Vol.2022-ARC-248 No.17
Vol.2022-SLDM-198 No.17
Vol.2022-EMB-59 No.17
2022/3/10

アドレスとタイミングの予測を分離したデータプリフェッチャ

小泉 透[1,a)]　中村 朋生[1]　出川 祐也[1]　入江 英嗣[1]　坂井 修一[1]　塩谷 亮太[1]

概要：プリフェッチはキャッシュミスを削減することでプロセッサの性能を向上させる技術である．既存のプリフェッチャの多くは，メモリレイテンシを隠蔽できるよう，デマンドアクセスより十分早くプリフェッチを発行することに焦点を当てていた．しかし，プリフェッチがデマンドアクセスより早すぎると，プリフェッチしたラインが利用前にキャッシュから追い出されてしまい，性能向上につなげることができない．我々は既存のプリフェッチャがそのような早すぎるプリフェッチを多く発行しており，そこに大きな性能向上の機会があることを見出した．この観察に基づき，我々はプリフェッチすべきタイミング自体をアドレス予測と分離して行うプリフェッチャ T-SKID を提案する．T-SKID は既存のプリフェッチャのように十分早くプリフェッチを発行することに加え，必要であれば適切なタイミングまでプリフェッチの発行を遅らせることができる．SPEC CPU 2017 を用いたシミュレーションにより T-SKID を評価したところ，最新のプリフェッチャ IPCP と比べてシングルコア性能で 1.5%，マルチコア性能で 5.6%高い性能を得た．

1. はじめに

キャッシュミス時に生じるメインメモリアクセスのレイテンシは一般に非常に長く，多くのアプリケーションにとって実行のボトルネックとなる．プリフェッチは，デマンドアクセスより前に必要となるデータをキャッシュに挿入することで，この問題を解決する技術である．プリフェッチでは，どのアドレスに存在するデータが必要になるかを正確に予測する必要があり，これまでに様々なプリフェッチ手法が研究されてきた．たとえば，ストリームアクセスやストライドアクセスといった簡単なメモリアクセスパターンを捉えて予測する手法 [1], [2], [3], [4] から，アドレスの差の系列やアクセス履歴とロード命令の PC を複合的に用いることで複雑なメモリアクセスパターンにも対応できる手法 [5], [6], [7], [8], [9] まで，様々なものが研究されている．

プリフェッチにおいて，正しいアドレスを予測することは重要であるが，十分に早くプリフェッチを発行することも同様に重要である．たとえアドレスを正しく予測できても，デマンドアクセスに対して遅すぎるタイミングでのプリフェッチ発行は，レイテンシを有効に隠蔽することができず十分な性能向上を実現できない．そのため，多くの研究者らが十分に早くプリフェッチを発行することに着目してきた．典型的なプリフェッチャでは，アドレス予測の確信度が十分に高い範囲で，なるべく遠い未来のアクセスを予測する [1], [2], [5], [6], [10]．また，Best-Offset Prefetcher (BOP) [11] はアドレス予測にタイミング予測を埋め込んでおり，デマンドアクセスに間に合うかどうかを考慮してプリフェッチアドレスを決定する．

既存のプリフェッチャがプリフェッチを十分に早く発行しようとしていたのに対し，我々は逆にプリフェッチを**遅らせる**ことに着目した．デマンドアクセスよりも早すぎるタイミングでプリフェッチを発行してしまうと，プリフェッチされたラインは使用される前にキャッシュから追い出されてしまう．このような場合，プリフェッチの発行を遅らせることで性能を向上させることができる．

上記の場合の特殊だが頻出の例として，キャッシュから追い出されるほど時間をあけての同じラインへの再参照がある．我々はこのような再参照を**ゼロストライド**パターンと名付けた．ゼロストライドパターンは同一アドレスへの再参照であるため，予測すべきアドレスは自明である．たとえばストライドプリフェッチャにおいてストライドをゼロと設定すれば，すなわち最初のアクセスと同じアドレスを生成すれば，アドレスを正しく予測できる．しかし，最初のアクセスの際に同じアドレスをプリフェッチしても当然に意味をなさない．そうではなく，そのアドレスのライ

[1] 東京大学 大学院情報理工学系研究科
[a)] koizumi@mtl.t.u-tokyo.ac.jp

Vol.2022-ARC-248 No.17
Vol.2022-SLDM-198 No.17
Vol.2022-EMB-59 No.17
2022/3/10

ンが一度追い出された後までプリフェッチの発行を遅らせることで，有効なプリフェッチを行い性能向上につなげることができる．

我々は，このようにプリフェッチを遅らせることで性能向上に繋がるケースが多くあり，そこに大きな性能向上の機会があることを見出した．この観察に基づき，我々は従来のプリフェッチャのように早くプリフェッチをすることに加え，プリフェッチを遅らせることもできる T-SKID プリフェッチャを提案する．T-SKID はアドレス予測とは独立に，プリフェッチの適切なタイミングを予測する．タイミングの予測にはデータアドレスを用いず，ロード命令の PC 間の時間的な相関を利用する．あるミスが起きた際に予測されたアドレスを用いて即座にプリフェッチを発行するのではなく，プリフェッチを行うべきと予測されたタイミングまでプリフェッチを遅らせる．これにより従来のプリフェッチャでは予測が困難なケース，すなわち挿入が早すぎてそのラインがデマンドアクセスまでに置き換えられてしまうケースやゼロストライドパターンでも，T-SKID は有効なプリフェッチの発行ができる．

本研究の貢献は以下のとおりである．

- 既存のプリフェッチャが早すぎるプリフェッチを多数発行していることを見出し，プリフェッチを**遅らせる**ことによる性能向上の機会があることを示した．
- アドレス予測とタイミング予測の分離された T-SKID を提案した．T-SKID は従来のプリフェッチャが行っていたように十分に早くプリフェッチを発行することに加え，必要に応じて適切なタイミングまでプリフェッチの発行を遅らせることができる．また，ゼロストライドパターンとと呼ぶ，従来のプリフェッチャではうまく扱えない再参照パターンでも T-SKID は有効なプリフェッチを発行できる．
- データプリフェッチに利用できる新たな相関関係として PC 間の順序およびタイミングの再現性を提案し，またそれを利用するための新しい機構を提案した．
- 提案した T-SKID を the 3rd Data Prefetching Championship (DPC3)[12] と同じコンフィギュレーションにおいてシミュレーションし，評価した．プリフェッチ無しの場合に対する性能向上の全ベンチマークにおける幾何平均は，シングルコアでは 46.0%，マルチコアでは 26.2% を達成した．この性能向上値は，DPC3 で優勝した Instruction Pointer Classifier based Prefetching (IPCP) をシングルコアでは 1.5%，マルチコアでは 5.6%上回るものである．

2. モチベーション

プリフェッチャがアドレスを正しく予測しても，デマンドアクセスよりも早すぎるタイミングでプリフェッチを発行すると，プリフェッチされたラインは使用される前に

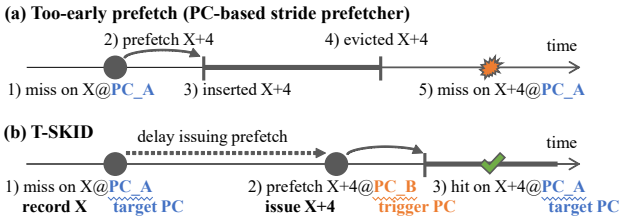

図 1 早すぎるプリフェッチと T-SKID.

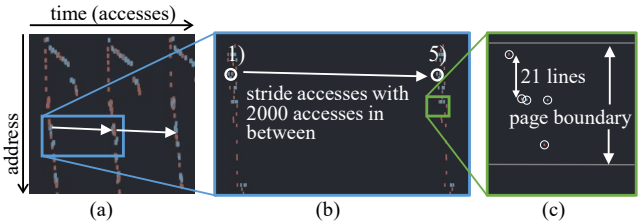

図 2 607.cactuBSSN_s-2421B のメモリアクセスパターン．縦軸がアクセスアドレスを，横軸が時間を表す．図中の各点がメモリアクセスを表す．(a) アクセスパターンを可視化したもの．白矢印はストライドアクセスを示している．この図の中には，千以上のストライドアクセスが存在する．(b) 拡大図．左端の系列と右端の系列とがストライドアクセスを構成している．(c) さらに拡大した図．アクセスは非常にスパースである．

追い出されてしまいミスが生じる．このようなミスの例を図 1(a) に示す．この例ではプリフェッチャとして PC ベースのストライドプリフェッチャを用いている．この図において，(1) アドレス X でミスが生じたため，(2) ストライドプリフェッチャはあらかじめ学習していたストライド幅 4 に基づきアドレス X+4 のプリフェッチを発行する．(3) プリフェッチされたラインが挿入された後，(4) 参照されるまでに多くの命令が実行されたためそのラインは追い出される．(5) 結果としてアドレス X+4 へのアクセスはプリフェッチがされていたにも関わらずデマンドミスを生じる．このようなミスのことを，以下では **misses prefetched too-early** と呼ぶ．

そのような実際のアクセスパターンの例を，SPEC CPU 2017 に含まれる 607.cactuBSSN_s-2421B のメモリアクセスパターン（図 2）を使って説明する．この図では左から右に時間が進んでおり，(a) 内の矢印はストライドアクセスを構成している．(a) の一部を拡大した (b) 内の 1) と 5) は，同一の PC によるストライドアクセス系列であり，PC ベースのストライドプリフェッチャは容易にアドレスを予測できる[*1]．しかし 1) から 5) までの間にはおよそ 2000 ラインがアクセスされており，これは L1 キャッシュのサイズ（512 ライン）を大きく超える．この結果，1) でプリ

[*1] なお，空間的な相関に基づくプリフェッチャ（spatial memory streaming (SMS) [7] や Bingo [8], [13]）はこれらのアドレスを予測できない．これらのプリフェッチャはページ単位で空間的な相関を捉えているため，拡大図 (c) に示すようにページ内でアクセスされるラインが非常に少なく，またアクセスパターンがアドレス方向（縦方向）に不規則である場合に対応することができない．

Vol.2022-ARC-248 No.17
Vol.2022-SLDM-198 No.17
Vol.2022-EMB-59 No.17
2022/3/10

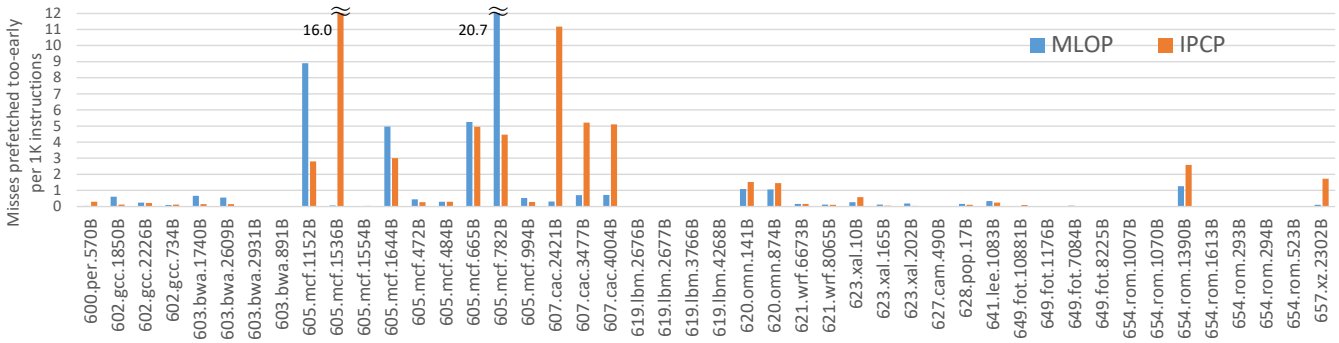

図 3 各トレースにおける，1000 命令当たりの Misses prefetched too-early の数.

フェッチしたラインは 5) までの間に追い出されてしまう．

我々は，既存のプリフェッチャが早すぎるプリフェッチを多く発行しており，プリフェッチを遅らせることによる大きな性能向上の機会があることを見出した．既存のプリフェッチャを使用した際，L1 キャッシュにおいて misses prefetched too-early がどの程度発生しているか，DPC3 で使用された SPEC CPU 2017 のトレース 46 本を用いて調査した．ここでは，DPC3 で入賞した，近年で最も性能が高いと考えられるプリフェッチャである IPCP と MLOP [10] を用いた．図 に，各トレースにおける 1000 命令当たりの misses prefetched too-early の数を示す．その値が 1 より大きいトレースの数は，IPCP で 12 本，MLOP で 7 本だった．また，その値は IPCP では最大 16.0，MLOP で最大 20.7 だった．これらの結果から，プリフェッチを適切に遅らせることによる性能向上の機会があることが分かる．

3. T-SKID

3.1 概略

2 章で説明したように，我々はさらなる性能向上のカギは適切にプリフェッチの発行を遅らせることにあると考えた．我々は、そのようなプリフェッチ・タイミングの調整を実現する T-SKID を提案する。

T-SKID は，アドレス予測だけでなくタイミング予測も行い，これら双方の予測は独立に行う．アドレス予測には PC ベースのストライド予測器を用いる．また，タイミング予測のためには，PC の順序の再現性を利用する．

図 1(b) に，T-SKID の動作の概略を示す．この図では，図 1(a) と同じようにアドレス X と X+4 にアクセスが行われている．図 1(a) では，前述したようにプリフェッチを早すぎるタイミングで発行したためミスが生じた．これに対し，T-SKID はプリフェッチの発行を図 1(b) 内の (2) の PC_B よるロードのアクセスまで遅らせる．これにより，プリフェッチしたラインは追い出されず，(3) の PC_A のアクセスはキャッシュヒットとなる．

既存の PC ベースなプリフェッチャと異なり，T-SKID ではプリフェッチを発行するタイミングを決めるロード命令の PC（PC_B）は，アドレス予測に使われる PC（PC_A）と異なってもよい．PC_A のような，アドレス予測に使われる PC を **target PC** と呼ぶ．また，PC_B のような発行タイミングを決めるロード命令の PC を **trigger PC** と呼ぶ．T-SKID は，target PC に対して適切な発行タイミングを実現する trigger PC を学習する．このタイミング学習により適切なタイミングでのプリフェッチが可能となる．

以下ではこのような T-SKID の動作を実現する実装について述べる．まず T-SKID の構成を説明し，その後で学習や予測の詳細を説明する．

3.2 構成

T-SKID は，図 4 に示すように，以下の四つのコンポーネントを持つ．次の二つはそれぞれアドレス予測とタイミング予測に使われるテーブルである．

1. **Address prediction table:** このテーブルは，PC ベースのストライド予測を行うために必要な情報（`stride`, `last_addr`, `degree` 等）を保持する．
2. **Target table:** このテーブルは，trigger PC と target PC の対応関係を保持する．

次の二つは、キャッシュフィルを監視し，target table の学習を行うためのコンポーネントである。

3. **Inflight prefetch table (IPT):** このテーブルは，フィルが完了していない発行済みプリフェッチの情報（その trigger PC とプリフェッチ対象アドレス）を記録する．この情報は，プリフェッチを追跡しレイテンシを測定するために用いる．
4. **Recent request PC queue (RRPCQ):** このキューは，直近にフィルが完了したプリフェッチの trigger PC を記録する．

3.3 プリフェッチの発行

T-SKID のプリフェッチ発行時の動作を，図 4(a) に示す．T-SKID は，以下の手順でプリフェッチを発行する．

(1) まずタイミング予測のために，ロード命令の実行ごとに，その PC を trigger PC として target table を引く．

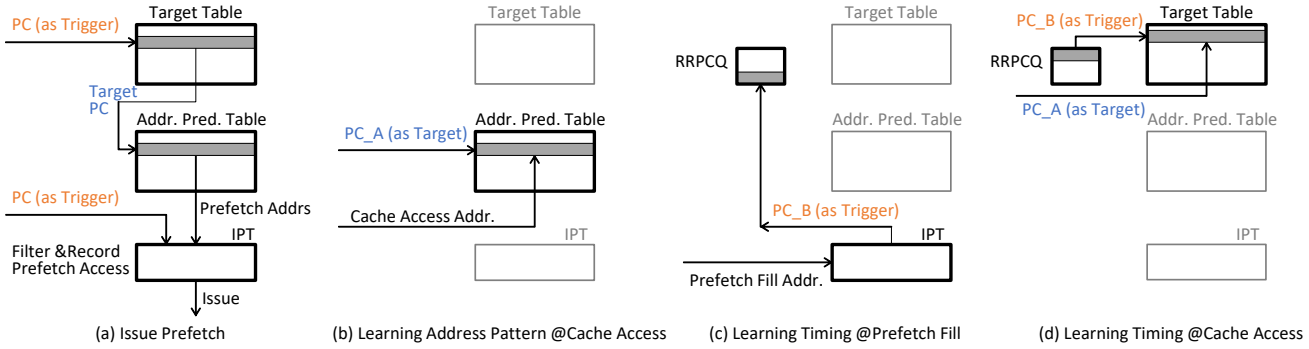

図 4 T-SKID におけるプリフェッチの発行と，アドレスとタイミングの学習.

もし target table にヒットし target PC が得られた場合は，次に示すアドレス予測を行う.

(2) アドレス予測のため，target PC で address prediction table を引く．address prediction table にヒットした場合，得られた情報 を用いてプリフェッチすべきアドレスを計算し，プリフェッチを発行する．アドレスは last_addr + stride × {1, 2, . . . , degree} により求める.

(3) ここで発行したプリフェッチについて，その trigger PC とプリフェッチ対象アドレスの組を IPT に記録する。この情報は，後ほど説明する timing learning のために使われる.

3.4 アドレス予測器の学習

T-SKID は，既存の PC ベースのストライドプリフェッチャとほぼ同様に予測と学習を行う．前述したように，address prediction table には、target PC ごとの `last_addr` と `stride` が記録されている．図 4(b) に示すように，PC_A の命令によるメモリアクセスごとに，PC_A で address prediction table を引き，対応するエントリの `last_addr` と `stride` を更新する．なお，`degree` は後述する timing learning により独立に更新される．また，複数の PC が単一のストリームアクセスを発生させる場合に対処するため，`last_addr` から一定範囲内の直近のアクセスの数を数え，しきい値を超えた場合は `stride` をラインサイズに設定する.

T-SKID は既存の PC ベースのストライドプリフェッチャとは異なり，ストライドが 0 であったとしても，それを有効なストライドとして学習する．既存の PC ベースのストライドプリフェッチャではストライドが 0 の場合にプリフェッチを発行しても意味がないが，T-SKID ではそのプリフェッチの発行を遅らせることにより，1 章で述べた長期再参照によるゼロストライドパターンをうまくプリフェッチすることができる.

3.5 タイミング予測器の学習

T-SKID はプリフェッチにかかる時間を学習し，適切なタイミングでプリフェッチを発行する．T-SKID はプリフェッチを発行してからそのラインが挿入されるまでにかかる時間を，具体的なサイクル数として記憶するのではなく，trigger PC と target PC の紐づけの中に埋め込む．すなわち，trigger PC のロード命令実行時にプリフェッチすればそのフィルが target PC のロード命令でのアクセスに間に合う，という条件を満たすような trigger PC と target PC の紐付けを学習する.

以下ではこのタイミング学習の動作を説明する.

(1) 前述したように，プリフェッチを発行する際に，そのアドレスと発行をトリガした命令の PC を紐付けて IPT に登録する.

(2) trigger PC 候補の登録：図 4(c) に示すように，プリフェッチのフィルが完了するとそのアドレスを IPT から検索し，対応する trigger PC を読み出す．読み出した trigger PC（PC_B）は RRPCQ に書き込まれる．この結果，RRPCQ には，最近フィルが完了したプリフェッチをトリガしたロード命令の PC，すなわちそのロード命令の時点でプリフェッチを発行すれば現在までにフィルの完了が間に合うような PC が記録される．したがって，RRPCQ に記録されている PC は trigger PC のよい候補となる.

(3) target PC と trigger PC の紐付け：図 4(d) に示すように，PC_A のロード命令がキャッシュミスを起こすと，RRPCQ 内の全エントリの trigger PC を読み出す．たとえば我々の評価では 2 エントリの RRPCQ を用いているため，2 つの trigger PC が得られる．読みだした trigger PC をインデクスとして target table の対応するそれぞれのエントリに PC_A を書き込む．これにより，target PC と trigger PC の紐付けが行われる.

以下では，図 5 に示すタイムラインを用いて，上記の動作の例を示す．これらの (1)–(3) の番号は，上記の (1)–(3) の各ステップに対応する.

(1) PC_B のロード命令がプリフェッチを発行すると，その際に PC_B を IPT のエントリに書き込む.

(2) PC_B のロード命令の時に発行したプリフェッチの

Vol.2022-ARC-248 No.17
Vol.2022-SLDM-198 No.17
Vol.2022-EMB-59 No.17
2022/3/10

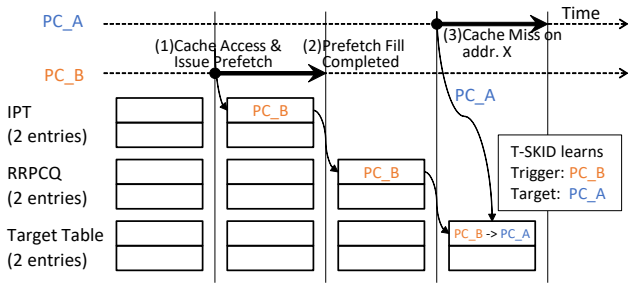

図 5 タイミング学習の時系列.

フィルが完了したため，対応する IPT のエントリから PC_B を読み出し，RRPCQ に PC_B を書き込む.

(3) PC_A のロード命令がアドレス X においてミスを起こす．このときに RRPCQ には PC_B が格納されているためこれを読み出し，target table の PC_B に対応するエントリに PC_A を書き込む.

これにより，PC_B を trigger PC として target table を引くと，target PC である PC_A が得られるようになる．図 1(b) において target PC である PC_A で予測したアドレスのプリフェッチを適切に trigger PC である PC_B によるアクセスの時点まで遅らせる動作は，以上の仕組みにより実現される.

上記の学習の結果，trigger PC と target PC が一致する場合がある．これは，同一 PC によるアクセスが短い期間に複数あり，それらがストライドアクセスを構成していることを意味する．このような場合，上記のメカニズムは既存のストライドプリフェッチャと同様，プリフェッチを直ちに発行することになる.

さらに早くプリフェッチを発行する必要がある場合を検出するため、T-SKID では IPT を活用して degree を動的に学習する．IPT の各エントリはキャッシュアクセスをモニタしており，プリフェッチが発行されてからフィルされるまでの間に各 PC がアクセスしたラインの数をカウンタによって数える． たとえばラインサイズが 64B であり PC_A によるプリフェッチがフィルされるまでにアドレス 64, 128, 192 にアクセスした場合，カウンタは 3 となる．フィルが完了した際にこのカウンタの値を読み出し，address prediction table に degree として保存する．これにより，アクセスの間隔が短い場合に数アクセス先を見越してプリフェッチを発行することができる.

表 1 シミュレーションに用いたパラメータ.

Core parameters	1 or 4 cores, 5.0 GHz, 192-entry ROB, 3 ALUs, 2 Loads, 1 Store
Private L1D cache	32 KiB, 8-way, 4 cycle, 2 line/cycle, 8 MSHRs, LRU
Private L2 cache	256 KiB, 8-way, 8 cycle, 1 line/cycle, 16 MSHRs, LRU
Shared L3 cache (LLC)	2 MiB/core, 16-way, 20 cycle, 1 line/cycle, 32 MSHRs/core, LRU
DRAM	4 GiB 1-channel (single core) or 8 GiB 2-channels (multi-core) 1/24 line/cycle/channel, 48 shared WQs, 48 shared RQs

4. 評価

4.1 方法

我々は T-SKID を ChampSim [14] を用いて評価した．ChampSim は，トレースベースのアウトオブオーダー CPU シミュレータで， メモリシステムを詳細にシミュレートできる．表 1 に，我々が評価で用いたコンフィグレーションを示す．コンフィグレーションはシングルコア実行とマルチコア実行の二つを用いた．また，物理アドレスは仮想アドレスに対して 4 KiB のページ単位でランダムにマッピングされている．これらは，The third Data Prefetching Championship (DPC3) [12] で用いられたコンフィグレーションと同一である.

4.1.1 ワークロード

評価には，SPEC CPU 2017 [15] のトレースをベンチマークとして用いた．プリフェッチによる性能向上を評価するため，プリフェッチしない場合の LLC における MPKI が 1.0 以上の simpoint [16]46 本を選択して用いた．すべての結果は，200M 命令のシミュレーションにおける結果である．ただし、200M 命令のシミュレーション前に 50M 命令のウォームアップを行った．ワークロードにおける instruction per cycle (IPC) speedup 値は，プリフェッチなしの場合に対する IPC の比として算出した.

また，複数コアで LLC やメインメモリが共有されている場合にコア間のインタラクションが性能に与える影響を測定するため，マルチコアのシミュレーションも行った．マルチコアシミュレーションを行うにあたって，46 の mixed-trace ワークロードを作った．各 mixed-trace ワークロードは，シングルコア評価で使ったものからランダムに選択された 4 つのトレースからなる．トレースが偏って選択されるのを避けるため，46 の mixed-trace ワークロード中にすべてのトレースが同じ回数（つまり，4 回）出現するようにした.

マルチコアシミュレーションを行うにあたっては，各コアに一つのトレースを割り振った．まず，最後のコアが 50M 命令の実行を終えるまでウォームアップした．その後，最後のコアが 200M 命令の実行を終えるまでシミュレーションした．各トレースにおいて，最初の 200M 命令のシミュレーションの実行にかかったサイクル数を測定結果とした．mixed-trace ワークロードにおける IPC speedup 値は，各コアの IPC speedup の幾何平均として算出した.

4.1.2 評価対象のプリフェッチャ

T-SKID の性能を評価するにあたり，IPCP ，SPP with PPF ，MLOP ，Bingo を比較対象に用いた．DPC3 において IPCP と SPP with PPF はシングルコア性能において一位・二位のスコアを達成し，MLOP と Bingo はマルチコア性能において一位・二位のスコアを達成している．比

Vol.2022-ARC-248 No.17
Vol.2022-SLDM-198 No.17
Vol.2022-EMB-59 No.17
2022/3/10

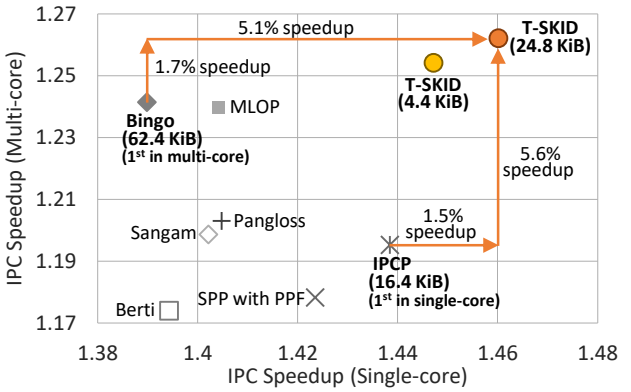

図 6 評価したプリフェッチャのシングルコア性能・マルチコア性能.

較に用いたプリフェッチャのソースコードは DPC3 でアップロードされているものをそのまま使用した.

T-SKID は，合計の容量が 24.75 KiB となるようにしたものを評価対象とした．その内訳は，1024 エントリ・8-way セットアソシアティブの target table に 12.12 KiB，1024 エントリ・8-way セットアソシアティブの address prediction table に 9.50 KiB，16 エントリの IPT，2 エントリの RRPCQ，その他のメタデータに 3.13 KiB である．T-SKID は L1D キャッシュにのみ取り付け，他のキャッシュにはプリフェッチャをつけずに評価した.

4.2 結果

4.2.1 性能向上

図 6 に，DPC3 に出場した prefetchers のシングルコア性能とマルチコア性能を示した．この評価には，DPC3 に出場したほかのプリフェッチャである Sangam，Berti，Pangloss も含まれている．T-SKID は，シングルコア・マルチコアの両方で，既存プリフェッチャよりも高い性能を示した.

プリフェッチャによってはシングルコアのみまたはマルチコアのみで T-SKID に近い性能を達成しているが，もう片方のコンフィグレーションでは低い性能しか達成できていない．マルチコアのコンフィグでは，T-SKID は既存プリフェッチャの中でシングルコア性能が最も高い IPCP に対して 5.6%の性能向上を達成した．T-SKID の容量を 4.4KiB まで削減した場合でも，IPCP に対して 4.9%の性能向上を達成した．シングルコアのコンフィグでは，T-SKID は既存プリフェッチャの中でマルチコア性能が最も高い Bingo に対して 5.1%の性能向上を達成した．これらの結果は，T-SKID が様々な環境に適応できることを示すものである.

4.2.2 シングルコア性能の評価

図 7 は，評価対象プリフェッチャのシングルコアにおける IPC speedup を示したものである．ワークロードは，T-SKID での IPC speedup の高い順に並べた．T-SKID は，

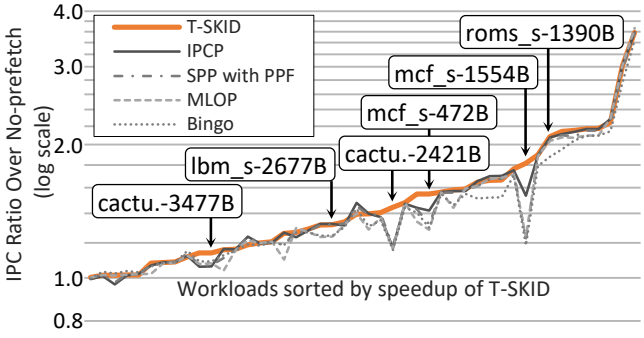

図 7 シングルコアにおける IPC speedup.

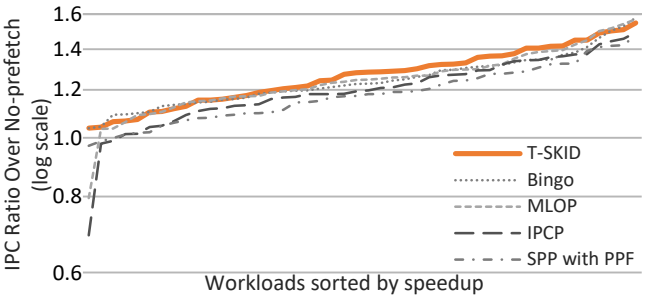

図 8 マルチコアにおける IPC speedup.

ほとんどすべてのトレースにおいて，最も高い性能向上値を達成している．特に，607.cactuBSSN_s や 605.mcf_s において，既存プリフェッチャでは達成できない大きな性能向上値を達成している．これは T-SKID がプリフェッチを適切なタイミングまで遅延させることができることによる．607.cactuBSSN_s-2421B では，T-SKID は IPCP に対して 23.4%の性能向上である 44.4%の性能向上を達成した.

4.2.3 マルチコア性能の評価

図 8 は，評価対象プリフェッチャのマルチコアにおける IPC speedup を示したものである．ワークロードは，プリフェッチャごとに性能向上値の高い順に並べた．図の左端から，IPCP と MLOP は特定のトレースの組み合わせに対して大きく性能が低下することがあることがわかる．また，SPP with PPF でも，no prefetch に対して性能が低下することがあることがわかる．一方，T-SKID では，性能低下することはなく，このワークロードに対して 3.8%性能向上する．また，T-SKID では，図中の中央付近で他のプリフェッチャに対して大きく性能向上していることが見てとれる.

これらの結果は，T-SKID のタイミング学習がロバストであり，多くの環境に適応できることを示している．複数のプログラムが同時に実行されている場合，それら間の相互作用により，単一プログラムを実行する場合とプリフェッチのレイテンシが大きく変化しうる．T-SKID のタイミング学習方式は，そのような場合に広く対応することができている.

4.2.4 プリフェッチのカバレッジと精度

図 9 に各プリフェッチャのカバレッジを，図 10 に各プ

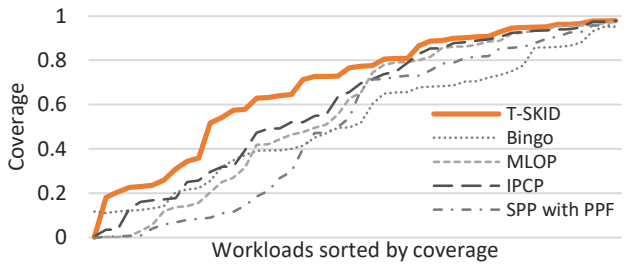

図 9 シングルコアにおける各プリフェッチャのカバレッジ.

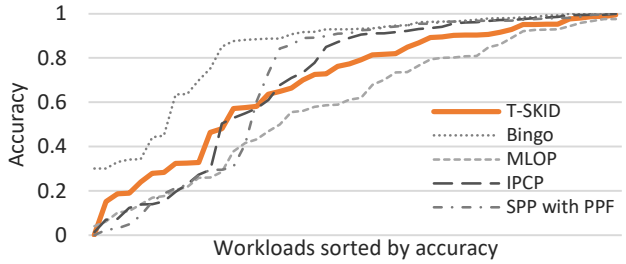

図 10 シングルコアにおける各プリフェッチャの精度.

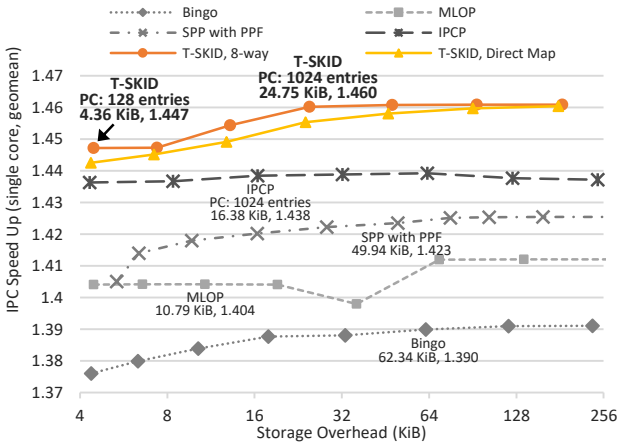

図 11 ストレージ容量センシティビティ．各プリフェッチャについて，他の評価で用いた構成が使用するストレージ容量も示した.

リフェッチャの精度を示す．ワークロードは，プリフェッチャごとに値の高い順に並べた．T-SKID は，精度の大幅な低下なしにカバレッジを大きく向上させている．この高いカバレッジは，タイミング学習によるところが大きい．図 9 の左端は，従来のプリフェッチャではプリフェッチが困難なメモリアクセスパターンについて，T-SKID がカバレッジと精度を同時に向上させたことを示している．

4.2.5 ストレージ容量センシティビティ

図 11 は，評価対象となったプリフェッチャのストレージ容量を変化させたときの，シングルコア IPC speedup の変化を示したものである．T-SKID は，図中のどの容量においても最も高い性能向上を達成した．

IPCP [9] を含む既存の PC ベースのプリフェッチャと T-SKID は，どちらも PC ベースのストライド予測を行うため，ストレージ容量を増やすことで多くのストライドパターンを学習することができるようになる．しかし，既存の PC ベースのプリフェッチャは，覚える PC の数を増やしても性能向上がすぐ飽和する．これは，通常のストライドアクセスを起こす PC の数は，それほど多くないためである．一方，T-SKID はストライド幅が非ゼロのストライドパターンに加え，ゼロストライドパターンも取り扱うことができる．新たなプリフェッチ可能パターンであるゼロストライドパターンの学習が行えるため，T-SKID はストレージ容量を増やすことでさらに性能を伸ばすことができる．

また，8-way セットアソシアティブのテーブルを使った構成に加えて，これらのテーブルをダイレクトマップに変更した構成の評価も行った．ダイレクトマップテーブル構成の T-SKID は，コンフリクトによって性能向上値が若干低下するものの，依然としてどの容量でも最も高い性能向上効果を示した.

5. 関連研究

これまでに様々なプリフェッチャが提案されてきた．それらはプリフェッチを十分早く発行することに着目していて，発行を遅らせることはできない．(1) ストリームプリフェッチャは，典型的には degree と distance と呼ばれる，プリフェッチをどれだけ早く発行するかを決定する 2 つのパラメータを持つ．Degree は一度にどれだけの連続したラインがプリフェッチされるかを表し，distance は現在のラインからどれだけ遠くのラインをプリフェッチするかを表す．これらのパラメータは，IPCP [9] のように静的に決められている [1], [2], [17] こともあれば，無駄なプリフェッチ発行を避けるために動的に変更される場合もある [18], [19], [20]．(2)VLDP [5] と SPP [6]，そして IPCP [9] の IP complex stride prefetcher はアクセスのあったアドレスの差 (delta) に着目する．これらは再帰的に delta の系列を予測することで数アクセス先までをプリフェッチし，デマンドアクセスに間に合うプリフェッチ発行の可能性を向上している．(3)BOP [11] は遅すぎるプリフェッチ発行を防ぐために一つの最良のオフセットを学習する．MLOP [10] はアクセスの順序を考慮に入れて複数のオフセットを決定し，BOP より多くの価値あるプリフェッチを発行することを試みている.

6. おわりに

我々は，既存のプリフェッチャはしばしば早すぎるプリフェッチを発行しており，ここに性能を向上させる新たな機会があることを見出した．早すぎるプリフェッチの問題を解決するために，プリフェッチアドレスとタイミングを別々に予測し，適切なタイミングまでプリフェッチアクセスを発行することを遅らせる T-SKID を提案した，SPEC CPU 2017 を用いたシミュレーションを通して T-SKID を

評価した結果，T-SKID はプリフェッチをしないプロセッサと比較してシングルコアにおいて 46.0%性能向上を，マルチコアにおいて 26.2%性能向上を達成した．

謝辞 本研究の一部は JSPS 科研費 JP19H04077, JP20H04153, JP20J22752, JP21J11687 による．

参考文献

[1] Jouppi, N. P.: Improving Direct-mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers, *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, pp. 364–373 (1990).

[2] Palacharla, S. and Kessler, R. E.: Evaluating Stream Buffers As a Secondary Cache Replacement, *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, pp. 24–33 (1994).

[3] Baer, J. and Chen, T.: An effective on-chip preloading scheme to reduce data access penalty, *ACM/IEEE Int. Conf. on Supercomputing (SC)*, pp. 176–186 (1991).

[4] Fu, J. W. C., Patel, J. H. and Janssens, B. L.: Stride Directed Prefetching in Scalar Processors, *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pp. 102–110 (1992).

[5] Shevgoor, M., Koladiya, S., Balasubramonian, R., Wilkerson, C., Pugsley, S. H. and Chishti, Z.: Efficiently Prefetching Complex Address Patterns, *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pp. 141–152 (2015).

[6] Kim, J., Pugsley, S. H., Gratz, P. V., Reddy, A., Wilkerson, C. and Chishti, Z.: Path Confidence Based Lookahead Prefetching, *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pp. 1–12 (2016).

[7] Somogyi, S., Wenisch, T. F., Ailamaki, A., Falsafi, B. and Moshovos, A.: Spatial Memory Streaming, *ACM/IEEE Int. Symp. on Computer Architecture (ISCA)*, pp. 252–263 (2006).

[8] Bakhshalipour, M., Shakerinava, M., Lotfi-Kamran, P. and Sarbazi-Azad, H.: Accurately and Maximally Prefetching Spatial Data Access Patterns with Bingo, *The 3rd Data Prefetching Championship (DPC3)* (2019).

[9] Pakalapati, S. and Panda, B.: Bouquet of Instruction Pointers: Instruction Pointer Classifier based Hardware Prefetching, *The 3rd Data Prefetching Championship (DPC3)* (2019).

[10] Shakerinava, M., Bakhshalipour, M., Lotfi-Kamran, P. and Sarbazi-Azad, H.: Multi-Lookahead Offset Prefetching, *The 3rd Data Prefetching Championship (DPC3)* (2019).

[11] Michaud, P.: Best-offset Hardware Prefetching, *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, pp. 469–480 (2016).

[12] The 3rd Data Prefetching Championship, `https://dpc3.compas.cs.stonybrook.edu/`.

[13] Bakhshalipour, M., Shakerinava, M., Lotfi-Kamran, P. and Sarbazi-Azad, H.: Bingo Spatial Data Prefetcher, *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, pp. 399–411 (2019).

[14] ChampSim, `https://github.com/ChampSim/ChampSim/`.

[15] Standard performance evaluation corporation CPU2017 benchmark suite, `http://www.spec.org/cpu2017/`.

[16] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B.: Automatically characterizing large scale program behavior, *ACM SIGPLAN Notices*, Vol. 37, No. 10.

[17] Tendler, J. M., Dodson, J. S., Fields, J. S., Le, H. and Sinharoy, B.: POWER4 system microarchitecture, *IBM Journal of Research and Development*, Vol. 46, No. 1, pp. 5–25 (2002).

[18] Srinath, S., Mutlu, O., Kim, H. and Patt, Y. N.: Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers, *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, pp. 63–74 (2007).

[19] Ebrahimi, E., Mutlu, O. and Patt, Y. N.: Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems, *IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, pp. 7–17 (2009).

[20] Ebrahimi, E., Mutlu, O., Lee, C. J. and Patt, Y. N.: Coordinated Control of Multiple Prefetchers in Multi-core Systems, *IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, pp. 316–326 (2009).