JAERI-Data/Code
2000-018

原子力コードの高速化(ベクトル/並列化編)
平成10年度作業報告書

2000年3月

石附　茂*・小笠原忍・川井　渉*・根本俊行*
久米悦雄・足立将晶・川崎信夫・筒竹陽一*

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

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

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

編集兼発行　日本原子力研究所

JAERI－Data/Code 2000－018

原子力コードの高速化 (ベクトル／並列化編)
－ 平成10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
石附 茂＊・小笠原 忍※・川井 渉＊・根本 俊行＊
久米 悦雄・足立 将晶※・川崎 信夫※・箭竹 陽一＊＊

(2000年2月1日受理)

本報告書は，平成10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高速化作業のうち，VPP500（一部AP3000含む）におけるベクトル化／並列化作業部分について記述したものである．原子力コードの高速化作業は，平成10年度に12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「ベクトル／並列化編」，「スカラ並列化編」及び「移植編」の3分冊にまとめた．

本報告書の「ベクトル／並列化編」では，汎用トカマク回路シミュレーションプログラムGTCSPを対象に実施したベクトル化作業について，イオン性融体分子動力学計算コードMSP2，渦電流解析コードEDDYCAL，受動的冷却システム試験解析コードTHANPACST2及びMHD平衡コードSELENEJを対象に実施したベクトル並列化作業について記述している．

別冊の「スカラ並列化編」では，連続エネルギー粒子輸送モンテカルロコードMCNP4B2，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVP及び光量子による固体溶融蒸発シミュレーションコードPHCIPを対象に実施したParagon向けの並列化作業について記述している．また，別冊の「移植編」では，連続エネルギー粒子輸送モンテカルロコードMCNP4B2及び軽水炉安全性解析コードRELAP5(RELAP5/MOD2/C36-05, RELAP5/MOD3.2.1.2)のAP3000への移植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
＊＊ 株式会社日立製作所

JAERI－Data/Code 2000－018

Vectorization, Parallelization and Porting of Nuclear Codes (Vectorization and Parallelization) - Progress Report Fiscal 1998 -

Shigeru ISHIZUKI*, Shinobu OGASAWARA※, Wataru KAWAI*,
Toshiyuki NEMOTO*, Etsuo KUME, Masaaki ADACHI※,
Nobuo KAWASAKI※ and Yo-ichi YATAKE**

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

(Received February 1 , 2000)

everal computer codes in the nuclear field have been vectorized, parallelized and transported on the FUJITSU VPP500 system, the AP3000 system and the Paragon system at Center for Promotion of Computational Science and Engineering in Japan Atomic Energy Research Institute. We dealt with 12 codes in fiscal 1998. These results are reported in 3 parts, i.e., the vectorization and parallelization on vector processors part, the parallelization on scalar processors part and the porting part. In this report, we describe the vectorization and parallelization on vector processors.

In this vectorization and parallelization on vector processors part, the vectorization of General Tokamak Circuit Simulation Program code GTCSP, the vectorization and parallelization of Molecular Dynamics NTV (n-particle, Temperature and Velocity) Simulation code MSP2, Eddy Current Analysis code EDDYCAL, Thermal Analysis Code for Test of Passive Cooling System by HENDEL T2 code THANPACST2 and MHD Equilibrium code SELENEJ on the VPP500 are described.

In the parallelization on scalar processors part, the parallelization of Monte Carlo N-Particle Transport code MCNP4B2, Plasma Hydrodynamics code using Cubic Interpolated Propagation Method PHCIP and Vectorized Monte Carlo code (continuous energy model / multi-group model) MVP/GMVP on the Paragon are described. In the porting part, the porting of Monte Carlo N-Particle Transport code MCNP4B2 and Reactor Safety Analysis code RELAP5 on the AP3000 are described.

Keywords : GTCSP, MSP2, EDDYCAL, THANPACST2,SELENEJ, Parallelization, Vectorization, VPP500, AP3000, Nuclear Codes

---

※ On leave from FUJITSU, Ltd
* FUJITSU,Ltd
** HITACHI,Ltd

目　　次

Contents

1. はじめに

計算科学技術推進センター情報システム管理課では，原研が保有する各種のスーパーコンピュータの効率的な運用とコンピュータ資源の有効利用を促進するため，計算需要の多い原子力コードをユーザに代わってスーパーコンピュータ上に整備し，それぞれのコードに最適な高速化を施す作業を実施している．この作業は，コンピュータの効率的利用を推進するのみならず，ユーザの計算待ち時間の短縮を通じてユーザの仕事の効率化へも貢献するものと思われる．

原子力コードの高速化作業は，平成10年度に12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，「ベクトル／並列化編」，「スカラ並列化編」及び「移植編」の3分冊にまとめた．なお、例年分冊としていた「ベクトル化編」と「並列化編」については，近年，ベクトル化のみでチューニングを終えるコードが減少し，更なる高速化のため，ベクトル化のみならず並列化をも施すチューニングが増大しており，今年度からこれら2つを会わせて1分冊とすることにした．また，本年度から新たにParagonでの高速化作業成果を加え，編成を前述のように見直している．

本報告書の「ベクトル／並列化編」では，汎用トカマク回路シミュレーションプログラムGTCSPを対象に実施したVPP500向けベクトル化作業について，イオン性融体分子動力学計算コードMSP2，渦電流解析コードEDDYCAL，受動的冷却システム試験解析コードTHANPACST2及びMHD平衡コードSELENEJを対象に実施したVPP500向けベクトル並列化作業について記述している．別冊の「スカラ並列化編」では，連続エネルギー粒子輸送モンテカルロコードMCNP4B2，連続エネルギー及び多群モデルモンテカルロコードMVP/GMVP及び光量子による固体溶融蒸発シミュレーションコードPHCIPを対象に実施したParagon向けの並列化作業について記述している．また，別冊の「移植編」では，連続エネルギー粒子輸送モンテカルロコードMCNP4B2及び軽水炉安全性解析コードRELAP5 (RELAP5/MOD2/C36-05, RELAP5/MOD3.2.1.2)のAP3000への移植作業について記述している．なお，平成10年度に実施した高速化作業のうち，ここで取り上げなかったいくつかのコードに関しては，ユーザとの連名により別途JAERI-Data/Codeを執筆する予定であるので，そちらを参照されたい．

2章では，汎用トカマク回路シミュレーションプログラムGTCSPを対象に実施したVPP500向けのベクトル化作業について述べる．本作業は計算時間の短縮が目的であり、計算コストの集中するサブルーチンを中心にベクトル化チューニングを施すと同時に，不連続データアクセスやメモリバンク競合の改善を行うことで，実行効率の向上を図っている．この結果，オリジナル版のベクトル実行時に比較して約12.6倍の速度向上が得られた．

3章では，イオン性融体分子動力学計算コードMSP2を対象としたVPP500における並列化作業について述べる．本作業では，大規模計算への対応を目的とした，メモリ拡張のための並列化を行っている．本コードの並列化は、粒子数をプロセッサ台数分に分割して，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粒子分割の手法を適用した．この結果，4PEを使用した並列実行時でメモリ使用量を従来の約20分の1に，実行時間で約1.8倍の高速化がなされた．

4章では，渦電流解析コードEDDYCALを対象としたVPP500におけるベクトル並列化及びAP3000への移植作業について述べる．本コードは既にクレイ計算機向けにベクトル化されており，ベクトル化に関しては残された部分を中心に補足作業として実施した．並列化に関しては，メモリ使用量削減を目的に処理分割による並列化作業を実施した．また，AP3000への移植作業については，メモリ量の更なる拡大のために実施した作業である．この結果，オリジナル版スカラ実行時に比較して，VPP500における16PEのベクトル並列実行で約21倍の，AP3000における4PEの並列実行で約2.7倍の速度向上が得られた．

5章では，受動的冷却システム試験解析コードTHANPACST2を対象としたVPP500におけるベクトル並列化作業について述べる．本作業では，計算精度向上のためのメッシュ数増大に伴う計算時間の増加に対応するため、計算コストの高いソルバー部のベクトル化，及び流動計算部，温度計算部の並列化を実施した．この結果，オリジナル版スカラ実行時に比較して，VPP500における4PEのベクトル並列実行で約6.8倍の速度向上が得られた．

6章では，MHD平衡コードSELENEJを対象としたVPP500におけるベクトル並列化作業について述べる．本コードは実験データ解析に用いられており，実験中に短いサンプリング周期で出力される実験データを専用計算機で解析し，この解析結果を実験装置にフィードバックすることを検討している．このため，計算量はさほど多くないが，即座に値を返す必要があり，ベクトル並列化による高速化を施した．この結果，オリジナル版スカラ実行時に比較して，VPP500における4PEのベクトル並列実行で約11.5倍の速度向上が得られた．

なお，本報告書の2章の作業は小笠原が，3章及び5章の作業は石附が，4章の作業は川井が，6章の作業は根本が担当した．

2. GTCSP コードのインストール・ベクトル化

汎用トカマク回路シミュレーションプログラム GTCSP (General Tokamak Circuit Simulation Program) [1] (以下 GTCSP) の富士通製分散メモリ型ベクトル並列計算機 VPP500 [2] (以下 VPP) へのインストール，及び，ベクトル化を行った．GTCSP は，日本原子力研究所那珂研究所の解析サーバ SP2 [3](以下 SP2) 上で実行されているが，実行に長時間を要していた．そこで本作業において，VPP における高速化を行い，計算時間の短縮を図った．以下に，その作業について報告する．

2.1 コード概要

GTCSP は，日本原子力研究所の臨界プラズマ試験装置 JT-60 の電源システムの挙動解析を目的としているプログラムである．回路モデル（同期発電機，トランス・サイリスタ変換器，コイル群等）とそれらの接続ノード等の入力データより，グラフ理論に基づき回路方程式を構築し，それらを台形法で時間積分することで回路を解析していく．また，スイッチング素子として働くサイリスタのオン・オフ状態により，回路の構成が変化する場合，それらを検出し，回路方程式を再構築し，解析を続けるものである．

プログラムは，回路方程式の構築や微分方程式を解く中核 (メイン) プログラム `gtcsp.f` (以下 `gtcsp.f`)，制御系全般の模擬や初期状態の設定などをする制御ルーチン `diode.f` (以下 `diode.f`)，計算条件やパラメータの設定をする入力データ `data`，そして，可変配列データ `inc.f` (以下 inc.f) と各モデルプログラム (サブプログラム) から構成されている．但し，各モデルプログラムは，`gtcsp.f` に含まれている．

2.2 インストール作業

ここでは，SP2 上で実行されている GTCSP コードを VPP へインストールした際の作業について述べる．SP2 の FORTRAN は，XL FORTRAN for AIX [4, 5] を使用しており，これを VPP FORTRAN77/EX [6] の仕様に変換した．今回，インストールの対象となった解析モデルは vmodel 版であるが，この解析モデルは，実行時間が長大なため，はじめにデモ版 (demo1，demo2，demo3) のインストールを行なった．各解析モデル (vmodel，demo1，demo2，demo3) 間では，`gtcsp.f` 以外のソースは異なり，各解析モデルごとに用意されている．よって，vmodel 版のインストールを行った際に，`gtcsp.f` は，デモ版のインストール時のものを用い，その他のソースは，デモ版でのインストールの結果を参考に修正をした．

2.2.1 デモ版のインストール

ここでは，デモ版 (demo1，demo2，demo3) のインストール作業について述べる．

2.2.1.1　制御ルーチン (diode.f) の修正

diode.f の修正箇所について，以下に述べる．

(1) 宣言順の修正

demo1，demo3 のそれぞれの inc.f 内では，IMPLICIT 文が記述されている．demo1 の inc.f を Fig.2.1 に示す．また，diode.f において，Fig.2.2 のように，INCLUDE 文が，型宣言文 (INTEGER) より後に宣言されている．そのため，INCLUDE 文により，inc.f が INCLUDE 文が現れた行位置に組み込まれると，型宣言文が，IMPLICIT 文より前に宣言されてしまう．VPP FORTRAN77/EX では，IMPLICIT 文は，型宣言文より先に宣言されなければならないため，エラーが起きていた．よって，Fig.2.3 のように修正した．

(2) IMPLICIT 文による型宣言の修正

demo2，demo3 において，

```
fldpx:  diode.o:  warning:  symbol:
        'covrl_' has different size in file gtcsp.o
```

という WARNING が，リンク時に発生していた．covrl について調査した結果，covrl は，COMMOM 文で宣言される共通ブロック名で，vm，r，l という配列が宣言されていた．さらに調査した結果，diode.f では，IMPLICIT 文は，

```
implicit real*8(a-h,o-z)
```

と宣言されており，gtcsp.f では，

```
implicit real*8(a-h,l-z)
```

と宣言されていた．このような IMPLICIT 文の宣言の違いにより，配列 l の型が異なり，WARNING が発生していた．配列 l は，gtcsp.f だけで使用し，diode.f では，使用していなかった．diode.f の IMPLICIT 文 を，"(a-h,l-z)" としても，他の変数に影響がないことを確認し，修正した．

(3) 配列サイズの修正

demo2，demo3 において，配列 l のサイズは，inc.f で宣言されているが，diode.f ではパラメータ $cor (= 80) を，gtcsp.f ではパラメータ $col (= 6400) を指定していたためサイズが異なり，WARNING が発生していた．配列 l は，diode.f 内では，使われていなかったので，diode.f で宣言されている配列 l のサイズを，gtcsp.f での宣言と同じにした．

2.2.1.2　中核プログラム (gtcsp.f) の修正

gtcsp.f の修正箇所について，以下に述べる．

(1) 不変式先行評価の最適化抑止

gtcsp.f の サブルーチン K3THV1 内の DO 10 ループにおいて， 不変式の先行評価の最適化が行なわれ，0 除算が発生し，実行が異常終了した．この最適化は，ループ内の不変式（値の変わらない式）をループの外に移動するというものである．調査した結果，ここでは，Fig.2.4 の (b) における，"2.0d+0 / deltat" が不変式である．また (b) の演算は，Fig.2.4の (a) より

```
if(deltat.eq.0.0d+0) go to 136 !(k7)
```

とあり， deltat が 0 でないとき演算を行なうことになっている．ところが，最適化により，その不変式がループの外に移動され，deltat の値に関係なく実行される．そのため，deltat の値が 0 のときも実行され，プログラムが異常終了した．そこで，ここでの最適化を最適化制御行 "*vocl loop,nopreex" を用いて抑止した．修正箇所を Fig.2.4 の修正後に示す．

(2) 演算評価方法変更の最適化抑止

実行結果は，コード内で動的に決定される時間ステップの刻み毎に出力されるが，demo1 において， SP2 上での実行結果と VPP 上での実行結果を比較した結果，その時間を示す変数 time に違いが生じており，その影響で，他の変数の値にも違いが生じていた．調査した結果，演算評価方法の変更の最適化が行なわれたことが原因で変数 time に誤差が生じていた．この最適化は，サブルーチン MINVD の DO 50 ループの不変項 pivot による除算の箇所において行なわれており，ループの外でその不変式の逆数を求め，ループ内の除算はその逆数の乗算に変更される．この最適化を最適化制御行 "*vocl loop,noeval" を用いて抑止した．修正箇所を Fig.2.5 に示す．

ここでの演算評価方法の変更の最適化による誤差について，ある時刻 t での SP2 の出力値と， VPP での時刻 t の前後の時刻の出力値から，1 次補間近似で求めた時刻 t の値を比較した結果，その差が 1 %以下であった．この誤差について，本コードでは問題ないことがユーザより判断され， vmodel 版のインストールの際，ここでの最適化を実施させることにした．

2.2.1.3 その他の修正

diode.f， gtcsp.f 以外の修正箇所について，以下に述べる．

(1) Makefile の修正

demo1 のオリジナルの Makefile において， gtcsp.f のオブジェクトファイルの作成手段を記述している箇所において，依存関係を記す部分で指定される gtcsp.f ファイルと，コマンド行で指定される gtcsp.f ファイルが次のように異なっていた．

```
gtcsp.o :  ../sys/gtcsp.f inc.f
        f77 -O -c /grp03/j3799/gtcsp/sys/gtcsp.f
```

このことに関して，ユーザより，"../sys/gtcsp.f"を使用することを確認した．そして，そのファイルをカレントディレクトリ（作業ディレクトリ）へコピーし，Makefile における，gtcsp.f に関する記述を次のようにした．

```
gtcsp.o :  gtcsp.f inc.f
        f77 -O -c gtcsp.f
```

(2) 実定数の倍精度化

オリジナル・ソースでは，変数や配列は倍精度化されているが，実定数は単精度で記述されていた．VPP では，実定数は4バイト，倍精度実定数は8バイトの記憶領域を占める．実定数で記述された数値は，倍精度演算において，4バイト→8バイトへ変換される．そして，その変換のときに誤差が生じる．単精度実定数と倍精度実定数の誤差の様子を次に示す．

2.13　と単精度実定数で記述	2.130000114440918
2.13d0 と倍精度実定数で記述	2.130000000000000

よって，実定数は倍精度実定数として記述する必要がある．オリジナル・ソースの倍精度化されていない実定数に対し，倍精度化を行なった．倍精度化の例を Fig.2.6 に示す．

2.2.1.4　デモ版の結果確認

ここまで作業した時点での，オリジナル・ソースの SP2 上での実行結果 (a) と，倍精度化したソースの SP2 上での実行結果 (b)，そして，倍精度化したソースの VPP 上での実行結果 (c) との比較を行なった．

demo1 では，(a) (b) (c) 全ての実行結果が一致した (但し，演算評価方法の最適化を抑止した場合)．demo2 では，(b) (c) が一致し，demo3 では，(a) (b) が一致した．

demo2 の結果の違いにおいて，XL FORTRAN における実数値の取り扱いに関して，中目黒の計算科学技術推進センターの SP2 担当者に問い合わせをした結果，以下のような回答が得られた．

> (1) 実数計算は，単精度，倍精度とも，8バイトで行なっている．
> (2) 従って，単精度実数は，計算前後に単精度←→倍精度の変換が入る．
> 　また，その変換時での，誤差は，保証されていない．

このことより，定数が倍精度化されていないオリジナル・ソースでは，実数計算において，誤差が生じていたと考えられる．それによって，計算結果に違いが生じてしまったと考えられる．

また，demo3 の結果の違いに関して，詳しく調査した結果，コード内で指定している配列 currnt と voltag の値について，ある時刻で Table 2.1 のような違いがあった．

ここでの出力の書式は，"d10.3" である．配列 currnt では，小数点以下第 11 位 (10 進 11 桁め) で違いが生じている．VPP 及び SP2 上での実数のデータタイプで表すことのできる値について Table 2.2 に示す．Table 2.2 より，15 桁より大きい桁では，誤差が生じてしまうことがわかる．よって，demo3 の結果の違いは，配列 currnt を決定する過程での演算の繰り返しによる誤差と思われる．

配列 voltag では，1 の位 (10 進 4 桁め) で違いが生じているが，値をリスト指示 WRITE 文 (write(6,*)) で出力すると，Table 2.3 のようになる．Table 2.3より，小数点以下第 8 位 (10 進 11 桁め) から違いが生じていることがわかる．これも，配列 currnt と同様に演算の繰り返しによる誤差と思われる．よって，VPP と SP2 の出力書式の変換方法の違いにより生じる違いである．

以上の demo2，demo3 の誤差に関して，本コードにおいて問題ないことが，ユーザによって確認されている．

2.2.2 vmodel 版のインストール

デモ版のインストールで得られた結果をもとに，vmodel 版の各ソースの修正を行なった．

2.2.2.1 中核プログラム (gtcsp.f) の修正

gtcsp.f において，領域外のアクセスエラーが発生したため，プログラムが正常に動作しなかった．これは，配列 CONSTA のサイズが 200 であるのに対し，233 を参照する処理があるためであった．オリジナル版でのアクセスを確認した結果，オリジナル版においても，233 を参照する処理があった．これについて，ユーザ殿に問い合わせた結果，対処として，単純に配列のサイズを大きくするという指示を受けた．よって，配列 CONSTA のサイズを決定している inc.f のパラメータ $cll の値を 200 から，250 に修正した．

2.2.2.2 vmodel 版の結果確認

実定数を倍精度化したソースを SP2 上で実行させた結果と，VPP 上で実行させた結果の比較を行なった．その結果，demo1 と同様，変数 time に違いが生じており，その影響で，ほかの変数の値にも違いが生じていた．これは，最適化や，数値の取り扱いによる誤差と思われる．time の値が，条件文に使われている箇所があるが，その誤差により実行パスが変わってしまい，他の値に影響を与えていると思われる．この実行結果の違いに関して，本コードで問題ないことが，ユーザによって確認されている．

2.3 ベクトル化作業

ベクトル化作業は，VPP にインストールしたソースコード (以下 VPP オリジナル版) に対して実施した．チューニングは，メインプログラム gtcsp.f を対象とした．

2.3.1 コスト分布

VPP における，各ルーチンのベクトルコスト分布を動的挙動解析ツール ANALYZER [7] を用いて調べた結果を Table 2.4 に示す．これより，サブルーチン MINVD がもっともベクトルコスト (v-cost) が高く，ベクトル長 (v-leng) が 111 あることがわかる．また，動的挙動解析ツール SAMPLER [8] によるプログラムの実行状態を調べた結果を Table 2.5 に示す．Table 2.5 より，サブルーチン MINVD は，全体の処理時間の 95 % を占めていることがわかる．よって，このサブルーチンに関して，効率のよいチューニングをすることにより，より高いベクトル化効果を得ることができ，高速に演算することができる．更に，MINVD 内のコスト分布を Table 2.6

に示す．Table 2.6 より，DO 70 ループが最もコストが高いことがわかる．VPP オリジナル版 MINVD を Fig.2.7 に示す．

2.3.2 チューニング作業

効率良くベクトル化されるように，次の手順でチューニング作業を進めた．

2.3.2.1 バンク競合の回避

ソースを確認した結果，Fig.2.8 のように，2次元配列の行方向（一定間隔引用）に計算をし，メモリ上のデータをとびとびにアクセスしている箇所がある．これは，バンクの競合を引き起こし，性能を悪くする場合がある．一般に，行方向に計算を進める場合，配列の第一寸法宣言子の値を Table 2.7 のようにするとよい．

バンクの競合とは，1つのバンク内でメモリアクセスが集中することである．ここで，バンクとは，独立に動作できる複数個の部分に分割された主記憶の各々の部分のことである．各バンクは，独立してアクセスができる．例えば，バンク数を4とした場合，"real*8 A(4,4)" と宣言された配列のメモリ上の配置は，Fig.2.9 のようになる．

行方向の計算をする場合，メモリ上を Fig.2.9 の○印のように，同じバンクに属する配列要素を順次アクセスしていくと，バンクの競合を起こし，効率が悪い．この時，配列 A を "real*8 A(5,4)" と宣言し，Fig.2.10 の☆印のようなアクセスをすることにより，バンクの競合を回避できる．

このことを利用し，2 次元配列のサイズを決定している inc.f のパラメータ $syl に 1 を加え，バンクの競合を回避した．

2.3.2.2 if 文の削除

サブルーチン MINVD の DO 10，DO 60 ループにおいて，if 文をはずして処理をさせると，より効率よく実行することができる．DO 10 ループは，配列 r を配列 a へ代入し，変数 i と変数 j が等しいとき 1 を，それ以外は 0 を配列 r へ代入するという処理を行なっている．この処理は，

- 配列 r を配列 a へ代入する．
- 配列 r へ 0 を代入する．
- 配列 r へ 1 を代入する．

という3つの処理に分けることで，if 文をはずすことができる．修正した DO 10 ループを Fig.2.11 に示す．

DO 60 ループ内の処理は，変数 i と 変数 j が等しくないとき実行される．そこで，変数 i と変数 j が等しいときも実行されるようにすると，DO 70 ループ内の pivot は，a(j,i) と置き換えることができ，if 文をはずすことができる．変数 i と 変数 j が等しいときの処理について，そのときに使用される配列 a，r を，DO 70 ループの外で，配列 tmpa，tmpr，piv へ格納しておく．そして，DO 70 ループの処理後，配列 tmpa，tmpr の値を配列 a，r へ戻すことで，DO 60 ループと同等の処理が行なわれる．if 文をはずしたものを，Fig.2.12 に示す．

2.3.2.3　ロード / ストアの効率化

ロード / ストアを効率的に動作させるために，DO 60 ループの if 文をはずしたときに作成した DO 61 ループと元からある DO 50 ループの 2 つのループを合成した．合成後の DO 50 ループを Fig.2.13 に示す．

2.3.2.4　ループアンローリング

DO 70 ループにおいて，演算密度を高くするために，DO ループの繰り返し回数を減らすループアンローリングを行なった．最適化制御行 "*vocl loop,unroll(n)" を外側 DO ループの前に挿入し，ループアンローリングの指示をした．n は展開数であり，2 ～ 9 の符合なし正定数を指定する．ここでは，「n = 4」のとき，もっとも性能がよかった．修正した DO 70 ループと，DO 70 ループにおけるループアンローリングの実行イメージを Fig.2.14 に示す．

2.3.2.5　メモリアクセスの連続化

サブルーチン MINVD の DO 40，及び，DO 50 ループでは，2 次元配列の行方向に計算をしており，メモリ上のデータをとびとびにアクセスし効率が悪い．そこで，DO 40，DO 50 ループを含む DO 20 ループ内の配列に関して，添字の入れ換えを行うことで，効率のよい列方向（連続引用）の計算にできる．その列方向の計算のため，DO 10 ループの a に配列 r を代入する処理を，Fig.2.15 のように 配列 a の添字を入れ換え代入した．そして，DO 20 ループ内において，配列 r で演算を行なってたものを，新たな配列 b で演算させ，DO 20 ループの演算が終了後，その配列 b を添字を入れ換え，配列 r に代入した．チューニング後のサブルーチン MINVD を Fig.2.16 に示す．

2.4　ベクトル化効果

ここでは，チューニング版の計算結果，及び効果について述べる．

2.4.1　計算結果

チューニング作業を行なったソース (以下 VPP チューニング版) を用いて実行した実行結果と，倍精度化したソースを SP2 上で実行した実行結果の比較をした．その結果，違いが生じていた．そこで，既に問題ないと判断された，VPP オリジナル版での実行結果と，VPP チューニング版の実行結果の比較を行ない，一致することを確認した．

2.4.2　ベクトル化効果

SP2 オリジナル版の経過時間 (REAL) と CPU 時間を，timef 及び mclock プロシージャを用いて測定した．また，VPP オリジナル版，VPP チューニング版における経過時間，CPU 時間，VU 時間 (CPU 時間のうちベクトル命令の動作した時間) を，GETTOD 及び CLOCKV サービスサブルーチンを用いて測定した．測定した結果を Table 2.8 に示す．

これより，VPP オリジナル版，VPP チューニング版の CPU 時間を比較すると約 12.6 倍，同じく SP2 オリジナル版と VPP チューニング版を比較すると，約 5.9 倍の倍率が得られた．

VPP チューニング版の ANALYZER によるコスト分布を Table 2.9に示す．サブルーチン MINVD に関しては，VPP オリジナル版の 17.2 - 28.2 倍から，19.0 - 33.5 倍とより高いベクトル化効果が得られた．また，チューニング後の SAMPLER による実行状態を調べた結果を Table 2.10 に示す．これより，全体の 95 % も時間がかかっていたサブルーチン MINVD が，全体の半分ほどの 48 % になったことがわかる．

2.5 まとめ

本作業では，GTCSP を VPP へインストールし，ベクトル化を行なった．サブルーチン MINVD が，実行時間の 95 % と大部分を占めており，このサブルーチンを効率よくチューニングすることにより，よりよい性能が得られると考えた．チューニング後のサブルーチン MINVD のベクトルコストは，チューニング前とほとんど変わりないのだが，このサブルーチン内では，不連続なデータアクセスをしている箇所や，メモリバンクの競合を起こしている箇所があり，これらの実行効率を悪くしている原因を解決することで，実行時間を短くすることができた．データのアクセスの仕方によって，特に，メモリバンクの競合を起こす場合と起こさない場合では，性能に大きな違いがでることがわかった．

この高速化により，より大きな回路モデルの実行が可能になると思われる．

Table. 2.1 Difference with value of array currnt and voltag.

配列名	SP2	VPP
currnt	3.569D-11	1.302d-11
voltag	1.355D+03	1.356d+03

Table. 2.2 Data type of real number.

	SP2	VPP
浮動小数点データ形式	IEEE 形式	IEEE 形式
絶対値 (最小～最大)	2.225074D-308 ～ 1.797693D+308	約 2.2e-308 ～ 約 1.8e+308
近似精度 (10 進数)	15 桁	約 16 桁

Table. 2.3 Difference with value with list-directed formatting.

SP2	1355.49993879100748
VPP	1355.499938789490

Table. 2.4 Dynamic behaivor of original version.

name	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
MINVD	125465	.3502E12	65.0	.8033E13	97.5	111	99.3	17.2- 28.2
COMP	125465	.1033E12	19.2	.1079E12	1.3	23	5.0	1.0- 1.0
CALM	234225	.6199E11	11.5	.6235E11	0.8	111	0.6	1.0- 1.0
K3THV1	4014816	.5020E10	0.9	.1046E11	0.1	5	85.5	1.2- 2.7
K2THV1	3480352	.4393E10	0.8	.6195E10	0.1	5	47.3	1.1- 1.5
K1THV1	3480352	.4330E10	0.8	.6031E10	0.1	5	45.8	1.1- 1.5
DCOMN	234224	.2829E10	0.5	.2959E10	0.0	4	66.6	0.9- 1.3
CONCG	108761	.2165E10	0.4	.5857E10	0.1	9	74.3	2.2- 3.0
DIODE	3480352	.1629E10	0.3	.1806E10	0.0	4	16.5	1.1- 1.1
K5THV1	374304	.7280E09	0.1	.9441E09	0.0	6	37.2	1.1- 1.4
SYSWEI	120462	.5624E09	0.1	.1001E10	0.0	5	82.2	1.2- 2.2
GTCSP	1	.5404E09	0.1	.5442E09	0.0	34	0.8	1.0- 1.0
GEN1BC	120460	.3652E09	0.1	.4676E09	0.0	4	48.8	1.1- 1.4
K4THV1	3854656	.2541E09	0.0	.6081E09	0.0	5	94.6	1.3- 3.4
K1GEN1	108762	.1041E09	0.0	.1803E09	0.0	4	76.6	1.2- 2.1
K2GEN1	108761	84635532	0.0	.1573E09	0.0	5	80.0	1.2- 2.3
K6THV1	16733	72626018	0.0	.1208E09	0.0	5	67.7	1.2- 1.9
DISCTH	66080	65133093	0.0	.1560E09	0.0	5	94.6	1.3- 3.4
K3GEN1	125463	62931813	0.0	.1048E09	0.0	4	74.2	1.2- 2.0
K2COIL	108761	31067385	0.0	28408972	0.0	2	60.8	0.8- 1.2
K1COIL	108761	30594486	0.0	28116288	0.0	2	57.3	0.8- 1.2
SYSANA	27	15411359	0.0	15450433	0.0	3	1.5	1.0- 1.0
K5GEN1	11697	11195490	0.0	21577283	0.0	4	83.4	1.2- 2.4
K4COIL	120458	7875706	0.0	7190862	0.0	2	61.9	0.8- 1.2
SYSON	16733	7835444	0.0	7791656	0.0	8	3.7	1.0- 1.0
K5COIL	11697	6667748	0.0	6019839	0.0	2	70.0	0.7- 1.2
K4GEN1	120458	6334807	0.0	14552935	0.0	4	91.8	1.3- 3.1
K3COIL	125463	4369148	0.0	4369148	0.0	1	0.0	1.0- 1.0
SYSPRT	2065	3371316	0.0	3371316	0.0	20	0.0	1.0- 1.0
THMAP	16733	3095286	0.0	3095286	0.0	1	0.0	1.0- 1.0
DISCG1	2065	1672142	0.0	1890599	0.0	4	37.2	1.0- 1.2
VMECNTV	2031	1209106	0.0	1209106	0.0	1	0.0	1.0- 1.0
VMECNTF	2031	1199154	0.0	1199154	0.0	1	0.0	1.0- 1.0
DISCCO	2065	898951	0.0	811817	0.0	2	69.8	0.7- 1.2
K0THV1	32	15200	0.0	20576	0.0	4	42.5	1.1- 1.5
DCOMO	1	8614	0.0	9009	0.0	4	66.6	0.9- 1.3
SYSINP	1	2708	0.0	2708	0.0	13	0.0	1.0- 1.0
K0COIL	1	684	0.0	628	0.0	2	57.6	0.8- 1.2
K6COIL	1	467	0.0	422	0.0	2	69.2	0.7- 1.2
K0GEN1	1	363	0.0	499	0.0	4	60.7	1.1- 1.6
MAIN	1	5	0.0	5	0.0	1	0.0	1.0- 1.0
BLOCKD	0	0	0	0	0	0	0	0
DISCGE	0	0	0	0	0	0	0	0
DISCNB	0	0	0	0	0	0	0	0
				:				
				:				
(total)	-------	.5389E12	100.0	.8241E13	100.0	------	97.1	12.4- 17.5

Table. 2.5 Distribution of computational costs of original version.

Count	Percent	VL	Name
2034459	95.4	112	minvd_
58269	2.7	21	comp_
9584	0.4	4	k3thv1_
5105	0.2	-	gtcsp_
5055	0.2	31	concg_
4998	0.2	1	dcomn_
3192	0.1	4	k1thv1_
2902	0.1	4	k2thv1_
2664	0.1	12	diode_
2185	0.1	4	discth_
849	0.0	112	calm_
608	0.0	5	syswei_
506	0.0	5	gen1bc_
425	0.0	7	k4thv1_
213	0.0	6	k2gen1_
146	0.0	6	k1gen1_
119	0.0	5	k5thv1_
116	0.0	3	discg1_
114	0.0	2	k1coil_
92	0.0	2	k2coil_
82	0.0	-	discco_
67	0.0	6	k3gen1_
58	0.0	-	sysprt_
56	0.0	2	k4coil_
50	0.0	2	syson_
43	0.0	9	k6thv1_
30	0.0	6	k4gen1_
27	0.0	2	sysana_
22	0.0	-	k3coil_
13	0.0	-	k0thv1_
8	0.0	-	k5coil_
7	0.0	-	thmap_
7	0.0	6	k5gen1_
4	0.0	-	vmecntf_
1	0.0	-	sysinp_
1	0.0	-	vmecntv_
2132077		111	TOTAL

Table. 2.6 Dynamic behaivor of DO loops in subroutine MINVD.

```
vectorize - loop list ------- MINVD ----------------------------------------

do-id  kind  v  ex-count   v-cost    %    s-cost    % v-leng v-rate    v-effect
70     do    V  .2262E10 .2877E12 82.1 .7854E13 97.8    112 100.0  19.5- 35.1
10     do    S  20381635 .4569E11 13.0 .4569E11  0.6    112   0.0   1.0-  1.0
60     do    V  20381635 .1206E11  3.4 .3177E11  0.4    112  64.4   2.6-  2.7
50     do    V  20381635 .3261E10  0.9 .8902E11  1.1    112 100.0  19.5- 35.1
30     do    V  20381635 .7251E09  0.2 .1131E11  0.1     55 100.0  11.7- 19.5
20     do    S    181987 .6730E09  0.2 .6730E09  0.0    112   0.0   1.0-  1.0
10     do    V    181987 .1019E09  0.0 .1019E09  0.0    112   0.0   1.0-  1.0
40     do    V     76757 11020629  0.0 .3009E09  0.0    112 100.0  19.5- 35.1
-----  total      125465  2729803  0.0  2729803  0.0      1   0.0   1.0-  1.0
```

Table. 2.7 Definition of array size.

type*m	REAL*4 integer*4 logical*4	REAL*8 complex*8 complex*16
k	4n+2	2n+1

k: 第一寸法子 a(k,m)

n: 正定数

Table. 2.8 Comparison of execution time.

	REAL	CPU 時間	VU 時間
SP2 オリジナル版	7h 27m 2s	6h 47m 34s	——
VPP オリジナル版	14h 36m 25s	14h 35m 03s	14h 14m 15s
VPP チューニング版	1h 09m 39s	1h 09m 29s	49m 41s

Table. 2.9 Dynamic behaivor of vectorized version.

name	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
MINVD	298415	.5586E12	64.7	.1470E14	97.8	111	99.9	19.0- 33.5
COMP	298415	.1694E12	19.6	.1769E12	1.2	23	5.0	1.0- 1.0
CALM	522589	.9861E11	11.4	.9917E11	0.7	111	0.6	1.0- 1.0
K3THV1	9549248	.8007E10	0.9	.1661E11	0.1	5	85.3	1.2- 2.7
K2THV1	7173568	.6724E10	0.8	.9483E10	0.1	5	47.3	1.1- 1.5
K1THV1	7173568	.6628E10	0.8	.9232E10	0.1	5	45.8	1.1- 1.5
DCOMN	522588	.4500E10	0.5	.4707E10	0.0	4	66.6	0.9- 1.3
CONCG	224174	.3313E10	0.4	.8964E10	0.1	9	74.3	2.2- 3.0
DIODE	7173568	.2782E10	0.3	.3130E10	0.0	4	18.6	1.1- 1.2
K5THV1	1594112	.1464E10	0.2	.1899E10	0.0	6	37.2	1.1- 1.4
SYSWEI	273993	.9001E09	0.1	.1602E10	0.0	5	82.2	1.2- 2.2
GTCSP	1	.8614E09	0.1	.8691E09	0.0	34	1.0	1.0- 1.0
GEN1BC	273991	.5844E09	0.1	.7483E09	0.0	4	48.8	1.1- 1.4
K4THV1	8767680	.4067E09	0.0	.9732E09	0.0	4	94.6	1.3- 3.4
SYSANA	428	.2785E09	0.0	.2792E09	0.0	3	1.5	1.0- 1.0
K1GEN1	224174	.1594E09	0.0	.2760E09	0.0	4	76.6	1.2- 2.1
K6THV1	74271	.1516E09	0.0	.2521E09	0.0	5	67.7	1.2- 1.9
K2GEN1	224174	.1296E09	0.0	.2408E09	0.0	4	80.0	1.2- 2.3
K3GEN1	298414	.1032E09	0.0	.1719E09	0.0	4	74.2	1.2- 2.0
DISCTH	127968	96576657	0.0	.2312E09	0.0	5	94.6	1.3- 3.4
K2COIL	224174	47559376	0.0	43489756	0.0	2	60.8	0.8- 1.2
K1COIL	224174	46835122	0.0	43041408	0.0	2	57.3	0.8- 1.2
K5GEN1	49816	22513000	0.0	43389736	0.0	4	83.4	1.2- 2.4
SYSON	74271	16370481	0.0	16279071	0.0	7	3.6	1.0- 1.0
K5COIL	49816	13408168	0.0	12105288	0.0	2	70.0	0.7- 1.2
K4COIL	273990	12603540	0.0	11507580	0.0	2	61.9	0.8- 1.2
K4GEN1	273990	10137630	0.0	23289150	0.0	4	91.8	1.3- 3.1
K3COIL	298414	7163444	0.0	7163444	0.0	1	0.0	1.0- 1.0
THMAP	74271	6461577	0.0	6461577	0.0	1	0.0	1.0- 1.0
SYSPRT	3999	4998849	0.0	4998849	0.0	18	0.0	1.0- 1.0
DISCG1	3999	2479383	0.0	2803302	0.0	4	37.2	1.0- 1.2
VMECNT	3946	1768464	0.0	1768464	0.0	1	0.0	1.0- 1.0
VMECNT	3946	1738786	0.0	1738786	0.0	1	0.0	1.0- 1.0
DISCCO	3999	1332927	0.0	1203729	0.0	2	69.8	0.7- 1.2
K0THV1	32	15200	0.0	20576	0.0	4	42.5	1.1- 1.5
DCOMO	1	8614	0.0	9009	0.0	4	66.6	0.9- 1.3
SYSINP	1	2708	0.0	2708	0.0	13	0.0	1.0- 1.0
K0COIL	1	684	0.0	628	0.0	2	57.6	0.8- 1.2
K6COIL	1	467	0.0	422	0.0	2	69.2	0.7- 1.2
K0GEN1	1	363	0.0	499	0.0	4	60.7	1.1- 1.6
MAIN	1	11	0.0	11	0.0	1	0.0	1.0- 1.0
BLOCKD	0	0	0	0	0	0	0	0
DISCGE	0	0	0	0	0	0	0	0
MULTV	0	0	0	0	0	0	0	0
				:				
				:				
(total)	--------	.8639E12	100.0	.1503E14	100.0	------	97.9	13.8- 20.3

Table. 2.10 Distribution of computational costs of vectorized version.

Count	Percent	VL	Name
208457	48.4	111	minvd_
131694	30.6	20	comp_
20441	4.7	4	k3thv1_
12608	2.9	2	dcomn_
12114	2.8	26	gtcsp_
10506	2.4	27	concg_
7635	1.8	4	k2thv1_
7013	1.6	13	diode_
6831	1.6	4	k1thv1_
4524	1.0	4	discth_
1741	0.4	112	calm_
1539	0.4	5	syswei_
1285	0.3	5	gen1bc_
914	0.2	7	k4thv1_
737	0.2	5	k5thv1_
455	0.1	2	sysana_
364	0.1	6	k1gen1_
293	0.1	6	k2gen1_
253	0.1	3	discg1_
211	0.0	6	k3gen1_
211	0.0	6	k6thv1_
202	0.0	2	k1coil_
195	0.0	2	k2coil_
174	0.0	2	syson_
147	0.0	2	discco_
121	0.0	-	sysprt_
84	0.0	2	k4coil_
72	0.0	-	k3coil_
54	0.0	6	k4gen1_
50	0.0	-	thmap_
47	0.0	6	k5gen1_
44	0.0	3	k5coil_
13	0.0	-	k0thv1_
13	0.0	-	vmecntf_
8	0.0	-	vmecntv_
431050		86	TOTAL

```
      implicit integer($)
      parameter ($ttt=  20,$nth=  40,$syl= 160,$ilp=1000,$psy=1000,
     >           $tem= 100,$smt=  50,$mt =  80,$alc= 300,$bml= 600,
     >           $cll= 200,$cop=  10,$nco=  10,$cor=  20,$col= 200,
     >           $nge=   2,$nbi=  30,$nbc=  30)
      character*4 name
```

Fig. 2.1 Variable array data (demo1).

```
      subroutine diode(jd)
      implicit real*8(a-h,o-z)
      integer bmap,nbwid
      include 'inc.f'
```

Fig. 2.2 Specification statement and compiler directive statement in diode.f (demo1).

```
      subroutine diode(jd)
      implicit real*8(a-h,o-z)
      include 'inc.f'
      integer bmap,nbwid
```

Fig. 2.3 Modification of specification statement in diode.f (demo1).

```
(修正前)
      do 10 i=1,6
      if(ithsta(i,id).ne.3) go to 10        ! already offed
      sumn=0.0d0
      do 20 j=1,4
   20 sumn=sumn       +scstcn(ifscst+j+2)*d(i,j,id)
      if(sumn.gt.0.0d0) go to 10          !         no  cut off point
c *** cut off ***  two degree interpolation of sum
      sumo=0.0d0
      sumdn=0.0d0
      sumdo=0.0d0
      do 40 j=1,4
      sumo =sumo         +scstco(ifscst+2+j)*d(i,j,id)
      sumdn=sumdn        +scstdn(ifscst+2+j)*d(i,j,id)
   40 sumdo=sumdo        +scstdo(ifscst+2+j)*d(i,j,id)
      if(sumn+sumdn*1.0d-07.ge.0.0d+0) go to 10
      if(deltat.eq.0.0d+0) go to 136   !(k7)                    ← (a)
      if(sumo.lt.0.0d0) go to 50            ! error    already cut off
      epss=dabs(sumdn)*1.0d-8+eps
      ssum=sumdn-sumdo
      if(dabs(ssum).lt.epss) go to 60     ! linear interpolation
      dsum=sumdo*sumdo-2.0d+0*sumo*ssum/deltat                  ← (b)
      if(dsum.lt.0.0d0) go to 70            ! error   no real root

                        :
(修正後)
*vocl loop, nopreex                                             ←(追加行)
      do 10 i=1,6
      if(ithsta(i,id).ne.3) go to 10        ! already offed
      sumn=0.0d0
      do 20 j=1,4
                        :
```

Fig. 2.4 Modification of DO 10 loop in subroutine K3THV1.

```
*vocl loop, noeval                                          ←(追加行)
        do 50 k = 1,n
          a(k,i) = a(k,i) / pivot
          b(k,i) = b(k,i) / pivot
          tmpa(k)  = a(k,i)
          tmpb(k)  = b(k,i)
          piv(k)   = a(i,k)
   50   continue
                        :
```

Fig. 2.5 Modification of DO 50 loop in subroutine MINVD.

```
(修正前)
                        :
      do 10 iwork=1,psysm
      do 20 jwork=1,psysm
   20 syscal(iwork,jwork)=0.0     ! syscal clear
   10 syscon(iwork)=0.0           ! syscon clear
                        :
      if(sume.gt.0.0) go to 190    ! no on-possible  eq---off mode ---
                        :

(修正後)
                        :
      do 10 iwork=1,psysm
      do 20 jwork=1,psysm
   20 syscal(iwork,jwork)=0.0d0     ! syscal clear
   10 syscon(iwork)=0.0d0           ! syscon clear
                        :
      if(sume.gt.0.0d0) go to 190    ! no on-possible  eq---off mode ---
                        :
```

Fig. 2.6 Example of modification of double precision type.

```
      subroutine minvd(r,n)
      include 'inc.f'
      real*8 r($syl,$syl),a($syl,$syl),amax,pivot,temp
      integer imax
c
      do 10 i = 1,n
      do 10 j = 1,n
      a(i,j) = r(i,j)
      if(i.eq.j) then
        r(i,j) = 1.0d0
      else
        r(i,j) = 0.0d0
      endif
   10 continue
c
      do 20 i = 1,n
        amax = dabs(a(i,i))
        imax = i
        do 30 j = i+1,n
          if(dabs(a(j,i)).gt.amax) then
            amax = dabs(a(j,i))
            imax = j
          endif
   30   continue
        if(imax.ne.i) then
          do 40 k = 1,n
            temp = a(i,k)
            a(i,k) = a(imax,k)
            a(imax,k) = temp
            temp = r(i,k)
            r(i,k) = r(imax,k)
            r(imax,k) = temp
   40     continue
        endif
c
        pivot = a(i,i)
        do 50 k = 1,n
        a(i,k) = a(i,k) / pivot
   50   r(i,k) = r(i,k) / pivot
c
        do 60 j = 1,n
          if(i.ne.j) then
            pivot = a(j,i)
            do 70 k = 1,n
              a(j,k) = a(j,k) - a(i,k) * pivot
   70         r(j,k) = r(j,k) - r(i,k) * pivot
          endif
   60   continue
   20 continue
      return
      end
```

Fig. 2.7 Subroutine MINVD (original version).

```
(サブルーチン MINVD)
          :
      do 40 k = 1,n
        temp = a(i,k)
        a(i,k) = a(imax,k)
        a(imax,k) = temp
          :
```

Fig. 2.8 Example of coding which causes bank-rivalry.

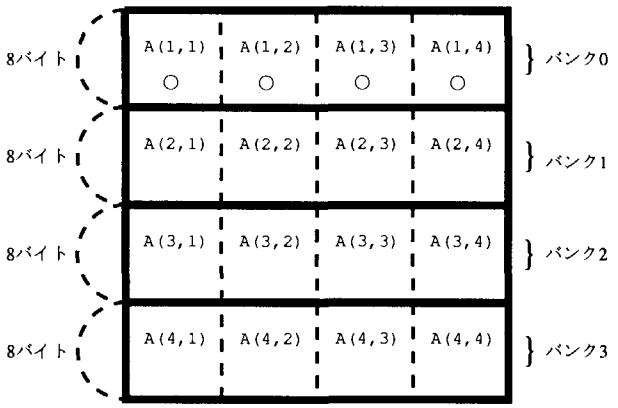

Fig. 2.9 Memory map (in case of array A(4,4)).

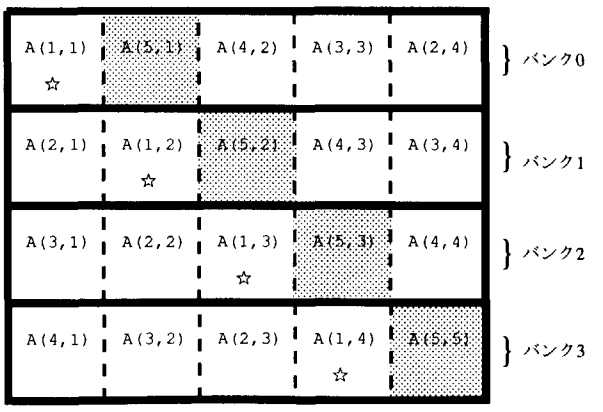

Fig. 2.10 Memory map (in case of array A(5,5)).

```
(修正前)
 v2     do 10 i = 1,n
 s2     do 10 j = 1,n
 v2     a(i,j) = r(i,j)
 v2     if(i.eq.j) then
 v2       r(i,j) = 1.0d0
 v2     else
 v2       r(i,j) = 0.0d0
 v2     endif
 v2 10  continue

(修正後)
 s2        do 10 j = 1,n
 v2        do 10 i = 1,n
 v2          a(i,j) = r(i,j)
 v2     10 continue
c
 s2        do 11 j = 1,n
 v2        do 11 i = 1,n
 v2          r(i,j) = 0.0d0
 v2     11 continue
c
 v         do 12 i = 1,n
 v           r(i,i) = 1.0d0
 v      12 continue
```

Fig. 2.11 Modification of DO 10 loop in subroutine MINVD.

```
(修正前)
 s            do 60 j = 1,n
 v            if(i.ne.j) then
 v              pivot = a(j,i)
 v              do 70 k = 1,n
 v                a(j,k) = a(j,k) - a(i,k) * pivot
 v       70       r(j,k) = r(j,k) - r(i,k) * pivot
 v            endif
 v       60   continue

(修正後)
 v           do 61 j = 1,n
 v             tmpa(j) = a(i,j)
 v             tmpr(j) = r(i,j)
 v             piv(j)  = a(j,i)
 v       61  continue
     c
 s2          do 70 k = 1,n
 v2          do 70 j = 1,n
 v2            a(j,k) = a(j,k) - tmpa(k) * piv(j)
 v2            r(j,k) = r(j,k) - tmpr(k) * piv(j)
 v2      70  continue
     c
 v           do 62 j = 1,n
 v             a(i,j) = tmpa(j)
 v             r(i,j) = tmpr(j)
 v       62  continue
```

Fig. 2.12 Modification of DO 60 loop in subroutine MINVD.

```
v           do 50 k = 1,n
v           a(i,k) = a(i,k) / pivot            ( DO 50 loop )
v           r(i,k) = r(i,k) / pivot
v             tmpa(k) = a(i,k)                 ( DO 61 loop )
v             tmpr(k) = r(i,k)
v             piv(k)  = a(k,i)
v      50   continue
```

Fig. 2.13 Merger of DO 61 loop and DO 50 loop in subroutine MINVD.

```
(修正後)
            *vocl loop,unroll(4)                          ← (追加行)
            do 70 k = 1,n
            do 70 j = 1,n
              a(j,k) = a(j,k) - tmpa(k) * piv(j)
              r(j,k) = r(j,k) - tmpr(k) * piv(j)
       70   continue


(実行イメージ)
            do 70 k = 1,n,4
            do 70 j = 1,n
              a(j,k)   = a(j,k)   - tmpa(k)   * piv(j)
              a(j,k+1) = a(j,k+1) - tmpa(k+1) * piv(j)
              a(j,k+2) = a(j,k+2) - tmpa(k+2) * piv(j)
              a(j,k+3) = a(j,k+3) - tmpa(k+3) * piv(j)
              r(j,k)   = r(j,k)   - tmpr(k)   * piv(j)
              r(j,k+1) = r(j,k+1) - tmpr(k+1) * piv(j)
              r(j,k+2) = r(j,k+2) - tmpr(k+2) * piv(j)
              r(j,k+3) = r(j,k+3) - tmpr(k+3) * piv(j)
       70   continue
```

Fig. 2.14 Direction of loop unrolling to DO 70 loop in subroutine MINVD.

```
s2        do 10 j = 1,n
v2        do 10 i = 1,n
v2          a(j,i) = r(i,j)
v2     10 continue
```

Fig. 2.15 Replacement of subscript list of array a in subroutine MINVD.

```
          subroutine minvd(r,n)
          include 'inc.f'
          real*8 r($syl,$syl),a($syl,$syl),amax,pivot,temp
          integer imax
          real*8 piv($syl),tmpa($syl),tmpb($syl)
          real*8 b($syl,$syl)
   c
s2        do 10 j = 1,n
v2        do 10 i = 1,n
v2          a(j,i) = r(i,j)
v2     10 continue
   c
s2        do 11 j = 1,n
v2        do 11 i = 1,n
v2          b(i,j) = 0.0d0
v2     11 continue
   c
v         do 12 i = 1,n
v            b(i,i) = 1.0d0
v      12 continue
   c
s         do 20 i = 1,n
s           amax = dabs(a(i,i))
s           imax = i
v           do 30 j = i+1,n
v              if(dabs(a(i,j)).gt.amax) then
v                amax = dabs(a(i,j))
v                imax = j
v              endif
v      30    continue
s           if(imax.ne.i) then
v             do 40 k = 1,n
v               temp = a(k,i)
v               a(k,i) = a(k,imax)
v               a(k,imax) = temp
v               temp = b(k,i)
v               b(k,i) = b(k,imax)
v               b(k,imax) = temp
v      40     continue
s           end if
   c
s           pivot = a(i,i)
   c
v           do 50 k = 1,n
v             a(k,i) = a(k,i) / pivot
v             b(k,i) = b(k,i) / pivot
v             tmpa(k)  = a(k,i)
v             tmpb(k)  = b(k,i)
v             piv(k)   = a(i,k)
v       50    continue
 c
```

Fig. 2.16 Subroutine MINVD (vectorized version) (1/2).

```
   *vocl loop,unroll(4)
v4           do 70 k = 1,n
s4           do 70 j = 1,n
v4             a(k,j) = a(k,j) - tmpa(k) * piv(j)
v4             b(k,j) = b(k,j) - tmpb(k) * piv(j)
v4       70   continue
   c
v             do 62 j = 1,n
v               a(j,i) = tmpa(j)
v               b(j,i) = tmpb(j)
v        62   continue
    c
s        20 continue
    c
s2          do 100 j = 1,n
v2          do 100 i = 1,n
v2            r(i,j) = b(j,i)
v2       100 continue
    c
          return
           end
```

Fig. 2.16 Subroutine MINVD (vectorized version) (2/2).

参 考 文 献

[1] 松川誠, 青柳哲雄, 三浦友史;「汎用トカマク回路シミュレーションプログラム GTCSP」, JAERI-Data/Code 97-017, 1997年5月.

[2] 「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 情報システム管理室, 1995年6月.

[3] 「那珂地区データ解析サーバ IBM RS/6000 SP System 利用者手引」, 日本原子力研究所 情報システム管理課, 1996年9月.

[4] 「XL Fortran for AIX Language Reference Version3 Release2」, 日本IBM(株), 1995年9月.

[5] 「XL Fortran for AIX User's Guide Version3 Release2」, 日本IBM(株), 1995年9月.

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

[7] 「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.

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

3. MSP2 コードの並列化

3.1 はじめに

本作業では，イオン性融体分子動力学計算コードの一部である，動径分布関数を求めるルーチン（msp2）[1,2] の並列化を行なった．この並列化では，大規模シミュレーションを実施したいとのユーザの要望を元に，領域拡張を目的としている．今回の並列化により粒子数を現状の１０倍程度まで増加させたシミュレーションが可能となっている．

本コードの並列化には、粒子数をプロセッサ台数分に分割して，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粒子分割の手法を適用した．実行対象計算機は，分散メモリ型ベクトル並列計算機 VPP500 [2,3] である．

3.2 作業方針

本コードは既にベクトル化されており，今回の作業ではベクトル化の必要はない．並列化作業は粒子数の拡張に主眼を置くものとする．しかし，並列化に伴いベクトル処理の効果が変化することがある．さらに，本コードの場合は空間内の全粒子を対象としているため，プロセッサ間のデータ転送は必要不可欠となる．よって，演算時間についてはオリジナル実行時間より並列実行時間が小さければ良いことにする．並列化手法は，粒子数の拡張が目的であることを考慮し，以下の３点とした．

1）粒子分割とする．
2）配列の定義については，粒子に関わるものは全て分割配列として定義する．粒子数と関係ない他の配列については，変更点を少なくするために分割せず重複配列として定義する．
3）演算時間は，オリジナル実行時間と比較して等倍以下とする．

3.3 ワーク領域の縮小

前回のベクトル化により，メモリ上にベクトル処理用のワーク領域として使用される配列が確保されている．この配列は粒子数の二乗のオーダーであり，全体の８０％を占める大きな配列である．ゆえに，ワーク領域用配列を削除すればメモリ使用量を大幅に削減出来る．ところが，単純にワーク領域用配列を削除しただけでは，ベクトル化の効果が無くなり，性能が劣化する．そこで，ベクトル効果を引き出せるように処理フローを変更することで，性能を維持したままワーク領域用配列を削除する．削除後，ワーク領域用配列に変わるものとしてワーク領域用変数を導入する．しかし，処理フローを変更するとプログラムの構造が変化するという欠点がある．今まで，処理毎に分かりやすく記述していた部分を一括記述するため，どのような演算をしているかが分かり難くなる．そこで，なるべく処理毎の計算式を温存し，コメントを挿入する等の手段で混乱を避けられるように努め，ワーク領域用配列の削除およびワーク領域用変数の導入を行なっ

た．これにより，粒子数の二乗必要であった領域を変数のレベルにまで縮小することに成功した．変更箇所について，オリジナル版のワーク領域用配列定義部分を Fig. 3.1 に，変更後のワーク領域用変数定義部分を Fig. 3.2 に示す．次に，計算処理部の変更箇所について，オリジナル版を Fig. 3.3 に，変更後を Fig. 3.4 に示す．

3.4 計算精度

本コードの変数配列は，ＲＥＡＬ＊４（単精度）であった．これでは演算順序を変更した場合，丸め誤差により演算結果が変化する可能性が高い．その変化は微々たるものであるが，数千タイムステップ経過すると，無視できない程大きなものになる．このような誤差により，条件文（ＩＦ文）の真偽率が変わり，演算結果に影響を及ぼすこともあり得る．また，並列処理時にはベクトル状況がオリジナル版とは異なるため，丸め誤差が生じる．このような誤差を極力抑えるには，変数配列の精度を拡張すればよい．つまり，ＲＥＡＬ＊４（単精度）からＲＥＡＬ＊８（倍精度）へと拡張することである．精度拡張により計算精度が向上し，より正確な演算が可能となる．そこで，本コード内の全変数をＲＥＡＬ＊４からＲＥＡＬ＊８へと変更し，オリジナルのＲＥＡＬ＊８版を作成した．その後，ワーク領域用配列を削除し，処理フローの変更を加えた．ところが，精度拡張したにも関わらず，オリジナルＲＥＡＬ＊８と変更後ＲＥＡＬ＊８での実行結果を比較すると，座標位置に関する出力結果に相違が見られた．この現象に関する調査の結果，条件文の真偽率が変化したためであることが判明した．オリジナル版では，判定が真であったものが丸め誤差の影響により偽と判定される，あるいはその逆の現象である．このような現象を表面的に解決する手法として，判定に幅をもたせる方法がある．しかし，このような方法では根本的な解決とはならず，あいまいさを残すだけである．そこで，今回の作業では条件文に判定幅をもたせる方法は適用しないこととした．　座標位置に関する出力結果については，一つの特定粒子だけを見ると間違った演算をしているように見えるが，系全体を考慮した場合の物理的現象が意味のあるのもであれば，分子動力学コードとして正しいシミュレーションをしていると言える．今回の出力結果の正当性について，ユーザに確認をとったところ，誤差はあるものの許容範囲内であることが確かめられた．よって，座標位置データについても一つの系として見た場合，その分布状況は物理的に意味のあるものであり，正しいシミュレーション結果であることが確認された．

3.5 並列化

3.5.1 分割軸の決定

作業方針にも述べたが，分割方法は粒子分割である．本コード内における粒子数ループは，Ｉ軸である．よって，分割軸はＩ軸とした．本コードにおいて，Ｉ軸を分割軸とすると，ベクトル性能が劣化する要因となる．テストケースでの粒子数は１０００であり，４並列では各プロセッサにおけるベクトル長は２５０となる．ベクトル長１０００と２５０では，１０００の方がベクトル効果が高くなるためである．しかし，粒子数の増加に対応するには，粒子分割が必要である．今回の並列化では粒子数の拡張を主目的としているため，多少のベクトル性能の劣化はやむを得ないこととし，並列処理による効果で劣化分をカバーできると判断した．将来的に粒子数

が数万のオーダーになれば，１プロセッサ当たりの粒子数は数千となり，十分なベクトル効果が期待できる．ゆえに，大規模データの場合には，ベクトル性能も十分発揮できると予測される．並列処理用に新たに追加した変数配列の分割指定部を Fig. 3.5 に示す．パラメータ iipe は，並列数を指定する変数である．この数値を変更することにより，並列数を変えることができる．実行対象マシンである VPP500 の場合は，１～１６の整数を指定できる．ただし，変更した場合は，リコンパイルの必要がある．

3.5.2 データ入力部の変更

並列処理内部において，ファイルから粒子の位置データを読み込む部分が存在する．粒子属性の配列は分割されており，そのままでは位置データを読み込むことができない．そこで，並列処理用に追加したデータ転送用配列（配列 Qwork）を一時的に利用し，この配列へファイルから読み込むように変更した．その後，本来の分割配列へ代入する手法となっている．位置データ読み込み後，配列 Qwork を初期化しているため，他処理への影響はない．データ入力部の変更前を Fig. 3.6 に，データ入力部の変更後を Fig. 3.7 に示す．

3.5.3 データ転送

データ転送とは，プロセッサ間でデータを送受信することである．本作業におけるデータ転送はマスタプロセッサへの転送であり，マスタプロセッサ以外をスレーブプロセッサと呼ぶこととする．本コードの並列化に伴い，各粒子の属性値は複数プロセッサ上に分散している．よって，全粒子を対象に各粒子間の距離を計算する際には，スレーブプロセッサ上の値が必要となる．そのためには，スレーブプロセッサ上の値をマスタプロセッサ上に転送する必要がある．また，係数計算時には全粒子の属性値の総和計算が行なわれている．その際にも，スレーブプロセッサ上のデータをマスタプロセッサ上に集約し，合算する仕組みが必要となる．これらのデータ転送は，各タイムステップ内で実行する必要がある．１タイムステップ毎に粒子は移動し，その属性値も変化するためである．ゆえに，タイムステップ毎のデータ転送は必要不可欠である．ところが，タイムステップは通常数千から数万回実行されることを考慮すると，データ転送にかかるコストが大きくなり，並列効果が劣化する要因となる．しかし，今回の並列化は粒子分割であり，粒子属性のデータ転送は避けられない事象である．よって，並列効率の劣化はやむを得ないものとし，タイムステップ毎にデータ転送を実施することとした．

3.5.4 ファイル出力について

本コードはタイムステップ終了時に，各粒子の位置データをファイルへ出力する仕様である．通常の分散並列化では，各粒子の位置データは複数プロセッサ上に分散する．そのため，ファイル出力前にデータ転送が必要となる．しかし，グローバル配列を使用することでこのような転送を回避することができる．グローバル配列とは，全てのプロセッサから定義参照可能な配列を言う．このような配列を使用することで並列演算終了時に直接ファイルへ書き出すことが可能となる．ただし，オリジナルＩ／Ｏインターフェースについて，出力配列名をグローバル配列名へ変更する必要がある．今回の作業では、配列Ｐに対するグローバル配列を配列 PworkG として定義した．配列Ｑに対しては，グローバルＳＵＭ機構を使用するため重複ローカル配列

として配列Qworkを定義している．よって，今後プログラムを変更する場合は注意が必要である．並列処理終了後には，配列Pあるいは配列Qは使用不可能となる．つまり，配列Pは配列PworkGに変更されており，並列処理終了後の処理では配列Pに相当する配列PworkGを使用する必要がある．配列Qについても同様であり，配列Qに相当する配列Qworkを使用する必要がある．もし，並列処理終了後に配列Pおよび配列Qを使用すると，正常なシミュレーションは行なわれない．配列P、配列Qに関する変更箇所をFig. 3.8 に示す．

3.6 計算結果

本コードによる出力対象配列は４個あり，配列P，配列Q，配列JFR，配列FRが出力対象である．配列Pと配列Qについては出力結果がオリジナル版と一致していない．この点については計算精度の章でも述べたが，条件文の真偽率が変化したためであり，全体系として見た場合は正しいシミュレーションであることが確認されている．他の配列については，ほぼ等しい値となっている．オリジナル版配列Pの先頭５０行をFig. 3.9 に，配列QをFig. 3.10 に，配列JFRをFig. 3.11 に，配列FRをFig. 3.12 にそれぞれ示す．並列版については配列PをFig. 3.13 に，配列QをFig. 3.14 に，配列JFRをFig. 3.15 に，配列FRをFig. 3.16 にそれぞれ示す．

3.7 性能

並列化作業によるメモリ使用量削減効果について述べる．処理フローを変更し，ベクトル処理用のワーク領域配列を削除したことにより，メモリ使用量は対オリジナル比（並列版／オリジナル版）で約１／２０に削減されている．さらに，粒子属性の配列を分割したことにより，並列度による効果が上記効果に加算される．つまり，オリジナル版に対して２０倍以上の粒子数を指定することが出来る．Table 3.1 にケース別のメモリ使用量比較表を示す．今回のテストデータでは粒子数が１０００個であったため，分割並列による効果は大きくない．粒子属性を持たない配列の方が大きかったためである．

次に演算性能について述べる．まず，処理フローを変更したことに伴うベクトル性能の劣化は発生していない．むしろ，複数処理を一括記述したことによるロード・ストア回数の低減により，ベクトル性能は向上している．よって，変更後はオリジナル版と同様以上のベクトル性能を発揮している．並列性能については，並列処理に伴うデータ転送をタイムステップ毎に行なっているため，並列効果は４０%程度である．（４並列で計測）粒子数が少ない場合は並列数を増やすと性能が悪化する可能性がある．そのような場合は低並列で，粒子数が多い場合は高並列で実行する等，使い分けた方がより高いベクトル並列効果を得られる．Table 3.2 にケース別の実行時間比較表を示す．

3.8 考察

ポテンシャルに基づく力の計算処理内では，リストによる総和演算が成されている．このリスト総和演算は，大きなワーク領域を確保できれば，より効率良くベクトル処理できる．その処理ロジックの詳細については触れないが，必要となるおおよそのワーク領域の容量は式 (3.1) のよ

うに見積もることができる．（リスト総和演算に使用される配列名はJFRである）

$$\text{ワーク領域容量}=\text{配列}JFR\text{の容量}\times５００ \tag{3.1}$$

５００という数値は固定ではなく，通常であれば２００～１０００程度までの間でもっとも効率の良いものを選定する．この値の選定には，配列JFRの大きさも関係するため，配列JFRの大きさが変化した場合には選定値において最高性能が発揮できるとは限らないという欠点が含まれている．もし，配列JFRの大きさが固定であれば，最高性能時の値を固定値としてプログラム内に記述すればよい．　現状のVPP500では，使用可能メモリ量は約２００ＭＢ程度であり，上記のような処理を組み込むことはできなかった．今後，メモリが増強されるようであれば，この点についても考慮すべきであると判断する．

Table. 3.1 Comparision of quantity of memory each program used.

CASE	シングル	4並列	16並列
オリジナル	41.79 MB	——	——
ワーク領域削除ベクトル版	1.79 MB	——	——
通常並列化版	42.19 MB	12.00 MB	4.47 MB
ワーク領域削除並列化版	2.19 MB	2.00 MB	1.95 MB

ワーク領域削除ベクトル版：オリジナル版からベクトル用ワーク領域配列を削除したもの
通常並列化版　　：オリジナル版に忠実に並列化したもの
ワーク領域削除並列化版　：オリジナル版からベクトル用ワーク領域配列を削除し並列化したもの

Table. 3.2 Execution time.

CASE	シングル	4並列	16並列
オリジナル	4534 Sec	——	——
ワーク領域削除ベクトル版	3728 Sec	——	——
通常並列化版	4311 Sec	2531 Sec	2235 Sec
ワーク領域削除並列化版	4247 Sec	2471 Sec	2414 Sec

ワーク領域削除ベクトル版：オリジナル版からベクトル用ワーク領域配列を削除したもの
通常並列化版　　：オリジナル版に忠実に並列化したもの
ワーク領域削除並列化版　：オリジナル版からベクトル用ワーク領域配列を削除し並列化したもの

```
C -----------------------------------------------------------------------
C
C     FOR VECTORIZATION IN VPP500
C
C       ( Original )
C
C
      DIMENSION ZAA1(NT,NT,3), ZRR2(NT,NT), ZEDQF(NT,NT)
C =======================================================================
```

Fig. 3.1 Work area definition of vector processing before modification.

```
C -----------------------------------------------------------------------
C
C     FOR VECTORIZATION IN VPP500
C
C       ( Change on 1998'05 at FJT )
C
C
      real*8 ZAA1, ZAA2, ZAA3, ZEDQF
C =======================================================================
```

Fig. 3.2 Work area definition of vector processing after modification.

```
C
C     ADDITIONAL CALCULATION TERM FOR HIGH-LEVEL VECTORIZATION
C
      DO  7000  K = 1, 3
        DO  7000  J = 1, NT
          DO  7000  I = 1, NT
            ZAA1(I,J,K) = Q(I,K) - Q(J,K)
            IF( ZAA1(I,J,K) .GT. AR ) ZAA1(I,J,K) = ZAA1(I,J,K) - AL
            IF( ZAA1(I,J,K) .LT. AS ) ZAA1(I,J,K) = ZAA1(I,J,K) + AL
 7000 CONTINUE
      DO  7001  J = 1, NT
        DO  7001  I = 1, NT
          ZRR2(I,J) =  ZAA1(I,J,1)*ZAA1(I,J,1)
     &               + ZAA1(I,J,2)*ZAA1(I,J,2)
     &               + ZAA1(I,J,3)*ZAA1(I,J,3)
 7001 CONTINUE
C ---------------------------------------------------------------------
C
C   <<<<<<<< CALCULATION OF FORCE BASED ON POTENTIAL >>>>>>>>
C
C     ALMOST CPU TIME IN CALCULATION MUST BE USED FOR THIS LOOP.
C     BE CAREFUL, IF NEW CALCULATION IS ADDED IN THIS LOOP!
C
      EN2=0.0
      DO  7010  J = 1, NT
        JON = KIM(J)
        DO  7010  I = 1, NT
          ION = KIM(I)
          MF  = IPR(ION,JON)

          RR2 = ZRR2(I,J)
          IF( (RR2.LE.AR2) .AND. (I.NE.J) ) THEN
             R           = SQRT(RR2)
             N           = 100.0*R
             NH          = N/2+1
C         ------------------------------------------------
C          CALCULATION OF JFR YIELDS POOR VECTORIZATION
C          (NO CALCULATION OF JFR IS RECOMMENDED, IF POSSIBLE)
C         ------------------------------------------------
            JFR(NH,MF)  = JFR(NH,MF) + 1
C         ------------------------------------------------
             EN2         = EN2 + PE(N,MF)
             RFA         = R - 0.01*REAL(N)
             ZEDQF(I,J)  = 1.0E-04*(RF(N,MF)
     &                        + 100.0*(RF(N+1,MF)-RF(N,MF))*RFA)
          ELSE
             ZEDQF(I,J) = 0.0
          ENDIF

 7010 CONTINUE
C ---------------------------------------------------------------------
C
C     CALCULATION OF FORCE TTF(J,K)
C
      DO  7020  K = 1, 3
        DO  7020  J = 1, NT
          DO  7020  I = 1, NT
            TTF(J,K) = TTF(J,K) - ZEDQF(I,J)*ZAA1(I,J,K)
 7020 CONTINUE
C ---------------------------------------------------------------------
C
C     CALCULATION OF POTENTIAL PART OF STRESS TENSOR
C
      DO  7030  J = 1, NT
        DO  7030  I = 1, NT
          TFI(J)   = TFI(J)   + ZEDQF(I,J)*ZAA1(I,J,1)*ZAA1(I,J,2)
 7030 CONTINUE
C ---------------------------------------------------------------------
```

Fig. 3.3 Computational processing part in the original code.

```
C
C     ADDITIONAL CALCULATION TERM FOR HIGH-LEVEL VECTORIZATION
C
        EN2=0.0
        DO  7000  J = 1, NT
          JON = KIM(J)
          DO  7000  I = 1, NT
            ION = KIM(I)
            ZAA1 = Q(I,1) - Q(J,1)
            ZAA2 = Q(I,2) - Q(J,2)
            ZAA3 = Q(I,3) - Q(J,3)
            IF( ZAA1 .GT. AR ) ZAA1 = ZAA1 - AL
            IF( ZAA1 .LT. AS ) ZAA1 = ZAA1 + AL
            IF( ZAA2 .GT. AR ) ZAA2 = ZAA2 - AL
            IF( ZAA2 .LT. AS ) ZAA2 = ZAA2 + AL
            IF( ZAA3 .GT. AR ) ZAA3 = ZAA3 - AL
            IF( ZAA3 .LT. AS ) ZAA3 = ZAA3 + AL
C ---------------------------------------------------------------------
C
C   <<<<<<<< CALCULATION OF FORCE BASED ON POTENTIAL >>>>>>>>
C
C     ALMOST CPU TIME IN CALCULATION MUST BE USED FOR THIS LOOP.
C     BE CAREFUL, IF NEW CALCULATION IS ADDED IN THIS LOOP!
C
            MF  = IPR(ION,JON)
            RR2 =  ZAA1*ZAA1
     &           + ZAA2*ZAA2
     &           + ZAA3*ZAA3
            IF( (RR2.LE.AR2) .AND. (I.NE.J) ) THEN
               R           = SQRT(RR2)
               N           = 100.0*R
               NH          = N/2+1
C           ------------------------------------------------
C           CALCULATION OF JFR YIELDS POOR VECTORIZATION
C           (NO CALCULATION OF JFR IS RECOMMENDED, IF POSSIBLE)
C           ------------------------------------------------
               JFR(NH,MF)  = JFR(NH,MF) + 1
C           ------------------------------------------------
               EN2         = EN2 + PE(N,MF)
               RFA         = R - 0.01*REAL(N)
               ZEDQF  = 1.0E-04*(RF(N,MF)
     &                  + 100.0*(RF(N+1,MF)-RF(N,MF))*RFA)
            ELSE
               ZEDQF = 0.0
            ENDIF
C ---------------------------------------------------------------------
C
C     CALCULATION OF FORCE TTF(J,K)
C
            TTF(J,1) = TTF(J,1) - ZEDQF*ZAA1
            TTF(J,2) = TTF(J,2) - ZEDQF*ZAA2
            TTF(J,3) = TTF(J,3) - ZEDQF*ZAA3
C ---------------------------------------------------------------------
C
C     CALCULATION OF POTENTIAL PART OF STRESS TENSOR
C
            TFI(J)   = TFI(J)   + ZEDQF*ZAA1*ZAA2
C ---------------------------------------------------------------------
 7000 CONTINUE
```

Fig. 3.4 Modification of computational processing with work area reduction.

```
C
C     For Parallel System
C
      dimension PworkG(NT,3)
      dimension Qwork(NT,3),KIMwork(NT)
      parameter(iipe=4)
!xocl processor pen(iipe)
!xocl index partition qn=(proc=pen,index=1:NT,part=band)
!xocl local P(/qn,:),Q(/qn,:)
!xocl local SI(/qn),CO(/qn),XSI(/qn),XCO(/qn),TTF(/qn,:)
!xocl local O(/qn,:),P2(/qn,:),Q2(/qn,:),O2(/qn,:),PQO(/qn,:),OQ2(/qn)
!xocl local KIM(/qn),CX(/qn),TFI(/qn),VJ(/qn,:)
!xocl global PworkG
      equivalence( PworkG,P )
!xocl parallel region
```

Fig. 3.5 Definition of data decomposition.

```
C
C     READING INITIAL 2 STEPS POSITION DATA
C
      READ(5,6000) (P(I,1),P(I,2),P(I,3),I=1,NT)
      READ(5,6000) (Q(I,1),Q(I,2),Q(I,3),I=1,NT)
 6000 FORMAT(3F10.6)
```

Fig. 3.6 Data input part before modification.

```
C
C     READING INITIAL 2 STEPS POSITION DATA
C
      READ(5,6000) (Qwork(I,1),Qwork(I,2),Qwork(I,3),I=1,NT)
      do 301 K=1, 3
!xocl spread do
        do 300 I=1, NT
          P(I,K) = Qwork(I,K)
  300   continue
!xocl end spread
  301 continue
      READ(5,6000) (Qwork(I,1),Qwork(I,2),Qwork(I,3),I=1,NT)
      do 303 K=1, 3
!xocl spread do
        do 302 I=1, NT
          Q(I,K) = Qwork(I,K)
  302   continue
!xocl end spread
  303 continue
 6000 FORMAT(3F10.6)
 6100 FORMAT(3F12.6)
```

Fig. 3.7 Data input part after modification.

```
                .
                .
                .
C     For Parallel System
      dimension PworkG(NT,3)
      dimension Qwork(NT,3),KIMwork(NT)
                .
                .
!xocl local P(/qn,:),Q(/qn,:)
                .
                .
!xocl global PworkG
      equivalence( PworkG,P )
                .
                .
                .
C
C     OUTPUT LAST 2 STEPS POSITION DATA FOR NEXT CALCULATION
C
      WRITE(6,6000) (PworkG(I,1),PworkG(I,2),PworkG(I,3),I=1,NT)
      WRITE(6,6000) (Qwork (I,1),Qwork (I,2),Qwork (I,3),I=1,NT)
                .
                .
```

Fig. 3.8 Definition of local array P,Q and global array PworkG.

```
 5.883139 17.076926 11.581665
10.121720 18.333595 16.891994
21.176637 24.963134 24.182932
30.446091 28.719292 30.769702
 9.582691 20.227540 30.085936
 0.272213  9.382243 20.497287
27.227571 17.681127 31.554651
22.243773  6.981586  1.398672
 8.680054 19.668740  8.089915
23.013612 22.612938 19.641619
19.478874  3.154547  7.250516
18.831358 29.008466 27.694715
29.884066 30.737823 12.981803
 2.829333 16.920986  3.298173
15.773248 24.541495  3.701071
23.932953  1.516860 19.239047
26.430788 12.193250  7.008216
19.451192  4.696381 24.179286
12.640793 13.160906 18.224910
 5.927193 28.665126 19.773937
16.052361 23.122608 24.258376
21.932517  2.428114 28.478960
17.812561 30.661276  1.493642
 9.137686 19.054388 24.699984
 1.144863 19.087444 22.493145
 3.087538 10.222280 25.108852
28.607477  5.478625 17.606963
29.661870 24.110372 22.820704
12.452219 23.945487 15.202658
 8.187759  0.292600 13.059474
 5.874144 25.937073 25.751506
13.125851 11.905211 22.648094
 3.066548 28.682880 30.162506
 3.604331 31.070277  8.933130
 1.648675 29.489626 17.623590
 8.120716  2.473909  8.048820
11.002270  1.966632 29.258289
28.079895 16.891997  4.899420
18.654530  5.611430 14.947720
22.020820 22.895537  6.158743
14.862740  9.730548 13.268793
25.257435  1.517861 24.053356
21.980828 12.277661 28.305584
26.669602 26.610360  2.012759
 6.812354 15.527958 29.315009
30.596384  7.937097 28.143924
23.667189 16.599570 26.722335
27.205269  3.814889 27.205666
27.572375 18.039777 14.095183
 1.445954 15.257381 26.536125
```

Fig. 3.9 Data of array P (original).

```
 1.134205 26.569138 29.410554
 5.887905 17.085445 11.580799
10.124640 18.337023 16.895199
21.182327 24.961343 24.176292
30.441910 28.720259 30.774043
 9.585535 20.233572 30.084738
 0.269385  9.377488 20.492568
27.232115 17.682006 31.548635
22.244842  6.978823  1.396264
 8.680111 19.665332  8.087426
23.011136 22.615125 19.638515
19.473517  3.151122  7.252215
18.833561 28.999640 27.696219
29.881046 30.736634 12.974140
 2.828035 16.925467  3.302074
15.777839 24.541082  3.712663
23.929392  1.520080 19.230191
26.428139 12.189655  7.012061
19.450538  4.690820 24.177526
12.633672 13.167444 18.220325
 5.924950 28.667251 19.765111
16.046548 23.120983 24.258158
21.928582  2.431275 28.477860
17.812480 30.660606  1.482925
 9.141161 19.051351 24.703621
 1.143737 19.092242 22.498298
 3.092747 10.220421 25.107027
28.613139  5.477586 17.597974
29.659887 24.113980 22.825576
12.448982 23.942416 15.204389
 8.184466  0.291168 13.056592
 5.868109 25.938284 25.751850
13.126437 11.898115 22.642861
 3.071011 28.684391 30.156281
 3.605892 31.066557  8.926550
 1.651748 29.491746 17.617546
 8.112134  2.474204  8.048914
10.999771  1.969921 29.259121
28.074188 16.892679  4.898416
18.652597  5.609704 14.949723
22.020359 22.902841  6.153554
14.860485  9.738047 13.266947
25.253934  1.507876 24.056438
21.977892 12.282784 28.304474
26.670361 26.611611  2.013491
 6.809256 15.530368 29.316186
30.594078  7.940444 28.143264
23.665425 16.612700 26.724670
27.208959  3.815392 27.196521
27.568750 18.039306 14.099427
 1.447005 15.257693 26.536393
```

Fig. 3.10 Data of array Q (original).

2.010	0	0	0
2.030	0	0	0
2.050	0	0	0
2.070	0	0	0
2.090	0	0	0
2.110	0	0	0
2.130	0	0	0
2.150	0	10	0
2.170	0	17	0
2.190	0	64	0
2.210	0	126	0
2.230	0	279	0
2.250	0	537	0
2.270	0	908	0
2.290	0	1500	2
2.310	0	2512	2
2.330	0	3901	6
2.350	0	6036	7
2.370	0	9194	6
2.390	0	13405	7
2.410	0	18937	26
2.430	0	25887	55
2.450	0	33900	66
2.470	0	44141	145
2.490	0	55882	180
2.510	0	69649	236
2.530	0	86092	369
2.550	0	103887	586
2.570	0	122549	775
2.590	0	143915	1206
2.610	0	165819	1820
2.630	0	189514	2459
2.650	0	212429	3367
2.670	0	232785	4789
2.690	0	255619	6340
2.710	0	277102	8362
2.730	0	298601	10798
2.750	0	318136	14202
2.770	0	333857	18043
2.790	0	349052	22414
2.810	0	361293	27425
2.830	0	375811	34166
2.850	0	384536	41170
2.870	0	391829	49475
2.890	0	398413	58928
2.910	0	403620	69637
2.930	0	404964	82257
2.950	0	407656	94375
2.970	0	405605	109303
2.990	0	404434	124663
3.010	0	402495	140510

Fig. 3.11 Data of array JFR (original).

2.010	0.000	0.000	0.000
2.030	0.000	0.000	0.000
2.050	0.000	0.000	0.000
2.070	0.000	0.000	0.000
2.090	0.000	0.000	0.000
2.110	0.000	0.000	0.000
2.130	0.000	0.000	0.000
2.150	0.000	0.000	0.000
2.170	0.000	0.000	0.000
2.190	0.000	0.001	0.000
2.210	0.000	0.002	0.000
2.230	0.000	0.004	0.000
2.250	0.000	0.007	0.000
2.270	0.000	0.012	0.000
2.290	0.000	0.019	0.000
2.310	0.000	0.031	0.000
2.330	0.000	0.048	0.000
2.350	0.000	0.073	0.000
2.370	0.000	0.109	0.000
2.390	0.000	0.157	0.000
2.410	0.000	0.218	0.000
2.430	0.000	0.293	0.000
2.450	0.000	0.377	0.000
2.470	0.000	0.483	0.001
2.490	0.000	0.601	0.001
2.510	0.000	0.738	0.002
2.530	0.000	0.898	0.003
2.550	0.000	1.066	0.004
2.570	0.000	1.238	0.005
2.590	0.000	1.432	0.008
2.610	0.000	1.624	0.012
2.630	0.000	1.828	0.016
2.650	0.000	2.019	0.021
2.670	0.000	2.179	0.030
2.690	0.000	2.357	0.039
2.710	0.000	2.518	0.051
2.730	0.000	2.674	0.065
2.750	0.000	2.807	0.084
2.770	0.000	2.904	0.105
2.790	0.000	2.992	0.128
2.810	0.000	3.053	0.155
2.830	0.000	3.131	0.190
2.850	0.000	3.159	0.226
2.870	0.000	3.174	0.268
2.890	0.000	3.183	0.314
2.910	0.000	3.181	0.366
2.930	0.000	3.148	0.427
2.950	0.000	3.126	0.483
2.970	0.000	3.068	0.552
2.990	0.000	3.019	0.621
3.010	0.000	2.965	0.691

Fig. 3.12 Data of array FR (original).

```
 6.228324 20.525846 16.949108
12.005896 17.673276 18.851903
18.365995 25.260083 22.619631
28.741966 29.909461 26.029236
15.631125 23.363995  0.797083
29.137602  5.108498 20.350298
23.920452 17.635451 28.288315
18.928345  8.047696 30.924438
14.078346 21.984603  9.993005
24.712876 18.916353 16.048909
16.400966  6.914035  5.269176
18.881114  0.371203 21.597906
26.182656 24.908639 10.976471
 2.771479 16.948208  3.483252
19.813062 26.965269 30.619775
25.427012 28.934789 22.746142
28.252193  9.676361 13.479410
19.284439  5.624243 22.919359
12.470382 12.280868 20.941917
 7.689948 26.261332 19.937670
15.592448 22.511362 17.984180
21.258535  5.582844 27.712939
17.192936  4.017021  0.331483
12.994708 18.923804 30.607888
 5.431471 11.961653 24.535524
 4.152973  9.817661 29.199088
24.334159  0.952781 18.929227
 0.528660 20.934147 27.143823
14.189731 27.224759 13.002823
 5.277368 28.559964  9.717367
 8.227161 26.965071 26.065264
17.705484 10.908146 20.703863
 0.744526  0.078790 29.429627
 4.419221 24.860525  1.945378
 1.603311 27.147074 15.352505
 7.711525 27.921853  5.046979
 9.633856  8.237517 25.150199
24.636475 14.811730  5.243552
16.694693  1.651402 15.669698
22.173237 26.526126  6.446118
17.431729 12.123809 14.974882
23.744402  1.238633 25.232657
28.456111 10.194436 29.663730
23.660528 25.853708  1.827561
 6.374352 12.967478  2.023933
31.133486  5.551906 28.974019
27.724336 14.299854 26.513106
27.583161  2.074908 29.481138
28.593882 14.273634 18.077289
 1.076542 13.034147 30.513467
```

Fig. 3.13 Data of array P (4PE parallel).

```
 8.818872 16.470164 26.883142
 6.226679 20.524190 16.949705
12.009409 17.674954 18.844263
18.376047 25.257231 22.616646
28.749170 29.907497 26.034992
15.632308 23.364954  0.801560
29.140116  5.112204 20.349681
23.917125 17.644435 28.288155
18.929170  8.045674 30.922941
14.086522 21.985026 10.002379
24.712707 18.915735 16.047535
16.391252  6.921198  5.272912
18.881534  0.369741 21.594037
26.182392 24.905087 10.969234
 2.768086 16.949127  3.482080
19.817474 26.969900 30.616094
25.427133 28.930753 22.748829
28.249416  9.682359 13.477636
19.280778  5.622216 22.918019
12.469416 12.285064 20.945694
 7.693128 26.263183 19.934812
15.593391 22.507226 17.984414
21.260569  5.575220 27.713896
17.191709  4.016642  0.327769
12.993372 18.927457 30.605676
 5.432707 11.960238 24.542426
 4.152401  9.816088 29.205224
24.336644  0.952476 18.927062
 0.526823 20.932650 27.143288
14.189349 27.224451 12.998943
 5.269599 28.556597  9.718917
 8.231539 26.971515 26.059886
17.711941 10.904907 20.702894
 0.740885  0.075359 29.429561
 4.417444 24.857868  1.947784
 1.601742 27.147995 15.357548
 7.714663 27.926612  5.048447
 9.637914  8.236967 25.156102
24.636108 14.818036  5.249092
16.694342  1.643248 15.665581
22.176780 26.528485  6.449447
17.435940 12.123235 14.973705
23.743430  1.235440 25.234852
28.450931 10.189939 29.672672
23.658653 25.855193  1.826523
 6.379458 12.961526  2.026143
31.136912  5.548873 28.971459
27.719902 14.300542 26.511857
27.587106  2.078366 29.479390
28.591081 14.277178 18.078740
 1.078137 13.031075 30.510572
```

Fig. 3.14 Data of array Q (4PE parallel).

2.010	0	0	0
2.030	0	0	0
2.050	0	0	0
2.070	0	0	0
2.090	0	0	0
2.110	0	5	0
2.130	0	3	0
2.150	0	17	0
2.170	0	22	0
2.190	0	61	0
2.210	0	134	0
2.230	0	272	0
2.250	0	568	0
2.270	0	994	0
2.290	0	1577	3
2.310	0	2630	2
2.330	0	4041	2
2.350	0	6446	1
2.370	0	9584	2
2.390	0	13704	8
2.410	0	19218	10
2.430	0	26295	43
2.450	0	35030	64
2.470	0	45401	104
2.490	0	57491	185
2.510	0	71395	263
2.530	0	87659	374
2.550	0	105262	526
2.570	0	124806	819
2.590	0	145681	1189
2.610	0	167474	1702
2.630	0	189495	2532
2.650	0	211625	3476
2.670	0	233788	4702
2.690	0	254876	6180
2.710	0	276695	8140
2.730	0	299182	10661
2.750	0	317898	13522
2.770	0	334293	17223
2.790	0	350749	22488
2.810	0	362774	27826
2.830	0	374713	34613
2.850	0	384839	41498
2.870	0	393377	48895
2.890	0	399602	58612
2.910	0	403795	68963
2.930	0	406225	80991
2.950	0	408654	93685
2.970	0	407868	108040
2.990	0	406079	122133
3.010	0	403498	139293

Fig. 3.15 Data of array JFR (4PE parallel).

2.010	0.000	0.000	0.000
2.030	0.000	0.000	0.000
2.050	0.000	0.000	0.000
2.070	0.000	0.000	0.000
2.090	0.000	0.000	0.000
2.110	0.000	0.000	0.000
2.130	0.000	0.000	0.000
2.150	0.000	0.000	0.000
2.170	0.000	0.000	0.000
2.190	0.000	0.001	0.000
2.210	0.000	0.002	0.000
2.230	0.000	0.004	0.000
2.250	0.000	0.007	0.000
2.270	0.000	0.013	0.000
2.290	0.000	0.020	0.000
2.310	0.000	0.033	0.000
2.330	0.000	0.050	0.000
2.350	0.000	0.078	0.000
2.370	0.000	0.114	0.000
2.390	0.000	0.160	0.000
2.410	0.000	0.221	0.000
2.430	0.000	0.297	0.000
2.450	0.000	0.389	0.000
2.470	0.000	0.497	0.001
2.490	0.000	0.619	0.001
2.510	0.000	0.756	0.002
2.530	0.000	0.914	0.003
2.550	0.000	1.080	0.004
2.570	0.000	1.261	0.006
2.590	0.000	1.449	0.008
2.610	0.000	1.641	0.011
2.630	0.000	1.828	0.016
2.650	0.000	2.011	0.022
2.670	0.000	2.188	0.029
2.690	0.000	2.350	0.038
2.710	0.000	2.514	0.049
2.730	0.000	2.679	0.064
2.750	0.000	2.805	0.080
2.770	0.000	2.907	0.100
2.790	0.000	3.007	0.129
2.810	0.000	3.066	0.157
2.830	0.000	3.122	0.193
2.850	0.000	3.162	0.228
2.870	0.000	3.187	0.264
2.890	0.000	3.193	0.313
2.910	0.000	3.182	0.363
2.930	0.000	3.158	0.420
2.950	0.000	3.134	0.480
2.970	0.000	3.086	0.546
2.990	0.000	3.031	0.609
3.010	0.000	2.972	0.685

Fig. 3.16 Data of array FR (4PE parallel).

参 考 文 献

[1] 岡本芳浩・横川三津夫・小川徹 ; 分子動力学法による溶融塩の粘性率の計算, JAERI-M 93-242, 1993 年 12 月.

[2] 岡本芳浩, 小川徹; 分子動力学法による核燃料物質の物性推定, 私信, 1995 年 8 月.

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

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

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

4. EDDYCALコードのベクトル並列化

本章は，主に富士通製分散メモリ型ベクトル並列計算機 VPP500/42（以下，VPP）上で行なった新EDDYCALコードのベクトル化・並列化作業の報告である．また，VPP上のメモリ量の都合から，富士通製共用 UNIX サーバ AP3000/24（分散メモリ型スカラ並列計算機（以下，AP3000））への移植作業も行なったため，これについても述べる．

当初，以前（平成 8 年 3 月）にベクトル化を行なった EDDYCAL コード（旧版）を基にVPP 上で並列化作業を行なう予定であったが，ユーザ側が新 EDDYCAL コードを開発したため，これについて作業を行なうことになった．このコードは，主に入力データを作成するコード，主計算を行なうコード，その結果を図形表示するコードから成るが，高速化の対象となったのは，主計算を行なうコードである（以下，オリジナルコードと呼ぶ）．

ベクトル化については，オリジナルコードが既にベクトル化済であったことから，更にベクトル化が可能な部分のみを選んで作業を行なった．並列化については，ユーザ側が 1.6GB 以上のメモリ量を希望していたため，1PE(PE:Processing Element) 当たりのメモリ使用量削減を目的にしたデータ分割を中心に並列化を検討した．しかし検討の結果，プロセス間データ転送が多頻度に発生し，実行速度の極端な劣化が予想されたため，処理分割のみの並列化作業を行なった．

AP3000 への移植作業は，AP3000 がメモリを多く使えることから，VPP 上で使えるメモリ量（200MB 弱 /1PE）がユーザの希望量より少ないことの理由で行なった作業である（AP3000:2GB/1PE）．最終的なコードは，VPP 上，AP3000 上共に同じものであり，両計算機で並列実行で使用できるようになった．

4.1 コード概要

EDDYCAL コード [1] ～ [4] は，有限要素法を用いて渦電流の固有値解析を行なうとともに，ビオサバールの式を用いて外部磁場との結合を求め，渦電流の時間変化を求める．具体的には，有限要素法により導体をモデル化し，薄板近似により電流ポテンシャルを導入し，これらの電流ポテンシャルの励起モードを固有値解析により求める．電流ポテンシャル法による電磁場解析においては，インダクタンス行列中に非局所的な境界積分項が現れ，行列が実対称となることが特徴として挙げられる．

コードの構造としては，大きく 3 つのステップに分かれる．ステップ 1 は，回路方程式の組み立てから固有値計算まで，ステップ 2 は，外部磁場，ベクトルポテンシャルの計算から渦電流の計算まで，ステップ 3 は，外部磁場，磁束密度の計算から磁束密度，電磁力の計算までである．これらの内，計算時間については，ステップ 1 の行列計算の部分が支配的になる．

4.2 オリジナルコードのコスト解析

オリジナルコードをベクトル実行した場合，どの部分の計算コストが高いかを把握するため，サブルーチン毎の実行時間の測定を行なった．測定の対象とするサブルーチンは，全サブルーチンではなく，処理単位毎にまとまったルーチン群の最上位ルーチンとした．この際，最も適当なサブルーチンがメインルーチンから呼び出されているサブルーチン main1 である．main1 内には測定対象となる最上位ルーチンの呼び出し部が存在するからである．そこで，まずこのルーチンにおいて測定を行ない，計算コストの高いルーチンを親ルーチンから徐々に絞り込んでいく方法でコストの把握を行なった．以下に全体の実行時間とサブルーチン main1 から呼び出しているコストの高いサブルーチンの実行時間を示す．

尚，測定は，実行時の CPU 占有時間を取得できるサービスルーチン clock をサブルーチン呼び出しの前後で用い，それらの差を実行時間とした．

全体	：1400sec
サブルーチン MKINDUCT	：1100sec
サブルーチン EIGEN	：159sec

上の２つのサブルーチン MKINDUCT と EIGEN で，全体の約 90% の実行時間を占めていることが分った．また，MKINDUCT 内の子ルーチンの実行時間を測定したところ，1100sec の大部分がその子ルーチンである INDUCT に費されていることがわかった．一方，EIGEN 内では，その子ルーチンである SSPEV にその大部分が費されていることがわかった．

そこで，主作業である並列化はサブルーチン INDUCT とサブルーチン SSPEV に絞って行なうことにした．

4.3 I/O のチューニング

原研 VPP システムにおいて，VPP（バックエンドシステム）で実行中のジョブ I/O は，通常 GSP（フロントエンドシステム）のディスクに対してネットワーク経由（NFS）によって行なわれる．そのため，I/O 時間が予想以上にかかり，それがジョブの経過時間（実時間であり，CPU 時間とは別）に影響する場合がある．これを解決するために，VPP システムでは，NFS の変わりに SCFS が用意されている．また，システム標準設定（32KB）より大きい FORTRAN I/O バッファサイズの指定も可能である（利用法は参考文献 [5] を参照されたい）．ジョブの実行開始から終了までの時間は，CPU 時間ではなく，経過時間であるため，これを短縮することは重要である．ここで，本コード実行時に，これら I/O のチューニングを行なった場合と行なわない場合の経過時間を以下に示す．

チューニング後の I/O	：1440sec
通常の I/O	：3134sec

以上の効果があることから，本コードを実行する場合についても，文献 [5] の利用法に従った設定を行なうことにした．

4.4 ベクトル化

今回ベクトル化を行なったのは，次の 11 個のサブルーチンである．

INDUCT, FLUXG, FLUXN, AHALO, XMODEX, CRTBBB, CRTELEM, ELEMNT, INTPNT3, INTPNT, INDQA

今回のベクトル化作業では，計算コストの低い部分も含み，機械的にベクトル化可能な部分はすべてベクトル化の対象として作業を行なった．また，次の 3 つのサブルーチンに対しては，内部で呼び出しているサブルーチンや関数をインライン展開し，呼出しのオーバーヘッドを省いた．

AARC, BARC, CURRNT

4.4.1 サブルーチン INDUCT

本ルーチンには，Fig.4.1 に示すように，外側に 3 重ループがあり大部分の計算がそれに囲まれ，更にこの 3 重ループの内側にも多重ループが存在する．VPP 上でベクトルコンパイルを行なうと，これら多重ループ中でベクトル化の対象となるのは最内のループであり，本ルーチンにおいても，最内ループの大部分がコンパイラによってベクトル化されていた．しかし，ベクトル長（ループの回転数）が短いためにベクトル化効果が小さいと考えられる部分があり，この部分についてベクトル化の変更を行なった．変更前を Fig.4.2 に示す．この図において，A と B の部分が最内ループでありベクトル化がされている．しかし，A はベクトル長が 12，B は 7 であり，ベクトル化の効果を上げるには短い．そこで，図中の最外ループである C を用いたベクトル化を行なった．ここでの，IESTRT は，外側の 3 重ループによって値が，1,2,3,4,.... と変化し，N_ELMT はシミュレーション対象となるモデルの要素数であり，約 4000 以上の値になるものである．これであれば充分なベクトル長になる．

ベクトル化は，以下の a. ～ d. のように行なった．変更後を Fig.4.3 に示す．

a. Fig.4.2 のループ C 中において IF 文 D に注目し，それぞれの IF 文で真になる場合の IE2 を選び，それを新たに用意したリスト配列に格納する（Fig.4.3のループ A 内の B）．
b. a. で IF 文が真になる場合を同時にカウントする（Fig.4.3のループ A 内の C）．
c. Fig.4.3のループ D の 最内ループでは，a. で作成したリスト配列を配列 RIN に対する新たな添字とし，b. でのカウント数をループの回転数に利用する．
d. Fig.4.3のループ D の最内ループの直前に，E の最適化制御行を挿入する．これは，コンパイラが配列 RIN について回帰演算が発生する可能性があると判断し，最適化制御行によって回帰演算が無いことを指示しないとベクトル化されないためである．変更前（Fig.4.2）のループ C と IF 文 D より，配列 RIN に対する添字が必ず異なることから，回帰演算は発生しないため最適化制御行の挿入が可能である．

以上のベクトル化を行なう際に，選んだ IE2 を格納するリスト配列 i07v と i12v を用いたため，新たにそれらの配列宣言をする必要がある．Fig.4.3のループ A のカウント数は最大で，N_ELMT-IESTRT+1 であるため，配列宣言をする場合には，それぞれサイズを N_ELMT として宣

言すればよい．ただし，本コードで用いられる各配列は，メインルーチンで宣言されているかなり大きなサイズの配列（元になる配列）上のアドレスと，入力データを読み込むことで決められる配列サイズによって動的に決められている（整合配列）．そこで，i07v と i12v についても同様に動的に決めるようにした．そこで，入力データを読み込んで元の配列におけるアドレスを計算するサブルーチン INPST1A 内とその呼出し部分，INDUCT の親ルーチンである MKINDUCT 内とその呼出し部分の変更を行なった．それぞれ，INPST1A 内の変更を Fig.4.4，INPST1A の呼出し部分を Fig.4.5，MKINDUCT 内の変更を Fig.4.6，MKINDUCT の呼出し部分を Fig.4.7 に示す．

4.4.2 サブルーチン FLUXG

本ルーチンは，サブルーチン INDUCT と同じようなループ構造であった．そこで，INDUCT と同様の変更を行なった．変更前を Fig.4.8に，変更後を Fig.4.9に示す．

また，ベクトル化する際に，新たな配列 i07v と i12v を用いたため，これらの配列サイズを G_GRID ととして宣言し，INDUCT の場合と同様に動的に決めるようにした．そこで，入力データを読み込んで元の配列におけるアドレスを計算するサブルーチン INPST3A 内とその呼出し部分，FLUXG の親ルーチンである EDCMAG 内とその呼出し部分の変更を行なった．それぞれ，INPST3A 内の変更を Fig.4.10，INPST3A の呼出し部分を Fig.4.11，EDCMAG 内の変更を Fig.4.12，EDCMAG の呼出し部分を Fig.4.13 に示す．

4.4.3 サブルーチン FLUXN

本ルーチンも，FLUXG と同様にベクトル化の変更を行なった．変更前を Fig.4.14 に，変更後を Fig.4.15 に示す．また同様に，新たな配列 i07v と i12v を，配列サイズを N_ELMT として宣言し，同様に動的に決めるようにした．入力データを読み込んで元の配列におけるアドレスを計算するルーチンは，同様に INPST3A であり，FLUXN の親ルーチンも同様に EDCMAG であった（変更部分は Fig.4.10～ Fig.4.13参照）．

4.4.4 サブルーチン AHALO

本ルーチンは，シミュレーションの対象となるモデルの各要素を構成するノード数で回転するループのみが存在し，大部分の処理がそれに囲まれる．このループでは，ループ内においてサブルーチンを呼び出していること，そのサブルーチンに渡している実引数において，コンパイラが回帰演算が発生する可能性があると判断していることから，ベクトル化されていなかった．そこで，呼び出しているサブルーチンのインライン展開を行なってベクトル化を行なうことにした．

変更前を Fig.4.16 に示す．また，Fig.4.16 の A で呼び出しているサブルーチン INTGS を Fig.4.17 に示す．Fig.4.17 の B で用いているユーザ関数 FNC は INTGS の仮引数 FNC によって決まるが，AHALO では FNAZ，FNARX を実引数で渡している．この FNAZ と FNARX を Fig.4.18 に示す．

ベクトル化は以下の a. ～ c. のように行なった．

a. サブルーチン INTGS を AHALO へインライン展開した場合，最内になる INTGS 内のループがベクトル化の対象となり，充分なベクトル長であるノード数でのベクトル化がされない．

しかし，INTGS 内のループは，ルーチンの前半部分で求められる K の値が回転数になり，更に K を決める際に参照される N は，AHALO において 16 と固定であることがわかる．そこで，このループを K を 16 として展開した．これを Fig.4.19 に示す．

b. サブルーチン INTGS 内では仮引数 FNC を用いて，関数 FNAZ か FNARX を呼び出す．これは実行時に動的に決められる．しかし，ベクトル化はコンパイラにより実行前に静的に決まるものであるため，呼び出す関数が静的に決まっていないとインライン展開することはできない．そのため，陽に FNAZ や FNARX の名前で呼び出されていなければインライン展開されない．そこで，a. で改良した INTGS 内を FNAZ と FNARX に変更した．ただし，AHALO で FNAZ を使用する場合と FNARX を使用する場合とを分けなければならないため，AHALO から FNAZ を INTGS に渡した時は実引数にフラグ 1 を追加し，FNARX を INTGS に渡した時は実引数にフラグ 2 を追加することにした．そして，INTGS 内ではそのフラグと IF 文を用いて，FNAZ を使用する場合と FNARX を使用する場合とに分けることにした．AHALO の INTGS の呼び出しの変更を Fig.4.20 に，INTGS の変更を Fig.4.21 に示す．

c. a. と b. の変更をしただけでは，AHALO と INTGS をコンパイル対象にした場合と，INTGS と FNAZ，FNARX をコンパイル対象にした場合は，それぞれインライン展開が可能であったが，AHALO と INTGS，FNAZ，FNARX をすべてコンパイル対象にするとインライン展開が不可能であった．そこで，INTGS から呼び出している関数 FNAZ と FNARX を，INTGS の宣言部で文関数として宣言し，呼び出されていた FNAZ と FNARX はコメント化した．文関数の宣言を Fig.4.22 に示す．

以上で，サブルーチン AHALO のベクトル化が可能になった．ベクトル化後を Fig.4.23 に示す．

4.4.5 サブルーチン XMODEX

本ルーチンにおいても，ループ内でユーザ関数が呼ばれているため，ベクトル化されていなかった．よって，インライン展開によりベクトル化を行なった．ベクトル化前の XMODEX 内のループを Fig.4.24 に示す．また，ループ内で呼ばれている関数 XINT を Fig.4.25 に，XINT2 を Fig.4.26 に示す．ここで，XINT では分岐処理が IF ～ GO TO 文によって行なわれている．これではインライン展開を行なってもベクトル化されないため，IF ～ GO TO 文構造を IF 文構造に変更してベクトル化を可能にした．IF 文構造に変更した XINT を Fig.4.27 に，また，ベクトル化後の XMODEX を Fig.4.28 に示す．

4.4.6 サブルーチン CRTBBB

本ルーチンでは，回帰演算が発生する可能性があるというコンパイラの判断により，ベクトル化されていない部分があった．これを Fig.4.29 に示す．コンパイラは Fig.4.29 中の配列 BNODALL について回帰演算が発生する可能性があると判断していたが，図中の A と B における添字の計算から，定義される BNODALL と参照される BNODALL は別の領域であり，回帰演算は発生しないことが分る．よって，回帰演算が発生しないことをコンパイラに知らせる最適化制御行を挿入することによってベクトル化を可能にした．これを Fig.4.30 に示す．

4.4.7 サブルーチン CRTELEM

本ルーチンにおいても，回帰演算が発生する可能性があるというコンパイラの判断により，ベクトル化されていない部分があった．これを Fig.4.31 に示す．コンパイラは DO 30 ループにおいては配列 XYZ_NODE と L_NODE，DO 40 ループにおいては配列 L_ELMT と MAT_ELMT，AREA_ELMT，NODE_ELMT について回帰演算が発生する可能性があると判断していた．しかし，図中の A と B における添字の計算より，それぞれのループで定義される領域と参照する領域が異なることから，回帰演算が発生しないことがわかる．よって，回帰演算が発生しないことをコンパイラに知らせる最適化制御行を両ループの直前に挿入することによって，ベクトル化を可能にした．これを Fig.4.32 に示す．

4.4.8 サブルーチン ELEMNT

本ルーチンには回転数が N_ELMT のループとその内側にそれぞれ回転数が 3 の 2 重ループが存在する．これらはコンパイラにより，外側の DO 変数が N のループと内側の DO 変数が K のループが入れ替わり，外側の DO 変数が N のループでベクトル化されていた．これを Fig.4.33 に示す．しかし，ここではすべて N で回転するループでベクトル化できるように，内側のループをすべて展開して変更を行なった．これを Fig.4.34 に示す．

4.4.9 サブルーチン INTPNT3

本ルーチン内のループでは，回帰演算が発生する可能性があるというコンパイラの判断により，ベクトル化されていない部分があった．このループを Fig.4.35 に示す．ここでは，配列 XYZ_GAUS について回帰演算が発生する可能性があると判断していた．しかし，Fig.4.35 中の A の IRS の計算と B の IF 文，C での IRS を用いた配列 XYZ_GAUS の添字計算から，ループ内では XYZ_GAUS の定義領域と参照領域が異なることが分る．よって，最適化制御行を挿入することによりベクトル化を可能にした．これを Fig.4.36 に示す．

4.4.10 サブルーチン INTPNT

本ルーチンには，外側に N_ELMT で回転するループ，その内側にベクトル化効果を上げるにはあまり充分ではないベクトル長を持つループが存在していた．これを Fig.4.37 に示す．ベクトル化は，ベクトル長が充分に得られる外側の N_ELMT で回転するループを利用することにした．この際，内側のループ内の回転数の少ない部分はすべて展開した．これを Fig.4.38 に示す．

4.4.11 サブルーチン INDQA

本ルーチンも，多重ループの内側にベクトル化効果を上げるにはあまり充分ではないベクトル長を持つループが存在しており，またベクトル化不可能な原因である GO TO 文も存在していた．これを Fig.4.39 に示す．本ルーチンにおいては，Fig.4.39 中のループ A を対象にベクトル化を検討した．単純には，このループの内側のループ回転数が少ないため，すべて展開すればよいが，ベクトル化不可能な原因になる GO TO 文が存在している．

まず，ループ A より内側のループをすべて展開した後のループを Fig.4.40に示す．ここで Fig.4.40 中の A，B，C はそれぞれ Fig.4.41 に示すように表すことができ，これを Fig.4.40

の A', B', C' にそれぞれ代入することができる．そして，Fig.4.40 中の GO TO 文による処理構造は，Fig.4.42 に示す IF 文による処理構造に変更することができる．この2つの変更を施すことにより，元のループ A のベクトル化を可能にした．ベクトル化後のループ A を Fig.4.43 に示す．

4.4.12 サブルーチン AARC, BARC, CURRNT

これらのルーチン内では，回転数の多いループから下位サブルーチンを呼び出している．このような場合，呼び出しのオーバーヘッドがかかり，実行時間に影響する場合がある．そこで，下位サブルーチンのインライン展開が可能であったことから，これを行なって呼び出しのオーバーヘッドを省いた．これらのインライン展開は，AARC, BARC, CURRNT が格納されているファイルへ，それぞれ呼び出されているルーチンをコピーするだけで済んだため，ここでは特に図示しない．

4.5 並列化

並列化を行なった部分は，インダクタンス行列作成部分（サブルーチン INDUCT）と，標準固有値問題解法部分（サブルーチン SSPEV），標準固有値問題で解いた固有ベクトルから一般固有値問題の固有ベクトルへの変換部分（サブルーチン DTPSV）である．その他の部分は，計算コストが低いか，またはベクトル化による高速実行が実現されているため，並列化をすると逆に並列化コーディングが原因で実行時間が増えると考えられる部分であり，並列化は必要ない．

今回の作業では，ソースに挿入する並列化命令として，MPI(Message Passing Interface) ライブラリ [6] を使用して並列化コーディングを行なった．VPP, AP3000 共に用意されている並列化命令は，富士通製並列計算機のみで動作する命令（VPP FORTRAN, 並列 Fortran/AP）と MPI がある．並列化指示行を挿入して並列化した場合には，VPP, AP3000 で実行可能であるが，今後将来，計算機のリプレース等があった場合は，並列化コードの移植作業（並列化命令の変換作業）が発生してしまう．よって，現在，各メーカーの並列計算機上に標準で搭載されている MPI ライブラリを用いて並列化を行なうことにした．EDDYCAL の並列化で利用した MPI ライブラリを付録 Aに示す．

4.5.1 サブルーチン INDUCT

本ルーチンは，最も計算コストの高い部分である．並列化は，データ分割が困難だったことから処理分割のみを行なった．

(1) 並列化の方針

ここでは，INDUCT の処理分割の方針と，データを分割しなかった理由等について述べる．

・処理分割について

サブルーチン INDUCT 内は，外側に 3 重ループがあり大部分の計算がここで行なわれる (Fig.4.44 参照)．この内の最内ループ A はシミュレーションの対象となるモデルの要素数で回転するループであり，ループ中の処理は要素毎に独立している．一方，外側の 2 つのループは分割するには回転数が少ない．よって，ループ A を処理分割の対象とした．

ループ分割の方法は，通常，均等分割かサイクリック分割が用いられる．
例えば，1 ～ 8 まで回転するループを 4 つのプロセスに分割した場合，

1 番目のプロセス：1,2
2 番目のプロセス：3,4
3 番目のプロセス：5,6
4 番目のプロセス：7,8

のように回転範囲が配分される場合は均等分割であり，

1 番目のプロセス：1,5
2 番目のプロセス：2,6
3 番目のプロセス：3,7
4 番目のプロセス：4,8

のように回転範囲が配分される場合はサイクリック分割である．ループ A の分割の方法は，次の 2 つの理由によりサイクリック分割とした．

a. ループ分割する場合は，各プロセスの担当する回転範囲を，計算負荷が均等になるように予め計算する（別にコーディングする）必要があるが，サイクリック分割を行なうと，オリジナルのループ A を次のように変更するだけで計算負荷が均等になる．

DO IE1=1,N_ELMT → DO IE1=1+my_pe,N_ELMT,NPE
※ my_pe：ランク． NPE：プロセス数．

よって，回転範囲を計算するコーディングとこの部分の実行時間を省くことができる．

b. ループ A が DO IE1=1,N_ELMT であり，その中のループの大部分が DO IE2=IE1,N_ELMT であるため，ループ A を均等分割するとその中のループの回転数の均等（計算負荷の均等）がとれないが，サイクリック分割すれば各プロセスへの負荷分散が均等化される．

・データ分割について

本コードでは，インダクタンス行列，レジスタンス行列，固有ベクトルが最もメモリを使用するデータである．これらを合わせると，現在のシミュレーションモデルでも 70MB 以上に達する．また将来的にはもっと大きな行列を解く予定もある．そこで，今後の大規模計算に対応できるように，サブルーチン INDUCT 内で求められるインダクタンス行列の 1 プロセス当たりのメモリ使用量の削減（配列 SINDU の分割）を検討した．インダクタンス行列を求める部分を Fig.4.45 に示す．

a. 効率良いデータ転送が困難

インダクタンス行列を格納する配列は一次元配列 SINDU であるが，その添字を決めるための M1， M2， N_MODE， K_NODE 等は，本コードではなく，入力データを作成する（別の）コードによって決められる値である．よって， IJ の値が不明なことから， SINDU の定義箇所が不明であるため，分割すると最悪の場合定義する毎にプロセス間でデータ転送が必要になる．これは極端な実行時間の増大につながる．

b. 配列宣言（分割宣言）が困難

実際に並列化コーディングをする際にインダクタンス行列を格納するSINDUを分割するには，通常 SINDU の配列サイズ／プロセス数 で配列宣言を行なう．しかし，本コードでは SINDU も含め大部分の配列が動的に決められること，且つ行列が一次元配列で実現されていることから，分割宣言が非常に困難である．

以上 2 つの理由から，インダクタンス行列のデータ分割は行なわないことにした．更に同じようなことがレジスタンス行列にも言えるため，レジスタンス行列のデータ分割も行なわないことにした（固有ベクトルについては，4.5.2参照）．

(2) コーディング

ループ A をサイクリック分割すると，インダクタンス行列を格納する配列 SINDU については，各プロセスでの担当分しか求められない．そこで，配列 SINDU の各要素についてプロセス間で総和を行ない，各要素の値を完全にする必要がある．プロセス間の総和は，MPI ライブラリの mpi_allreduce を用いて行なった．これを用いると，総和後は全プロセスで完全な値を持つ SINDU が求められる．ループ A の分割後と，配列 SINDU の総和部分を Fig.4.46に示す（変更前は Fig.4.45）．

総和は，配列 SINDU のサイズが大きいことから，一括で（一回の命令で）総和を行なうと，受信バッファのための配列を同サイズ用意する必要があることと，総和時にシステム内部で確保されるバッファのサイズが極端に大きくなることからメモリを圧迫する．最悪の場合メモリオーバーで総和が不可能になる．よって，受信バッファの配列を適当なサイズにし，総和を分割して（繰り返して）行なわせるようにした．

Fig.4.46 中において受信バッファの適当なサイズは B の nmtrx2 であり，受信バッファ配列は，インクルードファイル INC_BSIZE において SINDU2(nmtrx2) と宣言した．nmtrx2 も同ファイル中において parameter 文でサイズを決めている．インクルードファイル INC_BSIZE を Fig.4.47 に示す．

配列 SINDU のサイズは，nmtrx であるため，図中 B でそれを nmtrx2 で割ることにより，総和処理の繰り返し数が決まる．C では割り切れなかった場合の余りの要素数が求められる．D においては，実際の総和処理が行なわれる．各プロセス上の配列 SINDU の各要素が，MPI_SUM によってプロセス間で総和された後，受信バッファ配列 SINDU2 へ転送される．E では SINDU2 から元の配列 SINDU へのコピーを行なう．これらの処理が複数回繰り返される．F と G では余りの要素についての総和と，元配列へのコピーが行なわれる．

以上の総和後は，mpi_allreduce を用いたことから各プロセスで同値のインダクタンス行列を持つことになる．

4.5.2 サブルーチン SSPEV

本ルーチンは，オリジナルコーディングが複雑で並列化が困難であったことから，原研計算科学技術推進センターにより公開されている（ソース提供されている）並列数値計算ライブラリ（付録 B参照）を利用して並列化を行なった．

(1) 並列化の方針

サブルーチン SSPEV は標準固有値問題の計算部であり，インダクタンス行列を A，レジスタンス行列を B とした，一般固有値問題

$$Ax = \lambda Bx \tag{4.1}$$

の計算部の一部に当たる．この部分については，オリジナルコーディングが複雑であり，並列化するには，チューニングを行ない易いようにコードの再整理が必要と考えられたこと，GO TO 文による上下への分岐がありその再コーディングが困難であったこと等から，原研計算科学技術推進センターで開発され，また公開されている標準固有値問題を解く並列数値計算ライブラリを用いて並列化することにした．これにより，標準固有値問題の解法が

ハウスホルダ変換 と QL/QR 切替え法 → ハウスホルダ変換 と 2 分法・逆反復法

に変更になった．

標準固有値問題を解く並列数値計算ライブラリでは，行列を 2 次元配列として用い，データ分割方法は，列方向のサイクリック分割を採用している．そのため，ライブラリに渡す行列も予めサイクリック分割した後のものでなければならない仕様であった．また，固有値は全プロセスで共通のデータが求められるが，固有ベクトルは列方向のサイクリック分割に従ってサイクリックに分割されたままで求められる仕様であった．ここで，行列の列方向サイクリック分割の例を Fig.4.48 に示す．

一方，オリジナル版の SSPEV では，標準固有値問題計算部の直前では，配列の 2 次元化もされていないし，列方向のサイクリック分割もされていない．

そこで，並列数値計算ライブラリを呼び出す前に，標準固有値問題へ渡す行列を格納している配列の 2 次元化とそのサイクリック分割を行なうことにした．また，サイクリックに求められる固有ベクトルを分割されたままで以降の計算で用いると，以降の計算についても並列化を行なうことが必要になる．これは，計算コストの低い部分についての並列化を意味するため，実行時間の増大につながる．よって，サイクリックに求められた固有ベクトルについては，オリジナルコードと同様の配列の形に戻し，以降の計算の並列化を避けることにした．

(2) コーディング

サブルーチン SSPEV は大部分を並列数値計算ライブラリに変更したため，その呼び出し部分の変更も行なった．また，ライブラリの呼び出しを SSPEV の代わりに新たに作成したサブルーチン sspev_para で行なうようにした．

・サブルーチン SSPGV

本ルーチンは，サブルーチン SSPEV を呼び出しているルーチンである．SSPGV 内の オリジナル版 SSPEV の呼出し部分を Fig.4.49に，並列化版 SSPEV の呼出し部分を Fig.4.50 に示す．これら 2 つの図からもわかるように，SSPGV から呼び出していた SSPEV を，今回 sspev_para に変更した．実引数については次の通りである．

ap　　：一般固有値問題から標準固有値問題へ変換した後の，Zx= λ x の行列 Z が格納されている配列である．

lmtrx	：行列 Z の下三角部分だけを一次元配列 ap に格納した場合の配列サイズ.
Z	：計算された固有ベクトルが格納される配列.
LDLJYD_para	：行列の行数. オリジナル版では LDLJYD.
W	：計算された固有値が格納される配列.
LJYD	：行列の列数. オリジナル版も同一.
PW	：作業配列.

上の配列の内，オリジナル版と異なるのは，ap のサイズと作業配列 PW，行列の行数である．

ap は並列数値計算ライブラリを使用する上でフル行列のサイズが必要である．よって，元の配列におけるアドレス計算を行なうサブルーチン INPST1A の変更を行なった．INPST1A 内の変更部分を Fig.4.51 に示す．

また，作業配列 PW も並列数値計算ライブラリで用いるものであることから，新たにサイズを確保できるように，INPST1A の呼び出し部分と内容の変更を行なった．これを Fig.4.52 に示す．更に，SSPGV の親ルーチンである EIGEN の呼び出し部分と内容（SSPGV の呼び出し部分）の変更も行なった．これを Fig.4.53 に示す．

行列の行数は，オリジナル版では列数よりもサイズが 1 大きい場合があったが，標準固有値計算部ではその領域は使用しないため，また並列数値計算ライブラリでも使用しないために，列数と同一にした．

・サブルーチン sspev_para

SSPEV の並列化版サブルーチン sspev_para を Fig.4.54に示す．本ルーチンについては，説明上，行番号を付加して説明する．

a. 1 ～ 10 行目

この部分は，サブルーチンに渡される仮引数とそれら宣言部分である．

c(N,N)	：これは，本ルーチンを呼び出す前に，一般固有値問題から標準固有値問題へ変換した後の，Zx= λ x の行列 Z が格納されている配列である．Z は実対称行列である．今回使用した並列ライブラリでは，下三角部分のみではなく，フル行列で且つ 2 次元配列をサイクリック分割したものを用いなければならない仕様であるため，上三角部分も格納できるようにサイズを増やしてこのように宣言した．
Z(NNN1,NNN2)	：標準固有値問題における固有ベクトルを格納する配列である.
W(NNN2)	：標準固有値問題における固有値を格納する配列である.
その他配列	：並列数値計算ライブラリで使用するワーク配列である.

b. 12 ～ 16 行目

固有ベクトルを格納する配列を 0 クリアする．

c. 18 ～ 34 行目

sspev_para の呼出しで実引数として渡した AP は，本ルーチンでは仮引数 c で受け，c(N,N) として使用するようにした．そのため，本ルーチンの呼出し直後は，c(N,N) 内は一次元配列

AP 内のデータ並びになっている．そこで，この部分において，c(N,N) のデータ並びを 2 次元配列での並びに変更する．

d. 36 ～ 40 行目

実対称行列（フル行列）にする．

e. 44 ～ 50 行目

2 次元配列の並びに変更した c(N,N) を各プロセスにサイクリック分割する．

f. 54 ～ 63 行目

この部分では並列数値計算ライブラリを呼出す．配列 Z に標準固有値問題の固有ベクトルが，配列 W には固有値が求められる．W については，これら並列数値計算ライブラリにより，結果が各プロセスに分配されるため，これらとは別に収集するためのコーディングは必要ない．

g. 67 ～ 72 行目

固有ベクトル Z は，各プロセスで求められた（各プロセスの計算担当分のみが求められた）ままの状態であるため，この部分で mpi_allgather による収集操作を行なうようにした．ここで，c(N,N) はこの時点以降では使用されないデータであるため，これを収集操作での受信バッファとして使用した．収集の仕方は，各プロセス上の配列 Z の 1 列を収集単位とし，1PE → 2PE → 3PE → 4PE の順にそれぞれの 1 列を収集する方法（サイクリック分割の逆を考えればよい）で行なった．これは，並列ライブラリにおいて，各固有値に対する各固有ベクトルの計算の分担方法がサイクリックに分担される仕様であるためであり，更に求められる固有ベクトルは配列 Z に列番号の若い方に詰めて配置されるためである．

h. 74 ～ 94 行目

固有ベクトルを収集する際，使用するプロセス数で固有ベクトル数が割り切れない場合がある．例えば，固有ベクトル総数が 22，プロセス数が 4 の場合は割り切れずに余りが 2 になり，その余りの固有ベクトルがランク 0 とランク 1 で求められる場合である．よって，余りの固有ベクトルの収集操作を行なうようにした．余りの部分を収集する方法は，配列 Z に求められている余りの固有ベクトルを，一旦 g. で行なった収集後の配列 c に代入し，その部分を mpi_bcast で全プロセスの配列 c に転送する方法にした．

i. 96 ～ 100 行目

ここでは，全プロセスで共通な，受信バッファ配列 c に求められている収集後の固有ベクトルを，元の配列 z にコピーする処理である．こうすることにより，以降の処理で使用する配列名を変更する必要が無くなる．

j. サブルーチン SSPGV の sspev_para 呼出し後の処理

この部分は（Fig.4.50参照），配列 Z についての処理であるが，オリジナル版の配列 Z は，一次元目のサイズが 2 次元目のサイズより 1 大きくなる場合がある．その場合に，その（元の）配列へのコピーをここで行なうようにした．また，一般固有値問題解法部分において，その 1 大きい部分にはアクセスしないことから，コピー後に 0 クリアも行なうことにした．

4.5.3 サブルーチン DTPSV

本ルーチンは，サブルーチン SSPGV のあるループ中から呼び出されている．このループを分割することで並列化を行なった．

(1) 並列化の方針

本ルーチンの呼出し部分を Fig.4.55 に示す．また，DTPSV を Fig.4.56 に示す．Fig.4.55 において DTPSV を囲んでいる DO 10 ループは，J で回転するループであり，サブルーチンには Z(1,J) を渡す．Fig.4.56 の DTPSV 内では Z(1,J) ～ Z(N,J) を定義する．DTPSV に渡している変数や配列の中で更新されるのは，配列 Z のみであり，その他は不変である（参照するのみ）．更に，J=1 の時は Z(1,1) ～ Z(N,1)，J=2 の時は Z(1,2) ～ Z(N,2) のように更新されるため，J について独立であることから，J で回転する DO 10 ループを分割できることがわかる．そこで，このループを分割して並列化することにした．分割の方法はサイクリック分割とした．こうすると，サブルーチン INDUCT のループ分割時と同様に DO ループ行を少々変更するだけであるため，各プロセスで担当させる回転範囲の計算のオーバーヘッドを減らすことができる．

(2) コーディング

DO 10 の DO 文は少々変更する必要はあるが，呼び出されるサブルーチン DTPSV 内を変更する必要は無い．ただし，DO ループのサイクリック分割により配列 Z が各プロセスで求められるため，それらのデータを収集し，全プロセスに転送する必要がある．そこで，新たにサブルーチン z_trans を用意した．DO 10 の DO 文の変更と z_trans の呼び出しの追加結果を Fig.4.57 に示す．実引数で渡している Z は各プロセスで求められた Z であり，返り値は，全プロセスで収集された Z になる．また，配列 AP は z_trans 内で収集処理を行なう際の受信バッファとするために渡してある．NEIG は，ループ回転数である．

(3) サブルーチン z_trans

ここでは，各プロセスで求められた配列 z の全プロセスでの収集操作を mpi_allgather と mpi_bcast によって行なった．収集の仕方は，サブルーチン sspev_para における固有ベクトルを格納する配列 z の場合と同様である．ただし，sspev_para における場合は，配列 Z の列番号の若い方にデータが詰められていたが，ここでは，詰められていないため，その分の処理が異なっている．サブルーチン z_trans を Fig.4.58 に示す．

4.5.4 ファイルへの I/O 制御

並列実行において複数プロセスによる I/O を行なう場合，基本的には各プロセスは，独立に（別々に）ファイルポインタを持ち，更にファイルのオープンやクローズも独立に行なう．よって，プロセス毎にファイルを用意するか，または，I/O プロセスを 1 つ決めて，このプロセスで読み込んだデータを他のプロセスへデータ転送してやったり，他のプロセスから I/O プロセスへデータ転送してやり I/O プロセスで書き出してやることが必要になる．ここでは，この I/O について本コードに施した変更について述べる．

(1) I/O プロセスの決定

本コードでは中間ファイルや標準出力のファイルのサイズが大きいことから，プロセス毎にファイルを用意することを避け，I/O プロセスを用意する方法で変更を行なった．I/O プロセスは，インクルードファイル INC_MPI 内の prameter 文による iw_pe で決めることができるようにし，I/O プロセスにランク 0 を指定した．インクルードファイル INC_MPI を Fig.4.59 に示す（他の宣言については 4.5.5参照）．

(2) I/O プロセスとデータ転送

実際のコード中の I/O 部分には，(1) で述べた iw_pe を IF 文で用いて I/O を行なわせるようにし，データ転送は READ の直後のみで行なわせるようにした．これは，並列化を行なったサブルーチン INDUCT と SSPEV 以外の部分では，各データが全プロセスで共通であることから，WRITE する場合は他プロセスから I/O プロセスへのデータ転送は必要無いからである．ところで，本コードにおいては，READ 部分でも直後でデータ転送を行なっていない箇所がある．これは，実行開始時に既にディスク上に存在するデータに対しては，複数プロセスからの READ が可能であるためである．よって，既に存在するファイルからの READ を行なった直後ではデータ転送は行なわせていない．READ 直後のデータ転送の例を Fig.4.60 に示す．

(3) 無駄なデータ転送を省くための変更

今回の作業では，サブルーチン INDUCT と SSPEV に並列化を施したが，それらは，全体の実行ステップのステップ 1 の部分になる．ステップ 2 ～ステップ 3 では，全プロセスが共通のデータを用い，同じ計算を行なう．よって，この部分における，WRITE 時の他プロセスから I/O プロセスへのデータ転送や，READ 直後の他プロセスへのデータ転送は無駄である．そこで，サブルーチン main1 においてステップ 1 での最終 I/O からステップ 2 以降では I/O プロセスのみで実行させるようにした．main1 の変更部分を Fig.4.61 に示す．

4.5.5 インクルードファイル

EDDYCAL の並列化で用意したインクルードファイルは，次の 3 つである．

INC_NPE, INC_MPI, INC_BSIZE

INC_BSIZE と INC_MPI の一部については，それぞれ 4.5.1 と 4.5.4 で説明した．ここでは，その他について説明する．

INC_NPE ：使用するプロセス数を parameter 文によって指定する．このインクルードファイルを Fig.4.62 に示す．

INC_MPI ：インクルードファイル mpif.h は MPI ライブラリ使用上必須であり，my_pe と status(MPI_STATUS_SIZE) は MPI ライブラリを使用する殆んどの場合に必要になる．iw_pe を用いるルーチンでは MPI ライブラリを使用している場合が多いため，このインクルードファイルで一括で宣言した．

4.6 ベクトル並列化の評価

ここでは，VPP 上での高速化後の実行結果の妥当性と実行時間について述べる．

4.6.1 実行結果について

ユーザから頂いた本コードの実データを用いて，オリジナルコードの実行結果と高速化後の実行結果を比較した．その結果，ベクトル化後コードの実行結果も並列化後の実行結果も完全一致であった．

4.6.2 実行時間について

ベクトル並列化後の本コードの性能評価を行なうため，実行結果評価時と同一の実データを用いて，オリジナルコードとベクトル並列化コードの実行時間を測定した．測定は，実時間を取得できるサービスルーチンと CPU 占有時間を取得できるサービスルーチンを本コードのメインルーチンに挿入して行なった．測定結果を Table 4.1 に示す．これより，4 並列実行まではほぼ妥当な性能と考えられるが，それ以上の並列実行ではあまり性能が上がらない．これは，並列化を施したルーチンの性能が並列数が増える毎に飽和状態になっていることが挙げられる．

4.7 VPP 版の AP3000 への移植

VPP 上では，並列化においてデータを分割しなかったことから，使用できるメモリ量は 200MB/1PE が上限になる．しかし，AP3000 では，2GB/1PE のメモリを使用できることから，VPP 版のベクトル並列化コードを AP3000 に移植した．

前述したように，VPP と AP3000 共に MPI ライブラリが搭載されていることから，VPP 版コードをそのまま使用し，並列実行が可能である．また，AP3000 はベクトル計算機ではないため，VPP 上でベクトル実行が行なわれる部分はスカラ実行が行なわれることになるが，プログラムは同一であり，プログラムから判断される計算結果への影響は無い（実際の実行結果については 4.8 参照）．実際の移植作業も最終的にはソースファイルの転送を行うだけで済んだ．

4.8 AP3000 での実行評価

ここでは，AP3000 へ移植した VPP 版高速化コードの実行結果の妥当性と実行時間について述べる．

4.8.1 実行結果について

VPP 上での実行結果評価時と同一のデータを用いて，移植後のコードの実行結果と VPP での実行結果を比較した．その結果，異なる部分はあるものの大部分が完全一致していた．また，AP3000 上での 1PE 並列実行結果と 2 ～ 4PE 実行結果も完全一致であった．

4.8.2 実行時間について

AP3000 版コードの性能評価を行なうため，実行時間評価時と同一のデータを用いて，オリジナルコードと移植コード（並列化版）の実行時間を測定した．測定結果を Table 4.2 に示す．まず，移植された並列化版 1PE 並列実行がオリジナル版よりも実行時間が短い．これは，標準固有値問題計算部の変更によるものである．そして，並列化版の複数 PE 並列実行の効果は，

VPP 上の効果ほどではない．これは，VPP 上でベクトル実行されていた部分が AP3000 ではスカラ実行になるためで，その部分のコストが VPP より目立っていることが原因である．

4.9 まとめ

今回の並列化で，VPP 上のメモリ量の都合から AP3000 への移植も行なった．その結果，従来までの VPP でのシングル実行のみではなく，並列実行も可能になり，AP3000 でのシングル実行，並列実行が可能になった．よって，両計算機上のジョブクラスをフルに利用できるようになった．これは，ユーザにとって，VPP と AP3000 から，よりスループットの短いジョブクラスを選択できることを意味する．また，AP3000 ではメモリを 2GB/1PE まで利用できることから，従来よりも大規模なモデルのシミュレーションが可能になり，更に，MPI ライブラリで並列化を行なったことは，今後の並列計算機間での移植をスムーズにするはずである．

尚，MPI を用いると，利点として，コード中の並列化命令部のインターフェースが各ベンダーの計算機上で共通になることが挙げられるが，一方，各計算機の仕様の違いにより，データ転送用のメモリが予想外に必要になったり，使用する並列化命令によって各計算機間で処理速度が異なるデメリットが発生する場合があることを最後に付け加えておく．

今回の作業が今後の EDDYCAL コードの利用拡大につながることを期待したい．

Table. 4.1 Execution time after vectorization and parallelization on VPP.

※ ユーザ実行環境
倍率計算の基

	実時間	倍率	CPU 時間	倍率
・オリジナル版				
スカラ実行 (NFS)	6691 sec		4979 sec	
スカラ実行 (SCFS)	4971 sec		4949 sec	
ベクトル実行 (NFS) ※	3134 sec		1418 sec	
ベクトル実行 (SCFS) #	1440 sec		1382 sec	
・ベクトル化版				
スカラ実行 (NFS)	6110 sec		4928 sec	
スカラ実行 (SCFS)	4971 sec		4901 sec	
ベクトル実行 (NFS)	2503 sec		1147 sec	
ベクトル実行 (SCFS)	1213 sec	1.2 倍	1116 sec	1.2 倍
・ベクトル並列化版				
1PE 実行 (SCFS)	1212 sec	1.2 倍	1193 sec	1.2 倍
2PE 実行 (SCFS)	686 sec	2.1 倍	678 sec	2.0 倍
3PE 実行 (SCFS)	514 sec	2.8 倍	508 sec	2.7 倍
4PE 実行 (SCFS)	428 sec	3.4 倍	418 sec	3.3 倍
8PE 実行 (SCFS)	297 sec	4.8 倍	289 sec	4.8 倍
16PE 実行 (SCFS)	237 sec	6.1 倍	228 sec	6.1 倍

Table. 4.2 Execution time on AP3000.

倍率計算の基

	実時間	倍率	CPU 時間	倍率
・オリジナル版	3393sec		3354sec	
・並列化版				
1PE 実行#	2772sec		2699sec	
2PE 実行	1768sec	1.56 倍	1722sec	1.57 倍
3PE 実行	1443sec	1.92 倍	1406sec	1.92 倍
4PE 実行	1265sec	2.20 倍	1249sec	2.20 倍

```
      SUBROUTINE INDUCT(....
          .
          .

      DO IR = 1, K_ROT_Z

      DO IS = 1, N_SYM

      DO IE1 = 1, N_ELMT
            .
            .
        DO .....
            .
        ENDDO
            .
            .
            .
        DO .....
            .
        ENDDO
            .
            .
      ENDDO
      ENDDO
      ENDDO
```

Fig. 4.1 Outline of subroutine INDUCT.

```
s           DO IE2 = IESTRT, N_ELMT                          ← C
m              IF( RIN(IE2).GT.R12 ) THEN                    ← D
s                 RIN(IE2) = 0.0D0
s2                DO J1 = 12, 23
v2                   XX = XYZ_GAUS(1,J1,IE1)
v2                   YY = XYZ_GAUS(2,J1,IE1)
v2                   ZZ = XYZ_GAUS(3,J1,IE1)
v2                   WW = WH(J1)
v2                   DO J2 = 12, 23                           ← A
v2                      RIN(IE2)=RIN(IE2) + WW*WH(J2)/DS
        +QRT((XX-XYZ_GAUS2(1,J2,IE2))**2+
        +(YY-XYZ_GAUS2(2,J2,IE2))**2+(ZZ-
        +XYZ_GAUS2(3,J2,IE2))**2)
v2                   END DO
v2                END DO
s              ELSE IF( RIN(IE2).GT.R07 ) THEN               ← D
s                 RIN(IE2) = 0.0D0
s2                DO J1 = 5, 11
v2                   XX = XYZ_GAUS(1,J1,IE1)
v2                   YY = XYZ_GAUS(2,J1,IE1)
v2                   ZZ = XYZ_GAUS(3,J1,IE1)
v2                   WW = WH(J1)
v2                   DO J2 = 5, 11                            ← B
v2                      RIN(IE2)=RIN(IE2) + WW*WH(J2)/DS
        +QRT((XX-XYZ_GAUS2(1,J2,IE2))**2+
        +(YY-XYZ_GAUS2(2,J2,IE2))**2+(ZZ-
        +XYZ_GAUS2(3,J2,IE2))**2)
v2                   END DO
v2                END DO
v              END IF                                         ← D
v           END DO
```

Fig. 4.2 Before modification of vectorized DO loops in subroutine INDUCT.

```
      SUBROUTINE INDUCT(NMTRX,N_ELMT,N_ELMT2,N_NODE,N_MODE,K_NODE, N
     +ODE_ELMT,     XYZ_NODE,XYZ_GAUS,AREA_ELMT,RI_ELMT,EELMT,SINDU,
c.org     + XYZ_GAUS2,RIN,EELEM2,XXX)
c.vec-s
     + XYZ_GAUS2,RIN,EELEM2,XXX,
     +i07v,i12v )
c.vec-e
                .
                .
                .
c.vec-s
      integer*4 i07,i12,i07v(N_ELMT),i12v(N_ELMT)
c.vec-e
                .
                .
                .
         i07=0
         i12=0
v        do ie2=iestrt,n_elmt                                  ← A
v          if( RIN(IE2).GT.R12 ) then
v            i12=i12+1                                       ← C
v            i12v(i12)=ie2                                   ← B
v            RIN(IE2) = 0.0D0
v          elseif( RIN(IE2).GT.R07 ) then
v            i07=i07+1                                       ← C
v            i07v(i07)=ie2                                   ← B
v            RIN(IE2) = 0.0D0
v          endif
v        end do

s        do j1=12,23                                           ← D
v          XX = XYZ_GAUS(1,J1,IE1)
v          YY = XYZ_GAUS(2,J1,IE1)
v          ZZ = XYZ_GAUS(3,J1,IE1)
v          WW = WH(J1)
s        do j2=12,23
```

Fig. 4.3 After modification of vectorized DO loops in subroutine INDUCT(1/2).

```
   *vocl loop,novrec(rin)                                          ← E
v         do ie2=1,i12
v           RIN(i12v(ie2))=RIN(i12v(ie2)) + WW*WH(J2)/DSQRT((XX-XYZ_
         &GAUS2(1,J2,i12v(ie2)))**2+(YY-XYZ_GAUS2(2,J2,i12v(ie2)))**
         &2+(ZZ-XYZ_GAUS2(3,J2,i12v(ie2)))**2)
v          end do
s          end do
v          end do

s          do j1=5,11                                              ← D
v            XX = XYZ_GAUS(1,J1,IE1)
v            YY = XYZ_GAUS(2,J1,IE1)
v            ZZ = XYZ_GAUS(3,J1,IE1)
v            WW = WH(J1)
s          do j2=5,11
   *vocl loop,novrec(rin)                                          ← E
v          do ie2=1,i07
v            RIN(i07v(ie2))=RIN(i07v(ie2)) + WW*WH(J2)/DSQRT((XX-XYZ_
         &GAUS2(1,J2,i07v(ie2)))**2+(YY-XYZ_GAUS2(2,J2,i07v(ie2)))**
         &2+(ZZ-XYZ_GAUS2(3,J2,i07v(ie2)))**2)
v          end do
s          end do
v          end do
                .
                .
                .
```

Fig. 4.3 After modification of vectorized DO loops in subroutine INDUCT(2/2).

```
            .
            .
      SUBROUTINE INPST1A( N_NODE, N_MODE, N_ELMT, N_ELMTK, N_ELMT2,
     +NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,     N_MATDA, N_MATNO, N_
     +MAT, NMPC, MXTEN,     N_LINE, N_MATL, N_MATLDA, N_MATLNO,     N_CO
     +IL2, N_PASS, N_CURAP, N_CURNP, M_CURNP,     M1,M2,M3,M4,M5,M6,MA1,
     +MA2,MA3,MA4,M7,M8,M9,MC1,MC2,MZ1,     N1,N2,N3,N4,N5,N6,NB,NA1,NA2
     +,N7,N8,NC1,NC2,NC3,NC4,NC5,     N9,N10,N11,N12,N14,NZ1,NC6D,NC7D,N
c    +CZ1,     N11B,NZ2,N15,N16,N17,NZ3)
c.vec-s
     +CZ1,     N11B,NZ2,N15,N16,N17,NZ3,
     +M_VW1,M_VW11 )
c.vec-e
            .
            .
      M9=M8+NMPC          +1
      MC1=M9+NMPC*MXTEN   +1
      MC2=MC1+2*N_COIL2    +1
      MZ1=MC2+3*N_PASS
c.vec-s
      M_VW1=MZ1+N_ELMT +1
      M_VW11=M_VW1+N_ELMT +1
c.vec-e
      N1=1                +1
      N2=N1+3*N_NODE       +1
      N3=N2+N_ELMT         +1
      N4=N3+3*3*N_ELMT     +1
      N5=N4+N_MAT          +1
            .
            .
```

Fig. 4.4 Modification of statements in subroutine INPST1A.

```
            .
            .
         CALL INPST1A( N_NODE, N_MODE, N_ELMT, N_ELMTK, N_ELMT2,
     +NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,        N_MATDA, N_MATNO,
     + N_MAT, NMPC, MXTEN,        N_LINE, N_MATL, N_MATLDA, N_MATLNO,
     +     N_COIL2, N_PASS, N_CURAP, N_CURNP, M_CURNP,        M1,M2,M3,M
     +4,M5,M6,MA1,MA2,MA3,MA4,M7,M8,M9,MC1,MC2,MZ1,        N1,N2,N3,N4,N
     +5,N6,NB,NA1,NA2,N7,N8,NC1,NC2,NC3,NC4,NC5,        N9,N10,N11,N12,N
c.org     +14,NZ1,NC6D,NC7D,NCZ1,        N11B,NZ2,N15,N16,N17,NZ3 )
c.vec-s
     +14,NZ1,NC6D,NC7D,NCZ1,        N11B,NZ2,N15,N16,N17,NZ3,
     +M_VW1,M_VW11 )
c.vec-e
            .
            .
```

Fig. 4.5 Modification of call statement for subroutine INPST1A.

```
      SUBROUTINE MKINDUCT(N_ELMT,N_ELMT2,N_NODE,N_MODE,NJYD,LDNJYD,
     +NMTRX, K_NODE,NODE_ELMT,     XYZ_NODE,XYZ_GAUS,AREA_ELMT,RI_ELMT,E
     +ELMT,SINDU, PP,     NMPC, MXTEN, NA, NDPN, NIDP, WD, WI,     N_COI
c.org     +L2,N_PASS,KIND_COIL2,PARAMCOIL2,ANOD2,AAA2, WORK)
c.vec-s
     +L2,N_PASS,KIND_COIL2,PARAMCOIL2,ANOD2,AAA2, WORK,
     +MV_WORK1,MV_WORK11 )
c.vec-e
            .
            .
c.vec-s
      integer*4 MV_WORK1(N_ELMT),MV_WORK11(N_ELMT)
c.vec-e
            .
            .
      CALL INDUCT(NMTRX,N_ELMT,N_ELMT2,N_NODE,NJYD,K_NODE,NODE_ELMT,
     + XYZ_NODE,XYZ_GAUS,AREA_ELMT,RI_ELMT,EELMT,SINDU,     WORK(NL3),WO
c.org     +RK(NL4),WORK(NL5),WORK(NL6))
c.vec-s
     +RK(NL4),WORK(NL5),WORK(NL6),
     +MV_WORK1,MV_WORK11 )
c.vec-e
            .
            .
```

Fig. 4.6 Modification of statements in subroutine MKINDUCT.

```
            .
            .
         CALL MKINDUCT(N_ELMT,N_ELMT2,N_NODE,N_MODE,NJYD,LDNJYD,NMTRX,
     +       IA(M2),IA(M5),        A(N1),A(N12),A(N2),A(N11),A(N3),A(N9),
     + A(N11B),        NMPC, MXTEN, IA(M7), IA(M8), IA(M9), A(N7), A(N8)
c.org     +,     N_COIL2,N_PASS,IA(MC1),A(NC1),A(NC6D),A(NC7D), A(N14) )
c.vec-s
     +,        N_COIL2,N_PASS,IA(MC1),A(NC1),A(NC6D),A(NC7D), A(N14),
     +IA(M_VW1),IA(M_VW11) )
c.vec-e
            .
            .
```

Fig. 4.7 Modification of call statement for subroutine MKINDUCT.

```
s           DO IG1 = 1, N_GRID
v              IF( (RR2(IG1).GE.R12).AND.(RR2(IG1).LT.R07) ) THEN
m                 BBB(1,IG1) = 0.0D0
s                 BBB(2,IG1) = 0.0D0
s                 BBB(3,IG1) = 0.0D0
v                 DO J2 = 5, 11
v                    XXX = XYZ_GRID(1,IG1)-XYZ_GAUS2(1,J2,IE2)
v                    YYY = XYZ_GRID(2,IG1)-XYZ_GAUS2(2,J2,IE2)
v                    ZZZ = XYZ_GRID(3,IG1)-XYZ_GAUS2(3,J2,IE2)
v                    RRR = XXX**2 + YYY**2 + ZZZ**2
v                    RRI = WH(J2)/(RRR*DSQRT(RRR))
v                    BBB(1,IG1) = BBB(1,IG1) + XXX * RRI
v                    BBB(2,IG1) = BBB(2,IG1) + YYY * RRI
v                    BBB(3,IG1) = BBB(3,IG1) + ZZZ * RRI
v                 END DO
v              ELSE IF( RR2(IG1).LT.R12 ) THEN
m                 BBB(1,IG1) = 0.0D0
s                 BBB(2,IG1) = 0.0D0
s                 BBB(3,IG1) = 0.0D0
v                 DO J2 = 12, 23
v                    XXX = XYZ_GRID(1,IG1)-XYZ_GAUS2(1,J2,IE2)
v                    YYY = XYZ_GRID(2,IG1)-XYZ_GAUS2(2,J2,IE2)
v                    ZZZ = XYZ_GRID(3,IG1)-XYZ_GAUS2(3,J2,IE2)
v                    RRR = XXX**2 + YYY**2 + ZZZ**2
v                    RRI = WH(J2)/(RRR*DSQRT(RRR))
v                    BBB(1,IG1) = BBB(1,IG1) + XXX * RRI
v                    BBB(2,IG1) = BBB(2,IG1) + YYY * RRI
v                    BBB(3,IG1) = BBB(3,IG1) + ZZZ * RRI
v                 END DO
v              END IF
v           END DO
```

Fig. 4.8 Before modification of vectorized DO loops in subroutine FLUXG.

```
      SUBROUTINE FLUXG(N_GRID,N_ELMT,N_NODE,NODE_ELMT,XYZ_NODE,      AREA
c.org     +_ELMT,XYZ_GAUS,XYZ_GRID,RR2,XYZ_GAUS2,BBB,EDY,BGRDALL)
c.vec-s
     +_ELMT,XYZ_GAUS,XYZ_GRID,RR2,XYZ_GAUS2,BBB,EDY,BGRDALL,
     +i07v,i12v)
c.vec-e

c.vec-s
      integer*4 i07,i12,i07v(N_GRID),i12v(N_GRID)
c.vec-e

m         i07=0
s         i12=0
v         do ig1=1,n_grid
v           if( (RR2(IG1).GE.R12).AND.(RR2(IG1).LT.R07) ) then
v             i07=i07+1
v             i07v(i07)=ig1
v             bbb(1,ig1)=0.0d0
v             bbb(2,ig1)=0.0d0
v             bbb(3,ig1)=0.0d0
v           elseif( RR2(IG1).LT.R12 ) then
v             i12=i12+1
v             i12v(i12)=ig1
v             bbb(1,ig1)=0.0d0
v             bbb(2,ig1)=0.0d0
v             bbb(3,ig1)=0.0d0
v           endif
v         end do
```

Fig. 4.9 After modification of vectorized DO loops in subroutine FLUXG(1/2).

```
s         do j2=5,11
    *vocl loop,novrec(bbb)
v         do ig1=1,i07
v           XXX = XYZ_GRID(1,i07v(ig1))-XYZ_GAUS2(1,J2,IE2)
v           YYY = XYZ_GRID(2,i07v(ig1))-XYZ_GAUS2(2,J2,IE2)
v           ZZZ = XYZ_GRID(3,i07v(ig1))-XYZ_GAUS2(3,J2,IE2)
v           RRR = XXX**2 + YYY**2 + ZZZ**2
v           RRI = WH(J2)/(RRR*DSQRT(RRR))
v           BBB(1,i07v(ig1)) = BBB(1,i07v(ig1)) + XXX * RRI
v           BBB(2,i07v(ig1)) = BBB(2,i07v(ig1)) + YYY * RRI
v           BBB(3,i07v(ig1)) = BBB(3,i07v(ig1)) + ZZZ * RRI
v         end do
s         end do

s         do j2=12,23
    *vocl loop,novrec(bbb)
v         do ig1=1,i12
v           XXX = XYZ_GRID(1,i12v(ig1))-XYZ_GAUS2(1,J2,IE2)
v           YYY = XYZ_GRID(2,i12v(ig1))-XYZ_GAUS2(2,J2,IE2)
v           ZZZ = XYZ_GRID(3,i12v(ig1))-XYZ_GAUS2(3,J2,IE2)
v           RRR = XXX**2 + YYY**2 + ZZZ**2
v           RRI = WH(J2)/(RRR*DSQRT(RRR))
v           BBB(1,i12v(ig1)) = BBB(1,i12v(ig1)) + XXX * RRI
v           BBB(2,i12v(ig1)) = BBB(2,i12v(ig1)) + YYY * RRI
v           BBB(3,i12v(ig1)) = BBB(3,i12v(ig1)) + ZZZ * RRI
v         end do
s         end do
```

Fig. 4.9 After modification of vectorized DO loops in subroutine FLUXG(2/2).

```
      SUBROUTINE INPST3A( N_NODE,N_MODE,N_ELMT,N_ELMTK,KSYM,     N_NODE3
     +,N_ELMT3,N_ELMT3K,     NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,
     +   N_MATDA,N_MATNO,N_MAT,     N_COIL2, N_PASS, N_CURAP, N_CURNP, M
     +_CURNP,     N_COIL1, N_ACTI, N_CURAA, N_CURNA, M_CURNA,     N_TIME
     +, N_PHAS,     N_GRID, N_COIL0, N_CONST,     NFTOT, NODCOIL,NEGF,N_
     +TIM3,     M1,M2,M3,M4,M5,M6,MC1,MC2,MC3,MC4,M15,M16,MC6,MC7,MC8,MZ
     +3,     N1,N2,NB,N12,NC1,NC2,NC3,NC4,NC5,NC6,NC7,NC8,NC9,     NC10,
     +NC13,NC14,NC15,N22,NA6,N33,NC16,NC17,NC18,NC19,NC20,     NC21,NC22
     +,NC23,NC24,N44,N45,NC25,N46,NC26,NC27,ND1,NZ6,     N39,N40,N41,N42
c.org     +,N43,N47,NC28,NC29,NC30,NCQ,NZ7,     MD1,ND2,ND3,ND4,ND5,ND6)
c.vec-s
     +,N43,N47,NC28,NC29,NC30,NCQ,NZ7,     MD1,ND2,ND3,ND4,ND5,ND6,
     +M_VW1,M_VW11,M_VW2,M_VW22)
c.vec-e
                .
                .
      MC8=MC7+N_CONST         +1
      MD1=MC8+11*NFTOT    +1
      MZ3=MD1+N_TIM3    +1
c.vec-s
      M_VW1=MZ3+N_ELMT  +1
      M_VW11=M_VW1+N_ELMT  +1
      M_VW2=M_VW1+N_GRID  +1
      M_VW22=M_VW2+N_GRID +1
c.vec-e
      N1=1                +1
      N2=N1+3*N_NODE3      +1
      NB=N2+N_ELMT3        +1
      N12=NB+N_MAT             +1
                .
                .
```

Fig. 4.10 Modification of statements in subroutine INPST3A.

```
                .
                .
      CALL INPST3A( N_NODE,N_MODE,N_ELMT,N_ELMTK,KSYM,     N_NODE3,N_ELM
     +T3,N_ELMT3K,     NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,     N_M
     +ATDA,N_MATNO,N_MAT,     N_COIL2, N_PASS, N_CURAP, N_CURNP, M_CURNP
     +,     N_COIL1, N_ACTI, N_CURAA, N_CURNA, M_CURNA,     N_TIME, N_PH
     +AS,     N_GRID, N_COIL0, N_CONST,     NFTOT, NODCOIL,NEGF,N_TIM3,
     +    M1,M2,M3,M4,M5,M6,MC1,MC2,MC3,MC4,M15,M16,MC6,MC7,MC8,MZ,
     +N1,N2,NB,N12,NC1,NC2,NC3,NC4,NC5,NC6,NC7,NC8,NC9,     NC10,NC13,NC
     +14,NC15,N22,NA6,N33,NC16,NC17,NC18,NC19,NC20,     NC21,NC22,NC23,N
     +C24,N44,N45,NC25,N46,NC26,NC27,ND1,NZ6,     N39,N40,N41,N42,N43,N4
c.org     +7,NC28,NC29,NC30,NCQ,NZ7,     MD1,ND2,ND3,ND4,ND5,ND6)
c.vec-s
     +7,NC28,NC29,NC30,NCQ,NZ7,     MD1,ND2,ND3,ND4,ND5,ND6,
     +M_VW1,M_VW11,M_VW2,M_VW22)
c.vec-e
                .
                .
```

Fig. 4.11 Modification of call statement for subroutine INPST3A.

```
      SUBROUTINE EDCMAG(NEGF,N_ELMT,N_ELMT3,N_NODE,N_NODE3,     N_MATDA,
     +N_MATNO,N_MAT,N_GRID,NFTOT,     N_ACTI,N_CURAA,N_CURNA,M_CURNA,KIN
     +D_CUR1,COEF1,     TIMECUR1,PHASCUR1,     N_PASS,N_CONST,KIND_CURO,
     +COEFO,     L_ELMT,MAT_ELMT,L_MAT,L_GRID,LTIME,N_TIME,N_PHAS,     A
     +REA_ELMT,PHAS,     BNOD1,BNOD2,BNODO,BGRD1,BGRD2,BGRDO,     BCOL1,
     +BCOL2,BCOLO,     EDY,EDY3,TCOILP,     BNODALL,FORCO,FORCALL,BGRDAL
     +L,BCOLALL,FCOLALL,     NFCOIL,FCOIL,     NODE_ELMT,XYZ_NODE,XYZ_GR
     +ID,XYZ_GAUS,     RR2,BBB,XYZ_GAUS2,     N_TIM3,LTIM3,TIM3,EDY_B,TC
c.org     +OILP_B,EDY_D,TCOILP_D)
c.vec-s
     +OILP_B,EDY_D,TCOILP_D,
     +MV_WORK1,MV_WORK11,MV_WORK2,MV_WORK22)
c.vec-e
                .
                .
c.vec-s
      integer*4 MV_WORK1(N_ELMT),MV_WORK11(N_ELMT),MV_WORK2(N_GRID),
     +MV_WORK22(N_GRID)
c.vec-e
                .
                .
      CALL FLUXN(N_ELMT,N_ELMT3,N_NODE,NODE_ELMT,XYZ_NODE,     AREA_ELMT
c.org     +,XYZ_GAUS,RR2,XYZ_GAUS2,BBB,EDY,BNODALL)
c.vec-s
     +,XYZ_GAUS,RR2,XYZ_GAUS2,BBB,EDY,BNODALL,
     +MV_WORK1,MV_WORK11)
c.vec-e
                .
                .
      CALL FLUXG(N_GRID,N_ELMT,N_NODE,NODE_ELMT,XYZ_NODE,     AREA_ELMT,
c.org     +XYZ_GAUS,XYZ_GRID,RR2,XYZ_GAUS2,BBB,EDY,BGRDALL)
c.vec-s
     +XYZ_GAUS,XYZ_GRID,RR2,XYZ_GAUS2,BBB,EDY,BGRDALL,
     +MV_WORK2,MV_WORK22)
c.vec-e
                .
                .
      CALL FLUXN(N_ELMT,N_ELMT3,N_NODE,NODE_ELMT,XYZ_NODE,     AREA_ELMT
c.org     +,XYZ_GAUS,RR2,XYZ_GAUS2,BBB,EDY,BNODALL)
c.vec-s
     +,XYZ_GAUS,RR2,XYZ_GAUS2,BBB,EDY,BNODALL,
     +MV_WORK1,MV_WORK11)
c.vec-e
                .
                .
      CALL FLUXG(N_GRID,N_ELMT,N_NODE,NODE_ELMT,XYZ_NODE,     AREA_ELMT,
c.org     +XYZ_GAUS,XYZ_GRID,RR2,XYZ_GAUS2,BBB,EDY,BGRDALL)
c.vec-s
     +XYZ_GAUS,XYZ_GRID,RR2,XYZ_GAUS2,BBB,EDY,BGRDALL,
     +MV_WORK2,MV_WORK22)
c.vec-e
                .
                .
```

Fig. 4.12 Modification of statements in subroutine EDCMAG.

```
                .
                .
      CALL EDCMAG(NEGF,N_ELMT,N_ELMT3,N_NODE,N_NODE3,     N_MATDA,N_MATN
     +O,N_MAT,N_GRID,NFTOT,     N_ACTI,N_CURAA,N_CURNA,M_CURNA,IA(MC4),A
     +(NC8),A(NC9),A(NC10),     N_PASS,N_CONST,IA(MC7),A(NCQ),     IA(M3
     +),IA(M4),IA(M6),IA(M16),IA(M15),N_TIME,N_PHAS,     A(N2),A(NA6),
     +   A(NC16),A(NC17),A(NC18),A(NC19),A(NC20),A(NC21),     A(NC22),A(
     +NC23),A(NC24),     A(N43),A(N47),A(NC30),     A(N40),A(N41),A(N42)
     +,A(N39),A(NC28),A(NC29),     IA(MC8),A(NC14),     IA(M5),A(N1),A(N
     +33),A(N12),     A(N46),A(NC27),A(ND1),     N_TIM3,IA(MD1),A(ND2),A
c.org     +(ND3),A(ND4),A(ND5),A(ND6) )
c.vec-s
     +(ND3),A(ND4),A(ND5),A(ND6),
     +IA(M_VW1),IA(M_VW11),IA(M_VW2),IA(M_VW22) )
c.vec-e
                .
                .
```

Fig. 4.13 Modification of call statement for subroutine EDCMAG.

```
s            DO IE2 = 1, N_ELMT
v               IF ( (RR2(IE2).GE.R12).AND.(RR2(IE2).LT.R07) ) THEN
m                  BBB(1,IE2) = 0.0D0
s                  BBB(2,IE2) = 0.0D0
s                  BBB(3,IE2) = 0.0D0
s2                 DO J1 = 5, 11
v2                    XX = XYZ_GAUS(1,J1,IE1)
v2                    YY = XYZ_GAUS(2,J1,IE1)
v2                    ZZ = XYZ_GAUS(3,J1,IE1)
v2                    WW = WH(J1)
v2                    DO J2 = 5, 11
v2                       XXX = XX-XYZ_GAUS2(1,J2,IE2)
v2                       YYY = YY-XYZ_GAUS2(2,J2,IE2)
v2                       ZZZ = ZZ-XYZ_GAUS2(3,J2,IE2)
v2                       RRR = XXX**2 + YYY**2 + ZZZ**2
v2                       RRI = WW*WH(J2)/(RRR*DSQRT(RRR))
v2                       BBB(1,IE2) = BBB(1,IE2) + XXX * RRI
v2                       BBB(2,IE2) = BBB(2,IE2) + YYY * RRI
v2                       BBB(3,IE2) = BBB(3,IE2) + ZZZ * RRI
v2                    END DO
v2                 END DO
v               ELSE IF( RR2(IE2).LT.R12 ) THEN
m                  BBB(1,IE2) = 0.0D0
s                  BBB(2,IE2) = 0.0D0
s                  BBB(3,IE2) = 0.0D0
s2                 DO J1 = 12, 23
v2                    XX = XYZ_GAUS(1,J1,IE1)
v2                    YY = XYZ_GAUS(2,J1,IE1)
v2                    ZZ = XYZ_GAUS(3,J1,IE1)
v2                    WW = WH(J1)
v2                    DO J2 = 12, 23
v2                       XXX = XX-XYZ_GAUS2(1,J2,IE2)
v2                       YYY = YY-XYZ_GAUS2(2,J2,IE2)
v2                       ZZZ = ZZ-XYZ_GAUS2(3,J2,IE2)
v2                       RRR = XXX**2 + YYY**2 + ZZZ**2
v2                       RRI = WW*WH(J2)/(RRR*DSQRT(RRR))
v2                       BBB(1,IE2) = BBB(1,IE2) + XXX * RRI
v2                       BBB(2,IE2) = BBB(2,IE2) + YYY * RRI
v2                       BBB(3,IE2) = BBB(3,IE2) + ZZZ * RRI
v2                    END DO
v2                 END DO
v               END IF
v            END DO
```

Fig. 4.14 Before modification of vectorized DO loops in subroutine FLUXN.

```
      SUBROUTINE FLUXN(N_ELMT,N_ELMT3,N_NODE,NODE_ELMT,XYZ_NODE,      ARE
c.org     +A_ELMT,XYZ_GAUS,RR2,XYZ_GAUS2,BBB,EDY,BNODALL)
c.vec-s
     +A_ELMT,XYZ_GAUS,RR2,XYZ_GAUS2,BBB,EDY,BNODALL,
     +i07v,i12v)
c.vec-e

c.vec-s
      integer*4 i07,i12,i07v(N_ELMT),i12v(N_ELMT)
c.vec-e

m         i07=0
s         i12=0
v         do ie2=1,n_elmt
v         if( (RR2(IE2).GE.R12).AND.(RR2(IE2).LT.R07) ) then
v           i07=i07+1
v           i07v(i07)=ie2
v           BBB(1,IE2) = 0.0D0
v           BBB(2,IE2) = 0.0D0
v           BBB(3,IE2) = 0.0D0
v         else if( RR2(IE2).LT.R12 ) then
v           i12=i12+1
v           i12v(i12)=ie2
v           BBB(1,IE2) = 0.0D0
v           BBB(2,IE2) = 0.0D0
v           BBB(3,IE2) = 0.0D0
v         endif
v         end do
```

Fig. 4.15 After modification of vectorized DO loops in subroutine FLUXN(1/2).

```
s         do j1=5,11
v           XX = XYZ_GAUS(1,J1,IE1)
v           YY = XYZ_GAUS(2,J1,IE1)
v           ZZ = XYZ_GAUS(3,J1,IE1)
s         do j2=5,11
    *vocl loop,novrec(bbb)
v         do ie2=1,i07
v           XXX = XX-XYZ_GAUS2(1,J2,i07v(ie2))
v           YYY = YY-XYZ_GAUS2(2,J2,i07v(ie2))
v           ZZZ = ZZ-XYZ_GAUS2(3,J2,i07v(ie2))
v           RRR = XXX**2 + YYY**2 + ZZZ**2
v           RRI = WW*WH(J2)/(RRR*DSQRT(RRR))
v           BBB(1,i07v(ie2)) = BBB(1,i07v(ie2)) + XXX * RRI
v           BBB(2,i07v(ie2)) = BBB(2,i07v(ie2)) + YYY * RRI
v           BBB(3,i07v(ie2)) = BBB(3,i07v(ie2)) + ZZZ * RRI
v         end do
s         end do
v         end do

s         do j1=12,23
v           XX = XYZ_GAUS(1,J1,IE1)
v           YY = XYZ_GAUS(2,J1,IE1)
v           ZZ = XYZ_GAUS(3,J1,IE1)
s         do j2=12,23
    *vocl loop,novrec(bbb)
v         do ie2=1,i12
v           XXX = XX-XYZ_GAUS2(1,J2,i12v(ie2))
v           YYY = YY-XYZ_GAUS2(2,J2,i12v(ie2))
v           ZZZ = ZZ-XYZ_GAUS2(3,J2,i12v(ie2))
v           RRR = XXX**2 + YYY**2 + ZZZ**2
v           RRI = WW*WH(J2)/(RRR*DSQRT(RRR))
v           BBB(1,i12v(ie2)) = BBB(1,i12v(ie2)) + XXX * RRI
v           BBB(2,i12v(ie2)) = BBB(2,i12v(ie2)) + YYY * RRI
v           BBB(3,i12v(ie2)) = BBB(3,i12v(ie2)) + ZZZ * RRI
v         end do
s         end do
v         end do
```

Fig. 4.15 After modification of vectorized DO loops in subroutine FLUXN(2/2).

```
m         DO I = 1, NTEN
v            XO = XYZ(1,I)
v            YO = XYZ(2,I)
v            ZO = XYZ(3,I)
s            AA2 = (R2-R1)*(Z2-Z1)/PI
s            RRR = (R2+R1)/2.0D0
s            CCC = PI*BTESLA*(RRR+DSQRT(RRR**2-AA2))/UYM
m            CURTOTAL = 4.0D0*PI*(R2-R1)*BTESLA/(UYM*LOG(R2/R1))
v            CURTOTAL = CURTOTAL/2.0D0
m            CURZ2 = CURTOTAL/(2.0D0*PI*R2)
m            CURZ1 = CURTOTAL/(2.0D0*PI*R1)
v            CURR  = CURTOTAL/(2.0D0*PI)
m            RO = DSQRT(XO**2.0D0+YO**2.0D0)
v            TH = DATAN2(YO,XO)
s            NBUN = 16
m            Z = Z2-ZO
s            CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R2,RO,Z)     ← A
m            Z = Z1-ZO
s            CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R2,RO,Z)     ← A
m            AZ2 = (FNC2-FNC1)*UYM*CURZ2/(4.0D0*PI)
s            AZ2 = AZ2
s            NBUN = 16
s            Z = Z2-ZO
s            CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R1,RO,Z)     ← A
s            Z = Z1-ZO
s            CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R1,RO,Z)     ← A
m            AZ1 = (FNC2-FNC1)*UYM*CURZ1/(4.0D0*PI)
v            AZ1 = -AZ1
s            NBUN = 16
s            Z = Z2-ZO
s            CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,RO,Z)    ← A
s            CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,RO,Z)    ← A
m            AR2X = (FNC2-FNC1)*UYM*CURR/(4.0D0*PI)
v            AR2X = -AR2X
s            NBUN = 16
s            Z = Z1-ZO
s            CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,RO,Z)    ← A
s            CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,RO,Z)    ← A
m            AR1X = (FNC2-FNC1)*UYM*CURR/(4.0D0*PI)
v            AR1X = AR1X
v            AR = AR2X + AR1X
v            AZ = AZ2 + AZ1
v            AX = AR*DCOS(TH)
v            AY = AR*DSIN(TH)
v            AAA(1,I) = AX * AB
v            AAA(2,I) = AY * AB
v            AAA(3,I) = AZ * AB
v         END DO
```

Fig. 4.16 Original DO loop in subroutine AHALO.

```
      SUBROUTINE INTGS(GAUSS,N,C,D,FNC,R,RO,Z)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      EXTERNAL FNC
      COMMON / GAUSS2 / XD(25,25),WD(25,25)
      IF(N.LE.0) THEN
         K = 1
      ELSE IF(N.LE.6) THEN
         K = N
      ELSE IF(N.LE.7) THEN
         K = 8
      ELSE IF(N.LE.9) THEN
         K =10
      ELSE IF(N.LE.11) THEN
         K =12
      ELSE
         K =16
      END IF
      C1=(D+C)/2.0D0
      C2=(D-C)/2.0D0
      S=0.D0
      DO I = 1, K
         U1 = C1-C2*XD(I,K)
         U2 = C1+C2*XD(I,K)
         S = S + WD(I,K)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )    ← B
      END DO
      GAUSS=C2*S
      RETURN
      END
```

Fig. 4.17 Original subroutine INTGS.

```
      FUNCTION FNAZ(TH,R,X,Z)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      FNAZ = R*LOG( DSQRT(R**2 + X**2 + Z**2 - 2.0D0*R*X*DCOS(TH)) + Z)
      END

      FUNCTION FNARX(TH,R,X,Z)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      FNARX = DCOS(TH)        *LOG(DSQRT(R**2+X**2+Z**2-2.0D0*R*X*DCOS(T
     +H))+R-X*DCOS(TH) )
      END
```

Fig. 4.18 Original functions FNAZ and FNARX.

```
      SUBROUTINE INTGS(GAUSS,N,C,D,FNC,R,RO,Z)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      EXTERNAL FNC
      COMMON / GAUSS2 / XD(25,25),WD(25,25)
      IF(N.LE.0) THEN
         K = 1
      ELSE IF(N.LE.6) THEN
         K = N
      ELSE IF(N.LE.7) THEN
         K = 8
      ELSE IF(N.LE.9) THEN
         K =10
      ELSE IF(N.LE.11) THEN
         K =12
      ELSE
         K =16
      END IF
      C1=(D+C)/2.0D0
      C2=(D-C)/2.0D0
      S=0.D0
         U1 = C1-C2*XD(I,1)
         U2 = C1+C2*XD(I,1)
         S = S + WD(I,1)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,2)
         U2 = C1+C2*XD(I,2)
         S = S + WD(I,2)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,3)
         U2 = C1+C2*XD(I,3)
         S = S + WD(I,3)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,4)
         U2 = C1+C2*XD(I,4)
         S = S + WD(I,4)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,5)
         U2 = C1+C2*XD(I,5)
         S = S + WD(I,5)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,6)
         U2 = C1+C2*XD(I,6)
         S = S + WD(I,6)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,7)
         U2 = C1+C2*XD(I,7)
         S = S + WD(I,7)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,8)
         U2 = C1+C2*XD(I,8)
```

Fig. 4.19 After Integration DO loop in subroutine INTGS(1/2).

```
      S = S + WD(I,8)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,9)
      U2 = C1+C2*XD(I,9)
      S = S + WD(I,9)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,10)
      U2 = C1+C2*XD(I,10)
      S = S + WD(I,10)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,11)
      U2 = C1+C2*XD(I,11)
      S = S + WD(I,11)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,12)
      U2 = C1+C2*XD(I,12)
      S = S + WD(I,12)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,13)
      U2 = C1+C2*XD(I,13)
      S = S + WD(I,13)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,14)
      U2 = C1+C2*XD(I,14)
      S = S + WD(I,14)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,15)
      U2 = C1+C2*XD(I,15)
      S = S + WD(I,15)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
      U1 = C1-C2*XD(I,16)
      U2 = C1+C2*XD(I,16)
      S = S + WD(I,16)*( FNC(U1,R,RO,Z) + FNC(U2,R,RO,Z) )
    GAUSS=C2*S
    RETURN
    END
```

Fig. 4.19 After Integration DO loop in subroutine INTGS(2/2).

```
      DO I = 1, NTEN
         XO = XYZ(1,I)
         YO = XYZ(2,I)
         ZO = XYZ(3,I)
         AA2 = (R2-R1)*(Z2-Z1)/PI
         RRR = (R2+R1)/2.0D0
                .
                .
                .
         Z = Z2-ZO
c        CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R2,RO,Z)
         CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R2,RO,Z,1) ← フラグ 1
         Z = Z1-ZO
c        CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R2,RO,Z)
         CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R2,RO,Z,1) ← フラグ 1
                .
                .
                .
c        CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,RO,Z)
         CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,RO,Z,2) ← フラグ 2
c        CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,RO,Z)
         CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,RO,Z,2) ← フラグ 2
         AR1X = (FNC2-FNC1)*UYM*CURR/(4.0D0*PI)
                .
                .
                .
      END DO
```

Fig. 4.20 Modification of call statement for subroutine INTGS in subroutine AHALO.

```
      SUBROUTINE INTGS(GAUSS,N,C,D,FNC,R,RO,Z,IF_TYPE) ← IF_TYPE：フラグ
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      EXTERNAL FNC
      COMMON / GAUSS2 / XD(25,25),WD(25,25)
                .
                .
                .
      IF(IF_TYPE.EQ.1) THEN          ← FNAZ を呼び出す場合
         U1 = C1-C2*XD(I,1)
         U2 = C1+C2*XD(I,1)
         S = S + WD(I,1)*( FNAZ(U1,R,RO,Z) + FNAZ(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,2)
         U2 = C1+C2*XD(I,2)
         S = S + WD(I,2)*( FNAZ(U1,R,RO,Z) + FNAZ(U2,R,RO,Z) )
                .
                .
                .
      ELSE IF(IF_TYPE.EQ.2) THEN    ← FNARX を呼び出す場合
         U1 = C1-C2*XD(I,1)
         U2 = C1+C2*XD(I,1)
         S = S + WD(I,1)*( FNARX(U1,R,RO,Z) + FNARX(U2,R,RO,Z) )
         U1 = C1-C2*XD(I,2)
         U2 = C1+C2*XD(I,2)
         S = S + WD(I,2)*( FNARX(U1,R,RO,Z) + FNARX(U2,R,RO,Z) )

                .
                .
                .
      ELSE
      ENDIF

      GAUSS=C2*S
      RETURN
      END
```

Fig. 4.21 Modification of subroutine INTGS.

```
      SUBROUTINE INTGS(GAUSS,N,C,D,FNC,R,RO,Z,IF_TYPE)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      EXTERNAL FNC
      COMMON / GAUSS2 / XD(25,25),WD(25,25)

      FNAZ(VA,VB,VC,VD)=VB*LOG(DSQRT(VB**2+VC**2+VD**2-2.0D0*VB*VC*DCOS(
     &VA))+VD)
      FNARX(VE,VF,VG,VH)=DCOS(VE)*LOG(DSQRT(VF**2+VG**2+VH**2-2.0D0*VF*V
     &G*DCOS(T+H))+VF-VG*DCOS(VE))

                .
                .
                .
```

Fig. 4.22 Declaration of statement functions FNAZ and FNARX.

```
v          DO I = 1, NTEN
v             XO = XYZ(1,I)
v             YO = XYZ(2,I)
v             ZO = XYZ(3,I)
v             AA2 = (R2-R1)*(Z2-Z1)/PI
v             RRR = (R2+R1)/2.0D0
v             CCC = PI*BTESLA*(RRR+DSQRT(RRR**2-AA2))/UYM
v             CURTOTAL = 4.0D0*PI*(R2-R1)*BTESLA/(UYM*LOG(R2/R1))
v             CURTOTAL = CURTOTAL/2.0D0
v             CURZ2 = CURTOTAL/(2.0D0*PI*R2)
v             CURZ1 = CURTOTAL/(2.0D0*PI*R1)
v             CURR  = CURTOTAL/(2.0D0*PI)
v             RO = DSQRT(XO**2.0D0+YO**2.0D0)
v             TH = DATAN2(YO,XO)
v             NBUN = 16
v             Z = Z2-ZO
     c        CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R2,RO,Z)
  i           CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R2,RO,Z,1)
v             Z = Z1-ZO
     c        CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R2,RO,Z)
  i           CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R2,RO,Z,1)
v             AZ2 = (FNC2-FNC1)*UYM*CURZ2/(4.0D0*PI)
v             AZ2 = AZ2
v             NBUN = 16
v             Z = Z2-ZO
     c        CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R1,RO,Z)
  i           CALL INTGS(FNC2,NBUN,TH1,TH2,FNAZ,R1,RO,Z,1)
v             Z = Z1-ZO
     c        CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R1,RO,Z)
  i           CALL INTGS(FNC1,NBUN,TH1,TH2,FNAZ,R1,RO,Z,1)
```

Fig. 4.23 Vectorized DO loop in subroutine AHALO(1/2).

```
v            AZ1 = (FNC2-FNC1)*UYM*CURZ1/(4.0D0*PI)
v            AZ1 = -AZ1
v            NBUN = 16
v            Z = Z2-Z0
   c         CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,R0,Z)
 i           CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,R0,Z,2)
   c         CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,R0,Z)
vi           CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,R0,Z,2)
v            AR2X = (FNC2-FNC1)*UYM*CURR/(4.0D0*PI)
v            AR2X = -AR2X
v            NBUN = 16
v            Z = Z1-Z0
   c         CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,R0,Z)
 i           CALL INTGS(FNC2,NBUN,TH1,TH2,FNARX,R2,R0,Z,2)
   c         CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,R0,Z)
vi           CALL INTGS(FNC1,NBUN,TH1,TH2,FNARX,R1,R0,Z,2)
v            AR1X = (FNC2-FNC1)*UYM*CURR/(4.0D0*PI)
v            AR1X = AR1X
v            AR = AR2X + AR1X
v            AZ = AZ2 + AZ1
v            AX = AR*DCOS(TH)
v            AY = AR*DSIN(TH)
v            AAA(1,I) = AX * AB
v            AAA(2,I) = AY * AB
v            AAA(3,I) = AZ * AB
v         END DO
```

Fig. 4.23 Vectorized DO loop in subroutine AHALO(2/2).

```
      SUBROUTINE XMODEX(T00, T11, LJYD, LDLJYD, VECTX, EQMOD,     EIGEX,
     + ETMOD,     N_CURAA, COEF1, M_CURNA, N_CURNA,  TIMECUR1, N_ACTI
     +, KIND_CUR1, IT1,     N_CURAP, COEF2, M_CURNP, N_CURNP,     TIMECU
     +R2, N_PASS, KIND_CUR2, IT2 )


         DO J = 1, N_ACTI
            JJ = KIND_CUR1(2,J)
            IF(KIND_CUR1(1,J).EQ.1) THEN
s8             DO I = 1, LJYD
m8                ALAMDA = VECTX(I)
m8                ETMOD(I,J) = EQMOD(I,J)*XINT(T11,ALAMDA,COEF1(1,JJ))
v8             END DO
            ELSE


         DO J = 1, N_PASS
            J2 = N_ACTI + J
            JJ = KIND_CUR2(2,J)
            IF(KIND_CUR2(1,J).EQ.1) THEN
s6             DO I = 1, LJYD
m6                ALAMDA = VECTX(I)
m6                JD = (LJYD-N_PASS) + J
m6                ETMOD(I,J2) = EIGEX(JD,I)*XINT2(T11,ALAMDA,COEF2(1,JJ))
v6             END DO
            ELSE
            .
            .
            .
```

Fig. 4.24 Original DO loops in subroutine XMODEX.

```
      FUNCTION XINT(T,ALAMDA,CONST)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION CONST(9)
      GO=CONST(1)
      G1=CONST(2)
      G2=CONST(3)
      G3=CONST(4)
      G4=CONST(5)
      G5=CONST(6)
      TAU=CONST(7)
      OMEGA1=CONST(8)
      OMEGA2=CONST(9)
      IF(ALAMDA.LT.1.0D-10) XINT=0.0D0
      IF(ALAMDA.LT.1.0D-10) RETURN
      EXPTL=DEXP(-T/ALAMDA)
      G=G1+G3+G5-GO
      X=G*EXPTL
      IF(G2.EQ.0) GO TO 10
      X=X+G2*ALAMDA*(1.0D0-EXPTL)
   10 CONTINUE
      IF(G3.EQ.0) GO TO 20
      X=X+(DEXP(-T/TAU)-EXPTL)/(1.D0-TAU/ALAMDA)*G3
   20 CONTINUE
      IF(G4.EQ.0) GO TO 30
      ALOME=ALAMDA*OMEGA1
      OMET=OMEGA1*T
      X=X+ALOME/(1.D0+ALOME*ALOME)*(ALOME*DSIN(OMET)+DCOS(OMET)-EXPTL)*G
     +4
   30 CONTINUE
      IF(G5.EQ.0) GO TO 40
      ALOME=ALAMDA*OMEGA2
      OMET=OMEGA2*T
      ALOME2=ALOME*ALOME
      X=X+ALOME2/(1.D0+ALOME2)*(DCOS(OMET)-DSIN(OMET)/ALOME-EXPTL)*G5
   40 CONTINUE
      XINT=-X/ALAMDA
      RETURN
      END
```

Fig. 4.25 Original function XINT.

```
      FUNCTION XINT2(T,ALAMDA,CONST)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION CONST(9)
      G0=CONST(1)
      G1=CONST(2)
      G2=CONST(3)
      G3=CONST(4)
      G4=CONST(5)
      G5=CONST(6)
      TAU=CONST(7)
      OMEGA1=CONST(8)
      OMEGA2=CONST(9)
      IF(ALAMDA.LT.1.0D-10) THEN
         XINT2=0.0D0
         RETURN
      END IF
      EXPTL=DEXP(-T/ALAMDA)
      X = 0.0D0
      IF(G1.NE.0) THEN
         X = X + G1*ALAMDA*(1.0D0-EXPTL)
      END IF
      IF(G2.NE.0) THEN
         X = X + G2*ALAMDA*(T-ALAMDA+ALAMDA*EXPTL)
      END IF
      IF(G3.NE.0) THEN
         X = X + G3*ALAMDA*TAU*(DEXP(-T/TAU)-EXPTL)/(ALAMDA-TAU)
      END IF
      IF(G4.NE.0) THEN
         ALOME=ALAMDA*OMEGA1
         OMET=OMEGA1*T
         X = X + G4*ALAMDA/(1.0D0+ALOME**2)*          (-ALOME*DCOS(OMET)+D
     +SIN(OMET)-ALOME*EXPTL)
      END IF
      IF(G5.NE.0) THEN
         ALOME=ALAMDA*OMEGA2
         OMET=OMEGA2*T
         X = X + G5*ALAMDA/(1.0D0+ALOME**2)*          (DSIN(OMET)+ALOME*DC
     +OS(OMET)-EXPTL)
      END IF
      XINT2 = X/ALAMDA
      RETURN
      END
```

Fig. 4.26 Original function XINT2.

```
      FUNCTION XINT(T,ALAMDA,CONST)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION CONST(9)
      GO=CONST(1)
      G1=CONST(2)
      G2=CONST(3)
      G3=CONST(4)
      G4=CONST(5)
      G5=CONST(6)
      TAU=CONST(7)
      OMEGA1=CONST(8)
      OMEGA2=CONST(9)
c     IF(ALAMDA.LT.1.0D-10) XINT=0.0D0

      IF(ALAMDA.LT.1.0D-10) then
        XINT=0.0D0
      else

      EXPTL=DEXP(-T/ALAMDA)
      G=G1+G3+G5-GO
      X=G*EXPTL

      if(G2.eq.0.and.G3.eq.0.and.G4.eq.0.and.G5.eq.0) then

        XINT=-X/ALAMDA

      elseif(G2.eq.0.and.G3.eq.0.and.G4.eq.0) then

        ALOME=ALAMDA*OMEGA2
        OMET=OMEGA2*T
        ALOME2=ALOME*ALOME
        X=X+ALOME2/(1.DO+ALOME2)*(DCOS(OMET)-DSIN(OMET)/ALOME-EXPTL)*G5
        XINT=-X/ALAMDA

      elseif(G2.eq.0.and.G3.eq.0) then

        ALOME=ALAMDA*OMEGA1
        OMET=OMEGA1*T
        X=X+ALOME/(1.DO+ALOME*ALOME)*(ALOME*DSIN(OMET)+DCOS(OMET)-EXPTL)
     &*G4
        ALOME=ALAMDA*OMEGA2
        OMET=OMEGA2*T
        ALOME2=ALOME*ALOME
        X=X+ALOME2/(1.DO+ALOME2)*(DCOS(OMET)-DSIN(OMET)/ALOME-EXPTL)*G5
        XINT=-X/ALAMDA
```

Fig. 4.27 Modification of function XINT(1/2).

```
      elseif(G2.eq.0) then

        X=X+(DEXP(-T/TAU)-EXPTL)/(1.D0-TAU/ALAMDA)*G3
        ALOME=ALAMDA*OMEGA1
        OMET=OMEGA1*T
        X=X+ALOME/(1.D0+ALOME*ALOME)*(ALOME*DSIN(OMET)+DCOS(OMET)-EXPTL)
     &*G4
        ALOME=ALAMDA*OMEGA2
        OMET=OMEGA2*T
        ALOME2=ALOME*ALOME
        X=X+ALOME2/(1.D0+ALOME2)*(DCOS(OMET)-DSIN(OMET)/ALOME-EXPTL)*G5
        XINT=-X/ALAMDA

      else

        X=X+G2*ALAMDA*(1.0D0-EXPTL)
        X=X+(DEXP(-T/TAU)-EXPTL)/(1.D0-TAU/ALAMDA)*G3
        ALOME=ALAMDA*OMEGA1
        OMET=OMEGA1*T
        X=X+ALOME/(1.D0+ALOME*ALOME)*(ALOME*DSIN(OMET)+DCOS(OMET)-EXPTL)
     &*G4
        ALOME=ALAMDA*OMEGA2
        OMET=OMEGA2*T
        ALOME2=ALOME*ALOME
        X=X+ALOME2/(1.D0+ALOME2)*(DCOS(OMET)-DSIN(OMET)/ALOME-EXPTL)*G5
        XINT=-X/ALAMDA

      endif

      endif

      return
      END
```

Fig. 4.27 Modification of function XINT(2/2).

```
      SUBROUTINE XMODEX(T00, T11, LJYD, LDLJYD, VECTX, EQMOD,     EIGEX,
     + ETMOD,     N_CURAA, COEF1, M_CURNA, N_CURNA,     TIMECUR1, N_ACTI
     +, KIND_CUR1, IT1,     N_CURAP, COEF2, M_CURNP, N_CURNP,     TIMECU
     +R2, N_PASS, KIND_CUR2, IT2 )
                .
                .
           DO J = 1, N_ACTI
              JJ = KIND_CUR1(2,J)
              IF(KIND_CUR1(1,J).EQ.1) THEN
v               DO I = 1, LJYD
v                  ALAMDA = VECTX(I)
vi                 ETMOD(I,J) = EQMOD(I,J)*XINT(T11,ALAMDA,COEF1(1,JJ))
v               END DO
              ELSE
                 .
                 .
           DO J = 1, N_PASS
              J2 = N_ACTI + J
              JJ = KIND_CUR2(2,J)
              IF(KIND_CUR2(1,J).EQ.1) THEN
v                DO I = 1, LJYD
v                   ALAMDA = VECTX(I)
v                   JD = (LJYD-N_PASS) + J
vi                  ETMOD(I,J2) = EIGEX(JD,I)*XINT2(T11,ALAMDA,COEF2(1,JJ))
v                END DO
              ELSE
                 .
                 .
```

Fig. 4.28 Vectorized DO loops in subroutine XMODEX.

```
      SUBROUTINE CRTBBB( N_ELMT, N_ELMT3, BNODALL )
                .
                .
s        DO 10 IR=1,K_CRE_RZ
v           ROP11=ROTAT(1,1,IR)
v           ROP12=ROTAT(1,2,IR)
                .
                .
            DO 20 IS=1,N_CRE
               IF( (IR.EQ.1).AND.(IS.EQ.1) ) THEN
                  GO TO 21
               END IF
                .
                .
               IE = IE + N_ELMT          ← A
s4             DO 40 I=1,N_ELMT
s4                IEI = IE + I           ← B
s4                BNODALL(1,IEI,1)=ROPB11*BNODALL(1,I,1)+
        +         ROPB12*BNODALL(2,I,1)
s4                BNODALL(2,IEI,1)=ROPB21*BNODALL(1,I,1)+
        +         ROPB22*BNODALL(2,I,1)
s4                BNODALL(3,IEI,1)=ROPB33*BNODALL(3,I,1)
s4  40         CONTINUE
    21         CONTINUE
    20      CONTINUE
v   10    CONTINUE
          RETURN
          END
```

Fig. 4.29 Original DO loop(DO 40) in subroutine CRTBBB.

```
      SUBROUTINE CRTBBB( N_ELMT, N_ELMT3, BNODALL )
                .
                .
s        DO 10 IR=1,K_CRE_RZ
v           ROP11=ROTAT(1,1,IR)
v           ROP12=ROTAT(1,2,IR)
                .
                .
            DO 20 IS=1,N_CRE
               IF( (IR.EQ.1).AND.(IS.EQ.1) ) THEN
                  GO TO 21
               END IF
                .
                .
               IE = IE + N_ELMT
   *vocl loop,novrec(bnodall)
v              DO 40 I=1,N_ELMT
v                 IEI = IE + I
v                 BNODALL(1,IEI,1)=ROPB11*BNODALL(1,I,1)+
        +          ROPB12*BNODALL(2,I,1)
v                 BNODALL(2,IEI,1)=ROPB21*BNODALL(1,I,1)+
        +          ROPB22*BNODALL(2,I,1)
v                 BNODALL(3,IEI,1)=ROPB33*BNODALL(3,I,1)
v   40         CONTINUE
    21         CONTINUE
    20      CONTINUE
v   10   CONTINUE
         RETURN
         END
```

Fig. 4.30 Vectorized DO loop(DO 40) in subroutine CRTBBB.

```
      SUBROUTINE CRTELEM(N_NODE,N_ELMT,N_NODE3,N_ELMT3,N_GRID,      L_NOD
     +E,L_ELMT,MAT_ELMT,NODE_ELMT,XYZ_NODE,AREA_ELMT)
               .
               .
      IN = N_NODE          ← A
      IE = N_ELMT          ← B
               .
               .
s        DO 10 IR=1,K_CRE_RZ
s           ROP11=ROTAT(1,1,IR)
s           ROP12=ROTAT(1,2,IR)
s           ROP21=ROTAT(2,1,IR)
               .
               .
m           DO 30 I=1,N_NODE
v              IN = IN + 1
s6             XYZ_NODE(1,IN)=ROPP11*XYZ_NODE(1,I)+ROPP12*XYZ_NODE(2,I)
s6             XYZ_NODE(2,IN)=ROPP21*XYZ_NODE(1,I)+ROPP22*XYZ_NODE(2,I)
s6             XYZ_NODE(3,IN)=ROPP33*XYZ_NODE(3,I)
s6             L_NODE(IN) = L_NODE(I) + JNODEMAX*IC
v   30      CONTINUE
m           DO 40 I=1,N_ELMT
v              IE = IE + 1
s3             L_ELMT(IE) = L_ELMT(I) + JELEMMAX*IC
s3             MAT_ELMT(IE) = MAT_ELMT(I)
s3             AREA_ELMT(IE) = AREA_ELMT(I)
s3             NODE_ELMT(1,IE) = NODE_ELMT(1,I) + IC*N_NODE
s3             NODE_ELMT(2,IE) = NODE_ELMT(2,I) + IC*N_NODE
s3             NODE_ELMT(3,IE) = NODE_ELMT(3,I) + IC*N_NODE
v   40      CONTINUE
    21      CONTINUE
    20      CONTINUE
s   10   CONTINUE
               .
               .
```

Fig. 4.31 Original DO loops in subroutine CRTELEM.

```
      SUBROUTINE CRTELEM(N_NODE,N_ELMT,N_NODE3,N_ELMT3,N_GRID,      L_NOD
     +E,L_ELMT,MAT_ELMT,NODE_ELMT,XYZ_NODE,AREA_ELMT)
                .
                .
      IN = N_NODE          ← A
      IE = N_ELMT          ← B
                .
                .
s        DO 10 IR=1,K_CRE_RZ
s           ROP11=ROTAT(1,1,IR)
s           ROP12=ROTAT(1,2,IR)
s           ROP21=ROTAT(2,1,IR)
                .
                .
   *vocl loop,novrec(xyz_node,l_node))
v           DO 30 I=1,N_NODE
v              IN = IN + 1
v              XYZ_NODE(1,IN)=ROPP11*XYZ_NODE(1,I)+ROPP12*XYZ_NODE(2,I)
v              XYZ_NODE(2,IN)=ROPP21*XYZ_NODE(1,I)+ROPP22*XYZ_NODE(2,I)
v              XYZ_NODE(3,IN)=ROPP33*XYZ_NODE(3,I)
v              L_NODE(IN) = L_NODE(I) + JNODEMAX*IC
v   30      CONTINUE
   *vocl loop,novrec(l_elmt,mat_elmt,area_elmt,node_elmt)
v           DO 40 I=1,N_ELMT
v              IE = IE + 1
v              L_ELMT(IE) = L_ELMT(I) + JELEMMAX*IC
v              MAT_ELMT(IE) = MAT_ELMT(I)
v              AREA_ELMT(IE) = AREA_ELMT(I)
v              NODE_ELMT(1,IE) = NODE_ELMT(1,I) + IC*N_NODE
v              NODE_ELMT(2,IE) = NODE_ELMT(2,I) + IC*N_NODE
v              NODE_ELMT(3,IE) = NODE_ELMT(3,I) + IC*N_NODE
v   40      CONTINUE
    21      CONTINUE
    20      CONTINUE
s   10   CONTINUE
                .
                .
```

Fig. 4.32 Vectorized DO loops in subroutine CRTBBB.

```
      SUBROUTINE ELEMNT(N_ELMT,N_ELMT2,N_NODE,NODE_ELMT,      XYZ_NODE,AR
     +EA_ELMT,EELMT)
                .
                .
v         DO N=1,N_ELMT
s          DO K=1,3
v            L=K+2
v            M=K+1
v            IF(L.GT.3) L=L-3
v            IF(M.GT.3) M=M-3
v            L=NODE_ELMT(L,N)
v            M=NODE_ELMT(M,N)
v3           DO I=1,3
v3             EELMT(I,K,N)=(XYZ_NODE(I,L)-XYZ_NODE(I,M))/2.DO/AREA_ELMT(
          +N)
v3           END DO
s           END DO
v          END DO
           RETURN
                .
                .
```

Fig. 4.33 Before modification of vectorized DO loops in subroutine ELEMNT.

```
      SUBROUTINE ELEMNT(N_ELMT,N_ELMT2,N_NODE,NODE_ELMT,      XYZ_NODE,AR
     +EA_ELMT,EELMT)
            .
            .
v         DO N=1,N_ELMT
v           L=1+2
v           M=1+1
v           IF(L.GT.3) L=L-3
v           IF(M.GT.3) M=M-3
v           L=NODE_ELMT(L,N)
v           M=NODE_ELMT(M,N)
v            EELMT(1,1,N)=(XYZ_NODE(1,L)-XYZ_NODE(1,M))/2.D0/AREA_ELMT(N)
v            EELMT(2,1,N)=(XYZ_NODE(2,L)-XYZ_NODE(2,M))/2.D0/AREA_ELMT(N)
v            EELMT(3,1,N)=(XYZ_NODE(3,L)-XYZ_NODE(3,M))/2.D0/AREA_ELMT(N)
v           L=2+2
v           M=2+1
v           IF(L.GT.3) L=L-3
v           IF(M.GT.3) M=M-3
v           L=NODE_ELMT(L,N)
v           M=NODE_ELMT(M,N)
v            EELMT(1,2,N)=(XYZ_NODE(1,L)-XYZ_NODE(1,M))/2.D0/AREA_ELMT(N)
v            EELMT(2,2,N)=(XYZ_NODE(2,L)-XYZ_NODE(2,M))/2.D0/AREA_ELMT(N)
v            EELMT(3,2,N)=(XYZ_NODE(3,L)-XYZ_NODE(3,M))/2.D0/AREA_ELMT(N)
v           L=3+2
v           M=3+1
v           IF(L.GT.3) L=L-3
v           IF(M.GT.3) M=M-3
v           L=NODE_ELMT(L,N)
v           M=NODE_ELMT(M,N)
v            EELMT(1,3,N)=(XYZ_NODE(1,L)-XYZ_NODE(1,M))/2.D0/AREA_ELMT(N)
v            EELMT(2,3,N)=(XYZ_NODE(2,L)-XYZ_NODE(2,M))/2.D0/AREA_ELMT(N)
v            EELMT(3,3,N)=(XYZ_NODE(3,L)-XYZ_NODE(3,M))/2.D0/AREA_ELMT(N)
v         END DO
          RETURN
              .
              .
```

Fig. 4.34 After modification of vectorized DO loop in subroutine ELEMNT.

```
      SUBROUTINE INTPNT3(N_ELMT,N_ELMT3,N_NODE,NODE_ELMT,     XYZ_NODE,X
     +YZ_GAUS)
               .
               .
          IRS = 0                          ← A
s         DO IR=1,K_CRE_RZ
v            ROP11=ROTAT(1,1,IR)
v            ROP12=ROTAT(1,2,IR)
v            ROP21=ROTAT(2,1,IR)
v            ROP22=ROTAT(2,2,IR)
             DO IS=1,N_CRE
                ROPP11=ROP11*CRE_LOC(1,IS)
                ROPP12=ROP12*CRE_LOC(2,IS)
                ROPP21=ROP21*CRE_LOC(1,IS)
                ROPP22=ROP22*CRE_LOC(2,IS)
                ROPP33=CRE_LOC(3,IS)
                IRS = IRS + 1              ← A
                IF( IRS.NE.1 ) THEN        ← B
s                  DO L = 1, 23
s6                    DO IE1 = 1, N_ELMT
s6                       IE2 = IE1 + N_ELMT*(IRS-1)   ← C
s6                       XYZ_GAUS(1,L,IE2) = ROPP11*XYZ_GAUS(1,L,IE1) +
        +                                    ROPP12*XYZ_GAUS(2,L,IE1)
s6                       XYZ_GAUS(2,L,IE2) = ROPP21*XYZ_GAUS(1,L,IE1) +
        +                                    ROPP22*XYZ_GAUS(2,L,IE1)
s6                       XYZ_GAUS(3,L,IE2) = ROPP33*XYZ_GAUS(3,L,IE1)
s6                    END DO
s                  END DO
                END IF
             END DO
v         END DO
          RETURN
          END
```

Fig. 4.35 Original DO loops in subroutine INTPNT3.

```
                .
                .
                .
          IRS = 0
s         DO IR=1,K_CRE_RZ
v            ROP11=ROTAT(1,1,IR)
v            ROP12=ROTAT(1,2,IR)
v            ROP21=ROTAT(2,1,IR)
v            ROP22=ROTAT(2,2,IR)
             DO IS=1,N_CRE
                ROPP11=ROP11*CRE_LOC(1,IS)
                ROPP12=ROP12*CRE_LOC(2,IS)
                ROPP21=ROP21*CRE_LOC(1,IS)
                ROPP22=ROP22*CRE_LOC(2,IS)
                ROPP33=CRE_LOC(3,IS)
                IRS = IRS + 1
                IF( IRS.NE.1 ) THEN
s2                 DO L = 1, 23
   *vocl loop,novrec(xyz_gaus)
v2                    DO IE1 = 1, N_ELMT
v2                       IE2 = IE1 + N_ELMT*(IRS-1)
v2                       XYZ_GAUS(1,L,IE2) = ROPP11*XYZ_GAUS(1,L,IE1) +
         +                                   ROPP12*XYZ_GAUS(2,L,IE1)
v2                       XYZ_GAUS(2,L,IE2) = ROPP21*XYZ_GAUS(1,L,IE1) +
         +                                   ROPP22*XYZ_GAUS(2,L,IE1)
v2                       XYZ_GAUS(3,L,IE2) = ROPP33*XYZ_GAUS(3,L,IE1)
v2                    END DO
s2                 END DO
                END IF
             END DO
v         END DO
          RETURN
          END
```

Fig. 4.36 Vectorized DO loop in subroutine INTPNT3.

```
      SUBROUTINE INTPNT(N_ELMT,N_ELMT2,N_NODE,NODE_ELMT,XYZ_NODE,      XY
     +Z_GAUS)
               .
               .
s          DO IE = 1, N_ELMT
v             DO J = 1, 3
v                NOD=NODE_ELMT(J,IE)
v3               DO  I = 1, 3
v3                  XYZ(I,J)=XYZ_NODE(I,NOD)
v3               END DO
v             END DO
v             DO L = 1, 23
s3               DO I = 1, 3
v3                  X=0.0D0
v9                  DO J = 1, 3
v9                     X=X+XH(J,L)*XYZ(I,J)
v9                  END DO
v3                  XYZ_GAUS(I,L,IE)=X
s3               END DO
v             END DO
s          END DO
           RETURN
           END
```

Fig. 4.37 Before modification of vectorized DO loops in subroutine INTPNT.

```
      SUBROUTINE INTPNT(N_ELMT,N_ELMT2,N_NODE,NODE_ELMT,XYZ_NODE,      XY
     +Z_GAUS)
               .
               .
s2           DO L = 1, 23
v2         DO IE = 1, N_ELMT

v2            X=0.0D0
v2            X=X+XH(1,L)*XYZ_NODE(1,NODE_ELMT(1,IE))
v2            X=X+XH(2,L)*XYZ_NODE(1,NODE_ELMT(2,IE))
v2            X=X+XH(3,L)*XYZ_NODE(1,NODE_ELMT(3,IE))
v2            XYZ_GAUS(1,L,IE)=X
v2            X=0.0D0
v2            X=X+XH(1,L)*XYZ_NODE(2,NODE_ELMT(1,IE))
v2            X=X+XH(2,L)*XYZ_NODE(2,NODE_ELMT(2,IE))
v2            X=X+XH(3,L)*XYZ_NODE(2,NODE_ELMT(3,IE))
v2            XYZ_GAUS(2,L,IE)=X
v2            X=0.0D0
v2            X=X+XH(1,L)*XYZ_NODE(3,NODE_ELMT(1,IE))
v2            X=X+XH(2,L)*XYZ_NODE(3,NODE_ELMT(2,IE))
v2            X=X+XH(3,L)*XYZ_NODE(3,NODE_ELMT(3,IE))
v2            XYZ_GAUS(3,L,IE)=X

v2         END DO
s2           END DO

           RETURN
           END
```

Fig. 4.38 After modification of Vectorized DO loop in subroutine INTPNT.

```
      SUBROUTINE INDQA(N_ELMT,N_NODE,N_MODE,NJYD,K_NODE,NODE_ELMT,      X
     +YZ_NODE,AREA_ELMT,      EELMT,N_ACTI,ANOD1,QINDU)
                .
                .
s          DO IE=1,N_ELMT
v3            DO I=1,3
s3               NO(I)=NODE_ELMT(I,IE)
m3            END DO
m             S=AREA_ELMT(IE)
v             DO K=1,3
v3               DO J=1,3
v3                  E(J,K)=EELMT(J,K,IE)
v3               END DO
v             END DO
s             DO N=1,N_ACTI                          ← A
v                DO I=1,3
v                   A(I)=0.0D0
v3                  DO K=1,3
v3                     A(I)=A(I)+ANOD1(I,NO(K),N)
v3                  END DO
v                   A(I)=A(I)/3.0D0
v                END DO
 3               DO I=1,3
 3                  NN=K_NODE(NO(I))
 3                  IF(NN.EQ.0) GO TO 100
 3                  X=0.D0
 3                  DO J=1,3
 3                     X=X+E(J,I)*A(J)
 3                  END DO
                    QINDU(NN,N)=QINDU(NN,N)+X*S
    100              CONTINUE
                 END DO
s             END DO
v          END DO
```

Fig. 4.39 Original DO loops in subroutine INDQA.

```
      DO N=1,N_ACTI

        A(1)=0.0D0
        A(1)=A(1)+ANOD1(1,NO(1),N)
        A(1)=A(1)+ANOD1(1,NO(2),N)
        A(1)=A(1)+ANOD1(1,NO(3),N)
        A(1)=A(1)/3.0D0                 ← A
        A(2)=0.0D0
        A(2)=A(2)+ANOD1(2,NO(1),N)
        A(2)=A(2)+ANOD1(2,NO(2),N)
        A(2)=A(2)+ANOD1(2,NO(3),N)
        A(2)=A(2)/3.0D0                 ← B
        A(3)=0.0D0
        A(3)=A(3)+ANOD1(3,NO(1),N)
        A(3)=A(3)+ANOD1(3,NO(2),N)
        A(3)=A(3)+ANOD1(3,NO(3),N)
        A(3)=A(3)/3.0D0                 ← C

        NN=K_NODE(NO(1))
        IF(NN.EQ.0) GO TO 100
        X=0.D0
        X=X+E(1,1)*A(1)                 ← A'
        X=X+E(2,1)*A(2)                 ← B'
        X=X+E(3,1)*A(3)                 ← C'
        QINDU(NN,N)=QINDU(NN,N)+X*S
100     CONTINUE
        NN=K_NODE(NO(2))
        IF(NN.EQ.0) GO TO 101
        X=0.D0
        X=X+E(1,2)*A(1)                 ← A'
        X=X+E(2,2)*A(2)                 ← B'
        X=X+E(3,2)*A(3)                 ← C'
        QINDU(NN,N)=QINDU(NN,N)+X*S
101     CONTINUE
        NN=K_NODE(NO(3))
        IF(NN.EQ.0) GO TO 102
        X=0.D0
        X=X+E(1,3)*A(1)                 ← A'
        X=X+E(2,3)*A(2)                 ← B'
        X=X+E(3,3)*A(3)                 ← C'
        QINDU(NN,N)=QINDU(NN,N)+X*S
102     CONTINUE

      END DO
```

Fig. 4.40 Modification of DO loop in subroutine INDQA.

```
A : A(1)=( ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,NO(3),N) )/3.0d0
B : A(2)=( ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,NO(3),N) )/3.0d0
C : A(3)=( ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,NO(3),N) )/3.0d0
```

Fig. 4.41 Modification of statements about A(1), A(2) and A(3).

```
      NN=K_NODE(NO(1))
      IF(NN.EQ.0) then
        NN=K_NODE(NO(2))
        IF(NN.EQ.0) then
          NN=K_NODE(NO(3))
          IF(NN.EQ.0) then
          else
          endif
        else
          NN=K_NODE(NO(3))
          IF(NN.EQ.0) then
          else
          endif
        endif
      else
        NN=K_NODE(NO(2))
        IF(NN.EQ.0) then
          NN=K_NODE(NO(3))
          IF(NN.EQ.0) then
          else
          endif
        else
          NN=K_NODE(NO(3))
          IF(NN.EQ.0) then
          else
          endif
        endif
      endif
```

Fig. 4.42 Modification into IF statement from GO TO statement in subroutine INDQA.

```
v           DO N=1,N_ACTI
v         NN=K_NODE(NO(1))
v         IF(NN.EQ.0) then
v           NN=K_NODE(NO(2))
v           IF(NN.EQ.0) then
v             NN=K_NODE(NO(3))
v             IF(NN.EQ.0) then
v             else
v               X=0.D0
v               X=X+E(1,3)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
        &NO(3),N))/3.0D0
v               X=X+E(2,3)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
        &NO(3),N))/3.0D0
v               X=X+E(3,3)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
        &NO(3),N))/3.0D0
v               QINDU(NN,N)=QINDU(NN,N)+X*S
v             endif
v           else
v             X=0.D0
v             X=X+E(1,2)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
        &NO(3),N))/3.0D0
v             X=X+E(2,2)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
        &NO(3),N))/3.0D0
v             X=X+E(3,2)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
        &NO(3),N))/3.0D0
v             QINDU(NN,N)=QINDU(NN,N)+X*S
v             NN=K_NODE(NO(3))
v             IF(NN.EQ.0) then
v             else
v               X=0.D0
v               X=X+E(1,3)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
        &NO(3),N))/3.0d0
v               X=X+E(2,3)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
        &NO(3),N))/3.0D0
v               X=X+E(3,3)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
        &NO(3),N))/3.0D0
v               QINDU(NN,N)=QINDU(NN,N)+X*S
v             endif
v           endif
v         else
v           X=0.D0
v           X=X+E(1,1)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
        &NO(3),N))/3.0d0
v           X=X+E(2,1)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
        &NO(3),N))/3.0D0
v           X=X+E(3,1)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
        &NO(3),N))/3.0D0
v           QINDU(NN,N)=QINDU(NN,N)+X*S
v           NN=K_NODE(NO(2))
```

Fig. 4.43 Vectorized DO loop in subroutine INDQA(1/2).

```
v          IF(NN.EQ.0) then
v            NN=K_NODE(NO(3))
v            IF(NN.EQ.0) then
v            else
v              X=0.D0
v              X=X+E(1,3)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
     &NO(3),N))/3.0d0
v              X=X+E(2,3)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
     &NO(3),N))/3.0D0
v              X=X+E(3,3)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
     &NO(3),N))/3.0D0
v              QINDU(NN,N)=QINDU(NN,N)+X*S
v            endif
v          else
v            X=0.D0
v            X=X+E(1,2)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
     &NO(3),N))/3.0d0
v            X=X+E(2,2)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
     &NO(3),N))/3.0D0
v            X=X+E(3,2)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
     &NO(3),N))/3.0D0
v            QINDU(NN,N)=QINDU(NN,N)+X*S
v            NN=K_NODE(NO(3))
v            IF(NN.EQ.0) then
v            else
v              X=0.D0
v              X=X+E(1,3)*(ANOD1(1,NO(1),N)+ANOD1(1,NO(2),N)+ANOD1(1,
     &NO(3),N))/3.0d0
v              X=X+E(2,3)*(ANOD1(2,NO(1),N)+ANOD1(2,NO(2),N)+ANOD1(2,
     &NO(3),N))/3.0D0
v              X=X+E(3,3)*(ANOD1(3,NO(1),N)+ANOD1(3,NO(2),N)+ANOD1(3,
     &NO(3),N))/3.0D0
v              QINDU(NN,N)=QINDU(NN,N)+X*S
v            endif
v          endif
v        endif
v          END DO
```

Fig. 4.43 Vectorized DO loop in subroutine INDQA(2/2).

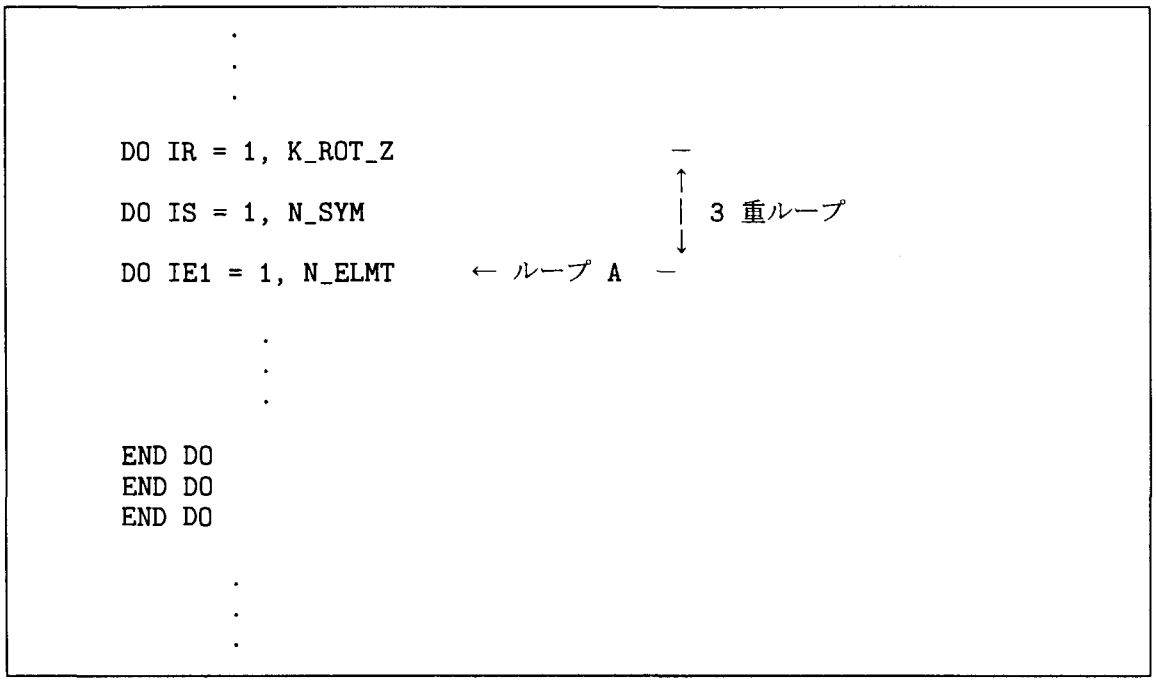

Fig. 4.44 Structure of DO loops in subroutine INDUCT.

```
      DO IR = 1, K_ROT_Z

      DO IS = 1, N_SYM

      DO IE1 = 1, N_ELMT      ← ループ A

                .
                .
                .

         DO IE2 = IE1+1 , N_ELMT
            DO IN1 = 1, 3
               M1 = K_NODE( NODE_ELMT(IN1,IE1) )
               IF(M1.NE.0) THEN
                  DO IN2 = 1, 3
                     M2 = K_NODE( NODE_ELMT(IN2,IE2) )
                     IF(M2.NE.0) THEN
                        IF(M1.GE.M2) THEN
                           IJ=((N_MODE*2-M2)*(M2-1))/2+M1
                        ELSE
                           IJ=((N_MODE*2-M1)*(M1-1))/2+M2
                        END IF
                        IF(M1.EQ.M2) THEN
                           FAC=2.0D0
                        ELSE
                           FAC=1.0D0
                        ENDIF
CCCCCCCCCCCCCC   インダクタンス行列 : SINDU
                        SINDU(IJ)=SINDU(IJ)+XXX(IN1,IN2,IE2)*FAC
                     END IF
                  END DO
               END IF
            END DO
         END DO
      END DO
      END DO
      END DO

                .
                .
                .
```

Fig. 4.45 Computation of inductance matrix in subroutine INDUCT.

```
*include INC_NPE
*include INC_MPI
*include INC_BSIZE

      DO IR = 1, K_ROT_Z

      DO IS = 1, N_SYM

c     DO IE1 = 1, N_ELMT
      DO IE1 = 1+my_pe, N_ELMT, npe  ← ループA

                .
                .
                .
         DO IE2 = IE1+1 , N_ELMT
            DO IN1 = 1, 3
               M1 = K_NODE( NODE_ELMT(IN1,IE1) )
               IF(M1.NE.0) THEN
                  DO IN2 = 1, 3
                     M2 = K_NODE( NODE_ELMT(IN2,IE2) )
                     IF(M2.NE.0) THEN
                        IF(M1.GE.M2) THEN
                           IJ=((N_MODE*2-M2)*(M2-1))/2+M1
                        ELSE
                           IJ=((N_MODE*2-M1)*(M1-1))/2+M2
                        END IF
                        IF(M1.EQ.M2) THEN
                           FAC=2.0D0
                        ELSE
                           FAC=1.0D0
                        ENDIF
CCCCCCCCCCCCCC   インダクタンス行列 : SINDU
                        SINDU(IJ)=SINDU(IJ)+XXX(IN1,IN2,IE2)*FAC
                     END IF
                  END DO
               END IF
            END DO
         END DO
      END DO
      END DO
      END DO
```

Fig. 4.46 Procedure decomposition and summation in subroutine INDUCT(1/2).

```
CCCCCCCCCCCCCC    B 総和の繰り返し数
      num_mat = nmtrx/nmtrx2
CCCCCCCCCCCCCC    C 余った部分
      num_mat_mod = mod(nmtrx,nmtrx2)

      do num_sum =  0,num_mat-1
CCCCCCCCCCCCCC    D 総和の繰り返し
        call mpi_allreduce(sindu(num_sum*nmtrx2+1), sindu2(1), nmtrx2, M
     &PI_DOUBLE_PRECISION, MPI_SUM, mpi_comm_world, ierr)
        do icl=1,nmtrx2
CCCCCCCCCCCCCC    E 元の配列へコピー
          sindu(num_sum*nmtrx2+icl)=sindu2(icl)
        end do
      end do

CCCCCCCCCCCCCC    F 余った部分の総和
      call mpi_allreduce(sindu(num_mat*nmtrx2+1), sindu2(1), num_mat_mod
     &, MPI_DOUBLE_PRECISION, MPI_SUM, mpi_comm_world, ierr)

      do icl=1,num_mat_mod
CCCCCCCCCCCCCC    G 元の配列へコピー
        sindu(num_mat*nmtrx2+icl)=sindu2(icl)
      end do

           .
           .
           .
```

Fig. 4.46 Procedure decomposition and summation in subroutine INDUCT(2/2).

```
      parameter(nmtrx2=130000)
      double precision sindu2(nmtrx2)
```

Fig. 4.47 Include file INC_BSIZE.

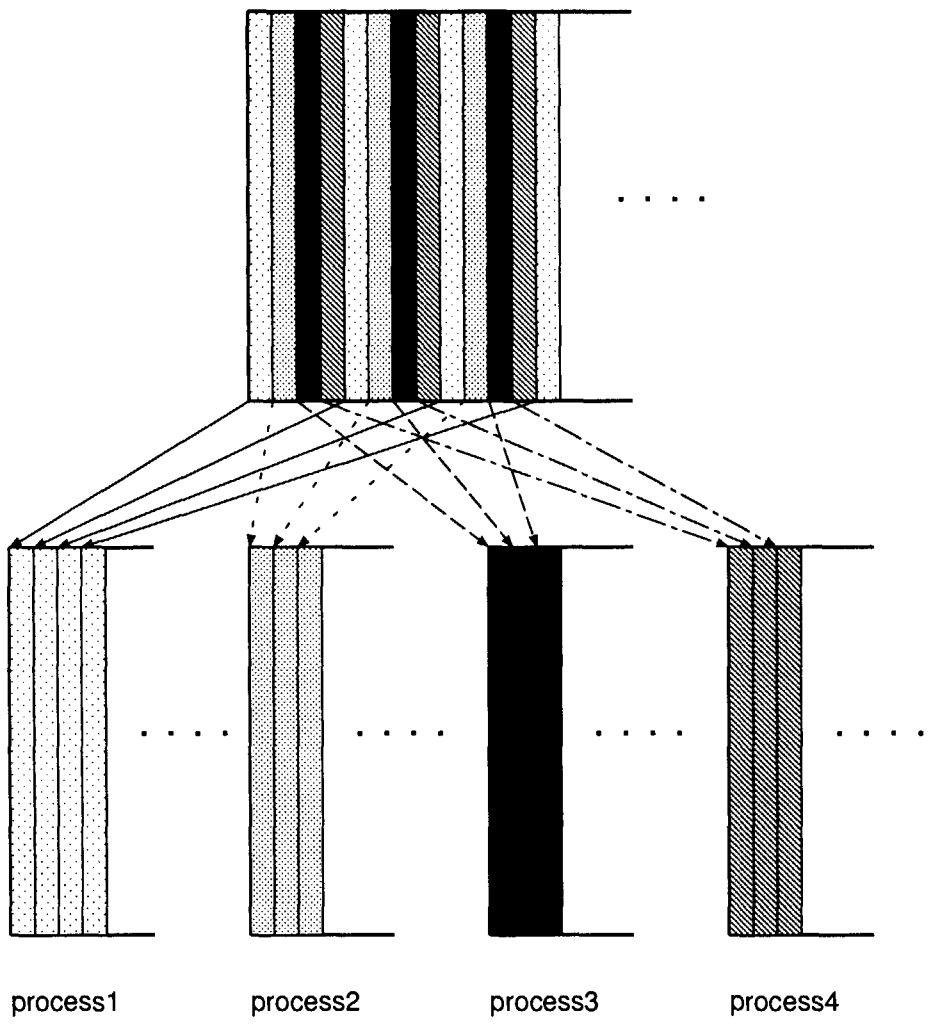

Fig. 4.48 Example of cyclic partition of matrix by using 4 processors.

```
      SUBROUTINE SSPGV( ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK,
     +                  INFO )
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDZ, N
      DOUBLE PRECISION                AP( * ), BP( * ), W( * ), WORK( * )
     +,                   Z( LDZ, * )
      LOGICAL            UPPER, WANTZ
      CHARACTER          TRANS
      INTEGER            J, NEIG
      LOGICAL            LSAME
      EXTERNAL           LSAME
      EXTERNAL           SPPTRF, SSPEV, SSPGST, DTPMV, DTPSV, XERBLA

                .
                .
                .

      CALL SSPGST( ITYPE, UPLO, N, AP, BP, INFO )
CCCCCCCCCCCCCC   サブルーチン SSPEV
      CALL SSPEV( JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, INFO )
      IF( WANTZ ) THEN
         NEIG = N

                .
                .
                .
```

Fig. 4.49 Original call statement for subroutine SSPEV in subroutine SSPGV

```
      SUBROUTINE SSPGV( ITYPE, JOBZ, UPLO, N, AP, BP, W, Z, LDZ, WORK,
     +                  INFO, lmtrx, ldljyd, ljyd, PW )
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      CHARACTER          JOBZ, UPLO
      INTEGER            INFO, ITYPE, LDZ, N
      DOUBLE PRECISION              AP( * ), BP( * ), W( * ), WORK( * )
     +,                  Z( LDZ, * ), PW(*)
      LOGICAL            UPPER, WANTZ
      CHARACTER          TRANS
      INTEGER            J, NEIG
      LOGICAL            LSAME
      EXTERNAL           LSAME
      EXTERNAL           SPPTRF,SSPEV_para,SSPGST,DTPMV,DTPSV,XERBLA

*include INC_NPE
*include INC_MPI

                  .
                  .
                  .

      CALL SSPGST( ITYPE, UPLO, N, AP, BP, INFO )

      if(ldljyd.ne.ljyd) ldljyd_para=ljyd

CCCCCCCCCCCCCC   サブルーチン sspev_para
      call sspev_para(ap,lmtrx,n,Z,LDLJYD_para,W,LJYD,
     &PW(1),PW(n+1),PW(2*n+1),PW(3*n+1),PW(4*n+1),PW(5*n+1),PW(6*n+1),
     &PW(7*n+1),PW(8*n+1),PW(9*n+1),PW(10*n+1),PW(11*n+1),PW(12*n+1))
```

Fig. 4.50 Modification of call statement for subroutine sspev_para in subroutine SSPGV(1/2).

```
      if(ldljyd.ne.ljyd) then
      jj=N
      ii=N
      do 35 kk=ljyd*ljyd,ldljyd,-1
        i=mod(kk,ldljyd)
        if(i.eq.0) i=ldljyd
        j=kk/ldljyd
        if(mod(kk,ldljyd).ne.0) j=j+1
        if(ii.ge.1) then
          z(ii,jj)=z(i,j)
          ii=ii-1
        else
          jj=jj-1
          ii=N
          z(ii,jj)=z(i,j)
          ii=ii-1
        endif
   35 continue
      do 36 kk=1,ljyd
        z(ldljyd,kk)=0.0d0
   36 continue
      endif

      IF( WANTZ ) THEN
         NEIG = N

                .
                .
                .
```

Fig. 4.50 Modification of call statement for subroutine sspev_para in subroutine SSPGV(2/2).

```
      SUBROUTINE INPST1A(
     +N_NODE, N_MODE, N_ELMT,   N_ELMTK, N_ELMT2, NJYD,  LJYD, LDNJYD,
     +LDLJYD, NMTRX,  LMTRX,    N_MATDA, N_MATNO, N_MAT, NMPC, MXTEN,
     +N_LINE, N_MATL, N_MATLDA, N_MATLNO,N_COIL2, N_PASS, N_CURAP,
     +N_CURNP, M_CURNP,
                .
                .
                .
      N9=NC5+3*N_PASS          +1
c.org N10=N9+NMTRX         +1
      N10=N9+NJYD*NJYD         +1
                .
                .
                .
```

Fig. 4.51 Modification of subroutine INPST1A.

```
INPST1A の呼び出し部分
                    .
                    .
                    .
          CALL INPST1A( N_NODE, N_MODE, N_ELMT, N_ELMTK, N_ELMT2,
     +NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,        N_MATDA, N_MATNO,
     + N_MAT, NMPC, MXTEN,         N_LINE, N_MATL, N_MATLDA, N_MATLNO,
     +      N_COIL2, N_PASS, N_CURAP, N_CURNP, M_CURNP,         M1,M2,M3,M
     +4,M5,M6,MA1,MA2,MA3,MA4,M7,M8,M9,MC1,MC2,MZ1,         N1,N2,N3,N4,N
     +5,N6,NB,NA1,NA2,N7,N8,NC1,NC2,NC3,NC4,NC5,         N9,N10,N11,N12,N
c.org      +14,NZ1,NC6D,NC7D,NCZ1,         N11B,NZ2,N15,N16,N17,NZ3 )
c.vec-s
     +14,NZ1,NC6D,NC7D,NCZ1,         N11B,NZ2,N15,N16,N17,NZ3,
     +M_VW1,M_VW11,N_EIGEN )  ← 新たなアドレスは N_EIGEN
c.vec-e
                    .
                    .
                    .
- - - - - - - - - - - - - - - - - - -
INPST1A 内の変更部分
       SUBROUTINE INPST1A(
     +N_NODE, N_MODE, N_ELMT,   N_ELMTK, N_ELMT2, NJYD,  LJYD, LDNJYD,
     +LDLJYD, NMTRX,  LMTRX,    N_MATDA, N_MATNO, N_MAT, NMPC, MXTEN,
     +N_LINE, N_MATL, N_MATLDA, N_MATLNO,N_COIL2, N_PASS, N_CURAP,
     +N_CURNP, M_CURNP,
     +M1,M2,M3,M4,M5,M6,MA1,MA2,MA3,MA4,M7,M8,M9,MC1,MC2,MZ1,
     +N1,N2,N3,N4,N5,N6,NB,NA1,NA2,N7,N8,NC1,NC2,NC3,NC4,NC5,
     +N9,N10,N11,N12,N14,NZ1,NC6D,NC7D,NCZ1,
c.org      +N11B,NZ2,N15,N16,N17,NZ3)
c.vec-s
     +N11B,NZ2,N15,N16,N17,NZ3,
     +M_VW1,M_VW11,N_EIGEN )  ← 新たなアドレスは N_EIGEN
c.vec-e
                    .
                    .
                    .
       N10=N9+NJYD*NJYD          +1
c.para-s
       N_EIGEN=N10+NMTRX    +1
       N11=N_EIGEN+13*NJYD  +1
       N12=N11+N_ELMT       +1
c.para-e
                    .
                    .
                    .
```

Fig. 4.52 Modification of call statement for subroutine INPST1A and statements in subroutine INPST1A.

```
EIGEN の呼び出し部分
                .
                .
                .
         CALL EIGEN(LMTRX,LJYD,LDLJYD,A(N9),A(N10),A(N15),A(N16),
     + A(N17), A(N_EIGEN) )  ← 作業配列 A(N_EIGEN)
                .
                .
                .
- - - - - - - - - - - - - - - - - -
EIGEN 内の変更
c.org SUBROUTINE EIGEN(LMTRX,LJYD,LDLJYD,SINDU,RRESI,EIGEX,VECTX,WORK)
c.para-s
      SUBROUTINE EIGEN(LMTRX,LJYD,LDLJYD,SINDU,RRESI,EIGEX,VECTX,WORK,P_
     &WORK)
c.para-e
                .
                .
c.para-s
     +,VECTX(LJYD),WORK(3*LDLJYD),P_WORK(13*LJYD)  ← 作業配列 P_WORK
c.para-e
                .
                .
      CALL SSPGV( 1, 'V', 'L', LJYD, SINDU, RRESI, VECTX, EIGEX,     LDL
     +JYD, WORK, INFO, lmtrx, ldljyd, ljyd, P_WORK) ← P_WORK を渡す
                .
                .
```

Fig. 4.53 Modification of call statement for subroutine EIGEN and statements in subroutine EIGEN.

```
 1:       subroutine sspev_para(c,lmtrx,N,Z,NNN1,W,NNN2,
 2:      &jdg,ie,ip,t0,t1,u,pq,w1,r0,r1,r2,f,ipv)
 3:       implicit real*8 (a-h,o-z)
 4:
 5:*include  INC_NPE
 6:*include  INC_MPI
 7:       real*8 c(N,N),Z(NNN1,NNN2),W(NNN2)
 8:
 9:       real*8 jdg(N),ie(N),ip(N),t0(N),t1(N)
10:       real*8 u(N),pq(N),w1(N),r0(N),r1(N),r2(N),f(N),ipv(N)
11:
12:       do iaaa=1,NNN2
13:       do ibbb=1,NNN1
14:          Z(ibbb,iaaa)=0.0d0
15:       end do
16:       end do
17:
18:       jj=N
19:       ii=N
20:       do 35 kk=lmtrx,N+1,-1
21:         i=mod(kk,N)
22:         if(i.eq.0) i=N
23:         j=kk/N
24:         if(mod(kk,N).ne.0) j=j+1
25:         if(ii.ge.jj) then
26:           c(ii,jj)=c(i,j)
27:           ii=ii-1
28:         else
29:           jj=jj-1
30:           ii=N
31:           c(ii,jj)=c(i,j)
32:           ii=ii-1
33:         endif
34:    35 continue
35:
36:       do 50 i=1,N
37:       do 51 j=1,N
38:          c(i,j)=c(j,i)
39:    51 continue
40:    50 continue
41:
42:       call mpi_barrier(mpi_comm_world,ierr)
43:
44:       ic1=0
45:       do k=1+my_pe, N, npe
46:         ic1=ic1+1
47:         do kk=1,N
48:           c(kk,ic1)=c(kk,k)
49:         end do
50:       end do
51:
```

Fig. 4.54 Parallelized subroutine sspev_para(1/2).

```
 52:       call mpi_barrier(mpi_comm_world,ierr)
 53:
 54:       call PJ_HOUSEH( my_pe, 0, npe, mpi_comm_world, c, N, N, ic1,
 55:      &                t0, t1, ierr, u, pq, w1 )
 56:
 57:       call PJ_EIGTRI( my_pe, 0, npe, mpi_comm_world, t0, t1, N, NNN1,
 58:      &                W,N, 1, N, Z, ic1, ie, ip, 0, jdg, ierr,
 59:      &                r0, r1, r2, f, ipv, w1 )
 60:
 61:       call PJ_TEVCNV( my_pe, 0, npe, mpi_comm_world, c, t0, t1, N,
 62:      &                NNN1,ic1, W, N, 1, N, Z, ic1, ie, ip, jdg,
 63:      &                ierr, w1 )
 64:
 65:       call mpi_barrier(mpi_comm_world,ierr)
 66:
 67:       ir_p=1
 68:       do is_p=1,NNN2/npe
 69:       call mpi_allgather(z(1,is_p),N,mpi_double_precision,c(1,ir_p),
 70:      &N,mpi_double_precision,mpi_comm_world,ierr)
 71:       ir_p=ir_p+npe
 72:       end do
 73:
 74:       ir_p_hold=ir_p
 75:       NN=N
 76:       if(mod(NN,npe).ne.0) then
 77:
 78:       do ic2=1,mod(NN,npe)
 79:       if(ic2-1.eq.my_pe) then
 80:         do ic3=1,NN
 81:         c(ic3,ir_p)=z(ic3,is_p)
 82:         end do
 83:       endif
 84:       ir_p=ir_p+1
 85:       end do
 86:
 87:       ic2=0
 88:       do ic4=ir_p_hold,ir_p-1
 89:       ic2=ic2+1
 90:       call mpi_bcast(c(1,ic4),NN,mpi_double_precision,ic2-1,
 91:      &mpi_comm_world,ierr)
 92:       end do
 93:
 94:       endif
 95:
 96:       do iaaa=1,NNN2
 97:       do ibbb=1,NNN1
 98:         z(ibbb,iaaa)=c(ibbb,iaaa)
 99:       end do
100:       end do
101:
102:       return
103:       end
```

Fig. 4.54 Parallelized subroutine sspev_para(2/2).

```
              .
              .
              .
      CALL SSPEV( JOBZ, UPLO, N, AP, W, Z, LDZ, WORK, INFO )
      IF( WANTZ ) THEN
         NEIG = N
         IF( INFO.GT.0 )      NEIG = INFO - 1
         IF( ITYPE.EQ.1 .OR. ITYPE.EQ.2 ) THEN
            IF( UPPER ) THEN
               TRANS = 'N'
            ELSE
               TRANS = 'T'
            END IF
CCCCCCCCCC サブルーチン DTPSV
            DO 10 J = 1, NEIG
             CALL DTPSV( UPLO, TRANS, 'NON-UNIT', N, BP, Z( 1, J ), 1)
   10       CONTINUE
              .
              .
              .
```

Fig. 4.55 Original part of call statement for subroutine DTPSV.

```
      SUBROUTINE DTPSV ( UPLO, TRANS, DIAG, N, AP, X, INCX )

      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
              .
              .
              .
            KK = ( N*( N + 1 ) )/2
            IF( INCX.EQ.1 )THEN
               DO 140, J = N, 1, -1
                  TEMP = X( J )
                  K = KK
                  DO 130, I = N, J + 1, -1
                     TEMP = TEMP - AP( K )*X( I )
                     K    = K    - 1
  130             CONTINUE
                  IF( NOUNIT )  TEMP = TEMP/AP( KK - N + J )
                  X( J ) = TEMP
                  KK     = KK   - ( N - J + 1 )
  140          CONTINUE

              .
              .
              .
```

Fig. 4.56 Original subroutine DTPSV.

```
          .
          .
          .
    IF( WANTZ ) THEN
       NEIG = N
       IF( INFO.GT.0 )      NEIG = INFO - 1

       IF( ITYPE.EQ.1 .OR. ITYPE.EQ.2 ) THEN
          IF( UPPER ) THEN
             TRANS = 'N'
          ELSE
             TRANS = 'T'
          END IF
          DO 10 J = 1+my_pe, NEIG, npe     ← サイクリック分割
             CALL DTPSV( UPLO, TRANS, 'NON-UNIT', N, BP, Z( 1, J ),

     +                   1 )
10        CONTINUE
          call z_trans(Z,AP,NEIG)          ← サブルーチン z_trans

          .
          .
          .
```

Fig. 4.57 Modification of DO statement and addition call statement for subroutine `z_trans`.

```
      subroutine z_trans(z,c,N)
      implicit real*8 (a-h,o-z)
      real*8 z(N+1,N),c(N,N)

*include INC_MPI
*include INC_NPE

      call mpi_barrier(mpi_comm_world,ierr)

      ir_p=1
      N_TMP=N-mod(N,npe)              ←  sspev_para 内の固有ベクトルの
      do is_p=1+my_pe,N_TMP,npe       ←  収集操作と異なる部分
      call mpi_allgather(z(1,is_p),N,mpi_double_precision,c(1,ir_p),N,mp
     &i_double_precision,mpi_comm_world,ierr)
      ir_p=ir_p+npe
      end do

      call mpi_barrier(mpi_comm_world,ierr)

      ir_p_hold=ir_p
      if(mod(N,npe).ne.0) then

        do ic1=1,mod(N,npe)
          if(ic1-1.eq.my_pe) then
            do ic2=1,N
            c(ic2,ir_p)=z(ic2,is_p)
            end do
          endif
          ir_p=ir_p+1
        end do

        ic1=0

        do ic2=ir_p_hold,ir_p-1
          ic1=ic1+1
          call mpi_bcast(c(1,ic2),N,mpi_double_precision,ic1-1,mpi_comm_
     &world,ierr)
        end do

      endif

      call mpi_barrier(mpi_comm_world,ierr)

      do iaaa=1,N
      do ibbb=1,N
        z(ibbb,iaaa)=c(ibbb,iaaa)
      end do
      end do

      call mpi_barrier(mpi_comm_world,ierr)

      return
      end
```

Fig. 4.58 Parallelized subroutine z_trans.

```
      parameter(iw_pe=0)
*include mpif.h
      integer*4 status(MPI_STATUS_SIZE)
      common/MY_RANK/my_pe
```

Fig. 4.59 Include file INC_MPI.

```
                .
                .
                .
      if(my_pe.eq.iw_pe) then
         READ(16)  (VECTX(I),I=1,LJYD)
      endif
      call mpi_barrier(mpi_comm_world,ierr)
      call mpi_bcast(vectx(1),LJYD,mpi_double_precision,iw_pe,mpi_comm_w
     &orld,ierr)
                .
                .
                .
```

Fig. 4.60 Example of data transfer after READ statement.

```
      SUBROUTINE MAIN1(MO,NO,IA,A)
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION A(NO),IA(MO)
      COMMON / FILE / LU01,LU08,LU06,LU10,LU11,LU20,LU23,LU24,LU25
                .
                .
                .
      IF( LF_OPTION(2) .LE. 1 ) THEN                    ← ステップ 1 開始
         CALL INPST1A( N_NODE, N_MODE, N_ELMT, N_ELMTK, N_ELMT2,
     +NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,        N_MATDA, N_MATNO,
     + N_MAT, NMPC, MXTEN,        N_LINE, N_MATL, N_MATLDA, N_MATLNO,
                .
                .
                .
      if(my_pe.eq.iw_pe) then                          ← ステップ 1 最終 I/O

      CALL OUTRST1( N_NODE, N_MODE, N_ELMT, N_ELMTK, N_ELMT2,     NJYD,
     +LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,     N_MATDA, N_MATNO, N_MAT, N
     +MPC, MXTEN,     N_LINE, N_MATL, N_MATLDA, N_MATLNO,     N_COIL2, N
     +_PASS, N_CURAP, N_CURNP, M_CURNP,     IA(M1), IA(M2), IA(M3), IA(M
     +4), IA(M5), IA(M6),     IA(MA1), IA(MA2), IA(MA3), IA(MA4),     IA
     +(M7), IA(M8), IA(M9), IA(MC1), IA(MC2),     A(N1),A(N2),A(N3),A(N4
     +),A(N5),A(N6),A(NB),A(NA1),     A(NA2),     A(N7), A(N8), A(NC1),
     +A(NC2), A(NC3), A(NC4), A(NC5),     A(N15), A(N16) )

      endif

      IF( LF_OPTION(3) .LE. 1 ) THEN
         RETURN
      END IF

CCCC only 1PE execution
      if(my_pe.eq.iw_pe) then                          ← ステップ 2 開始

      IF( LF_OPTION(2) .LE. 2 ) THEN
         CALL INPST2A( N_NODE, N_MODE, N_ELMT,        NJYD, LJYD, LDNJYD
     +, LDLJYD, NMTRX, LMTRX,        N_MAT, NMPC, MXTEN,        N_LINE,
     +N_MATL,        N_COIL2, N_PASS, N_CURAP, N_CURNP, M_CURNP,
                .
                .
                .
     +IA(M15),        A(NC6), A(NC8), A(NC9), A(NC10), A(NC7), A(N22),
     +      A(NA6), A(N29), A(NC11) )
      END IF

      endif
```

Fig. 4.61 Modification of subroutine **main1** to control I/O process(1/2).

```
CCCC only 1PE execution
      if(my_pe.eq.iw_pe) then                    ← ステップ 2 最終 I/O

      CALL OUTRST2(     N_TIME,N_PHAS, N_PASS,     IA(MC3), IA(MC4), IA(
     +M15),     A(NC6), A(NC8), A(NC9), A(NC10), A(NC7), A(N22),     A(N
     +A6), A(N29), A(NC11),     N_ELMT, N_COIL1, N_ACTI, N_CURAA, N_CURN
     +A, M_CURNA )

      endif

      IF( LF_OPTION(3) .LE. 2 ) THEN
         RETURN
      END IF

CCCC only 1PE execution
      if(my_pe.eq.iw_pe) then                    ← ステップ 3 開始

      CALL INPST3A( N_NODE,N_MODE,N_ELMT,N_ELMTK,KSYM,     N_NODE3,N_ELM
     +T3,N_ELMT3K,     NJYD, LJYD, LDNJYD, LDLJYD, NMTRX, LMTRX,     N_M
     +ATDA,N_MATNO,N_MAT,     N_COIL2, N_PASS, N_CURAP, N_CURNP, M_CURNP
     +,     N_COIL1, N_ACTI, N_CURAA, N_CURNA, M_CURNA,     N_TIME, N_PH
     +AS,     N_GRID, N_COIL0, N_CONST,     NFTOT, NODCOIL,NEGF,N_TIM3,
                .
                .
                .
     +,A(N39),A(NC28),A(NC29),     IA(MC8),A(NC14),     IA(M5),A(N1),A(N
     +33),A(N12),     A(N46),A(NC27),A(ND1),     N_TIM3,IA(MD1),A(ND2),A
c.org     +(ND3),A(ND4),A(ND5),A(ND6) )
c.vec-s
     +(ND3),A(ND4),A(ND5),A(ND6),
     +IA(M_VW1),IA(M_VW11),IA(M_VW2),IA(M_VW22) )
c.vec-e
      CALL PASSTIME(12)

      endif

      RETURN

 900  CONTINUE
      STOP 900
 903  CONTINUE
      STOP 903
 904  CONTINUE
      STOP 904
      END
```

Fig. 4.61 Modification of subroutine `main1` to control I/O process(2/2).

```
      parameter(npe=16)
```

Fig. 4.62 Include file `INC_NPE`.

参 考 文 献

[1] 亀有昭久，相川裕史他；カットのあるシェル上の渦電流，JAERI-M 6468，1976年3月.

[2] 亀有昭久，二宮博正他；一様でない抵抗を持つトーラス上の渦電流（臨界プラズマ試験装置設計報告・XXXIV），JAERI-M 6953，1977年2月.

[3] 亀有昭久，鈴木康夫；有限要素回路法による渦電流解析（臨界プラズマ試験装置設計報告・XXXVIII），JAERI-M 7120，1977年6月.

[4] 奥野清，島本進他；薄板任意形状渦電流解析コード：EDDYARBT(付EDDYPLOT)，JAERI-memo 8973，1980年8月.

[5] 「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 情報システム管理室，1995年7月.

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

[7] 並列数値計算ライブラリ，日本原子力研究所計算科学技術推進センター，
http://guide.tokai.jaeri.go.jp/program/software/list/index.html（日本語），
http://guide.tokai.jaeri.go.jp/program/eng/software/list/index.html（英語）
よりダウンロード.

5. THANPACST2 コードのベクトル並列化

5.1 はじめに

本作業では，大型構造機器実証試験ループ炉内構造物実証試験部による受動的冷却システム試験解析コードTHANPACST2 [1] の並列化を行なった．今回の並列化は，実行時間の短縮が主目的であり，使用メモリ量の拡大は２次的なものとなっている．しかしながら，本コードの並列化には演算性能と２次目的である使用メモリ量の拡大の両方を考慮し，粒子数をプロセッサ台数分に分割して，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粒子分割の手法を適応した．実行対象計算機は，分散メモリ型ベクトル並列計算機VPP500 [2,3] である．

5.2 動的解析

VPP500 上の SAMPLER [4] を使用し，THANPACST2 コードの動的解析を行なった．入力データとして，非定常問題と定常問題の２種類を用いた．非定常問題入力時の解析結果を Fig. 5.1 に，定常問題入力時の解析結果を Fig. 5.2 に示す．Fig. 5.1 を見ると，約５０%がソルバー部に費やされていることが分かる．Fig. 5.2 も同様に，ソルバーが上位を占めている．今回のデータは体系が小さいものであったが，データ体系が大きくなれば，ソルバー部のコストがさらに高くなることが予測される．

5.3 作業方針

本コードにはベクトル並列計算機用のチューニングが施されていない．そこで，まずベクトル化作業を行ない，次いで並列化作業を行なう手順とした．ソルバー部（PCG0,ICCG0）については，ベクトル化のみを行なう．PCG0 及び ICCG0 ルーチンはベクトル化により高い演算性能を発揮し，全体に占める割合が小さくなると予測できるためである．また，並列処理を行なうとプロセッサ間のデータ転送が頻繁に発生し，並列効率が極端に劣化する可能性が高く，問題によっては非並列の方が早くなる場合もあり得るため，安定して性能を出せるベクトル化版を作成する．他処理部に関してはベクトル化及び並列化の対象となる．

5.4 ベクトル並列化作業

5.4.1 新規作成インクルードファイル

新規に２個のインクルードファイルを作成した．ファイル名は，penum と indx である．penum では，並列処理に必要なパラメータの定義とコモン変数の宣言を行なっている．インクルードファイル penum の内容を Fig. 5.3 に示す．図中の，「parameter(iipe=4)」により並列数を決定する．この場合は，４並列であることを指示している．「common /jscnt/」は，各

プロセッサが処理するＤＯループの区間を格納する変数の宣言である．indx では，ハイパープレーン用のリスト作成時に用いる配列が宣言されている．インクルードファイル indx の内容を Fig. 5.4 に示す．

5.4.2 ＭＡＩＮの変更

ＭＡＩＮは制御部である．また，並列処理の範囲はＭＡＩＮ中のタイムループ内である．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言と分割ローカル配列の宣言及び速度と圧力に関する配列のグローバル化を行なっている．さらに，グローバル配列とローカル配列を結合し，データ転送に備えている．

・処理フローの変更

オリジナル版では，境界条件をセットした後に放射に関する定数を計算していた．並列化版では，放射に関する定数計算部をタイムステップループの直前に移動した．放射定数計算部はコストが高く，並列化を施した部分である．一方，並列処理はタイムステップ内を対象におこなっており，タイムステップループ開始からタイムステップループ終了までが並列処理される範囲である．そこで，並列化を施した放射定数計算部を並列処理の範囲であるタイムステップループの直前に移動した．放射定数計算部はタイムステップループに入る前であればどこで計算しても良く，今回の変更によるコードへの影響は無い．オリジナル版の処理フローと並列化版の処理フローを Fig. 5.5 に示す．

・並列処理の開始と終了

「!xocl parallel region」により並列処理が開始され，「!xocl end parallel」により並列処理が終了する．並列処理の開始終了はメインで制御している．

・データ転送

「!xocl overlapfix」により，配列袖部分のデータを転送している．他プロセッサ上の値を参照する時には転送命令が必要となる．変更箇所について，配列定義部分を Fig. 5.6 に示す．次に，計算処理部の変更箇所について， Fig. 5.7 に示す．

5.4.3 ＯＵＴＰＵＴの変更

OUTPUT では，演算結果をファイルに書き出す処理を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言と拡散項の各成分用分割ローカル配列の宣言を行なっている．

・データ転送

並列処理により，演算結果は各プロセッサ上に分散しているため，それらを１つのプロセッサ上に集める必要がある．よって，書き出し対象となっている全ての物性値について転送を行なっ

た．データ転送処理追加部分を Fig. 5.8 に示す．

5.4.4 ＰＣＧ０の変更

PCG0 は方程式解法部である．このルーチンは，ベクトル処理により高い演算性能を発揮している．もし，このルーチンに対して並列化を施すとすれば，残差計算時に転送が発生する．この転送は，イタレーション毎に発生し並列効率を悪化させることになる．さらに，並列処理によりベクトル長が短くなりベクトル効果が劣化することになる．よって，PCG0 に関しては並列化は施していない．ただし，他ルーチンの並列処理との整合性を保つため，処理の最初と最後に１度だけデータ転送を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言と定数ベクトル用グローバル配列と解ベクトル初期値用グローバル配列の宣言を行なっている．データ転送を行なうには並列処理用宣言が必要となる．

・データ転送

解くべき方程式中の定数ベクトルと解ベクトルの初期値は，VFIELD ルーチン内で並列処理されているため，データ転送が必要となる．まず，PCG0 ルーチンの先頭でデータ転送を行ない，次いで解法処理を行なう．解法処理終了後，解ベクトルの初期値用配列のみグローバル空間へ転送しておく．これは，収束計算の過程で本来の解ベクトルの値が初期値用配列に代入されており，この値を次回の方程式解法時の初期値とするためである．このようにすることにより，収束回数を減らすことができる．

・収束判定部変更

収束判定を行なう処理において，残差を求める時に MAX 関数が使用されている．ベクトル処理時には，MAX 関数より if 文による処理の方が効率が良い．よって，MAX 関数使用部分を if 文による最大値求解処理に変更した．PCG0 ルーチンに対して行なった各々の追加変更箇所を Fig. 5.9 に示す．

5.4.5 ＰＲＯＰの変更

PROP では，密度，熱伝導率やプラントル数などの初期値を計算している．このルーチンでは，処理を完全に並列化しており，且つデータ転送も行なっていない．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言と引数に関する分割ローカル配列の宣言を行なっている．親ルーチンとの整合性を保つために分割ローカル配列の宣言が必要になる．

・並列処理部分

ＤＯ１００ループが並列処理の対象であり，並列化指示行を追加した．このループ内では，他プロセッサ上の値を参照あるいは定義することが無いため，並列処理を行なってもデータ転送は不要である．PROP ルーチンに対して行なった各々の追加部分を Fig. 5.10 に示す．

5.4.6 ＲＡＤＣＡＬの変更

RADCAL では，放射計算を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言と引数に関する分割ローカル配列の宣言を行なっている．親ルーチンとの整合性を保つために分割ローカル配列の宣言が必要になる．さらに，データ転送用配列の定義も行なっている．

・並列処理部分

ＤＯ１０ループはＲＡＤＣＯＮルーチンで定義された変数 mow により並列処理される．ＤＯ４０ループ及びＤＯ６０ループは変数 mow とＲＡＤＣＯＮルーチンでソート処理時に定義された配列 llc を用いて並列処理している．ＲＡＤＣＡＬルーチンからは行列解法部であるＲＡＤＣＧルーチンが呼び出される．ＲＡＤＣＧルーチンは並列化困難なため非並列処理となっている．よって，ＲＡＤＣＧルーチンに入る前に各々のプロセッサ上にある値を集約する必要がある．そのため，プロセッサ間データ転送処理を新規追加した．ＤＯ８０ループ及びＤＯ１００ループは変数 mow により並列処理される． RADCAL ルーチンに対して行なった追加部分及び変更部分を Fig. 5.11 に示す．

5.4.7 ＲＡＤＣＯＮの変更

RADCON では，放射計算に必要な定数の設定を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言を行なっている．

・並列処理部分

ＤＯ１０ループ内の配列 LINK の定義部では，各プロセッサが演算すべき範囲のメッシュインデックスのみを代入している．以下の条件文により，各プロセッサが担当すべきインデックスを判定している．

```
if(jr(m,n).ge.jstt .and. jr(m,n).le.jenn) then
```

これにより，他プロセッサ上の値を参照定義することが無くなる．また，各プロセッサが担当すべきインデックスの個数をカウントするために変数 mow を新規に追加した．以降の処理では，変数 mow により各プロセッサのループ長が決まることになる．ＤＯ７０ループでは，ＤＯ１０ループで定義された変数 mow により並列処理が行なわれる．さらに，ＤＯ８０ループ及び，ＤＯ９０ループも同様に変数 mow により並列処理が行なわれる．サブルーチンの最後には，インデックス値をソートする処理を追加してあり，ソートした値はＲＡＤＣＡＬルーチンで使用される． RADCON ルーチンに対して行なった追加部分及び変更部分を Fig. 5.12 に示す．

5.4.8 ＲＡＤＰＲＥの変更

RADPRE では，放射計算格納用変数の設定を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言を行なっている．また， alamd,visc,pr,ts,vav それぞれの配列に関して分割ローカル配列指定を行なっている．

・並列処理部分

ＲＡＤＰＲＥルーチンの前処理として配列 VAV の袖転送命令を追加した．配列 VAV はＲＡＤＰＲＥルーチン内で他プロセッサ上の値を参照するため，事前に転送しておく必要がある．ＤＯ２０ループ及びＤＯ３０ループはＲＡＤＣＯＮで定義された変数 mow により並列処理される． RADPRE ルーチンに対して行なった追加部分及び変更部分を Fig. 5.13 に示す．

5.4.9 ＴＣＡＬの変更

TCAL では，エネルギー方程式に関する計算を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言を行なっている．

・並列処理部分

ＴＣＡＬルーチンの前処理として温度配列Ｔの作業配列への代入処理を追加した．この追加処理は，まず最初にＴＣＡＬルーチンが実行された場合と２回目以降に実行された場合とで代入元の配列が変わるようになっている．１回目の実行時には，並列処理による影響を受けていないためオリジナル配列からの代入で良い．ところが２回目以降の実行時には，並列処理により値が各々のプロセッサ上に分散することになる．各々のプロセッサ上での温度配列に対応する分割ローカル配列名は TL であり，２回目以降はこの分割ローカル配列から代入することになる．１回目の実行か２回目以降の実行かの判定は，ｉｔｃｆｌｇにより行っている．ＤＯ１００ループ及びＤＯ２０００ループは並列指示行により均等に並列処理される． TCAL ルーチンに対して行なった各々の追加部分を Fig. 5.14 に示す．

5.4.10 ＴＳＣＡＬの変更

TSCAL では，放射熱伝導を考慮した温度計算を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイル penum を追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言を行なっている．

・並列処理部分

配列 TS から配列 TSO への代入処理部分の代入元を変更した．配列 TS は並列処理により値が各々のプロセッサ上に分散している．各々のプロセッサ上での配列 TS に対応する分割配ロー

カル列名はTSLであり，分割ローカル配列TSLが代入元となる．ＤＯ２０００ループは並列指示行により均等に並列処理される．変更箇所をFig. 5.15 に示す．TSCALルーチンに対して行なった追加部分及び変更部分をFig. 5.15 に示す．

5.4.11 ＶＦＩＥＬＤの変更

VFIELDでは，速度に関する計算を行なっている．

・インクルードファイルの追加

並列処理用インクルードファイルpenumを追加した．

・並列処理用宣言文の追加

使用プロセッサ数の宣言を行なっている．

・並列処理部分

ＶＦＩＥＬＤルーチンの並列処理用前処理として各速度成分と圧力に関する分割ローカル配列へ値をセットする処理を追加した．この処理により，各プロセッサ上の分割ローカル配列に値がセットされる．以降の処理では，分割ローカル配列を使用することになる．ＤＯ３０００ループ，ＤＯ４５００ループ，ＤＯ８０００ループ，ＤＯ９２９０ループ，ＤＯ９５００ループ，ＤＯ９５２０ループは並列指示行により均等に並列処理される．ＶＦＩＥＬＤルーチンの後処理として各速度成分と圧力に関する配列のデータ転送を行っている．これはファイルへの出力に対応したものである．VFIELDルーチンに対して行なった各々の追加部分をFig. 5.16 に示す．

5.4.12 ＩＣＣＧ０の変更

ICCG0は，方程式解法部である．このルーチンには並列化を施していない．第一の理由は，ベクトル化により十分な性能を発揮することができ，コード全体に占める割合を小さくできるためである．第二の理由は，差分法であるためイタレーション毎にデータ転送が発生してしまい，並列化効率が悪いためである．よって，このルーチンにはベクトル化のみを施すことにした．ベクトル化の手法として，リストアクセスハイパープレーン法を適応した．この手法を用いると，使用メモリ増加量を少なく抑え，ベクトル化できるためである．まず，非並列であるため，ICCG0ルーチンの前処理としてデータ転送部分を追加した．この転送は，ICCG0ルーチンが呼ばれた時に１回だけ実行される．リストアクセスハイパープレーン法の適用によりＤＯループの配列インデックスを変更した．対象となったループは，ＤＯ３０ループ，ＤＯ４０ループ，ＤＯ９０ループ，ＤＯ１００ループである．ICCG0ルーチンの後処理としてデータ転送部分を追加した．ICCG0ルーチン以降の処理は並列化されており，並列処理用分割ローカル配列に計算値を代入する必要があるためである．この転送は、ICCG0ルーチンが呼ばれた時に１回だけ実行される．ICCG0ルーチンに対して行なった追加部分及び変更部分をFig. 5.17 に示す．

5.4.13 indexの追加

indexでは、リストアクセスハイパープレーン用の配列アクセスリストを作成する．ICCG0におけるオリジナルコーディングの配列アクセス方法は以下のようになっている．

$$a(i,j,k) = a(i,j,k) - bx(i,j,k) * a(i-1,j,k)$$

$$-by(i,j,k)*a(i,j-1,k)$$
$$-bz(i,j,k)*a(i,j,k-1) \tag{5.1}$$

しかし，このアクセス順序ではリカレンスが発生してしまい，ベクトル演算が出来ない．そこで，リカレンスが発生しない面に着目し，その面内をベクトル演算する手法であるハイパープレーン法について考える．例えばi+j+k=5となる点を抜き出すと (3,1,1) (2,2,1) (1,3,1) (2,1,2) (1,2,2) (1,1,3) なる６点が相当する．これらの点に対する演算をするには，Fig. 5.18 の参照面１に含まれる点を参照することになる．ここで，参照面１については，i+j+k=4 なる面を計算した時点で既に求められており，これらの点の参照と i+j+k=5 なる面の定義間にリカレンスは発生しない．よって，一般的に i+j+k=const となる面内ではベクトル演算が可能となる．この面に対する演算を順次進めて行けば，ベクトル処理を実現できる．よって，以下のようなロジックによりベクトル化が可能となる．

１） i+j+k=const となる点へのリストを作成する．

２）リストに従い，下位の面から上位の面へと順次処理を進める．

リスト作成ルーチンはｉｎｄｅｘである．このルーチンでは，i+j+k=const となる点を抜き出し，リスト用配列へ格納する処理を行なっている．ｉｎｄｅｘルーチン終了後，ソルバー（ＩＣＣＧＯ）へ処理が移行する．このソルバー内では，ｉｎｄｅｘで作成されたリストに従い，ベクトル処理を実現している．このような手法をリストを利用したハイパープレーン法と呼ぶ．ｉｎｄｅｘルーチンを Fig. 5.19 に示す。

5.5 性能

オリジナルスカラ実行，オリジナルベクトル実行，チューニング後ベクトル実行，チューニング後並列実行（４並列実行）の４ケースについてコード全体の実行時間を計測した．非定常データ使用時の実行時間比較表を Table 5.1 に示し，定常データ使用時の実行時間比較表を Table 5.2 に示す．表中の ratio は倍率であり，オリジナルスカラ実行に対する比率を表している．この値を見ると，並列効率が５０％程度しか出ていないことが分かる．原因はデータ転送にかかるコストが大きいためである．ソルバー部が非並列であるためタイムステップ毎に配列全体の転送が発生しており，並列性能を劣化させている．とはいえ，並列化に伴うベクトル長の短縮を補う大規模データを入力した場合には，ベクトル効果が上がり４並列実行で１０倍程度（対オリジナルスカラ）の倍率が出せると予測できる．

5.6 まとめ

今回のベクトル並列化において，ソルバー部のみ非並列とした．ソルバー部を並列処理することは可能であるが，変更作業に多大の時間を要する．また，反復法におけるイタレーション中のデータ転送は避けられず，十分な並列効果を発揮する保証はない．しかし，ソルバー部非並列に伴うデータ転送よりは，イタレーション中のデータ転送の方がコストが小さい．さらに，ソルバー部を並列化すればメモリ量の関係で見送った連続アクセスハイパープレーン法を適応できる．これらの効果により現状より性能アップできると判断する．ただし，どのような手法により並列化を行なうかは今後の課題である．

Table. 5.1 Executive time comparative table for unstational data.

data-type	version	cpu time	elaps time	ratio
s37dat2(TFIN=29)	original scalar	6516sec	7118sec	1.00
	original vector	4017sec	4349sec	1.64
	tuning vector	2107sec	2132sec	3.34
	tuning parallel (4PE)	3997sec	1017sec	7.00

Table. 5.2 Executive time comparative table for stational data.

data-type	version	cpu time	elaps time	ratio
t27data(TFIN=700)	original scalar	4503sec	4905sec	1.00
	original vector	2128sec	2651sec	1.85
	tuning vector	1139sec	1210sec	4.05
	tuning parallel (4PE)	2313sec	590sec	8.31

```
------- T2 (TFIN=29) Vector ------- s37dat2

 Status                        : Serial
 Number of Processors          : 1

 Type                          : cpu
 Interval (msec)               : 10

 Synthesis Information
     Count|   Percent|    VL| Name
    189677|      47.2|   180|  radcg_
     64661|      16.1|   191|  tcal_
     35503|       8.8|   571|  vfield_
     30938|       7.7|    46|  radcal_
     21420|       5.3|     3|  convv_
     16994|       4.2|     3|  convu_
     15277|       3.8|   603|  pcg0_
      7699|       1.9|  1083|  tscal_
      7378|       1.8|     -|  prop_
      4829|       1.2|     -|  alamds_
      3818|       1.0|     4|  radpre_
      1476|       0.4|     -|  MAIN__
       824|       0.2|     -|  rhos_
       736|       0.2|     -|  cps_
       293|       0.1|     -|  bcset_
        75|       0.0|     -|  output_
        69|       0.0|     -|  input_
         8|       0.0|     -|  tcalin_
         3|       0.0|     -|  radcon_

    401678|          |   212|  TOTAL


   radcg : newton-rapson 法による解法部
   radcal: ソルバー制御部
   pcg0  : PCG 法による解法部
```

Fig. 5.1 Dynamic behavior of original version for unsteady data.

```
------- T2 (TFIN=700) Vector ------- t37data

 Status                        : Serial
 Number of Processors          : 1

 Type                          : cpu
 Interval (msec)               : 10

 Synthesis Information
     Count|   Percent|    VL| Name
     90181|      42.4|   180|  radcg_
     32211|      15.1|   552|  tcal_
     21576|      10.1|   585|  pcg0_
     17917|       8.4|   541|  vfield_
     17119|       8.0|    45|  radcal_
     11094|       5.2|     3|  convv_
      8524|       4.0|     3|  convu_
      3893|       1.8|  1083|  tscal_
      3649|       1.7|     -|  prop_
      2428|       1.1|     -|  alamds_
      1909|       0.9|     6|  radpre_
      1004|       0.5|    17|  MAIN__
       424|       0.2|     -|  rhos_
       412|       0.2|     -|  cps_
       291|       0.1|     -|  bcset_
        78|       0.0|     -|  output_
        65|       0.0|     -|  input_
         4|       0.0|   180|  radcon_

    212779|          |   314|  TOTAL


   radcg : newton-rapson 法による解法部
   radcal: ソルバー制御部
   pcg0  : PCG 法による解法部
```

Fig. 5.2 Dynamic behavior of original version for steady data.

```
      parameter(iipe=4)
      common /coflg/ ivfflg , itcflg , itsflg
      common /jscnt/ jstt , jenn , jst , jen
```

Fig. 5.3 Include file penum.

```
      common /indx/ lcnt(lijk),lplane_cnt(lijk),lplane_add(lijk),
     &              numiccg(lijk),no_p
```

Fig. 5.4 Include file indx.

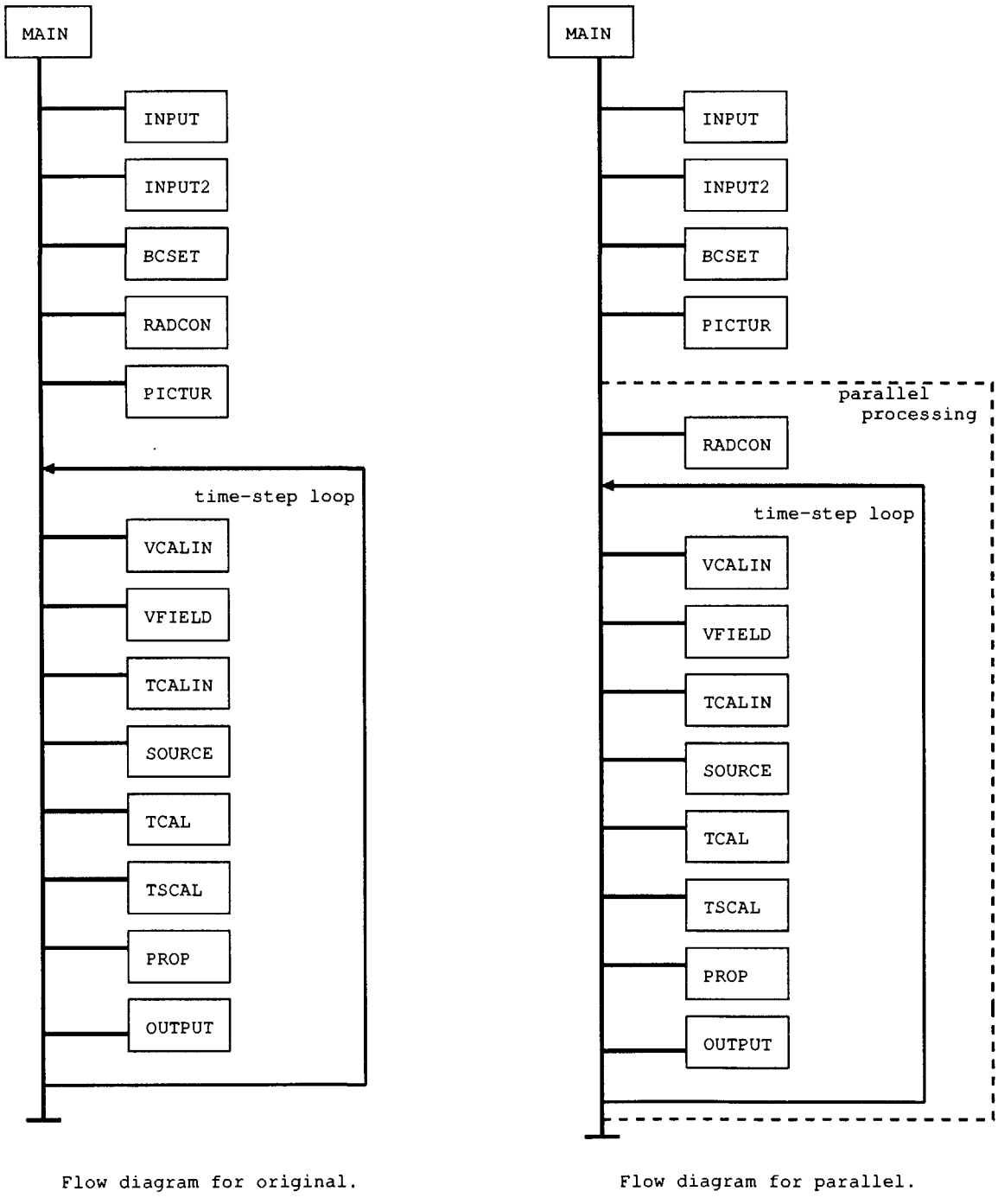

Fig. 5.5 Flow diagrams for original and parallel version.

```
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      INCLUDE (ARRAY3)
      INCLUDE (ARRAY1)
      INCLUDE (INP)
      INCLUDE (MASS)
      INCLUDE (OUT)
      INCLUDE (ARRAYT)
      INCLUDE (COM01)
      INCLUDE (COM00)
      INCLUDE (COM02)
      INCLUDE (COM05)
      INCLUDE (DELT)
      INCLUDE (MATPR0)
      include (penum)        <-------- インクルードファイル追加
      DIMENSION QSUM(LI,LJ,LK)
      real  tsl(li,lj,lk)                                    -------+
      real  pl(li,lj,lk)                                            |
      common/gldm/ug(li,lj,lk),vg(li,lj,lk),wg(li,lj,lk),           |
     &            pg(li,lj,lk)                                      |
C                                                                   |
!xocl processor pe(iipe)                                           |
!xocl index partition qn=(proc=pe,index=1:JMAX,part=band)          |
!xocl local  pl(:,/qn(overlap=(0,1)),:)                            | 並列処理用宣
!xocl local  tsl(:,/qn(overlap=(1,1)),:)                           | 言追加
!xocl local  alamd(:,/qn(overlap=(1,1)),:)                         |
!xocl local  visc(:,/qn(overlap=(1,1)),:)                          |
!xocl local  pr(:,/qn(overlap=(1,1)),:)                            |
!xocl local  rho(:,/qn(overlap=(0,1)),:)                           |
!xocl global ug,vg,wg,pg                                           |
!xocl global tsg,alamdg,viscg,prg,rhog                 -------+
```

Fig. 5.6 Modification of array definition for MAIN.

```
              .
              .
              .
C ----------------  TIME  LOOP  START  ---------------
c----- parallel process -----
!xocl parallel region          <------- 並列処理開始
c initial value set
      ivfflg = 0
      itsflg = 0
      jstt = 0
!xocl spread do /qn                    -------+
      do lm = 1,jmax                          |
        if(jstt.eq.0) jstt=lm                 |
      enddo                                   |
!xocl end spread                              |
      jenn = lm-1                             |
!xocl barrier                                 |
!xocl spread do                               | 並列処理用ループ変数定義部
      do ipp=1,iipe                           |
        if(idvproc().ne.1) then               |
          jst=jstt-1                          |
        else                                  |
          jst=jstt                            |
        endif                                 |
        if(idvproc().ne.iipe) then            |
          jen=jenn+1                          |
        else                                  |
          jen=jenn                            |
        endif                                 |
      enddo                                   |
!xocl end spread                       -------+
c set initial value for TS & ALAMD,VISC,PR,RHO
!xocl   spread do /qn                  ------------+
        do j=1,jmax                                |
          do k=1,kmax                              |
            do i=1,imax                            |
              tsl(I,J,K) = TS(I,J,K)               |
              alamd(i,j,k)=alamdr(i,j,k)           | 分割配列への初期値代入
              visc(i,j,k)=viscr(i,j,k)             |
              pr(i,j,k)=prr(i,j,k)                 |
              rho(i,j,k)=rhor(i,j,k)               |
            enddo                                  |
          enddo                                    |
        enddo                                      |
!xocl   end spread                                 |
!xocl overlapfix (tsl,alamd,visc,pr,rho) (xt)      |
!xocl movewait (xt)                    ------------+
              .
              .
              .
```

Fig. 5.7 Modification calculation for MAIN. (1/2)

```
              .
              .
              .
C                                  ********************
      IF(MOD(ITIME,ITCALL).EQ.0)  THEN
        IF(ITDPN .EQ. 1)           CALL TCALIN(TIME)
        IF(IQDPN .EQ. 1)           CALL SOURCE(TIME)
        CALL TCAL (ITIME,TIME,tsl,alamd,visc,pr)
        CALL TSCAL(ITIME,TIME,tsl,alamd,visc,pr)
!xocl   overlapfix (tsl) (xt)          ----------+
!xocl   movewait (xt)                  ----------+ 袖転送
        CALL PROP(alamd,visc,pr,rho)
!xocl   overlapfix (alamd,visc,pr,rho) (xt) ---+
!xocl   movewait (xt)                          ---+ 袖転送
      ENDIF
C                                  ********************
              .
              .
              .
C----------------- TIME LOOP END -------------------------
 7000 CONTINUE
c     CALL OUTPUT( ITIME, TIME, ITER1)
!xocl end parallel                 <------- 並列処理終了
      WRITE(NTWRT) ITIME,TIME,IPB
     *            ,((( U(I,J,K),V(I,J,K),W(I,J,K),P(I,J,K)
     *            ,T(I,J,K),TS(I,J,K),QRAD(I,J,K),QI(I,J,K)
     *            ,QSUM(I,J,K),K=1,KMAX),I=1,IMAX),J=1,JMAX)
     *            ,(RHO0(L),RHO1(L),L=1,LSYS)
     *            ,(XXR(I),I=1,MOMAX)
      WRITE( 6,6300) TIME,ITIME,ITER1
      WRITE( 6,6400)
              .
              .
              .
```

Fig. 5.7 Modification calculation for MAIN. (2/2)

```
C----- data transfer section -----
!xocl spread do /qt
      do j=1,jmax
        do k=1,kmax
          do i=1,imax
            tauil(i,j,k)=taui(i,j,k)
            taujl(i,j,k)=tauj(i,j,k)
            taukl(i,j,k)=tauk(i,j,k)
            qil(i,j,k)=qi(i,j,k)
          enddo
        enddo
      enddo
!xocl end spread
!xocl spread move
      do k=1,kmax
        do j=1,jmax
          do i=1,imax
            taui(i,j,k)=tauig(i,j,k)
            tauj(i,j,k)=taujg(i,j,k)
            tauk(i,j,k)=taukg(i,j,k)
            qi(i,j,k)=qig(i,j,k)
          enddo
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
c--- transfer for XXR ---
      idp=idvproc()
      do i=1,iipe
        mowlist(i)=0
      enddo
!xocl spread do
      do ipp=1,iipe
        mowlist(ipp)=mowlist(ipp)+mow
      enddo
!xocl end spread sum(mowlist)
!xocl spread do
      do ipp=1,iipe
        mowmx=mow
      enddo
!xocl end spread max(mowmx)
      do i=1,mowlist(idp)
        XXRT(i,idp) = XXR(i)
        mXXRL(i,idp) = mol2(i)
      enddo
!xocl barrier
```

Fig. 5.8 Addition of data transfer for OUTPUT. (1/6)

```
!xocl spread move
      do j=1,iipe
        do i=1,mowmx
          XXRTW(i,j)=xxrtg(i,j)
          mXXRTL(i,j)=mxxrlg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mowlist(j)
          xxrtmp(mxxrtl(i,j))=xxrtw(i,j)
        enddo
      enddo
c--- transfer for SA ---
      nmaxv=0
!xocl spread do
      do ipp=1,iipe
        do l=1,mow
          if(nmaxv.lt.nmax(l)) then
            nmaxv=nmax(l)
          endif
        enddo
      enddo
!xocl end spread max(nmaxv)
      do nlp=1,nmaxv
      if(nmax(nlp).ne.0) then
        do i=1,mow
          XXRT(i,idp) = SA(i,nlp)
        enddo
      else
          XXRT(i,idp) = 0.0
      endif
!xocl barrier
!xocl spread move
      do j=1,iipe
        do i=1,mowmx
          XXRTW(i,j)=xxrtg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mowlist(j)
          satmp(mxxrtl(i,j),nlp)=xxrtw(i,j)
        enddo
      enddo
      enddo
```

Fig. 5.8 Addition of data transfer for OUTPUT. (2/6)

```
c--- transfer for SB,SD ---
      do i=1,mow
        XXRT(i,idp) = SB(i)
      enddo
!xocl barrier
!xocl spread move
      do j=1,iipe
        do i=1,mowmx
          XXRTW(i,j)=xxrtg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mowlist(j)
          SBtmp(mxxrtl(i,j))=xxrtw(i,j)
        enddo
      enddo
      do i=1,mow
        XXRT(i,idp) = SD(i)
      enddo
!xocl barrier
!xocl spread move
      do j=1,iipe
        do i=1,mowmx
          XXRTW(i,j)=xxrtg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mowlist(j)
          SDtmp(mxxrtl(i,j))=xxrtw(i,j)
        enddo
      enddo
c--- transfer for LINK ---
!xocl spread do
      do ipp=1,iipe
        nmaxmx=nmax(mow)
      enddo
!xocl end spread max(nmaxmx)
      do m=1,4
      do n=0,nmaxmx
      do i=1,mow
        XXRT(i,idp) = LINK(i,n,m)
      enddo
!xocl barrier
```

Fig. 5.8 Addition of data transfer for OUTPUT. (3/6)

```
!xocl spread move
      do j=1,iipe
        do i=1,mowmx
          XXRTW(i,j)=xxrtg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mowlist(j)
          linktmp(mxxrtl(i,j),n,m)=xxrtw(i,j)
        enddo
      enddo
      enddo
      enddo
c mol2_list for link5
      do i=1,mow
        XXRT(i,idp) = mol2(i)
      enddo
!xocl barrier
!xocl spread move
      do j=1,iipe
        do i=1,mowmx
          XXRTW(i,j)=xxrtg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mowlist(j)
          mol2lst(i,j)=xxrtw(i,j)
        enddo
      enddo
c ----- link5
!xocl spread do
      do ipp=1,iipe
        do i=1,mow
          nmaxmx=nmax(i)
        enddo
      enddo
!xocl end spread max(nmaxmx)
```

Fig. 5.8 Addition of data transfer for OUTPUT. (4/6)

```
      ijc=0
      do k=1,mo
!xocl spread do
        do ipp=1,iipe
        ljc=nmaxlist(k)
        ljcmx=nmaxlist(k)
        if(ljc.ne.0) then
          ijc=ijc+1
          do i=1,ljc
            XXRT(i,idp) = mol2(LINK(ijc,i,5))
          enddo
        else
          do i=1,ljc
            XXRT(i,idp) = 0.0
          enddo
        endif
        enddo
!xocl end spread max(ljcmx)
!xocl spread move
      do j=1,iipe
        do i=1,ljcmx
          XXRTW(i,j)=xxrtg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
!xocl spread do
      do ipp=1,iipe
        do i=1,iipe
          ijcc(i)=0
          ljcc(i)=0
        enddo
      enddo
!xocl end spread
!xocl spread do
      do ipp=1,iipe
        ijcc(ipp)=ijcc(ipp)+ijc
        ljcc(ipp)=ljcc(ipp)+ljc
      enddo
!xocl end spread sum(ijcc),sum(ljcc)
      do j=1,iipe
        do i=1,ljcc(j)
          if(xxrtw(i,j) .ne. 0.0) then
            linktmp(mol2lst(ijcc(j),j),i,5)=xxrtw(i,j)
          endif
        enddo
      enddo
      enddo
```

Fig. 5.8 Addition of data transfer for OUTPUT. (5/6)

```
c--- transfer for NMAX ---
      do i=1,mo
        mXXRL(i,idp) = nmaxlist(i)
      enddo
!xocl barrier
!xocl spread move
      do j=1,iipe
        do i=1,mo
          mXXRTL(i,j)=mxxrlg(i,j)
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
      do j=1,iipe
        do i=1,mo
          nval=mxxrtl(i,j)
          if(nval.ne.0) then
            nmaxval(i)=nval
          endif
        enddo
      enddo
c--- transfer for TS ---
!xocl spread move
      do k=1,lk
        do j=1,lj
          do i=1,li
            ts(i,j,k)=tsg(i,j,k)
          enddo
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
c--- transfer for QS ---
!xocl spread move
      do k=1,kmax
        do j=1,jmax
          do i=1,imax
            do l=1,6
              qs(l,i,j,k)=qsg(l,i,j,k)
            enddo
          enddo
        enddo
      enddo
!xocl end spread (xt)
!xocl movewait (xt)
C----- data transfer section end -----
```

Fig. 5.8 Addition of data transfer for OUTPUT. (6/6)

```
C    SOLV LINEAR EQUATION BY PCG METHOD
C
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      INCLUDE (INP)
      INCLUDE (MASS)
      include (penum)              <------- インクルードファイル追加
C
!xocl processor pe(iipe)                          -------+
!xocl subprocessor pes(iipe)=pe(1:iipe)                  | 並列処理用宣言追加
!xocl global dsg,delpg                            -------+
               .
               .
!xocl spread move                                 -------+
      do i=1,li                                          |
        do j=1,lj                                        |
          do k=1,lk                                      |
            dsl(k,j,i)=dsg(k,j,i)                        | データ転送処理
            delpl(k,j,i)=delpg(k,j,i)                    |
          enddo                                          |
        enddo                                            |
      enddo                                              |
!xocl end spread (xt)                                    |
!xocl movewait (xt)                               -------+
               .
               .
C    ------  CONVERGENCE CHECK  ------
        DMAX=1.E-50                                      -------+
        DO 2800 NN=1,NUME                                       |
          NNN=NO(NN)                                            |
          DD=ABS(R(NNN))/DVM(NOI(NNN),NOJ(NNN),NOK(NNN))        | 収束判定部変更
          if(DD.gt.DMAX) then                                   |
            DMAX = DD                                           |
          endif                                                 |
 2800   CONTINUE                                         -------+
               .
               .
!xocl spread move                                 -------+
      do i=1,li                                          |
        do j=1,lj                                        |
          do k=1,lk                                      |
            delpg(k,j,i)=delpl(k,j,i)                    | データ転送処理
          enddo                                          |
        enddo                                            |
      enddo                                              |
!xocl end spread (xt)                                    |
!xocl movewait (xt)                               -------+
               .
               .
```

Fig. 5.9 Addition and modification of PCG0.

```
      SUBROUTINE  PROP(alamd,visc,pr,rho)
C
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      include (penum)              <------- インクルードファイル追加
      INCLUDE (ARRAY3)
      INCLUDE (INP)
      INCLUDE (MATPRO)
      INCLUDE (COM01)
      INCLUDE (COM00)
      PARAMETER ( G = 9.8 )
C
!xocl processor pe(iipe)                                    -------+
!xocl subprocessor pes(iipe)=pe(1:iipe)                             |
!xocl index partition qn=(proc=pes,index=1:JMAX,part=band)    | 並列処理用宣
!xocl local alamd(:,/qn(overlap=(10,10)),:)                  | 言追加
!xocl local visc(:,/qn(overlap=(10,10)),:)                   |
!xocl local pr(:,/qn(overlap=(10,10)),:)                     |
!xocl local rho(:,/qn(overlap=(0,1)),:)                     -------+
              .
              .
              .
!xocl spread do /qn              <------- 並列化指示行追加
      DO 100 J = 2, JM
        DO 100 K = 2, KM
          DO 100 I = 2, IM
              .
              .
  100 CONTINUE
!xocl end spread
              .
              .
              .
```

Fig. 5.10 Additional part for PROP.

```
      SUBROUTINE  RADCAL(tsl,alamd,visc,pr)
C
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      INCLUDE (COM00)
      INCLUDE (COM02)
      INCLUDE (RADDBG)
      INCLUDE (MATPRO)
      include (penum)              <------- インクルードファイル追加
      DIMENSION AA(MOMAX*MOMAX),BB(MOMAX),XR(MOMAX)
      dimension aat(32400,iipe),aal(32400,iipe),      -------+
     &          aatw(32400,iipe),aatl(32400,iipe),           |
     &          aatg(32400,iipe),aalg(32400,iipe)            |
      dimension bbt(momax,iipe),bbl(momax,iipe),             |
     &          bbtw(momax,iipe),bbtl(momax,iipe),           |
     &          bbtg(momax,iipe),bblg(momax,iipe)            |
      dimension moll(momax),dms(momax)                       |
      dimension tsl(li,lj,lk)                                |
      dimension lct(iipe)                                    | 並列処理用宣
      common/rdm/mol(momax),mol2(momax),mow,llc(iipe),llcmx  | 言追加
!xocl processor pe(iipe)                                     |
!xocl subprocessor pes(iipe)=pe(1:iipe)                      |
!xocl index partition qn=(proc=pes,index=1:JMAX,part=band)   |
!xocl index partition qnt=(proc=pes,index=1:iipe,part=band)  |
!xocl local  aat(:,/qnt),aal(:,/qnt)                         |
!xocl local  bbt(:,/qnt),bbl(:,/qnt)                         |
!xocl local  tsl(:,/qn(overlap=(1,1)),:)                     |
!xocl local  alamd(:,/qn(overlap=(1,1)),:)                   |
!xocl local  visc(:,/qn(overlap=(1,1)),:)                    |
!xocl local  pr(:,/qn(overlap=(1,1)),:)                      |
!xocl global aatg,aalg,bbtg,bblg,tsg                 -------+
             .
             .
             .
!xocl spread do
      do ipp=1,iipe
      DO 40 I = 1 , llc(ipp)          <------- 変数 llc による並列処理
        DM = 0.
        jjj=0
        DO 50 J = 1 , MOw             <------- 変数 mow による並列処理
          dmc=ABS(AA(MO*(mol2(J)-1)+mol(I)))
          IF (dmc.GT.DM) then
            DM = dmc
            jjj=j
          endif
   50   CONTINUE
        dms(mol(i))=dm
   40 CONTINUE
      enddo
!xocl end spread max(dms)
```

Fig. 5.11 Addition and modification of RADCAL. (1/3)

```
!xocl spread do
      do ipp=1,iipe
      DO 41 I = 1 , llc(ipp)           <------- 変数 llc による並列処理
        DO 60 J = 1 , MOw              <------- 変数 mow による並列処理
         AA(MO*(mol2(J)-1)+mol(I))=AA(MO*(mol2(J)-1)+mol(I))/DMS(mol(i))
   60   CONTINUE
        BB(mol(I)) = BB(mol(I)) / DMS(mol(i))
        BB(mol(I)) = BB(mol(I)) / DMS(mol(i))
   41 CONTINUE
      enddo
!xocl end spread
              .
              .
              .
C----- data transfer -----                    --------+
      idp=idvproc()                                    |
      do i=1,iipe                                      |
        lct(i)=0                                       |
      enddo                                            |
!xocl spread do                                        |
      do ipp=1,iipe                                    |
      do i=1,llc(idp)                                  |
        do j=1,mow                                     |
          lct(ipp)=lct(ipp)+1                          |
          lis = mo*(mol2(J)-1)+mol(I)                  |
          AAT(lct(ipp),ipp) = AA(lis)                  |
          AAL(lct(ipp),ipp) = lis                      | データ転送処理
        enddo                                          |
      enddo                                            |
      enddo                                            |
!xocl end spread sum(lct)                              |
!xocl spread do                                        |
      do i=1,iipe                                      |
        lctmx=lct(i)                                   |
        llcmx=llc(i)                                   |
      enddo                                            |
!xocl end spread max(lctmx),max(llcmx)                 |
!xocl spread move                                      |
      do j=1,iipe                                      |
        do i=1,lctmx                                   |
          AATW(i,j)=aatg(i,j)                          |
          AATL(i,j)=aalg(i,j)                          |
        enddo                                          |
      enddo                                            |
!xocl end spread (xt)                                  |
!xocl movewait (xt)                                    |
      do j=1,iipe                                      |
        do i=1,lct(j)                                  |
          aa(aatl(i,j))=aatw(i,j)                      |
        enddo                                          |
      enddo                                            |
```

Fig. 5.11 Addition and modification of RADCAL. (2/3)

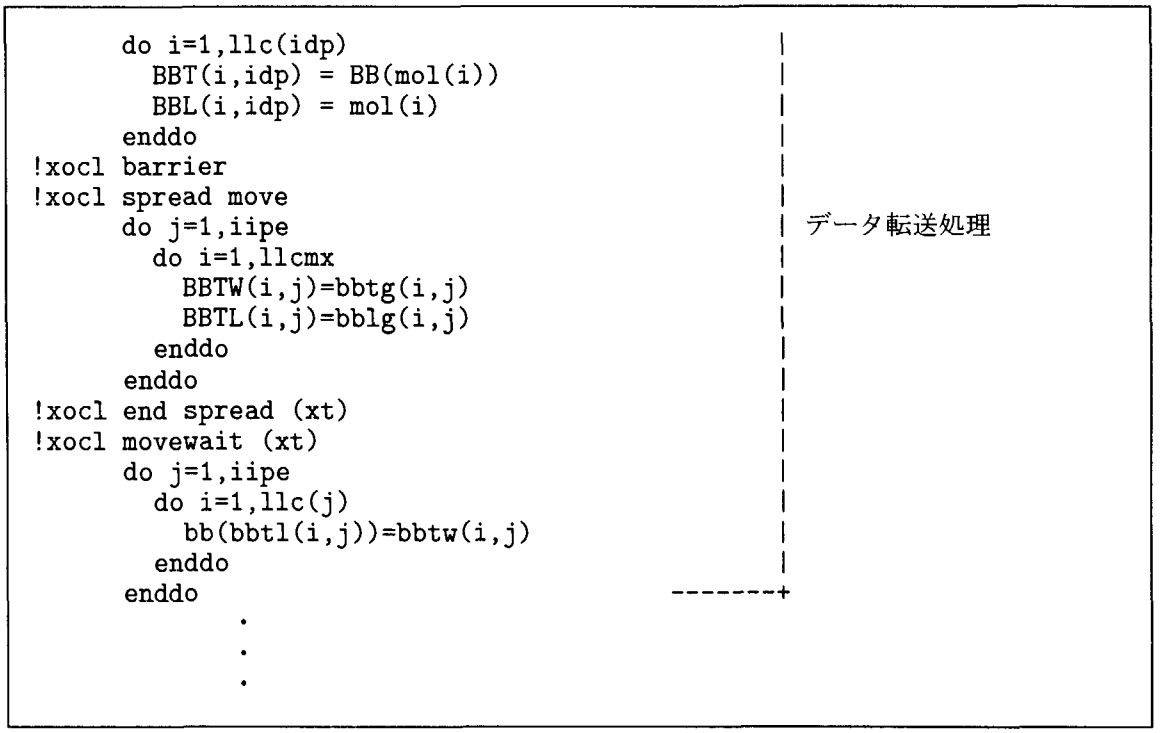

Fig. 5.11 Addition and modification of RADCAL. (3/3)

```
      SUBROUTINE  RADCON
      INCLUDE (PARAM0)
      INCLUDE (INP)
      INCLUDE (COM01)
      INCLUDE (COM00)
      INCLUDE (COM02)
      INCLUDE (COM05)
      INCLUDE (RADDBG)
      INCLUDE (ARRAY1)
      include (penum)              <------- インクルードファイル追加
      common/rdm/mol(momax),mol2(momax),mow,llc(iipe),llcmx
      PARAMETER ( PI=3.1415926 )
!xocl processor pe(iipe)                                    -------+
!xocl subprocessor pes(iipe)=pe(1:iipe)                            | 並列処理用宣
!xocl index partition qn=(proc=pes,index=1:iipe,part=band) --+ 言追加
              .
              .
      DO 10 N = 1 , LRAD
        DO 10 M = 1 , 2
          lzflg=0
          itpflg=0
!xocl spread do
          do ipp=1,iipe
          DO 40 LZ = 1 , MOw
            IF(IR(M,N).EQ.LINK(LZ,0,1).AND.JR(M,N).EQ.LINK(LZ,0,2).AND.
     1         KR(M,N).EQ.LINK(LZ,0,3).AND.ID(M,N).EQ.LINK(LZ,0,4)) then
              lzflg = lzflg + 1
            endif
   40     CONTINUE
          enddo
!xocl end spread sum(lzflg)
        if(lzflg.eq.0) then
          MO = MO + 1
          IF(MO.GT.MOMAX)  THEN
            WRITE(6,'(A,I5)')
     1      '      TOO MUCH SURFACE AT RADIATION CALCULATION : NUMBER OF
     2 SURFACE IS' , MO
            STOP
          ENDIF
          if(jr(m,n).ge.jstt .and. jr(m,n).le.jenn) then ----+
            itpflg=1                                         |
            mow=mow+1                                        |
            mol(mo)=mo                                       |
            mol2(mow)=mo                                     | 並列処理向け
            LINK(MOw,0,1) = IR(M,N)                          | インデックス
            LINK(MOw,0,2) = JR(M,N)                          | 定義
            LINK(MOw,0,3) = KR(M,N)                          |
            LINK(MOw,0,4) = ID(M,N)                          |
          endif                                          ----+
              .
              .
          endif
   10 CONTINUE
```

Fig. 5.12 Addition and modification of RADCON. (1/2)

```
              .
              .
      DO 70 M = 1 , MOw
        DO 60 N = 1 , NMAX(M)
          DO 50 MM = 1 , MOw              <------- 変数 mow による並列処理
            IF( LINK(MM,0,1).EQ.LINK(M,N,1) .AND.
     1          LINK(MM,0,2).EQ.LINK(M,N,2) .AND.
     2          LINK(MM,0,3).EQ.LINK(M,N,3) .AND.
     3          LINK(MM,0,4).EQ.LINK(M,N,4) ) then
                LINK(M,N,5) = MM
            endif
   50     CONTINUE
   60   CONTINUE
   70 CONTINUE
C     XXR : INITIAL SURFACE TEMPERATURE ('K)
      DO 80 M = 1 , MOw                   <------- 変数 mow による並列処理
      IF(IRSTRT.EQ.0)
     1   XXR(M) = TS(LINK(M,0,1),LINK(M,0,2),
     2                    LINK(M,0,3)) + 273.15
      TR(LINK(M,0,1),LINK(M,0,2),LINK(M,0,3),
     1   LINK(M,0,4)) = XXR(M) - 273.15
   80 CONTINUE
      DO 90 M = 1 , MOw                   <------- 変数 mow による並列処理
      IF( LINK(M,0,4).EQ.1 )
     1     LFLI( LINK(M,0,1)-1 , LINK(M,0,2)   , LINK(M,0,3)    ) = 1
      IF( LINK(M,0,4).EQ.2 )
     1     LFLI( LINK(M,0,1)   , LINK(M,0,2)   , LINK(M,0,3)    ) = 1
      IF( LINK(M,0,4).EQ.3 )
     1     LFLJ( LINK(M,0,1)   , LINK(M,0,2)-1 , LINK(M,0,3)    ) = 1
      IF( LINK(M,0,4).EQ.4 )
     1     LFLJ( LINK(M,0,1)   , LINK(M,0,2)   , LINK(M,0,3)    ) = 1
      IF( LINK(M,0,4).EQ.5 )
     1     LFLK( LINK(M,0,1)   , LINK(M,0,2)   , LINK(M,0,3)-1  ) = 1
      IF( LINK(M,0,4).EQ.6 )
     1     LFLK( LINK(M,0,1)   , LINK(M,0,2)   , LINK(M,0,3)    ) = 1
   90 CONTINUE
      do i=1,iipe                        ------+
        llc(i)=0                               |
      enddo                                    |
!xocl spread do                                |
      do ipp=1,iipe                            |
        do i=1,mo                              | インデックスソート処理
          if(i.eq.mol(i)) then                 |
            llc(ipp)=llc(ipp)+1                |
            mol(llc(ipp))=mol(i)               |
          endif                                |
        enddo                                  |
      enddo                                    |
!xocl end spread sum(llc)              ------+
              .
              .
```

Fig. 5.12 Addition and modification of RADCON. (2/2)

```
      SUBROUTINE  RADPRE(tsl,alamd,visc,pr)
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      INCLUDE (ARRAY3)
      INCLUDE (ARRAY1)
      INCLUDE (MATPR0)
      INCLUDE (COM01)
      INCLUDE (COM00)
      INCLUDE (COM02)
      INCLUDE (RADDBG)
      INCLUDE (COM05)
      include (penum)              <------- インクルードファイル追加
      common/rdm/mol(momax),mol2(momax),mow,llc(iipe),llcmx    ------+
      dimension tsl(li,lj,lk),vav(li,lj,lk)                          | 並
!xocl processor pe(iipe)                                             | 列
!xocl subprocessor pes(iipe)=pe(1:iipe)                              | 処
!xocl index partition qn=(proc=pes,index=1:JMAX,part=band)           | 理
!xocl local alamd(:,/qn(overlap=(1,1)),:)                            | 用
!xocl local visc(:,/qn(overlap=(1,1)),:)                             | 宣
!xocl local pr(:,/qn(overlap=(1,1)),:)                               | 言
!xocl local tsl(:,/qn(overlap=(1,1)),:)                              | 追
!xocl local vav(:,/qn(overlap=(1,1)),:)                              | 加
!xocl global tsg,vavg                                          ------+
C
!xocl overlapfix (vav) (xt)                          -----+
!xocl movewait (xt)                                  -----+ 袖転送処理追加
              .
              .
              .
!xocl spread do
      do ipp=1,iipe
        DO 20 L = 1 , MOw              <------- 変数 mow による並列処理
          I  = LINK(L,0,1)
          J  = LINK(L,0,2)
          K  = LINK(L,0,3)
          IDD = LINK(L,0,4)
              .
              .
          SB(L)=AH*UH             +AK*UK
          SC(L)=AH*UH*(TC+273.15)+AK*UK*(TSL(I,J,K)+273.15)
          SD(L)=                   AK*UK
   20   CONTINUE
      enddo
!xocl end spread
C
      DO 30 L = 1 , MOw                <------- 変数 mow による並列処理
        DO 30 N = 1 , NMAX(L)
          SA(L,N) = ARAD(L,N) * FRAD(L,N) * SIGMA * ERAD(L,N)
   30 CONTINUE
              .
              .
```

Fig. 5.13 Addition and modification of RADPRE.

```
      SUBROUTINE  TCAL(ITIME,TIME,tsl,alamd,visc,pr)
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      INCLUDE (ARRAY3)
      INCLUDE (ARRAY1)
      INCLUDE (MATPRO)
      INCLUDE (INP)
      INCLUDE (INLET)
      INCLUDE (COM01)
      INCLUDE (COM00)
      INCLUDE (COM05)
      include (penum)                <------- インクルードファイル追加
      common/gldm/ug(li,lj,lk),vg(li,lj,lk),
     &            wg(li,lj,lk),pg(li,lj,lk)
      real    tl(li,lj,lk),tg(li,lj,lk)
      real    qs(6,li,lj,lk)
      real    tsl(li,lj,lk)
      real    vav(li,lj,lk)
C
!xocl processor pe(iipe)                                          並列処理用宣
!xocl subprocessor pes=pe(1:iipe)                                 言追加
!xocl index partition qn=(proc=pes,index=1:JMAX,part=band)
!xocl local tl(:,/qn(overlap=(1,1)),:)
!xocl local vav(:,/qn(overlap=(1,1)),:)
!xocl local qs(:,:,/qn(overlap=(1,1)),:)
!xocl local alamd(:,/qn(overlap=(1,1)),:)
!xocl local visc(:,/qn(overlap=(1,1)),:)
!xocl local pr(:,/qn(overlap=(1,1)),:)
!xocl local tsl(:,/qn(overlap=(1,1)),:)
!xocl global tg,vavg,qsg
!xocl global tsg
              .
              .
      if(itcflg.eq.0) then
      do k=1,kmax
        do j=1,jmax
          do i=1,imax
            TO(I,J,K) = T(I,J,K)
          enddo
        enddo
      enddo
!xocl spread do /qn
      do j=1,jmax                                                 作業配列への
        do k=1,kmax                                               代入処理追加
          do i=1,imax
            tl(I,J,K) = T(I,J,K)
          enddo
        enddo
      enddo
!xocl end spread
      itcflg=1
```

Fig. 5.14 Addition of parallel parts for TCAL. (1/2)

```
      else                                                          |
!xocl overlapfix (tl) (xt)                                         |
!xocl movewait (xt)                                                |
      do j=jst,jen                                                 | 作業配列への
        do k=1,kmax                                                | 代入処理追加
          do i=1,imax                                              |
            to(I,J,K) = tl(I,J,K)                                  |
          enddo                                                    |
        enddo                                                      |
      enddo                                                        |
      endif                                                  ------+
c
!xocl spread do /qn                  <------- 並列指示行による並列処理
      DO  100 J=1, JMAX
        DO  100 K=1, KMAX
          DO  100 I=1, IMAX
            VAV(I,J,K) = FUNCV(U(I,J,K),V(I,J,K),W(I,J,K))
            QI(I,J,K) = 0.
  100 CONTINUE
!xocl end spread
!xocl overlapfix (vav) (xt)
!xocl movewait (xt)
              .
              .
              .
!xocl spread do /qn                  <------- 並列指示行による並列処理
      DO  2000 J=JS, JM
        DO  2000 I=IS, IM
          RI= 1.
          IF(ICYL.EQ.1) RI=RRI(I)-0.5*DELI(I)
              .
              .
              .
          IF(FLOWK(I,J, 1).EQ.0.) Tl(I,J,   1)=Tl(I,J,KM)
          IF(FLOWK(I,J,KM).EQ.0.) Tl(I,J,KMAX)=Tl(I,J, 2)
 2000 CONTINUE
!xocl end spread
              .
              .
              .
!xocl spread move                                      ------+
      do k=1,kmax                                             |
        do j=1,jmax                                           |
          do i=1,imax                                         | データ転送処理追加
            T(I,J,K) = TG(I,J,K)                              |
          enddo                                               |
        enddo                                                 |
      enddo                                                   |
!xocl end spread (xt)                                         |
!xocl movewait (xt)                                    ------+
              .
              .
              .
```

Fig. 5.14 Addition of parallel parts for TCAL. (2/2)

```
      SUBROUTINE  TSCAL(ITIME,TIME,tsl,alamd,visc,pr)
C
      INCLUDE (PARAM0)
      INCLUDE (PARAM1)
      INCLUDE (ARRAY3)
      INCLUDE (ARRAY1)
      INCLUDE (ARRAYT)
      INCLUDE (MATPRO)
      INCLUDE (INP)
      INCLUDE (COM01)
      INCLUDE (COM00)
      INCLUDE (COM04)
      INCLUDE (COM05)
      include (penum)              <------- インクルードファイル追加
      real    tsl(li,lj,lk),qs(6,li,lj,lk)                  ------+
      real    tso(li,lj,lk)                                       |
!xocl processor pe(iipe)                                          |
!xocl subprocessor pes(iipe)=pe(1:iipe)                           | 並列処理用
!xocl index partition qn=(proc=pes,index=1:JMAX,part=band)        | 宣言追加
!xocl local  tsl(:,/qn(overlap=(1,1)),:)                          |
!xocl local  tso(:,/qn(overlap=(1,1)),:)                          |
!xocl local  qs(:,:,/qn(overlap=(1,1)),:)                         |
!xocl global tsg,qsg                                       ------+
C
        do J=jst,jen
          do K=1,KMAX
            do I=1,IMAX
              TSO(I,J,K) = TSL(I,J,K)     <------- 代入元変更
            enddo
          enddo
        enddo
C     RADIATION CALCULATION : QDRAD
      IF(RAD)  CALL RADCAL(tsl,alamd,visc,pr)
!xocl spread do /qn                 <------- 並列指示行による並列処理
      DO  2000 J=JS, JM
        DO  2000 I=IS, IM
          RI= 1.
          IF(ICYL.EQ.1) RI=RRI(I)-0.5*DELI(I)
              .
              .
              .
        IF(FLOWK(I,J, 1).EQ.0.)  TSL(I,J,   1) = TSL(I,J,KM)
        IF(FLOWK(I,J,KM).EQ.0.)  TSL(I,J,KMAX) = TSL(I,J, 2)
 2000 CONTINUE
!xocl end spread
              .
              .
              .
```

Fig. 5.15 Addition and modification of TSCAL.

```
            .
            .
            .
      INCLUDE (MASS)
      INCLUDE (CROSS)
      INCLUDE (ICCG1)
      INCLUDE (COM01)
      INCLUDE (COM00)
      INCLUDE (COM05)
      include (penum)              <------- インクルードファイル追加
      REAL  BETA(LI,LJ,LK)
      common/gldm/ug(li,lj,lk),vg(li,lj,lk),wg(li,lj,lk),
     &            pg(li,lj,lk)
      real  a(lk,lj,li),b(lk,lj,li),c(lk,lj,li),ei(lk,lj,li)
      real  ds(lk,lj,li),delp(lk,lj,li)
      real  dsg(lk,lj,li),delpg(lk,lj,li)
      real  ul(li,lj,lk),vl(li,lj,lk),wl(li,lj,lk),pl(li,lj,lk)
      INTEGER NUM(LI,LJ,LK)
      INTEGER NO(LIJK),NOI(LIJK),NOJ(LIJK),NOK(LIJK),MP(4)
!xocl processor pe(iipe)
!xocl subprocessor pes(iipe)=pe(1:iipe)
!xocl index partition qn=(proc=pes,index=1:JMAX,part=band)
!xocl local  ul(:,/qn,:)
!xocl local  vl(:,/qn(overlap=(1,0)),:)
!xocl local  wl(:,/qn,:)
!xocl local  pl(:,/qn(overlap=(0,1)),:)
!xocl local  ds(:,/qn,:)
!xocl local  delp(:,/qn(overlap=(0,1)),:)
!xocl local  rho(:,/qn(overlap=(0,1)),:)
!xocl global ug,vg,wg,pg
!xocl global dsg,delpg
C
!xocl spread do /qn
      do j=1,lj
        do k=1,lk
          do i=1,li
            pl(i,j,k)=p(i,j,k)
          enddo
        enddo
      enddo
!xocl end spread
!xocl overlapfix (pl) (xt)
!xocl movewait (xt)
!xocl spread do /qn
      do j=1,lj
        do k=1,lk
          do i=1,li
            ul(i,j,k)=u(i,j,k)
            vl(i,j,k)=v(i,j,k)
            wl(i,j,k)=w(i,j,k)
          enddo
        enddo
      enddo
!xocl end spread
```

並列処理用宣言追加

分割ローカル配列への代入処理追加

Fig. 5.16 Addition of parallel parts for VFIELD. (1/2)

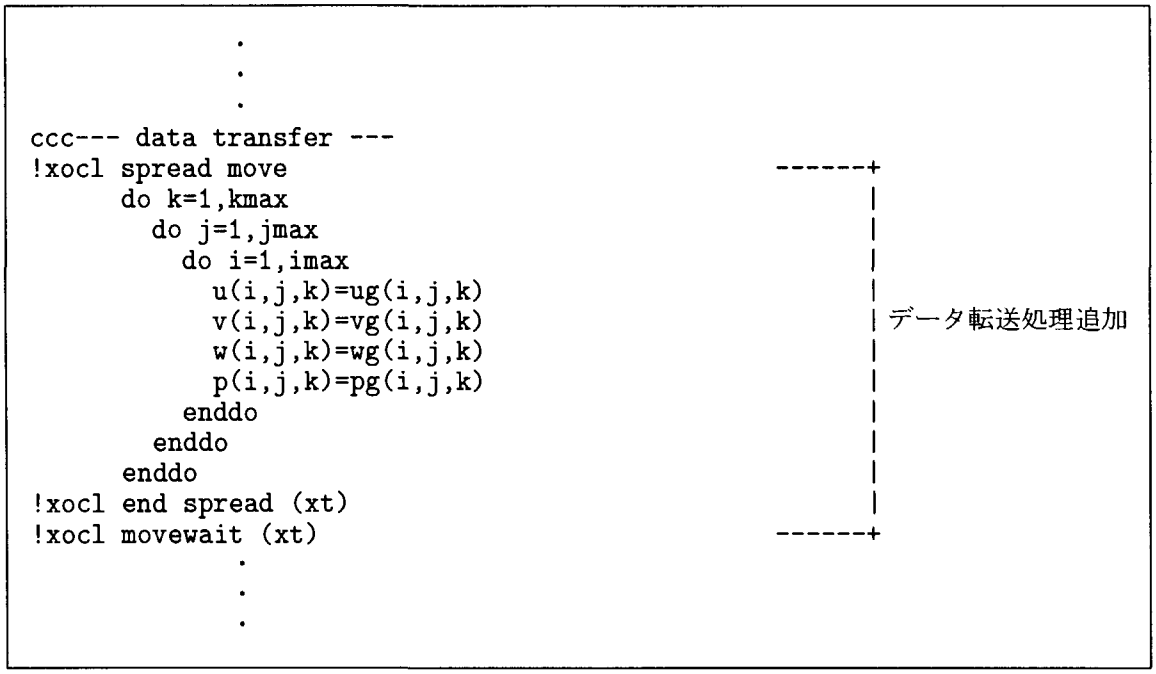

Fig. 5.16 Addition of parallel parts for VFIELD. (2/2)

```
              .
              .
              .
!xocl spread move                                        ------+
      do i=1,li                                                 |
        do j=1,lj                                              |
          do k=1,lk                                            |
            dsl(k,j,i)=dsg(k,j,i)                              | データ転送処理追加
            delpl(k,j,i)=delpg(k,j,i)                          |
          enddo                                                |
        enddo                                                  |
      enddo                                                    |
!xocl end spread (xt)                                          |
!xocl movewait (xt)                                      ------+
              .
              .
              .
      DO 30 I = 2 , no_p-1
        ist = lcnt(i) + 1       <----------+
        ien = lcnt(i+1)         <----------+ index で作成されたインデックスの
*vocl loop,novrec                          | 個数を設定
        DO 30 J = ist , ien     <----------+
          link=numiccg(j)       <------------ index で作成されたインデックス
          PD(link) = (R(link)                 を使用
     *               -EI(link-1)*PD(link-1)
     *               -C(link-kmax)*PD(link-kmax)
     *               -B(link-jkmax)*PD(link-jkmax))
     *               *D(link)
   30 CONTINUE
C
      DO 40 I = no_p-1 , 2 , -1
        ist = lcnt(i) + 1       <----------+
        ien = lcnt(i+1)         <----------+ index で作成されたインデックスの
*vocl loop,novrec                          | 個数を設定
        DO 40 J = ist , ien     <----------+
          link=numiccg(j)       <------------ index で作成されたインデックス
          P(link) = PD(link)                  を使用
     *             -(EI(link)*P(link+1)
     *             +C(link)*P(link+kmax)
     *             +B(link)*P(link+jkmax))
     *             *D(link)
   40 CONTINUE
```

Fig. 5.17 Addition and modification of ICCG0. (1/2)

```
              .
              .
              .
      DO 90 I = 2 , no_p-1
        ist = lcnt(i) + 1       <---------+
        ien = lcnt(i+1)         <---------+ index で作成されたインデックスの
*vocl loop,novrec                         | 個数を設定
        DO 90 J = ist , ien     <---------+
          link=numiccg(j)       <----------- index で作成されたインデックス
          PD(link) = (R(link)                を使用
     *               -EI(link-1)*PD(link-1)
     *               -C(link-kmax)*PD(link-kmax)
     *               -B(link-jkmax)*PD(link-jkmax))
     *               *D(link)
   30 CONTINUE
C
      DO 100 I = no_p-1 , 2 , -1
        ist = lcnt(i) + 1       <---------+
        ien = lcnt(i+1)         <---------+ index で作成されたインデックスの
*vocl loop,novrec                         | 個数を設定
        DO 100 J = ist , ien     <---------+
          link=numiccg(j)       <----------- index で作成されたインデックス
          P(link) = PD(link)                 を使用
     *             -(EI(link)*RD(link+1)
     *             +C(link)*RD(link+kmax)
     *             +B(link)*RD(link+jkmax))
     *             *D(link)
  100 CONTINUE
              .
              .
              .
!xocl spread move                          ------+
      do i=1,li                                     |
        do j=1,lj                                   |
          do k=1,lk                                 |
            delpg(k,j,i)=delpl(k,j,i)               | データ転送処理追加
          enddo                                     |
        enddo                                       |
      enddo                                         |
!xocl end spread (xt)                               |
!xocl movewait (xt)                        ------+
      RETURN
```

Fig. 5.17 Addition and modification of ICCG0. (2/2)

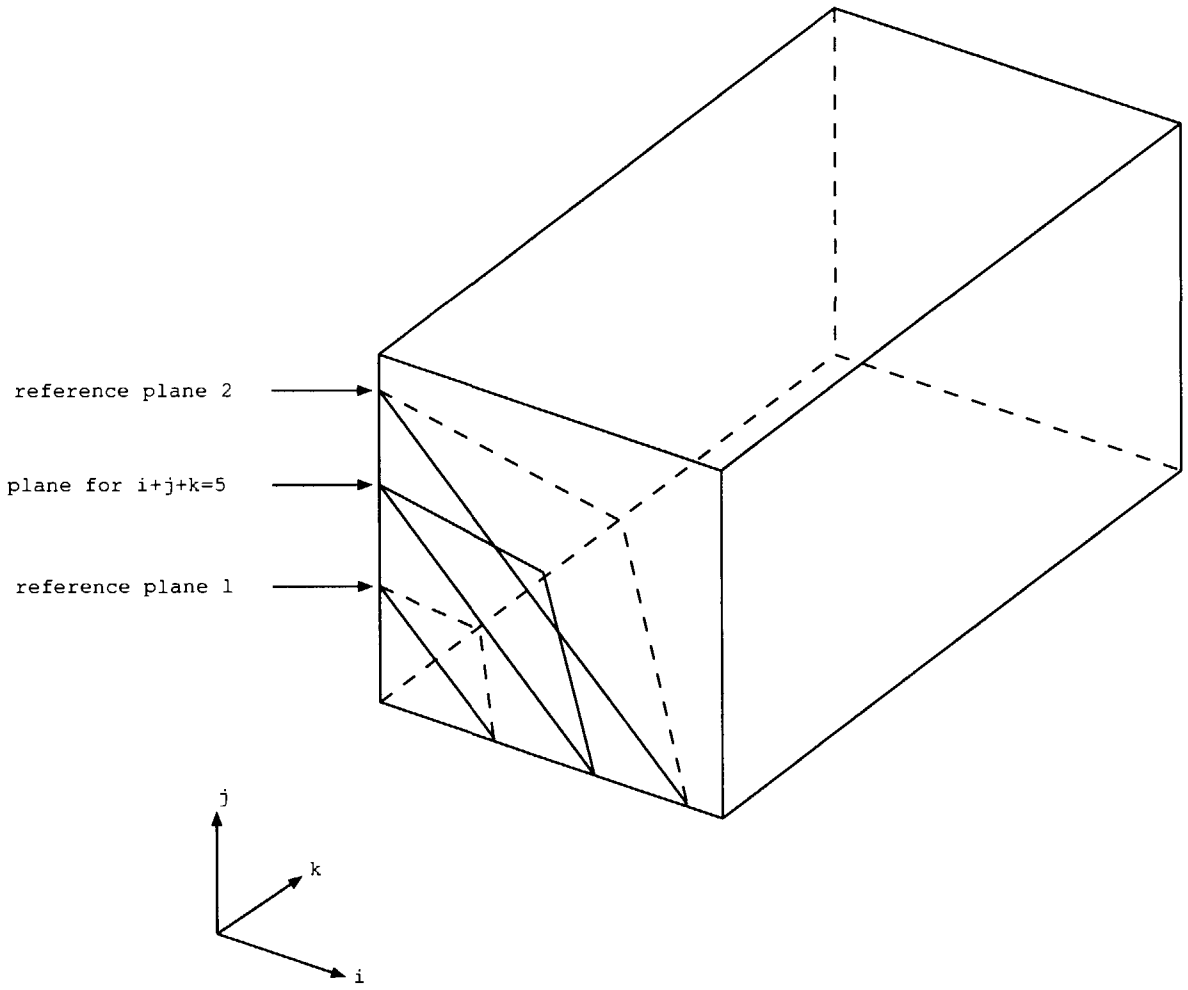

Fig. 5.18 Reference plane of Hyper-Plane.

```
      subroutine index
c
c Make index for hyper-plane.
c
      include (PARAM0)
      include (PARAM1)
      include (indx)
c
      no       = imax*jmax*kmax
      i_line   = kmax
      ij_plane = kmax*jmax
      no_p     = imax+jmax+kmax-2
c
c     initialize
c
      do 10 i = 1 , no
        lcnt(i) = 0
   10 continue
      do 15 i = 1 , no_p
        lplane_add(i) = 0
   15 continue
c
c     make index
c
      do 40 i = 1 , imax-2
         do 30 j = 1 , jmax-2
            ind = i*ij_plane + j*i_line
            do 20 k = 2 , kmax-1
               lplane_cnt(ind+k) = i + j + k
   20       continue
   30    continue
   40 continue
      do 50 j = 1 , no
        if(lplane_cnt(j).ne.0) then
          lplane_add(lplane_cnt(j)) = lplane_add(lplane_cnt(j)) + 1
        endif
   50 continue
      do 60 i = 1 , no_p
         lcnt(i+1) = lcnt(i) + lplane_add(i)
         lplane_add(i)  = lcnt(i)
   60 continue
      do 70 j = 1 , no
        if(lplane_cnt(j).ne.0) then
          lplane_add(lplane_cnt(j))    = lplane_add(lplane_cnt(j)) + 1
          numiccg(lplane_add(lplane_cnt(j))) = j
        endif
   70 continue
c
      return
      end
```

Fig. 5.19 "index" routine.

参 考 文 献

[1] 高田昌二，鈴木邦彦，鈴木嘉之; 大型構造機器実証試験ループ（ＨＥＮＤＥＬ）炉内構造物実証試験部（Ｔ 2）による受動的冷却システム試験解析コード，JAERI-Data/Code 95-005，1995 年 6 月.

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

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

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

6. SELENEJ コードのベクトル並列化

6.1 概要

MHD 平衡コード SELENEJ [1] のベクトル並列化を行なった．炉心プラズマ研究部では，核融合実験装置に専用計算機を直結し，実験中のデータを即座に解析しながらその結果をフィードバックすることを検討しており，本コードはこのデータ解析に用いられる．このため，１回当たりの計算量はさほど多くないが，即座に値を返す必要があるため，ベクトル並列化による高速化を施した．

SELENEJ コードのベクトル並列化は，最初にベクトル化を行い，次のステップで並列化を行なった．実行計算機は富士通（株）製 VPP500 （以下 VPP）である．

この報告書では，ベクトル化，並列化で行なった作業内容と共に，VPP，および NEC 製 SX-4 （以下 SX）で測定したベクトル化版の性能，および VPP 上でのベクトル並列化版の性能について述べる．

6.2 ベクトル並列化の目的

将来，核融合実験装置に専用計算機を直結し，実験中に短いサンプリング周期（100ms 以下）で出力される実験データを使って専用計算機が解析を行ない，この解析結果を実験装置にフィードバックすることを検討している．実験装置からのデータ出力周期は短いほどよいことから，専用計算機による解析（計算）時間を，極めて短くする必要がある．

専用計算機には，どの方式の計算機が適応するか，各種の計算機について検討を行なう．今回の調査では，メモリ分散型ベクトル並列計算機の適応性を調べるため，VPP 上でベクトル並列化を行ない，どの程度高速化されるか調査を行なった．

メモリ分散型並列計算機では，計算を各 PE(Processing Element) に振り分けて行なわせ，必要があれば PE 間で通信によるデータ転送を行なう．データ転送は，実際にネットワークによってデータの送受信を行なうことで成立するが，今回のような，極めて短い時間内にひとつのジョブを終了させるためには，効果的なベクトル並列化と共に，いかに（転送すべき）データ量を少なくするかが重要である．

6.3 コード概要

核融合炉におけるトカマクの MHD 平衡方程式 グラッド・シャフラノフ方程式を [1] 解いている．数値解法は２次元楕円型編微分方程式の自由境界問題を反復法で解いている．１方向は高速フーリエ変換，もう１方向は差分解法である．

6.4 ベクトル化

SELENEJ コードは，ベクトル化，並列化の順で高速化を行なった．この章ではベクトル化について述べる．

6.4.1 オリジナルコードの動的挙動調査

動的挙動解析ツール ANALYZER [2] を使用し，実際にコードを実行させてコスト分布・ベクトル化状況を調査した． ANALYZER の結果を Table 6.1に示す．同様に SAMPLER [3] による解析結果を Table 6.2に示す．

6.4.2 ベクトル化の指針

第1ステップとして，コストが高くベクトル化がされていないルーチンに対してベクトル化を行う．そして並列化を行なう．

(1) サブルーチン EQLIN

このサブルーチンはプラズマ圧力の等高線に沿って，線積分を行なうが，コストが高く，かつベクトル化されていないため，高速化が必要である．

等高線は複数あり，それぞれ線積分を行なうが，これらは一部の例外処理を除いて独立して計算することができる．よって，この軸をベクトル軸として，ベクトル化を行なう．

(2) サブルーチン SFFT

FFT ルーチンである．コストは高いがベクトル化されており，倍率も出ている．よって，ベクトル化の必要は無い．

(3) ファンクション FLUX

このルーチンは フラックス値を計算するファンクションである．すでにベクトル化されているので，簡単な高速化チューニングを加えるだけでよい．

(4) サブルーチン EQRCU

コストは比較的高く，かつベクトル化されていないため，高速化が必要である．

(5) サブルーチン FFFT2

FFT ルーチンである．コストは比較的高く，ベクトル化が不十分のため高速化が必要である．FFT は一般にベクトル化できる数値解法である．

6.4.3 ベクトル化のための変更

ここでは，いくつかの副プログラムに対して行なったベクトル化について述べる．

(1) サブルーチン EQLIN

このルーチンはベクトル化がまったくされていない．オリジナルの EQLIN のリストの一部を Fig. 6.1に示す．処理は NV 方向に線積分を行っている処理である． DO ループは存在するが，処理の都合上， DO ループ内で上向きの GO TO 文があり，これがベクトル化できない原因のひとつとなっている．ベクトル化の準備として，この DO ループをいくつかのブロックに分ける．この様子を Fig. 6.2に示す．

(b) の開始点の検索処理部分はベクトル化不可であるが，その他はベクトル化可能である． (d) のブロックは，このままの状態 (上向き GO TO 文がある) ではベクトル化できないため，計算の順序を変更する．このループは DO ループの回転方向 (NV 方向， DO 変数は M) が独立でないとベクトル化はできない．'GO TO 5' は変数 PSIO と配列 PSI の値によって実行されるかどうかが決定する．これが実行されるときは，刻み幅の値となる PSIO の値が変わるため， NV 方向の独立性が失われる．ただし，この処理は例外処理的な性質があり，今回の入力データでは実行されることは無かった．ループの独立性がないとベクトル化（並列化も）が不可能になるため，ここではコメント化し，必要な場合は別途対策を考えることにした．

これにより独立性があれば，計算順序を変えることによりベクトル化が可能となる．オリジナルの計算順序は Fig. 6.3にに示すものであるが，ベクトル化するためには， Fig. 6.4のような変更をする． GO TO 20 によって分岐するものをリスト MLT に保存し， 1 回の DO ループ処理が終了し，分岐するものがあれは再び DO ループを実行するようにした．参考までに Fig. 6.5にベクトル化した EQLIN のリストの一部を示す．

(2) ファンクション FLUX

このルーチンは FLUX 値を計算するファンクションであり，ベクトル化されている． DO ループ内に IF 文があり，この IF 文よってこれ以降の計算の実行の有無が決定される． IF 文以降の計算式は，この DO ループ内の計算式の約 50 %のコストを占めており，計算量が多い． VPP ，SX は通常にベクトル化した時の IF 文の処理は，マスク処理方式が採用される．この方法は，IF 文の真偽に係わらず実際に計算を行う方法で， CPU 時間もその分かかってしまう．

このルーチンでの IF 文の真率は 0 %であり，むだな CPU 時間を消費していることになる．このため， VPP においては '*VOCL STMT,IF(0)' という最適化制御行を挿入した．これにより，マスク処理からリストベクトル処理に変更がなされ，実際に IF 文が真の時のみ，計算が行なわれる．ただし， SX の場合は，マスク処理方式以外の処理にするには，条件が満たされないためできなかった．ベクトル化版は最適化制御行を挿入したのみなので， Fig. 6.6にベクトル化版 FLUX のリストの一部を示し，オリジナル版は省略する．

(3) サブルーチン FFFT2

このルーチンは高速フーリエ変換におけるフーリエ係数を求めるルーチンであるが，ベクトル化が不十分であり， DO ループ構造を変更することでベクトル化を行った．オリジナル版の FFFT2 のリストの一部を Fig. 6.7に示す．ここでベクトル化されていない DO 100, および DO 200 の 2 重ループに注目する．

ここでは NZ=65 であり， J に関して Fig. 6.7の (a), (b) 式で定義される範囲を調査すると次のようになる．

式 (a)　　FAINF1(1,I)　～　FAINF1(64,I)

式(b)　　FAINF1(64,I)　～　FAINF1(127,I)

ここで FAINF1(64,I) は2重に定義されることになりベクトル化できない．このため，2重に定義を避けるように変更した (他の同様な箇所も変更した) ．ベクトル化版のリストを Fig. 6.8に示す.

(4) サブルーチン EQRCU

このサブルーチンには，'*VOCL TOTAL,SCALAR' というスカラ処理を指示する最適化制御行が挿入されているため，ベクトル化できていない．調査の結果，このサブルーチンはベクトル処理を行っても，特に問題がないため，この制御行を外した．このサブルーチンは自動ベクトル化により，ベクトル処理が可能である．なお，最適化制御行をはずしただけなので，リストは省略する.

6.4.4　ベクトル化効果

ここではベクトル化を行なった SELENEJ コードの効果について述べる.

(1) コスト分布

6.4.3節で述べたベクトル化を行ったバージョンに対して ANALYZER，およびサンプラを実行した．この結果を，Table 6.3，Table 6.4に示す.

(2) VPP500，および SX-4 での CPU 測定

オリジナル版，およびベクトル化版を使って VPP，SX で実際に実行し CPU 時間を測定した．この結果を以下に述べる.

(i) VPP500

VPP での CPU 時間の測定結果を Table 6.5に示す．この表のベクトル化版における OPT=E，OPT=F はそれぞれインライン展開を行なわない，インライン展開を行なうを意味する.

(ii) SX-4

SX での CPU 時間の測定結果を Table 6.6に示す．この表のベクトル化版における Npi，pi はそれぞれインライン展開を行なわない，インライン展開を行なうを意味する.

6.5　並列化

ベクトル化版 SELENEJ コードを使って並列化を行った．しかし，経過時間 100ms 以内を目標に行ったが，実現することはできなかった．ここでは並列化のための手法，変更箇所，およびベクトル並列化版の性能について述べる．なお，並列化には VPP FORTRAN [4] を使用した.

6.5.1 並列化の指針

ベクトル化版のANALYZER，およびSAMPLERの結果より，SELENEJコードはSFFT，EQLIN，FLUX等のコストが多く，これらの並列化がキーポイントと思わる．

(1) サブルーチンSFFT

FFTルーチンである．FFTは並列化が一般に適合できる．SELENEJの場合も同様と思われる．

(2) サブルーチンEQLIN

M（プラズマ等高線）に関するDOループが独立しているので，これが並列化対象部分となる．ただし，保留事項として**6.4.3**節で述べたPSIOの取扱い，およびエラー処理の取扱いについては，ベクトル化同様，コメントのままとする．

(3) ファンクションFLUX

FLUXは，このルーチン自身を並列化するのではなく，これを呼び出している親ルーチンを並列化すべきである．これは，親ルーチンでの呼び出しは，すべてDOループ内で呼び出され，かつ独立して呼び出し可能のためである．FLUXの実行回数も多いため，負荷分散となる．

6.5.2 作業前の準備作業

SELENEJコードのベクトル並列化の作業に入る前に，正しくカーネル部分の時間計測ができることと，将来，核融合実験装置に直結した専用計算機で実行されるので，現在行っているI／Oとは異なる手法が使われる．これらのことより，オリジナルコードの一部を改良した．

(1) 入出力の削減

実験装置直結の専用計算機では，計算結果は制御情報として直接実験装置に返す．よって，現在行っているような（ディスクに対する）入出力は行わないことになる．そこで，なるべく，実行時間測定の中の入出力にかかる時間を除くために，入力データの読み込みを廃止することと，計算結果出力の削減を行った．

SELENEJコードでは入力データの読み込みは，サブルーチンSTAAR，およびAAOPTで行っている．ネームリストによる入力であるが，これを廃止し，BLOCK DATAによる定義にて行うようにした．しかし，ここでの入力に使用される変数，配列が他のサブルーチンでも使用（定義・参照）されているところがある．このような場合，BLOCK DATAで定義した値が生かされない場合が発生する．このため，一部の変数・配列においては，READ文があった場所の直後に代入文によって定義するようにした．例として，AAOPTの変更前と変更後の入力部分をFig. 6.9，Fig. 6.10に，また新規に作成したBLOCK DATAをFig. 6.11に示す．

出力に関しては，可能な限り削減した．SELENEJは機番6番以外での出力があるが，これら出力は10番，11番，12番，20番，60番，61番であり，この機番を使用したwrite文はすべてコメント化した．

6番に関する出力についても，入力データのエコーバック，計算途中の出力等はすべて無くし，解析計算終了時の結果出力のみにした．このために，大幅な出力削減になった．

(2) VFL の使用

前項で述べたように，入出力の削減を行ったが，サブルーチン EQOUT などは，WRITE 文による出力量が多く，I/O に要する時間が多い．出力先は UFS であるが，今回の並列化では，専用計算機のことを考慮しているので，なるべく I/O にかかる時間は削減するため，VFL に書き出すようにした．VFL への変更は，コードを変更するのではなく，実行シェルスクリプトを変更するのみである．6 番出力（標準出力）を VFL に書き出すには，標準出力の機番変更を行う必要があり，実行オプションによって，標準出力を SELENEJ コードでは使用しない機番 30 番に変更し，6 番の出力を setenv によって VFL 上のファイルと連結した．変更した実行シェルスクリプトを Fig. 6.12に示す．

(3) 時間計測ルーチンの挿入

SELENEJ コードは，実行開始後，初期化・モデル作成等があり，これら処理の後にカーネルの計算が始まる．今回の SELENEJ コードの計測対象はカーネルのみでよいので，メインルーチン SELENEJ の中のサブルーチン EQPVAC を呼び出した直後から最後の計算結果の出力を終えたところまでを測定の対象とした．それぞれの箇所に，経過時間を測定するサービスサブルーチン gettod，および CPU・VU 時間を測定する clockv を挿入した．この部分を Fig. 6.13に示す．

6.5.3 並列化のための変更

並列化版の作成には，ベクトル化版を使用するが，このコスト分布を表す ANALYZER，および SAMPLER の結果は，Table 6.3，Table 6.4に示した．

これらによると，サブルーチン EQLIN，FLUX，SFFT，EQOUT 等のコストが高い．よって，これらを中心に並列化を行った．

(1) サブルーチン EQLIN

EQLIN は，プラズマの境界線単位を軸として（M 方向）ベクトル化を行った．並列化においてもこれを軸として行う．

一般にベクトル化の軸と並列化の軸を同じにすると，ベクトル長が台数分に分割されることになるので，ベクトル化効果が薄れることになる．しかし，EQLIN は，これ以外には並列処理の軸がないため採用した．

ベクトル化版の EQLIN の一部を Fig. 6.14に示す．この中で，ベクトル処理をしているループでかつコストが高いものを並列化の対象とした．一番コストが高い線積分を行う DO 1240 ループについて説明する．

M 方向でベクトル化されており，今回の入力では，終値 NV=101 であるためベクトル長は（M=2 より始まっているので）100 である．しかし，Fig. 6.14の DO 1240 ループの直後にある上向き GOTO 文があるように，繰り返し処理されることがあるが，常にすべての等高線に対して行うのではなく，(繰り返し処理が進むにつれ）ある判定条件を満たした等高線については処理を行わない．このためベクトル化版では DO ループの終了時に，再び計算する等高線についてのリス

トを作成し，そのリストを元に再びDOループを実行する．このため，DOループの回転数は，毎回同じではなく減少していく．

並列化において，各PEに対し均等に負荷分散させることは必須であるが，上で述べたように，M方向を均等に分割しても各PEへのコストの均等な分割にはならない．この繰り返し処理について詳しく調査してみたところ，Mの値が大きくなるにつれ，繰り返し数が少なくなる傾向がある．Mの値ごとの平均繰り返し数のグラフをFig. 6.15に示す．X軸はMの範囲（2～101）を示し，Y軸は繰り返し数を示す．これによると，Mの値が2の時，最大の繰り返し160回であり，Mの値が（終値NVである）101の時，最小である14である．このため，M方向を均等分割すると，第1PEの負荷が高くなり，これ以外のPEに遊びが生じてしまう．

これらのことより，各PEで線積分を行うMの範囲を4PEを用いた並列処理時にはTable 6.7に示すように振り分けた．これは，今回用意された入力データ専用に並列化しているため，固定である．VPPのようなメモリ分散型計算機では，計算対象となるデータ（配列等）は，（計算を）担当するPE上に配置する必要がある．VPPでは，配列を分割して各PEに配置する機能がある．配列の分割は，Table 6.7で示したように，線積分対象の等高線を各PEに振り分けたので，これと同様に等高線に関する物理値をもつ配列を分割する．これらの配列は，ベクトル化の際に変数から配列（リスト）に変更を行ったものである．この分割した配列の宣言文をFig. 6.16に示す．VPPでは，変数・配列の概念として，ローカルとグローバルの2種類がある．ローカルとはひとつのPEでのみ有効なものであり，グローバルとはネットワークを介して，他のPEからアクセスできるものである．VPPでは，EQUIVALENCE文によってグローバル変数・配列とローカル変数・配列を結合することができ，ローカル変数・配列の内容をグローバル変数・配列を使って他のPEに転送することができる．そこで，今回分割した配列はローカル配列なので，グローバル配列を設け，EQUIVALENCE文によって結合している．

並列処理を行うDOループについて説明する．並列処理を行う場合，各PEで処理をするDOループの範囲が異なるため，各PEごとにDOループの初期値・終値を設定する必要がある．そこでFig. 6.17に示すように，設定を行った．このDOループの並列処理の指示は，DOループの直前にある'!XOCL SPREAD DO'によって行われる．この指示行は，直後のDOループの処理範囲を指定された分割範囲ごとにPEに割り当てる機能を持っている．この場合のように分割範囲が指定されていない場合は，均等に分割を行う．ここではDO変数Iが1から4までであり，4PEによる並列処理を行うようにしたので，第1PEから第4PEまで，それぞれIは1，2，3，4が割り当てられる．ここで，各PEでの(DOループの)初期値(変数ISTA)と終値(変数IEND)が定義される．なお，この部分は，1回のジョブの実行につき，1回のみ実行する仕様にした．

このように，各PEでDOループの初期値・終値が決定されたため，これらの値（変数）を使用したDOループは並列処理が可能になった．並列化版のEQLINの一部をFig. 6.18に示す．コストの高い部分は，並列処理が可能になった．ただし並列化に伴い，データ転送も生じることになった．EQLINでは，2箇所でデータ転送を行う必要がある．

前者の転送はDO 1230，およびDO 1240ループで計算された配列RSU，ZSU，CSU，および変数NSUである．これらは，それぞれのループにおいて，M = 2のときのみ定義される．よって，必ず第1PEで定義される．このため，第1PEから他のPEへ転送を行う．転送の命令は'!XOCL BROADCAST'を使用した．

後者の転送は，DO 1230，DO 1240，およびDO 1250の線積分の処理で定義されたいくつかの（ローカル）配列を，全PEに転送するものである．この転送は，DO 8100で行っている．これは，DOループの形になっているが，'!XOCL SPREAD MOVE'と'!XOCL END SPREAD'で囲むことにより，転送命令となる．右辺のグローバル配列から左辺の重複ローカル配列（各PEに存在するローカル配列）に転送することになる．

'!XOCL SPREAD MOVE'による転送では，'!XOCL MOVEWAIT'によって転送の終了の同期をとる．転送開始から終了の同期をとる間に，他の計算を並行して行うことができる．このため，線積分の一部の式をこの部分で行うようにした．

(2) サブルーチンSFFT

FFT（高速フーリエ変換）を行うサブルーチンである．このルーチンは大部分が2重ループによって構成されている．構造的には，ほぼベクトル処理がされている．いくつかの多重ループが存在し，内側がDO変数Iによる回転数NR(データセット数）の（ベクトル化済み）ループである．並列化は，このループに対して行った．通常，多重ループの場合は，外側を並列化の軸とし，内側ループのベクトル性能を生かすことが多いが，このサブルーチンの場合は外側ループの回転数が実行中に変化することが多い．回転数が変化するループは並列化が困難である．このため内側ループに対し並列化を行った．

並列処理をするループの初期値，終値の設定は，サブルーチンEQLIN同様，前もって各PEごとに決定した．この決定部分をFig. 6.19に示す．NRの値はSFFTを呼び出す親ルーチンによって，（今回のデータでは）65，あるいは63である．4 PEでの並列処理における各PEでの負荷について，Table 6.8，およびTable 6.9に示す．これら初期値，終値を設定した後，各ループにおいて並列処理を行う．典型的な例として，オリジナルをFig. 6.20に，並列化したループをFig. 6.21に示す．ただし，一番コストの高いループ（DO 500 J = 1，NOO）に関しては，NOOの値が127の固定値のため，このループを内側，外側ループを並列ループにしてベクトル性能の低下を防いだ．このループのオリジナルをFig. 6.22，並列化したループをFig. 6.23に示す．

並列化されたループの内部で使用される配列について説明する．配列C，X，およびYの3種類である．この中で，Cは参照のみなので，並列化に特に処置をする必要はない．X，Yは定義・参照があるが，Yに関してはこのサブルーチンのみで使われ，Xは（引数渡しで）他の副プログラムで使用される．このため，このサブルーチン終了時には，すべてのPEのXに同一の値が定義されている必要がある．そこで，並列処理をした場合，計算が行われたPEより全PEにデータ転送を行うようにした．データ転送を行うには，EQLINで行った'!XOCL SPREAD MOVE'を使用する．このため，Yをデータ分割することとグローバル配列を設け，このグローバル配列とEQIVALENCEで共有させた．なお，Yは，サブルーチンFACR2で領域確保され，引数によってSFFTに渡されていた．この引数（配列）はSFFT内でしか使用されないので，引数による受渡しを廃止し，このサブルーチン内で領域を確保するようにした．Yに関する宣言文をFig. 6.24に示す．

データ転送に関しては，SFFTの最後の部分で配列Yのグローバル配列YGを使用して全PEの配列Xに転送した．この転送部分をFig. 6.25 に示す．

(3) ファンクション FLUX

ファンクション FLUX はサブルーチン EQBND から呼ばれる関数で，フラックス値を求めている．元々（オリジナル版より）ベクトル化されているルーチンである．並列化は，直接 FLUX に対して行うのではなく，EQBND に対して行う．これは，FLUX の呼び出しが並列で呼び出すことが可能なためである．オリジナルの EQBND の呼び出し部分を Fig. 6.26に示す．例えば DO 220 ループについて注目すると，配列 PSI は FLUX によって定義された後，総和計算を行う．FLUX による配列 PSI の定義部分は独立であることがわかる．DO 240，および DO 260 に関しても定義される PSI の範囲は異なるが，同様な構造になっている．

そこで，これらを別ループとして独立させ，並列ループとして実行する．DO 220 における回転数は NR，DO 240，および DO 260 の回転数は (NZ-1) である．よって，回転数の同じものはひとつのループにまとめた．並列化したこれらのループを Fig. 6.27に示す．並列処理をするループは DO 810，および DO 820 である．DO 240，および DO 260 において定義される PSI の添字 IM，IP は，ループ内で計算されて決定している．この計算式は総和計算の形をしており，単純に（そのままの形では）並列ループの中では使用できない．また，並列ループの実行後のデータ転送においても '!XOCL SPREAD MOVE' を使用するが，添字の指定方法には条件があり，このままの形では使用できない．そこで，並列ループを実行する前に 1 度だけ添字のリストを作成し，このリストを並列処理とデータ転送に使用することにした（リスト作成，および並列ループ直後の '!XOCL SPREAD MOVE' による転送部分は Fig. 6.27に含んでいる）．

並列処理を行うループ内では，PSI を直接使用せず，分割した作業用ローカル配列を使用した．これは，'!XOCL SPREAD MOVE' による条件によるものである．この作業用ローカル配列の宣言部分を Fig. 6.28に示す．

データ転送に関しては，なるべく転送時間を他の計算と同時に行うようにするため，転送終了の同期取り（'!XOCL MOVEWAIT'）の間に，転送しているデータ（PSI）とは無関係な処理（範囲の異なる PSI の計算部分）を行うようにした．この部分を Fig. 6.29に示す．

6.6 ベクトル並列化版の性能調査

ベクトル並列化版を使って性能調査を行った．並列化前のベクトル化版と 4PE，および 16PE でのベクトル並列化版のカーネル部分の経過時間と，並列化のための変更を行なったサブルーチン，および I/O が多いサブルーチン EQOUT を個別に測定したものを Table 6.10に示す．

これによると，4 PE，16PE の台数効果がでていないことがわかる．そこで，サブルーチンごとに詳しく調査してみた．

(1) サブルーチン EQLIN

EQLIN で一番コストが高い部分は，Fig. 6.18で示した「線積分 I」を行なう DO 1240 ループである．もともとベクトル長が 100 で開始し，処理が進むにつれてベクトル長が短くなっていく．ベクトル長が長い間はベクトル化効果があり，高速処理がされる．一方，並列化版は Table 6.7に示したように，最初のベクトル長がそれぞれ（4PE の場合），15，19，23，43 で始まり，処理が進むにつれさらに短くなっていく．このため，ベクトル化効果は十分に発揮されず，極端に短くなった場合は，スカラ処理よりも遅くなる．このため，台数分の効果はでない．Table 6.11に，このループの経過時間を示す．

また，データ通信に関しても，２箇所で行なっている．前者の通信は‘!XOCL BROADCAST’は，‘!XOCL SPREAD MOVE’のように，転送の終了の同期をとる間に他の計算はできない．よって，データ通信にかかる時間は，そのままプログラムの実行時間に反映する．‘!XOCL BROADCAST’，および‘!XOCL SPREAD MOVE’にかかった経過時間をTable 6.12に示す．これによると，PE数が多くなるにつれ，経過時間も増加することがわかる．また，EQLINの処理時間に占める割合も高いことがわかる．

(2) サブルーチンSFFT

SFFTも並列化により，ベクトル長が短くなりベクトル性能の低下で台数効果が出ていない．いくつかの（並列化された）多重ループにおいて，最内ループが並列の軸となっている．元々のベクトル長は65，または63であり，これを並列処理のために分割したため，4PEでは17，16PEでは5となる．試験的に外側のループ(DO J = 1, NHB-1)を内側（ベクトル）ループに，並列ループを外側にして測定したが，こちらの方が効果は悪かった．これは，NHBの値が64，32，16，8，4，2であり，極端にベクトル長が短くなる場合があるためである．

また，データ転送に関してであるが，最後の部分で‘!XOCL SPREAD MOVE’で行なっている（Fig. 6.25参照）．‘!XOCL SPREAD MOVE’は，‘!XOCL MOVEWAIT’によって転送終了の同期をとり，この間で他の計算を行なうことができることは前述したが，SFFTではルーチンの最後の部分で行なうので，他の計算がない．このため，転送時間がそのままプログラムの処理時間に反映する．ここでの転送時間をTable 6.13に示す．

(3) ファンクションFLUX，サブルーチンEQBND

FLUXに関しては，直接このサブルーチンを並列化したのではなく，親ルーチンであるEQBNDでの呼び出しを並列に呼び出している．このため，FLUX自体の処理時間は並列処理による台数効果は期待できる．しかし，EQBNDにおいて，FLUXによって求められた値を転送する必要があり，ここでデータ転送に要する時間が発生する．データ転送に要する時間の一部は他の計算を同時に行なうことによって，抹消されている．しかし，この他の計算部分も計算量が多いとは言えないため，実際に測定した．計測の方法は最初の‘!XOCL SPREAD MOVE’の開始から，最後の‘!XOCL MOVEWAIT (ID3)’の終了に要する時間から，他の計算にかかった時間を除いたものである．この結果をTable 6.14に示す．

この表によれば，16PEの場合はデータ転送の時間が増加していることがわかる．FLUXの処理自体は台数効果が出ているので，データ転送時間の増加がFLUXを含むEQBNDの処理時間を増加させている．

6.7 まとめ

今回の並列化の目標処理時間の100ms以下は達成することはできなかった．達成できない原因はベクトル処理効果の低減と通信時間の２点に集約されると思われる．ベクトル処理効果の低減は，コードに依存するものであるが，ベクトル処理の軸と並列処理の軸の両方が存在すれば，防ぐことができる．しかし，EQLINVの場合は軸が１つしかないため，（ベクトル処理）効果の低減は免れなかった．また，各PEの負荷分散という点に着目すると，処理の分割の方法で難題

があった．今回は１つの入力データ専用ということで，処理分割を決定したが，実際には（ある程度）種々の入力データに対応できるように並列化を行なう必要がある．均等な負荷分散を完全に行うことは，困難であろう．

Table 6.1 Dynamic behavior of original version (ANALYZER).

name	count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect	overhd
EQLIN	20	65015313	73.4	65015313	17.2	2	0.0	1.0- 1.0	0
SFFT	58	7185343	8.1	.1232E09	32.6	61	99.3	14.1- 21.0	0
FLUX	5309	5112477	5.8	.1365E09	36.2	98	99.9	19.2- 34.1	0
EQRCU	21	4426437	5.0	4426437	1.2	62	0.0	1.0- 1.0	0
FFFT2	29	3172977	3.6	4889777	1.3	63	37.0	1.5- 1.6	0
FAIC2	29	1164125	1.3	20200356	5.4	64	99.3	14.2- 21.3	0
JJMAIN	1	490975	0.6	3760037	1.0	184	89.6	7.3- 7.9	0
EQBND	20	442487	0.5	910780	0.2	10	69.4	1.5- 2.3	0
EQRBP	21	342289	0.4	6447357	1.7	62	99.8	15.2- 23.6	0
EQVACT	9	278835	0.3	683289	0.2	54	62.9	2.4- 2.5	0
PSIC2	29	230996	0.3	3891742	1.0	63	99.1	13.9- 20.6	0
DRCAL	30	148080	0.2	148080	0.0	32	0.0	1.0- 1.0	0
GRZ2	29	113783	0.1	1935489	0.5	63	99.2	14.0- 20.8	0
TABLE	30	68970	0.1	68970	0.0	9	0.0	1.0- 1.0	0
DINV	8	39799	0.0	49328	0.0	3	53.2	1.0- 1.4	0
EQADJ0	8	34449	0.0	183848	0.0	12	96.3	3.5- 6.9	0
EQOUT	1	34177	0.0	169309	0.0	50	84.5	4.6- 5.2	0
DALU	8	26264	0.0	26264	0.0	3	0.0	1.0- 1.0	0
CLEAR	29	25866	0.0	1197265	0.3	4128	100.0	39.9- 53.9	0
EQCHK	20	24945	0.0	601370	0.2	69	99.5	17.7- 30.1	0
EQIND	1	23905	0.0	463793	0.1	94	99.6	15.7- 24.1	0
EQADJ	20	22561	0.0	560352	0.1	141	99.0	18.8- 29.3	0
TRANFR	14	21580	0.0	47712	0.0	40	58.4	2.2- 2.2	0
SEEDCR	20	17580	0.0	357020	0.1	67	99.6	15.9- 25.3	0
ERATO	1	16137	0.0	68178	0.0	65	80.4	4.0- 4.4	0
BVACF	7	15916	0.0	16913	0.0	33	9.3	1.1- 1.1	0
INIPSI	1	13905	0.0	284087	0.1	125	99.8	16.5- 25.5	0
WRTSK	11	11176	0.0	11176	0.0	45	0.0	1.0- 1.0	0
FLXTAB	1	10220	0.0	473199	0.1	1025	100.0	39.9- 53.9	0
RESETR	69	9524	0.0	407813	0.1	590	99.8	37.3- 49.2	0
AMAX	1013	9117	0.0	9117	0.0	1	0.0	1.0- 1.0	0
AMIN	973	8757	0.0	8757	0.0	1	0.0	1.0- 1.0	0
JJFLUX	20	6992	0.0	151340	0.0	67	99.7	16.7- 27.1	0

Table 6.2 Dynamic behavior of original version (SAMPLER).

```
Status                          : Serial
Number of Processors            : 1

Type                            : cpu
Interval (msec)                 : 1

Synthesis Information
    Count|    Percent|     VL| Name
       74|       54.0|      -|  eqlin_
       18|       13.1|    133|  flux_
       11|        8.0|     64|  sfft_
        5|        3.6|      -|  MAIN__
        4|        2.9|      -|  eqchk_
        4|        2.9|      -|  eqrcu_
        3|        2.2|      -|  staar_
        3|        2.2|      -|  wrtsk_
        2|        1.5|      -|  ffft2_
        2|        1.5|      -|  jjmain_
        2|        1.5|      -|  seedcr_
        2|        1.5|      -|  eqout_
        2|        1.5|      -|  erato_
        1|        0.7|      -|  jobinf_
        1|        0.7|      -|  stplsm_
        1|        0.7|      -|  rarray_
        1|        0.7|     63|  faic2_
        1|        0.7|      -|  ivar_

      137|           |     84|  TOTAL
```

Table 6.3 Dynamic behavior of vector version (ANALYZER).

name	count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect	overhd
SFFT	58	7185343	32.8	.1232E09	30.8	61	99.3	14.1- 21.0	0
EQLIN	20	5350124	24.4	87752977	21.9	39	99.0	13.6- 19.9	0
FLUX	5309	5112477	23.3	.1365E09	34.1	98	99.9	19.2- 34.1	0
FAIC2	29	1164125	5.3	20200356	5.1	64	99.3	14.2- 21.3	0
JJMAIN	1	490975	2.2	3760037	0.9	184	89.6	7.3- 7.9	0
EQBND	20	442487	2.0	910780	0.2	10	69.4	1.5- 2.3	0
EQRBP	21	342289	1.6	6447357	1.6	62	99.8	15.2- 23.6	0
EQRCU	21	314542	1.4	4426437	1.1	62	97.9	12.0- 16.5	0
FFFT2	29	283571	1.3	4861937	1.2	62	99.3	14.1- 21.0	0
EQVACT	9	278835	1.3	683289	0.2	54	62.9	2.4- 2.5	0
PSIC2	29	230996	1.1	3891742	1.0	63	99.1	13.9- 20.6	0
DRCAL	30	148080	0.7	148080	0.0	32	0.0	1.0- 1.0	0
GRZ2	29	113783	0.5	1935489	0.5	63	99.2	14.0- 20.8	0
TABLE	30	68970	0.3	68970	0.0	9	0.0	1.0- 1.0	0
DINV	8	39799	0.2	49328	0.0	3	53.2	1.0- 1.4	0
EQADJ0	8	34449	0.2	183848	0.0	12	96.3	3.5- 6.9	0
EQOUT	1	34177	0.2	169309	0.0	50	84.5	4.6- 5.2	0
DALU	8	26264	0.1	26264	0.0	3	0.0	1.0- 1.0	0
CLEAR	29	25866	0.1	1197265	0.3	4128	100.0	39.9- 53.9	0
EQCHK	20	24945	0.1	601370	0.2	69	99.5	17.7- 30.1	0
EQIND	1	23905	0.1	463793	0.1	94	99.6	15.7- 24.1	0
EQADJ	20	22561	0.1	560352	0.1	141	99.0	18.8- 29.3	0
TRANFR	14	21580	0.1	47712	0.0	40	58.4	2.2- 2.2	0
SEEDCR	20	17580	0.1	357020	0.1	67	99.6	15.9- 25.3	0
ERATO	1	16137	0.1	68178	0.0	65	80.4	4.0- 4.4	0
BVACF	7	15916	0.1	16913	0.0	33	9.3	1.1- 1.1	0
INIPSI	1	13905	0.1	284087	0.1	125	99.8	16.5- 25.5	0
WRTSK	11	11176	0.1	11176	0.0	45	0.0	1.0- 1.0	0
FLXTAB	1	10220	0.0	473199	0.1	1025	100.0	39.9- 53.9	0
RESETR	69	9524	0.0	407813	0.1	590	99.8	37.3- 49.2	0
AMAX	1013	9117	0.0	9117	0.0	1	0.0	1.0- 1.0	0
AMIN	973	8757	0.0	8757	0.0	1	0.0	1.0- 1.0	0
JJFLUX	20	6992	0.0	151340	0.0	67	99.7	16.7- 27.1	0

Table 6.4 Dynamic behavior of original version (SAMPLER).

```
Status                        : Serial
Number of Processors          : 1

Type                          : cpu
Interval (msec)               : 1

Synthesis Information
    Count|   Percent|    VL| Name
       22|      28.6|    66|  eqlin_
       13|      16.9|   200|  flux_
        8|      10.4|    64|  sfft_
        5|       6.5|     -|  MAIN__
        4|       5.2|     -|  seedcr_
        4|       5.2|     -|  eqchk_
        3|       3.9|     -|  staar_
        3|       3.9|     -|  jjmain_
        2|       2.6|    64|  faic2_
        2|       2.6|     -|  eqout_
        2|       2.6|     -|  wrtsk_
        1|       1.3|     -|  jobinf_
        1|       1.3|     -|  stplsm_
        1|       1.3|     -|  eqbnd_
        1|       1.3|     -|  eqrbp_
        1|       1.3|     -|  vachk_
        1|       1.3|    63|  eqrcu_
        1|       1.3|  2048|  tranfr_
        1|       1.3|     -|  erato_
        1|       1.3|     -|  staad_

       77|          |   144|  TOTAL
```

Table 6.5 Execution time on VPP500 (sec.).

	オリジナル		ベクトル化版	
	スカラ	ベクトル	OPT= E	OPT=F
REAL	9.51	11.44	8.08	17.41
CPU	4.35	1.33	0.80	0.75
SYS	0.27	0.28	0.27	0.27
VU	0.0	0.25	0.44	0.44

Table 6.6 Execution time on SX-4 (sec.).

	オリジナル		ベクトル化版	
	スカラ	ベクトル	Npi	pi
REAL	3.22	1.25	0.77	0.65
CPU	2.44	0.80	0.464	0.468
SYS	0.04	0.02	0.01	0.2
VU	0.0	0.12	0.24	0.24

Table 6.7 Each PE's charge (EQLIN).

	1PE	2PE	3PE	4PE
M	2 ～ 16	17 ～ 35	36 ～ 58	59 ～ 101
処理本数	15	19	23	43

Table 6.8 Each PE's charge (NR = 65, SFFT).

	1PE	2PE	3PE	4PE
I	1 ～ 17	18 ～ 34	35 ～ 51	52 ～ 65
処理本数	17	17	17	14

Table 6.9 Each PE's charge (NR = 63, SFFT).

	1PE	2PE	3PE	4PE
I	1 ～ 17	18 ～ 34	35 ～ 51	52 ～ 63
処理本数	17	17	17	12

Table 6.10 Execution time of parallel version (msec.).

バージョン	カーネル全体	EQLIN	SFFT	EQBND	EQOUT
ベクトル化版 (1PE)	487	214	76	145	18
並列化版 (4PE)	378	172	67	73	28
並列化版 (16PE)	455	237	87	61	32

Table 6.11 Execution time of DO 1240 loop in EQLIN (msec.).

	ベクトル化版	並列化版 (4PE)	並列化版 (16PE)
時間	211.9	129.6	110.8

Table 6.12 Data transfer time in EQLIN (msec.).

転送命令	ベクトル化版	並列化版 (4PE)	並列化版 (16PE)	実行回数
!XOCL BROADCAST	—	13.8	22.1	20
!XOCL SPREAD MOVE	4.5	35.8	115.1	20

Table 6.13 Data transfer time in SFFT (msec.).

転送命令	ベクトル化版	並列化版 (4PE)	並列化版 (16PE)	実行回数
!XOCL SPREAD MOVE	—	23.1	44.8	58

Table 6.14 Data transfer time in EQBND (msec.).

転送命令	ベクトル化版	並列化版 (4PE)	並列化版 (16PE)	実行回数
!XOCL SPREAD MOVE	—	25.1	39.5	3ヶ所× 20

```
      SUBROUTINE EQLIN
      . . . .
      DO 120 M=2,NV
      N=N-1
      PSI0=PSI0+DPSI
   5  SIW(N)=PSI0
C-----------------------------------------------------------------------
C..SEARCH STARTING POINT
      I=IST+1
      IR=IRST+1
      IZ=NZ
  10  I=I-1
      IR=IR-1
      IF(IR.LT.IRAXIS)GOTO 800
      IF((PSI0-PSI(I))*(PSI0-PSI(I+1)).GT.0.D0)GOTO 10
      IST=I
      IRST=IR
C-----------------------------------------------------------------------
C..INITIALIZE LINE INTEGRALS
C
      BFAV(N)=0.D0
      VLV(N)=0.D0
      SDW(N)=0.D0
      CKV(N)=0.D0
      SSV(N)=0.D0
      AAV(N)=0.D0
      BBV(N)=0.D0
      RRV(N)=0.D0
      X=(PSI0-PSI(I))/(PSI(I+1)-PSI(I))
      R1=RG(IR)+X*DR
      Z1=ZG(IZ)
      BP1=(RBP(I)+X*(RBP(I+1)-RBP(I)))/R1
      VL1=R1*R1
      DS1=1.D0/BP1
      CK1=BP1
      SS1=VL1*BP1
      AA1=DS1/VL1
      RR1=VL1/BP1
      ZBT=RBV(N)/R1
      BV1=BP1+ZBT**2/BP1

      BFAV1=SQRT(BP1**2+ZBT**2)/BP1
C..TRACE CONTURE
      K=1
      IF(N.NE.NV)GOTO 20
      NSU=1
      RSU(1)=R1
      ZSU(1)=Z1
      CSU(1)=BP1
  20  R0=R1
```

Fig. 6.1 Original version of subroutine EQLIN(1/4).

```
      Z0=Z1
      BP0=BP1
      VL0=VL1
      DS0=DS1
      CK0=CK1
      SS0=SS1
      AA0=AA1
      RR0=RR1
      BV0=BV1
C
      BFAV0=BFAV1
C
      I1=I
      I2=I1+1
      I3=I2-NR
      I4=I1-NR
      S1=PSI0-PSI(I1)
      S2=PSI0-PSI(I2)
      S3=PSI0-PSI(I3)
      S4=PSI0-PSI(I4)
      IF(S1*S2.LT.0.D0.AND.K.NE.1)GOTO 30
      IF(S2*S3.LT.0.D0.AND.K.NE.2)GOTO 40
      IF(S3*S4.LT.0.D0.AND.K.NE.3)GOTO 50
      IF(S4*S1.LT.0.D0.AND.K.NE.4)GOTO 60
      PSI0=PSI0+1.D-05*SAXIS
      GOTO 5
   30 X=S1/(S1-S2)
      R1=RG(IR)+DR*X
      Z1=ZG(IZ)
      BP1=(RBP(I1)+X*(RBP(I2)-RBP(I1)))/R1
      I=I+NR
      IZ=IZ+1
      K=3
      GOTO 70
   40 X=S2/(S2-S3)
      R1=RG(IR+1)
      Z1=ZG(IZ)-DZ*X
      BP1=(RBP(I2)+X*(RBP(I3)-RBP(I2)))/R1
      I=I+1
      IR=IR+1
      K=4
      GOTO 70
   50 X=S4/(S4-S3)
      R1=RG(IR)+DR*X
      Z1=ZG(IZ-1)
      BP1=(RBP(I4)+X*(RBP(I3)-RBP(I4)))/R1
      I=I-NR
      IZ=IZ-1
      K=1
      GOTO 70
```

Fig. 6.1 Original version of subroutine EQLIN(2/4).

```
  60  X=S1/(S1-S4)
      R1=RG(IR)
      Z1=ZG(IZ)-DZ*X
      BP1=(RBP(I1)+X*(RBP(I4)-RBP(I1)))/R1
      I=I-1
      IR=IR-1
      K=2
C..CHECK
  70  IF(IR.LE.  1)GOTO 810
      IF(IR.GE.NRM)GOTO 810
      IF(IZ.LE.  1)GOTO 810
C..LINE INTEGRALS
      VL1=R1*R1
      DS1=1.D0/BP1
      CK1=BP1
      SS1=VL1*BP1
      AA1=DS1/VL1
      RR1=VL1/BP1
      ZBT=RBV(N)/R1
      BV1=BP1+ZBT*ZBT/BP1
C
      BFAV1=SQRT(BP1**2+ZBT**2)/BP1
C
      DL=SQRT((R1-R0)*(R1-R0)+(Z1-Z0)*(Z1-Z0))
      VLV(N)=VLV(N)+(VL0+VL1)*(Z0-Z1)
      SDW(N)=SDW(N)+DL*(DS0+DS1)
      CKV(N)=CKV(N)+DL*(CK0+CK1)
      SSV(N)=SSV(N)+DL*(SS0+SS1)
      AAV(N)=AAV(N)+DL*(AA0+AA1)
      RRV(N)=RRV(N)+DL*(RR0+RR1)
      BBV(N)=BBV(N)+DL*(BV0+BV1)
C
      BFAV(N)=BFAV(N)+DL*(BFAV0+BFAV1)
C..
      IF(N.NE.NV)GOTO 80
      NSU=NSU+1
      RSU(NSU)=R1
      ZSU(NSU)=Z1
      CSU(NSU)=BP1
C..
   80 IF(IZ.LE.NZ)GOTO 20
 100  VLV(N)=ZPI*VLV(N)
      SDW(N)=1.D0/(2.D0*ZPI*SDW(N))
C
      BPV(N) = 2.D0*ZPI*CKV(N)*SDW(N)
      CKV(N)=2.D0*ZPI*CKV(N)/SDW(N)
      SSV(N)=2.D0*ZPI*SSV(N)/SDW(N)
      AAV(N)=2.D0*ZPI*AAV(N)*SDW(N)
      RRV(N)=2.D0*ZPI*RRV(N)*SDW(N)
```

Fig. 6.1 Original version of subroutine EQLIN(3/4).

```
      BBV(N)=2.D0*ZPI*BBV(N)*SDW(N)
C
      BFAV(N)=2.D0*ZPI*BFAV(N)*SDW(N)
  120 CONTINUE
      . . . .
      RETURN
      END
```

Fig. 6.1 Original version of subroutine EQLIN(4/4).

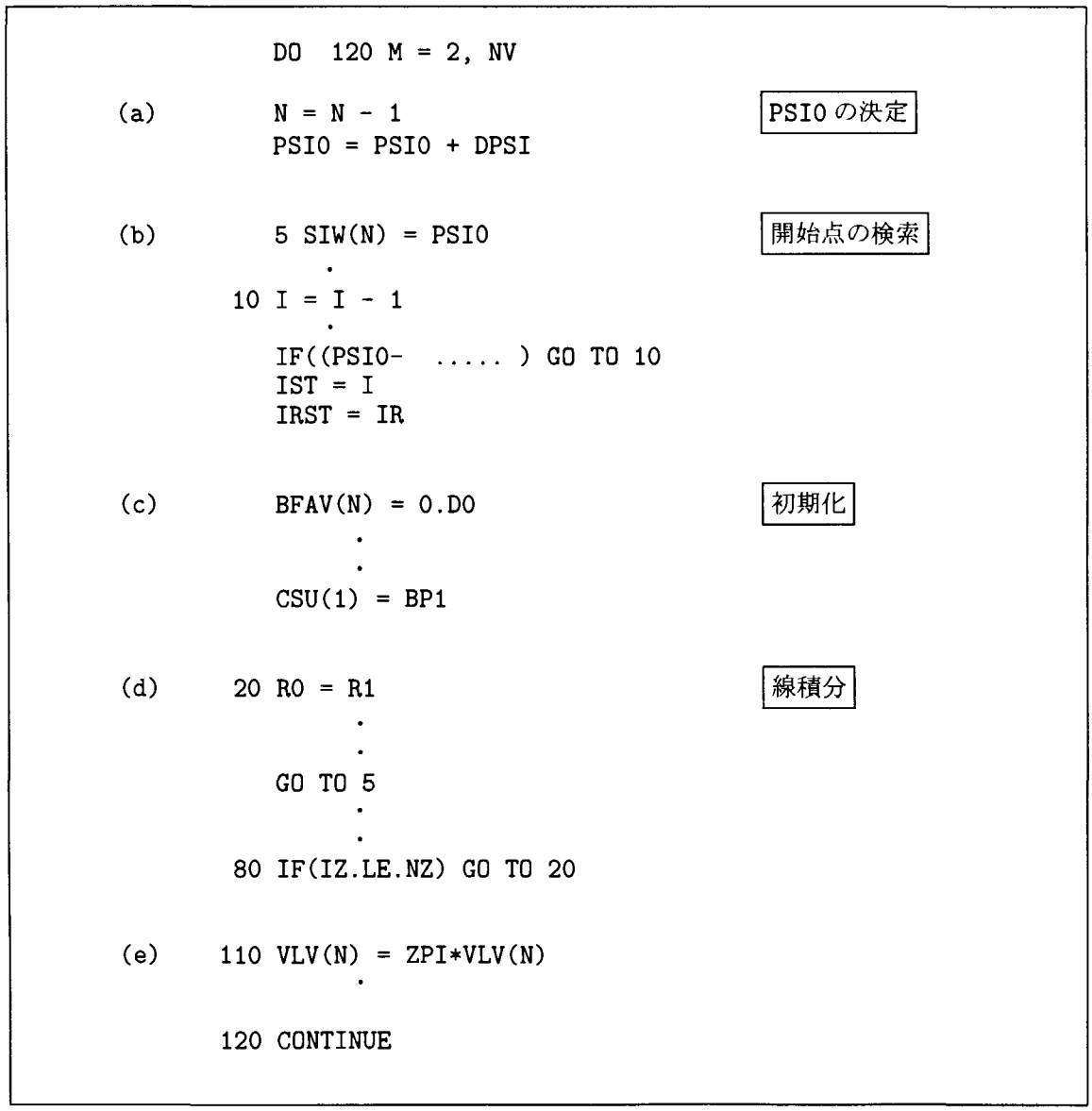

Fig. 6.2 DO loop structure in subroutine EQLIN.

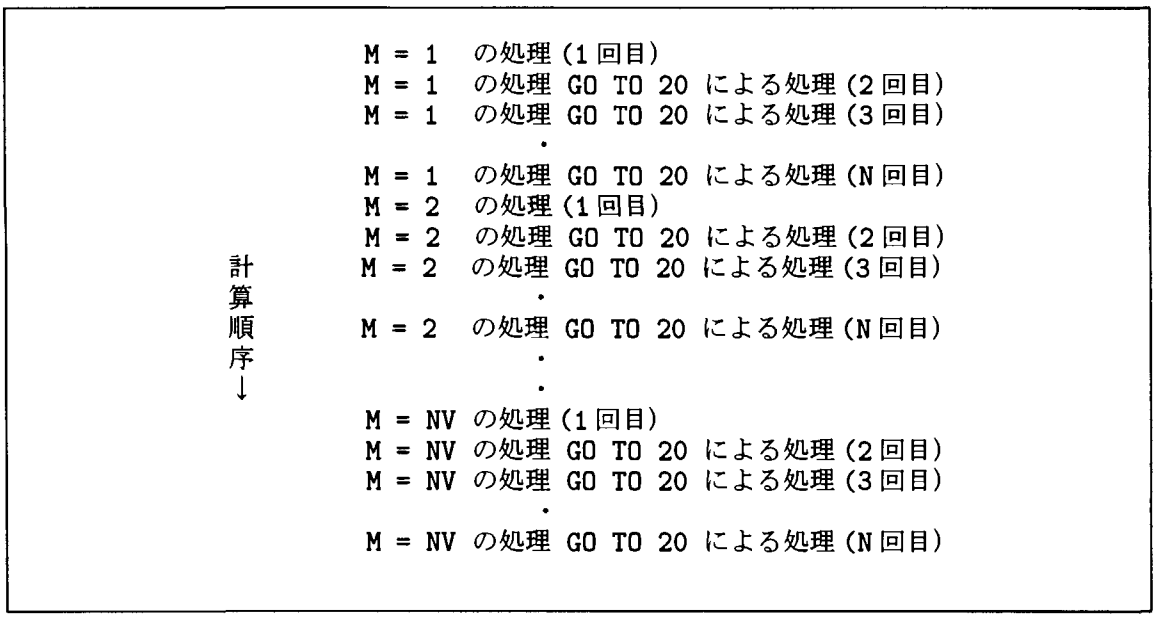

Fig. 6.3 Computation sequence of original version.

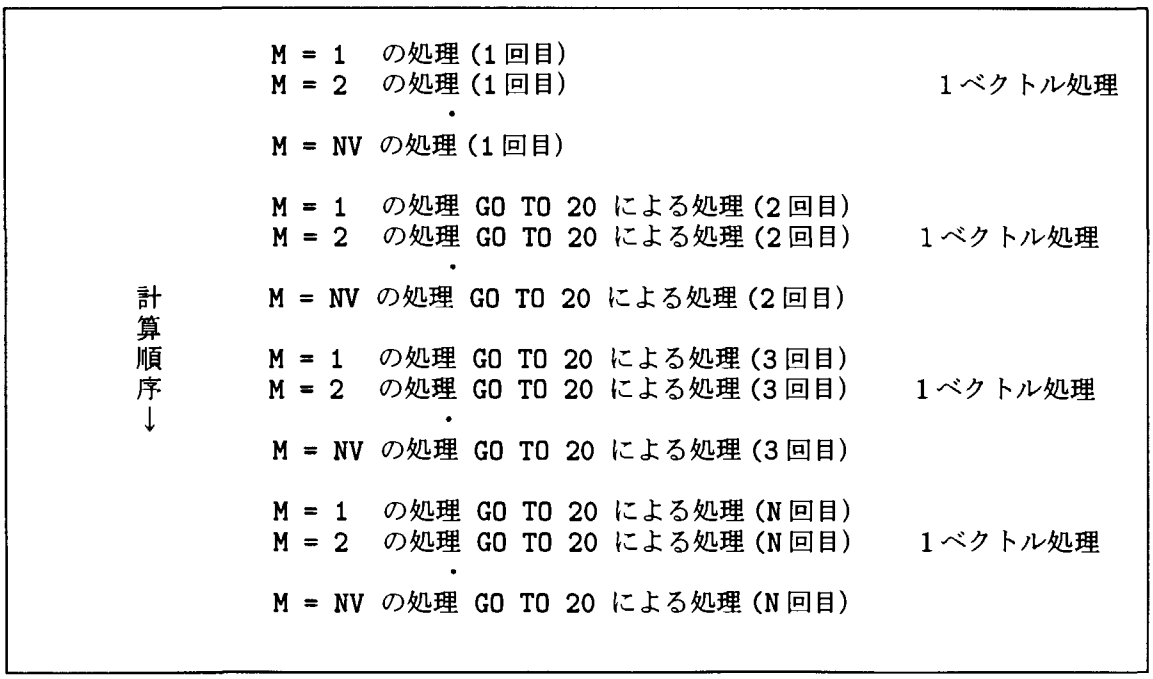

Fig. 6.4 Computation sequence of vector version.

```
      SUBROUTINE EQLIN
      . . . .
CVP---------  ARRAYS FOR VECTOR
      PARAMETER (NN=101)
      DIMENSION ILT(NN),IRLT(NN),IZLT(NN),PSIOL(NN)
      DIMENSION R1LT(NN),Z1LT(NN),BP1LT(NN),VL1LT(NN),DS1LT(NN),
     1          CK1LT(NN),SS1LT(NN),AA1LT(NN),RR1LT(NN),BV1LT(NN),
     2          BFAV1LT(NN),KLT(NN)
      DIMENSION MLT(NN*2)
CVP-----------------------------------------------------------------

v     DO 1210 M = 2, NV
v     PSIOL(M) = DPSI*(M-2)
v 1210 CONTINUE

      DO 1220 M = 2, NV
      N=N-1
CVP5  SIW(N)=PSIO
      SIW(N)=PSIOL(M)
C-----------------------------------------------------------------
C..SEARCH STARTING POINT
      I=IST+1
      IR=IRST+1
      IZ=NZ
   10 I=I-1
      IR=IR-1
      IF(IR.LT.IRAXIS)GOTO 800
CVP   IF((PSIO-PSI(I))*(PSIO-PSI(I+1)).GT.0.D0)GOTO 10
      IF((PSIOL(M)-PSI(I))*(PSIOL(M)-PSI(I+1)).GT.0.D0)GOTO 10
      IST=I
      IRST=IR
      ILT(M) = I
      IRLT(M) = IR
      IZLT(M) = IZ
 1220 CONTINUE

      N=NV+1
v     DO 1230 M = 2, NV
C-----------------------------------------------------------------
C..INITIALIZE LINE INTEGRALS
C
v     N=N-1
v     I = ILT(M)
v     BFAV(N)=0.D0
v     VLV(N)=0.D0
v     SDW(N)=0.D0
v     CKV(N)=0.D0
v     SSV(N)=0.D0
v     AAV(N)=0.D0
```

Fig. 6.5 Vector version of subroutine EQLIN(1/5).

```
v         BBV(N)=0.D0
v         RRV(N)=0.D0
v         X=(PSIOL(M)-PSI(I))/(PSI(I+1)-PSI(I))
v         R1LT(M)=RG(IRLT(M))+X*DR
v         Z1LT(M)=ZG(IZLT(M))
v         BP1LT(M)=(RBP(I)+X*(RBP(I+1)-RBP(I)))/R1LT(M)
v         VL1LT(M)=R1LT(M)*R1LT(M)
v         DS1LT(M)=1.D0/BP1LT(M)
v         CK1LT(M)=BP1LT(M)
v         SS1LT(M)=VL1LT(M)*BP1LT(M)
v         AA1LT(M)=DS1LT(M)/VL1LT(M)
v         RR1LT(M)=VL1LT(M)/BP1LT(M)
v         ZBT=RBV(N)/R1LT(M)
v         BV1LT(M)=BP1LT(M)+ZBT**2/BP1LT(M)
      C
v         BFAV1LT(M)=SQRT(BP1LT(M)**2+ZBT**2)/BP1LT(M)
      C..TRACE CONTURE
v         K=1
v         IF(N.NE.NV)GOTO 1230
v         NSU=1
v         RSU(1)=R1LT(M)
v         ZSU(1)=Z1LT(M)
v         CSU(1)=BP1LT(M)
v    1230 CONTINUE

v         DO 1231 M = 2, NV
v            MLT(M) = 1
v            KLT(M) = 1
v    1231 CONTINUE

     1111 CONTINUE
          MCNT = 0
     *VOCL LOOP,NOVREC
v         DO 1240 M = 2, NV
v           IF(MLT(M).EQ.0) GO TO 1240
v           N = NV - M + 2
v         I = ILT(M)
v         IR = IRLT(M)
v         IZ = IZLT(M)
v         K = KLT(M)
v      20 RO=R1LT(M)
v         ZO=Z1LT(M)
v         BPO=BP1LT(M)
v         VLO=VL1LT(M)
v         DSO=DS1LT(M)
v         CKO=CK1LT(M)
v         SSO=SS1LT(M)
v         AAO=AA1LT(M)
v         RRO=RR1LT(M)
v         BVO=BV1LT(M)
```

Fig. 6.5 Vector version of subroutine EQLIN(2/5).

```
  C
v         BFAV0=BFAV1LT(M)
  C
v         I1=I
v         I2=I1+1
v         I3=I2-NR
v         I4=I1-NR
v         S1=PSI0L(M)-PSI(I1)
v         S2=PSI0L(M)-PSI(I2)
v         S3=PSI0L(M)-PSI(I3)
v         S4=PSI0L(M)-PSI(I4)
v         IF(S1*S2.LT.0.D0.AND.K.NE.1)GOTO 30
v         IF(S2*S3.LT.0.D0.AND.K.NE.2)GOTO 40
v         IF(S3*S4.LT.0.D0.AND.K.NE.3)GOTO 50
v         IF(S4*S1.LT.0.D0.AND.K.NE.4)GOTO 60
v         PSI0=PSI0+1.D-05*SAXIS
    CXXXX GOTO 5
v     30  X=S1/(S1-S2)
v         R1LT(M)=RG(IR)+DR*X
v         Z1LT(M)=ZG(IZ)
v         BP1LT(M)=(RBP(I1)+X*(RBP(I2)-RBP(I1)))/R1LT(M)
v         I=I+NR
v         IZ=IZ+1
v         K=3
v         GOTO 70
v     40  X=S2/(S2-S3)
v         R1LT(M)=RG(IR+1)
v         Z1LT(M)=ZG(IZ)-DZ*X
v         BP1LT(M)=(RBP(I2)+X*(RBP(I3)-RBP(I2)))/R1LT(M)
v         I=I+1
v         IR=IR+1
v         K=4
v         GOTO 70
v     50  X=S4/(S4-S3)
v         R1LT(M)=RG(IR)+DR*X
v         Z1LT(M)=ZG(IZ-1)
v         BP1LT(M)=(RBP(I4)+X*(RBP(I3)-RBP(I4)))/R1LT(M)
v         I=I-NR
v         IZ=IZ-1
v         K=1
v         GOTO 70
v     60  X=S1/(S1-S4)
v         R1LT(M)=RG(IR)
v         Z1LT(M)=ZG(IZ)-DZ*X
v         BP1LT(M)=(RBP(I1)+X*(RBP(I4)-RBP(I1)))/R1LT(M)
v         I=I-1
v         IR=IR-1
v         K=2
    C..CHECK
```

Fig. 6.5 Vector version of subroutine EQLIN(3/5).

```
   c.tn                                T.NEMOTO
   CX70  IF(IR.LE.  1)GOTO 810
   CXXXX IF(IR.GE.NRM)GOTO 810
   CXXXX IF(IZ.LE.  1)GOTO 810
v    70  continue
   C..LINE INTEGRALS
v        VL1LT(M)=R1LT(M)*R1LT(M)
v        DS1LT(M)=1.D0/BP1LT(M)
v        CK1LT(M)=BP1LT(M)
v        SS1LT(M)=VL1LT(M)*BP1LT(M)
v        AA1LT(M)=DS1LT(M)/VL1LT(M)
v        RR1LT(M)=VL1LT(M)/BP1LT(M)
v        ZBT=RBV(N)/R1LT(M)
v        BV1LT(M)=BP1LT(M)+ZBT*ZBT/BP1LT(M)
   C
v        BFAV1LT(M)=SQRT(BP1LT(M)**2+ZBT**2)/BP1LT(M)
   C
v        DL=SQRT((R1LT(M)-R0)*(R1LT(M)-R0)+(Z1LT(M)-Z0)*(Z1LT(M)-Z0))
v        VLV(N)=VLV(N)+(VL0+VL1LT(M))*(Z0-Z1LT(M))
v        SDW(N)=SDW(N)+DL*(DS0+DS1LT(M))
v        CKV(N)=CKV(N)+DL*(CK0+CK1LT(M))
v        SSV(N)=SSV(N)+DL*(SS0+SS1LT(M))
v        AAV(N)=AAV(N)+DL*(AA0+AA1LT(M))
v        RRV(N)=RRV(N)+DL*(RR0+RR1LT(M))
v        BBV(N)=BBV(N)+DL*(BV0+BV1LT(M))
   C
v        BFAV(N)=BFAV(N)+DL*(BFAV0+BFAV1LT(M))
   C..
v        IF(N.NE.NV)GOTO 80
v        NSU=NSU+1
v        RSU(NSU)=R1LT(M)
v        ZSU(NSU)=Z1LT(M)
v        CSU(NSU)=BP1LT(M)
   C..
v     80 IF(IZ.LE.NZ) THEN          ← リスト作成部分
v          MCNT = 1
v          MLT(M) = 1
v          ILT(M) = I
v          IRLT(M) = IR
v          IZLT(M) = IZ
v          KLT(M) = K
v        ELSE
v          MLT(M) = 0
v        END IF
v   1240 CONTINUE
         IF(MCNT.NE.0) THEN
            GO TO 1111
         END IF
         N=NV+1
```

Fig. 6.5 Vector version of subroutine EQLIN(4/5).

```
v         DO 1250 M = 2, NV
v         N = N - 1
v    100  VLV(N)=ZPI*VLV(N)
v         SDW(N)=1.D0/(2.D0*ZPI*SDW(N))
     C
v         BPV(N) = 2.D0*ZPI*CKV(N)*SDW(N)
v         CKV(N)=2.D0*ZPI*CKV(N)/SDW(N)
v         SSV(N)=2.D0*ZPI*SSV(N)/SDW(N)
v         AAV(N)=2.D0*ZPI*AAV(N)*SDW(N)
v         RRV(N)=2.D0*ZPI*RRV(N)*SDW(N)
v         BBV(N)=2.D0*ZPI*BBV(N)*SDW(N)
     C
v         BFAV(N)=2.D0*ZPI*BFAV(N)*SDW(N)
v    1250 CONTINUE
          . . . .
          RETURN
          END
```

Fig. 6.5 Vector version of subroutine EQLIN(5/5).

```
          FUNCTION FLUX(R,Z,R0,Z0,C0,N)
          . . . .
          FLUX=0.D0
     *VOCL LOOP,NOVREC
v         DO 10 M=1,N
v         X(M)=R*R0(M)
v         XX=4.D0*X(M)/((R+R0(M))**2+(Z-Z0(M))**2)
v         I=1.D0-DDTAB*LOG(1.D0-XX)
v           D=(XX-XTAB(I))/(XTAB(I+1)-XTAB(I))
v           WORK(M)=(FTAB(I)+D*(FTAB(I+1)-FTAB(I)))
     CVP  IF(I.GT.NNTAB) WORK(M)=FLUX0(XX)
     *VOCL STMT,IF(0)
v         IF(I.GT.NNTAB) THEN
v          XXV=1.D0-XX
v          XL=DLOG(1.D0/XXV)
v          XK=A0+XXV*(A1+XXV*A2)+(B0+XXV*(B1+XXV*B2))*XL
v          XE=C0V+XXV*(C1+XXV*C2)+    XXV*(D1+XXV*D2) *XL
v          WORK(M)=((1.D0-XX/2.D0)*XK-XE)/(PI*SQRT(XX))
v         END IF
v     10  CONTINUE
v         DO 20 M = 1 ,N
v           FLUX=FLUX-C0(M)*SQRT(X(M))*WORK(M)
v     20  CONTINUE
          RETURN
          END
```

Fig. 6.6 Vector version of subroutine FLUX.

```
      SUBROUTINE FFFT2
      . . . .
      DO 100 I=2,NR-1
         DO 200 J=1,NZ-1
            FAINF1(J,I)= PSI(I,J+1)          (a)
            FAINF1(NZ+J-2,I)=PSI(I,NZ-J+1)   (b)
 200     CONTINUE
 100  CONTINUE
      . . . .
      DO 10 I=1,NR,NR-1
         DO 20 J=1,NZ-1
            FAINF1(J,I)=PSI(I,J+1)-PSI(I,1)
            FAINF1(NZ+J-2,I)=PSI(I,NZ-J+1)-PSI(I,1)
 20      CONTINUE
 10   CONTINUE
C
      CALL SFFT( NR,2*NZF,TABLE ,FAINF1,FAINF3)
C
      ANZ =-1.0D0/DFLOAT(NZF)
      DO 40 I=1,NR
         DO 30 J=1,NZN
            GN(J,I)= ANZ*FAINF1(2*J-1,I)
 30      CONTINUE
 40   CONTINUE
C
      RETURN
      END
```

Fig. 6.7 Original version of subroutine FFFT2.

```
          SUBROUTINE FFFT2

s2        DO 100 I=2,NR-1
v2           DO 200 J=2,NZ-1
v2              FAINF1(J,I)= PSI(I,J+1)          (a)
v2              FAINF1(NZ+J-2,I)=PSI(I,NZ-J+1)   (b)
v2   200    CONTINUE
s2   100  CONTINUE
v         DO 201 I=2,NR-1
v            FAINF1(1,I) = PSI(I,2)
v    201  CONTINUE

s         DO 10 I=1,NR,NR-1
v            DO 20 J=2,NZ-1
v               FAINF1(J,I)=PSI(I,J+1)-PSI(I,1)
v               FAINF1(NZ+J-2,I)=PSI(I,NZ-J+1)-PSI(I,1)
v    20      CONTINUE
s    10   CONTINUE
v         DO 11 I=1,NR,NR-1
v              FAINF1(1,I)=PSI(I,2)-PSI(I,1)
v    11   CONTINUE
     C
          CALL SFFT( NR,2*NZF,TABLE ,FAINF1,FAINF3)
     C
          ANZ =-1.0D0/DFLOAT(NZF)
s2        DO 40 I=1,NR
v2           DO 30 J=1,NZN
v2              GN(J,I)= ANZ*FAINF1(2*J-1,I)
v2   30      CONTINUE
s2   40   CONTINUE
     C
          RETURN
          END
```

Fig. 6.8 Vector version of subroutine FFFT2.

```
C=======================================================================
      NAMELIST/DSK/IREAD,  JREAD,  IWRITE, JWRITE, MCPU
     >            ,ITDSK,  IERATO, JERATO, NPRNT,  KTMAX, IFILE
C=======================================================================
      IFILE = 60
      IREAD =   0
      JREAD =   0
      IWRITE=   0
      JWRITE=   0
      MCPU  = 570
      ITDSK =1000
      IERATO=   0
      JERATO=   0
      NPRNT =   6
      KTMAX =   1
C-----------------------------------------------------------------------
      WRITE(6,*) '   READS NAMELIST &DSK  AND &PLT AT *AAOPT*'
      READ(5, DSK)
      WRITE(6,DSK)
```

Fig. 6.9 Example of statements in subroutine AAOPT before modification.

```
C=======================================================================
CVP   NAMELIST/DSK/IREAD,  JREAD,  IWRITE, JWRITE, MCPU
CVP  >            ,ITDSK,  IERATO, JERATO, NPRNT,  KTMAX, IFILE
C=======================================================================
      IFILE = 60
      IREAD =   0
      JREAD =   0
      IWRITE=   0
      JWRITE=   0
      MCPU  = 570
      ITDSK =1000
      IERATO=   0
      JERATO=   0
      NPRNT =   6
      KTMAX =   1
C-----------------------------------------------------------------------
      WRITE(6,*) '   READS NAMELIST &DSK  AND &PLT AT *AAOPT*'
CVP   READ(5, DSK)
CVP   WRITE(6,DSK)
CVP>>
```

Fig. 6.10 Example of statements in subroutine AAOPT after modification.

```
      BLOCK DATA
      INCLUDE 'AAA'
      INCLUDE 'PAR'
      INCLUDE 'GEO'
      INCLUDE 'EQUIVALENCE'
      INCLUDE 'CTETIP'
      INCLUDE 'VAC'
      INCLUDE 'EQV'
      INCLUDE 'CNT'
      INCLUDE 'BAL'
      INCLUDE 'COMBOP'
      INCLUDE 'COMFMT'
      INCLUDE 'CNEOTB'
      INCLUDE 'NEOSPL'
      INCLUDE 'NEODGN'
      INCLUDE 'CMOLYM'
CVP-------------------------------------------------------------
CVP   NAMELIST READ -> DATA statement       [DSK]
      DATA NPRNT/6/, KTMAX/1/, MCPU/1000/, IERATO/66/
CVP-------------------------------------------------------------
CVP-------------------------------------------------------------
CVP   NAMELIST READ -> DATA statement       [NEWRUN]
      DATA ITMAX/1/, LOHM/.FALSE./, LNEO/.FALSE./,
     1  AION/2.D0/, ZION/1.D0/, BTOL/3.8D0/, CIEXT/ 0.0D0/,
     2  RMAJ/3.42D0/, RPLA/1.00D0/, ELIP/1.2D0/, TRIG/0.20D0/,
     3  RCNT/3.42d0/,
     4  CD/1.0D20, 1.0D19 ,0.5D0, 1.5D0, 4.D0, 0.D0, 0.D0/,
     5  CTE/1.0D4, 1.0D4, 0.5D0/,
     6  CTI/1.0D4, 1.0D4, 0.5D0/,
     7  CSE/1.0D0, 1.0D0, 1.5D0, 0.2D0, 1.0D7, 0.D0/,
     8  ZLOW/6.D0/, ALOW/12.D0/,
     9  CZEFF/1.0D0, 1.0D0, 1.0D0, 1.5D0, 4.0D0/,
     A  NR/65/, NZ/65/, NV/101/, NSUMAX/101/,
     B  JSHAPE/-1/ ,FIXSET/1.0D-2/,
     C  MABIKI/2/, IGUESS/0/, NUMAX/101/,
     D  IEQMAX/20/,EEQMAX/1.0D-3/,ESETUP/1.0D-4/,IBLMAX/50/,
     E  CFCT/1.0D0, 0.D0, 0.90D0, 0.1D0, 20.D0/
CVP    NAMELIST READ -> DATA statement      [COILP]
      DATA NCLNUM/7/, WIV/1.D0/,
     1  (RCOIL(1,I),I=1,7)/1.80D0, 1.80D0, 1.80D0,
     2             2.60D0, 4.00D0, 4.80D0, 4.80D0/
     3  (ZCOIL(1,I),I=1,7)/0.10D0, 0.50D0, 1.00D0,
     4             2.00D0, 2.00D0, 0.90D0, 0.30D0/
CVP    NAMELIST READ -> DATA statement      [FIXP]
      DATA NSFX/-13/, NFREAD/0/,
     1     (WNV(I),I=0,2)/1.0D2, 1.0D2, 1.0D2/
CVP    READ -> DATA statement               [TITLE]
      DATA TITLE/
     1 '   === IN.21.dat ====  JT-60 Ex. 1  TOKAMAK ====='/
CVP-------------------------------------------------------------
      END
```

Fig. 6.11 Block data 'BLKDTA'.

```
#!/bin/csh -f
#@$-eo
#@$-q vppd4
#@$-r BxV
#@$-lPv 4
#@$-C SELENEJ
#@$-lM  50mb
#@$-lT 10:00
cd $QSUB_WORKDIR
setenv fu06bf 1024
setenv fu06 $HOME/wkvfl/SELENEJbf4ke.test.vfl.outlist.nasi
timex -Hn ../parasrc/apm.out  -Wl,-p30
```

Fig. 6.12 Modification of execution Shellscript.

```
*DECK AAMAIN
      PROGRAM SELENEJ
      INCLUDE 'AAA'
      INCLUDE 'CNT'
      INCLUDE 'COMBOP'
      INCLUDE 'NEODGN'
                    .
                    .
                    .
         CALL STGRD
C                       FLUXES BY COIL SYSTEMS
C                       AT THE EDGE OF THE COMPUTATIONAL BOX
         CALL EQBVAC
C                       FLUXES BY COIL SYSTEMS
C                       AT FIXED POINTS
         CALL EQPVAC
C                       PROFILES OF PLASMA & CURRENT and toroidal flow
CVP>>
         WRITE(6,*) ' #######   CPU KEISOKU START   ######'
         call gettod(rtimes)
         CALL CLOCKV(SVU,SCPU,2,2)
CVP<<
         CALL STPLSM
C                       INITIAL PSI (GUESS VALUE)
                    .
                    .
                    .
CC    WRITE( IFTS13 ) ( REAL( RG(IR) ), IR=1,NR )
CC    WRITE( IFTS14 ) ( REAL( ZG(IR) ), IR=1,NZ )
CC    WRITE( IFTS15 ) ( REAL( PSI(I) ), IR = 1, NR * NZ )
CTWATA97/02/04e-----------------------------------------------
C-----
C%        IF(LNEO) CALL NEOCEF
C-------------------------------
CVP>>
         call gettod(rtimee)
         CALL CLOCKV(EVU,ECPU,2,2)
         WRITE(6,*) ' #######   CPU KEISOKU FINISH   ######'
         rtime = rtimee - rtimes
         TVU = EVU - SVU
         TCPU = ECPU - SCPU
         WRITE(6,*) ' ELPAS = ',rtime, ' micro SEC '
         WRITE(6,*) ' CPU = ',TCPU,' micro SEC '
         WRITE(6,*) ' VU  = ',TVU ,' micro SEC '
CVP<<
                    .
                    .
                    .
       END
```

Fig. 6.13 Addition of time measurement statements in Main routine.

```
      SUBROUTINE EQLIN
          .
          .
      DO 1210 M = 2, NV
      PSIOL(M) = DPSI*(M-2)
 1210 CONTINUE

      DO 1220 M = 2, NV
C..SEARCH STARTING POINT

    開始点の検索（コスト0.7%）

 1220 CONTINUE

      N=NV+1
      DO 1230 M = 2, NV
C..INITIALIZE LINE INTEGRALS   ← 並列化対象

       初期化（コスト0.5%）


 1230 CONTINUE

      DO 1231 M = 2, NV
         MLT(M) = 1
         KLT(M) = 1
 1231 CONTINUE

 1111 CONTINUE
      MCNT = 0
*VOCL LOOP,NOVREC
      DO 1240 M = 2, NV          ← 並列化対象

       線積分I(コスト97.2%)


 1240 CONTINUE

      IF(MCNT.NE.0) THEN
         GO TO 1111
      END IF

      N=NV+1
      DO 1250 M = 2, NV          ← 並列化対象

      線積分II(コスト0.2%)

 1250 CONTINUE

      STOP
      END
```

Fig. 6.14 Outline of vector version of subroutine EQLIN.

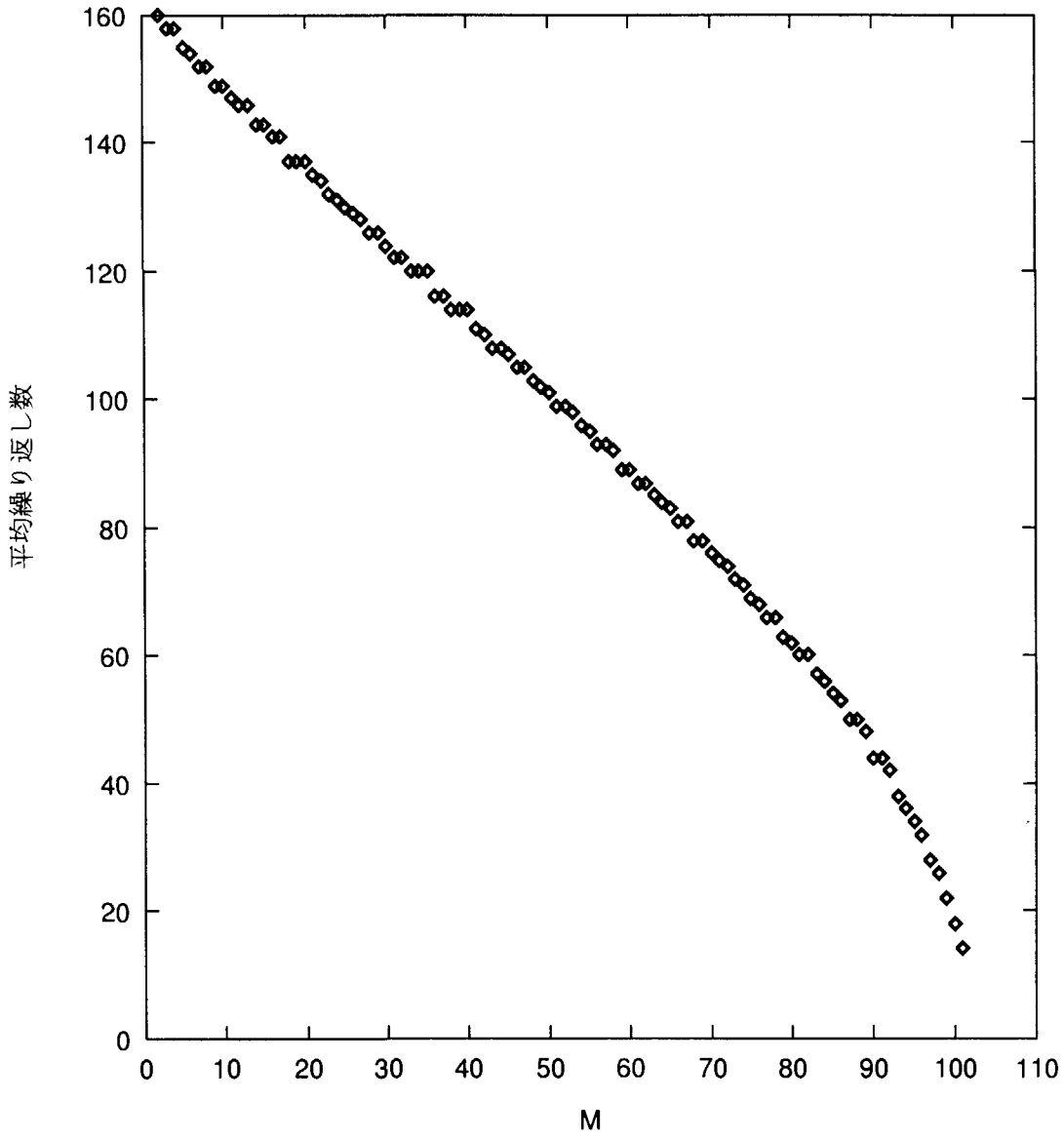

Fig. 6.15 Number of iteration.

```
!XOCL INDEX PARTITION Q=(PROC=P,INDEX=2:101,PART=(15,19,23,43))
      INCLUDE 'AAA'
      INCLUDE 'GEO'
      INCLUDE 'EQUIVALENCE'
      INCLUDE 'EQV'
c.tn>>
      DIMENSION VLVL(2:LN),SDWL(2:LN),BPVL(2:LN),CKVL(2:LN),SSVL(2:LN),
     1          AAVL(2:LN),RRVL(2:LN),BBVL(2:LN),BFAVL(2:LN),
     2          VLVG(2:LN),SDWG(2:LN),BPVG(2:LN),CKVG(2:LN),SSVG(2:LN),
     3          AAVG(2:LN),RRVG(2:LN),BBVG(2:LN),BFAVG(2:LN)
!XOCL LOCAL VLVL(/Q),SDWL(/Q),BPVL(/Q),CKVL(/Q),SSVL(/Q)
!XOCL LOCAL AAVL(/Q),RRVL(/Q),BBVL(/Q),BFAVL(/Q)
!XOCL GLOBAL VLVG,SDWG,BPVG,CKVG,SSVG
!XOCL GLOBAL AAVG,RRVG,BBVG,BFAVG
      EQUIVALENCE (VLVL,VLVG),(SDWL,SDWG),(BPVL,BPVG),(CKVL,CKVG),
     1            (SSVL,SSVG),(AAVL,AAVG),(RRVL,RRVG),(BBVL,BBVG),
     2            (BFAVL,BFAVG)
```

Fig. 6.16 Declaration of divided arrays (local and global arrays).

```
      IF(NZFLG.EQ.0) THEN
!XOCL SPREAD DO
      DO 8140 I = 1, 4
         IF(I.EQ.1) THEN
            ZFLG = .TRUE.
         ELSE
            ZFLG = .FALSE.
         END IF
c.tn>>
         IF(I.EQ.1) THEN
             ISTA = 2          ← 第 1PE での初期値，終値の定義
             IEND = 16
         ELSE IF(I.EQ.2) THEN
             ISTA = 17         ← 第 2PE での初期値，終値の定義
             IEND = 35
         ELSE IF(I.EQ.3) THEN
             ISTA = 36         ← 第 3PE での初期値，終値の定義
             IEND = 58
         ELSE IF(I.EQ.4) THEN
             ISTA = 59         ← 第 4PE での初期値，終値の定義
             IEND = 101
         END IF
c.tn<<
 8140 CONTINUE
!XOCL END SPREAD
      NZFLG = 1
      END IF
```

Fig. 6.17 Definition of initial and terminal parameter of DO for each PE (EQLIN).

```
      SUBROUTINE EQLIN
      PARAMETER (NPE = 4)                    ← PE数の宣言
      PARAMETER (LN = 101,LN2 = LN + 2)
!XOCL PROCESSOR PP(NPE)
!XOCL SUBPROCESSOR P(NPE)=PP(1:NPE)
!XOCL INDEX PARTITION Q=(PROC=P,INDEX=2:101,PART=(15,19,23,43))

      IF(NZFLG.EQ.0) THEN
!XOCL SPREAD DO
      DO 8140 I = 1, 4        ←各PEでのDOループの初期値・終値の決定

 8140 CONTINUE
!XOCL END SPREAD
      NZFLG = 1
      END IF

      DO 1230 M = ISTA,IEND        ← 並列化対象
C..INITIALIZE LINE INTEGRALS
      [初期化]
 1230 CONTINUE

*VOCL LOOP,NOVREC
      DO 1240 M = ISTA,IEND        ← 並列化対象
      [線積分 I]
 1240 CONTINUE

      IF(MCNT.NE.0) THEN
         GO TO 1111
      END IF

!XOCL BARRIER
!XOCL BROADCAST (RSU,ZSU,CSU,NSU) (ZFLG)  ← データ転送

      DO 1250 M = ISTA,IEND        ← 並列化対象
      [線積分 II]
 1250 CONTINUE

!XOCL SPREAD MOVE                  ← データ転送
      DO 8100 M = 2, NV

 8100 CONTINUE
!XOCL END SPREAD (ID1)

C --- SIMPLE LINEAR INTER POLATION

!XOCL MOVEWAIT (ID1)
      RETURN
      END
```

Fig. 6.18 Parallel version of subroutine EQLIN.

```
      IF(IPFLG.EQ.0) THEN
      ICN = NY/NPE
      A = REAL(NY)/REAL(NPE)
      B = REAL(ICN)
      IF(A.GT.B) THEN
        ICN = ICN + 1
      END IF
      MD = NY-(ICN*(NPE-1))

      I = IDVPROC()
      IF(I.EQ.NPE) THEN
         IST = (I-1)*ICN + 1
         IEN = NR
      ELSE
         IST = (I-1)*ICN + 1
         IEN = ist + ICN - 1
      END IF

      IPFLG = 1
      END IF

      IF(IDVPROC().EQ.NPE) IEN = NR
```

Fig. 6.19 Definition of initial and terminal parameter of DO loop in each PE (SFFT).

```
            DO 100 J = 1, NHB-1
               DO 110 I = 1,NR
                  Y(N2-J,I) = X(J,I) + X(NNB-J,I)
                  Y(   J,I) = X(J,I) - X(NNB-J,I)
110            CONTINUE
100         CONTINUE
```

Fig. 6.20 Example of multiplex loop in subroutine SFFT original version.

```
            DO 100 J = 1, NHB-1
               DO 110 I = IST,IEN
                  Y(N2-J,I) = X(J,I) + X(NNB-J,I)
                  Y(   J,I) = X(J,I) - X(NNB-J,I)
110            CONTINUE
100         CONTINUE
```

Fig. 6.21 Example of parallel multiplex loop in subroutine SFFT.

```
C* MOVE
         DO 500 J = 1, NOO
            DO 510 I = 1,NR
               X(J,I) = Y(J,I)
 510        CONTINUE
 500     CONTINUE
```

Fig. 6.22 DO 500 loop of subroutine SFFT original version.

```
C* MOVE
         DO 510 I = IST,IEN
         DO 500 J = 1, NOO
               X(J,I) = Y(J,I)
 500     CONTINUE
 510     CONTINUE
```

Fig. 6.23 DO 500 loop of subroutine SFFT parallel version.

```
CVPP  SUBROUTINE SFFT (NR, N, C, X, Y)
      SUBROUTINE SFFT (NR, N, C, X, WORK2)
C
      IMPLICIT REAL*8 (A-H,O-Z)
CVP   DIMENSION       C(N/4),X(N/2-1,NR),Y(N/2-1,NR)
      DIMENSION       C(N/4),X(N/2-1,NR)
CVPP>>
*INCLUDE PARA1
      PARAMETER (NX=127,NY=65)
!XOCL PROCESSOR PP(NPE)
!XOCL SUBPROCESSOR P(NPE)=PP(1:NPE)
!XOCL INDEX PARTITION Q=(PROC=P,INDEX=1:NY,PART=BAND)
CVPP  DIMENSION Y(NX,NY),X(NX,NY)
      DIMENSION Y(NX,NY)
      DIMENSION YG(NX,NY)
cXOCL LOCAL Y(:,/Q),X(:,/Q)
!XOCL LOCAL Y(:,/Q)
!XOCL GLOBAL YG
EQUIVALENCE (Y,YG)
      DATA IPFLG/0/
                .
                .
```

Fig. 6.24 Declaration of array Y (SFFT).

```
!XOCL SPREAD MOVE
      DO 800 I = 1, NR
         DO 810 J = 1, NX
            X(J,I) = YG(J,I)
  810   CONTINUE
  800 CONTINUE
!XOCL END SPREAD (ID1)
!XOCL MOVEWAIT (ID1)
```

Fig. 6.25 Data transfer in subroutine SFFT.

```
      DO 220 IR=1,NR
      PSI(IR)=FLUX(RG(IR),ZG( 1),RS,ZS,CS,NS)
      PSI(IR)=PSI(IR)+CVAC(0)
      IBL = IBL+1
      DO 210 N=1,NCLNUM
 210  PSI(IR)=PSI(IR)+CVAC(N)*BVAC(IBL,N)
 220  CONTINUE
C-----
      IM=1
      DO 240 IZ=2,NZ
      IM=IM+NR
      PSI(IM)=FLUX(RG( 1),ZG(IZ),RS,ZS,CS,NS)
      PSI(IM)=PSI(IM)+CVAC(0)
      IBL=IBL+1
      DO 230 N=1,NCLNUM
 230  PSI(IM)=PSI(IM)+CVAC(N)*BVAC(IBL,N)
 240  CONTINUE
C-----
      IP=NR
      DO 260 IZ=2,NZ
      IP=IP+NR
      PSI(IP)=FLUX(RG(NR),ZG(IZ),RS,ZS,CS,NS)
      PSI(IP)=PSI(IP)+CVAC(0)
      IBL=IBL+1
      DO 265 N=1,NCLNUM
 265  PSI(IP)=PSI(IP)+CVAC(N)*BVAC(IBL,N)
 260  CONTINUE
```

Fig. 6.26 Call statement of function FLUX in subroutine EQBND original version.

```
      IF(NZFLG.EQ.0) THEN              ← リスト作成
      DO 800 IZ = 2, NZ
         IL2(IZ) = (IZ-1)*NR+1
         IL3(IZ) = IZ*NR
  800 CONTINUE
      NZFLG = 1
      END IF

!XOCL SPREAD NOBARRIER DO/Q
      DO 810 IR = 1,NR
         WPSI(IR)=FLUX(RG(IR),ZG( 1),RS,ZS,CS,NS)
  810 CONTINUE
!XOCL END SPREAD NOBARRIER
!XOCL SPREAD NOBARRIER DO/R
     DO 820 IZ = 2,NZ
         WPSI2(IZ)=FLUX(RG( 1),ZG(IZ),RS,ZS,CS,NS)
         WPSI3(IZ)=FLUX(RG(NR),ZG(IZ),RS,ZS,CS,NS)
  820 CONTINUE
!XOCL END SPREAD NOBARRIER

!XOCL SPREAD MOVE
      DO 830 IR = 1, NR
         PSI(IR) = WPSIG(IR)
  830 CONTINUE
!XOCL END SPREAD (ID1)

!XOCL SPREAD MOVE
      DO 840 IZ = 2, NZ
         PSI(IL2(IZ)) = WPSI2G(IZ)
  840 CONTINUE
!XOCL END SPREAD (ID2)

!XOCL SPREAD MOVE
      DO 850 IZ = 2, NZ
        PSI(IL3(IZ)) = WPSI3G(IZ)
  850 CONTINUE
!XOCL END SPREAD (ID3)
```

Fig. 6.27 Call statement of function FLUX in subroutine EQBND parallel version.

```
*INCLUDE PARA1          ! PARAMETER(NPE=4)
      PARAMETER (NZZ = 65)
      PARAMETER (NY = 65)
!XOCL PROCESSOR PP(NPE)
!XOCL SUBPROCESSOR P(NPE)=PP(1:NPE)
!XOCL INDEX PARTITION Q=(PROC=P,INDEX=1:NY,PART=BAND)
!XOCL INDEX PARTITION R=(PROC=P,INDEX=2:NZZ,PART=BAND)
      DIMENSION WPSI(NY),WPSIG(NY),WPSI2(2:NY),WPSI2G(2:NY)
      DIMENSION WPSI3(2:NY),WPSI3G(2:NY),IL2(2:NY),IL3(2:NY)
!XOCL LOCAL WPSI(/Q),WPSI2(/R),WPSI3(/R)
!XOCL GLOBAL WPSIG,WPSI2G,WPSI3G
      EQUIVALENCE (WPSI,WPSIG),(WPSI2,WPSI2G),(WPSI3,WPSI3G)
      DATA NZFLG/0/
```

Fig. 6.28 Declaration of local arrays.

```
!XOCL SPREAD MOVE
      DO 850 IZ = 2, NZ
        PSI(IL3(IZ)) = WPSI3G(IZ)
  850 CONTINUE
!XOCL END SPREAD (ID3)
CVPP<<
!XOCL MOVEWAIT (ID1)          ←  同期
      IBL = 0
      DO 220 IR=1,NR
CVPP  PSI(IR)=FLUX(RG(IR),ZG( 1),RS,ZS,CS,NS)
      PSI(IR)=PSI(IR)+CVAC(0)
      IBL = IBL+1
      DO 210 N=1,NCLNUM
 210  PSI(IR)=PSI(IR)+CVAC(N)*BVAC(IBL,N)
 220  CONTINUE
C-----
!XOCL MOVEWAIT (ID2)          ←  同期
      IM=1
      DO 240 IZ=2,NZ
      IM=IM+NR
CVPP  PSI(IM)=FLUX(RG( 1),ZG(IZ),RS,ZS,CS,NS)
      PSI(IM)=PSI(IM)+CVAC(0)
      IBL=IBL+1
      DO 230 N=1,NCLNUM
 230  PSI(IM)=PSI(IM)+CVAC(N)*BVAC(IBL,N)
 240  CONTINUE
C-----
!XOCL MOVEWAIT (ID3)          ←  同期
      IP=NR
      DO 260 IZ=2,NZ
      IP=IP+NR
CVPP  PSI(IP)=FLUX(RG(NR),ZG(IZ),RS,ZS,CS,NS)
      PSI(IP)=PSI(IP)+CVAC(0)
      IBL=IBL+1
      DO 265 N=1,NCLNUM
 265  PSI(IP)=PSI(IP)+CVAC(N)*BVAC(IBL,N)
 260  CONTINUE
```

Fig. 6.29 Synchronism for data transfer.

参 考 文 献

[1] TATSUOKI TAKEDA AND SHINJI TOKUDA, "Computation of MHD Equilibrium of Tokamak Plasma", JOURNAL OF COMPUTATIONAL PHYSICS, Vol.93, No.1 March 1991.

[2] UXP/M アナライザ使用手引書 (FORTRAN,VP 用) V10L20 用, 富士通株式会社, 1992 年 2 月.

[3] UXP/M VPP アナライザ使用手引書 V10 用, 富士通株式会社, 1994 年 1 月.

[4] UXP/M VPP VPP FORTRAN77 EX/VPP 使用手引書 V12 用, 富士通株式会社, 1994 年 1 月.

7. 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10年度に12件の作業を完了，平成11年度にも14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，原子力コードを原研が保有する各種スーパーコンピュータ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，汎用トカマク回路シミュレーションプログラムGTCSPを対象に実施したベクトル化作業について，イオン性融体分子動力学計算コードMSP2，渦電流解析コードEDDYCAL，受動的冷却システム試験解析コードTHANPACST2及びMHD平衡コードSELENEJを対象に実施したベクトル並列化作業について記述した．

各コードともベクトル化／並列化によりその高速化効果が顕著に現れている．しかしながら，高速化による計算規模の拡大はメモリ使用量の更なる増加を招き，現状のシステムでは並列化によるメモリ分散の効果だけでは十分に処理できなくなってきている．特にVPP500システムにおいては256MB/PEであり，16並列でも実際にユーザが利用できるのは4GBにも満たない．このため，メモリ不足からスカラ並列機であるAP3000（最大約8GB：4PE×2GB）を利用する場合も少なくない．このような現状を踏まえ，平成11年度にはVPP500システムの増力が実施され，2GB/PE，最大15.2GBの利用が可能となった．今後これらのシステムの利用によりユーザコードのメモリ不足の解消に役立つことと思われる．

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 JT-60 第1実験室 三浦友史氏（2章）、融体燃料プロセス研究室 岡本芳浩氏（3章）、炉構造研究室 武田信和氏（4章）、熱利用システム研究室 高田昌二氏（5章）、プラズマ理論研究室 徳田伸二氏（6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秋元正幸氏，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 EDDYCALの並列化で使用した MPI ライブラリ

・MPI で用いる言葉の説明

コミュニケータ　：通信空間を規定し通信操作の適切なスコープを定めるもの.
グループ　：順序付けされたプロセスの集合.
ランク　：n 個のプロセスから成るグループ内における，各プロセスに 0 ～ (n-1) のプロセスを識別するために割り当てられた番号.

・mpi_init(ierr)

機能　：MPI 環境の初期化処理を行なう.

ierr　：エラーコード

・mpi_comm_size(comm, size, ierr)

機能　：コミュニケータの中のプロセス数を返す.

comm　：コミュニケータ
size　：コミュニケータのグループ内のプロセス数
ierr　：エラーコード

・mpi_comm_rank(comm, rank, ierr)

機能　：呼び出しプロセスのコミュニケータ内のランクを返す.

comm　：コミュニケータ
rank　：コミュニケータのグループ内の呼び出しプロセスのランク
ierr　：エラーコード

・mpi_comm_finalize(ierr)

機能　：MPI 環境の終了処理を行なう.

ierr　：エラーコード

・mpi_barrier(comm, ierr)

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

comm　　：コミュニケータ
ierr　　：エラーコード

・mpi_bcast(buf, count, datatype, root, comm, ierr)

機能　　：ルートのプロセスから，グループ内のすべてのプロセスにデータを転送する．

buf　　：バッファの開始アドレス
count　　：バッファ内の要素数
datatype　　：バッファのデータ型
root　　：ブロードキャストのルートのランク
comm　　：コミュニケータ
ierr　　：エラーコード

・mpi_allreduce(sendbuf, recvbuf, count, datatype, op, comm, ierr)

機能　　：グループ内の各プロセスのデータに対し演算を行ない，結果をグループ内のすべてのプロセスに返す．

sendbuf　　：送信バッファの開始アドレス
recvbuf　　：受信バッファの開始アドレス
count　　：送信バッファ内の要素数
datatype　　：送信バッファ内の要素のデータ型
op　　：演算
comm　　：コミュニケータ
ierr　　：エラーコード

・mpi_allgather(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm, ierr)

機能　　：グループ内のすべてのプロセスから集めたデータをグループ内のすべてのプロセスに転送する．

sendbuf　　：送信バッファのアドレス
sendcount　　：送信バッファ内の要素数
sendtype　　：送信バッファの要素のデータ型
recvbuf　　：受信バッファのアドレス

recvcount	：1つのプロセスから受信する要素数
recvtype	：受信バッファの要素のデータ型
comm	：コミュニケータ
ierr	：エラーコード

付録 B EDDYCALの並列化で使用した並列数値計算ライブラリ（標準固有値問題）

・subroutine PJ_HOUSEH(mytid, mypid, nproc, mpi_group, b, n, n1, np, t0, t1, ierr,
u, pq, w1)

機能	：並列プロセス上に列方向サイクリック分割で配置された実対称行列をHouseholder 変換によって三重対角化する．変換後の三重対角行列はすべてのプロセスにコピーされる．
mytid	：呼び出したプロセスの相対プロセス番号（ランク：0～nproc-1）．input データ．
mypid	：データの収集などを行なうプロセスの相対プロセス番号（ランク：0～nproc-1）． input データ．
nproc	：処理に参加しているプロセス数． input データ．
mpi_group	：コミュニケータ． input データ．
b	：各プロセスにサイクリック分割された実対称行列（b(n1,np)）．input&output データ．
n	：行列の次数． input データ．
n1	：配列 b の第一次元のサイズ． input データ．
np	：配列 b に格納されている列ベクトルの数． input データ．
t0	：三重対角行列の対角要素（t0(n)）． output データ．
t1	：三重対角行列の非対角要素（t1(n)）． output データ．
ierr	：エラーコード（0 で正常終了）． output データ．
u	：作業配列（u(n)）． work データ．
pq	：作業配列（pq(n)）． work データ．
w1	：作業配列（w1(n)）． work データ．

・subroutine PJ_EIGTRI(mytid, mypid, nproc, mpi_group, t0, t1, n, n1, e, ne, jne,
nv, v, nv0, ie, ip, ke, jdg, ierr, r0, r1, r2, f, ipv, w)

機能	：三重対角行列（実対称）の固有値を最大のものまたは最小のものから指定された個数だけ計算する．さらに計算された固有値に対する固有ベクトルを指定した数だけ計算する．
mytid	：呼び出したプロセスの相対プロセス番号（ランク：0～nproc-1）．input データ．

mypid　：データの収集などを行なうプロセスの相対プロセス番号（ランク：0 ～ nproc-1）．input データ．
nproc　：処理に参加しているプロセス数．input データ．
mpi_group　：コミュニケータ．input データ．
t0　：三重対角行列の対角要素（t0(n)）．input データ．
t1　：三重対角行列の非対角要素（t1(n)）．input データ．
n　：行列の次数．input データ．
n1：　配列 v の第一次元のサイズ．input データ．
e　：計算された固有値（e(ne)）．output データ．
ne　：計算する固有値の数（全プロセス合計）．input データ．
jne　：0 のとき最大固有値から計算し，それ以外のときは最小の固有値から計 算する．input データ．
nv　：最初の nv 個の固有値の固有ベクトルを計算する．input データ．
v　：固有ベクトルの計算値（v(n1,nv0)）．output データ．
nv0　：配列 v に格納可能な固有ベクトルの数の上限．input データ．
ie　：各固有値を計算するプロセスの番号（ie(ne)）．output データ．
ip　：各固有値に対する固有ベクトルが配列 v(*,ip(i)) に格納される（ip(ne)）．output データ．
ke　：ie(*) は本ルーチンで決められる（ke=0）．input データ．
jdg　：jdg(j) ≠ 0 のとき j 番目と j-1 番目の固有値は縮退している（jdg(ne)）．output データ．
ierr　：エラーコード（0 で正常終了）．output データ．
r0　：作業配列（r0(n)）．work データ．
r1　：作業配列（r1(n)）．work データ．
r2　：作業配列（r2(n)）．work データ．
f　：作業配列（f(n)）．work データ．
ipv　：作業配列（ipv(n)）．work データ．
w　：作業配列（w(n)）．work データ．

・subroutine PJ_TEVCNV(mytid, mypid, nproc, mpi_group, b, t0, t1, n, n1, np, e, ne, jne, nv, v, nv0, ie, ip, jdg, ierr, w)

機能　：PJ_EIGTRI ルーチンで計算された三重対角行列の固有ベクトルを，PJ_HOUSEH で変換された元の実対称行列の固有ベクトルに変換する．

mutid　：呼び出したプロセスの相対プロセス番号（ランク：0 ～ nproc-1）．input データ．
mypid　：データの収集などを行なうプロセスの相対プロセス番号（ランク：0 ～ nproc-1）．input データ．

nproc	：処理に参加しているプロセス数．input データ．
mpi_group	：コミュニケータ．input データ．
b	：各プロセスにサイクリック分割された実対称行列（b(n1,np)）．input データ．
t0	：三重対角行列の対角要素（t0(n)）．input データ．
t1	：三重対角行列の非対角要素（t1(n)）．input データ．
n	：行列の次数．input データ．
n1	：配列 b の第一次元のサイズ．input データ．
np	：配列 b に格納されている列ベクトルの数．input データ．
e	：計算された固有値（e(ne)）．input データ．
ne	：計算する固有値の数（全プロセス合計）．input データ．
jne	：0 のとき最大固有値から計算し，それ以外のときは最小の固有値から計 算する．input データ．
nv	：最初の nv 個の固有値の固有ベクトルを変換する．input データ．
v	：固有ベクトルの計算値（v(n1,nv0)）．input&output データ．
nv0	：配列 v に格納可能な固有ベクトルの数の上限．input データ．
ie	：各固有ベクトルを持っているプロセスの番号（ie(ne)）．input データ．
ip	：各固有値に対する固有ベクトルが配列 v(*,ip(i)) に格納されている（ip(ne)）．input データ．
jdg	：jdg(j) ≠ 0 のとき j 番目と j-1 番目の固有値は縮退している（jdg(ne)）．input データ．
ierr	：エラーコード（0 で正常終了）．output データ．
w	：作業配列（w(n)）．work データ．

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)

原子力コードの高速化（ベクトル／並列化編）　平成10年度作業報告書