ｖｍａｐマクロ，ｖｍａｐ関数を用いたプログラミング及びベクトル化Ｌｉｓｐコンパイラでの実行とその考察

4M－1

小林一隆，阿部一裕，安井 裕
(大阪大学・工学部)

1．はじめに

我々の研究室で開発したベクトル化Ｌｉｓｐコンパイラ[1)2)]は，スーパーコンピュータのベクトル演算機能を積極的に利用するためにＬｉｓｐプログラムをベクトル処理化されるコードに変換する処理系である．そして，すでに新しいデータ構造及びvmapマクロ，vmap関数の提案，実現と共にその他ベクトルプロセッサを使用する関数を作成，実装しているが，本研究では，グラフ問題[3)]，その他探索問題のいくつかの例題について，ベクトル化Ｌｉｓｐコンパイラを用いて処理を行ない，実行した．

本稿では，vmapマクロ，vmap関数を用いたプログラミングについて述べ，実行結果とともに本処理系の有効性を示す．

2．ベクトル化の対象

ベクトル演算は多数の要素に同一の演算を施すものであるので，多数の要素に異なる関数を同時に実行することはできない．しかし，我々は本処理系においてベクトル処理に適した新しいデータ構造とともに，vmapマクロ（図１）及びvmap関数（図２）を提案，導入し，並列処理の対象として，複数の引数に同一の関数を適用することにより積極的にベクトル処理を行なうことを可能としている．

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

図１．vmapマクロ形式の構文

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

図２．vmap関数の構文

式が評価される過程をプロセスと呼ぶことにすると，vmapマクロまたはvmap関数が評価されるたびに，プロセスの数は次々と増加する．増加したプロセスで呼ばれる関数は，リスト操作関数や条件分岐により複雑な制御を行なう関数も含め，全ての関数がベクトル化の対象となっており，ベクトル演算機能により高速に処理される．

3．vmapマクロ，vmap関数の利用法

本処理系では，プロセスの数がベクトル長に比例している．そのため，プロセスの平均の数が少なく，すなわちベクトル長が短くなるようなプログラムでは，ベクトル処理を行なうために必要となるオーバーヘッドによる時間的な損失が大きく占め，スカラ実行した場合よりも低速になる場合がある．従って，より高速な処理を実現するためには，オーバーヘッドによる損失が見かけ上なくなるベクトル長以上のプロセス数が常に存在しているのが望ましく，プログラムの作成に当り，適度にプロセス数が増大するように考慮する必要がある．本処理系では，vmapマクロもしくはvmap関数が評価されるときにプロセスの数が増加する．従って，多くのプロセスを生成するためには，vmapマクロ，vmap関数を十分に多くのデータに適用し，また，再帰的に適用することが必要である．

vmapマクロはコンパイル時に増加するプロセスの数が確定しているのに対して，vmap関数はコンパイル時に増加するプロセスの数がわからないものに適用できる（図３(a)）．しかし，vmap関数は実行時に引数に関する操作を逐次的に行なったりするので，vmapマクロに比べて同じベクトル長の処理に要する時間は多くなる．従って，可能な限りvmapマクロを使用するのが望ましい．

次に，我々が試みたこのような特徴を持つvmapマクロ，vmap関数の効果的な利用のためのプログラミングのいくつかの手段について述べる．

ここでは例として，それぞれのプロセスで生まれるプロセスの数がわからなくてもその数の上限がある場合について，vmap関数を使用せず，vmapマクロを用いる手段を示す．

vmapマクロは並列に処理される各プロセスにおいて増加するプロセスの数が全て等しいときにのみ使用できるのであるが，増加するプロセスの数について場合分けを行なえば，vmapマクロを使用することができる（図３(b)）．

この方法の短所としては，対象となる例題によってはベクトル長の短いものが多数にでき，高速化が失われる場合もある．

vmapマクロを用いるための別の手段として，各々のプロセスで生まれる数を等しくするために，生まれるプロセスの最大数にあわせて，ダミーのプロセスをつくることを考える（図3(c)）．これにより，ダミープロセスを処理するのに必要なオーバーヘッドが生じるが，ベクトル長の長さを損なうことなく処理を行なうことができ，高速化が期待できるであろう．

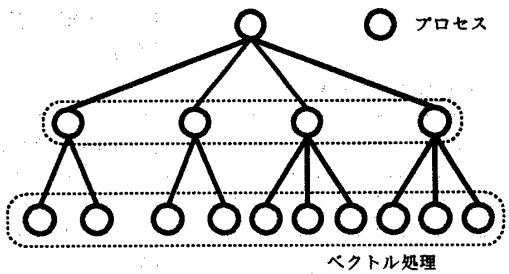

図３(a)．vmap関数の処理

A program using vmap macro and vmap function on the Vectorizing Lisp Compiler, and its evaluation.
Kazutaka KOBAYASHI, Kazuhiro ABE, Hiroshi YASUI.
OSAKA University.

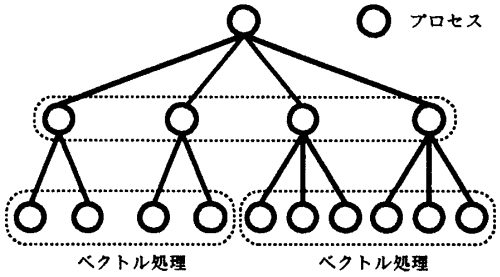

図３(b)．場合分けによるvmapマクロの処理

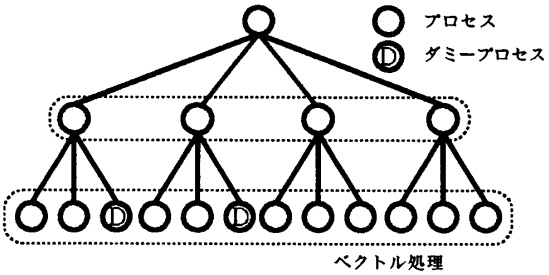

図３(c)．ダミープロセス生成によるvmapマクロの処理

本研究では，以上のようなプログラミング上の戦略を考慮し，8-puzzle問題，彩色数問題，最大クリーク問題等の例題について全解を探索するプログラム等を作成した．

4．実行結果

いくつかの例題を本コンパイラでコンパイルし，SX-2 N[4]上で実行した．ここでは一例として，増加するプロセスの数が確定しておりvmapマクロを用いて記述した8-puzzle問題と，増加するプロセスの数がわからない問題としてのvmap関数を用いた彩色数問題のそれぞれについて，ベクトル実行した場合に要した時間Tvと，スカラ実行した場合に要した時間Tnvを表１に示す．また，ベクトル長の最大値（プロセスの最大数）Pmaxと作業用領域で最も使用量の多い作業用スタックｆｓとｖｓの最大の使用量を表２に示す．

表１．実行時間

・8-puzzle問題

depth	Tv /ms (β%)	Tnv /ms	Tnv/Tv
4	10.09 (80.2)	11.76	1.16
8	76.84 (85.1)	110.45	1.44
12	622.99 (86.1)	966.27	1.55

depth:探索する解の深さ
β:ベクトル化率

・彩色数問題(彩色数 3)

node数	Tv /ms (β%)	Tnv /ms	Tnv/Tv
16	41.40 (87.9)	67.14	1.62
24	254.96 (94.1)	694.27	2.72

表２．最大ベクトル長とｆｓ，ｖｓの最大使用量

・8-puzzle問題

depth	Pmax	fs /word	vs /word
4	60	1731	221
8	540	15795	3380
12	4756	140595	29928

・彩色数問題(彩色数 3)

node数	Pmax	fs /word	vs /word
16	48	5784	1107
24	384	49608	9685

5．おわりに

本研究では，グラフ問題，その他探索問題のいくつかについて，vmapマクロ，vmap関数を用いてベクトル化Ｌｉｓｐコンパイラ向きのプログラムを作成し，実行した．8-puzzle問題を除くほとんどのプログラムでは，増加するプロセスの数が実行時でなければわからないので，主にvmap関数を用いて記述した．

vmap関数はその使用により，vmapマクロに比べ汎用的なプログラミングが可能になるが，並列に実行するプロセスの数を増やすのにコストがかかる．従って，vmapマクロよりも処理速度の低下が予想される．しかしながら，実行結果にもあるように処理の高速化が達成されていることがわかった．

8-puzzl問題はvmapマクロを用いて記述したが，1.5倍程度の高速化にとどまっている．これは途中のプロセスの数の増えかたが少ないためであり，オーバーヘッドの影響が大きく支配しているものと考える．

また，表２より8-puzzleの解の深さ（駒の移動回数）12程度の解でｆｓに140KWもの領域を使用している．スタックとして使用されるｆｓの使用領域が探索する解の深さとともに激増することから，駒の移動回数が多くなる解を探索する際には，時間と使用領域とのトレードオフ等のアルゴリズム上の戦略を考えなければならない．

［参考文献］

(1)阿部一裕 他:スーパコンピュータのためのベクトル化Lispコンパイラ，情処，記号処理54-2(1990)．

(2)阿部一裕 他:スーパコンピュータのための並列Lispコンパイラ，情処第40回全大1G-8(1990)．

(3)A.V.Aho, J.E.Hopcroft and J.D.Ullman:The Design and Analysis of Computer Algorithms, Addison-Welsley,Reading Mass.

(4)NECｽｰﾊﾟｰｺﾝﾋﾟｭｰﾀ SX-2/SX-1/SX-1E ｼｽﾃﾑ概説書 GAZ 01-3 NEC 日本電気株式会社