並列高級言語計算機の研究

古 谷 立 美

(電子技術総合研究所)

昭和５９年 ９ 月

論文要旨

本研究はマイクロプロセッサレベルの計算機を多数結合して汎用並列計算機を実現する方法を確立するために行ったものである。ここで汎用並列計算機とは任意の並列プログラムを並列実行できる計算機を指す。汎用並列計算機を実現する方法として本論文で提案するものは、並列高級言語計算機と呼ぶもので、並列計算機を並列記述用高級言語の高級言語計算機として設計する方法である。これは従来独立に行われていた高級言語計算機と並列計算機の研究を融合し新たな能力を引き出すもので、並列計算機の障害と考えられている並列プログラムの問題は高級言語で解決され、ソフトウェアとハードウェアの間のセマンティックギャップは高級言語計算機で解決される。

本論文は並列高級言語計算機を設計する系統的方法を示すものである。並列高級言語計算機を実現するために必要な研究開発課題として本論文で扱う主なテーマは（ｉ）並列計算機構成決定のための性能解析と評価システム、(ii)マイクロプログラムによる高級言語計算機の実現、(iii)高級言語計算機を構成するマイクロ命令の使用特性、(iv)並列高級言語の並列制御機構を並列計算機の機械語に組込む技術、等である。

これらを扱う本論文の構成は以下のとおりである。

第１章は、本研究の目的、意義について述べる。

第２章は、並列計算機と高級言語計算機を中心にサーベイを行い、本研究の並列高級言語計算機というアプローチと、その実現に必要な研究課題の導出を行う。

第３章は、並列計算機の構成を決定する評価法を示すもので、ここではバス結合並列計算機を実際のハードウェアに近いレベルで解析するシステムと、任意の並列計算機のシミュレータをモジュール化と階層化を用いて構成できるシミュレーションシステムを示す。又ここではそれらのシステムで行った並列計算機の評価結果を示す。

第４章は、マイクロプログラムによる高級言語計算機の実現法を示すもので、並列プロセスをサポートするために開発した中間言語とその評価を示す。又ここでは高級言語計算機を実現するマイクロ命令の使用頻度をもとにユニバーサルホストプロセッサのあるべき姿を議論する。

第５章は、並列記述用高級言語の並列構造をプロセスのキューイングネットワークで表現し、並列記述用高級言語の仮想計算機を構成する方法を示す。さらにここではその仮想計算機を可変構造マルチリードメモリとハードウェアキューを用いて実現するアーキテクチャを示す。

最後に、第6章では本研究の成果を要約すると共に、本研究の経過と将来への展望を述べる。

目　次

第1章　序論

1．1　本研究の目的と意義

電子計算機は１９５０年代に実用化が始まって以来、”より速く”、”より使い易く”、を目指して驚異的な発展を続けている。この発展に貢献してきた研究開発要素を整理すると次の様になる。まず処理速度の点では素子の能力に大きく依存してきた。素子は真空管、トランジスタ、ＩＣ、ＬＳＩと発展し、計算機の処理速度を飛躍的に増大させてきた。しかし現在素子速度の向上は理論的限界に近付き、更に処理速度を上げるためには、従来の逐次処理方式の枠を破る並列処理方式のアーキテクチャが不可欠と考えられている。

一方計算機の使い易さの面では、高級言語とＯＳの進歩が重要な役割を果してきた。そしてこれらの技術を効率良くハードウェアでサポートするアーキテクチャの研究として高級言語計算機やＯＳのハードウェア化の研究がある。

並列計算機は計算機の歴史の初期の頃から色々な提案が行なわれてきたが、初期のものはほとんどがペーパマシンである。しかし、近年半導体技術が急速な進歩を遂げ高集積化が可能になるに伴い、様々な並列計算機が作られてきている。並列計算機を基本処理要素のレベルで見ると、ＡＬＵを多数結合する低レベルのものから大型計算機を結合する高レベルのものまで多種多様である。これらの内最近のＶＬＳＩの進歩をよく反映しているものの代表としてマイクロプロセッサレベルの計算機を多数結合するタイプの並列計算機がある。これはＭＩＭＤ型処理を中心に価格／性能比の良い並列計算機の構成法であると共に、汎用並列計算機実現の有望なアプローチと考えられている。１９７０年代初頭からこのような並列計算機は各所で作られ、専用機の分野では一設計法としての地位を確立している反面、汎用機の分野でまだ実験の域を出ず、実用化の見通しが立っていない。ここで汎用並列計算機とはユーザが任意の並列プログラムを書き、これを並列処理できる並列計算機を指している。

世の中には電子機器の制御やＯＳ等元来並列記述を行い並列処理するのが望ましい応用分野は多いが、従来並列プログラムを記述できる高級言語がなかったり、並列計算機がなかったためシーケンシャルなプログラミングが行なわれてきた。しかし今後益々高速化が要求されるようになると高級言語による並列記述とそれを並列処理するという自然な形態が要求されることは疑いない。そしてそのような環境を与えるものとして汎用並列計算機の存在は不可欠である。本研究はマイクロプロセッサレベルの計算機を多数結合して汎用並列計算機を構成する方法を確立するために行ったものである。

汎用並列計算機が良い成果を上げていない大きな理由は二つ考えられる。第一はプログラミングの問題である。最近まで並列プログラムを構造的に記述できる高級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難しく、生産性、保守性等に多くの問題があった。しかし最近になって、並列プログラム用高級言語の研究は急速な進歩を見せ、有望な言語が次々と提案され、並列記述の概念も整理されてきた。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、並列計算機の設計者は並列計算機を独立性の高いシーケンシャル計算機の集合体ととらえ、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を与えた。そのため並列プログラムを走らせるには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。このような問題を解決する方法として本論文で提案するのは並列計算機を並列プログラミング言語用高級言語計算機として設計するものである。これは従来、並列計算機を独立性の高い計算機の集合体と考えＯＳでプロセス間通信や同期をサポートしてきたのに対し、並列計算機を１台の計算機としてとらえ、並列プログラミング言語の持つ並列制御機構を並列計算機の機械語として取り込む方式である。この方法を用いれば、ＯＳの内プロセスの並列動作を制御する部分が不要となり効率良い処理が可能となる他、並列計算機上で走るプログラムも高級言語で書くことができるため並列計算機のプログラムの問題も同時に解決されることになる。これは従来独立に行われてきた並列計算機と高級言語計算機の研究を融合し新しい能力を引き出すものである。

以上述べてきたように、本研究の目的はマイクロプロセッサレベルの計算機を多数使って汎用並列計算機を実現する方法を確立することにある。この目的を達成する一つの方法として筆者が提案するのは、並列高級言語計算機と呼ぶところの“並列計算機を並列プログラミング言語用高級言語計算機として設計する”という考え方であり、本研究ではこれを実現する系統的な方法を確立した。本研究では、まず並列アーキテクチャの性能評価法を確立すると共に評価を行い、次にマイクロプログラムによる高級言語計算機の構築法を開発し、最後に並列プログラム用高級言語の並列構造を系統的に表現し、それをハードウェア化していく方法を示した。　この論文は、並列計算機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から系統的に設計する一つの方法を示したもので、従来汎用並列計算機の障害となっていた大きな問題を解決したもので、並列処理技術の確立に大いに貢献するものと確信する。

１．２　本研究の構成と範囲

並列高級言語計算機の確立には色々な側面からの研究が必要である。それらは互いに関連を持ち必ずしも独立ではないが、主なものとしては次のようなものがある。

（１）並列計算機構成

並列計算機はハードウェア先行で開発が行われてきたため、並列計算機の構成法や制御法に関する個別のアイデアは色々提案されてきたが、応用を考慮してそれらをどう組み合せ構成を決定していくかという問題は未解決である。そのため現在並列計算機構成に関して最も重要な研究開発課題と考えられているのは、並列計算機の性能評価を行うシステムや並列計算機設計時に必要なシミュレーションシステムの開発である。

（２）並列プログラム用高級言語

並列プログラムを構造的に記述でき、同期等に関する誤りをできるだけコンパイル時に発見できる高級言語が必要である。

（３）マイクロプログラムによる高級言語計算機

マイクロプログラムによる高級言語計算機の実現にはマイクロプログラマブルプロセッサとその上で走るマイクロプログラムが必要である。マイクロプログラムを用いた高級言語計算機の実現法は既にいくつか提案されているが、並列プロセスを扱う高級言語計算機の実現は例がなく、解決を必要とする新たな問題が存在する。又、マイクロプログラムによる高級言語計算機の実現は色々行われているが、その評価、高級言語計算機とマイクロプログラムの関係、マイクロ命令の使用頻度等に関する定量的データは不足している。これらのデータはマイクロプログラマブルプロセッサの設計や、マイクロプログラムによる高級言語計算機の製作に欠くことのできないものである。

（４）並列制御構造

並列プログラム用高級言語の並列制御構造を効率良くサポートする機構を開発し並列計算機を１台の高級言語計算機として設計する手法を確立する必要がある。

以上の様な考えられる項目の内、（２）の言語は最近の研究の進歩で十分使用に耐えるレベルに達しているという認識に立ち、これを除いたアーキテクチャに関する三つの面を取り上げて研究開発を行った。

まず（１）に関してはバス結合並列計算機の新しい解析法を示しそれを使ったいくつかの

解析結果を示すと共に、様々な並列計算機のシミュレータを容易に構成して性能評価のできる新しいシミュレーションシステムを示す。筆者は工業技術院大型プロジェクトパターン情報処理システムにおいてマイクロプログラマブルプロセッサ（ＰＵＬＣＥ）とそれを処理要素とする並列計算機（ＡＣＥシステム）の開発に参加してきた。（３）に関してはＡＣＥシステム上への並列Ｐａｓｃａｌマシンの実現を通して並列プログラム用高級言語計算機のマイクロプログラムによる実現法を示す。並列プロセスを扱う高級言語計算機ではプロセスの制御を行う部分があるためマイクロプログラムが膨大になる。ここでは、これをさける高級言語計算機の実現法を示すと共にその評価を示す。又、ＰＵＬＣＥを用いて作られた様々な高級言語計算機のマイクロプログラムを分析し、マイクロ命令の使用特性を明らかにし、マイクロプログラムと高級言語計算機の関係を示すと共にＰＵＬＣＥアーキテクチャについて考察を加える。（４）に関しては、並列プログラム用高級言語の並列構造を系統的に表現する新しいモデルを与えると共に、このモデルを効率良く実現するアーキテクチャを提案する。これにより、並列計算機は１台の高級言語計算機として系統的に設計することが可能となる。

以下、各章の内容を概説する。

まず、第２章では並列処理と高級言語計算機についてサーベイを行い並列高級言語計算機の諸問題を明確にし研究開発要素を整理する。

第３章ではバス結合並列計算機の解析法と解析結果、及び並列計算機用シミュレーションシステムについて述べる。

第４章は並列計算機上に実現した並列Ｐａｓｃａｌマシンをもとにマイクロプログラムによる仮想命令の解釈を議論する。又、高級言語計算機を構成するマイクロ命令の使用特性の測定結果と評価を示す。

第５章は並列プログラムの並列制御構造を記述するモデルとそれを実現するアーキテクチャを提案し、その評価結果を示す。

最後に、第６章では本研究の成果を要約すると共に、本研究の経過と将来の展望を述べる。

第2章　基礎考察

2．1　まえがき

既に述べたように、計算機は“より速く”、“より使い易く”を目指して発展を続けている。そして今後このような発展を支えていくアーキテクチャ技術として並列処理と高級言語計算機が重要な役割を果すことは疑いない。

本章ではこれらの点を中心に簡単なサーベイを行い、現在それらが持っている問題点を整理、検討し、本論文の主題である並列高級言語計算機の必要性を示すと共に、これを実現するに必要な研究開発のポイントを示す。

2．2　並列計算機

並列計算機の統一された定義は存在しないが、一般に広義には“各機能ユニットやプロセッサユニットが独立に、同時に動作して一つの仕事を行う計算機システム”と考えられている。並列計算機はいろいろな側面を持っているため種々の分類が考えられるが、計算機システムの歴史と特徴を良く反映しているものとしてＨｏｂｂｓの分類[Hob70]がある。表２．１はＨｏｂｂｓに従って主な並列計算機を分類したものである。　表２．１から分るとおり一般に並列計算機と言った場合非常に多岐に渡るが、この論文で扱う並列計算機はミニコンピュータやマイクロコンピュータ規模のコンピュータやプロセッサを密結合した性能／価格比の高い計算機システムを主に考えており、Ｈｏｂｂｓの分類によればＴｙｐｅ１の多重計算機と多重プロセッサにほぼ対応する。但し、扱う対象としてＴｙｐｅ２，Ｔｙｐｅ３への応用も有りうる。以下本論文で並列計算機という場合ミニコンピュータやマイクロコンピュータ規模のコンピュータやプロセッサを密結合した計算機システムという狭義の意味で使用する。

本節では、まず並列計算機の研究開発の歴史を振り返り、現在の並列計算機の状況を明確にする。次に、アーキテクチャ利用形態等で、現在抱えている問題点を整理し、今後必要な研究課題を示す。

2．2．1　研究開発の経緯と背景

半導体技術が進歩し、高機能計算機がミニコンピュータやマイクロコンピュータという小型で安価な形で入手できるようになるに伴い、それ等を結合して並列計算機を構成しようと

表 2．1　Ｈｏｂｓによる並列計算機の分類

Ｔｙｐｅ１ Multicomputer and Multiprocessor	Ｔｙｐｅ２ Associative Processor	Ｔｙｐｅ３ Parallel Net work of Array Processor	Ｔｙｐｅ４ Functional Machine
IBM9020	Rosin-Associative	Holland Machine	IBM360/91
RW-400	Cryogenic Computer	Solomon Machine	CDC 6600
Pilot		Unger Machine	CDC 7600
IMP	Librascope's Associative Parallel Processor	Knapp-Iterative Array Computer	RCA LIMAC
Burroughs D825		Conway's Machine	Bull Gamma 60
Multics	Davies Assotiative Processor	ILLIAC IV	CRAY1
Hughes3118		Automatic's Distributed Processor	
C.mmp	Hughes Associative Processor		
Cm*	DAP	PACS	
ACE	STARAN	MPP	

	1972	1974	1976	1978	1980	1982
バス		MICS(日電) ACE(電総研) Puluribus	MINERVA(Minesota)		EPOS(東芝) BTI8000 iAPX(INTEL)	
スイッチ	C.mmp(CMU)			PASM(Purdue)		
マルチポートメモリ	PRIME(UCB)				KDSS(慶大)	
ネットワーク				Xtree(UCB) Tandem16	CORAL(徳大) PACS(筑大)	

図 2．1　並列計算機の歴史

いう動きが出てきた。この最初のものは１９７２年のＣ．ｍｍｐ[Wul72]とＰＲＩＭＥ[Bas72]である。Ｃ．ｍｍｐはマルチプロセッサに関する多くの知識を得ることを目的に設計されたものでありＰＲＩＭＥは高信頼性を目指して設計された。その後１９７５年頃から並列計算機の研究は本格化し、ＡＣＥ[Iiz75]，ＭＩＳＣ[Com74]，ＭＩＮＥＲＶＡ[Wid76]，ＥＰＯＳ[Mae79]，Ｃｍ＊[Swa77]，ＢＴＩ８０００[Lew79]等次々と発表された。そして、これらのほとんどは汎用計算機を目指しており、研究的要素の強いものであった。図２．１はこれらの計算機の製作時期と結合形態を示している。１９８０年頃からは、それらの成果や反省をもとに実用期に入り、主に図形処理用やシミュレーション用等専用機の形で製作されてきている。このように並列計算機は専用計算機の一設計法としての地位を確立しつつあるのに対し、汎用計算機の方面では実用化が進んでいない。並列計算機が汎用機の分野で成果を上げていない大きな理由の一つはソフトウェアにあると考えられる。世の中にはＯＳや電子機器制御プログラム等元来並列記述を用いた方が自然な応用が沢山有るにもかかわらず、並列プログラムを実行できる計算機が無いことや並列アルゴリズムをマシンインデペンデントに記述できる高級言語がなかったことから並列記述はほとんど行われてきていない。しかし、今後高速性を追求していく場合には汎用並列計算機のアプローチが不可欠になってくることは明らかである。そしてこれを実現するためには、並列記述を行う高級言語とそれを効率良く実行する並列計算機が必要と言えよう。

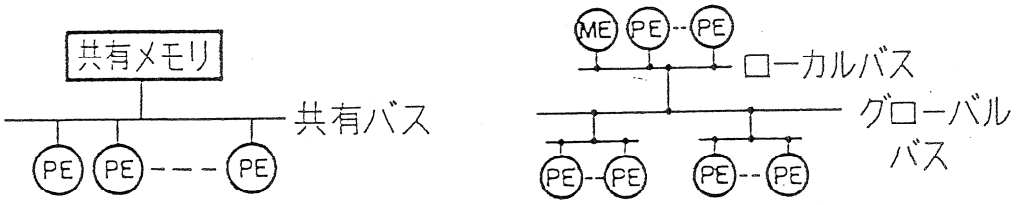
２．２．２　アーキテクチャとその評価

並列計算機アーキテクチャを特徴付けるものとしては、計算機結合形態、制御方式、通信方式などがある[Han73]。そしてこれらの組み合せによりでき上がる並列計算機の構成は多数存在する。さらに計算機の種類や、ソフトウェアを考えると、並列計算機の形態は多数存在する。

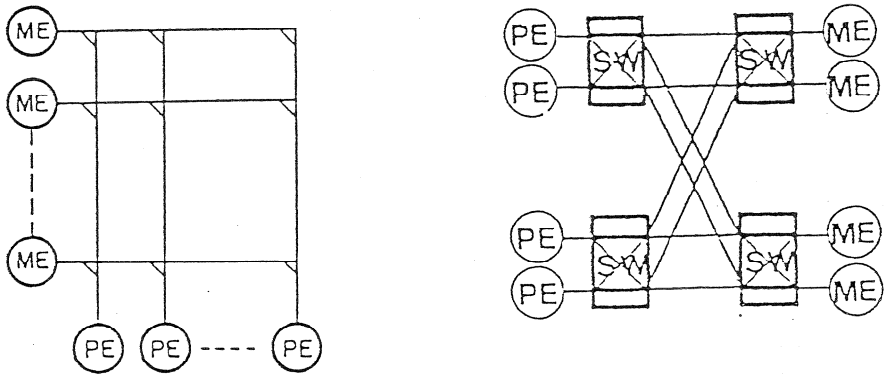
図２．１は前項で述べた並列計算機を計算機結合形態に関して分類したもので、図２．２は主な結合形態を図示したものである。これらの結合形態の特徴は一般に次のように考えられている[Tak82]。

（１）バス方式：融通性の点で優れているが、プロセッサ数が増えるとバスにおける衝突による性能劣化が問題となる。これを避ける方法としてＣｍ＊のクラスタ方式やＥＰＯＳのマルチバス方式がある。

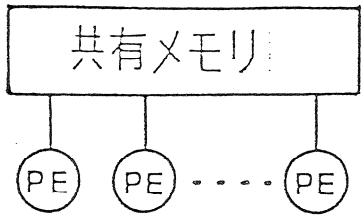
（２）スイッチ方式：これにはマトリクススイッチや置換スイッチ方式が含まれる。マトリ



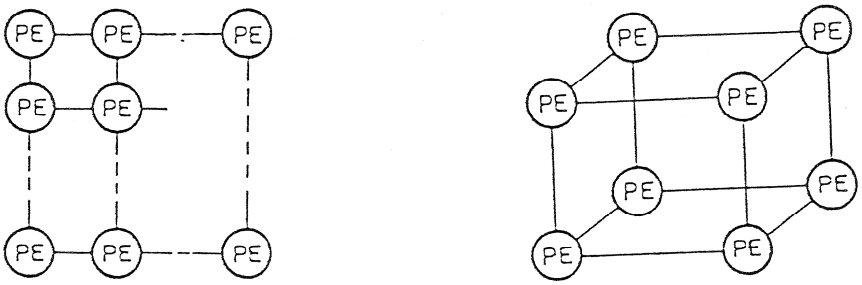
（a）バス方式



（b）スイッチ方式



（c）マルチポートメモリ方式



（d）ネットワーク方式

図 2．2　並列計算機の結合形態

クススイッチはデータ転送の際の衝突が無くなる利点がありＣ．ｍｍｐで用いられたが、プロセッサ数が増えるとスイッチの量が膨大になる。スイッチ量を減らす方式としてオメガネット等に用いられた置換スイッチ方式がある。

（３）マルチポートメモリ：これはメモリに複数のポートを付け、各ポートにプロセッサを結合するもので、論理的には同時読み出し、書き込みが可能となる。しかし実際にはメモリでの衝突による遅れが避けられない。ＫＤＳＳなどはこれをマルチリードに制限して高速化を図っている。

（４）ネットワーク方式：これはプロセッサ同士を直接結合するもので最近傍結合方式やキューブ方式に代表される。この方式はハードウェアの構造と応用の構造が合った場合に有効に働くため専用機には適するが汎用機には不向きと考えられる。

このように並列計算機を構成する結合形態や制御方式に関する提案は多数行われているが、それらの性能や評価を裏付けるデータはあまり得られていない。並列計算機の性能についてのデータが少ない理由は二つ考えられる。第一は従来並列計算機が実用レベルから遠いため、そのようなデータがあまり必要なかったこと。第二には、実際のマシンを考える場合に一般的データだけでは不足で、応用プログラムを考慮したより詳細なデータが必要なためである。今後並列計算機が広く用いられるようになると並列計算機に対する一般的データと共に、システム設計時に応用を考慮した解析やシミュレーションを行うことは常識化して来る。現在このような要求に応えようとしているものとしてはバス結合並列計算機を解析するＲｅｙｌｉｎｇのもの[Rey74]やＬｅｖｙのシミュレータ[Lev73]があるが、Ｒｅｙｌｉｎｇの解析モデルは仮定が単純すぎて実際のシステムとは隔たりが大きすぎる。又Ｌｅｖｙのシミュレーションシステムは対象とするシステムが特定されており、種々の並列計算機をシミュレートすることはできない。このような事情から今後並列計算機を構築するためには、より現実に近い解析を可能にする解析法やシミュレーションシステムの開発が不可欠である。

２．２．３　並列計算機の利用技術

並列計算機の研究はハードウェア主導型で進められてきた。これは半導体技術の急速な進歩に、利用技術が追い付かなかったものといえよう。前項で取り上げた並列計算機を並列処理の形態という点から見てみると次の三つに分かれる。第一は、各計算機の独立性が高いものである。一般に並列性を高信頼性の実現に利用しようという場合には、各計算機が独立の

仕事を行うものが多い。ＰＲＩＭＥやＥＰＯＳでは各ＣＰＵが独立の仕事を行っている。第二は、各計算機が低い頻度で交信を行い仕事を行うことを前提に設計されたもので、Ｃ．ｍｍｐやＣｍ＊がこの例である。これらの応用に対しては、並列計算機用ＯＳが研究開発されてきている。この代表はＣ．ｍｍｐのＨｙｄｒａ[Wul75]とＣｍ＊用のＳｔａｒＯＳ　Ｋｅｒｎｅｌ[Jon77]である。ＨｙｄｒａはＯＳから基本機構を分離し、様々なＯＳを容易に構成できるようにしている。又、ここで用いられたＣａｐａｂｉｌｉｔｙの考え方はＳｔａｒ　ＯＳ　Ｋｅｒｎｅｌにも踏襲されて一つのＯＳ構成法として認められている。しかし汎用システムを対象にしたＯＳは規模が大きくなったり、オーバーヘッドが増大するという点は避けることができない。第三は、各計算機の依存関係が高い場合である。これはプログラミングが非常に難しいという難点があるため、専用機を除いてほとんど扱われていない。この種の応用は、時間に依存する誤りが発生しやすく、高級言語により誤りをコンパイル時にチェックする方法を採らない限り汎用計算機としては利用できない。

２．３　高級言語計算機

並列プログラムの難しさは時間に依存した再現性のない誤りの検出である。このような誤りをコンパイル時に発見するには高級言語が不可欠である。本節では並列プログラム記述用高級言語の進歩を振り返ってみると共に、高級言語化によって生ずる高級言語とハードウェアの間のセマンティックギャップを埋める高級言語計算機の技術を考察する。

２．３．１　並列プログラム用高級言語

高級言語は計算機の使い易さに最も貢献しているものの一つである。高級言語で書かれたプログラムはアセンブリ言語で書かれたものに対して２０～３０％性能が落ちると言われるが、生産性、保守性、管理性で優れていることから、今日では高級言語によるプログラムがＯＳやマイクロコンピュータの分野でも一般化している。高級言語の発展の流れを見るとＦＯＲＴＲＡＮなど初期に開発されたものはハードウェアの構造の影響を大きく受けているのに対し、最近ではアルゴリズム記述という立場に重点をおいて、より書き易いＰａｓｃａｌ系言語が主になってきている。

この流れは並列プログラム用高級言語にも通ずるものである[Fur80b]。並列プログラムの記述も最初はシーケンシャルな言語にプロセス間同期用プリミティブを加える形で出発した。ｓｅｎｄ／ｒｅｃｅｉｖｅ、ｌｏｃｋ／ｕｎｌｏｃｋ、ｔｅｓｔ＆ｓｅｔ、Ｐ・Ｖ操作*などはこの代表である。これらは、シーケンシャルなプログラミング言語でいえばアセンブリ言語に対応する。プログラムを簡潔で理解し易くし、誤りをできるだけコンパイル時に検出する高級言語にとってはより高レベルの言語概念が必要である。この基礎となったのはＤｉｊｋｓｔｒａのｃｒｉｔｉｃａｌ　ｒｅｇｉｏｎ　であり、Ｈｏａｒｅのｃｏｎｄｉｔｉｏｎａｌ　ｃｒｉｔｉｃａｌ　ｒｅｇｉｏｎである。Ｈｏａｒｅはこれをさらに整理しＭｏｎｉｔｏｒ[Hoa74]という概念を生みだした。世界最初の並列プログラム用高級言語はこのＭｏｎｉｔｏｒを用いた並列Ｐａｓｃａｌ[Bri75]である。Ｍｏｎｉｔｏｒはその後Ｍｏｄｕｌａ[Wir77]やＣＳＰ／Ｋ[Hol78]でも用いられた。Ｍｏｎｉｔｏｒが共有メモリを持った並列計算機に向いているのに対し、共有メモリを持たないタイプの並列計算機に向いた言語概念としては、Ｄｉｊｋｓｔｒａのｇｕａｒｄｅｄ　ｃｏｍｍａｎｄ[Dij75]　とＨｏａｒｅの　ｃｏｍｍｕｎｉｃａｔｉｎｇ　ｓｅｑｕｅｎｔｉａｌ　ｐｒｏｃｅｓｓ[Hoa78]　が提案された。これらを用いた並列プログラム用高級言語としてはＡｄａ[Dod79]がある。

＊　Dijkstraのセマフォア操作命令

このようにプログラムの記述の点では、ここ数年急激な進歩を見せているが、シーケンシャル言語の場合にもあったように、言語のレベルが高くなればなるほど、プログラム構造とハードウェア構造の距離は離れ、効率良い実行が難しくなった。これを補うアーキテクチャとしては、次項で述べる高級言語計算機がある。

２．３．２　高級言語計算機

高級言語を効率良く実行する高級言語計算機の提案は１９６１年のＡｎｄｅｒｓｏｎのＡＬＧＯＬマシン[And61]、ＢａｒｔｏｎのＢ５０００【Bar61]に始まる。図２．３は言語別の高級言語計算機の歴史であり、これから分かるように初期のものはペーパマシンが多い。実際のマシンが本格的に作られるようになったのは半導体技術の進歩によりハードウェアのコスト低下が実現する１９７０年代である。

高級言語計算機の構築法は大きく分けるとハードウェアによるものとファームウェアによるものに分かれる。ハードウェア化を行ったものの代表はＳＹＭＢＯＬ[Smi72]である。この方法は処理速度は早いが、言語仕様の変更のたびに論理設計をやり直す必要があるため、以下で述べるファームウェアによるものが広く用いられている。

マイクロプログラムは、もともと計算機設計法としてＭ．Ｖ．Ｗｉｌｋｓにより提案されたものである。これを高級言語計算機に用いる提案は１９６３年のＷｉｇｉｎｇｔｏｎ[Wig63]のマシンが最初である。実際にマイクロプログラムを用いて高級言語計算機が作られたのは１９６７年のＷｅｂｅｒによるＥｕｌｅｒのマシン[Web67]である。高級言語計算機の実現を目指すマイクロプログラマブル計算機はその後ＱＭ－１[Ros71]、Ｂ１７００、Ｂ１８００により一つの頂点に達した。しかしさらに半導体技術が進歩しマイクロプロセッサでマイクロプログラマブルプロセッサを作ることが可能になり、工業技術院大型プロジェクトパターン情報処理システムではＰＵＬＣＥと呼ばれるマイクロプログラマブルプロセッサを開発した[Iiz76a][Iiz79]。このような高級言語計算機実現ハードウェアの進歩を背景にその後ＦＯＲＴＲＡＮマシン、ＬＩＳＰマシン、Ｐａｓｃａｌマシンなどの高級言語計算機が各所で製作されてきた。そしてそれらのほとんどが高級言語用中間言語を設定してそのインタプリタをマイクロプログラムで製作するもので、多くの経験が積まれてきたが次のような未解決の点がある。（１）並列プログラム用高級言語計算機は開発されていない。（２）高速化をねらうとマイクロプログラム量が膨大になる。（３）マイクロプログラムを用いた高級言語計算機の実現は定着してきたにもかかわらず、その評価、特にマイクロ命令と高級言語計算機

の関係やマイクロ命令の使用特性は明らかにされていない。

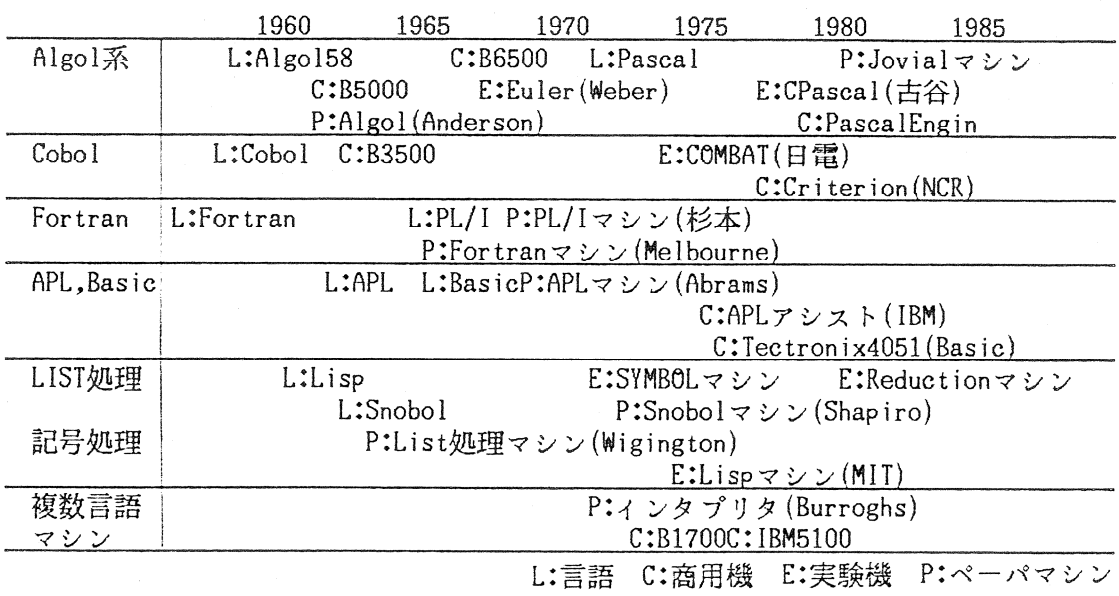


図 2．3　高級言語計算機の歴史

2．4　まとめ

以上の考察から次のような考え方が導かれる。まず、汎用並列計算機は高級言語で書かれた並列プログラムを並列処理するべきである。そしてこの考え方を実現する一つの方法として、筆者は従来個々の計算機の集合と考えられてきた並列計算機を、一台の並列プログラミング言語用高級言語計算機として設計する方法を提案する。この考え方が実現されると、並列計算機の言語の問題は高級言語で解決されるし、並列プログラミング言語の並列制御機構は並列計算機の機械語としてハードウエア化されるため、ハードウエアとソフトウエアの間のセマンティックギャップは小さくなる。

そしてこのような技術を確立するために並列計算機と高級言語計算機の分野で行わねばならぬ主な研究開発課題は次のような領域である。

まず高級言語計算機に関しては（1）マイクロプログラムによる高級言語計算機の実現法を示すと共に　（2）マイクロ命令と高級言語計算機の関係を明らかにする。並列計算機に関しては（3）並列計算機の性能解析、評価システムの開発と　（4）並列プログラム用高級言語計算機を実現するための並列制御機構の確立等がある。

本論文ではこれらの問題点に解を与える。以下三章で（3）を、四章で（1）（2）を、五章で（4）について述べる。

第３章　並列計算機の解析とシミュレーション

３．１　まえがき

第２章で示したように並列計算機の形態に関する提案は色々あるが、それらの評価データは少ない。並列計算機の構造を決定する場合には、その上で走るプログラムの構造との関係が重要で、応用と並列計算機の関係を評価する評価システムが必要がある。本章では並列計算機の性能測定と評価のために開発した二つの性能評価システムとそれらを用いて行った並列計算機に関する二、三の解析結果を示す。

最初の性能評価システムはバス結合された並列計算機の性能評価用解析モデルと解析法である。今まで並列計算機の解析法としては、Ｒｅｖｉｎのもの[Rev73]があるが、解析の対象となる並列計算機の構造は単純なもので実際のシステムとの隔たりが大きい。ここに示す解析モデルは種々のバス結合並列計算機システムをＰＭＳレベル*で解析できるもので、システム内部の負荷状態も求めることができる[Fur76]。解析法は、一階マルコフ過程の考え方に基づいて差分方程式を立て、それを計算機で解いていくものでＲｕｄｉｎが一つの待ち行列に対して示した解法[Rud71]を待ち行列が複数有り、かつそれらの間に相関関係がある場合のために拡張したものである。又ここでは、バス結合並列計算機に関する二、三の解析結果を示すが、それらは並列計算機設計時に特に重要な点で、従来設計者が勘で行っていた決定に一つの参考資料を与えることとなろう。

第二の性能評価システムは、いろいろな立場の並列計算機設計者が自分の要求に合った並列計算機のシミュレータを容易に作成し、その評価を可能にするシミュレーションシステムである。並列計算機のシミュレータとしてはＬｅｖｙのもの[Lev73]があるが、これは特定の並列計算機に対して作られたもので種々の並列計算機に適用することはできない。これに対し筆者が開発したシミュレーションシステムは任意の並列計算機のシミュレータを容易に構成できるものである[Fur77a]。本シミュレーションシステムから作られる並列計算機のシミュレータはユーザが意識することなく階層化とモジュール化が成されており、部分的変更が容易に行える。

以下３．２節にバス結合並列計算機の解析を、３．３節に並列計算機シミュレーションシステムを述べる。

＊　プロセッサ・メモリ・スイッチ　レベル：C.G.Bellの記法

３．２　並列計算機の解析モデルと解析

３．２．１　バス結合並列計算機

本節では、まず解析の対象となるバス結合並列計算機とそのモデル化及び解析法を示し、次にそれを用いた二、三の解析結果を示す。ここで解析の対象とするバス結合並列計算機は、ｂ本のバスにｐ台のプロセッサ、ｍ台のメモリ、ｉ台の入出力装置等が結合され、それらの間でデータ転送が可能なものである。即ち、世の中の大部分のバス結合並列計算機はこの範囲に入り、解析が可能となる。

３．２．２　モデルの設定と基本モデル

モデル化は、様々なタイプのバス結合並列計算機の性能解析を可能にするため、まず基本モデル（ＰＭ）と呼ぶ全体のモデルを構成するための基本単位を設定し、これを重ね合わせて目的とする被解析システムのモデルを作る方式を取る。以下にモデル化のための仮定と、ＰＭの構成を示す。

３．２．２．１　モデル化のための仮定

モデル化のための仮定は次のとおりである。

（１）システムの同期動作：システムの状態変化は、等時間間隔（Ｔｓ）の離散的時刻に発生する。

（２）プロセッサの状態：プロセッサは、図３．１に示すランとデータ待ちウェイトの２つの状態を持ち、以下に示す２種類のリクエストを発することにより状態が変化する。１リクエストによるデータ転送量は１単位データとする。

（ｉ）リードリクエスト：プロセッサがラン中に発するデータ要求で、プロセッサは要求したデータが到着するまでウエイト状態になる。図３．２（ａ）がリードリクエストに対するデータの流れで、メモリの待ちに入るものがデータでなく、リクエストである点がライトリクエストと異なる。

（ii）ライトリクエスト：プロセッサがラン中に発するデータ送出要求で、これが発生してもプロセッサの状態は変わらない。但し、プロセッサとメモリは、このリクエストに対するバッファを持ち、このリクエストが頻発してバッファサイズを越えたときは、オーバフローとして処理する。図３．２（ｂ）は、ライトリクエストによるデータの流れである。又、メ

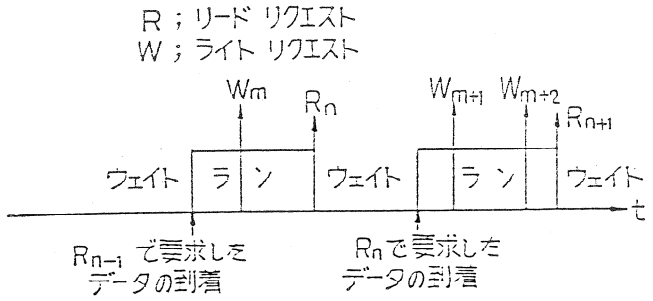
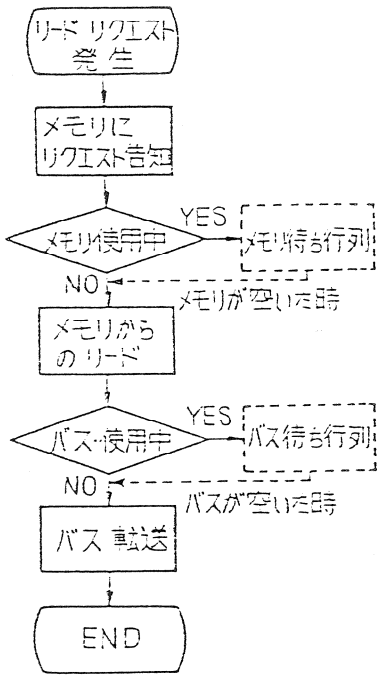
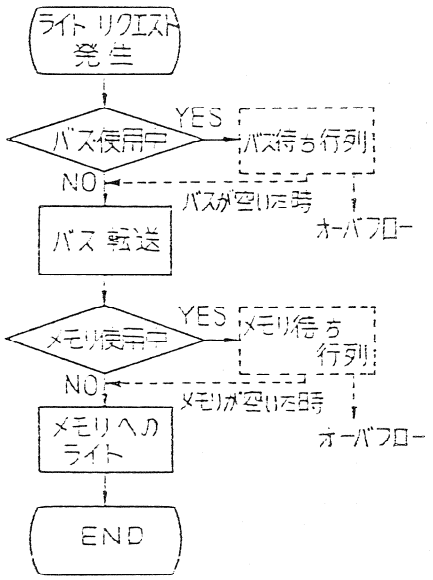


図 3．1　プロセッサの状態



(a) リード　リクエスト



(b) ライト　リクエスト

図 3．2　リクエストによるデータの流れ

モリアクセスの際アドレスがバスをへてメモリに送られるタイプのバス結合並列計算機では、アドレス転送をメモリへのライトリクエストと扱うことで処理可能である。

（3）データ転送リクエストの発生頻度: リクエストは、式（3．1）に示す二項分布に従って発生する。

$$B(n, N) = {}_{N}C_{n}\ \lambda^{n}\ (1-\lambda)^{N-n} \qquad (3.1)$$

ここでB（n，N）は、N台のプロセッサがラン状態のとき、単位時間間隔にn台のプロセッサがリクエストを発生する確率で、λは各プロセッサが単位時間に1リクエストを発生する確率。

（4）各種データ転送の統一処理: 各種リソース間、バス間のデータ転送は、プロセッサとメモリ間の転送に置き換えて処理する。

3．2．2．2　基本モデル（PM）

PMは、データ転送リクエスト頻度が等しく、かつm台の共有メモリのそれぞれに等確率でアクセスするプロセッサ群が、一本のバスで共有メモリに結合された構成に対して一つ設定される。図3．3はPMの対象となるハードウェア構成を、図3．4はPMを示す。

一つのPMは、リードリクエストとライトリクエストに対応した、リードリクエスト待ち行列系（PM−R_K，kはk番目のPMであることを示す。）とライトリクエスト待ち行列系（PM−W_K）の一対よりなり、この待ち行列系のそれぞれに3．2．3．1で述べる確率状態行列が作られる。図3．4でλr，λwはそれぞれ各プロセッサのリード及びライトリクエストの発生頻度を示す。

被解析システムが、図3．4に示すような構成で各プロセッサが同確率で共有メモリにリクエストを発するような単純な場合のモデル化は、一つのPMを作るだけで済む。図3．4（c）は、そのばあいのPMとハードウェアの対応関係を示している。即ち、図3．4（a）のメモリとバスは、それぞれ図3．4（b）のメモリとバスと同一のハードウェアであり実際のシステムでは、これら二つの待ち行列の内容が混りあって一つのハードウェアの待ち行列に入っている。そこでPMでは、（a），（b）それぞれの待ちの内容がどれだけTs間に処理されるかを示す確率変数としてOM，OBを導入している。この詳しい求め方は、式（3．5），（3．6）に示す。

又、解析されるシステムが複雑で、等頻度でリクエストを発するプロセッサ群が多数ある

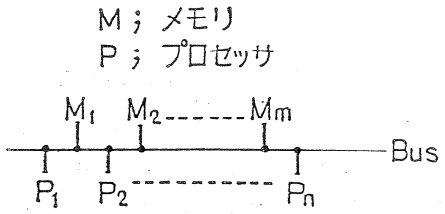
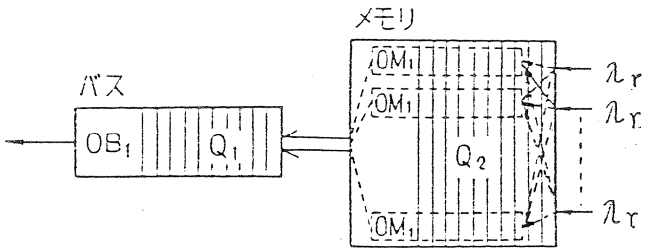
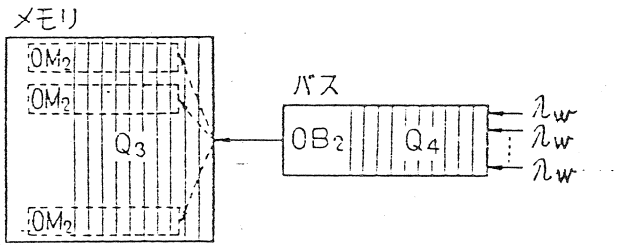


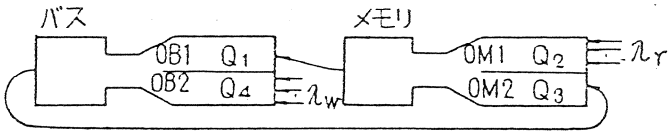
図 3．3　基本モデルのハードウェア構成



（a）リード　リクエスト 待ち行列モデル（PM－Rκ）



（b）ライト　リクエスト 待ち行列モデル（PM－Wκ）



（c）ハードウェア構造と基本モデルの対応関係

OM_n; 待ち行列 n (Q_n) のデータが T_s 間にメモリで処理される割合.
OB_n; 待ち行列 n (Q_n) のデータが T_s 間にバスで処理される割合.

図 3．4　基本モデル

場合などは、各プロセッサ群に一つずつＰＭを作り、それぞれのＯＢ，ＯＭを決定する時に、共有資源を使用する全ＰＭの待ち行列長を計算に入れることで処理できる。即ち図３．４（ｃ）で言えば、バスとメモリへ入る待ち行列の数がＰＭが増える分だけ増え、各待ち行列のデータが処理される割合が減ることになる。その他優先度のあるプロセッサ群の処理もＯＭ，ＯＢ決定に重みを付けることで可能である。なお以下では、図３．４のメモリとバスで処理中及び待ち行列中のデータとリクエストを、それぞれメモリ処理フェーズ，バス処理フェーズのデータとリクエストと呼ぶ。

３．２．３　状態方程式とその解法

３．２．３．１　状態確率行列と状態方程式

システム解析においては、システムの状態変数として、共有リソースの処理フェーズにあるデータ及びリクエスト数の確率を選び、状態方程式は、マルコフ過程の考え方で、差分方程式を立てる要領で行う。以下に状態方程式を立てる際必要な変数の定義と、ＰＭに対する状態方程式を示す。但し以下では、ｔを非負とし、変数名の添字ｋは、ｋ番目のＰＭを示す。

Ｔｓ：システムの状態変化の単位時間間隔。以下では、これをバスにおける一単位データ転送時間とする。

NP_K：PM_K のプロセッサの台数。

NM_K：PM_K のメモリの台数。

RM_K：一単位データのメモリでの処理時間と、バスでの転送時間の比。

$$RM_K = \frac{\text{一単位データのメモリでの処理時間}}{\text{一単位データのバス転送時間}}$$

又以下では、$RM_K \geqq 1$とする。

$IR_K(i, p, t)$：時刻ｔにｐ台のプロセッサがランの時次のＴｓ間にｉ台がリードリクエストを発生する確率で、式（３．１）に従う。但しｉ＞ｐまたはｉ＜０またはｐ＜０のとき　$IR_K(i, p, t) = 0$。

$$\sum_{i=0}^{P} IR_K(i, p, t) = 1 \qquad (3.2)$$

$IW_K(i, t)$：ｔ～ｔ＋Ｔｓ間にｉ個のライトリクエストが発生する確率。

$$IW_k(i,\ t) = \sum_{j=0}^{NP_k} RUN_k(j,\ t) \cdot IWT_k(i,\ j,\ t) \qquad (3.3)$$

$(i \geqq 0)$

ここで$RUN_k(j, t)$は、時刻tにj台のプロセッサがラン状態の確率、
$IWT(i, j, k)$は、時刻tにj台のプロセッサがランの時、次のTs間にi台のプロセッサがライト・リクエストを発生する確率で式（3．1）に従う。

$$\sum_{i=0}^{NP_k} IW_k(i,\ t) = 1 \qquad (3.4)$$

$OB(d, b, t)$；時刻tに、ある待ち行列系でb個のデータがバス処理フェーズにあるとき、次のTs間にb個中d個がバス上を転送される確率。

$$OB(1,\ b,\ t) = \begin{cases} \sum_{r=0}^{maxT} (b/b+r) \cdot OOB(r,\ t) & b \geqq 1 \text{のとき} \\ 0 & \text{その他} \end{cases} \qquad (3.5)$$

$$OB(0,\ b,\ t) = \begin{cases} 1 - OB(1,\ b,\ t) & b \geqq 0 \text{のとき} \\ 0 & \text{その他} \end{cases} \qquad (3.5')$$

ここで$OOB(r, t)$は、時刻tにバスを共有する他の待ち行列系のバス処理フェーズにあるデータの総和がr個である確率。

$OM(d, m, t)$：時刻tに、ある待ち行列系でメモリ処理フェーズにm個のデータ及びリクエストがあるとき、次のTs間にd個が処理される確率。

（$d \neq 0$のとき）

$$OM(d,\ m,\ t) = \begin{cases} 0 & d<0 \text{又は} m<0 \text{又は} d>min(NM_k,\ m) \text{のとき} \\ \sum_{S=0}^{maxS} R(d,\ m,\ s) \cdot OOM(s,\ t)/RM & \text{その他} \end{cases} \qquad (3.6)$$

$$OM(0,\ m,\ t) = \begin{cases} 1 & m=0 \text{のとき} \\ 1 - \sum_{d=1}^{Y} OM(d,\ m,\ t) & \text{その他} \end{cases} \qquad (3.6')$$

但し、$Y = min(NM_k,\ m)$．

ここで$OOM(s, t)$は、時刻tに自分自身の待ち行列系以外のリクエスト及びデータがメモリ処理フェーズにs個存在する確率。$R(d, m, s)$は、メモリ処理フェーズにある全データ及びリクエストのうち、m個が自分自身の待ち行列系のもので、s個が他の待ち

行列系のもののとき、自分の系のうちｄ個が処理される確率。

$$R(d, m, s) = \begin{cases} 0 & m \neq 0,\ s = 0,\ d = 0 \text{ のとき} \\ \sum_{w=d}^{W} OR(w, m+s) \cdot ({}_mC_d \cdot {}_sC_{w-d} / {}_{m+s}C_w) & \text{その他} \end{cases} \quad (3.7)$$

但し、W＝ｍｉｎ（ＮＭκ，ｓ＋ｄ）．

ここでＯＲ（ｗ，ℓ）は、ℓ個のデータ及びリクエストがｗ台のメモリに存在する確率。

ＰＭに対する状態変数と状態方程式は、図３．４（ａ），（ｂ）の二つの待ち行列系にそれぞれ設定する。状態変数は、各時刻に、待ち行列系（ＰＭ－Ｒκ，ＰＭ－Ｗκ）の各リソースの処理フェーズにあるデータ及びリクエスト数の確率とすることにより、次に示す状態確率行列ができる。

（ｉ）　ＰＭ－Ｒκの状態確率行列（ＰＲκ）

時刻ｔに、ＰＭ－Ｒκのデータ及びリクエストがメモリ処理フェーズにｍ個、バス処理フェーズにｂ個存在する確率ＰＲκ（ｍ，ｂ，ｔ）をエレメントとするＮＰκ＋１（ｍ＝０，１，・・・，ＮＰκ）行＊ＮＰκ＋１（ｂ＝０，１，・・・，ＮＰκ）列の行列である。但し、リードリクエストがプロセッサ数以上に発生することはないので図３．５（ａ）に示す三角行列になる。

（ii）ＰＭ－Ｗκの状態確率行列（ＰＷκ）

時刻ｔに、ＰＭ－Ｗκのライトリクエストによるデータが、メモリ処理フェーズにｍ個、バス処理フェーズにｂ個存在する確率ＰＷκ（ｍ，ｂ，ｔ）をエレメントとする行列である。ライトリクエストは、リードリクエストと異なり、リクエスト数に制限がないため、行列サイズは、無限大となり得るが、プロセッサとメモリに有限長バッファを導入し、データがバッファサイズを越えたときはオーバフローとして処理することにより、（メモリのバッファサイズサイズ（ＭＢ）＋１）行＊（ＮＰκ＊各プロセッサのバッファサイズ（ＰＢ）＋１）列となり、解析可能にしている。図３．５（ｂ）の白地部分がＰＷκで斜線部がオーバフローに相当する部分である。但し、オーバフローの考え方を導入することにより、オーバフロー分の確率は、行列のＭＢ＋１行とＮＰκ＊ＰＢ＋１列に集まるため、行列のエレメントの総和が１であることは保証される。又オーバフローは、バッファサイズを適当に選ぶことで、無視できる程度に減少する。

次に、待ち行列系の状態方程式を示す。

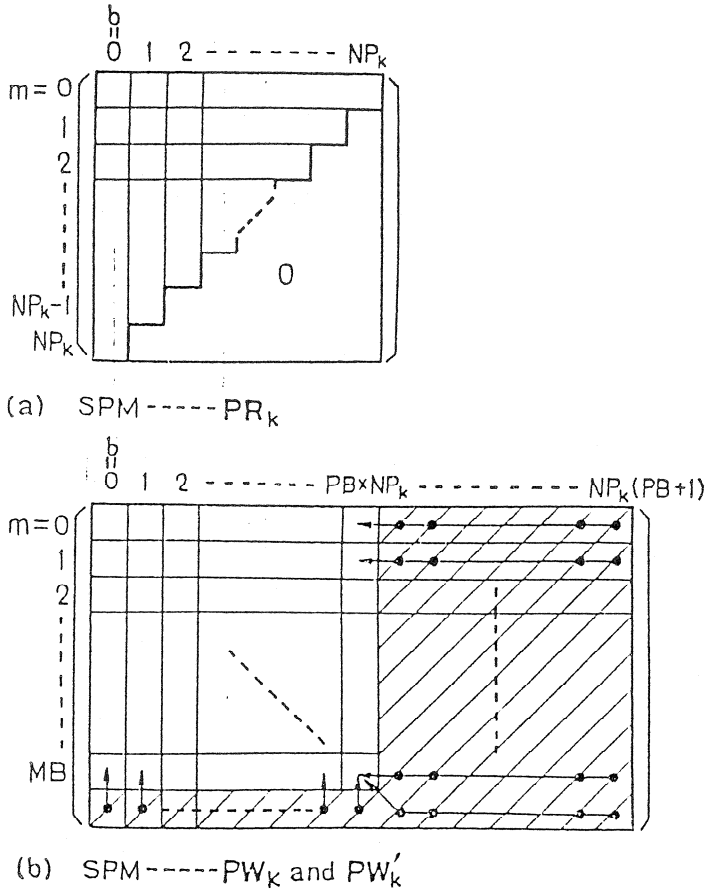


図 3．5　状態確率行列（ＳＰＭ）

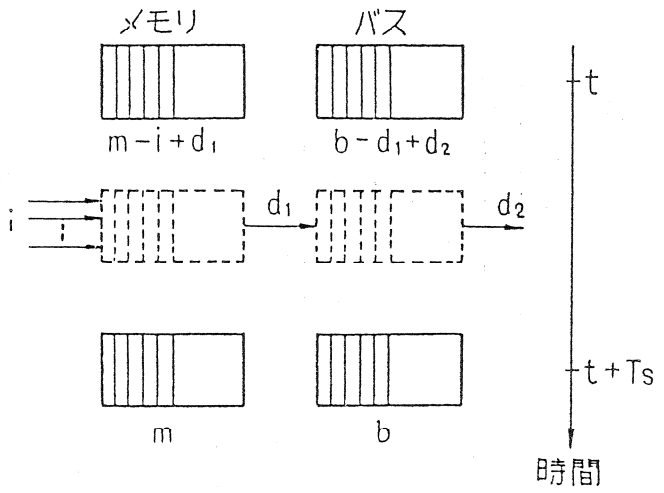


図 3．6　状態方程式の原理

(i) PM−R_k の状態方程式

$$PR_k(m, b, t+Ts) = \begin{cases} 0 & m<0 \text{ 又は } b<0 \text{ 又は } m+b>NP_k \text{ のとき} \\ \sum_{j=0}^{NM'} \sum_{i=0}^{m} & (PR_k(m+j-i, b-j, t) \cdot OM(j, m+j-i, t) \cdot OB(0, b-j, t) \cdot IR_k(i, NP_k-(m-i+b), t) + PR_k(m+j-i, b+1-j, t) \cdot OM(j, m-i+j, t) \cdot OB(1, b+1-j, t) \cdot IR_k(i, NP_k-(m+b-i+1), t)) \quad \text{その他} \end{cases} \quad (3.8)$$

ここで $NM' = min(NM_k, b)$

図3．6 は、$NM_k=1$，$RM_k=1$ のときの式（3．9）の解釈図で、時刻 t に PM−R_k のデータ及びリクエストが、メモリ処理フェーズに m−i+d1 個、バス処理フェーズに b−d1+d2 個あり、次の Ts 間に i 個の リードリクエスト が発生し、d1 個がメモリで処理されバスに送られ、d2 個がバスを通してプロセッサに送られた時、t+Ts 時のメモリ及びバス処理フェーズのデータ数は、それぞれ m，b となることを示している。

(ii) PM−W_k の状態方程式

ライトリクエスト 系に対する状態方程式も同様の方法によるが、 データ オーバフロー の処理が必要である。ここでは、式が直観的に理解し易いこと、計算機での処理に都合がよいという理由から、オーバフローを許す確率行列 PW'_k を定義し、オーバフローに相当する確率を MB+1 行と NP_k * PB+1 列に集める操作で PW_k を求める方法を示す。

$$PW'_k(m, b, t+Ts) = \begin{cases} 0 & m<0 \text{ 又は } b<0 \text{ のとき} \\ \sum_{j=0}^{NM_k} \sum_{i=0}^{b} & (PW_k(m+j, b-i, t) \cdot OM(j, m+j, t) \cdot OB(0, b-i, t) \cdot IW_k(i, t) + PW_k(m-1+j, b+1-i, t) \cdot OM(j, m-1+j, t) \cdot OB(1, b+1-i, t) \cdot IW_k(i, t)) \quad \text{その他} \end{cases} \quad (3.9)$$

PW'_k から PW_k を求める方法は次のとおりで図3．5（b）は、その操作を示している。

$$PW_K(m, b, t)$$

$$= \begin{cases} PW'_K(m, b, t) & m < MB,\ b < NP_K * PB \text{のとき} \\ \sum_{g1=MB}^{MB+1} PW'_K(g1, b, t) & m = MB,\ b < NP_K * PB \text{のとき} \\ \sum_{g2=NP_K \times PB}^{NP_K \times PB + NP_K} PW'_K(m, g2, t) & m < MB,\ b = NP_K * PB \text{のとき} \\ \sum_{g1=MB}^{MB+1} \sum_{g2=NP_K \times PB}^{NP_K \times PB + NP_K} PW'_K(g1, g2, t) & m = MB,\ b = NP_K * PB \text{のとき} \end{cases} \quad (3.10)$$

以上がPM_Kの状態方程式であるが、このようにPMを設定して方程式を立てることにより、若干の工夫、変形、組み合せで、種々のバス結合並列計算機の解析が可能となる。

3.2.3.2 状態方程式の解法

状態方程式の解法は、Tsごとに逐一状態確率行列を求め、平衡状態に達するまでその操作を繰り返していく方法が計算機処理に便利であり、システムの動特性も同時に求まる利点がある。図3.7がその流れ図である。

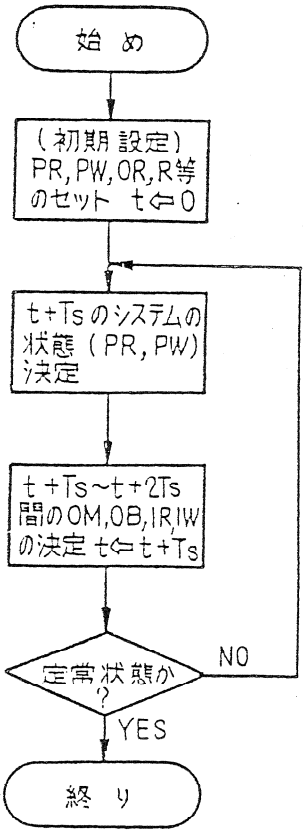


図 3.7 計算機解析の流れ

３．２．４　バス結合並列計算機の解析

本項ではバス結合並列計算機設計において重要な３点に関する解析結果を示す。なお以下の解析では、特にことわらない限り、ライトリクエスト用バッファとして、各プロセッサが３単位データ分、メモリ群が３＊ＮＰ単位データ分を持っているものとする。なお、この項での解析ではＰＭを区別する必要がなく、変数名としては３．２．２の変数名から添字を除いたものを用いる。

３．２．４．１　バッファサイズの検討

最初の解析はプロセッサとメモリに用意するバッファの大きさに関するものである。ここでは、バス結合並列計算機の設計でバッファサイズを決定する際の参考資料を示すと共に、この解析モデルの仮定の一つであるオーバフロー処理の妥当性の検証を示す。図３．８は、ＮＰ＝８のＰＭで、メモリとバス処理フェーズのあるデータ及びリクエスト数の分布をＲＭ，λ_r，λ_wをパラメータとして求めたものである。この結果、ライトリクエスト頻度がリードリクエスト頻度に比べてあまり多くないとき、各プロセッサは、ライトリクエスト用に数単位データ分のバッファを用意することで、オーバフローはほとんど起こらないこと、また単にバッファサイズを増やしてもそれに比例した性能向上は望めないことが明らかになった。図３．９は、図３．８の代表的曲線について、バッファサイズとオーバフローの関係を求めたもので、これから数単位データ分のバッファの導入で、オーバフローの確率は、10^{-6}程度に減少し解析への影響は無視できる。

３．２．４．２　共有メモリ数の評価

共有メモリの数を増やしデータを分散させるとメモリでの衝突は減少する。この方法は、メモリ処理時間が、プロセッサ速度に比べて大きいときなどに有効なものである。ここでは、メモリ台数が２以上で各プロセッサが各メモリに等確率でアクセス場合について性能評価を行った。図３．１０はプロセッサ数とプロセッサ稼働率の関係を種々のケースについて求めたものであり、図３．１１はそのときのリソース利用率と待ちの長さの関係を調べたものである。この結果、バス結合並列計算機の共有メモリ数としては、ＲＭ（19 ページ参照）程度に分けることが合理的であることがわかる。しかしリソースの利用率がかなり高い（７０%程度）場合でも、システムの性能に与える影響は、一般の待ち行列（無限長待ちを許す行列，図３．１１の破線）に比べ、非常に少ないことも明らかになった。

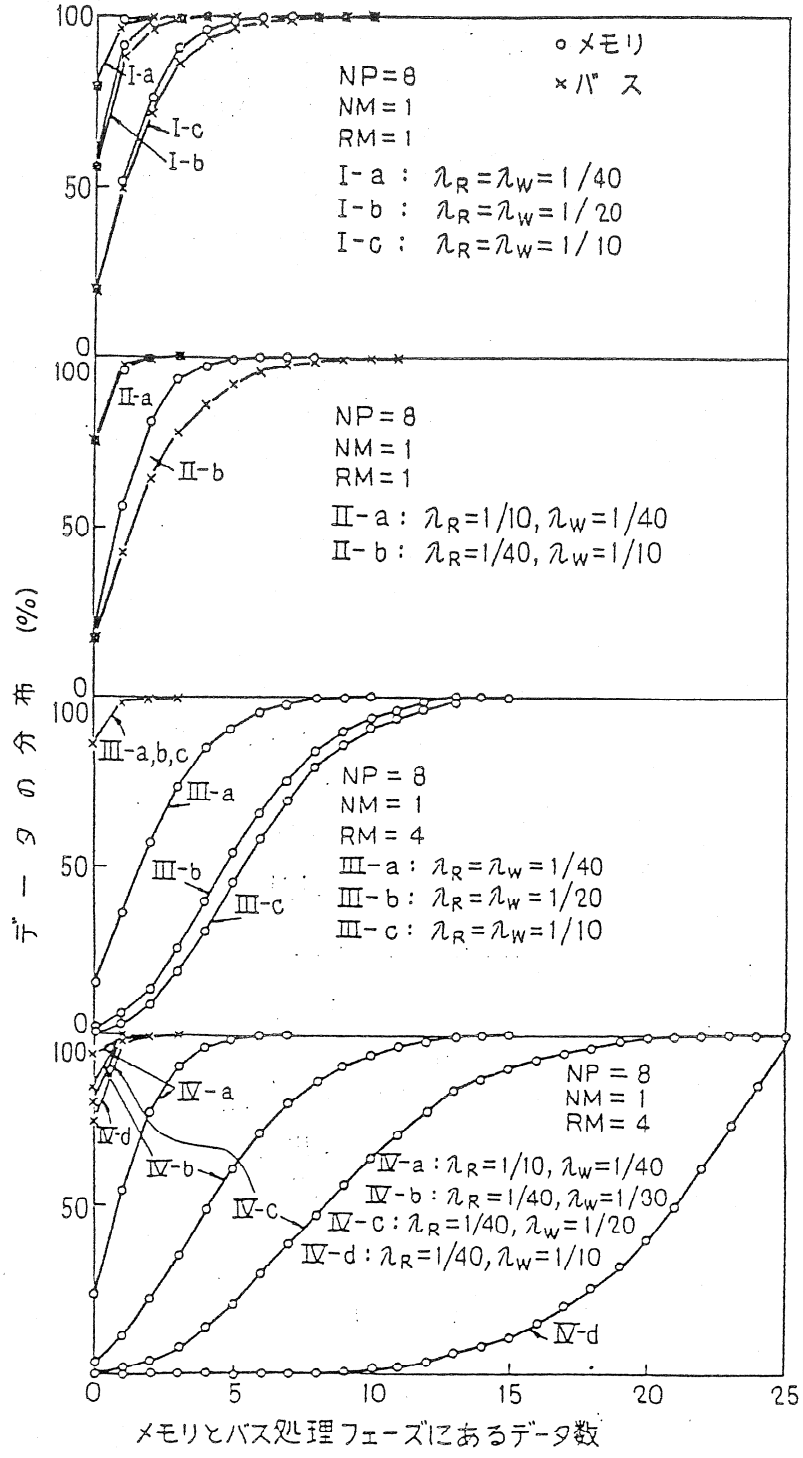


図 3．8　資源上のデータの分布

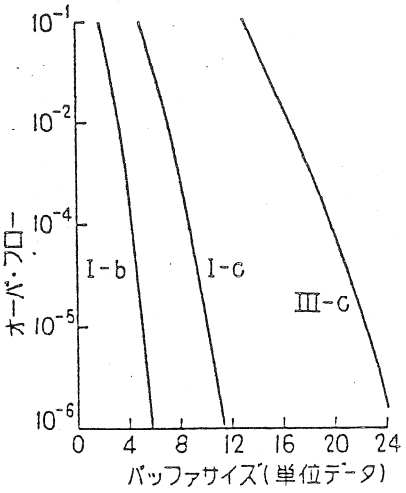


図 3．9　オーバフローの確率

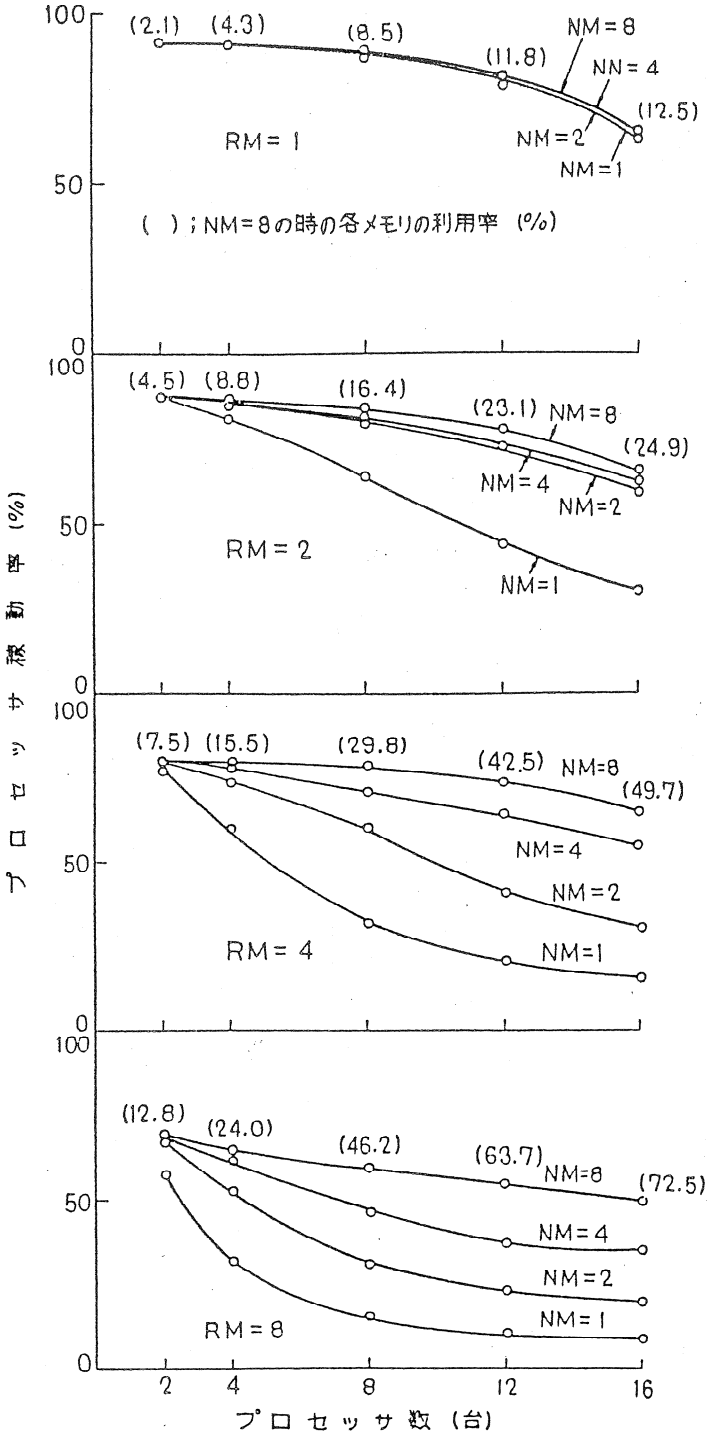


図3．10　共有メモリ数の評価

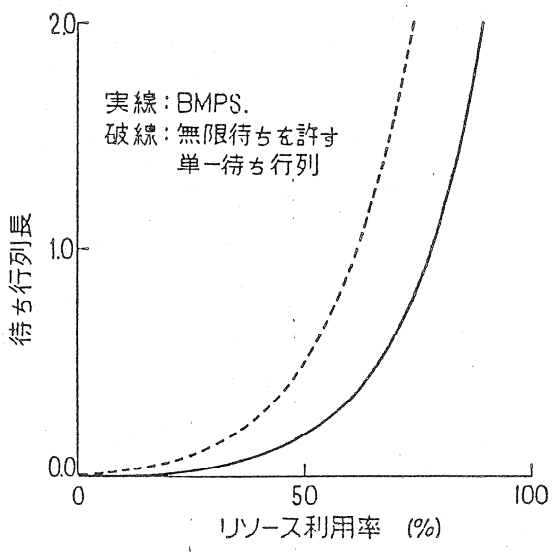


図 3．11 システムの特性

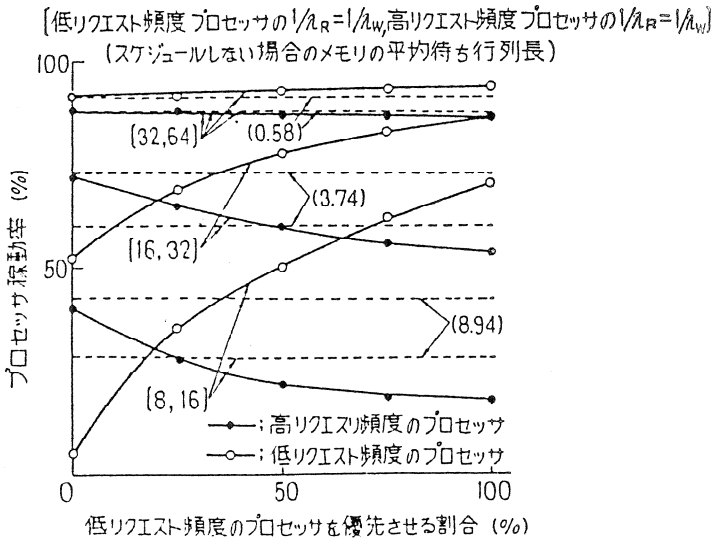


図 3．12　リソース・スケジューリングの効果

3．2．4．3　リソース・スケジューリング

リクエスト頻度の異なる複数のＰＭからなるシステムでは各ＰＭのリクエストに優先度を与えることにより性能を制御することができる。ここでは複数のプロセッサに共有されるリソースに、簡単なスケジュール機能を設け、プロセッサに優先度を付けて処理したときのシステムへの影響を調べてみた。ここでは、１台のメモリ（ＲＭ＝２）を共有するリクエスト頻度の異なる２群（各群４台）のプロセッサ群の一方に確率的優先度を与えたときのシステムの性能を調べた。ここで確率的優先度とは、一般に優先度といったとき、高優先度のものを全ての低優先度のものに優先させるのに対し、高優先度のものをある確率で優先させるというものである。図３．１２は、メモリとバスにスケジュール機構を持たせたときの効果を、λ_r，λ_wをパラメータとして求めたものであり、図中の破線は、スケジュールを行わないときのものである。この結果、待ちが長い時には、スケジュールが有効といえるが、３．２．４．２の結果と総合して考えると、リソースがリソースバウンドにならない程度に利用する限りは、あまり性能に影響を与えないことが分かった。

３．３　並列計算機シミュレーションシステム

３．３．１　並列計算機シミュレーションシステム（ＰＰＳＳ）の概要

並列計算機の設計を行う場合図３．１３のような設計フェーズがあり、試行錯誤を繰り返しながら、目的のシステムを設計していく。ＰＰＳＳは、種々の並列計算機のシミュレータをＰＭＳレベルで構成し、設計した並列計算機の性能測定と動作の確認を行うことを目的に設計したシミュレーション・システムである。図３．１４は、シミュレーションの流れを示している。並列計算機のシミュレータを作ろうとする人は、図３．１４の四つの設計フェーズに対応した四つの記述を行うと、それぞれに対応したルーチンで処理され、四つの分離されたサブシミュレータがＦＯＲＴＲＡＮのソースプログラムの形で生成される。このサブシミュレータ間のインタフェースは、均一化され、単純化されているため、ある設計フェーズの変更は、他の記述と独立に行うことができる。又サブシミュレータは、ＳＩＭＴＲＡＮのイベント管理部を利用しているが、ユーザは、これに関する知識は不要である。四つのサブシミュレータは、ＳＩＭＴＲＡＮのプログラムと共にコンパイルされ、リンクされて目的のシミュレータが完成し、シミュレーションが実施される。

ＰＰＳＳは二つのモードのシミュレーションを行うことができる。第一は、各プロセッサの処理内容には関知せず、システム全体の性能を求めるものである。第二は、各プロセッサの動きを命令レベルでシミュレートするもので、システムの性能の他並列処理の処理結果も求まるものである。なお、後者のモードでは、アプリケーションプログラムの記述次第でシミュレートする命令のレベルは、マイクロ命令からマクロ命令まで任意に設定できる。

性能に関する測定データは、各資源の利用率、待ち時間、使用頻度等であり、各記述中で測定対象を指定するため、任意の項目に対して測定ができる。又並列計算機では、各プロセッサの処理過程が固定の場合等に、ある資源がボトルネックとなり、システムが同期動作をする場合があるため、これに関するデータも求まるようになっている。

３．３．２　設計の要点

本項は、ＰＰＳＳの設計思想と主な設計思想を示す。

（１）ＰＭＳレベルでのシステム把握

多数のＬＳＩを使って並列計算機を構成する場合、資源の配置、結合は、重要な設計ポイントである。このような点を明確に評価するため、並列計算機をＰＭＳレベルでとらえるよ

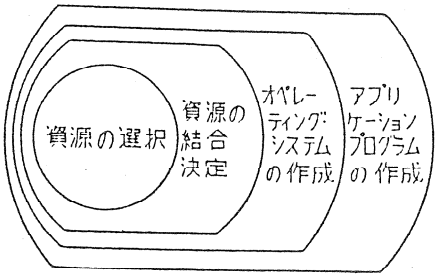


図 3．13　並列計算機の設計フェーズ

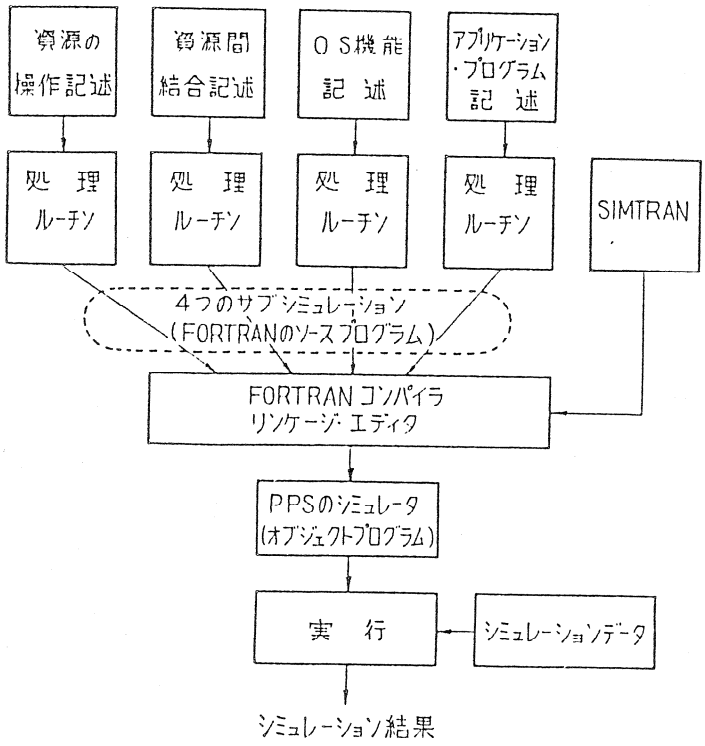


図 3．14　シミュレーションの流れ

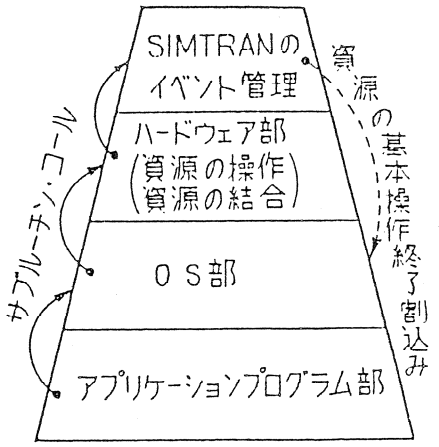


図 3．15　シミュレータの階層構造

うにした。

（２）設計フェーズによるシミュレータの分割と階層化

図３．１３に示した各フェーズの設計者が、自分の設計しているフェーズにのみ着目してシミュレータを構成できるようにするため、シミュレータは設計フェーズに対応した三つのレベル（資源操作部と資源結合部は同一レベル）から構成され、レベル間のインタフェースは原則として一つ上のレベルへのサブルーチンコールに限定する。図３．１５は、その様子を示したもので、最上位はＳＩＭＴＲＡＮのイベント管理部である。図３．１５の各レベルの概要は次のとおりである。

（ｉ）ハードウェア部：各資源の基本操作と、資源間結合状態を示すデータベースよりなり、このレベルからのみＳＩＭＴＲＡＮのイベントスケジュール表にアクセスできる。

(ii) オペレーティングシステム（ＯＳ）部：資源のスケジュール等を主な仕事とし、アプリケーションプログラム部中で使われるプリミティブを解釈し、ハードウェア部の基本操作に起動をかける。

(iii)アプリケーションプログラム部：並列計算機上で走るアプリケーションプログラムをＦＯＲＴＲＡＮ文とＯＳ部へのアクセスプリミティブで書いたもの。

（３）資源間結合の可変性

並列計算機のシミュレーションでは、使用する資源が決定した後で、それらの結合形態をいろいろ変えて性能評価を行う場合が多い。ＰＰＳＳでは、このような結合形態の変更を容易に行なえるように、図３．１５のハードウェア部を資源の操作を記述する部分と、資源間のインタフェースを記述する部分に分離している。

（４）ＯＳの機能をユーザが決定する

並列計算機のＯＳは、システム毎にシステム構成や応用を考慮して個別に設計されることが多い。そのためＰＰＳＳではＯＳの機能を予め固定することをせず、ユーザが自分の好むＯＳ機能を自由に組立てられる機能を提供している。これは、ＯＳの核に相当するものである。

（５）ＦＯＲＴＲＡＮベースのモジュール化

ハードウェア部とＯＳ部は、サブルーチン化された基本機能の集合であり、前節で述べたように１レベル上のサブルーチンをＦＯＲＴＲＡＮプログラム中から呼ぶ形で下位レベルの基本機能ができている。これは、構造化プログラミングの概念を実現するもので、シミュレータの構成が明確になると共に、シミュレータの部分的変更が全体に影響を与えないように

する上で重要である。

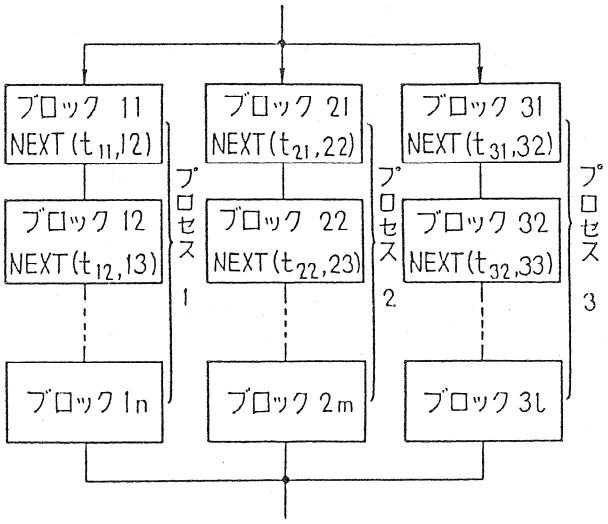
ＰＰＳＳの生成するシミュレータは、ＦＯＲＴＲＡＮのソースプログラムであり、システム構成に融通性を与え、プログラムのチェックも容易に行える。またＰＰＳＳ自体もＦＯＲＴＲＡＮで書かれているため、多くの計算機システムでシミュレーションを実行することができる。

（６）二つのタイプのシミュレーション

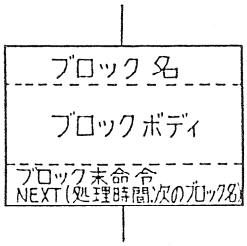
３．３．１で簡単に述べたように二つのタイプのシミュレーションが可能である。第一のタイプは、並列計算機上で走るプログラムを命令レベルでシミュレートするもので、実際のプロセッサで実行される処理がそのままシミュレートされるため、並列処理の演算結果も求まり、並列処理のタイミングに由来するプログラムエラーなどのチェックに利用できる。第二のタイプは各プロセッサの発生する他の資源へのアクセス頻度の統計データに基づいて、各資源の利用状況等を求めるものである。これは、各プロセッサの処理内容には関知せず、システムの大まかな性能を調べるときに便利である。

（７）アプリケーションプログラムのシミュレート法

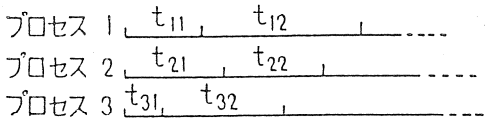
（６）で示した第一のタイプのシミュレーションの場合、命令にもマイクロ命令、機械命令、マクロ命令等、色々のレベルがあり、必要に応じたレベルでのシミュレーションが行える必要がある。ＰＰＳＳでは、ユーザがアプリケーションプログラム部を記述する際自由に設定できるようにしてある。ＰＰＳＳでは、並列計算機の処理をプロセスの集合ととらえ、各プロセスは割り込み不能なブロックの集合とし、シミュレーションは、ブロック単位に進める。図３．１６はその様子を示している。図の（ａ）は、３プロセスの例を示しており、（ｂ）は、ブロックの構成を示している。ブロックは、ブロック名、ブロックボディ、ブロック末命令よりなる。ブロックボディは、ブロックでの操作をＦＯＲＴＲＡＮ文で書くもので、マイクロ命令レベルのシミュレーションを行いたい人はマイクロ命令に相当する操作を、マクロ命令レベルのシミュレーションを行いたい人はマクロ命令に相当する操作をここで指定すればよい。ブロック末命令（ＮＥＸＴ）は、ブロックの処理時間と、次に実行すべきブロック名を指定する。図３．１６（ｃ）は、プロセッサ数がプロセス数以上ある場合のブロックの処理時間例と、計算機の処理順序を示している。



(a) 並列プログラムの構造



(b) ブロックの構成



$t_{11}, t_{21}, t_{31}, t_{32}, t_{12}, t_{22}$ ----

(c) ブロックの処理時間例と計算機での処理順序

図 3．16　並列処理のシミュレーション法

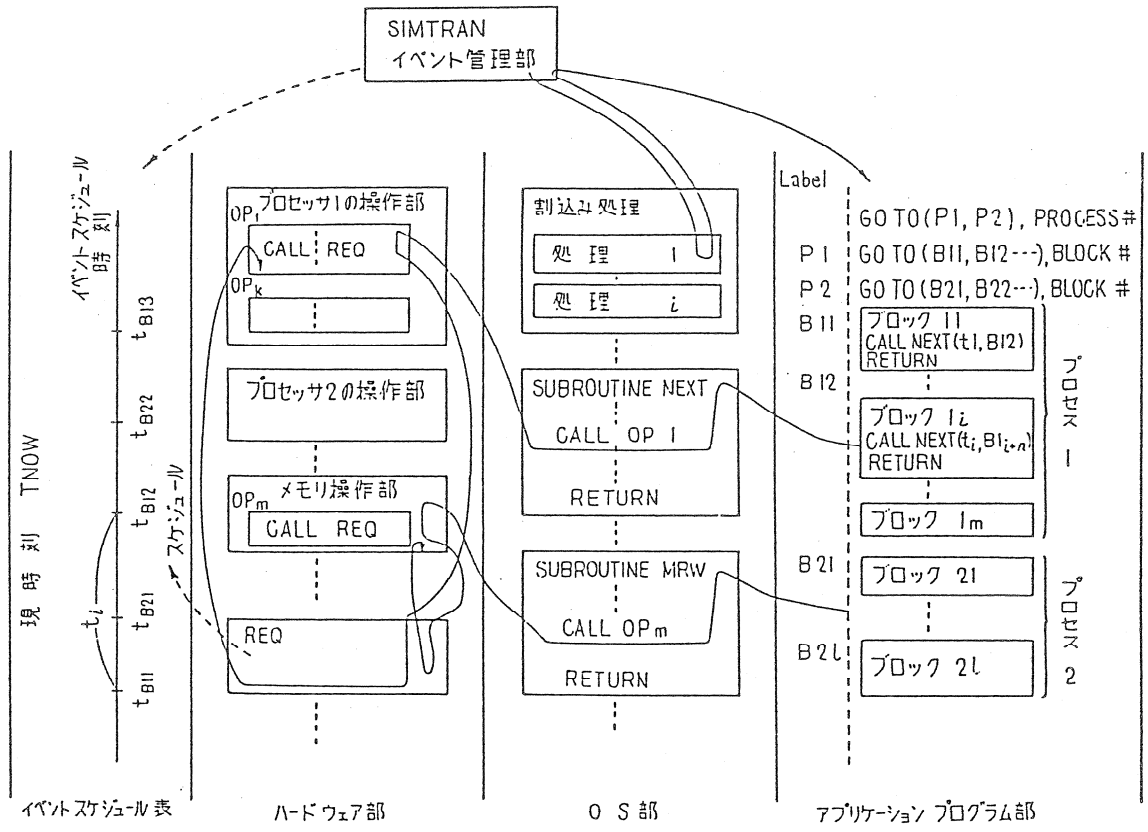
３．３．３　並列計算機のシミュレート法とシステム構成

本項は、前項の設計思想を実現する並列計算機のシミュレート法と、シミュレーションシステムの構成を述べる。

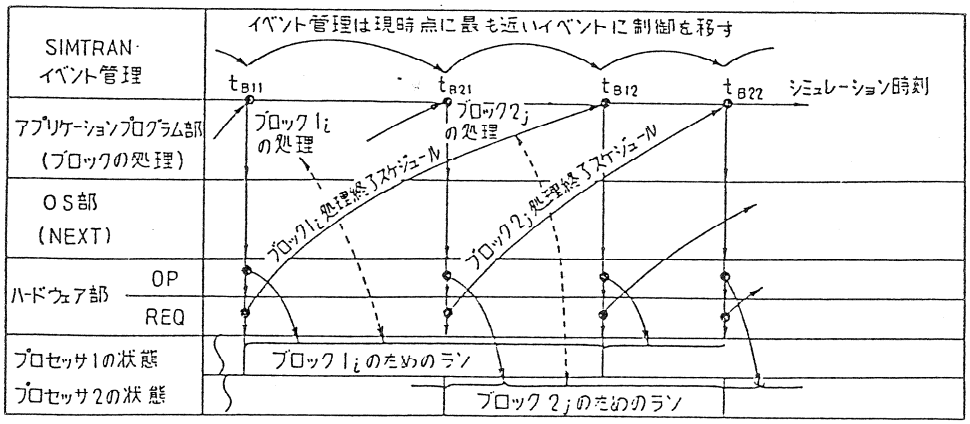
３．３．３．１　並列計算機のシミュレート法

ここでは、並列計算機の処理を１台の計算機でどのようにシミュレートするかを、図３．１７の例で説明する。これは、二つのプロセスを２台のプロセッサで実行する単純な場合を示している。ＳＩＭＴＲＡＮの基本機能はイベント管理であり、これはイベント管理部とイベントスケジュール表で実現される。イベントスケジュール表には、イベントの発生時刻とイベントの種類が登録され、イベント管理部は、現在のイベントの処理が終わると、現在のシミュレーション時刻に最も近い時刻に起こる次のイベント発生時刻にシミュレーション時刻を進め次のイベント処理に制御を移す。今、シミュレーション時刻が t_{B11} で、これはプロセッサ１がブロック１ｉの処理を開始するという時刻とすると、制御はアプリケーションプログラム部のブロック１ｉの処理に移る。アプリケーション部のブロック１ｉにはユーザプログラムのブロック１ｉの処理が書かれていて、それが実行される。ブロック１ｉの終わりにはＮＥＸＴ命令があり、ここに実際のマシンでのブロック１ｉの処理時間と次に実行すべきブロック名が書かれている。この例では処理時間が t_i で　t_{B11} ＋ t_i　が t_{B12} であるとしている。ブロック１ｉの処理が終わった時点では、プロセッサ１が t_{B11}～t_{B12} の間に行う処理は既に終わっているがシミュレーション時刻はまだ t_{B11} であるため、プロセッサ１が実際には t_i 間の処理を行っているようにシミュレートする必要がある。そこで、ブロック末命令（ＮＥＸＴ）からハードウェア部にアクセスしてプロセッサ１をラン状態に登録すると共にブロック１ｉの処理が t_{B12} に終了する旨をＲＥＱ命令でイベントスケジュール表に記入する。この後、制御はイベント管理部に移り、上記の処理を繰り返しシミュレーションが進められる。図３．１７（ａ）の実線は制御の流れを示し、（ｂ）は、それをモデル化して示している。

又、ブロックの処理が共有メモリへのアクセスなど、他の資源の状態が関係し、処理時間を予め決定できない場合には、ＮＥＸＴ命令を使わない。図のサブルーチンＭＲＷは、メモリアクセス用のＯＳ機能で、あるブロックからこれが呼ばれると、ここから、ハードウェア部のメモリ操作部が呼ばれる。メモリ操作部は、もしメモリが空いていれば、メモリを使用中にし、読み出し終了時刻をイベントスケジュール表に登録するし、メモリが使用中なら、



(a) Simulation process example



(b) Simulation process model.

図 3．17 シミュレーションプロセス

待ち行列にいれる。待ちに入ったリクエストは　メモリで現在処理中の仕事が終わった時点に、割り込み処理から再度ハードウェア部のメモリ操作部が呼ばれ、待ちから出される。以上がシミュレート法の概要である。

３．３．３．２　システム構成

ここでは、シミュレータを構成する四つの記述と、それらの処理部に関して述べる。

（１）ハードウェアに関する記述とその処理部

ハードウェアに関する記述は、資源の操作記述と、資源間結合記述からなる。図３．１８は操作記述の例で、資源名（ＲＮ）、他の資源からアクセス可能な状態信号（ＧＳ）、基本操作（ＯＰ１，・・・，ＯＰｎ）からなる。基本操作は、資源の動作をＦＯＲＴＲＡＮ文、システムの提供する基本命令、ユーザがハードウェア部で定義した命令により記述したものである。結合記述は操作記述内でアクセスする他の資源の状態信号（ＳＴ１，ＳＴ２，・・・，ＳＴｎと記述される）が他の資源のＧＳ部で示されるどの信号に相当するかを指定する。

ハードウェア記述に対する処理部は、記述からソースプログラムを生成するルーチン、システムの提供する基本命令、資源間結合を記憶するデータベースからなる。基本命令には、表３．１に示すような、待ち行列操作命令、イベントスケジュール表の操作命令、統計データを得る命令、乱数発生命令等がある。故に、ユーザはＳＩＭＴＲＡＮの知識を持っていなくてもＦＯＲＴＲＡＮと基本命令さえ知っていれば、この記述を行うことができるが、特に細かな制御にＳＩＭＴＲＡＮの命令を使いたい場合には、この記述中にそれを用いることができる。

（２）ＯＳ部の記述とその処理部

ＯＳ部は、ＯＳの様々な機能をユーザが定義する部分で、その記述はハードウェア部同様ＦＯＲＴＲＡＮプログラム中からハードウェア部で定義した資源の操作や、ＯＳ部でユーザが定義した機能、及び以下に述べるシステムの提供する機能を呼ぶことにより行われる。

この記述に対する処理部は、記述からＦＯＲＴＲＡＮのソースプログラムを生成するルーチンの他、ユーザのＯＳ機能定義を容易にするための次の３機能を用意している。

（ｉ）資源のスケジュール機能：並列計算機では資源のスケジュールが重要な機能である。ここでは、表３．１に示す機能（ＲＥＱ命令は除く）の他、プロセスをプロセッサに割り当てるのを容易にするため図３．１９に示すデータ構造やプロセッサ管理用待ち行列とそれに対する操作命令を提供する。

操　作　記　述	コ　メ　ン　ト
@RN	リソースネーム.
MEM 1.	メモリ #1.
@GS	グローバルステータス.
.	ナシ.
@OP 1	オペレーション 1.
(WN, NB)	パラメータ.
IF(MS, EQ. 0) GO TO 100	もしメモリが使用中でなければ 100 へ飛べ.
ST 1(ICP)=0	プロセッサを待ちにする.
¥S(ICP+1)	プロセッサの待ち時間の統計を取る.
.	
¥QI(2, 999, WN, NB, 0)	待ち行列に入れる.
¥E	終り.
100 MS=1	メモリを使用中にする.
¥S(1)	メモリの使用時間の統計を取る.
TIME=WN*ACTI	メモリの読出し時間を計算.
¥REQ(TIME, 1, NB, 0, 0)	メモリの読出し終了時刻をスケジュール.
¥E	終り.
@OP 2	オペレーション 2.
基本操作記述 2	
@END	終り.

図 3．18　資源操作の記述

表 3．1　ハードウェア記述命令

- REQ($t, i, a_1, a_2, \cdots, a_6$): 資源への操作起動命令　SIMTRAN のイベントスケジュール表に，操作終了時刻を登録.
  - t; 操作の所要時間
  - i; 操作終了時の処理ルーチン番号
  - $a_1, a_2, , \cdots a_6$; ユーザの指定する属性
- QI($n, c, a_1, a_2, , \cdots a_6$); 待ち行列 n へのデータの加入
  - c; データの挿入位置指定
  - $a_1, a_2, \cdots, a_6$; ユーザの指定する属性

  但し，プロセス名，現ブロック名，現時刻は自動的に属性として登録される.
- QO($n, c, a_1, a_2, \cdots, a_6$); 待ち行列 n からデータを取出す. 引数は QI と同じ.
- F(n, i, d, c); 待ち行列 n の i 番目の属性値が d である待ち行列中のエレメント番号を c に求める.
- S(m); この一つ前の命令の変数に関する統計データを集める.
- E; 操作終了記号.
- 各種乱数発生命令.

注)　属性 a_1〜a_6 は，6 個以下任意の個数にセット可.

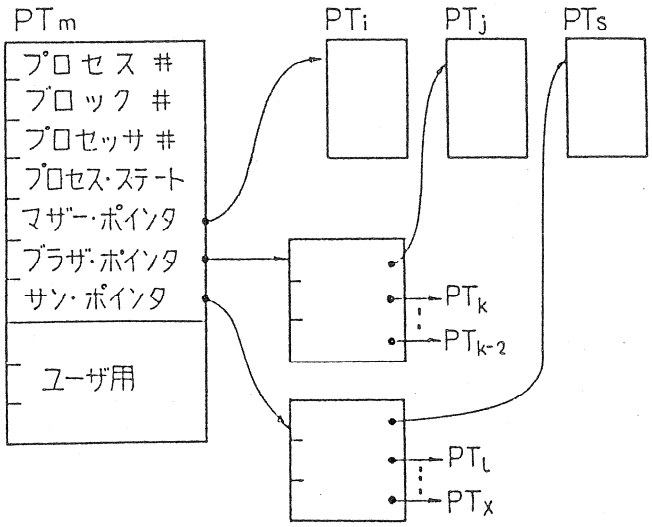


図 3．19　プロセス・テーブル

表 3．2　OS機能

1, プロセスの生成・消去命令.
2. プロセス間の同期命令.
3. 資源間の通信命令.
4. ブロック末命令.
5. 割込み処理.

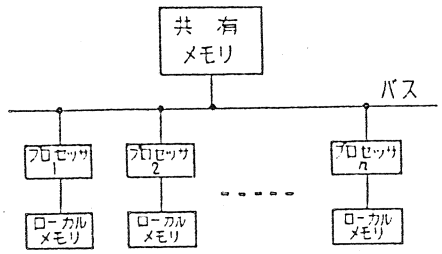


図 3．20　ハードウェア構成

(ii) データ転送管理: データ転送には様々な形態があり、単純なものはハードウェアの資源記述部で管理できる。ここでのデータ転送管理とは、ＯＳの介入が必要なような、複雑なデータ転送をサポートするもので、資源の物理的結合関係を示すテーブル、転送路を決定するルーチン等からなっている。

(iii)割り込み処理: この部分は、ハードウェア操作記述中のＲＥＱ命令により起動をかけられた資源の操作が終了したとき次に何をするかを指定する。即ち、ＲＥＱ命令の第2引数(i)がこの割り込みルーチンの番号を示す。

以上がＯＳ記述サポート機能であるが、この他に予め用意されたＯＳ機能として表３．２に示すようなプリミティブを用意している。

（3）アプリケーションプログラムとその処理部

アプリケーションプログラムは、一つのサブルーチンとして構成される。記述例は次項で示す。これに対する処理部は、ユーザの指定をもとに、ブロックを決定すること、ブロック名とＦＯＲＴＲＡＮのラベルとの対応付けを行う。

３．３．４　シミュレーション例

高速フーリエ変換を、図３．２０のシステム構成で行う場合のアプリケーションプログラムの主要部を図３．２１に示す。プログラムの処理の流れは、ラベルＢ２以下のＦＯＲＫ命令で並列プロセスを起動し、ラベルＢ１１のＪＯＩＮ命令で並列プロセスを終了させるという操作を繰り返す。ここで共有メモリには、被変換データとその中間結果のみを格納し、回転因子その他は各プロセッサがローカルメモリに格納していると仮定する。被変換データは１０２４点の単精度で、プロセッサはサイクルタイム２００ナノ秒のマイクロプロセッサＰＵＬＣＥ（４．３．１参照）を想定し、メモリのアクセス時間は１マイクロ秒でシミュレーションを行った。図３．２２はプロセッサ台数を変えたときのシステムの性能を求めたものである。測定結果は、プロセッサ数８台程度まで効率良く動作することを示している。又、メモリを７０％程度の高利用率で使用しても性能にさほど影響がないことが分かる。これは、３．２．４の解析結果とも符合する。

```
B1   〈初 期 設 定〉
B2   FORK(2,B3)
       ⋮
     FORK(16,B3)
B3   KLEFT(PN)=IR1(L,PN)+IR2(L,PN)-1
     KRIGHT(PN)=KLEFT(PN)+DLR
     JLEFT(PN)=JLEFT(PN)+1
     JRIGHT(PN)=JLEFT(PN)+N2
     K(PN)=(IR2(L,PN)-1)*K1+1
     WK(PN)=W(K(PN))
     ICTR(PN)=ICTR(PN)+1
     NEXT(170.0,B4)
B4   W1(PN)=A(JLEFT(PN))
     MRW(4,B5)
B5   W3(PN)=A(JRIGHT(PN))
     MRW(4,B6)
B6   W2(PN)=W3(PN)*WK(PN)
     W3(PN)=W1(PN)+W2(PN)
     NEXT(440.0,B7)
B7   B(KLEFT(PN))=W3(PN)
     MRW(4,B8)
B8   W3(PN)=W1(PN)-W2(PN)
     NEXT(162.0,B9)
B9   B(KRIGHT(PN))=W3(PN)
     MRW(4,B10)
B10  IF(ICTR(PN).EQ.MAX) GO TO 10
     IR2(L,PN)=IR2(L,PN)+1
     IF(IR2(L,PN).GT.DLR) GO TO 20
     NEXT(27.5,B3)
  10 NEXT(12.5,B11)
  20 IR1(L,PN)=IR1(L,PN)+DLR2
     IR2(L,PN)=1
     NEXT(62.5,B3)
B11  JOIN(PN)
B12  DO 100 I=1,N
 100 A(I)=B(I)
     MRW(9192,B13)
B13  IF(L.EQ.10) GO TO 200
     〈新たなFORKのための設定〉
     NEXT(15.0,B2)
 200 RETURN
```

〈注 Bn;ユーザの設定するブロック〉

図 3．21　FFTプログラム

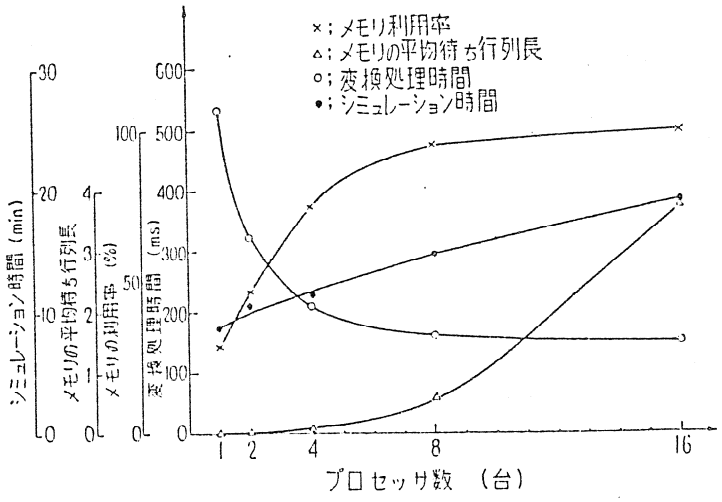


図 3．22　シミュレーション結果

3．4　まとめ

本章では二つの性能評価システムとそれを用いた二、三の評価結果を示した。バス結合並列計算機の解析法は、バスという並列計算機の一つの形態に対して示した解析法であるが、バス方式は、並列計算機を構成する基本単位であるためバス方式で他の方式をシミュレートすることが可能な場合が多い。ここで示した解析法の利点は、

（1）システム内部の負荷状態が求まる。

（2）バスバウンド、メモリバウンドいずれのシステムにも適用可能。

（3）システムの動特性が求まる。

等である。

この解析法は、バッファサイズが解析時間に大きな影響を与える。バッファサイズを、解析中にオーバフローをチェックして無駄なく決めるアルゴリズムを導入することにより解析時間の大幅な改良が可能であろう。

3．3節のシミュレーションシステムは今後ますます盛んになるであろう並列計算機の構築に重要な道具となろう。一般に計算機をソフトウェアでシミュレートすると膨大な時間がかかることが知られている。本シミュレーションシステムを用いれば、目的に即した並列計算機のシミュレータが容易に構成できるため無駄のない効率良いシミュレーションが可能となる。本システムはこの点が高く評価され、東芝で実用化されて[Tak76]多くの並列計算機開発に利用された。又、ここで示した階層化とモジュール化を用いたシミュレータ構成法は、並列計算機用のみならずシミュレータ構成法の有力なアプローチと思われる。

3．2．4ではバス結合並列計算機に関する共有メモリ数や、リソーススケジューリング等に関する幾つかの評価を行った。これらの結果は並列計算機設計者に有用な資料となろう。なお、この結果については、3．3節でもシミュレーションで確認している。

第４章　高級言語計算機とマイクロプログラム

４．１　まえがき

第２章で示したように高級言語計算機を構築する手段としてマイクロプログラムを用いる方法は有力な方法と認められている。マイクロプログラム方式による高級言語計算機の実現には二つの開発要素がある。第一はマイクロプログラマブルプロセッサの開発で第二はマイクロプログラムの開発である。筆者は工業技術院大型プロジェクトパターン情報処理システムにおいてマイクロプログラマブルプロセッサＰＵＬＣＥ[Iiz76a][Iiz79b]とそれを基本処理要素とする並列計算機ＡＣＥシステム[Iiz75][Iiz76a][Iiz79a]の開発に参加してきた。本章では、ＡＣＥシステム上へのマイクロプログラム用いた並列Ｐａｓｃａｌマシン[Bri75][Bri77]の実現を通して、マイクロプログラムによる並列プログラミング言語用高級言語計算機の実現法を示すと共にその評価を示す[Fur80]。マイクロプログラムを用いて高級言語計算機を実現する方法としては、高級言語の中間言語を設定し、そのインタプリタをマイクロプログラムで実現するという方式が定着してきているが、並列プログラミング言語用高級言語計算機では並列制御部分があるため従来の方法ではマイクロプログラムのサイズが膨大になる。そこでここでは新たに開発した中間言語を用いてこれを解決する方法を示すと共にその評価を示す。又ここではプロセスの同期サポートのために新たに開発した専用ハードウェア[Fur77b]とそれを用いたプロセススケジュール機構についても述べる。

マイクロプログラムを用いた高級言語計算機は幾つも作られているが、その結果に関する評価報告は少なく、報告されているものでもその性能測定程度までである。今後よりよいマイクロプログラマブルプロセッサを開発するためには、マイクロプログラムと高級言語計算機の関係を明らかにし、アーキテクチャにフィードバックしていくことが重要である。幸いＰＵＬＣＥは多くの高級言語計算機の実現に使用されてきた。４．３節では高級言語計算機実現に用いられたマイクロプログラムを解析し、マイクロプログラムと高級言語計算機の関係やマイクロ命令の使用特性を明らかにすると共にＰＵＬＣＥに対する設計者の立場からの考察を試みる[Fur82]。

４．２　並列計算機上の並列Ｐａｓｃａｌマシン

４．２．１　ＡＣＥシステムと並列Ｐａｓｃａｌマシン

ここでは、ＡＣＥシステムと並列Ｐａｓｃａｌマシンの概要を述べる。

４．２．１．１　ＡＣＥシステム

ＡＣＥシステムは、応用適応性を持つモジュール型並列計算機の試みである。システム構成は、図４．１に示すように、３台のプロセッサモジュール，メモリモジュール、同期モジュール、Ｉ/Ｏプロセッサが２本のバスを介して結合されている。プロセッサモジュールは、マイクロプログラマブルプロセッサ（ＰＵＬＣＥアーキテクチャ）やマイクロプログラムメモリ等から構成され高級言語計算機を実現するホストプロセッサとなる。同期モジュールは並列制御機構をサポートするハードウェアで、Ｐ・Ｖ操作が容易に実現できる。モジュール間通信は、汎用性の大きなブロードカスト方式を採用している。

４．２．１．２　並列Ｐａｓｃａｌマシン

Ｂｒｉｎｃｈ　Ｈａｎｓｅｎは、並列ＰａｓｃａｌでＳＯＬＯと呼ばれるＯＳとその下で走るコンパイラやエディタを作成した。即ちＳＯＬＯの下では、一人のユーザが並列Ｐａｓｃａｌ やＰａｓｃａｌプログラムをコンパイルしたり編集したり実行したりできる。Ｂｒｉｎｃｈ　Ｈａｎｓｅｎはこの並列Ｐａｓｃａｌシステムにポータビリティを与えるため並列Ｐａｓｃａｌ用の仮想計算機（並列Ｐａｓｃａｌマシン）を設定した。ＳＯＬＯ等並列Ｐａｓｃａｌで書かれたプログラムは、コンパイルされると並列Ｐａｓｃａｌマシンの命令によるプログラムになる。Ｂｒｉｎｃｈ　Ｈａｎｓｅｎの提供するＳＯＬＯ、コンパイラ、エディタ等もこのコンパイルされた結果、即ち仮想計算機の命令に落とされたものである。これにより並列Ｐａｓｃａｌシステムを使いたい人はこの仮想計算機のシミュレータを作ればよい。なお、Ｂｒｉｎｃｈ　Ｈａｎｓｅｎはこのシミュレータを ＰＤＰ１１／４５で実現したものを提供しており利用できる。並列Ｐａｓｃａｌマシンのシミュレータは、並列Ｐａｓｃａｌマシンの命令のインタプリタと並列プロセスを１台の計算機に写像するカーネルと呼ばれる部分からできている。

筆者はこのエミュレータをＡＣＥシステム上に実現し、ＳＯＬＯシステムを走らせた。ＡＣＥシステムのプロセッサはマイクロプログラム計算機で機械語を持たない。マイクロプロ

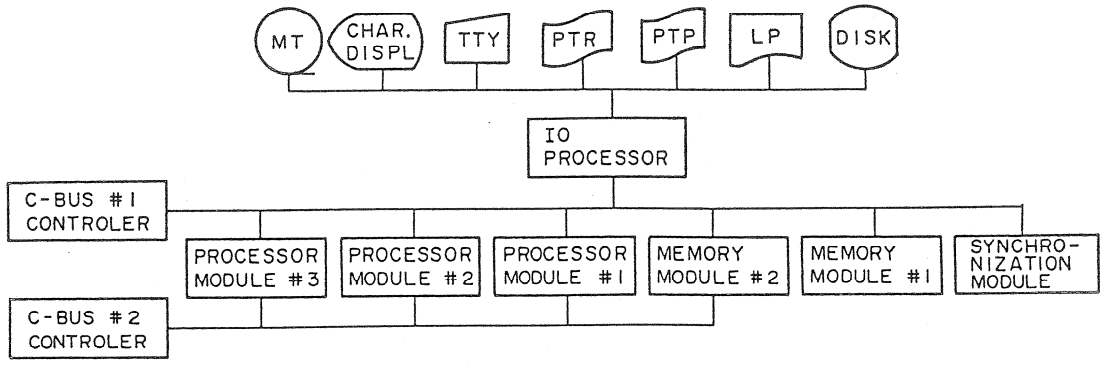
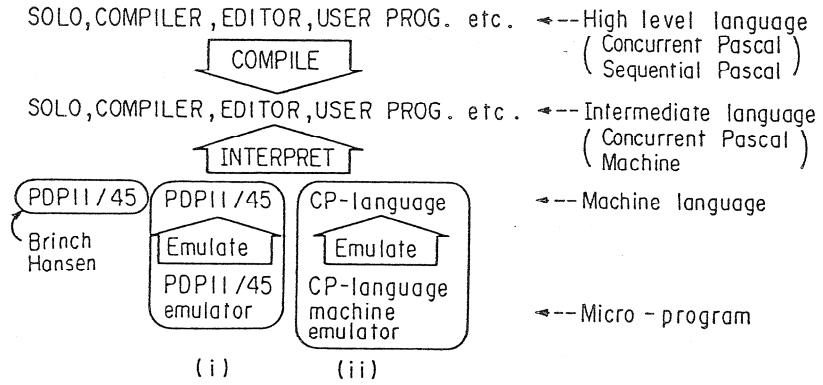
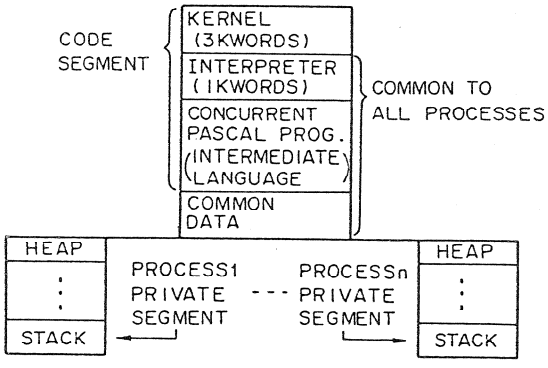


図 4．1　ACEシステム



(a)　並列Ｐａｓｃａｌマシン実行プロセス



(b)　並列Ｐａｓｃａｌマシンメモリ割り当て

図 4．2　並列Ｐａｓｃａｌマシン

グラムでカーネル、インタプリタを実現する方法は、いくつか考えられるが、ここでは以下の方法で実現し評価を行った（図４．２（ａ））。

（ｉ）ＰＤＰ11/45のエミュレータを作り、カーネル、インタプリタはＢｒｉｎｃｈ　Ｈａｎｓｅｎのものを使う。

（ｉｉ）並列Ｐａｓｃａｌマシン向きの新言語（ＣＰＬ）を設定しそのエミュレータを作り、ＣＰＬ言語の命令でカーネル、インタプリタを書く。

(iii)　インタプリタは (ii)の方法で実現しカーネルを並列計算機用に新たに制作する。

以下４．２．２では、（ｉ）、(ii)のマイクロプログラムによるカーネルとインタプリタの実現と評価を、４．２．３では並列計算機用カーネルの設計方針を、４．２．４では並列化における性能評価を中心に述べる。

図４．２（ｂ）はＰＤＰ１１/45用の並列Ｐａｓｃａｌマシンのメモリアロケーションである。ユーザの書いた並列Ｐａｓｃａｌプログラムはコンパイルされ図中のＣＰ　ＰＲＯＧＲＡＭという所に入る。これは各プロセス用ルーチンとそれらが共通に使うモニタルーチンからなる。各プロセスはスタックを持ち、これらは各プロセスにローカルなアドレス空間を作る。

４．２．２　マイクロプログラムによるインタプリタの実現

並列Ｐａｓｃａｌマシンの実現には、カーネルとインタプリタを作成すればよいことは前項で述べた。本項ではマイクロプログラムによる並列Ｐａｓｃａｌマシンの実現とその評価を示す。

４．２．２．１　並列Ｐａｓｃａｌマシンの実現法

図４．３は、並列Ｐａｓｃａｌプログラムが解釈実行される過程を示している。コンパイラが生成する並列Ｐａｓｃａｌマシンの命令はインタプリタにより解釈実行される。そしてモニタ等並列プロセスの制御が必要な場合にカーネルが呼ばれる。マイクロプログラムプロセッサで並列Ｐａｓｃａｌマシン命令を解釈する方法には次のようなものが考えられる。

（１）ＰＤＰ11/45のエミュレータを作る。（図４．３（ｂ））：Ｂｒｉｎｃｈ　Ｈａｎｓｅｎのカーネル、インタプリタをそのまま利用できるが効率が悪い。

（２）カーネル、インタプリタを全てマイクロプログラムで書く（図４．３（ｄ））：効率

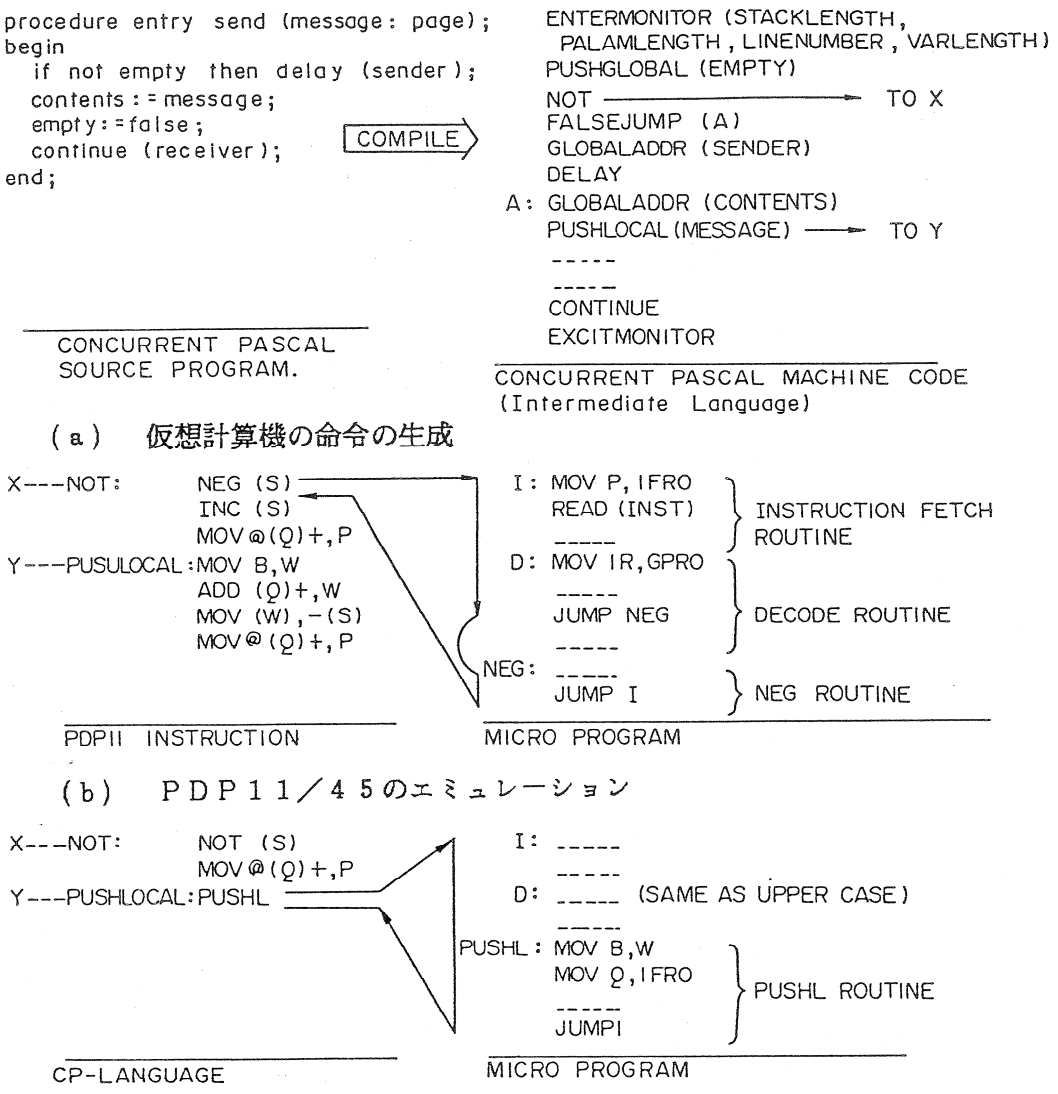


図 4．3 並列Pascalプログラムの実行プロセス

は良いがマイクロプログラム量が膨大になる。

（3）並列Ｐａｓｃａｌマシン向き言語を作成（図４．３（ｃ））：（1）と（2）の中間的なものでカーネルとインタプリタを効率良く実現する言語を作りそのエミュレータを作る。この方法は、（1）より効率が良く、（2）よりマイクロプログラム量が少ない。

並列Ｐａｓｃａｌマシンの実現法は他にも考えられるが、ここでは最も有効と考えられる（1）と（3）の方法で実現を行い、それらを評価した。

４．２．２．２　ＰＤＰ11/45　のエミュレータ

ＰＵＬＣＥとＰＤＰ11/45の語長は共に１６ビットで等しい。またＰＵＬＣＥは３２語のレジスタファイルを持っているためＰＤＰ11/45のレジスタは全てＰＵＬＣＥ内に割り当てられる。ＰＤＰ11/45の仮想アドレス方式は、ＡＣＥプロセッサモジュールのものと異なるため変換処理が必要である。

４．２．２．３　並列Ｐａｓｃａｌマシン向き言語（ＣＰＬ）

表４．１がＣＰＬの命令形式で大きく二つの型に分れる。第一の型（Ｎｏ．０～１０）は従来の機械語に相当するものであるが、ＰＤＰ１１／４５に比べると単純である。これはカーネルとインタプリタの記述にＰＤＰ１１／４５のような論理の深い命令が必要ないことと、第二の型が複雑な機能を吸収したためである。第一の命令形式はＰＵＬＣＥでエミュレートし易いように配慮した。第二の型（Ｎｏ．１１～１３）は頻繁に使われる並列Ｐａｓｃａｌマシン命令をファームウェア化したもので、第一の型に比べるとレベルが高くマイクロプログラムの能力を有効に利用するものである。表４．２はＳＯＬＯで使われる並列Ｐａｓｃａｌマシン命令の静的使用頻度で上位１０命令の頻度の総和は約７０％に達する。この特性は動的使用頻度でもあまり変わらないためＮｏ．１１ではこれら１０種の命令とインタプリタ内でよく使われるルーチンをファームウェア化した。Ｎｏ．１２はカーネルでよく使われる待ち行列管理ルーチン等をファームウェア化した。Ｎｏ．１３は浮動小数点演算命令である。

４．２．２．４　性能評価

表４．３は４．２．２．１で示した（1）と（3）の場合のカーネルとインタプリタのメモリ量とエミュレータのマイクロプログラム量を示している。カーネルとインタプリタのサ

表 4．1　　CPL　の命令形式

No.	BIT NUMBER 0 1 2 3 4 5 6 7 8 9 A B C D E F	COMMENTS
0	0 0 0 SC S. . . D. . . OP1	S*op D⟶D
1	0 0 0 DC S. . . D. . . OP1	S op D*⟶D*
2	0 1 0 0 ♯ DC D. . . Op1	CONST op D*⟶D*
3	0 1 0 1 ♯ ADDRESS OP1	CONST op (ADDR)⟶(ADDR)
4	0 1 1 0 D. . . ADDRESS.	LAOD
5	0 1 1 1 S. . . ADDRESS.	STORE
6	1 0 0 0 CONT D. . . OP2	SHIFT
7	1 0 0 1 CONT. OP1	STACK OPERATION
8	1 0 1 0 ADDRESS.	JUMP
9	1 0 1 1 OFFSET COND.	CONDITIONAL JUMP/JUMP SUBR.
10	1 1 0 0 MISCELLANE.	
11	1 1 0 1 INTERPRETER	FOR FREQUENTRY USED CPM INST.
12	1 1 1 0 KERNEL	QUEUE MANIPULATION ETC.
13	1 1 1 1 FLOATING	FLOATING INSTRUCTION

S. . . : Source register
D. . . : Destination register
*: operand mode
SC : Source, DC : Destination } operand mode
0 : Rn　1: (Rn)　2: (Rn)+　3: X (Rn).
OP1 : OPERATION 1
ADD, SUB, ADC, INC, DEC, OR, AND, XOR.
CONT.: Detail control
COND.: Condition
INTERPRETER: LOCALA, GLOBAL, PUSHLO, FIELD, PUSHCO, CALL, COPYWO, FALSEJUMP, EXCIT, STACKLIM, etc.
♯ : Whether the result is store or not

表 4．2　並列Ｐａｓｃａｌマシン命令の静的使用特性

PUSH LOCAL	13.9
GLOBAL	12.5
PUSH CONSTANT	8.4
FIELD	7.6
CALL	6.5
COPY WORD	6.0
PUSH GLOBAL	5.9
LOCAL	3.8
FALSE JUMP	3.0
ENTER PROCEDURE	2.2
PUSH LABEL	2.2
OTHERS	28.0

表 4．3　プログラムサイズ

(i) Kernel and Interpreter size (16 bits/word)

	PDP 11/45	CP-Language
Kernel	1180	1081
Interpreter	1084	850

(ii) Emulator size (32 bits/word)
PDP 11/45……1456 words
CP-Language……981 words

表 4．4　並列Ｐａｓｃａｌマシン命令の実行ステップ数

CPM Instruction	PDP 11 Emulator		CP-Language		Direct Exec.		PDP 11	
	μ Steps	MMAC*	μ Steps	MMAC	μ Steps	MMAC	Steps	MMAC
GLOBAL, LOCAL	321	10	71	5	62	4	3	9
FIELD	218	7	80	6	71	5	2	7
PUSH CONST.	223	7	69	5	60	4	2	6
COPY WORD	245	8	79	5	70	4	2	7
PUSH GLOBAL PUSH LOCAL	402	10	75	5	64	4	4	9
CALL	482	10	71	4	62	3	5	9
FALSE JUMP (True Case)	439	13	68	4	59	3	5	10

* MMAC: Main Memory Access Count.

表 4．5　実験プログラムの実行時間

Program Name	PDP 11 Emulator	CP-Language
1. HANOI	82 ms	36 ms
2. PERMUTATION	38 ms	21 ms
3. A Concurrent Pascal Program	24.51 s	16.04 s

ィズはＣＰＬの方が少ないのは当然である。マイクロプログラムのサイズはＣＰＬの方が並列Ｐａｓｃａｌマシンのファームウェア化を行っているにもかかわらずＰＤＰ１１エミュレータより少ない。これはＰＤＰ11/45の命令の論理が深く、命令デコードに多くのステップ数を要することと、並列Ｐａｓｃａｌマシン命令の多くが比較的少ないマイクロプログラム量で実現できることによる。ＰＤＰ11/45のオペランド部のデコード操作に３３６ステップ要し、表４．２の並列Ｐａｓｃａｌマシンのファームウェア化は１６８ステップで済む。表４．４は並列Ｐａｓｃａｌマシンの主な命令について実行マイクロステップ数とメモリアクセス数を４．２．２．１の三つの方法について求めたものである。同表中には、同命令をＰＤＰ11/45で実現した時のステップ数（機械語）も参考のために付け加えた。ＣＰＬがＰＤＰ11/45に比べてステップ数が多い理由は、機械命令とマイクロ命令の違いはあるが、仮想アドレスによるところがおおい。ＣＰＬではインプリメントの都合上ＰＤＰ11/45の仮想アドレス方式を踏襲したため１回のメモリアクセスで約５マイクロステップを必要とする。また並列Ｐａｓｃａｌマシン命令がＰＤＰ11/45の数ステップで書ける場合が多いこともファームウェア化による命令フェッチ数減少の利点を十分生せない原因である。表中の直接実行とは、４．２．２．１の（２）に相当し、ＣＰＬの値から命令フェッチを除いたものである。

表４．５はマイクロプログラムの効果を示すために行った実験結果である。ここでは、Ｐａｓｃａｌで書いた２つのプログラムと並列Ｐａｓｃａｌで書いた１つのプログラムを４．２．２に示した２種類のインタプリタとカーネルで実行した時の処理時間を示している。なお、ＡＣＥシステムのプロセッサプロセッサモジュールはマイクロプログラム用キャッシュメモリを持ち、マイクロ命令はキャッシュメモリからフェッチされる。実システムでの測定結果はマイクロプログラムキャッシュのヒット率が処理時間に影響するため厳密な比較結果とはいえないが、表４．４の値とほぼ等しいことは分かる。

４．２．３　並列Ｐａｓｃａｌマシンの並列実行

本項では並列Ｐａｓｃａｌマシンを並列計算機（ＡＣＥ）で実行する機構を示す。図４．４はＡＣＥシステム上の並列Ｐａｓｃａｌマシンの機能割り当てである。並列Ｐａｓｃａｌの並列プロセスは、２台のプロセッサに割り当てられて実行される。並列Ｐａｓｃａｌプログラム、カーネル、インタプリタは全て共有メモリに割り当てられる。

以下、プロセッサのスケジューリング、メモリの割り当て、モニタルーチンの相互排除について述べる。

４．２．３．１　プロセッサのスケジューリング

並列計算機ではプロセス数がプロセッサ数より多い場合には、プロセス切り換えに伴って、プロセッサのスケジュールが必要になる。ここではまずプロセス切り換えを引き起こす並列Ｐａｓｃａｌマシン命令と、その概要を示す。

（１）ＥＮＴＥＲＭＯＮＩＴＯＲ：モニタプロシージャの相互排除のための命令。この命令ではモニタゲートを調べ、“ｏｐｅｎ”ならモニタプロシージャへのアクセスを許し、“ｃｌｏｓｅ”ならモニタの待ちに入れる。

（２）ＥＸＩＴＭＯＮＩＴＯＲ：モニタプロシージャの出口で出される命令。モニタプロシージャへのアクセスを待っているプロセスを“レディ”状態にする。

（３）ＤＥＬＡＹ：並列ＰａｓｃａｌのＤＥＬＡＹ命令に対応し、走っているプロセスを待ち（ディレイ状態）にしてモニタプロシージャを出る。

（４）ＣＯＮＴＩＮＵＥ：ＤＥＬＡＹで待ちになったプロセスを再び走らせる。

（５）ＩＯ：入出力起動命令。プロセスは待ちになり、ＩＯ割り込みで復帰する。

ＡＣＥシステムでのスケジュール法は次のとおりである。

プロセスとプロセッサの対応付けは、予めユーザの指定により固定的に割り当てることもできるし、スケジューラに管理を委ねることもできる。スケジューラはプロセッサの状態、レディキュー、ウエイトキューを管理してプロセッサをスケジュールする。ＩＯやＤＥＬＡＹ命令により待ちになったプロセスは、もしレディキューにプロセッサを待っているプロセスがあればウェイトキューに移されるし、無ければそのまま待つ。この待ち状態でもウェイトキューには移されずにプロセッサを保持したまま待つ状態をオンプロセッサと呼び、他にプロセッサを要求するプロセスが発生した時はいつでもウェイトキューに移される。ＩＯの終了やＣＯＮＴＩＮＵＥ命令でプロセスが再度実行可能になると、そのプロセスがウェイト

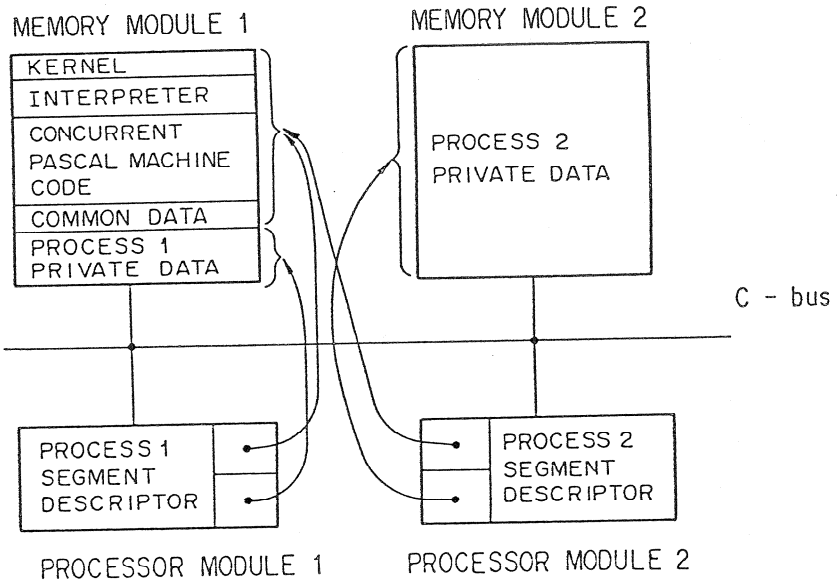


図 4．4　ACEシステムでのプログラム配置

キューに入っている時はレディキューに移されプロセッサが割り当てられるのを待つし、オンプロセッサなら、すぐに再実行する。

４．２．３．２　メモリの割り当て

Ｂｒｉｎｃｈ　Ｈａｎｓｅｎの仮想計算機を並列計算機にマップする場合の論理的メモリアロケーションは図４．２と同じであるが、物理的配置は色々考えられる。各プロセッサにローカルメモリを持たせる方法も考えられるが、ＡＣＥシステムはキャッシュメモリ方式を採っており、各プロセッサは大きなローカルメモリを持っていないため、プログラムはローカル、共有いずれも共有メモリに割り当てる。この方法は、プロセッサにプロセスを動的に割り当てる場合に有利である。各プロセスのアドレス空間は、図４．４に示すように、各プロセッサモジュールのセグメントデスクリプタにより生成している。

４．２．３．３　モニタの相互排除

各モニタは一時に高々１プロセスからしかアクセスされないよう制御する必要がある。モニタの相互排除法としては、全部のモニタを一括して相互排除する方法と各モニタ別に相互排除する方法が考えられる。前者はモニタルーチンをすべて１つの共有メモリに置き、共有メモリ単位で相互排除してしまえば簡単に実現できる。しかし効率の点で後者の方が優れているため、後者を選んだ。モニタ別に相互排除する方法では、前項で示した並列制御命令が別のプロセッサで同時に実行される可能性がある。そうするとそれらの命令の中ではプロセッサをプロセスに割り当てるスケジューラにアクセスするため、矛盾が発生する可能性がある。ＡＣＥシステムでは、同期モジュールを用いてこれ等の命令や割り込み処理が複数のプロセッサで同時に実行されても問題がないように制御している。同期モジュールを用いたプロセスの相互排除を次に簡単に示す[Fur77b]。クリティカルリージョン（この場合スケジューラ）へアクセスしようとするプロセスは同期モジュールにリクエスト（ＲＥＡＤ）を出し、クリティカルリージョンを出る時は同期モジュールにリクエスト（ＷＲＩＴＥ）を出す。同期モジュールはクリティカルリージョンへのアクセス要求にユニークな番号を与えるためのインクリメントカウンタ（ＩＮＣ）とクリティカルリージョンにアクセスしているプロセスとアクセス待ちしているプロセスの和を示すカウンタ（Ｔｅｓｔ＆Ｓｅｔ）を持ち、それらの値はＲＥＡＤリクエストでバスに読みだされる。ＩＮＣとＴｅｓｔ＆ＳｅｔはＲＥＡＤでインクリメントされ、Ｔｅｓｔ＆ＳｅｔはＷＲＩＴＥでデクリメントする。クリティカルリ

ージョンへアクセスしたいプロセスはＲＥＡＤで２つのカウンタを読み、Ｔｅｓｔ＆Ｓｅｔが０ならクリティカルリージョンにアクセスする。あるプロセスがクリティカルリージョンにアクセスしていればＴｅｓｔ＆Ｓｅｔは０でないので待つ。クリティカルリージョンへのアクセスを終えたプロセスは、自分がＲＥＡＤで読んだＩＮＣの値に１を加えＷＲＩＴＥリクエストでこの値をブロードカストする。

このＷＲＩＴＥで、同期モジュールのＴｅｓｔ＆Ｓｅｔはデクリメントされ、アクセス待ちしていたプロセッサはブロードカストされた値を自分がＲＥＡＤで読んだＩＮＣの値と比べることにより自分の番がきたか否かを知る。以上が同期モジュールの原理で、先に示したプロセススケジュールに関する命令では、その実行の前に同期モジュールにリクエストを出すし、Ｉ／Ｏ割り込みではＩ/Ｏプロセッサが割り込む前にリクエストを出すことにより相互排除が実現できる。

４．２．４　並列計算機における並列Ｐａｓｃａｌマシンの性能評価

並列計算機上で並列Ｐａｓｃａｌプログラムを走らせる利点としては、プロセスを並列実行できることとプロセス切り換えのオーバヘッドを減少させられることが考えられる。一方プロセスの並列実行を妨害する主な要因としてはモニタプロシージャ入り口での待ちとディレイがある。本項では、並列計算機上で並列Ｐａｓｃａｌプログラムを並列処理するときの性能をシミュレーションと実測で示す。

最初のシミュレーションはモニタ入り口と共有資源での待ちに関するものである。あるプロセスがモニタにアクセスする際、そのモニタが他のプロセスにアクセスされていれば、モニタ入り口で待たねばならない。又、いくつかのモニタが一つの共有メモリに入っていると、他のモニタがアクセスされていれば、その間待たねばならない。ここでは各プロセスが稼働している割合を、モニタルーチンへのアクセス頻度とモニタ内での共有メモリへのアクセス頻度をパラメータにして求めた。シミュレーションモデルは図４．５であり、プロセッサのマシンサイクルで動作するバスと１つの共有メモリ、ｎ台のプロセッサから構成され、プロセスはプロセッサに一対一に割り当てられる。また各プロセスのプライベートルーチンは各プロセスのローカルメモリに格納され、モニタルーチンのみを共有メモリに置くものとする。図４．５（ｂ）はプログラムのモデルで、ジョブが左から右へ次々と各プロセスによって処理されて行く様子をモデル化したものである。各プロセス（両端のプロセスは除く）はα／２の確率で２つのモニタにアクセスし、モニタ内ではβの確率で共有メモリにアクセスする。図４．６は、プロセッサ数を変えていった時のプロセッサの稼働率である。αが０．１ではプロセッサ数が１０台でもさほど影響がない。αが０．５というような高いアクセス頻度では、別のシステム設計によるアプローチが必要であろう。大体の目安としては共有資源の利用率が５０％以下ではモニタ入り口、共有資源での待ちの影響がないと言えよう。ここでβ＝１というのは、モニタアクセスの間中共有メモリを独占するもので、あるモニタ実行中他のモニタの実行は待たされる。我々の採用したモニタ別に相互排除する方法はβ＝１ではないため共有メモリの待ちが減っている様子がよく分かる。

第二のシミュレーションはプロセスのバランスに関するものである。多くのプロセッサが協力して仕事をするようなシステムでは、全体の性能が最も負荷の大きいプロセッサによって決まる場合が多く、並列計算機の利点が生きない場合も少なくない。ここでは、プロセスの不均衡の影響を生産者・消費者モデルについて求めている。生産プロセスは平均t_α時間でデータを生産しバッファに送る。この時バッファが満状態なら生産プロセスはディレイで

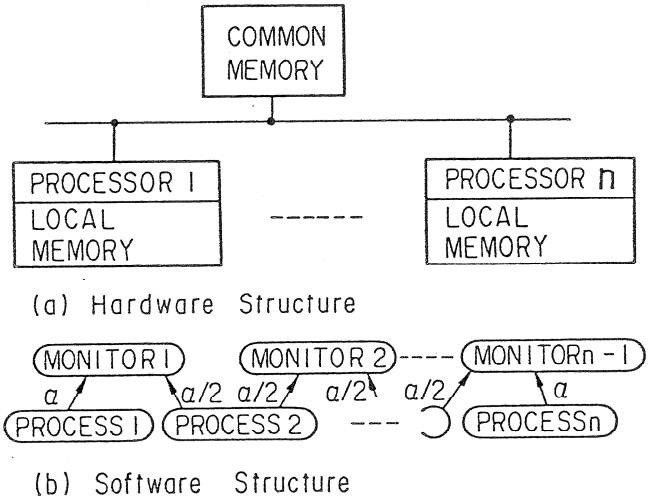


図 4．5　シミュレーションモデル

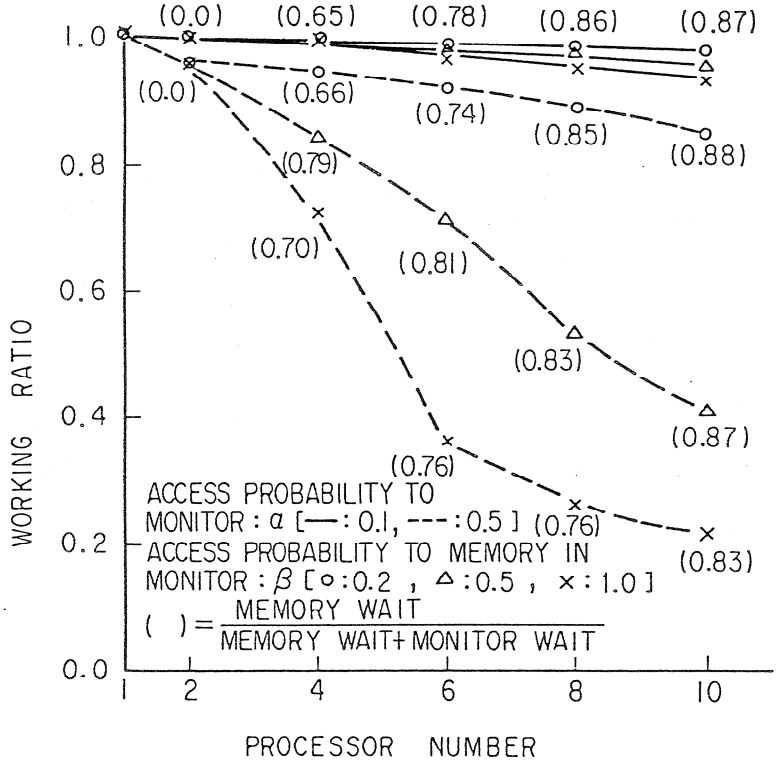
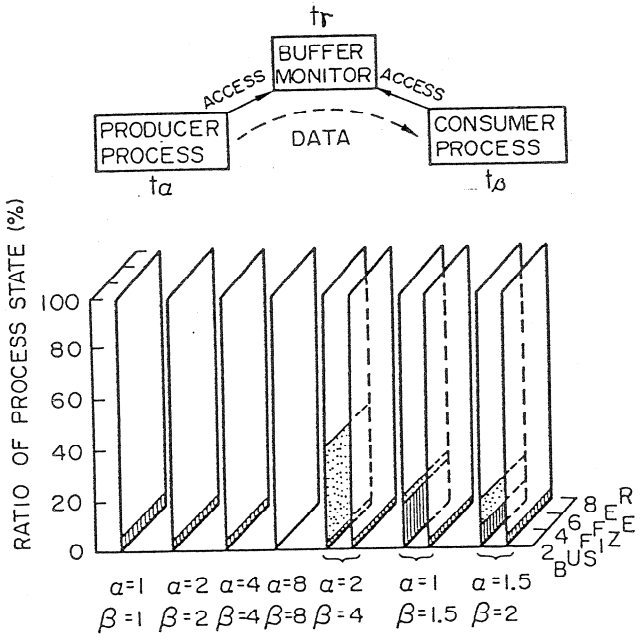


図 4．6　プロセッサの稼働率

待たされる。消費プロセスは平均t_β時間毎にデータをバッファから取り出し処理するがバッファが空ならデータが到着するまでディレイで待つ。ここでバッファはモニタルーチンにより操作され、モニタでの平均処理時間はt_γとする。図４．７はこのような状況でのプロセスの実行、モニタ入り口での待ち、ディレイの割合をモニタプロシージャへのアクセス頻度とバッファサイズをパラメータにして求めたものである。この結果は負荷バランスの良いシステムが効率よく働くという当然のことを示しているが、バッファサイズがあまりシステムに大きな影響を与えていないことがわかる。

最後は並列Ｐａｓｃａｌプログラムを並列計算機上で実際に走らせて行った実測である。プログラム１は２プロセスプログラムで、プログラム２は３プロセスである。これらのプログラムを１台と２台のプロセッサで実行した結果が図４．８である。このうちプログラム１は図４．７と同形でαとβはそれぞれ約１．５と２．０であり図４．７の結果と大体一致する。

以上から分かるとおり、並列計算機の性能はアプリケーションにより大きく異なる。実測のプログラムはプロセスのバランスが大体取れているがモニタへのアクセス頻度はかなり高いものであった。ＳＯＬＯシステムのばあいはプロセスのバランスは悪く、モニタへのアクセス頻度は少なく、２台のプロセッサによる性能向上は１０％程度であった。



Blank region : Run state.
Doted region : Delay state.
Hatched region : Waiting state at monitor procedure entry.
t_α = Average time required to produce datum.
t_γ = Average processing time of monitor procedure.
t_β = Average time required to consume datum.
$\alpha = t_\alpha / t_\gamma$, $\beta = t_\beta / t_\gamma$

図 4．7　生産者・消費者モデル

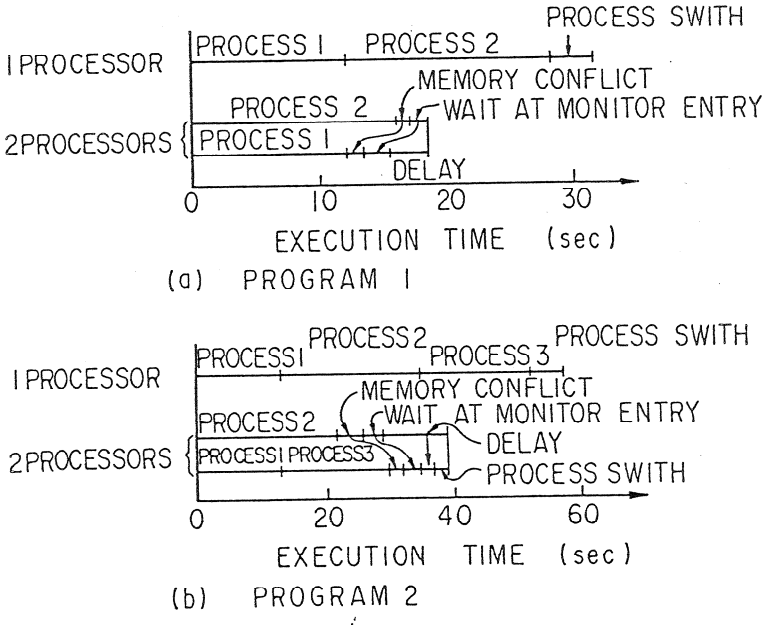


図 4．8　ＡＣＥシステムでの実測結果

４．３　高級言語計算機を実現するマイクロ命令の静的使用特性

工業技術院大型プロジェクトパターン情報処理システムで開発されたマイクロプログラマブルプロセッサＰＵＬＣＥは、様々な高級言語計算機実現に用いられ、そこで作られたマイクロプログラムの総ステップ数は約７万に達した。本節では、それらのマイクロプログラムを解析しマイクロ命令の静的使用頻度をもとに高級言語計算機とマイクロ命令の関係を明らかにすると共に、ＰＵＬＣＥに対する評価を行い今後のマイクロプログラマブルプロセッサ設計へのデータを与える。

４．３．１　ＰＵＬＣＥを用いた計算機とマイクロプログラム

ＰＵＬＣＥは様々な計算機実現の基本要素とすることを目的に設計された。そのためＰＵＬＣＥには多くの計算機に共通な部分であるＡＬＵとレジスタのみが組込まれている。ＰＵＬＣＥを用いて計算機を作るには図４．９に示されるようにＰＵＬＣＥの周辺にシーケンサを付ける必要がある。シーケンサは次の２つの部分からなる。

（１）マイクロプログラムカウンタを管理しＰＵＬＣＥへのマイクロ命令の供給を制御する部分（図４．９のＳ部）。

（２）ＰＵＬＣＥとメモリ及びＩ/Ｏ装置等との間のデータ授受を制御する部分（図４．９のＭ部）。

表４．６にはＰＵＬＣＥを用いて作られた計算機名とそれらの相違が示してある。

ＰＵＬＣＥの構成は図４．１０であり、Ｒ、Ｌ、Ｔ３本のバスにＡＬＵ，シフタ，レジスタ、レジスタファイルが接続されている。ＰＵＬＣＥの基本動作は、レジスタの被演算データ（１６ビット）をＲ及びＬバスでＡＬＵ又はシフタに送り、演算結果をＴバス経由でレジスタに格納するというものである。ＰＵＬＣＥと外部とのデータ交換はインタフェースレジスタ（ＩＦＲ）を通して行われる。図４．１１はＰＵＬＣＥチップ上のレイアウトを示している。図４．１２はＰＵＬＣＥマイクロ命令で５つの型が存在し、その基本機能は次の通りである。

（１）ＲＲ型：レジスタ間演算で、ＲとＴ(Ｌ)フィールドで示されるレジスタの値にＯＰフィールドで示される演算を施し、結果をＴ(Ｌ)フィールドで示されるレジスタに格納する。ＳＴフィールドはステータスレジスタ（ＳＴＲ）の１ビットを指定し、その０、１により次の命令をスキップするか否か指定する。ＭＫフィールドはＲ、Ｌ、Ｔバスのマスクを指定する。その他のフィールドは次の意味を持つ。ＳＳ：演算結果のステータスをＳＴＲに格納す

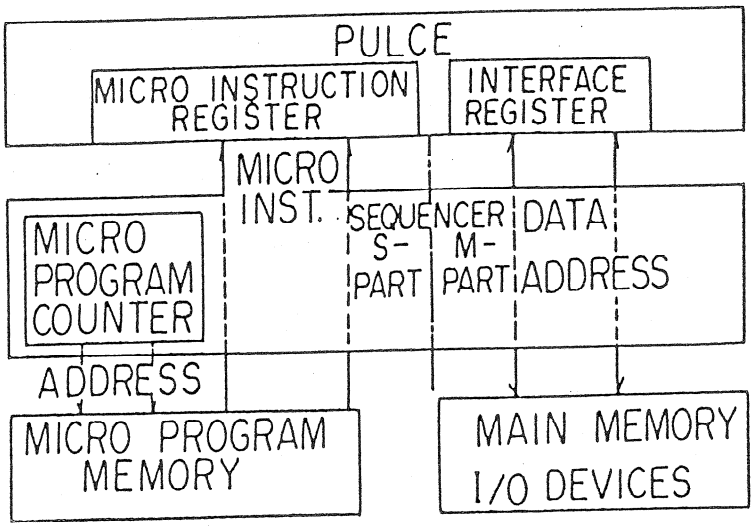
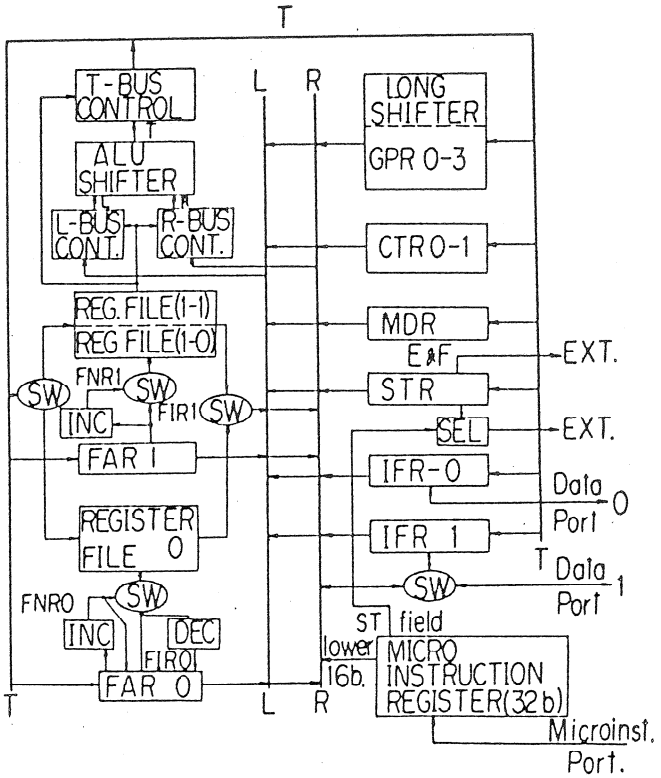


図 4．9　PULCEを用いた計算機構成

表 4．6　PULCEを用いた計算機とマイクロプログラム

計算機 名	シーケンサ	文献	マイクロプログラム 番号	名	性質	作者
ACE（適応構造形計算機複合体）	1	(3)	1	ACEのOS	O	1
			2	並列PASCALマシン	E・I	2
			3	NOVAエミュレータ	E	3
PMCS（パーソナルコンピュータ）	2－1*	(4)	4	TOSBAC 40 エミュレータ	E	4
			5	PASCALマシン	I	5
			6	手書き文字認識	O・E	6
LISPマシン	2－2*	(5)	7	LISPマシン	I・O	5
磁気バブルデータベースマシン	2－3*	(5)	8	データベースマシンのOS	O	1
			9	〃　コンパイラ	O	7
			10	〃　インタプリタ	I	1
EPOS（ポリプロセッサ）	2－4*	(5)	11	EPOSのOS	O	多数
			12	APLマシン	I	8
			13	PASCALマシン	I	9

＊：2－nのnはシーケンサのうちメモリ管理部（M-PART）の違いを示す．



GPR: General Purpose Reg., STR: Status Reg., IFR: Interface Reg.,
CTR: Counter, MDR: Mode Reg., REG.FILE: Register file,
FAR, FNR, FIR: Reg. file control reg..

図 4．10　PULCEのブロック図

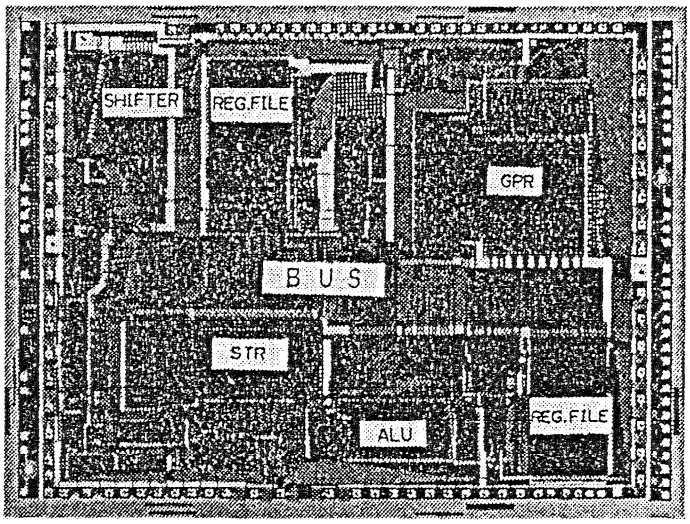


図 4．11　PULCEのレイアウト

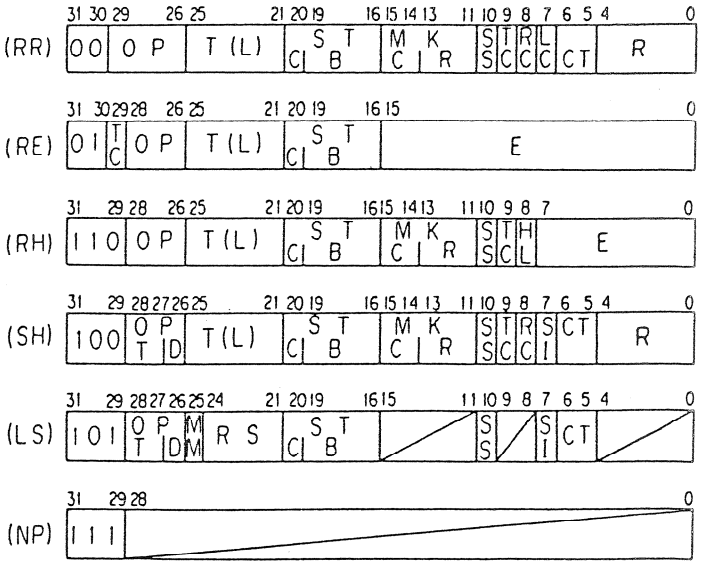


図 4．12　マイクロ命令の形式

るか否か。ＴＣ：演算結果をレジスタに格納するか否か。ＬＣ：Ｌバスにデータを乗せるか否か。ＲＣ：Ｒバスにレジスタの値を出すか、Ｒフィールドの値をそのまま出すか。ＣＴ：カウンタ（ＣＴＲ）のデクリメント指定。

（２）ＲＥ型：Ｔ(Ｌ)フィールドで示されるレジスタとＥフィールドの１６ビットエミット値を演算し、結果をＴ(Ｌ)フィールドのレジスタに格納する。

（３）ＲＨ型：Ｔ(Ｌ)フィールドとＥフィールドのエミット値を演算し、結果をＴ(Ｌ)フィールドのレジスタに格納する。ＨＬフィールドは、エミット値をレジスタの上位８ビットと演算するか、下位８ビットと演算するかの指定。

（４）ＳＨ型：Ｔ(Ｌ)フィールドのレジスタの内容をＲで示されるビット数（０～１５）だけシフトする。ＳＩはキャリーのシフトイン指定。

（５）ＬＳ型：汎用レジスタ（ＧＰＲ０～３）を複数個連結して１ビットシフトする。連結形態はＲＳフィールドで指定。ＭＭは１ビットシフトを繰り返し実行させる指定。

（６）ＮＰ型：シーケンサを制御する命令。ジャンプ、ジャンプサブルーチン、メモリアクセス命令等がある。

表４．６はＰＵＬＣＥを用いた計算機とそこで作られたマイクロプログラムの名前、性質、プログラマを示している。プログラムの性質は処理内容により次の３種類に分けた。

（１）Ｅ型：機械語等、低レベルの命令のエミュレータ。

（２）Ｉ型：高級言語計算機の中間言語の命令等、高レベル命令のインタプリタ。

（３）Ｏ型：ＯＳ、コンパイラ等、プログラム全体で１つの仕事をするもの。

又これらのいくつかの性質を合せ持つものには、Ｅ・Ｉ等複数併記する。

４．３．２　測定結果と考察

本項では表４．６に示す１３のマイクロプログラム（計６８７８９ステップ）で用いられたマイクロ命令の使用特性に基づいて高級言語計算機とマイクロ命令の関係、アーキテクチャの評価等について考察する。ここでの議論は、ＰＵＬＣＥアーキテクチャの影響を受けていることは明らかであるが、できるだけ一般性のあるように整理している。

４．３．２．１　主操作の使用頻度

表４．７は、マイクロ命令中の操作（ＯＰ）フィールドで指定される主な操作の使用頻度を、マイクロプログラム別に整理したものである。表４．７から以下のことを読み取ることができる。

（１）ＰＵＬＣＥ用命令（ＲＲ，ＲＥ，ＲＨ，ＳＨ、ＬＳ型）とシーケンサの命令（ＮＰ型）の比は約一対一である。

（２）ＰＵＬＣＥ用命令について見ると、データ移動（ＭＯＶＥ）命令が５０％以上を占める。ＬＩＳＰマシンではＭＯＶＥ命令が少ないが、これはシーケンサの機能を充実し、データ移動をできるだけシーケンサに行わせているためである。

（３）１０進加減算命令を４種類用意したがほとんど使われなかった。これは今回の応用の中に事務処理的なものがなかったためであり、この結果だけから１０進命令の価値を判断するのは尚早と思われる。

（４）マイクロプログラムはアセンブリ言語のプログラムに比べてハードウェアに近いため、エミットデータとの演算が多く用いられると考え、エミット演算の充実を図った。その結果ＲＥ型は予想どおり高い使用頻度を示したがＲＨ型はほとんど使われていない。ＲＨ型のＲＥ型に対する利点は、マスクが可能なことと、演算結果のステータスをＳＴＲにセットしない指定ができることにある。しかし実際にはＲＨ型でなければならない場合は少なく、ほとんどＲＥ型で済まされていると思われる。

（５）ＳＨ型の特別シフトとは、ソースデータを汎用レジスタに入れておき、シフト結果を他のレジスタに入れるものである。これはソースデータを破壊しないため、命令のデコードなどに便利な機能と考え設けたものだが、使用しているものとしていないものが極端に分かれた。これはよく使い方を知らせなかったためではないかと思われる。

（６）ＬＳ型は８７％以上が論理シフトである。

（７）全プログラム中、全く使われなかった命令は、ＲＨ、ＲＥ型のＮＯＲとパターンマッ

表 4．7　マイクロ命令の静的使用頻度

プログラム番号			1	2	3	4	5	6	7	8	9	10	11	12	13	平均
総ステップ数			3,192	1,760	336	2,581	2,439	3,008	11,267	1,827	7,127	6,513	18,043	5,483	5,213	
PULCE用命令	RR型	MOVE	12.84	15.45	34.21	21.08	27.63	22.14	7.76	17.02	27.11	27.13	19.79	20.55	18.20	20.84
		OR	0.29	1.93	2.98	5.00	2.34	5.05	1.07	1.31	0.17	0.18	1.51	2.32	2.74	2.07
		XOR	0.53	0.23	0	2.29	1.76	0.37	0.34	0.38	0.21	0.18	0.13	0.11	2.03	0.66
		AND	0	0.11	0.30	0.62	0.57	0.27	0.04	0.11	0.03	0.17	0.22	0.07	0.27	0.21
		ADD	0.56	0.85	3.57	2.13	5.62	3.95	1.80	0.71	0.68	2.49	3.27	2.26	3.80	2.44
		SUB	0.13	0.74	0.89	1.98	3.48	2.16	1.72	0.76	0.45	2.18	1.73	1.19	4.55	1.69
		十進演算	0	0	0	0	0	0	0.03	0	0	0	0.01	0	0	0.00
		その他	0	0.12	2.10	0.04	0.33	0.06	0.01	0	0.21	0.17	0	0	0.02	0.24
		RP型計	14.35	19.43	44.05	33.13	41.74	34.01	12.75	20.31	28.85	32.50	26.66	26.50	31.61	28.15
	RE型	MOVE	19.33	3.24	2.98	3.91	9.06	10.57	10.62	9.41	8.19	6.77	10.26	11.34	15.33	9.31
		OR	1.07	2.05	4.17	0.04	0.04	1.06	0.56	0.49	0.29	0.71	0.74	1.06	0.38	0.97
		XOR	1.69	0.34	1.79	0.19	1.76	0.40	1.50	1.37	1.11	4.15	0.02	0.05	0.61	1.15
		AND	2.57	7.78	5.65	1.90	1.11	1.89	4.47	3.12	3.03	2.18	2.63	3.57	1.75	3.20
		ADD	3.73	4.20	1.49	1.05	0.29	5.22	0.87	4.00	10.64	8.37	5.87	4.85	1.96	4.04
		SUB	0.85	5.00	0.60	2.75	0.16	4.12	2.52	1.86	1.22	1.84	8.74	5.14	3.64	2.96
		RE型計	29.23	22.61	16.67	9.84	12.42	23.27	20.54	20.25	24.48	24.01	28.27	26.03	23.67	21.64
	RH型	RH型計	0	0	0.30	2.63	0	0	0	0	4.08	0.02	0.02	0.02	0	0.54
	SH型	回転シフト	0.03	0.39	0.89	0.23	0	0	0.02	0.11	0.25	0.23	0.14	0.04	0.10	0.19
		論理シフト	1.03	5.17	4.46	2.71	1.19	1.79	0.93	1.26	0.47	0.32	2.12	2.50	1.67	1.97
		算術シフト	0	0.68	0.30	0.08	0.37	0.70	0	0	0	0.06	0	0	0.02	0.17
		特別シフト	0	2.27	2.68	0	0	0	0	0	0	0	0	0	0.12	0.39
		SH型計	1.07	8.52	8.33	3.02	1.56	2.49	0.95	1.37	0.72	0.61	2.26	2.54	1.90	2.72
	LS型	論理シフト	0.06	0.12	0	0.97	0.53	1.23	0.18	0	0.10	0.15	0.14	0.16	0.69	0.33
		その他	0.06	0	0	0.32	0.08	0	0	0	0	0	0	0	0.23	0.05
		LS型計	0.13	0.12	0	1.28	0.62	1.23	0.18	0	0.10	0.15	0.14	0.16	0.92	0.39
	PULCE用命令計		44.77	50.68	69.35	49.90	56.33	61.00	34.41	41.93	58.23	57.30	57.35	55.24	58.10	53.43
外部命令	ジャンプ		8.05	22.33	22.03	28.24	19.13	9.31	18.86	11.16	11.08	13.25	9.22	16.36	18.40	15.96
	ジャンプサブルーチン		6.55	13.81	0	5.81	5.78	5.12	10.83	5.15	8.42	3.95	12.74	6.66	9.76	7.28
	リターンサブルーチン		3.70	3.41	0	1.32	3.90	1.00	1.81	2.85	1.41	1.72	3.33	1.37	3.49	2.25
	マイクロプログラムメモリアクセス		—	—	—	2.37	8.57	13.68	8.98	20.75	0.45	1.48	1.55	0	0	4.45
	メモリリード		11.81	1.48	3.33	1.05	2.74	0.30	8.48	2.02	4.13	6.57	3.55	5.53	2.72	4.13
	メモリライト		9.68	2.50	2.41	0.77	0.16	0.66	6.52	2.29	5.38	5.59	3.02	5.91	2.17	3.62
	その他		15.44	5.80	2.88	10.55	3.37	8.94	10.12	13.85	10.08	10.15	9.26	8.94	5.35	8.83
	外部命令計		55.23	49.32	30.65	50.10	43.67	39.00	65.59	58.07	41.77	42.70	42.65	44.76	41.90	46.57

注：① 表中の値の和は，丸め誤差のため必ずしも計の値と一致しない．
② ACEシステムには，マイクロ命令で直接マイクロプログラムメモリを読み書きする命令がない（表中の－印）．
③ PULCE用命令には無操作命令がないため，同一レジスタ間のデータ移動（MOVE）命令などで無操作を代用している場合がある．

チである。パターンマッチとは4ビットでできるビットパターンの集合を扱えるものである。これらの命令は少し難解なため“難しい命令はあまり使われない”という機械語レベルの話がマイクロレベルにもあてはまるようである。

（8）マイクロプログラムの種類による差は表4．7からはあまり明確にならず、E型プログラムのシフトが多少多い程度であった。

4．3．2．2　スキップ

マイクロ命令中には5ビットのスキップ（ST）フィールドがあり、この内4ビット（B部）でステータスレジスタ（STR）の任意の1ビットを指定する。そしてSTフィールドのC部が“0”のときは、STRの指定されたビットが“1”なら次の命令をスキップする。一方、STフィールドのC部が“1”の時は、STRの指定されたビットが“0”なら次の命令をスキップする。表4．8はSTフィールドで指定されるスキップ条件の指定率である。この結果、83．58%スキップ指定がなく（次の命令をスキップしない）、スキップ指定がある場合では演算結果が零か否かでスキップする場合が多い。表4．9は、マイクロ命令中の主操作とスキップ指定の関係を示している。即ち、どの命令でどれくらいスキップ指定が行われるかを示している。これを見るとXOR（排他的論理和）、AND、SUB命令の結果でスキップする場合が多い。そしてこれらの命令の演算結果のステータスで重要なものは零か否かであり、表4．8の“演算結果が零によるスキップ”が多いこととも符合する。スタック“満”と“空”は両方共“満”と“空”の補集合でスキップを指定するという特徴がある。表4．8の“その他”の中にはSTR中にユーザのアクセスできるフラグを4ビット用意して、その状態によってスキップするものがあるが、使われたのは1ビットだけで、他の3ビットはほとんど使われなかった。スキップ条件は31種類用意したが、主に使われたのは表4．8に示すように比較的少ない。これに対し、STRがチップ上に占める領域は図4．11のとおり非常に大きい。このようなことを考えると、STRやSTフィールドの圧縮は検討に値すると思われる。

スキップ指定の次の命令がジャンプ命令であると条件ジャンプを意味する。このようにスキップ指定とジャンプ命令が対になる割合は、スキップ指定の約85%と非常に高い。そこで、演算を行い、その結果で条件ジャンプする命令を用意すればステップ数を節約できる。しかし、この種の命令には、演算に関係するフィールドとジャンプアドレスフィールドが必要になり、実行サイクルも演算とジャンプの2マイクロサイクルになる。演算に関するフィ

表 4．8　スキップ指定

スキップ指定なし（スキップしない命令）…83.58％
無条件スキップ（常にスキップする命令）… 0.46％
条件スキップ（以下この詳細）　　…15.96％

スキップ条件		指定率	スキップ条件		指定率
演算結果	零	4.92	演算結果	負数	0.43
	否零	4.56		否負数	0.83
演算結果	正数	0.40	演算結果	偶数	0.10
	否正数	0.50		奇数	0.75
スタック	満	0	スタック	空	0
	否満	0.20		否空	0.62
キャリー	有	0.48	その他		1.61
	無	0.56			

表 4．9　操作に対するスキップ指定率

命令型	操作名	スキップ指定率	命令型	操作名	スキップ指定率
RR型	OR	10.21	RE型	OR	14.01
	XOR	37.54		XOR	92.38
	AND	44.07		AND	38.45
	ADD	4.53		ADD	2.20
	SUB	53.94		SUB	69.80
	その他	25.58		その他	1.18
RH型	XOR	97.06	SH型	−	5.19
	その他	29.79	LS型	−	22.38

表 4．10　レジスタの使用ひん度

	ビット数	数	使用ひん度(%)
汎用レジスタ（GPR）	16	4	28.45
インタフェースレジスタ（IFR）	16	2	37.39
レジスタファイル制御レジスタ	16, 4	6	6.31
専用レジスタ（STRなど）	16	4	7.08
レジスタファイル（RF）	16	32	20.77

ールドとアドレスフィールドを３２ビットに入れようとすると、いずれかのフィールドを圧縮することとなり命令形式が変則的になる。又、マイクロ命令長を３２ビット以上にすることはＬＳＩのピン数の制限から難しい。更に２マイクロサイクル命令の導入は均一性を乱し、ハードウェアが複雑になり集積度と高速化の点で好ましくない。以上を考慮するとＰＵＬＣＥにおける分岐命令は妥当なものと考えられる。

４．３．２．３　レジスタ

マイクロ命令中のレジスタフィールド（Ｔ(Ｌ)及びＲ）で指定されるレジスタの頻度を表４．１０に示す。インタフェースレジスタ（ＩＦＲ）はＰＵＬＣＥと外部とのデータ交換の仲介となるレジスタである。データやアドレスを外部に送出するときは一度ＩＦＲにデータをセットしてから出力命令を出すし、入力の際にもデータは外部からＩＦＲにセットされ、その後ＰＵＬＣＥ内で使用可能となる。このためＩＦＲが使用される割合は高くなり３７．３９％に達した。ＰＵＬＣＥのように外部との交信にレジスタを介する方法に対して、マイクロコンピュータ等によくあるように、アドレスバスとデータバスを直接外部に引き出し、内部レジスタの内容をアドレスとして１命令でデータを読み出す方法もある。後者は一般のマイクロコンピュータのようにアーキテクチャがある程度固定している場合には命令数を減らす意味で有効であるが、アドレスやデータ幅が１６ビット以下であったり、外部にメモリ制御用レジスタを幾つも置くような応用ではＰＵＬＣＥ方式の方が有利な場合が多い。汎用レジスタ（ＧＰＲ）は４個あるが、その使用頻度はＧＰＲ０＝１２．０７％、ＧＰＲ１＝７．０８％、ＧＰＲ２＝５．４６％、ＧＰＲ３＝３．８３％となっている。レジスタファイルへのアクセス率は低く、レジスタファイルの語数に不足はないと思われる。レジスタファイルに関しては、レジスタファイル中の語を間接指定できるレジスタ（レジスタファイル制御レジスタ）を用意し、ゲスト計算機のレジスタ割り当てに便利なようにしたが、実際には命令中から直接レジスタファイルの語を指定する場合が多かった。総じてレジスタ数や機能割り当てに対する不満はないように思われた。

４．３．２．４　シフト

シフト操作はエミュレーション等のフィールド抽出に重要と考え、０～１５ビットのシフトを１マイクロ命令で実行するバレルシフタを用意した。表４．１１はシフト量の測定結果であり、１、２、４、８ビットのシフトは合計で８０％近くに達する。シフタがチップ上に

占める領域は図４．１１に示すとおり大きいが、８ビットシフト等使用率が高いことを考えるとバレルシフタは有効に使われているといえよう。１、２、４、８ビット以外のシフトは少なく、機能的には１、２、４、８ビットシフトだけでも性能にさほど影響はないと思われる。しかしハードウェア量でいえばＰＵＬＣＥでは、１、２、４、８ビットシフタを直列につなぎ、これらの組み合せで、０～１５ビットのシフトを行っており、１、２、４、８ビットだけのものに比べてさほど増えていない。

ＬＳ型はＧＰＲ０～３を幾つか連結して１ビットシフトするものである。レジスタの連結形態は命令中のＲＳフィールドで１６種類指定できるが、実際に使われたものは限られており表４．１２に示すものである。又、ＬＳ型ではマルチモード（ＭＭフィールド）といい、ＬＳ型命令を何サイクルか繰り返し実行するモードを用意している。マルチモードが指定される割り合いは６０％と非常に高かった。

表 4．11　シフト量

シフト量(bit)	0	1	2	3	4	5	6	7
利用率(%)	0.2	29.7	3.7	4.7	11.0	3.2	1.9	2.5
シフト量(bit)	8	9	10	11	12	13	14	15
利用率(%)	34.7	0.4	1.3	0.8	2.3	2.0	1.2	0.4

表 4．12　ロングシフトの連結形態

汎用レジスタの結合形態	百分率(%)
GPR0-GPR1	43.9
GPR2-GPR3	28.3
GPR0-GPR1-GPR2-GPR3	21.5
GPR0-GPR1-GPR2	3.9
GPR1-GPR2-GPR3	1.5
GPR1-GPR2	1.0

表 4．13　マスク利用率

(a)　プログラム別マスク利用率

プログラム番号	1	2	3	4	5	6	7	8	9	10	11	12	13
マスク利用率(%)	0	10.4	4.6	8.4	0	0	1.2	0	5.5	0	0.1	3.4	2.3

(b)　マスクの種類別利用率

マスクの種類	マスクしない	Tバスマスク	Lバスマスク	Rバスマスク	R,L,Tバスマスク
利用率(%)	97.88	0.85	0.13	0.90	0.23

表 4．14　数値定数

PULCE	定数	0	1	2	3	4	5～7	8	9～16	17～32	33～max
	利用率	12.8	38.4	14.4	2.4	4.2	4.6	2.9	5.1	2.8	12.5
XPL	利用率	15.6	17.0	7.9		6.0		9.5		9.4	34.6
	定数	0	1	2～3		4～7		8～15		16～31	32～max

４．３．２．５　マスク

エミュレーションにマスクは有効であろうと考えＲＲ、ＲＨ、ＳＨ型命令中に５ビットを取って、Ｒ、Ｌ、Ｔバスに対する様々なマスク指定を用意した。表４．１３（ａ）はマイクロプログラムごとのマスク利用率（マスクフィールドにマスクを指定する割合）を、（ｂ）はマスクの種類別利用率を求めたものである。マスクは比較的マイクロプログラムの性質（表４．６）を反映しており、Ｅ、Ｉ型でよく利用され、Ｏ型ではあまり利用されていない。マスクはエミュレーション等で有効性を発揮しているが、平均すると利用率は低い。この点を考えると、応用毎にマイクロ命令のマスクフィールドの意味を変えるとか、マスクフィールドを圧縮するとか、工夫の余地があると思われる。

４．３．２．６　数値定数

表４．１４にはマイクロプログラム中で使われる数値定数の値を示した。値としては１５以下のものが７６．７％を占めている。又、同表にはＡｌｅｘａｎｄｅｒらがＸＰＬプログラムに対して調べた数値定数の結果[Ale75]を示したが、これに比べるとマイクロプログラムでは“１”、“２”が非常に多いことが分かる。これは数値計算プログラム等では、様々な値の数値が使われるのに対し、マイクロプログラムではゲストマシンのレジスタのインクリメントやデクリメント等小さな値が使われることが多いことによると思われる。

４．３．２．７　ジャンプ、ジャンプサブルーチン

図４．１３はジャンプ命令の飛距離の累積で、横軸は飛距離を表すのに必要なビット数である。図中の（･･･○･･･)線は正方向への飛距離（･･･× ･･･）線は負方向はの飛距離、実線はその合計である。又、同図はＡｌｅｘａｎｄｅｒらがＸＬＰプログラムで調べた無条件ジャンプの飛距離も参考のために示した。ＰＵＬＣＥの場合がＸＰＬに比べて近距離へのジャンプが多いのはＰＵＬＣＥに条件ジャンプ命令が無いことと、プログラムサイズが小さいためと思われる。一般に条件ジャンプは飛び先が近い。ＰＵＬＣＥでは条件ジャンプをステップ指定とジャンプ命令の組み合せで実現するため、近距離へのジャンプが増える傾向にある。図４．１３を見る上で注意を要する点は、図中の曲線が２つの成分からできていることである。即ち、近距離へのジャンプはプログラマが直接記述した飛び先であるのに対し、遠距離へのジャンプはモジュールのリンクにより決まるものでプログラマが関与できない飛距離である。この２つの要素の境いは約１０ビット辺りである。

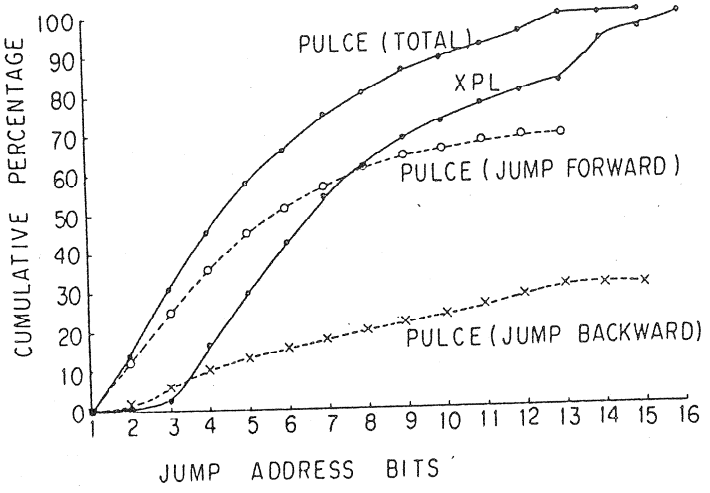


図 4。13　ジャンプ命令の飛距離

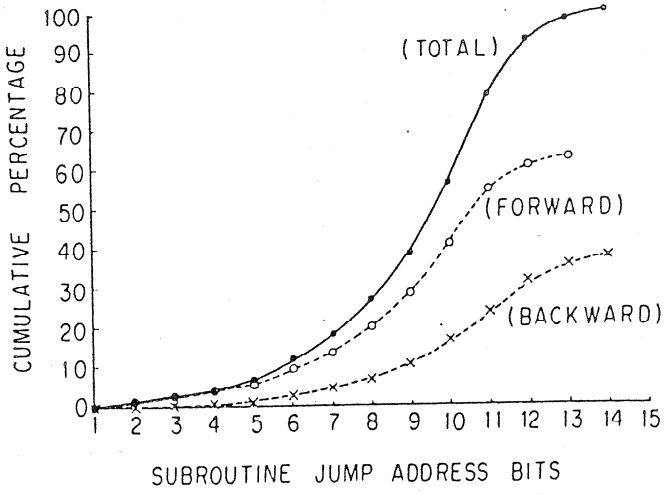


図 4。14　ジャンプサブルーチン命令の飛距離

図４．１４はジャンプサブルーチンの飛距離であり、図４．１３と性格が異なる。これは、プログラマがサブルーチンをプログラムの最初か最後にまとめて置くことによる。

４．４　まとめ

本章では、まず並列Ｐａｓｃａｌマシンの並列計算機へのインプリメントを通して、並列プログラムを並列計算機で処理する方法を示した。またここではマイクロプログラムで高級言語計算機を実現する方法を採ったが、ここで用いた方法はマイクロプログラム量の増加を抑えるものでマイクロプログラムメモリに制限がある場合等に有効な方法である。並列計算機上の並列Ｐａｓｃａｌマシンの実測とシミュレーションは並列プロセスの並列計算機上での振舞いをできるだけ一般性のあるように求めた。この結果、応用により差はあるものの並列プログラムを並列計算機で実行する利点は明らかになった。しかし測定結果は並列プロセス制御のオーバヘッドも少なくないことを示している。このことが、次章で示すハードウェアによる並列プロセス制御機構開発の動機となった。

４．３節は高級言語計算機とマイクロプログラムの関係を明らかにしようというものであった。結果はマイクロプログラムによる命令の静的使用特性の違いはマスク、シフト等にいくらか見受けられるが当初予想していたような差はなかった。即ち、マイクロプログラムの性格の差より、プログラムの方法やシーケンサの違いの方が大きな影響を持つということであった。又、マイクロプログラムで使用される命令は機械語によるプログラムと同様難しい命令は使われなかった。これは短期間に大規模なプログラムが作成されたということはあるが、ユーザにサンプルプログラム等を提供し設計者の意図を伝える努力が必要なことが分かる。

ＰＵＬＣＥは幅広い応用への適用を考えＶＬＳＩ化の枠内でできるだけ多くの機能を満辺なく取り入れる方向で設計された。しかしマイクロ命令の使用特性をみると、応用によらずマイクロ命令の利用率に大きな片寄りがあることは明らかである。このことを考慮すると使用頻度の低い命令や機能を減らして、使用頻度の高い命令や機能を充実するマイクロプロセッサ設計法も有力なアプローチと考えられる。

ここで示したＰＵＬＣＥに対する評価は今後のマイクロプログラマブルプロセッサ設計者に有用なデータとなろう。

第５章　並列構造記述モデルとそれを実現する並列計算機

５．１　まえがき

第４章ではバス結合された並列計算機上への並列Ｐａｓｃａｌマシンの実現を通して、マイクロプログラムによる高級言語計算機の実現法と、特殊ハードウェアを用いた並列制御機構を示した。このうち並列制御機構については、並列Ｐａｓｃａｌとバス結合並列計算機という特定の組み合せからボトムアップに出発したもので、特殊ケースの実現といった印象が強い。並列計算機を並列プログラミング言語用高級言語計算機として設計する方法の確立には、様々な並列高級言語や並列計算機に適用可能な一般性のある並列制御機構を導き、その実現法を示す必要がある。ここで示す方法は並列計算機を複数の計算機の集合ととらえるのではなく、１台の計算機として考え並列高級言語の持つ制御機構から基本機能を整理し、並列計算機の機械語に組み入れていく方法である。即ち、今までは並列計算機上での並列プロセスをサポートするものはＯＳであると考えられていたのに対し、並列計算機を１台の計算機と考えることにより、並列制御機構を並列計算機の機械語で実現するものである。これにより、並列プログラムを並列計算機で動作させるためのＯＳは不要となり、プログラミング言語を効率良くサポートできる。これは従来から行なわれてきたＯＳのハードウェア化の側面を持つが、並列プログラミング言語サポートという局面から機能が整理されるため、より高水準な命令を作ることができる[Fur83][Fur84]。

本節では、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プロセス単位に分割し、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。５．３節では、このモデルを効率良く実現すると共に、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。可変構造マルチリードメモリはＫＤＳＳ[Nak80]で用いられたマルチリードメモリを多層化しさらに可変構造を与えることにより、メモリでの衝突を減らすものである。プロセッシングエレメントとしてはマイクロ命令を圧縮し機械語を生成するアーキテクチャを示す[Fur81]。これはその後盛んになったＲＩＳＣアプローチ[Pat81]の先駆けである。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結果を示す。

５．２　並列構造記述モデル

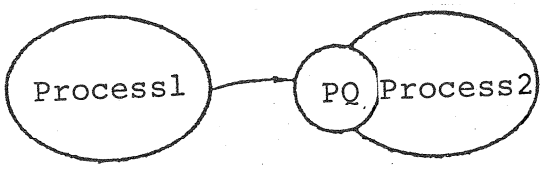
近年並列プログラミングの研究は急速に進み、構造的プログラミングの可能な様々な高級言語が提案されてきている。ＭＩＭＤ用Ａｄａ、並列Ｐａｓｃａｌ、ＳＩＭＤ用Ａｃｔｕｓ等がその代表である。本節ではこのような種々の並列プログラミング言語で書かれたプログラムの構造を簡潔な規則の下に整理して表現するモデルを与える。そこではプロセスと呼ばれる並列処理の基本単位と、それらの間の依存関係を数種類の待ち行列を用いて表しており、並列プログラムはプロセスのキューイングネットワークの形で表される。このモデルを作成するに当っては、単にプログラムの構造の表し易さだけでなく、並列計算機で効率良く実現できることも考慮している。プロセスの依存関係を表すのに待ち行列を選んだ理由は次のとおりである。まず、現在並列プログラム用高級言語の並列制御機構としては、Ｍｏｎｉｔｏｒ又はＣｏｍｍｕｎｉｃａｔｉｎｇ　Ｓｅｑｕｅｎｔｉａｌ　Ｐｒｏｃｅｓｓ（２．３．１参照）をベースとしたメッセージ交換が支配的である。そしてＭｏｎｉｔｏｒの実現には、共有資源のアクセスを制御する待ち行列が用いられ、メッセージ交換の実現にも柔軟性のあるメッセージバッファとして待ち行列が用いられる。又、プロセスのスケジュールには一般に待ち行列が用いられる。このような理由から本モデルでは並列プロセスの制御に４種類の性格の異なる待ち行列を用意している。図５．１はこのグラフ表現であり、図５．２は並列プログラムの構造を待ち行列を用いて表した例である。

一般にプロセスはシーケンシャルプログラムで、プロセス中には任意個の待ち行列を設定できる。プロセスから待ち行列へのアクセスはプログラム中の待ち行列操作命令で行われる。表５．１は待ち行列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。

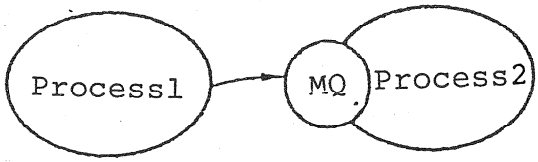
プロセスは図５．３に示す４種類の状態の何れかにあり、その状態は待ち行列操作命令で変化する。

次に各待ち行列と待ち行列操作命令に伴う状態変化を説明する。

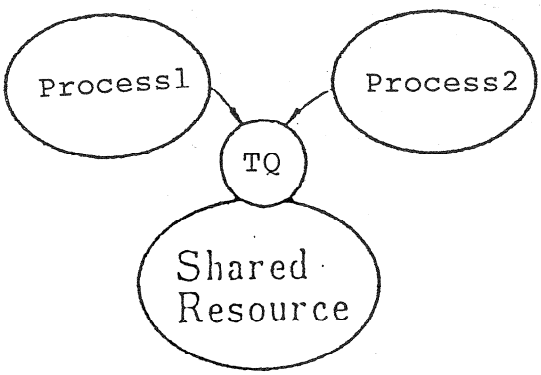
（１）Ｐｒｏｃｅｓｓ　Ｑｕｅｕｅ（ＰＱ）：これはプロセス間の同期を取るためのもので、ソースプロセス（図５．１（ｉ）のＰｒｏｃｅｓｓ１）がプログラム中でＥＮＴＥＲ　ＰＱ命令を出すと、ソースプロセスが、命令で指定するＰＱに入る。ＰＱに入るとプロセスはＷＡＩＴ状態になる。ＰＱはデスティネーションプロセス（図５．１（ｉ）のＰｒｏｃｅｓｓ２）に付属している。プロセスは自分のＰＱの先頭のプロセスを取り出し（ＧＥＴ命令）たり、取り出したプロセスを他の待ち行列へ入れる（ＰＵＴ命令）ことができる。ＧＥＴ命令



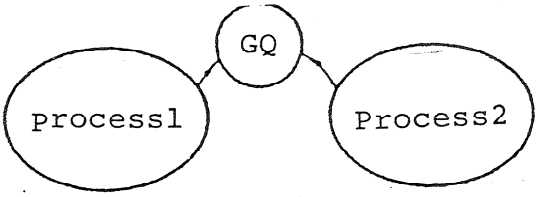
(i) Process Queue (PQ)



(ii) Message Queue (MQ)



(iii) Test&Set Queue (TQ)



(iv) Global Queue (GQ)

図 5．1　待ち行列のタイプ

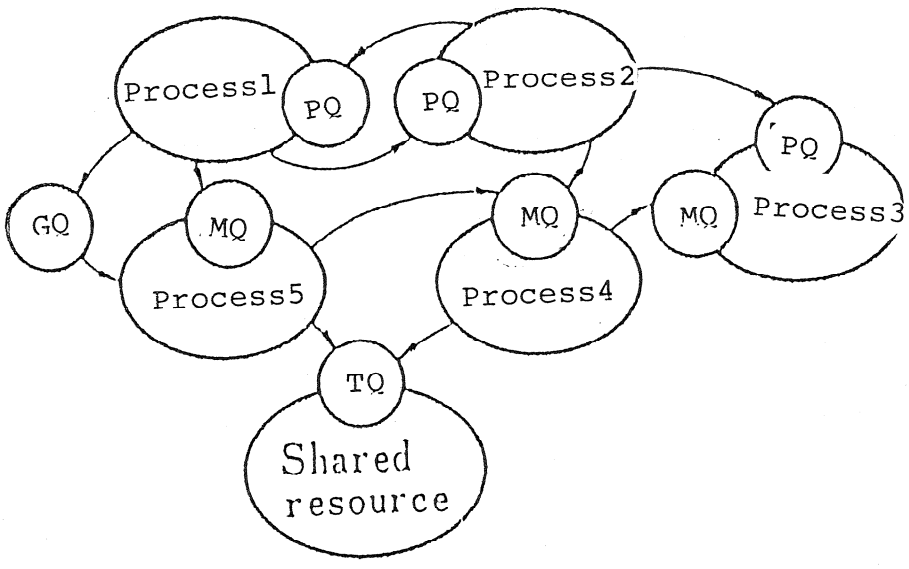


図 5．2　プロセス依存グラフ（意味のないプログラム）

表　5.1　待ち行列操作命令

Operation	Queue Type	Queue Name	Data to PCB	Data from PCB
Enter	PQ	PN,QN	PN*,PC(,M)	
	GQ	QN	PN*,PC(,M)	
Put	PQ	PN,QN	PN,PC(,M)	
	Ready	Queue	PN,PC(,M)	
	GQ	QN	PN,PC(,M) orM	
	MQ	PN,QN	M	
Get	PQ	PN*,QN		PN,PC(,M)orE
	MQ	PN*,QN		M or E
	GQ	QN		PN,PC(,M)or M or E
Test$Set Reset	TQ	QN	PN*,PC(,M)	Set or Reset
	TQ	QN		
Halt	QN1...QNm		PN*,PC	

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

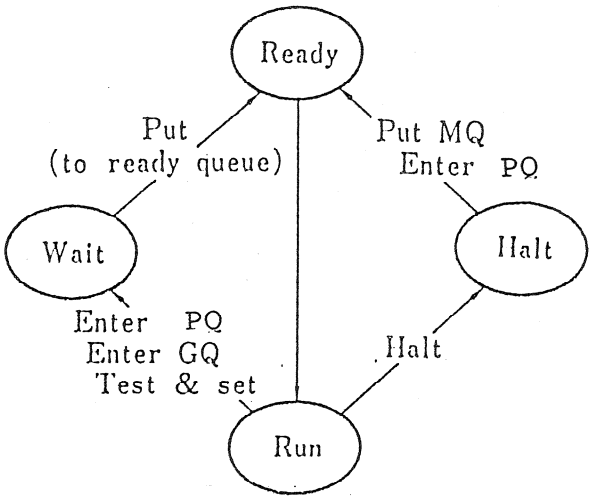


図 5．3 プロセスの状態遷移

では待ち行列が空のときＥＭＰＴＹが返される。一般にはＧＥＴ命令でプロセス名と要求を知り、必要な処理を行った後、ＰＵＴ命令（ｔｏ　ＲＥＡＤＹ　ＱＵＥＵＥ）で、ソースプロセスをＲＥＡＤＹ　状態にしＲｅａｄｙ　Ｑｕｅｕｅに送る。Ｒｅａｄｙ　ＱｕｅｕｅとはＲＥＡＤＹ状態のプロセスが入るシステムサポート待ち行列である。

ＰＱにはこの他、ＰＱへのプロセス到着で、ＰＱの属するプロセスをＷＡＫＥ　ＵＰする機能がある。デスティネーションプロセスがＨＡＬＴ命令を出すと、そのプロセスに属する指定された待ち行列が調べられ、何れの待ち行列にもプロセスが到着していない場合にプロセスはＨＡＬＴ状態になる。ＨＡＬＴ状態になったプロセスは、ＨＡＬＴ命令で指定した待ち行列にプロセスが到着した時ＷＡＫＥ　ＵＰされＲＥＡＤＹ状態になる。図５．４（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プロセスを待ち行列操作命令で記述したものである。

（２）Ｍｅｓｓａｇｅ　Ｑｕｅｕｅ（ＭＱ）：これはプロセス間でメッセージ転送を行うためのもので、ＰＱ同様あるプロセスに付属して、そのプロセスに制御される。ソースプロセスはＰＵＴ　ＭＱ命令でメッセージを送る。ＰＱとの違いは、ソースプロセスの状態が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プロセスは自分のＭＱの内容を取り出して処理することができ、ＰＱ同様ＭＱへのメッセージ到着によるＷＡＫＥ　ＵＰ機能を持つ。

（３）Ｔｅｓｔ＆Ｓｅｔ　Ｑｕｅｕｅ（ＴＱ）：この待ち行列は複数のプロセス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、一時に高々一つのプロセスしか共有資源にアクセス出来ないように制御する。Ｔｅｓｔ＆Ｓｅｔ　ＴＱ　命令によりＴＱに付属したＴｅｓｔ＆Ｓｅｔビットがプロセスに返され、同時にＴｅｓｔ＆Ｓｅｔビットはセットされる。Ｔｅｓｔ＆Ｓｅｔ　ＴＱ　命令でＴｅｓｔ＆Ｓｅｔビットがリセットされていた場合にはプロセス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ＴＱに入りＷＡＩＴ状態になる。共有資源の使用を終えたプロセスはＲＥＳＥＴ　ＴＱ命令を出すと、ＴＱにプロセスが入っている時は、先頭のプロセスが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５．４（ｂ）はＨｏａｒｅの条件付きクリティカルリージョンをＴＱを用いて実現した例である。

（４）Ｇｌｏｂａｌ　Ｑｕｅｕｅ（ＧＱ）：メッセージやプロセスを一時的にリンクしておく待ち行列である。プロセスはＥＮＴＥＲ　ＧＱ命令でこの待ち行列に入りＷＡＩＴ状態に

```
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) An 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 (TQ BUFFERING,PRODUCER.PC);
   DEPOSIT (M);
   COUNT:=COUNT+1;
   reset (TQ BUFFERING);
   goto A;

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

(b) An example of conditional critical region.

図 5．4　生産者・消費者プログラム

なる。ＰＱとの違いは、特定のプロセスに所属せず、ＷＡＫＥ　ＵＰ　機能がないことである。

待ち行列操作命令には、以上説明してきたように、プロセスを制御する機能が含まれており普通の機械語に比べてレベルが高く、待ち行列操作命令は高級言語計算機の仮想命令と考えられよう。図５．４は並列プログラム用高級言語の並列制御機構の代表であるメッセージ交換と、Ｍｏｎｉｔｏｒの原形であるＣｏｎｄｉｔｉｏｎａｌ　ｃｒｉｔｉｃａｌ　ｒｅｇｉｏｎを本モデルで実現する様子を示す例である。プログラムは生産者・消費者プログラムである。左の列がソースプログラムで、右の列が本モデルの待ち行列操作命令で実現したプログラムである。この図から待ち行列操作命令はソースプログラムの命令とほぼ一対一に対応が付くことが分かる。

５．３　並列計算機のアーキテクチャ

本節では、前節のモデルを効率良く実現すると共に、従来並列計算機が抱えていた共有データ操作の問題を解決するアーキテクチャを示す。

５．３．１　設計思想

主な設計思想は次のとおりである。

（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得るものとして、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

前節のモデルは、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、これを効率良く実現する必要がある。ここでは待ち行列をハードウェア化（ファームウェア化を含む）して高速化を図る。

（３）　プロセススケジューリング

並列計算機ではプロセスをプロセッサに割り当てるスケジューリングが性能に大きな影響を与える。ここではプロセス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プロセスが低負荷のプロセッサへ流れていく方式を選び、ハードウェアキューを用いてプロセッサとプロセスの高速バインディングを実現する。またＦｏｒ　ａｌｌ命令等による複数プロセス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。これによりプロセス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、プロセッサ数分だけのポートを持つ共有メモリで、メモリへの書き込みは一度に一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ある。マルチリードメモリは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方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良く実現するには、ハードウェアに可変性を持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易くしている。

５．３．２　並列アーキテクチャ

図５．５が以上の考えを実現する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、プロセス制御部（ＰＣＢ）、レディキュー（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バスと呼ばれる書き込み専用バスで、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。５．２節で示したキューイングネットワークは、ＰＣＢとＲＱで実現される。ＰＣＢはプロセスに付随した待ち行列と、待ち行列操作命令に伴うプロセスの状態変化を管理する部分で、待ち行列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。ＰＣＢでＲＥＡＤＹ状態になったプロセスはメッセージとしてＲＱへ送られる。ＰＥは隣接するＰＥとの間にパスを持つ。

５．３．２．１　待ち行列サポート機能

５．２節のモデルの待ち行列機能はＰＣＢとＲＱで実現される。図５．６はプロセス制御の機構を示したものである。プロセスの実体は共有メモリにあり全ＰＥからアクセスできる。各プロセスにはデスクリプタが付随し、これはＰＣＢに置かれる。デスクリプタはプロセスへのポインタ、プロセスに付属した待ち行列の状態を示すＥＭＰＴＹビットとＨＡＬＴの状態を示すＨＡＬＴビットから成る。待ち行列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。プロセスの状態がＲＥＡＤＹになるとプロセス名と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。これによりプロセスのプロセッサへのスケジュールはプロセスの実体は移動せず、ポインタのみの移動で済む。

ＰＣＢの仕事はＰＥからの待ち行列操作命令に応じて待ち行列の管理とプロセスの状態の管理を行うことであるが、各待ち行列に対する待ち行列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ＰＣＢをＷＲＩＴＥバスに接続し、ＰＥからの待ち行列操作命令により、ライト命令と同様にＷＲＩＴＥバスを専有し、他のライト命令に中断さ

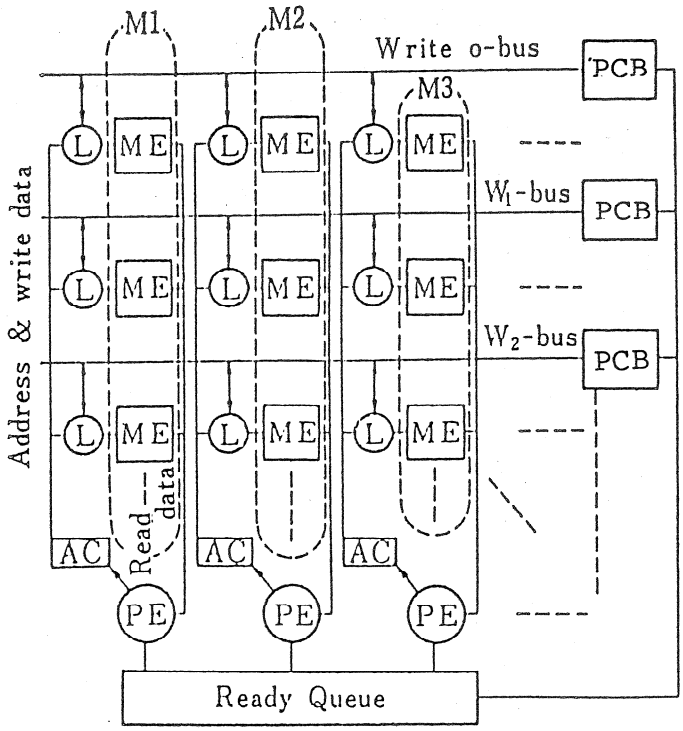


図 5．5　ハードウェア構成

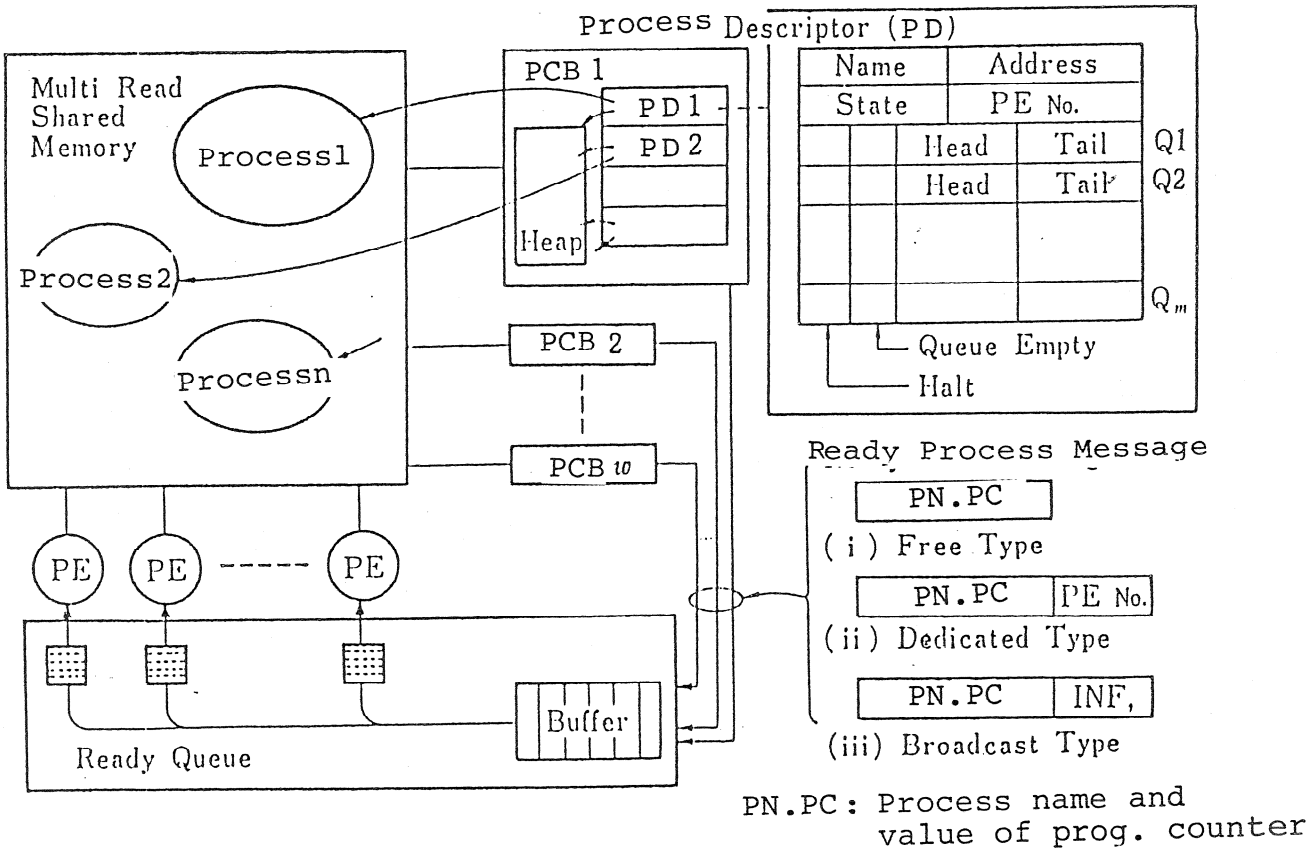


図 5．6　プロセス制御機構

れることなく一つの待ち行列操作命令を実行する。ＰＣＢ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ＰＣＢ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待ち行列管理の負荷は分散される。

ＲＱはＲＥＡＤＹ状態になったプロセスをできるだけＰＥの負荷が均等になるように、しかも高速にＰＥに割り当てる必要がある。ＲＱの構成は図５．６のとおりＰＣＢからのＲＥＡＤＹプロセスをバッファを介して各ＰＥのバッファに送り込む。ＲＥＡＤＹプロセスのメッセージには図５．６に示す３種類がある。デディケーテッド型はプロセスの実行されるＰＥが固定のものでメッセージの行き先は決まっている。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のハードウェアキュー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、終了したプロセスからのＰＵＴ　ＭＱ命令を受ける終了確認プロセス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の仕事が無くなったときや、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というシステムプロセスを起動することもできる。

５．３．２．２　可変構造共有メモリ

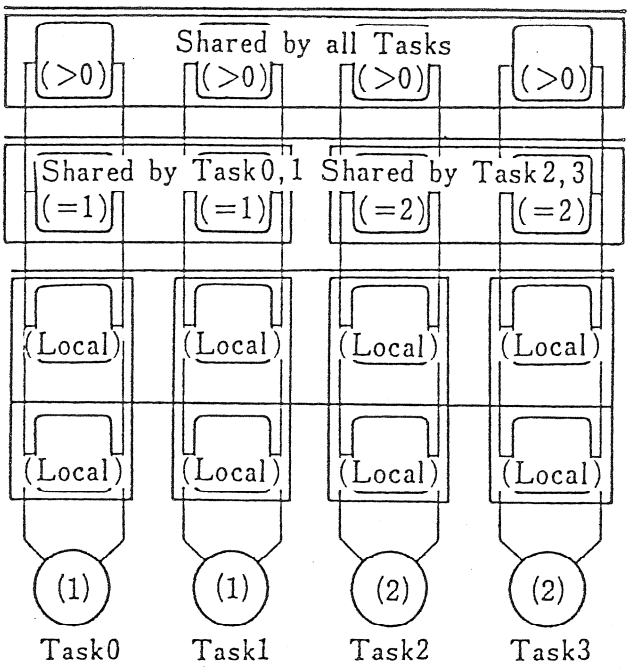
高級言語で書かれたプログラムを効率良く実行するには、プロセスやデータ構造を効率良く扱えることが重要であり、ここでは共有メモリに可変性を与えて対処する。ここで示す可変構造はマルチリードメモリでの衝突減少やメモリの有効利用にも役立つ。

可変性は図５．５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。

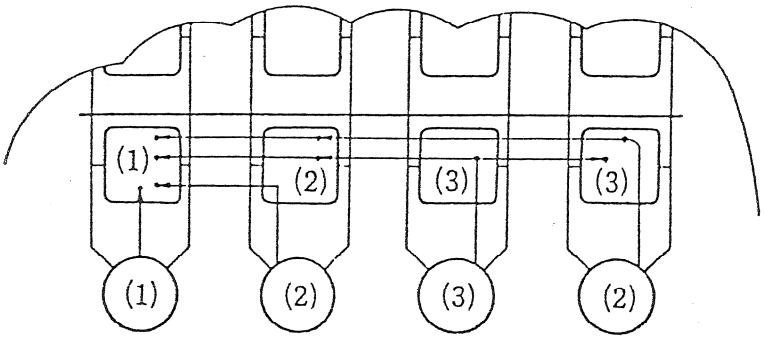
まず第一の可変性は、ＭＥのアドレス方式である。アドレス方式にはブロック型とスライス型がある。ブロック型は、アドレスをＭＥ内の隣り合った語に連続的に割り付けるものであるのに対し、スライス型は隣り合ったアドレスが隣りのＭＥになるようにインタリーブ方式でアドレス付けをする。このアドレス変換はＡＣにより行われる。

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、階層構造を容易に実現できる。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なり、これがＷＲＩＴＥ命令の持つ優先度とＬで比較され、ＭＥへの書き込みが制御される。ＭＥに書き込みを許さない優先度（図５．７（ａ）のＬＯＣＡＬ）を与えて置くと、そのＭＥはＷＲＩＴＥバスと切り離なされ、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５．７（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）はプロセス間の書き込み権を制御する例である。



(a) Partitioning



(b) Hierarchical structure

図 5．7　可変構造共有メモリ

５．３．３　プロセッシングエレメント（ＰＥ）

以上で示したアーキテクチャからＰＥに対する制約は出てこないため、どのようなものを用いてもよいが、ここでは筆者が４章の経験をもとにＰＵＬＣＥを用いて設計、制作したＰＥを示す。

４章の経験から学んだ事柄を整理すると、次のようになる。

（１）ＰＥのマイクロ命令のフィールドの利用率には片寄りが大きい。言い換れば、一部の命令や機能は頻繁に使われるが、ほとんど使われないものもある。

（２）シーケンサの内、メモリ制御部は頻繁に使われるため、ゲスト計算機の構成をハードウェア化すると性能が大幅に向上する。

（３）高級言語計算機とマイクロ命令の間にはさほど相関がない。

（４）マイクロプログラム中には１万ステップを越えるものもあり、大容量のマイクロプログラムメモリが必要である。

このような認識から、筆者等は、マイクロ命令をコード化した可変長マイクロ命令を導入してマイクロプコグラムメモリを節約するアーキテクチャを考案した。以下その概要を示す。

５．３．３．１　プロセッシングエレメントの構成

ＰＥの構成は図５．８に示すようにＣＰＵとローカルメモリからなり、ＣＰＵはＰＵＬＣＥ、シーケンサ、デコーダ、ミリメモリ、アドレス変換部よりなる。ＣＰＵではマイクロ命令を圧縮したミリ命令と呼ぶ命令を導入している。ミリ命令はマイクロ命令とほぼ一対一に対応し、８、１６、２４、３２ビット長のものがある。図５．８のミリメモリはミリ命令を格納するメモリである。ミリ命令はシーケンサからアドレスされて読み出され、デコーダでデコードされて３２ビットのマイクロ命令になりＰＵＬＣＥへ送られる。シーケンサの構造は、ミリ命令用プログラムカウンタのインクリメントを行う部分が可変長命令を扱えるようになっている他はＡＭＤ２９０９シーケンサとほぼ同じである。マイクロ命令のフェッチと実行はパイプライン式に行われマイクロ（ミリ）サブルーチンジャンプ用スタックも８語用意している。

このシステムにおいて、デコーダは重要な鍵である。ＰＵＬＣＥの１マイクロ命令実行時間は、２００ナノ秒であり、パイプライン方式ではこの間に、次に実行するべきミリ命令の

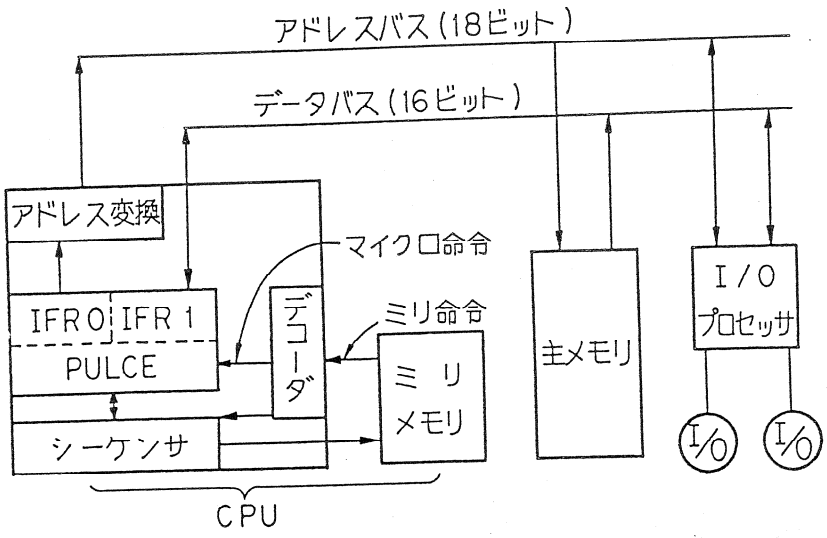


図5．8　プロセッシングエレメントの構造

読み出しとデコードを済ませねばならない。今回のインプリメントでは、ミリ命令の、アドレス時間が９０ナノ秒、ミリ命令のデコードは８０ナノ秒であった。アドレス変換機構は今回のインプリメントでは並列Ｐａｓｃａｌマシンの実現を想定し、ＰＤＰ１１の仮想アドレス方式を用いている。

５．３．３．２　並列Ｐａｓｃａｌマシン用ミリ命令

ミリ命令は応用に合わせて設計することで、よりフィールドの圧縮が可能となる。ここでは並列Ｐａｓｃａｌマシンを効率良く実現するミリ命令とその評価を示す。Ｂｒｉｎｃｈ Ｈａｎｓｅｎの仮想計算機はＷ，Ｘ，Ｙ，Ｇ，Ｓ，Ｂ，Ｑと呼ばれる７つのレジスタ（ＣＰＭレジスタ）を使い、スタックマシン構成を採っている。Ｗ，Ｘ，Ｙはワークレジスタ、Ｓはスタックポインタ、Ｑは仮想計算機のプログラムカウンタ、Ｇ、Ｂはベースレジスタである。ミリ命令の決定は、使用頻度の高いマイクロ命令をビット数の少ないミリ命令に割り当てることと、コード化が複雑でないことが重要である。ここでは４章の結果をもとにミリ命令を決定した。ミリ命令には８、１６、２４、３２ビット長の４タイプがあり、図５．９ではこのうち８、１６、２４ビットの命令を示している。この決定根拠は次のとうりである。

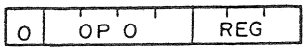
（１）ジャンプ、サブルーチンジャンプ、条件ジャンプにはアドレスフィールドが必要である。しかしこれらの命令の使用頻度は高いため、できるだけ短い命令が望まれる。そこで分岐先は自分の命令からのディスプレースメントで示し、しかも３２ビット単位のアドレス指定にし、１６ビット命令に納めた。

（２）ＰＵＬＣＥとメモリのデータ転送には、アドレスをＩＦＲ０に入れ、データはＩＦＲ１に読み書きされる。そのため、ＩＦＲ１とＣＰＭレジスタの演算が多くなる。そこでＩＦＲ１とＣＰＭレジスタ間の演算には８ビット命令を割り当てた。

（３）マイクロ命令では３２個のレジスタ指定が可能であるが、並列Ｐａｓｃａｌマシンでは１６のレジスタが主に使われるため、４ビットのレジスタフィールドを作り、また演算命令の使用頻度の片寄りを考慮して操作フィールドを圧縮し１６ビット命令を作った。

（４）エミット演算では４ビット以下のエミットの割合が高い。そこで４ビットのエミットとレジスタの演算を１６ビット命令にした。４ビットエミットのうち、半分以上がインクリメント、デクリメント、レジスタの値を２減らすであり、これらに対しては８ビット命令を割り当てた。

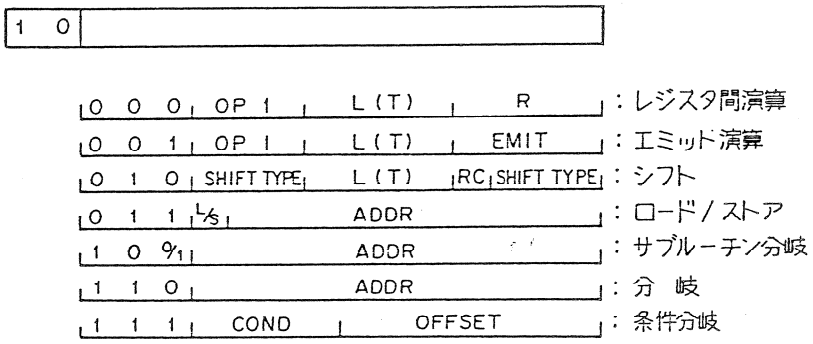
(i) 8ビット命令



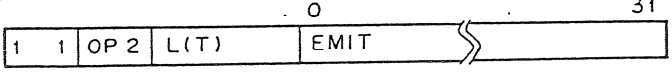
	レジスタ名 REG	操作 OP 0	SP 1	SP 2
0	W (CPMレジスタ(レジスタファイル内に割当る))	CLEAR Reg	0 → IFR 1	RTS
1	X	COMPARE Reg. IFR 1	IFR 1 + 1	NEXT
2	Y	Reg → IFR 1	IFR 1 − 1	JUMP IFR 1
3	Q	IFR 1 → Reg	Test IFR 1	JSR IFR 1
4	B	Reg + 1	Read Seg Reg.	K/U change
5	G	Reg + IFR 1 → Reg	Write Seg Reg	WAIT
6	S	IFR 1 + Reg → IFR 1		HALT
7	IFR 0	Reg − 1		
8		Reg − 2		
9		Reg − IFR 1 → Reg		
A		IFR 1 − Reg → IFR 1		
B		Read (addr, IFR 0 / data, IFR 1)		
C		Read INC		
D		WRITE		
E		SP 1		
F		SP 2		

(ii) 16ビット命令

	レジスタ名 L(T), R	操作 OP 1	条件 COND
0	W	OR	ALWAYS
1	X	MOVE	PLUS
2	Y	AND	MINUS
3	Q	XOR	$\overline{\text{OVERFLOW}}$
4	B	ADD	ZERO
5	G	SUB	$\overline{\text{CARRY}}$
6	S	ADC	CARRY
7	IFR 0	COMPARE	OVERFLOW
8	IFR 1		R 0
9	GPR 0 (ワークレジスタ)		$\overline{\text{PLUS}}$
A	GPR 1 (ワークレジスタ)		$\overline{\text{MINUS}}$
B	GPR 2 (ワークレジスタ)		$\overline{\text{ZERO}}$
C	F 12 (フローティング演算用(レジスタファイル内))		U0 (ステータスレジスタのユーザビット)
D	F 13 (フローティング演算用(レジスタファイル内))		U 1 (ステータスレジスタのユーザビット)
E	F 14 (フローティング演算用(レジスタファイル内))		U 2 (ステータスレジスタのユーザビット)
F	F 15 (フローティング演算用(レジスタファイル内))		U 3 (ステータスレジスタのユーザビット)



(iii) 24ビット命令



OP 2

0	OR
1	MOVE
2	AND
3	32ビット命令

図5．9　並列Ｐａｓｃａｌマシン用ミリ命令

（５）レジスタの内容をアドレスとしたメモリアクセスではアドレスをＩＦＲ０に送り、次に入出力命令を出す。そこでこの２ステップをＲＥＡＤとＷＲＩＴＥという８ビット命令で実現した。またＲＥＡＤＩＮＣという命令は入力命令と同時に、レジスタ内のアドレスに２を加えるものでスタックマシンの実現に使われる。

（６）レジスタクリア、比較、次に実行する仮想命令のフェッチ（ＮＥＸＴ）等は頻繁に使われるもので、可能なものは８ビット命令にした。

（８）１６ビットのエミットを使う命令は２４ビット命令になる。

（９）ＰＵＬＣＥのマイクロ命令の内ステップフィールドを無くし、ＭＳＢ側４ビットを“１”にして３２ビット命令にした。

５．３．３．３　ミリ命令の効果

ここでは、ミリ命令によるマイクロプログラム領域の減少効果と、ミリ命令を用いた主な仮想計算機命令の実行ステップ数を示す。

表５．２は、ミリ命令を用いて並列ＰａｓｃａｌマシンのインタプリタとＯＳカーネルをプログラムした時のメモリ量で、マイクロ命令を用いた時に比べ、６５％以上のメモリ節約になった。又表中には同プログラムをＰＤＰ１１/４５で実現した時のステップ数とメモリ領域を参考のために示した。これを比較すると、一つのミリ命令の記述力ではＰＤＰ１１/45に劣るが、８ビット命令の効果でメモリ領域はＰＤＰ１１/４５より少なくて済むことが分かる。

表５．２　　ミリ命令による並列 PASCAL マシン

（a） 仮想マシン命令のインタプリタ

	ミリ命令				PDP 11/45 マシン命令
	8ビット命令	16ビット命令	24ビット命令	32ビット命令	
ステップ数	582	361	21	7	685
バイト数	1395				1712

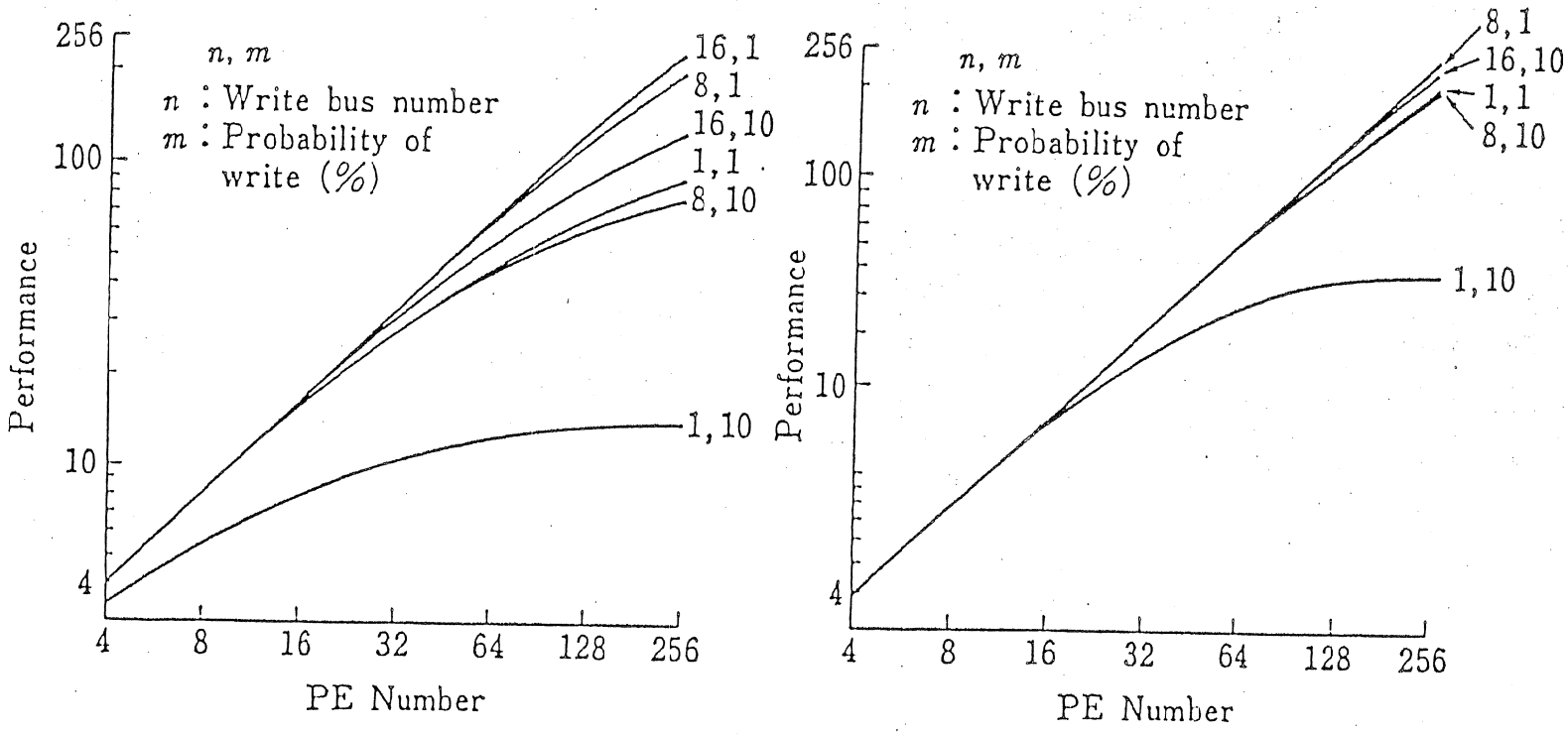
（b） OS のカーネル

	ミリ命令			PDP 11/45 マシン命令
	8ビット命令	16ビット命令	24ビット命令	
ステップ数	573	366	59	516
バイト数	1482			1720

５．４　アーキテクチャの評価

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。ここでは二つのシミュレーションを行い性能を評価した。第一のシミュレーションは、ＰＥ数とＷＲＩＴＥバスの本数に関するものである。シミュレーションモデルはハードウェア構成が図５．５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の稼働率の和を求めた。図５．１２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分かる。

第二のシミュレーションは図５．５のアーキテクチャで５．３節のＰＥを用いて３種類のプログラムを実行した時のシステムの性能である（図５．１１）。テストプログラムは（ｉ）１０２４データをシェルソートでソートする、(ii)１２８＊１２８のデータを８近傍値と自分自身の値を用いて空間フーリエ変換する、(iii)マトリクス乗算（１６＊１６、３２＊３２、１２８＊１２８）、の３種類である。この結果からこのアーキテクチャはマトリクス乗算等ライト命令の頻度が少ない応用で非常に良い効果を示すことが分かる。



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

図5．10 マルチリードメモリのシミュレーション結果

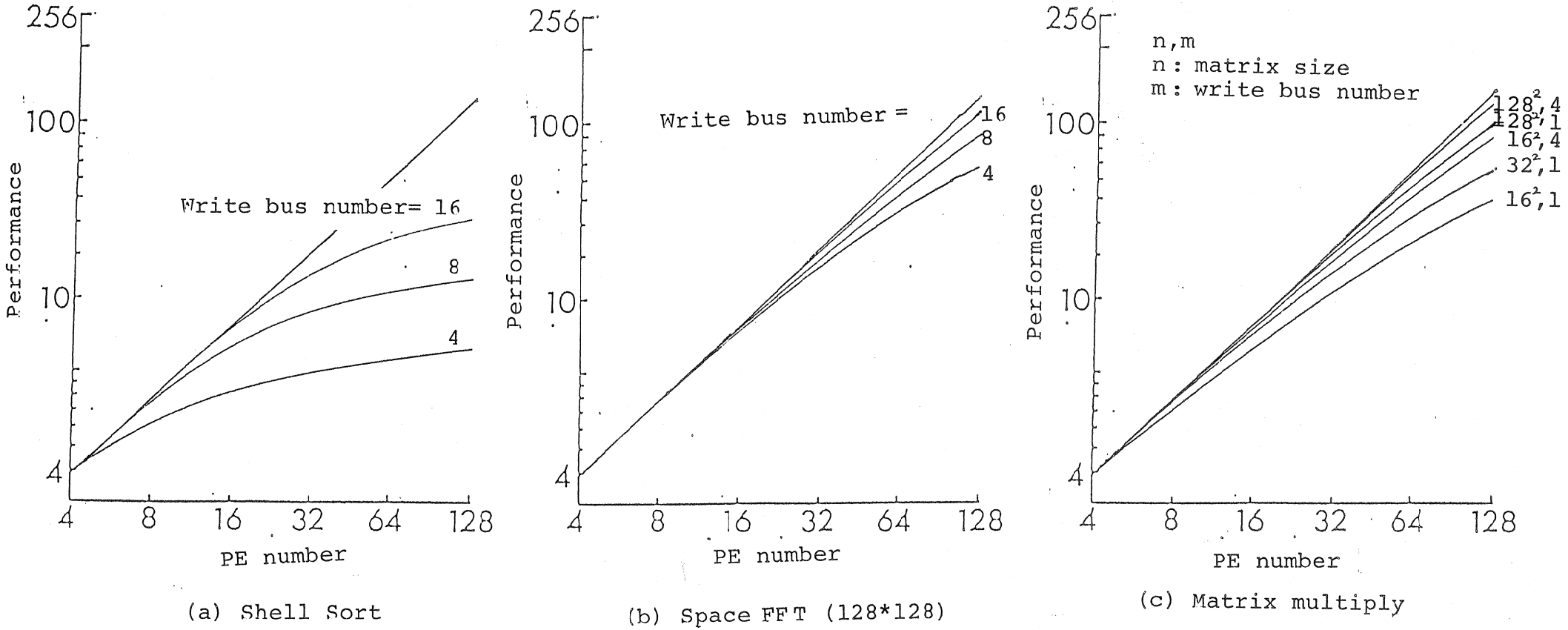


図５．１１　プログラム実行シミュレーション結果

５．５　まとめ

この 章 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共に、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。

本章で示したモデルは、並列プログラミング言語のための仮想計算機と考えられる。本モデルでは並列プロセス間の相互作用を待ち行列の形で整理した。待ち行列は相互作用の表現力のみならず、ハードウェア化が容易であることから適当な選択と思われる。本モデルは種々の高級言語をサポートするため４種の待ち行列を用意したが、言語を限ればより少ない待ち行列で済むであろうし、より高機能な待ち行列を用いることもできよう。

ここで示したアーキテクチャは、ＭＩＭＤを基本とし幅広い並列処理への応用を念頭に置いて設計した。又、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。今回試作したプロセッシングエレメントは、４章の結果に基づいて使用頻度の少ない命令や機能を減らし、使用頻度の高い命令や機能を充実させる方針を採った。これはその後盛んになったＲＩＳＣ（Ｒｅｄｕｃｅｄ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｓｅｔ　Ｃｏｍｐｕｔｅｒ）アプローチの先駆けとなったもので、本マシンではプログラム記憶容量が約６５％節約できた。

最後のシミュレーションでは、ＰＥ数とＷＲＩＴＥバス数の割合を示すと共に、マトリクス演算等が本システムで効率良く実行できることを示した。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。現在使用できるＴＴＬ集積回路を用いて２０メガビット／秒の速度を実現する場合、理論上百数十台のＰＥが接続可能であり、実装の余裕をみても百台程度で高速動作が可能であろう。

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。

第6章　結論

6．1　研究結果の概要

本研究は、汎用並列計算機の諸問題を解決し、実用化への道を開くべく行った一連の研究成果をまとめたものである。本研究の成果を要約すると、汎用並列計算機の成功を阻んでいるソフトウェア及びソフトウェアとハードウェアの間のセマンティックギャップに焦点を当て、これを解決する方法として並列計算機を並列プログラミング言語用高級言語計算機として設計することを提案しその実現法を示したことといえよう。それは並列計算機上で走るプログラムを高級言語で書ける環境をプログラマに与えると共にその並列プログラムを効率良く実行する並列計算機アーキテクチャの実現法を示すものである。

並列計算機を並列プログラミング言語用高級言語計算機として設計する方法を確立するために行った主な研究開発成果は次のとおりである。

（1）並列計算機構築をサポートする評価システム

並列計算機構築に当っては予め解析やシミュレーションでアーキテクチャの妥当性を検証しておく必要がある。本研究では次の二つの評価システムを開発した。

（i）バス結合並列計算機の解析システム：バス結合並列計算機を実システムに近いレベルで解析できるモデルと解析法を示した。これは被評価システムの内部の負荷状態を求めることができる。本論文中にはこのシステムを用いた二、三の評価結果を示した。それらはバス結合並列計算機の設計に重要な資料である。

（ii）並列計算機シミュレーションシステム：種々の並列計算機のシミュレータを容易に構成でき、性能評価を行うことのできるシミュレーションシステムである。ここでは、階層化とモジュール化を用いて並列計算機のシミュレータを構成していく方法を開発した。なお、この成果は東芝に技術移行し工業技術院大型プロジェクトパターン情報処理システムにおいて並列計算機システムの開発に重要な役割を果した。

（2）並列計算機上の並列Ｐａｓｃａｌマシン

これは並列プログラミング言語用高級言語計算機を並列計算機上に実現する世界最初のケーススタディである。この実現に当っては二つの問題の解決が必要であった。第一はマイクロプログラムによるインタプリタの実現である。並列プログラミング言語用高級言語計算機をマイクロプログラムで実現する場合、従来の方法ではマイクロプログラムの量が膨大になる。そこでここでは新しい仮想計算機を設定しそのマイクロプログラムを作る方法でこの問

題を解決した。第二は並列プロセスをサポートする機構の開発で、これは従来の逐次処理型言語の高級言語計算機では必要なかったものである。ここでは、モニタを効率良くサポートする専用ハードウェアとプロセススケジューラを開発した。

又、このシステムの性能を評価した結果、並列プログラムを並列計算機で実行することの合理性が明らかになると同時に、オーバヘッドも少なくないことが分かった。このことが後述（4）開発の動機となった。

（3）マイクロプログラムの解析

マイクロプログラムと高級言語計算機の関係を明らかにすることは、マイクロプログラマブルプロセッサの設計に重要である。本論文では約7万ステップのマイクロプログラムを分析し高級言語計算機との関係を明らかにするとともに、マイクロプログラマブルプロセッサＰＵＬＣＥの評価を示した。これらは従来求められていないデータでアーキテクトにとって大変興味深いものである。

（4）並列構造記述モデルと並列アーキテクチャ

並列高級言語計算機の開発には高級言語の持つ並列制御構造をハードウェア化していく必要がある。ここでは並列制御構造をハードウェアで実現し易い簡潔な形に表現できるモデルを開発し、それを実現するアーキテクチャを示した。並列計算機を並列プログラミング言語用高級言語計算機として設計するという考え方の重要性は最近認識され、同様の思想に基づくＴｒｓｎｓｐｕｔｅｒ[Bar83]が提案されている。この分野の研究は今後ますます盛んになると思われる。

5章で示した並列アーキテクチャは、並列計算機の共有データの扱いを容易にするもので今後広く用いられると思われる。又、そこで試作したプロセッシングエレメントは（3）の解析をもとに設計したもので、ＲＩＳＣ（Ｒｅｄｕｃｅｄ　Ｉｎｓｔｒｕｃｔｉｏｎ　Ｓｅｔ　Ｃｏｍｐｕｔｅｒ）アプローチの先駆けとなったものである。

６．２　研究の経過

本研究は、筆者が電子技術総合研究所において行った計算機アーキテクチャの研究成果をまとめたものである。筆者は昭和４８～５３年に渡り工業技術院大型プロジェクトパターン情報処理システムの中で元電子技術総合研究所計算機方式研究室（現成蹊大学教授）飯塚肇博士が中心になって行った並列計算機（ＡＣＥ）とマイクロプログラマブルプロセッサ（ＰＵＬＣＥ）の開発に参加してきており、筆者の研究はそれらと切り離して見ることはでき

ない。第３章の評価システムは同プロジェクトの並列計算機開発のために開発した。その後１９７３年に世界最初の並列プログラム用高級言語（並列Ｐａｓｃａｌ）が使用できるようになると、並列プログラムを並列計算機で並列処理することは並列プログラムの効率良い実行方法であるばかりでなく並列計算機の問題点であるソフトウェアの問題を高級言語で解決することでもあると考えＡＣＥシステム上に並列Ｐａｓｃａｌマシンを実現した。又この研究にはマイクロプログラムによる高級言語計算機実現の意味も含まれていた。これにより、並列高級言語を並列計算機上で実現する方法は示すことができたが、この開発はＡＣＥシステムと並列Ｐａｓｃａｌという組み合せからボトムアップに行われたものであった。筆者はこれをもっと一般性のある形に整理し、どんな並列高級言語と並列計算機にも簡単に適用できる方法を確立したいと考え並列プログラムの並列制御構造を表現するモデルを考案しそのハードウェアによる実現法を示した。

この間ＰＵＬＣＥのマイクロプログラムを分析し、マイクロ命令の静的使用特性を測定しマイクロプログラムと高級言語計算機の関係を明らかにした。

６．３　おわりに

並列計算機は多種多様な側面を持っており、現在のところこれら全ての面を満足させる並列処理の方法は見つかっていない。本研究ではマイクロプロセッサレベルの計算機を多数使って汎用並列計算機を構成する一つの方法を確立した。今後このような努力が並列計算機の様々な領域に対して行われていけば、やがて並列計算機の全ての側面が利用可能になるであろうし、科学技術の進歩によく見られるように、このような多くの提案が積み重ねられていく内に、時として新しい道が開けることがある。並列計算機に関しても今までなかった新しい概念が発明され全ての問題が解決される可能性もないとはいえない。本研究がそのような並列処理技術完成への一助となれば幸いである。

謝辞

本研究遂行に当っては多くの方々の直接あるいは間接の御指導、御支援を戴いた。

元電子技術総合研究所計算機方式研究室長（現成蹊大学教授）飯塚　肇博士は筆者を計算機アーキテクチャの分野へ導かれ、入所以来永年に渡って貴重な御指導、御意見を与えられ本研究の遂行に多大な御協力を戴いた。又、同博士には本論文をまとめるに当っても適切な御指導を戴いた。成蹊大学教授和田　弘博士には大学院時代から御指導戴き、卒業後も並列プログラムの御教示を戴いた他終始研究姿勢について貴重な御助言を戴いた。元電子技術総合研究所計算機方式研究室長（現明電舎）藤井捐介氏には永年に渡り多くの御指導、御援助を戴いた。成蹊大学教授高木正英博士、同大学教授中西俊男博士には本論文をまとめるに当って貴重な御助言を戴いた。電子技術総合研究所電子計算機部長柏木　寛博士、同計算機方式研究室長弓場敏嗣博士、元ソフトウェア部長（現日本工業大学教授）石井　治博士、元電子計算機部長（現東京理科大学教授）黒川一夫博士、元パターン情報部長（現筑波大学教授）西野博二博士は本研究の機会を与えられ、終始暖かい御支援を下さった。これらの方々に心から感謝致します。

又、本論文中の各章の研究には次の方々の御援助を受けた。第4章のマイクロプログラムの評価では所内外の多くの方々からマイクロプログラムの御提供を戴いた。又、同章のＰＵＬＣＥアーキテクチャの評価には設計者である成蹊大学教授飯塚　肇博士に御参加戴いた。第5章の並列アーキテクチャの検討では内堀義信技官、西田健次技官の御協力を得た。同章のプロセッシングエレメントの試作では元成蹊大学（現明治生命）猪俣正男氏の御協力を得た。又、計算機方式研究室の皆様には本研究全般に渡り御議論、御検討戴いた。これらの方々に深く感謝の意を表します。

そして最後に永い間筆者を御指導、御支援下さった先生方、家族、電子技術総合研究所の方々の御協力に厚く感謝いたします。

参考文献

Ahj82 Ahuja,S.R.,and Asthana,A. "A Multi-Micro Processor Architecture with Hardware Support for Communication and Scheduling," Sigplan Notice, Vol.17, No.4, pp205-209,(Apr. 1982).

Ale75 Alexander,W.G. and Wortman,D.B."Static and Dynamic Characteristis of XPL Program,"IEEE Computer,(Nov.1975).

And61 Anderson,J.P."A Computer for Direct Execution of Algolithmic Languages," AFIPS Proc.,Vol.20,pp184-193,(1961).

Bar61 Barton,R.S."A New Approach to the Functional Design of a Digital Computer," Proc. WJCC (1961).

Bar83 Barron,I.,et al."Transputer Does 5 or More MIPS Even When Not Used in Parallel," Electronics,Nov.17,pp109-115,(1983).

Bas72 Baskin,H.L.,Borges,B.R.and Roberts,R."PRIME-A Modular Architecture for Terminal-Oriented Systems,"Proc.AFIPS Spring Jt.Computer Conf.,pp.431-437 ,(1972).

Bel70 Bell,C.G. and A. Newell. "The PMS and ISP Descriptive Systems for Computer Structures," SJCC, Vol. 36, (1970).

Bri75 Brinch Hansen,P."The Programming Language Concurrent Pascal,"IEEE Trans. Software Engineering,Vol.SE-1,No.2,pp199-207,(June 1975).

Bri77 Brinch Hansen,P."The Architecture of Concurrent Programs,"Prentice-Hall, (1977)

Bri78 Brinch Hansen,P."Multiprocessor Architectures for Concurrent Programs," Sig.Arch., Vol.7,No.4,(Dec. 1978)

Cha72 Chandy,K.M. "The Analysis and Solutions for General Queuing Networks, Proc. Sixth Annual Princeton Conference on Information Sciences and Systems," pp. 224-228,(March. 1972).

Dij68 Dijkstra,E.W. "The Structure of The Multiprogramming System," CACM 11,5, (April 1968).

Dij72 Dijkstra,E.W et al. "Structured Programming," Academic Press.(1972).

Dij75 Dijkstra,E.W."Guarded Commands,Nondeterminancy and Formal Derivation of Program,"CACM,Vol.18,No.8,pp.453-457,(Aug. 1975).

Dod79 Dod. "Rationale for the Design of the ADA Programming Language," Sigplan Notice,Vol.14(Jun.1979).

Fur76 古谷."バス結合マルチプロセッサシステムの解析モデルと解析,"情報処理,Vol17 ,No.5,pp394-401,(May 1976).

Fur77a 古谷."ポリプロセッサ・シミュレーション・システムーＰＰＳＳ,"情報処理, Vol.18,No.6,pp534-541,(June 1977).

Fur77b 古谷,飯塚,大表."マルチプロセッサシステム用相互排斥モジュールの一設計," 情報処理,Vol.18,No.6,pp609-611,(June 1977).

Fur80a 古谷."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 .Vol.21,No.2,pp125-132,(Mar 1980).

Fur80b 古谷."高級言語による並列処理の記述,"情報処理,Vol.21,No.9,pp949-958,(1980)。
Fur81 古谷."マイクロプロセッサ(ＰＵＬＣＥ)を用いた Concurrent Pascal マシン," 電子技術総合研究所彙報,Vol.45,No.7,8.(July 1981)
Fur82 古谷,飯塚."マイクロプロセッサ(ＰＵＬＣＥ)におけるマイクロ命令の静的使用特性," 電子通信学会論文誌,Vol.J65-D,No.2,pp258-265,(Feb. 1982)。
Fur83 古谷,内堀,西田."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," 情報処理,Vol.24,No.3,pp311-317,(May 1983)
Fur84 Furuya,T.,et al."A Parallel High Level Language Computer,"The First International Conf.on Computers and Applications,(June 1984)。
Han73 汎用マイクロプロセッサ研究会."マイクロプロセッサ-設計のための予備的検討," パターン情報処理システム調査研究報告, 電総研(Oct. 1973)。
Hoa74 Hoare,C.A.R."Monitors:An Operating System Structuring Concept,"CACM Vol.17,No.10,pp.549-557,(Oct.1974)。
Hoa78 Hoare,C.A.R."Communicating Sequential Process,"CACM,Vol.21,No.8,pp666-677,(Aug. 1978)
Hob70 Hobbs,L.C."Parallel Processor Systems,Technologies and Applications," Spartan Books,(1970)。
Hol78 Holt,R.C.,et al."Structured Concurrent Programming with Operating System Application,"Addison-Wesley(1978)。
Iiz75 Iizuka,H.,et al.,"ACE-A New Modular Computer Architecture,"Proc.US-J Comp.Con.2 ,pp36-41(1975)。
Iiz76a 飯塚,古谷."マイクロプロセッサアーキテクチャの一設計,"電子通信学会論文誌, Vol.59D-3,pp188-195,(Mar. 1976)。
Iiz76b 飯塚."適応構造計算機に関する研究"電子技術総合研究所研究報告,No.767, (Nov. 1976)。
Iiz79a 飯塚,藤井,古谷."モジュール型複合計算機(ＡＣＥ)の試み,"情報処理,Vol.20, No.4,pp314-318,(April 1979)。
Iiz79b Iizuka,H.,et al."Development of a High Performance Universal Computing Element-PULCE,"AFIPS Conf.Proc.,Vol.47,1970 NCC,pp1255-1264,(1979)。
Jon77 Jones,A.K.,et al.,"Software Management of Cm*-A Distributed Multi-processor,"AFIPS Conf. Proc.,Vol.46,NCC,pp.657-663,(1977)。
Knu73 Knudsen,M.J. "PMSL, An Interactive Language for System-level Description and Analysis of Computer Structures," Carnegie-Mellon Univ. (1973)。
Kog80 工業技術院編,"工業技術院大型プロジェクトパターン情報処理システム研究開発成果発表会論文集",日本産業技術振興協会,(Oct. 1980)。
Lev73 Levy.J.V, "Computing with Multiple Microprocessors," Ph.D. Dissertation, Stanford Univ. (1973)。
Lew79 Lewis,G.R.and Henry,J.S."The BTI8000-Homogeneous,General-Purpose Multi-processing,Proc.AFIPS NCC,pp513-528,(1979)。
Mae79 Maekawa, et al. "Experimental polyprocessor system(EPOS)-Architecture," Proc. Int. Symp.Computer Architecture, pp188-195,(1979)。

Nak80 Nakagawa,T.et al."A Multi-Processor Approach to Discrete System Simulation,"Comp.con.(Spring 1980).

Ohm74 Ohmori,K.,et al."MICS-a Multi-Microprocessor," Proc.IFIP Cong.,pp.98-102 ,(1974).

Par72 Parnas.D.L. "A Technique for Software Module Specification with Examples," CACM, 15,5, (May 1972).

Pat81 Patterson,D.A. and Sequin,C.H."RISC I:A Reduced Instruction Set VLSI Computer,"Proc.Eighth International Symp. on Computer Architecture, pp443-457,(MAY 1981).

Rav73 Ravidran,V.K.,and Thampy,T. "Characterization of Multiple Microprocessor Networks," Digest 1973 IEEE Computer Society International Conference,pp133-137,(1973).

Rey74 Reyling,George,Jr. "Performance and Control of Multiple Microprocessor Systems," Computer Design, pp.81-86,(March,1974).

Ros71 Rosin,R.F.et al.."An Environment for Research in Microprogramming and Emulation,"CACM,Vol15,No.8,pp748-760,(Aug. 1972).

Rud71 Rudin,Harry,Jr. "Performance of Simple Multiplexer Concentrators for Data Communication," IEEE Trans. Communication. Vol.COM-19, No.2, PP178-187, (April 1971).

Sas74 Sastry,K.V. et al. "Markovian Models for Performance Analysis of Multiprocessor System," Minesota Univ. (January, 1974).

Smi72 Smith,W.R.,et al."A Large Experimental System Exploring Major Hardware Replacement of Software,"Proc.SJCC,pp601-616,(1972).

Swa77 Swan,R.J., Fuller,S.H. and Siewiorek,D.P."Cm*-A modular multimicroprocessor,"Proc.AFIPS NCC,pp637-644,(1977).

Tak76 高橋, 藤田. "ポリプロセッサのソフトウェアシミュレーションＩＰＰＳＳ," 第１８回情報処理学会大会 (1976).

Tak82 高橋."並列処理のためのプロセッサ結合方式,"情報処理,Vo.23,No.3,pp.201-209, ,(Mar.1982).

Wal66 Wallance,V.L. et al. "Markovian Models and Numerical Analysis of Computer System Behavior," Proc. AFIPS SJCC, Vol. 28, pp141-148,(1966).

Web67 Weber,H."A Microprogrammed Implementation of EULER on IBM360/30," CACM,Vol10,No.9,pp549-558,(Sept. 1967).

Wid76 Widdoes Jr,L.C."The Minerva Multiprocessor,"3rd Annual Symposium on Computer Architecture, (1976)

Wig63 Wigington,R.L."A Machine Organization for a General Purpose List Processor,"IEEE Tran.EC-12,No.12,pp707-714,(Dec. 1963).

Wir77 Wirth,N."Modula:A Language for Modular Multiprogramming,"Software-Practice and Experience,Vol.7,No.1,pp3-35.(Jan.,Feb. 1977).

Wul72 Wulf,W.A.and Bell,C.G."C.mmp A Multi-Miniprocessor,"1972 AFIPS Fall Jt. Computer Conf,pp765-777,(Dec.1972).

Wul75 Wulf, W.A, and Roy Levin, "The Hydra Operating System," Carnegie-Mellon Univ.(1975).