Vol.2014-ARC-213 No.11
Vol.2014-HPC-147 No.11
2014/12/9

ブルーム・フィルタを用いた メモリ・アクセス順序違反検出機構の評価

倉田 成己[1,a)]　塩谷 亮太[2]　五島 正裕[3]　坂井 修一[1]

概要：ロード・ストア・キュー (LSQ) の CAM を排除するため，RAM で構成されたハッシュ・フィルタを用いてメモリ・アクセス順序違反を検出する手法が提案されてきた．その中で，我々はパラレル・カウンティング・ブルーム・フィルタを用いた手法を提案している．ブルーム・フィルタ (BF) は，複数のハッシュ関数を利用することで非常に低い偽陽性率を達成している．また，我々はロード/ストア命令のアクセス・サイズの違いに対する対策なども行っている．本稿では，近年のハイエンド・プロセッサをモデルとしてシミュレーションによる評価を行い，平均 99.0%の IPC を保ちながら，LSQ の面積が 14.3%，消費電力が平均 22.0%となることを確かめた．

1. はじめに

ロード/ストア・キュー (LSQ) は，out-of-order スーパスカラ・プロセッサの構成要素の中で最も高コストなものの 1 つとなっている．

LSQ と CAM

Out-of-order スーパスカラ・プロセッサにおいて，LSQ は，ロード/ストア命令の依存による先行制約を守りつつ，out-of-order に実行する役割を果たす．その他の命令とは異なり，ロード/ストア命令は「曖昧」である，すなわち，先行制約を満たすためには，依存元のストア命令の発見，あるいは，メモリ・アクセス順序違反の検出と，動的なターゲット・アドレスの比較が必須である．ターゲット・アドレスの比較は，従来，**CAM** を用いて LSQ を構成することによって行われて来た．しかし CAM は，その構造上，回路面積と消費電力が大きい．

LSQ 規模の増加

ハイエンドの out-of-order スーパスカラ・プロセッサの規模の拡大は，ゆっくりとだが確実に続いている [1], [2]．特に，メモリの下位階層との速度差を埋めるため，in-flight なロード/ストア命令の数を増加させることは極めて重要であり，LSQ のエントリ数は拡大の一途をたどっている．また，同時実行可能なロード/ストア命令の数を増やすことは，LSQ を構成する CAM のポート数の増加につながる．

[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] 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
[3] 国立情報学研究所
National Institute of Informatics
[a)] kurata@mtl.t.u-tokyo.ac.jp

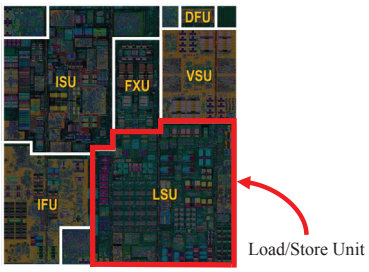

図 1　POWER8 のチップ写真 [3]

CAM の面積は，ポート数の 2 乗に比例して増加する．

これら 2 つの理由により，最近のハイエンド・プロセッサでは，LSQ は最も高コストな構成要素の 1 つとなっている．**図 1** に，POWER8 プロセッサのチップ写真を示す [3]．LSU がコアの 1/4 程度を占めている．L1D アレイの占める割合は高くなく，LSQ の面積と消費電力の削減が極めて重要であることが分かる．

フィルタを用いた順序違反検出

そこで我々は，RAM によって構成されたフィルタである**パラレル・カウンティング・ブルーム・フィルタ** (PCBF) を用いて順序違反/フォワーディング・ミス検出を行う手法を提案している [4]．ここで用いるフィルタは，ターゲット・アドレスをキーとするハッシュ・テーブルであり，ハッシュ値の衝突による偽陽性を不可避的に伴う．しかし我々は，**ブルーム・フィルタ** [5] の特徴である，複数のハッシュ関数を用いることによって極めて低い偽陽性率を達成することを利用し，CAM を排除することを可能にしている．

本稿では，この PCBF を用いて順序違反/フォワーディング・ミス検出を行う手法について，過去のフィルタを用いた手法 [6], [7], [8] との比較を含めて評価を行った．評価のモデルは POWER8[3] や HASWELL[2] を模した近年の

ハイエンド・プロセッサである．これにより，提案手法が実際のプロセッサにおいて性能を犠牲にせずに LSQ の面積や消費電力を大きく下げられることを示す．

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，提案手法のポイントであるブルーム・フィルタについて説明し，複数のハッシュ関数を用いることが本質的に重要であることを示す．ついで，3 章で，フィルタを用いた順序違反/フォワーディング・ミス検出について，提案手法を例にまとめる．その後，4 章で提案手法の詳細について説明し，5 章で IPC，面積および消費電力の評価を行う．

2. ブルーム・フィルタ

提案手法の第一の特長は，フィルタとして**ブルーム・フィルタ** (**BF**) [5] を用いることにある．本章では BF について説明する．まず次節で，BF の基本について説明する．その後 2.2 節で，ハッシュ関数の数と偽陽性率との関係を解析し，複数のハッシュ関数を用いることが BF において本質的に重要であることを示す．その後，2.3 節で，実際に提案で用いる**パラレル・カウンティング・ブルーム・フィルタ**を紹介する．

2.1 ブルーム・フィルタ

ブルーム・フィルタ (**BF**) とは，1970 年に Burton H. Bloom が考案した空間効率のよい確率的データ構造で，要素が要素の集合に含まれているかどうかを判定するために用いられる [5]．判定には，**偽陽性** (False Positive) があるが，偽陰性 (False Negative) はない．したがって，BF の応答が陽性である場合には，普通，偽陽性か真陽性かを判定するための**確認検査**が必要となる．

BF は，m エントリのビットの配列と，k 個のハッシュ関数 $h_0, \ldots, h_{k-1}$ からなる．**図 2** の例を用いて，BF の動作を説明する．この例では，要素は八進で 000 から 077 までの 64 通りとする．BF のエントリ数は $m = 8$，ハッシュ関数は h_0 = 要素の八の位，h_1 = 要素の一の位 の 2 つとする．

(A) 初期状態では，BF の全てのビットが 0 である（図では 0 は省略）．
(B) 要素 012 を BF に追加するとき，この要素のハッシュ値 $h_0 = 1$，$h_1 = 2$ に対応するビットをセットする．
(C) 同様に，要素 041 を追加するときは，$h_0 = 4$，$h_1 = 1$ に対応するビットをセットする．
$h_1 = 1$ に対応するビットは既にセットされているが，特別な動作は行わない．実装上は，盲目的に 1 を上書きすればよい．

ここで，例えば (B) で追加された 012 を検索すると，$h_0 = 1$，$h_1 = 2$ に対応するビットがいずれもセットされているため，012 が追加されていたと判定できる．

ブルーム・フィルタの偽陽性

BF には，ハッシュ値の偶然の衝突によって，偽陽性が発生する．例えば，図 2 (C) の，012 と 041 が追加されている状態において，024 を検索すると，$h_0 = 2$，$h_1 = 4$ に対応するビットのいずれもセットされているため，実際には追加されていないにもかかわらず結果は陽性となる．

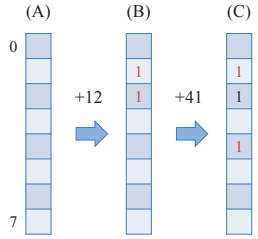

図 2 ブルーム・フィルタの例

このような偽陽性が発生する確率は，配列のエントリ数を増加させるよりも，以下で述べるように，ハッシュの数を適切に設定することで劇的に削減できる．

2.2 ブルーム・フィルタの陽性率

BF の**陽性率** = 偽陽性率 + 真陽性率 は，n 個の要素が配列に追加されており，ハッシュ値が一様に分布している場合，以下のような式になることが知られている．

$$P_{true} = \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \qquad (1)$$

この式からでも，m を増加させるより，k をわずかに増加させることによって，P_{true} が劇的に減少することが分かるであろう．実用上は，P_{true} をある一定の値以下にすることを要求される場合が多い．その場合には，k をわずかに増加させることによって，必要なエントリ数 m を劇的に減少させることができる．**図 3** に，エントリ数 m に対する陽性率 P_{true} を示す．曲線は，$k = 1, 2, 3$ と，m に対して最適な k を選択した場合の，計 4 本ある．BF に追加されている要素数 n は，$n = 30$ である．ここで，例えば陽性率を 0.1%未満にしたい場合，$k = 1$ では約 $m \approx 30,000$ ものエントリが必要だが，$k = 2$ では約 $m \approx 2,000$，$k = 3$ では約 $m \approx 850$ となっており，k をわずかに増やすだけで必要エントリ数 m が劇的に小さくできることが分かる．

また，m，n が決まっているとき，偽陽性率を最小とする k は $k = \ln 2(m/n)$ で与えられ，この時の偽陽性率は，$P_{true} = (1/2)^k \approx 0.6185^{m/n}$ となる．すなわち，P_{true} を一定に保つためには $m \propto n$ なる m で十分である．このことはスケーラビリティの点で極めて重要である．例えば提案手法では，in-flight なロード命令数を 2 倍に増やした場合でも，フィルタのビット数も 2 倍に増やせば，同程度の偽陽性率を達成できることになる．

このように，ハッシュの数が $k \geq 2$ であることこそが，BF において本質的であると言える．しかし，BF を用いたと主張する研究はいくつかあるが，そのいずれにおいても $k \geq 2$ について言及されていない [6], [7], [8].

2.3 パラレル・カウンティング・ブルーム・フィルタ

我々は，**パラレル・ブルーム・フィルタ** (**PBF**) [9] と**カウンティング・ブルーム・フィルタ** (**CBF**) [10] を組み合わせた**パラレル・カウンティング・ブルーム・フィルタ** (**PCBF**)

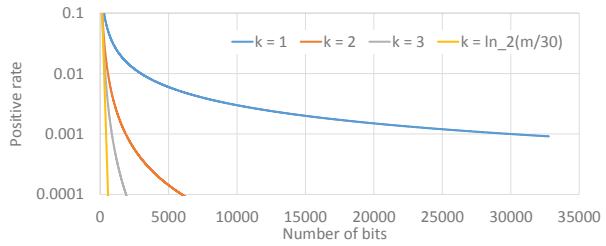

図 3 BF のエントリ数と 陽性率の関係

をアクセス順序違反検出に利用することを提案している．

PBF は，エントリ数 m の配列を，エントリ数 $m' = m/k$ の k 個の部分配列に分割し，各部分配列は，それぞれ 1 個のハッシュ関数をインデクスとして読み書きすることで，k 個の RAM のポート数はそれぞれ 1 とし，$k \geq 2$ の BF において RAM の面積の増加を抑える手法である．

また，CBF では BF の各エントリをビットからカウンタに拡張することで要素の削除を可能にした手法である．後の評価でも示すが，BF に含まなければならない要素数は高々 in-flight なロード命令の数であるため，削除が可能な CBF を用いたほうが効率が良い．

提案手法では，この PBF と CBF を組み合わせたパラレル・カウンティング・フィルタを用いる．

3. フィルタを用いた検出手法

本章では，既存手法と提案手法を含む，フィルタを用いてメモリ・アクセス順序違反/フォワーディング・ミス検出を行う手法についてまとめる．

以下，3.1 節で，ベースとなるプロセッサのモデルについて述べ，順序違反検出とフォワーディング・ミス検出が双子の問題であることを明らかにする．3.2 節で手法の分類について述べた後，3.3 節で，提案手法を用いてフィルタによるメモリ・アクセス順序違反/フォワーディング・ミス検出を説明する．最後に，3.4 節で，ロード/ストア命令のアクセス・サイズに起因した，フィルタを用いる手法の問題点を説明する．

3.1 順序違反検出とフォワーディング・ミス検出

1 章で述べたように，out-of-order スーパスカラ・プロセッサにおいて，**ロード・キュー** (**LQ**) と**ストア・キュー** (**SQ**) は，ロード/ストア命令の依存関係を守りつつ，out-of-order に実行する役割を果たす．ロード/ストア命令は曖昧であり，依存関係を守るためには，動的なターゲット・アドレスの比較が必須である．

LSQ の CAM

CAM ベースの実装では，LQ/SQ に対して，以下のように優先順位付きの連想検索が行われる：

SQ L1D の更新は，不可逆的に行うため，通常ストア命令のコミット時に行われる．ストア・データは，ストア命令の実行〜コミットの間，SQ に置かれ，SQ から後続のロード命令への**フォワーディング**が行われる．ロード命令は実行時に SQ に対して連想検索をかける．

LQ Store Set などの依存予測器 [11], [12] を用いてロード/ストア命令を投機的に発行する場合，予測が誤りであるとメモリ・アクセス順序違反として検出される．順序違反検出は，ストア命令の実行時に LQ を連想検索し，当該ストア命令の後続のロード命令で，実行済みで，かつ，ターゲット・アドレスが一致するものがないかを探すことになる．なお，順序違反検出の結果は，予測器の学習に用いられる．

LQ/SQ の CAM は，同程度か，SQ の方が大きい．エントリ数は LQ の方大きいのが普通であるが，検索ポート数は SQ の方が等しいか大きいからである．LQ/SQ の検索ポート数はストア/ロード命令の同時発行数で与えられるが，ストア命令の同時発行数はロード命令のそれに等しいかより小さいからである．

したがって，LSQ の面積と消費電力を問題にするのであれば，LQ と SQ の CAM を同時に省略することが望ましい．次節以降で述べる手法では，ほぼすべてが LQ の CAM を省略している一方で，SQ の CAM を省略しているものは SVW と提案手法だけである．SQ の CAM を省略できない手法は，LSQ の問題の半分以下しか解決していないことになる．

投機的フォワーディング

SQ の CAM を省略するためには，更に**投機的フォワーディング**を組み合わせることになる [8], [13], [14]．予測には，依存予測器の結果をそのまま用いることができる．先行するストア命令に依存すると予測されたロード命令は，当該ストア命令の発行後に発行されると同時に，当該ストア命令から直接ストア・データを受け取ればよい．ただし，投機的フォワーディングを行えば，当然のことながら，フォワーディング・ミス検出を行う必要がある．

次節以降で述べる SVW と提案手法では，これら 2 つの問題をほぼ同様の方法で解決している．このように，順序違反/フォワーディング・ミス検出は，LQ/SQ の CAM を省略するための手法であり，問題の規模と解決方法においても「双子」の問題であると言える．

3.2 フィルタを用いた手法の分類

RAM によって作られたフィルタを用いる手法の基本は，

(1) ロード/ストア命令のターゲット・アドレスのハッシュ値をキーとするテーブル (RAM) に対して，
(2) ロード/ストア命令の実行/コミット時にリード/ライトを行うことで，

順序違反/フォワーディング・ミス検出を行うことにある．同一のターゲット・アドレスに対するロード/ストア命令は，テーブルの同一エントリへのリード/ライトを行うことになる．このエントリへのライトによって一方の命令がある特定の状態にあることを示し，他方の命令がそのエントリをリードし，値を検査することで検出を行うのである．

表 1 に，従来のフィルタを用いた代表的なアクセス順序違反検出手法と提案手法をまとめる．ここでは，従来の手法として **Store Vulnerability Window** (**SVW**) [8], **Delayed Memory Dependence Checking** (**DMDC**)

Vol.2014-ARC-213 No.11
Vol.2014-HPC-147 No.11
2014/12/9

表 1　各手法の比較

		SVW	DMDC		SHF	Proposal
			1st	2nd		
Table	value	Sequence Number	Bit/Counter			
	# hash functions	1				≧2
Access	Order Violation	Normal	Reverse	Normal	Reverse	Reverse
	Forwarding Miss		N/A			

[15]，**Single Hash Filter** (**SHF**) [6] を挙げる．提案手法との違いのや問題点の詳細は文献 [4] に委ねるが，それぞれの手法において，問題がある項目を網掛けで示している．

各手法は，① テーブルの構成 と，② テーブルへのアクセス の 2 点によって特徴づけられる．以下，それぞれについて説明する．

① テーブルの構成

いずれの手法においてもテーブルのキーにはロード/ストア命令のターゲット・アドレスのハッシュ値が用いられるが，テーブルのバリューには以下の 2 種類がある：

a. シーケンス・ナンバ　プログラム・オーダにしたがってロード/ストア命令にシーケンシャルに付された番号．

b. ビット　対応する命令が特定の状態にあることを示す 1 ビット．

b. ビットを用いる場合には，提案手法のように，複数のハッシュ関数を用いることが考えられるが，それについて言及した既存研究はない．また，a. シーケンス・ナンバを用いる場合には，複数のハッシュ関数を用いる方法は知られていない．なお，テーブルのバリューが b. ビット の場合には，リード/ライトに加えて，ビットのリセットを行う必要がある．

② テーブルへのアクセス

テーブルへのライト → リードは，ロード/ストア命令の実行/コミットのいずれかのタイミングで行われる．提案手法を含め，既存の手法は，以下の 2 つに分類される：

i. st-cmt → ld-cmt　先行するストアのコミット時にライトし，後続のロードのコミット時にリードする．

ii. ld-exec → st-exec/cmt　後続のロードの実行時にライトし，先行するストアの実行，もしくは，コミット時にリードする．

i. st-cmt → ld-cmt は，確認検査ができない，依存予測器の学習ができない という 2 つの問題がある．解決のためには，LSQ とは別の手立て（例えば SVW における SPCT [8]）を必要とする．また，いくつかの手法はフォワーディング・ミス検出に対応しておらず，LQ の CAM は省略する一方で，SQ の CAM はそのまま残る．

以下，提案手法をもとに，順序違反検出とフォワーディング・ミス検出の基本的な動作を説明する．

3.3　フィルタを用いた検出手法

本節では，提案手法を例に，フィルタによる順序違反/フォワーディング・ミス検出の基本的な方法ついて説明する．

パイプライン図の表記法

各手法のパイプライン動作を説明する前に，本論文で用いるパイプライン図の表記法について説明する．順序違反/フォワーディング・ミス検出において意味があるのは，L1D アクセスを行うロード/ストア命令の実行（L1D アクセス）とコミット・ステージの前後関係のみである．したがって，以下に示すような，簡略化されたパイプライン図を用いることにする．

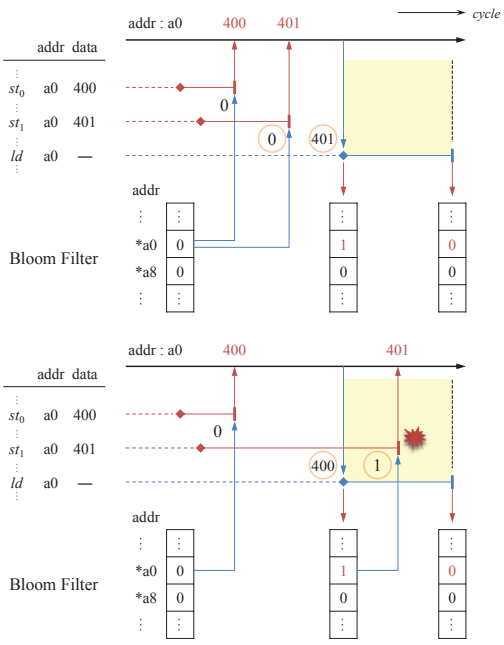

図 4　提案手法の順序違反検出
順序違反がない場合（上）とある場合（下）

：フェッチ〜実行ステージ

：実行ステージ（L1D からのロード）

：実行〜コミット・ステージ

：コミット・ステージ（L1D へのストア）

順序違反検出

この簡略化されたパイプライン図を用いて，**図 4** に提案手法の順序違反検出の様子を示す．

同図では，ストア命令 st_0，st_1，ロード命令 ld が，その順序でフェッチされている．各命令のターゲット・アドレスはすべて $a0$ であり，st_0，st_1 のストア・データは，それぞれ，400，401 であるとする．プログラム・オーダ上，st_0 より st_1 の方が下流にあるから，ld は，st_0 のストア・データ 400 ではなく，st_1 のストア・データ 401 をロードしなければならない．

同図中，上が順序違反がない場合を示す．上部の右向き矢印は，（L1D の）アドレス $a0$ の値の変化を表す．st_1 のコミット後に ld が実行されており，ld は（L1D の）アドレス $a0$ から 401 をロードすることができる．一方，同図中下では，st_1 のコミットが ld の実行より遅れてしまったため，ld は st_0 のストア・データ 400 をロードすることになり，順序違反検出として検出しなければならない．

提案手法は，前節の分類で言えば，b. ビット と，ii. ld-exec → st-cmt の組み合わせにあたる．ロード命令は，実行時にテーブルのビットをセットし，ストア命令は，コミット時にテーブルをリードしてビットを検査する．

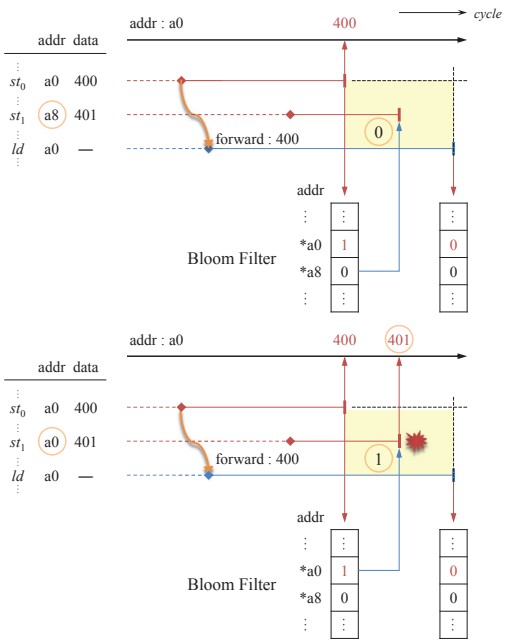

図 5 提案手法のフォワーディング・ミス検出
フォワーディング・ミスがない場合（上）とある場合（下）

フォワーディング・ミス検出

図 5 に提案手法のフォワーディング・ミス検出の様子を示す．同図では，予測器からの指示に従い，st_0 のストア・データ 400 が ld に投機的にフォワーディングされている．ld は，コミット時に自身のターゲット・アドレス $a0$ をフォワーディング元の st_0 のそれと比較し，一致を確認する．しかしそれだけでは，フォワーディング・ミスがないと判断するには十分ではない．同図下では，st_0 より下流の st_1 のターゲット・アドレスもまた $a0$ となっており，ld は正しくは st_1 のストア・データ 401 をロードしなければならなかった．すなわち，フォワーディング・ミスである．

提案手法では，前述の順序違反検出の手法をわずかに変更することによって，この状況を検出することができる．すなわち，フォワーディングを行った場合には，ロード命令に代わって，フォワーディング元のストア命令がテーブルのセットを行うのである．その結果，監視領域は，図中網掛けした部分に変わる．順序違反検出の場合と同様に，この領域内で同一アドレスに対してコミットを行った ST があれば，フォワーディング・ミスである．同図下では，実際に st_1 がテーブルから 1 をリードし，フォワーディング・ミスとして検出されることになる．

なお，順序違反検出の場合と同様の理由により，監視領域はロード命令のコミット時までよい．

フィルタと確認検査

ここで用いられているテーブルはハッシュ・テーブルであり，ハッシュ値の偶然の**衝突**による偽陽性 (false positive, **偽陽性**) が起こり得る．前出の例では，たとえ st_1 と ld のターゲット・アドレスが異なっていても，そのハッシュ値が一致した場合には，順序違反，フォワーディング・ミスとして検出されることになる．

そのため，これらの手法は**フィルタ**としての役割を果たすことになる．すなわち，低コストだが偽陽性が生じ得るフィルタによって，高コストだが偽陽性のない**確認検査**の実行頻度を減らすのである．提案手法の場合，確認検査は LQ のシーケンシャル・サーチによって行うことを想定しており，これには数十サイクルもの時間がかかる．

したがってこれらのフィルタの性能は，一次的にはフィルタの容量に対する偽陽性率の低さによって評価されることになる．

3.4 ロード/ストア命令のアクセス・サイズ

一般的な命令セットでは，ロード/ストア命令には，1，2，4，8 B のサイズが存在する．CAM による順序違反検出では，マスクを用いることで比較的容易にこサイズの違いに対応することができる．しかし，フィルタを用いた場合の対応は容易ではない．例えば，アドレス 0x08〜0x0f の 8 B にアクセスするストア命令の後に，0x0c〜0x0f の 4 B にアクセスするロード命令があるとする．これらの命令は，始点となるアドレスが異なるが，アクセスされるバイトは重なっているため，当然順序違反検出の対象となる．しかし，単純に始点となるアドレスを用いて PCBF にアクセスすると，これらの命令は異なるアドレスへのアクセスとみなされ，偽陰性が発生してしまう．

例えば SVW では，該当アクセスを含む 8 B ワードを単位として順序違反検出を行っている [8]．先ほどの例で言えば，ストア命令とロード命令のいずれもアドレス 0x08〜0x0f までの 8 B ワードへのアクセスとみなし，アドレスを 0x08 とすることで順序違反を検出することが可能になる．しかし，この手法では 1，2，4B の隣接するアドレスへアクセスするストアとロードが同一アドレスへのアクセスとみなされ，一部のプログラムでは偽陽性が多発する．

また，DMDC ではこの問題に触れているものの，評価は行っていない．

4. 提案手法の詳細

提案手法は，2 章で紹介したパラレル・カウンティング・ブルーム・フィルタ (**PCBF**) を用いて，順序違反/フォワーディング・ミス検出を行う．本手法では，CAM や，フィルタを用いた従来の手法と比較して，回路面積と消費電力を大幅に削減しながら，CAM に匹敵する性能を達成できる．

提案手法の動作は，基本的には 3.3 節で述べたとおりである．ただし，BF ではなく CBF を用いるため，ビットのセット/リセットがカウンタのインクリメント/デクリメントに変わる．

本章では，提案手法の詳細について述べる．以下，まず陽性やオーバーフローが発生した際の動作について述べる．その後，4.3 節で，サイズを考慮した提案手法のフィルタ構成について説明する．

4.1 陽性時の動作

陽性は，ストアによるリード時に k 個すべてのハッシュ値が一致していると発生する．PCBF が陽性となったと

き，実際に順序違反が発生している（真陽性）か否か（偽陽性）を確認検査する．提案手法では，PCBF の低い偽陽性率により，コストの高い方法を選択することができる．

そのため，提案手法の確認検査は，LQ のシーケンシャル・サーチによって行うこととする．バックエンドをストールし，LQ を検索し，ターゲット・アドレスが一致するロードが存在するかどうか探す．アドレスが一致するロード命令が見つからなければ偽陽性であるので，何もせずに実行を再開すればよい．実際にアドレスが一致するロード命令が存在すれば真陽性であり，そのロード命令（以降の命令）を再実行する．同時に，Store Set などの依存予測器 [11], [12] にロード/ストア命令の PC を通知し，学習する．

バックエンド・ストール中であるので，LQ の検索には，LQ の発行ポートを用いることができる．LQ の発行ポート数 1〜2 程度で，LQ 内のロードの 32〜64 命令程度を検索するとすると，バックエンドがストールするサイクル数は平均で数十サイクル程度にもなる．しかし，5 章で述べるように，PCBF の低い陽性率ならば，この程度のペナルティにも耐えることができる．

4.2 オーバーフローへの対処

陽性が k 個のハッシュ値すべてが一致しないと起こらないのに対して，オーバーフローは，k 個のハッシュのうち 1 個でもフルになっていると起こるので，より問題である．

オーバーフローに際して，単にストールすることはできない．当該カウンタをデクリメントするはずのロード命令の実行が停止すると，デッドロックが発生するからである．

オーバーフロー時のペナルティは，以下のようにして軽減することができる．まずバックエンド・パイプラインはストールするが，コミット・パイプラインはストールさせない．この状態で，当該カウンタをデクリメントするはずのロード命令が既に実行済みであった場合には，いずれコミットし，カウンタをデクリメントするので，実行を再開することができる．実行済みのすべての命令がコミットしてもカウンタがフルのままであった場合には，オーバーフローを起こすロード命令と後続の命令をフラッシュし再実行する．5 章で述べるが，多くの場合フラッシュは必要なく，数サイクル待てば実行を再開することができる．

4.3 ロード/ストア命令のサイズへの対応

3.4 で述べたように，フィルタを用いた順序違反検出ではサイズの違いに対応するのは容易ではない．我々は，以下の 2 種類の手法を用いることでこの問題に対応する．

(1) 4 B や 8 B の命令は，LP64 や LLP64 モデルを用いた多くのプログラムによって利用されている．一方で，1，2 B の命令はあまり使われていない．そこで，4，8 B 命令のための **MCBF** と，1，2 B 命令のための **SCBF** の 2 種類の PCBF を用いる．また，1，2 B 命令を振り分けるための **StBF** も用意する．

(2) 2.3 節で述べたように，PCBF は k 個の部分配列に分割される．それら k 個のうちいくつかを 8 B 未満のアクセスを区別するために用いる．

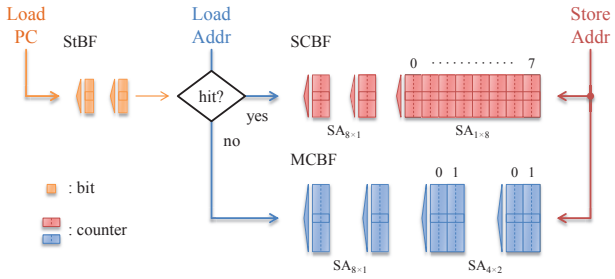

図 6 提案フィルタの構成

図 6 に，提案手法の全体構成を示す．

MCBF

MCBF は，主に 4，8 B のロード命令の情報を保持する．MCBF は，$\mathbf{SA_{8B\times1}}$ と $\mathbf{SA_{4B\times2}}$ と呼ぶ 2 種類のサブ・アレイで構成される．$SA_{8B\times1}$ と $SA_{4B\times2}$ はいずれも 8B 単位でアクセスされるが，そのエントリが異なる．$SA_{8B\times1}$ は 8 B 単位のカウンタが 1 つのみだが，$SA_{4B\times2}$ のエントリは 2 つのカウンタがあり，それぞれ 8 B 中の隣接する 4 B に対応している．4 B 命令は $SA_{4B\times2}$ の対応するカウンタのみにアクセスし，8 B 命令は両方のカウンタにアクセスすることにより，4 B アクセスの区別が可能となる．

サブ・アレイの数

MCBF は $k_{8B\times1}$ 個の $SA_{8B\times1}$ と，$k_{4B\times2}$ 個の $SA_{4B\times2}$ の合わせて $k = k_{8B\times1} + k_{4B\times2}$ 個で構成される．k を一定としたとき，$k_{4B\times2}$ を増やすとサイズによる偽陽性は減少するが，エントリあたりのカウンタが 2 個であるため，面積は増加する．最適な $k_{4B\times2}$ は，5 章で評価している．

SCBF

SCBF は，1 B と 2 B のロード命令の情報のみを保持する．1, 2 B のロード命令は少ないため，SCBF も小容量で十分である．SCBF は MCBF と同様に，$SA_{8B\times1}$ と $SA_{1B\times8}$ の 2 種類のサブ・アレイによって構成される．$SA_{1B\times8}$ はエントリが 8 個のカウンタで構成され，8 B 中の対応するカウンタにアクセスする．

StBF

StBF は，ロード命令を振り分ける際に用いる．MCBF や SCBF とは異なり，StBF はカウンタではない通常の PBF である．また，キーはターゲット・アドレスではなく命令の PC である．

ロード命令が PCBF にアクセスする際，まず PC を用いて StBF を参照する．ここで StBF が陽性であれば SCBF を，陰性であれば MCBF を，それぞれインクリメント/デクリメントする．

一方で，ストア命令のアクセス順序違反検出時には StBF を用いず，MCBF と SCBF の両方にアクセスし，いずれか一方でも陽性であれば PCBF が陽性であるとみなす．これは，ストア命令とそれに依存するロード命令の PC が異なるためである．また，ストア命令専用の StBF は用いることができない．これは，StBF が偽陽性であると PCBF が偽陰性を示してしまうためである．

StBF への PC の追加は，MCBF が陽性を示し，さらに

確認検査で 4 B 中の異なるアドレスに対するアクセスが原因の偽陽性であると判明した際に行われる．このようにすることで，上記のような偽陽性を起こさない 1，2 B 命令を MCBF で管理することができ，SCBF をさらに小さくすることができる．

StBF のリセットは，一定のサイクル —— 5 章の評価では，128K サイクルが経過するごとに全ビットをフラッシュすることによって行う．

5. 評価

提案手法と既存手法について，主に，フィルタのサイズと IPC についての評価を行った．以下，5.1 節で評価環境についてまとめた後，5.2 節で提案手法の評価を行い，5.4 節で既存手法との比較を行う．

5.1 評価環境

ベンチマーク

ベンチマークは SPEC CPU 2006 [16] の全 29 プログラムで，データ・セットは *ref* を使用し，各プログラムは gcc 4.6.1 の -O3 でコンパイルした．評価は最初の 1G 命令をスキップし，直後の 100M 命令をシミュレートする．

シミュレータ

シミュレーションには cycle-accurate なプロセッサ・シミュレータである鬼斬弐 [17] を用いた．ベースラインとなるプロセッサの構成は**表 2** に示す通りである．

なお，命令セットは Alpha で，拡張命令セットとして byte-word extensions を適用している．そのため，1 B，2 B のロード/ストア命令が出現する．

また，依存予測器としては，Store Set [11] を用いた．

5.2 提案手法の構成の検討と IPC の評価

本節では，提案手法の以下の効果を評価する：

(1) フィルタの容量に対する k および c の効果
(2) サイズを考慮した PCBF の効果
(3) IPC 低下の内訳

表 3 に，提案手法のデフォルトのパラメタを示す．特に断りのない場合には，これらの値を用いた．

以下では，偽陽性率とベースラインに対する相対 IPC の，ベンチマークのクラスごとの幾何平均を示す．ベースラインは，CAM を用いて順序違反/フォワーディング・ミス検出を行うモデルで，フィルタを用いた手法のような偽陽性による性能低下はない．いくつかのグラフを示すが，横軸はフィルタの総ビット数で，縦軸は偽陽性率，もしくは，ベースラインに対する相対 IPC である．偽陽性率のグラフでは左下にある曲線ほど，相対 IPC のグラフでは左上にある曲線ほど，性能がよいことになる．

表 2 プロセッサの構成

Parameter	Value
ISA	Alpha 21264A w/ byte-word ext.
fetch/issue/cmt	8/8/8 inst./cycle
inst window	64 entries unified
ROB	192 entries
LQ/SQ	72/42 entries
branch pred	16KB:g-share/8K:local hybrid
miss penalty	15 cycles
BTB	2K-entry, 4-way
L1D	64KB, 8-way, 64B/line, 2 cycles
L2C	512KB, 8-way, 64B/line, 8 cycles
L3C	8MB, 8-way, 64B/line, 24 cycles
main memory	200 cycles

表 3 PCBF のパラメータ

BF		$m' \times c \times b \times k$ = total
MCBF	$SA_{8B\times1}$	$128 \times 3 \times 1 \times 2 = 768$
	$SA_{4B\times2}$	$128 \times 3 \times 2 \times 2 = 1536$
SCBF	$SA_{8B\times1}$	$16 \times 3 \times 1 \times 2 = 96$
	$SA_{1B\times8}$	$8 \times 3 \times 8 \times 1 = 192$
StBF	—	$64 \times 1 \times 1 \times 2 = 128$
total		2720

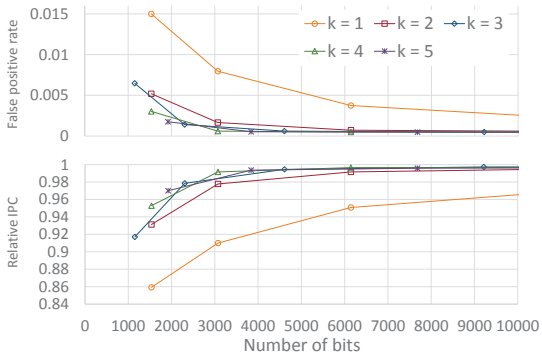

図 7 k の効果 偽陽性率（上）と IPC（下）

k の効果

2.2 節では，ハッシュ関数の数 k をわずかに増加させることで陽性率を劇的に減少させることができることを解析的に明らかにした．ここでは，シミュレーションで実際に偽陽性率が減少し，IPC の低下が抑えられることを示す．

図 7 に，結果を示す．曲線はそれぞれ $k = 1, 2, \ldots, 5$ の場合で，m' を変化させたものである．また，3.4 で述べたサイズの問題を分離するため，フィルタは k 個の $SA_{4B\times2}$ で構成された MCBF のみを用いた．

$k = 1$ の場合と比べ，$k \geq 2$ の場合に，偽陽性率が劇的に減少し，相対 IPC の低下も低く抑えられている．$k = 4$，$m' = 128$ の場合に良い結果が得られているため，以降ではこれらのパラメータを用いる．

c の効果

ここでは，m' を増加させるよりも c をわずかに増加させるほうがカウンタのオーバーフローを効率的に回避できることを示す．そのため，フィルタは MCBF のみを用いた．

図 8 に結果を示す．曲線は，$c = 1, \ldots, 4$ ごとに m' を変化させたものである．また図 7 と同様に，評価には k 個の $SA_{4B\times2}$ で構成された MCBF のみを用いている．

$c = 1$ の場合は，オーバーフローによる性能低下が大きく，m' を増やしてもその効果は小さい．一方で，$c \geq 2$ では IPC が改善しており，特に $c = 3, 4$ の場合にベースラインとほぼ同等の IPC を達成していることがわかる．したがって，以降 $c = 3$ に固定する．

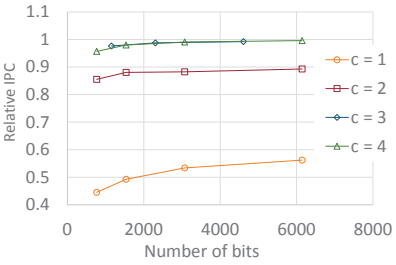

図 8 c の効果 相対 IPC

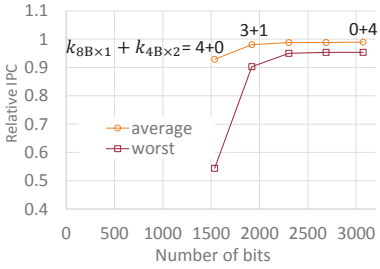

図 9 MCBF の構成 相対 IPC

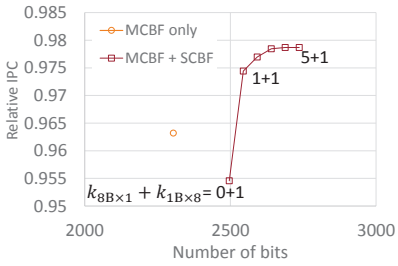

図 10 SCBF の構成 相対 IPC

MCBF の効果

ここでは，4.3 で提案した，ロード/ストア命令のサイズに対応した PCBF の効果について評価する．

図 9 は，MCBF の k を固定し，$SA_{8B\times1}$ と $SA_{4B\times2}$ の数を変化させたときの平均相対 IPC と最悪相対 IPC である．図では，サブ・アレイの数を $k = 4 = k_{8B\times1} + k_{4B\times2} = 4+0, 3+1, 2+2, 1+3, 0+4$ と変化させている．$SA_{8B\times1}$ を $SA_{4B\times2}$ に置き換えるたびにエントリあたりのカウンタ数が増加するため，合計のビット数も増加する．

図の結果では，$k_{8B\times1} + k_{4B\times2} = 4+0$ の場合に IPC が大きく低下している．一方で，$3+1$ 以降の構成では相対 IPC が大きく改善している．これは，4 B のロード/ストア命令が多くのプログラムで広く利用されており，フィルタを用いた手法においてはサイズの問題に対処することが重要であることを示している．評価によると，MCBF の構成が $2+2$ の場合で十分であるので，以降この構成を用いる．

SCBF の効果

4.3 で示したように，SCBF は 1, 2 B のロード/ストア命令によるサイズの問題が発生するプログラムの性能を改善するために用いる．SPEC CPU 2006 の中でこの問題が発生するプログラムである gcc, h264ref, astar, xalancbmk を評価のために用いた．

図 10 に評価結果を示す．丸で示された点が SCBF を用いなかった場合の結果であり，曲線が SCBF をそれぞれ異なる構成 $k = k_{8B\times1} + k_{1B\times8} = 0+1, 1+1, 2+1, \ldots, 5+1$ で用いた場合の結果である．$SA_{8B\times1}$ と比べて $SA_{1B\times8}$ は 8 倍の大きさとなり，$k_{1B\times8}$ を大きくすると効率も大きく低下する．そのため，ここでは $k_{1B\times8}$ は 1 に固定している．

評価では，$k_{8B\times1} \geq 1$ のときに，SCBF を用いない場合よりも良好な結果を得られた．

以降の評価では，表 3 の構成を用いることとする．

オーバーフロー時のストールによる効果

4.2 節で述べたように，オーバーフロー時にはすぐにフラッシュするのではなく，バックエンドをストールさせ，カウンタのデクリメント 1 を待ってからフラッシュした方がよい．これを示すため，我々は 4.2 節で述べた手法と，オーバーフロー時にすぐにフラッシュする手法を評価・比較した．ここで，フラッシュに伴う正確なペナルティを測定することは困難であるため，フラッシュ発生時のペナルティを 15 サイクルとし，フラッシュ数に乗算することで合計のペナルティを算出した．

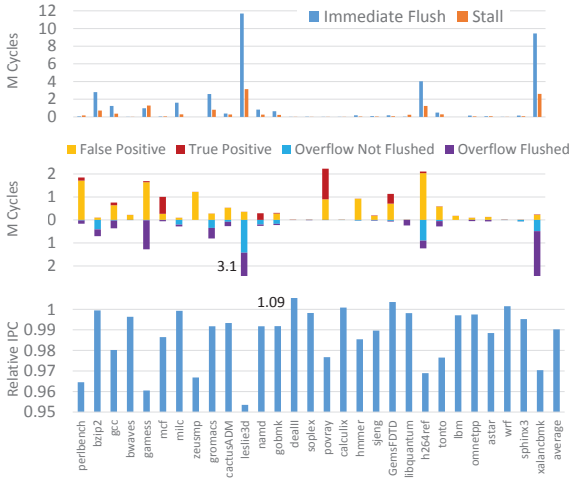

図 11 オーバーフロー時のペナルティ（上）
PCBF によるペナルティの内訳（中）相対 IPC（下）

図 11（上）に評価結果を示す．図はベンチマークごとのペナルティの合計サイクル数を表し，グラフの左がすぐにフラッシュするモデル，右がストールするモデルである．ここで，ストールするモデルには，ストールしたサイクル数の他，カウンタのデクリメントが行われなかった場合のフラッシュ・ペナルティも含まれている．

評価結果をみると，ストールするモデルでは，フラッシュするモデルに比べてペナルティが約 1/3 になっていることがわかる．

ベンチマークごとのペナルティと相対 IPC

図 11（中）は，プログラムごとの性能低下要因の内訳を示している．図の正方向は陽性によるペナルティを，負方向はオーバーフローによるペナルティをそれぞれ表す．

また，図 11（下）はベンチマークごとのベースラインに対する相対 IPC を示す．これらの図を比較すると，相対 IPC はペナルティと強く相関していることがわかる．しかし，その原因は様々である．たとえば，perlbench や zeusmp, povray は陽性によるペナルティが原因で IPC が低下している一方，leslie3d や xalancbmk はオーバーフローによって IPC が低下している．また，gamess と h264ref は陽性とオーバーフローの両方が原因となって IPC が低下している．また，dealII などのいくつかのプログラムでは，ベースラインの CAM を用いた手法よりも IPC が向上している．これは，ベースライン・モデル特有の偽陽性が発生しているためである．後続のストア命令のデータをロード命令にフォワードしたあとで，先行するストア命令が実行さ

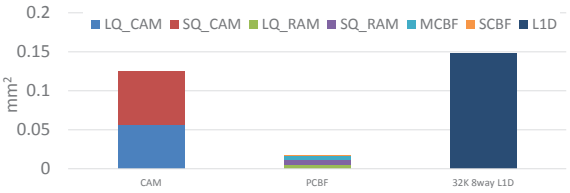

図 12 ベースラインと提案手法および L1D キャッシュの面積

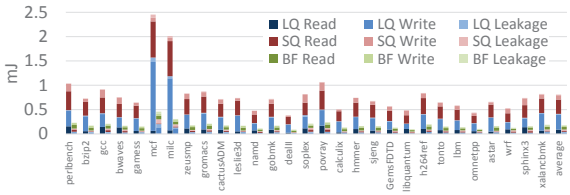

図 13 ベースラインと提案手法のプログラムごとの消費エネルギー

れると，CAM による手法では偽陽性が発生するのだが，紙面の都合上詳細は割愛する．

5.3 提案手法の面積と消費エネルギーの評価

ここでは，前節で決定した構成の提案手法の消費電力および面積評価を，CACTI 5.3 [18] ITRS 32nm を利用して，ベースラインの CAM を用いた手法と比較する．

ただし，PCBF を構成する MCBF などは CACTI が想定していないほど小規模なため，CACTI の最小構成を線形に縮小するなどして概算していることに注意されたい．

LSQ の構成

LSQ の構成は POWER7[1] や HASWELL[2] を参考とした．つまり，LQ および SQ は発行キューとは別体であり，LQ は参照アドレスが格納された CAM または RAM cell array，SQ は参照アドレスが格納された CAM cell array + データが格納された RAM cell array または参照アドレスとデータが格納された RAM cell array であるとする．

評価では，ロード/ストア命令を 2 命令同時発行，2 命令同時コミット可能であるとしている．また，ストア命令の L1D への書き込みはサイクル・スチールによって行われることを想定している．このとき，LQ，SQ ともに 2-write，2-search，2-read の合計 6 ポートが必要となる．一方 PCBF では，書き込みポートは LQ，SQ ともに CAM モデルと同様の 2-write 必要だが，読み出しとアクセス順序違反検出は共通のポートを利用するため 2-read とし，合計 4 ポートずつとしている．PCBF は，ロード命令によるインクリメント/デクリメントが read-modify-write となるため，4-read，4-write と比較的ポート数は多くなるものの，エントリあたりのビットが少ないため，後述のように面積や消費電力は低く抑えられている．

面積の評価

面積の評価結果を**図 12** に示す．左のグラフが CAM を用いた手法の LQ と SQ の面積の合計，中央のグラフが PCBF を用いた手法の LQ と SQ，PCBF の合計である．また，右のグラフは参考として Intel HASWELL[2] の L1D キャッシュを想定した 32K エントリ，8-way のキャッシュの面積を表している．

表 4 評価モデル

	Filter	Forwarding	Confirmatory Test	Predictor Learning	False Positive Stall Cycles
Baseline	N/A	SQ CAM	N/A	LQ CAM	0
SVW	SVW	Speculative	Load Re-Exec	SPCT	1
DMDC	DMDC	SQ CAM	N/A		Flush
DMDC + DPRED			Load Re-Exec	SPCT	1
SHF	CBF ($k = 1$)		LQ CAM		1
PCBF	PCBF ($k \geq 2$)	Speculative	LQ Sequential Search		16.9 (avg)

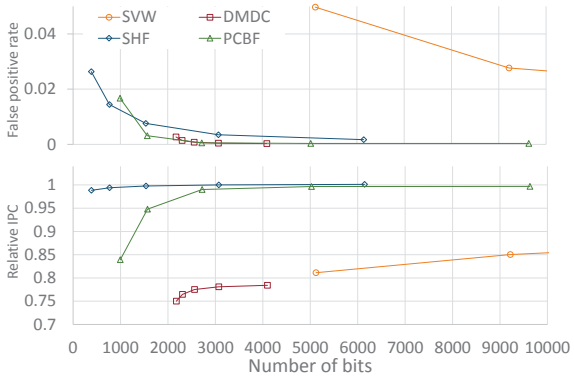

図 14 表 4 の評価モデル 偽陽性率（上）と相対 IPC（下）

図のとおり，PCBF を用いた手法は CAM を用いた手法と比較して 14.3%の面積であることがわかる．また，CAM を用いた手法では，LQ と SQ の面積の合計がキャッシュの面積に対して 84.3%もの大きさを占めているのに対し，PCBF を用いた手法では 12.1%まで削減されている．

消費エネルギーの評価

プログラムごとの消費電力を**図 13** に示す．LQ，SQ，PCBF に対してそれぞれ読み込み，書き込み，リーク電力の内訳を表している．一般に，プログラム中にはストア命令よりもロード命令が多い．そのため，LQ の書き込みと SQ の読み込みのエネルギーが大きく，特に CAM を用いた手法でその傾向が顕著に現れている．また，PCBF を用いた手法では消費電力が大きく削減されており，CAM を用いた手法と比較して平均で 22.0%となっている．

5.4 既存手法との IPC の比較

最後に，既存手法と，サイズと IPC との関係を比較する．

評価モデル

表 4 に，評価したモデルをに示す．ただし，既存手法の網掛けの部分は以下のように理想化されている：

- SQ-CAM/LQ-CAM は，SQ/LQ がそれぞれ CAM で構成されていることを表す．グラフの横軸はフィルタのビット数であり，CAM による面積増はグラフには反映されてない．
- LQ-CAM による確認検査は，1 サイクルで可能である．
- ロード再実行は，陽性が検出された直後の 1 サイクルで可能である．

既存手法のパラメタは，各文献で示された値とした．ただし，SPCT（Store PC Table）のエントリ数は，文献 [8] には明記されていないので，我々の事前の評価で最も高い IPC を示した 16 とした．

評価結果

図 14 の結果から，以下のことが分かる：

- SVW は効率が非常に悪く，IPC を維持するには大きな容量が必要となる．これは，テーブルのエントリがシーケンス・ナンバであり，ビット・テーブルと比較してテーブルの利用効率が悪いのに加えて，複数のハッシュ関数を用いられず偽陽性率が高くなるためである．
- DMDC の偽陽性率は，提案手法と同等である．しかし，依存予測器を用いないため，偽陽性に加えて真陽性も頻発するので IPC は大きく低下している．この結果は，Store Set などの依存予測器に関する研究における optimistic モデルの評価結果と整合的である [11], [12]．
- SHF は，偽陽性率に関しては提案手法の $k = 1$ の場合とほぼ同一となる．ただし SHF は，LQ-CAM によって 1 サイクルで確認検査が行われるため，シーケンシャル・サーチを行う提案手法と比較すると，IPC が同等になっている．逆に言えば，提案手法は LQ-CAM を省略しながら，低い偽陽性率によって LQ-CAM を用いた場合と同等の IPC を達成していると言える．

6. おわりに

本稿では，PCBF を用いたメモリ順序違反/フォワーディング・ミス検出手法の IPC と面積，消費エネルギーの評価を行った．PCBF によるメモリ順序違反/フォワーディング・ミス検出は，BF の低い偽陽性率に着目して高コストな確認検査の回数を削減することで，RAM を用いた LSQ で CAM を用いた LSQ と同等の性能を可能とし，LSQ の面積や消費電力を削減することに成功している．評価の結果，IBM POWER7 や HASWELL といった近年のハイエンド・プロセッサをモデルとした構成において，平均 99.0%の IPC を保ちながら，LSQ の面積が 14.3%，消費電力が 22.0%となることが確かめられた．

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. 23300013，26280012 による．

参考文献

[1] Sinharoy, B., Kalla, R., Starke, W. J., Le, H. Q., Cargnoni, R., Van Norstrand, J. a., Ronchetti, B. J., Stuecheli, J., Leenstra, J., Guthrie, G. L., Nguyen, D. Q., Blaner, B., Marino, C. F., Retter, E. and Williams, P.: IBM POWER7 multicore server processor, *IBM Journal of Research and Development*, Vol. 55, No. 3, pp. 1:1–1:29 (online), DOI: 10.1147/JRD.2011.2127330 (2011).

[2] Krewell, K.: INTEL'S HASWELL CUTS CORE POWER, Vol. 2 (2012).

[3] Stuecheli, J.: Next Generation POWER microprocessor, *Hot Chips 25*, (2013)
(the photograph is an excerpt from: http://www.hotchips.org/wp-content/uploads/hc_archives/hc25/HC25.20-Processors1-epub/HC25.26.210-POWER-Studecheli-IBM.pdf).

[4] 倉田成己，塩谷亮太，五島正裕，坂井修一：ブルーム・フィルタを用いたメモリ・アクセス順序違反検出，情報処理学会研究報告，2014-ARC-212, No. 17, pp. 1–15 (2014).

[5] Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM*, Vol. 13, No. 7, pp. 422–426 (online), DOI: 10.1145/362686.362692 (1970).

[6] Sethumadhavan, S., Desikan, R., Burger, D., Moore, C. R. and Keckler, S. W.: Scalable Hardware Memory Disambiguation for High ILP Processors, *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, Washington, DC, USA, IEEE Computer Society, pp. 399– (2003).

[7] Castro, F., Chaver, D., Pinuel, L., Prieto, M., Tirado, F. and Huang, M.: Load-store queue management: an energy-efficient design based on a state-filtering mechanism, *2005 International Conference on Computer Design*, pp. 617–624 (online), DOI: 10.1109/ICCD.2005.70 (2005).

[8] Roth, a.: Store Vulnerability Window (SVW): Re-Execution Filtering for Enhanced Load Optimization, *32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 458–468 (online), DOI: 10.1109/ISCA.2005.48 (2005).

[9] Sanchez, D., Yen, L., Hill, M. D. and Sankaralingam, K.: Implementing Signatures for Transactional Memory, *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pp. 123–133 (online), DOI: 10.1109/MICRO.2007.24 (2007).

[10] Almeida, J. and a.Z. Broder: Summary cache: a scalable wide-area Web cache sharing protocol, *IEEE/ACM Transactions on Networking*, Vol. 8, No. 3, pp. 281–293 (online), DOI: 10.1109/90.851975 (2000).

[11] Chrysos, G. Z. and Emer, J. S.: Memory dependence prediction using store sets, *ACM SIGARCH Computer Architecture News*, Vol. 26, No. 3, pp. 142–153 (online), DOI: 10.1145/279361.279378 (1998).

[12] Roesner, F., Burger, D. and Keckler, S. W.: Counting Dependence Predictors, *2008 International Symposium on Computer Architecture*, pp. 215–226 (online), DOI: 10.1109/ISCA.2008.6 (2008).

[13] Martin, M. and Roth, a.: Scalable Store-Load Forwarding via Store Queue Index Prediction, *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'05)*, pp. 159–170 (online), DOI: 10.1109/MICRO.2005.29 (2005).

[14] Sha, T., Martin, M. M. K. and Roth, A.: NoSQ: Store-Load Communication Without a Store Queue, *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, Washington, DC, USA, IEEE Computer Society, pp. 285–296 (online), DOI: 10.1109/MICRO.2006.39 (2006).

[15] Castro, F., Pinuel, L., Chaver, D., Prieto, M., Huang, M. and Tirado, F.: DMDC: Delayed Memory Dependence Checking through Age-Based Filtering, *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pp. 297–308 (online), DOI: 10.1109/MICRO.2006.21 (2006).

[16] The Standard Performance Evaluation Corporation: SPEC CPU2006 suite, http://www.spec.org/cpu2006/.

[17] 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).

[18] Thoziyoor, S., Ahn, J.-H., Monchiero, M., Brockman, J. and Jouppi, N.: A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies, *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 51–62 (online), DOI: 10.1109/ISCA.2008.16 (2008).