固定長インターバルを用いないフェーズ検出手法

赤松雄一[†] 五島正裕[†] 坂井修一[†]

プロセッサの開発にはシミュレーションによる詳細な性能測定が不可欠である．しかしシミュレーションにかかる時間は膨大であり，数週間から数ヶ月かかるものまである．そこで必要となるのがシミュレーション高速化である．シミュレーション高速化手法の１つにプログラムのフェーズ検出がある．プログラムの動的な命令列を，そのプログラムを実行するプロセッサの動作の段階に応じて分類することにより，プログラムの一部をシミュレーションするだけで全体のシミュレーション結果を推定することができる．従来の手法では固定長インターバルを用いてフェーズを検出していたが，固定長インターバルはシミュレーションの結果推定の誤差の原因となる．本稿では従来のフェーズ検出手法にはないセグメントという概念を導入することにより，より高精度なフェーズ検出手法を提案した．

A Phase Detection Technique Not Using Fixed-Length Intervals

Yuichi Akamatsu,[†] Masahiro Goshima[†] and Shuichi Sakai [†]

Modern architecture reseach relies on simulation that can evaluate a performance of a processor. However, it takes a lot of time to simulate architectures. So, a study on phase detection is very useful. Phase detection catches some simulation points, and they tells us a performance of processor. Only simulating some simulation points, we know a performance of processor. We don't simulate all the trace of a processor. So, phase detection is very useful for reducing simulation time. There are some phase detection techniques, but they use fixed length intervals. Fixed-length intervals have a bad influence on phase detection in point of making a calculation error. I introduce a new phase detection technique not using fixed length intervals in this paper. This method leads to a more accurate phase detection system than those currently in use.

1. はじめに

シミュレーションは，プロセッサの研究・開発には不可欠であるが，非常に長い時間がかかるという問題がある．実機に対する実行時間の割合を **SD** (Speed-Down) と呼ぶ．SD はエミュレータでは 100 程度，cycle-accurate なシミュレータでは 1000 以上にもなる．SD を 1000 とすると，実機で 10 分かかるプログラムのシミュレーションには 10,000 分 ≃7 日かかることになる．そのため，シミュレーションの高速化に対するニーズは大きい．

シミュレーションの高速化の方法としては，シミュレータ自体の高速化の他に，シミュレーション対象のプログラムの実行命令数の削減が考えられる．プログラムには，その繰り返し構造に起因して，そこだけを実行すれば全体の振る舞いが推定できるような部分が存在する．そのような部分を**シミュレーション・ポイント**と呼ぶ．シミュレーション・ポイント選択には，いわゆる**フェーズ検出**手法を用いることができる．

SimPoint

シミュレーション・ポイントを選択する代表的な手法として，**SimPoint**[1),2)] が挙げられる．SimPoint は，以下のようにしてフェーズ検出を行う：

(1) **インターバルへの分割**　まず，実行された PC の列を固定長の 1M〜100M 命令程度の固定長の**インターバル**に区切る．

(2) **基本ブロック・ベクトル生成**　次に，各インターバルに対して，**基本ブロック・ベクトル**を生成する．基本ブロック・ベクトルの各次元はプログラム中に存在する基本ブロックに対応し，各次元の値はそのインターバル内でその**基本ブロック** (basic block) が実行された回数を表す．したがって基本ブロック・ベクトルは，次元数が数万〜数十万以上の，多次元のスパースなベクトルになる．

(3) **クラスタリング**　最後に，得られた基本ブロック・ベクトルの集合に**クラスタリング**を施す．SimPoint は，多次元ベクトルのクラスタリング手法として代表的な **k-means 法**を用いている．同一のクラスタに分類されたインターバル

† 東京大学 大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo

がフェーズとみなされる．シミュレーション・ポイントとしては，各クラスタの代表的なインターバルを選択すればよい．

SimPoint は，1M〜100M 命令程度という長い固定長のインターバルを用いているため，その精度に関して，以下のような 2 つの問題がある：

1. インターバルより十分に長いフェーズしか検出できない．
2. 以下で述べるように，内分点の基本ブロック・ベクトルが存在するため，正しいクラスタリングが難しい．

内分点の基本ブロック・ベクトル

図 1 上に，固定長インターバルによるフェーズ検出の様子を示す．横軸は命令数で数えた時間で，縦軸は基本ブロックの ID である．同図は，2 種類のループ L_A，L_B が順に実行されている様子を表しており，それぞれがフェーズとして検出されることが期待される．しかし，インターバルは 1M〜100M 命令程度と非常に長いため，インターバル I_2 と I_4 には，L_A と L_B の両方が含まれている．I_2，I_4 の基本ブロック・ベクトルは，L_A，L_B が含まれる割合に応じて，I_1 の基本ブロック・ベクトルと I_3 の基本ブロック・ベクトルの内分点になる．

L_A と L_B が切り替わる度に，このような内分点の基本ブロック・ベクトルが現れる．その結果，**図 2** 左に示すように，基本ブロック・ベクトルは多次元空間に散在することになる．図 1 の例では，インターバル I_2 と I_4 は，含まれる L_A と L_B の割合に応じてクラスタに分類されることになる．どちらかに偏っていれば，I_1 や I_3 と同じクラスタに分類されるだろう．偏りが少なければ，I_2 と I_4 からなるクラスタが生成されるかもしれないし，I_2 のみ，I_4 のみからなるクラスタが生成されるかもしれない．このような状態では，クラスタリングが難しいというだけでなく，正解を定義することすら難しい．

提 案 手 法

そこで本稿では，固定長のインターバルではなく，基本ブロック 100 個程度を最小単位とする可変長の**セグメント**を用いてフェーズ検出を行う手法を提案する．図 1 下に，提案手法によるフェーズ検出の様子を示す．提案手法では，以下のようにフェーズ検出を行う：

(1) **セグメントへの分割**　まず，基本ブロック 100 個程度の**セグメント・ユニット**を単位として，セグメントに分割する．以下，セグメント・ユニットは，単にユニットと言う．

(2) **基本ブロック・ベクトル生成**　次に，セグメントの基本ブロック・ベクトルを生成する．基本ブロック・ベクトルは，セグメントに含まれるユニットの数によって正規化しておく．

(3) **クラスタリング**　最後に，得られた基本ブロック・ベクトルの集合に対してクラスタリングを施す．

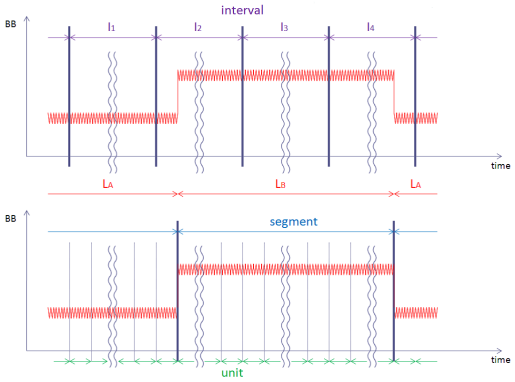

図 1　SimPoint（上）と提案手法（下）のフェーズ検出

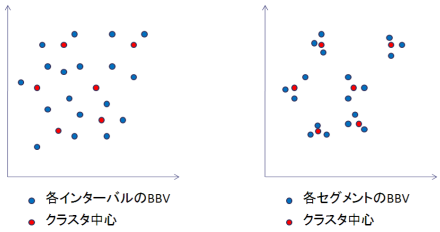

図 2　基本ブロック・ベクトルの分散

この手法では，前述した固定長インターバルを用いる手法の問題は，以下のように解決される：

1. ユニットは基本ブロック 100 個程度と小さく，その程度のフェーズを検出することができる．1M〜100M 命令程度のインターバルに比べて，基本ブロック 100 個程度のユニットの大きさは $1/1,000$〜$1/100,000$ に過ぎない．
2. ユニット自体は固定長であるので，内分点的なユニットが存在することは避けられない．しかし，基本ブロック・ベクトルは，ユニットごとにではなくセグメントごとに計算されるので，セグメントに含まれるユニットの数が十分多ければ，その影響は無視できる．

本稿は，以下のように構成されている．続く 2 章では，一般的なフェーズ検出について述べる．3 章では SimPoint についてまとめる．4 章で提案手法を詳しく説明し，5 章で評価結果を示す．

2. フェーズ検出

前述したように，シミュレーション・ポイントを適切に選択することができれば，ターゲット・プログラムのシミュレーションすべき命令の数を大幅に削減す

ることができる．シミュレーション・ポイントの選択には，フェーズ検出手法が用いられる．本章では，一般的なフェーズ検出の手法について述べる．

2.1 フェーズとシミュレーション・ポイント

プログラムの実行は，そこを実行するプロセッサの振る舞いが互いに似ている部分に分割することができる．このように，プログラム実行の一部であって，プロセッサの振る舞いが互いに似ている部分のことを**フェーズ**と呼ぶ[3),4)]．フェーズが存在するのは，プログラムに繰り返し構造があるからである．

シミュレーション・ポイントとは，プログラム実行の一部で，そこだけシミュレーションすればプログラム実行の全体に渡るプロセッサの振る舞いが推定できる部分と定義できる．したがって，シミュレーション・ポイントはフェーズの部分集合として与えられ，シミュレーション・ポイント選択の主要な部分はフェーズ検出となる．

2.2 フェーズ検出

以下では，シミュレーション・ポイント選択で用いられるフェーズ検出について概説する．

動的/静的フェーズ検出

フェーズ検出には，動的なものと静的なものがある．

動的なフェーズ検出は，プログラムを実行しながらフェーズ検出を行うもので，主に省電力化のために用いられる[5),6)]．省電力化のためには，例えば，フェーズに応じてハードウェアの構成を変更し，不必要なハードウェアを使用しないようにすることなどが考えられる．

一方，静的なフェーズ検出は，一度プログラムをシミュレーションした結果を分析しフェーズを検出するものである．本稿の主眼であるシミュレーションの高速化に用いるのは，静的なフェーズ検出である[7),8)]．

PC と基本ブロック

フェーズ検出は PC に着目して行うことが一般的である．その場合，実行された PC の時系列を**基本ブロック** (basic block) の列に変換することで扱うデータの量を大幅に削減することができる．基本ブロック 1 つあたりの平均命令数は 10 程度であるので，PC 列を基本ブロック列に変換することで，情報量を減らさずに，扱う列のデータサイズを 1/10 程度に減らすことができる．

クラスタリング

静的フェーズ検出は一般に，PC（，あるいは，基本ブロック）の列を適当に分割した系列に対して，**クラスタリング**を施すことによって行われる．

フェーズ分類の際にはデータの量が膨大となるため，計算量をできるだけ少なくする必要がある．クラスタリングの手法には，階層的な手法と非階層的な手法がある．階層的手法とは，似ているデータを階層的にまとめていきクラスタを作る手法である．非階層的手法とは，データをランダムにクラスタに割り振り結果的に似たものが同じグループに入るようにする方法である．計算量は，階層的手法の場合 $O(N^3)$，非階層的手法の場合は $O(N)$ であることが多い．

階層的手法の代表的なものとしてウォード法が，非階層的手法の代表的なものとして k-means 法[9)] が挙げられる．特に k-means 法は，次章で詳しく述べる SimPoint でも採用されている．

シミュレーションの手順

シミュレーション・ポイントに基づくシミュレーションの手順は以下のようにまとめられる：

(1) エミュレーションにより基本ブロックの時系列を得る．
(2) 得られた基本ブロック列を適当に分割し，クラスタリングを施す．
(3) 各クラスタから 1 つを選び，シミュレーション・ポイントとする．
(4) シミュレーション・ポイントのシミュレーションを行い，その区間の評価値（IPC など）を得る．
(5) 重みづけ平均により，プログラム全体の評価値を得る．

3. SimPoint

シミュレーション・ポイント選択のためのフェーズ検出手法としては，**SimPoint**[7),10)] が代表的である．本章では，SimPoint を紹介する．

3.1 SimPoint の概要

1 章で述べたように，SimPoint は，プログラムの実行を 1M〜100M 命令の固定長のインターバルに分割し，それらの基本ブロック・ベクトルをクラスタリングすることによりフェーズを検出している．

SimPoint は，以下のようにしてフェーズ検出を行う：

(1) **インターバルへの分割**　まず，PC の列を 1M〜100M 命令程度の固定長のインターバルに区切る．インターバルの長さは，計算量と精度のトレードオフによって決める．
(2) **基本ブロック・ベクトル生成**　次に，各インターバルに対して，基本ブロック・ベクトルを生成する．
(3) **クラスタリング**　最後に，得られた基本ブロック・ベクトルの集合に k-means 法によるクラスタリングを施す．同一のクラスタに分類されたインターバルがフェーズとみなされる．

3.2 k-means 法

基本ブロック・ベクトルの各次元はプログラム中に存在する基本ブロックに対応し，各次元の値はそのインターバル内でその基本ブロックが実行された回数を表す．したがって基本ブロック・ベクトルは，次元数が数万〜数十万以上の，多次元のスパースなベクトルになる．

SimPoint は基本ブロック・ベクトルのクラスタリ

ングのために，k-means 法を用いている．k-means 法のアルゴリズムは以下のとおりである：

(1) 各データをランダムに k 個のクラスタに分類する．
(2) 各クラスタの平均値を計算する．
(3) 各データがどのクラスタの平均値に近いか計算し，最も近いクラスタに分類し直す．
(4) (2), (3) を繰り返す．データの移動がなくなった時点で終了する．

k-means 法では，k の値を予め決める必要があり，k-means 法自体によっては最適な k の値は分からない．したがって，さまざまな k を用いて k-means 法を実行し，最適な k を選択するという方法を採らざるを得ない．

図 5 下に，SPEC2000 の gzip に対してクラスタリングを施した結果を示す[1]．同図からは，gzip は 6 つのクラスタにクラスタリングされることが分かる．

SimPoint は，各クラスタの平均値に最も近いインターバルをシミュレーション・ポイントとして選択する．

3.3 SimPoint の問題点

1 章で述べたように，SimPoint の問題点は，命令列を固定長インターバルで分割していることに起因する．SimPoint は，1M〜100M 命令程度という長い固定長のインターバルを用いているため，その精度に関して，以下のような 2 つの問題がある：

1. インターバルより十分に長いフェーズしか検出できない．
2. 内分点の基本ブロック・ベクトルが存在するため，正しいクラスタリングが難しい．

k-means 法のような一般的なクラスタリング手法を用いるのは，内分点の基本ブロック・ベクトルが多数存在するためであると言える．

4. 提 案 手 法

本章では，提案手法について詳細に説明する．提案手法は，基本ブロック列を可変長の**セグメント**に分割してクラスタリングを行う．セグメントへの分割は，短い固定長の**ユニット**に基づいて行われる．以下，4.1 節でセグメントとユニットの関係について述べた後，4.2 節でセグメントへの分割について，4.3 節でクラスタリングについて，それぞれ説明する．

4.1 セグメントとユニット

従来の手法の問題点は固定長の基本ブロック列に分割していることであった．その問題を解決するために，提案手法では可変長の基本ブロック列に分割する．ここで，ユニットとセグメントを以下のように定義する：

ユニット 固定長の基本ブロック列．インターバルと比べ十分に小さく，基本ブロック 100 個程度とする．

セグメント 1 個以上の連続するユニットの系列．

ユニットは，専ら，次節で述べるセグメントへの分割を行う際に分割点を求めるために導入するものである．

4.2 セグメントへの分割

まず，プログラム全体の基本ブロックをユニット毎に分割する．

次に，連続する 2 つのユニットの基本ブロック・ベクトルのマンハッタン距離を計算し，閾値以上の値であれば分割点とする．**マンハッタン距離**は 2 つのベクトルの各座標の差の絶対値の総和である．分割点毎に区切った基本ブロック列をセグメントとなる．5 章の評価では，ユニットの大きさは 100 基本ブロック，閾値は 100 とした．

フェーズの切れ目ではその前後のユニットの基本ブロック・ベクトルの距離が大きな値となるので，分割点はフェーズの切れ目であるための十分条件と言える．ただし，フェーズの切れ目でなくても基本ブロック・ベクトルの距離が大きくなることはあり得る．

1 章でも参照したが，従来手法のインターバルと提案手法のセグメント，ユニットを比較したものが図 1 である．

上の図ではインターバル I_1，I_3 内ではフェーズが変化していないので，生成される基本ブロック・ベクトルはそのインターバルが属するフェーズの特徴を表すものとなる．しかし，フェーズの切れ目を含むインターバル I_2，I_4 では切れ目の前後両方のフェーズの特徴を持つ基本ブロック・ベクトルが生成され，その基本ブロック・ベクトルは I_1，I_3 どちらのフェーズとも異なる基本ブロック・ベクトルとなる．あるいは，全く別のフェーズと基本ブロック・ベクトルの値が近ければ意図しないクラスに分類されてしまう．

一方下の図では，十分に小さい基本ブロック列で区切ったユニットの距離を比較することで可変長のセグメントに分割しているため，フェーズの切れ目毎にセグメントを区切ることが可能となる．内分点的なユニットが存在することが避けられないが，ユニットの大きさがフェーズに比べ十分小さければセグメントの基本ブロック・ベクトルに与える影響は少ない．

なお，前述したように，ユニットは専らセグメントへの分割点を求めるために導入したものであり，分割点が十分な精度で求まるのであれば別の手法を用いても構わない．固定長のウインドウをスライドさせる方法や，ウェーブレット解析などを用いれば，より高精度な分割が可能になるであろう．

4.3 セグメントのクラスタリング

次に，得られたセグメントをクラスタリングする．

セグメントの長さはセグメント毎に異なるので，セグメントの基本ブロック・ベクトルをセグメント長（ユニット数で計る）で割って正規化する．その結果，例えば同じ命令が 100 回繰り返されるループと 500 回繰り返されるループは同じクラスタに分類される．

また可変長のセグメントに分類したことはクラスタリング手法にも影響する．前述したように，インターバルには内分点的な基本ブロック・ベクトルが存在するため，k-means 法のようなクラスタリング・アルゴリズムを用いる必要がある．一方，セグメントの基本ブロック・ベクトルはセグメント毎に類似度の高いものとなり，分散が小さいことが期待できる（図 2）．

よって提案手法では以下のような単純なアルゴリズムを用いた：

(1) セグメントと，各クラスタの中心との距離を比較する．
(2) 距離が閾値以内のクラスタがあれば，最も近いクラスタに分類する．
(3) 閾値以内のクラスタがなければ，新しいクラスタを生成する．
(4) 次のセグメントに対して，(1) から繰り返す．

この手法では，閾値の大小によってクラスタ数が決まる．したがって，k-means 法のように何回も実行して最適なクラスタ数を求める必要はない．

5. 評　　価

本章では，提案手法の評価結果について述べる．

5.1 gzip の結果

図 5 に，SPEC CPU2000 の gzip に対して，提案手法と SimPoint を適用した結果を示す．同図は，4 つのグラフからなる．4 つのグラフの横軸は，実行された基本ブロックの数である．すなわち，グラフの左端がプログラムの開始時に，右端が終了時に，それぞれ対応する．4 つのグラフのうち，一番上が CPI，2 番目が基本ブロックの実行頻度のカラーマップ，3 番目と 4 番目が，それぞれ，提案手法と SimPoint の結果である．

2 番目の基本ブロックの頻度別カラーマップは，以下のようにして得た．このグラフの横軸は実行された基本ブロックの数，縦軸は基本ブロックの ID である．基本ブロックの ID は，基本ブロックの出現順に番号を振ったものである．グラフ全体は，横 1,000 個，縦 1,200 個のブロックに分割されている．各ブロックの中で各基本ブロックが何回実行されているかをカウントし，その数の対数に対して色を割り振っている．

3 番目のグラフは，提案手法の結果を色別に表したものである．異なるクラスタに異なる色を割り当てている．4 番目のグラフは SimPoint によるクラスタリング結果である．

ただし，提案手法では test 入力を与えたもの，SimPoint では ref 入力を与えたものとなっている．そのため，結果を直接比較することはできないが，以下のことが言える．

まず，提案手法のほうが桁違いに細かくクラスタリングしていることが分かる．SimPoint ではクラスタ数 6 に対し，提案手法ではクラスタ数 1242 となった．

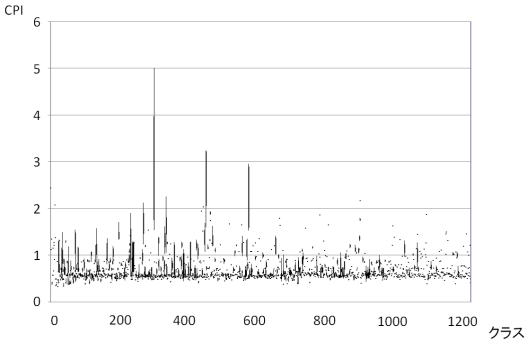

図 3 クラスタ毎の CPI の分散（gzip）

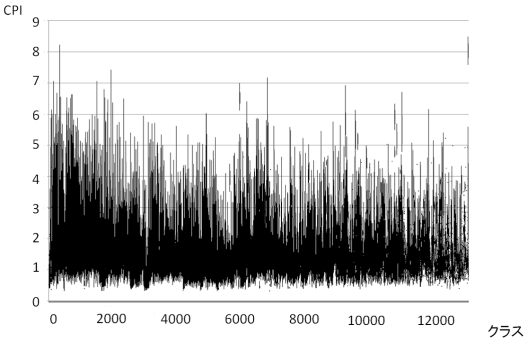

図 4 クラスタ毎の CPI の分散（gcc）

2 番目のグラフ（カラーマップ）において，0〜0.5，0.5〜1.0（G 命令）の基本ブロック ID 850 のあたりに注目されたい．0.5〜1.0 の 850 あたりにはうっすらと帯が見えており，0〜0.5 と 0.5〜1.0 は別のフェーズであることを示唆している．SimPoint は，これらを同じフェーズとしてクラスタ ID 2 に分類しているが，提案手法では別のフェーズとしている．

5.2 クラスタ毎の CPI の分散

次に，クラスタ毎にセグメントの CPI の分散を調べた．結果を**図 3**，**4** に示す．図 3，4 は横軸にクラスタ，縦軸にそのクラスタに属する全てのセグメントの CPI をプロットしてある．gzip ではクラスタ毎のセグメントの CPI の分散が小さく，正しくフェーズ検出できていると言える．はずれ値を含むクラスタでは分散が大きくなっているものもあるが，はずれ値の CPI を持つセグメントはフェーズの切れ目を含むもの，かつセグメントの長さが短いものであった．ただし，gzip ではセグメントあたりの平均ユニット数が大きいので，セグメントの長さが小さいものは相対的に重みが小さくなり，CPI 推定にはあまり影響しない．

一方，gcc では分散が大きく，gzip より正しくフェーズを検出できていない．これは gzip ではセグメントの長さが長いものが多く，フェーズの切れ目がセグメントの基本ブロック・ベクトルに与える影響が相対的に

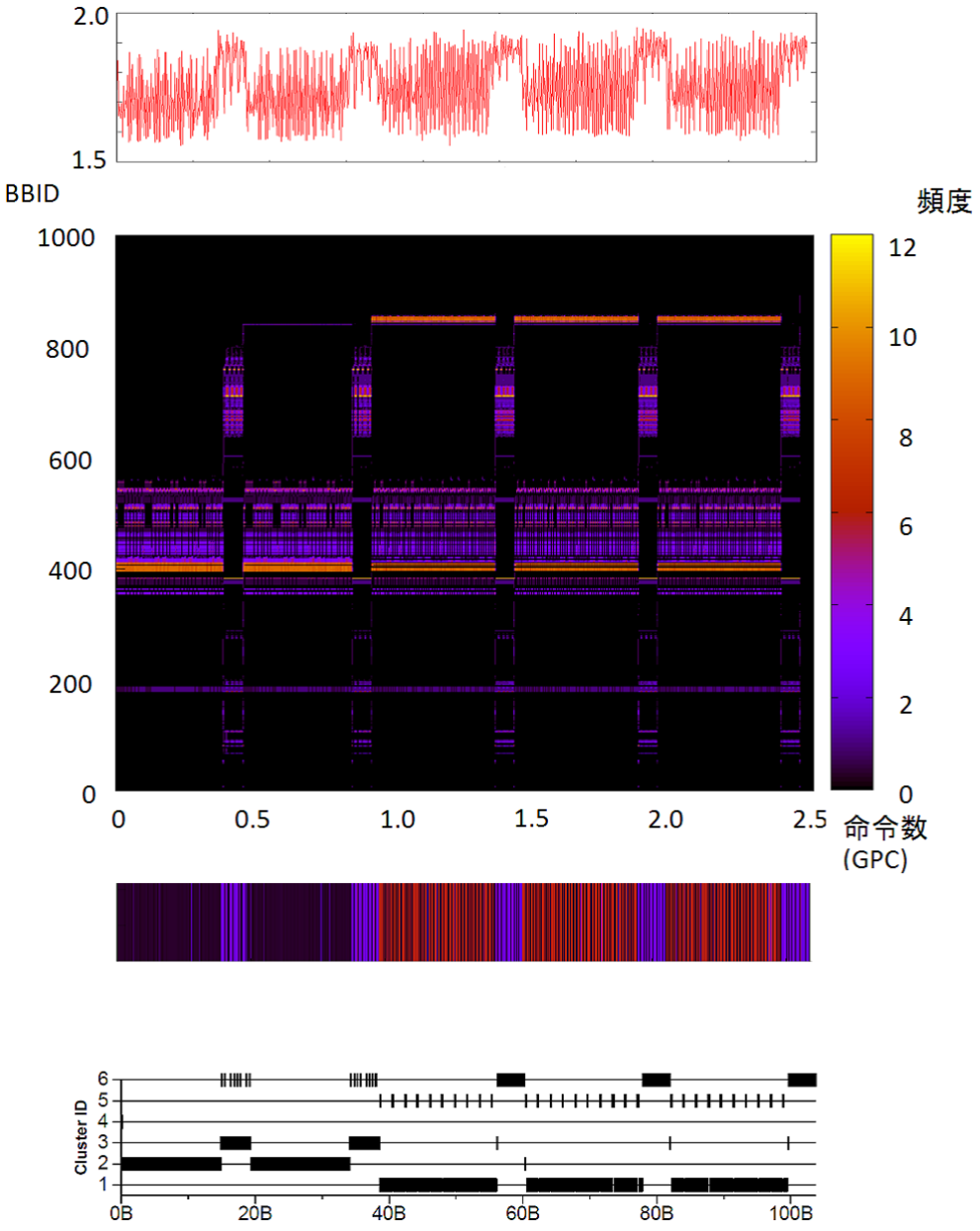

図 5 gzip の結果
上から，CPI，基本ブロックの実行頻度，提案手法の結果，SimPoint の結果

表 1 CPI 誤差

プログラム	CPI 誤差 (%)
gzip	0.9074
gcc	3.507
mcf	10.32
vortex	0.07731
bzip2	0.03947
twolf	2.355

表 2 クラスタ数と CPI 誤差（gzip）

閾値	クラスタ数	CPI 誤差 (%)
1	4868	0.03043
10	1242	0.9074

小さいことに対して，gcc ではセグメントの長さが短く，フェーズの切れ目がセグメントに与える影響が相対的に大きいからだと思われる．セグメントあたりの平均ユニット数が gzip では 200，gcc では 1.5 であった．以上よりセグメントあたりの平均ユニット数が少ないベンチマークでは誤差が大きくなると考えられる．

5.3 CPI 推 定

次に CPI の推定を行った．

各クラスタの中で最初に現れたセグメントの区間 CPI を測定し．クラスタ毎の合計の PC 長で重み付けすることで全体のシミュレーション実行による CPI を推定し，その誤差を**表 1** に示した．

SimPoint では，正確な値は載っていないが，誤差の平均は 3% 程度となっている．提案手法では gzip, vortex，bzip2 は高精度で CPI を推定できているが，gcc，mcf，twolf では精度が悪い．

次に，gzip に関して，クラスタリングの閾値を 1 に変えて CPI の測定をした．**表 2** にクラスタ数と誤差を示す（gzip のセグメント数は 16648）．クラスタ数，誤差ともに少ないほうが良いが，表 2 よりクラスタの数が増えるほど誤差は小さくなる，つまりクラスタ数と誤差はトレードオフの関係にあることが分かる．

6. おわりに

本稿では可変長のセグメントを用いるフェーズ検出手法を提案した．SPEC CPU2000 ベンチマークを用いた評価の結果，gzip，vortex，bzip2 では誤差 0.1%以内と良い結果を得ることができた．一方で，gcc，twolf，mcf についてはかなり大きい誤差が出てしまった．特に mcf では 10%を超えてしまっている．

予備的に，クラスタリングの評価尺度としてよく用いられる **BIC（Bayesian Information Criterion，ベイズ情報量基準）**を計測してみたところ，gcc，twolf，mcf についても，SimPoint よりよい値を示した．このことは，単に基本ブロック系列に対するフェーズの検出という意味においては，提案手法はうまくいっていることを示唆している．

CPI の誤差が大きくなったのは，以下のような理由によるものと推測される：

- セグメントが短いと区間 CPI が正しく測定できない．
  今回誤差が大きかったプログラムは，平均セグメント長が短いことが挙げられる．
  問題は，シミュレーション・ポイントが極端に短い場合にはシミュレーション・ポイントのみをシミュレーションしても正確な区間 CPI が得られない．シミュレーション・ポイント以外の点では，エミュレーションによりシミュレーションをスキップするため，シミュレーション・ポイントの開始点ではキャッシュや分岐予測器が暖まっていない．そのため，シミュレーション・ポイントが極端に短いと，CPI が実際より悪い値となる．
  解決策としては，シミュレーション・ポイントの直前の数 M 命令程度も合わせてシミュレーションする，エミュレーション時にもキャッシュと予測器のシミュレーションは行う などが考えられる．
- PC（基本ブロック）の系列が一致しても，フェーズとして同じであるとは限らない．
  特に mcf で顕著であるが，プログラムの全く同じ静的な部分を実行していても，動的にはキャッシュのヒット/ミスの具合が大きく異なることがある．
  解決策としては，キャッシュ・シミュレーションを行い，キャッシュ・ヒット/ミスの具合まで入力として，シミュレーション・ポイント選択を行うなどが考えられる．

SimPoint は，比較的長いインターバルを用いるために，これらの問題が顕在化せずに済んでいたのだと考えられる．結局，単に PC（基本ブロック）の系列だけから精度よくフェーズ検出を行うだけでは，シミュレーション・ポイントの選択のためには不十分であることを示唆している．

謝辞

本論文の研究は一部，文部科学省科学研究費補助金 No. 23300013 による．

参 考 文 献

1) Sherwood, T., Perelman, E., Hamerly, G., Sair, S. and Calder, B.: Discovering and Exploiting Program Phases, *Int'l Symp. on Computer Architecture (ISCA)* (2002).
2) Sherwood, T., Perelman, E. and Calder, B.: Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications, *Int'l Conf. on Parallel Architectures and Compilation Techniques* (2001).
3) Sherwood, T. and Calder, B.: Time varying behavior of programs, Technical Report UCSD-

CS99-630, UC San Diego (1999).
4) Hind, M., Rajan, V. and Sweeney, P.F.: Phase detection: A problem classification, Technical Report 22887, IBM Research (2003).
5) Dhodapkar, A. S. and Smith, J. E.: Managing Multi-Configuration Hardware via Dynamic Working Set Analysis, *Int'l Symp. on Microarchitecture (MICRO)*, pp. 84–93 (2003).
6) 上野裕也, ルォンディンフォン, 高田正法, 田代大輔, 坂井修一: Signature を用いたフェーズ分類手法の改良とキャッシュの電力削減への応用, 信学技報, Vol. 105, No. 42, pp. 25–30 (2005).
7) Perelman, E., Hamerly, G., Biesbrouck, M.V., Sherwood, T. and Calder, B.: Using SimPoint for Accurate and Efficient Simulation, *SIGMETRICS* (2003).
8) Sherwood, T., Perelman, E., Hamerly, G. and Calder, B.: Automatically characterizing large scale program behavior, *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (2002).
9) Hamerly, G. and Elkan, C.: Learning the k in k-means, Technical Report CS2002-0716, University of California (2002).
10) Hamerly, G., Perelman, E. and Calder, B.: How to use SimPoint to pick simulation points, *SIGMETRICS Perform. Eval. Rev.*, Vol. 31, pp. 25–30 (2004).