Vol.2016-ARC-219 No.15
Vol.2016-SLDM-175 No.15
Vol.2016-EMB-40 No.15
2016/3/24

フィルタを用いたメモリ・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いる．我々はこれまでの提案で，偽陽性率の低いブルーム・フィルタの特性を応用した，パラレル・カウンティング・ブルーム・フィルタを用いる手法や， Bloom-like SVW を提案してきた．本稿では，フィルタによる実行順序違反検出手法である，Delayed Memory Dependence Checking に対して改良案を提案する．評価では，実装が間に合った Bloom-like SVW に SQ-CAM 簡略化手法を適用した時の IPC を測った．この評価では，2% の IPC 低下に収まった．

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のための，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** を用いた構成の LSQ 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，in-flight 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り，ポート数の二乗に比例して CAM の面積は大きくなる．

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** が，IBM POWER8 プロセッサのチップ写真であるが [1]，この図から見て分かるように Load Store Unit (LSU) はコアの 1/4 程度の領域を占める大きなものとなっている．また**図 2** は Intell Haswell プロセッサにおける L1D の面積と LSQ の CAM の面積を CACTI [3] によって算出したものである．同図による評価から，LSQ の CAM は L1D に匹敵するほどの大きな面積になっていることが分かる．

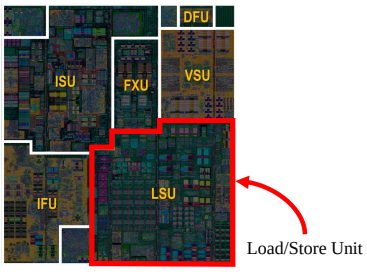

図 1　POWER8 のチップ写真 [1]

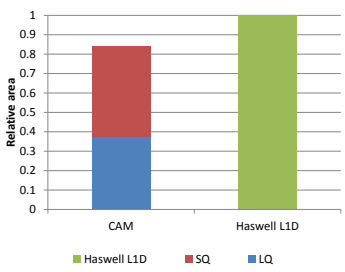

図 2　L1D と LSQ の CAM の面積 [1]

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

Vol.2016-ARC-219 No.15
Vol.2016-SLDM-175 No.15
Vol.2016-EMB-40 No.15
2016/3/24

フィルタを用いた順序違反検出

LQ の CAM を排除する手法に RAM で構成されるフィルタを用いて，ロード/ストアの先行制約を守る手法がいくつか提案されている [4], [5], [6], [7], [8]．これらの手法は**フィルタに要素を登録/参照/消去**を行い，メモリ・アクセス順序違反違反を検出することに大きな特徴がある．検出に用いられるフィルタは，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．我々は，偽陽性が低い**ブルーム・フィルタ (BF)**[9] の特性を応用した，**パラレル・カウンティング・ブルーム・フィルタ (PCBF)** を用いた手法 [7] や，**Bloom-like SVW** [8] を提案している．

投機フォワーディング

SQ CAM の排除は，投機的にフォワーディングによって実現可能である．投機フォワーディングでは，ロード命令とストア命令の依存関係を予測することで，アドレス比較のない，つまり SQ-CAM を用いないフォワーディングを実現してる．投機フォワーディングの性能は，ロード命令とストア命令の依存関係を予測する依存予測器の精度の高さによって決まる．依存予測器には，ロード命令と過去に依存関係のあったストア命令を覚えるストア・セット依存予測器 [10] や，過去に依存関係にあったロード命令とストア命令の動的な命令間距離を学習する距離ベースの依存予測器 [11], [12] がある．

本稿の構成は以下のようになる．2 章でロード/ストア命令のメモリアクセスのタイミングを述べ，フィルタを用いた順序違反検出手法である，Bloom-like SVW，Delayed Memory Dependence Checking(DMDC)[6] について簡単に説明する．2 章では，DMDC に投機フォワーディングを適用するための改良案を示している．4 章で，フィルタを用いる手法に投機フォワーディングを適用した時の簡単な IPC の評価を行い，3 章で本稿をまとめる．

2. フィルタによる メモリ順序違反検出

ロード/ストア命令の先行制約を守るためには，同一アドレスへの実行順序の逆転を検出する必要がある．アドレスの一致比較を行う CAM を用いた確認検査手法では正確にメモリ・アクセス順序違反を検出できるが，面積の増加と，それに伴う消費電力の増加といった問題がある．そこで，メモリ・アクセス順序違反を引き起こさないと判明した命令をフィルタリングすることで，メモリ・アクセス順序違反の確認検査の頻度を減らすフィルタを用いた手法が有る．

フィルタを用いた手法ではロード/ストア命令がフィルタに要素を登録/参照/削除することで，メモリ・アクセス順序違反を生じた可能性のある命令を検出する．この判定には，偽陽性はあるが，偽陰性はない．正しい依存関係の学習のために，フィルタによる検出手法では，実際に実行順序違反が発生したかの確認検査を行う手法がある．偽陽性による確認検査は IPC 低下が伴うため，フィルタによる検出手法では，偽陽性率を下げることが重要である．

図 3 一般的なパイプライン・チャート (上) と
本稿で用いるパイプライン・チャート (下)

順序違反検出を行うフィルタには，Store Vulnerability Window[5], パラレル・カウンティング・ブルーム・フィルタ [7]，そして Delayed Memory Dependence Checking[6] がある．特に，PCBF や，Bloom-like SVW では，BF の特性を用いることで，低い偽陽性が実現できることを示してきた [7], [8].

2.1 ロード/ストア命令のメモリ・アクセス

out-of-order 実行において，命令の実行は out-of-order に行われるが，コミットは in-order に行われる．コミットとは，レジスタやメモリなどのアーキテクチャ・ステートの不可逆的更新の事である．ロード命令は実行時にアドレス計算と，L1D に値を読み込み，コミット時はメモリに関しては何も行わない．ストア命令は実行時にはアドレス計算のみを行い，コミット時に L1D に書込み，つまりステートの更新を行う．

ここで，ロード命令とストア命令においてメモリアクセスのタイミングが違うことに注意をしていただきたい．依存関係にあるロード命令の実行と，ストア命令のコミット (もしくは実行) のタイミングが入れ替われば，正しいストア・データをロードできない，つまりメモリ・アクセス順序違反である．

そのため，順序違反検出において意味があるのは，実行ステージとコミットステージであるので，本稿では**図 3** 上図の一般的なパイプライン・チャートではなく，図 3 下図のようなパイプライン・チャートを用いる．このチャートでは，ストア命令を st，ロード命令を ld，実行ステージを菱型，コミットステージは縦棒，フェッチステージから実行ステージまでを破線部，実行ステージからコミットステージまでを実線部で表している．

2.2 Bloom-like SVW

Store Vulnerability Window[5] は，ロード/ストア命令がフィルタに要素を登録/参照をすることで，先行するストア命令のコミットを追い越して実行されたロード命令を

Vol.2016-ARC-219 No.15
Vol.2016-SLDM-175 No.15
Vol.2016-EMB-40 No.15
2016/3/24

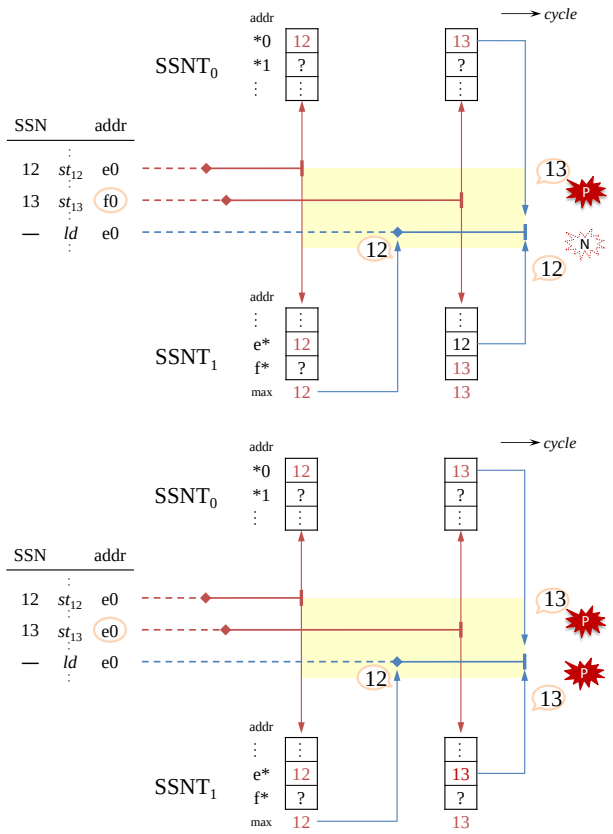

図 4 Bloom-like SVW の順序違反検出：
順序違反がない場合（上）とある場合（下）

検出するフィルタである．我々はかつて，SVW に対して複数ハッシュ関数を適用することで，偽陽性率を低下させる Bloom-like SVW を提案した．

SVW では，ストア命令にフェッチ順に割り当てた，Store Sequence Number(SSN) とよばれるシーケンス・ナンバが要素であり，フィルタは SSNT である．SVW でのフィルタへの要素の登録/参照手順は，

(1) ストア命令がコミット時に，フィルタの該当エントリへ自身の SSN を登録する．

(2) ロード命令が実行時に，フィルタに書き込まれている最大の要素を参照する．

(3) ロード命令がコミット時に，フィルタの該当エントリから要素を読込む．

の 3 つの手順からなっており，(2) と (3) で読み込まれる要素の値の大小によって，実行順序違反を引き起こし得るロード命令を検出する．この大小比較によって検出された命令が，実際にメモリ・アクセス順序違反を引き起こしているかの確認検査は，ロード再実行によって行われる．

Bloom-like SVW は，SSNT を複数用意して，要素の登録/参照を行う．Bloom-like SVW は，複数のフィルタによる検出が全て陽性のときのみに，実行順序違反を検出するため，偽陽性率を低くすることができる．

順序違反検出

図 4 の $SSNT_0$ はアドレスの右側の桁を，$SSNT_1$ はアドレスの左側の桁をハッシュ値として SSN を記録するテーブルであり，max は SSNT に書き込まれている最大値，つまり最後にコミットしたストア命令の SSN である．

図 4 の上図は，順序違反が生じていない例である．SVW では，ロード命令がコミット時に読み込む値が，実行時に読み込む値より大きい時，ストア命令のコミットより先に実行された可能性を意味する．

例えば，$SSNT_0$ では，st_{13} と ld において，右側の桁が等しいことによるハッシュ値の衝突が起きている．これにより，ロードが実行時とコミット時に $SSNT_0$ から読み込む値には，大小関係が生じる．そのため，実際には順序違反が起きていないのに，$SSNT_0$ では陽性を，つまり偽陽性を返す．

一方で，アドレスの左側の桁は異なっているため，ロードが実行時とコミット時に $SSNT_1$ から読み込む値は等しくなる．そのため，$SSNT_1$ は陰性を返す．このとき，Bloom-like SVW は，$SSNT_1$ で陰性であるため，全体として陰性を返す．

図 4 の下図は，st_{13} と ld のアドレスが一致しているため，順序違反を起こしている例である．このとき，アドレスの右側の桁もアドレスの左側の桁も一致しているため，st_{13} が実行時に書き込むエントリと，ld がコミット時に読み込むエントリは，$SSNT_0$，$SSNT_1$ でともに一致する．そのため，$SSNT_0$ と $SSNT_1$ の両方で陽性を示すため，Bloom-like SVW は全体として陽性を返す．

2.3 Delayed Memory Dependence Checking

Delayed Memory Dependence Checking(DMDC) では 2 段のフィルタに要素を登録することで，メモリ・アクセス順序違反を検出する手法である [6]．1 段目のフィルタの要素は，Load Store Number (LSN) と呼ばれるシーケンス・ナンバで，2 段目のフィルタの要素はビットである．

DMDC におけるフィルタへの登録/参照/削除の手順は

(1) ロード命令が実行時に， 1 段目のフィルタの該当エントリへ自身の LSN を登録．

(2) ストア命令が実行時に， 1 段目のフィルタから該当エントリの要素を参照．

(3) 1 段目のフィルタにより検出されたストア命令のみが実行時に， 2 段目のフィルタの該当エントリへ要素を登録

(4) ロード命令のコミット時に，2 段目のフィルタから該当エントリの要素を参照．

(5) 1 段目のフィルタで検出されたストア命令を追い越し得るロード命令が，すべてリタイアした時，2 段目フィルタの全ての要素を削除．

の 5 つの手順からなる．1 段目フィルタの (1) と (2) とで読み書きされた値の大小関係を比較することで，後続の

Vol.2016-ARC-219 No.15
Vol.2016-SLDM-175 No.15
Vol.2016-EMB-40 No.15
2016/3/24

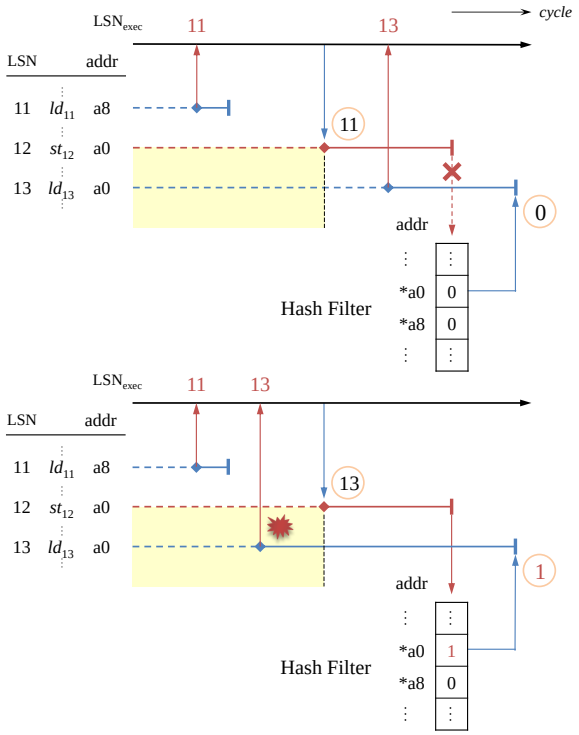

図 5 DMDC における順序違反検出:
順序違反がない場合（上）とある場合（下）

ロード命令の実行に遅れて実行されたストア命令を検出する．そして，1 段目フィルタで検出されたストア命令のみが 2 段目のフィルタに書込 (登録) を行い，その書込先を読み込んだロード命令がメモリ・アクセス順序違反を起こし得る命令として検出される．そして，2 段目のフィルタの要素の削除は，(5) のタイミングで行われる．

順序違反検出

図 5 は，上下図共に，st_{12} と ld_{13} に依存関係が有る．同上図では，st_{12} の実行の後に，ld_{13} が実行されているので，実行順違反は発生していない．同上図では，st_{12} は実行時に 1 段目のフィルタに書かれている 11 を読込む．フィルタから読み込んだ値より，st_{12} の実行より先に実行されたロード命令が存在しないため st_{12} は 2 段目のフィルタへの書込は行わない．ld はコミット時に 2 段目のフィルタから 0 を読み込むため，実行順序違反は検出されない．

一方，同下図では，ld_{13} の実行が早まることによって st_{12} と ld_{13} でメモリ・アクセス順序違反が生じている図である．実行が早まったことで，st_{12} が 1 段目のフィルタから読み込む値は，13 となる．これは，先行する st_{12} の実行よりも，先に実行したロード命令が存在することを意味する．そのため，1 段目のフィルタは st_{12} を実行順序違反され得る命令として検出する．1 段目のフィルタで検出された st_{12} は，2 段目のフィルタの自身の該当エントリに，書込を行う．ld_{13} は実行時に 2 段目のフィルタから st_{12} が書き込んだ値を読込むので，これを実行順序違反として検出する．

DMDC の問題

DMDC では，ストア命令の実行とロード命令のコミットの入れ替わりを，1 段目のフィルタで検出している．しかし，この検出は SQ-CAM を用いたフォワーディングを前提である．もし，DMDC で SQ-CAM の排除を試みようとするならば，ストア命令のコミットとロード命令の実行の入れ替わりを検出するように 1 段目のフィルタを変更しなければならない．

このようにフィルタへと変更を加えた時，ストア命令からフォワーディングが行われたロード命令は，そのストアのコミットを追い越す．そのため，1 段目のフィルタでフォワーディングを行ったストア命令が必ず検出されてしまう．当然，アドレスも一致するので，フォワーディングされたロード命令は 2 段目フィルタで，実行順序違反を引き起こしうる命令として検出されてしまう．

このように，投機フォワーディングを適用しようと思うと，フォワーディングに関わった命令は全て実行順序違反を引き起こしうると判定されてしまい，フィルタの偽陽性発生率が大きくなってしまう．DMDC において，フォワーディングを検出しないようなフィルタの制御については未だに知られていない．

3. DMDC の改良

DMDC で投機フォワーディングを可能にするために，1 段目のフィルタで，ストア命令のコミットとロード命令の実行の入れ替わりを検出するように変更をする．また，実行からコミットに変更を加えたことで，1 段目のフィルタへの書込が増加して，偽陽性が増えることが予想される．これは，1 段目のフィルタを BF にすることで緩和可能である．

フォワーディングの処理

改良した DMDC でフォワーディングされたロード命令を検出しないようにするには，パラレル・カウンティング・ブルーム・フィルタ (PCBF) を追加すれば良い．このフィルタは，
(1) フォワーディングを行ったストア命令が 2 段目のフィルタへ登録し，更に PCBF へ要素を登録 (インクリメント)
(2) コミットされるストア命令が PCBF を参照
(3) フォワーディングされたロード命令は，2 段目のフィルタを参照せず，PCBF から要素を削除 (デクリメント)
するように制御すれば良い．下記の例では，BF を用いて動作の説明を行う．

図 6 の上下図共に，st_{11} から ld_{13} にフォワーディングが行われている．

同上図では，st_{11} と ld_{13} の間にある，st_{12} は，アドレスが一致していないので，フォワーディングが成功している．st_{11} はコミット時に，2 段目のフィルタに書込を行う

Vol.2016-ARC-219 No.15
Vol.2016-SLDM-175 No.15
Vol.2016-EMB-40 No.15
2016/3/24

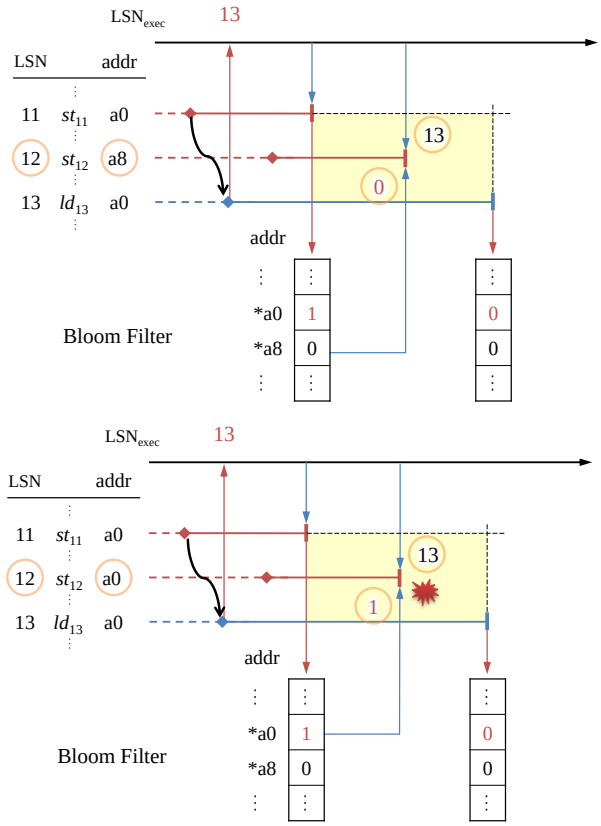

図 6 DMDC におけるフォワーディングの処理：
正しいフォワーディングの場合（上）と誤ったフォワーディングの場合（下）

と同時に，追加した BF にも書込を行う．st_{12} はコミット時に，BF の読込を行うが，要素が登録されていないため，無事にコミットされる．そして，フォワーディングされた ld_{13} はコミット時に，2 段目のフィルタを参照せずに，st_{11} が登録した要素を削除する．

一方で，同下図では，st_{11} ，ld_{13} と st_{12} のアドレスが一致し，$st11$ が誤ったフォワーディングを行っている．st_{12} は，本来の依存関係にないフォワーディングが行われたことを，コミット時に BF を読み込むことで検出できる．

4. 評価

本章では，Bloom-like SVW に投機フォワーディングを適用した時の評価を行う．3 章 で提案した DMDC の改良手法や PCBF を用いた手法については，実装が間に合わなかったため，評価が行えなかった．

ベンチマークは SPEC CPU 2006 [13] 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 [14] を用いた．ベースラインとなるプロセッサの構成は**表 1** 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，

評価モデル

本評価での投機フォワーディングは，SQIP [15] をベースとしており，アドレスが一致しないかぎり投機フォワーディングを行わない．また，評価したモデルは以下のとおりである．

- 距離ベースの依存予測器を用いた Bloom-like SVW の投機フォワーディング．但し，予測に用いる分岐履歴長を 0〜6 ビットと変化させ，それぞれに応じた 7 つのモデルを用意した．
- Store Set 依存予測器を用いた Bloom-like SVW の投機フォワーディング

ここで，用いる Bloom-like SVW は，LQ-CAM を用いて順序違反検出を行うモデルと比べて，性能低下が全ベンチマーク平均で 0.5% 程度のごく僅かなパラメタを用いた．但し，距離ベースの依存予測器を用いたモデルでは，予測テーブルの数を分岐履歴毎に持つようにした．例えば，分岐履歴長を 2 ビット用いるならば，予測テーブルの数は 4 つにするといった具合である．また，Store Set 依存予測器を用いたモデルでは，予めパラメタ・チューニングを行い最良な結果を示したものを用いている．それぞれの依存予測器の細かなパラメタを**表 2** にのせる．

また評価におけるベースラインは，CAM による実行順序違反検出とフォワーディングを行うモデルである．

評価結果

評価結果を**図 7** に示す．同図の凡例において，GHTx が距離ベースの依存予測器のモデルである．このモデルでは，GHTx の x が分岐履歴長にあたる．また SS が Store

表 1 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 2 依存予測器 の構成

Parameter	Value
Store Set	
SSID Table	4K entries/ 8way
LFST Table	512 entries/ 1way
距離ベースの依存予測器	512 entries/ 8way

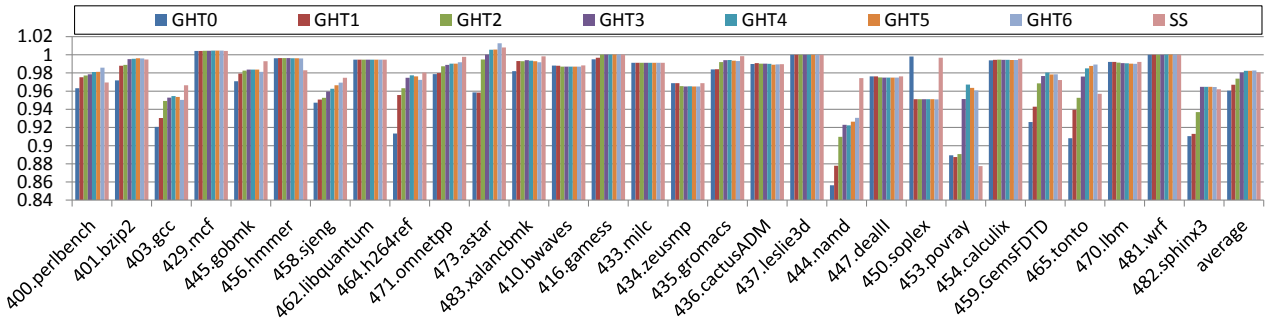

図 7 ベースラインに対する相対 IPC

Set 依存予測器を用いたモデルである．同図横軸は，SPEC CPU 2006 の各々のベンチマークにあたり，縦軸が相対 IPC である．

同図からは，投機フォワーディングの適用による性能 (相対 IPC) 低下は全ベンチマークを平均して 2%〜5% に抑えらている．特に，GHT3〜GHT6 と SS では，2% 程度に収まる IPC 低下であった．また，距離ベースの依存予測器では，分岐履歴長を長くすればするほど，相対 IPC が高くなる傾向が有ることがわかる．

また，gcc, zeusmp, namd, soplex において，Store Set 依存予測器を用いるモデルが良い性能を示している．

5. 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，LQ-CAM の簡略化手法であるフィルタによる順序違反検出手法を幾つか紹介し，DMDC の改良案を示した．それから，フィルタによる順序違反検出手法の 1 つである Bloom-like SVW に対して，投機フォワーディングを適用した時の相対 IPC を評価した．この評価では 2% の性能低下に収まった．

今後は，実装の間に合わなかった DMDC の改良手法の詳細評価を行い，容量効率に優れるフィルタを示したい．

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] Thoziyoor, S., Muralimanohar, N., Ahn, J. and Jouppi, N.: CACTI 5.1., Technical report, HP Laboratories (2008).

[4] Sethumadhavan, S., Desikan, R., Burger, D., Moore, C. R. and Keckler, S. W.: Scalable Hardware Memory Disambiguation for High ILP Processors, *36th International Symposium on Microarchitecture (MICRO'03)*, pp. 399–410 (2003).

[5] 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).

[6] 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).

[7] 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).

[8] 西川卓，塩谷亮太，入江英嗣，五島正裕，坂井修一：Bloom-like SVW の評価 (コンピュータシステム) – (萌芽的コンピュータシステム研究展示会)，電子情報通信学会技術研究報告 = IEICE technical report : 信学技報，Vol. 115, No. 243, pp. 27–34 (2015).

[9] 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).

[10] 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).

[11] 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).

[12] Subramaniam, S. and Loh, G. H.: Fire-and-Forget: Load/Store Scheduling with No Store Queue at All, *39th International Symposium on Microarchitecture (MICRO'06)*, MICRO 39, pp. 273–284 (online), DOI: 10.1109/MICRO.2006.26 (2006).

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

[14] 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).

[15] 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).