OR 並列実行のための論理型言語プログラムのベクトル化法[†]

金田　泰[††]　菅谷正弘[††]

ベクトル計算機（スーパ・コンピュータ）を使用して論理型言語プログラムを並行処理するための一種のプログラム変換法（ベクトル化法）を開発した．このベクトル化法は，OR 並列性があり，引数の入出力モードが確定しているプログラムを対象とする．この方法では，原始プログラムの論理変数ごとに，それが探索木上のことなる位置でとる複数の値を各要素とするベクトルをつくり，それをベクトル処理するプログラムを生成する．この方法を N クウィーン問題のプログラムに適用して自動変換し，生成されたプログラムがただしく動作することを確認するとともに，ベクトル計算機 S-810 において 2.6 MLIPS という高い実行速度をえた．

1. はじめに

Prolog で代表される論理型言語は，ユニフィケーションと自動バックトラックという２つの機能をそなえている．ユニフィケーションは一種のパタン・マッチングであり，それによってリスト処理などのプログラムを容易に記述することを可能にしている．また，自動バックトラックは，解探索のプログラムを非常に容易に記述することを可能にしている．これらの機能によって論理型言語は，リスト処理，言語処理，知識ベース処理などの記号処理に非常に適したプログラミング言語となっている．したがって，今後，論理型言語の応用がひろがっていくとかんがえられる．

論理型言語の応用がひろがれば，その高速処理の必要性も増加してくる．逐次計算機上での高速化をめざす研究[1),2)]も重要だが，飛躍的な高速化のためには並行処理が不可欠である．論理型言語プログラムの並列処理法はつぎの２つに大分類される．

（1） OR 並列処理

複数の解（部分解）を並列に計算する．

（2） AND 並列処理

１つの解をもとめる計算の部分計算どうしを並列に計算する．

論理型言語プログラムの高速処理の方法は，また，使用するハードウェアによって，MIMD 型並列計算機を使用する方法と，S-820, Cray-2 などのベクトル計算機（スーパ・コンピュータ）や SIMD 型並列計算機を使用する方法とにわけることができる．以下，ベクトル計算機と SIMD 型並列計算機とをあわせて SIMD 型計算機とよぶ．MIMD 型並列計算機による高速処理をめざした研究としては ICOT の PIM に関する研究[3)]などがあり，SIMD 型計算機による高速処理をめざした研究としては Nilsson ら[4),5)]，辰口ら[6)]などの研究がある．

上記の MIMD 型並列計算機や Nilsson らによる並行処理法は，柔軟性がたかいという利点がある．しかし，前者は実行時に負荷分散などのためのオーバヘッドがおおきい．また，後者は SIMD 型計算機の特性をうまくいかせず，むだがおおい．すなわち，SIMD 型計算機は複数のデータに同種の演算を適用することすなわちベクトル演算を得意としているが，Nilsson らの実行方式では異種の演算をまとめて処理しようとしている．SIMD 型計算機の特性をいかすには，極力，ベクトル演算が適用できるようにコンパイル時にプログラムを変換しておくのがよいとかんがえられる．

ベクトル計算機の Fortran コンパイラは，ベクトル化とよばれる一種のプログラム変換によって，ベクトル計算機による実行を可能にしている．すなわち，DO ループ内の配列計算をベクトル演算に変換し，ベクトル演算命令を生成している．しかし，論理型言語プログラムには DO ループもないし配列もあらわれない．したがって，Fortran コンパイラにおけるベクトル化技術は論理型言語プログラムには適用できない[7)]．

この論文では，コンパイル時のプログラム変換にもとづいてベクトル計算機における論理型言語の実行を可能にするための方法をしめす．第２章ではそのプログラム変換の方針をしめす．第３章および第４章で

---

† Vectorization Techniques for OR-Parallel Execution of Logic Programming Languages by YASUSI KANADA and MASAHIRO SUGAYA (Central Research Laboratory, Hitachi Ltd.).

†† (株)日立製作所中央研究所

は，その方針にもとづいて論理型言語の決定性手続きおよび非決定性手続きを変換する方法をしめす．第5章では，第3～4章の方法を N クウィーン問題のプログラムに適用して，ベクトル計算機 S-810 において実測した結果をしめす．最後に，第6章で結論をのべる．

2. OR ベクトル計算法

この論文でしめす論理型言語プログラムのベクトル処理方法は，OR ベクトル計算法[8]にもとづいている．この方法では，論理型言語プログラムをつぎのような手順で翻訳・実行する．

(1) ベクトル化

論理型言語で記述されたプログラムを，OR ベクトル計算法によって計算するプログラムに変換する．このプログラム変換をベクトル化とよぶ．ベクトル化は，くりかえし構造の交換[7]，くりかえし構造の一重化[7]という2つの変換戦略にもとづいている．変換結果のプログラムは，ベクトルがあつかえる言語で表現する必要がある．この論文では，Plolog にくみこみ手続きを追加してベクトルがあつかえるようにした論理型言語を使用する．この言語を IL (Intermediate Language) とよぶ．

(2) 実　　行

OR ベクトル計算法にもとづいて実行する．ただし，(1)で生成されるのが IL のような高水準言語のプログラムの場合には，コンパイルしてから実行する．

この章では OR ベクトル計算法についてかんたんに説明するとともに，OR ベクトル計算法による計算過程と論理型言語プログラムとのおよその対応づけをしめす．詳細な対応は第3～4章でしめす．

OR ベクトル計算法は，8クウィーン問題で代表される探索問題において，複数の解候補を要素とするベクトルをつくり，それに対する計算をベクトル処理によっておこなう計算法である．4クウィーン問題の全解探索を例として，OR ベクトル計算法を逐次計算法と比較しながら説明する．4クウィーン問題を逐次計算機で実行する場合には，バックトラック計算法にもとづいて実行する．すなわち，探索の進行にしたが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バックトラック（あともどり）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，全解候補を1つのベクトルに蓄積して，その各要素に対して並行処理をおこなう．4クウィーンの場合には，各解候補はクウィーンがのせられたチェス盤である．

図1に，論理型言語による4クウィーンのプログラムをしめす（中島[9]の8クウィーンのプログラムを一部かきかえたものである）．OR ベクトル計算法による4クウィーンの実行過程で生成されるベクトルとその内容を図2にしめす．各ベクトルはリストを要素としてふくんでいる．図2には，リストを Prolog の記法とチェス盤のイメージの両方でしめしている．また，図1のプログラムと対応づけるために，実行される手続き名を記述している．ただし，図1のプログラムを OR ベクトル計算法で実行するためには，プログラム変換をへる必要があるから，この対応は正確ではない．

手続き put は再帰よびだしをふくめて5回くりかえしてよびだされる．put を構成する2個の節のうち，つねに1個だけが実行される．最初の4回では第2節が実行され，手続き select と not_take がよびだされる．最後のよびだしでは第1節が実行される．

たとえばステップ②においては，つぎのような4個の同一の手続きのよびだしを並列処理するのと等価な処理がベクトル処理によっておこなわれる．

?-put ([2, 3, 4], [1], Q_1).
?-put ([1, 3, 4], [2], Q_2).
?-put ([1, 2, 4], [3], Q_3).
?-put ([1, 2, 3], [4], Q_4).

OR ベクトル計算法においては，このように，ことなる解候補に関する同一の手続きの実行を複数回まとめてベクトル処理する．論理変数や引数ごとに，それらがとりうる複数の値を各要素とするベクトルをつく

```
?- put([1, 2, 3, 4], [], Q).                    ----4クウィーン問題をとく.
put([], B, B).
put(Qs, B, Q) :-
    select(Qs, Q1, R), not_take(B, Q1), put(R, [Q1 | B], Q).
          ----putが再帰よびだしされるたびに1個のクウィーンが盤面におかれる.
select([A | L], A, L).                          ----クウィーンを1個えらぶ手続き.
select([A | L], X, [A | L1]) :- select(L, X, L1).
not_take(R, Q) :-
    Qa is Q+1, Qs is Q-1, not_take1(R, Qa, Qs).
      ----not_takeは、えらんだクウィーンがすでに盤面においたクウィーンでとられ
      ----ないかどうかをしらべる手続き.
not_take1([], Qa, Qs).
not_take1([Q | R], Qa, Qs) :-
    Q =\= Qa, Q =\= Qs,
    Qaa is Qa+1, Qss is Qs-1, not_take1(Q, Qaa, Qss).
                          ----not_take1はnot_takeのしたうけの手続き.
```

図1　4クウィーン問題の Prolog プログラム
Fig. 1　A Prolog program of the four-queens problem.

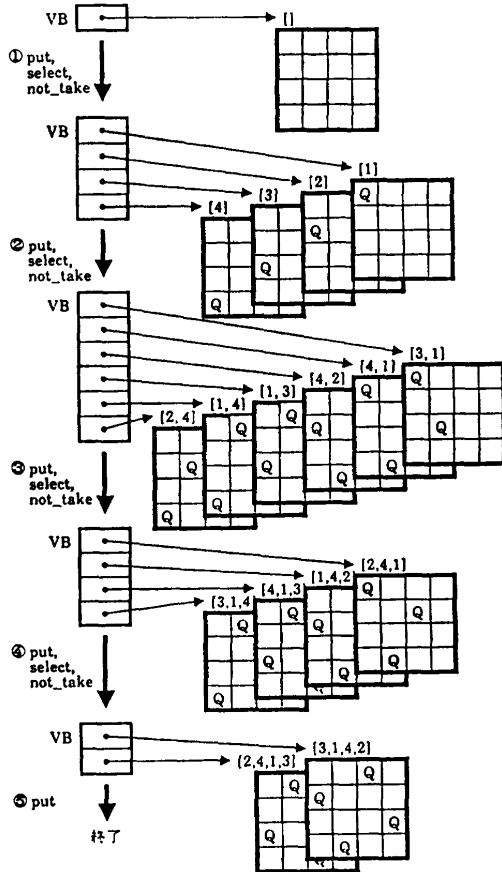


図 2 OR ベクトル計算法による 4 クウィーンの実行過程
Fig. 2 The execution process 4-Queens by OR vector method.

り，それをベクトル処理する．図2にしめしたデータは，手続き put の第2引数に対応するベクトルとその内容である．他の引数や変数に対応するベクトルの値は図にはしめしていないが，第2引数と同様に，図1のプログラムの実行における引数や変数とひとしい値を要素とするベクトルが生成・使用される．図2のデータの処理にはリストを要素とするベクトルの処理が必要になるが，その方法については金田ら[7]でのべている．

3. 決定性手続きのベクトル化法

論理型言語の手続きは決定性手続きと非決定性手続きとに分類することができる．この章では決定性手続きを定義し，そのベクトル化法についてのべる．

3.1 決定性手続きの定義

つぎのような m 個の節から構成される手続き P のベクトル化についてかんがえる．

$P1$:- $S11$, …, $S1s_1$, $U11$, …, $U1u_1$.

…

Pm :- $Sm1$, …, Sms_m, $Um1$, …, Umu_m.

ここで，各 Sij はただ1つしか解をもたないくみこみ手続きのよびだしであり，各 Ui1 は第 i 節の最初のユーザ定義手続きのよびだしだとする．また，各 Uij ($j>1$) はくみこみ手続き，ユーザ定義手続きのうちのいずれかのよびだしである．

手続き P のすべてのよびだしに対してつぎの条件がともになりたつとき，手続き P を**決定性手続き**とよぶ．

(1) そのよびだしにおいて，頭部 Pi とのユニフィケーションが成功し，かつ Si1, …, Sis_i のすべてが成功するような節が1個または0個しかない．

(2) U11, …, $U1u_1$, …, Umu_m は決定性手続きのよびだしまたは手続き P の再帰よびだしである．

決定性手続き以外の手続きを**非決定性手続き**とよぶ．

N クウィーン問題のプログラムにおける手続き not_take および not_take 1 は決定性手続きである．なぜなら，第1に，N クウィーンのプログラムのなかでは，これらの手続きには第1引数に基底項 ground term) いいかえれば定数がわたされるため，いずれか一方の頭部としかユニフィケーションが成功しないし，第2にこれらの節の本体には not_take 1 以外の手続きよびだしは存在しないからである．

この章では決定性手続きだけをあつかう．

3.2 手続きよびだしのベクトル化

第2章でのべたように，決定性手続きをベクトル化してえられるプログラムにおいては，もとの手続きにおける複数のよびだしをまとめてベクトル処理する．そこで，N クウィーンのプログラムを構成する手続き not_take 1 をベクトル化した手続きを v_not_take 1 とし，つぎの2個の手続きよびだしに対応する v_not_take 1 のよびだしのベクトル化をかんがえる．

?-not_take 1 ([2, 4, 1], 4, 2). (3.2.1)

?-not_take 1 ([4, 1, 3], 3, 1). (3.2.2)

これらの手続きよびだしを実行すると，(3.2.1)は失敗し，(3.2.2)は成功する．

以下，$e_1, \cdots, e_n$ を要素とするベクトルを $\#(e_1, \cdots, e_n)$ とあらわす．よびだし(3.2.1)および(3.2.2)をベクトル化した結果は，IL でつぎのように表現することができる．

?-v_not_take 1(#([2, 4, 1], [4, 1, 3]),
#(4, 3), #(2, 1),

#(true, true), M0). (3.2.3)

(3.2.3)における第1〜3引数は，それぞれ(3.2.1)〜(3.2.2)における第1〜3引数とひとしい値を要素とするベクトルである．第4引数は v_not_take 1 の実行開始時の第1〜3引数の有効性をしめす論理型ベクトルである．また，第5引数は実行終了時の第1〜3引数の有効性をしめす論理型ベクトルである．このような，ベクトルの各要素の有効性をしめす論理型ベクトルを**マスク・ベクトル**という[7]．実行の結果，(3.2.1)が失敗するのと対応して，各ベクトルの第1要素は無効になる．また，(3.2.2)が成功するのと対応して，各ベクトルの第2要素は有効なままである．したがって，M0 の値はつぎのようになる．

M0=#(false, true).

このようにベクトル要素の有効性をしめすのにマスク・ベクトルをつかうベクトル処理方式を**マスク演算方式**とよぶ．

例をつかって説明してきたが，他の手続きでもベクトル化の方法はかわらない．ただし，つねに成功する手続きの場合は，追加する引数は1個でよく，その手続きのよびだしの前後の手続きよびだしで同一のマスク・ベクトルを使用すればよい．not_take 1 のように失敗することがある手続きの場合は，2個の引数を追加して，そのよびだしのまえではそのうちの第1の引数をマスク・ベクトルとして使用し，よびだしのあとでは第2の引数をマスク・ベクトルとして使用する．

引数の有効性をあらわす方法としては，ほかに，有効な要素のインデクスをふくむベクトルを使用する方法がある．このベクトルは**インデクス・ベクトル**とよばれる[7]．インデクス・ベクトルをつかって上記の手続きよびだしを IL で表現すると，つぎのようになる．

?-v_not_take 1 (#([2, 4, 1], [4, 1, 3]),
#(4, 3), #(2, 1),
#(1, 2), X0). (3.2.4)

第4引数は実行開始時のインデクス・ベクトルである．第5引数は実行終了時のインデクス・ベクトルであり，つぎのような1要素のベクトルになる．

X0=#(2).

このインデスク・ベクトルは，各ベクトルの有効な要素が第2要素だけであることをしめしている．インデクス・ベクトルをつかう方式を**インデクス方式**とよぶ．

また，有効な要素だけを入出力する方式を**圧縮方式**とよぶ．圧縮方式では4.1節でのべるようなベクトル要素の対応づけが必要になる．

3.3 手続き定義のベクトル化

この節では，前節でしめしたようなベクトル化された手続きよびだしでよびだされるべき手続き定義のベクトル化法をしめす．

この節でのべるベクトル化手順を適用するためには，対象となる手続きPはつぎの2つの条件をともにみたさなければならない．

条件1　Pは決定性手続きである．

条件2　Pの定義は‘!’(カット)，‘\+’(not) などの論理性をくずす手続きよびだしや，‘;’(すなわち‘or’) をふくまない．

これらの条件のうち条件1がなりたたない手続きの一部は第4章でのべる方法でベクトル化すればよい．条件2については，手続き‘;’は手順をかんたんにするためにのぞいたのであり，これをふくんでよいように手順を拡張することは可能である．‘!’，‘\+’などについては，適当な仮定をおけば拡張可能だとかんがえられるが，未検討である．

決定性手続きのベクトル化は，およそつぎのような手順でおこなうことができる．

ステップ1　決定性判定

ベクトル化すべき手続きが決定性手続きかどうかを，3.1節でしめした定義にしたがって判定する．決定性手続きと判定されたときだけ，以下のステップを実行することができる．

ステップ2　標準化

ベクトル化を容易にするため，プログラムを標準形に変換する．標準形については後述する．

ステップ3　本変換（ベクトル化）

各手続き名をベクトル化された手続きの名称で置換するとともに，マスク・ベクトル，インデクス・ベクトルなどの引数を追加する．

これらのステップについて，さらにくわしく説明する．

まず，決定性判定についてのべる．決定性判定の結果はしばしば引数のモード，すなわち手続きの引数が入力であるか出力であるかに依存する．したがって，引数のモードをしらべる必要がある．この解析をモード解析という．自動モード解析は上田[10]などでのべられているのと同様の方法でおこなえばよい．しかし，自動解析だけでは十分でない場合があるので，ユーザ・オプションというかたちでプログラマから情報をうけとる手段をもうける必要があるとかんがえられ

る．たとえば，つぎのようなモード宣言を手続きの直前に記述する方法が，自然でのぞましいとかんがえられる．

mode not_take 1 (+, +, +).　　(3.3.1)

このモード宣言は，not_take の引数がすべて入力モードであることをしめしている．

つぎに，標準化について説明する．標準化によって，ベクトル化対象の手続きを構成する各節はつぎのような形式の**標準形**に変換される．

P(X 1, …, Xn) :- I 1, …, Ik,
　Q 1, …, Ql, Ol, …, Om.　　(3.3.2)

ここで X 1, …, Xn はすべてことなる論理変数である．また，I1, …, Ik および O1, …, Om はこれらの変数と変換前の節の仮引数とのユニフィケーションをおこなう手続きよびだしである．Q 1, …, Ql は変換前の節の本体の手続きよびだしである．効率上は，出力モードの仮引数とのユニフィケーションは変換前の本体よりあとにおき，それ以外の仮引数とのユニフィケーションは変換前の本体よりまえにおくのがのぞましい．しかし，モードが不明のときはそれらすべてを変換前の本体のまえにおけばよい．

本体に関しても，つぎのような変換をおこなう．手続き is のよびだしを手続き '+', '−' などの算術演算をおこなう手続きよびだしの列に置換する．

例を2つあげる．

例1　手続き not_take 1 の標準形

各節の標準形はつぎのとおりである．

not_take 1(T 1, Qa, Qs) :- T 1=[].　　(3.3.3)
not_take 1(T 1, Qa, Qs) :-
　T 1=[Q|R], Q=\=Qa, Q=\=Qs,
　'+'(Qa, 1, Qaa), '−'(Qs, 1, Qss),
　not_take 1(R, Qaa, Qss).　　(3.3.4)

各節の第1仮引数は論理変数ではなかったため，T 1 で置換されている．not_take 1 の引数はすべて入力モードであるから，これらの引数に対するユニフィケーションは節の先頭におかれている．

例2　手続き append の標準形

リストを連接する手続き append のモード宣言つき原始プログラムをしめす．

mode append (+, +, −).　　(3.3.5)
append ([], X, X).　　(3.3.6)
append([A|D], Y, [A|D 1]) :-
　append(D, Y, D 1).　　(3.3.7)

標準形はつぎのとおりである．

append(T 1, X, T 2) :- T 1=[], T 2=X.
　　(3.3.8)
append(T 1, Y, T 2) :- T 1=[A|D],
　append(D, Y, D 1), T 2=[A|D 1].
　　(3.3.9)

標準形第1節の第3引数は原始プログラムにおいても変数であるにもかかわらず置換されているが，これは，同一の変数が2回引数としてあらわれているからである．append の第1引数は入力モードであるから，T 1 に対するユニフィケーションは各節の先頭でおこなっている．また，各節の第3引数は出力モードであるから，T 2 に対するユニフィケーションは各節の末尾でおこなっている．

最後に，本変換について説明する．かんたんにするため，2つの節から構成される手続きのマスク演算方式への変換にかぎってのべる．1個または3個以上の節から構成される手続きについては後述する．インデクス方式への変換についても後述する．

変換すべき手続き P の標準形がつぎのとおりだとする．

P(X 1, …, Xl) :- Q 11, …, Q 1 n_1.　　(3.3.10)
P(X 1, …, Xl) :- Q 21, …, Q 2 n_2.　　(3.3.11)

ここで，各節に対応する引数は同名の変数としている．

変換後の手続き名を VP とする．本変換によって，つぎのような IL の手続きが出力される．

VP(_, …, _, MI, MI) :-
　v_finished(MI), !.　　(3.3.12)
VP(X 1, …, Xl, MI, MO) :-
　Q 11′, …, Q 1 n_1′, v_or 1(MI, MO 1, MI 2),
　Q 21′, …, Q 2 n_2′, v_or 2(MO 1, MO 2, MO).
　　(3.3.13)

手続き VP の実行中には，v_finished 以外の部分では深いバックトラックはおこらない．すなわち，VP からよばれる手続きからバックトラックでとびだしたり，それへとびこんだりすることはない．

第1節(3.3.12)は変換前の手続きには対応する部分がない．この部分は，むだな計算をはぶくとともに，VP の無限再帰をふせいでいる．手続き v_finished は IL のくみこみ手続きであり，入力されたマスク・ベクトルのすべての要素の値が false ならば成功し，手続き VP の実行を終了させる．VP が再帰よびだしされない場合は第1節がなくてもただしく動作するが，第2節(3.3.13)の実行がむだに実行されることがあり

うる．VP が再帰よびだしされる場合は，第1節がなければ再帰が無限にくりかえされ，手続き VP の実行は停止しない．

$Q11', \ldots, Q1n_1'$ を**第1節対応部**，$Q21', \ldots, Q2n_2'$ を**第2節対応部**とよぶ．第1節対応部および第2節対応部は，変換前の第1節本体および第2節本体における手続きの名称をベクトル化後のそれに置換するとともに，マスク・ベクトルを引数として追加したものである．Qij が引数としてマスク・ベクトルを1個とる場合は，Qij の前後で同一のマスク・ベクトルを使用する．また，Qij が引数としてマスク・ベクトルを2個とる場合は，Qij のまえでは第1のマスク・ベクトル，Qij のあとでは第2のマスク・ベクトルを使用する．

手続きよびだし Qij がベクトル化できない場合についてのべる．この場合は，Qij′ を，その引数のベクトル長の回数だけ Qij をくりかえしよぶ手続きとすることによって，他の部分をベクトル化することができる．

手続きよびだし Qij がくみこみ手続きである場合についてのべる．原始プログラムのくみこみ手続きに対しては，それぞれ対応するくみこみ手続きを IL に用意する．この論文では，IL のくみこみ手続きの名称は，原始プログラムのそれの名称に ‘v_’ を接頭したものとする．たとえば，‘=\=’ に対しては，‘v_=\=’ を用意する．ただし，現在のベクトル計算機では論理型言語のくみこみ手続きのすべての機能を効率よく実装することは困難なので，かぎられた範囲で使用できる手続きを用意するのが，効率上はのぞましい．たとえば，リストの合成と分解（Lisp でいえば cons と car, cdr）は論理型言語ではともに C=[H|T] によっておこなわれるが，モード解析の結果にしたがって，合成の場合は v_cons，分解の場合は v_carcdr を使用する．また，リストと空リスト [ ] とのユニフィケーションは，同様にモード解析の結果にしたがって，空リストかどうかの判定の場合には v_null，空リストの代入の場合には v_nullify を使用する．

(3.3.13) における v_or 1 および v_or 2 はマスク・ベクトルの値を合成する IL のくみこみ手続きであり，これらの引数はすべてマスク・ベクトルである．v_or 1(MI, MO 1, MI 2) は手続きの開始前のマスク・ベクトル MI と，原始プログラムの第1節対応部が出力する MO 1 とから第2節実行開始時のマスク・ベクトル MI 2 を合成する．MO 1 は $Q1n_1'$ が出力するマスク・ベクトルであり，MI 2 は Q 21′ が入力するマスク・ベクトルである．また，v_or 2(MO 1, MO 2, MO) は MO 1 と第2節対応部から出力される MO 2 とから，手続き VP が出力すべきマスク・ベクトルを合成する．

v_or 1 および v_or 2 は論理値を入力するから，論理演算をおこなう．したがって，これらの手続きの機能はつぎのようにあらわれる．

$$MI2 = MI \wedge \sim MO1. \qquad (3.3.14)$$

$$MO = MO1 \vee MO2. \qquad (3.3.15)$$

ここで ‘$\sim$’ は論理否定，‘$\wedge$’ は論理積，‘$\vee$’ は論理和をあらわす．これらの手続きの意味はつぎのとおりである．決定性手続きにおいては，第1節が失敗したときだけ第2節が実行される．したがって，v_or 1 は IL の手続きの入力マスクが true かつ第1節対応部の出力マスクが false のときだけ第2節対応部の入力マスクを true とする．また，第1節対応部，第2節対応部のうちのいずれかが成功すれば手続きの実行が成功したことになるので，v_or 2 はこれらが出力するマスクの論理和をとったものを結果とする．

例として not_take 1 をとりあげる．前記の標準形に本変換をほどこした結果はつぎのようになる．

```
v_not_take 1(_, _, _, MI, MI) :-
    v_finished(MI), !.                        (3.3.16)
v_not_take 1(B, Qa, Qs, MI, MO) :-
    v_null(B, MI, MO 1), v_or 1(MI, MO 1, M 1),
    v_carcdr(Q, R, B, M 1, M 2),
    'v_=\='(Q, Qa, M 2, M 3),
    'v_=\='(Q, Qs, M 3, M 4),
    'vs_+'(Qa, 1, Qaa, M 4),
    'vs_-'(Qs, 1, Qss, M 4),
    v_not_take 1(R, Qaa, Qss, M 4, MO 2),
    v_or 2(MO 1, MO 2, MO).                   (3.3.17)
```

金田ら[12]では，手続き append のベクトル化過程と IL によるプログラムをしめしている．

原始プログラムの節の個数が2個以外の場合も，節の本体の変換方法は，節が2個の場合と基本的にかわらない．マスク・ベクトル合成は，節が1個の場合は必要がない．節が3個以上の場合は，図3にしめすようにマスク・ベクトルを逐次的に合成する．

最後に，マスク演算方式以外の条件制御方式への変換についてのべる．インデクス方式の場合は，節の本体の変換方法はほぼおなじである．ただし，マスク・ベクトルではなくインデクス・ベクトルを引数として

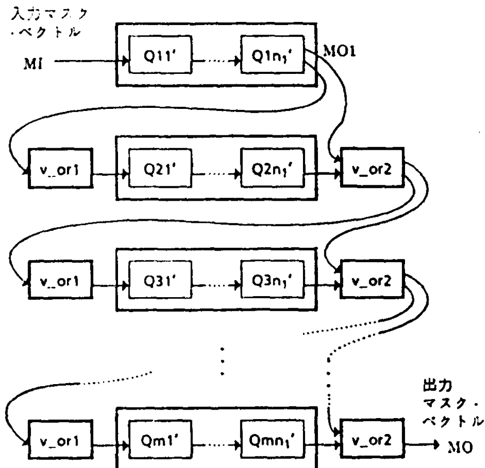


図 3 マスク演算方式におけるマスク・ベクトルの合成
Fig. 3 Merging mask vectors by masked operation method.

追加し，マスク・ベクトルの合成のかわりにインデクス・ベクトルを合成する．すなわち，各節本体から出力されるインデクス・ベクトルの全要素からなるベクトルを手続きの出力インデクス・ベクトルとする．

4. 非決定性手続きのベクトル化法

この章では非決定性手続きのベクトル化法を，再帰的な手続きと非再帰的な手続きとにわけてのべる．ただし，手続きよびだしのベクトル化法は，両者に共通である．

4.1 手続きよびだしのベクトル化

N クウィーンのプログラムを構成する手続き select をベクトル化した手続きを v_select とし，つぎのような 2 個の非決定性手続きのよびだしに対応する v_select のよびだしのベクトル化をかんがえる．

?- select([2, 4], X, Y). (4.1.1)

?- select([1, 3], X, Y). (4.1.2)

これらの手続きよびだしを実行すると，(4.1.1) からは (4.1.3) と (4.1.4)，(4.1.2) からは (4.1.5) と (4.1.6) とが解としてえられる．

X=2, Y=[4]. (4.1.3)

X=4, Y=[2]. (4.1.4)

X=1, Y=[3]. (4.1.5)

X=3, Y=[1]. (4.1.6)

よびだし (4.1.1) および (4.1.2) をベクトル化した結果は，IL でつぎのように表現することができる．

?-v_select(#([2, 4], [1, 3]), VX, VY,
#(true, true), BI, BO). (4.1.7)

ここで，v_select のよびだしの第 1 引数は，select の各よびだしにおける第 1 引数とひとしい値を要素とするベクトルである．第 4 引数は入力マスク・ベクトルであり，第 1 引数の各要素の有効性をしめす．第 1 引数の 1 個の要素から複数の解が生成されうるので，第 2～3 引数のベクトルの各要素は第 1 引数の各要素とは対応しない．たとえば，(4.1.7) をよびだした結果はつぎのようになる．

VX=#(2, 4, 1, 3),
VY=#([4], [2], [3], [1]). (4.1.8)

VX と VY の要素数はひとしく，それらの要素は対応している．ただし，要素の順序は上記のとおりだとはかぎらない．

つぎに第 5～6 引数 BI, BO について説明する．これらの引数は，ベクトルの要素の対応づけのためにもうけている．したがって，**対応づけ引数**とよぶ．図 1 の手続き put の第 2 節において，put に入力されたデータ Qs, B のうち手続き select のよびだし後に Qs はもはや使用されないが，B は使用される．したがって，B についてだけは X, Y との要素の対応をつける必要がある．たとえば，(4.1.1) においては B=[1, 3]，(4.1.2) においては B=[2, 4] だとする．

BI=#([1, 3], [2, 4]).

BO=#([1, 3], [1, 3], [2, 4], [2, 4]).

上記の説明からあきらかなように，対応づけ引数の個数は，非決定性手続きのよびだし点前後のデータフローに依存する．したがって，原始プログラムの手続き定義だけがあたえられただけでは何個の対応づけ引数が必要であるかがわからない．

4.2 非再帰的手続き定義のベクトル化

この節では，再帰よびだしをふくまない非決定的手続き定義のベクトル化法をしめす．この節のベクトル化手順を適用するためには，対象となる手続き P はつぎのすべての条件をみたさなければならない．

条件 2 P の定義は '!', '\+' などの論理性をくずす手続きよびだしや，';' をふくまない．

条件 3 P のすべての引数は入力モード，出力モードのうちのいずれかである．ここで，ある引数が入力モードであるとは，その仮引数が基底項であることを意味する．また，ある引数が出力モードであるとは，その仮引数が論理変数であることを意味する．

条件 4 P のよびだし前で定義され，よびだし後に使用されるすべての変数の値は P のよびだし時

に基底項である．

条件5 Pの定義は再帰よびだしをふくまない．

条件2は3.3節におけるのと同一である．条件3，条件4がなりたたない手続きは，ベクトル化不能である．ただし，この節でのべる手順の拡張などによって条件3，条件4をゆるめることによって一部の手続きはベクトル化可能になるが，その方法はこの論文ではのべない．条件5はかなりきびしい条件であるから，これをゆるめる方法を4.3節で検討する．なお，非決定性手続きであることは，条件ではない．決定性手続きをベクトル化するのにこの節の手順を使用すれば，3.3節の手順で生成されるのより効率がわるい手続きが出力されるだけである．

非決定性手続きのベクトル化は，およそつぎのような手順でおこなうことができる．

ステップ1 モード解析

ベクトル化すべき手続きの各引数が入力モードであるかどうか，また，出力モードであるかどうかを判定する．解析の結果，すべての引数が入力モードか出力モードかに分類されたときだけ，以下のステップを実行することができる．

ステップ2 標準化

ベクトル化を容易にするため，プログラムを標準形に変換する．

ステップ3 本変換（ベクトル化）

各手続き名をベクトル化された手続きの名称で置換するとともに，マスク・ベクトル，インデクス・ベクトルなどの引数を追加する．

モード解析については3.3節でのべた．また，標準化は，決定性手続きとまったく同様におこなえばよい．

本変換について説明する．

かんたんにするため，つぎのような制限をもうける．まず，2つの節から構成される手続きのマスク演算方式への変換にかぎってのべる．また，引数のモードは第1〜k引数が入力モード，第$k+1$〜l引数が出力モードだとする．さらに，対応づけ引数は1組（2個）とする．これらの制限をゆるめるのは容易である．

変換すべき手続きPの標準形は(3.3.10)〜(3.3.11)のとおりだとする．変換後の手続き名を VP とすれば，本変換の結果，つぎのような IL の手続きが出力される．

```
VP(_, ..., _, MI, _, _):-
    v_finished(MI), !.                        (4.2.1)
VP(X1, ..., Xl, MI, BI, BO):-
    Q11', ..., Q1n1', Q21', ..., Q2n2',
    v_merge([[X1 k+1, X2 k+1],
             [X1 k+2, X2 k+2], ...,
             [X1l, X2l], [B1O, B2O]],
            [Xk+1, Xk+2, ..., Xl, BO],
            [M1, M2]).                        (4.2.2)
```

第1節(4.2.1)のはたらきは，決定性手続きの場合と同様にむだな計算をはぶき，無限再帰をふせぐことである．(4.2.2)におけるデータのながれを**図4**にしめす．第1節対応部 Q11', ..., Q1n1' および第2節対応部 Q21', ..., Q2n2' は，変換前の第1節，第2節がすべて決定性手続きのよびだしで構成されていれば，決定性手続きの本変換と同様に変換すればよい．非決定性手続きのよびだし Qij' があらわれるときはつぎのようにする．前換前の手続き Qij よりまえすなわち Q11', ..., Q1 j−1' で定義され，Qij のあとすなわち Qi j+1', ..., Qini で使用される変数は，Qij' の対応づけ引数として入出力するようにする．

上記の説明では，変換前の各論理変数をどのように置換することによって変換後の各論理変数をえるかをしめさなかった．正確な変数の置換方法をしめすと煩雑になるので，かわりに後述の例において説明する．

IL のくみこみ手続き v_merge の機能を，**図5**を使用して説明する．v_merge は，ベクトルを併合する．すなわち，複数のベクトルにふくまれるすべての要素を1つのベクトルの要素にする．ベクトル計算機においては，ベクトル長が数10以上にならないとパイプライン演算器をいかしきれないので，v_merge によってベクトル長をのばしている．v_merge に入力する複数のベクトルはリストにしている．(4.2.2)でいえば，

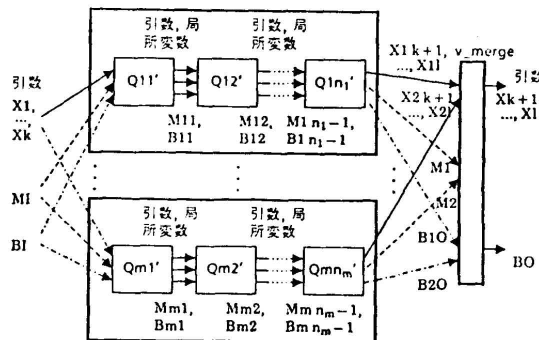


図4 ベクトル化後の非決定性手続きの実行におけるデータのながれ

Fig. 4 The dataflow in the execution of vectorized nondeterministic procedure.

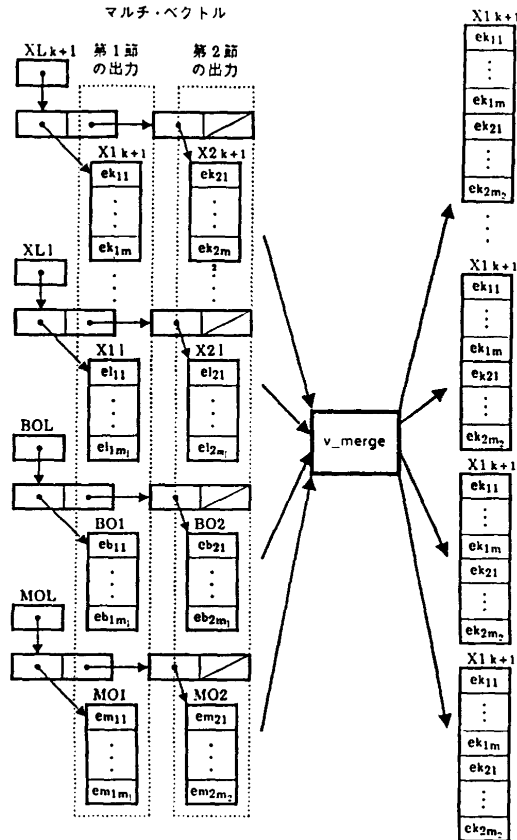


図 5　ベクトルを併合する手続き v_merge の機能
Fig. 5　The function of v_merge procedure which merges vectors.

X 1 k+1 と X 2 k+1, B1O と B2O などをそれぞれリストにしている．このようなリストを**マルチ・ベクトル**とよぶ．v_merge は，ベクトルの併合を複数のマルチ・ベクトルのそれぞれについて実行する．

以下，例をしめす．手続き p のつぎのような標準形を本変換する．p の第 1 引数は入力モード，第 2 引数は出力モードだとする．

p(X, Y) :- q 11(Y), q 12(X, Y), q 13(Y).　　(4. 2. 3)

p(X, Y) :- q 2(X, Y).　　(4. 2. 4)

手続き q 12, q 2 は決定性手続きであり，手続き q 11, q 13 は非決定性手続きだとする．また，p のよびだし以前に定義された 1 個の値がよびだし後に使用されるとする．ベクトル化後の手続きはつぎのようになる．

```
v_p(_, _, MI, _, _) :-
    v_finished(MI), !.                    (4. 2. 5)
v_p(XI, YO, MI, BI, BO) :-
    v_q 11(Y 11, MI, XI, X 11, BI, B 11),
    v_q 12(X 11, Y 11, _, M 12),
    v_q 13(Y 11, M 12, Y 11, Y 12, B 11, B 12),
    v_q 2(XI, YI, MI, M 21),
    v_merge([[Y 12, YI], [B 12, BI]],
            [YO, BO], [M 12, M 21]).
                                          (4. 2. 6)
```

v_q 11, v_q 13 にはそれぞれ 2 組の対応づけ引数がある．v_p の第 1 引数は入力であるため，第 1 仮引数が v_q 11 のよびだしにおける対応づけ引数としてつかわれている．v_q 11 のよびだし以後は，変換前の変数 X は X 11 で置換されている．また，v_p の第 2 引数は出力であるため，v_merge の出力がそのまま第 2 仮引数とされている．v_q 11 のよびだし以前では，変換前の変数 Y は Y 11 で置換され，v_q 11 のよびだしから v_q 13 のよびだしのあいだでは Y 12 で置換されている．'_' は全要素が true であるマスク・ベクトルをあらわしている．

v_merge は，ベクトル Y 12 と YI とを併合して YO をつくり，B12 と BI とを併合して BO をつくるために使用されている．v_merge で入力引数 YI, BI をそのまま使用しているのは，変換前の第 2 節 (4. 2. 4) には決定性手続きだけがあらわれるからである．

つぎに，上記のプログラム変換の 3 つの性質についてのべる．

第 1 に，解がえられる順序は，一般には保存されない．すなわち，解をふくむベクトルにおいて，要素がならぶ順がふかさ優先順であることは保証できない．また，幅優先順であることも保証できない．ただし，特定の条件がなりたてば，解の順序を保証することができる．*N* クウィーン問題のプログラムに関する解の順序の保証に関しては後述する．

第 2 に，もとのプログラムが全解探索で停止しない場合は，ベクトル化されたプログラムは停止しない．なぜなら，変換後のプログラムにおいては，非決定性手続きにおいてすべての解がベクトルに蓄積されるまでは，その実行が終了しないからである．

第 3 に，第 2 の性質と同様の原因から，もとのプログラムが例外をおこす場合は，ベクトル化後のプログラムの動作は保証されないということがいえる．

4.3 再帰的手続き定義のベクトル化

この節では，再帰よびだしをふくむ非決定性手続きのベクトル化法をしめす．この節でのべるベクトル化手順を適用するためには，対象となる手続き P は 4.2 節の条件 2 ～ 4 とともにつぎの条件 6 をみたさなければならない．

条件6　Pの定義を構成する節のうちの1つが，1個の自己再帰よびだしをふくむ．他の節は再帰よびだしをふくまない．

条件6はゆるい条件とはいえないが，実際にみられるほとんどの非決定性再帰手続きはこの条件をみたしている．必要であれば，v_merge にかわる適当なくみこみ手続きを IL にとりいれて使用することによって，条件6をゆるめることができる．

ベクトル化の概略手順は4.2節でのべたのとかわらない．標準化まではまったく同一である．したがって本変換についてのべる．かんたんにするため，2つの節から構成され，第2節が再帰よびだしをふくむ手続きのマスク演算方式への変換にかぎってのべる．変換すべき手続きPの標準形はつぎのとおりだとする．

```
P(X 1, ..., Xl) :- Q 11, ..., Q 1 n1.          (4.3.1)
P(X 1, ..., Xl) :-
  Q 21, ..., Q 2 n21, P(X 1', ..., Xl'),
  Q 2 n21+1, ..., Q 2 n22.                     (4.3.2)
```

ここで，引数のうち X 1, ..., Xi が入力，Xi+1, ..., Xl が出力だとする．変換後の手続き名を VP とすれば，本変換の結果，つぎのような IL の手続きが出力される．

```
VP(X 1, ..., Xl, MI, BI, BO) :-
    VP_1(X 1, ..., Xi, XLi+1, ..., XLl,
        MI, ML),
    v_merge([XLi+1, ..., XLl],
        [Xi+1, ..., Xl], [BI], [BO], ML).
                                               (4.3.3)
VP_1(_, ..., _, [], ..., [], MI, []) :-
  v_finished(MI), !.                           (4.3.4)
VP_1(X 1, ..., Xi,
    [XHi+1|XLi+1], [XHl|XLl],
    MI, [MH|ML]) :-
  Q 11', ..., Q 1 n1', Q 21', ..., Q 2 n21',
  VP_1(X 1', ..., Xi', XLi+1', ..., XLl',
      MI', ML'),
  MQ 2 n21+1', MQ 2 n22'.                      (4.3.5)
```

手続き VP_1 においては，第 $i+1$〜l 引数および最後の引数はマルチ・ベクトルである．VP_1 の再帰よびだしのたびに，これらの各マルチ・ベクトルに1個の要素がくわえられていく．手続き VP は VP_1 をよびだしたあと，v_merge をつかってこれらのマルチ・ベクトルを併合して，それぞれ1個のベクトルにする．

第1節対応部と Q 21, ..., Q 2 n21 の変換法は4.2節におけるのと同様である．これらがベクトルに作用するのに対して，Q 2 n21+1, ..., Q 2 n22 は，マルチ・ベクトルに作用する．ただし，VP_1 のよびだし以前の部分で定義される値はマルチ・ベクトルではないので，これらのベクトルの要素をマルチ・ベクトルの要素ベクトルの要素と対応づけて計算する必要がある．

非決定性再帰的手続きのベクトル化の例として，N クウィーンの手続き v_select をとりあげる．まず標準形をしめす．2つの節に対応する仮引数の変数名を統一するために，一部の変数名は原始プログラムとはかえてある．

```
select(T 1, Y, T 2) :- T 1=[Y|T 2].
                                               (4.3.6)
select(T 1, Y, T 2) :- T 1=[A|L],
  select(L, Y, X), T 2=[A|X].                  (4.3.7)
```

本変換の結果はつぎのようになる．

```
v_select(AL, X, Y, MI, BI, BO) :-
  v_select_1(AL, X 1 L, Y 1 L, MI, ML),
  v_merge([X 1 L, Y 1 L], [X, Y],
        [BI], [BO], ML).                       (4.3.8)
v_select_1(_, [], [], MI, []) :-
  v_finished(MI), !.                           (4.3.9)
v_select_1(AL, [A'|X 1 L], [L'|Y 1 L],
        MI, [M 1'|ML]) :-
  v_carcdr(A', L', AL, MI, M 1'),
  v_carcdr(A, L, AL, MI, M 1),
  v_select_1(L, X 1 L, L 1 L, M 1, ML 1),
  map_v_cons(A, L 1 L, Y 1 L, ML 1, ML).
                                               (4.3.10)
map_v_cons(_, [], [], [], []).                 (4.3.11)
map_v_cons(A, [XH|XL], [XH|YL],
        [MIH|MIL], [MOH|MOL]) :-
  v_cons(A, XH, YH, MIH, MOH),
  map_v_cons(A, XL, YL, MIL, MOL).
                                               (4.3.12)
```

変換前の T 2=[A|X] をマルチ・ベクトルの全要素について実行するのが手続き map_v_cons である．

なお，4.2節でのべたように一般には解がえられる順序はベクトル化によって保存されないが，N クウィーンにおいては順序を保存することが可能である．その方法と順序が保存される理由についてのべる．v_select_1 には非決定性手続きよびだしがふくまれないため，これらの手続きをマスク演算方式で実装すれ

ば，v_select_1 が出力する各マルチ・ベクトルを構成する要素ベクトルのベクトル長はひとしくなる．さらに，要素ベクトル V 1, …, Vm の第 i 要素は，v_select のよびだしにおける入力ベクトルの第 i 要素からえられたものである．したがって，これらを v_merge でつぎのような順にならべれば，原始プログラムのふかさ優先順と一致する．

V 1[1], …, Vm[1], V 1[2], …, Vm[2], …

v_put においては一般には解の順序は保存されないが，N クウィーンの場合には解が v_put の $N+1$ 回めのよびだしにおいて一度にえられるため，保存される（ことなる N について同時に解をもとめようとすれば，解の順序は保存されない）．

金田ら[12]では，非決定性の場合の手続き append のベクトル化過程と IL プログラムもしめしている．

なお，上記のように条件 2〜4, 6 がみたされれば変換をおこなうことはできるが，変換可能なすべてのプログラムにおいて高速実行が実現されるわけてはない．すなわち，この変換によって十分なベクトル長がえられることが必要である．マクロにみれば，この処理方式は，少数の節が集中的に実行されるプログラムに適している．

5. 評　　価

第 3 〜 4 章でしめした論理型言語 IL のインタプリタを Prolog で作成して，N クウィーンのプログラムなどのベクトル化後のプログラムの動作をたしかめた．また，N クウィーンのプログラムを Fortran と Pascal とに手動で変換し，Fortran 部分をベクトル・コンパイラでコンパイルしたうえ実行時間を測定した．その結果は金田ら[11),12)]でしめしたが，マスク演算方式の場合，8 クウィーンで 4.5 MLIPS の実行速度をえた．

さらに，この論文でしめした手順でマスク演算方式によるベクトル化後のプログラムを（IL ではなく）Lisp の構文・意味にしたがって出力する自動ベクトライザを作成した．また，自動ベクトライザが出力する Lisp プログラムをベクトル計算機 S-810 のベクトル命令をふくむ目的プログラムに変換するコード生成系およびその実行系を Lisp 上に作成した．図 1 にしめした N クウィーンの Prolog プログラムをこの処理系で翻訳して実行し，動作を確認した．その結果，8 クウィーンで 2.6 MLIPS の実行速度をえた．最適化が不十分なため，自動変換したプログラムは手動変換したものにくらべて低速だが，今後改良すれば，後者をうわまわる実行速度がえられるであろう．

6. 結　　論

ベクトル計算機を使用して OR 並列性がある論理型言語プログラムを並行処理するためのプログラム変換法（ベクトル化法）を開発した．この方法においては，原始プログラムの論理変数ごとに，それがとりうる複数の値を各要素とするベクトルをつくってベクトル処理する．

この方法を N クウィーン問題のプログラムに適用して自動変換し，生成されたプログラムの動作のただしさを確認するとともに，ベクトル計算機 S-810 によって実行して 2.6 MLIPS という高速性能をえた．現在の方法は，引数の入出力モードが確定していなければならない，高速化されるプログラムの範囲がかぎられているなどの制約がある．また，金田ら[9]でしめしたようなメモリ消費をおさえる方法がとりいれられていないため，OR 並列性が非常にたかいプログラムにおいては計算がつづけられなくなりうるという問題点がある．したがって，今後，ベクトル化法を改良して適用範囲の拡大をはかりたい．また，最適化による速度向上をはかりたい．

謝辞　この研究をすすめる過程でしばしば議論をしていただいた日立製作所中央研究所の小島啓二研究員と安村通晃主任研究員，この研究を支援していただいた同ソフトウェア工場 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ほかの方々に感謝します．

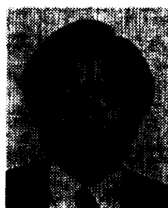
参 考 文 献

1) Komatsu, H., Tamura, N., Asakawa, Y. and Kurokawa, T.: An Optimizing Prolog Compiler, *The Logic Programming Conference '86*, pp. 143-149, Japan (1986).
2) Yamaguchi, S., Bandoh, T., Kurosawa, K. and Morioka, M.: Architecture of High Performance Integrated Prolog Processor IPP, *Fall Joint Computer Conference*, pp. 175-182 (1987).
3) Goto, A. and Uchida, S.: Toward a High Performance Inference Machine—The Intermediate Stage Plan of PIM—, *Future Parallel Computers*, Lecture Notes in Computer Science, No. 272, Springer-Verlag (1986).
4) Nilsson, M. and Tanaka, H.: A Flat GHC Implementation for Supercomputers, *Fifth International Symposium on Logic Programming*,

(1988).
5) Nilsson, M. and Tanaka, H.: Massively Parallel Implementation of Flat GHC on the Connection Machine, *International Conference on Fifth Generation Computer Systems*, pp. 1031-1040 (1988).
6) 辰口和保，村岡洋一：ベクトル計算機上の並列論理型言語処理系，第 35 回情報処理学会全国大会論文集，5Q-1, pp. 753-754 (1987).
7) 金田 泰，菅谷正弘：プログラム変換にもとづくリストのベクトル処理方法とそのエイト・クウィーン問題への適用，情報処理学会論文誌投稿中.
8) 金田 泰，小島啓二，菅谷正弘：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」，情報処理学会論文誌，Vol. 29, No. 10, pp. 985-994 (1988).
9) 中島秀之：Prolog，産業図書 (1983).
10) Ueda, K.: Making Exhaustive Search Programs Deterministic, *Third International Conference on Logic Programming*, Lecture Notes in Computer Science, No. 255, pp. 283-297, Springer-Verlag (1986).
11) 金田 泰：ベクトル計算機による論理型言語プログラムの高速実行をめざして一各種 OR ベクトル実行方式の実現と性能一，情報処理学会プログラミング言語研究会，PL-87-12 (1987.6).
12) Kanada, Y., Kojima, K. and Sugaya, M.: Vectorization Techniques for Prolog, *ACM International Conference on Supercomputing*, pp. 539-549 (1988).

(昭和 63 年 11 月 18 日受付)
(平成元年 1 月 17 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．



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．