JAERI-Data/Code
99-026

原子力コードのVPP500におけるベクトル化，並列化及び移植（ベクトル化編）

－平成９年度作業報告書－

1999年5月

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

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

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

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

編集兼発行　日本原子力研究所

JAERI－Data/Code 99－026

原子力コードのVPP500におけるベクトル化，並列化及び移植 (ベクトル化編)
－ 平成9年度作業報告書 －

日本原子力研究所計算科学技術推進センター
川崎 信夫※・石附 茂＊・田邊 豪信＊＊・根本 俊行＊
川井 渉＊・小笠原 忍※・足立 将晶※・渡辺 秀雄＊・久米 悦雄

(1999年4月6日受理)

本報告書は，平成9年度に計算科学技術推進センター情報システム管理課で行った原子力コードのVPP500における高速化作業のうち，ベクトル化作業部分について記述したものである．原子力コードのVPP500（一部AP3000）における高速化作業は，平成9年度に14件行われた．これらの作業内容は，今後同種の作業を行う上での参考となりうるよう，作業を大別して，「並列化編」，「ベクトル化編」及び「移植編」の3分冊にまとめた．

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

日本原子力研究所（東海駐在）：〒319-1195 茨城県那珂郡東海村白方白根2-4
※ 外来研究員：富士通株式会社
＊ 富士通株式会社
＊＊ 株式会社ケーシーエス

JAERI－Data/Code 99－026

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

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

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

(Received April 6 , 1999)

Several computer codes in the nuclear field have been vectorized, parallelized and transported on the FUJITSU VPP500 system and/or the AP3000 system at Center for Promotion of Computational Science and Engineering in Japan Atomic Energy Research Institute. We dealt with 14 codes in fiscal 1997. These results are reported in 3 parts, i.e., the vectorization part, the parallelization part and the porting part. In this report, we describe the vectorization.

In this vectorization part, the vectorization of multidimensional two-fluid model code ACE-3D for evaluation of constitutive equations, statistical decay code SD and three-dimensional thermal analysis code for in-core test section (T2) of HENDEL SSPHEAT are described. In the parallelization part, the parallelization of cylindrical direct numerical simulation code CYLDNS44N, worldwide version of system for prediction of environmental emergency dose information code WSPEEDI, extension of quantum molecular dynamics code EQMD and three-dimensional non-steady compressible fluid dynamics code STREAM are described. In the porting part, the porting of transient reactor analysis code TRAC-BF1 and Monte Carlo radiation transport code MCNP4A on the AP3000 are described. In addition, a modification of program libraries for command-driven interactive data analysis plotting program IPLOT is described.

Keywords : ACE-3D, SD, SSPHEAT, Vectorization, VPP500, Nuclear Codes

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

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VPP500（一部AP3000）における高速化作業は，平成9年度に14件行われた．これらの作業内容は，今後同種の作業を行う上での参考となりうるよう，作業を大別して，「並列化編」，「ベクトル化編」及び「移植編」の3分冊にまとめた．

本報告書の「ベクトル化編」では，多次元二流体モデル構成方程式評価用コードACE-3D，原子核統計崩壊計算コードSD及びHENDEL炉内構造物実証試験部（T2）3次元熱伝導解析コードSSPHEATを対象に実施したベクトル化作業について記述している．別冊の「並列化編」では，円筒座標系直接数値解析コードCYLDNS44N，放射能粒子拡散予測コードWSPEEDI，拡張量子分子動力学コードEQMD及び三次元熱流体解析コードSTREAMを対象に実施した並列化作業について記述している．また，別冊の「移植編」では，沸騰水型原子炉熱水力解析コードTRAC-BF1，連続エネルギー粒子輸送モンテカルロコードMCNP4AのAP3000への移植作業，及び汎用図形処理解析システムIPLOT用ライブラリの改良作業について記述している．なお，平成9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，多次元二流体モデル構成方程式評価用コードACE-3Dを対象とした高速化作業について述べる．本作業では，計算コストの高いソルバー部をチューニングの対象としベクトル化作業を実施した．また，ここで用いられているガウス消去法のアルゴリズムを変更し，二段二列同時消去法を採用することで処理の高速化を図ると同時に，消去演算部の並列化の検討を行った．この結果，2次元のシミュレーション計算で約12.8倍の，3次元のシミュレーション計算で約28.7倍の速度向上が得られた．

3章では，原子核統計崩壊計算コードSDを対象とした高速化作業について述べる．当該コードは，既に一部ベクトル化されているものの，計算コストの最も高い部分がベクトル化されていない．本作業では，この部分を中心にDOループの処理の分割や一重化によりベクトル化効率向上のチューニングを施し，ベクトル化作業を行った．この結果，オリジナル版のベクトル実行時に比較して約19.8倍の速度向上が得られた．

4章では，HENDEL炉内構造物実証試験部（T2）3次元熱伝導解析コードSSPHEATを対象とした高速化作業について述べる．当該コードは特に非定常解析において解析効率が悪く，解析時間ステップ数増大への大きな障害となっていた．本作業では，計算コストの集中している3つのルーチンを中心に処理の見直し，指示行挿入による回帰データの回避等によるベクトル化を施した．また，更なる高速化の可能性調査ののため並列化の検討及びAP3000へのインストールも実施した．この結果，VPP500のベクトル実行時において最大で約9.8倍の速度向上が得られた．

なお，本報告書の2章の作業は石附が，3章の作業は田邊が，4章の作業は川崎が担当した．

2. ACE-3D コードのベクトル化

多次元二流体モデル構成方程式評価用コードＡＣＥ－３Ｄ [1] の高速化作業を行なった．ＡＣＥ－３Ｄコードは，受動的安全炉の設計研究を多次元の二流体モデルコードにより高精度に行なうための二流体モデル構成方程式の評価に使用できる解析コードである．基礎式は３次元の二流体モデルであり，解析体系として直行座標系または円筒座標系が選択できる．また，流体としては，空気－水または蒸気－水のパターンが選択できる．メッシュスキームはスタッガードメッシュであり有限差分で定式化し，対流項は風上差分，拡散項は中心差分を用いている．時間進行については，半陰解法で解いている．乱流モデルとして二相 $\kappa-\varepsilon$ 乱流モデルを導入している．本報告書では，ＡＣＥ－３Ｄコードの富士通ＶＰＰ５００ [2,3] 向けベクトル化作業について記述する．

2.1 動的解析

ＶＰＰ５００上のＳＡＭＰＬＥＲ [4] を使用し，ＡＣＥ－３Ｄコードの動的解析を行なった．解析時の使用データとして，２種類の入力データを用いた．１つは２次元のシミュレーションであり，もう一つは３次元のシミュレーションである．２次元と３次元では，ソルバーが変わるため２種類のデータで解析を行なった．２次元の場合は，ソルバーとしてガウスの消去法（ＢＧＬＵ）が使用される．３次元では，双共役勾配法（ＩＬＵＢＣＧ）が使用される．それぞれの解析結果を Fig. 2.1 , Fig. 2.2 に示す．Fig. 2.1 を見ると，約４０％がソルバー部に費やされていることが分かる．Fig. 2.2 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．よって，ソルバー部を対象にチューニングすることとした．

2.2 不完全ＬＵ分解双共役勾配法（ＩＬＵＢＣＧ）

2.2.1 アルゴリズム

ＩＬＵＢＣＧ法のアルゴリズムを以下に示す．

（１）初期ベクトル X_0 を用意する．

（２）係数行列 $\boldsymbol{A}$ の不完全ＬＵ分解を行なう．

（３）$\boldsymbol{r}_0 = \boldsymbol{r}_0^* = \boldsymbol{p}_0 = \boldsymbol{p}_0^* = (LU)^{-1}\,(b - Ax_0)$

（４）k =1,2,..... について，（５）から（１１）までの計算を繰り返す．

（５）$q_k = (LU)^{-1}Ap_k,\quad q_k^* = ((LU)^{-1}A)^T p_k$

（６）$\alpha_k = (r_k, r_k^*)/(q_k, p_k^*)$

（７）$X_{k+1} = X_k + \alpha_k\, p_k$

（８）$r_{k+1} = r_k - \alpha_k\, q_k,\quad r_{k+1}^* = r_k^* - \alpha_k\, q_k^*$

（９）$||r_{k+1}|| < \epsilon\, ||b||$ なら処理を終える．

（１０）$\beta_k = (r_{k+1}, r_{k+1}^*)/(r_k, r_k^*)$

（１１） $p_{k+1} = r_{k+1} + \beta_k\, p_k$，　$p^*_{k+1} = r^*_{k+1} + \beta_k\, p^*_k$

Ｌは下三角行列を，Ｕは上三角行列を表し，$\boldsymbol{A} \fallingdotseq$ＬＵである．$\boldsymbol{r}$ は残差ベクトルを，ｐは探索方向ベクトルを，ｂは定数ベクトルを，Ｘは解ベクトルをそれぞれ表す．α はＸが最小となるよう決めたパラメータであり，β は p_k と p_{k+1} が $\boldsymbol{A}$ に関して共役となるよう決めたパラメータである．q_k はＢＣＧ法における残差ベクトル求解時の $r_{k+1} = r_k - \alpha_k\, Ap_k$ に対し，$r_{k+1} = r_k - \alpha_k\, (LU)^{-1} Ap_k$ とおき，ＬＵ分解を考慮し $(LU)^{-1}Ap_k$ を独立して計算している．

2.2.2　前進代入・後退代入処理

アルゴリズムの（３）と（５）にある前進・後退代入について述べる．基本的な前進・後退代入処理時の配列アクセスは下のようになる．

$$\begin{aligned} a(i,j,k) = a(i,j,k) &- bx(i,j,k) * a(i-1,j,k) \\ &- by(i,j,k) * a(i,j-1,k) \\ &- bz(i,j,k) * a(i,j,k-1) \end{aligned} \tag{2.1}$$

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．前進代入の場合，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. 2.3 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，ベクトル化された前進・後退代入処理を実現できる．よって，以下のようなロジックにより前進・後退代入処理の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ｍｋｉｎｄｘ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．ｍｋｉｎｄｘルーチン終了後，ソルバー（ＩＬＵＢＣＧ）へ処理が移行する．このソルバー内では，ｍｋｉｎｄｘで作成されたリストに従い，不完全ＬＵ分解処理部と前進・後退代入処理部の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．オリジナルソースを Fig. 2.4 に，リストを利用したハイパープレーン法を適用した変更後ソースを Fig. 2.5 に示す．

2.3　ガウス消去法（ＢＧＬＵ）

2.3.1　アルゴリズム

２次元体系の場合は，ソルバーとしてガウス消去法が選択される．ガウス消去法のアルゴリズ

ムを次に示す．また，処理概念図を Fig. 2.6 に示す．

$$-\mathrm{t}^{\mathrm{r}}_{\mathrm{j}} = \mathrm{a}^{\mathrm{r}}_{\mathrm{rj}} = \mathrm{a}^{\mathrm{r-1}}_{\mathrm{rj}} / \mathrm{a}^{\mathrm{r-1}}_{\mathrm{rr}} \tag{2.2}$$

$$\mathrm{a}^{\mathrm{r}}_{\mathrm{ij}} = \mathrm{a}^{\mathrm{r-1}}_{\mathrm{ij}} + \mathrm{t}^{\mathrm{r}}_{\mathrm{j}} * \mathrm{a}^{\mathrm{r-1}}_{\mathrm{ir}} \tag{2.3}$$

i=1 ～ n であり，j=r+1 ～ n+1 である．r はステップ数（段数）を表す．a は係数行列を表し，t は r 段における a の j 列を表す．

2.3.2 二段二列同時消去法

ガウス消去法に対しては，二段二列同時消去法を適応した．二段二列同時消去法とは，消去計算時に２列に対し並行演算する手法である．以下に処理アルゴリズムについて述べる．

第 r 段と第 r ＋１段の二段消去を行なうには，以下に示すようなアルゴリズムとなる．

$$-\mathrm{t}^{\mathrm{r}}_{\mathrm{j}} = a^{r}_{rj} = a^{r-1}_{rj} / a^{r-1}_{rr} \tag{2.4}$$

$$-\mathrm{t}^{\mathrm{r+1}}_{\mathrm{j}} = a^{r+1}_{r+1j} = a^{r}_{r+1j} / a^{r}_{r+1r+1} \tag{2.5}$$

$$\mathrm{a}^{\mathrm{r+1}}_{\mathrm{ij}} = a^{r-1}_{ij} + t^{r}_{j} * a^{r-1}_{ir} + t^{r+1}_{j} * a^{r}_{ir+1} \tag{2.6}$$

さらに，第 j 列と第 j ＋１列についても連続消去を行なうには，以下に示すようなアルゴリズムを上式に追加すればよい．

$$-\mathrm{t}^{\mathrm{r}}_{\mathrm{j+1}} = a^{r}_{rj+1} = a^{r-1}_{rj+1} / a^{r-1}_{rr} \tag{2.7}$$

$$-\mathrm{t}^{\mathrm{r+1}}_{\mathrm{j+1}} = a^{r+1}_{r+1j+1} = a^{r}_{r+1j+1} / a^{r}_{r+1r+1} \tag{2.8}$$

$$\mathrm{a}^{\mathrm{r+1}}_{\mathrm{ij+1}} = a^{r-1}_{ij+1} + t^{r}_{j+1} * a^{r-1}_{ir} + t^{r+1}_{j+1} * a^{r}_{ir+1} \tag{2.9}$$

このようなアルゴリズムに変更することには２つの利点がある．

（1）ＤＯループの回転数を減らすことが出来る．
（2）ベクトル処理中の演算密度を向上させることが出来る．

これらの効果により，計算時間を短縮することが可能となる．処理概念図を Fig. 2.7 に示す．また，オリジナルソースを Fig. 2.8 に，二段二列同時消去法を適用した修正後ソースを Fig. 2.9 に示す．

2.3.3 処理ロジックの変更

二段二列同時消去法適用に伴い，オリジナルロジックを変更した部分がある．オリジナル版では，行を計算する時に列の１番目の要素を求めるようになっていた．今回の修正により，１行と１列を全て求めておき，その後，消去演算を実行するように変更した．これにより，ＤＯループ内で毎回行なわれていた除算をループ外へ出すことができた．つまり，まず一度だけ除算を実行し，その演算結果を変数に代入し，以降のループ内では，その変数を用いて積の形で演算を実行する．このようにすることで，低速な除算処理を回避することができる．一般的に，除算より積算のほうが高速演算可能であるため上記の変更を施した．処理概念図を Fig. 2.10 に示す．

2.3.4 消去演算部の並列化

今後の２次元大規模問題への対応を考慮すれば，並列化による更なる高速化も必要となる．ここでは，計算コストの高い消去演算部に関して並列化の検討を行った．

並列処理用の分割軸は，ｉ方向あるいはｊ方向のどちらでも選択可能である．しかし，ベクトル長の確保を考慮し，ｊ方向を分割軸とした．（Fig. 2.11 を参照）各プロセッサでは重複配列内の担当領域のみを演算する．その時に必要となる初期行の値をデータ転送により各プロセッサへ転送すれば，以降は独立演算が可能である．しかし，各プロセッサに割り当てられる領域は，処理（消去演算の段数）が進むにつれて徐々に小さくなる．そのため，処理の終盤ではデータ転送のオーバーヘッドのため効率が悪くなってしまう．さらに，体系が小さいとデータ転送にかかるコストの方が演算コストよりも大きくなってしまい，並列化のメリットがなくなってしまう．よって，並列版は大規模データの時に有効となる．

オリジナルソースを Fig. 2.12 に，並列化後ソースを Fig. 2.13 に示す．

2.4 性能測定

2.4.1 ＩＬＵＢＣＧ

ＩＬＵＢＣＧルーチンの性能測定結果を Fig. 2.14 に示す．ＩＬＵＢＣＧルーチンではリストによるハイパープレーンアクセスが行なわれている．性能はベクトル長と関連があるため，実データ（３×３×１０）と，ダミーデータ（３０×３０×１０）を使用し性能を測定した．まず，データ体系（３×３×１０）を見ると，修正後の方が性能劣化している．その原因としては，以下の２つの理由が考えられる．

・メモリへのアクセスをリストで行なっている．
（リストを使うとメモリへのアクセスがストライドアクセスとなるためベクトル化されても十分な性能を発揮することが出来ない）
・ベクトル長が短い
（ｘ－ｙ面にプレーンをとり，ｚ方向へ演算が進行するようになっている．そのため当体系の場合のベクトル長は１～９の値をとることになる．この程度のベクトル長では，ベクトル処理に伴うオーバーヘッドが目立ってしまいベクトル演算による効果が現れない）

これら２つの理由により，データ体系が小さい場合は修正後の方が性能が悪くなっている．次に，データ体系（３０×３０×１０）の測定結果を見てみる．このデータになると，オリジナルのスカラに対して修正後のベクトルは１２．８倍の性能がでている．これはベクトル長が長くなり効率的なベクトル演算がなされたためである．より大きなデータを使用すれば，さらに性能が向上すると予測される．

2.4.2 ＢＧＬＵ

ＢＧＬＵルーチンに対しては，二段二列同時消去法を適用した．性能はベクトル長と関連があるため，実データ（１９×１×２４）と，ダミーデータ（３８×１×４８）を使用し性能を測定した．データ体系（１９×１×２４）の場合は，対オリジナルスカラ比は１１．７倍，データ体

系（３８×１×４８）の場合は，対オリジナルスカラ比は２８．７倍という性能をだすことができた．これは以下の３つの理由により性能向上できたと考えられる．

・ループ回数の低減
（最外ループの回転数を半減することができた）
・演算密度の向上
（ループ回転数を減らした分の演算を１本の式に集約し，１式当たりの加算乗算を増やした）
・ロード，ストアの低減
（演算密度の向上により，ロード，ストア回数が減っている．１式で２段分を計算するため，１式で１段の時よりもロード，ストア回数が少なくなる）

これらの効果により性能を向上させることができた．また，より大きな体系のデータを使用すれば，さらに性能が向上すると予測される．

2.5 考察

2.5.1 ＩＬＵＢＣＧ

ハイパープレーン法にはリストを使用したものと，リストを使用せずに連続処理するものがある．リストを使用した手法よりリストを使用しない連続アクセスの方が高性能である．しかし，連続にするには配列へのデータ格納方法を変更する必要がある．そのためには配列を大きく取らなければならない．例えば体系が（ｉ，ｊ，ｋ）の場合，プレーンをｋ方向にとると，変更後の体系として（ｉ，ｊ，ｉ＋ｊ＋ｋ）という大きさが必要となる．つまり，（ｉ＋ｊ＋ｋ）／ｋ 倍のメモリを要することとなる．原研ＶＰＰ５００はメモリ容量が２００ＭＢであり，計算のためのワーク領域を取ることが出来ない．将来，ワーク領域を取っても余る程のメモリが装備された場合は，連続アクセスのハイパープレーン法にすべきであると判断する．

2.5.2 ＢＧＬＵ

不完全ＬＵ分解部の並列化では処理の分割は行なっているが，メモリの分割は実施していない．理由は以下の２点による．

・各タイムステップ毎にデータ転送が発生し，効率が悪い．
・メモリ分割を施すには，ブロックガウス法を適応することになり大幅なアルゴリズムの変更が必要となる．

しかし，今後データが巨大になりメモリ不足に陥るような場合にはメモリ分割が必要となる．メモリ分割手法としては，ブロックガウス法が適応可能である．この手法は，係数行列をブロック単位に分けて各プロセッサに振り分け計算するものである．行列をブロック単位に分割出来るため，プロセッサ数が多くなる程，各プロセッサ上の必要メモリ容量は少なくなる．このような並列化が今後の課題として残されている．

```
Synthesis Information
    Count|    Percent |      VL|  Name
   268604|        39.8|       -|  bglul_
    67927|        10.1|       -|  tf3ds_
    44897|         6.7|       -|  thermo_
    26572|         3.9|       -|  rholi_
    23117|         3.4|       -|  ssl44_
    16860|         2.5|       -|  satprs_
    16673|         2.5|       -|  tf3ds3_
    15908|         2.4|       -|  wf3d_
    14988|         2.2|       -|  ci3d_
    14089|         2.1|       -|  sattmp_
    12575|         1.9|       -|  tf3dxr_
    12043|         1.8|       -|  sfa44_
    10911|         1.6|       -|  tf3dzz_
     8481|         1.3|       -|  kemodl_
     8299|         1.2|       -|  bgslvl_
     8041|         1.2|       -|  stress_
     7613|         1.1|       -|  cibs_
     6806|         1.0|       -|  satder_
     6289|         0.9|       -|  phschk_
     6050|         0.9|       -|  sigma_
     5870|         0.9|       -|  setbd_
     4952|         0.7|       -|  walfri_
     4814|         0.7|       -|  hev_
     4547|         0.7|       -|  intfri_
     4504|         0.7|       -|  zor_
     4471|         0.7|       -|  stdirl_
     4349|         0.6|       -|  timstp_
     4336|         0.6|       -|  flexes_
     4018|         0.6|       -|  vssll_
     3869|         0.6|       -|  hi3d_
     3526|         0.5|       -|  zxor_
     3133|         0.5|       -|  turb_
     3001|         0.4|       -|  timupd_
     2941|         0.4|       -|  hibs_
     2875|         0.4|       -|  fprop_
     2171|         0.3|       -|  zand_
     2029|         0.3|       -|  cpll_
     1818|         0.3|       -|  viscv_
     1765|         0.3|       -|  mix3d_
     1729|         0.3|       -|  thcl_
     1467|         0.2|       -|  intht_
     1340|         0.2|       -|  thcv_
     1271|         0.2|       -|  viscl_
      736|         0.1|       -|  tfsdyt_
      700|         0.1|       -|  sluast_
      445|         0.1|       -|  dvpscl_
      437|         0.1|       -|  bacit_
      398|         0.1|       -|  cpwl_
      195|         0.0|       -|  proof_
      184|         0.0|       -|  prnta3_
      177|         0.0|       -|  setva_
       19|         0.0|       -|  start_
        7|         0.0|       -|  vssl2_
        4|         0.0|       -|  trans_
        3|         0.0|       -|  edward_
        3|         0.0|       -|  edit_
        1|         0.0|       -|  input_
        1|         0.0|       -|  rvssl_
        1|         0.0|       -|  rbc_
        1|         0.0|       -|  vssl3_

   674851|            |      - |  TOTAL

    使用データ : nr = 19 , nth = 1 , nz = 24
```

Fig. 2.1 Dynamic behavior of the original version for data type 1.

Synthesis Count	Information Percent	VL	Name
1052	11.6	-	bglul_
806	8.9	-	tf3ds_
601	6.6	-	ilubcg_
506	5.6	-	thermo_
456	5.0	-	sattmp_
440	4.9	-	rholit_
318	3.5	-	ssl44_
267	3.0	-	tf3ds3_
251	2.8	-	walfri_
247	2.7	-	satprs_
220	2.4	-	wf3d_
190	2.1	-	tf3dzz_
188	2.1	-	htcn3s_
182	2.0	-	sfa44_
175	1.9	-	tf3dyt_
172	1.9	-	cpwl_
163	1.8	-	ci3d_
159	1.8	-	satder_
153	1.7	-	stress_
148	1.6	-	cibs_
144	1.6	-	thcv_
131	1.4	-	tf3dxr_
123	1.4	-	bgslvl_
120	1.3	-	htcor_
119	1.3	-	3igma_
103	1.1	-	zor_
103	1.1	-	hvfilm_
97	1.1	-	deet_
95	1.1	-	phschk_
94	1.0	-	htstrl_
87	1.0	-	setbd_
80	0.9	-	chen_
75	0.8	-	timstp_
66	0.7	-	prnta3_
66	0.7	-	vssll_
63	0.7	-	zxor_
60	0.7	-	zand_
54	0.6	-	fprop_
52	0.6	-	intfri_
48	0.5	-	chfl_
44	0.5	-	flexes_
41	0.5	-	hi3d_
39	0.4	-	hibs_
39	0.4	-	viscv_
38	0.4	-	viscl_
36	0.4	-	stdir2_
34	0.4	-	cpll_
34	0.4	-	timupd_
33	0.4	-	intht_
29	0.3	-	chf_
27	0.3	-	hev_
23	0.3	-	mix3d_
19	0.2	-	dfht_
15	0.2	-	mtmetl_
13	0.1	-	nation_
13	0.1	-	tmsfb_
12	0.1	-	plotf_
12	0.1	-	htstrm_
12	0.1	-	turb_
8	0.1	-	bacit_
8	0.1	-	ciam_
7	0.1	-	hiam_
6	0.1	-	dvpscl_
6	0.1	-	thcl_
4	0.0	-	htstru_
3	0.0	-	edward_
2	0.0	-	vssl2_
1	0.0	-	input_
1	0.0	-	russo_
1	0.0	-	prnta2_
1	0.0	-	start_
1	0.0	-	setva_
1	0.0	-	trans_
1	0.0	-	edit_
9038		-	TOTAL

使用データ : nx = 3 , ny = 3 , nz =10

Fig. 2.2 Dynamic behavior of the original version for data type 2.

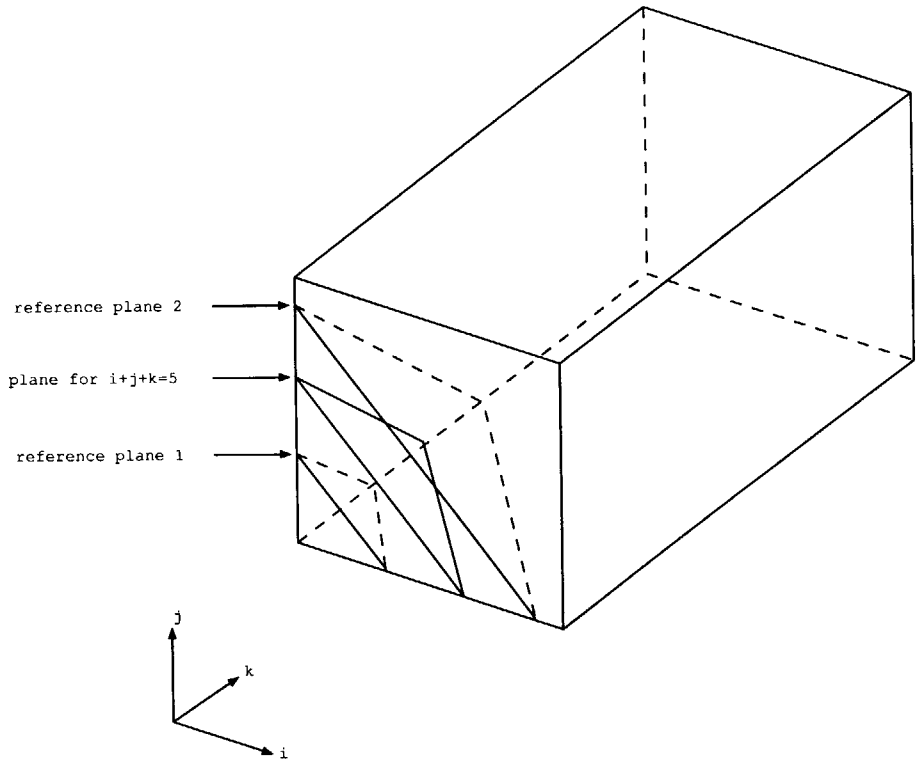

Fig. 2.3 Reference plane of Hyper-Plane.

```
00000227    s3          do 130 i = 1 , n
00000228    s3             w(i,4) = w(i,1)*( w(i,4) - a(i,3)*w(i-1,4)
00000229          .                                 - a(i,2)*w(i-m1,4)
00000230          .                                 - a(i,1)*w(i-m2,4) )
00000231    s3  130     continue
00000232    s3          do 140 i = n , 1 , -1
00000233    s3             w(i,4) = w(i,4)
00000234          .                - w(i,1)*( a(i,5)*w(i+1,4)
00000235          .                         + a(i,6)*w(i+m1,4)
00000236          .                         + a(i,7)*w(i+m2,4) )
00000237    s3  140     continue
```

Fig. 2.4 Forward-backward substitution.

```
00000229      s        do 130 j = 1 , no
00000230         *vocl loop,novrec
00000231      v          do 93 k = ln(j)+1 , ln(j+1)
00000232      v            i = l(k)
00000233      v            w(i,4) = w(i,1)*( w(i,4) - a(i,3)*w(i-1,4)
00000234                  .                          - a(i,2)*w(i-m1,4)
00000235                  .                          - a(i,1)*w(i-m2,4) )
00000236      v    93     continue
00000237      s   130   continue
00000238      s        do 140 j = no , 1 , -1
00000239         *vocl loop,novrec
00000240      v          do 94 k = ln(j)+1 , ln(j+1)
00000241      v             i = l(k)
00000242      v             w(i,4) = w(i,4)
00000243                  .          - w(i,1)*( a(i,5)*w(i+1,4)
00000244                  .                   + a(i,6)*w(i+m1,4)
00000245                  .                   + a(i,7)*w(i+m2,4) )
00000246      v    94     continue
00000247      s   140   continue
```

Fig. 2.5 Hyper-plane method application of F-B substitution.

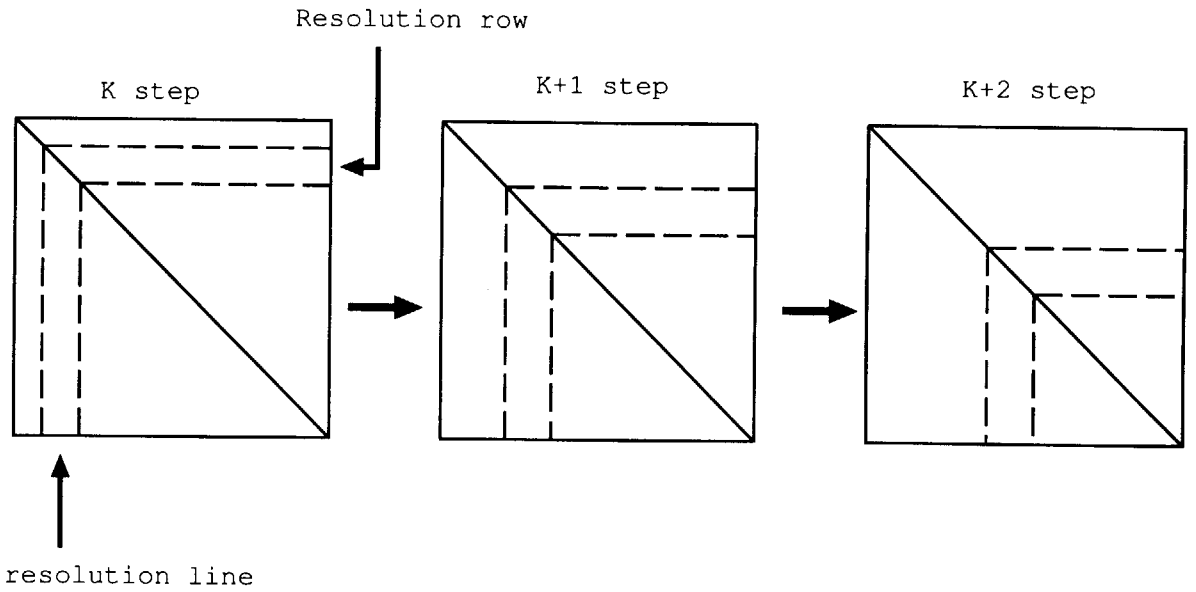

Fig. 2.6 Gaussian elimination.

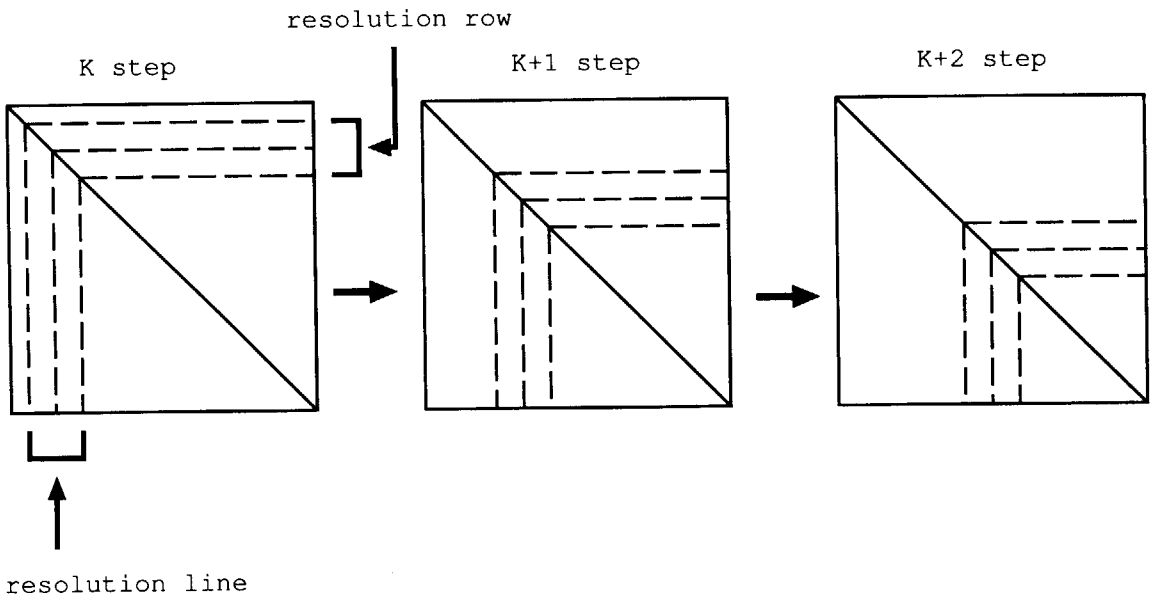

Fig. 2.7 Gaussian elimination for two-steps-two-lines.

```
00000051                ier = 0
00000052                do 1000 k = 1 , n
00000053           c
00000054           c       find maximum element in the k-th column.
00000055           c
00000056                   amax = abs(a(0,k))
00000057                   ipk = k
00000058         v         do 100 i = k+1 , min(k+ml,n)
00000059         v            aik = abs(a(k-i,i))
00000060         v            if ( aik .gt. amax )  then
00000061         v               ipk = i
00000062         v               amax = aik
00000063         v            end if
00000064         v  100    continue
00000065                   ip(k) = ipk
00000066           c
00000067           c       check whether matrix is singular.
00000068           c
00000069                   if ( amax .le. eps ) then
00000070                      ier   = 1
00000071                      ip(k) = k
00000072         v            do 110 i = k+1 , min(k+ml,n)
00000073         v               a(k-i,i) = 0.0d0
00000074         v  110       continue
00000075                      write(*,*) ' *bglu1* matrix is singular'
00000076                      return
00000077                   end if
00000078           c
00000079           c       change pivot
00000080           c
00000081                   if ( ipk .ne. k ) then
00000082         s8           do 120 j = k , min(k+mu+ml,n)
00000083         s8              w            = a(j-ipk,ipk)
00000084         s8              a(j-ipk,ipk) = a(j-k,k)
00000085         s8              a(j-k,k)     = w
00000086         s8  120      continue
00000087                   end if
00000088           c
00000089           c       compute alfa and perform gaussian elimination.
00000090           c
00000091         v         do 130 j = k+1 , min(k+mu+ml,n)
00000092         v            wk(j) = a(j-k,k)
00000093         v  130    continue
00000094         v         do 150 i = k+1 , min(k+ml,n)
00000095         v            a(k-i,i) = -a(k-i,i)/a(0,k)
00000096         v            t        = a(k-i,i)
00000097         v2           do 140 j = k+1 , min(k+mu+ml,n)
00000098         v2              a(j-i,i) = a(j-i,i) + t*wk(j)
00000099         v2 140       continue
00000100         v  150    continue
00000101             1000 continue
00000102                  return
00000103                  end
```

Fig. 2.8 Gaussian elimination.

```
00000058                iwk2 = n+1
00000059                iwt  = 2*n+1
00000060                iwt2 = 3*n+1
00000061                ier = 0
00000062                do 1000 k = 1 , n-2 , 2
00000063           c-----------------------------------------------------
00000064           c
00000065           c        find maximum element in the k-th column.
00000066           c
00000067                   amax = dabs(a(0,k))
00000068                   ipk = k
00000069        v          do 100 i = k+1 , min(k+ml,n)
00000070        v             aik = dabs(a(k-i,i))
00000071        v             if ( aik .gt. amax )  then
00000072        v                ipk = i
00000073        v                amax = aik
00000074        v             end if
00000075        v   100    continue
00000076                   ip(k) = ipk
00000077           c
00000078           c        check whether matrix is singular.
00000079           c
00000080                   if ( amax .le. eps ) then
00000081                      ier   = 1
00000082                      ip(k) = k
00000083        v             do 110 i = k+1 , min(k+ml,n)
00000084        v                a(k-i,i) = 0.0d0
00000085        v   110        continue
00000086                      return
00000087                   end if
00000088           c
00000089           c        change pivot
00000090           c
00000091                   if ( ip(k) .ne. k ) then
00000092           *vocl loop,novrec
00000093        v             do 120 j = k , min(k+mu+ml,n)
00000094        v                w            = a(j-ip(k),ip(k))
00000095        v                a(j-ip(k),ip(k)) = a(j-k,k)
00000096        v                a(j-k,k)     = w
00000097        v   120        continue
00000098                   end if
00000099           c
00000100           c        compute alfa and perform gaussian elimination.
00000101           c
00000102        v          do 130 j = k+1 , min(k+mu+ml,n)
00000103        v             wk(j) = a(j-k,k)
00000104        v   130    continue
```

Fig. 2.9 Gaussian elimination for two-steps-two-lines. 1/4

```
00000105                    t = 1/a(0,k)
00000106      v             do 134 i = k+1 , min(k+ml,n)
00000107      v                a(k-i,i) = -a(k-i,i)*t
00000108      v                if(dabs(a(k-i,i)).gt.1.0e-37) then
00000109      v                  wk(iwt+i) = a(k-i,i)
00000110      v                else
00000111      v                  wk(iwt+i) = 0.0
00000112      v                endif
00000113      v    134       continue
00000114      v              do 135 j = k+1 , min(k+mu+ml,n)
00000115      v                a(j-k-1,k+1)=a(j-k-1,k+1)+wk(iwt+k+1)*wk(j)
00000116      v    135       continue
00000117           c-------------------------------------------------
00000118           c-------------------------------------------------
00000119           c
00000120           c         find maximum element in the k-th column.
00000121           c
00000122                     amax = dabs(a(0,k+1))
00000123                     ipk = k+1
00000124      v              do 102 i = k+1 , min(k+1+ml,n)
00000125      v                aik = dabs(a(k+1-i,i))
00000126      v                if ( aik .gt. amax )  then
00000127      v                   ipk = i
00000128      v                   amax = aik
00000129      v                end if
00000130      v    102       continue
00000131                     ip(k+1) = ipk
00000132           c
00000133           c         check whether matrix is singular.
00000134           c
00000135                     if ( amax .le. eps ) then
00000136                        ier   = 1
00000137                        ip(k+1) = k+1
00000138      v                 do 112 i = k+1 , min(k+1+ml,n)
00000139      v                    a(k+1-i,i) = 0.0d0
00000140      v    112          continue
00000141                        return
00000142                     end if
00000143           c
00000144           c         change pivot
00000145           c
00000146                     if ( ip(k+1) .ne. k+1 ) then
00000147           *vocl loop,novrec
00000148      v                 do 122 j = k+1 , min(k+1+mu+ml,n)
00000149      v                    w            = a(j-ip(k+1),ip(k+1))
00000150      v                    a(j-ip(k+1),ip(k+1)) = a(j-k-1,k+1)
00000151      v                    a(j-k-1,k+1)     = w
00000152      v    122          continue
00000153                     end if
```

Fig. 2.9 Gaussian elimination for two-steps-two-lines. 2/4

```
00000154          c
00000155      v           do 136 i = k+1 , min(k-1+ml,n-1)
00000156      v              a(k-i,i+1)=a(k-i,i+1)+wk(iwt+i+1)*wk(k+1)
00000157      v    136    continue
00000158          c
00000159          c       compute alfa and perform gaussian elimination.
00000160          c
00000161      v           do 132 j = k+1 , min(k+1+mu+ml,n)
00000162      v              wk(iwk2+j) = a(j-k-1,k+1)
00000163      v    132    continue
00000164                  t = 1/a(0,k+1)
00000165      v           do 137 i = k+1 , min(k+1+ml,n)
00000166      v              a(k-i,i+1) = -a(k-i,i+1)*t
00000167      v              if(dabs(a(k-i,i+1)).gt.1.0e-37) then
00000168      v                wk(iwt2+i) = a(k-i,i+1)
00000169      v              else
00000170      v                wk(iwt2+i) = 0.0
00000171      v              endif
00000172      v    137    continue
00000173             c-----------------------------------------------------
00000174      s           do 150 i = k+2 , min(k+ml,n) , 2
00000175      v            do 140 j = k+2 , min(k+mu+ml,n)
00000176      v             a(j-i,i)=a(j-i,i)+wk(iwt+i)*wk(j)
00000177                .                       +wk(iwt2+i-1)*wk(iwk2+j)
00000178      v             a(j-i-1,i+1)=a(j-i-1,i+1)+wk(iwt+i+1)*wk(j)
00000179                .                       +wk(iwt2+i)*wk(iwk2+j)
00000180      v    140      continue
00000181      s    150    continue
00000182                  m = min(k+ml,n)
00000183                  if(mod(m-k,2).eq.1) then
00000184                    if(m.lt.n) then
00000185                      m = m+1
00000186      v               do 142 j = k+2 , min(k+mu+ml,n)
00000187      v                 a(j-m,m)=a(j-m,m)+wk(iwt+m)*wk(j)
00000188                  .                     +wk(iwt2+m-1)*wk(iwk2+j)
00000189      v    142          continue
00000190                    endif
00000191                  endif
00000192          1000 continue
```

Fig. 2.9 Gaussian elimination for two-steps-two-lines. 3/4

```
00000193        c-------------------------------------------------
00000194                if(mod(n,2).eq.0) then
00000195        c
00000196        c       find maximum element in the k-th column.
00000197        c
00000198                amax = abs(a(0,n-1))
00000199                ipk = n-1
00000200                aik = abs(a(-1,n-1))
00000201                if ( aik .gt. amax )  then
00000202                   ipk = n
00000203                   amax = aik
00000204                end if
00000205                ip(k) = ipk
00000206        c
00000207        c       check whether matrix is singular.
00000208        c
00000209                if ( amax .le. eps ) then
00000210                   ier   = 1
00000211                   ip(n-1) = n-1
00000212                   a(-1,n) = 0.0d0
00000213                   write(*,*) ' *bglu1* matrix is singular'
00000214                   return
00000215                end if
00000216        c
00000217        c       change pivot
00000218        c
00000219                if ( ipk .ne. n-1 ) then
00000220                   w            = a(-1,n)
00000221                   a(-1,n) = a(1,n-1)
00000222                   a(1,n-1)     = w
00000223                end if
00000224        c
00000225        c       compute alfa and perform gaussian elimination.
00000226        c
00000227                wk(n)   = a(1,n-1)
00000228                a(-1,n) = -a(-1,n)/a(0,k)
00000229                t       = a(-1,n)
00000230                a(0,n)  = a(0,n) + t*wk(n)
00000231                ip(n) = n
00000232                endif
00000233        c-------------------------------------------------
00000234              return
00000235              end
```

Fig. 2.9 Gaussian elimination for two-steps-two-lines. 4/4

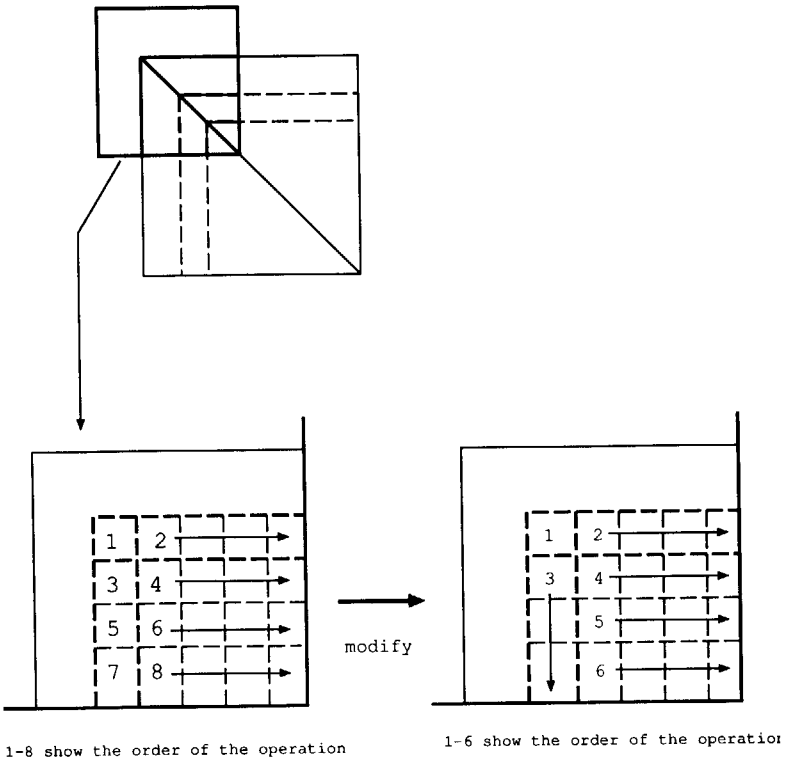

Fig. 2.10 modification of processing.

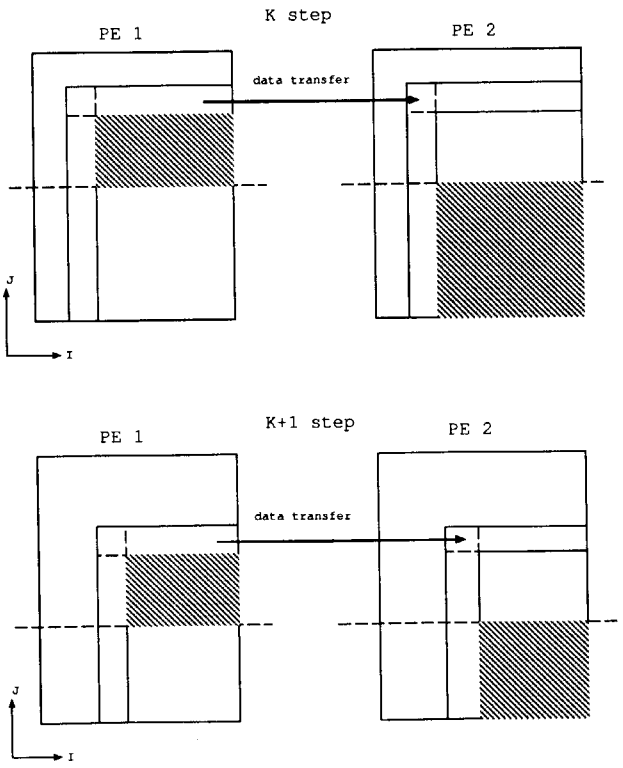

Fig. 2.11 Access region of parallel processing.

```
00000088          c
00000089          c        compute alfa and perform gaussian elimination.
00000090          c
00000091       v            do 130 j = k+1 , min(k+mu+ml,n)
00000092       v               wk(j) = a(j-k,k)
00000093       v    130     continue
00000094       v            do 150 i = k+1 , min(k+ml,n)
00000095       v               a(k-i,i) = -a(k-i,i)/a(0,k)
00000096       v               t        = a(k-i,i)
00000097       v2              do 140 j = k+1 , min(k+mu+ml,n)
00000098       v2                 a(j-i,i) = a(j-i,i) + t*wk(j)
00000099       v2   140        continue
00000100       v    150     continue
00000101            1000 continue
```

Fig. 2.12 Original code befor parallelization.

```
00000041              real*8 a(-mxmax:mxmax+mzmax,mxmax*mzmax),wt(500)
00000042
00000043         !xocl processor pe(2)
00000044         !xocl index partition p=(pe)
00000045         !xocl global a                                          宣言・定義
00000046
00000047              flag = 0
00000048         !xocl parallel region



00000104         c
00000105         c        compute alfa and perform gaussian elimination.
00000106         c
00000107       v           do 130 j = k+1 , min(k+mu+ml,n)
00000108       v             wk(j) = a_l(j-k,k)
00000109       v   130     continue
00000110           !xocl broadcast(a_l(0,k)) (flag)
00000111           !xocl barrier
00000112           !xocl spread do                                       並列処理
00000113       s8          do 135 i = k+1 , min(k+ml,n)
00000114       s8             a_l(k-i,i) = -a_l(k-i,i)/a_l(0,k)
00000115       s8             wt(i) = a_l(k-i,i)
00000116       s8  135     continue
00000117           !xocl end spread
00000118           !xocl spread do                                       並列処理
00000119       s2          do 150 i = k+1 , min(k+ml,n)
00000120       v2             do 140 j = k+1 , min(k+mu+ml,n)
00000121       v2                a_l(j-i,i) = a_l(j-i,i) + wt(i)*wk(j)
00000122       v2  140        continue
00000123       s2  150     continue
00000124           !xocl end spread
00000125           c -----------------------------------------
00000126       s        i = k+1
00000127       s        l = k+2
00000128           !xocl spread do
00000129       s        do 152 m = 1,2
00000130       s           if(idvproc().eq.1) then
00000131           !xocl spread move
00000132                      do 142 j = k+1 , min(k+mu+ml,n)
00000133                         a(j-l,i) = a_l(j-l,i)
00000134            142        continue
00000135           !xocl end spread (xt)
00000136           !xocl movewait (xt)                                   データ転送
00000137       s           endif
00000138           !xocl barrier
00000139       s           if(idvproc().eq.2) then
00000140           !xocl spread move
00000141                      do 143 j = k+1 , min(k+mu+ml,n)
00000142                         a_l(j-l,i) = a(j-l,i)
00000143            143        continue
00000144           !xocl end spread (xt)
00000145           !xocl movewait (xt)
00000146       s           endif
00000147           !xocl barrier
00000148       s    152  continue
00000149           !xocl end spread
```

Fig. 2.13 Gaussian elimination parallelization of BGLU method. 1/2

```
00000150          c ----------------------------------------
00000151      s    1100 continue
00000152          !xocl spread do
00000153      s         do 158 l = 1 , 2
00000154      s         if(idvproc().eq.2) then
00000155          !xocl spread move
00000156                do 157 j = 1 , n
00000157                  do 157 i = -ml , mu+ml
00000158                     a(i,j) = a_l(i,j)
00000159            157 continue
00000160          !xocl end spread (xt)                    マスタＰＥへの転送
00000161          !xocl movewait (xt)
00000162      s         endif
00000163      s     158 continue
00000164          !xocl end spread
00000165          !xocl barrier
00000166                call gettod(ten)
00000167          !xocl end parallel
```

Fig. 2.13 Gaussian elimination parallelization of BGLU method. 2/2

data		3*3*10			30*30*10		
		execute time		ratio for original scalar	execute time		ratio for original scalar
		VU	CPU		VU	CPU	
original	vector	28	504	1.6	1,252	42,325	1.9
	scalar		808	1.0		82,040	1.0
modify	vector	551	741	1.1	6,274	6,414	12.8
	scalar		896	0.9		127,370	0.6

Result of performance measurement for ILUBCG (UNIT: μ sec)

data		19*1*24			38*1*48		
		execute time		ratio for original scalar	execute time		ratio for original scalar
		VU	CPU		VU	CPU	
original	vector	4,888	8,466	6.1	15,985	21,967	17.7
	scalar		51,494	1.0		390,075	1.0
modify	vector	3,469	4,386	11.7	6,274	13,602	28.7
	scalar		34,191	1.5		246,567	1.6

Result of performance measurment for BGLU (UNIT: μ sec)

Fig. 2.14 Result of performance measurement for ILUBCG and BGLU.

参 考 文 献

[1] 大貫晃，加茂英樹，秋本肇；多次元二流体モデル構成方程式 評価用コードＡＣＥ３Ｄの開発，JAERI-Data/Code 96-033，1996 年 11 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.

3. SD コードのベクトル化

3.1 はじめに

原子核統計崩壊計算コード SD [1] は，原子核の核種，励起エネルギー，スピン角運動量をもとに，軽粒子放出及び核分裂による統計崩壊の計算を行うためのコードである．本作業では，富士通 (株) 製分散メモリ型ベクトル並列計算機 VPP500/42(以下， VPP) 向けにベクトル化作業を行った．本コードは，元々 VPP 上でベクトル実行されていたが，最も計算に時間を要する部分の処理がベクトル化されておらず，プログラムの実行に時間がかかっていた．そこで今回は，この計算に時間を要する部分を中心にベクトル化を行い，計算時間の短縮を図ることとなった．

以下，本コードに対して行ったベクトル化作業について報告する．

3.2 コード概要

原子核統計崩壊計算コード SD の計算手順について述べると，まず，崩壊のチャンネルを，放出核種または核分裂生成核， bin に切った崩壊後の励起エネルギー， bin に切った崩壊後のスピン角運動量とにより区別し，それぞれの崩壊チャンネルごとの「部分崩壊幅」を「全崩壊幅」で割った「部分崩壊確率」に従ってモンテカルロ的に崩壊チャンネルを選択する．ここで，「部分崩壊幅」は崩壊前後の状態の「状態密度」の比と，崩壊後の核の運動状態から計算する．また，「全崩壊幅」は「部分崩壊幅」をすべてのチャンネルで和をとる事で得られる．原子核の励起エネルギーが十分小さくなるまでこの崩壊を繰り返すことで計算を行っている．

3.3 事前調査

3.3.1 コスト分布

本コードのどの部分に計算が集中しているかを調べるために，動的挙動解析を行った．動的挙動解析には VPP 上のツールである SAMPLER [2] を使用した． SAMPLER はタイマ割り込みを利用してプログラムの実行状態を調べるツールで，割り込み時にプログラムのどの部分を実行しているかを通知する．従って，割り込み回数の多いサブルーチン程，処理が集中していると判断できる．

Fig. 3.1に SAMPLER を使用して行った動的挙動解析の結果を示す．この結果から，本コードでは，外部手続き `denlev` 及び `calw` の処理に時間を費やしていると判断できる．ここで，Fig. 3.2， Fig. 3.3に， `denlev`， `calw` のソースプログラムを示す． Fig. 3.2から， `denlev` 関数にはループが存在しないことが判る．従って， `denlev` 関数単独ではベクトル実行の対象とならない．一方， Fig. 3.3に示したサブルーチン `calw` では， `do 30` ループが存在しており，既にコンパイラによって自動的にベクトル化が行われている．しかし， SAMPLER の出力 (Fig. 3.1 参照) では，サブルーチン `calw` の平均ベクトル長が 2 と短いことから，ベクトル実行による効果が得られていないことが推測できる．

ここで，VPP のフロントエンドプロセッサである GSP 上で動作する動的挙動解析ツール ANALYZER [3] を使用して，サブルーチン毎のループ単位情報を取得した．サブルーチン calw のループ単位情報を Table 3.1 に示す．この表から，スカラ実行時に比べてベクトル実行時の do 30 ループのコストの方が高くなってしまったことが判る．

そこで，denlev 及び calw の呼び出し元であるサブルーチン tcdec に注目すると，do 20 ループの中で関数 denlev が，do 50 ループの中では 関数 denlev とサブルーチン calw がそれぞれ呼び出されている．Fig. 3.4 にサブルーチン tcdec の do 20 ループを，Fig. 3.5に do 50 ループを示す．特に do 50 ループは tcdec ルーチンで行われる処理の 6 割以上を占めていた．

また，denlev 及び calw は tcdec ルーチン以外では呼び出されていないことから，コンパイルオプション'-Ne' を使用して外部手続きのインライン展開を行った．この時の，SAMPLER の出力結果を Fig. 3.6 に示す．この図から サブルーチン tcdec の処理がプログラム全体の 98.6 % を占めていることが判る．

一方，do 20 ループは，do 50 ループの構造と非常に似通ったものであるため，do 50 ループをベクトル化することができれば，同様にベクトル化することが可能である．従って，本作業においては，先ず，サブルーチン tcdec の do 50 ループを中心にベクトル化作業を進めることとした．

3.3.2 ベクトル化における問題点と方針

サブルーチン tcdec の do 50 ループのベクトル化を考える場合，注目する変数は，ifcalc と ifsave である．この変数の値は，tcdec ルーチンの最初の部分で，変数 ifdec の値によって定義されている (Fig. 3.7参照)．更に，変数 ifdec は，このサブルーチンへの引数である．サブルーチン tcdec は，上位ルーチン casc から呼び出されている．該当部分を Fig. 3.8に示す．この図から，tcdec ルーチンは二回呼び出される可能性があり，変数 ifdec の値は，最初に tcdec が呼ばれる時は 0，二回目に呼ばれる時は 1 となっていることが判る．このことから，最初に tcdec ルーチンが呼ばれた時には，変数 ifcalc の値が 1 となって，ループ内の全ての計算が行われ，変数 tcoeff の値が，配列 tcsave に格納される．この時，値を格納する位置は，変数 ntc の値によって指定される．この変数 ntc の値が配列の大きさ (maxtc) を越えた場合，変数 ifsave の値を 0 にセットして，配列への代入を行わないようになっている．次に，二回目に tcdec ルーチンが呼ばれた場合には，変数 ifcalc の値は 0 となり，ループ内での処理は，前回の呼び出し時に配列 tcsave に格納した値を使用して処理を行うこととなる．この時，変数 ntc の値が配列の大きさを越える場合には，変数 ifcalc の値が 1 にセットされ，tcoeff の値を計算しながら処理が進んでいくようになっている．この時，tcsave または tcoeff の値を変数 tcip0 に足し込みながら，tcip0 の値が一定値を越えた場合にループを飛び出すこととなっている．

また，tcdec ルーチンの引数として使用されている変数 ipdec にも注目しておく必要がある．do 50 ループの外側には，do 40 ループが存在し，更にその外側に do 30 が存在している．tcdec ルーチンが二回目に呼ばれた場合 (ifdec=0 の時) は，この do 30 ループ中で，do 変数 ip が ipdec に等しい時にのみ後続する処理が行われる (Fig. 3.9 参照)．

do 50 ループのベクトル化を行う場合に問題となるのは，ループからの飛び出しが二箇所存在することである．VPP-FORTRAN では，このようなループはベクトル化の対象とならないため，変更が必要となる．しかし，先に述べたように，ループからの飛び出しは，二回目に呼ばれた時にのみ行われるため，変数 ifdec の値に応じて処理を分割し，それぞれ if ブロックの内部にループを構成することで，この問題を解決することが可能である．この様子を Fig. 3.10 に簡潔に示す．

しかし，この状態で新たにベクトル化の妨げとなる問題が発生する．それは，二回目に呼ばれた時 (ifdec=0 の場合) の処理で，ループからの飛び出しが存在することになるが，このようなループ中で変数への値の代入が存在する場合，このループはベクトル化できない．この部分については，Fig. 3.11 に示すような処理にプログラムを変換することでベクトル化が可能であるが，余分なループを作成するため，計算コストが増加することになってしまう．実際には，そのままの形でループをスカラ実行させた場合の方が計算が早く終了することから，このようなループについてはベクトル化を行わないこととした．

また，一回目に呼ばれた時 (ifdec=1 の場合) の処理についても，変数 ntc の参照が定義の前に存在するとの理由でベクトル化されなかった (Fig. 3.12 参照)．これは，ntc の 更新と条件文での参照が回帰計算となってしまうためである．従って，ntc の値が maxtc を越えたかどうかの判定はループの外で行う必要がある．これは，do 50 ループの初期値と終端値から ntc の増分を先に求め，その値が maxtc を越えているかどうか判定し，それからループの実行を行うようにすれば良い．

これまでに述べてきたことから，tcdec ルーチンをベクトル化する場合，do 50 ループは Fig. 3.13 に示すような構造とする．なお，do 50 ループの外側に位置する do 40 ループも if ブロックの内部に置くことにする．これは，do 40 ループと do 50 ループを一重化することを考慮してのことである．これについては後述する．

3.4 ベクトル化

ここでは，実際に本コードに対して行ったベクトル化作業について述べる．前節で述べた方針に従って，サブルーチン tcdec の do 50 ループをベクトル化し，その後，同サブルーチン中の他のループについてもベクトル化を行った．また，サブルーチン calw をインライン展開したが，do 30 ループの平均ベクトル長が 2 と短く，スカラ実行時よりも計算コストが大きくなっていたことから，このループを展開し，プログラム中に書き下すことで，ベクトル化の効果が得られるように修正を行った．更に，do 50 ループについては，ベクトル長を長くしてベクトル化による効果を増加させるため，do 40 ループと do 50 ループを一重化し，ループの回転を ntc の増分によって行うこととした．これらの作業について順を追って報告する．

3.4.1 do 40, do 50 ループ

Fig. 3.13 に示した構造で do 50 ループのベクトル化を行う．この時，ループの一重化を考慮して do 40 ループも含めてプログラムの変換を行うこととする．

・ ipdec=1 の場合

まず，一回目に tcdec ルーチンが呼ばれた時の処理について述べる．ここで行う処理は，

1．先に，変数 ntc の増分を求め，

2．現在の ntc の値に求めた増分値を加え maxtc を越えるかどうか判定し，

3．tcoeff の値を求めて， ntc の値が maxtc を越えていなければ，配列 tcsave に格納する．

となる．

Fig. 3.14 に変数 ntc の増分を求めるために作成した処理を示す．ここで，作業配列 itmp1, itmp2 はそれぞれ do 50 ループの初期値と終端値を格納しておくために使用している．また，配列 iesv は， if 文の判定結果を示すフラグとして，後述する tcoeff を求めるためのループで使用する．なお， ntc の増分値は変数 ntctmp に求められる．

次に， ntc の値が maxtc を越えるかどうかの判定を行う処理を Fig. 3.15 に示す．ここで，maxtc の値を越えた場合に代入される変数 ip は，二回目に tcdec ルーチンが呼ばれた場合に，配列 tcsave 内にデータが存在するかどうかを判定するために使用する．すなわち，変数 ng の値が，負である場合には， ntc の値が maxtc を越えなかったことを表し，すべてのデータが tcsave 内に存在することとなる．それ以外の場合は， maxtc を越えた時の ip 以降の計算で求めた値は tcsave 内に存在しないことを意味する．

最後に， tcoeff を求めるための計算部分を Fig. 3.16に示す．ここで呼び出される外部関数はインライン展開されている．また，配列 tcsave に値を代入する処理は変数 ng が負の値となっている時にのみ行われる．

・ipdec=0 の場合

二回目の tcdec ルーチンの呼び出し時に行う処理を Fig. 3.17に示す．ここでは，変数 ng の値が ip 以下の場合には，配列 tcsave 内にデータが存在するため，これを利用し，それ以外の場合には，新たに計算を行うようにプログラムを記述した．しかし，このループには，変数 tcip0 への定義とループからの飛び出しが存在するため，ベクトル化することはできない．そのことをコンパイラに明示するために，最適化制御行 *vocl loop,scalar を ループの直前に記述した．

3.4.2 do 11 ループ, do 21 ループ

do 11 ループを Fig. 3.18 に示す．このループでは，変数 ee1 が回帰参照となっているためベクトル化されていない．そこで， Fig. 3.19 に示すようにループを分割し，回帰参照を回避することでベクトル化を行った．また， do 21 ループも同一の構造であったため，同じ変更を行った．

3.4.3 do 10, do 20 ループ

do 10, do 20 ループを Fig. 3.20 に示す．このループは，先に述べた do 40, do 50 ループと似通った構造をしているが，変数 ntc の値が maxtc を越えたかどうかの判定を行っていな

い．ここでは，やはり ifdec の値の判定をループの外で行うように変更することでベクトル化を行った．変更した do 10，do 20 ループを Fig. 3.21 に示す．

3.4.4 サブルーチン calw

サブルーチン calw については既に図示しているが，do 50 ループ内での呼び出し部分を含めて，再び Fig. 3.22 に示す．サブルーチン calw 内の do 30 ループの初期値，終端値は引数 j2，j3 によって決定される．そこで，呼び出し側からこれらの引数を見ると，それぞれ 2*j2+mod(m2,2) 及び 2*j3+mod(m3,2) となっている．ここで j2 はループの do 変数であり，j3，m2 及び m3 は do 30 ループ内で定義されている (Fig. 3.23参照)．しかし，m2，m3 については，どのような値であっても，2 で割った余りが使用されるため，mod( ) の結果は 0 か 1 である．j3 の値に注目すると，配列 jip は，メインルーチン内で Fig. 3.24 に示すように定義されている．従って，jip が取り得る値も 0 か 1 である．また，サブルーチン calw が呼ばれるのは変数 ifj が 1 の時であるため，j3 の値も 0 か 1 となる．

これらのことから，サブルーチン calw 内での j2，j3 の値は，0 または 2 となることが判る．従って，do 30 ループの初期値と終端値との差は，0，2，4 の いずれかである．また，ループの増分値が 2 であることから，このループの回転数は，1，2，3 となる．そこで，このループ内の処理を Fig. 3.25 に示すように記述し，ループを使用しないようにした．

3.4.5 ループ構造の変更

3.4.5.1 do 40，do 50 ループ

これまでの変更によって，最も計算コストの高かった tcdec ルーチンの do 50 ループをベクトル実行させることが可能となった．ここで，その外側に存在する do 40 ループと合わせて，ループを一重化することができれば，ベクトル長が長くなるため，ベクトル化による効果を更に向上させることができる．そこで，変数 ntc に注目すると，do 40，do 50 ループの回転数は，ntc の増分値に等しいことが容易に判断できる．このことは，ループを ntc の増分値で回転させることで簡単にループの一重化を行えることを意味している．

この場合，do 変数がこれまでの ie2 や j2 から別の変数に変更されるため，ループ内の演算で必要となるこれらの変数を作業配列を使用して，ntc に対応する値を保存しておくことが必要となる．その部分を Fig. 3.26 に示す．ここで保存しておくのは，ie2，j2 及び lmax とした．

しかし，このように変更したことで，ntc の値が maxtc を超えた場合には，他のデータ領域を破壊してしまう可能性があるため，先に導入した変数 ng の値が負である場合にのみループを ntc の増分値で回転させることとし，それ以外の場合には，前に示した Fig. 3.16 と同じループ構造とした．この部分を Fig. 3.27 に示す．以前のものと異なるのは，ntc の値が maxtc を超えていることが判っているため，配列 tcsave に値を格納していないことだけである．

Fig. 3.28 に ntc の増分値を使用するように変更したループを示す．ここで，変数 ntcold に元の ntc 値を保存しておき，これに 1 を加えた値をループの初期値としている．また，ループの終端値には do 44，do 50 によって更新された ntc 値を用いることで，このループが ntc の増分値で回転することになる．

ループの構造をこのように変更したことによって，二回目にtcdec ルーチンが呼び出されたとき (ifdec=0 のとき) の処理も ntc の増分値によってループを回転させることができる．この部分を Fig. 3.29 に示す．ここで，配列 ntcoffset は，do 30 ループの中で Fig. 3.30 に示すように，それまでに更新された ntc の値を保存しておくために使用されているものである．したがって，これをループの初期値と終端値として利用することができる．しかし，do 30 ループの中で定義されるのは，do 変数 ip が ipmax のときの ntc の初期値だけであり，この時の処理によって更新された ntc の値は保存されない．このため，ifdec=1 のときの処理の最後に Fig. 3.31 に示す処理を追加した．これによって 図示した do 441 ループが正しく実行できる．

また，このループは ntc の値が maxtc を超えないときにのみ実行可能であるため，それ以外の場合の処理が必要となる．この部分を Fig. 3.32 に示す．

3.4.5.2 do 10，do 20 ループ

Fig. 3.33 に一重化した do 10，do 20 ループを示す．ここでは，それぞれのループの回転数が予め判っているため，これを変数 max10 に求めておき，ループの終端値とする．また，従来の do 変数 ie2 及び j2 は，ifdec=0 のときに使用するため，作業配列に保存しておく．Fig. 3.34 に ifdec=0 のときの do 101 及び do211 ループを一重化したものを示す．

3.4.6 mastbl ルーチン

Fig. 3.35 にサブルーチン mastbl の後半に存在する三重ループを示す．このループは，変数 qmin に 変数 qval の最小値を求めるのと，また，そのときの do 変数 j を配列 idec に求めるためのものである．最小値を求める場合，ベクトル実行時には内部で特殊な処理が行われる．この時，ベクトル化の対象となる最内ループの do 変数が配列の添字として使用されている場合，配列への代入処理もベクトル化されるが，ここでは，その外側に存在するループの do 変数 i が使用されているため，ベクトル化されていない．そこで，Fig. 3.36に示すようにプログラムを変更することで，ベクトル化を行った．

3.5 作業結果

Table 3.2 にオリジナル版とベクトル化版の計算時間を示す．ここで，ベクトル化版－1は，ループの一重化を行っていないもの，ベクトル化版－2はループの一重化を行ったものである．オリジナル版のベクトル実行時に比べて，ベクトル化版－1のベクトル実行で約 6.1 倍，ベクトル化版－2のベクトル実行で約 19.8 倍の速度向上が得られた．

また，Fig. 3.37にベクトル化版－1をベクトル実行した場合の SAMPLER の出力結果を，Fig. 3.38 にベクトル化版－2をベクトル実行した場合の SAMPLER の出力結果を示す．ここから，tcdec ルーチンの平均ベクトル長が，ベクトル化版－1では 19 であるのに対し，ベクトル化版－2では 323 と長くなっていることが判る．また，割り込み回数もベクトル化版－2の方がはるかに少ないことが判る．これと先に示した表から，ベクトル化版－1とベクトル化版－2を比較した場合，ベクトル長を長くしたことによる効果が大きいことが判る．

なお．計算結果については，出力ファイルをオリジナル版のものと，それぞれのベクトル化版のものを比較し，完全に一致することを確認した．

3.6 まとめ

今回の作業では，変数 ifdec の値に応じてループを分割することでベクトル化を行った．しかし，ifdec=0 の場合には，仕様によりベクトル化が行えないため，スカラ実行のままとなっている．参考までに，ifdec=1 と ifdec=0 の場合の計算時間の割合を述べると，ifdec=0 の部分の処理は，オリジナル版のベクトル実行時で，ifdec=1 に対して 1% 程度であったため，スカラ実行による影響は特に無いと判断した．

それよりも，ntc の値が $\hat{\text{m}}$axtc を越えた場合，ifdec=0 の処理での再計算処理を行った時の方が，実行時間に影響を与えるものと考えられる．従って，プログラムのコンパイル時に，配列 tcsave の領域をできるだけ大きく確保しておく (maxtc の値を大きくする) ことが望ましい．ntc の値が $\hat{\text{m}}$axtc を越えなければ，ベクトル化による効果が十分に得られると考える．

Table 3.1 Do loops information in subroutine calw by usiong ANALYZER.

do-id	kind	v	ex-count	v-cost	%	s-cost	%	leng	rate	v-effect
30	do	V	208997	16820377	65.2	14577660	61.9	2	100.0	0.7-1.3
-----	total		208997	8986871	34.8	8986871	38.1	1	0.0	1.0-1.0

Table 3.2 Comparision of execution.

	オリジナル版	ベクトル化版－1	ベクトル化版－2
スカラ実行	611.82s	553.30s	605.32s
VU時間（スカラ実行）	0.08s	0.08s	0.08s
ベクトル実行	571.75s	93.46s	28.78s
VU時間（ベクトル実行）	65.74s	68.44s	21.22s
ベクトル化率	17.3%	95.5%	98.8%

```
Synthesis Information
    Count|   Percent|    VL| Name
    54785|      74.5|     -|  denlev_
    12120|      16.5|     2|  calw_
     5260|       7.2|    27|  tcdec_
      885|       1.2|   321|  mastbl_
      235|       0.3|     -|  casc_
      162|       0.2|     -|  eb_
       50|       0.1|     -|  massdis_
       25|       0.0|     1|  ranu2_
       10|       0.0|     -|  MAIN__
        4|       0.0|     -|  rnd_

    73536|          |     4|  TOTAL
```

Fig. 3.1 Dynamic behavior of Original Code.

```
      FUNCTION DENLEV(IZ,IN,E,DE,J)
c**********  Default ****
      DENL=0
      if(E.lt.EMIN.and.J.eq.0) DENL=1./DE
      M=IZ+IN
      AMAS=M*1.0
      if(M.gt.4) then
c**********  A > 4 ****
        IF((M/2)*2.NE.M) THEN
          DELTA=-0.7D0
        ELSEIF((IZ/2)*2.NE.IZ) THEN
          DELTA=-2.0D0
        ELSE
          DELTA=0.7D0
        END IF
        A=AMAS/8.0D0
        R=4*(AMAS**(5./3.))*RMASS*(1.2**2)/5.0/HC/HC/A
        U1=E-(J+MOD(M,2)/2.0D0)**2/A/R-DELTA
        U2=E-(J+MOD(M,2)/2.0D0+1)**2/A/R-DELTA
        IF(U2.gt.0) then
          T1=(1.5D0+SQRT(1.5D0*1.5D0+4*A*U1))/2.0D0/A
          T2=(1.5D0+SQRT(1.5D0*1.5D0+4*A*U2))/2.0D0/A
          DENL=1.0D0/12.0D0/SQRT(R)/A/A*
     &       (EXP(2.0D0*SQRT(A*U1))/T1/T1/T1-
     &        EXP(2.0D0*SQRT(A*U2))/T2/T2/T2)
        endif
        IF(LDFORM.eq.1) DENL=exp(2*sqrt(A*E))/1000
      endif
999   CONTINUE
      DENLEV=DENL
      END
```

Fig. 3.2 Function denlev.

```
      SUBROUTINE CALW(J1,J2,J3,LMAX,W)
      IMPLICIT REAL*8(A-H,O-Z)
      W=0D0
      IF(MOD(J1+J2+J3,2).EQ.1)GOTO 15
      JJ1=ABS(J2-J3)
      JJ2=J2+J3
      DO 30 JJ=JJ1,JJ2,2
        L1=ABS(J1-JJ)/2
        L2=MIN((J1+JJ)/2,LMAX)
        W=W+DIM(L2+1-L1,0)
30    CONTINUE
15    CONTINUE
      END
```

Fig. 3.3 Subroutine calw.

```
          DO 20 J2=IFJ*MAX(0,J1-2),IFJ*(J1+2)
            NTC=NTC+1
            IF(IFDEC.eq.0)THEN
              DL=DENLEV(IZ1,IN1,E2,DE2,J2)
              TCOEFF=0.D0
              IF((ABS(J2-J1).LE.2).AND.(J2+J1+MOD(M1,2).GE.2))
     &              TCOEFF=       XI2*EREL**5*DL*DE2/DENOMI
              IF((ABS(J2-J1).LE.1).AND.(J2+J1+MOD(M1,2).GE.1))
     &              TCOEFF=TCOEFF+XI1*EREL**3*DL*DE2/DENOMI
              TCSAVE(NTC)=TCOEFF
              TCIP0=TCIP0+TCOEFF
            ELSE
              TCIP0=TCIP0+TCSAVE(NTC)
              if(TCIP0.ge.TCS*RANDOM) goto 999
            ENDIF
20        CONTINUE
```

Fig. 3.4 Do 20 loop in subroutine tcdec.

```
              DO 50 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
                 NTC=NTC+1
                 IF(NTC.gt.MAXTC)THEN
                   IFCALC=1
                   IFSAVE=0
                 ENDIF
                 IF(IFCALC.eq.1)THEN
                   IF(IFJ.eq.1)THEN
                     CALL CALW(2*J1+MOD(M1,2)
     &                        ,2*J2+MOD(M2,2)
     &                        ,2*J3+MOD(M3,2)
     &                        ,LMAX,W)
                   ELSE
                     W=1
                   ENDIF
                   DL=DENLEV(IZ2,IN2,E2,DE2,J2)
                   TCOEFF=W*DL*DE2/DENOMI
                   TCIP0=TCIP0+TCOEFF
                   IF(IFSAVE.eq.1) TCSAVE(NTC)=TCOEFF
                   IF(IFDEC.eq.1.and.TCIP0.ge.TCS*RANDOM) GOTO 999
                 ELSE
                   TCIP0=TCIP0+TCSAVE(NTC)
                   IF(TCIP0.ge.TCS*RANDOM) GOTO 999
                 ENDIF
50             CONTINUE
```

Fig. 3.5 Do 50 loop in subroutine tcdec.

```
Synthesis Information
    Count|   Percent|    VL| Name
    57389|      98.6|     2|  tcdec_
      366|       0.6|     -|  mastbl_
      322|       0.6|     -|  casc_
       58|       0.1|     -|  massdis_
       29|       0.0|     1|  ranu2_
       13|       0.0|     -|  MAIN__

    58177|          |     2|  TOTAL
```

Fig. 3.6 Dynamic behavior of Original code (Inlining).

```
      IF(IFDEC.eq.1)THEN
        TCS=TCIP(IPDEC)
        IFSAVE=0
        IFCALC=0
      ELSE
        IFSAVE=1
        IFCALC=1
        DO 5 IP=0,IPMAX
          TCIP(IP)=0
5       CONTINUE
      ENDIF
```

Fig. 3.7 Definition of ifcalc and ifsave.

```
      TCS=0
      TCSF=0
      IF(ECAS(INUM).gt.EMIN) THEN
        IF(IFEVAP.eq.1) THEN
          CALL TCDEC(IZ1,IN1,E1,J1
     &            ,IDUMZ2,IDUMN2,DUME2,JDUM2
     &            ,IDUMZ3,IDUMN3,DUME3,JDUM3,TCIP,TCS,NTC,EDUM,0,0)
          TCN=TCIP(1)
        ELSE
          TCN=1
        ENDIF
      ENDIF
       < 中略 >
      IF(TCS+TCSF.gt.0) THEN
        IF(TCSF/(TCSF+TCS).gt.RND(1D0))THEN
          SUMAS(3,IZ1,IN1)=SUMAS(3,IZ1,IN1)+X/NREP
          CALL FISSION(IZ1,IN1,E1,J1
     &                ,IZ2,IN2,E2,J2,IZ3,IN3,E3,J3,BARF,SAF,ER)
        ELSE
          RANDOM=RND(1D0)
          TCIP0=0
          DO 100 IPDEC=0,IPMAX
            TCIP0=TCIP0+TCIP(IPDEC)
            IF(TCIP0.gt.TCS*RANDOM) GOTO 101
100       CONTINUE
          < 中略 >
          IPDEC=ipdmx
          goto 101
101       CONTINUE
          CALL TCDEC(IZ1,IN1,E1,J1
     &         ,IZ2,IN2,E2,J2,IZ3,IN3,E3,J3,TCIP,TCS,NTC,ER,1,IPDEC)
        ENDIF
```

Fig. 3.8 Subroutine call of tcdec in subroutine casc.

```
      DO 30 IP=1,IPMAX
      IF(IFDEC.eq.1) NTC=NTCOFFSET(IP)
      NTCOFFSET(IP)=NTC
      IF(IFDEC.eq.0.or.IP.eq.IPDEC) THEN
        IZ3=IZIP(IP)
        < 中略 >
        DO 40 IE2=1,IEMAX
           < 中略 >
           DO 50 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
                        .
                        .
                        .
```

Fig. 3.9 Conditional statement with ipdec in subroutine tcdec.

```
          IF(IFDEC.EQ.1)THEN
             DO 50 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
                          .
                          .
50           CONTINUE
          ELSE
             DO 51 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
                TCIPO=TCIPO+TCSAVE(NTC)
                if(TCIPO.ge.TCS*RANDOM) goto 999
51           CONTINUE
          ENDIF
```

Fig. 3.10 Division of do 50 loop.

```
      do 4000 I=1,NTC
         work(I)=0.0
         do 4100 J=1,I
            work(I)=work(I)+TCSAVE(J)
 4100    continue
 4000 continue
C
      do 4200 I=1,NTC
         if(work(I).ge.TCS*RANDOM) GOTO 999
 4200 continue
```

Fig. 3.11 Example of procedure which it is possible to vectorize.

```
 s              DO 501 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
 m                 NTC=NTC+1
 s                 IF(NTC.gt.MAXTC)THEN
    c                 IFCALC=1
 v                    IFSAVE=0
 v                 else
 v                    ifsave=1
 v                 ENDIF
    c               IF(IFCALC.eq.1)THEN
 v                    IF(IFJ.eq.1)THEN
 vi                      CALL CALW(2*J1+MOD(M1,2)
         &                      ,2*J2+MOD(M2,2)
         &                      ,2*J3+MOD(M3,2)
         &                      ,LMAX,W)
 v                    ELSE
 v                       W=1
 v                    ENDIF
 vi                   DL=DENLEV(IZ2,IN2,E2,DE2,J2)
 v                    TCOEFF=W*DL*DE2/DENOMI
 v                    TCIP0=TCIP0+TCOEFF
 v                    IF(IFSAVE.eq.1) TCSAVE(NTC)=TCOEFF
    c                  IF(IFDEC.eq.1.and.TCIP0.ge.TCS*RANDOM) GOTO 999
    c                endif
 m   501         continue
```

Fig. 3.12 DO loop that becomes a recursive reference.

```
if(ifdec.eq.1)then
   do ie2 = 1,iemax
     do j2 = ・・・・・
       ・ntc の増分値の計算
     continue
   continue
  ・maxtc を超えたかどうかの判定
   do ie2 = 1,iemax
     do j2 = ・・・・・
       ・ tcoeff の計算
       if(ntc < maxtc)then
         tcsave(ntc)=tcoeff
       endif
     continue
   continue
else
   do ie2 = 1,iemax
      do j2 = ・・・・・
        if(ntc < maxtc)then
           tcip0 = tcip0+tcsave(ntc)
        else
           ・tcoeff の計算
           tcip0 = tcip0+tcoeff
        endif
        if(tcip0 .ge. tcs*random) goto 999
      continue
   continue
endif
```

Fig. 3.13 Outline of vectorization of subroutine tcdec.

```
          ntctmp=0
          do 401 ie2=1,iemax
             DE2=DE(IE2)
             E2=EE(IE2)
             EREL=E1-E2-QVAL12
             ANGMX2=2*RMASS*M3*M2/M1
     &              *(R2+R3)**2
     &              *(EREL-B23)
             if(ANGMX2.ge.0) then
                ltmp(ie2)=INT(SQRT(ANGMX2)/HC)
                iesv(ie2)=1
                itmp1(ie2)=IFJ*MAX(0,J1-ltmp(ie2)-1)
                itmp2(ie2)=IFJ*(J1+ltmp(ie2)+1)
                ntctmp=ntctmp+itmp2(ie2)-itmp1(ie2)+1
             else
                iesv(ie2)=0
             endif
 401      continue
```

Fig. 3.14 Computation of increment of ntc.

```
          if(ntctmp+ntc.gt.maxtc)then
             ng=ip
             write(6,*) ' Over MAXTC : NTC = ',NTC
          else
             ng=-1
          endif
```

Fig. 3.15 Conditional statement about maxtc.

```
          DO 40 IE2=1,IEMAX
             if(iesv(ie2).eq.1)then
                DO 50 J2=itmp1(ie2),itmp2(ie2)
                   NTC=NTC+1
                   DE2=DE(IE2)
                   E2=EE(IE2)
                   LMAX=ltmp(ie2)
                   IF(IFJ.eq.1)THEN
                      CALL CALW(2*J1+MOD(M1,2)
    &                        ,2*J2+MOD(M2,2)
    &                        ,2*J3+MOD(M3,2)
    &                        ,LMAX,W)
                   ELSE
                      W=1
                   ENDIF
                   DL=DENLEV(IZ2,IN2,E2,DE2,J2)
                   TCOEFF=W*DL*DE2/DENOMI
                   TCIP0=TCIP0+TCOEFF
                   if(ng.lt.0)then
                      TCSAVE(NTC)=TCOEFF
                   endif
50              CONTINUE
             endif
40        CONTINUE
```

Fig. 3.16 Computation of tcoeff.

```
          DO 44 IE2=1,IEMAX
             DE2=DE(IE2)
             E2=EE(IE2)
             EREL=E1-E2-QVAL12
             ANGMX2=2*RMASS*M3*M2/M1
     &              *(R2+R3)**2
     &              *(EREL-B23)
             if(ANGMX2.ge.0) then
                LMAX=INT(SQRT(ANGMX2)/HC)
*vocl loop,scalar
                DO 55 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
                   NTC=NTC+1
                   if(ng.lt.ip)then
                      TCIP0=TCIP0+TCSAVE(NTC)
                   else
                      IF(IFJ.eq.1)THEN
                         CALL CALW(2*J1+MOD(M1,2)
     &                           ,2*J2+MOD(M2,2)
     &                           ,2*J3+MOD(M3,2)
     &                           ,LMAX,W)
                      ELSE
                         W=1
                      ENDIF
                      DL=DENLEV(IZ2,IN2,E2,DE2,J2)
                      TCOEFF=W*DL*DE2/DENOMI
                      TCIP0=TCIP0+TCOEFF
                   endif
                   IF(TCIP0.ge.TCS*RANDOM) GOTO 999
 55             CONTINUE
             endif
 44       CONTINUE
```

Fig. 3.17 Procedure in case of ipdec=0.

```
      DO 11 IE2=1,IEMAX
c       YY=1.0*IE2/IEMAX
        YY=(IE2-0.5)/(IEMAX-1)
        IF(IE2.eq.IEMAX)YY=1.0
        EE2=EMAX*YY**pow
c       EE2=EMAX*(acos(1-2*YY)/PI)**pow
        EE(IE2)=(EE2+EE1)/2
        DE(IE2)=EE2-EE1
        EE1=EE2
11    CONTINUE
```

Fig. 3.18 Do 11 loop in subroutine tcdec.

```
      DO 11 IE2=1,IEMAX
        YY=(IE2-0.5)/(IEMAX-1)
        IF(IE2.eq.IEMAX)YY=1.0
        work1(ie2)=EMAX*YY**pow
11    continue
      ee(1)=work1(1)/2
      de(1)=work1(1)
      do 12 ie2=2,iemax
        EE(IE2)=(work1(ie2)+work1(ie2-1))/2
        DE(IE2)=work1(ie2)-work1(ie2-1)
12    CONTINUE
```

Fig. 3.19 Divided do 11 loop for vectiorization.

```
        DO 10 IE2=1,IEMAX
          DE2=DE(IE2)
          E2=EE(IE2)
          EREL=E1-E2
          DO 20 J2=IFJ*MAX(0,J1-2),IFJ*(J1+2)
            NTC=NTC+1
            IF(IFDEC.eq.0)THEN
              DL=DENLEV(IZ1,IN1,E2,DE2,J2)
              IF((ABS(J2-J1).LE.2).AND.(J2+J1+MOD(M1,2).GE.2))
     &              TCOEFF=       XI2*EREL**5*DL*DE2/DENOMI
              IF((ABS(J2-J1).LE.1).AND.(J2+J1+MOD(M1,2).GE.1))
     &              TCOEFF=TCOEFF+XI1*EREL**3*DL*DE2/DENOMI
              TCSAVE(NTC)=TCOEFF
              TCIP0=TCIP0+TCOEFF
            ELSE
              TCIP0=TCIP0+TCSAVE(NTC)
              if(TCIP0.ge.TCS*RANDOM) goto 999
            ENDIF
20        CONTINUE
10      CONTINUE
```

Fig. 3.20 Do 10 and do 20 loops in subroutine tcdec.

```
      IF(IFDEC.eq.0)THEN
         DO 10 IE2=1,IEMAX
            DE2=DE(IE2)
            E2=EE(IE2)
            EREL=E1-E2
            DO 20 J2=IFJ*MAX(0,J1-2),IFJ*(J1+2)
               NTC=NTC+1
               DL=DENLEV(IZ1,IN1,E2,DE2,J2)
c               TCOEFF=0.D0
c               IF((ABS(J2-J1).LE.2).AND.(J2+J1+MOD(M1,2).GE.2))
c     &            TCOEFF=       XI2*EREL**5*DL*DE2/DENOMI
c               IF((ABS(J2-J1).LE.1).AND.(J2+J1+MOD(M1,2).GE.1))
c     &            TCOEFF=TCOEFF+XI1*EREL**3*DL*DE2/DENOMI
               IF((ABS(J2-J1).LE.2).AND.
     &                           (J2+J1+MOD(M1,2).GE.2)) then
                  TCOEFF=XI2*EREL**5*DL*DE2/DENOMI
                  IF((ABS(J2-J1).LE.1).AND.
     &                           (J2+J1+MOD(M1,2).GE.1)) then
                     TCOEFF=XI2*EREL**5*DL*DE2/DENOMI+
     &                            XI1*EREL**3*DL*DE2/DENOMI
                  endif
               else
                  TCOEFF=0.D0
               endif
               TCSAVE(NTC)=TCOEFF
               TCIP0=TCIP0+TCOEFF
 20         CONTINUE
 10      CONTINUE
      ELSE
         DO 101 IE2=1,IEMAX
            DE2=DE(IE2)
            E2=EE(IE2)
            EREL=E1-E2
            DO 211 J2=IFJ*MAX(0,J1-2),IFJ*(J1+2)
               NTC=NTC+1
               TCIP0=TCIP0+TCSAVE(NTC)
               if(TCIP0.ge.TCS*RANDOM) goto 999
 211        CONTINUE
 101     CONTINUE
      ENDIF
```

Fig. 3.21 Vectorized do 10 and do 20 loops.

```
          IF(IFJ.eq.1)THEN
             CALL CALW(2*J1+MOD(M1,2)
     &                ,2*J2+MOD(M2,2)
     &                ,2*J3+MOD(M3,2)
     &                ,LMAX,W)

          . . . . . .

      SUBROUTINE CALW(J1,J2,J3,LMAX,W)
      IMPLICIT REAL*8(A-H,O-Z)
      W=0D0
      IF(MOD(J1+J2+J3,2).EQ.1)GOTO 15
      JJ1=ABS(J2-J3)
      JJ2=J2+J3
      DO 30 JJ=JJ1,JJ2,2
        L1=ABS(J1-JJ)/2
        L2=MIN((J1+JJ)/2,LMAX)
        W=W+DIM(L2+1-L1,0)
30    CONTINUE
15    CONTINUE
      END
```

Fig. 3.22 Subroutine calw.

```
IZ3=IZIP(IP)
IN3=INIP(IP)
IZ2=IZ1-IZ3
IN2=IN1-IN3
M2=IZ2+IN2
M3=IZ3+IN3
J3=JIP(IP)*IFJ
```

Fig. 3.23 Definition of m2, m3 and j3.

```
IZIP(1)=0
INIP(1)=1
JIP(1)=0

IZIP(2)=1
INIP(2)=0
JIP(2)=0

IZIP(3)=2
INIP(3)=2
JIP(3)=0

IZIP(4)=1
INIP(4)=1
JIP(4)=1

IZIP(5)=1
INIP(5)=2
JIP(5)=0

IZIP(6)=2
INIP(6)=1
JIP(6)=0
```

Fig. 3.24 Definition of array jip.

```
      SUBROUTINE CALW(J1,J2,J3,LMAX,W)
      IMPLICIT REAL*8(A-H,O-Z)
      W=0D0
      IF(MOD(J1+J2+J3,2).EQ.1)GOTO 15
      JJ1=ABS(J2-J3)
      JJ2=J2+J3
      if(jj2-jj1.eq.0)then
         L1=ABS(J1-JJ1)/2
         L2=MIN((J1+JJ1)/2,LMAX)
         W=DIM(L2+1-L1,0)
      elseif(jj2-jj1.eq.2)then
        L1=ABS(J1-JJ1)/2
        L2=MIN((J1+JJ1)/2,LMAX)
        L3=ABS(J1-(JJ1+2))/2
        L4=MIN((J1+JJ1+2)/2,LMAX)
        W=DIM(L2+1-L1,0)+DIM(L4+1-L3,0)
      else
        L1=ABS(J1-JJ1)/2
        L2=MIN((J1+JJ1)/2,LMAX)
        L3=ABS(J1-(JJ1+2))/2
        L4=MIN((J1+JJ1+2)/2,LMAX)
        L5=ABS(J1-(JJ1+4))/2
        L6=MIN((J1+JJ1+4)/2,LMAX)
        W=DIM(L2+1-L1,0)+DIM(L4+1-L3,0)+DIM(L6+1-L5,0)
      endif
15    CONTINUE
      END
```

Fig. 3.25 Subroutine calw without do loop.

```
            do 41 ie2=1,iemax
               if(iesv(ie2).eq.1)then
                  do 50 j2=itmp1(ie2),itmp2(ie2)
                     ntc=ntc+1
                     j2idx(ntc)=j2
                     ieidx(ntc)=ie2
                     lmaxsv(ntc)=ltmp(ie2)
50                continue
               endif
41          continue
```

Fig. 3.26 Storing procedure of ie2, j2 and lmax variables.

```
            do 411 ie2=1,iemax
               if(iesv(ie2).eq.1)then
                  do 501 j2=itmp1(ie2),itmp2(ie2)
                     ntc=ntc+1
                     lmax=ltmp(ie2)
                     de2=de(ie2)
                     e2=ee(ie2)
                     IF(IFJ.eq.1)THEN
                        CALL CALW(2*J1+MOD(M1,2)
    &                           ,2*J2+MOD(M2,2)
    &                           ,2*J3+MOD(M3,2)
    &                           ,LMAX,W)
                     ELSE
                        W=1
                     ENDIF
                     DL=DENLEV(IZ2,IN2,E2,DE2,J2)
                     TCOEFF=W*DL*DE2/DENOMI
                     TCIP0=TCIP0+TCOEFF
501               continue
               endif
411         continue
```

Fig. 3.27 Do loop that not use increase ntc.

```
              do 444 i=ntcold+1,ntc
                 ie2=ieidx(i)
                 j2=j2idx(i)
                 de2=de(ie2)
                 e2=ee(ie2)
                 lmax=lmaxsv(i)
                 IF(IFJ.eq.1)THEN
                    CALL CALW(2*J1+MOD(M1,2)
     &                   ,2*J2+MOD(M2,2)
     &                   ,2*J3+MOD(M3,2)
     &                   ,LMAX,W)
                 ELSE
                    W=1
                 ENDIF
                 DL=DENLEV(IZ2,IN2,E2,DE2,J2)
                 TCOEFF=W*DL*DE2/DENOMI
                 TCIPO=TCIPO+TCOEFF
                 TCSAVE(i)=TCOEFF
 444          continue
```

Fig. 3.28 Do loop that use increase ntc.

```
*vocl loop,scalar
              do 441 i=ntcoffset(ip)+1,ntcoffset(ip+1)
                 TCIP0=TCIP0+TCSAVE(i)
                 IF(TCIP0.ge.TCS*RANDOM)then
                    ie2=ieidx(i)
                    de2=de(ie2)
                    e2=ee(ie2)
                    erel=e1-e2-qval12
                    lmax=lmaxsv(i)
                    j2=j2idx(i)
                    GOTO 999
                 endif
 441           continue
```

Fig. 3.29 Do loop that use increase ntc in case of ifdec=0.

```
      IF(IFDEC.eq.1) NTC=NTCOFFSET(IP)
      NTCOFFSET(IP)=NTC
```

Fig. 3.30 Definition of array ntcoffset.

```
      if(ifdec.eq.0)then
            . . .
         if(ng.lt.0)then
            . . .
         else
            . . .
         endif
         ntcoffset(ip+1)=ntc      ← *
      else
           . . .
      endif
```

Fig. 3.31 Storing procedure of increased ntc.

```
*vocl loop,scalar
              DO 44 IE2=1,IEMAX
                 DE2=DE(IE2)
                 E2=EE(IE2)
                 EREL=E1-E2-QVAL12
                 ANGMX2=2*RMASS*M3*M2/M1
     &                *(R2+R3)**2
     &                *(EREL-B23)
                 if(ANGMX2.ge.0) then
                    LMAX=INT(SQRT(ANGMX2)/HC)
*vocl loop,scalar
                    DO 55 J2=IFJ*MAX(0,J1-LMAX-1),IFJ*(J1+LMAX+1)
                       NTC=NTC+1
                       IF(IFJ.eq.1)THEN
                          CALL CALW(2*J1+MOD(M1,2)
     &                              ,2*J2+MOD(M2,2)
     &                              ,2*J3+MOD(M3,2)
     &                              ,LMAX,W)
                       ELSE
                          W=1
                       ENDIF
                       DL=DENLEV(IZ2,IN2,E2,DE2,J2)
                       TCOEFF=W*DL*DE2/DENOMI
                       TCIP0=TCIP0+TCOEFF
                       IF(TCIP0.ge.TCS*RANDOM) GOTO 999
 55                 CONTINUE
                 endif
 44           CONTINUE
```

Fig. 3.32 Procedure in case of ifdec=0 and ntc exceeds maxtc.

```
      itmp=ifj*(j1+2)-ifj*max(0,j1-2)+1
      max10=iemax*itmp
      IF(IFDEC.eq.0)THEN
         do 10 i=1,max10
            ie2=(i-1)/itmp+1
            j2=ifj*max(0,j1-2)+mod(i-1,itmp)
            ieidx(i)=ie2
            j2idx(i)=j2
            DE2=DE(IE2)
            E2=EE(IE2)
            EREL=E1-E2
            DL=DENLEV(IZ1,IN1,E2,DE2,J2)
            IF((ABS(J2-J1).LE.2).AND.
     &           (J2+J1+MOD(M1,2).GE.2)) then
               TCOEFF=XI2*EREL**5*DL*DE2/DENOMI
               IF((ABS(J2-J1).LE.1).AND.
     &              (J2+J1+MOD(M1,2).GE.1)) then
                  TCOEFF=XI2*EREL**5*DL*DE2/DENOMI+
     &                 XI1*EREL**3*DL*DE2/DENOMI
               endif
            else
               TCOEFF=0.D0
            endif
            TCSAVE(i)=TCOEFF
            TCIP0=TCIP0+TCOEFF
10       CONTINUE
         ntc=max10
```

Fig. 3.33 Modified do 10 and do 20 loops in case of ifdec=1.

```
            do 101 i= 1,max10
              ie2=ieidx(i)
              j2=j2idx(i)
              DE2=DE(IE2)
              E2=EE(IE2)
              EREL=E1-E2
              TCIP0=TCIP0+TCSAVE(i)
              if(TCIP0.ge.TCS*RANDOM) goto 999
 101        CONTINUE
```

Fig. 3.34 Modified do 10 and do 20 loops in case of ifdec=0.

```
      DO 20 I=1,NNUC
        QMIN=0D0
        IDEC(I)=0
        DO 30 J=I,1,-1
          DO 40 K=I,1,-1
            IF((MAS(I)-MAS(J)-MAS(K).EQ.0).AND.
     &              (NZ(I)-NZ(J)-NZ(K).EQ.0)) THEN
              QVAL=EB0(I)-EB0(J)-EB0(K)
              IF(QVAL.LT.QMIN.and.
     &             K.le.IFDATA(IZIP(IPMAX),INIP(IPMAX))) THEN
                QMIN=QVAL
                IDEC(I)=J
              END IF
            END IF
 40       CONTINUE
 30     CONTINUE
 20   CONTINUE
```

Fig. 3.35 Do 20,do 30 and do 40 loops in subroutine mastbl.

```
      DO 20 I=1,NNUC
        QMIN=0D0
        jsave=0
        DO 30 J=I,1,-1
          DO 40 K=I,1,-1
            IF((MAS(I)-MAS(J)-MAS(K).EQ.0).AND.
     &              (NZ(I)-NZ(J)-NZ(K).EQ.0)) THEN
              QVAL=EB0(I)-EB0(J)-EB0(K)
              if(K.le.IFDATA(IZIP(IPMAX),INIP(IPMAX))) THEN
                 IF(QVAL.LT.QMIN)then
                    QMIN=QVAL
                    jsave=j
                 endif
              END IF
            END IF
 40       CONTINUE
 30     CONTINUE
        IDEC(I)=Jsave
 20   CONTINUE
```

Fig. 3.36 Vectorized Do loops in mastbl.

```
Synthesis Information
    Count|   Percent|    VL| Name
     9375|      95.7|    19|  tcdec_
      281|       2.9|     -|  casc_
       59|       0.6|   354|  mastbl_
       56|       0.6|     -|  massdis_
       13|       0.1|     -|  MAIN__
       11|       0.1|     1|  ranu2_

     9795|          |    23|  TOTAL
```

Fig. 3.37 Dynamic behavior of vectorized version 1.

```
Synthesis Information
    Count|   Percent|    VL| Name
     2688|      87.2|   323|  tcdec_
      258|       8.4|     -|  casc_
       59|       1.9|   317|  mastbl_
       53|       1.7|     -|  massdis_
       13|       0.4|     -|  MAIN__
       11|       0.4|     1|  ranu2_

     3082|          |   321|  TOTAL
```

Fig. 3.38 Dynamic behavior of vectorized version 2.

参考文献

[1] T.Maruyama,A.Ono,A,Ohnishi and H.Horiuchi,Prog. Theor. Phys. 87(1992) 1367.

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

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

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

4. SSPHEATコードのベクトル化

HENDEL炉内構造物実証試験部(T_2)3次元熱伝導解析コードSSPHEATの高速化作業を行った．本コードはベクトル化等高速化の配慮が不十分であり，特に非定常解析において解析効率を落していた．例えば，50時間を解析時間ステップ数(繰返しステップ数)3000回で解析する場合，cpu時間は約380分と予想されている．cpu時間は繰返しステップ数と共に増大するため，このステップ数を大きくするには高速化対応が必要であった．

高速化作業は，ベクトルパラレルコンピュータVPP500(以降，単にVPPと呼ぶ)におけるベクトル化作業が大半であったが，更に高速化を図るため並列化の可能性について調査を行った．また，高速化されても，混雑等によりターンアラウンドタイムが長くなる可能性がある．こうした状況を少しでも緩和できるように，分散メモリ型並列サーバAP3000へのインストール作業も行った．

以下に，これらの作業内容について報告する．

4.1 コード概要

SSPHEATコード[1]は，複雑な冷却材ヘリウムガスの流路を持つ炉内構造物実証試験部(T_2)の温度分布を求めるために開発された数値解析コードである．T_2試験部は大型構造機器実証試験ループ(HENDEL)に設置された第2の試験部であり，その内側の二重管の内部を高温ヘリウムガスが流れ，二重管の外側と圧力容器の間を低温ヘリウムガスが流れている．この低温ヘリウムガスは圧力容器が設計温度以下になるように調節している．本コードは，このT_2試験部全体の解析を行うため，この冷却材ヘリウムガスの流れ方向の温度変化，熱放射，内部発熱を考慮した3次元熱伝導解析を行っている．

解析手法としては有限要素法が使用されている．解析対象を有限な要素に分割し，すべての要素について，ガラーキン法に基づく有限要素法により定式化し，境界条件を含めたマトリクス式の形を導いている．そして，解析対象全体の有限要素式を全ての要素について集めて，全体の有限要素式を得ている．また，この時の連立一次方程式の解法として，ウェイブ・フロント法を使用している．ウェイブ・フロント法では，全体を要素ごとに処理し，作成した1つの要素剛性マトリクスを全体マトリクスに累積的に組み立てていき，その時点で，その後要素データ内に現れない節点があれば三角分解を行う．三角分解した節点は組み立てエリアには不用となるため，外部ファイルに出力し，空きエリアとする．そして，この空きエリアを今後の組み立て処理に再利用している．こうして使用エリアを少なくしている．

4.2 ベクトル化

ベクトル化作業を進めるにあたり，種々のケースを考慮して6種類のテストデータを選択した．内訳は以下の通りである．要素HPIPE2(3次元2節点熱・流路要素)を含まないデータと含むデータとでは，対応処理ルーチンが異なる．

- 定常解析データ：A5VPP3, COLOMN34, RPVVCS1, SP, SP2 の 5 種類

  要素 HPIPE2 を含まないデータ：A5VPP3, COLOMN34
  要素 HPIPE2 を含むデータ　　：RPVVCS1, SP, SP2

- 非定常解析データ：sp61.txt の 1 種類

これらのテストデータを利用して動的挙動解析を行い，高速化のチューニングを行った．

4.2.1 動的挙動解析

計算コストの分布状況，及び自動ベクトル化の状況を調査するため，動的挙動解析ツールである SAMPLER [2]，及び ANALYZER [3] を利用した．SAMPLER は，VPP 上で動作するツールであり，実行可能プログラムを対象にしている．これによりルーチン単位の計算コスト分布を得ることができる．一方，ANALYZER の方は，GSP 上で動作するツールであり，ソースプログラムを対象にしている．ルーチン単位，ループ単位，文単位の計算コスト，実行回数，及びループの回転数等の情報が得られる．ルーチン単位の計算コスト分布に関しては，SSPHEAT が実際に動作するのが VPP 上であるため，VPP 上の解析結果である SAMPLER のものの方がより実情に近い値を示している．

各テストデータに対するベクトル化対応前の SAMPLER の出力結果を Fig. 4.1 に示す．計算コストが BKINDX,SOLWAV, SOLWVX の 3 ルーチンに集中していることが判る．その占める割合はテストデータ RPVVCS1 の場合 54.8% であったが，他の場合は 83.9 ～ 92.2% であった．この 3 ルーチンのベクトル化対応前の ANALYZER の出力結果 (抜粋) を Fig. 4.2 に示す．Fig. 4.1, 4.2 における 3 ルーチンの計算コストの分布状況，ベクトル化状況は以下のとおりであった．

(1)BKINDX ルーチン

相対行番号 529 ～ 536 にルーチン全体の 74.5 ～ 93.7% の計算コストが集中している．対応箇所を Fig. 4.3 に示す．Fig. 4.3 からも判るように，この箇所は DO ループになっておらず，ベクトル化の対象になっていない．また，ルーチン全体としてのベクトル化率は 0% であった．

(2)SOLWAV ルーチン

本ルーチンはテストデータに要素 HPIPE2 を含まない場合に実行される．従って，A5VPP3, COLOMN34 の場合に計算コストがカウントされた．Fig. 4.2 から判るように，DO 1500 に計算コストが集中しており，97.3，98.6% を占めている．ベクトル化率は 2.8，1.3% であった．

(3)SOLWVX ルーチン

本ルーチンはテストデータに要素 HPIPE2 を含む場合に実行される．従って，RPVVCS1, SP，SP2，sp61.txt の場合に計算コストがカウントされた．Fig. 4.2 から判るように，計算コストは DO 1600 に集中しており，88.5 ～ 94.9% を占めている．ベクトル化率は 38.8 ～ 39.5% であった．

4.2.2 チューニング

動的挙動解析により得たデータを参考にして，高速化のためのチューニングを実施した．以下にその作業内容を記述する．

4.2.2.1 BKINDX ルーチン

BKINDX ルーチンの計算コストは，次の3ヵ所に99%以上が集中している．DO 3000 ループ(相対行番号 529～536 を含む)，DO 200 ループ，そして DO 4000 ループである．本ルーチンにおいては，この3ヵ所のベクトル化チューニングを行った．それぞれの DO ループのコーディング内容を Fig. 4.4 の修正前欄に示す．

(1)DO 3000 ループのベクトル化

DO 3000 がベクトル化されないのは，内部にベクトル化できない上向き GO TO 文によるループ(相対行番号 529～536)を含んでいることが主な理由であった．この内部ループをベクトル化することが最低限必要なことであったが，処理の一部にしか過ぎないため，DO 3000 全体の処理方式を見直すことにより，ベクトル化を推進することにした．

DO 3000 の処理は，配列 MASTER の各要素をその絶対値に関して昇順に整列させるための，インデックス配列 NBKCOL を作成する処理である．つまり，1 ≦ JJ ≦ MXUSED の範囲の JJ について，MASTER(JJ) ≠ 0 となる MASTER(JJ) を，その絶対値が昇順になるよう式(4.1)のように整列させる．

$$|\mathrm{MASTER(I1)}| < |\mathrm{MASTER(I2)}| < \cdots < |\mathrm{MASTER(In)}| \tag{4.1}$$

この時，式(4.2)となるように配列 NBKCOL を作成する．

$$\mathrm{NBKCOL(1)} = \mathrm{I1},\ \mathrm{NBKCOL(2)} = \mathrm{I2},\ \cdots,\ \mathrm{NBKCOL(n)} = \mathrm{In} \tag{4.2}$$

配列 MASTER はこの処理の前までに MASTER(JJ) ≠ 0 である MASTER(JJ) について，I ≠ J ならば MASTER(I) ≠ MASTER(J) となるよう処置されている．

従ってこの処理をベクトル化するため，まず MASTER(JJ) ≠ 0 である｜MASTER(JJ)｜とそのインデックス JJ を抽出した．つぎに，抽出した各｜MASTER(JJ)｜について，昇順の順位を求めてインデックス JJ との対応をとるように変更した．

(2)DO 200 ループのベクトル化

DO 200 の処理は配列 MASTER に新たなデータを取り込む処理の一部である．ここで DO 200 の処理が関係するのは，次の2つである．1つは，MASTER の中に既に同じ値が存在する場合，取り込み処理をスキップして配列 KBIG にその時の MASTER のインデックス KP を格納する処理である．もう1つは，MASTER(KP) ＝ 0 のものがあれば，この時のインデックス KP を LOCATE に格納する処理である．

DO 200 がベクトル化されないのは，KP 格納の際に，「変数定義後の途中からの飛び出し」と「定義前の引用」があるためである．カウンタ icnt を設けてそれぞれの KP 値を求めることにより回避した．

(3)DO 4000 ループのベクトル化

DO 4000 がベクトル化されないのは，MASTER(MD) ＝－ MASTER(MD) において，添字変数 MD がリストになっており，配列 MASTER が回帰データとみなされたためである．実際には，MD は本ループの中で MD ＝ NBKCOL(JJ) として定義されているが，式 (4.3) の関係があり回帰参照は発生しない．従って，最適化制御行による NOVREC 情報により回帰参照がない旨の指示を行うことによりベクトル化の対応を行った．

I≠Jならば

$$\text{NBKCOL(I)} \neq \text{NBKCOL(J)}$$
$$\text{MASTER(NBKCOL(I))} \neq \text{MASTER(NBKCOL(J))} \tag{4.3}$$

BKINDX ルーチンの変更内容を Fig. 4.4 の修正後欄に示す．

4.2.2.2 SOLWAV ルーチン

本ルーチンは DO 1500 ループに計算コストが集中しており，そのループの中で文関数 NSS が計算されている．コーディング内容を Fig. 4.5 の修正前欄に示す．従って，本ルーチンにおいては，計算コストの集中している DO 1500 ループのベクトル化，及び文関数 NSS の先行計算を行った．

また，文関数 NSS の先行計算については，二重ループ DO 2000・1500，及び DO 4000・400 も対象にした．

(1)DO 1500 ループのベクトル化

DO 1500 がベクトル化されないのは，実行文 BK(NLOW) ＝ BK(NLOW) － SHIFT*BKVEC(LL) において，添字変数 NLOW の値が不明のため配列 BK が回帰データとみなされたためである．NLOW，及び NLOW の定義式の 1 変数である KL は，本ループの中で以下のように定義されている．

$$\text{KL} = \text{NBKCOL(LL)} \quad (2 \leqq \text{LL} \leqq \text{NCOLM}) \tag{4.4}$$
$$\text{NLOW} = \text{KL}-\text{K0}+\text{L0} \quad (\text{KL} > \text{K0}) \tag{4.5}$$
$$\text{NLOW} = \text{K1}-\text{KL}+\text{NSS(KL,MXCOLM)} \quad (\text{KL} \leqq \text{K0}) \tag{4.6}$$

DO 1500 の中では，K1，K0，L0，MXCOLM が定数であるため，NLOW の値は KL の値に依存することになり，式 (4.7)，(4.8) のような関係となる．

$$\text{NLOW} = \text{KL}+\text{定数} \quad (\text{KL} > \text{K0}) \tag{4.7}$$
$$\text{NLOW} = (-1/2)(\text{KL}-(2\times\text{MXCOLM}+1)/2)^2+\text{定数} \quad (\text{KL} \leqq \text{K0}) \tag{4.8}$$

2 ≦ LL ≦ NCOLM の範囲では，2 ≦ KL ＝ NBKCOL(LL) ≦ MXCOLM であるため，LL ＝ n の時の KL の値を KLn，それに対応する NLOW の値を NLOWn で表すと，以下のことが言える．

2 ≦ i,j ≦ NCOLM となる任意の整数 i,j について
i ≠ j　ならば　KLi ≠ KLj であり，NLOWi ≠ NLOWj である．

これにより，DO 1500の中では，配列BKは回帰データではないことが判る．従って，回帰参照がない旨の指示を行うことによりベクトル化の対応を行った．

(2) 文関数NSSの先行計算

文関数NSS(NBKCOL(I),MXCOLM)の計算が二重ループDO 4000・400，DO 2000・1500の中で，Iに関して重複して行なわれている．従って，Iが1～NCOLMまでの計算を先行して集中的に実施することにより計算量の削減を図った．結果としてA5VPP3，及びCOLOMN34のテストデータにおいて，それぞれ約30分の1，70分の1の計算量とすることができた．

SOLWAVルーチンの変更内容をFig. 4.5の修正後欄に示す．

4.2.2.3 SOLWVXルーチン

本ルーチンはDO 1600ループに計算コストが集中しており，直前のDO 1500ループのループ範囲はDO 1600ループのものと一致している．コーディング内容をFig. 4.6の修正前欄に示す．従って，本ルーチンにおいては，計算コストの集中しているDO 1600ループのベクトル化を中心に，それを含むDO 2000ループのベクトル化を行った．

(1)DO 1600ループのベクトル化

DO 1600がベクトル化されないのは，以下のように添字変数LIがリストになっており，配列BKが回帰データとみなされたためである．

$$\mathrm{LI} = \mathrm{NBKCOL(LL)} \tag{4.9}$$

$$\mathrm{BK(LJ,LI)} = \mathrm{BK(LJ,LI)-VEC1(LL)} \tag{4.10}$$

$$\mathrm{BK(LI,LJ)} = \mathrm{BK(LI,LJ)-VEC2(LL)} \tag{4.11}$$

実際には，

$$\mathrm{LJ} = \mathrm{NBKCOL(II)} \tag{4.12}$$

$$\mathrm{LI} = \mathrm{NBKCOL(LL)} \quad (\mathrm{LL:II+1\sim NXOLM}) \tag{4.13}$$

であり，

$$\mathrm{I\neq J\text{ならば}NBKCOL(I)\neq NBKCOL(J)} \tag{4.14}$$

であるため，DO 1600の中では，配列BKは回帰データではないことが判る．従って，回帰参照がない旨の指示(NOVREC)を行うことによりベクトル化の対応を行った．

(2)DO 2000ループのベクトル化

DO 2000の処理では，｜BUVEC(II)｜＞EPSまたは，｜BLVEC(II)｜≧EPSが満たされるIIについてのみ，DO 1500，DO 1600等の処理が行われている．そこで，この判定処理をベクトル化するため，この条件を満足するインデックス抽出処理をDO 2000のループの外に移動した．そして，抽出したインデックスについてDO 1500，DO 1600を行うように変更した．この時，DO 1500とDO 1600のループ範囲が一致しているので1本化した．

SOLWVXルーチンの変更内容をFig. 4.6の修正後欄に示す．

4.2.2.4 その他

BKINDX, SOLWAV, 及びSOLWVXの3ルーチン以外のルーチンについては，全体の中でのコスト比率が低くベクトル長の長いDOループを含むものが少ない．そこでANALYZERの出力結果を調査して，解析部のルーチンの中でベクトル化効果が期待できる可能性のあるDOループ(ベクトル長が10以上)を含むルーチンを選択し，ベクトルコンパイルの対象にした．残りのルーチン群については，スカラコンパイルの対象にした．ベクトルコンパイルの対象にしたルーチンは以下の12ルーチンである．

```
BKCLER,BKCLEX,FBR   ,FBRX  ,RESTIN,RIOBB ,
RIODD ,SHIFT ,SRCR  ,VSREAD,VSWRIT,WIODD
```

但し，BKCLER, FBR, 及びFBRXルーチンについては，以下の変更を行った．

(1)BKCLERルーチン

本ルーチンでは，DO 400がベクトル化されていなかった．添字変数NLOWがリストになっており，配列BKが回帰データとみなされたためである．コーディング内容をFig. 4.7の修正前欄に示す．DO 400の構成はSOLWAVルーチンのDO 1500のものと似ており，同様の対応を行った．つまり，文関数NSSの先行計算，及びNOVREC情報の挿入である．BKCLERルーチンの変更内容をFig. 4.7の変更後欄に示す．

(2)FBR, FBRXルーチン

FBR, FBRXルーチンのどちらの場合も，DO 700がベクトル化されていなかった．この二つのDO 700はコーディングが全く同じであり，添字変数K(=INVEC(I))がリストになっており，配列YVECTが回帰データとみなされたためいずれもベクトル化されなかった．コーディング内容をFig. 4.8の修正前欄に示す．実際は，I ≠ JならばINVECL(I) ≠ INVECL(J)であるため回帰参照は発生しない．従って，いずれの場合も，NOVREC情報により回帰参照がない旨の指示を行うことによりベクトル化の対応を行った．変更内容をFig. 4.8の変更後欄に示す．

4.2.3 計算結果の評価及びベクトル化効果

4.2.3.1 計算結果の評価

VPP上でFORTRANプログラムを実行する場合，同じソースコードを用いたとしても，スカラ演算とベクトル演算との丸め方法の違い，コンパイラによる最適化方法の違いにより計算結果に違いが生じる．従って，ベクトル化の変更等何らかの変更により，計算結果に違いが生じるのは一般的である．今回の変更作業においても，オリジナル版をスカラコンパイルした実行可能プログラムによる出力結果とベクトル化版実行可能プログラムによる出力結果とでは，違いが生じていた．この出力結果の違いが今回の変更によるものなのか，丸め方法，最適化方法等の計算環境によるものなのかの確認を行った．

確認方法は，基本的には計算環境を同じくするため，オリジナル版，ベクトル化版共に最適化抑止したスカラコンパイルを行いその計算結果を比較するのであるが，ここでは計算時間短縮のため，以下のような簡便な方法で行った．

- 丸め方法を合わせるためオリジナル版，ベクトル化版共にスカラ演算の丸め方法を切り捨てモードにするため MAIN ルーチンの先頭に CALL CHROUND(3) を挿入した．

- オリジナル版，ベクトル化版共に実行結果の精度に影響する「演算評価方法を変更する最適化」を抑止するが，オリジナル版の方はスカラコンパイルを行い，ベクトル化版の方はスカラコンパイル対象ルーチンを除きベクトルコンパイルを行った．

  オリジナル版のコンパイルオプション： -Oe,-E -Wv,-sc
  ベクトル化版のコンパイルオプション：
  -Oe,-E　　(ベクトルコンパイル対象ルーチン)
  -Oe,-E -Wv,-sc　(スカラコンパイル対象ルーチン)

本確認作業で対象とした出力結果は，テストデータ A5VPP3，COLOMN34，RPVVCS1，SP，SP2，sp61.txt(50)，及び sp61.txt(100) において，出力機番 6 に出力されたデータである．ここで，( ) 内はテストデータ中の L カードに指定した繰り返しステップ数である．オリジナル版とベクトル化版との出力結果を diff コマンドを利用して比較したところ，いずれの場合も，一桁の違いもなく一致した．

以上のことから，出力結果に影響する変更誤りは見出されず，当初の違いは，計算環境によるものと判断した．

4.2.3.2　ベクトル化効果

オリジナル版，及びベクトル化版の実行時間を測定した．ここで，測定の対象としたテストデータは「4.2.3.1 計算結果の評価」で利用したものと同じである．測定値を Table 4.1 に示す．Table 4.1 の測定値をもとに算出したオリジナル版とベクトル化版のベクトル化率を Table 4.2 に示す．また各データに対するベクトル化版の SAMPLER の結果を Fig. 4.9 に示す．ベクトル化前の SAMPLER の結果とベクトル化後の SAMPLER の結果から計算コストがおよそどの程度削減できたか判る．Fig. 4.1 と Fig. 4.9 により求めた SAMPLER の結果の推移及びコストダウン率を Table 4.3 に示す．Table 4.1 ～ 4.3 からも判るように，テストデータ RPVVCS1 の場合を除き，ある程度の効果が得られている．実行時間については，倍率にして 5.84 ～ 9.81 倍の効果を得ることができ，ベクトル化率については，3.8 ～ 21.2% から 88.0 ～ 92.6% へと向上させることができた．また主要 3 ルーチン (BKINDX，SOLWAV，SOLWVX) 自身のコストダウン率については，91.3 ～ 94.9% の効果を得た．一方テストデータ RPVVCS1 の場合は，実行時間：2.23 倍，ベクトル化率：65%，主要 3 ルーチン自身のコストダウン率：80.7% と他のデータと比較して良い結果が得られなかった．

RPVVCS1 の場合，Fig. 4.1 からオリジナル版における主要 3 ルーチンのコスト比率が他と比較して小さいことが判り，Table 4.3 からそのコストダウン率も小さいということが判る．従って倍率も他に比較して小さくなっている．つまり，コストダウン対象ルーチンの全体の中でのコスト比率を α，そのコストダウン率を δ とおくと，倍率は式 (4.15) のように表される．

$$倍率 = 1/(1-\alpha \times \delta) \tag{4.15}$$

$\alpha \times \delta$ の値が小さい程倍率は小さくなる．各テストデータにおける主要3ルーチンのコスト比率，コストダウン率を Table 4.4 に示す．RPVVCS1 が最も小さい値を示している．

BKINDX，SOLWVX ルーチンを含む解析部の処理構成を Fig. 4.10 に示す．BKINDX と SOLWVX の処理時間の和 T_{BS} は，Fig. 4.10 から式 (4.16) のように表せる．

$$T_{BS} = t_{BS} \times ne \times m \tag{4.16}$$

ここで，

t_{BS} ： BKINDX と SOLWVX の1処理平均時間の和
ne ： 要素数
m ： 総繰り返し数

コスト比率 α は m が大きくなると $t_{BS} \times ne/(定数 + t_{BS} \times ne)$ に近づき，t_{BS}，ne の値に影響される．大きくなれば，α も大きくなる．t_{BS} の値は BKINDX ルーチンと SOLWVX ルーチンの主要ループのベクトル長に影響される．このベクトル長は，NCOLM の値つまりウェイブ・フロント法のバンド幅に影響される．NCOLM の値が大きくなれば，ベクトル長が大きくなり，t_{BS} の値そして α が大きくなる．一方，NCOLM の値，つまり，ベクトル長が大きくなればコストダウン率も大きくなる．テストデータ RPVVCS1，SP，SP2，sp61.txt(繰り返し回数：100) に関する，主要ループのベクトル長，及び要素数を Table 4.5 に示す．RPVVCS1 が最も小さい値を示している．

以上から判ることは，ベクトル化の効果はバンド幅に依存するということであり，データの作りに依存するということである．だだし，ベクトル化の効果が小さいということは実行時間が長くなるということではない．テストモデルが同じなら，バンド幅が極端に短くならなければ (例えば 10 未満)，バンド幅が短い程実行時間は短い．つまり，ある DO ループがあって，そのベクトル長を n1，n2，それぞれの実行時間を t1，t2 とすると，式 (4.17) が成立する．

$$n1 < n2 \quad ならば \quad t1 < t2 \tag{4.17}$$

処理する要素の順番を検討してバンド幅をできるだけ短くすることには，実行時間を短縮するという意味がある．

ベクトル化の効果を判断するパラメタとして，倍率といったものの他に実際に使用されるテストモデル，ないしはそれに近いモデルにおける実行時間があげられる．実用に耐えられないのであれば，あまり意味をなさないからである．今回の場合，ユーザが当初提示された繰り返し回数 3000 回の実行時間を目安にした．

実際に，繰り返し回数を大きくして実行ができたのは，sp61.txt のみであった．他は定常解析のデータであり，50 回にも満たないうちに繰り返しループが収束してしまった．そこで，繰り返しの1処理時間を実測と計算により算出し 3000 回実施時の実行時間を予測した．対象としたテストデータは RPVVCS1，SP，SP2，sp61.txt である．実行予測時間を Table 4.6 に示す．sp61.txt については，3000 回の繰り返し実行が可能であったので実際に実行時間を測定した．実測値は，4143.76 秒であった．実行予測時間 (4224 秒) に近い値を示している．

従って，今回使用したテスト・データに近いモデルであれば，実行は 45 ～ 83 分で終了する．実用に耐えられないほどではないと判断する．

4.3 並列化調査

更に，高速化を進めるため，並列化の可能性について検討した．計算のコスト分布は，コスト比率の高いBKINDX，SOLWAV，SOLWVXの3ルーチンを含む部分に集中することが予想できる．その部分は要素マトリクス作成ルーチンHEAT1，またはHEAT1XのDO 1000ループである．HEAT1Xルーチンの場合の位置付けを，Fig. 4.10に示してある．この図からも判るように繰り返し回数が多くなると，解析部のコスト比率が大部分を占めることになる．DO 1000ループの解析部の中でのコスト比率を算出するため，サービスルーチンCLOCKMを利用した．DO 1000ループのコスト比率は約75～85%を占めており，効果を考慮すると，並列化の対象としてこの部分以外は考えにくい．

HEAT1ルーチンとHEAT1Xルーチンはテストデータの中に要素HPIPE2を含むか含まないかで，どちらか一方が選択される．要素HPIPE2を含む場合HEAT1Xが選択され，含まない場合HEAT1が選択される．処理構造が同形のため並列化の調査にはHEAT1Xルーチンを対象にした．

HEAT1XルーチンのDO 1000ループの処理フローをFig. 4.11に示す．ここは，本ルーチンの心臓部であるウェイブ・フロント法の処理部分である．ウェイブ・フロント法のポイントは，アセンブルエリアをできるだけ少なく，かつ有効利用することであり，不要となったアセンブルエリアを再使用している．Fig. 4.11において，前段の要素剛性マトリクスの作成部分は，各要素対応の要素剛性マトリクスを作成しており，要素間で処理が独立しているため，並列化が容易である．しかし，後段のコストの大部分を占めるBKINDXルーチン～SOLWVXルーチンの部分については，並列化することは難しい．この範囲は，自由度テーブルをキーデータとして，前段で作成した要素単位の要素剛性マトリクスを，それまでの要素に対して累積的に組み立てられてきた要素剛性の全体マトリクスの中に組み込み，全体マトリクスの中で三角分解できるところがあれば三角分解するところである．

自由度テーブルは，その要素として自由度番号を持ち，BKINDXルーチンにおいて新たな自由度番号群が取り込まれ，これにより更新され，累積されていく．各エントリの番号，つまり自由度テーブルの添字の値が全体マトリクスの行番号及び列番号に相当し，0以外の値が設定されているエントリに対応する行，列のエリアが全体マトリクスの中で有効な所である．BKINDXルーチンの中では，自由度番号を小さい順に並べるインデックスのテーブルが作成され，全体マトリクスの中で要素剛性マトリクスを組み立てる位置が選択される．この組み立てエリアの中で0クリアが必要な新規または再利用の箇所も選択される．また，自由度テーブルは，SOLWVXルーチンの中で，自由度番号が小さい順の消去自由度数分のエントリ，つまり不要となったエントリが消去される．これは，対応エリアを空きにして再使用の対象にすることである．

以上のように，自由度テーブル及び要素剛性マトリクスを組み立てる全体マトリクスは，要素間で累積的に利用されており，このため要素間で処理の独立性が保てず，並列化が難しくなっている．

4.4 AP3000 へのインストール

(1)AP3000 版ソースコード

AP3000 へのインストールには，基本的には，ベクトル化版のソースコードを使用した．ベクトル化の過程で計算量の削減が図られているものを含むからである．ただし，BKINDX ルーチンにについては逆である．計算コストの比率が最も高い二重ループ DO 3010・3020 のソーティングロジックとして，逐次的には計算ステップ数が多くなるロジックを，ベクトル化効果を引き出すために採用している．AP3000 はスカラ計算機なので，逐次処理上の計算ステップ数の少ない方がより高速であるため，この部分を変更した．変更内容を Fig. 4.12 に示す．IF 文の計算量を比較すると，変更前が $NCOLM^2$ であり，変更後は NCOLM × (NCOLM+1)/2 である．従って，AP3000 版のソースコードはベクトル化版ソースコードのうち BKINDX ルーチンのみこの変更後のものと取り替えたものである．

(2) 計算結果の評価

AP3000 版の評価については，その出力結果をオリジナル版のものと比較することにより行った．比較対象の出力結果は VPP の場合と同じく，出力機番 6 に出力されたものである．

オリジナル版，AP3000 版共に最適化のオプションとして「演算評価方法の変更」，「不変式の先行評価」を指定してコンパイルを行った．AP3000 上で双方を実行したところ，テストデータ A5VPP3，COLOMN34，RPVVCS1，SP では，出力結果が一致した．しかし SP2，sp61.txt では，最後の 1 ～ 2 桁が異なる程度の違いを生じていた．

この違いを引き起こす原因としては，プログラムの変更誤りか最適化に伴う副作用が考えられる．そこで，オリジナル版，AP3000 版共に上記の 2 つの最適化オプションを抑止してコンパイルを行った．双方を実行したところ，いずれのテストデータにおいても出力結果は一致した．以上から出力結果の違いは，最適化の影響が現れたもので AP3000 版の問題ではないといえる．

(3) 実行時間

オリジナル版及び AP3000 版について，AP3000 上での実行時間を測定した．測定した実行時間は CPU タイムと実時間である．この測定値を Table 4.7 に示す．Table 4.7 には参考として，オリジナル版及びベクトル化版の VPP 上での実行時間の測定値も載せてある．実時間は，その時の混み具合により大きく影響されるデータであり，実行の度に値が変わる．従って，ここでの値は 1 例にしか過ぎない．

Table 4.7 から判ることがいくつかある．AP3000 版とオリジナル版を比較した場合，いずれのテストケースにおいても，AP3000 版の CPU タイムの方が短い．計算ステップの削減効果が現れていることを示している．CPU タイムに関しては，AP3000 上の AP3000 版と VPP 上のベクトル化版を比較した場合，RPVVCS1 を除きベクトル化版の方が短い (速い) ことが判る．しかし，RPVVCS1 のようなベクトル化効果の小さいテストケースの場合，AP3000 上で実行した方がより良好な結果が得られている．このことは，VPP 上のオリジナル版の CPU タイム

との比較に顕著に現れている．実時間に関してはVPP上での実行の場合，I/Oファイルをデータ領域またはワークV領域に設定した方がホーム領域に設定するよりも短くすることができる．また，混雑の程度に関係することであるが，ワークV領域を使用したとしても，AP3000上での実行の方が良好な結果を得ている場合があることも判る．

以上のことから，CPUタイムに関しては，大部分のところでVPPにおけるベクトル化版の方が短い．しかし実時間まで考慮すると，AP3000上での実行の方が短いテストケースが増えてくる．特にRPVVCS1のようなベクトル化効果の小さいテストケースの場合には，AP3000上での実行の方がCPUタイム，実時間共に短くなることが判る．

4.5 まとめ

今回のベクトル化チューニング作業では6種類のテストデータを利用した．これらのデータを利用したテスト・モデルの特徴は，BKINDX，SOLWAV，SOLWVXの3ルーチンに計算コストが集中していることであった．従って，チューニング作業の大半は，この3ルーチンのコストダウンを行うことであった．結果として，80.7～94.9%のコストダウン率を得ている (Table 4.3)．全体のチューニング効果としては，テストモデルが同じであればバンド幅が長いほど大きな効果が得られるということが判った．逆に，テストデータRPVVCS1のようにバンド幅の小さいデータではチューニングの効果が小さいものになる (Table 4.1 ～4.3)．これは，実行時間が長くなるということではない．繰り返し回数を増加していくと，テストモデルが同じならバンド幅の短いテストデータほど実行時間は短くなると予想できるからである．

ベクトル化チューニングの対象としたルーチンは，ベクトルコンパイルのみのルーチンを含めて15ルーチンである．残りの149ルーチンはオリジナルルーチンをそのまま利用してスカラコンパイルしており，初期設定部のルーチンか，解析部のルーチンであってもベクトル化効果がが期待できないものである．初期設定部のルーチンは繰り返し回数が増加するにしたがいそのコスト比率は下がっていく．従って，ベクトル化チューニングは概ね終了したものと考える．

更に高速化するには並列化を行う必要があるが，並列化調査の結果，有効な並列性を見出すことはできなかった．幸い，今回のテスト・モデルの範囲では，現在の計算機環境で実行できないほどのものではなかったため並列化するまでもなかった．今後，更に長時間のテスト・モデルを対象とする場合には，処理方式にまで立ち入って並列化の検討を行う必要がある．

計算機の利用にあたっては，システムトラブル等何らかの理由により利用できない場合もあれば，混雑によりジョブがなかなか受け付けてもらえず，受け付けられてもターンアラウンドタイムが長くなる場合もある．こうした場合，複数の実行環境を有していれば，こうした不自由さの緩和がある程度期待できる．この観点から，SSPHEATをAP3000にインストールしたが，例えば，RPVVCS1のようなベクトル化効果を発揮しにくいテストモデルに効果的であることが判った (Table 4.7)．テストモデルの適性，混雑度等を考慮して実行環境を選択することで効率的な計算実行が可能となる．

Table 4.1　Execution times on VPP500.

(CPU, VUの単位：分:秒.xx)

No.	入力データ		A5VPP3	COLOMN34	RPVVCS1	SP	SP2	sp61.txt①*5	sp61.txt②*5
1	オリジナル 版	スカラ*1 CPU	2:09.08	1:17.76	44.61	1:25.53	1:38.26	8:05.73	15:53.56
2		ベクトル*2 CPU	2:07.67	1:16.80	41.32	1:12.45	1:21.78	6:58.84	13:42.11
		VU	3.50	0.67	3.93	2.04	4.34	21.27	42.02
3	ベクトル化版	スカラ*3 CPU	1:51.18	1:01.10	45.32	1:23.85	1:31.61	7:43.93	15:11.39
4		ベクトル*4 CPU	21.70	7.93	20.00	10.25	16.83	1:15.72	2:24.69
		VU	8.34	3.39	4.13	3.91	5.83	25.17	49.65
倍率	No.1 ／No.2 (CPU)		1.01	1.01	1.08	1.18	1.20	1.16	1.16
	No.1 ／No.4 (CPU)		5.95	9.81	2.23	8.34	5.84	6.41	6.59

＊1：オリジナル版の全ルーチンをスカラ・コンパイル（最適化オプション：−0e）
＊2：オリジナル版の全ルーチンをベクトル・コンパイル（最適化オプション：−0e）
＊3：ベクトル化版の全ルーチンをスカラ・コンパイル（最適化オプション：−0e）
＊4：ベクトル化版の中でベクトル・コンパイル対象のルーチンのみベクトル・コンパイル、
その他はスカラ・コンパイル（最適化オプション：−0e）
＊5：sp61.txt①，sp61.txt②は，入力データsp61.txt中の Lカードに繰り返しステップ数として
50,100を指定した場合のデータである。

Table 4.2　Vectorization ratio.

No.	入力データ	A5VPP3	COLOMN34	RPVVCS1	SP	SP2	sp61.txt①	sp61.txt②
1	オリジナル版	3.8 %	2.1 %	16.2 %	17.7 %	21.2 %	18.2 %	18.2 %
2	ベクトル化版	88.0 %	92.6 %	65.0 %	92.4 %	88.0 %	89.1 %	89.6 %

(1)オリジナル版、ベクトル化版のベクトル化率は以下の式により求めた。

ベクトル化率(%) ＝ $\{t_{sc}-(t_{vc}-t_{vu})\}/t_{sc}\times 100$

ここで、

t_{sc}：スカラ・コンパイル版を実行した時のCPU タイム
　オリジナル版の場合：Table 4.1 のNo.1 の値
　ベクトル化版の場合：Table 4.1 のNo.3 の値

t_{vc}：ベクトル・コンパイル版を実行した時のCPU タイム
　オリジナル版の場合：Table 4.1 のNo.2 の値
　ベクトル化版の場合：Table 4.1 のNo.4 の値

t_{vu}：ベクトル・コンパイル版を実行した時のVUタイム
　オリジナル版の場合：Table 4.1 のNo.2 の値
　ベクトル化版の場合：Table 4.1 のNo.4 の値

(2)sp61.txt①, sp61.txt②：Table 4.1 の＊5と同様

Table 4.3 Computational costs and cost down ratio.

No.	項目 ＼ 入力データ		A5VPP3	COLOMN34	RPVVCS1	SP	SP2	sp61.txt①*4	sp61.txt②*4
1	BKINDXルーチン	オリジナル 版*1	1927	1172	768	3569	2512	19016	37629
		ベクトル化版*2	287	97	181	189	194	1281	2520
		コストダウン率*3	85.1%(12.5%)	91.7%(13.6%)	76.4%(13.8%)	94.7%(45.3%)	92.3%(27.5%)	93.3%(41.0%)	93.3%(41.4%)
2	SOLWAVルーチン	オリジナル 版*1	9221	6098	—	—	—	—	—
		ベクトル化版*2	569	272	—	—	—	—	—
		コストダウン率*3	93.8%(65.8%)	95.5%(73.9%)	—	—	—	—	—
3	SOLWVXルーチン	オリジナル 版*1	—	—	1561	3137	4550	17410	34436
		ベクトル化版*2	—	—	269	215	421	1383	2838
		コストダウン率*3	—	—	82.8%(30.4%)	93.1%(39.2%)	90.7%(49.0%)	92.1%(37.1%)	91.8%(37.2%)
4	その他	オリジナル 版*1	2002	617	1923	748	1356	6801	12777
		ベクトル化版*2	1486	537	1845	785	1292	5198	9607
		コストダウン率*3	25.8%(3.9%)	13.0%(1.0%)	4.1%(1.8%)	-4.9%(-0.5%)	4.7%(0.8%)	23.6%(3.7%)	24.8%(3.7%)
No.1～3のコストダウン率*3			92.3%(78.3%)	94.9%(87.5%)	80.7%(44.2%)	94.0%(84.5%)	91.3%(76.6%)	92.7%(78.1%)	92.6%(78.6%)

＊1オリジナル版の欄 ：Table 4.1 のNo.2 （オリジナル版（ベクトル））に対するSAMPLER の出力結果
＊2ベクトル化版の欄 ：Table 4.1 のNo.4 （ベクトル化版）に対するSAMPLER の出力結果
（＊1、＊2共に、SAMPLER のインターバル値は100ms である）
＊3コストダウン率の欄：項目毎のコストダウン率（全体のコストダウン率）の並び
＊4 sp61.txt①, sp61.txt②：Table 4.1 の＊5と同様

Table 4.4　Computational cost ratio and cost down ratio of 3 routines (BKINDX, SOLWAV, AND SOLWVX).

入力データ	RPVVCS1	A5VPP3	COLOMN34	SP	SP2	sp61.txt *1
コスト比率：α	0.548	0.848	0.922	0.900	0.839	0.850
コストダウン率：δ	0.807	0.923	0.949	0.940	0.913	0.926
$\alpha \times \delta$	0.436	0.783	0.875	0.846	0.766	0.787

＊1：sp61.txt(100) の場合のデータ

Table 4.5 Vector length of main loops and number of elements in test cases (RPVVCS1, SP, SP2, and sp61.txt).

入力データ	RPVVCS1	SP	SP2	sp61.txt *1
BKINDX DO 3010 のベクトル長	56	253	142	161
SOLWVX DO 1600 のベクトル長	26	101	65	86
要素数	2628	6190	3123	3170

＊1：sp61.txt(100) の場合のデータ

Table 4.6 Execution time in test cases (RPVVCS1, SP, SP2, and sp61.txt).

入力データ	RPVVCS1	SP	SP2	sp61.txt
総繰り返し回数 N	14	4	10	101 *1
繰り返し部の実行時間（sec） T_{iter}	12.792	6.638	13.366	142.194
繰り返し部の平均実行時間（sec） $T_{iter} \div N$	0.914	1.660	1.337	1.408
繰り返し回数：3000回の実行時間（予測）（sec） $(T_{iter} \div N) \times 3000$	2742	4980	4011	4224

＊1：sp61.txt(100) の場合のデータ

Table 4.7　Execution time on AP3000.　　(CPU, ELAPS の単位：分:秒.xx)

No.		入力データ		A5VPP3	COLOMN34	RPVVCS1	SP	SP2	sp61.txt①*5	sp61.txt②*5
1	AP 3000 実行	ｵﾘｼﾞﾅﾙ 版	CPU	40.99	26.83	18.39	39.52	46.33	3:40.19	7:09.65
			ELAPS	48.26	31.29	26.86	44.43	52.28	3:55.11	7:30.25
2		AP3000版	CPU	31.33	20.19	17.29	36.31	42.03	3:22.79	6:46.30
			ELAPS	40.34	22.74	24.79	40.58	52.99	3:38.61	7:03.18
3	VPP 実行	ｵﾘｼﾞﾅﾙ 版*1	CPU	2:09.08	1:17.76	44.61	1:25.53	1:38.26	8:05.73	15:53.56
			ELAPS	3:13.61	2:47.78	1:56.06	2:08.03	2:24.95	11:47.43	22:27.67
4		ﾍﾞｸﾄﾙ化版*2 ﾜｰｸ V *3	CPU	21.70	7.93	20.00	10.25	16.83	1:15.72	2:24.69
			ELAPS	59.49	27.17	1:03.42	23.03	38.74	1:57.44	3:08.32
5		ﾎｰﾑ 領域*4	CPU	21.69	7.92	19.95	10.22	16.82	1:15.62	2:24.51
			ELAPS	2:48.45	1:00.57	3:01.08	1:05.56	1:53.28	8:09.38	10:42.42
倍率	対 VPP ｵﾘｼﾞﾅﾙ 版	No.3/No.2 (CPU)		4.12	3.85	2.58	2.36	2.34	2.42	2.35
		No.3/No.4 (CPU)		5.95	9.81	2.23	8.34	5.84	6.41	6.59
	対 AP3000版	No.2/No.4 (ELAPS)		0.68	0.84	0.39	1.76	1.37	1.86	2.25
		No.2/No.5 (ELAPS)		0.24	0.38	0.14	0.62	0.47	0.45	0.66

＊1：オリジナル版をスカラ・コンパイル（最適化オプション：−0e）
＊2：ベクトル化チューニング版をベクトル・コンパイル（最適化オプション：−0e）
＊3：I/O ファイルをワークV領域に設定した場合。
＊4：I/O ファイルをホーム領域に設定した場合。他の条件はNo.4と同様。
＊5：sp61.txt①, sp61.txt②は，Table 4.1 の＊5と同様

入力データ： A5VPP3

Synthesis Information

Count	Percent	VL	Name
9221	70.1	115	solwav
1927	14.7	-	bkindx
282	2.1	3	srmr
207	1.6	-	skhr4
207	1.6	-	bkcler
	(計90.1%)		
13150		38	TOTAL

入力データ： COLOMN34

Synthesis Information

Count	Percent	VL	Name
6098	77.3	85	solwav
1172	14.9	-	bkindx
145	1.8	-	incard
58	0.7	2	srmr
54	0.7	-	bkcler
	(計95.4%)		
7887		12	TOTAL

入力データ： RPVVCS1

Synthesis Information

Count	Percent	VL	Name
1561	36.7	29	solwvx
768	18.1	-	bkindx
331	7.8	-	incard
187	4.4	3	srmr
164	3.9	-	disph
	(計70.9%)		
4252		34	TOTAL

入力データ： SP

Synthesis Information

Count	Percent	VL	Name
3569	47.9	-	bkindx
3137	42.1	117	solwvx
235	3.2	-	incard
52	0.7	4	srmr
46	0.6	-	ecard
	(計94.5%)		
7454		112	TOTAL

入力データ： SP2

Synthesis Information

Count	Percent	VL	Name
4550	54.1	70	solwvx
2512	29.8	-	bkindx
215	2.6	-	incard
140	1.7	3	srmr
103	1.2	-	skhr4
	(計89.4%)		
8418		64	TOTAL

入力データ： sp61.txt

Synthesis Information (iteration=50)

Count	Percent	VL	Nam
19016	44.0	-	bkindx
17410	40.3	97	solwvx
897	2.1	3	srmr
632	1.5	4	srmt
592	1.4	-	tris
	(計89.3%)		
43227		59	TOTAL

Synthesis Information (iteration=100)

Count	Percent	VL	Nam
37629	44.4	-	bkindx
34436	40.6	94	solwvx
1698	2.0	3	srmr
1313	1.5	-	tris
1229	1.4	3	srmt
	(計89.9%)		
84842		55	TOTAL

Fig. 4.1 Computational costs distribution in original code (Top 5).

* A5VPP3 *　　－BKINDXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00000529-00000536	-----	if		2929591	.1169E10	81.3	.1169E10	81.3	24	0.0	1.0- 1.0
00000446-00000452	200	do	S	139688	.1529E09	10.6	.1529E09	10.6	73	0.0	1.0- 1.0
00000524-00000544	3000	do		35587	77772396	5.4	77772396	5.4	86	0.0	1.0- 1.0
00000546-00000550	4000	do	S	35587	32836180	2.3	32836180	2.3	83	0.0	1.0- 1.0
00000441-00000511	2000	do		35587	3091832	0.2	3091832	0.2	4	0.0	1.0- 1.0

* COLOMN34 *　　－BKINDXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00000529-00000536	-----	if		878460	.7195E09	90.1	.7195E09	90.1	49	0.0	1.0- 1.0
00000446-00000452	200	do	S	25080	44193700	5.5	44193700	5.5	120	0.0	1.0- 1.0
00000524-00000544	3000	do		6270	24651270	3.1	24651270	3.1	171	0.0	1.0- 1.0
00000546-00000550	4000	do	S	6270	9775650	1.2	9775650	1.2	141	0.0	1.0- 1.0
00000441-00000511	2000	do		6270	562360	0.1	562360	0.1	4	0.0	1.0- 1.0

* RPVVCS1 *　　－BKINDXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00000529-00000536	-----	if		1789354	.4004E09	74.5	.4004E09	74.5	14	0.0	1.0- 1.0
00000446-00000452	200	do	S	116228	62395578	11.6	62395578	11.6	37	0.0	1.0- 1.0
00000524-00000544	3000	do		34832	50759800	9.4	50759800	9.4	63	0.0	1.0- 1.0
00000546-00000550	4000	do	S	34832	20222454	3.8	20222454	3.8	52	0.0	1.0- 1.0
00000441-00000511	2000	do		34832	2557044	0.5	2557044	0.5	3	0.0	1.0- 1.0

* SP *　　－BKINDXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00000529-00000536	-----	if		1750572	.2397E10	93.7	.2397E10	93.7	81	0.0	1.0- 1.0
00000446-00000452	200	do	S	32096	93466472	3.7	93466472	3.7	195	0.0	1.0- 1.0
00000524-00000544	3000	do		8540	47004656	1.8	47004656	1.8	223	0.0	1.0- 1.0
00000546-00000550	4000	do	S	8540	19400032	0.8	19400032	0.8	206	0.0	1.0- 1.0
00000441-00000511	2000	do		8540	712624	0.0	712624	0.0	4	0.0	1.0- 1.0

* SP2 *　　－BKINDXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00000529-00000536	-----	if		2188200	.1589E10	88.7	.1589E10	88.7	44	0.0	1.0- 1.0
00000446-00000452	200	do	S	74920	.1180E09	6.6	.1180E09	6.6	105	0.0	1.0- 1.0
00000524-00000544	3000	do		19260	57864300	3.2	57864300	3.2	118	0.0	1.0- 1.0
00000546-00000550	4000	do	S	19260	24399600	1.4	24399600	1.4	115	0.0	1.0- 1.0
00000441-00000511	2000	do		19260	1660680	0.1	1660680	0.1	4	0.0	1.0- 1.0

* sp61.txt(50) *　　－BKINDXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00000529-00000536	-----	if		13048860	.1205E11	90.9	.1205E11	90.9	55	0.0	1.0- 1.0
00000446-00000452	200	do	S	385764	.7070E09	5.3	.7070E09	5.3	123	0.0	1.0- 1.0
00000524-00000544	3000	do		100062	.3445E09	2.6	.3445E09	2.6	135	0.0	1.0- 1.0
00000546-00000550	4000	do	S	100062	.1452E09	1.1	.1452E09	1.1	131	0.0	1.0- 1.0
00000441-00000511	2000	do		100062	8549232	0.1	8549232	0.1	4	0.0	1.0- 1.0

Fig. 4.2 Dynamic behavior of original code (in BKINDX subroutine) (1/3).

＊ A5VPP3 ＊　　－SOLWAVルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00010654-00010663	1500	do	S	2986382	.4382E10	97.3	.4382E10	94.8	58	0.0	1.0- 1.0
00010648-00010664	2000	do	S	36119	.1045E09	2.3	.1045E09	2.3	83	0.0	1.0- 1.0
00010622-00010631	400	do	V	36537	8952025	0.2	84584010	1.8	83	94.2	8.5- 10.5
00010615-00010673	4000	do		32851	2765640	0.1	2765640	0.1	1	0.0	1.0- 1.0
00010607-00010609	100	do	V	32851	1260936	0.0	24588261	0.5	83	100.0	15.6- 24.7

＊ COLOMN34 ＊　　－SOLWAVルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00010654-00010663	1500	do	S	1021445	.2808E10	98.6	.2808E10	97.4	105	0.0	1.0- 1.0
00010648-00010664	2000	do	S	7150	35750575	1.3	35750575	1.2	143	0.0	1.0- 1.0
00010622-00010631	400	do	V	7270	1870595	0.1	23813875	0.8	142	95.6	10.8- 14.1
00010615-00010673	4000	do		5015	549310	0.0	549310	0.0	1	0.0	1.0- 1.0
00010644-00010644	1000	do	V	7150	299325	0.0	8171560	0.3	143	100.0	19.5- 35.1

Fig. 4.2 Dynamic behavior of original code (in SOLWAV subroutine) (2/3).

＊ RPVVCS1 ＊　　－SOLWVXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00010770-00010773	1600	do	S	944674	.7700E09	88.5	.7700E09	56.9	26	0.0	1.0- 1.0
00010755-00010774	2000	do		22232	46214039	5.3	46214039	3.4	51	0.0	1.0- 1.0
00010762-00010767	1500	do	V	944674	45378581	5.2	.4719E09	34.9	26	100.0	7.8- 13.0
00010742-00010746	700	do	V	22232	2910686	0.3	12424720	0.9	51	81.8	4.0- 4.5
00010718-00010723	400	do	V	26068	2026589	0.2	31437364	2.3	53	100.0	11.7- 19.4

＊ SP ＊　　－SOLWVXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00010770-00010773	1600	do	S	546178	.1730E10	94.9	.1730E10	59.2	102	0.0	1.0- 1.0
00010755-00010774	2000	do		8260	46865136	2.6	46865136	1.6	208	0.0	1.0- 1.0
00010762-00010767	1500	do	V	546178	38840623	2.1	.1060E10	36.3	102	100.0	19.5- 35.1
00010742-00010746	700	do	V	8260	3856987	0.2	18879432	0.6	208	81.8	4.8- 5.0
00010718-00010723	400	do	V	8300	1097152	0.1	39821064	1.4	209	100.0	28.5- 42.8

＊ SP2 ＊　　－SOLWVXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00010770-00010773	1600	do	S	1254861	.2565E10	93.9	.2565E10	59.3	66	0.0	1.0- 1.0
00010762-00010767	1500	do	V	1254861	80618069	3.0	.1572E10	36.3	66	100.0	15.6- 24.7
00010755-00010774	2000	do		19500	75129266	2.8	75129266	1.7	114	0.0	1.0- 1.0
00010742-00010746	700	do	V	19500	5192858	0.2	24519110	0.6	114	81.8	4.5- 4.9
00010718-00010723	400	do	V	19590	1905010	0.1	51871070	1.2	115	100.0	19.5- 35.0

＊ sp61.txt(50) ＊　　－SOLWVXルーチン－

isn	do-id	kind	v	ex-count	v-cost	%	s-cost	%	v-leng	v-rate	v-effect
00010770-00010773	1600	do	S	3649049	.9773E10	93.2	.9773E10	58.2	86	0.0	1.0- 1.0
00010755-00010774	2000	do		99501	.3457E09	3.3	.3457E09	2.1	132	0.0	1.0- 1.0
00010762-00010767	1500	do	V	3649049	.3072E09	2.9	.5990E10	35.7	86	100.0	15.6- 24.7
00010742-00010746	700	do	V	99501	30705842	0.3	.1450E09	0.9	132	81.8	4.5- 4.9
00010718-00010723	400	do	V	99909	11236059	0.1	.3061E09	1.8	133	100.0	19.5- 35.0

Fig. 4.2 Dynamic behavior of original code (in SOLWVX subroutine) (3/3).

```
isn       v o c--------1---------2---------3---------4---------5---------6---------7---------8

                    .   .   .

00000522              NCOLM =0
00000523              DO 3000 JJ=1,MXUSED
00000524              IF( MASTER(JJ).EQ.0 )         GO TO 3000
00000525          C...
00000526              IF( NCOLM.EQ.0 )          GO TO 2600
00000527              MVALUE =IABS (MASTER(JJ))
00000528              LOCATE =NCOLM
00000529          C                                                    ┐
00000530           2400 IF( KVALUE.LT.MVALUE )  GO TO 2700             │
00000531              NBKCOL(LOCATE+1) =NBKCOL(LOCATE)                 │  計算コストが集中
00000532              LOCATE =LOCATE-1                                 │    74.5～93.7%
00000533              IF( LOCATE.EQ.0 )         GO TO 2700             │
00000534              MD =NBKCOL(LOCATE)                               │
00000535              KVALUE =IABS (MASTER(MD))                        │
00000536              GO TO 2400                                       │
00000537          C....                                                ┘
00000538           2600 LOCATE =0
00000539           2700 LOCATE =LOCATE+1
00000540              NBKCOL(LOCATE) =JJ
00000541              NCOLM =NCOLM+1
00000542              MD =NBKCOL(NCOLM)
00000543              KVALUE =IABS (MASTER(MD))
00000544           3000 CONTINUE

                    .   .   .
```

Fig. 4.3 Loop 2400 of subroutine BKINDX in original code.

```
SUBROUTINE    BKINDX (MASTER, NBKCOL, NCOLM, MXCOLM, MXUSED, MBIG, NDEGS, KBIG, MSFLAG, MPCXX, FCTXX, MEQDEX, MCOUPL, COUPLE, MODES)
```

修正前

```
s             LOCATE =0
s             DO 200 KP=1, NSTATE
s               IF( MASTER(KP).EQ.0 )    GO TO 100
s               MVALUE =IABS (MASTER(KP))
s               IF( MVALUE.EQ.MD )       GO TO 500
                GO TO 200
s     100       IF( LOCATE.NE.0 )        GO TO 200
s               LOCATE =KP
s     200     CONTINUE
```

⇨

修正後

```
              DIMENSION       LOC(500), ncol(500)

              icnt = 0
v             do 180 KP=1, NSTATE
v               if( MASTER(KP) .ne. 0 ) then
v                 MVALUE =IABS (MASTER(KP))
v                 IF( MVALUE .EQ. MD )  then
v                   icnt = icnt +1
v                   LOC(icnt) = KP
v                 endif
v               endif
v     180     continue
              if( icnt .ge. 1 )   then
                KBIG(JJ) = LOC(1)
              else
                LOCATE =0
v               DO 200 KP=1, NSTATE
v                 if( MASTER(KP) .eq. 0 )  then
v                   icnt = icnt +1
v                   LOC(icnt) = KP
v                 endif
v     200       CONTINUE
                if( icnt .ge. 1 ) LOCATE = LOC(1)
```

Fig. 4.4　Modification of subroutine BKINDX (1／2).

修正前

```
          NCOLM =0
          DO 3000 JJ=1,MXUSED
            IF( MASTER(JJ).EQ.0 )          GO TO 3000
            IF( NCOLM.EQ.0 )           GO TO 2600
            MVALUE =IABS (MASTER(JJ))
            LOCATE =NCOLM
     2400   IF( KVALUE.LT.MVALUE )  GO TO 2700
            NBKCOL(LOCATE+1) =NBKCOL(LOCATE)
            LOCATE =LOCATE-1
            IF( LOCATE.EQ.0 )          GO TO 2700
            MD =NBKCOL(LOCATE)
            KVALUE =IABS (MASTER(MD))
            GO TO 2400
     2600   LOCATE =0
     2700   LOCATE =LOCATE+1
            NBKCOL(LOCATE) =JJ
            NCOLM =NCOLM+1
            MD =NBKCOL(NCOLM)
            KVALUE =IABS (MASTER(MD))
     3000 CONTINUE
```

⇨

修正後

```
          NCOLM = 0
  v       DO 3000 JJ=1,MXUSED
  v         IF( MASTER(JJ) .ne. 0 )   then
  v           NCOLM = NCOLM + 1
  v           ncol(NCOLM) = IABS(MASTER(JJ))
  v           iloc(NCOLM) = JJ
  v         endif
  v  3000 CONTINUE

  s2      DO 3010 I = 1, NCOLM
  v2        iorder = 0
  v2        iobj   = ncol(I)
  v2        do 3020 J = 1, NCOLM
  v2          if(iobj .ge. ncol(J))  then
  v2            iorder = iorder + 1
  v2          endif
  v2  3020  continue
  v2        NBKCOL(iorder) = iloc(I)
  v2  3010 CONTINUE
```

修正前

```
s         DO 4000 JJ=1,NCOLM
v           MD =NBKCOL(JJ)
m           IF( MASTER(MD).GT.0 )          GO TO 4000
s           MASTER(MD) =-MASTER(MD)
v           NBKCOL(JJ) =-MD
v    4000 CONTINUE
```

⇨

修正後

```
     *VOCL LOOP,NOVREC
  v       DO 4000 JJ=1,NCOLM
  v         MD =NBKCOL(JJ)
  v         IF( MASTER(MD) .le. 0 )   then
  v           MASTER(MD) =-MASTER(MD)
  v           NBKCOL(JJ) =-MD
  v         endif
  v  4000 CONTINUE
```

Fig. 4.4 Modification of subroutine BKINDX (2／2).

```
      SUBROUTINE    SOLWAV (BK,MASTER,NBKCOL,NCOLM,MXCOLM,NSIZE,NOEL,MODES,INVEC,BKVEC)
```

修正後

```
      DIMENSION     NS(1000)
```

```
v       do 10 I = 1,NCOLM
v         NS(I) = NSS(NBKCOL(I),MXCOLM)
v    10 continue
```

修正前

```
      DO 4000 JJ=1,NSIZE
        KJ =NBKCOL(JJ)
        K0 =NBKCOL(JJ)-1
        L0 =NSS(KJ,MXCOLM)
        K1 =KJ+1
v       DO 400 II=JJ,NCOLM
v         KI =NBKCOL(II)
v         IF( KI.LE.K0 )      GO TO 300
v         NLOW =KI-K0 + L0
v         BKVEC(II) =BK(NLOW)
v         GO TO 400
v  300    NLOW =KI-K1 + NSS(KI,MXCOLM)
v         BKVEC(II) =BK(NLOW)
v  400  CONTINUE
          .  .  .
```

⇨

修正後

```
      DO 4000 JJ=1,NSIZE
        KJ =NBKCOL(JJ)
        K0 =NBKCOL(JJ)-1
        L0 = NS(JJ)
        K1 =KJ+1
v       DO 400 II=JJ,NCOLM
v         KI =NBKCOL(II)
v         IF( KI.LE.K0 )  then
v           NLOW =K1-KI + NS(II)
v           BKVEC(II) =BK(NLOW)
v         else
v           NLOW =K1-K0 + L0
v           BKVEC(II) =BK(NLOW)
v         endif
v  400  CONTINUE
          .  .  .
```

Fig. 4.5 Modification of subroutine SOLWAV (1／2).

修正前

```
s           DO 2000 II=JX,NCOLM
v             KI =NBKCOL(II)
v             KO =NBKCOL(II)-1
v             LO =NSS(KI,MXCOLM)
v             KI =KI+1
v             SHFT =PIVOT*BKVEC(II)
s             DO 1500 LL=II,NCOLM
v               KL =NBKCOL(LL)
v               IF( KL.LE.KO )      GO TO 1300
v               NLOW =KL-KO + LO
m               BK(NLOW) =BK(NLOW) - SHFT*BKVEC(LL)
v               GO TO 1500
v  1300         NLOW =KI-KL + NSS(KL,MXCOLM)
m               BK(NLOW) =BK(NLOW) - SHFT*BKVEC(LL)
v  1500       CONTINUE
v  2000     CONTINUE
```

⇨

修正後

```
v               DO 2000 II=JX,NCOLM
v                 KI =NBKCOL(II)
v                 KO =NBKCOL(II)-1
v                 LO = NS(II)
v                 KI =KI+1
v                 SHFT =PIVOT*BKVEC(II)
   *VOCL LOOP,NOVREC
v                 DO 1500 LL=II,NCOLM
v                   KL =NBKCOL(LL)
v                   IF( KL .le. KO )  then
v                     NLOW =KI-KL + NS(LL)
v                   else
v                     NLOW =KL-KO + LO
v                   endif
v                   BK(NLOW) =BK(NLOW) - SHFT*BKVEC(LL)
v  1500           CONTINUE
v  2000         CONTINUE
```

Fig. 4.5 Modification of subroutine SOLWAV (2／2).

```
SUBROUTINE     SOLWVX  (BK, MASTER, NBKCOL, NCOLM, MXCOLM, NSIZE, NOEL, MODES, INVEC, BLVEC, BUVEC)
```

修正前

```
            DO 2000 II=JX,NCOLM
              IF( ABS(BUVEC(II)).GT.EPS )  GO TO 1000
              IF( ABS(BLVEC(II)).LT.EPS )  GO TO 2000
    1000      LJ =NBKCOL(II)
              BK(LJ,LJ) =BK(LJ,LJ) - BLVEC(II)*BUVEC(II)
              IF( II.EQ.NCOLM )            GO TO 3000
              KK =II+1
v             DO 1500 LL=KK,NCOLM
v               LI =NBKCOL(LL)
v               VEC1(LL)=BLVEC(II)*BUVEC(LL)
v               VEC2(LL)=BLVEC(LL)*BUVEC(II)
v   1500      CONTINUE
m             DO 1600 LL=KK,NCOLM
v               LI=NBKCOL(LL)
m5              BK(LJ,LI)=BK(LJ,LI)-VEC1(LL)
m               BK(LI,LJ)=BK(LI,LJ)-VEC2(LL)
v   1600      CONTINUE
    2000    CONTINUE
```

⇨

修正後

```
      DIMENSION  LII(1000),LLJ(1000)
```

修正後

```
    *VOCL LOOP,NOVREC
v                 do 2000 II=JX,NCOLM
v                   IF((ABS(BUVEC(II))  .gt.  EPS) .or.
          +           (ABS(BLVEC(II)) .ge.  EPS))       then
v                   IX = IX + 1
v                   LII(IX) =II
v                   LLJ(IX) =NBKCOL(II)
v                   BK(LLJ(IX),LLJ(IX)) =
          +         BK(LLJ(IX),LLJ(IX)) - BLVEC(II)*BUVEC(II)
v                 endif
v   2000        continue
    *VOCL LOOP,NOVREC
s                 do 2010 II=1,IX
v                   do 1500 LL=LII(II)+1,NCOLM
v                   LI =NBKCOL(LL)
v                   VEC1(LL)=BLVEC(LII(II))*BUVEC(LL)
v                   VEC2(LL)=BLVEC(LL)*BUVEC(LII(II))
v                   BK(LLJ(II),LI)=BK(LLJ(II),LI)-VEC1(LL)
v                   BK(LI,LLJ(II))=BK(LI,LLJ(II))-VEC2(LL)
v   1500          continue
s   2010        continue
```

Fig. 4.6 Modification of subroutine SOLWVX.

```
SUBROUTINE     BKCLER  (BK, NBKCOL, NCOLM, MXCOLM, MXUSED)
```

修正後

```
        DIMENSION       NS(1000), INBK(1000)
```

```
v       do 10 I = 1, NCOLM
v         INBK(I) = IABS(NBKCOL(I))
v         NS(I) = NSS(INBK(I), MXCOLM)
v    10 continue
```

修正前

```
s       DO 1000 KJ=1, NCOLM
v         IF( NBKCOL(KJ).GT.0 )        GO TO 1000
v         JJ =-NBKCOL(KJ)
v         K0 =JJ-1
v         K1 =JJ+1
v         L0 =NSS(JJ, MXCOLM)
s         DO 400 KI=1, NCOLM
v           II =IABS (NBKCOL(KI))
v           IF( II.LE.K0 )      GO TO 300
v           NLOW =II-K0 + L0
m           BK(NLOW) =0.0
v           GO TO 400
v   300     CONTINUE
v           NLOW =K1-II + NSS(II, MXCOLM)
m           BK(NLOW) =0.0
v   400   CONTINUE
m         NBKCOL(KJ) =IABS (NBKCOL(KJ))
v  1000 CONTINUE
```

⇨

修正後

```
s       DO 1000 KJ=1, NCOLM
v         IF( NBKCOL(KJ) .le. 0 )    then
v           JJ =-NBKCOL(KJ)
v           K0 = JJ-1
v           K1 = JJ+1
v           L0 =NS(KJ)
  *VOCL LOOP, NOVREC
v           DO 400 KI=1, NCOLM
v             II =INBK(KI)
v             IF( II .gt. K0 )    then
v               NLOW =II-K0 + L0
v             else
v               NLOW =K1-II + NS(KI)
v             endif
v             BK(NLOW) =0.0
v   400     CONTINUE
v           NBKCOL(KJ) =INBK(KJ)
v         endif
v  1000 CONTINUE
```

Fig. 4.7　Modification of subroutine BKCLER.

```
SUBROUTINE     FBR    (INVEC, BKVEC, NSIZE, MODES, FORCE, DISP, YVECT)
SUBROUTINE     FBRX   (INVEC, BKVEC, NSIZE, MODES, FORCE, DISP, YVECT)
```

修正前

```
s6          DO 700 I=2, LENG
v6          K =INVEC(I)
m6          YVECT(K) =YVECT(K) + YVECT(KD)*BKVEC(I)
v6     700  CONTINUE
```

⇨

修正後

```
      *VOCL LOOP, NOVREC
v             DO 700 I=2, LENG
v               K =INVEC(I)
v               YVECT(K) =YVECT(K) + YVECT(KD)*BKVEC(I)
v      700    CONTINUE
```

Fig. 4.8 Modification of subroutine FBR, and subroutine FBRX

入力データ： A5VPP3

Synthesis Information

Count	Percent	VL	Name
569	24.3	69	solwav
287	12.3	103	bkindx
284	12.1	-	incard
220	9.4	-	skhr4
80	3.4	-	srmr
	(計61.5%)		
2342		80	TOTAL

入力データ： COLOMN34

Synthesis Information

Count	Percent	VL	Name
272	30.0	133	solwav
214	23.6	-	incard
97	10.7	209	bkindx
62	6.8	-	disph
45	5.0	-	fcard
	(計76.1%)		
906		150	TOTAL

入力データ： RPVVCS1

Synthesis Information

Count	Percent	VL	Name
487	20.9	-	incard
269	11.6	28	solwvx
213	9.2	-	disph
181	7.8	54	bkindx
141	6.1	-	valm
	(計55.6%)		
2327		42	TOTAL

入力データ： SP

Synthesis Information

Count	Percent	VL	Name
346	29.1	-	incard
215	18.1	137	solwvx
189	15.9	252	bkindx
47	4.0	-	skhr4
45	3.8	-	ecard
	(計70.9%)		
1189		181	TOTAL

入力データ： SP2

Synthesis Information

Count	Percent	VL	Name
421	22.1	117	solwvx
317	16.6	-	incard
194	10.2	118	bkindx
125	6.6	-	skhr4
125	6.6	-	stresx
	(計62.1%)		
8418		125	TOTAL

入力データ： sp61.txt

Synthesis Information (iteration=50)

Count	Percent	VL	Nam
1383	17.6	126	solwvx
1281	16.3	165	bkindx
535	6.8	-	skhr4
482	6.1	-	trish
324	4.1	-	srmtr
	(計50.9%)		
7862		145	TOTAL

Synthesis Information (iteration=100)

Count	Percent	VL	Nam
2838	19.0	115	solwvx
2520	16.8	162	bkindx
1058	7.1	-	skhr4
937	6.3	-	trish
661	4.4	-	valm
	(計53.6%)		
14965		137	TOTAL

Fig. 4.9 Computational costs distribution in vectorized code (Top 5).

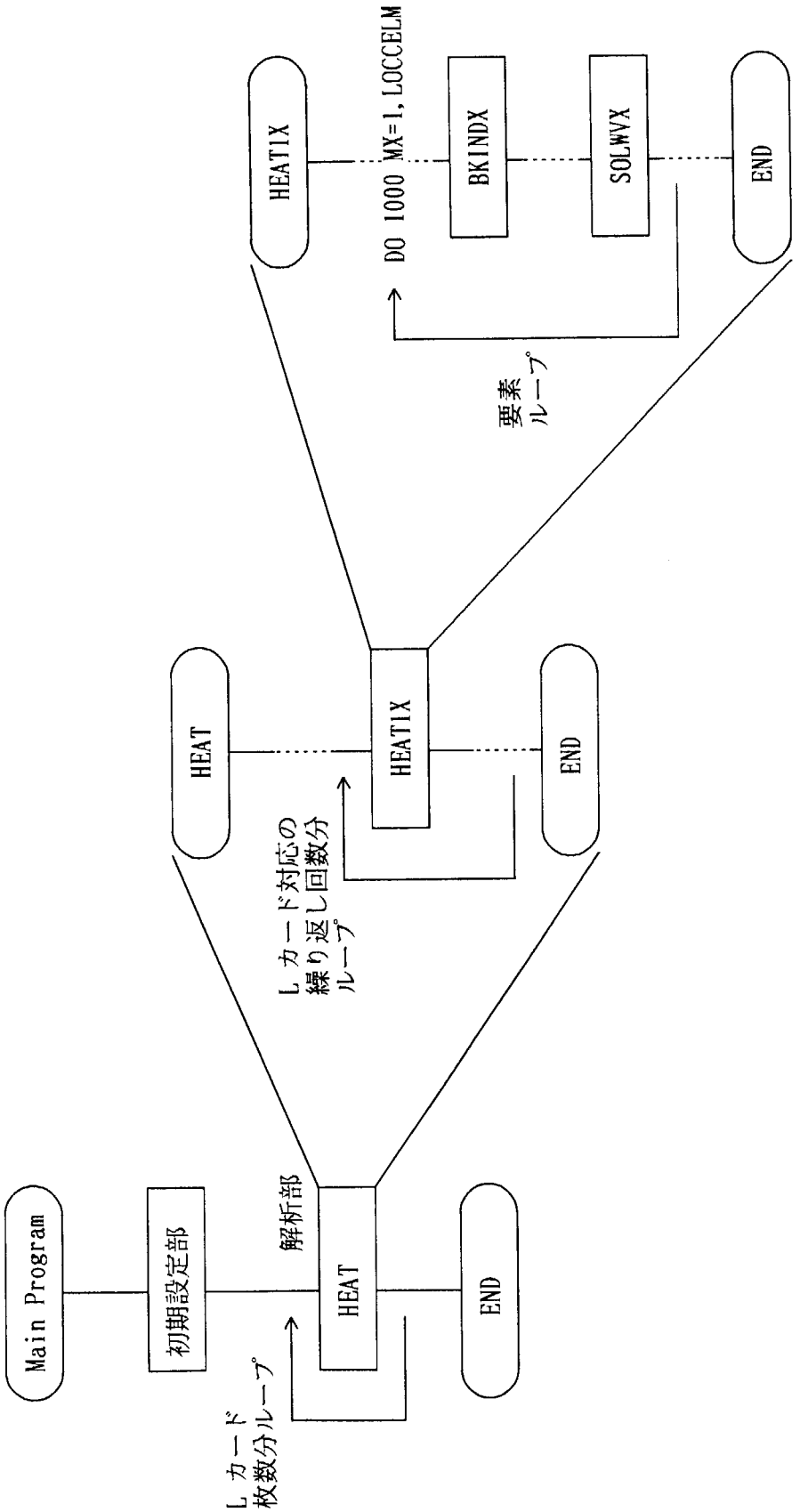

Fig. 4.10 Structure of analisys part in SSPHEAT code.

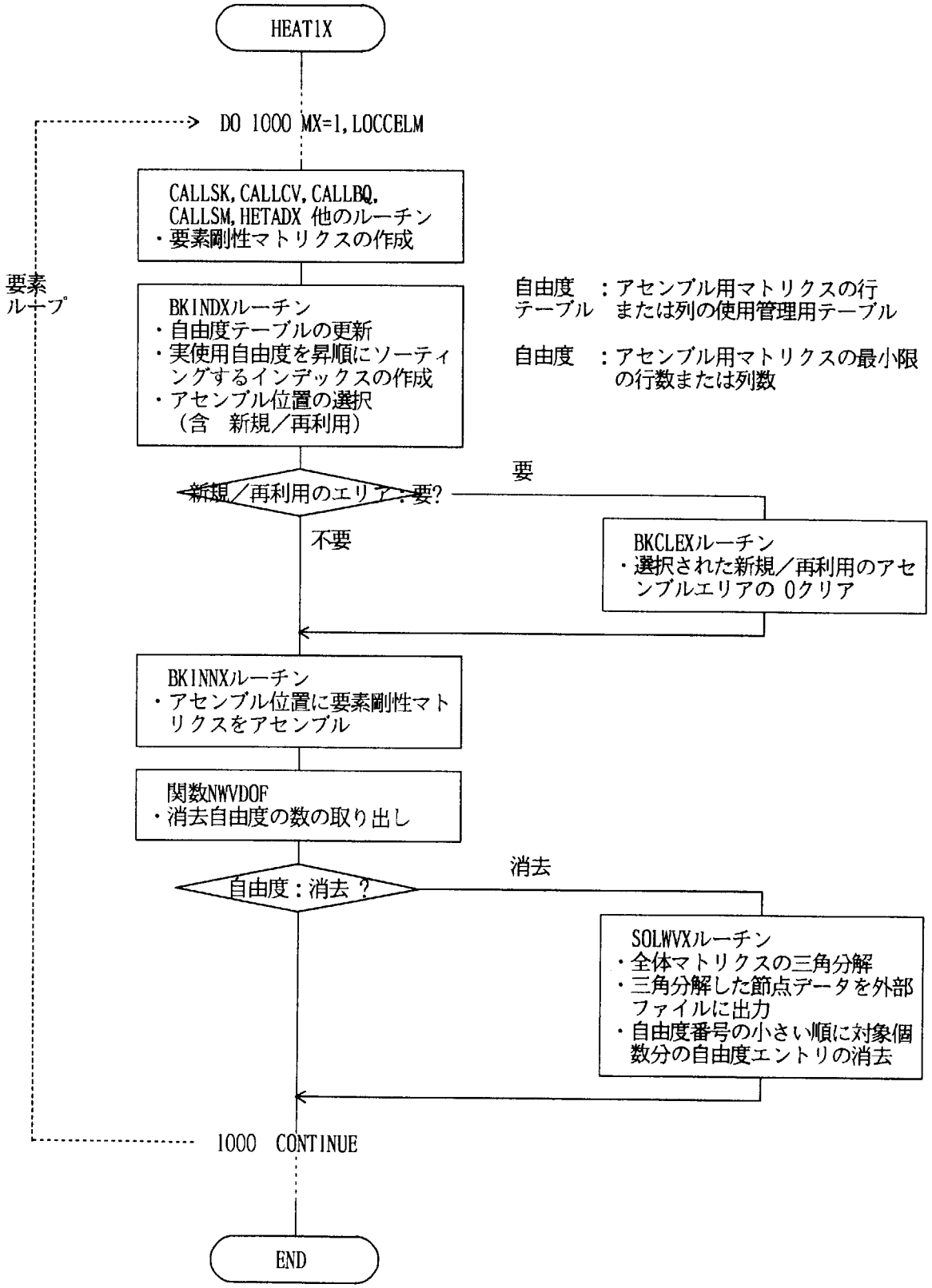

Fig. 4.11　Flow chart of DO 1000 loop in subroutine HEAT1X.

SUBROUTINE BKINDX (MASTER, NBKCOL, NCOLM, MXCOLM, MXUSED, MBIG, NDEGS, KBIG, MSFLAG, MPCXX, FCTXX, MEQDEX, MCOUPL, COUPLE, MODES)

変更前

```
      NCOLM = 0
      DO 3000 JJ=1,MXUSED
        IF( MASTER(JJ) .ne. 0 )   then
          NCOLM = NCOLM + 1
          ncol(NCOLM) = IABS(MASTER(JJ))
          iloc(NCOLM) = JJ
        endif
3000 CONTINUE

      DO 3010 I = 1, NCOLM
        iorder = 0
        iobj   = ncol(I)
        do 3020 J = 1, NCOLM
          if(iobj .ge. ncol(J))  then
            iorder = iorder + 1
          endif
3020    continue
        NBKCOL(iorder) = iloc(I)
3010 CONTINUE
```

⇨

変更後

```
      NCOLM = 0
      DO 3000 JJ=1, MXUSED
        IF( MASTER(JJ) .ne. 0 )   then
          NCOLM = NCOLM + 1
          ncol(NCOLM) = IABS(MASTER(JJ))
          iloc(NCOLM) = JJ
        endif
3000 CONTINUE

      do 3010 I = 1, NCOLM
        iobj   = ncol(I)
        inum = I
        do 3020 J = I+1, NCOLM
          if(iobj .ge. ncol(J)) then
            iobj = ncol(J)
            inum = J
          endif
3020    continue
        NBKCOL(I) = iloc(inum)
        ncol(inum) = ncol(I)
        iloc(inum) = iloc(I)
3010 continue
```

Fig. 4.12 Modification of subroutine BKINDX (VPP→AP3000).

参 考 文 献

[1] 井岡郁夫，他; HENDEL 炉内構造物実証試験部 (T2) 3 次元熱伝導解析コード－ SSPHEAT －，JAERI-M 88-032，1988 年 2 月.

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

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

5. おわりに

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

本報告書では，多次元二流体モデル構成方程式評価用コードACE-3D，原子核統計崩壊計算コードSD及びHENDEL炉内構造物実証試験部（T2）3次元熱伝導解析コードSSPHEATの3本の原子力コードのベクトル化作業について記述した．各コードともベクトル化によりその高速化効果が顕著に表れているが，まだ残された課題，改良すべき点，すなわち高速化の余地があるものも少なくない．今後これらの点を見極め，各コードの計算アルゴリズムの見直し，メモリの効率的利用，並列化の検討等を進めることで，より一層の高速化が期待できるものと思われる．

最後に，本報告書がこれらの仕事に携わる人々に多少なりとも参考になれば幸いである．

謝　辞

本作業を行う上で，作業を依頼された伝熱流動研究室大貫晃氏(2章)，ハドロン輸送研究グループ丸山敏毅氏（3章），HTTR技術開発室竹田武司氏（4章）には，コード内容の把握に際し御協力頂きました．また，本報告書の作成に当り数値実験グループの渡辺正氏には御指導と御助言をいただきました．さらに，本作業を円滑に遂行するための各種事務処理については山田圭子氏に御協力を頂きました．ここにこれらの方々に感謝の意を表します．最後に，本報告書を執筆する機会を与えて下さいました計算科学技術推進センター長竹田辰興氏，情報システム管理課長藤井実氏，(株)富士通R&Dシステム部長平沢健一氏に感謝致します．

This is a blank page.

国際単位系 (SI) と換算表

表1　SI基本単位および補助単位

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

表2　SIと併用される単位

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

1 eV = 1.60218 × 10^{-19} J

1 u = 1.66054 × 10^{-27} kg

表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

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

（注）

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

換算表

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

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

動粘度　1 m^2/s = 10^4 St(ストークス)(cm^2/s)

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

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

1 cal = 4.18605 J (計量法)

= 4.184 J (熱化学)

= 4.1855 J (15 °C)

= 4.1868 J (国際蒸気表)

仕事率　1 PS (仏馬力)

= 75 kgf·m/s

= 735.499 W

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

吸収線量	Gy	rad
	1	100
	0.01	1

照射線量	C/kg	R
	1	3876
	2.58 × 10^{-4}	1

線量当量	Sv	rem
	1	100
	0.01	1

（86年12月26日現在）

原子力コードのVPP500におけるベクトル化，並列化及び移植（ベクトル化編）
— 平成9年度作業報告書 —