Vol.2020-ARC-241 No.2
2020/7/30

複合型メインメモリの メタデータ管理のためのデータアクセス解析

塚田　竣介[1,a)]　佐藤　雅之[1,b)]　高屋敷　光[1,c)]　小松　一彦[1,d)]　小林　広明[1,e)]

概要：複合型メインメモリは異なる種類のメモリを組み合わせることにより，高バンド幅と大容量を両立できることが期待されている．複合型メインメモリが性能を発揮するためには，データを各メモリに適切に配置することが重要である．また，アクセス状況に合わせて，配置したデータを他のメモリへ移動する再配置といった管理も重要である．そのためにはデータの位置をどこに配置，または再配置したか正確に把握する必要があり，元の位置と再配置後の位置を記録したメタデータを用いる管理が必要となる．しかしながら，複合型メインメモリの容量増大，及びより高度なデータ管理に必要な追加情報のために，メタデータの総量増加，およびそれにともなう管理のオーバヘッドの増加が問題となる．本報告では，メタデータ量が増えても性能低下を抑えるデータ管理方式の実現に向けて，データ管理単位に基づくメタデータアクセスの分析を行う．ここで，データ管理単位とは，複合型メインメモリ内でデータを再配置する際の最小単位であり，1 つのメタデータが付随する．さらに，分析結果に基づいて，メタデータ量の増大に対応可能な複合型メモリの管理方法について議論する．

1. はじめに

近年，プロセッサは複数のコアを搭載することで演算性能を高めている．しかし，バンド幅やレイテンシなどのメインメモリのデータ転送性能はプロセッサの性能に追いついていないため，プロセッサとメインメモリの間には深刻な性能差が生じている．そのため，メインメモリにはより一層のデータ転送性能向上が求められる．また，メインメモリに求められているのは性能の向上だけではない．ビッグデータ解析や，機械学習，高性能計算などの発展・大規模化により，大きなデータサイズを扱うデータ容量指向アプリケーションが増加し，より大きなメモリ容量の実現も求められる．このように，メインメモリには高性能化と大容量化の 2 つを実現することが求められる．

しかし，近年開発が急速に進んでいる高いバンド幅を持つ HBM [1] や容量が大きい不揮発メモリ [2] といったメモリのみでは，高性能化と大容量化を両立することができない．そこで図 1 に示されるような，複数のメモリを組み合わせる技術である複合型メインメモリが注目されている．

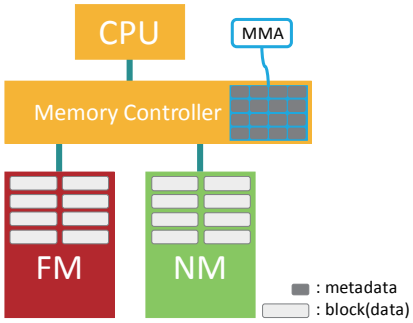

図 1　従来の複合型メインメモリ．

複合型メインメモリとは，高性能かつ小容量のメモリモジュール（Near Memory, NM）と低性能かつ大容量のメモリモジュール（Far Memory, FM）を組み合わせたメインメモリである．NM と FM を組み合わせることで，NM に足りない容量を大容量の FM で補い，FM では不十分なデータ転送性能を NM で補う．これによって，複合型メインメモリは，高性能と大容量の両立を実現できるとして期待されている．

しかし，複合型メインメモリを用いて高い性能を実現するためには，NM と FM のメモリアクセス特性を考慮したデータ管理が必要となる．具体的には，今後アクセスされる可能性の高いデータを NM へ，アクセスされる可能性の低いデータを FM へ配置するような管理が必要である．こ

1　現在，東北大学
Presently with Tohoku University
a)　shunsuke.tsukada.p1@dc.tohoku.ac.jp
b)　masa@tohoku.ac.jp
c)　hikaru.takayashiki.p5@dc.tohoku.ac.jp
d)　komatsu@tohoku.ac.jp
e)　koba@tohoku.ac.jp

ⓒ 2020 Information Processing Society of Japan

れは，アクセスされる可能性の高いデータを低性能な FM に保存し続けることは，パフォーマンス悪化の原因になり得るためである．また，データへのアクセス状況は時間とともに変化し続けるため，アクセス状況に基づいたデータの再配置も必要である．

このようなデータ管理には，管理用データ（以降，メタデータと呼ぶ）が必要である．複合型メインメモリでのメタデータとは，主にデータの再配置前と後の位置情報を記録したものであり，データアクセスの際に必ず参照される．これまでの研究でもメタデータを使ったデータ管理手法が提案されている [3]．しかしながら，さらなる複合型メモリの大容量化が予想される将来においてはメタデータが増大してしまい，メタデータを保存するためのオンチップ面積やそれに伴う消費電力の増大が問題となる．これらの問題を解決しながらメタデータをどのように管理するかが今後の重要な課題となる．

そこで本研究では，今後のメタデータ量の増大に対応可能な複合型メインメモリの管理方法について議論する．また，複合型メインメモリで想定されるデータ管理単位に基づくメタデータアクセスパターンの調査を行い，メタデータ管理に有用な知見を明らかにする．

本報告の構成は以下のとおりである．第 1 節では，本報告の概要を述べた．第 2 節では，複合型メインメモリとメタデータについて説明し，さらに問題点について述べる．第 3 節では，関連研究について述べる．第 4 節では，本研究で想定する複合型メインメモリについて述べる．第 5 節では，データ管理方式の実現に向けて，データ管理単位に基づくメタデータアクセスの分析を行った結果と考察を述べる．第 6 節はまとめである．

2. 複合型メインメモリ

2.1 複合型メモリシステムにおけるメタデータ管理

複合型メモリ上で再配置されたデータが再配置後にも正確にアクセスできるようにするには，再配置前・再配置後のアドレス関係を記録しているメタデータが必要である．従来の複合型メインメモリは全てのメタデータをメモリコントローラ内で管理している．図 1 に具体例を示す．メモリコントローラはメタデータを保存するためのメモリアレイを有する．本報告ではこのメモリアレイを Metadata Memory Array (MMA) と呼ぶ．さらに，メタデータはメタデータ管理機構によって管理される．

複合型メインメモリにおいて，実際のデータにアクセスする流れは以下の通りである．まず，初めに CPU からメモリコントローラにデータへのアクセス要求が来て，アクセス要求を受け取ったメモリコントローラが MMA 内に保存したメタデータを参照する．参照したことでデータが NM または FM のどちらに保存されているかが分かるため，それをもとに NM または FM 内のデータにアクセスする．

2.2 複合型メインメモリの容量増大化に伴うメタデータ増加の問題

近年，ビックデータ解析や，機械学習，高性能計算などの発展により，大きなデータサイズを扱うデータ容量指向アプリケーションが増加し，メインメモリにはより大きな容量が求められている．そのため，今後さらに複合型メインメモリに要求される容量が大きくなると予想される．さらに複合型メインメモリの容量が増大化するに伴って，メタデータも一緒に増加することが考えられる．

今後，メタデータがどの程度増加するかを知るために，複合型メインメモリの容量とメタデータの総量の関係について試算を行う．複合型メインメモリのメモリ容量を S_{memory} [Bytes] とし，データ管理単位のサイズを S_{data} [Bytes] とする．

これらの条件から，すべてのメタデータを格納するのに必要なメモリアレイの容量 S_{MMA} [Bytes] を計算する．まず，メタデータの総数 N_{meta} [個] は，メモリ容量 S_{memory} [Bytes] と S_{data} [Bytes] で式 (1) のように表される．

$$\boldsymbol{N_{meta} = \frac{S_{memory}}{S_{data}}}. \tag{1}$$

次に，メタデータのサイズ S_{meta} [Bytes] は，N_{meta} [個] を用いることで式 (2) のように表される．

$$\begin{aligned}\boldsymbol{S_{meta}} &= \boldsymbol{2 \times \log_2 N_{meta}[bits]} \\ &= \boldsymbol{2 \times \frac{1}{8} \log_2 N_{meta}[Bytes]}.\end{aligned} \tag{2}$$

ここで，$\log_2 N_{meta}$ はアドレス表現に必要なビット数を表している．また，メタデータには再配置前と後のアドレスが含まれるので 2 を掛けている．

以上より，必要なメモリアレイの容量 S_{MMA} [Bytes] は，メタデータの総数 N_{meta} [個] とメタデータサイズ S_{meta} [Bytes] を用いることで式 (3) のように表される．

$$\begin{aligned}\boldsymbol{S_{MMA}} &= \boldsymbol{N_{meta} \times S_{meta}} \\ &= \boldsymbol{N_{meta} \times \frac{2}{8} \log_2 N_{meta}} \\ &= \boldsymbol{\frac{S_{memory}}{S_{data}} \times \frac{1}{4} \log_2 \frac{S_{memory}}{S_{data}}}.\end{aligned} \tag{3}$$

ここで，式 (3) を用いて，図 2 にメインメモリの容量 S_{memory} [Bytes] とメタデータのメモリ容量 S_{MMA} [Bytes] の関係を示す．図 2 のグラフについて，データ管理単位のデータサイズが $S_{data} = 4$ [KB] の場合を青色のグラフとし，また $S_{data} = 8$ [KB] の場合をオレンジ色のグラフ，さらに $S_{data} = 16$ [KB] の場合を緑色のグラフとして示す．図 2 から，$S_{data} = 4, 8, 16$ [KB] のどの場合でも，複合型メインメモリが大容量化するほどメタデータの総量は，ほぼ比例して大きくなることが分かる．ここで，ブロックサイズが 4KB のグラフを見ると，S_{memory} が 512GB の場合，

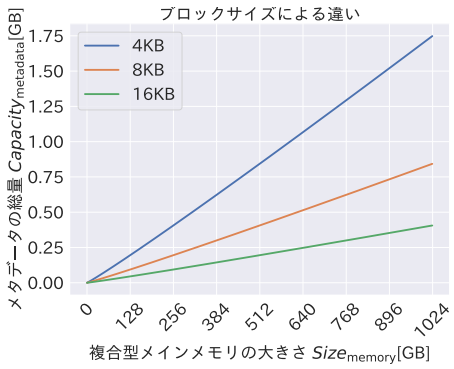

図 2 メインメモリの大きさとメタデータ総量の関係.

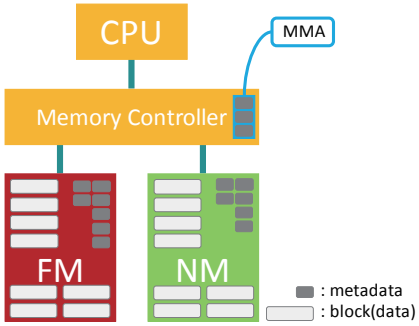

図 3 本研究の複合型メインメモリ.

メタデータの総量は 800MB を超えることが分かる．また，ブロックサイズが 16KB のグラフを見ると，S_{memory} が 512GB の場合，メタデータの総量は 200MB を超えることが分かる．したがって，現在のプロセッサのキャッシュ総容量をはるかに超えるメタデータ量となり，オンチップ上にすべてのメタデータを載せることは現実的ではないと考えらえる．

また図 2 から，ブロックサイズが 4 [KB], 8 [KB], 16 [KB] の順に大きくなるほど，グラフの傾きが小さくなることが分かる．このことからデータ管理単位のサイズ S_{data} [Bytes] を大きくするほど，メタデータ量の増加が軽減することが分かる．また，ブロックサイズを小さく 4 [KB] から，8 [KB] または 16 [KB] にすると，メタデータの総量を 2 分の 1 または 4 分の 1 にすることができることが分かる．

3. 関連研究

これまでの研究では，NM をキャッシュとしながら FM をメインメモリとして用いる研究と，NM と FM を同一のメモリアドレス空間で扱う研究が多く行われてきた．前者の先行研究の一例として，Chou らの CAMEO [4] が挙げられる．CAMEO では，データが再配置される場所を限定させることでメタデータのデータサイズを小さくし，さらにメインメモリ内のデータと共存させることでハードウェアにおけるオーバヘッドの増加を避けている．また，後者の先行研究の一例として，Yu らの Banshee [6] が挙げられる．Banshee では，TLB(Translation lookaside buffer) や PTE(Page table entry) と協調して，OS のページ単位でデータを管理する．メモリコントロール内の Tag buffer に最近再配置されたブロックのメタデータを保持し，Tag buffer 内がいっぱいになるタイミングで TLB と PTE に再配置したアドレスを知らせる．これにより，メタデータの増加を防ぎ，性能の向上を図っている..

そして，Vasilakis らの研究 [3] では，両者の機能を合わせた複合型メインメモリとして，Hybrid^2 が提案されている．Hybrid^2 は，NM の一部をキャッシュ領域として残りの領域を FM と同一アドレス空間で用いる新しい複合型メモリシステムである．Hybrid^2 では，NM キャッシュの管理に利用するタグと再配置の管理に利用するメタデータを，共通のメタデータを用いて管理することでメタデータの総量増加を軽減し，メタデータ管理を効率的にしている．さらに Hybrid^2 には，NM キャッシュ中にあるメタデータを保存する eXtended Tag Array(XTA) がある．XTA は，NM 中に保存している再配置前後の位置関係を記録した Remap table のキャッシュのような振舞いをする．さらに，Hybrid^2 では 512KB を超えない大きさが想定されている．一方で，Hybrid^2 には，必要なメタデータを予測し事前にプリフェッチする機構が実装されていないため，メタデータ参照に伴う性能低下を改善する余地が残されている．そこで，本研究では，Hybrid^2 では実装されなかったメタデータ予測機構を設計・実装することで更なるパフォーマンス向上を目指す．

4. 複合型メインメモリのためのメタデータ管理機構の実現

4.1 メタデータ管理

本研究では，図 3 のようにメタデータを複合型メインメモリ自身に保存し，必要なメタデータのみをメモリコントローラ内の MMA に保存するような管理機構を提案する．必要なメタデータのみを MMA に保存するので，メタデータの総量が増加してもメタデータを保存する MMA の容量を小さくできると考えられる．また，MMA が小さいので，MMA のハードウェアオーバーヘッドと消費電力を大きく削減することができると考えている．

しかし，MMA が小さいと，参照したいメタデータが MMA 内にない場合に，メインメモリ内のメタデータへアクセスする必要がある．このとき，NM や FM のレイテンシは MMA のレイテンシよりも大きいため，性能低下は避けられない．したがって，事前に必要なメタデータのみを格納しておけるようにメタデータを管理する必要がある．

4.2 メタデータアクセスパターンの分析

そこで本報告では，このような管理機能の実現を目指し

表 1　実験条件.

L1I キャッシュ	4ways, 64KB
L1D キャッシュ	4ways, 64KB
L2 キャッシュ	8ways, 512KB
L3 キャッシュ	16ways, 2MB
ニアメモリ	1GB
ファーメモリ	3GB

て，複合型メインメモリのデータ管理単位に基づき，メタデータアクセスパターンの調査を行う．シミュレータを使ってデータ管理単位のアドレストレースを取得し，データ管理単位ごとのメタデータアクセス回数を求める．

メタデータアクセス調査のポイントは以下の 3 つである．まず，1 つ目は，単位時間当たりに必要なブロック数についてである．これについて調べることで，適切な MMA のサイズが明らかになる．2 つ目は，アクセス回数についてである．これについて調べるため，ブロックごとのアクセス回数分布を求める．また，時間経過ごとに必要なブロックは異なるため，時間経過に沿ったアクセス回数を求める．そして 3 つ目は，アドレス差分についてである．ここで，アドレス差分を式 (4) のように定義する．

$$\text{アドレス差分} = \text{現在アクセスされたアドレス} - \text{直前にアクセスされたアドレス}. \tag{4}$$

アクセス回数以外の方法で必要なメタデータを予測することを目指して，アドレス差分を用いた予測の可能性について調べる．

5. 実験

5.1 実験環境

本研究で想定するメタデータ管理機構について，メタデータ管理に有用な知見を得るために，データ管理単位（以降，ブロック単位と呼ぶ）ごとのアドレストレースを取得する．評価環境として，NVMain と組み合わせた gem5 シミュレータを用いる．本シミュレータ上で複合型メインメモリを想定したシステムを構築し，それによりアドレストレースを取得する．表 1 は，1 コア当たりにおけるメモリ階層の詳細なパラメータについて示す．ここで，今回は再配置をはじめとしたデータ管理を行っておらず，メタデータの管理は行っていない．

ベンチマークとしては SPEC CPU 2000, 2006，2017 を用いる．それぞれのベンチマークは，ウォームアップに 10 億命令，評価にそれに続く 10 億命令実行した．またブロックサイズとして，4 KB，16 KB，64 KB，256 KB，1 MB，4 MB，16 MB，64 MB の 8 つのブロックサイズを想定した．以降，メモリ性能要求の多いベンチマーク 7 つ（2000.ammp, 2000.gzip, 2006.perlbench, 2017.deepsjeng, 2017.lbm，2017.mcf，2017.xz）とメモリ性能要求の少ないベンチマーク 1 つ（2006.sphinx3）を代表として結果を示し，ここで示せなかったベンチマークでも見られる特徴について議論する．

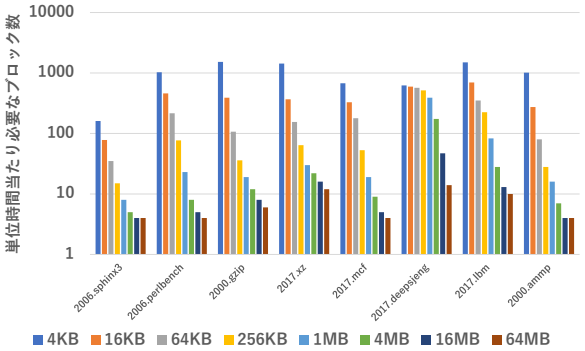

図 4　単位時間当たり必要なブロック数.

5.2 実験結果と考察

5.2.1 単位時間当たり必要なブロック数

まず，図 4 に，ベンチマークそれぞれについて，ブロックサイズごとの単位時間当たりの最大アクセス回数を示す．ここで，単位時間として 10000 クロックサイクルを想定し，ブロックサイズは 4KB，16KB，64KB，256KB，1MB，4MB，16MB，64MB を想定している．図 4 の横軸はベンチマークごとのブロックサイズを表し，縦軸は単位時間当たりのアクセス回数を対数表示で表している．図 4 から，すべてのベンチマークでブロックサイズが大きくなるほど単位時間あたりにアクセスされるブロック数が少なくなることが分かる．具体的に，ブロックサイズが 4KB の場合，ほとんどのベンチマークで必要なブロック数は 1600 個以下となっているが，ブロックサイズが 16KB の場合は，必要なブロック数が 700 個以下となっている．これは，ブロックサイズが大きいほど 1 ブロック分の領域が拡大し，その分単位時間当たりに必要なブロック数が減少するためであると考えられる．ここで，2006.sphinx3 のグラフが他のベンチマークよりも相対的に小さいのは，実行時間中のメインメモリへのアクセスが少なく，その分必要なブロック数も少なくなったためであると考えられる．

以上から，メタデータ管理の点で以下 2 つのことが分かった．1 つ目は，ブロックサイズを大きくすることで単位時間あたりに必要なメタデータの数を少なくすることができるということである．メモリコントロール内に保存できるメタデータの数は限られているため，単位時間当たりに必要なメタデータの数を減らせれば，結果的にその分を他のメタデータに割くことができる．2 つ目は MMA の更新頻度が 10000 クロックサイクルに 1 回以上であれば，MMA のサイズは 1600 エントリで十分であるということである．これは，全エントリ数の 0.15%になる．また，MMA

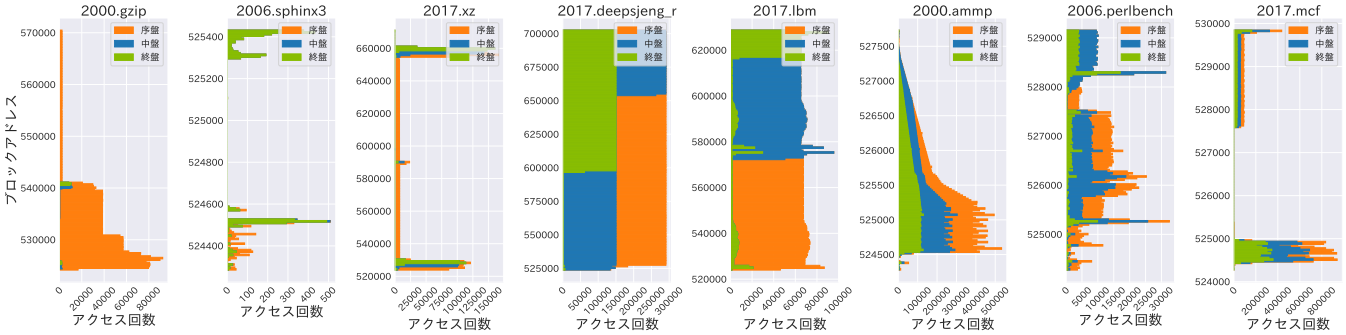

図 5 経過時間ごとのアクセス回数.

の更新頻度がそれより多くなれば，MMA のエントリ数はより少なくて済むと考えられる．

5.2.2 実行時間に沿ったアクセス回数

次に，図 5 に時間経過に沿ったアクセス回数の変化を示す．図 5 は，縦軸がブロックアドレスを表し，横軸はアクセス回数を表している．ここで，それぞれのベンチマークの経過時間を均等に 3 つに分け，3 つの時間帯ごとのアクセス回数を色で区別して示す．それぞれの色については，オレンジ色が経過時間序盤にアクセスされた回数で，青色が経過時間中盤にアクセスされた回数，そして緑色が経過時間終盤にアクセスされた回数を表している．

図 5 から，一部のベンチマーク（2006.sphinx3 や 2000.gzip, 2017.xz, 2017.deepsjeng, 2017.lbm）は，経過時間序盤，経過時間中盤，経過時間終盤で頻繁にアクセスされるアドレスが変化しており，実行時間帯によってアクセスされやすいブロックが変わることが分かる．一方で，他のベンチマーク（2006.perlbench や 2017.mcf，2000.ammp）は経過時間に依らず，頻繁にアクセスする位置があまり変化していないことが分かる．

さらに，図 5 から，多くのベンチマークでブロックアドレスによってアクセス回数に偏りがあることが分かる．図 5 の 2006.perlbench の 528200，525200 付近のアドレスや，2017.lbm の 522000，580000，625000 付近のアドレスのブロックは，周りと比べて突出してアクセスされている様子が確認できる．一方で，2006.sphinx3 の 524600 から 525200 の間，2000.gzip の 540000 から 570000 の間，2017.xz の 540000 から 640000 の間，そして 2017.mcf の 525000 から 527500 の間では，アクセスが極端に少ないことがそれぞれ確認できる．このことから，隣接するブロック同士でアクセス傾向が似ていることが分かる．

以上から，メタデータ管理の点で以下の 2 つのことが分かった．まず 1 つ目が，メモリコントローラ内に保存しておくべきメタデータとしては，その時点でアクセス頻度が高いものが良いということである．これは，ある時点でアクセスされたデータがその後もアクセスされるような時間的局所性のあるベンチマークでは，効果的な管理方法であると考えられる．また，メモリコントローラ内から追い出すべきメタデータとして，その時点でアクセスが無かったり，少ないものが良いということである．これは，追い出すべきメタデータを LRU と LFU を組み合わせた LRFU(Least recently frequent used) [8] で十分に選択できることを示している．2 つ目は，あるブロックにおけるアクセス頻度を隣接するブロックへのアクセス回数から予測しやすいことである．このことを利用すれば，アクセス回数を記録する場合に，すべてのブロックのアクセス回数を記録する必要はないと考えられる．したがって，一定のアドレス間隔に代表ブロックを選び，アクセス回数を記録すれば，おおよそのアクセス数分布を知ることができると考えられる．

5.2.3 アドレス差分

ここでは，直前と直後にアクセスされるブロックアドレスの特徴について調査するため，アドレス差分について調べる．図 6 に，それぞれのベンチマークを実行したときのアドレス差分の分布を示す．図 6 は，縦軸がアドレス差分を表していて，横軸がそのアドレス差でアクセスした回数を表している．図 6 から，すべてのベンチマークについて，アドレス差分 0 やそれに近い値でアクセスする頻度が高いことが分かる．具体的に，アドレス差分の絶対値が 200 以下の割合は，2006.sphinx3 が 40.3%，2006.perlbench が 49.9%，2000.gzip が 42.8%，2017.xz が 63.9%，2017.mcf が 11.0%，2017.deepsjeng が 99.6%，2017.lbm が 95.2%，2000.ammp が 97.8%である．2017.deepsjeng，2017.lbm，2000.ammp は，全アクセスの 90%以上でアドレス差が 200 以下の位置にあるブロックにアクセスをするので，空間的局所性が非常に高いと言える．従って，現在アクセスされているメタデータのアドレス付近のメタデータを MMA にプリフェッチすることが有効であると考えられる．一方，2017.mcf では，全アクセスの 11%のみでアドレス差が 200 以下の位置にあるブロックにアクセスをする．そのため，他のベンチマークと比べると空間的局所性が低い．ここで，同じような傾向が 3 割近くのベンチマークで見られた．また，このようなベンチマークを実行した場合，アドレス差の小さい位置だけを保存する方法では，メタデータの予測精度を低くすると考えられる．このような場合に予

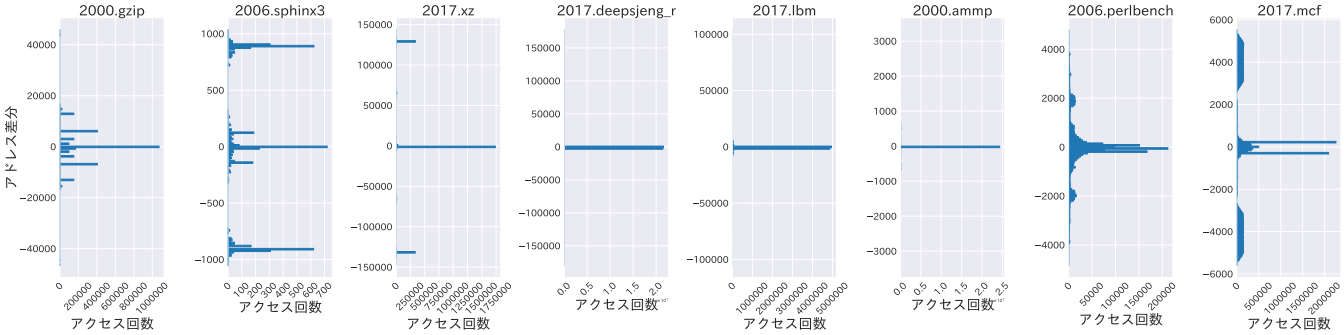

図 6 アドレス差分.

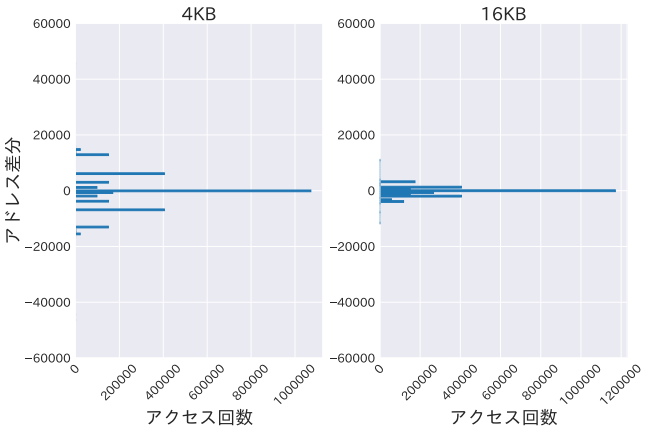

図 7 ブロックサイズが 4KB と 16KB の場合のアドレス差分.

測精度を改善するためには，アドレス差分が 0 に近い位置だけではなく，それ以外の差分にあるメタデータも積極的に MMA に保存する必要があると考えられる．

ここで，ブロックサイズが異なる場合のアドレス差分について，2000.gzip を例に見ていく．図 7 にブロックサイズが 4KB の場合と 16KB の場合についてそれぞれ示す．図 7 から，ブロックサイズが 4KB の場合，アドレス差分の絶対値が 5000 以下のアクセス回数は全体のアクセス回数の約 60%である．一方で，ブロックサイズが 16KB の場合，アドレス差分の絶対値が 5000 以下のアクセス回数は全体のアクセス回数の約 99%である．従って，ブロックサイズが 16KB の場合は，4KB の場合と比較して，ブロックにおける空間的な局所性が高いと言える．このことから，空間的な局所性が高い 16KB の方が，メタデータ予測の精度を高めやすいことが分かる．

以上のことから，メタデータ管理の点で以下の 3 つのことが分かった．まず 1 つ目は，メモリコントローラ内に保存するメタデータとして，現在アクセスされているアドレスから一定の範囲内にあるメタデータを保存するのが適切ということである．空間的局所性が強く見られるようなベンチマークを実行した際に，効率的にメタデータを管理することができると考えらえる．2 つ目は，メモリコントローラ内に保存するメタデータとして，アドレス差分に注目した際，アクセスが集中しているような差分の位置にあるメタデータの保存も有効ということである．これは，過去のアドレス差分を参考にすることで，アクセスが集中しているような位置に存在するメタデータは，今後もアクセスが集中すると判断できるからである．そして 3 つ目は，ブロックサイズが大きい方がメタデータの予測精度を高めやすいということである．これは，ブロックサイズが大きいほど空間的局所性が高くなるためである．また，このとき，同じメタデータへのアクセス回数も増えるため，メタデータの再利用性を高めることもできる．

5.3 メタデータ予測機構の設計

今回得られた知見でメタデータ管理機構の設計に役立つものは，以下の 3 つである．まず 1 つ目が，MMA のサイズとしておよそ 1000 エントリから 2000 エントリあれば十分であるという知見である．2 つ目は，アクセス頻度の高いデータ付近はその後もアクセス頻度が高く，逆にアクセス頻度の低いデータ付近はその後もアクセス頻度が低いという知見である．3 つ目は，アドレス差分を用いて次にアクセスされる可能性の高いブロックを予測できるという知見である．以上の 3 つの知見を用いることで，メタデータを予測し MMA に格納する際のアルゴリズムを設計する．

ここで，今回得られた知見に基づいて設計したメタデータ管理機構のアルゴリズムを示す．このメタデータ管理機構によるメタデータ予測は，MMA ミスが発生したときに以下の順で行われる．

(1) メタデータを取得するために複合型メインメモリにアクセスする．
(2) 取得するメタデータは，過去のアドレス差分を参考にすることで，現在アクセスしているアドレス近くのもの，もしくは頻繁にアクセス頻度の高いアドレス差に位置するものに決める．ここで，取得するメタデータの量は，性能に影響を与えるため，バンド幅の余裕を考慮しながら決める必要がある．
(3) もし MMA に空きが無ければ，現在アクセスが少ないブロックやアクセス頻度の少ないアドレス差に位置するブロックのメタデータが今後もアクセスが少ないと考えられるので，これらのメタデータを優先して排除

する．

また，MMA ヒットの場合はアクセスをカウントするだけで，メタデータの取得などは行わない．このような一連のメタデータ予測の流れをアプリケーション実行中に繰り返し行う．

本メタデータ管理機構では，必要なメタデータをアクセス回数という指標だけでなく，アドレス差分というもう一つの指標も用いるため，予測精度を高められると考えている．

6. おわりに

高いデータ転送性能と大容量を両立すると期待される複合型メインメモリは，容量の増大に伴い MMA が大きくなってしまう問題を抱えている．そこで本研究では，MMA の増大に対応した効率的なメモリ管理方式の実現を目指し，まずメタデータアクセスパターンの調査を行った．アクセスパターンの調査結果より，単位時間当たりに必要なブロック数，ブロックごとのアクセス回数，アドレス差分の 3 つについての知見を得ることができた．

また，これら知見に基づいて，メタデータを予測し MMA に格納するアルゴリズムの概念設計を行った．今後の課題としては，ここで得られた設計を詳細化し，実際にメタデータ管理機構を実装・評価することが挙げられる．

謝辞 本研究の一部は，文部科学省「次世代領域研究開発」（高性能汎用計算機高度利用事業費補助金）量子アニーリングアシスト型次世代スーパーコンピューティング基盤の開発，科研費若手研究（課題番号：19K20232）の一環として実施したものである．

参考文献

[1] Gabriel H Loh, Yuan Xie, and Bryan Black. Processor design in 3D die-stacking technologies. *Ieee Micro*, Vol. 27, No. 3, pp. 31–48, 2007.

[2] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *Proceedings of the 36th annual international symposium on Computer architecture*, pp. 2–13, 2009.

[3] Hybrid2: Combining Caching and Migration in Hybrid Memory Systems, author=Vasilakis, Evangelos and Papaefstathiou, Vassilis and Trancoso, Pedro and Sourdis, Ioannis. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 649–662. IEEE, 2020.

[4] Chia Chen Chou, et al. Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache. In *Proc. of Micro*, pp. 1–12, 2014.

[5] Mitesh R Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H Loh. Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pp. 126–136. IEEE, 2015.

[6] Xiangyao Yu, Christopher J Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas. Banshee: Bandwidth-efficient dram caching via software/hardware cooperation. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–14. IEEE, 2017.

[7] Djordje Jevdjic, Gabriel H Loh, Cansu Kaynak, and Babak Falsafi. Unison cache: A scalable and effective die-stacked DRAM cache. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 25–37. IEEE, 2014.

[8] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE transactions on Computers*, No. 12, pp. 1352–1361, 2001.

ⓒ 2020 Information Processing Society of Japan