Vol.2018-HPC-167 No.6
2018/12/17

SCM/DRAM 混載主記憶向けハイブリッドアクセス制御方式

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．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，高速・低消費電力でかつスケーラブルな主記憶を実現するためには，SCM と DRAM を組み合わせて主記憶を構成し，アプリケーションのメモリアクセス特性に応じて適応的に使い分ける階層制御を行い，両者の良さを引き出す必要がある．本研究では，SCM/DRAM 混載主記憶において，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メモリ階層制御方式であるハイブリッドアクセス制御方式を提案する．本稿では，機械学習を用いて生成したワーキングセットサイズ予測モデルを用いて，低コストで取得可能な時系列パフォーマンスデータからワーキングセットサイズを実行時に推定し，ハイブリッドアクセス制御方式における各制御方式を適応的に切り替える自動最適化手法を検討し評価を行った．

キーワード：ストレージクラスメモリ(SCM)，機械学習，仮想記憶，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, 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，MRAM（Magnetoresistive Random Access memory）や PCM（Phase-change Memory），ReRAM（Resistive Random Access Memory）等のストレージクラスメモリ（SCM: Storage-class Memory）[2][7]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大きいため，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，高性能化のためには，SCM と DRAM を組み合わせて主記憶を構成し，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をすることで両者の良さを引き出し，効率良いデータ処理を実現する必要がある[3][10][11].

本研究では，SCM/DRAM 混載主記憶において，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メモリ階層制御方式であるハイブリッドアクセス制御方式を提案する．本稿では，機械学習を用いて生成したワーキングセットサイズ予測モデルを用いて，低コストで取得可能な時系列パフォーマンスデータからアプリケーションプログラムのワーキングセットサイズを実行時に推定し，ハイブリッドアクセス制御方式における各制御方式を適応的に切り替える自動最適化手法を検討し評価を行う．

以下，第 2 章では，研究の背景について説明する．第 3 章では，ハイブリッドアクセス制御方式について述べる．第 4 章では，提案方式の有効性を示す．第 5 章で関連研究について説明し，最後に第 6 章でまとめと今後の課題について述べる．

2. SCM/DRAM 混載主記憶におけるメモリ制御の課題

次世代の HPC システムにおける大規模データ処理で要求されている低消費電力でかつスケーラブルな主記憶を実現するためには，小容量の高速な DRAM と大容量の相対的には低速な SCM を混載させる必要があるが，新しいメモリ階層の中で特性が異なる 2 つのメモリをどのように階層制御するかが大きな課題となる．さらに，2 つのメモリをアプリケーション開発者に陽に見せるとこれらをどのように使い分けるかを開発者に強いることになりプログラミングが複雑になるため，データ配置等のメモリ管理の自動最適化手法が重要になる．

先行研究[3][10]では，OS の仮想記憶を拡張しアプリケーション開発者には単一の主記憶があるように見せて大規模データ処理向けのプログラミングをシンプルにする方式が提案されている．具体的には，SCM を仮想記憶のスワップデバイスとして DRAM と共に主記憶としてシステムに混載し，高速かつ待機消費電力が小さい SCM の特性を活かして DRAM 上のデータを積極的に SCM に退避して，使用する DRAM サイズを削減するとともに未使用

† (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
{yusuke1.shirota,satoshi.shirai,tatsunori.kanai}@toshiba.co.jp

DRAM の電源をオフすることで動作時の待機消費電力を削減する．この方式は，SCM のアクセスレイテンシが，一般的に想定されている数百 ns～数 μs という幅広いレンジの中で最も遅い数 μs の場合でもメモリアクセスのローカリティの高いアクセスパターンを持つアプリケーションでは速度性能を維持しつつ低消費電力化でき，SCM に適した有効な方式であることが明らかになっている．

しかし，ローカリティが低い場合，従来型の仮想記憶方式を踏襲し DRAM を SCM のキャッシュとして使うと，性能が大幅に低下する課題が確認されている[3]．細かい粒度で広域にランダムアクセスするアクセスパターンを持つアプリケーションでは，極端なケースでは必要なページを 4KB などの単位でページインしたあとで，そのページの数バイトのみにアクセスし，すぐにページアウトされてしまう．このように，従来型の仮想記憶方式では SCM の低レイテンシを持つスワップデバイスが想定されていなかったため，これまで顕在化しなかった課題が明らかになっていた．

3. ハイブリッドアクセス制御方式

本章では，2 章で述べた課題を解決する SCM/DRAM 混載主記憶におけるハイブリッドアクセス型のメモリ制御方式について述べる．

3.1 ハイブリッドアクセス制御方式の概要

HPC システムの主記憶として従来の DRAM に加えて SCM を搭載するようになると，両者をうまく使い分けることが必要になる．SCM は DRAM より大容量だがアクセスレイテンシが大きいので，処理対象のデータの特性に合わせて，データを配置するメモリを DRAM と SCM にうまく分散させることで，効率良いデータ処理が可能になる．すなわち，メモリアクセスのローカリティが高いデータは SCM から DRAM にコピーして，プロセッサは DRAM 上のデータをアクセスするページングが望ましい．また，メモリアクセスのローカリティが低くサイズの大きいデータは，SCM に配置してプロセッサが SCM をダイレクトにアクセスするのが望ましい．しかし，このような異なる性質を持つ主記憶に，どのようにデータを分散させると効率が良いかをアプリケーション開発者が判断するのは困難である．そのため，ハイブリッドアクセス制御方式は，アプリケーションのメモリアクセス特性に応じてページングとダイレクトアクセスの切り替えを自動的に行う．

3.1.1 SCM の低レイテンシを活かした積極的ページング

ページング方式では，SCM 上のデータをアクセスする際に，OS のページ単位で DRAM へ転送することで，DRAM を SCM のキャッシュとして利用する．従来の仮想記憶では，DRAM にデータが乗り切らなくなると SSD/HDD で構成されるスワップデバイスとの間の msec オーダの低速なデータ転送により性能が大幅に低下するため，大規模データ処理ではページフォルト回避のために大容量の DRAM を搭載する必要があった．SCM アウェアな新しい仮想記憶では，従来とは異なり，ある程度のデータ転送の増加は容認して積極的に DRAM 上のページを SCM に退避させることができるので，大容量 SCM に小容量の DRAM を組み合わせた主記憶を構成することができる．さらには，使用 DRAM 量を動的に削減できる可能性もある．この場合，アプリケーションのメモリアクセス特性に応じて DRAM サイズをどのように調整するかが重要になる．

3.1.2 SCM のバイトアドレッサビリティを活かしたダイレクトアクセス

SCM はストレージ的な側面とメモリ的な側面を持ち合わせるが，ダイレクトアクセス方式は後者を活かした制御方式である．本制御方式は，SCM のバイトアドレッサビリティを活かして，プロセッサがメモリバス接続された SCM に対して通常のロード・ストア命令によりキャッシュライン単位で直接アクセスする．SCM は，相対的には高レイテンシだが，ページングで効率よく処理できないメモリアクセスのローカリティが低いアクセスパターンに対しては，必要なキャッシュラインのみアクセスできるので，ページ単位のデータ転送を行うページングに対して速度や消費電力の面で有利になる可能性がある．

3.2 ワーキングセットサイズ予測モデルを用いた最適化方式

本節では，ページングとダイレクトアクセスをアプリケーションのメモリアクセス特性に応じて適応的に切り替える自動最適化方式の概要について説明する．

3.2.1 メモリ制御最適化フレームワーク

異種メモリ混載主記憶では，複雑な階層制御が求められる．我々は，システムレベルの時系列パフォーマンスデータから機械学習を用いてメモリ制御最適化を実現するフレームワークを提案している[19]．

図 1 に示すメモリ制御最適化フレームワークは，2 つのフェーズによって構成される．

オフライン学習フェーズは，複数の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取得できるシステムレベルのパフォーマンスデータと，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，事前にパフォーマンスデータ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する．最適化ポリシーとは，処理時間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

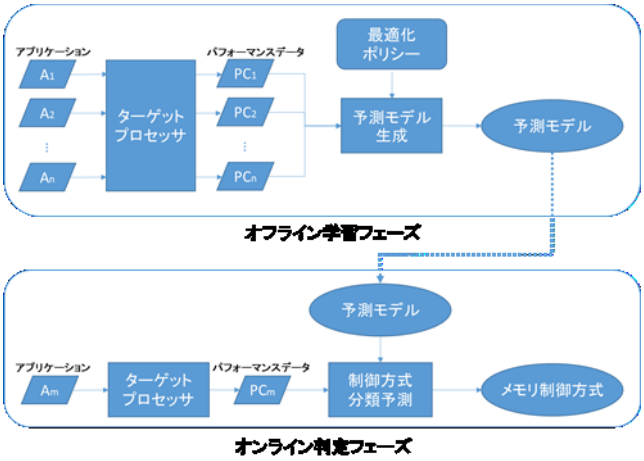

図 1　メモリ制御最適化フレームワーク

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．

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パフォーマンスデータから，オフライン学習フェーズで生成し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パフォーマンスデータとメモリ制御方式の関係性を事前学習することで，アプリケーション開発者に 2 つのメモリの使い分けを意識させることなく，システムレベルのパフォーマンスデータのみを用いてリアルタイムにメモリ制御の最適化を行う．

3.2.2 ワーキングセットサイズ予測モデルを用いた最適化方式の概要

我々は，図 1 で示したフレームワークでハイブリッドアクセス制御の最適な制御方式を決定する方法として，[19]で示したように直接最適な制御方式を予測する方法や，最適な制御方式と相関の高いメモリアクセス特性を予測して最適な制御方式を決定する方法など，様々なアプローチで検討を進めている．本稿では，後者について，ハイブリッドアクセス制御方式が有する 2 つのパラメータである，制御方式切り替えタイミングとページング時の使用 DRAM サイズの動的制御をするために，両パラメータと相関の高いメモリアクセス特性としてワーキングセットサイズに着目し，ワーキングセットサイズを用いてハイブリッドアクセス制御の最適な制御方式を決定する方式を提案する．

ワーキングセットサイズ（Working Set Size，以降 WSS）[20][21]とは，プログラムの全体の実行区間を分割した各区間（以降，セグメント）においてプログラムが実際に使用する仮想アドレス空間のサイズである．例えば，あるセグメントにおいて，仮想ページを N ページ参照した場合，ワーキングセットサイズは，N * 4K バイトとなる．

ワーキングセットサイズ予測モデルを用いてハイブリッドアクセス制御の最適な制御方式を判定する方式の概要を示す．ページングとダイレクトアクセスの切り替え判定に必要なメモリアクセス特性として最適化対象プログラムのワーキングセットサイズを求め，同ワーキングセットサイズを用いて最適な制御方式を判定する．しかし，後述するように，ワーキングセットサイズを実行時に算出するのは難しいため，事前に機械学習を用いて生成したワーキングセットサイズ予測モデルを使って高速にワーキングセットサイズを推定し，最適な制御方式を判定する．

最適な制御方式の判定は，前述のように，大きく 2 つのフェーズによって行われる．

オフライン学習フェーズは，複数の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パフォーマンスデータを入力として，事前にパフォーマンスデータの特性とワーキングセットサイズの関係について機械学習を用いて学習し，最適な制御方式を決定する予測モデルを生成する．

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パフォーマンスデータから，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ワーキングセットサイズを予測する．そして，推定ワーキングセットサイズを用いて最適な制御方式を決定する．

3.2.3 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パフォーマンスデータは，Intel®プロセッサ[a]が備えるプロセッサコアやメモリコントローラなどの周辺回路におけ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）[5] より収集する．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を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．

3.2.4 オフライン学習フェーズ

オフライン学習フェーズでは，時系列パフォーマンスカウンタデータや OS 内部で管理している各種統計情報などのパフォーマンスデータを入力としワーキングセットサイズを出力とする予測モデルを学習する．

ワーキングセットサイズは，アプリケーションの実行中にアプリケーションの処理内容に応じて動的に変化する．これを実行中に精度よく求めて，最適な制御方式の判定に利用したいが，一般的には難しい．ワーキングセットサイ

[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．

表 1　使用するパフォーマンスカウンタのイベント

Hardware Event Name	Description
DTLB_LOAD_MISSES.STLB_HIT	Number of cache load STLB hits. No page walk.
DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	Misses in all TLB levels that cause a page walk of any page size.
DTLB_STORE_MISSES.STLB_HIT	Store operations that miss the first TLB level but hit the second and do not cause page walks.
DTLB_STORE_MISSES.MISS_CAUSES_A_WALK	Miss in all TLB levels causes a page walk of any page size (4K/2M/4M/1G).
PAGE_WALKER_LOADS.DTLB_L1	Number of DTLB page walker loads that hit in the L1+FB.
PAGE_WALKER_LOADS.DTLB_L2	Number of DTLB page walker loads that hit in the L2.
PAGE_WALKER_LOADS.DTLB_L3	Number of DTLB page walker loads that hit in the L3.
PAGE_WALKER_LOADS.DTLB_MEMORY	Number of DTLB page walker loads from memory.

ズを算出するためには，対象セグメント開始時に，アプリケーションのプロセスにマップされているページの参照フラグをクリアして，同セグメント終了後に同フラグを集計処理することでセグメントにおいて実際にアクセスされたページ数を求める必要がある．そのため，大規模データ処理でアクセスするデータサイズが多い場合時間が掛かってしまうため，リアルタイムに求めるのは難しい．

そこで，ワーキングセットサイズを直接算出せずに，実行時に低コストで取得できる上述のパフォーマンスデータを用いて，ワーキングセットサイズを予測し，高速に求める．

本稿では，予測モデルの生成に，機械学習の手法の 1 つである教師あり学習[22]を導入する．パフォーマンスデータを入力としワーキングセットサイズを出力とする予測モデルは，教師データセットを用いて学習して生成する．各教師データは，セグメントごとに生成された，複数のパフォーマンスデータと，これに対応する正解情報であるワーキングセットサイズである．

教師データセットを取得する際に，時系列パフォーマンスカウンタデータと正解となるワーキングセットサイズは，学習用のアプリケーションを 2 回実行して別々に取得する．後者の取得には，前述のとおりアクセス済の参照フラグの集計処理等の処理を単位セグメントごとに実行する必要があるため，後者の取得処理自体が前者に影響を与えてしまう可能性があるためである．また，時間の掛かる後者の取得処理に対して，オンライン判定時にも取得する前者は高速に取得可能であるため，別々に取得した後は，単位セグメントに対応する，アプリケーションプログラムの命令単位ごとに，パフォーマンスデータとワーキングセットサイズとの対応を示す教師データを生成する．

また，その際に，パフォーマンスデータは，ワーキングセットサイズと相関が強いものを選択する必要がある．予測モデルの生成では，ターゲットとなるメモリアクセス特性に関係するパフォーマンスカウンタを膨大な種類の PMC で収集されるハードウェアイベントの中から選択することも重要になる．

教師あり学習のアルゴリズムは，線形回帰，K-Nearest Neighbor (KNN)，Regression Tree [6]などを用いることができる．例えば，線形回帰は，予測子変数であるパフォーマンスデータの入力ベクタが与えられると，多重線形回帰モデルにより応答変数であるワーキングセットサイズを推定することができる．データの近傍性に基づく手法である KNN 等も回帰問題にも適用できる．これらを用いた学習の結果として，ワーキングセットサイズを推定可能な予測モデルが生成される．

3.2.5 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生成した予測モデルを用いて，最適化対象のアプリケーションの実行時のパフォーマンスデータからリアルタイムにワーキングセットサイズを推定し，最適な制御方式を判定する．

図 2 に示す, 制御方式を判定する方法について説明する．判定は，ワーキングセットサイズの予測値の，アプリケーションのプロセスに割り当てられた物理メモリサイズである Resident Set Size [24] (以降，RSS) に対する比率と，閾値とを比較して行う．推定ワーキングセットサイズの RSS に対する比率が閾値より小さい場合，ページング方式を選択する．これは，アプリケーション全体で利用する可能性があるメモリ（つまり RSS）に対して，その一部にメモリアクセスが集中している状況であるため，ローカリティが高いと判断でき，ページングが望ましいためである．

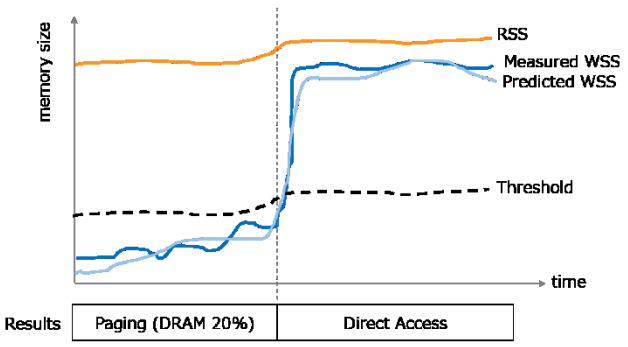

図 2　ハイブリッドアクセス制御の最適な制御方式の判定

一方，推定ワーキングセットサイズに対する比率が閾値より大きい場合，ダイレクトアクセス方式を選択する．これは，アプリケーション全体で利用する可能性があるメモリに対して，全域的にアクセスが散っている状況であるため，ローカリティが低く，ページング方式を採用するとDRAM へページをコピーしてもすぐに SCM へ追い出され入れ替え処理が頻発してしまうため，かえって性能が低下してしまうためである．そのため，ダイレクトアクセスを採用することでメモリアクセスを効率化できる．

また，推定ワーキングセットサイズを用いてページング方式が最適な制御方式と判定された場合，さらに，同推定値を使って DRAM サイズを調整することができる．例えば，使用可能な DRAM サイズを推定値に近いサイズにしてもよく，それ以外の DRAM 領域はパワーオフあるいはセルフリフレッシュモードなどの低消費電力モードに設定することで低消費電力化できる可能性がある．

4. 評価実験

本稿では，ワーキングセットサイズ予測モデルを用いたハイブリッドアクセス制御方式の有効性の評価を行った．まずは，機械学習を用いて生成したワーキングセットサイズ予測モデルの評価を行った．次に，予測した推定ワーキングセットサイズを用いてハイブリッドアクセス制御方式における各制御方式を適応的に切り替える自動最適化方式の評価を行った．

4.1 実験環境

実験には，プロセッサに Intel® Xeon® プロセッサ E7-8870 v3 (Haswell) 2.1GHz，メモリに DDR4-2400，OS は Ubuntu 16.04.2 LTS（Linux 4.4.0-134-generic）を搭載したサーバを用いた．

また，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である PARSEC[4]に含まれる，メモリアクセスのローカリティが高い facesim とローカリティが極端に低い canneal を用いた．問題サイズは一番大きい native を使用した．

4.2 ワーキングセットサイズ予測モデルの評価

まず，facesim と canneal を対象にワーキングセットサイズと RSS を測定した結果をそれぞれ図 3 と図 4 に示す．測定は 100ms 間隔で行った．各測定区間の測定開始時に，アプリケーションプロセスが使用するページの参照フラグをクリアして，区間終了時に同フラグを集計処理することでワーキングセットサイズを直接的に求めた．図より，facesim は RSS の一部を集中的にアクセスしているのに対し，canneal は RSS の広域をアクセスしているのが分かる．

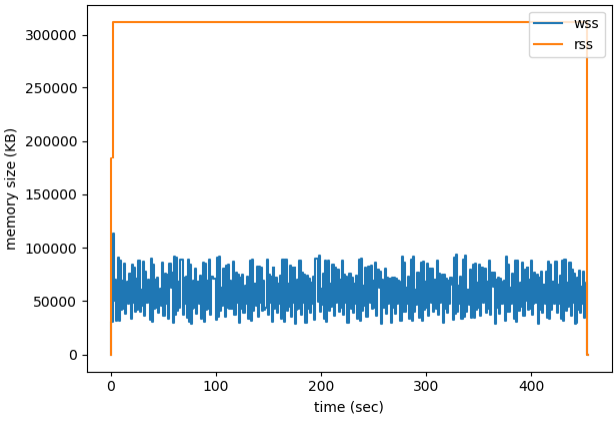

図 3　実測ワーキングセットサイズ（facesim）

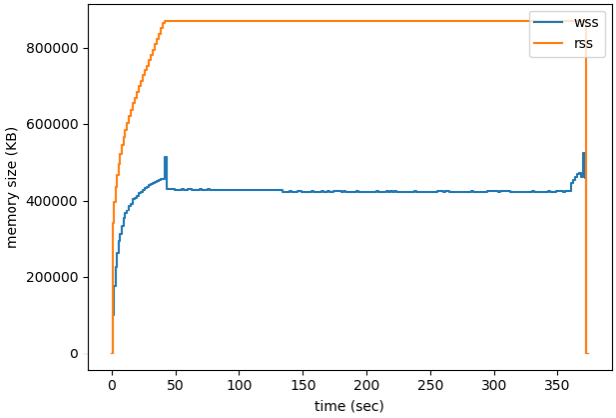

図 4　実測ワーキングセットサイズ（canneal）

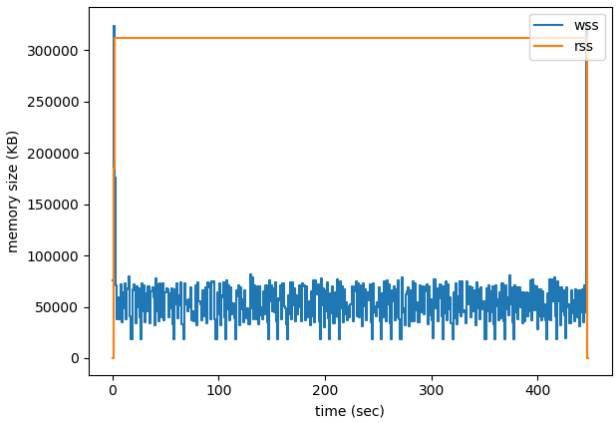

図 5　推定ワーキングセットサイズ（facesim）

次に，facesim と canneal についてワーキングセットサイズ予測モデルを用いて推定ワーキングセットサイズを求めた結果をそれぞれ図 5 と図 6 に示す．

本稿では，ワーキングセットサイズ予測モデルは，線形回帰を利用して作成した．予測モデルで利用するパフォーマンスデータには，TLB ミス率に関連する表 1 に示す 8 つ

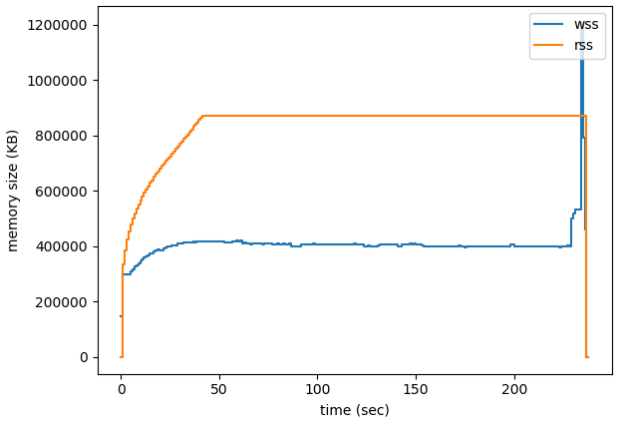

図 6　推定ワーキングセットサイズ（canneal）

の時系列パフォーマンスカウンタデータと RSS を用いた．ワーキングセットは RSS に対するメモリアクセスの散り具合を示すので，RSS と，メモリアクセスが広域に対して行われ TLB リーチより大きくなると発生する TLB ミスは，ワーキングセットサイズと強い相関があると考えられる．

推定ワーキングセットサイズと前述の実測ワーキングセットサイズを比較した結果，大きな誤差なく予測できることが確認できた．

4.3 ハイブリッドアクセス制御自動最適化方式の評価

推定ワーキングセットサイズを用いてハイブリッドアクセス制御方式における各制御方式を適応的に切り替える最適化方式の評価を行った．

4.3.1 異種メモリ混載主記憶エミュレーションによる評価方法

提案方式の評価は，提案方式で決定した制御方式による性能と，最適な制御方式による性能を比較することで行った．

ただし，現在，評価に利用可能な実 SCM が入手できないため，本稿ではエミュレーションにより評価を行った．具体的には，各制御方式について，SCM と DRAM の 2 種類のアクセスレイテンシを持つ主記憶上でアプリケーションを実行する挙動を，DRAM のみで実行した挙動から精緻に推測する手法を用いた．

ページング方式のエミュレーションには，仮想記憶の持つスワップ機構を利用した．まず，RAM ディスク上にスワップ領域を作成することで高速スワップを実現しこれを SCM 領域と見做し，さらに，特定のプロセスのリソース使用量を制限する Linux カーネル機能 control groups を使って利用可能な DRAM 容量を制限し積極的ページングを模擬した．使用する DRAM 容量に対してそれに見合う性能効率化が得られる場合に限定してページングを選択するのが望ましいため，DRAM 割当比率は RSS に対して 10%から 30%まで 10%刻みで変化させてベンチマークプログラムの実行時間を計測し，ページング適用時の推定実行時間とした．

一方，ダイレクトアクセス方式のエミュレーションについては，DRAM 上で実行した際に取得可能な時系列パフォーマンスカウンタデータを用いて SCM 上で実行した際に増加する遅延を予測する手法[17][25]を用いて，ダイレクトアクセス適用時の推定実行時間を算出した．

以上の異種メモリ混載主記憶エミュレーションの結果の例を図 7 に示す．図のマトリクスの各行は各制御方式（下からダイレクトアクセス，ページングは DRAM 割当比率 10%から 100%まで 10%刻みの合計 11 通り）のエミュレーション結果に対応し，各列はアプリケーションプログラムの実行時間に対応する実行命令数により分割したセグメントに対応している．セグメント毎の推定実行時間は，垂直方向の棒グラフで示している．

最適な制御方式による性能は，最適化ポリシーに基づいてセグメント毎に決定した最適な制御方式を適用した際の性能として，このマトリクスを基に算出する．今回の評価実験では，最適化ポリシーは，電力効率の良いメモリ制御を目指し，各セグメントにおける DRAM 量を抑えることを考慮した．まずは誤差 5%を許容した上で各制御方式（ページングの DRAM 割当比率は 30%以下に限定）の推定実行時間を比較し，推定実行時間による差が許容誤差内に収まればそのセグメントの最適解は DRAM 割当比率が小さい方とした．

図 7 に，最適化ポリシーに基づいたセグメント毎の最適制御方式の選択例を示す．まず，実行時間が最小の制御方式が選択（ピンク）された後，実行時間差が許容範囲内の DRAM 量を抑えた方式があれば最適制御方式として選択（青）される．なければ前者が最適制御方式として選択（紫）される．

4.3.2 評価結果

提案方式で決定した制御方式による性能と，最適制御方式による性能を比較した．

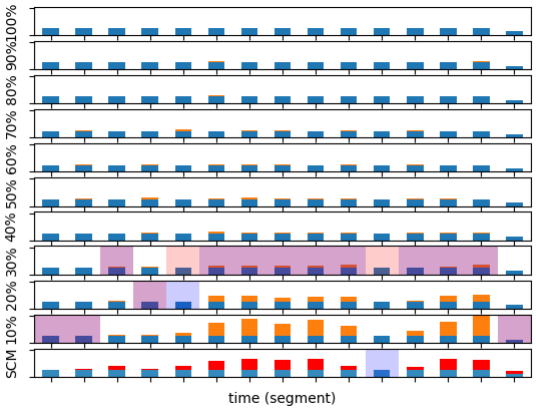

図 7　最適化ポリシーに基づいた最適制御方式の選択例

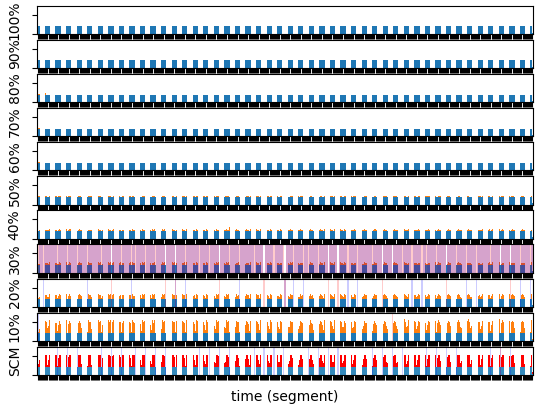

図 8　最適化ポリシーに基づいた最適制御（facesim）

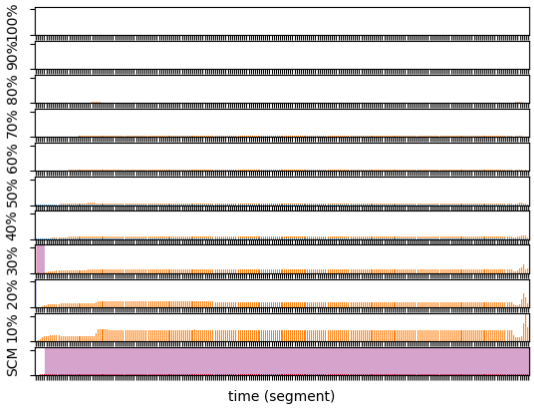

図 9　最適化ポリシーに基づいた最適制御（canneal）

まず，facesim と canneal の最適制御をオフラインで求めた結果をそれぞれ図 8 と図 9 に示す．図より，メモリアクセスのローカリティが高い facesim は DRAM 割当比率を 30%にしたページングに最適解が集中し，ローカリティが低い canneal は基本的にはダイレクトアクセスが最適解として選択されていることが確認できた．

この最適制御による性能と，提案方式による性能を比較した．提案方式の評価に用いた制御方式の判定用閾値は，使用する DRAM 容量に対してそれに見合う性能効率化が得られる場合に限定してページングを選択するのが望ましいため，本稿では RSS の 1/3 とした．

まず，図 10 に，facesim の実行時間を比較した結果を示す．推定ワーキングセットサイズを用いた提案方式に基づいた制御（Predicted WSS）を行うことで，最適解（Optimal）に近い実行時間が得られることが確認できた．実行時間が最適解より大きいのは，実行時間の観点からは結果的に小さな DRAM サイズの選択がややアグレッシブ過ぎたためである．また，提案方式と実測ワーキングセットサイズに基づく制御（Measured WSS）の実行時間差が小さいことから予測モデルの精度に問題がないことも確認できた．

次に，図 11 に，各セグメントで使用した DRAM サイズの累積値を比較した結果を示す．DRAM のみで実行した場合（ALL DRAM）に対し，提案方式は DRAM サイズが大きく削減できているのが分かる．最適解より削減できているのは，前述の DRAM サイズのアグレッシブな選択のためである．

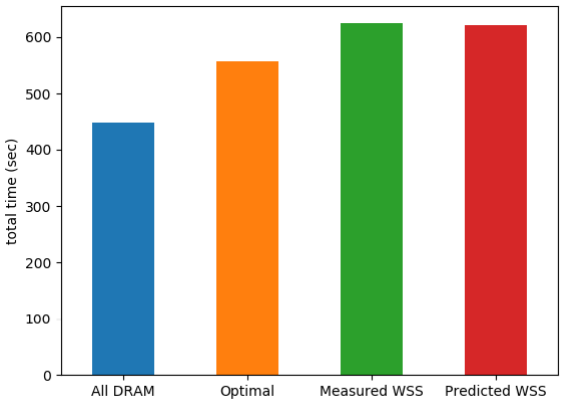

図 10　実行時間の比較（facesim）

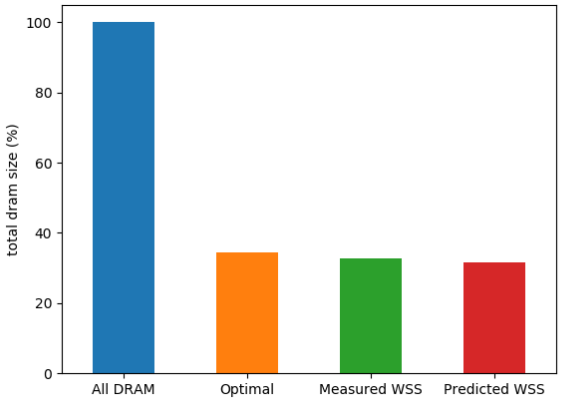

図 11　DRAM サイズの比較（facesim）

5. 関連研究

メモリ階層の変化に伴い複雑化するメモリ制御の最適化を自動化する研究には，[3][8][9][10][23]などがある．ページ単位で最適化する方法としては，参照されてから時間が短いページを DRAM に配置する手法[8]やメモリ間でのページマイグレーションがシステム性能に与える影響を予測しながらマイグレーション判定を行う手法[9]等がある．

また，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][13][14]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，ターゲット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．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．

6. まとめと今後の課題

SCM/DRAM 混載主記憶において，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メモリ階層制御方式であるハイブリッドアクセス制御方式を提案した．本稿では，機械学習を用いて生成したワーキングセットサイズ予測モデルを用いて，低コストで取得可能な時系列パフォーマンスデータからワーキングセットサイズを実行時に推定し，ハイブリッドアクセス制御方式における各制御方式を適応的に切り替える最適化方式を検討し評価を行い，有効性を確認した．

今後は，予測モデルの精度向上を目指すと共に，時系列のパフォーマンス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，本稿では最適な制御方式と相関の高いメモリアクセス特性を予測して最適な制御方式を決定する方式について検討したが，直接最適な制御方式を予測する方式との比較も行っていく．また，ページ単位でハイブリッドアクセス制御方式の各方式を切り替える方式についても検討を進める．

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] G. Sun, J. Zhao, M. Poremba, C. Xu, Y. Xie, “Memory that never forgets: emerging nonvolatile memory and the implication for architecture design, “, National Science Review, vol. 5, no. 4, pp. 2018, pp. 577-592, 2018.
[8] M. K. Qureshi, V. Srinivasan, J. A. Rivers, “Scalable High Performance Main Memory System Using Phase-Change Memory Technology,” In Proceedings of ISCA, pp. 24-33, 2009.
[9] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang and O. Mutlu, “Utility-Based Hybrid Memory Management,” In Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp. 152-165, 2017.
[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,” In Proceedings of IEEE International Conference on Cluster Computing (CLUSTER), pp. 239-247, 2014.
[12] T. Katagiri, D. Takahashi, “Japanese Auto-tuning Research: Auto-tuning Languages and FFT,” In Proceedings of the IEEE, vol. 106, no. 11, pp. 2056-2067, 2018.
[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] S. R. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, K. Schwan, “Data Tiering in Heterogeneous Memory Systems,” In Proceedings of EuroSys, no.15, pp. 1-16, 2016.
[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.
[17] H. Volos, G. Magalhaes, L. Cherkasova and J. Li, “Quartz: A Lightweight Performance Emulator for Persistent Memory Software,” In Proceedings of the 16th Annual Middleware Conference, pp. 37-49, 2015
[18] K. K. Pusukuri, “Working Set Model for Multithreaded Programs,” In Proceedings of the 2014 International Conference on Timely Results in Operating Systems (TRIOS'14), pp. 1-1, 2014.
[19] 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, 2018.
[20] X. Xiang, B. Bao, C. Ding, Y. Gao, “Linear-time Modeling of Program Working Set in Shared Cache,” In Proceedings of PACT, pp. 350-360, 2011.
[21] P. J. Denning, “The working set model for program behavior,” In Communications of the ACM, vol.11, no.5, p.323-333, 1968.
[22] T. Hastie, R. Tibshirani, J. H. Friedman, “The Elements of Statistical Learning: Data Mining, Inference, and Prediction,” In Springer Series in Statistics, 2nd ed, 2009.
[23] T. Hirofuchi, R. Takano, “RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems,” In Proceedings of ACM Symposium on Cloud Computing (SoCC), pp. 112-125, 2016.
[24] D. Gove, “CPU2006 working set size,” In ACM SIGARCH Computer Architecture News archive, News 35, pp. 90-96, 2007.
[25] D. Sengupta, Q. Wang, H. Volos, L. Cherkasova, J. Li, G. Magalhaes, K. Schwan, “A Framework for Emulating Non-volatile Memory Systems with Different Performance Characteristics,” In Proceedings of ACM/SPEC ICPE, pp. 317-320, 2015.