Vol.2012-ARC-201 No.8
2012/8/1

固定長インターバルを用いないフェーズ検出手法の改良

早川 薫[†1]　塩谷 亮太[†2]　五島 正裕[†1]　坂井 修一[†1]

概要：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

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

[†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
[†2] 現在，名古屋大学工学研究科
Presently with Dept. of Electrical Engineering and Computer Science, Nagoya Univ

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．

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．

今回は提案手法をプロセッサ・シミュレータ鬼斬弐 [3] に実装して評価を行った．

本稿は，以下のように構成されている．続く 2 章でフェーズ検出について述べ，次に 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 章で提案手法をどのように実装したかを述べる．6 章で評価結果を示す．

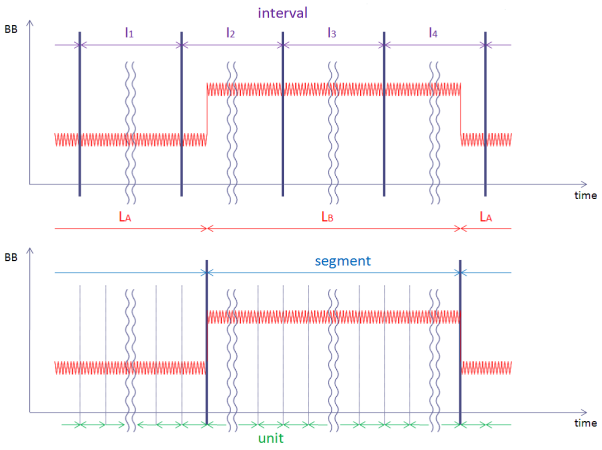

図 1　SimPoint（上）と提案手法（下）のフェーズ検出

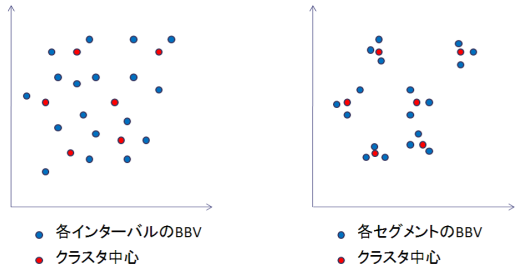

図 2　基本ブロック・ベクトルの分散

2. フェーズ検出とその評価

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とその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互いに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．

2.2 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 [6], [7]．省電力化のためには，例えば，クラスタ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．

本稿で行うのは，静的なフェーズ検出である．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を行うもので，主にシミュレーションの高速化などに利用される [8], [9].

2.3 シミュレーション・ポイントのためのフェーズ検出

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．

フェーズ検出の評価の手順は以下の通りである：

(1) エミュレーションにより基本ブロック列を得る．
(2) 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．
(3) 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．
(4) 重みづけ平均により，プログラム全体の評価値を得る．

2.4 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．

階層的手法の代表的なものとしてウォード法．非階層的手法の代表的なものとして，SimPointでも採用されているk-means法 [10] がある．k-means法については，3.2節で詳しく述べる．

[*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].

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[8], [11] 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．このとき，基本ブロック・ベクトルの各要素は「基本ブロックが出現した回数 × その基本ブロックに含まれる命令数」である．
(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．

3.2 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 3 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．

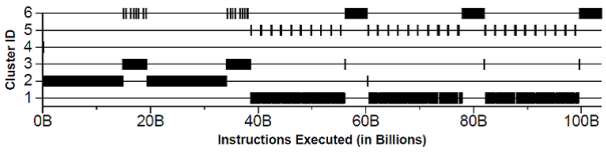

図 3 SimPoint による gzip のフェーズ分類

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・ベクトルを選ぶ.

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

4.1 基本ブロック単位での分割

従来の手法の問題点は固定長の命令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**に分割した.

SimPoint は命令単位でインターバルを区切っているため，その基本ブロック・ベクトルの各要素は「基本ブロックが出現した回数 × その基本ブロックに含まれる命令数」であった．それに対し提案手法では基本ブロック単位で分割を行うので，基本ブロック・ベクトルの各要素は単に「基本ブロックが出現した回数」でよい.

4.2 セグメントによる基本ブロック列の分割

まず**ユニット**と**セグメント**というものを次のように定義する.

- ユニット
  固定長の基本ブロック列．ただしフェーズと比べ十分に小さいものとする.
- セグメント
  可変長の基本ブロック列．ユニットの集まり.

まず基本ブロック列全体を固定長の基本ブロック列であるユニットに分割する．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と定義される．分割点毎に区切った基本ブロック列をセグメントとする．今回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，フェーズと区別した.

第一章でも参照したが，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.

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.

以上のようにして基本ブロック列を可変長であるセグメントに分割することができた.

4.3 クラスタリング

次に分割したセグメントをクラスタリングした．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必要がある．正規化することによりセグメントの長さが違っても実行される基本ブロックと各基本ブロックの回数の割合が同じであれば同じクラスタに分類することができる．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．今回は正規化により 100 基本ブロックあたりの基本ブロック・ベクトルをクラスタリングした.

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．これを示した図が図 2 である.

左のインターバルの基本ブロック・ベクトルは分散が大

きいので K-means 法などを用いてクラスタリングする．一方右のセグメントの基本ブロック・ベクトルは分散が小さいので各クラスタ中心との距離が閾値以下であればそのクラスタに分類，閾値以下のクラスタがなれけば新しいクラスタを作る，という手法が適している．この手法であればクラスタ数 K が一回で決まるので，K-means 法のように何回もクラスタリングをする必要がない．よって本稿では以下の手法でクラスタリングをする．

(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測定
(2) 距離が閾値以内かつ最も距離の短いクラスタに分類
(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) 要素が加わったクラスタの中心を再計算

以上の方法でセグメントの基本ブロック・ベクトルをクラスタリングした．

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，そのクラスタに含まれるユニット数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 を推定する．

5. 提案手法の実装

5.1 BBTracker を利用した基本ブロック列の生成

提案手法では基本ブロック列に対してユニットを生成するため，まず基本ブロック列を生成する必要がある．ここでは **BBTracker** の出力に手を加えることで基本ブロック列を生成した．以下 BBTracker の仕様と，その出力からどうやって基本ブロック列を生成するかを説明する．

BBTracker は SimPoint の基本ブロック・ベクトルを生成するために使われるツールである．BBTracker は，それぞれの命令が何番の基本ブロックに属するかを解析しながら，インターバル値を超える命令数を読み込む．インターバル値を超えた場合，現在読み込んでいる命令の属する基本ブロック内の最後の命令が，そのインターバルの切れ目となる．次のインターバルの大きさは，「指定したインターバル値 - 前のインターバルの端数 + 今回のインターバルの端数」となる．この端数により，各インターバルはそれぞれ長さの違うものになってしまう．

また，BBTracker は SimPoint 用の基本ブロック・ベクトルを生成するため，各要素は「基本ブロックが出現した回数 × その基本ブロックに含まれる命令数」となっている．

例えば，図 4 のようなループがあるとする．このときインターバルの値を 6 命令として BBTracker を使用すると，命令 1 から始まるインターバルの長さは端数の影響で，3+2+3=8 命令となってしまう．またこの基本ブロック・ベクトルは，$\{0, 0, 2 \times 3, 1 \times 2, 0, ...\} = \{0, 0, 6, 2, 0, ...\}$ となる．

BBTracker を利用して基本ブロック列を生成するために，まずインターバルの値を 1 命令として BBTracker を動かす．すると各インターバルでは，基本ブロックが 1 つしか出現しないことになるため，各基本ブロック・ベクトルには値が 1 つしか入らない．図 4 の例では，命令 1 から始める場合，各基本ブロック・ベクトルは $\{0, 0, 3, 0, 0, ...\}, \{0, 0, 0, 2, 0, ...\}, \{0, 0, 3, 0, 0, ...\}, ...$ となる．

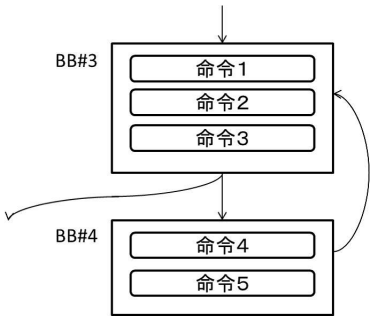

図 4 BBTracker

ここで，各基本ブロック・ベクトルから値の入っている基本ブロックを挙げていくと基本ブロック列が得られることがわかる．このような方法で，今回は基本ブロック列を生成した．

提案手法を BBTracker を用いて実装することで，SimPoint が使っている K-means 法のプログラムを，生成したセグメントに対して実行できるようになった．これにより K-means 法と提案手法のクラスタリング法を比較することができる．

5.2 ユニットの生成方法

以上の方法で基本ブロック列を生成し，それをユニット長で区切ることでユニットが生成できる．しかし 1 つの基本ブロックにおよそ 10 命令が含まれることを考えると，BBTracker のインターバルの値を「ユニット長 ×10」に指定することで，基本ブロック列を生成せずに命令列から直接ユニットを生成する方法も考えられる．以下，この方法について考察する．

5.1 節で述べたように，BBTracker が生成する各基本ブロック・ベクトルの大きさは端数の影響により互いに異なっている．この端数の影響は，インターバルの値が小さくなるほど大きくなる．そのためユニットのような小さな単位を用いる場合，端数による大きさのばらつきが適切なシミュレーション・ポイントを選択するときに悪影響を及ぼす可能性が考えられる．

以上を踏まえて，今回は基本ブロック列を生成してからユニットに分割する方法を選択した．

5.3 鬼斬弐への実装

従来は BBTracker がプロセッサ・シミュレータ SimpleScalar に実装されていたのに対し，今回はプロセッサ・シミュレータ「鬼斬弐」[3] に実装した．

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

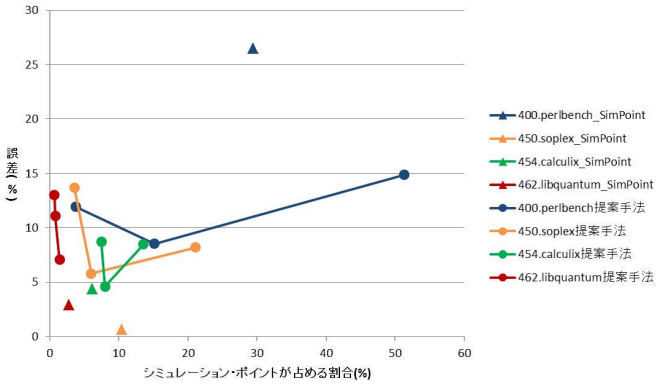

図 5　SPECCPU2006 シミュレーション・ポイントの割合と誤差

6. 評価

6.1 評価モデル

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・ポイントと提案手法の評価を行った．鬼斬弐では SPECCPU2006 を実行できるため，今回は 400.perlbench，450.soplex，454.calculix，462.libquantum の 4 本のベンチマーク・プログラムに対して評価を行った．入力データ・セットには *test* を用いた．評価したプロセッサの基本的なパラメータは**表 1** の通りである．

シミュレーション・ポイントのインターバルの値を 1M に，提案手法のユニット長を 100，セグメント生成時のマンハッタン距離の閾値を 100 に，クラスタリング時のマンハッタン距離の閾値を 10，50，100 に設定した．

6.2 シミュレーション・ポイントが占める割合と推定 CPI の誤差

図 5 にシミュレーション・ポイントが占める割合と推定 CPI の誤差の関係を表したグラフを示す．シミュレーション・ポイントが占める割合が小さいほど実行すべき命令数が少なくなるため，グラフの左側ほど実行時間が短いことを表している．各ベンチマーク・プログラムの全命令数は，400.perlbench で約 17M，450.soplex で約 67M，454.calculix で約 245M，462.libquantum で約 288M 命令であった．

提案手法において，クラスタリングの閾値を上げるほどシミュレーション・ポイントが占める割合が小さくなった．これは，閾値が大きいほど生成されるクラスタ数が少なくなり，シミュレーション・ポイントの数が減るためだと考えられる．400.perlbench，450.soplex，454.calculix は，閾値 50 のとき，462.libquantum は閾値 10 のとき最も誤差が小さくなった．閾値が 10 のときシミュレーション・ポイントが占める割合が最大になるが，それが誤差の低下には直接つながらないことがわかる．

400.perlbench，450.soplex，462.libquantum では提案手法の方がシミュレーション・ポイントよりもシミュレーション・ポイントが占める割合が小さくなっている．誤差については，450.soplex，454.calculix，462.libquantum において提案手法よりもシミュレーション・ポイントの方が小さくなっている．

6.3 セグメント内のユニット数

提案手法のセグメントに含まれる平均ユニット数は，400.perlbench で約 2 個，450.soplex で約 3 個，454.calculix で約 4 個，462.libquantum で約 46 個であった．

400.perlbench，450.soplex，454.calculix では平均ユニット数が非常に少ない．これはこの 3 つのベンチマーク・プログラムの各フェーズが非常に短いか，提案手法で正しくフェーズ検出ができていないことを意味する．各フェーズが短い場合，フェーズの切れ目を含むセグメントが推定 CPI に悪影響を与えることが考えられる．

7. おわりに

従来のフェーズ検出手法では固定長のインターバルを用いることによりフェーズ検出において誤差が生じていた．本稿ではセグメントという新しい概念を導入することにより新しいフェーズ検出手法を提案した．結果，400.perlbench，450.soplex，462.libquantum でシミュレーション・ポイントが占める割合が SimPoint より小さくなり，400.perlbench で誤差が小さくなった．しかし 450.soplex，454.calculix，462.libquantum では SimPoint よりも誤差が大きくなっている．

区間 CPI の測定の方法も課題として挙げられる．問題は，シミュレーション・ポイントが短い場合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 CPI が得られない．SimPoint のようにシミュレーション・ポイントが長い場合は，シミュレーション・ポイントまで命令をスキップし（エミュレーションで行い），シミュレーション・ポイントのみをシミュレーションすることで少ない誤差で区間 CPI の予測が可能である．しかしこの測定方法ではエミュレーションでスキップしている部分において

キャッシュヒット・ミス判定や分岐予測を行っていないので，シミュレーション・ポイントの初めの部分ではCPIの値が実際の値より高くなってしまう．シミュレーション・ポイントが数M命令程度あればこの影響は少ないが，提案手法のように各シミュレーション・ポイントの長さが短いと影響が大きくなり，誤差の原因となると考えられる．

よってキャッシュ・ヒット率，分岐予測ヒット率なども考慮してフェーズを検出する必要がある．

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] 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 SACSIS, pp. 120–121 (2009).
[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] Dhodapkar, A. S. and Smith, J. E.: Managing Multi-Configuration Hardware via Dynamic Working Set Analysis, *MICRO*, pp. 84–93 (2003).
[7] 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, *RECONF* (2005).
[8] Perelman, E., Hamerly, G., Biesbrouck, M. V., Sherwood, T. and Calder, B.: Using SimPoint for Accurate and Efficient Simulation, *SIGMETRICS* (2003).
[9] 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).
[10] Hamerly, G. and Elkan, C.: Learning the k in k-means, Technical Report CS2002-0716, University of California (2002).
[11] Hamerly, G., Perelman, E. and Calder, B.: How to Use SimPoint to Pick Simulation Points, *ACM SIGMETRICS Performance Evaluation Review* (2004).