記　号　処　理　54－2
（１９９０．　３．　１２）

スーパコンピュータのための ベクトル化Ｌｉｓｐコンパイラ

阿部一裕　安井　裕
（大阪大学　工学部）

本稿では，スーパコンピュータのベクトル演算機能を積極的に用いてLispプログラムを並列処理する（ベクトル化する）ための変数束縛の管理法とプロセスの制御法を提案する．また，これらの方式を適用してベクトル化したコードを生成するコンパイラ　-Vectorizing Lisp Compiler- についてのべる．

このコンパイラをもちいることにより，リスト操作関数や条件分岐により複雑な制御をおこなう関数をベクトル化できるようになった．また いくつかのプログラムをコンパイルしスーパコンピュータSX-2上で実行したところ，8-Queen問題のような並列性の高いプログラムではベクトル化率は95%以上，処理速度は逐次的に実行する場合にくらべ10倍以上の向上が得られた．

The Vectorizing Lisp Compiler for Supercomputers

Kazuhiro Abe　Hirosi Yasui

Department of Applied Physics, Faculty of Engineering, Osaka University
2-1, Yamadaoka, Suita, Osaka 565, Japan

In this paper we propose a new method for parallel-executing Lisp programs on supercomputers. This method includes a variable binding management and a process control. We also describe the implementation of a compiler using by this method. We call this compiler Vectorizing Lisp Compiler.

Our compiler can generate vectorized code corresponding to Lisp functions that manipulate list structures and that include conditional expressions. We compiled some benchmark programs by the compiler and then executed them on SX-2. In the case of highly parallel programs such as the 8-Queen problem, the vector processing performance is 10 times higher than the sequential processing one and a vectorization ratio is more than 95 percent.

1 はじめに

スーパコンピュータのベクトル演算機能を，数値処理ばかりでなく記号処理でも利用しようとする研究が各方面でなされている．

論理型言語をスーパコンピュタ上で高速実行しようとした研究[1),2),3)]では，複数解を並列に探索する過程でベクトル演算が使用できることが示された．

Lispには多数のデータに同一の関数を適用する制御構造としてmap関数があり，map関数から呼び出す関数を実行するときにベクトル演算をもちいて並列処理することが考えられる．しかし これまでに発表されてきた，スーパコンピュータ上のLisp処理系ではベクトル演算をもちいて並列処理をおこなう関数は，算術演算，論理演算などの単純な関数だけであった．[4),5),6)]

本稿では，複数個のLispの式を並列に評価する過程でベクトル演算が使用できることを示し，この並列処理を実現するための変数束縛の管理法とプロセスの制御法を提案する．そして これらの方法を適用してベクトル化されたコードを生成する Lispコンパイラ －Vectorizing Lisp Compiler－ についてのべる．

2 Lispプログラムのベクトル化法

2-1 スーパコンピュータのベクトル演算機能

スーパコンピュータは，ベクトルデータの要素に対する一様な処理をベクトル命令をもちいて高速に実行する．

ベクトル命令には次の３つの機能がある

- 図1に示すような，主記憶上に連続的に配置されているデータ，等間隔に配置されているデータ，離散的に配置されている間接指標データとベクトルレジスタ間を一括してロード／ストアする機能．
- ベクトルレジスタの多数の要素に算術演算，論理演算を一括して実行する機能．
- ベクトルレジスタの要素ごとに演算を行うかどうかを制御をする機能．

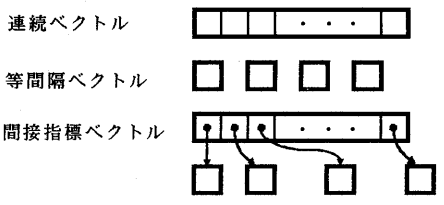

図1 ベクトルデータ

ただしベクトル命令を実行するときには処理する要素の数によらず一定な立ち上がり時間がかかるので，要素の数が少ない場合ベクトル命令を使用することによりかえって処理時間が遅くなる．

スーパコンピュータのベクトル演算機能を用いてLispの高速化を達成するためには，Lispの処理の中で大量のデータにたいして上記の操作をおこなう機会を見出さなければならない．

2-2 ベクトル演算の対象

Lispの式を評価するときにおこなわれる操作は，プログラムとして定義された関数の場合

- 変数の値の参照
- 関数呼び出し／復帰
- 条件分岐

であり，また組み込み関数の場合

- リスト処理
- 算術演算

である．

逐次的にひとつの式を評価する場合を考えると，列に同一の演算をおこなう組み込み関数を除けば上記の操作の過程で多くのデータに同一の操作をおこなうことはなくベクトル演算を使用する機会はない．逐次実行するLispをスーパコンピュータ上で実行しても，ベクトル演算が使用できず高速化が得られない．

我々は，ひとつの式を逐次的に評価していくのではなく，複数の式を並列に評価することにより上でのべた操作にベクトル演算が適用できる可能性を見出した[7)]．ただし異なる関数を同時に実行することはできないので，

"複数の引数リストに同一の関数を適用すること"

を並列処理の対象とした．すなわち呼び出す関数が同じである複数の式の関数適用を並列におこなう．この様な並列性は，map関数における関数適用を並列化することにより得られる．

式が評価される過程をプロセスと呼ぶことにすると，並列に実行している各プロセスは，同じ関数を呼び出し，その関数の本体を評価する時にパラメータの値が異なる複数の式を並列に評価する．

このような並列処理をベクトル演算をもちいて実行するために本研究で採用した方法の考え方についてのべる．

並列に実行している複数のプロセスが関数を呼び

出すときに，各プロセスごとに変数の値を格納するフレームをつくり，それらをスタックの連続した領域に置く．また各プロセスが値を返す場所を指すポインタ（vp）を別のスタック上に連続して置く（図2）．フレームを置くスタックをフレームスタック（fs），vpを置くスタックをvalueポインタスタック（vs）と呼ぶ．

フレームの中である変数の値を格納するフィールドは決まった位置にあるので，フレームを連続して配置すると，各変数のフィールドはスタック上で等間隔の位置に存在する．したがって変数の値を参照する操作，関数に引数の値を渡す操作をベクトル命令を用いて高速に実行することができる．リスト処理関数や数値演算関数でも引数がスタック上に連続してあるのでリスト処理や数値演算もベクトル命令を使用することができる．

複数の関数適用を一括しておこなうので，関数の呼び出し／復帰 時の分岐の回数が逐次的に実行する場合に比べて減少する．この面からも高速化が得られる．

しかし並列に実行しているプロセスは一様な処理を続けていくのではなく条件分岐があった場合，各プロセスごとに異なった制御の流れをもつ．また局所変数のような宣言されている関数の中だけで参照可能な変数の場合は先に示した方法で実現できるが，大域変数の参照は，このような単純な方法では実現できない．Lispプログラムをベクトル演算処理するには解決しなければならない問題がまだ残っている．

これらの問題を解決するために本研究でとった方法を

並列性を生む形式
プロセスごとの変数束縛の管理法
プロセスの制御法

の順にのべる．

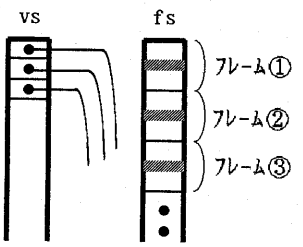

図2　フレームスタック，valueポインタスタック

2-3　並列性を生む形式

本研究では，複数の引数リストに同一の関数を適用することを並列処理の対象とした．　このような並列性が生じる形式として本研究ではvmap関数（vmapcar, vmaplist, vmapcan, vmapcon）及び　vmap関数を拡張したvmapマクロ形式を導入した．

vmapマクロ形式は次式の構文をもつ

(vmap fnβ　fnα　<argList-1> … <argList-N>)
<argList-i> ::= (arg-i1 … arg-iM)

fnαは，組み込み関数だけでなく，プログラムとして定義された関数であってもよい．ただし引数の数が決っている関数でなければならない．fnβは，関数であればどのようなものでもよい．

この式はコンパイル時に次のように展開される．

(fnβ　(fnα arg-11 … arg-1M)
　… (fnα arg-N1 … arg-NM)

ただし評価の順番は，全てのfnαの引数を左から右の順にすべて評価した後，引数リストに<fnα>を適用することを並列におこなう．このときvmapマクロを評価するプロセスひとつあたり，fnαを関数とする式の数（N個）だけプロセスが生成される．

vmapマクロ形式をL個のプロセスが評価したとするとfnαを実行するプロセスの数は，L×N個になる．

vmapマクロ形式は各プロセスにつき増加するプロセスの数が同じであり，またその数がコンパイル時に決定できる形式である．

vmap関数は次式の構文をもつ

(<vmap関数>　fn　list-1 … list-M)
<vmap関数> ::= vmapcar | vmaplist |
　　vmapcan | vmapcon

vmap関数では，ひとつまたは複数のリストの要素のそれぞれに対して関数を適用することを並列におこなう．

vmap関数では，各プロセスにより呼び出す関数が異なる可能性がある．本システムでは実行時に，vmap関数の第1引数の値を調べ，全てのプロセスでvmap関数から呼び出す関数が同じであるかどうかを判断する．すべて同じである場合には，すべてのプロセスで関数の適用を一括して並列におこなう．ひとつでも異なる場合には，ひとつのプロセスごとに関数の適用を並列におこなう．

2-2 変数束縛の管理法

本システムでは，変数の種類として，local変数，special変数，global変数がある．

local変数は，参照の有効範囲が，宣言されているプログラムテキストの範囲内に限られる変数である．仮引数やlet, let*により宣言された変数はspecial変数宣言がなされなければlocal変数となる．

special変数は，参照の有効範囲が，宣言されているプログラムテキストの範囲にとどまらず，そこから呼び出された関数の入れ子の中にも及ぶ変数である．変数をspecial変数として使用する場合には，special変数宣言をおこなう必要がある．

global変数は，参照の有効範囲がプログラム全体に及ぶ変数である．local変数でもspecial変数でもない変数はglobal変数となる．

変数束縛の環境を管理するために，関数を実行するときに各プロセスごとに図3で示されるフレームがつくられる．

図3　フレーム

ある時点におけるspecial変数-値の束縛の数は関数が呼び出された履歴により変化し，コンパイル時に決定することはできない．そのため参照可能なspecial変数-値の束縛は，変数名-値のドット対のリスト（a-list）で管理する．

フレームは実引数，let, let*により宣言された変数，special変数，a-list のためのフィールドから構成される．ただし組み込み関数ではスペシャル変数を参照することはないので，組み込み関数を実行する時につくられるフレームには，a-listを格納するための領域はない．関数が呼び出された時点では，フレームには仮引数とa-listのフィールドにしか値が格納されていず let, let*で宣言される変数，special変数のフィールドの値は不定の状態になっている．

local変数は，フレームのフィールドに値が直接格納されている．

special変数は，フィールドに変数名-値のドット対へのポインタを格納する．special変数の参照は，そのつどa-listを検索するのではなく，あらかじめフィールドにドット対を指すポインタを格納しておく．

仮引数がspecial変数宣言されている場合は次のような処理をおこなう．関数が呼び出された時には変数のフィールドに値が直接格納されている．関数本体の処理をする前に変数-値のドット対をつくり，そのドット対へのポインタをspecial変数のフィールドに格納する．そしてドット対をa-listの先頭に付け加える（図4）．let, let*で宣言された変数がspecial変数宣言されている場合は，変数の初期値を与える式を評価し，その値をフレームに格納した後，同様な操作がなされる．

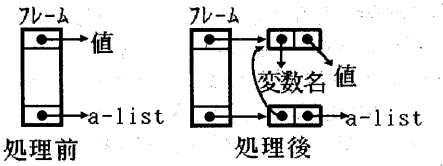

図4　special変数の処理（I）

参照可能なプログラムテキストで宣言されていない変数をspecial変数として参照する場合，関数が呼び出された時にa-listを検索し，変数-値のドット対へのポインタをフィールドに格納する（図5）．

special変数の処理もベクトル演算を用いて実行する．

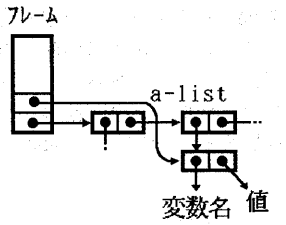

図5　special変数の処理（II）

global変数はシンボルの値フィールドに直接値が書き込まれる．global変数は，すべてのプロセスで同じ束縛が参照される．

引数がすべて変数である式を評価する時には，プロセスの数だけフレームをfs上にとり，変数の値をフレームに格納し，関数を実行する．

引数に式をとる場合，例えば

(foo x (zoo y))

のような式を評価するときは，まず関数fooのためのフレームをfs上にとる．そして，このフレームの第2引数のフィールドにxの値を格納する．次にこのフレームの第2引数のためのフィールドに値を返すようにvsを設定して(zoo x)を評価する（図6）．

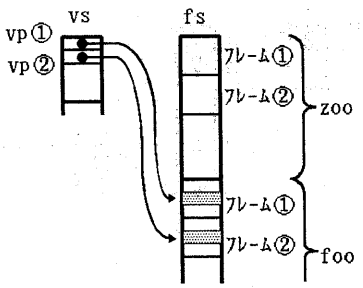

図6　(foo x (zoo y))の処理

2-3　プロセスの制御法

本システムでは，プログラムを構成するための制御構造をとして次の特殊形式，関数がある．

progn, let, let*, setq, vmap, vmapcar, vmaplist, vmapcan, vmapcon, cond, and, or, while

本節では，この中で基本的な制御構造であるvmapマクロ，vmap関数，cond特殊形式を評価する時のプロセスの制御法についてのべる．

・vmapマクロ形式

(vmap fnβ fnα (arg-11 … arg-1N)
…(arg-M1 … arg-MN))

vmapマクロ形式は，コンパイル時に

(fnβ (fnα arg-11 … arg-1N)
…(fnα arg-M1 … arg-MN))

のように展開される．vmapマクロ形式を評価する時には，まずfnβのためのフレームをフレームスタック上にとる．そして，はじめのfnαのためのフレームをとり，その引数を評価する．次に第２番目のfnαのためのフレームを１番目のfnαのフレームの上にとり，その引数を評価する．この操作を最後のfnαについてまで行う．このときフレームスタック上にfnαのフーレムが"vmapマクロ形式を並列に評価するプロセスの数×fnαを関数とする式の数"個積まれる．この数のプロセスがfnαを並列に実行する．

・vmap関数

vmap関数が行う操作には，1)vmap関数を並列に実行するプロセスでvmap関数から呼び出す関数が全て同じかどうかを調べる操作，2)vmap関数から呼び出す関数のためのフレームにリストの要素を格納する操作，3)関数を適用する操作がある．

vmap関数を実行するプロセスi のリストの長さをM_iとする．

1)の操作で，全てのプロセスで呼び出す関数が同じであった場合，まず2)の操作を全てのプロセスが順番に逐次的におこなう．これはリストの長さが各プロセスでことなるためである．このときフレームスタックには，ΣM_i個のフレームが積まれている．そして3)の関数の適用をΣM_i個のプロセスが並列におこなう．

1)の操作で，vmapが呼び出す関数がひとつでも異なる場合，"M_i個のフレームをフレームスタック上に積み，関数の適用をMi個のプロセスが並列におこなう"ことをvmap関数を実行するプロセスが繰り返しておこなう．

・cond特殊形式

(cond (test consequent)…(test consequent))

cond形式を評価する場合，フレームスタック上にそのcond形式を並列に評価するプロセスの数だけ領域をとる．これは，cond式のtest部の値を格納する領域であり，プロセスの実行を制御するマスクとしても使用する．この領域のことをマスク領域と呼ぶ．

最初の節のtest部は全てのプロセスが評価する．このときのvsにマスク領域を指すポインタを格納する（図7）．

consequent部を評価するときは，マスクの値がnil, endでないプロセスのみ評価をおこない，対応するマスクの値をendにする．マスクの値がnil, endであるプロセスは評価を行わない．

endは対応するプロセスの評価が終了したことを表わすシンボルでシステム内でのみ使用される．

２番目以降の節のtest部は，対応するマスクがnilであるプロセスのみ評価をおこない，他のプロセスは評価を行わない．

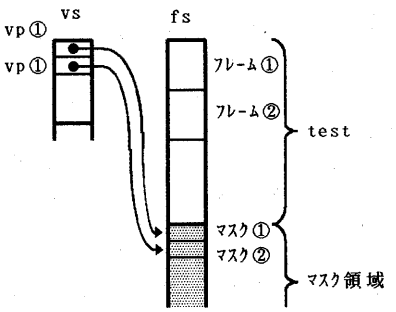

図7　条件分岐の処理

2-4　リスト操作基本関数のベクトル化法

Lispの場合 リスト操作基本関数として car, cdr replca, replcd, cons がある．

複数のプロセスで関数 car を実行する時の，fs,

vsは図8のようになる．car は1引数の組み込み関数であるので，プロセス1個あたりのフレームの大きさは変数1個分である．

関数 car の処理は，次の3つの操作からなる．

1) 各引数の値をベクトルレジスタにロードする．（連続ベクトルのロード命令）

2) 1)でロードしたポインタが指すリストセルのcar部の値をベクトルレジスタにロードする．（ベクトル収拾命令）

3)vsから値を返す場所を指すポインタをロードする．（連続ベクトルのロード命令）

4) 2)でロードした値を3)でロードしたポインタが指す場所にストア する．（ベクトル拡散命令）

括弧の中は使用するベクトル命令である．

cdr, replca, replcd関数も同様にしてベクトル化することが可能である．

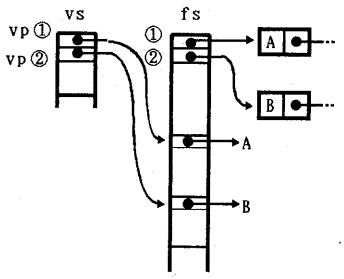

図8　carのベクトル処理

複数のプロセスで関数consを実行する時の，fs, vsは図9のようになる．cons は2引数の組み込み関数であるので，プロセス1個あたりのフレームの大きさは変数2個分である．またガーベッジコレクションはRobsonの一括型コピー方式[8] アルゴリズムを採用しているのでフリーセルは常に連続した領域に存在する．

関数 cons の処理は，次の5つの操作からなる．

1) 各引数の値をベクトルレジスタにロードする．（連続ベクトルのロード命令）

2) フリーセルのcar部，cdr部に，1)でロードした値をストアする．（連続ベクトルのストア命令）

3)vsから値を返す場所を指すポインタをロードする．（連続ベクトルのロード命令）

4)各フリーセルへのポインタを3)でロードしたポインタが指す場所にストアする．（ベクトル拡散命令）

括弧の中は使用するベクトル命令である．

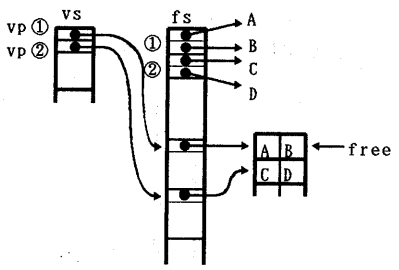

図9　consのベクトル処理

3 コンパイラの概要

本研究で使用したスパーコンピュータSX-2Nでは，FORTRANのみしかユーザに解放されていなかったので，試作したコンパイラは，コンパイル オブジェクトをFORTRANのコードで出力する． 直接ベクトル命令を生成することができないので，2章で示したようなベクトル演算処理がなされるようにFORTRANの do ループで記述し，間接的に用いるベクトル命令を指定する方法をとった．

このコンパイラは2章でのべた方式にもとづき並列実行するコード（並列型）だけでなく，逐次的に実行するコード（逐次型）も生成することができる．

Lispプログラムは，一括してコンパイルし，組み込み関数ライブラリ，I/Oルーチン，ガーベッジコレクタ（これらも全てFORTRANで記述している）を結合してSX-2Nに転送する．そしてSX-2NのFORTRAN77/SXコンパイラでコンパイルし実行する．

FORTRAN77/SXコンパイラは，コンパイル オプションの指定によりスカラ命令のみを用いてコンパイルすることも，ベクトル命令を用いてコンパイルすることもできる．したがって ひとつのLispプログラムの実行方式として

並列型ベクトル実行
並列型スカラ実行
逐次型

の3種類の方式がある．（逐次型ではベクトル命令が使える場所がないので，スカラ命令のみでコンパイルされる）

4 実行結果

いくつかの例題を，今回試作したコンパイラでコンパイルしSX-2N上で実行した．

Tarai, List-Tarai[9] をvmapマクロをもちいて書き直したプログラムと8-Queen-a, b, cのプログラムを本稿の最後のページに示す．8-Queen-bでは，

8-Queen-aの引数の受け渡しの一部をspecial変数をもちいておこなう．8-Queen-cでは，8-Queen-aのvmapマクロの代わりにvmap関数をもちいた．

各プログラムの実行時間と並列型実行した時に最も実行費用がかかった関数の，"費用の割合"*と"並列に実行するプロセスの数の平均値"を表1に示す．またList-Tarai-4について実行中の各操作の実行費用の割合をANALYZER/SXを使用して測定した結果を図10にしめす．

・全体の傾向として Tpv < Tps < Ts の関係がある．

・並列度が高くなるとTs/Tpv, Ts/Tps, Tps/Tpvの値がいずれも大きくなる．

・並列型ベクトル実行で8-Queen-cは，

8-Queen-a, bに比べ2倍以上実行時間がかかる．ことが実行結果よりわかる．

並列度が上がるにつれ Tps/Tpv の値が大きくなりベクトル実行したことの効果があらわれる．しかし，ベクトル命令は処理する要素の数によらず一定の立ち上がり時間がかかる．そのため並列度が低いList-Tarai-3では，ベクトル命令をもちいることによりかえって処理速度が遅くなったと思われる．

本研究で提案した並列処理方式では，関数呼び出しを一括しておこなう．そのため関数間の制御の移動は逐次型と比べ1/(並列に実行するプロセスの数)に減少する．この効果は並列型スカラ実行の場合にもおよぶ．この効果により並列型スカラ実行は，逐次型よりも高速化が得られたと思われる．

vmap関数を使用した8-Queen-cでは，8-Queen-a, bよりベクトル実行したことの効果が上がっていないがこれは，vmap関数ではリストの要素をフレームに積む操作が逐次的におこなわれるためであると考えられる．

4 おわりに

本稿では，スーパコンピュータのベクトル演算機能を積極的に用いてLispプログラムを並列処理するための変数束縛の管理法とプロセスの制御法を提案した．またこの方法にもとづいてベクトル化したコードを生成するコンパイラ -Vectorizing Lisp Compiler- についてのべた．また実行結果よりこの方式がベクトル演算を多くの場所で利用し，高速化が得られることがわかった．

今後の課題として，プロセスの爆発的増加をふせぐために，並列に実行するプロセスの数を動的に変更する方式の開発が望まれる．

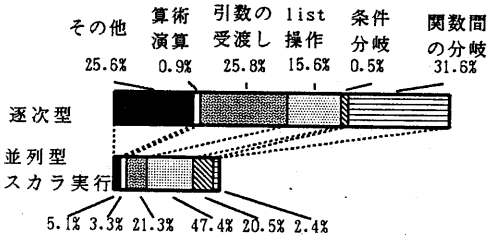

図10 List-Tarai-4実行時の各操作の実行費用

表1 実行結果

	Tpv/ms(β/%)	Tps/ms	Ts/ms	Ts/Tpv	Ts/Tps	Tps/Tpv	並列度 (関数名, 費用/%)
8-Queen-a	37 (99.8)	344	795	21.5	2.3	9.3	1176.1 (conflict, 51.7)
8-Queen-b	41 (99.8)	392	939	22.9	2.4	9.6	1176.1 (conflict, 56.2)
8-Queen-c	95 (96.8)	418	925	9.7	2.2	4.4	1176.1 (conflict, 46.2)
Tarai-3	3 (90.8)	3	2	0.7	0.7	1.0	7.6 (tarai , 81.5)
Tarai-4	16 (94.1)	29	38	2.4	1.3	1.8	33.6 (tarai , 82.1)
Tarai-5	121 (99.0)	663	678	5.6	1.0	5.5	225.0 (tarai , 82.2)
List-Tarai-3	12 (75.7)	9	9	0.8	1.0	0.8	5.6 (copy , 55.0)
List-Tarai-4	56 (89.4)	69	171	3.1	2.5	1.2	23.4 (copy , 52.8)
List-Tarai-5	451 (98.6)	1815	4796	10.6	2.6	4.0	52.0 (copy , 50.0)

Tpv:並列型ﾍﾞｸﾄﾙ実行時間　Tps:並列型ｽｶﾗ実行時間
Ts:逐次型実行時間　β:ﾍﾞｸﾄﾙ化率

*実行費用の測定には，SXシステムに用意されていた性能測定ツール ANALYZER/SX[10] を使用した．ただしANALYZER/SXから求められるFORTRANの各文の実行費用はスカラ命令をもちいた時の費用で換算される．

```
Tarai
(defun  tarai   (x y z)
  (cond
    ((greaterp x y)
      (vmap tarai tarai ((subl x) y z)
                        ((subl y) z x)
                        ((subl z) x y) ))
    (t
          y) ))

List-Tarai
(defun  List-Tarai  (x y z)
  (cond
    ((lessp (car x) (car y))
      (vmap List-Tarai List-Tarai ((copy (cdr x)) y z)
                                  ((copy (cdr y)) z x)
                                  ((copy (cdr z)) x y) ))
    (t
          y) ))

8-Queen-a
(defun  queen-a ()
  (q-a 9 nil nil))
(defun  q-a     (x y board)
  (cond
    ((greaterp x 8)
      (vmap conc q-a (8 1 nil) (8 2 nil) (8 3 nil) (8 4 nil) (8 5 nil)
            (8 6 nil) (8 7 nil) (8 8 nil)))
    ((conflict-a  x y (addl x) board)
      nil)
    ((eq x 1)
      (list (cons y board)))
    ( t
      (let ((xx  (subl x))
            (b (cons y board)))
        (vmap conc q-a (xx 1 b) (xx 2 b) (xx 3 b) (xx 4 b)
                       (xx 5 b) (xx 6 b) (xx 7 b) (xx 8 b))))))
(defun  conflict-a  (x y  xl board)
  (cond
    ((null board)
       nil)
    ((eq y (car board))
      t)
    ((eq (abs (difference y (car board))) (difference xl x))
      t)
    ( t
      (conflict-a  x y  (addl xl) (cdr board)))))

8-Queen-b
(defun  queen-b ()
  (q-b 9 nil nil))
(defun  q-b     (x y board)
  (declare (special x y))
  (cond
    ((greaterp x 8)
      (vmap conc q-b (8 1 nil) (8 2 nil) (8 3 nil) (8 4 nil)
                     (8 5 nil) (8 6 nil) (8 7 nil) (8 8 nil)))
    ((conflict-b  (addl x) board)
      nil)
    ((eq x 1)
      (list (cons y board)))
    ( t
      (let ((xx  (subl x))
            (b (cons y board)))
        (vmap conc q-b (xx 1 b) (xx 2 b) (xx 3 b) (xx 4 b)
                       (xx 5 b) (xx 6 b) (xx 7 b) (xx 8 b))))))

(defun  conflict-b  (xl board)
  (declare (special x y))
  (cond
    ((null board)
      nil)
    ((eq y (car board))
      t)
    ((eq (abs (difference y (car board))) (difference xl x))
      t)
    ( t
      (conflict-a  x y  (addl xl) (cdr board)))))

8-Queen-c
(defun  queen-c ()
  (q-c 9 nil nil))
(defun     q-c    (x y board)
  (cond
    ((greaterp x 8)
      (vmapcan 'q-c '(8 8 8 8 8 8 8 8) '(1 2 3 4 5 6 7 8)
                    '(nil nil nil nil nil nil nil nil)))
    ((conflict-a  x y (addl x) board)
      nil)
    ((eq x 1)
      (list (cons y board)))
    ( t
      (let ((xx  (subl x))
            (b (cons y board)))
        (vmapcan 'q-c
          (list xx xx xx xx xx xx xx xx) '(1 2 3 4 5 6 7 8)
          (list b b b b b b b b))))))
```

図１１　vmapマクロ，vmap関数を用いたプログラム例

【参考文献】

1)金田 泰 他: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，情報処理学会論文誌，vol. 30, no. 4, pp. 495-506 (1989).
2)M. Nilaaon, et all.:A 1.1 MLips(i.e. MHz)Flat GHC Interpreter for the Hitachi Supercomputer S-820，情報処理学会第37回全国大会，7Y-3 (1988).
3)辰口 和保 他:スーパコンピュータ上の並列論理型言語処理系，情報処理学会第36回全国大会，5H-2 (1988).
4)梅村 恭司:スーパコンピュータ上での記号処理系，情報処理学会第32回全国大会，4G-7 (1985).
5)J. W. Anderson, et all.:Implementing and Optimizing Lisp for the Cray, IEEE Software, July, pp. 74-pp. 83 (1987).
6)植松 尚士 他:Lispからのベクトルプロセッサの利用，情報処理学会第40回全国大会，1G-9 (1990).
7)阿部 一裕 他:スーパコンピュータ（ベクトル計算機）のための並列 Lispコンパイラ，情報処理学会第40回全国大会，1G-8 (1990).
8)J. M. Robson:A Bounded Storage Algorithm for Copying Cyclic Structure, Comm. ACM, vol. 20, pp. 431-433 (1977)
9)奥乃 博:第3回 Lispコンテスト及び第1回 Prologコンテスト報告，情報処理学会，SYM-33-4 (1985).
10)ANALYZER/SX説明書，GGB14-2，NEC (1985).