キャッシュ・メモリ・システム* (1)

飯　塚　　肇**

キャッシュ・メモリシステムは現代の計算機システム技術の重要なものの一つである．本稿ではこれまでの研究結果を2部にわけて，統一的に整理，解説する．まず，この第1部では，キャッシュ・システムの歴史をたどり，背景となる技術，アイデアの基本を考える．特に，キャッシュと主記憶間のマッピング方式，ストア方式，キャッシュ・エリアの入れ換えアルゴリズムの三点を中心に検討し，マッピング方式については新しい観点で，分類，整理を行なった．第2部においてはキャッシュ・システム実現上の具体的な諸問題，シミュレーション，解析の結果等について述べ，将来への展望をもって，全体のまとめとする．

1. はしがき

最近，中容量の IC メモリを大容量の主記憶の写しとして使用し，記憶システムの全体としての速度を高める方法が一般化してきている．これはアイデアとしてはコアメモリと IC メモリ間に存在する価格・性能比の差とアドレスパターンのローカル性の2つを基礎に成立していて，最近利用されたデータの写しを高速記憶中に残しておいて，以後そのデータのアクセスは高速記憶から行ない，主記憶のアクセス頻度を減らして実効速度を向上させようというものである．この方法は通常，スレーブメモリ[2]，バッファメモリ，キャッシュ(Cache***)[8]，ルック・アサイドメモリ(Look-aside memory)[14] 等と呼ばれているが，第二，第三の用語が用いられることが多い．バッファメモリはやや一般名詞的で別の意味に用いられることもあるので，ここではキャッシュと呼んでおくことにする．

1.1 歴　史

記憶装置に階層構造 (Hierarchy) を持たせることは第一世代の計算機以来行なわれてきたことであるが，低速大容量のいわゆる外部記憶（磁気ドラム，磁気テープ等）は命令によって指定されるアドレス空間にはなく，いわゆる I/O として，取りあつかわれていた．

これに対して，1960年代の初め，外部記憶（ドラム）にも主記憶同様にアドレスづけをし，これをブロックに分割し（ページ化），アクセスされたデータを含むページを高速記憶（コア）に取出し，以後のアクセスを高速化するという考えが生れた．これは1レベル記憶と呼ばれ，目的において若干キャッシュ方式とは異なる点もあるが，初めにのべた基本となる考えは同一で*，その実現例としては，1963年，マンチェスタ大学の Atlas[1] が有名である．

一方このころ，ローカルに（複数個所）高速小容量記憶を設け，実質的に記憶制限 (Memory limited) の状態を打解しようという試みも ETLMK-6[15] 等で行なわれた．ついで，仮想記憶のアイデアを主記憶に適用し，高速記憶と低速記憶の入れかえを自動的に行なって，見かけ上高速記憶に近い速度で，低速記憶による主記憶の大容量化を実現しようとする考えが1965年頃からあらわれている．これは MK-6 流の考え方からすれば，ローカルの高速記憶を整理して一個所に集中したものと見ることもできる．

この年 Wilkes は直接法（2参照）マッピング方式のキャッシュを提案して，これをスレーブメモリと呼んでおり[2]，同年，Scarrott[3] も同様の考えを発表しているが，この場合高速記憶はわずか16語程度しか想定されていない．一方，Wilkes はスレーブメモリの大量化に言及しているが，スレーブとして使われるのが一般の高速コアで，本籍の方がいわゆる低速大容量コアであり，TSS のページ方式にかわるものとして考えられている．すなわち，主記憶とスレーブ間のスワップ頻度をへらすことを目的としていて，一般のキャッシュ方式とは，目的において若干異なっていた．

こうした歴史を背景にして，現今のキャッシュシス

---

* A Survey of Cache Memory System, by Hajime Iizuka (Eletrotechnical Laboratory, Electronic Computer Division)

** 電子技術総合研究所電子計算機部

*** 本来は貯蔵所，隠し場所の意，探険家が食料等の必要品を隠しておく場所のことをいう．

* いずれもプログラムが用いる論理アドレス空間の構造をハードウェアに依存する物理的アドレス空間から独立させ，その構造に論理的抽象性を与えようとするものである．この観点からは仮想記憶[22,33] (Virtual Memory) といわれている．キャッシュは仮想記憶のサブクラスと考えてもよい．

テムに対するトリガとなったのは Gibson の論文[5] である．ここでは，まだキャッシュということばは現れず，主記憶と高速記憶間のブロック転送方式に重点をおいているが，恐らく，IBM が実際の計算機にキャッシュ方式を採用するために行なった基礎的検討の成果を発表したものであろう．

次いで，1968年 IBM は史上初めてキャッシュメモリを採用した大型計算機 360/85 を発表し，内外の計算機関係者に大きなショックを与えたが，それは多くの計算機関係者はこの時点で 8KB とか 16KB の大容量 IC メモリが採算のとれるものと考えていなかったことによる．IBM 360/85 のキャッシュについては文献 6, 7, 8) に詳しい．

その後，同社は 360/195[12,18]，370/155[20]，165[21] と次々にキャッシュつきの計算機を発表し，近頃では，大型機では，キャッシュつきがむしろ一般化しつつある．

キャッシュが大型機の記憶を全部 IC にきりかえるまでの過渡的なものか，半永久的なものかは議論のあるところであるが，現在の記憶素子の階層性が完全にくつがえされるようなことは，ここ当分考えられず，個々の素子の改良によって，全体の階層構造はそのままで，性能がシフトすると見られるから，少くとも70年代前半，大容量の記憶を持つ大型機ではキャッシュ方式が広く使われるとするのが妥当であろう．

2. マッピング方式

キャッシュの形式を大きく定めるものの一つに，主記憶とキャッシュの対応関係をきめるマッピング方式がある．普通は大きく 4 種類に分けているが[10]，ここではセクタ形とブロック形の 2 つに整理して考えてみよう．

まず用語の定義をする．バイトとはメモリアドレスによって位置を指定される (addressable) 最小の情報単位で，通常 8 ビットからなる．キャッシュと主記憶を対応させる連続したバイト数をブロックと呼び*，その大きさを $B=2^b$ とする．また，主記憶とキャッシュ間の一回の転送量を転送ユニットと呼び，そのサイズを $T=2^t$ とする．キャッシュの容量を $C=2^c$ とし $C/B \equiv N=2^n$ (ブロック数) とおく．また，主記憶のそれぞれのブロックが占めることのできるキャッシュ中のブロックの数をマッピングブロック数というこ

* 統一するために，一般に使用されている用語と一致していない．またサイズは現実的な取りあつかい易さから，2^n ($n=0,1,2,\ldots$) の形にかぎる．

とにし，これを $M=2^m$ とおく．$N>M\geq 2$ の時はこの集合をセットという場合もある．

M が大きいほど自由度があり，常に必要度の高いブロックをキャッシュ中に残しておくことができる．同じセットに写像される主記憶のブロックが同時に多数必要になった時に M が小さいと入れかえが多くなって能率の低下を生じるが，逆に M が大きいことは各アクセスに対し，M 個の比較が必要なことを意味するから，ブロックのサーチに時間がかかったり，そのための特殊ハードウェアに費用がかかることになる．結局 B を小さく*，M を大きくすることが望ましいが，上記のような実現上の配慮から，ある妥協をせねばならない．

2.1 ブロック形

一般に B が比較的小さく，$B\simeq T$ のものをブロック形と呼び，M と N との相対関係で，次のような 3 種に分けて呼ばれる．

2.1.1 完全アソシアティブ (Fig. 1)

$M=N$ の場合で，先に述べたように，必要なブロックをキャッシュに残す上には最もよいが，一般には N が大きい (128—512 程度) ので現在の技術では実現不可能である．

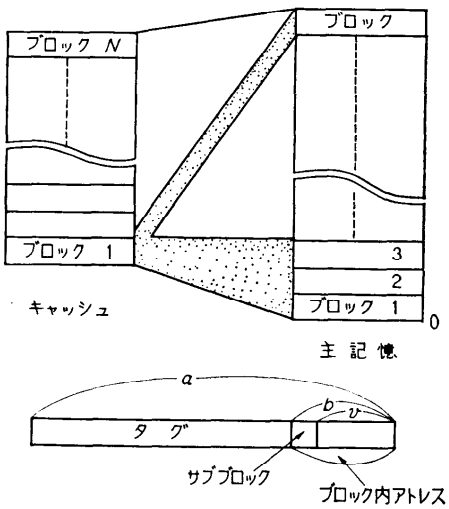

Fig. 1 Full associative mapping ($M=N$)
(a: アドレスビット数，通常 $b=v$ でサブブロックなし.)

* もちろん T より小さくすることは意味がない．また $C\gg T$ である一般の場合を考えている，B の最適値の決定については第 2 部にのべるが，いろいろの要素が入ってくるので，マッピング上だけからは決まらない．ここでは B を小さくして，N を大にする意味に解しておいてほしい．

2.1.2 直接マッピング (Fig. 2)

コングルエント法ともよばれ，$M=1$ にとる．すなわち，主記憶のN個目ごとのブロックはキャッシュの同一ブロックにマッピングされる．この方式はWilkes等が最初に提案したもので，非常に単純であり，ただ1回の比較だけで，判定ができハードウェアが安くすむ利点はあるが，たまたまキャッシュ中の同じブロックに割当てられた複数ブロックが同時に使用されると，入れかえが激しくなって，キャッシュの効果がほとんどなくなってしまう．Bが小さく，Nが大きいほどその確率は小さくなるが，キャッシュの各ブロックにいま，主記憶のどのブロックが入っているかを示すためのタグビットを格納するエリアがNに比例して多くなる．また，命令の入っているブロックとオペランドの入っているブロックの関係は独立な場合が多いから同じマッピングブロックになる確率はそれほど小さくはない．そのため，現在実際にはあまり使用されていない．

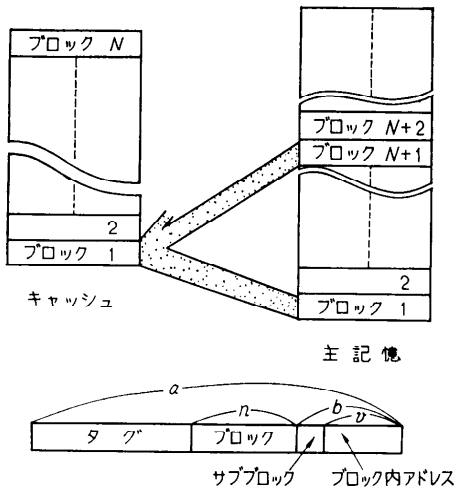

Fig. 2 Direct mapping ($M=1$)
(通常 $b=v$ でサブブロックなし.)

2.1.3 セットアソシアティブ形 (Fig. 3)

前記2種類のマッピングの中間的なもので $2 \leqq M < N$ のような M の値を持つものをよぶ．M はあまり大きくとらない．(現用されているのは $M=2$, と $M=4$.) M が大きいと比較チェックに時間がかかるわりに，それだけの性能向上は望めないからである．一般にアドレスパターンには命令，2つのオペランド，結果の4個の中心があるから，$M>4$ とすることはそれほどの意味がない．しかしながら，この方式は比較的簡単なハードウェアで完全アソシアティブに近い効果を上げられるので，主流になりつつある．

2.2 セクタ形 (Fig. 4)

Bが小さく M が大きい ($B=T$, $M=N$) 理想的な完全アソシアティブ形に対して，M を小さくすることで妥協したものが，セットアソシアティブ形であったが，逆に $M=N$ のまま，Bを大きくしてNを減らすことにより実現を可能にしたのが，このセクタ形であると考えられる．この場合，$M \cdot B=C$ で一定であるから，Bを大にすれば，M を実現可能な程度に小さくできる．しかし，M は普通，8, 16, 32 のいずれであって，$B \fallingdotseq 1K$ 程度になる (この形ではブロックのことを特にセクタと呼ぶ) から $B \gg T$ となってこのままでは，主記憶とキャッシュ間の転送の際，主記憶

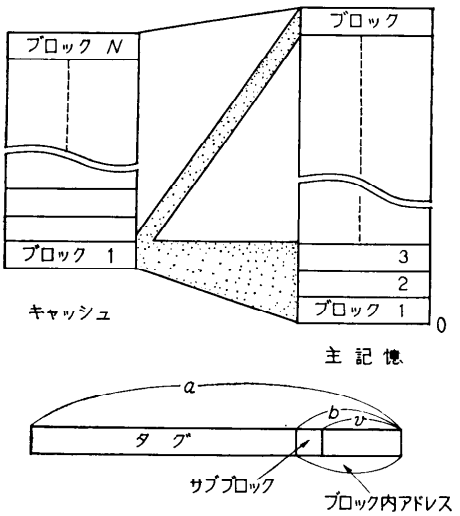

Fig. 3 Set associative mapping ($M=2$ の場合)
(図では $v=b$)

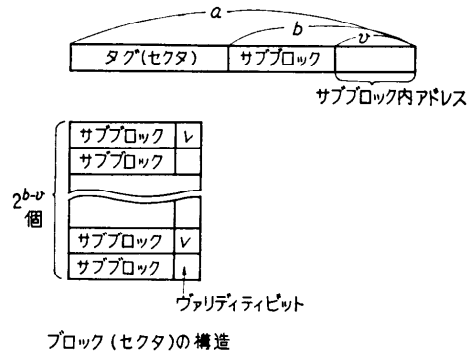

Fig. 4 Sector mapping (Mapping structure is same as Fig. 1)

の多数回の読出しが必要になり，アクセス時間が極めて長時間になる．そこで，1回の主記憶の読出しでよいようにするため，セクタをさらにサブブロック* に分割し，一度の転送単位とする．したがって，この方式ではブロック中のデータは全部が主記憶の写しになっているとはかぎらないので，各サブブロックごとに，いまキャッシュの中にあるデータが有効かどうかを示すビット（ヴァリディティビット）がいることになる．比較に関しては，普通 $M(=N)$ 個の連想レジスタを用意し，必要なセクタをアソシアティブサーチする．この方法の欠陥はアソシアティブレジスタが必要なことと，セクタの中で，有効なサブブロック数が通常半分以下で，キャッシュの使用効率が悪いことである．しかし，キャッシュを初めて採用した IBM 360/85 はこの方式であった．

2.3 主記憶とキャッシュ間の転送

上記の説明においては T が相当大きい（$T \geq 16$）ことを前提としている．一般に，キャッシュが有効な働らきをするためには主記憶とキャッシュ間のデータ転送量が十分大きいことが保証されねばならない．本節ではこの点について若干の補足をする．

理想的には最適の大きさのブロック全体が主記憶から同時に読出され，同時に転送されることが望ましいが，現実にはコストや物理的制限のために困難な場合が多い．したがって，ブロックは何回かに分割して，転送することになるが，この場合，主記憶のサイクル時間は普通，転送時間より，はるかに大であるから，記憶バンクを横切って，アドレス付けするインタリービングを用いるのが有効である．すなわち，たとえば，$B=64$ とすると，主記憶のバンクあたりの読出し幅が16なら4ウェイの，8なら8ウェイのインタリービングによって，ほとんど1サイクル時間内にTバイトの読出し，転送を完了することができる．

しかし，比較的小さいシステムでは主記憶の読出し幅が小さく，インタリービングを行なっても，1転送ユニットの読出しに，2回以上のメモリサイクルが必要になる場合がブロック形でもあり得る．そこで，この時はセクタ形のように，ブロックをサブロックに分割し，ヴァリディティユニットを1回のサイクルで読出せる記憶量に一致させる方法が行なわれている．前にブロック形で $B \simeq T$ と書いたのは通常 $B=T$ であるが，このように $B=2T$ になるような場合があることを意味している．

なお，細部にわたるが，主記憶からの転送を複数回に分けて行なう時，アドレス順でなく，必要な部分をまっ先きに転送する方法や，その時，キャッシュとCPU に同時に転送を行なう方法 (load through) は複雑さをあまり増さないで性能をあげられるので，広く用いられている．

2.4 比 較

各方式の相対関係を見るために，表1に必要ビット数，サーチ回数等をまとめる．この表からセットアソシアティブ形が最も一般的形式であることがわかる．

なお，具体的，機種で採用されているマッピング方式については第2部付録の一覧表を参照してほしい．

Table 1 Comparison of mapping methods

	ダゴビット数	比較の回数	ブロックあたりのザブブロック数	全 Valid ビット数
直 接	$a-c$	1	1	2^{c-b}*
完 全 アソシアティブ	$a-b$	$2^{c\ b}$	1*	2^{c-b}*
セット アソシアティブ	$a-(c-m)$	2^m	2^{b-v}	2^{c-v}
セクタ	$a-b$	2^{c-b}	2^{b-v}	2^{c-v}

* 通常 $b=v$ だから．
a: アドレスの全ビット数．

3. ストア方式

CPU からの記憶装置に対する要求は大部分読出しである．この場合主記憶とキャッシュ内のデータは同一であるから何等問題はないが，ストアが発生した時もし，キャッシュだけを書きかえると主記憶とキャッシュ内に不一致を生じる．この対策には次の2つが考えられる．

3.1 直接ストア方式 (Immediate store)

この方法はストアに対しては，必ず主記憶とキャッシュ（もしあれば）を直ちに書きかえておくものである．その結果，常に一致はとられるが，ストアに対してはキャッシュの効果は全く失なわれる．特に長いフィールドに対するストアでは各アクセスごとに主記憶の同一のブロックをアクセスすることによって，いちじるしく性能がおちる．これは主記憶側のバッファリングや，フィールド長の指示等である程度防げるが，ストアが多いと性能が落ちることには変りない．しかし，主記憶とキャッシュの一致が保たれる点と簡単さが買われて，IBM を始め市販機種は全てこの形式を用いている．

なお，キャッシュに対応ブロックがなかった時の処

* 一般にはこれをブロックと呼んでいるが，ここでは統一的扱いのためこのサブブロックをヴァリディティユニットと呼び大きさを $V=2^v$ とする．

理として2つあることに注意してほしい．すなわち，主記憶だけを書きかえるにとどめるが，キャッシュのエリアを割当ててそこにも書きかえておく(すなわち，読出しと同じ取り扱いをする）かである．その得失はマッピング方式，アドレスパターン，ブロック長等いろいろの要素と関連するので，一概にはいえないが，それほど大きなちがいはない．

3.2 スワップ方式 (Swapping)

この方法ではキャッシュ中に対応するデータユニットが存在すれば，キャッシュのみ書きかえて，主記憶はそのままにして，キャッシュのデータに対応がくずれたことを示すビットをセットしておく．そして，そのブロックのキャッシュ上での割当てを変更する際に初めて，ビットがONなら，主記憶のデータを書きかえる．したがって，書きかえビットがONである間，主記憶とキャッシュの対応関係はとれていないが，同一ブロックへの書きこみが続いておこる場合（一般に多い）には，直接ストアに比べて，かなり性能の向上がある．欠点は，対応がとれていないために入出力の書き出しの際に，いちいちキャッシュをチェックする必要があることである．（読みこみの場合にはいずれの場合もチェックしなければならない．）また，セクタ形では，1ブロック中に多数のサブブロックがあるため，割当て変更時に，複数個のサブブロックの書きかえビットがONであることがあり，その時は一時的にメモリのスワッピングに長い時間がとられ，先へ進めなくなるので，この方法は望ましくないといわれている．なお，直接ストアの場合同様，キャッシュ中にデータがない場合，新らたな割当てを行なうか，単に主記憶の書き変えにとどめるかの選択がある．

3.3 比 較

両者のいずれを採用するかはどの特徴をより高く，評価するかによって異なる．最も合理的と考えられる“性能”という基準に対しても，具体的に何をもって性能と定義するかがはっきりしなければ比較のしようはない．たとえば，同一プログラムに対するメモリサイクル時間の総和をもって，性能を表わすとすれば，スワップ形の方がかなりよくなると予想される．実際，ある仮定をおけばそれを証明することもできるし，シミュレーションの結果（第2部参照）もそれを示している．しかしながら，サイクル時間の総和はレジスタによるバッファリングやインタリービング等の一種の並列動作のやり方によって異なってくるから，実際にかかる時間と必ずしも一致するわけではない．

一方，主記憶とキャッシュ間の不一致をどのくらい不利と判断するかが大きくきいてくるが，これに対しては論者によって意見は異なっている．IBMはこの点とコストの利点をかなり大きく考えて，直接ストア法を採用したものと思われる．しかし，セクタ形はスワップが一時に起きる点がうまくないとしても，ブロック形ではスワップ方式を採用する方が性能的によく，価格，性能比をよく検討して決定すべきである．

スワップ形の欠点については次のように考えることができる．キャッシュの故障に対する保護は，故障が発見された時は必ずしも故障の始まった時とは限らないから，少し前のデータが主記憶にある方がもどりやすいといえないこともないし，入出力やマルチプロセッサにおけるデータチェックの必要性は書きこみにおいてはどちらも同じことである．また，入出力機器への書き出しはそのリクエスト時に一回チェックするだけでも，大きな不都合はないと考えられるし，各ストア時にそのエリアに入出力が行なわれている時だけ主記憶へ入れてもよい．マルチプロセッサでは問題が残るが，IBMの使い方でのキャッシュは本質的にマルチプロセッサに向いておらず，もし使おうとするなら，やや異なった方法を案出せねばならないと思うが，どうであろうか．IBMはキャッシュを採用するにあたって，従来の360シリーズのアーキテクチャに変更を与えず，キャッシュをプログラマに見えないようにする（transparent）ことを前提条件としていた．われわれがキャッシュを用いて新しいシステムを考えるには，この制限をはずして考えてみなければならない．

4. 入れ換えアルゴリズム

2にのべた4種のマッピング方式のうち，直接マッピング以外は主記憶のそれぞれのブロックがマップされるキャッシュの中のブロックは一定していないので，新たにアロケーションをしようとする時，なんらかのアルゴリズムによって，キャッシュからおい出すブロックを決定しなければならない．この入れ換えアルゴリズムは，パフォーマンスに対し，他のパラメータに比して2次的影響しか持たないというシミュレーションの結果もある[5]が，やはり性能やハードウェアとしての実現の容易さ等を十分検討して，最適のものを選ばねばならない．

この種の入れかえの状況は，ページングの際のスワップアウトするページの決定，連想レジスタの割当て，コンパイリングの際のインデクスレジスタの割当てな

どに同様のことが起こるので，古くから研究されている．Belady は“最も後に使用されるブロックを追い出しブロックとする”という最適アルゴリズムを定義したが[4]，コンパイルの場合と異なり，キャッシュでは将来のことを知ることはできないので，この方法は事実上使用することができない．現実的方法としては次のようなものが考えられる．

（1） ランダム．
（2） サイクリック（FIFO）．
（3） 最後に使用されてから最も時間の経過したもの．
（4） 過去の使用状況から，適当な重みで計算を行ない最適ブロックを予測するもの．
（5） （3）を変形し，一部（1）または（2）をとり入れて，簡略化したもの．

（1）は文字通り全てのブロックを等確率，でたらめに選ぶのであって，直観ほど悪い方法ではないが，完全なランダムを達成するのは必ずしも容易ではない．

（2）はブロックを順番に割当てていくだけで，最も単純であるが，（1）と同様，過去の使用状況に関するデータはいっさい使用されておらず，すぐに必要なデータがスワップアウトされてしまうこともある．

（3）は長い間使用されなかったブロックほど今後，使用される確率が小さいであろうという想定にもとづいている．実際にその通りかどうかはシミュレーションによらねばならないが，セクタ形の IBM 360/85 と大型プロジェクト計算機ではこの方法を採用している．ハードウェア的には各ブロックごとにポインタを入れる記憶語を用意し，古いものを先頭にリスト状につないでおき，新たにアクセスされたブロックが，常に最後に来るように操作しておく．そして，ブロックの割当ての際にリストの先頭にあるものを採用する．これらの操作機構にはかなりのハードウェアが必要なのが，この方式の欠点である．

（3）をさらに一般化した（4）は予測関数のよいものが，わかっていないし，それに必要なデータをとるにもかなりのハードウェアを用意しなければならない．たとえば，ETL MK-6 のページングでは，各ページごとに6ビットのページ参照カウンタをおき，参照頻度の統計をとっているが，そのデータを用いたよいアルゴリズムは判明していない．また，Atlas ではかなり複雑なアルゴリズムが用いられているが，場合によっては単純なものより，悪くなることすらあるといわれている．

要するに，これにこっても，それだけの性能向上はあまり望めず，費用の点でも損であるが，いろいろのデータを取ることによって，新しい予測方式が見つかればおもしろい．しかし，アドレスパターン依存性が大きいのであまり望みはないであろう．

（3）の方法は，セット（マッピングブロック）ごとのブロックについてリストを作ればセットアソシアティブ形にも適用することができ，実際，360/195, 165 等の大型のものでは（3）を採用しているが，ハードウェアは大きくなる．大規模でない場合にはもっと単純な方式が望ましい．そこで，そのような場合，簡単化して，（1）または（2）に近づけてしまうことが考えられる．たとえば，ブロックごとに1ビットずつ用意し，始めリセットしておき，アクセスされた時，これをオンにする．もし全部のサブブロックのビットがオンになったら，その時点でそれ以外のビットを全てリセットし，ブロックの割当てをする時はリセットされているブロックの中から適宜選ぶというような（1）と（3）の折衷案も考えられている．

また，これより，少し（3）に近いものとして，1ブロックに2ビット（P, A とする）を用いる次のような方法もある[4]．すなわち，データが発見された時はAをセットし，もしそのブロックのPが1ならこれをリセットした後，ブロックを順番に（ラップアラウンド）サーチして，始めて見つかった $A=0$ のもののPをセットし，その間のもののAをリセットする．このように操作しておいて，入れ換えにあたっては$P=1$ のブロックを用いる．

一方，（1）に近いものとして，IBM 360/155 で採用されているものがある．これは非常に簡単で，マッピングブロックは2個しかないから，それぞれのセット内のブロックを2つに分割してグループ化する．全体に1個のラッチをおき，最後に読出しを行なった側のグループを示しておき，それと反対のグループに属するブロックを割当てる．ラッチはセットごとではないから，他のセットとアクセスがまざることを考えればこれはむしろ（1）の簡単な実現方式ともみられよう．ラッチ1個という非常に簡単なハードウェアしかいらないのが利点である．

ともあれ，先に述べたように入れ換えアルゴリズムはパフォーマンスに2次的効果しか及ぼさないし，プログラムの特徴の影響との相関があるから，どのプログラムにも最適というものは恐らく存在しない．シス

テムの大きさによって，どれを採用するか決まるが，あまり複雑なものは価格，性能比の点で得策でない．

参考文献

キャッシュシステム関係の主な文献を以下にしめす．* 印を付したものは本文または第2部で引用したものである．なお，学会における口頭発表や委員会等の内部資料は引用したもの以外は省いた．

*1) T. Kilburn, D. B. G. Edwards, M. J. Lanigan, F. H. Summer: "Onelevel storage system," IRE Trans. EC-11 No. 2, pp. 223-235 (1962).

*2) M. V. Wilkes: "Slave memories and dynamic storage allocation," IEEE Trans. C-14, No. 4, pp. 270-271 (1965).

*3) G. G. Scarrott: "The efficient use of multilevel Storage," Proc. IFIP, pp. 137-141 (1965).

*4) L. A. Belady: "A study of replacement algorithms for a virtual storage computer," IBM Systems J. Vol. 5, No. 2, pp. 78-101 (1966).

*5) D. H. Gibson: "Consideration in block-oriented system design," Proc. SJCC, Vol. 30, pp. 75-80 (1967).

*6) "IBM System/360 model 85. Functional characteristics," IBM Corp. Form A 22-6916.

*7) C. J. Conti, D. H. Gibson and S. H. Pitkowsky: "Structual aspects of the System/360 model 85. I. General organization," IBM Systems J., Vol. 7, No. 1, pp. 2-14 (1968).

*8) J. S. Liptay: "Structual aspects of the System/360 model 85. II. The cache," IBM Systems J. Vol. 7, No. 1 pp. 15-21 (1968).

*9) S. S. Sisson, M. J. Flynn: "Addressing patterns and memory handling algorithms," Proc. FJCC, Vol. 33, pp. 957-967 (1968).

*10) C. J. Conti: "Concepts for buffer storage," Computer Group News, Vol. 2, No. 8, pp. 9-13 (1969).

*11) 加納弘: "バッファメモリ方式のシミュレーション", 電気学会, 論理装置の設計・製造自動化専門委員会資料 (1969).

*12) R. A. Mclaughlin: "The IBM 360/195," Datamation, pp. 119-120, Oct. (1969).

13) D. H. Gibson: "'Cache' turns up a treasure," Electronics, pp. 105-107, Oct. 13 (1969).

*14) F. F. Lee: "Study of 'look-aside' memory," IEEE Trans. C-18, No. 11, pp. 1062-1064 (1969).

*15) 相磯秀夫: "ETL MK-6 の先回り制御," 電気試験所研究報告, 第702号 (1969).

*16) D. J. Kuck & D. H. Lawrie: "The use and performance of memory hierarchies. A survey," Report No. 363 Department of computer science, University of Illinois, Dec. (1969).

*17) H. S. Stone: "A Logic-in-Memory computer," IEEE Trans. C-19, No. 1, pp. 73-78 (1970).

*18) J. O. Murphey, R. M. Wade: "The IBM 360/195," Datamation, pp. 72-79, April (1970).

*19) R. M. Meade: "On memory system design," Proc. SJCC, Vol. 36, pp. 33-43 (1970).

*20) "A guide to the IBM System/370 model 155," IBM Corp. Form GC 20-1729 (1970).

*21) "A guide to the IBM System/370 model 165," IBM Corp. Form GC 20-1730 (1970).

*22) P. J. Denning: "Virtual memory," Computing Surveys, Vol. 2, No. 3, pp. 153-189 (1970).

*23) 堀越, 小高: "超高性能電子計算機のメモリ制御," 情報処理学会大会 (1970).

24) R. M. Meade: "Design approaches for cache memory control," Computer Design, Vol. 10, No. 1, pp. 87-93 (1971).

25) H. Iizuka: "A note on the simplified analysis of slave memory system," Bul. Electtech. Lab., Vol. 35, No. 2 pp. 61-76 (1971)

*26) 飯塚肇: "スレーブメモリを持つ計算機システムの簡単な解析," 電子通信学会論文誌 (C), Vol. 54, No. 8, pp. 698-705 (1971).

27) H. Katzan: "Storage hierarchy system," Proc. SJCC, Vol. 38, pp. 325-336 (1971).

*28) D. J. Bordsen: "Univac hardware instrumentation System," Proc. workshop on system performance evaluation, ACMSIGOPS (1971).

*29) 飯塚, 照井: "GPSS によるバッファメモリシステムのシミュレーション," 電気関係学会東北支部大会, 2C-2, Oct. 1971.

*30) C. G. Bell: "Computer structure-Past, present, future," Proc, FJCC, Vol. 39, pp. 387-396 (1971).

*31) C. G. Bell, D. Casasent: "Implementation of a buffer memory in minicomputers," Computer Design, Vol. 10, No. 11, pp. 33-89 (1971).

*32) R. M. Meade: "How a cache memory enhances a computer's performance," Electronics, pp. 58-63, Jan. 17 (1972).

*33) 武井欣二: "仮想記憶方式," 電子工業振興協会報告 47C-241, pp 35-64 (1972).

(昭和47年3月9日受付)