Vol. 16 No. 12　　情　報　処　理　　Dec. 1975

論　文

仮想メモリシステムにおけるプログラムの局所性とその最大化*

益田隆司** 塩田博行**

Abstract

It has been shown that locality of program reference can be improved by reorganizing it, based on relocatable program sectors. However, more microscopic analysis of internal structures of these sectors has revealed low density of memory utilization of their internal codes in many cases.

In this paper, an idea to express memory utilization of each sector is proposed and experimental data for about 600 relocatable program sectors of our virtual operating system and compiler ave given. Also are described some basic properties of program locality. From these analysis some programming techniques to increase memory utilization are also given.

1. まえがき

仮想メモリ方式による計算機システムの性能を向上させるための方策が種々考察されている．これらを分類すると以下のようである．

(1) ハードウェア面：ページ・サイズ，アドレス変換の方式，等．

(2) ソフトウェア面：タスク，メモリのスケジューリング方式．

(3) 利用面：与えられたハードウェア，ソフトウェア・システムのもとでの利用方式．

仮想メモリ方式の実験，研究段階の時期から，(1)，(2)，特に(2)に関する報告が数多く発表されている[8),9),10)]．その実用化に伴ない，(3)による性能向上策がいくつかの面からとられている．(3)を細分すると現在までに次のようなものがある．

(a) 開発済プログラムの再構成可能な単位での再構成法[1),2),5),6)]．

(b) 仮想メモリシステム向きの数値計算法，特に大次元行列の取り扱い法[11)]．

(c) 仮想メモリシステムのもとでユーザがプログラムを作成する際の留意すべき点．

プログラムが一定時間内に利用する異なったページの集合をワーキング・セット[4)]とよび，時間に対してこの大きさが小さいプログラムを局所性のよいプログラムとよぶならば，局所性のよさがシステムの性能に大きな影響を与えることが知られている．(a)～(c)はいずれもプログラムの局所性を上げることを狙っている．

本論文は(a)を更に発展させたものである．(a)では，プログラムを再構成可能な最小要素であるセクタを単位として，実行時に時間的に近いところで利用されるセクタは，できるだけ同一のページに集めるように並べかえることにより，プログラムの局所性を上げることを目的としている．ここでは，各セクタ内のメモリ使用効率には全く触れていない．本論文では，これを更に発展させ，各セクタごとのメモリ使用効率を定義し，プログラムの局所性を把える新しい考え方を提案した[3)]．そして，メモリ使用効率の低いセクタについては，それを向上させるための標準手続きを与えている．同時に，仮想メモリシステム下で動作するプログラムのメモリ使用効率に関する基本的な性質をいくつかの側面から求めた結果を報告する．さらに，これらのメモリ使用効率に関するプログラムの動作特性の分析結果から，新しいプログラムを開発する場合に，その局所性を上げるために留意すべき点（セクタの標準的な大きさ，プログラムの書き方等）に対して，1つの定量的な指針を与える．

これらの結果は，特に，繰り返し多く利用され，また，プロシージャ部のウェイトが大きいプログラム(システム・プログラム等)の局所性を上げるために有

* Program Locality and Its Maximization in Virtual Storage Systems by Takashi MASUDA and Hiroyuki SHIOTA (Systems Development Laboratory, Hitachi, Ltd.)

** (株)日立製作所システム開発研究所

効である.

2. ページ内におけるメモリ使用効率

この章では，仮想メモリシステムの下で動作するプログラムの，実行時のページ内におけるメモリ使用効率について述べる.

1ブロックの大きさをbバイト，プログラム実行中の時刻 $t-T\sim t$ 間で使用される異なったブロックの数を n_t とすると，この間で使用されるメモリ量は，

$$w_b(t, T)=n_t \cdot b \text{ バイト}$$

となる．bをページ・サイズにとると，n_t は通常のワーキング・セット・サイズに一致する．$w_b(t, T)$ の時間平均を $\overline{w_b(T)}$ で表わす.

ここで，本論文で分析の対象としたモデル・プログラムについて述べておく．本論文の内容は，特にシステム・プログラムのように，繰り返し多く利用され，プロシージャ部の占める割合の大きいプログラムに対して有効であるので，システム・プログラムとして最も代表的な管理プログラム，言語プロセッサを分析の対象とした．具体的には，HITAC 8700/8800 システムの仮想メモリ用の OS である OS 7 の管理プログラム（以下 CP と記す），FORTRAN コンパイラを選んだ．これらのプログラムは，仮想メモリシステム向きに開発されたプログラムであり，開発時に人手により局所性を上げるための努力がある程度なされている.

いずれのプログラムについても，ある標準問題を設定し，その実行過程，コンパイル過程のアドレス軌跡を求めた．使用されたセクタ(再構成可能な単位)数，セクタ・サイズの平均値，実行命令数を **Table 1** に示す．また，FORTRAN について，使用されたセクタの大きさの分布を **Fig. 1** に示す（紙面の都合で，CP のそれは略す）．両者のモデル・プログラムで，セクタ・サイズの平均が数倍異なるのは，設計時のプログラム作成法の違いによる．FORTRAN では，ルーチンとよばれるプログラムの最小単位を1つのセクタとしてプログラムを構成しているのに対して，CP では，平均的に，互いに関連を有するいくつかのルーチンで1つのセクタを構成している.

Table1 Programs to be analyzed

	number of sectors used.	average sector size. (bytes)	nunber of instructions executed
FORTRAN	286	496	880,693
CP	320	1,662	1,611,193

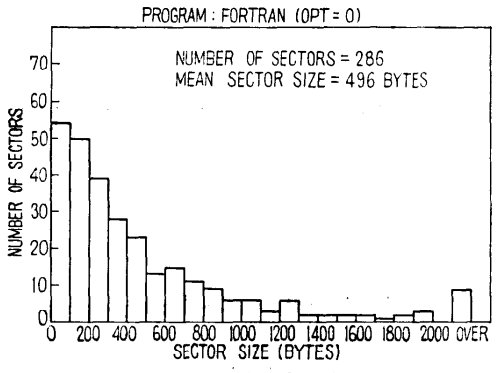


Fig. 1 Sector size of FORTRAN compiler

これら2つのプログラムについて，$\overline{w_b(T)}$を種々の b, T の値に対して求めた結果を **Fig. 2** (次頁参照)に記す．たて軸は 4K バイトが1単位である．尚，同図は，Table 1 に示す全区間より求めたものではなく，FORTRAN の場合には，最初の約 35 万命令（構文チェック，中間語作成のフェーズ），CP では，適当な範囲の 50 万命令から求めた．この理由は，第1に計算時間を非常に要すること，第2に，このようなミクロな分析は全範囲を行わなくても傾向を把握するには十分であるためである.

Fig. 2 は，プログラムの実行時のページ内でのメモリ使用効率に関する基本的な特性を非常にミクロなレベルで表わしていると考えられる．同図を見ると，CP の $\overline{w_b(T)}$ が FORTRAN に比較し，はるかに大きい．また，CP のそれは，window size T が大きくなっても飽和していない．これは，管理プログラムとコンパイラの一般的な特性であると考えられるが，この違いは **Fig. 3** (次頁参照)により明確に表わされている.

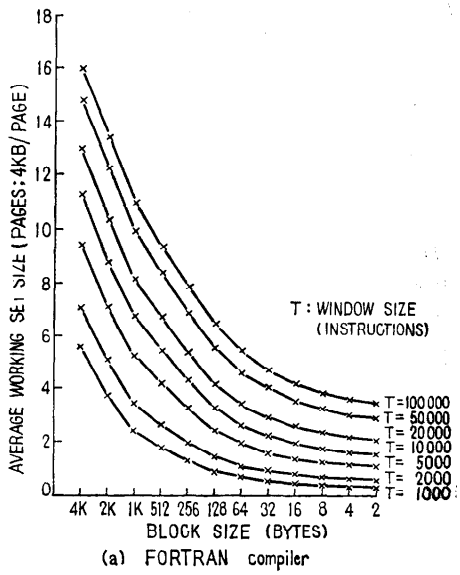
次に，各ページ内におけるメモリの使用効率を求めて見る．プログラムがあるまとまった処理をするためには，最小 32 バイト（約8命令：最小のセクタの本体部分がこの程度の大きさである）程度は必要であると考えると，window size $T=5000$ で比較して，FORTRAN の場合，

$$\frac{\overline{w_{4096}(5000)}}{\overline{w_{32}(5000)}}=\frac{9.47\times 4096}{203\times 32}\simeq 6.0$$

CP の場合，

$$\frac{\overline{w_{4096}(5000}}{\overline{w_{32}(5000)}}=\frac{20.4\times 4096}{384\times 32}\simeq 6.8$$

となる．すなわち，ページ・サイズを4K バイトとすると，$T=5000$ では，使用されている各ページ内を32 バイト単位のブロックで見た場合，実際に使用さ



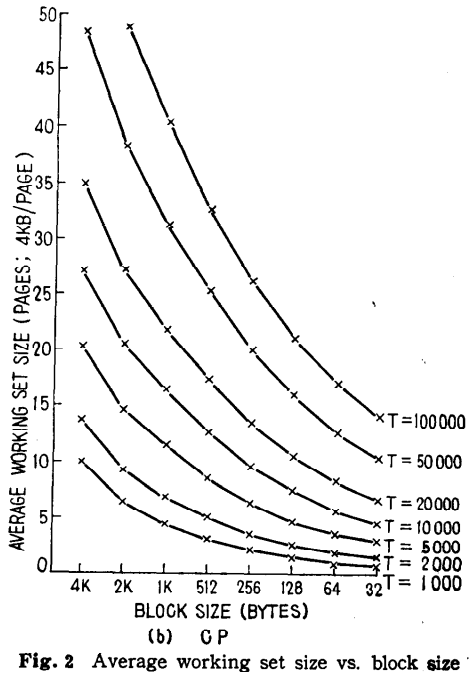


Fig. 2 Average working set size vs. block size

れるブロックの割合は，1/6～1/7 のみである．

この割合を如何にして大きくするかが，我々の目的である．第1の方法は，使用される各セクタ内のメモリ使用効率を上げること，第2は，関連の強いセクタをできるだけ同一ページに集めることである．次章以降では，主として，この第1の，各セクタのメモリ使用効率について論じ，それを上げるための方式について述べる．

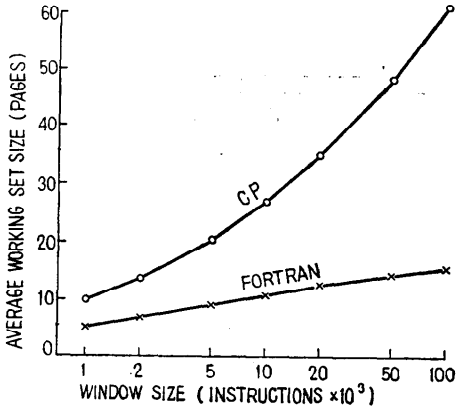
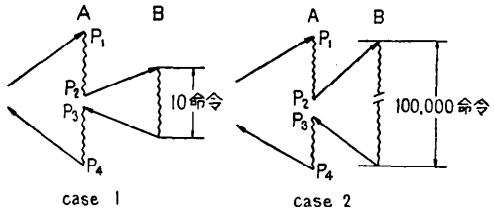


Fig. 3 Average working set size vs. window size

3. セクタのメモリ使用効率

3.1 定　義

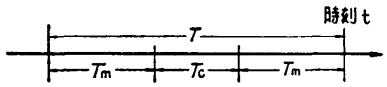
各セクタが利用されたとき，そのメモリ使用効率を定義する．基本的な考え方は，セクタAに制御が移り，Aから実質的に制御が離れるまでに，Aの中で参照されたメモリ量の割合をその時点でのAのメモリ使用効率とする．これをより明確にするために，下図の2つの場合を考えてみる．Aに制御が移り，Aから実質的に制御が離れるところを何処に定義するか，case 1では，AからBが call され 10 命令実行後Aに戻る



ので，P_1～P_4 がその範囲，case 2 では，Bに制御が移ると当分Aに戻らないので，Aから実質的に制御が離れる点は P_2 と考えるべきである．このような曖昧さを除くために，また，我々の最終的な目的は，ワーキング・セットの平均的な大きさを小さくすることにあるので，メモリ使用効率もこの観点から以下のように定義する．

観測時刻 t で，そこから T 命令実行分，下図に示すように過去をふりかえり，その中央の T_c 命令の範囲内で利用されている異なったセクタの集合 I_t 内に

含まれるセクタを，時刻 t での測定対象セクタとする．



時刻 t におけるセクタ i のメモリ使用効率 D_i を次のように定義する．

$$D_i(t,T,T_c,b)=\begin{cases}\dfrac{u_i(t,T,b)}{s_i(b)}\times 100 & \text{for } i\in I_t\\ 0 & \text{otherwise.}\end{cases}$$

ここで，$s_i(b)$ は，セクタ i を b バイト単位で数えたときのブロック数，$u_i(t,T,b)$ は，時間範囲 $t-T\sim t$ の間で，セクタ i を b バイト単位に見たときに使用されたブロックの数を表わす．

観測対象を T_c 内で利用されているセクタに限るのは，T_m を適当に大きく選ぶことにより，window size を T としたときの境界の影響を除くためである．

観測点間の巾を k としたとき，セクタ i の時間的な平均メモリ使用効率を次のように定義する．

$$\langle D\rangle_i=\frac{1}{z_i}\sum_{l=1}^{[N/k]} D_i(kl,T,T_c,b)$$

ここで，N は観測対象とした全実行命令数，z_i は，

$D_i(kl,T,T_c,b)\neq 0$ for $l=1,2,\cdots[N/k]$

の個数を表わす．

$\langle D\rangle_i$ は，セクタ i（大きさ s_i）が利用されるときに着目し，それを window size T の範囲で観測したとき，s_i の内のどの程度が利用されるかの平均的な使用効率を表わしていることになる．

3.2 結果とその評価

上記の定義にもとづいて，前節に述べたFORTRANコンパイラ，CP の各セクタについて，メモリ使用効率 $\langle D\rangle_i$ を求めた．$T=5000$, $T_m=2000$, $T_c=1000$, $k=1000$, $b=32$ として結果を求めた．これらの値の設定は，これまでの報告[1),5)]からも，上記程度の値が現実のシステムでは妥当である．$T=5000$ は，1つのセクタ内での走行命令数を求めるのに一般には十分な安全側の値であり，また，$b=32$ バイトは，2. と同様である．

Fig. 4 は，FORTRAN の $\langle D\rangle_i$ を求めた結果である．図内の各番号が個々のセクタ，折線はセクタ・サイズ 100 バイトきざみでの $\langle D\rangle_i$ の平均値である．紙面の都合で，CP の結果は省略する．

Fig. 4 の第1の利用法は，開発済のプログラムのセクタのメモリ使用効率の改良に用いることである．同図の例では，$\langle D\rangle_i$ が 10% 以下のものが5個，20%以下のものが 24 個存在している．たとえば，#603 のセクタ（大きさ 1092 バイト）は，$\langle D\rangle_{603}=5.7\%$

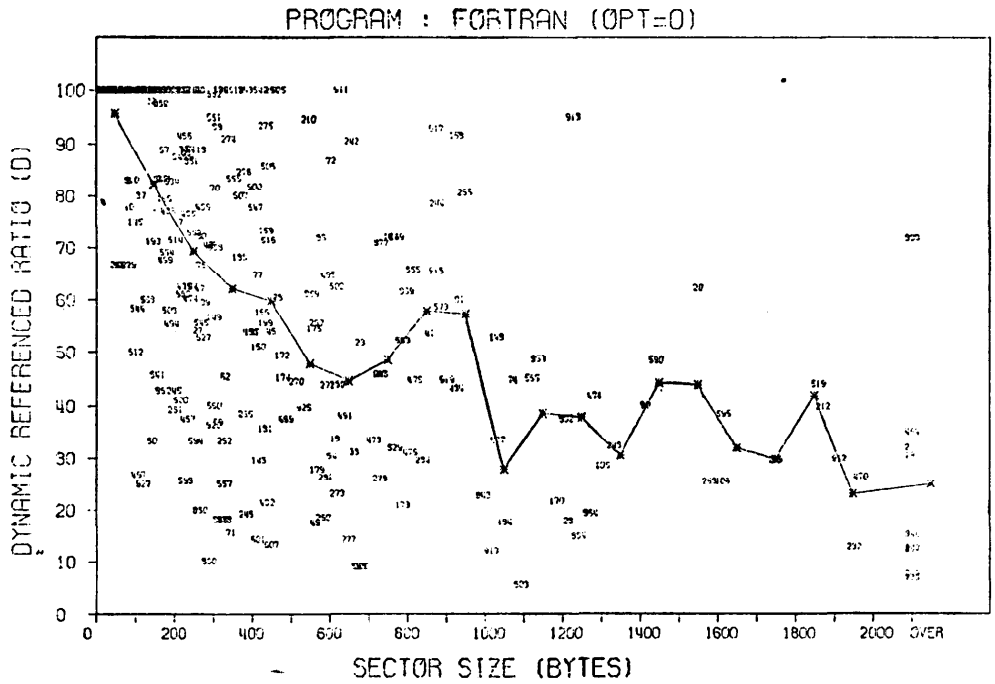


Fig. 4 Memory utilization of sectors

であり，平均的には，35 ブロックのうちの 2～3 ブロックのみが利用されている．$\langle D\rangle_i$ が小さいセクタは，多くの場合，1 つのセクタ内にいくつかの機能を同時に含んでおり，ある条件でそのセクタが利用された場合には，それらの機能のうちの一部のみが利用される．したがって，各セクタのメモリ使用効率を上げ，実行時のワーキング・セットの大きさを小さくするためには，各セクタの有する機能をできるだけ単一化し，$\langle D\rangle_i$ の小さいものを無くすることが大きな効果を有する．すでに開発済のプログラムの局所性を上げるには，Fig. 4 のように $\langle D\rangle_i$ を求め，その値が小さいセクタに対しては，セクタの細分割，プログラム構成上の変更等により，$\langle D\rangle_i$ を大きくすることを試みればよい．

FORTRAN, CP について，各セクタの $\langle D\rangle_i$ の結果から，その平均を表わすいくつかの指標を求めてみる．

（1） 単純平均使用率

$$\sum_i \langle D\rangle_i/n : \begin{cases}\text{FORTRAN}=66.4\% \\ \text{CP} \quad\quad =36.7\%\end{cases}$$

ここで，n は使用されているセクタの総数を表わす．

（2） セクタ・サイズによる重み付け平均使用率

$$\frac{\sum_i s_i(b)\cdot\langle D\rangle_i}{\sum_i s_i(b)} : \begin{cases}\text{FORTRAN}=48.0\% \\ \text{CP} \quad\quad =22.8\%\end{cases}$$

（3） セクタの実行命令数，および，セクタ・サイズによる重み付け平均使用率

セクタ i の実行命令数 r_i の割合を w_i，すなわち，$w_i=r_i/N$ として，

$$\frac{\sum_i w_i s_i(b)\cdot\langle D\rangle_i}{\sum_i w_i\cdot s_i(b)} : \begin{cases}\text{FORTRAN}=30.1\% \\ \text{CP} \quad\quad =29.3\%\end{cases}$$

（1），（2）で FORTRAN の値が大きいのは，そのセクタ・サイズが CP に比較し，平均して約 1/3 であり，かつ，セクタ・サイズが小さい方がメモリ使用効率が高いためである．また，次章で述べるように，同一のセクタ・サイズでもコンパイラの方が若干メモリ使用効率が高い傾向にある．

（2），（3）を比較して見ると，CP の値は 22.8% から 29.3% に増加しているのに対して，FORTRAN の値は，48% から 30.1% に減少している．これは，2 つのプログラムを分析して見ると，CP の場合には小さいセクタの実行命令数のウェイトが高いのに対して，FORTRAN の場合には，各フェーズごとに，大きさの大きい，かつ，実行命令数の非常に大きいセクタが 1 つずつ存在し，その影響を強く受けたためである．

（3）の結果は，いずれのプログラムでも，使用されている全セクタのメモリ使用効率の平均は，各セクタの実行命令数，大きさまでを含めて重みづけをし，かつ，32 バイト単位のブロックで観察したとき，約 30 %であることを表わしている．

以上の（1）～（3）の指標は，数値そのものは各プログラムごとに，セクタ・サイズのばらつき，プログラム構成法の違い等からかなり変動する可能性があるが，Fig. 4 に示すような各セクタごとの $\langle D\rangle_i$ を求めたときに，そのプログラムの全体的なメモリ使用効率を把握するための指標として有効である．

4. セクタ・サイズの影響

本章，および，次章では，Fig. 4 の第 2 の利用法として，セクタの大きさとセクタ内でのメモリ使用効率の関係，さらに，メモリ使用効率の低いセクタのプログラム構造等を求め，これらの分析結果から，メモリ使用効率の高いプログラムを作成するために，留意すべき事項を求めてみる．

ソフトウェア・システムを作成する際，セクタ，すなわち，プログラムの最小構成要素の大きさをどの位にすべきかは，各構成要素にもたせるべき機能，効率，バグ発生の割合，保守の容易さ等と関連し，重要な要因の 1 つであるが，ここでは，それをメモリ使用効率との関係から検討する．

FORTRAN, CP の各セクタについて，セクタ・サイズとメモリ使用効率 $\langle D\rangle_i$ の関係を求め，結果を **Fig. 5**（次頁参照）に示す．同図で，ある曲線に沿って，横軸の値を α，縦軸の値を β とすると，$\langle D\rangle_i$ が α% 以上であるセクタの個数の割合が β% であることを表わしている．FORTRAN では，各セクタを大きさによって 4 種類に分類した．32 バイトを 1 ブロックとしたとき，セクタ・サイズ 0～200 バイトでは，全ブロックが参照されるセクタの割合が 65%，セクタ・サイズの 70% 以上参照されるものが 80% を占めている．これに対して，セクタ・サイズ 200～500 バイトでは，全ブロックが参照されるセクタの割合は 15%，70% 以上が約 50% である．さらに，セクタ・サイズが 1K バイトをこえると，50% 以上利用されるセクタがようやく全体の 10% である．このように，セクタ・サイズが大きくなると，一般的にはメモリ使用効

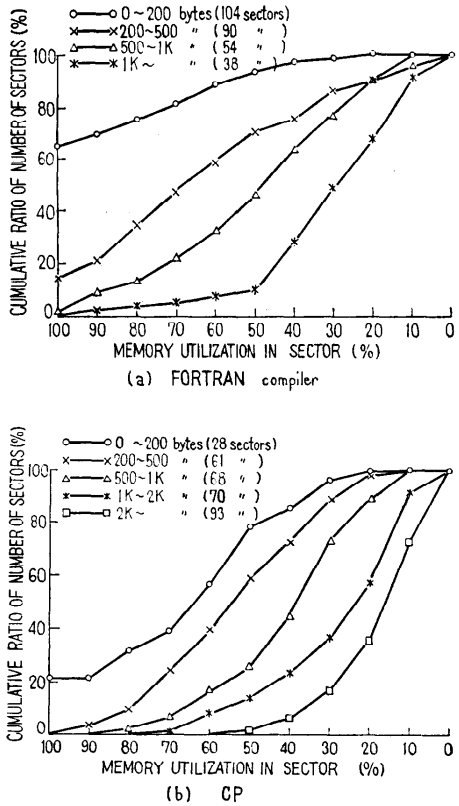


Fig. 5 Sector size vs. memory utilization

率が急速に低下する.

FORTRAN, CP を比較すると，特にセクタ・サイズが 500 バイト以下では，FORTRAN が CP に比較しかなり $\langle D\rangle_i$ が高い．CP の場合，大きさの小さいセクタの数が少ないことの影響も考えられるが，特に，大きさの小さいセクタでは，コンパイラの方が管理プログラムに比較し，より '処理' 中心，管理プログラムの方がより '判定，制御の移動' 中心であり，そのために後者ではメモリ使用効率が低いと考えることができる．セクタ・サイズが大きいところでは差が見られないが，これは，管理プログラムの場合にも，セクタ・サイズが大きいものは，データ管理，ジョブ管理等のセクタに多く，'処理' 中心の傾向が強まってくるためと考えられる.

これらの結果から，新しいシステムを開発する場合に，セクタ・サイズの基準をどの程度にすべきかの目安がメモリ使用効率の観点から得られる．セクタ・サイズが数 100 バイトまでならば，局所性は非常によいが，セクタ・サイズが 1K バイトを越えると，メモリ使用効率はかなり悪く，約 50% のセクタはメモリ使用効率が 10 数%以下である．したがって，できるだけ 1K バイトを越えるような大きなセクタは作成すべきでない．ただし，セクタ・サイズが小さくなると，セクタ間のリンクが頻繁になり，そのためのオーバヘッドが増加する，セクタ・サイズの基準は，これら他の要因をも考えあわせて決めるべきであることは無論である.

5. プログラム作成上の留意事項

仮想メモリシステムのもとで動作するプログラムを作成する際，局所性の高いプログラムを書くために留意すべき事項を，J. E. Morrison がユーザの立場からまとめている[7].

我々は，具体的に，前章までの 2 つのモデル・プログラムを例にとりあげて，メモリ使用効率 $\langle D\rangle_i$ が低いセクタの構造を分析した．$\langle D\rangle_i$ が小さいものを FORTRAN から約 70 個，CP から約 50 個選択し分析の対象とした．その結果，**Fig. 6** (次頁参照) に示すような基本的ないくつかのパターンに分類できることが判明した．図内の点線が通常の処理の場合の流れを表わす．各々について簡単に説明する.

(I) 分析した中で最も多い型である．メモリ使用効率は非常に低い．'判定' C の前に若干の '処理' R を行っているセクタも多い．call されたセクタ側で '処理' P の必要性を判断し，その必要性がなければ直ちに caller にリターンする型である．P がエラー処理の場合，特殊処理の場合，初期設定のため最初の 1 回目だけ P の処理が必要な場合等がこれに属する.

この型のセクタの局所性を上げるには，'判定' C までを caller 側で行うか，または，R がある程度の大きさの場合は，P を独立したセクタにすべきである.

(II) '処理' が P_1, P_2 の 2 つに分かれる場合である．P_2 が P_1 に比較して小さい場合，$\langle D\rangle_i$ が小さくなる．'判定' C までを caller 側で行い，P_1, P_2 を各々独立したセクタにするか，または，P_1 のみを別のセクタにすればよい.

(III) 上記のより一般的な場合である．全体の構成を考慮しつつ，各々の P_i を，または，いくつかの P_i をまとめて，それぞれ独立したセクタにすればよい.

(IV) 他の多くのセクタを call するメイン・プログラムである．この型のセクタの $\langle D\rangle_i$ が小さいのはある程度やむを得ない．ただ，メインプログラムで

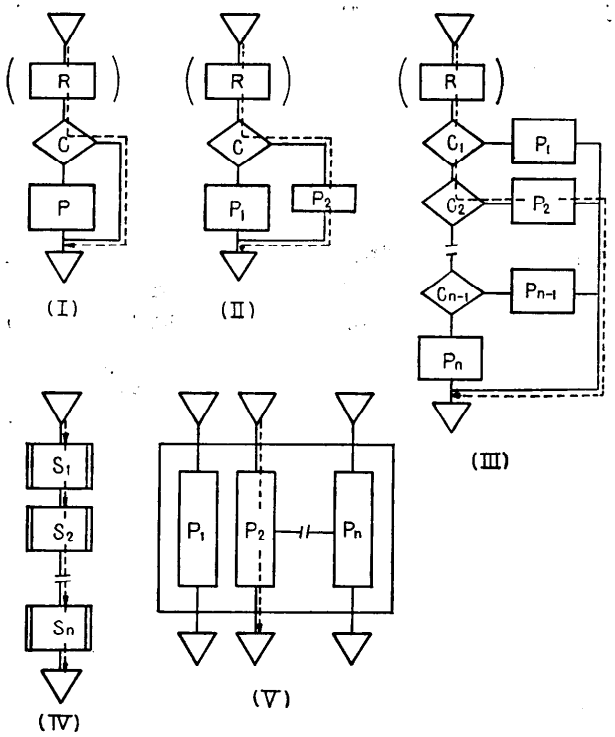


Fig. 6 Program structure of low memory utilization

は，できるだけ内部的な処理は別セクタとし，その大きさを小さくすべきである．

(V)　1つのセクタ内に，相互の論理的な関係は密であるが，プログラム構造的には全く独立ないくつかのプログラムが存在している．個々の使用頻度，相互の時間的な結びつきの強さに合わせていくつかに分離した方がよい．

$\langle D \rangle_i$ が小さい多くのセクタは，これらの型のいずれかに属するが，いくつかの型が組み合わされているようなセクタも存在している．

前章の内容も合わせて，プログラム作成法の見地から以上を更にまとめると，

(1)　各セクタはできるだけ単一の機能とすること，

(2)　論理的にまとまった"処理"を行うべきか否かの"判定"は，caller 側で行うこと，

(3)　セクタの大きさは，原則的にはできるだけ小さくすること，

(4)　1K バイトを越えるセクタは，局所性を十分意識したプログラム構造とすること，

等に要約できる．これらにしたがって，局所性の良いセクタができれば，次のステップとして，セクタ単位の再構成を行うことにより，プログラム全体の局所性を高めることができる．

6. セクタの細分割によるワーキング・セットの最小化

これまでに，セクタのメモリ使用効率について種々論じ，また，メモリ使用効率の悪いセクタについてはそれを機能単位に細分割することにより，使用効率の向上が期待できることも述べた．本章では，このようなメモリ使用効率の悪いセクタをいくつかに分割することにより，各セクタのメモリ使用効率を上げ，その後，それらを再構成することにより，全体としてそのプログラムのワーキング・セットの大きさがどの位まで小さくなるかを実験的に求めた結果を報告する．

FORTRAN コンパイラを例にとり，その $\langle D \rangle_i$ が 65% 以下のセクタを細分割の対象とし，細分割後の各セクタの $\langle D \rangle_i$ を 65% 以上に引き上げることを原則として，各セクタごとのトレース・データ，アドレス参照の分析結果から，セクタの細分割を行った．この細分割は，機能的な分割になっておらず，プログラムの構造は破壊するが，ワーキング・セットがどの位小さくなるかの1つの目安が得られる．

細分割後のセクタにもとづいて，セクタ間の関連度行列 $\boldsymbol{R}$ を求め，修正平均法[1]によってクラスタリングを行うことによって，各セクタのページへの割り当てを実施した．これによって求めた平均ワーキング・セット・サイズを，他のものと合わせて，**Fig. 7** (次頁参照) に示す．図中の各場合について説明する．

(1)　randomized: 細分割前の状態で，セクタの順序をランダムに並べかえる．

(2)　original: もとのプログラムのままの順序づけ．

(3)　WS & MA[1]: 細分割前の状態でセクタを並べかえる．セクタ間の結びつきについては，ワーキング・セットの考えにより，かつ，セクタのページへの割り当てには，クラスリング法として修正平均法を利用する．

(4)　partition: 本章の結果であり，メモリ使用効率の悪いセクタはいくつかに細分割し，細分割後のセクタのメモリ使用効率を 65% 以上に引

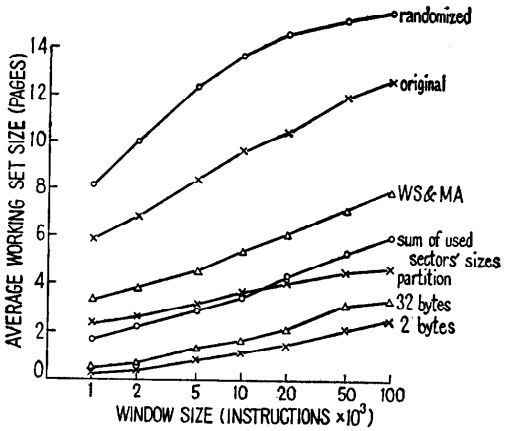


Fig. 7 Comparison of avarage working set size～FORTRAN compiler

き上げ，その後，WS & MA 法によりクラスタリングを実施する．

以下の3つは，ワーキング・セット・サイズの最小化を行う際の目安となる．

(5) sum of used sectors' sizes: セクタを細分割する前の状態で，window size T 間で使用される異なったセクタの大きさの和の時間平均．同一のセクタを多重におくことを許した際のワーキング・セット・サイズの最小値に対応する．

(6) 32 bytes: プログラムを32バイト単位のブロックで見たとき，window size T 内で利用される異なったブロックの総数の時間平均をページ・サイズで基準化したもの．

(7) 2 bytes: 上に同じ (32 バイトの代りに2バイト)．実際に参照，利用したバイト数を表わす．

window size T=5000 でみると，本章で述べたセクタの細分割後の再構成は，細分割前でのWS & MA 法によるクラスタリングよりもかなり効果が大きく，ワーキング・セット・サイズが original ordering に比較し，38% となっている．メモリ使用効率のよいセクタを作成することが，プログラム全体の局所性を高めるために大きな効果を有していることが分かる．

7. むすび

本論文では，第1に，プログラムの局所性をセクタを単位として把える新しい考え方を提案し，その局所性を上げるための標準手続きを与えた．

第2に，実際のモデル・プログラムの分析結果から，プログラムの局所性に関するいくつかの性質を把握した．さらに，そこから，仮想メモリシステムに向いたプログラムを書くためのいくつかの留意点を得た．

本論文で選んだ2つのモデル・プログラムからの結果については，それらが経験あるシステム設計者により設計され，また，分析の対象とした約600個のセクタは大勢のプログラマによって作成されていることから，その一般性は十分保証できると考えられる．他の1つのコンパイラについての各種の分析結果もFORTRAN の結果と非常に近いことを確かめている．

本論文は，特に，プログラムのプロシージャ部のワーキング・セットの大きさを小さくするために有効であるが，データ部については，セクタの概念がプロシージャ部のように明確でないことが多い．一般のユーザ・プログラムを含めて，データ部の局所性をどのようにして高めるかは今後の課題である．

最後に本研究の遂行に当たり，方式上の検討を賜わった東京大学穂坂衛教授，大須賀節雄助教授，データの分析にご協力戴いた当社ソフトウェアエ場の野口健一郎，大木尚，東常修也，高貫隆司の諸氏，本研究の機会を与えて下さった同永井部長，大西主任技師に厚く感謝致します．

参考文献

1) 益田，塩田：仮想メモリシステム向きの最適プログラム構成方式と実験，情報処理，Vol. 15, No. 9. pp. 662～669 (1974).
2) T. Masuda, et al.: Optimization of Program Organization by Cluster Analysis, Proc. of IFIP Congress 74, pp. 261～265 (1974).
3) 益田，塩田：仮想メモリシステムにおけるプログラムのメモリ有効利用率について，情報処理学会第 15 回大会予稿集，1974.
4) P. J. Denning: The Working Set Model for Program Behavior, Comm. ACM, Vol. 11, No. 5, pp. 323～333 (1968).
5) D. J. Hatfield & J. Gerald: Program Restructuring for Virtual Memory, IBM Sys. J., Vol 10, No. 3, pp. 168～192 (1971).
6) D. Ferrari: Improving Locality by Critical Working Sets, Comm. ACM, Vol. 17, No. 11, pp. 614～620 (1974).
7) J. E. Morrison: User Program Performance in Virtual Storage Systems, IBM Sys. J., Vol. 12, No. 3, pp. 216～237 (1973).
8) 益田，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).
9) L. A. Belady: A Study of Replacement Algorithms for a Virtual Storage Computer, IBM Sys. J., Vol. 5, No. 2, pp. 78～101 (1966).
10) W.W. Chu & H. Opderbeck: The Page Fault Frequency Replacement Algorithms, Proc. of FJCC, pp. 597～609 (1972).
11) 村田，堀越：対称帯行列を三重対角化するための新アルゴリズム，情報処理，Vol. 16, No. 2, pp. 93～101 (1975).

(昭和50年1月30日受付)