1G－8　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

阿部一裕　安井裕
（大阪大学・工学部）

1．はじめに

これまでに発表されてきた，スーパコンピュータ上の関数型言語処理系[1),2)]では，配列の各要素に算術演算，論理演算などをおこなう関数をベクトル演算の対象としていた．しかしLispの処理中で主になされるのは，関数呼び出し，リスト操作，条件分岐であり，それらの方式を用いても十分な高速化は得られない．

本稿では，より多くの種類の関数を，ベクトル演算機能を用いて並列処理する（ベクトル化する）ための，プロセス制御の方法とフレーム・スタックの構造について述べる．この方式により，リスト操作関数car,cdr,cons,rplaca,rplacdがベクトル化でき，条件分岐によって複雑な制御をする関数もベクトル化するようにコンパイルできるようになった．

2．並列処理の対象

本研究では，複数の引数リストに同一の関数をapplyする部分を並列処理の対象としている．こ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とになる．

式が評価される過程をプロセス，並列に実行される式の数のことをプロセスの数と呼ぶ．

プロセスの数を増やす形式としてmap関数を拡張したvmapマクロと呼ぶ形式を導入した．これ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，ユーザ定義関数であってもよい．ただし引数の数がきま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，

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

のように展開される．ただし評価の順番は，arg-ijを左から右に順に評価した後，N個の<arglist>に<fnα>を並列にapplyする．このときプロセスの数は，"applyする前のプロセスの数"×N個に増える．

このようにvmapマクロ形式を評価するたびにプロセスの数が次々と増えていく．

関数fnα本体と，そこから呼び出される関数はすべて並列に実行される．したがって，すべての関数を，複数のプロセスが並列に実行できるようにコンパイルする必要がある．

3．ベクトル化の方法

複数のプロセスを並列に実行する際に，ベクトル演算の対象としているのは，スタックに引数を積む操作と，リスト操作である．

これらの操作と，条件分岐の時のプロセスの制御の方法を例をあげて説明する．

前節で示したvmapマクロ形式により，N個の引数リストに関数fnαを並列にapplyし，N個のプロセスが生成される場合を考える．fnαの定義は次式であるとする．

```
(defun  fnα (x y)
     (cond ((listp x) (car x))
           ( <test>  <consequent>)
             ····))
```

引数の値と作業用領域のためのフレーム（図１）を各プロセスごとにフレームスタックfs上に与える．

フレームスタック（図２）は，条件分岐のときにプロセスを制御するためのマスクとしても使用される．他に各プロセスの環境を保存するものとして，値を返す場所（vp）を保持しているvalueポインタスタックvsがある．また，関数を呼び出すときには，fs,vsの先頭を示すポインタとプロセスの数を渡す．

関数listpはcond式の最初の節のtest部なので，全てのプロセスから呼び出さる．このときのfs,vsは，図２の

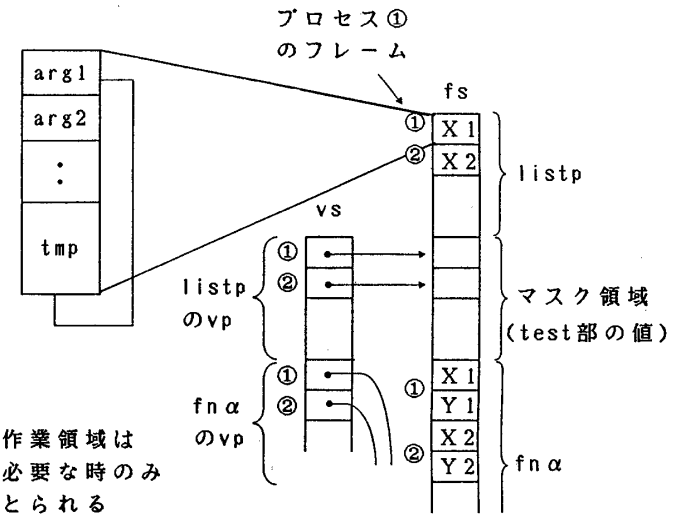

図１ フレーム　図２ フレーム・スタック

The Parallel Lisp Compiler for Supercomputer.
Kazuhiro ABE and Hiroshi YASUI.
OSAKA University.

ようになるが，listpの場合は引数の数がひとつで，また作業用領域を必要としないのでフレームはargひとつ分の大きさである．

各プロセスのフレームはfs上に連続して並んでいるので，引数の値をロード・ストアする操作はベクトル処理できる．また，test部の値を返す場所（マスク領域）はfs上に連続してとられるので，vpをvsにストアする操作もベクトル処理できる．マスク領域は，節のconsequent部を評価するときにマスクとして使用する．

consequent部の式を評価するときは，対応するマスクがnil,endでないプロセスのみ評価をおこない，マスクの値をendにする．他のプロセスは，その式の評価が終わるまでサスペンドされる．consequent部が値を返す場所は，fnαが値を返す場所になるので，fnαのvpをconsequent部のvsにロード・ストアする操作もベクトル処理できる．

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され他のプロセスはサスペンドされる．

(fn1 (fn2 x))のように式の引数が式である場合は，まずfn1の式のためのフレームをfs上にとる．次にこのフレームに値を返すようにvsをセットしてfn2の式を評価する（図３）．fn2の式を評価し終わった時点で引数の値がそろうのでfn1を呼び出す．

次にリスト操作にベクトル演算を適用する例を示す．

関数carを呼び出したとき，fs，vsは図４のようになる．間接指標ベクトルのロード，ストアもベクトル処理できるので，fsの指すリストセルのcar部のポインタをロードし，vsの指す場所にストアする操作がベクトル処理できる．cdr,rplacr,rplacdなどのリスト操作関数も同様にベクトル演算できる．

ガーベッジコレクションはRobsonの圧縮型コピー方式アルゴリズム[3]をもちいておりフリーセルは常に連続した領域に存在する．そのため関数consの操作（図５）は，第１引数の値をロードしcar部にストアすること，第２引数の値をロードしcdr部にストアすること，セルのポインタをvsのさすfsにストアすることで完了し，それぞれの操作がベクトル演算できる．

また,逐次的に処理する場合に比べ関数をcall returnする回数が減るので，この面からも高速化が得られる．

４，コンパイラの概要

本研究で使用したSX-2Nでは，FORTRANのみしかユーザに解放されていていなかったので，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．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を結合してSX-2Nに転送する．

５，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上でベクトル実行した結果Tv，スカラ実行した結果Tnvを表１に示す．比較のため，逐次実行するようにFORTRANで記述したものの実行結果Tsも併記しておく．Tarai-5,Tak-18-12-6,List-Tarai-4[4]はvmapマクロ形式を用いた形にプログラムを書換えコンパイルした．

６，まとめ

本研究の方式により，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ことを示せた．また実行結果より，処理の多くの部分でベクトル演算機能を利用し，高速化が得られることがわかった．

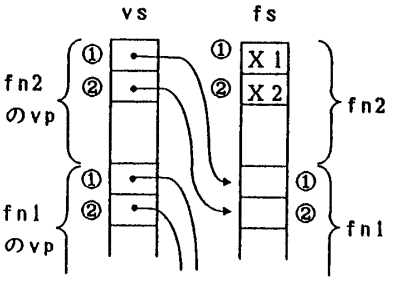

図３ (fn1 (fn2 x))の処理

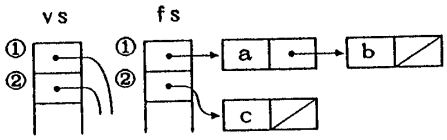

図４ carのベクトル処理

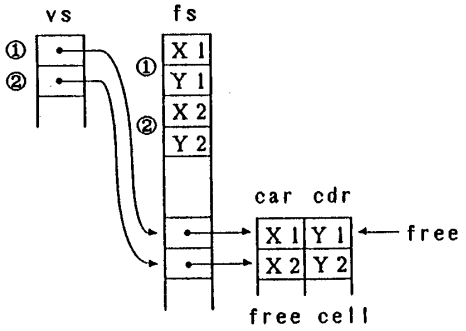

図５ consのベクトル処理

表１ 実行結果

	Tv/ms	(β/%)	Tnv/ms	Ts/ms
8-Queen	39	(99.7)	336	613
Hanoi-10	8	(99.7)	15	16
Tarai-5	99	(99.3)	713	678
Tak-18-12-6	28	(98.8)	164	346
List-Tarai-4	145	(86.2)	204	338

Tv :ベクトル実行時間，Tnv:スカラー実行時間
Ts :逐次実行時間，　β :ベクトル化率

［参考文献］

（１）植松 尚士 他:ＬＩＳＰからのベクトルプロセッサの利用，本大会．

（２）島崎 真昭:ベクトル計算機上のＦＰ型言語の処理系，信学技報，CPSY89-21(1989)．

（３）J.M.Robson:A Bounded Storage Algorithm for Copying Cyclic Structure,Comm.ACM,Vol.20,pp.431-433(1977)．

（４）奥野 博:第３回LISPコンテスト及び第１回Prologコンテスト報告，情処，記号処理資料33-4(1985)．