Vol.2012-ARC-201 No.13
2012/8/1

データ・キャッシュ・ミスの周期的発生を利用したヒット/ミス予測器

山田 亮輔[1]　塩谷 亮太[1]　安藤 秀樹[1]

概要：スーパスカラ・プロセッサにおいて高い性能を得るためには，生産者の結果が得られるタイミングで消費者がそれを受け取る動的命令スケジューリングを行わなければならない．このために，発行キューでは，生産者の発行サイクルから実行レイテンシの経過後に消費者が発行されるよう制御が行われる．しかし，ロード命令では，その実行レイテンシがキャッシュのヒット/ミスによって変化するため，消費者の発行動作の開始タイミングを事前に決めることが難しい．そこで，キャッシュのヒット/ミスを予測することでレイテンシを予測し，それに基づいて命令を投機的に発行することが広く行われている．このために様々なヒット/ミス予測器が提案されているが，多くの予測器では静的なロードごとにヒット/ミスがどちらに偏っているかに着目しており，短期的なヒット/ミスの変化にうまく対応することができない．我々は，そのような短期的なヒット/ミスの変化が周期的に起こることに着目し，それに基づいた予測器を提案する．提案手法を実装したシミュレータで評価を行ったところ，SPEC CPU2006 ベンチマークにおいて，飽和型カウンタを用いる方式と比べて予測精度を最大で 8.0 ポイント，IPC を最大で 4.9%改善できることを確認した．

1. はじめに

out-of-order 実行スーパスカラ・プロセッサでは，性能向上のために動的命令スケジューリングを行っている．高い性能を得るためには，生産者の結果が得られるタイミングで消費者がそれを受け取れるようスケジューリングしなければならない．このために，発行キューでは，生産者の発行サイクルから実行レイテンシの経過後に消費者が発行されるよう動作の制御が行われる．しかし，ロード命令では，その実行レイテンシがキャッシュのヒット/ミスによって変化するため，依存先命令の発行タイミングを事前に決めることが難しい．そこで，キャッシュのヒット/ミスを予測することでレイテンシを予測し，それに基づいて命令を投機的に発行することが広く行われている [1,5,6,8]．

命令の発行は投機的に行われるため，キャッシュ・ヒット/ミスが予測結果と異なった場合にはペナルティが発生し，性能に大きな影響を与える．後の 2.4 節で詳しく述べるように，このペナルティは非常に大きく，最大で 8 命令から 16 命令程度の実行が取り消される [2,4]．

これまでに様々な予測器が提案されているが，それらの多くは静的な命令ごとのキャッシュ・ヒット/ミスの偏りに基づいて予測を行うものである．最も簡単な方法は，常にキャッシュ・ヒットすると予測することである (以降，**静的方式**と呼ぶ)．通常，多くのロード命令はキャッシュにヒットするため，静的にヒットすると予測してもある程度高い予測精度を達成できる．この方法は MIPS R10000 [7] や Intel Pentium 4 [3] などの多くのプロセッサで採用されている．より進んだ方法として，飽和型カウンタを用いて静的な命令ごとにヒット/ミスの履歴を記録し，それに基づいて予測を行う方法がある (以降，これを**飽和型カウンタ方式**と呼ぶ)．この方法は DEC Alpha 21264 [2,4] で採用されている．

これらの手法では，静的な命令ごとのヒット/ミスが長期間変化しない場合はうまく働くものの，短期間にヒット/ミスが変化する場合はそれをうまく予測することができない．このような短期間の変化は，例えば，メモリ・アクセスがキャッシュ・ラインの境界をまたぐ場合に現れる．最も単純な例は，シーケンシャルにメモリへアクセスした場合，アドレスが新しいラインに差し掛かるたびに生じるミスである．このような場合，静的な命令ごとのヒット/ミスの偏りに基づいて予測する方法では，すべてヒットと予測してしまう．また，静的方式ではパーシャル・ヒット発生時にもペナルティが発生する問題がある．一般に，メモリ・アクセスには局所性があるため，一度アクセスされたラインは短時間のうちに再度アクセスされることが多い．

[1] 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University

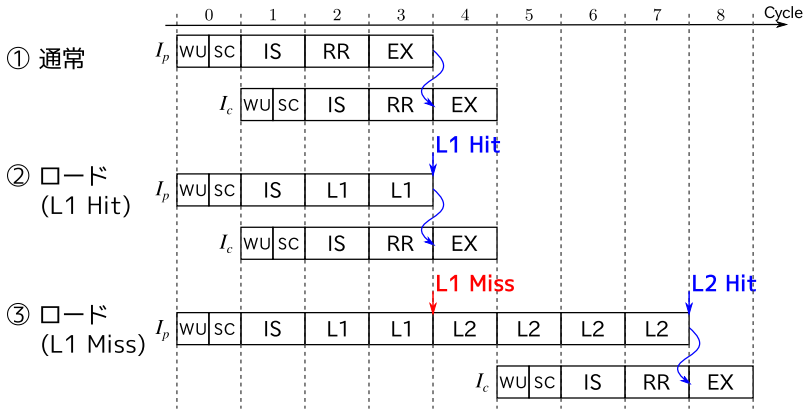

図 1 実行コアにおける生産者 I_p と消費者 I_c の実行の様子

このため，キャッシュ・ミスによってラインがフィルされる際，後続の命令がフィル中のラインに対してアクセスを行うことが多く発生する．パーシャル・ヒットしたロード命令に依存している命令は，そのフィルが終わるまでは発行することができないため，これらのロード命令をヒットと予測してスケジューリングするとペナルティが発生してしまう．したがって，静的方式では，単純なキャッシュのミス率以上の割合でペナルティが発生する．

これに対し，我々はヒット/ミスの周期的な変化に着目した予測器を提案する．後の 4 節で詳しく述べるように，前述のラインを跨いだアクセス際に発生するミスや，パーシャル・ヒットの発生は周期的に繰り返される性質を持つ．提案手法ではこれに着目し，周期的な振る舞いを予測することで，高精度な予測を行う．

本論文の構成は以下の通りである．まず，2 節でロードに依存した命令の投機的発行について説明する．3 節では既存の予測器について述べ，4 節では既存の予測器の問題点について述べる．5 節では提案手法について述べた後，6 節で評価を行い，7 節でまとめる．

2. ロードに依存した命令の投機的発行

本節では，キャッシュ・ヒット/ミス予測の背景として，ロードに依存した命令の発行について述べる．

2.1 ロードに依存した命令の発行タイミング

スーパスカラ・プロセッサでは，生産者の発行サイクルから実行レイテンシの経過後に消費者が発行されるよう制御が行われる．しかし，ロード命令では，その実行レイテンシがキャッシュのヒット/ミスによって変化するため，消費者の発行動作の開始タイミングを事前に決めることが難しい．

このことを図 1 を用いて説明する．同図は実行コアにおける生産者 I_p と消費者 I_c の実行の様子である．図内の①は I_p がロード命令以外の通常の命令の場合を表す．この命令の実行レイテンシは 1 サイクルで固定である．②と③は I_p がロード命令であり，それぞれ L1 データ・キャッシュにヒットした場合とミスした場合を表す．L1 データ・キャッシュと L2 キャッシュのレイテンシはそれぞれ 2 サイクルと 4 サイクルである．図上のラベルの意味については表 1 に示す通りである．

表 1 パイプライン・ステージの略称

WU	ウェイクアップ
SC	セレクト
IS	発行
RR	レジスタ読み出し
EX	実行
L1	L1 データ・キャッシュ・アクセス
L2	L2 キャッシュ・アクセス

通常の命令の場合

I_p がロード以外の命令の場合，図 1 上段の①のように，I_p は第 0 サイクルにセレクトされた後，発行とレジスタ読み出しを経て実行され，第 3 サイクルに実行結果が得られる．I_c を 1 サイクル遅れてウェイクアップすることにより，I_p の実行結果が使える最も早いタイミングである第 4 サイクルに実行することができる．

ロード命令が L1 ヒットする場合

ロード命令 I_p はアドレス計算を行い，ロード/ストア・キュー (LSQ: Load/Store Queue) で先行するストア命令との依存関係を調べる．依存がなければ LSQ から発行され，L1 データ・キャッシュにアクセスする．キャッシュにヒットすれば，図 1 中段②のようにロード・データは第 3 サイクルに得られる．このとき，I_c が第 1 サイクルにウェイクアップおよびセレクトされていれば，ロード・データを第 4 サイクルに受け取り，実行することができる．

ロード命令が L1 ミスする場合

図 1 下段③のように，ロード命令 I_p がキャッシュにミスした場合には，第 3 サイクルの時点ではロード・データが得られず，I_c を実行することができない．このような場合，最も単純な方法としてはキャッシュからデータが得られるまでパイプライン中でストールさせる方法があるが，

この方法では後続の依存のない命令もパイプラインを進むことができず，性能への影響が大きい．そのため，必要なデータが得られるタイミングまで消費者のウェイクアップを遅らせることが一般的である．図 1 の場合，I_c のウェイクアップを第 5 サイクルに行っている．

2.2 ロードに依存した命令の投機的発行

前節で説明したように，ロード命令の消費者は，キャッシュのヒット/ミスによってその最適な発行タイミングが変化する．このため，ロードに依存した命令の発行は，通常，ロードのキャッシュのヒット/ミスを予測し，それに基づいて投機的に行われる．以下では，まず非投機的に発行を行った場合について説明した後，投機的な発行の方法を説明する．

2.2.1 非投機的に発行を行った場合

キャッシュのヒット/ミスが判明した後に消費者のウェイクアップを行った場合，消費者を最速のタイミングで発行することはできない．図 1 に示したタイミングの場合，ヒット/ミスが第 3 サイクルに判明するため，第 4 サイクルまで消費者のウェイクアップを行えない．その結果，ロード命令の実質的なレイテンシは最短でも 5 サイクルとなり，性能を大きく低下させてしまう．

2.2.2 ヒット/ミス予測に基づく投機的発行

このような性能低下を避けるため，多くのプロセッサではキャッシュのヒット/ミス予測に基づいた投機的な発行を行っている [1,5,6,8]．これらのプロセッサでは，キャッシュのヒット/ミスを予測することでレイテンシを予測し，それに基づいてロード命令の消費者を投機的に発行する．

ロード命令がキャッシュ・ヒットすると予測した場合，図 1 中段②のように I_c を第 1 サイクルにウェイクアップする．一方，ミスすると予測した場合は L2 キャッシュにはヒットすると予測し，図 1 下段③のように I_c を第 5 サイクルにウェイクアップする．予測通りに I_p がキャッシュにヒットないしミスした場合，それぞれ図 1 の②ないし③の通りに発行が行われ，実行を継続する．

2.3 予測誤り時の動作

ヒット/ミス予測の結果が誤っていた場合の動作は，ミスと予測して実際にはヒットした場合と，その逆では異なる．

ミスと予測して実際にはヒットした場合は単純である．この場合，キャッシュ・ヒットが判明した時点で消費者のウェイクアップを開始すればよい．

これに対して，ヒットと予測して実際にはミスした場合の動作は複雑である．投機的に発行された消費者は生産者の実行が終了していないため，ロード・データを受け取れない．この状態からの回復にはいくつかの方法があるが，次節で述べるように，命令を再発行することが一般的である．

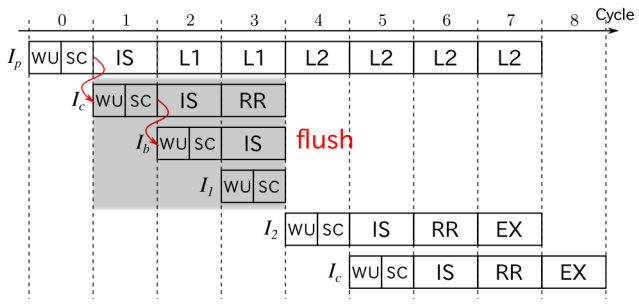

図 2 命令の再発行

2.3.1 命令の再発行

命令の再発行とは，ロード命令以降に発行され，まだ実行されていない命令の発行を取り消し，再度スケジューリングをやり直す手法である [1,2,4,5,7]．発行を取り消した後，消費者は L1 データ・キャッシュからデータが得られるタイミングに合わせて再スケジューリングされる．図 2 に，命令の再発行の様子を示す．同図の I_p はロード命令であり，I_c は I_p に，I_b は I_c にそれぞれ依存している．また，I_1，I_2 はいずれの命令にも依存していない．同図では，I_p のキャッシュ・ミスが第 3 サイクルに発覚すると，それ以降に発行された命令の発行を取り消している．その後，第 5 サイクルに L2 キャッシュからロード・データが得られるタイミングに合わせてウェイクアップされる．

上記の例では，I_p や I_c に直接依存していない I_1 の発行も取り消されている点に注意されたい．このような無依存の命令も無効化されることは，さらなる性能低下を生じさせる．これを抑制するには，ロード命令に直接または間接的に依存している命令のみを選択的に再発行すればよいが，そのためには複雑な機構が必要となる．このため，一般には上述したようにロード命令の発行以降に発行された命令を全て再発行する [1,5]．

2.4 投機ミスによる性能低下

投機ミス・ペナルティの性能への影響は大きい．投機ミスに伴い多くの命令が再発行されることに加え，プログラム中のロード命令の出現頻度は一般に高いためである．Alpha 21264 では，ヒットと予測して誤った場合には消費者が発行された可能性のある 2 サイクルの間に発行された，すべての命令を再発行する．最大で毎サイクル 4 命令が発行されるため，投機ミスにより再発行される命令は最大で 8 命令に達する [2,4]．最近のプロセッサでは Alpha 21264 よりパイプラインが深いため，取り消される命令の数はより多くなる．

3. 関連研究

既存のキャッシュ・ヒット/ミス予測器は，静的なロード命令ごとのキャッシュ・アクセス結果がヒットあるいは

ミスのどちらに偏っているかを利用するものである．具体的には，以下に示す方法がある．

(1) **静的方式 [3, 7]**
すべてのロード命令がキャッシュ・ヒットすると予測して，依存した命令の発行を行う．一般に，L1 データ・キャッシュのヒット率は比較的高いため，静的にヒットすると予測しても，ある程度高い精度が実現できる．

(2) **飽和型カウンタ方式**
静的な各ロード命令について過去のキャッシュ・ヒット/ミスの頻度をもとに予測を行う．例えば Alpha 21264 で採用されている予測器では，各エントリが 4 ビットの飽和型カウンタをもち，ロード命令の命令アドレスをインデクスとするテーブルからなる [2,4]．各カウンタの値はキャッシュ・ヒットすれば 1 増やされ，ミスすれば 2 減らされる．ロード命令の発行時には，カウンタの値の最上位ビットが 1 であればヒットと予測し，0 であればミスと予測する．

(3) **グローバル分岐履歴を利用した飽和型カウンタ方式 [9]**
福田らは，インデクス生成にグローバル分岐履歴を利用したライン・バッファ・ヒット/ミス予測器を提案している．ライン・バッファとは，最近アクセスしたキャッシュ・ラインを保持するバッファで，ロード命令のレイテンシを短縮するために用いられる．この予測器は，ヒット/ミスが制御フローに依存する性質を利用している．テーブルは (2) の飽和型カウンタ方式と同様の構成をとり，それにアクセスするためのインデクスには命令アドレスに加えてグローバル分岐履歴を用いる．

4. 既存手法の問題点

既存手法は静的なロード命令ごとのヒット/ミスが長期間変化しない場合の偏りを利用するものである．このため，短期間にヒット/ミスが変化するような場合にはそれをうまく予測することができない．

4.1 キャッシュ・ミスの周期的な発生

短期間のヒット/ミスの変化は，例えばメモリ・アクセスがキャッシュ・ラインの境界を跨ぐ場合に現れる．最も単純な例は，シーケンシャルにメモリをアクセスした場合，アドレスが新しいラインに差し掛かるたびに生じるミスである．例として，図 3 に示すコードを使ってこれを説明する．このコードでは，データを 8 バイトずつ読み出し

```
int64_t data[ M ];
for( int i = 0; i < M; i++ ){
    sum += data[ i ];
}
```

図 3 メモリへのシーケンシャル・アクセス

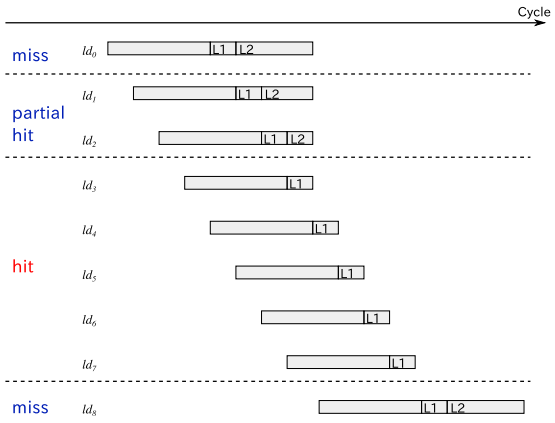

図 4 周期的なキャッシュ・ミスとパーシャル・ヒット

ている．ライン・サイズを 64 バイトとした場合，キャッシュが次のラインを保持していなければ 8 回ごとにキャッシュ・ミスが生じる．飽和型カウンタ方式ではヒット/ミスの頻度からこのロード命令は毎回ヒットすると予測をするため，キャッシュ・ミスが発生するたびに予測に失敗してしまう．

4.1.1 パーシャル・ヒットによる影響

前節では，8 回に 1 回の割合で予測を誤ることを示したが，命令スケジューリングの観点からは，予測誤りに起因するペナルティはより大きな頻度で被る．それは，ミスしたデータが下位のメモリ階層から得られるまでに同じロードが発行されると，完全なヒットではないパーシャル・ヒットとなるためである．パーシャル・ヒットでは，完全なヒットの場合よりレイテンシが長いため，ヒットと予測すると投機誤りを生じる．

図 4 に図 3 に示したコードを実行した際のパイプライン・チャートを示す．同図ではロード命令以外の命令と，ロード命令中のキャッシュ・アクセス以外のステージは省略している．図上の ld_n は，図 3 のコード内の `data[i]` に対応するロード命令であり，添字は最初にキャッシュ・ミスを起こしてからの実行回数に対応する．

ld_0 から ld_8 までの命令は以下のように実行される．

(1) ld_0 がキャッシュ・ミスを起こす．ld_0 は L2 キャッシュにアクセスし，L1 キャッシュにはラインがフィルされる．

(2) ld_1，ld_2 は ld_0 と同じラインにアクセスするが，フィルが完了しておらず，パーシャル・ヒットする．

(3) ld_3 から ld_7 はラインのフィルが完了しているためヒットとなる．

(4) ld_8 はラインの範囲外にアクセスするため，ラインがキャッシュに保持されていなければ ld_0 と同様にミスとなる．

同図のように実行された場合，ld_0 に加えて ld_1，ld_2 もキャッシュ・ヒット時のレイテンシで値を得ることができない．このため，ld_1 と ld_2 がヒットするものとしてそれ

らの消費者をウェイクアップするとペナルティが発生してしまう．

このような場合，飽和型カウンタに基づく方式ではペナルティの発生回数が大幅に増えてしまう．このロード命令は，キャッシュにミスするよりもヒットする回数のほうが多いため，偏りにより予測すればすべてヒットと予測される．そのため，8 回のアクセスのうち 3 回でペナルティを被ると考えられる．

5. 提案手法

本節では，キャッシュ・ヒット/ミスの短期的な変化を捉えて予測するヒット/ミス予測器を提案する．この予測器は，静的なロード命令ごとにキャッシュ・ミス以降のアクセス回数を記録し，その回数ごとの過去のヒット/ミスの偏りを基に予測を行う．

5.1 予測器の構成

提案する予測器の構成を図 5 に示す．この予測器は，CHT(Count History Table) と PHT(Pattern History Table) と呼ぶ 2 種類のテーブルによって構成される．CHT は各ロード命令のキャッシュ・ミス以降のアクセス回数を数えるためのテーブルである．PHT は飽和型カウンタを要素とするテーブルであり，静的なロード命令についてアクセス回数ごとのキャッシュ・ヒット/ミスの過去の偏りを記録する．

5.1.1 CHT の構成

CHT の各エントリは以下のフィールドを持つ．

- キャッシュ・ミスを起こしてからのロード命令のアクセス回数
- 命令アドレスの上位ビット (命令とエントリの対応を確認するためのタグ)

CHT には，命令アドレスの下位 i ビットをインデクスとしてアクセスする．アクセス時には，アクセスしたエントリのタグが命令アドレスの上位ビットと一致していれば，カウンタの値を出力する．

5.1.2 PHT の構成

PHT の各エントリは以下のフィールドを持つ．

- ヒット/ミスの偏りを記録するための 2 ビット飽和型カウンタ
- 命令アドレスの上位ビット (命令，カウンタ値とエントリの対応を確認するためのタグ)

PHT のインデクスは，CHT から出力されたカウンタ値と PC の下位 k ビットを XOR して得る．ただし，カウンタ値の方がビット長は短く，PC のビット列の上位につめて XOR する．

5.2 予測器の動作

CHT および PHT を用いた予測器の動作を説明する．

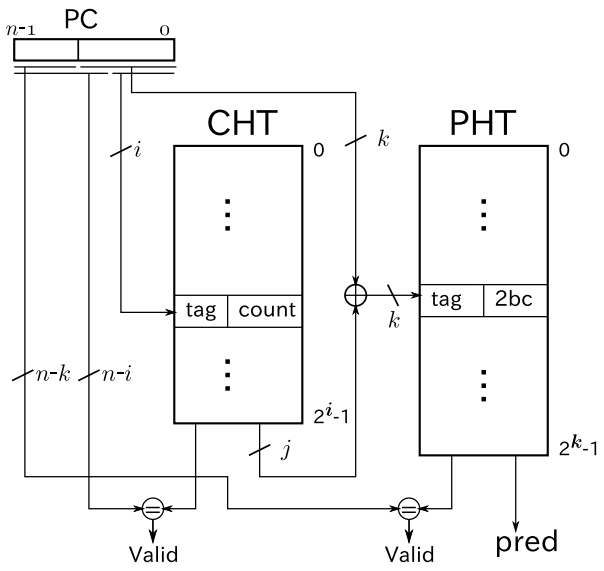

図 5 予測器の構成

5.2.1 学習の動作

学習時には，まず以下のようにして CHT にアクセスし，更新する．

(1) 対応するエントリがあった場合
  (a) 前回ヒットしたが今回はミスだった場合，リセットする．
  (b) それ以外ならばカウンタを +1 する．
(2) 対応するエントリがない場合
  (a) キャッシュ・ヒットしていれば，これ以降何もしない．
  (b) キャッシュ・ミスしていれば新しいエントリを割当てる．割当て時の初期値は 0 とする．

CHT のエントリの内容が読み出された場合，それを使って以下のように PHT にアクセスする．

(1) 対応するエントリがあれば，PHT のカウンタを更新する
  (a) キャッシュ・ヒットならば +1
  (b) キャッシュ・ミスまたはパーシャル・ヒットならば −1
(2) 対応するエントリがなければ，新しいエントリを割り当てる
  (a) ヒットならば初期値は 3
  (b) ミスならば初期値は 1

5.2.2 予測の動作

予測時には，まず以下のようにして CHT にアクセスする．

(1) 対応するエントリがあれば，カウンタの値を読みだして PHT にアクセスする
(2) 対応するエントリがなければ，キャッシュ・ヒットと予測する

CHT に対応するエントリがあった場合，カウンタの値

と PC を用いて PHT にアクセスし，予測を行う．
(1) 対応するエントリがあった場合，カウンタの値を読みだして予測を行う．
　(a) 2 以上ならばヒットと予測する
　(b) 1 以下ならばミスと予測する．
(2) 対応するエントリがなければ，キャッシュ・ヒットと予測する

5.3 周期性への対応

図 4 におけるヒット/ミスの変化を提案手法により予測した場合を考える．提案手法では，アクセス回数ごと，つまり，ld_0 から ld_7 までの各命令に対して独立に予測を行うことができる．これにより，ld_0 から ld_2 はキャッシュ・ミスまたはパーシャル・ヒットと，ld_3 から ld_7 はキャッシュ・ヒットと正確に予測することができる．

6. 評価

プロセッサ・シミュレータ鬼斬弐 [10] をベースに以下のモデルを実装して評価した．
(1) **AlwaysHit**: 常にキャッシュ・ヒットと予測するモデル
(2) **CounterBased**: Alpha 21264 と同様の飽和型カウンタ方式で予測を行うモデル
(3) **Periodic**: 提案手法により予測を行うモデル

ベンチマーク・プログラムには，SPECint2006 および SPECfp2006 を使用した．プログラムは，gcc 4.2.2 を用い，オプション”-O3” でコンパイルした．シミュレーションでは ref データ・セットを入力とし，先頭の 1G 命令をスキップした後，100M 命令を測定に用いた．評価したプロセッサの構成を表 2 に示す．また，AlwaysHit 以外のモデルでは，予測器のテーブルのエントリ数を表 3 のように変化させて測定した．

表 2 プロセッサの構成

フェッチ/リネーム幅	4 命令
発行幅	int:2 命令, fp:2 命令, mem:2 命令
ROB	128 エントリ
命令ウィンドウ	整数 32 エントリ
	浮動小数点 32 エントリ
LSQ	64 エントリ
L1 命令キャッシュ	32KB, 4-way, 64B ライン
	3 サイクル・ヒット・レイテンシ
L1 データ・キャッシュ	32KB, 4-way, 64B ライン
	3 サイクル・ヒット・レイテンシ
L2 キャッシュ	4MB, 8-way, 64B ライン
	15 サイクル・ヒット・レイテンシ
メイン・メモリ	200 サイクル・レイテンシ

表 3 パラメータの変化

CounterBased	4〜64K エントリ
Periodic	CHT: 64〜4K エントリ (4-way)
	PHT: 4〜64K エントリ (4-way)

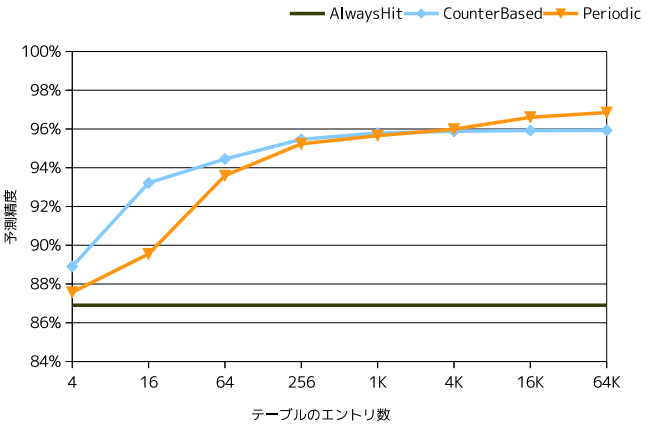

図 6 予測精度と予測器のエントリ数

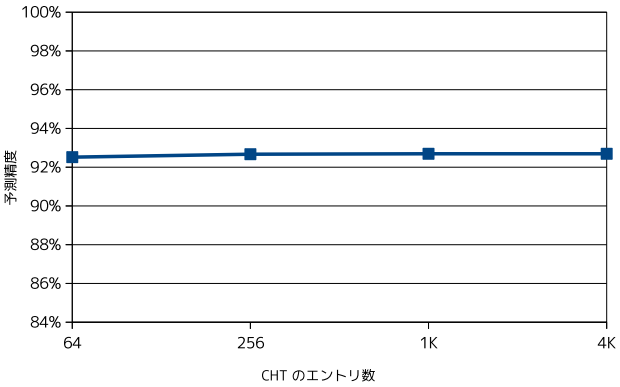

図 7 Periodic の予測精度と CHT のエントリ数

6.1 評価結果

6.1.1 予測精度

図 6 に，テーブルのエントリ数 (Periodic の場合，PHT のエントリ数) に対する予測精度の全ベンチマーク平均を示す．なお，CHT のエントリ数は 1K，カウンタは 6 ビットで固定である．

図 6 より，AlwaysHit モデルの予測精度は 86.9%であるが，CounterBased モデルの予測精度は 4K エントリの時に 95.9%と高い精度を示した．ただし，4K エントリ以上テーブル・サイズを大きくしても予測精度は向上しなかった．一方，Periodic モデルでは，少ないエントリ数では CounterBased モデルより低い予測精度となったものの，十分なエントリ数があれば CounterBased モデルより正確な予測をすることができる．PHT を 64K エントリとした場合，AlwaysHit モデルと比べて 10.0 ポイント，CounterBased モデルと比べて 0.9 ポイント高い予測精度を達成している．

図 7 に，PHT のエントリ数を 64K エントリで固定し，

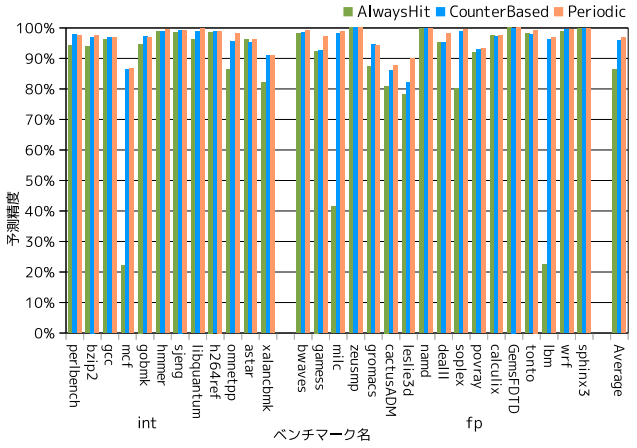

図 8　ベンチマーク別の予測精度

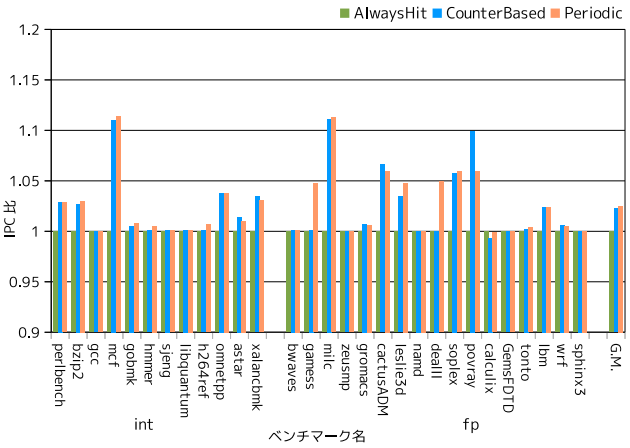

図 10　ベンチマーク別の IPC の変化

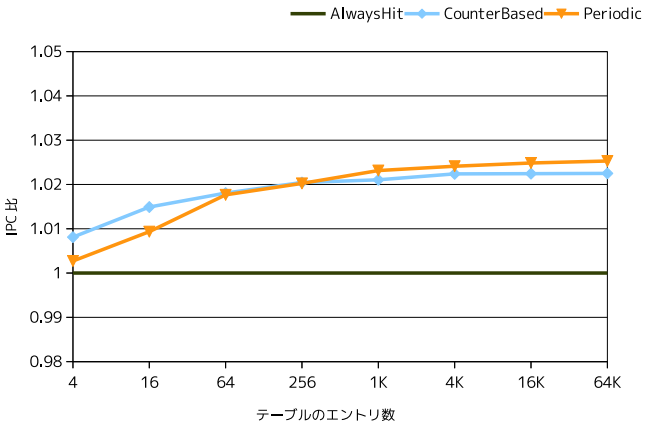

図 9　IPC の変化とエントリ数

CHT のエントリ数を変化させた場合の予測精度の全ベンチマーク平均を示す．同図より，CHT のエントリ数が予測の精度に与える影響は小さいことがわかる．

各モデルについて，ベンチマーク別の予測精度を図 8 に示す．ただし，CounterBased モデルでは予測器のエントリ数を 64K エントリとし，Periodic モデルでは PHT を 64K エントリとした．

zeusmp や GemsFDTD では，Periodic モデルでは CounterBased と同様に 99.9%以上の予測精度を達成した．また，lbm では AlwaysHit と比べて予測精度が 74.4 ポイント，leslie3d では CounterBased と比べて 8.0 ポイント向上した．一方で，mcf や cactusADM などは予測精度が低く，それぞれ 81.4%，87.6%に留まった．これらのベンチマークで精度が伸びなかった原因は，データ・アドレスが不規則に変化したため，ヒット/ミスの変化に周期性が現れなかったためであると考えられる．

6.1.2　IPC

図 9 に，テーブルのエントリ数 (Periodic の場合，PHT のエントリ数) を変化させた場合の AlwaysHit モデルに対する CounterBased および Periodic の性能比を示す．なお，CHT のエントリ数は 1K，カウンタは 6 ビットで固定である．

CounterBased モデルでは AlwaysHit モデルに対し，性能が 0.8%から 2.2%向上した．Periodic モデルでは AlwaysHit モデルに対し，性能が 0.3%から 2.5%向上した．また，CounterBased モデルではエントリ数を増やしても AlwaysHit に対する IPC の伸びが 2.2%に留まったのに対し，Periodic モデルにおける IPC の伸びはそれ以上となり，PHT を 64K エントリとした場合は CounterBased モデルと比べても IPC が 0.3%向上した．

図 10 に，AlwaysHit モデルに対する CounterBased および Periodic の性能比をベンチマーク毎に示す．ここで，CounterBased モデルでは予測器のエントリ数を 64K エントリとし，Periodic モデルでは PHT を 64K エントリとしている．

mcf や milc といったベンチマークでは Periodic では AlwaysHit モデルに対して性能が 10%以上向上した．また，cactusADM や soplex などの多くのベンチマークでも性能が 5%程度向上している．特に，gamess や dealII では，AlwaysHit だけでなく CounterBased と比べても性能を 5%程度向上させている．

7.　まとめ

スーパスカラ・プロセッサでは通常，キャッシュのヒット/ミスを予測することでレイテンシを予測し，それに基づいて命令を投機的に発行する．このために様々な予測器が提案されているが，それらの多くは静的な命令ごとのヒット/ミスの偏りに基づいて予測を行っている．これらの既存手法では，静的な命令ごとのヒット/ミスの偏りが長時間変化しない場合はうまく働くものの，短時間にヒット/ミスが変化する場合はそれをうまく予測することができない．

これに対し，本論文ではヒット/ミスの周期的な変化に着目した予測器を提案した．提案手法では，各ロード命令のキャッシュ・ミス以降のアクセス回数ごとにヒット/ミ

スの頻度を記録することで，周期的なヒット/ミスの変化を捉えることができる．提案した予測器をシミュレータに実装して評価を行ったところ，IPC を常にヒットと予測する場合と比べて最大で 11.4%，平均で 2.5%，飽和型カウンタ方式と比べて最大で 4.9%，平均で 0.3%向上させることができた．

謝辞

本研究の一部は，日本学術振興会 科学研究費補助金 若手研究 (A)（課題番号 24680005）による補助のもとで行われた．

参考文献

[1] Borch, E., Tune, E., Manne, S. and Emer, J.: Loose Loops Sink Chips, *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pp. 299 – 310 (2002).
[2] Compaq Computer Corporation: *Alpha 21264/EV6 Microprocessor Hardware Reference Manual* (2002).
[3] Hinton, G., Sager, D., Upton, M., Boggs, D., Carmean, D., Kyker, A. and Roussel, P.: The Microarchitecture of the Pentium 4 Processor, *Intel Technology Journal Q1* (2001).
[4] Kessler, R. E.: The Alpha 21264 microprocessor, *IEEE Micro*, Vol. 19, No. 2, pp. 24 –36 (1999).
[5] Morancho, E., Llaberia, J. M. and Olive, A.: Recovery mechanism for latency misprediction, *Proceeding of 2001 IEEE International Conference on Parallel Architectures and Compilation Techniques*, pp. 118 –128 (2001).
[6] Peir, J.-K., Lai, S.-C., Lu, S.-L., Stark, J. and Lai, K.: Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching, *Proceedings of the 16th international conference on Supercomputing*, pp. 189 – 198 (2002).
[7] Yeager, K. C.: The MIPS R10000 superscalar microprocessor, *IEEE Micro*, Vol. 16, No. 2, pp. 28 –41 (1996).
[8] Yoaz, A., Erez, M., Ronen, R. and Jourdan, S.: Speculation Techniques for Improving Load Related Instruction Scheduling, *Proceedings of the 26th International Symposium on Computer Architecture*, pp. 42 – 53 (1999).
[9] 福田祥貴，片山清和，島田俊夫：ライン・バッファ・ヒット/ミス予測を利用した動的命令スケジューリングの高精度化手法，2003 年先進的計算基盤システムシンポジウム SACSIS 2003，pp. 227–234 (2003).
[10] 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，2009 年先進的計算基盤システムシンポジウム SACSIS 2009，pp. 120–121 (2009).