Vol.2015-OS-134 No.5
2015/8/4

DMA転送におけるVM技法の提案

堀 敦史[1,a)]　佐藤 未来子[2,b)]　島田 明男[3,c)]　並木 美太郎[2,d)]　石川 裕[1,e)]

概要：近年，CPU とメモリの速度の乖離が従来に増して大きな問題となっており，より階層の深いメモリシステムや，異なる特性を持つメモリを複数持つようなシステムが提案されている．この時，近い，遠い，といった異なる特性のメモリ間で内容をコピーする必要が生じる．本稿は，DMA と OS カーネルの仮想メモリ管理を組み合わせることで，低遅延かつ高バンド幅で，遠いメモリの内容を近いメモリにコピーする手法を提案する．この手法は “Overshadow” と名付けられ，プロトタイプによる予備評価の結果，従来方式の DMA を用いてコピーする場合より遅延時間を 38%に短縮するなど，従来方式の DMA 転送に比べ高い性能を確認することができた．

Novel VM technique utlizing DMA

Atsuhi Hori[1,a)]　Mikiko Sato[2,b)]　Akio Shimada[3,c)]　Mitaro Namiki[2,d)]　Yutaka Ishikawa[1,e)]

Abstract: The gap between CPU speed and memory speed becomes more crucial than ever. To attack this problem, deeper hierarchical memory systems and/or heterogeneous memory systems where memory subsystems having different characteristics coexist are being proposed. In such systems, there are many situations where memory contents are copied between different memory subsystems. This paper proposes a new VM technique to utilize DMA, named “Overshadow,” which enables low-latency and high bandwidth memory copying. Evaluation result of the Overshadow prototype shows that the Overshadow latency is 38% of the DMA latency. Thus, Overshadow can outperform naive DMA transfer.

1. 背景

エクサスケール規模のシステムの実現に向けて，メモリシステムはより複雑になるとだけでなく，より深い階層を持つと考えられている [3]． これは，主に，CPU の速度向上に見合うメモリのバンド幅の向上が実現されていないことに起因する．一方，メモリシステムが消費する電力においても一桁以上の省エネが必要とされている [8]．

現在広く使われているサーバーにおいて，NUMA メモリシステムが用いられているが，より一層の消費電力性能比が要求される分野においては，GP-GPU などの加速器をサーバー機に組み込んだハイブリッドアーキテクチャ構成が現在注目を集めている（例えば [2]）．

このような技術の現状と，将来のコンピュータアーキテクチャに対する要求から，メモリシステムは 2 つに分類されると考えられる．ひとつは，異なる特性を持つメモリサブシステムをより深く階層化したもの（図 1），もうひとつは，特性が異なる複数の CPU とメモリサブシステムの対が存在するようなハイブリッドなメモリシステム（図 2），である．本稿では，これら 2 つのメモリシステムを総称して「複雑なメモリシステム」(complex memory system）と呼ぶものとする．

Intel 社は，現行の Xeon Phi の後継機（Knights Landing，以下 “KNL”）では，“on-package” メモリと “platform” メモリと呼ばれる 2 種類のメモリサブシステムを搭載する予定であると発表している [13]．On-package メモリは積

[1] 理研 AICS
RIKEN AICS
[2] 東京農工大学
Tokyo University of Agriculture and Technology
[3] 日立
Hitachi Ltd.
[a)] ahori@riken.jp
[b)] mikiko@namikilab.tuat.ac.jp
[c)] akio.shimada.ht@hitachi.com
[d)] namiki@cc.tuat.ac.jp
[e)] yutaka.ishikawa@riken.jp

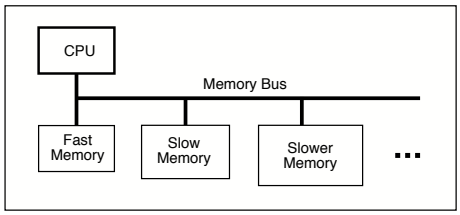

図 1 深いメモリ階層

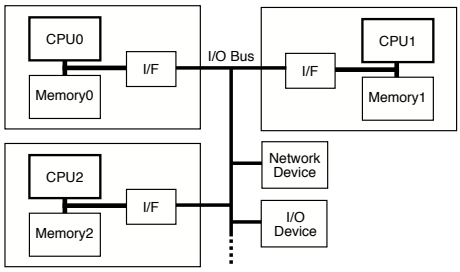

図 2 ヘテロなアーキテクチャ

層メモリであり，DDR4 の 5 倍のバンド幅を持つとされている．この on-package メモリには，platform メモリの last-level キャッシュ，あるいは，“flat”，つまりメインメモリの一部となる，の 2 つのモードが存在し，BIOS パラメータの設定によりモードの選択が可能になる．特に後者の場合，使う物理ページによりアクセス性能が異なるため，メモリページの割当に際しその点に注意する必要があるだけでなく，状況に応じて異なるメモリ間でデータを移動する必要も生じる．

以上記したように，既にハイブリッドなメモリシステムは広く使われており，より深い階層のメモリシステムの登場も間近である．一方で，これらの複雑なメモリシステムに対応するためのシステムソフトウェアの研究は始まったばかりであり，十分に研究されたとは言い難い状況である．本論文は，複雑なメモリシステムを有効に活用するための，新たな OS カーネルの機能を提案する．本稿のアイデアについては文献 [17] で既に発表されているが，本稿は，プロトタイプ実装による基礎評価結果を示し，本稿の手法の応用についての議論を深める．

次章において，本研究を始めるに至った動機を記し，第 3 章で本研究で提案する VM 技法について提案する．提案された手法の基礎的な評価結果は第 4 章で示す．以降，議論（第 5 章），関連研究（第 6 章），まとめと今後について（第 7 章）と続く．

2. 動機

2.1 PVAS と M-PVAS

我々は，科学技術振興機構 (JST) の戦略的創造研究推進事業「CREST」における研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」において「メニーコア混在型並列計算機用基盤ソフトウェア」の

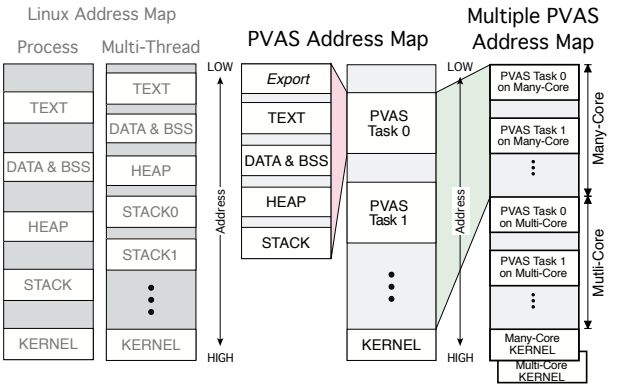

図 3 PVAS と MPVAS のメモリマップの例

研究を推進している．本研究において，メニーコア CPU 上で効率的な並列実行環境を提供する “Partitioned Virtual Address Space (PVAS)”[15], [16], [18]，および，PVAS をメニーコアとマルチコアから成るヘテロ構成のアーキテクチャに向けて拡張した “Multiple PVAS”[14] をそれぞれ提案している．

PVAS とは，従来のプロセスとスレッドの中間的な新しいタスクモデルである．それぞれの PVAS タスクは，プロセス同様，TEXT，DATA，BSS，STACK，HEAP などのセグメントを持ち，任意のプログラムが PVAS タスクとして実行可能である．また同じ PVAS アドレス空間には複数の PVAS タスクが存在し，それぞれの PVAS タスクは他の PVAS タスクに無関係に独立したプログラムを実行可能である．これは PVAS タスクのプロセス的な側面である．一方，同じ PVAS アドレス空間に存在する，ある PVAS タスクは他の PVAS タスクのメモリをアクセス可能である．これは PVAS タスクの，同じアドレス空間を共有するスレッド的な側面である．

M-PVAS は，ハイブリッドなアーキテクチャ，ここではメニーコア CPU とマルチコア CPU，用に PVAS を拡張したものである．アドレス空間は，それぞれの CPU の数に分割され，その分割されたアドレス空間のぞれぞれは，それぞれの CPU における PVAS となる．図 3　の左側に，従来のプロセスとスレッドのメモリマップの例，右側に PVAS と M-PVAS のアドレスマップの例を示す．

M-PVAS において，メニーコア CPU のメモリとマルチコア CPU のメモリは，互いに普通の load/store 命令によりアクセス可能なことを前提にしている．具体的には，メニーコア CPU として Intel 社の Xeon Phi[5]，マルチコアの CPU として x86_64 を想定しているが，特に限定するものではない．ここでメニーコア，マルチコアのそれぞれが独立したメモリシステムを持ち，それらが PCIe の I/O バスで結合されている（図 2 の構成）．PCIe 越しに互いのメモリをそれぞれの物理アドレス空間にマップすることができるため，互いのメモリを参照するが可能になっている（memory mapped I/O）．Xeon Phi は PCIe Gen2 規格

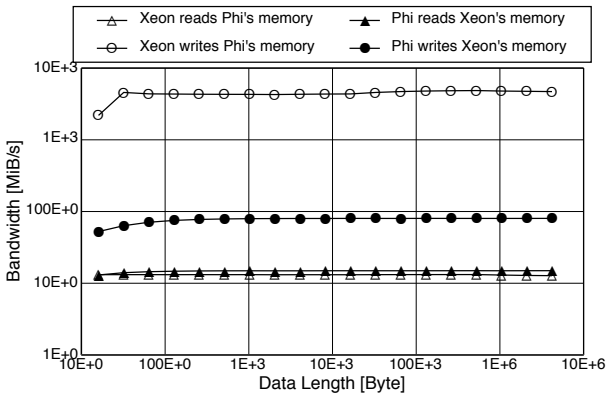

図 4 Bandwidth between Xeon and Xeon Phi

に準拠しており，理論性能は 8GiB/s のバンド幅となっている．

図 4 は，マルチコア CPU として Intel Xeon E5-2650（2.6GHz）と Intel Xeon Phi 5110P（Knights Corner，60 コア）で構成されるマシン上で，M-PVAS を動かし，互いにメモリをアクセスした時のバンド幅を示した結果である．このグラフから，Xeon から Xeon Phi への書込みは理論性能に近い性能が出ていることが分かる．しかしながら，相手のメモリを読込んだ場合，特に，Xeon Phi が Xeon のメモリを読む場合，理論性能の 1,000 分の一程度に相当する 10MiB/s 程度のバンド幅しか出せていないことが分かる．このことから，特に読込み性能の向上が重要と考えられる．そこで，本稿では，特にヘテロな環境における相手のメモリに対する読込み性能を向上させる方策として “Overshadow” と名付けた新しい VM 技法を提案する．

3. Overshadow

3.1 想定

本稿で提案する Overshadow は以下の 3 つの条件が満たされていることを想定している．

(1) 2 つ以上のメモリサブシステムが存在すること
(2) それぞれのメモリサブシステムがなんらかの方法により論理アドレスで互いに参照可能であること
(3) DMA エンジンを有し，メモリサブシステム間で DMA エンジンによるデータのコピーが可能であること

図 4 を計測したような環境，つまり，1) Xeon と Xeon Phi がそれぞれ独立したメモリサブシステムを持っていること，2) 互いのメモリサブシステムを PCIe 経由でアクセス可能であること，3) Xeon は IO-AT[4] と呼ばれる DMA エンジンを持ち，Xeon Phi も SCIF[5] API で Xeon Phi の DMA エンジンが利用可能であること，はこれらの条件を満たしている．

3.2 アイデア

複雑なメモリシステムにおいて，ある CPU から見た場合，遠い（遅い）メモリ参照と，近い（速い）メモリ参照

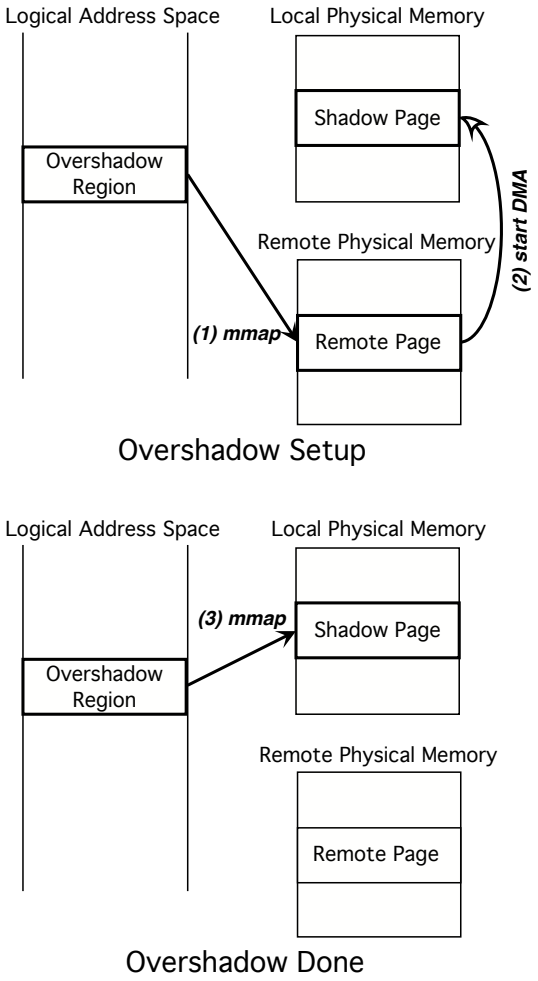

図 5 Overshaodw の基本動作

が存在することになる．本稿では，遠いメモリ参照の読込み速度を向上させることが目的である．もし，遠いメモリの参照したい領域を DMA を用いて近いメモリにコピーすれば，DMA 終了後は速いメモリ読込みが可能になる．しかしながら，DMA によるコピーでは，1) DMA エンジンの設定と起動，および，2) 割込による DMA 完了の同期が不可欠であり，DMA が完了するまで対象となるメモリ領域を参照することはできない．

Overshadow は，この DMA の完了までに待ち時間を有効に活用することを目的とする．遠いメモリは遅いながらも参照可能であるため，DMA が完了するまでの間は遠いメモリとしてそのまま参照し，DMA が完了したら近いメモリを参照することができれば，従来の DMA 転送の完了を待つまでの遅延時間を隠蔽可能である．

図 5 に Overshadow の動作を示す．読込み対象となる遠いメモリ領域が指定されたなら，そのメモリ領域を `mmap` し，アクセス可能とし，同時に遠いメモリの対象領域を近いメモリ領域にコピーすべく DMA を起動する．この時，近いメモリ領域はカーネル内部に確保し（図中，“Shadow Page”），この時点ではその近いメモリ領域はユーザプログラムからは見ることはできない（図 5 上）．DMA の起動が

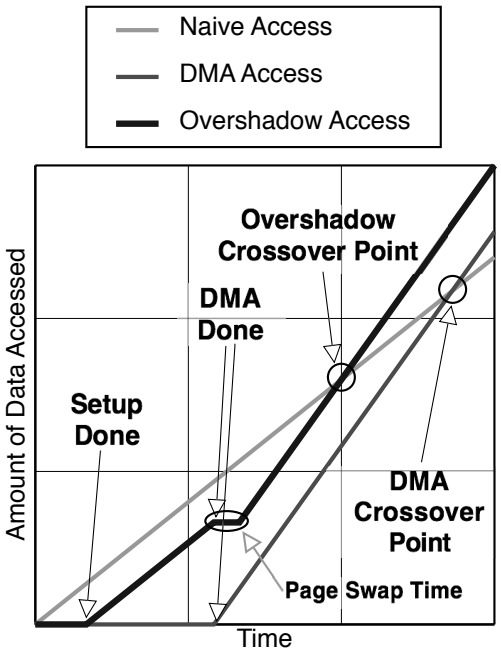

図 6 タイムチャート

完了した時点から，ユーザプログラムは遠いメモリ領域をアクセスすることが可能となる．

DMA によりコピーの完了を通知する割込により，カーネルは先に mmap していた遠いメモリ領域を munmap し，近い shadow ページ（ここは DMA により遠いメモリ領域と同じ内容となっている）に mmap する（(図 5 下)．これにより，ユーザプログラムは，同じ論理メモリ領域をアクセスしている間に，最初は遠い物理メモリを，DMA 転送が終了した時点で近い物理メモリを，参照することになる．この DMA 終了時の munmap および mmap はカーネル内でユーザプログラムとは非同期におこなわれ，ユーザプログラムは基本的にいつ切り替わったのかは分からない．

3.3 理論

図 6 に Overshadow の動作を時系列的に示す．この図には，遠いメモリをそのままアクセスした時の “Naive Access”，遠いメモリを DMA を使い近いメモリにコピーして，その後から近いメモリにアクセスする “DMA Access”（従来方式の DMA），そして，Overshadow を用いたときの “Overshadow Access” の 3 本の線が描かれている．横軸は時間で，縦軸はメモリを参照し何らかの処理をおこなったとしたときのメモリアクセス量を表す．もし処理が何もなければ（単にメモリからの読込みのみの場合），グラフ中の線の傾きは読込みバンド幅と等価になる．

遠いメモリをそのままアクセスする場合，バンド幅が低いため線の傾きは緩く，近いメモリのアクセスは，バンド幅が高く線の傾きも急になる．DMA で近いメモリにコピーした後のアクセスは急な傾きとなるが，DMA 転送が終了するまで待たなければならない．一方，Overshadow

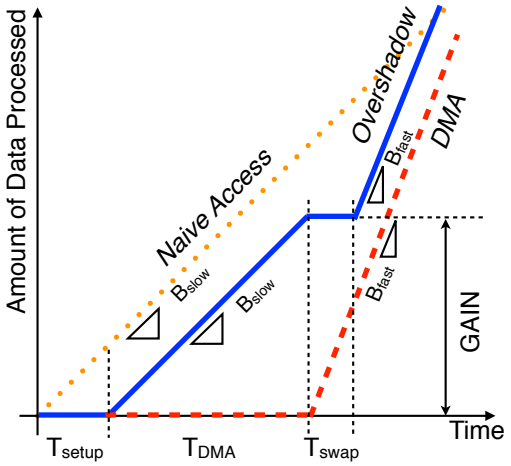

図 7 Overshadow の利得

では，DMA の設定が完了した時点から遠いメモリの傾きで処理を進めることが可能であり，DMA 完了後は近いメモリの傾きとなり，より高速に処理が進む．こうすることで，最初にアクセスできるまでの時間と（レイテンシ），メモリ領域全体をアクセスする時間（バンド幅）の双方に効果を期待できる．

従来方式 DMA に対する Overshadow の利得は，DMA 転送時間内に Overshadow 領域をアクセスする量である（図 7 の “GAIN”）．この図において，T_{setup} は DMA 転送のための DMA エンジンの設定に要する時間，T_{DMA} は Overshadow 領域全てを DMA 転送するのに要する時間，T_{swap} は Overshadow における shadow ページのメモリマップを切り替える時間，B_{slow} は遠い（遅い）メモリにアクセスするときのバンド幅（傾き），B_{fast} は近い（速い）メモリにアクセスするときのバンド幅である（傾き）．Overshadow 領域の大きさを S，DMA 転送のバンド幅を B_{DMA}，とすると，Overshadow の時刻 t におけるメモリアクセス量 $M(t)$ は以下のようになる．

- $0 \leq t \leq T_{setup}$ の時，
  $M(t) = 0$
- $T_{setup} < t \leq T_{setup} + T_{DMA}$ の時，
  $M(t) = B_{slow} \times (t - T_{setup})$
- $T_{setup} + T_{DMA} < t \leq T_{setup} + T_{DMA} + T_{swap}$ の時，
  $M(t) = B_{slow} \times T_{DMA}$
- $T_{setup} + T_{DMA} + T_{swap} < t$ の時，
  $M(t) = B_{slow} \times T_{DMA} + B_{fast} \times (t - (T_{setup} + T_{DMA} + T_{swap}))$

従来方式 DMA に対する Overshadow の利得（G）は，T_{DMA} 時間に B_{slow} でアクセスできるメモリ量になる．

$$G = T_{DMA} \times B_{slow} = \frac{S}{B_{DMA}} \times B_{slow}$$

従来方式 DMA と比較した場合の Overshadow の損失は，DMA 終了時に発生する shadow ページの切替時間 T_{swap}

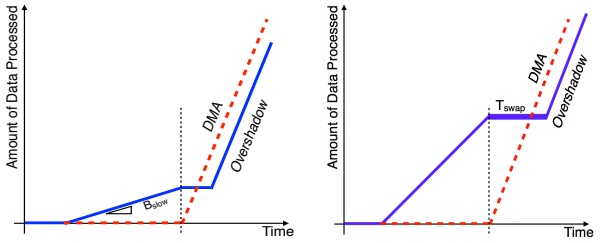

図 8 Overshadow が従来方式 DMA に対し有利でないケース

である．Overshadow が従来方式 DMA に対し有利なのは，利得がこの T_{swap} 時間内で近いメモリに高速にアクセスする量より大きくなければならない．すなわち，次の不等式を満たす必要がある．この式の左辺は Overshadow による利得（メモリ量）であり，右辺は T_{swap} 時間内で高速にアクセスするメモリ量である．

$$T_{DMA} \times B_{slow} \quad > \quad T_{swap} \times B_{fast}$$

一方，データ転送を開始してから最初にデータをアクセスできるまでの時間（レイテンシ，L）に関しては，

- 従来方式 DMA の場合

  $$L = T_{setup} + T_{DMA}$$

- Overshadow の場合

  $$L = T_{setup}$$

となり，Overshadow の方が従来方式 DMA に比べ有利である．

Overshadow の利得は，DMA 転送時間中に遠いメモリにアクセスできる量であり，損失は DMA 転送終了時に shadow ページを切り替えるコストである．このため，例えば，1) 遠いメモリのアクセスが非常に遅く，利得が十分に得られない場合（図 8 の左図），2) shadow ページの切替コストが大きく，利得を相殺するような場合（図 8 の右図），などが考えられる．

実際の DMA 転送は物理アドレスでおこなわれる関係から，データの転送はメモリページ単位となる．このため，複数ページにまたがるような Overshadow では，図 6 や図 7 で示したような性能曲線にはならない．

3.4 API

Overshadow には以下の関数が用意されている．

`overshadow_create(void *from,size_t sz,void **to)`
: `from` で指定された（遠い）メモリを `sz` で指定された長さ分，`*to` で返される（近い）メモリアドレスに Overshadow する．この関数呼出が終了すると，`*to` のアドレスをアクセス可能であるが，以下に示す 関数により Overshadow の完了を確認した後であれば，近いメモリでアクセ スすることが保証される．

`overshadow_wait(void *to)`
: `to` で指定された Overshadow 領域での Overshadow の完了を待つ．

`overshadow_test(void *to)`
: `to` で指定された Overshadow 領域での Overshadow の完了を調べる．

`overshadow_destroy(void *to)`
: `to` で指定された Overshadow 領域を廃棄する．

`*to` で示される Overshadow 領域は `mlock` などのシステムコールでピンダウンされていても，物理アドレスが Overshadow により変化するため，Overshadow が終了するまでの間，対象領域のピンダウンは無効となることに注意しなければならない．このため，Overshadow 領域をさらに (R)DMA しようとする場合は，ユーザの責任で Overshadow の完了後に (R)DMA がおこなわれるように保証する必要がある．

4. 予備評価

評価に用いた環境は，図 4 を計測したのと同じ，Intel Xeon E5-2650（2.6GHz）と Intel Xeon Phi 5110P（60 コア）で構成されるマシンである．この環境では，Xeon の持つ I/O AT DMA と Xeon Phi の持つ SCIF の DMA が存在するが，今回の予備評価では，SCIF の DMA を用いた．M-PVAS を用いて，Xeon と Xeon Phi が同じ論理アドレス空間を持つようにし，遠いメモリとして Xeon Phi のメモリを，近いメモリとして Xeon のメモリとなるように，Xeon 側から Overshadown を起動した．また比較のための従来方式の DMA 転送も同じ SCIF を用い，Xeon 側から起動している．

図 9 と図 10 に Overshadow と従来方式の DMA を用いた時の，対象メモリ領域（1MiB）を先頭から順にアクセスする時間（`rdtsc` 命令 によるクロック数）と，アクセスする量の関係を示したものである．Overshaow では，shadow ページを最初に確保する必要がある．調査の結果，この処理に 10^6 クロック以上も要していることが判明した．一方，従来方式 DMA では予め確保されたメモリ領域に DMA 転送をおこなっている．このため，公平さに観点から，Overshadow の計測では，shadow ページの確保に要した時間を差し引いてある．対象メモリ領域へのアクセスは，8 バイト単位で別なメモリ領域に `memcpy` することとし，これを全領域に対しおこなう．クロック数はこの繰り返しの都度計測している．これらの図では，Overshadow の効果が良く現れている繰り返し数が 30,000 までの部分を表示してある．

図 9 では，遅延（最初にデータアクセスを開始するまでの時間）では Overshadow が従来方式 DMA を上回ったものの，全てのデータをアクセスする時間という尺度では，Overshadow が従来方式 DMA の性能に及ばない結果

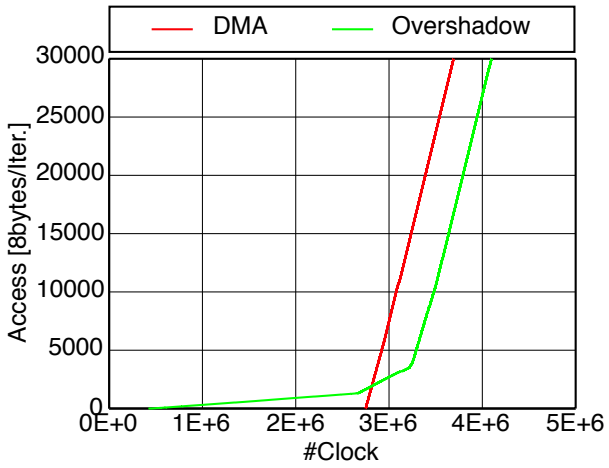

図 9 Overshadow と従来方式 DMA の比較（1）

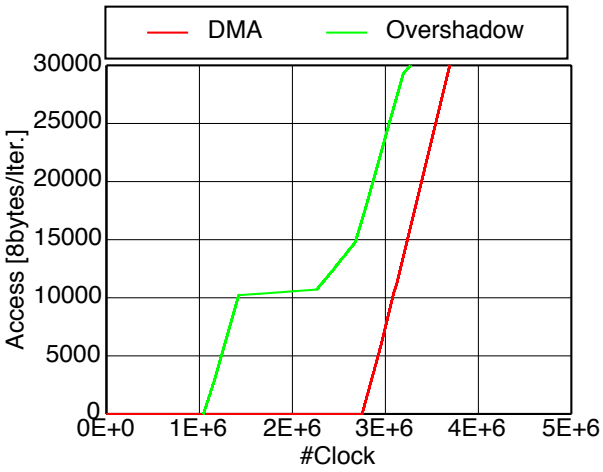

図 10 Overshadow と従来方式 DMA の比較（2）

となった．この原因のひとつとして，Xeon から PCIe 越しに Xeon Phi のメモリにアクセスするときのアクセス速度が遅い（この評価に用いたアクセス方式では 50 倍以上）ために，第 3.3 章で示したように，利得が十分に得られなかったことが一番の要因と考えられる．

従来方式 DMA を用いた場合では 2,748,444 クロックで DMA 転送が終了し，転送された領域に対しアクセスが可能になっているのに対し，Overshadow では 420,988 クロックからアクセスが開始され始めており，Overshadow の効果が確認できる．これは従来方式 DMA の場合の遅延時間において 15%に改善されたことになる．また，この時点から，およそ DMA が終了する時刻までは遅いメモリにアクセスしている様子が分かる．このことから，Overshadow と従来方式 DMA において，DMA エンジンの設定を含めた DMA 転送時間はほぼ同じと見ることができる．

しかしながら，従来方式 DMA の場合と異なり，Overshadow では DMA 転送の終了後に shadow ページを切り替える必要がある．この評価においてはは，全てのメモリページの DMA 転送が終了した時点から発生し，全てのページの切替が終了して初めて Overshadow が完了する．その後は，従来方式 DMA の場合と同様に，近いメモリへの高速なアクセスに移行しているのが分かる．

そこで，ページ単位の DMA 転送が終了するたびに shaodw ページを切り替えれば，そのページに対するアクセスは高速になるため，性能向上が期待できる．この方式で計測したものが図 10 である．Overshadow のアクセス開始時刻は 1,041,984 クロックと前よりも遅くなったものの，全ての DMA 転送が終了しなくても，DMA 転送が終わったページへのアクセスによる高速なメモリアクセスが観測され，結果的に従来方式 DMA の性能を上回ることを確認することができた（38%に改善）．

5. 議論

Overshadow 中の書込み

残念ながら，Overshadow 中に対象領域に書込みがあった場合の結果は保証されない．対象となるメモリ領域を read-only とすることで，誤った書込みを防ぐことができる．あるいは，Overshadow 中に該当ページで書込み禁止に対するページフォールトが発生した場合，Overshadow が終了するまでそのプロセスを中断させる方式も考えられる．

キャッシュという側面

近いメモリを遠いメモリのキャッシュとして使うのは自然な発想である．Overshadow の API は，非同期な DMA 転送として捉えることもできる．近いメモリが遠いメモリのキャッシュという使い方であった場合，Overshadow は近いメモリへの prefetch と考えることもできる．

コヒーレンシへの対応

ページ毎にどの CPU が書込み権利を持つかという情報を持たせることで，粒度が粗いもののある程度のコヒーレンシを保つことが可能である．この書込み権利は排他的であり，一度にひとつの CPU があるページの書込み権利を持つことができないとする．また，メモリの近い，遠いという概念が相対的，つまり，ある CPU からあるメモリをアクセスした時の速度と，違う CPU から同じメモリにアクセスした時の速度が異なるような場合，Overshadow により該当ページを CPU の近いメモリにコピーするケースが考えられる．この時，そのページの書込みの権利を排他的に得ることができたとする．さらに，Overshadow が完了した時点で，コピー元のページを破棄する．こうすることで，Overshadow 中は制限が生じるものの，複数の CPU が同じページの内容をコヒーレンシを維持したまま読み書きできることになる．

6. 関連研究

Linux では，NUMA 対応として，NUMA ノード間でページを移動させる `migrate_pages(2)`[6] をサポートしている．Linux は基本的に I/O 以外の DMA はサポートしていないが，この `migrate_pages(2)` に Overshadow を応用することは可能と考える．本稿における Overshadow では，データのコピーに DMA エンジンを用いているが，DMA エンジンの代わりにカーネル内で `memcpy` する実装もあり得る．Overshadow によりメモリページのコピーの終了を待つことなく，メモリのアクセスが可能になるという利点が期待できる．

Phadke らは，低レイテンシなメモリ，高バンド幅なメモリ，低消費電力なメモリの 3 種類のメモリをアプリケーションの特性に併せて使い分ける方式を提案している [12]．この提案では，メモリの要求時に，特性に合ったメモリを割り当てるが，例えば，プログラム実行の途中で，メモリに対する要求が変化する場合や，動的に特性の異なるメモリの確保や解放を繰り返すようなアプリケーションには対応が難しい．このため，アプリケーションの実行途中で，動的な変化に併せて，現在使用しているメモリから異なる特性のメモリへの移送も考える必要があり，Overshadow をこの移送に用いることが考えられる．

第 1 章でも述べたように，次期 Xeon Phi である KNL は，高速だが小容量な on-package メモリと低速だが大容量な platform メモリの 2 種類のメモリを持つとされている．Dong らはこのような 2 種類のメモリからなるアーキテクチャにおいて，詳細なシミュレーションベースの解析をおこなっており [1]，on-package メモリを Last-Level Cache（LLC）とするよりも，別なメモリとして扱った方が有利としている．この論文では，異なるメモリ間のデータの移動には専用のハードウェアがあり，このハードウェアが全自動で，あるいは OS の補助を得て，データをコピーするとしている．これに対し Overshadow は，OS カーネルが提供する機能として提案されており，どのように使うかはアプリケーション次第である．

複雑なメモリシステムは基本的にノード内に閉じた話であるが，例えば，旧 DEC 社の memory channel[7] ハードウェアを用いることで，他のノードのメモリをローカルに参照することが可能になる．当然ながら，他のノードのメモリ参照はローカンルなメモリ参照より遅くなるが，ローカルに見れば新たなメモリ階層，あるいは，ヘテロなメモリ階層と考えることもできる．このような場合においても Overshadow を有効に使うことができると考える．

Nieplocha らは，Global Array と呼ばれるプログラミングモデルを用いて，多階層なメモリの複雑さを隠蔽することを提案している [10]．Overshadow はそのような機能をサポートする可能性がある．

Peña らは，複雑なメモリシステムにおいて，データの最適な配置を決定するためのプロファイリングツールを開発している [11]．このツールは Valgrind[9] をベースに開発されている．

7. まとめと今後について

近い将来に広く使われるであろう複雑なメモリシステム上で，遠いメモリから近いメモリに内容をコピーする Overshadow を提案した．Overshadow におけるデータのコピーは，DMA ハードウェアを用いているが，同時に OS カーネルの仮想記憶管理を用いた技法により，従来方式 DMA だけによるコピーよりも，低遅延かつ高バンド幅でこコピーが可能である．プロトタイプによる評価結果では，従来方式 DMA を用いた場合に比べ，遅延時間を 38%に短縮することができ，同時に，該当メモリ領域を全てアクセスするのに要した時間も短くなった．

現時点においても Overshadow の開発は進められている．現時点での主な改良点は，1) IO-AT の DMA エンジンを用いた実装，2) M-PVAS に依らない実装，の 2 点である．同時に，新しい Xeon Phi（KNL やそれ以降）など，他の複雑なメモリシステム上での評価をおこなう予定である．

謝辞 本研究は，科学技術振興機構 (JST) の戦略的創造研究推進事業「CREST」における研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」によるものである．また，Overshadow の実装および評価において，株式会社 SRA の平野基孝，今井潔の両氏の支援があった．

参考文献

[1] Dong, X., Xie, Y., Muralimanohar, N. and Jouppi, N. P.: Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support, *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, Washington, DC, USA, IEEE Computer Society, pp. 1–11 (online), DOI: 10.1109/SC.2010.50 (2010).

[2] Endo, T., Nukada, A. and Matsuoka, S.: TSUBAME-KFC: A modern liquid submersion cooling prototype towards exascale becoming the greenest supercomputer in the world, *Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on*, pp. 360–367 (online), DOI: 10.1109/PADSW.2014.7097829 (2014).

[3] Exascale Operating Systems and Runtime (OS/R) Software Technical Council: Exascale Operating System and Runtime Softwar Report, Technical report, U.S. Department of Energy (2012).

[4] Intel: Accelerating High-Speed Networking with Intel I/O Acceleration Technology, White paper, Intel Corp. (2006).

[5] Intel Corp.: *Intel Xeon Phi Coprocessor System Soft-*

ware Developers Guide, Revision 2.0.3 edition (2012).

[6] Kerrisk, M.: migrate_pages(2) - Linux manual page (2015). `http://man7.org/linux/man-pages/man2/migrate_pages.2.html`.

[7] Kontothanassis, L., Hunt, G., Stets, R., Hardavellas, N., Cierniak, M., Parthasarathy, S., Meira, W., Jr., Dwarkadas, S. and Scott, M.: VM-Based Shared Memory on Low-Latency, Remote-Memory-Access Networks, Technical Report Technical Report #643, University of Rochester (1996).

[8] Li, D., Vetter, J. S., Marin, G., McCurdy, C., Cira, C., Liu, Z. and Yu, W.: Identifying Opportunities for Byte-Addressable Non-Volatile Memory in Extreme-Scale Scientific Applications, *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, IPDPS '12, Washington, DC, USA, IEEE Computer Society, pp. 945–956 (online), DOI: 10.1109/IPDPS.2012.89 (2012).

[9] Nethercote, N. and Seward, J.: Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation, *SIGPLAN Not.*, Vol. 42, No. 6, pp. 89–100 (online), DOI: 10.1145/1273442.1250746 (2007).

[10] Nieplocha, J., Harrison, R. and Foster, I.: Explicit Management of Memory Hierarchy, *Advances in High Performance Computing* (Grandinetti, L., Kowalik, J. and Vajtersic, M., eds.), NATO ASI Series, Vol. 30, Springer Netherlands, pp. 185–199 (1997).

[11] Peña, A. J. and Balaji, P.: Toward the efficient use of multiple explicitly managed memory subsystems, *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014*, pp. 123–131 (online), DOI: 10.1109/CLUSTER.2014.6968756 (2014).

[12] Phadke, S. and Narayanasamy, S.: MLP aware heterogeneous memory system, *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6 (2011).

[13] Reinders, J.: Knights Corner: Your Path to Knights Landing (2014). `https://software.intel.com/sites/default/files/managed/e9/b5/Knights-Corner-is-your-path-to-Knights-Landing.pdf`.

[14] Sato, M., Fukazawa, G., Shimada, A., Hori, A., Ishikawa, Y. and Namiki, M.: Design of Multiple PVAS on InfiniBand Cluster System Consisting of Many-core and Multi-core, *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, New York, NY, USA, ACM, pp. 133:133–133:138 (online), DOI: 10.1145/2642769.2642795 (2014).

[15] Shimada, A., Gerofi, B., Hori, A. and Ishikawa, Y.: PGAS Intra-node Communication towards Many-Core Architecture, *In PGAS 2012: 6th Conference on Partitioned Global Address Space Programing Model*, PGAS'12 (2012).

[16] Shimada, A., Gerofi, B., Hori, A. and Ishikawa, Y.: Proposing a new task model towards many-core architecture, *Proceedings of the First International Workshop on Many-core Embedded Systems*, MES '13, New York, NY, USA, ACM, pp. 45–48 (online), DOI: 10.1145/2489068.2489075 (2013).

[17] 佐藤未来子，島田明男，吉永一美，辻田祐一，　堀敦史，石川裕，並木美太郎：Xeon Phi 搭載計算機における DMA・MMIO 併用型 CPU 間データ通信機構，技術報告 17，東京農工大学, 理化学研究所 AICS, 理化学研究所 AICS, 理化学研究所 AICS, 理化学研究所 AICS, 理化学研究所 AICS ／東京大学, 東京農工大学 (2014).

[18] 島田明男，堀　敦史，石川　裕：新しいタスクモデルによるメニーコア環境に適した MPI ノード内通信の実装，情報処理学会論文誌 コンピューティングシステム， Vol. 8, No. 2, pp. 36–54 (2015).