フェイズの移動特性に適応する記憶管理方式の提案†

益田隆司†† 倉田克彦†††

ワーキング・セット法は，プログラムが実行中に局所参照しているページ集合を，そのプログラムが過去一定時間内に参照したページ集合であるワーキング・セットによって推定し，それを主記憶上に保持してプログラムを実行する方式であり，現実のシステムでも広く利用されているすぐれた仮想記憶の管理方式である．しかしながら，その一つの問題点は，局所参照しているページ集合が変化したときに，一時的に不要なページ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大きいことである．本論文では，原始プログラム内のループ構造を利用して，この問題を改善する方法を提案した．提案した方法の根拠は，プログラム実行時の局所参照性，あるいは，局所参照するページ集合が変化するという特性は，原始プログラム内のループ構造に原因があるということである．局所参照の原因となっているループについては，実行時にループの入り口で，そのループ内で参照されないと考えられるページを，まとめてページ・アウトする．提案した方式の具体的な実現方法を述べ，その特性をシミュレーションによって評価した結果をワーキング・セット法の結果と比較して述べる．

1. まえがき

プログラムには，局所参照 (locality of references) の性質が存在することが知られている[3]．これは，プログラムがその実行時に，プログラム全体を時間的に一様に参照するのではなく，ある時間間隔ではプログラム全体の一部分のみを参照し，参照する部分が時間とともに移動するという性質である．そして，プログラムの動作が一つの局所参照しているページ集合内に留まっている間隔は，そのなかで参照されるページ集合とあわせて，フェイズ (phase) と呼ばれている．

現在，仮想記憶の制御方式として最も広く利用されているワーキング・セット (working set) 法[2),4)] は，プログラムが各時点で局所参照しているページ集合を，ワーキング・セットによって推定し，その推定量を主記憶にロードしてプログラムを実行する記憶管理方式である．時刻 t で推定量として利用されるワーキング・セット $W(t, T)$ は，時刻 t からそのプログラムが実行された時間 (仮想時間; virtual time) だけを累積して，ウィンド・サイズ T 時間だけ過去をふりかえり，その間に参照されたページ集合として定義される．

ワーキング・セット法は，概念的にはすぐれた記憶管理方式であるが，その一つの問題は，フェイズの移動時に，新しいフェイズにはいってウィンド・サイズの時間が経過するまでは，それまでに実行されていたフェイズで使用され，新しいフェイズでは使用されないページが数多くワーキング・セット内に含まれてしまうような現象が発生することである．とくに，隣り合ったフェイズ内で使用されるページの種類が大幅に異なるような場合には，フェイズ移動時のワーキング・セットの大きさは，両方のフェイズのワーキング・セットの大きさを加えたものに近くなるので，その結果として，他のプロセスがスワップ・アウトされる可能性も発生し，システムの性能への影響も大きなものになることが予想される．

この問題を改善するための方法としてよく知られている方法は，A. J. Smith が提案した DWS (Damped Working Set) 法[10] である．DWS 法では，ページ・フォールト発生時に，アドレス参照特性からフェイズの移動に伴うページ・フォールトであるかどうかを判断し，フェイズ移動中であると判断された場合には，割当て主記憶ページ枠数を増加させないような制御を行うものである．しかしながら，フェイズ移動の検出法の精度がどの程度であるかはまったく明確ではなく，また，それを実現するために特別なハードウェアが必要であるという問題もある．

われわれは，仮想記憶の制御を行うために，これまでのように，アドレス参照列から得られる情報だけを利用するのではなく，原始プログラム内の構造から得られる情報をあわせて利用することを試みている．そして，これまでに，プログラム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原因があることを確かめた[7]．そこで，本論文では，この結果を利

† A Memory Management Strategy Which Adapts to the Phase Transition Behavior by TAKASHI MASUDA (Institute of Information Sciences and Electronics, University of Tsukuba) and KATSUHIKO KURATA (Doctoral Program in Engineering, University of Tsukuba).

†† 筑波大学電子・情報工学系
††† 筑波大学工学研究科

用して，原始プログラム内のループ構造から得られる情報を利用することにより，フェイズの移動時点を検出し，フェイズ移動時にワーキング・セット内に不要なページが含まれる現象を改善する方式を提案する．そして，その特性をシミュレーションによって確かめた結果を報告する．

近藤ら[5]は，本論文の内容と近いアプローチによって，プリフェッチを主体とした記憶管理方式を提案しているが，最も重要なプリフェッチを行う時点の決定が数多くの候補からのサンプリングで行っている点が本論文の提案と基本的に異なる点である．

2. フェイズの移動特性とワーキング・セット法

提案方式の位置付けをはっきりさせるために，本章では，1章で述べた問題の所在をもう少し明確にしておく．

あるプログラムを実行中，i 番目のフェイズから $i+1$ 番目のフェイズに移動する場合を考える．フェイズ i で使用したページ集合を A，フェイズ $i+1$ で使用するページ集合を B とすると，新しいフェイズにはいって，ウィンド・サイズ T の時間が経過するまでは，ワーキング・セットの大きさが，**図1**に示すように，$|A \cup B|$ 程度に大きくなってしまう．ここで，$|X|$ は集合 X に属する要素の数を表す．フェイズ $i+1$ にはいると，フェイズ i で使用したページのうち，$A-B$ に属するページはすでに不要であるにもかかわらず，その間主記憶内に保持されたままになる．

現実のシステムでの実現は不可能であるが，理想的な方法は，ある時刻でページの参照が終わったとき，その時刻から将来の T 時間以内にそのページが再び参照されていなければ，そのページはただちにページ・アウトしてしまう方法である．この場合の割当て主記憶ページ枠数の変化を，図1内に破線で VMIN 法[9]として示した．ワーキング・セット法も VMIN 法もページ・フォールトの発生は同時刻で生ずる．

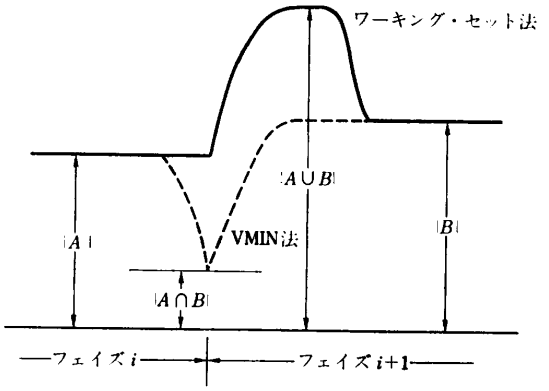

図1 フェイズ移動時のワーキング・セットの大きさ
Fig. 1 Working set size during transitions of phase.

図1に示したような，ワーキング・セット法での，フェイズ移動時のワーキング・セットの大きさの増加を軽減するための方法として提案された DWS 法は，以下のように要約することができる．

（i） ウィンド・サイズ T の間，参照されなかったページは主記憶からページ・アウトする．

（ii） ページ・フォールト発生時に，ワーキング・セット $W(t,T)$ 内の LRU ページが，現時点 t から，mult$\times T$ (mult<1) 以上過去に参照され，それ以降は参照されていない場合には，フェイズ移動の現象が発生したと考えて，新しいページを読み込む際に，その LRU ページを置換えの対象として選択する．mult$\times T$ 以内にワーキング・セット内のすべてのページが参照されていた場合には，フェイズのなかを実行中であると考えて，新たにページ枠の割当てを行い，必要なページを読み込む．

mult$=1$ の場合には，通常のワーキング・セット法である．DWS 法の問題点は，フェイズの移動の検出がどの程度正確に行うことができるかが明らかでないことである．mult の値を小さくするほど，フェイズ移動が発生したと判断される率が大きくなるが，ワーキング・セット内の近い将来参照される可能性があるページまでもページ・アウトしてしまう確率も大きくなる．mult の値をどのように設定すべきかは明らかでない．さらに，DWS 法を実現するためには，LRU スタックの管理，および，そのなかの各要素が最後に参照された時刻を管理するようなハードウェアが必要である．フェイズ移動時の性能への影響がかなり大きいと考えられるために，DWS 法は，しばしば考慮の対象とされる記憶管理の方法であるが，実システムで実現された例はまだないようである．

3. フェイズの移動に適応する記憶管理方式

フェイズ移動時に，ワーキング・セット内に不要なページが含まれる問題を，ループ構造から得られる情報によって改善することを試みる．

フェイズはループ構造によって作られるものである

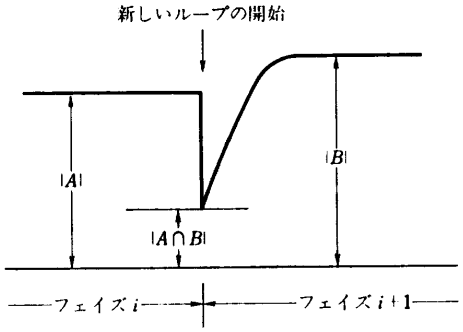

図 2 フェイズ制御方式のもとでの割当て主記憶量
Fig. 2 Allocated memory size under the phase control algorithm.

ので[7]，新しいループ構造に制御が移って，フェイズの移動が生じたと判断されたときには，そのとき主記憶内に存在しているページのうち，新しいフェイズ内で使用されないと考えられるページは，まとめてページ・アウトしてしまう．すると，フェイズ移動時の割当て主記憶量の変化は，**図 2** に示すようになるはずである．すなわち，フェイズ移動時にワーキング・セット内に不要なページが含まれる可能性は減少し，先に述べた VMIN 法に近い制御が可能になる．このような考え方に基づいた制御方式を，以下，フェイズ制御 (phase control) 方式と呼ぶことにする．

フェイズ制御を行うためには，まず第1に，新しいフェイズのなかで利用されるページ集合を予測しなければならない．あるループ構造のなかで使用されるページ集合は，ループが実行されるごとに大きくは変化しない場合が多いと考えられる．そこで，予測ページ集合としては，そのループが前回に実行されたときに使用したページ集合を利用するものとする．したがって，フェイズを構成するループが実行されるときには，そのなかでの参照ページ集合を記憶しておく必要がある．あるループがはじめて実行されるときには，使用されるページ集合がわからないので，上記の制御は行わずに，通常のワーキング・セット法による制御を行う．フェイズを構成するループ構造は静的には数が限られているので，それらのループの第1回目の実行に伴うワーキング・セット内の不要ページの性能への影響は大きくはないと判断してよい．

提案するフェイズ制御方式で，考慮しておかなければならない第2の点は，新しいフェイズの実行時間である．**図 3** に示すように，フェイズ i からフェイズ $i+1$ に移動するときに，フェイズ $i+1$ の実行時間が

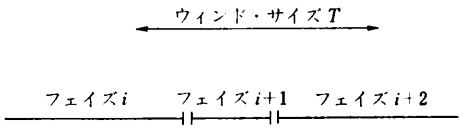

図 3 フェイズの継続時間とウィンド・サイズの関係
Fig. 3 Relationship between phase interval and window size.

短く，フェイズ $i+1$ の終了後，再び，フェイズ i の原因となったループ構造が実行されるような場合には，フェイズ $i+1$ への移動時に，そこで使用されないページ集合をページ・アウトしてしまうと，ワーキング・セット法に比較して，かえって，効率が低下してしまう可能性がある．そこで，図2に示すような制御は，フェイズの継続時間が，ワーキング・セット法の制御パラメータであるウィンド・サイズよりも長いと考えられる場合だけに限ることにする．このフェイズの継続時間の推定量としても，前回そのフェイズが実行されたときの継続時間を用いることにする．先に述べた DWS 法では，このようなフェイズの継続時間は考慮することができない．

4. 実現の方法

本章では，提案したフェイズ制御方式の具体的な実現方法を述べる．われわれは，先に，一つのフェイズの実行中に，ウィンド・サイズが小さすぎると判断されたとき，ワーキング・セット法による制御をやめて，その対応した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して利用するループ制御方式を提案した[8]．ループ制御がフェイズ内での性能改善を目的としているのに対して，フェイズ制御は，フェイズ移動時の性能改善を目的としており，両者は同時に実現することが可能である．ループ制御の場合には，ループへの入り口，ループからの出口，および，ループの繰返し終了時に記憶管理プログラムに制御を移す必要があったが，フェイズ制御を実現するためには，ループへの入り口，および，ループからの出口だけで記憶管理プログラムに制御を移せばよい．記憶管理プログラムへの制御の移動は，スーパバイザ・コール (SVC: Supervisor Call) 命令を利用する．

これらの制御を実現するためには，コンパイラは，原始プログラム内のループ構造に対応した目的プログラムのなかの必要な場所に，SVC 命令をループ識別番号とともに埋め込む．フェイズ制御のためには，ループの入り口，および，出口に SVC 命令が埋め込

まれればよい.

フェイズ制御を実現するために必要なシステムの管理情報としては，以下のようなものがあればよい.

(i) loop_table [ ]:

それまでに実行されたループに関する情報を格納しておくための表で，次の要素からなっている.

loop#: ループ識別番号

start_time: ループの開始時刻

referenced_page [0..MAXPAGE#]: ループ内で参照されたページ集合

(ii) active_loop_stack [ ]:

現在制御がそのなかに存在しているループの識別番号等を格納しておくための表. ループは階層的に存在している. 実行中の最も内側のループが階層のレベル l に存在している場合には，スタックの最上位には，そのレベル l に存在するループに関する情報が格納される. スタック上には，以下順に，階層のより上位に存在する $l-1$ 個のループに関する情報が格納されている. スタックの各エントリは,

loop#: ループ識別番号

loop_table_index: loop_table [ ] の対応するエントリのインデックス

recursive: 再帰的な実行か否か

からなっている.

(iii) stack_pointer:

active_loop_stack のスタック・ポインタ

これらの管理情報を利用して，ループへの入り口，および，ループからの出口での SVC 命令の処理の概略を，それぞれ，**図4**，および，**図5**に示す.

SVC の処理に若干説明を加える.

まず，ループが再帰的に実行される場合には，個々の再帰的実行の都度，それをフェイズ制御の対象とすることはせずに，そのループに最初に制御が移ったときから，再帰的実行がすべて終了して，最後にそのループから制御が離れるまでを一つのフェイズと考えるような処理をする.

階層構造をなしているフェイズの構造において，各フェイズ内での参照ページ集合は次のようにして求めることができる. 階層のレベル l に存在するフェイズに対応したループでの処理を考える.

ループへの入り口での処理:

(1) $l \neq 1$ (最上位のレベルでない) ならば，各ページの参照ビット R を調べ，それまでに参照されたページを求め，再帰的実行のループを除いた階層の1段上に存在するループの referenced_page [ ] に加える. $l=1$ ならば何もしない.

(2) R ビットをリセットする.

```
loop_table [ ] から loop#=j のエントリを捜す
if 該当するエントリがない then {ループ j の最初の実行}
    loop_table [ ] にループ j のためのエントリを設ける
{loop_table [ ] におけるループ j のエントリのインデックスを i とする}
increment stack_pointer
if ループ j の最初の実行 then
    push_to_active_loop_stack (j, i, false)
    record_in_loop_table (j, 現在の仮想時刻, 空集合)
    active_loop_stack [ ] から，再帰的な実行でなく1レベル上のループ k を捜す
    if 該当するループ k が存在する then
        ループ k の referenced_page [ ] に R bit [ ] を加える
    reset R bit [ ]
else {ループ j の2回目以降の実行}
    active_loop_stack [ ] 中に，すでに loop#=j のエントリがあるかどうかにより，今回の実行が再帰的な実行か否かを調べる
    if 再帰的な実行 then push_to_active_loop_stack (j, i, true)
    else
        {フェイズ制御を行う}
        push_to_active_loop_stack (j, i, false)
        record_in_loop_table (j, 現在の仮想時刻, 空集合)
        active_loop_stack [ ] から，再帰的な実行でなく1レベル上のループ k を捜す
        if 該当するループ k が存在する then
            ループ k の referenced_page [ ] に R bit [ ] を加える
        reset R bit [ ]
```

図 4 ループ入り口での SVC 処理
Fig. 4 SVC at the entrance of a loop.

```
{loop_table [ ] におけるループ j のエントリのインデックスを i とする}
if active_loop_stack [stack_pointer] の recursive=false then
    ループ j の referenced_page [ ] に R bit [ ] を記録する
    active_loop_stack [ ] から，再帰的な実行でなく1レベル上のループ k を捜す
    if 該当するループ k が存在する then
        ループ k の referenced_page [ ] にループ j の referenced_page [ ] を加える
    ループ j の実行時間を計算する
        {現在の仮想時刻−ループ j の start_time}
    if ループ j の実行時間<ウィンド・サイズ then
        ループ j に対応する SVC 命令を NOP 命令に変える
        ループ j の loop_table [ ] エントリ (loop_table [i]) を削除する
decrement stack_pointer
```

図 5 ループ出口での SVC 処理
Fig. 5 SVC at the exit of a loop.

ループからの出口での処理

(1) Rビットから，それまでに参照されたページを求め，そのレベルlのループの referenced_page [ ] に加える．

(2) $l \neq 1$ ならば，そのレベルlのループの referenced_page [ ] を，再帰的実行のループを除いた階層の1段上に存在するループの referenced_page [ ] に加える．$l=1$ ならば何もしない．

ループへの入り口，ループからの出口でのこのような処理によって，各ループの終了時に，対応した referenced_page [ ] には，そのループ内で参照したページ集合を求めることができる．

前章で述べたように，ウィンド・サイズよりも継続時間が短いループは，フェイズ制御の対象とはしないので，あるループを実行したとき，その継続時間がウィンド・サイズよりも短かった場合には，ループ入り口，出口での SVC 命令を NOP (no operation) 命令に変えてしまう処理をする．このような処理によって，大半のループは，最初の実行でフェイズ制御の対象からはずされ，フェイズ制御に伴うオーバヘッドは大幅に軽減されることになる．

5. 実験結果

本章では，提案したフェイズ制御方式の特性をシミュレーションによって評価した結果を述べる．フェイズ制御が動作するのは，ウィンド・サイズ T より長い継続時間を有するループが繰り返し実行される場合である．現実のシステムで利用されているウィンド・サイズの値は，数 100 ミリ秒程度であることが多いが，命令トレース・データに基づいたシミュレーションでこのような長いループを繰り返し実行するようなプログラムを解析対象とすることは，そのデータ量からも実質的にむずかしい．本章で述べるシミュレーションでは，ウィンド・サイズをずっと小さな値に設定し，継続時間の短いループもフェイズ制御の対象とすることによって，より少ないデータ量でその特性を把握する．

PASCAL 言語で書かれたプログラムを解析の対象とすることにし，そのために，PASCAL コンパイラに修正を加えて，ループ構造の目的プログラムには，その入り口と出口で SVC 命令を発行するようにした．3種類のプログラム[7]を解析の対象とした．

まず，はじめに，フェイズ制御方式の全体的な動作特性について述べる．**図6**は，サンプル・プログラムの一つである BASIC について，その実行に伴う割当て主記憶量の変化の様子を示したものである．横軸が実行命令数，たて軸が割当てページ枠数を表している．実線がフェイズ制御方式を採用した場合，点線が通常のワーキング・セット法による制御を行った場合の結果である．ウィンド・サイズ T は 1,000 (命令実行時間) とした．

BASIC の場合，トレースされたデータ量は 318,885 命令であるが，仮想時刻 160,000 をこえたところで，ページ参照の特性が大きく変化している[7]．それ以降では，ウィンド・サイズが 1,000 のとき，フェイズ制御の対象となるループは存在せず，ワーキング・セット法の場合とまったく同じ結果を示したので，図6では，その部分は省略した．

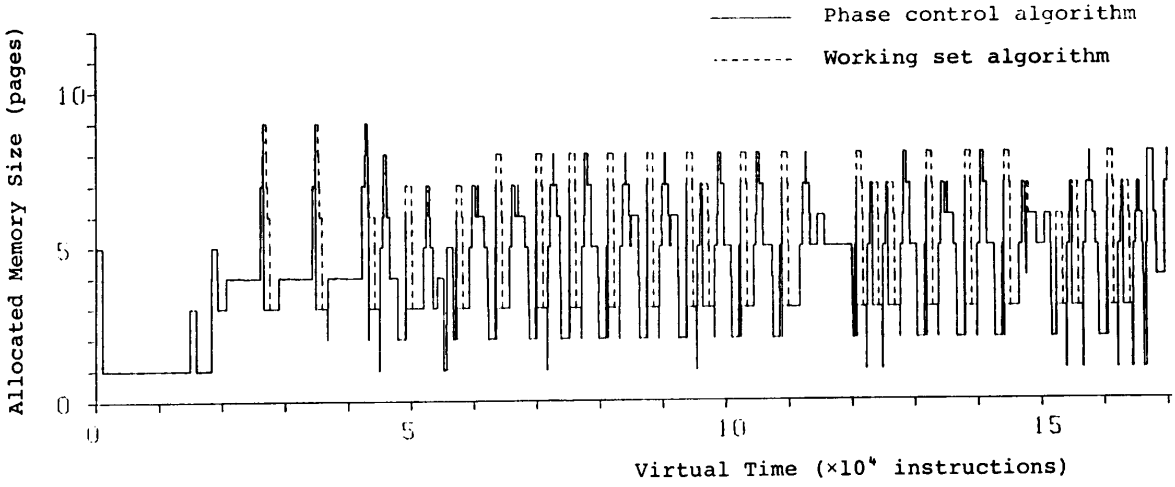

図 6 割当て主記憶量の時間変化
Fig. 6 Allocated memory size during the execution of a sample program.

図6をみると，フェイズ制御方式を採用した場合には，ワーキング・セット法に比較すると，多くのところで，はやめにページ・アウトが行われており，期待した制御がなされている．たとえば，仮想時刻60,000から150,000のあいだでは，継続時間がウィンド・サイズよりも長い，同一のループが繰り返し実行されており，フェイズ制御の対象となっていることがわかる．いずれの場合にも，このループ内での使用ページは3ページであるが，ループにはいるときに，ワーキング・セットは7ページからなっている．ワーキング・セット法では，ループに制御が移ってから，ほぼ，$T=1,000$ の時間が経過するあいだ，ループ内で新たに参照した1ページを含めて，8ページがワーキング・セットとして主記憶内に保持される．これに対して，フェイズ制御方式を用いた場合には，ループにはいると同時に，7ページのうち5ページが，ループ内では使用されないであろうという予測のもとに，ページ・アウトされてしまっていることがわかる．

この例で，DWS 法を用いた場合を考えてみる．ループにはいってすぐに，1回だけページ・フォールトが生じて，DWS 法が動作し，ワーキング・セット内に置換えの対象となるページが存在するかどうかが調べられる．その時点で，ワーキング・セット内に存在する7ページは，いずれもその直前に参照されたページばかりであるので，置換え対象ページは存在せず，新たな主記憶ページ枠の割当てがなされる．その後，そのループ内では，ページ・フォールトは生じないので，DWS 法が動作することはなく，結果として，ワーキング・セット法と同様の制御しかなされないことになる．

他のサンプル・プログラムで，図6と同様の結果を求めてみると，部分的にではあるが，図2のような，フェイズ制御の特性をよりはっきり示しているパターンも存在していることが確かめられた．

フェイズ制御方式の特性をもう少し定量的に把握してみる．**図7**は，フェイズ制御方式を採用した場合のページ・フォールト率を，ワーキング・セット法のページ・フォールト率で基準化して求めたものである．フェイズ制御方式では，ページ・アウトされたページがウィンド・サイズ内で再参照され，ページ・フォールト率がワーキング・セット法の場合に比較して，より大きくなる可能性があるが，図7をみると，BASIC，XREF の場合には，フェイズ制御を行ってもページ・フォールト率はまったく増加していない．これは，繰り返し実行され，フェイズ制御の対象となるループ内で参照されるページ集合が毎回同じであるためである．これに対して，PASCAL では，フェイズ制御の対象となっているループで，実行のたびに，参照するページの集合が若干異なるループが存在するために，ごくわずかであるが，フェイズ制御を行った場合のページ・フォールト率が，ワーキング・セット

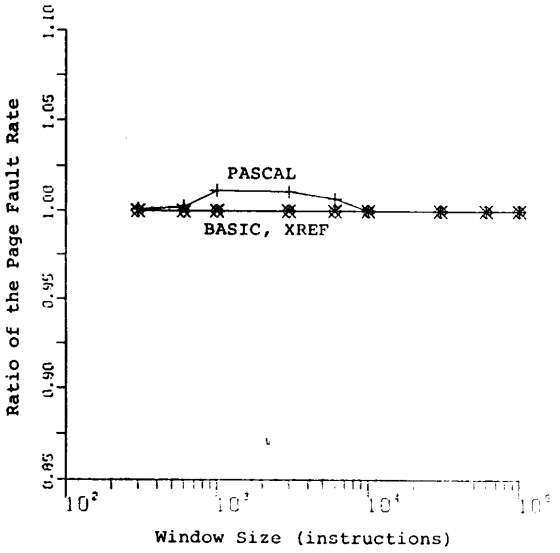

図 7 フェイズ御制方式とワーキング・セット方式のページ・フォールト率の比
Fig. 7 Ratio of the page fault rate of the phase control algorithm to that of the working set algorithm.

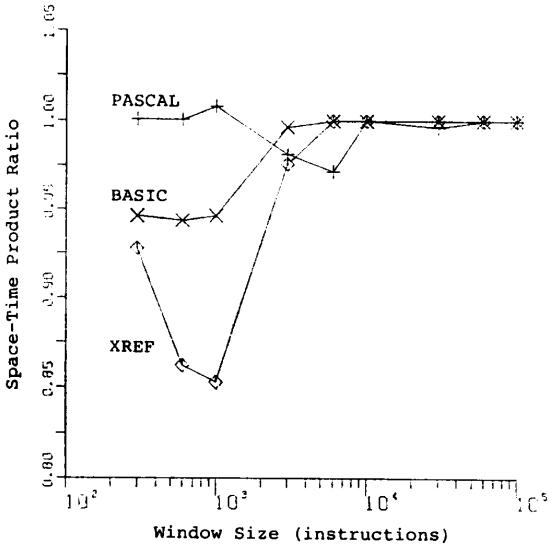

図 8 フェイズ制御方式とワーキング・セット方式のSTP の比
Fig. 8 STP ratio of the phase control algorithm to that of the working set algorithm.

法の場合よりも増加していることがある.

図6と図7を組み合わせた総合的な評価尺度として, STP (space time product; 割当て主記憶量の時間積分) を用いて, フェイズ制御の効果を求めてみる. **図8**は, 三つのサンプル・プログラムについて, フェイズ制御を行った場合の STP をワーキング・セット法の STP で基準化したものである. 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としては, 50,000 (命令実行時間) を用いた場合の結果である. STP の減少率は, PASCAL の場合には小さいが, BASIC, XREF では, フェイズ制御が頻度多く動作しているようなウィンド・サイズの範囲では, 5〜15 パーセント程度とかなりの効果を示している.

6. むすび

ワーキング・セット法の問題点として指摘されている, フェイズ移動時にワーキング・セット内に不要なページが数多く含まれる現象を改善する方法を提案した. 提案したフェイズ制御方式は, プログラム実行時の局所参照の性質が, プログラムのループ構造に起因するという点に注目して, フェイズ発生の原因となるループについて, そのループの実行が開始される直前に, そのループ内で参照される可能性がないと判断された主記憶内のページをすべて, まとめてページ・アウトしてしまうような方式である. フェイズ制御方式の有効性を確かめるために, PASCAL コンパイラを修正して, PASCAL 言語で書かれたプログラムに対して, フェイズ制御方式のシミュレーションを実施した. ワーキング・セット法についてもシミュレーションを行い, 両者を比較検討した結果, 提案したフェイズ制御方式は, 予想したような制御の効果を示していることが確かめられた.

今後の課題としては, ループ内で参照するページ集合の推定量として用いた, 前回そのループが実行されたときに参照したページ集合がどの程度よい推定量であるかを, 現実に近い環境で多くのプログラムに対して調査する必要がある. さらに, システム全体としてみたときに, フェイズ移動時の性能に及ぼす影響の程度に関しての実測データはこれまでほとんど報告されておらず, いくつかのプログラムに対しての結果が存在するだけである. フェイズ制御の効果を定量的に確かめるためには, この点に関する検討も必要である.

これらいずれを検討するためにも, 最もよい方法は, 多数のプログラムに対しての実測を行うことであり, フェイズ制御方式の実システムへの実現が今後の課題である.

参考文献

1) Batson, A. P., Blatt, D. W. E. and Kearns, J. P.: Structure within Locality Intervals, Proc. Symposium on Modeling and Performance Evaluation of Computer Systems, Beilner, H. and Gelenbe, E. (eds.), pp. 221-232, North-Holland, Amsterdam (1977).
2) Denning, P. J.: The Working Set Model for Program Behavior, *Comm. ACM*, Vol. 11, No. 5, pp. 323-333 (1968).
3) Denning, P. J.: On Modeling Program Behavior, Proc. 1972 SJCC, pp. 937-944 (1972).
4) Denning, P. J.: Working Sets Past and Present, *IEEE Trans. Softw. Eng.*, Vol. SE-6, No. 1, pp. 64-84 (1980).
5) 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).
6) Madison, A. W. and Batson, A. P.: Characteristics of Program Localities, *Comm. ACM*, Vol. 17, No. 11, pp. 614-620 (1974).
7) 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).
8) 益田隆司: 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, 情報処理学会論文誌, Vol. 25, No. 6, pp. 990-997 (1984).
9) Prieve, B. G. and Fabry, R. S.: VMIN—An Optimal Variable Space Replacement Algorithm, *Comm. ACM*, Vol. 19, No. 5. pp. 295-297 (1976).
10) Smith, A. J.: A Modified Working Set Paging Algorithm, *IEEE Trans. Comput.*, Vol. C-25, No. 9, pp. 907-914 (1976).

(昭和59年7月6日受付)
(昭和59年11月15日採録)