Vol.2014-ARC-211 No.12
2014/7/28

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短くできる。その検出には、従来の手法ではプロセッサの命令セットアーキテクチャの情報のみを用いていた。本研究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、他のあらゆるアーキテクチャに適用できるシミュレーション・ポイントを検出する手法を提案する。また、予備評価としてて 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

ⓒ 2014 Information Processing Society of Japan

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．

提案手法

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手法を提案する．

図 1　サーキットのたとえ話

サーキットのたとえ話

提案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．

例えばレーシングカーは直線が二度続いた場合と，下りの場合に速度が上がり、S 字と上りの場合には速度が下がっている．他の移動手段でもその特徴に応じて速度が上下している．

ここで R1 と R6，R4 と R5 はどの移動手段を使っても速度に差がないので，似たような区間であると考えることができる．今，未知の移動手段 X について，サーキットのタイムを推定するには, 図 1 でオレンジ色の丸で印をつけた区間のみを実際走行することでサーキットのタイムが推定できる．この印の区間はタイムを求めるのに走行すべき最低限の区間であり，ベンチマークで言うところの，シミュレーション・ポイントの役割を担っている．この方法では複数の移動手段の走行結果をもとに「走行すべき最低限の区間」を求める手法であるいえる．

この方法では，事前に用意する移動手段が，互いに十分異なる性質をもったものであることが重要である．上の例で，もしスクータとレーシングカーの結果のみから，「走行すべき最低限の区間」を求めた場合本来異なる性質を持つ区間である R7 と R4，R5 が同じ性質の区間に分類されてしまい，車 X についての推定の精度が下がってしまう可能性がある．

このようなサーキットのたとえ話と同様のことがベンチマークを用いたプロセッサの性能評価についてもいえる．複数の特徴的なマイクロアーキテクチャでベンチマークを

ⓒ 2014 Information Processing Society of Japan

全長実行した結果が得られれば，区間ごとのIPCを比較することで「シミュレーションすべき最低限の区間」すなわちシミュレーション・ポイントを得ることができる．

SimPointとの相違点

SimPointのPCに着目するフェーズ検出手法は，前節のサーキットのたとえ話をもちいると，区間の静的な性質からサーキットのフェーズを知る方法であると言える．すなわち，図の直線，くだり，S字といった，区間の静的な性質から走行すべき最低限の区間を知る方法であるといえる．しかし，この方法では，レーシングカーが異なるタイムを示しているはずの，R1とR2が同じ性質の道として分類されてしまい．Xについての推定の精度が下がってしまう可能性がある．これは区間の静的な性質が同じでも（レーシングカーが直線が二度続くと速くなるように）その区間に意移動手段が差し掛かったときの状態によってタイムが変化してしまうためである．提案手法は，実際の走行結果からサーキットのフェーズを知るため，このような区間にさしかかったときの移動手段の状態が考慮されている．

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章で本稿の提案手法について述べる．4章では提案手法の評価を行い，5章で今後の課題について述べる．

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，

ⓒ 2014 Information Processing Society of Japan

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. 提案手法

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，本提案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 提案手法の工程

本節では提案手法の工程について説明する．提案手法の流れは

(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 区間の切り分け方

本手法では，SimPoint と同様にシミュレーション・ポイントを抽出するプログラムを区間に分割する．区切る区間の長さは固定，あるいは本研究室の先行研究であるメディアンフィルタを用いるような可変の手法 [?] も考えられる．

3.3.2 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．

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（１．１１．２０．９）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．

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．

4. 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 14 個のベンチマークについて評価を行った．

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

4.1　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 CPI 推定を行ったアーキテクチャは以下の 3 つである．

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

pht singlebit　分岐予測器の PHT のカウンタビットを 1 にしたもの

norcs　本研究室で提案している非レイテンシ指向レジスタキャッシュシステム [9]. レジスタキャッシュ容量が十分である場合には性能低下がほとんどないので，今回は実験のため，敢えて RC 容量が極端に少ないモデルを採用している．

尚, 評価対象のこれらのアーキテクチャを 14 個のアーキテクチャに関して全長シミュレーションして「正解」CPI を得た結果基本構成に対して，cache half で 10%程度，PHT1bit で 3%程度，norcs で 15%程度の性能の低下となった．

4.2　全長実行の様子

ベンチマークのシミュレーション・ポイントを得る際に，

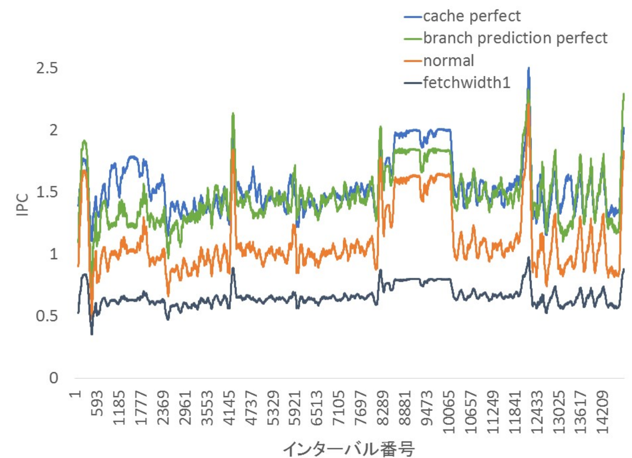

図 2　400.perlbench の全長実行の様子

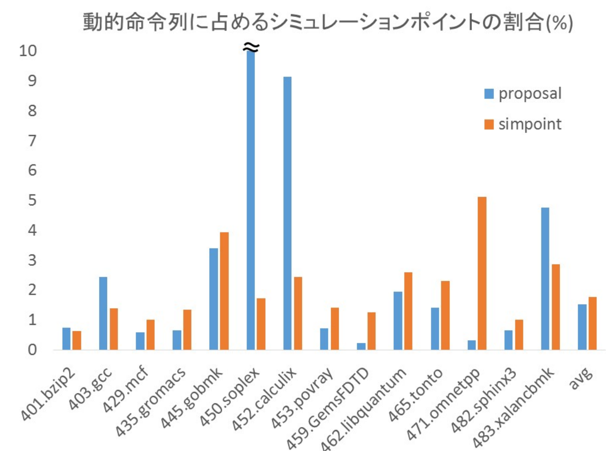

図 3　シミュレーション・ポイントが占める割合

前章で述べた 4 つのアーキテクチャで事前シミュレーションを行った．図 2 はその際に得られた区間 IPC の様子をプロットしたものである．このグラフからアーキテクチャが変わっても，フェーズが再現されていることが伺える．この図で橙色が基本構成のアーキテクチャであり，青色が基本構成でキャッシュヒット率を 100%（以下キャッシュパーフェクトと呼ぶ）にしたものである．橙色と青色の IPC の遷移を比べたときに，基本構成で見られる CPI の急な上昇が，キャッシュパーフェクトでは認められない箇所がいくつもある．両者の比較からこのような箇所はキャッシュヒット率が性能に大きく及ぼしている箇所であることがわかる．同様のことが分岐予測器パーフェクトやフェッチ幅＝1 のアーキテクチャと基本構成の比較においても言える．

4.3　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，先述の 3 つのアーキ

ⓒ 2014 Information Processing Society of Japan

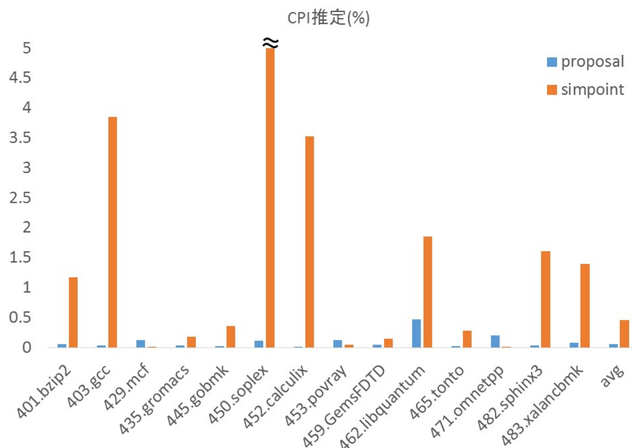

図 4　pht singlebit の推定

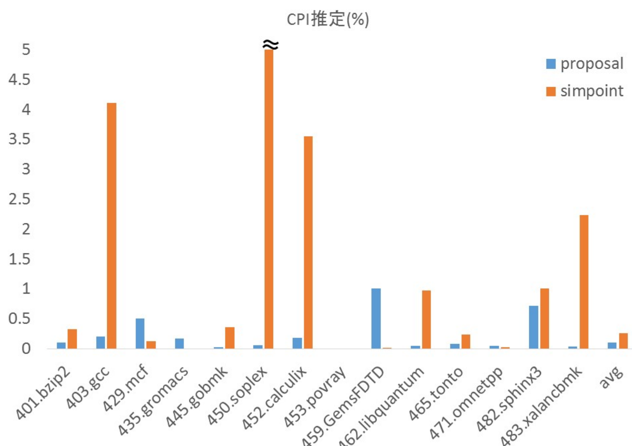

図 5　cache half の推定

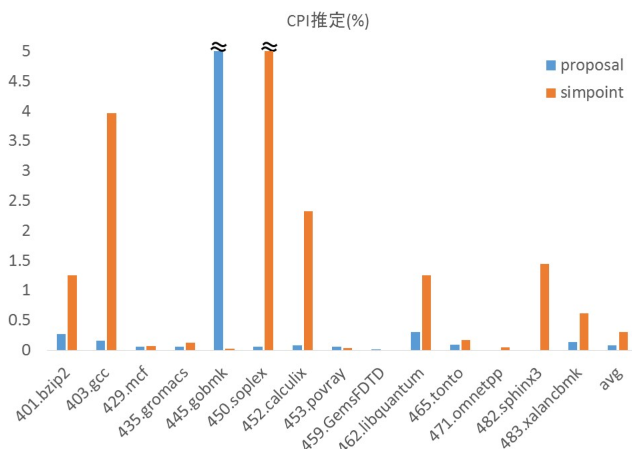

図 6　norcs の推定

テクチャの CPI 推定を行った．まず，SimPoint，提案手法ともにパラメータを固定して得られた結果を示す．

図 3 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・ポイントは SimPoint はインターバル 1M 命令，K-means における最大のクラスタ数を 300 に設定して得られたものである．一方提案手法は事前シミュレーションにおいて IPC を測定する間隔を 10K 命令にし (以下，これもインターバル 10K 命令と呼ぶ)，クラスタリングの際の閾値を 0.05 にして得られたものである．次にこのシミュレーション・ポイントを用いて得られた CPI の推定結果について図 4，図 5，図 6 に示す．

ここで 462.libquantum,465tonto,482.sphinx3, の 3 つは，SimPoint よりも少ない命令数で，どの評価対象のアーキテクチャでも，SimPoint より正確な推定を行っている．

また，450.soplex や 452.calculix は動的命令列の全長が 50 から 200M 命令程度と短く，インターバル 1 M の SimPoint でシミュレーション・ポイントが数個しか検出できなかった結果，誤差率が大きくなった．逆に提案手法では，この二つは誤差率は少ないかわりにシミュレーション・ポイントの割合が非常に大きくなっている (図 3)．

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 1.4%であるのに対し SimPoint は 1.8%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 CPI 推定の誤差率では PHT1bit は SimPoint が 0.4%に対して提案手法は 0.06%,cache half では SimPoint が 0.27%に対して提案手法が 0.11%,norcs では SimPoint が 0.3%に対して提案手法が 0.08%となっている．

4.4　445.gobmk について

445.gobmk の norcs でのシミュレーション推定では誤差率は 9%となった．norcs の gobmke の全長実行の様子を示す．図**??**は norcs および，事前シミュレーションを行ったアーキテクチャで gobmk をシミュレーションした際の IPC の遷移を部分的にグラフ化したものである．このグラフは一番上の黄線が norcs であり下 4 つが事前シミュレーションを行ったアーキテクチャの IPC の遷移である．このグラフから，事前シミュレーションを行ったアーキテクチャで矩形の IPC 遷移をしていたフェーズが norcs では別の形に変化していることがわかる．gobmk は全長に渡ってこの矩形が何度も出現している．

提案手法では，例えばインターバルが 10K の場合，この矩形一つが 1000 個程度の区間に分割されることになる．そして，IPC が同じ値を示す部分を同じフェーズに分類するため，矩形の平坦な部分はすべて一つのフェーズに分類されてしまう．このため，norcs で，矩形が別の形に変化すると，誤差が非常に大きく出てしまう．これを防ぐには，矩形一個をシミュレーション・ポイントとして選ぶ必要がある．

5.　まとめと今後の課題

本稿では，事前に複数の特徴的なアーキテクチャシで

ⓒ 2014 Information Processing Society of Japan

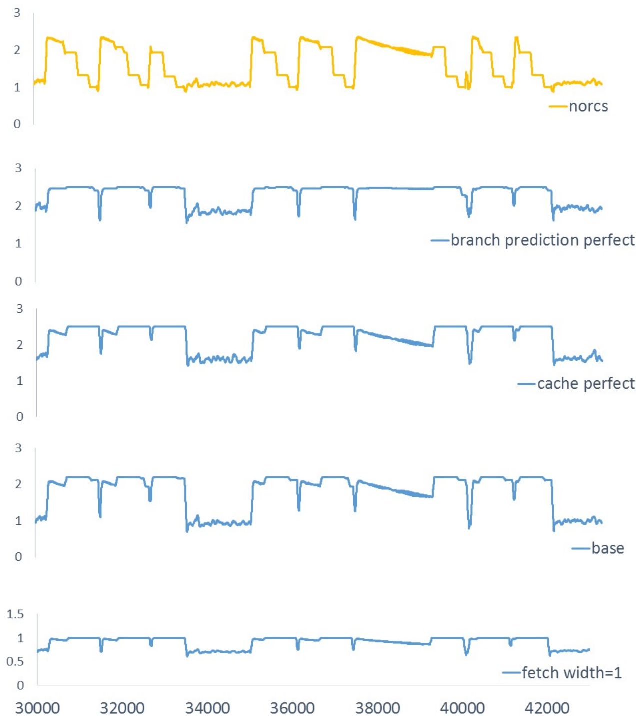

図 7 norcs と事前シミュレーションを行ったアーキテクチャでの 445.gobmk の IPC の遷移の比較

ミュレーションを行い，そのシミュレーション結果を用いてシミュレーション・ポイントを検出する手法を提案した．さらに SPEC2006 のベンチマークのうち 14 種類について test 入力を用いて提案手法の評価結果を示した．

今回の評価では，実際にシミュレーション・ポイントだけをシミュレーションしたわけではない．「正解」の IPC を得るために，評価対象の 3 つのアーキテクチャを全長実行した際に得た区間 CPI を，シミュレーション・ポイントの CPI として推定を行う際に用いた．実用の上ではシミュレーション・ポイントだけをシミュレーションすると，全長シミュレーションの場合と比較して，キャッシュや分岐予測機の温まり具合から，正確な CPI が得られない可能性がある．したがってそれらの影響についても検討を行う必要がある．

また，前節チューニングの結果からわかるとおり，それぞれのベンチマークで最適なインターバルやクラスタリングの閾値が異なっていた．現在は手動でパラメータを切り替えているが，ベンチマークに応じて適切なインターバルとクラスタリングの値を見つける方法についても検討が必要である．例えば，インターバルに関しては，ベンチマークのフェーズの長さのばらつきに対応する手法として，可変長にベンチマークを区切る手法が考えられる．すなわちベンチマークを区間に分ける際，長さ一定の区間に分けるのではなく，IPC の大きく変化する部分で区切る手法が考えられる．

さらに，本稿では SPEC2006 のベンチマークに test 入力を与えて評価を行ったが，今後は ref 入力ににについても評価も行っていく必要がある．

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

ⓒ 2014 Information Processing Society of Japan