Vol.2019-ARC-237 No.11
2019/7/24

階層分離型キャッシュシミュレーションを用いた高速アドレストレーサの実装と評価

内匠真也[†1]　城田祐介[†2]　白井智[†2]　吉村礎[†2]　金井達徳[†2]

概要：大規模データ処理において高速不揮発メモリであるストレージクラスメモリ（SCM）に対する期待が高まっている．高速な SCM と DRAM を組み合わせた主記憶や低速な SCM と NAND を組み合わせたストレージなど異種メモリ混載により複雑化するメモリ階層においては，アプリケーションのメモリアクセスパターンに応じたメモリ階層制御が重要になる．メモリ階層制御方式の設計には主記憶／ストレージ層の正確なアドレストレース情報が必要になり，それを高速に生成するための新たなキャッシュシミュレーション手法が求められる．そこで本研究では，多段キャッシュシミュレーションを，正確性に影響が出ない範囲で高速化を重視する上位層と，正確性を重視する下位層に分離して実行することで，シミュレーション全体を高速化する階層分離型のキャッシュシミュレーション方式を提案する．本稿では，L1 キャッシュと L2 キャッシュ以下で分離する階層分離型キャッシュシミュレーションを用いたアドレストレーサを実装し，評価を行った．

キーワード：ストレージクラスメモリ(SCM)，メモリ階層制御，キャッシュシミュレーション，アドレストレーサ

1. はじめに

メモリ階層における DRAM と SSD/HDD の間のアクセスレイテンシの大きなギャップを埋める，ストレージクラスメモリ（SCM: Storage-class Memory） [1, 2, 3]と呼ばれるバイトアドレッサブルな新型の高速不揮発メモリの実用化が期待されている．SCM は，待機電力が低くかつ DRAM を凌ぐ高集積化が可能であるため，高速／低消費電力でスケーラブルな主記憶を実現できる可能性がある．しかし，SCM は DRAM に比べてアクセスレイテンシが高く, DRAM を SCM に単純に置き換えると処理性能の低下を招く可能性がある. よって, 主記憶の高性能化を実現するためには，アプリケーションのメモリアクセス特性や SCM の特性に応じて DRAM と SCM を適応的に使い分ける階層制御により，両者の良さを引き出し効率良いデータ処理を行う必要がある [4]．例えば，文献 [5]では，SCM/DRAM 混載主記憶において，機械学習を利用して，SCM へのダイレクトアクセスと DRAM をキャッシュとするページングを使い分けるハイブリッドアクセス方式のメモリ階層制御による性能向上の可能性が示されている．

また，一般的に，SCM は書換え可能回数に制約があるため，書換え頻度を平均化するウェアレベリング制御方式が必要となる．さらに，SCM は DRAM に比べてアクセスレイテンシが高く，アクセス時の動的電力が高い特性を持つため，高レイテンシを隠蔽する電力効率の良いプリフェッチ制御方式も重要になる [6].

メモリ階層制御方式の設計に関係する重要な要素技術として，プロセッサごとにメモリバスを分離することで，キャッシュアクセスのストール時間を削減する NUMA（Non-Uniform Memory Access）がある．NUMA を活用し，アプリケーション特性に応じてプロセッサの使用するメモリバスの設定を最適化することで，アプリケーションの高速化を実現できることが知られている [7]．SCM の登場によりプロセッサとメモリの関係も変化する可能性があり，SCM に合わせた最適な NUMA 方式の設計も検討する必要がある．

このように複雑化する異種メモリ混載主記憶においてアーキテクチャ探索を行うためには，アプリケーションの実行トレースから主記憶への参照先を時系列に記録したアドレストレースを求める必要がある．さらに，アドレストレースを正確に取得するには，CPU の多段キャッシュをシミュレーションし，主記憶のメモリアクセス情報を求める必要がある．このとき，アドレストレース情報に与える影響の大きい下位層のキャッシュメモリを高精度にシミュレーションすることが重要となる．

キャッシュメモリのシミュレーション手法として，実行トレース中にキャッシュシミュレーションを行う実行ドリブン方式と実行トレースが収集したメモリアクセス命令のトレースファイルを入力とするトレースドリブン方式がある [8, 9].

実行ドリブン方式は毎回オーバヘッドの大きい実行トレースを行う必要がある．よって，キャッシュ構成を変更するたびに全階層のキャッシュシミュレーションを行うため，変更のない上位層までシミュレーションが必要になりアドレストレース収集の効率が悪い．

一方で，トレースドリブン方式はトレースファイルの利用によりキャッシュ構成変更時の実行トレースを省略し，高速化が可能であるものの，実行ドリブン方式と同様に変更のない上位層もシミュレーションする必要がある．さらに, SCM を対象とする大規模データのインメモリ処理を対象にした場合には全てのメモリアクセスを記録したトレースファイルのサイズが課題となる．

†1 東芝インフラシステムズ(株) インフラシステム技術開発センター
†2 (株) 東芝 研究開発センター

本稿では，両者を組み合わせたキャッシュシミュレーション方式を提案する．具体的には，主記憶アドレストレース情報の正確性に影響が少なくかつ構成の変更が少ないキャッシュメモリの上位層のみシミュレーションしつつアプリケーションの実行トレースを行うオンラインキャッシュシミュレータと，オンラインキャッシュシミュレータが生成した中間アドレストレースを元に下位層を正確にシミュレーションするオフラインキャッシュシミュレータの協調により，効率的なアドレストレース取得を可能にする階層分離型キャッシュシミュレーション方式を提案する．オンラインキャッシュシミュレータは上位キャッシュをシミュレーションすることにより局所的なメモリアクセスを削減し，中間アドレストレースのサイズ抑制を実現する．オフラインキャッシュシミュレータでは，アーキテクチャ探索の主ターゲットである下位層のキャッシュの設定を変更した場合に，中間アドレストレースを基にした高速なキャッシュシミュレーションを実現する．

本稿では階層分離型キャッシュシミュレーションを用いたアドレストレーサを実装し，評価を行う．以下，2 章で提案手法の詳細について述べ，3 章で提案手法の評価結果について述べる．4 章で関連研究について述べ，5 章で結論を述べる．

2. 階層分離型キャッシュシミュレーション

2.1 概要

従来のキャッシュシミュレーションはキャッシュメモリ構成の設計や性能評価が主な目的だった．しかし，ビッグデータ処理やディープラーニング(AI)などの大規模データのインメモリ処理が要求されるアプリケーションやSCM の登場により，アーキテクチャ探索のフォーカスが主記憶／ストレージ層に移っているため，目的も変化している．主記憶をターゲットとするアドレストレーサでは，主記憶におけるメモリ階層制御方式の検討に必要な下位層のメモリアドレス情報を高精度に取得することが重要となるため，必ずしも上位層のキャッシュの挙動自体を正確にシミュレーションする必要はない．

そこで本研究では，目的に応じて多段キャッシュシミュレーションを柔軟に分離し，正確性に影響が出ない範囲で大胆な高速化を行う上位層と，メモリ階層制御方式の検討に必要な正確性を重視する下位層に分離して実行することで，シミュレーション全体を高速化する階層分離型のキャッシュシミュレーション方式を提案する．

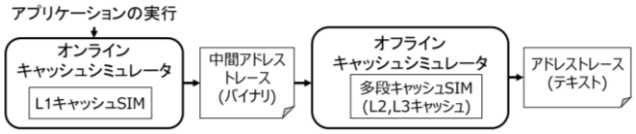

図 1　提案手法の構成

2.2 階層分離型キャッシュシミュレーションを用いたアドレストレーサ

2.2.1 全体構成

図 1 に，本稿で実装する階層分離型キャッシュシミュレーションを用いたアドレストレーサの全体構成を示す．本稿では，キャッシュシミュレーションを L1 キャッシュと L2 キャッシュ以下で分離する．これは，プライベートキャッシュである L1 キャッシュのキャッシュコヒーレンシが主記憶のアドレストレースの正確性に与える影響が少ないと考え，キャッシュコヒーレンシのシミュレーションを省略し，高速化を図るためである．また，L2 以下のキャッシュではキャッシュコヒーレンシをシミュレーションするため, L1 でシミュレーションを省略したとしても正確性は維持できると考えられる．よって，全体としてみた場合，高速かつ正確なアドレストレースの実現が可能となる．

オンラインキャッシュシミュレータは L1 キャッシュを対象とした実行ドリブン方式のキャッシュシミュレーションを行い, L1 キャッシュから下層メモリへのメモリアクセス情報を中間アドレストレースに出力する．オフラインキャッシュシミュレータは L2 キャッシュ以下を対象にして，中間アドレストレースを基にしたトレースドリブン方式のシミュレーションを行い，主記憶へのアクセスを記録したアドレストレースを出力する．

オンラインキャッシュシミュレータは L1 キャッシュをシミュレーションすることで，局所的なアクセスの記録による中間アドレストレースのサイズ増加を抑制する. L2 キャッシュまでオンラインシミュレーションに含めると同サイズをさらに削減可能だが, L1 キャッシュまででも大幅に削減可能であり, L2 キャッシュシミュレーションによる速度低下も考慮して L1 キャッシュと L2 キャッシュ以下で分離した．オフラインキャッシュシミュレータは中間アドレストレースに基づいたトレースドリブン方式のキャッシュシミュレーションにより, L2 キャッシュ以下の高速なキャッシュシミュレーションを実現する．よって，構成変更の要求が高い L2 キャッシュ以下の下位層の構成を変更した場合でも高速なキャッシュシミュレーションが可能となる．さらに, 下位層はアドレストレースへの影響が大きいため，オフラインキャッシュシミュレータは正確性を重視してキャッシュコヒーレンシをシミュレーションする．一方，オフラインキャッシュシミュレータはトレースドリブン方式により高速に動作可能なため，正確性を重視することによ

る速度低下は少ない．

2.2.2 オンラインキャッシュシミュレータ

オンラインキャッシュシミュレータは Intel®*1 CPU の動的バイナリ計装ツール Intel® Pin [10]を用いて実行トレースを行い，対象アプリケーションのメモリアクセスを解析する．オンラインキャッシュシミュレータでは物理コアごとにキャッシュコヒーレンシのシミュレーションを省略した L1 キャッシュを用意し，L1 キャッシュ以下へのアクセスを中間アドレストレースに記録する．L1 キャッシュには，一般的に利用されている，セットアソシアティブ方式，ライトバック方式，Virtual Index Virtual Tag 方式を採用する．

オンラインキャッシュシミュレータではアドレストレースに影響の少ないキャッシュコヒーレンシをシミュレーションせずに高速化する．一方で，オフラインキャッシュシミュレータではキャッシュコヒーレンシを考慮したシミュレーションが行われるため，アドレストレースの正確性低下は最小限となる．

SCM 搭載主記憶において重要性がより高まるプリフェッチ機能については，オンラインキャッシュシミュレータにおいてオプションで同機能を有効にできる．オンラインキャッシュシミュレータでは，同一キャッシュラインに連続的にアクセスした場合，次のキャッシュラインを L1 キャッシュに事前に取得する Data Cache Unit (DCU) Prefetcher を文献 [11]を参考に実装する．

キャッシュの合計サイズ，キャッシュラインサイズ，アソシアティブ(連想数)などのキャッシュ構成情報や仮想 CPU と物理 CPU の対応情報は，本アドレストレーサを動作させるシステムから値を取得する．キャッシュの構成情報についてはオプションにより変更できる．

中間アドレストレースにはメモリアクセスごとに以下の情報が記録される．

時間(クロック)

メモリアクセスの発生したときの CPU クロックを記録する．CPU クロックは rdtsc 命令により取得する．

アドレス

メモリアクセスの発生した物理アドレスを記録する．L1 キャッシュにアドレスを記録するとき，もしくはキャッシュミスが発生したときに仮想アドレスから物理アドレスの変換を行う．物理アドレスを取得するため，メモリ空間が異なるアプリケーションでもアドレストレースが取得可能になる．

命令カウント

計測を開始してからの仮想 CPU ごとの命令カウントを記録する．命令カウントは共有メモリ上に保存しており，fork したプロセス間でも命令カウントを計測できる．

物理 CPU

メモリアクセスを発生させた物理 CPU 番号を記録する．

仮想 CPU

メモリアクセスを発生させた仮想 CPU 番号を記録する．

読出し(R)・書込み(W)

書込みの有無を記録する．

アクセス方法

メモリへのアクセスの方法を記録する．CPU 命令による明示的なメモリアクセスか，キャッシュからの追出しによるメモリアクセスか，プリフェッチによるメモリアクセスかを識別する情報を記録する．

2.2.3 オフラインキャッシュシミュレータ

オフラインキャッシュシミュレータは中間アドレストレースに基づいて，L2 以下のキャッシュをオフラインでシミュレーションし，主記憶へのアクセスを記録したアドレストレースを出力する．

オフラインキャッシュシミュレータは実行トレースを伴わないため，高速なシミュレーションが可能である．高速なシミュレーションが可能なため，オフラインキャッシュシミュレータでは正確性を重視し，キャッシュコヒーレンシや高度なプリフェッチもシミュレーション対象とする．キャッシュコヒーレンシプロトコルには一般的に利用されている MESI プロトコル，MOESI プロトコルを実装する．プリフェッチについては文献 [11]を参考に，隣接するキャッシュラインのペアを取得する Spatial Prefetcher と一定ストライドでのキャッシュラインへのアクセスを検知した場合に同一ストライド分先のキャッシュラインを取得する Streamer を実装する．

図 2 に示すコンフィグレーションファイルでシミュレーションする L2 以下のキャッシュ構成を設定できる．

1. キャッシュラインサイズ
   キャッシュラインのサイズを設定する．基本的にはオンラインキャッシュシミュレータでのキャッシュラインサイズと同じ値が設定されることを想定している．
2. キャッシュサイズ
   キャッシュサイズをバイト単位で設定する．
3. アソシアティブ
   同セットのキャッシュラインの保持数を設定する．
4. プリフェッチ機能
   有効にするプリフェッチ機能を選択する．Spatial Prefetcher と Streamer の有効をそれぞれ設定できる．
5. Inclusive or Non-Inclusive
   上位キャッシュが要求したデータを保持する場合 (Inclusive) は 1 を，上位キャッシュが要求したデータを保持しない場合 (Non-Inclusive) は 0 を指定する．

*1 インテル, Intel, Xeon は，アメリカ合衆国及びその他の国における Intel Corporation 又はその子会社の商標又は登録商標です．

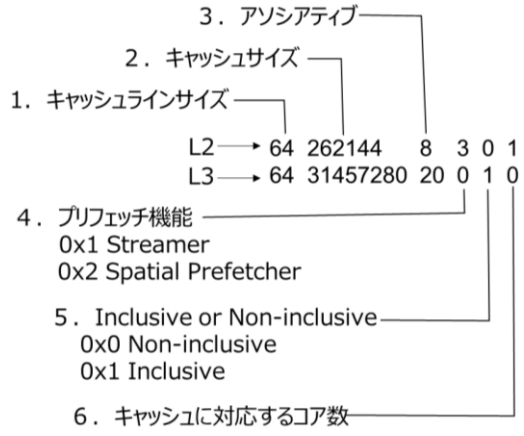

図 2　オフラインキャッシュシミュレータのキャッシュコンフィグレーションファイルの例

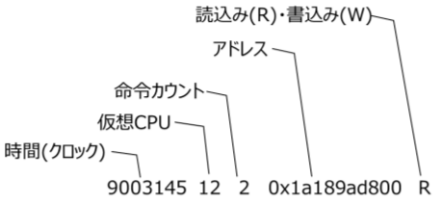

図 3　主記憶へのアクセスのアドレストレースの例

6.　キャッシュに対応するコア数

複数のソケットを持つ CPU の場合, ソケットごとにキャッシュが存在する．この項目では一つのキャッシャを共有する物理コアの数を設定する．例えば，4 コアの CPU が 2 つ存在する環境を想定する場合，この項目を 4 と設定することで，項目 1—5 で設定したキャッシュを 2 個作成し，4 コアが一つのキャッシュを共有する．0 を指定した場合はキャッシュを 1 つ作成し，全コアで一つのキャッシュを共有する．

図 3 に，アドレストレースに記録される各メモリアクセスの例を示す. アドレストレースの各メモリアクセスには，実時間のクロック，主記憶へのアクセスが発生した仮想 CPU, 各仮想 CPU で計測された命令カウント数, アドレス，読出し・書込みかの情報が記録される．仮想 CPU と命令カウント，または時間を見ることで，時系列に主記憶へのアクセスを観測できる．また，例えば，主記憶へのアクセス先アドレスを見ることで，アクセスが集中している箇所を観測できる．

図 3 の各出力項目はオフラインキャッシュシミュレータのオプションにより，出力の有無を選択できる．用途により，出力項目を選択することで，アドレストレースのサイズ抑制や高速化が可能となる．

表 1　実行評価環境

HW	Xeon E7-8870 v3 1.20 GHz, Hyper-Threading off, DDR4-2333,
SW	Ubuntu 16.04.2 LTS, Linux 4.4.0-141, gcc 5.4.0, Intel Pin 3.2

表 2　キャッシュの設定

L1D	32KB, 8-way, LRU, Write-allocate, DCU prefetcher 有効
L2	256KB, 8-way, LRU, Adjacent Prefetcher・Streamer 有効, non-inclusive, MESI coherence protocol,
L3	45MB, 20-way, LRU, inclusive

3. 評価

提案手法のオーバヘッドとなる実行時間と生成ファイルのサイズと,取得するアドレストレースの正確性を評価する．アドレストレースの正確性については，Intel® Performance Monitoring Unit [12] を用いてハードウェア上で計測した正確なメモリアクセスの回数と，提案手法により求めるメモリアクセスの回数を比較し，評価する．

ベンチマークプログラムには，マルチプロセッサベンチマーク集 PARSEC [13]を利用する．今回の評価にはメモリアクセスのローカリティが低い canneal, ローカリティの高い facesim，さらにローカリティの高い raytrace の 3 つのベンチマークプログラムを利用し，いずれもシングルコアで動作させる．

表 1 に評価時の実行トレース取得環境を示し，表 2 にキャッシュシミュレーションの設定を示す.

3.1 実行時間と生成ファイルのサイズの評価

オンラインキャッシュシミュレータの実行時間と生成ファイルサイズの結果を表 3 に示し，オフラインキャッシュシミュレータの実行時間と生成ファイルサイズの結果を表 4 に示す.

表 3 のデータアクセス回数はメモリアクセスを要求する命令の発行回数を示しており，表 3 のファイル削減率は 1-(ファイル記録データアクセス回数/データアクセス回数)を示している．トレースドリブン方式のキャッシュシミュレーションを利用する場合，その入力となるトレースファイルにはデータアクセスを全て記録する必要がある. 一方で, 提案手法は実行トレース中に L1 キャッシュをシミュレーションすることで，オフラインキャッシュシミュレータのトレースファイルとなる中間アドレストレースに記録されるメモリアクセスを 9 割以上削減している．また，メモリアクセスのローカリティが大きいほど, L1 キャッシュへのヒット率が高いため，ファイル削減率の効果が高い.

表 3　提案手法のオンラインキャッシュシミュレータのオーバヘッド評価結果

アプリケーション	実行時間(s)	slowdown	MIPS	ファイルサイズ	ファイル記録データアクセス回数	データアクセス回数	アドレストレース削減率
canneal	273.2	33.4	26.9	7.6GB	254M 回	2.59G 回	0.9
facesim	799.6	94.6	35.6	17GB	545M 回	12G 回	0.95
raytrace	896.6	105	41.4	2.2GB	73.6M 回	15.9G 回	0.995

表 4　提案手法のオフラインキャッシュシミュレータのオーバヘッド評価結果

アプリケーション	実行時間(s)	MIPS	MIPS (+オンラインキャッシュシミュレータ)	ファイルサイズ
canneal	135	544	18.0	1.5GB
facesim	199	1432	28.5	852MB
raytrace	36.5	10171	39.8	309MB

オンラインキャッシュシミュレータはデータアクセスに対してキャッシュシミュレーション処理を行う．そのため，表 3 に示されたデータアクセス回数が多いアプリケーションほど，オンラインキャッシュシミュレータのオーバヘッドによる速度低下(slowdown)が大きくなっている．さらに，表 4 より，オフラインキャッシュシミュレータの速度性能を示す MIPS (Million Instructions Per Second)値は中間トレースファイルのサイズに比例しており，ファイルサイズが小さいほど高速なシミュレーションを実現している．

表 4 より，提案手法のオンラインキャッシュシミュレータとオフラインキャッシュシミュレータと合わせて，一回のアドレストレースは 18.0～39.8 MIPS で動作していることが分かる．Intel Pin を用いた実行ドリブン方式のキャッシュシミュレータには CMPSim [14]や Exana [15], ZSim [16]がある．CMPsim の速度性能が 8-12 MIPS，Exana が 21.4 MIPS であり，提案手法と同程度の速度で動作する．また，提案手法はファイルサイズの大きいアドレストレースを出力する処理も含まれているため，キャッシュシミュレーションに限定すれば，さらに高速に動作する．

一方で，ZSim [16]については並列処理シミュレーションを用いた高速化により，6 並列の場合には平均 40.2 MIPS で動作可能であり，提案手法よりも高速に動作する．しかし，提案手法のオフラインキャッシュシミュレータに限れば 544～10171 MIPS と ZSim よりも高速に動作する．よって，L2 以下のキャッシュ構成を変更するだけの場合，提案手法は高速なオフラインキャッシュシミュレータのみの実行で良いため，アーキテクチャ探索に伴うキャッシュの変更が多発する環境では提案手法の方が高速なキャッシュシミュレーションを実現できる．また，ZSim と同様に並列処理を導入することにより，オンラインキャッシュシミュレータ及びオフラインキャッシュシミュレータの更なる高速化が可能である．

3.2 正確性評価

提案手法のアドレストレースに記録されたメモリアクセス回数とハードウェア上で計測した正確なメモリアクセス回数を，命令数に基づいて時系列に比較した結果と，メモリアクセス回数の総合計を比較した結果を，facesim, canneal, raytrace のそれぞれについて図 4, 図 5, 図 6 に示す．

図 4～図 6 より，計測対象のアプリケーションに依らず，ハードウェアでの計測と提案手法は同様のメモリアクセスの増減が記録されていることが分かる．メモリアクセスの総合計も最小で 1%，最大でも 10%以内に収まっており，大きな誤差も生じていない．

4. 関連研究

メモリアーキテクチャの性能解析を目的とした関連研究について紹介する．

Gem5 [17]，MARSSx86 [18]はキャッシュを含む CPU アーキテクチャ全体の動作を再現するフルシステムシミュレータである．OS の動作を含めた全体の動作と命令サイクルごとの詳細なシミュレーションを可能とする．一方で，低速なシミュレーションが課題となっており，シミュレーション速度は 1 MIPS 以下となる．

Intel CPU のハードウェアイベント収集機能 PEBS (Precise Event Based Sampling) [12]を用いたアドレストレース計測ツールが提案されている [19]．PEBS にはキャッシュミスが発生したアドレスを記録する機能があり，これを利用することにより，アドレストレースを高速に収集できる．

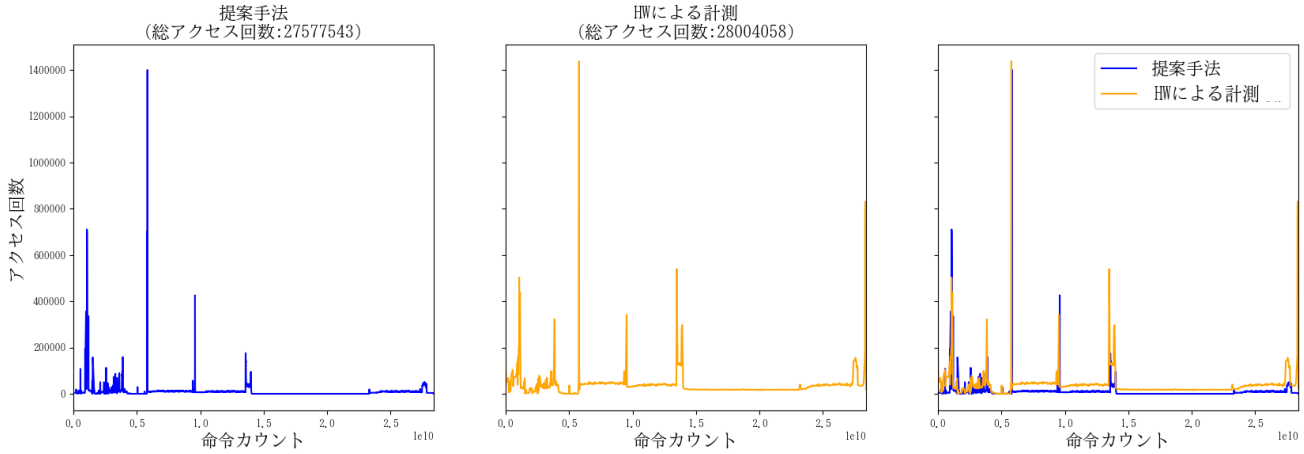

図 4　メモリアクセス回数比較（facesim）

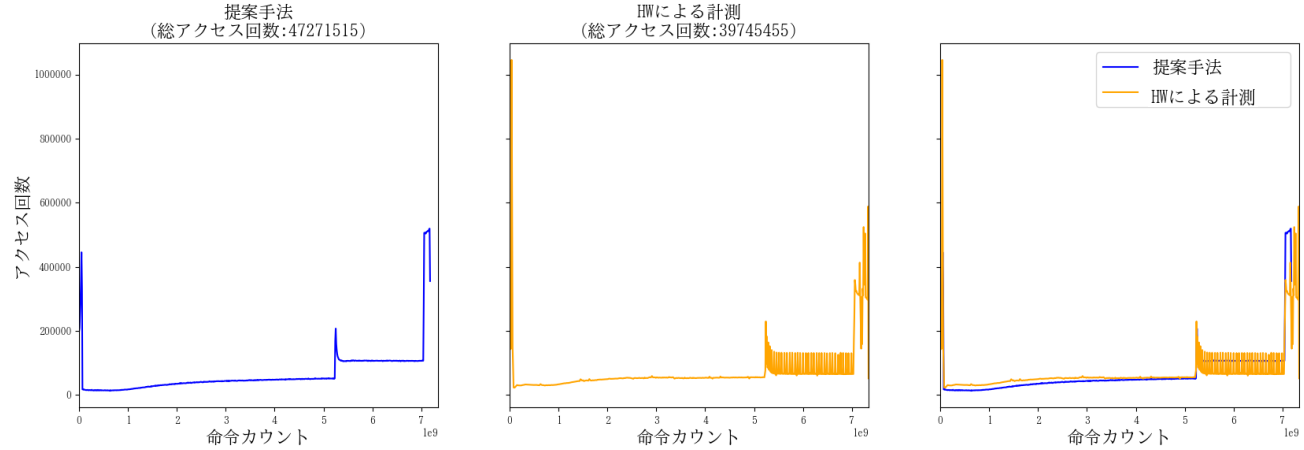

図 5　メモリアクセス回数比較（canneal）

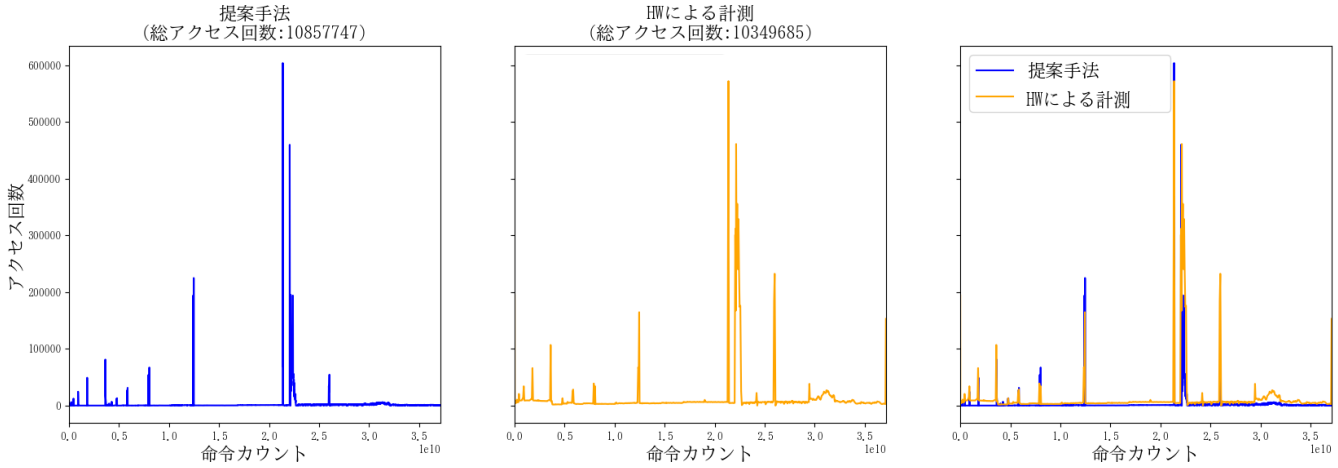

図 6　メモリアクセス回数比較（raytrace）

一方で，PEBS ではプリフェッチによるメモリアクセスやライトバックキャッシュで追い出されて書き戻されるデータのメモリアクセスを収集できない．

CacheSim [14]，Exana [15]，Zsim [16]は Intel Pin ベースの性能解析ツールであり，キャッシュストールにかかったサイクル数やキャッシュミスの多い箇所など，ボトルネック情報を出力できる．一方で，時系列のアドレストレースを出力しない．提案手法と上記ツールの出力情報を組み合わせることで，より詳細なメモリアーキテクチャの解析ができる可能性がある．

Dinero IV [20]はトレースドリブン方式のキャッシュシミュレータである．提案手法の中間アドレストレースや主記

憶のアドレストレースを入力として Dinero IV を利用できる可能性があり，Dinero IV と提案手法を組み合わせて利用できる可能性がある．

5. おわりに

本稿では，高速かつ正確なアドレストレースの収集を目指した階層分離型キャッシュシミュレータについて述べた．評価の結果，提案手法は 18.0～39.8 MIPS でアプリケーションの実行トレースが可能であり，他の Intel Pin ベースのキャッシュシミュレータと同等の速度で動作することを確認した．さらに，頻繁に構成が変更される可能性の高い L2 以下の下層のキャッシュ構成を変更したアドレストレースの取直しは 544～10171 MIPS で動作可能であり，既存のキャッシュシミュレータより高速にアドレストレースが取得可能なことを確認した．また，正確性については，提案手法とハードウェアで計測したメモリアクセス回数を時系列に比較し，メモリアクセス回数の増減が一致し，かつ，総メモリアクセスの差が最大 10%以内に抑えられることを確認した．

今後は，Performance Monitoring Unit などのキャッシュ情報を正確かつ高速に収集できるハードウェア機能との連携や，並列化による更なるアドレストレース収集の高速化手法について検討する．

参照文献

[1] G. Sun, J. Zhao, M. Poremba, C. Xu , Y. Xie, “Memory that Never Forgets: Emerging Non-volatile Memory and the Implication for Architecture Design,” National Science Review, vol. 5, no. 4, pp.577-592, 2017.

[2] R. F. Freitas , W. W. Wilcke, “Storage-class Memory: The Next Storage System Technology,” IBM Journal of Research and Development, vol. 52, no. 4, pp. 439-447, 2008.

[3] K. Hoya, K. Hatsuda, K. Tsuchida, Y. Watanabe, Y. Shirota , T. Kanai, “A Perspective on NVRAM Technology for Future Computing System,” International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA), 2019.

[4] Y. Shirota, S. Yoshimura, S. Shirai , T. Kanai, “Powering-off DRAM with Aggressive Page-out to Storage-class Memory in Low Power Virtual Memory System,” In Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips XIX), pp. 1--3, 2016.

[5] Y. Shirota, S. Shirai , T. Kanai, “Hybrid Access in Storage-class Memory-aware Low Power Virtual Memory System,” In Proceedings of 2019 IEEE Symposium on Low-Power and High-Speed Chips (COOL Chips), pp. 1--3, 2019.

[6] 肥塚真由子, 城田祐介, 白井智 , 金井達徳, “機械学習を用いた SCM 主記憶向けプリフェッチ制御方式,” 研究報告ハイパフォーマンスコンピューティング(HPC), vol. 2018-HPC-166, no. 15, pp. 1-7, 2008.

[7] C. McCurdy , J. S. Vetter, “Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms,” In Proceedings of International Symposium on Performance Analysis of Systems and Software(ISPASS), 2010.

[8] A. R. Hassan, N. Harris , A. Topham, “Efthymiou, Synthetic Trace-Driven Simulation of Cache Memory,” In Proceedings of International Conference on Advanced Information Networking and Applications Workshops(AINAW), pp. 764-771, 2007.

[9] R. A. Uhlig , T. Mudge, “Trace-driven memory simulation: a survey,” ACM Computing Surveys, vol 29, issue 2, pp. 128-170, 1997.

[10] C. K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi , K. Hazelwood, “Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation,” In Proceedings of Programming Language Design and Implementation(PLDI), 2005.

[11] Intel Corporation, “Intel® 64 and IA-32 Architectures Optimization Reference Manual,” 2019.

[12] Intel Corporation, “Intel® 64 and IA-32 Architectures Software Developer’s Manual,” 2019.

[13] C. Bienia, S. Kumar, J. P. Singh , K. Li, “The PARSEC benchmark suite: Characterization and architectural implications.,” In Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT),, 2008.

[14] A. Jaleel, R. Cohn, C.-K. Luk , B. Jacob, “CMP$im: A Pinbased on-the-fly multi-core cache simulator.,” In Proceedings of Workshop on Modeling, Benchmarking and Simulation(MOBS), 2008.

[15] Y. Sato, S. Sato , T. Endo, “Exana: an execution-driven application analysis tool for assisting productive performance tuning,” In Proceedings of Workshop on Software Engineering for Parallel Systems(SEPS), 2015.

[16] D. Sanchez , C. Kozyrakis, “ZSim: Fast and accurate microarchitectural simulation of thousand-core systems.,” In Proceedings of International Symposium on Computer Architecture (ISCA), pp.175--486, 2013.

[17] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill , D. A. Wood, “The GEM5 Simulator,” SIGARCH Computer Architecture News, vol. 39, no. 2, 2011.

[18] A. Patel, F. Afram , K. Ghose, “Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors.,” In Proceedings of International QEMU Users’ Forum, pp. 29--30, 2011.

[19] H. Servat, G. Llort, J. Gonzalez, J. Gimenez , J. Labarta, “Low-Overhead Detection of Memory Access Patterns and Their Time Evolution,” In Proceedings of International Conference on Parallel and Distributed Computing (Euro-Par). pp. 57--69, 2015.

[20] J. Edler , M. D. Hill, “Dinero IV trace-driven uniprocessor cache simulator,” 2019.