マルチプロセッサシステム（ACE）における Concurrent Pascal マシン

Concurrent Pascal Machine on Multiprocessor System (ACE)

古谷立美
Tatsumi FURUYA
電子技術総合研究所 電子計算機部
Computer Division, Electrotechnical Laboratory

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。そしてこの様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。しかし汎用の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作りが行なわれ，その結果ハードウェアはでき上っ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大きな原因の一つにソフトウェアの問題が有る。MPSで走るソフトウェアとしては，相互作用の有る並列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は本来相互作用の有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もとづいて，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ものであるが，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えによるオーバヘッドが多いのではないかと考えられている。

そこでMPS上でCPの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が減少し効率向上が望めると共に，CPをMPS用の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P.B. 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マルチプロセッサシステム上にこれを実現した。但し仮想マシンの命令セットはP.B. Hansen のものと同一である。

以下この論文では，マイクロプログラム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P.B. Hansenの仮想マシン（Concurrent Pascal machine）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

ていく試みで，次の様な特徴を持つ。

(1) PMS（Processor-memory-switch）レベルのモジュール構成を採用し，モジュール間通信には汎用性のあるバス方式を採用した。

(2) 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サ（PULCE[3]アーキテクチャ）を使用。

(3) プロセッサモジュールにはその他ダイナミックマイクロプログラミングを可能にするマイクロプログラムキャッシュ，データキャッシュ，データ長アラインメント機構等の機能が有る。

(4) ブロードカスト方式の通信方式。

ACEシステムは現在Fig.1の様に構成され，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。同期モジュール[4]はプロセス間の同期をとるために設計したもので，P.Vオペレーションが容易に実現できる。

2.2 Concurrent Pascalマシン[5]

P.B.Hansen はCPを用いてSOLOというOSを作成し配布した。そこでは1人のユーザがConcurrent PascalとSequential Pascalプログラムをコンパイルしたりエディットしたり実行することができる。彼はこのシステムを作るに当ってマシンインディペンデントな仮想マシン（CPM）を設定した。即ちSOLOシステムを使いたい人は，CPMの命令のインタプリタと並列プロセスを1台のプロセッサに写像するマシンディペンデントなカーネルを作成すればよい。尚彼はCPMをPDP 11/45に実現したため，PDP 11/45用のインタプリタとカーネルの資料は提供されており利用できる。Fig.2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，CPMの命令から成るプログラムが生成される。それがV-CODEで，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出来ている。各プロセスには，それぞれのスタックがデータセグメントに割当てられる。Fig.3はCPプログラムがどの様に実行されるかを示した例である。

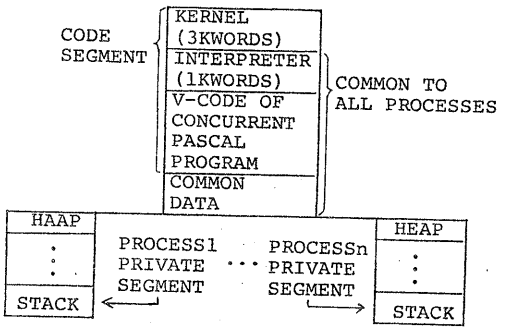


Fig.2. Concurent Pascal Machine Store Allocation.

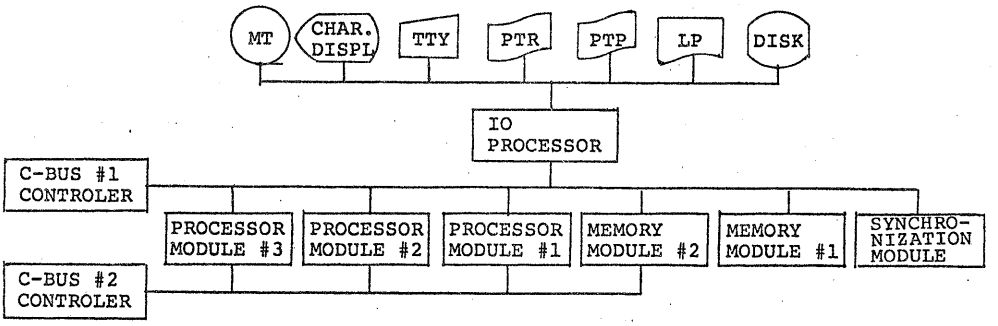


Fig.1. ACE system.

3. マイクロプログラムによるインタプリタの実現

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

3.1 インタプリタの実現法

CPプログラムが解釈実行されていく過程がFig.3である。CPプログラムはコンパイルされCPMの命令であるV-CODEのプログラムが生成される。(Fig.3(a))。CPMの命令は実際のマシン(P.B.Hansenの場合PDP11/45)によってシミュレートされる(Fig.3(b))。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はカーネルが呼ばれる。

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有る。

(1) PDP11/45のエミュレータを作る。：これはFig.3(b)に示すもので，マイクロプログラムでエミュレータを作ればP.B.Hansenから提供されているインタプリタとカーネルをそのまま使用出来る。この方式は作業量は最も少なくてすむが効率が悪い。

(2) 機械語レベルを設定せず，カーネルとインタプリタを全てマイクロプログラムで書く。この方法はFig.3(d)に示すもので，効率は良いが，マイクロプログラム量が膨大になる。

(3) CPM向き言語(CL)を設定する。：(1)と(2)の中間的なもので，インタプリタとカーネルの特性を考慮に入れて，これらを効率良く実現できる機械語(CL)を設定しそのエミュレータを作る。この方法は効率は(1)よりも良く，マイクロプログラム量は(2)より少ない。(Fig.3(c))。

インタプリタの実現法としては，これ以外の方法も考えられるが，今回の試作は(1)と(3)の方法で行なった。

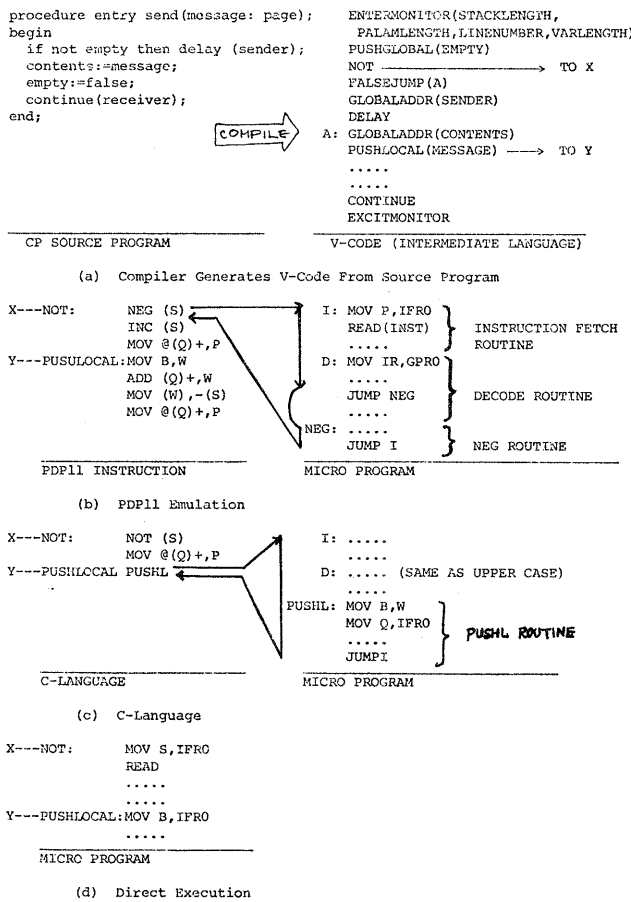


Fig.3. Concurrent Pascal program execution process.

3.2 PDP 11/45 のエミュレータ

マイクロプログラムプロセッサ(PULCEアーキテクチャ)は16ビットのデータ処理巾を持ち，32語のレジスタファイルを有するため，PDP 11/45のレジスタ(フローティングプロセッサ用は除く)は全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る。

3.3 CPM向き言語（C-language）

Table 1はこの言語の命令型式表で，大きく2つの型に分けられる。第1の型（No.0～10）は機械語に相当するレベルであるがPDP 11/45の命令と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PDP 11の様な論理の深い命令を必要としないことと，次に示す第2の型が多くの機能を吸収したためである。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つの機械語を割当てるもので，マイクロプログラムの能力を利用するものである。Table 2はSOLOシステムで使われている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の命令ではこれら10種の命令をファームウェア化した他，この命令の他でもインタプリターでよく使われるルーチンをファームウェア化している。No.12はカーネル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を割当てた。

Table 1. C-Language Instruction FORMAT.

NO	BIT NUMBER 0123456789ABCDEF	COMMENTS
0	000SCS...D...OP1	S*op D⟶ D
1	000DCS...D...OP1	S op D*⟶ D*
2	0100# DCD...OP1	CONST op D*⟶ D*
3	0101#ADDRESS OP1	CONST op (ADDR) ⟶ (ADDR)
4	0110D...ADDRESS.	LAOD
5	0111S...ADDRESS.	STORE
6	1000 CONTD...OP2	SHIFT
7	1001 CONT. OP1	STACK OPERATION
8	1010ADDRESS.....	JUMP
9	1011OFFSET COND.	CONDITIONAL JUMP/JUMP SUBR.
10	1100MISCELLANE.	
11	1101INTERPRETER	FOR FPEQUENTRY USED CPM INST.
12	1110 KERNEL	QUEUE MANIPULATION ETC.
13	1111 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: OPERATION1
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

3.4 性能評価

Table 3は，3.1節の(1)と(3)の場合のカーネル，インタプリタ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。カーネルとインタプリターのサイズは，これらのために設計したC-language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CPM命令のファームウェア化を行な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，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ステップを要し，CPM命令のファームウェア化は168ステップで済む。

Table 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）も参考のために付加えた。C-languageがPDP 11/45と比

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

Table 3. Program Size.

	PDP11Emulator	C-Language
μprogram (32bit/word)	1456	981
Kernel 2 (16bit/word)	1180	1081
Interpreter (16bit/word)	1084	850

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有るが，仮想アドレッシングによるところが多い。1回のメモリアクセスで約5ステップを必要とする。又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，C-languageからその命令フェッチを除いたものである．

Table 5 はACEシステムで実際に3つの実験プログラムを走らせた時の処理時間測定結果である。実システムで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関係することと，システムの動作を安定させるためシステムの動作速度を目標値(200ns)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，大体Table 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

ユーザの記述した並列プロセスを実際のマシンに写像するのがカーネルである。P.B. Hansenの場合にはプロセッサが1台であったが，ACEシステムの場合プロセッサが複数で，そのためのカーネルが必要となる。並列プロセスをマルチプロセッサで動作させる場合の，1プロセッサの時との違いはプロセッサのスケジュール問題である。

この章では先ず，スケジュールを必要とするCPM命令の概要を示し，次にACEシステムに適用したプロセッサスケジュール法を示し，最後にそれらの評価を示す。

4.1 CPMのプロセス同期命令

1台のプロセッサで複数のプロセスが走って

Table 4. Concurrent Pascal Machine Instruction Execution Steps.

CPM Instruction	PDP11Emulator		C-Language		Direct Exec.		PDP11	
	μSteps	MMAC*	μSteps	MMAC	μSteps	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.

Table 5. Run Time of Example Programs.

Program Name	PDP11Emulator	C-Language
1. HANOI	82ms	36ms
2. PERMUTATION	38ms	21ms
3. A Concurrent Pascal Program	24.51s	16.04s

いる様に見せるため，P.B. Hansen はプロセスの実行を17 msごとに切り，TSSの要領でプロセス切換を行なっている。CPプログラムでは，この様なタイムスライスによるプロセス切換えの他に，ソフトウェア的なプロセススケジュールが可能である。CPM命令のうちプロセススケジュールに関係するものには，プロセスの同期命令と入出力命令が有る。

この節ではそれらの概要を示す。

(1) ENTER MONITOR：モニタのプロシージャには同時に2つ以上のプロセスがアクセスしてはいけない。この命令はモニタのゲートを調べ"open"ならばモニタプロシージャへのアクセスを許し，"close"なら待ちに入れる。

(2) EXCIT MONITOR ：モニタプロシージャの出口で出される命令で，ゲートで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が有れば，それをレディ状態にする。

(3) DELAY ：CPのディレイ命令に対応し，走っているプロセスを待ちにしてモニタプロシージャを出る。

(4) CONTINUE：DELAYで待たされている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割込みで復帰する。

4.2 メモリ割当て

マルチプロセッサの場合のストアアロケーションは，論理的には1プロセッサの場合(Fig.2)と同じである。しかし物理的配置は，マルチプロセッサのアーキテクチャに応じて，いろいろの方法が考えられる。Fig.4はACEシステムでのメモリ割当て例を示している。これは2つのプロセスが2台のプロセッサで走っている場合で，カーネル，コモンセグメント，プロセス1用プライベートセグメントをメモリモジュール1に，プロセス0用プライベートセグメントをメモリモジュール0に割当てている。各プロセッサモジュールは，セグメントデスクリプタによって自分のアドレス空間を決定している。

コモンセグメントはコモンメモリに置くべきであり，プライベートセグメントは各プロセッサモジュールのローカルメモリに置く方が望ましいが，ACEのローカルメモリは容量が不足のためコモンメモリに置いている。メモリの配置はこの他にもいろいろ考えられる。例えば，インタプリタとV-CODEのコピーをメモリモジュール0におきプロセッサモジュール0はそちらを使う等。Fig.4のHEADとはカーネルとV-CODEのインタフェースとなるプライベート領域である。

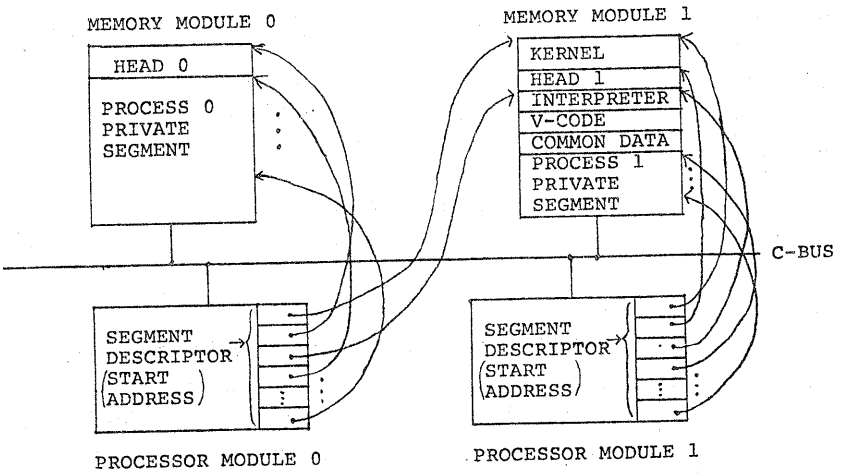


Fig.4. Physical storage assignment of ACE-system.

4.3 プロセッサのスケジューリング

マルチプロセッサ用のCPMを実現するには，プロセス切換え時のプロセッサスケジューリングと，そのスケジューラが複数のプロセスから同時にアクセスされない様にする相互排除が必要となる。スケジューラの相互排除にはACEシステムの同期モジュールを使用している。同期モジュールはTest&Setビット，インクリメントカウンタ，リクエストカウンタから出来ていて，プロセッサモジュールからのREAD命令とWRITE命令により動作する。同期モジュールを利用したスケジューラの相互排除は次の様にして実現される。

ステップ1：4.1節であげた命令ではプロセススケジューラへのアクセスが起こるため，それらの命令の最初に同期モジュールへREAD命令を出す。

ステップ2：上のREAD命令で，同期モジュールのTest&Setビットとインクリメントカウンタの値が読出されプロセッサモジュールに送られる。これと同時に同期モジュール内ではTest&Setビットがセットされインクリメントカウンタとリクエストカウンタがインクリメントされる。

ステップ3：プロセッサモジュールでは，読出されたTest&Setビットがセットされていなければスケジューラに入る。セットされていれば自分の番がブロードカストされるまでプロセッサを放棄せずに待つ。

ステップ4：スケジューラを出る時プロセッサモジュールは，自分がREAD命令で読込んだインクリメントカウンタの値に1をかえた値をWRITE命令でブロードカストする。即ち，この値が次にスケジューラへアクセスしようとしているプロセスへの起動信号となる。これと同時に同期モジュールではリクエストカウンタの値がデクリメントされ，もしカウンタが0ならTest&Setビットをリセットする。

プロセッサのスケジューリングは，プロセス数がプロセッサ数以下の時は，プロセッサとプロセスを一対一に対応付けることが出来て問題は無い。プロセス数がプロセッサ数より多い場合が問題で，今回のインプリメントではTable6の様な待ち行列と状態を導入して効率向上を計った。同表の"ONPROCESSOR"とは，プロセス状態ベクトルのセーブ，ロードのオーバヘッドを最小にするために導入したもので，プロセスがWAIT状態になってもプロセスとプロセッサの関係を切離さずに保っておく状態である。そしてそのプロセスが再びREADY状態になった時にはGO信号により再スタートさせられるし，他のプロセスがREADY状態になり，かつ空きプロセッサが無い時はCHANGE信号で状態ベクトルの退避が行なわれ"OFFPROCESSOR"のWAIT状態にされる。Fig.5はマルチプロセッサ用プロセス同期命令のアルゴリズムを示している。

4.4 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利点は，プロセスの並列実行とプロセス切換えのオーバヘッドの減少である。そしてプロセスの並列実行を妨害する要因としては，モニタプロシージャ入口での待ち，ディレイ，そしてスケジューラへのアクセス待ちなどが有る。

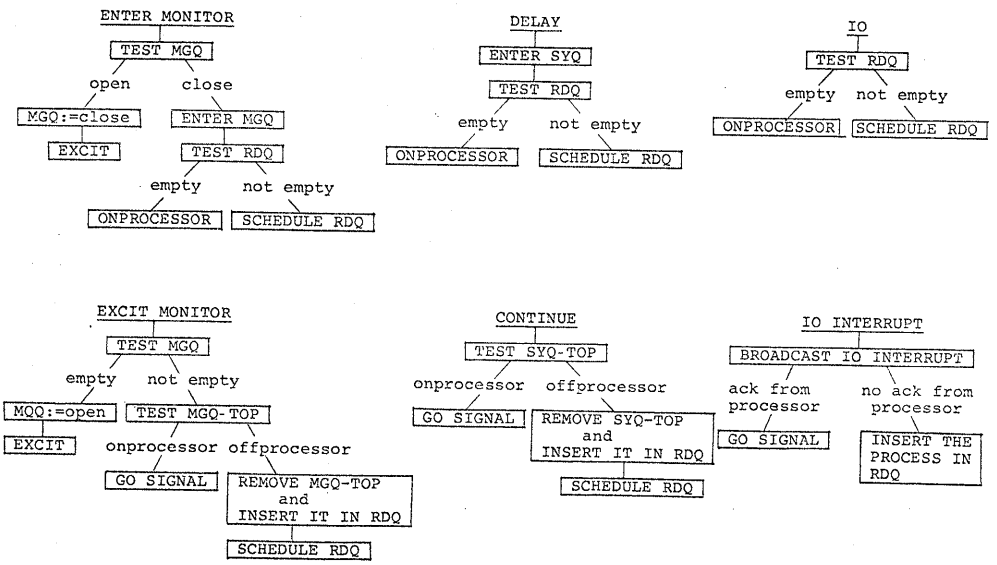
この節ではこれらの事項について行なったシミュレーション結果とACEシステムでの実測データにもとずいた性能評価を行なう。並列処理の性能は応用により大きく異なる。ここでのシミュレーションは，単純であるが重要な一般性を持つ3つのモデルに対して行なった。

第1のモデルは，生産者－消費者モデルであり，次のモデルと共にモニタの最も標準的型である。生産プロセスは，ある時間処理を行なってデータを生成し，バッファへ送るという処理をくり返す。消費プロセスは，バッファからデータを取って来て処理をするという操作をくり返す。以上2つのプロセスが共通にアクセスするバッファの管理を行なうのがモニタルーチンで，バッファの状態に応じてプロセスのスケジュールを行なう。即ちバッファが満の時生産プロセスがデータを送り込もうとした時は生産プロセスをディレイさせ，逆に空のバッファに対して消費プロセスがリクエストを出すと消費プロセスをディレイさせる。又ディレイさせられているプロセスに対しては，条件が整った時にCONTINUEさせる。Fig.6(a)は，プロセスの実行時間，モニタプロシージャ入口での待ち時間，ディレイ時間の割合を，モニタプロ

Table 6. Data Structure and State for Processor Scheduling.

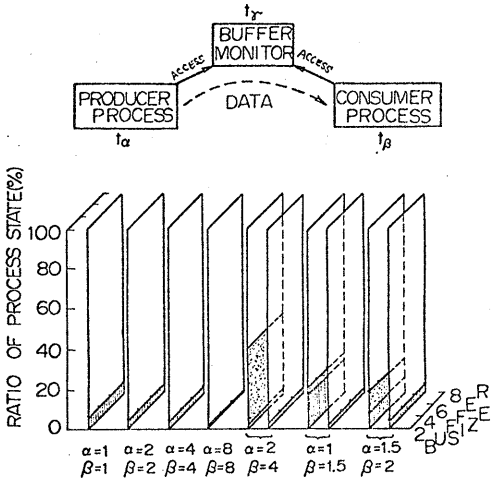
A. Data Structure
(1) Process Table: This containes state of each process and link to the queue which is listed below.
(2) Ready Queue: The ready process which is waiting a idle processor enters this queue.
(3) Monitor Queue with Gate: If the gate is open, process can enter monitor procedure. The process which can not enter the monitor procedure enters this queue.
(4) Process Synchronization Queue: This queue corresponds to semaphore variable. Delayed process enter this queue.

B. Process State
(1) RUN: The state in which process is executing on a processor.
(2) WAIT: The state in which process is waiting some events.
  (i) ONPROCESSOR: Process state is kept on processor.
  (ii) OFFPROCESSOR: Process state is saved to main memory.
(3) READY: The state in which process is waiting processor for run.



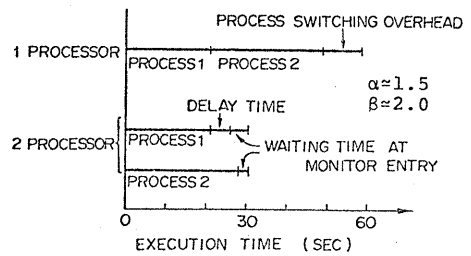
RDQ: Ready Queue
MGQ: Monitor Queue with Gate
SYQ: Process Synchronization Queue
SCHEDULE RDQ: This includes some algorithm for deciding which of the several processes that are ready to run should be run next.

Fig. 5. Process scheduling algorithm.



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

(a) Ratio of process state.



(b) Measurement of a example program on ACE-system.

Fig. 6.Producer-Consumer model.

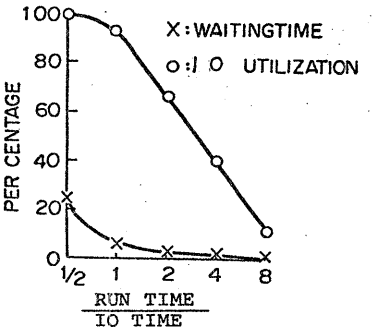


Fig. 7.Waiting time and IO utilization.

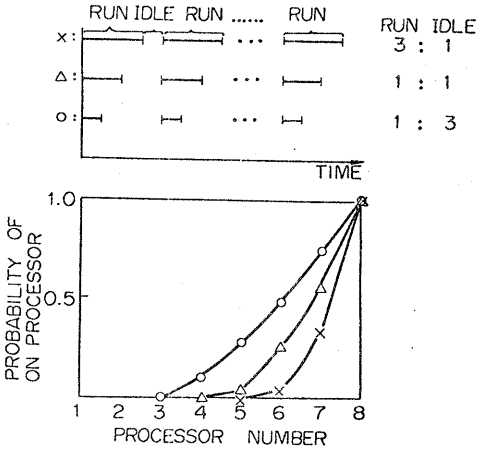


Fig. 8. Probability of ONPROCESSOR.

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ものである。尚，ここではZつのプロセスがZ台のプロセッサにそれぞれ割当てられているものとする。一般にプロセッサがn台有り，各プロセッサの稼動率がρであれば1台のプロセッサがフル運転している場合と等価になる。Fig.6(a)では待ち時間の割合が悪い場合でも20%程度であり，マルチプロセッサの効果が充分得られることがわかる。この様に並列性が上がる理由としては，あるプロセスがモニタを出ようとする時他のプロセスがモニタ待ちをしているとすると，この後2つのプロセスは必ず並列実行に入るという同期性があげられる。このことは筆者が先に行ったマルチプロセッサシステムの解析[6]で得られた結果と一致している。Fig.6(b)は，このモデルと同型のCPプログラムを実際に書き，ACEシステム上で走らせ実測を行なったものである。この時のαとβは約1.5と2.0であり，図(a)の結果と大体合っている。

次のモデルは相互排除型モニタの例で，周辺器械の管理などに使われる。Fig.7は，2つのプロセスがEOを共有する場合のモニタで，EOの利用率と待ち時間を求めたものである。ここでも2つのプロセスは，2つのプロセッサに割当てられているものとしている。これからも待ち時間はさほど多くないことがわかる。

最後のシミュレーションは，プロセス数がプロセッサ数より多い場合のプロセス切換えに関するもので，"実行"と"待ち"を繰返している8つのプロセスが，n(1…8)台のプロセッサで動作していると仮定して，あるプロセスが待ちから"実行"に移る時プロセスの状態がONPROCESSORのままに保たれている確率を求めた。(Fig.8) この結果ONPROCESSORの効果はプロセッサ数とプロセス数が非常に近いところで特に上がることがわかる。スケジューラ入口での待ちは，上のいずれの場合でも1%以下で無視出来る。

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の有る方には参考となろう。筆者は，これらにもとずいてさらに効率良いエミュレータの設計を行なっている。4章のシミュレーション結果は，マルチプロセッサによるCPMの一側面を照らしたものであるが，全体に通じる真理を含んでおり，マルチプロセッサによる実現の有意性が十分読取れる。尚CPをマルチプロセッサ用ソフトウェアとして使う場合，言語から来る制限で，プロセスの生成，消去に制限が有る。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，藤井捐介の両氏，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) H.Iizuka etal：" 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, 59D-3, 1976
(4)古谷他：" 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) P.B.Hansen：" The architecture of Concurrent Programs" Prentice-Hall, 1977.
(6)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17-5，1976