ソフトウェア基礎論19－7
プログラミング言語 9－7
（1986. 12. 12）

並列コンピューターの為のプログラミング技法

Programming for parallel computer

プログラミング言語PARAC と 逐次プログラムの自動並列化

Programming language PARAC and Auto-Parallelization for serial program

中川博満
Hiromitsu NAKAGAWA
松下電器産業（株） デバイス開発研究所
Matsushita Electric Industrial Co., Ltd.
Advanced Devices Laboratory

あらまし

現在、非常に多くの並列処理コンピューターが開発されている。 これら並列処理コンピューターはハードウエア的には優れたアーキテクチュアを持ち高いパフォーマンスを得られるものである。しかしこのハードウエアを動かすプログラムの開発は大変むつかしい。 特に 多くの互いに干渉しあいながら処理を進めるプログラムを同時に何本も並行して実行できるコンピューターの能力を最大限に発揮するように与えられた問題をプログラム化する事は超熟練した並列プログラマーにとっても非常にむつかしい課題である。 今の並列コンピューター開発に於いて最大の問題はこの並列コンピューターの能力を最大限発揮できるプログラムをいかに作り易くする環境を提供できるかにある。

我々は並列処理プログラムを容易に作成する為の言語としてC言語を拡張したPARACを開発した。更に、 通常の逐次処理用であるC言語で作成したプログラムを並列コンピューターのハードウエア・リソースに割り当て同時並行処理するプログラムに自動的に変換するエキスパートシステムを開発した。

Abstract

Many parallel computers have been developed with excellent architectures that are getting high performance through hardware, but programming for these computers is very difficult.

To utilize the full performance potential from these computers it is necessary to execute many programs concurrently, interfacing each to the other, which is most difficult even for the skilled programmer. The most important factor for parallel computers is to develop an environment in which programmers can develop parallel programs easily. We developed PARAC, that is an extension of 'C' for parallel programming.

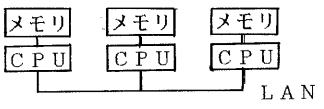
In addition, we developed an expert-system which translates programs written in normal 'C' language into PARAC language to be executed concurrently automatically assigning the programs to each hardware resource.

1．はじめに

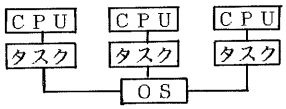
近年、デバイス技術の進展等により　コンピューターハードウエアは非常に高性能化している。それに伴いコンピューターの応用面でも　ＡＩ　（Ａｒｔｉｆｉｃｉａｌ　Ｉｎｔｅｌｌｅｇｅｎｃｅ）　や　ＣＡＤ（　Ｃｏｍｐｕｔｅｒ　Ａｉｄｅｄ　Ｄｅｓｉｇｎ　）　に代表されるように　非常に高度化し、これが更にハードウエアに高性能を要求するようになっている。　このため、　超ＬＳＩ化等による高性能化が図られているがコンピューターの性能を画期的に向上させる方法としてＣＰＵの並列化が注目を集め　最近急激に研究開発が広がった。　しかし、それには非常に多くの問題がある。　その最大のものはソフトウエアプログラムに関するものである。　つまり、数１０、数１００、或いは数１０００　のＣＰＵを並列に接続した時、そのコンピューターを動かす為のプログラムはどのようにすれば実現できるかということは　まだ未開決の領域である。　　並列処理コンピューターのハードウエアはマイクロコンピューターの低価格化と高機能化により比較的容易に種々のものが開発されるようになった。　これに対して　並列処理コンピューター用プログラムつまり　並列アルゴリズムの開発は従来の逐次処理プログラムの開発に比べて格段にむつかしい。　並列処理コンピューターはコンピューターの性能を飛躍的に向上させる可能性を持っているかわりに　そのプログラム開発は非常に難かしいという欠点を持っている。　逆に並列処理アルゴリズムが簡単に開発出来れば並列処理コンピューターの持つ無限のパフォーマンスを誰もが簡単に利用することができるようになる。　　このため我々は並列処理アルゴリズムを簡単に記述する為の言語としてＰＡＲＡＣを開発した。　　更に、従来の逐次処理用言語｀Ｃ｀で記述されたプログラムを並列処理する為、自動的にＰＡＲＡＣ語による並列プログラムに最適化して変換する一種のエキスパートシステムを開発した。

2．並列コンピューター或いは並列処理とは

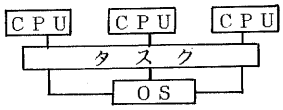
並列コンピューター或いは並列処理にはいろいろな形態が考えられる。例えば　第１図に示すように複数個のＣＰＵがネットワークで連なっており互いにデーターを通信しあいながら　それぞれ独立に処理を同時に平行して行なうものや　第２図に示すように、各ＣＰＵがそれぞれ独立したタスクを一つのＯＳの管理下で同時に平行して行なうもの等もある。このようなものもある意味では並列処理の一例といえる。　しかし　今対象としている並列処理コンピューター、　性能を飛躍的に向上できる代わりにプログラム開発が非常に難かしいのはこれ等では無く、第３図に示すように複数個のＣＰＵで一つのタスクを分担して実行し、処理の高速化を目指すものである。このような形式の並列処理コンピューターでは　一つのタスクから並列処理できる部分を抜き出し、　それをプログラムで表現できれば　このタスクを複数個のＣＰＵで高速に処理することができる。



第１図



第２図



第３図

3．並列コンピューターのプログラミング法

並列コンピューターのプログラム作成は　大きく分けて３つの方法がある。　それらを以下に示す。

（１）　従来の１ＣＰＵの為の手法で作成されたプログラムを　そのまま複数のＣＰＵで分担して　実行処理する方法である。　しかし　これには複数のＣＰＵで処理させる為の、何らかの変換手続きを得る必要がある。　この変換手続きをうまくできれば　従来の１ＣＰＵ用として蓄積されたソフトウエア資産がそのまま並列コンピューターによって高速に処理できるという非常に大きなメリットがある。

（２） プログラマが ＣＰＵの数、 各ＣＰＵに属するメモリの大きさ、 各ＣＰＵがアクセスでき 共有できるメモリの大きさと番地、 或いは、 各ＣＰＵ間の通信等を操りながら、 新たにプログラムを作成する方法がある。 この方法はプログラマに多大な負担をかけるが、 一方できたプログラムは、並列コンピュータハードウエア資産に全くｆｉｔしたものであり、 コンピューターの性能を 完全に生かしたプログラムを作成することができる。

（３） 人間のプログラマが１ＣＰＵを仮定してプログラムを作り、 それを（１）と同様な変換手法で複数のＣＰＵによって、 処理されるプログラムに変換する。 （１）では、 そのプログラムをそのまま実行していたが、ここでは、その結果を人間のプログラマに示し、 実行してよいか決断を求める。 人間のプログラマはそれを見て、 更に変更した方が良い所、 変更しない方が良い所をコンピュータに助言する。 コンピュータはそれをもとに、再度 変換手続きをやり直す。 これを何回か繰り返し実行プログラムを得る。 このような 一種のエキスパートシステムにより、並列プログラムを作成する手法が考えられる。

４．並列処理の機能要素

並列処理コンピュータには最低出来なければならない機能がある。 ここでは それについて述べる。
第２章で述べた様に ここで言っている並列コンピューターは これらの機能を実現することによって一つのタスクを複数個のＣＰＵで分担し 高速に処理することが出来るものである。

（１） 最低限必要な機能

（ａ） ｐｒｏｃｅｓｓ間通信

共有メモリー、又は通信回線を介して 並列に処理されているｐｒｏｃｅｓｓ間でデーターを送受する機能。

（２） 拡張機能

（ｂ） 同期

並列に実行されているｐｒｏｃｅｓｓ間で処理の待ち合わせをする機能。

（ｃ） 排他

一時的に共有メモリーの或る領域を 他のＣＰＵがａｃｃｅｓｓすることを禁止する機能。

（ｄ） 割り込み

或るＣＰＵから 他のＣＰＵへ割り込みをかける機能。

（３） あって当然 （ 言うまでも無い ） 機能

（ｅ） ＣＰＵ間の共有メモリー 又は 通信回線。

（ｆ） ｐｒｏｃｅｓｓを生成し、 それを或るＣＰＵに割り当てて処理させる機能。

５．ＰＡＲＡＣ とは

ＰＡＲＡＣは並列コンピューターのプログラムを作る為のプログラム言語である。 但しこのプログラムは人間が作るとは限らない。 第３章で述べたように １ＣＰＵを想定して開発されたプログラムを機械が複数ＣＰＵで処理するプログラムに変換して ＰＡＲＡＣ言語により出力することもできる。 この時、 人間はＰＡＲＡＣ言語により表現された並列プログラムを検証し 更に改良点を助言する。最終的にはコンピュータ自身が自分達で実行できる機械語にコンパイルして実行する。ＰＡＲＡＣは並列プログラムという目的の為の人間のプログラマと並列コンピューターとの間の接点（Ｉｎｔｅｒｆａｃｅ）である。よって ＰＡＲＡＣは効率的に並列プログラムが表現でき、かつ人間のプログラマにとってわかり易いことが要求される。

ＰＡＲＡＣはこれらの要求を満たすため　下記のような特徴を有する。

（１）　基本的な部分は＂Ｃ＂の構文をそのまま用いる。

（２）　並列処理機能要素を独自の構文で記述する。　（　排他、プロセス生成等　）

（３）　プログラマーは意識して”陽”に並列処理部をＰＡＲＡＣの構文を使い表現する。

この言語を用いることでプログラマーは比較的容易に　並列処理するユーザー・アプリケーション
プログラムを作成することができる。

現在、ＰＡＲＡＣは並列処理コンピューターである　Ｓｅｑｕｅｎｔ社の　Ｂａｌａｎｃｅ８０００用
コンパイラがあり　この上でコンパイルと実行ができる。

６．ＰＡＲＡＣによる並列処理機能要素の記述

（１）　共有メモリーによる通信

（ａ）　ＰＡＲＡＣでは或る変数を共有メモリーに置き　並行処理される全プロセスでこれを使うことができる。　変数を共有メモリーに置くには下のような変数の宣言をする。

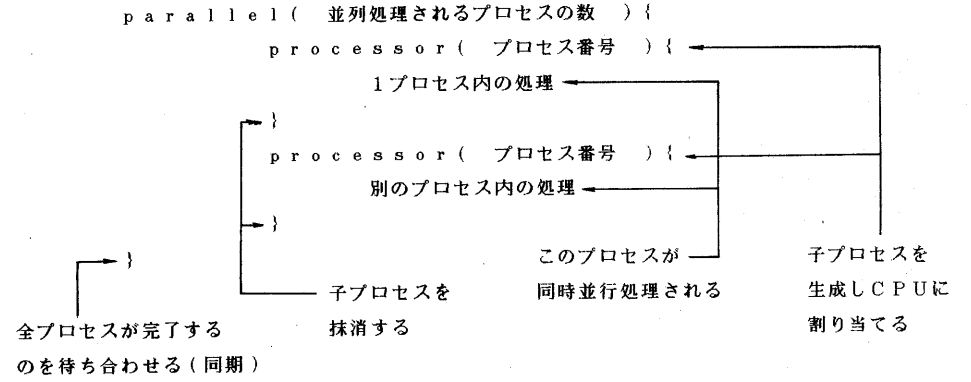
```
common、　変数の型　　変数名，・・・　；
    [例]　common　　int　　i，j，k；
          common　　char　　c［１０００］；
```

（ｂ）　この共有変数を介してプロセス間でデーターを送受することができる。
その方法を下に例で示す。

```
[例]　common　　int　　flag；
      common　　char　　a［１００］；
  （　プロセス０　）
        flag　＝　０；　　（　初期化　）
        a［i］　＝　・・・；　　（　データー・セット、送信　）
        flag　＝　１；　　（　データーセット終了、受信要求　）
  （　プロセス１　）
        while（　!flag　）；　　（　データー・セット待ち　）
        ・・・　＝　a［i］；　　（　データ取り出し、受信　）
```

（２）　プロセスの生成・抹消

（ｃ）　ＰＡＲＡＣで子プロセスを生成し　それを或るＣＰＵに割り当て　処理が終わった後に抹消させるには下のように記述する。



（３） 排他制御

（ｄ） ＰＡＲＡＣでは共有宣言した変数を一時的に 他のプロセスがアクセスすることを禁止することができ、これによりｓｅｍａｐｈｏｒｅの機能が実現される。それには下の様に記述する。

```
lock( アクセスを禁止する共有変数 ){
        アクセスを禁止した状態で行なう処理
}
```
← 排他状態の解除を示す

（４） その他

（ｅ） 同期 ・・・・ 各プロセス間の同期を共有変数を使って取ることが出来る。
その他に子プロセスの終了を自動的に待ち合わせることもできる。
（ 第（２）項（ｃ）参照 ）

（ｆ） 割り込み ・・・ 現在のバージョンでは表現できない。

（ｇ） 共有変数を介さないプロセス間通信 （ 通信回線を使うもの ）
・・・ 現在、ｓｅｎｄ／ｒｅｃｅｉｖｅ の機能を作成中である。

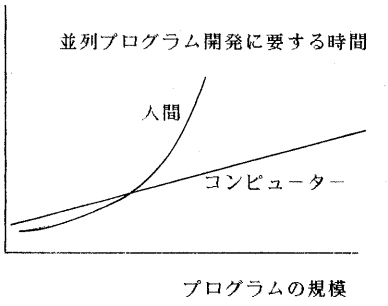
７．ＰＡＲＡＣによるプログラム例

以下にＰＡＲＡＣを使ったプログラム例を一つ示す。 これは行列の乗算を非常に多くのプロセスによって並列に同時処理するものである。 尚、このプログラムはＰＡＲＡＣによる並列処理の一つの例を示したものであって このプログラムが最適というものではない。

```
common    int    a[AI][AJ], b[AJ][AI], c[AI][AI], i, j;
mult_matrix(){
    int    k, x, n, li, lj;
    i=0;
    j=0;
    parallel(AI*AI){
        for(n=0; n<AI*AI; n++){
            processor(n){                    /*並列に処理される部分*/
                lock(i, j){
                    li=i;                    /*排他した状態で共有変数の  */
                    lj=j++;                  /*値をローカル変数へ取り込む*/
                    if(j==AI){    j=0;
                                  i++;
                    }
                }                                    /*lockを解除*/
                x=0;                                 /*実際の並列処理が始まる*/
                for(k=0; k<AJ; k++){
                    x+=(a[li][k]*b[k][lj]);
                }
                c[li][lj]=x;
            }
        }
    }
}
```

8．自動並列化エキスパート・システム　ＡＰＡＲＡＣ

ＰＡＲＡＣを使うことにより我々は比較的容易に並列処理コンピューターの為のプログラムを作成し問題を高速に解くことができるようになった。　しかしながら並列アルゴリズムの開発は　その相互排除の問題や人間の不慣れ　等により　大変難かしいものである。　そこで我々はこのプログラムを機械で自動的に作成することを考え　エキスパート・システム　ＡＰＡＲＡＣ　を開発した。　これは通常の〝Ｃ〟言語で開発され逐次処理されるプログラムを並列コンピューターにより並列処理されるプログラムに変換しＰＡＲＡＣ語表現で出力するものである。第４図に示すように人間のエキスパートは比較的小規模な問題に関しては効率の良い並列化プログラムを作成することができる。　しかし問題の規模が大きくなると人間はその並列処理アルゴリズム全体を把握しきれなくなってしまう。　しかしながら、　その時でも人間のエキスパートは何らかの　逐次プログラムを並列化することに関する知識やルールを持って作業しているはずである。



第４図

この知識やルールをコンピューターに移したエキスパート・システムを開発すれば　それは人間のエキスパートが小規模な逐次プログラムを　うまく並列化するのに用いた知識やルールを　そのまま大規模な問題にも適用して、うまく並列化できるはずである。　このような考えのもとに我々は　ＡＰＡＲＡＣ　を開発した。　ＡＰＡＲＡＣは現在、Ｂａｌａｎｃｅ８０００　や　その他のワークステイション上で動く　Ｌｉｓｐ／Ｉ　で記述されており約２０００行程度であるが　更に改良中である。

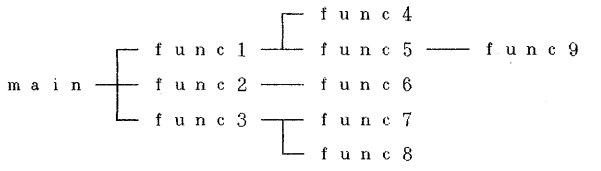
9．ＡＰＡＲＡＣの知識構造

ＡＰＡＲＡＣは大きく分けて３つの知識ベースを持っている。それは　プログラム構造知識、文属性知識、並列化ルールである。

（１）　プログラム構造知識

これは　このエキスパート・システムにより並列化される被対象プログラム全体の〝呼び出し関係〟つまり　どのサブプログラム（関数）が　どのサブプログラム（関数）を呼んで使っているか、の構造を木チャートの形で記憶している。

例えばｍａｉｎの関数がその中で子供の関数として　ｆｕｎｃ１，ｆｕｎｃ２，ｆｕｎｃ３を呼出しこのｆｕｎｃ１が　ｆｕｎｃ４とｆｕｎｃ５を呼び、更にｆｕｎｃ５が



第５図

ｆｕｎｃ９を呼び出して使用しているような　被変換プログラムが入力された時、　ＡＰＡＲＡＣは内部に第５図のような　プログラム構造知識ベースを作成する。　この知識ベースは　エキスパート・システムが　与えられたプログラムを並列化する時、プログラムの　どの部分を並列化するのが最も効率が良いか判断するのに使われる。　つまり　第５図のような例では　まず始めに　システム側でｍａｉｎ関数の中でｆｕｎｃ１，ｆｕｎｃ２，ｆｕｎｃ３を呼び出している部分を並列化するのが最善では無いかと考え　その案をプログラマに提示し　決断をもとめる。　もしプログラマがそれで良いと判断すればシステムはその部分の並列化を試みる。　もしもプログラマが　それは余り良くないと判断すればシステムは他の部分、例えばｆｕｎｃ１の中の並列化を提案する。

（２）　文属性知識

これも　このエキスパート・システムにより並列化される被対象プログラムを元にした知識ベースでこのプログラムの中に含まれる　各一文一文を解析し　その属性を取り出して　Ｌｉｓｐのｆｒａｍｅの形で記憶しているものである。ここで言う各文の属性には　この文で行なわれる操作（ｏｐｅｒａｔｉｏｎ）、この文で参照される変数（ｒｅｆｅｒｅｎｃｅｄ　ｖａｌｉａｂｌｅ）、　この文で代入される変数（ｓｅｔ　ｖａｌｉａｂｌｅ）、この文が関数呼び出しを行なう時の呼び出される関数（ｓｕｂ－ｆｕｎｃｔｉｏｎ）、この文が宣言文の時の宣言型（ｄｅｆｉｎｉｔｉｏｎ）、この文が条件判断を伴う時の条件（ｊｕｄｇｅ　ｃｏｎｄｉｔｉｏｎ）、条件成立時処理（ｔｒｕｅ　ｐａｒｔ）、条件不成立時処理（ｆａｌｓｅ　ｐａｒｔ）、この文の本体（ｂｏｄｙ）、それに　この文が実行されるのに要する時間（ｒｕｎ　ｔｉｍｅ）、この文を処理するＣＰＵ（ｐｒｏｃｅｓｓｏｒ）等がある。

エキスパート・システムに自動並列化の対象となる被変換プログラムが入力されると　当システムはこのｆｒａｍｅ構造の　文属性知識ベースを、その中に作り　保存する。この知識ベースが並列化の基本となる。　こうして作られた知識ベースは　次の第（３）節で説明する　並列化ルールに適応して、各文が並列化可能かどうかの判断に使われる。下に示すような文例が与えられた時に作られる文属性知識ベースのｆｒａｍｅ構造を例として第６図に示す。

（文例）

```
while(a!=10){
        x=y;
        d=c;
        a++;
}
```

facet名	value

body	(while(a!=10){
	x=y;
	d=c;
	a++; }
	)

operation	(while)
ref. value	(a y c)
set. value	(x d a)
function	nil
definition	nil
judge cond	(a!=10)

true part	(x=y;
	d=c;
	a++;
	)

false part	next sentence
run time	nil
processor	nil

第６図

（３）　並列化ルール

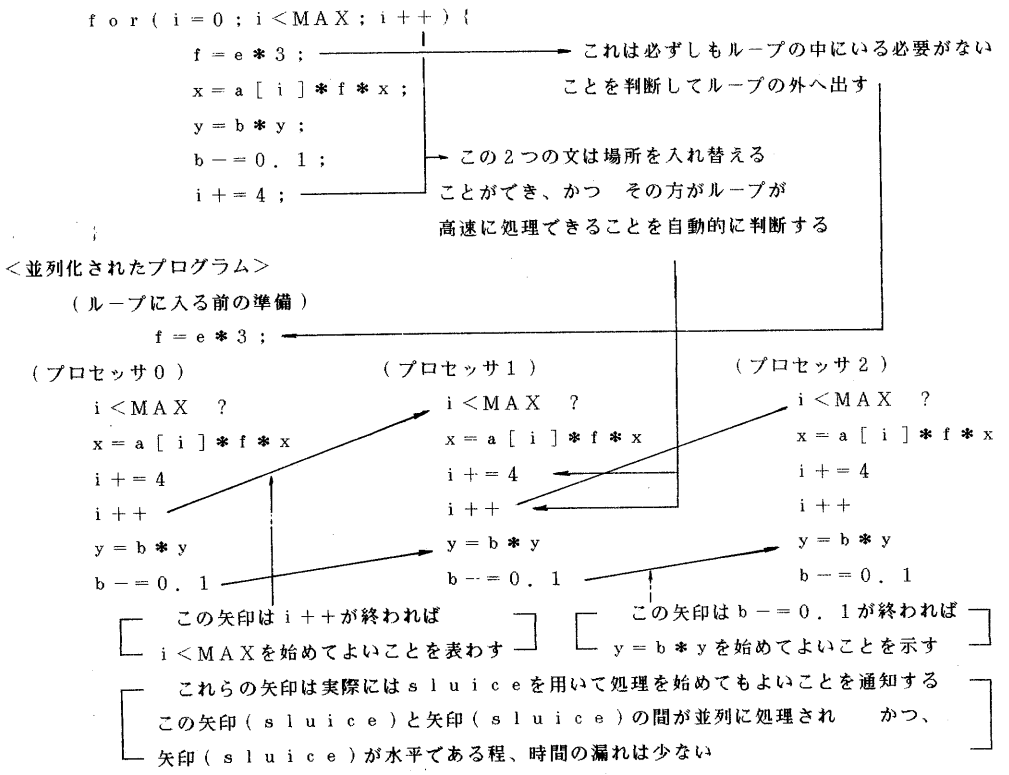
今までに述べた２つの知識ベースが共に　入力された被変換プログラムから作られていたのに対し　これは始めからこのエキスパート・システムの中に存在する　知識ベースで　変換の対象となるプログラムには依存しないものであり、　これは　プロダクション・ルールの形で　記憶されている。　前述したように当システムは　このルールを、入力された被変換プログラムから作ったｆｒａｍｅ構造の文属性知識ベースに適応して　各文が並列に処理できるかどうか判断し　ＰＡＲＡＣによる並列プログラムに変換してゆく。　この並列化ルールの代表的なものを　例を示しながら以下に述べる。

（ａ）　２つの文　ｓ１とｓ２が有る時、この２つの文の　文属性知識ベース中の　”参照される変数”のリストと”代入される変数”のリストに全く共通要素が無ければこの２つの文は並列に処理できる。例えばａ＝ｂ＋ｃ；とｄ＝ｅ＊ｆ；は並列に処理できる。　これは最も簡単な並列化ルールである。

（ｂ） 新たな文ｓ１が今までの処理結果に全く依存せず、 これ以前の文とは完全に並列に ＣＰＵ０～ＣＰＵｎ のどれで実行してもよい時、 今までの文の処理に要する時間の最も短いＣＰＵに この文ｓ１の処理は割り当てる。 これも理由は自明の大変簡単なルールである。

（ｃ） 新たな文ｓ１がＣＰＵ０とＣＰＵ１の処理結果に依存する時、この文ｓ１の処理は ＣＰＵ０かＣＰＵ１のうち今までの文の処理に要する時間の長い方に割り当てる。そして両方に共通する変数をｃｏｍｍｏｎ宣言する。 更に ｆｌａｇの一種であるｓｌｕｉｃｅと言う 新しい概念を導入して文ｓ１の処理を割り当てなかったＣＰＵからｓ１の処理を行なうＣＰＵへｓ１を実行するのに必要な処理は全て終わったことを通知するのに用いる。

（ｄ） ｆｏｒループはそのとき空いているＣＰＵ、例えばＣＰＵ０～ＣＰＵ２を使い、ループの１周目はＣＰＵ０、２周目はＣＰＵ１、３周目はＣＰＵ２、４周目はまたＣＰＵ０のように処理する。
この時ループの各周期に依存する変数の処置と最適化の方式を第７図に例を用いて示す。



第７図

１０．まとめ

並列コンピューターの為の、プログラマーにとってかなり使いやすいと思われる言語ＰＡＲＡＣと、 逐次処理プログラムを自動的に並列化するエキスパート・システムＡＰＡＲＡＣを開発した。 ＰＡＲＡＣは現在我々も毎日、プログラム作成に使用しており かなり良い評価結果も出ている。 ＡＰＡＲＡＣは改良の要素は無限であるが並列化ルールを追加することにより次々と良い並列プログラムを作成することができるように改良できる。 この改良を続けることによって、近い将来 並列コンピューターが全く通常のコンピューターとして 使われるようになると考える。