マイクロコンピュータ 48－2
計算機アーキテクチャ 69－2
（1988. 1. 21）

並列処理システム－晴－の実行方式

丸島敏一　山名早人　萩原孝
草野義博　村岡洋一

早稲田大学　理工学部

－晴－は従来の逐次型言語を入力とする科学技術計算用並列処理システムである。－晴－ではプログラムをいくつかのマクロブロックという集まりに分割し、マクロブロック同志の上位レベルの並列処理とマクロブロック内のデータフロー実行という2段階の並列処理を実現する。本稿では、まず、この－晴－の思想を説明し、－晴－におけるマクロブロック化の概念について述べる。また、－晴－のアーキテクチャ及びその処理方式について説明し、最後にソフトウェアシミュレータによる評価結果を報告する。

Execution Mechanism of Parallel Processing System -Harray-

(in Japanese)

Toshikazu MARUSHIMA, Hayato YAMANA, Takashi HAGIWARA,
Yoshihiro KUSANO and Yoichi MURAOKA

School of Science and Engineering, Waseda University
3-4-1 Okubo, Shinjuku, Tokyo, 160, Japan

We Propose the parallel processing system -Harray- for supercomputing with an ordinary language. The -Harray- adopts dataflow control in each processor, and it has a mechanism to control the execution order of program blocks. Thus, the -Harray- implements both instruction level parallelism and task level parallelism.

This paper reports the architectural concept of -Harray- and the evaluation results by software simulation.

1．はじめに

科学技術計算を対象とした演算パイプライン型ベクトル計算機、所謂スーパコンピュータは商業的には成功を収めた。しかし、このようなベクトル演算をベースとした計算機では、今後複雑化する高速処理に対応することが困難になってくる。一方、複数のプロセッサによる並列処理マシンは、専用マシンとして多くの成果を挙げているものの、一般的な科学技術計算用として用いるには未だ幾つかの問題点がある。

その問題点の一つに、使用する側の言語の問題がある。並列型言語による方法は、記述する系の持つ並列性をそのまま反映させるという意味で、ある特定のアプリケーションでは有効である。しかし、より一般的に枠を広げると、多数のプロセスに対して全て明示的に並列性を記述することは、もはや現実的ではなくなる。また、現状では並列処理用のアルゴリズムは開発段階にあり、その手法も確立されていない段階で全面的な意識改革を望むのは容易ではない。さらに、科学技術の分野では、従来からのソフトウェア資産の蓄積は膨大な数に上るため、アーキテクチャの改良によりそれらが全て利用できなくなっては並列処理マシンの普及は難しい。

我々は以上のような観点から、既存するプログラムやアルゴリズムを高速に処理するための処理方式、並びに並列処理アーキテクチャを提案している【１】。本稿では、科学技術計算を並列処理するための方式を説明し、我々の提案するマクロブロック化手法について述べる。さらに、これを効率良く実行する並列処理システムー晴ーのアーキテクチャを示し、ソフトウェアシミュレータによる性能評価を行う。

2．マルチプロセッサによる科学技術計算

科学技術計算をひとたび言語として記述すると、その処理はループ処理とスカラ演算処理とに大別される。以下ではこれらのマルチプロセッサ上での処理方式について概観する。

2．1　ループ処理

マルチプロセッサ上でループ処理を行うには、大きく分けて pipelining と forall という２通りの方法がある【２】。

pipelinig とは、ループ内の各文毎にプロセッサを割当て(厳密には、非循環グラフにより結合される、πブロック【３】と呼ばれるブロック毎にプロセッサを割当てる)、プロセッサ内ではインデックス方向(i=1→i=n)に実行する方式である(図1-(b))。この方式では、循環参照関係がπブロック内に閉じるように割当てが行われる(partial partition【４】)ため、循環参照に対して通信頻度を抑えるような形で実行される。しかし、πブロック間に依存関係が存在する場合には、プロセッサ間通信は必要になる。

一方、forall では、ループのインデックス毎にプロセッサを割当て、プロセッサ内では文方向に実行する(図1-(c))。この方式では、インデックス間の通信を前提としているため、依存関係によっては通信が頻発する反面、並列度の高い実行が可能である。この方式はさらに幾つかの特別な形態に類別される。まず、インデックス間で通信がおこなわれない場合を doall と呼び【５】、この時は各プロセッサ毎に独立に実行することができる。また、循環参照関係にあるものを forall 方式で実行するものを doacross と呼び、細かいレベルの並列性を引き出せることが確認されている【６】。

上に述べたように、一般にループ処理にはプロセッサ間通信に伴う同期が頻発する。そのため、これをマルチプロセッサ上で行う場合には、それぞれのマルチプロセッサに強力な通信同期機構を備えることが必須となる。図２は上述の doacross に対してソフトウェアによる同期を実現したものであるが【７】、このようなソフトウェア的同期によるオーバヘッドは一般に非常に大きなものとなる。

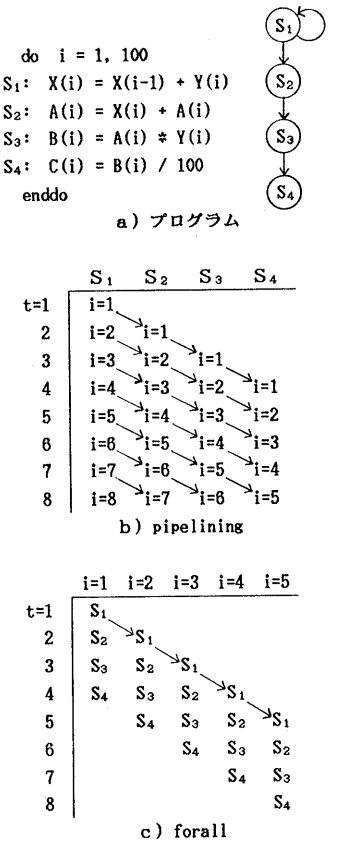

図1　ループ処理方式

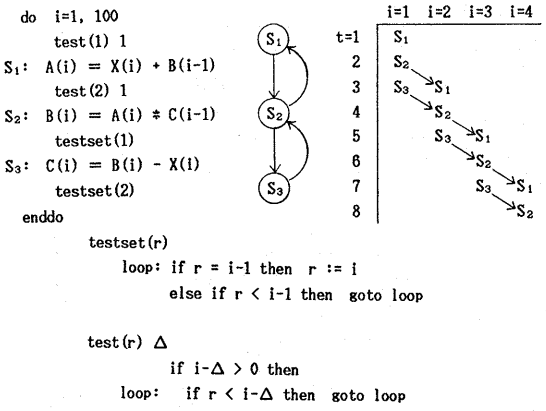

図2 doacross における同期

2．2 スカラ演算処理

スカラ演算を高速に処理するためには、並列に実行可能な演算を別々のプロセッサに割当てる必要がある。この方式を low-level spreadingと呼び【8】、これを実現するための一手法として tree-height reduction 等が提案されている【9】。しかし、一般のプロセッサでは、このような粒度の小さいタスクに対するプロセッサ割当てのスケジューリングは困難なものとなる。また、通信同期機構が強力でないと、通信オーバヘッドが並列処理による効果を上回ってしまう可能性もある。

3．マクロブロック化

前章で述べたように、マルチプロセッサ上で科学技術計算を行うには、通信同期機構や動的なスケジューリング機構が必要になる。そこで、我々は上述の機構を備えたデータフローコンピュータの機能に着目し、これを利用することにした。すなわち、既存プログラムを高速に処理するアーキテクチャとして、データフロー実行を行う複数プロセッサによる並列処理アーキテクチャを採用したのである。

ところで、これを言語の側から考えた場合、既存の逐次型言語をそのまま純粋なデータフローアーキテクチャに適用することは些か抵抗がある。それは、両者間に存在するセマンティックギャップが大きく、元来所持していた情報を失ってしまうことになるからである。特に、逐次型言語の持つ順序付けというのは、制御の移動として実現されることが多く、その全てを捨て去るのは重大な損失となる。そこで、これを解決するために我々はプログラム全体をいくつかの『マクロブロック』というステートメントの集合に分割し、このブロック間の順序付けを保持することにした。これを我々は『マクロブロック化』と呼んでいる。

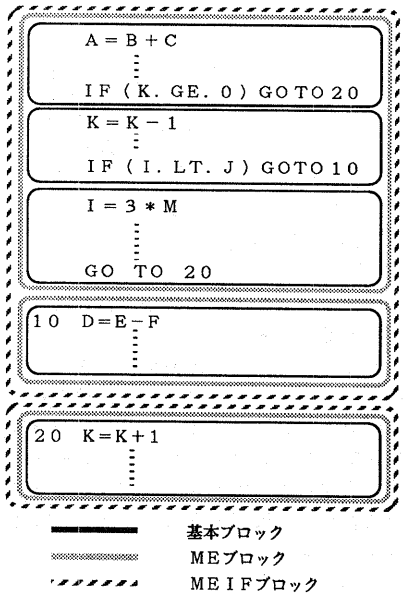

図3 マクロブロックの定義

3．1 マクロブロックの定義

マクロブロック化の単位について、我々は次の3つの基準を用いている【10】。

(1) Basic Block (基本ブロック; BAS)
途中の飛込みや飛出しのないブロック

(2) Multiple Exit Block (MEB)
(1)にブロック外への飛出しを許したブロック

(3) Multiple Exit + perfect IF Block (MEIFB)
(2)にブロック内に完全にネストした if-then-else文を許したブロック

これらを比較すると、BAS から MEB, MEIFB となるにつれて、ブロックの粒度が大きくなり、ブロック数が減少するため、ブロック間の同期に伴うオーバヘッドの改善が見込まれる(図3)。実際にいくつかのアプリケーションに適用した結果、BAS に対して MEB と MEIFB を比較すると、データの待ち合わせに伴う実行時間の増加を抑えながら、ブロック数を減少させ、ブロック内の並列度を増加できることを確認した。

3．2 マクロブロック実行の起動

マクロブロック毎に制御を行い、実行の起動を行うには、表1に示す3つの実行形式が考えられる。

(1) 同期実行
if文等の条件分岐により必ず実行されることが決定し、かつ必要なデータが全て揃っていることを保証して実行する方式

表1　マクロブロックの実行形態

	実行呼称	制御条件	データ条件
本実行	同期実行	必ず実行されることが決定している	全てのデータが到着済
	先行実行		到着してるか不明
	仮実行	実行されるか不明	

（2）先行実行

（1）の方式に比べ、全データの到着を待たずに実行を開始する方式

（3）仮実行

条件分岐による実行の決定が行われない段階で、実行を開始する方式

これらを比較すると、同期実行から先行実行，仮実行となるにつれ、ブロック同士の並列度が増加し、実行速度の向上が見込まれる。しかし、先行実行や仮実行を無制限に行うことはプロセッサ利用率の低下を招き、特に仮実行では膨大な数のプロセッサを浪費する場合があるため、使用にあたっては適切な選択を行わなければならない。

3．3　マクロブロック化の利点

純粋なデータフロー方式に対してマクロブロック化を行うことの得失を考える。まず、前述の同期実行により実行の開始が制限されることから、マクロブロック化後は実行時間が増加する可能性がある。これについては、通信・同期機構によるオーバヘッドを考慮しないモデルでは、純粋なデータフロー実行を行った場合と比べて実行時間の増加を殆ど招かないことが、いくつかのアプリケーションについて確認されている【11】。

マクロブロック化による利点としては、以下のものがあげられる。

(1) 並列性を陽に秩序付けることが可能となり、資源管理を容易に行うことができる。データフローコンピュータでは並列性を引き出す機能はあるが、その並列性を制御する機能は基本的には備えていない。そのため、爆発的な並列度に対して資源が枯渇してしまう可能性がある。マクロブロック化によれば、並列性を適宜制御して実行を進めることができる。この一例として待ち合わせ記憶のワーキングセットを容易に導入することができる【12】。

(2) コントロールフローを積極的に利用することができるため、制御の切り換えに伴うデータフローオーバヘッドを軽減することができる。データフローコンピュータでは、制御の切り換えに際して、必要なデータ全てに対して条件分岐命令を付加する必要があるため、実行命令数が増加してしまう。マクロブロック化によれば、ブロック毎のコントロールフローとして制御の切り換えを行うことができ、冗長な命令を省くことができる(図4)。

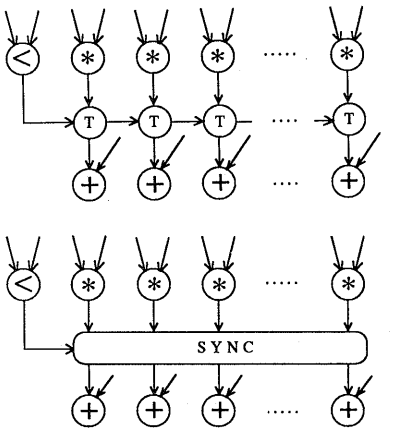

図4　コントロールフローの利用

(3) 全体の実行の流れが明確になり、システムの挙動を把握し易くなる。一般に並列処理計算機のデバッグは難しいといわれているが、データフローコンピュータになるとさらに困難なものとなる。マクロブロック化によれば、並列処理の世界をブロック内とブロック間という二層に分離するため、階層的な制御・管理が実現できる。

4．－晴－のアーキテクチャ

並列処理システム－晴－は、科学技術計算を目的としたデータフロー・マルチプロセッサシステムである。－晴－では、マクロブロックを単位としてコントロールフロー制御を行い、マクロブロック内では演算レベルのデータフロー制御を行う。これにより、マクロブロック同士の上位レベルの並列処理とマクロブロック内の低レベル並列処理という、集中管理下の非同期分散制御を実現する。

このマクロブロックの思想は、Cedar のマクロデータフロー【13】や、ＰＤＦでの基本ブロック【14】にも見られるが、前者ではマクロ内の並列性を十分生かせない、後者では高並列処理を指向していない、等の問題点が指摘される。－晴－では、独立に処理可能なデータを高並列に分散し、それぞれスカラレベルの並列処理を行うことにより最大限の並列性を生かすことができる。

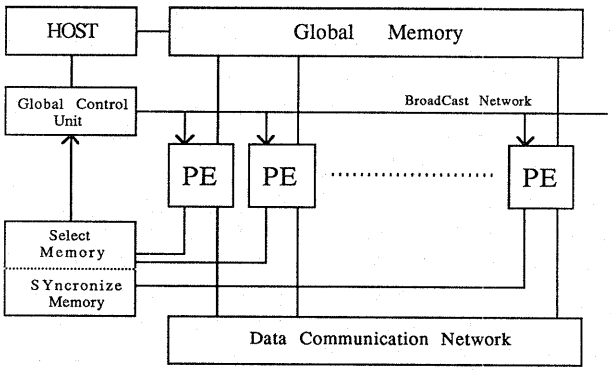

図５　全体構成

４．１　全体構成

－晴－の全体構成を図５に示す。全体的な制御を指示するマクログラフは、集中統括を司どる集中制御機構(ＧＣＵ：Global Control Unit)により管理され、要素処理装置(ＰＥ：Processing Element)への実行指示は、起動放送ネットワーク(ＢＣＮ：BroadCast Network)を通じて各ＰＥに放送することにより行われる。この際、１つのマクロブロックを複数のＰＥへ分散して割り付けることが可能だが、１つのＰＥでは同時期には１つのマクロブロックしか実行しない。また、実行の終了は同期メモリ(ＳＹＭ：SYncronize Memory)に報告され、ＧＣＵによる次の制御に委ねられる。

実行中はデータ通信ネットワーク(ＤＣＮ：Data Communication Network)を通じてＰＥ間のデータ交換が行なわれ、また共有データメモリ(ＧＭ：Global Memory)において配列等の大域的なデータの更新が行われる。このとき、ＧＭはある種の放送機構を持つため、ＧＭにおける共有データの管理、ＤＣＮにおけるローカルデータの交換という役割分担ができる。

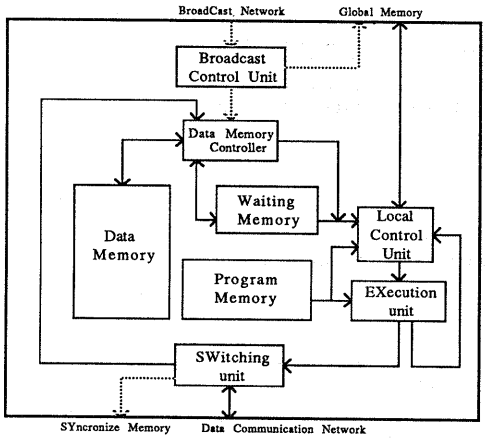

図６　ＰＥ構成

４．２　要素処理装置

要素処理装置(ＰＥ)の構成を図６に示す。ＧＣＵによるマクロブロック実行要求は起動放送制御ユニット(ＢＣＵ：Broadcast Control Unit)によりとりこまれ、実行すべきマクロブロックであった場合は、ＰＥ全体にその制御情報を伝える。これにより、既にデータメモリ(ＤＭ：Data Memory)内で実行条件を満たしているデータ対を持つ命令から実行が行われ、続いて待ち合わせ記憶(ＷＭ：Waiting Memory)によりデータ駆動的に他の命令が起動されていく。そして、そのマクロブロックの終了条件が満たされたことをスイッチユニット(ＳＷ：Switch unit)が検知した場合、ＳＹＭを通じてＧＣＵにマクロブロック終了報告をし、それと同時にＷＭ内のデータをクリアし、次のマクロブロックのデータをＷＭ内にロードする。

実行可能となったパケット対はＰＥ内制御ユニット(ＬＣＵ：Local Control Unit)により実行可能パケットとなり、実行ユニット(ＥＸ：EXecution unit)に送られる。実行ユニットは、同一の機能をもつ複数の演算処理装置からなり、各々並列に実行を行うことができる。このＥＸでの実行に並行して、プログラムメモリ(ＰＭ：Program Memory)から次の命令がフェッチされ、結果パケットを生成する。また、データメモリ制御ユニット(ＤＭＣ：Data Memory Controller)はこの結果パケットの行き先ブロックを識別し、ＷＭとＤＭとに振り分ける。これにより、ＷＭには実行中のマクロブロックに必要なデータのみが存在することになる。

ＰＥ内での処理方式は、ここまでに述べたデータフロー実行によるモード(データフローモードと呼ぶ)の他に、単純かつ定型的な配列演算のためにベクトルモードを設けている。これは、ＥＸの出力をＷＭに送らずに、ＬＣＵへ直接渡すことにより実現され、ＷＭにまつわるオーバヘッドを削減することができる。また、ＬＣＵ内にはベクトルレジスタを設けているので、定型的データの供給能力を向上することができる。

４．３　集中制御装置

集中制御装置(ＧＣＵ)の構成を図７に示す。マクロブロック間の制御情報はマクロプログラムメモリ(ＭＰＭ：Macro Program Memory)にあり、前述の実行条件を満たしたマクロブロックから起動がかけられる。マクロブロックの起動時には、プロセッサの割当て状況と次のマクロプログラムへのポインタがＳＹＭに登録され、続いてＢＣＮを通じて各ＰＥに対して起動を促す放送が行われる。各ＰＥでの実行終了はＳＹＭを通じて報告され、該当マクロブロックに対応する全プロセッサが実行終了した時点、もしくは分岐先が決定した時点で、次のマクロプログラムへのポインタがＳＹＭからＧＣＵに報告される。この分岐先が動的に求められる場合には、ＰＥが セレクトメモリ (Select Memory) に書き込んだ分岐マクロブロック番号を使用する。また、複数のマクロブロックの待ち合わせを行うために、待ち合わせバッファ(Waiting Buffer)が使用される。

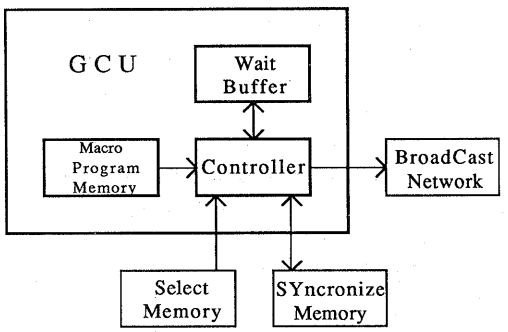

図７　ＧＣＵ構成

５．ソフトウェアシミュレータによる評価

－晴－の基本的な性能を評価するために、機械語レベルのソフトウェアシミュレータを作成した。このシミュレータは、VAX11/785 上の Modula-2 言語において約13,000行で書かれている。

５．１　ＰＥ１台による性能比較

ＰＥ１台当りの性能，ベクトルモードの立上り，配列データのプリフェッチ効果を測るために、まずＰＥ１台によるシミュレーションを行った。但し、ＰＥ内のＡＬＵ数は３台とした。

図８はベクトル加算におけるベクトル長と実行速度との関係を示している。ここで、データ先行フェッチによるデータフローモード(data prefetch dataflow mode)とは、配列をＧＭに置かずにＰＥ内のＤＭに取っておく方法で、データ先行フェッチしないデータフローモード(data non-prefetch dataflow mode)では１要素毎にＧＭにアクセスする。また、ベクトルモードは、４．２に述べたようにＷＭによる同期を介さないモードで、ＧＭには 128要素毎にアクセスする。

図８の結果から、ベクトル長 15程度のところで、ベクトルモードとデータ先行フェッチによるデータフローモードとの逆転が起こっているのがわかる。このことは、ＧＭにフェッチするにもかかわらず、ＷＭを介さないことによるベクトルモードの効果が、単純な演算に対して比較的短いベクトル長で効くことを示している。また、ベクトルモードの最大性能約 0.8MFLOPSに対する半性能長【14】(half-performance length; 最大性能を達成するのに必要なベクトル長)は、約５となっており、演算パイプラインであるCRAY-1の10前後に比べて、短いベクトル長に対しても効果があることを示している。

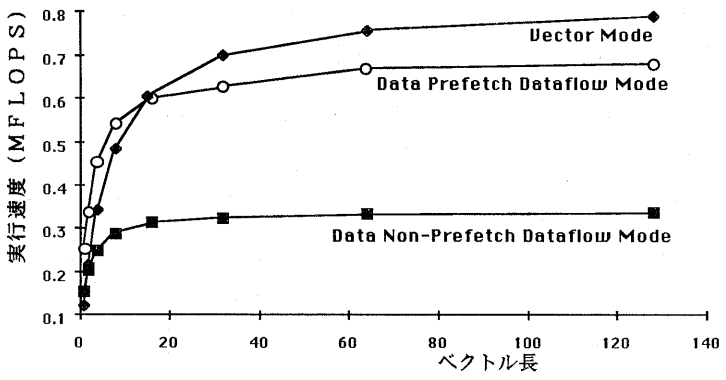

図８　ベクトル加算のベクトル長による実行速度

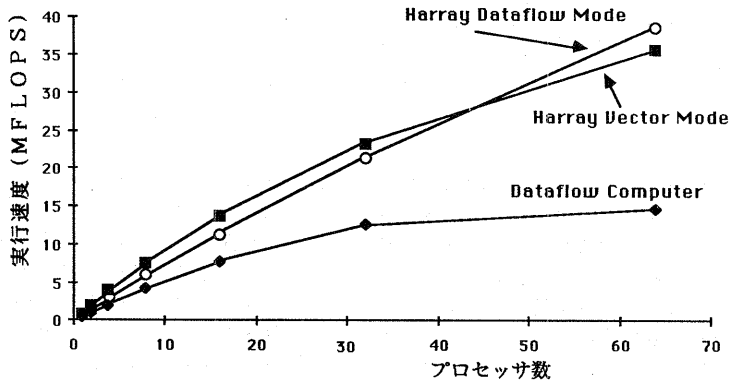

図９　ＥＸ１における条件分岐成立時の実行速度

表２　ＥＸ１における条件分岐成立時を１とした、不成立時の実行時間比

対象マシン	－晴－ ベクトルモード	－晴－ データフローモード	データフロー コンピュータ
実行時間比	１．３７	１．５４	１．９６

また、データ先行フェッチを行うデータフローモードと行わないものとの関係から、配列データのプリフェッチによる効果が高いことがわかる。これは、加算という簡単な演算に対してＡＬＵ数を３台としたことからＧＭの速度がネックとなってしまった結果で、ベクトルモード時のようなＧＭへの連続アクセスの重要性を示している。

５．２　複数ＰＥによる台数効果とマクロブロック化

複数ＰＥによるシミュレーションには、次のプログラムを用いた。

```
EX1: doall 10 i=1, 128
       A(i)=B(i)+C(i)          block:1
  10 continue
     if (m.GE.n) GOTO 99       block:2
     doall 20 i=1, 128
       D(i)=A(i)+1             block:3
  20 continue
  99 doall 30 i=1, 128
       E(i)=A(i)-1             block:4
  30 continue
```

マクロブロック化の基準として前述の基本ブロックを用いている。ブロック２の条件分岐でm≧nが成り立った時の台数効果が図９である。ここにあるデータフローコンピュータとは、マクロブロック化を行わない純粋なもので、但し、ＰＥ１台当りのＡＬＵ数は－晴－と同じく３台としてある。また、－晴－のデータフローモードとして前述の先行フェッチ方式を用いている。

図９で－晴－のデータフローモードとベクトルモードとを比較すると、始めのうちはベクトルモードの方が良い性能を示しているが、ＰＥ数40を過ぎたあたりでこれが逆転している。これは、ベクトルモードにおけるベクトル長が、ＰＥ数の増加に従って減少しているためで、ベクトルモードによる効果が相殺されてしまったからである。

同じく図９よりわかるのは、ＰＥ数が増加するにつれデータフローコンピュータに対する－晴－の性能が際立っていることである。この原因の１つは、－晴－では複数ＰＥにわたる実行の同期をハードウェアによって行っているため、この部分のオーバヘッドが少なくてすむからである。もう１つとしては、データフローコンピュータでは全てのトークンを実行候補としてＷＭに送ってしまうため、ＷＭに対する負荷の集中が起こることが原因となる。

EX1では、ブロック２の条件分岐が不成立であった場合には、成立時よりも多く命令を実行する。この条件分岐成立時と不成立時の比を表したのが表２である。これからわかることは、－晴－ベクトルモード，－晴－データフローモード，データフローコンピュータの順に実行時間比が伸びていることである。このことは、－晴－においては条件分岐をマクロブロック単位で行う為、制御の切り換えに掛かるコストが比較的小さくてすむことを示している。

ここに示した結果より、制御の切り換えを伴うプログラムでは、－晴－におけるマクロブロック化の効果が顕著であることがわかった。

6．おわりに

本稿では、並列処理システム－晴－の概要および評価結果について述べた。元来データフローアーキテクチャの研究は、関数型言語との親和性や動的な並列性抽出機能を中心として進められてきた。これに対し、我々は既存プログラムの高速化を出発点として、データフローアーキテクチャを実用的に利用するという立場から研究を進めている。データフロー方式とコントロールフロー方式は、各々が独自の領域を担っており、両者の欠点を補う形で融合していくことは今後益々重要になるであろう。

参考文献

【1】丸島，村岡："科学技術計算用並列処理システム－晴－：全体構成"，第３４回情処全大，1Q-7，(1987)

【2】D.A.Padua, D.J.Kuck, D.H.Lawrie : "High-Speed Multiprocessors and Compilation Techniques", IEEE Trans. on Computers, Vol. C-29, No.9, pp.763-776 (1980)

【3】U.Banerjee, S.Chen, D.J.Kuck, R.A.Towl : "Time and Parallel Processor Bounds for Fortran-like Loops", IEEE Trans. on Computers, Vol. C-28, No.9, pp.660-670 (1979)

【4】D.J.Kuck and D.A.Padua : "High-Speed Multiprocessors and Their Compilers", Proc. of Int. Conf. on Parallel Processing, pp.5-16 (1979)

【5】S.F.Lundstron and G.H.Barnes : "Controllable MIMD Architecture", Proc. of Conf. on Parallel Processing, pp.19-27 (1980)

【6】R.G.Cytron : "Compile-time Scheduling and Optimization for Asyncronous Machines", Ph.D Thesis, University of Illinois at Urbana-Champaign (1984)

【7】S.P.Midkiff and D.J.Kuck : "Compiler Generated Syncronization for Do Loops", Proc. of Conf. on Parallel Processing, pp.544-551 (1986)

【8】D.J.Kuck, A.H.Sameh, R.Cytron, A.V.Veidenbaum, C.D.Polychronopoulos, G.Lee, T.McDaniel, B.R.Leasure, C.Beckman, J.R.B.Davies, and C.P.Kruskal : "The Effects of Program Restructuring, Algorithm Change, and Architecture Choice on Program Performance", Proc. of Int. Conf. on Parallel Processing, pp.129-138 (1984)

【9】Y.Muraoka : "Parallelism Exposure and Exploitation in Programs", Ph.D Thesis, University of Illinois at Urbana-Champaign (1971)

【10】萩原，丸島，村岡："並列処理システムにおけるFortranプログラムのマクロブロック化の評価"，第３５回情処全大，1C-4，(1987)

【11】萩原，山名，丸島，村岡："科学技術計算用並列処理システム－晴－のマクロブロック化手法の検討"，第３４回情処全大，1Q-9，(1987)

【12】山名，丸島，萩原，村岡："科学技術計算用並列処理システム－晴－の要素プロセッサ構成の検討"，第３４回情処全大，1Q-8，(1987)

【13】D.D.Gajski, D.J.Kuck, D.H.Lawrie, A.H.Sameh : "Cedar-A Large Scale Multiprocessor", Proc. of Int. Conf. on Parallel Processing, pp.524-529 (1983)

【14】J.E.Requa and J.R.McGraw : "The Piecewise Data Flow Architecture : Architectural Concept", IEEE Trans. on Computers, Vol. C-32, No.5, pp.425-438 (1983)

【15】R.W.Hockney, C.R.Jesshope : "Parallel Computers", Adam Hilger Ltd. (1981)