JAERI-M
9235

拡散方程式の並列計算

1980年12月

石黒美佐子・古志　裕司*

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

この報告書は，日本原子力研究所が JAERI-M レポートとして，不定期に刊行している研究報告書です．入手，複製などのお問合わせは，日本原子力研究所技術情報部（茨城県那珂郡東海村）あて，お申しこしください．

JAERI-M reports, issued irregularly, describe the results of research works carried out in JAERI. Inquiries about the availability of reports and their reproduction should be addressed to Division of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken, Japan.

JAERI－M 9235

拡散方程式の並列計算

日本原子力研究所東海研究所計算センター
石黒美佐子・古志裕司*

（1980年11月20日受理）

拡散方程式の並列計算について述べる。ここでは，とりわけ高速計算が要望される3次元の問題に重点を置いた。3次元拡散コードは，中性子の拡散を計算する7点階差式を解く部分，つまりinner iterationに大部分の計算時間が費される。したがって，拡散コードはこの部分を並列計算することにより，並列化の効果を上げることができる。

ここでは，主に，(1)7点階差式の計算に関してSOR，SLOR，新しく開発された改良SLOR法，ADI，Cyclic reduction法，さらに行列分解による直接的手法について，数値実験により，それらの並列計算への適応性を検討した。(2)中性子拡散コードADCの並列化を目的とした分析を行った。

この結果，7点階差式の計算については，反復回数の増加をうまく防いでいる改良SLOR法が勝れていることが判った。また，ADCコードについては，SOR法によりinner iterationを並列計算すれば，CRAY－1タイプの計算機では，うまくゆけば計算時間は30％に短縮できることが判った。

* 慶応義塾大学

JAERI-M 9235

Parallel Computation for Neutron Diffusion Equation

Misako ISHIGURO and Yuji KOSHI*

Computing Center, Tokai Research Establishment, JAERI

(Received November 20, 1980)

The paralell computation for neutron diffusion equation is discussed, especially the three dimensional problem for which rapid calculation is demanded. The computing time for the three dimensional diffusion code is mostly consumed by the calculation of neutron diffusion equation which is approximated by seven points difference equations, refferred to as inner iteration. Hence, the diffusion code is indeed able to make effective use of paralell capability, by vectorizing the inner most loop, if possible.

Mainly investigated in this report are as follows:

(1) Six general methods of solution of the seven points difference equations, SOR, SLOR, Modified SLOR, ADI, Cyclic reduction algorithm and Direct method are numerically studied from the viewpoint of their adaptability for parallel computations.

(2) Neutron diffusion code ADC is analyzed to vectorize.

It is seen that the modified SLOR which is newly developed is best to calculating the seven points difference equations, in the sense that the method well prevents the increase of iterations. It is also seen that the computing time for the ADC code can be reduced to thirty percent by vectorizing the inner iteration with the method of SOR, if it goes well.

Keywords: Paralell computers, Vector processors, Paralell computations, Diffusion equations, Three dimensional diffusion calculation, Computer codes

---

* Keio-Gijuku University

目　　次

CONTENTS

1. まえがき

近年，ＣＲＡＹ－１に代表されるベクトルまたはアレイ計算機と呼ばれる並列計算機が国内で商業ベースで利用可能となり，原研でも，富士通のＦ７５ ＡＰＵが原子力カードの並列計算手法の開発研究のために導入されようとしている。

並列計算機の本格的な開発は，１９６０年代の後半からイリノイ大学で始まりＩＬＬＩＡＣⅣを生んだ。合わせて同大学のKuck，Ｊ.Ｄらのグループにより並列計算用数値計算手法の研究が行われてきた。

並列計算機は，気象及び軍事関係の分野で既に効力が認められている。原子力関係の分野でも既に報告がなされており〔１〕，今後適用分野が開発されて行くものと思われる。

原研においては，中央処理装置（ＣＰＵ）を長時間使用する計算コードが多く存在し，それが全ＣＰＵ使用時間の相当部分を占めている。先の機種交換の検討時に，よく利用される原子力コードの並列計算の適用について検討されたが，直ちに有効となるコードは少なかった〔２〕。しかしながら，この際の検討はプログラムの流れを大きく変更しないで最も内側のループ（inner most loop）を並列化するという程度であったこと，また新しい設計の並列計算機が次々に市場に出現しつつあり，より勝れた並列計算のための機能が整備され得ることを考慮すると，原子力分野で適用できる余地がある。

ここでは，原子力コードの代表的な分野の１つである中性子拡散コードについて，特に長時間の計算時間を要する３次元の場合に重点をおいて，並列計算の適用性について調査研究する。拡散コードは，中性子の拡散を差分近似により計算する部分が計算の核を成しており，この部分の並列計算に着目し，並列化した場合の効果を計算手法などの比較により論じる。

報告では，次の問題が主に論じられる。

(1) 並列計算機と並列計算処理のあらまし，拡散方程式の並列計算化へのアプローチ，並列化に利用可能なソフトウェア・ツール，３次元拡散方程式の差分近似による数学モデルの提示など問題の背景について述べる。

(2) 中性子拡散を計算する７点階差式の解法を，逐次過大緩和法（ＳＯＲ，ＳＬＯＲ，改良ＳＬＯＲ），交互方向法（ＡＤＩ）〔３〕，ブロック行列に対するCyclic reduction アルゴリズム〔４〕，直接的手法（Matrix factorization）などの手法を用いて並列計算すれば，どの程度の計算ステップ数になるか，またどの解法が有利かについて調査する。

(3) ７点階差式を反復解法：ＳＯＲ，ＳＬＯＲ，改良ＳＬＯＲ，ＡＤＩ で計算する場合の数値実験を行い，並列計算の効果，並列計算により生じる反復回数の増加などについて手法間の比較を行う。数値実験は，原研で秋元氏らにより開発された多次元拡散コードＡＤＣ〔５〕の定数を使用して行われる。

(4) ＡＤＣを並列計算した場合どの程度計算時間の短縮になるかについて，ソフトウェア・ツールFORTUNE（FORTRUN tuning tool）〔６〕などを使用して分析する。

(5) ２次元拡散方程式の直接解法の改良についての桂木氏の案〔７〕に対する並列計算の効果

の推定を行う。

これらのことから判ったことの概略を次にまとめる。

(1) 3次元拡散コードの並列化

最も良く走行する部分（inner most loop）を並列計算すれば，CRAY－1タイプの計算機では，逐次計算（スカラー計算ともよぶ）に比してうまくゆけば計算時間が約30％に短縮できる。inner most は，7点階差方程式を，SOR，SLORなどの手法により解く部分から成り，この部分の走行時間が全計算時間の約90％を占める。この部分の並列化が課題となり，プログラムの構造，メッシュ数にもよるが，この部分を20％程度に減らすことができれば，上記の計算時間の短縮となる。

(2) 7点階差方程式の解法

ADCから得た定数を使用した16×16×15のメッシュによる数値実験によって以下のことがわかった。

(i) 解法の比較

SOR，改良SLOR，SLOR，ADI の順に並列化の効果が上がる。直接解法，並列計算が有効とされるCyclic redction アルゴリズム〔4〕は冗長な計算が多く不適であることが判った。

(ii) 並列計算範囲

一方向（例えばx方向）だけの並列計算ではあまり効果が上らない。CRAY－1のように演算の立上り時間が短いとされている計算機においても，ベクトル長が10の場合は100の場合に比して，1データ当りの計算速度は約2倍かかる。並列計算の効果を上げるためには，2方向を並列化し，ベクトル長を大きくする必要がある。

(iii) 並列計算による反復回数の増加

並列計算では，同時に計算されるデータは互に使用できないので前回の計算値（古いデータ）を使用する率が高くなっている（Fig.1）。このために反復回数は当然増加する。条件の良いデータの場合には，SOR，SLORの場合，1方向並列計算で10％，2方向並列計算で10～30％増となる。たちの悪いデータの場合には，50％づつ増え，2方向並列計算では通常のSOR，SLORでは収束しない場合も起り，並列計算が有効であると断言できない。

(iv) 緩和因子

緩和因子ωの選択は，スカラー計算，1方向並列，2方向並列に数ってシビアになる。特に2方向並列では，ωの選び方に反復回数はセンシティブである。ωの値は，並列化度合が増すに従って小さくなり，予測したより小さい値（1.0～1.2）が適している。

(v) 改良SLOR

この手法は，並列化による反復回数増を防ぐために，著者により考案されたものである。並列化の効果も良好であり，注目すべき手法である。

2次元の5点階差式の場合のWave front method〔8〕からヒントを得て，斜め方向つまり，$u=j+k$，$u=2, 3, \cdots\cdots$の順に計算していく（Fig.2）。

$$\{ (j, k) \mid j+k=u \}$$

を並列計算するｘ方向（メッシュｉ）のＳＬＯＲの１種の変形手法である。７点階差式では，直前の計算値をスカラー計算と同程度に利用できるのが特徴である。たちの悪いデータに対しても緩和因子の選び方に左右されずに比較的安定した反復回数で収束する。並列化効果についても２方向のＳＬＯＲに近い点で有利である。

(3) ＡＤＣコードの並列化効果

３次元拡散計算で並列計算の効果を引き出すには，２方向の並列計算が不可欠である。７点階差式の部分が１つのサブルーチンでまとめられている場合には，改良ＳＬＯＲなど有利な並列化手法が選択できる。ところが，ＡＤＣの場合には，（ｘ，ｙ）方向に対する７点階差式が１つのサブルーチンで，群，ｚ方向に関して外のルーチンから呼ばれる構造になっている。プログラムに大きく手を加えないかぎり，ＳＯＲ以外の並列化適用は難しい。ＳＬＯＲは，基本となる１方向（例えばｘ方向）については，その解法から本質的に並列化できない手法であり，残るｙ方向を並列化してもメッシュ数が２０程度ではそれ程効果が上らない。

ＡＤＣでは，ＳＯＲよりむしろＳＬＯＲがよく利用され，特にinner／outer iterationsを一度に実行するone throughループの場合には，ＳＬＯＲのみが使用されている。ＳＯＲを使用するように変更するなど必要となる。この変更は容易である。

３次元計算ではone throughループの方がもともとかなり（約３割程度）計算時間が短いのでよく利用されている。one throughループの場合には固有値・中性子源を計算する部分も合わせて並列化する必要があり，この部分の並列化にはかなりの調査が必要となり，３人／月程度の人手がかかるであろう。

ＡＤＣコードにおける問題点の大部分は，他の拡散コードでも言える。VENTURE〔9〕を例にとれば，inner most loopの部分が実行時の主記憶とディスク上のデータ移動の方法によって別々にサブルーチン化されており，またこれらのサブルーチンが数個の外部ファンクションを呼ぶ形をとっている。第三者が並列計算用に書換えるのはかなり難しそうである。

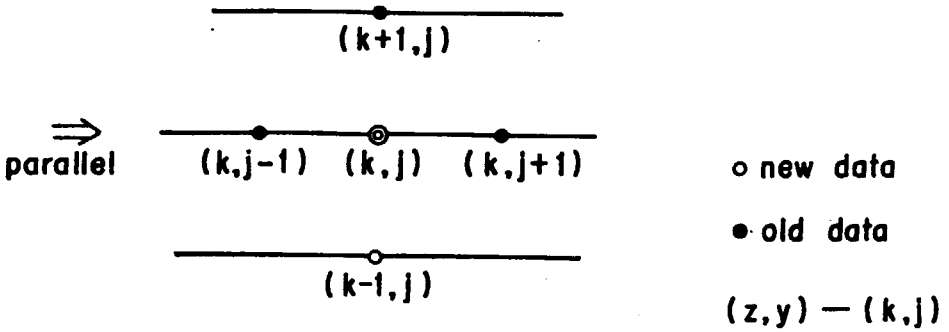


Fig. 1 j-direction parallel

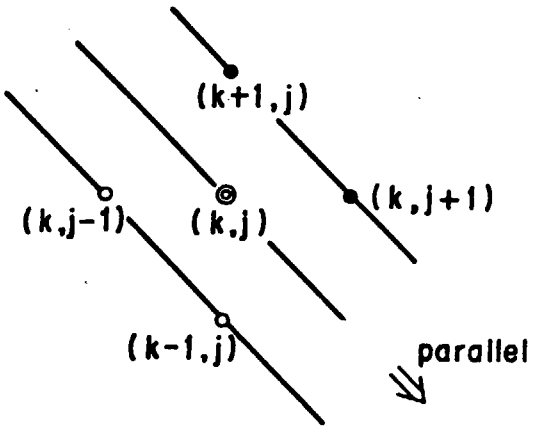


Fig. 2 Parallel to slanting direction

2. 背　　景

2.1 並列計算機と並列計算

並列処理計算機と呼ばれるものは，主に3種の方式に分類される〔10－12〕。

第1のものは，並列処理装置(perallel－processor equipments)方式で，これは，多くのCPUにより同時に演算を実行する方式である。この方式の代表はILLIAC Ⅳであり，64個のCPUが各々2K語（64ビット／語）のメモリを有している〔12〕。この方式の特徴は計算実行前の立上り時間がないことである。一般にn－ベクトルに対する演算処理時間は

$$n\tau_{op} + \sigma_{op} \qquad (2-1)$$

で表わされる。τは1データ当りの演算時間，σは立上り時間であり，opは+，－，×，÷ などの演算を示す。ILLIAC Ⅳでは σ_{op} は0である。しかしながら計算の対象となるデータをベクトル・レジスターに乗せ，CPUの間を通り抜ける制御方式をとり，データの流れの制御が複雑であるという欠点を持っている。

第2の方式は連想記憶（Associative）方式である。これは，メモリの内容に基づくアクセスに主眼が置かれており，科学計算の並列処理には最近あまり用いられない。

第3の方式は，現在主流を占めるパイプ・ライン（pipeline）計算機である。ここでは，1つの演算が複数のステップに分けられ，各ステップに次々とデータが入り，少しづつずらされて処理される（Fig.3）。この結果，複数のデータに対し，単一の演算が実行されることになり，SIMD（Single Instruction Multiple Data）型計算機と呼ばれるものの代表となっている。1つの演算に対するステップ数は5～12程度であり，この数が一般の計算機：スカラー計算機との処理速度の倍率に関係してくる。

パイプ・ライン計算機の代表的な計算機としてCRAY－1が挙げられる。CRAY－1における並列計算について，文献，資料から判ったことについて，ここで少し述べてみる。

CRAY－1は立上り時間が短かく，nが10程度で並列計算の効力を発揮できるという特徴を持っている。演算速度についてはTable 1に挙げられる〔13〕。これによると，ループ・ボディ14－16の計算に関しては，ベクトル長nが10程度では，スカラー計算（最右欄の1000スカイラーの場合)に比して約4倍，nが100以上の場合は約8倍となっており，(2－1) 式の τ_{op} 及び σ_{op} はそれぞれ数10ns，数100nsとなっている。τ_{op} だけから論じればFACOM M 200に比べて10倍程度の処理速度を持つと言える。またDOループの中にIF文がある場合も，ハードウェアとしてはベクトルの各エレメントに対応したMASKedオペレーションが準備されているが，FORTRAN（ベクトル演算向に拡張されたCRAY FORTRAN）では，現在のところ前方への分枝の場合しかベクトル化されない。参考までにCRAY FORTRANによるベクトル化の条件を次に示す〔14〕。

ベクトル化の条件(1)

- 最も内側のＤＯループで配列要素の値を計算
- ＤＯループの中にある配列要素名，変数名がすべて次のいずれかである。

1. 不変変数名（不変配列要素名）
   値が参照されるだけで変更されない
2. 等差整変数名
   ＤＯループの繰返しごとに一定の整数値だけ増減していく整数を等差整数といい，コンパイラーが等差整数を含むと認識できる整変数
3. ベクトル配列要素名
   配列の添字のうち１つだけ等差整変数で，他の添字が不変変数または不変配列要素
4. 一時変数名（一時配列要素名）
   右辺が不変変数，不変配列要素，ベクトル配列要素の値だけで計算されるベクトル化式の左辺の変数（配列要素）
5. 等差整変数は，等差整変数を定義するときだけのために参照できる。

ベクトル化の条件(2)

- ＤＯループの中に次の文がないこと。

1. ＩＦ
2. ＧＯ　ＴＯ
3. ＣＡＬＬ
4. 外部関数を参照
5. スカラー演算による組込み関数を参照

- 未完型依存関係が生じないこと
  配列要素を使っている代入文で実行時にその値が未完成となる。
- 破壊型依存関係が生じないこと
  配列要素を使っている代入文でその値を使う前に壊される。

これらのことからわかるように，一般のプログラムを並列計算用に変更する場合の制約がかなり強いものであり，一方，新たに並列計算用にプログラムを作成する場合においては，機種に依存した注意深いコーディングが要求されることがわかる。したがって実際のプログラムの中では並列化できなかったり，しても効果が上らない部分が多く，必らずしも全部並列化するわけではない。並列計算の有効性は，最も実行時によく走行する部分（inner most loop）がうまく並列化できるかどうかにかかっている。当然のことながら，入出力処理の多いものは，実行経過時が減少しないという点であまり効果がない。

Fig.4にベクトル化率に対するプログラムの処理速度の倍率について示している〔15〕。ここでαはＣＲＡＹ－１では４～８，Ｆ７５ＡＰＵでは５程度となっている。この図から，５倍の速度を得るためにＣＲＡＹ－１の場合においてもベクトル化率が９０％以上でなければならないことが判る。

これらのことから，一般のプログラムを並列化するための現実的なアプローチについて述べてみる。

(1) 全処理時間に占めるＣＰＵ時間比が８０％以上かどうか。

(2) ９０％程度を占めるinner most loop が存在し，それが比較的短かいプログラム・ステップに収まっており，並列化が可能かどうか（FORTUNE による分析）。

(3) ベクトライザを使用してベクトル計算向拡張FORTRAN プログラムに自動的に変換する（VECTORIZER による分析）。

(4) inner most loop のうち自動的にベクトル化できなかった部分の書換を行う（人手による書換）

2.2 並列化のためのソフトウェア・ツール

ベクトル計算向き言語としては，前節のCRAY－FORTRAN のようにFORTRANの拡張言語が主に使用されている。

Ｆ７５－APUに関連して，現在提供されている利用可能なソフトウェア・ツールについて次のものがある。

(1) APU FORTRAN

ベクトル計算用FORTRAN（マニュアル参照）〔16〕

(2) FORTUNE

FORTUNE (FORTRAN tuning tool)〔6〕はFORTRAN で作られたプログラムの実行時の振舞を解析するためのものである。プログラムの実行時の各実行ステートメント及びプログラム単位毎に，その実行回数と相対計算時間（ステートメントの種類などに応じて重みづけられた値）が出力される。inner most loop の部分を探すために主に使用される。

実行ステートメントの相対計算時間は，簡単な代入文，例えばＡ＝Ｂを１とした相対的な計算時間の目安であって，実際の処理時間とは完全な対応づけは出来ない。他の実行ステートメントとの比較程度には使用できる。

(3) VECTORIZER

VECTORIZERは，通常のFORTRAN プログラムを自動的にAPU FORTRANに変換するためのものである。この他に，事前にどの程度各ステートメントのベクトル化が出来るかどうかをチェックする機能を持っている。後者については，

Ｖ（ベクトル化可能）

VC（条件付にベクトル化可能）

Ｓ（ベクトル化不能）

などが表示され，Ｖと表示されたもののみが自動的にベクトル化される。

しかしながら自動ベクトル化については限界がある。ＡＤＣをVECTORIZERにかけた場合について言えば，Ｖの表示は全体の１０％程度であり，ＤＯループの大半はVC となっている。ＶＣの場合は人手により努力すればベクトル化できる可能性があるという意味であり，結局人手をかけなければベクトル化できないというのが現実である。自動ベクトル化については，今後この分野の経験が積みあげられれば，またハードウェア技術が進歩すれば，もう少し良いものが開発されるであろう。

2.3 拡散方程式の並列計算

拡散コードに関しては，軽水炉の場合は群の数が少ないことから，2次元以下の問題は，スカラー計算機（ベクトル計算機に対応して，通常の計算機をそう呼ぶことにする）でもそれ程長い計算時間を要しない。したがって3次元の場合の処理が並列化の対象となる。

一般に拡散コードはFig．5に示すinner／outerの二重ループによる反復計算により計算される〔17〕。最近はEQUIPOISEの手法，つまりinner－iterationとouter－iterationを同時に行う1重ループの反復計算方式の方がよく利用されており，特に中性子束の収束よりも固有値K－effective，を求めることが目的の場合に1重ループ方式が有効であるとされている。Fig。5で示すように，inner iteration 或いはそれを拡大した一重ループの部分が計算の核をなし，この部分を並列計算できればかなりの速度アップが計れると期待できる。

2.4 inner iteration の数学モデル

多群，中性子拡散方程式は，次式で与えられる〔5〕。

$$-D_g(r)\,\nabla^2\phi_g(r)+(\Sigma_{a,g}(r)+\Sigma_{r,g}(r)+D_g(r)B^2)\,\phi_g(r)$$

$$=\chi_g\sum_{g'}\frac{\nu\Sigma_{f,g'}(r)\,\phi_{g'}(r)}{k_{eff}}+\Sigma_{r,g-1}(r)\,\phi_{g-1}(r),\quad g=1,2,\cdots\cdots G,\quad (2-2)$$

where

∇^2 = Laplacian differential operator : $\frac{\partial^2}{\partial X^2}+\frac{\partial^2}{\partial Y^2}+\frac{\partial^2}{\partial Z^2}$ in slab geometry，(cm^{-2})，

$\phi_g(r)$ =Neutron flux at location，r，and in energy group，g，$(n/sec-cm^2)$，

$\Sigma_{a,g}(r)$ =Macroscopic absorption cross section．(cm^{-1})，

$\Sigma_{r,g}(r)$ =Macroscopic slowing down cross section from energy group g to (g+1)，(cm^{-1})，

$D_g(r)$ =Diffusion coefficient，

B_g^2 =Buckling term to account for the effect of the neutron leakage to the perpendicular direction，if necessary，(cm^{-2})，

$\nu\Sigma_{f,g}(r)$ =Macroscopic emission cross section（ν is the number of neutrons emitted by a fission and Σ_f is the fission cross section），(cm^{-1})，

χ_g =Fission neutron energy spectrum（$\sum_g\chi_g=1.0$），

k_{eff} =Effective multiplication factor，or the largest eigenvalue of the equation．

(2－2) 式の空間方向（x，y，z）に対し，インデックス（i，j，k）を対応させ，差分法によ

り近似すると，中性子束；ϕは3次元7点階差によって次式のように示される。

$$\phi(i,j,k)=S(i,j,k)-C_6(i,j,k)\,\phi(i,j,k+1)-C_5(i,j,k)\,\phi(i,j,k-1)$$
$$-C_4(i,j,k)\,\phi(i,j+1,k)-C_3(i,j,k)\,\phi(i,j-1,k)$$
$$-C_2(i,j,k)\,\phi(i+1,j,k)-C_1(i,j,k)\,\phi(i-1,j,k). \qquad (2-3)$$

この式をϕについて解く部分が拡散コードのinner most loopをなす。従来から(2－3)式の解法には主にＳＯＲやＳＬＯＲなどの逐次過大緩和法が用いられてきた。2次元拡散コードＫＡＫ〔18〕のように直接解法が用いられた場合もあるが，後に示すように3次元の問題では，行列の逆転が必要となり，直接法はスカラー計算機においてもそれ程有効なものではない。また並列計算にも適応しないことが判った。

(2－3)式を行列の形式に書いてみることにより，1次元，2次元の場合との式の対比ができる。

(2－3)式を簡略化して次式のように書くことにする。

$$a_{k,j,i}f_{k+1,j,i}+b_{k,j,i}f_{k,j+1,i}+c_{k,j,i}f_{k-1,j,i}+d_{k,j,i}f_{k,j-1,i}$$
$$+p_{k,j,i}f_{k,j,i+1}+q_{k,j,i}f_{k,j,i-1}-e_{k,j,i}f_{k,j,i}+h_{k,j,i}=0,$$
$$1\leqq k\leqq N,\quad 1\leqq j\leqq M,\quad 1\leqq i\leqq L\ . \qquad (2-4)$$

$$\mathbf{e}_{k,j}=\begin{pmatrix} e_{k,j,1} & -p_{k,j,1} & & & \\ -q_{k,j,2} & e_{k,j,2} & -p_{k,j,2} & & \\ & \ddots & \ddots & \ddots & \\ & & & -q_{k,j,L} & e_{k,j,L} \end{pmatrix},$$

$$\mathbf{a}_{k,j}=\begin{pmatrix} a_{k,j,1} & & & \\ & a_{k,j,2} & & \\ & & \ddots & \\ & & & a_{k,j,L} \end{pmatrix},\qquad \mathbf{b}_{k,j}=\begin{pmatrix} b_{k,j,1} & & & \\ & b_{k,j,2} & & \\ & & \ddots & \\ & & & b_{k,j,L} \end{pmatrix},$$

$$\mathbf{c}_{k,j}=\begin{pmatrix} c_{k,j,1} & & & & \\ & c_{k,j,2} & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & c_{k,j,L} \end{pmatrix}, \qquad \mathbf{d}_{k,j}=\begin{pmatrix} d_{k,j,1} & & & & \\ & d_{k,j,2} & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & d_{k,j,L} \end{pmatrix},$$

$$\mathbf{f}_{k,j}=\begin{pmatrix} f_{k,j,1} \\ f_{k,j,2} \\ \cdot \\ \cdot \\ \cdot \\ f_{k,j,L} \end{pmatrix}, \qquad \mathbf{h}_{k,j}=\begin{pmatrix} h_{k,j,1} \\ h_{k,j,2} \\ \cdot \\ \cdot \\ \cdot \\ h_{k,j,L} \end{pmatrix}, \qquad (2-5)$$

と表わし，これをまとめる。

$$M=\begin{pmatrix} \begin{pmatrix} e_{11} & -b_{11} & & \\ -d_{12} & e_{12} & -b_{12} & \\ & \cdot & \cdot & \cdot \\ & & & -d_{1M}\ e_{1M} \end{pmatrix}\begin{pmatrix} -a_{11} & & & \\ & -a_{12} & & \\ & & \cdot & \\ & & & -a_{2M} \end{pmatrix} & & \bigcirc \\ & \begin{pmatrix} -c_{21} & & & \\ & -c_{22} & & \\ & & \cdot & \\ & & & -c_{2M} \end{pmatrix}\begin{pmatrix} e_{21} & -b_{21} & & \\ -d_{22} & e_{22} & -b_{22} & \\ & \cdot & \cdot & \cdot \\ & & & -d_{2M}\ e_{2M} \end{pmatrix}\begin{pmatrix} -a_{21} & & & \\ & -a_{22} & & \\ & & \cdot & \\ & & & -a_{2M} \end{pmatrix} & \\ \bigcirc & & \begin{pmatrix} -c_{N1} & & & \\ & -c_{N2} & & \\ & & \cdot & \\ & & & -c_{NM} \end{pmatrix}\begin{pmatrix} e_{N1} & -b_{N1} & & \\ -d_{N2} & e_{N2} & -b_{N2} & \\ & \cdot & \cdot & \cdot \\ & & & -d_{NM}\ e_{NM} \end{pmatrix} \end{pmatrix},$$

$$F = \begin{pmatrix} f_{1,1} \\ f_{1,2} \\ \vdots \\ f_{1,M} \\ f_{2,1} \\ f_{2,2} \\ \vdots \\ f_{2,M} \\ \vdots \\ f_{N,M} \end{pmatrix}, \quad H = \begin{pmatrix} h_{1,1} \\ h_{1,2} \\ \vdots \\ h_{1,M} \\ h_{2,1} \\ h_{2,2} \\ \vdots \\ h_{2,M} \\ \vdots \\ h_{N,M} \end{pmatrix} \qquad (2-6)$$

(2−4) 式は (2−5), (2−6) 式によって次の 1 次方程式となる。

$$MF = H \qquad (2-7)$$

つまり, 3 次元の場合の係数行列Mは, 2 重にブロック化された 3 対角行列をなす。2 次元の場合にはブロック 3 対角行列, 1 次元の場合は 3 対角行列となる。

Table 1

Execution time in clock periods per result for various simple DO loops of the form

(1 clock = 12.5 ns)

```
DO 10 I = 1,N
10 A(I) = B(I)
```

Loop Body	N = 1	10	100	1000	1000 Scalar
1. A(I) = 1.	41.0	5.5	2.6	2.5	22.5
2. A(I) = B(I)	44.0	5.8	2.7	2.5	31.0
3. A(I) = B(I) + 10.	55.0	6.9	2.9	2.6	37.0
4. A(I) = B(I) + C(I)	59.0	8.2	3.9	3.7	41.0
5. A(I) = B(I)*10.	56.0	7.0	2.9	2.6	38.0
6. A(I) = B(I)*C(I)	60.0	8.3	4.0	3.7	42.0
7. A(I) = B(I)/10.	94.0	10.8	4.1	3.7	52.0
8. A(I) = B(I)/C(I)	89.0	13.3	7.6	7.2	60.0
9. A(I) = SIN(B(I))	462.0	61.0	33.3	31.4	198.1
10. A(I) = ASIN(B(I))	430.0	209.5	189.5	188.3	169.1
11. A(I) = ABS(B(I))	61.0	7.5	2.9	2.6	
12. A(I) = AMAX1(B(I),C(I)) C(I) = A(I)	80.0	11.2	5.2	4.8	
13. A(I) = B(I) B(I) = CCI	90.0	12.7	6.3	5.8	47.0
14. A(I) = B(I)*C(I) + D(I)*E(I)	110.0	16.0	7.7	7.1	57.0
15. A(I) = B(I)*C(I) + (D(I)*E(I))	113.0	14.7	6.6	6.0	63.0
16. A(I) = B(I)*C(I) + D(I)	95.0	12.7	5.5	5.0	52.0

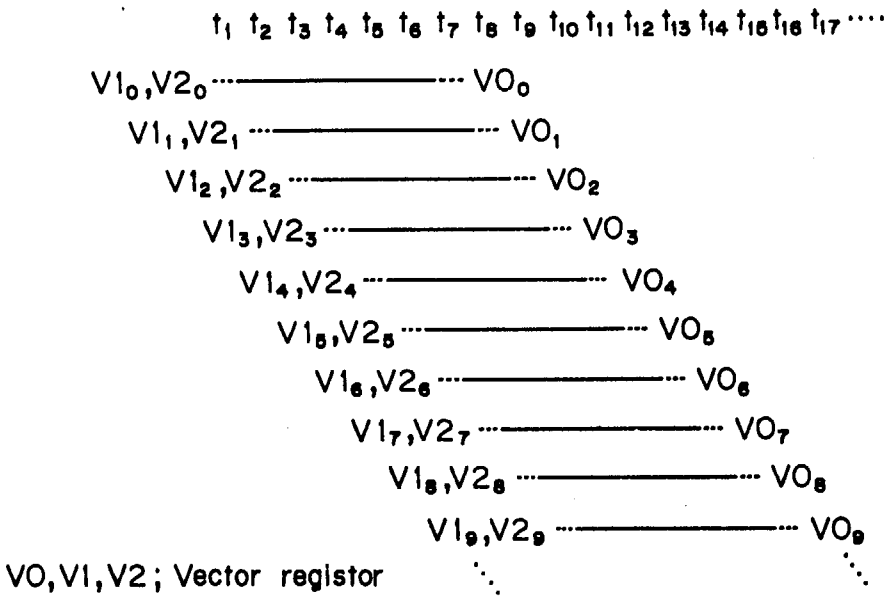


Fig.3 Vector instruction timing example

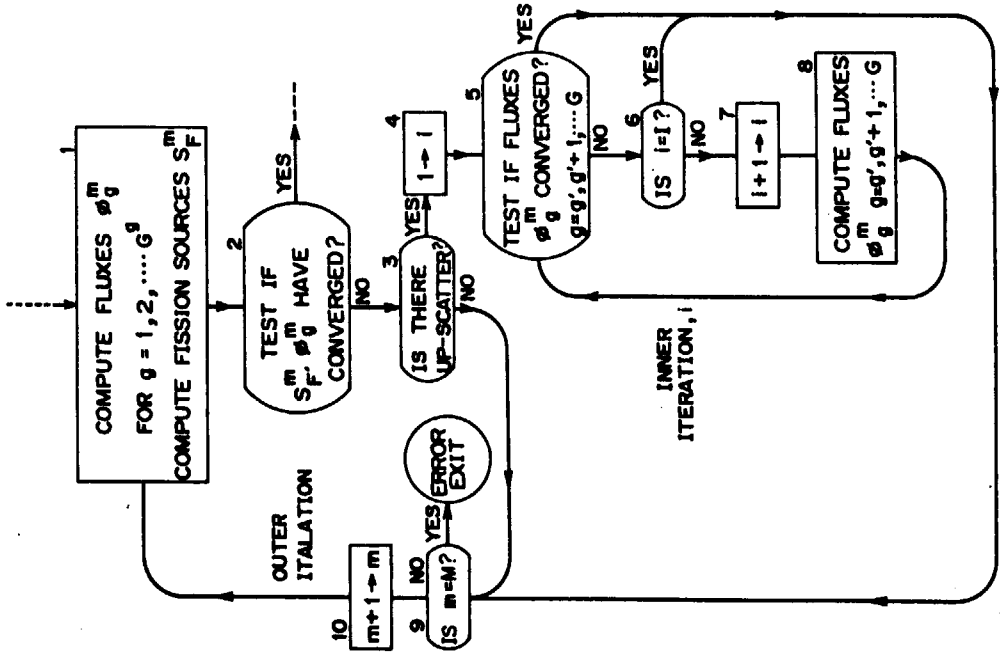


Fig.5 Flow diagram of source iteration procedure

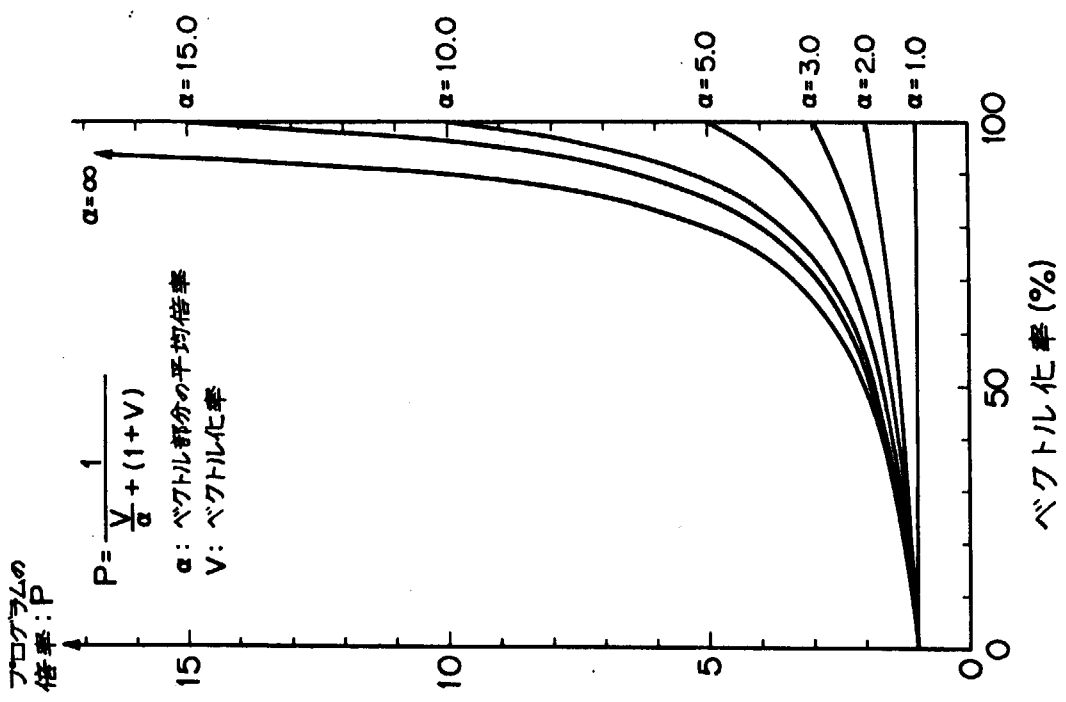


Fig.4 Efficiency of parallel processing

3. 並列計算手法の演算数の比較

3.1 前 提

(2－4) 式で示された7点階差方程式を並列計算すれば，どの程度の演算ステップになるかについて調べる。手法としては，点ごとの逐次過大緩和法SOR (pointwise successive over－relaxation)，行ごとの逐次過大緩和法SLOR (linewise successive over－relaxation)，斜め方向に計算を進める改良SLOR，交互方向法ADI (alternating－direction implicit iterative method) matrix factorization による直接法，さらにCyclic odd even reduction アルゴリズム〔4〕の6種をとり挙げる。

並列計算による演算回数については，Kuck らにより通常よく行なわれているあまり現実的でない次の仮定を用いる〔19〕。

(1) K台のプロセッサが利用可能である。

(2) 演算の立上り時間（メモリーからベクトル・レジスターへの移動など含む）を考慮しない。

(3) 各演算（+, －, ×, ÷, 大小比較）の時間については一律化して考える（ここでは各演算時間を考慮している）。

パイプライン計算機では必らずしも上記の仮定に基づく演算ステップの推定は適合しないが，各計算手法の計算時間の比較の目安にはなるのでこの仮定を用いることにした。

演算回数については，例えば，MXN行列の計算では，

t_A 加減算に要する1データ当りの演算時間

t_M 乗算に要する1データ当りの演算時間

t_D 除算に要する1データ当りの演算時間

とおくと，次のような演算回数となる。

加減算　$M\cdot\lceil N/K\rceil\cdot t_A$

乗　算　$\{M t_M+(M-1)t_A\}\lceil N^2/K\rceil \fallingdotseq M\cdot\lceil N^2/K\rceil (t_A+t_M)$

逆　転　$\left\{N t_D+\frac{1}{2}N(N+1)(t_A+t_M)+t_D\right\}\cdot\lceil N/K\rceil$

$$\fallingdotseq\left\{N t_D+\frac{1}{2}N^2(t_A+t_M)\right\}\cdot N/K$$

(Gauss－Jordan Algorithm〔20〕)

ここで $\lceil N/K\rceil$ はN次ベクトル計算をK個のプロセッサで実行した時の倍率で，ILLIAC Ⅳタイプの場合は，$\lceil\ \rceil$ は“切り上げ”と考えてよい。CRAY－1のような計算機の場合はN／8～N／4の値となる。

3.2 SOR (Pointwise successive over－relaxation)

(2－4) 式から通常のスカラー計算では，次の反復式が得られる。

$$f^{(n)}_{k,j,i} = e^{-1}_{k,j,i} \{ a_{k,j,i} f^{(n-1)}_{k+1,j,i} + b_{k,j,i} f^{(n-1)}_{k,j+1,i} + c_{k,j,i} f^{(n)}_{k-1,j,i}$$

$$+ d_{k,j,i} f^{(n)}_{k,j-1,i} + p_{k,j,i} f^{(n-1)}_{k,j,i+1} + q_{k,j,i} f^{(n)}_{k,j,i-1} + h_{k,j,i} \}$$

ところが並列計算では，同時に計算されるものについては，計算前の値が使用されることになる。i 方向の並列計算では，$f^{(n)}_{k,j,i-1}$ の代りに $f^{(n-1)}_{k,j,i-1}$ が使用される。

(1) i 方向に並列計算

$$f^{(n)}_{k,j,i} = e^{-1}_{k,j,i} \{ a_{k,j,i} f^{(n-1)}_{k+1,j,i} + b_{k,j,i} f^{(n-1)}_{k,j+1,i} + c_{k,j,i} f^{(n)}_{k-1,j,i}$$

$$+ d_{k,j,i} f^{(n)}_{k,j-1,i} + p_{k,j,i} f^{(n-1)}_{k,j,i+1} + q_{k,j,i} f^{(n-1)}_{k,j,i-1} + h_{k,j,i} \},$$

$$f^{(n)}_{k,j,i} = \omega f^{(n)}_{k,j,i} + (1-\omega) f^{(n-1)}_{k,j,i}, \quad \omega \text{は緩和因子} \quad 1 \leqq \omega < 2.$$

$$1 \leqq i \leqq L, \quad 1 \leqq j \leqq M, \quad 1 \leqq k \leqq N \qquad (3-1)$$

◎ 演算回数 $(t_D + 8t_M + 7t_A) \cdot MN \cdot \lceil L/K \rceil \cdot (\text{反復回数})_{(i)}$

(2) (i,j) 方向に並列計算

(i,j) 方向については，(3－1) の $f^{(n)}_{k,j-1,i}$ の代りに，$f^{(n-1)}_{k,j-1,i}$ が使用される。

◎ 演算数回数 $(t_D + 8t_M + 7t_A) \cdot N \cdot \lceil LM/K \rceil \cdot (\text{反復回数})_{(i,j)}$

ここで $(\text{反復回数})_{スカラー} \leqq (\text{反復回数})_{(i)} \leqq (\text{反復回数})_{(i,j)}$

3.3 SLOR (Linewise successive over－relaxation)

(2－6) 式より，k 方向に並列計算する場合には，

$$\mathbf{k}_{k,j} = \mathbf{c}_{k,j} \mathbf{f}^{(n-1)}_{k-1,j} + \mathbf{d}_{k,j} \mathbf{f}^{(n)}_{k,j-1} + \mathbf{b}_{k,j} \mathbf{f}^{(n-1)}_{k,j+1} + \mathbf{a}_{k,j} \mathbf{f}^{(n-1)}_{k+1,j} + \mathbf{h}_{k,j},$$

$$\mathbf{e}_{k,j} \cdot \tilde{\mathbf{f}}^{(n)}_{k,j} = \mathbf{k}_{k,j}$$

但し，$\mathbf{e}_{k,j}$ はL×Lの3角行列，$\mathbf{a}_{k,j}$，$\mathbf{b}_{k,j}$，$\mathbf{c}_{k,j}$，$\mathbf{d}_{k,j}$ は対角行列である。

L×L行列 $\mathbf{e}_{k,j}$ の行列分解（matrix factorization）により，

$$u_1 = p_{k,j,1} / e_{k,j,1}, \quad u_i = p_{k,j,i} (e_{k,j,i} - q_{k,j,i} u_{i-1})^{-1}$$

$$i = 2, 3, \cdots, L-1$$

$$v_1 = k_{k,j,1} / e_{k,j,1}, \quad v_i = (k_{k,j,i} + q_{k,j,i} v_{i-1})(e_{k,j,i} - q_{k,j,i} v_{i-1})^{-1}$$

$$i = 2, 3, \cdots, L$$

$$\tilde{f}^{(n)}_{k,j,L} = v_L, \quad \tilde{f}^{(n)}_{k,j,i} = v_i + u_i f_{k,j,(i+1)}$$

$$i = L-1, L-2, \cdots, 1 \qquad (3-3)$$

u_i, v_i, $\tilde{f}^{(n)}_{k,j,i}$ の計算は i については前回の値に依存した計算であるので並列計算は本質的に不可能である。

◎ **演算回数について**

1組の(j，k)に対して $f^{(n)}_{k,j,i}$，(i＝1, 2,…, L) を求める。

① k の計算　　$4L(t_A + t_M)$

② u の計算　　$t_D + (L-2)(t_D + t_A + t_M)^*$;　　* 最初の1回のみ必要

③ v の計算　　$t_D + (L-1)(t_D + t_A + t_M)$

④ $\tilde{f}$ の計算　　$(L-1)(t_A + t_M)$

⑤ $\mathbf{f}^{(n)}_{k,j} = \omega \tilde{\mathbf{f}}^{(n)}_{k,j} + (1-\omega)\mathbf{f}^{(n)}_{k,j}$ の計算　　$(2t_M + t_A) \cdot L$

①－⑤合計　約 $(7t_A + 8t_M + t_P)L + (t_A + t_M + t_D)L$ *

(1) k 方向に並列計算する場合

$$(7t_A + 8t_M + t_D) \cdot LM \cdot \lceil N/K \rceil \cdot (\text{反復回数})_{(k)}$$
$$+ (t_A + t_M + t_D) LM \cdot \lceil N/K \rceil$$

(2) (k，j) 方向に並列計算する場合

$$(7t_A + 8t_M + t_D) \cdot L \cdot \lceil MN/K \rceil \cdot (\text{反復回数})_{(k,j)}$$
$$+ (t_A + t_M + t_D) L \cdot \lceil MN/K \rceil$$

3.4 直接解法（Direct method）

(2－4) 式を次のように2重ブロック化して，行列分解により解く。

$$\begin{pmatrix} E_1 & -A_1 & & & \\ -C_1 & E_2 & -A_2 & & \\ & \cdot & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & & & -C_N E_N \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \\ \cdot \\ \cdot \\ \cdot \\ F_N \end{pmatrix} = \begin{pmatrix} H_1 \\ H_2 \\ \cdot \\ \cdot \\ \cdot \\ H_N \end{pmatrix} \qquad (3-4)$$

ここで，E_i（i＝1，2，…，N）は，M×Mブロック3対角行列で，各ブロックはまたL×L行列である。A_i，C_iについては，LM×LMの対角行列である。F_i，H_iはLM次のベクトルである。

$$E_i=\begin{pmatrix} e_{i,1} & -b_{i,1} & & & \\ -d_{i,2} & e_{i,2} & \ddots & & \\ & & & -b_{i,M-1} & \\ & & -d_{i,M} & e_{i,M} & \end{pmatrix},\quad A_i=\begin{pmatrix} a_{i,1} & & & \\ & a_{i,2} & & \\ & & \ddots & \\ & & & a_{i,M} \end{pmatrix},$$

$$C_i=\begin{pmatrix} c_{i,1} & & & \\ & c_{i,2} & & \\ & & \ddots & \\ & & & c_{i,M} \end{pmatrix},\quad F_i=\begin{pmatrix} f_{i,1} \\ f_{i,2} \\ \vdots \\ f_{i,M} \end{pmatrix},\quad H_i=\begin{pmatrix} h_{i,1} \\ h_{i,2} \\ \vdots \\ h_{i,M} \end{pmatrix}. \qquad (3-5)$$

(3－4) 式より

$$F_{i+1}=A^{-1}(E_iF_i-C_iF_{i-1}-H_i)$$

$$=P_iF_i-Q_iF_{i-1}-R_i \qquad \text{とおく。} \qquad (3-6)$$

ここで

$$P_i=A_i^{-1}E_i,\quad Q_i=A_i^{-1}C_i,\quad R_i=A_i^{-1}H_i$$

次にF_iを次の後進漸化式に表現する。

$$F_i=Q_{i+1}(S_{i+1}F_{i+1}+J_{i+1}) \qquad (3-7)$$

(3－7) 式を (3－6) 式に代入していくことにより，次式を得る。

$$S_{i+1}=Q_{i+1}(P_i-S_i)^{-1}, \qquad (3-8)$$

$$J_{i+1}=S_{i+1}(J_i+R_i), \qquad (3-9)$$

$$F_i=(P_i-S_i)^{-1}(F_{i+1}+J_i+R_i). \qquad (3-10)$$

ここで初期値は $S_1=Q_1$，$J_1=0$，$F_N=0$，$F_{N-1}=Q_N^{-1}J_N$

P_i　M×Mブロック行列で，各ブロックはL×Lの3対角行列

Q_i　LM×LMの対角行列

R_i　ＬＭ－ベクトル

S_i　Ｍ×Ｍブロック行列で，各ブロックは一般のＬ×Ｌ行列

◎　演算回数について

(3－6) 式各ｉに対して

①　A_i^{-1} の計算

A_i はＬＭ×ＬＭの対角行列だから　　$t_D \cdot \lceil LM/K \rceil$

②　$P_i = A_i^{-1} E_i$　　$3t_M \cdot \lceil LM/K \rceil$

③　$Q_i = A_i^{-1} C_i$　　$t_M \cdot \lceil LM/K \rceil$

④　$R_i = A_i^{-1} H_i$　　$t_M \cdot \lceil LM/K \rceil$

(3－6) 式計　　$(5t_M + t_D) \cdot \lceil LM/K \rceil$

(3－8) 式

①　$P_i - S_i$　　$3t_A \cdot \lceil LM/K \rceil$

②　$(P_i - S_i)^{-1}$　　$\{ LMt_D + \frac{1}{2}(LM)^2 (t_A + t_M) \} \cdot \lceil LM/K \rceil$

③　$Q_{i+1}(P_i - S_i)^{-1}$　　$t_M \cdot \lceil LM/K \rceil$

(3－8) 式計

$$\{ \frac{1}{2}(ML)^2 (t_A + t_M) + t_M + 3t_A \} \cdot \lceil LM/K \rceil$$

(3－9) 式　　$\{ t_A + LM(t_A + t_M) \} \cdot \lceil LM/K \rceil$

(3－10) 式　　$\{ 3t_A + LM(t_A + t_M) \}, \cdot \lceil LM/K \rceil$

(3－6)，(3－8) － (3－10) 式合計

$$N\{ \frac{1}{2}(ML)^2 (t_A + t_M) + LM(2t_A + 2t_M + t_D) + 7t_A + 6t_M + t_D \} \cdot \lceil LM/K \rceil$$

3.5　改良ＳＬＯＲ（Modified SLOR）

(2－6) 式をまとめると，

$$e_{k,j} f_{k,j} = a_{k,j} f_{k+1,j} + b_{k,j} f_{k,j+1} + c_{k,j} f_{k-1,j} + d_{k,j} f_{k,j-1} + h_{k,j} \qquad (3-11)$$

$k + j = u$ とおき (3－11) 式を ｕ の順に並べかえ，ｕ についてまとめると，次式のようになる。

$$
\begin{pmatrix} e_{1,u-1} f_{1,u-1} \\ e_{2,u-2} f_{2,u-2} \\ \vdots \\ e_{u-2,2} f_{u-2,2} \\ e_{u-1,1} f_{u-1,1} \end{pmatrix}
=
\begin{pmatrix} d_{1,u-1} & & & \\ c_{2,u-2} & d_{2,u-2} & & \\ & \ddots & \ddots & \\ & & c_{u-2,2} & d_{u-2,2} \\ & & & c_{u-1,1} \end{pmatrix}
\begin{pmatrix} f_{1,u-2} \\ f_{2,u-3} \\ \vdots \\ f_{u-3,2} \\ f_{u-2,1} \end{pmatrix}
$$

$$
+
\begin{pmatrix} b_{1,u-1} & a_{1,u-1} & & \\ & b_{2,u-2} & a_{2,u-2} & \\ & & \ddots & \ddots \\ & & & b_{u-1,1} \end{pmatrix}
\begin{pmatrix} f_{1,u} \\ f_{2,u-1} \\ \vdots \\ f_{u-1,2} \\ f_{u,1} \end{pmatrix}
$$

$$
+
\begin{pmatrix} h_{1,u-1} \\ h_{2,u-2} \\ \vdots \\ h_{u-2,2} \\ h_{u-1,1} \end{pmatrix}
\qquad (3-12)
$$

$$
F_u = \begin{pmatrix} f_{1,u-1} \\ f_{2,u-2} \\ \vdots \\ f_{u-1,1} \end{pmatrix}, \qquad
H_u = \begin{pmatrix} h_{1,u-1} \\ h_{2,u-2} \\ \vdots \\ h_{u-1,1} \end{pmatrix},
$$

$$
D_u = \begin{pmatrix} d_{1,u-1} & & & 0 \\ c_{2,u-2}, & d_{2,u-2} & & \\ & \ddots & \ddots & \\ 0 & & \ddots & \\ & & & c_{u-1,1} \end{pmatrix},
$$

ここで，$f_{k,u}$，$h_{k,u}$ は (2－5) で定義されているL 次のベクトルである。

$$B_u = \begin{pmatrix} \mathbf{b}_{1,u-1} & \mathbf{a}_{1,u-1} & & & 0 \\ & \mathbf{b}_{2,u-2} & \mathbf{a}_{2,u-2} & & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \\ 0 & & & \mathbf{b}_{u-1,1} & \mathbf{a}_{u-1,1} \end{pmatrix},$$

$$E_u = \begin{pmatrix} \mathbf{e}_{1,u-1} & & & 0 \\ & \mathbf{e}_{2,u-2} & & \\ & & \cdot & \\ & & & \cdot \\ 0 & & & \mathbf{e}_{u-1,1} \end{pmatrix}.$$

ここで，$\mathbf{a}_{k,u}$，$\mathbf{b}_{k,u}$，$\mathbf{c}_{k,u}$，$\mathbf{d}_{k,u}$ 及び $\mathbf{e}_{k,u}$ は (2－5) で定義されているL×L行列である。

(3－13)

以上の定義より (3－12) 式は以下に書直すことができる。

$$E_u F_u = H_u + D_u F_{u-1} + B_u F_{u+1} \tag{3－14}$$

そこで

$$E_u \tilde{F}_u^{(n)} = H_u + D_u F_{u-1}^{(n)} + B_u F_{u+1}^{(n-1)} \tag{3－15}$$

$$F_u^{(n)} = \omega F_u^{(n)} + (1-\omega) F_u^{(n-1)} \tag{3－16}$$

とおき，(3－15)，(3－16) を収束するまでくり返す。(3－15) 式の $F_u^{(n)}$ の計算にはSLOR法で用いたのと同様の方法を使用する。つまり，

$$\mathbf{e}_{k,u-k} \mathbf{f}_{k,u-k}^{(n)} = \mathbf{a}_{k,u-k} \mathbf{f}_{k+1,u-k}^{(n-1)} + \mathbf{b}_{k,u-k} \mathbf{f}_{k,u-k+1}^{(n-1)} + \mathbf{c}_{k,u-k} \mathbf{f}_{k-1,u-k}^{(n)} + \mathbf{d}_{k-1,u-k} \mathbf{f}_{k,u-k-1}^{(n)} + \mathbf{h}_{k,u-k} \tag{3－17}$$

を (3－2) 式と同様の計算により実行する。$\{ (i,j,k) \mid j+k=u \}$ (Fig.6) に関して並列計算する。

◎ 演算の回数について

3.3 の記述に対応させて述べる。

① $\mathbf{k}$ の計算 ((3－16) 式右辺)

i 方向 ($i=1, 2, \cdots, L$) とk方向 ($k_{min} \leqq k \leqq k_{max}$) に並列計算可能，従って

$$\sum_{n=2}^{N+M} 4(t_A + t_M) \cdot \lceil L N_u / K \rceil$$

ここで，$N_u = k_{max} - k_{min} = \min\{u-1, \min(N, M)\}$.

② u の計算　（最初の1回のみ）

$$\sum_{u=2}^{N+M}\{t_D+(L-2)(t_D+t_A+t_M)\}\cdot\lceil N_u/K\rceil^*, \quad N_u \text{は後述}$$

③ v の計算

$$\sum_{u=2}^{N+M}\{t_D+(L-1)(t_D+t_A+t_M)\}\cdot\lceil N_u/K\rceil$$

④ $\tilde{f}$ の計算

$$\sum_{u=2}^{N+M}\{(L-1)(t_A+t_M)\}\cdot\lceil N_u/K\rceil$$

u, v, f の計算は i 方向について逐次計算が必要である。

⑤ $\mathbf{f}_{k,j}^{(n)} = \omega\tilde{\mathbf{f}}_{k,j}^{(n)} + (1-\omega)\mathbf{f}_{k,j}^{(n-1)}$ の計算

$$\sum_{u=2}^{N+M}(2t_M+t_A)\cdot\lceil LN_u/K\rceil$$

合計約

$$\sum_{u=2}^{N+M}\{(5t_A+6t_M)\cdot\lceil LN_u/K\rceil + L(2t_A+2t_M+t_D)\cdot\lceil N_u/K\rceil + (t_A+t_M+t_D)\cdot\lceil N_u/K\rceil^*\}$$

Fig。4 より

$N_u = 1, 2, 3, \cdots, N, \cdots N, N-1, N-2, \cdots, 2, 1$　　（N＜Mの時）

平均 $\frac{N}{2}$ とすると，上式は約

$$\{(N+M)(5t_A+6t_M)\cdot\left\lceil\frac{1}{2}LN/K\right\rceil + (N+M)L(2t_A+2t_M+t_D)\cdot\left\lceil\frac{1}{2}N/K\right\rceil\}\cdot(\text{反復回数})_{(n)} + (N+M)L(t_A+t_M+t_D)\cdot\left\lceil\frac{1}{2}N/K\right\rceil$$

3.6 ADI (Alternating-direction implicit iterative method)

ADI法は既約スティルチェス行列（実対称正定符号既約行列で，非対角要素が非正である）に対して適用可能な反復解法である〔3〕。

7点階差の1次方程式を解く場合は，（2－6）のLMN次元行列Mを4つの行列E，P，

B，Aに分けて次のように表示する。

$$M = E + P + B + A$$

ここで　$E = (e_{kji})$ の対角行列

$$kji = \{(k-1)\times N + j - 1\}\times M + i \ ,$$

$$1 \leqq i \leqq L \ , \quad 1 \leqq j \leqq M , \quad 1 \leqq k \leqq N \ .$$

$P = (-p_{kji} \ , \ 0 , \ -q_{kji})$ の3対角行列

$B = (-d_{kji} , \ 0 , \ -b_{kji})$ の3対角行列

$A = (-c_{kji} , \ 0 , \ -a_{kji})$ の3対角行列 .

次に

$$X = \frac{1}{3} E + P , \qquad Y = \frac{1}{3} E + B , \quad Z = \frac{1}{3} E + A \tag{3－18}$$

と置けば，7点近似公式で得られる行列問題は，

$$(X + Y + Z)\,\mathbf{f} = \mathbf{h} \tag{3－19}$$

となる。$\mathbf{f}^{(m)}$ を次式のような反復法で計算する。

$$(X + r\,I)\,\mathbf{u}^{(1)}_{m+1} = 2\mathbf{h} + (r\,I - X - 2Y - 2Z)\,\mathbf{f}^{(m)} \tag{3－20}$$

$$(Y + r\,I)\,\mathbf{u}^{(2)}_{m+1} = 2\mathbf{h} + (r\,I - X - Y - 2Z)\,\mathbf{f}^{(m)} - X\mathbf{u}^{(1)}_{m+1} \tag{3－21}$$

$$(Z + r\,I)\,\mathbf{f}^{(m+1)} = 2\mathbf{h} + (r\,I - X - Y - Z)\mathbf{f}^{(m)} - X\mathbf{u}^{(1)}_{m+1} - Y\mathbf{u}^{(2)}_{m+1} \tag{3－22}$$

ここで，X，Y，ZはLMN×LMNの3対角行列で，非対角要素と対角要素との距離は，

$$X \quad \pm 1 \ , \qquad Y \quad \pm L \ , \qquad Z \quad \pm LM$$

となっている。

(3－20)－(3－22)は，右辺がそれぞれLMNの次数で並列計算が可能である。左辺はSLORで用いた行列分解法により解を得る。行列分解は(3－3)式で示すように，並列計算は一般に不可能であるが，(2－6)式に立ち返れば，ブロック間の並列計算は可能である。この性質を利用して演算回数を計算する。

① $X\mathbf{f}^{(m)}$，$Y\mathbf{f}^{(m)}$，$Z\mathbf{f}^{(m)}$ の計算（1反復あたり）

$$(6t_A + 9t_M) \cdot \lceil LMN/K \rceil$$

② $2\mathbf{h} + r\,\mathbf{f}^{(m)} - X\mathbf{f}^{(m)} - Y\mathbf{f}^{(m)} - Z\mathbf{f}^{(m)}$ の計算

$$(4t_A + 2t_M) \cdot \lceil LMN/K \rceil$$

③ (3－20)式の右辺の計算

$$2t_A \cdot \lceil LMN/K \rceil$$

④ (3－20) 式の解 (4.3 SLOR の②③④⑤を参照)

$$(2t_A+2t_M+t_D)\cdot L\cdot\lceil MN/K\rceil$$

$$+(t_A+t_M+t_D)\cdot L\cdot\lceil MN/K\rceil^{*}\qquad *最初の1回のみ$$

⑤ $Xu_{m+1}^{(1)}$ の計算

$$(2t_A+3t_M)\cdot\lceil LMN/K\rceil$$

⑥ (3－21) 式右辺の計算

$$2t_A\cdot\lceil LMN/K\rceil$$

⑦ (3－21) 式の解

$$(2t_A+2t_M+t_D)\,MN\cdot\lceil L/K\rceil$$

$$+(t_A+t_M+t_D)\,MN\cdot\lceil L/K\rceil^{*}$$

⑧ $Yu_{m+1}^{(2)}$ の計算

$$(2t_A+3t_M)\cdot\lceil LMN/K\rceil$$

⑨ (3－22) 式の右辺の計算

$$2t_A\lceil LMN/K\rceil$$

⑩ (3－22) 式の解

$$(2t_A+2t_M+t_D)\,N\cdot\lceil LM/K\rceil$$

$$+(t_A+t_M+t_D)\,N\lceil LM/K\rceil^{*}$$

①－⑩ 合計

$$\{(20t_A+17t_M)\cdot\lceil LMN/K\rceil+(2t_A+2t_M+t_D)(L\cdot\lceil MN/K\rceil$$

$$+MN\cdot\lceil L/K\rceil+N\cdot\lceil LM/K\rceil)\}\cdot(反復回数)$$

$$+(t_A+t_M+t_D)(L\cdot\lceil MN/K\rceil+MN\cdot\lceil L/K\rceil+N\cdot\lceil LM/K\rceil)$$

3.7 Cyclic reduction アルゴリズム

Cyclic odd－even reduction アルゴリズムは，ブロック3対角行列を解くための手法として〔4〕で示された。

N×N次のブロック3対角行列の1次方程式 $\mathbf{Ax}=\mathbf{v}$ ，すなわち

$$e_j x_{j-1}+d_j x_j+f_j x_{j+1}=v_j,\quad j=1,\cdots,N,$$

$$N=2^{m+1}-1,\quad m\geqq 1,\quad e_1=f_N=0 \qquad (3-23)$$

に対し，奇数インデックスを持つ変数の消去を行うと次式を得る。ここで各 d_j, e_j, f_j は

n×n 行列で v_j, x_j は n ベクトルである。

$$(-e_{2j}d_{2j-1}^{-1}e_{2j-1})x_{2j-2}$$
$$+(d_{2j}-e_{2j}d_{2j-1}^{-1}f_{2j-1}-f_{2j}d_{2j+1}^{-1}e_{2j+1})x_{2j}+(-f_{2j}d_{2j+1}^{-1}f_{2j+1})x_{2j+2}$$
$$=v_{2j}-e_{2j}d_{2j-1}^{-1}v_{2j-1}-f_{2j}d_{2j+1}^{-1}v_{2j+1}\ . \quad (3-24)$$

新しい (3－24) 式は再びブロック 3 対角行列でその次数は N／2 次元である。同様の操作をくり返していく。

$A^{\circ}=A$, $x^{(0)}=x$, $v^{(0)}=v$ とおき, $A^{(i)}x^{(i)}=v^{(i)}$

を計算する。ここで $A^{(i)}$ は $N_i\times N_i$ $(N_i=2^{m-i+1}-1)$ 次元の 3 対角ブロック行列である。$\log_2 N$ (=m) 回の操作で, 1 ブロックから成る $A^{(m)}$ が残り,

$$A^{(m)}\ x^{(m)}=v^{(m)} \quad (3-25)$$

より $x^{(m)}$ が計算される。$x=x^{(0)}$ は $x^{(m)}$ を使用して後進代入により求める。

すなわち,

$i=0, 1, \cdots, m-1$, $j=1, 2, \cdots, N_{i+1}$ に対し,

$$e_j^{(i+1)}=-e_{2j}^{(i)}(d_{2j-1}^{(i)})^{-1}e_{2j-1}^{(i)}\ ,$$

$$d_j^{(i+1)}=d_{2j}^{(i)}-e_{2j}^{(i)}(d_{2j-1}^{(i)})^{-1}f_{2j-1}^{(i)}-f_{2j}^{(i)}(d_{2j+1}^{(i)})^{-1}e_{2j+1}^{(i)}\ ,$$

$$f_j^{(i+1)}=-f_{2j}^{(i)}(d_{2j+1}^{(i)})^{-1}f_{2j+1}^{(i)}\ ,$$

$$x_j^{(i+1)}=x_{2j}^{(i)}\ ,$$

$$v_j^{(i+1)}=v_{2j}^{(i)}-e_{2j}^{(i)}(d_{2j-1}^{(i)})^{-1}v_{2j-1}^{(i)}-f_{2j}^{(i)}(d_{2j+1}^{(i)})^{-1}v_{2j+1}^{(i)}\ . \quad (3-26)$$

$y^{(m)}=x^{(m)}$ とおき, 次の手順で $y^{(m-1)}$, $y^{(m-2)}$, $\cdots$, $y^{(0)}$ を求める。$y^{(0)}$ が求める x である。

$$y_{2j}^{(i)}=y_j^{(i+1)}\ ,\quad j=1,\cdots,N_{i+1}\ ,$$

$$y_1^{(i)}=(d_1^{(i)})^{-1}(v_1^{(i)}-f_1^{(i)}y_2^{(i)})\ ,$$

$$y_{2j-1}^{(i)}=(d_{2j-1}^{(i)})^{-1}(v_{2j-1}^{(i)}-e_{2j-1}^{(i)}y_{2j-1}^{(i)}-f_{2j-1}^{(i)}y_{2j-1}^{(i)}),\ j=2,\cdots,N_{i+1}\ ,$$

$$y_{Ni}^{(i)}=(d_{Ni}^{(i)})^{-1}(v_{Ni}^{(i)}-e_{Ni}^{(i)}y_{Ni-1}^{(i)}). \quad (3-27)$$

(3－26)，(3－27) 式はj について並列計算が可能である。

3次元計算では，(2－6) 式で示されるように2重にブロック化された3対角行列になっている。したがって，(3－23) 式のd_j, e_j, f_jが2重にブロック化された行列に対応する。(3－24) 式および(3－26)，(3－27) のアルゴリズムでは$d_{2j-1}^{(i)}$及び$d_{2j+1}^{(i)}$の逆行列の計算が必要であり，$d_j^{(i)}$がML ×ML 行列でしかも，i＝0，1，2…，と進むに従って密行列となるから，逆行列の計算に時間を費さねばならない。直接法と同様に並列化しても効果がない。

3.8 各種解法の比較

3次元L×M×Nメッシュで K 個のプロセッサで並列計算した時の演算ステップ数をまとめるとTable 2のようになる。Cyclic reduction アルゴリズムについては明らかに直接法と同程度の計算時間がかかるので演算ステップ数を数えなかった。この結果次の結論を得る。

(1) 直接法

(i) 計算途中から大次元でスパースでない行列の逆転が必要となり，これに時間を費す。

(ii) 行列逆転時のまるめ誤差が大きくなり，解が安定しない。

(2) 反復法

(i) 元々の3次元拡散方程式はスパース行列であり，この特徴を生かした計算の方が演算数が少なくなる。

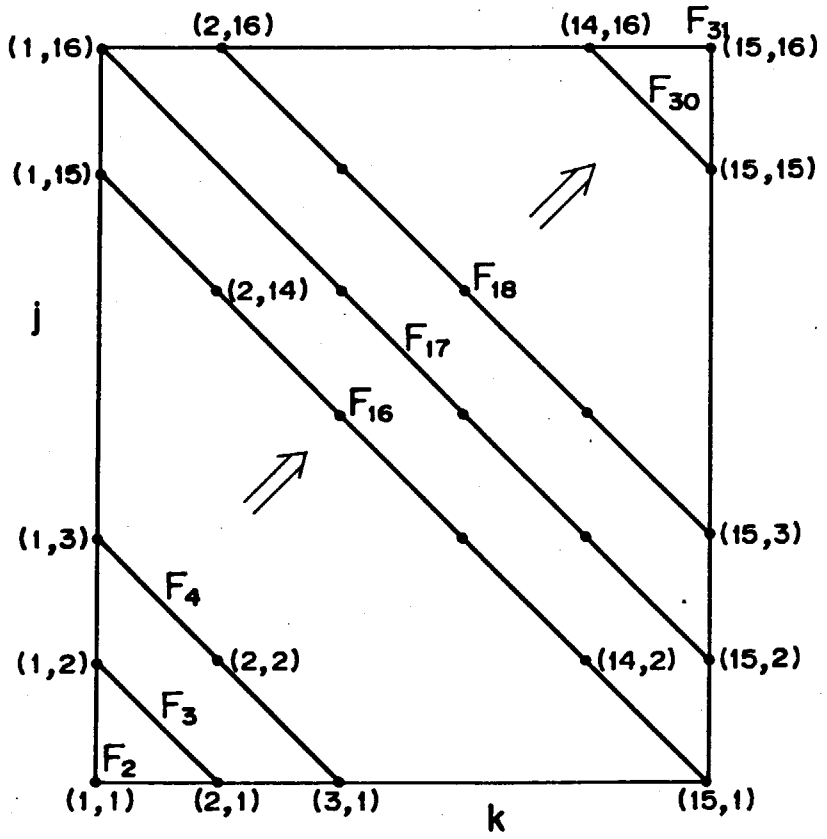
(ii) 反復数については，並列計算により，前回の計算値しか利用できない場合が多いので増加する傾向にある。これについては，数値実験をしてみる必要がある（次章）。

Table 2では例としてCRAY－1タイプの計算機を仮定して，数値を入れて計算した結果が第1欄に示されており，直接法がかなり不利なことが解る。

Table 2 Time step counting for parallel methods

Method		Time steps (L×M×N meshes, K-processors)	L=16, M-16, N=15 ( ) Num. of iters.	
SOR	Scalar	$(7t_A+8t_M+t_D)LMN\cdot$(Num. of iters.)	652,800 (10)	1.0
	Parallel to i	$(7t_A+8t_M+t_D)MN\cdot[L/K]\cdot$(Num. of iters.)	179,520 (11)	0.27
	Parallel to (i,j)	$(7t_A+8t_M+t_D)N\cdot[LM/K]\cdot$(Num. of iters.)	106,080 (13)	0.16
SLOR	Scalar	$(7t_A+8t_M+t_D)LMN\cdot$(Num. of iters.) + $(t_A+t_M+t_D)LMN$	668,160 (10)	1.03
	Parallel to k	$\{(7t_A+8t_M+t_D)$(Num. of iters.) + $(t_A+t_M+t_D)\}LM\cdot[N/K]$	183,360 (11)	0.28
	Parallel to (j,k)	$\{(7t_A+8t_M+t_D)$(Num. of iters.) + $(t_A+t_M+t_D)\}L\cdot[MN/K]$	108,160 (13)	0.17
Modified SLOR		$\{(M+N)(5t_A+6t_M)\cdot[1/2\ LN/K]+(M+N)L(2t_A+2t_M+t_D)\cdot[1/2\ N/K]\}$(Num. of iters.)$+(M+N)L(t_A+t_M+t_D)[1/2\ N/K]$	114,638 (10)	0.18
ADI		$(20t_A+17t_M)\cdot[LMN/K]\cdot$(Num. of iters.)$+\{(2t_A+3t_M+t_D)\cdot$(Num. of iters.)$+(t_A+t_M+t_D)\}\cdot(L\cdot[MN/K]+MN\cdot[L/K]+N\cdot[LM/K])$	257,280 (8)	0.39
Direct Method		$N\{1/2(LM)^2(t_A+t_M)+LM(2t_A+2t_M+t_D)+(7t_A+6t_M+t_D)\}\cdot[LM/K]$	32,201,250	49.3

- $[N/K]$ Multiplier to the time steps when N-dimensional vector is computed with K-processors.
- In the column 3, time steps are counted for a CRAY-1 type computer, where $[N/K] = N/4$ for $N \leq 20$ and $N/8$ for $N > 20$ and $t_A=t_M=1$, $t_D=2$.



$$F_2 = (f_{1,2}) \qquad F_4 = \begin{pmatrix} f_{1,3} \\ f_{2,2} \\ f_{3,1} \end{pmatrix} \qquad F_{17} = \begin{pmatrix} f_{1,16} \\ f_{2,15} \\ \vdots \\ f_{15,2} \end{pmatrix} \qquad F_{31} = (f_{15,16})$$

$$F_3 = \begin{pmatrix} f_{1,2} \\ f_{2,1} \end{pmatrix}$$

Fig. 6 Slanting direction mesh

4. 反復解法に関する数値実験

4.1 解法概略

中性子拡散コードADCのデータを用いて，7点階差式を解く問題についての数値実験を行なう。ここでは，3章の考察から直接解法よりも反復法に重点を置き，SOR，SLOR，改良SLOR，ADI法による計算を行う。計算を並列化したとしてスカラー計算の場合と比較して計算時間がどれ程短縮されたかを数値実験するのが目的である。

4.2 SOR，SLOR

(1) 解法概要

A. SOR法

まずx, y, zの座標を空間メッシュ上の点i, j, kに対応させる。ここで$1 \leqq i, j \leqq 16$, $1 \leqq k \leqq 15$である。従って，中性子束：fは

$$f^{(n)}(x,y,z) \rightarrow f^{(n)}(i,j,k) \qquad (n)：反復回数$$

3次元7点階差によって差分化された中性子束の式を整理すれば，

$$\begin{aligned}\tilde{f}^{(n)}(i,j,k) = S(i,j,k) &- C_6(i,j,k) f^{(m-1)}(i,j,k+1) \\ &- C_4(i,j,k) f^{(n-1)}(i,j+1,k) - C_5(i,j,k) f^{(n)}(i,j,k-1) \\ &- C_3(i,j,k) f^{(n)}(i,j-1,k) - C_2(i,j,k) f^{(n-1)}(i+1,j,k) \\ &- C_1(i,j,k) f^{(n)}(i-1,j,k) \qquad (4-1)\end{aligned}$$

求められた$f^{(n)}(i,j,k)$と適当な緩和因子：ωを用いて，次式で示される計算により反復を繰り返せば，$f^{(n)}(i,j,k)$はしだいに収束してゆく。

$$f^{(n)}(i,j,k) = \omega f^{(n)}(i,j,k) + (1-\omega) f^{(n-1)}(i,j,k) \qquad (4-2)$$

このとき収束条件は

$$\max \left| \frac{f^{(n-1)}(i,j,k)}{f^{(n)}(i,j,k)} - 1 \right| < \varepsilon (= 10^{-4}) \qquad -Ⓐ$$

この手法を以下3つのケースで数値実験する。

(i) 通常のスカラー処理 － (4－1)(4－2)をⒶの収束条件で解く。

(ii) i 方向に並列処理 － (4－1) の右辺第7項の $f^{(n)}(i-1,j,k)$ は $f^{(n)}(i,j,k)$ と同時に計算されるため古い $f^{(n-1)}(i-1,j,k)$ が用いられる。従って収束が遅れ反復回数が増加してしまう可能性がある。

(iii) (i,j)方向に並列処理 － (4－1) の右辺第5及び第7項 $f^{(n)}(i,j-1,k)$, $f^{(n)}(i-1,j,k)$ がそれぞれ $f^{(n-1)}(i,j-1,k)$, $f^{(n-1)}(i-1,j,k)$ に変わり同様に収束を遅らせる可能性がある。

B. SLOR法

$$\left.\begin{aligned}\tilde{f}^{(n)}(16,j,k) &= v(16,j,k)\\ \tilde{f}^{(n)}(i,j,k) &= v(i,j,k)+u(i,j,k)f(i+1,j,k)\end{aligned}\right\} \qquad (4-3)$$

$$(1 \leqq i \leqq 15)$$

ここで

$$\left.\begin{aligned}a(i,j,k) &= -C_7(i,j,k)\,C_6(i,j,k)\\ b(i,j,k) &= -C_7(i,j,k)\,C_4(i,j,k)\\ c(i,j,k) &= -C_7(i,j,k)\,C_5(i,j,k)\\ d(i,j,k) &= -C_7(i,j,k)\,C_3(i,j,k)\\ e(i,j,k) &= C_7(i,j,k)\\ h(i,j,k) &= C_7(i,j,k)\,S(i,j,k)\\ p(i,j,k) &= -C_7(i,j,k)\,C_2(i,j,k)\\ q(i,j,k) &= -C_7(i,j,k)\,C_1(i,j,k)\end{aligned}\right\}$$

$$\left.\begin{aligned}u(i,j,k) &= P(i,j,k)/e(i,j,k)\\ u(i,j,k) &= P(i,j,k)/(e(i,j,k)-q(i,j,k)u(i-1,j,k))\end{aligned}\right\}$$

$$(2 \leqq i \leqq 15)$$

$$\begin{aligned}k(i,j,k) = {} & c(i,j,k)\,f^{(n)}(i,j,k-1)+d(i,j,k)\,f^{(n)}(i,j-1,k)\\ & +b(i,j,k)\,f^{(n-1)}(i,j+1,k)+a(i,j,k)\,f^{(n-1)}(i,j,k+1)\\ & +h(i,j,k)\end{aligned} \qquad (4-5)$$

$$\left.\begin{aligned}v(1,j,k) &= k(1,j,k)/e(1,j,k)\\ v(i,j,k) &= (k(i,j,k)+q(i,j,k)\,v(i-1,j,k))\\ &\quad /(e(i,j,k)-q(i,j,k)\,u(i-1,j,k))\end{aligned}\right\} \qquad (4-6)$$

$$(2 \leqq i \leqq 16)$$

(4－5), (4－6) を用いて (4－3) を計算。そして (4－2) の式で反復を繰り返し，収束の判定条件式により中性子束：$f^{(n)}(i,j,k)$ を求める。

SLOR法の場合にも以下3つのケースの数値実験をする。

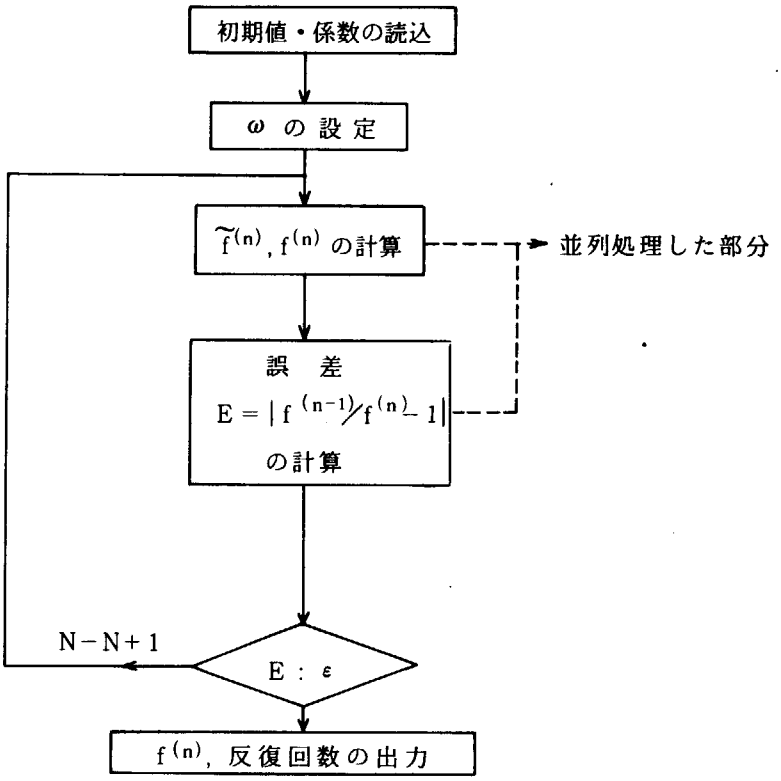
(i) 通常のスカラー処理 － (4－3)－(4－6)と(4－2)を解く。

(ii) k方向に並列処理 － (4－5)の右辺第一項の $f^{(n)}(i,j,k-1)$ が $f^{(n-1)}(i,j,k-1)$ に変わり収束が遅れる可能性がある。

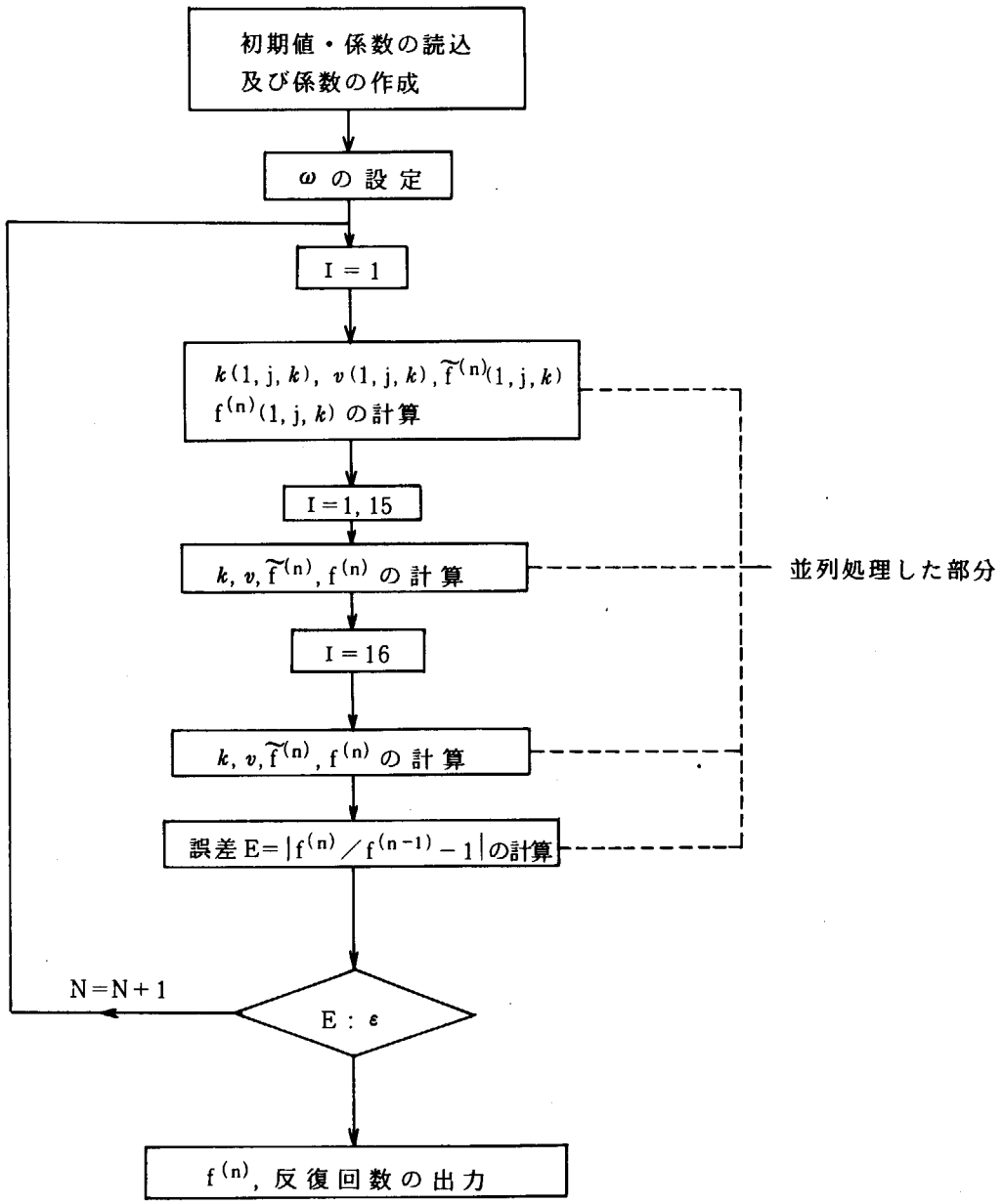
(iii) (j,k)方向に並列処理 － (4－5)の右辺第一，第二項の $f^{(n)}(i,j,k-1)$, $f^{(n)}(i,j-1,k)$ が $f^{(n)}(1,j,k-1)$, $f^{(n-1)}(i,j-1,k)$ に変る。同様に収束が遅れる可能性がある。

(2) プログラム・フロー

A. SOR法



B. SLOR 法



(3) 並列化の条件

プログラムが並列処理された場合一演算当りの所要時間は何分の一かに減少する。しかしこの減少率は処理される配列の大きさによって異なる。本実験では計算機としてCRAY－1を想定しTable 1のスカラー，ベクトル演算間の速度化をもとに並列処理された時の計算時間を指定した。

計算時間化（14－16）を見れば，配列の長さが10程度では平均として約4倍強，100程度では8倍強であり多少のオーバーヘッドを考慮して，20回程度までを約4倍，それ以上の場合は8倍として計算した。

つまり

(i) 15～16個の配列を並列処理した場合，演算速度は4倍。

(ii) 16×15～16×16個の配列を並列処理した場合，演算速度は8倍。

以上のような仮定にもとづきFORTUNEの出力より計算時間の推定を行なう。

(4) 計算結果

① 反復回数について

実際計算させたSOR法及びSLOR法による中性子束：fの収束値はいずれの場合でも，あるいは緩和因子ωを変化させても小数点4桁位までは一致している。

しかし，一方反復回数：Nはωの値によって大きく異なっているし，同一のωの値でも，SOR法とSLOR法では値が変わる。ここでは，それぞれの手法及びωの値に対してのNの変化を調べる。そして，第1群g－1，及び第2群g－2の2つのケースに対して計算を試してみる。

≪考 察≫

(i) 各Tableに対し，最小の反復回数：N_{min}とその時のωの値を整理すると以下のようになる。

SOR (g－1)

スカラー処理	$N_{min}=27$	at $\omega=1.30$
i 方向並列処理	$N_{min}=53$	at $\omega=1.20$
(i,j)方向並列処理	$N_{min}=70$	at $\omega=1.15$

SLOR (g－1)

スカラー処理	$N_{min}=28$	at $\omega=1.35$
k 方向並列処理	$N_{min}=44$	at $\omega=1.25$
(i,k)方向並列処理	$N_{min}=60$	at $\omega=1.15$

SOL (g－2)

スカラー処理	$N_{min}=7$	at $\omega=1.05\sim1.15$
i 方向並列処理	$N_{min}=8$	at $\omega=1.10$
(i,j)方向並列処理	$N_{min}=10$	at $\omega=1.05\sim1.10$

SLOR (g－2)

スカラー処理	$N_{min}=7$	at $\omega=1.10$
k 方向並列処理	$N_{min}=8$	at $\omega=1.05\sim1.10$
j 方向並列処理	$N_{min}=10$	at $\omega=1.00\sim1.10$

- g－1，g－2いずれに対してもSOR法（Table 3，5）とSLOR法（Table 4，6)共にN_{min}の値はほぼ等しい。つまり同じ演算回数で計算が行なわれるならばいずれの方法でも結果はあまりかわらない。
- g－1，g－2いずれもスカラー処理をベクトル処理化すると，N_{min}は増加するが，g－1が特にその傾向が強い。

  従って，g－1はg－2にくらべ，ベクトル処理による計算の速度アップの効果が大きく減少する可能性がある。

② 並列処理による計算時間の短縮率について

CRY－1の演算速度による仮定(3)の(i)(ii)を用いて，並列処理の効果が高いと思われるg－2に対してSOR及びSLOR法による計算時間を推定する。この指定は，FORTUNEによる相対計算時間の結果をもとに行ない，並列処理の効果を定量的に評価する。

A．SOR法の場合

スカラー，i並列，ij並列処理のそれぞれに必要な計算時間の推定を行なう。ただしここでは(2)Aのプログラムフロー中の“$\tilde{f}^{(n)}$，$f^{(n)}$の計算”から“E：εの判定”までの時間を計算した。

又，それぞれのケースに対して，Table 5より

- スカラー処理　－　$N_{min}=7$　at　$\omega=1.05\sim1.15$
- i 方向並列処理　－　$N_{min}=8$　at　$\omega=1.10$
- ij 方向並列処理　－　$N_{min}=10$　at　$\omega=1.05\sim1.10$

さらに仮定(3)(i)，(ii)を考慮して計算を行なえば，Table 7，第1～3欄のようになる。

B．SLOR法の場合

(2)Bのプログラムフロー中の“$\tilde{f}^{(n)}$，$f^{(n)}$の計算”から“E：εの判定”までの時間の推定を行なう。

Aの場合と同様にTable 6より

- スカラー処理　－　$N_{min}=7$　at　$\omega=1.10$
- k 方向並列処理　－　$N_{min}=8$　at　$\omega=1.05\sim1.10$
- jk 方向並列処理　－　$N_{min}=10$　at　$\omega=1.00\sim1.10$

さらに，仮定(3) i) ii) を考慮すればTable 7 第4～5欄のようになる。

(5) 結　論

○ Table 7から次のようなことがわかる。

＊SOR法において，スカラー処理と比較して

i 方向に並列処理した場合，計算時間は3／10に短縮，計算スピードは3.3倍に改善された。

ij 方向に並列処理した場合，計算時間は約1／5に，計算スピードは約5倍に改善された。

＊SLOR法においても

k 方向に並列処理した場合，計算時間は約1／3に，計算スピードは約3倍に改善された。

ik 方向に並列処理した場合，計算時間は約2／9に，計算スピードは約4.5倍改善された。

以上の結果より，本実験においては，並列処理の有効性が示されたと言えそうである。

○ g－2のように条件のよいデータの場合には，反復回数が並列処理化されたことによって余り増加はしないけれどもg－1群のように並列処理化によって大きく反復回数が増加してしまうような（Table 3，5参照）データを扱う場合には必ずしも並列処理が有効であるとは断言できないと思われる。

○ 緩和因子(ω)は，行列のスペクトル半径(ρ)から求めた値（$\omega=1.8$）に較べてずっと小さい値をとる方が良いことがわかった。

4.3 改良SLOR

(3－11)－(3－16)式で示した方法により計算する。

(3－15)，(3－16)式を収束するまでくり返す。この方法による反復回数については，Table 8，9で示す。SORやSLORに比して並列計算による収束遅れの影響が少なく，スカラーの場合と同じ程度の反復回数で，きわめて良好な収束状態を保つ（Fig．7)。これは，斜め方向，すなわち，u＝j＋k，u＝2，3，… の順に計算されるので(3－12)式からも判るように，u－1については，直前に計算した値を用いることができるので，スカラーと同程度の新しいデータの利用が可能となっていることが理由となっている。この方法は，2次元の5点階差の場合にWave front method として〔8〕に記述されている。

たちの悪いデータ（g－1）に対し，緩和因子：ωの値にあまり左右されない反復回数で収束する点で利点がある。一般に，SORやSLOR法では，ωの値については，計算実行時は未知数であり，理論的にスペクトル半径から求める方法も過大すぎる傾向にあり，ωの選択が難しい。この方法はωの選択に巾がある点でも勝れている。

どの程度並列化の効果があるかに関しては，斜め方向（u＝j＋k）及び部分的にはi方向の並列計算が可能であることから，Table 7で示すように，2方向を並列化したSOR，SLORと同等であるスカラー計算に比して5倍のスピードアップとなる（Table 7，第8欄）

但し，データの並べかえなど，並列化への準備に多少時間を要すること，中性子束データを並べかえて保存するメモリーを別に要するなどオーバヘッドが加わる。

4.4 ADI

ADI法は(3－20)－(3－22)式の反復式を見てもわかるように，並列計算が可能である。しかしながらSORなどに比べて3段の反復式から成る冗長な計算アルゴリズムであり，並列化による高速化の利益を相殺している。計算速度は，SOR，SLORの1方向並列計算より劣る，つまりスカラー計算に比して約2倍強である（Table 7，第10欄）。加速因子rはこの実験においては0.2～0.5が適当である。また反復回数は，SOR，SLORのスカラー計算より10％程度少ない。

Table 3 SOR method (g－1)

処理法＼ω値	1.00	1.05	1.10	1.15	1.20	1.25	1.30	1.35	1.40
スカラー処理	53	48	43	38	－	30	27	32	37
	0.700	0.671	0.765	0.595	－	0.812	0.534	0.896	0.691
i方向に並列処理	70	65	61	57	53	57	64	－	－
	0.939	0.985	0.842	0.778	0.852	0.935	0.733	－	－
(ij)方向に並列処理	80	76	71	70	∞	∞	－	－	－
	0.982	0.806	0.978	0.604	－	－	－	－	－

上段：反復回数：N

下段：前回と今回の中性子束：fの差を規格化したもの（収束条件）〔$\times 10^{-4}$〕

Table 4 SLOR method (g − 1)

処理法＼ωの値	1.00	1.05	1.10	1.15	1.20	1.25	1.30	1.35	1.40
スカラー処理	50	46	43	39	36	33	30	28	35
	0.923	0.916	0.624	0.814	0.679	0.598	0.565	0.868	0.868
i 方向に	60	56	53	50	47	44	49	68	−
並列処理	0.861	0.907	0.745	0.665	0.662	0.700	0.925	0.801	−
jk 方向に	70	67	63	60	78	−	−	−	−
並列処理	0.923	0.743	0.885	0.883	0.997	−	−	−	−

Table 5 SOR method (g−2)

処理法＼ωの値	1.00	1.05	1.10	1.15	1.20	1.25	1.30
スカラー処理	8	7	7	7	8	9	10
	0.572	0.887	0.496	0.981	0.715	0.876	0.629
i 方向に	9	9	8	9	10	12	14
並列処理	0.744	0.353	0.734	0.677	0.899	0.654	0.801
(i, j)方向に	11	10	10	11	13	16	19
並列処理	0.515	0.629	0.649	0.877	0.770	0.620	0.750

Table 6 SLOR method (g − 2)

処理法＼ωの値	1.00	1.05	1.10	1.15	1.20	1.25	1.30	1.35
スカラー処理	9	8	7	8	9	9	11	12
	0.610	0.257	0.467	0.362	0.345	0.997	0.544	0.830
k 方向に	9	8	8	9	10	11	12	15
並列処理	0.534	0.687	0.515	0.362	0.536	0.629	0.996	0.615
(i,j)方向に	10	10	10	11	13	16	19	27
並列処理	0.753	0.334	0.553	0.959	0.875	0.553	0.637	0.913

Table 7 Estimate of time steps in parallel computation

Method / Estimated value	SOR Scalar	SOR Parallel to i	SOR Parallel to(i,j)	SLOR Scalar	SLOR Parallel to k	SLOR Parallel to(j,k)	Modified SLOR Scalar	Modified SLOR Parallel	ADI Scalar	ADI Parallel
Total time steps	2,576,553			3,022,804			2,719,297		7,085,656	
Steps/iteration	308,079	94,521	48,928	431,829	109,635	55,936	453,216	80,198	1,180,943	176,911
Total steps considering the increase of iterations	2,576,553	756,168	489,280	3,022,804	877,080	559,360	2,719,297	561,386	7,085,658	1,061,466
Number of iterations (Optimum value of ω)	7 (1.1)	8 (1.1)	10 (1.1)	7 (1.1)	8 (1.1)	10 (1.1)		7 (1.1)		6 (0.4)
Speed-up ratio	1.00	0.30	0.19	1.18	0.34	0.22	1.06	0.22	2.75	0.42

Time steps are measured by FORTUNE program

L = 16, M = 16 and N = 15 is used as the mesh size.

Table 8 Number of inner iterations (First outer iteration)

data	method \ w		0.2	0.4	0.6	0.8	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
F-1	SOR	scalar					53	43	34	(27)	38	51	-		
		i-parallel					70	61	(53)	64	-				
		(i,j)-parallel					80	(71)	-						
	SLOR	scalar					50	43	36	(30)	35	67	-		
		k-papallel					60	53	(47)	49	90	-			
		(j,k)-parallel					70	(63)	78	-					
	modified SLOR	u-parallel (u=j+k)					40	33	(27)	34	40	50			
	ADI		37	(23)	33	43	53		63		73		82		92
F-2	SOR	scalar					8	(7)	8	10	13	16	21	30	50
		i-parallel					9	(8)	10	12	17	28	98	-	
		(i,j)-parallel					11	(10)	13	19	42	-			
	SLOR	scala.					8	(7)	9	11	14	18	26	60	-
		k-parallel					9	(8)	10	12	17	28	98	-	
		(j,k)-parallel					(10)	(10)	13	19	69	-			
	modified SLOR	u-parallel (u=j+k)					(7)	(7)	8	10	13	17	22	32	54
	ADI		14	(6)	7	8	10		11		13		14		16
F-3	SOR	scaler					13	11	(10)	(10)	13	18	22	34	62
		i-parallel					15	14	(13)	19	26	69	-		
		(i,j)-parallel					18	(16)	21	61	-				
	SLOR	scalar					13	11	(10)	11	16	23	32	83	-
		k-parallel					15	13	(12)	16	27	93	-		
		(j,k)-parallel					17	(15)	21	42	-				
	modified SLOR	u-parallel (u=j+k)					(9)	10	11	15	19	22	42	60	-
	ADI		16	(8)	9	11	14		16		18		20		22

○ optimum, - diverge

Table 9 Number of inner iteration (Last outer iteration)

data	mothod \ w		0.2	0.4	0.6	0.8	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
L-1	SOR	scalar					26	23	(20)	21	27	-			
		i-parallel					32	29	(28)	58	-				
		(i,j)-parallel					34	(32)	78	-					
	SLOR	scalar					25	23	(20)	(20)	24	46	-		
		k-parallel					27	25	(23)	26	64	-			
		(j,k)-parallel					(29)	(29)	52	-					
	modified SLOR	u-parallel (u=j+k)					22	20	(18)	22	27	40	-		
	ADI		(7)	11	15	18	22		24		28		31		34
L-2	SOR	scalar					(4)	5	7	10	17	30	42		
		i-parallel					(4)	5	7	9	16	44	-		
		(i,j)-parallel					(5)	6	8	11	42	-			
	SLOR	scalar					(4)	5	7	9	12	21	32	-	
		k-parallel					(5)	(5)	7	10	16	28	67	-	
		(j,k)-parallel					(5)	6	8	12	23	-			
	modified SLOR	u-parallel (u=j+k)					(5)	6	8	11	19	27	36	50	-
	ADI		7	4	(3)	4	4		4		5		5		5
L-3	SOR	scalar					11	(10)	14	18	25	31	-		
		i-parallel					13	(12)	17	28	-				
		(i,j)-parallel					14	(13)	33	-					
	SLOR	scalar					11	(10)	12	15	19	26	49	-	
		k-parallel					11	(10)	15	20	29	58	-		
		(j,k)-parallel					(11)	12	32	59	-				
	modified SLOR	u-parallel (u=j+k)					(10)	12	15	19	23	29	40	-	
	ADI		7	(5)	7	8	10		11		12		13		14

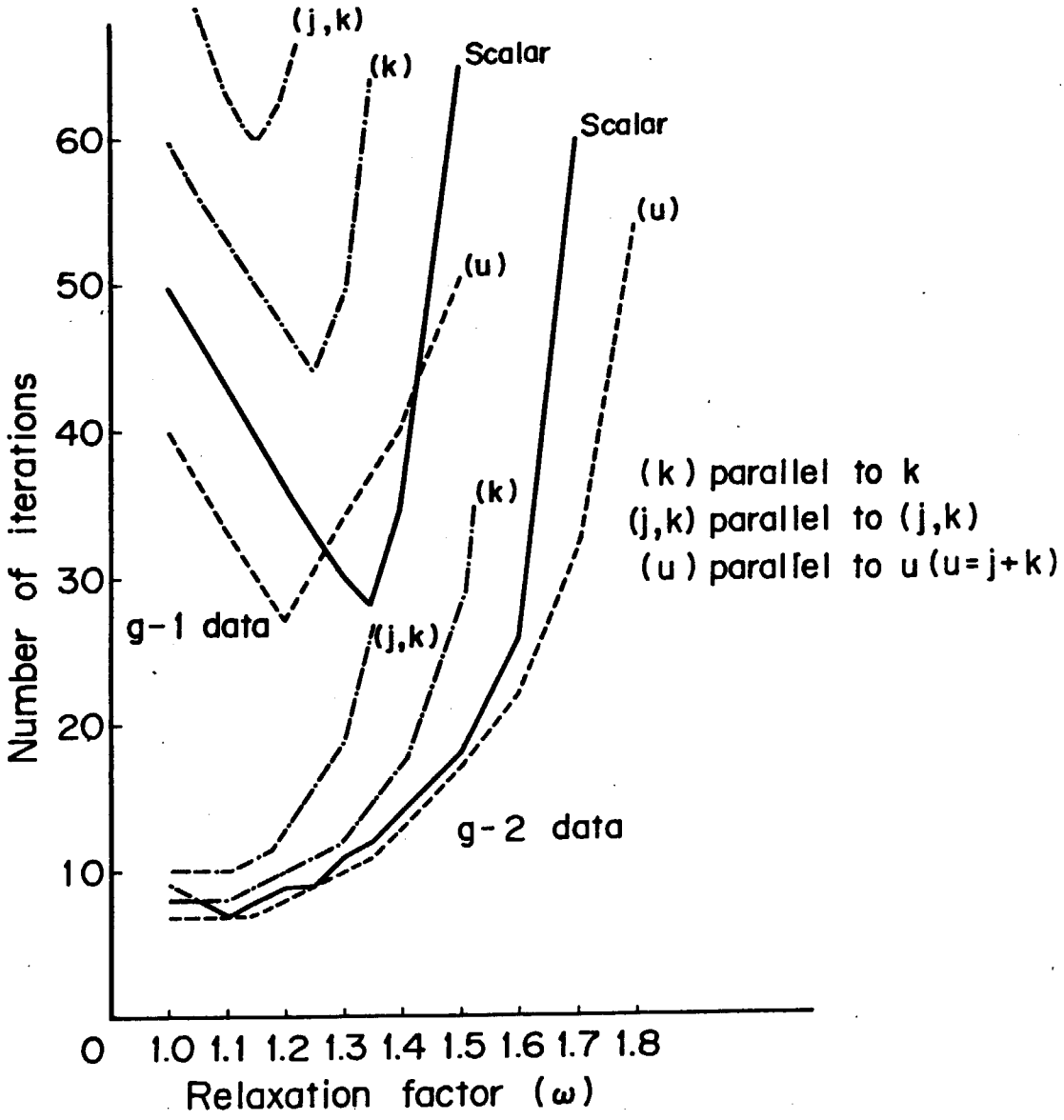


Fig. 7 Number of iterations for SLOR

5. 拡散コードADCの並列化

5.1 APCコード

ADCは1970年中ばに，当初，F230－60で開発され，F230－75で使用され，現在FACOM M200で利用可能である。並列化に関連するコードのプログラムとしての特徴は次のものが挙げられる。

(1) プログラムはF230－60のメモリー（256K語）に照準を合わせて作成され，小メモリー向きの構成となっている。群およびz方向のデータを一時的にディスクに保存しながら計算を進めていく。

(2) 差分方程式の解法は，inner／outer の2重ループ方式とEQUIPOISEで採用された1重ループ方式が選択できる。2重ループの場合のinner iterationはSORとSLOR法が選択でき，outer については，チエビセフ外挿法が主に使用されている。1重ループの場合には，SLORが使用されている。

5.2 ADCの実行時の振舞の分析

各問題に対し，90％程度費されるinner most の部分のサブルーチン名とそれらの計算時間比について，FORTUNEによる分析結果（例，Fig。8）をもとに表にしたのがTable 10である。Table 11には，関連するルーチンの機能と並列計算した場合どの程度計算時間が短縮されるかの推定値について示している。ベクトル／スカラー比は，FORTUNEのステートメント毎の相対計算時間，VECTORIZERによる各ステートメントのベクトル化への可能性，さらに各DOループの詳細分析に基づいて得た推定値である。

2重ループ方式（inner／outer iterations）では，inner iterationである7点階差式の計算，INNER1 またはINNER2 に90％近い計算時間が費されている（Fig。8）。この部分についてはSORの場合には並列化が簡単でその結果全計算時間は30％に短縮されると推定される。

5.3 各解法に対する並列効果

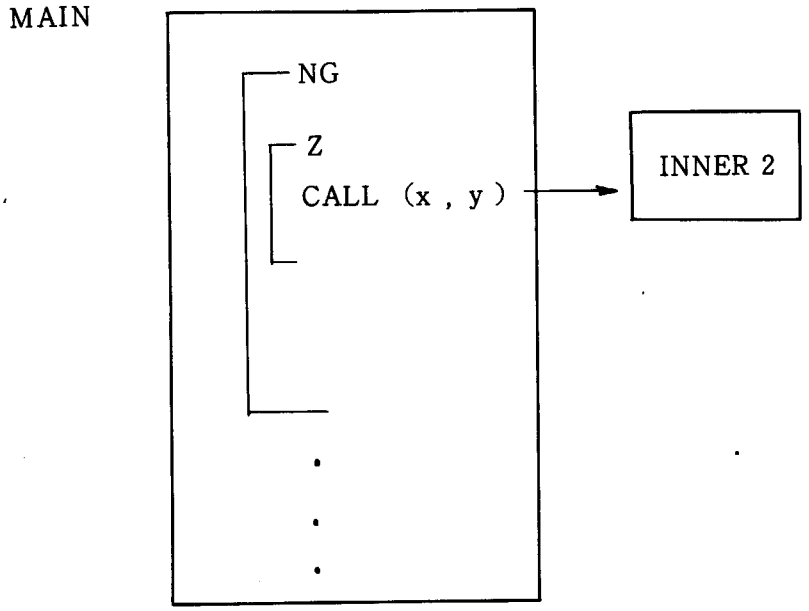
(1) SOR

SOR（INNER1 ）は40ステップ程度のサブルーチンで，容易に並列化できる。この場合x方向／y 方向を1次元化してベクトル長を大きくすれば，多少のオーバーヘッドを入れても20％に計算時間が短縮でき得る。全体的には，次の計算時間となる。

INNER1	（並列計算）	0.9×0.2
INNER1 以外	（逐次計算）	0.1
全プログラム		0.28

(2) SLORの場合

ところがSLORでは，既に示したように，いずれか1方向（INNER2 ではx方向）について逐次計算を行わねばならない解法となっている。ADCでは（x,y）方向の7点階差式の計算がサブルーチンINNER2 として処理され，群及び，z方向については主プログラム（MAIN）のループの中で次のように



呼ばれる構造となっている。従ってSLORの場合には，プログラムの構造を変更しないかぎり，y 方向の並列化しかできないが，既に述べたように，ベクトル長が10～20程度では，それ程並列化効果が上らない。

(3) 1重ループ方式は2重ループに比して計算時間がもともと3割程度短いのでADCでは，1重ループ方式が主に使用されている。1重ループ方式では，

中性子束の計算（7点階差式の計算）

固有値・中性子源の計算（7点階差式の定数項の計算）

のくり返しにより実行される。この方式は2重ループ方式の outer iteration に比して反復回数が多いので，中性子源計算に費される時間が多くなる。従って7点階差式の計算だけでなく中性子源の計算の部分の並列化も合わせて行なう必要がある。ADCでは，中性子源の計算は，MAINプログラムに含まれており（MAIN4），この部分は長くて複雑なことから，INNERを並列化する程簡単に並列化できない。かなりの調査が必要で3人月程度の人手を要することになるであろう。

Table 10 Execution-time behavior of ADC code

Problem No.	Geometry	Num. of Group	Num. of meshes	Method	Inner most
5	(x,y)	3	46×46	SLOR	INNER2 (97.9%)
7	(x,y,z)	3	16×16×15	one loop	INNER2 (63.8), MAIN4 (31.1)
7'	(x,y,z)	3	16×16×16	SLOR	INNER2 (89.5), SOURS (4.4)
7''	(x,y,z)	3	16×16×15	SOR	INNER1 (84.8), SOURS (6.5)
9	(γ,θ)	3	14×10	SLOR	INNER2 (97.1)

Table 11 Inner most routines and their effects on parallel processing

Name	rate of CPU time vector / scalar (estimation)	Functions
MAIN4	0.35	Controls for three dimensional calculation by one through iterations including the calculation for new fission source distribution
INNER1	0.2	Calculates flux distribution by pointwise over-relaxation method
INNER2	0.35	Calculates flux distribution by linewise over-relaxation method
SOURS	0.25	Calculates fission source and slowing down sources
MAIN3		Controls for three dimensional calculation by inner and outer iteration
OUTER		Calculates new fission source distribution

ROUTINE	EXECUTIONS	COST	%	0....1....2....3....4....5....6....7....8....9....
MAIN	1	7080	0.0	I
ADDED	0	0	0.0	I
ADJU	0	0	0.0	I
CDEF	45	3342394	0..	I
EDIT	0	0	0.0	I
FGCAL	0	0	0.0	I
FGAK	0	0	0.0	I
INNER1	0	0	0.0	I
INNER2	1980	430591440	89.5	I**
INPUT1	0	0	0.0	I
INPUT2	0	0	0.0	I
INPUT3	1	4667	0.0	I
MAIN1	0	0	0.0	I
MAIN2	0	0	0.0	I
MAIN3	1	10539727	2.2	I*
MAIN4	0	0	0.0	I
MAP	2	49430	0.0	I
OUT	25	1109265	0.2	I
OUTER	44	14327552	3.0	I*
OUTER1	0	0	0.0	I
OVERIN	132	73392	0.0	I
POUT	0	0	0.0	I
READ	0	0	0.0	I
READ	4	39599	0.0	I
IBCD	623	6544	0.0	I
REDIT	0	0	0.0	I
SOURS	1980	20961160	4.4	I**

Fig. 8 Running time behavior of ADC code (FORTUNE output)

6. 拡散方程式の直接解法の並列計算

2次元拡散方程式の直接解法の改良について桂木氏からIterative Matrix Factorization法（IMF法）として〔7〕で提案されたものについて並列計算による効果を検討するのが本章の課題である。Matrix Factorization（MF法）との比較により論じる。

6.1 手 法

(1) IMF法

2次元5点階差式を，　メッシュ ℓ，$1 \leqq \ell \leqq L$（Fig.9）に対し，

(i) 領域B_1，B_2，…，B_M間の中性子束の計算

(ii) 各領域内の中性子束の計算

に分離して，横方向メッシュk，$1 \leqq k \leqq K$に対し，$\phi_k = (\phi_{k,2}, \phi_{k,2}, \cdots, \phi_{k,L})$が収束するまで(i)，(ii)を反復計算する。(i)により領域の境界上の中性子束を求め，(ii)によりそれを初期値として各領域内の計算を行う。この場合MF法による直接解法を用いる。詳細は，文献〔7〕の(2)－(17)式を参照。

(2) MF法

KAK〔18〕で用いられた方法で，5点階差式をブロック3対角行列に置き，直接解く方法である。詳細は〔18〕の(8)－(19)式を参照。

MF法では，大次元行列の逆転が必要でこれに時間がかかる。IMFは行列が細分されているので逆転に要する時間が少ないという特徴を持つ。

MF法，IMF法共にk方向については，SLORと同様に逐次計算が必要である。従ってℓ方向の並列計算しかない。

6.2 並列計算の対象

(1) IMF法

(i) 領域に対して並列計算（領域数10～30？）

(ii) 領域内メッシュに対して並列計算（メッシュ数5～10？）

(i)，(ii)いずれも，それ程大きいベクトル長とはならない。

(2) MF法

ℓ方向のメッシュが全部同時に計算できて有利な場合が多い。

6.3 演算ステップ数

〔7〕で用いたのと同じ基準を用い，また「演算はベクトル長にかかわらず，すべて1回で行え

る」という非常に現実的でない仮定による。

(1) スカラー計算の場合

(a) MF法（〔7〕－(23)式より）

$$\frac{1}{2}L(L+1)(2L+5)+3L^2+2$$

(b) IMF法（〔7〕－(24)式より）

$\nu=\dfrac{L+2}{M}-2$ とおき

$$M\left\{\frac{1}{2}\nu(\nu+1)(2\nu+5)+3\nu^2+2\right\}+(2M-2)\left(\frac{K-1}{K}\nu+\frac{K-3}{2}\nu^2+K\nu^2+K\nu\right)$$

$$+(M-1)\{(2K+7)+(t-1)(2K+4)+t'(K-1)\}\ ,$$

t，t′は反復回数

(2) 並列計算

(c) MF法

$$L^2+6L+4$$

(d) IMF法領域に対して並列計算する場合

$$\frac{1}{2}(\nu+1)(2\nu+5)+3\nu^2+2+2\left(\frac{K-1}{K}\nu+\frac{K-3}{2}\nu^2+K\nu^2+K\nu\right)$$

$$+(2K+7)+(t-1)(2K+4)+t'(K-1)$$

(e) 領域内のメッシュに対して並列計算する場合

$$M(\nu^2+6\nu+4)+(2M-2)\left(\frac{K-1}{K}+\frac{K-3}{2}\nu+K\nu+K\right)$$

$$+9(M-1)+(t-1)\{6(M-1)+2(M-1)\}+t'(K-1)$$

6.4 考 察

(a)－(e)の結果をL＝30，60，100について図示すれば，Fig.10，Fig.11，Fig.12のようになる。

(1) 領域数が多ければ，IMF法－(d)が有利である。そうでなければMF法が良い。

(2) 並列計算により1桁以上速くなるというこれらの図からの結果については，そのまま信じない方が良い。ここでは，あまりにも理想化されたハードウェアを仮定しており，CRAY－1のようなパイプライン計算機には適合しない。2～5章で述べた結果がより現実的である。

(3) 直接法を3次元の問題に適用するのは得策ではない。（3.8参照）

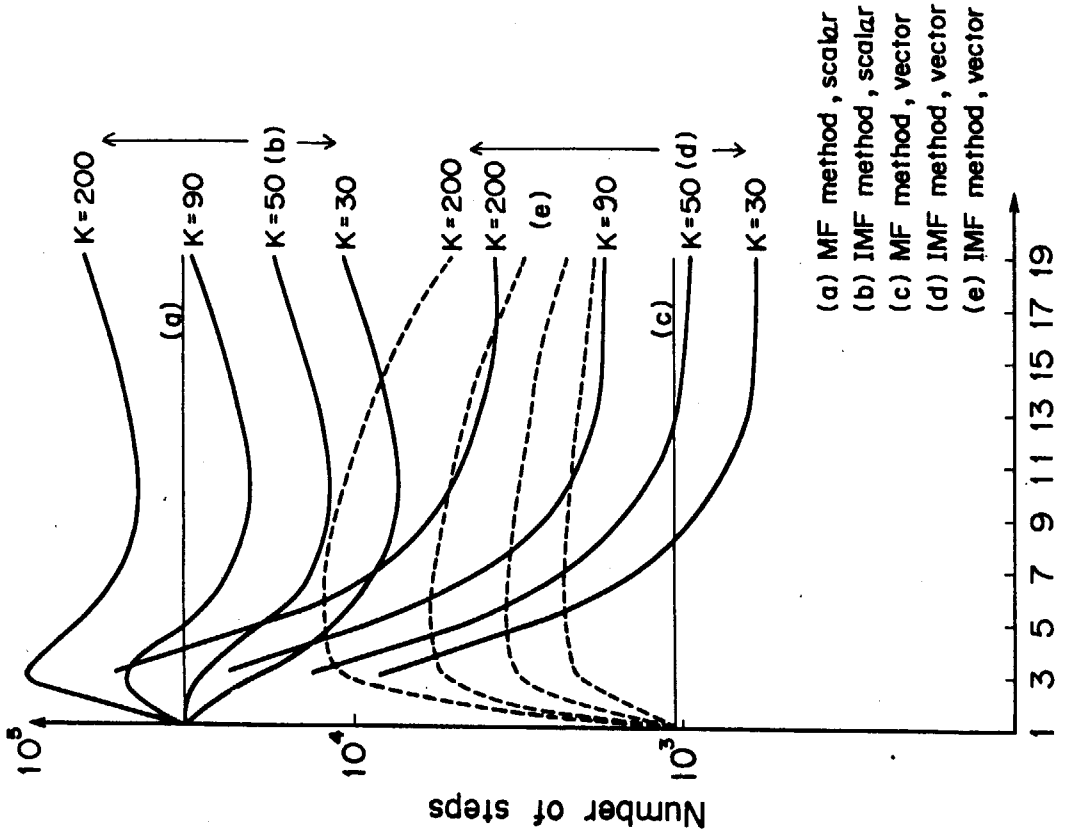


Fig.10 Number of steps (L=30, t=8, t'=3)

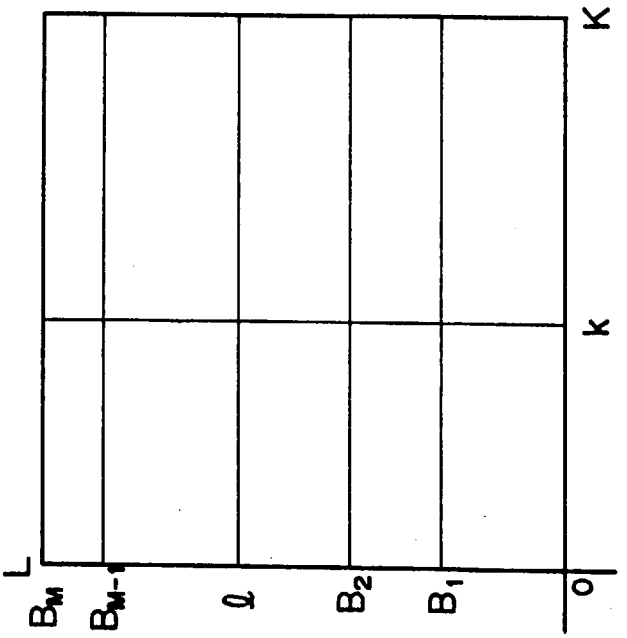


Fig.9 Two dimensional mesh

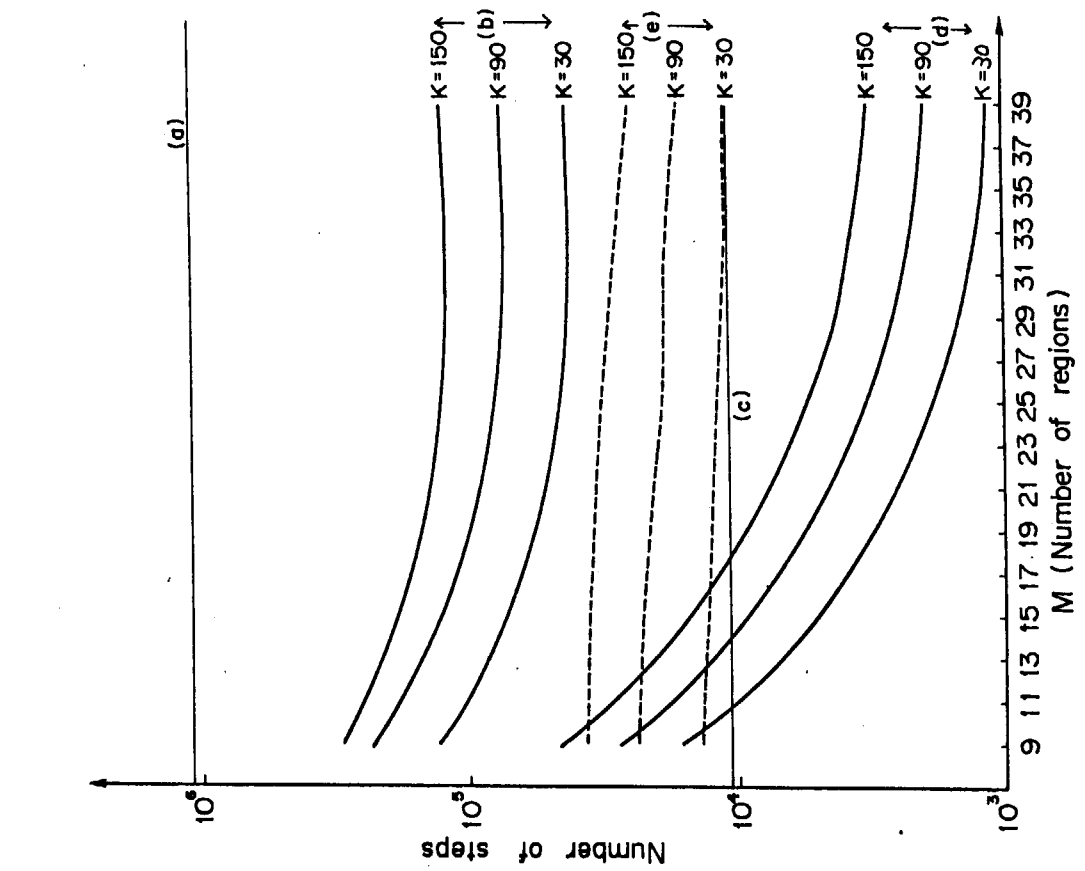


Fig.12 Number of steps (L=100, t=8, t'=3)

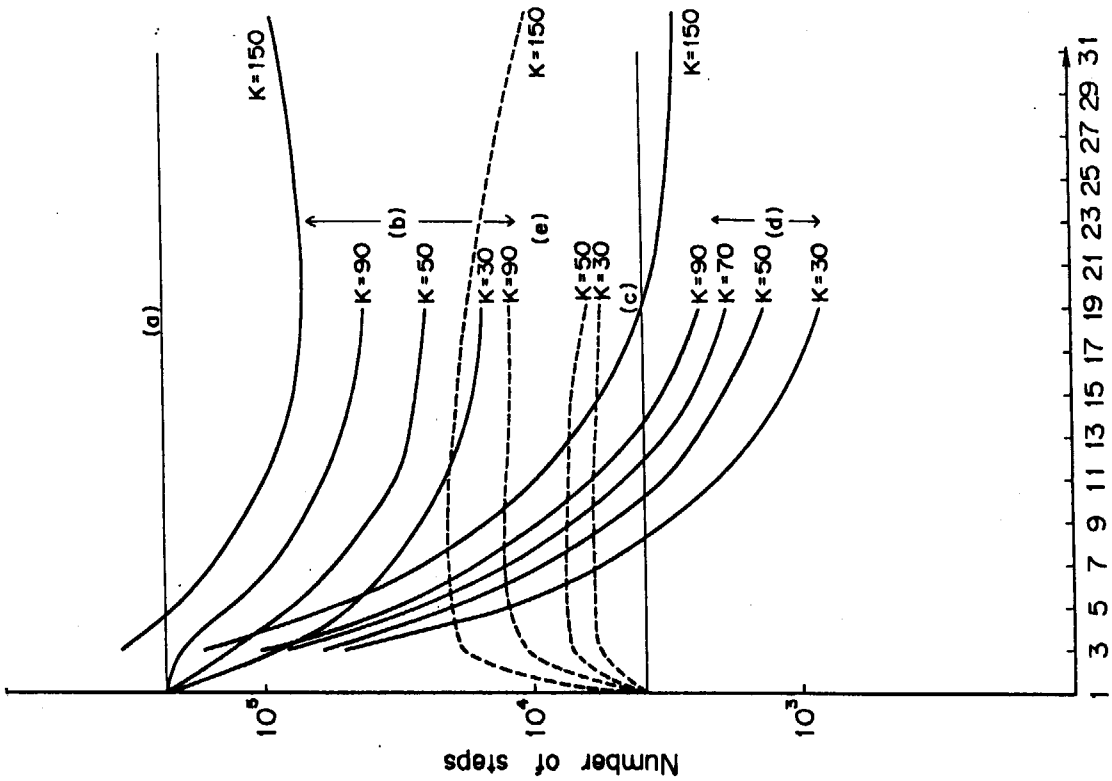


Fig.11 Number of steps (L=60, t=8, t'=3)

7. おわりに

拡散方程式の並列計算については，今回差分近似による場合について論じてきた。拡散コードでは，中性子拡散の計算の部分がinner most loopとなり，大部分の計算時間が費されるという並列計算による効果が大いに期待できるプログラム・パターンとなっている。最初から並列計算用にプログラムを作成すれば，計算時間は1／5程度に短縮することが可能であるだろう。しかしながら，既成の拡散コードでは，メモリーの制約から，3次元の問題では，このinner most loopの部分を全メッシュに対して1つのサブルーチンで計算していない場合が多い。一定のデータ領域の中で補助メモリーへ掃出しながら計算を進めるのが現実である。このように入出力処理が多い状態でCPU時間の短縮を計っても，それ程有効ではない。並列計算の効果を高めるには，充分大きいメモリーが使用できることが前提となる。

若し，3次元の問題で全メッシュを一度に並列計算するとすれば，差分近似の場合には，

（メッシュ数）×（係数の種類）×（群数）

が必要となる。ここで係数の種類は，並列計算のために余分に必要となるものを考慮すると15種程度になる。したがって10,000メッシュで3群の場合の計算で，450K語（約2Mバイト）のデータ領域が必要となる。

しかしながら，メモリーは安価になり，最大32Mバイト実装する大型計算機も出現しているのでこの量は今後利用可能となるであろう。

原研でよく利用されている拡散コードCITATIONや今後それに代るものとなるVENTUREについての並列計算が期待される。これらのコードは，始めて並列化についての分析を行うには，大きくて複雑すぎて，問題の本質に追るまで時間を要するものと思われた。この分野の経験と知識に乏しい筆者にとって理解の範囲を越えないこと，またある程度簡単に並列化できる見通しが得られることなどを理由に，今回は，原研で開発され，コードの作者からの協力が得られ易いADCコードを分析の対象に選んだ。

ADCコードを，本格的に並列計算用に書直すことは，利用者があまり多くないことから今回行わないつもりである。しかしながら，ADCで得られた分析結果を確めるために，F75 APUが入り次第INNER1（過大緩和法による中性子束の計算）を並列化して実験してみたい。

反復解法では，緩和因子の選び方により収束の良し悪しが左右される。並列計算によりこの緩和因子をどう決めれば最適となるのかについて解析的に求めることは興味のある課題となる。

また並列計算用数値計算アルゴリズムの検討と評価，一般の（大型）プログラムの並列計算技法の確立など今後この分野での取組べき課題が山積しているので，研究を続けたいと思っている。

謝　　辞

拡散コード：ＡＤＣの利用にあたり，安全性コード開発室の秋元正幸氏の適切な助言と討論に感謝します。また，２次元拡散方程式の直接解法について並列計算への動機づけをいただきました桂木学 安全解析部部長に感謝します。

富士通の松浦俊彦氏には，アレイプロセッサの使用経験に基づく種々の助言といくつかの問題点の示唆をいただきましたことを感謝します。また富士通の原口誠氏にはＡＤＣ コードのFORTUNEによる分析を，また筑波大学から夏期実習生として来られた難波克光君には，数値実験の一部分を分担していただいたことを合わせて感謝します。

最後に計算センター室長代理浅井清氏には，全般にわたる助言と仕事の環境の設定に助力いただきましたことを感謝いたします。

参 考 文 献

[1] Derstine K.L., "An Experimence with the Conversion of the Large-scale Production Code DIF3D to the CRAY-1", CONF 790902, ANL (1979).

[2] " 次期大型計算機検討会資料 " （1979年４月），（内部資料）。

[3] Varga R.S.，" 大型行列の反復解法 "，サイエンス社（1972）。

[4] Heller D., "Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems", SIAM J. Numerical Analysis, 13-4 (1976).

[5] Akimoto M. and Naito Y., " A General Dimensional Neutron Diffusion Calculation Code : ADC", JAERI-1260 (1978).

[6] 富士通マニュアル，" FORTRAN HE 使用手引書 "，P160 － 167，富士通（1978）。

[7] 桂木 学，" ２次元拡散に対するマトリックス・ファクトリゼーション法の改良 "（1968），（内部資料）。

[8] Kuck D.J., Muraoka Y., and Chen SC., "On the Number of Operations Simultaneously Executable in Fortran-like Programs and theisr Resulting Speedup", IEEE Trans. on Computers, C21-12 (1972).

[9] Vondy D.R., Fowler T.B., and Cunningham G.W., "VENTURE : A Code Block for Solving Multi-group Neutronics Problems Applying the Finite-Difference Diffusion-Theory Approximation to Neutron Transport", ORNL-5062 (1975).

[10] Enslow JR. P.H., "Multiprocessor Organization-A Survey", Computing Surveys, 9-1 (1977).

[11] Romamoorthy C.V., "Pipeline Architecture", Computing Surveys, 9-1 (1977).

[12] Yau S.S. and Fung H.S., "Associative Processor Architecture-A Survey", Computing Surveys, 9-1 (1977).

[13] Russell R.M., "The CRAY-1 Computer System", Comm. ACM, 21-1 (1978).

[14] "CARY－1セミナー・テキスト"，センチュリー・リサーチ・センタ株式会社（1980）。

[15] 松浦俊彦（内部資料）。

[16] 富士通マニュアル，"AP－FORTRAN文法書"，富士通。

[17] Greenspan H., Kelber C.N., and Okrent D., "Computing Methods in Reactor Physics", Gordon and Breach Science Publishers (1968).

[18] Akanuma M. et al., "The KAK Program for the Numerical Solution of Few-Group Neutron Diffusion Equations in Two Dimensions", JAERI 1127 (1966).

[19] Kuck D.J., "A Survey of Parallel Machine Organization and Programming", Computing Surveys, 9-1 (1977).

[20] Heller D., "A Survey of Parallel Algonithms in Numerical Linear Algebra", SIAM Review, 20-4 (1978).