

バーチャルメモリについて[†] II吉 広 誠 一 竹 黒 沢 格^{††}

5. バーチャルメモリの効率

前回は、バーチャルメモリの種々の構成法について述べたが、今回はそのうちの1つ、ページング方式のバーチャルメモリの効率を中心に述べる。

バーチャルメモリの効率については、ページング方式の場合多くの文献があるが（主に性能に関してまとめたものとしては、文献9）がある、ここではそのいくつかを紹介しながら、ページング方式のバーチャルメモリの性質を概観したい。効率について見る場合、どのような条件のもとで実験されたかに十分注意を払う必要がある。バーチャルメモリの効率は、その設計と走らせるプログラムによってさまざまに変るからである。

バーチャルメモリでは、見かけ上のアクセス時間 Δ' は内部メモリ（以下、主記憶を指す）のアクセス時間 Δ より遅くなる。その原因は、アドレス変換のために時間がかかるためと、ページフォルトの際、その処理に時間がかかるためである。アドレス変換にともなう遅れを含めた内部メモリへのアクセス時間を $g \cdot \Delta$ ($g > 1$)、1回の参照がページフォルトを引き起こす確率を f （以後ページフォルト率とよぶ）、ページフォルトにともなう処理時間を t とすれば、バーチャルメモリの見かけ上のアクセス時間は、

$$\Delta' = g \cdot \Delta + f \cdot t$$

と表わせる。バーチャルメモリの効率をよくするためには、 g と $f \cdot t$ を十分小さくしなければならない。 t は主に補助メモリへのアクセス時間からなり、普通 $t/\Delta \sim 10^4$ であるからまず f を小さくする必要がある。

g については、1回の参照につき内部メモリ上のテーブルを1～2回引くことから、本来2～3の値になるはずであるが、連想レジスタなどの専用ハードウェアを使うことにより見かけ上の値を1に近づけること

ができる。

5.1 ページフォルト

ページフォルトに影響を与える因子としては、ユーザ・プログラムへの内部メモリ割り当て量、ページサイズ、リプレースメント・アルゴリズム、プログラム構造等がある。

a) 内部メモリ割り当て量 (m)

直観的にいって、プログラムに対する内部メモリ割り当て量が増えれば、ページフォルトの回数は減少するであろう。それがどのような曲線になるかが、ここでの問題である。ページフォルト率 f の逆数は、ページフォルト間の平均命令実行数を表わしており、メモリ割り当て量 m に対し、

$$e = \frac{1}{f} = am^k \quad (1)$$

と近似され^{9), 10)}、図9のような曲線になる。ここで a は、プログラム、ページサイズ等により変る値であり、 k は2に近い値をとるといわれている¹⁰⁾。図9よりわかる通りメモリ割り当て量が大きくなるに従いページフォルトが少なくなるが、デマンドページングでは必要に応じ1ページずつがロードされるので、仮に

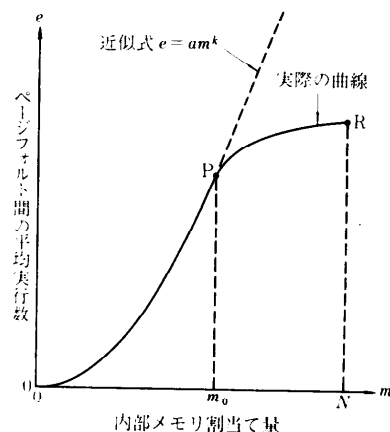


図9 ページフォルト間の平均実行数とメモリ割当て量の関係のモデル図

[†] Virtual Memory

^{††} 電子技術総合研究所電子計算機部記憶システム研究室

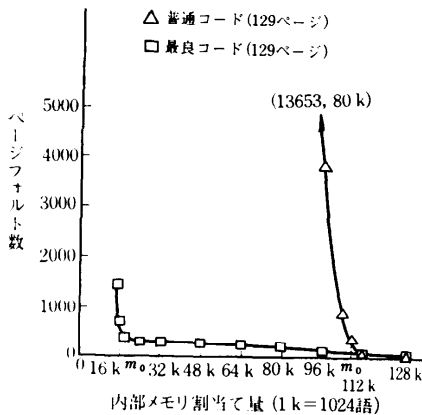


図 10 ページフォルトとメモリ割当て量の関係
(ページサイズ 1k 語, FIFO リプレースメント・
アルゴリズム, 10,000 (×10 語) 項目ソーティ
グ・プログラム, 単一実行の場合)

どんなに多くのメモリ量を割り当てたとしてもページフォルトが生じる。ある程度以上のメモリ容量を割り当てると、 e は図 9 のように P 点から R 点へかけて飽和する。プログラムを効率よく実行するには、P 点を越えた領域を利用しなければならないが、マルチプログラミングのような場合、1 人のユーザが P 点を越えて内部メモリを使うことはシステム全体の効率を悪くするであろう。

f - m 曲線については、(縦軸は必ずしも f に限らず、ページフォルトの回数の場合も含めて) 多くの実験例がある¹¹⁻¹⁴⁾。一例を図 10 に示す¹⁵⁾。いずれもあるメモリ割り当て量以上では、ページフォルトの減少がゆるやかになる傾向を示している。その割り当て量を m_0 とすれば、そこでは m_0 とプログラム・サイズ N との比が重要な値になる。もちろんその値はプログラムの種類やコーディングによって異なるが、実験例^{14), 15)} から見積ってみると、 $N/m_0=1.3\sim 5$ 程度になると思われる。すなわちシステムの効率を考慮すれば、見かけ上の内部メモリ容量の増加は、1.3~5 倍程度ということになる。 (むしろ最良コーディングにした場合、この値を越えることもあり得る。)

b) ページサイズ (z)

ページサイズがページフォルト率に影響を与える要因は 2 つある。1 つは余剰語が内部メモリに入る割合がページサイズに依存するためであり、もう 1 つは、ページサイズが大きくなるほどページフォルト処理時間に対する転送効率がよくなるからである。

N 語のプログラムがあるとき、実際に参照される語数 N' は普通 N より小さい。このとき $N-N'$ の参照されない語を余剰語とよぶ。ページサイズが z のとき、プログラム全体のページ数は、

$$P = \left\lceil \frac{N}{z} \right\rceil \text{ ページ} \quad \left(\frac{N}{z} \leq P < \frac{N}{z} + 1, P: \text{整数} \right)$$

であるが、実際に参照のあるページ数 P' は P よりも小さい。余剰語のみで 1 ページが構成される場合があるからである。いま、プログラムの圧縮率を、

$$C(z) = \frac{P'}{P} \leq 1$$

と定義する。特殊な場合、 $C(1)=N'/N$, $C(N)=1$ で、その間では z が大きくなるに従い $C(z)$ は増加する。その実験式としては、

$$C(z) = a_0 + a_1 \log_2 z$$

が提示されている。プログラムに大きく依存するが、 z が 32 語以下では、 $C=0.1\sim 0.4$ 、512 語以上では、 $C \geq 0.8$ となる場合が多い³⁾。(圧縮率に関して、文献 9) には、 $C(z) = b_0 + b_1 \cdot z$ という式が出ている。) この $C(z)$ を用いれば、プログラムのうち実際にロードされる部分は $N \cdot C(z)$ と表わされ、ページサイズが小さい程実質的なプログラム・サイズは小さくなり、必要なメモリ割り当て量を少なくすることができる。

一方、メモリ割り当て量が十分ある場合、特に $m \geq N \cdot C(z)$ のときにはプログラム終了までの平均ページフォルト率は、

$$f(m) = \frac{N \cdot C(z)}{R \cdot z} \quad R: \text{全参照数}$$

となり、ページサイズが大きい程その値は小さくなる。すなわち、メモリ割り当て量が多い場合には、 $C(z)$ を小さくする、つまりページサイズを小さくする必要性が薄れる。

以上のように、メモリ割り当て量が小さいときにはページサイズが小さい方が有利であるが、大きくなるとその逆の効果が出てくる。図 11 は FORTRAN コンパイラについての実験例⁹⁾ であるが、上記のような傾向を示している。同様の実験結果は、文献 12), 14) にもある。

つまり内部メモリ容量 (主記憶容量) が小さくて、各プログラムに十分な内部メモリ容量を割り当てることができない場合には、ページサイズを小さくして $C(z)$ を小さくする方が望ましいであろう。しかし各プログラムに割り当てる内部メモリ容量を大きくでき

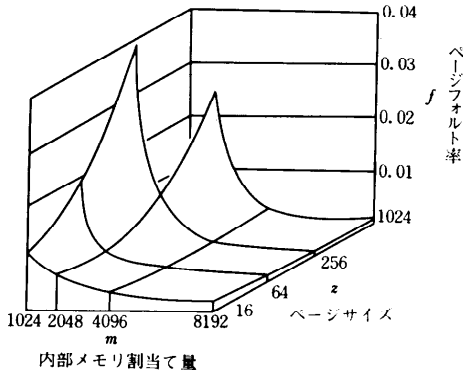


図 11 ページフォルト率と内部メモリ割当て量、ページサイズの関係 (FORTRAN コンパイラによるデータ)

る場合は、ページサイズは大きい方が有利であると思われる。

しかし、ページサイズを決定する要因にはこのほかに、補助メモリの転送速度とか、ページ内の不使用領域の大きさ（ページサイズの小さい方が有利）とページテーブルのエントリーの数（ページサイズが大きければエントリーは少なくすむ）との兼ね合い等があり、簡単には決らない。通常ページサイズは 256～1,024 語が用いられているが、2通りのページサイズを用意しているシステムもある。

c) リプレースメント・アルゴリズム

リプレースメント・アルゴリズム（新しく必要になったページを入れるべき内部メモリ領域の選び方）のいくつかの方法については前回述べた。ページフォルト率は、これらのうちのどのアルゴリズムを選ぶかにより変化し、ひいてはシステムの効率に影響する。

理想的 (Optimal) あるいは MIN) アルゴリズム¹⁶⁾とは、「将来の最も長い時間にわたって、参照の起らないページを選ぶ」というアルゴリズムであるが、プログラムの進行状況を予知し得ない以上、実現不可能である。したがって実現可能なアルゴリズムでは、現在までのプログラム進行の情報から、再参照が最も遠いであろうページを選ぶことになる。この場合、よりくわしく予想しようとするれば、一般によりコストがかかるといえる。実際のシステムでは、前参照時からの時間を利用する LRU (least-recently-used), WS (working set) アルゴリズムの種々の変形が採用されている。これらは、長い間使用されなかったページほど今後使用される確率も小さいであろう、という想定に基づいている。この想定がプログラムの多くについ

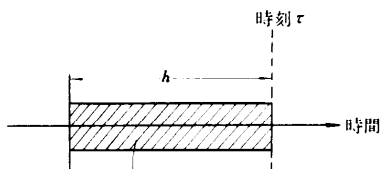
て正しいことは、リプレースメント・アルゴリズムに関する実験で前参照時からの時間を使わないアルゴリズム（例えば、FIFO）に比べ、LRU アルゴリズムの方がページフォルトが少ないという結果からもわかる¹⁷⁾。しかし前参照時からの時間を利用するアルゴリズムでも、その時間の精度は重要でなく、一定時間内に参照のあったものとそうでないものの2分類程度でも、性能は変わらないという結果もある¹⁶⁾。FIFO (first-in-first-out) アルゴリズムでは、ページは内部メモリに持ち込まれた順に出ていくので、頻繁に使われているページでも追い出されてしまう可能性がある。そのためページフォルトに関してはランダム・アルゴリズムと比べても、はっきりした差は認められないようであるが^{13), 16)}、実現が簡単のため多くの実験で用いられている。

d) プログラム構造

プログラムは通常、小さいループ形式を含みながら順次進む。その参照パターンは、かなりの時間にわたって比較的少ないページ内におさまる傾向がある。これを参照の局地性という。（バーチャルメモリはプログラムのこの性質を巧みに利用したものといえる。）しかし、行列の処理のような非連続的なデータ参照の多いプログラムも存在し、ページフォルト率は参照パターンによって著しく変化するので¹³⁾、プログラマは局地性の高いプログラムを作る努力をしなければならない。図 10 はその 1 例である。局地性の高いプログラムを構成する方法については文献 18) を参照されたい。このようなプログラミング技術は効率に対し、リプレースメント・アルゴリズムといったシステムに内在する要因よりも重要であるといわれている¹⁵⁾。

局地性を表わす指標として何をを使うか、はっきりした定義はないようであるが、ここでは局地性を含むプログラムの性質を数量的に表わしている 2つの指標について説明する。

- ① ワーキングセット¹⁹⁾: ワーキングセットとは、プログラムの実行中、一定数 (h) の連続した参照の中に現われるページのすべての集合をいう。図 12 に示すようにワーキングセットは、時刻 t とパラメータ h で指定される。ワーキングセットを構成するページの数にワーキングセット・サイズといい、一般にその数が少ないほど局地性が高いと考えられるが、その場合、ワーキングセットが時間的に急激に変化しないという条件が必要である。



この期間 (h) に参照のあったページがワーキングセット: $W(r, h)$ を構成する。

図 12 ワーキングセットの定義

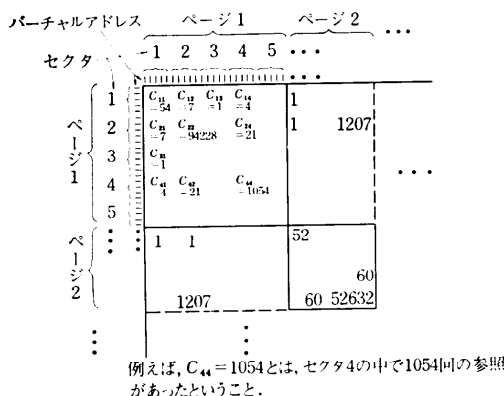


図 13 ニアネス・マトリクスの例

- ② ニアネス・マトリクス¹³⁾: マトリクス要素 C_{ij} は、プログラム (バーチャルアドレス) の i 番目の部分 (セクタ) から j 番目の部分へアクセスが移る回数を表わしている。図 13 にマトリクスの一例を示す。セクタの大きさは、1 語から 1 ページまで考えることができる。ページ X からページ X' へアクセスが移る回数は、

$$V(X, X') = \sum_{i \in X} \sum_{j \in X'} C_{ij} \quad X, X': \text{ページ番号}$$

と表わされる。そこで、異なるページへアクセスが移る回数は、プログラムの全実行の間に、

$$C(X_1 \dots X_P) = \sum_{1 \leq k \neq l \leq P} V(X_k, X_l)$$

P : ページ数

回ということになり、この値を最小にするようなページ割付け (語の並べ替えとページの区切り方) が最適ということになる³⁾。Hatfield^{13), 14)} は、ニアネス・マトリクスを使ってプログラムの自動的な再構成を行ない、局地性の高いプログラムを得ている。

以上、いずれの指標をとってみても、その数値は全プログラムの実行をトレースした後でなければ得られない。

5.2 ページフォルト処理時間

ページフォルトを処理するためにかかる時間は、

$$t = \alpha + \beta + \gamma + \delta$$

α : 補助メモリへのアクセス時間

β : データ転送時間

γ : キュー待ち時間

δ : 割込み処理ソフトウェア時間

と表わされる。この中で最も長いのは α である。IBM 2301 固定ヘッドディスクを使ったバーチャルメモリの実験で、 $\{\alpha + \beta(4k \text{ バイト}) + \delta\} / (\alpha + 1/2\beta + \delta) = 1.14$ という結果がある¹⁴⁾。 $\alpha = 8.6 \text{ ms}$, $\beta(4k \text{ バイト}) = 3.3 \text{ ms}$ であるから、 $\delta = 1.6 \text{ ms}$ と見積もれる。ただし、 δ はページフォルトの状況によって変わる。

補助メモリ自身の効率性は、 $\beta/(\alpha + \beta)$ と考えられ、その値が極端に小さくなるような使い方は適当でない³⁾。ドラム・ディスクの場合、ページサイズを 500 ~ 1,000 語以下にすることはその意味からは望ましくない。

補助メモリにかかる時間を小さくする手法は、いろいろ考えられている。回転数を上げたり、ヘッド数を増やしたり、ビット密度を上げたりするハードウェア上の改善の他に、方式上の工夫もある。例えば、キュー待ち時間短縮のために、FCFS (要求順アクセス) ドラムに代る最短アクセス優先のページングドラムとか、複数のドラムに並列的にアクセスするインターリーブ方式などがある。また、機械的な回転機構をともしないバブル・メモリなども研究されているが、ここではメモリ上の情報配列を工夫してアクセス時間を短縮することも期待できる²⁰⁾。

6. マルチプログラミングおよびタイムシェアリングの場合

6.1 マルチプログラミング

マルチプログラミングの目的の一つは CPU 効率を上げることにある。まず、CPU 効率について見てみよう。Gaver²¹⁾ は確率モデルを用いてマルチプログラミングの CPU 効率を解析している。バーチャルメモリの場合、Kuck ら⁹⁾ はプログラムに内在する I/O 要求が、ページフォルト分だけ増加した場合に等しいと考えて、

$$\lambda = r + f$$

λ : 平均 I/O 要求率

r : プログラム内在 I/O 要求率

f : ページフォルト率

を Gaver の結果に代入して、マルチプログラミングの CPU 効率を求めている。内部メモリ容量を M ページ、マルチプログラミング・レベルを J とし、それらは同じプログラムで、同じ容量 (M/J) のメモリ割り当て量があるとする。 f は (1) 式 ($f=1/e$) より与えられる。一例として、Fine らの結果²²⁾ (ページサイズ 1k 語の場合で、14~44k 語の 5 種類のプログラムの平均) を利用して、

$$f = \frac{1}{3.8(M/J)^{2.4}}$$

M : 内部メモリページ数

J : マルチプログラミング・レベル

と仮定している。図 14 は、 $M=64$ ページ、 $r=10^{-4}$ (10^4 ステップに 1 回の I/O 要求) の場合の例である。この結果は f の仮定に依存するので数値的な意味は余り持たないが、あるマルチレベルまでは効率は上昇するがそれ以後はページフォルトのために低下することがわかる。図 14 の極大の尾根の効率を単一プログラムの場合との比に直し、 $T(=t/d)$ につきプロットしたのが図 15 である。パラメータ r, M を変えているが、効率に対する内部メモリ容量の与える影響が大きい。

図 16 は、バーチャルメモリのもとでのマルチプログラミングの効率に関する Brawn ら¹⁵⁾の実験結果の一部であるが、そこで使われているプログラムは図 10 で用いたものと同じプログラムである。図 10 から、

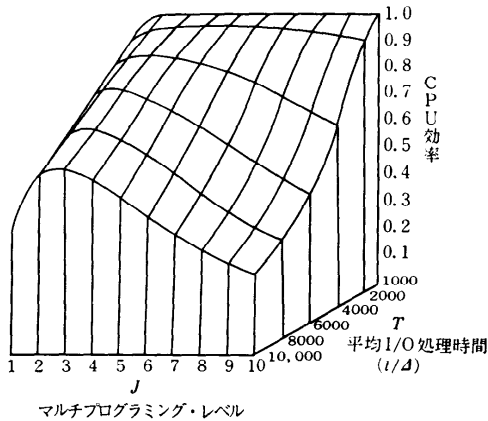


図 14 CPU 効率とマルチ・レベル、I/O 処理時間の関係
(内部メモリ 64k 語、ページサイズ 1k 語、ページフォルト率 $1/(3.8(64/J)^{2.4})$ 、プログラム内 I/O 要求率 $1/10,000$ の場合)

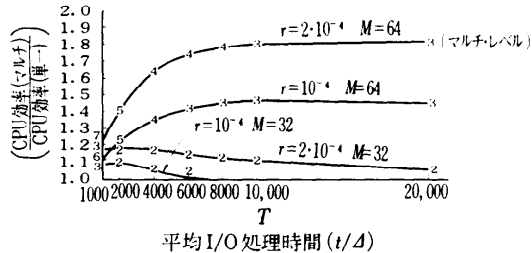


図 15 マルチプログラミングにおける CPU 効率の上昇

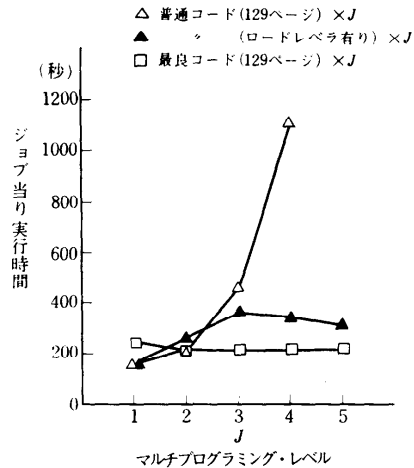


図 16 マルチ・レベルとジョブ当り実行時間の関係
(内部メモリ容量 184k 語、ページサイズ 1k 語、ソート・プログラム、BIFO リブレースメント・アルゴリズム、0.1 秒タイムスライスの場合)

効率のよい実行に必要なメモリ割り当て量は、1ジョブ当り、最良コードで約 20k 語、普通コードでは約 110k 語と予想される。図 16 の結果はその予想を裏付けている。すなわち普通コードでは、マルチ・レベルが 3 以上になると各プログラムへの割り当て量が 62k 語以下になり効率が急激に悪くなる。これはジョブ当りのメモリ割り当て量が一定数以下になると、ページフォルトが急激に増加し、ページ待ちの状態が起きるようになるからである。この現象をスラッシングといい、このようなオーバーロードを防ぐために、システムはページフォルト頻度と CPU 効率をモニタして、それらがある設定値を超えたときには、ジョブの 1 つを一時的に CPU 待ち行列からはずし、そのジョブが使用していた内部メモリのページ枠を他の活動状態にあるジョブに解放し各ジョブのメモリ割り当量を大き

くする。この機構をロードレベラあるいはスラッシングモニタという。図 16 では普通コードのマルチレベル 3 以上でその効果が著しい。スラッシングモニタとは各ジョブに対し f がある設定値以上悪くならぬようメモリ割り当量を調整する機構ということができる。

6.2 タイムシェアリング

タイムシェアリング・システムでは、多数のユーザのプログラムが同時に処理されるので、内部メモリ内でのプログラムの位置を固定せずに実行できるような機能が必要になる。また、多数のユーザに対しては内部メモリ容量だけでは不十分であるので、補助メモリとの間の転送が必要になり、それも自動的に管理される。このような動的なメモリ管理を実現する方式としては、ベースレジスタ方式とバーチャルメモリ方式がある。この 2 つの方式を比較した場合一般にバーチャルメモリ方式は、システムの融通性は高いがシステム・オーバーヘッドに問題があるといわれる。このオーバーヘッドは主にページフォルトによるもので、タイムシェアリングの場合は、プログラムの切り換えの都度大きなページスワッピングが生じるためである。デマンド・ページングはメモリ容量が少なくすむかわりにページフォルトが多くなり効率が落ちる。そこで、書き換え方式に種々の工夫がなされ、例えば単純なデマンド・ページングではなしに、ある種のプリ・ページングを用いたり、必要なくなったプログラムはその時点で一挙に補助メモリに転送するような方式がとられる。

バーチャルメモリを使ったタイムシェアリング・システムの実測データの報告はいくつかある^{23), 24)}。文献 25) の例では、内部メモリ容量 65 k 語 (うちシステムプログラム 20 k 語)、ページサイズ 256 語、デマンド・ページングに基いた FIFO リプレースメント・アルゴリズムでユーザ数 10~12 のとき、CPU 時間を占める割合はユーザ時間 8~15%、OS 時間 35~45%、残りは I/O 待ちであった。OS 時間の 3/4 がページフォルト処理に関するものであったと報告されている。

文献 26) の例では、ページサイズ 4 k バイト、ユーザ数 1~28 の場合の統計をとっている。その結果として、例えばユーザ数 12 のときユーザ時間 25%、OS 時間 25% であったのが、内部メモリ容量を 1.5 倍に拡大した場合ユーザ時間が 35% と改善された、と報告されている。メモリの拡大によってページフォルト

頻度が減少したためと考えられる。

7. 通常システムとの比較

通常システムとバーチャルメモリ・システムを総合的に比較することは重要である。その評価の対象となるものはいろいろ考えられるが、ここでは、その一部である処理速度とプログラミング・コストについての比較例を紹介する。

Brawn ら¹⁵⁾ は処理速度を、メモリの自動管理とプログラマ管理との間で比較実験している。これは単一プログラムのオーバーレイ処理に関するもので、同じ問題について、デマンド・ページング方式のメモリ管理 (バーチャルメモリ) とプログラマがオーバーレイのための命令を書き込んだプログラムとで実行し、両者の全実行時間を比較している。内部メモリ割り当て量は 21 k~46 k 語の間でそれぞれのプログラムにつき固定し、ページサイズ 1 k 語、FIFO リプレースメント・アルゴリズムのときの結果が表 1 である。表の 1 列、2 列ではプログラマはオーバーレイ処理にあたって、補助メモリへのアクセス時間を利用してプログラムの実行を続けている (オーバーラップ処理)。3 列、4 列はそうではない場合である。既に述べたように、バーチャルメモリの処理効率はコーディング法によって著しく変る。表の普通コードと最良コードはその違いを表わしている。オーバーレイ・プログラムもプログラマの腕によりその処理効率が変わることはもちろんであるが、ここでは普通の時間をかけた普通のプログラムということである。

表 1 プログラム実行時間の比較 (自動メモリ管理/プログラマ管理)
(補助メモリと内部メモリの速度比 21,000,
内部メモリ・サイクルタイム 8 μ s, アドレス変換テーブル用メモリ・サイクルタイム 2 μ s)

バーチャルメモリ 通常システム	普通コード	最良コード	普通コード	最良コード
	オーバーラップ	オーバーラップ	オーバーラップなし	オーバーラップなし
逆行列計算 (200×200)	2.33	1.25	1.47	0.79
データ相関	0.81	0.81	0.71	0.71
クイック・ソート 100,000 語ファイル (4ウェイ・マージ)	>41	1.26	>32	0.99
クイック・ソート 100,000 語ファイル (2ウェイ・マージ)	>27	0.84	>19	0.59
クイック・ソート 1,000,000 語ファイル (4ウェイ・マージ)	—	—	—	1.15

Sayre²⁷⁾ は Brawn らのデータを使って、次のようにまとめている。リプレースメント・アルゴリズムが FIFO の場合、バーチャルメモリ・システムのプログラム実行時間は平均で 21% 程遅くなる (表 1 の 2 列目に相当)。しかし LRU のようなアルゴリズムを使えば、その値は 15% 程度になるだろうといっている。

Sayre はさらに、プログラミング・コストについてもふれている。それによると、コーディング、テスト、デバッグ等全プログラミング・コストの約 30% がオーバーレイ処理のために費やされる。バーチャルメモリの場合も前述のようにいくらかの配慮は必要であるが、それはプログラミング・コスト全体の 5% 程度ですむ。プログラムのすべてがオーバーレイを必要とする程大きくはないので、そこで得られるコストの節約は 15% ぐらいになる。また、メモリ容量などが変わったときに再びプログラムする手間が省けるが、これはコンピュータ生産者で 30%、ユーザで 10% ぐらいのコスト節約と見積もれる。そしてプログラミング・コストの全体での 25~45% の節約は、バーチャルメモリ自身のコストおよび効率低下を帳消しにするであろう、といっている。これらの数値が一般的なものかどうかはわからないが、システムの評価のための一端としてここでとり上げた。表 1 のような比較は OS の効率も関係し正確な比較は難しい面もあると思われる。

8. おわりに

今回はバーチャルメモリの機構を、今回はその効率を中心に見てきた。バーチャルメモリの特徴をまとめると次のようになるであろう。

プログラマ側から見た場合には、

- 1) プログラムとデータのために、大容量のアドレス空間を使うことができる。オーバーレイなどの特殊な工夫はいらなくなり、問題解決に専心できる。
- 2) システムに依存しないプログラムが書ける。プログラムの互換性が増す。

以上の点は、プログラミングの生産性向上として評価されよう。ただし、プログラミングに際しての留意点がいくつかある。

- 1) プログラマは自分のプログラムが実行されるバーチャルメモリ・システムの機構を理解する必要がある。
- 2) なるべくプログラムの局地性を上げるように努

力する必要がある。

次にシステムとして見た場合には⁴⁾、

- 1) メモリ内の情報配置の融通性が増し、メモリが動的に管理される。
- 2) 不必要な情報は内部メモリへ持ち込まれないため、メモリの使用効率上がる。
- 3) プログラムの実行が一部ロードされただけで可能になる。
- 4) 共有プログラムの利用が容易になる。

既に見てきたように、適切に設計されさえすればページフォルトは十分小さくできるようであるが、空間的オーバーヘッド (内部メモリに占めるレジデント・プログラム、アドレス変換テーブルなどの割合) が大きくなることもあって、ある程度以上の大きな容量の内部メモリが必要となる。

バーチャルメモリがシステムの大きな改善と考えられるかどうかは、この方式がもたらすシステムの融通性と時間的、空間的オーバーヘッドの兼ね合いをどう評価するかにかかっており、その場合コンピュータの使用法とプログラミング・コストの評価が重要になるであろう。ともかく、コンピュータ技術の発達の流れの 1 つはその使い易さを目指しており、バーチャルメモリもそのための一つの方式と考えられる。

参考文献

- 9) D. J. Kuck & D. H. Lawrie: The Use and Performance of Memory Hierarchies: A Survey (Software Engineering, Vol. 1, p. 45, Academic Press, New York (1970)).
- 10) L. A. Belady & C. J. Kuehner: Dynamic Space-Sharing in Computer Systems, Comm. ACM, Vol. 12, No. 5, pp. 282-288 (1969).
- 11) R. W. O'Neill: Experience using a time-shared multi-programming system with dynamic address relocation hardware, Proc. SJCC, pp. 611~621 (1967).
- 12) M. Joseph: An analysis of paging and program behaviour, Computer Journal, Vol. 13, No. 1, pp. 48~54 (1970).
- 13) D. J. Hatfield & J. Gerald: Program restructuring for virtual memory, IBM Systems Journal, Vol. 10, No. 3, pp. 168~192 (1971).
- 14) D. J. Hatfield: Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance, IBM Journal Res. Develop., Vol. 16, No. 1, pp. 58~66 (1972).
- 15) B. S. Brawn & F. G. Gustavson: Program behavior in a paging environment, Proc. FJCC, pp. 1019~1032 (1968).

- 16) L. A. Belady: A study of replacement algorithm for a virtual storage computer, IBM Systems Journal, Vol. 5, No. 2, pp. 78~101 (1966).
- 17) 益田隆司, 他: ページング・マシンにおけるスワッピング・アルゴリズムの比較とプログラムの動作解析, 情報処理, Vol. 13, No. 2, pp. 81~88 (1972).
- 18) R. P. Parmelee, et al.: Virtual storage and virtual machine concepts, IBM Systems Journal, Vol. 11, No. 2, pp. 99~129 (1972).
- 19) P. J. Denning: The Working Set Model for Program Behavior, Comm. ACM, Vol. 11, No. 5, pp. 323~333 (1968).
- 20) 石井 治, 吉広誠一: 記憶階層における磁気バブルの適応分野について, 応用磁気第 137 委員会第 27 回研究会資料, pp. 28~36 (昭 48. 3 月).
- 21) D. P. Gaver, Jr.: Probability models for multiprogramming Computer systems, Journal ACM, Vol. 14, No. 3, pp. 423~438 (1967).
- 22) G. H. Fine, et al.: Dynamic program behavior under paging, Proc. 21st Nat. Conf. ACM, pp. 223~228 (1966).
- 23) W. M. De Meis & N. Weizer: Measurement and Analysis of a Demand Paging Time Sharing System, Proc. Nat. Conf. ACM, pp. 201~216 (1969).
- 24) M. Parupudi: Interactive Task Behavior in a Time-Sharing Environment, Proc. Nat. Conf. ACM, pp. 680~692 (1972).
- 25) 益田隆司, 他: セグメンテーション機構を有するタイムシェアリングシステムの解析, 電子通信学会論文誌, Vol. 54-C, No. 9, pp. 843~849 (1971).
- 26) Y. Bard: Performance criteria and measurement for a time-sharing system, IBM Systems Journal, Vol. 10, No. 3, pp. 193~216 (1971).
- 27) D. Sayre: Is Automatic "Folding" of Programs Efficient Enough to Displace Manual? Comm. ACM, Vol. 12, No. 12, pp. 656~660 (1969).

(昭和 48 年 8 月 2 日受付)