Vol.2017-ARC-225 No.2
Vol.2017-SLDM-179 No.2
Vol.2017-EMB-44 No.2
2017/3/9

ストライドアクセスの階層構造に着目したフェーズ検出

渋江 陽人[†1,a)]　野村 隼人[†1]　入江 英嗣[†1]　坂井 修一[†1]

概要：
プログラム実行は異なる傾向を持つ複数の「フェーズ」に分割できることが知られている．フェーズ検出はシミュレーションポイントの検出からマイクロアーキテクチャの適応制御まで，静的・動的双方の面でプロセッサの効率化に有用である．しかしながら，キャッシュの解析や適応制御においては，他の解析において広く用いられてきたフェーズ検出手法は精度が不十分であり，活用が進んでこなかった．本論文ではメモリの大域的なアクセスパターンを基にフェーズ検出を行う手法を提案し，フェーズ検出をキャッシュ最適化用途に利用可能なことを明らかにする．提案手法および従来手法についてキャッシュヒット率が変化する箇所の検出精度を比較したところ，スレットスコアにおいて従来手法よりも平均 0.20，最大 0.87 高いスコアが得られ，提案手法の有効性が示された．

Shibue Akito[†1,a)]　Nomura Hayato[†1]　Irie Hidetsugu[†1]　Sakai Shuichi[†1]

1. はじめに

プログラムを実行した際のプロセッサの動作は一般に多様であるが，1 つのプログラムについて処理段階ごとに見たとき，それぞれの段階では一定の傾向を示すことが知られている．この一定の傾向を示す期間のことをフェーズ [1]，傾向の変化を検出することをフェーズ検出 [2] と呼ぶ．

フェーズ検出はプロセッサの動的な最適化等を目的として行われる．プロセッサのどの動作に注目して検出したかによって，異なるフェーズが定義される．そのため，目的に応じた検出手法を用いる必要がある．

既存のフェーズ検出として，プログラムカウンタ（PC）に注目したものである，BBV[3] が広く用いられている [4], [5]．全てのプログラムカウンタに注目すると情報量が多いため，PC の代わりに基本ブロックを用いて検出している．

一方，メモリアクセスがプロセッサの性能のボトルネックとなっていることが知られている．そこで，メモリアクセスの最適化に用いることができるフェーズが有用である．

そこで，ストライドに着目してフェーズを検出する，新たな手法を提案する．アクセス同士のアドレスの差（ストライド）に注目し，一定のストライドでアクセスが続いているものを検出する．この一連のアクセスをストライドアクセスと定義する．さらにストライドアクセスの先頭同士のストライドを高次ストライドと定義し，これを用いてフェーズを検出する．

ベンチマークを実行した際のアクセスログからフェーズ検出を行い，キャッシュヒット率が大きく変化していた箇所を検出できるか実験を行った．BBV を用いた場合の結果とスレットスコアを比較して，提案手法では 1 を最大としたスコアにおいて平均 0.20，最大 0.87 高い結果が得られ，BBV よりもキャッシュヒット率変化を正確に検出できることが示された．

2. 関連研究

2.1 BBV

フェーズとは，プログラム実行に関する注目している測定値の変動が小さい連続した実行期間のことである [6]．

プログラムカウンタに着目してフェーズ検出を行う手法として，BBV を用いた手法 [6] がある．この手法のアルゴリズムを述べる．

一定期間ごとに，どの PC が何度実行されたかを記録する．記録する期間が終了したとき，直前の記録と比較する．プログラムカウンタの実行回数が大きく変わっていた場合，フェーズが切り替わったと検出する．しかし PC ごとに調べると情報量が大きく，記録コストがかかってしまう．そこで，分岐や合流を含まない連続した命令をまとめた，基本ブロック（Basic Block）を用いる．基本ブロック

[†1] 現在，東京大学
Presently with The University of Tokyo
[a)] shibue@mtl.t.u-tokyo.ac.jp

Vol.2017-ARC-225 No.2
Vol.2017-SLDM-179 No.2
Vol.2017-EMB-44 No.2
2017/3/9

の実行回数を記録したベクトルと，同様に記録したベクトルのマンハッタン距離を比較し，一定以上の距離があればフェーズが異なると見做す．ただし，基本ブロックの長さは一定ではないため，比較を公平にするために，カウンタの値に基本ブロックの長さを乗算した値を用いる．

ここで，記録を行う期間の長さをインターバルと呼ぶ．インターバルの長さを調節することで，短期的なフェーズから長期的なフェーズまで，目的に応じて検出することができる．

しかしこの手法は PC のみに着目しており，メモリアクセスに関して最適なフェーズを検出するものではない．

2.2 メモリアクセスの局所性によるフェーズ

メモリアクセスに注目した手法に，データの再参照間隔を用いたもの [7] がある．この手法では同じアドレスへの参照間隔に注目している．

全てのアクセスのうちから特徴的なアクセスを抽出し，その参照間隔を記録する．記録した参照間隔に対してウェーブレットフィルタをかけることで参照間隔の変化を調べ，参照間隔が大きく変化した箇所でフェーズを検出する．

しかしこの手法は処理に時間のかかるウェーブレットフィルタを利用しており，動的なフェーズ検出に用いることができない．

3. 提案手法

3.1 提案

メモリアクセスがプロセッサの処理速度のボトルネックとなっていることから，本研究ではその改善に向けた，キャッシュヒット率が変化する箇所のフェーズ検出を目的とする．広く用いられている先行研究の BBV による検出は PC のみに着目しており，メモリアクセスを扱う上で不適切な場合がある．

キャッシュにプリフェッチをかけた場合，配列に対するシーケンシャルアクセスではプリフェッチによって高いキャッシュヒット率が得られる．しかしシーケンシャルアクセスが終了するごとにプリフェッチが効果を失い，キャッシュヒット率が低下する．そのためシーケンシャルアクセスの終了は検出されるべきであるが，BBV は多重ループやループ内の分岐において不適切な検出をすること，適切な検出ができないことがある．図 1 に，BBV による手法でフェーズを誤検出する場合の具体例を示す．ループ中でデータの読み出しを行い，そのデータによって分岐するコードを考える．このとき，読み出したデータによって，分岐のどちらの基本ブロックが実行されるかが変化する．配列 a に格納されていた値によっては，ひとつのループ処理を繰り返しているにも関わらず，分岐の結果が偏ってフェーズの切り替わりが検出されてしまうことがある．図 2 に，BBV による手法でフェーズの切り替わりを検出

```
1:  for(int i = 0; i < a.size; i++){
2:      int t = a[i];
3:      if(t < 0)
4:          do_something0;
5:      else
6:          do_something1;
7:  }
```

図 1 BBV がフェーズを誤検出する例

```
1:  for(int i = 0; i < a.size; i++){
2:      for(int j = 0; j < b.size; j++){
3:          a[i] += b[j];
4:      }
5:  }
```

図 2 BBV がフェーズを検出できない例

できない場合の具体例を示す．多重ループを持ち，内側のループ処理で 2 つの配列に対してシーケンシャルアクセスが行われるコードを考える．このとき，内側のループが終了しても，外側のループが続く場合には，図 2 の 1 行目の基本ブロックが 1 回カウントされるのみで，BBV のカウンタに大きな影響を与えない．その結果として，内側のループの終了が検知できず，検出すべきフェーズの切り替わりが検出できない．

そこで，提案手法ではシーケンシャルアクセスの頂点に注目する．シーケンシャルアクセスの頂点同士のストライドを比較することによって大域的な傾向を捉え，フェーズを検出する．特に，シーケンシャルアクセスの頂点に注目することでシーケンシャルアクセスの始まりを捉え，BBV で誤った検出をする場合にも対処する．

本研究では，ストライドに着目した新たなフェーズ検出手法を提案する．特に，シーケンシャルなアクセスが階層的な構造を持つことに注目し，3.2 で定義する高次ストライドアクセスを用いる．

高次ストライドアクセスの検出では以下を行う．

(1) 1 次ストライドアクセスの検出
(2) 1 次ストライドアクセスの先頭による 2 次
(3)

フェーズ検出では以下を行う．

(1) ログの時間分割
(2) 分割した各区間における 2 次ストライドアクセスのカ

Vol.2017-ARC-225 No.2
Vol.2017-SLDM-179 No.2
Vol.2017-EMB-44 No.2
2017/3/9

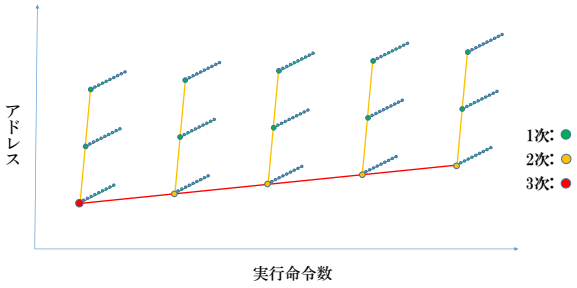

図 3　高次ストライドアクセスの検出

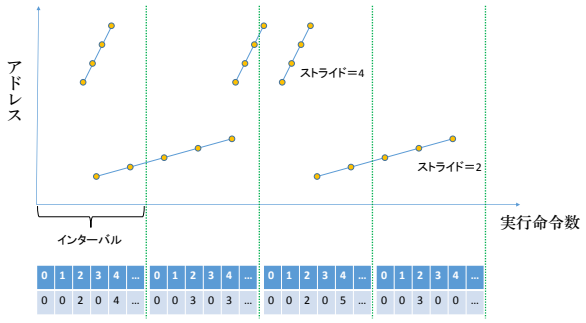

図 4　ストライドごとのアクセス回数カウンタ

ウント

(3) カウンタの比較

本章では，複数の低次のストライドの先頭のみを記録することで高次のストライドアクセスを提案手法ではアクセスのアドレス間隔とシーケンシャルアクセスに提案手法では，ベンチマークプログラムを実行した際のメモリアクセスログを元にしてフェーズ検出を行う．

(1) 全てのアクセスから 3.2 節で述べる高次ストライドアクセスを検出する．

(2) 次に，ログを一定時間ごとに分割し，それぞれの区間に高次ストライドアクセスがいくつあったかをカウンタに記録する．このカウンタを隣り合った区間同士で比較し，一定以上の差が見られたときにフェーズの切り替わりを検出する．

3.2　高次ストライドアクセスの検出

3.2.1　ストライドアクセスと階層的構造

複数のメモリアクセスがあったとき，アクセスしたアドレスの差をストライドと呼ぶ．ストライドが一定であるシーケンシャルアクセスが存在するとき，このアクセス群を 1 次ストライドアクセスと定義する．

図 3 のように複数の 1 次ストライドアクセスが存在するとき，ストライドアクセスの始点のみに注目すると，別のストライドアクセスが見られる場合がある．このとき，1 次ストライドアクセスの頂点のみからなる別のストライドアクセスを 2 次ストライドアクセスと定義する．2 次ストライドアクセスの頂点に注目することで 3 次ストライドアクセスを定めることができ，以下同様に n 次ストライドアクセスを定義する．

3.3　高次ストライドを用いたフェーズ検出

3.3.1　ログの時間分割とカウンタ

ベンチマークの実行ログに対し，n 次ストライドアクセスを検出する．次に，ログを一定の時間間隔で分割し，分割した各区間中の特徴を抽出する．この各区間の長さをインターバルと呼ぶ．本研究では時間間隔の基準として，そのアクセスが行われた時点までに実行された命令数を採用した．

図 4 のように，分割した区間のそれぞれについてアクセス回数カウンタを用意する．アクセス回数カウンタは，多くのカウンタの集まりである．このカウンタを用いて，ストライドごとにアクセス回数をカウントする．具体的には，区間内で行われた各メモリアクセスについて，それが n 次ストライドアクセスの一部だった場合，その n 次ストライドに対応するカウンタの値をインクリメントする．このようにして，インターバルごとにアクセス回数のベクトルを得る．

3.3.2　ベクトル間のマンハッタン距離によるフェーズ検出

3.3.1 項の手法によって得たベクトルを用いてフェーズ検出を行う．

実行命令数が少ない方から順に各区間のベクトルを調べる．マンハッタン距離を用いて，注目している区間のベクトルと直前の区間のベクトルを比較する．マンハッタン距離が閾値より大きい場合，区間の特徴が大きく変化したとしてフェーズが切り替わったと見做す．

フェーズ検出を行う閾値について，アクセスの回数は各インターバルで一定ではなく，全ての期間について一定の閾値を設けることは必ずしも適切ではない．また，ベンチマーク同士を比較したときにも大きくアクセス回数が異なるため，全てのベンチマークにおいて最適なパラメータ設定は困難である．

そこで，比較している 2 つのインターバル中にいくつのアクセスが見られたかによって，動的に閾値を変化させる．具体的には，比較している 2 つのカウンタについて，その総和の一定割合を閾値とする．これによってアクセス回数のばらつきによる検出への影響を抑えることができる．

可変なパラメータとして，インターバルの長さ，カウンタの大きさ，フェーズ検出の閾値がある．インターバルの長さによってどの程度の長さのフェーズを捉えられるかが大きく異なる．本研究ではシミュレータおよびログを用いた解析のため理想的な状況を想定し，十分多くのエントリと十分大きな最大値を持つとする．

Vol.2017-ARC-225 No.2
Vol.2017-SLDM-179 No.2
Vol.2017-EMB-44 No.2
2017/3/9

3.4 アウト・オブ・オーダ・スーパースカラにおける問題

メモリアクセスをアウト・オブ・オーダに実行するプロセッサでは，アクセスの順番が入れ替わることがある．一方で，ストライドアクセスは一定のストライドで順番にアクセスがあることを前提としているため，このような場合にストライドアクセスの検出，およびフェーズの検出ができなくなってしまう．そのため，次のような工夫を行った．

ストライドアクセスの一部に遅延が生じると，その後のアクセスが先に起こるためにその箇所で一連のストリームが切れてしまう．そこで過去に検出したストライドアクセスを調べ，ストライドアクセスの終点からストライド 1 つ分だけ先のアドレスから始まるものがあれば，ストライドが等しい場合に限り連結することとした．ストライドアクセスが検出されないアクセスに対しては，本来はストライドアクセスの先頭だったものが遅延されて実行されたものである可能性があるため，過去に検出したストライドアクセスの先頭と連結してストライドに矛盾がない場合にのみ連結することとした．

また，スーパースカラプロセッサやキャッシュミスを並列処理できるプロセッサなど，複数のアクセスを同時に処理できる場合には，実行命令数が増えないまま，複数のアクセスが記録されることがある．この場合，ストライドアクセスが検出されるような順序でアクセスが起こったと仮定して解析を行った．特に，スタック領域へのアクセスを除く多くの場合で小さいアドレスから大きいアドレスに向かってストライドアクセスが生じる様子が見られたため，簡単のために実行命令数が等しいメモリアクセス同士についてはアドレスの小さい順に発生したと仮定した．

アウト・オブ・オーダ実行されたアクセスに対して工夫を適用した場合（以下，実行終了時と表記）と，イン・オーダに並べ替え，工夫を適用しない場合（以下，リタイア時と表記）の両方について，提案手法による解析，および従来手法との比較を行った．実行終了時を用いた場合の結果が有用であれば，イン・オーダであることが保証されるリオーダバッファに限らず，キャッシュなどのアウト・オブ・オーダの影響を受けるものに対しても提案手法の機構を実装可能であることが示される．

4. 評価

4.1 使用ベンチマークと使用ツール

SPEC CPU 2006[8] の全ベンチマークを対象に提案手法を用いた解析を行った．シミュレータ「鬼斬弐」[9] を用い，10G 命令をスキップし，その後の 1G 命令を実行してメモリアクセスのログを取得した．データへのアクセスは全て捕捉し，プリフェッチは行わなかった場合とストリームプリフェッチを行った場合の 2 種類を用いた．

ストリームプリフェッチを行った場合について，L2 キャッシュおよび L3 キャッシュのそれぞれに同じ構成のストリームプリフェッチャを適用した．プリフェッチャの構成は [10] による．プリフェッチャのパラメータは，Degree を 16，Distance を 16，追跡するストリームの最大数を 16 とした．

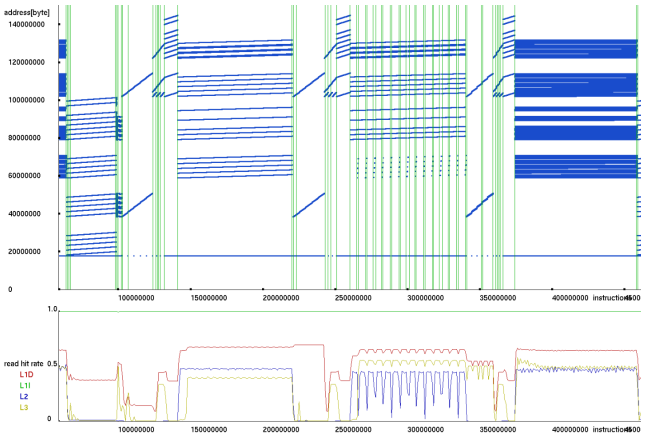

図 5 437.leslie3d における提案手法によるフェーズ検出結果

また，キャッシュの容量はそれぞれ，L1D キャッシュは 32KB，L1I キャッシュは 32KB，L2 キャッシュは 256KB，L3 キャッシュは 4MB とし，置換アルゴリズムはいずれも LRU を採用した．

4.2 高次ストライドアクセスの検出

各アクセスは，アクセス命令が実行終了した時点，もしくはリタイア時に行われたものとした．あるアクセスに対して，直後 32 アクセスに対してストライドを調べ，さらにストライドを調べたアクセスの直後 128 個のアクセスを見て，それがストライドアクセスと見做せるかどうか調べた．ストライドアクセスは，無関係なアクセスが 128 個連続したときに終了したと見做した．なお，高次ストライドアクセスの場合，これらの数字には 1 次元下のストライドアクセスの頂点のみをカウントした．

4.3 2 次ストライドアクセスを用いたフェーズ検出

4.3.1 パラメータと評価尺度

インターバルの長さは 1M 命令および 10M 命令とした．カウンタの大きさはほぼ理想的な状態を仮定した．ストライドは最大 1.0e+6，ストライドが負のときは絶対値を扱うとし，カウンタの最大値は飽和やオーバーフローしないほど十分大きいとした．

フェーズを検出している様子の例として，437.leslie3d ベンチマークについて，リタイア時，インターバル 1M，プリフェッチなし，閾値 25 %の場合の検出結果を図 5 に示す．横軸に実行命令数，縦軸にアドレスを取り，アクセスを青い点で示した．下部に，キャッシュの読み出しにおける，各インターバル中でのヒット率の推移を示している．また，縦の緑色の線は，その箇所でフェーズ検出したこと

Vol.2017-ARC-225 No.2
Vol.2017-SLDM-179 No.2
Vol.2017-EMB-44 No.2
2017/3/9

を示している．

検出したフェーズの切り替わり箇所が正しいかどうか，キャッシュのヒット率の推移と比較することで評価する．各インターバルでキャッシュの読み出しにおけるヒット率を記録し，連続した 2 つのインターバル間で，いずれかのキャッシュのヒット率が 10 %以上変化したとき，その 2 つのインターバルの境目を検出することを正しい検出と見做す．フェーズの切り替わりは稀にしか起きないため，true-negative の数が非常に大きくなる．よって，評価基準としてスレットスコアを用いた．スレットスコアは，true-positive を true-positive，false-positive，false-negative の和で除算した値である．

既存研究との比較として，BBV を用いたフェーズ検出と比較を行う．BBV は，フェーズ検出手法の比較実験 [11] によれば，BBV 同士のマンハッタン距離の取りうる最大値の 4 %，すなわちインターバルの 8 %が汎用的に高性能であるため，そのパラメータを採用する．また，インターバルおよびプリフェッチの有無は提案手法と揃えて比較する．

4.3.2 閾値を変化させた場合

インターバル 1M，インターバル 1M プリフェッチあり，インターバル 10M プリフェッチあり，インターバル 10M プリフェッチありの 4 条件についてそれぞれ命令実行終了時を扱った場合とリタイア時を扱った場合に分けた，計 8 条件について各ベンチマークで閾値を変化させ，スコアを得た．閾値を 2 区間におけるカウンタの総和の割合とした場合について，命令実行終了時の 4 条件で平均スコアが高かったのは 6 %，リタイア時の 4 条件で平均スコアが高かったのは 4 %，8 条件で平均スコアが高かった割合は 7 %に割合を設定したときであった．よって，以下の比較では 7 %に設定した際の検出結果を用いる．さらに，7 %に設定した場合と，固定値 100，固定値 500，固定値 1000，固定値 5000，固定値 10000 に設定した場合の 6 通りについて比較したところ，7 条件において 7 %に設定した場合に最も高い平均スコアが得られた．

4.3.3 アウト・オブ・オーダとイン・オーダ，既存手法の比較

実行終了時を扱った場合とリタイア時を扱った場合と BBV との 3 つについて，各ベンチマークでスコアを比較した．提案手法 2 種類の閾値は 7 %とした．BBV の閾値はインターバル長の 8 %とした．インターバル 10M，プリフェッチありとした場合の比較結果を図 6 に示す．その他の場合もほぼ同様の傾向であることが確認された．また，average は 29 ベンチマークにおける各スコアの平均値である．

実行終了時を扱った場合と BBV の比較では，提案手法が平均 0.21，最大 0.80 高いスコアを得た．リタイア時を扱った場合と BBV の比較では，提案手法が平均 0.23，最大 0.87 高いスコアを得た．実行終了時を扱った場合，リタイア時を扱った場合どちらも，インターバルおよびプリフェッチの有無に依らず 22 以上のベンチマークにおいて BBV のスコアを上回り，平均スコアでも BBV のスコアを上回った．インターバル 10M の場合において，プリフェッチありの場合，なしの場合の両方で，BBV が全く検出できていないフェーズを提案手法は検出した．

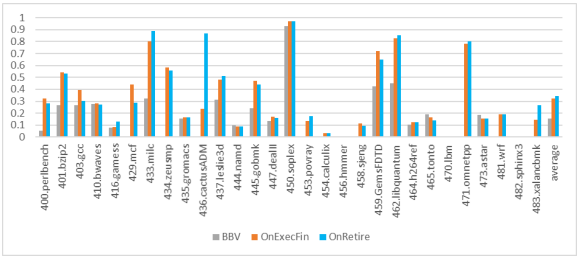

図 6　提案手法と BBV との比較
（インターバル 10M，プリフェッチあり）

閾値を変化させた場合について，動的にアクセス数の 7 %とした場合がほぼ全ての場合において固定の閾値とした場合と比較して最もスコアが高く，汎用的に有効であることが示された．

4.3.4 評価のまとめ

提案手法は 29 本中 22 本以上のベンチマークにおいて BBV よりも高いスコアを得た．また，平均スコアも BBV のものより 0.22 高く，提案手法はキャッシュのヒット率に影響を与えるようなアクセスパターンの変化を検出する上で有効であることが示された．

リタイア時を扱った場合と実行終了時を扱った場合を比較したとき，ほぼ同様の結果が得られた．従って，実行終了時を扱う上で行った工夫が有効であることが示された．

閾値の設定について，アクセス数の 7 %と設定することが，汎用的に有効であることが示された．

5. 考察

どのような状況で提案手法は特に有効に働くのか考察する．437.leslie3d ベンチマークでは，多重ループとみられる，フラクタルなストライドアクセスの構造が見られた．この場合，図 6 の通り，提案手法は BBV よりも高いスコアを得ている．従って多重ループが起こる場合には有効であると推測できる．

一方，1 つの長大なストライドアクセスのみが見られる場合，それが全て 1 次ストライドアクセスと見做されるため，2 次ストライドアクセスは検出されず提案手法ではフェーズが検出できない．長いシーケンシャルアクセスが繰り返されるベンチマークである，462.libquantum の結果に注目すると，長いストライドアクセスが繰り返されるにも関わらず，図 6 で示した結果では 0.8 を超える高スコアを得ている．

462.libquantum では，20MB もの領域で続く長いストラ

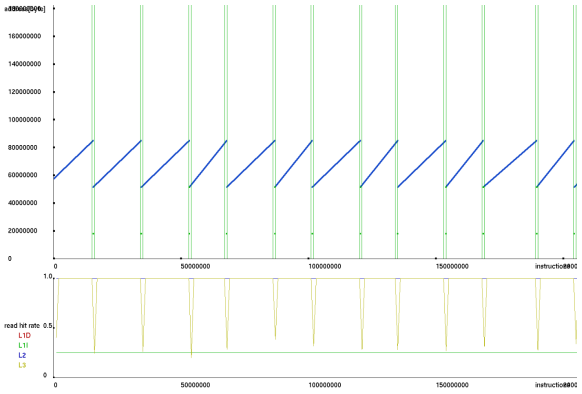

図 7　462.libquantum の様子

イドアクセスと別の小さな領域で起こる 50 個程度のアクセスとが交互に繰り返される．その様子を図 7 に示す．長いストライドアクセスは 1 次ストライドアクセスと見做されており，基本的にフェーズ検出に影響していないことが分かった．そこで，長いストライドアクセスとは別の領域で起こっているアクセス群に注目したところ，この箇所で複数の 2 次ストライドアクセスが検出されていた．

すなわち，シーケンシャルアクセスとその他の処理を交互に行う場合，その他の処理で 2 次ストライドアクセスを検出できることがある．この場合には，その他の処理がシーケンシャルアクセスの終わりと次のシーケンシャルアクセスの始まりを分ける境界となるため，正しいフェーズが検出できる．

6. まとめと今後の課題

本研究では SPEC CPU 2006 ベンチマークを実行した際のメモリアクセスに注目し，そのパターンからフェーズ検出を行った．その際，メモリアクセス間のストライド，および再帰的に定義した高次ストライドアクセスを利用した．その結果，2 次ストライドアクセスを用いて BBV よりも正確なフェーズ検出ができた．特に長いシーケンシャルアクセスのあるベンチマークではプリフェッチをかけた際のキャッシュヒット率変化と

今後の課題

本研究では理想的な状況を仮定し，ストライドを正確に捉えて解析を行った．これを元に，実際にプロセッサに実装して検出する機構を考えられれば，フェーズ検出を用いたプロセッサの性能向上も検討できる．そこで，今後の課題として以下の 2 点が挙げられる．

(1) フェーズの分類と予測

(2) 動的なフェーズ検出への利用

謝辞　本論文の研究は，一部科学研究費補助金 課題番号 25730028 による．

参考文献

[1] Batson, A. P. and Madison, A. W.: Measurements of Major Locality Phases in Symbolic Reference Strings, *Proceedings of the 1976 ACM SIGMETRICS Conference on Computer Performance Modeling Measurement and Evaluation*, SIGMETRICS '76, New York, NY, USA, ACM, pp. 75–84 (online), DOI: 10.1145/800200.806184 (1976).

[2] Hind, M., Rajan, V. and Sweeney, P. F.: Phase detection: A problem classification, Technical Report 22887, IBM Research (2003).

[3] Sherwood, T., Perelman, E. and Calder, B.: Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications, *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, PACT '01, Washington, DC, USA, IEEE Computer Society, pp. 3–14 (online), available from ⟨http://dl.acm.org/citation.cfm?id=645988.674158⟩ (2001).

[4] Nair, A. A. and John, L. K.: Simulation points for SPEC CPU 2006, *IEEE International Conference on Computer Design, 2008. ICCD 2008*, pp. 397–403 (online), DOI: 10.1109/ICCD.2008.4751891 (2008).

[5] Fang, Z., Li, J., Zhang, W., Li, Y., Chen, H. and Zang, B.: Improving Dynamic Prediction Accuracy Through Multi-level Phase Analysis, *Proceedings of the 13th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*, LCTES '12, New York, NY, USA, ACM, pp. 89–98 (online), DOI: 10.1145/2248418.2248432 (2012).

[6] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B.: Automatically Characterizing Large Scale Program Behavior, *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, New York, NY, USA, ACM, pp. 45–57 (online), DOI: 10.1145/605397.605403 (2002).

[7] Shen, X., Zhong, Y. and Ding, C.: Locality Phase Prediction, *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XI, New York, NY, USA, ACM, pp. 165–176 (online), DOI: 10.1145/1024393.1024414 (2004).

[8] Corporation, S. P. E.: SPEC CPU 2006.

[9] 塩谷亮太，五島正裕，坂井修一：プロセッサ・シミュレータ「鬼斬弐」の設計と実装，先進的計算基盤システムシンポジウム SACSIS2009， Vol. 2009, No. 4, pp. 120–121 (2009).

[10] Srinath, S., Mutlu, O., Kim, H. and Patt, Y. N.: Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers, *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, pp. 63–74 (online), DOI: 10.1109/HPCA.2007.346185 (2007).

[11] Dhodapkar, A. S. and Smith, J. E.: Comparing Program Phase Detection Techniques, *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, Washington, DC, USA, IEEE Computer Society, pp. 217– (online), available from ⟨http://dl.acm.org/citation.cfm?id=956417.956539⟩ (2003).