並列構造記述モデルとそれを実現する高水準並列計算機†

古谷立美†† 内堀義信†† 西田健次††

この論文では，従来汎用並列計算機のもっていたプログラム構造とハードウェア構造の間のセマンティックギャップの問題を解決する方法として，並列計算機を並列プログラム用高級言語計算機として設計することを提案する．そしてこれを実現する方法として，種々の並列プログラミング言語で書かれた並列プログラムの構造を系統的に表現するモデルを与えるとともに，このモデルを効率よく実現するアーキテクチャを提案する．このモデルは，並列プログラムをタスク単位に分割し，タスク間の依存関係をキューイングネットワークの形で表現するもので，これは並列プログラミング言語のための仮想計算機に相当する．次に示すアーキテクチャではキューのハードウェア化と可変構造マルチリードメモリが用いられる．最後にマルチリードメモリを用いたシステムの性能評価シミュレーション結果を示す．

1. まえがき

いままでに多くの並列計算機システムが提案され製作されてきた．それらのシステムを振り返ってみると，専用システムの分野ではよい成果を得ている反面，汎用システムでは必ずしもよい成果が得られていない．

一方，並列システムに対するニーズをみると，汎用システムに対する要求は少なくない．これは一つのシステムを複数の応用に適用しようという場合だけでなく，一つの応用のなかにも種々の並列形態が要求されることがあるためで，画像処理等はその代表である。

汎用システムがよい成果を上げなかったおもな理由は二つある．第一は，プログラミング言語の問題である．最近まで並列プログラムを構造的に記述できる言語がなかったため，プログラミングは従来のシーケンシャルな言語に通信や同期用プリミティブを加えて行っていた．このためプログラミングはむずかしく，生産性，読みやすさ，保守性等に多くの問題があった．第二はハードウェアの構造とプログラムの構造の間のセマンティックギャップである．従来，並列計算機システム設計者はシステムを独立性の高いシーケンシャル計算機の集合体としてとらえ，計算機間の通信や同期の機能は融通性を重視して基本的な低レベルのものしか与えなかった．そのため並列プログラムを走らせる場合には OS の助けが必要となり，OS は複雑になりオーバヘッドは増大した．

† A Model for Parallel Structure and a High-Level Parallel Computer which Implements the Model by TATSUMI FURUYA, YOSHINOBU UCHIBORI and KENJI NISHIDA (Computer System Section, Electronic Computer Division, Electrotechnical Laboratory).

†† 電子技術総合研究所電子計算機部計算機方式研究室

まず第一の問題である並列プログラミング言語に関しては，近年研究が進み Ada, 並列 Pascal, Actus 等有力な言語が開発され，今後さらに優れたものが現れると思われる．

第二の問題に対する有力な方法として本論文で提案するのは，並列計算機システムを並列プログラミング言語用高級言語計算機[1] として設計する方法である．これは並列システムを個々の独立した計算機の集りではなく一台の計算機としてとらえ，高級言語のもつ制御機構から基本機能を整理し機械語に組み入れていくトップダウンアプローチであり，プログラミング言語を効率よくサポートできる．これは従来からある OS のハードウェア化[2] の側面をもつが，並列プログラミング言語サポートという面から機能が整理されている点が，いままでになかった点である．

この論文では，まず種々の並列プログラム用高級言語で書かれたプログラムの構造を簡潔な形で表現できるモデルを提案する．これはプログラムをタスク単位に分割し，それらの間の依存関係をキューイングネットワークの形で表現するものである．これは，並列プログラミング言語のための仮想計算機に相当する．3章では，このモデルを効率よく実現するとともに従来の並列計算機の問題点を解決するアーキテクチャを示す．そこでは，ハードウェアキューと可変構造マルチリードメモリが用いられる．最後にマルチリードメモリを用いたシステムの性能評価シミュレーション結果を示す．

2. 並列構造記述モデル

並列プログラミングの研究は近年盛んで，構造的プログラムが可能なさまざまな高級言語が提案されてき

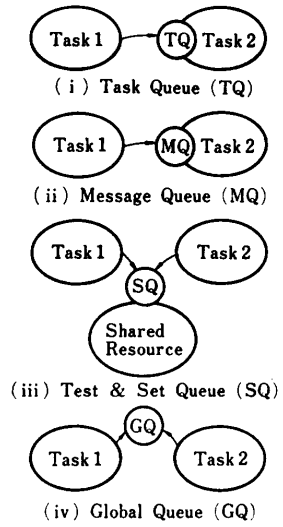


図 1 キュータイプ
Fig. 1 Queue type.

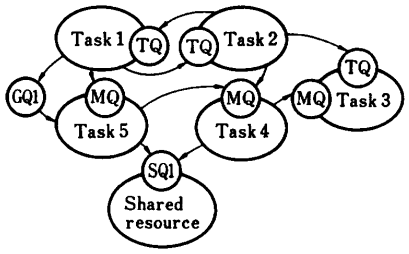


図 2 タスクの依存グラフ（意味のないプログラム）
Fig. 2 A dependency graph of tasks.

た．MIMD に適した Ada，並列 Pascal，SIMD 用 Actus 等がその代表例である．ここで示す記述モデルとは，種々の並列プログラミング言語で書かれたプログラムの構造を簡潔な規則の下に整理し表現するものである．ここでは，タスクと呼ばれる並列処理の基本単位と，それらの間の依存関係を待ち行列を用いて表しており，並列プログラムはタスクのキューイングネットワークの形で表される．このモデルを作成するに当たっては，たんにプログラムの構造の表しやすさだけでなく，並列処理計算機で効率よく実現できることも重視している．

タスクの依存関係を表すのに待ち行列（以下 Q と略）を選んだ理由は次の三つである．

（1） タスクのスケジュールは一般に Q が用いられる．

（2） メッセージ交換等のハードウェア化では，メッセージバッファに柔軟性を与えるため Q が用いられる．

（3） 共有資源へのアクセス制御には Q が用いられる．

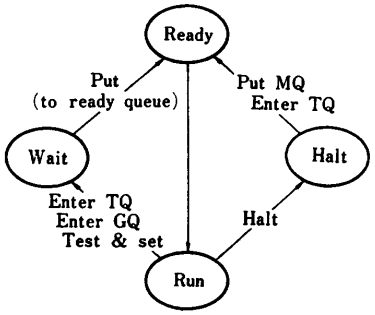


図 3 タスクの状態遷移
Fig. 3 Task state diagram.

表 1 待ち行列操作命令
Table 1 Queue handling instruction.

Operation	Queue Type	Queue Name	Data to TCB	Data from TCB
Enter	TQ	TN, QN	TN*, PC(,M)	
	GQ	QN	TN*, PC(,M)	
Put	TQ	TN, QN	TN, PC (,M)	
	Ready Queue		TN, PC (,M)	
	GQ	QN	TN, PC(,M) or M	
	MQ	TN, QN	M	
Get	TQ	TN*, QN		TN, PC(,M) or E
	MQ	TN*, QN		M or E
	GQ	QN		TN, PC(,M) or M or E
Test & set	SQ	QN	TN*, PC(,M)	Set or Reset
Reset	SQ	QN		
Halt	QN 1…QN$_M$		TN*, PC	

TN. PC: Task name & Value of program counter,
TN*: Name of task which issues this instruction,
QN: Queue name. M: Message. E: Empty. ( ): If necessary.

このモデルでは4種類の性格の異なる Q を用意している．**図1**はこのグラフ表現であり，**図2**は並列プログラムの構造を Q を用いて表した例である．

一般にタスクはシーケンシャルプログラムで，タスク中には任意個の Q を設定できる．タスクから Q へのアクセスはプログラム中の Q 操作命令で行われる．**表1** は Q 操作命令を示している．この命令は並列プログラム用仮想計算機の命令と考えることができる．

タスクは**図3** に示す4種類の状態のいずれかにあり，その状態は Q 操作命令で変化する．

次に各 Q と Q 操作命令に伴う状態変化を説明する．

（1） Task Queue (TQ)

これはタスク間の同期を取るためのもので，ソースタスク（図1（i）の Task 1）がプログラム中で ENTER TQ 命令を発すると，ソースタスクが，命令で指定する TQ に入る．TQ に入るとタスクは WAIT 状態になる．TQ はデスティネーションタスク（図1（i）の Task 2）に付属している．タスクは自分の TQ の先頭のタスクを取り出し (GET 命令)たり，取り出したタスクを他の Q へ入れる (PUT 命令) ことができる．GET 命令では Q が空のとき EMPTY が返される．一般には GET 命令でタスク名と要求を知り，必要な処理を行った後，PUT (to READYQUEUE) 命令で，ソースタスクを READY 状態にしレディ Q に送る．レディ Q とは READY 状態のタスクが入るシステムサポート Q である．

このほか TQ には，TQ へのタスク到着で，TQ の属するタスクを WAKE UP する機能がある．デスティネーションタスクが HALT 命令を出すと，そのタスクに属する指定された Q が調べられ，いずれの Q にもタスクが到着していない場合にタスクは HALT 状態になる．HALT 状態になったタスクは，HALT 命令で指定した Q にタスクが到着したとき WAKE UP され READY 状態になる．**図4**（a）は Ada による生産者・消費者プログラムのバッファリングタスク[3]を Q 操作命令で記述したものである．

（2） Message Queue (MQ)

これはタスク間でメッセージ転送を行うためのもので，TQ 同様あるタスクに付属して，そのタスクに制御される．ソースタスクは PUT MQ 命令でメッセージを送る．TQ との違いは，ソースタスクの状態が PUT MQ 命令で影響を受けない点である．デスティネーションタスクは自分の MQ の内容を取り出して処理することができ，TQ 同様 MQ へのメッセージ到着による WAKE UP も可能である．

（3） Test & Set Queue (SQ)

この Q は複数のタスクが共有データや共有プログラムといった共有資源にアクセスする場合の排他制御を実現するもので，一時にたかだか一つのタスクしか共有資源にアクセスできないように制御する．Test & Set SQ 命令により SQ に付属した Test & Set ビットがタスクに返され，同時に Test & Set ビットはセットされる．Test & Set SQ 命令で Test & Set ビットがリセットされていた場合にはタスクは共有資源にアクセスすることができ，セットされていた場合には SQ に入り WAIT 状態になる．

```
begin
 loop
  select
   when COUNT < SIZE⇒
    accept WRITE(E :in ITEM) do
      BUFFER(INX) := E;
    end;
    INX :=INX mod SIZE + 1;
    COUNT :=COUNT + 1;
  or
   when COUNT > 0⇒
    accept READ(V :out ITEM) do
      V := BUFFER(OUTX);
    end;
    OUTX := OUTX mod SIZE + 1;
    COUNT := COUNT -1;
  end select;
 end loop
end BUFFERING;
```

```
Task name=Buffering
A: if COUNT≧SIZE then goto B;
   get(TQ BUFFERING,WRITE,TN.PC,INITEM);
   if TN=EMPTY then goto B;
   BUFFER(INX):=INITEM;
   INX:=INX mod SIZE + 1;
   COUNT:=COUNT+1;
   put (READYQUEUE,TN.PC);
B: if COUNT=0 then goto C;
   get(TQ BUFFERING,READ,TN.PC,OUTITEM);
   if TN=EMPTY then goto C;
   OUTITEM:=BUFFER(OUTX);
   OUTX:=OUTX mod SIZE + 1;
   COUNT:=COUNT-1;
   put (READYQUEUE,TN.PC,OUTITEM);
C: halt (WRITE,READ,BUFFERING.PC);
   goto A;
```

(a) A example of message-passing.

```
loop PRODUCE (M);
region SV when COUNT≠SIZE do
 begin
  DEPOSIT (M);
  COUNT := COUNT + 1
 end
end

loop
region SV when COUNT≠0 do
 begin
  FETCH (M);
  COUNT :=COUNT - 1
 end;
CONSUME (M)
end
```

```
Task name=Producer
A: PRODUCE (M);
B: if COUNT=SIZE then goto B;
   test&set (SQ BUFFERING,PRODUCER.PC);
   DEPOSIT (M);
   COUNT:=COUNT+1;
   reset (SQ BUFFERING);
   goto A;

Task name=Consumer
C: if COUNT=0 then goto C;
   test&set (SQ BUFFERING,CONSUMER.PC);
   FETCH(M);
   COUNT:=COUNT-1;
   reset (SQ BUFFERING);
   CONSUME (M);
   goto C;
```

(b) A example of conditional critical region.

図4 生産者・消費者プログラム
Fig. 4 Producer・consumer program.

共有資源の使用を終えたタスクは RESET SQ 命令を出すと，SQ にタスクが入っているときは先頭のタスクが READY 状態になり，入っていないときは Test & Set ビットがリセットされる．

図4（b）は Hoare の条件付クリティカルリージョンを SQ を用いて実現している例である．

（4） Global Queue (GQ)

メッセージやタスクを一時的につないでおく Q である．タスクは ENTER GQ 命令でこの Q に入り WAIT 状態になる．TQ との違いは，特定のタスクに所属しておらず，WAKE UP 機能がないことである．

Q 操作命令には，以上説明してきたように，タスクの状態を制御する機能が含まれており普通の機械命令に比べてレベルが高く，図4に示すように高級言語の命令とほぼ一対一に対応が付くことから，Q 操作命令は高級言語計算機の仮想命令と考えられよう．またこのモデルはタスクの相互作用を実現するものであるが，アーキテクチャ（3章）では，このモデルの実現だけでなく，タスクの生成，消去，並列実行開始，終了等の機能もサポートしている．

3. 並列計算機アーキテクチャ

この章では，前章のモデルを効率よく実現するとともに，従来並列計算機がかかえていた諸問題を解決するアーキテクチャを示す．

3.1 設計思想

ここではおもな設計思想を示す．

（1） MIMD 制御

SIMD, Multi-SIMD, MIMD といった幅広い並列処理形態をサポートしうるものとして，最も広い概念である MIMD を基本方式とし，他の方式は MIMD でシミュレートする．

（2） 並列プログラミング言語のサポート

前章のモデルは，並列プログラムのための一つの仮想アーキテクチャであり，これを効率よく実現する必要がある．ここでは Q をハードウェア（ファームウェア化を含む）化して高速化を図る．

（3） タスクスケジューリング

並列計算機ではタスクをプロセッサに割り当てるスケジューリングが大きな問題である．ここではタスクとプロセッサの結合を静的だけでなく動的にも行える方式として READY タスクが低負荷のプロセッサへ流れていく方式を選び，ハードウェア Q を用いてプロセッサとタスクの高速バインディングを実現する．また For all 命令等による複数タスクの同時スタートやストップを実現する機構を用意する．

（4） プロセッサ間結合

プロセッサ間結合方式としては，共有資源の操作が容易な共有メモリ方式を選ぶ．これによりタスク本体はスケジュール時に移動する必要はなくなる．共有メモリ方式の弱点は，メモリでの競合問題である．ここでは共有メモリをマルチリードメモリにして対処する．マルチリードメモリとは[4]，プロセッサ数分だのポートをもつ共有メモリで，メモリへの書込みは一度に1台のプロセッサしかできないが，読み出しは他のプロセッサの読み出しと同時に行えるもので，内容が同一になるように制御されたプロセッサ数分だけのメモリモジュールを用意することで実現できる．これは一般にライト命令のほうがリード命令より発生頻度が少ないことと，VLSI の進歩により大容量メモリが安価に入手できる傾向にあることを利用したものである．

（5） 可変構造

種々の並列処理形態を効率よく実現するには，ハードウェアに可変性をもたせることが有用である．このアーキテクチャでは共有メモリに可変性を与え階層構造等を実現しやすくしている．

（6） オブジェクトのサポート

オブジェクト指向言語を効率よく実現する．タスクはオブジェクトとして扱われ，タスク名をアドレスに変換する機能を有する．

3.2 アーキテクチャ

図5が並列計算機の構成であり，プロセッシングエレメント (PE), 共有メモリ，タスク制御部 (TCB), レ

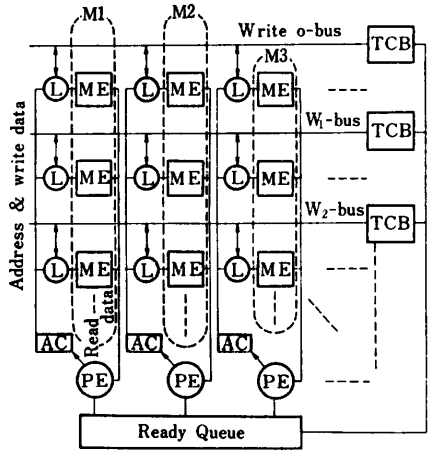


図 5 ハードウェア構成
Fig. 5 Hardware structure.

ディQ(RQ)より成る．共有メモリはメモリエレメント(ME)が二次元に配列されているが，縦の列はPEの専用バスで結合されている．横に走っているバスはWRITEバスと呼ばれる書込み専用バスで，このバスを用いてデータがブロードカストされ図中に破線で囲んだM1，M2,…の内容はすべて同一になりマルチリードメモリが形成される．2章で示したキューイングネットワークは，TCBとRQで実現される．TCBはタスクに付随したQと，Q操作命令に伴うタスクの状態変化を管理する部分で，Q操作命令により起動され，PEが次の命令を実行する前に処理を終える．TCBでREADY状態になったタスクはメッセージとしてRQへ送られる．PEは隣接するPEとの間にパスをもつ．

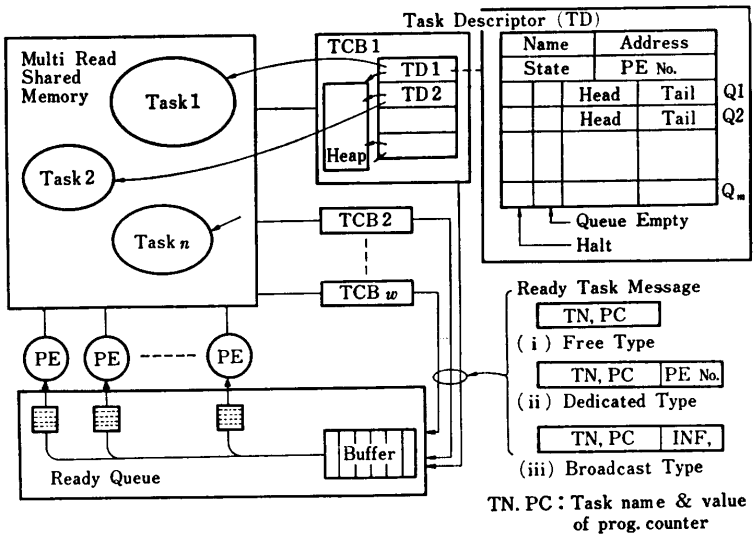


図6 タスク制御機構
Fig. 6 Task control mechanism.

待ち行列サポート機能

2章のモデルのQの機能はTCBとRQで実現される．**図6**はタスク制御の機構を示したものである．タスクの実体は共有メモリにあり全PEからアクセスできる．各タスクにはデスクリプタが付随し，これはTCBに置かれる．デスクリプタはタスクへのポインタ，タスクに付属したQ，およびQの状態を示すEMPTYビットとHALTの状態を示すHALTビットから成る．Qはポインタのみがデスクリプタにあり，要素はヒープに入れられる．タスクの状態がREADYになるとタスク名とWAITになったときのプログラムカウンタの値がメッセージの形でRQに送られる．これによりタスクのプロセッサへのスケジュールはタスクの実体は移動せず，ポインタのみの移動で済む．

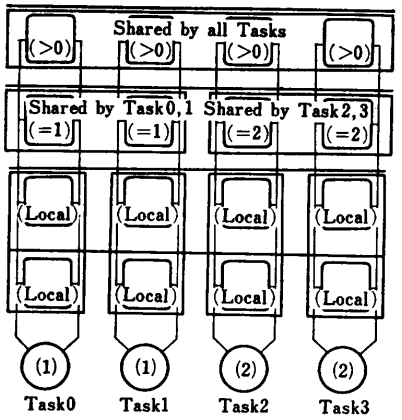
TCBの仕事はPEからのQ操作命令に応じてQの管理とタスクの状態の管理を行うことであるが，各Qに対するQ操作命令は排他的に実行されなくてはならない．このアーキテクチャではTCBをWRITEバスに接続し，PEからの操作命令により，ライト命令と同様にWRITEバスを専用し，他のライト命令に中断されることなく一つのQ操作命令を実行する．TCBは高速処理が要求されるため，マイクロプロセッサと高速メモリにより実現される．またTCBは各WRITEバスに接続され，これらは独立に動作することができるためQ管理の負荷は分散される．

RQはREADY状態になったタスクをできるだけPEの負荷が均等になるように，しかも高速に割り当てる必要がある．レディQの構成は図6のとおりで，TCBからのREADYタスクをバッファを介して各PEのバッファに送り込む．READYタスクのメッセージには図6に示す3種類がある．デディケーテッド型はタスクの実行されるPEが固定のものでメッセージの行先は決まっている．フリー型はどのPEで実行することも可能で，最も負荷の少ないPEへ送られる．ブロードカスト型は，FOR ALLやSIMDを実現するもので，全PEのハードウェアQにブロードカストされる．ブロードカスト型メッセージによる仕事の終了は，終了したタスクからのPUT MQ命令を受ける終了確認タスクによってソフトウェア的に検出することもできるが，全PEの仕事がなくなったとき，またはある{PEが終了命令を発したとき，ハードウェア的に検出し，JOINというシステムタスクを起動することもできる．

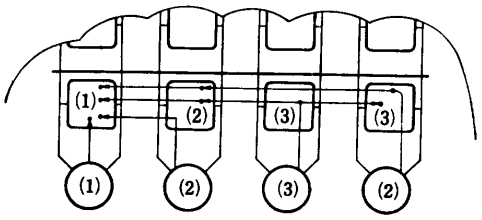
可変構造共有メモリ

高級言語で書かれたプログラムを効率よく実行するには，タスクやデータ構造を効率よく扱えることが重要であり，ここでは共有メモリに可変性を与えて対処する．ここで示す可変構造はマルチリードメモリでの衝突減少やメモリの有効利用にも役立つ．

可変性は図1に示したアドレス変換部(AC)とMEに付属した論理回路（L）によって実現される．



(a) Partitioning



(b) Hierarchical structure

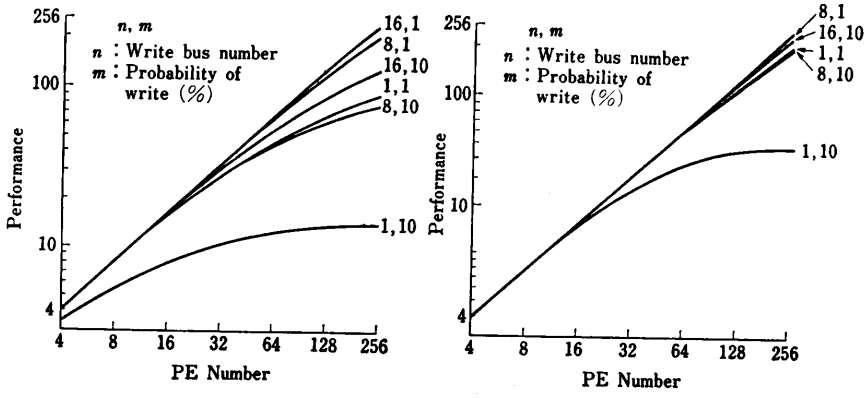
図 7 可変構造共有メモリ
Fig. 7 Reconfigurable shared memory.

まず第1の可変性は，ME のアドレス方式である．アドレス方式にはブロック型とスライス型がある．ブロック型はアドレスを ME 内の隣り合った語に連続的に割り付けるものであるのに対し，スライス型は，隣り合ったアドレスが隣の ME になるようにインタリーブ方式でアドレス付けする．このアドレス変換は AC により行われる．

第二の可変性は，Lを利用して共有メモリを領域分けするもので，階層構造の実現が容易である．各 ME とライト命令には優先度を付加することができる．ME の優先度は優先番号と比較記号から成り，これが WRITE 命令のもつ優先度と L で比較され ME への書込みが制御される．ME に書込みを許さない優先度（図7（a）の LOCAL）を与えておくと，その ME は WRITE バスと切り離され，いわゆる PE のローカルメモリとなり，他の PE と独立に読み書きが可能となる．図7（a）は優先度（図中の（ ）内の番号）による領域分けを行っている例であり，（b）はタスク間の書込み権を制御する例で，これらにより階層構造が容易に実現できる．

4. マルチリードメモリのシミュレーション

このシステムではマルチリードメモリでの衝突が性能に大きな影響を与えると思われる．また設計に当たっては，PE 数と WRITE バスの本数が重要な設計パラメータである．そこでここでは，マルチリードメモリを用いたシステムの性能をシミュレーションで求めた．シミュレーションモデルはハードウェア構成が図1と同じであり，各 PE は実行とメモリアクセスを交互に繰り返すものとする．メモリアクセスにはリードとライトがある．全 PE は同時にリードを実行することができるが，いずれかの PE がライトを行っている場合にはメモリアクセスは待たされる．シミュレーションでは，メモリアクセス時間を一単位時間として，実行時間，リードとライトの発生比率，PE 数および WRITE バスの本数をパラメータとして PE の



(a) Average exec. time is 1 unit time. (b) Average exec. time is 5 unit time.
(1 unit time: memory access time)

図 8 マルチリードメモリのシミュレーション結果
Fig. 8 Simulation results of multi-read memory.

稼動率の和を求めた．**図8**がシミュレーションの結果である．シミュレーションパラメータの値は，応用により大きく異なる．一般にライト命令はリード命令に比べて発生頻度が低いことが知られており，行列演算等ではさらに低くなる．シミュレーション結果から，ライトの比率が高い応用ではバスの本数とPEの数の比が1対8，ライトの比率が低い場合には1対16位までPE数を増やせることがわかる．

5. むすび

この論文では，並列計算機システムを，並列プログラミング言語用高級言語計算機という立場で設計することの重要性に着目し，並列プログラムの並列構造を系統的に表現するモデルを与えるとともに，それを効率良く実現できるアーキテクチャを提案した．これは並列計算機のセマンティックギャップの問題を解決する有力な方法と考える．

ここで示したアーキテクチャは，リモートセンシング画像データを前処理から後処理まで一貫して処理できることを目的に設計されたもので，マトリックス演算等の高速処理を最も得意とするが，汎用性を有する構成となっているため，その他にも幅広く応用可能であると考えられる．また，このアーキテクチャはモデルの構造を反映しているが，モデルから一意的に決まるものではなく，他のアプローチも考えられよう．

最後のシミュレーションでは，PE数とWRITEバスの数の割合が求まった．PE数の制限は，WRITEバスの高速駆動の可能性の面から決まってくると思われるが128台ぐらいまでは十分高速に働くと思われる．PEにキャッシュメモリを用意すれば，さらに性能向上が期待できよう．

謝辞　本研究をまとめるに当たってご検討いただき貴重なご意見を賜った飯塚肇成蹊大学教授，藤井狷介前計算機方式研究室長（現，明電舎），日頃ご指導いただく弓場敏嗣計算機方式研究室長ほか研究室の皆様，そして研究の機会を与えられた柏木寛電子計算機部長に感謝の意を表します．

参考文献

1) 古谷立美：マルチプロセッサシステムにおけるConcurrent Pascalマシン，情報処理学会論文誌，Vol. 21, No. 2 (1980).
2) Ahuja, S. R. and Asthana, A.: A Mult-Microprocessor Architecture with Hardware Support for Communication and Scheduling, *Sigplan Notice*, Vol. 17, No. 4 (Apr. 1982).
3) DoD: Rationale for the Design of the ADA Programming Language, *Sigplan Notice*, Vol. 14 (Jun. 1979).
4) Nakagawa, T. et al.: A Multi-Processor Approach to Discrete System Simulation, Comp. Con. (Spring, '80).

(昭和57年7月23日受付)
(昭和57年11月8日採録)