社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－ARC－169 (13)
2006／8／1

# Decodeステージにおけるメモリロード命令のキャッシュミス予想

石川 健一郎†

CAMPHITはパイプライン化されたプロセッサのフロントエンドステージにおいてメモリロード命令のキャッシュヒット／ミスを予想する技術である。CAMPHITではヒストリテーブルとカウンタを用い予想を可能にする。ヒストリテーブルはメモリリード命令の命令アドレスと関連づけられており、過去のメモリリード命令のキャッシュヒット／ミスを記憶している。カウンタはメモリリード命令によるメモリアクセスがキャッシュヒットの場合、値は増加し、キャッシュミスの場合、値は減少する。CAMPHITはヒストリテーブルの履歴及びカウンタの値を評価しキャッシュヒット／ミスを予想する。CAMPHITはMIBench、SPECint2000を用いて評価された。その結果、キャッシュミスと予想した場合の99.5%、99.0%において予想に成功した。

# A prediction of a cache miss in a memory load instruction for the decode stage

KEN-ICHIRO ISHIKAWA†

CAMPHIT is a technique which can predict a cache hit/miss of a memory read instruction for a front end stage on a pipelined processor.CAMPHIT can predict using for a history table and a counter. The history table is associated with a instruction address of the memory read instruction and it memorize the cache hit/miss of the memory read instruction in the past. The counter is incremented when the memory read instruction is a cache hit, is decremented when the memory read instruction is a cache miss. CAMPHIT can predict by evaluating a history in the history table and value of the counter. CAMPHIT is evaluated what is used for MIBench and SPECint2000. As a result, 99.5% in MIBench and 99.0% in SPECint2000 of predicted as a cache miss are the cache miss.

## 1. はじめに

近年のプロセス技術の進歩により，CPU、メモリの高速化が進んでいる．だが，CPUの高速化と比較しメモリの高速化は緩やかであり，両者の速度差が大きくなってきている．この速度差を埋めるためにキャッシュメモリが存在する．小容量だが高速なキャッシュメモリに使用頻度の高いデータを置くことにより高速な動作を実現している．だが，キャッシュミスした場合，多くの場合，パイプラインがストールするなど，深刻な速度低下を招くことが問題となっている．キャッシュミスは命令キャッシュのキャッシュミスとデータキャッシュのキャッシュミスに分けられる．データキャッシュのキャッシュミスによる速度低下は2種類ある．

(1) スーパースカラCPUにおいて命令のスケジューリングを行う際，メモリロード命令がキャッシュヒットする場合とキャッシュミスする場合、それぞれそのメモリロード命令に依存する命令の効率のよいスケジューリングが異なる．多くの既存のCPUではキャッシュヒットを前提にスケジューリングが行われるが，キャッシュミスした場合スケジューリングからやり直す必要がある．

(2) メモリロード命令がキャッシュミスした場合，そのメモリロード命令に依存する命令は動作速度の遅いメモリからデータが読み込まれるまで実行できない．動作速度の遅いメモリからデータを読み込むためには長い時間が必要なため，パイプラインがストールする．

これらの問題に対処するため，メモリロード命令のキャッシュヒット／ミスをあらかじめ予想し，その予想を元に処理を行う手法が注目を集めている．Alpha21264[1]ではキャッシュヒット／ミスにより値が増減するカウンタを用い，スケジューリングの段階でメモリロード命令によるメモリアクセスのキャッシュヒット／ミスを予想し，その結果を元にスケジューリングを行っている．スケジューリングの際にキャッシュヒット／ミスを予想する技術の研究は盛んに行われており，キャッシュヒット／ミスを記憶したテーブルを用いる方法，キャッシュヒット／ミスにより増減するカウンタを用いる方法，両者をハイブリッドした方法など様々な手法が提案されている[2][3][4]．

マルチスレッドCPUではメモリロード命令によるメモリアクセスがキャッシュミスした場合，実行する命令のスレッドを変更することにより，パイプラインのストールを最小限に抑えることができる．このとき，フロントエンドステージでキャッシュミスを予想することにより，より効率のよい実行が可能になる．現在，この問題に対処するためのキャッシュヒット／ミス予想技術の研究はほとんど行われていない．本論文ではこの問題に対処するた

† 防災科学技術研究所
National Research Institute for Earth Science and Disaster Prevention

めのキャッシュヒット／ミス予想技術を提案する．

本論文は以下の様な構成となっている．第 2 章で既存の技術についてサーベイを行い，第 3 章で CAMPHIT について具体的な手法を示す．第 4 章で評価方法について述べ，第 5 章で結果を記述する．第 6 章で考察を行い，最後に第 7 章で結果をまとめる．

## 2. 関連研究

メモリロード命令のキャッシュヒット／ミスを予想する手法を取り入れた CPU として Alpha21264[1] があげられる。Alpha21264 ではスケジューリングの際にメモリロード命令のキャッシュヒット／ミスを予想し、その結果を元にスケジューリングを行い、パフォーマンスを向上させている。Alpha21264 では 4 ビットのカウンタを用い判定を行う。カウンタはメモリアクセス命令がキャッシュヒットした場合は 1 加算され、キャッシュミスした場合は 2 減算される。評価の際にはカウンタの値が一定値を超える場合キャッシュヒットと判定し、一定値を超えない場合キャッシュミスと判定する。

メモリロード命令のキャッシュヒット／ミスを予想する技術としてロードするメモリアドレスによって判定する方法がある。分岐予想と同様にヒストリテーブルを用いる。ヒストリテーブルには過去数アクセス分のキャッシュヒット／ミスを記憶するエントリがあり、各エントリはロードするメモリアドレスに関連づけられている。[3] では Bloom Filter と呼ばれる手法を用いキャッシュヒット／ミスを記憶したヒストリテーブルにアクセスすることによりキャッシュヒット／ミスを予想する。Bloom Filter とはビット列のハッシュを求めそのハッシュを元にテーブルに含まれているか判定する確率的アルゴリズムである。[3] では Bloom Filter を用いることにより、高速にキャッシュヒット／ミスを記憶したテーブルにアクセスし、予想を行う。この手法により 1KB のテーブルを用いた場合、キャッシュミスを予想した場合確実にキャッシュミスし、全キャッシュミスの 97%を予想することが可能である。

スケジューリングの際にメモリロード命令のキャッシュヒット／ミスを予想する技術はすでに開発されている。だが、パイプラインのフロントエンドステージでメモリロード命令メモリアクセスのキャッシュヒット／ミスを予想する手法には開発の余地がある。本論文ではパイプラインのフロントエンドステージでのキャッシュヒット／ミス予想について述べる。

## 3. CAMPHIT

CAMPHIT(CAche hit/Miss Prediction by a HIstory Table and a global counter) はヒストリテーブルとカウンタを用いることによりメモリロード命令のレイテンシの長短を予想する技術である．以下、レイテンシの短いアクセスをキャッシュヒットリード、レイテンシの長いアクセスをキャッシュミスリードと呼ぶ

本論文では予想はプロセッサのパイプラインのデコードステージで行われる事を想定しているが簡単なデコード回路を追加することによりキャッシュメモリに読み込まれる段階，Fetch ステージで処理する段階などにおいても同じ手法を使用することができる．

ヒストリテーブルは過去数アクセスのメモリロード命令のキャッシュヒットリード／キャッシュミスリードを記憶するエントリを持っており，各エントリはメモリロード命令の命令アドレスに関連づけられている．関連付ける方法としてはキャッシュ同様ダイレクトマップ方式，n ウエイセットアソシエイティブ方式などが考えられる．

カウンタは数ビットの値を保持し，キャッシュヒットリードの場合インクリメントされ，キャッシュミスリードの場合デクリメントされる．

デコードステージにおいてメモリリード命令のキャッシュヒットリード／キャッシュミスリードを次のように判定する．

(1) 命令アドレスと関連づけられたヒストリテーブルのエントリを参照し，過去数アクセスがキャッシュミスリードであった場合，ヒストリテーブルによる判定においてキャッシュミスリードとする．

(2) カウンタを参照し，一定の値以下であった場合，カウンタによる判定においてキャッシュミスリードとする．

(3) ヒストリテーブルによる判定及びカウンタによる判定においてキャッシュミスリードと判断された場合，そのメモリロード命令のメモリアクセスはキャッシュミスリードと判定する．

ヒストリテーブルは判定に使用可能な時間，チップ面積，必要な精度に応じて，過去何アクセスまでキャッシュミスリード／キャッシュヒットリードを記憶するのか，何エントリ記憶するのか決定する．

また，カウンタのカウンタのビット幅，キャッシュヒットリードの場合加算される値，キャッシュミスリードの場合減算される値は様々な値が考えられる．

CAMPHIT の用途としては以下のようなものが考えられる。

- 粗粒度マルチスレッディングプロセッサの場合、CAMPHIT によりキャッシュミスリードが予想された場合スレッドを切り替えることにより、メモリロードのためパイプラインがストールすることを防ぐ。
- 細粒度マルチスレッディングプロセッサの場合、CAMPHIT によりキャッシュミスリードが予想された場合そのスレッドに切り替えることを一定時間止めることにより、より効率的に実行する。

マルチスレッディングプロセッサはパイプラインのステージ数が多いものが一般的である。そのため、メモリロード命令がキャッシュミスリードになることが確定した段階で上記のようにスレッドを制御した場合、パイプラインにはすでにストールする可能性の高い命令が存在し

ており、効果が期待できない。CAMPHIT はパイプラインのフロントエンドにおいてキャッシュヒットリードを予想することが可能であり、大きな効果が期待できる。

## 4. 評　　価

CAMPHIT の評価を行った。キャッシュミスリード／キャッシュヒットリードの判定は、L1 データキャッシュにアクセスする際最小限必要なレイテンシでアクセス可能であることをキャッシュヒットリード、それより多くのレイテンシを必要とする場合をキャッシュミスリードと定義した。評価にはヒストリテーブル機能及びカウンタ機能を追加した SimpleScalar-3.0d を用いた。SimpleScalar-3.0d に含まれるシミュレーションプログラムの中の sim-outorder を用いた。sim-outorder は複数命令を同時に実行可能であり、Fetch、Decode 及び Dispatch、Issue、Write Back のステージに分かれている。命令の実行は Decode 及び Dispatch ステージで行われるが実行結果が有効になるタイミングを調整することにより Issue ステージ後の実行になるようになっており、実質的に Issue ステージと Write Back ステージの間に Execution ステージがある。4 命令同時読み込み、4 命令同時発行が可能である。演算器は次のような構成となっている。

- 4 整数命令実行器
- 1 整数乗算除算器
- 2 ロードストア命令実行器
- 4 浮動小数点演算器
- 1 浮動少数点乗算除算器

キャッシュは次のような構成となっている。

- L1 データキャッシュ:128 テーブル、1 テーブル 32 バイト、4 ウエイセットアソシエイティブ、1 レイテンシ
- L1 命令キャッシュ:512 テーブル、1 テーブル 32 バイト、ダイレクトマップ方式、1 レイテンシ
- L2 キャッシュ:ユニファイドキャッシュ、1024 テーブル、1 テーブル 64 バイト、4 ウエイセットアソシエイティブ、6 レイテンシ

CAMPHIT のシミュレーションは次のように行う。デコードステージにおいてヒストリテーブルの対応するエントリの値及びカウンタの値を記憶し、Issue ステージにおいてキャッシュヒットリードの場合ヒストリテーブルのエントリの値及びカウンタの値別に、キャッシュヒット数を記憶する変数の値を増加させ、キャッシュミスリードの場合、ヒストリテーブルのエントリの値及びカウンタの値別に、キャッシュミス数を記憶する変数の値を増加させる。同時にヒストリテーブルとカウンタのアップデート用データを保存し、1 クロック終了毎にヒストリテーブルとカウンタのアップデートを行う。

ベンチマークプログラムとして MIBench 及び SPECint を用いた。使用したプログラムは MIBench は AES、basicmath、bf、bitcnts、crc、fft、gs、gsm、ispell、jpeg、lame、qsort、raw、search、sha、susan、tiff の計 17 種類であり、入力ファイル、使用する機能の違いなど 56 パターンで評価を行った。SPECint2000 は gzip、vpr、gcc、mcf、wupwise、vortex、bzip2 の計 7 種類 19 パターンで評価を行った。ヒストリテーブルは記憶するアクセス数を M、エントリ数を N としたとき (M,N)=(3,128)、(6,64)、(12,32) の 3 通りで評価を行った。ヒストリテーブルの容量は全評価条件で同一である。カウンタはビット幅 A ビット、キャッシュヒットリード時に B インクリメントされ、キャッシュミスリード時に C デクリメントされるとすると (A,B,C)=(2,1,1)、(4,1,2) の 2 通り、及びカウンタを使用しない場合の評価を行った。(2,1,1) は標準的な分岐予想に使用される値を、(4,1,2) は Alpha21264 で使用された値を参考に決定した。

評価項目はキャッシュヒットリード予想時のキャッシュヒットリード率、キャッシュミスリード予想時のキャッシュミスリード率、全キャッシュミスリードに対するキャッシュミスリード予想時のキャッシュミスリードの割合、予想の正解率である。

## 5. 結　　果

カウンタによるキャッシュミスリード／キャッシュヒットリード予想の結果を表 1 に示す。評価には MIBench を用いた。表 1 の値は $\alpha$ が各 counter の値におけるメモリリード命令に対するキャッシュミスリードの割合、$\beta$ が全キャッシュミスリードに対する各 counter の値のキャッシュミスリードの割合である。この結果を元に (A,B,C)=(2,1,1) の時、カウンタが 2 以下の場合をカウンタによる予想でキャッシュミスリード、それより大きい場合カウンタによる予想でキャッシュヒットリードとした。(A,B,C)=(4,1,2) の時、カウンタが 4 以下の場合をカウンタによる予想でキャッシュミスリード、それより大きい場合カウンタによる予想でキャッシュヒットリードとした。

| | (2,1,1) | | (4,1,2) | |
|---|---|---|---|---|
| counter | $\alpha$ | $\beta$ | $\alpha$ | $\beta$ |
| 0 | 3.4% | 1.9% | 4.2% | 2.1% |
| 1 | 36.2% | 10.8% | 46.4% | 6.9% |
| 2 | 49.3% | 18.3% | 87.6% | 12.7% |
| 3 | 1.1% | 69.0% | 67.3% | 7.2% |
| 4 | | | 89.2% | 24.8% |
| 5 | | | 25.3% | 0.9% |
| 6 | | | 27.5% | 1.0% |
| 7 | | | 13.7% | 0.6% |
| 8 | | | 29.3% | 1.1% |
| 9 | | | 18.6% | 2.5% |
| 10 | | | 22.6% | 1.1% |
| 11 | | | 20.2% | 1.5% |
| 12 | | | 14.2% | 1.5% |
| 13 | | | 20.0% | 4.2% |
| 14 | | | 16.7% | 4.3% |
| 15 | | | 0.4% | 27.6% |

表 1　カウンタによる予想
Table 1　Prediction by counter

MIBench による評価の結果を表 2 に示す。キャッシュ

ヒットリード予想時にキャッシュヒットリードであった確率を PHAH-Ratio(表中 (1))、キャッシュミスリード予想時にキャッシュミスリードであった確率を PMAM-Ratio(表中 (2))、全キャッシュミスリードに対するキャッシュミスリード予想時のキャッシュミスリードの割合を PMAM/AllM-Ratio(表中 (3))、キャッシュミスリード／キャッシュヒットリード予想の正解率を All-Ratio(表中 (4)) と呼ぶ。

使用した MIBench プログラム全てのメモリロード命令のキャッシュミスリードの割合は 1.6% となった。

| Table Entry | Counter Type | (1) | (2) | (3) | (4) |
|---|---|---|---|---|---|
| 32 | none | 99.2% | 99.1% | 48.2% | 99.2% |
| | (2,1,1) | 98.8% | 99.3% | 18.7% | 98.8% |
| | (4,1,2) | 99.2% | 99.5% | 46.4% | 99.2% |
| 64 | none | 99.3% | 98.2% | 52.0% | 99.3% |
| | (2,1,1) | 98.8% | 98.9% | 20.1% | 98.8% |
| | (4,1,2) | 99.2% | 99.3% | 49.0% | 99.2% |
| 128 | none | 99.5% | 94.0% | 58.6% | 99.3% |
| | (2,1,1) | 98.8% | 96.5% | 22.2% | 98.8% |
| | (4,1,2) | 99.2% | 98.1% | 51.0% | 99.3% |

表 2 MIBench の結果
Table 2 MIBench result

カウンタタイプ別の PMAM-Ratio は (4, 1, 2) > (2, 1, 1) > *none* となった。PMAM/AllM-Ratio は *none* > (4, 1, 2) >> (2, 1, 1)、All-Ratio は *none* > (4, 1, 2) > (2, 1, 1) であり (2,1,1) の実用性はほとんどない。テーブルのエントリ数は 32 の時 PMAM-Ratio がよく、128 の時 PMAM/AllM-Ratio, All-Ratio がよい。64 は 32 と 128 の中間の値を示した。MIBench による評価の結果から次のような結論が得られる。

- PMAM-Ratio を重視する場合テーブルのエントリ数を 32、カウンタの設定を (4,1,2) とする
- PMAM/AllM-Ratio、All-Ratio を重視する場合テーブルのエントリ数を 128、カウンタを使用しないとする
- バランスを重視する場合テーブルのエントリ数を 64、カウンタの設定を (4,1,2) とする

All-Ratio を重視した場合キャッシュミスリード／キャッシュヒットリード予想が外れる可能性は 0.7%であり、常にキャッシュヒットリードと予想する場合 (CAMPHIT を使用しない場合) と比較し半分以下となっている。また、PMAM-Ratio を重視した場合 PMAM-Ratio は 99.5%であり、ほぼ正確に予想できることが示された。

SPECint2000 による評価を行った。SPECint2000 の場合、全ロード命令に対するキャッシュミスリードの割合は約 8.9%である。SPECint2000 による評価では MIbench による評価をふまえカウンタによる評価は行わない場合と (4,1,2) を採用した場合に限った。表中の項目は MIBench の表に準じる。

テーブルエントリの増加とともに

- PHAH-Ratio は増加する
- PMAM-Ratio は減少する

| Table Entry | Counter Type | (1) | (2) | (3) | (4) |
|---|---|---|---|---|---|
| 32 | none | 93.9% | 98.7% | 32.8% | 94.0% |
| | (4,1,2) | 93.6% | 99.0% | 29.8% | 93.8% |
| 64 | none | 94.4% | 96.3% | 38.3% | 94.4% |
| | (4,1,2) | 93.9% | 97.7% | 33.0% | 94.1% |
| 128 | none | 95.0% | 90.7% | 46.3% | 94.8% |
| | (4,1,2) | 94.2% | 94.9% | 36.4% | 94.4% |

表 3 SPECint2000 の結果
Table 3 SPECint2000 result

- PMAM/AllM-Ratio は増加する
- All-Ratio は増加する

カウンタによる予想を行わない場合、カウンタによる予想を行う場合と比較し PHAH-Ratio は増加、PMAM-Ratio は減少し、PMAM/AllM-Ratio は大きく増加するため、All-Ratio は増加する。各テーブルエントリ数、カウンタによる予想の有無による結果の変化は MIBench の場合とほぼ同じである。

最も All-Ratio が高いテーブルエントリ数 128、カウンタによる予想を行わない場合キャッシュミスリード／キャッシュヒットリード予想が外れる可能性は 5.2%であり、常にキャッシュヒットリードと予想する場合 (CAMPHIT を使用しない場合) と比較し半分強となっている。また、PMAM-Ratio を重視した場合 PMAM-Ratio は 99.0%であり、ほぼ正確に予想できることが示された。

## 6. 考　　察

CAMPHIT によりキャッシュミスリードの予想が可能である理由として以下の 2 つが考えられる。

- メモリアクセス時に同じキャッシュテーブルに書き込む命令が近接しており、それらの命令が同じキャッシュテーブルを書き換え合うため同じメモリロード命令が常にキャッシュミスリードになる (a)
- 同じメモリアドレスのメモリリード命令が常に新しいアドレスにアクセスするためキャッシュにヒットせず常にキャッシュミスリードになる (b)

(a) が原因であれば、キャッシュの容量、Way 数を見直すべきである。また、将来のプロセス技術、回路設計技術などの進歩によりキャッシュ容量、Way 数が増加し (a) を原因とするキャッシュミスロードは減少するため、CAMPHIT の意味は薄れる。(b) が原因であればプリフェッチ等の対応策はあるが、メモリロード命令の直前にメモリロード先が決定することは頻繁に起きるため根本的な対応策は無く、CAMPHIT は将来的に渡って有用な技術である。。

キャッシュミスリードの予想が可能である理由を確認するためメモリアクセスパターンを解析した。L1 データキャッシュは 128 テーブル存在し、4 ウエイセットアソシエイティブ制御であることから FIFO 制御する 512 テーブルのバッファを用い、以下の評価を行い解析した。

(1) メモリリード、メモリライトがあったアドレスをバッファに記憶する。同じアドレスがバッファに存在する場合、新しく記憶せず、アドレスを FIFO の

先頭に移動する。

(2) キャッシュミスリードの場合バッファにアドレスが記憶されていることを確認し、結果を記憶する。

MIBench、SPECint2000 による評価の結果、キャッシュミスリードの時、バッファにアドレスが記憶されている場合はそれぞれ約 8.6%、約 11.4%にすぎないことが分かった。このことから、(a) のケースは (b) のケースと比較し非常に少ないことが分かった。よって CAMPHIT が今後も有効な予想手段であることが示された。

## 7. 結　　論

パイプラインのフロントエンドステージにおいてメモリロード命令のキャッシュヒット／ミスを予想する手法 CAMPHIT を提案した。SimpleScalar-3.0d の sim-outorder を改造したものを使用し、MIBench および SPECint2000 による評価を行ったところ、キャッシュミスと予想したメモリロード命令の 99.5%、99.0%がキャッシュミスした。

今後さらに精度の高い予想手法を開発する予定である。

## 参 考 文 献


1) "THE ALPHA 21264 MICROPROCESSOR" RE Kessler IEEE Micro, 1999
2) "Speculation Techniques for Improving Load Related Instruction Scheduling" A Yoaz, M Erez, R Ronen, S Jourdan ACM SIGARCH Computer Architecture News, 1999
3) "Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching" J.-K. Peir, S.-C. Lai, S.-L. Lu, J. Stark, and K. Lai. In Proc. of the 16th Int'l Conference on Supercomputing,
4) "ACCESS-MODE PREDICTIONS FOR LOW-POWER CACHE DESIGN" Z Zhu, X Zhang IEEE Micro, 2002