Vol.2009-ARC-184 No.19
2009/8/5

追い出しラインに着目した プリフェッチスロットリング手法

入江英嗣[†1]　本城剛毅[†1]　平木敬[†1]

シングルスレッド性能の限界が指摘される中，プリフェッチは軽量なハードウェアで IPC を大きく向上させている．近年のプリフェッチ手法は大量のアドレスを想起する傾向にあり，これは多くのプログラムで有効となっている．しかし，その一方で，僅かなプリフェッチに対しても，キャッシュポリューションを生じて性能を大きく下げてしまうプログラムが存在する．このため，プリフェッチの積極度を適切に制御する技術が研究されるようになってきた．

本論文では，プリフェッチの成功率に着目していた従来手法とは異なり，キャッシュの余裕度に着目した “Cache Metabolism” 値を提案し，軽量化と高性能の双方を実現する．提案手法は，置き換えラインを観測することによって CM 値を推測し，その値を元にプリフェッチ積極度を制御する．ハイブリッド・ストライド・プリフェッチャーをベースとした評価では，SPEC2006 全体の調和平均で，最も良い積極度に対しさらに 2.3%の性能向上を得た．

Replaced Line Guided Prefetch Throttling

Hidetsugu Irie,[†1] Goki Honjo[†1] and Kei Hiraki[†1]

Although single-thread performance is believed to be saturated, prefetchers still significantly improve IPC with simple hardware. Latest prefetches are tending to generate a large amount of predicted address which are effective for many programs. However, there are some sensitive programs that degrade performance severely by cache pollution for even small amount of prefetches. Thus, techniques thst adjust prefetching aggressiveness are being researched lately.

In this paper, “Cache Metabolism” metric is proposed and both performance and cost are improved. Unlike previous works which focused on accuracy or coverage of prefetches, proposed metric focused on the room of the cache at that time. Proposed technique estimates CM by replaced lines, and throttles the aggressiveness of prefetch. In harmonic mean for the total SPEC 2006 CPU programs, proposed technique achieved 2.3% performance improvement against the prefetcher with best fixed-aggressiveness when implemented with the hybrid stride prefetcher.

1. はじめに

メモリレイテンシを隠蔽してコアの待機時間を減らすことは，マイクロアーキテクチャの主要な課題である．特に，近年ではオフチップメモリレイテンシは数百サイクルに達し，数十エントリの命令ウィンドウによるアウト・オブ・オーダ実行では軽減は難しい．数百サイクルを隠蔽してコアの稼働率を高める技術として，広大な命令ウィンドウによる超 ILP 実行[1]，複数スレッドから並列性を取り出す SMT 実行[2]，アドレス予測によるプリフェッチ[3),4)]，先行実行による slipstream[5)] など様々な方式が検討されてきた．しかし，総電力消費やホットスポットが設計を制限する近年の傾向では，プリフェッチの軽量軽快さは特に魅力的なものとなっている．シングルスレッド性能の限界が指摘される中，単純なストリーム予測[6)] やストライド予測[7)] によるプリフェッチ機構は IPC を容易に向上させるため，商用プロセッサで一般的な技術となっている[8)]．

プリフェッチには長い研究の歴史があり，アーキテクチャのバランス変化に伴って常に新しい技術が提案されている[4),7),9)–16)]．従来，キャッシュやバンド幅は最も貴重な資源であり，プリフェッチでは精度 (*accuracy*) が重視されていた．しかし，近年では最下層キャッシュの大型化に伴い，多少無駄なラインを載せてでも適用率 (*coverage*) を上げる事が有効となってきており，プリフェッチ候補を大量に想起する手法が多く提案されている[10),11),16)]．

積極的な想起に基づくプリフェッチは，一部のアプリケーションで有効に働く一方で，一部のアプリケーションではキャッシュを攪乱して性能を激減させてしまう (キャッシュポリューション)．また，無駄なリクエストが本来のデマンドアクセスの数倍以上に及ぶこともあり，メモリトラフィック量を増加させ，バス競合や DRAM のバンク競合の原因となってしまう．このため，無駄なプリフェッチを検出して，プリフェッチの積極度をスロットリングする手法が検討されている[15),17),18)]．しかし，これらの研究ではポリューション予測のための大量のハードウェアや，閾値の適切な設定などを必要とする．大量想起プリフェッチの登場を受け，スロットリング手法は研究が始まったばかりであり，更なる改良を必要としている．

†1 東京大学情報理工学系研究科
Graduate School of Information and Technology, the University of Tokyo

本論文では，どのようなプリフェッチ手法にも組み合わせることができ，簡易且つ高性能なスロットリング手法を提案する．従来手法と異なり，提案手法はプリフェッチの成功度ではなく，キャッシュの余裕度に注目する．また，キャッシュから追い出されるラインのみを監視して余裕度を推測することにより，ハードウェア軽量化と推測精度向上を得る．推測した指標を元に積極度制御を行い，プリフェッチによる性能向上を最大化する．

以下，本論文は次のように構成される．第 2 章では関連するプリフェッチスロットリング手法について述べる．第 3 章では提案手法の核となる，*cache metabolism* の考え方と，ハードウェアによる計測手法について述べる．第 4 章ではスロットリング機構について述べる．第 5 章で評価環境を示し，第 6 章で評価と議論を行う．第 7 章でまとめを述べる．

2. 関連研究

Hur ら[14] の adaptive stream では，平均的なストリーム長を動的に推測し，その長さに達したところで該当ストリームのプリフェッチを停止する．通常のストリームプリフェッチでは，常にミスアドレスから distance 先のアドレスまでリクエストをかけるため，ストリームの終了ポイントで distance 個のリクエストが無駄となる．adaptive stream では，この終了時の無駄を減らし，短いストリーム長のアクセスパターンに対して有効なプリフェッチを行うことができる．また石井ら[16] の AMPM では，パターンマップとミスアドレスの対比からポリューション発生を推測し，アドレス想起を止めたり，リクエスト数を増減させるなどの制御が行われている．これらのフィードバック手法は，それぞれのアドレス想起手法に固有の最適化がなされたものである．

想起手法を選ばず適用できるフィードバック手法として，Zhuang ら[17] はプリフェッチ有効性予測を提案している．彼らの手法では，ポリューションフィルタと呼ばれる 1KB 程度のテーブルを追加し，そこに，過去のプリフェッチの有効度を 2bit 飽和カウンタで保持する．ポリューションフィルタはアドレスまたは PC でアクセスされ，カウンタの値が閾値以下であれば，そのプリフェッチは止められる．有効度の学習は，ラインリプレース時に，アクセスされなかったプリフェッチラインを検出することで行う．

プリフェッチ積極度のスロットリングを体系的に行う試みとして，Srinath ら[15] は，各期間毎にプリフェッチの精度，遅さ，ポリューションの三つの指標を算出し，その値を元に次の期間の積極度を制御する手法を提案している．精度はプリフェッチ総数に対するプリフェッチラインヒット回数を数えることで算出し，遅さは MSHR でヒットしたプリフェッチラインの数を数えることで算出する．また，キャッシュから追い出されたラインの情報を圧縮して保持し，ポリューションの検出に用いる．これら 3 つの指標についてそれぞれ閾値を設定し，それぞれの高中低の組み合わせによって，積極度を 1 ずつ増減させる．閾値および増減ポリシーは適切に設定する必要がある．また，Ebrahimi ら[18] は，複数のプリフェッチャーを併用する時のスロットリング手法を提案している．Srinath らの手法と同様に，期間中の精度や適用率を計測し，閾値と増減表から次の期間の積極度を決定する．

しかし，これらの手法では，プリフェッチャー本体と比較して，大量のハードウェアと複雑な制御を必要とする．Zhuang らのポリューションフィルタ，Srinath らの追い出しライン保持，Ebrahimi らの追加プリフェッチビットは，いずれもキロビット単位の容量を必要とする．予測の有効度をさらに予測するためにこれらの容量と複雑な制御を加えることは，プリフェッチの利点である軽量軽快さとは相反している．また，精度や適用率の閾値は，発見的に決定されており，多様なプログラムに対する適切な閾値設定が課題となっている．

3. Cache Metabolism 値

3.1 キャッシュポリューションの本質

従来，キャッシュは貴重な資源であり，プリフェッチには高い精度が求められてきた．しかし，近年の大容量キャッシュでは，いくらか無駄なプリフェッチを生じてでも適用率を上げることが有効であり，大量想起プリフェッチが登場している．つまり，キャッシュには無駄なラインを載せる余裕が生じている．キャッシュポリューションは，その時点でのキャッシュの余裕を越えて無駄なラインをプリフェッチしてしまうことで生じる．このように考えると，プリフェッチスロットリングにおいては，キャッシュの余裕度こそが本質である．プログラムによってこの余裕度は様々である．SPECCPU2006 の一部のプログラムでは 256M 命令の実行中に一度も最下層キャッシュからのライン追い出しが起こらない一方で，大量に最下層キャッシュミスが生じるプログラムも存在する．従来のプリフェッチスロットリングでは，プリフェッチの成功度にのみ注目し，プログラムによって変化するキャッシュの余裕度は考慮されていない．

3.2 キャッシュ余裕度モデル

ここでキャッシュの余裕度について考える．余裕があるときのキャッシュ状態をやや近似して示すと，**図 1** のように MRU 側から LRU 側にかけて良く参照される濃い部分と載せられたものの参照されない薄い部分が分布していると考えられる．「キャッシュヒット率が 90%以上」と「濃い部分がキャッシュの 90%以上を占める」は異なる．キャッシュ全体における濃い部分の割合を α とすると，キャッシュヒットはこの α 領域へのアクセスによって

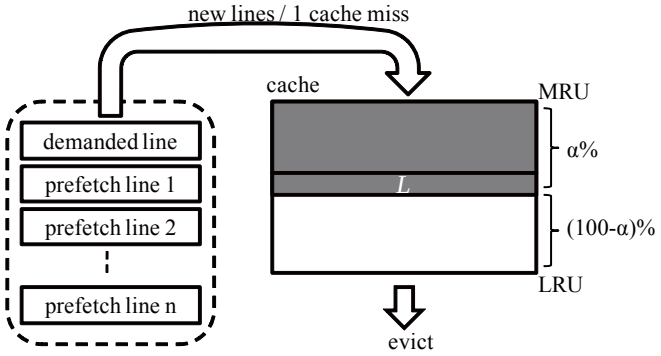

図 1 キャッシュ余裕度モデル
Fig. 1 Model of the Room of Cache

生じている．

この状態から，ポリューションの生じる条件を考える．今，濃い領域の中で最も LRU に近いライン L が次にアクセスされるまでのサイクル数を β とする．β サイクルの間に，新たに載せられるラインによって L が追い出されればポリューション発生であり，追い出されなければ L は次のアクセスによって MRU 側へ復帰する．キャッシュ全体のラインを S としたとき，L から LRU 側へは $(1-\alpha)\times S$ 個のラインがあり，これが β サイクルの間に新しくキャッシュに載せることができるラインの容量である．

この容量に対し，β サイクルの間にキャッシュに載せられるライン数は，その間に生じるキャッシュミスの数と，一回のミスアドレスから想起されるプリフェッチ数 $pwidth$ との積で近似することができ，式 1 の第一項のように表せる．

$$\frac{pwidth\times\beta}{miss\_interval} : (1-\alpha)\times S \tag{1}$$

式 1 は β サイクルの間に新たに載せられるライン数とキャッシュの余裕ライン数との比になっており，第二項を第一項で割った値が大きいほどキャッシュに余裕がある状態だと言える．更に，β はキャッシュの α 領域を一通りアクセスが巡る時間と近似すれば $\beta=\alpha\times S\times access\_interval$ となる．これを式 1 に代入すると，キャッシュサイズ S に拠らない式 2 を得る．

$$\frac{pwidth\times\alpha\times access\_interval}{miss\_interval} : (1-\alpha) \tag{2}$$

表 1 追い出しラインの分類
Table 1 Classification of Replaced Lines

	not accessed	accessed
demand	d0	d1
prefetch	p0	p1

3.3 置き換えライン情報からの余裕度推定

実行中にキャッシュ余裕度を知るためには，キャッシュ全体の情報 α が必要となるが，キャッシュから追い出されるラインの状態に注目することで，キャッシュを走査することなく推測することができる．注目する状態は，i) そのラインがプリフェッチによって載せられたか，ii) そのラインは一度でもアクセスされたか，の二点である．これらはプリフェッチビットとアクセスビットの 2bit をタグアレイに付加することで得られる．この 2bit は，多くのハードウェアプリフェッチで用いられているため[17]，ハードウェアを殆ど追加することなく得ることができる．

追い出された時のこの 2 ビットの組み合わせを，$p0$,$p1$,$d0$,$d1$ と分類し (**表 1**)，この数の統計を用いて余裕度を推測する．式 2 の第二項を第一項で割った式に代入して変形し，指標を得る (式 3)．この指標はいわばキャッシュの代謝に注目した値であるので，以降 CM(Cache Metabolism) 値と呼ぶ．

$$CM=\frac{d0+p0}{d1+p1}\times\frac{p1}{p0+p1}\times\frac{access\_count}{miss\_count} \tag{3}$$

図 2 はプリフェッチの積極度を変化させた時の IPC と CM 値の相関をプロットしたものである．SPECCPU2006 から，プリフェッチ積極度の変化が IPC に大きく影響するプログラムについて示している．プロセッサやプリフェッチのパラメタは後述する評価のものを用いた．計測は 1G 命令スキップ後の 100M 命令について行っている．横軸がプリフェッチ積極度となっており，数値が大きいほど大量の想起を行う設定となっている (5 章で詳しく述べる)．プログラムによって最適な積極度は全く異なることが分かる．CM 値もプログラムによって多様だが，CM 値の高い積極度と IPC の高い積極度との間に強い相関があり，スロットリング指標として有用であることが分かる．

4. スロットリング機構

4.1 手法の概要

CM 値を指標として，プログラムのフェーズ変化に合わせて，適切なプリフェッチ積極度

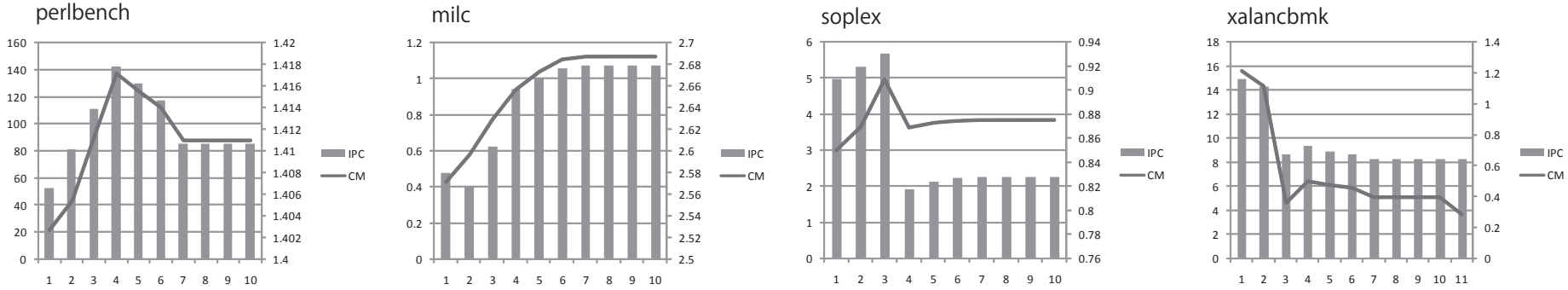

図 2 CM 値と IPC の相関
Fig. 2 Relationship between CM and IPC

を選ぶスロットリング手法を提案する (**図 3**)．図 2 より，適切な積極度は CM 値を最大にする積極度であり，CM 値は積極度を徐々に上げていった時に最適点を頂点とする山のようなグラフを描くことが分かる．そこで，今回は簡単な制御手法として TCP/IP のような AIMD 制御を導入する．

提案スロットリング手法は，プリフェッチ手法や適用するキャッシュ階層とは直交しており，どのような組み合わせでも使うことができる．後述の評価では，レイテンシ隠蔽が最も困難な最下層キャッシュを対象とし，精度と想起量を両立したハイブリッドプリフェッチに対して適用している．複数の指標とテーブルの組み合わせて加減速を決定する関連手法に比較して，一つの指標を元に AIMD 制御を行う提案手法はハードウェア量，アルゴリズム共に軽量である．また，閾値設定の必要がなく，容易に導入することができる．

4.2 制御アルゴリズム

提案手法では $p0$，$p1$，$d0$，$d1$，cache_access，cache_miss の 6 つのカウンタを追加し，それぞれ以下のように更新する．

p0 p0 状態のラインが追い出された時に+1 する
p1 p1 状態のラインが追い出された時に+1 する
d0 d0 状態のラインが追い出された時に+1 する
d1 d1 状態のラインが追い出された時に+1 する
cache_access キャッシュアクセスがあった時に+1 する
cache_miss キャッシュミスがあった時に+1 する

ラインリプレース回数 ($p0$，$p1$，$d0$，$d1$ の合計) が規定回数に達したら，その時点のカウンタ値を用いて式 3 の CM 値を計算し，各カウンタをリセットする．このように，提案手法では積極度を調整する単位期間として,「一定回数のキャッシュリプレースが起きるまでの期間」を用いる．この手法は，一般的な命令数あるいはサイクル数ベースの期間設定よりも，キャッシュフェーズの変化に即した期間設定と考えられる．関連研究でも同様の期間設定が用いられている[15),18)]．

バースト的な挙動に対してロバストにするため，計算された CM 値と前の期間の CM 値との平均を取り，これを新しい CM 値とする．新しい CM 値が前回の CM 値を上回っていたらプリフェッチ積極度を 1 増加させる．逆に，下回っていたら積極度を半分にし，保持 CM 値を 0 へリセットする．

提案手法は，履歴を保持するテーブル容量などを必要とせず，計測用の 6 つのカウンタと前回 CM 値と現在の積極度を保持するレジスタのみで実現でき，また積極度制御もシンプルである．

5. 評価環境

5.1 ベースラインプロセッサ

提案手法について，プロセッサシミュレータ鬼斬弐[19)] を用いた評価を行った．プロセッサ命令セットは Alpha AXP アーキテクチャを用い，マイクロアーキテクチャパラメタは **表 2** に示した値を用いた．シングルスレッドシングルコア実行，メインメモリレイテンシ

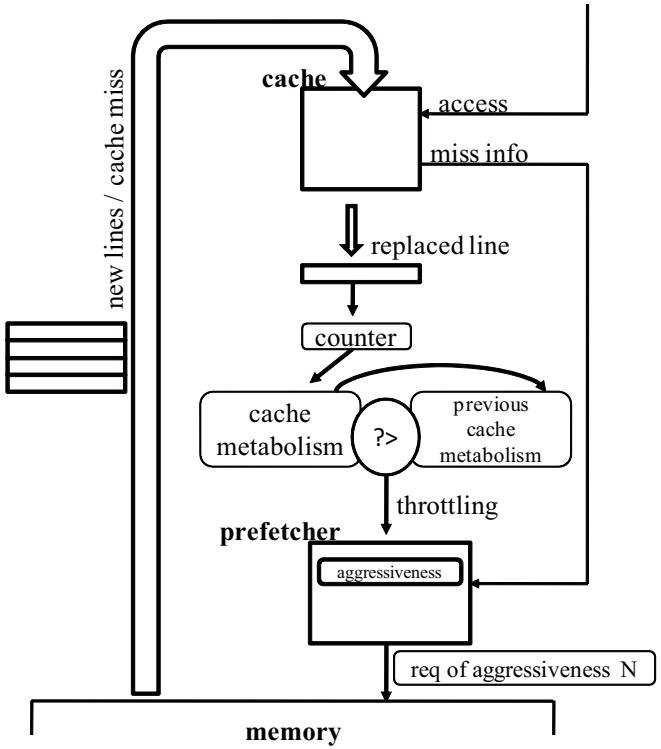

図 3 提案手法の概要
Fig. 3 Outline of the Proposed Technique

表 2 ベースラインプロセッサ緒元
Table 2 Microarchitectural Parameters for Baseline Processor

命令セット	Alpha ISA
フロントエンド	4way, 7cycle
命令ウィンドウ	i64 entries, f32 entries
LSQ	32entries
機能ユニット	2 iALU, 1 iMUL/DIV, 2 LD/ST, 1 fpADD, 1 fpMUL/DIV/SQRT
L1 ICache	32KB, LRU, 8way, 64B line, 1cycle latency
L1 DCache	32KB, LRU, 8way, 64B line, 1cycle latency
L2 I/DCache	1MB, LRU, 16way, 64B line, 20cycle latency
memory access	200cycle latency

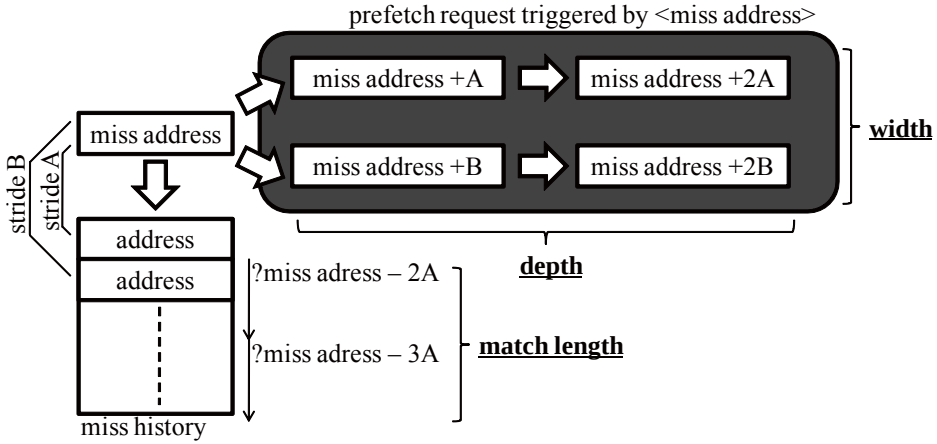

図 4 ジェネリック・プリフェッチャー (ストライド予測)
Fig. 4 Generic Prefetcher(stride prediction)

は一律 200 サイクルとした．プリフェッチは最下層キャッシュ(L2) に対してのみ適用する．メモリバンド幅について，今回は制限をしていない．文献 20)，21) や評価の章で示すように，シングルスレッド実行ではバンド幅に余裕があるため，評価値に大きな影響はない．ただし，バースト的な転送要求が発生したときにやや近似的な挙動となっている．

5.2 ジェネリック・プリフェッチャー

スロットリングを適用するプリフェッチャーとして，与えられたパラメタによって想起手法を変化させることのできるジェネリック・プリフェッチャーをシミュレータ上に実装した．これは，既存のプリフェッチャーのアドレス想起部分を一般化したもので，履歴検索スコープ，照合長，想起スキーム，幅，深さ，を動的に変化させることができる (**図 4**)．今回は深さ 8 の czone[9] ストライド予測[7] をベースとし，照合長と幅を変化させ，更に，パラメタ設定の異なる予測をハイブリッドに組み合わせることでプリフェッチ積極度を変化させた．積極度の設定を**表 3** に示す．

実装したプリフェッチアルゴリズムについて詳しく説明する．実装アルゴリズムでは，キャッシュミス時に履歴の更新とプリフェッチのリクエスト生成を行う．まず，キャッシュミス発生時にミスアドレスが履歴に記憶される．キャッシュミスをトリガとする手法では，プリフェッチ自身によって履歴が乱されてしまうため[6]，プリフェッチによって載せられたライ

表 3 プリフェッチ積極度設定
Table 3 Settings of Prefetcher Aggressiveness

level0	none
level1	czone 4-match 1-width stride 8-depth
level2	czone 3-match 1-width stride 8-depth
level3	czone 2-match 1-width stride 8-depth
level4	czone 2-match 1-width stride 8-depth + czone 1-match 1-width stride 8-depth
level5	czone 2-match 1-width stride 8-depth + czone 1-match 1-width stride 8-depth + 0-match 1-width stream 8-depth
levelN (N 〉 5)	czone 2-match (N-4)-width stride 8-depth + czone 1-match (N-4)-width stride 8-depth + 0-match 1-width stream 8-depth

ンについては，ヒットした場合でも，履歴登録とプリフェッチのトリガを行う．このようにして，ミスアドレス履歴には，プリフェッチが行われていなかった場合と同じミス履歴が保持される．

履歴更新と並行して，過去のミス履歴の検索を行う．czone スコープでは，この時，アドレスの上位ビットが同じもののみを対象とする．これは，近い領域内のミスアドレスのパターンを抽出するためである．履歴の検索方針として，他に，PC の同じものを抽出する local，全てを対象とする global などがあり，それぞれ得手不得手のアクセスパターンを持つ．

検索対象となったアドレス群から，もっとも最近のアドレスと今回のミスアドレスとの差を取り，ストライド候補とする．このストライド候補について，さらに履歴をさかのぼり，照合長として与えられたパラメタの数だけ合致すれば，ストライド検出とする．合致しなかった場合は，更に前のミスアドレスと今回のミスアドレスとの差を取り，このストライド候補についてチェックを行う．この動作を履歴の最後に達するまで行い，最大でプリフェッチ幅の数だけストライドを検出する．

最後に，各検出ストライドについて，深さの数だけ予測アドレスを生成する．例えば幅が 2，深さが 2 であれば，想起されるアドレスは，$missaddress + strideA$，$missaddress + 2 \times strideA$，$missaddress + strideB$，$missaddress + 2 \times strideB$，の 4 つとなる．メモリアクセスレイテンシが長いため，「次」のアドレスを予測したのでは間に合わない可能性があり，深さは，十分前もってプリフェッチリクエストを出すための手段である．近似的には，深さが $memory\_latency/cache\_access\_interval$ だけあれば，実行の進行とプリフェッチが釣り合うことになる．

ジェネリック・プリフェッチャにおけるパラメタの読み方の補足となるが，照合長について，例えば照合長 2 の場合は，今回のミスアドレスと同じ czone 内の前回のミスアドレスとの差と，前回のミスアドレスとそれより前のいずれかのミスアドレスとの差が合致すれば検出とする．つまり，プリフェッチがトリガされるまでに照合長+1 回のミスが必要となる．照合長 1 の場合は前回との差によって得られたストライドを無条件で適用する．照合長 0 の場合は，初回のミスで，予め設定されているデフォルトストライドを適用する．このデフォルトストライドをキャッシュラインサイズとすれば，これは次ラインをプリフェッチするストリームプリフェッチ[4),9)] に他ならない．

本論文で用いた積極度設定 (表 3) は，まず，積極度が低い時には照合長の長い幅 1 のストライド予測を適用する．照合長が長い場合，プリフェッチ精度が高くなるが，プリフェッチの始まりは遅く，また検出数も減少する．このため，積極度増加とともに，照合長を短くして，プリフェッチの出を早くする．さらに積極度を上げると照合長 2 と照合長 1 のストライド，およびストリームプリフェッチのハイブリッドとなる．このハイブリッドによって，長いスパンで有効なストライド，瞬間的に変化するストライド，近傍の空間的局所性などをカバーする．さらに積極度が上がると，各ストライド予測について幅を線形に増やし，複数のストライド候補をカバーするようにしている．一般に積極度調整では深さのみが対象となっているが，本論文では，プリフェッチの出の早さや想起の幅を重視した積極度設定とした．

本論文ではプリフェッチャ自体の実装や最適化については議論しない．様々な種類のプリフェッチャを混合した積極度を設定しているように，本手法は原理的にどのようなプリフェッチャにも有効である．なお，このようなジェネリック・プリフェッチャは GHB[12)] をベースとすることで実装できる．また，フットプリント[13)] を照合するパターンマッチによって，幅の広いストライド想起を高速に行う手法が提案されている[16)]．

5.3 実行ベンチマーク

SPECCPU2006 の全ベンチマークについて計測を行った．先頭 1G 命令をスキップし，その後の 256M 命令について cycle accurate に実行し，計測を行った．各ベンチマークプログラムは gcc バージョン 4.2.2 を用いて O3 オプションでコンパイルした．ベースラインプロセッサの L2 キャッシュサイズは 1MB としたが，この容量がある場合，434.zeusmp，444.named，453.povray，459.GemsFDTD，465.tonto などのベンチマークでは殆どキャッシュミスが起こらず，L2 プリフェッチは性能に影響しない．評価ではこのようなベンチマークも含めて統計を取っているため，キャッシュインテンシブなセットを抜き出した場合には数字の変化は更に大きいものとなる．

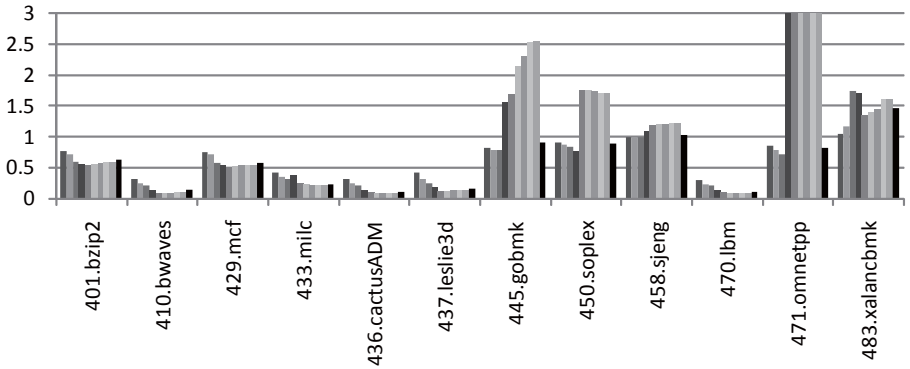

図 5 積極度および提案手法の効果 (cache miss ratio)
Fig. 5 Effect of the Proposed Mechanism and Prefetch Aggressiveness(cache miss ratio)

6. 評　　価

6.1 提案手法の評価

プリフェッチの積極度を変更したときのミス率の変化を**図 5** に示す．SPECCPU2006 ベンチマークから，L2 ミスが性能に影響するものについて示した．ベンチマーク毎に，異なる積極度を表す 10 本のバーが並んでいる．それぞれプリフェッチを行わなかった時に対するミス率の比で表しており，左端が積極度 1，そこから右のバーへ移るほど積極度が上がっていく．これらは積極度を固定している．一方，右端の黒いバーは提案手法を適用したときのミス率を表している．積極度が上がるとミス率が下がるもの，逆に上がるもの，特定の積極度に最適点があるものなど，SPECCPU2006 の中でもプログラムによって適切な積極度は様々であることがわかる．提案手法によるスロットリングを適用すると，xalancbmk を例外とすれば，これらのセッティングの中で常に最適に近いミス率削減を達成していることが分かる．xalancbmk では，ミス率の最適点の山が二つあり，今回用いた簡単な AIMD 制御では局所最適解に陥ってしまっている．

図 6 に IPC への影響を示す．ここでは全ベンチマークを絶対 IPC で示した．各ベンチマークについて左側から積極度 0 から 9 の固定プリフェッチが並び，右端の黒いバーが提案手法となっている．milc，cactusADM，leslie3d，lbm など積極的なプリフェッチが有効なプログラムでは積極度の高い時と似た性能を示し，一方で gobmk，soplex，omnetpp，

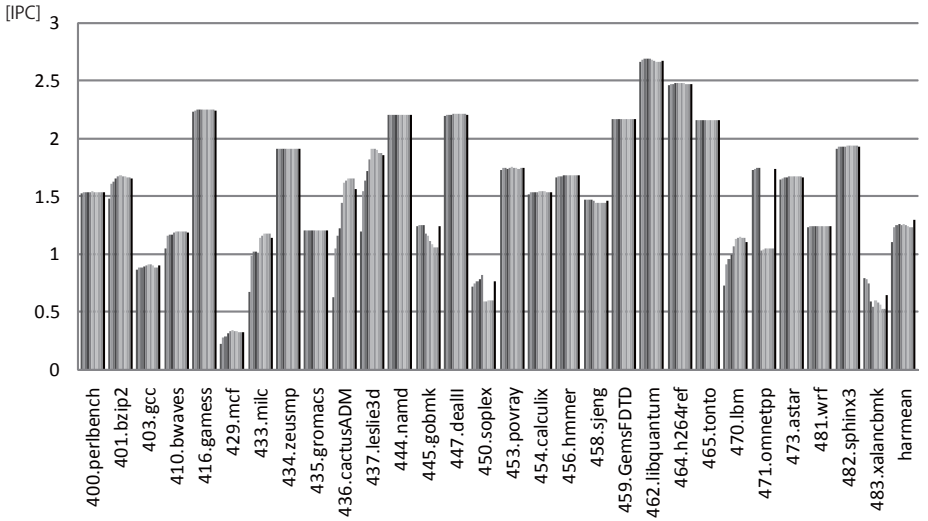

図 6 積極度および提案手法の効果 (IPC)
Fig. 6 Effect of the Proposed Mechanism and Prefetch Aggressiveness(IPC)

xalancbmk などポリューションが発生するプログラムではプリフェッチを押さえた時と同様の性能となっている．特に，soplex 等ではポリューションをおさえつつ，プリフェッチによる性能向上も得られる積極度を選択している．このような積極度調整により，SPECCPU2006 総合の調和平均では，提案手法はベストの積極度に対してさらに 2.3%の性能向上を得ている．IPC 改善の数値自体はプリフェッチャとして選ぶスキームに依存する．比較的精度の高いストライドプリフェッチャに適用して有意な差がでており，加減速とも適切にできていると言える．

次に，**図 7** は同様に，使用バンド幅を示したものである．プロセッサの動作周波数を 2GHz と仮定した値となっている．この値の限界は例えば Intel Core i7 であれば約 20GB/s 以上である．プリフェッチの積極度を上げても，精度が高ければバンド幅の使用はそれほど増加しない．例としては lbm が挙げられる．lbm は積極的にプリフェッチを行うことにより性能は大幅に上昇するが (図 6)，バンド幅はそれほど増加していない．

一方，プリフェッチ積極度の上昇とともに顕著にバンド幅の使用が上昇するベンチマークがいくつかあることが分かる．ポリューションの生じるプログラムは，このようなグルー

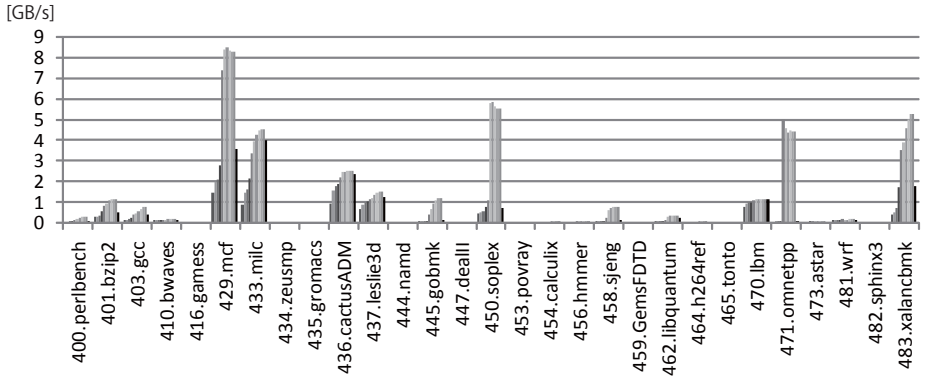

図 7 積極度および提案手法の効果 (バンド幅)
Fig. 7 Effect of the Proposed Mechanism and Prefetch Aggressiveness(bandwidth)

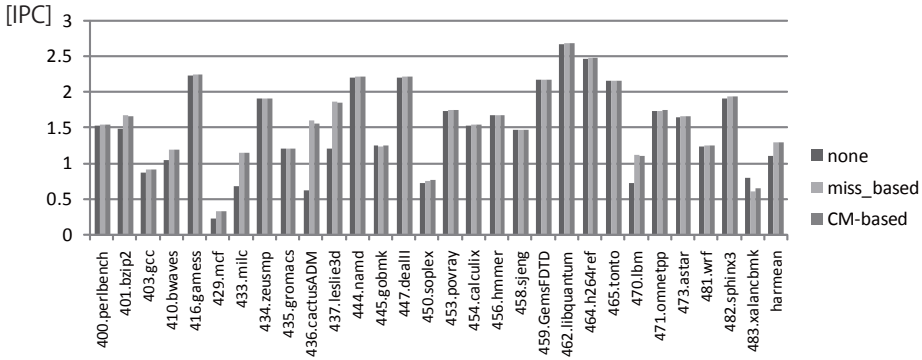

図 8 キャッシュミス率を指標としたスロットリングの効果 (IPC)
Fig. 8 Effect of the Cache Miss-based Throttling(IPC)

プに属しており，ポリューションは，プリフェッチのアルゴリズムが無駄なアドレスを「想起できてしまう」ために発生していることが分かる．しかし一方で，milc，cactusADM，leslie3d など，バンド幅の無駄と引き替えに，良い性能向上を得ているプログラムも含まれている．また，mcf のように，ある点までは無駄が許容できるプログラムも存在する (図 6)．プリフェッチの精度のみに着目するアプローチでは，この判別は難しい．提案手法は，前者ではバンド幅を絞り，後者ではバンド幅を利用して性能を向上させている．

6.2 キャッシュミス率を指標とするスロットリング

CM 値に含まれる $miss\_count/access\_count$ はキャッシュヒット率に他ならない．プリフェッチはキャッシュミスを減らすことが目的であり，ミス率をそのまま指標としてスロットリングすることは自然な方法だと考えられるが，我々の知る限り，ミス率を元にプリフェッチのスロットリングを行った例はまだない．理由の一つとして，ミス率はプログラム毎に多様であるため，従来のやり方では閾値が設定できないことが挙げられる．しかし，本論文で提案したスロットリングを用いれば，ミス率をそのままスロットリング指標として用いることができる．

ミス率を指標として同様の IPC 評価を行ったグラフを**図 8** に示す．CM 値を用いた場合と同様に，各プログラムに合わせて最適セッティングと近い性能を達成している．ただし，CM 値に比較すると全体の調和平均は 0.6%下がっている．これは，置換ライン情報を用いることによる先読み効果と考えられる．しかし，ミス率を元にスロットリングする手法は CM 値よりもさらに少ないハードウェアで可能であり，良い精度を得ているといえる．

7. おわりに

本論文では，プリフェッチの積極度を最適化する新しいスロットリング手法を提案した．従来のスロットリング手法では，プリフェッチの成功度のみが注目されていたが，提案手法ではキャッシュの余裕度に着目し，最適化の精度向上とハードウェア軽量化の両立を目指した．指標として，キャッシュの有効ライン率と載せ替え頻度に着目した Cache Metabolism 値を提案し，IPC と強い相関があることを示した．更に，CM 値またはキャッシュミス率を指標としてプリフェッチ積極度を調整する機構を提案した．SPECCPU2006 とプロセッサシミュレータを用いた評価により，提案手法によるスロットリングは，プログラム毎に異なる最適な積極度に追随できることが分かった．ストライドベースのハイブリッドプリフェッチャーに適用した場合，最も良い積極度に設定されたプリフェッチャーと比較して，IPC 調和平均でさらに 2.3%の性能向上を確認した．また，さらに軽量なスロットリング手法として，ミス率のみを元に制御する手法を評価し，CM 値と同様に高い精度を持つことを確認した．

今回の評価ではポリューションの少ないストライドプリフェッチをベースとした．僅かな差でも最適な積極度を選ぶセンシティビティの評価となっている一方で，性能上のインパクトは少なくなっている．CDP[11] や Spatial Locality[10] などの積極的なプリフェッチアルゴ

リズムに適用すれば性能への影響はさらに高くなると考えられる．

今後の課題として，バンド幅の正確な評価が挙げられる．ただし，図 7 に示すように，性能を指標とした制御によっても，バンド幅浪費はある程度改善されている．更なるバンド幅についての議論は，I/O パッドが消費する電力を考慮するような，電力改善の議論になると考えられる．また，提案手法のマルチコアへの適用も今後の課題である．マルチコアでは他コアのデマンドアクセスや他コアのプリフェッチなど，考えるべき要素が複雑となり，様々な手法が提案されているが，CM 値を用いれば軽量な最適化が可能であると考えられる．

参 考 文 献

1) Akkary, H., Rajwar, R. and Srinivasan, S.: Checkpoint processing and recovery: towards scalable large instruction window processors, *Int. Symp. on Microarchitecture*, pp.423–434 (2003).
2) Tullsen, D.M., Eggers, S.J., Emer, J.S. and Levy, H.M.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *22nd Int. Symp. on Computer Architecture*, pp.392–403 (1995).
3) Smith, A.: Cache Memories, *ACM Computer Surveys*, Vol.14, No.3, pp.473–530 (1982).
4) Jouppi, N.: Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, *Int. Symp. on Computer Architecture*, pp.364–373 (1990).
5) Purser, Z., Sundaramoorhy, K. and Rotenberg, E.: A Study of Slipstream Processors, *33rd Int. Symp. on Microarchitecture*, pp.269–280 (2000).
6) Smith, A.: Sequential Program Prefetching in Memory Hierarchies, *IEEE Computer*, Vol.11, No.12, pp.7–21 (1978).
7) Fu, J. and Patel, J.: Stride directed prefetching in scalar processors, *Int. Symp. on Microarchitecture*, pp.102–110 (1992).
8) Tendler, J., Dodson, J., J.S.Fields, J., H.Lee and Sinharoy, B.: Power4 system microarhitecture, *IBM Journal of Research and Development*, Vol.46, No.1, pp.5–26 (2002).
9) Palacharla, S. and Kessler, R.: Evaluating stream buffers as a secondary cache replacement, *Int. Symp. on Computer Architecture*, pp.24–33 (1994).
10) Johnson, T., Merten, M. and Hwu, W.: Runtime spatial locality detection and optimization, *Int. Symp. on Microarchitecture*, pp.57–64 (1997).
11) Cooksey, R., Jourdan, S. and Grunwald, D.: A stateless, content-directed data prefetching mechanism, *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.279–290 (2002).
12) Nesbit, K. and Smith, J.: Data Cache Prefetching Using a Global History Buffer, *Int. Symp. on High Performance Computer Architecture*, pp.96–105 (2004).
13) Somogyi, S., Wenisch, T., Ailamaki, A., Falsafi, B. and Moshovos, A.: Spatial Memory Streaming, *Int. Symp. on Computer Architecture*, pp.252–263 (2006).
14) Hur, I. and Lin, C.: Memory Prefetching Using Adaptive Stream Detection, *Int. Symp. on Microarchitecture*, pp.397–408 (2006).
15) Srinath, S., Mutlu, O., Kim, H. and Patt, Y.: Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers, *Int. Symp. on Highi-Performance Computer Architecture*, pp.63–74 (2007).
16) Ishii, Y., Inaba, M. and Hiraki, K.: Access Map Pattern Matching Prefetch: Optimization Friendly Method, *Workshop on JILP Data Prefetching Championship* (2009).
17) Zhuang, X. and Lee, H.: A Hardware-based Cache Pollustion Filtering Mechanism for Aggressive Prefetches, *Int. Conf. on Parallel Processing*, pp.286–293 (2003).
18) Ebrahimi, E., Mutlu, O. and Patt, Y.: Techniques for Bandwidth-Efficient Prefetching of Linked Data Structures in Hybrid Prefetching Systems, *Int. Symp. on Highi-Performance Computer Architecture*, pp.7–17 (2009).
19) 江口修平，塩谷亮太，五島正裕，坂井修一：主記憶バンド幅がプロセッサ性能に与える影響の評価，先進的計算基盤システムシンポジウム 2009 ポスター.
20) 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 2009 ポスター (2009).
21) Murphy, R. and Kogge, P.: On The Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications, *IEEE Trans. on Computers*, Vol.56, No.7, pp.937–945 (2007).