記 号 処 理　42－2
（１９８７．９．２５）

Ｐａｒａｌｌｅｌ Ｌｉｓｐ Ｃｏｍｐｉｌｅｒ　ＰＬＣ

中川博満
松下電器産業株式会社

近年、従来からあるノイマン型コンピューターのボトル・ネックを解消し、飛躍的なスピード性能を得る新しいテクノロジィとして並列処理（Ｐａｒａｌｌｅｌｉｓｍ）が注目を集めている。特に高性能マイクロ・コンピューターを多数結合した並列コンピューターはそのコスト／パフォーマンスや信頼性の点に於いて充分商業ベースに適うものとなり非常に多くの機種が商品化されるに至った。しかしながらこれらの並列コンピューターを動かす為のソフトウエア・ツールとしては僅かに、数値演算を並列に実行し高速化する為のＦＯＲＴＲＡＮコンパイラがサポートされている程度である。

我々は今回、このような市販の並列コンピューター上で実際にコンパイルし並列実行できるＬＩＳＰコンパイラを試作し、その性能評価を行なった。

Parallel Lisp Compiler PLC

Hiromitsu NAKAGAWA
Matsushita Electric Industrial Co.,Ltd.
3-15, Yagumo-Nakamachi
Moriguchi, Osaka, 570 Japan

Recently, "Parallel Processing" has attracted much attention, because computers based on parallel architectures can achieve great improvements in processing speeds, breaking through the "bottle-neck" of the Von Neumann type computer.

In particular, some computers designed with many tightly-coupled, high performance micro-processors have been designed and have achieved broad commercial success because of their excellent price/performance, scalability, and reliability.

However, with regard to software tools which can automatically parallelize single thread computational programs to achive accelerated processing on parallel systems, only automatic Fortran compilers have been offered up to now.

With this in mind, we have developed a Lisp compiler which can compile ordinary Lisp source code to run on these new commercial parallel architecture computers.

1．はじめに

近年、従来からあるノイマン型コンピューターのボトル・ネックを解消し、飛躍的なスピード性能を得る新しいテクノロジィとして並列処理（Ｐａｒａｌｌｅｌｉｓｍ）が注目を集めている。特に高性能マイクロ・コンピューターを多数結合した並列コンピューターはそのコスト／パフォーマンスや信頼性の点に於いて充分商業ベースに適うものとなり非常に多くの機種が商品化されるに至った（参考文献［１］）。　これらの並列コンピューターはそのハードウエア的なアーキテクチュアや結合形態（ｔｏｐｏｌｏｇｙ）が非常に種々雑多であるが大まかには　複数のＣＰＵが１本のＢＵＳを介して共有メモリーにアクセスするもの（ＵＭＡ／Uniform Memory Access Multiprocessor）と複数のＣＰＵはそれぞれ専用メモリーを持ち各ＣＰＵ間は互いに通信回線によってデータを交換しあうもの（ＮＵＭＡ／Non Uniform Memory Access Multiprocessor）に分けられる。
いずれに於いてもこれらのハードウエアをどのようなアプリケーションに対して用いるか、又　その時どのようにしてソフトウエアを作成するかは最も重要な課題である。

しかしながらいずれのタイプの並列コンピューターに於いてもこのハードウエアを動かす為のソフトウエア・ツールとしては僅かに従来よりベクトル・プロセッサやアレイ・プロセッサで培われた技術を用いて開発された、数値演算の繰り返しを並列に実行し高速化するＦＯＲＴＲＡＮコンパイラがサポートされているのみであった。

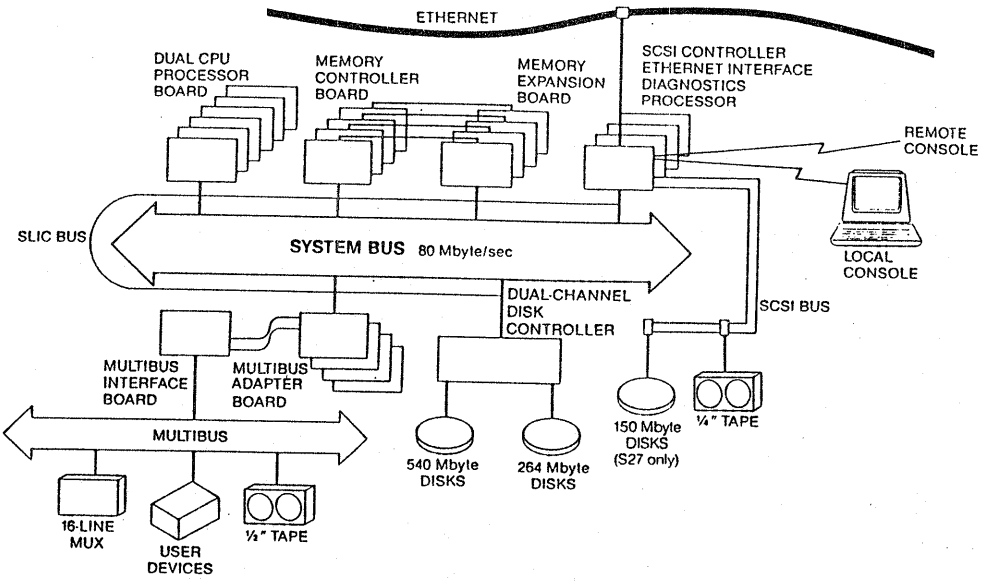
我々は今回、このような市販の並列コンピューターで或る種のアーキテクチュアと結合形態を持ったものに対し、その上で実際にコンパイルし並列実行できるＬＩＳＰｺﾝﾊﾟｲﾗを試作し、性能評価を行なった。

現在、コンピューターの高速性を強く要求している分野の１つがＡＩ（Artificial Intelligence）であり、並列コンピューターのハードウエア資源をフルに活用し　尚、且つアルゴリズムの記述も容易にできるＬＩＳＰを開発することは充分に価値のあることだと考える。

2．対象としたハードウエア構造とメモリー・セルの構成

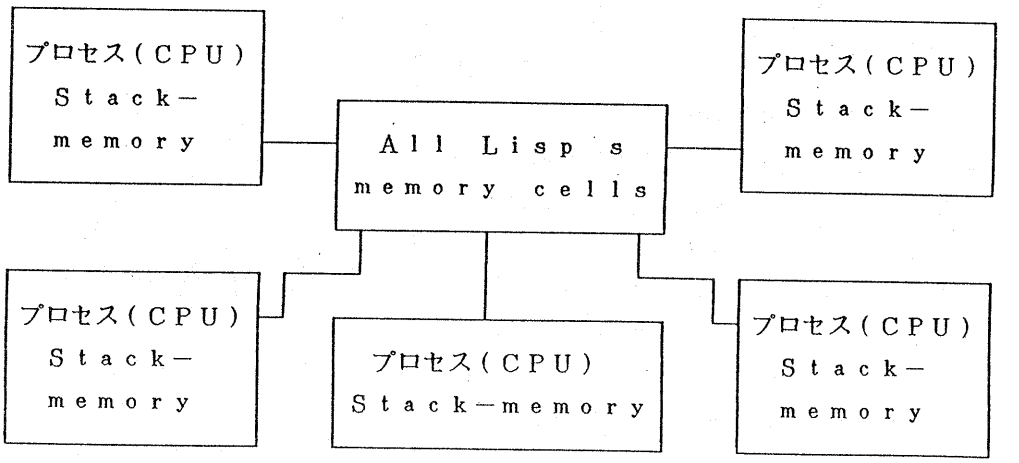
ＰＬＣ(Parallel Lisp Compiler)が動作できる為のハードウエア上の条件として大きな共有メモリーを有することがある。つまり上記分類の中ではＵＭＡでなくてはいけない。この条件はＬＩＳＰが扱う全メモリー・セルを共有メモリー上に置き、各々の関数を評価する全てのＣＰＵがこのメモリー・セルを参照することに起因している。

具体的にはＳｅｑｕｅｎｔ社製のＢａｌａｎｃｅ／Ｓｙｍｍｅｔｒｙシステムを用いた。これは最大３０個のＣＰＵと　２４０MbyteのＲＡＭ、及び各種Ｉ／Ｏが１つのＢＵＳに接続された並列処理コンピューターである（参考文献［２］）。　このコンピューター・システムのハードウエア・ブロック構成を第１図に示す。



第１図

このコンピューターでは全ＲＡＭはハードウエア的には全てのＣＰＵからアクセスできる。このＲＡＭをＯＳ／ＵＮＩＸがユーザー・プログラムの要求に応じて共有メモリーとローカル・メモリーに割り当てる。更に　ＯＳ／ＵＮＩＸはユーザー・プログラムの実行により生成された複数個のプロセスをそれぞれどれかのＣＰＵに割り当てて処理させる。この時、あるプロセスの処理を割り当てるＣＰＵとしてその時点で最も負荷の軽いＣＰＵをＯＳ／ＵＮＩＸが自動的に選択する。ユーザー・プログラムの中から新たにプロセスを生成するには通常のＵＮＩＸと同様にシステム・コールのｆｏｒｋ（）とｅｘｅｃ（）を用いる。このようにして生成された新しいプロセスをどのＣＰＵによって処理させるかはユーザー・プログラム側からは制御できない。



第２図

ＰＬＣ（Parallel Lisp Compiler）が生成するプログラムは全メモリーの内、数１００ＫＢＹＴＥ～数ＭＢＹＴＥを６４ＢＩＴ／１セルから成るＬＩＳＰメモリーセルとして用いる。更に　このＬＩＳＰメモリーセルは全てのプロセスで共有する旨、ＯＳに対して宣言する。ＬＩＳＰ関数（Ｓ式）の評価で用いるＬＩＳＰメモリーセル以外の作業領域は各プロセス単位に作られる。

ＰＬＣにより翻訳して得られるオブジェクト・プログラムの全プロセスはこのＬＩＳＰメモリー・セル中にあるデーターを共有し、これを各プロセス毎に固有の作業領域（主にスタック・メモリー）を使って処理する。この様子を第２図に示す。

３．並列処理の起動

ＰＬＣは通常のＬＩＳＰ組み込み関数の他に特殊な組み込み関数としてｃｏ＿ｅｖａｌを有する。　ＰＬＣによって翻訳されたオブジェクト・プログラムの中でｃｏ＿ｅｖａｌ関数が評価されるとＯＳ／ＵＮＩＸに対してシステム・コールのｆｏｒｋ（）が呼ばれ、ｃｏ＿ｅｖａｌ関数の各引数を評価する為の子プロセスが生成される。この各子プロセスがその時点で最も負荷の軽いＣＰＵに１つずつ割り当てられて引数の評価を行なう。

ｃｏ＿ｅｖａｌ関数は入れ子にすることもできる。

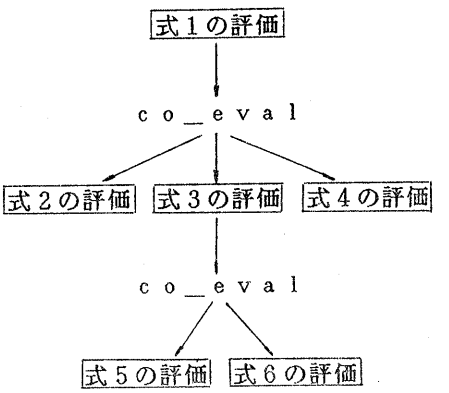
例えば　第３図のような構造の式を評価すると第４図のような子（孫）プロセスが生成され、それぞれのｃｏ＿ｅｖａｌ関数の引数である式を評価する。

```
（式１　（ｃｏ＿ｅｖａｌ　（式２　　　　　　）
                          （式３　（ｃｏ＿ｅｖａｌ　（式５　）
                                                    （式６　）
                                  ）
                          ）
                          （式４　　　　　　）
        ）
）
```

第３図

即ち、　通常は逐次実行される　式２，式３，式４の評価、　及び　式５と式６の評価が間に１段ｃｏ＿ｅｖａｌ関数を挿入することにより並列実行されることになる。

子プロセスは与えられた式の評価が終わるとｅｘｉｔ（）のシステム関数を呼び、自動的に消滅する。何個（何重）の子（孫）プロセスを生成することも可能であるが最適な数はアプリケーションの立場から　プログラムを作る者が判断しなくてはならない。



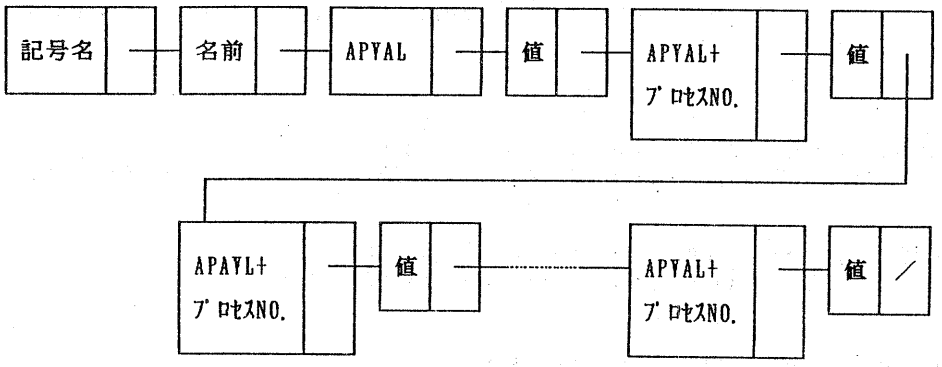
第４図

4．変数値の保持と授受

ＰＬＣでは変数の値は基本的に属性リストとして保持している。しかし高速化を図る為評価中の関数の局所変数で可能なものはＣ言語の局所変数としてスタック又はレジスターに保持することもある。ｃｏ＿ｅｖａｌが実行されると属性リストの　特に、ＡＰＶＡＬ属性に代わり「ＡＰＶＡＬ＋プロセス番号」の属性名が新たに使われる。

具体的にはｃｏ＿ｅｖａｌ関数により生成された子プロセスに於いてｓｅｔｑ等の代入操作を行なうと、この新しい属性名（APVAL+ ﾌﾟﾛｾｽ番号）の値として記憶される。

このようにすることで　１つの変数にそれぞれの子プロセスが重複すること無く、値を保持することができる。　この時の属性リストの状態を第５図に示す。

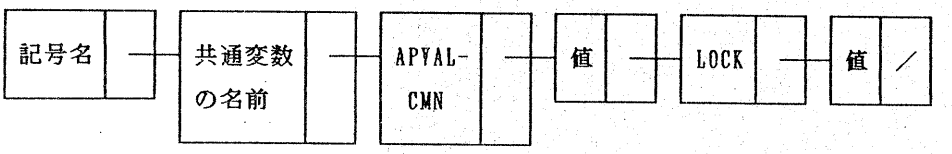


第５図

又、変数値が参照された時は自分のプロセス番号を含むＡＰＶＡＬ属性名があればその値を使い、無ければ親プロセスのプロセス番号を含むＡＰＶＡＬ属性を探す。これにより子（孫）プロセスは親（祖母）プロセスの値を受け継ぐことになる。逆に、子プロセスでのＡＰＶＡＬ属性値は親プロセスに引き継がれ無い。　子プロセスでの値を親プロセスに渡す、或いは兄弟プロセスで値を共有するには共通変数を使わなければならない。共通変数はｃｏｍｍｏｎ関数の引数にして、（ｃｏｍｍｏｎ　共通変数名）　と言う関数を評価することで設定できる。　共通変数にはＡＰＶＡＬ－ＣＭＮと言う属性名が作られ、この属性を有する変数に対しては全プロセスがこの属性値に対して代入と参照を行なう。

更に、共通変数にはＬＯＣＫと言う属性名が作られる。　この属性値は一時的に他のプロセスが共通変数をアクセスすることを禁止する２進セマフォとなる。

このセマフォをロックする関数が　（ｌｏｃｋ　共通変数名）、　リリースする関数が（ｕｎｌｏｃｋ　共通変数名）　である。共通変数の属性リストの状態を第６図に示す。



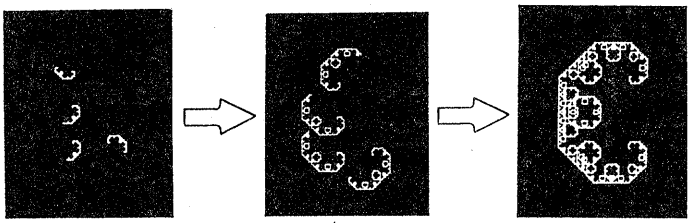
第６図

5．プログラム例

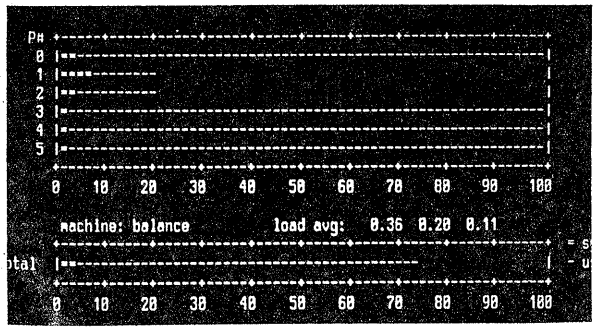
（１）C曲線を描く

```
(defun  c_curve  (length  angle  start_x  start_y  depth)
  (cond ((lessp length  MIN_LENGTH)
         (c_curve_plot  length  angle  start_x  start_y)
        )
        ((lessp depth CO_EVAL_DEPTH)
         (co_eval
            (c_curve_right  length  angle  start_x  start_y  (1+ depth))
            (c_curve_left   length  angle  start_x  start_y  (1+ depth))
         )
        )
        (t
            (c_curve_right  length  angle  start_x  start_y  (1+ depth))
            (c_curve_left   length  angle  start_x  start_y  (1+ depth))
        )
  )
)

(defun  c_curve_plot  (len  ang  st_x  st_y)
   (line st_x st_y (+ st_x (* len (cos ang))) (+ st_y (* len (sin ang))))
)

(defun  c_curve_right  (le  an  s_x  s_y  de)
   (c_curve  (/ le (sqrt 2))  (+ an (/ PI 4))  s_x  s_y  de)
)

(defun  c_curve_left   (le  an  s_x  s_y  de)
   (c_curve  (/  le  (sqrt 2))  (- an (/ PI 4))
             (+ s_x (* (/ le (sqrt 2)) (cos (+ an (/ PI 4)))))
             (+ s_y (* (/ le (sqrt 2)) (sin (+ an (/ PI 4)))))  de)
)

(c_curve  100  (/ PI 2)  0  -50  0)
```

第７図



第８図



第９図

第７図はＰＬＣによりＣ曲線（第８図、右端）を描くプログラムである。第８図はこのプログラムを実行中のグラフィック・ディスプレイ画面の状態遷移を時系列的に示したものである。第９図はやはりこのプログラムを実行中の各ＣＰＵの負荷状況を示すシステム・コンソール画面である。

関数ｃ＿ｃｕｒｖｅは描こうとする線分の長さが或る値（ＭＩＮ＿ＬＥＮＧＴＨ）以上ならばその線分をｃ＿ｃｕｒｖｅ＿ｒｉｇｈｔとｃ＿ｃｕｒｖｅ＿ｌｅｆｔの２本の線分に分割し、再度自分自信を呼び出す。その値よりも短ければｃ＿ｃｕｒｖｅ＿ｐｌｏｔを呼び直接線分を描く。この様にして再帰の深さが或る値（ＣＯ＿ＥＶＡＬ＿ＤＥＰＴＨ）になるまでは、ｃｏ＿ｅｖａｌ関数を経由してｃ＿ｃｕｒｖｅ＿ｒｉｇｈｔとｃ＿ｃｕｒｖｅ＿ｌｅｆｔを呼び出すことにより　分割された２つの線分を同時に並列的に描く。
一方、再帰の深さがこの値を超えた時はｃ＿ｃｕｒｖｅ＿ｒｉｇｈｔとｃ＿ｃｕｒｖｅ＿ｌｅｆｔを通常どうり逐次的に呼び出し　順次この線分を描いて行く。　第７図のプログラム例では再帰の深さ２までｃｏ＿ｅｖａｌすることで４プロセスを生成している。
第８図の写真では１つのＣ曲線が４つに分割されて並列的に描画される様子がよくわかる。
第９図システム・コンソール画面の写真では全６ＣＰＵ（Ｐ＃０～５）の内、４ＣＰＵ（Ｐ＃０，３，４，５）が１００％の負荷を負ってこの線分の描画を担当している様子が示されている。

（２）８－ＱＵＥＥＮの全解を求める

```
(common r1 r2 r3 r4 r5 r6 r7 r8)

(defun qu (n b)
   (cond ((zerop n) nil)
         ((or (member n b) (qp 1 b)) (qu (sub1 n) b))
         (t (nconc  (cond ((eq 7 (length b)) (list (cons n b)))
                          (t                 (qu 8 (cons n b))) )
                    (qu (sub1 n) b) )) ) )
(defun qp (k m)
  (cond ((null m) nil)
        ((eq k (abs (- n (car m)))) t)
        (t (qp (add1 k) (cdr m))) ) )

(progn () (time
  (co_eval (setq r1 (qu 8 '(1)))
           (setq r2 (qu 8 '(2)))
           (setq r3 (qu 8 '(3)))
           (setq r4 (qu 8 '(4)))
           (setq r5 (qu 8 '(5)))
           (setq r6 (qu 8 '(6)))
           (setq r7 (qu 8 '(7)))
           (setq r8 (qu 8 '(8)))
  )
  (setq r (append r1 r2 r3 r4 r5 r6 r7 r8))
))
```

第１０図

第１０図は８－ＱＵＥＥＮの全解を求めるプログラムをＰＬＣで書いたものである。８ＱＵＥＥＮ問題を８個の７ＱＵＥＥＮ問題に分割し、各々解を求め、それを共通変数ｒ１～ｒ８に代入し、全部求まれば　それらを結合して１つのリストを作る。

従って　このプログラム例では８個のプロセスが並列に実行される。

第１１図にこのプログラムをＳＥＱＵＥＮＴ社製パラレル・コンピューター／Ｂ８Ｋシステム上で実行した時の所要時間を示す。

８－ＱＵＥＥＮの全解を求めるのに要する時間		
条件　：　ハードウエア　・・　ＳＥＱＵＥＮＴ／Ｂ８Ｋ ＯＳ　・・　ｄｙｎｉｘ　（ｕｎｉｘ　４．２ｂｓｄ） 言語　・・　ＰＬＣ ＭＰＵ　・・　ＮＳ３２０３２（10MHz）　＊　６個 ﾒｲﾝ･ﾒﾓﾘｰ　・・　８ＭＢ		
インター・プリター	ＵＮＩ－ＰＲＯＣＥＳＳＯＲ	４８０秒
	４　－ＰＲＯＣＥＳＳＯＲ	１２０秒
コンパイラ	ＵＮＩ－ＰＲＯＣＥＳＳＯＲ	２８秒
	４　－ＰＲＯＣＥＳＳＯＲ	１３秒

第１１図

６．まとめ

今後、ＰＬＣの改良として人間はなるべく並列性を意識しなくとも機械が自動的に並列実行型プログラムに展開できるようにすること、或いは並列処理記述部をより人間にわかりやすい形で表現できるようにすることが考えられる。

更に大きな課題として、　例えばキューブ型の並列コンピューターのような、より疎な結合で巨大な　マルチプロセッサー・システムのようなものにもＰＬＣをインプリメントしたいと考えている。

＜参考文献＞

[1] 期待と不安を乗せて飛び立った並列処理コンピュータ　日経ｴﾚｸﾄﾛﾆｸｽ 1986.11.3

[2] Balance(TM)8000 System Technical Summary Sequent Computer Systems, Inc