Vol.2016-ARC-220 No.13
2016/5/10

集合的な帰納的シミュレーション・ポイント選出手法の提案

崔 ミン誠[1]　福田 隆[2]　神保 潮[3]　五島 正裕[3]　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 (SP) を選出することで短縮することができる．我々は，いくつかの特徴的なプロセッサのシミュレーションの結果から，他の一般のアーキテクチャにも適用可能な SP を選出する手法を提案している．従来は，個別のプログラムから個別のプログラムに対する SP を選出していた．本研究では，すべてのプログラムの実行をあたかも 1 つのプログラムの実行であるかのようにみなして，全プログラムから全プログラムに対する SP を選出する手法を提案する．SPEC 2006 の ref 入力における先頭 100G 命令の IPC 推定を行った結果を示す．

1. 初めに

現代のプロセッサの研究・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，プロセッサの性能評価には極めて長い時間がかかってしまうという問題点がある. 長いシミュレーション時間この問題の原因は以下の 2 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挙げられる. 実機による，いわゆるリアル (actual machine) 実行速度に対して，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. これは、エミュレータは，プログラムの命令セット・アーキテクチャ情報のみを用いる一方で，シミュレータはターゲットプロセッサのマイクロアーキテクチャの情報を用いてサイクルーバイーサイクルに動作を再現したためである.

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[13] の場合，長いプログラムでは百 T 命令程度にもなる**?**. このプログラムをシミュレータで実行するには，年単位の時間がかかる. また，このような問題を抱えながら性能評価のためには，何十通りもパラメータの組み合わせによるシミュレーションを行う必要がある.

2. 従来手法

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[4],[5],[7] が挙げられる． SimPoint[58] は、プログラムの同じ静的セクションを実行する命令の動的な間隔が同じ位相であることを前提としてターゲット・プロセッサのエミュレータ実行によって得られた命令の PC（プログラム・カウンタ）に基づいてフェーズ検出（シミュレーション・ポイント選定）を行う. k-means 法 [3] によってクラスタリングし、各クラスタに属するインターバル (プログラムの実行の全長を一定長の区間) が同じフェーズとみなされる．プログラムの IPC（または他のパフォーマンス指標）は、各クラスタ内の間隔の数によって重み付け選択したシミュレーション・ポイントのシミュレーション結果の平均値として推定する. これは，以下の仮定に基づく: 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，処理されるデータ量に応じて，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．そこで，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，その結果から帰納的なシミュレーション・ポイントを選定する手法が帰納的シミュレーション・ポイント選出する手法である.

[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] ソフトウェア& システム開発研究所
The Tokyo Software and Systems Development Laboratory, IBM Japan
[3] 国立情報学研究所
National Institute of Informatics

3. 帰納的シミュレーション・ポイント選出方法

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 IPC 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 帰納手法の工程

SimPoint は PC(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，帰納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 IPC をもとにフェーズを検出する．帰納手法の流れは以下の通りである．

3.2.1 シミュレーション対象のプログラムの動的命令列を区間に分割

先ず，シミュレーション対象プログラムをいくつかの特徴的なアーキテクチャで事前にシミュレーションする．このようなアーキテクチャを基準アーキテクチャと呼ぶ．基準アーキテクチャは互いに十分異なり，特徴的な IPC の振る舞いを示すものが望ましい．事前シミュレーションの際には，区間ごとの IPC を求める．帰納手法も SimPoint 同様に，シミュレーションするプログラムを区間に切り分ける必要がある．今回は固定長インターバル区間でプログラムを区切ることとする．

固定長で区切った場合，SimPoint の説明の際に触れた通り，フェーズの切れ目を含んだ区間生じ，クラスタリングの精度が下がってしまう可能性がある．しかし，帰納手法でインターバルから生成される IPC ベクトルは，SimPoint のベーシックブロックベクトルに比べ次元数は非常に低い．このため，SimPoint に比べ，インターバルを短くして区間数が増えても，後に述べるクラスタリングの高速化を行えば，クラスタリングすることができる．すなわち，帰納手法は，フェーズの切れ目の影響を無視できるほどに，細かくインターバルを取ることが可能である．

3.2.2 IPC ベクトル生成

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シミュレーション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．IPC ベクトルとは基準アーキテクチャのその区間での IPC を並べたベクトルである．この IPC 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．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 a1，a2，a3 でプログラムを全実行したとする．ある区間において，IPC が a1 は 1.1，a2 が 1.2，a3 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．

3.2.3 クラスタリング

上に述べたように得られた 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

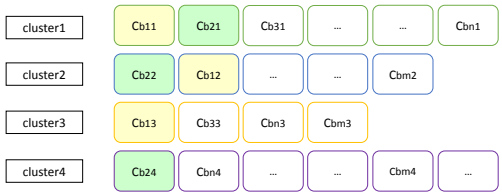
このクラスタリング方法では，閾値の大小によってクラスタ数が決まる．また，クラスタの数が大きくてしたがって，k-means 法のように何回も実行して最適なクラスタ数を求める必要はない．

4. 集合的な帰納的シミュレーション・ポイント選出方法

さらに、我々の研究室では、集合的な帰納的シミュレーション・ポイントの方法を提案する. 違うベンチマークプログラムでも IPC の振る舞いが近い区間がある. そういう利点を利用して全プログラムから全プログラムに対するシミュレーション・ポイントを選出することによって, より少ないシミュレーション・ポイントを選出するのが予想される.

一般的にプロセッサの性能評価はベンチマークプログラムごとに行う. したがって、従来は、個別のベンチマークプログラムから個別のベンチマークプログラムに対する シミュレーション・ポイントを選出した. つまり、SPEC2006 の中 22 種類（残り 7 種類データ待つ）のベンチマークの ref 入力先頭 100G 命令に対して一個ずつそれぞれのプロセッサ、基準アーキテクチャ 4 種を利用して、シミュレーション・ポイントを検出し、推定アーキテクチャ 4 種をシミュレーションし評価を行った.

本研究では，すべてのプログラムの実行をあたかも 1 つのプログラムの実行であるかのようにみなして，集全体のプログラムから集全体プログラムに対する シミュレーション・ポイントを選出して、それぞれベンチマークごとに評価を行う.



- $Error\ Rate = abs\left(\frac{1}{trueCPI} - \frac{1}{estimateCPI}\right)$
- $ComER = abs\left(1 - \frac{Cb11+Cb12+Cb13+Cb21+Cb22+Cb24}{Cb11*2+Cb22*2+Cb13+Cb24}\right) = abs(\frac{Cb11-Cb12-Cb21+Cb22}{Cb11*2+Cb22*2+Cb13+Cb24})$
- $B1ER = abs\left(1 - \frac{Cb11+Cb12+Cb13}{Cb11+Cb22+Cb13}\right) = abs\left(\frac{(Cb22-Cb12)}{Cb11+Cb22+Cb13}\right) = abs(\frac{Cb22-Cb12}{m})$
- $B2ER = abs\left(1 - \frac{Cb21+Cb22+Cb24}{Cb11+Cb22+Cb24}\right) = abs\left(\frac{Cb11-Cb21}{Cb11+Cb22+Cb24}\right) = abs(\frac{Cb11-Cb21}{n})$
- $ComER = \frac{m*B1ER+n*B2ER}{m+n}$

図 1　CollectiveEstimation error formula of colletive method.

図 2 は例として 2 つのベンチマークをくっつけてシミュレーション・ポイントを選出した時の誤差率の計算式を示したものである.

5. 評価

5.1 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。現代のスーパスカラプロセッサにおける IPC 変動の主な要因は以下の三つである.

(1) キャッシュミス
(2) 分岐予測ミス
(3) 命令の依存関係

プロセッサの研究・開発ではこれらの影響を減少させることによって、IPC を維持することに焦点が置かれている.

基準アーキテクチャ

基準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ので、今回はプロセッサの IPC 劣化の要因に基づいて基本モデルを選択した. 表 1 で, 三つの要因が基準アーキテクチャに与える影響をまとめた。

- superscalar　基本構成となる 4-way アウト・オブ・オーダーなスーパスカラプロセッサ
- scalar スカラプロセッサ
- 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ヒット率を 100%に変更したもの

表 2 は, 基本構成となるスーパスカラのアーキテクチャ詳細の構成を示す. なお，スカラプロセッサはごく初歩的なパイプラインプロセッサを想定し，ロード命令以外のすべての命令のレイテンシを 1 とし，分岐予測ミスの影響もないモデルを仮定した.

表 1　Degradation Factors of Basis Models

Basis Models	1) cache miss	2) bpred miss	3) inst. dep.
superscalar	O	O	O
scalar	O		
cache-perfect		O	O
bpred-perfect	O		O

推定対象アーキテクチャ

前述した基準アーキテクチャの事前シミュレーションから得たシミュレーション・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の 4 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 single bit 分岐予測器の PHT のカウンタビットを 1 にしたもの
- fetch eight 8-way のスーパスカラプロセッサ．power8 相当の演算器構成にしたもの
- regcache 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 [10].

これらのアーキテクチャのうち，最初の 3 つが，IPC の変動要因のフェーズを手法が反映できているか評価するためのアーキテクチャである．最後の regcache は，これらの要因が絡み合った実際の研究で提案されたアーキテクチャに対して帰納手法が適用可能か評価するためのものである．また、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，敢えて容量が極端に少ないモデルを採用している.

5.2 帰納的なシミュレーション・ポイント評価

図 2 はインターバルを変化させた際のシミュレーション・ポイントの割合と IPC 推定の誤差率を示したものである. このグラフは，合計 88 通り (ベンチマーク 22 種類×アーキテクチャ 4 種類）の組み合わせで推定を行った際の，幾何平均値をプロットしている．ここで、インターバル 10K を示す橙色が最も原点に近いところにあり，もっとも望ましいパラメータであることがわかる.

帰納手法の特徴の一つが閾値を変化させることにより，クラスタの数が変化し、シミュレーション・ポイントの割合が変わる。**図 3** は閾値を変化させた際のシミュレーショ

表 2　Configuration of superscalar Model.

ISA	Alpha w/ byte/word ext.
pipeline stages	Fetch:3, Rename:2, Dispatch:2, Issue:4
fetch width	4 inst.
issue width	Int:2, FP:2, Mem:2
inst. window	Int:32, FP:16, Mem:16
branch pred.	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

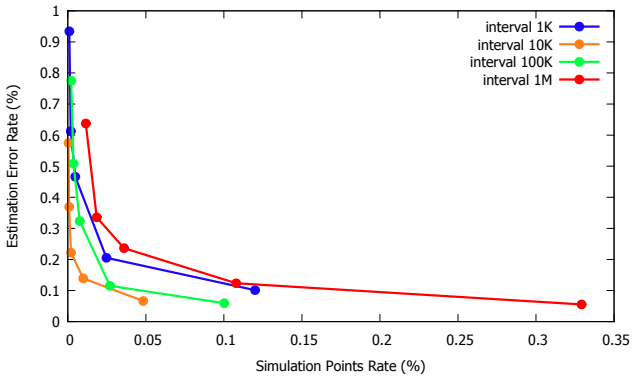


図 2　Estimation error against rate of simulation points with interval change.

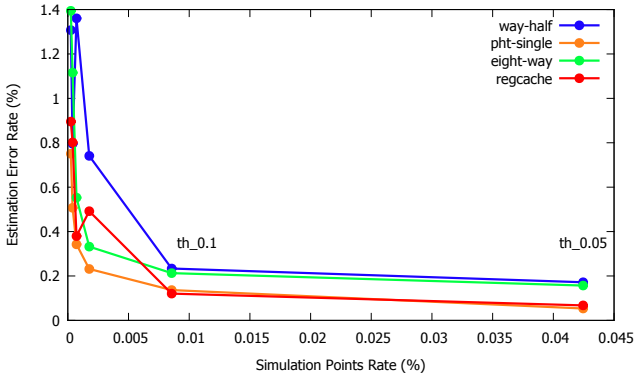


図 3　Estimation error against rate of simulation points with threshold change.

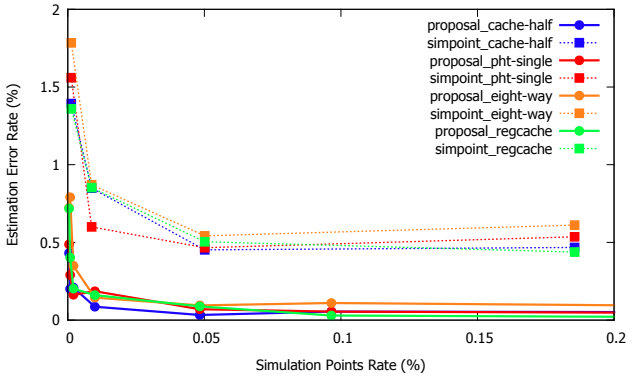


図 4　Estimation error against rate of simulation points of inductive method and SimPoint.

ン・ポイントの割合と IPC 推定の誤差率を示したものである．このグラフは，ここで、閾値 0.05-0.1 が飽和状態にあることを確認できる．

図 4 は推定アーキテクチャに対してパラメータを変化させた時の様子を示している．この図より，帰納的方法を示す実線が SimPoint を示す破線より，原点に近い位置にあり，帰納的方法が全体の平均で，優れたシミュレーション・ポイントを選択できていることがわかる．

bzip2 は帰納手法と SimPoint の比較の中では特に提案手法で良い結果が出たベンチマークの一つである．**図 5** と

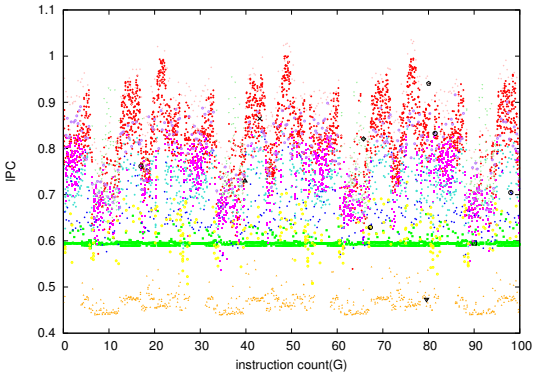


図 5　IPCs of clustered intervals of bzip2 on regcache SimPoint.

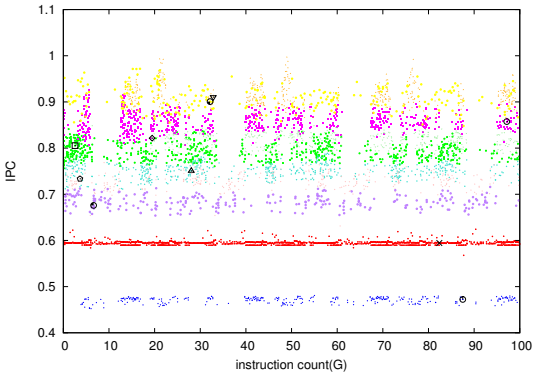


図 6　IPCs of clustered intervals of bzip2 on regcache inductive method.

図 6 は同じフェーズに分類したインターバルの実際の norcs でのシミュレーションの IPC を示している．図は点一つがインターバル 1 つ分に相当し，そのインターバルの実際の IPC を縦軸に，そのインターバルが何命令目に位置しているかを横軸に取っている．図において同じ色どうしのインターバルは同じフェーズであることを示している．また，黒い点はシミュレーション・ポイントとして選ばれた点を示している．（すべてのフェーズをグラフに乗せると煩雑になるため，大きなクラスタを形成した上位 10 フェーズのみをプロットしている．また，上下のグラフで色の対応関係はないことに留意されたい）．SimPoint のフェーズの分類では，黄色のフェーズや赤色のフェーズに分類された点の IPC に大きなばらつきがみられる．一方で帰納手法は SimPoint に比べ，同じフェーズに分類された区間同士の IPC がそろっており，きれいな帯が認められる．これは，帰納手法が SimPoint に比べ，IPC を推定する上で正確なフェーズの分類が行えていることを示している．

5.3　集合的な帰納的シミュレーション・ポイント評価

図 7 は pht single bit プロセッサに対して動的実行シミュレーション・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・ポインは事前シミュレーションにお

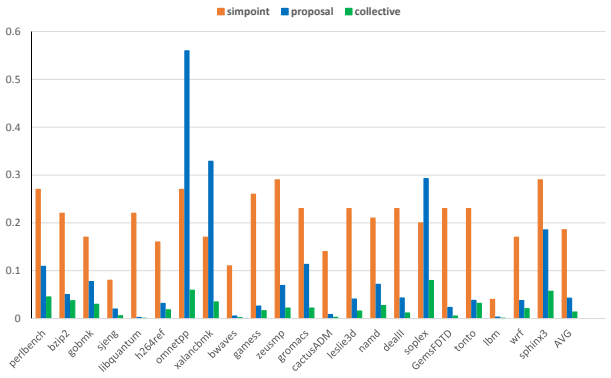


図 7　Simulation points rate of collective method, inductive method and SimPoint.

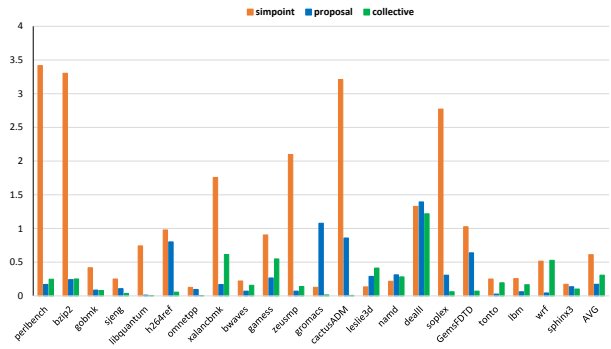


図 8　Estimation error rate of collective method, inductive method and SimPoint.

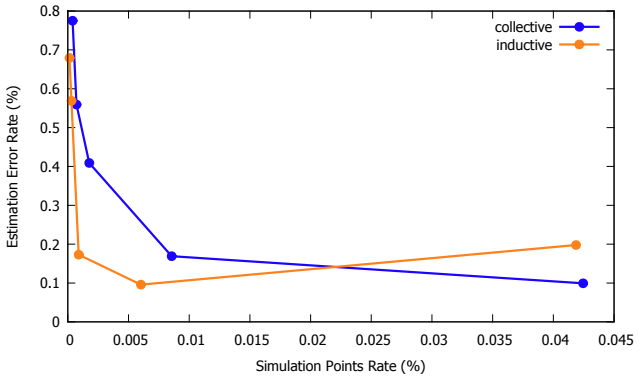


図 9　Estimation error against rate of simulation points of inductive method and collective method.

いて IPC を測定する間隔を 10K 命令にして, クラスタリングの際の閾値を 0.05 にして得られたのである。このシミュレーション・ポイントはをベンチマークプログラムごとに比較したグラフである。　ベンチマークプログラムをくっつけて共通フェーズのまとめによって個別のプログラムでシミュレーションしたときより削減したことがわかる.

一方で、図 **8** はシミュレーション・ポイントを用いて得られた IPC の推定結果についてベンチマークごとに示す. ここで, bzip2, libquantum, omnetpp, gromacs, cactusADM, GemsFDTD の 6 つは, 帰納的手法より少ない命令数で, 集合的手法がより正確な推定を行っている. また, soplex と sphinx3 はより多くの命令数なのに, より小さな誤差率を持つ. ところが, fetch eight 8-way のスーパスカラプロセッサでは, 集合的手法が 少ない命令数で, より正確な推定を行うベンチマークプログラムが bzip2, bwaves, soplex の 3 つだけである。

全体では, 図 **9** に示したように, 閾値によってシミュレーション・ポイントに対して, シミュレーション・ポイント割合と推定率が逆転するポイントがある. これに対してもっと検討する予定である.

6.　まとめと今後の課題

本稿では，くつかの特徴的な基準アーキテクチャのシミュレーション結果から帰納的にシミュレーション・ポイントを選択する手法を提案した．さらにベンチマーク全体を使って集合的な帰納的にシミュレーション・ポイントを推定する手法も提案した．また，SPEC2006 のベンチマークのシミュレーション・ポイントを選出し，IPC 推定を行って，実際のシミュレーションにより得られた IPC との誤差を評価した．評価の結果，帰納手法が SimPoint よりも，少ないシミュレーション・ポイントで正確な IPC の推定が行えることを確認した．しかし、我々の方法は、キャッシュ容量によるフェーズ切り目がまだ不十分である。したがって、我々は現在、わずかに異なる作業セットのサイズに対して異なる IPC が表示される巨大なレベルのキャッシュを基本モデルとしての評価を検討する。また、事前シミュレーションでは，ref 入力 100G 命令を鬼斬でサイクルアーキュレートにシミュレーションするのに 1 か月の期間を要した. プロセッサの特定のユニットだけを，シミュレーションし，その性能を測定することで，事前シミュレーショの計算コストを抑えることについても今後検討すべきである.

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] Greg Hamerly Erez Perelman Brad Calder.
How to Use SimPoint to Pick Simulation Points
[8] Timothy Sherwood and Erez Perelman and Greg Hamerly

and Suleyman Sair and Brad Calder
Discovering and Exploiting Program Phases,ISCA,2002
[9] M. Hind, V. Rajan, and P. F. Sweeney.
Phase detection: A problem classi
cation. Technical Report 22887, IBM Research, 2003.
[10] 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
[11] Jiaxin Li, Weihua Zhang, Haibo Chen, and Binyu Zang.
Multi-level phase analysis for sampling simulation. Technical report, 2013.
[12] Jeremy Lau, Erez Perelman, and Brad Calder.
Selecting software phase markers with code structure analysis. In Code Generation and Optimization(CGO), 2006.
[13] Arun A. Nair, Lizy K. John
Simulation Points for SPEC CPU 2006, 2008
[14] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsaf, James C. Hoe
SimFlex: Statistical Sampling of Computer System Simulation