Vol.2019-ARC-236 No.14
2019/6/12

RISC-V上で実行するSPEC CPU 2017のSimulation Point解析

中村 朋生[1,a)]　小泉 透[1,b)]　出川 祐也[1,c)]　稲岡 航大[1,d)]　有馬 裕一郎[1,e)]　塩谷 亮太[1,f)]
入江 英嗣[1,g)]　坂井 修一[1,h)]

概要：プロセッサの研究や開発においては，プロセッサのシミュレータによる評価が広く用いられている．このプロセッサ・シミュレータや使用するベンチマーク・プログラム，評価方法にはさまざまなものが提案されており，それらには大きな特性の違いがある．また，近年では RISC-V と呼ばれる命令セットの登場や，SPEC CPU 2017 ベンチマークなどの新しいベンチマークの登場により，その環境は大きく変化している．これに対し本論文では，各種の評価方法によるシミュレーション結果の違いや，新しく登場した命令セット/ベンチマークなどの特性を明らかにすべく，さまざまな評価と解析を行った．

1. はじめに

プロセッサの研究や開発において，実際に LSI を作成するためには非常に長い時間と大きなコストが必要となる．このため，新しい方式を評価する際はまずはシミュレータに実装を行い，その上で評価や比較を行うことが普通である．そして，そのようなプロセッサの評価に用いるために，さまざまなプロセッサ・シミュレータやベンチマーク，そして評価の方法が提案され使用されている．

たとえば，プロセッサ・シミュレータとしては gem5 [1]，鬼斬弐 [2]，ChampSim [3] などが提案されており，広く使用されている．これらのシミュレータは，入力として受け取ることができるバイナリの ISA (instruction set architecture) やシミュレーション可能なマイクロアーキテクチャの機能（アウト・オブ・オーダ実行や分岐予測など）などが大きく異なる．また，シミュレーション方式にもさまざまな違いがあり，たとえば専用のツールや実際のプロセッサが出力したトレースを入力として駆動されるトレース・シミュレータや，シミュレータ自体が実行を行うものなどがある．これらの間では実装の容易さやシミュレーション精度などに大きな違いがある．

[1] 東京大学大学院情報理工学系研究科
[a)] tomokin@mtl.t.u-tokyo.ac.jp
[b)] koizumi@mtl.t.u-tokyo.ac.jp
[c)] degawa@mtl.t.u-tokyo.ac.jp
[d)] inaoka@mtl.t.u-tokyo.ac.jp
[e)] arimax@mtl.t.u-tokyo.ac.jp
[f)] shioya@ci.i.u-tokyo.ac.jp
[g)] irie@mtl.t.u-tokyo.ac.jp
[h)] sakai@mtl.t.u-tokyo.ac.jp

また，一般にプロセッサ・シミュレータによる実行は実際のプロセッサと比較して非常に遅いため，ベンチマーク・プログラム全体を実行することは現実的ではない．たとえば広く用いられている SPEC CPU 2017 [4] の中には 10^{13} 命令近くの実行命令数を持つものもあり，これらを実際のプロセッサ・シミュレータで実行した場合には数週間から数ヶ月が必要となることも多い．

このように現実的な時間でシミュレーションを完了させることは困難であるため，一部の実行命令区間を抽出し，その区間のみでシミュレーションを行うことが広く行われている．実行命令区間の抽出には SimPoint [5] と呼ばれる手法が広く用いられている．SimPoint はプログラム実行全体の中から，そのプログラムの特性を良く表す区間を抽出するためのプログラムである．SimPoint により抽出した実行区間を実行することにより，短い時間で高い精度のシミュレーションを行うことができる．

上記のように，プロセッサのシミュレータやベンチマーク，その評価方法にはさまざまなものが存在し，それらの特性は大きく異なる．しかしそれらの違いにより，シミュレーション結果にどのような違いが生じるかはそれほど研究されていない．また，近年では，RISC-V [6] と呼ばれる ISA が登場し，ベンチマークとしても SPEC CPU 2017 が登場するなど，それらの環境は大きく変化をしている．しかし，これらの新しく登場した環境について，それらの特性はまだあまり明らかになっていない．

そこで，本論文では様々なシミュレーション環境における違いを評価し，解析した．具体的には，RISC-V に対応

したサイクル・アキュレートな実行駆動のシミュレータ上で SPEC CPU 2017 を実行し，さまざまな特性を評価した．また，トレース・シミュレータと実行駆動シミュレータの比較を行い，トレース・シミュレータでは簡略化されている機構が全体の実行結果に与える影響を解析した．これに加え，SPEC CPU 2017 の SimPoint における部分実行の精度について評価を行った．

以降, 第 2 章では, シミュレーションに用いた最新の ISA である RISC-V について説明し, 第 3 章では, サイクル・アキュレート・シミュレータを用いて, 主に分岐予測についての投機実行が与える影響の調査結果と考察を述べる．第 4 章では, SimPoint について説明し, SimPoint によって定めた実行命令区間と長期間の実行を比較する．最後に, 第 5 章では, 本論文をまとめる．

2. RISC-V

RISC-V は 2010 年にカリフォルニア大学バークレイ校によって開発された最新の ISA である．RISC-V は以下のような特徴を持った ISA として設計されている．

- 特定のマイクロ・アーキテクチャに過度に依存しないような設計となっている
- オープン標準である
- 小規模の基本 ISA とオプションの拡張 ISA に分かれ，モジュール性を持つ
- 32bit，64bit の両方のアドレス空間に対応している
- 密結合な機能ユニットから疎結合なコ・プロセッサまで ISA 拡張が対応している
- 可変長命令セット拡張がある
- IEEE 754-2008 や C11，C++11 などの現代標準向けのハードウェア・サポートを提供している
- ユーザ・レベルと特権レベルのアーキテクチャが分離されており，完全な仮想化が可能である

RISC-V は特定のマイクロ・アーキテクチャへ過度に依存しない設計になっている．MIPS や SPARC などの 1980 年代に開発された過去の ISA はシングル・イシュー，イン・オーダ，5 段パイプラインに最適化されており，アウト・オブ・オーダ，スーパー・スカラに不向きな設計となってしまっている．MIPS などに実装されている遅延スロットは，パイプラインのステージ数が増え分岐以前に多くの命令がフェッチされるようになった現在，プロセッサの設計を複雑にする要因になってしまっている．他アーキテクチャと比較し，RISC-V は一部の実装方式向けの機能や他の実装方式を妨げるような機能を搭載しておらず，多様なマイクロ・アーキテクチャに対応できうるように設計されている．

RISC-V の重要な点として非営利団体が所有しオープン標準であるという特徴がある．一般的な商用 ISA はアーキテクチャの研究において障害となる部分が出てきてしまい，成功した研究の商業化などにも障害となる．同様に他の ISA は複雑なものが多く，研究分野により向いている “単純なサブセット” という特徴を持った ISA がないため，基本 ISA が非常にシンプルな設計となっている RISC-V は研究分野に適した ISA となっている．

ARM のように様々な機能全てを実装しないと動作しないような ISA では，その機能を必要としない場合コストの増加，効率の低減に繋がってしまう．一方，RISC-V は基本 ISA という中核となる非常に単純な命令セットと，オプションとして任意に選択できる多様な拡張 ISA に分かれているというモジュール性を持つ．これによりチップ開発のコストや消費電力を抑えることができる．

3. 投機的実行が高性能キャッシュ・マネジメントに与える影響

3.1 トレース・シミュレータとサイクル・アキュレート・シミュレータ

プロセッサ・シミュレータは，トレース・シミュレータとサイクル・アキュレート・シミュレータに大別される．トレース・シミュレータは，実機などで得られたプログラムの正しい動作のダンプ（トレース）を元にしたシミュレーションを行う．サイクル・アキュレート・シミュレータは，コア内部の挙動を一サイクル単位で正確にシミュレーションする．

トレース・シミュレータといっても，メモリ・アクセス系列のみのダンプによる簡易的なものから，命令列のダンプによっているためアウト・オブ・オーダ実行の影響を部分的に評価できるものまで存在する．しかし，そのいずれであっても，近年のプロセッサで採用されている各種投機的実行を正確に再現することができない．これは，トレースにはレジスタ上やメモリ上の値が含まれていないため，プログラムの誤った実行パス上では実行をシミュレーションできないためである．

一方，サイクル・アキュレート・シミュレータは，各種投機実行の副作用まで含めて CPU コアの挙動を完全に再現した正確なシミュレーションを提供する．しかし，その実行には非常に長い時間がかかるため，ベンチマークを完走させることは困難である．4 章では，ベンチマークを完走させるのではなく，部分的に実行することにより評価を行う手法を紹介する．

キャッシュの研究では，長期間のシミュレーションが必要とされるため，正確性を犠牲にしつつも高速なトレース・シミュレータを用いることが多い．実際，キャッシュ置換アルゴリズムやプリフェッチャの世界的な競争の場である Cache Replacement Championship (CRC) [7] や Data Prefetching Championship (DPC) [8] ではトレース・シミュレータ「ChampSim」による成績評価を行っている．しかし，キャッシュ・アクセスには，プログラムの誤った実

行パス上でのメモリアクセスも含まれる．特に投機予測ミスが発生し命令がフラッシュされた場合でも，その副作用がキャッシュ等に残ることがよく知られている [9]．Access Map Pattern Matching (AMPM) Prefetcher [10] は，このような投機的実行がプリフェッチャに与える影響を検討している．

以下に続く 3.2 節及び 3.3 節では，サイクル・アキュレート・シミュレータを使用することにより，近年の高性能キャッシュ置換アルゴリズムや高性能プリフェッチャはアウト・オブ・オーダ実行や投機実行の影響にどの程度耐性があるのかについて詳しい評価を行う．

3.2 シミュレーション環境

RISC-V 命令セットのシミュレーションが可能なサイクル・アキュレート・シミュレータとして，「鬼斬弐」 [2] を使用した．表 1 にシミュレーションのコンフィグレーションを示す．シミュレーションに用いたコンフィグレーションはキャッシュ・マネジメントに関して 2 種類ある．表 1 中 (a) は, 置換アルゴリズムに LRU を用い, プリフェッチャを接続しない, キャッシュ・マネジメントに関する研究でベースラインとして用いられるコンフィグレーションである．表 1 中 (b) は, 高性能なキャッシュ・マネジメントによる高速化の効果を測定するためのコンフィグレーションであり, プリフェッチャには DPC1 で最も成績の高かった AMPM を接続し, 置換アルゴリズムには CRC2 で好成績であった Signature-Based Hit Predictor (SHiP++) [11] を採用した．ベンチマークとしては, SPECspeed 2017 [4] に含まれる 20 本のベンチマークを RISC-V 向けにクロス・コンパイルしたバイナリを用いた．プログラムの先頭から 10G 命令が完了するまでスキップし, そこから 100M 命令の実行にかかるサイクル数をサイクル・アキュレートに測定した．

3.3 投機実行の影響の評価

図 1 にシミュレート結果を示す．CPI を示しているため, 棒グラフが短い方が性能が高いことを示している．Right はプログラムの正しい実行パス上の命令を実行するのにかかったサイクル数, Wrong はプログラムの誤った実行パス上の命令を実行するのにかかったサイクル数であり, これらの合計がプログラムを実行するのにかかったサイクル数となる．その横には, オラクルを用いた一切誤らない分岐予測器を使った場合のサイクル数を記載した．

ベンチマークの結果は, 大きく 3 種類に分けることができる．

1) 分岐予測がほとんどあたるもの．*603.bwaves_s* 等のベンチマークのグラフは, 分岐予測器の違いは性能にほとんど影響を与えないことが示している．

2) 分岐予測ミスが正しい実行パスの実行に影響を及ぼさないもの．*600.perlbench_s*, *641.leela_s*, *644.nab_s* のグラフは, これらのベンチマークでは正しいパスを実行するのにかかるサイクル数は, 分岐予測ミスの有無にほとんど影響されないことを示している．

表 1 シミュレーションのコンフィグレーション

プロセッサ	
ISA	RV64G
issue width	int:2，fp:2，mem:2
instruction window	int:32，fp:16，mem:16
branch predictor	8KB，8 components TAGE
BTB	2K entries，4-way
LSQ	load:48 entry store:48 entry
キャッシュ	
L1 I/D	32KB，8-way，64B line 4cycle latency，LRU
L2	256KB，8-way，64B line 8cycle latency，LRU
L3	2MB，16-way，64B line 20cycle latency， (a) LRU (b)SHiP++
メインメモリ	200cycle latency
プリフェッチャ	
L1 I/D	(a) なし (b)NextLine Prefetcher Distance: 0，Degree: 1
L2	(a) なし (b)Stream Prefetcher Distance: 8，Degree: 8
L3	(a) なし (b)AMPM Prefetcher 16KB zone，52 entries

3) 分岐予測ミスにより, 正しいパスの実行が高速化されるもの．*602.gcc_s*, *605.mcf_s* 等のベンチマークでは, 正しいパスの実行にかかるサイクル数は, 分岐予測ミスがないときよりも短くなっている．*619.lbm_s* に至っては, 間違ったパスの実行サイクル数以上短くなっているため, 有意な差ではないとはいえ, 分岐予測ミスがあった方が性能が高いという直観に反する結果を生んでいる．これらは, 正しいパスでのメモリ・アクセスと間違ったパスでのメモリ・アクセスに共通する部分があるため, 間違ったパスでのメモリ・アクセスが一種のプリフェッチャ（slipstream prefetcher）として働いていることによる．その影響は 4 つのベンチマークで 10%を超え, *620.omnetpp_s* では最大 23%に達する．

3 の影響はトレースシミュレータでは計測できない．しかし, *619.lbm_s* をのぞき, コンフィグレーション (a) であっても (b) であってもその相対関係は大きく変わっていない．よって slipstream prefetcher と高性能プリフェッチャは直交する性能向上を与えていることになる．そのため, 少な

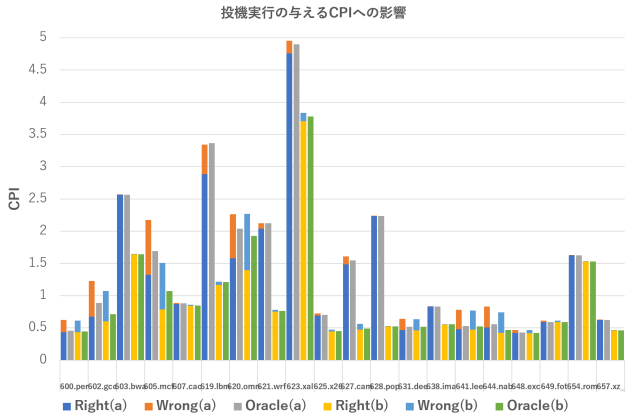

図 1　SPEC CPU 2017 における投機実行が CPI に与える影響

くとも今回のコンフィグレーションにおいては, トレース・シミュレータを用いた性能向上幅の計測は実際に即したものになっていることが明らかになった.

4. SimPoint による測定

4.1 関連研究

シミュレーションする命令数を減らすために良く用いられる手法として SimPoint [5] について説明する. SimPoint は実行するアプリケーションをいくつかのフェーズに分割して, 代表的なフェーズを抽出する手法である. 多くのフェーズで同一の命令を行う場合, アプリケーションの挙動は安定するので, 一部のフェーズのみの統計データ (ex IPC，Cache Miss) をサンプリングすることで, 全体の統計データが取得できるという考えに基づいている. つまり, 複数のフェーズの中で同一のフェーズをグルーピングし, 実行命令数を減らす. 本節では, SimPoint の手法について簡単に説明する.

はじめに, アプリケーションを定められた実行命令区間に分割する. 実行命令区間が大きすぎると, 適切なフェーズのみをサンプリングすることができず, 実行命令数を十分に減らすことができない. 一方で, 実行命令区間が小さいすぎても, 適切なフェーズが複数の実行命令区間にまたがり, 代表的な実行命令区間の特定が難しくなる. これらのトレードオフを踏まえ, SPEC CPU 2006 の場合では, 100M 命令もしくは 1G 命令を実行命令区間として定義する場合が多い [7], [8], [12].

次に, 複数の実行命令区間の間の類似性を計測する. SimPoint では, Basic Block Vectors (BBV) を実行命令区間の特徴量として用いる. ここでいう Basic Block とは, 内部に分岐を含まない命令区間を指す. 先頭からイニシャライズされた各 Basic Block を実行命令区間ごとにカウントした値を BBV と呼ぶ.

次に, 実行命令区間ごとの BBV を利用してクラスタリングを行う. 実行命令数の多いアプリケーションでは, Basic Block の数も非常に多くなり， BBV の次元数が巨大化する. そこで, 正規化および, ランダムに線形写像を取ることで, BBV の次元数を 15 になるまで削減する. 次元削減した BBV を利用して, SimPoint では, クラスタリングのアルゴリズムとして K-means を用いる. K-means はランダムに各 BBV を多次元空間内のクラスタに割当て, クラスタの重心を計算する. 各 BBV とクラスタの重心を比較し, 最も近いクラスタに各 BBV を割り当て直す. 上記の作業を規定した変化量以下になるまで繰り返すことで, クラスタリングを完了する. SPEC CPU 2006 の場合では, クラスタ数は最大 30 に限定するか, 単一のクラスタを定義するのが一般的である [5], [7], [8], [12].

最後に, K-means によって分類した各クラスタについて最も重心に近い実行命令区間を抽出する. この実行命令区間が SimPoint である. 各 SimPoint は, 分類した各クラスタの含む要素数によって重み付けされる.

4.2 シミュレーション環境

本論文では, SPEC CPU 2017 [4] を RISC-V 向けにクロスコンパイルしたバイナリに対して SimPoint の導出を行った. 実行命令区間は 100M で定義し, 単一のクラスタと最大 30 クラスタの 2 通りで分類した. 導出した SimPoint を表 2 に示す. それぞれのベンチマークにおける SimPoint の数と, 単一クラスタに限定した SimPoint(1 slice) での実行命令区間を表している. ベンチマークによって SimPoint の数は多岐に渡ることがわかる. 最小で *602.gcc_s* の 5 に対して最大で *654.roms_s* の 28 の SimPoints を実行する必要がある. また, 1slice においても, *602.gcc_s* のように先頭から 11.5G 命令後と, 比較的アプリケーションの序盤に SimPoint が存在するベンチマークから, *607.cactuBSSN_s* のように約 9T 先とアプリケーションの後半に SimPoint が存在するベンチマークに分かれる.

次に 1 slice の SimPoint を用いて, アプリケーションの性能を測定する. 各 SimPoint と 100G 実行シミュレートした結果を比較し, 実行結果の乖離を調査する. シミュレーションに用いるコンフィグレーションは，表 1 と同じものを用いた.

4.3 シミュレーション結果

図 2〜6 にシミュレート結果を示す. SimPoint での実行, 100G 実行とコンフィグレーション (a),(b) での値を示す. IPC と分岐予測の HitRate に関しては, SimPoint での実行, 100G 実行の誤差を示す. 各キャッシュの Miss per Kilo Instructions (MPKI) の結果では測定値の絶対値が小さく，誤差が大きくなってしまうので，表示していない.

まず，図 2 に示す IPC について，*603.bwaves_s*, *623.xalancbmk_s* において誤差が約 250%と非常に大きい. これらのベンチマークでは, 支配的なフェーズが多数存在しかつ, それらのフェイズの性質が大きく異なるため, 単

表 2　SPEC CPU 2017 の各ベンチマークごとの SimPoint 数と 1slice での実行命令区間

Benchmark	# SimPoints	1 slice (instructions)
600.perlbench_s	20	564.6G
602.gcc_s	5	11.5G
603.bwaves_s	25	110.3G
605.mcf_s	17	923.8G
607.cactuBSSN_s	27	9022.3G
619.lbm_s	27	4468.1G
620.omnetpp_s	23	573.8G
621.wrf_s	25	901.2G
623.xalancbmk_s	19	294.3G
625.x264_s	17	474.5G
627.cam4_s	13	275.1G
628.pop2_s	21	513.9G
631.deepsjeng_s	14	939.8G
638.imagick_s	18	1594.7G
641.leela_s	16	71.8G
644.nab_s	19	290.5G
648.exchange2_s	16	1576.3G
649.fotonik3d_s	20	4586.8G
654.roms_s	28	933.3G
657.xz_s	15	3097.1G

一の SimPoint では, 1 つのフェーズの情報しか得ることができないからである. また, これらのベンチマークの誤差はコンフィグレーション (a) に対して (b) の場合減少する. これは, 実行命令区間のボトルネックがメモリアクセスであり, キャッシュ・マネジメントの影響が大きく, IPC の絶対値が小さいほど効果が大きくなるからである. キャッシュ・マネジメントによる性能向上効果を見る場合, コンフィグレーション (a), (b) の誤差に相違は少ないので, 正規化 IPC で比較する場合には, SimPoint による誤差の影響は小さいことがわかる. コンフィグレーション (a), (b) のどちらの場合でも, 幾何平均で誤差は約 12%である.

図 3 に示す分岐予測器の HitRate について, 任意のベンチマークについて誤差は 10%以下, 幾何平均で約 1.5%である. キャッシュ・マネジメントと分岐予測器の相関は小さいので, コンフィグレーション (a), (b) の違いによっても有意な差はない. SimPoint の導出に用いる BBV が分岐に大きく依存するため, 分岐予測器の性能を表す指標として SimPoint は適切だと言える.

図 4, 図 5, 図 6 にキャッシュの MPKI を示す. *603.bwaves_s*, *649.fotonik3d_s*, *657.xz_s* において, L1 の MPKI の誤差が巨大であることがわかる. *649.fotonik3d_s*, *657.xz_s* では, SimPoint での L1 の MPKI が 100G 実行に対して大きいので, SimPoint によってメモリアクセスがボトルネックとなる実行命令区間を抽出することができている. 一方で, *603.bwaves_s* においては, SimPoint での L1 の MPKI が 100G 実行に対して小さく, SimPoint によってメモリアクセス自体が少ない実行命令区間を抽出してい

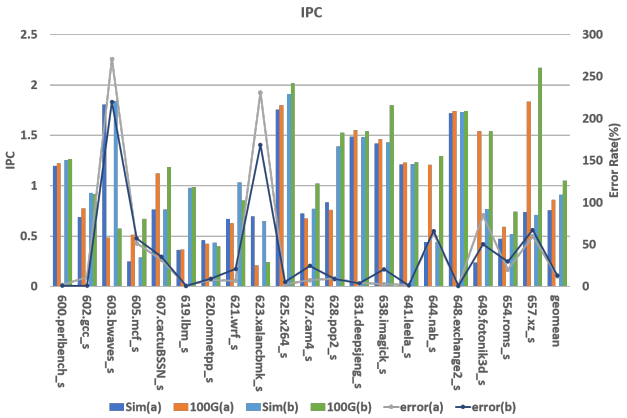

図 2　SPEC CPU 2017 の IPC

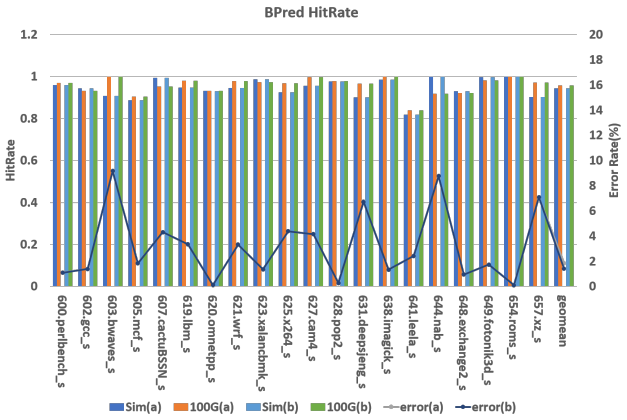

図 3　SPEC CPU 2017 の分岐予測器の HitRate

る. L2 の MPKI においては, *607.cactuBSSN_s*, *607.mcf_s* が, LLC の MPKI においては, *623.xalancbmk_s* が誤差が大きい. このように, 各キャッシュでの MPKI を個別に確認しないと, アプリケーションのフェーズは特定できないことがわかる. L1 の MPKI では, 幾何平均で約 120%, L2 の MPKI では, 幾何平均でコンフィグレーション (a) のとき約 160%, コンフィグレーション (b) のとき約 360%, LLC の MPKI では, 幾何平均で約 250%の誤差である. それぞれのコンフィグレーションを用いて正規化 MPKI で比較した場合, L1 では約 1%, L2 では約 90%, LLC では約 50%の性能向上の誤差があることがわかる. IPC に対して, MPKI の絶対値が小さいため, 正規化した値でキャッシュ・マネジメントによる性能向上効果を見る場合, 誤差が大きいことがわかる.

5. おわりに

本論文では RISC-V を実装したサイクル・アキュレート・シミュレータにより, SPEC CPU 2017 における投機実行と SimPoint が実行に与える影響を計測した.

投機実行が生み出す誤ったメモリ・アクセスが正しいメモリ・アクセスと共通しているため, 一種のプリフェッチャ (slipstream prefetcher) として働き, その影響を考慮しない

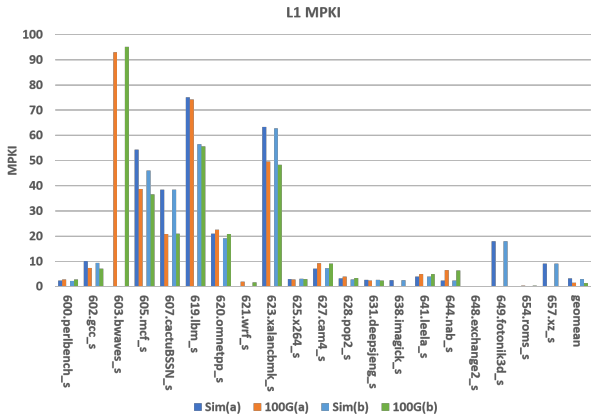

図 4　SPEC CPU 2017 の L1 キャッシュにおけるの MPKI

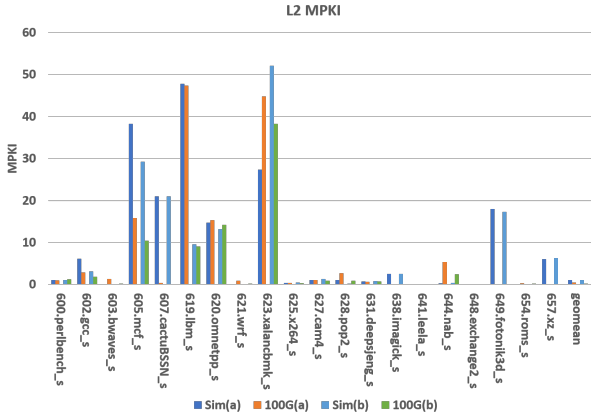

図 5　SPEC CPU 2017 の L2 キャッシュにおける MPKI

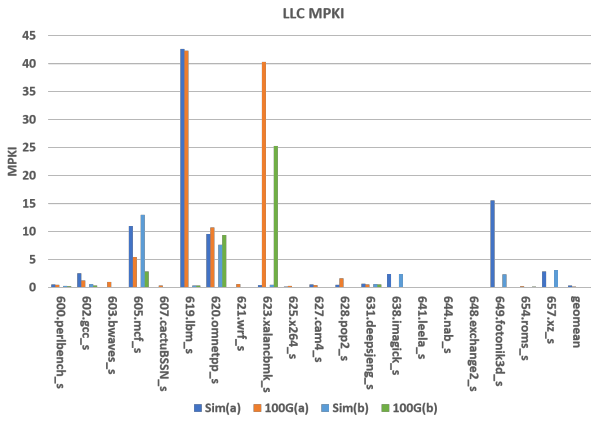

図 6　SPEC CPU 2017 の LLC における MPKI

場合と 10%以上の性能差が出るベンチマークは SPECspeed 2017 の中に 20%程度含まれることを示した．一方, AMPM Prefetcher は slipstream prefetcher と直交した性能向上を示していることも明らかにした．

Simpoint 区間と本実行区間の実行結果は IPC, 分岐予測器の HitRate, キャッシュの MPKI によって比較した．分岐予測器の HitRate に関して, ベンチマーク全体で誤差は 10%程度以下であった．一方で, IPC に関して, 最大で約 250%, キャッシュの MPKI に関しても大きく乖離することを明らかにした．

参考文献

[1] The gem5 simulator. https://github.com/gem5/gem5.
[2] 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 SACSIS2009, Vol. 2009, No. 4, pp. 120–121, 2009.
[3] Champsim simulator. https://github.com/ChampSim/ChampSim.
[4] SPEC CPU® 2017. https://www.spec.org/cpu2017/.
[5] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings 2001 International Conference on Parallel Architectures and Compilation Techniques*, pp. 3–14, September 2001.
[6] Andrew Waterman. *Design of the RISC-V Instruction Set Architecture*. PhD thesis, EECS Department, University of California, Berkeley, Jan 2016.
[7] CRC2. http://crc2.ece.tamu.edu/.
[8] Home — DPC3. https://dpc3.compas.cs.stonybrook.edu/.
[9] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, and Daniel Genkin. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 973–990, 2018.
[10] Yasuo Ishii, Mary Inaba, and Kei Hiraki. Access Map Pattern Matching for Data Cache Prefetch. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pp. 499–500, New York, NY, USA, 2009. ACM.
[11] Vinson Young, Chia-Chen Chou, Aamer Jaleel, and Moinuddin Qureshi. SHiP++: Enhancing Signature-Based Hit Predictor for Improved Cache Performance. 2017.
[12] Arun A. Nair and Lizy K. John. Simulation points for SPEC CPU 2006. In *2008 IEEE International Conference on Computer Design*, pp. 397–403, Lake Tahoe, CA, USA, October 2008. IEEE.