THEN
      IF(MU.EQ.0 .OR. MU.EQ.IG/2) THEN
      DO 11 J=1,JG
      DO 11 K=1,KG
CP   11 CP2(MU,J,K)=0.0D0
   11 WK2(MU,J,K)=0.0D0
      GOTO 915
      ENDIF
      DO 315 I=M,N+M2
  315  XXX(I)=0.0D0
C      WRITE(6,*) 'ILUCGS. imag part start'
C**********************IMAGINARY PART CALC
      DO 2220 J=1,JG
      DO 2220 K=1,KG
      JX=KG*(J-1)
      II=K+JX
CP 2220  BBB(II)=CQ2(MU,J,K)
 2220  BBB(II)=WK2(MU,J,K)
      ITR=30000
C      EPS=1.0D-40
      EPS=1.0D-6
      S=0.950D0
      CALL ILUCGS(N,L,M,M1,M2,EPS,ITR,IER,S)
C      WRITE(6,*) 'EXAMPLE OF ILUCGS. IMAG',NU, ITR, EPS, S
C      WRITE(6,*) 'ILUCGS. imag part end'
C
      DO 3330 J=1,JG
      DO 3330 K=1,KG
      JX=KG*(J-1)
      II=K+JX
CP 3330 CP2(MU,J,K)=XXX(II)
 3330 WK2(MU,J,K)=XXX(II)
C
  915 CONTINUE
!xocl end spread
C
                          ! I 軸方向を分割した配列から J 軸方向を分割し
                          ! た配列にデータを転送する
!xocl spread move /ipi
      DO 203 I=0,IG-1
         DO 201 K=1,KG
            DO 202 J=1,JG
               CP_G (I,J,K)=WK1(I,J,K)
               CP2_G(I,J,K)=WK2(I,J,K)
 202        CONTINUE
 201     CONTINUE
 203  CONTINUE
!xocl end spread (mw2)
!xocl movewait (mw2)
       :
```

Fig. 3.12 Contents of parallelized subroutine **SPERMG** (2/2)

```
MAIN
     CALL SEEDJJ
          READ P,U1,U2,U3
     CALL PREFFT
     CALL PREFFT
     CALL AMATJ
     CALL JWARIN
     CALL JWARCI
     CALL CDR
     P <-
     CALL TAIRYU
     CALL AVEAJ
     CALL RUBCG
          U1,U2,U3 <-
*         (U1,U2,U3 の袖転送)
  1  CONTINUE
----
     CALL R3F2JR
          <- U1(1,1),U2(1,0)
          CALL SPEW4S
               UT1 <-
          <- U1(0,1),U2(1,1),U3(0,1)
          CALL SPEV4S
               UT2 <-
          <- U2(1,0),U3(1,1)
          CALL SPEU4S
               UT3 <-
     CALL SMATJG
          UT1,UT2,UT3 <-
*         (UT2 の袖転送)
          <- UT2(1,0)
          CALL SPERMG
               FAP <-
          FAP <-
*         (FAP の袖転送)
          U1 <-
          U2 <- FAP(0,1)
          U3 <-
          P <-
     CALL RUBCG
          U1,U2,U3 <-
*         (U1,U2,U3,U5 の袖転送)
     CALL R3F2TT
          U4 <- U2(1,0),U3(1,1),U5(1,1)
          CALL SPET4S
               U5 <-
     CALL RUBCT
          U5 <-
----
```

Fig. 3.13 Relation between definition and reference of arrays divided with overlap

```
      IF( ... )THEN
      CALL AVEAJTT
      CALL GRAFV
          U2,U3 <-
*          (U2,U3 の袖転送)
      ENDIF
      IF( ... )THEN
      CALL SEED
      ENDIF
      IF( ... )THEN
      CALL JSAT23
          DUHEI,DVHEI,DPHEI <-
*          (DUHEI,DVHEI の袖転送)
          <- DUHEI(1,1),DVHEI(1,1)
          CALL POME
              UT1,UT2,UT3 <- U1(0,1),U3(0,1)
*          (P,UT1,UT3 の袖転送)
          <- U1(1,1),U2(1,0),U3(1,1),P(0,1),UT1(1,0),UT3(1,0)
      CALL JSAT23C
          EUHEI,EVHEI,EWHEI <-
          U2,U3 <-
*          (U2,U3,EUHEI,EVHEI,EWHEI,DPHEI の袖転送)
          <- EUHEI(1,1),EVHEI(2,1),EWHEI(1,1),U1(1,1),U2(2,1),U3(1,1),
                 P(1,1),DPHEI(1,1)
      ENDIF
      IF( ... )THEN
      CALL JWAR
      CALL JWARC
      CALL SEED
      GOTO 2
      ENDIF
      GOTO 1
    2 STOP
```

Fig. 3.13 Relation between definition and reference of arrays divided with overlap

```
 (サブルーチン SPERMG の一部)
        :
!xocl spread do /ipj
      DO 777 J=1,JG
      JJ=J
      CALL FFTS1(JJ)
      DO 78 K=1,KG
      DO 78 I=1,IG
   78 FAP(I,J,K)=UZ1(K,I)
  777 CONTINUE
!xocl end spread
        :

 (サブルーチン FFTS1 の一部)
        :
      L1=N1
      L3=N3
C-------------------------------------------------------------
C        INVERSE FOURIER TRANSFORM IN X-XZDM2RECTION
C
       DO 1000 K=0,N11
        DO 1000 I=0,N33
C
      UZ1(K,I)=CP(I,JJ,K+1)
      UZ2(K,I)=CP2(I,JJ,K+1)
C
 1000 CONTINUE
C
C
      CALL FFTY(L1,N3,UZ1,UZ2,XZDM1,XZDM2,NFAX3,IFAX3,TRIG3,1,N33,
     &            NN1)
        :
```

Fig. 3.14 Example of wrong parallelization

```
(サブルーチン SPERMG の一部)
      :
!xocl spread do /ipj
      DO 777 J=1,JG
      JJ=J
CD<<
CD      CALL FFTS1(JJ)
      L1=N1
      L3=N3
C-------------------------------------------------------------
C        INVERSE FOURIER TRANSFORM IN X-XZDM2RECTION
C
       DO 1000 K=0,N11
        DO 1000 I=0,N33
C
      UZ1(K+1,I+1)=CP(I,JJ,K+1)
      UZ2(K+1,I+1)=CP2(I,JJ,K+1)
C
 1000 CONTINUE
C
C
      CALL FFTY(L1,N3,UZ1,UZ2,XZDM1,XZDM2,NFAX3,IFAX3,TRIG3,1,N33,
     &            NN1)
C-------------------------------------------------------------
C        INVERSE FOURIER TRANSFORM IN X-XZDM2RECTION
CD>>
      DO 78 K=1,KG
      DO 78 I=1,IG
   78 FAP(I,J,K)=UZ1(K,I)
  777 CONTINUE
!xocl end spread
      :
```

Fig. 3.15 Example of legal parallelization (improved upon Fig. 3.14)

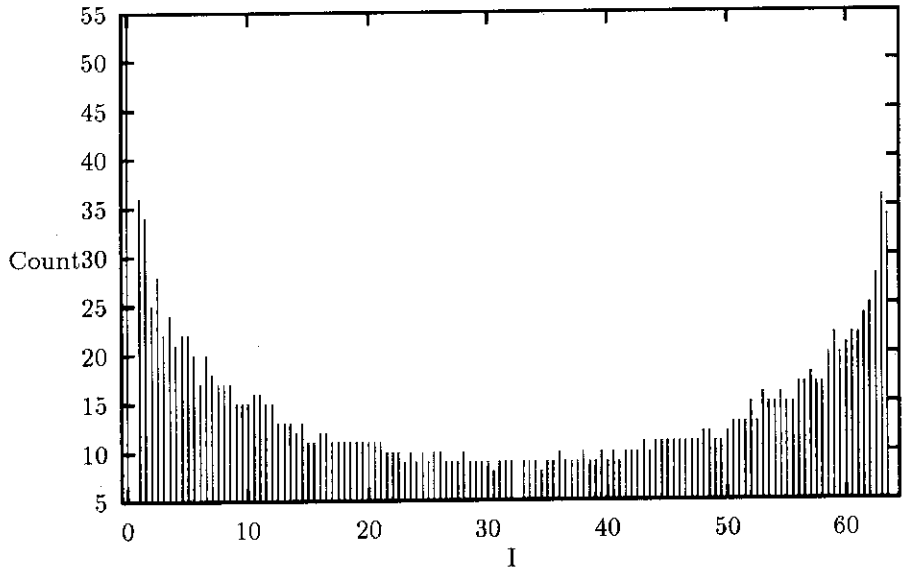


Fig. 3.16 Relation between I axis and repetition of ILUCGS loop

4. DGR コードのベクトル並列化

本章は，富士通(株)製ベクトル型計算機 VP2600 上でベクトル化済の DGR コードを，富士通(株)製分散メモリ型ベクトル並列計算機 VPP500/42 （以下 VPP500）上で並列化を行った結果の報告である．

今回の作業では，今まで利用されてきた本コードのオリジナル版とベクトル化版においてバグを発見した．そのため，富士通製大型汎用計算機 M780 上に存在している本コードの VPP500 への移植と並列化の他に，オリジナル版のバグ修正とベクトル化のやり直し作業も行うこととなった．以降では，移植に伴うコードの変更内容，バグの内容とその修正方法，やり直したベクトル化の内容，並列化の内容について述べる．

4.1 コード概要

DGR は，ダイヤモンド型結晶における照射損傷シミュレーションのための分子動力学コードであり [2]，PART1，PART2，PART3 から構成されている．それぞれがいくつかのルーチンから成る 1 つのコードであり，実行はそれぞれ個別に行われる．

PART1 では，結晶とその初期条件の設定，原子間の力やポテンシャル関数のテーブル化，および相互作用する隣接原子を検索するテーブルの設定等が行われる．

PART2 では，PART1 で設定されたデータを読み込み，結晶を構成する各原子に対する運動方程式を解き，種々のエネルギーの計算が行われる．中心力の他に，共有結合における変角力を模擬するための非中心力相互作用を扱っていることが本コードの特徴とされる．非中心力を扱うことにおいて中心力のみを扱うものと異なることは，1 個の原子は 4 本の結合手を持ち，4 個の原子と結合し，更にその結合手間の角度に依存した力を受けているので，いずれの 2 個も独立には扱えず，各原子対間に作用・反作用の法則が適用できないことである．

PART3 は，PART2 での計算結果の作図を行う．

・PART2 の処理

DGR コードにおいて今回ベクトル並列化を行った部分は，PART2 である．ここで，PART2 の処理について簡単に説明する．まず，PART2 のオリジナル版のフローを Fig. 4.1 に示す．

(1) サブルーチン GR2D08

全体の制御を行う PART2 でのメインルーチンであり，物理時間ステップ（GR2D3 ～ GR2D60）が存在する．物理時間ステップ幅は，計算過程でその変更が可能になっていて，最大 7 パターン（PART1 から引き継ぐものが 1 つ，PART2 の入力データでの指定が 6 つまで）を指定することが可能である．また，前物理時間ステップでの全エネルギーと現物理時間ステップでの全エネルギーの誤差と，入力データで指定した誤差の許容値を比較し，誤差が許容値を越える場合は物理時間ステップ幅を前のパターンに戻し，再度計算をさせるようになっている．

(2) サブルーチン GR2DI0

PART2 に対するテキスト入力データを読み込む．強制的に結合させる原子，誤差の許容値，各制御フラグを読み込む．

(3) サブルーチン GR2D10

PART1 で作成されたデータをファイルから読み込む．結晶とその初期条件，相互作用する隣接原子を検索するテーブル等を読み込む．ここで，簡単な結晶構造の例を Fig. 4.2 に示す．結晶の例は 2x2x2 の面心立方格子であり，黒丸は原子，黒丸の近傍に添付している小さい数字は原子番号，大きい数字は結晶における境界番号である．この結晶がダイヤモンド結晶の基本となるものである．各原子（6 枚の面や 12 本の辺，8 個の頂点に位置しないもの，いわゆる境界に位置しないもの）は，4 本の結合手を持って 4 個の原子と結合し，結合手間の角度を持つ．

(4) サブルーチン GR2D20

リスタート実行の場合に，前回の実行での出力データを読み込む．

(5) サブルーチン GR2D30

ある原子とその近傍の原子との間の相互作用を計算する前に，周辺の原子を検索する必要がある．その場合すべての原子を検索してその中から周辺の原子を決めていたのでは効率が悪い．そのため，DGR コードでは箱検索という処理を行っている．これは，検索主原子が格納されている箱を中心に，その周辺の箱を検索し（最大 27 個：自分自身の箱も含む），それらの箱の中の原子のみを検索し，それらの原子を周辺の原子として採用するという処理である．各箱には最大 6 個まで原子が格納されるようになっていて，箱には箱番号が付けられる．本ルーチンでは，各箱の中へ結晶内の各原子を格納する処理を行っている．各原子の位置によって格納される箱が異なる（各箱にはある範囲に存在する原子が格納される）

(6) サブルーチン GR2D40

GR2D41 で計算した各原子に作用する力に表面力の補正を行い，各原子の移動パラメータを求める．

(7) サブルーチン GR2D41

DGR コードでは，中心力相互作用と非中心力相互作用を扱っている．

中心力相互作用についての部分では，相互作用しているいずれの原子間にも作用・反作用が成り立つので，ある原子に作用する力を計算した時に相手原子の受ける力として負号を付けて保存しておき，その原子についての力を計算する時にそれらの値を使用すればよい．この場合は半分の原子を扱うだけでよい．

非中心力相互作用についての部分では，作用・反作用が成り立たないので，全原子について最近接の 4 原子を見つけ，それらとの相互作用を計算しなければならない．計算するものとしては，結合手間の 6 個の角度，4 本の結合手と 6 個の角度による位置エネルギー，結合手の伸縮による力，各原子に働く変角力がある．

本ルーチンでは，ある検索主原子を中心に箱検索を行って周辺の原子を検索し，被検索原子に対し中心力についての計算を行っている．更に被検索原子の中から最近接の 4 つ（結合手が 4 つある場合）の原子を決め，非中心力についての計算を行っている．4 つの原子を決める条件には，距離と各原子に設けられた計数がある．計数とは，ある原子にとって最近接の 4 個が決まったとしても，逆に選ばれた原子は選んだ原子を最近接の 4 個に選ぶとは限らない場合があるため，選ぶか選ばないかを判定するための値である．尚，PART2 のテキスト入力データにより，予め検索主原子と強制的に結合させる原子を，最大 50 個の原子に対しそれぞれ 1 つずつ指定することもできる．

(8) サブルーチン GR2D50

原子の新しい座標と古い座標とを比較し運動エネルギーを計算する．他にポテンシャルエネルギー，表面につなぎとめるスプリングエネルギー，表面から散逸するエネルギー，全エネルギーを計算する．全エネルギーについては，前時間ステップでの全エネルギーと比較して誤差も計算する．

(9) サブルーチン GR2D60

計算結果のファイルへの書き出しを行う．書き出された計算結果は，PART3 での入力データ，またはリスタート実行時の入力データになる．

4.2 M780 から VPP500 への移植

まず，M780 上に存在している DGR コードの VPP500 への移植を行った．M780 と VPP500 という計算機自体が異なることから，それらに搭載されているソフトウェアも異なる．相違点を Table 4.1 に示す．

移植作業は Table 4.1 の相違点を考慮しながら，必要であればソースの変更を行い，最終的に M780 上での実行結果と VPP500 上での実行結果を一致させることが目的となる．また，従来よりも更に大きいサイズの結晶を扱えるように，今まで 4000 個までしか扱えなかった（結晶サイズは 14x14x14 まで）原子数の上限をパラメータ化し，4000 個以上扱えるようにソースを変更した．以下では，移植によるソースの変更と扱える原子数のパラメータ化によるソースの変更について述べる．

4.2.1 移植によるソースの変更

PART1，PART2 については M780 上からコピーしただけで，問題なくコンパイル & 実行が可能であった．移植によってソースを変更する必要があった部分は PART3 のみである．PART3 は メインルーチン GR3D00 とサブルーチン XYPOUT の 2 つのルーチンから成る．

PART3 は PART2 の実行結果を作図する部分であるが，作図には図形処理ライブラリ CALCOMP [3] を利用している．M780 上では，単精度実数型と倍精度実数型のビットフォーマットを比較した場合，先頭から 4 バイトが同じフォーマットであるため，M780 版の PART3 ではライブラリへの実引数が倍精度実数型で渡されても問題は無かった．

しかし，VPP500 上ではビットフォーマットが異なるため，ライブラリへの実引数は単精度

実数型で渡さなければならない．そこで，新たに単精度実数型変数を用意し，その変数に倍精度実数を代入してから，代入された変数をライブラリへ渡すように変更した．Fig. 4.3 に実引数の変更の例を示す．

更に，PART3 にはある IF 文において倍精度実数どうしの EQUAL の判定を行っている条件式があった．これは，M780 と VPP500 でデータ形式が異なるため，M780 上で真であっても VPP500 では真でなくなる可能性がある．今後このような判定方法では，計算機システムの違いや計算誤差によって異なる結果になる可能性があるため，許容値を設けた判定方法に変更した．実際の変更を Fig. 4.4 に示す．ここで，許容値を 1.D-11 にしたのは，IF 文の判定に用いる倍精度実数型変数について，M780 上での実際の値と VPP500 上での実際の値を確認したところ，12 桁目以降が異なっていたことによる．

4.2.2 原子数上限のパラメータ化によるソースの変更

移植作業の追加作業として，DGR コードで 4000 個以上の原子を扱えるようにソースの変更を行った．従来は，原子の番号と座標を格納しておく配列等がコード中において陽に 4000 と指定され宣言されていたが，扱える原子数の上限をパラメータ文で指定できるようにし，各配列の宣言はパラメータ文で定義された定数名を利用して宣言するように変更した．

・PART1 について

PART1 では，大きいサイズのコモンブロック中のコモン変数と各ルーチンで跨って使用する変数や配列が，メモリ共有されて使用されている．従って，各配列のサイズを定数名を用いて宣言するように変更するのと同時に，コモンブロック中のコモン変数も定数名を用いて宣言するように変更した．先ず，本作業で追加したパラメータ文（原子数 6000 個の場合）を示す．

```
PARAMETER(NPAR=6000)
PARAMETER(NWPAR=NPAR*3)
```

シミュレーションの対象とする結晶の原子数が変更になった場合，この原子数の効率的な指定を可能にするため，このパラメータ文をインクルードファイル INCPAR に格納し，各ルーチンからはこのファイルのインクルードを行って利用するようにした．

更に変更を行う前の，コモンブロック中のコモン変数と各変数・配列との EQUIVALENCE 文について，コモンブロック GEOM 中のコモン変数 GEOM を Fig. 4.5 に，コモンブロック DYNA 中のコモン変数 DYNA を Fig. 4.6 に示す．各ルーチンにおいて実際には，そのルーチンで使用する変数・配列だけを EQUIVALENCE 宣言しているが，ここでは説明のため全変数・配列について示してある．

Fig. 4.5, Fig. 4.6 において，定数名を利用して宣言しなければならない配列は，コモン変数 GEOM とメモリ共有されている配列では TB と IBND，コモン変数 DYNA とメモリ共有されている配列では X と VX である．これらを各ルーチン毎に次のように変更した（ただし，ルーチンによっては以下のすべての配列が宣言されているとは限らない）．

```
TB(4000,3), IBND(4000), X(4000,3), VX(4000,3)
                ↓
```

```
TB(NPAR,3), IBND(NPAR), X(NPAR,3), VX(NPAR,3)
```

この変更に伴い，コモン変数 GEOM と DYNA の宣言を定数名を用いて宣言するように変更し，更に EQUIVALENCE 文においても定数名を用いた宣言へと変更した．変更結果を，コモン変数 GEOM を Fig. 4.7 に，コモン変数 DYNA を Fig. 4.8 に示す．

PART1 での宣言文の変更は以上であるが，実行文中において陽に原子数 4000 を用いた DO 文が存在した．その DO 文についても定数名を用いるように変更した．変更した部分を Fig. 4.9 に示す．

• PART2 について

PART2 では，検索箱の管理も行っているが，それは原子数の増減（結晶サイズの大小）に伴い増減するものである．M780 版ではこの箱の数はコード中で陽に 4096 個として使用されていたが，それもパラメータ文によって指定できるようにした．パラメータ化に伴うソースコードの変更の仕方は PART1 と同様であるため，ここでは，変更を行ったソースコードについてのみ示す．PART2 に対し追加したパラメータ文を Fig. 4.10 に，定数名を利用した宣言への変更を行った配列を Fig. 4.11 に示す．更に，コモンブロック GEOM2 中のコモン変数 GEOM2 と各変数・配列の EQUIVALENCE 文の変更前と変更後を Fig. 4.12 に，コモンブロック DYNA 中のコモン変数 DYNA については Fig. 4.13 に，コモンブロック WORK 中のコモン変数 WORK については Fig. 4.14 に示す．尚，追加したパラメータ文は，インクルードファイル INCPAR に格納した．このファイル名は PART1 と同一であるが，実際にコードを利用する場合に，PART1 と PART2 を別ディレクトリで管理すれば問題ない．

また，PART2 においても PART1 と同様，実行文中において陽に原子数 4000 や箱の数 4096 を用いた DO 文や IF 文が存在した．その DO 文や IF 文についても定数名を用いるように変更した．変更した部分を Fig. 4.15 と Fig. 4.16 に示す．

• PART3 について

PART3 においても変更の仕方は PART1，PART2 と同様である．PART3 では，コモンブロックは使用されておらず，またそれとの各配列や変数の EQUIVALENCE 文も無い．変更の対象になったものは，ローカル配列の宣言のみである．ここでは，PART3 に対し追加したパラメータ文と定数名を利用した宣言への変更を行った配列をまとめて，Fig. 4.17 に示す．尚，同様に追加したパラメータ文は，インクルードファイル INCPAR に格納した．

4.3 オリジナル版コードのバグの修正

VPP500 向けのコード変換が終了し，各入力データを用いて VPP500 上での動作確認を行っている際，オリジナル版コードにおいてバグを発見した．このバグは，コード変換前の M780 版でも存在するものであるが，結晶サイズ 16x16x16 の入力データを用いた VPP500 での実行において，前物理時間ステップの全エネルギーと現物理時間ステップの全エネルギーの誤差が許容値を越えて異常終了し，その原因を探った結果判明したものである．

4.3.1 デバッグ

全エネルギーを求めているサブルーチン GR2D50 の一部を Fig. 4.18 に示す．ここでは，27 行目と 31 行目において各エネルギーの和によって全エネルギーが求められており，その内異常に大きいエネルギー値が格納されていた変数は，ENGPO であった．これはポテンシャルエネルギーが格納されている変数である．ポテンシャルエネルギーが求められているのは，サブルーチン GR2D41 の中心力についての計算部分の最後と，非中心力についての計算の部分の最後である（Fig. 4.19）．これらの値を確認したところ，既に前半の中心力についての計算部分において異常に大きいポテンシャルエネルギー値が求められていた．そこで Fig. 4.19 の 20 行目の配列 EE の値，その添字 I5 の値を調査したところ，添字 I5 には配列 EE のサイズを越える値が格納されており，そのため配列 EE が異常値になっていたことが判明した．次にその添字の値を決めるための 8 行目の E11，E1(3)，WRK，E2(3) の値を調査した．その結果，WRK に異常に大きい値（ただし結晶サイズは越えない）が格納されることが判った．この WRK は検索主原子と被検索原子との距離が格納される変数であるため，箱検索処理においてかなり距離が離れている原子が検索されていることが予想された．ここで箱検索処理での 4 通りの周りの箱の探し方について説明する．

(1) 検索主原子が結晶構造の内部に位置する場合，検索主原子が格納されている箱を含み，且つその箱を中心とした 3x3x3 個積み重なった箱が検索される．この場合，全部で箱は 27 個存在するが，PART1 で作成された箱検索のためのテーブル中に用意されている 27 個のポインタを用いて検索される．検索主原子が格納されている中心の箱番号に 27 個の内のある 1 個のポインタを足すと 27 個の中からそれに該当する箱が検索される．

(2) 検索主原子が結晶構造の 6 枚の面に位置する場合，検索主原子が格納されている箱を含み，且つその箱を中心とし結晶構造の中心に向けて 3x3x2 個積み重なった箱が検索される．この場合，全部で箱は 18 個存在するが，(1) と同様に箱検索のためのテーブル中に用意されている 18 個のポインタを用いて検索される．ただし，6 枚の面のうちどの面に位置するかによって使用する 18 個のポインタのセットも異なる．

(3) 検索主原子が結晶構造の 12 本の辺に位置する場合，同様に結晶構造の中心に向けて 3x2x2 個積み重なった箱が検索される．この場合，全部で箱は 12 個存在するが，同様に箱検索のためのテーブル中に用意されている 12 個のポインタを用いて検索される．ただし，この場合も 12 本の辺のうちどの辺に位置するかによって使用する 12 個のポインタのセットも異なる．

(4) 検索主原子が結晶構造の 8 個の頂点に位置する場合，同様に結晶構造の中心に向けて 2x2x2 個積み重なった箱が検索される．この場合も同様に箱検索のためのテーブル中に用意されている 8 個のポインタを用いて検索される．ただし，同様にどの頂点に位置するかによって使用する 8 個のポインタのセットも異なる．

以上の箱の探し方に注目し WRK に異常に大きい値が格納される場合を机上で調査したところ，遠く離れた積み重なっていない箱を検索していることが判った．これにより，原子同士の距離が

異常に大きな値になっていた．更に調査したところ，このようになってしまうのは，検索主原子は結晶構造の内部に存在しているが，その原子が格納されている箱はある面（境界部分）に位置しているような場合，つまり，原子の位置情報と箱の位置情報が異なっている場合であることが判った．箱検索処理は，上の４通りの箱の探し方から判るようにように原子の位置情報によって箱検索のパターンが決まるが，ポインタは箱を検索するために利用されているため，このようなことが発生してしまう．従って，箱の位置と検索主原子の位置が一致する場合のみ正しい検索を行い，一致しない場合は異常処理になってしまうことが判った．

4.3.2 バグに対する修正

修正する方法としては，箱に対して結晶構造の位置情報を与えてやることが最適であると考えられる．しかし，箱検索処理に関連する PART1 での箱検索のためのテーブル作成の仕方や，箱番号の決め方，位置情報の決め方等のアルゴリズムがかなり過去に作成された経緯があり，且つかなり複雑であったため詳細までは不明であった．従ってユーザとの相談の結果，今回は根本からの修正は行わないことにした．

そこで，代替修正として， PART2 の箱検索処理においてかなり遠くの離れた原子を検索した場合は，ある一定の距離を設けておき，その距離以上離れている場合はその原子を計算対象から外す方針にした．この距離に採用したものは，共有結合伸張力のテーブル作成における結合距離の最大値であり， PART1 のテキスト入力データにより設定されるものである．これはユーザの指導により決定した．箱検索処理自体が，相互作用ポテンシャルの作用範囲外にある原子まで検索 & チェックするのは効率が悪いために行われている処理であり，また厳密にはその距離の範囲内に存在する周りの原子だけを考慮すれば良いとのことであった．以下，この修正のために行ったコードの変更内容について述べる．

本修正で利用することになった共有結合伸張力テーブル作成における結合距離は， PART1 のテキスト入力データで設定される値である．この結合距離値を，原子を計算対象から外すかどうかの判定に利用する（実際に実行文として利用する）のは， PART2 の箱検索処理の部分（サブルーチン GR2D41）であるため， PART1 で設定される距離値を PART2 でも利用できるようにしなければならない．本コードでは， PART1 で作成されたファイルを PART2 では入力データとして利用している．そこで， PART1 で作成されるファイルに距離値も追加し，更に PART2 でそれを入力して実行文に利用できるようにした．

・ PART1 についての変更

変更を行ったルーチンは， GR1DIO, GR1D60 の 2 つのルーチンである． GR1DIO は PART1 のテキスト入力データを読み込むルーチンであり， GR1D60 は PART2 の入力データになるファイルに PART1 の結果を書き出すルーチンである．結合距離値のファイルへの書き出しは， GR1DIO 中において，入力データから読み込んだ直後に行うように変更した．しかし， GR1D60 では，先頭において REWIND 文が実行されているため，書き込んだ距離値が他のデータに上書きされてしまう．そこで， GR1D60 ルーチンの先頭の REWIND 文をコメント化し，代わりに GR1DIO の距離値の書き出しを行う直前に REWIND 文を実行させるようにした． GR1DIO の変更した部分

を Fig. 4.20 に示す. 以上の変更によって, PART1 から PART2 へ渡されるファイルの先頭レコードには, 結合距離値が存在することになった.

・PART2 についての変更

変更を行ったルーチンは, GR2D10, GR2D41 の 2 つのルーチンである. GR2D10 は PART1 で作成されたデータをファイルから読み込み, 必要であれば (制御フラグによる) 機番 19 へ読み込んだデータを書き出すルーチンである. GR2D41 は 箱検索処理があり, 結合距離値を実際に利用するルーチンである.

PART1 から渡されるファイルの先頭レコードに距離値が存在しているため, GR2D10 では先頭の READ 文の直前に, 距離値を読み込むための READ 文を追加し, 更に機番 19 へ距離値を書き出すための WRITE 文も追加した. これに伴い, 距離値を格納しておく配列を新たに宣言し, コモンブロック DYNA 中の コモン変数 DYNA とメモリ共有させることにした. GR2D10 の変更部分を Fig. 4.21に示す.

また, GR2D41 では 箱検索処理中において, 検索主原子と被検索原子との距離の計算直後に IF 文を追加し, PART1 から渡された距離値との比較を行い, その距離より遠い場合はその原子についての計算は行わず, 次の原子の検索をさせるように変更した. 距離値を格納する配列宣言については, コモンブロック DYNA 中のコモン変数 DYNA に存在する GR2D10 ルーチンでメモリ共有させた配列名と同一のものを利用することにした. GR2D41 の変更部分を Fig. 4.22 に示す. Fig. 4.22 では, 検索主原子に強制的に結合させる原子が指定されていた場合で, その原子が距離値より遠く離れていた場合には, フラグ ICOMB と NAL の値を強制的に結合させる原子が指定されていない場合の通常の値に定義し直すようにした.

・PART3 についての変更

PART1 と PART2 の変更により, PART2 で書き出されたファイルは, PART3 で作図する場合の入力データになるが, その先頭には結合距離値が存在することになった. しかし, PART3 ではその距離値は必要無いため, ダミー処理としてその距離値を READ する処理をメインルーチン GR3D00 に追加した. これを, Fig. 4.23 に示す.

4.3.3 実行結果の確認

本コードには, オリジナル版 (スカラ版) についてバグが存在していたため, 移植の妥当性とバグ修正の妥当性については, バグ修正後のソースコードを使用し, 且つ各入力データを使用した VPP500 での実行結果を再評価するという方法で行った. その結果妥当であるとの評価を得て, M780 から VPP500 への移植とバグの修正を終えることにした.

4.4 ベクトル化

今回の作業では, 従来利用されていたベクトル化版 (以下旧ベクトル化版と呼ぶ) においてもバグを発見した [4]. 従って, 本ベクトル化作業は, そのバグが存在するルーチンに対するベクトル化のやり直しに相当するものである. 本章では, バグの内容とやり直したベクトル化の内容について述べる. 尚, ベクトル化はバグ修正後のオリジナル版を基に行った.

4.4.1 旧ベクトル化版のバグについて

旧ベクトル化版においてバグは 4 つ存在した．以下，4 つのバグの内容について述べる．

(1) サブルーチン GR2D3 について（1 つ）

旧ベクトル化版のサブルーチン GR2D3 の一部を Fig. 4.24 に示す．ここでは，DO 4100 ループに対し，回帰演算が発生しないことをコンパイラに指示する最適化制御行 *vocl loop,novrec[5] が指定されていることが問題である．このループ内においては，定義・参照されている配列 IBOX について回帰演算が発生する．従って，この最適化制御行は指定してはいけない．

(2) サブルーチン GR2D41 について（3 つ）

本ルーチンはステップ数が多いため，ここでは図に示さず，バグの内容についてのみ述べる．

・箱検索処理におけるフラグ（ICOMB, NAL）の設定

これらのフラグには，検索主原子に対し強制的に結合させる原子が指定されていた場合と指定されていない場合の，箱検索処理における最近接の 4 つの原子の決め方の違いを制御する働きがある．旧ベクトル化版のバグにより処理が異常になるのは，検索主原子に対し強制的に結合させる原子が指定されている場合である．

オリジナル版では，強制的に結合させる原子が指定されている場合，箱検索処理の直前で (ICOMB,NAL)=(1,2) のペアの初期値が設定され，検索主原子に対する 4 つの最近接原子の 1 つに強制的に結合させる原子が採用された後は，そのペア値は，(ICOMB,NAL)=(0,2) に変えられる．強制的に結合させる原子は入力データによって指定されるため，箱検索処理を行って検索する必要はないが，その原子を最近接の 1 つとして定義する実行文は箱検索処理中に存在する．その処理中では，強制的に結合させる原子を検索主原子の結合手の第 1 本目 (NAL-ICOMB=1) に定義させ，その後，残りの 3 つの最近接原子を第 2 本目 (NAL-ICOMB=2) 以降に定義させるようになっている．従って，ペア値の変更が必要になっている．

しかし，旧ベクトル化版では，(ICOMB,NAL)=(1,2) のペアの初期値は設定されるが，(ICOMB,NAL)=(0,2) への変更がされていないために問題である．これにより，常に結合手の第 1 本目から定義されるため，強制的に結合させる原子が 4 つの最近接原子の 1 つとして採用されない場合があり，これが異常な結果を出力する原因になる．

・配列 AC の定義・参照

この配列には，相互作用による各原子の持つ力が求められる．中心力についての計算においては，作用・反作用が成り立つので，被検索原子が受ける力としてこの力に負号を付けて保存しておく．その保存した値は，その原子についての力を計算する場合に力の初期値として使用すればよい．

しかし，旧ベクトル化版においては，ある原子について作用する力を計算する場合に以前に保存されていた力を使用していないために問題である．

・配列 ICOV の定義・参照

この配列には，各原子に設けられる計数が格納される．この計数については，本ルーチンがCALL される直前に（親ルーチン GR2D40 において）すべての原子に対し初期値が設定される．

オリジナル版では，ある検索主原子に対し最近接の 4 つの原子が決まった場合に，その 4 つの原子についての計数値を更新し，他の検索主原子が最近接の 4 つの原子を決める際にその値を参照するようになっている．

しかし，旧ベクトル化版においては，計数の初期値のみで各検索主原子の最近接の 4 つの原子を決めているために問題である．これにより，オリジナル版とは選ばれる 4 つの原子が異なってしまう．

4.4.2 ベクトル化コーディング

ベクトル化作業は，バグの存在したサブルーチン GR2D3 と GR2D41 についてのみ行った．GR2D41 については，実行結果をオリジナル版と一致させることを目的としたバグの修正だけでは，旧ベクトル化版での DO ループの構造が問題となりベクトル化されない．そのため，ベクトル化のやり直しを行わなければならなかったが，旧ベクトル化版でのベクトル化方法を利用できるところは利用するようにし，作業の効率化を図った．

(1) サブルーチン GR2D3

このルーチンについては，問題となる最適化制御行 `*vocl loop,novrec` の指定を削除した．

(2) サブルーチン GR2D41

オリジナル版のこのルーチンは，処理内容により大きく 2 つの部分に別れる．

1 つ目は，各原子に対して 4 つの最近接原子を検索するための処理である．その中では，箱検索により周りの箱に格納されている原子をすべて探し出し，それらの原子について中心力について計算を行う．更に探し出された原子の中から，距離と計数により最近接の 4 つの原子を決める．

2 つ目は，各原子とそれらの最近接の 4 つの原子を利用し，結合手間の 6 個の角度，4 本の結合手と 6 個の角度によるエネルギー，結合手による伸縮による力，各原子に働く変角力を計算する．ここは非中心力についての計算を行う部分である．

今回ベクトル化を行ったのは 1 つ目の部分である．この部分は旧ベクトル化版のループの構造が原因で実行結果を一致させることを目的としたバグの修正だけではベクトル化されなかった部分である．2 つ目の部分は，各原子について最近接の 4 つの原子のリストを作成しておけば，各々の原子についての計算は独立に行うことが可能であるので，ベクトル化が可能である．旧ベクトル化版においてもこの方法でベクトル化されている．従って，本作業では 2 つ目の部分についてのベクトル化は行っていないため，ここでは GR2D41 の先頭から 1 つ目の部分までについて説明する．

・オリジナル版の処理内容

オリジナル版の GR2D41 の 1 つ目の部分を Fig. 4.25に示す．この中で，1000 CONTINUE ～ GO TO 1000 が周りの箱のループ（最大 27 回），2000 CONTINUE ～ GO TO 2000 が周りの 1 つの箱当たりに格納されている原子の数のループ（最大 6 回）であり，この 2 重ループは箱検索処理に相当する．

37 行目～ 41 行目では検索主原子と被検索原子との距離を計算する．中心力についての計算を行う部分は 52 行目～ 62 行目であり，直前のいくつかの IF 文は相手の原子を計算の対象にするかどうかの選別をする働きをする．63 行目の GO TO 2400 以降の処理は，探し出された原子の中から，距離と計数により最近接の 4 つの原子を決める処理になる．63 行目の IF 文は計数の判定であり，最近接の 4 つの候補になるかどうかを判定する．64 行目～ 91 行目は，計数により候補になった原子の中から，検索主原子に最も距離が近いものから順に各配列に整列される．その各配列に整列される数は，検索主原子が持つ結合手の数になる．

ここで，もし検索主原子に対し強制的に結合させる原子が指定されている（PART2 の入力データで最大 50 個の原子に対し 1 個ずつ指定される）場合は，この箱検索処理には従わずに，専用の GO TO 文（16 行目，20 行目，84 行目）によって処理され，最近接の 4 つの内の 1 つとして採用される．従って，この場合，後の箱検索処理によって各配列に整列される配列の数は，（結合手の数－ 1）個になる．この数は，強制的に結合させる原子が指定されている場合は 14 行目～ 15 行目と 18 行目～ 19 行目，22 行目で設定されるフラグで制御され，指定されていない場合は 6 行目～ 7 行目で設定されるフラグによって制御される．

97 行目～ 100 行目は，最近接の 4 つの原子について計数を更新する部分である．

尚，オリジナル版ではサブルーチン GR2D40 の原子ループ（結晶中の全原子の個数を回転数とする）中からサブルーチン GR2D41 を CALL しているため，この部分の処理が検索主原子 1 個につき 1 回実行される．

・新ベクトル化版の変更

新ベクトル化版では，旧ベクトル化版と同様に親ルーチンの GR2D40 から原子ループをサブルーチン GR2D41 に引き込み，その引き込んだ DO ループを利用してベクトル化を行った．しかし，単に引き込んだだけではベクトル化されないため，GR2D41 中の機能毎に処理を分割し，それらを引き込んだ原子ループで囲んでベクトル化させるようにした．

従って，新ベクトル化版では 1 つ目の部分を，箱検索だけを行う処理，検索主原子と被検索原子の距離を求める処理，中心力についての計算，4 つの最近接原子を決める処理に分割し，それぞれ原子ループで囲んでベクトル化させるようにした．各々の分割された処理において求められる各原子毎の値は，分割された処理間で値の引き渡しが行えるように，各原子毎のリストを作成して，その中に格納した．

オリジナル版では，各物理時間ステップ毎に，GR2D41 は原子の数分実行されていたのに対し，新ベクトル化版では，原子ループを引き込んだことにより，GR2D41 は 1 回だけ実行されることになった．ここでは，サブルーチン GR2D41 の分割した処理毎に，新ベクトル化版の内容について述べる．新ベクトル化版 GR2D41 中の処理構造を Fig. 4.26 に示す．

a. DO 9000 ループについて

この部分を Fig. 4.27 に示す．ここは，各配列に 0 の初期値設定を行っている部分である．他の分割した処理に値を引き渡せるように，原子毎に設定を行っている（リスト配列に格納している）ため，これらの配列を新たに宣言している．

b. DO 9001 ループについて

この部分を Fig. 4.28 に示す．各原子への結合手の定義は，1 回の実行で最初に本ルーチンが実行される時だけ行えばよい．それは最外の IF 文で制御している．NVAL_V には各原子の結合手数が格納され，I2_V には箱検索処理に用いるポインタの 1 番目のアドレス，IC_V には最後のアドレスが格納される．ここでも，リスト配列を用意してそれに各原子毎の値を格納している．

c. DO 9002 ループ，DO 5000 ループについて

この部分を Fig. 4.29 に示す．DO 9002 ループでは各原子が属する箱番号が J1_V に格納される．また，DO 5000 ループと DO 5001 ループにより，(ICOMB,NAL) のペアの初期値がそれぞれ，ICOMB_V，NAL_V に格納される．ISEARCH_NUM は原子毎の被検索原子の数が格納される．ここでも，リスト配列を用意してそれに原子毎の値を格納している．

d. DO 7001 ループ，DO 7000 ループについて

この部分を Fig. 4.30 に示す．ここは，検索主原子に対し強制的に結合させる原子が指定されていた場合のみ実行する．強制的に結合させる原子は，各原子の被検索原子のリスト配列 I4_LIST の先頭に格納される．その時の (ICOMB,NAL) のペアは (1,2) に更新される．また，DO 7000 ループを内側ループにし，ベクトル実行時のベクトル長を大きくした．

e. DO 8000 ループについて

この部分を Fig. 4.31 に示す．DO 8000 は原子ループで，DO 8001 が周りの箱数のループ，DO 8002 が箱中の原子数のループである．ここでは検索主原子に対する被検索原子を，リスト配列 I4_LIST に格納し，検索主原子毎の被検索原子の数 ISEARCH_NUM をカウントしている．この処理により，リスト配列 I4_LIST には検索主原子毎の被検索原子が格納されることになる．また，c. ～ e. が前に属していた箱から原子がずれた場合のみの処理になる．

f. DO 600 ループについて

この部分を Fig. 4.32 に示す．ここでは，リスト配列 I4_LIST を利用して，各原子と被検索原子との距離を求め，計算対象となる距離範囲に存在する原子のみを，新しいリスト配列 I4_LIST2 に格納する．また，それぞれの距離についても，各原子と被検索原子のリスト配列の添字と一致するように，リスト配列 WRK に格納する．計算対象となる新しい被検索原子の数を格納するリスト配列は，I_MAIN である．

DO 601 ループの直前では，強制的に結合させる原子が指定されていた場合で，距離が計算対象から外れる場合，フラグ ICOMB の値を通常の場合に戻している．ここで求められたリスト配列 I4_LIST2 と WRK，XW，YW，ZW は，中心力についての計算部分と 4 つの最近接原子を決め

る部分において同じものが利用される.

g. DO 800 ループについて

この部分を Fig. 4.33 に示す. 各原子と相手原子（被検索原子）の相互作用計算で求められた原子の持つ力は，相手原子が受ける力として負号を付けて，X，Y，Z 成分毎にそれぞれ ZZ1，ZZ2，ZZ3 に保存される. その後，その相手原子の持つ力を計算する場合に保存されていた値を初期値とする. 旧ベクトル化版ではこの初期値が考慮されていなかった.

ここでは，DO 801 ループの直前で *VOCL LOOP,NOVREC が指定されているが，この指定が無いとコンパイラが配列 ZZ1，ZZ2，ZZ3 について回帰計算であると判断しベクトル化されない. しかし，被検索原子のリスト配列 I4_LSLT2 には同じ原子が 2 度現われないことから，回帰計算は発生しない. *VOCL LOOP,NOVREC はコンパイラに回帰計算が発生しないことを指示するための最適化制御行である. これを指定することによって，DO 801 ループの DO 変数 I22 でベクトル化された.

h. DO 700 ループについて

この部分を Fig. 4.34 に示す. ここは，各検索主原子に対する 4 つの最近接原子を決め，それらを各検索主原子の最近接原子のリスト配列 NF に格納する部分である. DO 701 ループが 4 つの最近接原子を決める部分で，DO 703 ループが検索主原子と最近接原子の距離のリスト（リスト配列 RNF，R）を作成し，更に各原子の計数値を更新する部分である. ここで作成された各リストは非中心力についての計算で利用する.

DO 701 ループは，DO 702 ループとそれ以外の部分に別れる.

DO 702 ループは検索主原子毎のすべての被検索原子を対象とし，その中から計数が 4 以下で且つ最も距離が近い原子を一つ選ぶ. そしてそれ以外の部分において，被検索原子リストの一番先頭の原子との入れ換えを行い，4 つの最近接原子の 1 つ目として採用する. 次に 2 つ目の最近接原子を選ぶ場合は，被検索原子リストの先頭の原子を除いた原子を対象とし，その中から最も距離が近い原子を 1 つ選ぶ. そしてリストの先頭から 2 番目の原子との入れ換えを行い，最近接原子の 2 つ目として採用する. 以下同様に最大 4 つまでの最近接原子を決定する.

DO 702 ループは，変数 WRK_L について回帰計算であるが，DO 変数 I22 でベクトル化される. これは，コンパイラが該当する部分を最小値や最大値を求める処理として認識すると，回帰計算が発生しないように特別に命令を置き換えてくれるためである. ここでは，コンパイラに最小値を求める処理と認識させるようなコーディングを行うことにより，ベクトル化が可能になった.

4.5 計算結果の評価及びベクトル化の効果

ここでは，新ベクトル化版コードの計算結果とベクトル化の効果について述べる.

4.5.1 計算結果について

チューニング前のオリジナル版コード（バグ修正版）のベクトル実行結果と，チューニング後の新ベクトル化版コードのベクトル実行結果を比較した. 実行は，ダイヤモンド結晶を 4x4x4

～ 16x16x16 までの各サイズに設定した場合の各入力データと，それらに強制的に結合させる原子を指定した場合の各入力データについて行った．その結果，計算結果は一致していた．

4.5.2 ベクトル化の効果について

新ベクトル化版コードの性能を測るため，VPP500 上において，オリジナル版コード（バグ修正版）をベクトル実行した場合の実行時間と，新ベクトル化版コードをベクトル実行した場合の実行時間を比較した．実行時間は，プログラム終了直前の CPU 占有時間と開始直後の CPU 占有時間の差とし，終了直前と開始直後にそれぞれ CPU 占有時間を取得できるサービスルーチン clock を挿入して計測した．計測した結果と使用した入力データを Table 4.2 に示す．尚，各入力データは物理時間ステップ幅を 0.2 にし，物理時間 4.0 までシミュレーションを行う設定にした．

Table 4.2 より，VPP500 上での新ベクトル化版コードのベクトル実行では，オリジナル版コードのベクトル実行時間と比較すると，1.8 倍～ 3.3 倍の速度性能の向上を得ることができた．

4.6 並列化

本並列化は，データの分割は行わずに処理の分割のみを行う方法で作業を行った．処理の分割として，具体的には原子ループの PE（Processing Element）台数による分割と，物理時間ステップの並列化版実行処理ステップへの分割を行った．以降では本並列化の作業内容について述べる．尚，作業は新ベクトル化版コード（PART2 のみ）を基に行った．

4.6.1 動的挙動解析

並列化を行う前に，新ベクトル化版コードのベクトル実行での計算コストを把握するため，動的挙動解析を行った．この解析には VPP500 上で動作する動的挙動解析ツール SAMPLER [6] を使用した．この結果を Fig. 4.35 に示す．

4.6.2 並列化の方針

今回の並列化では，原子ループの分割と物理時間ステップの分割を行う手法をとった．原子ループを分割しただけでは，依存関係が存在するために並列化による高速化は不可能である．そこで，メインルーチンで制御していた従来の物理時間毎の処理ステップの，並列化版実行処理ステップへの分割も行い，高速化を図ることにした．新ベクトル化版は，物理時間ステップ毎に全原子の計算を行わせる構造であったが，並列化版では，新しい実行処理ステップにおいて，PE 毎に異なる物理時間について，且つ PE 毎に異なる原子群についての計算を行わせる構造になっている．以降では，原子ループを分割した場合の問題点（依存関係）と，物理時間ステップの分割，それらの分割による処理構造の問題点，更にデータを非分割にした理由について述べる．

4.6.2.1 原子ループの分割について

本コードで計算コストの高いルーチンは，箱検索処理部分と中心力についての計算部分，および 4 つの最近接原子を決める部分が存在する GR2D41 ルーチンである．新ベクトル化版のこの

ルーチン中では，この 3 つの処理部分を始め，各処理部分は原子ループに囲まれているものがほとんどであるので，処理分割としてこれらの DO ループの回転数を分割する方針とした．

VPP500 向けの並列化を行う場合には，コード中の並列処理を行わせたい部分の直前に並列化指示行 [7] を挿入する（4.6.3.1 参照）．ここで，中心力についての計算部分と 4 つの最近接原子を決める部分の直前に原子ループ（DO ループ）の分割を行わせる並列化指示行を指定した場合の実行イメージを Fig. 4.36 に示す．このイメージは，例として全原子数が 20 個の結晶について 4 台の PE を利用した場合における DO ループ分割の並列実行である．T1 〜 TN は物理時間ステップ，exe1 〜 exeN は実行処理ステップを表す．Fig. 4.36 の箱で囲まれた部分を以降では並列処理構造における処理単位と呼ぶ．

• Fig. 4.36 の並列実行を行った場合の中心力についての計算部分の問題

この部分においては各原子の力が格納されている配列 AC，ZZ1，ZZ2，ZZ3 で問題が生じる（Fig. 4.33 参照）．AC には相互作用によって原子が持つ力が求められるが，この時相手の原子が受ける力として負号をつけて ZZ1，ZZ2，ZZ3 に保存しておく．そして後にその相手原子についての力の計算を行う場合には，保存されていた ZZ1，ZZ2，ZZ3 を力の初期値として採用する．従って，Fig. 4.36 のような並列実行を行うと，検索主原子 ID が属する PE と被検索原子 ID が属する PE が異なる場合においては ZZ1，ZZ2，ZZ3 を初期値として採用しなくなってしまう．更に本コードの中心力についての計算部分では，例えば，1 度検索主原子 α と被検索原子 β との相互作用計算を行うと，後の計算で β が検索主原子になった場合，原子 α を検索したとしても計算の対象から外していることから，原子 1 番〜 n 番まで昇順に計算を行わなければならない仕様になっている．このことからも Fig. 4.36 のような実行は行えない．

• Fig. 4.36 の並列実行を行った場合の 4 つの最近接原子を決める部分の問題

この部分においては各原子の計数が格納されている配列 ICOV で問題が生じる（Fig. 4.34 参照）．この ICOV は検索主原子に対する 4 つの最近接原子が決まった場合に，それら 4 つの原子の計数値が更新され，以降で別の検索主原子に対する 4 つの最近接原子を決める場合にその更新された値を参照する．従って，中心力についての計算部分と同様に，検索主原子 ID が属する PE と被検索原子 ID が属する PE が異なる場合において，更新された ICOV の参照が不可能になってしまう．

以上より，正しい計算を行わせるため，原子ループの分割による並列実行は Fig. 4.37 のようにすることにした（本来であれば Fig. 4.38 のようにするべきだが，これでは問題があり Fig. 4.37 のようにした．この問題については 4.6.3.4 を参照）．ただし，この処理構造では，順番に計算をすることになり，シングル実行と変わらないため並列化による高速化は不可能である．尚，この 2 つの部分をこの処理構造にしたため，他の原子ループもこの処理構造で行うことになる．

4.6.2.2 物理時間ステップの分割について

原子ループの分割による高速化は不可能であったが，Fig. 4.39のような処理構造の可能性が考えられた．これは Fig. 4.37を基にすると，各物理時間ステップの計算を上に詰めたものに相当し，従来の各物理時間ステップをこの処理構造の各実行処理ステップへ分割したものになる．

しかし，本コードでは各物理時間ステップ間において，現物理時間ステップで求めた各原子の座標は次の物理時間ステップで利用されるという依存関係が存在するためこの処理構造で正しい計算が可能かどうか検討した．その結果，次の物理時間ステップで使用する各原子の新しい座標は，全原子についての計算終了後に求めなければならないという制約は無く，原子毎に求めることが可能であることが判った．従って，この処理構造の縦の列だけに注目すると計算は可能である．具対的には，(exe1,4PE)（処理構造中のある処理単位を表す時は，処理構造を(exe の数（行の数），PE の数（列の数））のマトリックスとみて以下このように示す）で求められた各原子の新しい座標は，直下の (exe2,4PE) で利用することになる．また，他の処理単位についても同様で，ある処理単位で求められた各原子の座標は直下の処理単位で利用することになる．

そこで，次に例えば (exe2,4PE) に注目する．各 PE 上で計算される原子は箱検索処理等を行う場合，隣の PE に属している原子を検索する場合がある．その時の被検索原子の座標は，検索主原子と同一の物理時間の座標が求められている必要がある．しかし，この構造だと (exe2,4PE) では物理時間 T2 の計算を行っているが，その隣の (exe2,3PE) ではまだ物理時間 T1 の計算を行っているため，(exe2,4PE) での検索主原子が箱検索により (exe2,3PE) での原子を検索した場合，前の物理時間の座標を保持している原子を検索する可能性があるため，この処理構造には問題がある．そこで，次の物理時間ステップの計算開始を Fig. 4.39 よりも実行処理ステップを 1 タイミング遅らせ，両隣の PE の計算終了後に各 PE の計算を行わせるように処理構造を変更した．この構造を Fig. 4.40 に示す．

Fig. 4.36 においては各物理時間ステップと実行処理ステップが同一であったのに対し，この処理構造は実行処理ステップ毎に，PE 毎に異なる物理時間について，且つ PE 毎に異なる原子群についての計算を行う構造になっている．また，Fig. 4.37 と Fig. 4.40 を比較すると，Fig. 4.37 では 4 回の実行処理ステップで 1 回結果を吐き出すのに対し，Fig. 4.40 では 2 回吐き出す．これより，この処理構造では最大で，使用する PE 台数 /2 倍（並列化効率 50 %）の性能向上が予想できる．この処理構造を今回の並列化処理構造として確立し，コーディングもこの処理構造向けに行うことにした．

4.6.2.3 並列処理構造の問題点と対策

今回の並列処理構造において，本来コードが持っている 2 つの機能が利用できないことが判った．また，機能とは別にこの処理構造では並列実行時に問題が発生する可能性があることが判った．ここでは，利用できない 2 つの機能についての問題とコーディング前に予想された並列実行時の問題，それぞれの対策を述べる．

(1) 利用できない 2 つの機能について

・全エネルギーの誤差が許容値を越えた場合に，時間ステップ幅を前のパターンに戻し，1 ステップ戻って計算を行う機能．

全エネルギーを求めることができるタイミングは，全原子についての計算が終了した後である．並列処理構造においては，この時点以前に既に次以降の時間ステップの計算が開始されているため，1ステップ戻して計算を行わせるとそれらの計算が無駄になってしまう．しかし，これまでの経験により，1ステップ戻して計算し直しても期待する結果は得られないことが多く，今回はこの機能は省くことにした．

・クエンチング機能

これは全原子によって求められる全運動エネルギーに注目し，シミュレーション中に全運動エネルギーが減り始めた時点で全原子の運動エネルギーを0にする機能である．この全エネルギーも，全原子についての計算が終了した後に求められる．また同様に，本並列処理構造では，既に次以降の時間ステップの計算が開始されているため，全原子の運動エネルギーを0にするタイミングがオリジナル版とは異なってしまい，実行結果も異なってしまう．そのため，この機能を利用する場合は，新ベクトル化版コードを利用することにした．よって，並列化版コードからはこの機能は省くことにした．

(2) 予想された並列実行時の問題

本並列処理構造の実行で問題になるのは，あるPE上の検索主原子が2PE分隣のPE上に属する原子を検索する場合である．この場合，物理時間が検索主原子と被検索原子で異なる可能性があるため問題である．これについては，比較的小さい結晶サイズに対して使用するPE台数が多い程発生する確率が高くなる．しかし，これを防ぐために結晶サイズに使用するPE台数を対応させたとしても，2PE分隣のPE上の原子を検索する可能性がまだ存在する．それは，2PE分隣のPE上の原子が計算結果隣のPE上に移動してくる場合である．

これら2つの問題については，実際のシミュレーションでは，各原子の移動計算はゆっくり移動するように設定され，また多くの場合は振動を扱うため，隣のPE上に移動してくる場合はほとんど無いとのことであった．しかし，その確率は0ではないので，もし，2PE分隣のPE上の原子を検索した場合には，実行を強制的にストップさせることにし，リスタート実行を行うことで対処する方針にした．リスタート実行では，PE台数を前の実行より少なく設定することが必要となる．

4.6.2.4 データの非分割について

本並列処理構造において，新ベクトル化版コードで使用しているデータの分割は行わないことにした．処理の分割は各原子ループの分割によって行うことにしたため，データの分割を行うとすれば，各原子に属する（原子のIDで管理される）データ類の分割が考えられる．

それは例えば，各原子の座標が格納されている配列，各原子の力が格納される配列，各原子の計数が格納される配列，各原子が格納される箱番号が格納されている配列等を挙げることができる．しかし，これらの配列を分割し各PE上にローカルなデータを持たせると，計算コストの高いGR2D41ルーチンの箱検索処理と中心力についての計算部分，4つの最近接原子を決める部分において，配列アクセスによるPE間のデータ転送が発生する．これは各検索主原子に対する被検索原子が隣のPE上に存在する場合があるからである．

VPP500 においてPE 間のデータ転送が発生すると実行時間が極端にかかってしまう恐れがあるが，本コードで上記のそれぞれの配列を各 PE に分割して持たせた場合，実行時間を増やさない方法としては，各配列の袖付き分割を行い，自 PE の計算を行う前に袖転送を行って，予め自 PE 上にデータを転送しておく方法がある．この袖というのは，自 PE 上の配列から隣の PE 上の同じ配列のどこまでの範囲をアクセスするかが判っている場合のその範囲のことを言う．この範囲は，コンパイルを行う前に陽にコード中に指定しなければならないが，本コード中の箱検索処理や中心力についての計算部分では，各 PE からその隣の PE 上の原子のどこまでをアクセスするかが不明である．そこで，袖付き分割をおこなわせるとしたら，隣の PE 上の原子すべてを袖として定義する必要がある．しかし，使用する PE 台数と結晶サイズによって，袖部分のメモリサイズがかなり大きくなってしまう場合があり，袖部分を格納するグローバルメモリのオーバーが予想され，実行不可能になる可能性がある．また，使用する PE 台数や結晶サイズを変更すると袖の範囲も予め計算してコード中に指定しなければならないという煩わしさもあるため，データの分割は行わないことにした．

4.6.3 並列化コーディング

並列化の大部分は，オリジナル版の各物理時間ステップを基にした構造から，並列化版の実行処理ステップを基にした構造（並列処理構造）への変更作業と言える．従って，この並列処理のため大きな変更を行ったのは，全体の処理を制御しているメインルーチン GR2D08 である．

4.6.3.1 並列化指示行

ここで，以降の説明のため，今回の並列化コーディングで利用した並列化指示行について説明する．並列化指示行とは，ソース中において !XOCL で始まる行のことを言い，コード中の並列処理を行わせる部分に対し指定する．行わせたい並列処理の内容により，利用する並列化指示行は異なる．

- `!XOCL PROCESSOR P(NPE)`

P はプロセッサグループ名，NPE はプロセッサグループが持つ形状を表す．プロセッサグループとは，複数個のプロセッサを配列状にまとめて順序付けたものである．例えば，NPE=4 とした場合には，P は大きさ 4 の 1 次元プロセッサグループの名前として宣言させる（NPE は並列実行時に使用する PE 台数）．並列化版コードにおいては，インクルードファイル INC_PE を用意しその中で使用する PE 台数を指定できるようにしている．

- `!XOCL INDEX PARTITION PP=(P,INDEX=1:NPE,PART=BAND)`

INDEX PARTITON 文は分割方法を定義する場合に利用し，ここでの PP は分割方法名である．括弧内の P は分割方法の対象となるプロセッサグループ名であり，INDEX はインデックス範囲（分割方法の対象となる範囲），PART にはどのように分割するかを指定する．BAND は均等分割を意味する．例えば，NPE=4 の場合には，P(1):1, P(2):2, P(3):3, P(4):4 という分割方法 PP が定義される．

・!XOCL SUBPROCESSOR SUBP(NPE)=P(1:NPE)

SUBPROCESSOR 文はサブルーチン内で指定する．ここでの SUBP は P の別名になる．

・!XOCL LOCAL X(:,:,:,/PP)

配列 X のローカル分割宣言である．例えば，X(10,10,10,4) という配列 X が宣言されているとすると，この宣言により，4 次元目が分割方法 PP により分割され，P(1):X(10,10,10,1)，P(2):X(10,10,10,2)，P(3):X(10,10,10,3)，P(4):X(10,10,10,4) のように各 PE にローカルデータとして配置される．このローカルデータはそれぞれの PE からのみ定義・引用が可能である．

・!XOCL GLOBAL X_G

配列 X のグローバル配列宣言である．ローカル分割されたデータはそれぞれの PE からしか定義・引用できないのに対し，グローバル宣言されたデータ（グローバルデータ）は，すべての PE からの定義・引用が可能である．

・!XOCL PARALLEL REGION

この行により，パラレル実行が開始される（各 PE で冗長実行）．この行の直前までは通常のシングル実行であり直後はパラレル実行になる．この行の直後のすべての PE の状態はシングル実行時と同じである．

・!XOCL END PARALLEL

!XOCL PARALLEL REGION に対して指定する行であり，パラレル実行からシングル実行になる．ただし，この直後のシングル実行ではパラレル実行時のどの PE の状態が引き継がれるかは予測不可能である．また，パラレル実行終了直前には，すべての PE において同期がとられる．尚，!XOCL PARALLEL REGION と !XOCL END PARALLEL に囲まれた部分をパラレルリージョンと呼ぶ．

・!XOCL SPREAD DO

これは，DO ループの分割を指定する場合に利用する．この場合は，DO ループの回転数はプロセッサグループに従って均等に分割される．プロセッサグループが P(4) の場合は，4 台の PE に対し回転数が均等に割り当てられる．尚，この行の直前ではすべての PE において同期がとられる．

・!XOCL END SPREAD

!XOCL SPREAD DO に対して指定する行であり，DO ループ分割の終了を意味する．この行の直前でも，すべての PE において同期がとられる．

・!XOCL END SPREAD SUM(A)

!XOCL SPREAD DO に対して指定する行であり，同様に DO ループ分割の終了を意味する．異

なるのは SUM(A) であるが，これは DO ループの分割の終了と同時に，各 PE で求められた変数 A または配列 A について総和を行わせる指定である．この後，総和された A の値は各 PE で冗長に持つようになる．尚，この場合もこの行の直前ですべての PE において同期がとられる．

- !XOCL SPREAD MOVE

ローカルデータとグローバルデータとの間の代入の開始を意味する．代入は各 PE が分担して行う．ローカルデータからグローバルデータへの代入，グローバルデータからローカルデータへの代入の両方が可能である．

- !XOCL END SPREAD (MW1)

!XOCL SPREAD MOVE に対して指定する行である．(MW1) はトランスファー ID である．尚，この行の直前では代入待ちやすべての PE において同期はとらない．

- !XOCL MOVEWAIT (MW1)

この行では，ローカルデータとグローバルデータとの間の代入終了を待つ．ここでの (MW1) もトランスファー ID であるが，これはどの代入を待つかの ID として利用される．この場合は !XOCL END SPREAD (MW1) についての代入待ちを行う．代入は各 PE が分担して行うため，この代入待ちが終了するのは PE 毎に同じタイミングとは限らない．

- !XOCL BROADCAST(DT_HOLD) (OUT_FLAG)

ここでの，DT_HOLD，OUT_FLAG は各 PE に冗長に存在するデータで，OUT_FLAG はこの行を実行するかしないかのフラグとなる．この行は，OUT_FLAG が真である PE 上の DT_HOLD の値をすべての PE に転送する働きをする．

4.6.3.2 並列処理構造の制御と必要なデータ転送

ここで，並列処理構造の制御の実現の仕方と，どのタイミングでどのデータを転送する必要があるかを説明する．

(1) 制御について

Fig. 4.40 において箱に囲まれた部分が実行するべき処理単位になる．VPP500 向け並列化を行うために用意されいる並列化指示行を単純に指定しただけでは，このような制御は不可能である．

この処理構造において，並列実行時に使用する PE 台数が偶数台の時は，実行処理ステップの奇数番目で偶数 PE 上での実行が必要であり，偶数番目で奇数 PE 上での実行が必要であることがわかる．そして，使用する PE 台数が奇数台の時は，実行処理ステップの奇数番目で奇数 PE 上での実行が必要であり，偶数番目で偶数 PE 上での実行が必要であることがわかる．そこで，この法則性を利用し，各実行処理ステップの先頭において，そのステップで実行させるべき PE に対し予め実行フラグを立てておくことにより，実行処理ステップの制御を実現することにした．

また，各実行処理ステップにおける各物理時間については，PE 毎に管理させている．これにより，各実行処理ステップにおいて PE 毎に異なる物理時間を持つことができ，これを PE 毎に計算を終了させる値としても利用している．ただし，物理時間更新の際には実行処理ステップ毎に同期を取る必要がある．

更に，PE 毎に異なる原子群を与えることについては，新たに追加したサブルーチン INIT_PARA において，予め各 PE に処理させる原子群を求めておくことにより実現することにした．以上より，コード中では次のような並列実行をさせるコーディングを多用することになった．

```
!XOCL SPREAD DO
      DO 100 I=1,NPE
      IF(FLAG) THEN
            •
            •
       (FLAG が真の PE における原子群で,
        且つ FLAG が真の PE における物理時間についての計算)
            •
            •
      ENDIF
  100 CONTINUE
!XOCL END SPREAD
```

※ DO 100 ループは PE 台数分回転する．この回転数は並列化指示行 !XOCL SPREAD DO により各 PE に均等に分割される．従って，3 行目から 10 行目の部分が各 PE で 1 回実行されることになる．その内，フラグが真の PE のみが 4 行目から 9 行目の部分を実行することになる．

(2) データ転送について

並列処理構造において転送が必要なデータは，各原子に設けられる計数が格納される配列 ICOV，中心力についての計算時に被検索原子の受ける力が保存される配列 ZZ1，ZZ2，ZZ3，各原子の座標が格納される配列 X，座標の移動パラメータが格納される配列 VX，箱の中にどの原子が格納されているかを管理する配列 IBOX，各エネルギーを格納する変数 ENGL，ENGSP，ENGPO，ENGKG である．これらのデータ転送は各実行処理ステップ間で必要になる．

A. 配列 ICOV について

この配列は，実行処理ステップ毎ではなく物理時間毎に初期値が与えられ，且つ 4 つの最近接原子について更新され，更にこの更新された値が他の原子についての 4 つの最近接原子を決める場合に参照される．従って，各処理単位で更新された値が Fig. 4.40 の並列処理構造上，左下の処理単位で必要になるため，更新された後に左隣の PE への転送が必要になる．初期値設定は，

各物理時間ステップの計算開始時において行う．ICOV の転送は，検索主原子は，ID の近い原子を検索するとは限らないため，すべての範囲について行う必要がある．

B. 配列 ZZ1, ZZ2, ZZ3 について

これらの配列には，被検索原子が後の計算で検索主原子になった場合の力の初期値が格納されている．これも実行処理ステップ毎ではなく，物理時間毎に必要になる値である．従って，A. と同様に各処理単位で保存された値が並列処理構造上，左下の処理単位で必要になるため，ZZ1, ZZ2, ZZ3 に値が保存された後に左隣の PE への転送が必要になる．これらの転送も，すべての ZZ1, ZZ2, ZZ3 について行う必要がある．

C. 変数 ENGL, ENGSP, ENGPO, ENGKG について

これらの変数についても同様で，各処理単位の原子群の計算によって求められたエネルギーの総和値が，処理構造上左下の処理単位で必要になる．従って，値が格納された後に左隣の PE への転送が必要になる．これらも実行処理ステップ毎にはなく，物理時間毎に必要になる値である．

D. 配列 X, VX について

これらの値は，物理時間毎に必要で，且つ新しい値が次の物理時間で必要になる．従って，これらの配列において更新された値は，処理構造上各処理単位の左下，下，右下の処理単位で必要になる．

先ず左下の処理単位についてであるが，左下の処理単位に属する原子群が箱検索により周りの原子を検索する場合，右隣の PE に属する原子も検索する可能性があるため，左隣の PE 上の配列 X, VX に対して，予め右隣で求めた原子群の座標値を転送しておく必要がある．

次に下の処理単位については，各座標は物理時間ステップ間で依存関係があるため，下の処理単位でも必要になるが，上の処理単位と同一の原子群の座標値を使用する（同一の PE）ので転送の必要はない．

最後に右下の処理単位についてであるが，右下の処理単位に属する原子群が箱検索により周りの原子を検索する場合，左隣の PE に属する原子も検索する可能性があるため，右隣の PE 上の配列 X, VX に対して，予め左隣で求めた原子群の座標値を転送しておく必要がある．右下の処理単位については，新しい物理時間用のデータ転送になる．

以上より，配列 X, VX については，下の処理単位への転送は必要ないが，両隣の PE への転送が必要になる．更に 1PE 上において全原子の座標をファイルへ書き出す必要があるため，左隣の PE に対しては，右側のすべての PE 上で求められた全原子群の座標を転送する必要がある．

E. 配列 IBOX について

この配列は，実行処理ステップ毎ではなく，物理時間毎に必要になる値である．この配列については並列処理構造上，各処理単位の左下，右下の処理単位で必要になる．シミュレーションでは，各原子の座標が変わるのと同時に原子の属する箱も変わる可能性があるので，原子の座標値

配列 X, VX と同様の理由で配列 IBOX の転送が必要である．ただし，1PE 上でのファイルへの書き出しは無いため，この理由によるすべての範囲についての転送は必要無いが，検索主原子が属する箱番号と周りの箱番号は近いとは限らないため，配列 ICOV, ZZ1, ZZ2, ZZ3 と同様にすべての範囲の転送が必要になる．

以上，A.～E. の配列は，各実行処理ステップにおいて，PE 毎に異なる物理時間を持たせる必要があるため，コーディング（各データの利用）には工夫が必要であった．実際にこれらの配列を利用するには，どのようなデータ宣言が必要であったかを，ここで，各原子の座標を格納する配列 X を例にして説明する．

まず，PE を 1 台のみ使用して実行する場合には，シングル実行と変わらないため，シングル実行時と同様に次のように宣言すれば良いことになる．

```
X(NPAR,3)
```

次に PE を 2 台使用して並列実行する場合であるが，その場合の並列処理構造は Fig. 4.41 のようになる．これは，PE を 2 台使用した並列実行であるが，実行順序はシングル実行時と変わらない．よって，同様に次のように宣言すれば良いことになる．

```
X(NPAR,3)
```

次に PE を 4 台使用して並列実行する場合であるが，その場合の並列処理構造は既説の Fig. 4.40になる．ここで各物理時間ステップについての計算が終了するタイミングに注目すると，物理時間 T1 については，(exe4,1PE) で，物理時間 T2 については，(exe6,1PE) で終了することになる．逆に，計算開始のタイミングに注目すると，物理時間 T3 については，(exe5,4PE) で開始される．以上より，実行処理ステップ exe4 が終了すると T1 で使用していたデータは必要なくなるので，その時使用していたデータ領域を実行処理ステップ exe5 で開始される T3 についての計算に使用できることがわかる．同様にある物理時間 Tm までを考慮すると，各物理時間ステップで使用しているデータ領域が，他の物理時間ステップの計算によるデータにより，異常に重複して定義・参照されないようにするには，最低，物理時間 2 ステップ分のデータ領域が必要になる．従って，この並列実行の場合は，次のように宣言すれば良いことになる．

```
X(NPAR,3,2)
```

以下，同様にして PE を 8 台使用した場合，16 台使用した場合の並列処理構造から考慮すると，使用する PE 台数を NPE とすると，最低，物理時間 NPE-NPE/2 ステップ分のデータ領域が必要になることが判った．以上より，汎用的には次のように宣言すれば良いことになる．

```
X(NPAR,3,NPE-NPE/2)
```

こうすることにより，物理時間ステップ数分の領域は確保する必要はない．

また，今回の並列化ではデータの分割は行わないことにしたため，配列 X(NPAR,3,NPE-NPE/2) は各 PE 上に同じものが配置されることになる．実際の配置の仕方としては，先ず次のような配列を用意する．

```
X(NPAR,3,NPE-NPE/2,NPE)
```

そして，第 4 次元目を PE 台数で分割し，結果的には分割ローカルデータとして配置するようにした．従って，4 台の PE を使用した場合には，次のような配置になる．

```
PE1 :   X(NPAR,3,NPE-NPE/2,1)
PE2 :   X(NPAR,3,NPE-NPE/2,2)
PE3 :   X(NPAR,3,NPE-NPE/2,3)
PE4 :   X(NPAR,3,NPE-NPE/2,4)
```

以上より，上記の A.～E. の各配列は最終的に Fig. 4.42 のように宣言することになった．EQUIVALENCE 文によって分割ローカルデータとグローバルデータがメモリ共有されることにより，自 PE 以外に存在するグローバルデータをアクセスする場合に特にアドレスについて考慮する必要はない．

4.6.3.3 コーディング

ここでは，並列化版の全体のフロー図と処理制御について述べ，次に，並列化で新たに用意したファイルや各ルーチンの実際のコーディングについて説明する．

(1) 全体の処理フローについて

新ベクトル化版の処理フローを Fig. 4.43 に，並列化版の処理フローを Fig. 4.44 に示す．これより，サブルーチン GR2D3 の実行順序が変わったこととサブルーチン GR2D60 が無くなったことがわかる．更にフロー図からは判断できないが，サブルーチン GR2D50 の一部をサブルーチン GR2D40 に移動してある．

・サブルーチン GR2D3 の実行順序について

GR2D3 では，結晶内の各原子を箱に格納する処理を行う．新ベクトル化版では，箱検索処理が行われる GR2D41 ルーチンの前に GR2D3 が実行されていた．並列処理構造において，箱検索処理では隣の PE に属する原子 ID を検索する可能性があるため，隣の PE に属する原子についても同時に箱に格納する処理をしておく必要がある．そうすると，NPE 上での箱格納の処理では，NPE-1 上での原子についての箱格納も同時にしなければならない．

例えば，全原子数が 20 個あり，4 台の PE を利用して並列化した場合でこの原子の数が各 PE に均等に配分されていたとすると (Fig. 4.40)，4PE 上での箱格納処理をする原子 ID の範囲は 1～10，3PE 上では 11～15，2PE 上では 16～20，1PE 上では処理は行わないことになり，各 PE の計算負荷が均等では無くなってしまう．VPP500 上で並列計算を行う場合には，できるだけ負荷を各 PE で均等になるようにするのが高速化のポイントになる．

これを改善するため，各物理時間ステップにおいて新しい座標が求まった後に，その新しい座標に対して箱格納を行うようにした．こうすることにより，PE 毎に各 PE に属する原子 ID だけの箱格納を行えばよいことになり，それを次の物理時間ステップで利用すればよいことになる．ただし，最初の物理時間の計算で用いる箱データについては，予め求めておく必要があるため，実行処理ステップの前に 1 回だけ GR2D3 を実行するようにした．

この機能は省くことになったので，Fig. 4.50 にはこの機能は存在しない．並列化版の実行において，この機能を利用するような状態になった場合は，Fig. 4.50 の 13 行目の GO TO 文によって実行を強制終了するようにしている．強制終了の仕方は次のようになる．

Fig. 4.50 の 13 行目の GO TO 文によって 68 行目で配列 ILEGAL2_L(I_MYPE) にフラグ 3 が設定される（I_MYPE は PE-ID）．この設定が終ると，GR2D50 ルーチンから親ルーチン GR2D08 へ処理が戻される．しかし，GR2D50 ルーチンは 1PE 上のみで実行させているため（Fig. 4.50 の 1 行目～ 3 行目，71 行目～ 73 行目），設定したフラグは 1PE 上のみの値になる．VPP500 上で，ある PE のみの値を利用して処理を制御した場合，例えばその制御の先にすべての PE において同期をとる処理が存在するような場合には，永久同期待ちが発生し実行は終了しない．従って，このような場合には，全 PE 上で冗長な値を利用し，冗長部で処理を制御する必要があるため，1PE 上のみに存在する値をすべての PE に転送する必要がある．そのため，親ルーチン GR2D08（Fig. 4.49）の 43 行目～ 47 行目においてその転送を行っている．ここで，ILEGAL2(IK) は全 PE 上に冗長に持たせている配列であり，ILEGAL2_L(IK) は各 PE にローカルに持たせている配列である．GR2D08 では，実行処理ステップ毎にこれらの配列を 0 で初期化しているが，これと SUM(ILEGAL2) によって，

```
ILEGAL2(1) には PE1 の ILEGAL2_L(1),
ILEGAL2(2) には PE2 の ILEGAL2_L(2),
                 •
                 •
ILEGAL2(NPE) には NPE の ILEGAL2_L(NPE)
```

が定義（転送）される．配列 ILEGAL2 の内，1PE 上で定義された値を参照するには，ILEGAL2(1) を参照すればよい．それは，49 行目～ 52 行目において行われているが，フラグに 3 が設定されている場合は，Fig. 4.49 において 51 行目の処理になる．ここで GO TO 3200 により，実行を強制終了するようにしているが，冗長部で行っているため永久同期待ちは発生せず，すべての PE が強制終了する．

ここで，並列化版 GO TO ループの他の制御の仕方について説明すると，Fig. 4.49 の 49 行目に当たる場合は次の実行処理ステップへ進む場合で，Fig. 4.48 では 35 行目か 42 行目または 59 行目に相当する．(ILEGAL2(1).EQ.0) の場合は，初期値から何ら更新が無い場合で，GR2D50 で何も実行しない場合（PE1 以外の実行）を表し，(ILEGAL2(1).EQ.1) の場合は，最終物理時間ステップでない場合を表す．これについては，Fig. 4.50 の 43 行目で 1 が設定される．

Fig. 4.49 の 50 行目は全エネルギーの誤差の許容値が越えた場合に強制終了させる場合で，Fig. 4.48 では 11 行目か 22 行目に相当する．これについては，Fig. 4.50 の 12 行目の GO TO 文により 66 行目で 2 が設定される．

Fig. 4.49 の 52 行目に当たる場合は正常終了の場合で，Fig. 4.48 では 43 行目に相当する．これについては，Fig. 4.50 の 38 行目の GO TO 文により 64 行目で 4 が設定される．

・並列処理構造特有の物理時間ステップ制御について

追加した制御は，Fig. 4.49 において 54 行目～ 63 行目になる．I_TIME1 は並列処理構造中の各処理単位における前物理時間を示し，I_TIME2 は現物理時間を示す．各処理単位において，前物理時間と現物理時間の間で依存関係があるものについては，両変数値が必要になる．例えば，各座標値を格納する配列 X(NPAR,3,NPE-NPE/2,NPE) は，各処理単位で，前物理時間ステップで求められた座標を用い，現物理時間ステップでの座標を求めるからである．

そのため，59，60 行目では，現物理時間ステップを I_TIME1 に，新物理時間として I_TIME2 に 1 を足し，同一 PE の次の処理単位では I_TIME1 を前物理時間，I_TIME2 を現物理時間として利用している．しかし，単純に I_TIME2 を増やしていくと，物理時間ステップ分のデータを格納するメモリ領域が必要になってしまう．そこで，既説のようにデータ領域を別の物理時間ステップのデータにより異常に重複して定義・参照されないようにするためには，NPE-NPE/2 数分のデータ領域だけあればよい．従って，55 行目の IF 文より，(I_TIME2.EQ.NPE-NPE/2) になった場合に，次の新しい物理時間ステップを 1 にし，必要無くなった以前の物理時間ステップのデータ領域を利用するようにしている．

この I_TIME1 と I_TIME2 の更新をフラグが真である PE に対して行わせることにより，各実行処理ステップにおいて，PE 毎に異なる物理時間についてのデータ領域を使用することが可能になる．I_TIME1 と I_TIME2 の初期値はサブルーチン INIT_PARA で定義している．

・物理時間ステップ幅の変更機能について

この機能は，Fig. 4.48 では 44 行目～ 55 行目に相当する．物理時間ステップ幅は 7 パターン設定することができ，それは PART1 から引き継ぐ幅と，DTN(1)...DTN(6) に設定する幅になる．そして幅を変更する時刻は TT(1)...TT(6) に設定される．これらの値は，Fig. 4.48 の 36，37 行目，42，43 行目において参照され，幅を変更する場合に 44 行目以降が実行される．

並列処理構造では PE 毎に物理時間ステップを管理させているため，幅を変更する物理時刻になった PE から順番に幅の変更を行う必要がある．また，Fig. 4.48 の 44 行目～ 46 行目で更新する配列 VX については，PE 毎の原子群について，PE 毎に順番に更新を行う必要がある．この並列化版での物理時間ステップ幅を変更する処理は，2PE ～ nPE 上については Fig. 4.49 の 8 行目～ 38 行目で，1PE 上については GR2D50 ルーチンの 47 行目～ 62 行目で行わせている．

幅を変更する状態は，Fig. 4.48 より，26 行目の条件式が真で，36，37 行目，42，43 行目の条件式がすべて偽になる場合である．並列化版においてもこれらの条件式を Fig. 4.49 の 8 行目～ 38 行目，Fig. 4.50 の 47 行目～ 62 行目で採用している．そして，これらの条件式の判定は PE 毎に独立に行うことができるので，Fig. 4.49 では 9 行目の DO 769 ループを !XOCL SPREAD DO で単純に分割して 2PE ～ nPE 上で行わせ，Fig. 4.50 では 1 行目～ 3 行目と 71 行目～ 73 行目で 1PE 上のみで行わせるようにしている．

Fig. 4.49 の 20 行目の DO ループの回転範囲 I_MIN(I_MYPE)，I_MAX(I_MYPE) は，各 PE における原子群 ID の範囲になる．I_MYPE は PE-ID が格納され，I_MIN(I_MYPE) にはその PE における原子 ID の最小値，I_MAX(I_MYPE) には原子 ID の最大値が格納される．I_MYPE と

各 PE の原子 ID の最小値，最大値はサブルーチン INIT_PARA で定義される．

(3) パラレルリージョン内の処理

パラレルリージョン内の処理構造を Fig. 4.51 に示す．また，今回の並列化で作成したインクルードファイル INC_PE, INC_MAIN, INC_SUB, INC_PARA をそれぞれ Fig. 4.52, Fig. 4.53, Fig. 4.54, Fig. 4.55 に示す．Fig. 4.51 において，a.～c. は GO TO ループでの並列実行処理ステップで用いるデータの準備を行う．d.～l. は並列処理構造になる部分である．以下，各インクルードファイルと a.～l. について述べる．

- INC_PE について

このファイルは，パラメータ文を含む．NPE には使用する PE 台数を指定する．

- INC_MAIN について

このファイルはメインルーチンでインクルードする．配列 ILEGA1, ILEGAL2 はエラー処理用の配列である．

- INC_SUB について

このファイルがメインルーチン以外のサブルーチンでインクルードする．

- INC_PARA について

このファイルはメインルーチン GR2D08，サブルーチン GR2D3, GR2D40, GR2D41, GR2D50 でインクルードする．各変数・配列の説明を次に示す．

X(NPAR,3,NPE-NPE/2,NPE):	各原子の座標
VX(NPAR,3,NPE-NPE/2,NPE):	各原子に関する移動パラメータ
ENGKE(NPE-NPE/2,NPE):	前物理時間ステップでの運動エネルギー
ENGL(NPE-NPE/2,NPE):	結晶表面から散逸するエネルギー
ENGKG(NPE-NPE/2,NPE):	現物理時間ステップでの運動エネルギー
ENGPO(NPE-NPE/2,NPE):	ポテンシャルエネルギー
ENGSP(NPE-NPE/2,NPE):	表面につなぎとめるスプリングエネルギー
ENGLO(NPE-NPE/2,NPE):	各物理時間ステップで蓄積される散逸エネルギー
IIBND(NPAR,NPE-NPE/2,NPE):	各原子の箱番号を格納する
ICOV(NPAR,NPE-NPE/2,NPE):	各原子の計数を格納する
IBOX(6,NBOXES,NPE-NPE/2,NPE):	箱データ（どの箱にどの原子が格納されているか）
ZZ1(NPAR,NPE-NPE/2,NPE):	相手原子の受ける力を格納する（X 方向）
ZZ2(NPAR,NPE-NPE/2,NPE):	相手原子の受ける力を格納する （Y 方向）
ZZ3(NPAR,NPE-NPE/2,NPE):	相手原子の受ける力を格納する（Z 方向）
LSLT(3,NPE-NPE/2,NPE):	時間ステップを変更するかしないかのフラグ
I_MIN(NPE):	各 PE に属する原子 ID の最小値

I_MAX(NPE+1):	各 PE に属する原子 ID の最大値
I_MYPE:	各 PE の ID
I_INIT:	実行フラグを立てる際の制御変数
I_LCOUNT:	並列処理構造の実行処理ステップ数
I_TIME1:	前物理時間ステップを示す
I_TIME2:	現物理時間ステップを示す
FLAG:	実行フラグ
OUT_FLAG:	OUTPUT フラグ
DT_HOLD(2):	リスタート時の時刻と物理時間ステップ幅
I_UNTIL(2,NPE):	前実行での終了時刻

a. サブルーチン INIT_PARA

このルーチンでは，並列処理構造を制御するための制御変数を設定している．このルーチンを Fig. 4.56 に示す．11 行目～ 16 行目で求められる I_MYPE と I_INIT には各 PE で異なる値になる．I_MYPE には PE-ID が，I_INIT はその逆で 1PE 上では NPE-ID の値になり NPE 目の I_INIT は 1 になる．18 行目～ 28 行目では各 PE に属する原子 ID の最小値と最大値を求めている．30 行目～ 37 行目では実行処理ステップ数と物理時間ステップを表す変数の初期化である．1PE 実行時と 2PE 実行時にはデータ領域の異常な定義・参照を考える必要はないので，前物理時間ステップと現物理時間ステップを区別する必要はない．39 行目～ 43 行目で時間ステップを変更するかしないかのフラグを初期化しているが，このフラグは並列化版の配列に格納する必要があるため，オリジナル版で使用していた配列名を LSLT から LSLT2 へ変更してある．

b. 読み込んだ座標とエネルギーの並列化版配列への代入

PART1 から引き継ぐデータ，または前の実行から引き継ぐデータ（リスタート時）の内，並列処理構造に合わせた並列化版データとして取り扱わなければならないものには，X, VX, ENGKE, ENGLO がある．そこで，READ 用の配列から並列処理構造で利用される並列化版の配列への代入が必要になる．この部分を Fig. 4.57 に示す．

3 行目～ 7 行目は X, VX の並列化版配列への代入，8 行目～ 14 行目が ENGKE, ENGLO の並列化版配列への代入である．8 行目の (I_FIRST.EQ.0) はリスタート実行でない場合を示す．この I_FIRST はリスタートデータを読み込むルーチンを実行しない場合に 0 を代入している．ここで代入されているすべてのデータ並列処理構造の第 1 物理時間ステップでの初期値になるため，I_TIME1 を利用している．また，並列化指示行 !XOCL SPREAD DO で DO 40 ループの回転数（PE の数）が分割されることにより，各 PE 上にすべて同じ値の物理値が格納される．

c. 箱データの初期化

ここは，並列処理構造の第 1 物理時間ステップで用いる箱データを求める部分である．これを Fig. 4.58 に示す．

1 行目～ 9 行目がどの箱にどのような原子が格納されているかを管理する配列 IBOX の初期化

を行っている．3 行目の ILEGAL1_L はサブルーチン GR2D3 中でエラーが発生した場合にフラグを格納する配列である．この初期化は，DO 39 ループに対し，!XOCL SPREAD DO を指示することで，全 PE で行われる．

11 行目～ 13 行目はサブルーチン GR2D3 を CALL し，各原子の初期座標（READ したもの）に対し，箱格納を行う．

ここで，並列化版サブルーチン GR2D3 を Fig. 4.59 に示す．このルーチンは，並列処理構造の実行処理ステップで利用することができるようなコーディングをしてある．よって，実行フラグが真である PE 上での実行しか行わないようにしている．それは，3 行目の IF 文で制御されるが，このフラグの値は ここで CALL される前に親ルーチンによって真にされるため，全 PE 上で実行されることになる．第 1 回目の CALL では，ISW の値により 5 行目～ 8 行目が実行され，ここでは全原子（I_START=1 ～ I_END=N）を対象に箱格納を行う．求められた値は，並列処理構造の第 1 物理時間ステップでの初期値になるため，（I_TIMEA=I_TIME1）に設定する．14 行目以降では全原子を対象にした箱格納が全 PE 上で行われることになり，各 PE 上ではシングル実行時と同様の実行が行われる．14 行目～ 21 行目は各原子が属するべき箱番号を求める処理で，22 行目～ 42 行目は配列 IBOX に最大 6 個の原子まで格納する処理と配列 IIBND に各原子が属する箱番号を格納する処理になる．43 行目～ 47 行目で IER に値が代入されているが，0 なら本ルーチンの正常終了，1 なら異常な箱番号が発生，2 なら 1 つの箱に 7 個以上原子が格納された場合を意味する．それらの値を配列 ILEGAL1_L に代入する．この配列は分割ローカルデータであり，各 PE 上での値が格納される．

Fig. 4.58 の 18 行目～ 28 行目が GR2D3 で格納された ILEGAL1_L をチェックし，エラー処理を行う部分である．この部分ではローカル分割データ ILEGAL1_L 中の値をすべての PE に転送する処理を行っている．これは既説のとおり，同期による永久待ち状態を防ぐためである．18 行目～ 20 行目で配列 ILEGAL1 を 0 で初期化し，22 行目～ 25 行目の転送により 0 以外の値が転送されればエラーである．その場合 26 行目～ 28 行目の処理によって GO TO 800 が実行され強制終了される．

d. フラグの設定と変数の初期化

この d. から以降の l. までが実行処理ステップ内の処理になる．d. の部分を Fig. 4.60 に示す．

1 行目～ 3 行目の内，I_LCOUNT は実行処理ステップの回数をカウントし，FLAG は実行を行わせる PE 上だけで真にする実行フラグ，OUT_FLAG は実行結果をファイルに書き出す（実行処理ステップ 50 回に 1 回）タイミングの時に真にする output フラグである．これらの変数は全 PE 上に持たせ，且つ同一の値で初期化しておく．

4 行目～ 19 行目は !XOCL SPREAD DO により PE 毎に処理させる部分である．ILEGAL1_L は箱格納処理のエラーフラグを保持する配列，ILEGAL2_L は全体の制御をする GO TO ループの制御値を保持する配列，T_UNTIL_L は検索主原子が 2PE 分隣の PE 上の原子を検索した時点の物理時刻と時間ステップ幅を保持する配列であり，ここで初期化を行っている．

11 行目～ 17 行目は各実行処理ステップでどの PE を実行させるかの実行フラグを立てる部分である．11 行目の IF 文の条件式中の (I_LCOUNT.GE.I_INIT) は実行処理ステップ毎に NPE 目

から順番に真になり，各 PE での第 1 物理時間ステップの実行開始タイミングを測る．一方，(T.LT.TT(NT)+DT.OR.DTN(NT).NE.0.0D0) は実行終了タイミングを測る．並列実行で使用する PE 台数が奇数か偶数かによって実行フラグの立て方は異なる．奇数台数使用時は，実行処理ステップ回数が奇数回目の時は奇数 PE，実行処理ステップ回数が偶数回目の時は偶数 PE に対し実行フラグを立てる必要がる．一方，偶数台数使用時は，実行処理ステップ回数が偶数回目の時は奇数 PE，実行処理ステップ回数が奇数回目の時は偶数 PE に対し実行フラグを立てる必要がある．13 行目～ 16 行目はこの法則性に従って実行フラグを立てている．しかし，この制御では 1PE 実行時のフラグが異常になるため，6 行目において 1PE 実行時用のフラグを立てることにした．20 行目～ 25 行目は，配列 ILEGAL1_L，ILEGAL2_L，T_UNTIL_L に対して各 PE で冗長に持たせている配列 ILEGAL1，ILEGAL2，T_UNTIL の初期化部分である．

e. 各物理時間ステップで必要になるデータの初期化

この部分を Fig. 4.61 に示す．ここでは，物理時間ステップ毎に初期化が必要なデータについてその初期化を行っている．並列化指示行 !XOCL SPREAD DO と IF(FLAG.AND.I_MYPE.EQ.NPE) 文により，NPE 上の実行フラグが真の場合のみ実行が行われる．この場合は各物理時間ステップの開始タイミングに相当する．

f. サブルーチン GR2D40

並列化版のこのルーチンを Fig. 4.62 に示す．新ベクトル化版から変更したのは，配列 ICOV の初期化を GR2D08 ルーチンに移したことと，インクルードファイル INC_PARA に宣言した並列化版用の配列を利用したこと，各ループの回転範囲を各 PE 上の原子群の範囲にしたこと，新ベクトル化版の GR2D50 ルーチンの現物理時間の各原子の座標を求める部分と物理時間の更新部分をこのルーチンの最後に追加したことである ((1) を参照)．また，各ループにおいて，結晶表面から散逸するエネルギー ENGL の並列化版配列と，スプリングエネルギー ENGSP の並列化版配列をループから外している．これは，これらの並列化版配列をループ中で利用するとベクトル化されないためである．また，このルーチンは実行フラグが真の PE 上でのみ処理すればいいので，並列化指示行 !XOCL SPREAD DO で DO 9999 ループを分割し，その後の IF 文で実行の制御を行うようにした．

g. サブルーチン GR2D41

このルーチンはステップ数が多いので図には示さないが，新ベクトル化版から変更したのは，GR2D40 と同様に，INC_PARA に定義した並列化版用の配列を利用したこと，各ループの回転範囲を各 PE 上の原子群の範囲にしたこと，本ルーチンをすべて PE 台数分回転する DO 9999 ループで囲み，それを並列化指示行 !XOCL SPREAD DO で分割し，フラグが真の PE のみを実行させるように IF 文で制御したことである．また，このルーチンには箱検索処理が存在するため，検索主原子が 2PE 分隣の PE 上の原子を検索した場合に強制終了させる処理を検索主原子と被検索原子の距離を求める処理部分に追加している．この追加した処理のみを Fig. 4.63に示す．検索主原子と被検索原子の距離を求める部分において，相互作用の範囲内にある被検索原子中に，2PE 分隣の PE 上の原子が存在する場合に I_AREA_HOLD にフラグ 1 を立てる．この場

合 49 行目～ 53 行目により，その時点の物理時刻と物理時間幅を配列 T_UNTIL_L に格納しこのルーチンを終了する．

h. 強制終了処理

検索主原子が 2PE 分隣の PE 上の原子を検索した場合，並列化版では使用する PE 台数を前実行より少なく設定してリスタート実行を行わなければならないことになった．この場合前実行で何物理時刻まで実行が行われたかとその時の物理時間幅，リスタートを開始する物理時刻とその時の物理時間幅をメッセージとして出力して強制終了させるようにした．この部分を Fig. 4.64に示す．1 行目～ 7 行目は GR2D40 ルーチンで格納された物理時刻と物理時間幅を全 PE 上に転送する処理である．8 行目～ 16 行目により，2PE 分隣の PE 上の原子を検索した物理時刻 T_NOW，その時に物理時間幅 DT_NOW，リスタート時の開始物理時刻 T_REST，その時の物理時間幅 DT_REST をそれぞれ求めて，GO TO 3300 でメッセージを出力して終了させている．DT_HOLD(1) にはリスタート実行時に読み込む前実行の計算結果（ファイル）の最終レコードの物理時刻が格納され，DT_HOLD(2) にはその時の物理時間幅が格納されてるが，それらは，ファイルへの書き出しを行う GR2D50 ルーチンで格納する（k. を参照）．尚，メッセージは次のように出力することにした．

```
THERE ARE SOME WRONG PARTICLES IN GR2D41 ON PROCESSOR 4.
IT HAPPENS AT T=1.00 AND DT=0.02.THEY DON'T HAVE TO SELECT.
RESTART AFTER MODIFING SUBROUTINE GR2D20 AND TEXT INPUT DATA FOR PART2.
ABOUT THEIR CONTENTS TO MODIFY, REFER T=0.94, DT=0.03.
```

i. GR2D40 後に決まるデータの転送

ここでは，既説の A.～E. のデータの内，A.～F. のデータの転送を行う．この部分を Fig. 4.65に示す．ここは，2PE ～ nPE 上で転送するデータと 1PE から (n − 1)PE 上で転送するデータに別れる．

3 行目～ 23 行目が 2PE ～ nPE 上で転送するデータで，配列 ICOV, ZZ1, ZZ2, ZZ3, ENGL, ENGSP, ENGPO, ENGKG がある．これらのデータは既説の通り左隣の PE にのみ転送を行えばよいので，自 PE (I_MYPE) から左隣の PE (I_MYPE-1) の転送をしている．左隣の PE (I_MYPE-1) 上のデータは自 PE (I_MYPE) 上には存在しないため，それぞれグローバルデータ ICOV_G, ZZ1_G, ZZ2_G, ZZ3_G, ENGL_G, ENGSP_G, ENGPO_G, ENGKG_G への転送をしなければならない．従って，ここではローカルデータとグローバルデータ間の転送（代入）を行うことができる SPREAD MOVE 転送を利用することにした．ここではグローバルデータへの転送をしているが，転送されたデータは左隣の PE 上ではローカルデータとして扱えることができる．

24 行目～ 41 行目が 1PE ～ (n − 1)PE 上で転送するデータで，配列 X, VX がある．これらのデータは既説の通り両側の PE に転送を行う必要がある．25 行目～ 32 行目は自 PE (I_MYPE) 上の原子群についてのみ計算した X, VX について右隣の PE (I_MYPE+1) 上へ転送している．この直後では，右隣の PE 上の X, VX には現物理時間についてのすべての X, VX の計算結果が存在することになる．これら X, VX は 1PE 上でファイルに書き出す必要がある

ため，34 行目～ 40 行目において右隣の X，VX のすべての範囲の自 PE 上への転送を行っている．

j. 新しい原子座標についての箱格納

この部分を Fig. 4.66に示す．ここでは，相互作用の計算の結果新たに求められた原子の座標について，次の物理時間ステップで用いる箱データを作成するため，箱格納処理を行う．

1 行目～ 10 行目は配列 IBOX の初期化である．これは，各物理時間ステップの開始時に行っておけばよいので，NPE 上でのみ行わせている．

11 行目～ 25 行目は 1PE ～ (n − 1)PE 上で行われる箱データの転送である．これは，既説の通り，箱データは右隣の PE 上からすべての範囲の転送が必要になる．ここでも SPREAD MOVE 転送を行っている．26 行目はサブルーチン GR2D3 を CALL し，転送されたデータを参照して自 PE 上の原子群について箱格納処理を行う．ここから CALL された場合，Fig. 4.59 においては 10 行目～ 12 行目の部分が実行され，I_START には自 PE 上の原子群の最小値，I_END には自 PE 上の原子群の最大値が，I_TIMEA には現物理時間を表す I_TIME2 が代入される．

35 行目～ 49 行目では自 PE 上の原子群について求めた箱データを右隣の PE に転送を行っている．

k. サブルーチン GR2D50

このルーチンには，物理時間毎に全原子についての計算が終了しなければ行えない処理をまとめてある．それは各種エネルギーの計算とそれらの和による全エネルギーの計算，ファイルへの計算結果の出力（サブルーチン GR2D60 を展開したもの）を行う処理である．全エネルギーの誤差が許容値を越えた場合の処理や全体の処理制御を行う処理については既に述べているので，ここでは各種エネルギーについての計算とファイルへの計算結果の出力について述べる．その部分を Fig. 4.67に示す．

5 行目～ 18 行目が各種エネルギーを求める部分である．各物理時間の計算過程で，1PE ～ nPE 上の各 PE 上の原子群について計算された各エネルギーはすべて 1PE 上に転送され，この部分で求められる．ENGKA は前物理時間の運動エネルギーと現物理時間の運動エネルギーの平均であるが，この部分だけで求めればよいので，並列化版配列を用意する必要はない．同様に，この部分だけで求めればよいのは，全エネルギーが格納される ENGTO，前物理時間の全エネルギーと現物理時間の全エネルギーの誤差が格納される DISCR がある．

4 行目の IOUT_COUNT は物理時間ステップの回数が求められる．新ベクトル化版では，物理時間ステップ 2 回に 1 回ファイルへの書き出しを行っていたが，並列化版では 50 回に 1 回書き出せばよいことになったため，そのステップをカウントし，50 になったらファイルへ書き出すようにしている．そのファイルへの書き出しは 23 行目～ 55 行目で行っている．13 行目～ 35 行目が機番 19 に結合されているファイルへの書き出しで，41 行目～ 52 行目が機番 18 または 17 に結合されているファイルへの書き出しである．17 番へは，検索主原子が 2PE 分隣の PE 上の原子を検索した場合のリスタート実行において必要であれば使用される ((4) 参照)．ここでは各機番に対して書き出すデータは同一である．書き出す回数になったら，IOUT_COUNT を 0 に初期化し，配列 DT_HOLD と OUT_FLAG を更新する．検索主原子が 2PE 分隣の PE 上の原子

を検索した場合のリスタート実行では，前の実行で書き出されたファイル中の最終レコードは何物理時刻であるかを知る必要があるため，ここでは，DT_HOLD に書き出す時の物理時刻 T と物理時間幅 DT を代入している．また，この DT_HOLD は本ルーチンでのみ代入が行われるため，1PE 上のみの値になってしまう．従って，永久同期待ちを防ぐためこのデータもすべての PE に転送を行う必要があるが，この転送には BROADCAST を用いるため，ここでは OUT_FLAG を真にしている．

1. 配列 DT_HOLD の転送

GR2D50 から処理が戻った直後に，DT_HOLD の転送を行う．これを，Fig.4.68に示す．ここでは BROADCAST 転送を行っているが，OUT_FLAG が真である PE 上の配列 DT_HOLD の値がすべての PE に転送される．今までは，配列 ILEGAL1_L や ILEGAL2_L のすべての PE への転送には !XOCL SPREAD DO ～ !XOCL END SPREAD SUM(データ名) を利用していた．この方法は，エラーメッセージにエラーが発生した PE-ID を付ける必要性がある場合に利用している．

(4) サブルーチン GR2D20

このルーチンはリスタート実行時に，前の実行結果をファイルから読み込むルーチンである．

PART2 の実行結果が書き出されるファイルの先頭部分（いくつかのレコード）には，PART1 から引き継いだデータ（シミュレーションの初期値）が格納され，それ以降の部分に PART2 の実行結果（各物理時間の計算結果）が格納される．シミュレーションの初期値は，PART3 で作図する場合に用いるデータである．PART2 の実行で最終物理時刻についての計算結果をファイルに格納する際には，INSEN という変数に 0 が設定される．この INSEN は通常のリスタート実行時に前の実行結果の読み込みを終了させるフラグになる．

しかし，並列実行では，検索主原子が 2PE 分隣の PE 上の原子を検索する場合があり，それはどの PE 上で発生するか予測できないため，INSEN に 0 を設定することができない．そこで，この場合のリスタート実行時には，前の実行結果が格納されているファイルの最終レコードの物理時刻を知り，それを読み込みを終了させるフラグにすることにした．最終レコードの物理時刻については，前実行での終了メッセージから取得することができる．

ここで，並列化版 GR2D20 を Fig. 4.69 に示す．上半分，1 行目～ 32 行目が通常のリスタート実行で用いる部分，下半分，33 行目～ 72 行目が検索主原子が 2PE 分隣の PE 上の原子を検索した場合のリスタート実行で用いる部分である．これらは，コメント化または有効化してそれぞれの実行で使い分ける必要がある．以下，下半分について説明する．

PART2 の実行では，書き出し先の機番を区別するフラグが設けられており，それが 0 の時は機番 18 に，1 の時は機番 19 に書き出すようになっている．しかし，データを読み込む機番は常に 18 である．従って，フラグが 0 の場合には，機番 18 に対して上書きされる．これは，検索主原子が 2PE 分隣の PE 上の原子を検索した場合のリスタート実行時に問題になる．この実行では，前の実行と同一な実行を行わせるようにするため，前の実行で書き出したデータを再度書き出すようにしている．そのため，読み込むべきファイルに不正に上書きが行われるからである．そのような上書きが行われないように，Fig. 4.69 の 37 行目において新しい機番 17 を用意してそれに書き出すようにした．38 行目～ 48 行目はシミュレーションの初期値を機番 17 に

・サブルーチン GR2D50 （一部）のサブルーチン GR2D40 への移動について

新ベクトル化版の GR2D50 を Fig. 4.45 に示す．ここで DO 1000 ループが各原子の新しい座標値を求める部分，7 行目～ 17 行目が各種エネルギーを求める部分である．18 行目は新しい物理時間を求めるために，時間ステップ幅を現物理時間に加える部分で，他は並列化で省くことになったクエンチングを行う部分である．このサブルーチン内で並列処理構造で並列化の必要があるのは，各原子の新しい座標値を求める部分と物理時間の更新部分である．各種エネルギー変数については，物理時間ステップ毎の全原子についての計算が終了しなければ求められないため，並列処理構造においては，1PE 上のみで求めればよく並列化の必要がない．従って，各原子の新しい座標を求める部分と物理時間の更新部分を GR2D50 から削除する代わりにサブルーチン GR2D40 の最後の部分に追加し，GR2D50 を 1PE 上のみで実行させるようにした．

・サブルーチン GR2D60 の消去について

このルーチンはメインルーチンから CALL されるサブルーチンである．このルーチンを Fig. 4.46 に示す．また，メインルーチンからの CALL 部分を Fig. 4.47 に示す．Fig. 4.47 において GR2D60 を CALL する前に，8 行目において全エネルギーの誤差のチェックが存在する．この全エネルギーは 1PE 上だけで求めればよいものであり，GR2D60 も WRITE 文しかないので 1PE 上だけで処理すればよいものである．

従って，GR2D50 から GR2D60 まで 1PE 上で求めればよいものはすべてサブルーチン GR2D50 にまとめることにした．そうすると，GR2D50 を並列化指示行 !XOCL SPREAD DO により 1PE だけで実行させるように制御しサブルーチン GR2D60 を CALL することになるが，!XOCL SPREAD DO 内からサブルーチンを CALL することは VPP500 の文法上好ましくない．そのために，サブルーチン GR2D60 の内容をすべて GR2D50 ルーチンに展開し，GR2D60 ルーチンは削除することになった．

(2) メインルーチンで行われていた全体の処理制御について

ここでは，メインルーチンで制御される，新ベクトル化版の物理時間ステップ（GO TO ループ）から並列処理構造の実行処理ステップ（GO TO ループ）へのループ制御方法の変更について，変更内容を 3 つに分けて述べる．1 つ目は新ベクトル化版の GO TO ループと全く同様な制御を行う並列処理構造版 GO TO ループへの変更について，2 つ目はその GO TO ループに対して追加した並列処理構造特有の物理時間ステップ制御について，3 つ目は実行中に物理時間ステップ幅を変更する機能についてである．

・並列化版 GO TO ループへの変更について

新ベクトル化版の GO TO ループを Fig. 4.48 に，並列化版の GO TO ループを Fig. 4.49, Fig. 4.50に示す．Fig. 4.48 は新ベクトル化版のメインルーチン GR2D08 に存在する．Fig. 4.49 は同様に並列化版のメインルーチン GR2D08 に存在し，Fig. 4.50 は並列化版のサブルーチン GR2D50 に存在するものである．

Fig. 4.48 の 8 行目～ 40 行目は，Fig. 4.50 の 10 行目～ 36 行目に相当する．その内，Fig. 4.48 の 13 行目～ 24 行目は“1 ステップ戻って計算を行う機能”であるが，並列化版では

対して書き出す部分である．これは，機番 18 からデータを読み込み，機番 19 へ書き出す場合と同じ方法になる．

前の実行の最終レコードの物理時刻は，68 行目の条件式中に設定すればよい．これにより，この物理時刻のレコードが読み込まれるまで，49 行目の READ による前の実行結果とファイルへのデータの書き出しが繰り返される．更に，前の実行の終了メッセージから取得できる最終レコードの物理時間幅を 69 行目に設定することにより，このリスタート実行が，前の実行の最終物理時刻のデータを初期値として開始されることになる．

(5) 変数・配列宣言について

並列化版では新ベクトル化版で利用していた変数・配列をそのまま利用せずに，インクルードファイル INC_PARA に宣言した並列化版固有の変数・配列を利用することになった．これに伴い，その他の変数・配列の宣言も変更している．

・既存のコモン変数の変更

新ベクトル化版では，コモンブロック GEOM2 においてコモン変数 GEOM2 が，コモンブロック DYNA においてコモン変数 DYNA が宣言されていた．各サブルーチンに跨って使用される変数・配列をそれらのコモン変数と EQUIVALENCE を行って使用していた．しかし，並列化に伴い，インクルードファイル INC_PARA に並列化版用に宣言した配列は，それらのコモン変数との EQUIVALENCE から省くことにした．これに伴い各コモン変数のサイズも変更した．コモン変数 GEOM2 の変更後を Fig. 4.70に，DYNA の変更後を Fig. 4.71に示す．

・新たに追加したコモンブロックについて

並列化版で新たに追加したコモンブロックは INC_PARA 以外では次の 2 つである．

```
COMMON /PARA_READ/X_R(NPAR,3),VX_R(NPAR,3),ENGKE_R,ENGLO_R
COMMON /PARA_OUT/IUNTNN
```

PARA_READ のそれぞれのコモン変数はファイルからデータを読み込む場合に使用し，また並列化版のそれぞれの配列への代入にも使用される．このコモンブロックは，サブルーチン GR2D08, GR2D10, GR2D20 で宣言している．PARA_OUT の コモン変数 IUNTNN はリスタート実行時の書き出し先の機番を保持する（18 または 17）．これはサブルーチン GR2D08, GR2D20, GR2D50 で宣言している．

4.6.3.4 Fig. 4.38 から Fig. 4.37 への変更について

原子ループの分割の際，本来ならば Fig.4.38 （1PE から右下）の構造にするところを，今回は Fig. 4.37 （NPE から左下）の構造にし，これを基にして並列処理構造を確立した．これは，各物理時間ステップで求められた計算結果をファイルへ書き出す処理が，1PE 上で行わせた場合と 2PE ～ nPE 上で行わせた場合では，1PE 上で行わせた方が書き出しの処理時間が短縮できたためである．並列化版コードでは，サブルーチン GR2D50 においてファイルへの書き出し処理を行っているが，ここの WRITE 部分だけを抜きだし，1PE ～ 4PE の各 PE 上での WRITE

時間を測定した．これに使用したテストプログラムを Fig. 4.72 に示す．ここで，1PE 上のみで WRITE を行わせる場合には，45 行目の IF 文の条件式中の？ の値を 1 にし，2PE 上のみで行わせる場合には 2 にしてテスト実行を行った（3PE，4PE も同様）．測定結果を Table. 4.3 に示す．

4.7 計算結果の評価及び並列化の効果

ここでは，並列化版コードの計算結果と並列化の効果について述べる．

4.7.1 計算結果について

新ベクトル化版コードのベクトル実行結果と，並列化版コードのベクトル並列実行結果を比較した．並列化版コードの実行は，PE を 1 台使用した場合，2 台使用した場合，4 台，8 台，16 台使用した場合のそれぞれのベクトル並列実行を行い比較を行った．入力データは，結晶サイズ 16x16x16 で強制的に結合させる原子の指定無しで行った．その結果，計算結果は一致していた．

4.7.2 並列化の効果について

並列化版 DGR コード (PART2) の性能を測るため，VPP500 上において，

(1) オリジナル版コードのベクトル実行
(2) ベクトル化版コードのベクトル実行
(3) 並列化版コードのベクトル実行
(4) 並列化版コードのベクトル並列実行（1PE のみ使用）
(5) 並列化版コードのベクトル並列実行（2PE を使用）
(6) 並列化版コードのベクトル並列実行（4PE を使用）
(7) 並列化版コードのベクトル並列実行（8PE を使用）
(8) 並列化版コードのベクトル並列実行（16PE を使用）

を行い，それぞれの実行時間を測定した．使用した入力データは，結晶サイズ 16x16x16（強制的に結合させる原子の指定無し）の大きさで，物理時間ステップ幅 0.02 で 20.0 までシミュレーションを行う設定とした．更に，入出力の高速化を図るため，FORTRAN バッファを 8MB（システム上最大サイズ）とし，SCFS による I/O を採用した [8]．実行時間は，プログラム終了直前の経過時間と開始直後の経過時間の差とし，終了直前と開始直後にそれぞれ経過時間を取得できるサービスルーチン gettod を挿入して計測した．計測した結果を Table 4.4に示す．

・(2) → (3) への時間増について

この時間増は，毎時間ステップで箱検索を行うか行わないかの違いによるものである．(2) のベクトル化版では，原子が前時間ステップの箱からずれた場合のみ箱検索を行っているが，(3) の並列化版では，毎時間ステップで箱検索を行っている．箱からずれたかずれないかを並列処理構造で管理するには，何ステップか戻ることが必要になるため（今回の並列化で省いたクエンチ

ング機能と同様），管理は行わなかった．箱検索の処理はベクトル化不可能なためにこのような時間増になっている．

以上のように，(2) から (3) への時間増はあるが，16PE を使用したベクトル並列実行では，新ベクトル化版コードのベクトル実行に比較して約 3 倍，オリジナル版コードのベクトル実行に比較すると，約 12 倍の速度性能の向上が得られた．今回の並列処理構造での理論上の最大性能向上は，倍率では PE 台数 /2 倍であり，並列化効率では 50 %である．ここで，並列処理構造による並列化のみによる効果を評価するため，(4) と (8) を比較すると，倍率は約 5.5 倍，並列化効率は約 34 % である．理論上の性能に達していないのは，並列処理構造を制御するためのデータの準備や，ある PE 上の処理でエラーが発生した場合のエラー処理，並列処理構造上の物理時間 T1 での 3PE ～ nPE までの処理（この部分はシングル実行と同じ）等のオーバーヘッドの時間が含まれることが原因と考えられる．

4.8 まとめ

今回の DGR コードの高速化作業は，当初予定されていなかったオリジナル版コードの修正とベクトル化のやり直し作業も行ったため，作業時間が増えてしまった．しかし，それらの作業の時間分コードの内容を充分に検討することができたため，原子ループと物理時間ステップの依存関係を考慮した上での分割と，その並列処理構造における実行制御により，ベクトル並列化により高速化することができた．また，ユーザにはコード内容の検討の際には多大な御協力をして頂き，更に今回の並列処理構造の基になる案も提供して頂いた．この案に基づいた今回の並列処理構造は，従来ではあまり事例が無いものと思われる方法である．この方法は，今後の並列化においての 1 つの有効な手法として期待できるものと考える．

Table 4.1 Difference of M780 from VPP500

	M780	VPP
OS	MSP	UXP/M(UNIX)
データ形式	数値:M, 文字:EBCDIC	数値:IEEE, 文字:ASCII
コンパイラ	FORTRAN77EX	VPP FORTRAN77EX/VP

Table 4.2 Measured results of processing time and input data

入力データ	VPP500(オリジナル版)	VPP500(ベクトル化版)
A	2.5 sec	1.4 sec
B	8.3 sec	3.2 sec
C	19.5 sec	6.8 sec
D	37.5 sec	12.3 sec
E	65.3 sec	20.9 sec
F	103.6 sec	31.8 sec
G	155.3 sec	47.5 sec

入力データ：

A: 結晶サイズ 4x4x4 原子数 95
B: 結晶サイズ 6x6x6 原子数 280
C: 結晶サイズ 8x8x8 原子数 621
D: 結晶サイズ 10x10x10 原子数 1166
E: 結晶サイズ 12x12x12 原子数 1963
F: 結晶サイズ 14x14x14 原子数 3060
G: 結晶サイズ 16x16x16 原子数 4505

Table 4.3 Measured results of WRITE processing time

1PE_write	2PE_write	3PE_write	4PE_write
19.27sec	80.24sec	78.20sec	78.28sec

Table 4.4 Measured results of processing time

exe_type	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
total	733 sec	190 sec	319 sec	338 sec	338 sec	182 sec	104 sec	62 sec

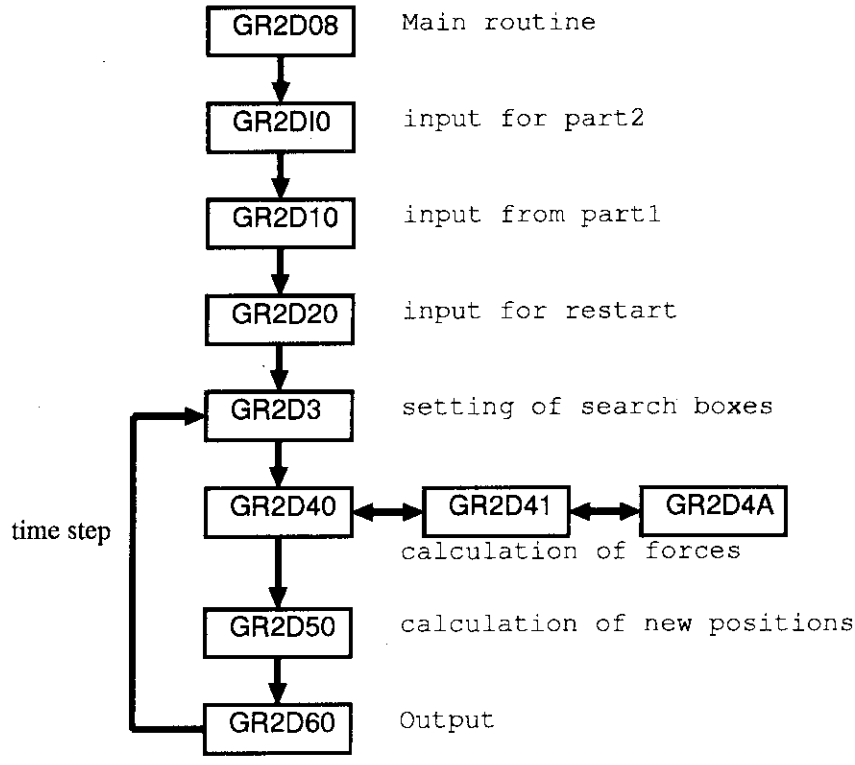


Fig. 4.1 Original flow chart of PART2

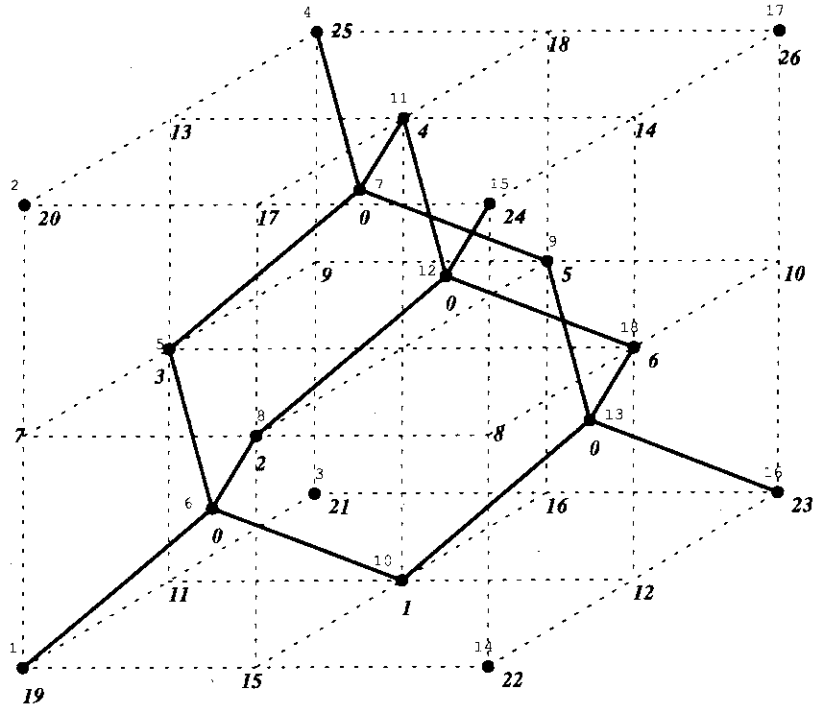


Fig. 4.2 Example of a diamond crystallization

```
C.VPP
      REAL*4 XINT4,YINT4,X114,Y114

             .
             .
             .
             .
C.VPP
      XINT4=XINT
      YINT4=YINT
      CALL PLOT(XINT4,YINT4,-3)

             .
             .
             .
             .
```

Fig. 4.3 Modification of actual arguments from the double precision to the single precision in subroutine GR3D00 and XYPOUT

```
M780 :

                          .
                          .
      IF(X0(IX0(I),1).EQ.XYP0(I,1).AND.X0(IX0(I),2).EQ.XYP0(I,2))
     1                                    GO TO 120
                          .
                          .
                          ↓ 変更
VPP500 :
                          .
      REAL*8 TMP1, TMP2

                          .
                          .
C.VPP
      TMP1=DABS(X0(IX0(I),1)-XYP0(I,1))
      TMP2=DABS(X0(IX0(I),2)-XYP0(I,2))
      IF(TMP1.LE.1.D-11.AND.TMP2.LE.1.D-11) GO TO 120
                          .
                          .
```

Fig. 4.4 Modification of a judgment method in subroutine GR3D00

```
                    .
                    .
COMMON /GEOM/GEOM(19032)
                    .
                    .
EQUIVALENCE  (GEOM(1),IA),
.            (GEOM(2),IB),
.            (GEOM(3),IC),
.            (GEOM(13),IALP),
.            (GEOM(14),IBET),
.            (GEOM(15),IGAM),
.            (GEOM(16),TB(1,1)),
.            (GEOM(12016),IBND(1)),
.            (GEOM(16016),IIB(1)),
.            (GEOM(16042),IK(1)),
.            (GEOM(18777),IAUX(1)),
.            (GEOM(18804),IDUX(1)),
.            (GEOM(18831),NV),
.            (GEOM(18832),NR),
.            (GEOM(18883),INV(1)),
.            (GEOM(18983),IV(1))
                    .
                    .
```

Fig. 4.5 Original common block GEOM of PART1

```
                    .
                    .
COMMON /DYNA/DYNA(39088)
                    .
                    .
EQUIVALENCE  (DYNA(1),X(1,1)),
.            (DYNA(12001),VX(1,1)),
.            (DYNA(24001),XM(1,1)),
.            (DYNA(24301),D(1,1)),
.            (DYNA(24601),E(1)),
.            (DYNA(27601),EE(1)),
.            (DYNA(30601),A1(1)),
.            (DYNA(30604),B1(1)),
.            (DYNA(30607),RMIN(1)),
.            (DYNA(30610),RMAX(1)),
.            (DYNA(30613),E1(1)),
.            (DYNA(30616),E2(1)),
.            (DYNA(30619),A(1)),
.            (DYNA(30697),B(1)),
.            (DYNA(30775),C(1)),
.            (DYNA(30853),A2(1)),
.            (DYNA(30931),B2(1)),
.            (DYNA(31009),C2(1)),
.            (DYNA(31087),ENGKE),
.            (DYNA(31088),MOV)
                    .
                    .
```

Fig. 4.6 Original common block DYNA of PART1

```
                    .
                    .
COMMON /GEOM/GEOM(3032+NPAR+NWPAR)
                    .
                    .
EQUIVALENCE  (GEOM(1),IA),
.            (GEOM(2),IB),
.            (GEOM(3),IC),
.            (GEOM(13),IALP),
.            (GEOM(14),IBET),
.            (GEOM(15),IGAM),
.            (GEOM(16),TB(1,1)),
.            (GEOM(NWPAR+16),IBND(1)),
.            (GEOM(NWPAR+NPAR+16),IIB(1)),
.            (GEOM(NWPAR+NPAR+42),IK(1)),
.            (GEOM(NWPAR+NPAR+2777),IAUX(1)),
.            (GEOM(NWPAR+NPAR+2804),IDUX(1)),
.            (GEOM(NWPAR+NPAR+2831),NV),
.            (GEOM(NWPAR+NPAR+2832),NR),
.            (GEOM(NWPAR+NPAR+2883),INV(1)),
.            (GEOM(NWPAR+NPAR+2983),IV(1))
                    .
                    .
```

Fig. 4.7 Declaration of common block GEOM and EQUIVALENCE statement by using parameters of PART1

```
                    .
                    .
      COMMON /DYNA/DYNA(15088+2*NWPAR)
                    .
                    .
      EQUIVALENCE  (DYNA(1),X(1,1)),
     .             (DYNA(NWPAR+1),VX(1,1)),
     .             (DYNA(2*NWPAR+1),XM(1,1)),
     .             (DYNA(2*NWPAR+301),D(1,1)),
     .             (DYNA(2*NWPAR+601),E(1)),
     .             (DYNA(2*NWPAR+3601),EE(1)),
     .             (DYNA(2*NWPAR+6601),A1(1)),
     .             (DYNA(2*NWPAR+6604),B1(1)),
     .             (DYNA(2*NWPAR+6607),RMIN(1)),
     .             (DYNA(2*NWPAR+6610),RMAX(1)),
     .             (DYNA(2*NWPAR+6613),E1(1)),
     .             (DYNA(2*NWPAR+6616),E2(1)),
     .             (DYNA(2*NWPAR+6619),A(1)),
     .             (DYNA(2*NWPAR+6697),B(1)),
     .             (DYNA(2*NWPAR+6775),C(1)),
     .             (DYNA(2*NWPAR+6853),A2(1)),
     .             (DYNA(2*NWPAR+6931),B2(1)),
     .             (DYNA(2*NWPAR+7009),C2(1)),
     .             (DYNA(2*NWPAR+7087),ENGKE),
     .             (DYNA(2*NWPAR+7088),MOV)
                    .
                    .
```

Fig. 4.8 Declaration of common block DYNA and EQUIVALENCE statement by using parameters of PART1

```
C.VP26DO 1000 I=1,4000
C.VPP
      DO 1000 I=1,NPAR
      IBND(I)=0
      DO 1000 J=1,3
      TB(I,J)=0.DO
      X(I,J)=0.DO
 1000 VX(I,J)=0.DO
```

Fig. 4.9 Modification of DO statement by using parameter in subroutine GR1DOO of PART1

```
    PARAMETER(NPAR=6000)
    PARAMETER(NWPAR=NPAR*3)
    PARAMETER(NBOXES=5000)
```

Fig. 4.10 Added PARAMETER statements for PART2

```
TB(4000,3), IBND(4000), IBOX(6,4096), X(4000,3), VX(4000,3),
ZZ1(4000), ZZ2(4000), ZZ3(4000), IIBND(4000), ICOV(4000)

          ↓

TB(NPAR,3), IBND(NPAR), IBOX(6,NBOXES), X(NPAR,3), VX(NPAR,3),
ZZ1(NPAR), ZZ2(NPAR), ZZ3(NPAR), IIBND(NPAR), ICOV(NPAR)
```

Fig. 4.11 Declaration of some arrays by using parameters of PART2

```
                .
                .
COMMON /GEOM2/GEOM2(43377)
                .
EQUIVALENCE  (GEOM2(10),IALP),
.            (GEOM2(11),IBET),
.            (GEOM2(12),IGAM),
.            (GEOM2(13),TB(1,1)),
.            (GEOM2(12013),IBND(1)),
.            (GEOM2(16013),IK(1)),
.            (GEOM2(18748),IAUX(1)),
.            (GEOM2(18775),IDUX(1)),
.            (GEOM2(18802),IBOX(1))

             ↓

                .
                .
COMMON /GEOM2/GEOM2(2801+(NWPAR+NPAR+6*NBOXES))
                .
EQUIVALENCE  (GEOM2(10),IALP),
.            (GEOM2(11),IBET),
.            (GEOM2(12),IGAM),
.            (GEOM2(13),TB(1,1)),
.            (GEOM2(NWPAR+13),IBND(1)),
.            (GEOM2(NWPAR+NPAR+13),IK(1)),
.            (GEOM2(NWPAR+NPAR+2748),IAUX(1)),
.            (GEOM2(NWPAR+NPAR+2775),IDUX(1)),
.            (GEOM2(NWPAR+NPAR+2802),IBOX(1))
```

Fig. 4.12 Declaration of common block GEOM2 and EQUIVALENCE statement by using parameters of PART2

```
      .
      .
COMMON /DYNA/DYNA(42498)
      .
EQUIVALENCE  (DYNA(1),X(1,1)),
.            (DYNA(12001),VX(1,1)),
.            (DYNA(24001),AC(1)),
.            (DYNA(24004),ZZ1(1)),
.            (DYNA(28004),ZZ2(1)),
.            (DYNA(32004),ZZ3(1)),
.            (DYNA(36004),E(1,1)),
.            (DYNA(39007),EE(1,1)),
.            (DYNA(42010),E1(1)),
.            (DYNA(42013),E2(1)),
.            (DYNA(42016),A(1,1)),
.            (DYNA(42094),B(1,1)),
.            (DYNA(42172),C(1,1)),
.            (DYNA(42250),A2(1,1)),
.            (DYNA(42328),B2(1,1)),
.            (DYNA(42406),C2(1,1)),
.            (DYNA(42484),ENGKE),
.            (DYNA(42485),ENGL),
.            (DYNA(42486),ENGKG),
.            (DYNA(42487),ENGKA),
.            (DYNA(42488),ENGPO),
.            (DYNA(42489),ENGSP),
.            (DYNA(42490),ENGLO),
.            (DYNA(42491),ENGTO),
.            (DYNA(42492),DISCR),
.            (DYNA(42493),EGKA1),
.            (DYNA(42494),EGPO1),
.            (DYNA(42495),EGSP1),
.            (DYNA(42496),EGLO1),
.            (DYNA(42497),EGTO1),
.            (DYNA(42498),DSCR1)
                    ↓
```

Fig. 4.13 Declaration of common block DYNA and EQUIVALENCE statement by using parameters of PART2 (1/2)

```
                    .
                    .
      COMMON /DYNA/DYNA(6498+(2*NWPAR+3*NPAR))
                    .
      EQUIVALENCE  (DYNA(1),X(1,1)),
     .             (DYNA(NWPAR+1),VX(1,1)),
     .             (DYNA(2*NWPAR+1),AC(1)),
     .             (DYNA(2*NWPAR+4),ZZ1(1)),
     .             (DYNA(2*NWPAR+NPAR+4),ZZ2(1)),
     .             (DYNA(2*NWPAR+2*NPAR+4),ZZ3(1)),
     .             (DYNA(2*NWPAR+3*NPAR+4),E(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+3007),EE(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6010),E1(1)),
     .             (DYNA(2*NWPAR+3*NPAR+6013),E2(1)),
     .             (DYNA(2*NWPAR+3*NPAR+6016),A(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6094),B(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6172),C(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6250),A2(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6328),B2(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6406),C2(1,1)),
     .             (DYNA(2*NWPAR+3*NPAR+6484),ENGKE),
     .             (DYNA(2*NWPAR+3*NPAR+6485),ENGL),
     .             (DYNA(2*NWPAR+3*NPAR+6486),ENGKG),
     .             (DYNA(2*NWPAR+3*NPAR+6487),ENGKA),
     .             (DYNA(2*NWPAR+3*NPAR+6488),ENGPO),
     .             (DYNA(2*NWPAR+3*NPAR+6489),ENGSP),
     .             (DYNA(2*NWPAR+3*NPAR+6490),ENGLO),
     .             (DYNA(2*NWPAR+3*NPAR+6491),ENGTO),
     .             (DYNA(2*NWPAR+3*NPAR+6492),DISCR),
     .             (DYNA(2*NWPAR+3*NPAR+6493),EGKA1),
     .             (DYNA(2*NWPAR+3*NPAR+6494),EGPO1),
     .             (DYNA(2*NWPAR+3*NPAR+6495),EGSP1),
     .             (DYNA(2*NWPAR+3*NPAR+6496),EGLO1),
     .             (DYNA(2*NWPAR+3*NPAR+6497),EGTO1),
     .             (DYNA(2*NWPAR+3*NPAR+6498),DSCR1)
```

Fig. 4.13 Declaration of common block DYNA and EQUIVALENCE statement by using parameters of PART2 (2/2)

```
WORK(8006)
      EQUIVALENCE  (WORK(7),IIBND(1)),
     .             (WORK(4007),ICOV(1))

                ↓

WORK(8006)  →  WORK(8006-8000+(2*NPAR))  →  WORK(6+2*NPAR)
      EQUIVALENCE  (WORK(7),IIBND(1)),
     .             (WORK(NPAR+7),ICOV(1))
```

Fig. 4.14 Declaration of common block WORK and EQUIVALENCE statement by using parameters of PART2

```
              .
              .
              .
C.VP26DO 2100 I=1,4096
C.VPP
      DO 2100 I=1,NBOXES
      DO 2100 J=1,6
 2100 IBOX(J,I)=0
      ENGL=0.DO
      ENGKG=0.DO
      ENGPO=0.DO
      ENGSP=0.DO
C.VP26DO 2200 I=1,4000
C.VPP
      DO 2200 I=1,NPAR
 2200 IIBND(I)=0
              .
              .
              .
C.VP26DO 2400 I=1,4000
C.VPP
      DO 2400 I=1,NPAR
      ZZ1(I)=0.DO
      ZZ2(I)=0.DO
 2400 ZZ3(I)=0.DO
              .
              .
              .
```

Fig. 4.15 Modification of some DO statements by using parameters in subroutine GR2D08 of PART2

```
C.VP26IF(I3.GE.4097)          GO TO 7000
C.VPP
      IF(I3.GE.NBOXES+1)      GO TO 7000
```

Fig. 4.16 Modification of IF statement by using parameter in subroutine GR2D41 of PART2

・追加したパラメータ文

```
PARAMETER(NPAR=6000)
```

・パラメータ変数を利用した配列の宣言

```
サブルーチン GR3D00 :
    XO(5000,3), XN(5000,3), XYPO(5000,3), XYP1(5000,3), IXO(5000)
                    ↓
    XO(NPAR,3), XN(NPAR,3), XYPO(NPAR,3), XYP1(NPAR,3), IXO(NPAR)
サブルーチン XYPOUT :
    XYPO(5000,3), XYP1(5000,3)
                    ↓
    XYPO(NPAR,3), XYP1(NPAR,3)
```

Fig. 4.17 Added PARAMETER statement and declaration of some arrays by using parameter of PART3

```
 1:                .
 2:                .
 3:                .
 4:                .
 5:       DO 1000 I=1,N
 6:       DO 1000 J=1,3
 7:  C                                          NEW POSITION
 8:       X(I,J)=(VX(I,J)+X(I,J))*0.5D0
 9:  C                                          KINETIC ENERGY
10:   1000 ENGKG=ENGKG+(VX(I,J)-X(I,J))**2
11:  C
12:       ENGKG=1000.D0*ENGKG/DDT
13:  C                                          POTENTIAL ENERGY       1
14:       ENGPO=2000.D0*ENGPO
15:  C                                          PRESSURE,SPRING ENERGY 2
16:       ENGSP=2000.D0*ENGSP
17:  C                                          AVERAGE KINETIC ENERGY
18:       ENGKA=0.5D0*(ENGKG+ENGKE)
19:  C                                          ENERGY LOSS(DISSIPATION)
20:       ENGLO=ENGLO+2000.D0*ENGL
21:  C                                          INITIAL PROCESS ?
22:       IF(LSLT(2).EQ.0)        GO TO 1500
23:  C                                          INITIAL TOTAL ENERGY
24:       ENGTO=ENGPO+ENGSP+ENGKA
25:       LSLT(2)=0
26:  C                                          NEW TOTAL ENERGY
27:   1500 ENGTN=ENGPO+ENGSP+ENGKA+ENGLO
28:  C                                          DISCREPANCY
29:       DISCR=ENGTO-ENGTN
30:  C                                          TOTAL ENERGY
31:       ENGTO=ENGTN
32:  C                                          NEW TIME
33:       T=T+DT
34:                .
35:                .
36:                .
37:                .
38:                .
```

Fig. 4.18 Part of original subroutine GR2D50 of PART2

```
 1:                .
 2:                .
 3:                .
 4:       I11=I1-I4
 5:       IF(I11.EQ.0)            GO TO 3000
 6:       IF(I11.GT.0)            GO TO 2400
 7: C                                      PHI3(RP1)--->PHI3(RPN)
 8:       E11=E1(3)*WRK+E2(3)
 9:       I5=E11
10:       H=E11-I5
11: C
12:       ACC=E(I5,3)+H*(E(I5+1,3)-E(I5,3))
13:       AC(1)=AC(1)+ACC*XW/WRK
14:       AC(2)=AC(2)+ACC*YW/WRK
15:       AC(3)=AC(3)+ACC*ZW/WRK
16:       ZZ1(I4)=ZZ1(I4)-ACC*XW/WRK
17:       ZZ2(I4)=ZZ2(I4)-ACC*YW/WRK
18:       ZZ3(I4)=ZZ3(I4)-ACC*ZW/WRK
19: C
20:       ENGP0=ENGP0+EE(I5,3)+H*(EE(I5+1,3)-EE(I5,3))
21:
22:                .
23:                .
24:                .
25:                .
26:                .
27:                .
28:                .
29:                .
30:                .
31:
32:       DO 8010 I=1,4
33:  8010 IF(RNF(I).NE.0.0D0) RNF(I)=1.0D0
34:       ENGP0=ENGP0+(
35:      * RNF(1)*     PHI1(1)*PHI2(1)*PHI2(2)*PHI2(3)
36:      *+RNF(2)*    PHI1(2)*PHI2(1)*PHI2(4)*PHI2(6)
37:      *+RNF(3)*    PHI1(3)*PHI2(2)*PHI2(4)*PHI2(5)
38:      *+RNF(4)*    PHI1(4)*PHI2(3)*PHI2(5)*PHI2(6))/2.D0
39:       RETURN
40:       END
```

Fig. 4.19 Part of original subroutine GR2D41 of PART2

```
                    .
                    .
                    .
      WRITE(6,6003) RMIN,RMAX,
     .              ( A1(I),B1(I) ,I=1,3)
C.VPP
      REWIND IUNT18
      WRITE(IUNT18) ((RMIN(I),RMAX(I)),I=1,3)
                    .
                    .
                    .
```

Fig. 4.20 Modification of original subroutine GR1DIO of PART1

```
                    .
                    .
C.VPP
     .              (DYNA(2*NWPAR+3*NPAR+6499),RMIN(1)),
     .              (DYNA(2*NWPAR+3*NPAR+6502),RMAX(1))
                    .
                    .
C.VPP
      read(IUNT18)  ((RMIN(I),RMAX(I)),I=1,3)
                    .
                    .
```

Fig. 4.21 Modification of original subroutine GR2D10 of PART2

```
                    .
                    .
C.VPP
     .              (DYNA(2*NWPAR+3*NPAR+6499),RMIN(1)),
     .              (DYNA(2*NWPAR+3*NPAR+6502),RMAX(1))
                    .
                    .
                    .
                    .
                    .
                    .
      XW=X(I4,1)-X(I1,1)
      YW=X(I4,2)-X(I1,2)
      ZW=X(I4,3)-X(I1,3)
      WRK2=XW**2+YW**2+ZW**2
      WRK= SQRT(WRK2)
C.VPP
      if(WRK.gt.RMAX(1)) then
      if(ICOMB.eq.1) then
        ICOMB=0
        NAL=1
      endif
      go to 3000
      endif
                    .
                    .
                    .
```

Fig. 4.22 Modification of original subroutine GR2D41 of PART2

```
                    .
                    .
      REWIND IUNT18
C.VPP
      READ(IUNT18) ((RMIN(I),RMAX(I)),I=1,3)

      READ(IUNT18) T1,IL,IL,IL,IL,N
                    .
                    .
```

Fig. 4.23 Added dummy READ statement in original subroutine GR3D00 of PART3

```
                .
                .
                .
      IF(I2FLG.EQ.1) THEN
*VOCL LOOP,NOVREC
      DO 4100 I1 = 1, N
         I2 = I2$(I1)
C                                        TEST THE CONTENTS OF BOX
      IF(IBOX(6,I2).NE.0)      GO TO  800
      IBOX(6,I2)=I1
                               GO TO 3000
  800 IF(IBOX(5,I2).NE.0)      GO TO  900
      IBOX(5,I2)=I1
                               GO TO 3000
  900 IF(IBOX(4,I2).NE.0)      GO TO 1000
C                                        EMPTY ---> THE FIRST PART
      IBOX(4,I2)=I1
                               GO TO 3000
 1000 CONTINUE
      IF(IBOX(3,I2).NE.0)      GO TO 1500
C                                        ONLY PARTICLE 1 ---> SECOND PART
      IBOX(3,I2)=I1
                               GO TO 3000
 1500 CONTINUE
      IF(IBOX(2,I2).NE.0)      GO TO 1600
C                                        PARTICLE 1,2 ---> THIRD PART
      IBOX(2,I2)=I1
                               GO TO 3000
C
 1600 CONTINUE
      IF(IBOX(1,I2).NE.0)      GO TO 8100
C                                        PARTICLE 1,2,3---> FOURTH PART
      IBOX(1,I2)=I1
C
 3000 IIBND(I1)=I2
C4000 CONTINUE
 4100 CONTINUE
                .
                .
                .
```

Fig. 4.24 Part of old vectorized subroutine GR2D3 of PART2

```
 1:                    .
 2:                    .
 3:                    .
 4:                    .
 5:                    .
 6:         ICOMB=0
 7:         NAL=1
 8:         DO 10 K=1,KK
 9:         IF(I1.EQ.III1(K)) GO TO 11
10:         IF(I1.EQ.III4(K)) GO TO 12
11:      10 CONTINUE
12:         GO TO 1000
13:      11 I4=III4(K)
14:         ICOMB=1
15:         NAL=2
16:         GO TO 2001
17:      12 I4=III1(K)
18:         ICOMB=1
19:         NAL=2
20:         GO TO 2001
21:    1000 CONTINUE
22:         ICOMB=0
23:         I3=J1+IK(I2)
24:         IF(I3.LE.0)                GO TO 7000
25:         IF(I3.GE.4097)             GO TO 7000
26:         DO 1500 I=1,6
27:    1500 IBOXW(I)=IBOX(I,I3)
28:    2000 CONTINUE
29:         I4=IBOXW(6)
30:         IF(I4.EQ.0)                GO TO 7000
31:         IF(ICOMB.EQ.1) GO TO 2001
32:         DO 100 K=1,KK
33:         IF(I1.EQ.III1(K).AND.I4.EQ.III4(K)) GO TO 3000
34:         IF(I1.EQ.III4(K).AND.I4.EQ.III1(K)) GO TO 3000
35:       100 CONTINUE
36:    2001 CONTINUE
37:         XW=X(I4,1)-X(I1,1)
38:         YW=X(I4,2)-X(I1,2)
39:         ZW=X(I4,3)-X(I1,3)
40:         WRK2=XW**2+YW**2+ZW**2
41:         WRK= SQRT(WRK2)
42:         IF(WRK.GT.RMAX(1)) THEN
43:           IF(ICOMB.EQ.1) THEN
44:             ICOMB=0
45:             NAL=1
46:           ENDIF
47:           GO TO 3000
48:         ENDIF
49:         I11=I1-I4
50:         IF(I11.EQ.0)               GO TO 3000
51:         IF(I11.GT.0)               GO TO 2400
```

Fig. 4.25 Part of modified subroutine GR2D41 of PART2 (1/2)

```
 52:          E11=E1(3)*WRK+E2(3)
 53:          I5=E11
 54:          H=E11-I5
 55:          ACC=E(I5,3)+H*(E(I5+1,3)-E(I5,3))
 56:          AC(1)=AC(1)+ACC*XW/WRK
 57:          AC(2)=AC(2)+ACC*YW/WRK
 58:          AC(3)=AC(3)+ACC*ZW/WRK
 59:          ZZ1(I4)=ZZ1(I4)-ACC*XW/WRK
 60:          ZZ2(I4)=ZZ2(I4)-ACC*YW/WRK
 61:          ZZ3(I4)=ZZ3(I4)-ACC*ZW/WRK
 62:          ENGPO=ENGPO+EE(I5,3)+H*(EE(I5+1,3)-EE(I5,3))
 63:    2400  IF(ICOV(I4).GE.4)          GO TO 3000
 64:          IF(NVAL.EQ.0)              GO TO 3000
 65:          NVAL1=NAL-ICOMB
 66:          DO 2440 I=NVAL1,NVAL
 67:          IF(RNF(I).EQ.0.0D0.OR.RNF(I).GT.WRK) GO TO 2471
 68:    2440  CONTINUE
 69:          GO TO 3000
 70:    2471  J=NVAL
 71:    2472  IF(J.EQ.I) GO TO 2473
 72:          RNF(J)=RNF(J-1)
 73:          R(1,J)=R(1,J-1)
 74:          R(2,J)=R(2,J-1)
 75:          R(3,J)=R(3,J-1)
 76:          NF(J)=NF(J-1)
 77:          J=J-1
 78:          GO TO 2472
 79:    2473  RNF(I)=WRK
 80:          R(1,I)=XW/WRK
 81:          R(2,I)=YW/WRK
 82:          R(3,I)=ZW/WRK
 83:          NF(I)=I4
 84:          IF(ICOMB.EQ.1) GO TO 1000
 85:    3000  CONTINUE
 86:          IBOXW(6)=IBOXW(5)
 87:          IBOXW(5)=IBOXW(4)
 88:          IBOXW(4)=IBOXW(3)
 89:          IBOXW(3)=IBOXW(2)
 90:          IBOXW(2)=IBOXW(1)
 91:          IBOXW(1)=0
 92:                                     GO TO 2000
 93:    7000  CONTINUE
 94:          I2=I2+1
 95:          IF(I2.GT.IC)               GO TO 7500
 96:                                     GO TO 1000
 97:    7500  CONTINUE
 98:          DO 7520 I=1,4
 99:          IF(NF(I).NE.0) ICOV(NF(I))=ICOV(NF(I))+1
100:    7520  CONTINUE
101:                    .
102:                    .
103:                    .
104:                    .
105:                    .
```

Fig. 4.25 Part of modified subroutine GR2D41 of PART2 (2/2)

```
      DO 9000 I1=1,N
       (初期値設定)
9000  CONTINUE

      IF(ISW.EQ.1) THEN  (１回だけ実行)
      DO 9001 I1=1,N
       (結合手数の定義, 箱検索ポインタ定義)
9001  CONTINUE
      ENDIF

      IF(I2FLG.EQ.1) THEN  (原子が前に属していた箱からずれた場合だけ実行)
      DO 9002 I1=1,N
       (各原子が属する箱番号取得)
9002  CONTINUE
      DO 5000 I1=1,N
       (フラグ設定)
5000  CONTINUE
      DO 7001 K=1,KK
      DO 7000 I1=1,N
       (強制的に結合させる原子が指定されている場合だけ実行)
7000  CONTINUE
7001  CONTINUE
      DO 8000 I1=1,N
       (箱検索処理)
8000  CONTINUE
      ENDIF

      DO 600 I1=1,N
       (検索主原子と被検索原子の距離を計算)
 600  CONTINUE

      DO 800 I1=1,N
       (中心力についての計算)
 800  CONTINUE

      DO 700 I1=1,N
       (４つの最近接原子の決定)
 700  CONTINUE

      DO 9003 I1=1,N
       (非中心力についての計算)  ← 既にベクトル化済
9003  CONTINUE

      RETURN
      END
```

Fig. 4.26 Outline of new vectorized subroutine GR2D41 of PART2

```
                .
                .
          DIMENSION  NF(NPAR,4),RNF(NPAR,4),THETA(NPAR,6),R(NPAR,3,4)
                .
                .
v         DO 9000 I1=1,N
v         NF(I1,1)=0
v         NF(I1,2)=0
v         NF(I1,3)=0
v         NF(I1,4)=0
v         RNF(I1,1)=0.D0
v         RNF(I1,2)=0.D0
v         RNF(I1,3)=0.D0
v         RNF(I1,4)=0.D0
v         R(I1,1,1)=0.D0
v         R(I1,2,1)=0.D0
v         R(I1,3,1)=0.D0
v         R(I1,1,2)=0.D0
v         R(I1,2,2)=0.D0
v         R(I1,3,2)=0.D0
v         R(I1,1,3)=0.D0
v         R(I1,2,3)=0.D0
v         R(I1,3,3)=0.D0
v         R(I1,1,4)=0.D0
v         R(I1,2,4)=0.D0
v         R(I1,3,4)=0.D0
v         THETA(I1,1)=0.D0
v         THETA(I1,2)=0.D0
v         THETA(I1,3)=0.D0
v         THETA(I1,4)=0.D0
v         THETA(I1,5)=0.D0
v         THETA(I1,6)=0.D0
v    9000 CONTINUE
```

Fig. 4.27 DO 9000 loop in new vectorized subroutine GR2D41

```
                .
                .
          DIMENSION  NVAL_V(NPAR),I2_V(NPAR),IC_V(NPAR)
                .
                .
          IF(ISW.EQ.1) THEN
v         DO 9001 I1=1,N
v         IO=IBND(I1)
v         IF(IBND(I1).LT.0) IO=0
v         NVAL_V(I1)=4
v         IF(IO.GE.1.AND.IO.LE.6) NVAL_V(I1)=2
v         IF(IO.GE.7.AND.IO.LE.18) NVAL_V(I1)=1
v         IF(IO.EQ.19) NVAL_V(I1)=1
v         IF(IO.GE.20.AND.IO.LE.22) NVAL_V(I1)=0
v         IF(IO.GE.23.AND.IO.LE.25) NVAL_V(I1)=1
v         IF(IO.EQ.26) NVAL_V(I1)=0
v         I2_V(I1)=IAUX(IO+1)
v         IC_V(I1)=IAUX(IO+1)+IDUX(IO+1)-1
v    9001 CONTINUE
          ISW=0
          ENDIF
```

Fig. 4.28 DO 9001 loop in new vectorized subroutine GR2D41

```
                .
                .
          DIMENSION  J1_V(NPAR),ICOMB_V(NPAR,27*6),NAL_V(NPAR,27*6),
                     SEARCH_NUM(NPAR)
                .
                .
v         DO 9002 I1=1,N
v         J1_V(I1)=IIBND(I1)
v         IIBND(I1)=0
v    9002 CONTINUE
v         DO 5000 I1=1,N
v2          DO 5001 I1A=1,27*6
v2          ICOMB_V(I1,I1A)=0
v2          NAL_V(I1,I1A)=1
v2   5001    CONTINUE
v           ISEARCH_NUM(I1)=0
v    5000 CONTINUE
```

Fig. 4.29 DO 9002 loop in new vectorized subroutine GR2D41

```
                .
                .
          DIMENSION I4_LIST(NPAR,27*6)
                .
                .
s         DO 7001 K=1,KK
v         DO 7000 I1=1,N
v           IF(I1.EQ.III1(K)) THEN
v             ISEARCH_NUM(I1)=ISEARCH_NUM(I1)+1
v             I4_LIST(I1,ISEARCH_NUM(I1))=III4(K)
v             GO TO 7002
v           ENDIF
v           IF(I1.EQ.III4(K)) THEN
v             ISEARCH_NUM(I1)=ISEARCH_NUM(I1)+1
v             I4_LIST(I1,ISEARCH_NUM(I1))=III1(K)
v             GO TO 7002
v           ENDIF
v           GO TO 7000
v    7002   ICOMB_V(I1,ISEARCH_NUM(I1))=1
v           NAL_V(I1,ISEARCH_NUM(I1))=2
v    7000 CONTINUE
s    7001 CONTINUE
```

Fig. 4.30 DO 7001 loop in new vectorized subroutine GR2D41

```
s         DO 8000 I1=1,N
s           K1=ISEARCH_NUM(I1)+1
s           DO 8001 I2=1,IC_V(I1)-I2_V(I1)+1
s             IHAKO=J1_V(I1)+IK(I2_V(I1)+I2-1)
s             IF(IHAKO.GT.0.AND.IHAKO.LT.NBOXES+1) THEN
                DO 8002 I3=1,6
                  I4=IBOX(I3,IHAKO)
                  IF(I4.NE.0) THEN
                    IF(ICOMB_V(I1,1).EQ.1.AND.ICOMB_V(I1,K1).NE.1) THEN
s8                    DO 100 K=1,KK
s8                      IF(I1.EQ.III1(K).AND.I4.EQ.III4(K)) GO TO 8002
s8                      IF(I1.EQ.III4(K).AND.I4.EQ.III1(K)) GO TO 8002
s8     100              CONTINUE
                    ENDIF
                    I11=I1-I4
                    IF(I11.NE.0) THEN
                      I4_LIST(I1,K1)=I4
                      K1=K1+1
                    ENDIF
                  ENDIF
       8002       CONTINUE
s             ENDIF
s      8001   CONTINUE
s           ISEARCH_NUM(I1)=K1-1
s      8000 CONTINUE
```

Fig. 4.31 DO 8000 loop in new vectorized subroutine GR2D41

```
v         DO 600 I1=1,N
v         K1=0
v         IF(ICOMB_V(I1,1).EQ.1) THEN
v           XW_S=X(I4_LIST(I1,1),1)-X(I1,1)
v           YW_S=X(I4_LIST(I1,1),2)-X(I1,2)
v           ZW_S=X(I4_LIST(I1,1),3)-X(I1,3)
v           WRK2=XW_S**2+YW_S**2+ZW_S**2
v           WRK_S=SQRT(WRK2)
v           IF(WRK_S.GT.RMAX(1)) ICOMB_V(I1,1)=0
v         ENDIF
v         DO 601 I22=1,ISEARCH_NUM(I1)
v           XW_S=X(I4_LIST(I1,I22),1)-X(I1,1)
v           YW_S=X(I4_LIST(I1,I22),2)-X(I1,2)
v           ZW_S=X(I4_LIST(I1,I22),3)-X(I1,3)
v           WRK2=XW_S**2+YW_S**2+ZW_S**2
v           WRK_S=SQRT(WRK2)
v           I11=I1-I4_LIST(I1,I22)
v           IF(WRK_S.LE.RMAX(1).AND.I11.NE.0) THEN
v             K1=K1+1
v             XW(I1,K1)=XW_S
v             YW(I1,K1)=YW_S
v             ZW(I1,K1)=ZW_S
v             WRK(I1,K1)=WRK_S
v             I4_LIST2(I1,K1)=I4_LIST(I1,I22)
v           ENDIF
v     601 CONTINUE
v         I_MAIN(I1)=K1
v     600 CONTINUE
```

Fig. 4.32 DO 600 loop in new vectorized subroutine GR2D41

```
s         DO 800 I1=1,N
s           AC(I1,1)=ZZ1(I1)
s           AC(I1,2)=ZZ2(I1)
s           AC(I1,3)=ZZ3(I1)
v   *VOCL LOOP,NOVREC
v           DO 801 I22=1,I_MAIN(I1)
v             I11=I1-I4_LIST2(I1,I22)
v             IF(I11.LT.0) THEN
v               E11=E1(3)*WRK(I1,I22)+E2(3)
v               I5=E11
v               H=E11-I5
v               ACC=E(I5,3)+H*(E(I5+1,3)-E(I5,3))
v               AC(I1,1)=AC(I1,1)+ACC*XW(I1,I22)/WRK(I1,I22)
v               AC(I1,2)=AC(I1,2)+ACC*YW(I1,I22)/WRK(I1,I22)
v               AC(I1,3)=AC(I1,3)+ACC*ZW(I1,I22)/WRK(I1,I22)
v               ZZ1(I4_LIST2(I1,I22))=ZZ1(I4_LIST2(I1,I22))-ACC*
v        &XW(I1,I22)/WRK(I1,I22)
v               ZZ2(I4_LIST2(I1,I22))=ZZ2(I4_LIST2(I1,I22))-ACC*
v        &YW(I1,I22)/WRK(I1,I22)
v               ZZ3(I4_LIST2(I1,I22))=ZZ3(I4_LIST2(I1,I22))-ACC*
v        &ZW(I1,I22)/WRK(I1,I22)
v               ENGPO=ENGPO+EE(I5,3)+H*(EE(I5+1,3)-EE(I5,3))
v             ENDIF
v     801   CONTINUE
s     800 CONTINUE
```

Fig. 4.33 DO 800 loop in new vectorized subroutine GR2D41

```
s         DO 700 I1=1,N
s         ISTART=1
v         IF(ICOMB_V(I1,1).EQ.1) then
m           IF(ICOV(I4_LIST2(I1,1)).LT.4) THEN
s             ISTART=2
v           ENDIF
v         ENDIF
s         DO 701 IBOND=ISTART,NVAL_V(I1)
v         IMIN=27*6+1
s         WRK(I1,IMIN)=RMAX(1)
v         WRK_L=WRK(I1,IMIN)
v         DO 702 I22=IBOND,I_MAIN(I1)
v           IF(ICOV(I4_LIST2(I1,I22)).LT.4) THEN
v             WRK_S=WRK(I1,I22)
v             IF(WRK_S.LT.WRK_L) THEN
v               WRK_L=WRK_S
v               IMIN=I22
v             ENDIF
v           ENDIF
v     702 CONTINUE
m         IF(IBOND.NE.IMIN.AND.IMIN.NE.27*6+1) THEN
s           W_WRK1=WRK(I1,IBOND)
s           W_WRK2=XW(I1,IBOND)
s           W_WRK3=YW(I1,IBOND)
s           W_WRK4=ZW(I1,IBOND)
s           W_WRK5=I4_LIST2(I1,IBOND)
s           WRK(I1,IBOND)=WRK(I1,IMIN)
s           XW(I1,IBOND)=XW(I1,IMIN)
s           YW(I1,IBOND)=YW(I1,IMIN)
s           ZW(I1,IBOND)=ZW(I1,IMIN)
s           I4_LIST2(I1,IBOND)=I4_LIST2(I1,IMIN)
s           WRK(I1,IMIN)=W_WRK1
s           XW(I1,IMIN)=W_WRK2
s           YW(I1,IMIN)=W_WRK3
s           ZW(I1,IMIN)=W_WRK4
s           I4_LIST2(I1,IMIN)=W_WRK5
v         ENDIF
v     701 CONTINUE
 2        DO 703 II22=1,NVAL_V(I1)
 2          RNF(I1,II22)=WRK(I1,II22)
 2          R(I1,1,II22)=XW(I1,II22)/WRK(I1,II22)
 2          R(I1,2,II22)=YW(I1,II22)/WRK(I1,II22)
 2          R(I1,3,II22)=ZW(I1,II22)/WRK(I1,II22)
 2          NF(I1,II22)=I4_LIST2(I1,II22)
 2          IF(NF(I1,II22).NE.0) ICOV(NF(I1,II22))=ICOV(NF(I1,II22))+1
 2     703 CONTINUE
v      700 CONTINUE
```

Fig. 4.34 DO 700 loop in new vectorized subroutine GR2D41

```
Status                          : Serial
Number of Processors            : 1

Type                            : cpu
Interval (msec)                 : 10

Synthesis Information
    Count|   Percent|    VL| Name
    18303|      75.8|    55|  gr2d41_
     4804|      19.9|     -|  gr2d60_
      933|       3.9|     -|  main
       54|       0.2|     -|  gr2d10_
       41|       0.2|  1266|  gr2d40_
       12|       0.0|  1884|  gr2d50_
        8|       0.0|  2048|  MAIN__
        6|       0.0|     -|  gr2d3_

    24161|          |    67|  TOTAL
```

Fig. 4.35 Dynamic behavior of new vectorized PART2

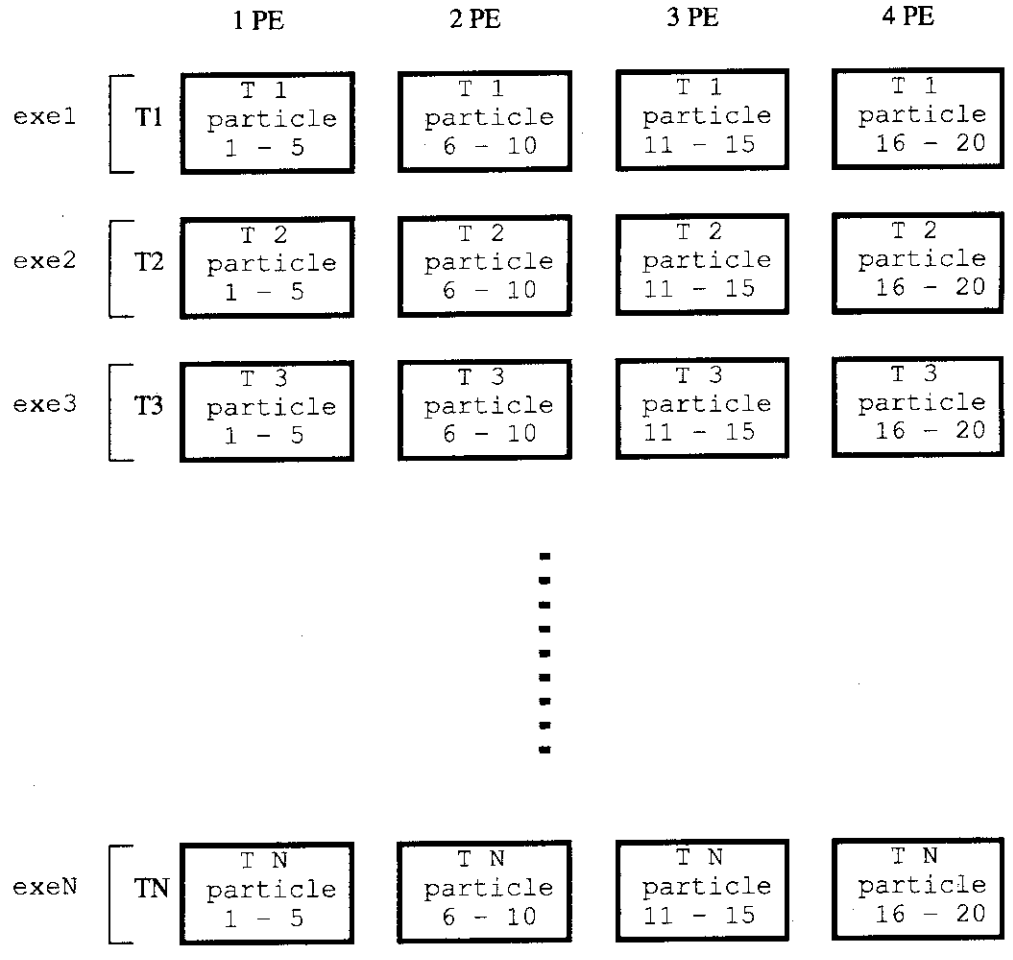


Fig. 4.36 Image of simple parallel processing of an atomic loop

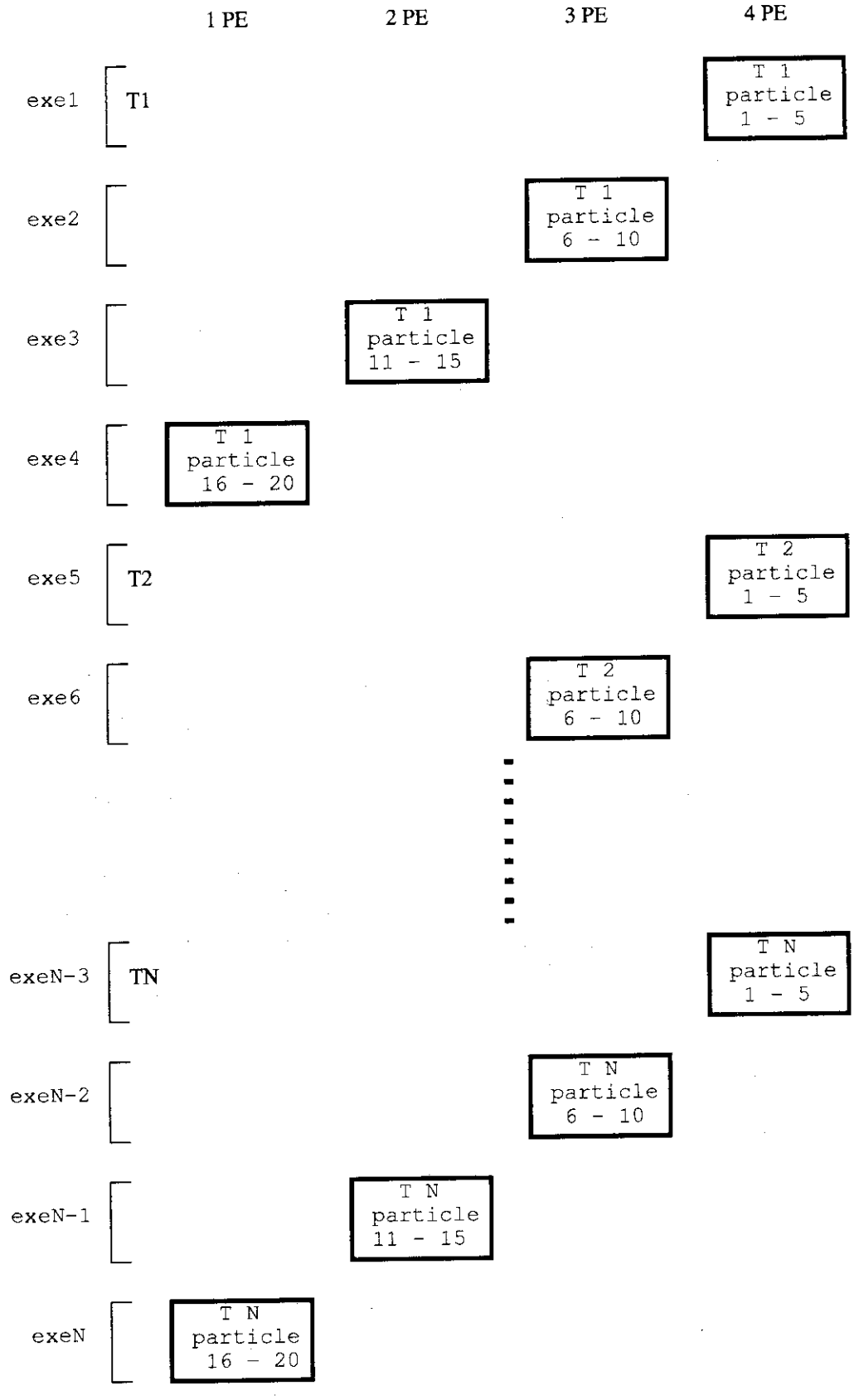


Fig. 4.37 Image of acceptable parallel processing of an atomic loop

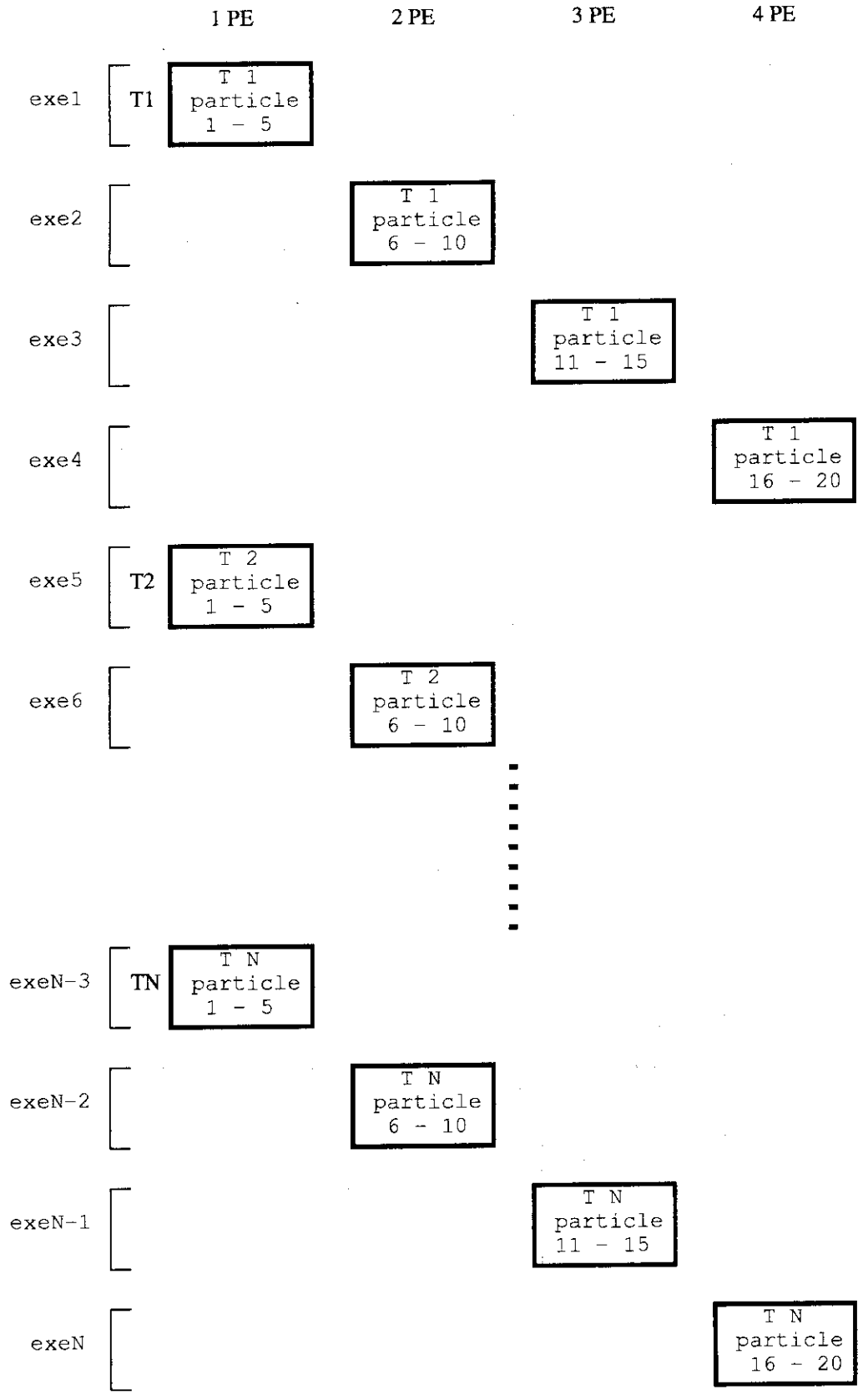


Fig. 4.38 Image of rejectable parallel processing of an atomic loop

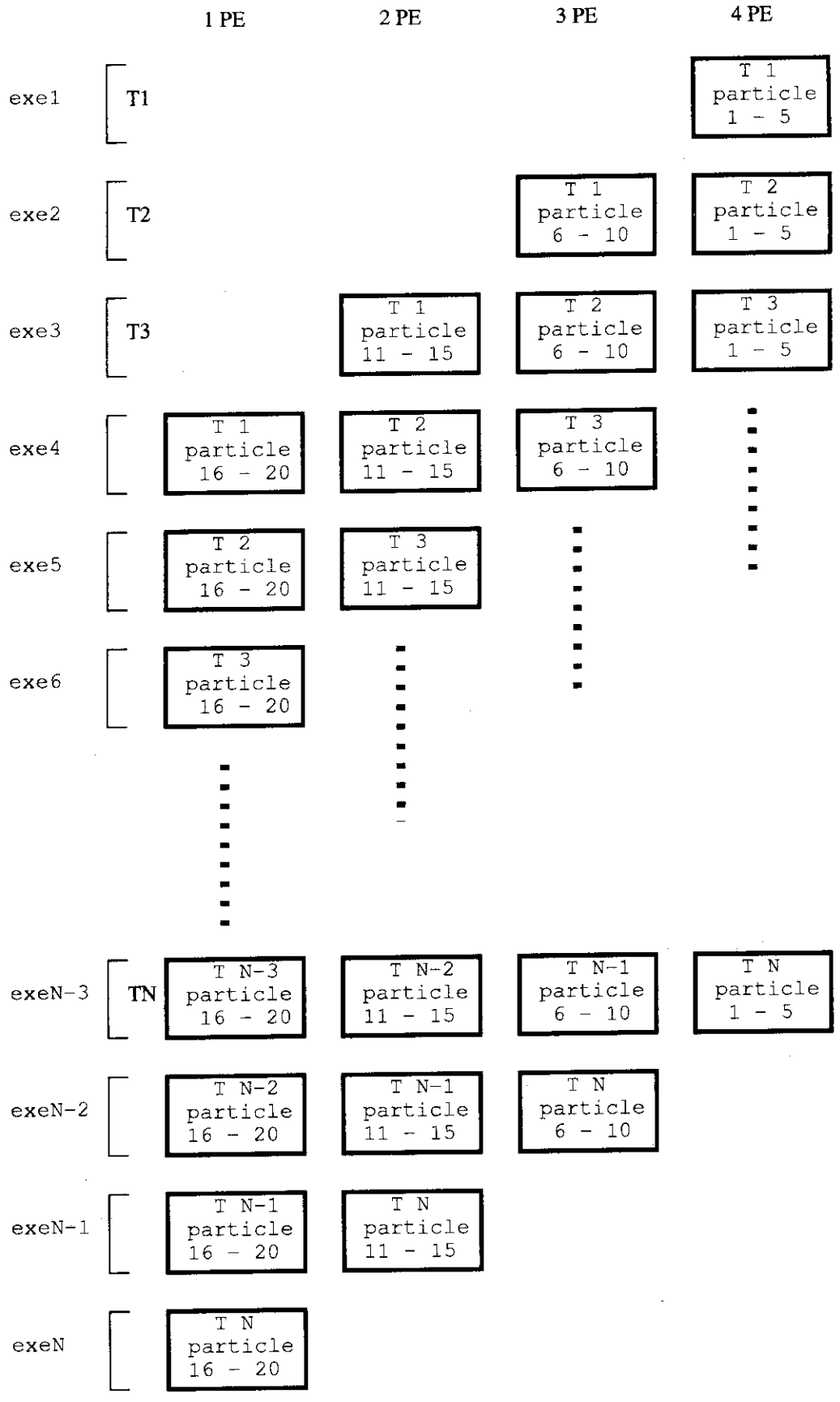


Fig. 4.39 Image of planned parallel processing

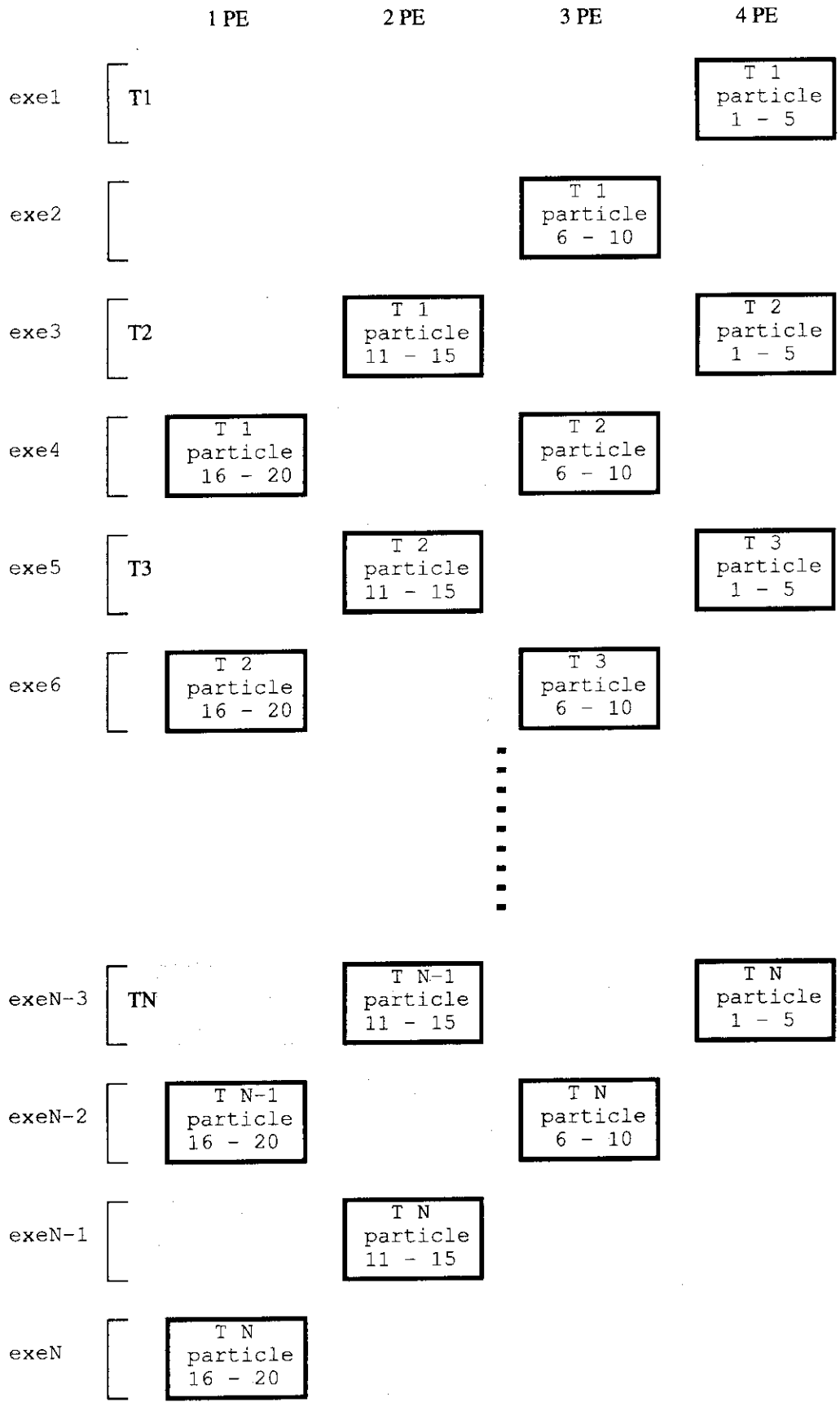


Fig. 4.40 Image of acceptable parallel processing

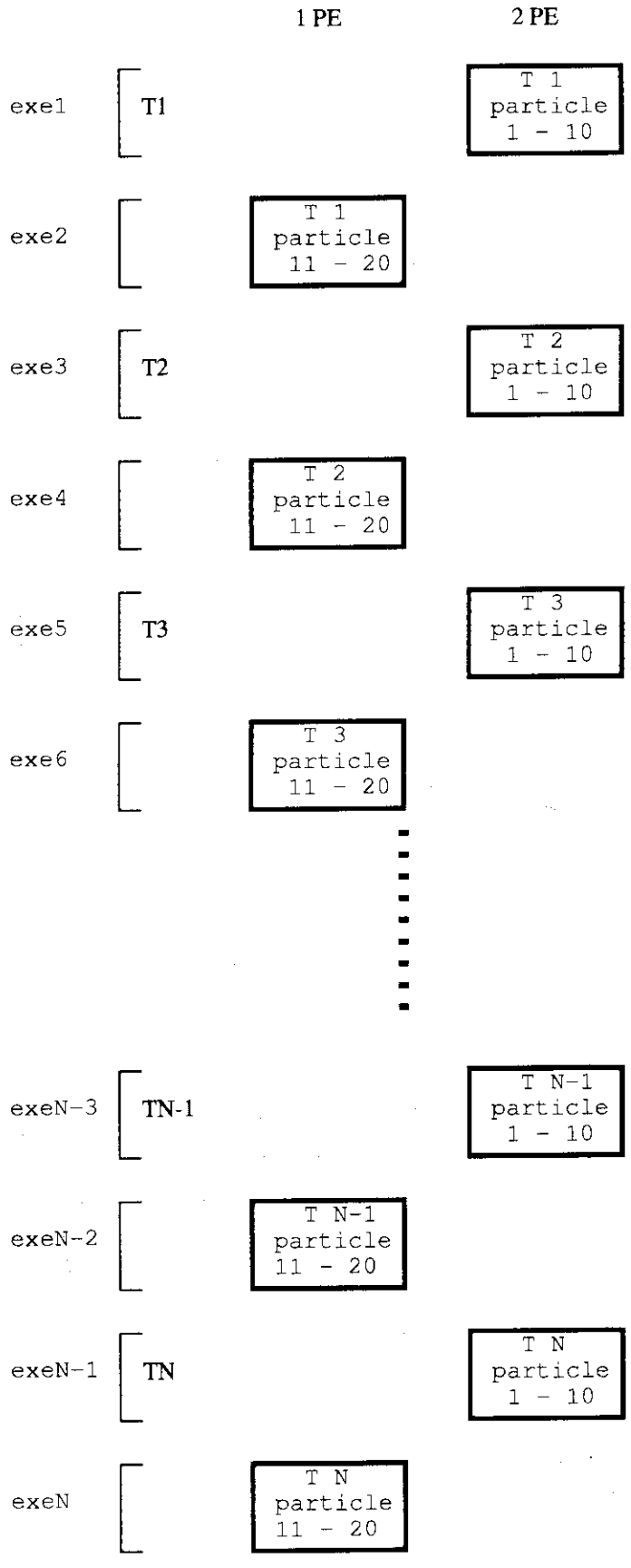


Fig. 4.41 Image of parallel processing by using 2 Processing Elements

```
      DIMENSION X(NPAR,3,NPE-NPE/2,NPE),
     .          VX(NPAR,3,NPE-NPE/2,NPE),
     .          ENGL(NPE-NPE/2,NPE),
     .          ENGKG(NPE-NPE/2,NPE),
     .          ENGPO(NPE-NPE/2,NPE),
     .          ENGSP(NPE-NPE/2,NPE),
     .          ICOV(NPAR,NPE-NPE/2,NPE),
     .          IBOX(6,NBOXES,NPE-NPE/2,NPE),
     .          ZZ1(NPAR,NPE-NPE/2,NPE),
     .          ZZ2(NPAR,NPE-NPE/2,NPE),
     .          ZZ3(NPAR,NPE-NPE/2,NPE),
      COMMON /PARA_ARRAY_G/X_G(NPAR,3,NPE-NPE/2,NPE),
     .                     VX_G(NPAR,3,NPE-NPE/2,NPE),
     .                     ENGL_G(NPE-NPE/2,NPE),
     .                     ENGKG_G(NPE-NPE/2,NPE),
     .                     ENGPO_G(NPE-NPE/2,NPE),
     .                     ENGSP_G(NPE-NPE/2,NPE),
     .                     ICOV_G(NPAR,NPE-NPE/2,NPE),
     .                     IBOX_G(6,NBOXES,NPE-NPE/2,NPE),
     .                     ZZ1_G(NPAR,NPE-NPE/2,NPE),
     .                     ZZ2_G(NPAR,NPE-NPE/2,NPE),
     .                     ZZ3_G(NPAR,NPE-NPE/2,NPE),
!XOCL LOCAL X(:,:,:,/PP),VX(:,:,:,/PP),ENGL(:,/PP)
!XOCL LOCAL ENGKG(:,/PP),ENGPO(:,/PP),ENGSP(:,/PP)
!XOCL LOCAL ICOV(:,:,/PP),IBOX(:,:,:,/PP)
!XOCL LOCAL ZZ1(:,:,/PP),ZZ2(:,:,/PP),ZZ3(:,:,/PP)
!XOCL GLOBAL X_G,VX_G,ENGL_G
!XOCL GLOBAL ENGKG_G,ENGPO_G,ENGSP_G
!XOCL GLOBAL ICOV_G,IBOX_G,ZZ1_G,ZZ2_G,ZZ3_G
      EQUIVALENCE (X,X_G),(VX,VX_G),(ENGL,ENGL_G),
     .            (ENGKG,ENGKG_G),(ENGPO,ENGPO_G),(ENGSP,ENGSP_G),
     .            (ICOV,ICOV_G),(IBOX,IBOX_G),
     .            (ZZ1,ZZ1_G),(ZZ2,ZZ2_G),(ZZ3,ZZ3_G)
```

Fig. 4.42 Declaration of some arrays of PART2 for parallelization

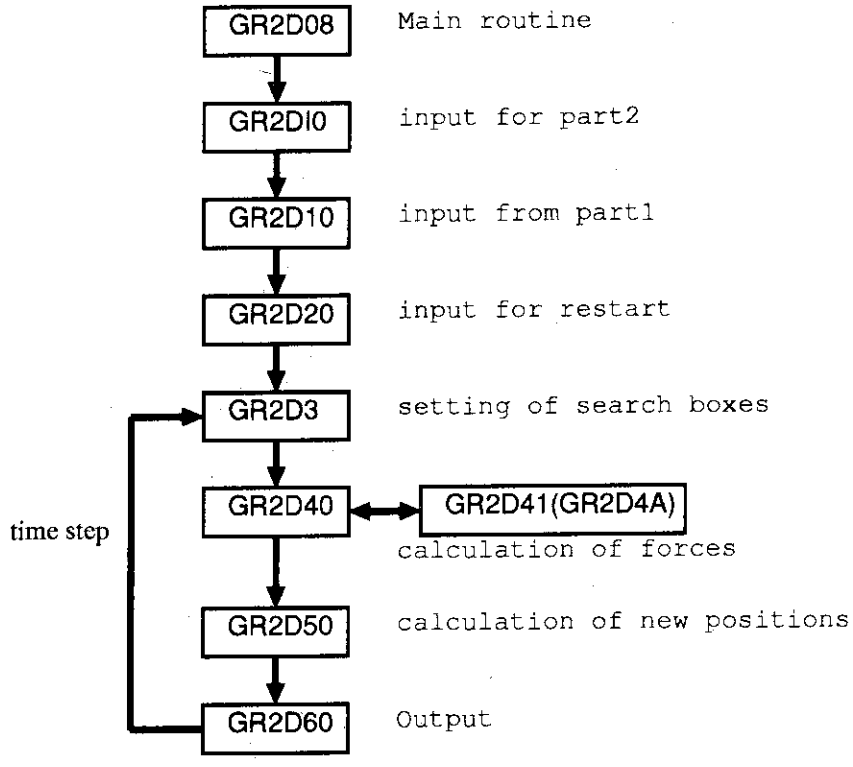


Fig. 4.43 New vectorized flow chart of PART2

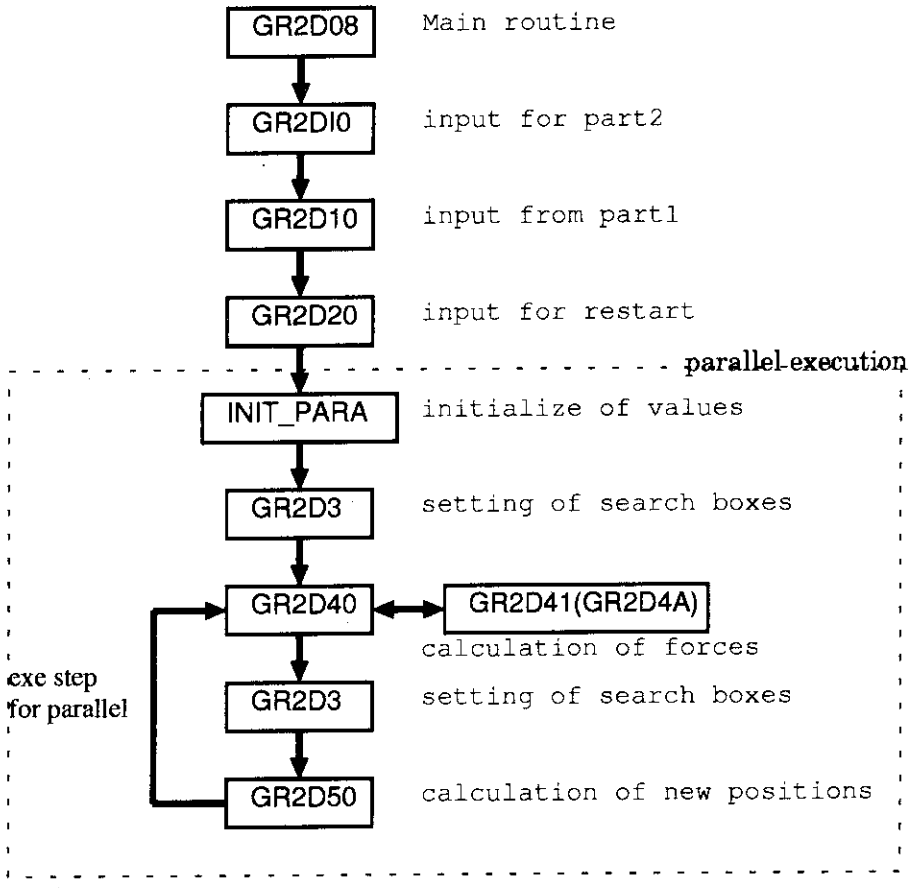


Fig. 4.44 Parallelized flow chart of PART2

```
 1:                    .
 2:                    .
 3:        DO 1000 I=1,N
 4:        DO 1000 J=1,3
 5:        X(I,J)=(VX(I,J)+X(I,J))*0.5D0
 6:   1000 ENGKG=ENGKG+(VX(I,J)-X(I,J))**2
 7:        ENGKG=1000.D0*ENGKG/DDT
 8:        ENGPO=2000.D0*ENGPO
 9:        ENGSP=2000.D0*ENGSP
10:        ENGKA=0.5D0*(ENGKG+ENGKE)
11:        ENGLO=ENGLO+2000.D0*ENGL
12:        IF(LSLT(2).EQ.0)         GO TO 1500
13:        ENGTO=ENGPO+ENGSP+ENGKA
14:        LSLT(2)=0
15:   1500 ENGTN=ENGPO+ENGSP+ENGKA+ENGLO
16:        DISCR=ENGTO-ENGTN
17:        ENGTO=ENGTN
18:        T=T+DT
19:        IQUEN=LSLT(5)
20:        IF(IQUEN.NE.1) GO TO 7000
21:        IF(ENGKG.GE.ENGKE) GO TO 7000
22:        TQ3=TQ2
23:        TQ2=TQ1
24:        TQ1=T
25:        IF(TQ3.EQ.0.0D0) GO TO 9
26:        TQ32=TQ3-TQ2
27:        TQ21=TQ2-TQ1+ 1.0D-7
28:        IF(TQ32.LE.TQ21) GO TO 6999
29:      9 DO 11 I=1,N
30:        DO 11 J=1,3
31:     11 VX(I,J)=X(I,J)
32:        WRITE(6,10) T
33:     10 FORMAT(1H ,'QUENCHED AT T=',F15.6)
34:        GO TO 7000
35:   6999 TQ3=0.0D0
36:        TQ2=0.0D0
37:        TQ1=0.0D0
38:   7000 ENGKE=ENGKG
39:        RETURN
40:        END
```

Fig. 4.45 Part of new vectorized subroutine GR2D50 of PART2

```
                    .
                    .
                    .
      IF(LSLT(1).EQ.1)
     1WRITE(19)      IPROB,MONTH,KAY,IEAR,T,N,
     .              (X(I,3),VX(I,3),X(I,2),VX(I,2),X(I,1),VX(I,1),I=1,N),
     .               DT,ENGKA,ENGPO,ENGSP,ENGLO,ENGTO,DISCR,ENGKE,
     .               EGKA1,EGPO1,EGSP1,EGLO1,EGTO1,DSCR1,INSEN
      IF(LSLT(1).EQ.1) RETURN
      WRITE(IUNT18) IPROB,MONTH,KAY,IEAR,T,N,
     .              (X(I,3),VX(I,3),X(I,2),VX(I,2),X(I,1),VX(I,1),I=1,N),
     .               DT,ENGKA,ENGPO,ENGSP,ENGLO,ENGTO,DISCR,ENGKE,
     .               EGKA1,EGPO1,EGSP1,EGLO1,EGTO1,DSCR1,INSEN
      WRITE(6,5000) T,ENGKA,ENGPO,ENGSP,ENGLO,ENGTO,DISCR
 5000 FORMAT(1H ,7F15.7)
      RETURN
      END
```

Fig. 4.46 Part of new vectorized subroutine GR2D60 of PART2

```
 1:   2000 CONTINUE
 2:
 3:           .
 4:           .
 5:           .
 6:        CALL GR2D50
 7:
 8:        IF(ABS(DISCR).LE.TOL)    GO TO 2500
 9:
10:        IF(LSLT(3)*LSLT(4)+LSLT(3)*LSLT(2).NE.0)
11:                                 GO TO 3000
12:           .
13:           .
14:           .
15:           .
16:           .
17:   2500 IF(LSLT(3).NE.0)         GO TO 2600
18:        LSLT(3)=1
19:
20:        EGKA1=ENGKA
21:        EGPO1=ENGPO
22:        EGSP1=ENGSP
23:        EGLO1=ENGLO
24:        EGTO1=ENGTO
25:        DSCR1=DISCR
26:                                 GO TO 2000
27:   2600 IF(T.LT.TT(NT))          GO TO 2620
28:        IF(DTN(NT).NE.0.D0)      GO TO 2620
29:
30:   2620 CALL  GR2D60
31:        LSLT(3)=0
32:           .
33:           .
34:           .
```

Fig. 4.47 Part of new vectorized subroutine GR2D08 of PART2

```
 1:   2000 CONTINUE
 2:
 3:           .
 4:           .
 5:           .
 6:        CALL GR2D50
 7:
 8:        IF(ABS(DISCR).LE.TOL)    GO TO 2500
 9:
10:        IF(LSLT(3)*LSLT(4)+LSLT(3)*LSLT(2).NE.0)
11:                                 GO TO 3000
12:
13:        WRITE(6,6200)
14:   6200 FORMAT(1HO,'* TOL OVER LIMIT REPEAT *')
15:
16:        BACKSPACE IUNT18
17:
18:        READ(IUNT18)  IPROB,MONTH,KAY,IEAR,T,N,
19:       .      (X(I,3),VX(I,3),X(I,2),VX(I,2),X(I,1),VX(I,1),I=1,N),
20:       .      DT,ENGKA,ENGPO,ENGSP,ENGLO,ENGTO,DISCR,
21:       .      ENGKE,EGKA1,EGPO1,EGSP1,EGLO1,EGTO1,DSCR,INSEN
22:        IF(NT.EQ.1)              GO TO 3000
23:        NT=NT-1
24:                                 GO TO 2730
25:
26:   2500 IF(LSLT(3).NE.0)         GO TO 2600
27:        LSLT(3)=1
28:
```

Fig. 4.48 GO TO loop in new vectorized subroutine GR2D08 of PART2 (1/2)

```
29:         EGKA1=ENGKA
30:         EGPO1=ENGPO
31:         EGSP1=ENGSP
32:         EGLO1=ENGLO
33:         EGTO1=ENGTO
34:         DSCR1=DISCR
35:                                   GO TO 2000
36:    2600 IF(T.LT.TT(NT))           GO TO 2620
37:         IF(DTN(NT).NE.0.DO)       GO TO 2620
38:
39:    2620 CALL  GR2D60
40:         LSLT(3)=0
41:
42:         IF(T.LT.TT(NT))           GO TO 2000
43:         IF(DTN(NT).EQ.0.DO)       GO TO 3100
44:         DO 2700 I=1,N
45:         DO 2700 J=1,3
46:    2700 VX(I,J)=((VX(I,J)-X(I,J))/DT)*DTN(NT)+X(I,J)
47:
48:         DO 2720 I=1,M
49:         DO 2720 J=1,3
50:         C(I,J)=(C(I,J)*DT)/DTN(NT)
51:    2720 C2(I,J)=(C2(I,J)*DT)/DTN(NT)
52:
53:    2730 DT=DTN(NT)
54:         DDT=DT*DT
55:         NT=NT+1
56:         WRITE(6,6300) T,DT
57:    6300 FORMAT(1HO,'PROGRAM RUN THROUGH T = ',F9.3/1X,
58:        .           'AND DT IS NOW BEING CHANGED TO ',F7.3)
59:                                   GO TO 2000
```

Fig. 4.48 GO TO loop in new vectorized subroutine GR2D08 of PART2 (2/2)

```
 1:   2000 CONTINUE
 2:
 3:           .
 4:           .
 5:           .
 6:           .
 7:
 8:  !XOCL SPREAD DO
 9:        DO 769 IK=1,NPE
10:        IF(FLAG.AND.I_MYPE.NE.1) THEN
11:          IF(LSLT(3,I_TIME1,I_MYPE).EQ.0) THEN
12:            LSLT(3,I_TIME2,I_MYPE)=1
13:            GO TO 769
14:          ELSEIF(LSLT(3,I_TIME1,I_MYPE).NE.0) THEN
15:            LSLT(3,I_TIME2,I_MYPE)=0
16:            IF(T.LT.TT(NT)) THEN
17:              GO TO 769
18:            ENDIF
19:            IF(DTN(NT).NE.0.0D0) THEN
20:              DO 2701 I=I_MIN(I_MYPE),I_MAX(I_MYPE)
21:              DO 2701 J=1,3
22:              VX(I,J,I_TIME2,I_MYPE)=((VX(I,J,I_TIME2,I_MYPE)-
23:       &X(I,J,I_TIME2,I_MYPE))/DT)*DTN(NT)+X(I,J,I_TIME2,I_MYPE)
24:    2701      CONTINUE
25:              DO 2721 I=1,M
26:              DO 2721 J=1,3
27:              C(I,J)=(C(I,J)*DT)/DTN(NT)
28:              C2(I,J)=(C2(I,J)*DT)/DTN(NT)
29:    2721      CONTINUE
30:              DT=DTN(NT)
31:              DDT=DT*DT
32:              NT=NT+1
33:              GO TO 769
34:            ENDIF
35:          ENDIF
36:        ENDIF
37:     769 CONTINUE
38:  !XOCL END SPREAD
```

Fig. 4.49 GO TOO loop in parallelized subroutine GR2D08 of PART2 (1/2)

```
39:
40:         CALL GR2D50
41:   !XOCL BROADCAST(DT_HOLD) (OUT_FLAG)
42:
43:   !XOCL SPREAD DO
44:         DO 515 IK=1,NPE
45:         ILEGAL2(IK)=ILEGAL2_L(IK)
46:     515 CONTINUE
47:   !XOCL END SPREAD SUM(ILEGAL2)
48:
49:         IF(ILEGAL2(1).EQ.0.OR.ILEGAL2(1).EQ.1) GO TO 2010
50:         IF(ILEGAL2(1).EQ.2) GO TO 3000
51:         IF(ILEGAL2(1).EQ.3) GO TO 3200
52:         IF(ILEGAL2(1).EQ.4) GO TO 3150
53:
54:    2010 IF(FLAG) THEN
55:           IF(I_TIME2.EQ.NPE-NPE/2) THEN
56:             I_TIME1=I_TIME2
57:             I_TIME2=1
58:           ELSE
59:             I_TIME1=I_TIME2
60:             I_TIME2=I_TIME2+1
61:           ENDIF
62:         ENDIF
63:         GO TO 2000
```

Fig. 4.49 GO TOO loop in parallelized subroutine GR2D08 of PART2 (2/2)

```
 1:  !XOCL SPREAD DO
 2:        DO 9999 IPENUM=1,NPE
 3:        IF(FLAG.AND.I_MYPE.EQ.1) THEN
 4:
 5:                .
 6:                .
 7:                .
 8:                .
 9:
10:        IF( ABS(DISCR).LE.TOL ) GO TO 2500
11:        IF(LSLT(3,I_TIME1,I_MYPE)*LSLT2(4)+LSLT(3,I_TIME1,I_MYPE)*
12:       &LSLT2(2).NE.0) GO TO 3001
13:        GO TO 3201
14:   2500 IF( LSLT(3,I_TIME1,I_MYPE).NE.0 ) GO TO 2600
15:        LSLT(3,I_TIME2,I_MYPE)=1
16:        EGKA1=ENGKA
17:        EGPO1=ENGPO(I_TIME2,I_MYPE)
18:        EGSP1=ENGSP(I_TIME2,I_MYPE)
19:        EGLO1=ENGLO(I_TIME2,I_MYPE)
20:        EGTO1=ENGTO
21:        DSCR1=DISCR
22:        GO TO 2005
23:
24:   2600 CONTINUE
25:        IF(T.LT.TT(NT)) GO TO 2620
26:        IF(DTN(NT).NE.0.0D0) GO TO 2620
27:        INSEN=0
28:
29:   2620 CONTINUE
30:
31:                .
32:                .
33:                .
34:                .
35:
36:        LSLT(3,I_TIME2,I_MYPE)=0
37:       IF(INSEN.EQ.0) THEN
38:         GO TO 2002
39:       ELSE
40:         GO TO 2001
41:       ENDIF
42:
```

Fig. 4.50 Part of GO TO loop in parallelized subroutine GR2D50 of PART2 (1/2)

```
43:  2001 ILEGAL2_L(I_MYPE)=1
44:       IF(T.LT.TT(NT)) THEN
45:         GO TO 2005
46:       ENDIF
47:       IF(DTN(NT).NE.0.0D0) THEN
48:         DO 2700 I=I_MIN(I_MYPE),I_MAX(I_MYPE)
49:         DO 2700 J=1,3
50:         VX(I,J,I_TIME2,I_MYPE)=((VX(I,J,I_TIME2,I_MYPE)-X(I,J,
51:      &I_TIME2,I_MYPE))/DT)*DTN(NT)+X(I,J,I_TIME2,I_MYPE)
52:   2700   CONTINUE
53:         DO 2720 I=1,M
54:         DO 2720 J=1,3
55:         C(I,J)=(C(I,J)*DT)/DTN(NT)
56:         C2(I,J)=(C2(I,J)*DT)/DTN(NT)
57:   2720   CONTINUE
58:         DT=DTN(NT)
59:         DDT=DT*DT
60:         NT=NT+1
61:         GO TO 2005
62:       ENDIF
63:
64:   2002 ILEGAL2_L(I_MYPE)=4
65:       GO TO 2005
66:   3001 ILEGAL2_L(I_MYPE)=2
67:       GO TO 2005
68:   3201 ILEGAL2_L(I_MYPE)=3
69:   2005 CONTINUE
70:
71:       ENDIF
72:   9999 CONTINUE
73:  !XOCL END SPREAD
74:       RETURN
75:       END
```

Fig. 4.50 Part of GO TO loop in parallelized subroutine GR2D50 of PART2 (2/2)

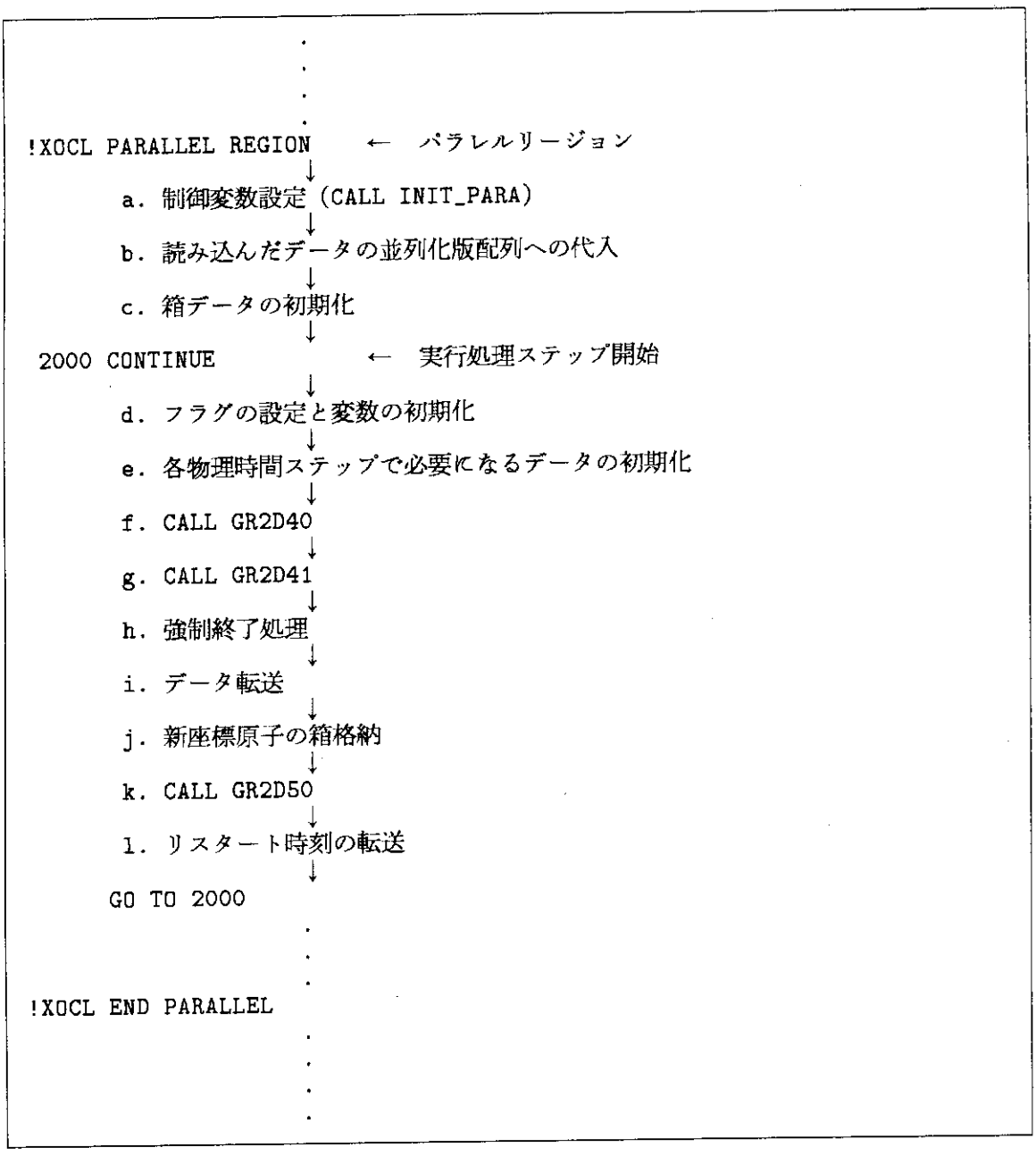


Fig. 4.51 Outline of parallel region

```
      PARAMETER(NPE=1)
```

Fig. 4.52 Include file INC_PE

```
*INCLUDE INC_PE
!XOCL PROCESSOR P(NPE)
!XOCL INDEX PARTITION PP=(P,INDEX=1:NPE,PART=BAND)
      INTEGER ILEGAL1(NPE),ILEGAL2(NPE)
```

Fig. 4.53 Include file INC_MAIN

```
*INCLUDE INC_PE
!XOCL PROCESSOR P(NPE)
!XOCL SUBPROCESSOR SUBP(NPE)=P(1:NPE)
!XOCL INDEX PARTITION PP=(SUBP,INDEX=1:NPE,PART=BAND)
```

Fig. 4.54 Include file INC_SUB

```
      LOGICAL FLAG,OUT_FLAG
      DIMENSION  X(NPAR,3,NPE-NPE/2,NPE),
     .           VX(NPAR,3,NPE-NPE/2,NPE),
     .           ENGKE(NPE-NPE/2,NPE),
     .           ENGL(NPE-NPE/2,NPE),
     .           ENGKG(NPE-NPE/2,NPE),
     .           ENGPO(NPE-NPE/2,NPE),
     .           ENGSP(NPE-NPE/2,NPE),
     .           ENGLO(NPE-NPE/2,NPE),
     .           IIBND(NPAR,NPE-NPE/2,NPE),
     .           ICOV(NPAR,NPE-NPE/2,NPE),
     .           IBOX(6,NBOXES,NPE-NPE/2,NPE),
     .           ZZ1(NPAR,NPE-NPE/2,NPE),
     .           ZZ2(NPAR,NPE-NPE/2,NPE),
     .           ZZ3(NPAR,NPE-NPE/2,NPE),
     .           LSLT(3,NPE-NPE/2,NPE)
      COMMON /PARA_ARRAY2/I_MIN(NPE),
     .                    I_MAX(NPE+1)
      COMMON /PARA_VALUE/I_MYPE,
     .                   I_INIT,
     .                   I_LCOUNT,
     .                   I_TIME1,
     .                   I_TIME2,
     .                   FLAG,
     .                   OUT_FLAG,
     .                   DT_HOLD(2),
     .                   T_UNTIL(2,NPE)
      COMMON /PARA_ARRAY_G/X_G(NPAR,3,NPE-NPE/2,NPE),
     .                     VX_G(NPAR,3,NPE-NPE/2,NPE),
     .                     ENGKE_G(NPE-NPE/2,NPE),
     .                     ENGL_G(NPE-NPE/2,NPE),
     .                     ENGKG_G(NPE-NPE/2,NPE),
     .                     ENGPO_G(NPE-NPE/2,NPE),
     .                     ENGSP_G(NPE-NPE/2,NPE),
     .                     ENGLO_G(NPE-NPE/2,NPE),
     .                     IIBND_G(NPAR,NPE-NPE/2,NPE),
     .                     ICOV_G(NPAR,NPE-NPE/2,NPE),
     .                     IBOX_G(6,NBOXES,NPE-NPE/2,NPE),
     .                     ZZ1_G(NPAR,NPE-NPE/2,NPE),
     .                     ZZ2_G(NPAR,NPE-NPE/2,NPE),
     .                     ZZ3_G(NPAR,NPE-NPE/2,NPE),
     .                     LSLT_G(3,NPE-NPE/2,NPE)
      COMMON /PARA_CON2/ILEGAL1_L(NPE),
     .                  ILEGAL2_L(NPE),
     .                  T_UNTIL_L(2,NPE)
```

Fig. 4.55 Include file INC_PARA (1/2)

```
!XOCL LOCAL X(:,:,:,/PP),VX(:,:,:,/PP),ENGKE(:,/PP),ENGL(:,/PP)
!XOCL LOCAL ENGKG(:,/PP),ENGPO(:,/PP),ENGSP(:,/PP),ENGLO(:,/PP)
!XOCL LOCAL IIBND(:,:,/PP),ICOV(:,:,/PP),IBOX(:,:,:,/PP)
!XOCL LOCAL ZZ1(:,:,/PP),ZZ2(:,:,/PP),ZZ3(:,:,/PP)
!XOCL LOCAL LSLT(:,:,/PP)
!XOCL LOCAL ILEGAL1_L(/PP),ILEGAL2_L(/PP),T_UNTIL_L(:,/PP)
!XOCL GLOBAL X_G,VX_G,ENGKE_G,ENGL_G
!XOCL GLOBAL ENGKG_G,ENGPO_G,ENGSP_G,ENGLO_G
!XOCL GLOBAL IIBND_G,ICOV_G,IBOX_G,ZZ1_G,ZZ2_G,ZZ3_G
!XOCL GLOBAL LSLT_G
      EQUIVALENCE (X,X_G),(VX,VX_G),(ENGKE,ENGKE_G),(ENGL,ENGL_G),
     .            (ENGKG,ENGKG_G),(ENGPO,ENGPO_G),(ENGSP,ENGSP_G),
     .            (ENGLO,ENGLO_G),(IIBND,IIBND_G),(ICOV,ICOV_G),
     .            (IBOX,IBOX_G),(ZZ1,ZZ1_G),(ZZ2,ZZ2_G),(ZZ3,ZZ3_G),
     .            (LSLT,LSLT_G)
```

Fig. 4.55 Include file INC_PARA (2/2)

```
 1:        SUBROUTINE INIT_PARA
 2:              IMPLICIT REAL*8(A-H,O-Z)
 3:  *INCLUDE INCPAR
 4:  *INCLUDE INC_SUB
 5:  *INCLUDE INC_PARA
 6:
 7:        COMMON  /CNTRL/CNTRL(29)
 8:        COMMON  /SENSE/LSLT2(5)
 9:        EQUIVALENCE (CNTRL(6),N)
10:
11:  !XOCL SPREAD DO
12:        DO 10 I=1,NPE
13:        I_MYPE=I
14:        I_INIT=NPE-(I-1)
15:     10 CONTINUE
16:  !XOCL END SPREAD
17:
18:        I_MAX(NPE+1)=0
19:        I_PAR=N
20:        I_AREA=0
21:        DO 11 I=NPE,1,-1
22:        I_PAR=I_PAR-I_AREA
23:        I_AREA=I_PAR/I
24:        I_MAX(I)=I_MAX(I+1)+I_AREA
25:     11 CONTINUE
26:        DO 20 I=NPE,1,-1
27:        I_MIN(I)=I_MAX(I+1)+1
28:     20 CONTINUE
29:
30:        I_LCOUNT=0
31:        IF(NPE.EQ.1.OR.NPE.EQ.2) THEN
32:          I_TIME1=1
33:          I_TIME2=1
34:        ELSE
35:          I_TIME1=1
36:          I_TIME2=2
37:        ENDIF
38:
39:  !XOCL SPREAD DO
40:        DO 144 IK=1,NPE
41:        LSLT(3,I_TIME1,IK)=LSLT2(3)
42:    144 CONTINUE
43:  !XOCL END SPREAD
44:
45:        RETURN
46:        END
```

Fig. 4.56 Subroutine INIT_PARA

```
!XOCL SPREAD DO
      DO 40 IK=1,NPE
      DO 41 IM=1,N
      DO 41 JM=1,3
      X(IM,JM,I_TIME1,IK)=X_R(IM,JM)
      VX(IM,JM,I_TIME1,IK)=VX_R(IM,JM)
   41 CONTINUE
      IF(I_FIRST.EQ.0) THEN
        ENGKE(I_TIME1,IK)=2000.D0*ENGKE_R
        ENGLO(I_TIME1,IK)=0.0D0
      ELSE
        ENGKE(I_TIME1,IK)=ENGKE_R
        ENGLO(I_TIME1,IK)=ENGLO_R
      ENDIF
   40 CONTINUE
!XOCL END SPREAD
```

Fig. 4.57 Substitution read data for some arrays

```
!XOCL SPREAD DO
      DO 39 IK=1,NPE
      ILEGAL1_L(IK)=0
      DO 38 IM=1,NBOXES
      DO 38 JM=1,6
      IBOX(JM,IM,I_TIME1,IK)=0
   38 CONTINUE
      DO 37 IM=1,N
      IIBND(IM,I_TIME1,IK)=0
   37 CONTINUE
   39 CONTINUE
!XOCL END SPREAD

      FLAG=.TRUE.
      CALL GR2D3(IER)
      FLAG=.FALSE.

      DO 221 IK=1,NPE
      ILEGAL1(IK)=0
  221 CONTINUE
!XOCL SPREAD DO
      DO 222 IK=1,NPE
      ILEGAL1(IK)=ILEGAL1_L(IK)
  222 CONTINUE
!XOCL END SPREAD SUM(ILEGAL1)
      DO 223 IK=1,NPE
      IF(ILEGAL1(IK).NE.0) GO TO 8000
  223 CONTINUE
```

Fig. 4.58 Initialization of boxes

```
!XOCL SPREAD DO
      DO 9999 IPENUM=1,NPE
      IF(FLAG) THEN
      IF(ISW.EQ.1) THEN
        I_START=1
        I_END=N
        I_TIMEA=I_TIME1
        ISW=0
      ELSE
        I_START=I_MIN(I_MYPE)
        I_END=I_MAX(I_MYPE)
        I_TIMEA=I_TIME2
      ENDIF
      DO 4002 I1=I_START,I_END
      IX=X(I1,1,I_TIMEA,I_MYPE)+0.55D0
      IY=X(I1,2,I_TIMEA,I_MYPE)+0.55D0
      IZ=X(I1,3,I_TIMEA,I_MYPE)+0.55D0
      I2=1+IX+IALP*IY+IBET*IZ
      IF(I2.GT.IGAM) GO TO 8000
      I2_V(I1)=I2
 4002 CONTINUE
      DO 4000 I1=I_START,I_END
      I2=I2_V(I1)
      IF(IBOX(6,I2,I_TIMEA,I_MYPE).NE.0) GO TO 800
      IBOX(6,I2,I_TIMEA,I_MYPE)=I1
      GO TO 3000
  800 IF(IBOX(5,I2,I_TIMEA,I_MYPE).NE.0) GO TO 900
      IBOX(5,I2,I_TIMEA,I_MYPE)=I1
      GO TO 3000
  900 IF(IBOX(4,I2,I_TIMEA,I_MYPE).NE.0) GO TO 1000
      IBOX(4,I2,I_TIMEA,I_MYPE)=I1
      GO TO 3000
 1000 IF(IBOX(3,I2,I_TIMEA,I_MYPE).NE.0) GO TO 1500
      IBOX(3,I2,I_TIMEA,I_MYPE)=I1
      GO TO 3000
 1500 IF(IBOX(2,I2,I_TIMEA,I_MYPE).NE.0) GO TO 1600
      IBOX(2,I2,I_TIMEA,I_MYPE)=I1
      GO TO 3000
 1600 IF(IBOX(1,I2,I_TIMEA,I_MYPE).NE.0) GO TO 8100
      IBOX(1,I2,I_TIMEA,I_MYPE)=I1
 3000 IIBND(I1,I_TIMEA,I_MYPE)=I2
 4000 CONTINUE
      IER=0
      GO TO 9000
 8000 IER=1
      GO TO 9000
 8100 IER=2
 9000 CONTINUE
      ILEGAL1_L(I_MYPE)=IER
      ENDIF
 9999 CONTINUE
!XOCL END SPREAD
      RETURN
      END
```

Fig. 4.59 Parallelized subroutine GR2D3 of PART2

```
 1:         I_LCOUNT=I_LCOUNT+1
 2:         FLAG=.FALSE.
 3:         OUT_FLAG=.FALSE.
 4:   !XOCL SPREAD DO
 5:         DO 50 IK=1,NPE
 6:         IF(NPE.EQ.1) FLAG=.TRUE.
 7:         ILEGAL1_L(IK)=0
 8:         ILEGAL2_L(IK)=0
 9:         T_UNTIL_L(1,IK)=0.0D0
10:         T_UNTIL_L(2,IK)=0.0D0
11:         IF(I_LCOUNT.GE.I_INIT.AND.(T.LT.TT(NT)+DT.OR.DTN(NT).NE.
12:        &0.0D0)) THEN
13:           IF(MOD(NPE,2).NE.0) THEN
10:             IF(MOD(IK,2).NE.0.AND.MOD(I_LCOUNT,2).NE.0) FLAG=.TRUE.
10:             IF(MOD(IK,2).EQ.0.AND.MOD(I_LCOUNT,2).EQ.0) FLAG=.TRUE.
10:           ELSE
14:             IF(MOD(IK,2).NE.0.AND.MOD(I_LCOUNT,2).EQ.0) FLAG=.TRUE.
15:             IF(MOD(IK,2).EQ.0.AND.MOD(I_LCOUNT,2).NE.0) FLAG=.TRUE.
16:           ENDIF
17:         ENDIF
18:      50 CONTINUE
19:   !XOCL END SPREAD
20:         DO 500 IK=1,NPE
21:         ILEGAL1(IK)=0
22:         ILEGAL2(IK)=0
23:         T_UNTIL(1,IK)=0.0D0
24:         T_UNTIL(2,IK)=0.0D0
25:     500 CONTINUE
```

Fig. 4.60 Initialization of flags and variables to control execution

```
 1:  !XOCL SPREAD DO
 2:        DO 502 IK=1,NPE
 3:        IF(FLAG.AND.I_MYPE.EQ.NPE) THEN
 4:          DO 503 I=1,N
 5:          ICOV(I,I_TIME2,IK)=0
 6:          IO=IBND(I)
 7:          IF(IO.GE.1.AND.IO.LE.6)   ICOV(I,I_TIME2,IK)=2
 8:          IF(IO.GE.7.AND.IO.LE.18)  ICOV(I,I_TIME2,IK)=3
 9:          IF(IO.EQ.19)              ICOV(I,I_TIME2,IK)=3
10:          IF(IO.GE.20.AND.IO.LE.22) ICOV(I,I_TIME2,IK)=4
11:          IF(IO.GE.23.AND.IO.LE.25) ICOV(I,I_TIME2,IK)=3
12:          IF(IO.EQ.26)              ICOV(I,I_TIME2,IK)=4
13:    503   CONTINUE
14:          DO 505 IM=1,N
15:          ZZ1(IM,I_TIME2,IK)=0.0D0
16:          ZZ2(IM,I_TIME2,IK)=0.0D0
17:          ZZ3(IM,I_TIME2,IK)=0.0D0
18:    505   CONTINUE
19:          ENGL(I_TIME2,IK)=0.0D0
20:          ENGKG(I_TIME2,IK)=0.0D0
21:          ENGPO(I_TIME2,IK)=0.0D0
22:          ENGSP(I_TIME2,IK)=0.0D0
23:        ENDIF
24:    502 CONTINUE
25:  !XOCL END SPREAD
```

Fig. 4.61 Initialization that needs once a step

```
      CALL  GR2D41(N)
!XOCL SPREAD DO
      DO 9999 IPENUM=1,NPE
      IF(FLAG) THEN
      IM=0
      IP=0
      DO 5000 I=I_MIN(I_MYPE),I_MAX(I_MYPE)
      IF(IBND(I).LT.0) THEN
        IM=IM+1
        IM_BOUN(IM)=I
      ELSEIF(IBND(I).EQ.0) THEN
      VX(I,1,I_TIME2,I_MYPE)=2.D0*(AC(I,1)*DDT+VX(I,1,I_TIME1,I_MYPE))-X
     &(I,1,I_TIME1,I_MYPE)
      VX(I,2,I_TIME2,I_MYPE)=2.D0*(AC(I,2)*DDT+VX(I,2,I_TIME1,I_MYPE))-X
     &(I,2,I_TIME1,I_MYPE)
      VX(I,3,I_TIME2,I_MYPE)=2.D0*(AC(I,3)*DDT+VX(I,3,I_TIME1,I_MYPE))-X
     &(I,3,I_TIME1,I_MYPE)
      ELSEIF(IBND(I).GT.0) THEN
        IP=IP+1
        IP_BOUN(IP)=I
      ENDIF
 5000 CONTINUE
```

Fig. 4.62 Parallelized subroutine GR2D40 of PART2 (1/3)

```
      ENGL_BAK=ENGL(I_TIME2,I_MYPE)
      ENGSP_BAK=ENGSP(I_TIME2,I_MYPE)
*VOCL LOOP,NOVREC
      DO 5001 LL=1,IM
      I=IM_BOUN(LL)
      L=-IBND(I)
      ENGL_BAK=ENGL_BAK-C2(L,1)*VX(I,1,I_TIME1,I_MYPE)-X(I,1,I_TIME1,I_M
     &YPE)**2
      DXYZ=X(I,1,I_TIME1,I_MYPE)-TB(I,1)
      ENGSP_BAK=ENGSP_BAK-A2(L,1)*DXYZ-0.5D0*B2(L,1)*DXYZ**2
      AC(I,1)=AC(I,1)+A2(L,1)+B2(L,1)*DXYZ+C2(L,1)*(VX(I,1,I_TIME1,I_MYP
     &E)-X(I,1,I_TIME1,I_MYPE))
      ENGL_BAK=ENGL_BAK-C2(L,2)*VX(I,2,I_TIME1,I_MYPE)-X(I,2,I_TIME1,I_M
     &YPE)**2
      DXYZ=X(I,2,I_TIME1,I_MYPE)-TB(I,2)
      ENGSP_BAK=ENGSP_BAK-A2(L,2)*DXYZ-0.5D0*B2(L,2)*DXYZ**2
      AC(I,2)=AC(I,2)+A2(L,2)+B2(L,2)*DXYZ+C2(L,2)*(VX(I,2,I_TIME1,I_MYP
     &E)-X(I,2,I_TIME1,I_MYPE))
      ENGL_BAK=ENGL_BAK-C2(L,3)*VX(I,3,I_TIME1,I_MYPE)-X(I,3,I_TIME1,I_M
     &YPE)**2
      DXYZ=X(I,3,I_TIME1,I_MYPE)-TB(I,3)
      ENGSP_BAK=ENGSP_BAK-A2(L,3)*DXYZ-0.5D0*B2(L,3)*DXYZ**2
      AC(I,3)=AC(I,3)+A2(L,3)+B2(L,3)*DXYZ+C2(L,3)*(VX(I,3,I_TIME1,I_MYP
     &E)-X(I,3,I_TIME1,I_MYPE))
      VX(I,1,I_TIME2,I_MYPE)=2.D0*(AC(I,1)*DDT+VX(I,1,I_TIME1,I_MYPE))-X
     &(I,1,I_TIME1,I_MYPE)
      VX(I,2,I_TIME2,I_MYPE)=2.D0*(AC(I,2)*DDT+VX(I,2,I_TIME1,I_MYPE))-X
     &(I,2,I_TIME1,I_MYPE)
      VX(I,3,I_TIME2,I_MYPE)=2.D0*(AC(I,3)*DDT+VX(I,3,I_TIME1,I_MYPE))-X
     &(I,3,I_TIME1,I_MYPE)
 5001 CONTINUE
      ENGL(I_TIME2,I_MYPE)=ENGL_BAK
      ENGSP(I_TIME2,I_MYPE)=ENGSP_BAK
```

Fig. 4.62 Parallelized subroutine GR2D40 of PART2 (2/3)

```
      ENGL_BAK=ENGL(I_TIME2,I_MYPE)
      ENGSP_BAK=ENGSP(I_TIME2,I_MYPE)
*VOCL LOOP,NOVREC
      DO 5002 LL=1,IP
      I=IP_BOUN(LL)
      L=IBND(I)
      ENGL_BAK=ENGL_BAK-C(L,1)*(VX(I,1,I_TIME1,I_MYPE)-X(I,1,I_TIME1,I_M
     &YPE))**2
      DXYZ=X(I,1,I_TIME1,I_MYPE)-TB(I,1)
      ENGSP_BAK=ENGSP_BAK-A(L,1)*DXYZ-0.5D0*B(L,1)*DXYZ**2
      AC(I,1)=AC(I,1)+A(L,1)+B(L,1)*DXYZ+C(L,1)*(VX(I,1,I_TIME1,I_MYPE)-
     &X(I,1,I_TIME1,I_MYPE))
      ENGL_BAK=ENGL_BAK-C(L,2)*(VX(I,2,I_TIME1,I_MYPE)-X(I,2,I_TIME1,I_M
     &YPE))**2
      DXYZ=X(I,2,I_TIME1,I_MYPE)-TB(I,2)
      ENGSP_BAK=ENGSP_BAK-A(L,2)*DXYZ-0.5D0*B(L,2)*DXYZ**2
      AC(I,2)=AC(I,2)+A(L,2)+B(L,2)*DXYZ+C(L,2)*(VX(I,2,I_TIME1,I_MYPE)-
     &X(I,2,I_TIME1,I_MYPE))
      ENGL_BAK=ENGL_BAK-C(L,3)*(VX(I,3,I_TIME1,I_MYPE)-X(I,3,I_TIME1,I_M
     &YPE))**2
      DXYZ=X(I,3,I_TIME1,I_MYPE)-TB(I,3)
      ENGSP_BAK=ENGSP_BAK-A(L,3)*DXYZ-0.5D0*B(L,3)*DXYZ**2
      AC(I,3)=AC(I,3)+A(L,3)+B(L,3)*DXYZ+C(L,3)*(VX(I,3,I_TIME1,I_MYPE)-
     &X(I,3,I_TIME1,I_MYPE))
      VX(I,1,I_TIME2,I_MYPE)=2.D0*(AC(I,1)*DDT+VX(I,1,I_TIME1,I_MYPE))-X
     &(I,1,I_TIME1,I_MYPE)
      VX(I,2,I_TIME2,I_MYPE)=2.D0*(AC(I,2)*DDT+VX(I,2,I_TIME1,I_MYPE))-X
     &(I,2,I_TIME1,I_MYPE)
      VX(I,3,I_TIME2,I_MYPE)=2.D0*(AC(I,3)*DDT+VX(I,3,I_TIME1,I_MYPE))-X
     &(I,3,I_TIME1,I_MYPE)
 5002 CONTINUE
      ENGL(I_TIME2,I_MYPE)=ENGL_BAK
      ENGSP(I_TIME2,I_MYPE)=ENGSP_BAK
      ENGKG_BAK=ENGKG(I_TIME2,I_MYPE)
      DO 1001 I=I_MIN(I_MYPE),I_MAX(I_MYPE)
      DO 1002 J=1,3
      X(I,J,I_TIME2,I_MYPE)=(VX(I,J,I_TIME2,I_MYPE)+X(I,J,I_TIME1,I_MYPE
     &))*0.5D0
      ENGKG_BAK=ENGKG_BAK+(VX(I,J,I_TIME2,I_MYPE)-X(I,J,I_TIME2,I_MYPE))
     &**2
 1002 CONTINUE
 1001 CONTINUE
      ENGKG(I_TIME2,I_MYPE)=ENGKG_BAK
      T=T+DT
      ENDIF
 9999 CONTINUE
!XOCL END SPREAD
      RETURN
      END
```

Fig. 4.62 Parallelized subroutine GR2D40 of PART2 (3/3)

```
 1:         I_AREA_HOLD=0
 2:         DO 600 I1=I_MIN(I_MYPE),I_MAX(I_MYPE)
 3:         K1=0
 4:         IF(ICOMB_V(I1,1).EQ.1) THEN
 5:           XW_S=X(I4_LIST(I1,1),1,I_TIME1,I_MYPE)-X(I1,1,I_TIME1,
 6:        &I_MYPE)
 7:           YW_S=X(I4_LIST(I1,1),2,I_TIME1,I_MYPE)-X(I1,2,I_TIME1,
 8:        &I_MYPE)
 9:           ZW_S=X(I4_LIST(I1,1),3,I_TIME1,I_MYPE)-X(I1,3,I_TIME1,
10:        &I_MYPE)
11:           WRK2=XW_S**2+YW_S**2+ZW_S**2
12:           WRK_S=SQRT(WRK2)
13:           IF(WRK_S.GT.RMAX(1)) ICOMB_V(I1,1)=0
14:         ENDIF
15:         DO 601 I22=1,ISEARCH_NUM(I1)
16:           XW_S=X(I4_LIST(I1,I22),1,I_TIME1,I_MYPE)-X(I1,1,I_TIME1,
17:        &I_MYPE)
18:           YW_S=X(I4_LIST(I1,I22),2,I_TIME1,I_MYPE)-X(I1,2,I_TIME1,
19:        &I_MYPE)
20:           ZW_S=X(I4_LIST(I1,I22),3,I_TIME1,I_MYPE)-X(I1,3,I_TIME1,
21:        &I_MYPE)
22:           WRK2=XW_S**2+YW_S**2+ZW_S**2
23:           WRK_S=SQRT(WRK2)
24:           I11=I1-I4_LIST(I1,I22)
25:           IF(WRK_S.LE.RMAX(1).AND.I11.NE.0) THEN
26:           I_AREA=0
27:           IF(I_MYPE.GT.1.AND.I_MYPE.LT.NPE) THEN
28:             IF(I4_LIST(I1,I22).LE.I_MAX(I_MYPE-1).AND.I4_LIST(I1,I22)
29:        &.GE.I_MIN(I_MYPE+1)) I_AREA=1
30:           ELSEIF(I_MYPE.EQ.1) THEN
31:             IF(I4_LIST(I1,I22).GE.I_MIN(I_MYPE+1)) I_AREA=1
32:           ELSEIF(I_MYPE.EQ.NPE) THEN
33:             IF(I4_LIST(I1,I22).LE.I_MAX(I_MYPE-1)) I_AREA=1
34:           ELSE
35:             I_AREA_HOLD=1
36:           ENDIF
37:           IF(NPE.EQ.1) I_AREA=1
38:             K1=K1+1
39:             XW(I1,K1)=XW_S
40:             YW(I1,K1)=YW_S
41:             ZW(I1,K1)=ZW_S
42:             WRK(I1,K1)=WRK_S
43:             I4_LIST2(I1,K1)=I4_LIST(I1,I22)
44:           ENDIF
45:       601 CONTINUE
46:           I_MAIN(I1)=K1
47:       600 CONTINUE
48:
49:           IF(I_AREA_HOLD.EQ.1) THEN
50:             T_UNTIL_L(1,I_MYPE)=T
51:             T_UNTIL_L(2,I_MYPE)=DT
52:             GO TO 9998
53:           ENDIF
```

Fig. 4.63 Discrimination process for program suspending in parallelized subroutine GR2D41 of PART2

```
 1:  !XOCL SPREAD DO
 2:        DO 347 IK=1,NPE
 3:        DO 348 IM=1,2
 4:        T_UNTIL(IM,IK)=T_UNTIL_L(IM,IK)
 5:    348 CONTINUE
 6:    347 CONTINUE
 7:  !XOCL END SPREAD SUM(T_UNTIL)
 8:        DO 743 IK=1,NPE
 9:        IF(T_UNTIL(1,IK).NE.0.0D0) THEN
10:          T_NOW   = T_UNTIL(1,IK)
11:          DT_NOW  = T_UNTIL(2,IK)
12:          T_REST  = DT_HOLD(1)
13:          DT_REST = DT_HOLD(2)
14:          GO TO 3300
15:        ENDIF
16:    743 CONTINUE
```

Fig. 4.64 Program suspending process in parallelized subroutine GR2D08 of PART2

```
!XOCL SPREAD DO
      DO 59 IK=1,NPE
      IF(FLAG) THEN
      IF(I_MYPE.GT.1) THEN
!XOCL SPREAD MOVE
        DO 511 JM=1,N
        ICOV_G(JM,I_TIME2,I_MYPE-1)=ICOV(JM,I_TIME2,I_MYPE)
 511    CONTINUE
!XOCL END SPREAD (MW1)
!XOCL MOVEWAIT (MW1)
!XOCL SPREAD MOVE
        DO 512 JM=1,N
        ZZ1_G(JM,I_TIME2,I_MYPE-1)=ZZ1(JM,I_TIME2,I_MYPE)
        ZZ2_G(JM,I_TIME2,I_MYPE-1)=ZZ2(JM,I_TIME2,I_MYPE)
        ZZ3_G(JM,I_TIME2,I_MYPE-1)=ZZ3(JM,I_TIME2,I_MYPE)
 512    CONTINUE
!XOCL END SPREAD (MW4)
!XOCL MOVEWAIT (MW4)
        ENGL_G(I_TIME2,I_MYPE-1)=ENGL(I_TIME2,I_MYPE)
        ENGSP_G(I_TIME2,I_MYPE-1)=ENGSP(I_TIME2,I_MYPE)
        ENGPO_G(I_TIME2,I_MYPE-1)=ENGPO(I_TIME2,I_MYPE)
        ENGKG_G(I_TIME2,I_MYPE-1)=ENGKG(I_TIME2,I_MYPE)
      ENDIF
      IF(I_MYPE.LT.NPE) THEN
!XOCL SPREAD MOVE
        DO 507 JM=I_MIN(I_MYPE),I_MAX(I_MYPE)
        DO 507 IM=1,3
        X_G(JM,IM,I_TIME2,I_MYPE+1)=X(JM,IM,I_TIME2,I_MYPE)
        VX_G(JM,IM,I_TIME2,I_MYPE+1)=VX(JM,IM,I_TIME2,I_MYPE)
 507    CONTINUE
!XOCL END SPREAD (MW3)
!XOCL MOVEWAIT (MW3)
!XOCL SPREAD MOVE
        DO 506 JM=1,N
        DO 506 IM=1,3
        X(JM,IM,I_TIME2,I_MYPE)=X_G(JM,IM,I_TIME2,I_MYPE+1)
        VX(JM,IM,I_TIME2,I_MYPE)=VX_G(JM,IM,I_TIME2,I_MYPE+1)
 506    CONTINUE
!XOCL END SPREAD (MW2)
!XOCL MOVEWAIT (MW2)
      ENDIF
      ENDIF
  59 CONTINUE
!XOCL END SPREAD
```

Fig. 4.65 Data transfer

```
!XOCL SPREAD DO
      DO 7788 IK=1,NPE
      IF(FLAG.AND.I_MYPE.EQ.NPE) THEN
        DO 504 IM=1,NBOXES
        DO 504 JM=1,6
        IBOX(JM,IM,I_TIME2,IK)=0
 504    CONTINUE
      ENDIF
7788  CONTINUE
!XOCL END SPREAD
!XOCL SPREAD DO
      DO 558 IK=1,NPE
      IF(FLAG) THEN
      IF(I_MYPE.LT.NPE) THEN
!XOCL SPREAD MOVE
        DO 513 IM=1,NBOXES
        DO 513 JM=1,6
        IBOX(JM,IM,I_TIME2,I_MYPE)=IBOX_G(JM,IM,I_TIME2,I_MYPE+1)
 513    CONTINUE
!XOCL END SPREAD (ID4)
!XOCL MOVEWAIT (ID4)
      ENDIF
      ENDIF
 558  CONTINUE
!XOCL END SPREAD
      CALL GR2D3(IER)
!XOCL SPREAD DO
      DO 509 IK=1,NPE
      ILEGAL1(IK)=ILEGAL1_L(IK)
 509  CONTINUE
!XOCL END SPREAD SUM(ILEGAL1)
      DO 510 IK=1,NPE
      IF(ILEGAL1(IK).NE.0) GO TO 8000
 510  CONTINUE
!XOCL SPREAD DO
      DO 53 IK=1,NPE
      IF(FLAG) THEN
      IF(I_MYPE.LT.NPE) THEN
!XOCL SPREAD MOVE
        DO 514 IM=1,NBOXES
        DO 514 JM=1,6
        IBOX_G(JM,IM,I_TIME2,I_MYPE+1)=IBOX(JM,IM,I_TIME2,I_MYPE)
 514    CONTINUE
!XOCL END SPREAD (ID5)
!XOCL MOVEWAIT (ID5)
      ENDIF
      ENDIF
  53  CONTINUE
!XOCL END SPREAD
```

Fig. 4.66 Definition of boxes about new coordinates of atoms

```
 1:  !XOCL SPREAD DO
 2:        DO 9999 IPENUM=1,NPE
 3:        IF(FLAG.AND.I_MYPE.EQ.1) THEN
 4:        IOUT_COUNT=IOUT_COUNT+1
 5:        ENGKG(I_TIME2,I_MYPE)=1000.D0*ENGKG(I_TIME2,I_MYPE)/DDT
 6:        ENGPO(I_TIME2,I_MYPE)=2000.D0*ENGPO(I_TIME2,I_MYPE)
 7:        ENGSP(I_TIME2,I_MYPE)=2000.D0*ENGSP(I_TIME2,I_MYPE)
 8:        ENGKA        =0.5D0*(ENGKG(I_TIME2,I_MYPE)+ENGKE(I_TIME1,I_MYPE))
 9:        ENGLO(I_TIME2,I_MYPE)=ENGLO(I_TIME1,I_MYPE)+2000.D0*ENGL(I_TIME2,I
10:       &_MYPE)
11:        IF(LSLT2(2).EQ.0) GO TO 1500
12:        ENGTO=ENGPO(I_TIME2,I_MYPE)+ENGSP(I_TIME2,I_MYPE)+ENGKA
13:        LSLT2(2)=0
14:   1500 ENGTN=ENGPO(I_TIME2,I_MYPE)+ENGSP(I_TIME2,I_MYPE)+ENGKA+ENGLO(I_TI
15:       *ME2,I_MYPE)
16:        DISCR=ENGTO-ENGTN
17:        ENGTO=ENGTN
18:        ENGKE(I_TIME2,I_MYPE)=ENGKG(I_TIME2,I_MYPE)
19:                              .
20:                              .
21:                              .
22:                              .
```

Fig. 4.67 Part of parallelized subroutine GR2D50 of PART2 (1/2)

```
23:        IF(LSLT2(1).EQ.1) THEN
24:        IF(IOUT_COUNT.EQ.50.OR.IOUT_COUNT.EQ.51.OR.INSEN.EQ.0) THEN
25:        DT_HOLD(1)=T
26:        DT_HOLD(2)=DT
27:        OUT_FLAG=.TRUE.
28:        IOUT_COUNT=0
29:        WRITE(19) IPROB,MONTH,KAY,IEAR,T,N,
30:       .          (X(I,3,I_TIME2,I_MYPE),VX(I,3,I_TIME2,I_MYPE),X(I,2,I_TI
31:       .ME2,I_MYPE),VX(I,2,I_TIME2,I_MYPE),X(I,1,I_TIME2,I_MYPE),VX(I,1,I_
32:       .TIME2,I_MYPE),I=1,4505),DT,ENGKA,ENGPO(I_TIME2,I_MYPE),ENGSP(I_TIM
33:       .E2,I_MYPE),ENGLO(I_TIME2,I_MYPE),ENGTO,DISCR,ENGKE(I_TIME2,I_MYPE)
34:       .,EGKA1,EGPO1,EGSP1,EGLO1,EGTO1,DSCR1,INSEN
35:        ENDIF
36:        WRITE(6,5001) T,ENGKA,ENGPO(I_TIME2,I_MYPE),ENGSP(I_TIME2,I_MYPE),
37:       &ENGLO(I_TIME2,I_MYPE),ENGTO,DISCR
38:    5001 FORMAT(1H ,7F15.7)
39:        ENDIF
40:        IF(LSLT2(1).EQ.1) GO TO 6900
41:        IF(IOUT_COUNT.EQ.50.OR.IOUT_COUNT.EQ.51.OR.INSEN.EQ.0) THEN
42:        DT_HOLD(1)=T
43:        DT_HOLD(2)=DT
44:        OUT_FLAG=.TRUE.
45:        IOUT_COUNT=0
46:        WRITE(IUNTNN) IPROB,MONTH,KAY,IEAR,T,N,
47:       &              (X(I,3,I_TIME2,I_MYPE),VX(I,3,I_TIME2,I_MYPE),X(I,2,
48:       &I_TIME2,I_MYPE),VX(I,2,I_TIME2,I_MYPE),X(I,1,I_TIME2,I_MYPE),VX(I,
49:       &1,I_TIME2,I_MYPE),I=1,N),DT,ENGKA,ENGPO(I_TIME2,I_MYPE),ENGSP(I_TI
50:       &ME2,I_MYPE),ENGLO(I_TIME2,I_MYPE),ENGTO,DISCR,ENGKE(I_TIME2,I_MYPE
51:       &),EGKA1,EGPO1,EGSP1,EGLO1,EGTO1,DSCR1,INSEN
52:        ENDIF
53:        WRITE(6,5000) T,ENGKA,ENGPO(I_TIME2,I_MYPE),ENGSP(I_TIME2,I_MYPE),
54:       &ENGLO(I_TIME2,I_MYPE),ENGTO,DISCR
55:    5000 FORMAT(1H ,7F15.7)
56:                                .
57:                                .
58:                                .
59:                                .
```

Fig. 4.67 Part of parallelized subroutine GR2D50 of PART2 (2/2)

```
                           .
                           .
      CALL GR2D50
!XOCL BROADCAST(DT_HOLD) (OUT_FLAG)
                           .
                           .
```

Fig. 4.68 BROADCAST transfer

```
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@  NORMAL RESTART  @@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
      J=0
 1000 READ(IUNT18)  IPROB,MONTH,KAY,IEAR,T,N,
     .              (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
     .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
     .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
      J=J+1
      IF(T.LE.4.0D0) GO TO 1000
      IF(LSLT2(1).EQ.1.AND.INSEN.EQ.1)
     1 WRITE(19)    IPROB,MONTH,KAY,IEAR,T,N,
     .              (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
     .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
     .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
      IF(INSEN.EQ.1)          GO TO 1000
      INSEN=1
      IF(LSLT2(1).EQ.1)
     1WRITE(19)     IPROB,MONTH,KAY,IEAR,T,N,
     .              (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
     .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
     .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
      IF(LSLT2(1).EQ.1) RETURN
      BACK SPACE  IUNT18
      WRITE(IUNT18) IPROB,MONTH,KAY,IEAR,T,N,
     .              (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
     .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
     .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC---------------------------------------------------------------CCC
```

Fig. 4.69 Parallelized subroutine GR2D20 of PART2 (1/2)

```
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@  RESTART FROM PARALLEL EXECUTION ERROR  @@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
c      IF(LSLT2(1).EQ.1) GO TO 2000
c      IUNTNN=17
c      WRITE(IUNTNN)    (RMIN(I),RMAX(I),I=1,3)
c      WRITE(IUNTNN)    T,IPROB,MONTH,KAY,IEAR,
c     .                 N,M,DT,E1,E2
c      WRITE(IUNTNN)    ((X_R(I,J),VX_R(I,J),I=1,N),J=1,3)
c      WRITE(IUNTNN)    ((TB(I,J),I=1,N),J=1,3)
c      WRITE(IUNTNN)    ((A(I,J),B(I,J),C(I,J),I=1,M),J=1,3),
c     .                 ((A2(I,J),B2(I,J),C2(I,J),I=1,M),J=1,3)
c      WRITE(IUNTNN)    (IBND(I),I=1,N),((E(I,J),I=1,1000),J=1,3),
c     .                                 ((EE(I,J),I=1,1000),J=1,3),
c     .                 IALP,IBET,IGAM,ENGKE_R,(IK(I),I=1,2735),
c     .                 (IAUX(I),IDUX(I),I=1,27),INSEN
c2000 READ(IUNT18)  IPROB,MONTH,KAY,IEAR,T,N,
c     .                (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
c    .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
c    .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
c      IF(LSLT2(1).EQ.1) THEN
c      WRITE(19)      IPROB,MONTH,KAY,IEAR,T,N,
c     .                (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
c    .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
c    .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
c      WRITE(6,5001) T,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR
c5001 FORMAT(1H ,7F15.7)
c      GO TO 2001
c      ENDIF
c      WRITE(IUNTNN) IPROB,MONTH,KAY,IEAR,T,N,
c     .                (X_R(I,3),VX_R(I,3),X_R(I,2),VX_R(I,2),X_R(I,1),VX_R(
c    .I,1),I=1,N),DT,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR,ENGKE_R,EGK
c    .A1,EGP01,EGSP1,EGL01,EGT01,DSCR1,INSEN
c      WRITE(6,5000) T,ENGKA,ENGPOO,ENGSPO,ENGLO_R,ENGTO,DISCR
c5000 FORMAT(1H ,7F15.7)
c2001 IF(T.LT.0.94D0) GO TO 2000
c      DT=0.03D0
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
CCC ### @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ### CCC
       RETURN
       END
```

Fig. 4.69 Parallelized subroutine GR2D20 of PART2 (2/2)

```
                    .
                    .
COMMON /GEOM2/GEOM2(NWPAR+NPAR+2792)
                    .
EQUIVALENCE  (GEOM2(1),IALP),
.            (GEOM2(2),IBET),
.            (GEOM2(3),IGAM),
.            (GEOM2(4),TB(1,1)),
.            (GEOM2(4+NWPAR),IBND(1)),
.            (GEOM2(4+NWPAR+NPAR),IK(1)),
.            (GEOM2(4+NWPAR+NPAR+2735),IAUX(1)),
.            (GEOM2(4+NWPAR+NPAR+2735+27),IDUX(1))
```

Fig. 4.70 Parallelized common block GEOM2 of PART2

```
                    .
                    .
COMMON /DYNA/DYNA(NWPAR+6495)
                    .
EQUIVALENCE  (DYNA(1),AC(1)),
.            (DYNA(NWPAR+1),E(1)),
.            (DYNA(NWPAR+3003+1),EE(1)),
.            (DYNA(NWPAR+6006+1),E1(1)),
.            (DYNA(NWPAR+6006+4),E2(1)),
.            (DYNA(NWPAR+6006+7),A(1,1)),
.            (DYNA(NWPAR+6006+7+26*3),B(1,1)),
.            (DYNA(NWPAR+6006+7+26*3*2),C(1,1)),
.            (DYNA(NWPAR+6006+7+26*3*3),A2(1,1)),
.            (DYNA(NWPAR+6006+7+26*3*4),B2(1,1)),
.            (DYNA(NWPAR+6006+7+26*3*5),C2(1,1)),
.            (DYNA(NWPAR+6006+7+26*3*6),ENGKA),
.            (DYNA(NWPAR+6006+8+26*3*6),ENGTO),
.            (DYNA(NWPAR+6006+9+26*3*6),DISCR),
.            (DYNA(NWPAR+6006+10+26*3*6),EGKA1),
.            (DYNA(NWPAR+6006+11+26*3*6),EGPO1),
.            (DYNA(NWPAR+6006+12+26*3*6),EGSP1),
.            (DYNA(NWPAR+6006+13+26*3*6),EGLO1),
.            (DYNA(NWPAR+6006+14+26*3*6),EGTO1),
.            (DYNA(NWPAR+6006+15+26*3*6),DSCR1)
.            (DYNA(NWPAR+6006+16+26*3*6),RMIN(1)),
.            (DYNA(NWPAR+6006+19+26*3*6),RMAX(1))
```

Fig. 4.71 Parallelized common block DYNA of PART2

```
 1:        PROGRAM MAIN
 2:        IMPLICIT REAL*8(A-H,O-Z)
 3:        PARAMETER(NPAR=4000)
 4:        PARAMETER(NPE=4)
 5:  !XOCL PROCESSOR P(NPE)
 6:        REAL*8 T,X(NPAR,3),VX(NPAR,3),DT,ENGKA,ENGPO,ENGSP,ENGLO,ENGTO,
 7:       &DISCR,ENGKE,EGKA1,EGPO1,EGSP1,EGLO1,EGTO1,DSCR1
 8:        REAL*8 STIME,ETIME
 9:        LOGICAL FLAG
10:        FLAG=.FALSE.
11:        ITMP=0
12:        WRITE(10) ITMP
13:        REWIND 10
14:  !XOCL PARALLEL REGION
15:        IPROB=0
16:        MONTH=0
17:        KAY=0
18:        IEAR=0
19:        T=0.0D0
20:        N=NPAR
21:        DO 222 JM=1,3
22:        DO 222 IM=1,4000
23:        X(IM,JM)=0.0D0
24:        VX(IM,JM)=0.0D0
25:    222 CONTINUE
26:        DT=0.0D0
27:        ENGKA=0.0D0
28:        ENGPO=0.0D0
29:        ENGSP=0.0D0
30:        ENGLO=0.0D0
31:        ENGTO=0.0D0
32:        DISCR=0.0D0
33:        ENGKE=0.0D0
34:        EGKA1=0.0D0
35:        EGPO1=0.0D0
36:        EGSP1=0.0D0
37:        EGLO1=0.0D0
38:        EGTO1=0.0D0
39:        DSCR1=0.0D0
40:        INSEN=0
```

Fig. 4.72 Test program to measure WRITE processing time (1/2)

5. おわりに

平成6年度末に，VPP500 が導入されて，丸2年が経過した．この間，利用者数も順調に増加し，実行される原子力コードも多数に及んでいる．計算科学技術推進センター情報システム管理課では，ユーザからの依頼に応じて，これらの原子力コードを VPP500 向けに最適なベクトル化，並列化を施すチューニングを行ってきている．この作業は，大規模シミュレーション・ジョブの計算時間を短縮するだけでなく，単一プロセッサ上ではメモリ不足から実行できないジョブを並列化により可能にするなど，計算機のスループットの向上，ユーザの仕事時間の短縮のみならず，計算可能なジョブの範囲を拡大するなど，非常に意義のある仕事と考えている．

本報告書で記述した原子力コードのベクトル並列化については，その効果が顕著に表れているものが多いが，一方では VPP500 のベクトル並列計算機としての能力を十分に引き出せない原子力コードも数多く存在することが分かっている．このようなコードについては，平成9年度から導入された，1プロセッサの処理能力の高いスカラ並列型の AP3000 計算機システムへの移行も現在進めている．原子力コードの性格に応じて，計算機を使い分けていくことが，計算機資源の効率的利用，ユーザの仕事の効率化の観点から重要であると考えている．本報告書が，これらの仕事に関係する人々に多少とも参考になれば幸いである．

謝　辞

本作業を行う上で，作業を依頼された 炉心プラズマ研究部 炉心プラズマ解析室 岸本泰明氏 (2章)，原子炉工学部 伝熱流動研究室 功刀資彰氏 (3章)，先端基礎研究センター 極低温放射線物性研究グループ 田次邑吉氏 (4章) には，コード内容の把握について御協力頂きました．また，VPP500 システムのハードウェア，ソフトウェアに関する相談については (株) 富士通 R&D システム部石附茂氏に，本作業を円滑に遂行するための各種事務処理については山田圭子氏に御協力を頂きました．ここにこれらの方々に感謝の意を表します．

最後に，本報告書を執筆する機会を与えて下さいました計算科学技術推進センター長浅井清氏，前次長秋元正幸氏 (現原子炉安全工学部部長)，(株) 富士通 R&D システム部前部長橋本道夫氏及び現部長平沢健一氏に感謝致します．

5. おわりに

平成6年度末に，VPP500 が導入されて，丸2年が経過した．この間，利用者数も順調に増加し，実行される原子力コードも多数に及んでいる．計算科学技術推進センター情報システム管理課では，ユーザからの依頼に応じて，これらの原子力コードを VPP500 向けに最適なベクトル化，並列化を施すチューニングを行ってきている．この作業は，大規模シミュレーション・ジョブの計算時間を短縮するだけでなく，単一プロセッサ上ではメモリ不足から実行できないジョブを並列化により可能にするなど，計算機のスループットの向上，ユーザの仕事時間の短縮のみならず，計算可能なジョブの範囲を拡大するなど，非常に意義のある仕事と考えている．

本報告書で記述した原子力コードのベクトル並列化については，その効果が顕著に表れているものが多いが，一方では VPP500 のベクトル並列計算機としての能力を十分に引き出せない原子力コードも数多く存在することが分かっている．このようなコードについては，平成9年度から導入された，1プロセッサの処理能力の高いスカラ並列型の AP3000 計算機システムへの移行も現在進めている．原子力コードの性格に応じて，計算機を使い分けていくことが，計算機資源の効率的利用，ユーザの仕事の効率化の観点から重要であると考えている．本報告書が，これらの仕事に関係する人々に多少とも参考になれば幸いである．

謝　辞

本作業を行う上で，作業を依頼された 炉心プラズマ研究部 炉心プラズマ解析室 岸本泰明氏 (2章)，原子炉工学部 伝熱流動研究室 功刀資彰氏 (3章)，先端基礎研究センター 極低温放射線物性研究グループ 田次邑吉氏 (4章) には，コード内容の把握について御協力頂きました．また，VPP500 システムのハードウェア，ソフトウェアに関する相談については (株) 富士通 R&D システム部石附茂氏に，本作業を円滑に遂行するための各種事務処理については山田圭子氏に御協力を頂きました．ここにこれらの方々に感謝の意を表します．

最後に，本報告書を執筆する機会を与えて下さいました計算科学技術推進センター長浅井清氏，前次長秋元正幸氏 (現原子炉安全工学部部長)，(株) 富士通 R&D システム部前部長橋本道夫氏及び現部長平沢健一氏に感謝致します．

参考文献

[1] 「行列計算ソフトウェア WS, スーパーコン, 並列計算機」, 小国 力, 丸善株式会社.

[2] 田次邑吉；ダイヤモンド型と黒鉛型結晶における照射損傷シミュレーションのための分子動力学コード：DGR および GGR, JAERI レポート 1291, 1984 年 6 月.

[3] CALCOMP ソフトウェアマニュアル グラフィック COM プログラミング－ベーシックソフトウェア－, 吉沢ビジネスマシンズ (株), 1972 年 6 月.

[4] 根本俊行, 江口則地, 他；原子力コードのベクトル化と改良 (III), JAERI-Data/Code 94-021, 1995 年 1 月.

[5] 「UXP/M VPP FORTRAN77 EX/VP 使用手引書 V12 用」, 富士通 (株), 1994 年 9 月.

[6] 「UXP/M VPP アナライザ 使用手引書 V10 用」, 富士通 (株), 1994 年 1 月.

[7] 「UXP/M VPP FORTRAN77 EX/VPP 使用手引書 V12 用」, 富士通 (株), 1994 年 1 月.

[8] 「原研 VPP500/42 システム利用手引 第 2 版」, 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, 1995 年 7 月.

[9] 「原研 VPP500 並列化入門」, 日本原子力研究所 計算科学技術推進センター, 1995 年 7 月.