正誤表

1 ページ目タイトル

(誤)　既存アーキテクチャのシミュレーション結果を用いる
　　汎用シミュレーション・ポイント検出手法

(正)　帰納的なシミュレーション・ポイント選出手法の改良

Vol.2014-ARC-212 No.20
2014/10/7

既存アーキテクチャのシミュレーション結果を用いる汎用シミュレーション・ポイント検出手法

福田　隆[1]　倉田 成己[1]　五島 正裕[2]　坂井 修一[1]

概要：プロセッサのシミュレーションには極めて長い時間がかかるが、シミュレーション・ポイントを選出することでこれを実効的に短くすることができる。本研究室では基準アーキテクチャによるプログラムのシミュレーション結果からフェーズを検出し、汎用的なシミュレーション・ポイントを選出する手法を提案している。先行手法では、プログラムを固定長インターバルに分割していたのに対し、本研究ではフェーズの切れ目に即した可変長のセグメントにプログラムを分割して、フェーズ検出を行う手法を提案する。さらに、予備評価として SPEC2006 で test 入力における IPC 推定を行った結果を示す.

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性能評価が欠かせない．ところがシミュレーションには，極めて長い時間がかかるという問題がある．この問題の原因は，次の 3 つに分けられる．

まず第 1 に，シミュレータの実行速度の遅さがあげられる．

シミュレータの実行速度

プロセッサのシミュレーションによる性能評価では，キャッシュ・ヒット率や分岐予測ヒット率などの個々の指標に加えて，総合的な速度指標である **CPI** (Cycles Per Instruction) を得る必要があることが多い．そのために，サイクルごとの振る舞いを正確に再現するシミュレータは，**cycle-accurate** なシミュレータと呼ばれる．

実機による，いわゆる native 実行の速度に対して，エミュレータやシミュレータによる実行速度の低下の比を Speed-Down (**SD**) と呼ぶ．エミュレータの SD は 10 程度であるが，cycle-accurate なシミュレータの SD は 1,000〜10,000 程度にもなる．SD が 1,000 としても，実機で 10 分かかるプログラムには，10,000 分，すなわち，7 日以上かかる計算になる．

ベンチマークの命令数

第 2 に，評価に用いるベンチマークの命令数が非常に多いことがあげられる．例えば，プロセッサの評価に用いられる代表的なベンチマークである SPEC 2006 の場合，長いプログラムでは百 T 命令程度にもなる**?**．このプログラムを cycle-accurate なシミュレータで実行するには，年単位の時間がかかる．

第 3 に，性能評価のためには，何十通りもパラメータを変えてシミュレーションを行う必要がある．

同様の問題への対処法としてはまず実行するプログラム —— シミュレータの高速化が考えられるが，この場合には根本的な解決にはならない．たとえ数倍程度の高速化が可能であったとしても，年単位が月単位に削減されるだけで，評価に用いるには依然として非現実的であるからである．

そのため実際には，ベンチマークのごく一部のみをシミュレートする方法がとられる．典型的には，各ベンチマーク・プログラムのごく一部，例えば，プログラムの最初の 1G 命令をエミュレーションによってスキップし，その後の 100M 命令のみをシミュレートする．1G 命令をスキップするのは，初期化部分を評価に含めないためである．ベンチマークには様々な特徴を持つプログラムが選定されているはずであるが，各プログラムの初期化部分ばかりを評価したのでは，その意図を蔑ろにすることになる．

このような方法は，一般に認められてはいるものの，実行した部分がベンチマーク・プログラムの特徴を確かに反映したものであるかどうか疑問が残る．実際，SPEC 2006 の astar, mcf, omnetpp などでは，1G 命令では初期化部分をスキップするには不十分であることが分かっている．

シミュレーション・ポイントとフェーズ検出

このような問題に対処するために，**シミュレーション・ポイント**を選定することが考えられる．シミュレーショ

[1] 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
[2] 国立情報学研究所
National Institute of Informatics

ン・ポイントとは，そこだけを実行することによってプログラム全長にわたるプロセッサの振る舞いが推定できるようなプログラムの動的な一部のことである．例えば，前出の「1G 命令をスキップした後の 100M 命令」は（単純すぎる）シミュレーション・ポイントの一種と考えることができる．

よりよいシミュレーション・ポイントを選定するには，実行の**フェーズ** [7][8] の考え方を用いればよい．プログラムの繰り返し構造に起因して，プログラムの動的な区間にはプロセッサが同様の振る舞いを示すものが多くある．区間のうち，同様な振る舞いを示す区間は同じフェーズ，異なる振る舞いを示す区間は異なるフェーズと呼ぶことができる．このような考え方に基づけば，プログラムの実行の全区間を，例えば数十〜数百種類のフェーズに分類し，同じフェーズに属する区間から 1 つずつを選んでシミュレーション・ポイントとすればよい．

SimPoint

フェーズ検出に基づいてシミュレーション・ポイントを選定する代表的な手法として **SimPoint** が挙げられる [4]．SimPoint をはじめとするほとんどの手法の特徴は，ISA の情報のみを用い，マイクロアーキテクチャの情報を用いないことにある．すなわち，エミュレーションによって得られたプログラム・カウンタの系列のみを基に，フェーズ検出（とシミュレーション・ポイント選定）を行うのである．

SimPoint は，プログラムの全長を固定長の**インターバル**に分割し，それぞれのインターバルに含まれるプログラム・カウンタ (正確には基本ブロック．2 章で詳述する．) の種類を基に，k-means 法によってクラスタリングを行う．[3] 各クラスタに属するインターバルが同じフェーズとみなされる．これは，以下の仮定に基づく；すなわち，プログラムの同じような（静的）部分を実行している（動的）区間は同じフェーズである．

しかしこの仮定には明らかな反例がある．同じコードであっても，入力が異なれば，プロセッサが同じ振る舞いを示すとは限らない．典型的には，処理されるデータの量が異なれば，キャッシュ・ヒット率は大きく異なり，CPI にも大きな影響を及ぼす．実際，SPEC 2006 の一部のベンチマークでは，同じ部分が異なるサイズのデータに対して繰り返し実行されており，SimPoint の予測精度を大きく悪化させている．

帰納的なシミュレーション・ポイントの選出手法

しかし，既存の手法が ISA の情報のみを用い，マイクロアーキテクチャの情報を用いないことは，ある意味 当然である．特定のマイクロアーキテクチャの情報を用いてシミュレーション・ポイントを選定したとして，それを別のマイクロアーキテクチャの評価に用いられる保証がないからである．

そこで本研究室の先行手法では，特徴的なマイクロアーキテクチャを持ついくつかのプロセッサ（これを基準アーキテクチャと呼ぶ）によってプログラムのシミュレーションを行うことで，その結果から汎用的なシミュレーション・ポイントを選定する．

図 1 サーキットのたとえ話

この手法はサーキットのたとえ話を用いるとわかりやすい．ある長大なサーキットを走る車のタイムを，サーキットの全長を走行することなく求めるにはどうしたらよいかを考える．まず，あらかじめ，色々なタイプの移動手段を用意してサーキットの全長を走行させる．サーキットはスタートからゴールまでを R1 から R7 の区間で区切られており，それぞれの区間での速度が図 1 のように計測されたとする．この図において，区間ごとの速度を比べることにより，R1 と R6，R4 と R5 がそれぞれ同じ「フェーズ」の区間であることが分かる．

同様に基準アーキテクチャによるシミュレーションの結果を用いて区間をフェーズに分類すれば，マイクロアーキテクチャの情報を含んだフェーズを検出することができる．

提案手法

SimPoint および本研究室の先行手法ではプログラムの動的命令列を固定長のインターバルに区切っている．そのため，フェーズの切れ目を含むインターバルが生じてしまう．このようなインターバルは，前後のフェーズとは関係のないクラスタに分類されてしまう可能性があり．クラスタリングの精度を悪化させてしまう．そこで，提案手法では本研究室の先行手法を改良し，可変長の区間（これを**セグメント**と呼ぶ）を用いてフェーズ検出を行う．提案手法では，基準アーキテクチャの IPC の遷移に対して一次元のメディアンフィルタをかけることでフェーズを顕在化させる．フィルタをかけた IPC の遷移において，IPC の値が大きく変化している部分をフェーズの切れ目とみなし，セグメントの区切れとする．このようにして得られたセグメントは精度よくクラスタリングすることができる．

本稿の構成

本稿の構成は以下のとおりである．まず 2 章ではフェーズ検出の代表的な手法である SimPoint の紹介と問題点について述べる．3 章で本研究室の選考手法である帰納的

なシミュレーション・ポイントの選出手法ついて述べる．5 章では先行手法を改良した提案手法について述べ，6 章でその評価を行う．最後に 7 章でまとめと今後の課題を述べる．

2. SimPoint

本章ではフェーズ検出の代表的な手法である SimPoint[5] について説明する．SimPoint は PC の振る舞いに着目してフェーズを検出する手法である．本章ではまず SimPoint で用いる特徴量である基本ブロックベクトルについて説明し，その後フェーズ検出の手順，問題点を述べる．

2.1 SimPoint の原理

SimPoint は以下の仮定に従ってフェーズ検出を行う；すなわち「プログラムの同じような（静的）部分を実行している（動的）区間は同じフェーズである」という仮定である．これに則り，SimPoint ではプログラムの動的命令列を一定の長さの区間に区切り，それぞれの区間がプログラムのどの部分を含んでいるのかを調べる．この，「プログラムの部分」の単位となるのが基本ブロックである．この動的命令列を固定長で分割した区間のことを **interval** と呼ぶ．SimPoint ではインターバルに含まれる基本ブロックの種類・頻度から特徴量（後述の基本ブロックベクトル）を定義し，これを比較することでインターバルをフェーズに分類する．

2.2 基本ブロック

基本ブロックとは分岐や合流を含まない命令のまとまりである．SimPoint は PC の振る舞いをもとにフェーズを検出する．一般に PC 上でフェーズの変わり目となるのは分岐や合流が発生する命令である．よって，分岐と合流のない命令についてはひとまとめに扱ってよいため，PC 列を基本ブロック (basic blok) 列にまとめてしまう．基本ブロック列は PC 列と同じ情報量を持っているがデータサイズは小さくなっており，扱いやすい．

2.3 基本ブロックベクトル

SimPoint では節で述べた，フェーズの性質を示す特徴量として**基本ブロックベクトル**を用いる．基本ブロックベクトルはインターバルごとに定義される．その要素数はプログラム全体に含まれる基本ブロックベクトルの種類数に等しい．従って基本ブロックベクトルは一般に高次元のベクトルとなる．基本ブロックベクトルの要素は，その区間における，それぞれの基本ブロックの出現回数が入る．例えば，あるプログラムがあり，このプログラムの全（静的）命令が 3 個の基本ブロックにまとまることがわかったとする．(これは非常に短いプログラムの例である．）この場合，インターバルに対して定義される基本ブロックベクトルは要素数が 3 となる．あるインターバルの基本ブロックベクトルが（0 3 1) ならば，このインターバルの 1 番目の基本ブロックを 0 個，2 番目の基本ブロックを 3 個，3 番目の基本ブロックを 1 個含むことになる．

2.4 フェーズの検出方法およびクラスタリング

まず，プログラムを，事前にエミュレータで実行して動的命令列を得る．動的命令列を固定長で分割してインターバルを得る．通常分割するサイズは 1M から 100M 程度である．SimPoint ではそれぞれのインターバルに対して基本ブロックベクトルを求める．こうして得られたベーシックブロックを k-means 法を用いてクラスタリングする．同一のクラスタに分類されたベーシックブロックベクトルを持つインターバル同士は同じフェーズであるといえる．得られた結果から，それぞれのクラスタで代表的なインターバルを取り出し，これをシミュレーション・ポイントとする．このシミュレーション・ポイントから先ほど述べたようにプログラム全体を実行したときの評価指標を推定することができる．

2.5 SimPoint の問題点

SimPoint はプログラムカウンタの振る舞いの似たような部分同士を同じフェーズとして検出する手法である，しかし，1 章でも述べたが SimPoint の「プログラムの同じような（静的）部分を実行している（動的）区間は同じフェーズである」という仮定には，反例がある．例えば同じコードでも入力が異なると処理するデータ量に応じてキャッシュヒット率が変化し，CPI に大きな影響を及ぼす．すなわち，プログラムの同じ部分を実行していてもプロセッサの状態によってプロセッサの性能は変化してしまう．SimPoint ではこのために，一部のベンチマークの予測精度が大きく悪化している．

3. 帰納的なシミュレーションポイント選出手法

SimPoint はプログラムカウンタの振る舞いに着目したフェーズ検出の手法であった．それに対し，本手法ではあらかじめ，特徴的なアーキテクチャでプログラムをシミュレーションして得られた区間 CPI をもとにフェーズを検出する．

3.1 原理

n 種のアーキテクチャ $a_i\,(i = 1, 2, \ldots, n)$ で，2 つの区間 s，t をシミュレートした結果，$2n$ 個の CPI 値 $c_1(s), c_1(t), c_2(s), c_2(t), \ldots, c_n(s), c_n(t)$ を得，これらに対して $c_1(s) \simeq c_1(t)$，$c_2(s) \simeq c_2(t), \ldots, c_n(s) \simeq c_n(t)$ が成り立つとする．アーキテクチャ a_i が互いに十分異なっていれば，区間 s と t は同じフェーズで，未知のアーキテク

チャ a_{n+1} に対しても $c_{n+1}(s) \simeq c_{n+1}(t)$ となることが期待できる．

3.2 事前シミュレーションするアーキテクチャの選定

1 章の車のたとえ話からもわかる通り，事前にシミュレーションするアーキテクチャは互いに十分異なっているのが望ましい．ここで，どのようなアーキテクチャを選べばよいか考察する．現代のスーパスカラプロセッサにおける IPC 低下の主な要因はキャッシュミス，分岐予測ミス，命令の依存関係である．プロセッサの研究・開発ではこれらの影響をどのように減らして性能を向上（あるいは維持）させることに主眼が置かれている．そこで，本提案手法ではこれらの IPC 低下要因の影響を受けないアーキテクチャを，事前シミュレーションするものに含め，これらの低下要因をフェーズに反映させる．具体的には，スーパスカラを基本構成として，これに以下の 3 つの構成を加えた合計 4 つのアーキテクチャで事前シミュレーションを行う．

- キャッシュのヒット率を 100%にしたもの
- 分岐予測のヒット率を 100%にしたもの
- 命令の発行幅を 1 にしたもの

以上 4 つのアーキテクチャで区間ごとの IPC を計測する．

3.3 手法の工程

本節では手法の工程について説明する．本手法の流れは

(1) 実行対象のプログラムの動的命令列を区間に分割
(2) 複数の特徴的なアーキテクチャをもつプロセッサによる事前実行
(3) 区間ごとの IPC ベクトルの生成およびクラスタリング

3.3.1 事前シミュレーション

次に，複数の特徴的なアーキテクチャを持つプロセッサで，プログラムを全長にわたってシミュレーションし，区間ごとのの IPC を計測する．この際，区間は SimPoint と同様に一定命令数おきに区切ったインターバルを用いる．

3.4 IPC ベクトルの作成

次にそれぞれの区間でフェーズを検出する際の特徴量となる IPC ベクトルを作る．この CPI ベクトルの要素数は実際にシミュレーションしたアーキテクチャの数である．（今回は 4 である）また，要素はそれぞれのアーキテクチャのその区間における IPC である．例えば，アーキテクチャ a_1，a_2，a_3 でプログラムを全実行したとする．ある区間において，CPI が a_1 は 1．1，a_2 が 1．2，a_3 が 0．9 だとすると，この区間における IPC ベクトルは（1．1 1．2 0．9）と表すことができる．このようにしてプログラムの全区間において IPC ベクトルを生成する．二つの区間の IPC ベクトルの距離が近いということは，その区間は事前シミュレーションしたどのアーキテクチャでも性能差が少ないということである．すなわち二つの区間は同じフェーズであると考えることができる．

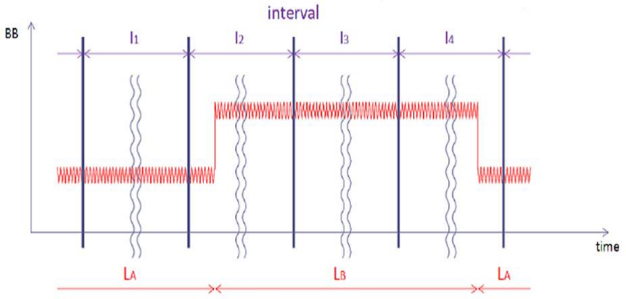

図 2 固定長インターバルの問題点

3.4.1 クラスタリング

次に得られた IPC ベクトルに対しクラスタリングを行う．すなわち，距離の近い IPC ベクトルを同じフェーズに，離れているものを別のフェーズに分類する．

今回の評価では k-means 法ではなく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．

ある IPC ベクトル，V に対して

(1) V 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，V を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．また，クラスタの数が大きくてしたがって，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．

3.5 固定長インターバルの問題点

本手法は，SimPoint 同様固定長のインターバルを用いている．インターバルは固定長であるため，その中に図 2 のようにフェーズの内分点が生じてしまう．このようなインターバルの IPC ベクトルはフェーズ A や B とは全く関係のないクラスタに分類されてしまう可能性がある．このようなインターバルが IPC ベクトルの分散を大きくしてしまい，クラスタリングの精度を悪化させる．また，フェーズの切れ目を正確に捉えるために，インターバルの命令長を短く取ると，IPC の細かな変動に対して，敏感になり，細かすぎるフェーズを検出してしまう．これらは，インターバルの命令長が一定のために生じる問題である．

4. 提案手法

先行手法では固定長のプログラムの全長を固定長のインターバルに区切ってクラスタリングを行った．これに対し提案手法では，フェーズの切れ目で区切ったセグメントを用いることで，IPC ベクトルのばらつき (図 3) を抑え，ク

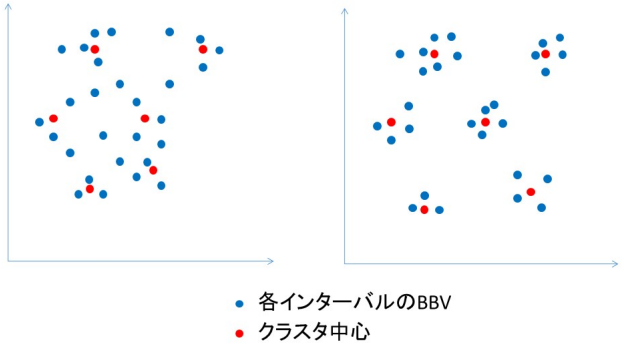

図 3　IPC ベクトルの分散

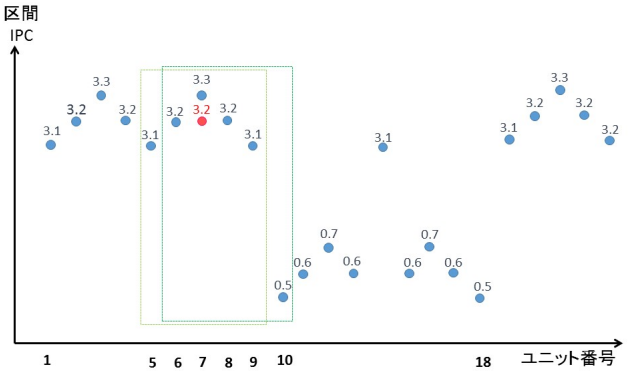

図 4　メディアンフィルタ前の様子

クラスタリングの精度の向上を目指す．

4.1　ユニットとセグメント

ユニットとは数百から数千命令程度の命令列である (評価では 1K 命令)．まずあらかじめ，基準アーキテクチャでシミュレーションを全長にわたって行いユニットの命令長おきに区間 IPC を求める。このようにしてユニットの IPC 列が得られる．このユニットを後述の方法で一個以上つなげたものがセグメントとなる．提案手法ではこのセグメントに対して，IPC ベクトルを求め，クラスタリングを行う．ここで，ユニット長は最も短いシミュレーションポイントの長さに等しい．

4.2　ユニット長の下限

ユニット長さが短ければ短いほど，フェーズの切れ目を正確に見つけることができる．一方で，スーパスカラプロセッサでは 1 サイクルで複数の命令が同時にリタイアするため，過不足なくユニット長分の IPC を計測することは困難である．例えば 4 ウェイのスーパスカラプロセッサではユニット長± 3 命令分のばらつきが生じてしまう．ユニット長を短くすると，これに伴ってそのユニット長に等しい短いシミュレーションポイントが出現する．ここで極端に短いシミュレーションポイントでは，先ほどのばらつきが IPC 推定の際の誤差に現れてしまう．したがって，ユニット長を極端に短くすることは好ましくない．**??**章の評価ではユニット長は 1 K 命令とした．

4.3　メディアンフィルタによるセグメントの生成

ユニットの区間 IPC 列に対してメディアンフィルタをかける．その様子を図 4,5 で説明する．図 4,5 はそれぞれ縦軸にユニットの区間 IPC をとり横軸はそのユニットが先頭から数えて何番目なのかを示している．図 4 は区間 IPC 列に対してフィルタがスライドしている状態を表している．

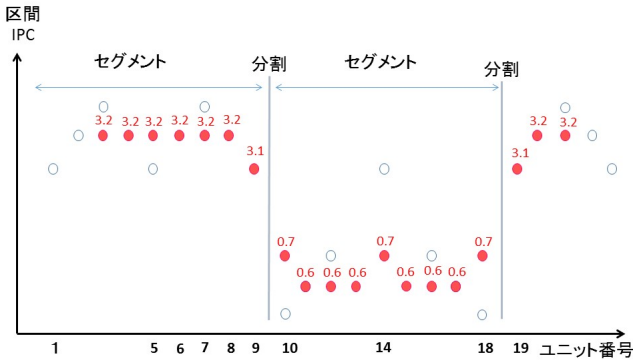

図 5　メディアンフィルタ後の様子

今回の例ではフィルタのサイズは 5，閾値を 0.5 とする．

図 4 のように緑色のフィルタがスライドさせることにより，上の青の IPC 列は下の赤の IPC 列に変換される．メディアンフィルタは前後 5 個の IPC の中央値をとっているため，IPC の小さな変動は除去され、大きな変動は保存される．これにより，フェーズを顕在化させることができる．

フィルタをかけて得られた IPC 列において，閾値以上 IPC が変化している部分をフェーズの切れ目とみなし，そこで区切ってセグメントを生成する．

4.4　クラスタリング

まず，各セグメントにおける基準アーキテクチャの区間 IPC を求める．そして，これを並べた IPC ベクトルを各セグメントに対して生成する．得られたベクトルを先行手法と同様の方法でクラスタリングする．この際，セグメントは長さが可変長であるため，クラスタの中心の計算や，シミュレーションポイントの重みづけの際には，セグメントの長さに応じた IPC ベクトルの重みづけを行う必要がある．

表 1　Configuration of Processor

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,15cycles,128B/line
main memory	200cycles

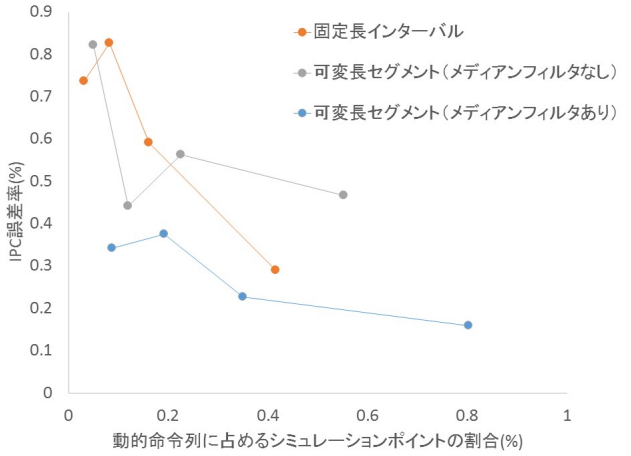

図 6　平均値の比較

5. 予備評価

本章では提案手法の評価について，現時点でデータの得られている SPEC2006 の 8 個のベンチマークについて予備評価を行った．得られた結果について，本研究室の先行手法である，固定長のインターバルを用いる手法と提案手法の比較を行った．

5.1 評価環境

今回の評価にはプロセッサ・シミュレータとして**鬼斬弐**を用いてシミュレーションを行った. 今回事前シミュレーションに用いたアーキテクチャ 4 つは先行手法，提案手法ともに以下のとおりである．

baseline　ベースとなるスーパスカラのアーキテクチャである．表 1 に詳細の構成を示す．

cache perfect　基本構成のキャッシュのヒット率を 100%に変更したもの

bpred perfect　基本構成の分岐予測器の予測率を 100%に変更したもの

fetch single　基本構成のフェッチ幅を 1 にしたもの

上記 4 つのアーキテクチャの事前シミュレーションからシミュレーション・ポイントを得た．このシミュレーション・ポイントを用いて IPC 推定を行ったアーキテクチャは以下での一種類である．

cache half　基本構成において L1 および L2 キャッシュの容量を半分にしたもの

尚, 評価対象アーキテクチャを全長シミュレーションして「正解」の IPC を得た結果，基本構成に対して，cache half で 10%程度の性能低下となった．

5.2 手法のパラメータについて

5.2.1 固定長インターバル

先行研究におけるパラメータである固定長のインターバルの命令長は 1K 命令とした．

5.3 可変長セグメント

提案手法におけるパラメータであるユニットの命令長は，インターバルの命令長と同じく 1K 命令とした．したがってセグメントの命令長の最小値は 1K 命令となる．そして，となり合う，ユニットの IPC が 0.1 より大きい場合にセグメントの区切りを入れることとした．また，セグメントのサイズが極端に大きくなると，そのセグメントがシミュレーションポイントとして選択された場合に，非常に長いシミュレーション・ポイントが生じてしまう．これを防ぐために，セグメントの長さの上限はユニット 10 個分とし，10 個を超えた時点でセグメントの分割を行った．よって，セグメントのサイズは 1K〜10K 命令となる．

5.3.1 メディアンフィルタ

提案手法で用いるメディアンフィルタのウインドウサイズは 5 とした．

5.4 評価結果

SPEC2006 のベンチマークに test 入力を与えた結果について，シミュレーション・ポイントを得た．得られたシミュレーション・ポイントをもとに，cache half アーキテクチャにおける，8 つのベンチマークの IPC 推定を行った．得られた結果について，本研究室の先行手法である固定長インターバルを用いた手法，および提案手法である可変長セグメントを用いた手法の比較を行った．

5.4.1 8 種類のベンチマークの平均値の比較

図 6 は横軸に動的命令列に占めるシミュレーションポイントの割合を，縦軸に IPC 推定の誤差率をとったグラフである．すなわち，このグラフ上では原点に近い点ほど，少ないシミュレーションポイントで正確な推定が行われていると考えることができる．このグラフ上に，8 種類のベンチマークのシミュレーション・ポイントの割合と IPC 推定

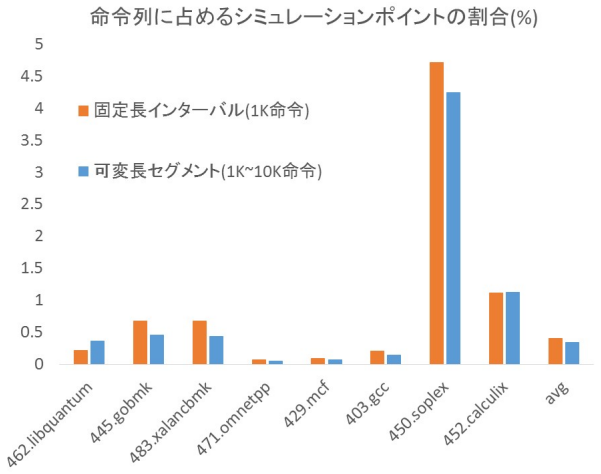

図 7 シミュレーション・ポイントが占める割合

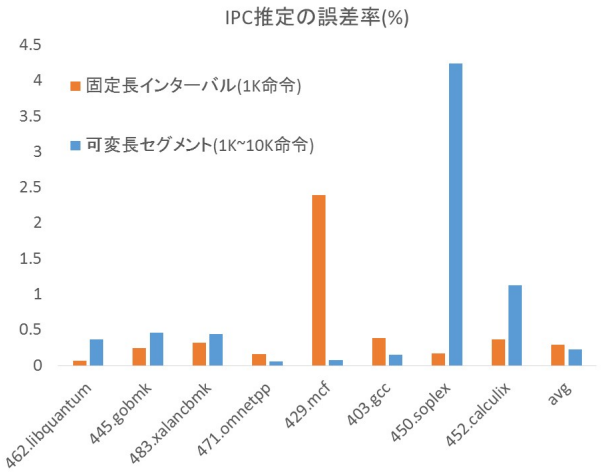

図 8 cache half の推定

の誤差率の平均値を，プロットした．なお，先行研究の固定長インターバルを用いた手法（橙色），ユニットの区間 IPC 列をメディアンフィルタを通さずに可変長セグメントをもとめたもの（灰色），提案手法通り，メディアンフィルタに通して可変長セグメントを求めたものの 3 種類をプロットした．これら 3 種類の手法について，クラスタリングの閾値を 0.1〜0.4 の間で変化させて得られた点を線で結んだ．グラフを見ると，青色の提案手法は橙色の先行手法よりも下にあり，提案手法の有効性が伺える．また，灰色と青色でも青色が下に来ており，ユニットの区間 IPC 列にメディアンフィルタをかけることで，クラスタリングの精度が向上していることがわかる．

5.4.2 ベンチマークごとの比較

図 6 において，先行手法（橙色）について，閾値が 0.1 のものを，提案手法 (青色) について閾値が 0.15 のものを今度は図 7，8 でベンチマークごとに詳しく見ていく．図 7 は動的実行命令にシミュレーション・ポイントが占める割合について示している．このシミュレーション・ポイントを用いて得られた IPC 推定の誤差率が図 8 である． 2 つの図はともに橙色が先行手法，青色が提案手法となっている．

ここで，471.omnetpp，429.mcf，403.gcc，の 3 つは，提案手法のほうが，先行手法よりも少ない命令数で，より正確な推定を行っている．逆に 462.libquantum については先行手法のほうが提案手法より，よい結果を示している．

全体では，平均して動的命令列が占めるシミュレーション・ポイントの割合は提案手法が 0.4%であるのに対し SimPoint は 0.35%である．すなわち提案手法のほうがシミュレーション・ポイントの割合が少ない．一方で IPC 推定の誤差率では先行手法が 0.29%に対して提案手法が 0.23%と誤差率が低い．すなわち，今回の予備評価の範囲では提案手法のほうがより少ないシミュレーション・ポイントで正確な推定を行っているといえる．

6. まとめと今後の課題

本稿では，帰納的なシミュレーション・ポイント検出手法に関して，可変長セグメントを適用することで，クラスタリングの精度を向上させる手法を提案した．さらに SPEC2006 のベンチマークのうち 8 種類について test 入力を用いて提案手法の予備評価結果を示した．

今回の評価では，SPEC2006 ベンチマークの test 入力の中でも短いもののみを用いて評価を行った．今後は，ref 入力についても評価を行っていく必要がある．

提案手法ではメディアンフィルタによってフェーズの顕在化を図った．しかし，メディアンフィルタがクラスタリングの向上にどの程度の影響を及ぼしているかを見るには，本発表では不十分であった．提案手法において，メディアンフィルタを用いた場合とそうでない場合で比較が必要である．

あわせて，ユニットサイズやメディアンフィルタのウインドウサイズ，クラスタリングの閾値は今回は，何通りかを試み，もっとも良い結果を示したものを掲載した．しかし，そのパラメータの動かし方は網羅的とは言えなかった．これらパラメータの適切な組み合わせについてはより検討が必要である．

謝辞 本研究の一部は，JST A-STEP「動的情報追跡による注入攻撃の普遍的な検出方式の実用化」による．

参考文献

[1] 早川薫, 倉田成己, 坂井修一, 坂井修一. 可変長セグメントを用いたフェーズ検出手法. SWoPP 2013
[2] 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. SACSIS, 2011.
[3] G. Hamerly and C. Elkan Learning the k in k-means CS2002-0716 University of California 2002
[4] Greg Hamerly, Erez Perelman, Jeremy Lau, Brad Calder, and Timothy Sherwood. Using machine learning to guide architecture simulation. Machine Learning Research,

2006.
[5] Greg Hamerly, rez Perelman, Jeremy Lau, and Brad Calde. SimPoint 3.0: Faster and More Flexible Program Analysis, 2005.
[6] Erez Perelman Greg Hamerly Brad Calder. Picking statistically valid and early simulation points. Parallel Architectures and Compilation Tech-niques (PACT), 2003
[7] Timothy Sherwood and Erez Perelman and Greg Hamerly and Suleyman Sair and Brad Calder Discovering and Exploiting Program Phases,ISCA,2002
[8] M. Hind, V. Rajan, and P. F. Sweeney. Phase detection: A problem classi
cation. Technical Report 22887, IBM Research, 2003.
[9] 塩谷亮太, 入江英嗣, 五島正裕, 坂井修一: 回路面積指向レジスタ・キャッシュ, SACSIS,2008