Vol.2018-ARC-231 No.8
2018/6/14

高機能なPrefetcherに適するキャッシュ置換アルゴリズムの検討とHybrid Prefetcherの提案

中村 朋生[1]　甲地 弘幸[1]　入江 英嗣[1]　坂井 修一[1]

概要：メモリ・レイテンシの隠蔽のためにキャッシュ・マネジメントが重要である．この研究では，キャッシュ・マネジメントの中で，キャッシュ置換アルゴリズムとプリフェッチャに注目する．これらの手法は互いに影響し合うことが知られていて，どちらか一方のみを改良しても本来の性能を発揮しない．そこで，特徴的な手法のキャッシュの置換アルゴリズムとプリフェッチャの組み合わせを調査し，プリフェッチャの促進と抑制の切り替えを行う手法を検討する．また，学習速度の違う 2 つのプリフェッチャを用いる Hybrid Prefetcher を提案する．SPEC CPU2006 をシミュレートした結果，Instructions per Cycle(IPC) を 2.5 %向上させることを示す．さらに，学習のためのストレージを共有することで容量を 10 %削減できることを示す．

1. はじめに

プロセッサのコアの動作速度に対して，メイン・メモリへのアクセスにかかるレイテンシは大きいため，レイテンシを隠蔽するためにキャッシュを用いることが普通である．特に，最もメイン・メモリに近いキャッシュである Least Level Cache（LLC）でミスが発生し，レイテンシを隠蔽することが困難なメイン・メモリへのアクセスが増加するため，可能なかぎりミスを減らすキャッシュ・マネジメントが重要である．キャッシュ・マネジメントには，ラインの追い出しを決定する置き換えアルゴリズムと，必要になりそうなデータをあらかじめキャッシュに挿入するプリフェッチ，キャッシュパーティションや way 数の動的制御，コヒーレンシの制御，圧縮による容量の節約などがある．本研究では，互いに与える影響の大きいキャッシュ置換アルゴリズムとプリフェッチという 2 つの技術に焦点を当てる．

現在，キャッシュ置換アルゴリズムとプリフェッチャの研究は多様化しており，どちらかの技術のみに焦点をあてた研究が多い．例えば，置換アルゴリズムに着目すると，キャッシュ置換アルゴリズムのチャンピオンシップである CRC2[1] においては L1，L2 キャッシュには簡単なプリフェッチャを適用するが LLC には適用しない．また，LLC にプリフェッチャを適用する場合でも基本的なプリフェッチャである Stream Prefetcher[2] や Stride Prefetcher[3] を適用する場合が多い．しかし，置換アルゴリズムとプリフェッチャという 2 つの技術が互いに与える影響は大きく，どちらか一方のみを改良しても本来の性能を発揮しない．また，プリフェッチャに着目すると，単一のアルゴリズムを用いて，未来に来るアクセスを予測する．しかし，重視するアドレスのパターンによっては，予測するためにかかる学習時間は変わる．

そこで，本研究では，近年のキャッシュ置換アルゴリズムとプリフェッチャ組み合わせを網羅的に調査し，プリフェッチャの動作に着目した置換アルゴリズムを検討する．また，プリフェッチャに着目すると，学習時間の異なる複数の予測手法を用いる Hybrid Prefetcher を提案する．本研究では短い時間でプリフェッチが可能なストリーミング・アクセスによる予測 [2] と，学習に時間のかかるアドレスの差の相関関係に着目した予測 [4] を同時に用いることで Instructios per Cycle(IPC) を 2.5 %向上することを示す．さらに，学習のためのストレージを共有させることでプリフェッチャを 2 つ適用した場合よりも容量を 10 %圧縮できることを示す．

本研究の貢献は以下の通りである．

- プリフェッチャの影響が少ないアクセス・パターンはプリフェッチャの種類が変わっても不変であることを示した．
- 学習速度の違う 2 つのプリフェッチャを用いることで，IPC が 2.5 %向上することを示した．
- 本研究でハイブリッド化したプリフェッチャのストレージを共有することで，容量を 10 %圧縮した．

以降，第 2 章では，キャッシュ置換アルゴリズム，プリ

[1] 東京大学大学院情報理工学系研究科

フェッチャの既存手法を説明し，第 3 章では，キャッシュ置換アルゴリズムとプリフェッチャの組み合わせについての調査結果と考察を述べる．第 4 章では，Hybrid Prefetcher の概要を述べ，第 5 章では，Hybrid Prefetcher の評価した内容とその結果について述べる．最後に，第 6 章では，この論文をまとめる．

2. 既存のキャッシュ・マネジメント

2.1 置換アルゴリズム

2.1.1 RRIP

オン・チップ・キャッシュで基本となっている LRU[5] では，一度参照がきた後に次の参照が来ないアクセス（ストリーミング・アクセス）や再参照がキャッシュ・サイズを超える長期間後になるアクセス（スラッシング・アクセス）等の時間的局所性に乏しいアクセスが生じると性能が著しく低下する．また，LLC ではウェイの連想度が大きく，ハードウェア構成が複雑になる．そこで，RRIP[6] ではラインの再参照の有無を予測することで，ストリーミング・アクセスとスラッシング・アクセスに耐性を持たせる．RRIP は，新しいラインの挿入位置を操作する Dynamic Insert Policy(DIP)[7] をもとにしており，ラインの挿入時とヒット時のマネジメントを分離することで，再参照のあるデータを保持しやすくしている．

ライン 1 つにつき Re-Reference Prediction Values(RRPV）という M bit のカウンター（一般に M=2 とする）を持ち，アクセスの順番を記録する．RRIP には挿入時の RRPV の値が不変である Static RRIP（SRRIP）とランダムで変化させる Bimodal RRIP(BRRIP)，Set Dueling Moniter(SDM)[7][8] を用いて，SRRIP と BRRIP を切り替える Dynamic RRIP（DRRIP）がある．

LLC ではウェイの連想度が大きく，LRU を用いると，アクセスの順番を記録するために必要なサイズが大きくなってしまうが，RRIP では 2bit なので，メタデータを減らすことができる．

RRIP では，挿入時とヒット時での RRPV の変化は不変，もしくはランダムで決定していた．以降で，アクセスの履歴や種類等を学習し動的に変化させる研究 [9][10][11][12][13] を紹介する．

2.1.2 PACMan

RRIP の構成をベースとして，プリフェッチによる悪影響を防止する置換アルゴリズムが PACMan[9] である．プリフェッチによるラインとデマンドによるラインを挿入時とヒット時の RRPV を変えることで差別化している．挿入時の挙動を変える PACMan-M，ヒット時の挙動を変える PACMan-H，どちらも変える PACMan-HM，これらを動的に切り替える PACMan-DYN があり，本研究では PACMan-DYN を用いる．

2.1.3 SHiP

RRIP の構成をベースとして再参照のされやすいラインを学習することに着目した置換アルゴリズムが Signature-based Hit Prediction(SHiP)[10] である．RRIP と違い，Signature History Counter Table(SHCT) というテーブルに再参照のあるアドレスの PC を記録することで，一度追い出されたアドレスについても学習できる．SHiP では RRPV は 2bit が用いられる．ライン挿入時のアドレスについて，SHCT の学習結果によって RRPV の値を変えることで，再参照されたことがあるラインをより追い出されにくいようにしている．

また，プリフェッチによる SHCT の学習を考慮した SHiP++[11] という手法も提案されている．SHiP++ではプリフェッチとデマンドはそれぞれ別に学習する．また，writeback によるキャッシュの汚染も考慮しており，writeback によって挿入されたラインは追い出されやすくする．

2.1.4 Hawkeye

Hawkeye[12][13] は RRIP の構成をベースとして OPT[14] の考え方を加えた置換アルゴリズムである．実際に未来のアクセスを予測するのではなく，過去のアクセス履歴において最適な置換を PC をキーとして学習する．学習結果を用いて，ラインの挿入時とヒット時の RRPV の値を差別化することで，過去においての最適な置換を再現する．Hawkeye の場合は学習結果を強く反映させるために，RRPV は 3bit が用いられる．

2.2 プリフェッチ

2.2.1 Stream Prefetcher

Stream Prefethcer[2] は連続したアドレス領域にアクセスするストリーミング・アクセスに着目したプリフェッチャである．ストリームを追跡するエントリを持つテーブルにキャッシュへのアクセス履歴を記録する．格納されたアドレスの前後にアクセスがあるかを監視し，同じ方向に複数のアクセスがあった場合にはストリーミング・アクセスが生じていると検出する．ストリーミング・アクセスを検出したエントリが監視している領域にキャッシュ・ミスがあった場合，ストリーミング・アクセス上のアドレスの連続帯をプリフェッチし，監視する領域をずらしてさらに監視する．このとき，プリフェッチをかけてキャッシュに挿入するまでの時間を考慮して，連続したアドレスのうち，少し進んだアドレスの値をプリフェッチする．このアドレスに加算する差分のことを Distance と呼び，プリフェッチするライン数のことを Degree と呼ぶ．

2.2.2 Stride Prefetcher

Stride Prefetcher[3] はループでの構造体や配列へのアクセス等の周期的なアクセスに着目したプリフェッチャである．Stride Prefetch Table(SPT) と呼ばれるテーブルに

PC ごとのアドレスの差を記録する．同一 PC のアクセスが同一の差（stride）をもってアクセスしていれば，検出し，アドレスに stride を加算してプリフェッチする．

2.2.3 Distance Prefetching

Stride Prefetcher では，同じ PC でのメモリアクセスの stride を単一と仮定しているので，stride が単一ではないパターンは検出できない．そこで，複数の stride のパターンに着目したのが，Distance Prefetching[15] である．アクセス履歴をアドレスの差（Delta）の数列として記録する．アドレスを予測する際には記録した Delta から特定の Delta の次に来る確率の高い stride を検出し，アドレスに加算してプリフェッチする．

2.2.4 GHB

Stride Prefetcher，Distance Prefetching やアドレスの相関を予測する Markov Prefetcher[16] などのテーブルに過去のアクセス履歴を格納する必要があるプリフェッチャに対して，データをキーとバリューで格納するのが Global History Buffer(GHB)[17] である．GHB はサーキュラ・バッファを用いたキューで構成されていて，キューにはアクセスされたアドレスとキュー内の同一キーへのポインタを格納する．Stride Prefetcher や Distance Prefetching では PC をキーとしてデータを格納しているが，czone[18] と呼ばれるアドレスの上位 n bit をタグとする方式も提案されている．また，最適な czone のサイズとプリフェッチする Degree を working set[8] を用いて動的に切り替える Adaptive Czone Delta Correlation(AC/DC)[19] という手法も提案されている．

2.2.5 ハイブリッド方式

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 [20] では，3 つプリフェッチャを動的に切り替える手法である．切り替えるための特徴量として予測の信頼度を用いる．信頼度はキャッシュ・ミスの結果からフィードバックする．ハイブリッドにするプリフェッチャは Block Prefetcher[21] と Stride Prefetcher，Delta Correlation Prefetcher であった．

2.2.6 VLDP

czone を用いた Distance Prefetching では同一 czone 内での delta の相関関係のみを考慮していたが，Variable Length Delta Prefetcher(VLDP)[4] では，czone ではなくアドレスの集合をページ単位で考え，delta の相関関係を共有することに着目する．VLDP は Delta History Tabale(DHB) という各ページの情報を格納するテーブルと，Delta Prediction Table(DPT)，Offset Prediction Table(OPT) という delta の相関関係を記録する 2 つのテーブルを持つ．

DHB でページごとのアクセス履歴を記録している．ページごとに最大で直近 4 アクセス分のアドレスの差 (Delta) を記録しており，新しくページにアクセスがあった場合，この Delta の列 (Delta sequence) と DPT によって未来のアクセスのアドレスを予測する．

DPT には特定の Delta の次に来る確率の高い Delta を記録する．Distance Prefetching と違って，単一の Delta をインデックスにするのではなく，最大で 4 アクセス分のアドレスの差，つまり，Delta3 つ分をインデックスとして，その次に来る確率の高い Delta を記録する．これによって，予測の精度を向上している．

また，初めてアクセスするページではアドレスの差を計算することはできない．そこで，ページごとに頻度の高い Delta を OPT に記録して，初めてアクセスしたページで予測をする際に用いることで，プリフェッチする頻度を向上している．さらに，同じアドレスをプリフェッチすることでキャッシュを汚染するのを防ぐために，Prefetch Filter[22] を用いる．

3. 置換アルゴリズムとプリフェッチャの組み合わせの調査

3.1 モチベーション

置換アルゴリズムとプリフェッチャという 2 つの技術が互いに与える影響は大きく，どちらか一方のみを改良しても本来の性能を発揮しない．しかし，LLC 向け置換アルゴリズムは適用する前提として，プリフェッチなしや Stream Prefetcher，Stride Prefetcher といった基本的なプリフェッチャを適用している．だが，前章で述べた GHB や VLDP のように複雑なプリフェッチャも提案されている．そこで，複雑なプリフェッチャを置換アルゴリズムに適用して，性能の傾向を調査する．

3.2 評価環境

アーキテクチャ構成は表 1 のパラメータを用いて，L3 キャッシュの置換アルゴリズムは LRU，RRIP，PACMan，SHiP++，プリフェッチャは Stream Prefethcer，GHB，VLDP を適用した．SRRIP，DRRIP，PACMan，SHiP++での RRPV の値は 0〜3 とし，プリフェッチャの Distance，Degree は置換アルゴリズムによって最も性能の高いパラメータとした．また，GHB は PC ベースで Stride Prefetcher と Distance Prefetching を適用した．シミュレータは当研究室で開発しているサイクル・アキュレイトなシミュレータである鬼斬弐 [23] を用いて，ベンチマークは SPEC CPU2006[24] を利用した．10G 命令分はスキップし，その後の 1G 命令分を計測した．

3.3 性能比較

図 1〜図 8 にそれぞれ Stream Prefetcher，GHB，VLDP を L3 キャッシュに適用し，置換アルゴリズムを LRU，SRRIP，DRRIP，PACMan，SHiP++にした時の IPC を示す．図 1〜図 3 はプリフェッチャごとに，図 4〜図 8 では置換アルゴリズムごとにグラフを分割している．IPC は，

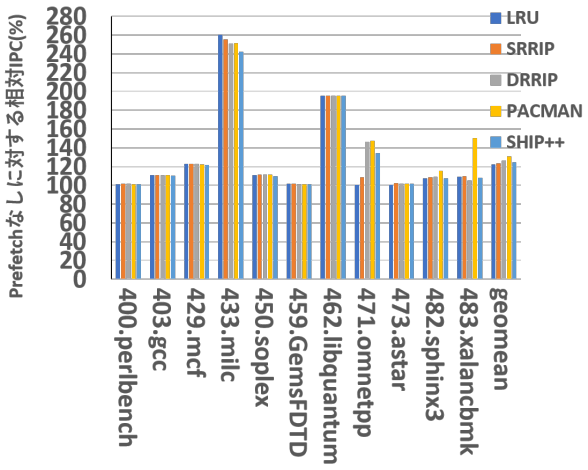

図 1　L3 キャッシュに Stream Prefetcher を適用した際の相対 IPC

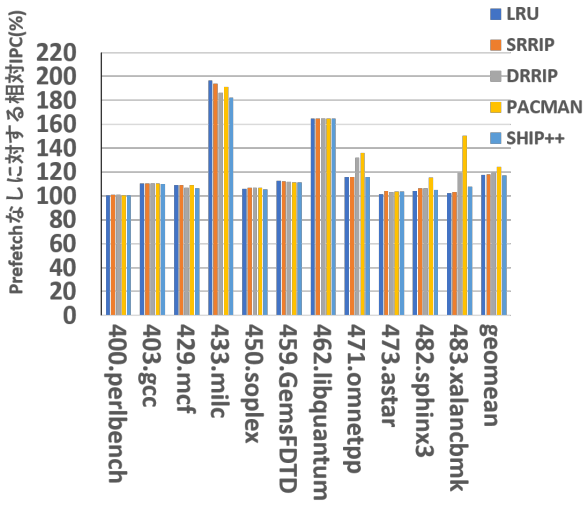

図 2　L3 キャッシュに GHB を適用した際の相対 IPC

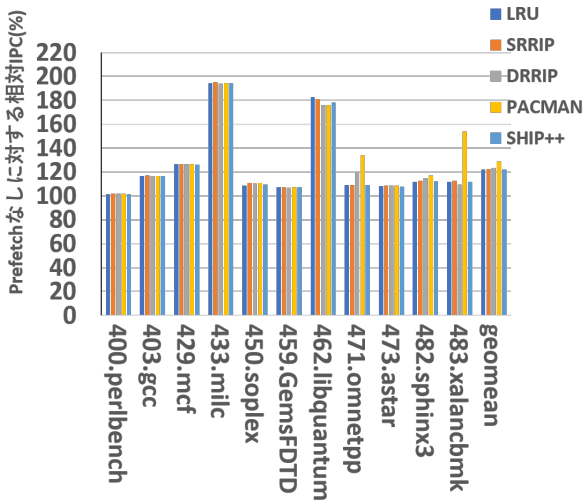

図 3　L3 キャッシュに VLDP を適用した際の相対 IPC

L3 に Prefetch を適用せず LRU を適用した場合をベースとして，これに対する性能の相対値で示す．なお，ベースに対して差異が 1 %未満のベンチマークは除外した．

表 1　アーキテクチャの構成

プロセッサ	
issue width	int:2, fp:2, mem:2
instruction window	int:32, fp:16, mem:16
branch predictor	8KB, g-share
BTB	2KB entry, 4way
LSQ	load:48 entry，store:48 entry
キャッシュ	
L1 I/D キャッシュ	32KB, 4way, 64B line, 3cycle latency, LRU
L2 キャッシュ	256KB, 8way, 64B line, 10cycle latency, LRU
L3 キャッシュ	2MB, 16way, 64B line, 24cycle latency
メインメモリ	200cycle latency
プリフェッチャ	
L1D プリフェッチャ	NextLine Prefetcher, Distance:0, Degree:1
L2 プリフェッチャ	Stream Prefetcher, Distance:4, Degree:4

どのプリフェッチャにおいても，433.milc，462.libquantum で大きく性能が向上している．この 2 つのベンチマークにはストリーミング・アクセスが多量あり，プリフェッチでの予測が当たりやすいためである．ストリーミング・アクセスに対して，Stream Prefetcher が非常に効果的なので，有用なのはプリフェッチによるラインの挿入の阻害しないアルゴリズムである．DRRIP や PACMan では，スラッシング・アクセス耐性のために新しいラインをランダムな位置に挿入する．これによって，プリフェッチが挿入するラインが参照される前に追い出して，プリフェッチャを阻害してしまう．LRU だとプリフェッチとデマンドのアクセスを区別せず最も追い出されにくい位置に挿入する．つまり，ストリーミング・アクセスに対しては Stream Prefetcher と LRU の組み合わせが最適である．

459.GemsFDTD に着目すると，プリフェッチャとして GHB を用いた際に IPC が約 12 %と向上し，LRU を用いた際に最も高くなっている．このベンチマークは，ループでの配列や構造体の参照が多く，同一 PC で異なるメモリアドレスへのアクセスが多いので，PC ベースの GHB の予測が当たりやすいためである．このアクセスパターンでも 433.milc，462.libquantum と同様にプリフェッチによるラインを挿入しない置換アルゴリズムである LRU との組み合わせが最適である．

429.mcf において，VLDP を用いた際に他のプリフェッチャに対して約 26 % IPC が向上している．429.mcf については VLDP で予測しやすい似たような空間局所性が色々なアドレス帯で起こるようなアクセスなので，プリフェッチによるラインの挿入の阻害しないようなアルゴリズムで

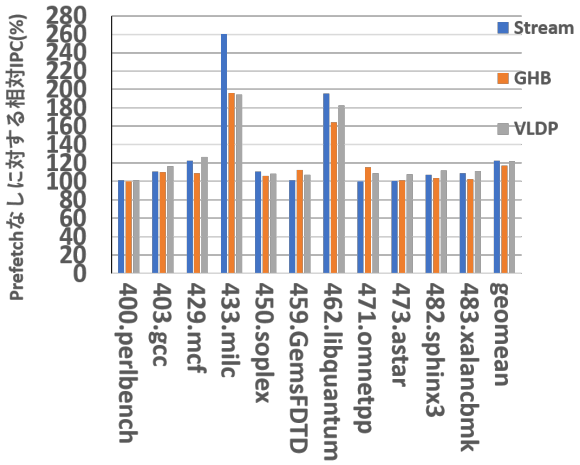

図 4　L3 キャッシュに LRU を適用した際の相対 IPC

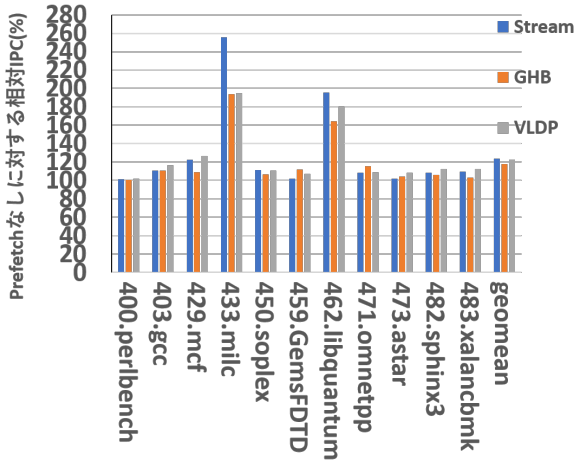

図 5　L3 キャッシュに SRRIP を適用した際の相対 IPC

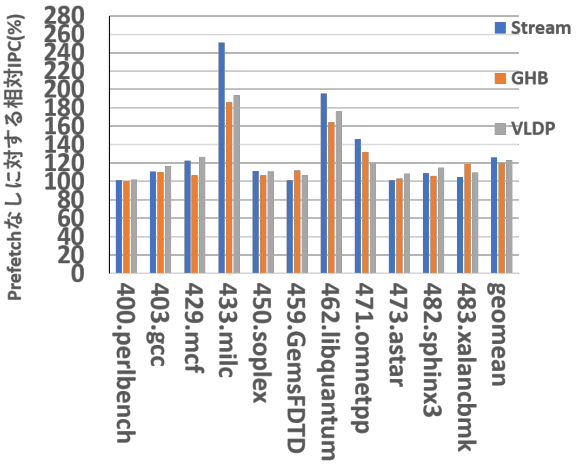

図 6　L3 キャッシュに DRRIP を適用した際の相対 IPC

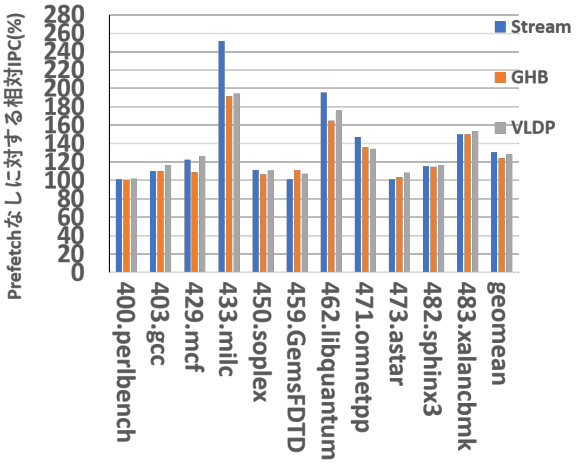

図 7　L3 キャッシュに PACMan を適用した際の相対 IPC

ある LRU が最適である．

471.omnrtpp，473.xalancbmk では，どのプリフェッチャを用いても PACMan が IPC を最大化している．これらのベンチマークでは，アクセスがパターン化できず，参照されるアドレスの予測が難しいので，置換アルゴリズムで不要なプリフェッチを除くことが有効である．また，アクセスのパターンが複雑なので SHiP++のようなプリフェッチに対するペナルティを学習が難しく，プリフェッチによって挿入されたライン全体にペナルティを加える PACMan のような置換アルゴリズムが最適である．

3.4　調査結果

置換アルゴリズムとプリフェッチャの組み合わせを網羅的に調査することで，2 つの調査結果が得られた．

1 つめは，プリフェッチャに着目した調査結果である．プリフェッチャの種類がなんであっても，プリフェッチャを適用することで性能が向上するアクセス・パターンの種類は変わらない．しかし，そのアクセス・パターンないでの性能の優劣は存在する．

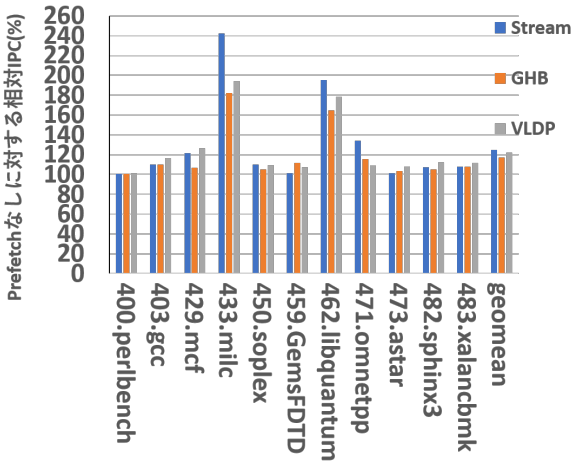

図 8　L3 キャッシュに SHiP++を適用した際の相対 IPC

2 つめは，置換アルゴリズムに着目した調査結果である．，PACMan と Stream Prefetcher を適用した場合に幾何平均

での性能が約 31 %向上と最大化することがわかった．しかし，プリフェッチが有効なベンチマークでは LRU 適用時の方が約 160 %向上と性能の高いベンチマークも存在する．各々のベンチマークについて置換アルゴリズムとプリフェッチャの理想的な組み合わせによる幾何平均をとると，約 35 %性能向上する．つまり，プリフェッチャによるラインを総じて追い出しやすくするのではなく，アクセス・パターンによって，プリフェッチが効果的か否かを学習し，ヒット・ミス時のポリシーを変える手法が有効的である．

4. Hybrid Prefetcher の提案

4.1 モチベーション

3 章より，プリフェッチャを適用することで性能が大きく上昇することがわかった．特に，その中でプリフェッチャによってより効果の大きいプリフェッチャは異なる．3 章で幾何平均が同程度の性能であった Stream Prefetcher と VLDP を着目する．

図 7 をみると，433.milc や 462.liquantum では，Stream Prefetcher を用いた際の IPC の向上効果が大きい．しかし，403.gcc，429.mcf，482.sphinx3 等のストリーミング・アクセスばかりではないパターンの場合，VLDP を用いた方が性能が向上する．特に 459.GemsFDTD，473.astar では VLDP では約 10 %性能向上している．

これらより，ストリーミング・アクセスが多い単純なアクセス・パターンでは学習時間の短い Stream Prefetcher が有効で，複雑なアクセス・パターンだと多少学習時間のかかるが複雑なアドレスを予測できる VLDP が有効なことがわかる．そこで，Stream Prefetcher と VLDP をハイブリッド化することで，性能向上させることを提案する．また，Stream Prefetcher と VLDP のストレージを共有化することで，単純に 2 つのプリフェッチャを適用する場合よりも，容量を削減できることを示す．

4.2 Hybrid Prefetcher の動作

2 章で説明した VLDP の構成と同様に，DHB，DPT，OPT の 3 つのテーブルを用いる．そして，VLDP にストリーミング・アクセスを検出する機構を追加する．ゆえに，Delta の相関関係を検出する DPT と OPT に関しては VLDP と変わらず，DHB にエントリを追加する．Stream Prefetcher を単独で用いた際に，最適な監視するストリームのエントリ数は 64 本であり，DHB のエントリ数は 16 本なので，DHB の 1 つのエントリで 4 つのストリームを追跡させる．

DHB のエントリはページ番号がインデックスとなっているので，エントリ内のみで追跡を行うと，ページを跨ぐストリーミング・アクセスを追跡することはできない．なので，エントリ内のストリーミング・アクセスの監視する領域がページを跨ぐと，もとのページ内での追跡するエントリを無効化し，跨いだ先のページのインデックスに追跡するエントリを格納する．

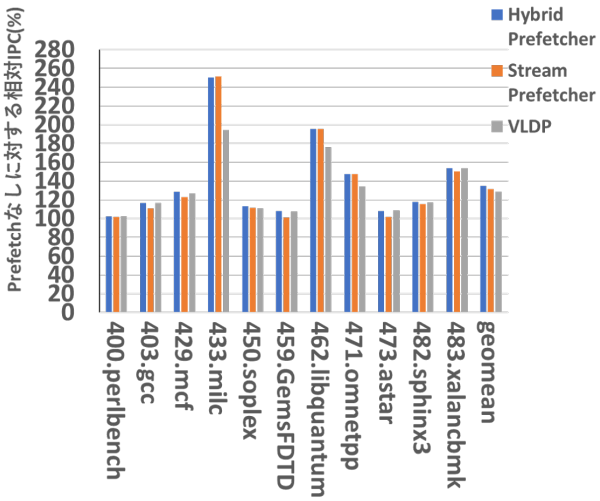

図 9 Hybrid Prefetcher の性能

また，VLDP はキャッシュ・ミス時にのみプリフェッチによるラインの挿入を行うが，Stream Prefetcher では監視するアドレス帯にアクセスがあれば，ヒット時にもプリフェッチによるラインの挿入を行う．よって，DHB にはヒット・ミスにかかわらずアクセスをするようにし，ミス時のみにテーブルの更新を行う．

5. Hybrid Prefetcher の評価

5.1 評価環境

3 と同様にアーキテクチャ構成は表 1 のパラメータを用いて，L3 キャッシュの置換アルゴリズムは PACMan，プリフェッチャは Stream Prefethcer，VLDP，Hybrid Prefetcher を適用した．プリフェッチャの Distance，Degree は最も性能の高いパラメータとした．

5.2 性能評価

図 9 に Stream Prefetcher，VLDP，Hybrid Prefetcher を L3 キャッシュに適用した時の IPC を示す．3 章と同様に，IPC は，L3 に Prefetch を適用せず LRU を適用した場合をベースとして，これに対する性能の相対値で示す．なお，ベースに対して差異が 1 %未満のベンチマークは除外した．Stream Prefetcher を適用した際に大きく IPC が向上する 433.milc，462.liquantum に着目すると，Hybrid Prefetcher を適用した場合でも IPC は同程度に向上する．また，VLDP を適用した際に IPC が向上するベンチマークに着目すると，403.gcc，473.astar， 483.xalancbmk では VLDP 適用時に対して IPC が約 0.8 %低下しているが，429.mcf，450.soplex，459.GemsFDTD，482.sphinx3 では VLDP 適用時よりも IPC が向上している．

つまり，Stream Prefetcher が効果的なアクセス・パターンでは Stream Prefetcher と同程度の性能向上を，VLDP が効果的なアクセス・パターンでは VLDP と同程度の性能向上をすることがわかる．ベンチマークの幾何平均に着目すると，Hybrid Prefetcher 適用時が，Stream Prefetcher 適用時に対して約 2.5 % IPC が向上している．

5.3 面積評価

図 10 に各プリフェッチャのストレージのエントリを示す．

Stream Prefetcher のテーブルのエントリは監視するアドレスと状態を表すフィールドで構成される．エントリの状態は初期化している場合，アドレスを追跡しているだけで予測はしていない場合，アドレスの追跡と予測を同時に行う場合の 3 種類ある．本研究では追跡の期間をアクセス 2 回としているので，3bit 必要である．また，エントリの置換のために nMRUbit(1bit) とアドレスの追跡する方向 (正負) を表すのに 1bit 必要である．アドレスは 64bit を想定しているが，キャッシュのオフセットが 6bit なので，58bit あればよい．これらより，1 エントリあたり 62bit 必要で，エントリ数は 64 本なので，512B 必要となる．

次に VLDP に着目する．DHB にはページの上位 15bit，ページとオフセットを除いたアドレス 43bit，delta4 つで 32bit，delta の数を判定するためのページのアクセス回数を表す 3bit，nMRUbit の合計 94bit 必要である．エントリ数は 16 本なので，188B 必要である．また，OPT に 72B，DPT には 648B 必要なので，合計して 808B 必要となる．

これらより，単に VLDP と Stream Prefetcher のストレージを別々に確保すると 1320B 必要となる．しかし，ストリームと DHB は重複したデータを記録しているので，容量を削減できる．4.2 章で述べたように，DHB のエントリにストリームを追跡するエントリを 4 つ追加する．このとき，DHB のエントリはページ単位なので，記録するアドレスは 43bit でよい．さらに，ストリームの状態を表すフィールドより，ページにアクセスした回数がわかるので，DHB にもともとあったアクセス回数を表すフィールドを削減できる．これによって DHB のエントリ 1 つに 287bit 必要となるが，ストリームのストレージが不要となる．DHB のエントリ数は変わらず 16 本なので，574B 必要である．このとき，全体で 1194B 必要となる．よってストレージを別々に確保した際に対して容量を約 10 %圧縮することができた．

6. おわりに

本研究では，キャッシュ・マネジメントに関する技術の中で互いに与える影響の大きい置換アルゴリズムとプリフェッチャに着目した．

置換アルゴリズムとプリフェッチャの最適な組み合わせの調査によって，プリフェッチの効果が高いアクセス・パターンの中には PACMan のような手法ではなく，LRU のような手法が適するパターンが存在することを示した．そこで，プリフェッチャによるラインを総じて追い出しやすくするのではなく，アクセス・パターンによって，プリフェッチが効果的か否かを学習し，ヒット・ミス時のポリシーを変える手法が有効的であることを示した．

また，Stream Prefetcher と VLDP を同時に適用することを提案し，今回の評価により， Stream Prefetcher に対して IPC を約 2.5 %向上することを示した．また，2 つのプリフェッチャの学習用ストレージを共有することで，同時に適用する場合に対して，容量を約 10 %削減することを示した．

今後の課題は，PC をベースとしないプリフェッチャの効果による学習方法と，ハイブリッド化して性能の落ちるアクセス・パターンの原因を議論することである．

謝辞 本論文の研究は一部，共同研究「Deep Learning 向け Straight アーキテクチャの研究」（株式会社富士通研究所）による．

参考文献

[1] : CRC2, http://crc2.ece.tamu.edu/.
[2] Smith, A. J.: Sequential Program Prefetching in Memory Hierarchies, *Computer*, Vol. 11, No. 12, pp. 7–21 (1978).
[3] Fu, J. W. C., Patel, J. H. and Janssens, B. L.: Stride Directed Prefetching In Scalar Processors, *[1992] Proceedings the 25th Annual International Symposium on Microarchitecture MICRO 25*, pp. 102–110 (online), DOI: 10.1109/MICRO.1992.697004 (1992).
[4] Shevgoor, M., Koladiya, S., Balasubramonian, R., Wilkerson, C., Pugsley, S. H. and Chishti, Z.: Efficiently Prefetching Complex Address Patterns, *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 141–152 (online), DOI: 10.1145/2830772.2830793 (2015).
[5] Coffman, E. G. and Denning, P. J.: *Operating Systems Theory*, Vol. 973, Prentice-Hall Englewood Cliffs, NJ (1973).
[6] Jaleel, A., Theobald, K. B., Steely, Jr., S. C. and Emer, J.: High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP), *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, New York, NY, USA, ACM, pp. 60–71 (online), DOI: 10.1145/1815961.1815971 (2010).
[7] Qureshi, M. K., Jaleel, A., Patt, Y. N., Steely, S. C. and Emer, J.: Adaptive Insertion Policies for High Performance Caching, *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, New York, NY, USA, ACM, pp. 381–391 (online), DOI: 10.1145/1250662.1250709 (2007).
[8] Denning, P. J.: Working Sets Past and Present, *IEEE Transactions on Software Engineering*, Vol. SE-6, No. 1, pp. 64–84 (online), DOI: 10.1109/TSE.1980.230464 (1980).
[9] Wu, C.-J., Jaleel, A., Martonosi, M., Steely, Jr., S. C. and Emer, J.: PACMan: Prefetch-Aware Cache Management for High Performance Caching, *Proceedings of the 44th Annual IEEE/ACM International*

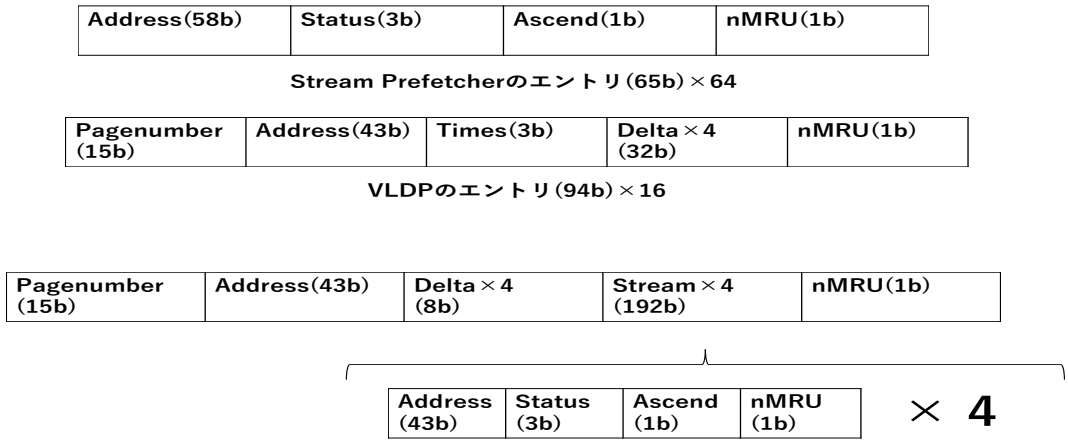

図 10 Prefetcher のストレージ

Symposium on Microarchitecture, MICRO-44, New York, NY, USA, ACM, pp. 442–453 (online), DOI: 10.1145/2155620.2155672 (2011).

[10] Wu, C.-J., Jaleel, A., Hasenplaugh, W., Martonosi, M., Steely, Jr., S. C. and Emer, J.: SHiP: Signature-Based Hit Predictor for High Performance Caching, MICRO-44, New York, NY, USA, ACM, pp. 430–441 (online), DOI: 10.1145/2155620.2155671 (2011).

[11] Young, V., Chou, C.-C., Jaleel, A. and Qureshi, M.: SHiP++: Enhancing Signature-Based Hit Predictor for Improved Cache Performance (2017).

[12] Jain, A. and Lin, C.: Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement, IEEE, pp. 78–89 (2016).

[13] Jain, A. and Lin, C.: Hawkeye: Leveraging Belady's Algorithm for Improved Cache Replacement (2017).

[14] Belady, L. A.: A Study of Replacement Algorithms for a Virtual-Storage Computer, *IBM Systems journal*, Vol. 5, No. 2, pp. 78–101 (1966).

[15] Kandiraju, G. B. and Sivasubramaniam, A.: Going the Distance for TLB Prefetching: An Application-Driven Study, *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ISCA '02, Washington, DC, USA, IEEE Computer Society, pp. 195–206 (2002).

[16] Joseph, D. and Grunwald, D.: Prefetching Using Markov Predictors, *IEEE Transactions on Computers*, Vol. 48, No. 2, pp. 121–133 (online), DOI: 10.1109/12.752653 (1999).

[17] Nesbit, K. J. and Smith, J. E.: Data Cache Prefetching Using a Global History Buffer, *Software, IEE Proceedings-*, IEEE, pp. 96–96 (2004).

[18] Palacharla, S. and Kessler, R. E.: Evaluating Stream Buffers as a Secondary Cache Replacement, *Computer Architecture, 1994., Proceedings the 21st Annual International Symposium On*, IEEE, pp. 24–33 (1994).

[19] Nesbit, K. J., Dhodapkar, A. S. and Smith, J. E.: AC/DC: An Adaptive Data Cache Prefetcher, *Proceedings. 13th International Conference on Parallel Architecture and Compilation Techniques, 2004. PACT 2004.*, pp. 135–145 (online), DOI: 10.1109/PACT.2004.1342548 (2004).

[20] 本城剛毅，石井康雄，入江英嗣，稲葉真理，　平木敬：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，研究報告計算機アーキテクチャ (ARC)，Vol. 2010, No. 14, pp. 1–6 (2010).

[21] Gindele, J. D.: Buffer Block Prefetching Method, *IBM Technical Disclosure Bulletin*, Vol. 20, No. 2, pp. 696–697 (1977).

[22] Zhuang, X. and Lee, H. h. S.: Reducing Cache Pollution via Dynamic Data Prefetch Filtering, *IEEE Transactions on Computers*, Vol. 56, No. 1, pp. 18–31 (online), DOI: 10.1109/TC.2007.250620 (2007).

[23] 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 SACSIS2009，Vol. 2009, No. 4, pp. 120–121 (2009).

[24] : Standard Performance Evaluation Corporation. Spec cpu 2006.