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

2009－ARC－182 (16)
2009－HPC－119 (16)
2009／2／27

メモリ・アクセス・パターンを利用した高精度ハードウェア・プリフェッチ手法

堀部悠平† 張鵬† 小笠原嘉泰†
三輪忍† 中條拓伯†

キャッシュ・ミス率を改善するには，プリフェッチを行うのが効果的である．しかし，現在実用化されているハードウェア・プリフェッチ手法は，規則的なメモリ・アクセス・パターンに対しては有効であるものの，不規則なアクセス・パターンに対しては効果がないことが多い．我々は，不規則なパターンに対応するため，プログラムの実行パスに着目してプリフェッチを行うことを考えている．今回，実行パスを考慮したプリフェッチャをシミュレータ上に実装し，評価を行った．本稿ではその結果を報告する．

Hardware prefetching to achive high accuracy according to memory access patterns

Yuhei Horibe ,† Hou Tyou ,† Yoshiyasu Ogasawara ,† Shinobu Miwa † and Hironori Nakajo†

Prefetching is an effective technique to improve a cache miss rate. Recently, practical hardware prefetching is effective for regular patterns of memory accesses, however, it is ineffective for irregular patterns. We propose a new hardware prefetching technique focusing on a execution path of a program. We implemented it on a simulator, and evaluated it. This paper shows the results of the evaluation.

1. はじめに

プロセッサの動作周波数の改善率とDRAMの動作周波数のそれとの違いから，プロセッサとメイン・メモリ間の速度差は拡がる一方となっている．近年のプロセッサでは，通常，ALUのレイテンシは1サイクルなのに対し，メモリ・アクセスには数百サイクルを要する．そのため，ロード命令がメモリへのアクセスを開始すると，それに依存する命令の実行が大幅に待たされてしまい，**命令レベル並列性**は大きく低下してしまう．**メモリ・ウォール問題**として知られるこの問題は，Out-of-Orderスーパスカラ・プロセッサにおける重要な問題の1つとなっている．

大きなメモリ・アクセス・レイテンシを隠ぺいするため，通常，メモリ上のデータの一部を保持する**キャッシュ**が用いられる．アクセスするデータがキャッシュに存在（**ヒット**）すれば，メモリ・アクセスのレイテンシはキャッシュのそれで済む．ただし，**ミス**した場合には，下位のキャッシュへ改めてアクセスする，(ヒットするものとして）投機的に発行してしまった命令を再発行するなどのペナルティを被ることになる．

キャッシュ・ミスを減らすには**プリフェッチ**を行うのが効果的である．実際のメモリ・アクセスが発生する以前に，近々アクセスされると予想されたデータを，下位のキャッシュから上位のものへと移動させておく．移動させたデータへのアクセスが実際に発生すれば，ミスするはずだったものがヒットすることになる．

一方，実際にアクセスしたデータと移動したものとが異なっていれば，プリフェッチの効果はない．むしろ，有効なキャッシュ・ラインに代わって間違ったラインをキャッシュしたことで，ミス率は低下してしまう．そのため，プリフェッチを行う際はその精度が重要である．

プリフェッチには，**ソフトウェア**によるものと**ハードウェア**によるものとがある．

ソフトウェア・プリフェッチでは専用命令を用いてプリフェッチが行われる．プリフェッチ命令は，プリフェッチ対象のデータをロードする命令の数命令前に，プログラマ，あるいは，コンパイラによって追加される．ロード命令に先行してプリフェッチ命令が実行されることで，プリフェッチが行われる．任意のメモリ・アクセスに対してプリフェッチできるため，ハードウェアによるものと比べ，高精度なプリフェッチも可能である．ただし，専用命令を用いているため，バイナリ互換性に問題がある．

一方，**ハードウェア・プリフェッチ**は，後述するように，ソフトウェアに比べて単純なものが多い．そのため，特徴を持ったメモリ・アクセス・パターンでは効果的に機能するもの[2),4)~7),10)]，すべてのメモリ・ア

† 東京農工大学
Tokyo University of Agriculture and Technology

クセスをプリフェッチの対象にした場合は精度が十分でない．複雑で高精度なハードウェア・プリフェッチ手法[3]もあるが，その多くはマルチ・スレッド・プロセッサに限定されたものである．

我々は，シングル・スレッド・プロセッサにも適用できる，単純，かつ，高精度なハードウェア・プリフェッチ手法の開発を目的とする．従来のプリフェッチ手法とは異なり，すべてのメモリ・アクセスを対象とした場合でも有効な手法の実現を目指す．

ロード命令のメモリ・アクセス・パターンは，プログラムの実行パスに依存すると考えられる．すなわち，あるパス上でロードされたデータは，そのパスが再び実行された場合，再びロードされる可能性が高い．

そこで今回，パス毎のメモリ・アクセスの履歴を利用するプリフェッチャを実装し評価を行った．本稿ではその結果について述べる．以下，まず次章において，従来のハードウェア・プリフェッチ手法について詳しく説明する．続く3章において，パス毎のメモリ・アクセスの履歴を利用したプリフェッチ手法について述べる．評価は4章で行い，5章でまとめる．

2. ハードウェア・プリフェッチ

前述のように，現在，シングル・スレッド・プロセッサにおいて実用化されているハードウェア・プリフェッチ手法は，単純なものが多い．それらは，以下で述べるように，ある特徴を持ったメモリ・アクセス・パターンにおいては有効に機能する．しかし，そうした特徴を持たないパターンに対してはまったく効果がない．複雑で高精度なプリフェッチ手法も存在するが，その多くはマルチ・スレッド・プロセッサを前提としたものである．

以下では，まず2つの単純なプリフェッチ手法について述べる．これらの手法は単純であり，すべてのメモリ・アクセスを対象にプリフェッチした場合，精度が十分でない．これらの適用範囲は極めて限定的である．広範囲に適用できる，複雑なプリフェッチ手法については，続く2.2節で述べる．

2.1 単純なプリフェッチ手法

以下では，代表的なハードウェア・プリフェッチ手法である，**ネクスト・ライン・プリフェッチ**，および，**ストライド・プリフェッチ**，それぞれについて詳しく述べる．

2.1.1 ネクスト・ライン・プリフェッチ

ネクスト・ライン・プリフェッチは非常に単純である．単に，現在フェッチしているラインの次のラインをプリフェッチする．すなわち，空間的に連続するラインが時間的に連続してアクセスされる可能性が高いことを想定し，プリフェッチを行うのである．

この性質は命令によく当てはまる．分岐予測器が分岐を不成立と予測し続ける限り，連続するアドレスの命令は，連続してフェッチされる．すなわち，現在フェッチしているラインの次のラインに含まれる命令は，近々フェッチされる可能性が高い．そのため，次のラインをプリフェッチしておけば，高い確率でキャッシュ・ミスを防ぐことができる．

しかし，データにはこうした性質はあまりない．ネクスト・ライン・プリフェッチは，命令のストリーム・バッファ[7]として実装されることが多い．

2.1.2 ストライド・プリフェッチ

次いで，ストライド・プリフェッチ[2]について述べる．行列の列方向のアクセスのように，データがある一定のストライドでアクセスされている場合，次もそのストライドでアクセスされる可能性が高い．そうした場合，そのストライドでプリフェッチを行えば，後続のメモリ・アクセスはキャッシュにヒットすることになる．

実際，Intel Pentium 4 ではストライド・プリフェッチが行われている[4]．Pentium 4 では，ハードウェア・プリフェッチャが，L2 キャッシュにミスしたアクセス群を監視する．そして，それらが一定のストライドを持っていることを検知すると，現在アクセスしているアドレスの 256B 先のデータからプリフェッチを開始する．

ストライド・プリフェッチは，上述のように，一定のストライドを持ったアクセス・パターンに対しては有効である．しかし，当然のことながら，ストライドを持たないパターンに対してはほとんど効果がない．

2.2 複雑なプリフェッチ手法

上述した2つの手法は単純で，メモリ・アクセス・パターンに規則性がある場合にしか効果がない．そこで，不規則なアクセス・パターンにおける効果的なプリフェッチ手法の実現を目指し，いくつかの研究が行われている．特に近年では，マルチ・スレッド・プロセッサの普及にともない，マルチ・スレッドを利用した手法の研究が盛んである．

不規則なメモリ・アクセス・パターンにおいて，次にアクセスされるデータを予測する方法として，マルコフ・モデルを基にした表を用いて予測する方法が提案されている[6]．あるデータがキャッシュにミスした場合に，次にミスする確率が高いデータを表に記録する．次にミスするデータが複数ある場合は，それぞれの確率を記録する．再度そのデータがキャッシュ・ミスした場合は，表を参照し，次にミスする確率が高いデータをプリフェッチする．

SMT（Simultaneous Multi-Threading）プロセッサ[8]におけるプリフェッチ手法として，**Speculative Precomputation**[3]が提案されている．Speculative Precomputation では，**ヘルパー・スレッド**と呼ばれる，プリフェッチ専用のスレッドを用いてプリフェッチを行う．メイン・スレッドで近々ロードが発生すると予想されると，

ロードされるアドレス（またはそれを計算するのに必要なデータ）を予測し，それをプリフェッチするためのスレッドである，ヘルパー・スレッドを生成する．メイン・スレッドでロードが実行される前に，ヘルパー・スレッドで先にロードが実行されれば，メインのロードはキャッシュにヒットすることになる．アドレスなどの予測は，ロードが実行される直前のマシン状態にもとづいて行うため，それが正しい可能性は高い．

ヘルパー・スレッドを CMP に応用した研究もある[9]．並列プログラムには，データ並列性が低いものもある．そうしたプログラムを CMP で実行する場合，プログラムをコア数分のスレッドに分割し，各コアに割り当てても，スケーラブルな性能を示さない．そうするよりも，分割するスレッド数を実際のコア数以下に抑え，空いたコアでヘルパー・スレッドを実行した方がよい．

我々は，シングル・スレッド・プロセッサにも適用できる，単純，かつ，高精度なプリフェッチ手法の実現を目指す．次章からはその詳細について述べる．

3. 実行パスを考慮したハードウェア・プリフェッチ手法

分岐命令の分岐結果に代表されるように，プログラム実行時の事象の多くは，実行パスに依存することが知られている．あるパスが実行され，ある分岐命令に到達した場合，その分岐の結果は，過去に同様のパスをたどってそれが実行された時の結果と同じになる可能性が高い．そのため，gshare 予測器を含め多くの分岐予測器では，実行パスの情報を利用して予測精度の向上を図っている．

これと同様のことが，ロード命令のメモリ・アクセス・パターンについても言えると考えられる．すなわち，同じパスが実行されて同じロード命令に至った場合，再び同じデータがアクセスされる可能性は高いと予想される．

そこで我々は，**実行パスを考慮して，プリフェッチを行う手法**（**Execution-Path Based Prefetch**，以下 **EPBP** とする）を提案する．EPBP の構成を**図 1** に示す．

以下では，**図 2** のサンプル・プログラムを例に，EPBP の動作を説明する．同図において，コロンの右は命令を，左はそのアドレスを表す．例えば，ロード命令 “lw r3 0(r6)” のアドレスは “0*x*4004*e*8” である．

EPBP では，**Reference Prediction Table**（以下 **RPT** とする）を用いて予測を行う．RPT は，過去にロードされたデータのアドレスをエントリとする表である．

各データは，それをプリフェッチするための**トリガとなる命令**のアドレス，および，**グローバル分岐履歴**によってタグが付けられている．タグには，トリガ命令のアドレスとグローバル分岐履歴とを連結，もしくは XOR したものを用いる．図 1 は XOR した場合である．

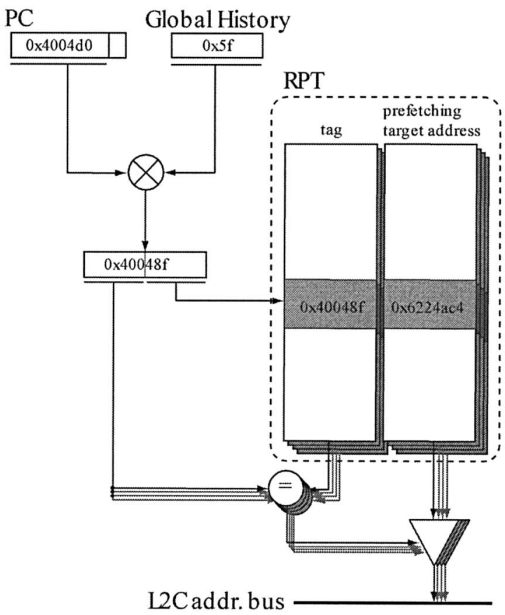

図 1 EPBP の構成

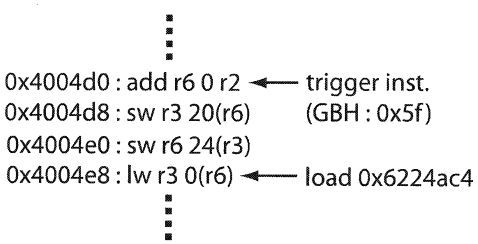

図 2 プログラム例

トリガ命令は，基本的には，in-order 上で，そのデータをロードした命令の数命令前の命令とする．例えば，図 2 では，ロード命令の 3 命令前の命令（add）をトリガ命令としている．数命令前の命令をトリガとするのは，ロードが実行されるまでにプリフェッチを完了させるためである．ただし，今回の評価では，簡単のため，プリフェッチはただちに完了するものとしている．

EPBP では，以下のようにしてプリフェッチ行う．まず，命令がフェッチされると，そのアドレス，および，その時のグローバル履歴によって RPT を参照する．参照が RPT に：

ヒットした場合　現在実行中のパスは過去に実行されたパスであり，そのパス上のロード命令が再び実行される可能性が高い．そこで，ヒットしたエントリのアドレスを RPT から読み出し，そのアドレスをプリフェッチする．

ミスした場合　何もしない．

RPTの更新は，ロード命令がコミットされた際に行う．ロード命令のコミットの際，それがロードしたアドレスを，トリガ命令のアドレスとそれがフェッチされた際のグローバル履歴とによって，RPTに登録する．RPTのエントリはLRUでリプレースする．エントリを有効に活用するため，キャッシュにヒットしたデータはRPTに登録せず，ミスしたデータのみを登録する．

図2のコードを実行した場合，EPBPの動作は以下のようになる．例えば，図のコードが初めて実行された際，ロード命令がアドレス“0*x*6224*ac*4”のデータをロードしたとする．すると，そのロードがコミットされる際，アドレス“0*x*6224*ac*4”がRPTに登録される．タグは，トリガ命令であるaddのアドレス“0*x*4004*d*0”と，それがフェッチされた際のグローバル履歴“0*x*5*f*”とをXORしたものである．

実行が進み，やがて，トリガ命令であるaddが再びフェッチされたとしよう．すると，そのアドレス“0*x*4004*d*0”とグローバル履歴“0*x*5*f*”とを用いて，RPTを参照する．RPTへの参照がヒットすると，前回ロードされたデータのアドレス“0*x*6224*ac*4”が判明するので，それをプリフェッチする．

このようにして，EPBPは，同じパスが実行された場合にロードされるであろうデータを予測する．

4. 評　　価

EPBPをシミュレータに実装し，評価を行った．以下ではその結果を述べる．

4.1 評価環境

次の2つのモデルについて評価した．

BASE　プリフェッチをまったく行わないモデル

EPBP　前章で述べた，EPBPを用いたモデル．プリフェッチはL1データ・キャッシュに対して行う．ただし，プリフェッチャが予測したデータがL2キャッシュに存在しなかった場合は，プリフェッチは行われない．

前章で述べたように，EPBPでは，本来，ロード命令の数命令前の命令をトリガ命令とする．これは，通常，プリフェッチが要求されてから完了するまでにはある程度のサイクル数を要するからである．

しかし，今回の評価では，簡単のため，プリフェッチのレイテンシを無視することにした．すなわち，トリガ命令は，そのデータをアクセスした，ロード命令自体としている．ロード命令がフェッチされると，そのアドレスとグローバル履歴とによってRPTを参照する．そして，それがヒットした場合にプリフェッチが行われる．

EPBPに用いるグローバル履歴の長さは，0bから32bまで変化させた．RPTのハードウェア量は，8KB, 16KB, 32KB, ∞の4つについて測定した．タグは，有限のものについてはトリガ命令のアドレスとグローバル履歴とをXORしたものを，無限のものについては連結したものを用いている．有限のRPTのマッピングはセットアソシアティブとし，ウェイ数は4とした．

2つのモデル（BASE，EPBP）をSimpleScalarツール・セット（Ver.3.0）[1]のsim-outorderシミュレータ上に実装し，評価を行った．評価に使用したプロセッサのパラメータを表1に示す．L1データ・キャッシュは8KBで，ヒット・レイテンシを1 cycleとした．また，L2キャッシュは命令・データ共有の512KBととし，レイテンシは10 cycleとした．

ベンチマーク・プログラムには，SPEC CINT 2000より，プリフェッチによるキャッシュ・ミス率の増減が大きい，6本のプログラムを使用した．使用したプログラムを**表2**に示す．入力セットはtrainとした．各プログラムは，最初の1G命令をスキップし，続く20M命令を実行した．

4.2 評価結果

以下では，まずハードウェア量が無限の場合の結果について述べ，次いで有限の場合について述べる．

4.2.1 ハードウェア量が無限の場合

グローバル履歴を32b，RPTのサイズを無限とした場合の，プログラム毎のキャッシュ・ミス率を**図3**に示す．グラフの横軸はプログラム名を，縦軸はミス率を表す．2本の棒グラフは，左がBASE，右がEPBPである．

グラフより，RPTのサイズが無限の場合は，すべてのプログラムにおいて，EPBPによりミス率が改善する．その効果は特に*perlbmk*において大きい．*perlbmk*では，2.72%（BASE）から0.69%（EPBP）へと，ミス率が2.03ポイント改善する．平均すると，5.92%（BASE）から4.82%（EPBP）へと，1.10ポイント改

表1　プロセッサ構成

parameter	remarks
way	4
ruu/lsq size	32/16
exec. unit	INT:2, FP:2, LD/ST:2
branch predictor	8KB gshare (12 hist.)
BTB	512 set, 4-way
RAS	8-entry
L1 I-cache	8KB, 32B/line, 2-way, 1 cycle
L1 D-cache	8KB, 32B/line, 2-way, 2 cycle
L2 cache (unified)	512KB, 64B/line, 4-way, 10 cycle
memory access latency	100cycle

表2　SPEC CINT2000ベンチマーク・プログラム

program	input
175.vpr	net.in arch.in
176.gcc	cp-decl.i
197.parser	train.in
253.perlbmk	scrabbl.in
255.vortex	lendian.raw
300.twolf	train.blk train.cel train.net train.par

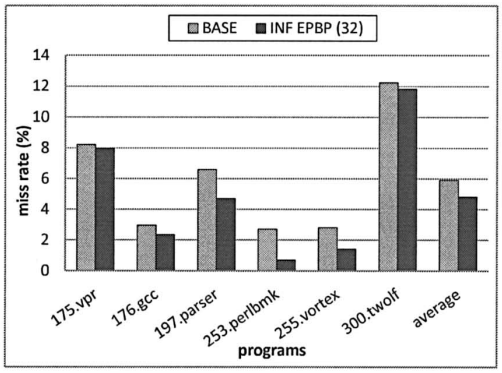

図 3 プログラム毎のキャッシュ・ミス率（グローバル履歴：32b）

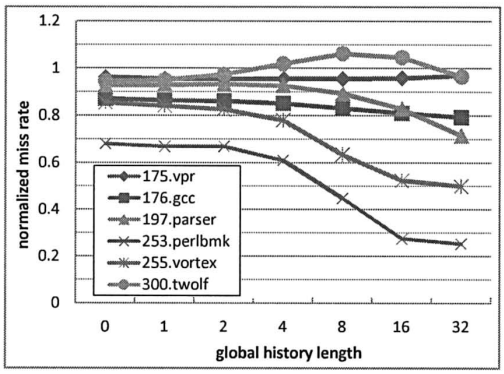

図 5 プログラム毎の履歴長に対するキャッシュ・ミス率

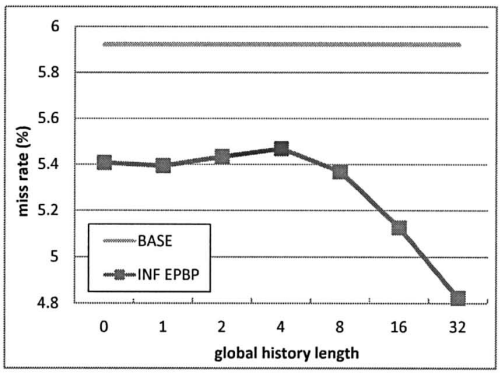

図 4 履歴長に対する平均キャッシュ・ミス率

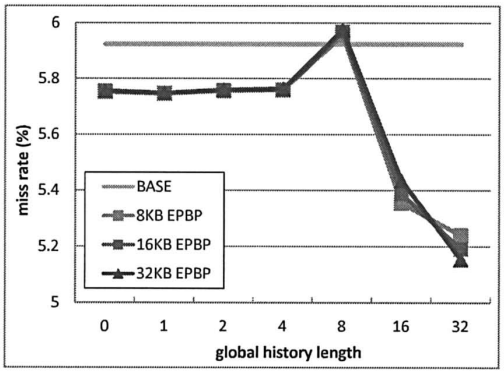

図 6 ハードウェア量を変化させた場合の平均キャッシュ・ミス率

善する．

グローバル履歴の長さを 0b から 32b まで変化させた際の平均キャッシュ・ミス率を，**図 4** に示す．グラフの横軸は履歴長，縦軸は平均ミス率である．2 本のグラフは，直線が BASE，折れ線が EPBP である．

グラフからわかるように，平均キャッシュ・ミス率は，どのような長さのグローバル履歴を用いようとも EPBP の方がよい．また，平均ミス率は，基本的には，グローバル履歴を長くする程改善する．平均ミス率は，グローバル履歴をまったく用いない場合は 5.41% だったのに対し，32b の履歴を用いた場合は 4.82% と，0.59 ポイント改善している．

グラフより，平均ミス率は，32b においてもまだ飽和していない．履歴を 32b よりも長くすればミス率はさらに改善すると予想されるが，この点については今後検証の必要があろう．

図 5 に，プログラム毎の履歴長に対するキャッシュ・ミス率を示す．各プログラムのミス率は，それぞれ，BASE のそれで正規化してある．横軸は，先程と同様，履歴長である．

グラフからわかるように，EPBP の効果は，プログラムによって差がある．*vpr* や *twolf* のように，ほとんど効果がない，あるいは，悪化してしまうプログラムも一部ある．

しかし，*vortex* や *perlbmk* といったプログラムにおいては効果が大きい．特に *perlbmk* では，正規化されたミス率は，16b の時で 0.276，32b の時で 0.254 であった．すなわち，テーブル・サイズが無限の場合，キャッシュ・ミス率は，BASE に対して最大 72.4 ~ 74.6% 改善する．

4.2.2 ハードウェア量が有限の場合

ハードウェア量を 8KB, 16KB, 32KB と変化させた場合の，履歴長に対する平均キャッシュ・ミス率を**図 6** に示す．グラフの見方は図 4 と同様である．マッピングは，前節で述べたように，4 ウェイ，セットアソシアティブとした．なお，タグの分のハードウェア量は評価に含めていない．

グラフより，いずれのハードウェア量においても，32b の履歴を用いた時の平均ミス率が最もよい．平均ミス率は，32KB の場合で 5.12%，8KB の場合でも 5.24% にまで改善する．それぞれ，BASE のそれと比べ，0.683，0.768 ポイント改善されている．

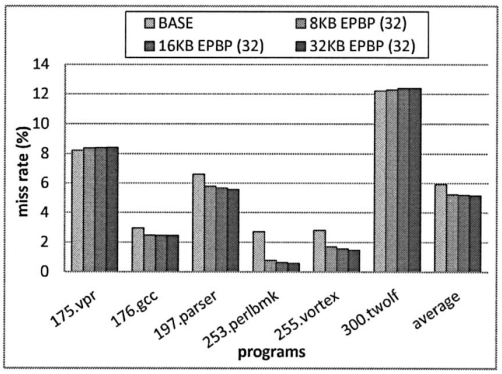

図 7 プログラム毎のミス率

各ハードウェア量におけるプログラム毎のミス率を**図 7** に示す．グラフの見方は図 3 と同様である．4 本の棒グラフは，一番左が BASE を表し，以後順に，8KB，16KB，32KB の EPBP を表す．履歴長はいずれも 32 とした．

グラフより，無限の場合と同様，提案手法は *perlbmk* で最も効果が高い．*perlbmk* においては，RPT が 32KB あれば，ミス率が 0.59% にまで改善する．BASE に対する改善率は 78.3% にものぼる．平均改善率は，8KB で 11.5%，32KB で 13.0% であった．このように，EPBP によって，ミス率は大幅に改善する．

5. おわりに

実用化されているハードウェア・プリフェッチ手法は，規則的なメモリ・アクセス・パターンに対しては有効に働くものの，不規則なパターンに対しては効果がない．不規則なパターンに対するプリフェッチ手法についても研究されているが，その多くは複雑である．

そこで本稿では，単純な構成で高い精度を実現するプリフェッチ手法 EPBP を提案した．EPBP では，ロード命令がアクセスするデータは実行パスに依存するものと考え，パス毎にアクセスされたデータを記録しておく．そのパスが再び実行された場合には，記録された過去のアクセス履歴をもとにプリフェッチする．EPBP を用いて L1 データ・キャッシュへのプリフェッチを行った結果，まったく行わない場合と比べ，キャッシュ・ミス率は平均 13.0%，最大 78.3% 改善することがわかった．

今後の課題としては，以下の点が挙げられる．

プリフェッチのコストの正確な見積もり　プリフェッチャが下位のキャッシュにデータを要求してからそれが上位へ反映されるまでには，実際には，ある程度のサイクル数を要する．既存のポートを利用してプリフェッチする場合には，競合が発生するため，そのサイクル数はさらに大きくなる．今回は簡単のためトリガ命令をロード命令として評価したが，実際には両者の距離を離さなければならない．そうした場合，プリフェッチ精度は低下するため，その場合でも EPBP が有効かどうかを検証する必要がある．

既存手法との比較　今回は行うことができなかったが，既存手法との定量的な比較は必要である．特に，本手法と同様，不規則パターン向けのプリフェッチ手法である，マルコフ・モデルを利用した手法との比較は必須である．ストライド・プリフェッチなどの一般的なプリフェッチ手法との比較も行いたい．

その他の課題　今回はキャッシュ・ミス率しか評価していないが，プロセッサ全体の性能への影響も評価する必要がある．また，SPEC CINT 2000 中の 6 本のプログラムしか評価が間に合わなかったが，その他のプログラムについても評価したい．

今後，これらの点について研究を進めていく予定である．

謝辞　本研究の一部は文部科学省 共生情報工学推進経費による．

参考文献

1) Burger, D. et al.: Evaluating Future Microprocessors: The SimpleScalar ToolSet, Technical Report CS-TR-1308, Univ. of Wisconsin-Madison (1996).
2) Chen, T.-F. et al.: Reducing memory latency via non-blocking and prefetching caches, *ASPLOS-5*, pp. 51–61 (1992).
3) Collins, J. D. et al.: Dynamic Speculative Precomputation, *MICRO-34*, pp. 306–317 (2001).
4) Hinton, G. et al.: The Microarchitecture of the Pentium 4 Processor, Technical Report Q1, Intel Technology Journal (2001).
5) Intel Corp.: *A detailed look inside the NetBurst micro-architecture of the Intel Pentium 4 processor* (2000).
6) Joseph, D. et al.: Prefetching using Markov predictors, *ISCA-24*, pp. 252–263 (1997).
7) Jouppi, N. P.: Improving direct-mapped cache performance by the addition of a small fully-associative cache prefetch buffers, *ISCA-25*, pp.388–397 (1998).
8) Tullsen, D. M. et al.: Simultaneous multithreading : Maximizing on-chip parallelism, *ISCA-22*, pp. 392–403 (1995).
9) 今里賢一ほか: 演算/メモリ性能バランスを考慮した CMP 向けヘルパースレッド実行方式の提案と評価, 情報処理学会研究報告 2008–ARC–178, pp. 75–80 (2008).
10) 山本哲宏ほか: 先行実行を利用したロード命令のレイテンシ削減および正確なスケジューリング手法, *SACSIS 2006*, pp. 403–410 (2006).