《解　説》

C.mmp マルチ・ミニ・プロセッサについて

飯塚　肇*　古谷立美*

0. はじめに

C.mmp (このシステムの提案者であるC. G. Bellの記法[1]を用いて，このように呼ばれている．mmp は，multi-mini-processor の意である．) は，1971年夏以来，Carnegie-Mellon University (以下 CMU) において開発中の，ミニ・コンピュータ複合システムである．現在開発中のこの種のシステムの中でも，もっとも本格的なものの1つであり，その後のこの分野の研究に，大きな影響を与えたといってよい．したがって，すでにしばしば引用・解説されており，その概要はかなりよく知られているが，計算機複合体を論じる時には省くことができない重要なシステムであるから，本稿では，若干の私見を加えながら，その特徴を整理し意義を考えてみたい．

1. 開発の背景と目的

C.mmp の開発の背景と目的は，このシステムの中心設計者である Wulf と Bell の論文[2]に，明瞭にのべられている．

まず，1971年に，コストの制約をはずして，人工知能の研究に望ましい計算機システムはどのようなものかについて，一般的考察が行なわれた．結果は，約8M語 (74ビット/1語) という大きな主記憶を，多数の汎用プロセッサと専用プロセッサで共用するという，きわめて大規模なものであった (C.ai[3] と呼ばれている) が，C.mmp は，その結果を背景に，もっと実際的なマルチ・プロセッサ・システムを構成しようとする努力である．

その開発には，つぎの2つの大きな目標がある．

(1)　現在ある他の研究プロジェクトに，必要な計算パワーを与えること．

(2)　計算機構造とシステム・プログラミングの研究を行なうこと．

(1)の研究プロジェクトは，具体的にはスピーチと画像 (vision) 認識に代表される人工知能の研究であり，ここでは，実時間処理を行なえることが大きな条件である．その他に，RTM* を用いた実験システムの設計，試験にも，用いることが考えられている．

(2)は，計算機アーキテクトの永年の夢であった，本格的マルチ・プロセッサ構造計算機の実際的研究と，OS の研究，とくにソフトウェアシステムの構成法の研究を行なうことである．

ここで，C.mmp に，2つの相反するとも思える目的があることについて考えてみよう．もっとも，これらを相反するとするのが誤りであって，本来両立されるべきなのがあたりまえなのであるが，普通は，(1)のグループは計算機パワーの取得に性急であり，また，新規開発のシステムに信頼をおいていないから，(2)の目的との協調を望まないし，(2)のグループは，(1)によって自己の考えをゆがめられることをきらう．したがって，(1)と(2)が同時に存在するプロジェクトは，これまでほとんど例を見ないといってよい．

(2)と同時に(1)をおくことは，(2)にとっては，そのシステムの不完全性に対するいいわけと取れないこともないが，新しいすぐれた計算機システムの開発には，こうしたあらゆる分野の協調がぜひ必要であることを示すもの**として，注意すべきであろう．

さて，話をもとへもどして，C.mmp の開発の背景であるが，これに対して，Wulf と Bell は，これまでマルチ・プロセッサの開発をさまたげてきた要因をあげ，それがいかにして解消されつつあるかを指摘して，この種のマルチ・プロセッサ開発に取り組むことの妥当性を説明している[2]．これを整理したものを，表1に示した．

要するに，集積回路の発展によって，ハードウェアのサポートが満たされ，ソフトウェアも，過去の経験の蓄積の結果，複雑な並列システム実現の可能性がでてきたということであろう．

つぎに，C.mmp を試作することによって行なう具体的テーマとしては，つぎのようなものがあげられている．マルチ・プロセッサ・ハードウェアの設計，プ

* 電子技術総合研究所電子計算機部計算機方式研究室

* Register Transfer Module

** CMU の V.R. Lesser は，CMU の計算機学科の最もよい点は，ハードウェア，ソフトウェア，応用などの研究者が密接に協力していることで，アメリカでも例が少ないことだといっている．

表 1 マルチ・プロセッサを妨げていた要因と情況の変化

	マルチ・プロセッサの開発を妨げた要因	対策あるいは情況の変化
1	プロセッサおよび主記憶の絶対コストが大きかった.	ミニ・コンピュータのマルチ・プロセッサに限定すれば，コストは研究あるいは実際に利用できる範囲内に入ってきた.
2	全体のコストに対して，プロセッサの相対コストが大きく，プロセッサ数を増しても，コスト・パフォーマンスを改善できなかった.	全システム・コストに対するプロセッサの相対コストが減少した.
3	OS，ソフトウェアの信頼性が劣り，パフォーマンスが劣化する.	多数の OS が作られた結果，その信頼性が向上した.
4	マルチ・プロセッサに必要な中央スイッチを製作する技術が発達していなかった. これは，部品の低集積度と高コストのためである.	LSI によってスイッチの製作が可能になった.
5	メモリ・アクセスの衝突やスイッチでの遅延によるパフォーマンス損失.	プロセッサ，メモリ，スイッチのバランスがよければ，メモリの衝突は大きくない.
6	1つのタスクを，並列に実行できるサブ・タスクに分解する問題が未知であった.	Illiac IV, CDC-STAR などの研究によって，ある程度解明された.
7	並列処理のできる環境において，それを制御する方法や複雑なプログラムを取り扱う方法が未発達であった.	fork-join (Conway), P. V. 操作 (Dijkstra) などが研究され，大きな複雑なプログラムの作成法も発展しつつある (Structured Programming など).

ロセッサとメモリ間の結合（とくに信頼性に関して），マルチ・プロセッサ上での計算処理の構成法，タスクの分解，OS の設計とパフォーマンス，全システムのパフォーマンスの測定と解析，高レベルでの構成法による高信頼計算の実施.

2. システムの特徴

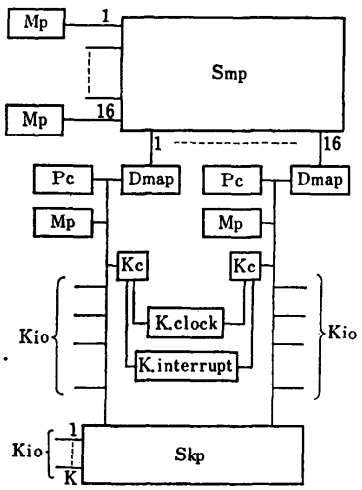
C.mmp のシステム構成は，3節に説明するように，ごく標準的なメモリ共有型マルチ・プロセッサ形式を採用している. 複合計算機の分類は確定していないが，筆者自身の分け方に従えば[4]，これは均質型に含まれる. つまり，C.mmp では，どのプロセッサもメモリ・モジュールも相互に区別はなく，まったく同一に扱われているから，一般性，汎用性は大きく，形式的美しさがあり，信頼性にすぐれている. 逆に，1つの問題を効率よく並列処理することは相当困難で，多分にソフトウェアによりかかっている.

興味あることは，こうしたシステム構成が，ほぼ先験的に採用され，先にあげた研究対象の中に含まれていないことである. 事の当否は別にしても，このことは，このシステムの，あるいはこのプロジェクトの，1つの性格を表わすものとして注意すべきであろう.

3. ハードウェア構成[2,5,6]

3.1 概要

C.mmp のハードウェア構成は，図1のように*，共有メモリ群 (Mp 1～Mp 16)，プロセッサ群 (Pc 1～Pc 16)，I/O 装置群が，2つのクロスポイント・スイッチ，Smp と Skp により結合された形態をとっている. Smp はプロセッサとメモリ間の結合を，Skp は，プロセッサと各種制御装置との間の結合をつかさどる.



Pc: プロセッサ (PDP-11/20)，Mp: 主記憶 (16 ビット，64 k 語)，Smp: プロセッサー主記憶間スイッチ，Skp: プロセッサー入出力装置間スイッチ，Dmap: アドレス変換ロジック，Kio(Ks, Kf): 入出力制御装置，Kc: K.clock, K.interrupt 用制御装置，K.clock: 60 ビット・クロック，K.interrupt: プロセッサ間割込み制御装置

図 1 C.mmp のハードウェアの構成

各プロセッサ・システムは，専用ローカル・メモリ，2次メモリ，入出力装置などを含むまとまった計算機であるが，それに加えて，プロセッサの仮想アドレスを物理アドレスに変換する，Dmap と呼ばれるハードウェアを持っている. ローカル・メモリは，共用メモリのアクセスを減らすためと，保守など，完全なオフライン独立操作用に用いられる.

図1に示される各構成要素の機能は，つぎのとおりである.

K.clock: 1 μs ごとにカウントされる 60 ビットの情

* 図中の記法は Bell[1] による.

報で，全プロセッサに送られる．測定用のほか，プロセスなどに，ユニーク・ネームを与えるために使われる．

K.interrupt：あるプロセッサから他のプロセッサ群に対して，優先権を持った割込みを発生する機構．

Mp：共用メモリ・モジュールであり，16台まで接続が可能で，1つのモジュールは16ビット語，64kからなり，8ウェイにインタリーブされている．アクセス時間は250 ns，サイクル時間は650 nsである．

Smp：16台のメモリ・モジュールを，16台のプロセッサに接続するもので，もし，全プロセッサが別のモジュールを参照すれば，16台のプロセッサは，同時に働く．スイッチを通過する時間は，往復で200 nsである．（ゆえに，メモリへのスイッチを通してのアクセス時間は，200＋250 ns.）

スイッチは，つぎの4部分からなり，ショット・キー TTL が用いられている．

（1）ポート・インタフェース（320素子）．

（2）優先度決定回路（800素子）．

（3）スイッチ回路（70×24素子）．

（4）メモリ・インタフェース（320素子）．

ここに，相当量のハードウェアが使用されており，しかも集中型であるから，初めからすべてを用意しておかねばならない．ここが，全体のコスト，信頼度，性能を決める重要な部分になっていることに注意したい．

Skp：k本のユニバスのうちの何本かを，P個のプロセッサの1つに接続するスイッチで，この接続は，比較的長い時間単位で行なわれる．ユニバスには，低速 I/O（テレタイプ，カード・リーダなど）用コントローラ Ks，および高速 I/O（ディスク，磁気テープ）用コントローラ Kf が接続される．

Pc：処理要素のプロセッサ．現在は，PDP-11/20を一部修正して使用している．将来は，マイクロ・プログラム制御の PDP-11/40 に交換することを検討している．

3.2 アドレス変換ハードウェア（Dmap）

Dmap は，ユニバスから Smp へのインタフェースで，仮想アドレスを物理アドレスへ変換するハードウェアであるが，具体的には，つぎの2つの機能を実現するために取り入れられた．

（1） PDP-11 の小さなアドレス空間（64 kB）を，共有メモリの大きなアドレス空間（32×64 kB）へマッピングする．人工知能などの応用分野では，非常に大きなアドレス空間が，要求されるからである．

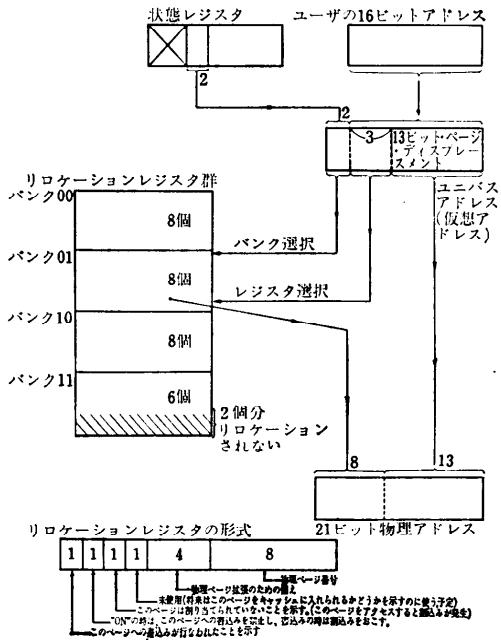


図 2 Dmap によるアドレス・マッピング方式

（2） プロテクションの機能を与える．

なお，Dmapによるオーバヘッドは，50 nsである．

図2は，Dmapによるアドレス・マッピング機構を示したもの．Dmap は，8個のリロケーション・レジスタを4組持ち（実際は30個），8kバイトのブロック単位でリロケーションを行なう．共有メモリの大きさは 2^{21} バイトであるから，アドレス指定に21ビット必要であるが，PDP-11のアドレス指定ビットは16ビットであるため，つぎのようにして変換を行なう．ユニバスは，18ビットまでアドレスを指定できるので，ユーザの16ビット・アドレスに，プログラム状態レジスタ中の2ビットをつなぎ合わせ，18ビットのユニバス・アドレスを作る．この18ビットのうち，上位2ビットは，4バンクあるリロケーション・レジスタの1つのバンクを指定し，つぎの3ビットで，各バンク中の8個のレジスタの1つを選ぶ．そして，選ばれたリロケーション・レジスタ中の8ビットと残りの13ビットで，21ビットの物理アドレスが作られる．ただし，リロケーション・レジスタの31, 32番目が指定された場合は，それぞれローカル・メモリ用およびプロセッサ・システムの I/O 用で，アドレス変換されない．

3.3 プロセッサ間の同期方式

割込みには，16ビットのマスク（各ビットがプロセッサに対応）が使われ，任意のプロセッサ群に対し割り込むことができる．割込みを受ける側のプロセッサは，発信側を直接知ることはできないので，メモリの特定語を読むことによって，相手の確認を行なう．この機構を用いれば，基本的な Lock/Unlock メカニズムを，つぎのように実現することができる．Lock の対象になる情報に対し，2語を割り当て，その第1語目にはセマフォア変数を入れ，第2語目には，そのセマフォア変数を待っているプロセッサ（16台のプロセッサが第2語目の各ビットに対応づけられている．）を記入しておく．セマフォア変数が変更された時には，第2語をマスクとして用いて全プロセッサに割込みをおこし，各プロセッサにセマフォア変数の参照をうながす．要するに，基本的には，連絡情報はメモリにおき，必要な時に，割込みによって他のプロセッサにそれを読みとらせる方式が用いられているわけである．

4. ソフトウェア構成[2,6,7]

マルチ・プロセッサ用の OS は，これまでほとんど扱われたことがなく，とくに，C.mmp のように多様なプロセス関係（たとえば並列，パイプラインなど）をサポートするシステムは，まったく作られていない．この事実とつぎの3点を考慮して，C.mmp の OS には，カーネル・アプローチがとられた．

（1）OS は，多種のサブ OS（TSS, AI用 OS など）のビルディング・ブロックでなければならない．またこのサブ OS は，同時に走れることが必要である．

（2）OS の機能は，分散されなければならない．アロケーションに関し，プロセッサ間にマスタ・スレーブ関係があってはならない．

（3）リソース共用方式は，簡単，かつ保護が完全でなければならない．

カーネル・アプローチとは，OSを“カーネル(核)”と“標準拡張部 (Standard extension)”とに分離する方式で，核は，それを用いていろいろの OS をつくるための OS 機能（メカニズム）の集合であり，標準拡張部は，従来の OS 機能（たとえば，スケジューラとかファイル・システム）に相当する．カーネルは，任意の数の標準拡張部を同時に実行でき，その変更や置換えも，容易にできるように作られる．この方式は，計算処理構造が，あらかじめ準備されたシステムからの制限を受けないという利点がある一方，カーネルの決定が早くなされるため，ある方向への拡張性に欠けたり，多層構造のためにオーバヘッドが増加するなどの危険性がある．しかし，解析とシミュレーションの結果，HYDRA の設計では，それがシステムの性能に大きな問題とならぬことが確認されている．

4.1 HYDRA の設計思想

HYDRA の目的は，先にものべたように，任意の OS を，簡単に，フレクシブルに，効率よく，しかも高信頼性に構成できるメカニズムの集合をつくり上げることであり，この設計に当たっては，つぎの点が留意されている．

（1） マルチ・プロセッサの環境

HYDRA は，限られた時間内に完成しなければならないため，保守的な設計になっているが，ハードウェアの能力の範囲内で，十分フレクシブルに，実験研究ができるように構成されている．

（2） ポリシーとメカニズムの分離

HYDRA のキー・アイデアの1つは，従来の OS と異なって，ポリシーとメカニズムが分離されていることである．ポリシーとは，どんなスケジュール方式をとるかなどといった方式的事項であり，サブシステムが対応する．メカニズムとは，いろいろなポリシーを構成するための，基本的道具である．両者を分離することによって，複雑な決定を高いレベルのシステム設計者にゆだね，システムのフレクシビリティをますことができる．

（3） 実現法

カーネルは，Dijkstra のストラクチャド・プログラミング[8]の手法と，Parnas のモジュール化法*を併用して実現することにし，それを設計に組み込んである．

（4）「厳格な階層構造」の排除

Dijkstra の提案以来，厳格な階層構造の考え方が一般化してきたが，HYDRA では，個別ユーザにとっては，システムは階層構造であるべきだが，グローバルな設計では，かえって高いレベルのユーザにとってのフレクシビリティを制限し，実験に不都合になるとの考えから，厳密な階層構造をとっていない．

（5） 保護機構 (Protection)

HYDRA では，プロテクションは，ユーザに不当な操作をさせないための単なる道具とは考えず，OS 設計の重要な機構としてとらえられている．したがって，プロテクションは，たとえばファイルなどといった特殊なものに対してのみほどこされるのではなく，Capability の考え方[9]を採用して，システムの全要素

* 詳細は不明．

に一様に与えられる.

(6) 信頼性

一般に，ハードウェアは，多重化により信頼性が向上する．ソフトウェアに対しても同様の信頼性を与えようというのが，HYDRA のねらいの1つである．HYDRA の設計に当たり，大切な点は，ハードウェアの異常を，知的に（知的にとは，ダンプをとってリロードするといった単純なものではないことを意味する.）検知し，回復することである．すなわち，悪いプロセッサをただちに離し，その負荷を他に分配するというもので，ハードウェアおよびソフトウェアにおける対称構成の考え方が，その中心となる．

4.2 カーネルの持つメカニズム

以上の要求を満たすには，カーネルとして何を選ぶべきであろうか，

OS の基本的役割は，もともとハードウェアに備わっていないリソースや操作をユーザにサポートする仮想マシンを造り上げることと，物理的および仮想的リソースの管理や割当てを行なうことであるという考えに立って，リソースの抽象化（抽象化されたリソースを，とくにオブジェクトと呼ぶ．この考え方は，ポリシーの分離とともに，HYDRA のキー・アイデアになっている.）を行ない，つぎのようなメカニズムが用意された.

(1) 新しい（仮想）リソースをつくる.

(2) 古いリソースにより，新しいリソースのレプリゼンテーション*をつくる.

(3) リソースや，それらのレプリゼンテーション上でのオペレーションの生成.

(4) リソースとそのレプリゼンテーション上の，許されない操作に対するプロテクション.

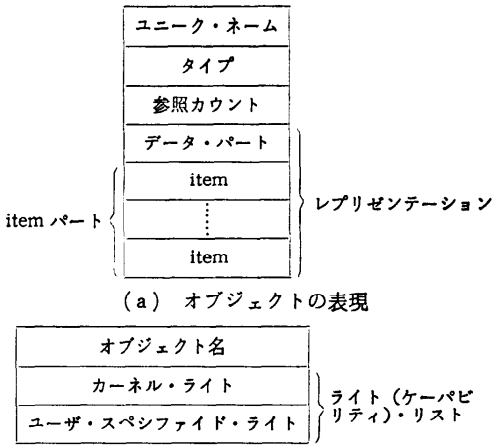
(5) プロセス間通信機構．これはポリシーの範ちゅうであり，当初は予定されていなかったが，外部に出すと時間がかかりすぎるため，カーネルに含められた.

4.3 HYDRA の概要

HYDRA の設計思想は，物理的および仮想的リソースを「オブジェクト」と呼んで一般化することにより実現されている．この節では，オブジェクトの構造と，システムに実行環境を与えるオペレーションについて，その概要をのべる.

4.3.1 オブジェクト

各オブジェクトは，図3のような情報を持つ．この情報は，自分自身に関する情報と，他のオブジェクト上での操作に関する情報に，大きく二分される．各項目の詳細は，つぎのとおりである.



(a) オブジェクトの表現

(b) "item" の内容

図 3 オブジェクトの持つ情報

(1) ユニーク・ネーム：自分のオブジェクト固有の名で，グローバル・クロックを利用して作られる.

(2) タイプ：オブジェクトのクラスを分類する．TYPE という名の特別のオブジェクトにより，新しいタイプが作られる.

(3) 参照カウント：このオブジェクトを参照しているものの数．これが0になると，オブジェクトは抹消される.

(4) データ・パート*：カーネルに解釈されないデータ.

(5) item パート*：item パートは，参照する他のオブジェクトに関する情報を含む，item の集合である．各 item は，図 3(b)のように構造をとり，他のオブジェクトを操作したい時は，そのオブジェクトに対応する item を，パラメータとして用いる．item の情報の詳細は，つぎのとおりである.

(1) オブジェクト名：参照されるオブジェクト名.

(2) ライト・リスト：Capability リスト．つぎの2つの要素からなる.

(a) カーネル・ライト：オブジェクトのタイプによらない権利.

(b) 補助 (auxiliary) ライト：タイプごとに異なる権利.

* 4.3.1 参照.

* データ・パートと item part を合わせて，レプリゼンテーションと呼ばれる.

4.3.2 操作

HYDRA での原始操作は，CALL と呼ばれる．CALL のパラメータは，プロシージャ・オブジェクト名と item のリストであり，item リストは，カーネルによって保護条件が満足されているかどうかを調べるのに用いられる．HYDRA の実行環境に関係するオブジェクト・タイプは，プロシージャ，LNS，プロセスの3つがある．

(1) プロシージャ

プロシージャとは，通常の意味のプロシージャ（あるいはサブ・ルーチン）を抽象化したもので，コードとデータよりなり，パラメータを受け取り，値を返すものである．HYDRA のプロシージャは，保護機構を含んでいるのが特徴で，それが実行される間に参照するオブジェクトへの参照リストを持つ．このリストは，つぎのように，コーラー (caller) に依存するものとしないものに分けることができる．

(a) コーラーに依存しない Capability：このプロシージャをコールした相手に依存しない Capability であり，このプロシージャが常にアクセスするオブジェクトに関するものである．

(b) コーラーに依存する Capability：このプロシージャをコールするものが送ってくるパラメータからつくられる Capability である．

また，プロシージャは，コールする側に依存しない Capability のほかに，item のためのテンプレートを持つことができる．これは，プロシージャが期待するパラメータを特徴づけるもので，与えられるパラメータから実行環境を作る際に使われる．つまり，そのパラメータに対し，どのようなチェックを行なうかを定義するもので，これにより，高度なプロテクションが可能となる．コールする方より，コールされる方が，大きな自由度を持つことになっていることに注意したい．

(2) LNS (Local Name Space)

LNS は，プロシージャがコールされた時に，その実行環境を与えるもので，プロシージャがコールされた時1つ作られ，終了した時消滅する．LNS は，プロシージャの持つコーラーに依存しない Capability と，コーラーに関係する実パラメータをいっしょにして，1つのライト・リストを作り上げたものである．LNS は，作られた時には，対応するプロシージャにもちろん依存しているが，その後はまったく独立で，LNS を変えてもプロシージャは変わらないから，プロシージャは，リエントラントになっているわけである．

あるプロシージャをコールしたい時には，まず，CALL 命令を出す．すると，カーネルは，コーラーからの実パラメータを，プロシージャの持つテンプレートと合わせ，プロテクションをチェックし，不都合がないとわかると，LNS を作り，これを新しい実行環境とし，コントロールを，呼ばれたプロシージャにわたす．その実行が終了すると，カーネルは，再びもとの LNS へもどしてリターンする．なお，LNS は，このほかに，ローカル・ネームをグローバル・ネームにマッピングする働きもする．

(3) プロセス

以上の2つは，C.mmp の並列処理能力とは独立のものである．プロセスは，並列処理に関するもので，外から見れば，実行を独立にスケージュールできる最小の単位であり，内から見れば，CALL がつぎつぎと出されることにより変わる，実行環境変化の記録である．言い換えれば，プロセスは，1つのシーケンシャルに実行されるタスクの状態を累積した，LNS のスタックである．

なお，プロセス間の同期は，セマフォアによっている．プロセスのスケジューリングは，ポリシーに入るが，カーネルは，適当な時間にプロセスを起動させる機械的作業のみを行なう．これに関し，カーネルとポリシーの関係を，図4に示した．

5. むすび

C.mmp は，初めにのべたように，単にマルチ・プ

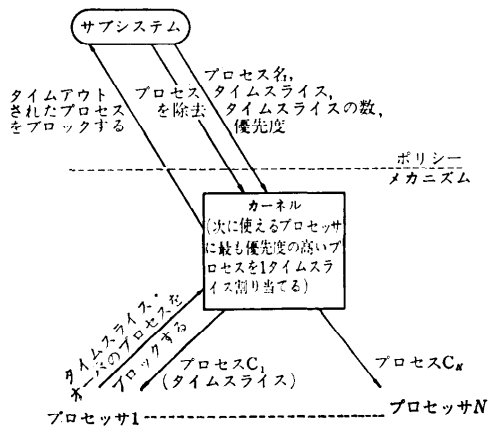


図 4 ポリシーとメカニズムの関係

ロセッサを研究する目的だけでなく，計算パワーへのニーズを満足させる目的を持っている．後者の目的のためには，これが唯一案であったわけではなく，初めは，独立した PDP-11 群とその I/O システムを，PDP-10 のフロントに並べる予定であったのが，現在の形態に発展したものらしい．しかし，この変更は，必ずしも経済性がよいというわけではなく，マルチ・プロセッサが，研究の対象として興味あるものであったためで，経済性については，研究あるいは実用品として十分満足できるものであるという以上の指摘は，各文献にものべられていない．

C.mmp は，ハードウェア，ソフトウェアとも，現在なお製作中のシステムであり，稼動実績といえるほどのデータはないようである．現状についていえば，1973年秋には，3 CPU×4 メモリ・モジュールのシステムが動作しており，スイッチは，16×16 のものがすでに完成している．残りの CPU は，発注はされているが，部品不足で入ってこないとのことであった．今頃には，動き出しているかもしれない．ソフトウェアは，約 30 k 語のカーネルがほぼ動作している．さしあたりの応用としては，従来 PDP-10 で走っていたスピーチ認識システムをのせる努力 (chaos) が急ピッチで進んでいるが，詳細は発表されていない．

システムの今後の拡張，修正としては，つぎのようなことが検討されている．

（1） プロセッサを，マイクロ・プログラム制御のモデル 40 にし，WCS* を付けて，ダイナミック・マイクロ・プログラミングを行なう．

（2） 信号プロセッサなどの特殊プロセッサを付け加える．

（3） 各プロセッサにキャッシュを取り付け，読出しのみのデータを保持する．これは，C.mmp の構成では，スイッチによるオーバヘッドが大きいという問題点を解決するためであるが，シミュレーションでは，512語のキャッシュで，20% 前後の性能向上が得られるという結果が出ている．この改善率は，高速のモデル 40 では，さらに高まるはずである．なお，マルチ・プロセッサのキャッシュでは，共用のデータを書き変えた時の処理が問題であるが，C.mmp では，リロケーション・レジスタ中の 1 ビットで，そのページをキャッシュに入れてよいかどうかを指示させ，読出し専用ページのみをキャッシュに入れる方法をとる予定にしている．

（4） ARPA 網に接続し，遠隔地からもアクセスできるようにする．

C.mmp の構成に対し，いろいろと問題点をあげることもできようが，早い時期にこうしたものを取り上げ，どんどん研究を進めていく馬力には，敬服させられる．最終的に，実用計算機システムとして物になるかどうかはまだ予断を許さないが，万が一そうなり得なかったとしても，その後のこの方面の研究に与えた刺激は非常に大きいし，ここから得られた結果は，つぎのステップで，さらにすぐれた複合システムを構成するのに，大いに役立つはずである．

計算機システム (OS も含めて) は，それを設計した組織体に類似した形態になるといわれる．CMU のハード，ソフト，応用のグループが協力したプロジェクト推進体制が反映されて，よくまとまった複合システムができ上がるよう希望しよう．

* Writable Control Storage (書込み可能制御記憶)

参考文献

1) C. G. Bell and A. Newell: The PMS and ISP descriptive systems for computer structures, Proc. of SJCC, Vol. 36, pp. 351～374 (1970).
2) W. A. Wulf and C. G. Bell: C.mmp——A multi-mini-processor, Proc. of FJCC, Vol. 39, pp. 765～777 (1972).
3) C. G. Bell and P. Freeman: C.ai——A computer architecture for AI research, ibid., pp. 779～790.
4) 飯塚ほか: 新しい計算機システムの調査と評価, 2.5節, 電総研調査報告, 第 179 号 (1974).
5) C. G. Bell, et al.: C.mmp; The CMU multi-miniprocessor computer, Requirements and overview of the initial design, CMU Department of computer science report, CMU-CS-72-112 (1971).
6) V. R. Lesser: Lecture notes on C.mmp, 電子工業振興協会 (1973).
7) W. A. Wulf, et al.: HYDRA; The kernel of a multiprocessor operating system, CMU Department of computer science report (1973).
8) E. W. Dijkstra: Structured programming, Academic Press (1972).
9) B. W. Lampson: Dynamic protection structures, Proc. of FJCC, Vol. 35, pp. 27～38 (1969).

(昭和 49 年 3 月 25 日受付)