第 2 章
解探索のベクトル処理

要旨

この章では，探索問題に適用することができる，ベクトル計算機むきのあたらしい計算法「並列バックトラック計算法」をしめす．この方法にしたがって Fortran でプログラムを記述すれば，数値計算専用とかんがえられていた S-810 のようなスーパコンピュータや，M-680H IAP/IDP (内蔵型アレイ・プロセッサ / 内蔵型データベース・プロセッサ) のようなベクトル計算機構を付加した汎用計算機で広範囲の探索問題を高速に実行することができる．またこの方法では，並列度を適切に制御することによって，必要な記憶量を逐次計算法とひとしいオーダにおさえることができる．

この計算法を N クウィーン問題に適用し，つぎのような実行性能をえた．逐次処理にくらべて，エイト・クウィーンの全解探索においては S-810 を使用して約 9 倍，M-680H IAP および IDP を使用して約 2 倍だった．また，S-810 においては $N \geq 14$ のとき単解探索でも逐次処理より高速に実行することができた．

これによって並列バックトラック計算法の有効性がたしかめられるとともに，ベクトル計算機 S-810 および M-680H IAP/IDP の記号処理への適用可能性がしめされた．また，並列バックトラック計算法は，Prolog のような論理型言語のベクトル計算機による高速実行の可能性を示唆している．

2.1 はじめに

パイプライン型ベクトル計算機 (以後は単にベクトル計算機とよぶ) は，ベクトル (配列) の各要素に同一の演算をパイプライン的に実行する演算機構をもうけることによって，ベクトル計算を高速に実行できるようにした計算機である．

ベクトル計算機のもっとも重要な族として，Cray-1 を祖とするパイプライン型スーパコンピュータ (以後は単にスーパコンピュータとよぶ) がある．スーパコンピュータは，数値計算専用機として発展してきた．その結果，Fortran プログラムを部分的にスーパコンピュータむきにかきかえることにより，おおくの数値計算プログラムを超大型汎用計算機にくらべて 10 倍以上高速に実行することができる．

第 1 世代のスーパコンピュータはベクトルとスカラに関する単純な四則演算以外の命令をほとんどもっていなかった．ところが, HITAC S-810 [Odaka 83], FACOM VP-200 [Hirakuri 83]，日電 SX-2 [Furumasa 84] などの第 2 世代のスーパコンピュータにおいては，大幅な機能の拡張がおこなわれた．すなわち，マスク演算命令，リスト・ベクトル命令，ベクトル圧縮・伸長命令など，条件文のもとの数値計算や複雑な配列添字の計算などに使用される命令がレパートリにくわえられた [Hirakuri 83]．そのため，通常の Fortran プログラムからスーパコンピュータむきのプログラムへの自動変換すなわちベクトル化をおこなうベクトル・コンパイラとあいまって，応用範囲の拡大とベクトル化率の向上による高速化が達成された．

一方，ベクトル計算機のもう 1 つの族として，汎用計算機の付加機構としての内蔵型アレイ・プロセッサ (Integrated Array Processor) がある．すなわち，HITAC M-180 IAP [Horikoshi 83] から M-680H IAP にいたる一連の計算機である．日電 ACOS 1000 IAP [Osaka 82]，IBM 3090 VF (Vector Facility) [Buchnolz 86] などもこの族のなかにくわえることができる．これらの内蔵型アレイ・プロセッサもまた，数値計算専用機として発展してきたが現在では大幅に機能が拡張されているという点は，スーパコンピュータのばあいと同様である．

さらに，汎用計算機の付加機構としては，関係データベースやソーティングの高速処理を目的とした M-680H IDP (内蔵型データベース・プロセッサ Integrated Database Processor) [Torii 87, Kojima 87] が開発されている．上記のベクトル計算機とはちがって IDP は記号処理を目的としているが，関係データベース処理に適したかぎられたデータ形式 (デュアル・ベクトル) だけをあつかうことができる．

このように，いまのところベクトル計算機は，関係データベース，ソーティング [Store 78, Brock 81, Roensch 87, Ishiura 88]，論理シミュレーション [Nagashima 86, Ishiura 86] などをのぞけば，リスト処理で代表される本格的な記号処理には使用されるにいたっていない．しかしそのおもな理由は，ベクトル計算機むきの記号計算のプログラミング方法お

よびベクトル化 (プログラム変換) 方法が開発されていなかったためであり，ベクトル計算機の機能不足のためではない．なぜなら，現在のベクトル計算機は，すでに本格的な記号的ベクトル処理に必要なベクトル命令をほぼひととおりそなえているからである．この研究の目的は，第 1 に，ある種の記号処理においてはベクトル計算機を適用することによって高速化がはかれることをしめすことである．そして第 2 に，高速化の対象としてえらんだ，探索問題という，記号処理のなかでも 1 つのもっとも重要な分野の問題の，ベクトル計算機で実行するのに適した計算方法を開発することである．

この章では，まず 2.2 節で従来からある探索問題の逐次的な計算方法であるバックトラック計算法についてのべるとともに，この方法によるプログラムをベクトル計算機で実行しようとするばあいの問題点をしめす．2.3 節ではバックトラック計算法によるプログラムに最小限の変更をくわえてベクトル処理する AND ベクトル計算法をしめすが，この方法では飛躍的な性能向上はえられない．そこで，2.4 節では上記の問題点を解決したベクトル計算機むきの計算法である OR ベクトル計算法についてのべるが，まず 2.4.1 節では OR ベクトル計算法のもっとも純粋なかたちである完全 OR ベクトル計算法についてのべるとともに，その問題点をしめす．つぎに 2.4.2 節では，その問題点をも解決した計算法である並列バックトラック計算法についてのべる．2.5 節では，並列バックトラック計算法にしたがって記述したプログラムの S-810 および M-680H IAP および IDP を使用したばあいの実行時間を，(逐次) バックトラック計算法によるプログラムの実行時間と比較するとともに，考察をくわえる．2.6 節では，N クウィーン問題においては必要がなかった並列バックトラック計算法におけるベクトル長増大のための方法についてのべる．

2.2 逐次バックトラック計算法

この節ではまず，N クウィーン問題を例として，探索問題をとく際に従来からつかわれてきた逐次計算法であるバックトラック計算法についてかんたんにのべる．並列バックトラック計算法と明確に区別するため，以後は「逐次バックトラック計算法」とよぶ．そしてつぎに，逐次バックトラック計算法にしたがって記述されたプログラムをそのままベクトル計算機で実行しようとするときに生じる問題点をしめす．

N クウィーン問題は，パズルの一種であるエイト・クウィーン問題を一般化したものである．すなわち，$N \times N$ のおおきさの「チェス・ボード」に，N 個のクウィーンを，どの 2 つもおなじ行，列，および対角線方向にないように配置する問題である．**図** 2.1 にエイト・クウィーン問題の 92 個の解のうちの 1 つをしめす．また，**表** 2.1 に N クウィーン問題の解の数をしめす．エイト・クウィーン問題は代表的な探索問題であり，Lisp や Prolog などのベンチマーク・プログラムとして，はばひろく使用されている [Okuno 84].

エイト・クウィーン問題の解法に関してはおおくの研究がおこなわれている．その解法はたとえば Bitner and Reingold [Bitner 75]，Dijkstra [Dijkstra 72]，Floyd [Floyd 84] にのべられている．Bitner and Reingold と Dijkstra のプログラムは逐次バックトラック計算法にもとづいている．また，Floyd のプログラムは，非決定性プログラミング法 [Floyd 84] にもとづいていて，Prolog によるエイト・クウィーンのプログラム [Okuno 84] のもとになっているといえる．Floyd のプログラムもその実現手段としては逐次バックトラック計算法がつかわれる．

逐次バックトラック計算法にもとづくエイト・クウィーン問題の全解探索 (すべての解をもとめること) の解法をしめす．解は 8 個の整数からなるリスト $(x_1, x_2, \ldots, x_8)$ $(0 \le x_i \le 7)$ で表現される．各 x_i が第 i 列第 x_i 行におかれたクウィーンをあらわす．たとえば図 2.1 にしめした解は $(3, 1, 7, 2, 5, 7, 0, 4)$ とあらわされる．

図 2.2 がそのアルゴリズムである．図 2.2 のプログラムは $x_1, x_2, \ldots, x_8$ をこの順に決定していく．x_i を決定する際には，x_i の値をかりにさだめて，安全性チェックをおこなう．すなわち，$x_1, x_2, \ldots, x_{i-1}$ があらわすすでに盤面におかれたクウィーンが，x_i があらわすクウィーンと同一の行にないかどうか，また対角位置にないかどうかをチェックする (すなわち，$x_i \neq x_j$, $x_i \neq x_j \pm (i-j)$ $(0 \le j \le i)$ がなりたつかどうかをしらべる)．もしその条件がみたされないときは，x_i の値をかえて，つぎの候補をさがす．x_i のすべての候補をつくしたときは，x_{i-1} の値をかえて，つぎの候補をさがす．図 2.2 のプログラムではこのようにしてエイト・クウィーンの全解探索をおこなう．なお，**付録** 1 に図 2.2 のアルゴリズムを Fortran でコーディングした例をしめす．

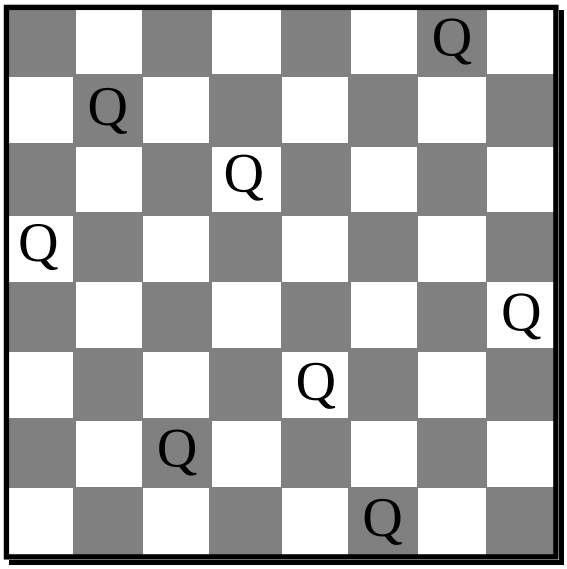


図 2.1　エイト・クウィーン問題の解の一例

表 2.1　*N* クウィーン問題の解の数 ($N \leq 13$)

N	1	2	3	4	5	6	7	8	9	10	11	12	13
解の数	0	0	0	2	10	4	40	92	352	724	2680	44200	73712

```
B := emptyChessBoard;                    — B を空のチェス・ボードとする.
QUEEN(B, 1);
    — 手続き QUEEN をよび, エイト・クウィーン問題の解をもとめて印刷する.

procedure QUEEN(B: chessBoard; x: row) is
    if x > 8 then
        PRINT(B);                        — すべてのクウィーンがおかれたので印刷する.
    else
        for y in 1 .. 8 loop
            if not TAKEN(B, x, y) then   — B 上の点 (x, y) にクウィーンをおいても
                                         — ほかのクウィーンにとられないなら,
            PUT_QUEEN(B, x, y);          — 点 (x, y) にクウィーンをおく.
                QUEEN(B, x + 1);         — のこりのクウィーンをおき, 解を印刷する.
                REMOVE_QUEEN(B, x, y);   — 点 (x, y) のクウィーンをとりのぞく.
            end if;
        end loop;
    end if;
end QUEEN;
```

図 2.2　エイト・クウィーン問題の逐次バックトラック計算法

2.3 AND ベクトル計算法 (AND 並列計算法)

逐次バックトラック計算法によるエイト・クウィーン問題のプログラムでもっとも計算時間がかかるのは，上記の安全性チェック，すなわち図 2.2 のプログラム上では関数 *TAKEN* の実行である (ただし，関数 *TAKEN* の定義は図 2.2 にしめしていない)．関数 *TAKEN* は，多少の注意をはらって Fortran で記述すれば，自動ベクトル・コンパイラでベクトル化することができる．すなわち，ベクトル計算機で実行できるようにプログラム変換することができる．このような計算法を AND ベクトル計算法とよぶことにする[注1]．**図** 2.3 にこのプログラムの S-810 Model 20 における実測結果をしめす．この図からわかるように，すくなくとも S-810 のばあいには，ベクトル化後のプログラムの実行速度はベクトル化前のそれにくらべて，かえっておそくなってしまう．それは，ベクトル長がみじかい (すでに盤面に配置されたクウィーン数以下) うえ，逐次実行にくらべてむだな計算がふえるからである．

以下，このプログラムとその実行結果についてよりくわしく解析する．図 2.2 のプログラムにおける関数 *TAKEN* においては，これからおくべきクウィーン x_i と，すでにおかれたクウィーンのそれぞれ x_j とが条件 $x_i \neq x_j,\ x_i \neq x_j \pm (i-j)$ をみたすかどうかをチェックする．したがって，すでにおかれたクウィーンの数だけのくりかえしが最内側ループとなる．しかも，条件をみたさないことがループの途中でわかれば，以後のくりかえしは実行する必要がないので，**goto** 文でループ外に脱出することになる．したがって，エイト・クウィーンのばあいで平均ループ長はわずか 4 程度になる．

このプログラムはループ外への脱出をなくせばベクトル化することができる．S-810 の Fortran コンパイラではこの変換は自動的におこなわれる．しかし，ループ外脱出をなくすことによってむだな計算がふえるうえ，もともとループ長がみじかいためにベクトル長がみじかい．したがって，図 2.3 にしめしたように，ループ外脱出があるプログラムと脱出をなくしてベクトル化したプログラムとをくらべると，ベクトル化によってかえって実行時間は増加してしまう．したがって，AND ベクトル計算法は *N* クウィーン問題のプログラムをベクトル化する方法としてはつかえない．

[注1] AND ベクトル計算法というなまえの由来は Prolog によって記述したエイト・クウィーン問題のプログラムにおいて AND 関係 (第 3 章参照) となる計算を並列化してベクトル処理していることによる。このなまえについては次章でさらにのべる．

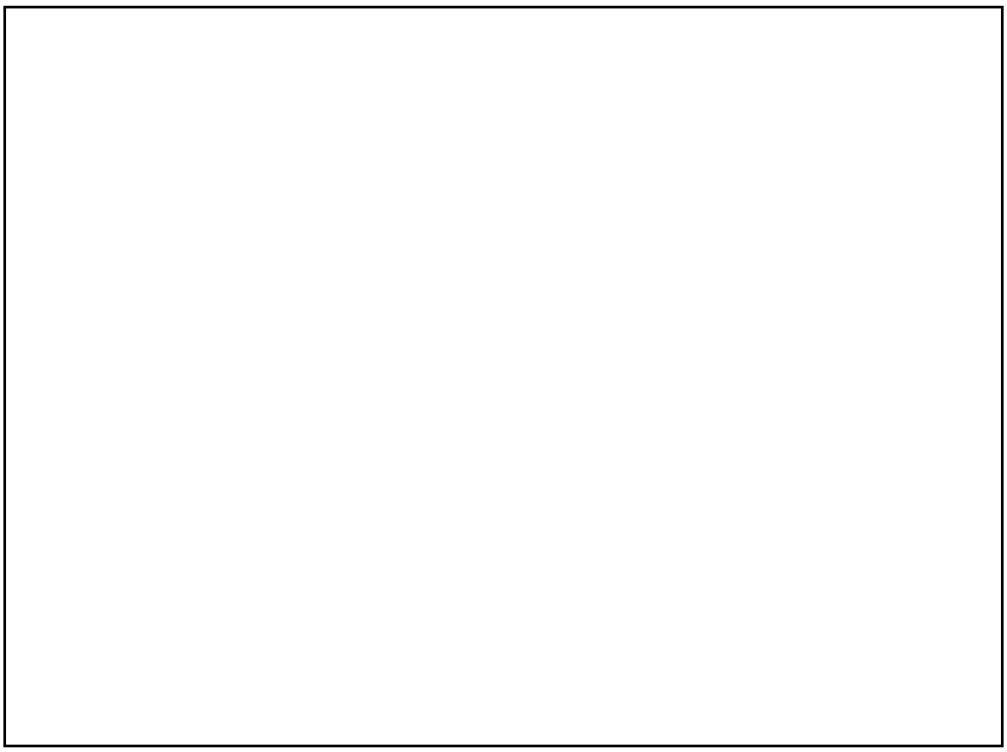


図 2.3　AND **ベクトル計算法による** *N* **クウィーンの相対実行時間**
(S-810 **旧 Fortran コンパイラ**)

2.4 OR ベクトル計算法 (OR 並列計算法)

この節では，探索問題の計算法である OR ベクトル計算法についてのべる．この方法にしたがって記述されたプログラムは，Fortran コンパイラでベクトル化することができ，かつそれによって高速化されることが期待できる．2.4.1 節では単純な OR ベクトル計算法 (完全 OR ベクトル計算法とよぶ) とその性能および問題点についてのべる．また，2.4.2 節ではその問題点を解決した計算法である並列バックトラック計算法についてのべる．

2.4.1 完全 OR ベクトル計算法

2.3 節でわかったように，逐次バックトラック計算法にもとづく探索問題のプログラムをそのままたは最小限の変更をくわえて Fortran コンパイラでベクトル化しても，十分な高速実行を期待することはできない．高速化のためには計算法のみなおしが必要である．

そこで，逐次バックトラック計算法を再検討してみる．2.3 節のエイト・クウィーン問題のプログラムをベクトル化してできた目的プログラムを実行すると，1 つの解をもとめる計算の一部が並列に (正確にはパイプライン的に) 実行されることになる．したがって，Prolog の並列処理法との類比でかんがえれば，この計算法は AND 並列処理法に相当する．AND ベクトル計算法とよんだのはそのためである．これに対して，ことなる解をもとめる計算を並列に実行する方法をかんがえることができる．この計算法は OR 並列処理法に相当する．したがって，この計算法を OR ベクトル計算法とよぶ.

OR ベクトル計算法を，エイト・クウィーン問題の全解探索を例として，逐次バックトラック計算法と比較しながら説明する．すでにのべたように，逐次バックトラック計算法では，探索の進行にともなって選択枝が生じるたびに，そのうちの 1 つをえらんで実行する．そして，その選択枝が失敗するとあともどり (backtrack) して，ほかの選択枝をあらためて実行する．これに対して OR ベクトル計算法では，すべての解候補の集合の各要素に対して並列に計算をすすめる．エイト・クウィーンのばあいには，解候補はクウィーンがのせられたチェス・ボードである．解候補および解候補の集合のデータ表現として配列を使用することによって，ベクトル計算機による並列処理 (パイプライン処理) を可能にする．OR ベクトル計算法によるエイト・クウィーンの計算過程の概要を**図** 2.4 にしめす．選択枝が生じるたびに，その数だけ解候補を複写してそれぞれにことなる選択の結果を反映し，それらのあたらしい解候補の全体からなる集合をつくる．そして，つぎの計算ステップはその集合全体に対して並行しておこなう．

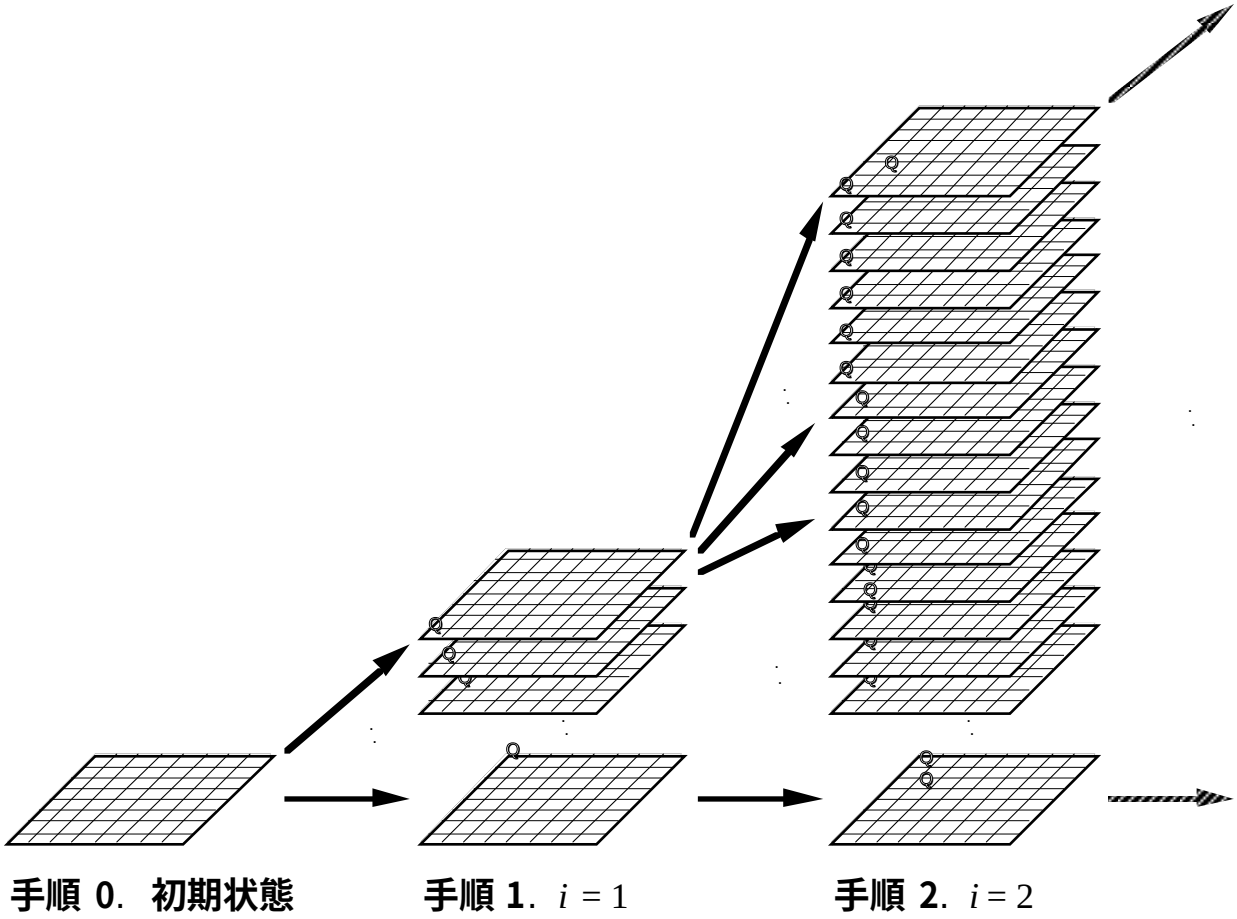


図 2.4　OR ベクトル計算法による計算過程の概要

図 2.5 にアルゴリズムの概要をしめす．図 2.5 のプログラムでは，まず変数 *C* に空のチェス・ボードからなる集合を代入する．そして，第 1 列から順に 8 個のクウィーンをおいていく．このプログラムにおいては，最初の代入文と関数 *nextCandidate* とが生成検査法 (Generate-and-test Method) における生成 (ただし部分的な解候補の生成) のはたらきをし，関数 *nextRow* が検査のはたらきをしている．

```
C := {emptyChessBoard};          — C を 1 個の空のチェス・ボードからなる集合 (ベクトル) とする．
for i in 1 .. 8  loop
    C1 := nextRow(C,  i);
        — C の各要素の第 i 列におくことができるクウィーンの番号 x とチェス・ボード b
        — との対 ⟨x, b⟩ からなる集合 (ベクトル) を C1 とする．

    C := nextCandidate(C1);
        — C1 の各要素 ⟨x, b⟩ におけるチェス・ボード b にクウィーンをおいたあらたな
        — チェス・ボードからなる集合 (ベクトル) を C とする．
end loop;
PRINT(C);
```

図 2.5　エイト・クウィーン問題の OR ベクトル計算法

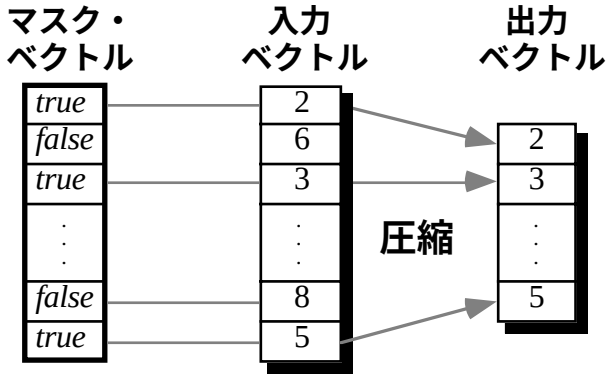


図 2.6　ベクトル計算機のベクトル圧縮命令の動作

関数 *nextRow*(*C, i*) はつぎのような機能の関数である．*nextRow* の入力は，第 1 ～ *i* – 1 列にクウィーンがおかれたチェス・ボードの集合 *C* と，つぎにクウィーンをおくべき列の番号 *i* とである．また，その出力すなわち関数値は，*C* の各要素の第 *i* 列におくことができるクウィーンの番号 *x* (0 ≤ *x* ≤ 7) とチェス・ボード *b* (∈ *C*) との対 ⟨*x, b*⟩ からなる集合である．すなわち，安全性チェックをおこない，それをみたす解候補を生成するのに必要なデータ ⟨*x, b*⟩ を出力する．なお，同一のチェス・ボード *b* が複数回使用される可能性があるが，この段階では *b* の複写はおこなわず，同一のデータをくりかえし使用する．また，関数 *nextCandidate*(*rb*) は上記の対 ⟨*x, b*⟩ を入力し，チェス・ボード *b* にクウィーン *x* をおいたあらたなチェス・ボードからなる集合を値とする関数である．

チェス・ボード b は複数の対に使用されている可能性があるので，クウィーン x をおくまえに複写する必要がある．

図 2.5 のプログラムで実行比率がたかい部分は関数 *nextRow*(*C, i*) と関数 *nextCandidate*(*rb*) とである．このうち関数 *nextRow*(*C, i*) は，第 2 世代のベクトル計算機においてはベクトル圧縮命令をつかうことによって高速に実行することができる．ベクトル圧縮命令とはベクトルの要素のうち条件をみたすものだけを選択し，圧縮されたベクトルを出力する命令である (**図** 2.6 参照)．また，M-680H IDP には，$\langle x, b \rangle$ のような対を要素とするベクトル (デュアル・ベクトル) を条件にしたがって圧縮する命令が用意されている [Kojima 87, Kojima 90] ので，やはり上記の機能を実現することができる．また，関数 *nextCandidate*(*rb*) の機能はベクトル計算機がもつベクトルのロード命令およびストア命令を使用することによって実現できる．なお，**付録** 1 に図 2.5 のアルゴリズムを Fortran でコーディングしたプログラム例をしめす．

つぎに，OR ベクトル計算法の性能予測についてのべる．OR ベクトル計算法では，選択点ごとに解候補を選択枝の数だけ複写しなければならないという，逐次バックトラック計算法にはないオーバヘッドがある．したがって，OR ベクトル計算法のプログラムをスカラ実行したばあいには，逐次バックトラック計算法のプログラムより低速になることが予想される．しかし，そのオーバヘッドにもかかわらず，ベクトル実行すれば逐次バックトラック計算法にくらべて実行時間は高速になりうる．それは，各解候補の計算をパイプラインにのせて短ピッチで計算できるためである．

OR ベクトル計算法は高速ではあるが，つぎのような 2 つの問題点がある．

第 1 の問題点は，解候補数に比例する記憶量が必要だという点である．N クウィーン問題のばあい，N がおおきくなるにつれて指数的に解の数が増加するので，現在の計算機の主記憶容量では N が 15 程度以上のとき計算不能になってしまう．また，計算可能なばあいでも，汎用計算機においては解候補がキャッシュや実記憶からあふれるために計算速度が低下する．

第 2 の問題点は，OR ベクトル計算法は単解探索すなわちただ 1 つ解をもとめるばあいに不向きだという点である．すなわち，すべての解を並列にもとめるため，単解探索のばあいにはむだな計算がおおくなる．

2.4.2 並列バックトラック計算法

2.4.1 節でのべた OR ベクトル計算法の 2 つの問題点を解決するために考案したのが，この節でのべる並列バックトラック計算法である．

一般にベクトル計算機においては，ベクトル長が十分にながければ逐次計算機より 1 桁程度高速に計算することができるが，ベクトル長が 2 〜 10 以下ではベクトル計算の準備などのオーバヘッドのために逐次計算機よりかえって低速になる (クロス・ポイントは

機種によってことなる)．ベクトル長がこれよりながくなるとしだいに性能が向上し，ベクトル長が 64 〜 1000 程度でほぼピークに達する．そして，それ以上のベクトル長では，ベクトルの 1 要素あたりの実行時間は一定か，またはキャッシュのヒット率低下のためにかえってながくなる．

したがって，OR ベクトル計算法においてベクトル長が十分にながくなったときは，つぎのようにするのがよいとかんがえられる．まず，ベクトルを 2 個以上に分割して，そのうちの 1 つについて計算を続行する．そして，その計算がおわったあとであともどりして，のこりのベクトルに関する計算をおこなう．この計算法を並列バックトラック計算法とよぶ．並列バックトラック計算法によるエイト・クウィーンの計算過程の概要を**図** 2.7 にしめす．

並列バックトラック計算法による N クウィーンの全解探索のアルゴリズムを**図** 2.8 にしめす．図 2.8 のプログラムでは，解候補の数 $|C'|$ が定数 *ulim* をこえたとき，解候補の集合 (ベクトル) を 2 個に分割する．解候補の集合の分割の方法としてはより巧妙な方法もかんがえられる．各種の分割法の比較はおこなっていないが，図 2.8 の分割法は単純なわりには比較的よいとかんがえられる．たとえば一度に 3 個以上に分割すると，平均ベクトル長が必要以上にみじかくなるので，2 分割のほうがこのましいとかんがえられる．なお，図 2.8 のアルゴリズムを Pascal と Fortran とでコーディングした例を**付録** 1 にしめす．

並列バックトラック計算法においては，2.4.1 節でのべた OR ベクトル計算法の 2 つの問題点はつぎのように解決されている．

第 1 の問題点すなわち解の候補数に比例する記憶が必要だという点については，つぎのようにいえる．並列バックトラック計算法で必要な記憶量は，集合の分割が頻繁におこなわれるばあいには，N に関して指数的にふえる解の候補数に依存しない．ただし，記憶量は *ulim* にほぼ比例するので，全解探索のばあい，十分な高速化のために *ulim* を数 100 程度とすると，逐次バックトラック計算法のばあいにくらべて 2 桁程度おおい記憶量が必要である．

第 2 の問題点すなわち単解探索に不向きだという点については，つぎのようにいえる．単解探索のばあいは，最初の解がもとめられた時点 (印刷がおわった直後) で計算をうちきるようにすれば，すべてではないにせよ，かなりのむだな計算をへらすことができる．たとえば，N クウィーンの逐次実行においては，1 つの解をもとめるだけでもかなりの量のバックトラックをくりかえすことが必要だから，単解探索においても逐次バックトラック計算法より高速に実行できる可能性がある．

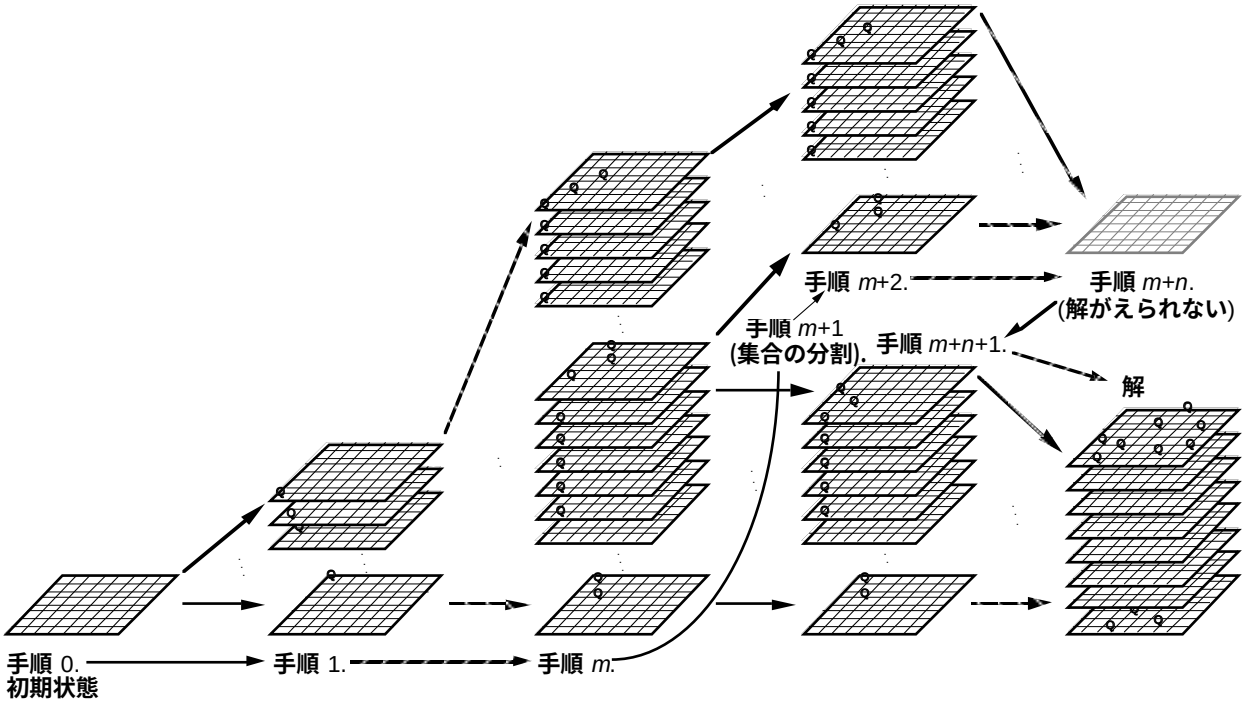


図 2.7　並列バックトラック計算法による計算過程の概要

```
B := {emptyChessBoard};                    — B を 1 個の空のチェス・ボードからなる集合とする.
QUEEN(B, 1);    — 手続き QUEEN をよび, エイト・クウィーン問題の解をもとめて印刷する.

procedure QUEEN(C : chessBoard; x : row) is
    if x > 8 then
        PRINT(C); — すべてのクウィーンがおかれたので印刷する.
    else
        C' := nextCandidate(nextRow(C, i));
            — C がふくむチェス・ボード b にクウィーンをおいたあらたな
            — チェス・ボードからなる集合を C' とする.
        if |C'| > ulim then                — もし C' の要素数が ulim よりおおきければ
        split C' into C1, C2;              — 解候補の集合 C' を 2 つに分解する.
            QUEEN(C1, x + 1);   — 解候補の集合 C1 に由来するすべての解をもとめて印刷する.
            QUEEN(C2, x + 1);   — 解候補の集合 C2 に由来するすべての解をもとめて印刷する.
        else
            QUEEN(C', x + 1);   — 解候補の集合 C に由来するすべての解をもとめて印刷する.
        end if;
    end if;
end QUEEN;
```

図 2.8　エイト・クウィーン問題の並列バックトラック計算法

2.5 *N* クウィーン解探索の実行結果

逐次バックトラック計算法と並列バックトラック計算法による *N* クウィーン問題のプログラムを計 3 本記述し，その全体および部分ごとの実行時間を測定した．OR ベクトル計算法の実行時間も測定したが，並列バックトラック計算法の実行時間とほぼひとしいので，ここでは省略する．

並列バックトラックのプログラムは非 IDP 用および IDP 用の 2 本であり，これらのプログラムは図 2.8 にしめしたように再帰よびだしをふくんでいる．そのため，再帰部分は Pascal で記述した．また，ベクトル計算部分は Fortran で記述して自動ベクトル化した．IDP は Fortran からは使用できないため，IDP 用のプログラムにおける IDP 使用部分はアセンブラで記述した．比較に使用した逐次バックトラック計算法のプログラムはすべて Fortran で記述した．IDP 用をのぞくこれらのプログラムについては，**付録 1** でさらに説明する．IDP における OR ベクトル計算法 (および並列バックトラック計算法) の実現法については鳥居ら [Torii 88a, Torii 88b] にかんたんに記述されている．

図 2.9 に S-810 による *N* クウィーンの全解探索の実行時間をしめし，**図** 2.10 にそのときの加速率をしめす．S-810 においては，並列バックトラック計算法のほうが測定したすべての *N* において逐次バックトラック計算法より高速である．エイト・クウィーンのばあいには実行時間は 8.7 ms であり，逐次バックトラック計算法の約 9 倍の速度である[注2]．また，S-820 によってエイト・クウィーンのばあいを測定したところ，スカラ処理時間は 53.4 ms，ベクトル処理時間は 3.4 ms であり，したがって加速率は 15.7 に達した．

[注2] 金田 [Kanada 84] では 4.5 倍の性能にとどまっていたが，その後の Fortran コンパイラの性能向上によって 9 倍に達した．

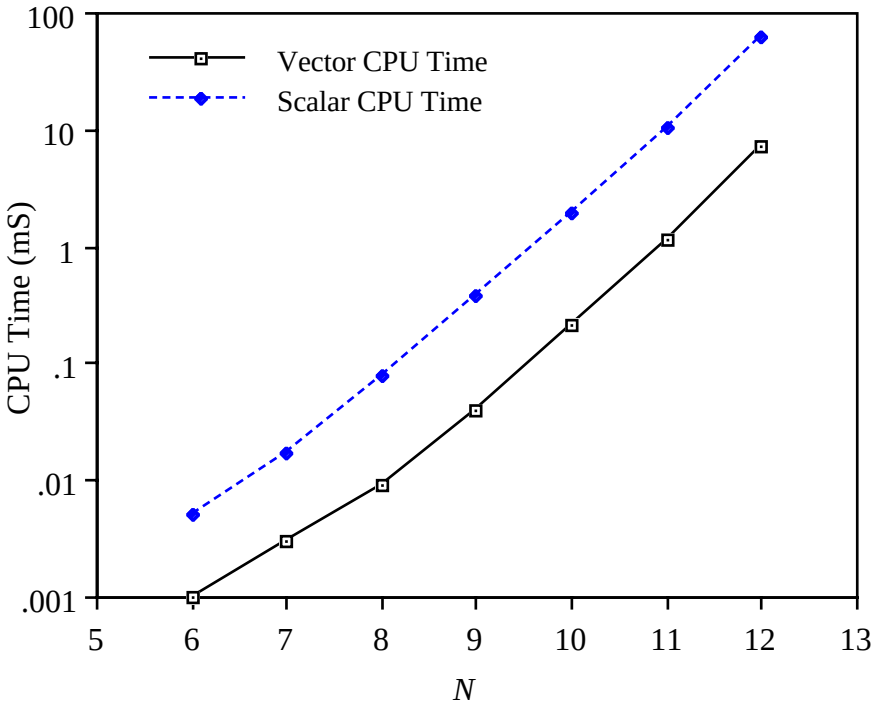


図 2.9　*N* **クウィーン全解探索実行時間** (S-810)

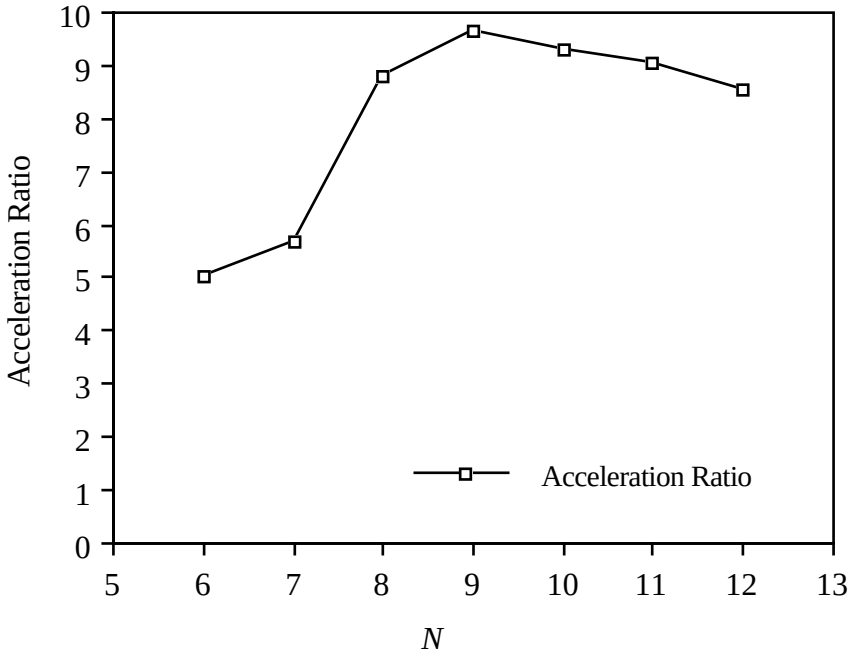


図 2.10　*N* **クウィーン全解探索における加速率** (S-810)

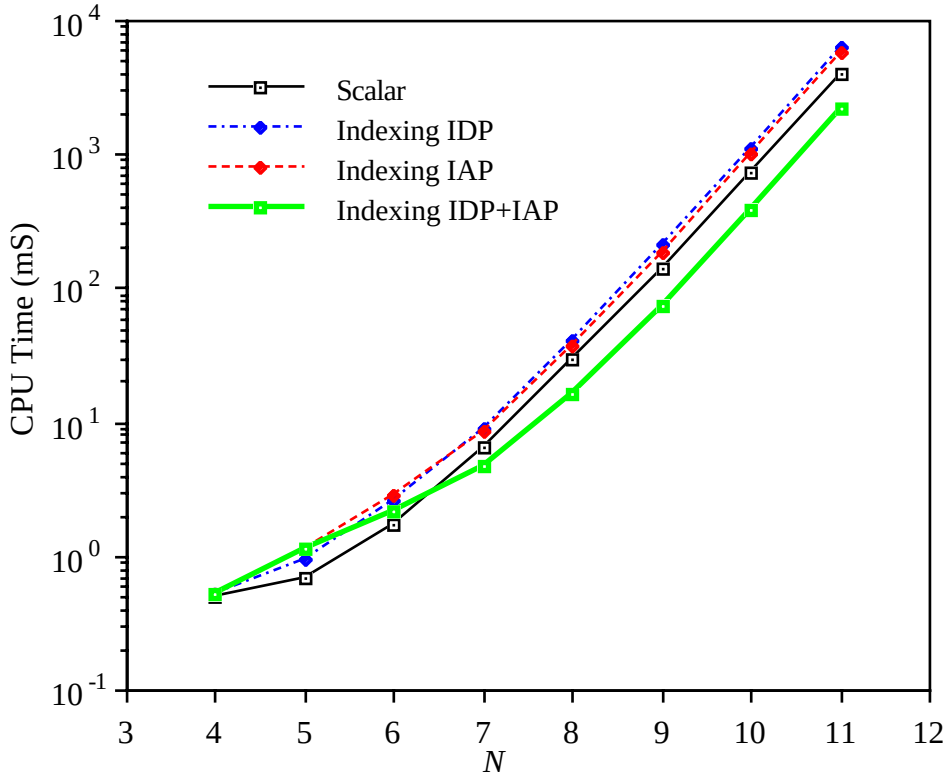


図 2.11　*N* クウィーン全解探索実行時間 (M-680H　IAP / IDP)

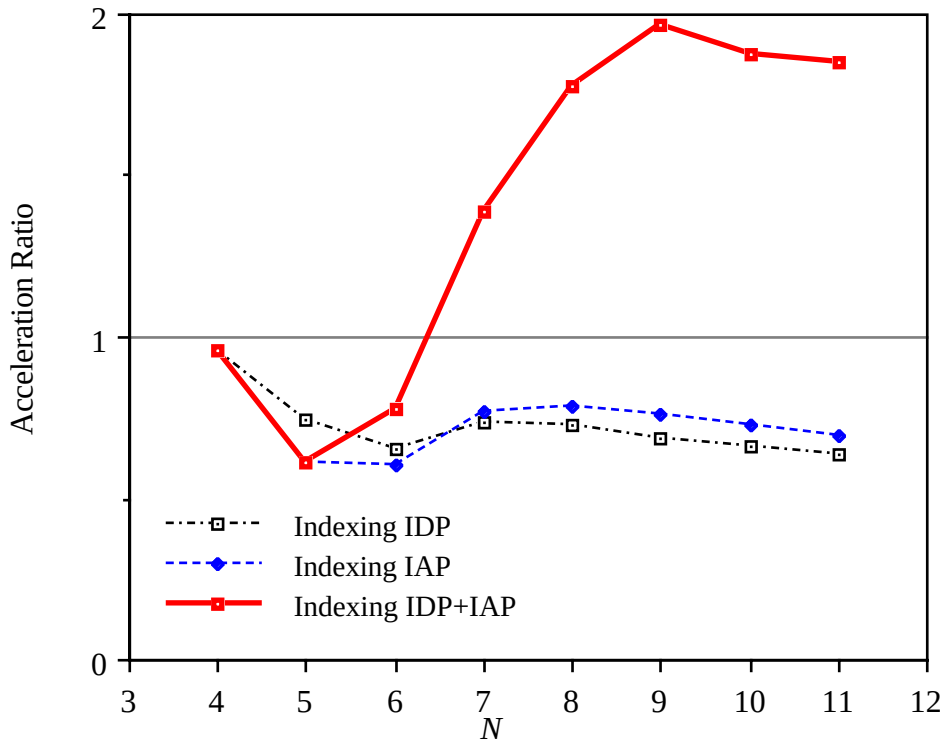


図 2.12　*N* クウィーン全解探索における加速率 (M-680H　IAP / IDP)

図 2.11 に M-680H IAP，IDP による N クウィーンの全解探索の実行時間をしめし，**図** 2.12 にそのときの加速率をしめす．M-680H IAP だけ，あるいは M-680H IDP だけを使用したばあいには逐次バックトラック計算法をうわまわる性能はえられていない．しかし，M-680H IAP，IDP をともに使用したばあいにおいては並列バックトラック計算法のほうが逐次バックトラック計算法より高速になっている[注3]．これは，N クウィーン問題のプログラムにおいては数値計算 (整数加減・比較) と記号処理とをともにおこなうためだとかんがえられる．エイト・クウィーンのばあいには実行時間は 17 ms であり，逐次バックトラック計算法の約 1.8 倍の速度である．

図 2.13 には，S-810 におけるエイト・クウィーン全解探索の 3 種類の実行時間をしめす．その 3 種類とは，逐次バックトラック計算法によるプログラムのスカラ実行，並列バックトラック計算法によるプログラムのスカラ実行およびベクトル実行である．並列バックトラック計算法は配列複写などのオーバヘッドがあるため，スカラ実行すると逐次バックトラック計算法のプログラムの 1.5 倍の実行時間がかかるが，これをベクトル実行することによって逐次バックトラック計算法の 1/9 にまで加速されることがわかる．

$N = 8$ すなわちエイト・クウィーンにおける並列バックトラック計算法の各プログラムの実行時間のうちわけを，図 2.13 にあわせてしめした．一般にプログラムを並列処理するばあいには，しばしばデータを複写する必要が生じるが，並列バックトラック計算法のプログラムにおいても，逐次バックトラック計算法では必要がなかった配列の複写・圧縮が必要になっている．しかし，その実行時間が全体にしめる割合が N クウィーンのばあいにはすくないことがわかる．これが，逐次バックトラック計算法より高速に実行することができた理由である．

図 2.14 は並列バックトラック計算法における記憶消費量の実測値をしめす．この図の縦軸はベクトルの記憶わりあて時におけるその要素数を累積したものである．l (= *ulim*) はベクトル以外の記憶消費量はふくんでいない．したがって逐次実行との比較もしていない．しかし，*ulim* の値がおおきくても N が十分おおきいときには *ulim* の値がちいさいばあいの記憶消費量にほぼ比例する記憶消費量で計算できることがわかる．すなわち，並列バックトラック計算法によって記憶消費量の爆発がさけられていることがわかる．

図 2.15 に S-810 による N クウィーンの単解探索の実行時間，**図** 2.16 にその加速率をしめす (ここでも $l = ulim$ である)．$N < 14$ のときは並列バックトラック計算法は逐次バックトラック計算法にくらべて低速だが，$N \geq 14$ のときは *ulim* の値がおおきく，したがってベクトル長が十分とれれば高速である．**表** 2.2 に，並列バックトラック計算法において計算のうちきり時にもとめられた解の数をしめす．この数が 1 にちかいほどむだな計算がすくなかったといえる．図 2.15, 2.16 から，つぎのようなことがいえる．N クウ

[注3] なお，$N \leq 5$ のばあいは誤差がおおきいため，データをしめしていない．

ィーン ($N \geq 14$) においては 1 つのクウィーンをもとめるまでの計算に十分な OR 並列性があり，単解探索においても並列バックトラック計算法で十分な並列度がえられる．

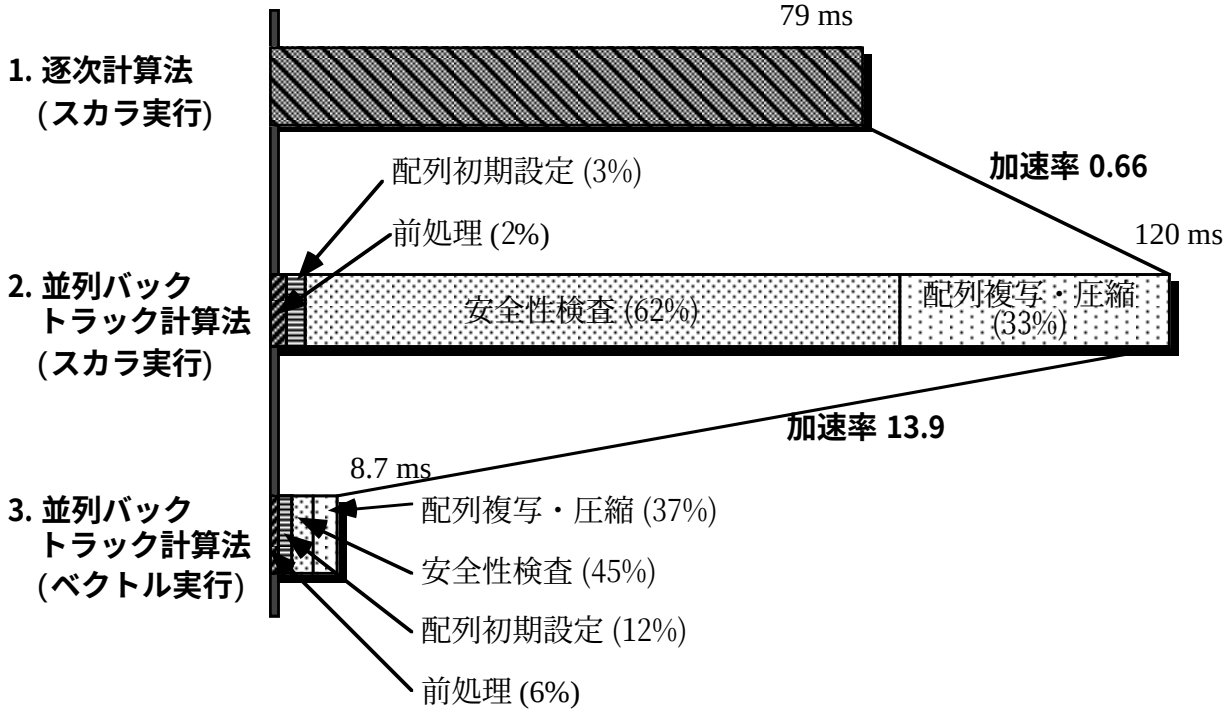


図 2.13　各種実行法によるエイト・クウィーン全解探索の実行時間

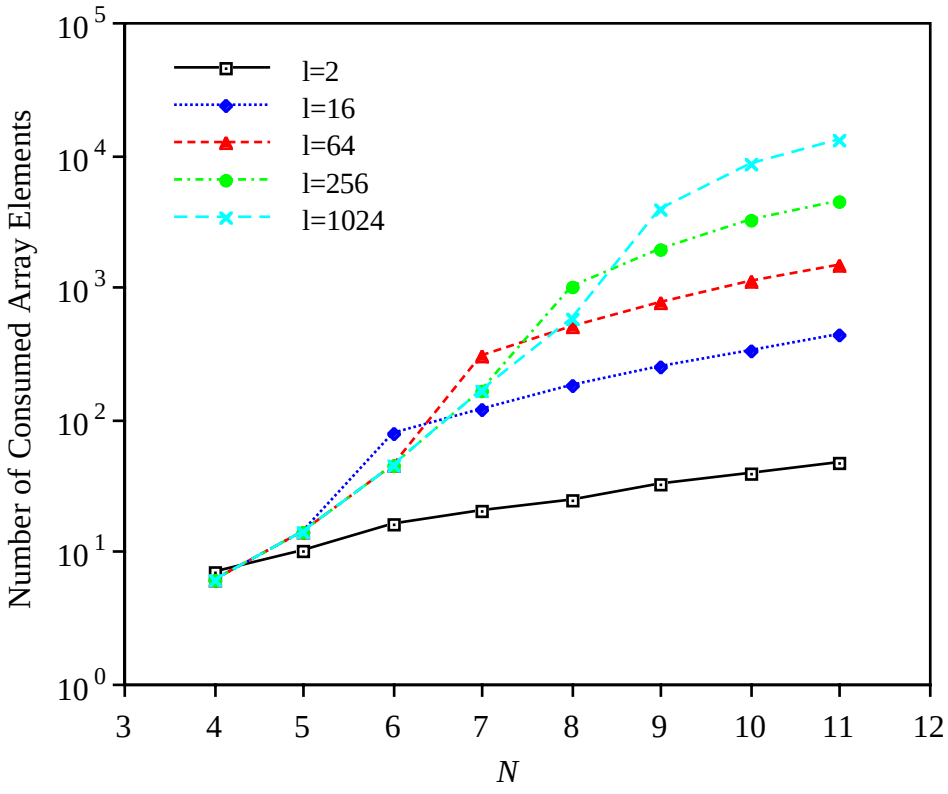


図 2.14　*N* クウィーン全解探索における記憶消費量 (S-810)

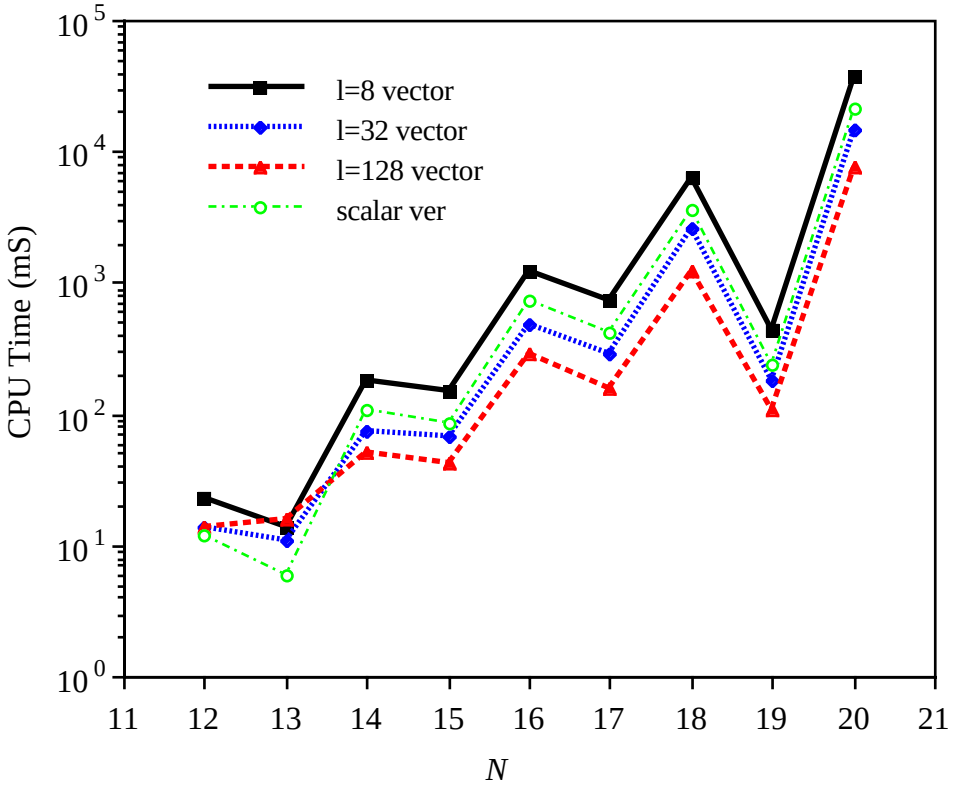


図 2.15　*N* クウィーン単解探索の実行時間 (S-810)

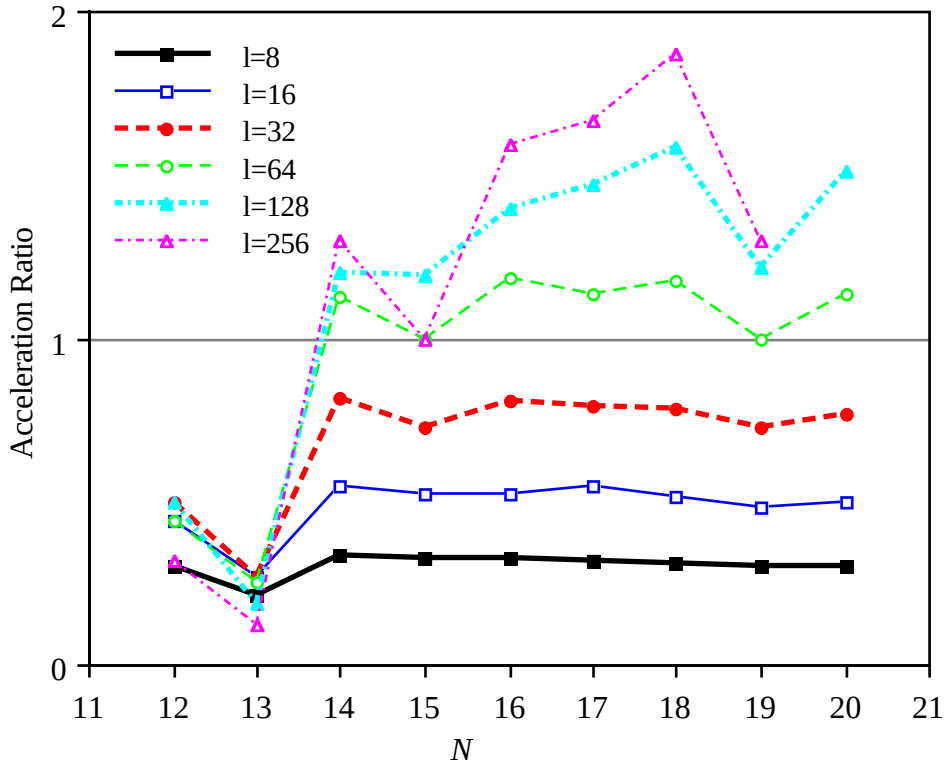


図 2.16　*N* クウィーン単解探索における加速率 (S-810)

表 2.2　*N* クウィーン単解探索で計算うちきりまでにもとめられた解の数 (- は未計算)

分割ベクトル長	*N*=8	*N*=9	*N*=10	*N*=11	*N*=12	*N*=13	*N*=14	*N*=15	*N*=16	*N*=17	*N*=18
2	1	1	1	1	2	1	1	1	1	1	1
4	1	1	1	1	1	1	2	1	1	1	1
8	1	2	2	2	1	1	2	1	1	1	1
16	3	3	1	1	1	1	1	1	1	1	1
32	6	5	6	1	2	1	1	1	1	1	1
64	20	32	19	1	3	1	1	-	-	-	1
128	23	64	17	15	9	4	1	-	-	-	

2.6 並列バックトラック計算法におけるベクトル長増大法

N クウィーン問題の並列バックトラック計算法によるプログラムにおいては，単純な並列バックトラック計算法によってつねにほぼ適当なベクトル長を確保することができたため，2.5 節にしめしたように満足すべき加速率をえることができた．しかし，問題によってはベクトル長が爆発的に増大したあと減少し，みじかいベクトル長のもとで計算がつづけられるばあいがありうる．このようなばあいには，ベクトル計算機の性能をひきだすことができない可能性があるので，つぎのように，はやめにバックトラックする方法によってベクトル長を増大させることがのぞましいとかんがえられる [Kanada 85].

解候補ベクトルの要素数がある一定数をしたまわったときには，そのときのプログラム上の点と解候補ベクトルとを保存してバックトラックする．あとでふたたびプログラム上の同一点まで計算がすすんだときに，あらたにもとめられた解候補ベクトルと保存してあった解候補ベクトルとを併合してひとつの解候補ベクトルとし，それについて計算をすすめる．これは，並列バックトラック方式における解候補ベクトルの分割と逆の操作である．この方式を金田 [Kanada 85] は**残留バックトラック方式**とよんでいる．残留バックトラック方式を実現するためには，どのようにして解候補ベクトルとその環境を保存するか，またどのような条件がなりたったときに解候補ベクトルを併合するかをきめるのが重要であり，そこに研究の余地がある．

2.7 まとめ

この節では，探索問題に適用することができる，ベクトル計算機むきの計算方法である並列バックトラック計算法をしめすとともに，それを N クウィーン問題に適用して，最高 15.7 というたかい加速率がえられることと，しかも並列バックトラック計算法においては記憶消費量の爆発なしにそれが達成されることをたしかめた．また，同時に S-810 や M-680H IAP/IDP のようなベクトル計算機の記号処理への適用可能性がしめされたといえる．

しかしながら，並列バックトラック計算法のプログラムは，ユーザが直接記述するにはあまりにも複雑であり，それを容易にすることが重要な課題だとかんがえられる．すなわち，より単純なプログラムから並列バックトラック計算法のプログラムを自動生成することがのぞましい．ところが，逐次バックトラック計算法にしたがって記述されたプログラムから並列バックトラック計算法のプログラムへのプログラム変換をおこなうのは非常に困難である．バックトラック制御が陽に記述されていない Prolog のような論理型言語のプログラムからの変換は，逐次バックトラック計算法による手続き型言語のプログラムからの変換よりは容易に実現できるとかんがえられる．したがって，それを最初の目標とするのが適当であろう．

また，逆に論理型言語の OR 並列処理方式 [Conery 81] の一種として並列バックトラック計算法をみると，論理型言語のベクトル計算機による高速実行の可能性をしめしているとともに，並列度の制御という点でも示唆をあたえているとかんがえられる．この章でしめしたプログラムはデータを表現するのに配列を使用しているが，論理型言語の実行方式とするためには，リスト処理などのベクトル化が必要である．しかし 2.1 節でものべたように，第 2 世代のスーパコンピュータや内蔵型アレイ・プロセッサはリスト処理に必要な命令もそなえている．したがって，プログラム変換によって十分なベクトル長がとれさえすれば，高速化されることはまちがいないとかんがえられる．

2.8 付録 1：測定用プログラムとその説明

測定に使用した各プログラムの主要部分を**図** 2.17 ～ 2.19 にしめす．図 2.17 が逐次バックトラック計算法のプログラム，図 2.18 が OR ベクトル計算法のプログラム，図 2.19 が並列バックトラック計算法のプログラムである．これらのうち逐次バックトラック計算法と OR ベクトル計算法のプログラムはすべての部分が Fortran で記述されている．並列バックトラック計算法のプログラムは計算部分を Fortran で記述しバックトラック制御部分を Pascal で記述しているが，これは再帰よびだしを使用したかったからである．

```
************************************************************
* N-QUEEN EXPANDED VER.2 (FOR A SCALAR PROCESSOR)          *
*            BY KANADA,Y.  1984-2-20                       *
************************************************************
      IMPLICIT INTEGER (A-Z)
      PARAMETER (NN=99)
      INTEGER X,Y(0:NN),CNT
      REAL*4  CPTIME
      LOGICAL FAIL
*
      READ*,N
      N1=N-1
      CALL CLOCK(CPTIME, 3)
      CNT=0
      X=0
      Y(0)=0
*     LOOP
    1   CONTINUE
*       FAIL = .NOT. SAFE(X,Y,TAKE,N1)
          FAIL = .FALSE.
          DO 10 XX=0, X-1
            IF (Y(X).EQ.Y(XX) .OR.
     *          X-Y(X).EQ.XX-Y(XX) .OR.
     *          X+Y(X).EQ.XX+Y(XX)) THEN
              FAIL = .TRUE.
              GOTO 15
            END IF
   10     CONTINUE
   15     CONTINUE
*
        IF (.NOT. FAIL) THEN
          IF (X.EQ.N1) THEN
*           CALL SUCCES(Y,CNT,N1)
            CNT = CNT + 1
            FAIL = .TRUE.
*           STOP
          ELSE
            X = X + 1
            Y(X) = 0
          END IF
        END IF
        IF (FAIL) THEN
*         CALL BACKTR(X,Y,N1)
            Y(X) = Y(X) + 1
*           LOOP
   21         IF (Y(X).LE.N1 .OR. X.EQ.0) GOTO 26
              X = X - 1
              Y(X) = Y(X) + 1
              GOTO 21
*           END LOOP
   26       CONTINUE
*
          IF (Y(X).GT.N1) THEN
*                           : FAILED
            WRITE(*, '('' # of solutions ='',I4)') CNT
            CALL CLOCK(CPTIME, 4)
            WRITE(*, '('' CPU time = '',F8.2,'' (ms)'')')
     *        CPTIME*1000
            STOP
          END IF
        END IF
        GOTO 1
*     END LOOP
      END
************************************************************
      SUBROUTINE SUCCES(Y,CNT,NN)
      IMPLICIT INTEGER (A-Z)
      INTEGER NN
      INTEGER Y(0:NN),CNT
*
*     WRITE(*,'X,I5,'':'',(X,40I3)') CNT, Y
      END
```

図 2.17　**エイト・クウィーン問題逐次バックトラック計算法の Fortran コーディング**

測定に使用した並列バックトラック計算法によるプログラムの構造について説明する．とくに，どのような条件制御命令を使用したかをしめす．(これらの命令の代表的な動作を**図** 2.20 にしめす)．並列バックトラック計算法にもとづく 2 つのプログラムの Fortran 部分 (図 2.8 の *nextRow*, *nextCandidate* に対応) は，それぞれ 4 つの部分から構成されている．

```
PROGRAM NQUEEN(INPUT, FT06F001);

CONST ulim = 1023;  llim = 1023;
      NMAX = 12; NMAX1 = 11;
      MAXSOL = 40000; (* << REGION_SIZE DIV (4 * NMAX) *)
      LLIMN = 13000; (* >= NMAX * LLIM *)

TYPE  BOARD  = ARRAY [0..NMAX1] OF INTEGER;
      BOARDS = ARRAY [0..MAXSOL] OF BOARD;
      BOARDWK= ARRAY [0..LLIM, 0..NMAX1] OF INTEGER;
      SAFETY = ARRAY [0..LLIMN, 0..NMAX1] OF BOOLEAN;

VAR   ANSCNT : INTEGER;
      SAFE   : SAFETY;
      Y      : BOARDS;
      YWORK  : BOARDWK;
      N, N1  : INTEGER;
      FT06F001 : TEXT;
      CPUTIME1, CPUTIME2 : REAL;
      Z      : INTEGER;

...

PROCEDURE PRINT(VAR Y : BOARDS;  FIRST, NSOL : INTEGER);
  VAR J, I : INTEGER;
BEGIN
  FOR J := FIRST TO FIRST + NSOL - 1 DO BEGIN
    ANSCNT := ANSCNT + 1;
    WRITE(FT06F001, ANSCNT:5,' : ');
    FOR I := 0 TO N-1 DO BEGIN
      WRITE(FT06F001, Y[J,I]:3);
    END(*LOOP*);
    WRITELN(FT06F001);
  END(*LOOP*);
END(*PRINT*);


PROCEDURE QUEEN(NEXTPOS, FIRST, NSOL, X0 : INTEGER);
  LABEL 99;
  VAR CNT, X, X1, J, I, N1, NREST : INTEGER;
BEGIN
  FOR X := X0 TO N-1 DO BEGIN
    QUEEN1(YWORK, Y, Y[NEXTPOS], SAFE, X, X0, FIRST, NSOL,
           MAXSOL-NEXTPOS, N, NMAX);
    IF NSOL > ULIM THEN BEGIN
      N1 := NSOL DIV LLIM;
      FOR I := 0 TO N1 - 1 DO BEGIN
        QUEEN(NEXTPOS+NSOL, NEXTPOS+I*LLIM, LLIM, X+1);
      END(*LOOP*);
      NREST := NSOL - N1 * LLIM;
      IF NREST <> 0 THEN BEGIN
        QUEEN(NEXTPOS+NSOL, NEXTPOS+NSOL-NREST, NREST, X+1);
      END(*IF*);
      GOTO 99;
    END ELSE BEGIN
      FIRST := NEXTPOS;
    END(*IF*);
  END(*LOOP*);
  ANSCNT := ANSCNT + NSOL;
99:
END(*QUEEN*);


BEGIN
  FINT; { Fortran initialization }
  READ(N); N1 := N - 1;
  REWRITE(FT06F001);
  WRITELN(FT06F001, 'SOLUTIONS OF ', N:1, ' QUEENS PROBLEM');
  WRITELN(FT06F001, '  LLIM = ', LLIM:1, '  ULIM = ', ULIM:1);

  {dummy}
    Z := 0;
    QUEEN1(YWORK, Y, Y[0], SAFE, Z, Z, Z, Z,
           2147483647, N, NMAX);

  ANSCNT := 0;
  CLOCK(CPUTIME1, 3);
  QUEEN(0, 0, 1, 0);
  CLOCK(CPUTIME2, 4);
  WRITELN(FT06F001, 'NUMBER OF SOLUTIONS = ', ANSCNT:1);
  WRITELN(FT06F001, 'CPU TIME =',
          (CPUTIME2 - CPUTIME1) * 1000 :8:2, 'ms');
  FEND; { Fortran finalization }
END.
```

```
      SUBROUTINE QUEEN1(Y, Y0, YTMP, SAFE, X, X0,
     *                  FIRST, NSOL,  MAXSOL, N, NMAX)
      IMPLICIT INTEGER (A-Z)
      INTEGER Y(0:NSOL-1, 0:NMAX-1), Y0(0:NMAX-1, 0:MAXSOL)
      INTEGER YTMP(0:NMAX-1, 0:MAXSOL)
      LOGICAL SAFE(0:NSOL-1, 0:NMAX-1)
      REAL*8  TODMCR, TIME1, TIME2, TIME30, TIME40
*
      IF (X.EQ.X0) THEN
        DO 10 X1 = 0, X-1
          DO 10 J = 0, NSOL-1
   10       Y(J,X1) = Y0(X1,FIRST+J)
      ELSE
        DO 20 X1 = 0, X-1
          DO 20 J = 0, NSOL-1
   20       Y(J,X1) = YTMP(X1,J)
      END IF
      DO 50 I = 0, N-1
        DO 30 J = 0, NSOL-1
          Y(J,X) = I
          SAFE(J,I) = .TRUE.
   30   CONTINUE
        DO 40 X1 = 0, X-1
* VOPTION NOFVAL
          DO 40 J = 0, NSOL-1
            T1A = X1 + Y(J,X1)
            TA  = X  + Y(J,X)
            T1S = X1 - Y(J,X1)
            TS  = X  - Y(J,X)
            IF (Y(J,X1) .EQ. Y(J,X) .OR. T1A .EQ. TA .OR.
     *          T1S .EQ. TS)
     *        SAFE(J,I) = .FALSE.
   40   CONTINUE
   50 CONTINUE
      DO 65 X1 = 0, X-1
        CNT = 0
        DO 60 I = 0, N-1
          DO 60 J = 0, NSOL-1
            IF (SAFE(J,I)) THEN
              YTMP(X1,CNT) = Y(J,X1)
              CNT = CNT + 1
            END IF
   60   CONTINUE
   65 CONTINUE
      CNT = 0
      DO 70 I = 0, N-1
        DO 70 J = 0, NSOL-1
        IF (SAFE(J,I)) THEN
          YTMP(X,CNT) = I
          CNT = CNT + 1
        END IF
   70 CONTINUE
      IF (CNT .GE. MAXSOL-NSOL) THEN
        PRINT*,'TOO MANY SOLUTIONS -- LARGEN "MAXSOL"'
        STOP
      END IF
      NSOL = CNT
      END
```

図 2.19　エイト・クウィーン全解探索並列バックトラック計算法の Pascal / Fortran コーディング例 (S-810 **用**)

```
************************************************************
* N-QUEEN FOR A VECTOR PROCESSOR (VER.3A)                  *
*     BY KANADA,YASUSI   1984-2-10                         *
************************************************************
      PROGRAM NQUEEN
      IMPLICIT INTEGER (A-Z)
      PARAMETER (N=11)
      PARAMETER (MAXSOL=89999)
      INTEGER   Y(0:MAXSOL,0:N-1), YTMP(0:MAXSOL,0:N-1)
      LOGICAL   SAFE(0:MAXSOL)
*
      WRITE(*,'(X,''SOLUTIONS OF'',I3,'' QUEEN PROBLEM'')') N
      NSOL = 1
      DO 80 X = 0, N-1
        DO 10 X1 = 0, X-1
          DO 10 J = 0, NSOL-1
   10       Y(J,X1) = YTMP(J,X1)
        CNT = -1
        DO 70 I = 0, N-1
          DO 30 J = 0, NSOL-1
            Y(J,X) = I
            SAFE(J) = .TRUE.
   30     CONTINUE
          DO 40 X1 = 0, X-1
            DO 40 J = 0, NSOL-1
              IF (   Y(J,X1).EQ.  Y(J,X) .OR.
     *            X1+Y(J,X1).EQ.X+Y(J,X) .OR.
     *            X1-Y(J,X1).EQ.X-Y(J,X)) SAFE(J) = .FALSE.
   40     CONTINUE
          DO 60 J = 0, NSOL-1
            IF (SAFE(J)) THEN
              CNT = CNT + 1
              DO 50 X1 = 0, X
   50           YTMP(CNT,X1) = Y(J,X1)
            END IF
   60     CONTINUE
   70   CONTINUE
        NSOL = CNT + 1
        IF (NSOL.GT.MAXSOL) GOTO 99
        PRINT*,'#SOL =', NSOL
   80 CONTINUE
      CALL PRINT(YTMP,NSOL,N,MAXSOL)
      STOP
   99 PRINT*,'TOO MANY SOLUTIONS -- LARGEN "MAXSOL"'
      END
************************************************************
      SUBROUTINE PRINT(Y,NSOL,N,MAXSOL)
      IMPLICIT INTEGER (A-Z)
      INTEGER Y(0:MAXSOL,0:N-1)
*
      CNT = 1
      DO 10 J = 0, NSOL-1
        WRITE(*,'X,I5,'':'',(X,40I3)') CNT, (Y(J,I),I = 0,N-1)
        CNT = CNT + 1
   10 CONTINUE
      END
```

図 2.18　エイト・クウィーン問題 OR ベクトル計算法の Fortran コーディング例

非 IDP 用のプログラムはつぎのような 4 部分から構成されている．S-810 においては各部分はいずれもベクトル化可能である．各部の説明の最後に図 2.19 における対応部分をしめす.

(1) 前処理部

Pascal 部分からわたされた解候補からなる配列 (図 2.8 の *C* に対応) から，内部処理用の配列への複写をおこなう (この部分はプログラムのかきかたしだいではなくすことが可能である)．第 2，第 3 の部分は，すでにおかれたクウィーンの個数だけの回数くりかえし実行される (図 2.19 では DO 10 〜 20).

(2) 配列初期設定部

この部分では，**(3)** で使用する解の有効性をしめす論理型配列 (マスク・ベクトル) の初期化をおこなう (その値をすべて `.true.` とする) (図 2.19 では DO 30).

(3) 安全性検査部

第 3 の部分は，図 2.8 の *nextRow* に対応する部分であり，解候補の安全性をチェックする．この部分ではマスク演算命令を使用する (図 2.20 a 参照) (図 2.19 では DO 40).

(4) 配列複写・圧縮部

そして第 4 の部分は，図 2.8 の *nextCandidate* に対応する．内部処理用の配列につくられたチェックに合格した解候補 (図 2.8 の $\langle x, b \rangle$ に対応) を，上記の論理型配列の値にしたがって Pascal 部分にわたす配列に圧縮しながら複写する．この部分は S-810 の圧

縮命令を使用することによってベクトル処理できる (図 2.20 c 参照) (図 2.19 では DO 65).

IDP 用のプログラムはつぎのような 4 部分から構成されている．M-680H においては，IDP を使用する部分以外はいずれも IAP によるベクトル処理が可能である．

(1) **前処理部**

Pascal 部分からわたされた解候補からなる配列 (*C*) から，内部処理用のデュアル・ベクトル (IDP 用のデータ形式 [Torii 87b, Kojima 87]) などの配列への複写をおこなう．

第 2，第 3 の部分は，すでにおかれたクウィーンの個数だけの回数くりかえし実行される．

(2) **配列初期設定部**

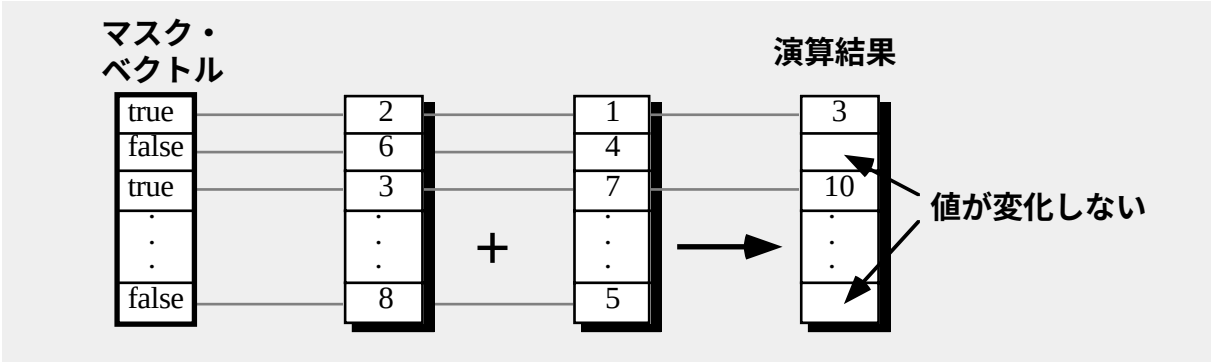
この部分は，**(3)** にそなえて上記のデュアル・ベクトルの一部の初期化をおこなう．

(3) **安全性検査部**

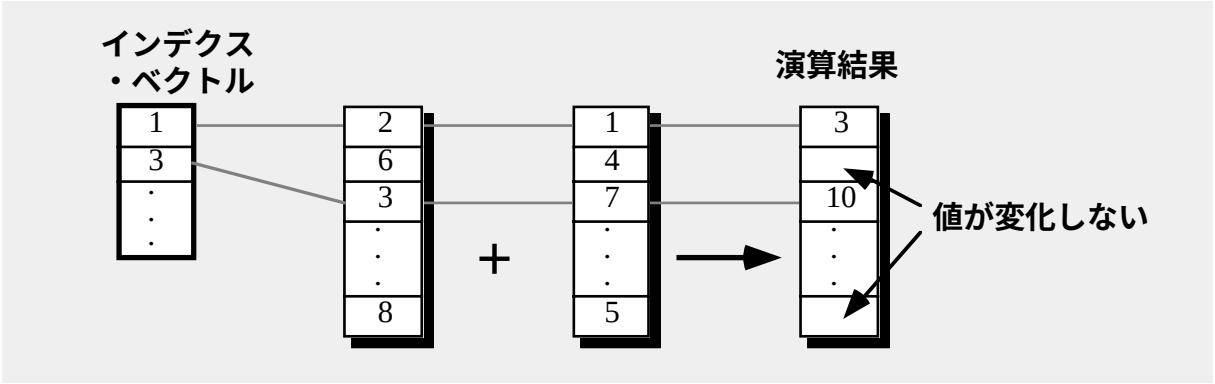
図 2.8 の *nextRow* にほぼ対応する部分である．解候補の安全性をチェックするとともに，解候補をふくむデュアル・ベクトルの圧縮をおこなう．この部分では IAP のリスト・ベクトル命令 (図 2.20 b 参照) と IDP のサーチ命令 [Kojima 87] を使用すればベクトル処理できる．

(4) **配列複写・圧縮部**

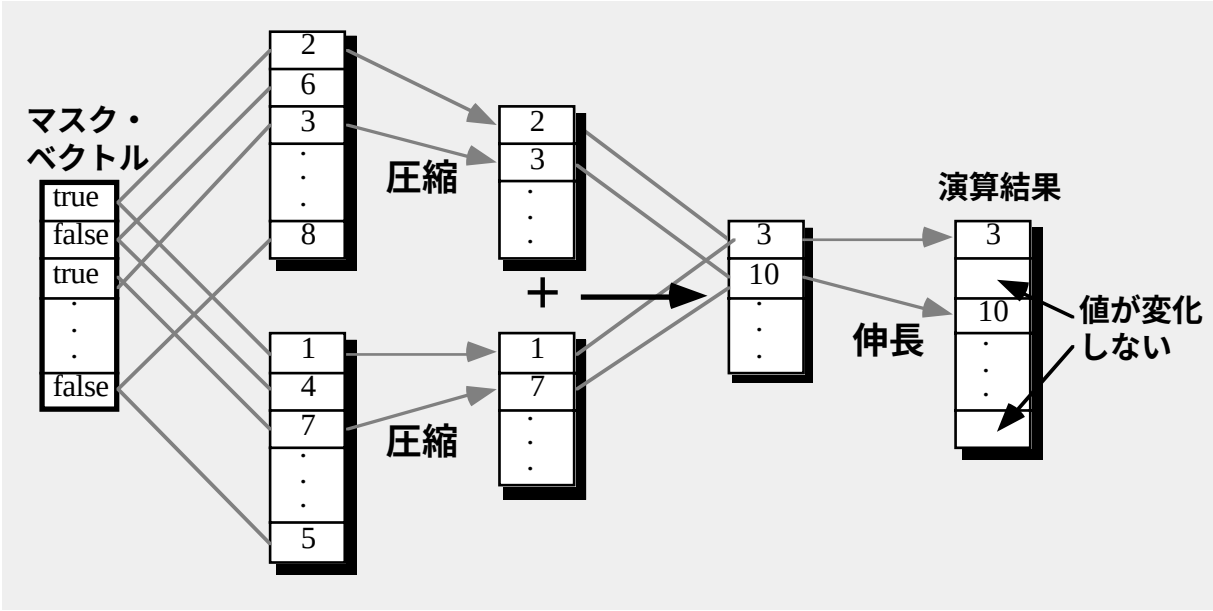
図 2.8 の *nextCandidate* にほぼ対応する部分である．内部処理用の配列につくられたチェックに合格した解候補を，Pascal 部分にわたす配列に圧縮しながら複写する．配列複写・圧縮部ではリスト・ベクトル命令を使用する．この部分の実行前には解候補の配列は圧縮されていないが，それをアクセスするのに，すでに圧縮されているデュアル・ベクトルを使用するので，非 IDP 用のプログラムの **(4)** にくらべるとはるかに高速に実行できる．



(a) **マスク演算方式**



(b) **インデクス方式**



(c) **圧縮方式**

図 2.20　**ベクトル計算機における条件制御方式**

2.9 付録 2：*N* クウィーン解探索の測定データ

表 2.3 に，図 2.9 ～ 2.10 でしめした逐次バックトラック計算法と並列バックトラック計算法による *N* クウィーン問題の全解探索の S-810 における実行時間および加速率を数値でしめす．**表** 2.4 に，図 2.11 ～ 2.12 でしめした M-680H IAP/IDP においておこなった同様の比較結果をしめす．**表** 2.5 には，図 2.14 でしめした *N* クウィーン問題の全解探索における記憶消費量の測定結果を数値としてしめす.

表 2.3　*N* **クウィーン全解探索実行時間** (S-810)

N	**並列** BT CPU **時間** (ms)	**逐次計算法** CPU **時間** (ms)	**加速率**
6	1	5	5.00
7	3	17	5.67
8	**9**	**79**	**8.78**
9	40	386	9.65
10	209	1,944	9.30
11	1,176	10,631	9.04
12	7,315	62,627	8.56

表 2.4　*N* **クウィーン全解探索実行時間** (M-680H IAP/IDP)

N	**スカラ処理**	CPU **時間** (ms)			**加速率**		
		IDP使用	**IAP使用**	**IDP+IAP**	**IDP使用**	**IAP使用**	**IDP+IAP**
4	0.5	0.52	0.52	0.52	0.96	0.96	0.96
5	0.7	0.94	1.15	1.15	0.74	0.61	0.61
6	1.7	2.60	2.81	2.19	0.65	0.60	0.78
7	6.5	8.85	8.44	4.69	0.73	0.77	1.39
8	29.4	40.52	37.35	16.56	0.73	0.79	1.78
9	143.4	208.44	188.09	72.81	0.69	0.76	1.97
10	732.1	1110.31	1011.77	389.48	0.66	0.72	1.88
11	4037.3	6351.35	5838.65	2183.85	0.64	0.69	1.85

また**表** 2.6 には，図 2.15 ～ 2.16 でしめした *N* クウィーン問題の単解探索の S-810 における実行時間および加速率を数値としてしめす.

表 2.5　*N* クウィーン全解探索における記憶消費量 (S-810, **単位: 配列要素数**)

N	l = 2	l = 16	l = 64	l = 256	l = 512	l = 1024
4	7	6	6	6	6	6
5	10	14	14	14	14	14
6	16	79	46	46	46	46
7	20	121	295	164	164	164
8	25	185	509	1009	1062	568
9	33	256	753	1899	3070	3941
10	40	331	1127	3151	5083	8388
11	47	438	1466	4471	8230	13223

表 2.6　*N* クウィーン単解探索の実行時間 (S-810, **単位** ms)

N	l = 8		l = 16		l = 32	
	ベクトル	**スカラ**	**ペクトル**	**スカラ**	**ベクトル**	**スカラ**
6	23	64	16	63	14	74
7	14	43	11	45	11	56
8	182	455	111	415	75	403
9	153	381	96	350	69	340
10	1224	3050	760	2745	492	2577
11	731	1847	428	1637	294	1558
12	6354	16023	3868	14288	2541	13450
13	435	1128	270	999	180	979
14	37567	96840	23032	86155	14915	80723

N	l = 64		l = 128		l = 256	
	ベクトル	**スカラ**	**ペクトル**	**スカラ**	**ベクトル**	**スカラ**
6	16	100	14	120	22	221
7	12	79	16	136	24	264
8	54	408	51	466	47	546
9	50	354	42	394	50	592
10	338	2526	286	2619	251	2632
11	206	1542	159	1589	140	1676
12	1684	13010	1250	12819	1059	13036
13	131	1002	108	1073	101	1280
14	10148	78140	7617	76717	-	-

最後に，AND ベクトル計算法に関する測定結果をしめす．**表** 2.7 に**図** 2.17 のプログラムの S-810 (旧 Fortran コンパイラ) におけるベクトル化率をしめす.

表 2.7　AND **ベクトル計算法のプログラムのベクトル化率**

N	8	9	10	11	12
ベクトル化率	55.6	59.5	63.8	66.7	69.0

S-810/20 旧 Fortran コンパイラ (S-810 初出荷当時) を使用して測定.