キャッシュ・メモリ・システム*（2）

飯　塚　　肇**

第1部***においてはキャッシュ・メモリ・システムに関する基本的事項，すなわち，その意義，マッピング方式，ストア方式，入れ換えアルゴリズム等を中心に述べた．本稿では，パラメータの最適値，パフォーマンスの測定，解析，および実現上の諸問題についてまとめ，終りに将来への展望を述べて結びとする．

5. パラメータの決定

本章では現用されているキャッシュ・システムで採用されている各パラメータ値について述べる．それらは主としてIBMによる大規模なシミュレーション解析の結果[5),8)]定まったものであるが，Sisson[9)]等も小規模に行なっており，これらはMeade[19)]によってよく整理されている．この他，ユニバック[28)]，日立[11)]等の結果やMeadeによる新しい報告[32)]もあるが，ここでは最もまとまっているMeadeの前の文献を利用し，その他で補うことにする．

5.1 マッピング方式

実験的なものを除けば現用されているのは，セクタ方式と$M=2$または4のセットアソシアティブ方式だけである．（付録参照．）これらの方式が基本的に適当であることは2章に述べた通りであるが，両者の間ではセットアソシアティブ形の方が通常，コスト/パフォーマンスがよいとされ，最近多く利用されている．

しかし，大形プロジェクトの計算機のように，他の条件の影響で総合的に見て，セクタ方式が有利と判断された例もある．すなわち，この計算機は仮想記憶方式（主記憶—磁気ドラム間）を用いているために，そのアドレス変換機構として，アソシアティブメモリがすでに使用されており，キャッシュ用にもアソシアティブメモリを用いる時，両者をまとめられるのでコストが単独の場合のように増加せず，全体として有利になるといわれる[33)]．

5.2 ストア方式

データ書込みの際のストアの方式はScarrot[3)]などのキャッシュシステムの原形ではスワップ方式が想定される場合が多かったが，2章に述べたように，データの一致を重視して，現在の商用機では全て，直接法が採用されている．（付録参照．）

しかしながら，シミュレーションや解析[26)]の結果，パフォーマンス的にはスワップ方式の方がよいことがわかっており，ミニコンピュータに実験的にキャッシュを付加したシステムでスワップ方式を用いた例[31)]も，最近あらわれている．

5.3 入れ換えアルゴリズム

第1部4章に詳述した通りであるが，要点をくり返すと，現在キャッシュメモリシステムで採用されているのはほとんど“最後にアクセスされてから最も時間の経過したものを追い出す方式”であり，他にはハードウェアとしてラッチ1個しか用いないIBM 370/155の方式があるのみである．この考え方が現在のところコスト/パフォーマンス的に最も妥当であることは，多数の研究結果からある程度立証されている．

5.4 ブロック長

それぞれのシミュレーションの場合によって，他の条件がまちまちで，正規化されたデータを得ることは難かしいが，（このことは以下の他のパラメータについても同様である．）“よさ”の目安として，キャッシュ内にデータの発見される確率（あるいはその逆の発見されない確率であるミス率）をとると，例えば図5のような結果が得られ，おおむね次のような結論がひき出される．

容量Cを一定にしておいて，ブロック長を大きくしていくと，ミス率は始め下降し，最小値に達した後，その後急激に上昇する．この上昇はキャッシュ中のブロック数，Nが小さくなりすぎ，入れ換えが頻ぱんに起るようになるためである．（CでなくNを一定にすれば図5の線5にあるようにミス率は0に向っていくらでも下る．）ミス率の最低が起るのは条件によって異なるが，8 kB～16 kBの容量キャッシュに対し，16～256バイトの時である．

このことだけからみると，ブロックはかなり大きい

---

* A Survey of Cache Meemory System Part II, by Hajime Iizuka (Electrotechnical Laboratory, Electronic Computer Division)

** 電子技術総合研究所電子計算機部

*** 情報処理学会誌 Vol. 13, No. 7, pp. 467～473, 1972

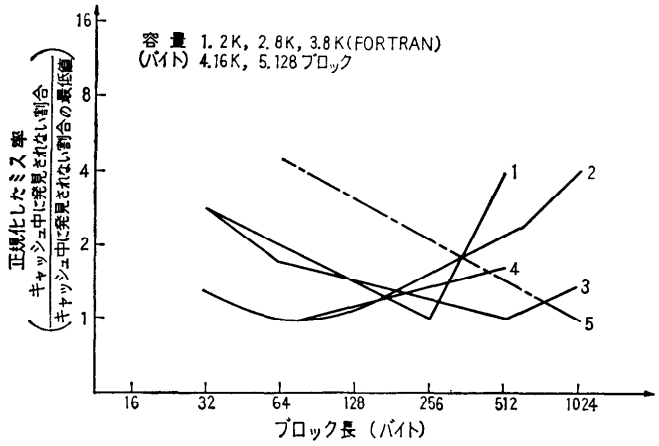

図 5 ブロック量とミス率の関係[19)]

Fig. 5 Relation between block length and miss rates

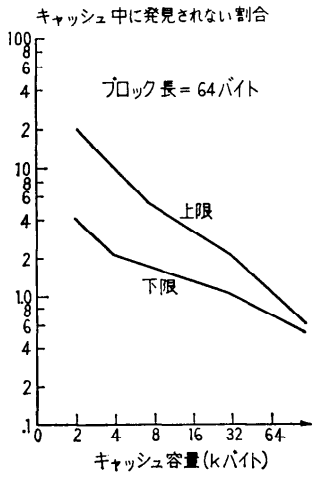

図 6 キャッシュ容量とミス率[19)]

Fig. 6 Effects of cahe capcity upon mis rates

方がよいように思われるが，以上の状況はブロック長が大きい時のブロック転送時間の増大や，サーチ時間等の考慮をせず，単に，一度転送したブロックの内容が後にどれだけ使われるかということにしか注目していないことに注意しなければならない．したがって，ブロックの転送時間がブロック長に比例し，またサーチ時間が対象となるブロック数によらないとすれば，ブロック長は逆に小さいほどよいという結論にさえなる．

現実にはセクタ形ではブロック長を 1 kB 程度にとり，1回の転送単位を 32～64 バイトにとって，8個以上のブロックがキャッシュに入るようにしており，また，サーチ時間が0かまたは非常に小い直接形やセットアソシアティブ形では，同時に転送してこれる最大ブロック長，(主としてインタリービング数とメモリ読出幅で決る.)以下で，かつ上記ミス率が最低に近い値を採用することになる．すなわち，64，または32バイトが一般に適当であるとされている．

5.5 容量

キャッシュの容量はミス率，ひいては実効メモリ速度を決める第一の要因であるが，コストの最大要因でもある．プログラムのトレースによって容量とミス率の関係を求めた結果の分布限界を図6に示す[19)]．これからわかるように，容量を次第に増していくと，ミス率は減るが，その割合は減じ，容量を大きくした効果はうすくなっていき，プログラムによる違いも次第になくなる．

一般に，キャッシュの効果は主記憶の容量にあまりよらないので，大容量の主記憶を持つシステムに対してより有効である．実際にそれぞれの場合どの程度の容量にするかは，コスト，性能比で決るが，以上のことから，ある容量のところにその最大値があることは明らかである．その位置はパフォーマンスの評価法とキャッシュメモリの値段によって変り，大容量に向って動いていっているといえよう．現在のところは 8 kB～32 kB が使用されている．(付録参照.)

5.6 速度比

キャッシュと主記憶の速度，また，それらの比をどのくらいに取るかは，ねらうパフォーマンスとコストの関係で定まる．パフォーマンスを上げるには，キャッシュの速度を上げるのがもっとも手っ取り早いが，その速度は普通，CPU のマシンサイクルに適合した値に選ばねばならない．

今，キャッシュのサイクル時間を t_c，主記憶との速度比を β，キャッシュ内にデータのある確率を P とすれば極めて大ざっぱに実効サイクル時間は

$$t_{eff}=t_cP+(1-P)\beta t_c=\{P+(1-P)\beta\}t_c \quad (1)$$

で表わされる．これから t_c は実効速度にじかにきくが，β には $(1-P)$ がかかっているため，キャッシュ容量が大きく，ミス率が小さい時はその影響は小さい．一般的にいって，t_c を一定とした時，β を大にすることはメモリコストガ下がることであって，β を大にしても実効速度があまり影響を受けなくなるところに，キャッシュ方式正当化の一つの根拠がある．普通は $\beta=10$ 前後 $t_c=50$～120 ns の値が採用され，素子と

してはキャッシュにモノリシックの IC メモリ，主記憶にはコアメモリが用いられる．

今，(1) において，$P=0.95$, $\beta=10$ とおくと，$t_{eff}=1.45$ となるが，実際には，ストア方式や，ストアの比率等の検討も必要であって，t_{eff} は2前後と考えられる場合が多い．実効サイクル時間のもう少し詳しい計算結果は(26)に報告されている．

6. 評価と測定

キャッシュシステムの性能の評価，測定には次の2つの面がある．

① キャッシュシステムとこれを採用しない場合の総合的比較．

② パラメータ値，方式等の手法による差の検討．

①の場合はパフォーマンスそのものより，コスト/パフォーマンスを求め，考えているシステムにキャッシュを採用するのが得策かどうかを決定することを目的としている．したがって，ここでの性能とは保守性や従来機との互換性等，直接数字で表れないものも含まれた極めて広義のものを指すことになる．単なるパラメータや手法による性能差を調べようという②よりは①の方がより重要な問題には違いないが．それに対するまとまったアプローチはなく，本稿の範囲外のことが多いので，ここでは述べないこととし，①の判断の重要な基礎データとなる②の問題についての結果を以下にまとめる．

選定したパラメータ，手法に対してそのパフォーマンスを推定することの必要は大きく，よいキャッシュシステムを設計できるかどうかはこの評価によるところが多い．まず評価基準のとり方が問題であるが次のようなものが使用されている．

(1) 主記憶全体がキャッシュと同一メモリで成り立つとした場合に対し，どれだけパフォーマンス損失があるかで示すもの．

(2) メモリシステムを切離して考え，1レベルとした時に実効的にどれだけの能力を持つメモリと見なせるかという実効メモリ速度で表わすもの．

(3) ミス率またはキャッシュ内にデータの存在する確率のみで表わすもの．

パフォーマンスを直かに評価基準にとっている点，(1)が最もよいと考えられるが，パフォーマンスとして，何をとっているのか，定量的なもので明示されない場合が多い．その点(2)はシステムパフォーマンスではないが，定量的でわかり易い．しかし，実効速度の定義の仕方にいろいろ考えられる．(3)は測定が容易であるが，メモリ速度比が異なれば，同じミス率でもパフォーマンスは変ってくる．中間データとして使用されるべきであろう．

また，これまで得られている結果はこのような表現形式の違いの他，それぞれの周囲条件がまちまちなものである．特にプログラムのアドレスパターンの違いは性能に大きな差を与えることがわかっており，最終的結論，特に①の問題への適用にあたっては，じゅうぶんな注意が必要である．

一方，評価の手法としては次のようなものが考えられる．

(1) パイロットモデルを作製し，実験を行なうもの．

(2) シミュレーションによるもの．

(a) プログラムのトレースやハードウェアモニタ等によって得られたデータをもとに実際と同じモデルを作って，逐一シミュレーションを行なうもの[5),8),19)]．

(b) 計算機およびアドレス発生機構をモデル化し，一般のシミュレーション言語を用いてシミュレーションを行なうもの[29),35)]．

(3) キャッシュメモリシステムをモデル化し，公表されたデータをもとに解析的にパフォーマンスを求めるもの[19),26)]．

厳密にパフォーマンスを評価するには (1) または (2) a を多数のプログラムについて，実行するしかないが，(1)は通常困難であり，IBM[5),8)]，日立[11)]，等では(2)aを用いている．この方法は精度のよいデータが得られるかわり，1個のプログラムのシミュレーションに，実際の時間の数十倍の時間を要するし，もととなるアドレスデータを取るのにも，ソフトウェアトレースでは同様の時間が必要であって，(ユニバックでは専用のハードウェアモニタを用意しキャッシュシミュレーションも直ちにできるシステムを持っている[28)].) 時間，費用共大きくかかるのが難点である．また，対象システムの構造を変える場合やマルチプログラミングの状況等タイミングををおり込んだシミュレーションはむずかしいと考えられる．

一方，(2) b は同じシミュレーションでもすでにある言語を利用しているので，統計機能もそろっているし，プログラム開発時間も短くてすむ．したがって，パラメータ変更も容易にでき，いろいろの場合のシミュレーションが簡単であるが，逆に細部にまで渡るシミュレーションが難かしいこと，アドレスパターンの

発生を高速にかつ，うまく（現実に近く）行なう方法がないので精度を高められない欠点がある．

（3）は費用はかからないが，複雑なシステムを式にのせるのは困難なので，いかに簡単化するかが大きな問題になる．精度も上らないが発表されている諸データから，おおよその推定をするのには便利である．

以下にそれぞれの例をあげておくが，条件は基準化されていないので，相互に比較することはあまり意味がない．

6.1 パフォーマンス測定例

（1） 厳密なシミュレーション

一例として IBM 360 モデル 85 の場合のデータ[8]を図7に示す．なおこの計算機では $C=16$ kB, $\beta=13$ である．

（2） シミュレーション言語によるもの．

筆者と照井による GPSS を使用したシミュレーション等があり，ワーキングセット*を利用したアドレス発生法が試みられている[35]が，ここでは省く．

（3） 解析的な方法によるもの．

Meade[19] および筆者[26]によるものをそれぞれ図8, 9に示す．できるだけ近くなるように書きなおしてあるが，細かい条件は異なるので詳細は原論文を参照されたい．

6.2 コスト/パフォーマンス

6.1 ではパフォーマンスを単独に考えたが実際の設計においては，コストとの対比で考慮されなければならない．主記憶およびキャッシュの価格に対し，仮定をおけば，パフォーマンスの測定結果から，コスト/パフォーマンス比は容易に計算できるが，ハードウェアのコストはたえず変化しているので，一般的結果を得ることはかなり難かしい．実際 Meade の計算例[19]があるが，評価基準のとり方が明確でなく，また，価格の仮定も妥当かどうか問題があるので，ここではこれ以上ふれない．

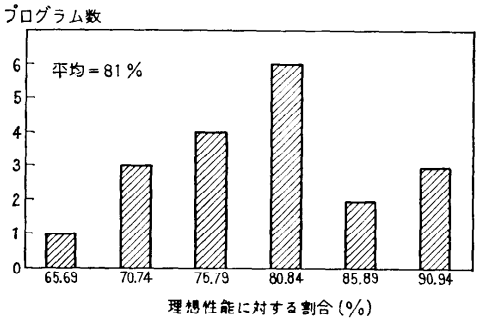

図 7 IBM 360 モデル 85 のパーフォーマンス[8]

Fig. 9 IBM 360 model 85 performance relative to single-level strage operationg at cahe speed

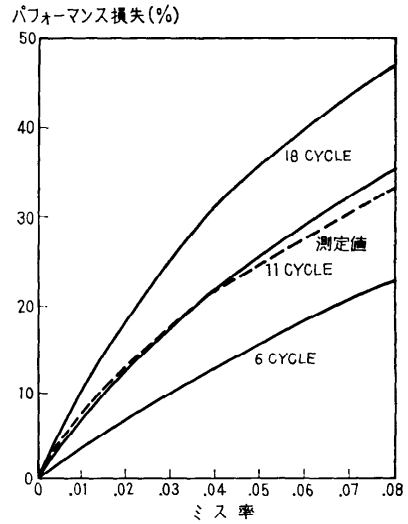

図 8 パフォーマンス計算例[19]

Fig. 8 An example of performance calculation (パラメータは主記憶のサイクル数)

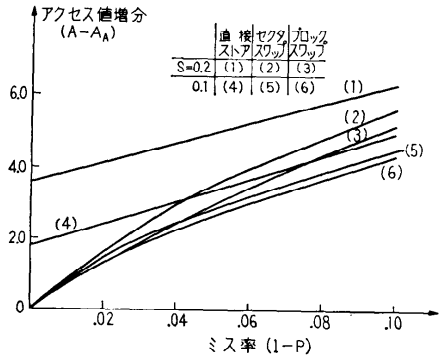

図 9 パフォーマンス計算例[26]

Fig. 9 An example of performance calculation

---

* ワーキングセットの概念はページングにおけるプログラムの状態を記述するために Denning[14] によって導入された．時刻 t におけるワーキングセット $w(t,\tau)$ は $(t-\tau, t)$ の間に参照された情報の集合と定義され，通常その要素はページである．

6.3 キャッシュの使用効率

あまり測定されていないが，キャッシュがどのくらいの割合で有効なデータを保持しているかを知ることは興味ある問題である．

ブロック形で直接ストアの場合は初期の過渡状態を除けば一応主記憶と同一のデータがキャッシュの全てのブロックに入っている．（サブブロックを用いる場合はブロック形といっても必ずしもそうはならない．また，シミュレーション結果では，ブロック形で完全にキャッシュがロードされるまでにはかなりの長い時間がかかり，プログラムのスイッチまでに100% に達しない場合もある．）しかし，その他の場合は常に全ブロックの一致がとれているわけではなく．特にセクタ形ではあるセクタがキャッシュ中に割り当てられている間に一度も主記憶からの転送を受けないサブブロックも多い．セクタ形でスワップ方式を用いる場合には，セクタの割り当てを変える際に，どれだけのサブブロックについて主記憶を書きかえる必要があるかを評価するために使用効率は重要である．

この値は 25%（16 kB—16 ブロック）程度という測定結果があるといわれるが[36]，詳しいデータは発表されていない．しかしあまり大きくならないのは確かである．

6.4 ブロック寿命

特定のブロックがキャッシュに割当てられてから，どのくらいキャッシュ中に滞留するかもあまり測定されていない．しかし，後述するプログラムスイッチングの状況でのキャッシュ効率の判断等には重要なデータである．

これまでのところ，この値はプログラムに依存することが大きく，また簡単なシミュレーションの結果でも，バラツキが非常に大きいことがある程度わかっている．更に平均値を計算することもできるが[26]，ここでは省く．

7. 実現上の諸問題

キャッシュシステム実現にはいろいろ細かい問題があるが，ここではこれまでの検討の補足として，マルチプロセッサとマルチプログラミングにおける問題点について若干付け加える．

7.1 マルチプロセッサ

キャッシュメモリは高速アクセスのために，通常 CPU 内に配置され，記憶装置におかれることはまずない．したがって，プロセッサが複数台あって，主記憶を共用するマルチプロセッサシステムではキャッシュはそれぞれの CPU 内に配置されるので，ストアに対し，たとえ直接法を用いて，主記憶を直ちに書きかえても，同時にその語が他の CPU キャッシュに出ているかどうかを調べ，もしあれば書き換えを行なうか，有効ビットを OFF にしなければならない．この操作は現在のように主記憶を基本に考えて，キャッシュを写しとするシステムでは避けることのできないものであるが，ハードウェアもいるかなりやっかいな操作を各ストアごとに行なわねばならず，他の CPU のキャッシュアクセスのぼう害になることもある．にもかかわらず，主記憶を共用していても，同時刻に同一のエリアをアクセス使用するケースは現在一般に用いられているマルチプロセッサシステムでは少なく，これはあまり実効がない操作である．また，この他にも，特殊命令を用意しソフトウェアの協力を得ることも考えられるが，キャッシュがプログラムに見えるようになってしまう．

結局，既存のアーキテクチャを背景に単一プロセッサをねらって考え出された現在のキャッシュの考え方を取る限り，キャッシュ方式はマルチプロセッサにはあまり適していないといわねばならない．

実際例では IBM はアルチプロセッサに使用した例を報告していないが，国内では大型プロジェクト計算機のデュアルプロセッサにキャッシュがいち早く採用されている[37]．この計算機では各 CPU に対応して主記憶 4 kB ごとにその内容がキャッシュに出ているかどうかを示す表示子を用意して，大部分のアクセスでキャッシュの直接チェックを省くという技術[23]を使って，前記問題の解決がはかられている．しかしながら，残念なことに，この技術の効果を含めて，採用にあたってこの検討結果はほとんど発表されていないし，実際の結果もまだ今後の測定にまたねばならない．

なお，この他 CMU* にもルマチプロセッサとキャッシュを組合せた計算機計画[30]があるが，一つのデータが多数のキャッシュに出ている問題をどう解決するであろうか．

7.2 プログラムスイッチング

前に指摘したように，キャッシュシステムの有効性にはアドレスパターンのローカル性が重要な鍵となっている．このことは手頃なτに対し，ワーキングセット**，W，の変化が非常にゆるやかなことを意味し，

---

* Carnegie-Mellon University

** p. 543 の脚注参照

それは単独のプログラム走行中には必ず成立している．しかしながら，プログラムのスイッチが起こった時は W は当然大きく変化するから，キャッシュ中のデータは通常，無効になるものと考えねばならない．しからば，一般に行なわれているマルチプログラミングの環境において，特に，TSS やリアルタイムシステムのようにプログラムスイッチが頻繁な場合にはキャッシュシステムの効率が相当悪化するのではないかという疑問が当然生れてこよう．

この問題に関して，まとまった報告は今のところない．また，今まで発表されたデータもモノプログラムの場合が主としてあげられ，他からの干渉は無視され，有効性がやや誇張されていた感がないでもないが，これまでの結果を総合してみると，効率の多少の劣化はあっても，通常の場合その応用の特殊性のために，キャッシュ採用が不適になる場合はまずないと思われる．

Meade はこの問題に対し，次の3つの要因をあげて弁護している[32]．①キャッシュ内のスーパーヴァイザプログラムの部分は共通である．②プログラムがスタートする時に必要なブロックはあまり多くない．③キャッシュを満すに必要な時間はプログラムの実行時間に比べてじゅうぶん短い．(2% 程度)

キャッシュシステムにおいて，プログラムの中断後の再開時に，前のデータの一部がキャッシュ中に残っていることはページングの場合と異なり，あまり期待できないが，③の主張はブロック寿命についての解析[26]やシミュレーションの結果[29),35]とも一致している．

8. 拡張と変形

この章では普通と異なる領域へのキャッシュの適用や，その機能の拡張等，新しいいくつかの話題について私見を混ぜて述べる．

8.1 機能の拡張

IC メモリの最大の利点はその高速性にあることはいうまでもない．キャッシュは正にこの点を利用しているわけである．しかし，キャッシュはまだ IC メモリのもう一つの特長である論理操作との組合せの容易さという点を，じゅうぶん利用しているとはいえない．これに対し，キャッシュメモリにサーチ，ソート等の機能を付加し，主記憶上のデータに対し，みかけ上高速に複雑な論理操作を施せるようにすることが当然考えられる．

IC メモリに論理機能を持たせたものは一般にロジック付きメモリ[38] (Logic-In-Memory) と呼ばれ，主として SRI で研究が行なわれているが，キャッシュの LIM 化の考えは Stone によって提案されている[17]．このアイデアのセクタ形キャッシュに LIM を使おうというものであって，発表されている検討結果はまだプリミティブなものだけである．しかし，この拡張かどうか明確でないが，大規模なキャッシュをベースにした計算機がスタンフォード大で開発中といわれる．

この LIM キャッシュの制御にマイクロプログラミングを取入れ，更に，整理すれば今後の計算機ストラクチャとして非常に有望であるように思われる．

8.2 ミニコンピュータへの適用

中形機以下ではキャッシュはあまり有効でないと一般に信じられており，このクラスではむしろ全 IC メモリになる傾向の方が強い．(例えば IBM 370/145, 135.) その主な理由はここでは大容量の主記憶が実装されることが少いからであるが，PDP-8 のような小さな計算機ですらキャッシュによってコストパフォーマンスをあげられるというシミュレーション結果[11]も最近現れている．しかしながら，コストの中には保守や調整の容易さ等間接的なものも含めて判断されねばならないので，この結果だけからミニコンピュータにもキャッシュが有効で，直ちに取り入れるべきであるとするのは性急に過ぎよう．

けれども，ミニコンピュータの処理能力の向上にともなって，大容量のメモリが必要になる場合も次第に増加してきており，ミニコンピュータをネットワークにして多数のメモリモジュールを共用しようとするようなシステムでは CPU とメモリ間にあるスイッチによる遅延を避けるため，キャッシュが有効になる．

8.3 マイクロプログラミング

マイクロプログラミングとキャッシュを結びつける方法は2通りある．一つは8.1に述べたマイクロプログラミングによる拡張形キャッシュの制御であるが，もう一つ，マイクロプログラムの記憶部にキャッシュを適用するという使い方がある．

通常マイクロプログラムは高速の固定記憶装置に入れるので，普通の場合キャッシュによる速度の利得はあまり大きくないが，これによって，マイクロプログラムを容易に大容量化でき，また，書きかえによる可変構造性を持たせられる利点がある．CMU のプロジェクト[30]では主記憶に普通のプログラムとマイクロプログラムを共存させ，キャッシュを別々におく方法が

採用されている．このように主記憶に両者を共存させると，柔軟性はあるが，その管理が複雑になったり，マイクロ命令とマクロ命令の語長を一致させねばならなくなる．

一方，筆者等が検討した[39]ものでは，マイクロプログラム記憶部は大容量の固定記憶または一部書込可能記憶とし，ここに各種の問題向きルーチンを用意して，これをキャッシュを持つ多数の CPU で共用する．一つのプログラムを走らせている時使用されるマイクロルーチンはかなり限られていると考えられるので，あまり大きなキャッシュでなくても効率をあげられよう．また，マイクロプログラムではストアがない（ワークメモリはそれぞれの CPU で持つ．）ので一般のマルチプロセッサのような問題も生じないですむ．

ここでどちらがよいと主張するつもりはないが，問題向き計算機の必要性が増してきた現在，マイクロプログラミングの大容量化のためにキャッシュは有効な手段である．

8.4 むすび

近頃の LSI 技術の発展はすばらしい．特に単純な構造のメモリは LSI の最も適した分野であって，今後，低価格化が期待される．一方，LSI 向きの計算機システムといわれるものが多方面で研究されているが，これぞというものはない．むしろキャッシュのようなところから大形機の LSI の利用は進んで行くだろう．

これまであげたのは今後考えられる 2,3 の拡張例にすぎないが，この他，中間にもう 1 レベル，キャッシュを入れて 3 段にするという考えも提案されている[19]．更に，一般の仮想記憶等と統一的に考え，システム全体としての階層とアドレス空間の構造が整理されていくのではなかろうか．

いわゆるリニアなアドレス空間でない多次元のアドレス空間やツリーやリストのアドレス空間等をとりあげた時のキャッシュの構造等，まだいろいろの問題が残っているように思われる．

謝 辞

本稿をまとめるにあたって，多数の文献を利用させていただいた．それらの著者に感謝すると共に，当然引用すべきで落したものの著者に筆者の不勉強をおわびする．

慶応大学工学部相磯秀夫教授は原稿を読んで，いろいろ有益な御意見をよせられた．深く感謝する．また，日頃，御指導いただき，このまとめを行なう機会を与えられた，黒川一夫電子計算機部長と前部長野田克彦博士に謝意を表する．なお，投稿にあたってパターン部淵一博室長の御世話になった．

参考文献

第 1 部にあげなかったもので，本稿で引用したものを以下に記す．

33) 小高：電子協，新機種動向調査委員会，討論，1971.
34) P. J. Denning: "The working set model for program behavior," CACM, Vol. 11, No. 5, pp. 323-333 (1972).
35) 飯塚，照井："GPSS によるアドレス発生とキャッシュメモリのシミュレーション実験" 情報処理学会夏のシンポジウム"システム評価"(1972)
36) 相磯：私信．
37) 大型プロジェクト関係，諸資料，たとえば，西野："超高性能電子計算機" 日立評論．Vol. 54, No. 3, pp. 49-51 (1972).
38) たとえば，W. H. Kautz: "Celluar logic-in-memory arrays," IEEE Trans. C-18, No. 8, pp. 719-727, (1969).
39) 飯塚，相磯：ETL 内部資料，(1970) 未発表．

付 録

現在，実用に供されているキャッシュ方式の主な計算機の特性表をかかげる．ただし，出典が異なるので，表現の方法が少し違っている場合もある．

付 2　主なキャッシュシステム

モデル	IBM 360/85	IBM 360/195	IBM 370/155	IBM 370/165	大形プロジェクト	H-8700
主記憶装置						
アクセス時間 (nsec)	880		1,490 (4) 1,610 (8) 1,960 (16)	1,440 (8)		
サイクル時間 (nsec)	960/1,040	756	2,100	2,000	600	900
インタリービング (ウエイ)	2/4	8/16	4	4	8	4
最小容量 (kバイト)	512	1,024	256	512	2,048	512
最大容量 (kバイト)	4,096	4,096	2,048	3,072	8,196	2,048
読出し幅 (バイト/バンク)	16	8	4	8	8	8
キャッシュ					MOS-LSI	ワイア
アクセス時間 (nsec)		162	230 (4) 345 (8)	160 (8) 240 (16)		
サイクル時間 (nsec)	80	54	115	80	100	
容量 (kバイト)	16/24/32	32	8	8/16	16/32	16
読出し幅 (バイト)			4	8	8	
マッピング	セクタ	セットアソシアティブ	セットアソシアティブ	セットアソシアティブ	セクタ	セクタ
ブロック長 (Mブロック数)	1,024 (16/24/32)	64 (4)	32 (2)	32 (4)	1,024 (16/32)	1,024 (16)
ヴァリディティユニット (バイト)	64	64	16	32	64	64
ストア方式	直接	直接	直接	直接	直接	
置換アルゴリズム	アクティヴィティリスト	アクティヴィティリスト	一種のランダム (全体に一個のラッチ)	アクティヴィティリスト	アクティヴィティリスト	

(昭和 47 年 4 月 8 日　受付)