解　説

記憶階層のためのソフトウェア技術†

新井克彦†† 高木明啓††

1. はじめに

一般に記憶装置の容量とアクセス速度は相反する要素であるため，大部分の計算機システムでは異なる技術を使った複数種類の記憶から構成される記憶階層が採用されている．このようなシステムでは，CPU が直接アクセスできる上位のレベルに小容量，高速の高価な記憶を置き，下位のレベルに大容量，低速の安価な記憶を置くという構成がとられる（図-1）．オペレーティング・システム（OS）の記憶管理は，必要なプログラムまたはデータを CPU から直接アクセスできる上位の記憶上に移動させる必要がある．記憶管理の目標は，このような情報の移動回数を極力少なくすることによって，最下位の記憶の容量に相当する大きな容量を持ち，かつ，最上位の記憶に匹敵するアクセス速度を持つ仮想的な記憶を実現することにある．本稿はこのような目標を達成するための種々の記憶管理技術について概観する．なお，記憶階層を採用しているシステムでは，一般のプログラムは仮想記憶上のアドレス（仮想アドレス）のみを意識し，仮想アドレスからの実際の（最上位）記憶上のアドレス（実アドレス）へのマッピングはハードウェアが OS の助けを借りて行う場合が多いため，アクセス速度と容量以外に更に高度な記憶の抽象化が可能である．特に，将来記憶の価格性能比の飛躍的な向上が見込まれる一方，プログラムし易さに対する要求は増大すると予想されるので，そのような抽象化の方がより重要性を増していくものと思われるが，本特集の主題からは多少はずれるのでここではあまり深く言及しないことにする．

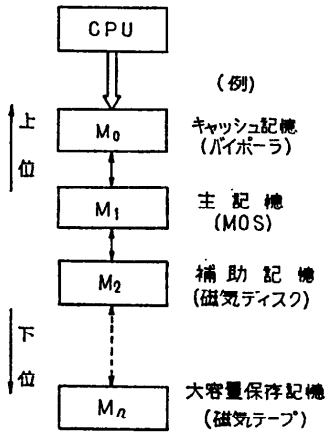

図-1　記憶階層概念図

† Software Technologies for Memory Hierarchies by Katsuhiko ARAI and Akihiro TAKAGI (Yokosuka Electrical Communication Laboratory).

†† 日本電信電話公社横須賀電気通信研究所

1.1 局所性

実行中のプログラムが命令やデータを参照するパターンはランダムではなく，現在アクセスしている情報を含む部分集合の近傍に集中する傾向があることが，種々の測定やシミュレーションによって示されている．この部分集合（局所参照集合）はそれほど大きくなく，かつ，その変化は大部分の時間，比較的ゆるやかであることが認められている[11]．この現象はプログラムの「局所性」としてよく知られており，これが記憶階層の有効性を支える基本原理となっている．すなわちこの原理が成立すれば，記憶管理はその時々の局所参照集合を常に最上位の記憶に置いておくことによって前述の目標を達成することが出来る．逆に，最上位の記憶が実行中のすべてのプログラムの局所参照集合を格納し得るだけの容量を持っていなければ，目標は達成されない．実際，後述するように，インプリメント上仮想記憶管理のキーポイントはプログラムの局所参照集合の推定方法であると言っても過言ではない．

1.2 記憶管理の構成要素

記憶階層の管理は以下に列挙するような諸要素から構成される．なお，以後は簡単のため，主記憶と補助記憶の2階層から成る記憶階層に話を限定するが，ここでの議論は多数レベルから成る記憶階層に容易に拡張することができる．

（1）管理主体

利用者プログラムが主として記憶管理を行う場合と，OS が主としてこれを行う場合の二つに大別することができる．前者は利用者が自分のプログラムの振舞い（特にその局所性）についての知識を十分に活用できるという利点がある反面，①利用者がやっかいな記憶管理を負担しなければならない，②多重プログラミング環境下ではシステム全体としての効率的な記憶管理を実現することが難しいという欠点も持っている．後者はこれと反対に，利用者側に記憶管理の負担が少なく，また，システム全体としての最適化が可能であるかわりに，十分に効率的なプログラム実行を実現するためには利用者プログラムの再構成[16]等のオフライン時調整が必要になる場合もある．更に，利用者がそのプログラムの振舞いに関する情報を OS に提供することによって OS の記憶管理の効率を上げる方法も種々考えられている[19]．

（2） 管 理 単 位

管理単位とは記憶割当ての単位のことであり，本稿ではこれをブロックと総称する．ここでは，ブロック・サイズとそれが固定長か可変長かの二つが主たる問題となる．一般にブロック・サイズが大きければ，必要な情報と一緒に多くの不要な情報も主記憶に上がってくるので主記憶の使用効率は下がる．逆にブロック・サイズが小さければ，記憶管理オーバヘッドやブロック転送回数が増加してシステムのスループットが低下する．一方，固定長ブロックの場合は記憶の内部断片化（まるめによる記憶使用効率の低下）が，可変長ブロックの場合は，外部断片化（市松模様現象による記憶使用効率の低下）が発生する[26]．

（3） 管 理 方 式

（a） 割当て方式

各記憶階層において情報に空きブロックを割当てる方式である．固定長ブロックの割当て方式は比較的簡単であるが*，可変長ブロックの場合は適切な空きスペースの探索や，外部断片化が顕著になった場合の詰め直し等，かなり複雑な管理が必要である[17]．

（b） 取出し方式（契機）

補助記憶上の情報を主記憶に転送する契機を決める方式である．実際に情報が必要になった時点で主記憶に転送するデマンド方式と，近い将来必要になる情報を予測して，あるいは，利用者プログラムに教えてもらってあらかじめ主記憶に上げておく先取り方式とがある．先取り方式はデマンド方式に比べて，主記憶への転送回数が減る可能性があるかわりに，補助記憶上にある情報を早期に主記憶に上げるため，主記憶滞留時間が大きくなり，主記憶効率が低下する．プログラムの振舞いを正確に予測することは大変難しいこともあって，大部分の状況においては，先取り方式の採用によって得られる利益はその損失分を補うことができないと言われている[30]．

（c） 置換え方式

補助記憶上の情報を主記憶に上げるにあたって，主記憶に空きを作るために代わりに補助記憶に追い出すべきブロックを選択する方式である．理想的には，近い将来に参照されないブロック，すなわちプログラムの局所参照集合に属さないブロックを選択すべきであるが，プログラムの将来の振舞いは知る術がない．そこで，現時点までのブロックの使用統計や利用者（プログラム）から提供された情報等をもとにして近い将来最も参照されそうにないブロックを予測するという方法がとられる．置換えブロックの予測アルゴリズムに応じていろいろな置換え方式が存在する．

（d） 主記憶分割方式

多重プログラミング・システムにおいて各プログラムにどのように主記憶を配分するかを決めるもので，固定分割方式と可変分割方式とに大別できる．固定分割方式では，各プログラムに固定的に主記憶量を割当て，ブロック置換えはそのプログラムに割当てられた主記憶内で行われるこの方式は，プログラムの局所参照集合があらかじめ予測でき，かつその変動も小さい場合には最適な方式となるが，そのような条件が成り立たない場合には主記憶使用効率を悪化させ，スループットの低下を引き起こす．一方，可変分割方式では，各プログラムはブロック置換えに際して固定分割のような制限を設けないので，各プログラムに割当てられる主記憶量は動的に変化する．この方式は一般に固定分割方式よりも高い主記憶使用効率を達成できるが，ブロック置換え方式および次に述べる負荷制御方式が適切でないと，補助記憶との間のブロック転送が頻発するスラッシング現象[9]によってシステムのスループットが著しく低下する．

（e） 負荷制御方式

主記憶の負荷を適正に保つようにジョブの多重度を制御する方式で，静的負荷制御方式と動的負荷制御方式とに大別できる．静的負荷制御方式はジョブ多重度

* それでも磁気ディスクや磁気ドラムのような補助記憶の場合は，完全なランダム・アクセス記憶ではないので，ブロック割当てアルゴリズムを工夫することによってアクセス時間の短縮を図ることが多い．

の上限値をあらかじめ定めておくだけで，主記憶負荷状態に応じた多重度制御は行わない．一般に各プログラムが必要とする主記憶量はそれぞれ異なり，かつ時間的にも変動するので，この方式のもとでスラッシングの起こる確率を低く抑えるためには，ピーク負荷に備えて余裕をもって主記憶を用意する必要がある．これに対し，動的負荷制御方式は主記憶の負荷状態に応じてジョブ多重度を制御するので高い主記憶使用効率の達成が期待できるが，負荷状態の評価尺度として何を採用するかが最大のポイントとなる．

次節においては，これまで開発された記憶容量管理技術の主要なものを，以上のような構成要素に照らして概説する．

2. 記憶管理技術の発展

本節ではこれまでに開発された主要な記憶管理技術，すなわち，動的オーバレイ，スワッピング（ロールイン・ロールアウト），ページング，セグメンテーションおよびページ化セグメンテーションについて順を追って解説する．

2.1 動的オーバレイ

PL/I や ALGOL 系の言語の有効範囲規則は，この種の言語で書かれたプログラムが図-2 のようなブロック構造（図式的にはトリー構造）を持つことを意味する．アセンブリ言語等のその他の言語で書かれたプログラムにしても，このような構造を持たせることはごく自然であると思われる．このようなプログラムでは，あるモジュール M から直接参照できるデータや手続きは M 又はその外側のモジュール（つまり，トリー上でルート R から M に至るパスに含まれるモジュール）で定義されたものに限られる．したがってパス R→M に含まれないモジュールは，少なくとも M 実行中は主記憶上に存在する必要はない．常に実行中のモジュールからルートに至るパスに含まれるモジュールだけを主記憶に保持する記憶管理方式を動的オーバレイと言う．この方式のもとでは，各プログラムにはその最長パスに含まれる全モジュールを保持するのに必要な主記憶量を割当てる必要がある．

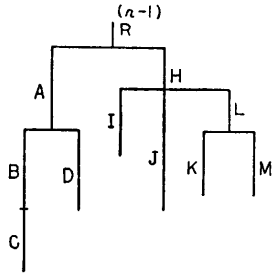

図-2 プログラムのブロック構造（例）

動的オーバレイでは，利用者が自分のプログラムのトリー構造を表わす情報（オーバレイ・マップ）をリンケージ・エディタに入力し，この情報をリンケージ・エディタから引き継いだ OS のローダが各モジュール（しばしばロード・セグメントと呼ばれる）が実際にコールされる時に必要なオーバレイ処理を行う．この方式はプログラムの振舞いや所要主記憶量が利用者によくわかっている場合には有効であるが，プログラムの動作が入力データの値によって変動する等の非決定性を有する場合には適切な方式とは言えない．また，個々の利用者はシステムの負荷や他のプログラムの動作を予測できないので，多重プログラミング・システムにおいては，システム全体から見て効率的な記憶管理を実現することは困難である．したがって，動的オーバレイが効果を発揮するのは，大きさのわかっている領域で頻繁に実行されるコンパイラや，バンキング・システム等に代表される実時間システムのプログラムに対してである．

2.2 スワッピング

多重プログラミング・システムでは，即座に CPU で実行し得る状態にあるプログラムもあれば，入出力装置や端末との間のデータ転送の完了を待っているプログラムもある．後者の待ち時間は一般に非常に長く，そのようなプログラムを待機中ずっと主記憶上に置いておくのは無駄が多い．OS が長い待ち状態にあるプログラムを補助記憶に移し，一方待ち状態が解消したプログラムを主記憶に上げることによって即座に実行できるプログラムのみを主記憶上に保持する方式をスワッピング（またはロールイン・ロールアウト）と言う．スワッピングは，各プログラムの独立性が強く，かつ端末入出力等長時間の待ちが頻繁に起こるタイム・シェアリング・システム等に特に向いている．しかしながら，この方式はプログラム全体（ジョブ）が主記憶上に存在するという明白な無駄は解消できるが，実行中のプログラムのうち本当に必要な部分のみを主記憶に置くといったより効率的なきめの細かい制御はできない．したがって，スワッピングは一次的な記憶管理技術と見るよりも，むしろページング等の他の記憶管理技術と組合わせて使用される補助的な記憶管理技術と考える方が妥当であろう．実際，ページングやセグメンテーション等を使って実現された仮想記

憶システムには，何らかの形でスワッピングを併用しているものが多い[19),20)]．

2.3 ページング

動的オーバレイとスワッピングに関する議論から，OS によるよりきめの細かい記憶管理方式の必要性が明らかになった．これに応える記憶管理方式の一つがページングである．代表的なページングにおいては，

1) 管理単位は固定サイズ・ブロック（ページと呼ばれる）

2) 補助記憶上のページを主記憶に転送する契機は，実際にそのページが CPU によって参照される時であり，そのページが主記憶上に存在しないこと（ページフォルトという）はハードウェアによって検出され，OS に通知される．このようなページングは特にデマンド・ページングと呼ばれる（ページが主記憶上に存在するか否かを示すマッピング情報はページ・テーブルと呼ばれ，CPU は主記憶参照の都度これを調べる）．

3) OS は実行中のプログラムの局所参照集合に含まれていそうにない主記憶上のページを何らかの方法で推定して，新しく必要になったページと置換え，これをページ・テーブルに反映させる．

ページングでは，ページというより細かな単位での記憶管理が OS によって行われるので，システム全体として効率の良い記憶管理が可能になる．一方，ページ参照ごとにページ・テーブルの索引を行うためのハードウェア量とオーバヘッドの増加，内部断片化の発生等の欠点も存在する．しかしながら，ページングの成否を左右する最大の要素は，ページ置換え方式，主記憶分割方式および負荷制御方式である．これらが適切に選択されないと，スラッシングが生じる．

（a） ページ置換え方式

代表的な置換えページ選択アルゴリズムとしては，FIFO (First In First Out), FINNUFO (First In Not Used First Out 又は Second Chance), LRU (Least Recent Used) 等がある[4),7),22)]．FIFO アルゴリズムは，特定のページが遠からず参照される確率はそのページが主記憶上に存在する時間の減少関数になるという仮定にもとづき，最も古いページを置換える．LRU アルゴリズムは，特定のページが参照される確率は前回そのページが参照された時からの経過時間の減少関数であるという仮定にもとづき，最近最も参照されていないページを置換える．また FINUFO アルゴリズムは，前回調べて以来参照されていないページが見つかるまで主記憶上のページを調べ続け，見つかったページを置換えるもので，LRU アルゴリズムの近似であると言える．1 巡目に置換ページが見つからなくても 2 巡目には必ず見つかるところから，Second Chance の別名がある．種々の実験や測定から，これらの置換えアルゴリズムについては，以下のことがわかっている[17)]．

1) FIFO アルゴリズムの（局所参照集合の）推定能力は最も低い

2) LRU アルゴリズムの推定能力は高いが，ハードウェアのコスト増，オーバヘッド増は非常に大きい

3) FINUFO アルゴリズムは簡単なわりに推定能力は以外に高い

（b） 主記憶分割方式

固定分割方式の他に，可変分割方式として，①ページ置換えアルゴリズムをシステム全体に一律に適用することによって，各プログラムへの主記憶割当てを間接的に制御する方式と②各プログラムに常にその局所参照集合の推定値（ワーキング・セット[10)]）分の主記憶を割当てる方式がよく用いられる[13)]．前者は簡単であるが，次のような欠点もある．

1) 負荷制御と連動しておらずスラッシングが発生し易い．

2) 各プログラムに割当てられる主記憶量が他のプログラムの局所参照性，入出力頻度，実行優先度等やその他種々の偶然に左右される．したがって最適な主記憶分割を実現することが難しい．

一方後者は，以下のような長所を有するが，ハードウェアおよび OS のオーバヘッドが大きくなる可能性がある．

1) 主記憶上にワーキング・セット分の空きがなければプログラムは実行が許されないので，本方式は負荷制御と本質的に連動している．

2) プログラムに割当てられる主記憶量はその局所参照性によってのみ決められ，他のプログラムの影響を直接受けることはない．

（c） 負荷制御方式

静的負荷制御方式の他に，動的負荷制御方式として，①主記憶の負荷状態をページ・フォルト率や空きページ数等によって監視し，直接的に多重度の制御を行う方式と②置換ページを最低実行優先度のプログラムから選択することの結果として間接的に多重度の制御を行う方式等が考案されている[5)]．

2.4 セグメンテーション

セグメンテーションは物理的な記憶管理技術として

よりも，むしろ論理的な情報管理技術（あるいは名前管理，オブジェクト管理）である．本稿の主題から若干はずれているので詳しくは立ち入らないが，セグメンテーションの情報管理技術を要約すると次の通りである[15),25)]．

1) セグメントは情報の論理的な単位として，利用者（プログラム）に見える．プログラムが情報を参照する時には，これを含むセグメントの番号とそのセグメント内での相対アドレスを指定することになる．すなわち，セグメンテーションによって実現される仮想記憶空間は，情報（セグメント）の名前空間とセグメント内のアドレス空間から成る2次元空間として利用者には見える．

2) このようにセグメントは論理的な単位としてとらえられるので，次のような情報管理機能がハードウェアおよびOSによってサポートされる場合が多い．

- ○セグメント間の動的リンク機能
- ○共用制御機能
- ○保護機能

一方，これを記憶管理の立場からみれば，ページングとセグメンテーションの違いは，ページが固定長のブロックであるのに対し，セグメントが可変長のブロックであることととらえられる．したがって，以下の点を除けば，ページングと大体同様の議論が成り立つ．すなわち，セグメントは可変であるから，各セグメントは論理的にまとまりのある情報のみから構成することができ，内部断片化は生じない．また同一のセグメントに同時期に参照される確率の高い情報を集めることにより，補助記憶との間の情報転送回数を減らすことも可能である．一方，外部断片化が発生し，これを解消するには詰め直し処理等のオーバヘッドを覚悟しなければならない．また補助記憶の管理もページングより大分複雑になる．

2.5 ページ化セグメンテーション

セグメンテーションは情報管理技術として大きな長所を持っているが，これをそのまま記憶管理技術としても利用しようとすると前述のような欠点がでる．そこで，セグメンテーションとページングを組合せて，情報管理はセグメンテーションで，記憶管理はページングでそれぞれ分担するページ化セグメンテーションが開発された．この方式では，各セグメントが更にページに分割される．ページ化セグメンテーションは，セグメンテーションの情報管理面での利点を保ちながら，外部断片化や記憶割当ての複雑さという問題点をページングによって解決しているという意味で，セグメンテーションとページングの両方の利点を合わせ持っている．唯一の欠点は，セグメンテーションとページングの両方をサポートするためのハードウェアおよびOSのコストとオーバヘッドの増加である．

3. 技術の現状

記憶階層の記憶管理技術の主要な部分はかなり成熟して来ていると言ってよい．前節で概観した種々の記憶管理技術は，現在の汎用コンピュータ・システムにおいて，キャッシュ記憶，仮想記憶およびマス・ストレージ・システム (MSS) の中でさまざまな形で使用されている．この中でキャッシュ記憶と MSS は，プログラムに見えない形で下位記憶に相当する容量で上位記憶に近いアクセス速度を実現するという目標をもっぱら追求しており，固定長ブロックを管理単位として LRU 又はこれと類似した置換えアルゴリズムを使って記憶管理を行っているものが多い．仮想記憶では，依然としてページングを採用したものが主流であるが，純粋な記憶管理に加えて（プログラムに見える世界における）情報管理もサポートできるように，ページ化セグメンテーションを採用するものも多くなっている[1),25)]．もちろん，小型システムや特殊用途向きシステムでは，その用途やコスト上の制約等に見合った（一般にはより簡単な）方式を採用している場合も多い．ここでは主として汎用コンピュータ・システムにおける仮想記憶実現技術の現状について簡単に述べる．

3.1 プログラム動作モデル

前述したように，プログラムの局所参照集合が大部分の時間比較的ゆるやかにしか変化しないことは早くから注目されて来たが，たまに起こるかも知れない急激な変化については比較的最近まで無視されて来た．プログラムの動作のモデルとして 1960 年代後半から 1970 年代前半にかけて提案された IRM (Independent Reference Model), LRUSM (LRU Stack Model) 等[20)]はこのような状況のもとで考えられたモデルである．しかしながら現在では，プログラムの動作は一般に局所参照集合の変化がゆるやかな安定期と局所参照集合が急激に変化する遷移期から成り，遷移期は安定期の 1/10 あるいはそれより短いにもかかわらず性能への影響の点では，安定期に劣らず重要な意味を持っていることがわかっている（図-3）．すなわち，ページ・フォルトの大部分はこの遷移期の動作によって発

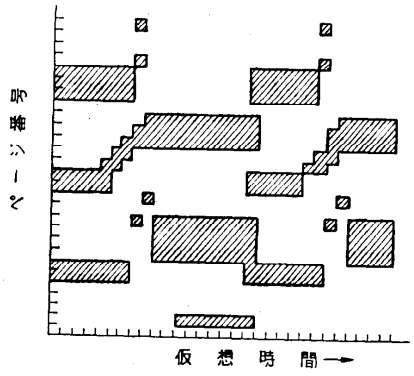

図-3 プログラムのページ番号参照パターン（例）

生すると言われている[26]．現在のところ，このようなプログラム動作のモデルとして十分に有効性が立証されているものはないが，安定期間の遷移をセミ・マルコフ・チェインで表現し，個々の安定期を従来のIRMやLRUSM等で表現した複合モデルがDenning等によって研究されている[14]．

3.2 記憶管理方式

前述したように，多くの汎用コンピュータ・システムはページ又はページ化セグメンテーションを採用する傾向にあり，これに加えてスワッピングが補助的に使われることが多い．内部的には，ページ置換え方式，主記憶分割方式および負荷制御方式の共通の基礎となる概念上のツールとして，ワーキング・セットまたはこれに類似の概念を採用したものが増加しつつある．ワーキング・セット・モデル[10],[13]の優れている点は，局所参照集合の近似であるワーキング・セットを収容するのに必要かつ十分な主記憶量をプログラムに与えるように，ページ置換え，主記憶分割および負荷制御を有機的かつ自然に統合しているところにある．以前はハードウェアおよびOSのオーバヘッドが極端に大きくなるのではないかという不安があってワーキング・セット・モデルにもとづく記憶管理はなかなか採用されなかったが，近年，

1) 制御パラメータの実用的な計算方法がかなり確立されてきた[12]．

2) 既存のページング・ハードウェアを使っても，十分許容できるOSオーバヘッドで実現可能であることが実験的に示された[27]．

3) 専用のハードウェア・サポートを追加するにしても，そのコスト増加はそれほど大きくないことがわかった（1972年の技術水準で主記憶1ページ当り約20ドル[23]），

ことから，IBMのMVS[6]を始め幾つかのシステムで採用され始めた．

3.3 性能評価および調整

記憶管理に関連した性能評価（モデル）は，OSの理論の中では進んだ部類に属する．さほど役に立たないという意見[28]もあるにせよ，プログラム動作や記憶管理方式についてはかなり精密なモデルが開発されてきた．これがdecomposabilityの原理[8]やキューイング・ネットワーク理論[3]等と組み合わされて，記憶管理方式のシステム性能に及ぼす影響，最適多重プログラミング・レベルの決定アルゴリズムの評価等，さまざまな分野に適用されている．

一方，実測等で得た情報をもとにして，同時期に参照される確率の高いモジュールを同一ページに集めることにより，プログラムの局所性を向上させるプログラム再構成法[16],[18]あるいは最初から局所性の高いプログラムを作成する技法[24]等も種々開発されて大きな効果を上げている．

4. むすび

VLSI技術等各種素子技術の急速な進歩は，CPUおよび主記憶装置の価格性能比の大幅な向上と，その結果としてシステム価格に占めるハードウェア（特に本体系装置）価格の割合の低下をもたらしている．このような傾向は，多数の利用者から共同利用される集中型の汎用大型コンピュータから，多数のパーソナル・コンピュータと各種専用コンピュータから成る分散システムへの移行を促すと予想される．一方，データベース処理の重要性はますます増大し，データベースは従来のようなコード化されたデータの他に画像や音声等のイメージ・データも含むようになろう．更に，従来の性能一辺倒から，使い易さやソフトウェアの信頼性等に次第に重点が移ってくると考えられる．このような環境変化のもとで記憶管理技術の主眼点がどこに移って行くかについて，2, 3私見を述べ，本稿を終りたい．

1) これまで，CPUや主記憶装置は高価であり極力効率的に使用することを目標に，各種の複雑な記憶管理方式が開発されてきた．しかしながら，これらのハードウェアが大幅に安くなれば，CPUや主記憶の使用効率は多少悪くても，より簡単な方式を採用することによってソフトウェアの複雑さを削減する方が有利になろう．特に，パーソナル・コンピュータ等単一

利用者に専有されるコンピュータの場合，この傾向は顕著になると思われる．ただし本稿では割愛した仮想記憶の論理的な側面は，プログラミングの容易さ等に直接関わりを持つので，より重要視されるであろう．このようにして，既に技術的にも成熟した(物理的な)記憶管理は，現在よりずっと簡単化されて，その全体がファームウェアに納まるようになると予想される．

2) 従来のプログラム動作のモデルは，逐次的な計算中心のプログラムを前提としてきた．しかしながら，データベースシステム等において，従来と同様に局所性があるか否か，あるいは，局所性やワーキング・セットの概念が多数のプログラムから共用されるデータベース環境のもとでどのように定義されるのか等については，まだ十分な研究がなされていないし，実測等による裏付けもほとんどない．逐次的な計算中心のプログラムの場合，逐次的あるいはループ形態の制御フローの存在が局所性の基礎となっていたが，データベースの場合，何がその役割を演じているのか不明な点が多い．更に，コード化されたデータと画像・音声等のイメージ・データとを一様に扱えるかどうかも疑問である．これらの解明については今後の研究成果に期待したい．

3) プログラム再構成は，確かに性能向上に大きな効果があるが，かなり高価でもある．したがって，これは実時間システムのプログラムのように頻繁に走行するプログラムの場合等にその適用が限られる．ところで，CCD や磁気バブル記憶のようなアクセス待ち時間の小さな補助記憶の登場は，小さなページ・サイズのサポートを実現可能にする．プログラム再構成の効果はページ・サイズが小さくなるほど低下すると考えられるので，今後この技法の重要性は次第に低下していくであろう．

参 考 文 献

1) 「ACOS シリーズ 77 NEAC システム 800/900 システム概説書」日本電気マニュアル．
2) Aho, A. V., Denning, P. J. and Ullman, J. D.: Principles of Optimal Page Replacement, JACM, Vol. 18, No. 2, pp. 80-93 (1971).
3) Baskett, F., Chandy, K. M., Muntz, R. R. and Palacios, F. G.: Open, Closed, and Mixed Networks of Queues with Different Classes of Customers, JACM, Vol. 22, No. 2, pp. 248-260 (1975).
4) Belady, L. A.: A Study of Replacement Algorithms for Virtual Storage Computers, IBM Syst. J., Vol. 5, No. 2, pp. 78-101 (1966).
5) Boon, J. and Bunyan, C. J.: Virtual Storage, Infotech State of Art Report 26, Infotech International (1976).
6) Buzen, J. P.: A Queueing Network Model of MVS, ACM Comput. Surv., Vol. 10, No. 3, pp. 319-332 (1978).
7) Corbató, F. J.: A Paging Experiment with the Multics System, In Honor of Morse, P. M., MIT Press, pp. 217-228 (1969).
8) Courtois, P. J.: Decomposability, Instabilities, and Saturation in Multiprogramming Systems, Commun. ACM, Vol. 18, No. 7, pp. 371-377 (1975).
9) Denning, P. J.: Thrashing: its Causes and Prevention, AFIPS Conf. Proc., Vol. 33, pp. 915-922 (1968).
10) Denning, P. J.: The Working Set Model for Program Behavior, Commun. ACM, Vol. 11, No. 5, pp. 323-333 (1968).
11) Denning, P. J.: On Modeling Program Behavior, AFIPS Conf. Proc., Vol. 40, pp. 937-944 (1972).
12) Denning, P. J.: Working Sets: Then and Now, In Operating Systems: Theory and Practice, North-Holland Publishing Co., pp. 115-148 (1979).
13) Denning, P. J. and Graham, G. S.: Multiprogrammed Memory Management, Proc. IEEE, Vol. 63, No. 6, pp. 924-939 (1975).
14) Denning, P. J. and Kahn, K. C.: A Study of Program Locality and Lifetime Functions, Proc. 5th ACM Symp. on Op. Syst. Principles, pp. 207-216 (1975).
15) Dennis, J. B.: Segmentation and the Design of Multiprogrammed Computer Systems, JACM, Vol. 12, No. 4, pp. 589-602 (1965).
16) Ferrari, D.: Improving Locality by Critical Working Sets, Commun. ACM, Vol. 17, No. 11, pp. 614-620 (1974).
17) Habermann, A. N.: Introduction to Operating System Design, Science Research Associates, Inc. (1976).
18) Hatfield, H. and Gerald, J.: Program Restructuring for Virtual Memory, IBM Syst. J., Vol. 10, No. 3, pp. 168-192 (1971).
19) 伊東，森元，河内，大南：DIPS-104OS の制御プログラム，電気通信研究所，研実報，Vol. 26, No. 8, pp. 2203-2220 (1977).
20) Lynch, H. W. and Page, J. B.: The OS/VS 2 Release 2 System Resource Manager, IBM Syst. J., Vol. 13, No. 4, pp. 274-291 (1974).
21) Madison, A. W. and Batson, A. P.: Characteristics of Program Localities, Commun. ACM, Vol. 19, No. 5, pp. 285-294 (1976).
22) Mattson, R. L., Gecsei, J., Slutz, D. R. and

Traiger, I. L.: Evaluation Techniques for Storage Hierarchies, IBM Syst. J., Vol. 9, No. 2, pp. 78-117 (1970).
23) Morris, J. B.: Demand Paging through Utilization of Working Sets on The MANIAC II, Commun. ACM, Vol. 15, No. 10, pp. 867-872 (1972).
24) Morrison, J. E.: User Program Performance in Virtual Storage Systems, IBM Syst. J. Vol. 12, No. 3, pp. 216-237 (1973).
25) Organick, E. I.: The Multics System: An Examination of Its Structure, MIT Press (1972).
26) Randell, B.: A Note on Strage Fragmentation and Program Segmentation, Commun. ACM, Vol. 12, No. 7, pp. 365-369, 372 (1969).
27) Rodriguez-Rosell, J. and Dupuy, J.: The Design Implementation and Evaluation of a Working Set Dispatcher, Commun. ACM, Vol. 16, No. 4, pp. 247-253 (1973).
28) Saltzer, J. H., et al.: Panel Session on the Role of Performance in Systems Design, 6th Symp. on Op. Syst. principles (1977).
29) Spirn, J. R. and Denning, P. J.: Experiments with Program Locality, AFIPS Conf. Proc. Vol. 41, pp. 611-621 (1972).
30) Tsichritzis, D. C. and Bernstein, P. A.: Operating Systems, Academic Press, Inc. (1974).

(昭和54年12月20日受付)