JAERI-Data/Code
99-020

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９年度作業報告書－

1999年3月

川井　渉*・川崎信夫・渡辺秀雄*・根本俊行*・石附　茂*
田邊豪信*・小笠原忍・足立将晶・久米悦雄

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

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

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

編集兼発行　日本原子力研究所

JAERI－Data/Code 99－020

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9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
川井 渉＊・川崎 信夫※・渡辺 秀雄＊・根本 俊行＊・石附 茂＊
田邊 豪信＊＊・小笠原 忍※・足立 将晶※・久米 悦雄

(1999 年 3 月3日受理)

本報告書は，平成9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（一部AP3000）における高速化作業は，平成9年度に14件行われた．これらの作業内容は，今後同種の作業を行う上での参考となりうるよう，作業を大別して，「並列化編」，「ベクトル化編」及び「移植編」の3分冊にまとめた．

本報告書の「並列化編」では，円筒座標系直接数値解析コードCYLDNS44N，放射能粒子拡散予測コードWSPEEDI，拡張量子分子動力学コードEQMD及び三次元熱流体解析コードSTREAMを対象に実施した並列化作業について記述している．別冊の「ベクトル化編」では，多次元二流体モデル構成方程式評価用コードACE-3D，原子核統計崩壊計算コードSD及びHENDEL炉内構造物実証試験部（T2）3次元熱伝導解析コードSSPHEATを対象に実施したベクトル化作業について記述している．また，別冊の「移植編」では，沸騰水型原子炉熱水力解析コードTRAC-BF1，連続エネルギー粒子輸送モンテカルロコードMCNP4AのAP3000への移植作業，及び汎用図形処理解析システムIPLOT用ライブラリの改良作業について記述している．

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
※ 外来研究員：富士通株式会社
＊ 富士通株式会社
＊＊ 株式会社ケーシーエス

JAERI－Data/Code 99－020

Vectorization,Parallelization and Porting of Nuclear Codes on the VPP500 System (Parallelization) - Progress Report Fiscal 1997 -

Wataru KAWAI*, Nobuo KAWASAKI※, Hideo WATANABE*, Toshiyuki NEMOTO*, Shigeru ISHIZUKI*, Hidenobu TANABE**, Shinobu OGASAWARA※, Masaaki ADACHI※ and Etsuo KUME

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

(Received March 3 , 1999)

Several computer codes in the nuclear field have been vectorized, parallelized and transported on the FUJITSU VPP500 system and/or the AP3000 system at Center for Promotion of Computational Science and Engineering in Japan Atomic Energy Research Institute. We dealt with 14 codes in fiscal 1997. 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 cylindrical direct numerical simulation code CYLDNS44N, worldwide version of system for prediction of environmental emergency dose information code WSPEEDI, extension of quantum molecular dynamics code EQMD and three-dimensional non-steady compressible fluid dynamics code STREAM are described. In the vectorization part, the vectorization of multidimensional two-fluid model code ACE-3D for evaluation of constitutive equations, statistical decay code SD and three-dimensional thermal analysis code for in-core test section (T2) of HENDEL SSPHEAT are described. In the porting part, the porting of transient reactor analysis code TRAC-BF1 and Monte Carlo radiation transport code MCNP4A on the AP3000 are described. In addition, a modification of program libraries for command-driven interactive data analysis plotting program IPLOT is described.

Keywords : CYLDNS44N, WSPEEDI, EQMD, STREAM, Parallelization, Vectorization, VPP500, Nuclear Codes

---

※ On leave from FUJITSU, Ltd
* FUJITSU,Ltd
** KCS Corp.

目　　次

Contents

1. Introduction

We have reported the results for the bare target neutronics on the target system of 5 MW spallation Neutron Source proposed by JAERI[1]. As pointed out in the report, optimizing target material and shape further needs indispensably, calculations on the full target-moderator-reflector systems. Calculations are essential to maximize the neutronic performance of slow neutrons from the moderators.

In the present study, we calculated energy spectral intensities of slow neutrons and time distributions from various moderators and energy deposition in cryogenic moderators, for full target-moderator-reflector systems.

The main purpose of the present investigation is to obtain the neutronic performance (slow neutron intensity and pulse shape) of the reference target-moderator-reflector system[2]. We compare the performance of the present results with those obtained at other laboratories or projects in order to evaluate and justify our reference system.

It is another important issue for the target engineering to investigate the effect of the target shape as well as the proton beam footprint on the time-averaged-slow-neutron intensity. At higher beam power target size becomes large compare to an ideal one due to the maximum acceptable beam power density and safety reason (triple target vessel, fourfold cryogenic moderator vessel, etc.). It may lead to a poor neutronic performance. We performed some calculations to investigate such effects and report the results.

2. Model of reference target-moderator-reflector system

A target-moderator configuration is illustrated in Fig. 1. The coupling scheme of the target and the moderator is a wing geometry type. The configuration of the target-moderator-reflector system is illustrated in Fig. 2.

Table 1 shows the proton beam, target and reflector parameters. The combinations of the main parameters of the proton beam, the target and the reflector examined in this study are summarized in table 2. The proton-current-density profile is assumed to be a rectangular (flat distribution) with a maximum acceptable density of 48 $\mu A/cm^2$ at beam power of 5 MW. A rectangular target with a hemispherical shape beam window is adopted to obtain a good neutronic performance. Lateral target dimensions are assumed to be beam height plus 3 cm in vertical direction and beam width plus 4 cm in horizontal direction in the reference system. The target material is contained by a double stainless steel vessel (SUS316). It is filled with water between double SUS316 structures.

Two coupled hydrogen (liquid or supercritical at 20 K) moderators with water premoderators (PM) at ambient temperature (hereafter we call coupled H_2 moderator) are located above the target to realize the highest peak intensity together with the highest time-integrated intensity.

作業では，圧縮性流体の解析部分のみに着目し，MPIによる並列化を行うとともに，大規模計算の対応のため，解析空間をプロセッサ台数個に分割させた．また，前処理付き反復解法による行列ソルバ部分については，パイプライン的な並列化を実施した．

なお，本報告書の2章及び5章の作業は渡辺が，3章の作業は川井が，4章の作業は川崎が担当した．

2. CYLDNS44N の並列化

2.1 概要

本作業では，円筒座標系直接数値解析コード CYLDNS44N の富士通製スーパーコンピュータ VPP500 [1,2] 向き並列化作業を行った．この CYLDNS44N 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 CYLDNS [1] コードのドライバ部として開発されたコードであり，本並列化の後，すでに並列化されている CYLDNS コードと結合される予定となっている．この CYLDNS コードは，円筒座標系を用いた運動方程式及び，圧力のポアソン式を中心差分を用いて離散化し，各方程式での円周方向について FFT 計算を行ったあと，各波数断面について 2 次元のポアソン方程式を解くものである．CYLDNS コードが，データ分割によって大規模な計算を可能とすることを主な目的として並列化されている事から，CYLDNS44N コードも大規模計算への対応を主な目的として並列化を行った．また，計算速度の向上についても考慮し，特にベクトル計算ができていなかった部分のベクトル化も行った．

2.2 前作業

本コードは，配列の要素数などを定義するためのパラメータ文が個々のルーチン中に直接記述されている．このようなプログラムは，計算の規模を変更するような場合に個々のルーチンのソースプログラムを編集する必要があり，非常に煩わしく，またミスも犯しやすい．そこで，ユーザの希望により，これらのパラメータ文を一つのインクルードファイルに記述しておき，個々のルーチンのソースプログラムには，このインクルードファイルを読み込むためのインクルード行を挿入しておく作業を行った．この作業で作成したインクルードファイル **PARAM** の内容を Fig. 2.1 に，このインクルードファイルの内容を取り込むために各ルーチンに挿入したインクルード行の例を Fig. 2.2 に示す．なお，Fig. 2.1 に示したインクルードファイルには，並列計算で使用するプロセッサ台数を定義するパラメータ文が例外として含まれている．

2.3 並列化

2.3.1 並列化の概要

本コードの主な処理内容として，J 軸方向を TDMA を使用したソルバで解いていること，及び I 軸方向の FFT 計算を行っていることが挙げられる．前者の計算を行っている部分では，I 軸方向と K 軸方向に並列性が存在しているが，I 軸方向は後でベクトル計算で利用することにして，K 軸方向の並列性を並列化で利用することにした．他方，後者の計算を行っている部分では，J 軸方向と K 軸方向に並列性がある．しかし，K 軸方向は既にベクトル計算で使用されているため，J 軸方向の並列性を利用して並列化することにした．また，その他の計算部分は，基本的にどの軸にも並列性があるが，I 軸方向はベクトル計算で用いられているため，並列化には J 軸または K 軸方向の並列性を利用することができる．本作業では，その他の計算部分の

分割軸としてJ軸方向を選択することにした．これは，本コードの1タイムステップあたり，J軸方向をTDMAを使用したソルバで解く処理が1度しか行われないのに対して，FFT計算は複数回実行されるためである．すなわち，FFT計算と同じ分割軸を選んでおけば，分割軸の変更に伴うデータ転送回数が少なくて済むためである．以上のことから，並列化後の本コードではFig. 2.3のように処理が行われる．

2.3.2 データの分割

データの分割はFig. 2.4のように行った．図中の`!xocl`で始まる行が，並列化のための指示行である．(a)の指示行は，計算で使用するプロセッサ台数とプロセッサの形状を宣言している．本コードの場合は特殊なプロセッサ形状を利用していないため，単純に何台のプロセッサを使用するのかを宣言しているものと考えてよい．(b)の指示行は，(a)で定義したプロセッサのうちのどのプロセッサを利用するかを定義している．本コードの場合は特にプロセッサを限定した処理は必要ないため，全てのプロセッサを使用するように宣言している．(c) ～ (g)の指示行は，インデックスパーティションの定義を行っている．このインデックスパーティションは，`DO`ループの回転数や配列の寸法を分割し，それぞれを各プロセッサに割り当てる目的で使用される．(c)で定義した`ipmj`はJ軸方向のデータの分割，及びJ軸方向の処理の分割で使用する．(d) ～ (g)で定義したインデックスパーティションは，同様にJ軸方向のデータの分割で使用するが，単純に分割するのではなく袖付きで分割するために使用する．袖については「2.3.2.2 袖」に示す．(i)，(k)の指示行では，個々の配列をグローバル配列として宣言している．また，(h)，(j)の指示行では，個々の配列を分割ローカル配列として宣言していると同時に，個々の配列の2次元目（J軸方向）の分割を行っている．グローバル配列と分割ローカル配列については「2.3.2.1 グローバル配列とローカル配列」に示す．

付録Aに分割した配列の一覧を示す．

2.3.2.1 グローバル配列と分割ローカル配列

VPP500は，分散メモリ型の並列計算機であり，一般にあるプロセッサから他のプロセッサ上のデータを直接アクセスすることはできない．しかし，変数や配列をグローバルとして宣言しておくと，変数や配列を他のプロセッサと共有することが可能となる．ただし，アクセスに要する時間が比較的大きいため，通常の計算では用いず，プロセッサ間のデータ転送やファイル入出力の際に利用される．分割ローカル配列は，複数のプロセッサが分割して領域を保持している配列である．グローバル配列と異なり，各プロセッサは自分が担当している領域のみを直接アクセスすることができる．プログラム中の主要な配列を分割ローカル配列とすることで，各プロセッサが保持すべきデータ量を削減することが可能となり，大規模な計算に対応できるようになる．なお，グローバル配列と分割ローカル配列は`EQUIVALENCE`文で結合することが可能で，高速なアクセスが可能な分割ローカル配列としての性質を通常の計算で利用し，またファイル入出力などの際にはグローバル配列としての性質を利用することができる．

その他として，重複ローカル変数，重複ローカル配列と呼ばれるデータが存在し，これらは各プロセッサに固有で，かつ分割されていないものを指す．

2.3.2.2 袖

本コードは有限差分法によって離散化されており，前後左右のメッシュ点の参照が非常に多く行われている．このような参照が行われる配列を分割すると，分割面の近傍で他プロセッサが保持する領域のデータを参照する必要が発生する．そこで，VPP500 の並列コンパイラが持つ「袖」の機能を利用することにした．袖とは，配列を分割した時に付加される余分なデータ領域を指す（Fig. 2.5 参照）．また，コンパイラは，配列名を指定しておくとその配列の袖部分を一括して埋めてくれる機能（以下，袖転送）も提供する．この「袖」と「袖転送」を利用すると，上記のような参照を含む計算を容易に処理できる．

2.3.3 処理の分割

処理の分割は，DO ループの前後を指示行「!xocl spread do /XXX」と「!xocl end spread」で囲むことで行った（Fig. 2.6 参照）．ここで，XXX の部分には，「2.3.2 データの分割」で述べたインデックスパーティションを指定する．これらの指示行は，囲まれた DO ループの回転数をインデックスパーティションにしたがって分割し，個々のプロセッサに割り当てることを示す．この時，ループ中で使用される配列とループそのものの分割を一致させておくことにより，プロセッサ間のデータ転送を極力抑えた並列化ができる．本コードの並列化も，データの分割と処理の分割を一致させるように処理の分割を行った．さらに，「!xocl spread do」を指定した DO ループ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ため，可能な限り DO ループの入れ換えを行って，分割した DO ループができるだけ外側のループとなるようにすることで，オーバヘッド時間を小さく抑えるようにした．

2.3.3.1 境界計算の並列化

境界計算部分の並列化も，データの分割と処理の分割が一致するように行ったが，特別な手法が必要となるため Fig. 2.7 (a) に示した境界計算部分を例としてその方法を示す．

この図でアクセスされる配列は，J 軸方向が分割の対象となっている．したがって，DO 88 と DO 8888 のループは他の通常の計算部分と同様に処理の分割を行えばよい．しかし，DO 888 ループは，J 軸方向両端の境界計算を行っているため，境界部分の領域を担当する特定のプロセッサのみに処理を行わせる必要がある．そこで，Fig. 2.7 (b) のように並列化を行った．図中の左矢印で示した DO ループと IF 文により，特定のプロセッサだけが処理を行う．また，この例でアクセスしている配列は J 軸方向が分割されており，J 番目の要素と J-1 番目の要素，及び J 番目の要素と J+1 番目の要素をそれぞれ同一のプロセッサが担当しているとは限らないため，通常の代入文ではなく，プロセッサ間のデータ転送によって処理を行うようにした．プロセッサ間のデータ転送については「2.3.4 プロセッサ間のデータ転送」に示す．

2.3.4 プロセッサ間のデータ転送

本コードの場合，プロセッサ間のデータ転送は，袖転送，境界計算部分，及び分割軸の変更部分で必要となる．以下では，これらの転送の概要について示す．

2.3.4.1 袖転送

袖転送は，指示行「**!xocl overlapfix (配列名の並び) (データ転送識別名)**」及び「**!xocl movewait (データ転送識別名)**」の組をソースプログラム中に挿入しておくことで実行される．前者の指示行によって指定された配列の袖転送が開始され，後者の指示行によって袖転送が完了するまで待つ処理が行われる．本作業では，付録 B に示した位置に袖転送のための指示行を挿入した．

2.3.4.2 境界計算に伴うデータ転送

本コードでは，Fig. 2.7 に示した形式の境界計算が数箇所で行われている．「2.3.3.1 境界計算の並列化」で述べたように，この計算でアクセスされる配列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，**spread move** によるプロセッサ間のデータ転送を利用することにした．

spread move によるデータ転送は次のように記述する．まず転送元の配列から転送先の配列にデータをコピーする DO ループを記述して，ループの前後を「**!xocl spread move**」と「**!xocl end spread** (データ転送識別名)」で囲む．ここで，コンパイラの仕様により，ループ中の代入文はローカル配列からグローバル配列への代入，またはグローバル配列からローカル配列への代入の形で記述する．また，代入文の右辺には**計算式**を記述することができないため，必要であればあらかじめ計算を行って結果を一時配列などに格納しておき，この一時配列を用いてデータ転送を行う必要がある．次に，転送の完了を確認するための指示行「**!xocl movewait (データ転送識別名)**」を挿入する．

2.3.4.3 分割軸の変更に伴うデータ転送

分割軸の変更は，実際には分割軸の相異なる配列間でデータ転送を行うことによって実現する．本コードにおいて分割軸の変更処理を行っている部分を Fig. 2.8 に示す．この図以前の処理は J 軸方向を分割軸として行われ，(a) で J 軸方向を分割した配列から K 軸方向を分割した配列にデータがコピーされる．その後 K 軸方向を分割した配列を使用して処理を行い，(b) で J 軸方向を分割した配列にデータを戻す処理が行われる．これ以降，J 軸方向を分割軸として処理が続けられる．

2.3.5 その他

2.3.5.1 MAIN プログラムに固有な変更内容

VPP500 の並列コンパイラの仕様では，**spread do** などによる並列実行部分は，指示行「**!xocl parallel region**」が挿入されている位置以降で，かつ「**!xocl end parallel**」が挿入されている位置以前でなければならない．本コードでは，これら 2 つの指示行を **MAIN** プログラム中に挿入した（Fig. 2.9 参照）．また，これらの指示行を挿入したルーチンには **subprocessor** 文を記述することができないため，Fig. 2.9 に示すように **proc alias** 文で別名を付け，**index partition** 文で参照できるようにした．

2.3.5.2 インライン展開

本コードには，処理分割の対象となる DO ループ中からサブルーチンを呼び出しており，このサブルーチン中で分割された配列のアクセスが行われる部分が存在する．ところが，VPP500 のコンパイラの仕様では，処理分割したループから他のサブルーチンを呼び出して，このサブルーチン中で分割した配列をアクセスすることを禁止しているため，何らかの変更が必要となる．そこで本作業では，呼び出される側のサブルーチン中の処理（分割した配列のアクセス部分）を，呼び出す側の DO ループ中に移動する作業を行った．この作業の対象となった箇所を Table 2.1 に示す．また，この作業の例を Fig. 2.10 に示す．

なお，この作業の結果，５つのサブルーチン DFTNVH，DFTRH，DFTSVH，DFTSH，及び SATSPH はプログラム中で使用されなくなったため，全て削除した．

2.4 ベクトル化

本コードには，ベクトル化の不十分な箇所が２ヶ所含まれていたため，これらの箇所のベクトル化作業を行った．以下，ベクトル化の内容について示す．

2.4.1 サブルーチン DPE4R

このサブルーチンの DO 915 ループでは，Fig. 2.11 に示す処理が行われている．ここでは，三重対角行列を係数行列とする連立一次方程式を作成して，TDMA と呼ばれる方法で解く処理を繰り返し行っている．この TDMA という方法には回帰計算が含まれているため，ベクトル計算することができない．しかし，個々の連立一次方程式の間に依存関係がないため，一つずつ方程式を生成して解くことを繰り返すのではなく，全ての方程式を一度に生成して一度に解くような形に変更すれば，ベクトル計算が可能となる．

変更後のプログラムを Fig. 2.12 に示す．変更前では連立一次方程式の係数行列などを格納するのに一次元配列を用いていたが，これらを全て二次元配列に変更して使用している．また，変更前では最も外側に位置していたループが，変更後では最も内側のループとなっており，このループでベクトル計算が行われる．

2.4.2 サブルーチン PSAT23C

このサブルーチンには，非常に大きな DO 1000 ループが含まれている．この DO ループにはベクトル化を阻害する要因が２点含まれていたため，これらの要因を取り除いて DO 1000 ループ中の処理が全てベクトル計算できるようにした．以下，これらの要因の概要と回避方法を示す．

2.4.2.1 ベクトル化を阻害する要因と回避・その１

DO 1000 ループには，Fig. 2.13 に示すように，１つの変数に値を加える処理を数回に分けて行う処理が含まれていた．このような処理は，Fig. 2.14 に示すように単純な DO ループであればベクトル化を阻害することはないが，本コードの場合は DO ループ中の処理が多いためにコンパイラが対処しきれず，ベクトル化されなかったのではないかと思われる．

そこで，Fig. 2.15 に示すように，それぞれの処理を相異なる変数に対して行い，DO ループの終了後に 1 つの変数にまとめるように処理を変更した．その結果，これらの処理がベクトル化を阻害することはなくなった．この作業の対象とした変数名の一覧を Table 2.2 に示す．

2.4.2.2　ベクトル化を阻害する要因と回避・その 2

DO 1000 ループには，Fig. 2.16 に示す処理が含まれていた．この処理が問題となるのは，変数 AFB，AFC が参照された後に値を代入されている点である．

本コードの場合，このループ中で配列 EWHEI の内容は不変であり，かつ配列 RDRR には DRR の各要素の逆数を計算したものが格納されている．したがって，(*1)，(*2)，(*3) は全く同じ計算をしていることになるため，(*2) と (*3) の処理を削除しておくことにした．この結果，変数 AFB，AFC についての定義・引用関係が改善され，この DO ループ中の処理全てがベクトル計算できるようになった．

2.5　効果

次の 3 通りのソースプログラムを用いた場合の計算時間を Table 2.3に示す．

- 作業前のソースプログラム
- 並列化後（ベクトル化前）のソースプログラム
- 並列化及びベクトル化後のソースプログラム

ここで，時間ステップ数を 4000 （NDRAW=20，NFIN=3980）とし，Table 2.4 に示すコンパイルオプションを指定した．

次に，Table 2.3 (b)，(c) に示した real 時間から求めた並列化効率を Table 2.5 に示す．ここで，並列化効率は式 (2.1)，(2.2) を用いて計算した．

$$\text{並列化効率（\%）} = \text{並列化による速度向上比}/\text{プロセッサ台数} \times 100 \tag{2.1}$$

$$\text{並列化による速度向上比} = \frac{\text{1 プロセッサでの計算時間}}{\text{並列実行時の計算時間}} \tag{2.2}$$

Table 2.3 からわかるように，スカラ実行した場合とベクトル実行した場合を比較すると，スカラ実行した時の方が良好な並列化効率が得られている．これは，ベクトル実行すると実際の計算部分に費やされる処理時間が短くなり，相対的に並列実行のためのオーバヘッド時間が大きくなってしまうためである．

2.6　まとめ

本作業では，円筒座標系直接数値解析コード CYLDNS44N の VPP500 向き並列化及びベクトル化を行った．並列化にあたっては，主要な配列の全てをデータ分割の対象とすることで数台分のプロセッサのメモリを必要とする規模の計算への対応を図った．また，台数効果による計算速度の向上を図った．さらに，ベクトル化については，元々ベクトル化の不十分な 2 ヶ所を対象として行い，計算速度の向上を図った．

本作業の結果，並列化及びベクトル化を行ったコードを 16 プロセッサでベクトル実行した場合に，作業前のコードをベクトル実行した時と比較して 9.98 倍の速度向上が得られた．

Table 2.1 Subroutines to be folded into above.

呼び出し側	呼ばれる側
DPE4R	DFTNVH, DFTRH
DPEU4S	DFTSVH, DFTSH
DPEV4S	DFTSVH, DFTSH
DPEW4S	DFTSVH, DFTSH
PSAT23	SATPWH, SATSPH

Table 2.2 Variables which are renamed for vectorization.

E11,	E12,	E13,	E22,	E23,
E33,	EKK,	DFKK,	PDF13,	PR012,
PSTKK,	TUKK,	VPGKK		

```
      PARAMETER(NPE=4)                    ←プロセッサ台数の定義
c      PARAMETER(NPE=16)
c
c$$      PARAMETER(J1=16,J3=192)
c$$      PARAMETER(J1=16,M2=96)
c$$      PARAMETER(K1=16,K3=192)
c$$      PARAMETER(JH1=J1/2)
c$$      PARAMETER(JH1=J1/2,JH3=J3/2)
c$$      PARAMETER(JH1=K1/2,JH3=K3/2)
c$$      PARAMETER(JH3=J3/2)
c$$      PARAMETER(JH3=JJ3/2)
c$$      PARAMETER(JJ1=17,JJ3=192)
cxx      PARAMETER(J1=16,J3=192,M2=96,K1=J1,K3=J3)
      PARAMETER(J1=16,J3=192,K1=J1,K3=J3)
      PARAMETER(JH1=J1/2,JH3=J3/2)
      PARAMETER(JJ1=17,JJ3=J3)
c$$      PARAMETER(KK1=17,KK3=192)
c$$      PARAMETER(KK1=17,KK3=192,MMMAX=49)
      PARAMETER(KK1=17,KK3=192,MMMAX=49)
c$$      PARAMETER(N1=64,N3=64)
c$$      PARAMETER(NH1=N1/2,NH3=N3/2)
c$$      PARAMETER(NDH1=2*J1+JH1+1)
c$$      PARAMETER(NDH1=2*N1+NH1+1)
      PARAMETER(N1=J1,N3=64,NH1=JH1,NH3=N3/2)
      PARAMETER(NDH1=2*N1+NH1+1)
c$$      PARAMETER(NIG=192,NJG=223)
c$$      PARAMETER(NIG=195,NJG=223)
c$$      PARAMETER(NIG=195,NJG=223,NKG=19)
      PARAMETER(NIG=195,NJG=223,NKG=19)
c
      PARAMETER(JH11=JH1-1)
      PARAMETER(J11=JJ1-1,J33=JJ3-1,JH33=JJ3/2-1)
      PARAMETER(JS=20)
      PARAMETER(KH3=KK3/2)
      PARAMETER(K11=KK1-1,K33=KK3-1,KH33=KK3/2-1)
      PARAMETER(KNKR=27)
      PARAMETER(M1=64,M3=64)
      PARAMETER(MIG=195,MJG=44,MKG=19)
      PARAMETER(MM1=65,MM3=64)
      PARAMETER(MH3=MM3/2)
      PARAMETER(NHS=J3/2-1)
      PARAMETER(KS=NKG-3)
      PARAMETER(NKR=27,NKS=22,NKM=5)
      PARAMETER(NN1=65,NN3=64)
      PARAMETER(N11=NN1-1,N33=NN3-1,NH33=NN3/2-1)
      PARAMETER(NPN=20)
```

Fig. 2.1 Include file 'PARAM' .

```
        :
C***********************************************************************
      PROGRAM  CYLDNS
      IMPLICIT REAL*8(A-H,O-Z)
c$$      PARAMETER(NIG=195,NJG=223,NKG=19)
c$$      PARAMETER(MIG=195,MJG=44,MKG=19)
c$$      PARAMETER(J1=16,J3=192)
c$$      PARAMETER(JJ1=17,JJ3=192)
c$$      PARAMETER(J11=JJ1-1,J33=JJ3-1,JH33=JJ3/2-1)
c$$      PARAMETER(JH1=J1/2,JH3=J3/2)
c$$      PARAMETER(K1=16,K3=192)
c$$      PARAMETER(KK1=17,KK3=192)
c$$      PARAMETER(KH3=KK3/2)
c$$C
c$$      PARAMETER(KS=NKG-3)
c$$      PARAMETER(JS=20)
c$$      PARAMETER(N1=64,N3=64)
c$$      PARAMETER(NN1=65,NN3=64)
c$$      PARAMETER(N11=NN1-1,N33=NN3-1,NH33=NN3/2-1)
c$$      PARAMETER(NH1=N1/2,NH3=N3/2)
c$$      PARAMETER(NDH1=2*N1+NH1+1)
c$$      PARAMETER(M1=64,M3=64)
c$$      PARAMETER(MM1=65,MM3=64)
c$$      PARAMETER(MH3=MM3/2)
c$$      PARAMETER(NPN=20)
*INCLUDE PARAM                              ←インクルード行
      COMMON /DGRID/ID,JD,KD,DRE
      COMMON /HIFIX/IFIX1(20),IFIX3(20)
      COMMON /HNFIX/NFIX1,NFIX3,MJH3
      COMMON /HTRKG/TRKG1(2,JJ1),TRKG3(2,JJ3)
        :
```

Fig. 2.2 Example of 'INCLUDE line' for including 'PARAMP'.

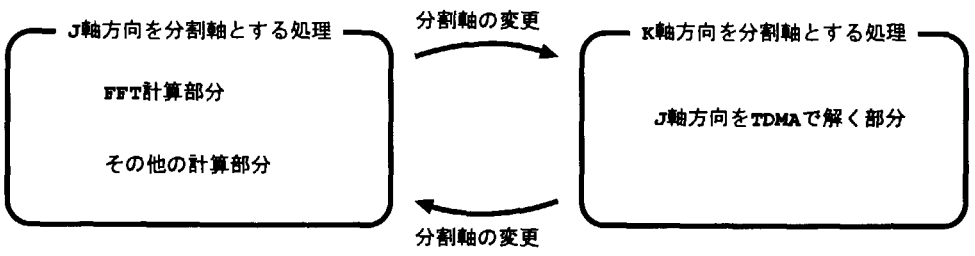


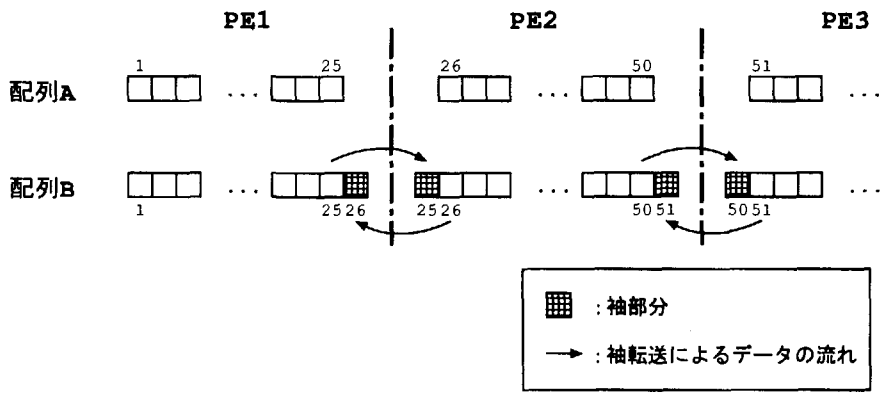
Fig. 2.3 Outline of parallelized code.

```
        :
      PARAMETER(NPE=4)
C
!xocl processor pes(NPE)                                    ←(a)
!xocl subprocessor spe=pes                                  ←(b)
C
!xocl index partition ipmj=(spe,index=-3:MJG,part=band)     ←(c)
C
!xocl index partition ipmj11=ipmj(overlap=(1,1))            ←(d)
!xocl index partition ipmj10=ipmj(overlap=(1,0))            ←(e)
!xocl index partition ipmj01=ipmj(overlap=(0,1))            ←(f)
!xocl index partition ipmj21=ipmj(overlap=(2,1))            ←(g)
C
      COMMON /DEW0/V_G(-3:MIG,-3:MJG,-3:MKG,3),VP_G(0:MIG,0:MJG,0:MKG)
      DIMENSION V(-3:MIG,-3:MJG,-3:MKG,3),VP(0:MIG,0:MJG,0:MKG)
      EQUIVALENCE (V_G,V), (VP_G,VP)
!xocl local  V  (:,/ipmj21,:,:)                             ←(h)
!xocl global V_G                                            ←(i)
!xocl local  VP  (:,/ipmj11,:)                              ←(j)
!xocl global VP_G                                           ←(k)
C
        :
```

Fig. 2.4 Example of the way to data decomposition.

```
        :
      PARAMETER(NPE=4,N=100)
!xocl processor pes(NPE)
!xocl subprocessor spe=pes
!xocl index partition ia=(spe,index=1:N,part=band)
!xocl index partition ib=(spe,index=1:N,part=band,overlap=(1,1))
      DIMENSION A(N), B(N)
!xocl local A(/ia), B(/ib)
        :
```

(a) 配列 A, B の分割宣言



(b) 分割の様子

Fig. 2.5 Example of data decomposition with or without overlaps.

Table 2.3 Execution time (second). (S: scalar mode,V: vector mode)

(a) 作業前

		1PE	4PE	8PE	16PE
S	real	10220.21	---	---	---
	user	10188.73	---	---	---
	sys	1.79	---	---	---
V	real	1665.64	---	---	---
	user	1635.12	---	---	---
	sys	0.74	---	---	---
	vu-user	1080.21	---	---	---

(b) 並列化後（ベクトル化前）

		1PE	4PE	8PE	16PE
S	real	10144.38	2952.89	1505.10	859.47
	user	10089.18	11765.39	11953.42	13246.26
	sys	1.98	3.28	3.35	3.56
V	real	1655.37	560.04	314.33	209.17
	user	1628.06	2187.29	2451.43	3113.29
	sys	0.73	2.08	2.19	2.48
	vu-user	1081.55	1088.14	1092.06	1078.62

(c) 並列化後（ベクトル化後）

		1PE	4PE	8PE	16PE
S	real	9882.55	2926.61	1509.25	804.70
	user	9844.33	11624.56	11922.52	12692.29
	sys	1.99	3.22	3.25	3.54
V	real	1341.49	469.61	264.09	166.83
	user	1323.00	1821.79	2053.91	2569.83
	sys	0.69	2.01	2.12	2.40
	vu-user	1001.53	1011.43	1019.20	1002.32

Table 2.4 Compiler options.

スカラ実行	1PE	-Wv,-sc -Oe
	4PE, 8PE, 16PE	-Wx -Wv,-sc -Oe
ベクトル実行	1PE	-Oe
	4PE, 8PE, 16PE	-Wx -Oe

Table 2.5 Parallel efficiencies (%). (S: scalar mode,V: vector mode)

(a) 並列化後（ベクトル化前）

	4PE	8PE	16PE
S	85.89	84.25	73.77
V	73.90	65.83	49.46

(b) 並列化後（ベクトル化後）

	4PE	8PE	16PE
S	84.42	81.85	76.76
V	71.42	63.50	50.26

```
        :
      DO 100 I=1,ID
      DO 100 J=1,JD
      DO 100 K=1,KD
  100  V(I,J,K,1)=VT(I,J,K,1)
     &-DT*(FAVP(I+1,J,K)-FAVP(I,J,K))*RDSI/DBE*2.0D0/RR(J)
        :
```

(a) 並列化前

```
        :
!xocl spread do /ipmj                                    ←挿入
      DO 100 J=1,JD
      DO 100 I=1,ID
CP      DO 100 J=1,JD                                    ←移動
      DO 100 K=1,KD
  100  V(I,J,K,1)=VT(I,J,K,1)
     &-DT*(FAVP(I+1,J,K)-FAVP(I,J,K))*RDSI/DBE*2.0D0/RR(J)
!xocl end spread                                         ←挿入
        :
```

(b) 並列化後

Fig. 2.6 Example of the way to procedure decomposition.

```
      :
      DO 88 J=1,JD
      DO 88 K=1,KD
      FAVP(0,J,K)=FAVP(ID,J,K)
   88 FAVP(ID+1,J,K)=FAVP(1,J,K)
      DO 888 I=1,ID
      DO 888 K=1,KD
      FAVP(I,JD+1,K)=FAVP(I,JD,K)
  888 FAVP(I,0,K)=FAVP(I,1,K)
      DO 8888 I=1,ID
      DO 8888 J=1,JD
      FAVP(I,J,KD+1)=FAVP(I,J,1)
 8888 FAVP(I,J,0)=FAVP(I,J,KD)
      :
```

(a) 並列化前

```
      :
!xocl spread do /ipmj
      DO 88 J=1,JD
      DO 88 K=1,KD
      FAVP(0,J,K)=FAVP(ID,J,K)
   88 FAVP(ID+1,J,K)=FAVP(1,J,K)
!xocl end spread
C
!xocl spread do /ipmj                              ←
      DO 888 J=1,JD,JD-1                           ←
         IF ( J.EQ.1 ) THEN                        ←
!xocl spread move
            DO 8883 K=1,KD
            DO 8883 I=1,ID
               FAVP_G(I,J-1,K)=FAVP(I,J,K)
 8883       CONTINUE
!xocl end spread (mw_8883)
!xocl movewait (mw_8883)
         ENDIF                                     ←
         IF ( J.EQ.JD ) THEN                       ←
!xocl spread move
            DO 8886 K=1,KD
            DO 8886 I=1,ID
               FAVP_G(I,J+1,K)=FAVP(I,J,K)
 8886       CONTINUE
!xocl end spread (mw_8886)
!xocl movewait (mw_8886)
         ENDIF                                     ←
 888  CONTINUE
!xocl end spread
C
!xocl spread do /ipmj
      DO 8888 J=1,JD
      DO 8888 I=1,ID
CP      DO 8888 J=1,JD
      FAVP(I,J,KD+1)=FAVP(I,J,1)
 8888 FAVP(I,J,0)=FAVP(I,J,KD)
!xocl end spread
      :
```

(b) 並列化後

Fig. 2.7 Example of border calculation which is parallelized.

```
        :
       (J軸を分割軸とした処理)
        :
!xocl spread move /ipmj                    +
      DO 9000 J=0,MJG                      |
      DO 9000 K=0,MKG                      |
      DO 9000 I=0,MIG                      |
         WKDQD_G(I,J,K) = DQD(I,J,K)       (a)
         WKDQ2_G(I,J,K) = DQ2(I,J,K)       |
 9000 CONTINUE                             |
!xocl end spread (mw_9000)                 |
!xocl movewait (mw_9000)                   +
        :
       (K軸を分割軸とした処理)
        :
!xocl spread move /ipmk                    +
      DO 9900 K=0,MKG                      |
      DO 9900 J=0,MJG                      |
      DO 9900 I=0,MIG                      |
         DP2_G(I,J,K) = WKDP2(I,J,K)       (b)
         DVP_G(I,J,K) = WKDVP(I,J,K)       |
 9900 CONTINUE                             |
!xocl end spread (mw_9900)                 |
!xocl movewait (mw_9900)                   +
        :
       (J軸を分割軸とした処理)
        :
```

Fig. 2.8 Data transfer.

```
        :
      PROGRAM  CYLDNS
        :
      PARAMETER(NPE=4)
        :
!xocl processor pes(NPE)
!xocl proc alias spe=pes                          ←
C
        :
         NPNT(19)=38
         NPNT(20)=40

!xocl parallel region                             ←
C
C       CALL SEDDRS(ALP,NDRAW,NFIN,CPRM,CMTD,NEND,DELTA
C     &            ,RR1,RR2,R1R2,NN,TIME,ZL,SHITA,RD,RU,UTAWD,UTAWU)
      CALL SEDDRI(ALP,NDRAW,NFIN,CPRM,CMTD,NEND,DELTA
     &            ,RR1,RR2,R1R2,NN,TIME,ZL,SHITA,RD,RU,UTAWD,UTAWU)
        :
CC
       GOTO 1
CP    2 STOP
 2    CONTINUE
!xocl end parallel                                 ←
      STOP
      END
```

Fig. 2.9 Insertion of directives to MAIN program.

```
      :
C<<                        ↓サブルーチン DFTNVH 中のコモン宣言
      COMMON /DKHYZ/XZHK1(KK1,KH3),XZHM2(KK1,KH3)                +
      COMMON /DKMYZ/XZDK1(KK1,KK3),XZDM2(KK1,KK3),               |
     1              XZDK3(KK1,KK3),XZDM4(KK1,KK3),               |
     1              XZDM5(KK1,KK3),XZDM6(KK1,KK3)                |
      COMMON /HIFIX/IFIX1(20),IFIX3(20)                          |
      COMMON /HNFIX/NFIX1,NFIX3,MJH3                             |
      COMMON /HTRKG/TRKG1(2,JJ1),TRKG3(2,JJ3)                    +
C>>
      :
!xocl spread do /ipmj
      DO 8  J=1,JD
C
      DO 5 K=1,J1
      DO 5 I=1,J3/2
      UHZ1(K,I)=QD(I      ,J,K)
      UHZ2(K,I)=QD(I+ID/2,J,K)
    5 CONTINUE
C
      JJ=J
C<<
C      CALL DFTNVH(JJ)  ←コメント
                        ↓サブルーチン DFTNVH 中の処理
      L1=J1                                                      +
      L3=J3                                                      |
C-------------------------------------------------------------   |
C        INVERSE FOURIER TRANSFORM IN X-XZDM2RECTION             |
C                                                                |
      CALL FFTR2(K1,K3,UHZ1,UHZ2,XZHK1,XZHM2,NFIX3               |
     &,IFIX3,TRKG3,-1,KK1,KH33)                                  |
C                                                                |
      DO 2222 K=0,J11                                            |
      UHZ2(K+1,0+1)=0.0                                          |
 2222 CONTINUE                                                   |
C                                                                |
      CALL FFT1(JH3,K1,UHZ1,UHZ2,XZHK1,XZHM2,NFIX1               |
     &,IFIX1,TRKG1,-1,K11,KH3)                                   |
C-------------------------------------------------------------   |
C        CALCULATION IN SPECTRAL SPACE                           |
      DO 222 K=0,J11                                             |
      DO 222 I=0,JH3                                             |
      DQD(I,JJ,K )=UHZ1(K+1,I+1)                                 |
      DQ2(I,JJ,K)=UHZ2(K+1,I+1)                                  |
  222 CONTINUE                                                   +
C>>
C
    8 CONTINUE
!xocl end spread
      :
```

Fig. 2.10 Example of folding into above (DPE4R).

```
          :
   !xocl spread do /ipmk
s         DO 915 NU=1,KD-1
s         DO 915 MU=1,ID/2-1
   C*-------- MU=NU= NOT ZERO ----------------------
v         DO 920 J=1,JD
v         C(J)=DRR(J+1)*R(J-1)/DRR(J)/R(J)
v         B(J)=1.0+C(J)+DR(J)*DRR(J+1)*RR(J)/R(J)
         &*(RMV(MNPV,MU)/(RR(J))**2+RNV(MNPV,NU))
v         D(J)=-1.*RR(J)/R(J)*DR(J)*DRR(J+1)*WKDQD(MU,J,NU)
v         PD(J)=-1.*RR(J)/R(J)*DR(J)*DRR(J+1)*WKDQ2(MU,J,NU)
v     920 CONTINUE
   C*----------
s         PF(0)=0.0D0
s         F(0)=0.0D0
s         E(0)=1.0D0
s         C(1)=0.0D0
   C
s3        DO 925 J=1,JD
s3         E(J)=1.0/(B(J)-C(J)*E(J-1))
s3         F(J)=(C(J)*F(J-1)+D(J))*E(J)
s3         PF(J)=(C(J)*PF(J-1)+PD(J))*E(J)
s3    925 CONTINUE
   C*C=
   C*C=
s         WKDP2(MU,JD+1,NU)=PF(JD)/(1.0-E(JD))
s         WKDVP(MU,JD+1,NU)=F(JD)/(1.0-E(JD))
   C*C=
s5        DO 930 J=JD,1,-1
s5        WKDP2(MU,J,NU)=WKDP2(MU,J+1,NU)*E(J)+PF(J)
s5        WKDVP(MU,J,NU)=WKDVP(MU,J+1,NU)*E(J)+F(J)
s5    930 CONTINUE
   C
s     915 CONTINUE
   !xocl end spread
          :
```

Fig. 2.11 'DO 915' loop of subroutine DPE4R (before vectorized).

```
          :
        DIMENSION WKB(0:MIG,0:MJG), WKC(0:MIG,0:MJG),
       $     WKD(0:MIG,0:MJG), WKE(0:MIG,0:MJG),
       $     WKF(0:MIG,0:MJG), WKPD(0:MIG,0:MJG),
       $     WKPF(0:MIG,0:MJG)
          :
   !xocl spread do /ipmk
s         DO 915 NU=1,KD-1
   C*-------- MU=NU= NOT ZERO ----------------------
s2        DO 920 J=1,JD
v2        DO 920 MU=1,ID/2-1
v2        WKC(MU,J)=DRR(J+1)*R(J-1)/DRR(J)/R(J)
v2        WKB(MU,J)=1.0+WKC(MU,J)+DR(J)*DRR(J+1)*RR(J)/R(J)
         &*(RMV(MNPV,MU)/(RR(J))**2+RNV(MNPV,NU))
v2        WKD(MU,J)=-1.*RR(J)/R(J)*DR(J)*DRR(J+1)*WKDQD(MU,J,NU)
v2        WKPD(MU,J)=-1.*RR(J)/R(J)*DR(J)*DRR(J+1)*WKDQ2(MU,J,NU)
v2     920 CONTINUE
   C*----------
v         DO 921 MU=1,ID/2-1
v         WKPF(MU,0)=0.0D0
v         WKF(MU,0)=0.0D0
v         WKE(MU,0)=1.0D0
v         WKC(MU,1)=0.0D0
v      921  CONTINUE
   C
s2        DO 925 J=1,JD
v2        DO 925 MU=1,ID/2-1
v2         WKE(MU,J)=1.0/(WKB(MU,J)-WKC(MU,J)*WKE(MU,J-1))
v2         WKF(MU,J)=(WKC(MU,J)*WKF(MU,J-1)+WKD(MU,J))*WKE(MU,J)
v2         WKPF(MU,J)=(WKC(MU,J)*WKPF(MU,J-1)+WKPD(MU,J))*WKE(MU,J)
v2     925 CONTINUE
   C*C=
   C*C=
v         DO 926 MU=1,ID/2-1
v         WKDP2(MU,JD+1,NU)=WKPF(MU,JD)/(1.0-WKE(MU,JD))
v         WKDVP(MU,JD+1,NU)=WKF(MU,JD)/(1.0-WKE(MU,JD))
v      926  CONTINUE
   C*C=
s         DO 930 J=JD,1,-1
v         DO 930 MU=1,ID/2-1
v         WKDP2(MU,J,NU)=WKDP2(MU,J+1,NU)*WKE(MU,J)+WKPF(MU,J)
v         WKDVP(MU,J,NU)=WKDVP(MU,J+1,NU)*WKE(MU,J)+WKF(MU,J)
v      930 CONTINUE
   C
s      915 CONTINUE
   !xocl end spread
          :
```

Fig. 2.12 'DO 915' loop of subroutine DPE4R (after vectorized).

```
      :
      E11=0.0D0
      :
      DO 1000 I=1,ID
      DO 1000 K=1,KD
      :
      E11=E11+DUMI
      :
      E11=E11+DUMI
      :
      E11=E11+DUMI
      :
 1000 CONTINUE
      :
```

Fig. 2.13 Processing which hinders vectrization.

```
        SUBROUTINE TEST(A,B,N,S)
        DIMENSION A(N),B(N)
        S=0.0
v       DO 100 I=1,N
v         S=S+A(I)
v         S=S+B(I)
v   100 CONTINUE
        RETURN
        END
```

Fig. 2.14 Processing which can be vectorized.

```
      :
      E11=0.0D0
      :
      E11$1=0.0D0                ←追加
      E11$2=0.0D0                ←追加
      :
      DO 1000 I=1,ID
      DO 1000 K=1,KD
      :
      E11=E11+DUMI
      :
      E11$1=E11$1+DUMI           ←変更
      :
      E11$2=E11$2+DUMI           ←変更
      :
 1000 CONTINUE
      :
      E11=E11+E11$1+E11$2        ←追加
      :
```

Fig. 2.15 Modification of the process fro vectorization.

```
      :
      AFB=(EUHEI(J+1)-EUHEI(J))*RDRR(J+1)                    ←定義(*1)
      AFC=(EUHEI(J)-EUHEI(J-1))*RDRR(J)                      ←定義(*1)
      :
      DO 1000 I=1,ID
      DO 1000 K=1,KD
      :
      DUMI=(V(I,J,K,3)-EUHEI(J))*(DB*AFB+DC*AFC)*.50D0       ←参照
      :
      AFB=(EWHEI(J+1)-EWHEI(J))/DRR(J+1)                     ←定義(*2)
      AFC=(EWHEI(J)-EWHEI(J-1))/DRR(J)                       ←定義(*2)
      :
      PROW=PROW+2.0D0*(V(I,J,K,1)-EWHEI(J))
     &*(DB*AFB+DC*AFC)*0.5D0                                 ←参照
      :
      AFB=(EWHEI(J+1)-EWHEI(J))/DRR(J+1)                     ←定義(*3)
      AFC=(EWHEI(J)-EWHEI(J-1))/DRR(J)                       ←定義(*3)
      :
      PRO23=PRO23+(DB**2*AFB+DC**2*AFC)*0.5D0                ←参照
      :
1000  CONTINUE
      :
```

Fig. 2.16 Another processing which hinders vectorization.

参 考 文 献

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

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

[3] 根本俊行，渡辺秀雄，他；原子力コードの VPP500 におけるベクトル化，並列化及び移植，JAERI-Data/Code 96-022，1996 年 7 月.

3. WSPEEDIコードの並列化

本章では，富士通製分散メモリ型ベクトル並列計算機 VPP500/42（以下 VPP と呼ぶ）向けに行なった WSPEEDI コードの並列化について述べる．本コードは，当計算機向けにベクトル化済みであり，これによる高速な計算が実現されていた．しかし，本コードは計算コストの高い部分に並列性（処理，データの独立性）を持っており，より高速な計算の実現が可能である．尚，並列化作業を行う前に，更にベクトル化可能な部分に対してベクトル化も行い，並列化はそのベクトル化版を基に行った．

以降では，ベクトル化作業，並列化作業について述べる．尚，今回のベクトル化を行う前のコードをオリジナル版，今回のベクトル化後のコードを追加ベクトル化版，並列化後のコードを並列化版と呼ぶ．

3.1 コード概要

放射能粒子拡散予測コード WSPEEDI は，風速場計算コード（本章で示す図においては WIND code と表記する）と濃度／線量計算コード（本章で示す図においては DENSITY/DOSE code と表記する）から成る．これら 2 つのコードはそれぞれ独立に実行され，風速場計算コードの実行結果を濃度／線量計算コードが使用する [1] [2].

3.1.1 風速場計算コード

本コードは，ネームリストで与えられる計算領域（例： ASIA）とサイト（例： AREA4）のパラメータから，それらに該当する地形データを読み込み，計算対象領域を 3 次元メッシュとして作成する．そして，地球規模の予報データを読み込み，計算対象領域のメッシュ上の風速場を内挿により求める．更に，内挿された風速場が質量保存則を満たすように補正される．

メインルーチンに存在する時間ループ内において，ネームリストで与えられる計算開始時刻と計算継続時間，出力時間間隔より，計算開始時刻から計算終了時刻（計算開始時刻 + 計算継続時間）まで出力時間間隔毎に予報データを読み込んで風速場計算を行う．予報データは 6 時間または 12 時間毎に複数個存在する．求められた各時間の風速場データは，濃度／線量計算コードの入力データになる．メインルーチンの処理構造を Fig. 3.1 に示す．

3.1.2 濃度／線量計算コード

風速場計算コードにおいて作成された計算対象領域（3 次元メッシュ）と同一の領域を作成し，風速場データを使用して，領域内の放射能の拡散を粒子によって模擬する．

粒子は，計算開始時から既に全粒子が領域内に存在するのではなく，δ t 毎に増えていく．そして粒子が，計算対象領域から外れた場合や持っている放射能量が微量になった場合には，それらの粒子を以降の計算対象から外す．従って，粒子数はδ t 毎に変化する．粒子の位置は，風速場や大気の乱れ，乱数等によって求められる．

メインルーチンには，風速場データを読み込むループとその中のδｔループが存在する．δｔループの直前で風速場データを読み込み，δｔループ前半部分で粒子を発生させ放射能拡散を模擬し，各粒子から放射能空気中濃度や地表面への放射能沈着量を求める．そしてδｔループの後半部分では，δｔの加算値がWIDRCSになった場合に，空気中濃度の平均値や沈着量の積算，線量当量等の計算と出力を行う．メインルーチンの処理構造をFig. 3.2に示す．

3.1.3 ファイルアクセスルーチン

WSPEEDIコードでは，入出力データは，計算領域別，且つサイト別に区別して使用する．例えば，ネームリストで計算領域ASIA，サイトAREA4と指定されると，それに対する入力データを読み込み，その実行結果は，計算領域ASIA，サイトAREA4の実行結果としてファイルに保存される．VPP上では，それらのデータの区別を次のようなディレクトリ構造によって実現している．

$HOME/WSPEEDI/DATA/計算領域名/サイト名/各データ

実行が開始され，これらのディレクトリを区別し，指定されたデータの読み込みまたは書き出しを行うのは，ファイルアクセスルーチン群であり，それらは，C言語でコーディングされている（WSPEEDIコードはFORTRANとC言語の2つの言語で作成されている）．そのため，本コードの実行結果としては，FORTRANからの標準出力と実データ出力，C言語からの標準出力と実データ出力がある．FORTRANからの実データは，主にリスタート用またはグラフィック用の出力であり，上記のようなディレクトリ構造によるデータの区別は行われていない．

3.2 動的挙動解析

チューニングを始める前に，2つのオリジナル版コードの計算コスト分布を把握するため，逐次実行（ベクトル実行）における動的挙動解析を行った．この解析には，VPP上に用意されている動的挙動解析ツール**SAMPLER** [3]を使用した．このツールは実行中のプログラムに対し割り込みをかけ，その割り込み回数から計算コスト分布を出力する．

解析時に使用した風速場計算コード用のネームリストを，Fig. 3.3に示す．このように設定すると，読み込む予報データは，Table 3.1の16個になる．それらは，$HOME/GPVGディレクトリ配下に置かれて使用されるものであるが，Cではなく FORTRANでアクセスされる．また，濃度／線量計算コードについては，扱える最大粒子数を40000個に，ネームリストはFig. 3.4のように設定し，解析を行なった．

風速場計算コードについての解析結果をTable 3.2に，濃度／線量計算コードについての解析結果をTable 3.3に示す．尚，割り込み間隔は，CPU占有時間で10msecとし，計算領域ASIA，サイトAREA4のデータで解析を行った．

3.3 ベクトル化

本コードは，ベクトル化済であったが，更にベクトル化可能な部分があったため，追加ベクトル化を行った．それは，濃度／線量計算コードのサブルーチン**INIL2**とサブルーチン**PCKNG**で

ある.

3.3.1 サブルーチン INIL2

本ルーチンにおいて，ベクトル化可能な部分は，DO 4000 ループ，DO 4200 ループ，DO 4650 ループ，DO 4700 ループである．それぞれ，Fig. 3.5，Fig. 3.6，Fig. 3.7，Fig. 3.8に示す．DO 4000 ループと DO 4650 ループはループ中で文字型配列を使用しているために，また，DO 4200 ループと DO 4700 ループはループ中でユーザ関数 DCY，DCZ を呼んでいるために，VPP ではベクトル化されない．

そこで，DO 4000 ループと DO 4650 ループについては，使用している文字型配列と新たに用意した 4 バイト整数型配列を equivalence 宣言し，整数型配列をループ中で使用するようにした．変更後の DO 4000，DO 4650 ループをそれぞれ Fig. 3.9，Fig. 3.10 に示す．

また，DO 4200 ループと DO 4700 ループについては，関数 DCY，DCZ をそれらのループ中にインライン展開し，関数中の GO TO 構造を IF-ELSEIF-ELSE-ENDIF 構造に変更した．GO TO 構造ではインライン展開してもベクトル化されないためである．実際にコンパイラによるインライン展開を有効にするためには，サブルーチン INIL2 を格納しているファイル inil2.f に 2 つの関数を追加し，コンパイルオプションに -Of を使用する必要がある．更に，関数 DCY，DCZ が格納されていたファイル dcy.f，dcz.f はコンパイル対象から外す必要がある．ファイル inil2.f に追加した関数 DCY，DCZ の変更を Fig. 3.11，Fig. 3.12 に示す．DO 4200，DO 4700 ループは変更する必要はない．

3.3.2 サブルーチン PCKNG

本サブルーチンでは，粒子が計算領域から外れたか，または放射能量が微量になったかをチェックする．更に，これらに該当する粒子が存在した場合には，粒子の属性データを格納する各配列内のデータをパッキングする．例えば，5 個の粒子があり，3 番目の粒子と 4 番目の粒子が計算領域から外れた（または微量になった）場合に，引き続き以降の計算対象にするのは，1 番目，2 番目，5 番目の粒子であり，個数は 5 個から 3 個になる．ここでパッキングとは，1 番目，2 番目のデータはそのままにし，従来の 5 番目のデータを新たに 3 番目に格納する（配列中でデータを詰める）ことを言う．

オリジナル版の DO 4000 ループを Fig. 3.13 に示す．ここでは，各粒子について，計算領域から外れたか，または放射能量が微量かをチェックし，これに該当しない粒子をリスト配列 J に格納している．このような 2 重ループの場合，最内ループ（DO 4001）がベクトル化の対象になるが，ベクトル長が短い（NCLID=2）ため高速化されない．そこで，このループについては，各粒子の放射能量を求める部分と，チェック & リスト配列への格納部分に分割し，それぞれを粒子ループでベクトル化することにした．また，分割を行うと，変数 QQ がまたがって使用されることになるため，この変数を格納する配列 QQ(ipp) を新たに宣言して使用するようにした．変更結果を Fig. 3.14 に示す．これより，*DO 4000 ループと DO 4001 ループはコンパイラにより一重化されることになった.*

次に，オリジナル版の DO 4100 ループを Fig. 3.15 に示す．ここでは，引き続き以降の計算対象となる粒子について，データのパッキングを行う．ここでも，DO 4000 ループの改良のよう

に，放射能量についてのパッキング部分とそれ以外のパッキング部分に分割した．また，放射能量についてのパッキング部分は，ベクトル長の大きい粒子ループでベクトル化されるように外側ループと内側ループを入れ換えた．更に，ループ中で使用されている文字型配列と新たに用意した 4 バイト整数型配列を equivalence 宣言し，整数型配列をループ中で使用するようにした．この変更結果を Fig. 3.16 に示す．しかし，このままでは，すべての配列においてリスト配列 J を用いてアクセスしていることから，コンパイラが回帰計算が発生すると認識してベクトル化されない．しかし，ここでのパッキング処理では，回帰計算は発生しないことから，コンパイラに回帰計算が発生しないことを指示する最適化制御行 *vocl loop,novrec を各ループに指定することによりベクトル化されるようになった．これを Fig. 3.17 に示す．

3.3.3 ベクトル化の評価

ここでは，今回行った濃度／線量計算コードの追加ベクトル化による性能評価を行うため，実行時間を測定し，オリジナル版コードとの性能を比較した．実行時間には，コード実行開始直後（メインルーチンの実行文第 1 行目）とコード実行終了直前（メインルーチン最終 stop 文の直前）に，CPU 占有時間を取得できるサービスルーチン clock を呼び出し，その差を採用した．入力データは，動的挙動解析時と同様のものを用い，扱うことのできる最大粒子数は 40000 個とした．測定結果を Table 3.4 に示す（単位：sec）．

計算結果は，一部 1.0e-17 ～ 1.0e-18 レベルで異なる部分はあったものの，大部分が完全一致であった．

3.4 並列化

ここでは，2 つのコードそれぞれの並列化の方針，実際のコーディング，並列化の評価について述べる．両コード共，計算コストの高い部分に並列性があったため，効率良い並列化による高速化が予想された．

3.4.1 並列化前コードの実行時間計測

Table 3.2 と Table 3.3 の動的挙動解析結果より計算コストの高い部分を予想することは可能であったが，並列化前の実際の実行時間を把握するため，経過時間を取得できるサービスルーチン gettod を使用し測定を行った．

3.4.1.1 風速場計算コードについて

本コードについては，オリジナル版コードのベクトル実行を行い，実行開始から時間ループ直前までの前処理部分，時間ループ部分，全体部分の 3 つについて測定した．入力データは，動的挙動解析時と同様のものを用いた．測定結果を Table 3.5 に示す．

3.4.1.2 濃度／線量計算コードについて

本コードについては，追加ベクトル化版コードのベクトル実行を行い，δ t ループ中の前半部分と後半部分（IF 文で分岐される部分），全体の 3 つの部分について測定した．入力データ

は，動的挙動解析時と同様のものを用い，扱える最大粒子数を 40000 個までとした．測定結果を Table 3.6 に示す．

3.4.2 並列化の方針

2 つのコードの計算コストの高い部分，且つ並列性のある部分について，並列処理方法を検討した．

3.4.2.1 風速場計算コードについて

本コードは，動的挙動解析結果（Table 3.2）と処理構造（Fig. 3.1），及び Table 3.5 より，時間ループ部分の計算コストが高い．そこで，時間ループの 1 回転毎の各予報データについての計算が他の予報データについての計算とは依存関係が無い（並列性がある）ことから，このループを分割し並列化を行うことにした．

具体的には，VPP の各 PE にデータ数が均等になるように予報データを振り分けてそれぞれ独立に計算させることにした．4PE で並列化した場合のイメージを Fig. 3.18 に示す．ここで，例えば，w97022312 等は予報データが格納されているファイル名を表し，"97022312" は 1997 年 2 月 23 日 12:00 を示す．

3.4.2.2 濃度／線量計算コードについて

本コードは，動的挙動解析結果（Table 3.3）と処理構造（Fig. 3.2），及び Table 3.6 より，δ t ループの計算コストが高い．

このループは，大きく前半部分と後半部分に分かれる．前半部分は，計算対象領域（3 次元メッシュ）上における各粒子の移流・拡散についての計算であり，δ t 毎に実行される部分である．後半部分は，ある時刻に達した時（ある時間粒子を模擬した結果）の，メッシュ上における放射能空気中濃度や地表面への放射能沈着量，線量当量（率）を計算してそれを出力する部分であり，δ t の和がある時刻に達した時にだけ行われる部分である．従って，δ t ループが何回か回転する毎に 1 回，後半部分が実行されることになる．

今後粒子数を増やして実行を行う場合，前半部分の計算コストが高くなることが予想される．また，前半部分は粒子について独立に計算できる（並列性がある）部分であり，処理構造も粒子ループが主体となっていることから，粒子について処理とデータを分割し並列化を行うことにした．

具体的には，扱うことのできる最大粒子数（**MAXP**）を並列実行時に使用する PE 台数で分割し，各 PE には分割された **MAXP** を最大粒子数として計算を行わせるようにした．従って，δ t ループにおける粒子の増減は，PE 毎に独立に行われるようになる．4PE で並列化した場合のイメージを Fig. 3.19 に示す．

3.4.3 コーディング

ここでは，2 つのコードの実際の並列化コーディングについて説明する．コーディングでは，特に I/O の仕方を工夫する必要があった．

3.4.3.1　並列化指示行

VPP 上での並列化は，VPP 特有の並列化指示行をコード中に挿入して行う．並列化指示行とは，コード中において !xocl で始まる行のことを言い，並列処理を行わせる部分に対し指定する．これについては，付録 C を参照されたい．

3.4.3.2　リージョンと並列ジョブの I/O

リージョンと並列ジョブの I/O の制御は，VPP 特有の概念であり，VPP FORTRAN 並列コンパイラでのみ有効になるものである．これについては，付録 Dを参照されたい．

3.4.3.3　風速場計算コードについて

本コードにおいて，計算コストの高い時間ループはメインルーチンに存在する．並列化コーディングでは，このメインルーチンを主にコーディングの対象とした．

(1) 並列化の方針に従った時間ループの分割とその問題

本コードの時間ループの分割は，例えば，予報データが a1 ～ a7 まで 7 個存在するとし，4PE 並列実行を行うとすれば，1PE 上には，a1，a2，2PE 上には，a3，a4，3PE 上には，a5，a6，4PE 上には，a7 のように各予報データを均等に割り当てるのではなく，1PE 上には，a1，a5，2PE 上には，a2，a6，3PE 上には，a3，a7，4PE 上には，a4 のようにサイクリックに各予報データを割り当てて並列化を行う必要があった．

・実データ逐次出力の問題

データをサイクリックに配分しなければならなかったのは，C 言語で作成されているファイルアクセスルーチンから出力する実データが，データの古い順（予報データの古い順，計算した風速場時間の古い順）に出力される仕様であったためである．

ファイルアクセスルーチンから順番に出力しなければならない実データには，風速場データとそのデータが何時のものであるかを表す時間，その順番（番号）がある．風速場データを格納するファイル名には古い順に番号が付けられ，その番号とその風速場データが何時のものであるかを表す時間がペアで，別に用意されるリストファイルに格納される（ここで，時間ステップ毎に新たに作成されるファイルは，風速場データを格納するファイルであり，更新されるファイルは，リストファイルである）．ファイルアクセスルーチンでは，ある時間ステップにおいて実データを出力する場合，前時間ステップまでのリストファイルを参照して，その時点で風速場データを格納するファイル名の番号を決める仕様であるために，データの古い順に出力する必要があった（データの書き出し順に依存関係があった）．

4PE 実行での a1，a2，a3，a4 それぞれの計算を並列化を行なった時間ループの 1 ステップとすると，1 ステップ毎に実データ出力の前に全 PE で同期をとり，1PE から 4PE まで順番に出力すれば，データの古い順での出力が正常になる．逆に，こうしないと PE 毎に同時に出力を行うため，時間ループの各ステップにおいて，各 PE からの出力による上書きが発生し，風速場データが格納されるファイルとリストファイル中のペアは，それぞれ 1 個しか作成されないことになる．これは，各 PE が独立に動作している状態から出力が行なわれるため，各 PE は自

己のデータを1番目のデータとして出力するからである.

以上のように順番に出力した場合は，逐次実行時と同様の出力順序であるため，この出力部分については並列化による高速化は望めない.

・計算負荷非均等の問題

予報データの読み込みは，時間ループ中で最初に呼ばれるサブルーチン DPWET 中で行われるため，読み込みの前に読み込む予報データの時刻を配分する必要がある．そこで，時間ループの前にその時刻の配分を行うことにした．ネームリストの計算開始時刻からそれに計算継続時間を足した時刻（計算終了時刻）の間に，読み込むべき予報データの時刻をすべて配列に保存しておき，各 PE ではその配列の添字を操作することで，前述のようなサイクリックな時刻の配分を行わせるようにした．しかし，設定した時刻に対する予報データが存在しない場合もあり得るため，ある PE では全く計算が行われない場合が発生する．これにより，計算負荷が均等化されないため，その分高速化は望めない.

(2) 問題の解決策

(1) で述べたように，実データの出力が逐次であること，予報データが存在しない場合に計算負荷が非均等になることから，高速化が期待できない．ここで詳細な実行時間は示さないが，4PE ベクトル並列実行では，1PE ベクトル実行に比較して 1/2 程度までしか時間短縮されなかった．そこで，これら 2 つの問題についてユーザを交え検討を行った.

検討した結果，ユーザ側でその 2 つの問題を解決するための FORTRAN サブルーチンとファイルアクセスルーチン（C 言語）を新たに 1 つずつ作成して頂くことになった．この新しい FORTRAN サブルーチンでは (1) の 2 つの問題を解決するための機能が 2 つ追加された．実データ逐次出力の問題については新 FORTRAN サブルーチンの 1 つの機能と新ファイルアクセスルーチンの機能によって解決され，計算負荷非均等問題については新 FORTRAN サブルーチンのもう一方の機能によって解決されることになった.

・データ逐次出力問題の解決について

従来は，実データ出力時に，ファイル名に付ける番号と何時の風速場データであるかを表す時間のペアリストを，前ステップまでのリストを参照して決定していた．そのため，それらペアを格納するファイルはステップ毎に更新していた．しかし，新 FORTRAN サブルーチンによって，このファイルを予め（時間ループ前に）作成しておく機能が追加され，更に，ファイルアクセスルーチンによって，実データ出力時にそのファイルを参照してその時のファイル番号を決定する機能が追加された.

・計算負荷非均等問題の解決について

従来は，予報データの読み込みのみによってその時刻の予報データが存在するか否かを判定していた．しかし，新 FORTRAN サブルーチンによって，どの時刻の予報データが存在するかを管理しているリストファイルを参照して，計算開始時刻から計算終了時刻の範囲に存在するすべての予報データの時刻を予め（時間ループ前に）配列に格納しておく機能が追加された.

以上，合計 3 つの機能の追加により，(1) の問題は解決した．そのため，時間ループにおいては，出力の順番は気にする必要はなくなり，且つ読み込んだ予報データは必ず存在することにな

るので，計算負荷が均等化されるようになった．

(3) 新しい時間ループの分割

2 つの新たなサブルーチンが追加されたことにより，時間ループの各ステップは，PE 毎に完全独立に実行可能になり，予報データは時刻の前後関係を意識せずに，どのように配分しても問題は無くなった．ただし，計算負荷を均等にするため，予報データの時刻数はできるだけ均等にする必要がある．

そこで，時間ループの直前に，PE 毎に予報データ時刻が格納されている配列の要素数を PE 毎に均等にアクセスするように，各 PE に対しその配列の添字の範囲を設定するサブルーチン **INIT_PARA1** を作成した．これを Fig. 3.20 に示す．このルーチンにより，各 PE には，**i_min(PE-ID)** ～ **i_max(PE-ID)** が添字範囲として求まる．そして，時間ループでは，各 PE の添字範囲を PE 毎にアクセスさせる方法で並列化を行った．実際のメインルーチンのコーディングを Fig. 3.21 に示す．

DO 1111 ループは PE 台数分回転させるループであるが，**spread do** 文で分割しているため，各 PE で 1 回しか回転しない．**DO** 変数 **ipe** は **PE-ID** として **DO 1111** ループ内で使用する．そして，**DO 4000** ループでは **ipe** 値を利用して各 PE 上で，**i_min(ipe)** ～ **i_max(ipe)** まで回転させるようにしている．尚，サブルーチン **INIT_PARA1** はメインルーチンが格納されているファイル **mainb.f** 内にコーディングした．

(4) 並列化版 I/O への対応

並列実行部分（時間ループ中）において，FORTRAN からの出力は，標準出力と機番 3 と結合されているファイルへの出力がある．時間ループは **spread do** 文により並列化を行っているため，標準出力ファイル（共有ファイル）への出力は，各 PE 上のリージョンからの出力になる．この場合，VPP FORTRAN 並列コンパイラの機能により，入出力の結果が入出力文を単位として実行されたものであることを保証しているため，上書きは発生しない．ただし，各 PE は独立に動作することから順不同書きが発生するため，作成されたファイルは非常に見にくいものになってしまう．仮に，標準出力ファイルに対し制御を行って順不同書きを無くしたとしても，上書きを発生させないための保証機能が動作するため出力時間がかかってしまう．

そこで，時間ループ中からの標準出力は，出力時間を短縮するため専有ファイルに，且つ上書きを発生させないため PE 毎に異なる機番で異なるファイルに書き出すように変更した．機番 3 に結合されているファイルも，パラレルリージョン外でオープンされているため共有ファイルとして扱われるが，これについても標準出力と同様に，時間ループ中では専有ファイルに，且つ上書きを発生させないため PE 毎に異なる機番で異なるファイルに書き出すようにした．実際に，各 PE 上のリージョンの専有ファイルにするには，そのリージョンで初めてオープン（書き出し）を行えばよい．そこで，PE 毎の機番の設定を時間ループの直前で行い，その機番をコモン変数として宣言し各該当ルーチンで使用するようにした．機番の設定は，Fig. 3.21 の 5 ～ 7 行目で行い，コモン変数はインクルードファイル **INC_WRITE** に格納し各該当ルーチンでそのファイルをインクルードするようにした．このインクルードファイルを Fig. 3.22 に示す．インクルードしているサブルーチンを Table 3.7 に示す．尚，時間ループ直前までの計算は，主にメッ

シュ作成部分であるため，それと時間ループ中の風速場計算の出力を区別するため，時間ループ直前までの標準出力ファイルと機番 3 と結合されているファイルへの出力は，オリジナル版と同一にした．

(5) I/O の高速化

本コードの出力においては，SSU キャッシュファイルシステム（SCFS） [8] を利用するように変更した．VPP からの I/O は標準で NFS を利用するようになっているが，NFS より SCFS を利用した方が I/O が高速になる．利用するには，各ユーザのホームディレクトリ上に用意されている，vfl または wkvfl ディレクトリ配下を使用すればよい．そこで，FORTRAN からの出力と C 言語からの出力をすべて wkvfl 配下に書き出すように変更した（wkvfl は空き領域をすべて使用可能であるためである）．

FORTRAN の出力については，実行ジョブを投入する際に使用する実行シェルスクリプト中において，出力ファイルを wkvfl 配下に指定すればよい．また，C 言語の出力については，コード中で陽にファイルパス名を指定してオープンを行っているので，その指定を wkvfl 配下のファイルとして指定するようにした．ここでは，C 言語の出力についての変更のみを Fig. 3.23 に示す．このパス名を変更しておけば，wkvfl 配下のファイルに C 言語から出力が行われる．

3.4.3.4 濃度／線量計算コードについて

本コードで計算コストの高い部分は，メインルーチンに存在する GO TO 8000 ループ（δ t ループ）中の粒子についての計算部分である．また，粒子については独立に計算可能であることから，この部分を並列化の対象とした．並列化を行なうと，この部分は PE 毎に異なる動作をするが，この他の部分は，逐次実行または全 PE で冗長実行となる．

(1) MAXP の分割

MAXP 値は，本コードで扱うことのできる最大粒子数を表す．本並列化では，この MAXP 値を分割させ，PE 毎に異なる MAXP 値を持たせ，各 PE には該当する MAXP 値に従って計算を行わせる方法で並列化を行った．例えば，MAXP=40000 と指定されていると，逐次実行では，1PE 上で 40000 までの粒子を扱うのに対し，4PE 並列実行では，各 PE に MAXP=10000 という値を持たせ，PE 毎に 10000 個までの粒子を扱わせるようにした．従って，δ t ループ毎の粒子の増減も PE 毎に独立に行われることになる．

この MAXP の分割は，メインルーチンの parallel region 文の直後において，今回新たに作成したサブルーチン INIT_PARA1 と spread do 文によって行った．この部分を Fig. 3.24 に示す．また，サブルーチン INIT_PARA1 を Fig. 3.25 に示す．Fig. 3.24 において，サブルーチン INIT_PARA1 より，PE 毎の MAXP（ここでは，i_maxp_pe(ipe)）が求められる．それを spread do 文を用いて各 PE 上で MAXP に代入する．Fig. 3.25 においては，8 行目～ 22 行目で仮引数 maxp を PE 台数で均等に分割した時の各 PE 上の MAXP 値 i_maxp_pe(ipe) を求める．この処理以降では，MAXP の値は各 PE 上の値が使用されることになる．

(2) データの分割

本コードは，粒子についての計算が独立に行われていることから，粒子についての属性データを格納する各配列を分割することにした．これにより，逐次実行時よりもPE毎のメモリ使用量が軽減されることから，従来よりも多くの粒子について計算を行うことが可能になる．

データの分割は，IPP の分割に合わせて行う必要がある．この IPP とは，粒子の各属性データの最大サイズであり，パラメータ文で値が指定され，各データは IPP を用いて宣言されている．そこで，データ分割については，IPP をインデックス範囲に指定し，且つ均等分割を指定した **index partition** 文を利用して分割を行った．例えば，**IPP=40000** で配列 **X(IPP)** が宣言されていれば，4PE 並列実行では，1PE には，**X(1～10000)**，2PE には，**X(10001～20000)**，3PE には，**X(20001～30000)**，4PE には，**X(30001～40000)** のように均等にデータが配分される．

今回の並列化で分割した配列等の宣言の大部分は，新たにインクルードファイルを用意してその中で行うようにした．メインルーチン用のインクルードファイル名は **INC_MAIN**，サブルーチン用のインクルードファイル名は **INC_SUB**，**INC_SUB2** とした．メインルーチンとサブルーチンでインクルードファイルを区別したのは，サブルーチンでは必ず宣言しなければならない並列化指示行があるためで，**INC_SUB** と **INC_SUB2** の 2 つを用意したのは，サブルーチンによって利用する配列名が異なっていたためである．これら 3 つのインクルードファイルをそれぞれ Fig. 3.26，Fig. 3.27，Fig. 3.28 に示す．

これら分割した配列は，実行開始時から分割されていることになる．尚，粒子の属性データの内，文字型配列と乱数を格納する配列は分割の対象にはしていない．文字型配列は分割が許されないためで，乱数を格納する配列は乱数発生ルーチンの実引数に PE 毎の分割範囲を渡すことができないからである．

(3) PE 毎の粒子数の設定

スタート実行（スタート：リスタート実行と区別するために用いた）においては，計算の対象とする粒子数の初期値は 0 であるが，リスタート実行においては，前の実行において計算対象としていた粒子数が初期値になる．そのため，δ t ループの前にその粒子数を均等に各 PE に配分しておく必要がある．その処理は，リスタート実行時に前の実行のデータを読み込むサブルーチン **PERCEL** 中で，今回の並列化で新たに作成したサブルーチン **INIT_PARA2** と **spread do** 文によって行った．この部分を Fig. 3.29 に，サブルーチン **INIT_PARA2** を Fig. 3.30 に示す．Fig. 3.30 において仮引数 **NPLST** を均等分割し，PE 毎の先頭番号 **i_min_par(ipe)** と最終番号 **i_max_par(ipe)** を求め，Fig. 3.29 において **spread do** 文を用い，PE 毎の粒子数 **NPLST** を設定する．この処理以降では，**NPLST** の値は各 PE 上の値を使用することになる．尚，スタート実行では，**NPLST=0** が **parallel region** 文により全 PE にコピーされるので特に配分処理は必要ない．

(4) 乱数発生ルーチンの変更

オリジナル版においては，富士通科学用サブルーチンライブラリ SSLII/VP [6] で用意されている乱数発生ルーチン **RANU2** が使用されていた．しかし，それを並列化版でそのまま用いると，全 PE で種が同一になることから，全 PE で同一の乱数が発生し同一の計算が行われる．

これは，並列化向きではない．そこで，富士通科学用サブルーチンライブラリ SSLII/VPP [7] で用意されている，DP_VRANU4 という並列向き乱数発生ルーチンを利用することにした．ここで，DP_VRANU4 の仕様を Table 3.8 に示す．これは，並列実行中の PE 間での乱数の重なりを避けることのできるサブルーチンである．乱数発生ルーチン自体の変更とそれに伴う宣言文の変更を行ったサブルーチンは，RANU2 の呼び出しを行っている DIFFU，INIL2，VOL1，VOL2 と，種の初期値を与えているメインルーチンである．メインルーチンの変更を Fig. 3.31 に，その他のサブルーチンの変更例を Fig. 3.32 に示す．各々，仕様通りに変更を行った．

(5) 粒子ループの変更と配列アクセスの添字の変更

MAXP の分割やデータの分割，粒子数の分割をしたことにより，粒子ループの回転範囲およびそのループ中の配列要素アクセスのための添字を変更する必要がある．これらの変更を行ったサブルーチンは，PERCEL，TRANP，DIFFU，INIL2，VOL1，PCKNG，DIVIDL，DEPO，PRDECAY，RROUT，RSTOUT であるが，PERCEL，RROUT，RSTOUT については，(8) で説明する．

a. サブルーチン TRANP

本ルーチンでは，DO 4000，DO 4100 ループの回転範囲の変更を行った．この変更を Fig. 3.33 に示す．オリジナル版は，c.org でコメント化し，並列化版は C.VPP500s------ ～ C.VPP500e------ で囲んである（以降同様）．ここで，配列 i_min(npe) には，各 PE に分割した配列の先頭添字が格納されている（Fig. 3.25 の 24 行目～ 37 行目で求めている）．例えば，X(40000) という配列があり，1PE では，1 ～ 10000，2PE では，10001 ～ 20000，3PE では，20001 ～ 30000，4PE では，30001 ～ 40000 という範囲で分割されているとすると，i_min(1)=1，i_min(2)=10001，i_min(3)=20001，i_min(4)=30001 が格納されている．配列 i_min をアクセスする添字 i_pe_num には PE-ID を与えているため，i_min(i_pe_num) によりその PE 上での粒子ループ開始値が得られる．終了値は，i_min(i_pe_num) に PE 毎に持たせた粒子数 NPLST を加えた数になる．

b. サブルーチン DIFFU

本ルーチンの粒子ループの回転範囲の変更は，a. と同様に行った．更に，本ルーチンでは乱数を格納している配列要素をアクセスするための添字も変更する必要があった．それは，乱数発生ルーチンに乱数の個数を与えると，乱数を格納する配列には，添字 1 （先頭）から乱数が格納されるため，変更したループの回転範囲の DO 変数をそのまま用いると乱数が定義されていない箇所をアクセスする場合があるからである．本ルーチンの変更を Fig. 3.34 に示す．また，本ルーチンでローカルに宣言されていた粒子の属性データ WDKY(IPP)，NSP(IPP) もデータ分割を行った．

c. サブルーチン INIL2

本ルーチンでは，δ t 毎に増加した粒子数のみを回転数にした粒子ループが主体である．それらのループの回転範囲の変更も，a. と同様に行い，乱数発生ルーチンも呼び出しているため，乱数格納配列の要素をアクセスするための添字も b. と同様に変更した．更に，粒子ループの DO 変数と回転範囲の先頭値の差を単精度実数型に変換する部分もあったため，それをオリジナ

ル版と同一の単精度実数値になるように変更した．また，本ルーチンの後半部分では，粒子数が MAXP を越えた場合に，越えた数分だけランダムに粒子を選び，それらをリサイクル発生用の粒子として使用している．この選ばれる粒子は MAXP と乱数の積値で決定しているが，他 PE 上の分割された配列をアクセスする値が求まる可能性がある．そのため，範囲外アクセスを防ぐために，選ばれた粒子 ID に i_min(i_pe_num) を加えるようにした．本ルーチンの変更をすべて Fig. 3.35 に示す．

d. サブルーチン VOL1

本ルーチンにおいても同様に，粒子ループの回転範囲と乱数格納配列の要素をアクセスする添字の変更を行った．また，分割した配列 Z の要素をアクセスする添字（リスト配列 IZ）の値の変更も行った．本ルーチンの変更を Fig. 3.36 に示す．

e. サブルーチン PCKNG

本ルーチンについても，粒子ループの回転範囲の変更を行った．ここでは，粒子が計算対象領域から外れたかまたは放射能量が微量になったかをチェックし，それらの粒子を今後の計算対象から外す処理を行う．前半がチェック部分であり，チェックした後の粒子数と今まで扱ってきた粒子数を比較し，異なっていれば，後半で各粒子の属性データのパッキングを行う．粒子数の比較は，各 PE 上の粒子数で独立に行うため，PE 毎のチェック後の粒子数と比較するように変更した．本ルーチンの変更を Fig. 3.37 に示す．

f. サブルーチン DIVIDL

本ルーチンでは，粒子ループの回転範囲を変更した．また，粒子についてのベクトル化のための作業用データを使用していたが，これらもベクトル化のための粒子の属性データであるため，同様に分割を行った．ただし，本ルーチンでローカルに使用されているものであったため，インクルードファイルには取り込まず，本ルーチンの宣言部分で分割宣言を行った．本ルーチンの変更を Fig. 3.38 に示す．

g. サブルーチン DEPO

本ルーチンも f. と同様に粒子ループの回転範囲を変更した．また，f. と同様な作業用データが使用されていたため，本ルーチンの宣言部分で分割宣言を行った．本ルーチンの変更を Fig. 3.39 に示す．

h. サブルーチン PRDECAY

本ルーチンについては，粒子ループの回転範囲を変更するのみであった．本ルーチンの変更を Fig. 3.40 に示す．

(6) 初期放射能量の各粒子への与え方の変更

サブルーチン INIL2 では，δ t 毎に新たに発生させた粒子に対し，放射能量の初期値を与える．各核種の初期値は以下の実行文により求められ（IN は核種 ID），これらが粒子毎の属性データを格納する配列に代入される．

$$\mathrm{QINL(IN)} = \mathrm{TSTEP} * \mathrm{RRATE(IN)/FLOAT(NP)/3600.} \qquad (3.1)$$

しかし，並列化では MAXP を分割しているため，この実行文で用いられる NP が逐次実行時と異なる．NP は同サブルーチン中で，次の実行文によって求められる．

$$NP = INT(FLOAT(MAXP) * TSTEP/TFULL \tag{3.2}$$

※ TSTEP=120.0 （δｔループ時間幅），TFULL=777600.0 （粒子放出時間）は逐次実行と並列実行で変わらない．

例えば，MAXP=40000 とすれば，逐次実行時は NP=6 であるが，4PE 並列実行時は各 PE で MAXP=10000 であるため，NP=1 になる．よって，各粒子に与える初期放射能量が逐次実行時と異なる．また 4PE 合計では NP=4 であるため，逐次実行時と同様の計算はできなくなる．

そこで，ユーザと検討結果，今回の並列化版では，逐次実行時の NP と並列実行時の全 PE 合計の NP を同一にして実行を行なうことと仮定し，上の実行文を次のように変更することにした．

$$QINL(IN) = TSTEP * RRATE(IN)/FLOAT(NP)/3600./FLOAT(NPE) \tag{3.3}$$

これにより，例えば，逐次実行時は NP=12 で，4PE 並列実行時は全 PE 合計で NP=12 になるように各 PE で NP=3 とすれば，更に FLOAT(NPE) （NPE:PE 台数）で割れば，初期放射能量は変わらないことになる．

(7) 空気中濃度と地表面沈着量の全 PE 値の総和

MAXP を分割し，PE 毎に独立に粒子についての計算を行わせているため，各 PE でそれぞれ放射能の空気中濃度と地表面沈着量が求められる．これらは，サブルーチン DIVIDL では配列 ASPC に，サブルーチン DEP0 では配列 DASPC に求められるが，δｔループの前半部分（並列性がある）では加算されていくだけであり，それらの値が初めて利用されるのは後半部分である．しかし，後半部分で利用される部分（冗長実行部）は，全 PE で同値である空気中濃度と地表面沈着量の総和値が必要になるため，予め PE 毎に求められた加算値を PE 間で総和し，その総和値を全 PE に配分して置かなければならない．そこで，総和値を初めて利用するサブルーチン REC を呼び出す直前に，それぞれ総和を行わせる処理をメインルーチンに追加した．これを Fig. 3.41 に示す．

(8) δｔ毎に変わるベクトル長への対応

シミュレーションで用いる粒子は，δｔ毎に増加していき，また減少する場合がある．そのため，ある程度の時間が経過するまでは，粒子ループのベクトル長（回転数）が小さくベクトル化の効果が期待できない．最悪の場合スカラ実行よりも遅くなる場合もある．このようにベクトル長が大きくなったり少なくなったりする場合に，その時点のベクトル長に合わせてベクトル実行させるか否かの最適化を行う制御が VPP にある．それは，最適化制御行 *vocl loop,vdopt であり，これを該当ループの直前に指定しておけば良い．本コードではこの制御行を，粒子ループや入力データによってベクトル長が極端に小さくなることが予想されるループの直前に指定した．指定箇所が多いことから，ここでは指定の例だけを Fig. 3.42 に示す．

(9) 並列化版 I/O への対応

本コードにおいても，風速場計算コードと同様に並列化版 I/O への対応を行った．変更を行ったのは，サブルーチン RROUT，RSTOUT の出力とサブルーチン PERCEL の入力，サブルーチン REC，RECDS の C 言語からの出力についてである．

a. サブルーチン RROUT

本ルーチンからは，機番 62 と結合されているファイルに対し，粒子の属性データである X，Y，Z，PTINDX の出力を行っている．オリジナル版の出力部分を Fig. 3.43 に示す．並列化版において，これらのデータ（文字型配列 PTINDX を除く）は，PE 毎に分割データとして持たせていることから，それぞれを制御を行って出力する必要がある．

そこで，先ず機番 62 に結合するファイルをパラレルリージョン外でダミーオープンを行い共有ファイルとし，各 PE から書き出しを試みた．これを Fig. 3.44 に示す．

また，別の方法も試みた．それは，分割ローカル配列 X，Y，Z については，それらと equivalence 宣言を行っているグローバル配列（どの PE からもアクセス可能）を用いてパラレルリージョンの専有ファイルに書き出し，文字型配列 PTINDX のみを共有ファイルに書き出すという方法である．機番 62 と結合するファイルを専有ファイルとして X，Y，Z を書き出し，機番 63 と結合するファイルを共有ファイル（パラレルリージョン外でダミーオープン）として PTINDX を書き出した．これを Fig. 3.45 に示す．文字型配列 PTINDX はローカル分割データとしてもグローバルデータとしても扱えないため，全 PE で配列全範囲を冗長に持つようになる．しかし，定義される範囲は PE 毎の計算範囲のみであるため，このような書き出し方が必要になる．これら 2 つの出力時間を測定したところ，Fig. 3.45 の方法が速かったためそれを有効にすることにした．ただし，配列 X，Y，Z と配列 PTINDX が異なるファイルに格納されるため，これらのデータを利用する側（例えば，この出力データを入力データとして使うようなプログラム）で変更が必要になる．今回は，その変更を避けるために Fig. 3.44 の方法もコメントとしてコーディングに残してあるので，場合によって使い分ければ問題ない．

更に，本ルーチンには，出力の前に出力データの個数を求める処理がある．粒子の属性データ出力は，各 PE 上から行われるため，出力データの個数の合計は，PE 毎の出力データ数を全 PE 間で総和したものでなければならない．そのため，出力の前にその総和処理を追加した．

b. サブルーチン RSTOUT

本ルーチンでは，配列 X，Y，Z，IMX，IMY，IMZ，IRCY，NSTL，PDT，PTINDX，Q を出力する．オリジナル版の出力部分を Fig. 3.46 に示す．ここでも，RROUT と同様に機番 NOUT3 と結合するファイルをパラレルリージョンの専有ファイルとしてグローバルデータ X，Y，Z，IMX，IMY，IMZ，IRCY，NSTL，PDT，Q を書き出し，機番 NOUT3+1 と結合するファイルを共有ファイルとして PTINDX を書き出すことにした．これを Fig. 3.47 に示す．更に，出力時間を短縮するため，テキスト出力を行っていたものを，バイナリ出力に変更した（バイナリ出力サイズはテキスト出力サイズよりも小さい）．書き出したファイルは，リスタート実行に使用するのみなので，リスタート時の読み込みもこの変更に合わせれば，この部分は最も出力時間を短縮できる．

更に，これらの出力の前に，全粒子数を出力する処理がある．しかしこのままでは，ある PE 上の出力だけが行われることから，並列化版では粒子数を分割しているため，全粒子数が出力さ

れないことになる．そのため，出力の前に全 PE 間で粒子数を総和する処理を追加した．

c. サブルーチン PERCEL

本ルーチンは，前実行の RSTOUT で書き出されたデータを読み込む部分であるため，読み込み方法を RSTOUT の書き出しに合わせた．この変更を Fig. 3.48 に示す．

d. サブルーチン REC, RECDS

これら 2 つのルーチンでは，ファイルアクセスルーチンを呼び出している．それらのルーチンでは，陽にファイル名を指定してファイルをオープンし，データ出力を行っている．この部分は冗長実行部であるが，C 言語では，FORTRAN と異なり冗長に出力を行うため，(n-1)PE 分の出力が無駄になる．そこで，ファイルアクセスルーチンを呼び出す前に，1PE 上だけでそれらのルーチンを呼び出すように FORTRAN 側で並列化指示行を指定し無駄を省いた．指定数が多いことから，ここでは指定の例を Fig. 3.49 に示す．ここで，**!xocl spread region /subp(1)** の **/subp(1)** の指定により，1PE 上のみからサブルーチンが呼び出されることになる．

(10) I/O の高速化

風速場計算コードと同様に本コードにおいても，SSU キャッシュファイルシステム（SCFS）を利用するように変更した．

FORTRAN の出力については，実行ジョブを投入する際に使用する実行シェルスクリプト中において，出力ファイルを wkvfl 配下に指定すればよい．また，C 言語の出力については，風速場計算コードと同様に，コード中で陽にファイルパス名を指定してオープンを行っているので，その指定を wkvfl 配下のファイルとして指定するようにした．ここでは，C 言語の出力についての変更のみを Fig. 3.50 に示す．このパス名を変更しておけば，wkvfl 配下のファイルに C から出力が行われる．

3.4.3.5 C 言語からの標準出力について

ファイルアクセスルーチンは，風速場計算コードでは時間ループ中から，濃度／線量計算コードでは δ t ループから呼び出され，両コード共にパラレルリージョンの範囲内から呼び出される．パラレルリージョンの範囲内における C 言語からの標準出力については，標準出力ファイルへのファイルポインタを各 PE が持つことは FORTRAN と同じであるが，PE 毎のファイルポインタを用いて出力が冗長に行われるため，上書きが発生する．しかし，これらの C からの標準出力は処理の確認用のみであったことから，すべての標準出力処理（**printf** 文）をコメント化した．

3.4.4 並列化の評価

並列化による性能評価を行うため，風速場計算コード，濃度／線量計算コードの両コードについて，実行時間の測定を行い，並列化前の実行時間との比較を行った．

3.4.4.1　風速場計算コードについて

測定の仕方は，並列化前の実行時間測定と同様に，実行開始から時間ループ直前までの前処理部分，時間ループ部分，全体部分の 3 つについて経過時間を測定した．入力データは，動的挙動解析時と同様のものを用いた．比較用として「(0) オリジナル版コードベクトル実行時間」も付けて測定結果を Table 3.9 に示す．これより，16PE ベクトル並列実行は，(0) オリジナル版コードのベクトル実行に比較すると 約 8.3 倍，(1) 並列化版コード 1PE ベクトル実行に比較すると 約 8.6 倍の性能向上を得た．実行に使用した PE 台数倍にならないのは，時間ループ内に存在するファイルアクセスルーチン（C 言語）からの出力時間の影響が考えられる．また，計算結果は，オリジナル版の場合と一致していた．

3.4.4.2　濃度／線量計算コードについて

本コードのオリジナル版においては，扱える最大粒子数は 40000 個に設定されていたが，3.4.3.4の (6) で述べたように，逐次実行時と並列実行時では NP 値が異なる．そこで，ユーザと検討の結果，MAXP=80000 個に設定することにし，1PE，2PE，3PE，4PE，6PE，12PE それぞれの並列実行を行って性能評価を行うことにした．1PE 実行では NP=12 個，2PE 並列実行では各 PE 6 個，3PE 並列実行では各 PE 4 個，4PE 並列実行では各 PE 3 個，6PE 並列実行では各 PE 2 個，12PE 並列実行では各 PE 1 個であり，いずれの場合も全 PE で NP=12 個になる．MAXP=80000 と変更したため，Table 3.10 に示す (1) ～ (9) の各実行を行って実行時間を比較した．Table 3.10 中の RANU2 はオリジナル版で使用されていた乱数発生ルーチン，DP_VRANU4 は今回の並列化で RANU2 から変更した並列化版乱数発生ルーチン，n 個 /PE は，並列化版における各 PE でのδ t 毎の発生粒子数を示す．測定の仕方は，並列化前の実行時間測定と同様に，δ t ループ中の前半部分（並列動作）と後半部分（大部分が冗長実行），全体の 3 つの部分について測定した．入力データは，動的挙動解析時と同様のものを用いた．これより，12PE ベクトル並列実行は，(3) 追加ベクトル化版コードのベクトル実行に比較すると約 4.0 倍，(4) 並列化版コードの 1PE ベクトル実行に比較すると約 3.2 倍の性能向上を得た．計算コストを小さくする目的で並列化を行った部分のδ t ループ前半部分は約 10 ～ 12 倍の性能向上を得ているが，常に，δ t ループ以外の部分に 約 200 ～ 300sec の時間がかかっていることとδ t ループ後半部分に約 700sec の時間がかかっていることが PE 台数倍にならない原因として挙げられる．特にδ t ループ後半部分の時間が目立っているが，この部分にはファイルアクセスルーチン（C 言語）からの出力があるため，その出力時間の影響が考えられる．また，計算結果についてユーザに確認して頂いたところ，少々のバラツキが見られるが，これは乱数の影響と考えられるもので問題無いとのことであった．

3.5　まとめ

今回の並列化では，両コード共，並列性のある部分の計算コストが高かったため，効率良い高速化が期待された．並列化後の時間測定結果の通り，期待はずれの性能向上に留まった．特に濃度／線量計算コードは全体で期待とはほど遠い性能向上に留まった．この最大の原因として，両コード共に C 言語からの出力があり，その時間がほとんど短縮されなかったことが挙げられ

る.

今後，VPP において更に I/O の効率化を進め，性能向上のネックにならないようにする必要があり，また，このように FORTRAN に限らず C 言語でコーディングされているコードも増えることが予想されるため，C 言語による高速化技術も取り込んでいく必要があると考える.よって，これらのことを踏まえて技術の蓄積をしていかなければならない.

Table 3.1 Meteorological forecast data.

w97022312	w97022412	w97022512	w97022612
w97022712	w97022812	w97030112	w97030212
w97030300	w97030312	w97030400	w97030412
w97030500	w97030512	w97030612	w97030712

Table 3.2 Analyzation result of dynamic behavior of WIND code.

Synthesis Information

Count	Percent	VL	Name
36904	44.2	-	sorts_
23777	28.5	506	relux_
12181	14.6	-	wwd30_
2177	2.6	51	rgpv_
1270	1.5	-	denx_
1215	1.5	-	wind0_
967	1.2	-	windad_
928	1.1	-	denz_
798	1.0	-	deny_
776	0.9	1798	cofset_
772	0.9	-	nondiv_
701	0.8	-	divchk_
247	0.3	1048	divset_
210	0.3	-	denc_
173	0.2	-	dpwet_
136	0.2	-	wdf30_
78	0.1	-	output_
53	0.1	-	ro_
50	0.1	-	gmerci_
17	0.0	-	_start
11	0.0	-	chart1_
6	0.0	2048	dpwind_
4	0.0	-	rma30_
2	0.0	-	timead_
2	0.0	-	wdf10_
2	0.0	-	MAIN__
1	0.0	-	vmsset_
1	0.0	-	tset_
1	0.0	-	wwd10_
1	0.0	-	time_get
83461		506	TOTAL

Table 3.3 Analyzation result of dynamic behavior of DENSITY/DOSE code.

```
Synthesis Information
    Count|    Percent|     VL| Name
    68071|       31.7|    827|  dividl_
    40960|       19.1|      -|  ranu2_
    37324|       17.4|    875|  depo_
    26981|       12.6|    476|  pckng_
    17078|        7.9|    374|  diffu_
     7323|        3.4|      -|  rwd30_
     4814|        2.2|      -|  wcn30_
     4327|        2.0|    603|  tranp_
     3012|        1.4|      -|  wds30_
     1728|        0.8|      2|  inil2_
      992|        0.5|      -|  rstout_
      845|        0.4|      5|  recds_
      392|        0.2|    413|  rec_
      252|        0.1|   2048|  MAIN__
      234|        0.1|      -|  rrout_
       86|        0.0|      -|  t_update
       63|        0.0|      -|  _start
       62|        0.0|     21|  metmsh_
       61|        0.0|      -|  dcz_
       54|        0.0|     19|  level1_
       54|        0.0|      2|  prdecay_
       48|        0.0|      -|  dcy_
       26|        0.0|      -|  cide_
       24|        0.0|      -|  time_get
       12|        0.0|      -|  wds10_
        8|        0.0|      -|  gmerci_
        5|        0.0|      -|  prise_
        5|        0.0|      -|  t_check
        4|        0.0|      -|  grand_
        3|        0.0|      -|  rma30_
        3|        0.0|      -|  wcn10_
        3|        0.0|      -|  metco1_
        3|        0.0|      -|  calen_
        3|        0.0|     21|  ibl_
        3|        0.0|      -|  istodt_
        2|        0.0|      -|  cide03_
        2|        0.0|      -|  nuccre_
        1|        0.0|      -|  input_
        1|        0.0|      -|  dpmap_
        1|        0.0|      -|  main
        1|        0.0|      -|  tcnvln_

   214871|           |    611|  TOTAL
```

Table 3.4 Comparison of execution time between vectorized and original of WIND code.

	CPU 占有時間
オリジナル版コードスカラ実行	7443.0 sec
オリジナル版コードベクトル実行	1780.0 sec
追加ベクトル化版コードスカラ実行	7653.0 sec
追加ベクトル化版コードベクトル実行	1668.8 sec

Table 3.5 Execution time of WIND code before parallelized.

前処理部分	時間ループ部分	全体
8.9 sec	1204.8 sec	1213.7 sec

Table 3.6 Execution time of DENSITY/DOSE code before parallelized.

δ t ループ前半部分	δ t ループ後半部分	全体
1514.7 sec	1082.3 sec	2876.2 sec

Table 3.7 Routines that include INC_WRITE.

CHART1	CHART2	CHART3	COFSET	DIVCHK
DPWET	DPWIND	MAIN	NONDIV	OUTPUT
RELUX	RGPV	WIND0	WINDAD	

Table 3.8 Random number generator DP_VRANU4 (for parallel).

```
DP_VRANU4 : CALL DP_VRANU4(IX,DA,N,DWORK,NWORK,ICON)

機能      : 各プロセッサ上で, 区間 [0,1) 上での一様分布から各々ことなる疑似
            乱数列を生成する.
パラメタ :
  IX      : 入力. 出発値, 利用者が与える種. 4 バイト整数型. 出力 0.
            初回呼び出しでは, IX に正の値を与え, 2 回目以降の呼び出しでは返却値
            0 のまま呼び出すこと. VPP では, IX < 8000000 であること.
  DA      : 出力. 各プロセッサで異なった, 区間 [0,1) で一様な N 個の疑似乱数.
            少なくとも N の大きさをもつ倍精度実数型 1 次元配列.
  N       : 入力. DA に返される一様分布疑似乱数の個数.
  DWORK   : 作業領域. 少なくとも大きさ NWORK の倍精度実数型 1 次元配列.
            本サブルーチンを繰り返し呼び出す場合は, 内容を変更してはならない.
            DWORK には, DP_VRANU4 が現在の位置から再度呼び出される場合に必要
            な, 全ての現情報が格納される.
  NWORK   : 入力. 配列 DWORK の大きさ. NWORK ≧ 388 であること. 推奨値 45000.
  ICON    : 出力. コンディションコード.
```

Table 3.9 Comparison of execution time between parallelized and original of WIND code.

time_type	前処理部分	時間ループ部分	全体
オリジナル版コードベクトル実行	8.9	1204.8	1213.7
並列化版コード 1PE ベクトル実行	19.3	1238.8	1258.1
並列化版コード 2PE ベクトル並列実行	9.4	523.7	533.0
並列化版コード 4PE ベクトル並列実行	17.7	305.3	323.0
並列化版コード 8PE ベクトル並列実行	9.4	201.9	211.3
並列化版コード 16PE ベクトル並列実行	12.2	133.4	145.6

Table 3.10 Comparison of execution time between vectorized-parallelized and original of DENSITY/DOSE code.

time_type	δ t ループ前半部分	δ t ループ後半部分	全体
(1)	16372.8	1031.4	17680.5
(2)	3168.9	1438.3	4985.2
(3)	3457.6	1128.6	4874.2
(4)	2885.9	833.5	3966.6
(5)	1454.6	719.5	2334.8
(6)	948.2	712.3	1817.6
(7)	723.2	712.2	1594.8
(8)	537.5	709.2	1413.9
(9)	289.5	719.0	1224.9

(1) 追加ベクトル化版コードスカラ実行（RANU2）
(2) 追加ベクトル化版コードベクトル実行（RANU2）
(3) 追加ベクトル化版コードベクトル実行（DP_VRANU4）
(4) 並列化版コード 1PE ベクトル実行（DP_VRANU4, 12 個 /PE）
(5) 並列化版コード 2PE ベクトル並列実行（DP_VRANU4, 6 個 /PE）
(6) 並列化版コード 3PE ベクトル並列実行（DP_VRANU4, 4 個 /PE）
(7) 並列化版コード 4PE ベクトル並列実行（DP_VRANU4, 3 個 /PE）
(8) 並列化版コード 6PE ベクトル並列実行（DP_VRANU4, 2 個 /PE）
(9) 並列化版コード 12PE ベクトル並列実行（DP_VRANU4, 1 個 /PE）

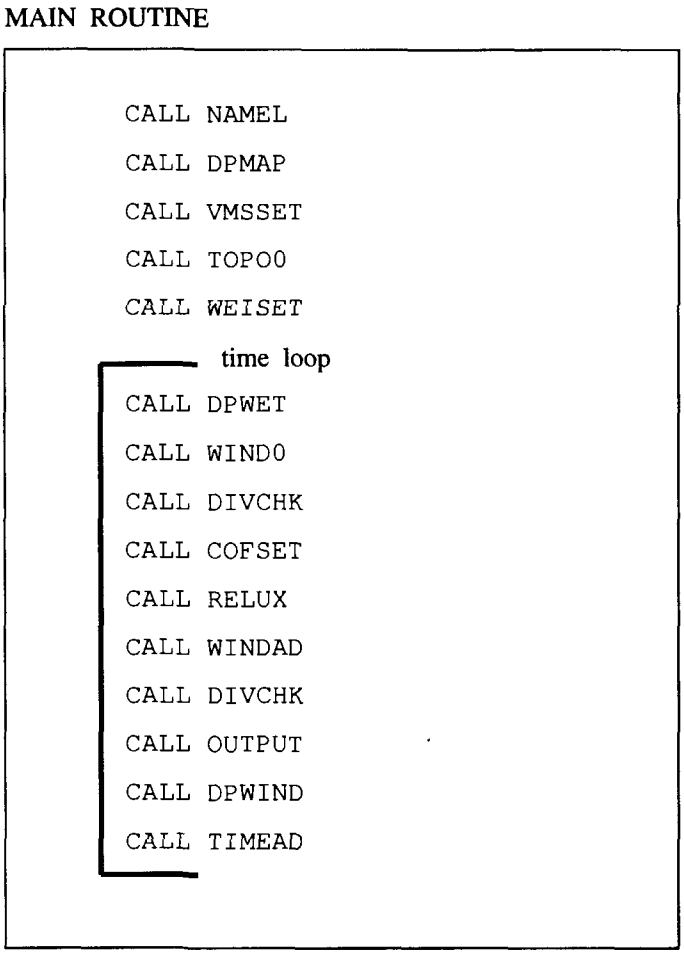


Fig. 3.1 Processing structure of **MAIN** routine of WIND code.

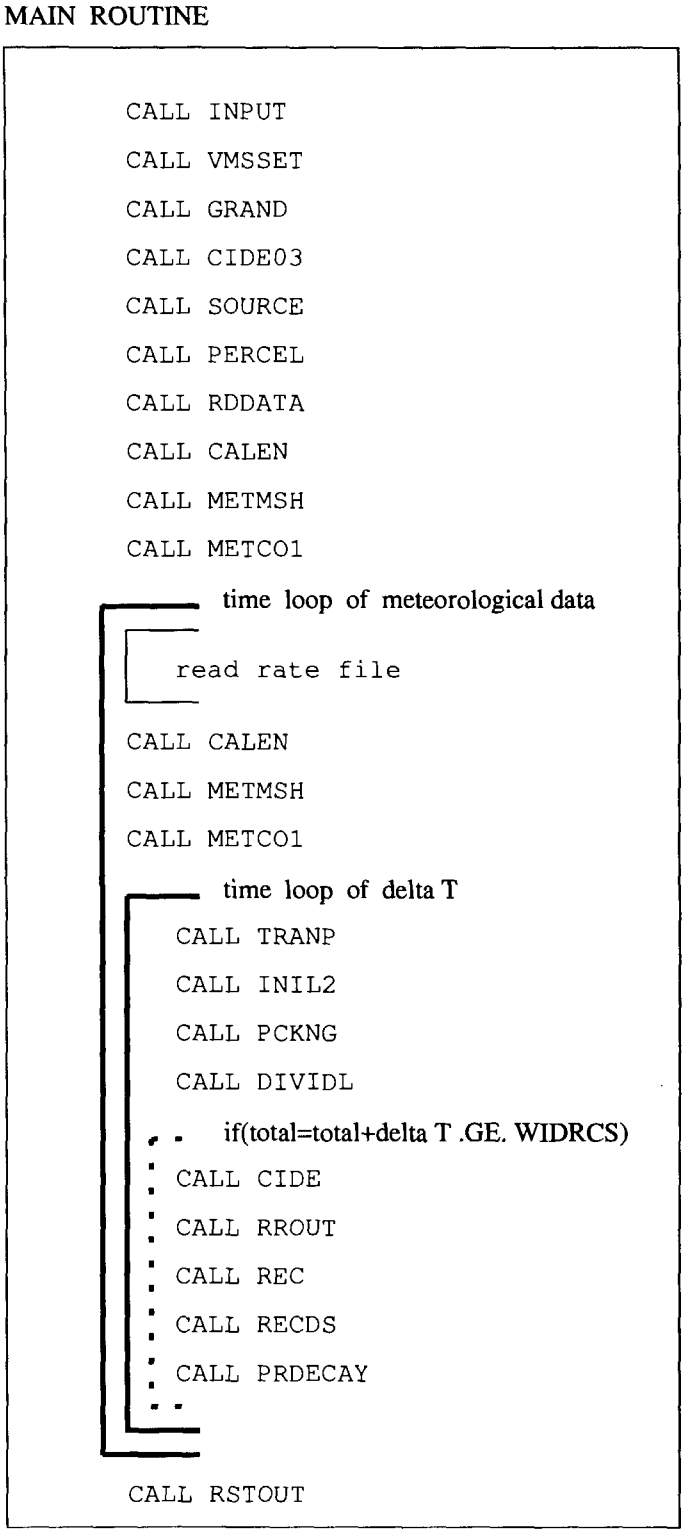


Fig. 3.2 Processing structure of MAIN routine of DENSITY/DOSE code.

```
&WSYNOP
MAPTP='AREA4',
ISTRD=970223,
ISTRT=120000,
ITINT=3000000,
IDMPI=120000,
NAMES='ASIA',
IOPEN=0,
&END
```

Fig. 3.3 Input data for **namelist** of WIND code.

```
&GEARN
ISNAME='ASIA',
IRLFLG=0,
IOPEN=0,
RLAT=30.0000,
RLONG=120.000,
Z00=10.000,
REACT='BWR',
BURNUP=10000,
MAPTP='AREA4',
IRTIM(1)=970223,
IRTIM(2)=120000,
ISDTIM(1)=970223,
ISDTIM(2)=100000,
ICSTIM(1)=970223,
ICSTIM(2)=120000,
ITRACE=3000000,
IWIDRC=120000,
&END
```

Fig. 3.4 Input data for **namelist** for DENSITY/DOSE code.

```
           CHARACTER*8  PTINDX(IPP)
           CHARACTER*12 NTIMEC
                 .
                 .
                 .
m          DO 4000 I=NPLST,NTP
v           X(I) = X00
v           Y(I) = Y00
v           Z(I) = (Z00 + ZRIS)/HRL
s8          PTINDX(I) = NTIMEC(1:8)
v  4000   CONTINUE
                 .
                 .
```

Fig. 3.5 DO 4000 loop in subroutine INIL2.

```
                .
                .
s6        DO 4200 I=NPLST,NTP
s6         R=UR*PDT(I)
s6         DH(I) = UR*DCY(NSS,R,SIGH0)*FCTM**2
s6         DV(I) = UR *DCZ(NSS,R)
s6  4200 CONTINUE
                .
                .
```

Fig. 3.6 DO 4200 loop in subroutine INIL2.

```
            CHARACTER*8  PTINDX(IPP)
            CHARACTER*12 NTIMEC
              .
              .
              .
s4          DO 4650  I = 1,NZAN
v4           IT = II(I)
v4           X(IT) = X00
v4           Y(IT) = Y00
v4           Z(IT) = (Z00 + ZRIS)/HRL
s4           PTINDX(IT) = NTIMEC(1:8)
v4  4650   CONTINUE
                .
                .
```

Fig. 3.7 DO 4650 loop in subroutine INIL2.

```
                .
                .
s5        DO 4700  I = 1, NZAN
v5          IT = II(I)
m5          R  = PDT(IT)*UR
s5          DH(IT) = UR*DCY(NSS,R,SIGH0)*FCTM**2
s5          DV(IT) = UR*DCZ(NSS,R)
v5  4700 CONTINUE
                .
                .
```

Fig. 3.8 DO 4700 loop in subroutine INIL2.

```
          CHARACTER*8  PTINDX(IPP)
          CHARACTER*12 NTIMEC
          INTEGER*4 PT_TMP(2,IPP),NTIME_TMP(3)
          EQUIVALENCE(PTINDX(1),PT_TMP(1,1)),(NTIMEC,NTIME_TMP(1))
            .
            .
            .
v         DO 4000 I=NPLST,NTP
v          X(I) = X00
v          Y(I) = Y00
v          Z(I) = (Z00 + ZRIS)/HRL
  c.org    PTINDX(I) = NTIMEC(1:8)
v          PT_TMP(1,I)=NTIME_TMP(1)
v          PT_TMP(2,I)=NTIME_TMP(2)
v  4000   CONTINUE
            .
            .
```

Fig. 3.9 Vectorized DO 4000 loop in subroutine INIL2.

```
          CHARACTER*8  PTINDX(IPP)
          CHARACTER*12 NTIMEC
          INTEGER*4 PT_TMP(2,IPP),NTIME_TMP(3)
          EQUIVALENCE(PTINDX(1),PT_TMP(1,1)),(NTIMEC,NTIME_TMP(1))
            .
            .
            .
v         DO 4650  I = 1,NZAN
v           IT = II(I)
v           X(IT) = X00
v           Y(IT) = Y00
v           Z(IT) = (Z00 + ZRIS)/HRL
  c.org     PTINDX(IT) = NTIMEC(1:8)
v           PT_TMP(1,IT)=NTIME_TMP(1)
v           PT_TMP(2,IT)=NTIME_TMP(2)
v 4650    CONTINUE
            .
            .
```

Fig. 3.10 Vectorized DO 4650 loop in subroutine INIL2.

```
            .
            .
c.org       IF(R.GT.1000.) GO TO 10
c.org       GAMMA = GAMY(1,N)
c.org       ALPHA = ALPY(1,N)
c.org       GO TO 100
c.orgC
c.org    10 CONTINUE
c.org       GAMMA = GAMY(2,N)
c.org       ALPHA = ALPY(2,N)
c.org   100 CONTINUE
C.VPP500s----------------------
      if(R.gt.1000.) then
        GAMMA = GAMY(2,N)
        ALPHA = ALPY(2,N)
      else
        GAMMA = GAMY(1,N)
        ALPHA = ALPY(1,N)
      endif
C.VPP500e----------------------
C
      DCY = GAMMA*ALPHA*R**(ALPHA-1)*(GAMMA*R**ALPHA+SIGI)
            .
            .
```

Fig. 3.11 Modified function DCY.

```
            .
            .
c.org       IF(R.GT.300.) GO TO 10
c.org       GAMMA = GAMZ(1,N)
c.org       ALPHA = ALPZ(1,N)
c.org       GO TO 100
C
c.org    10 IF(R.GT.500.) GO TO 20
c.org       GAMMA = GAMZ(2,N)
c.org       ALPHA = ALPZ(2,N)
c.org       GO TO 100
C
c.org    20 IF(R.GT.1000.) GO TO 30
c.org       GAMMA = GAMZ(3,N)
c.org       ALPHA = ALPZ(3,N)
c.org       GO TO 100
C
c.org    30 IF(R.GT.2000.) GO TO 40
c.org       GAMMA = GAMZ(4,N)
c.org       ALPHA = ALPZ(4,N)
c.org       GO TO 100
C
c.org    40 IF(R.GT.10000.) GO TO 50
c.org       GAMMA = GAMZ(5,N)
c.org       ALPHA = ALPZ(5,N)
c.org       GO TO 100
C
c.org    50 GAMMA = GAMZ(6,N)
c.org       ALPHA = ALPZ(6,N)
c.org   100 CONTINUE
```

Fig. 3.12 Modified function DCZ (1/2).

```
C
C.VPP500s-----------------------
      if(R.gt.300..and.R.gt.500..and.R.gt.1000..and.R.gt.2000..and.R.gt.
     &10000.) then
       GAMMA = GAMZ(6,N)
       ALPHA = ALPZ(6,N)
       elseif(R.gt.300..and.R.gt.500..and.R.gt.1000..and.R.gt.2000..and.R
     &.le.10000.) then
       GAMMA = GAMZ(5,N)
       ALPHA = ALPZ(5,N)
       elseif(R.gt.300..and.R.gt.500..and.R.gt.1000..and.R.le.2000.) then
       GAMMA = GAMZ(4,N)
       ALPHA = ALPZ(4,N)
       elseif(R.gt.300..and.R.gt.500..and.R.le.1000.) then
       GAMMA = GAMZ(3,N)
       ALPHA = ALPZ(3,N)
       elseif(R.gt.300..and.R.le.500.) then
       GAMMA = GAMZ(2,N)
       ALPHA = ALPZ(2,N)
       else
       GAMMA = GAMZ(1,N)
       ALPHA = ALPZ(1,N)
       endif
C.VPP500e-----------------------
      DCZ = GAMMA**2*ALPHA*R**(ALPHA-1)*(R**ALPHA)
            .
            .
```

Fig. 3.12 Modified function DCZ (2/2).

```
            .
            .
 s       DO 4000 I=1,NPLST
 v         QQ = 0.0
 v          DO 4001 IN=1,NCLID
 v             QQ = QQ + Q(I,IN)
 v  4001    CONTINUE
 v          IF(X(I).LT.XMTMIN.OR.X(I).GE.XMAX.OR.
       *      Y(I).LT.YMTMIN.OR.Y(I).GE.YMAX.OR.
       *      QQ.LT.1.D-76)                              THEN
 v          ELSE
 m            ICNT = ICNT + 1
 m            J(ICNT) = I
 v          ENDIF
 v  4000 CONTINUE
            .
            .
```

Fig. 3.13 DO 4000 loop in subroutine PCKNG.

```
           REAL*8 QQ(IPP)
              .
              .
v          DO 4000 I=1,NPLST
v          QQ(I)=0.0
v2         DO 4001 IN=1,NCLID
v2         QQ(I)=QQ(I)+Q(I,IN)
v2    4001 CONTINUE
v     4000 CONTINUE
v          DO 5000 I=1,NPLST
v          IF(X(I).LT.XMTMIN.OR.X(I).GE.XMAX.OR.
          *       Y(I).LT.YMTMIN.OR.Y(I).GE.YMAX.OR.
          *       QQ(I).LT.1.D-76)                    THEN
v          ELSE
v            ICNT = ICNT + 1
v            J(ICNT) = I
v          ENDIF
v     5000 CONTINUE
              .
              .
```

Fig. 3.14 Vectorized DO 4000 loop in subroutine PCKNG.

```
             CHARACTER*8 PTINDX(IPP)
                .
                .
s            DO 4100 I=1,MAX
s              IT = J(I)
m              IRCY(I) = IRCY(IT)
s              X(I) = X(IT)
s              Y(I) = Y(IT)
s              Z(I) = Z(IT)
v              DO 4101 IN= 1,NCLID
v                 Q(I,IN) = Q(IT,IN)
v      4101    CONTINUE
s              NSTL(I) = NSTL(IT)
s              PDT(I) = PDT(IT)
s              PTINDX(I) = PTINDX(IT)
v      4100 CONTINUE
                .
                .
```

Fig. 3.15 DO 4100 loop in subroutine PCKNG.

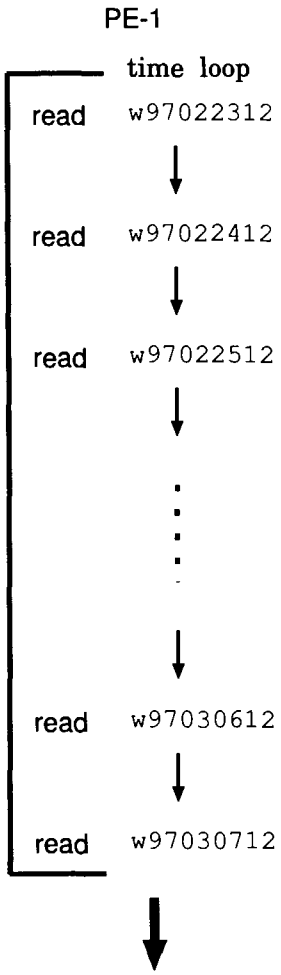
```
          CHARACTER*8 PTINDX(IPP)
          INTEGER*4 PT_TMP(2,IPP)
          EQUIVALENCE(PTINDX(1),PT_TMP(1,1))
            .
            .
 s        DO 4101 IN=1,NCLID
 s8       DO 4100 I=1,MAX
 m8         Q(I,IN)=Q(J(I),IN)
 v8  4100 CONTINUE
 s   4101 CONTINUE
 s2       DO 5100 I=1,MAX
 m2       NSTL(I)=NSTL(J(I))
 s2       IRCY(I)=IRCY(J(I))
 m2       X(I)=X(J(I))
 s2       Y(I)=Y(J(I))
 s2       Z(I)=Z(J(I))
 s2       PDT(I)=PDT(J(I))
 s2       PT_TMP(1,I)=PT_TMP(1,J(I))
 s2       PT_TMP(2,I)=PT_TMP(2,J(I))
 v2  5100 continue
            .
            .
```

Fig. 3.16 Divided DO 4100 loop in subroutine PCKNG.

```
          CHARACTER*8 PTINDX(IPP)
          INTEGER*4 PT_TMP(2,IPP)
          EQUIVALENCE(PTINDX(1),PT_TMP(1,1))
            .
            .
s2        DO 4101 IN=1,NCLID
   *vocl loop,novrec
v2        DO 4100 I=1,MAX
v2          Q(I,IN)=A(J(I),IN)
v2   4100 CONTINUE
s2   4101 CONTINUE
   *vocl loop,novrec
v         DO 5100 I=1,MAX
v         NSTL(I)=NSTL(J(I))
v         IRCY(I)=IRCY(J(I))
v         X(I)=X(J(I))
v         Y(I)=Y(J(I))
v         Z(I)=Z(J(I))
v         PDT(I)=PDT(J(I))
v         PT_TMP(1,I)=PT_TMP(1,J(I))
v         PT_TMP(2,I)=PT_TMP(2,J(I))
v    5100 CONTINUE
            .
            .
```

Fig. 3.17 Vectorized DO 4100 loop in subroutine PCKNG.

single execution :



parallel execution :

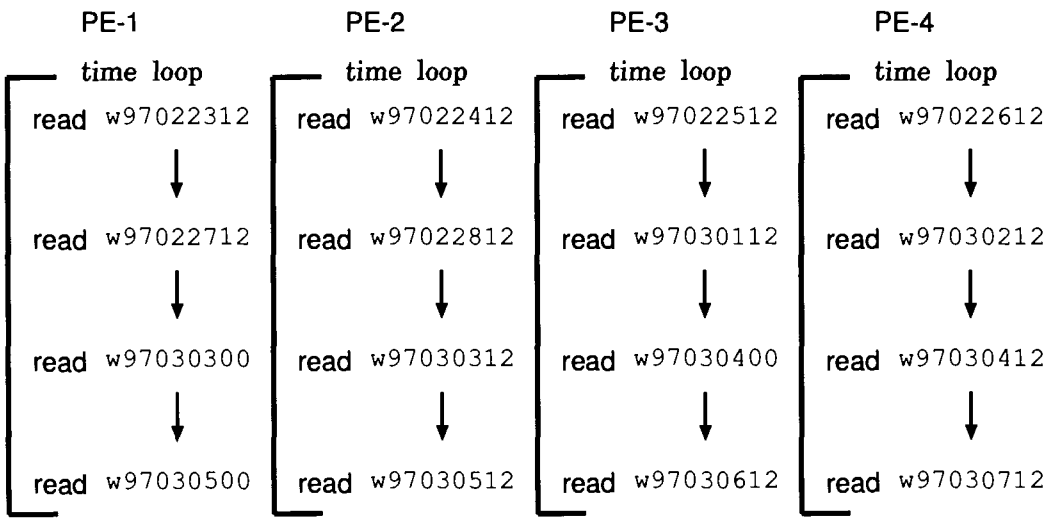


Fig. 3.18 Outline of parallel execution of WIND code.

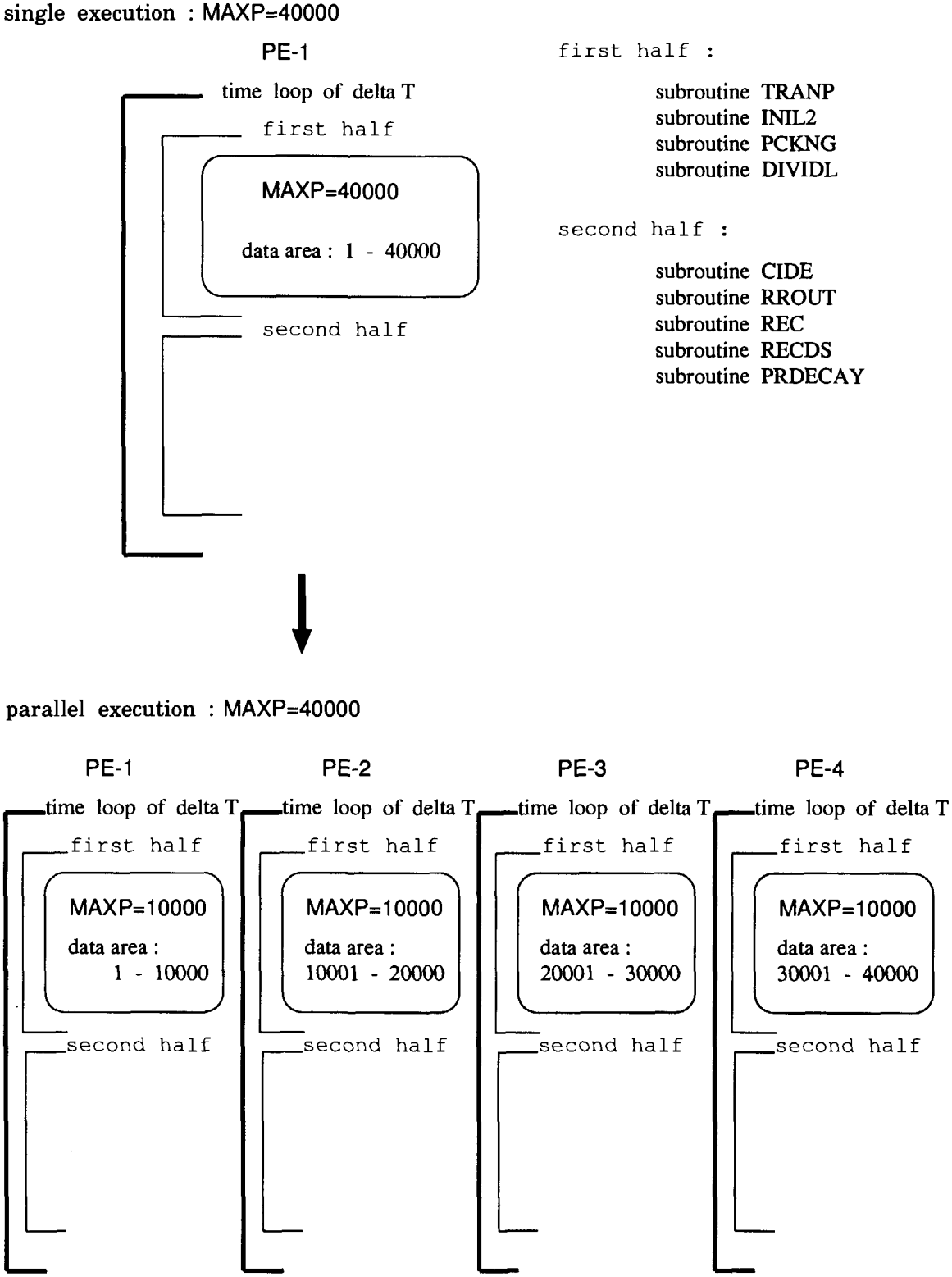


Fig. 3.19 Outline of parallel execution of DENSITY/DOSE code.

```
      subroutine init_para1(nloop)
*include INC_PE
!xocl processor p(npe)
!xocl subprocessor subp(npe)=p(1:npe)
      common /para_cntl/i_min(npe),i_max(npe+1)
      integer*4 i_interval(npe)

      i_max(npe+1)=0
      iloop=nloop
      ipe_int=0

      do 1 i=npe,1,-1
      iloop=iloop-ipe_int
      ipe_int=iloop/i
      i_max(i)=i_max(i+1)+ipe_int
    1 continue

      do 2 i=npe,1,-1
      i_min(i)=i_max(i+1)+1
      i_interval(i)=i_max(i)-i_min(i)+1
    2 continue

      do 6 ipe=1,npe
      i_min(ipe)=0
      i_max(ipe)=0
    6 continue

      i_soeji=0
      do 4 ipe=1,npe
      do 5 i=1,i_interval(ipe)
      i_soeji=i_soeji+1
      if(i_min(ipe).eq.0) i_min(ipe)=i_soeji
    5 continue
      i_max(ipe)=i_soeji
    4 continue

      return
      end
```

Fig. 3.20 Subroutine INIT_PARA1.

```
      common /para_cntl/i_min(npe),i_max(npe+1)
           .
           .
 1: !xocl parallel region
 2:       call init_para1(nloop)
 3: !xocl spread do
 4:       do 1111 ipe=1,npe
 5:       ip_num=ipe-1
 6:       i6_out=6*10+ip_num
 7:       i3_out=3*10+ip_num
 8:
 9:       DO 4000 I=i_min(ipe),i_max(ipe)
10:       ntimec = ntimet(i)
11:       CALL DPWET(IRET)
12:
13:       write(i6_out,'(// a )') 'Mainroutine '
14:       write(i6_out,*) 'DPWET is called for ',NTIMEC,': result=',IRET
15:
16:       if( IRET .eq. 0 ) then
17:         CALL WINDO
18:         CALL DIVCHK
19:         CALL COFSET
20:         CALL RELUX
21:         CALL WINDAD
22:         CALL DIVCHK
23:         CALL NONDIV
24:         CALL DIVCHK
25:         CALL OUTPUT( 3 )
26:         CALL DPWIND(IRET,ipe)
27:       end if
28:
29:  4000 CONTINUE
30:  1111 continue
31: !xocl end spread
32: !xocl end parallel
33:
34:   999 STOP
35:       END
```

Fig. 3.21 Parallelized **MAIN** routine of WIND code.

```
      common /out_id/i6_out,i3_out
```

Fig. 3.22 Include file **INC_WRITE**.

```
メインルーチン
           .
           .
      HPASS='/dg03/ufs09/j9145/wkvf1'                ←
      call tset
     . (ntimec,nitint,idumpi,gpass,hpass,names,maptp
     .               ,nloop,ntimet)
           .
           .
```

Fig. 3.23 Modified output file path for C language routines of WIND code.

```
                .
                .
!xocl parallel region
      call init_para1(maxp)        ←
!xocl spread do
      do 6222 ipe=1,npe
      i_pe_num=ipe
      IINI=ipe
      MAXP=i_maxp_pe(ipe)          ←
 6222 continue
!xocl end spread
                .
                .
```

Fig. 3.24 Distribution of MAXP in MAIN routine.

```
      subroutine init_para1(maxp)
*include AINCLDBL
*include CONCDS
*include INC_SUB

!xocl spread do
      do 11 ipe=1,npe
      i_min(ipe)=0
      i_max(ipe)=0
   11 continue
!xocl end spread

      i_min_tmp=0
!xocl spread do /p_sub
      do 20 i=1,maxp
      if(i_min_tmp.eq.0) then
        i_min_tmp=i
      endif
   20 continue
!xocl end spread
!xocl spread do
      do 30 ipe=1,npe
      i_min(ipe)=i_min_tmp
      i_max(ipe)=i-1
      i_maxp_pe(ipe)=i_max(ipe)-i_min(ipe)+1
   30 continue
!xocl end spread

      i_min_tmp=0
!xocl spread do /p_sub
      do 40 i=1,ipp
      if(i_min_tmp.eq.0) then
        i_min_tmp=i
      endif
   40 continue
!xocl end spread
!xocl spread do
      do 50 ipe=1,npe
      i_min(ipe)=i_min_tmp
      i_max(ipe)=i-1
   50 continue
!xocl end spread

      return
      end
```

Fig. 3.25 Subroutine INIT_PARA1 of DENSITY/DOSE code.

```
*include INC_PE
!xocl processor p(npe)
!xocl index partition p_main=(p,index=1:IPP,part=band)
!xocl index partition pe_main=(p,index=1:npe,part=band)

      common /para_cntl1/i_min(npe),i_max(npe),
     &                   i_min_l(npe),i_max_l(npe),
     &                   i_min_par(npe),i_max_par(npe),
     &                   i_maxp_pe(npe)
      common /para_cntl2/i_pe_num
      integer*4 ircy(ipp),nstl(ipp)
      real*8 q(ipp,nfp),x(ipp),y(ipp),z(ipp),pdt(ipp)
      common /reso/   ircy_g(ipp)
      common /src/    q_g(ipp,nfp)
      common /place/  x_g(ipp), y_g(ipp), z_g(ipp)
      common /trvl/   nstl_g(ipp), pdt_g(ipp)

!xocl local i_min(/pe_main),i_max(/pe_main),i_min_l(/pe_main)
!xocl local i_max_l(/pe_main),i_min_par(/pe_main)
!xocl local i_max_par(/pe_main)
!xocl local i_maxp_pe(/pe_main)

!xocl local ircy(/p_main)
!xocl local q(/p_main,:)
!xocl local x(/p_main),y(/p_main),z(/p_main)
!xocl local nstl(/p_main),pdt(/p_main)

!xocl global ircy_g
!xocl global q_g
!xocl global x_g,y_g,z_g
!xocl global nstl_g,pdt_g

      equivalence (ircy,ircy_g),(nstl,nstl_g),
     &            (q,q_g),(x,x_g),(y,y_g),(z,z_g),(pdt,pdt_g)
```

Fig. 3.26 Include file INC_MAIN.

```
*include INC_PE
!xocl processor p(npe)
!xocl subprocessor subp(npe)=p(1:npe)
!xocl index partition p_sub=(subp,index=1:IPP,part=band)
!xocl index partition pe_sub=(subp,index=1:npe,part=band)

      common /para_cntl1/i_min(npe),i_max(npe),
     &                   i_min_l(npe),i_max_l(npe),
     &                   i_min_par(npe),i_max_par(npe),
     &                   i_maxp_pe(npe)
      common /para_cntl2/i_pe_num
      integer*4 ircy(ipp),nstl(ipp)
      real*8 q(ipp,nfp),x(ipp),y(ipp),z(ipp),pdt(ipp)
      integer*4 imx(ipp),imy(ipp),imz(ipp)
      real*8 wspew(ipp),wspns(ipp),wspud(ipp),spdz(ipp),
     &       up(ipp),vp(ipp),wp(ipp)

      common /reso/   ircy_g(ipp)
      common /src/    q_g(ipp,nfp)
      common /place/  x_g(ipp), y_g(ipp), z_g(ipp)
      common /trvl/   nstl_g(ipp), pdt_g(ipp)
      common /meshpl/ imx_g(ipp),imy_g(ipp),imz_g(ipp)
      common /speed/  wspew_g(ipp),wspns_g(ipp),wspud_g(ipp),spdz_g(ipp)
      common /uvwp/   up_g(ipp),vp_g(ipp),wp_g(ipp)

!xocl local i_min(/pe_sub),i_max(/pe_sub),i_min_l(/pe_sub)
!xocl local i_max_l(/pe_sub),i_min_par(/pe_sub)
!xocl local i_max_par(/pe_sub)
!xocl local i_maxp_pe(/pe_sub)

!xocl local ircy(/p_sub)
!xocl local q(/p_sub,:)
!xocl local x(/p_sub),y(/p_sub),z(/p_sub)
!xocl local nstl(/p_sub),pdt(/p_sub)
!xocl local imx(/p_sub),imy(/p_sub),imz(/p_sub)
!xocl local wspew(/p_sub),wspns(/p_sub),wspud(/p_sub),spdz(/p_sub)
!xocl local up(/p_sub),vp(/p_sub),wp(/p_sub)

!xocl global ircy_g
!xocl global q_g
!xocl global x_g,y_g,z_g
!xocl global nstl_g,pdt_g
!xocl global imx_g,imy_g,imz_g
!xocl global wspew_g,wspns_g,wspud_g,spdz_g
!xocl global up_g,vp_g,wp_g
      equivalence (ircy,ircy_g),(nstl,nstl_g),
     &            (q,q_g),(x,x_g),(y,y_g),(z,z_g),(pdt,pdt_g),
     &            (imx,imx_g),(imy,imy_g),(imz,imz_g),
     &            (wspew,wspew_g),(wspns,wspns_g),(wspud,wspud_g),
     &            (spdz,spdz_g),(up,up_g),(vp,vp_g),(wp,wp_g)
```

Fig. 3.27 Include file INC_SUB.

```
*include INC_PE
!xocl processor p(npe)
!xocl subprocessor subp(npe)=p(1:npe)
!xocl index partition p_sub=(subp,index=1:IPP,part=band)
!xocl index partition pe_sub=(subp,index=1:npe,part=band)

      common /para_cntl1/i_min(npe),i_max(npe),
     &                   i_min_1(npe),i_max_1(npe),
     &                   i_min_par(npe),i_max_par(npe),
     &                   i_maxp_pe(npe)
      common /para_cntl2/i_pe_num

      integer*4 ircy(ipp),nstl(ipp)
      real*8 q(ipp,nfp),x(ipp),y(ipp),z(ipp),pdt(ipp)
      integer*4 imx(ipp),imy(ipp),imz(ipp)
      real*8 wspew(ipp),wspns(ipp),wspud(ipp),spdz(ipp),
     &       dlt(ipp),dh(ipp),dv(ipp)

      common /reso/   ircy_g(ipp)
      common /src/    q_g(ipp,nfp)
      common /place/  x_g(ipp), y_g(ipp), z_g(ipp)
      common /trvl/   nstl_g(ipp), pdt_g(ipp)
      common /meshpl/ imx_g(ipp),imy_g(ipp),imz_g(ipp)
      common /speed/  wspew_g(ipp),wspns_g(ipp),wspud_g(ipp),spdz_g(ipp)
      common /uvwp/   dlt_g(ipp),dh_g(ipp),dv_g(ipp)

!xocl local i_min(/pe_sub),i_max(/pe_sub),i_min_1(/pe_sub)
!xocl local i_max_1(/pe_sub),i_min_par(/pe_sub)
!xocl local i_max_par(/pe_sub)
!xocl local i_maxp_pe(/pe_sub)

!xocl local ircy(/p_sub)
!xocl local q(/p_sub,:)
!xocl local x(/p_sub),y(/p_sub),z(/p_sub)
!xocl local nstl(/p_sub),pdt(/p_sub)
!xocl local imx(/p_sub),imy(/p_sub),imz(/p_sub)
!xocl local wspew(/p_sub),wspns(/p_sub),wspud(/p_sub),spdz(/p_sub)
!xocl local dlt(/p_sub),dh(/p_sub),dv(/p_sub)

!xocl global ircy_g
!xocl global q_g
!xocl global x_g,y_g,z_g
!xocl global nstl_g,pdt_g
!xocl global imx_g,imy_g,imz_g
!xocl global wspew_g,wspns_g,wspud_g,spdz_g
!xocl global dlt_g,dh_g,dv_g

      equivalence (ircy,ircy_g),(nstl,nstl_g),
     &            (q,q_g),(x,x_g),(y,y_g),(z,z_g),(pdt,pdt_g),
     &            (imx,imx_g),(imy,imy_g),(imz,imz_g),
     &            (wspew,wspew_g),(wspns,wspns_g),(wspud,wspud_g),
     &            (spdz,spdz_g),(dlt,dlt_g),(dh,dh_g),(dv,dv_g)
```

Fig. 3.28 Include file INC_SUB2.

```
              .
              .
      call init_para2(NPLST)
!xocl spread do
      do 9000 ipe=1,npe
      NPLST=i_max_par(ipe)-i_min_par(ipe)+1
 9000 continue
!xocl end spread
              .
              .
```

Fig. 3.29 Distribution of the number of particles per PE in subroutine PERCEL.

```
      subroutine init_para2(nplst)
*include AINCLDBL
*include CONCDS
*include INC_SUB

!xocl spread do
      do 11 ipe=1,npe
      i_min_par(ipe)=0
      i_max_par(ipe)=0
   11 continue
!xocl end spread

      i_min_par_tmp=0
!xocl spread do
      do 25 j=1,nplst
      if(i_min_par_tmp.eq.0) then
        i_min_par_tmp=j
      endif
   25 continue
!xocl end spread
!xocl spread do
      do 30 ipe=1,npe
      i_min_par(ipe)=i_min_par_tmp
      i_max_par(ipe)=j-1
   30 continue
!xocl end spread
!xocl spread do
      do 31 ipe=1,npe
      i_min_l(ipe)=i_min(ipe)
      i_max_l(ipe)=i_min(ipe)+(i_max_par(ipe)-i_min_par(ipe))
   31 continue
!xocl end spread

      return
      end
```

Fig. 3.30 Subroutine INIT_PARA2.

```
c.org COMMON /VRAND/  IINI, RAND(IPP)
      COMMON /VRAND/  IINI, RAND(IPP), DWORK(IPP), NWORK   ←
c.org REAL      *4    RAND
      REAL      *8    RAND,DWORK                           ←
                .
                .
      NWORK=45000                                          ←
                .
                .
c.org IINI    = 0
      IINI    = 1                                          ←
                .
                .
!xocl parallel region
      call init_para1(maxp)
!xocl spread do
      do 6222 ipe=1,npe
      i_pe_num=ipe
      IINI=ipe                                             ←
      MAXP=i_maxp_pe(ipe)
 6222 continue
!xocl end spread
                .
                .
```

Fig. 3.31 Modified MAIN routine for random number generator.

```
c.org COMMON /VRAND/  IINI, RAND(IPP)
      COMMON /VRAND/  IINI, RAND(IPP), DWORK(IPP), NWORK      ←
c.org REAL*4 RAND
      REAL*8 RAND,DWORK                                       ←
                .
                .
c.org CALL RANU2(IINI, RAND, NPLST, ICON)
      CALL DP_VRANU4(IINI, RAND, NPLST, DWORK, NWORK, ICON)   ←
                .
                .
```

Fig. 3.32 Example of modified subroutines for random number generator.

```
                .
                .
c.org DO 4000 I=1,NPLST
C.VPP500s-------------
      DO 4000 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-------------
                .
                .
c.org DO 4100 I=1,NPLST
C.VPP500s-------------
      DO 4100 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-------------
                .
                .
```

Fig. 3.33 Modified DO loops in subroutine TRANP.

```
C.VPP500s-------------
!xocl local wdky(/p_sub),nsp(/p_sub)
C.VPP500e-------------
                .
                .
c.org DO 5000 I=1,NPLST
C.VPP500s-------------
      DO 5000 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-------------
                .
                .
c.org CALL RANU2(IINI, RAND, NPLST, ICON)
C.VPP500s-------------
      CALL DP_VRANU4(IINI, RAND, NPLST, DWORK, NWORK, ICON)
C.VPP500e-------------
c.org DO 5100 I=1,NPLST
c.org    UP(I) = WSPEW(I)*TSTEP +
c.org*           SQRT(24.0*WDKY(I)*TSTEP) * (0.5 - RAND(I))
C.VPP500s-------------
      DO 5100 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
         UP(I) = WSPEW(I)*TSTEP +
     *           SQRT(24.0*WDKY(I)*TSTEP) * (0.5 - RAND(I-i_min(i_pe_num
     &)+1))
C.VPP500e-------------
                .
                .
c.org CALL RANU2(IINI, RAND, NPLST, ICON)
C.VPP500s-------------
      CALL DP_VRANU4(IINI, RAND, NPLST, DWORK, NWORK, ICON)
C.VPP500e-------------
c.org DO 5200 I=1,NPLST
c.org    VP(I) = WSPNS(I)*TSTEP +
c.org*           SQRT(24.0*WDKY(I)*TSTEP) * (0.5 - RAND(I))
C.VPP500s-------------
      DO 5200 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
         VP(I) = WSPNS(I)*TSTEP +
     *           SQRT(24.0*WDKY(I)*TSTEP) * (0.5 - RAND(I-i_min(i_pe_num
     &)+1))
C.VPP500e-------------
                .
                .
```

Fig. 3.34 Modified DO loops and subscript of arrays in subroutine DIFFU (1/2).

```
c.org CALL RANU2(IINI, RAND, NPLST, ICON)
C.VPP500s---------------
      CALL DP_VRANU4(IINI, RAND, NPLST, DWORK, NWORK, ICON)
C.VPP500e---------------
c.org DO 5300 I=1,NPLST
C.VPP500s---------------
      DO 5300 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e---------------
                .
                .
c.org IF(RAND(I).GE.0.5) GO TO 30
C.VPP500s---------------
      IF(RAND(I-i_min(i_pe_num)+1).GE.0.5) GO TO 30
C.VPP500e---------------
                .
                .
```

Fig. 3.34 Modified DO loops and subscript of arrays in subroutine DIFFU (2/2).

```
                .
                .
c.org    DO 4000 I=NPLST,NTP
C.VPP500s-----------
         DO 4000 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
C.VPP500e-----------
                .
                .
c.org DO 4100 I = NPLST,NTP
c.org  T = TSTEP - FLOAT(I-NPLST)*DELT
C.VPP500s-----------
       DO 4100 I = i_minn(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
        T = TSTEP - FLOAT(I-i_min(i_pe_num)+1-NPLST)*DELT
C.VPP500e-----------
                .
                .
c.org DO 4200 I=NPLST,NTP
C.VPP500s-----------
       DO  4200 I= i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
C.VPP500e-----------
                .
                .
c.org  CALL RANU2 (IX, RAND, ICOUNT, ICON)
C.VPP500s----------
        CALL DP_VRANU4 (IX, RAND, ICOUNT, DWORK, NWORK, ICON)
C.VPP500e----------
c.org  DO 4310 I=NPLST,NTP
c.org   X(I) = X(I) + SQRT(24.0*DH(I)*DLT(I))*(0.5-RAND(I-NPLST+1))
C.VPP500s----------
        DO 4310 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
         X(I) = X(I) + SQRT(24.0*DH(I)*DLT(I))*(0.5-RAND(I-i_min(i_pe_num
     &)+1-NPLST+1))
C.VPP500e----------
                .
                .
c.org  CALL RANU2 (IX, RAND, ICOUNT, ICON)
C.VPP500s----------
        CALL DP_VRANU4 (IX, RAND, ICOUNT, DWORK, NWORK, ICON)
C.VPP500e----------
c.org  DO 4320 I=NPLST,NTP
c.org   Y(I) = Y(I) + SQRT(24.0*DH(I)*DLT(I))*(0.5-RAND(I-NPLST+1))
C.VPP500s----------
        DO 4320 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
         Y(I) = Y(I) + SQRT(24.0*DH(I)*DLT(I))*(0.5-RAND(I-i_min(i_pe_num
     &)+1-NPLST+1))
C.VPP500e----------
                .
                .
c.org  CALL RANU2 (IX, RAND, ICOUNT, ICON)
C.VPP500s----------
        CALL DP_VRANU4 (IX, RAND, ICOUNT, DWORK, NWORK, ICON)
C.VPP500e----------
```

Fig. 3.35 Modified DO loops and subscript of arrays in subroutine INIL2 (1/2).

```
c.org  DO 4330 I=NPLST,NTP
c.org    Z(I) = Z(I) + SQRT(24.0*DV(I)*DLT(I))*(0.5-RAND(I-NPLST+1))
C.VPP500s----------
       DO 4330 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
        Z(I) = Z(I) + SQRT(24.0*DV(I)*DLT(I))*(0.5-RAND(I-i_min(i_pe_num
     &)+1-NPLST+1))
C.VPP500e----------
                .
                .
c.org CALL  RANU2( IX, RAND, NZAN, ICON )
C.VPP500s----------
      CALL  DP_VRANU4( IX, RAND, NZAN, DWORK, NWORK, ICON )
C.VPP500e----------
      DO 4500  I = 1 , NZAN
        II(I) = INT( MAXP*RAND(I) ) + 1
        IF(II(I).GT.MAXP) II(I) = MAXP
C.VPP500s----------
        II(I)=II(I)+i_min(i_pe_num)-1
C.VPP500e----------
 4500 CONTINUE
                .
                .
c.org   CALL RANU2( IX, RAND, NZAN, ICON )
C.VPP500s----------
        CALL DP_VRANU4( IX, RAND, NZAN, DWORK, NWORK, ICON )
C.VPP500e----------
*VOCL LOOP,NOVREC
        DO 4810  I = 1, NZAN
                .
                .
c.org   CALL RANU2( IX, RAND, NZAN, ICON )
C.VPP500s----------
        CALL DP_VRANU4( IX, RAND, NZAN, DWORK, NWORK, ICON )
C.VPP500e----------
*VOCL LOOP,NOVREC
        DO 4820  I = 1, NZAN
                .
                .
c.org   CALL RANU2( IX, RAND, NZAN, ICON )
C.VPP500s----------
        CALL DP_VRANU4( IX, RAND, NZAN, DWORK, NWORK, ICON )
C.VPP500e----------
*VOCL LOOP,NOVREC
        DO 4830  I = 1, NZAN
                .
                .
c.org DO 320 IJ=1,NPLST
C.VPP500s----------
      DO 320 IJ=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e----------
                .
                .
```

Fig. 3.35 Modified DO loops and subscript of arrays in subroutine INIL2 (2/2).

```
                .
                .
c.org CALL RANU2 (IX, RAND, ICOUNT, ICON)
C.VPP500s-----------
      CALL DP_VRANU4 (IX, RAND, ICOUNT, DWORK, NWORK, ICON)
C.VPP500e-----------
Cc.org DO 4000 I=NPLST,NTP
c.org  X(I) = 2.0*SIGH0*RAND(I-NPLST+1) + X00 - SIGH0
C.VPP500s-----------
      DO 4000 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
       X(I) = 2.0*SIGH0*RAND(I-i_min(i_pe_num)+1-NPLST+1) + X00 - SIGH0
C.VPP500e-----------
                .
                .
c.org CALL RANU2 (IX, RAND, ICOUNT, ICON)
C.VPP500s-----------
      CALL DP_VRANU4 (IX, RAND, ICOUNT, DWORK, NWORK, ICON)
C.VPP500e-----------
c.org DO 4100 I=NPLST,NTP
c.org  Y(I) = 2.0*SIGH0*RAND(I-NPLST+1) + Y00 - SIGH0
C.VPP500s-----------
      DO 4100 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
       Y(I) = 2.0*SIGH0*RAND(I-i_min(i_pe_num)+1-NPLST+1) + Y00 - SIGH0
C.VPP500e-----------
                .
                .
      DO 4200 I=1,ICOUNT
c.org  IZ(I) = NPLST + I - 1
C.VPP500s-----------
       IZ(I) = i_min(i_pe_num)-1+NPLST + I-i_min(i_pe_num)+1 - 1
C.VPP500e-----------
 4200 CONTINUE
                .
                .
c.org DO 4500 I=NPLST,NTP
C.VPP500s-----------
      DO 4500 I=i_min(i_pe_num)-1+NPLST,i_min(i_pe_num)-1+NTP
C.VPP500e-----------
                .
                .
```

Fig. 3.36 Modified DO loops and subscript of arrays in subroutine VOL1.

```
                .
                .
c.org ICNT = 0
C.VPP500s--------------
      ICNT = i_min(i_pe_num)-1
C.VPP500e--------------
                .
                .
c.org do 4000 i=1,NPLST
C.VPP500s--------------
      do 4000 i=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e--------------
                .
                .
C.VPP500s--------------
      ICNT_tmp=ICNT-i_min(i_pe_num)+1
C.VPP500e--------------
c.org      IF( ICNT .EQ. NPLST )                          GO TO 3000
C.VPP500s--------------
      IF( ICNT_tmp .EQ. NPLST )                       GO TO 3000
C.VPP500e--------------
c.org      MAX = ICNT
C.VPP500s--------------
      MAX = ICNT_tmp
C.VPP500e--------------
                .
                .
c.org do 4100 i=1,max
C.VPP500s--------------
      do 4100 i=i_min(i_pe_num),i_min(i_pe_num)-1+max
C.VPP500e--------------
                .
                .
c.org do 5100 i=1,max
C.VPP500s--------------
      do 5100 i=i_min(i_pe_num),i_min(i_pe_num)-1+max
C.VPP500e--------------
                .
                .
```

Fig. 3.37 Modified DO loops and comparative method of the number of particles in subroutine PCKNG.

```
c.org COMMON /VPWORK/ J(IPP),K(IPP),L(IPP),JJ(IPP),KK(IPP),LL(IPP),
c.org.                A(IPP),B(IPP),C(IPP),D(IPP),E(IPP),F(IPP),
c.org.                G(IPP),H(IPP),SHEAR1(IPP),SHEAR3(IPP)
c.org COMMON /VPWRK2/ JG(IPP),KG(IPP),AA(IPP),BB(IPP)
c.org integer AA , BB
C.VPP500s--------------
      integer*4 j(ipp),k(ipp),l(ipp),jj(ipp),kk(ipp),ll(ipp),jg(ipp),
     &          kg(ipp),aa(ipp),bb(ipp)
      integer*4 j_g(ipp),k_g(ipp),l_g(ipp),jj_g(ipp),kk_g(ipp),
     &          ll_g(ipp),jg_g(ipp),kg_g(ipp),aa_g(ipp),bb_g(ipp)
      real*8 a(ipp),b(ipp),c(ipp),d(ipp),e(ipp),f(ipp),g(ipp),h(ipp),
     &shear1(ipp),shear3(ipp)
      real*8 a_g(ipp),b_g(ipp),c_g(ipp),d_g(ipp),e_g(ipp),f_g(ipp),
     &g_g(ipp),h_g(ipp),shear1_g(ipp),shear3_g(ipp)
!xocl local j(/p_sub),k(/p_sub),l(/p_sub),jj(/p_sub),kk(/p_sub)
!xocl local ll(/p_sub),jg(/p_sub),kg(/p_sub)
!xocl local a(/p_sub),b(/p_sub),c(/p_sub),d(/p_sub),e(/p_sub),f(/p_sub)
!xocl local g(/p_sub),h(/p_sub),shear1(/p_sub),shear3(/p_sub),aa(/p_sub)
!xocl local bb(/p_sub)
!xocl global j_g,k_g,l_g,jj_g,kk_g,ll_g,a_g,b_g,c_g,d_g,e_g,f_g,g_g,h_g
!xocl global shear1_g,shear3_g,jg_g,kg_g,aa_g,bb_g
      equivalence (j,j_g),(k,k_g),(l,l_g),(jj,jj_g),(kk,kk_g),(ll,ll_g),
     &            (a,a_g),(b,b_g),(c,c_g),(d,d_g),(e,e_g),(f,f_g),
     &            (g,g_g),(h,h_g),(shear1,shear1_g),(shear3,shear3_g),
     &            (jg,jg_g),(kg,kg_g),(aa,aa_g),(bb,bb_g)
C.VPP500e--------------
                .
                .
c.org  DO 5000 I = 1, NPLST
C.VPP500s--------------
       DO 5000 I = i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e--------------
                .
                .
c.org DO 5010 I=1,NPLST
C.VPP500s--------------
      DO 5010 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e--------------
                .
                .
```

Fig. 3.38 Modified DO loops in subroutine DIVIDL (1/2).

```
c.org DO 5100 I=1,NPLST
C.VPP500s---------------
      DO 5100 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e---------------
                .
                .
c.org DO 5200 I=1,NPLST
C.VPP500s------------
      DO 5200 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e------------
                .
                .
c.org DO 5300 I=1,NPLST
C.VPP500s------------
      DO 5300 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e------------
                .
                .
```

Fig. 3.38 Modified DO loops in subroutine DIVIDL (2/2).

```
c.org COMMON /VPWORK/ QD1(IPP),QD2(IPP),QD3(IPP),QD4(IPP),QD5(IPP),
c.org.                QD6(IPP),QD7(IPP),QD8(IPP),
c.org.                QDW1(IPP),QDW2(IPP),QDW3(IPP),QDW4(IPP),QDW5(IPP),
c.org.                QDW6(IPP),QDW7(IPP),QDW8(IPP)
c.org COMMON /VPWRK2/ J(IPP),K(IPP),JJ(IPP),KK(IPP)
C.VPP500s--------------------
      real*8 qd1(ipp),qd2(ipp),qd3(ipp),qd4(ipp),qd5(ipp),qd6(ipp),
     &qd7(ipp),qd8(ipp),qdw1(ipp),qdw2(ipp),qdw3(ipp),qdw4(ipp),
     &qdw5(ipp),qdw6(ipp),qdw7(ipp),qdw8(ipp)
      integer*4 j(ipp),k(ipp),jj(ipp),kk(ipp)
      real*8 qd1_g(ipp),qd2_g(ipp),qd3_g(ipp),qd4_g(ipp),qd5_g(ipp),
     &qd6_g(ipp),qd7_g(ipp),qd8_g(ipp),qdw1_g(ipp),qdw2_g(ipp),
     &qdw3_g(ipp),qdw4_g(ipp),qdw5_g(ipp),qdw6_g(ipp),qdw7_g(ipp),
     &qdw8_g(ipp)
      integer*4 j_g(ipp),k_g(ipp),jj_g(ipp),kk_g(ipp)
!xocl local qd1(/p_sub),qd2(/p_sub),qd3(/p_sub),qd4(/p_sub),qd5(/p_sub)
!xocl local qd6(/p_sub),qd7(/p_sub),qd8(/p_sub),qdw1(/p_sub)
!xocl local qdw2(/p_sub),qdw3(/p_sub),qdw4(/p_sub),qdw5(/p_sub)
!xocl local qdw6(/p_sub),qdw7(/p_sub),qdw8(/p_sub)
!xocl local j(/p_sub),k(/p_sub),jj(/p_sub),kk(/p_sub)
!xocl global qd1_g,qd2_g,qd3_g,qd4_g,qd5_g,qd6_g,qd7_g,qd8_g
!xocl global qdw1_g,qdw2_g,qdw3_g,qdw4_g,qdw5_g,qdw6_g,qdw7_g,qdw8_g
!xocl global j_g,k_g,jj_g,kk_g
      equivalence (qd1,qd1_g),(qd2,qd2_g),(qd3,qd3_g),(qd4,qd4_g),
     &            (qd5,qd5_g),(qd6,qd6_g),(qd7,qd7_g),(qd8,qd8_g),
     &            (qdw1,qdw1_g),(qdw2,qdw2_g),(qdw3,qdw3_g),
     &            (qdw4,qdw4_g),(qdw5,qdw5_g),(qdw6,qdw6_g),
     &            (qdw7,qdw7_g),(qdw8,qdw8_g),(j,j_g),(k,k_g),
     &            (jj,jj_g),(kk,kk_g)
C.VPP500e--------------------
                .
                .
c.org DO 5000 I=1,NPLST
C.VPP500s--------------------
      DO 5000 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e--------------------
                .
                .
c.org DO 5100 I=1,NPLST
C.VPP500s------------
      DO 5100 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e------------
                .
                .
c.org DO 5200 I=1,NPLST
C.VPP500s-----------------
      DO 5200 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-----------------
                .
                .
```

Fig. 3.39 Modified DO loops in subroutine DEPO (1/2).

```
c.org DO 5300 I=1,NPLST
C.VPP500s-----------------
      DO 5300 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-----------------
                .
                .
c.org DO 5400 I=1,NPLST
C.VPP500s-----------------
      DO 5400 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-----------------
                .
                .
```

Fig. 3.39 Modified DO loops in subroutine DEPO (2/2).

```
                .
                .
c.org DO 100 I=1,NPLST
C.VPP500s-----------------------
      DO 100 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e-----------------------
                .
                .
```

Fig. 3.40 Modified DO loop in subroutine PRDECAY.

```
                .
                .
      COMMON /SUM/ ASPC_S(IPCX,IPCY,NCELZ,NFP),DASPC_S(IPCX,IPCY,NFP)
                .
                .
      CALL RROUT
                .
C.VPP500s------------------
!xocl spread do
      do 123 ipe=1,npe
      DO 3123 L=1,NFP
      DO 3123 J=1,NCY
      DO 3123 I=1,NCX
      DO 3123 K=1,NCZ
      aspc_s(I,J,K,L) = aspc_s(I,J,K,L) + ASPC(I,J,K,L)
 3123 CONTINUE
  123 continue
!xocl end spread sum(aspc_s)
C
      do L=1,NFP
      do J=1,NCY
      do I=1,NCX
      do K=1,NCZ
      ASPC(I,J,K,L) = aspc_s(I,J,K,L)
      end do
      end do
      end do
      end do
C
!xocl spread do
      do 124 ipe=1,npe
      DO 3124 L=1,NFP
      DO 3124 J=1,NCY
      DO 3124 I=1,NCX
      daspc_s(I,J,L) = daspc_s(I,J,L) + DASPC(I,J,L)
 3124 CONTINUE
  124 continue
!xocl end spread sum(daspc_s)
C
      do L=1,NFP
      do J=1,NCY
      do I=1,NCX
      DASPC(I,J,L) = daspc_s(I,J,L)
      end do
      end do
      end do
C.VPP500e------------------
                .
      CALL REC
      WRITE(6,*) ' TFULL,NPLST , DELT'
                .
                .
```

Fig. 3.41 Global sum in **MAIN** routine.

```
Example 1:

                .
                .
c.org*VOCL LOOP,NOVREC
C.VPP500s----------
*VOCL LOOP,NOVREC,vdopt
C.VPP500e----------
        DO 4810  I = 1, NZAN
                .
                .

==========================================
Example 2:

                .
                .
c.org  DO 5000 I = 1, NPLST
C.VPP500s---------------
*vocl loop,vdopt
       DO 5000 I = i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
C.VPP500e---------------
                .
                .
```

Fig. 3.42 Examples of directive and addition of optimizing control statement VDOPT.

```
                .
                .
     NN = (NPLST+9)/10                                  ←
     WRITE ( 62 )   IDAY, ITIM, NN
     DO 100 I=1,NPLST,10
     WRITE ( 62 )                                       ←
    *         X(I)   , Y(I)   , Z(I), PTINDX(I)
 100 CONTINUE
                .
                .
```

Fig. 3.43 Outpur part of original subroutine RROUT.

```
MAIN routine :
                .
                .
C.VPP500s---------------
      i_tmp=0
      write(62) i_tmp
      rewind 62
C.VPP500e---------------
                .
                .

========================================

RROUT routine :
                .
                .
c.org NN = (NPLST+9)/10
C.VPP500s------------
      NN=0
!xocl spread do
      do 9000 ipe=1,npe
      NN = NN + (NPLST+9)/10
 9000 continue
!xocl end spread sum(NN)
C.VPP500e------------
      WRITE ( 62 )   IDAY, ITIM, NN
                .
                .
C.VPP500s------
      do 123 ip1=1,npe
!xocl spread do
      do 124 ip2=1,npe
      if(ip1.eq.ip2) then
      DO 100 I=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST,10
      WRITE ( 62 )
     *         X(I)  , Y(I)  , Z(I), PTINDX(I)
  100 CONTINUE
      endif
  124 continue
!xocl end spread
  123 continue
C.VPP500e------
                .
                .
```

Fig. 3.44 Output part for global file in subroutine RROUT.

```
MAIN routine :
                .
                .
C.VPP500s---------------
      i_tmp=0
      write(63) i_tmp
      rewind 63
C.VPP500e---------------
                .
                .

=========================================

RROUT routine :
                .
                .
c.org NN = (NPLST+9)/10
C.VPP500s------------
      NN=0
!xocl spread do
      do 9000 ipe=1,npe
      NN = NN + (NPLST+9)/10
 9000 continue
!xocl end spread sum(NN)
C.VPP500e------------
      WRITE ( 62 )   IDAY, ITIM, NN
                .
                .
C.VPP500s-------
      DO 100 ipe=1,npe
      write( 62 ) ( X_g(i),Y_g(i),Z_g(i), i=i_min(i_pe_num),i_min(i_pe_n
     &um)-1+NPLST,10 )
  100 continue
      do 101 ip1=1,npe
!xocl spread do
      do 102 ip2=1,npe
      if(ip1.eq.ip2) then
      write(62+1) ( PTINDX(i), i=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST
     &,10 )
      endif
  102 continue
!xocl end spread
  101 continue
C.VPP500e------
                .
                .
```

Fig. 3.45 Output part for local file and global file in subroutine RROUT.

```
                .
                .
      WRITE ( NOUT3, 6100 )   NTIMEC, NPLST                ←
                .
      write(6,*) ' In RSTOUT (write particle data): NPLST=',NPLST
                .
      DO  4100  I = 1, NPLST
C
C       WRITE ( NOUT3 )
        WRITE ( NOUT3, 6200 )                               ←
     *                        X(I), Y(I), Z(I), IMX(I), IMY(I),
     *                        IMZ(I), IRCY(I), NSTL(I),
     *                        PDT(I), PTINDX(I),(Q(I,J) , J=1,NCLID)
 4100 CONTINUE
                .
                .
```

Fig. 3.46 Output part of original in subroutine RSTOUT.

```
MAIN routine :
                .
                .
C.VPP500s---------------
      i_tmp=0
      write(NOUT3+1) i_tmp
      rewind NOUT3+1
C.VPP500e---------------
                .
                .

=========================================

RSTOUT routine :
                .
                .
C.VPP500s----------------------
      NPLST_total=0
!xocl spread do
      do 10 ipe=1,npe
      NPLST_total=NPLST_total+NPLST
   10 continue
!xocl end spread sum(NPLST_total)
C.VPP500e----------------------
c     WRITE ( NOUT3, 6100 )   NTIMEC, NPLST_total
      WRITE ( NOUT3 )   NTIMEC, NPLST_total
                .
      write(6,*) ' In RSTOUT (write particle data): NPLST=',NPLST_total
                .
C.VPP500s----------------------
      DO 4100 ipe=1,npe
      write( nout3 ) ( X_g(i),Y_g(i),Z_g(i),IMX_g(i),IMY_g(i),IMZ_g(i),I
     &RCY_g(i),NSTL_g(i) ,i=i_min(i_pe_num),i_min(i_pe_num)-1+NPLST )
 4100 continue
      do 4101 ipe=1,npe
      write( nout3 ) ( PDT_g(i) ,i=i_min(i_pe_num),i_min(i_pe_num)-1+N
     &PLST )
 4101 continue
      do 4102 ipe=1,npe
      write( nout3 ) ( (Q_g(i,j),j=1,NCLID) ,i=i_min(i_pe_num),i_min(i
     &_pe_num)-1+NPLST )
 4102 continue
```

Fig. 3.47 Output part for local file and global file in subroutine RSTOUT (1/2).

```
      do 4200 ip1=1,npe
!xocl spread do
      do 4300 ip2=1,npe
      if(ip1.eq.ip2) then
      write( nout3+1 ) ( PTINDX(i),i=i_min(i_pe_num),i_min(i_pe_num)-1+N
     &PLST )
      endif
 4300 continue
!xocl end spread
 4200 continue
C.VPP500e----------------------
                .
                .
```

Fig. 3.47 Output part for local file and global file in subroutine RSTOUT (2/2).

```
                .
                .
      READ( NOUT3 )   NTIMEF, NPLST
                .
                .
C.VPP500s----------------
      DO 4100 ipe=1,npe
      read( nout3, END=9998 ) ( X_g(i),Y_g(i),Z_g(i),IMX_g(i),IMY_g(i),I
     &MZ_g(i),IRCY_g(i),NSTL_g(i) ,i=i_min(i_pe_num),i_min(i_pe_num)-1+N
     &PLST )
 4100 continue
      do 4101 ipe=1,npe
      read( nout3+1, END=9998 ) ( PDT_g(i) ,i=i_min(i_pe_num),i_min(i_pe
     &_num)-1+NPLST )
 4101 continue
      do 4102 ipe=1,npe
      read( nout3+2, END=9998 ) ( (Q_g(i,j),j=1,NCLID) ,i=i_min(i_pe_num
     &),i_min(i_pe_num)-1+NPLST )
 4102 continue
      do 4200 ip1=1,npe
!xocl spread do
      do 4201 ip2=1,npe
      if(ip1.eq.ip2) then
      read( nout3+3, END=9998 ) ( PTINDX(i),i=i_min(i_pe_num),i_min(i_pe
     &_num)-1+NPLST )
      endif
 4201 continue
!xocl end spread
 4200 continue
C.VPP500e----------------
                .
                .
```

Fig. 3.48 Output part for local file and global file in subroutine PERCEL.

```
                .
                .
C.VPP500s-----------
!xocl spread region /subp(1)
C.VPP500e-----------
      CALL ...................
     &     ...................
C.VPP500s-----------
!xocl end spread
C.VPP500e-----------
                .
                .
```

Fig. 3.49 Directive to call C language routine only on 1PE .

```
REC routine :
                .
                .
      DBN='INCONC'
      DBND='SFCONC'
      NOSITE=0
      SHG=0
      ZHG4=0.
      NFLG=0
C
      HPASS='/dg03/ufs09/j9145/wkvfl'     ←
                .
                .
-----------------------------------------------
RECDS routine :
                .
                .
      DBNA = 'DOSEGM  '
      DBNE = 'DOSECUM '
      DBNI = 'DOSEINH '
      ALLG4  = XMW*NXC/1000.
      ALLT4  = YMW*NYC/1000.
      ZLT4   = YMTMIN/1000.
      ZLG4   = XMTMIN/1000.
      LISTA = 1
      LISTE = 3
      LISTI = 6
      NFLG  = 0
      write(6,*) NCLIDE,(NAMEX(IO),IO=1,NCLIDE)
      write(6,*) NCLIDI,(NAMEI(IO),IO=1,NCLIDI)
C
      HPASS='/dg03/ufs09/j9145/wkvfl'     ←
                .
                .
```

Fig. 3.50 Modified output file path for C language routines of DENSITY/DOSE code.

参 考 文 献

[1] 茅野正道・林　隆・石川裕彦・横川三津夫；小型計算機への導入を想定した実時間大気拡散・被曝評価数値計算コードの開発，JAERI-M 90-173，1990年10月.

[2] 日本原子力研究所環境放射線物理研究室；SPEEDIの計算の流れと計算モデル，私信，1991年3月.

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

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

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

[6] 「富士通　SSLII使用手引書（科学用サブルーチンライブラリ）」，富士通(株)，1987年12月.

[7] 「UXP/M SSLII/VPP使用手引書（科学用サブルーチンライブラリ） V10用」，富士通(株)，1994年3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.

4. EQMD コードのベクトル並列化

拡張量子分子動力学コード EQMD の高速化作業を行った．本コードはベクトル化による高速化を配慮してコーディングされている．しかし，実行時の cpu 時間は運動方程式を解くときの有効相互作用を計算する箇所に集中し，対象原子の質量数の増加と共に指数関数的に増加している．例えば，質量数が 200 近辺では優に 10 時間を越え，シミュレーション効率を悪化させることが予想できる．そのため，更に高速化が必要であった．今回対象にした EQMD コードには厳密版と近似版の二つの版があり，本報告書では近似版の高速化について記述している．

高速化作業は富士通 (株) 製ベクトルパラレルコンピュータ VPP500 (以下，VPP と呼ぶ) 上で行い，高速化の要点は共通計算項の集約化による計算量の削減，ベクトル化，及び並列化の 3 点であった．また，ジョブキューの混雑等諸般の事情により高速化の効果を発揮できないことも考えられる．そこで，こうした事を幾分でも軽減できるように富士通 (株) 製分散メモリ型並列サーバ AP3000 への適応性についても検討した．

以下に，これらの作業内容について報告する．

4.1 コード概要

EQMD コード [1] は原子核反応のシミュレーションコードであり，1 核子を一つの波束で表現して，古典運動方程式及び 2 体衝突により核子多体系の時間発展を記述している．EQMD の特徴は，普通は固定している波束の拡がりを動的に扱う点にある．波束の中心についてはニュートン方程式を解き，波束の拡がりについてはニュートン方程式に良く似た運動方程式を解いている．波束の拡がりを動的に扱うようにしたためエネルギー保存が差分の時間発展では悪くなるので，4 次の Runge-Kutta 法を用いている．2 体衝突については，各々の核子につき最近接の核子をみつけてランダムな散乱を行い，破れたエネルギーを戻すため散乱後の運動量を調節している．

4.2 動的挙動解析

高速化作業を進めるにあたり，計算コストの分布状況，及びベクトル化状況を調査した．本高速化作業の全体を通して利用したテストデータは，質量数が 12，40，100，197 の 4 種類である．それぞれに対応するテストケースを以降 T012，T040，T100，T197 と呼ぶことにする．

(1) 計算コストの分布状況

オリジナル版ベクトル実行のコスト分布を調査するため，動的挙動解析ツール SAMPLER [2]，及び ANALYZER [3] を利用した．SAMPLER は VPP のバックエンドシステムの PE 上で動作するツールであり，これによりルーチン単位の計算コスト分布が得られる．一方，ANALYZ-ER はフロントエンドシステム (GSP) 上で動作するツールであり，これによりルーチン単位，

ループ単位，文単位の計算コスト，実行回数，及びループの回転数等の情報が得られる．ルーチン単位の計算コスト分布に関しては，EQMD が実際に動作するのが PE であるため，SAMPLER のものの方がより実情に近い値を示している．

テストケース T012，T040 における SAMPLER の出力結果を Table 4.1 に示す．計算コストが CALF ルーチンに集中しており，その比率は質量数が 12 の場合 97.5%，40 の場合 98.9% である．本ルーチンは運動方程式を解くときの有効相互作用を計算しているが，3 体力を使っているため対象核子の質量数の 3 乗のオーダーで計算量が増加する．従って，質量数が大きくなるに従いその比率が大きくなっていくことが予測できる．T012 における CALF ルーチンに関する ANALYZER の出力結果 (抜粋) を Table 4.2 に示す．DO 10・20，DO 20・30 の二重ループ，及び DO 33・43・45 の三重ループに計算コストが集中している．CALF ルーチン内のより正確なコスト比率を得るため，これらループの実行時間をサービスサブルーチン CLOCKV を利用して測定した．T012，T040 における CALF ルーチン内のコスト比率を Table 4.3 に示す．DO 33・43・45 の三重ループに計算コストが集中しており，質量数が大きくなるに従いその比率が大きくなっている．以上のことは CALF ルーチンの処理構造，及び CALLER/CALLEE 関係を見ることにより理解できる．この関連図を Fig. 4.1 に示す．

(2) ベクトル化状況

オリジナル版のベクトル化率を知るため，テストケース T012，T040 について，オリジナル版のスカラ実行とベクトル実行の実行時間を測定した．Table 4.4 にその結果と対応するベクトル化率を示す．ベクトル化率は 98% を越え，質量数が大きくなるに従い，大きくなっている．また，性能についても，T040 の場合，CPU 時間が対スカラ実行比 19.3 倍であり，質量数が大きくなるに従い，向上している．つまり，オリジナル版は性能向上が図られており，十分にベクトル化されていることが判る．

4.3 チューニング

「4.2 動的挙動解析」からも判るように，本コードを高速化するには CALF ルーチンの計算コストを下げること，特にその中の三重ループ DO 33・43・45 の計算コストを下げることが最も重要な課題となる．以下にこのコストダウンの内容について記述する．

4.3.1 三重ループ DO 33・43・45 の計算量削減

DO 33・43・45 のオリジナルソースコードを Fig. 4.2 に示す．ANALYZER の結果より，ループ内の計算コストの状況は Fig. 4.2 の①，②の文に集中していることが判った．T012 では，70% を占めていた．実際，サービスサブルーチン CLOCKV による実測値では，CPU 時間の比率は T012 で 85%，T040 で 89% であり，ベキ乗計算にコストを費やしていることが判る．そこで，本ループのコストダウンを図るため①，②を数式的に同等な以下の文に変形した．

```
      DU3DIJ=-C3*FAC*2./3.*(2-SMASKIJ(I,J))
     &  *(RHORHO(J,K)**(2./3.))*(RHORHO(I,K)**(2./3.))*(RHORHO(I,J)**(-1./3.))
```

```
DU3DIK=-C3*FAC*2./3.
&  *(RHORHO(J,K)**(2./3.))*(RHORHO(I,J)**(2./3.))*(RHORHO(I,K)**(-1./3.))
```

RHORHO(I,J)**(2./3.)，RHORHO(I,J)**(-1./3.) のベキ乗計算が三重ループの中で冗長に行われるということが判る．従って，このベキ乗計算だけを事前に行うことにより計算量を削減できる．つまり，それぞれについて $2 \times$ MASNUM^3 回から MASNUM^2 回に削減できる．

同様に，Fig. 4.2 の③～⑤の文の中で以下の形の計算項を事前に計算することにより，それぞれについてて $2 \times$ MASNUM^3 回から MASNUM^2 回に削減できる．

```
(X(I,i)-X(J,i))*DRHODR(I,J)   i=1,2,3
```

また，この三重ループは，J，K に関して回帰演算となっている総和計算 (Fig. 4.2③～⑤) を含むため，一重化によるベクトル化はできない．従って，更にコストダウンを図るため，できるだけデータの連続引用が行えるように DO ループの順序を変更した．ループ内の処理は，各 DO 変数 I，J，K の間の依存関係がないため，DO ループの順序を変えても，数式的には，同等の結果が得られることが判る．そのため，配列の第 1 次元めに利用されている添字変数 I を DO 変数とする DO ループが最内ループとなるように変更した．

これらの対応により CPU 時間が T012 で 1/2.6，T040 で 1/5.0 となった．DO 33・43・45 の変更内容を Fig. 4.3 に示す．

4.3.2 二重ループ DO 30・40 の計算量削減

DO 30・40 のオリジナルソースコードの抜粋を Fig. 4.4 に示す．本ループの特徴は (PP−1) 乗を含む文 (Fig. 4.4 ①，②，及び他に 4 文) に計算コストの集中が見られる (ANALYZER の結果では T012 で 40% 占有) こと，J に関して回帰演算となっている総和計算 (Fig. 4.4③～⑩) を含むため，一重化によるベクトル化ができないことが挙げられる．また，内側ループの DO 40 は J に関してベクトル化されており，データの連続引用という点で改善の余地がある．

従って，ここでは，三重ループ DO 33・43・45 の場合と同様に，ベキ乗計算の計算量の削減及び DO ループの順序変更を行った．つまり，一つは DIM(F(I)−F0,0.0)**(PP−1) のベキ乗計算を事前に行うことにより計算量の削減を図った．これにより，計算回数を $(2 \times$ MASNUM$^2) \times 6$ 回から MASNUM 回に削減できた．二つめは，DO 30 と DO 40 を入れ換えることにより，ベクトル変数を J から I に変更した．ループ内の処理は，各 DO 変数 I，J の間に依存関係がないため，DO ループの順序を変えることにより計算の順序を変えても数式的に同等の結果が得られる．またこの時，Fig. 4.4⑪の処理は最内ループの計算に影響しないため，DO 30・40 の二重ループとは分離し，その処理の後にベクトル処理するよう変更した．

DO 30・40 の変更内容を Fig. 4.5 に示す．

4.3.3 二重ループ DO 10・20 のベクトル化

DO 10・20 のオリジナルソースコードを Fig. 4.6 に示す．本ループでは配列 RNORM，DIST2 等 10 の配列を定義しており，主要な部分は各配列の下三角部分の要素を定義するところである．

ある行(I)について，その行方向(J: 行内昇順)に計算が行われていて，Jについてベクトル化されているため，連続引用になっていない．またこのループの平均ベクトル長は(MASNUM+1)/2であり，質量数MASNUMが小さい範囲ではベクトル化の効果が期待できない．一方，一連の下三角部分の処理部分をみてみると，例えばFig. 4.6の①，②のように，同一のデータに対する定義引用関係があったとしても，回帰参照になっていない．一文一文をDOループとして分離しても数式的に同等の結果が得られることが判る．仮にそれぞれの文をDOループとして分離した場合，DO変数I，Jの各値に対応する定義データの値は，引用する配列要素が対応するI，Jにのみ依存するため，DOループの実行順序に依存しない．つまり本二重ループDO 10・20は，このループの実行順序に関して，回帰参照を含まず一重化によるベクトル化が可能であるということが判る．

従って，本ループの高速化方法として一重化を採用した．一重化にあたっては，オリジナルDOループの実行順に一重化した．前処理として以下に示すような基本的な計算項を抽出し，順序付けを行った．次に，ループ本体を一重化し，計算結果設定の後処理を行った．

DO 10・20の変更内容をFig. 4.7に示す．

(1) WLAM(I)+WLAM(J)
(2) WLAM(I)*WLAM(J)
(3) DELTA(I)−DELTA(J)
(4) X(I,1)−X(J,1)
(5) P(I,1)−P(J,1)
他 8項

4.3.4 並列化

4.3.1～4.3.3の対応を行ったCALFルーチン及びその主要ループのコスト比率をサービスサブルーチンCLOCKVを利用して測定した．その結果をTable 4.5に示す．オリジナル版の場合と同様，コスト比率は依然としてCALFルーチンに集中している．質量数が大きくなるに従い，CALFルーチンの比率が大きくなり，DO 45・43・33の三重ループの比率が大きくなっている．これは，この三重ループの比率が大きくなっていくのでCALFルーチンの比率が大きくなっていくのであり，Fig. 4.1からも予測できることである．一方，DO 10・20，DO 30・40の二重ループはその比率を下げている．従って，少なくともDO 45・43・33の三重ループは，ある質量数以上のところで並列化の対象として期待できる．

並列化を行って効果が期待できるのは，CALFルーチンまたはDO 45・43・33の三重ループを含む部分であり，Fig. 4.1における以下の3箇所である．

(1) Mainルーチンにおける DO 20，DO 30
(2) TEVOLVルーチンにおけるループ 1000
(3) CALFルーチンにおける DO 45・43・33

Main ルーチンにおける DO 20, DO 30 は時間ステップのループである．時間ステップ n+1 の計算は時間ステップ n のデータを用いて累積的に行っていき，時間発展を記述している．従ってこれらのループを並列化することはできない．同様に，TEVOLV ルーチンにおけるループ 1000 は，T0 から T1 までの時間発展を追っており並列化できない．3 番目の DO 45・43・33 の三重ループは，Fig. 4.3 からも判るようにインデックス I，J，K のいずれにおいても分割可能である．例えば，インデックス I について言えば，処理結果は I に関する計算順序に依存しないからである．J，K についても同様のことが言える．並列化にあたってはデータの連続引用を考慮してインデックス K に関して分割することにした．

インデックス K に関して並列処理を行う場合，それを協同して行う各 PE はそれぞれが分担する K の範囲の処理を行うため，計算結果は K の範囲に対応して各 PE に分散されてしまう．その中でも配列 DHDR，DHDL の各要素の値はこの並列処理後に参照されるため，各 PE が保持する値を合計し (全ての K の値について総和計算をする)，全 PE でこの総和値を保持するようにしなければならない．こうした PE 間のデータ整合を行うためにグローバル関数 SUM を利用した．

本並列化の変更内容を Fig. 4.8 に示す．

4.4 計算結果の評価及び効果

4.4.1 計算結果の評価

VPP 上で FORTRAN プログラムを実行する場合，同じソースコードを用いたとしても，スカラ演算とベクトル演算との丸め方法の違い，コンパイラによる最適化方法の違いにより計算結果に違いが生じる．計算の実行順序を変更すればなおさらである．今回の変更においては，計算量の削減等，計算の実行順序を変更しており，変更前と後で計算結果に違いを生じることが予想できる．ここでは，この違いが変更誤りによるものかどうかを評価する作業を行った．

評価は 4.3.1～4.3.4までの変更内容を順に取り込みながら段階的に行った．比較対象とした出力データは，標準出力に出力されたエネルギー等の 4 データ及びプロット図である．まず，時間ステップ 0 の時 (初期値) の 4 データの値が一致することを確認した．次に時間ステップ 2000 の時 (最終値) の 4 データの値がどの程度一致するかを確認し，プロット図が一致することを確認した．時間ステップ 0 の値が一致しなければ，変更に誤りがあると判断できる．時間ステップ 2000 の値が頭から 4 ～ 5 桁以上一致しないかプロット図が一致しなければ，比較対象にした版のそれぞれの倍精度版の出力結果とを比較した．この時，時間ステップ 2000 の値が頭から 4 ～ 5 桁以上一致しないかプロット図が一致しなければ，変更に誤りがあると判断した．

評価の基準としたものは Org(ベクトル) 版の出力データである．最初に Org(ベクトル) 版と Vec1－0 版を比較して Vec1－0 版の妥当性を確認した．次に Vec1－0 版と Vec1 版，Vec1 版と Vec2 版，Vec2 版と Vec3 版，Vec3 版と Vec3.para 版というように順を追って比較を行った．各版の内容は以下のとおりであり，使用テストケースは T012，T040，T100，T197 の 4 種類である．

(1)Org 版　　：高速化の対象となったソースコード (オリジナル版)

(Org(ベクトル) はオリジナル版のベクトルコンパイル版)

(2)Vec1－0 版 ： Org 版に 4.3.1の変更を行った版
(但し，DO ループの順序変更は行っていない)

(3)Vec1 版 ： Vec1－0 版で DO ループの順序変更まで行っている版

(4)Vec2 版 ： Vec1 版に 4.3.2の変更を行った版

(5)Vec3 版 ： Vec2 版に 4.3.3の変更を行った版 (ベクトル化版)

(6)Vec3.para 版 ： Vec3 版に 4.3.4の変更を行った版 (並列化版)

以上の評価結果を Table 4.6 に示す．倍精度版まで考慮した結果比較では，プロット図も一致し時間ステップ 2000 の値が少なくとも頭から 8 桁以上一致している．従って当初の差異は，変更誤りによるものというよりは計算誤差によるものと判断した方が妥当である．

4.4.2 高速化効果

オリジナル版，及びチューニング版の実行時間の測定値を Table 4.7 に示す．また，ベクトル化版及び並列化版の高速化効果として実行時間の比 (倍率) を Table 4.8 に示す．

Table 4.4，4.7，4.8 からも判るように，オリジナル版は十分にベクトル化されている．ベクトル化率 99% 以上，倍率にして 5.73，19.30 と質量数が大きくなるに従いさらに大きくなることが予想できる．こうした状況の中，「4.3 チューニング」による変更により更に高速化を図ることができた．特に効果のあった変更は，Table 4.7 からも判るように，一つは三重ループ DO 33・43・45 内のベキ乗計算の計算量削減 (Org 版→ Vec1 版の変更) であり，二つめが三重ループ DO 33・43・45 の並列化 (Vec3 版→ Vec3.para 版の変更) であった．これらの効果は質量数が大きくなるに従い向上している．Fig. 4.1 から判るように三重ループ DO 33・43・45 のコストは，スカラ実行の場合ほぼ質量数の 3 乗に比例して増加 (ベクトル実行の場合には 2 ～ 3 乗の割合で増加) しており，質量数が増加するに従い，この三重ループのコスト比率が高くなっていく．例えば，オリジナル版であれば，T040 で約 90%(Table 4.1，4.3)，Vec3 版であれば，T197 で約 82%(Table 4.6) である．従って，三重ループに対するこの二つの変更が効果を発揮した．

高速化の効果としては，最大テストケース T197 の実行時間の評価も必要である．実際に使用されるモデル，ないしはそれに近いモデルの実行時間が実用的でなければ，倍率効果が大きくても意味をなさないからである．実際に利用されるモデルは質量数が 200 以内のどのケースも考えられるため，今回の最大テストケース T197 の場合を参考にして，T200 の実行時間を予測した．実行時間は，高々，質量数の比の 3 乗倍程度であるため，ベクトル化版の場合約 87 分 (CPU 時間) であり，並列化版の場合約 35 分 (経過時間) である．実用に耐えうるものであると考える．

4.5 AP3000 への適応性

(1) EQMD コードの AP3000 上での実行

AP3000 上では逐次実行版と並列実行版の二つを実行した．VPP でチューニングした Vec3 版を逐次実行版として実行した．Vec3.para 版は並列処理言語 VPP FORTRAN により記述し

ているが，並列実行版はVec3.para 版を参考にしてMPI [4] を利用して書き換えている．MPIは欧米の大学，研究機関，ベンダ，ユーザで組織されたMPI フォーラムで仕様が決められている並列処理用メッセージパッシングライブラリである．本作業で使用したAP3000 用のMPI はMPI/AP V1.0 [5] であり，MPI 1.1 に準拠している．AP3000 並列実行版のVec3.para 版との主な違いは，MPI ライブラリを利用していることの他に，三つある．一つはDO 20・30 を並列化した点である．二つ目はファイルへの出力をプロセッサ0 番のみとした点である．三つ目はDO 10・20 に関して，スカラ実行における計算量を考慮して，オリジナル版を利用した点である．並列化対応の変更内容の抜粋をFig. 4.9，Fig. 4.10 に示す．

並列実行版の動作確認は逐次実行版と並列実行版の結果を比較することにより行った．VPPの場合と同様に，標準出力に出力されたエネルギー等の4 データ及びプロット図である．使用テストケースはT012，T040，T100 の3 種類である．いずれのテストケースにおいてもプロット図が一致し，時間ステップ2000 の値が頭から5 ～9 桁一致していたため計算結果は妥当と判断した．

逐次実行版と並列実行版の実行時間をVPP のものと対比してTable 4.9 に示す．ここでT197の値には予測値を示している．実行時間の比は，高々，質量数の比の3 乗倍程度であることを利用している．

(2) 実行時間の評価

Vec3(スカラ) 版，Vec3(ベクトル) 版，及びAP3000 逐次版は同じソースコードを利用しているので，これらの実行時間の比較からベクトル化効果，スカラ実行におけるAP3000 の速度向上効果がうかがえる．Vec3(スカラ) 版のCPU 時間に対する，他の版のCPU 時間の比 (倍率) をTable 4.10 に示す．AP3000 上での実行は，3.3 倍，4.3 倍，・・とVPP 上のスカラ実行に比べて，速度向上している．一方VPP 上のベクトル実行は6.0 倍，21.7 倍，・・と，それ以上に速度向上している．その差は質量数が大きくなるに従い広がることが予想できる．ベクトル化効果の方がAP3000 環境による速度向上を凌ぐ例である．AP3000 逐次版はVec3(ベクトル) 版と比較して実用的ではないと言える．

AP3000 並列版については，Table 4.10 からも判るように，4 ノードの並列実行では実用的とは言い難い．VPP の実行に対抗できるためには，更に並列化効果を引き出すことが要求される．少なくとも最大クラスの実行が可能でなければ，VPP が故障，混雑した時の代替手段として検討することはできない．

4.6 まとめ

EQMD コードにおける計算コスト分布の特徴は，CALF ルーチンのDO 33・43・45 の三重ループ (3 体力を用いて相互作用を計算しているループ) に集中していることであった．そして，その集中度は質量数が大きくなるに従い増加していくものであった．そのため，高速化の方法としてベキ乗計算の計算量削減，及びこのループの並列化が有効であった．実行時間の点では，ベクトル化版，並列化版のいずれの場合も，不都合のないところまで高速化できた．そして，実行時間だけを比較すると，並列化版の方が更に高速化されている．質量数100 以上のテストケー

スにおけるそれぞれの実行時間の測定値をTable 4.11 に示す．一方，ジョブの投入から実行開始までのキュー上での待ち合わせ時間を含めたターンアラウンドタイムに関しては，混雑等の影響を考慮しなければならないため，必ずしも並列化版優位になるとは限らない．キュークラスの定義に依存するが，実行時間が長いジョブ，同時に使用するPE数が多いジョブほど混雑時の影響を受け易い．例えば，小ジョブクラスの境がCPU 時間で30 分であれば，Table 4.11 から質量数130 近辺までは，ベクトル化版優位の可能性が高い．混雑等を見極めた使い分けが必要である．

また，混雑等については，その代替手段としてスカラ並列マシンAP3000 上の実行を検討した．結果として，ベクトル化効果の方がAP3000 の効果を上回っており現実的な解が得られなかった．EQMD コードの場合は，ベクトル並列化向きコードであるということが判った．

Table 4.1 Computational cost distributions of original version (vector compilation).

テストケース： T012

Synthesis Information

Count	Percent	VL	Nam
8962	97.5	11	calf__
128	1.4	89	bnde__
41	0.4	12	tevolv__
30	0.3	-	MAIN___
6	0.1	12	totl__
5	0.1	3	invrs __
4	0.0	-	rmvl__
3	0.0	-	scclst__
2	0.0	3	rmsrad__
2	0.0	12	calovr__
2	0.0	-	dsply __
1	0.0	-	__star
1	0.0	-	locate__
1	0.0	-	boost __
9188		12	TOTAL

テストケース： T040

Synthesis Information

Count	Percent	VL	Nam
87519	98.9	39	calf__
757	0.9	34	bnde__
75	0.1	40	tevolv__
45	0.1	-	MAIN___
37	0.0	40	calovr__
13	0.0	-	scclst__
5	0.0	-	deform__
2	0.0	-	invrs __
2	0.0	-	rmsrad__
2	0.0	-	dsply __
1	0.0	-	rmvl__
1	0.0	-	boost __
88459		39	TOTAL

Table 4.2 Dynamic behavior of original version (vector compilation) in case of T012.

vectorize - routine list

routine	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect	overhd
CALF	18180	.1429E11	99.7	.1975E11	98.9	12	34.1	1.3- 1.4	0

vectorize - loop list ------- CALF

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00001276-00001298	45	do	S	2617920	.1294E11	90.6	.1294E11	65.5	12	0.0	1.0- 1.0
00001140-00001259	40	do	V	218160	.8776E09	6.1	.5704E10	28.9	12	100.0	3.9- 9.1
00001058-00001110	20	do	V	218160	.3780E09	2.6	.9827E09	5.0	7	100.0	1.3- 3.9
00001117-00001119	21	do	S	218160	54976320	0.4	54976320	0.3	12	0.0	1.0- 1.0
00001275-00001299	43	do	S	218160	20943360	0.1	20943360	0.1	12	0.0	1.0- 1.0
00001123-00001135	31	do	V	18180	5403655	0.0	35123760	0.2	12	100.0	3.9- 9.1
00001001-00001304	-----	total		18180	3472380	0.0	3472380	0.0	1	0.0	1.0- 1.0
00001057-00001111	10	do	S	18180	1745280	0.0	1745280	0.0	12	0.0	1.0- 1.0
00001139-00001265	30	do	V	18180	1493557	0.0	3708720	0.0	12	70.6	2.1- 2.7
00001114-00001120	11	do	V	18180	1325742	0.0	2617920	0.0	12	58.3	1.8- 2.1
00001274-00001300	33	do	V	18180	1191489	0.0	1745280	0.0	12	37.5	1.4- 1.5

Table 4.3 Computational costs of main loops in subroutine CALF.

ﾃｽﾄｹｰｽ		T012	T040
DO 10・20	CPU VU	9.8 8.6	3.3 3.0
DO 30・40	CPU VU	17.1 13.8	6.4 5.2
DO 33・43・45	CPU VU	71.3 77.5	90.1 91.8

値はCALFルーチンの中での比率である（単位：%）

Table 4.4 Vectorization ratio of original version.

ﾃｽﾄｹｰｽ			T012	T040
ベクトル化率(%)			98.94 %	99.86 %
参考	オリジナル版 スカラ実行	CPU VU	8:29.77 ——	4:36:15.15 ——
	オリジナル版 ベクトル実行	CPU VU	1:28.90 1:23.50	14:18.69 13:56.20

(1)ベクトル化率＝ $\{t_{sc}-(t_{vc}-t_{vu})\}/t_{sc}\times 100$
t_{sc}：オリジナル（ｽｶﾗｺﾝﾊﾟｲﾙ）版を実行した時のCPU 時間
t_{vc}：オリジナル（ﾍﾞｸﾄﾙｺﾝﾊﾟｲﾙ）版を実行した時のCPU 時間
t_{vu}：オリジナル（ﾍﾞｸﾄﾙｺﾝﾊﾟｲﾙ）版を実行した時のVU 時間
(2) CPU、VUの単位：時：分：秒.XX

Table 4.5 Computational costs of subroutine CALF and its main loops in vectorized version.

ﾃｽﾄｹｰｽ	T012	T040	T100	T197
CALFルーチン*1	88.0	92.1	93.2	93.6
DO 10・20*2	13.4	8.2	5.4	3.4
DO 30・40*2	41.9	30.0	15.7	8.6
DO 45・43・33 *2	38.5	61.3	78.6	87.9

＊1：CALFルーチンの全体の中での比率（%）
＊2：CALFルーチンの中での比率（%）

Table 4.6 Evaluation of execution results.

ﾃｽﾄｹｰｽ	T012	T040	T100	T197
Vec1-0版 ※Org(ﾍﾞｸﾄﾙ) 版 との比較	単:7桁, 図○	単:7桁, 図○	単:3~4桁, 図△ 倍: 12~14 桁, 図○	単:2~3桁, 図× 倍: 11~12 桁, 図○
Vec1版 ※Vec1-0版 との比較	単:6~9桁, 図○	単:6~7桁, 図○	単:2~4桁, 図× 倍: 10~11 桁, 図○	単:2~3桁, 図× 倍: 10~11 桁, 図○
Vec2版 ※Vec1版 との比較	単:6~9桁, 図○	単:6~9桁, 図○	単:2~3桁, 図× 倍: 10~12 桁, 図○	単:2~4桁, 図× 倍:10 ~11 桁, 図○
Vec3版 ※Vec2版 との比較	単:5~9桁, 図○	単:6~9桁, 図○	単:5~9桁, 図○	単:5~6桁, 図○
Vec3.para 版 ※Vec3版 との比較	単:5~7桁, 図○	単:4~6桁, 図○	単:1~3桁, 図× 倍: 10~11 桁, 図○	単:1~3桁, 図× 倍: 8 桁, 図○

(1) 単：xx桁は単精度版の比較において頭からxx桁一致していることを示す。
(2) 図○は出力図が一致していることを示す。図×は一致していないことを示す。
図△は異なる箇所が数箇所。
(3) 倍：xx桁は倍精度版の比較を示す。xxは単精度と同様。

Table 4.7 Execution time on VPP500.

ﾃｽﾄｹｰｽ	T012	T040	T100	T197
Org (ｽｶﾗ・ｺﾝﾊﾟｲﾙ) 版 CPU	8:29.77	4:36:15.15	——	——
VU	——	——	——	——
Org (ﾍﾞｸﾄﾙ・ｺﾝﾊﾟｲﾙ)版 CPU	1:28.90	14:18.69	1:55:16.23	11:46:25.46
VU	1:23.50	13:56.20	1:53:35.82	11:41:29.74
Vec1(ﾍﾞｸﾄﾙ・ｺﾝﾊﾟｲﾙ)版 CPU	33.77	2:51.42	16:02.09	1:26:09.40
VU	28.18	2:35.73	15:24.65	1:24:57.71
Vec2(ﾍﾞｸﾄﾙ・ｺﾝﾊﾟｲﾙ)版 CPU	29.21	2:32.68	15:02.41	1:23:12.41
VU	25.89	2:24.29	14:42.23	1:22:32.29
Vec3(ﾍﾞｸﾄﾙ・ｺﾝﾊﾟｲﾙ)版 CPU	24.48	2:16.74	14:35.67	1:22:40.85
（ベクトル化版） VU	21.07	2:07.86	14:15.11	1:22:01.06
Vec3.para 版 経過時間 （並列化版）	42.10	1:32.38	7:04.07	33:07.76

(1) CPU 時間、VU時間、経過時間の単位： 時：分：秒.XX
(2) Vec3.para 版の値は4PE の時のものである。

Table 4.8 Speed up ratio.

No.	A	B	T012	T040	T100	T197
1	Org (ｽｶﾗ)	⇨Org (ﾍﾞｸﾄﾙ)	5.73	19.30	——	——
2	Org (ﾍﾞｸﾄﾙ)	⇨Vec3(ﾍﾞｸﾄﾙ)	3.63	6.28	7.90	8.54
3	Org (ｽｶﾗ)	⇨Vec3(ﾍﾞｸﾄﾙ)	20.8	121.2	——	——
4	Vec3(ﾍﾞｸﾄﾙ)	⇨Vec3.para	0.87	1.54	2.15	2.50
5	Org (ﾍﾞｸﾄﾙ)	⇨Vec3.para	3.30	9.43	16.4	21.4

(1)値（倍率）は（Aの実行時間）÷（Bの実行時間）である。
(2)No.1～3の実行時間はCPU 時間であり、No.4、5の実行時間は経過時間である。
(3)測定時のOrg (ﾍﾞｸﾄﾙ)版、Vec3(ﾍﾞｸﾄﾙ)版の経過時間は以下のとおりである。
・Org (ﾍﾞｸﾄﾙ)版：T012: 2:18.87, T040:14:31.21, T100:1:55:34.59, T197:11:47:31.20
・Vec3(ﾍﾞｸﾄﾙ)版：T012: 36.74, T040: 2:21.81、T100: 15:10.64, T197: 1:22:57.57

Table 4.9 Execution time on AP3000.

ﾏｼﾝ	ﾃｽﾄｹｰｽ		T012	T040	T100	T197
AP 3000	逐次版 (Vec3 版)	real	45.49	11:48.74	2:18:35.42	予測値
		user	44.52	11:37.75	2:12:42.61	16^{H} 48^{M}
	並列版(4ﾉｰﾄﾞ)	real	1:23.47	6:48.35	1:17:15.97	9^{H} 49^{M} 予測値
VPP 参考	Org 版 (ﾍﾞｸﾄﾙ･ｺﾝﾊﾟｲﾙ)	real	2:18.57	14:31.21	1:55:34.59	11:47:31.20
		user	1:28.90	14:18.69	1:55:16.23	11:46:25.46
	Vec3版 (ｽｶﾗ･ｺﾝﾊﾟｲﾙ)	real	2:31.48	49:38.34	——	——
		user	2:26.89	49:28.59		
	Vec3版 (ﾍﾞｸﾄﾙ･ｺﾝﾊﾟｲﾙ)	real	36.74	2:21.81	15:10.64	1:22:57.57
		user	24.48	2:16.74	14:35.67	1:22:40.85
	並列版(4PE)	real	42.10	1:32.38	7:04.07	33:07.76

Table 4.10 Comparison of speed up ratio between scalar turning on AP3000 and vector tuning on VPP500.

ﾃｽﾄｹｰｽ	T012	T040
Vec3(ｽｶﾗ) →AP3000逐次実行版	3.3	4.3
Vec3(ｽｶﾗ) →Vec3(ﾍﾞｸﾄﾙ)	6.0	21.7

倍率A→B：（AのCPU 時間）÷（BのCPU 時間）

Table 4.11 Execution time of vectorized version and parallelized version on VPP500.

ﾃｽﾄｹｰｽ		T100	T119	T140	T159	T197
Vec3版 (ﾍﾞｸﾄﾙ･ｺﾝﾊﾟｲﾙ)	real	15:10.64	21:29.04	32:39.73	43:05.33	1:22:57.57
	user	14:35.67	21:08.04	32:25.79	42:46.75	1:22:40.85
並列版(4PE)	real	7:04.07	10:05.15	15:13.07	18:49.75	33:07.76

テストケース TXXX ：質量数XXX のテストケース。質量数、原子番号以外は T100 と同じ

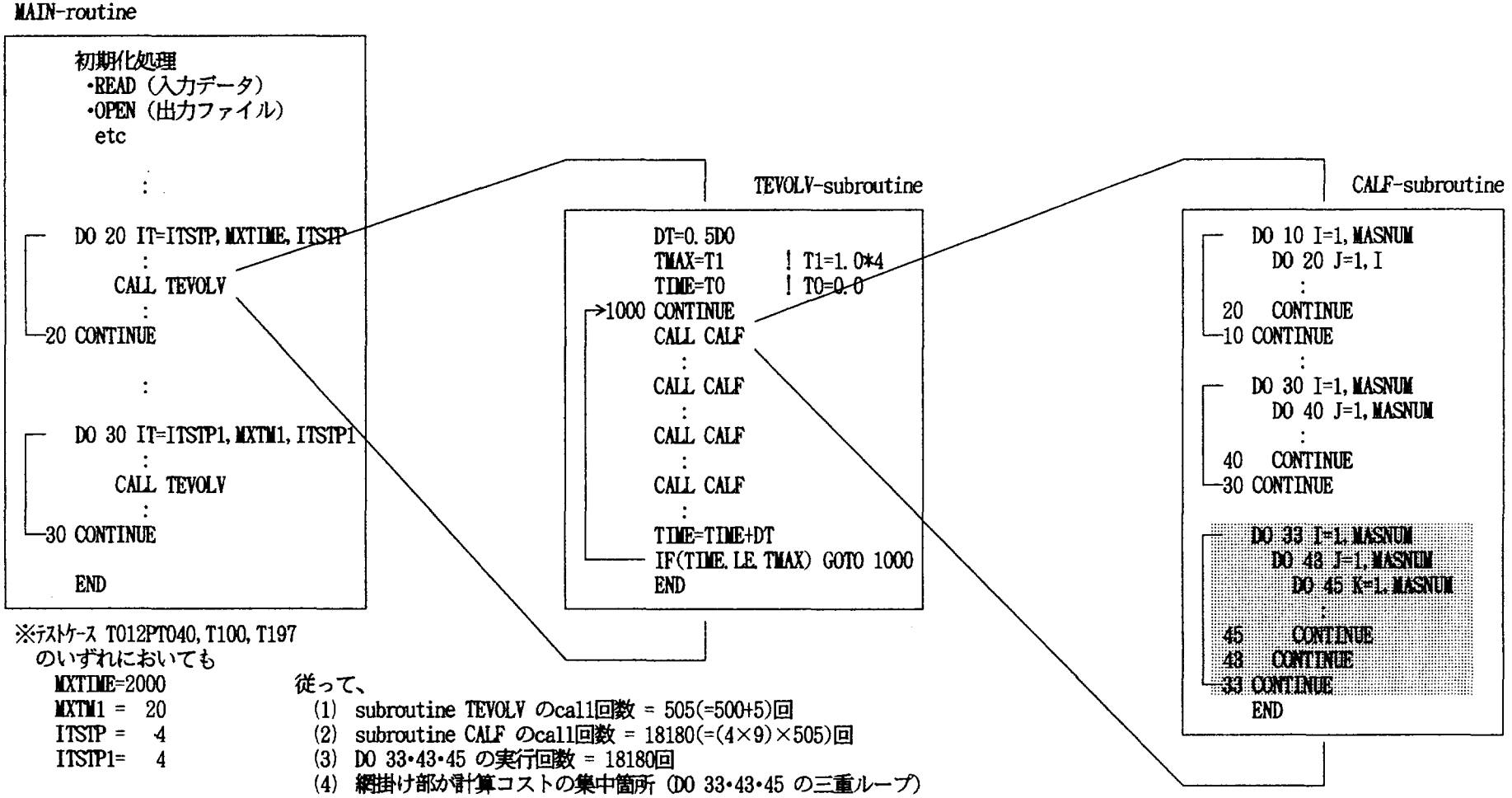


Fig. 4.1 Structure of EQMD code about subroutine CALF.

```
v          DO 33 I=1,MASNUM
s            DO 43 J=1,MASNUM
s2             DO 45 K=1,MASNUM
v2               SMASK=SMASKIJ(I,J)*(2. + SMASKIJ(I,K)*SMASKIJ(J,K))/3
v2               DU3DIJ=-C3* FAC*2./3. *(2-SMASKIJ(I,K))
         &       *(RHORHO(J,K)*RHORHO(I,K))**(2./3.)/RHORHO(I,J)**(1./3.)  ・・・①
v2               DU3DIK=-C3* FAC*2./3.
         &       *(RHORHO(J,K)*RHORHO(I,J))**(2./3.)/RHORHO(I,K)**(1./3.)  ・・・②
v2               DHDR(I,1)=DHDR(I,1)
         &       +((X(I,1)-X(J,1))*DU3DIJ*DRHODR(I,J)
         &       +(X(I,1)-X(K,1))*DU3DIK*DRHODR(I,K))
         &       *SMASK                                                    ・・・③
v2               DHDR(I,2)=DHDR(I,2)
         &       +((X(I,2)-X(J,2))*DU3DIJ*DRHODR(I,J)
         &       +(X(I,2)-X(K,2))*DU3DIK*DRHODR(I,K))
         &       *SMASK                                                    ・・・④
v2               DHDR(I,3)=DHDR(I,3)
         &       +((X(I,3)-X(J,3))*DU3DIJ*DRHODR(I,J)
         &       +(X(I,3)-X(K,3))*DU3DIK*DRHODR(I,K))
         &       *SMASK                                                    ・・・⑤
v2               DHDL(I)=DHDL(I)
         &       -(DU3DIJ*DRHODL(I,J)+DU3DIK*DRHODL(I,K))*SMASK
s2 45          CONTINUE
s  43        CONTINUE
v  33      CONTINUE
```

Fig. 4.2 Original version of DO 33・43・45 loops in subroutine CALF.

```
s2        DO 51 J=1,MASNUM
v2          DO 52 I=1,MASNUM
v2            vX1(I,J)=(X(I,1)-X(J,1))*DRHODR(I,J)
v2            vX2(I,J)=(X(I,2)-X(J,2))*DRHODR(I,J)
v2            vX3(I,J)=(X(I,3)-X(J,3))*DRHODR(I,J)
v2      52    CONTINUE
s2      51 CONTINUE

s2        DO 53 J=1,MASNUM
v2          DO 54 I=1,MASNUM
v2            vRH13(I,J)=RHORHO(I,J)**(-1./3.)
v2            vRH23(I,J)=RHORHO(I,J)**(2./3.)
v2      54    CONTINUE
s2      53 CONTINUE

s         DO 45 K=1,MASNUM
s2          DO 43 J=1,MASNUM
v2            DO 33 I=1,MASNUM
v2              SMASK=SMASKIJ(I,J)*(2. + SMASKIJ(I,K)*SMASKIJ(J,K))/3
v2              DU3DIJ=-C3* FAC*2./3. *(2-SMASKIJ(I,K))
          &     *(vRH23(J,K)*vRH23(I,K)*vRH13(I,J))
v2              DU3DIK=-C3* FAC*2./3.
          &     *(vRH23(J,K)*vRH23(I,J)*vRH13(I,K))
v2              DHDR(I,1)=DHDR(I,1)
          &     +(vX1(I,J)*DU3DIJ+vX1(I,K)*DU3DIK)*SMASK
v2              DHDR(I,2)=DHDR(I,2)
          &     +(vX2(I,J)*DU3DIJ+vX2(I,K)*DU3DIK)*SMASK
v2              DHDR(I,3)=DHDR(I,3)
          &     +(vX3(I,J)*DU3DIJ+vX3(I,K)*DU3DIK)*SMASK
v2              DHDL(I)=DHDL(I)
          &     -(DU3DIJ*DRHODL(I,J)+DU3DIK*DRHODL(I,K))*SMASK
v2      33      CONTINUE
s2      43    CONTINUE
s       45 CONTINUE
```

Fig. 4.3 Modification of DO 33・43・45 loops in subroutine CALF.

```
   *vocl loop,novrec(dhdr,dhdl)
v        DO 30 I=1,MASNUM
v          DO 40 J=1,MASNUM
v            DRHODR(I,J)=RHORHO(I,J)*2/(WLAM(I)+WLAM(J))
             DRHODL(I,J)=    ・・・

v                ・・・

v            DUFDRR(I,J)=VPAULI*DFDRR(I,J)
       & *PP/2*(DIM(F(I)-F0,0.0)**(PP-1)+DIM(F(J)-F0,0.0)**(PP-1))  ・・①
v            DUFDRP(I,J)=VPAULI*DFDRP(I,J)
       & *PP/2*(DIM(F(I)-F0,0.0)**(PP-1)+DIM(F(J)-F0,0.0)**(PP-1))  ・・②
v            DUFDPR(I,J)=VPAULI*DFDPR(I,J)

v                ・・・

v            IF(DIST2(I,J).GE.DFRTH**2) THEN
v              DHFRDR(I,J)=2*EXP(-(SQRT(DIST2(I,J))-DFRTH)**2/R0FR2)
       &       *(SQRT(DIST2(I,J))-DFRTH)/R0FR2/SQRT(DIST2(I,J))
       &       *(1.5D0*(1D0+WLAM(I)*WLAM(I)*DELTA(I)*DELTA(I))
       &       /WLAM(I)/2D0/RMASS/FRTOT(I)**2
       &       +1.5D0*(1D0+WLAM(J)*WLAM(J)*DELTA(J)*DELTA(J))
       &       /WLAM(J)/2D0/RMASS/FRTOT(J)**2  )
v            ELSE
v              DHFRDR(I,J)=0
v            ENDIF

v                ・・・

v            DHDR(I,1)=DHDR(I,1)
       &         +((X(I,1)-X(J,1))
       &             *(DU2DR(I,J)+DUCDR(I,J)+DUSDR(I,J)
       &              +DUFDRR(I,J)-DHFRDR(I,J) )
       &         +(P(I,1)-P(J,1))*DUFDRP(I,J))*SMASKIJ(I,J)      ・・・  ③
v            DHDR(I,2)=DHDR(I,2)
       &         +  ・・・                                       ・・・  ④
v            DHDR(I,3)=DHDR(I,3)
       &         +  ・・・                                       ・・・  ⑤
v            DHDP(I,1)=DHDP(I,1)
       &         +  ・・・                                       ・・・  ⑥
v            DHDP(I,2)=DHDP(I,2)
       &         +  ・・・                                       ・・・  ⑦
v            DHDP(I,3)=DHDP(I,3)
       &         +  ・・・                                       ・・・  ⑧
v            DHDL(I)=DHDL(I)
       &         +  ・・・                                       ・・・  ⑨
v            DHDD(I)=DHDD(I)+DUFDD(I,J)*SMASKIJ(I,J)            ・・・  ⑩
v  40       CONTINUE
v            DRHODL(I,I)=DRHODL(I,I)*2                         ・・・  ⑪
v  30     CONTINUE
```

Fig. 4.4 Original version of DO 30･40 loops in subroutine CALF.

```
v         DO 25 I=1,MASNUM
v           vF(I)=DIM(F(I)-F0,0.0)**(PP-1)
v           WLAM2(I)=WLAM(I)*WLAM(I)
v           FRTOT2(I)=FRTOT(I)*FRTOT(I)
v      25 CONTINUE

s2        DO 26 J=1,MASNUM
v2          DO 27 I=1,MASNUM
v2            sDIST2(I,J)=SQRT(DIST2(I,J))
v2     27     CONTINUE
s2     26 CONTINUE

    *vocl loop,novrec(dhdr,dhdl)
s         DO 40 J=1,MASNUM
v           DO 30 I=1,MASNUM
v             DRHODR(I,J)=RHORHO(I,J)*2/(WLAM(I)+WLAM(J))
              DRHODL(I,J)= ・・・

v                ・・・

v             DUFDRR(I,J)=VPAULI*DFDRR(I,J)*PP/2*(vF(I)+vF(J))
v             DUFDRP(I,J)=VPAULI*DFDRP(I,J)*PP/2*(vF(I)+vF(J))
v             DUFDPR(I,J)=VPAULI*DFDPR(I,J)*PP/2*(vF(I)+vF(J))
v             DUFDPP(I,J)=VPAULI*DFDPP(I,J)*PP/2*(vF(I)+vF(J))
v             DUFDL(I,J)=VPAULI*DFDL(I,J)*PP/2*(vF(I)+vF(J))
v             DUFDD(I,J)=VPAULI*DFDD(I,J)*PP/2*(vF(I)+vF(J))
v             IF(DIST2(I,J).GE.DFRTH**2) THEN
v               DHFRDR(I,J)=2*EXP(-(sDIST2(I,J)-DFRTH)**2/ROFR2)
        &        *(sDIST2(I,J)-DFRTH)/ROFR2/sDIST2(I,J)
        &        *(1.5D0*(1D0+WLAM2(I)*DELTA2(I))/WLAM(I)/2D0/RMASS/FRTOT2(I)
        &        +1.5D0*(1D0+WLAM2(J)*DELTA2(J))/WLAM(J)/2D0/RMASS/FRTOT2(J) )
v             ELSE
v               DHFRDR(I,J)=0
v             ENDIF
v             DHDR(I,1)=DHDR(I,1)
        &         +(X(I,1)-X(J,1))
        &             *(DU2DR(I,J)+DUCDR(I,J)+DUSDR(I,J)
        &               +DUFDRR(I,J)-DHFRDR(I,J) )
        &         +(P(I,1)-P(J,1))*DUFDRP(I,J))*SMASKIJ(I,J)
v             DHDR(I,2)=DHDR(I,2)+
        &         ・・・
v             DHDR(I,3)=DHDR(I,3)+
        &         ・・・
v             DHDP(I,1)=DHDP(I,1)+
        &         ・・・
v             DHDP(I,2)=DHDP(I,2)+
        &         ・・・
v             DHDP(I,3)=DHDP(I,3)+
        &         ・・・
v             DHDL(I)=DHDL(I)+
        &         ・・・
v             DHDD(I)=DHDD(I)+DUFDD(I,J)*SMASKIJ(I,J)
v  30       CONTINUE
s  40     CONTINUE

v         DO 41 I=1,MASNUM
v           DRHODL(I,I)=DRHODL(I,I)*2
v      41 CONTINUE
```

Fig. 4.5 Modification of DO 30·40 loops in subroutine CALF.

```
s          DO 10 I=1,MASNUM
v            DO 20 J=1,I
v              RNORM(I,J)=1D0/(PI*(WLAM(I)+WLAM(J)))**1.5D0
v              RNORM(J,I)=RNORM(I,J)
v              DIST2(I,J)=(X(I,1)-X(J,1))*(X(I,1)-X(J,1))
           &     +(X(I,2)-X(J,2))*(X(I,2)-X(J,2))
           &     +(X(I,3)-X(J,3))*(X(I,3)-X(J,3))                    ・・・①
v              DIST2(J,I)=DIST2(I,J)
v              PDIST2(I,J)=(P(I,1)-P(J,1))*(P(I,1)-P(J,1))
           &     +(P(I,2)-P(J,2))*(P(I,2)-P(J,2))
           &     +(P(I,3)-P(J,3))*(P(I,3)-P(J,3))
v              PDIST2(J,I)=PDIST2(I,J)
v              EXPD1(I,J)=EXP(-DIST2(I,J)/(WLAM(I)+WLAM(J)))       ・・・②
v              EXPD1(J,I)=EXPD1(I,J)
v              RHORHO(I,J)=RNORM(I,J)*EXPD1(I,J)
v              RHORHO(J,I)=RHORHO(I,J)
v              PX(I,J)=(X(I,1)-X(J,1))*(P(I,1)-P(J,1))
           &     +(X(I,2)-X(J,2))*(P(I,2)-P(J,2))
           &     +(X(I,3)-X(J,3))*(P(I,3)-P(J,3))
v              PX(J,I)=PX(I,J)
v              A(I,J)=(WLAM(I)+WLAM(J))**2+
           &     (WLAM(I)*WLAM(J)*(DELTA(I)-DELTA(J)))**2
v              A(J,I)=A(I,J)
v              B(I,J)=(WLAM(I)**2*WLAM(J)*DELTA(I)**2
           &     +WLAM(I)*WLAM(J)**2*DELTA(J)**2
           &     +WLAM(I)+WLAM(J)
           &     )*DIST2(I,J)
           &     +(2*WLAM(I)**2*WLAM(J)*DELTA(I)
           &     +2*WLAM(I)*WLAM(J)**2*DELTA(J)
           &     )*PX(I,J)
           &     +(WLAM(I)**2*WLAM(J)+WLAM(I)*WLAM(J)**2
           &     )*PDIST2(I,J)
v              B(J,I)=B(I,J)
v              FF(I,J)=8*(WLAM(I)*WLAM(J)/A(I,J))**1.5D0
           &     *EXP(-B(I,J)/A(I,J))
           &     *(1-ABS(ICHARG(I)-ICHARG(J)))
           &     *(1-ABS(ISPIN(I)-ISPIN(J))/2)
v              FF(J,I)=FF(I,J)
v              IF(DIST2(I,J).GE.DFRTH**2) THEN
v                FR(I,J)=EXP(-(SQRT(DIST2(I,J))-DFRTH)**2
           &       /ROFR2)
v              ELSE
v                FR(I,J)=1
v              ENDIF
v              FR(J,I)=FR(I,J)
v  20        CONTINUE
s  10      CONTINUE
```

Fig. 4.6 Original version of DO 10･20 loops in subroutine CALF.

```
          MNUM2 = MASNUM*(MASNUM+1)/2

v         DO 4 I=1,MASNUM
v           DELTA2(I)=DELTA(I)*DELTA(I)
v           WxD(I)=WLAM(I)*DELTA(I)
v           WxD2(I)=WxD(I)*DELTA(I)
v       4 CONTINUE

          IJ=0
s         DO 5 I=1,MASNUM
v           DO 6 J=1,I
v             IJ=IJ+1
v             WpW(IJ)     =WLAM(I)+WLAM(J)
v             WxW(IJ)     =WLAM(I)*WLAM(J)
v             WDpWD(IJ)   =WxD(I)+WxD(J)
v             WD2pWD2(IJ)=WxD2(I)+WxD2(J)
v             DmD(IJ)     =DELTA(I)-DELTA(J)
v             ICmIC(IJ)   =1-ABS(ICHARG(I)-ICHARG(J))
v             ISmIS(IJ)   =1-ABS(ISPIN(I)-ISPIN(J))/2
v             XmX1(IJ)    =X(I,1)-X(J,1)
v             XmX2(IJ)    =X(I,2)-X(J,2)
v             XmX3(IJ)    =X(I,3)-X(J,3)
v             PmP1(IJ)    =P(I,1)-P(J,1)
v             PmP2(IJ)    =P(I,2)-P(J,2)
v             PmP3(IJ)    =P(I,3)-P(J,3)
v       6   CONTINUE
s       5 CONTINUE

v         DO 10 IJ=1,MNUM2
v           RNORMij(IJ)=1D0/(PI*WpW(IJ))**1.5D0
v           DIST2ij(IJ)=XmX1(IJ)*XmX1(IJ)
         &  +XmX2(IJ)*XmX2(IJ)
         &  +XmX3(IJ)*XmX3(IJ)
v           PDIST2ij(IJ)=PmP1(IJ)*PmP1(IJ)
         &  +PmP2(IJ)*PmP2(IJ)
         &  +PmP3(IJ)*PmP3(IJ)
v           EXPD1ij(IJ)=EXP(-DIST2ij(IJ)/WpW(IJ))
v           RHORHOij(IJ)=RNORMij(IJ)*EXPD1ij(IJ)
v           PXij(IJ)=XmX1(IJ)*PmP1(IJ)
         &  +XmX2(IJ)*PmP2(IJ)
         &  +XmX3(IJ)*PmP3(IJ)
v           Aij(IJ)=WpW(IJ)**2+
         &  (WxW(IJ)*DmD(IJ))**2
v           Bij(IJ)=(WxW(IJ)*WD2pWD2(IJ)+WpW(IJ))*DIST2ij(IJ)
         &  +2*WxW(IJ)*WDpWD(IJ)*PXij(IJ)
         &  +WxW(IJ)*WpW(IJ)*PDIST2ij(IJ)
v           FFij(IJ)=8*(WxW(IJ)/Aij(IJ))**1.5D0
         &  *EXP(-Bij(IJ)/Aij(IJ))
         &  *ICmIC(IJ)
         &  *ISmIS(IJ)
v           IF(DIST2ij(IJ).GE.DFRTH**2) THEN
v             FRij(IJ)=EXP(-(SQRT(DIST2ij(IJ))-DFRTH)**2/R0FR2)
v           ELSE
v             FRij(IJ)=1
v           ENDIF
v      10 CONTINUE
```

Fig. 4.7 Modification of DO 10･20 loops in subroutine CALF (1/2).

```
          IJ=0
s         DO 15 I=1,MASNUM
v           DO 16 J=1,I
v             IJ=IJ+1
v             RNORM(I,J) =RNORMij(IJ)
v             RNORM(J,I) =RNORM(I,J)
v             DIST2(I,J) =DIST2ij(IJ)
v             DIST2(J,I) =DIST2(I,J)
v             PDIST2(I,J)=PDIST2ij(IJ)
v             PDIST2(J,I)=PDIST2(I,J)
v             EXPD1(I,J) =EXPD1ij(IJ)
v             EXPD1(J,I) =EXPD1(I,J)
v             RHORHO(I,J)=RHORHOij(IJ)
v             RHORHO(J,I)=RHORHO(I,J)
v             PX(I,J)    =PXij(IJ)
v             PX(J,I)    =PX(I,J)
v             A(I,J)     =Aij(IJ)
v             A(J,I)     =A(I,J)
v             B(I,J)     =Bij(IJ)
v             B(J,I)     =B(I,J)
v             FF(I,J)    =FFij(IJ)
v             FF(J,I)    =FF(I,J)
v             FR(I,J)    =FRij(IJ)
v             FR(J,I)    =FR(I,J)
v       16    CONTINUE
s       15 CONTINUE
```

Fig. 4.7 Modification of DO 10･20 loops in subroutine CALF (2/2).

```
v         DO 61 I=1,MASNUM
v           DHDRwk(I,1)=0.0
v           DHDRwk(I,2)=0.0
v           DHDRwk(I,3)=0.0
v           DHDLwk(I)=0.0
v      61 CONTINUE

    !XOCL SPREAD DO
s         DO 45 K=1,MASNUM
s2          DO 43 J=1,MASNUM
v2            DO 33 I=1,MASNUM
v2            SMASK=SMASKIJ(I,J)*(2. + SMASKIJ(I,K)*SMASKIJ(J,K))/3
v2            DU3DIJ=-C3* FAC*2./3. *(2-SMASKIJ(I,K))
         &       *(vRH23(J,K)*vRH23(I,K)*vRH13(I,J))
v2            DU3DIK=-C3* FAC*2./3.
         &       *(vRH23(J,K)*vRH23(I,J)*vRH13(I,K))
v2            DHDRwk(I,1)=DHDRwk(I,1)
         &        +(vX1(I,J)*DU3DIJ+vX1(I,K)*DU3DIK)*SMASK
v2            DHDRwk(I,2)=DHDRwk(I,2)
         &        +(vX2(I,J)*DU3DIJ+vX2(I,K)*DU3DIK)*SMASK
v2            DHDRwk(I,3)=DHDRwk(I,3)
         &        +(vX3(I,J)*DU3DIJ+vX3(I,K)*DU3DIK)*SMASK
v2            DHDLwk(I)=DHDLwk(I)
         &       -(DU3DIJ*DRHODL(I,J)+DU3DIK*DRHODL(I,K))*SMASK
v2      33    CONTINUE
s2      43  CONTINUE
s       45 CONTINUE
    !XOCL END SPREAD SUM(DHDRwk),SUM(DHDLwk)

v         DO 62 I=1,MASNUM
v           DHDR(I,1)=DHDR(I,1)+DHDRwk(I,1)
v           DHDR(I,2)=DHDR(I,2)+DHDRwk(I,2)
v           DHDR(I,3)=DHDR(I,3)+DHDRwk(I,3)
v           DHDL(I)  =DHDL(I)  +DHDLwk(I)
v      62 CONTINUE
```

Fig. 4.8 Parallelized DO 33・43・45 loops in subroutine CALF on VPP500.

```
      PARAMETER(NNPE=4)
      include 'mpif.h'
      COMMON/Para/npe,me
      COMMON/Dvar/ist,ien

            :

      call mpi_init(ierr)
      call mpi_comm_size(mpi_comm_world,numpe,ierr)
      call mpi_comm_rank(mpi_comm_world,mme,ierr)

      npe=numpe
      me=mme

            :

      icnt = MASNUM/npe
      ijud = MASNUM - icnt*npe
      if (ijud .ne. 0) icnt = icnt+1
      ist  = me*icnt + 1
      ien  = ist + icnt - 1
      if (me .eq. (npe-1)) ien=MASNUM

            :

      if(me .eq. 0) then
        OPEN(10,FILE=FNOUT)
      endif

            :

      if(me .eq. 0) then
          WRITE(10,2)IT,RMSRAD(X,WLAM,MASNUM),RMSRO(X,MASNUM)
     &               ,WLAM(0),DELTA(0)
     &               ,OVLPMN,OVLPAV,OVLPMX
     &               ,RMU,WMU,NCOLL,NRFLCT,NREFW,NCLST
     &               ,ENERGY,(MASCL(I),I=1,NCLST)
      endif

            :

      if(me .eq. 0) then
        CLOSE(10)
      endif

      call mpi_finalize(ierr)

      END
```

Fig. 4.9 Parallelized main routine on AP3000.

```
      include 'mpif.h'
      COMMON/Para/npe,me
      COMMON/Dvar/ist,ien
      DIMENSION DHDRwk(NNN,3),DHDPwk(NNN,3)
      DIMENSION DHDLwk(NNN),DHDDwk(NNN)

                :

      DO 31 I=1,MASNUM
        DHDRwk(I,1)=0D0
        DHDRwk(I,2)=0D0
        DHDRwk(I,3)=0D0
        DHDPwk(I,1)=0D0
        DHDPwk(I,2)=0D0
        DHDPwk(I,3)=0D0
        DHDLwk(I)  =0D0
        DHDDwk(I)  =0D0
   31 CONTINUE

      DO 32 I=ist,ien
        DHDPwk(I,1)=P(I,1)/RMASS
        DHDPwk(I,2)=P(I,2)/RMASS
        DHDPwk(I,3)=P(I,3)/RMASS
        DHDLwk(I)=3D0/4D0/RMASS*
     &            (-1D0/WLAM(I)/WLAM(I)+DELTA2(I))
     &          *(1-1/FRTOT(I))
        DHDDwk(I)=3D0*WLAM(I)*DELTA(I)/2D0/RMASS
     &          *(1-1/FRTOT(I))
   32 CONTINUE

                :

      DO 26 I=ist,ien
        DO 27 J=1,MASNUM
          sDIST2(I,J)=SQRT(DIST2(I,J))
   27   CONTINUE
   26 CONTINUE

      DO 30 I=ist,ien
        DO 40 J=1,MASNUM
          DRHODR(I,J)=RHORHO(I,J)*2/(WLAM(I)+WLAM(J))
          DRHODL(I,J)=RHORHO(I,J)*
     &           (-1.5D0/(WLAM(I)+WLAM(J))
     &            +DIST2(I,J)/(WLAM(I)+WLAM(J))**2)

                      :

          DHDDwk(I)=DHDDwk(I)+DUFDD(I,J)*SMASKIJ(I,J)
   40   CONTINUE
   30 CONTINUE

      DO 41 I=ist,ien
        DRHODL(I,I)=DRHODL(I,I)*2
   41 CONTINUE
```

Fig. 4.10 Parallelized subroutine CALF on AP3000(1/2).

```
      DO 51 I=ist,ien
        DO 52 J=1,MASNUM
          vX1(I,J)=(X(I,1)-X(J,1))*DRHODR(I,J)
          vX2(I,J)=(X(I,2)-X(J,2))*DRHODR(I,J)
          vX3(I,J)=(X(I,3)-X(J,3))*DRHODR(I,J)
          vRH13(I,J)=RHORHO(I,J)**(-1./3.)
52      CONTINUE
51    CONTINUE

      DO 53 J=1,MASNUM
        DO 54 I=1,MASNUM
          vRH23(I,J)=RHORHO(I,J)**(2./3.)
54      CONTINUE
53    CONTINUE

      DO 33 I=ist,ien
        DO 43 J=1,MASNUM
          DO 45 K=1,MASNUM
          SMASK=SMASKIJ(I,J)*(2. + SMASKIJ(I,K)*SMASKIJ(J,K))/3
          DU3DIJ=-C3* FAC*2./3. *(2-SMASKIJ(I,K))
     &       *(vRH23(J,K)*vRH23(I,K)*vRH13(I,J))
          DU3DIK=-C3* FAC*2./3.
     &       *(vRH23(J,K)*vRH23(I,J)*vRH13(I,K))
          DHDRwk(I,1)=DHDRwk(I,1)
     &        +(vX1(I,J)*DU3DIJ+vX1(I,K)*DU3DIK)*SMASK
          DHDRwk(I,2)=DHDRwk(I,2)
     &        +(vX2(I,J)*DU3DIJ+vX2(I,K)*DU3DIK)*SMASK
          DHDRwk(I,3)=DHDRwk(I,3)
     &        +(vX3(I,J)*DU3DIJ+vX3(I,K)*DU3DIK)*SMASK
          DHDLwk(I)=DHDLwk(I)
     &       -(DU3DIJ*DRHODL(I,J)+DU3DIK*DRHODL(I,K))*SMASK
45        CONTINUE
43      CONTINUE
33    CONTINUE
      call mpi__barrier(mpi__comm__world,ierr)
      call mpi__allreduce(DHDRwk,DHDR,NNN*3
     &                    mpi__real,mpi__sum,mpi__comm__world,ierr)
      call mpi__allreduce(DHDPwk,DHDP,NNN*3,
     &                    mpi__real,mpi__sum,mpi__comm__world,ierr)
      call mpi__allreduce(DHDLwk,DHDL,MASNUM,mpi__real,mpi__sum,
     &                                       mpi__comm__world,ierr)
      call mpi__allreduce(DHDDwk,DHDD,MASNUM,mpi__real,mpi__sum,
     &                                       mpi__comm__world,ierr)
      call mpi__barrier(mpi__comm__world,ierr)
```

Fig. 4.10 Parallelized subroutine CALF on AP3000(2/2).

参考文献

[1] Toshiki Maruyama,Koji Niita,and Akira Iwamoto, Phys. Rev. C 53, 297(1996)

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

[3] 「UXP/M アナライザ使用手引書（FORTRAN,VP 用） V10L20 用」，富士通（株），1992 年 2 月.

[4] 「MPI:A Message-Passing Interface Standard」，Message Passing Interface Forum, May 10, 1996.

[5] 「MPI/AP V1.0 使用手引書」，富士通(株)

5. STREAM V3.10 コードの並列化

5.1 概要

高温ガス炉や気体冷却方式の核融合炉設計では，密度の小さい気体を40気圧～ 100気圧程度に加圧して密度を大きくし，熱容量を大きくする工夫がなされている．この場合，炉内のガス流については近似的に非圧縮性流体として扱うことができる．しかし，配管や機器の破断事故時には炉内外で大きな圧力差が生じ，気体流れは衝撃波や超音速流を伴う流れへと変貌する．また，高温ガス炉や核融合炉等の水素利用システムにおいては，事故時の水素燃焼等の現象把握が機器システムの安全性評価の上で極めて重要となる．

以上のように，圧縮性流体の熱流動現象は原子炉及び核融合炉の機器設計や安全性評価を行う際に頻繁に現れる現象である．これまでに原研が整備を進めてきた多次元熱流体数値解析コードSTREAM は，圧縮性流体解析用バージョンV3.1 が最新であり，富士通製スーパーコンピュータ VPP500 上での高速な計算が可能となっている [1,2].

以下では，三次元熱流体解析コード STREAM V3.10 の並列化作業について述べる．この並列化は，分散メモリ型並列計算機 AP3000 及び VPP500 上での並列実行を可能とし，大規模計算への対応及び計算速度の向上を目指したものである．また，MPI [3] ライブラリを利用して並列化を行うことにより，その他様々な並列計算機環境での実行を可能とすることも目標とした．

本コードは，3次元の流体方程式（Navier-Stokes 方程式）を有限差分法を用いて離散化し，圧力のポアソン式をSIMPLE 法で解くものである．並列化には，メッシュで区切られた解析空間に対して，K 軸方向を並列化の軸としてプロセッサ台数個に分割し，それぞれに対応するデータを個々のプロセッサに割り当てる領域分割の手法を適用した．また，本コードに含まれている前処理付きの反復解法による行列ソルバ部分は，その不完全 LU 分解計算部分及び前進・後退代入計算部分にデータの依存関係があるために単純に並列計算させることはできないことから，パイプライン的に並列性を引き出す方法を適用して並列化を行った．

5.2 前作業

5.2.1 main プログラムの変更

本コードの main プログラムは C 言語で記述されており，以下，FORTRAN で記述された手続きを呼び出して計算を行う構造となっている．このような場合，AP3000 上の FORTRAN コンパイラ（`f90`）[4] 及び VPP500 用の FORTRAN コンパイラ（`frtpx`）[5] の仕様では，`main` 関数を `MAIN__` という名前の関数に変更して，FORTRAN コンパイラを使用してリンケージを行わなくてはならない．そこで，C 言語で記述されたソースファイル `streamMC.c` を編集して，`main` 関数の名前を `MAIN__` に変更した（Fig. 5.1）．

5.3 倍精度化

本コードで使用される実数値は全て単精度である．ところが，本コードで取り扱われる物性値の中には圧力などのように十分な精度を持つことが望ましいものが含まれており，精度を拡張する作業を行うことにした．ただし，特定の物性値のみを精度拡張することが困難であるため，本コードで使用される全ての実数値を一様に倍精度化することにした．

作業は次の手順で行った．まず，全ソースプログラム（FORTRAN で記述されたもののみ）を GS8400 システムに転送し，GS8400 上のツール STREAM77 [6] を用いて次に示す 3 種の変換を行った．

- `IMPLICIT REAL*8 (A-H,O-Z)` を挿入する．
- 型宣言文 `REAL ...` を `REAL*8 ...` に変更する．
- 単精度浮動小数点定数を倍精度に変換する．

この時用いた JCL を Fig. 5.2 に示す．

次に，C 言語で記述されたソースプログラム中の `float` 型を全て `double` 型に変更した．

5.4 並列化

本並列化作業は，次に示す方針を基に行った．

- 本コード [1,2] は，I 軸，J 軸，K 軸から構成される 3 次元の解析空間を持つ．本作業では，これら 3 軸のうちの一つ K 軸を選んで分割軸とし，領域分割の考え方を適用して並列化することにした．すなわち，場の量を格納する配列それぞれの K 軸方向を，計算に使用するプロセッサ台数個に分割してそれぞれを個々のプロセッサに保持させておく．基本的に，各プロセッサは自プロセッサ上のデータのみを参照して，自プロセッサ上のデータを定義する処理を行う．
- データ分割の際には，分割面の外側に冗長な領域を添付しておくことにした．この冗長な領域は「袖」と呼ばれるものである．本並列化では，作業を容易にするため，必要であるか必要でないかを問わず，分割された配列のほとんど全てに袖を付加しておく．この袖部分は，両隣のプロセッサから転送された分割面近傍のデータを格納しておく領域であり，両隣のプロセッサが担当する領域のデータを間接的に参照するために使用する．なお，袖部分の幅は，本コードの処理内容を検討した結果，左右それぞれ 2 とすることにした．
- MPI を利用して並列化を行う．
- 本作業で利用した MPI ライブラリには，並列処理用のファイル入出力に関する機能が含まれておらず，FORTRAN 言語固有の機能を用いてファイル入出力を行わなければならない．したがって，特に分割された配列に値を読み込む場合や，分割された配列の内容を書き出す場合には，特別な工夫が必要となる．本作業では，ファイル入出力は原則として特定の 1 台のプロセッサに処理させることにした．入力の場合にはこのプロセッサが読み

込んだデータを全プロセッサに転送する処理を行い，出力の場合には全プロセッサからこのプロセッサにデータを全て集めてから出力を行う．

- 本コードのソースプログラムの量が非常に多いため，本作業では数個のテスト用入力データを用いてコードを実行し，実際に使用された箇所を並列化の対象とすることにした．なお，入力データには，非圧縮性流体を取り扱うものと圧縮性流体を取り扱うものの2種類があるが，本作業では圧縮性流体を取り扱うもののみを対象とした．
- 本コードには，連立一次方程式の解法ルーチンが数種類用意されている．これらの行列解法ルーチンのうち，JACOBI 法を使用したもの及び SOR 法を使用したものについては，使用されることが稀であることから，並列化を行わないことにした．

以下，本作業の内容を示す．

5.4.1 並列処理のための初期化

本コードの処理が開始されると，C 言語で記述された MAIN__ 関数から FORTRAN 言語で記述されたサブルーチンが呼び出され，入力データの一部を読み込む処理が行われる．本並列化では，処理が開始されて最初に呼び出される FORTRAN 言語の手続き中に，並列処理のための初期化を行う処理を挿入した．この処理を Fig. 5.3 に示す．また，図の内容について以下に示す．

- インクルードファイル mpif.h は，MPI ライブラリで使用されるパラメータの宣言文などが記述されたファイルである．手続き中でこれらのパラメータ（この図では MPI_COMM_WORLD，MPI_PROC_NULL など）が参照される場合には，mpif.h をインクルードしておく必要がある．
- インクルードファイル PARA は，本並列化で作成したファイルである．このファイルの内容について Fig. 5.4 に示す．
- MPI_INIT は MPI ライブラリのサブルーチンであり，MPI ライブラリのための初期化処理を行う．
- コモン変数 ICOMM には，計算で使用するコミュニケータを格納しておく．コミュニケータとは，複数のプロセッサから構成されるグループの各種情報を定義したものである．本並列化では特別なコミュニケータを必要としないため，MPI 標準のコミュニケータ MPI_COMM_WORLD を代入している．
- MPI_COMM_SIZE は MPI ライブラリのサブルーチンである．計算に使用されるプロセッサ数が，引数に指定したコモン変数 NPROCS に格納される．
- MPI_COMM_RANK は MPI ライブラリのサブルーチンである．引数に指定したコモン変数 MYRANK には，自プロセッサのランクが格納される．ランクとは，各プロセッサに割り振られた通し番号で，0 ～（プロセッサ台数 -1）の値をとる．

- MPI_ABORT は，MPI ライブラリのサブルーチンであり，全プロセッサの処理を中断させる．このサブルーチンの引数として ICOMM の代わりに MPI_COMM_WORLD を指定しているが，これは ICOMM に特別なコミュニケータが格納されている場合を考慮したためである．本並列化ではコードの各所に MPI_ABORT の呼び出しを挿入したが，これら全ての箇所で MPI_COMM_WORLD を指定した．

- コモン変数 ILPROC には，左隣のプロセッサのランクを格納する．左隣が存在しないプロセッサでは，ILPROC に MPI_PROC_NULL を格納しておく．MPI_PROC_NULL は MPI ライブラリのパラメータである．プロセッサ間のデータ転送において，転送先プロセッサのランクとしてこのパラメータを指定しておくと，データ転送が行われないという性質を持つ．

- コモン変数 IRPROC には，右隣のプロセッサのランクを格納する．右隣が存在しないプロセッサでは，IRPROC に MPI_PROC_NULL を格納しておく．

- コモン変数 IROOT には，ファイル入出力を行うプロセッサのランクを格納しておく．ファイル入出力については「5.4.7 ファイル入出力」に示す．

5.4.2 各プロセッサが担当する領域の決定

入力データの一部が読み込まれて MPI 関連の初期化が行われた後，さらに入力データの残りの一部分が読み込まれ，解析空間のメッシュ数 IM，JM，KM が読み込まれる．この処理の直後に，各プロセッサの担当する領域を決定する処理を挿入した．Fig. 5.5 に，挿入した処理を，以下にこの図において定義されているコモン変数，コモン配列についてを示す．また，これらの変数，配列に格納された値の関係を Fig. 5.6 に示す．

- パラメータ MWDOLL と MWDOLR には，それぞれ左袖，右袖の大きさが格納されている．

- コモン配列 KCHM には，各プロセッサに割り当てられる領域の K 軸方向の大きさが格納される．この配列は，ファイル入出力などにおいて，分割された配列の内容を 1 台のプロセッサに集めたりする場合に参照される．

- コモン配列 KCHMP には，KCHM と同様に各プロセッサに割り当てられる領域の K 軸方向の大きさが格納される．KCHM との違いは，解析空間の右端を担当するプロセッサの領域の大きさが KCHM の場合より 1 だけ大きくなっている（KCHM(NPROCS-1)+1.EQ.KCHMP(NPROCS-1)）点のみである．

- コモン配列 KCHMPP にも同様なデータが格納される．KCHM との違いは，解析空間の左端を担当するプロセッサと右端を担当するプロセッサの領域の大きさが 1 だけ大きくなっている（KCHM(0)+1.EQ.KCHMPP(0)，KCHM(NPROCS-1)+1.EQ.KCHMPP(NPROCS-1)）点のみである．

- コモン変数 KST1，KST2，KST1PP，KEDM，KEDMP，KEDMPP には，プロセッサごとに異なる値が定義される．これらは，主に処理の分割を行う際に，K 軸方向の DO 変数の初期値，終端値として使用される．

- コモン変数 KLBM, KUBM, KLBMP, KUBMP, KLBMPP, KUBMPP には，プロセッサごとに異なる値が定義される．これらは，分割された配列を宣言する際に，K 軸方向の寸法の下限，上限として使用される．

5.4.3 データの分割

本コードは，必要な量のメモリを動的に確保し，確保した領域を細分化して引数渡しで下位のサブルーチンに受け渡す形で，個々の配列の領域を確保している．そこで，メモリを確保する際に，分割の対象となる配列に必要な要素数の計算を行う部分を変更するとともに，全てのルーチンについて分割された配列の配列宣言文を次のように変更することでデータの分割を行った．

- K 軸方向の要素数が KM である配列は，K 軸方向の寸法の下限を KLBM，上限を KUBM として宣言する．
- K 軸方向の要素数が KMP である配列は，K 軸方向の寸法の下限を KLBMP，上限を KUBMP として宣言する．
- K 軸方向の要素数が KMPP である配列は，K 軸方向の寸法の下限を KLBMPP，上限を KUBMPP として宣言する．

5.4.4 処理の分割

本並列化における処理分割は，分割された配列が定義または引用される部分に対して行った．本作業における処理分割の基本的な方針を次に示す．

- 分割された配列に値が定義される部分では，自プロセッサに割り当てられた領域内（袖部分を除く）のデータを定義する処理を個々のプロセッサに行わせる．
- 分割された配列の要素が引用される部分では，自プロセッサに割り当てられた領域内（袖部分を除く）のデータを引用する処理を個々のプロセッサに行わせる．
- 分割された配列に値が定義されると同時に分割された配列の要素が参照される場合は，定義される側を優先し，自プロセッサに割り当てられた領域内（袖部分を除く）のデータの定義を個々のプロセッサに行わせる．この時，参照される配列の要素が他プロセッサに割り当てられている場合は，参照される側の配列の袖転送を事前に行っておき，他プロセッサ上のデータを参照することなく処理できるようにしておく．

以下に，様々な場合に対する処理分割の例を示す．

5.4.4.1 標準的な処理

最も単純なループの例では，K 軸方向のループの初期値と終端値を，「5.4.2 各プロセッサが担当する領域の決定」に示したコモン変数の適当なものに置き換えることによって処理分割を行った．例を Fig. 5.7 に示す．

また，ループの初期値に K1，終端値に K2 などといった変数が指定されており，「5.4.2 各プロセッサが担当する領域の決定」に示したコモン変数で直接置き換えることができない場合に

は，初期値を MAX(K1,KST1)，終端値を MIN(K2,KEDMP) のように MIN，MAX 関数を使用した形に変更した（Fig. 5.8）．このようにすることで，K1，K2 で示された範囲のうち自プロセッサが担当する領域のみの計算を各プロセッサが行うようにした．

5.4.4.2 総和計算

複数台のプロセッサ上に存在するデータを総和する場合には，まず個々のプロセッサで通常の総和計算を行った後，プロセッサごとに得られた結果を MPI ライブラリのサブルーチン MPI_ALLREDUCE を用いて総和する方法を用いた．Fig. 5.9 に例を示す．

5.4.4.3 最大値，最小値計算

複数台のプロセッサ上に存在するデータから最大値または最小値を見つける計算部分については，総和計算の場合と同様に，まず個々のプロセッサで通常の最大値，最小値計算を行った後，MPI_ALLREDUCE を利用する方法を用いた．

また，最大値または最小値を求めると同時に，これらの値が格納された配列のインデックスなどといった他の情報も求めている場合には，次の 2 通りの方法を用いた．

- MPI ライブラリでは，パラメータ MPI_MAXLOC 及び MPI_MINLOC を使用すると，最大値または最小値を求めると同時に，これらの値に付随した値も一つだけ得ることができる．したがって，配列のインデックス I，J，K の値も必要となる場合には，一旦これらの値を一つの値に変換してから MPI_ALLREDUCE を行い，得られた値から I，J，K を復元する方法を用いた．例を Fig. 5.10 に示す．
- 他の情報を一つの値にまとめることができない場合には，最大値または最小値に付随する値としてこれらの値を保持しているプロセッサのランクを指定しておき，必要なデータをこのプロセッサからブロードキャストする方法を用いた．Fig. 5.11 に例を示す．

5.4.4.4 その他

本コードには，Fig. 5.12 に示す例のように一つのループ中で K 番目の要素と K+1 番目の要素を同時に定義している部分がある．このようなループを並列化する場合，ループ全体を適当に分割して K 番目の要素を計算するループと K+1 番目の要素を計算するループに分けてしまうなど，様々な方法が考えられるが，本作業では次のような方法を用いることにした．

Fig. 5.12 の並列化後を Fig. 5.13 に示す．並列化後は，並列化前のループを 2 回に分けて処理を行い，1 回目で K 番目の要素を，2 回目で K+1 番目の要素の定義を行う．1 回目では，KK.EQ.K が成立し，各プロセッサは自分が保持する K 番目の要素に値を代入する処理を行う．2 回目では，KK.EQ.K+1 が成立し，各プロセッサは自分が保持する K+1 番目の要素に値を代入する処理を行うことになる．

5.4.5 行列ソルバの並列化

他の処理部分と同様に，行列ソルバ部分も K 軸方向を分割軸としてデータの分割を行うことにした．しかし，本コードで使用される行列ソルバはいずれも前処理付きの解法であり，不完全

LU 分解計算部分と前進，後退代入計算部分では依存のある計算が行われる．そこで，これら依存のある計算を行う部分に対しては，パイプライン的に並列性を引き出す方法を用いて処理の分割を行うことにした．Fig. 5.14に，この方法の処理イメージを示す．この方法では，J 軸方向を複数の小ブロックに分けて処理を行う．まず，最初のステップではプロセッサ 0 だけが処理を行い，図中の 1 のブロックの計算を行う．次に，1 のブロックの右端のデータをプロセッサ 1 に転送する．この結果，次のステップでは，プロセッサ 0 が 2 のブロックを，プロセッサ 1 が 1 のブロックを処理することが可能となる．このように，各プロセッサが少しずつ担当分の計算を行い，その結果を隣接するプロセッサに転送する処理を繰り返すことで並列性を引き出す．

なお，1 ステップで処理する J 軸方向の幅の大きさは，並列処理効率に大きく影響を与えると考えられるため，最適な値を設定しておく必要がある．しかし，この値は計算に使用するプロセッサ台数や解析空間の形状に依存すると考えられ，最適な値を計算することが困難であるため，現状ではこの値を 2 に固定しておくことにした．

5.4.6 袖転送

袖転送は，対象となる配列に最後に値が定義されてから，その配列の要素が初めて参照されるまでの間に一度だけ実行されるようにしておけばよい．しかし，本コードの規模が非常に大きいこと，方針として本コードの一部のみを並列化作業の対象とすること，及び入力データに応じてプログラムの挙動が変化するなどの理由から，各配列の袖転送を行うべき箇所を特定しておくことが非常に難しい．そこで，次のような方法を用いることにした．

まず，袖を持つ配列のそれぞれについて，袖の状態を示す情報を付加しておく．この情報は，対応する配列の袖転送が完了していることを示す値と，まだ袖転送が実行されていないことを示す値の 2 種類の値をとる．次に，袖を持つ配列に値が代入される全ての箇所に，それぞれの配列に対応する情報を，袖転送が行われていないことを示す値に変更する処理を挿入する．さらに，個々の配列の袖部分が参照される全ての箇所の直前に，対象の配列の袖転送が完了しているかを調べ，行われていなければ袖転送を実行し，かつその配列に関する情報を袖転送が完了していることを示す値に変更する処理を挿入する．このような方法を用いることにより，どのような状況下でも常に必要最低限の袖転送で済ませることができるようになる．

以下では，上に示した機能を実現するために作成したサブルーチン群の説明と，本並列化でこれらのサブルーチンをどのように使用したかについて示す．

5.4.6.1 袖転送ルーチンについて

インクルードファイル OVLP　このインクルードファイルには，配列の袖部分に関する情報を格納するコモン配列などの宣言文が記述されている．このファイルの内容については Fig. 5.15 に示す．

サブルーチン OLINIT　このサブルーチンの内容を Fig. 5.16 に示す．このサブルーチンは，初期化処理として変数 NOOVLP に 0 を代入する．

サブルーチン OLREG　このサブルーチンの内容を Fig. 5.17 に示す．このサブルーチンは，袖転送の対象となる配列の登録を行う．登録される情報は，配列の先頭アドレス，配列の一次元

目の要素数と二次元目の要素数の積，配列の三次元目の寸法の下限，上限，袖部分を無視した場合の寸法の下限，上限，配列の型，及び配列名である．なお，引数に指定された配列の先頭アドレスを獲得するため，MPI ライブラリのサブルーチン MPI_ADDRESS を使用している．

サブルーチン OLSORT　このサブルーチンの内容を Fig. 5.18 に示す．このサブルーチンは，サブルーチン OLREG による全ての配列の登録が完了した後で呼び出し，登録された配列の情報を各配列のアドレスの昇順に並び替える処理を行う．これは，以下の手続きにおいて，指定されたアドレスに対応する情報を取り出す際に 2 値検索を用いるためである．

サブルーチン OLRST　このサブルーチンの内容を Fig. 5.19 に示す．このサブルーチンは，指定された配列の情報を，袖転送が行われていないことを示す値に変更する．

サブルーチン OLFIXI　このサブルーチンの内容を Fig. 5.20 に示す．このサブルーチンは，引数に指定された整数型配列の袖転送を必要に応じて実行する．このサブルーチンを呼び出す時に指定する引数のうち，2 番目から 6 番目までに指定する情報は，サブルーチン OLREG で登録したものと全く同じものである．これらの情報は，デバッグ（OLREG で登録した情報と OLFIXI の引数で指定した情報が一致するかどうか）の目的で使用している．7 番目の引数に 0 以外の整数値を指定すると左袖が処理の対象となり，8 番目の引数引数に 0 以外の整数値を指定すると右袖が処理の対象となる．9 番目の引数に 0 以外の値を指定しておくと，必要であるかどうかに関わらず強制的に袖転送が実行される．

サブルーチン OLFIXR　このサブルーチンの内容を Fig. 5.21 に示す．このサブルーチンについては，実数型配列を処理の対象としている点を除いて OLFIXI と同様である．

5.4.6.2　袖転送ルーチンの使用例

本作業で作成した袖転送ルーチンの使用例を Fig. 5.22 に示す．図中の配列 A，B は，分割された配列であり，かつ袖を付加されているものとする．また，サブルーチン OLINIT，OLREG，OLSORT がすでに実行されており，A，B 共に登録が完了しているものとする．

- DO 100 ループでは A，B に値が代入されているため，ループの直後でサブルーチン OLRST を呼び出して，袖転送が実行されていないことを定義している．
- DO 200 ループでは A の左袖部分と B の右袖部分が参照されている．そこで，ループの直前でサブルーチン OLFIXR を呼び出して，これらの配列の袖転送が実行されるようにしている．ただし，サブルーチン SUB1 中で A または B の袖転送が実行される場合は，DO 200 の直前に挿入した OLFIXR では，実際の袖転送は行われない．
- DO 300 ループでは，ループ中から呼び出されたサブルーチン SUB2 中で A と B の袖部分が参照される．このような場合，呼び出す側の手続きのループの直前で袖転送を実行するようにして，OLFIXR が繰り返し呼び出されるのを避けるようにしている．

5.4.7 ファイル入出力

並列化に伴うファイル入出力部分の変更は，次の方針に従って行った．

コモン変数 IROOT に特定の 1 台のプロセッサのランクを格納しておき，原則としてこのプロセッサのみにファイル入出力を行わせる．入出力対象のデータが分割されているかどうかに応じて，次に示す処理を行う．

分割されていないデータの入力　IROOT で示されたランクを持つ 1 台のプロセッサが読み込みを行った後，このプロセッサから他のプロセッサに対してブロードキャストを行う．例を Fig. 5.23 に示す．

分割されていないデータの出力　分割されていないデータについては基本的に全てのプロセッサが同じ値を保持するため，いずれか 1 台のプロセッサが WRITE 文を実行して出力を行えばよい．本コードでは，IF 文によって IROOT で示されたランクを持つ 1 台のプロセッサだけに WRITE 文を実行させ，自身の保持するデータの内容を出力させるようにした．

分割されたデータの入力　分割された配列へのデータ入力は次のような方法で行うことにした．まず，IROOT で示されたランクを持つ 1 台のプロセッサが全プロセッサ分のデータを読み込む．次に，MPI のサブルーチン MPI_SCATTERV を実行して，個々のプロセッサが担当する領域のデータを各プロセッサに転送する．

この MPI_SCATTERV によって比較的多量のデータが転送されることになるが，直前までに転送されたデータが転送先の（MPI ライブラリが内部的に使用する）受信バッファに残っていると，受信バッファが溢れて異常終了してしまうことがある．そこで，MPI ライブラリのサブルーチン MPI_BARRIER を MPI_SCATTERV の直前に挿入し，全プロセッサ間の同期によって受信バッファの内容を一旦空にすることで，受信バッファの溢れを防ぐことにした．分割した配列にデータを読み込む部分の並列化例を Fig. 5.24 に示す．

分割されたデータの出力　分割された配列からのデータの出力は，分割された配列への入力と逆の手順で行う．まず，MPI_GATHERV を実行して，出力対象の配列のデータを，IROOT で示されたランクを持つ 1 台のプロセッサに集める．この時も比較的多量のデータが転送されることになり，受信バッファが溢れてしまう可能性があるため，MPI_BARRIER を実行してから MPI_GATHERV を実行する．集められたデータの格納には，分割されたデータの入力を行う場合と同様に WORK または IWORK を使用する．次に，IROOT で示されたランクを持つプロセッサが，WORK または IWORK の内容の出力を行う．例を Fig. 5.25 に示す．

5.5 効果

並列化による効果を計測するため，並列化前後のソースプログラム中に実時間を計測する処理を挿入して AP3000 及び VPP500 上で実行し，プログラムの各部で費やされる計算時間を調査した．Table 5.1 に示した 3 種類の入力データを用いた結果を Table 5.2 ～ Table 5.4 に示す．表中の B から L までの範囲が時間発展ループに含まれている．L には，グラフィック用データやリスタートジョブ用データの出力が含まれており，比較的大きな時間が費やされる傾向があ

る．また，C から J については，それぞれの範囲内で呼び出される行列ソルバ部分で費やされた時間を括弧内に示した．

5.6 まとめ

本作業では，MPI ライブラリを用いて三次元熱流体解析コード STREAM V3.10 を分散メモリ型並列計算機向きに並列化する作業を行った．並列化の手法として領域分割を適用し，個々のプロセッサに割り当てられるデータ量をプロセッサ台数分の 1 程度に減少させることで大規模な計算への対応を可能とし，かつ領域分割にともなう処理の分散によって計算速度の向上を図った．また，データ間の依存が強く並列化の困難な行列ソルバ部分は，パイプライン的に並列性を引き出す方法を用いて並列化を行った．

しかし，本コードは非常に規模が大きく，今回は圧縮性流体を取り扱う部分のみを並列化の対象とし，かつ 3 種類のテスト用入力データを実行するのに必要最低限の部分のみを作業の対象とした．したがって，本コードを実際の計算で使用するためには，さらに並列化作業を継続して行う必要があると思われる．

本並列化作業の中で最も困難だったのが，特に流束に関連した計算を行う部分の並列化である．これらの計算部分では，Fig. 5.12 のように一つのループ中で隣合う要素の定義・引用を行っていることが多く，特別な方法を用いなければ並列計算させることができない．本作業では並列化の対象とならなかったが，その他の部分にはさらに複雑で巨大なループも含まれており，今後の並列化作業での困難が予想される．また，このようなループは，回帰計算の可能性があるためにベクトル計算させることもできないというデメリットがある．今後，これらのコーディング部分が，ベクトル計算あるいは並列計算向きに改善されることを期待したい．

Table 5.1 Input data.

ファイル名	メッシュ数	時間ステップ数	備考
B07.S	39x1x67	500	J 軸方向の要素数が 1 であるため，行列ソルバ部分での並列処理による速度向上が得られない.
B09.S	10x5x15	100	
B10.S	63x31x94	4	このデータは，B09.S を元に作業者が作成した.

Table 5.2 Data B07.S (1/2).

(a) AP3000，倍精度化版

計測範囲	計測時間（秒）
A	0.065
B	0.166
C	0.959 (0.000)
D	22.616 (4.969)
E	21.717 (4.874)
F	0.042 (0.000)
G	20.709 (4.920)
H	12.775 (6.651)
I	0.049 (0.000)
J	0.039 (0.000)
K	0.039
L	9.403

(b) VPP500，倍精度化版

計測範囲	計測時間（秒）
A	1.000
B	4.696
C	0.289 (0.000)
D	21.752 (5.911)
E	37.154 (5.948)
F	0.001 (0.000)
G	35.674 (5.907)
H	17.288 (7.826)
I	0.001 (0.000)
J	0.000 (0.000)
K	0.001
L	20.404

(c) AP3000，並列化版，1PE

計測範囲	計測時間（秒）
A	0.093
B	0.496
C	2.062 (0.000)
D	41.265 (12.401)
E	45.951 (14.620)
F	0.041 (0.000)
G	42.654 (13.625)
H	23.206 (14.857)
I	0.046 (0.000)
J	0.039 (0.000)
K	0.038
L	24.396

(d) VPP500，並列化版，1PE

計測範囲	計測時間（秒）
A	2.747
B	5.242
C	0.327 (0.000)
D	20.890 (5.216)
E	44.471 (5.268)
F	0.001 (0.000)
G	51.323 (5.239)
H	15.991 (6.239)
I	0.001 (0.000)
J	0.000 (0.000)
K	0.000
L	139.558

Table 5.2 Data B07.S (2/2).

(e) AP3000, 並列化版, 2PE

計測範囲	計測時間 (秒)
A	0.888
B	0.285
C	0.512 (0.000)
D	16.848 (6.578)
E	17.482 (7.284)
F	0.042 (0.000)
G	17.832 (8.348)
H	11.021 (7.322)
I	0.045 (0.000)
J	0.040 (0.000)
K	0.040
L	11.715

(f) VPP500, 並列化版, 2PE

計測範囲	計測時間 (秒)
A	2.620
B	4.766
C	0.221 (0.000)
D	14.374 (5.155)
E	26.702 (7.444)
F	0.001 (0.000)
G	30.109 (7.537)
H	11.599 (6.550)
I	0.001 (0.000)
J	0.000 (0.000)
K	0.000
L	34.540

(g) AP3000, 並列化版, 4PE

計測範囲	計測時間 (秒)
A	0.910
B	0.230
C	0.325 (0.000)
D	12.870 (6.199)
E	12.550 (5.555)
F	0.042 (0.000)
G	13.620 (5.927)
H	10.659 (7.625)
I	0.043 (0.000)
J	0.040 (0.000)
K	0.039
L	12.071

(h) VPP500, 並列化版, 4PE

計測範囲	計測時間 (秒)
A	0.834
B	4.577
C	0.146 (0.000)
D	10.797 (5.490)
E	17.778 (5.853)
F	0.001 (0.000)
G	19.368 (6.432)
H	10.620 (7.087)
I	0.001 (0.000)
J	0.000 (0.000)
K	0.000
L	14.431

Table 5.3 Data B09.S (1/2).

(a) AP3000, 倍精度化版

計測範囲	計測時間 (秒)
A	0.025
B	0.031
C	0.079 (0.000)
D	6.268 (0.298)
E	1.407 (0.259)
F	0.008 (0.000)
G	1.381 (0.268)
H	1.297 (0.869)
I	0.008 (0.000)
J	0.008 (0.000)
K	0.008
L	0.625

(b) VPP500, 倍精度化版

計測範囲	計測時間 (秒)
A	0.734
B	2.868
C	0.045 (0.000)
D	8.366 (0.533)
E	2.446 (0.506)
F	0.000 (0.000)
G	3.035 (1.136)
H	1.630 (0.868)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	2.288

(c) AP3000, 並列化版, 1PE

計測範囲	計測時間 (秒)
A	0.029
B	0.032
C	0.087 (0.000)
D	7.563 (0.531)
E	1.711 (0.401)
F	0.008 (0.000)
G	1.671 (0.368)
H	1.655 (1.203)
I	0.008 (0.000)
J	0.008 (0.000)
K	0.008
L	1.434

(d) VPP500, 並列化版, 1PE

計測範囲	計測時間 (秒)
A	0.850
B	0.938
C	0.052 (0.000)
D	11.827 (0.789)
E	2.482 (0.708)
F	0.000 (0.000)
G	2.510 (0.707)
H	2.582 (1.774)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	2.121

Table 5.3 Data B09.S (2/2).

(e) AP3000, 並列化版, 2PE

計測範囲	計測時間 (秒)
A	0.033
B	0.031
C	0.055 (0.000)
D	7.009 (0.642)
E	1.454 (0.540)
F	0.009 (0.000)
G	1.401 (0.528)
H	2.346 (1.957)
I	0.009 (0.000)
J	0.008 (0.000)
K	0.008
L	1.097

(f) VPP500, 並列化版, 2PE

計測範囲	計測時間 (秒)
A	2.286
B	0.976
C	0.039 (0.000)
D	9.291 (0.945)
E	2.715 (0.834)
F	0.000 (0.000)
G	2.704 (0.866)
H	2.790 (2.277)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	7.598

(g) AP3000, 並列化版, 4PE

計測範囲	計測時間 (秒)
A	0.597
B	0.089
C	0.034 (0.000)
D	8.884 (0.991)
E	1.666 (0.898)
F	0.008 (0.000)
G	1.504 (0.849)
H	3.153 (2.748)
I	0.009 (0.000)
J	0.008 (0.000)
K	0.008
L	0.946

(h) VPP500, 並列化版, 4PE

計測範囲	計測時間 (秒)
A	0.574
B	0.913
C	0.026 (0.000)
D	9.217 (1.117)
E	1.824 (1.019)
F	0.000 (0.000)
G	1.949 (1.131)
H	3.270 (2.934)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	1.854

Table 5.4 Data B10.S (1/3).

(a) AP3000, 倍精度化版

計測範囲	計測時間 (秒)
A	2.866
B	0.013
C	0.988 (0.000)
D	36.607 (12.674)
E	36.000 (20.330)
F	35.619 (18.334)
G	202.512 (188.388)
H	31.417 (26.068)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	34.122

(b) VPP500, 倍精度化版

(メモリ不足のため実行不可能)

(c) AP3000, 並列化版, 1PE

計測範囲	計測時間 (秒)
A	5.998
B	0.182
C	1.278 (0.000)
D	39.761 (14.196)
E	41.187 (23.041)
F	39.648 (20.970)
G	229.764 (212.001)
H	35.504 (29.093)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	71.307

(d) VPP500, 並列化版, 1PE

計測範囲	計測時間 (秒)
A	5.425
B	0.060
C	0.069 (0.000)
D	29.596 (4.245)
E	24.917 (4.932)
F	29.054 (5.049)
G	83.111 (60.862)
H	19.828 (8.434)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	26.665

Table 5.4 Data B10.S (2/3).

(e) AP3000, 並列化版, 2PE

(メモリ不足のため実行不可能)

(f) VPP500, 並列化版, 2PE

計測範囲	計測時間 (秒)
A	6.914
B	0.051
C	0.041 (0.000)
D	13.199 (2.540)
E	13.837 (2.858)
F	15.981 (2.966)
G	48.975 (35.827)
H	10.294 (4.720)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	32.013

(g) AP3000, 並列化版, 4PE

計測範囲	計測時間 (秒)
A	2.694
B	0.212
C	0.254 (0.000)
D	10.525 (4.621)
E	13.778 (9.373)
F	12.821 (8.380)
G	65.505 (62.037)
H	10.468 (9.042)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	42.780

(h) VPP500, 並列化版, 4PE

計測範囲	計測時間 (秒)
A	3.730
B	0.034
C	0.019 (0.000)
D	7.226 (1.490)
E	7.303 (2.045)
F	8.008 (2.208)
G	27.836 (22.221)
H	5.628 (3.011)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	17.368

Table 5.4 Data B10.S (3/3).

(i) AP3000，並列化版，8PE

(プロセッサ台数の関係で実行不可能)

(j) VPP500，並列化版，8PE

計測範囲	計測時間（秒）
A	6.118
B	0.035
C	0.010 (0.000)
D	4.258 (1.163)
E	4.736 (1.214)
F	5.089 (1.322)
G	20.759 (15.424)
H	3.453 (2.092)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	25.429

(k) AP3000，並列化版，16PE

(プロセッサ台数の関係で実行不可能)

(l) VPP500，並列化版，16PE

計測範囲	計測時間（秒）
A	8.619
B	0.039
C	0.005 (0.000)
D	2.504 (0.917)
E	3.824 (1.204)
F	4.993 (1.211)
G	19.121 (13.782)
H	2.578 (1.872)
I	0.000 (0.000)
J	0.000 (0.000)
K	0.000
L	26.647

```
        :
/* void main(int argc, char ** argv) */          ← コメント化
MAIN__()                                         ← 挿入
{
        :
}
        :
```

Fig. 5.1 Change of 'Main function' name.

```
  TWC(7 6 8 0 10) SRP
// EXEC CENGINE,REGION.STRM77=,
// A='READ,ERASE,NOSUB',INC=NOINCLUDE
//FT05F001 DD *
SUP(0200,0300,0400,0500,0600,0700,0800,0900)
SUP(1000,1100,1200,1300,1400,1500,1600)
/*
//FT11F001 DD DSN=JXXXX.STRM310.FORT77,DISP=SHR,LABEL=(,,,IN)
//FT12F001 DD DSN=JXXXX.STRM310.DBL.FORT77,DISP=(NEW,CATLG),
//    UNIT=TSSWK,SPACE=(CYL,(10,10,100),RLSE),
//    DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120,DSORG=PO)
```

Fig. 5.2 JCL for STREAM77.

```
        :
      INCLUDE 'mpif.h'
      INCLUDE 'PARA'
        :
      CALL MPI_INIT(IERROR)
      ICOMM = MPI_COMM_WORLD
      CALL MPI_COMM_SIZE(ICOMM,NPROCS,IERROR)
      CALL MPI_COMM_RANK(ICOMM,MYRANK,IERROR)
C
      IF ( NPROCS.GT.MXPRCS ) THEN
         WRITE(*,*) 'TOO MANY PROCESSORS'
         CALL MPI_ABORT(MPI_COMM_WORLD,0,IERROR)
      ENDIF
C
      IF ( MYRANK.EQ.0 ) THEN
         ILPROC = MPI_PROC_NULL
      ELSE
         ILPROC = MYRANK - 1
      ENDIF
      IF ( MYRANK.EQ.NPROCS-1 ) THEN
         IRPROC = MPI_PROC_NULL
      ELSE
         IRPROC = MYRANK + 1
      ENDIF
C
      IROOT = 0
        :
```

Fig. 5.3 Initialization process for parallel execution.

```
      PARAMETER ( MXPRCS = 256, MWDOLL = 2, MWDOLR = 2 )
      COMMON /PRCS/ NPROCS, MYRANK, ILPROC, IRPROC, IROOT, ICOMM
      COMMON /BODM/ KLBM, KUBM, KLBMP, KUBMP, KLBMPP, KUBMPP
      COMMON /DOPR/ KST1, KST2, KST1PP, KEDM, KEDMP, KEDMPP
      COMMON /CHRG/ KCHM(0:MXPRCS-1), KCHMP(0:MXPRCS-1),
     $              KCHMPP(0:MXPRCS-1)
```

名前	概要
MXPRCS	計算に使用するノード数の上限値.
MWDOLL	分割された配列の左袖の幅. 最左端の領域には左袖が付加されないことに注意.
MWDOLR	分割された配列の右袖の幅. 最右端の領域には右袖が付加されないことに注意.
NPROCS	計算に使用するノード数.
MYRANK	自ノードのノード番号.
ILPROC	左隣の領域を担当するノードのノード番号. 左隣が存在しないノードでは ILPROC に MPI_PROC_NULL が代入される.
IRPROC	右隣の領域を担当するノードのノード番号. 右隣が存在しないノードでは ILPROC に MPI_PROC_NULL が代入される.
IROOT	ファイル入出力を行うノードのノード番号. VPP500 及び AP3000 では全てノードがファイル入出力可能であるため, どのノードを指定しても良い. 例外として, IROOT 以外のノードがファイル入出力を行う場合もあることに注意.
ICOMM	プログラム中で使用されるコミュニケータ. 本コードでは MPI_COMM_WORLD の値を代入しておく.
KLBM KUBM KUBM KLBMP KUBMP KLBMPP KUBMPP	分割された配列の宣言において, K 軸方向のインデックスの初期値と終端値に使用.
KST1 KST2 KST1PP KEDM KEDMP KEDMPP	K 軸方向の DO ループにおいて, DO 変数の初期値と終端値に使用.
KCHM	K 軸方向の要素数が KM である配列を分割した際に各ノードに割り当てられる K 軸方向の要素数. ファイル入出力の際に, 分割されたデータをノード間で収集・拡散するために使用される.
KCHMP	K 軸方向の要素数が KMP である配列を分割した際に各ノードに割り当てられる K 軸方向の要素数. ファイル入出力の際に, 分割されたデータをノード間で収集・拡散するために使用される.
KCHMPP	K 軸方向の要素数が KMPP である配列を分割した際に各ノードに割り当てられる K 軸方向の要素数. ファイル入出力の際に, 分割されたデータをノード間で収集・拡散するために使用される.

Fig. 5.4 Outline of include file PARA.

```
        :
      KDIV = KM / NPROCS
      KMOD = KM - KDIV*NPROCS
      DO 100 I=0,NPROCS-1
         IF ( I.LT.KMOD ) THEN
            KCHM  (I) = KDIV + 1
            KCHMP (I) = KDIV + 1
            KCHMPP(I) = KDIV + 1
         ELSE
            KCHM  (I) = KDIV
            KCHMP (I) = KDIV
            KCHMPP(I) = KDIV
         ENDIF
 100  CONTINUE
      KCHMP (NPROCS-1) = KCHMP (NPROCS-1) + 1
      KCHMPP(0       ) = KCHMPP(0       ) + 1
      KCHMPP(NPROCS-1) = KCHMPP(NPROCS-1) + 1
C
      KST1   = 1
      KST1PP = 1
      DO 101 I=0,MYRANK-1
         KST1   = KST1   + KCHM  (I)
         KST1PP = KST1PP + KCHMPP(I)
 101  CONTINUE
      IF ( MYRANK.EQ.0 ) THEN
         KST2 = 2
      ELSE
         KST2 = KST1
      ENDIF
C
      KEDM   = KST1   + KCHM  (MYRANK) - 1
      KEDMP  = KST1   + KCHMP (MYRANK) - 1
      KEDMPP = KST1PP + KCHMPP(MYRANK) - 1
C
      IF ( MYRANK.EQ.0 ) THEN
         KLBM   = KST1
         KLBMP  = KST1
         KLBMPP = KST1PP
      ELSE
         KLBM   = KST1   - MWDOLL
         KLBMP  = KST1   - MWDOLL
         KLBMPP = KST1PP - MWDOLL
      ENDIF
      IF ( MYRANK.EQ.NPROCS-1 ) THEN
         KUBM   = KEDM
         KUBMP  = KEDMP
         KUBMPP = KEDMPP
      ELSE
         KUBM   = KEDM   + MWDOLR
         KUBMP  = KEDMP  + MWDOLR
         KUBMPP = KEDMPP + MWDOLR
      ENDIF
        :
```

Fig. 5.5 Regions decision of process for each processor.

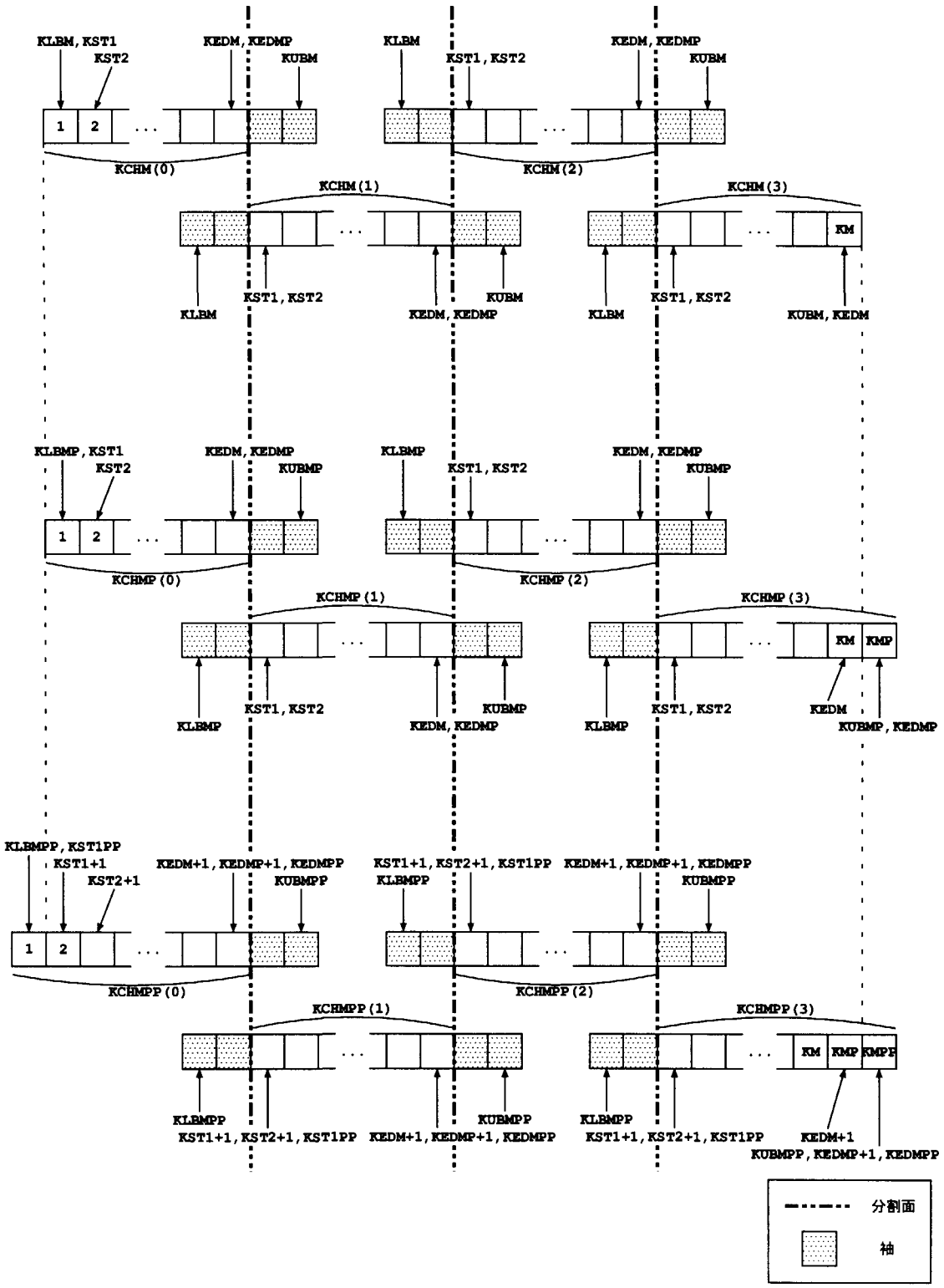


Fig. 5.6 Relationship between common variables.

```
      :
    DIMENSION A(IMP,JMP,KMP)
      :
    DO 100 K=1,KM
    DO 100 J=1,JM
    DO 100 I=1,IM
    A(I,J,K)=0.0D0
100 CONTINUE
      :
```

(a) 変更前

```
      :
    INCLUDE 'PARA'
      :
    DIMENSION A(IMP,JMP,KLBMP:KUBMP)   ! データの分割
      :
    DO 100 K=KST1,KEDM                 ! 処理の分割
    DO 100 J=1,JM
    DO 100 I=1,IM
    A(I,J,K)=0.0D0
100 CONTINUE
      :
```

(b) 変更後

Fig. 5.7 Example of procedure decomposition.

```
      :
    DIMENSION A(IMP,JMP,KMP)
      :
    K1=...                                  ! K1 の定義
    K2=...                                  ! K2 の定義
    J1=...
    J2=...
    I1=...
    I2=...
      :
    DO 100 K=K1,K2                          ! 対象の DO ループ
    DO 100 J=J1,J2
    DO 100 I=I1,I2
    A(I,J,K)=...
100 CONTINUE
      :
```

(a) 変更前

```
      :
    INCLUDE 'PARA'
      :
    DIMENSION A(IMP,JMP,KLBMP:KUBMP)  ! データの分割
      :
    K1=...                                  ! K1 の定義
    K2=...                                  ! K2 の定義
    J1=...
    J2=...
    I1=...
    I2=...
      :
    DO 100 K=MAX(K1,KST1),MIN(K2,KEDMP)        ! 処理の分割
    DO 100 J=J1,J2
    DO 100 I=I1,I2
    A(I,J,K)=...
100 CONTINUE
      :
```

(b) 変更後

Fig. 5.8 Procedure decomposition.

```
      :
    DO 100 K=MAX(K1,KST1),MIN(K2,KEDMP)        ! 処理の分割
    DO 100 J=J1,J2
    DO 100 I=I1,I2
    A = A + ...
100 CONTINUE
                                               ! ノードごとに得られた
                                               ! 結果の総和
    CALL MPI_ALLREDUCE(A,RTMP,1,MPI_REAL8,MPI_SUM,ICOMM,IERROR)
    A = RTMP
      :
```

Fig. 5.9 Summation by MPI libraries.

```
      :
      DO 100 K=KST1,KEDM          ! 処理の分割
      DO 100 J=1,JM
      DO 100 I=1,IM
      FMI1=...
      IF ( FMI1.GT.FMI0 ) GOTO 100
      IMI=I
      JMI=J
      KMI=K
      FMI0=FMI1
  100 CONTINUE
                                  ! IMI, JMI, KMI を一つの値にまとめる
      ITMP = (IMP+1)*(JMP+1)*(KMI-1)+(IMP+1)*(JMI-1)+IMI
      RTMP1(1) = FMI0
      RTMP1(2) = ITMP
      CALL MPI_ALLREDUCE(RTMP1,RTMP2,1,MPI_2DOUBLE_PRECISION,
     $                   MPI_MINLOC,ICOMM,IERROR)
      FMI0 = RTMP2(1)
      ITMP = RTMP2(2)
                                  ! IMI, JMI, KMI の復元
      KMI = ITMP / ((IMP+1)*(JMP+1)) + 1
      ITMP = ITMP - (IMP+1)*(JMP+1)*(KMI-1)
      JMI = ITMP / (IMP+1) + 1
      IMI = ITMP - (IMP+1)*(JMI-1)
        :
```

Fig. 5.10 Example of parallelization of calculating index of array element that has minimum value.

```
      :
      DO 100 K=KST1,KEDM          ! 処理の分割
      DO 100 J=1,JM
      DO 100 I=1,IM
      DX= ...
      DY= ...
      DZ= ...
      ANOW=...
      IF(ANOW.GT.AOLD) THEN
      IMAX=I
      JMAX=J
      KMAX=K
      DXX=DX
      DYY=DY
      DZZ=DZ
      AOLD=ANOW
      ENDIF
  100 CONTINUE
C
      RTMP1(1) = AOLD
      RTMP1(2) = MYRANK
                                  ! 最大値を持つプロセッサの決定
      CALL MPI_ALLREDUCE(RTMP1,RTMP2,1,MPI_2DOUBLE_PRECISION,
     $                   MPI_MAXLOC,ICOMM,IERROR)
      AOLD   = RTMP2(1)
      IROOT0 = RTMP2(2)
C
      RTMP3(1) = IMAX
      RTMP3(2) = JMAX
      RTMP3(3) = KMAX
      RTMP3(4) = DXX
      RTMP3(5) = DYY
      RTMP3(6) = DZZ
                                  ! 最大値を持つプロセッサから必要な情報
                                  ! をブロードキャストする.
      CALL MPI_BCAST(RTMP3,6,MPI_REAL8,IROOT0,ICOMM,IERROR)
      IMAX = RTMP3(1)
      JMAX = RTMP3(2)
      KMAX = RTMP3(3)
      DXX  = RTMP3(4)
      DYY  = RTMP3(5)
      DZZ  = RTMP3(6)
        :
```

Fig. 5.11 Example of parallelization of MAX calculation.

```
        :
      DO 100 K=K1,K2
      DO 100 J=J1,J2
      DO 100 I=I1,I2
        :
      IF ( ... ) THEN
        IF ( A(I,J,K  ).NE.0.0D0 ) THEN
          A(I,J,K  )=A(7,I,J,K  )+...
        ENDIF
        IF ( A(I,J,K+1).NE.0.0D0 ) THEN
          A(I,J,K+1)=COF(7,I,J,K+1)+...
        ENDIF
        :
```

Fig. 5.12 Example of a loop which is hard to be parallelized (before parallelization).

```
  :
DO 100 KKK=0,1              ! 2度に分けて処理を行う.
                            ! 処理の分割
DO 100 KK=MAX(K1+KKK,KST1),MIN(K2+KKK,KEDMP)
   K=KK-KKK
DO 100 J=J1,J2
DO 100 I=I1,I2
  :
IF ( ... ) THEN
  IF ( K.EQ.KK ) THEN
                            ! K.EQ.KK が成立する時 K 番目の要素の
                            ! 定義を行う.
  IF ( A(I,J,K)  .NE.0.0D0 ) THEN
    A(I,J,K  )=A(7,I,J,K  )+...
  ENDIF
  ELSE IF ( K+1.EQ.KK ) THEN
                            ! K+1.EQ.KK が成立する時 K+1 番目の要
                            ! 素の定義を行う.
  IF ( A(I,J,K+1).NE.0.0D0 ) THEN
    A(I,J,K+1)=A(7,I,J,K+1)+...
  ENDIF
  ENDIF
  :
```

Fig. 5.13 Example of a loop which is hard to be parallelized (after parallelization).

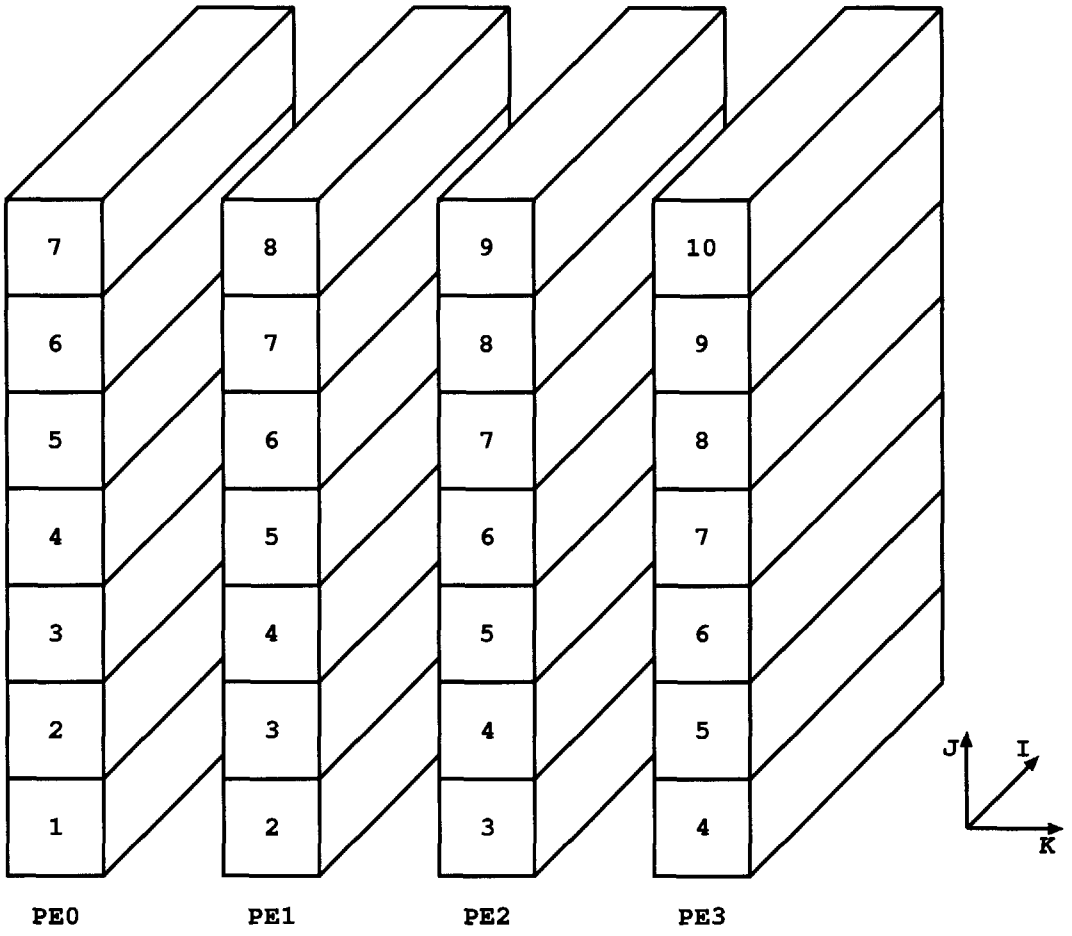


Fig. 5.14 Concept of procedure decomposition.

```
      PARAMETER ( MXOVLP = 100, NOINFO = 8 )
      COMMON /OVLP/ NOOVLP, IADDRS(MXOVLP), INFO(NOINFO,MXOVLP)
      COMMON /OVLC/ CNAME(MXOVLP)
      CHARACTER*6 CNAME
C
C     INFO(1,*) : STATUS OF LEFT SIDE SHASOW ( 0 : NOT FIXED, 1 : FIXED )
C     INFO(2,*) :            RIGHT
C     INFO(3,*) : NUMBER OF ELEMENTS IN FIRST DIMENSION
C     INFO(4,*) : LOWER BOUNDARY IN SECOND DIMENSION
C     INFO(5,*) : UPPER
C     INFO(6,*) : LOWER BOUNDARY EXCEPT OF LEFT SIZE SHADOW IN 2ND DIM.
C     INFO(7,*) : UPPER                    RIGHT
C     INFO(8,*) : TYPE ( 0 : INTEGER, 1 : REAL*8 )
```

名前	概要
MXOVLP	袖に関する情報が登録される配列の個数の上限.
NOINFO	配列 INFO の1次元目の要素数.
NOOVLP	袖に関する情報が登録された配列の総数.
IADDRS	各配列の先頭アドレス. 各配列は, この先頭アドレスで識別される.
INFO(1,*)	左袖の状態を格納しておく. 値が 0 であれば袖転送が行われていないことを, 1 であれば袖転送が完了していることを示す. この情報は, サブルーチン OLRST で 0 に, サブルーチン OLFIXI または OLFIXR で 1 に定義される.
INFO(2,*)	右袖の状態を格納しておく. 値が 0 であれば袖転送が行われていないことを, 1 であれば袖転送が完了していることを示す. この情報は, サブルーチン OLRST で 0 に, サブルーチン OLFIXI または OLFIXR で 1 に定義される.
INFO(3,*)	配列の I 軸方向の要素数と J 軸方向の要素数の積.
INFO(4,*)	K 軸（分割軸）方向のインデックスの初期値. 袖部分を含む.
INFO(5,*)	K 軸（分割軸）方向のインデックスの終端値. 袖部分を含む.
INFO(6,*)	K 軸（分割軸）方向のインデックスの初期値. ただし, 袖部分は含まない.
INFO(7,*)	K 軸（分割軸）方向のインデックスの終端値. ただし, 袖部分は含まない.
INFO(8,*)	配列の型情報. 値が 0 であれば整数型, 1 であれば倍精度実数型であることを示す.
CNAME	各配列の配列名. エラー出力などで使用する.

Fig. 5.15 Include file OVLP.

```
      SUBROUTINE OLINIT
      IMPLICIT REAL*8 ( A-H, O-Z )
      INCLUDE 'OVLP'
C
      NOOVLP = 0
C
      RETURN
      END
```

Fig. 5.16 Subroutine OLINIT.

```
      SUBROUTINE OLREG(ARRAY,N1,NK1,NK2,KLB,KRB,ITYP,CNAM)
      IMPLICIT REAL*8 ( A-H, O-Z )
      DIMENSION ARRAY(*)
      CHARACTER*6 CNAM
      INCLUDE 'mpif.h'
      INCLUDE 'OVLP'
C
      CALL MPI_ADDRESS(ARRAY,IADR,IERROR)
C
      DO 100 I = 1, NOOVLP
         IF ( IADR.EQ.IADDRS(I) ) THEN
C            WRITE(*,*) 'OLREG ALREADY REGISTERED'
C            CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
            RETURN
         ENDIF
 100  CONTINUE
C
      NOOVLP = NOOVLP + 1
C
      IF ( NOOVLP.GT.MXOVLP ) THEN
         WRITE(*,*) 'OLREG NOOVLP.GT.MXOVLP'
         CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
      ENDIF
C
      IADDRS(NOOVLP) = IADR
      INFO(1,NOOVLP) = 0
      INFO(2,NOOVLP) = 0
      INFO(3,NOOVLP) = N1
      INFO(4,NOOVLP) = NK1
      INFO(5,NOOVLP) = NK2
      INFO(6,NOOVLP) = KLB
      INFO(7,NOOVLP) = KRB
      INFO(8,NOOVLP) = ITYP
      CNAME(NOOVLP)  = CNAM
C
      RETURN
      END
```

Fig. 5.17 Subroutine OLREG.

```
      SUBROUTINE OLSORT
      IMPLICIT REAL*8 ( A-H, O-Z )
      INCLUDE 'OVLP'
      CHARACTER*6 CTMP
C
      DO 100 J = 1, NOOVLP-1
         K = J
         DO 110 I = J+1, NOOVLP
            IF ( IADDRS(I).LT.IADDRS(K) ) K = I
 110     CONTINUE
         IF ( J.NE.K ) THEN
            ITMP      = IADDRS(J)
            IADDRS(J) = IADDRS(K)
            IADDRS(K) = ITMP
C
            DO 120 L = 1, NOINFO
               ITMP      = INFO(L,J)
               INFO(L,J) = INFO(L,K)
               INFO(L,K) = ITMP
 120        CONTINUE
C
            CTMP      = CNAME(J)
            CNAME(J) = CNAME(K)
            CNAME(K) = CTMP
         ENDIF
 100  CONTINUE
C
      RETURN
      END
```

Fig. 5.18 Subroutine OLSORT.

```
      SUBROUTINE OLRST(ARRAY)
      IMPLICIT REAL*8 ( A-H, O-Z )
      DIMENSION ARRAY(*)
C
      INCLUDE 'mpif.h'
      INCLUDE 'OVLP'
C
      CALL MPI_ADDRESS(ARRAY,IADR,IERROR)
C
      IMIN = 1
      IMAX = NOOVLP
  100 CONTINUE
         I = IMIN + (IMAX-IMIN)/2
         IF ( IADR.EQ.IADDRS(I) ) THEN
            GOTO 200
         ELSEIF ( IMIN.EQ.IMAX ) THEN
            WRITE(*,*) 'OLRST INVALID ADDR'
            CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
         ELSEIF ( IADR.GT.IADDRS(I) ) THEN
            IMIN = I + 1
         ELSE
            IMAX = I - 1
         ENDIF
      GOTO 100
  200 CONTINUE
C
      INFO(1,I) = 0
      INFO(2,I) = 0
C
      RETURN
      END
```

Fig. 5.19 Subroutine OLRST.

```
      SUBROUTINE OLFIXI(IARRAY,N1,NK1,NK2,N2,KLB,KRB,ILFIX,IRFIX,IFORCE)
      IMPLICIT REAL*8 ( A-H, O-Z )
      DIMENSION IARRAY(N1,NK1:NK2,N2)
C
      INCLUDE 'mpif.h'
      INCLUDE 'PARA'
      INCLUDE 'OVLP'
      DIMENSION ISTATS(MPI_STATUS_SIZE)
C
      IF ( N1.LT.1 .OR. NK1.GT.NK2 .OR. N2.NE.1 .OR.
     $     ( KLB.NE.NK1 .AND. KLB.NE.NK1+MWDOLL ) .OR.
     $     ( KRB.NE.NK2 .AND. KRB.NE.NK2-MWDOLR ) .OR.
     $     ( ILFIX.EQ.0 .AND. IRFIX.EQ.0 ) ) THEN
         WRITE(*,*) 'OLFIXI INVALID ARGUMENT'
         CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
      ENDIF
C
      CALL MPI_ADDRESS(IARRAY,IADR,IERROR)
C
      IMIN = 1
      IMAX = NOOVLP
  100 CONTINUE
         I = IMIN + (IMAX-IMIN)/2
         IF ( IADR.EQ.IADDRS(I) ) THEN
            GOTO 200
         ELSEIF ( IMIN.EQ.IMAX ) THEN
            WRITE(*,*) 'OLFIXI UNREGISTERED ADDR'
            CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
         ELSEIF ( IADR.GT.IADDRS(I) ) THEN
            IMIN = I + 1
         ELSE
            IMAX = I - 1
         ENDIF
      GOTO 100
  200 CONTINUE
C
      IF ( N1 .NE.INFO(3,I) .OR.
     $     NK1.NE.INFO(4,I) .OR.
     $     NK2.NE.INFO(5,I) .OR.
     $     KLB.NE.INFO(6,I) .OR.
     $     KRB.NE.INFO(7,I) .OR.
     $     0  .NE.INFO(8,I) ) THEN
         WRITE(*,*) ' *** MISS MATCH OLFIX *** ', MYRANK,
     $        N1, NK1, NK2, KLB, KRB, 'I ',
     $        (INFO(II,I),II=3,8), CNAME(I)
      ENDIF
```

Fig. 5.20 Subroutine OLFIXI (1/2).

```
C
      IF ( ILFIX.NE.0 .AND.
     $     ( IFORCE.NE.0 .OR. INFO(1,I).EQ.0 ) ) THEN
          INFO(1,I) = 1
          CALL MPI_SENDRECV(IARRAY(1,KRB-MWDOLL+1,1),N1*MWDOLL,
     $                      MPI_INTEGER,IRPROC,0,
     $                      IARRAY(1,KLB-MWDOLL  ,1),N1*MWDOLL,
     $                      MPI_INTEGER,ILPROC,0,
     $                      ICOMM,ISTATS,IERROR)
      ENDIF
      IF ( IRFIX.NE.0 .AND.
     $     ( IFORCE.NE.0 .OR. INFO(2,I).EQ.0 ) ) THEN
          INFO(2,I) = 1
          CALL MPI_SENDRECV(IARRAY(1,KLB  ,1),N1*MWDOLR,
     $                      MPI_INTEGER,ILPROC,0,
     $                      IARRAY(1,KRB+1,1),N1*MWDOLR,
     $                      MPI_INTEGER,IRPROC,0,
     $                      ICOMM,ISTATS,IERROR)
      ENDIF
C
      RETURN
      END
```

Fig. 5.20 Subroutine OLFIXI (2/2).

```
      SUBROUTINE OLFIXR(ARRAY,N1,NK1,NK2,N2,KLB,KRB,ILFIX,IRFIX,IFORCE)
      IMPLICIT REAL*8 ( A-H, O-Z )
      DIMENSION ARRAY(N1,NK1:NK2,N2)
C
      INCLUDE 'mpif.h'
      INCLUDE 'PARA'
      INCLUDE 'OVLP'
      DIMENSION ISTATS(MPI_STATUS_SIZE)
C
      IF ( N1.LT.1 .OR. NK1.GT.NK2 .OR. N2.NE.1 .OR.
     $     ( KLB.NE.NK1 .AND. KLB.NE.NK1+MWDOLL ) .OR.
     $     ( KRB.NE.NK2 .AND. KRB.NE.NK2-MWDOLR ) .OR.
     $     ( ILFIX.EQ.0 .AND. IRFIX.EQ.0 ) ) THEN
         WRITE(*,*) 'OLFIXR INVALID ARGUMENT'
         CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
      ENDIF
C
      CALL MPI_ADDRESS(ARRAY,IADR,IERROR)
C
      IMIN = 1
      IMAX = NOOVLP
  100 CONTINUE
         I = IMIN + (IMAX-IMIN)/2
         IF ( IADR.EQ.IADDRS(I) ) THEN
            GOTO 200
         ELSEIF ( IMIN.EQ.IMAX ) THEN
            WRITE(*,*) 'OLFIXR UNREGISTERED ADDR'
            CALL MPI_ABORT(MPI_COMM_WORLD,999,IERROR)
         ELSEIF ( IADR.GT.IADDRS(I) ) THEN
            IMIN = I + 1
         ELSE
            IMAX = I - 1
         ENDIF
      GOTO 100
  200 CONTINUE
C
      IF ( N1 .NE.INFO(3,I) .OR.
     $     NK1.NE.INFO(4,I) .OR.
     $     NK2.NE.INFO(5,I) .OR.
     $     KLB.NE.INFO(6,I) .OR.
     $     KRB.NE.INFO(7,I) .OR.
     $     1  .NE.INFO(8,I) ) THEN
         WRITE(*,*) ' *** MISS MATCH OLFIX *** ', MYRANK,
     $        N1, NK1, NK2, KLB, KRB, 'R ',
     $        (INFO(II,I),II=3,8), CNAME(I)
      ENDIF
```

Fig. 5.21 Subroutine OLFIXR (1/2).

```
C
      IF ( ILFIX.NE.0 .AND.
     $     ( IFORCE.NE.0 .OR. INFO(1,I).EQ.0 ) ) THEN
          INFO(1,I) = 1
          CALL MPI_SENDRECV(ARRAY(1,KRB-MWDOLL+1,1),N1*MWDOLL,
     $                      MPI_REAL8,IRPROC,0,
     $                      ARRAY(1,KLB-MWDOLL  ,1),N1*MWDOLL,
     $                      MPI_REAL8,ILPROC,0,
     $                      ICOMM,ISTATS,IERROR)
      ENDIF
      IF ( IRFIX.NE.0 .AND.
     $     ( IFORCE.NE.0 .OR. INFO(2,I).EQ.0 ) ) THEN
          INFO(2,I) = 1
          CALL MPI_SENDRECV(ARRAY(1,KLB  ,1),N1*MWDOLR,
     $                      MPI_REAL8,ILPROC,0,
     $                      ARRAY(1,KRB+1,1),N1*MWDOLR,
     $                      MPI_REAL8,IRPROC,0,
     $                      ICOMM,ISTATS,IERROR)
      ENDIF
C
      RETURN
      END
```

Fig. 5.21 Subroutine OLFIXR (2/2).

```
      DIMENSION A(IMP,JMP,KLBMP:KUBMP), B(IMP,JMP,KLBMP:KUBMP)

          :

      DO 100 K=KST1,KEDMP
        A(I,J,K)=...
        B(I,J,K)=...
  100 CONTINUE
      CALL OLRST(A)
      CALL OLRST(B)

      CALL SUB1(...,A,B,...)

      CALL OLFIXR(A,IMP*JMP,KLBMP,KUBMP,KST1,KEDMP,1,0,0)
      CALL OLFIXR(B,IMP*JMP,KLBMP,KUBMP,KST1,KEDMP,0,1,0)
      DO 200 K=KST1,KEDM
        ...=A(I,J,K-1)+B(I,J,K+1)
  200 CONTINUE

          :

      CALL OLFIXR(A,IMP*JMP,KLBMP,KUBMP,KST1,KEDMP,1,0,0)
      CALL OLFIXR(B,IMP*JMP,KLBMP,KUBMP,KST1,KEDMP,0,1,0)
      DO 300 K=KST1,KEDM
        CALL SUB2(I,J,K,...,A,B,...)
  300 CONTINUE

          :

      END
C--------------------------------------
      SUBROUTINE SUB2(I,J,K,...,A,B,...)

          :
        ...=A(I,J,K-1)+B(I,J,K+1)
          :
```

Fig. 5.22 Use example of overlap-fix subroutines.

```
        :
      IF ( MYRANK.EQ.IROOT ) THEN
      READ(5) A
      ENDIF
      CALL MPI_BCAST(A,1,MPI_REAL8,IROOT,ICOMM,IERROR)
        :
```

Fig. 5.23 Example of reading data which are not decomposed.

```
        :
      INCLUDE 'mpif.h'
      INCLUDE 'PARA'
        :
      DIMENSION A(NI,NJ,KLBMP:KUBMP)     ! 分割された配列
      DIMENSION WORK(NWORK)              ! 作業用配列
      DIMENSION ISDCNT(0:MXPRCS-1), IDISPL(0:MXPRCS-1)
        :
                                         ! １台のプロセッサに全データを読み込む
        IF ( MYRANK.EQ.IROOT ) THEN
          READ(...) (WORK(I),I=1,NI*NJ*NK)
        ENDIF
C
                                         ! 各プロセッサへ転送する要素数の計算
           DO 101 I = 0, NPROCS-1
              ISDCNT(I) = KCHMP(I)*NI*NJ
 101       CONTINUE
                                         ! 各プロセッサへ転送するデータの先頭
                                         ! 要素のインデックス計算
           IDISPL(0) = 0
           DO 102 I = 1, NPROCS-1
              IDISPL(I) = IDISPL(I-1) + ISDCNT(I-1)
 102       CONTINUE
                                         ! バリア同期
           CALL MPI_BARRIER(ICOMM,IERROR)
                                         ! 全プロセッサへのデータの転送
           CALL MPI_SCATTERV(WORK,ISDCNT,IDISPL,MPI_REAL8,
     $                       A(1,1,KST1),ISDCNT(MYRANK),MPI_REAL8,
     $                       IROOT,ICOMM,IERROR)
           CALL OLRST(A)
        :
```

Fig. 5.24 Example of reading data which are decomposed.

```
      :
      INCLUDE 'mpif.h'
      INCLUDE 'PARA'
      DIMENSION A(NI,NJ,KLBMP:KUBMP)     ! 分割された配列
      DIMENSION WORK(NWORK)              ! 作業用配列
      DIMENSION ISDCNT(0:MXPRCS-1), IDISPL(0:MXPRCS-1)
      :
                                         ! 各プロセッサから転送される要素数
                                         ! の計算
         DO 201 I = 0, NPROCS-1
            ISDCNT(I) = KCHMP(I)*NI*NJ
201      CONTINUE
                                         ! 各プロセッサから転送されたデータを
                                         ! 格納する領域の先頭要素のインデック
                                         ! ス計算
         IDISPL(0) = 0
         DO 202 I = 1, NPROCS-1
            IDISPL(I) = IDISPL(I-1) + ISDCNT(I-1)
202      CONTINUE
                                         ! バリア同期
         CALL MPI_BARRIER(ICOMM,IERROR)
                                         ! 全プロセッサからのデータ転送
         CALL MPI_GATHERV(A(1,1,KST1),ISDCNT(MYRANK),MPI_REAL8,
     $                    WORK,ISDCNT,IDISPL,MPI_REAL8,
     $                    IROOT,ICOMM,IERROR)
      ENDIF
C
                                         ! １台のプロセッサからの出力
      IF ( MYRANK.EQ.IROOT ) THEN
        WRITE(...) (WORK(I),I=1,NI*NJ*NK)
      ENDIF
      :
```

Fig. 5.25 Example of output of decomposed data.

参考文献

[1] 根本俊行，渡辺秀雄，他；原子力コードの VPP500 におけるベクトル化，並列化及び移植，JAERI-Data/Code 96-022，1996 年 7 月.

[2] 渡辺秀雄，功刀資彰；3 次元非定常圧縮性熱流体コード「STREAM ver. 3.1」の利用法，JAERI-memo 08-098，1996 年 3 月.

[3] MPI: A Message-Passing Interface Standard，Message Passing Interfase Forum，May 10, 1996.

[4] FUJITSU Fortran 90 使用手引書 V2 用，富士通 (株)，1994 年 10 月.

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

[6] FACOM OS IV/F4 MSP STREAM77 説明書，富士通 (株)，1985 年 2 月.

6. おわりに

計算科学技術推進センター情報システム管理課で実施しているVPP500向けの原子力コードの高速化作業は，毎年10数件を順調にこなし，平成9年度に14件の作業を完了，平成10年度にも15件の作業が計画されている．これら作業は，ユーザからの依頼に応じ，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うものであり，コード実行時間の大幅な短縮に寄与している．さらに，単一プロセッサ上ではメモリ不足から実行できないようなジョブを並列化効果により可能にするなど，計算機のスループットの向上，ターンアラウンドタイムの短縮，それによるユーザの仕事の効率化，計算可能なジョブの範囲の拡大など，計算機の効率的な運用と計算機資源の有効利用に大いに貢献するものと考えている．

本報告書では，円筒座標系直接数値解析コードCYLDNS44N，放射能粒子拡散予測コードWSPEEDI，拡張量子分子動力学コードEQMD及び三次元熱流体解析コードSTREAMの4本の原子力コードの並列化（一部ベクトル化を含む）作業について記述した．各コードとも並列化によりその高速化効果が顕著に表れているが，まだ残された課題，改良すべき点，すなわち高速化の余地があるものも少なくない．今後これらの点を見極め，各コードの計算アルゴリズムの見直し，I/Oの効率化，C言語プログラムへの高速化対応等を進めることで，より一層の高速化が期待できるものと思われる．また，今後のより高性能な並列型計算機への交換をも考慮し，移植性・汎用性を重視した並列化技術であるMPIの利用を推進していく計画である．

最後に，本報告書がこれらの仕事に携わる人々に多少なり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 伝熱流動研究室 功刀資彰氏(2章及び5章)，環境物理研究室 茅野政道氏（3章），ハドロン輸送研究グループ 丸山敏毅氏（4章）には，コード内容の把握に際し御協力頂きました．また，本報告書の作成に当り数値実験グループの渡辺正氏には御指導と御助言をいただきました．さらに，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竹田辰興氏，情報システム管理課長藤井実氏，(株)富士通R&Dシステム部長平沢健一氏に感謝致します．

付録 A 分割した配列の一覧

A.1 コモン配列

本コードでは，コモンブロック名は同じでも，それに属するコモン配列名がルーチンごとに異なっている場合がある．以下では，このようなコモンブロックについては，それぞれの場合について（その1）または（その2）として記述した．

コモンブロック AAAA

No.	配列名	グローバル名	要素数	分割方法
1	AAL	AAL_G	0:MJG	/ipmj
2	AALW	AALW_G	0:MJG	/ipmj

コモンブロック AVERAG

No.	配列名	グローバル名	要素数	分割方法
1	AU	AU_G	0:MJG	/ipmj
2	AV	AV_G	0:MJG	/ipmj10
3	AW	AW_G	0:MJG	/ipmj11

コモンブロック DATR （その１）

No.	配列名	グローバル名	要素数	分割方法
1	UUSS	UUSS_G	0:MJG	/ipmj
2	VVSS	VVSS_G	0:MJG	/ipmj
3	WWSS	WWSS_G	0:MJG	/ipmj
4	XXSS	XXSS_G	0:MJG	/ipmj
5	YYSS	YYSS_G	0:MJG	/ipmj
6	ZZSS	ZZSS_G	0:MJG	/ipmj
7	PPSS	PPSS_G	0:MJG	/ipmj
8	UVSS	UVSS_G	0:MJG	/ipmj
9	PU1S	PU1S_G	0:MJG	/ipmj
10	PV2S	PV2S_G	0:MJG	/ipmj
11	PVSS	PVSS_G	0:MJG	/ipmj
12	PW3S	PW3S_G	0:MJG	/ipmj
13	PV1S	PV1S_G	0:MJG	/ipmj
14	PU2S	PU2S_G	0:MJG	/ipmj
15	PUSS	PUSS_G	0:MJG	/ipmj
16	VU2S	VU2S_G	0:MJG	/ipmj
17	UKUK	UKUK_G	0:MJG	/ipmj
18	VKVK	VKVK_G	0:MJG	/ipmj
19	WKWK	WKWK_G	0:MJG	/ipmj
20	ODOD	ODOD_G	0:MJG	/ipmj
21	VKPK	VKPK_G	0:MJG	/ipmj
22	UKVK	UKVK_G	0:MJG	/ipmj
23	D1D2	D1D2_G	0:MJG	/ipmj
24	XYSS	XYSS_G	0:MJG	/ipmj
25	U3W2	U3W2_G	0:MJG	/ipmj
26	U1U2	U1U2_G	0:MJG	/ipmj
27	V2U2	V2U2_G	0:MJG	/ipmj

コモンブロック DATR （その2）

No.	配列名	グローバル名	要素数	分割方法
1	URMS	URMS_G	0:MJG	/ipmj
2	VRMS	VRMS_G	0:MJG	/ipmj
3	WRMS	WRMS_G	0:MJG	/ipmj
4	XRMS	XRMS_G	0:MJG	/ipmj
5	YRMS	YRMS_G	0:MJG	/ipmj
6	ZRMS	ZRMS_G	0:MJG	/ipmj
7	PRMS	PRMS_G	0:MJG	/ipmj
8	UVSS	UVSS_G	0:MJG	/ipmj
9	PU1S	PU1S_G	0:MJG	/ipmj
10	PV2S	PV2S_G	0:MJG	/ipmj
11	PVSS	PVSS_G	0:MJG	/ipmj
12	PW3S	PW3S_G	0:MJG	/ipmj
13	PV1S	PV1S_G	0:MJG	/ipmj
14	PU2S	PU2S_G	0:MJG	/ipmj
15	PUSS	PUSS_G	0:MJG	/ipmj
16	VU2S	VU2S_G	0:MJG	/ipmj
17	UKUK	UKUK_G	0:MJG	/ipmj
18	VKVK	VKVK_G	0:MJG	/ipmj
19	WKWK	WKWK_G	0:MJG	/ipmj
20	ODOD	ODOD_G	0:MJG	/ipmj
21	VKPK	VKPK_G	0:MJG	/ipmj
22	UKVK	UKVK_G	0:MJG	/ipmj
23	D1D2	D1D2_G	0:MJG	/ipmj
24	XYSS	XYSS_G	0:MJG	/ipmj
25	U3W2	U3W2_G	0:MJG	/ipmj
26	U1U2	U1U2_G	0:MJG	/ipmj
27	V2U2	V2U2_G	0:MJG	/ipmj

コモンブロック DATS （その１）

No.	配列名	グローバル名	要素数	分割方法
1	UUUS	UUUS_G	0:MJG	/ipmj
2	VVVS	VVVS_G	0:MJG	/ipmj
2	WWWS	WWWS_G	0:MJG	/ipmj
4	PPPS	PPPS_G	0:MJG	/ipmj
5	UUUU	UUUU_G	0:MJG	/ipmj
6	VVVV	VVVV_G	0:MJG	/ipmj
7	WWWW	WWWW_G	0:MJG	/ipmj
8	PPPVP	PPPVP_G	0:MJG	/ipmj
9	TD1	TD1_G	0:MJG	/ipmj
10	TD3	TD3_G	0:MJG	/ipmj
11	TDS	TDS_G	0:MJG	/ipmj
12	TDE	TDE_G	0:MJG	/ipmj
13	TPE	TPE_G	0:MJG	/ipmj
14	RRST	RRST_G	0:MJG	/ipmj
15	RRSTA	RRSTA_G	0:MJG	/ipmj
16	SRST	SRST_G	0:MJG	/ipmj
17	SRSTA	SRSTA_G	0:MJG	/ipmj
18	SRSTB	SRSTB_G	0:MJG	/ipmj
19	FRST	FRST_G	0:MJG	/ipmj
20	FRSTA	FRSTA_G	0:MJG	/ipmj
21	FRSTB	FRSTB_G	0:MJG	/ipmj
22	FRSTC	FRSTC_G	0:MJG	/ipmj

コモンブロック DATS （その２）

No.	配列名	グローバル名	要素数	分割方法
1	SU	SU_G	0:MJG	/ipmj
2	SV	SV_G	0:MJG	/ipmj
3	SW	SW_G	0:MJG	/ipmj
4	SP	SP_G	0:MJG	/ipmj
5	FU	FU_G	0:MJG	/ipmj
6	FV	FV_G	0:MJG	/ipmj
7	FW	FW_G	0:MJG	/ipmj
8	FP	FP_G	0:MJG	/ipmj
9	TD1	TD1_G	0:MJG	/ipmj
10	TD3	TD3_G	0:MJG	/ipmj
11	TDS	TDS_G	0:MJG	/ipmj
12	TDE	TDE_G	0:MJG	/ipmj
13	TPE	TPE_G	0:MJG	/ipmj
14	RRST	RRST_G	0:MJG	/ipmj
15	RRSTA	RRSTA_G	0:MJG	/ipmj
16	SRST	SRST_G	0:MJG	/ipmj
17	SRSTA	SRSTA_G	0:MJG	/ipmj
18	SRSTB	SRSTB_G	0:MJG	/ipmj
19	FRST	FRST_G	0:MJG	/ipmj
20	FRSTA	FRSTA_G	0:MJG	/ipmj
21	FRSTB	FRSTB_G	0:MJG	/ipmj
22	FRSTC	FRSTC_G	0:MJG	/ipmj

コモンブロック DERMK

No.	配列名	グローバル名	要素数	分割方法
1	PRODK	PRODK_G	0:MJG	/ipmj
2	PDFFK	PDFFK_G	0:MJG	/ipmj
3	DSSPK	DSSPK_G	0:MJG	/ipmj
4	TDFFK	TDFFK_G	0:MJG	/ipmj
5	VDFFK	VDFFK_G	0:MJG	/ipmj
6	RSDRK	RSDRK_G	0:MJG	/ipmj

コモンブロック DERMR

No.	配列名	グローバル名	要素数	分割方法
1	PRDR1	PRDR1_G	0:MJG	/ipmj
2	PSTR1	PSTR1_G	0:MJG	/ipmj
3	DSPR1	DSPR1_G	0:MJG	/ipmj
4	TDFR1	TDFR1_G	0:MJG	/ipmj
5	VDFR1	VDFR1_G	0:MJG	/ipmj
6	PSTR2	PSTR2_G	0:MJG	/ipmj
7	DSPR2	DSPR2_G	0:MJG	/ipmj
8	TDFR2	TDFR2_G	0:MJG	/ipmj
9	PDFR2	PDFR2_G	0:MJG	/ipmj
10	VDFR2	VDFR2_G	0:MJG	/ipmj
11	PSTR3	PSTR3_G	0:MJG	/ipmj
12	DSPR3	DSPR3_G	0:MJG	/ipmj
13	TDFR3	TDFR3_G	0:MJG	/ipmj
14	VDFR3	VDFR3_G	0:MJG	/ipmj
15	PRDRS	PRDRS_G	0:MJG	/ipmj
16	PSTRS1	PSTRS1_G	0:MJG	/ipmj
17	PSTRS2	PSTRS2_G	0:MJG	/ipmj
18	DSPRS	DSPRS_G	0:MJG	/ipmj
19	TDFRS	TDFRS_G	0:MJG	/ipmj
20	PDFRS	PDFRS_G	0:MJG	/ipmj
21	VDFRS	VDFRS_G	0:MJG	/ipmj
22	RSDR1	RSDR1_G	0:MJG	/ipmj
23	RSDR2	RSDR2_G	0:MJG	/ipmj
24	RSDR3	RSDR3_G	0:MJG	/ipmj
25	RSDRS	RSDRS_G	0:MJG	/ipmj

コモンブロック DEW0

No.	配列名	グローバル名	要素数	分割方法
1	V	V_G	-3:MIG,-3:MJG,-3:MKG,3	:,/ipmj21,:,:
2	VP	VP_G	0:MIG,0:MJG,0:MKG	:,/ipmj11,:

コモンブロック DEW1

No.	配列名	グローバル名	要素数	分割方法
1	VT	VT_G	-1:MIG,-1:MJG,-1:MKG,3	:,/ipmj10,:,:

コモンブロック DFT

No.	配列名	グローバル名	要素数	分割方法
1	QD	QD_G	-1:MIG,-1:MJG,-1:MKG	:,/ipmj,:

コモンブロック DFTS2

No.	配列名	グローバル名	要素数	分割方法
1	DP2	DP2_G	0:MIG,0:MJG,0:MKG	:,/ipmj,:
2	DQ2	DQ2_G	0:MIG,0:MJG,0:MKG	:,/ipmj,:

コモンブロック DFTSP

No.	配列名	グローバル名	要素数	分割方法
1	DVP	DVP_G	0:MIG,0:MJG,0:MKG	:,/ipmj,:
2	DQD	DQD_G	0:MIG,0:MJG,0:MKG	:,/ipmj,:

コモンブロック DHEAR

No.	配列名	グローバル名	要素数	分割方法
1	REST	REST_G	0:MJG	/ipmj11
2	TSH	TSH_G	0:MJG	/ipmj
3	SRR	SRR_G	0:MJG	/ipmj
4	STIME	STIME_G	0:MJG	/ipmj
5	SLENG	SLENG_G	0:MJG	/ipmj

コモンブロック DHEI

No.	配列名	グローバル名	要素数	分割方法
1	DUHEI	DUHEI_G	0:MJG	/ipmj11
2	DVHEI	DVHEI_G	0:MJG	/ipmj
3	DWHEI	DWHEI_G	0:MJG	/ipmj
4	DPHEI	DPHEI_G	0:MJG	/ipmj11
5	DGRU	DGRU_G	0:MJG	/ipmj

コモンブロック DHET

No.	配列名	グローバル名	要素数	分割方法
1	DTHEI	DTHEI_G	0:MJG	/ipmj

コモンブロック DOLD

No.	配列名	グローバル名	要素数	分割方法
1	DFSI	DFSI_G	0:MIG,0:MJG,0:MKG	/ipmj
2	DFR	DFR_G	0:MIG,0:MJG,0:MKG	/ipmj
3	DFZ	DFZ_G	0:MIG,0:MJG,0:MKG	/ipmj

コモンブロック DMSTV

No.	配列名	グローバル名	要素数	分割方法
1	DMV	DMV_G	0:MJG,KNKR	/ipmj,:

コモンブロック DNISO

No.	配列名	グローバル名	要素数	分割方法
1	B11	B11_G	0:MJG	/ipmj
2	B22	B22_G	0:MJG	/ipmj
3	B33	B33_G	0:MJG	/ipmj
4	B12	B12_G	0:MJG	/ipmj
5	V11	V11_G	0:MJG	/ipmj
6	V22	V22_G	0:MJG	/ipmj
7	V33	V33_G	0:MJG	/ipmj
8	V12	V12_G	0:MJG	/ipmj
9	D11	D11_G	0:MJG	/ipmj
10	D22	D22_G	0:MJG	/ipmj
11	D33	D33_G	0:MJG	/ipmj
12	D12	D12_G	0:MJG	/ipmj

コモンブロック DORK

No.	配列名	グローバル名	要素数	分割方法
1	DUMM	DUMM_G	10,0:MJG	:,/ipmj
2	BAKA	BAKA_G	10,0:MJG	:,/ipmj
3	AHO	AHO_G	0:MJG	/ipmj

コモンブロック DTTCS （その1）

No.	配列名	グローバル名	要素数	分割方法
1	UMEAN	UMEAN_G	0:MJG	/ipmj
2	WMEAN	WMEAN_G	0:MJG	/ipmj
3	OXMEAN	OXMEAN_G	0:MJG	/ipmj
4	GRU	GRU_G	0:MJG	/ipmj
5	PMEAN	PMEAN_G	0:MJG	/ipmj
6	TMEAN	TMEAN_G	0:MJG	/ipmj

コモンブロック DTTCS （その2）

No.	配列名	グローバル名	要素数	分割方法
1	UM	UM_G	0:MJG	/ipmj
2	WMEAN	WMEAN_G	0:MJG	/ipmj
3	OXMEAN	OXMEAN_G	0:MJG	/ipmj
4	GRU	GRU_G	0:MJG	/ipmj
5	PMEAN	PMEAN_G	0:MJG	/ipmj
6	TMEAN	TMEAN_G	0:MJG	/ipmj

コモンブロック DUV

No.	配列名	グローバル名	要素数	分割方法
1	UV	UV_G	0:MJG	/ipmj
2	UVUV	UVUV_G	0:MJG	/ipmj

コモンブロック DVW

No.	配列名	グローバル名	要素数	分割方法
1	VW	VW_G	0:MJG	/ipmj
2	VWVW	VWVW_G	0:MJG	/ipmj

コモンブロック FFAD

No.	配列名	グローバル名	要素数	分割方法
1	FAVP	FAVP_G	-1:MIG,-1:MJG,-1:MKG	:,/ipmj01,:

コモンブロック GBD1

No.	配列名	グローバル名	要素数	分割方法
1	GDF11	GDF11_G	0:MJG	/ipmj
2	GTU11	GTU11_G	0:MJG	/ipmj
3	GVPG11	GVPG11_G	0:MJG	/ipmj
4	GE11	GE11_G	0:MJG	/ipmj

コモンブロック GBD2

No.	配列名	グローバル名	要素数	分割方法
1	GDF22	GDF22_G	0:MJG	/ipmj
2	GTU22	GTU22_G	0:MJG	/ipmj
3	GVPG22	GVPG22_G	0:MJG	/ipmj
4	GE22	GE22_G	0:MJG	/ipmj

コモンブロック GBD3

No.	配列名	グローバル名	要素数	分割方法
1	GDF33	GDF33_G	0:MJG	/ipmj
2	GTU33	GTU33_G	0:MJG	/ipmj
3	GVPG33	GVPG33_G	0:MJG	/ipmj
4	GE33	GE33_G	0:MJG	/ipmj

コモンブロック GBD4

No.	配列名	グローバル名	要素数	分割方法
1	GDF12	GDF12_G	0:MJG	/ipmj
2	GTU12	GTU12_G	0:MJG	/ipmj
3	GVPG12	GVPG12_G	0:MJG	/ipmj
4	GE12	GE12_G	0:MJG	/ipmj

コモンブロック GBD5

No.	配列名	グローバル名	要素数	分割方法
1	GDF13	GDF13_G	0:MJG	/ipmj
2	GTU13	GTU13_G	0:MJG	/ipmj
3	GVPG13	GVPG13_G	0:MJG	/ipmj
4	GE13	GE13_G	0:MJG	/ipmj

コモンブロック GBD6

No.	配列名	グローバル名	要素数	分割方法
1	GDF23	GDF23_G	0:MJG	/ipmj
2	GTU23	GTU23_G	0:MJG	/ipmj
3	GVPG23	GVPG23_G	0:MJG	/ipmj
4	GE23	GE23_G	0:MJG	/ipmj

コモンブロック GBDK

No.	配列名	グローバル名	要素数	分割方法
1	GDFKK	GDFKK_G	0:MJG	/ipmj
2	GTUKK	GTUKK_G	0:MJG	/ipmj
3	GVPGKK	GVPGKK_G	0:MJG	/ipmj
4	GEKK	GEKK_G	0:MJG	/ipmj

コモンブロック GCOND

No.	配列名	グローバル名	要素数	分割方法
1	GCTV	GCTV_G	0:MJG	/ipmj
2	GCTW	GCTW_G	0:MJG	/ipmj
3	GCT12	GCT12_G	0:MJG	/ipmj
4	GCT13	GCT13_G	0:MJG	/ipmj
5	GCT23	GCT23_G	0:MJG	/ipmj

コモンブロック GPD1

No.	配列名	グローバル名	要素数	分割方法
1	GPDF11	GPDF11_G	0:MJG	/ipmj
2	GPST11	GPST11_G	0:MJG	/ipmj

コモンブロック GPD2

No.	配列名	グローバル名	要素数	分割方法
1	GPDF22	GPDF22_G	0:MJG	/ipmj
2	GPST22	GPST22_G	0:MJG	/ipmj

コモンブロック GPD3

No.	配列名	グローバル名	要素数	分割方法
1	GPDF33	GPDF33_G	0:MJG	/ipmj
2	GPST33	GPST33_G	0:MJG	/ipmj

コモンブロック GPD4

No.	配列名	グローバル名	要素数	分割方法
1	GPDF12	GPDF12_G	0:MJG	/ipmj
2	GPS121	GPS121_G	0:MJG	/ipmj
3	GPS122	GPS122_G	0:MJG	/ipmj
4	GPST12	GPST12_G	0:MJG	/ipmj

コモンブロック GPD5

No.	配列名	グローバル名	要素数	分割方法
1	GPDF13	GPDF13_G	0:MJG	/ipmj
2	GPS131	GPS131_G	0:MJG	/ipmj
3	GPS133	GPS133_G	0:MJG	/ipmj
4	GPST13	GPST13_G	0:MJG	/ipmj

コモンブロック GPD6

No.	配列名	グローバル名	要素数	分割方法
1	GPDF23	GPDF23_G	0:MJG	/ipmj
2	GPS232	GPS232_G	0:MJG	/ipmj
3	GPS233	GPS233_G	0:MJG	/ipmj
4	GPST23	GPST23_G	0:MJG	/ipmj

コモンブロック GPDKK

No.	配列名	グローバル名	要素数	分割方法
1	GPDFKK	GPDFKK_G	0:MJG	/ipmj
2	GPSTKK	GPSTKK_G	0:MJG	/ipmj

コモンブロック GPRD

No.	配列名	グローバル名	要素数	分割方法
1	GPROKK	GPROKK_G	0:MJG	/ipmj
2	GPROU	GPROU_G	0:MJG	/ipmj
3	GPROV	GPROV_G	0:MJG	/ipmj
4	GPRO12	GPRO12_G	0:MJG	/ipmj

コモンブロック GPRD2

No.	配列名	グローバル名	要素数	分割方法
1	GPROW	GPROW_G	0:MJG	/ipmj
2	GPRO13	GPRO13_G	0:MJG	/ipmj
3	GPRO23	GPRO23_G	0:MJG	/ipmj

コモンブロック MMNTD

No.	配列名	グローバル名	要素数	分割方法
1	URMS	URMS_G	0:MJG	/ipmj11
2	VRMS	VRMS_G	0:MJG	/ipmj11
3	WRMS	WRMS_G	0:MJG	/ipmj11
4	PRMS	PRMS_G	0:MJG	/ipmj
5	USKW	USKW_G	0:MJG	/ipmj
6	VSKW	VSKW_G	0:MJG	/ipmj
7	WSKW	WSKW_G	0:MJG	/ipmj
8	PSKW	PSKW_G	0:MJG	/ipmj
9	UFLT	UFLT_G	0:MJG	/ipmj
10	VFLT	VFLT_G	0:MJG	/ipmj
11	WFLT	WFLT_G	0:MJG	/ipmj
12	PFLT	PFLT_G	0:MJG	/ipmj
13	XRMS	XRMS_G	0:MJG	/ipmj
14	YRMS	YRMS_G	0:MJG	/ipmj
15	ZRMS	ZRMS_G	0:MJG	/ipmj
16	AR	AR_G	0:MJG	/ipmj
17	OA	OA_G	0:MJG	/ipmj
18	RUV	RUV_G	0:MJG	/ipmj

コモンブロック TAU

No.	配列名	グローバル名	要素数	分割方法
1	TAU	TAU_G	0:MJG	/ipmj
2	TAV	TAV_G	0:MJG	/ipmj
3	TAW	TAW_G	0:MJG	/ipmj

A.2 ローカル配列

サブルーチン AVEAD

No.	配列名	グローバル名	要素数	分割方法
1	USGSA	USGSA_G	-1:MJG	/ipmj

サブルーチン DEV1

No.	配列名	グローバル名	要素数	分割方法
1	XX	---	10,0:MJG	:,/ipmj
2	X	X_G	10,0:NJG	/ipm10,:
3	Y	Y_G	10,0:NJG	/ipm10,:

```
(!xocl index partition ipm10=(spe,index=1:10,part=band))
```

サブルーチン DPE4R

No.	配列名	グローバル名	要素数	分割方法
1	WKDQD	WKDQD_G	0:MIG,0:MJG,0:MKG	:,:,/ipmk
2	WKDQ2	WKDQ2_G	0:MIG,0:MJG,0:MKG	:,:,/ipmk
3	WKDP2	WKDP2_G	0:MIG,0:MJG,0:MKG	:,:,/ipmk
4	WKDVP	WKDVP_G	0:MIG,0:MJG,0:MKG	:,:,/ipmk

```
(!xocl index partition ipmk=(spe,index=0:MKG,part=band))
```

サブルーチン PSAT23C

No.	配列名	グローバル名	要素数	分割方法
1	EUHEI	EUHEI_G	-2:MJG	/ipmj11
2	EWHEI	EWHEI_G	-2:MJG	/ipmj11

サブルーチン PTOUC

No.	配列名	グローバル名	要素数	分割方法
1	YRR	YRR_G	0:MJG	/ipmj
2	YR	YR_G	0:MJG	/ipmj
3	RESU	RESU_G	0:MJG	/ipmj
4	RESV	RESV_G	0:MJG	/ipmj
5	RESW	RESW_G	0:MJG	/ipmj
6	RESUV	RESUV_G	0:MJG	/ipmj
7	RESK	RESK_G	0:MJG	/ipmj
8	RESVW	RESVW_G	0:MJG	/ipmj
9	RESUW	RESUW_G	0:MJG	/ipmj
10	CHDIF1	CHDIF1_G	0:MJG	/ipmj
11	CHDIF2	CHDIF2_G	0:MJG	/ipmj
12	CHDIF3	CHDIF3_G	0:MJG	/ipmj
13	CHDIF4	CHDIF4_G	0:MJG	/ipmj
14	CHDIFK	CHDIFK_G	0:MJG	/ipmj
15	CHDIF12	CHDIF12_G	0:MJG	/ipmj
16	CHDIFP	CHDIFP_G	0:MJG	/ipmj

付録 B 処理の流れと袖転送の挿入箇所

主なサブルーチンの呼び出し，袖付き分割された配列への値の代入，参照，及び袖転送を挿入した箇所を以下に示す．なお，配列の参照については，袖部分が参照される場合についてのみ記してある．

```
MAIN
    CALL SEDDRI
        READ VP, V
    CALL PREFFT
    CALL PREFFT
    CALL CINTP
        GPDFKK の定義
    CALL WARINP
        URMS, VRMS, WRMS の定義
    CALL PWRINI
    CALL DCDR
    CALL DUBC
        V の定義
    VP の定義
    CALL DUBC
        V の定義
    CALL D3F2AR
        V の袖転送
        V の袖部分の参照
        CALL DPEW4S
            VT の定義
        V の袖部分の参照
        CALL DPEV4S
            VT の定義
        V の袖部分の参照
        CALL DPEU4S
            VT の定義
    CALL DMAR4R
        VT の定義
        VT の袖転送
        VT の袖部分の参照
        CALL DPE4R
            FAVP の定義
        FAVP の定義
        FAVP の袖転送
        V の定義 ← FAVP の袖部分の参照
        VP の定義
    CALL PSAT23
        DUHEI, DPHEI の定義
        DUHEI の袖転送
        DUHEI の袖部分の参照
        CALL DOME
            V の袖転送
            VT の定義 ← V の袖部分の参照
        V, VP, VT の袖転送
        V の袖部分の参照
        CALL FFTSPH
        CALL SATPWH
        V, VP, VT の袖部分の参照
        URMS, VRMS, WRMS の定義
(次ページに続く)
```

```
CALL PSAT23C
    EUHEI, EWHEI, DPHEI の定義
    V, VP, EUHEI, EWHEI, DPHEI の袖転送
    V, VP, EUHEI, EWHEI, DPHEI の袖部分の参照
    GPDFKK の定義
CALL AVEAD
    V の袖転送
    AV, AW の定義
    AV の袖転送
    AV, V の袖部分の参照
    AW の定義
    AW の袖転送
    AW の袖部分の参照
CALL WARP
    URMS, VRMS, WRMS の定義
CALL WARSPE
CALL CWARIP
    GPDFKK の定義
CALL TOUP
    CALL PCLST
        URMS, VRMS, WRMS, REST の定義
CALL PTOUC
    GPDFKK, URMS, VRMS, WRMS, REST の定義
    GPDFKK, URMS, VRMS, WRMS, REST の袖転送
    GPDFKK, URMS, VRMS, WRMS, REST の袖部分の参照
    GPDFKK の定義
CALL TOUSPE
CALL SEDDRO
```

付録 C 並列化指示行

・!xocl processor p(npe)

p はプロセッサグループ名，括弧中の 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れる（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た．

・!xocl index partition p1=(p,index=1:ipp,part=band)

これは，分割方法を定義する場合に使用し，ここでの p1 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 で ipp=40000 の場合には，p(1):1 ～ 10000，p(2):10001 ～ 20000，p(3): 20001 ～ 30000，p(4):30001 ～ 40000 という分割方法 p1 が定義される．

・!xocl subprocessor subp(npe)=p(1:npe)

これは，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(/p1)

配列 x のローカル分割宣言である．例えば，x(40000) という配列が宣言されていたとすると，このローカル分割宣言により，分割方法 p1 によって分割され，p(1):x(1) ～ x(10000)，p(2):x(10001) ～ x(20000)，p(3):x(20001) ～ x(30000)，p(4):x(30001) ～ x(40000) のように各 PE にローカルデータとして配分される．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

配列 x のグローバル宣言である．グローバル宣言された配列は，全 PE からの定義・引用が可能である．

・!xocl parallel region

この行により，並列実行が開始される（各 PE で冗長実行）．この行の直前までは通常の逐次実行であり，直後は並列実行になる．この行の直後のすべての PE の状態は逐次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，並列実行から逐次実行になる．ただし，この行以降の逐次実行では並列実行時のどの PE の状態が引き継がれるかは予測不可能であ

る．また，並列実行終了直前には，すべての PE において同期がとられる．

・!xocl spread do

これは，DO ループの分割を指定する場合に使用する．この場合は，DO ループの回転数が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．ただし，!xocl spread do /p1 のように指定した場合，/ 以降の分割方法によって DO ループが分割される．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 spread region

!xocl spread do が DO ループの分割を行うのに対して，これは，DO ループが無い場合の処理分割を指定する場合に使用する．分割の仕方は，!xocl spread do と同様で !xocl spread region /p1 という / を付けた指定も可能である．尚，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

!xocl spread do や !xocl spread region に対して指定するものであり，DO ループ分割や処理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(aspc)

!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．sum(aspc) は，DO ループ分割の終了と同時に，各 PE で求められた変数 aspc または配列 aspc について全 PE 間で総和を行わせる指定である．この後，総和された aspc 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．

付録 D リージョンと並列ジョブの I/O

・リージョン

parallel region 文と **end parallel** 文の間で実行される部分や，**spread do(or region)** 文と **end spread** 文の間で実行される部分をリージョンと呼ぶ．リージョンは入れ子構造が可能で，親リージョンと子リージョンが定義される．**spread do(or region)** 文は必ず**parallel region** 文と **end parallel** 文の間で実行される部分で指定しなければならないことから，**parallel region** 文と **end parallel** 文の間で実行される部分はすべてのリージョンの先祖になる．このリージョンを特にパラレルリージョンと呼ぶ．ここで，例えば，**spread do (or region)** 文により，各 PE に処理が分割された場合，各 PE では，子リージョンが生成され，これらを兄弟リージョンと呼ぶ．これら兄弟リージョンは独立であるため，実行順序は予測不可能である．リージョンと PE の間には，1 対 1，N 対 1，1 対 N の関係があるが，パラレルリージョンは，1 対 N の関係になる．この場合全 PE が同一のプログラム部分を実行するため，この部分を冗長実行部と呼ぶ．また，今回の並列化では，パラレルリージョン以外のリージョンはすべて 1 対 1 の関係にしている．

・並列ジョブの I/O

並列ジョブはほとんどの場合，逐次実行部分と並列実行部分から成る．逐次実行部分で I/O を行う分には何ら問題はないが，並列実行部分で I/O を行う場合は並列ジョブ特有の I/O になり，制御を行なった I/O 処理が必要になる場合がある．

VPP 上の並列ジョブでは，どの実行部分でファイルをオープンするかで，実行中のファイルに対する I/O 制御が決まる．標準出力，標準入力，標準エラー出力，リージョン以外でオープンしたファイルは共有ファイルと呼ばれ，リージョンでオープンしたファイルは専有ファイルと呼ばれる．更に専有ファイルにはオープンしたリージョン以外のリージョンでの I/O に制限がある．

(1) 共有ファイル

a. 共有ファイルに対し，逐次実行部分から出力を行う場合

逐次実行ジョブと同一である．制御をする必要ない．

b. 共有ファイルに対し，パラレルリージョンから出力を行う場合

パラレルリージョン直後は，すべての PE で同一の資源を持つため，共有ファイルへのファイルポインタも各 PE で持つ．しかし，VPP FORTRAN 並列コンパイラの機能により，パラレルリージョン内では入出力文のみ冗長に実行されないため，各 PE からの出力による上書きや，出力を行う PE の順番違いによる順不同書きは発生しない．この場合はある PE 上の出力だけが行われる．

c. 共有ファイルに対し，各 PE 上のリージョンから出力を行う場合

VPP FORTRAN 並列コンパイラの機能により各 PE からの出力による上書きは発生しない

が，順不同書きは発生する．上書きが発生しないのは，VPP FORTRAN 並列コンパイラが，並列に実行される複数の入出力文が存在しても，共有ファイルに対しては入出力の結果が入出力文を単位として実行されたものであることを保証しているからである．順不同書きが発生するのは，各 PE 上のリージョンが独立であるため順番を特定できないからである．従って，正常な順番で書き出すには，ユーザによる制御が必要になる．

(2) 専有ファイル

a. パラレルリージョンでオープンした専有ファイルに対し，パラレルリージョンから出力を行う場合

これは，(1) の b. と同様である．

b. パラレルリージョンでオープンした専有ファイルに対し，各 PE 上のリージョンから出力を行う場合

オープンを行ったリージョンの子リージョンでは，そのファイルへの I/O は許されない．実行エラーになる．

c. ある PE 上のリージョンでオープンした専有ファイルに対し，他の PE 上のリージョン（兄弟関係になる）から出力を行う場合

このような兄弟リージョン間では，同一の機番に異なるファイルを結合させることは可能で出力も可能である．しかし，同一の機番に同一のファイルを結合させた場合には，順不同書きと上書きが発生し，最終的にファイル内容は，ある PE 上の出力のみ有効になる．従ってこの場合，各 PE 上の出力をそれぞれ有効にするには，PE 毎に別々のファイルに出力させる等のユーザによる制御が必要になる．

d. ある PE 上のリージョンでオープンした専有ファイルに対し，パラレルリージョンから出力を行う場合

このように子リージョンでオープンしたファイルは，そのリージョンが終了する時点でクローズされるため，パラレルリージョンで同一ファイルに出力した場合には，前の内容は無くなってしまう．

国際単位系 (SI) と換算表

表 1　SI 基本単位および補助単位

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

表 2　SI と併用される単位

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

1 eV = 1.60218 × 10^{-19} J

1 u = 1.66054 × 10^{-27} kg

表 5　SI 接頭語

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

表 3　固有の名称をもつ SI 組立単位

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

表 4　SI と共に暫定的に維持される単位

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

1 Å = 0.1 nm = 10^{-10} m

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

1 bar = 0.1 MPa = 10^5 Pa

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

1 Ci = 3.7 × 10^{10} Bq

1 R = 2.58 × 10^{-4} C/kg

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

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

（注）

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

換算表

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

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

動粘度　1 m^2/s = 10^4 St(ストークス)(cm^2/s)

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

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

1 cal = 4.18605 J（計量法）
= 4.184 J　（熱化学）
= 4.1855 J（15 °C）
= 4.1868 J（国際蒸気表）

仕事率　1 PS（仏馬力）
= 75 kgf·m/s
= 735.499 W

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

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58 × 10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

（86 年 12 月 26 日現在）

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9年度作業報告書—