Vol.2013-ARC-206 No.26
2013/8/1

# 可変長セグメントを用いたフェーズ検出手法

早川 薫[†1]　倉田 成己[†1]　五島 正裕[†1]　坂井 修一[†1]

**概要**：プロセッサの研究・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，シミュレーションする命令数の削減である．シミュレーションする命令数を削減する手法の 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．本稿では可変長のセグメントを用いたフェーズ検出手法を提案した．また提案手法をプロセッサ・シミュレータ鬼斬弐に実装した．

## 1. はじめに

プロセッサの研究・開発には，**シミュレーション**によるプロセッサの性能測定が必要不可欠である．シミュレーションとは，プロセッサの 1 サイクルごとの振る舞いをソフトウェア上で模擬することである．シミュレーションにより，実際にハードウェアを組むことなくプロセッサの性能を測定できる．

しかし，シミュレーションは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，cycle-accurate なシミュレータでは 1000 以上にもなり，実機で 10 分かかる処理がシミュレーションでは 10,000 分 ≃7 日以上かかることになる．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・ベクトルは各基本ブロック（途中に分岐や合流を含まない命令列, Basic Block, BB）が何回実行されたかを示すベクトルである．基本ブロック・ベクトルのインデックスはプログラム中に存在する基本ブロックの ID に対応する．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

[†1] 現在，東京大学大学院情報理工学系研究科
Presently with Graduate School of Information Science and Technology, The University of Tokyo

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 8** 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．

**可変長セグメントを用いたフェーズ検出手法**

固定長インターバルを用いると上記のような問題が起こる．フェーズの切れ目に近い所で基本ブロック列を分割すれば，SimPoint の問題は解決される．そのようにフェーズの切れ目に近い所で分割された基本ブロック列を**セグメント**と呼ぶ．

セグメントを用いたフェーズ検出は以下のように行う：

(1) **セグメントへの分割**　フェーズの切れ目を見つけて，セグメントに分割する．

(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基本ブロック数で正規化しておく．

(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．

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. セグメントはフェーズの切れ目に合わせて基本ブロック列を分割するので検出可能なフェーズの大きさに制限がない．
2. セグメントの基本ブロック・ベクトルには内分点が存在しないので．クラスタリングは容易である．

セグメントを生成するために固定長ユニットを用いた手法がある．[3] この手法ではまず基本ブロック列全体を短い固定長のユニットに分割する．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ことでフェーズの切れ目を検出していた．しかし，**図 2** 上に示されている

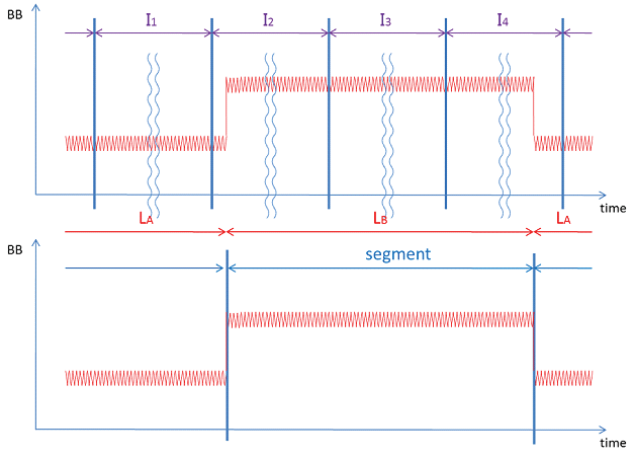


**図 1**　SimPoint（上）と可変長セグメントを用いた（下）のフェーズ検出

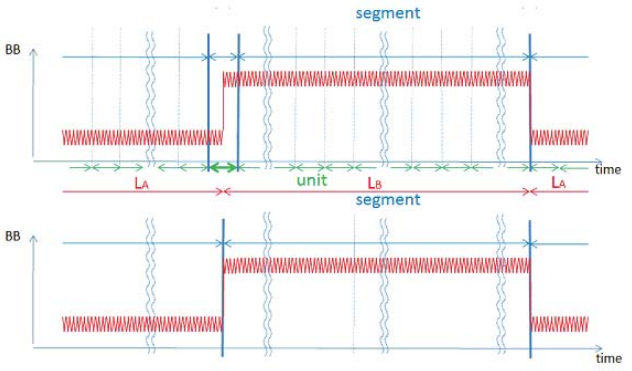


**図 2**　固定長ユニットを用いた手法（上）と提案手法（下）のフェーズ検出

ように，ユニットの中にフェーズの切れ目があった場合に適切に基本ブロック列を分割できない可能性があった．

そこで本稿では，基本ブロック列をメディアンフィルタに通し，セグメントを生成する．この方法により，図 2 下のようにフェーズの切れ目に合わせたセグメントを生成が可能になるため，ユニットで発生する問題を解決できる．

本稿は，以下のように構成されている．続く 2 章でシミュレーション・ポイントの必要性とその検出方法について述べ，次に 3 章では既存手法である SimPoint についてまとめる．4 章でユニットを用いた手法について説明する．5 章で提案手法に詳しく説明し，6 章で評価結果を示す．

## 2. シミュレーション・ポイントの必要性とその検出方法

本章では，シミュレーション・ポイントの必要性とその検出方法，および評価方法について述べる．

### 2.1 シミュレーション・ポイントが必要な理由

プロセッサの評価を SPECCPU2006 ベンチマークを用いて行うとき，最初の 1G 命令をスキップして 100M 命令のみ実行する方法がよく用いられる．しかし，この方法は

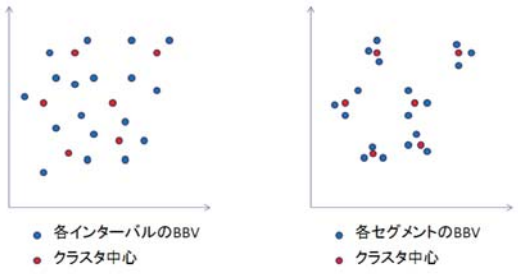


**図 3** 基本ブロック・ベクトルの分散

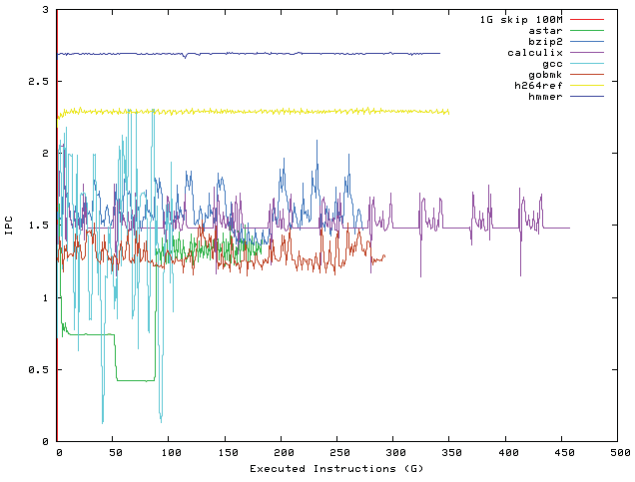


**図 4** SPECCPU2006 ref 入力の区間 IPC(1)

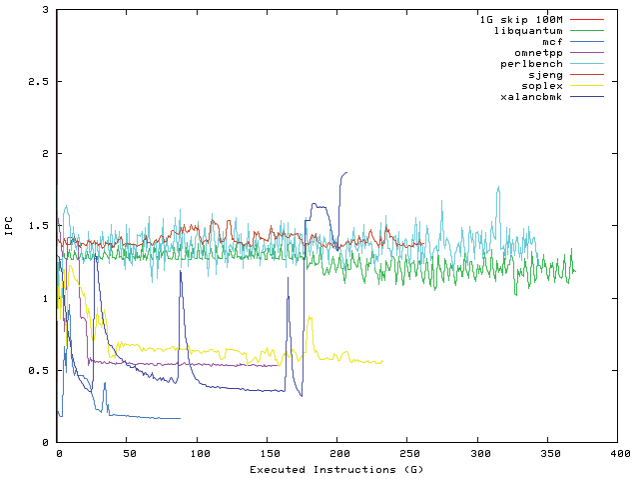


**図 5** SPECCPU2006 ref 入力の区間 IPC(2)

不適切であることを，**図 4**, **図 5** を用いて説明する．

図 4, 図 5 は，SPECCPU2006 ref 入力のうち，14 本の区間 IPC をプロットしたものである．gcc と gobmk は最後まで，その他は途中までの結果をプロットした．横軸は実行命令数で単位は G 命令，縦軸は区間 IPC である．また，区間の長さは 1G 命令とした．1G 命令スキップして 100M 命令実行する位置を，赤い縦線で表した．

図 4, 図 5 からわかるように，astar, mcf, omnetpp などのベンチマークは，1G 命令スキップしてもまだ初期化が終わっていない．そのため，そこから 100M 命令実行しても，ベンチマークの特徴的な部分を実行できない．また，ベンチマークの特徴的な部分が複数個所存在するものが多いため，1G 命令スキップして 100M 命令実行するだけでは，全ての特徴的な部分を実行できない．

以上の理由より，全ベンチマークを 1G 命令スキップして 100M 命令実行する方法は不適切であり，各ベンチマークの特徴的な部分をそれぞれ実行することが必要だと言える．次節で説明するが，このような特徴的な部分をシミュレーション・ポイントと呼ぶ．

### 2.2 シミュレーション・ポイントとフェーズ

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全体のプロセッサの振る舞いが推定できる部分のことである．シミュレーション・ポイントが存在するのは，プログラムには繰り返し構造が存在するからである．

このプログラムの繰り返し構造により，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似ている部分に分割できる．本稿では，曖昧さを避けるため，この各部分のことを**フェーズ**，プロセッサの振る舞いが互いに似ているフェーズの集合を**クラスタ**と定義する[*1]．この定義によれば，フェーズ検出とは，プログラムをフェーズに分け，フェーズをクラスタに分類することとなる．

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，PC の列を**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．

### 2.3 シミュレーション・ポイントの長さ

シミュレーション・ポイントを実行するときは，通常そこまでをエミュレーション等でスキップする．しかしこのスキップにより，シミュレーション・ポイントを実行し始めるときのパイプラインは，命令が詰まっていない状態となっている．その結果，シミュレーション・ポイントの CPI などが真の値からずれてしまう．

この誤差の影響を小さくするために，シミュレーション・ポイントはある一定以上の長さが必要となる．**図 6** は，perlbench のシミュレーション・ポイントの長さと CPI 誤差の関係をプロットしたもので，横軸はシミュレーション・ポイントの平均の長さ，縦軸は CPI の相対誤差を示している．図 6 からわかるように，シミュレーション・ポイント

[*1] 既存の論文では，クラスタもフェーズと呼ばれている [4], [5].

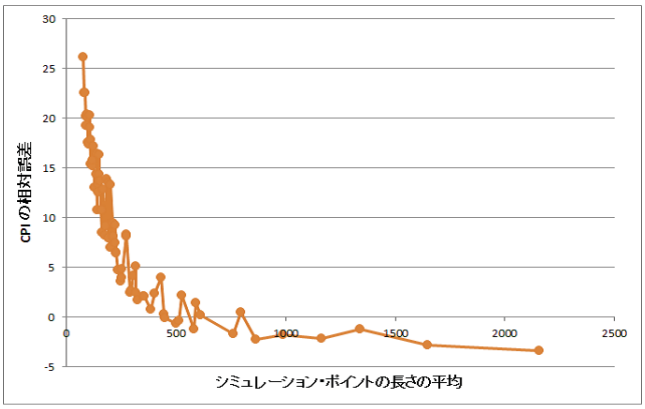


**図 6** perlbench のシミュレーション・ポイントの長さと CPI 誤差の関係

の長さを約 500 命令以上取れば，誤差をほぼ無視することができる．

### 2.4 フェーズ検出手法の評価方法

フェーズ検出手法の評価の手順は以下の通りである：

(1) あらかじめプログラム全体を実行し，プロセッサの評価値（CPI など）を得ておく．
(2) **エミュレーション**（後述）により基本ブロック列を得る．
(3) 基本ブロック列に対してフェーズ検出を行い，シミュレーション・ポイントを選択する．
(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取得し，あらかじめ取得しておいた評価値との誤差を測る．

なおエミュレーションとは，プロセッサの内部状態までは再現せずに，プログラム実行の出力が実機の出力と同じになるようにソフトウェア上で模擬することである．

最終的に取得した誤差の値，およびシミュレーション・ポイントがプログラム全体に対し占める割合が小さいほど，良いフェーズ検出手法だと言える．

### 2.5 クラスタリング手法

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．

非階層的手法の代表的なものとして，SimPoint で採用されている k-means 法 [6] がある．k-means 法については，3.1.2 節で詳しく述べる．

## 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], [8] が代表的である．

本章では SimPoint について述べる．

### 3.1 SimPoint

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．

#### 3.1.1 SimPoint の概要

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．

#### 3.1.2 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，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．

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) と (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 7** 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 は大きく 6 つにクラスタリングされることが分かる．

SimPoint は，シミュレーション・ポイントとして，各クラスタの平均値に最も近いインターバルの基本ブロック・

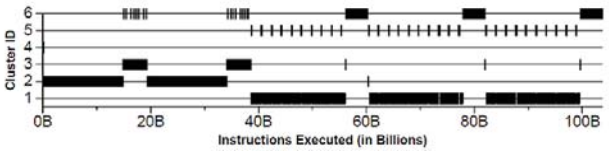


**図 7** SimPoint による gzip のフェーズ分類

ベクトルを選ぶ.

#### 3.1.3 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. 固定長インターバルを用いないフェーズ検出手法

SimPoint 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る手法が提案された.

本章では, 固定長ユニットを用いて可変長セグメントを生成しフェーズ検出を行う手法について述べる. 以下, 4.1 節でユニットを用いたセグメントへの生成について, 4.2 節でクラスタリングについて, それぞれ説明する. そして 4.3 節で SimPoint との比較とこの手法の問題点について述べる.

### 4.1 ユニットを用いたセグメントの生成

ユニットは固定長の基本ブロック列である. インターバルと比べ十分に小さく, 基本ブロック 100 個程度とされる. 次に述べる, セグメントへの分割を行うために導入されたものである.

まず基本ブロック列全体をユニットに分割する. 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そこで分割する. 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と定義される. このように分割された基本ブロック列をセグメントとする.

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ため, 分割点毎に区切った基本ブロック列をセグメントとし, フェーズと区別している.

以上のようにしてこの手法では基本ブロック列を可変長であるセグメントに分割をしている.

### 4.2 クラスタリング

次に分割したセグメントに対しクラスタリングを行う. セグメントの長さはセグメント毎に異なるので, クラスタリングするためにはセグメントの基本ブロック・ベクトルを正規化する. 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. インターバルの基本ブロック・ベクトルはどのフェーズにも属さない基本ブロック・ベクトルが存在するため, 基本ブロック・ベクトルの分散が大きくなる. 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 8)

よって可変長セグメントを用いる手法では以下のような単純なアルゴリズムを用いてクラスタリングを行う.

(1) 各セグメントの基本ブロック・ベクトルと全てのクラスタの中心との距離を比較
(2) 距離が閾値以内のクラスタがあれば, 最も距離の短いクラスタに分類し, 中心を再計算
(3) 閾値以内のクラスタがなければ新しいクラスタを作成
(4) 次のセグメントに対して, (1) から (3) を繰り返す

このクラスタリング方法では, 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.

### 4.3 SimPoint との比較と問題点

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より以下の点で優れている：

(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.

しかし, SimPoint とこの手法を比較すると優れていることがわかるが, 問題点もある. 上にも書いたように, ユ

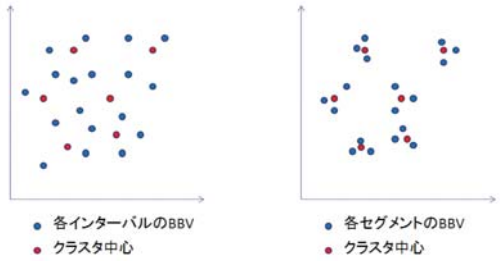


**図 8** 基本ブロック・ベクトルの分散

ニットは固定長であるため，内分点的なユニットが存在することは避けられない．そのため，セグメントに含まれるユニットの数が少ないと内分点の影響が大きくなる．

## 5. 提案手法:可変長セグメントを用いたフェーズ検出手法

4 章では固定長ユニットから生成した可変長セグメントを用いる手法についてとその問題点を説明した．その問題を解決するために，提案手法ではメディアンフィルタを用いてセグメントを生成し，フェーズを検出する．

本章では，5.1 節で提案手法におけるセグメント生成について，5.2 節でクラスタリングについて述べる．最後に 5.3 節でユニットを用いた手法との比較を行う．

### 5.1 提案手法における可変長セグメントの生成

提案手法におけるセグメント生成にはメディアンフィルタを用いる．メディアンフィルタは、n × n の局所領域における値を並び替え、 その中央の値を領域の中心に対し出力する．

本手法では，一次元のメディアンフィルタを用いる．基本ブロック ID を入力して，出力を元の基本ブロック列の中心と関連付けをする．以下メディアンフィルタを用いたセグメントの生成方法の流れを述べる：

(1) 基本ブロック列全体に対しフィルタを 1 基本ブロックずつスライドさせ，基本ブロック ID を入力する．
(2) フィルタから出力された基本ブロック ID の値を元の基本ブロック列の中心と関連付ける．
(3) 全ての関連付けが終了後，隣り合う出力値の差の絶対値が指定された閾値以上なら基本ブロック列を分割する
(4) 分割された基本ブロック列をセグメントとして扱う

メディアンフィルタを用いたセグメント生成についての具体例を図 9 を用いて説明する．図ではそれぞれ縦軸に基本ブロック ID を取り，横軸は各基本ブロックが先頭から何番目のものであるかを表す．上の図は元の基本ブロック列に対しフィルタがスライドしている様子を表しており，下の図はフィルタからの出力された値の関連付けが終了した後の様子を表している．今回の例ではフィルタのサイズを 5 とし，閾値を 50 とする．

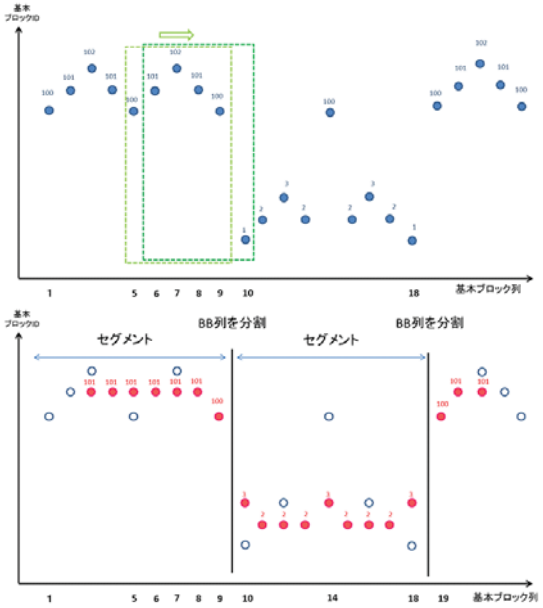


**図 9** メディアンフィルタを用いたセグメント生成の具体例

上の図で, 黄緑色のフィルタへの入力は 5〜9 の位置にある基本ブロック列の ID(100,.101,102,101,100) である．また 1 基本ブロック分スライドした緑色のフィルタへの入力は 6〜10 の位置にある基本ブロック列の ID(101,102,101,100,1) である．黄緑色のフィルタからの出力は 101 であり，7 の位置にある基本ブロックとの関連付けを行う．また緑フィルタからの出力も 101 であり 8 の位置にある基本ブロックとの関連付けを行う．下の図で 9 の位置にある基本ブロックへ関連付けされた値である 100 と，10 の位置にある基本ブロックへ関連付けされた値である 3 の差の絶対値は 97 であり，閾値を超えるので基本ブロック列の分割を行う．同様に 18 から 19 の位置でも基本ブロック列を分割し，10〜18 までの基本ブロック列を一つのセグメントとして扱う．

以上のような方法を用いると 1 基本ブロックごとにフェーズの切れ目があるかどうかを確認できるため，生成されるセグメントには基本ブロック・ベクトルの内分点が存在しない．また，図の 14 の位置にあるような小さな基本ブロック ID の変化を受けることなくフェーズの切れ目を検出することができる．

### 5.2 クラスタリング

提案手法におけるクラスタリングは 4.2 節 で説明したものと同様の方法を用いている．前節に述べたようにセグメントの基本ブロック・ベクトル内分点が存在しないのでより類似度の高いセグメントが同じクラスタにまとまりやすくなる．

表 1　プロセッサの構成

| | |
|---|---|
| ISA | Alpha21164A |
| pipeline stages | Fetch:3,Rename:2,Dispatch:2,Issue:4 |
| fetch width | 4 inst. |
| issue width | Int:2, FP:2, Mem:2 |
| instruction window | Int:32,FP:16, Mem:16 |
| branch predictor | 8KB g-share |
| BTB | 2K entries,4way |
| RAS | 8 entries |
| L1C | 32KB,4way,3cycles,64B/line |
| L2C | 4MB,8way,10cycles,64B/line |
| main memory | 200cycles |

### 5.3　固定長ユニットを用いた手法と提案手法におけるセグメントの比較

4.3 節で述べたように，ユニットを用いた手法では生成されたセグメントの基本ブロック・ベクトルに内分点が存在することは避けられないという問題があった．提案手法ではメディアンフィルタを基本ブロック列に対し 1 基本ブロックずつスライドさせ，その出力値を用いてセグメントを生成する．その結果図 2 下のようなセグメントが生成される．このようにして生成されたセグメントの基本ブロック・ベクトルには内分点が存在しない．よってユニットを用いた場合の問題を解決することができた．

## 6.　評価

### 6.1　クラスタリング結果

SPECCPU2006 の soplex の基本ブロック列を**図 10** に示す．同図は横軸に基本ブロック数，縦軸に基本ブロックの番号を表すグラフである．横軸，縦軸とも基本ブロック列をそのままプロットすると膨大な量になるため，1000*1200 のブロックに圧縮している．そして，圧縮されたブロックに含まれる基本ブロックの数を色で表している．

ユニットを用いた手法と提案手法のクラスタリング結果を**図 11** と**図 12** に示す．表示するための方法は図 10 と同様であるが，表示しているクラスタリング結果のカラーマップはクラスタ毎に所属している基本ブロック数が多い順に，基本ブロック列を並び替えて表示されている．また，所属しているクラスタが変わるたびに黄色の線が表示されている．よって，黄色の線で挟まれた間のカラーマップがより似通った形状，色を示していればクラスタごとに類似度の高い基本ブロック列がまとまっていると考えられる．

両手法とも，カラーマップで確認できる範囲においてはクラスタ内でより類似度の高い基本ブロック列が集まっていることが確認できる．しかし，ユニットを用いた手法に対し，提案手法のカラーマップの方がよりクラスタごとの基本ブロック列が多いため，提案手法を用いた方がよりセグメントの生成が適切であり，より類似度の高い基本ブロック列がまとまっていると考えることができる．

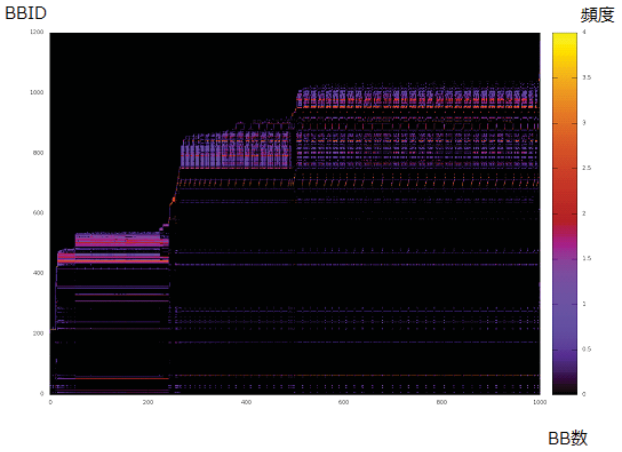


図 10　soplex の基本ブロック列

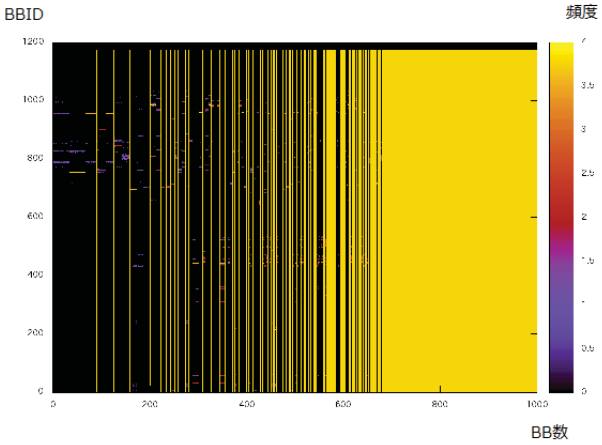


図 11　ユニットを用いた手法のクラスタリング結果

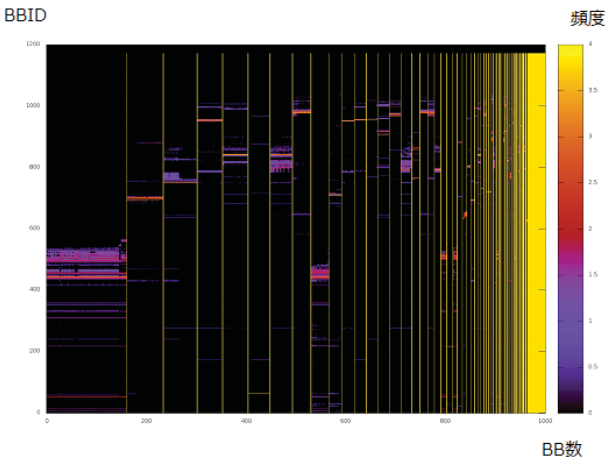


図 12　提案手法のクラスタリング結果

### 6.2　CPI 推定

#### 6.2.1　評価環境と評価した手法のパラメータ

次節で CPI 推定の結果を示すために，この節では評価を行った環境と評価した手法のパラメータについて説明する．

プロセッサ・シミュレータ「鬼斬弐」を用いてシミュレーション・ポイントを実行し，CPI 推定を行った．評価した

表 2　ユニットを用いた手法に関するパラメータ

| benchmark | ユニットサイズ | セグメント判定閾値 | クラスタリング閾値 |
|---|---|---|---|
| perlbench | 50BB | 80 | 12 |
| gobmk | 100BB | 60 | 6 |
| sopelx | 80BB | 128 | 12 |
| calculix | 100BB | 160 | 10 |
| libquantum | 100BB | 100 | 10 |
| xalancbmk | 100BB | 10 | 8 |

表 3　提案手法に関するパラメータ

| benchmark | フィルタサイズ | セグメント判定閾値 | クラスタリング閾値 |
|---|---|---|---|
| 全ベンチマーク | 256 | 64 | 8 |
| libquantum（チューニング後） | 16 | 16 | 6 |
| xalancbmk（チューニング後） | 512 | 256 | 10 |

プロセッサの基本的なパラメータは**表 1** の通りである．

評価は，SimPoint とユニットを用いた手法，提案手法についてそれぞれ行った．

評価対象として SPECCPU2006 の 400.perlbench, 445.gobmk，450.soplex，454.calculix，462.libquantum, 483.xalancbmk の 6 本のベンチマーク・プログラムを用いた．入力データ・セットには *test* を用いた．SimPoint ではのインターバルの長さを 1M 命令とした．

**表 2** に，ユニットを用いた手法に関してチューニングした後のパラメータを，**表 3** に提案手法におけるパラメータを示す．また提案手法において，libquantum と xalancbmk に関してはチューニング後の評価も行った．

### 6.2.2　評価結果

評価結果を図 13 に示す．グラフの横軸にシミュレーション全命令に対してシミュレーションポイントの占める割合を，縦軸は推定された CPI とベンチマーク全体をシミュレーションした時の CPI の相対誤差を示している．色でベンチマークの種類を，プロットの形で手法の種類を分類している．

一般に，シミュレーションの実行割合と，CPI 誤差はトレードオフであり，どちらとも数値が小さくなるようなプロットが出来ればよい手法だと言える．**図 13** からわかるように，perlbench, gobmk, soplex, libquantum に関しては，提案手法を用いると他のどの手法よりも原点に近い点が存在する．これは提案手法を用いると CPI 誤差が小さいかつ, シミュレーションの実行割合が少なくなるシミュレーションポイントを選択できていることを表している．

ユニットを用いた手法は非常に精度良くチューニングしないと良い結果が得られなかったが，提案手法はほとんどのベンチマークに対してチューニングせずに良い結果を得られた．また libquantum と xalancbmk に関しては，チューニングすることで提案手法でもより良い結果が得られた．

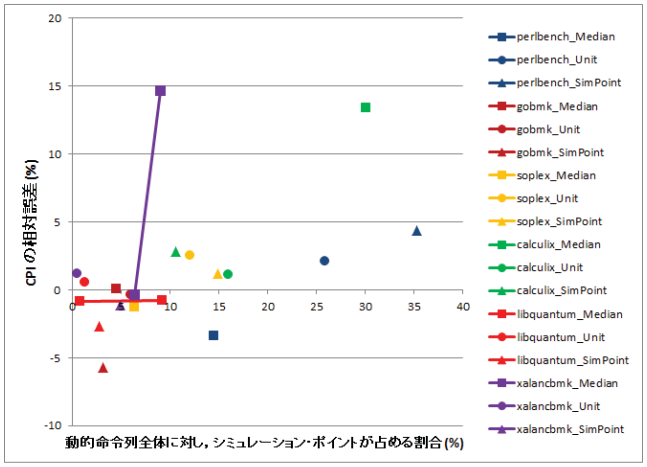


**図 13**　SPECCPU2006 における CPI 推定とシミュレーション実行割合

## 7.　おわりに

本稿では一次元のメディアンフィルタを基本ブロック列に対しスライドさせ，その出力結果を用いることで生成された可変長セグメントを用いるフェーズ検出手法を提案した．提案手法を用いて，SPECCPU2006 ベンチマークから 400.perlbench, 445.gobmk，450.soplex，454.calculix, 462.libquantum, 483.xalancbmk の test 入力のシミュレーションポイントを選択し CPI 誤差と実行割合を評価した．今回は 6 つのベンチマークの test 入力のみで評価を行ったが，今後は他のベンチマークや，実際のシミュレーションで使われる ref 入力などのより長いプログラムを用いた場合の性能評価を行う必要がある．

現在のクラスタリングの方法ではシミュレーションポイントの選択の際にセグメントの長さを考慮していない．この事により，可変長であるセグメントを用いるとシミュレーションポイントとして短かすぎるまたは長すぎり命令列を選択する可能性がある．シミュレーションポイントが短すぎると実行された命令列内でキャッシュミスなどが

発生した場合に CPI 推定に大きな影響を与える可能性がある．クラスタリングの閾値を低くした時に誤差が大きくなってしまったのはより多くのクラスタを作ってしまい，短いセグメントをシミュレーションポイントとして選択する割合が高まり誤差が生じたからだと考えられる．長いシミュレーションポイントを選択すると実行命令列が削減できない可能性がある．よって今後はシミュレーションポイントとして選択するセグメントの長さも考慮に入れる必要があると考えられる．

## 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, *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] 赤松雄一，五島正裕，坂井修一：固定長インターバルを用いないフェーズ検出手法，先進的計算基盤システムシンポジウム SACSIS2011，pp. 271–278 (2011).
[4] Sherwood, T. and Calder, B.: Time varying behavior of programs, Technical Report UCSD-CS99-630, UC San Diego (1999).
[5] Hind, M., Rajan, V. and Sweeney, P. F.: Phase detection: A problem classification, Technical Report 22887, IBM Research (2003).
[6] Hamerly, G. and Elkan, C.: Learning the k in k-means, Technical Report CS2002-0716, University of California (2002).
[7] Perelman, E., Hamerly, G., Biesbrouck, M. V., Sherwood, T. and Calder, B.: Using SimPoint for Accurate and Efficient Simulation, *SIGMETRICS* (2003).
[8] Hamerly, G., Perelman, E. and Calder, B.: How to Use SimPoint to Pick Simulation Points, *ACM SIGMETRICS Performance Evaluation Review* (2004).