マルチプロセッサシステムにおける Concurrent Pascal マシン†

古谷 立美††

コンカレントパスカル (CP) プログラムをマルチプロセッサシステム上で実行することが出来れば，それはCPの効果良い実現法であると共に，この言語がマルチプロセッサ用高級言語として使えることをも意味する．このような観点からマルチプロセッサ用 CP マシンを開発し，当研究所で開発したマルチプロセッサシステム (ACE) の上に実現した．

この論文では，先ずマイクロプログラムによる CP 命令の実現について，ACE システムで行った2つの実現法を示すと共に，それらの評価を行う．次にマルチプロセッサ化のための設計思想と機構を示す．最後にマルチプロセッサによる CP マシンの性能をできるだけ一般性のある形で示すためのシミュレーション結果と，ケーススタディとして ACE の場合の実測結果を示した．

1. はじめに

近年半導体集積技術はめざましく進歩し，豊富な機能を持った CPU や大容量メモリが安価に入手できるようになってきた．これに伴い LSI を多数使ってシステムを構成するという考え方が実現可能となり，いくつものマルチプロセッサシステム (MPS) が作られてきた．そしてこのようなアプローチは専用システムの分野で良い結果を示し一設計法としての地位を確立した．しかし汎用 MPS の分野では，プロセッサを何台もつなげば何か新しいものが出るのではないかという希望的観測のもとにハードウェア先行でシステムが作られ，その結果ハードウェアはできたもののそれを十分使いこなしていない場合が少なくない．MPS の能力を十分に引出せない原因の一つにソフトウェアの問題がある．MPS 用プログラミング言語としては，相互作用のあるプロセスを矛盾なく記述できる高級言語が望まれる．一方プログラミング言語に目を向けると，P. B. Hansen は OS 記述用言語として Concurrent Pascal (CP) を作成した．OS は本来相互作用のある並列プロセスからできている．CP はそれらの並列プロセスを矛盾なく記述し，並列プログラムの誤りをできるだけコンパイル時に検出しようという思想に基づいて，N. Wirth の Pascal にプロセス，モニタ，クラスという概念を加えてできている．この言語は並列プログラムの記述という点では満足できるが，1台のマシンにこれをインプリメントしようとするとプロセス切換のオーバヘッドが多く効率が悪い．そこで MPS 上で CP プログラムを走らせることができれば，上述のオーバヘッドを減少できると共に CP が MPS 用高級言語として使えることにもなる．CP を実際のマシンに実現するに当り Hansen は，1台のマシン (PDP 11/45) によってシミュレートされる仮想マシン (Concurrent Pascal Machine) を設定している．そこで筆者は MPS 用仮想マシンを設計し，当研究所で開発した ACE と呼ばれる適応構造型 MPS 上にこれを実現した．ただし仮想マシンの命令セットは Hansen のものと同一である．

以下この論文では，マイクロプログラミングによる仮想命令の実現，仮想マシンを MPS で実現するための機構，及びそれらの性能評価を示す．

2. ACE システムと Concurrent Pascal マシン

この章では ACE システムと Hansen の Concurrent Pascal マシン (CPM) の概要を述べる．

2.1 ACE システム[1),2)]

ACE システムは，処理しようとする問題や応用環境に適応させて計算機システムを構成しようという試みである．ACE システムは現在**図1**のように構成され，3台のプロセッサモジュール，2台のメモリモジュール，同期モジュール，I/O プロセッサが2本のCバスに結合されている．プロセッサモジュールの中核にはエミュレーション向マイクロプログラムプロセッ

† A Concurrent Pascal Machine on a Multiprocessor System by TATSUMI FURUYA (Computer Division, Electrotechnical Laboratory).

†† 電子技術総合研究所電子計算機部

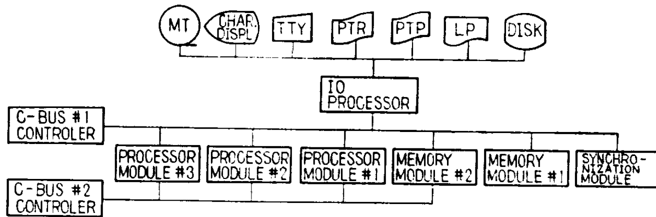
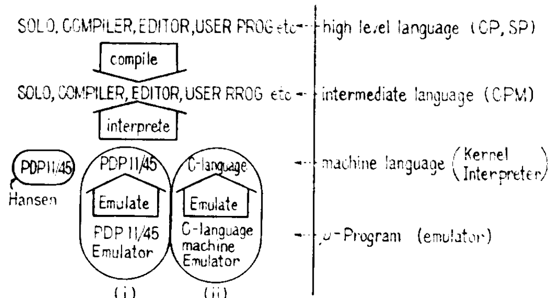


図 1 ACE システム
Fig. 1 ACE system.

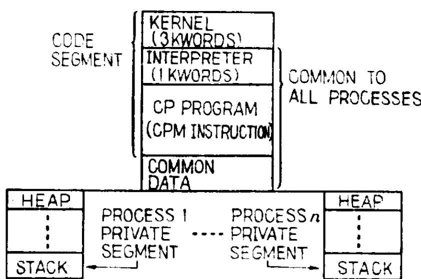
サ (PULCE[3] アーキテクチャ) があり，周辺にはダイナミックマイクロプログラムを可能にするマイクロキャッシュを有する．同期モジュール[4]はプロセッサ間の同期をとるためのもので P•V オペレーションが容易に実現できる．モジュール間通信には汎用性のあるブロードカスト方式を採用している．

2.2 Concurrent Pascal マシン[5]

Hansen は CP を用いて SOLO という OS を作成し配布した．SOLO の下では 1 人のユーザが Concurrent または Sequential Pascal (SP) プログラムをコンパイルしたりエディットしたり実行したりできる．そしてこのシステムの配布に当っては簡単にどの計算機にも実現できるような方式をとっている．ここでその実現方式と，ACE 上でどう実現したかを紹介する．図 2(a)が実現法を示している．SOLO，コンパイラ，エディタ等システムプログラムは元来 CP や SP といった高級言語で書かれたものであるが，Hansenから配布されるものはこれらをCP用に設定した仮想マシン (CPM) の命令で書いたものである．言い換えれば，CP と SP で書かれたプログラムをコンパイルして中間言語形式になったものが配布される．またユーザの作った CP や SP のプログラムもコンパイルされるとこの中間言語の命令からなるプログラムになるため，このシステムを使いたい人は，仮想マシンのシミュレータを実現すればよい．具体的には，CPM の中間言語のインタプリタ(I)と並列プロセスを1台のプロセッサに写像するカーネル(K)を自分の計算機に作ればよい．なお Hansen は CPM を PDP 11/45 に実現したため，その資料は利用できる．筆者は K, I を ACE システムに実現し，SOLO システムや SP または CP で書いた実験プログラムを走らせた．ACE システムの CPU (PULCE アーキテクチャ) はマイクロプログラム計算機であり機械語を持たない．マイクロプログラムで K, I を実現する方法はいろいろ考えられるが今回のインプリメントでは以下の方法で実現している．



(a) CPM execution process.



(b) Concurrent Pascal Machine store allocation.

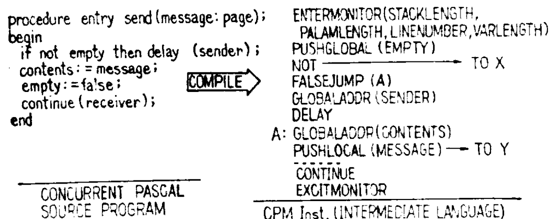
図 2 CP マシンのストアアロケーション
Fig. 2 Concurrent Pascal Machine.

(i) K, I は Hansen が PDP 11/45 で書いたものを使い，PULCE のマイクロプログラムで PDP 11/45 をエミュレートする (図 2(a)の(i))．

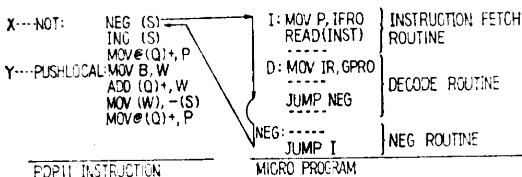
(ii) K, I は Hansen のアルゴリズムであるが，これをC-Languageと呼ぶ新たに設計した言語で記述し，マイクロプログラムでこの言語の命令を機械語とする仮想マシンをエミュレートする(図 2(a)(ii))．

(iii) 上の(i)，(ii)のKは Hansen のものであるため，1台のプロセッサ用である．そこでKをマルチプロセッサ用に新たに設計，製作し，並列プロセスを2台のプロセッサで実行できるシステムを作った．なおここでのIは(ii)と同じように実現している．

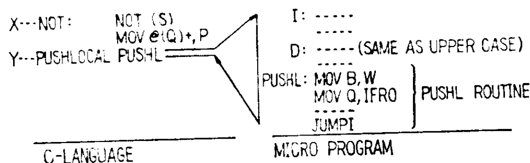
以下第3章では(i)，(ii)のマイクロプログラムによる K, I の実現と評価を，第4章では(iii)におけるマルチプロセッサ用 K の設計方針とマルチプロセッサ化における性能評価を中心に述べる．図 2(b)は PDP 11/45 用の CPM のストアアロケーションである．ユーザの書いた CP プログラムはコンパイルされ図中の CP PROGRAM というところに入り，これは各プロセス用ルーチンとそれらが共通に使うモニタルーチンから成る．各プロセスはスタックを持ち，これらは各プロセスにローカルなアドレス空間をつくる．



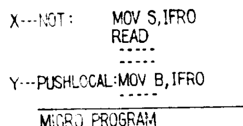
(a) Compiler generates V-code from source program.



(b) PDP11 emulation.



(c) C-language.



(d) Direct execution.

図 3 CP プログラムの実行プロセス
Fig. 3 Concurrent Pascal program execution process.

3. マイクロプログラムによるインタプリタの実現

ACE システムに CPM を実現するには，カーネル(K)とインタプリタ(I)を作成すればよい．この章では，マイクロプログラムによる CPM の実現とその評価を示す．

3.1 CPM の実現法

図 3 は CP 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 CPM 命令は，実際のマシンによって解釈実行される．そしてモニタなどプロセスのスケジュールが必要な場合に K が呼ばれる．

マイクロプログラムで動くプロセッサで CPM 命令を解釈する方法には次のようなものがある．

(1) PDP 11/45 のエミュレータを作る (図 3(b))：Hansen の K, I をそのまま利用できるが効率が悪い．

(2) K, I をすべてマイクロプログラムで書く (図 3(d))：効率は良いがマイクロプログラム量が膨大になる．

(3) CPM 向き用語 (CL) を設定する (図 3(c))：(1), (2) の中間的なもので I と K の特性を考慮に入れて，これらを効率良く実現する言語 (CL) を設定し，その命令のエミュレータを作る．この方法は (1) よりも効率が良く，マイクロプログラム量は (2) より少ない．

CPM の実現法としてはこれ以外にも考えられるが，今回は (1) と (3) の方法によって実現した．

3.2 PDP 11/45 のエミュレータ

ACE のプロセッサは 16 ビットのデータ処理幅を持つ 32 語のレジスタファイルを持つため，PDP 11/45 のレジスタ（フローティングプロセッサ用は除く）はすべてプロセッサ内に割当てることができる．PDP 11/45 の仮想アドレッシング方式は ACE のものと多少異なり，この部分で余計な処理が要求された．

3.3 CPM 向き言語 (C-Language)

表 1 は，命令形式表で，大きく 2 つの型に分かれる．第 1 の型 (No. 0〜10) は従来の機械語に相当するレベルであるが PDP 11/45 に比べて単純である．これは K と I の記述に PDP 11 のような論理の深い命令を必要としないことと，第 2 の型が多くの機能を吸収したためである．また第 1 の型の命令型式は ACE のマイクロ命令でエミュレートしやすいように決定している．第 2 の型 (No. 11〜13) は頻繁に使われる CPM 命令をファームウェア化し 1 つの命令を割当てるもので，第 1 の型に比べレベルが高く，マイクロプログラムの能力を利用できる．**表 2** は SOLO で使われている CPM 命令の静的使用頻度で上位 10 の総和が約 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．そこで No. 11 ではこれら 10 種の命令をファームウェア化した他，I 内でよく使われるルーチンもファームウェア化した．No. 12 は K でよく使われる待ち行列管理ルーチンなどをファームウェア化している．No. 13 はフローティング演算用命令である．

3.4 性 能 評 価

表 3 は 3.1 節の (1) と (3) の場合の K, I のメモリサイズとエミュレータのマイクロプログラム量を示している．K, I のサイズは CL 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では (3) の方が

表 1 C-Language の命令型式
Table 1 C-Language instruction FORMAT.

NO	BIT NUMBER 0 1 2 3 4 5 6 7 8 9 A B C D E F	COMMENTS
0	0 0 0 SCS .... D.........OP 1	S* op D ⟶ D
1	0 0 0 DCS .... D.........OP 1	S op D* ⟶ D*
2	0 1 0 0 ‡ DCD.........OP 1	CONST op D* ⟶ D*
3	0 1 0 1 ‡ ADDRESS ....OP 1	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 CONTD .... OP 2	SHIFT
7	1 0 0 1 CONT. OP 1	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 } operand mode
DC: Destination } operand mode
0: Rn 1: (Rn) 2: (Rn)+ 3: X(Rn).
OP 1: 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 2 CP マシン命令の静的使用頻度
Table 2 Percent destribution of concurrent pascal machine instruction.

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 3 プログラムサイズ
Table 3 Program size.

		PDP 11 Emulator	C-Language
μ program	(32 bits/word)	1456	981
Kernel 2	(16 bits/word)	1180	1081
Interpreter	(16 bits/word)	1084	850

CPM 命令のファームウェア化を行っているにもかかわらず(1)より少ない．これは PDP 11/45 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，CPM 命令の多くが比較的少ないマイクロプログラムで実現できることによる．ちなみに，PDP 11/45 のオペランド部のデコードと操作に 336 ステップ要し，表 2 に示した CPM 命令のファームウェア化は全部で 168 ステップで済む．

表 4 は CPM の主な命令について，その実行マイクロステップ数とメモリアクセス回数を 3.1 節の 3 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 PDP 11/45 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．CL が PDP 11/45 と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，仮想アドレッシングによるところが多い．CL ではインプリメントの都合上 PDP 11/45 の仮想アドレッシング方式を踏襲しているため 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 CPM 命令が PDP 11/45 の数ステップで書かれ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に生かせない原因である．表中の直接実行とは，3.1 節の(2)に相当し，CL の値から，その命令フェッチを除いたものである．

表 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 SP で書いた 2 つのプログラムと CP で書いた 1 つのプログラムをコンパイルし，CPM の命令になったプログラムを 2.2 節に示した 2 種類の I と K（1 プロセッサ用）で解釈実行した時の処理時間を示している．なお，実システムの測定ではマイクロプログラムキャッシュのヒット率が影響することと，ハードウェアシステムの動作を安定させるためバスの動作速度を目標値より落としているため厳密な比較データとは言えないが，表 4 の値を反映していることはわかる．

4. マルチプロセッサのための CPM

この章では，ACE システムで行った CPM のマルチプロセッサ化について述べると同時に，マルチプロセッサ CPM の性能をシミュレーションや実測をもとに示す．

4.1 マルチプロセッサの CPM

表 4 CP マシン命令実現ステップ数
Table 4 Concurrent pascal machine instruction execution steps.

CPM Instruction	PDP 11 Emulator		C-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 5 実験プログラムの実行時間
Table 5 Run time of example programs.

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

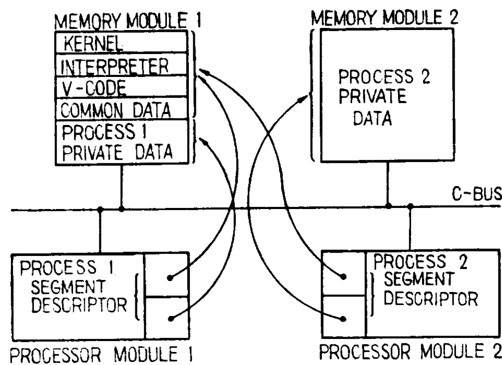


図 4 ACE システムの場合のプログラム配置
Fig. 4 Physical storage assignment of ACE system.

図4は ACE システムにおける CPM の構成である．CP の並列プロセスは仕様の同じ2台のプロセッサに割当てられて実行される．CP プログラム，カーネル，インタプリタはすべて共有メモリに置かれる．

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

4.1.1 プロセッサのスケジューリング

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．ここでは先ずプロセス切換えを引起こす CPM の命令と，その概要をあげる．

（1） ENTER MONITOR: 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） EXCIT MONITOR: 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: CP 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: 入出力起動命令でプロセスは待ちになり，IO 割込みで復起する．

次に ACE システムでのスケジュール法を示す．

プロセスとプロセッサの対応付けは，あらかじめユーザの指定により固定的に割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ようにした．スケジューラは，プロセッサの状態，レディキュー，ウェ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移されずにプロ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当てられるのを待つし，オンプロセッサなら，即再スタートさせる．

4.1.2 メモリの割当て

マルチプロセッサの場合の論理的ストアアロケーシ

ョンは図2と同じであるが物理的配置はいろいろ考えられる．Hansen は，各プロセッサにローカルメモリを持たせ，共有プログラムのみを共有メモリに置く方法[6]を示している．ACE システムはキャッシュメモリ方式をとっており，各プロセッサは大量の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，共有を問わずすべて共有メモリに割当てている．

この方法は，プロセッサとプロセスを動的に割当てる方式では便利である．各プロセスのアドレス空間は図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ィスクリプタにより生成している．

4.1.3 モニタの相互排除

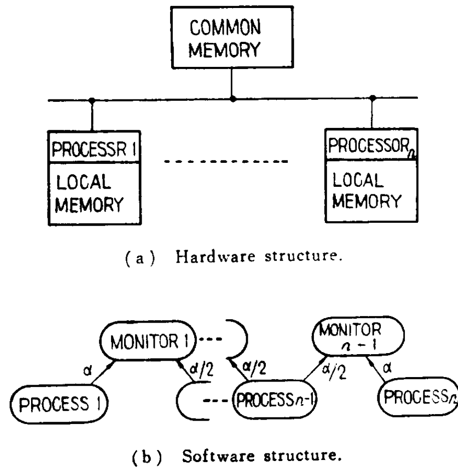
モニタの相互排除法としては，全モニタを一括して相互排除する方法と各モニタ別に相互排除する方法が考えられる．前者はモニタルーチンをすべて1つの共有メモリに置き，共有メモリ単位で相互排除してしまえば簡単に実現出来る．しかし，CP の思想及び効率の点で後者が優れるため今回のインプリメントでは後者を選んだ．これらに関する評価は次節に示す．我々の方式によれば，モニタルーチンの相互排除にはCPM命令の ENTERMONITOR, EXCITMONITOR をそのまま使える．しかしマルチプロセッサでプロセスをスケジュールする際，スケジューラが複数のプロセッサで同時に実行されると矛盾を引起こす可能性がある．スケジューラを駆動する命令には上述の ENTERMONITOR, EXCITMONITOR の他 CONTINUE, DELAY, IO があり，IO 割込みでもスケジューラにアクセスする．そこで ACE システムでは，同期モジュール[4]を用いてこれらの命令や割込み処理が複数のプロセッサで同時に実行されないよう制御している．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．クリティカルリージョン（上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) と現在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 (T&S) を待ち，それらの値は READ リクエストでバスに読出される．INC と T&S は READ でインクリメントされ，T&S 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，T&S 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T&S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 T&S 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めぐって来たかどうかを知ることができる．以上が原理であり，上述のプロセススケジュールに関する命令では，その実行に入る前に同期モジュールにリクエストを出すし，I/O 割込みではI/Oプロセッサが割込む前にリクエストを出すことにより相互排除を実現している．ちなみに I/O 割込み処理はどのプロセッサでも行えることが望ましいが，今回のインプリメントでは簡単化のため1台に限っている．この他プロセス切換えの原因として，タイマがある．1プロセッサ用 K ではタイマ割込みを IO 割込みと同様に扱ってサポートしているが，マルチプロセッサ用Kではタイマ割込みを行っていない．そのためプロセスがモニタで同期を取らず独立に走るようなものでは制限になる場合もある．

4.2 マルチプロセッサにおける CPM の性能評価

マルチプロセッサ上で CP プログラムを走らせることの利点は，プロセスを並列に実行できること，プロセス切換えのオーバヘッドを減少できることである．しかしプロセス並列実行を妨害する要因としてモニタプロシージャ入口での待ち，ディレイなどがあるため，それらに関する検討が必要である．この節では，マルチプロセッサ上で CP を走らせる場合の性能をシミュレーションと実測によって示す．

最初のシミュレーションは，モニタ入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，モニタ入口で待たねばならない．

また，いくつかのモニタが1つの共有メモリに入っているとすると，モニタにアクセスできても，共有メモリの他のモニタがアクセスされていれば，その間待たなければならない．ここでは各プロセスが稼動している割合（全体から待たされている割合を引いたもの）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タにして求めた．シミュレーションモデルは**図5**であり，プロセ



(a) Hardware structure.

(b) Software structure.

図 5　シミュレーションモデル
Fig. 5 Simulation model.

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当てられている．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い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 $\alpha/2$ 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6**は，プロセッサ数を変えていった時のプロセッサの稼動率である．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

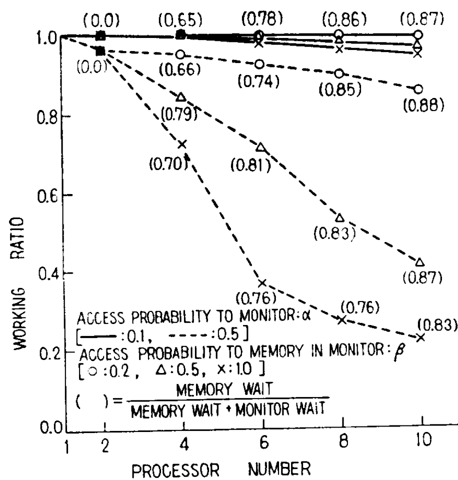


図 6　プロセッサの稼動率
Fig. 6 Working ratio of processor.

共有資源の利用率が 50% 以下ではモニタ入口，共有資源での待ちの影響が無いといえよう．ここで $\beta=1$ 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は待たされる．我々の採用したモニタ別に相互排除するスケジュール法は，$\beta=1$ ではないため共有メモリの待ちを減らしている様子がよくわかる．

第2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，マルチプロセッサの利点があまり生かせない場合もある．ここでは，そのような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待たされる．消費プロセススは平均 t_β 時間ごとにバッファから取出すが，バッファが空なら入って来るまでディレイして待つ．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7**はこのような状況でのプロセスの実行，モニタ入口での待ち，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タにして求めた結果である．この結果は負荷バランスの良いシステムが無駄なく働くという当然の事を示しているが，バッファサイズがあまりシステムの性能に大きな影響を与え

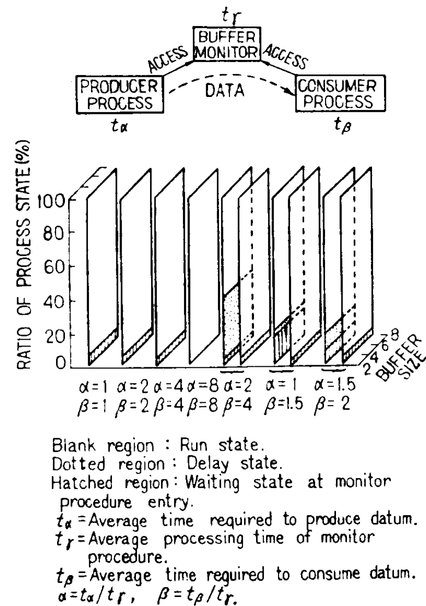


図 7　生産者・消費者モデル
Fig. 7 Producer consumer model.

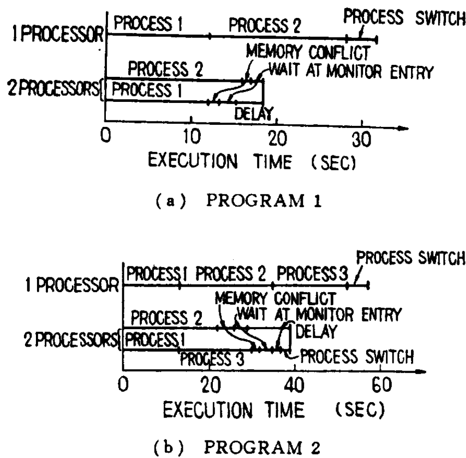


図 8　ACE システムにおける実測結果
Fig. 8　Measurement of example program on ACE.

ていないことがわかる.

最後は CP プログラムをマルチプロセッサ上で実際に走らせて行った実測である(**図 8**). ここでは CP プログラムをコンパイルし CPM 命令に落ちたものを, マルチプロセッサ用カーネルとインタプリタにより実行させ時間の測定を行った. 実行に当っては SOLO は使用しておらず I/O も含んでいない. プログラム 1 は 2 プロセスでプログラム 2 は 3 プロセスであり, これらをそれぞれ 1 台と 2 台のプロセッサで実行した. このうちプログラム 1 は図 7 と同型で α と β はそれぞれ約 1.5 と 2.0 であり図 7 の結果とも大体合っている.

以上からもわかる通り, マルチプロセッサの性能はアプリケーションにより大きく異なる. 実測での例題はプロセスのバランスが大体と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 システムの場合はその状況によっても大きく異なるが, 大体においてプロセス間のバランスが悪く, モニタへのアクセスは少ない場合に相当し, 平均すると 2 台のプロセッサによる効率向上は 10% 程度であった.

5. おわりに

ACE システムにおける CPM をマイクロプログラムによる CPM の実現と, マルチプロセッサのための機構という点からまとめた. この研究は, マルチプロセッサによる CP の効率良い実現法を示すと共に, マルチプロセッサシステムのソフトウェアの問題を CP という高級言語で解決しようという提案でもある. 3 章の評価データは, マイクロプログラムやエミュレーションに興味を持つ方には参考になろう. 筆者は, これらに基づいてさらに効率の良いエミュレータの設計を行っている. 4 章のシミュレーションはマルチプロセッサによる CPM の性能をできるだけ一般的な形で求めようと試みたものであり, マルチプロセッサによる実現の有意性が充分読み取れる. なお CP をマルチプロセッサ用言語として使う場合, 言語から来る制限で, プロセスの生成, 消去に制限がある. これらは今後解決しなければならない問題であるが, 実際には CP で充分な場合が多いと思われる.

最後に, この研究に興味を向けて下さった成蹊大学和田弘教授, 絶えざる御援助をいただいた当研究室の飯塚肇室長, 藤井狷介の両氏, そしてこの研究の機会を与えられた黒川一夫, 西野博二の各部長に感謝の意を表します.

参 考 文 献

1) Iizuka, H. et al.: ACE—A New Modular Computer Architecture, Proc. US-J comp. conf. 2, 36-41 (1975).
2) 飯塚: 適応構造計算機に関する研究, 電総研研究報告, No. 767 (昭 51 年 11 月).
3) 飯塚, 古谷: マイクロプロセッサアーキテクチャの一設計, 信学論 D, 59 D-3 (1976).
4) 古谷ほか: マルチプロセッサ用相互排斥モジュールの一設計, 情報処理 18-6 (1977).
5) Hansen, P. B.: The Architecture of Concurrent Programs, Prentice-Hall (1977).
6) Hansen, P. B.: Multiprocessor Architectures for Concurrent Programs, SIG ARCH Vol. 7, No. 4 (15 December 1978).

(昭和 54 年 6 月 13 日受付)
(昭和 54 年 10 月 25 日採録)