Vol.2010-ARC-187 No.14
Vol.2010-EMB-15 No.14
2010/1/29

省ハードウェア資源のフィードバックつきハイブリッドプリフェッチ方式

本城剛毅[†1] 石井康雄[†1,†2] 入江英嗣[†1]
稲葉真理[†1] 平木敬[†1]

メモリアクセス性能は，プロセッサ全体の性能を支配している．データ・プリフェッチはメモリアクセス性能を改善するための手法で，将来プロセッサに読み込まれるであろうアドレスを予測し，あらかじめデータをフェッチしておくものである．ハイブリッド・プリフェッチは，複数のプリフェッチ手法を同時に使うことでそれぞれの手法が持つ特長を活かす手法である．しかし，単に同時に複数の手法が予測したアドレスをすべて使ってプリフェッチをするだけでは，メモリから多くのデータをフェッチしようとしすぎて，メモリ帯域の飽和によって性能が下がってしまう．我々は先行研究として SHPF を提案し，プリフェッチ量を制御したが，ハードウェア資源を使いすぎて現実的ではなかった．

本論文では，フィードバックによってプリフェッチ量を制限するハイブリッド・プリフェッチとして HPFSH (Hybrid Prefetching with Feedback using Sampled History) を提案する．HPFSH は各プリフェッチ手法が予測した情報を記録するのに，キャッシュタグとバッファに分散することで，ハードウェア資源を節約する手法である．HPFSH は，CAM 構造をとるハードウェア量を SHPF に比べ 1/27.5 程度まで削減したにも関わらず，制御なしのハイブリッド・プリフェッチと比べた IPC 性能面では SHPF に対し 46.8%を達成した．

Hybrid Prefetching with Feedback using Sampled History

Goki Honjo,[†1] Yasuo Ishii,[†1,†2] Hidetsugu Irie,[†1]
Mary Inaba[†1] and Kei Hiraki[†1]

Memory access performance dominates processor performance. Data prefetching is a method to improve memory access performance, and it speculates memory addresses that will be loaded by processor in the future and fetches data from memory in advance. Hybrid prefetching uses multiple prefetching methods so that it makes use of their features. However, it tries to fetch too much data and decreases performance because of saturation of memory bandwidth when it uses all memory addresses speculated by prefetching methods. We previously proposed SHPF that controls the amount of prefetch accesses, but it is not realistic because it consumes too much hardware resources.

This paper proposes HPFSH (Hybrid Prefetching with Feedback using Sampled History) which limits the amount of prefetch accesses. HPFSH reduces consumed hardware resources by recording information of address speculation into cache tags and buffer separately. Although HPFSH reduced hardware resources of CAM structure to 1/27.5 compared to SHPF, in terms of IPC performance compared to hybrid prefetching without controls HPFSH achieved 46.8%.

1. はじめに

汎用プロセッサの性能向上に伴い，汎用プロセッサがより多くのデータを処理するアプリケーションを処理する機会が増えてきた．しかし，汎用プロセッサ本体の演算性能向上に比べ，メモリアクセス性能の向上が追いついていないため，メモリアクセス性能がプロセッサ全体の性能を支配している．これを解決するため，データ・プリフェッチやアウト・オブ・オーダー実行などのメモリアクセスレイテンシ隠蔽手法が数々提案されてきた．

データ・プリフェッチは，プロセッサが実際にロード命令を発行するより前に，あらかじめアクセスされるであろうメモリアドレスを予測して，データをメモリからプロセッサに移動しておく手法である．本論文では，プリフェッチが成功したときの利得が大きく，動的な情報を扱うことができるラストレベルキャッシュへのハードウェア・プリフェッチに注目する．

現在まで，ハードウェア・プリフェッチとして数々の手法が提案されてきたが，実行するソフトウェアの性質に対し向き不向きがあるため，幅広いソフトウェアに対し常に高い性能を示すプリフェッチ方式はなかった．ハイブリッド・プリフェッチは，複数のプリフェッチ手法を同時に使い，それぞれの手法が予測したアドレスを取捨選択してプリフェッチすることで全体としてプリフェッチのカバレッジを上げ，性能を向上させようとする手法である．しかし，制御をしないハイブリッド・プリフェッチは，精度の低いプリフェッチ手法が予測したアドレスもプリフェッチに使ってしまうため，大量のメモリアクセスが発生し，メモリ帯

†1 東京大学
The University of Tokyo
†2 日本電気株式会社
NEC Corporation

Vol.2010-ARC-187 No.14
Vol.2010-EMB-15 No.14
2010/1/29

域を埋めてしまう．その分，プロセッサ本体が発行するデマンドアクセスが遅れてしまい，むしろ性能が低下してしまう場合がある．

メモリ帯域が埋まらないようにするには，プリフェッチ手法の予測が不正確かどうか判断し，その手法が予測するアドレスはプリフェッチには使用しないようにする必要がある．しかし，プログラムによって，またプログラムのフェーズによって予測精度の高いプリフェッチ手法は変わるので，予測が不正確かどうかの判断は動的に行われなければならない．

先行研究として我々は，プリフェッチ手法ごとに予測精度を動的に調べ，予測精度の低いプリフェッチ手法を使わないようにすることでメモリ帯域を空け，性能を上げるハイブリッド・プリフェッチ手法として SHPF[13] を提案した．SHPF は多くのプログラムに対して性能を上げることができる優れた手法であったが，この手法ではテーブルに予測されたアドレスを全て記憶するという手法がとられており，消費するハードウェア量が大きかった．

本論文では，SHPF の利点を活かしながらハードウェア量を削減する「HPFSH (Hybrid Prefetching with Feedback using Sampled History)」を提案する．HPFSH においても予測されたアドレスをチップ上に記録するが，プリフェッチに使ったアドレスは情報をキャッシュタグに記録し，プリフェッチに使わなかったアドレスはその一部だけをテーブルに記録することで，記憶容量の低減を図る．

本論文は以下，次のように構成される．2 章では，ハイブリッド・プリフェッチと SHPF の性能評価を行い，3 章では，HPFSH を提案する．4 章で HPFSH を評価し，5 章では関連するプリフェッチ手法と，フィードバックについての関連研究について述べ，6 章でまとめを述べる．

2. ハイブリッド・プリフェッチ

本論文では，複数のハードウェア・プリフェッチ手法を同時に使用することで性能を向上させるハイブリッド・プリフェッチ[4] について議論を行う．本論文で取り上げる制御なしのハイブリッド・プリフェッチとは，複数のプリフェッチ手法によって同時に予測を行い，予測されたアドレスすべてをフィルタリングせずに使用してメモリアクセスを行う手法である．

本論文では，複雑なメモリアクセスパターンを予測することができ，精度の高い予測をする反面，予測を始めるまでの「学習」に時間のかかる C/DC Prefetch[7] と，学習には比較的時間がかからないものの精度も比較的低い予測をする Stride Prefetch[2]，学習は一切行わないため大量の予測を行うことができるが，予測精度が低い Block Prefetch[3),10)] を使用する．

まず，今回使用した 3 つの手法のプリフェッチ性能を評価した．評価環境は「4. 評価」に示したとおりである．評価の結果，3 つの手法のうち，すべてのベンチマークにおいて他の 2 つに勝るような手法はなく，どれも一長一短であることがわかった．(図 2 の “Block”，“Stride”，“C/DC”）つまり，この 3 種を組み合わせたハイブリッド・プリフェッチは，制御をしなくても，プリフェッチのカバレッジの面では 3 つの手法のうちのひとつだけを使った時よりも上回るはずである．

次に，3 つのプリフェッチ手法を使用した制御なしのハイブリッド・プリフェッチの性能を評価した．その結果，制御なしのハイブリッド・プリフェッチは，特に C/DC Prefetch のみが性能を上げ，Block Prefetch または Stride Prefetch がプリフェッチなしよりも性能を下げたベンチマークにおいて，性能が大きく下がった．(図 2 の “NoPref.”，“Block”，“Stride”，“C/DC”，“merge”）メモリ帯域の情報を見てみると，3 つの手法すべてを使用した時は，プリフェッチなしの時と比べてメモリアクセス数が最大で 617%増加していた．多量の不正確なプリフェッチがメモリ帯域を圧迫し，性能を低下させたと考えられる．実行時のフィードバックなどによって不要なプリフェッチを抑制すれば，その分メモリ帯域が空くことになり，性能低下を防げると考えられる．

そこで，上記の特性を実現してする方式の一つとして，我々が以前提案した SHPF の性能を評価した．SHPF は，制御なしのハイブリッド・プリフェッチが高い性能を示したベンチマークでは，ハイブリッド・プリフェッチと同程度の性能を示し，制御なしのハイブリッド・プリフェッチが性能を下げたベンチマークでは C/DC Prefetch とおよそ同程度の性能を示した．(図 2）SHPF は，無駄なプリフェッチが発生しているときは，そのプリフェッチを止め，性能の向上に寄与したと言える．

しかし，SHPF は 1024 エントリにわたるプリフェッチ・ヒストリー・テーブルを使うため巨大な CAM 構造を持つ必要がある．さらに，キャッシュアクセスがあるたびにそのテーブル全体を検索してアクセスのあったアドレスが予測されていなかったかを調べるため，ハードウェアに実装する上で現実的でなかった．

3. HPFSH (Hybrid Prefetching with Feedback using Sampled History)

我々は HPFSH (Hybrid Prefetching with Feedback using Sampled History) を提案する．HPFSH では，不要なプリフェッチが行われ，メモリ帯域が無用に埋まるのを防ぐために，精度・カバレッジの低いプリフェッチ手法を検出し，その手法が予測したアドレスをフィ

Vol.2010-ARC-187 No.14
Vol.2010-EMB-15 No.14
2010/1/29

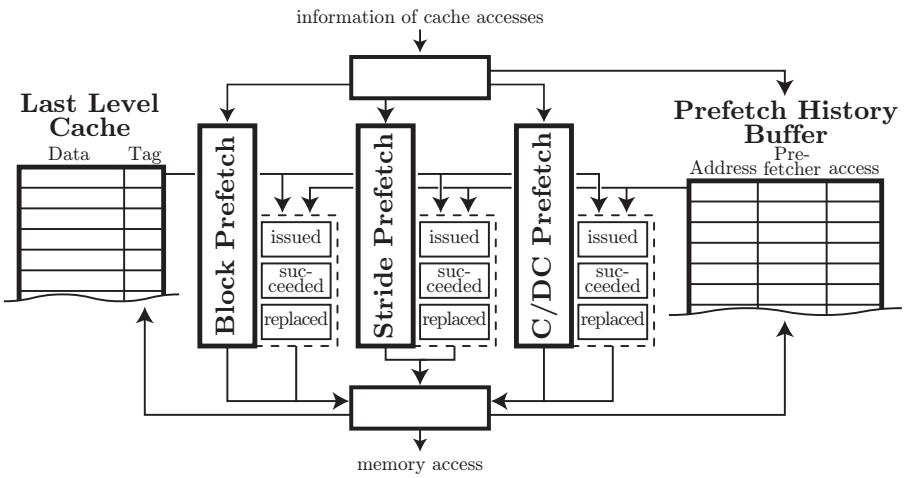

図 1 HPFSH の構造
Fig. 1 Mechanism of HPFSH

ルタリングしてプリフェッチを行う．プリフェッチの履歴を記録する際にキャッシュタグを利用したり，履歴の一部だけを記録したりすることで巨大な CAM 構造のハードウェアを削減し，現実的なハードウェア構成を実現した．

図 1 に示したのが，本手法の構造である．各プリフェッチ手法ごとに，“issued” “succeeded” “replaced” の 3 つのカウンタを持ち，精度とカバレッジの計算に使われる．また，プリフェッチ手法を使う／使わないの判断が変わる時にその手法のカウンタはすべてリセットされる．本手法の挙動は，大きく「プリフェッチ手法を使わないと判断するまで」と「プリフェッチ手法を再び使うと判断するまで」に分けることができる．

3.1 プリフェッチ手法を使わないと判断するまで

本論文では 3 種類のプリフェッチ手法を使用しているため，キャッシュタグは “C/DC” “Stride” “Block” “access” の 4 ビットで構成されている．

まず，プリフェッチが完了してキャッシュにデータを格納する際，そのアドレスをどのプリフェッチ手法が予測していたかを，キャッシュタグの対応するビットを立てることで記録する．プリフェッチ手法と同じ数のビットを用意することで，同時に複数の手法が同じアドレスを予測した際に正しく記録することができる．また，キャッシュにアクセスがあるたびに，プリフェッチ手法ごとのビットの値にかかわらず，そのラインの “access” ビットが立つ．

プリフェッチしてキャッシュに格納したラインがキャッシュから追い出されるとき，キャッシュタグの情報を見てそれぞれのプリフェッチ手法ごとに用意されたカウンタ “issued” “succeeded” “replaced” の値をインクリメントする．“issued” は “access” の値に関わらずプリフェッチ手法に対応するビットが立っていた場合にインクリメントし，“succeeded” は，手法に対応するビットが立ち，かつ “access” ビットも立っていた場合にインクリメントする．また，“replaced” は “access” ビットが立っているラインがキャッシュから追い出されるたびにインクリメントする．

そして，随時それぞれのプリフェッチ手法の精度とカバレッジが閾値を下回っていないかどうかを確認する．精度とカバレッジは以下のように計算される．

$$\text{accuracy} = \frac{\text{succeeded}}{\text{issued}},\ \text{coverage} = \frac{\text{succeeded}}{\text{replaced}}$$

両方がともに閾値を下回っていた場合はそのプリフェッチ手法は使わないと判断し，3 つのカウンタをクリアする．

3.2 プリフェッチ手法を再び使うと判断するまで

使わないと判断されたプリフェッチ手法が予測したアドレスのうち，乱数による選別を行い，一部をプリフェッチ・ヒストリー・バッファに記録する．このプリフェッチ・ヒストリー・バッファのエントリは，予測したアドレス，どのプリフェッチ手法が予測したか，アクセスがあったかどうか（“accessed”），プリフェッチ・ヒストリー・バッファに登録された時刻（サイクル）の下位数ビットから構成される．

キャッシュアクセスがあるたびに，アクセスのあったメモリアドレスとプリフェッチ・ヒストリー・バッファに記録されたアドレスが照合され，“accessed” に記録される．

このプリフェッチ・ヒストリー・バッファは，登録されてから一定時間が経つか，プリフェッチ・ヒストリー・バッファがいっぱいになると追い出される．追い出されるときに，プリフェッチ手法ごとに用意されたカウンタ “issued” の値がインクリメントされる．さらに，“accessed” ビットが立っていれば，“succeeded” の値もインクリメントされる．

先ほどと同じように，随時精度が計算され，閾値を超えていればそのプリフェッチ手法は再び使うと判断し，“issued” “succeeded” の両カウンタをクリアする．また，そのプリフェッチ手法が予測したアドレスとしてプリフェッチ・ヒストリー・バッファに記録されていたレコードはすべて無効化され，それらのレコードはキャッシュアクセスの行われたメモリアドレスとの照合も行わない．

Vol.2010-ARC-187 No.14
Vol.2010-EMB-15 No.14
2010/1/29

SHPF も HPFSH も，プリフェッチ・ヒストリー・バッファのデータ構造はほぼ同じで HPFSH には時刻フィールドが加わっているだけだが，HPFSH ではプリフェッチに使われなかったアドレスの情報のみを，乱数を使って数を絞って記録するため，SHPF に比べて小規模な CAM 構造のハードウェアで情報を処理することができる．

4. 評　　価

4.1 評価環境

本章では，Block Prefetch, Stride Prefetch, C/DC Prefetch を用いたハイブリッド・プリフェッチ方式に対して HPFSH を適用した際の性能とハードウェア資源とを評価した．比較対象に，SHPF と制御なしのハイブリッド・プリフェッチを評価した．

性能評価は Cycle Accurate なプロセッサシミュレータの鬼斬弐[14] を使用した．ベンチマークには SPEC CPU2006[⋆1] からメモリアクセス回数の多い 15 ベンチマークを採用した．実行時には，先頭の 4G 命令をスキップし，後続の 300M 命令を評価した．その他，詳細なパラメータは表 1 に示した通りである．

4.2 性能評価結果

図 2，図 3，図 4 に示したのは，プリフェッチなし，本論文で使用した 3 つのプリフェッチ手法単体の性能，制御なしのハイブリッド・プリフェッチ，SHPF，HPFSH の IPC，メモリアクセスバスのアクセス数，MPKI の性能評価結果である．

評価の結果，IPC は制御をしないハイブリッド・プリフェッチを上回り，SHPF に 46.8%近づくことができた．（図 2 の “merge”，“SHPF”，“HPFSH”）また，メモリアクセスのトラフィック量も制御なしのハイブリッド・プリフェッチから 73.3%削減することができた．（図 3）しかし，SHPF に比べて MPKI の改善幅が小さいことから，HPFSH が無用なプリフェッチだけでなく有用なプリフェッチも止められてしまったと推測される．（図 4）

また，bzip2，libquantum，lbm など，HPFSH がほかのどの手法よりも MPKI が高くなってしまうベンチマークがあった．HPFSH によってプリフェッチ手法が使われなくなった後，再び使われるようになる際のアルゴリズムに改善の余地があると思われる．

4.3 ハードウェア資源

ここでは，HPFSH の実装の実現性を SHPF と比較しながら論じる．

SHPF は全てのアクセス履歴をプリフェッチ・ヒストリー・バッファに保存するため，1024 エントリといった巨大なバッファが必要となる．さらに，このバッファはアクセスのたびに全エントリのタグを検索する必要があり，必然的に CAM 構造をとることになる．1024 エントリの CAM 構造は内部的に多くのアドレス比較回路を持つため回路サイズが大きく，かつ，消費電力も巨大となり，現代のプロセッサに適合しない．HPFSH では，バッファに記録する情報を，実際のプリフェッチには使われなかったもののみに絞ったうえ，乱数での選別をすることで CAM 構造のエントリ数を削減している．これによって，プリフェッチ・ヒストリー・バッファに記録するアドレスを削減し，このバッファの省資源化を実現する．

以下に，本評価に用いたパラメータでのハードウェア量を示す．各キャッシュタグに 4 ビットのフラグを付加するが，このフラグの総量は $4 \times 16\text{K} = 8\text{KB}$ である．これは，電力効率の良い RAM で実現されるため，その消費電力・面積は CAM 構造と比較して小さい．また，禁止したプリフェッチをプロファイルするためのバッファのサイズは 32 エントリで，各エントリが 71 ビット利用するため，2,272 ビットの CAM 構造が必要となる．CAM 構造の電力・面積はビット数に比例するため，HPFSH の 32 エントリのバッファは SHPF から 1/27.5 程度に削減できる．その他，小規模なカウンタ，パイプラインレジスタを用いるこ

表 1 シミュレーションのパラメータ
Table 1 Parameters of simulation

Processor	Alpha ISA, 4 way out of order, 2.0 GHz
L1 cache	64 B line, 8-way set associative, 32 KB, Latency: 1 cycle
L2 cache	64 B line, 16-way set associative, 1 MB, Latency: 20 cycles
Main Memory	Latency: 200 cycles, 5 GB/sec
Thresholds to disable prefetcher	accuracy: 50%, coverage: 25%
Threshold to enable prefetcher	accuracy: 25%
Prefetch buffer	maximum 32 entries, maximum 1024 cycles
Thresholds of SHPF	Block: 12, Stride: 11, C/DC: 5
Block Prefetch	depth: 4
Stride Prefetch	depth: 4, 256 histories
C/DC Prefetch	256 IT entries, 256 GHB entries, 4 KB CZone
compile	gcc 4.2.1 with option “-O3”

⋆1 http://www.spec.org/cpu2006/

Vol.2010-ARC-187 No.14
Vol.2010-EMB-15 No.14
2010/1/29

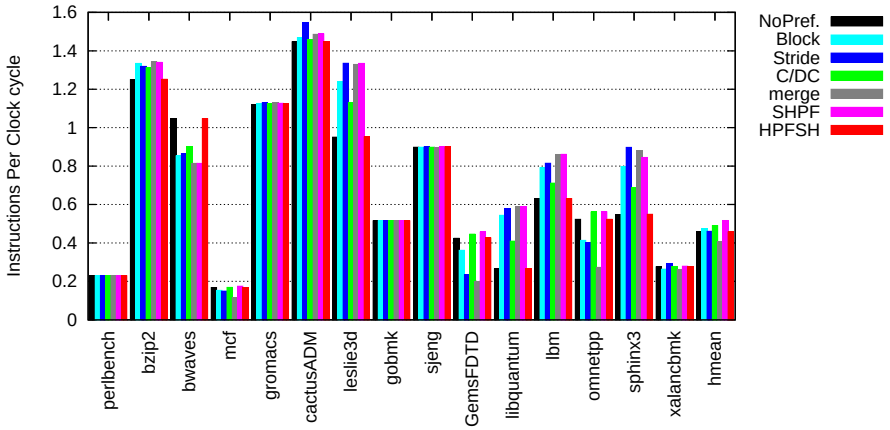

図 2 IPC（Instructions Per Clock cycle）性能評価結果
Fig. 2 Result of IPC (Instructions Per Clock cycle) performance evaluation

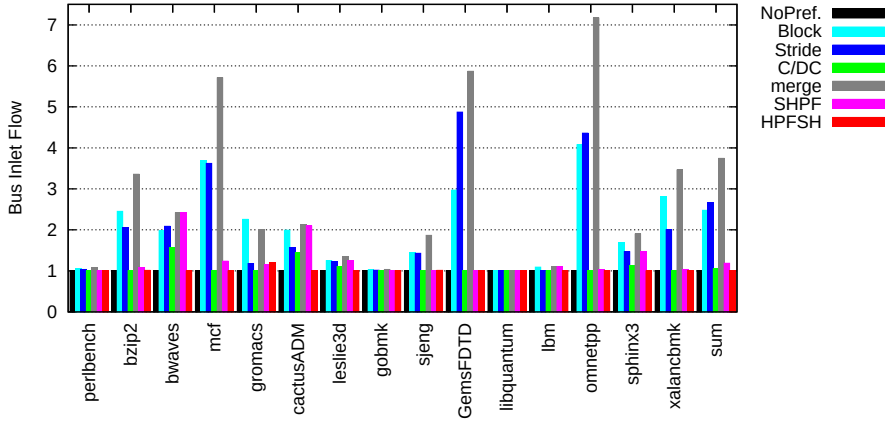

図 3 バス流入量性能評価結果（NoPref. の値で正規化）
Fig. 3 Result of bus inlet flow performance evaluation (normalized by NoPref.)

とで，HPFSH は現実的なハードウェア量と論理の複雑さを実現し，小規模で高電力効率のハードウェアを実現していることが確認できる．

5. 関連研究

Block Prefetch[3),10)] は，キャッシュミスが起きたラインの直後の数ラインに対してメモリアクセスを行う手法である．Stream Prefetch[6),9),12)] は Sequential Stream を検出し，検出できた時のみメモリアクセスを行うことで無意味なプリフェッチを減らそうとする手法である．

Stride Prefetch[2)] は，同じプログラムカウンタによるメモリアクセスがキャッシュミスを起こしたときに Stride を計算してメモリアクセスを行う手法である．

C/DC Prefetch[7)] はメモリを CZone と呼ばれる一定の大きさに区切り，それぞれの CZone の中で GHB Prefetch[8)] を行おうとする手法である．GHB Prefetch は，キャッシュミスを起こしたメモリアドレスの間隔に注目し，アクセスされたアドレスの間隔に注目し，アドレスの間隔に規則性を見つけようとする．複雑なメモリアクセスパターンを予測することができるものの，規則性が見つけられたときにしか予測できない．

Feedback Directed Prefetching[11)] は単位時間ごとにプリフェッチの精度・遅延・ポリューションを計測し，それによってプリフェッチの積極度と，キャッシュへの挿入ポリシーを変化させる手法である．

Adaptive Stream Detection[5)] は単位時間ごとに stream の長さを測り，stream の平均的な長さを計算し，その長さになるまでプリフェッチを出す手法で，プリフェッチを長く出し過ぎることを防ぐ手法である．

E. Ebrahimi らの手法[1)] は，2 種類のプリフェッチ手法を組み合わせたハイブリッド・プリフェッチ時に，単位時間ごとにそれぞれの精度とカバレッジを計測し，自分自身の精度・カバレッジ，そして相手のカバレッジによってプリフェッチのスロットリングを行う手法である．

6. おわりに

本論文では，SHPF の実装のしにくさを解消するためのハイブリッド・プリフェッチ方式 HPFSH （Hybrid Prefetch with Feedback using Sampled History）を提案した．SHPF ではプリフェッチによって予測されたアドレスをすべて，CAM 構造を取る大規模なハードウェアに記録していた．しかし，HPFSH は実際にプリフェッチを行った予測情報についてはスケーラビリティのあるキャッシュタグに記録し，プリフェッチを行わなかったアドレス

Vol.2010-ARC-187 No.14
Vol.2010-EMB-15 No.14
2010/1/29

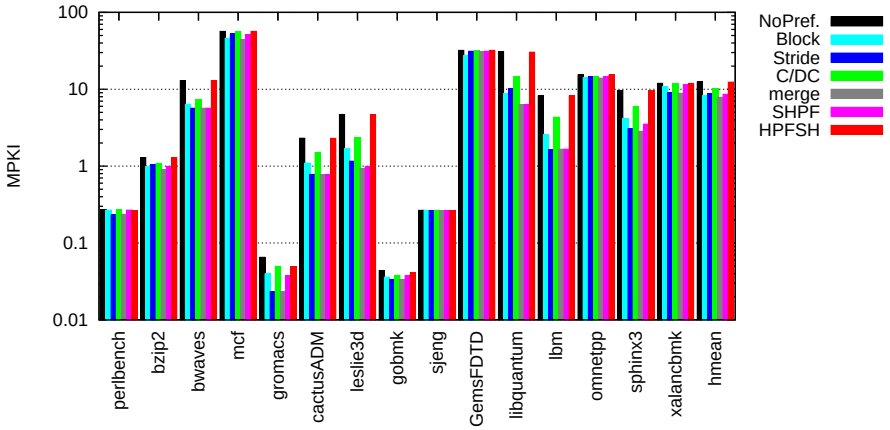

図 4 MPKI（Misses Per Kilo Instructinos）性能評価結果
Fig. 4 Result of MPKI (Misses Per Kilo Instructions) performance evaluation

についてはその一部をサンプリングして CAM 構造を取る小規模なハードウェアに記録するように情報を分散して記録したため，小規模化を実現することができた．

HPFSH の性能を評価した結果，HPFSH は制御なしのハイブリッド・プリフェッチに比べると，無駄なプリフェッチが減ったことで性能が向上したが，SHPF の性能には勝らず，制御なしのハイブリッド・プリフェッチと比べた IPC 性能面では SHPF に対し 46.8%の性能であった．ハードウェア資源を比較すると，CAM 構造をとるハードウェア量を SHPF に比べて 1/27.5 まで削減することができた．よって，HPFSH は，無用なプリフェッチを止める仕組みを，より少ないハードウェア資源量で実現したと言える．

しかし，一部のベンチマークでは HPFSH はほかのどの手法よりも MPKI 値が悪くなった．これがハードウェア資源量を小さくしたことによる誤差が原因なのかほかに原因があるのかはわからなかったが，アルゴリズムの変更によって改善する余地があるので，この点は今後の課題としたい．

参 考 文 献

1) Ebrahimi, E., Mutlu, O. and Patt, Y.: Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems, *HPCA-15: IEEE 13th Int. Symp. on High Performance Computer Architecture*, pp.7–17 (2009).
2) Fu, J. W.C., Patel, J.H. and Janssens, B.L.: Stride directed prefetching in scalar processors, *MICRO-25: Proc. of the 25th Annu. Int. Symp. on Microarchitecture*, Los Alamitos, CA, USA, IEEE Computer Society Press, pp.102–110 (1992).
3) Gindele, J.: Buffer Block Prefetching Method, *IBM Technical Disclosure Bulletin*, Vol.20, No.2, pp.696–697 (1977).
4) Hsu, W.-C. and Smith, J.: A performance study of instruction cache prefetching methods, *IEEE Transactions on Computers*, Vol.47, No.5, pp.497–508 (1998).
5) Hur, I. and Lin, C.: Memory Prefetching Using Adaptive Stream Detection, *MICRO-39: Proc. of the 39th Annu. IEEE/ACM Int. Symp. on Microarchitecture*, IEEE Computer Society, pp.397–408 (2006).
6) Jouppi, N.P.: Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, *ISCA 1990: Proc. of the 17th Annu. Int. Symp. on Computer Architecture*, New York, NY, USA, ACM, pp.364–373 (1990).
7) Nesbit, K., A.S., D. and Nesbit, K.: AC/DC: An Adaptive Data Cache Prefetcher, *PACT-2004: Proc. of the 13th Int. Conf. on Parallel Architectures and Compilation Techniques*, IEEE Computer Society, pp.135–145 (2004).
8) Nesbit, K. and Smith, J.: Data cache prefetching using a global history buffer, *Micro, IEEE*, Vol.25, No.1, pp.90–97 (2005).
9) Palacharla, S. and Kessler, R.E.: Evaluating stream buffers as a secondary cache replacement, *ISCA 1994: Proc. of the 21st Annu. Int. Symp. on Computer architecture*, Los Alamitos, CA, USA, IEEE Computer Society Press, pp.24–33 (1994).
10) Smith, A.: Sequential Program Prefetching in Memory Hierarchies, *Computer*, Vol.11, No.12, pp.7–21 (1978).
11) Srinath, S., Mutlu, O., Kim, H. and Patt, Y.: Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers, *HPCA-13: IEEE 13th Int. Symp. on High Performance Computer Architecture*, pp.63–74 (2007).
12) Tendler, J., Dodson, J., Fields, J., Le, H. and Sinharoy, B.: POWER4 system microarchitecture, *IBM Journal of Research and Development*, Vol.46, No.1, pp.5–25 (2002).
13) 本城剛毅，石井康雄，入江英嗣，稲葉真理，平木　敬：フィードバックを用いたハイブリッドプリフェッチング方式，情報処理学会研究報告書計算機アーキテクチャ研究会 2009-ARC-184，No.18 (2009).
14) 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，pp.120–121 (2009).