ベクトル計算機のための探索問題の計算法「並列バックトラック計算法」†

金田 泰†† 小島啓二†† 菅谷正弘††

この論文では，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，つぎのような性能をえた．エイト・クウィーンの全解探索においては S-810 で逐次処理の約 9 倍，M-680H IAP および IDP を使用して約 1.7 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なった．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．

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，日立 S-810[1]，富士通 VP-200[2]，日電 SX-2[3] 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，マスク演算命令など，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[2]．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，日立 M-180 IAP[4] から M-680 H IAP にいたる一連の計算機である．日電 ACOS 1000 IAP[5]，IBM 3090 VF (Vector Facility)[6] などもこの族のなかにくわえることもできる．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・コンピュータの場合と同様である．

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-680 H IDP (内蔵型データベース・プロセッサ Integrated Database Processor)[7],[8] 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[20)~23)]，論理シミュレーション[9),10)]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

† A Vector-Processor-Oriented Schema for Solving Searching Problems: Parallel Backtracking Schema by YASUSI KANADA, KEIJI KOJIMA and MASAHIRO SUGAYA (Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

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．

この論文では，まず第2章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場合の問題点をしめす．つぎに，第3章ではそ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．3.1 節では OR ベクトル計算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．そして3.2 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 を使用した場合の実行時間を，（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．最後に第5章で結論と今後の課題をしめす．

2. 逐次バックトラック計算法

この章ではまず，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）．そして次に，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，および対角線上にないように配置する問題である．**図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．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[11]．

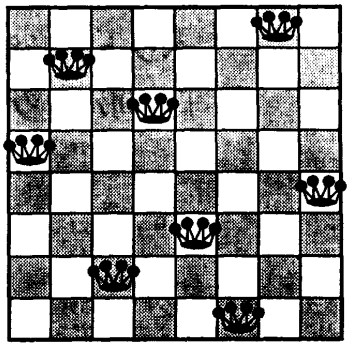

図1 エイト・クウィーン問題の解の一例
Fig. 1 A solution of the eight queens problem.

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[12], Dijkstra[13], Floyd[14] にのべられている．Bitner and Reingold, 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[14]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[11]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

```
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、8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 エイト・クウィーン問題の逐次バックトラック計算法
Fig. 2 Sequential backtracking method of the eight queens problem.

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 $(x_1, x_2, \cdots, x_8)$ $(0 \leqq x_1 \leqq 7)$ 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1の解は (3, 1, 7, 2, 5, 7, 0, 4) とあらわされる．

図2がそのアルゴリズムである．図2のプログラムは $x_1, x_2, \cdots, x_8$ 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，$x_1, x_2, \cdots, x_{i-1}$ 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，$x_i \neq x_j, x_i \neq x_j \pm (i-j)$ $(0 \leqq j < i)$ 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．

こ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．しかし，**図3**にしめすように，すくなくとも S-810 の場合には，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，ベクトルが長がみじかい（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．このプログラムとその実行結果については付録1でよりくわしくのべる．

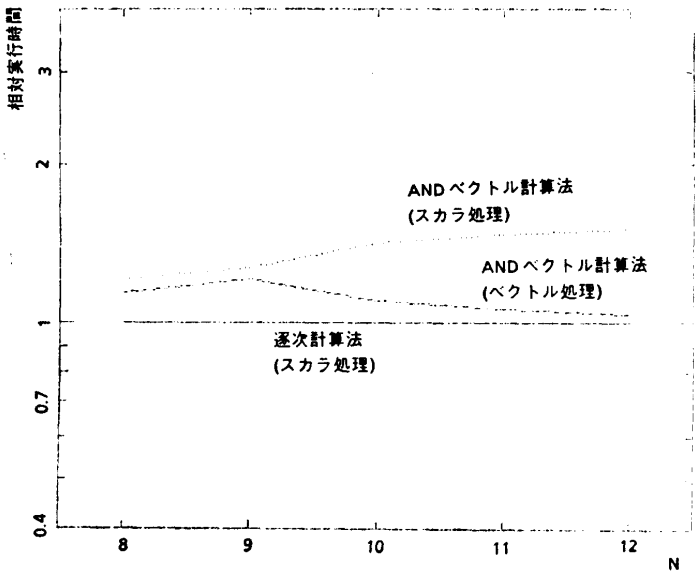

図3 AND ベクトル計算法（関数 TAKEN のベクトル化）による N クウィーンの相対実行時間

Fig. 3 Relative execution time of N queens by AND-vector method (with vectorizing function TAKEN).

3. OR ベクトル計算法（OR 並列計算法）

この章では，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．3.1 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，3.2 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．

3.1 完全 OR ベクトル計算法

第2章でのべたように，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 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．第2章のエイト・クウィーン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☆．

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

☆ Prolog の並行実行法との類比でかんがえれば，この計算法は AND 並列実行法に相当する．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 並列実行法に相当する．したがって，この計算法を OR ベクトル計算法とよぶ．

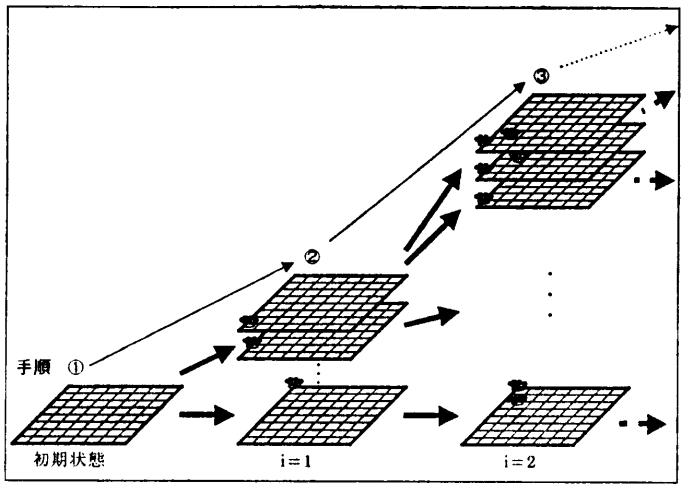

図 4 OR ベクトル計算法による計算過程の概要
Fig. 4 The outline of OR-vector computation process.

```
C := {emptyChessBoard};
        -- Cを1個のチェス・ボードからなる集合とする。
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をCとする。
end loop;
PRINT(C); -- 解を印刷する。
```

図 5 エイト・クウィーン問題の OR ベクトル計算法
Fig. 5 The OR-vector method of eight queens problem.

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（ベクトル）の各要素に対して並列に計算をすすめる．エイト・クウィーンの場合には，解候補はクウィーンがのせられたチェス・ボードである．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 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 5 にアルゴリズムの概要をしめす．図 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．

図 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，関数は 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 $(0 \leqq x \leqq 7)$ とチェス・ボード b $(\in C)$ との対 $\langle x, b \rangle$ 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 $\langle x, b \rangle$ 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．

第 2 世代のベクトル計算機のおおくは，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ベクトル圧縮命令をもっている．上記の機能はこの命令をつかって高速に実行することができる．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[7]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) は上記の対 $\langle x, b \rangle$ 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．こ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．

つぎに，OR ベクトル計算法の性能についてのべる．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を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場合には，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 クウィーン問題の場合，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 以下で計算不能になってしまう．また，計算可能な場合でも，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つ解をもとめる場合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，単解探索の場合にはむだな計算がおおくなる．

3.2 並列バックトラック計算法

3.1 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 6 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 7 にしめす．図 7 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 7 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．

並列バックトラック計算法においては，3.1 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場合には，N に関して指数

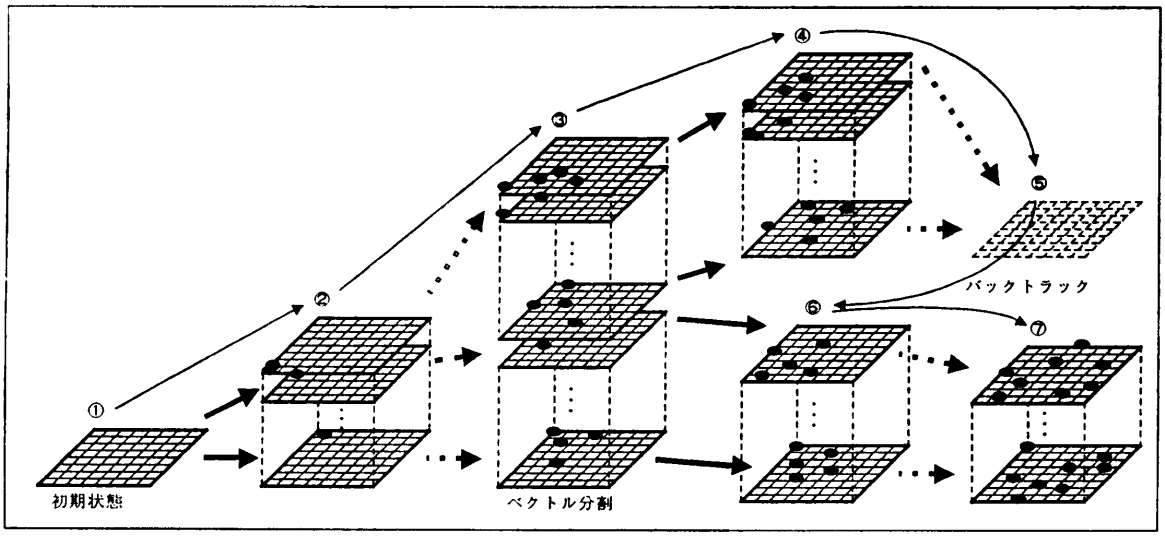

図 6 並列バックトラック計算法による計算過程の概要
Fig. 6 The outline of parallel backtracking computation process.

```
C := {emptyChessBoard}; -- Bを空のチェス・ボードとする。
QUEEN(C, 1);
   -- 手続き QUEENをよび、Nクウィーン問題の解をもとめて印刷する。
procedure QUEEN(C : setOfChessBoards; x : row) is
   if x > N 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, x));
         -- Cがふくむ各チェス・ボード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 7 N クウィーン問題の並列バックトラック計算法
Fig. 7 Parallel backtracking method of the N queens problem.

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，全解探索の場合，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場合にくらべて 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．単解探索の場合は，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．

4. N クウィーンの実行結果

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 7 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．これらのプログラムについては付録 2 でさらに説明する．

図 8 に N クウィーンの全解探索の実行時間をしめす．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・クウィーンの場合には実行時間は 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（金田[15]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）．M-680H IAP, IDP をともに使用した場合においては，並列バックトラック計算法のほうが $N \geqq 7$ のとき逐次バックトラック計算法より高速である．エイト・クウィーンの場合には実行時間は 9.7 ms であり，

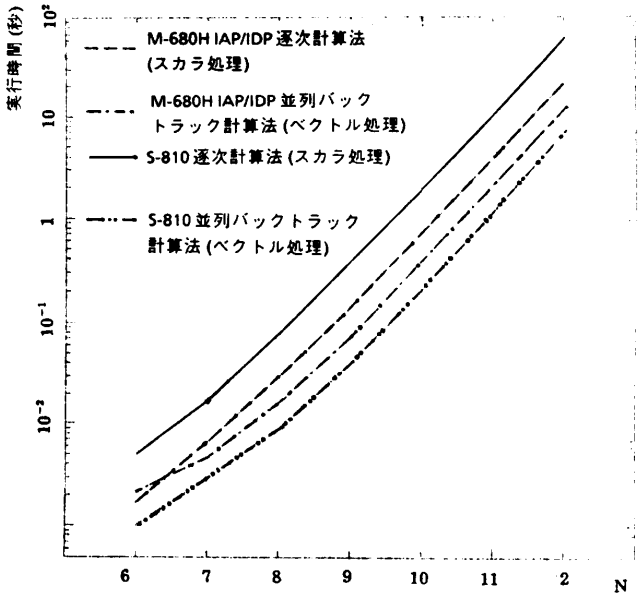

図 8 Nクウィーン全解探索実行時間
Fig. 8 Execution time of eight queens exhaustive search.

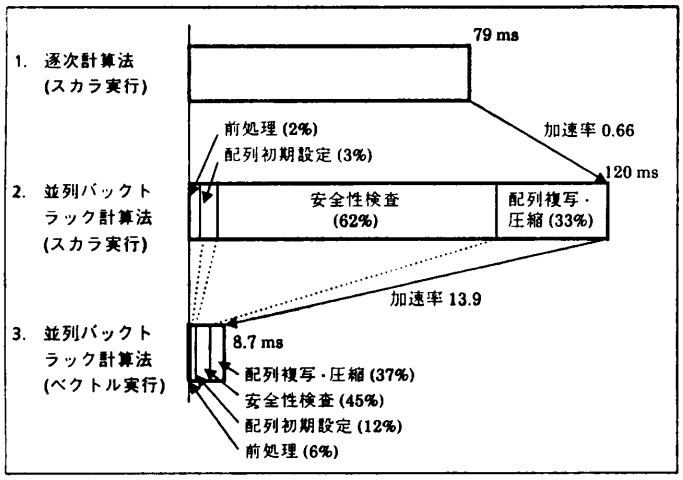

図 9 各種実行法によるエイト・クウィーン全解探索の実行時間
Fig. 9 Execution time of eight queens exhaustive search on S-810 by various methods.

逐次バックトラック計算法の約 1.7 倍の速度である (S-810 と M-680H とで実行時間にそれほど差がないのは，計算機の世代のちがいによる)．

図 9 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 9 にあわせてしめした．一般にプログラムを並列実行する場合には，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 クウィーンの場合には比較的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る理由である．

図 10 に S-810 による N クウィーンの単解探索の実行時間をしめす．$N<14$ で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，$N\geqq 14$ では，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．図 10 から，つぎのようなことがいえる．N クウィーン ($N\geqq 14$) 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．

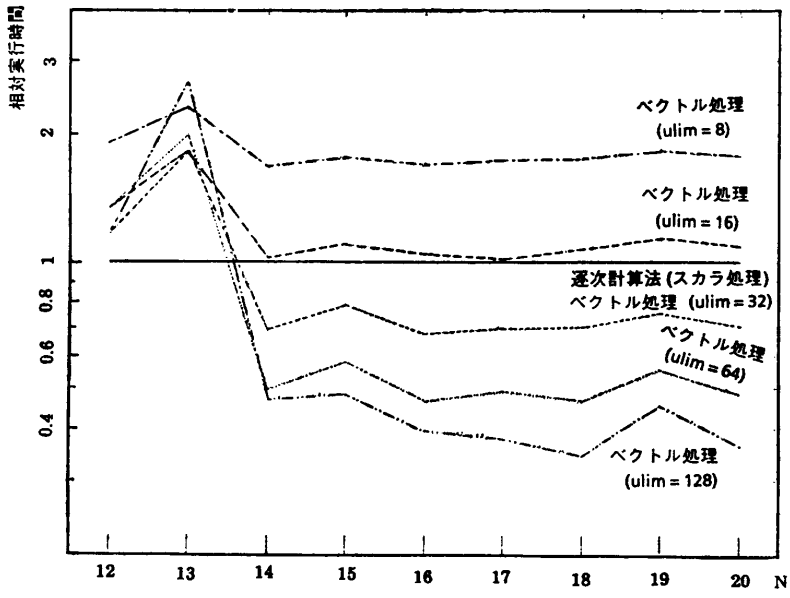

図 10 S-810 による N クウィーン単解探索の相対実行時間
Fig. 10 Relative execution time of N queens single solution search on S-810.

5. むすび

この論文では，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その高速性をたしかめた．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今後の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 のような論理的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．

また，逆に論理型言語の OR 並列実行方式[16]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．この論文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．しかし第1章でものべたように，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．われわれは，その方向の研究をすでに開始している[17)-19)]．

謝辞 この研究を支援していただいた現日立製作所ソフトウェア工場 AI 応用プログラム部の高橋栄部長，同中央研究所第8部の吉住誠一主任研究員はかの方々に感謝します．

参考文献

1) 小高俊彦，小林二三幸，河辺 峻，長島重夫：最大性能が 630 MFLOPS で 1G バイトの半導体拡張記憶が付くスーパコンピュータ HITAC S-810，日経エレクトロニクス，1983. 4. 11, pp. 159-184.
2) 平栗俊男，田畑 晃，樋本隆光，田中尚三：マシン・サイクル 7.5 ns を達成した並列パイプライン処理方式のスーパコンピュータ FACOM VP，日経エレクトロニクス，1983. 4. 11, pp. 131-155.
3) 古勝紀誠，渡辺 貞，近藤良三：最大性能 1.3 GFLOPS，マシン・サイクル 6 ns のスーパコンピュータ SX システム，日経エレクトロニクス，1984. 11. 19, No. 356, pp. 237-272.
4) 堀越 彌，梅谷征雄：汎用計算機のための内蔵ベクトル演算方式，情報処理学会論文誌，Vol. 24, No. 2, pp. 191-199 (1983).
5) 大阪大学計算センタ・ニュース，Vol. 12, No. 1, pp. 59-72 (1982).
6) Buchnolz, W.: The IBM System/370 Vector Architecture, *IBM Syst. J.*, Vol. 25, No. 1, pp. 51-62 (1986).
7) 鳥居俊一，小島啓二，吉住誠一，河辺 峻，高橋政美，久代康雄：リレーショナル・データベースの処理速度向上を図る CPU 内蔵型データベース・プロセサ，日経エレクトロニクス，1987. 2. 9, No. 414, pp. 185-210.
8) Kojima, K., Torii, S. and Yoshizumi, S.: IDP—A Main Storage Based Vector Database Processor, *International Warkshop on Database Machines*, pp. 60-73 (1987).
9) Nagashima, S., Nakagawa, T., Omota, K., Miyamoto, S., Kawabe, S. and Tsuchiya, Y.: Hardware Implementation of VELVET on the Hitachi S-810 Computer, *IEEE International Conference on Computer-Aided Design*, pp. 390-393 (1986).
10) 石浦菜岐佐，安浦寛人，矢島脩三：ベクトル計算機による高速論理シミュレーション，情報処理学会論文誌，Vol. 27, No. 5, pp. 510-517 (1986).
11) 奥乃 博：第3回 Lisp コンテストおよび第1回 Prolog コンテストの課題案，情報処理学会記号処理研究会資料，28-4 (1984).
12) Bitner, J. R. and Reingold, E. M.: Backtrack Programming Techniques, *Comm. ACM*, Vol. 18, No. 11, pp. 651-656 (1975).
13) Dijkstra, E. W.: Notes on Structured Programming, in Dahl, O.-J., Dijkstra, E. and Hoare, C. A. R., *Structured Programming*, Academic Press, London (1972), 訳書：野下浩平他訳：構造化プログラミング，サイエンス社.
14) Floyd, R.: Nondeterministic Algorithms, *J. ACM*, Vol. 14, pp. 636-644 (1967).
15) 金田 泰：N クウィン問題のベクトル計算機むき解法—並列バックトラック解法—，第29回情報処理学会全国大会論文集，5N-3, pp. 1251-1252 (1984).
16) Conery, J. S. and Killer, D. F.: Parallel Interpretation of Logic Programs, *Proc. ACM 1981 Conference on Functional Programming Languages and Computer Architecture*, pp. 163-170 (1981).
17) 金田 泰：スーパー・コンピュータによる Prolog の高速実行，第 26 回プログラミング・シンポジウム報告集，pp. 47-56 (1985).
18) 金田 泰：ベクトル計算機による論理型言語の高速実行をめざして—各種 OR ベクトル実行方式の実現と検討—，情報処理学会プログラミング言語研究会，PL-87-12 (1987. 6).
19) Kanada, Y., Kojima, K. and Sugaya, M.: Vectorization Techniques for Prolog, *Proc. ACM*

1988 International Conference on Supercomputing (1988).
20) Stone, H. S.: Sorting on STAR, *IEEE Trans. Softw. Eng.*, Vol. 4, No. 2, pp. 138-146 (1978).
21) Brock, H. K.: Brooks, B. J. and Sullivan, F.: DIAMOND: A Sorting Method for Vector Machines, *BIT*, Vol. 21, pp. 142-152 (1981).
22) Roensch, W. and Strauss, H.: Timing Results of Some Internal Sorting Algorithms on Vector Computers, *Parallel Computing*, Vol. 4, pp. 49-61 (1987).
23) 石浦菜岐佐，高木直史，矢島脩三：ベクトル計算機上でのソーティング，情報処理学会論文誌，Vol. 29, No. 4, pp. 378-385 (1988).

付録 1 AND ベクトル計算法と実行結果

本文で説明したように，逐次バックトラック計算法にもとづくエイト・クウィーンのプログラムは図2のようになる．こ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_i とが条件 $x_i \neq x_i, x_i \neq x_i \pm (i-j)$ 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・クウィーンの場合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3にしめしたように，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．

付録 2 測定用プログラムの説明

測定に使用した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 11** 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7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．

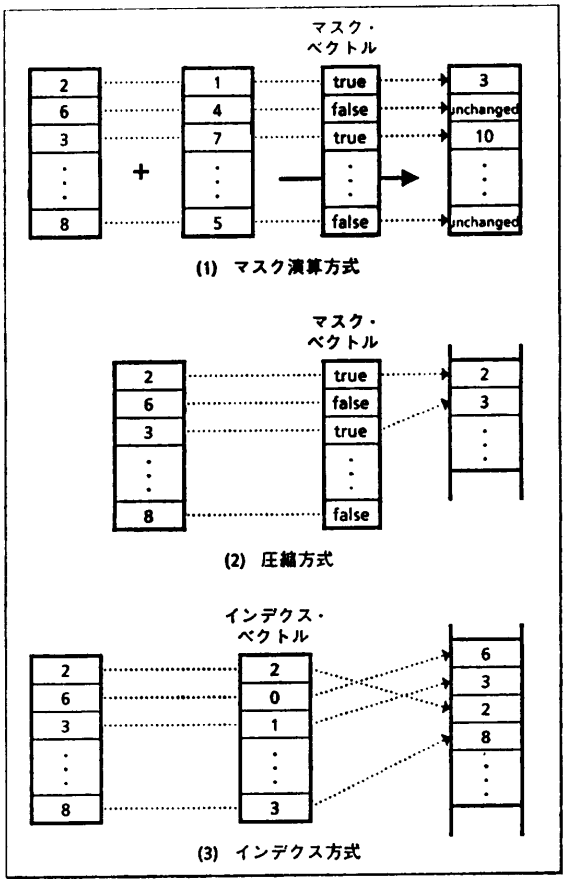

図 11 ベクトル計算機における条件制御方式
Fig. 11 Conditional control methods of vector processors.

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．

（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7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．

（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）．

（3） 安全性検査部

第3の部分は，図7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 11(1)参照）．

(4) 配列複写・圧縮部

そして第4の部分は，図7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7の〈x b〉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 11(2)参照）．

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[7),8)]）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7の nextRow に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 11(3)参照）と IDP のサーチ命令[8)]を使用すればベクトル処理できる．

(4) 配列複写・圧縮部

図7の nextCandidate に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．

(昭和62年12月14日受付)
(昭和63年 9 月 5 日採録)

金田 泰（正会員）

1956 年生．1979 年東京大学工学部計数工学科卒業．1981 年同大学院情報工学専門課程修了．同年4月から(株)日立製作所中央研究所第8部．入社後 Fortran コンパイラ開発に従事し，現在はスーパ・コンピュータの論理型言語処理系の研究に従事．1985 年山内奨励賞受賞．プログラミング言語とその処理系に興味をもつ．ACM，ソフトウェア科学会各会員．

小島 啓二（正会員）

昭和31年生．昭和55年京都大学理学部卒業．昭和57年京都大学大学院理学研究科修士課程修了．同年(株)日立製作所中央研究所に入社し，データベースマシンの研究開発に従事，現在に至る．ソフトウェア科学会会員．

菅谷 正弘（正会員）

1958年生．1982年電気通信大学電気通信学部機械工学科卒業．1984年同大学大学院電気通信学研究科機械工学専攻修士課程修了．同年4月(株)日立製作所入社．中央研究所第8部勤務．入社後，並列推論マシンの研究に従事．現在，スーパ・コンピュータの論理型言語処理系の研究に従事．