第 4 章
制御構造変換にもとづくベクトル化 — 2 くりかえし構造交換法の比較評価

要旨

リスト処理やデータ変換などのプログラムにおける可変長の内側くりかえしをふくむ 2 重のくりかえし構造 (ループ，再帰よびだしなど) に，変換前の内側くりかえし回数の最大値をあらかじめもとめることによって前章で定義したくりかえし構造の交換を適用する方法を開発し，それを S-810 において他の方法と比較した．その結果，交換によって十分なくりかえし回数がえられるときには，おおくのばあいこの方法による実行が他の方法より高速であることなどがわかった．この結果は，今後くりかえし構造の交換をつかった具体的なベクトル記号処理アルゴリズムの開発の参考になるとともに，論理型言語などの，最適化をともなう自動ベクトル化処理系の設計の基礎になるとかんがえられる．

4.1 はじめに

数値計算プログラムにおいて，ループ内の処理にループ依存性があるためにベクトル化できないかまたはベクトル化しても加速率がひくいばあいがある．このようなループをふくむ 2 重ループをベクトル化可能にするプログラム変換技法として，前章でのべた**ループ交換**がある．ループ交換がおこなえるための必要条件は，変換前の内側ループのくりかえし回数がループの実行開始前に確定することである．数値計算におけるループは，通常この必要条件をみたしている．

一方，記号処理プログラムにおいても，ループ交換やそれを再帰よびだしなどに拡張したプログラム変換戦略であるくりかえし構造の交換は，ベクトル化において有効な方法である．ここでくりかえし構造の交換とは，前章でのべたように，多重のくりかえし構造における内側のくりかえしと外側のくりかえしとをいれかえることによってベクトル処理できないプログラムをベクトル処理可能にするための，またはベクトル処理性能を向上させるためのプログラム変換戦略のことである．くりかえし構造の変換によって，図 3.2 (1.1) に PAD (Problem Analysis Diagram) をつかってしめしたようなプログラムが図 3.2 (1.2) にしめしたようなプログラムに変換される．もちろん，くりかえし構造の交換ができるためにはデータフロー等に関して一定の条件がなりたっていなければならないが，それについてはここでは省略する．

ところで，リストのような可変長のデータ構造，木やグラフのような可変構造のデータ構造をあつかう記号処理においては，リストをたぐりながらの処理のように内側のくりかえしがそのままではベクトル化できず，しかもそのくりかえし回数が実行開始前に確定しないばあいがおおい．このようなばあい，交換前の内側ループが可変長であるとすると，すなわち内側ループのくりかえし回数が外側ループのくりかえしごとにことなるとすると，内側ループと外側ループの制御変数をいれかえるだけの単純なループ交換ではプログラムが不正になってしまう．第 3 章ではくりかえし構造の交換を適用してこのようなリスト処理プログラムのベクトル化をはかっているが，この報告ではそこで暗につかわれている具体的なベクトル処理技法を明確化するとともに，その代案をしめして，配列の線形検索に関する実測データにもとづいて両者を比較検討する．この比較結果は，今後くりかえし構造の交換をつかった具体的なベクトル記号処理アルゴリズムの開発の参考になるとともに，論理型言語などの自動ベクトル化処理系の設計の基礎になるとかんがえられる．

4.2 節では，くりかえし構造変換が必要な可変長くりかえしをふくむ応用の例をしめす．4.3 節では，可変長くりかえし構造の交換にもとづく 2 つのベクトル化のためのプログラム変換方法をしめす．4.4 節では，4.3 節の 2 つの方法を配列の線形検索に適用したプ

ログラムをしめし，4.5 節でその実測データをしめし，それにもとづいて両者を比較する．4.6 節では関連研究についてのべる．

4.2 くりかえし構造交換が必要な可変長くりかえしの例

この節では，この報告の主題であるくりかえし構造交換の有用性をしめすため，もとのままではベクトル化に適さないがくりかえし構造を交換することによってベクトル化に適するようになる可変長くりかえし構造をふくむ応用の例をあげる．ここで可変長のくりかえし構造とは，Fortran の DO ループなどの固定長くりかえし構造とはちがって，可変長のくりかえし処理をおこなうくりかえし構造すなわち **while** ループや再帰よびだしなどのことをいう．ベクトル化に適さない理由としては，最内側のくりかえしがベクトル化に適さない，あるいはベクトル長がみじかいためにベクトル化しても十分な加速率がえられないなどの理由がありうる．

上記のような応用として，つぎのようなものがある．

❑ データベースにおけるデータ変換

レコード長が可変長でかつみじかいデータベースにおいて，データの暗号化や圧縮・伸長などの変換をベクトル処理でおこなうばあいには，2 重ループの交換またはくりかえし構造の交換をともなう可変長 2 重ループのベクトル化をおこなうのが有効である (**図** 4.1 参照)．なぜなら，1 つのレコードをベクトルとして表現してベクトル化しようとしても，おおくのアルゴリズムにおいてデータ依存のためにベクトル化ができないまたはできてもたかい性能がえられないからである．データ依存が存在するのは，レコードのある部分の処理にそれ以前の部分の結果がつかわれるばあいであるが，暗号化や圧縮・伸長のアルゴリズムにはこのような依存があるばあいがおおい．また，ベクトル化してもたかい性能がえられない理由は，レコード長がみじかいためにベクトル長もみじかく，たかい加速率がのぞめないからである．一方，2 重ループの交換をおこなえば，ベクトル要素間の依存関係はなくなってベクトル化に関する障害がなくなるうえ，レコード数が十分におおければ 十分なベクトル長を確保することができ，したがって実行を加速することができる．

データベースにおいては，上記のような操作以外にもくりかえし構造の交換の適用によってベクトル化が可能になったりベクトル処理性能が向上する処理があり，その一部は M-680H IDP においても実現されている．

❑ 複数のリストの各要素への処理

前節でもふれたように，リストの各要素に同種の処理をほどこすばあいにおいては，リストを逐次的にたぐらなくてはならないため，1 つのリストに関する処理をベクトル処理することは困難である．しかし，このような処理を多数のリストに対しておこなうばあいには，前章でしめしたように，2 重ループの交換 (または 2 重のくりかえ

し構造の交換) によってベクトル化し，実行を加速することができる．

❑ 複数の収束計算

ニュートン法で代表される収束計算は，直前のくりかえしの結果を使用する．したがって，1 つの収束計算をベクトル処理することはできない．しかし，リスト処理のばあいと同様に，同種の収束計算を多数のデータに対してくりかえし実行するばあいには，2 重ループの交換または 2 重のくりかえし構造の交換によってベクトル化し，実行を加速することができる．ただし，性質がよい関数に関するニュートン法のように収束がはやいばあいは定数回の反復で十分であり，このようなばあいには内側ループを展開してしまえばよい．ループ交換またはくりかえし構造の交換を適用するのが適当なのはより収束がおそい計算である．

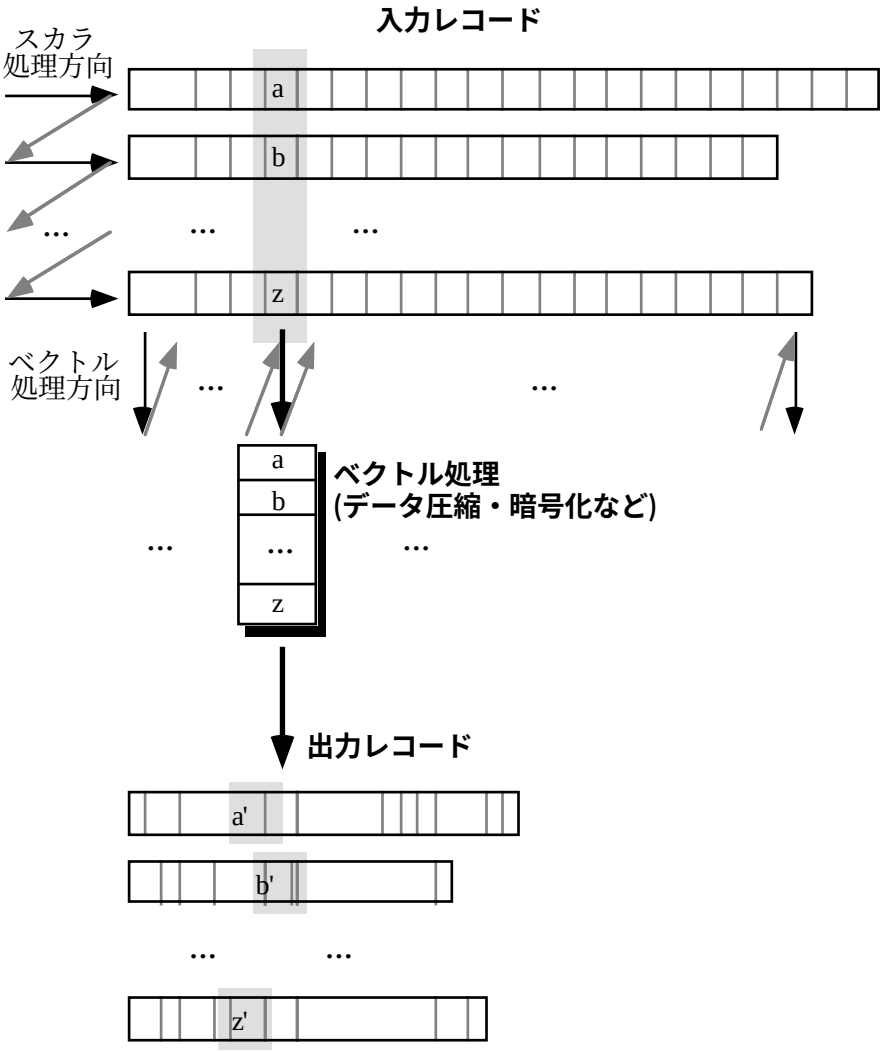


図 4.1　くりかえし構造交換によりベクトル処理好適となる例: データ変換

4.3 可変長くりかえし構造交換のための技法

この節では，可変長くりかえし構造の交換によるベクトル化において終了判定コード生成とオーバラン無効化という 2 つの処理をおこなう必要があることを説明し，残存要素検出法および最大回数反復法という，これらの処理に関してことなる方法をもちいた 2 つのくりかえし構造交換法をしめす．

そのまえに，記述の便宜のためにこの章で使用するいくつかの用語をここで定義しておく．くりかえし構造の交換において，対応している交換前の外側くりかえしと交換後の内側くりかえしとを**原外くりかえし**とよぶことにする．同様に，対応している交換前の内側くりかえしと交換後の外側くりかえしとを**原内くりかえし**とよぶことにする．また，原外くりかえしのくりかえし回数を原外くりかえし回数，原内くりかえしのくりかえし回数を原内くりかえし回数とよぶ．これらの名称はループ交換においても使用することにする．また，とくに交換前の外側ループと交換後の内側ループとを**原外ループ**，交換前の内側ループと交換後の外側ループとを**原内ループ**とよぶことにする．

4.3.1 オーバラン無効化と終了判定コード生成

4.2 節で例示したようなプログラムにおいては原内くりかえし回数が可変であるため，単純に 2 重のくりかえし構造の内外をいれかえてベクトル化しただけでは原内くりかえし回数がもとのプログラムとは変わってしまうため，不正な処理がおこなわれるようになる．もとのプログラムとちょうどおなじだけの処理をおこなうためには，つぎのような 2 つの処理が必要である (**図** 4.2 参照)．

第 1 に，ベクトル化前の原外くりかえしの i 回目における原内くりかえし回数を m_i とするとき，ベクトル化後の原内くりかえし回数 M をつぎの条件をみたすように適当にきめ，それをこえるくりかえしをおこなわないようにする必要がある．これを終了判定コード生成とよぶ．

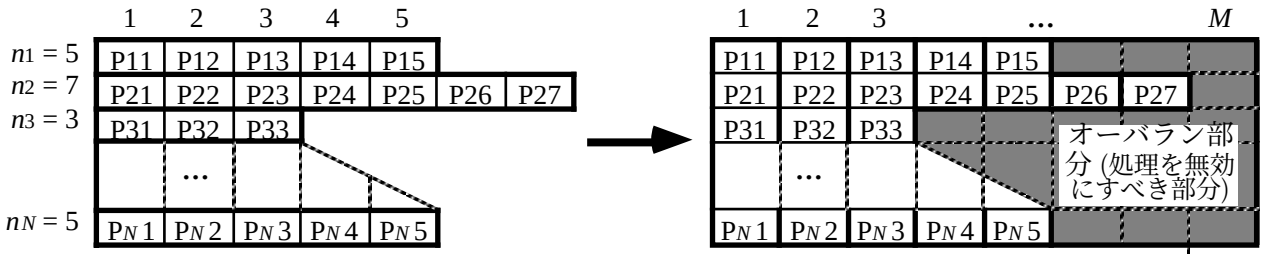
$$M \geq max_i(m_i)$$

ここで，$M = max_i(m_i)$ とするのがもっとも効率がよい．このようにくりかえしをうちきるコードは変換前のプログラムには対応するコードが存在しないため，それを明に生成しないと，変換後のプログラムにおいてはばあいによってはくりかえしが停止しない．たとえば，リスト処理のように最大長がおさえられない可変長データのばあいには，くりかえしが停止しなくなる．したがって，M 回をこえるくりかえしをおこなわないようにするコードを生成する必要がある．これに対して，4.4 節でしめすような配列を操作するプログラムのばあいには，変換前のプログラムにくりかえし回数を配列の最大長でおさえるコードがあるために問題はおこらない．したがって，このばあいにはかならず

しも終了判定コードを生成する必要はない.

第 2 に，ベクトル処理においてベクトルの第 i 要素であって $M > m_i$ をみたすものに対する処理は余分であって，この部分を実行すると不正な結果がえられるおそれがあるので，無効にする必要がある．この無効化すべき部分をオーバラン部分とよび，この無効化をオーバラン無効化とよぶことにする．オーバラン無効化のためには，ベクトル計算機に用意された 3 種類の条件制御方式 [Kamiya 83] すなわちマスク演算方式，収集拡散方式，あるいは圧縮伸長方式をつかえばよい．これらを利用した記号処理のための条件制御方式を，われわれはそれぞれマスク演算方式，インデクス方式，圧縮方式とよんでいる．リスト処理においてこれらの条件制御方式をどのように使用すればよいかについては前節でのべたとおりであるが，同様にしてリスト処理以外へも容易に適用することができる．したがって，ここではオーバラン無効化の方法をくわしくはのべないが，マスク演算方式のばあいだけについてかんたんにのべると，処理対象のベクトルと要素数のひとしいマスク・ベクトルを用意して，無効な要素に対応するマスク・ベクトルの要素は *false* とし，このマスク・ベクトルの制御のもとでベクトル処理を制御すればよいということになる.

次節以降では，上記のようなくりかえしの終了判定とオーバラン無効化に関してことなる方法をふくんだくりかえし構造交換法を 2 つしめし，配列の線形検索に関する実測データをもとにして，両者を比較する.



(a) 交換前の処理　　**(b) 交換後の処理**

図 4.2　可変長くりかえし構造の交換

4.3.2 残存要素検出法

残存要素検出法による 2 重ループの変換前および変換後のプログラムを**図** 4.3 にしめす (変換後のプログラムもスカラ処理のままのかたちでしめしている)．この方法は，2 重ループの実行を開始するまえに原外くりかえし回数がもとめられるばあいに適用することができ，原内くりかえし回数の最大値はあらかじめわからなくても適用することができる．残存要素検出法は従来から部分的にしられており，4.2 節でふれた M-680H IDP の可変長データ処理機能もインデクス方式

(リスト・ベクトルを使用する方式) にもとづく残存要素検出法の使用を前提として設計されている．

残存要素検出法においては，変換後の内側ループはベクトル処理とし，その実行においてくりかえしごと，すなわちベクトルの要素ごとに条件 $M > m_i$ がみたされているかどうかの判定をおこない，その結果をマスク・ベクトルとする．そのマスク・ベクトルによってオーバラン無効化をおこなうとともに，有効な要素 (残存要素とよぶ) が存在するかどうかを判定し，残存要素がなくなるまで外側ループを実行する．図 4.3 の変換後のプログラムにおいては内側ループに条件文がふくまれているが，このプログラムに自然なベクトル化をおこなうと，その条件式がマスク・ベクトルを生成する命令にコンパイルされる．

残存要素がなくなったかどうかの判定にはいくつかの方法がかんがえられるが，ベクトル計算機 S-810 / S-820 においてはこのためにマスク計数命令をつかうことができる．図 4.3 のプログラムはマスク計数命令の使用を前提としている．図 4.3 では変数 *count* によって上記の条件式が真となる内側くりかえし回数すなわち *true* であるマスク・ベクトルの要素の数がかぞえられる．なお，図 4.3 においては内側ループのくりかえし回数はつねに *M* 回としているが，ここで残存要素だけを処理するようにすればくりかえし回数をへらすことができる．また，このばあいはくりかえし回数が残存要素数にひとしいので，マスク計数命令を使用せずに終了判定をおこなうことができる．条件制御のためにインデクス方式または圧縮方式を採用すればこのような最適化が可能になるが，その具体的な方法については 4.5 節でのべる．

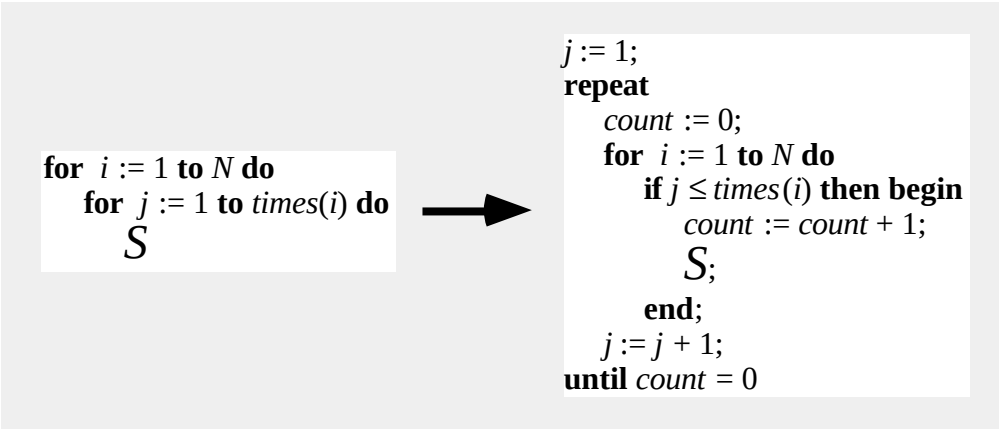


図 4.3　残存要素検出法による可変長 2 重ループのベクトル化 (マスク計数によるインプリメント)

4.3.3 最大回数反復法

上記のように，インデクス方式や圧縮方式にもとづく最大回数反復法においてはループのくりかえし回数が残存要素数にひとしいため，残存要素検出法を余分のオーバヘッドなしに実現することができる．しかし，インデクス方式においてはデータのアクセスにインデクス・ベクトルを使用するため，もともとそのオーバヘッドがおおきい．また，圧縮方式は外側ループのくりかえしごとにすべてのベクトルの圧縮をおこなうためにオーバヘッドがおおきい．一方，マスク演算方式と残存要素検出法とをくみあわせると，原内ループのくりかえしごとに，残存要素をかぞえるためにマスク・ベクトルの *true* という値をもつ要素の数をかぞえる必要が生じる[注1]．この演算は各要素の処理のあいだにデータ依存 (ループ依存) があるために比較的オーバヘッドがおおきいとかんがえられる．この高価な演算をおこなうことなしにマスク演算方式をつかったくりかえし構造交換を可能にするあらたな方法が，最大回数反復法である．

最大回数反復法による 2 重ループの変換前および変換後のプログラムを**図** 4.4 にしめす[注2]．この方法は，2 重ループの実行を開始するまえに原外くりかえし回数がもとめられるとともに, 変換前の原内くりかえし回数の最大値 (または最大値よりおおきい適当な値) *Mmax* があらかじめもとめられるばあいに適用することができる．

最大回数反復法においては，変換後のプログラムにおいて *Mmax* をループの実行前にもとめて，*Mmax* を変換後の原内くりかえし回数とする．図 4.4 においては *times*(*i*) が原外ループの *i* 回目のくりかえしにおける原内ループのくりかえし回数をあらわしている．したがって，その最大値をもとめて *Mmax* の値としている．変換後の内側ループ (原外ループ) はベクトル処理する．オーバラン無効化の方法については 4.3.1 節でのべたとおりであるが，図 4.4 においては原内ループ内の条件文によってオーバラン無効化がなされている．

[注1] S-810 / S-820 においては VMCO (Vector Mask Count Ones) 命令を実行する．

[注2] 変換後のプログラムも，ベクトル化前のかたちすなわちループ分散 (3.2 節参照) をほどこさないスカラ処理のままのかたちでしめしている．しかしながら，変換前のプログラムはベクトル化に適さず，変換後のプログラムはベクトル化に適する．

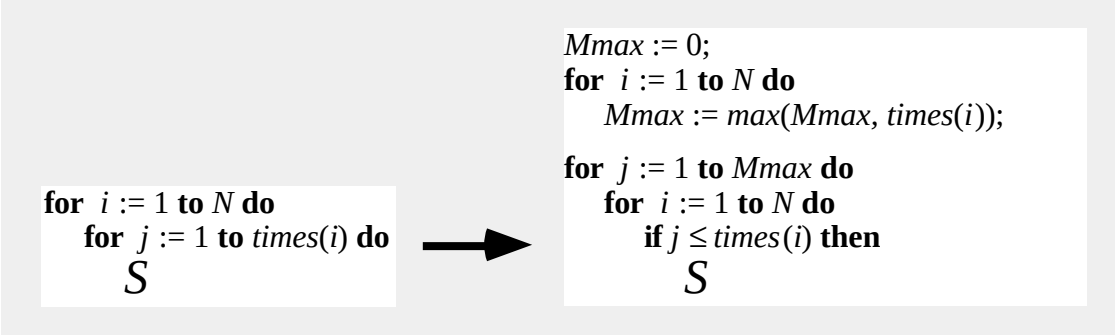


図 4.4　最大回数反復法による可変長 2 重ループのベクトル化

4.4 配列線形検索への適用例

この節では，最大回数反復法と残存要素検出法を使用したアルゴリズムの実例として，配列の線形検索のアルゴリズムをしめす．これらのアルゴリズムにもとづくプログラムの性能比較を次章でおこなう．

まず，配列の線形検索を例題としてとりあげる理由をのべる．第 1 の理由は，線形検索はアルゴリズムが単純であり，したがって最大回数反復法，残存要素検出法の本質がみきわめやすいとかんがえられることである．第 2 の理由は，これらの方法に直接の関係がない部分がしめる割合が実行のうえでもひくいため，性能上もこれらの方法の比較に適しているとかんがえられることである．第 3 の理由は，このアルゴリズムはループ交換なしでもベクトル化できるため，ループ交換せずにベクトル化したばあいとの比較もできることである．第 4 の理由は，みじかい配列の線形検索は実用的なアルゴリズムであり，この章でとりあげるベクトル処理アルゴリズムも実用性があり，その測定データは実用的な用途をかんがえるときに直接参考にすることができるとかんがえられることである．ただし，それにしても配列線形検索はひとつの例にすぎず，これだけで最大回数反復法と残存要素検出法の完全な比較がなされるわけではない．この点に関しては 4.5 節でふたたび考察する．

なお，4.5 節でしめす実測には Fortran によって記述したプログラムを使用するが，この章では Pascal 風の言語でアルゴリズムを記述する．その理由は，Fortran ではその言語上の制約のためにアルゴリズムの記述が複雑化する部分があるからである．Fortran のプログラムは**付録 1** にしめしているので，必要に応じて参照されたい．

4.4.1 スカラ処理アルゴリズム

スカラ処理のための配列線形検索のアルゴリズムを**図 4.5** にしめす．このアルゴリズムの機能を**図 4.6** に例示する．入力は検索表を要素とする配列 *table*（2 重の配列または 2 次元の配列) と検索すべきデータからなる配列 *data_to_search* であり，出力は検索表への添字からなる配列 *found* である．*data_to_search* と *found* の配列としてのおおきさはともに N であり，N も引数として入力するものとする．このアルゴリズムは，ベクトル処理アルゴリズムとあわせるため，通常の検索アルゴリズムとはことなるインタフェースを採用している．すなわち，検索すべきデータは複数個あり，それらを配列 にいれてわたしている．検索データ *data_to_search*[*i*] が検索表 *found*[*i*] に登録されているときは，その検索表エントリの添字 (図 4.6 ではイタリック体の数字でしめしている) を *found*[*i*] に代入する．登録されていないときは，*found*[*i*] の値は 0 とする．図 4.6 にしめしたように，検索表の複数のエントリに検索データとひとしい値がふくまれるときには，そのなかの最初のエントリの添字だけが *found*[*i*] に代入される．

なおこのアルゴリズムにおいては，むだな計算をはぶくため，検索データとひとしい値が検索表にみつかったときは内側ループを脱出するようにしているが，そのために自然なベクトル化 (ループ分散技法だけによるベクトル化) はできなくなっている．

```
procedure s_search (table, data_to_search,  found, N);
begin
   found[1 .. N] := 0;                  — 配列 found の全要素を 0 に初期化する．
   for i := 1 to N do begin             — 検索データに関するくりかえし．
L:    for j := 1 to size(table[i]) do   — 検索表のエントリに関するくりかえし．
         if data_to_search[i] = table[i, j] then begin
            found[i] := j;              — みつかった検索表エントリの添字を found[i] に代入する．
            exit L;                     — むだな計算をはぶくため，内側ループを脱出する．
         end;
   end;
end s_search;
```

図 4.5　配列線形検索スカラ処理アルゴリズム

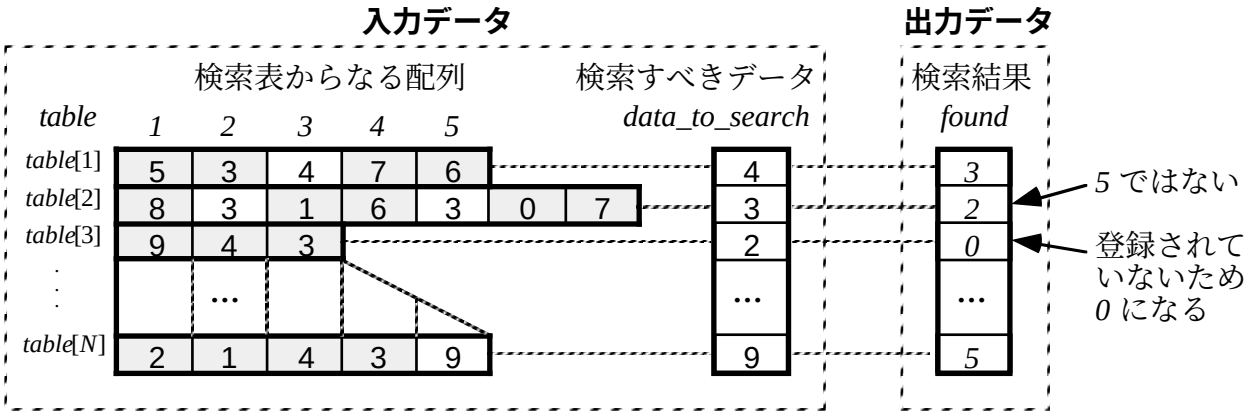


図 4.6　配列線形検索アルゴリズムの機能

4.4.2 ループ非交換版のベクトル処理アルゴリズム

図 4.6 のアルゴリズムに対しては自然なベクトル化ができないので，ループ脱出をなくしてベクトル処理可能にした配列線形検索アルゴリズムを**図** 4.7 にしめす．このアルゴリズムを Fortran でコーディングすれば，ループ交換などの特別なプログラム変換なしに自然にベクトル化することができる．すでにのべたように，スカラ処理版のアルゴリズムにおいては，検索表の複数のエントリに検索値がふくまれるときには，この **exit** 文があることによって最初のエントリの添字だけが *found*[*i*] に代入される．しかし，この **exit** 文をなくすとこれらの複数のエントリの添字がすべて *found*[*i*] に代入されるため，*found*[*i*] には最後に代入された添字の値がのこる．したがって，ベクトル処理版のアルゴリズムにおいてスカラ処理版とおなじ結果をえるためには，ループの実行の向きを逆転

させる必要がある[注3]．こうしてえられたアルゴリズムが図 4.7 である．

```
procedure vm_search (table, data_to_search,  found, N);
begin
   found[1 .. N] := 0;
   for i := 1 to N do
      for j := size(table[i]) to 1 by –1 do
                                  — スカラ処理版とひとしい結果をえるため，逆順に実行する．
         if data_to_search[i] = table[i, j] then
            found[i] := j;              — ベクトル化可能にするため，ループ外脱出を削除する．
end vm_search;
```

図 4.7　ループ非交換版の配列線形検索ベクトル処理アルゴリズム

4.4.3 マスク演算方式最大回数反復法によるベクトル処理アルゴリズム

最大回数反復法を使用してループ交換をおこなった配列線形検索アルゴリズムを**図** 4.8 にしめす．このアルゴリズムも特別なプログラム変換なしに自然にベクトル化することができる．このアルゴリズムにおいては，変数 *mx* にループ非交換版の原内ループのくりかえし回数の最大値をもとめている．原外ループの *i* 番目のくりかえしにおける原内くりかえし回数は，スカラ処理アルゴリズムのようにループからの脱出をしないかぎりは検索表 *table*[*i*] のおおきさにひとしいから，その最大値をもとめて *mx* に代入している[注4]．このアルゴリズムでは，条件制御の方法としてはマスク演算方式によっている．すなわち，最内側ループにふくまれる条件文が，自然なベクトル化によってマスクつき演算に変換される．

インデクス方式による最大回数反復法はためさなかった．その理由はつぎのとおりである．インデクス方式においてはいずれにしても有効要素の数をかぞえる必要がある．なぜなら，それがインデクス・ベクトルのベクトル長であり，したがってベクトル演算に必要だからである．有効要素数がわかれば残存要素検出法をつかうことができ，そのほうがむだな計算をしなくてすむ．したがって，インデクス方式を採用したばあいには，最大回数反復法よりも残存要素検出法のほうが有利である．これに対して，マスク演算方式においては有効要素の数をかぞえる必要はなく，2 重ループの外であらかじめくりかえし回数の最大値をもとめておくほうがむだがすくないとかんがえられる．したがっ

[注3] したがってこのプログラムにおいてはオーバラン部分をさきに実行している．

[注4] スカラ処理においてループから脱出する内側くりかえし以降のくりかえしを (どのアルゴリズムであろうと) 実行するのはむだであるから，できればスカラ処理におけるくりかえし回数の最大値を *mx* に代入するのがのぞましい．しかし，スカラ処理におけるくりかえし回数は実行してみてはじめて確定するものだから，それをあらかじめもとめるのは不可能である．したがって，最大回数反復法においては，むだを承知でループ非交換版におけるくりかえし回数の最大値をくりかえし回数 *mx* とするほかはない．

て，マスク演算方式を採用したばあいには，最大回数反復法のほうが有利になりうる．

```
procedure vmx_search (table, data_to_search,  found, N);
begin
   found[1 .. N] := 0;
   mx := max(size(table[1 .. N]));
         — 変換前の内側ループのくりかえし回数の最大値 mx をもとめる．
   for j := mx to 1 by –1 do                    — ループ非交換版における内側ループ．
      for i := 1 to N do                        — ループ非交換版における外側ループ．
         if data_to_search[i] = table[i, j] and j ≤ size(table[i]) then
                                                — j ≤ size(table[i]) は余計な処理をおさえるための条件．
            found[i] := j;
end vmx_search;
```

図 4.8　マスク演算方式最大回数反復法による配列線形検索ベクトル処理アルゴリズム

4.4.4 インデクス方式残存要素検出法によるベクトル処理アルゴリズム

残存要素検出法を使用してループ交換をおこなった配列線形検索アルゴリズムを**図** 4.9 にしめす．このアルゴリズムも特別なプログラム変換なしにベクトル化することができる．条件制御の方法としてはインデクス方式によっている．このアルゴリズムにおいては，配列 *index* がインデクス・ベクトルであり，配列 *table*，*data_to_search* および *found* のすべての処理すべき要素の添字 (インデクス) が格納されている．また，配列 *index* のおおきさが *ndata* に格納されている．

index には，最初はすべての配列要素の添字 1, 2, … , *N* が格納される．すなわち，初期値設定部においてこのように *index* および *ndata* の値が設定される．検索処理がすすむにつれて，すでに検索が終了した配列要素の添字は *index* からとりのぞかれていく．すなわち，さらに検索すべき要素の添字だけが *index* につめかえられる．内側ループのベクトル長は *ndata* であるから，外側ループのくりかえしがすすむにつれてしだいに内側ループのベクトル長はみじかくなっていくことになる．そして，*index* に格納された添字がなくなると，すなわち *ndata* の値が 0 になると，処理は終了する．

なお，マスク演算方式によって残存要素検出法を実現することも可能だが，そのアルゴリズムの記述はここでは省略する．

4.5 配列線形検索における性能実測結果とその検討

前節でしめした各アルゴリズムおよびマスク演算方式による残存要素検出法を Fortran でコーディングして，その性能をパイプライン型ベクトル計算機 S-810 で実測した．その主要な結果を**図** 4.10 〜 4.12 にまとめる．図 4.10 〜 4.12 で表現しきれない部分については，補足的にしめす．最大回数反復法および残存要素検出法の性能をうまく説明するモデルの記述に成功していないため，ここではそれらの測定データのなかから代表的とかんがえられるものをえらんでしめしている．

4.5.1 測定内容および条件

図 4.10 〜 4.12 は，それぞれ検索データ数 N が 16, 128, 1024 のときの配列線形検索の S-810 における検索時間をしめしている (数値を**付録** 2 にしめした)．これらの図に関する測定条件および表示条件はつぎのとおりである．検索データおよび検索表の数すなわち原外ループのくりかえし回数 N を横軸にとっている．図 4.10 に $N = 16$ のばあい，図 4.11 に $N = 128$ のばあい，図 4.12 に $N = 1024$ のばあいをしめしている[注5]．原内ループのくりかえし回数すなわち検索表のおおきさ m_i $(i = 1, 2, \ldots, N)$ は $[0, M)$ すなわち $0 \le m_i < M$ という範囲の整数値をとる一様乱数によってきめた．図 4.10 〜 4.12 の横軸にとったのが M の値である．登録データおよび検索データも $[0, M)$ という範囲の整数値をとる一様乱数を使用している (データの値域をかえると性能が変化するが，この変化については後述する)．検索表のおおきさが 上記の乱数であたえられるとき，最大回数反復法における $Mmax$ の推定値は $M(N+1)/(N+2)$ となる (**付録** 3 に証明をしめす) から，N が十分おおきいばあいには，$M \approx Mmax$ とみなしてさしつかえない．

[注5] この測定をおこなうまえに，さまざまな条件のもとでの測定をためしにおこなった．その結果，データを効果的にまとめるには $N = 16, 128, 1024$ のばあいをそれぞれ 1 枚のグラフにするのがよいという結論に達したため，これらのばあいのデータをとりなおした．これらの図で表現しきれない原内くりかえし回数への実行時間の依存性などについては，図 4.14 以下で補足的にしめす．

```
procedure vic_search (table, data_to_search,  found, N);
   var index, count, ndata, ix, j;
begin
   found[1 .. N] := 0;

/* 配列 index および変数 ndata の初期値設定: */
   count := 0;
L1:
   for i := 1 to N do
      if 1 ≤ size(table[i]) then begin
         count := count + 1;          — 検索すべきデータの数をかぞえる.
         index[count] := i;
               — 処理すべき要素の添字を配列 index に (S-810 / S-820 のばあい，ベクトル圧縮命令
               — VSTC (Vector STore Compressed) によって) 格納する.
      end;
   ndata := count;                    — くりかえし回数の初期値設定.

/* 検索: */
   j := 1;
   repeat                             — スカラ処理版における内側ループ.
      count := 0;
      for i := 1 to ndata do begin    — スカラ処理版における外側ループ.
         ix := index[i];
         if j ≤ size(table[ix]) then begin
            if data_to_search[ix] = table[ix, j] then
               found[ix] := j
            else begin
               count := count + 1;    — さらに検索をつづけるべきデータの数をかぞえる.
               index[count] := ix;
                  — 検索をつづけるべき要素の添字だけを，ベクトル圧縮命令によって
                  — 配列 index につめかえる.
            end;
         end;
      end;
      j := j + 1;
      ndata := count;                 — くりかえし回数の更新.
   until count = 0;                   — 残存要素数が 0 になるまでくりかえす.
end vic_search;
```

図 4.9　インデクス方式残存要素検出法による配列線形検索ベクトル処理アルゴリズム

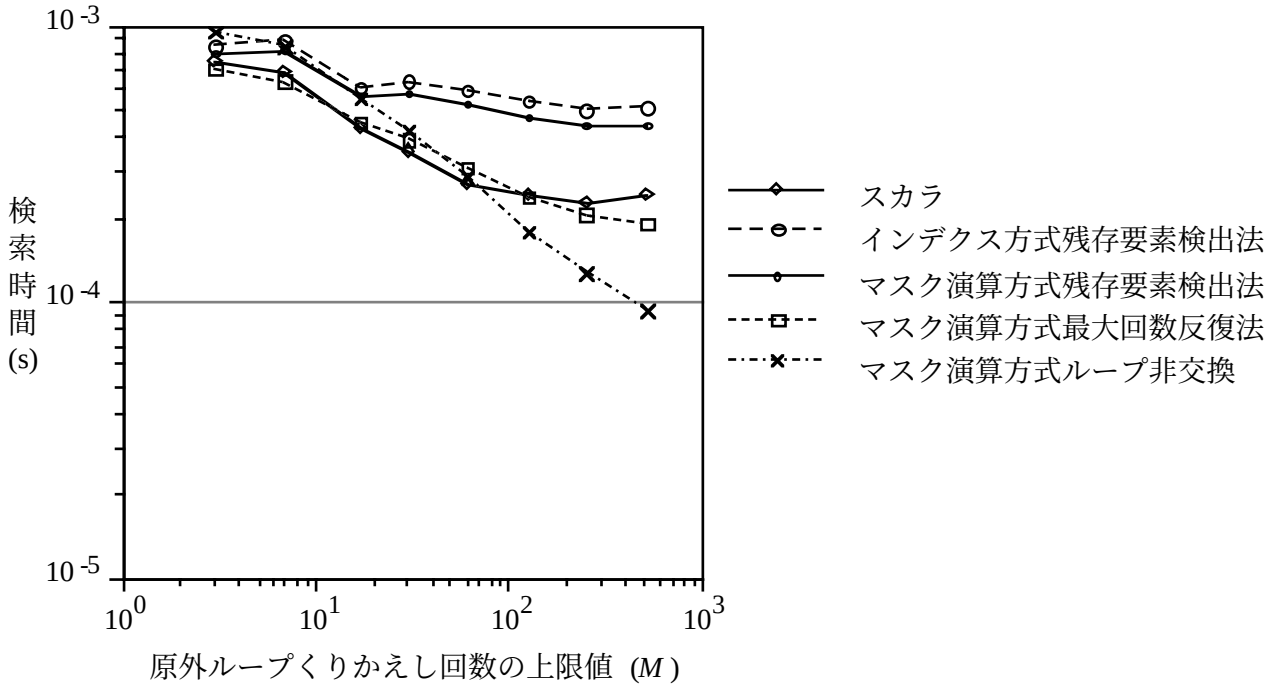


図 4.10　検索データ数 $N = 16$ のときの検索時間

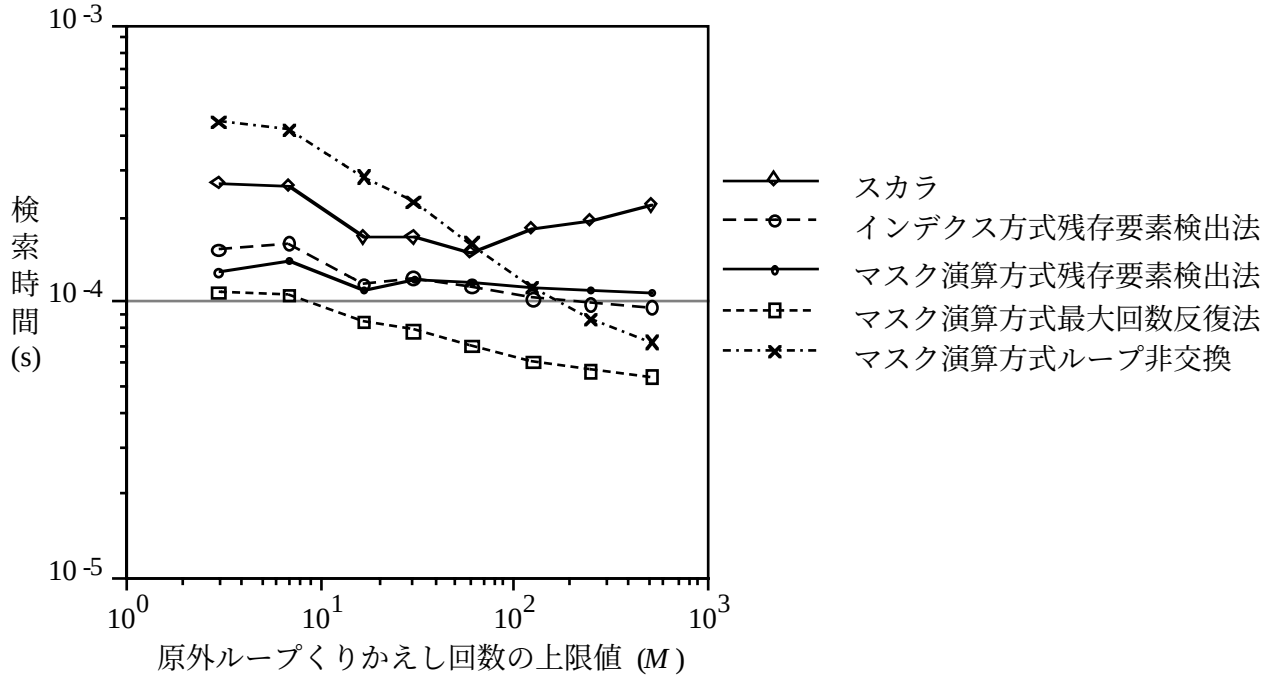


図 4.11　検索データ数 $N = 128$ のときの検索時間

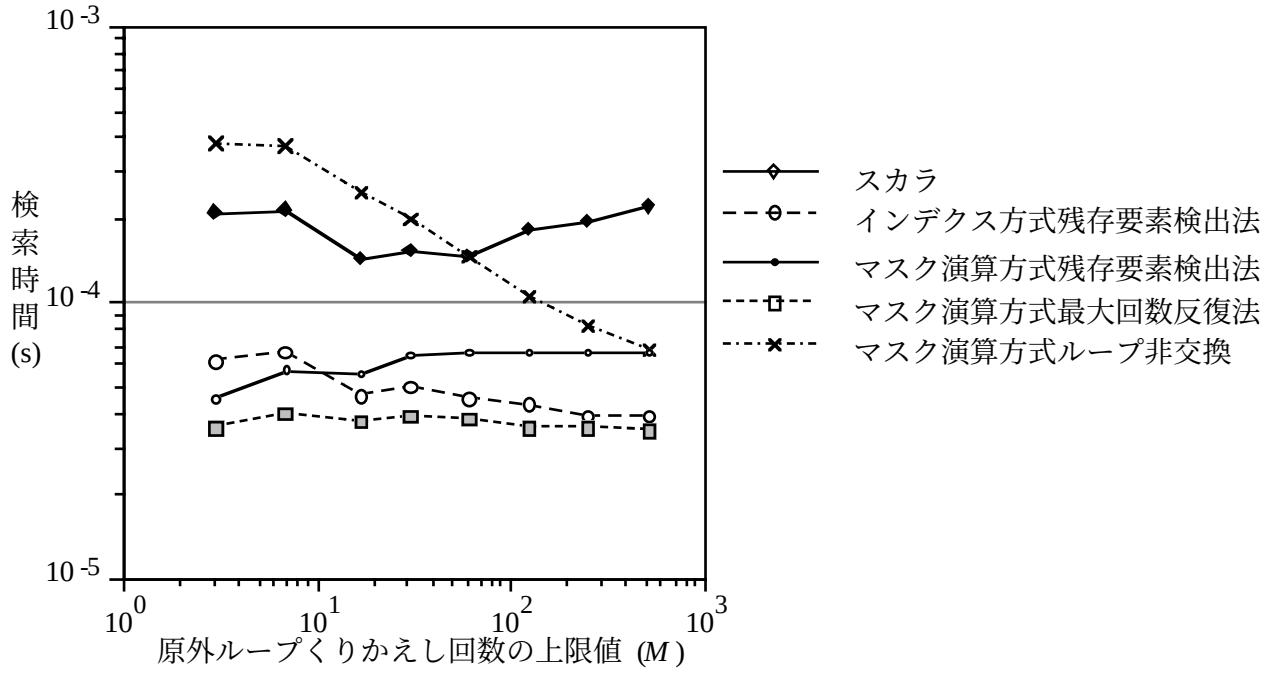


図 4.12　検索データ数 $N = 1024$ のときの検索時間

データの生成と検索表のおおきさの決定に乱数を使用しているために，スカラ処理における内側ループのくりかえし回数は一定にならず，測定値にはばらつきがでる．そのばらつきをおさえるために，図 4.10 〜 4.12 には乱数の値をかえて 100 回の測定をおこない，その平均値をしめしている．

各図の縦軸には，総登録時間を NM でわった値をしめしている．NM でわっているのは，これらのデータを比較しやすくする便宜のためであり，したがって縦軸の絶対値はとくに意味をもたない．ただし，検索表のエントリの平均値は $NM/2$ であるから，縦軸の値を 2 倍したものがエントリあたりの平均検索時間だとかんがえることができる．

各測定の意味をあきらかにするため，**図** 4.13 に乱数の上限値 M と原内くりかえし回数や各方式における総くりかえし回数などとの関係を図示する．図 4.13 においては，横方向に原内ループのくりかえし，縦方向に原外ループのくりかえしをとり，通常どおりの処理がおこなわれる部分は白，処理がおこなわれない部分は 2 とおりの濃い灰色，オーバラン部分すなわち無効化された処理がおこなわれる部分はうすい灰色で色分けしてある．また，$Mmax$ の意味についても図 4.13 (3) に図示している．

データの生成と検索表のおおきさの決定に乱数を使用しているために，原内ループのくりかえし回数は一定にならず，測定値にはばらつきがでる．そのばらつきをおさえるために，図 4.10 〜 4.12 には乱数の値をかえて 100 回の測定をおこない，その平均値をしめしている．

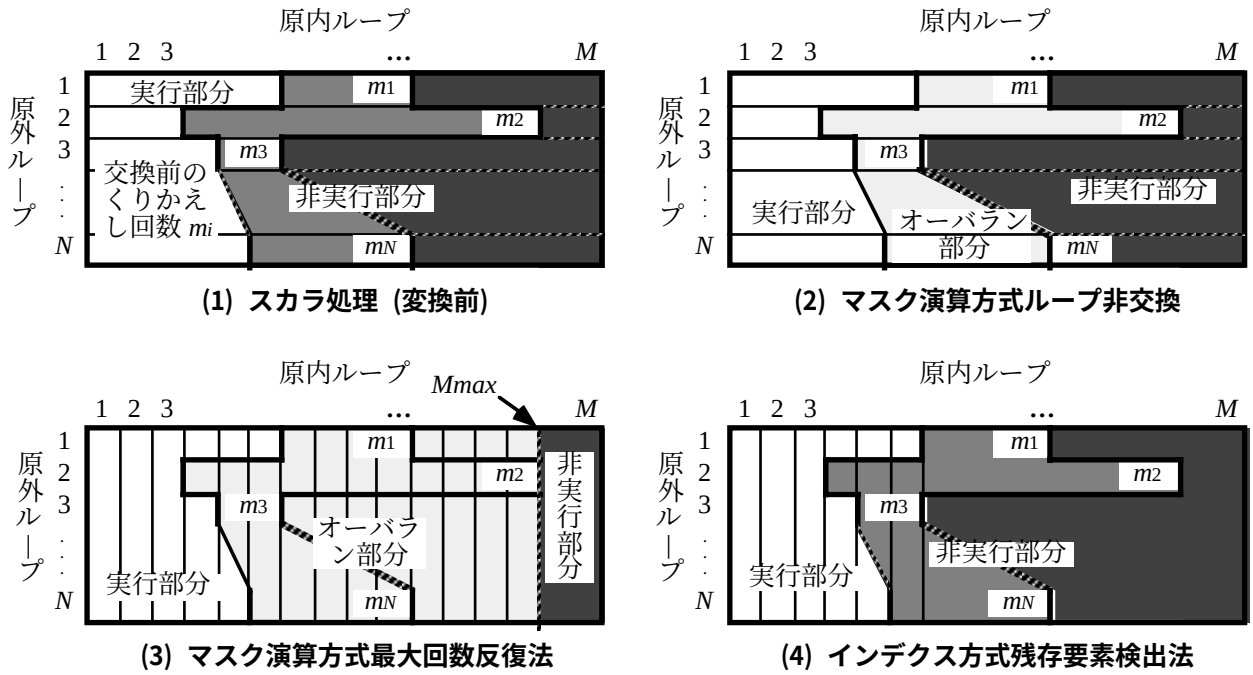


図 4.13　各方式における検索処理の実行部分と非実行部分

各図の縦軸には，総登録時間を NM でわった値をしめしている．NM でわっているのは，これらのデータを比較しやすくする便宜のためであり，したがって縦軸の絶対値はとくに意味をもたない．ただし，検索表のエントリの平均値は $NM/2$ であるから，縦軸の値を 2 倍したものがエントリあたりの平均検索時間だとかんがえることができる．

上記の測定結果を補足するために，さらに 2 種類のことなる条件のもとでの測定をおこない，図 4.14 〜 4.15 にしめした．**図 4.14** には，前記の測定条件とはちがって，登録データおよび検索データを発生させる乱数の上限を変化させたときの各方式の検索時間の変化をしめす[注6]．ここでは，検索表のおおきさをきめる乱数の上限 M は固定している．他の測定条件は図 4.10 〜 4.12 におけるのとひとしい．データ発生に関する乱数の上限を増加させると原内くりかえし回数の平均値が増加し，それにつれて総くりかえし回数も増加する．

図 4.15 は乱数の分布の変化により検索表のおおきさの平均値を変化させ，それによるループの平均長 (平均くりかえし回数) と最大長との比率の変化によって各方式の加速率がどのように変化するかをしめしている．この比率が，マスク演算方式によってつねに M 回のくりかえしをおこなうばあいのマスク成立比率すなわちマスク・ベクトルにおける真である要素の比率となる．ただし，マスク演算方式最大回数反復法においては，通常は M 回よりすくないくりかえししか実行しないので，実際のマスク成立比率はこれよりややたかい．測定条件は図 4.10 〜 4.12 のばあいとほぼおなじだが，乱数の分布だけがことなる．ループの平均長と最大長との比率は，つぎのようにして変化させた．一様乱

[注6] ただし，マスク演算方式による残存要素検出法の検索時間は測定していない．

数 X のべき乗をとることによって分布を変化させ,結果としてえられる乱数の値域は [0, *M*) に固定する．これにより，乱数 X^i を使用したばあいにはループの平均長と最大長との比率は $1/(i+1)$ となる．図 4.10 〜 4.12 と同様の $i = 1$ のばあいには成立比率は 0.5 である．

図 4.16 には，図 4.15 〜 4.16 の測定条件を図 4.10 〜 4.12 の測定条件とともに図示する．以下，測定結果およびその検討内容をのべる．

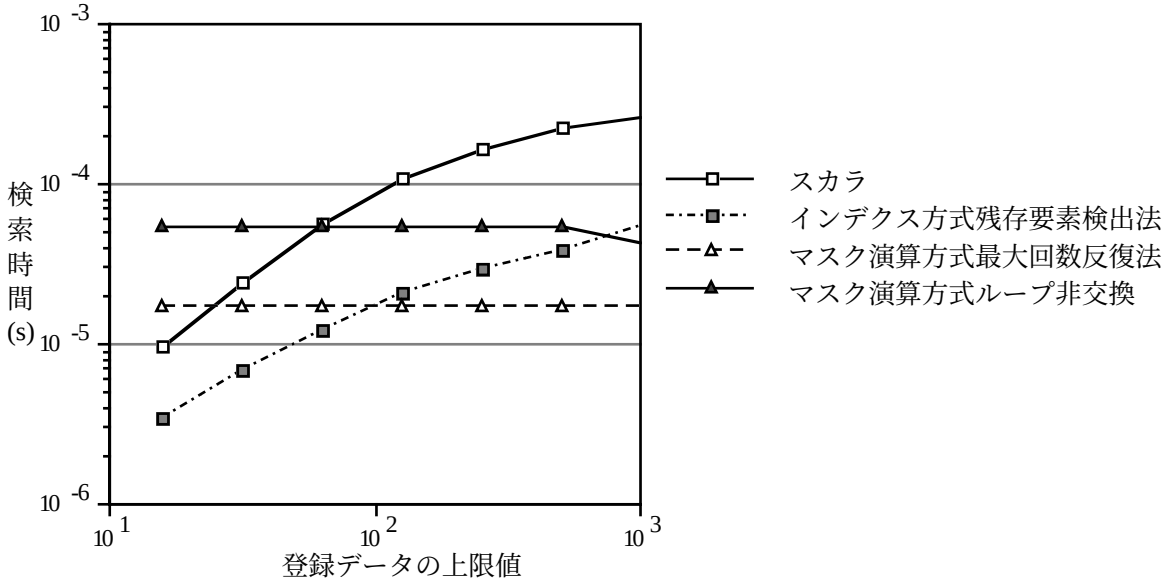


図 4.14　**登録データ (一様 乱数) 上限値の変化による検索時間の変化例**
(*N* = 1024, *M* = 511)

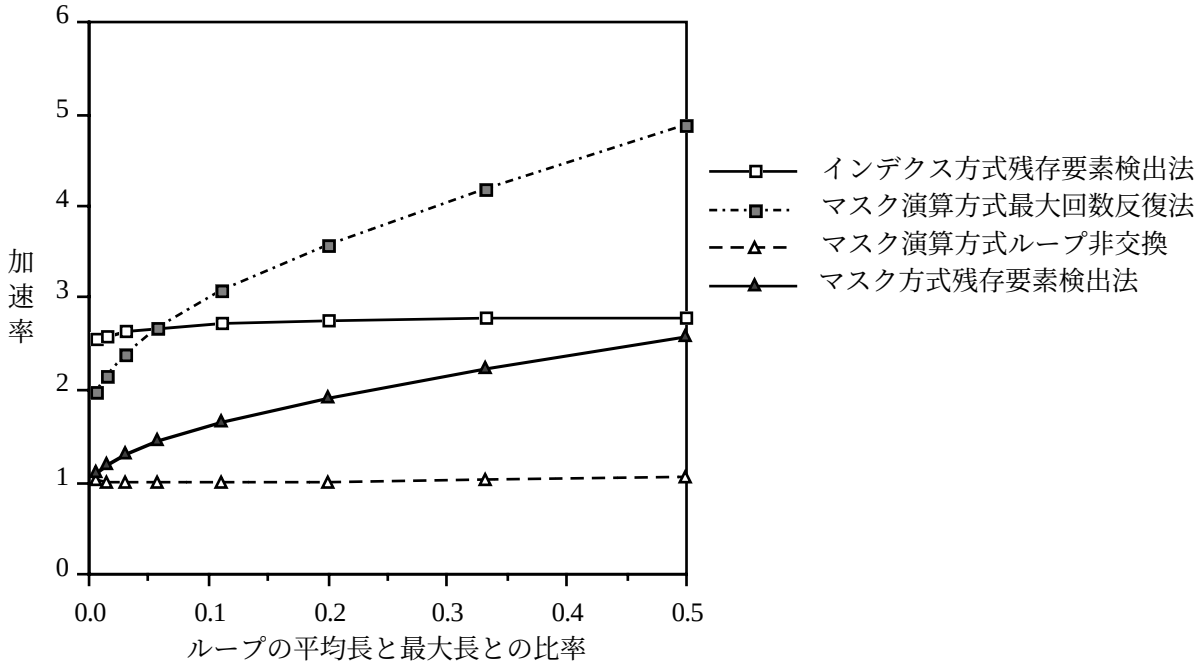


図 4.15　**ループの平均長と最大長との比率による加速率の変化例**
(*N* = 1024, *M* = 61)

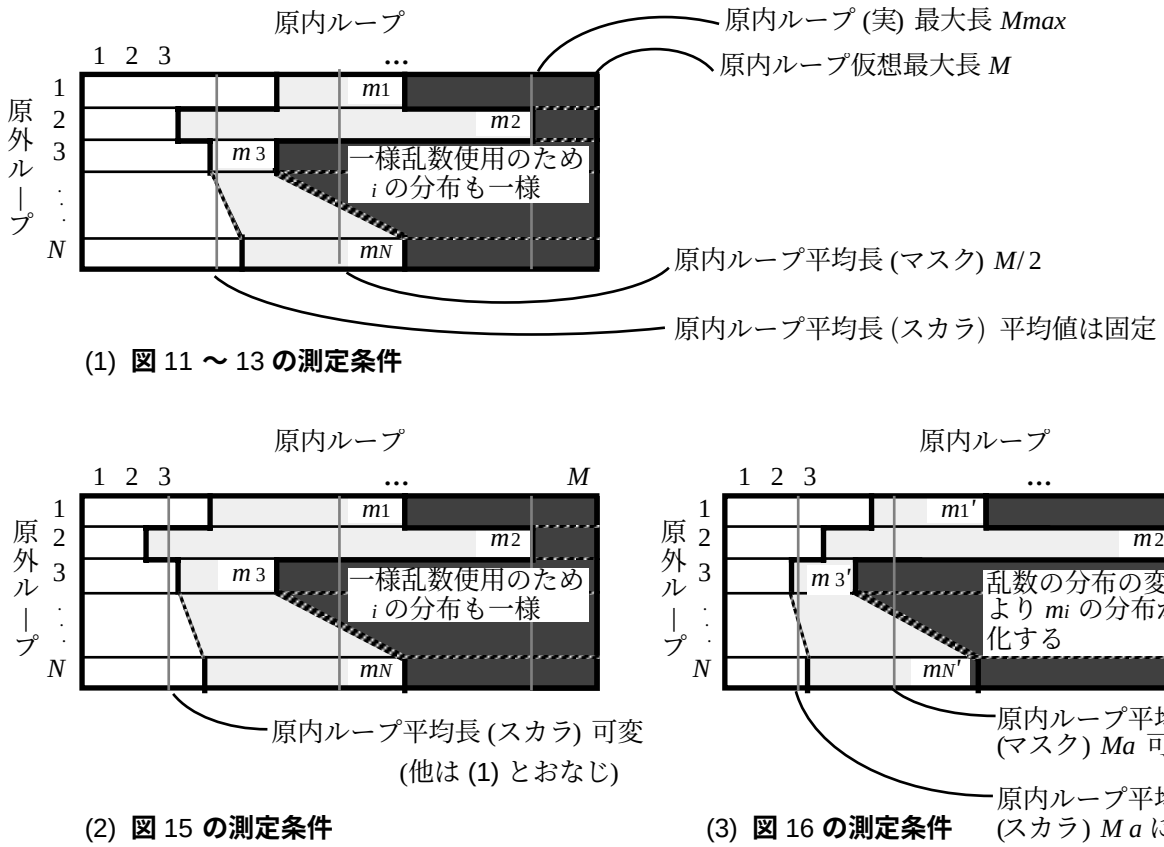


図 4.16　図 4.14 〜 4.15 の測定条件

4.5.2 ループ交換にもとづくベクトル化の実用性

線形検索が実用的なのは原内ループがみじかいばあい，したがってループ非交換にもとづく方法にくらべて 2 つのループ交換にもとづく方法が有利なばあいである．図 4.11 〜 4.12 は，このようなばあいにループ交換にもとづくベクトル化法によって高い加速率がえられ，したがって実用になりうることをしめしている．一方，線形検索じたいが実用的でない内側ループがながいばあいのデータは，線形検索のデータとしてはやくにたたないが，他の応用における最大回数反復法と残存要素検出法との関係およびこれらの方法と他の方法との関係を示唆しているという点において，示唆をあたえているとかんがえられる．すなわちこの結果は，ループ交換なしにはベクトル化できない応用においてはもちろん，ループ交換をしなくてもベクトル化可能でありかつ原内くりかえし回数がながい応用においても，ループ交換によって性能をたかめることができるばあいがあることを示唆している．

4.5.3 原外くりかえし回数への実行時間の依存性

原外くりかえし回数がちいさいばあいには，それをベクトル長とするループ交換をと

もなう 2 つの方法は不利であり，ループ非交換版のアルゴリズムが有利である．図 4.10 〜 4.12 の測定データに関していえば，$N = 16$ のときはスカラ処理あるいはループ非交換版のプログラムが比較的よく，ループ交換をともなう方法はむしろ不利である．ただし，$N = 16$ かつ $M \leq 16$ のばあいはいずれの方法でも性能に大差はない．

原外くりかえし回数が増加するにつれて，ループ交換したプログラムのほうが有利になっていくことがわかる．$N = 128$ のときは $M \geq 256$ においてまだループ非交換版のプログラムのほうがインデクス方式残存要素検出法とくらべて性能がたかいが，$N = 1024$ においては M の値によらずループ交換したプログラムのほうがともに他のプログラムより性能がよい．

4.5.4 最大回数反復法と残存要素検出法との比較

図 4.10 〜 4.12 においては最大回数反復法のほうが残存要素検出法にくらべて有利だが，インデクス方式残存要素検出法のほうが有利になるばあいもある．図 4.14 および図 4.15 では最大回数反復法とインデクス方式残存要素検出法との性能が交差する測定例をしめしている．

まず，図 4.14 に関する観察結果をしるす．図 4.14 では登録データ生成に使用する乱数の分布は一様分布のままでその上限値を変化させている．一方，検索表のおおきさをきめる一様乱数の上限は 511 に固定している．スカラ処理プログラムの実行時間は総くりかえし回数に比例して増加する．すなわち，実行時間はデータ発生に関する乱数の上限に比例する．インデクス方式残存要素検出法によるプログラムにおいてはむだなくりかえしがないため，総くりかえし回数がスカラ処理におけるそれと一致している．したがってその実行時間も，スカラ処理プログラムのばあいよりはゆるやかだが，ほぼならんで増加する[注7]．これらの変化曲線が上に凸となっているのは，原内くりかえし回数が検索表のおおきさでおさえられるからである[注8]．一方，ループ非交換版およびマスク演算方式最大回数反復法によるベクトル処理プログラムの原内くりかえし回数は検索表のおおきさに固定されていて変化しないため，総くりかえし回数も変化しない．したがって，それらの実行時間は乱数の上限を変化させてもほぼ一定である．なお，図 4.14 にしめした $N = 1024$, $M = 511$ のばあいは上記の観察結果がもっとも顕著にあらわれるが，N, M としてよりちいさな値をとっても，結果は基本的にはかわらないことがたしかめられている．

[注7] これらが比例しないおもな理由は，インデクス方式残存要素検出法にベクトル処理オーバヘッドが存在するからである．

[注8] ここではデータはしめさないが，M すなわち検索表のおおきさの上限を増大させるとスカラ処理プログラムおよびインデクス方式残存要素検出法のプログラムの実行時間の変化はより直線的になることがたしかめられている．

つぎに，最大回数反復法と残存要素検出法とを比較する．図 4.14 においては，データ生成の乱数上限値が 64 以下のばあいにはインデクス方式残存要素検出法のほうが高速になっている．すなわち，スカラ処理で検索表のごく一部しか検索されないばあい (同一のデータが多数登録されているばあいなど) は，ループ非交換版およびマスク演算方式最大回数反復法にくらべてインデクス方式残存要素検出法のほうが有利である．しかし，検索表のおおきさにくらべてデータの値域がせまいという条件はまれであり，通常使用される条件では最大回数反復法のほうが有利である．また図 4.15 でしめした $N = 1024$, $M = 61$ のばあいは比較的インデクス方式残存要素検出法に有利なばあいだが，それでもそれが最大回数反復法にくらべて有利になるのはマスク成立比率が 6 % 以下のばあいにかぎられている．図 4.14 〜 4.15 をとおしていえることは，通常の条件のもとではインデクス方式残存要素検出法より最大回数反復法のほうが高速だということである．なお，インデクス方式残存要素検出法とマスク演算方式残存要素検出法との比較においては，いずれがよいかは一概にいえない．

4.5.5 結果のまとめ

以上の結果をまとめるとつぎのようになる．なお，ここで M は原内くりかえし回数，N は原外くりかえし回数をあらわす．

(1) ループ交換にもとづくベクトル化の実用性

(1a) 一般的に

M がちいさいばあいはもちろん，M がおおきいばあいでもループ交換をおこなわないベクトル化にくらべて有利なばあいがある．

例：配列線形検索においては，$N = 128, 1024$ のとき，つねにマスク演算方式最大回数反復法のプログラムのほうがループ非交換のプログラムより高速である．

(1b) ベクトル線形検索は

線形検索が実用的である M がちいさいばあいにおいては，ループ交換にもとづくベクトル線形検索は加速率が 3 〜 6 倍程度であるから実用性はあるが，けたちがいの性能向上はえられない．

例：加速率は，$M = 3$ かつ $N = 128$ のとき 2.6, $M = 3$ かつ $N = 1024$ のとき 6.0 である．

(2) 原外くりかえし回数と有利な方式との関係

予想されるとおり，原外くりかえし回数が増加するにつれて，ループ交換したプログラムのほうが高速になる．

例：配列線形検索においては，$N = 16$ のときはスカラ処理あるいはループ非交換のプログラムがループ交換したプログラムより高速であるが，$N = 1024$ のときはルー

プ交換したプログラムのほうがつねに高速である.

(3) 最大回数反復法と残存要素検出法との比較

スカラ処理で非常に少数のくりかえし回数で原内ループから脱出するばあいは残存要素検出法がよいが，たいていの条件のもとで最大回数反復法のほうが高速である.

例: 配列線形検索のばあい，インデクス方式残存要素検出法がマスク演算方式最大回数反復法より高速になるのは，$N = 1024$, $M = 61$ のばあいでループの平均長と最大長との比 (マスク成立比率) が 0.06 以下のばあいにかぎられる.

4.6 関連研究

ループ実行前にはくりかえし回数がわからない **while** 文で記述されたループの並列化 (ベクトル化) をはかった研究として Wu ら [Wu 90] がある．しかし，この研究は 1 重の **while** ループにおいてそのくりかえし回数を決定する部分以外の部分をベクトル化するための技法に関するものであり，くりかえし回数が可変の多重のくりかえし構造に関するくりかえし構造の交換にもとづくベクトル化に関する研究はこれまでのところみられない．Wu らの技法は，ベクトル化すべき **while** ループが交換可能なループ内にないとき，外側ループのベクトル長がみじかいときなどに有効だが，当該ループのデータフローにループ依存性があってベクトル化できないときや，ループ内の演算のほとんどがくりかえし回数決定に関与するばあいには適用することができない．このようなばあいにはこの研究におけるくりかえし構造交換法が有効である．

4.7 まとめ

リスト処理やデータ変換などのプログラムにあらわれる，内側くりかえしが可変長の 2 重のくりかえし構造すなわちループ，再帰よびだしなどにループ交換技法の拡張であるくりかえし構造の交換を適用するために，われわれは従来から部分的にしられていた残存要素検出法とともにあらたな方法である最大回数反復法をしめし，配列線形検索を例題としてそれらの方法にもとづくプログラムの性能をベクトル計算機 S-810 において実測し，えられたデータを比較した．その結果，ループ交換によって十分なくりかえし回数がえられるときには，交換前の内側ループのループ長 (原内くりかえし回数) がおおきいばあいもふくめて，くりかえし構造の交換にもとづく方法にもとづく実行が他より高速であることなどがわかった．また，マスク演算方式による最大回数反復法とマスク演算方式およびインデクス方式による残存要素検出法との比較においては，通常使用される条件のもとでは前者のほうが有利だとかんがえられるが，検索表のおおきさがいちじるしくばらついているばあいや検索表のおおきさにくらべて交換前の内側ループのループ長がはるかにみじかいばあいなど，インデクス方式による残存要素検出法のほうが有利なばあいがあることもわかった．

これらの結果から，これら 2 つの方法にもとづくくりかえし構造の交換は，記号処理プログラムのベクトル化において非常に有効なプログラム変換法であるといえるであろう．くりかえし構造の交換法は，第 5 章でのべる上書きラベル・フィルタ法との併用でさらに応用がひろがるとかんがえられる．

この研究によって，可変長の 2 重くりかえし構造の交換，とくに可変長 2 重ループの交換の方法がより一般化され具体化された．しかし，ここでしめした範囲では，自動ベクトル化が可能になるほどこまかく手順が規定されたわけではない．論理型言語で記述された一部のプログラムに関しては，第 6 章においてベクトル化法を形式化する．しかし，今後の最大の課題は，他の種類のプログラムに関しても自動ベクトル化が可能になるところまで変換手順をより精密にすることだとかんがえられる．また，これまでの研究では配列線形検索などの非常にかぎられた応用プログラムへの適用をこころみただけであるが，今後はほかのプログラムへも適用をはかり，どのようなばあいにいずれの方法を適用することがより適切かを決定することがひとつの課題である．

4.8 付録 1：Fortran による配列線形検索のベクトル処理プログラム

この節では，測定に使用した Fortran プログラムをしめす．基本的には第 4 章でしめしたアルゴリズムをそのままコーディングしているが，Fortran 言語じたいの制約や S-810 / S-820 のための Fortran コンパイラがもつ制約のためにプログラムが複雑になっている部分がある．外部インタフェースに関しても，整合配列のおおきさや検索表の実際の上限を検索表じたいとはべつの引数でわたす必要があるなどの理由によって，やや複雑化している．外部インタフェースはつぎのとおりである．

```
* Inputs:
*     TABLE: Vector of the table to be searched.
*     TLENG: Vector of table length. TLENG(I) is the length
*            of TABLE(*, I).
*     VSIZE: Maximum vector length of TABLE.
*     TSIZE: Maximum table length.
*     VDATA: Data to be searched for.
*     NDATA: The number of data (= the allocated vector
*            length of TABLE).
* Outputs:
*     FOUND: Indices of the found element.  If not found,
*            FOUND contains 0.
```

4.8.1 スカラ処理版

スカラ処理のためのプログラムをしめす．このプログラムには 2 個のベクトル化オプション (*VOPTION) がふくまれている．これらは，部分的なベクトル化をおさえるために必要である．

```
      SUBROUTINE SSRCH(TABLE, TLENG, VSIZE, TSIZE,
     *                 VDATA, FOUND, NDATA)
      IMPLICIT INTEGER (A-Z)
      INTEGER VDATA(NDATA), FOUND(NDATA)
      INTEGER TABLE(VSIZE,TSIZE), TLENG(NDATA)
*VOPTION NOVEC
      DO 20 I = 1, NDATA
        FOUND(I) = 0
*VOPTION NOVEC
        DO 10 J = 1, TLENG(I)
          IF (VDATA(I) .EQ. TABLE(I,J)) THEN
            FOUND(I) = J
            GOTO 20
          END IF
   10   CONTINUE
   20 CONTINUE
      END
```

4.8.2 ループ非交換版

ループ交換せずにベクトル処理するためのプログラムをしめす．

```
      SUBROUTINE VMSRCN(TABLE, TLENG, VSIZE, TSIZE,
     *                  VDATA, FOUND, NDATA)
      IMPLICIT INTEGER (A-Z)
      INTEGER VDATA(NDATA), FOUND(NDATA)
      INTEGER TABLE(VSIZE,TSIZE), TLENG(NDATA)
      DO 20 I = 1, NDATA
        FOUND(I) = 0
        DO 10 J = TLENG(I), 1, -1
          IF (VDATA(I) .EQ. TABLE(I,J)) FOUND(I) = J
10      CONTINUE
20    CONTINUE
      END
```

4.8.3 マスク演算方式最大回数反復法版

マスク演算方式による最大回数反復法版のベクトル処理プログラムをしめす.

```
      SUBROUTINE VMSRCM(TABLE, TLENG, VSIZE, TSIZE,
     *                  VDATA, FOUND, NDATA)
      IMPLICIT INTEGER (A-Z)
      INTEGER VDATA(NDATA), FOUND(NDATA)
      INTEGER TABLE(VSIZE,TSIZE), TLENG(NDATA)
      MX = 0
      DO 20 I = 1, NDATA
        FOUND(I) = 0
        MX = MAX(MX, TLENG(I))
20    CONTINUE
      DO 40 J = MX, 1, -1
        DO 30 I = 1, NDATA
          IF (VDATA(I) .EQ. TABLE(I,J) .AND.
     *        J .LE. TLENG(I)) THEN
            FOUND(I) = J
          END IF
30      CONTINUE
40    CONTINUE
      END
```

4.8.4 インデクス方式残存要素検出法版

インデクス方式による残存要素検出法版のベクトル処理プログラムをしめす．4.4 節のプログラムと比較すると，内側ループ内の条件文の then 側と else 側がいれかわっている．そのために条件がややわかりにくくなっているとかんがえられるが，これは S-810 / S-820 の Fortran コンパイラの制約によって then 側と else 側をいれかえるとベクトル化されなくなるため，やむなくそうしている．他のサブルーティンとはややインタフェースがことなり，配列 INDEX を引数としてわたしているが，これは INDEX を整合配列としたかったためであり，配列 INDEX をつうじての入出力はおこなっていない．

なお，このプログラムにはベクトル化オプション (*VOPTION) が 1 箇所ふくまれているが，これにより，Fortran コンパイラが完全に解析できない配列 INDEX に関する自己ループ依存性がないことを保証することによってベクトル化を可能にしている．

```
      SUBROUTINE WISRCC(TABLE, TLENG, VSIZE, TSIZE,
     *                  VDATA, FOUND, INDEX, NDATA)
      IMPLICIT INTEGER (A-Z)
      INTEGER VDATA(NDATA), FOUND(NDATA)
      INTEGER TABLE(VSIZE,TSIZE), TLENG(NDATA)
      INTEGER INDEX(VSIZE)
      COUNT = 0
      DO 10 I = 1, NDATA
        FOUND(I) = 0
        IF (1 .LE. TLENG(I)) THEN
          COUNT = COUNT + 1
          INDEX(COUNT) = I
        END IF
   10 CONTINUE
      J = 1
      NDATA1 = COUNT
*     LOOP
   20   CONTINUE
        COUNT = 0
*VOPTION INDEP(INDEX)
        DO 30 I = 1, NDATA1
          IX = INDEX(I)
          VDIX = VDATA(IX)
          TIXJ = TABLE(IX,J)
          IF (J .LE. TLENG(IX)) THEN
            IF (VDIX .NE. TIXJ) THEN
*                               prepare for the next step
              COUNT = COUNT + 1
              INDEX(COUNT) = IX
            ELSE
              FOUND(IX) = J
            END IF
          END IF
   30   CONTINUE
        J = J + 1
        NDATA1 = COUNT
        IF (COUNT .GT. 0) GOTO 20
*     END LOOP
      END
```

4.9 付録 2：配列線形検索の実測データ

第 6 章ではしめさなかったマスク演算方式による残存要素検出法のデータもあわせ，検索データ数 $N = 16,\ 128,\ 1024$ のときの総検索時間を**表** 4.2 ～ 4.4 にしめす．図 4.11 ～ 4.13 に図示したデータは，すでにのべたようにこの表のデータを NM でわった値である．なお，スカラ処理の欄以外は，すべてベクトル処理による検索時間である．また，**図** 4.15 ～ 4.16 のもとになったデータをそれぞれ**表** 4.5 および**表** 4.6 にしめす．

表 4.2　検索データ数 $N = 16$ のときの検索時間および加速率

M	スカラ処理		ループ非交換		最大回数反復法（マスク演算方式）		残存要素検出法（マスク演算方式）		残存要素検出法（インデクス方式）	
	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率
3	0.04	1.00	0.05	0.78	0.03	1.09	0.04	0.88	0.04	0.92
7	0.08	1.00	0.10	0.80	0.07	1.07	0.10	0.77	0.09	0.83
17	0.12	1.00	0.15	0.78	0.12	0.95	0.16	0.72	0.15	0.76
31	0.17	1.00	0.21	0.84	0.19	0.91	0.31	0.56	0.29	0.61
61	0.26	1.00	0.28	0.95	0.30	0.87	0.57	0.46	0.51	0.51
127	0.50	1.00	0.37	1.35	0.49	1.01	1.09	0.46	0.96	0.52
253	0.92	1.00	0.51	1.81	0.85	1.09	2.02	0.46	1.76	0.53
511	2.00	1.00	0.76	2.63	1.56	1.28	4.18	0.48	3.61	0.55

表 4.3　検索データ数 $N = 128$ のときの検索時間および加速率

M	スカラ処理		ループ非交換		最大回数反復法（マスク演算方式）		残存要素検出法（マスク演算方式）		残存要素検出法（インデクス方式）	
	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率
3	0.11	1.00	0.17	0.61	0.04	2.56	0.06	1.78	0.05	2.14
7	0.24	1.00	0.38	0.63	0.10	2.49	0.14	1.65	0.13	1.88
17	0.37	1.00	0.62	0.61	0.18	2.05	0.25	1.50	0.24	1.55
31	0.68	1.00	0.90	0.75	0.31	2.17	0.48	1.41	0.48	1.41
61	1.18	1.00	1.27	0.93	0.54	2.17	0.87	1.35	0.92	1.28
127	2.96	1.00	1.85	1.60	0.98	3.01	1.68	1.76	1.81	1.63
253	6.37	1.00	2.82	2.26	1.82	3.49	3.17	2.01	3.53	1.81
511	14.69	1.00	4.61	3.18	3.48	4.22	6.30	2.33	7.02	2.09

表 4.4　検索データ数 $N = 1024$ のときの検索時間および加速率

M	スカラ処理		ループ非交換		最大回数反復法（マスク演算方式）		残存要素検出法（マスク演算方式）		残存要素検出法（インデクス方式）	
	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率
3	0.65	1.00	1.17	0.56	0.11	6.00	0.19	3.42	0.14	4.74
7	1.55	1.00	2.62	0.59	0.29	5.39	0.48	3.26	0.41	3.76
17	2.49	1.00	4.37	0.57	0.64	3.88	0.80	3.13	0.96	2.58
31	4.86	1.00	6.43	0.76	1.24	3.93	1.57	3.10	2.06	2.37
61	9.24	1.00	9.26	1.00	2.36	3.91	2.81	3.29	4.17	2.21
127	24.04	1.00	13.76	1.75	4.64	5.19	5.52	4.35	8.54	2.82
253	51.27	1.00	21.41	2.39	9.10	5.64	10.15	5.05	17.17	2.99
511	116.94	1.00	35.59	3.29	18.03	6.49	20.46	5.71	34.59	3.38

表 4.5　登録データ上限値の変化による検索時間の変化例 ($N = 1024$, $M = 511$)

M	スカラ処理（マスク演算方式）	ループ非交換（インデクス方式）	残存要素検出法（マスク演算方式）	最大回数反復法
16	4.95	1.80	8.95	28.71
32	12.51	3.59	8.98	28.70
64	29.23	6.34	8.97	28.71
128	56.67	10.72	8.98	28.70
256	86.73	15.13	8.94	28.68
512	116.46	19.72	8.94	28.79
1024	134.50	28.71	8.94	22.71

表 4.6　ループの平均長と最大長との比率による実行時間と加速率の変化例
($N = 1024$, $M = 61$)

平均長 / 最大長	スカラ処理		ループ非交換		最大回数反復法（マスク演算方式）		残存要素検出法（マスク演算方式）		残存要素検出法（インデクス方式）	
	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率	時間 (s)	加速率
0.50	5.47	1.00	5.21	1.05	1.12	4.87	1.96	2.80	2.13	2.58
0.33	9.41	1.00	9.25	1.02	2.25	4.19	3.40	2.77	4.24	2.22
0.20	12.04	1.00	12.07	1.00	3.37	3.58	4.38	2.75	6.34	1.90
0.11	13.75	1.00	13.90	0.99	4.49	3.06	5.06	2.72	8.38	1.64
0.06	14.94	1.00	15.11	0.99	5.60	2.67	5.58	2.68	10.36	1.44
0.03	15.83	1.00	15.94	0.99	6.69	2.36	6.01	2.63	12.21	1.30
0.02	16.55	1.00	16.57	1.00	7.76	2.13	6.38	2.59	13.91	1.19
0.01	17.18	1.00	17.10	1.00	8.77	1.96	6.71	2.56	15.40	1.12

4.10 付録 3：最大回数反復法におけるくりかえし回数 *Mmax* の推定

時刻 0 からはじめて，各時刻 t ($t = 1, 2, \ldots$) において値域が $[0, M)$ の一様乱数をもとめ，時刻 t 以前の試行でえられた値の最大値を $Mmax(t)$ とする．この章では，このときの *Mmax* の推定値をもとめる．時刻 t において *Mmax* の値が m ($m = 0, \ldots, M-1$) である確率を $P(t, m)$ とする．$P(t, m)$ を $P(t, i)$ ($i = 0, \ldots, M-1$) をつかってあらわすと，つぎのようになる．

$$P(t, m) = (m/M)\, P(t-1, m) + (1/M) \sum_{x=0}^{m-1} p(t-1, x). \qquad \cdots\cdots\cdots\cdots (16.1)$$

第 1 項は時刻 $t-1$ に *Mmax* が m よりちいさく (すなわち時刻 t よりまえの試行においては m よりおおきな値がえられず)，時刻 t の試行においてはじめて m がえられる確率をあらわす．第 2 項は時刻 $t-1$ に *Mmax* が m であり，時刻 t の試行で m よりちいさい値がえられる確率である．(16.1) で定義される $P(t, m)$ は，それがみたすべき条件 $\sum_{m=0}^{M-1} p(t, m) = 1$ をみたしている．離散分布のままでは計算がむずかしいため，(16.1) を連続分布によって近似する，すなわち P のかわりに連続分布の確率密度関数 p を定義すると，

$$p(t, x) = (x/M)\, p(t-1, x) + (1/M) \int_0^x p(t-1, x)\, dx. \qquad \cdots\cdots\cdots\cdots (16.2)$$

(16.2) で定義される $p(t, x)$ は，それがみたすべき条件 $\int_0^M p(t, x) = 1$ をみたしている．

$p(t, x)$ の値を $t = 0, 1$ および 2 のばあいについて計算すると，

$$p(0, x) = 1/M,$$

$$p(1, x) = (x/M)\, p(0, x) + (1/M) \int_0^x p(0, x)\, dx = 2x/M^2,$$

$$p(2, x) = 3x/M^3$$

となる．したがって，

$$p(t, x) = (t+1)/M^{t+1}\, x^t \qquad \cdots\cdots\cdots\cdots (16.3)$$

がなりたつことが推測される．実際，(16.3) から

$$p(t+1, x) = ((t+2)/M^{t+2})\, x\, (t+1)$$

が証明できる．したがって，数学的帰納法により (16.3) がしめされる．*Mmax* の推定値 $\underline{Mmax}(N)$ は $p(N, x)$ の x に関する平均値となる．すなわち，

$$\underline{Mmax}(N) = \int_0^x x\, p(N, x)\, dx = ((N+1)/M^{N+1}) \int_0^x x^{N+1} dx = M(N+1)/(N+2).$$

これで，目的の推定値がもとめられた．