Vol.2018-HPC-166 No.15
2018/9/28

機械学習を用いたSCM主記憶向けプリフェッチ制御方式

肥塚真由子† 城田祐介† 白井智† 金井達徳†

概要：HPCシステムでは大規模データ処理において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．SCMはDRAMを凌ぐ高集積化が可能であり待機電力が低い一方で，DRAMよりは低速でアクセス時の動的電力が高い特性を持つ．そのため，アプリケーションのメモリアクセス特性に応じて，DRAMとSCMを適応的に使い分ける階層制御やSCM主記憶向けの電力効率の良いプリフェッチ制御を行い，SCMの高速性と省電力性を引き出す必要があり，メモリ制御が複雑化するため最適なメモリ制御方式を決定することは難しい．本研究では，システムレベルの時系列パフォーマンスデータをもとに機械学習を用いて最適なメモリ制御を予測可能にするフレームワークを提案する．本稿では，提案フレームワークをプリフェッチ制御に適用し，電力効率の良いプリフェッチ制御の実現可能性を評価した．その結果，コンピュータシステムドメイン知識を反映した予測モデルが自動生成され，高速で正確なリアルタイム制御ができる可能性があることを確認した．

キーワード：ストレージクラスメモリ，プリフェッチ，機械学習，決定木，自動チューニング

1. はじめに

近年，HPCシステムではビッグデータ処理やディープラーニング（AI）など新しい応用技術の登場で大規模データのインメモリデータ処理の需要がますます高まってきている．従来，インメモリデータ処理では高速な主記憶上でデータを処理できるように大容量のDRAMを搭載する必要があったが，集積度や待機電力の大きさなどが課題となっている[1][16]．一方で，ストレージクラスメモリ（SCM）[2]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，高性能／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低いため，DRAMをSCMに単純に置き換えると処理性能の低下を招く恐れがある．そのため，SCMの高レイテンシを隠蔽する効果があるプリフェッチ制御などのメモリ制御がより重要になるが，不要なデータのプリフェッチはアクセス電力がDRAMより高いSCMでは消費電力の増加を招くため，従来よりも高精度な先読み制御が要求される．また，MRAMやPCMやReRAMなど種々のSCMが存在し，それぞれアクセスレイテンシや電力の特性に差があるため，特性の異なるSCMの性能を最大化するための複雑なメモリ制御[3]が求められる．さらに，DRAMとSCMを混載して主記憶を構成する場合には，アプリケーションのメモリアクセス特性に応じて両者を適応的に使い分ける階層制御をすることでSCMの高速性と省電力性を引き出し，効率良いデータ処理を実現する必要がある[11][10][3]．そのため，SCMとDRAMを混載することによりメモリ制御はさらに複雑化し，最適なメモリ制御方式を決定することが困難になる．精度の高いメモリ制御の実現のためには，事前にアプリケーションの性能情報の特性と最適な制御方式の関係を事前学習することで，メモリ階層化による複雑さに依らずに，特性に応じたメモリ制御方式を高速にオンライン判定する手法が求められる．本研究では，アプリケーション実行により得られるシステムレベルの性能情報をもとに機械学習を用いて最適なメモリ制御方式を予測可能にするフレームワークを提案する．提案方式をSCMの高レイテンシを隠蔽する効果があるプリフェッチ制御に適用し，電力効率の良いプリフェッチ制御方式の判定の有効性の評価を行う．

以下，第2章では，機械学習を用いたメモリ制御最適化のフレームワークについて説明する．第3章では，提案フレームワークをプリフェッチ制御に適用し有効性を示す．第4章では関連研究について説明し，最後に第5章でまとめと今後の課題について述べる．

2. 機械学習を用いたメモリ制御最適化

SCMで構成される主記憶におけるメモリ制御では，従来のDRAM主記憶以上に速度や電力を考慮したメモリ制御が必要となる．さらに，DRAMとの混載によって構成される主記憶では，メモリの階層化による複雑な階層制御が性能に重要な影響を及ぼす．本研究では，システムレベルの時系列性能情報から機械学習を用いて階層化された主記憶を含む様々な構成のメモリ制御最適化を実現するフレームワークを提案すると共に，レイテンシと消費電力に着目したSCMの効率使用を目指し，提案フレームワークをプリフェッチ制御に適用する方式について述べる．

† (株)東芝 研究開発センター
コンピュータアーキテクチャ・セキュリティラボラトリー
{mayuko.koezuka,yusuke1.shirota,satoshi.shirai,tatsunori.kanai}@toshiba.co.jp

2.1 メモリ制御最適化フレームワーク

図 1 に示されるメモリ制御最適化フレームワークは，大きく 2 つのフェーズによって構成される．図 1 のオフライン学習フェーズは，複数アプリケーションをターゲットプロセッサ上で実行することで得られるシステムレベルの性能情報と，最適化したい指標を取り纏めた最適化ポリシーを入力として，事前に性能情報の特性とメモリ制御方式の関係について機械学習を用いて学習し，最適なメモリ制御方式を決定する予測モデルを生成している．最適化ポリシーとは，アクセス速度や消費電力やメモリ寿命等，複数存在する最適化の指標に対して，各指標のトレードオフを考慮した最適化のルールを示したものである．図 1 に示されるオンライン判定フェーズは，最適化の対象となるアプリケーションを実行することで得られる性能情報から，オフライン学習フェーズで生成された予測モデルを用いて最適なメモリ制御方式を予測する．このように，本フレームワークを利用し，様々なアプリケーションの性能情報とメモリ制御方式の関係性を事前学習することで，システムレベルの性能情報のみを用いてリアルタイムにメモリ制御の最適化を実現することが可能となる．

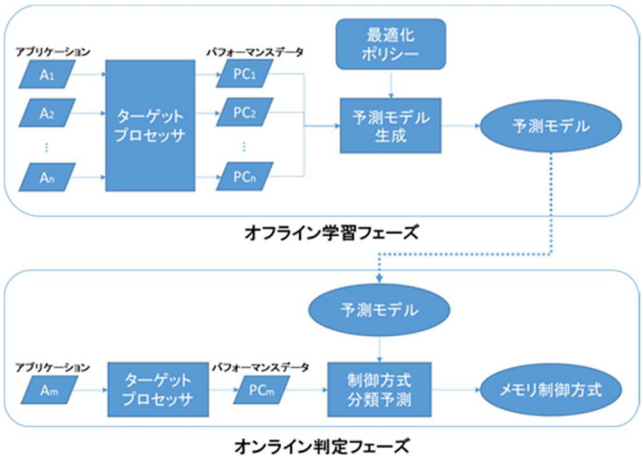

図 1　メモリ制御最適化フレームワーク

2.1.1 パフォーマンスカウンタ

オフライン学習フェーズにおいて予測モデル生成の入力となる性能情報は，Intel®プロセッサ[a]が備えるハードウェアイベントを測定するパフォーマンスカウンタ（Performance Monitoring Counter，以降 PMC）より収集される[5]．PMC では表 1 に示されるような各種ハードウェアイベントが測定可能である．本研究ではパフォーマンスカウンタから時系列に収集した時系列パフォーマンスカウンタデータを用いる．

表 1　パフォーマンスカウンタ（Haswell）

Hardware Event Name	Description
PAGE_WALKER_LOADS.DTLB_MEMORY	Number of DTLB page walker loads from memory
L2_LINES_OUT.DEMAND_DIRTY	L2 modified lines evicted by a demand request
BR_MISP_EXEC.ALL_BRANCHES	Counts all mispredicted near executed branches (not necessarily retired)
MEM_UOPS_RETIRED.ALL_STORES	All retired store instructions
LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache

2.1.2 機械学習

表 1 に示したハードウェアイベントは PMC で収集されるハードウェアイベントの一部であり，実際に PMC で収集可能なイベントの種類は膨大となる．プリフェッチ制御をはじめとする様々なメモリ制御の予測モデルの生成では，ターゲットとなるメモリ制御に関係するパフォーマンスカウンタを膨大な種類のイベントの中から選択する必要がある．さらに，異なるメモリ制御間に依存関係がある場合では，依存関係のあるメモリ制御の全てを考慮した予測モデルが必要となるが，このような予測モデルの作成に必要なイベントを手動で選択することは非常に困難である．そこで，パフォーマンスカウンタで取得可能なイベントを手動で選択することなく，予測モデルを自動的に生成する手法が求められる．機械学習は，入力データに対して自動的にルールを生成する手法であり，メモリ制御の予測モデル構築に最適である．本研究では，機械学習の手法の 1 つである教師あり学習を導入する．教師あり学習は，未知の入力データを自動分類する手法であり，事前に与えられる正解分類情報を例題とみなして，例題を学習することで予測モデルを生成し，未知の入力が与えられた際には予測モデルを用いて分類結果を予測するアルゴリズムである．

2.1.3 セグメント分割

メモリ制御ではパフォーマンスカウンタの特性に応じた制御が必要とされ，教師あり学習においても，性能情報の特性とメモリ制御方式の組み合わせを正解分類情報とすることから，時系列パフォーマンスカウンタデータをパターン分割し，パフォーマンスカウンタの特性とメモリ制御方式の関係を明確にする必要がある．アプリケーション実行によって取得された時系列パフォーマンスカウンタデータは時間毎にパフォーマンスカウンタが集計されているが，異なるメモリ制御方式を適用した場合には実行時間が変化してしまうため，実行時間を用いた時系列パフォーマンスカウンタデータのパターン分割では，性能情報の特性を正しく抽出できない．そこで，時系列パフォーマンスカウンタデータをアプリケーションプログラムの実行命令数によ

a)　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．

ってセグメント分割することで，メモリ制御方式に影響されること無く性能情報の特性を抽出可能となる．これにより，アプリケーションを変更することなく時系列パフォーマンスカウンタデータを収集できるため，多くのアプリケーションの学習が容易になる．

2.2 オフライン学習フェーズ

次にフレームワークの各フェーズについて説明する．オフライン学習フェーズは，学習用アプリケーションを複数回実行し，セグメント分割された時系列パフォーマンスカウンタデータと最適化ポリシーからメモリ制御の予測モデルを生成する．教師あり学習を用いた予測モデルの生成では，事前にセグメントの時系列パフォーマンスカウンタデータを統計情報に変換し，この統計情報と最適化ポリシーに基づいて決定されたセグメントにおける最適なメモリ制御方式を関連付けた正解分類情報を用いる．図 2 は 2 つのアプリケーション実行により得られた時系列パフォーマンスカウンタデータをセグメントに分割し，機械学習を用いて 2 種類のメモリ制御方式に分類する際のイメージ図である．図 2 では複数の正解分類情報から教師あり学習を用いて最適なメモリ制御方式毎のクラスタを生成している．正解分類情報はアプリケーション実行の結果得られた時系列パフォーマンスカウンタデータ A と B から算出された統計情報に対して，セグメント毎に 2 種類のメモリ制御方式を割り当てている．メモリ制御方式の分類では，収集した正解分類情報からメモリ制御毎のクラスタを生成する．クラスタは機械学習によって生成され，オンライン判定フェーズにおいて新しい時系列パフォーマンスカウンタデータが入力された際にセグメントが属するクラスタからメモリ制御方式を予測する．

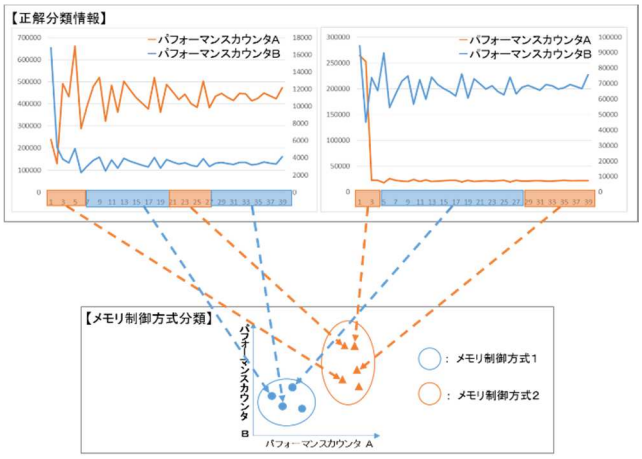

図 2　教師あり学習

教師あり学習には線形回帰，ロジスティック回帰，サポートベクターマシン（SVM），決定木，ランダムフォレスト，ニューラルネットワーク， k-近傍法（KNN）などが挙げられる[6]が，これら全てを提案するフレームワークに適用することはできない．本研究では，オンライン判定フェーズにおいて，リアルタイムにメモリ制御方式を予測する必要があるため，予測を行なう際に高速かつメモリ使用量が小さいことが求められる．予測速度が速い手法は決定木，線形回帰，ロジスティック回帰，KNN が挙げられる．ただし，KNN については，入力となるパラメータ数が増加する場合には予測速度が低下することが分かっており，本研究においてパフォーマンスカウンタのパラメータ数は制限されないため，除外する．さらに，メモリ使用量の少ないものは決定木と線形回帰に限られる．そこで，本研究では，モデルの妥当性を確認するため，クラスタの分類予測の基準が可視化される決定木を用いる．

決定木は，条件分岐によって属する分類クラスタを予測する手法であり，木構造によって複数の分岐が表現されている．予測対象のセグメントの統計情報は，ルートノードから探索を開始し，各ノードの条件に従った分岐によって，予測対象が分類されるべきクラスを示すリーフノードに辿り着く仕組みである．決定木は，リーフノードに辿り着くまでの分岐は決定木の高さ以下であることが保障されているため，限られた時間内で高速に予測結果を出すことが可能となる．

2.3 オンライン判定フェーズ

オンライン判定フェーズでは，オフライン学習フェーズで得られた予測モデルを用いて，最適化対象となるアプリケーションの実行時の動的な時系列パフォーマンスカウンタデータからリアルタイムにメモリ制御方式を予測し，決定する．本フレームワークでは，セグメント開始からメモリ制御方式判定終了までに期間は，メモリ制御方式が割り当てられないという問題が生じる．そこで，対象となるメモリ制御の特性に応じて，メモリ制御方式未設定期間のメモリ制御方式を決定する 2 つのアプローチを提案する．そのアプローチは，前のセグメントで実行されたメモリ制御方式を継続する継続型オンライン判定方式と，セグメントの切り替え時からメモリ制御方式判定が終了までの期間を常に決まったメモリ制御方式で実行する再設定型オンライン判定方式の 2 つである．継続型オンライン判定方式では，判定のためにメモリ制御方式を切り替えるオーバーヘッドが不要で，判定対象のセグメントと直前のセグメントにおける性能情報の特性は類似していることが多いケースでは，直前のセグメントで用いたプリフェッチ制御を継承することでより最適な制御が実現されると考えられる．一方で，再設定型オンライン判定方式では．入力となる時系列パフォーマンスカウンタデータがメモリ制御方式の変化に影響されるため，セグメント切り替え時には常に同一のメモリ制御に固定することで，決定木の精度が向上し，予測の精度が高められる．学習対象となるメモリ制御毎に最適なオ

ンライン判定方式は異なるが，本研究が対象とするプリフェッチ制御においては，プリフェッチ有効時に性能が向上する傾向があることとプリフェッチの切り替えによるオーバーヘッドが小さいことから，再設定型オンライン判定方式が適している．

2.4 プリフェッチ制御

本節では，SCM で構成される主記憶を用いるシステムにおいてメモリアクセス性能の向上に有用であり，メモリ階層制御方式の１つであるプリフェッチ制御を本フレームワークに適用する方法について述べる．

大規模データを処理するアプリケーションにおいては，メモリアクセスの最適化が重要であるが，その際に主記憶のアクセスレイテンシが大きな性能阻害要因となる．プロセッサのプリフェッチ制御は，将来アクセスするデータを予測し，キャッシュに事前にロードしキャッシュミスを防止することでこのアクセスレイテンシを隠蔽できるため，プロセッサの処理速度の向上に大きく寄与する．

現在の DRAM 主記憶を対象とする Intel® Xeon® プロセッサでは，プリフェッチ制御のための複数のハードウェアプリフェッチャが各コアに実装されている[9]．表 2 に Intel® Xeon® プロセッサ Haswell で実装されている 4 つのプリフェッチャを示す．各プリフェッチャは，例えばストリームアクセスを検出するとその次のキャッシュラインをプリフェッチするなど，シンプルな先読みアルゴリズムでデータアクセスを予測しプリフェッチを行っている．また，各プリフェッチャは個別に有効／無効が設定できるが，DRAM 主記憶では先読みが失敗してもペナルティが小さいため，デフォルトの設定ではすべてのプリフェッチャの設定が有効になっているのが一般的である．

表 2　Intel® Xeon® プロセッサ (Haswell)の
ハードウェアプリフェッチャ

ハードウェアプリフェッチャ	動作
L2 hardware prefetcher	連続するキャッシュラインがアクセスされると次のキャッシュラインもアクセスされると予測し，L2キャッシュへプリフェッチする
L2 adjacent cache line prefetcher	アクセスされたキャッシュラインのペアのキャッシュラインがアクセスされると予測し，L2キャッシュへプリフェッチする
DCU (Data Cache Unit) streamer prefetcher	一定期間内の同一キャッシュラインへのアクセスを検出すると次のキャッシュラインがアクセスされると予測し，L1キャッシュへプリフェッチする
DCU IP (Instruction Pointer-based) prefetcher	連続して実行されるロード命令の履歴からアクセスされるデータを予測し，L1キャッシュへプリフェッチする

DRAM より高レイテンシの SCM を主記憶として利用するようになると，キャッシュミス時のプロセッサのストール時間がより長くなるため，処理速度向上にプリフェッチの最適制御はますます重要になる．SCM のアクセスレイテンシを隠蔽するためには，実際のデータアクセス時にはキャッシュにデータが入っているように，プリフェッチは DRAM 主記憶の場合よりも早いタイミングで開始する必要も出てくる．そのため，使用する SCM のレイテンシ特性やその影響を考慮したプリフェッチ制御を行う必要がある．さらに，SCM は DRAM に比べてアクセス電力が高いため，予測ミスに起因する不要なデータのプリフェッチによりデータ読み出しに掛かる電力が増加する．また，不要なデータのプリフェッチにより必要なデータがキャッシュから追い出されることで，追い出しや再読み出しに掛かる電力も増加するため，消費電力への影響が大きい．これらの理由から，SCM 主記憶向けのプリフェッチでは，SCM のレイテンシ特性の影響を考慮した制御や，処理速度だけでなく電力効率も考慮した精度の高い制御が要求される．本研究ではではまずは後者に問題を限定し，機械学習を適用することで従来よりも高精度な予測制御による性能最適化を目指す．

3.　実験と評価

本章では，提案フレームワークのプリフェッチ制御への適用可能性について評価を行う．

現在，評価に利用可能な SCM が入手できないため，本稿では，Intel® Xeon®プロセッサと DRAM 主記憶で構成されるシステムを用いて，DRAM を SCM と見做して評価を行う．主記憶が DRAM のみで構成される場合に全てのハードウェアプリフェッチャの設定を有効にすることで速度性能及び電力性能が最適となるアプリケーションにおいても，主記憶が SCM で構成されている場合には性能が低下するケースがある．評価ではプリフェッチ制御を機械学習による予測モデルに応じて動的に切り替えることにより，アクセス速度だけでなく消費電力まで考慮した最適性能が得られることを示す．さらに，機械学習を適用し性能情報の特性とプリフェッチ制御の関係性を学習することで高速かつ電力効率の良いプリフェッチ制御の予測ができることを示す．

3.1 実験環境

実験には，プロセッサに Intel® Xeon® プロセッサ E5-2690 v3 (Haswell) 2.6 GHz，Ubuntu 16.04.2 (Linux 4.4.0-130-generic)を搭載したサーバを用いた．

実験データは，マルチプロセッサベンチマーク集である PARSEC[4]に含まれる 11 種類のアプリケーションを用いた．本稿では，最適化フレームワーク構築の最初のステップとして，オフライン学習フェーズ及びオンライン判定フェーズで同一のアプリケーションを用いて，特定のアプリケーションに対して適切な予測モデルが生成されることを示す．取得するハードウェアイベントは，プリフェッチ制

御の判定に影響がある可能性がある TLB，分岐等に関係する 31 個を選定した．また，正解分類情報に用いられるセグメントの時系列パフォーマンスカウンタデータの統計値には平均値を採用した．なお，本実験では，セグメントは 3G 命令毎に等間隔で分割することとした．

3.2 正解分類情報

本節では，決定木の生成に必要となる正解分類情報について説明する．正解分類情報はセグメント毎に生成され，最適化ポリシーに基づいて各セグメントの最適なプリフェッチ制御方式を選択する必要がある．今回の実験では，最適化ポリシーは，電力効率の良いプリフェッチ制御を目指し，各セグメントにおける実行時間と主記憶アクセスに掛かる電力量を考慮することとした．図 3 は最適プリフェッチ制御方式の判定を示しており，今回の実験では，最初に誤差 5%を許容した上でプリフェッチ有効時と無効時の実行時間（time_prefetcher_on / time_prefetcher_off）を比較する．誤差の許容は，プリフェッチの効果が小さく実行時間による差が少ない場合において，電力評価を最適プリフェッチ制御判定の基準とするためである．プリフェッチ有効時の実行時間がプリフェッチ無効時の実行時間の 95%よりも小さい場合には，そのセグメントの最適解をプリフェッチ有効とする．一方で，プリフェッチ有効時の実行時間が大きい場合には，プリフェッチ有効時と無効時の SCM 電力量（energy_prefetcher_on / energy_prefetcher_off）を比較し，より小さい方を最適なプリフェッチ制御結果とする．電力量は前提とする SCM の種類に応じて大きく異なるため[8]，本実験では簡単のため，DRAM へのアクセス回数を測定し，DRAM へのアクセス回数が多い方がメモリアクセス電力が高いと仮定した．

【最適プリフェッチ制御方式】

```
for i in segment_number:
  if time_prefetcher_on[i] <= time_prefetcher_off[i] * 0.95:
    best_prefetch[i] = "ON"
  elseif time_prefetcher_on[i] > time_prefetcher_off[i] * 0.95:
    if energy_prefetcher_on[i] <= energy_prefetcher_off[i]:
      best_prefetch[i] = "ON"
    elseif energy_prefetcher_on[i] > energy_prefetcher_off[i]:
      best_prefetch[i] = "OFF"
```

図 3 最適プリフェッチ制御方式

3.3 再設定型プリフェッチ方式の予測評価

再設定型プリフェッチ方式では，前節で生成した正解分類情報を元に，プリフェッチが有効である状態からどのようなプリフェッチ制御方式を予測する決定木を生成した．決定木の深さは 5, 6, 7 とし，それぞれの深さで評価を行う．決定木が深過ぎるケースでは，オーバーフィッティングによる精度の低下が発生するため，本研究では決定木の深さ 7 を上限とした．

図 4 は再設定型プリフェッチ学習によって得られたプリフェッチ制御の予測正答率を示したものであり，異なる深さの決定木から得られた結果を比較したものである．その結果，深さ 5，6，7 のいずれの決定木から導き出された予測結果に対しても予測正答率に変化は無かった．これは，プリフェッチ制御におけるパフォーマンスカウンタの特性を浅い決定木で表現できていること示す．また，6 つのアプリケーション（canneal, dedup, facesim, fluidanimate, freqmine, swaptions）で予測正答率が 60%を超えた一方で，精度の低いアプリケーション（blackscholes）では予測正答率が 43.48%となり，プリフェッチ制御に関わる特性が十分に決定木に反映されていないことが判明した．決定木を 8 以上にした場合には，さらに予測正答率が下がる結果も得られていることから，決定木の改善にはオフライン学習フェーズにおける改善が必要だと考えられる．

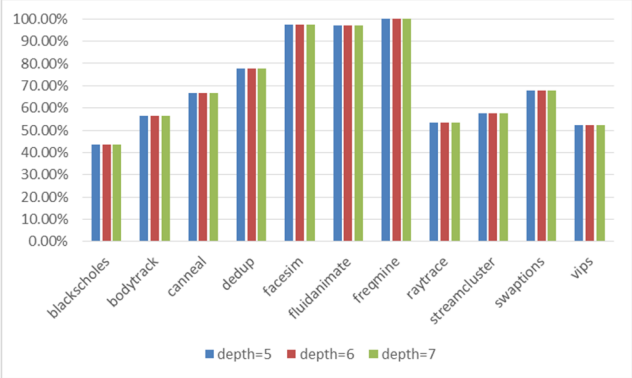

図 4 再設定型プリフェッチ学習の予測結果

図 5 は再設定型プリフェッチ学習によって生成された深さ 5 の決定木である．図 4 の結果より，深さ 5，6，7 の決定木においては予測正答率が変化しないという結果が得られたため，オンライン判定フェーズでは判定が最も高速となる深さ 5 の決定木を用いて予測を行う．決定木は，各セグメントの時系列パフォーマンスカウンタデータの平均値に基づいてルートノードから各ノードの条件に従って，リーフノードが示す分類結果である最適プリフェッチ制御方式に到達する仕組みである．各ノードには，セグメントにおける時系列パフォーマンスカウンタデータの平均値による分岐条件と，学習において各ノードに到達したサンプルの割合を示す sample と，該当ノード以降も含めた分類結果の比率（[プリフェッチ有効の比率, プリフェッチ無効の比率]）を示す value と，ノードの分類結果を示す class が表示されている．ノードの色は value によって決定され，プリフェッチ有効の比率が高いものは赤に，プリフェッチ無効の比率が高いものは青に色分けされている．色の濃淡についても同様に，value におけるプリフェッチ有効と無効の

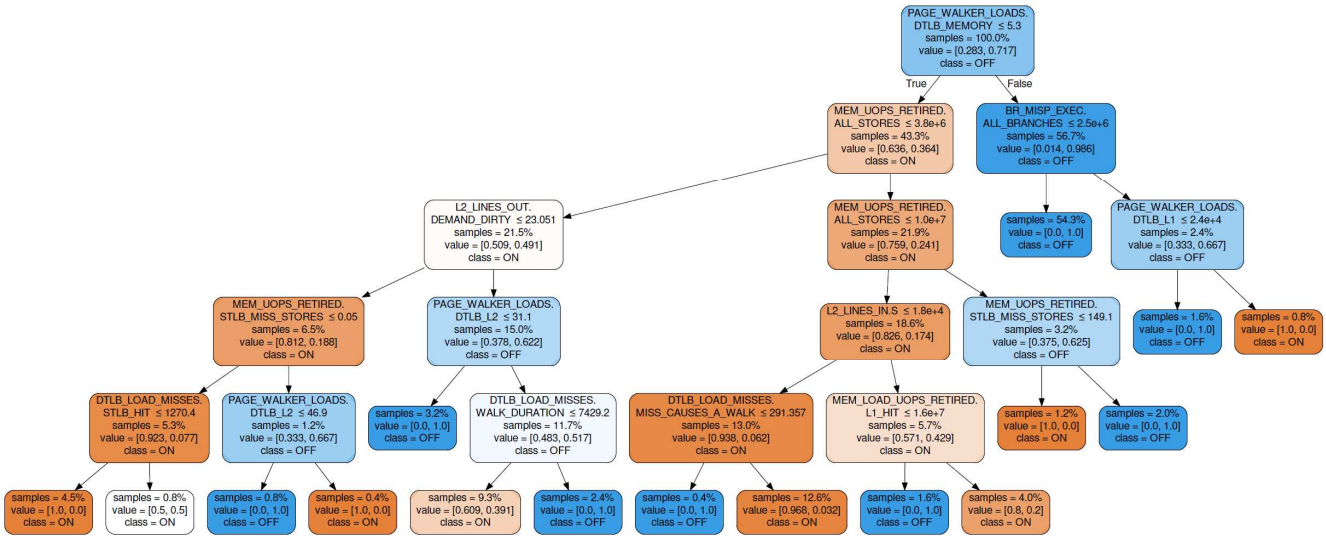

図 5　再設定型プリフェッチ学習の決定木

偏りが大きいものは濃く，偏りが小さいものは白に近づくようにカラーリングされている．

ここで，プリフェッチの有効無効の分類に大きく影響する 2 つのノードについて説明する．ルートノードは，ページウォークがメモリからロードした回数を示す PAGE_WALKER_LOADS.DTLB_MEMORY を用いた条件分岐となっている．この値が高い場合（FALSE），メモリアクセスがより広域に散っていることを示すため，プリフェッチのアクセス予測がしづらい状況であると考えられる．そのため，プリフェッチを無効にするのが望ましい．また，TRUE，TRUE によって到達するノードでは，デマンドリクエストにより追い出されたダーティな L2 キャッシュラインの数を示す L2_LINES_OUT.DEMAND_DIRTY を用いている．この値が高い場合（FALSE），プリフェッチが外れてキャッシュを汚すと不必要にこの値を増加させてしまう可能性がある．そのため，プリフェッチを抑えるのが有効である可能性がある．このように，プリフェッチ制御に大きく影響を与える分岐については妥当性があり，適切な学習が行われたと考えられる．

図 6 は 11 種のアプリケーションの実行速度を平均化したものであり，プリフェッチが常に無効である場合の実行時間（msec）と，プリフェッチが常に有効である場合の実行時間と，事前に準備した最適なプリフェッチ制御による実行時間と，再設定型プリフェッチ学習によって得られたプリフェッチ制御の実行速度の 4 種類で比較している．比較の結果，プリフェッチ有効時が最も実行時間が短くなっているが，これは正解分類情報生成の際に実行時間だけでなく電力量の評価を含めたためである．また，再設定型は最適値には及ばないものの，プリフェッチ最適値に近い値が得られており，速度性能に影響が出ないようにプリフェッチを無効にする判定ができていることが確認できる．本実験は DRAM を利用しているため，無効／有効の実行時間の差は DRAM 上での値だが，SCM になるとこの差はさらに拡大すると考えられるため，本実験で示した判定の効果はより大きくなると考えられる．

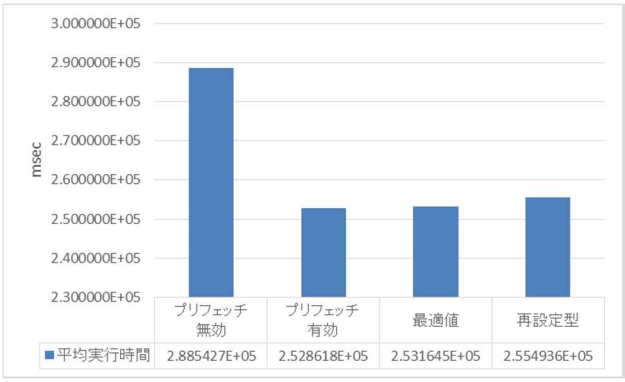

図 6　再設定型プリフェッチ学習の予測結果

3.4 考察

評価結果より，多くのアプリケーションにおいて深さ 5 の決定木によって精度の高い学習が行われたことが示された．また，決定木の深さ 5, 6, 7 の範囲においては，PARSEC に含まれるアプリケーションの予測の正答率に影響はないことが分かった．決定木における重要な分岐条件は SCM の特性を捉えており，妥当性があると確認されたことからも，上位のノードによる分岐が予測正答率に大きく影響したと考えられる．

速度性能については，提案した再設定型オンライン判定では最適値に近い結果が得られており，決定木の有用性が示された．また，プリフェッチを常に有効にした場合と比較して実行速度がわずかに低下している原因は，決定木が実行時間だけでなく電力量を含めた正解分類情報を用いて生成されているためである．DRAM／SCM 混載メモリを想定した場合には，電力量が実行速度と同様に重要な評価指

標となる．DRAM／SCM 混載のメモリ制御におけるプリフェッチの有効はアクセス電力の高い SCM を使う上で電力の増大を招くが，プリフェッチ無効の場合には大幅な速度低下が発生するため，決定木を用いたプリフェッチ制御は有用であると考えられる．

4. 関連研究

ソフトウェアの性能最適化を支援する技術としては，自動チューニング[12]が広く研究されている．自動チューニングは，プログラムの性能に影響を与える性能パラメータを組込み，対象計算機アーキテクチャに適合する値の組み合わせを自動的に決定する．また，自動チューニングに必要となる自動チューニング機能付きのソフトウェアを作成する自動チューニングフレームワークとして FIBER [13] や OpenTuner [14] などがある．本稿の提案方式は，時系列パフォーマンスカウンタデータなどのシステムレベルで得られる情報のみを利用して最適化しており，最適化のための対象アプリケーションプログラムの変更等も不要であるため，自動チューニングと組み合わせることも可能である．

コンピュータシステムの最適化をパフォーマンスカウンタデータなどのシステムレベルの情報を基に機械学習を用いておこなう研究には[7][15]などがある．本研究は SCM の階層制御を最適化の対象としている．

5. まとめと今後の課題

教師あり学習の 1 つである決定木を用いてメモリ制御方式を決定するフレームワークを提案した．SCM 主記憶においてキーとなるメモリ制御方式の１つであるプリフェッチ制御をモチーフに評価を行い，アプリケーションを効率よく学習し事前に決定木を生成することで，高速で正確なリアルタイム制御が出来る可能性があることを示し，フレームワークの有用性を確認した．

今後は，プリフェッチ制御の精度向上を目指し，決定木の改善と共に，時系列パフォーマンスカウンタデータの特徴をより正確に抽出するためのセグメント分割についても検討を進める．また，DRAM／SCM 混載主記憶の階層制御への適用を進めており，継続型オンライン判定を含めたフレームワークの拡張を行う．

謝辞 この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務の結果得られたものです．

参考文献

[1] Q. Deng, L. Ramos, R. Bianchini, D. Meisner and T. Wenisch, "Active Low-Power Modes for Main Memory with MemScale," in IEEE Micro, vol. 32, no. 3, pp. 60-69, 2012.

[2] R. F. Freitas and 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] 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.

[4] C. Bienia, S. Kumar, J. P. Singh, K. Li, “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” Princeton University Technical Report TR-811-08, 2008.

[5] Intel® 64 and IA-32 Architectures Software Developer Manuals: https://software.intel.com/en-us/articles/intel-sdm

[6] C. M. Bishop, “Pattern recognition and machine learning,” New York: Springer, 2006.

[7] S. Liao, T. Hung, D. Nguyen, C. Chou, C. Tu, H. Zhou, "Machine learning-based prefetch optimization for data center applications," In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009, pp. 1-10.

[8] S. Chen, P. B. Gibbons, S. Nath, “Rethinking Database Algorithms for Phase Change Memory,” In Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR'11), pp. 21-31, 2011.

[9] Intel® 64 and IA-32 Architectures Optimization Reference Manual: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[10] Y. Park and H. Bahn, “Efficient management of PCM-based swap systems with a small page size,” Journal of Semiconductor Technology and Science, vol. 15, no. 5, pp. 476-484, 2015.

[11] A. Suresh, P. Cicotti, L. Carrington, “Evaluation of emerging memory technologies for HPC, data intensive applications”, IEEE International Conference on Cluster Computing (CLUSTER), pp. 239-247, 2014.

[12] 片桐孝洋, “ソフトウェア自動チューニング －数値計算ソフトウェアへの適用とその可能性－,” 慧文社, 2004.

[13] T. Katagiri, K. Kise, H. Honda, T. Yuba, "FIBER: A General Framework for Auto-Tuning Software," In Proceedings of the Fifth International Symposium on High Performance Computing (ISHPC-V), pp.146-159, 2003.

[14] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U. M. O'Reilly, S. Amarasinghe, “OpenTuner: an extensible framework for program autotuning,” In Proceedings of the 23rd international conference on Parallel architectures and compilation (PACT '14), pp. 303-316, 2014.

[15] 佐々木広，浅井雅司，池田佳路，近藤正章，中村宏, “統計処理に基づく動的電源電圧制御手法”，情報処理学会論文誌, Vol.47, No.SIG18 (ACS 16), pp.80-91, 2006.

[16] S. Miwa, and H. Honda, “Memory Hotplug for Energy Savings of HPC systems,” The International Conference for High Performance Computing, Networking, Storage and Analysis (SC’15), 2015.