原始プログラム構造の記憶管理への利用可能性についての検討†

益田 隆司††

仮想記憶の制御方式に関しては，これまでに数多くの方式の提案，あるいは，開発がなされている．これらの制御方式はすべて，主記憶上に保持すべきページ集合の決定を，プログラム実行時のアドレス参照特性から得られる情報に基づいて行っている．プログラムは，実行時の特性として，局所参照の性質を有していることが多く，各時点で局所参照しているページ集合をどのように精度よく推定するかが，記憶管理の重要な役割である．現在，広く利用されているワーキング・セット法では，ある時刻で局所参照しているページ集合を，その時刻から過去一定のウィンド・サイズで指定される時間の範囲内で参照したページ集合によって推定している．しかしながら，視点をかえてみると，プログラム実行時の局所参照性の原因は，原始プログラム内の構造に関係しているはずであり，われわれは，これまでに，原始プログラム内の繰返し構造がその原因であることを確かめた．そこで，本論文では，この情報を記憶管理に利用することの可能性について論ずる．そして，具体的には，ワーキング・セット法において，あるプログラムに対してウィンド・サイズが小さすぎるために，それが効率よく動作しないような場合に，局所参照しているページ集合を繰返し構造に基づいて推定するような方法を提案し，その効果をシミュレーションによって確かめるとともに，実現可能性についての考察を加えることにする．

1. まえがき

仮想記憶システムのもとでのプログラムの動作特性として，局所参照 (locality of references) の性質が存在することが知られている．これは，プログラムは，その実行中に，それが使用するアドレス空間上を一様に参照するのではなく，時間的に使用する部分が移動するという性質である．そこで，このようなプログラムの局所参照の性質を利用した仮想記憶の制御方式がいくつか提案されている[10]．それらは，プログラムが各時点で局所参照しているページの集合を推定し，その推定量を主記憶上にロードして，プログラムを実行しようとするものである．したがって，各時点で，局所参照しているページ集合が精度よく推定できれば，ページ・フォールトの頻度も少なく，また，参照されないページに主記憶を割り当てる可能性も小さく，主記憶を有効に使用することができる．そこで，プログラムが各時点で局所参照しているページ集合をどのように推定するかが問題である．現在最も広く利用されている推定量は，ワーキング・セット (working set) である．

あるプログラムを実行中の時刻 t におけるワーキング・セット $W(t, T)$ とは，そのプログラムが走行した時間のみに注目し，時刻 t の T 時間前から時刻 t までの間にそれが参照したページの集合として定義される[3]．T は，ウィンド・サイズと呼ばれる．T の値を適当な大きさに選ぶと，$W(t, T)$ が，時刻 t で局所参照しているページ集合のよい推定量になる可能性が大きいということがその根拠である．

記憶管理方式としてのワーキング・セット法は，多重度を構成するアクティブ・プロセス* には，すべて，そのワーキング・セットを主記憶内に保証するものである．

ワーキング・セット法を実現する際の大きな問題は，ウィンド・サイズをどのように設定すべきかである．ワーキング・セットが局所参照しているページ集合の推定量であるという点から考えると，ウィンド・サイズは，実行中のプログラムのその時々のフェイズにあわせて，適当な値が設定されるべきである．ここで，フェイズとは，プログラムが局所参照しているあるページ集合のなかで実行されているとき，そのページ集合内での継続時間と，そこで参照しているページ集合を組み合わせたものであるとする．プログラムはその実行とともに，フェイズが移動していく特性を有する

† Possibility of Using the Structure of Source Programs for Memory Management Strategies by TAKASHI MASUDA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学電子・情報工学系

* 本論文のなかで“プログラム”と書いているもののいくつかは，“プロセス”と書きかえたほうがより正確かもしれないが，他の多くの報告と同様，とくに不自然にならない限り，“プログラム”という表現を用いた．これは，厳密に区別して表現すると，かえってわかりにくい場合が多いためである．たとえば，本文中のワーキング・セットの定義のなかでも，“プログラム”より“プロセス”を用いるほうが正確である．

ものが多い．ワーキング・セット法でのウィンド・サイズは，各時点で，そのとき制御が存在しているフェイズ内で局所参照されているページ集合が入りきる程度の大きさに設定することが望ましい．実行中のプログラムのあるフェイズに関して，ウィンド・サイズの値が小さすぎると，ワーキング・セットのなかに局所参照しているページ集合が入りきらず，ページ・フォールトが頻繁に生じる．その結果，そのプログラムに対する処理の効率がさがるとともに，システム全体の性能も低下することになる．逆に，ウィンド・サイズの値が大きすぎると，ワーキング・セットのなかに，現在局所参照しているページ集合だけでなく，過去に局所参照し，現在は利用されていないページが数多く入る可能性がある．

したがって，ウィンド・サイズは，個々のプログラムごとに，また，そのなかの各フェイズごとに，それぞれに適した値が設定されるべきである．しかしながら，プログラム実行中に，アドレス参照に関する情報のみから，このようなフェイズを正確に検出することはきわめてむずかしい．そこで，現実のシステムでは，ウィンド・サイズとして，システム全体に共通の値を設定していることが多い．そして，システム全体でのページ・フォールト率によって，ウィンド・サイズを調節する程度の制御がなされる．

このような制御方式の妥当性を裏づける報告として，プログラムを実行したときの割当て主記憶量の時間積分値である STP (space-time product) を評価基準にすると，ウィンド・サイズが，最適の値から若干ずれても，STP への影響はそれほどは大きくなく，したがって，システムに共通に利用できそうなウィンド・サイズが存在するというような報告もある[4),6)]．しかしながら，解析のためのサンプル・プログラムの種類が少なく，また，偏っていること，および，ページ参照列に基づいた解析であるので，各プログラムについて，数秒程度の実行時間のページ参照データの解析結果であることなどから，この実験結果によって上記のような結論を下すことはできない．

実測が面倒でこれまではっきりした解析結果の報告はないが，経験的にも，プログラムの各フェイズの継続時間，そのなかの使用ページをひととおり参照するのに要する時間等は，それぞれの場合によって大幅に変動しているはずである．システムに共通のウィンド・サイズでは，効率よく動作しないようなプログラム例は容易に作成することができる．このような点から考えると，システムを共通のウィンド・サイズで動作させることは性能への影響がかなりあるはずである．

個々のプログラムへの最適な割当て主記憶量に関しては，ウィンド・サイズによる制御以外に，プログラムのライフタイム関数上の，原点から最大の勾配をもつ点に対応した主記憶量を割り当てるのがよいという“knee 基準”[5)] 等もあるが，ライフタイム関数を実行時に求めることは現実的ではない．

このようなワーキング・セット法の問題点を改良，あるいは，解決するためには，アドレス参照列内に含まれる情報だけに頼っていたのではむずかしい．より原点に立ち帰って考えてみると，局所参照性等のプログラム実行時の動作に関する性質は，原始プログラム内の構造に起因している要素が多分に存在するはずである．アドレス参照列という完全に物理的な情報だけではなく，原始プログラム内の構造から得られるより論理的な情報が，記憶管理に有効に利用できる可能性がある．これまで，ワーキング・セット法の有効性の検討，あるいは，PFF 法[2)]，DWS 法[11)]のように，やはり，アドレス参照情報だけを用いて，ワーキング・セット法を修正したような記憶管理方式の提案等に関する報告はきわめて数多い．しかしながら，上述のような，ワーキング・セット法の問題点を，原始プログラム内の情報を利用することによって改善しようとする試みは，ほとんど見当たらない．Batson ら[1)]，近藤ら[7)] の報告が数少ない報告である．前者は，記憶管理の具体的な実現可能性の検討までには至っていない．また，後者は，プリフェッチを行うために，原始プログラム内の情報を利用することを試みている例である．

このような背景から，われわれは，アドレス参照の情報だけでなく，原始プログラム内に含まれる情報をあわせて利用することによって，仮想記憶の管理を行うことを試みている．そして，これまでに，原始プログラム内の構造を利用したプログラム動作に関する局所参照モデルを提案し，その有効性を，とくに，アドレス参照情報に基づいた従来の BLI モデル[8)] と比較することによって確かめた[9)]．そこで明らかになったことは，実行時の局所参照の性質は，原始プログラム内のループ構造に起因するということである．

本論文では，この考え方に基づいて，原始プログラム内のループ構造から得られる局所参照情報を利用した記憶管理方式の実現可能性について考察する．とく

に，ワーキング・セット法において，あるプログラムを実行中に，ウィンド・サイズが小さすぎると判断された場合には，そのプログラムに対してだけ，一時的に，局所参照しているページ集合をループ構造に基づいて推定する方法を提案し，その特性をシミュレーションによって評価した結果について報告する．

2. ループ構造を利用した記憶管理方式

2.1 基本的な考え方

システムが共通の標準のウィンド・サイズTで効率よく動作している場合には，ワーキング・セット法にしたがって動作させる．プロセスを構成するプログラムのあるフェイズを実行中に，ウィンド・サイズが小さすぎて，そのフェイズ内で局所参照しているページ集合をワーキング・セット内に含むことができず，そのために，ページ・フォールトが頻度多く発生するような場合を問題とする．1章で述べたように，実行時の局所参照の原因は，原始プログラム内のループ構造に対応づけることができるので，このような場合には，ワーキング・セット内にそのループ構造内で局所参照しているページ集合を含むことができないわけである．多くの場合には，ループの1回の繰返しに要する時間が，ウィンド・サイズよりも長い場合であると考えられる．

このようなときに，標準のウィンド・サイズを大きくすると，他のプロセスでは，ウィンド・サイズが大きすぎて，現在実行中のフェイズ内では使用されないページまでが数多くワーキング・セット内に含まれてしまう可能性がある．また，かりに，ウィンド・サイズを拡げるにしても，どこまで拡げるべきかの根拠がない．

ワーキング・セット法が，局所参照しているページ集合の推定量を主記憶にロードする記憶管理方式であるとの立場からすると，上記のような場合には，ループ構造内で局所参照しているページ集合が把握できれば，それを主記憶上にロードするのがよい．すなわち，問題となっているループ構造の実行中は，そのプロセスに関しては，ウィンド・サイズによる制御を中止して，ループ構造内での参照ページ集合を，局所参照集合の推定量とする方法がよいはずである．この方法を以下，ループ制御と呼ぶことにする．ループの実行終了時には，再び，共通のウィンド・サイズによるワーキング・セット法で制御する．

2.2 ループ制御への切りかえの検出

提案方式を実現するに際しての問題は，ループ制御への切りかえの検出方法である．そのためには，まず，2種類のページ・フォールトを区別する必要がある．第1は，プログラムの実行中，あるフェイズ内に制御があるときに，そのフェイズに入ってから一度は参照されたが，その後，ウィンド・サイズの外に出たとして，ページ・アウトされたページを再参照した場合に発生するページ・フォールトである．ウィンド・サイズが小さすぎるために発生するこのようなページ・フォールトを第1種のページ・フォールトと呼ぶことにする．これに対して，いま一つの種類のページ・フォールトは，第1種以外のもの，典型的には，プログラムの実行開始後はじめて参照されたページに対して発生するページ・フォールト，あるいは，新しいフェイズに入ったときに発生するページ・フォールトである．これを第2種のページ・フォールトと呼ぶことにする．

ループ制御を行う必要があるのは，第1種のページ・フォールト率が大きい場合である．第2種のページ・フォールト率が大きい場合には，むしろ，一時的に，ウィンド・サイズを小さくして，過去のフェイズに属するページをできるだけ早くページ・アウトするような方針をとるべきである[11]．

このように，どちらの種類のページ・フォールトが頻度多く発生するかによって，とるべき対策はまったく逆であるが，通常のワーキング・セット法では，この両者を区別することができない．そして，システム全体でのページ・フォールト率の多少によって，システムに共通のウィンド・サイズを調節する程度の制御がなされるが，期待通りの結果が得られるとは限らない．

原始プログラム内のループ構造から情報を得ることによって，制御が属しているループ構造について，そのなかでの第1種のページ・フォールトの頻度を以下のようにして求めることができる．

実行中のある時点で制御が属しているループ構造は階層構造をなしている[9]．その階層構造の各ループに対して，記憶管理の制御下に，参照ページ表を設ける．あるループに制御が移ったときに，対応する参照ページ表を初期化する．それ以降，そのループ内に制御が存在する間に参照されたページを参照ページ表に登録する．ページ・フォールト発生時に，フォールトを生じたページが，それぞれのループ構造の参照ページ表に登録されているページであるかどうかを調べ，

登録されていた場合には，そのループに関して，第1種のページ・フォールトが生じたと判断する．

この方法は，われわれの目的に対して精度よく第1種のページ・フォールトを識別するが，ループごとに参照ページ表を維持するのにかなりの手間を要する可能性がある．そこで，以下のような近似的な方法を利用することにする．

あるループ構造に制御が移って，その第1回目の繰返しのなかで発生するページ・フォールトは，第2種のページ・フォールトであると考え，第2回目以降の繰返しのなかで発生するページ・フォールトは，そのループに入ってからすでに参照されたページが置きかえられてそれを再参照したもの，すなわち，第1種のページ・フォールトであると考える．これは，あるループに制御が移ると，その第1回目の繰返しのなかで，ループの実行に必要なページは，ほぼ参照されるであろうという推測による．

これに基づいて，あるループ構造に関して，そのなかでの第1種のページ・フォールトの発生率を，ループの繰返し終了時に，そのループに入ってから発生した第1種のページ・フォールトの回数を，そのループの継続時間（仮想時間）で除して求める．その結果，第1種のページ・フォールト率がある基準値より大きくなったときには，そのループに関しては，ウィンド・サイズによる制御をやめて，ループ内での参照ページをすべて主記憶上にロードするようにする．ループ制御を行っているときに，階層のより内側のループが現れたときには，上位のレベルのループ制御をそのまま継続する．

3. 実現の概要

提案した記憶制御方式を実現することを考えてみる．基本として動作するワーキング・セット法は，通常のとおり実現されればよいので，ここでは，ループ制御に関する部分のみを述べる．

提案方法を実現するには，まず，ループへの入り口，ループからの出口，および，ループの繰返しの終了時に，オペレーティング・システム内の記憶管理プログラムに制御が渡されることが必要である．そこで，コンパイラは，原始プログラムをコンパイルするときに，ループ構造の目的プログラム内に，特別なスーパバイザ・コール（SVC: Supervisor Call）命令をループ識別番号を付けて埋め込む．一つの例として，図1は，PASCAL の WHILE 命令をコンパイルしたときの目的コードである．3個の SVC 命令が，それぞれ，順に，ループの入り口，繰返し終了時，および，出口での SVC 命令である．各 SVC 命令の後にある DC1 は，SVC 命令のパラメータであり，原始プログラム内の各ループに対して，コンパイラが割り当てたループ識別番号を表している．それ以外のコードは，WHILE 命令の目的コードである．

各 SVC 命令は，以下のような処理をすればよい．

（i） ループ入り口の SVC 命令：

より高位のレベルでのループ制御にない状態の場合には，このループに入った時刻（仮想時刻）を記憶する．また，このループに関する第1種のページ・フォールトの回数を数えるカウンタの初期化等を行う．高位のレベルでのループ制御中の場合には，何もしなくてよい．

（ii） ループの繰返し終了時の SVC 命令：

ループ制御にない状態で，第1回目の繰返し以外の場合には，ループに入ってからの第1種のページ・フォールトの回数，および，ループに入ってからの仮想経過時間から，前節で述べたように，第1種のページ・フォールト率を計算し，その結果がある基準値 P より大きい場合には，ループ制御に切りかえる．

（iii） ループ出口の SVC 命令：

```
WHILE  test  DO
  BEGIN
  (*  body  *)
  END;
```

```
SVC   1                    (begin WHILE loop)
DC    1                    (loop number)
L     10,1584(0,1)         (if test = false,
LTR   10,10                 exit from WHILE)
BC    8,52(0,14)

(*  object code body of WHILE  *)

SVC   2                    (exit iteration)
DC    1                    (loop number)
BC    15,30(0,14)          (execute next iteration)
SVC   3                    (exit WHILE loop)
DC    1                    (loop number)
```

図1 WHILE 文とその目的コード
Fig. 1 A WHILE statement and its object code.

ループ制御を行っているループの実行が終了したときには，再び，標準のウィンド・サイズによるワーキング・セット法に戻す．そのために，ワーキング・セットの管理は，ループ制御の状態にあるときにも継続して行う．

また，ページ・フォールト発生時には，その時点で制御が存在している階層の各レベルのループに対して，そのページ・フォールトが第1種のフォールトであるかどうかを判断する．第1種のページ・フォールトであると判断されたループに対しては，対応した第1種のページ・フォールト・カウンタの値を1だけ増加させる．

以上の処理を行うための制御情報としては，次のようなものだけがあればよい．

（i） 各時刻で，そのなかに制御が存在するループに関する情報の格納表：

この表には，各時刻で，その時刻でのループの階層の数だけの有効なエントリが登録される．ループの開始，終了に伴い，有効なエントリ数が増減する．それぞれのループに対応して，各エントリ内には，

（a） ループ識別番号

（b） ループの開始時刻

（c） 第1種のページ・フォールト用のカウンタ

（d） 第1回目の繰返し中かどうかを示す標示子

等が含まれる．

（ii） ループ制御標示子：

ループ制御中か否かを標示する．ループ制御中には，そのループ識別番号を記憶する．

4. 実験結果

本章では，提案した方式の効果をシミュレーションによって評価した結果を述べる．先の LC モデルの場合[9]と同様，PASCAL 言語で書かれたプログラムを対象とすることにする．3章で述べた PASCAL コンパイラの修正は，基本的には，LC モデルを実現するために行った修正と同じでよい．

PASCAL で書かれた3種類のプログラム[9]について提案方式の評価を行った．各プログラムのループの構造によってその効果には差があるが，傾向としては似た結果を示している．ここでは，クロス・リファレンスを出力するプログラム XREF[12] を例にして，提案方式の効果を説明することにする．

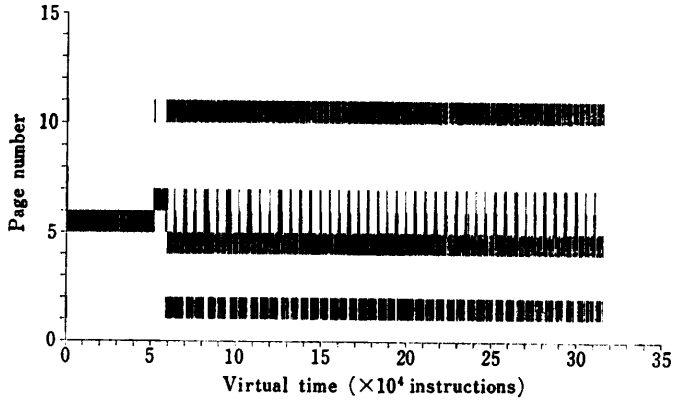

図 2 サンプル・プログラムのページ参照特性
Fig. 2 Page reference characteristics of the sample program.

図2は，最大最小値を求めるプログラムを XREF が処理しているときの，ページ参照の様子を表わしている．横軸は実行命令数であり，たて軸がページ番号である．参照ページを記入するための横軸の最小のきざみ幅は 500 命令であり，各 500 命令ごとに，その範囲で参照されたページに対して，たての線分が書き込まれている．

このページ参照特性に対して，通常のワーキング・セット方式を用いた場合と，それにループ制御を加えた場合（この後者を以下，たんに，ループ制御方式と呼ぶことにする）についての性能の比較を行う．性能の基準としては，割当て主記憶量の時間積分値である STP (space-time product) を用いる．プログラムをある制御方式のもとで実行した場合の STP は次式によって計算される．

$$\mathrm{STP}=\sum_{t=1}^{l} s(t)+\rho\cdot\sum_{j=1}^{k} s(t_j)$$

ここで，l はプログラムの実行時間（実行命令数），$s(t)$ は，時刻 t で主記憶上に存在しているページ数，ρ は，主記憶と2次記憶のあいだで1ページを転送するのに要する時間，t_j は j 回目のページ・フォールトの発生時刻，k は，プログラム実行終了時までに発生したページ・フォールトの回数である．同一プログラムの実行に対して，STP が小さい制御方式がよりよい方式であると考えられる．

図3は，XREF を実行した場合の両方式の STP を表している．横軸はウィンド・サイズである．ρ =50,000（1MIPS の計算機では，50 msec）とした．

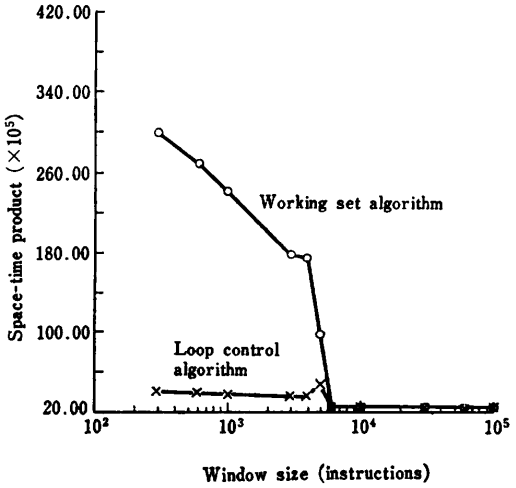

図 3 ループ制御方式とワーキング・セット方式の STP の比較
Fig. 3 Comparing STP of the loop control algorithm with that of the working set algorithm.

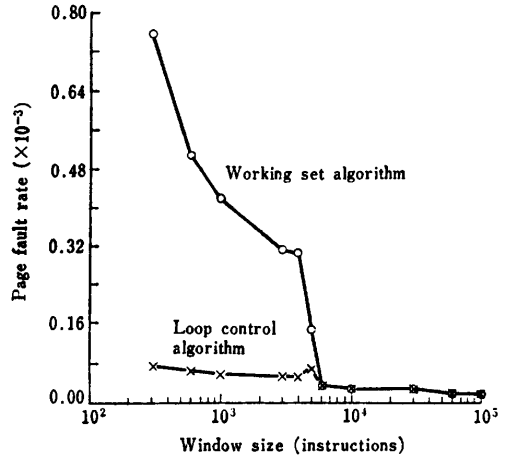

図 4 ループ制御方式とワーキング・セット方式のページ・フォールト率の比較
Fig. 4 Comparing page fault rate of the loop control algorithm with that of the working set algorithm.

また，ループ制御方式の場合，3章で述べた，ループ制御への切りかえのしきい値 P としては，1/10,000 を用いた．

ワーキング・セット方式の場合，ウィンド・サイズが6,000程度以下になると，急速に STP が大きくなっている．図2のページ参照特性からもほぼ読み取ることができるように，ウィンド・サイズが6,000程度以下になると，図2の後のほうのフェイズで参照しているページ集合が，ワーキング・セット内に入りきらない．そのために，図4に示すように，ページ・フォールト率も大きくなり，上式 STP の第2項の影響が大きくなってしまう．これに対して，ループ制御方式の場合には，ウィンド・サイズが小さくなっても，STP の増加は小さく，また，図4に示されているように，ページ・フォールト率もあまり増加していない．ウィンド・サイズが小さすぎると判断された場合には，ワーキング・セット方式からループ制御への切りかえがなされていることがわかる．

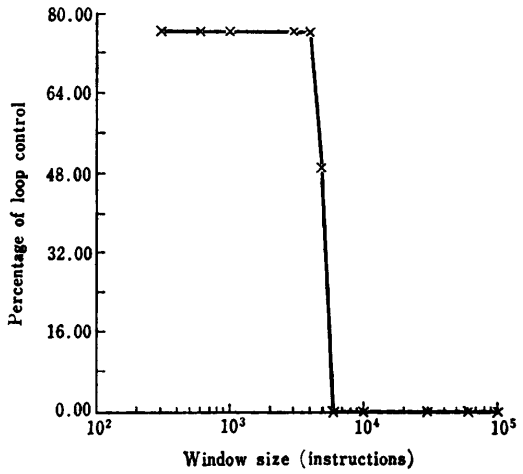

図 5 ループ制御方式でループ制御状態にある時間の割合
Fig. 5 Percentage of loop control time in the loop control algorithm.

ウィンド・サイズが 6,000 程度以上で両方式の STP に差がないのは，ワーキング・セット方式のページ・フォールト率が基準値より小さく，ループ制御が行われていないためである．

これらの状況は，図5から，よりはっきりと理解することができる．図5は，プログラムの実行時間のうち，ループ制御が働いている時間の割合を求めたものである．ウィンド・サイズが 6,000 程度以下になると，ループ制御の割合が急速に増加している．

5. 考　察

(1) 性　能

4章で，提案したループ制御方式の性能をワーキング・セット方式と比較した結果を述べたが，プログラムごとにその動作特性は異なるので，得られた数値から提案方式の定量的な効果を一般的に述べることはむ

ずかしい．現実のシステムでは，ウィンド・サイズも，図3で比較の対象となっている数 msec に相当する程度よりはかなり大きいので，その意味でも現実のシステムとの隔たりがある．

シミュレーションの結果として4章から得られたことは，提案したループ制御方式が，傾向としては妥当な特性を示しているということである．システムに共通のウィンド・サイズでは小さすぎて効率よく動作しないようなプログラムに対して，ループ制御方式は効果を発揮する可能性が大きいことが確かめられた．

（2） 実現のためのオーバヘッド

提案方式をそのまま実現すると，ループへの入り口，ループからの出口，および，ループの繰返しごとに，オペレーティング・システムの記憶管理プログラムに制御を移動させるための SVC 命令が実行される．ループ制御の可能性のない小さなループに対して，毎回このような処理を行うとオーバヘッドが大きくなりすぎる．そこで，このオーバヘッドを軽減するために，以下のような処理を行う．

第1に，コンパイラは，目的プログラムを作成する際に，使用ページ数が静的にも少ないと考えられるループ構造については，実行時にループ制御の対象になるループではないと判断し，SVC 命令の埋込みを行わない．第2には，実行時，最初の何回かの繰返しで，ループ制御が行われる可能性がないと判断されたループについては，オペレーティング・システムは，そのループのループ制御に関する SVC 命令等をすべて，NOP (no operation) 命令に変更する処理を行う．

これらの処理によって，ループ制御方式は，定常的には，ループ制御が必要になる可能性があるループに対してだけ動作させるようにすることができる．また，実際には，そのようなループに対しても，第1種のページ・フォールト率の計算を，各繰返しごとに行わずに，何回かの繰返しに一度だけ行うような目的コードを生成するのがよいと考えられる．

さらに，第3には，ループ制御方式をとり入れた記憶管理方式を利用するかどうかを利用者に選択させるようにするのがよいと考えられる．標準のウィンド・サイズでは小さすぎて，ワーキング・セット法では効率よく動作しない可能性があるようなプログラムは，むしろ，特別なプログラムであると考えられるので，その利用者もシステムの利用に慣れているはずである．ループ制御方式がよりよいと利用者が判断したようなプログラムに対してのみ，利用者がコンパイラにその情報を提供するようにすればよい．コンパイラは，そのような指定があったプログラムに対してだけ，ループ制御の可能性をもたらす SVC 命令を生成する．

以上のような処理によって，ループ制御方式の実質的なオーバヘッドを，かなり小さく抑えることが可能になると考えられる．

6. むすび

原始プログラム内のループ構造を利用して，局所参照しているページ集合を推定し，それに基づいて記憶管理を行うループ制御方式を提案した．そして，ループ制御方式が，従来のワーキング・セット法で，ウィンド・サイズが小さすぎる場合性能が低下するという問題点を改善できる可能性があることを，ページ参照列に基づいたシミュレーションによって，近似的に確かめた．従来，仮想記憶の管理に関しては，数多くの方式が提案されたにもかかわらず，そのほとんどが，アドレス参照列内の物理的な情報のみを利用することを考え，より高位のレベルにある原始プログラム内の構造に含まれている情報を利用することを考察の対象に入れていなかった．主記憶装置は，急速に大容量化する傾向にあり，場合によっては，主記憶の利用に若干の無駄が発生しても，本論文で述べたような，より論理的なレベルからの情報に基づいた記憶管理方式は，考慮の価値があるのではないかと考えられる．今後，その使用が大きな比重を占める傾向が強いデータベース・システムの動作環境等では，とくに，その可能性が強い．

本研究に関連した今後の課題としては，ループ制御方式の有効性をさらに確かめることである．それを行うための最もよい方法は，ループ制御方式を実システム上で実現し，数多くのプログラムの動作環境での実測を行ってみることである．そうすることにより，同時に，ワーキング・セット法が，現実のシステムでどの程度効率よく動作しているかに関する基本的なデータも得られるのではないかと期待される．

謝辞 最後に，本研究の動機となった，原始プログラムの構造を利用した局所参照モデルについての共同研究者であり，現在，日本アイ・ビー・エム サイエンス・インスティテュートに所属する Tong-Haing Fin 氏，本論文で提案したループ制御方式のシミュレーションを実施してくれた筑波大学工学研究科倉田

克彦氏に厚くお礼を申し上げる．なお，本研究は，一部，文部省科学研究費（課題番号 56460184，および，56380007）による．

参考文献

1) Batson, A. P., Blatt, D. W. E. and Kearns, J. P.: Structure within Locality Intervals, in Beilner, H. and Gelenbe, E. (Eds.), Proc. Symposium on Modeling and Performance Evaluation of Computer Systems, North-Holland, Amsterdam, pp. 221-232 (1977).
2) Chu, W. W. and Opderbeck, H.: The Page Fault Frequency Replacement Algorithm, Proc. FJCC, pp. 597-609 (1972).
3) Denning, P. J.: The Working Set Model for Program Behavior, *Comm. ACM*, Vol. 11, No. 5, pp. 323-333 (1968).
4) Denning, P. J.: Working Sets Past and Present, *IEEE Trans. Software Engineering*, Vol. SE-6, No. 1, pp. 64-84 (1980).
5) Denning, P. J., Kahn, K. C., Leroudier, J., Potier, D. and Suri, R.: Optimal Multiprogramming, *Acta Informatica*, Vol. 7, No. 2, pp. 197-216 (1976).
6) Graham, G. S.: A Study of Program and Memory Policy Behavior, Ph. D. Thesis, Department of Computer Sciences, Purdue University (1976).
7) 近藤正人，山口純一，近藤留美子：プリフェッチを主体とするページングアルゴリズム（PDP法）の作成と評価，情報処理学会論文誌，Vol. 23, No. 3, pp. 235-242 (1982).
8) Madison, A. W. and Batson, A. P.: Characteristics of Program Localities, *Comm. ACM*, Vol. 17, No. 11, pp. 614-620 (1974).
9) 益田隆司，フィン・トン・ハン：原始プログラムの構造を利用した局所参照モデルの実現と評価，情報処理学会論文誌，Vol. 24, No. 3, pp. 318-325 (1983).
10) 益田隆司，亀田壽夫：オペレーティングシステムの性能解析，情報処理叢書9，情報処理学会，東京 (1982).
11) Smith, A. J.: A Modified Working Set Paging Algorithm, *IEEE Trans. Comput.*, Vol. C-25, No. 9, pp. 907-914 (1976).
12) Wirth, N., Cichelli, R. J., Thompson, M. Q. and McGrath, J. P.: XREF—A Cross Reference Program, *PASCAL News*, No. 17, pp. 41-46 (1980).

（昭和59年4月19日受付）
（昭和59年5月15日採録）