計算機アーキテクチャ 140－7
（2000．11．30）

投機的データプリフェッチを行うキャッシュの一考察

村上 和希 弘中 哲夫 吉田 典可
広島市立大学大学院 情報科学研究科 情報工学専攻

ブロック単位で参照アドレスを予想し，データのプリフェッチを行うキャッシュを提案する．本方式では過去に参照されたデータアドレスの履歴情報はブロック毎に管理する．メモリ参照履歴情報を参照アドレスの出現順に格納し，順に予測を行うことにより予測時に検索する履歴情報を減らすことが出来る．本稿ではアドレス予測を行うPrefetch Cacheの構成例を示した．compressを用いた評価では従来のダイレクトマップのキャッシュのミス率に比べ，大幅に改善できる可能性を示した．

Study on Speculative Data Prefetch Cache

Waki MURAKAMI, Tetsuo HIRONAKA, Noriyoshi YOSHIDA
Hiroshima City University
Graduate School of Information Sciences

This paper proposes a new data prefetch cache architecture that predicts reference address based on basic blocks. In this architecture, memory reference address history is managed by grouping it by basic blocks. In basic blocks, the sequence of memory reference do not differ in each iteration, so by grouping memory reference history by basic blocks may save the search area need to predict the next reference address. This paper presents details of the prefetch cache architecture, which predicts reference address. From the results of performance evaluation using compress program as a benchmark, shows the possibility for a big performance improvement by the proposed architecture, compared to conventional direct map cache.

1 はじめに

近年，プロセッサとメモリの動作速度の差はますます開いてきている．この速度差を隠蔽するため，キャッシュの値予測やデータプリフェッチの研究が行なわれている．本研究ではキャッシュのミス率を下げるためのアドレス予測の方法について考え，少ないハードウェアコストで正確なアドレス予測を行うアドレス予測機構を考える．

アドレス予測は予測するロード/ストア(LD/ST)命令の過去のメモリ参照履歴情報を利用して行う．従来のアドレス予測研究[1]は主にアドレス予測のためのLD/ST命令毎のメモリ参照履歴情報を命令アドレスをタグとする連想メモリによって管理している．この方式では完全な履歴の管理のためには膨大なハードウェアが必要である．ハードウェアコストの削減のためダイレクトマップキャッシュのような不完全な連想メモリで履歴の管理を行うと，LD/ST命令毎のメモリ参照履歴情報が一部欠落し，予測の精度が低下する．

本研究では基本ブロック単位で各LD/ST命令のメモリ参照履歴情報を管理する新しいアドレス予測を提案する．本方式は基本ブロック内のLD/ST命令の実行順序が変化しないことに着目し，基本ブロックを単位としてメモリ参照履歴情報の管理を行うことで，LD/ST命令毎のメモリ参照履歴情報の管理を不要にし，小規模な連想メモリで精度の高いアドレス予測を実現する．

2 関連研究

Chenらによって提案されたハードウェアプリフェッチ機構[1]ではアドレス予測に参照予測テーブル(reference prediction table;RPT)を用いる．RPTでは各LD/ST命令のメモリ参照履歴情報を管理するため，タグとしてLD/ST命令の命令アドレスを用いる．RPTは履歴情報として前回参照したアドレス，過去2回の参照アドレスの差分，予測状態を保持する．RPTの各エントリのタグは毎サイクル，プログラムカウンタ(PC)と比較され，予測対象となっているLD/ST命令をRPT内から検出する．この予測対象LD/ST命令検出において履歴のタグとPCを比較するため，RPTのエントリ数と同数の比較器が必要となる．

提案方式では基本ブロックの最初の命令アドレスをタグとして利用することで，基本ブロック内の全てのLD/ST命令のメモリ参照履歴情報を管理する．基本ブロック内のLD/ST命令の実行順序は変化することがないので，各々の命令アドレスの識別は不要となる．つまり基本ブロック単位でLD/ST命令のメモリ参照履歴情報を管理することにより，タグ比較のための比較器数を，管理するLD/ST命令数から内部にLD/ST命令を複数持つ基本ブロック数へと大幅に削減できる．

この他に，ハードウェアプリフェッチ機構で使用する比較器数を削減する試みとして，ダイレクトマップキャッシュのように各LD/ST命令の命令アドレスから参照履歴テーブル内のインデックスとタグを生成し，両者を使って2次元的にエントリを管理する方式がある[2]．しかし，この方式では比較器の数は大幅に削減できるが，同じインデックスを持つLD/ST命令がメモリ参照履歴テーブル内でエントリの衝突を起こし，使用頻度の高いLD/ST命令もリプレースの対象になってしまう危険性がある．

提案方式ではプログラムの構造を考慮した基本ブロッ

ク単位でメモリ参照履歴情報を管理する．エントリの参照時間や参照回数を考慮したメモリ参照履歴情報のリプレースアルゴリズムを基本ブロック単位で用いることにより，より効率の良い予測が行なえると考えられる．また，基本ブロックを単位とするため，命令アドレスのインデックスやタグを使用する場合に比べてタグの数が減る．このためメモリ参照履歴情報テーブル内のメモリの使用効率が良くなる．

3 アドレス予測

本研究では，基本ブロック毎に各LD/ST命令のメモリ参照履歴情報を管理し，それを用いてアドレス予測を行う．しかし，動的に真の基本ブロックを抽出することは困難なため，基本ブロック相当のブロックを用いる．本節では最初に予測単位となる基本ブロック相当のもの(以降これをブロックと呼ぶ)について説明する．次にブロック毎にLD/ST命令のメモリ参照履歴情報を格納するための参照アドレステーブル(Reference Address Table;RAT)について説明する．最後にアドレス予測を行うハードウェアの動作を説明する．

3.1 ブロック

アドレス予測は基本ブロックで行う．ブロックと基本ブロックの違いは，ブロックの途中で制御の流れが合流する場合，図1に示すようにブロックの開始アドレスによって複数のブロックが生成される点である．

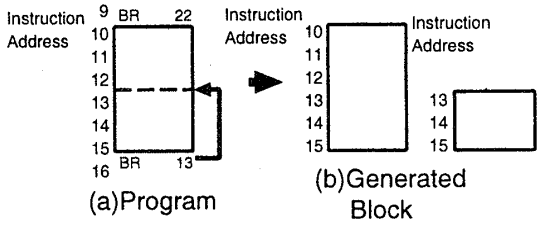

図 1: ブロック

ブロックの生成はプログラム実行時に以下のアルゴリズムで行なわれる．

1. 分岐命令が終了し，分岐先アドレス(BADR)が判明
2. 判明したBADRから始まるブロックが存在しなければ，BADRから始まる新しいブロックを生成
3. 分岐命令が出現した場合，分岐命令の直前の命令までのブロックを生成

3.2 参照アドレステーブル(RAT)

アドレス予測はメモリ参照履歴情報に基づいて行う．メモリ参照履歴情報はブロック毎にまとめて記憶される．1度以上実行され，メモリ参照履歴情報が既に存在するブロックを再び実行する場合は，RATの当該エントリの情報を利用して実行されるブロック内にあるLD/ST命令のアドレス予測が行なわれる．以下，ブロック内の参照アドレス(Reference Address;RA)を格納するRATエントリの構成要素，及びRATエントリの追加方法について説明する．

3.2.1 参照アドレステーブル(RAT)の構成要素

RATの構成要素の詳細を以下に示す(図2参照)．RATはメモリ参照履歴情報のうち，ブロック毎に管理される情報を格納するブロック状態テーブル(Block State Table;BST)と個々の命令毎に管理される情報を格納するアドレス予測テーブル(Address Prediction Table;APT)から成る．ブロック内で記憶できるLD/ST命令数を可変にするため，APTとBSTは分割されている．個々のLD/ST命令が前回参照した参照アドレス(Previous Reference Address; PRA)とストライド(Stride)はAPTに参照順に記憶され，各BSTエントリの参照ポインタ(Reference Pointer;RP)によって管理されている．Block Index(BI)毎に記録したAPT内の要素数は履歴数(Number of Records;N)によって指定される．

- BSTに記憶する情報
  - **BI(Block Index)**：予測対象ブロックの検索に使うブロックの開始アドレス
  - **N(Number of Records)**：ブロック内でAPTに記憶されたLD/ST命令数
  - **State**：ブロックの予測状態
  - **RP(Reference Pointer)**：ブロックが使用しているAPTの先頭を指すポインタ
- APTに記憶する情報
  - **PRA(Previous Reference Address)**：1つ前の参照アドレス履歴
  - **Stride**：参照アドレスの過去の履歴から求めた差分

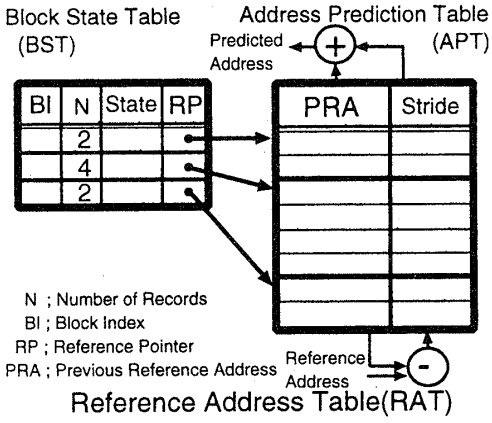

図 2: 参照アドレステーブル(RAT)のエントリ

3.2.2 参照アドレステーブル(RAT)エントリの追加

アドレス予測は分岐先アドレスと一致するBIをもつRATエントリの情報を利用して行う．

各RATエントリはブロック内に存在するLD/ST命令のメモリ参照履歴情報を保持している．分岐命令が実行されると，分岐先で最初に実行される命令のアドレスが各RATエントリのBIと比較される．RAT内にBIが一致するエントリがない場合，新しいRATエントリが生成される．新しいRATエントリのBIはブロックの開始アドレス，すなわち分岐先アドレスとなる．具体的なRATエントリの追加方法については，4.3.1項で動作例を用いて示す．

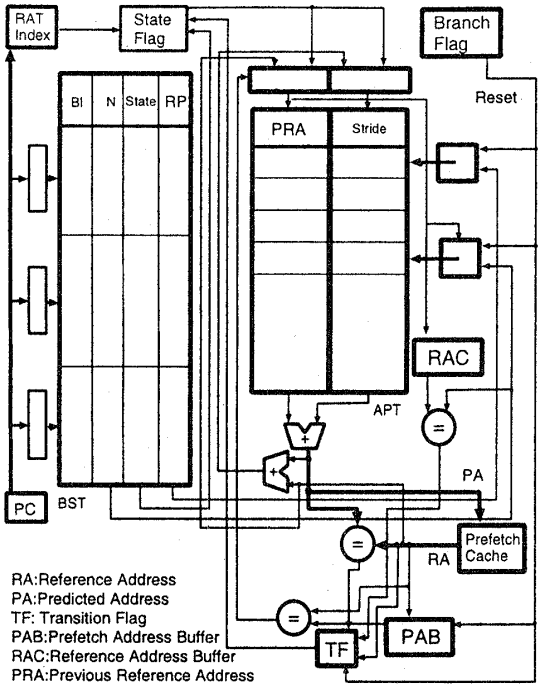

図 3: RAT 予測機構

3.3 アドレス予測機構

図3に RAT を用いた予測機構の構成例を示す．図3中の Prefetch Cache は 4 節で述べるプリフェッチを行うキャッシュである．データプリフェッチはキャッシュライン単位で行うものとしている．

本項では最初に図3の主なユニットについて説明する．次にブロックの実行状態別に図 3に示されたユニットの動作と RAT エントリの更新操作について述べる．

3.3.1 アドレス予測機構の構成要素

アドレス予測機構は以下のユニットで構成される．

RAC(Reference Address Counter): 実行中のブロック内で RAT に記憶された Reference Address の数を数える．

PAB(Previous Address Buffer) : ブロック内において直前の LD/ST 命令が使用した参照アドレスを保持する．

State Flag : 実行中のブロックに対応する RAT エントリの状態を保持する．

RAT Index : 実行中のブロックの RAT 内の場所を示す．

TF(Transition Flag): ブロック内のアドレス予測の結果により RAT の State の値を決定する．

以下の条件の判定結果により RAT の Status を決定する TF の更新が行なわれる．2 及び 3 の比較結果はブロック実行中に TF に記憶される．この値によりブロック終了時に RAT の State が更新される．

1. RAC の値が 0 でないか
2. 予測アドレスとブロック内の LD/ST 命令の参照アドレスが全て一致したか
3. RAC と N の値が一致したか

TF の値は条件 2 を満たさないとき A(アドレス不一致)，条件 3 を満たさないとき N(参照数不一致)，条件 2，3 を満たすとき S(アドレス，参照数一致) となる．

なお，予測アドレスとは RAT の PRA に Stride を足したものである．N のときは当該ブロックの現在の APT 内のデータをクリアし，メモリ参照履歴情報をとり直す．

3.3.2 アドレス予測機構の動作

以下にアドレス予測と RAT エントリの更新操作をブロックの実行状態毎に述べる．

ブロック開始時 分岐先のアドレスが判明したら，分岐先アドレスと RAT エントリの BI を比較し，比較結果が一致した RAT エントリによって予測を行う．比較結果が一致しなければ State Flag は Empty とし，新しい RAT エントリを生成する．生成した RAT エントリの BI にブロックの先頭アドレスを格納する．

ブロック実行時 ブロック実行開始時の RAT エントリの State により，以下のアドレス予測操作を行う．

1. State が Empty でない場合，RAT エントリの格納順に PRA と Stride を加算し，アドレス予測値とする．State が Steady であればデータのプリフェッチを行う．
2. 予測アドレスと参照アドレスを比較し，差分を計算する．差分の結果が 0 でなければ TF の値を A に設定する．
3. PAB 内に格納された PRA を最新の参照アドレスと比較し，比較結果が一致しなければ RAT に格納する．RAT エントリの State が Init であれば 2 で計算した差分も同時に RAT の Stride に記憶する．RAT エントリの State が Empty であれば Stride を 0 とする．
4. RAC をインクリメントし，参照アドレスを PAB に格納する．ブロック内で最初に参照アドレスを記憶する場合，当該 RAT エントリの APT 内での先頭アドレスを格納する．

ブロック終了時 1. RAC の値が 0 ならば RAT エントリを削除する．RAT エントリを削除しなかった場合以下の処理を行う．

2. RAT の N と TF の比較結果によって，図 4に従って State を設定する．ブロックの実行開始時の State によって，State フィールドの遷移は図 4のようになる．

各状態の意味は以下のとおりである．

Empty : ブロック開始時に RAT に予測対象ブロックに対応するエントリがなかったときの状態である．

Init : ブロック開始時の RAT の PRA と実行したブロックの参照アドレスから Stride を求める状態である．

Transient : PRA のみを更新し，Stride を更新しない状態である．

Steady : 実際に予測アドレスのプリフェッチを行ない，PRA を更新する状態である．

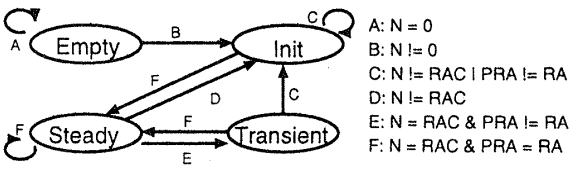

図 4: State の遷移

4 プリフェッチキャッシュ

提案するアドレス予測機構の一応用例として，アドレス予測を行うキャッシュである Prefetch Cache の構成例を示す．例に示す Prefetch Cache は通常のダイレクトマップの Normal Cache に 3 節で述べたアドレス予測機構と Prefetch Buffer，及びアドレス予測機構が生成した予測アドレスを保持するための Prefetch Queue を付加したものである．

本節では最初に Prefetch Cache の構成について説明する．次に Prefetch Buffer の動作について説明する．最後に Prefetch Cache へのデータの読み出し，及び書き込み方法について説明する．

アドレス予測機構によって計算された予測アドレスはまず Prefetch Queue に格納される．ブロック実行時に Prefetch Buffer 内のデータが無効化されると Prefetch Queue 内のアドレスのデータが Prefetch Buffer に格納される．Prefetch Buffer 内のデータがヒットするとヒットしたデータが Normal Cache に書き込まれる．

4.1 Prefetch Cache の構成要素

図 5 に Prefetch Cache と Prefetch Queue の構成図を示す．図 5 中に一点鎖線で示したのが Prefetch Cache である．Prefetch Buffer と Normal Cache のアドレスフィールドの破線は，参照アドレスの Tag の部分を表す．Prefetch Buffer の Tag は Prefetch Buffer ヒットを調べる場合に Prefetch Buffer 用の Tag として利用される．Normal Cache 移動時には Normal Cache と共通の Tag フィールドが Tag として残される．図 5 中の RAT は図 3 で示したアドレス予測機構である．

Prefetch Cache は以下の要素から構成される．

Normal Cache: 参照アドレスの offset を除いた部分を検索キーとするダイレクトマップのキャッシュである．以下の構成要素を持つ．

- **Address :** 参照アドレスを格納する．
- **Valid :** データの有効 / 無効を示す．
- **Data :** Address に格納されているデータを保持する．

Prefetch Buffer : 予測アドレスに基づいてプリフェッチしたデータを格納するための小容量のキャッシュである．今回の評価では Prefetch Buffer のエントリ数は 1 とした．これはプリフェッチのためのメモリのレイテンシを考慮していないためである．実際にはプリフェッチのためのメモリのレイテンシを考慮し，レイテンシの隠蔽のためエントリ数を増やすことが考えられる．

Prefetch Buffer は以下の要素を持つ．

- **Address :** 予測アドレスを格納する．
- **Valid :** データの有効 / 無効を示す．データが有効な場合はデータがプリフェッチされた際の RAT の BI の番号を保持する．
- **Data :** Prefetch Address に格納されているデータを保持する．

Address Selector : Prefetch Cache に対し，read/write を行なうデータアドレスを Predicted Address と参照アドレスの一方から選択するセレクタ．

Prefetch Queue は以下の要素から構成される．

Predicted Address : Prefetch Buffer に格納しきれなかった予測アドレスを保持する．

PQWP(Prefetch Queue Write Pointer) : Prefetch Queue に Predicted Address を書き込む場所を示すポインタ．

PQRP(Prefetch Queue Read Pointer) : Prefetch Queue から Predicted Address を読み出す場所を示すポインタ．

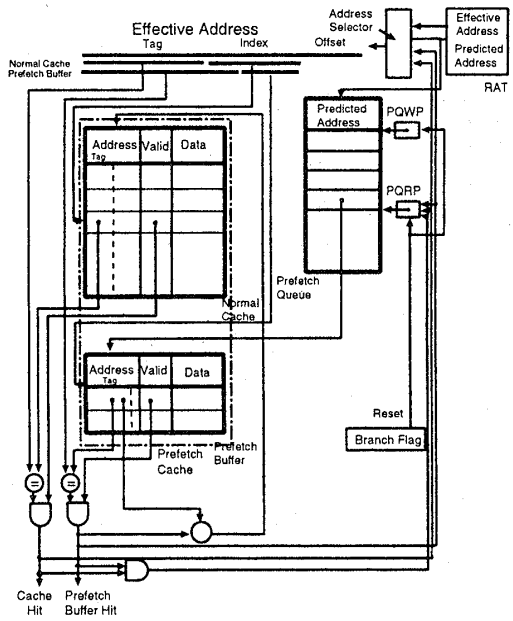

図 5: プリフェッチキャッシュ

4.2 Prefetch Cache の動作

本項では最初に Prefetch Cache 内でのデータの read/write について述べる．次に Prefetch Buffer 内のデータの無効化について述べる．

4.2.1 Prefetch Cache におけるデータの処理

Prefetch Cache では Prefetch Buffer と Normal Cache から構成され，Normal Cache はダイレクトマップキャッシュ，Prefetch Cache は小さなフルアソシアティブキャッシュとして動作する．それぞれのキャッシュアクセスに使用する参照アドレスの Tag，及び Index は Prefetch Buffer 用と Normal Cache 用の 2 種が生成される．Prefetch Cache に対し，read/write を行うデータアドレスは Address Selector によって選択される．Prefetch Cache 上のデータの検索時には，データアドレスから生成された Index と Tag を使用し，Normal Cache と Prefetch Buffer の各々から一致するアドレスが検索される．参照された参照アドレスが Normal Cache のみヒットした場合は，Prefetch Cache は通常のキャッシュと同様に動作する．ブロック実行時の参照アドレスが Prefetch Buffer 上に存在した場合，ヒットしたデータを Prefetch Buffer から読み出し，Normal Cache にコピーする．Prefetch Buffer 上のデータはアドレス予測が外れた場合は利用されない．予測が外れた場合も Normal Cache 上のデータは書き変わらないため，Prefetch Cache 全体のミス率の上限は Normal Cache と同じサイズのキャッシュのミス率に等

しい.

4.2.2 Prefetch Buffer データの無効化

Prefetch Buffer 上のデータは以下の場合に無効化される.

1. Prefetch Buffer がヒットし，データが Normal Cache にコピーされた場合
2. Prefetch Cache がミスした場合
3. Prefetch Buffer と Normal Cache の両方がヒットした場合
4. ブロック終了時

3 において，Prefetch Buffer と Normal Cache のヒットが同時に起こるのは，プリフェッチをキャッシュライン単位で行なった場合，RAT エントリの PRA に Normal Cache 上で衝突しないデータが複数回記憶されるために起こる.

1 以外の Prefetch Buffer 上のデータの無効化は，Prefetch Buffer 上にアドレス予測ミスを起こした Predicted Address が蓄積されるのを防ぐために行なわれる．アドレス予測ミスが起こった場合，アドレス予測ミス以降のアドレス予測をアドレス予測成功時と同じタイミングで行うため，キャッシュミスが起こった時に Prefetch Buffer 上のデータの無効化を行う．Prefetch Buffer 内のデータが無効化されたときに Prefetch Queue が空でない場合，Prefetch Queue から Predicted Address が読み出される.

以下では Prefetch Queue への予測アドレスデータ書き込みと Prefetch Queue からのデータ読み出し方法について述べる.

- Prefetch Queue へのデータ書き込み
  Prefetch Queue へのデータ書き込みは以下のように行なわれる.
  1. 前のブロックから残っている Prefetch Address を無効化するため，PQWP と PQRP がリセット
  2. RAT エントリから予測アドレスが計算されると，Prefetch Queue の PQWR が指すエントリと予測アドレスの Tag，Index を比較
  3. 結果が一致しなかった場合は Prefetch Queue の最後に予測アドレスを追加，PQWP をインクリメント
- Prefetch Queue からのデータ読み出し
  Prefetch Queue からのデータ読み出しは説明した Prefetch Buffer のデータが無効化された場合に起こる．読み出しは以下のように行なわれる.
  1. Prefetch Buffer 内のエントリの Valid の値が無効であれば，Prefetch Queue の先頭の Predicted Address，すなわち PQRP が指すエントリが読み出される.
  2. Prefetch Buffer に Predicted Address 内のデータの内容が書き込まれる.
  3. 書き込まれたデータの Valid は RAT の BI に設定される.
  4. PQRP がインクリメントされる.

4.3 参照アドレス履歴によるアドレス予測例

RAT エントリを用いてブロック単位でアドレス予測を行う例を示す．データのプリフェッチをキャッシュライン単位で行う場合について，同一アドレスへの参照の処理方法とストライド予測の方法について説明する.

4.3.1 ブロック実行時の RAT の更新

図 6(a) に示すプログラムが図 6の (b) のように実行された場合を考える．初期条件として

- このブロックに対応する RAT エントリは存在しない
- 108 番地の分岐は 2 回連続で成立する
- レジスタ r1 の値は 0

とする．四角内の数字はそれぞれの LD/ST 命令が参照した参照アドレスである．RAT エントリの変化を図 7に示す.

図 7の diff は実行中に参照された参照アドレスと PRA が一致しているか否かを示す.

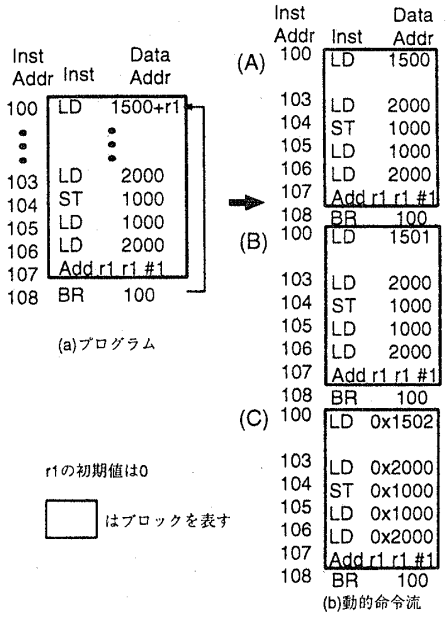

図 6: ブロック実行例

1 度目のブロック実行時 (図 6(A)，図 7(A) 参照) にはブロックに対応する RAT エントリが存在しない．このためブロックの開始アドレス (100 番地) を新しく生成した RAT エントリの BI に記憶する.

1 度目のブロックの実行では参照した参照アドレスをそのまま RAT の PRA に格納し，Stride を 0 に設定する．ただし 1000 番地への参照は，同一アドレスへの連続した参照であるため，RAT には 1 度しか格納しない．最初のアドレスの場所を RP に記憶する.

2000 番地への参照は同一アドレスへの参照であるが，PRA に 2 度格納される．これはダイレクトマップキャッシュにおいてキャッシュライン単位の予測を行なった場合，キャッシュライン上で参照アドレスが衝突すると再度プリフェッチが必要になるためである.

2 度目のブロック実行時 (図 6(B)，図 7(B) 参照) には参照された参照アドレスと予測アドレスの比較を行う．100 番地の命令で参照されている参照アドレスに注目する．予測アドレスは 1500，参照された参照アドレスが 1501 であるため，参照された実効アドレスと予測アド

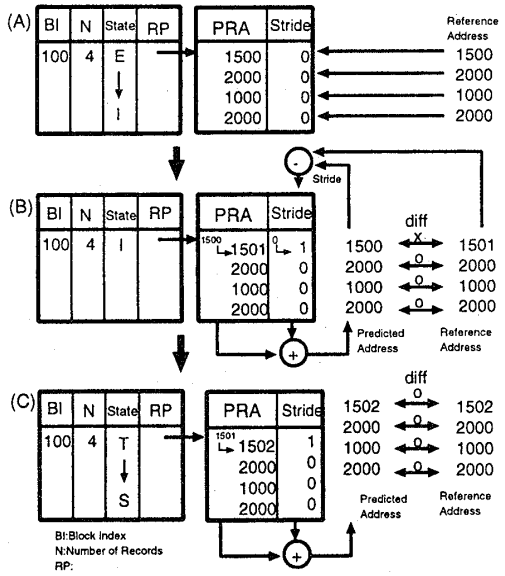

図 7: RAT エントリの更新

レスの比較結果が一致しない．予測アドレスが外れたため，2 度目のブロック実行後も State は Init である．

3 度目の実行開始時 (図 6(C)，図 7(C) 参照) には State が Init であるので参照された参照アドレスと予測アドレスの差分を計算し，RAT の Stride に書き込む．4 度目のブロック実行時には RAT による予測で正しいアドレスの予測が出来る．

4.4 評価環境

simplescalar のトレースを結果を用いて提案方式の性能評価を行なう．今回の評価では RAT エントリの数を無限個としている．ベンチマークプログラムとして simplescalar[3] 付属の compress と perl を使用した．compress の入力ファイルは nkf のプログラムとした．compress の実行命令数は 310 万命令，LD/ST 命令は 210 万命令，分岐は 30 万命令であった．perl のプログラムは入力ファイルに a2ps を使用した．perl の実行命令数は 460 万命令，LD/ST 命令は 170 万命令，分岐は 95 万命令であった．

4.5 評価結果

- Normal Cache のサイズを 32Kbyte とし，Prefetch Cache のラインサイズを変化させてミス率を測定した．compress での結果を表 1，perl での結果を表 2に示す．() 内はリプレースの回数である．

表 1: compress におけるラインサイズとミス率

Size[byte]	Cache Miss	Prefetch Cache Miss[%]
4	22(449000)	16(330000)
8	1.0(21000)	0.043(890)
16	0.10(2100)	0.0098(210)
32	0.028(590)	0.0011(23)
64	0.011(230)	0.00082(17)

- Prefetch Cache のラインサイズを 32byte とし，Normal Cache のサイズを変化させてミス率を測定した．compress での結果を表 3，perl での結果を表 4に示す．

表 2: perl におけるラインサイズとミス率

Size[byte]	Cache Miss	Prefetch Cache Miss[%]
4	1.4(24000)	0.76(13000)
8	0.38(6500)	0.14(2400)
16	0.080(1300)	0.0022(380)
32	0.002(360)	0.0095(160)
64	0.0056(95)	0.0054(92)

表 3: compress におけるキャッシュサイズとミス率

Size[Kb]	Cache Miss	Prefetch Cache Miss[%]
2	0.53(11000)	0.094(2000)
4	0.42(8800)	0.016(330)
8	0.16(3300)	0.016(330)
16	0.094 (2000)	0.0049(100)
32	0.028(590)	0.0011(23)

表 4: perl におけるキャッシュサイズとミス率

Size[Kb]	Cache Miss	Prefetch Cache Miss[%]
2	1.8(31000)	1.2(21000)
4	0.56(9800)	0.44(7500)
8	0.19(3200)	0.016(2600)
16	0.002(360)	0.0095(160)
32	0.002(360)	0.0095(160)

全ての評価で Prefetch Cache のミス率が下がっていることがわかる．特にラインサイズが小さい場合にプリフェッチの効果が顕著であった．

5 まとめ

ブロックを単位としてプリフェッチを行う予測機構を提案し，その構成例を示した．RAT の数に制限を設けなかった場合，perl，compress ともにミス率が低下した．このことから提案方式がミス率の削減に有効であることがわかった．今回の評価では RAT エントリの数を無限として評価した．RAT エントリ数を制限したときの性能評価と RAT エントリのリプレース方法に関する検討，及び他方式との比較が今後の課題である．

6 参考文献

参考文献

[1] Chen, T-F. and Baer, J-L.: “Effective hardware-based data prefetchig for high-performance processors, IEEE Trans.Comput.,Vol44,No5,pp.609-623(1995)” IEEE Trans. Comput., vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[2] 吉瀬謙二，坂井修一，田中英彦: “2 レベル・ストライド値予測機構の可能性検討” 情報処理学会論文誌,Vol.41,No.5,pp1340-1349

[3] Doug Burger and Todd M.Austin: “The SimpleScalar ToolSet, Version2.0”, University of Wisconsin-Madison Computer Sciences Department Techmical Report #1342, June, 1997.