Vol.2015-ARC-216 No.4
2015/8/4

メモリ・アクセス順序違反検出手法の評価

西川 卓[1]　塩谷 亮太[2]　入江 英嗣[1]　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．しかし，我々の以前の提案であるパラレル・カウンティング・ブルーム・フィルタを用いた手法を除いて，複数のハッシュ関数を利用する手法はこれまでに知られていなかった．そこで，既存手法の 1 つである Store Vulnerability Window について，複数のハッシュ関数を利用する手法を提案する．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，複数のハッシュ関数を利用することで低い偽陽性率を実現できることを確かめた．

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．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 [1]．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．

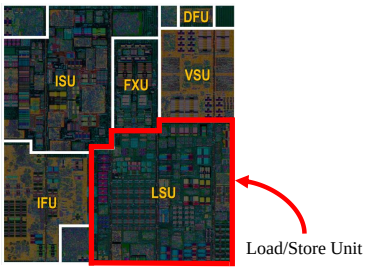

図 1　POWER8 のチップ写真 [1]

フィルタを用いた順序違反検出

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以前に提案している [3]．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** (**BF**)[4] 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し，CAM が排除可能であることを示してきた [3]．その評価の中で，我々は既存手法に対して，複数のハッシュ関数を適用する方法が知られていないとして，既存手法に対し優位性を示していた．そこで本稿では，フィルタによる順序違反検出を行う手法の一つである **Store Vulnerability Window(SVW)** に複数のハッシュ関数を適用する手法を提案する．

[1] 東京大学大学院情報理工学系研究科
[2] 名古屋大学大学院工学研究科
[3] 国立情報学研究所

本稿の構成

本稿の構成は以下の通りである．まず，2 章にて，順序違反検出の説明を交えながら，各種フィルタによる順序違反検出手法の分類について述べる．そして，3 章で，提案手法の元となる SVW について述べ，4 章で SVW に複数のハッシュ関数を適用する手法を提案する．続く 5 章で提案手法の評価を行い，PCBF を用いた手法との比較評価を行う．最後に 6 章 にて本論文をまとめる．

2. フィルタを用いた順序違反検出

本章では，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，2.1 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．2.2 節で手法の分類について述べた後，2.3 節で，ブルーム・フィルタ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，2.4 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．

2.1 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 [5] 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 の方が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 PCBF 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．

表 1 各手法の比較

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

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 [6], [7], [8]．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 PCBF 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．

2.2 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をまとめる．ここでは，従来の手法として **Store Vulnerability Window** (**SVW**) [6], **Delayed Memory Dependence Checking** (**DMDC**) [9], **Single Hash Filter** (**SHF**) [10], そして我々かつて提案した**パラレル・カウンティング・ブルーム・フィルタ** [3] による手法を挙げる．各種法の違いや問題点の詳細は文献 [11] 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．

これら， 2 種類の方式において，複数のハッシュ関数を用いることに言及した研究は我々の手法 [3] を除いて存在しない．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．フィルタによる順序違反検出手法は，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，Store Set のようにストア命令の PC を活用する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 [6]）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，我々が以前に提案したブルーム・フィルタを用いた手法をもとに，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．

2.3 フィルタを用いた検出手法

我々は以前にブルーム・フィルタを用いた順序違反検出機構を提案している．ブルーム・フィルタはある要素がある集合に帰属するかを判定するための確率的データ構造であり，複数のハッシュ関数を用いることで，低い偽陽性を実現するフィルタである．ブルーム・フィルタの数学的な解析結果など詳しい説明は文献 [3] を見ていただきたい．本節では，我々が以前に提案したブルーム・フィルタによる順序違反検出がシンプルで分かりやすいので，それ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．

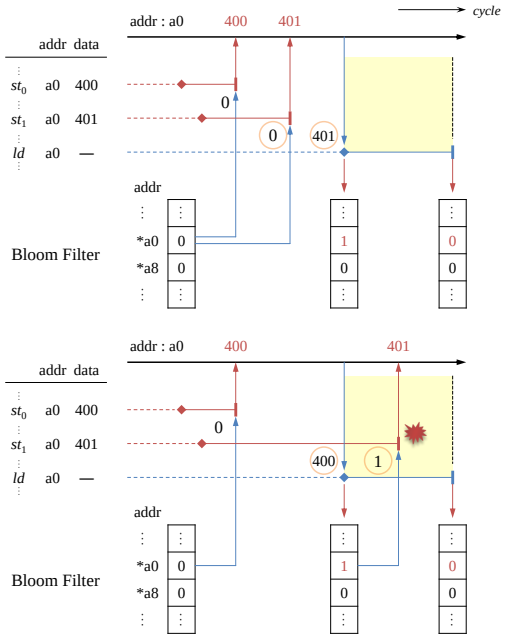

図 2 提案手法の順序違反検出
順序違反がない場合（上）とある場合（下）

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．

：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 2** にブルーム・フィルタ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

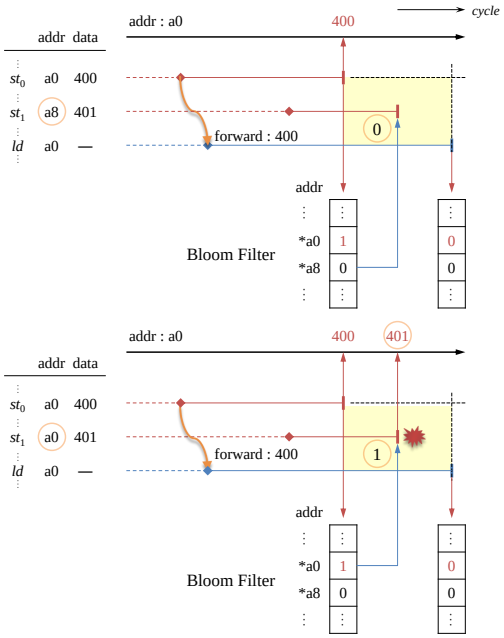

図 3 提案手法のフォワーディング・ミス検出
フォワーディング・ミスがない場合（上）とある場合（下）

なり，順序違反検出として検出しなければならない．

ブルーム・フィルタによる手法は，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．

フォワーディング・ミス検出

図 3 にフォワーディング・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．

ブルーム・フィルタによる手法では，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．確認検査の方法は，例えば我々の PCBF による手法では，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いる．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．

2.4 サイズの問題

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 [6]．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．

3. Store Vulnerability Window(SVW)

本章では本稿で改良手法を提案する SVW について説明する．

3.1 Store Vulnerability Window(SVW)

Store Vulnerability Window (**SVW**) は，元々は，ロード再実行と呼ばれる手法において，ロード再実行の頻度を削減するために提案された手法である [6]．**ロード再実行**とは，実行時に加えてコミット時にもロード命令を再

実行し，両者のロード・データを比較することで順序違反を検出する手法である．ロード再実行を確認検査と捉えれば，SVW の手法はフィルタの役割を果たしている．

2.2 節で述べた分類に従うと，SVW は，a. シーケンス・ナンバ と i. st-cmt → ld-cmt の組み合わせにあたる．

3.2 Store Sequence Number (SSN)

シーケンス・ナンバとしては，Store Sequence Number (**SSN**) と呼ぶ，ストア命令のみにプログラム・オーダ順にシーケンシャルに割り当てられた番号を用いる．

SSN をバリューとするテーブル本体は，Store Sequence Bloom Filter (**SSBF**) と呼ばれる．なお，Bloom Filter と名付けられてはいるが，ビットではなくシーケンス・ナンバをバリューとする場合，複数のハッシュ関数を用いる方法は知られておらず，文献 [6] でも複数のハッシュ関数に関しての言及はない．

また，最後にコミットしたストア命令の SSN を保持するレジスタを用意し，ロード命令は実行時にこの値をリードする．これを $\mathrm{SSN}_{\mathrm{cmt}}$ と呼ぶ[*1]．同一アドレスに対するストア命令とロード命令に関して，以下のことが言える；ロード命令の $\mathrm{SSN}_{\mathrm{cmt}}$ 以下の SSN を持つストア命令は，ロード命令実行時にはコミットしているのだから，このロード命令はそのストア命令のストア・データをロードしたことが保証される．逆に，$\mathrm{SSN}_{\mathrm{cmt}}$ より大きい SSN を持つストア命令があった場合には，順序違反である．

3.3 順序違反検出

2.2 節で述べた分類に従うと，SVW のテーブルへのアクセスは，i. st-cmt → ld-cmt となる．**図 4** の例では，ストア命令 st_{12} は，コミット時に，SSBF の $a0$ に対応するエントリと $\mathrm{SSN}_{\mathrm{cmt}}$ の 2 カ所に自身の SSN である 12 をライトする．st_{13} は，同じく 2 カ所に 13 をライトする．一方，ロード命令 ld は，実行時に $\mathrm{SSN}_{\mathrm{cmt}}$ をリードしておき，コミット時には SSBF の同じく $a0$ に対応するエントリをリードして，両者を比較することで検出を行う．

図 4（下）は，順序違反がある場合の動作である．ld は，実行時に $\mathrm{SSN}_{\mathrm{cmt}}$ として 12 をリードする．その後，st_{13} が SSBF の対応するエントリにコミット時に 13 をライトした後，ld がコミット時にリードすると 13 で，$\mathrm{SSN}_{\mathrm{cmt}}$12 より大きいため，順序違反が発生したことが分かる．

i. st-cmt → ld-cmt は，前節で述べた ii. ld-exec → st-cmt と逆の関係にあるため，ロード命令ではなくストア命令が「監視領域」を設定すると考えると分かりやすいかもしれない．図 4 では，st_{13} が設定する領域を網掛けで示した．この領域内で ld が実行を行うと，13 より小さい $\mathrm{SSN}_{\mathrm{cmt}}$ をリードすることになり，順序違反検出となる．ただし，ブルーム・フィルタを用いた手法の場合と異なり，この領域の設定は事後的である，すわなち，ld の実行時には領域は

[*1] 元論文では，SSN_{NVUL}.

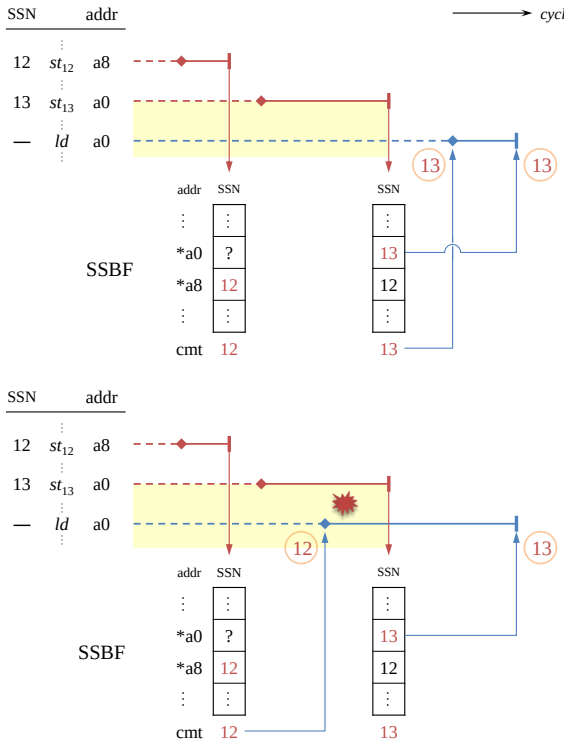

図 4 SVW の順序違反検出：
順序違反がない場合（上）とある場合（下）

まだ設定されておらず，この時点では順序違反検出であるかどうか分からない．そのため，ld の実行時には，コミット時に事後的に判定するための情報 —— $\mathrm{SSN}_{\mathrm{cmt}}$ を得ておくのである．

3.4 フォワーディング・ミス検出

SVW では，順序違反検出をわずかに変更することでフォワーディング・ミス検出も実現されている．SVW では，フォワーディング先のロード命令は，$\mathrm{SSN}_{\mathrm{cmt}}$ の代わりにフォワーディング元のストア命令の SSN を用いてコミット時の比較を行うことでフォワーディング・ミス検出を実現する．フォワーディングが行われる時は，ストア命令からロード命令にストア・データと同時に SSN も送られる．フォワーディング先のロード命令は，$\mathrm{SSN}_{\mathrm{cmt}}$ の代わりに送られて来た SSN を用いてコミット時の比較を行い，これらの値が一致していなければフォワーディング・ミスと判断できる．

図 5（下）では，ロード命令 ld がストア命令 st_{12} から値をフォワーディングされているが，実際に依存していたのは st_{13} である．このことは，送られて来た st_{12} の SSN 12 と，ld のコミット時に SSBF から読み出した st_{13} の SSN 13 を比較し，異なっているのでフォワーディング・ミスと判断できる．

3.5 a. シーケンス・ナンバ の問題点

テーブルのバリューとしてシーケンス・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

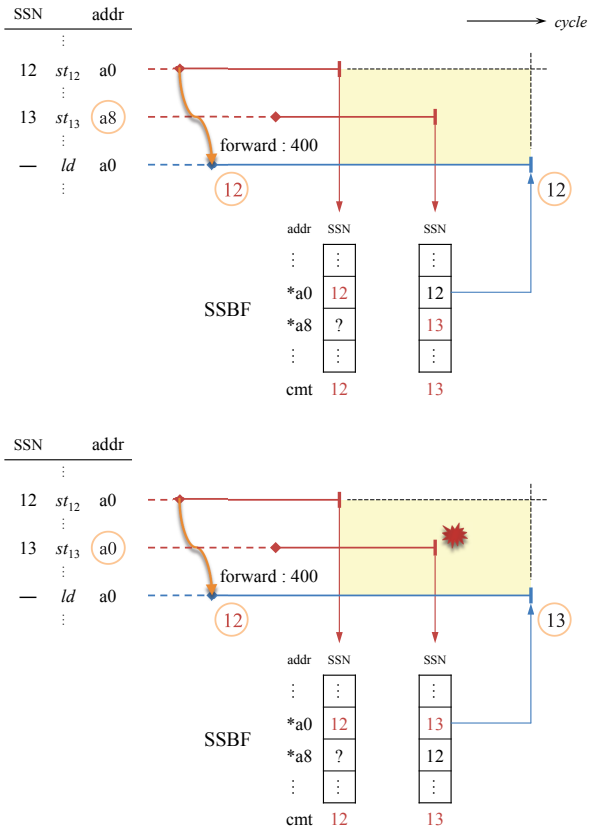

図 5 SVW のフォワーディング・ミス検出：フォワーディング・ミスがない場合（上）とある場合（下）

ない．そのため，SVW は容量の割に高い偽陽性率を持つとされている [3]．本稿は，4 章において，偽陽性の問題を解決するために，複数のハッシュ関数を適用する方法を提案している．

3.6 i. st-cmt → ld-cmt の問題点

2.2 節で述べたように，テーブルへのアクセスが i. st-cmt → ld-cmt の場合には，確認検査，及び依存予測器の学習ができないという問題があった．SVW では確認検査の問題は先にも述べたようにロード再実行により解決している．

また，依存予測器の学習のために，Committed Store PC Table (SPCT) と呼ぶテーブルを別途用意している．これは，ターゲット・アドレスをキー，ストア命令の PC をバリューとするタグレスの連想テーブルである．資源量の節約のためにタグレスとしたので，得られる PC が正しいとは限らない．間違っていた場合には依存予測器が汚されることになる．だがこのテーブルの一番の問題は，フィルタ本体に匹敵するほど大きいことである．

これはストア命令の PC をベースとする依存予測器を用いるために生じる問題で，[8] では PC を使わない動的な命令間距離をベースとした依存予測器が提案されている．

4. 提案手法

SVW のようなシーケンス・ナンバを用いた順序違反検出手法においては，複数のハッシュ関数を用いる手法がこれまでに知られていなかった．そこで，本章では SVW に改良を加え，複数のハッシュ関数を用いる手法を提案する

4.1 複数のハッシュ関数を用いた SVW

提案手法は，SVW に複数のハッシュ関数を用いる手法であるため，2.2 節で述べた分類は SVW と同じく，a. シーケンス・ナンバ と i. st-cmt → ld-cmt の組み合わせにあたる．SVW と異なる点は，SSN をバリューとするテーブル本体を，$SSNT_k(k = 1, 2,)$ としたところである．最後にコミットしたストア命令の SSN を保持するレジスタ SSN_{cmt} などは SVW と変わらない．しかし，ロード/ストア命令のコミット時に k 個の SSNT にアクセスし，それぞれの SSNT が持つハッシュ関数を用いて該当エントリにアクセスして，リード/ライトする点が異なる．このとき，$SSNT_k$ から読み込んだ値によって，それぞれのテーブルにおいて，順序違反検出を行うが，k 個の全てのテーブルにおいて順序違反を示さなければ，それは順序違反ではない．これは，順序違反検出を行うフィルタ一般に言えることだが，SVW に偽陽性はあるが偽陰性のない検出をしていることから考えてもらうと分かりやすい．つまり，それぞれのフィルタにおいて，陰性が検出されたならば，それは偽陰性ではないのである．それゆえに，陰性が少なくともひとつのフィルタで検出されれば，他のフィルタでの陽性は，偽陽性であることが原理的に分かる．

順序違反検出

図 6 は st_{12}，st_{13}，ld の順に命令がフェッチされており，ストア命令には添字で付したシーケンス・ナンバが割り当てられている．またそれぞれのメモリ命令のターゲット・アドレスは上図では上から順に，e0，f0，e0，下図では e0，e0，e0 である．またフィルタの役割を果たすテーブルを $SSNT_0$，$SSNT_1$ としており，今回はそれぞれアドレスの上位ビットのハッシュ値，下位ビットのハッシュ値をエントリとするテーブルとした．同図において，ld が実行時に SSN_{cmt} として 12 をリードする．その後，st_{13} は 2 つの SSNT の対応エントリにコミット時に 13 をライトする．ld はコミット時に， 2 つの SSNT の該当エントリからバリューをリードするのだが，同上図では $SSNT_0$ から 12，$SSNT_1$ から 13 が読み込まれる．ここでは $SSNT_0$ から読み込まれる値は順序違反を示すのだが，$SSNT_1$ から読み込まれる値は順序違反を示さないので，コレは結果として順序違反ではない．一方，同下図では 2 つの SSNT からリードされる値に対して，共に順序違反を示すので，順序違反が検出される．このようにして，ハッシュ値の偶然の衝突が起きたとしても，その他のフィルタにおいて衝突

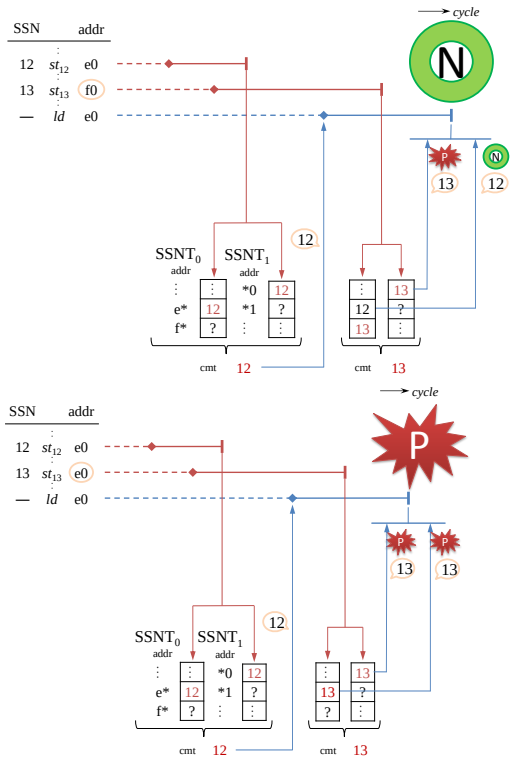

図 6 複数の SVW を用いた手法の順序違反検出
順序違反がない場合（上）とある場合（下）

が起きていなければ，偽陽性は生じない．これにより，提案手法はハッシュ値の偶然の衝突による偽陽性を下げるができる．またフォワーディング・ミスにおいても SVW と同様に検出を行えば，上記の機構によって偽陽性を低く実現可能である．

5. 評価

提案手法において用いるハッシュ関数の数と IPC および偽陽性率について評価を行った．そして我々の提案している PCBF による順序違反検出手法と，比較評価を行った．

5.1 評価環境

ベンチマーク

ベンチマークは SPEC CPU 2006 [12] 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 [13] 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 [5] を用いた．

5.2 IPC と偽陽性率の評価

本節では，提案手法のハッシュ関数の数 k の効果を評価する．

また評価において提案手法特有のパラメタは，SSN は 16 ビットのシーケンス・ナンバ，SPCT のエントリ数は我々の事前評価において最も良い性能を示した 16 エントリとした．またサイズの問題に対しては，SVW と同様に 8B ワード単位の検出を行うモデルを用いている．またロード再実行による確認検査は，陽性が検出された直後の 1 サイクルで可能であるとし，バックエンドを 2 サイクル，ストールすることによって完了できると理想化した．

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

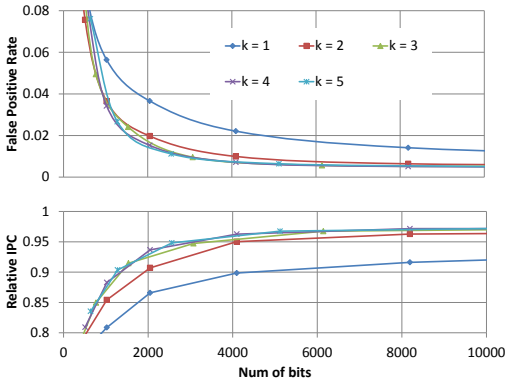

図 7 k の効果:偽陽性率 (上) と相対 IPC(下)

PCBF による順序違反検出手法では，ハッシュ関数の数 k を僅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確認されている [3]．ここでは，同様にして提案手法

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

においても，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，その結果を示す．曲線はそれぞれ $k = 1, 2, \ldots, 5$ の場合で，$SSNT_k$ のエントリ数をそれぞれ，$8, 16, 32, \ldots$ と変化させたものである．$k = 1$ 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ことが分かる．また，$k = 4$ において，フィルタの総ビット数を増やした時に，提案手法における相対 IPC の低下率は 3 %程度に収まった．

5.3 PCBF との比較

本節では，我々がかつて提案した PCBF による順序違反検出手法と比較する．PCBF を用いた手法と本稿で提案する手法には，サイズの問題への対応に大きな違い有る．PCBF を用いた手法では，1B，2B，4B，8B アクセスにも対応しており，サイズの問題によって生じる偽陽性の影響を小さくしている．詳しい説明に関しては文献 [3] を参照していただきたい．

評価モデル

PCBF は次の二つのモデルを考えた．

- 4B ワード単位で順序違反検出が可能な PCBF
- 1B ワード単位で順序違反検出が可能な PCBF

上記モデルに関する特有のパラメタや，具体的な実現方法については文献 [3] に記載されている値に従った．また，この 2 つのモデルは [3] において，最もよい性能を表したものをプロットすることにする．また我々の手法について用いるハッシュ関数の個数は文献 [3] に合わせ $k = 4$ とした．

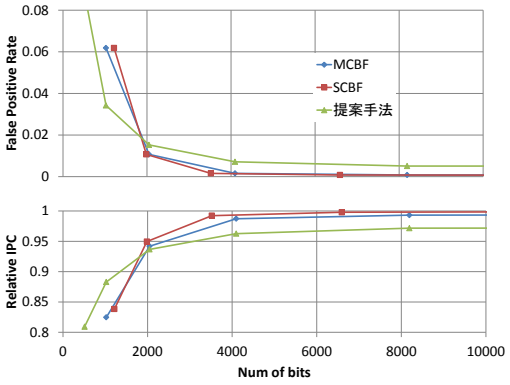

図 8 それぞれの手法の偽陽性率 (上)，相対 IPC(下)

図 8 は 2 つのモデルと提案手法を評価したものである図 8 中において，MCBF としてプロットされているものが 4B ワード単位で検出を行う PCBF，SCBF としてプロットされているものが 1B ワード単位で検出を行う PCBF にあたる．上記モデルのうち，1B 単位でサイズの問題に対応したモデルである，SCBF が最も良い性能を示していることが分かる．また，提案手法は他の 2 つの手法と比べ偽陽性率は高い．これはサイズの問題により，隣接するアドレスのおける偽陽性が原因であると考えられる．

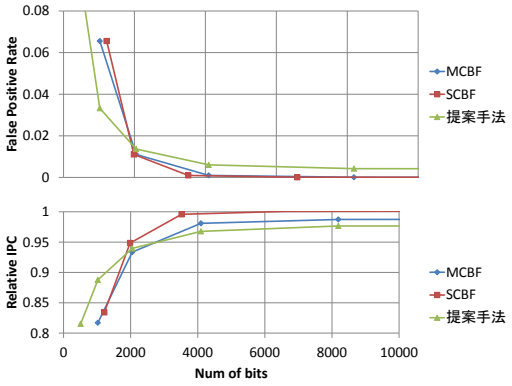

図 9 サイズの問題を除いたときのそれぞれの手法の偽陽性率 (上)，相対 IPC(下)

そこで，次に 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 を除いたベンチマークの平均をとった評価をしたグラフが**図 9** である．このグラフは，提案手法の偽陽性率は他の手法と比べ，依然として高かった．

6. おわりに

本稿では，最新のプロセッサにおける LSQ がコストの高いロジックの一つであることを説明した．そして，LSQ の高いコストは CAM と呼ばれるロジックによる構成が原因であり，LSQ の CAM を RAM で構成されるフィルタによって置き換えることで，LSQ のコストを下げる手法を幾つか紹介した．本稿では，シーケンス・ナンバで構成されているフィルタである SVW において，複数のハッシュ関数を用いた手法を提案した．そして，評価においては，複数のハッシュ関数を用いることで，低い偽陽性率を実現できることを示した．また，当研究室で提案している PCBF を用いた手法と比べ，高い偽陽性を示していた．現在，原因を調査中である．

参考文献

[1] Sinharoy, B., Van Norstrand, J., Eickemeyer, R., Le, H., Leenstra, J., Nguyen, D., Konigsburg, B., Ward, K., Brown, M., Moreira, J., Levitan, D., Tung, S., Hrusecky, D., Bishop, J., Gschwind, M., Boersma, M., Kroener, M., Kaltenbach, M., Karkhanis, T. and Fernsler, K.: IBM POWER8 processor core microarchitecture, *IBM Journal of Research and Development*, Vol. 59, No. 3, pp. 2:1–2:21 (2015).

[2] Hammarlund, P., Martinez, A. J., Bajwa, A. A., Hill, D. L., Jiang, E. H. H., Dixon, M., Derr, M., Hunsaker, M., Kumar, R., Osborne, R. B., Rajwar, R., Singhal, R., ReynoldD 'Sa, Chappell, R., Kaushik, S., Chennupaty, S., Jourdan, S., Gunther, S., Piazza, T., Burton, T.：Haswell: The Fourth-Generation Intel Core Processor, *Micro, IEEE*, Vol. 34 (2014).

[3] Kurata, N., Shioya, R., Goshima, M. and Sakai, S.: Address Order Violation Detection with Parallel Counting

Bloom Filters, *IEICE Trans. on Information and Systems* (2015).

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

[5] Chrysos, G. Z. and Emer, J. S.: Memory Dependence Prediction Using Store Sets, *25th International Symposium on Computer Architecture (ISCA'98)*, pp. 142–153 (1998).

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

[7] Martin, M. and Roth, A.: Scalable Store-Load Forwarding via Store Queue Index Prediction, *38th International Symposium on Microarchitecture (MICRO'05)*, pp. 159–170 (2005).

[8] Sha, T., Martin, M. M. K. and Roth, A.: NoSQ: Store-Load Communication Without a Store Queue, *39th International Symposium on Microarchitecture (MICRO'06)*, pp. 285–296 (2006).

[9] Castro, F., Pinuel, L., Chaver, D., Prieto, M., Huang, M. and Tirado, F.: DMDC: Delayed Memory Dependence Checking through Age-Based Filtering, pp. 297–308 (2006).

[10] 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).

[11] 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機構，情報処理学会研究報告 2014-ARC-212，No. 17, pp. 1–15 (2014).

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

[13] 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).