DIMM スロット装着型不揮発性メモリ上の RDMA によるメッセージキューイングシステムの試作

南里 豪志 [1)], 大江 和一 [2)], 吉田 英司 [3)]、大辻 弘貴 [3]、林 英里香 [3]

1) 九州大学 情報基盤研究開発センター
2) 国立情報学研究所
3) 株式会社富士通研究所

nanri.takeshi.995@m.kyushu-u.ac.jp

Preliminary Study of Message Queueing System based on RDMA over Non-Volatile DIMM

Takeshi Nanri[1)], Kazuichi Ooe[2)], Eiji Yoshida[3)], Hiroki Ohtsuji[3)], Erika Hayashi[3)]

1) Research Institute for Information Technology, Kyushu Univ.
2) National Institute Informatics
3) Fujitsu Laboratories Ltd.

概要

近年、DRAM と SSD の間に位置づけられる新たな記憶装置として、DIMM スロットに装着する不揮発性メモリである Non-Volatile DIMM が注目されている。本研究では、代表的な NVDIMM である Intel Optane Persistent Memory の読み書き速度が高速インターコネクト InfiniBand の性能に近い点に着目し、この記憶装置上に RDMA 通信を用いたメッセージキューイングシステムを試験的に構築して性能を評価した。これは、キュー領域を DCPMM 上に配置するメッセージキューイングシステムであり、実装プロトコルとしては Push 型と Pull 型の二通りを試した。このシステムにおけるメッセージ操作の性能を実機で計測したところ、キュー領域を DRAM 上に配置した場合に対して遜色のない結果が得られたことから、DCPMM 上のメッセージキューイングシステムの実用性を確認した。また、DCPM にキューを配置した場合、メッセージサイズが大きくなるにつれて memcpy の所要時間の影響が大きくなり、特にキューの出力側のプロセスでは、Push 型と Pull 型の優劣がメッセージサイズに応じて入れ替わることを確認した。一方、入力側のプロセスでは、Push 型の enqueue の時間が短いことから通信時間の隠ぺいが期待できることが分かった。

1 はじめに

計算の大規模化や、データサイエンスなどの計算機利用の多様化に伴い、主記憶の容量に対する要求が高まっている。そこで近年、容量単価が安価で集積度の高い不揮発性メモリを DIMM スロットに装着する Non-Volatile DIMM (NVDIMM) 技術が注目されている。例えば、この NVDIMM の代表的な製品の一つである Intel Optane Persistent Memory (PMem) は、現時点で DRAM の 4 倍となる 512GB のメモリモジュールが選択可能であり、今後さらに高集積化が見込まれていること、および読み書き遅延時間が数 μ 秒と SSD の数十分の一であることから、DRAM と SSD の間に位置づけられる新しい記憶階層として期待されている。

PMem は DRAM と組み合わせて利用することとなっており、DRAM を Last Level Cache として利用する Memory Mode、もしくは DRAM と PMem をアプリケーション内で明示的に使い分ける App Direct Mode のいずれかの使用モードを選択して使用する。Memory Mode では、アプリケーションを書き換えずに PMem の容量分の記憶空間を利用できるものの、局所性が低いアプリケーションでは PMem への読み書きが頻発するため、DRAM のみを使用する場合に対して性能が大幅に低下する。一方 App Direct Mode では、アプリケーションの変更が必要となるものの、DRAM と PMem の容量を合わせた記憶空間を利用でき、さらに、データの読み書き頻度に応じてメモリへの配置を調整することにより、DRAM のみを使用する場合に対する性能低下を抑制することが可能と

なる。

このような PMem の特性を活用したアプリケーションの例として、本研究では、Remote Direct Memory Access (RDMA) によるメッセージキューイングシステムの、PMem への実装を提案する。RDMA は、遠隔の計算期間でメモリを読み書きする片方向の通信であり、非同期で効率的なデータ転送を可能とする。この RDMA を用いたメッセージキューイングシステムは、サーバクライアントモデルにおけるリクエスト機構や、通信ライブラリにおけるメッセージ処理機構の実装に用いられている [1]。通常、メッセージキューイングシステムがメッセージの保管のために必要となるキュー領域は、データサイズや計算機の規模に比例して増大する。そこで、このキュー領域を PMem に配置することで、データや計算機の大規模化への対応が可能となる。また、InfiniBand のような高性能ネットワークで計算ノード群を接続した並列計算機では、この RDMA の遅延時間が 1μ 秒程度であり、PMem の遅延時間と同程度であるため、キュー領域として DRAM の代わりに PMem を利用することによる性能への影響は小さいと期待できる。

PMem の基本性能に関する調査は盛んに行われており、いくつかが報告されている [2, 3, 4]。しかし、PMem 上にキュー領域を配置したメッセージキューイングシステムについての研究は、まだなされていない。本稿では、PMem 上に試験的にメッセージキューイングシステムを構築し、DRAM 上に構築した場合との性能を比較して実用性を検証する。また、予備実験として実施した PMem の基本性能の計測結果も、合わせて報告する。

2 PMem の概要

2.1 PMem を装着する DIMM スロットの選択

PMem は Intel 社が 2019 年に販売を開始した NVDIMM であり、DRAM と同様に DIMM スロットに装着してバイト単位読み書きが可能である。ただし、マザーボード上の DIMM スロットを PMem のみで使用することはできず、必ず DRAM と組み合わせて使用することが求められている。DRAM と PMem を装着する DIMM の選択肢はマザーボードごとに決められており、それぞれのメモリモジュールの枚数や、使用モードに応じて選択する [5]。

2.2 Linux における PMem の設定

Linux では、PMem の利用方法として、従来のストレージと同様にブロック単位で読み書きする方法と、メモリとしてバイト単位で読み書きする方法が提供されている。このうち従来のストレージと同様の方法では、通常のファイル I/O により読み書きが可能であるものの、PMem の性能を発揮させることはできない。これに対してバイト単位で読み書きする方法としては、さらに Filesystem DAX と、Direct DAX の二通りから選択できる。Filesystem DAX は、PMem 上に構築したファイルシステムのファイルを mmap 関数でマップすることにより、直接読み書きできるようにするものである。一方、Direct DAX は、I/O デバイスとして認識された PMem をプログラム中で直接読み書きできるようにするものである。このうち本稿では、最も高性能が期待できる Direct DAX を選択する。

Linux における PMem の Direct DAX での利用方法は、Intel 社の Web サイトに記述されている [6]。まず、PMem 管理ユーティリティ ipmctl を利用して、装着されている PMem を、region と呼ばれる領域として定義する。この時、個々の PMem メモリモジュールを別の region として定義する non-interleaved region と、複数の PMem メモリモジュールをまとめて一つの region として定義する interleaved region のいずれかを選択できる。interleaved region では、複数のメモリチャネルを使用することにより、読み書きの帯域幅が向上する。

次に、Linux の NVDIMM 管理ユーティリティ ndctl コマンドを利用して、定義された region に namespace を登録する。これはファイルシステムにおけるパーティションと同様の操作である。定義された namespace は、I/O デバイスとして Linux の /dev ディレクトリの中に現れる。

この I/O デバイスをプログラム中で open し、さらに mmap でメモリ領域としてマップすることにより、直接読み書き可能となる。また、この領域を InfiniBand の memory region として登録することにより、RDMA による遠隔読み書きも可能である。なお、PMem 向けのプログラミングインタフェースとしては、Intel 社が Persistent Memory Developing Kit (PMDK) が提供されている [7]。これは、mmap 等の操作を隠蔽して簡便に PMem を操作するものである。これに対して本稿で計測に用いた各プログラムは、PMem 本来の性能を評価するため、PMDK を介さずに作成した。

3 RDMA によるメッセージキューイングシステム

3.1 プログラミングインタフェース

本稿で実装するメッセージキューイングシステムは、図 1 に示す通り、一対一の片方向キューで構成される。入力側プロセスが enqueue コマンドでキューに投入したメッセージを、出力側プロセスが dequeue コマンドで取り出すことで、メッセージの転送が完了する。また、入力側プロセスは、flush コマンドにより、投入したメッセージが取り出されたことを確認することができる。

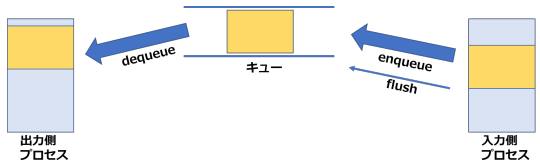

図 1　今回の実験で用いるメッセージキューイングシステム

今回実装したシステムでは、キューは固定サイズのエントリに分かれており、キュー内のエントリ数も固定となっている。エントリの大きさを超えるメッセージを enqueue することはできない。また、enqueue 時にキューが一杯になっている場合、出力側プロセスの dequeue により空きエントリが生じるまで待つ。一方、dequeue 時にキューが空である場合、入力側プロセスの enqueue によりメッセージがキューに投入されるまで待つ。

3.2 PMem 上への実装

3.2.1 実装の方針

今回の実装では、データ転送だけでなく、カウンタなどの管理情報の操作もすべて RDMA で実装する。これにより入力側は、キューが空いていれば、出力側のプロセスの状況によらずに enqueue 操作を完了することができる。RDMA を用いたメッセージキューイングシステムの一般的な実装として、キュー領域を出力側のプロセスのメモリ領域に配置する Push 型と、入力側のプロセスのメモリ領域に配置する Pull 型があげられる。

多くのメッセージキューイングシステムでは Push 型が選択されている。これは、Push 型が enqueue 時にデータを出力側のメモリに RDMA write するため、dequeue 側で並行して計算を進めることで、通信時間の隠ぺいが期待できるためである。一方 Pull 型は、dequeue 時に入力側のメモリから RDMA read するため、通信時間を隠蔽できない。

しかし、PMem では、RDMA write とローカルメモリへの書き込みが競合した場合の性能劣化の問題が報告されている [3]。そのため、PMem を用いたメッセージキューイングシステムの実装では、Push 型と Pull 型の優劣は、この書き込み競合による性能劣化の程度に依存する。そこで今回は、Push 型と Pull 型の両方を実装し、性能を比較する。

3.2.2 RDMA による実装

Push 型、Pull 型、それぞれによる実装の概要を、それぞれ図 2 および図 3 に示す。いずれの実装でも、入力側と出力側にそれぞれ二つのカウンタ変数を用意し、これらを更新することでキューを操作する。これらのカウンタは、実行開始時に 0 で初期化される。その後、入力側のプロセスが enqueue 命令を呼び出すと、キュー領域にメッセージをコピーし、自分の lctr_s カウンタを一つ増やして、その値を出力側のプロセスの rctr_t に RDMA write する。一方、出力側のプロセスは、dequeue 命令の中で、自分の rctr_t が lctr_t よりも大きくなるのを待って、キュー領域からメッセージをコピーし、自分の lctr_t を一つ増やして、その値を入力側のプロセスの rctr_s にに RDMA write する。入力側で、この rctr_s が lctr_s と一致するまで待つのが flush 命令である。これにより、メッセージが出力側から取り出され、キューが空になったことを確認できる。

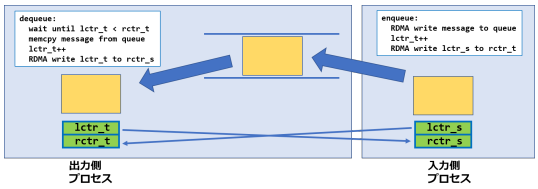

図 2　Push 型によるキューイングシステムの実装

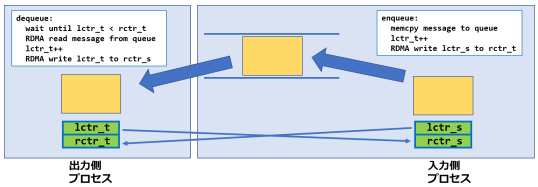

図 3　Pull 型によるキューイングシステムの実装

これらの実装で用いるカウンタは、全て DRAM 上に配置している。これは、カウンタの操作が RDMA だけでなくプロセス内でも頻繁に発生することから、

PMem に配置すると遅延時間が問題となると考えられたためである。また、RDMA 通信の対象とするため、これらのカウンタを並べた配列は posix_memalign 関数により 8 バイト境界で確保し、ibv_reg_mr 関数を用いて登録している。

Push 型ではキュー領域を出力側のプロセスのメモリ上に配置し、入力側の enqueue 命令発行時に RDMA write によってメッセージを書き込む。一方、出力側が dequeue 命令を発行すると、キュー領域上のメッセージを memcpy 関数で目的の場所にコピーする。

これに対して Pull 型では、キュー領域を入力側のプロセスのメモリ上に配置し、入力側の enqueue 命令発行時に memcpy 関数でメッセージをキュー領域にコピーする。出力側は、dequeue 命令発行時に RDMA read によってメッセージを直接目的の場所にコピーする。

このように、メッセージの転送元、もしくは転送先が RDMA の対象になるため、これらの領域もあらかじめ ibv_reg_mr 関数を用いて NIC に登録しておき、さらに、アドレスと rkey をプロセス間で交換しておく。今回は実験用のシステムであったため、これらの準備はプログラムの初期化処理で済ませているが、実際のプログラムでは、プログラマがメモリ上の任意の場所を登録するためのインタフェースを用意する必要がある。

このシステムについて、キュー領域を PMem に配置した場合と DRAM に配置した場合の性能を比較する。DRAM 上にキュー領域を配置する場合は、posix_memalign を用いて 8 バイト境界で領域を確保する。一方、PMem に配置する場合、PMem を Direct DAX で利用する。そのため、I/O デバイスをプログラム中で open し、さらに mmap でメモリ空間にマップしたものをキュー領域として使用する。なお、PMem を同時に複数の用途で使用する場合、Direct DAX では PMem 内の領域確保や空き領域管理をユーザレベルで行う必要がある。しかし今回の実験では PMem の全領域をキュー領域として占有するため、特別なメモリ管理等は行っていない。

4 実験結果

4.1 実験環境

本節では、PMem と DRAM を混載したシステムにおける、基本性能およびメッセージキューイングシステムの性能計測に関する実験結果を報告する。実験に使用した計算機の仕様を表 1 に示す。この計算機 2 台を InfiniBand スイッチで接続し、性能を計測した。

表 1　実験に使用した計算機

CPU	Intel Xeon Silver 4215 (2.50GHz) x 1
DRAM	DDR4 2400MT/s, 8GB x 4 slots
Pmem	Intel Optane Persistent Memory, 2400MT/s, 128GB x 2 slots
Motherboad	Supermicro X11SPL-F
OS	Cent OS 8.1 (kernel 4.18.0)
Network	InfiniBand Connect-IB, MLNX_OFED_LINUX-4.7-3.2.9.0

なお、この計算機のマザーボードで使用可能な DIMM スロット数は 8 である。そこで、DRAM と PMem の配置として、表 2 に示す PMem1 と PMem2 の二通りを試した。このうち PMem1 は、Intel 社推奨の構成であり、CPU が持つ 6 本のメモリチャネルにそれぞれメモリモジュールを 1 個ずつ割り当てるため、読み書き時のメモリチャネルの競合は発生しない。これに対して PMem2 は、同じメモリチャネルに DRAM と PMem を混在させることによる性能への影響を調査するための構成で、非推奨である。

表 2　DRAM と PMem の配置

配置名	PMem1	PMem2
スロット A1	DRAM	DRAM
スロット A2	空き	PMem
スロット B1	DRAM	DRAM
スロット C1	PMem	空き
スロット D1	DRAM	DRAM
スロット D2	空き	PMem
スロット E1	DRAM	DRAM
スロット F1	PMem	空き

4.2 基本性能調査

4.2.1 ローカル読み書き性能

ローカル読み書き性能として、まず、計算機内のメモリへの読み書きにおける遅延時間を計測した。計測に使用したプログラムは、コピー先とコピー元の位置をそれぞれ指定したストライドで移動しながら 1 バイトのデータコピーを繰り返すものである。コピー元とコピー先のメモリ領域を DRAM、PMem1、PMem2 から選択し、ストライドを変化させながら遅延時間を計測した結果を図 4 に示す。横軸がストライド、縦軸

が遅延時間である。

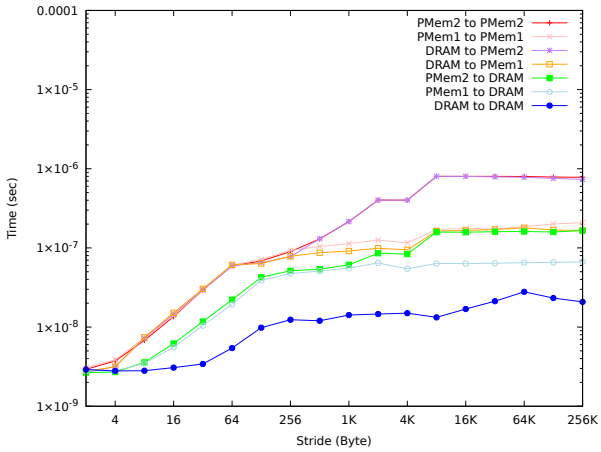

図 4　ローカル読み書き遅延時間

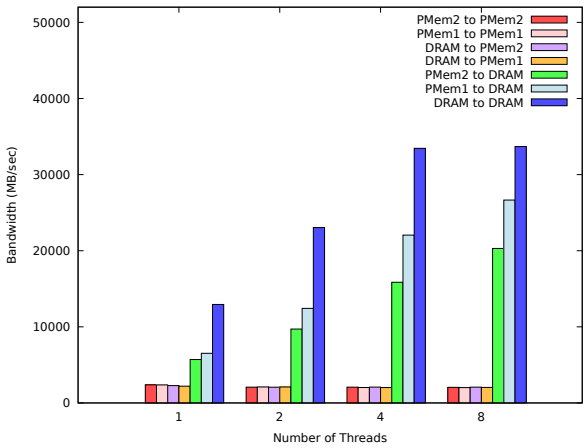

図 5　ローカル読み書き帯域幅

コピー元とコピー先としていずれのメモリ領域を選択した場合でも、ストライドが小さい間は遅延時間が短くなっているのはキャッシュの効果によるものである。また、いずれの場合も、8KB 以上のストライドでは遅延時間がほとんど変化しなくなるため、この範囲での計測値が、キャッシュの効果を含まない遅延時間であると考えられる。DRAM から DRAM へのコピーでは、この遅延時間が 20〜30n 秒であるのに対し、PMem1 から DRAM へのコピーが約 70n 秒であった。また、コピー先が PMem1 の場合は、コピー元が DRAM、PMem1 のいずれの場合も、遅延時間が約 180n 秒であった。これにより、特に PMem への書き込みに要する時間が大きいことが確認できた。

また、PMem2 から DRAM へのコピーが約 180n 秒、DRAM もしくは PMem2 から PMem2 へのコピーが約 800n 秒と、PMem1 の場合に比べてさらに大幅な性能低下が見られた。PMem1 と PMem2 の性能差についての原因は調査中である。

次に、計算機内のメモリへの読み書きの帯域幅を計測した。計測に使用したプログラムは、STREAM ベンチマーク [8] の COPY コードである。遅延時間計測と同様、コピー元とコピー先のメモリ領域を DRAM、PMem1、PMem2 から選択し、1GB のデータのコピーにおける帯域幅を、スレッド数を変化させながら計測した。計測結果を図 5 に示す。

計測結果から、コピー先が DRAM の場合の、PMem1 や PMem2 からのコピーの帯域幅は、DRAM からのコピーの帯域幅の約半分、もしくはそれ以上となることが確認できた。また、遅延時間ほどではないものの、PMem1 に対して PMem2 で読み出し性能が低下することも分かった。

一方、コピー先が PMem の場合は、メモリ配置が PMem1、PMem2 のいずれを選択しても帯域幅が約 1GB/秒であり、スレッド数を増やしても変化しないことが分かった。これにより、PMem の書き込み帯域幅が 1GB/秒程度であることが確認できた。

4.2.2　RDMA 性能

DRAM および PMem に対する RDMA の性能を、OFED に付属していた Performance Tests のコマンドにより計測した。使用したコマンドは、ib_read_lat（RDMA Read 遅延）、ib_read_bw（RDMA Read 帯域幅）、ib_write_lat（RDMA Write 遅延）、ib_write_bw（RDMA Write 帯域幅）である。これらは全て、サーバクライアント型となっており、サーバ側の計算機に対してクライアント側が RDMA 命令を発行する。サーバ側、およびクライアント側について、それぞれメモリ領域として DRAM、PMem1、PMem2 を選択した場合の、各コマンドの計測結果を図 6、7、8、9 に示す。

メッセージサイズが 128 バイト以下の場合の RDMA Read 遅延時間は、サーバ側のメモリ領域が DRAM である場合、すなわち遠隔の DRAM から読み出す場合には、クライアント側のメモリ領域が DRAM、PMem1、PMem2 のいずれの場合も約 1.9μ 秒程度であった。一方、サーバ側のメモリ領域が PMem1 もしくは PMem2 の場合、遅延時間が約 2.2μ 秒と、若干長くなった。

また、RDMA Read 帯域幅も、16KB 以下の転送サイズでは、PMem1 もしくは PMem2 から読み出す場合に性能低下が見られた。これはローカル読み書きの性能と逆の傾向であり、原因を調査中である。一方、

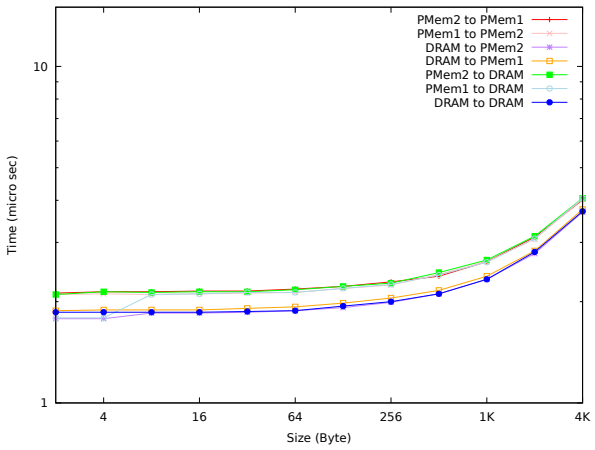

図 6　RDMA Read 遅延時間

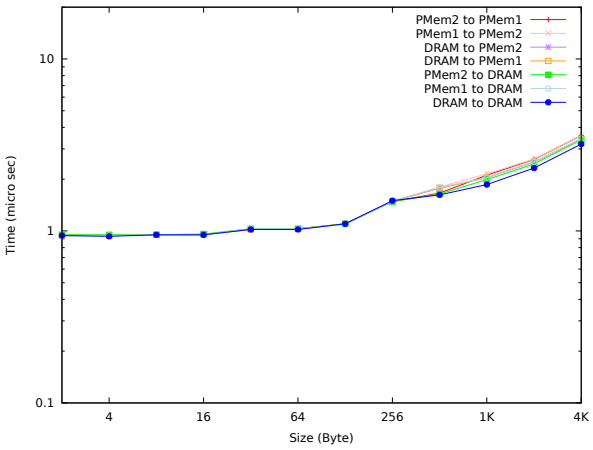

図 8　RDMA Write 遅延時間

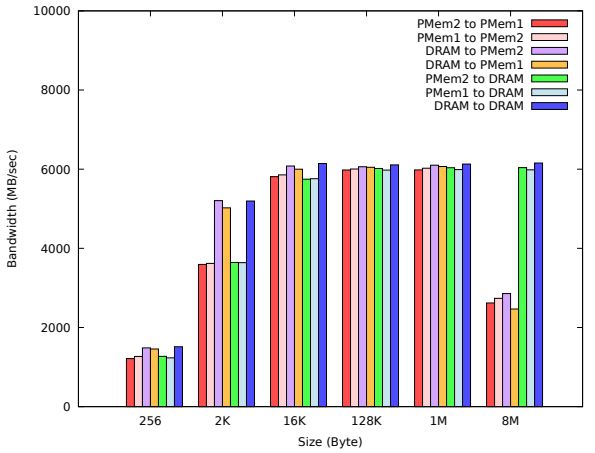

図 7　RDMA Read 帯域幅

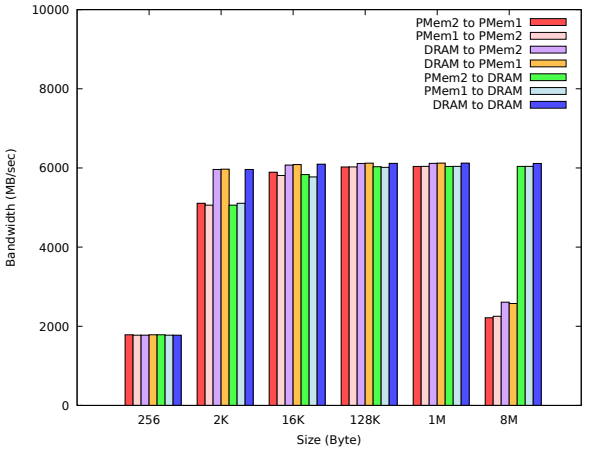

図 9　RDMA Write 帯域幅

8MB の転送時の帯域幅は、PMem1 もしくは PMem2 に書き込む場合に低下している。これはローカル読み書きと同様の傾向であるものの、8MB より小さい転送サイズではいずれの場合も高い帯域幅が計測される原因は、今のところ不明である。

次に、RDMA Write 遅延時間は、いずれのメモリ領域の組み合わせでもほとんど同じであった。なお、全体的に RDMA Write が RDMA Read より低遅延となるのは、Read ではクライアントからサーバに送付した読み出し要求に応じてデータ転送が開始されるのに対して、Write ではクライアントが直接サーバにデータ転送を開始するためである。一方、RDMA Write 帯域幅の傾向は、RDMA Read 帯域幅とほぼ同様であった。

4.3　メッセージキューイングシステムの性能

4.3.1　性能計測の概要

RDMA によるメッセージキューイングシステムの性能を計測する。計測では、2 台の計算機にそれぞれキューの入力側プロセスと出力側プロセスを起動し、その間でメッセージを転送する。前述の通り、Push 型の場合は出力側、Pull 型の場合は入力側のプロセスにキュー領域を確保する。さらに、Push 型、Pull 型、それぞれキュー領域を確保するメモリ領域として DRAM、PMem1、PMem2 を選択した場合の性能を比較する。いずれの場合も、キュー領域として 10GB を配置し、これを 32MB のスロットに分割して使用する。

計測に用いたプログラムでは、入力側プロセスがメッセージの enqueue と flush を繰り返し、出力側プロセスは、計算と dequeue を繰り返す。出力側プロセスの計算としては行列積を用い、各メッセージサイズ

での通信時間と計算時間の比率が大きく変動しないように、行列サイズをほぼメッセージサイズの三乗根に比例して増加させる。今回の実験では繰り返し回数を100 回とし、それぞれの操作の所要時間の平均値を計測結果として示す。

4.3.2 入力側プロセスの所要時間

図 10 に、入力側のプロセスの Enqueue と Flush に要した時間を示す。Enqueue と Flush の合計時間を見ると、メッセージサイズが小さい場合、Pull 型より Push 型が高速である。これは、Push 型では入力側の Enqueue でデータ転送が開始されるため、出力側の計算とデータ転送が並行して進むことにより通信時間を隠蔽できたことによるものと考えられる。また、Push 型で Enqueue 時間がメッセージサイズによらず一定になっているのは、Enqueue 内の RDMA Write 命令が、データ転送の完了を待たずに終了するためである。

一方、メッセージサイズが大きくなると Push 型と Pull 型の優劣がほとんど見られなくなる。特にキュー領域として PMem1 もしくは PMem2 を使用した場合、32MB で Push 型と Pull 型の優劣が逆転する。これは、Push 型では出力側の dequeue 時にキュー領域からアプリケーション領域へのメモリコピーが必要であり、この時間がメッセージサイズの増加に伴って無視できなくなることが原因である。特に PMem1 や PMem2 は読み出しの帯域幅が DRAM よりも狭いため、メモリコピーに要する時間の影響が顕著に出たと考えられる。なお、PMem1 と PMem2 の間での性能の差異はほとんど見られなかった。

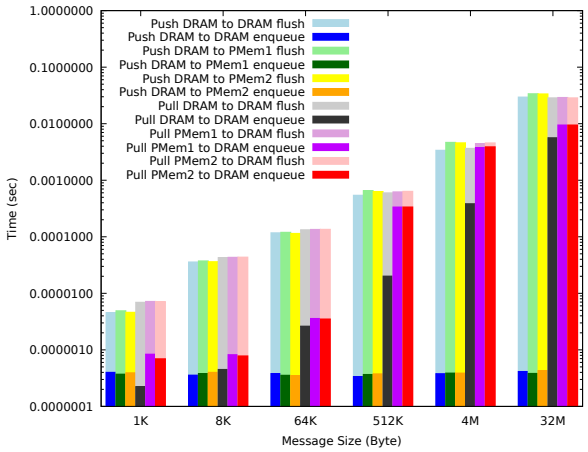

図 10　入力側プロセスの所要時間

4.3.3 出力側プロセスの所要時間

図 11 に、出力側のプロセスの Dequeue と、それ以外の処理に要した時間を示す。こちらも、入力側と同様、全体の所要時間は、メッセージサイズが小さい場合に Pull 型よりも Push 型が短く、メッセージサイズが大きくなるにつれてその優劣がほとんど見られなくなる。また、この傾向は Dequeue に要した時間にも現れている。これは、Push 型の場合、Dequeue の中で入力側プロセスによるキュー領域への RDMA Write とカウンタへの RDMA Write の完了を待つ時間、およびキュー領域からアプリケーション領域へのメモリコピーに要する時間が必要であり、特にメッセージサイズが大きい場合にメモリコピーの時間が無視できなくなるためである。さらに、今回用いたプログラムは 1 スレッドで動作するため、図 5 で見た通り、DRAM から DRAM へのコピーであっても帯域幅が 13GB/秒程度であり、図 7 で示した RDMA Read の帯域幅 6GB/秒に対して大きな優位性が無いことも、メモリコピーに要する時間による影響が大きい要因として考えられる。なお、出力側プロセスでも PMem1 と PMem2 の間での性能の差異はほとんど見られなかった。

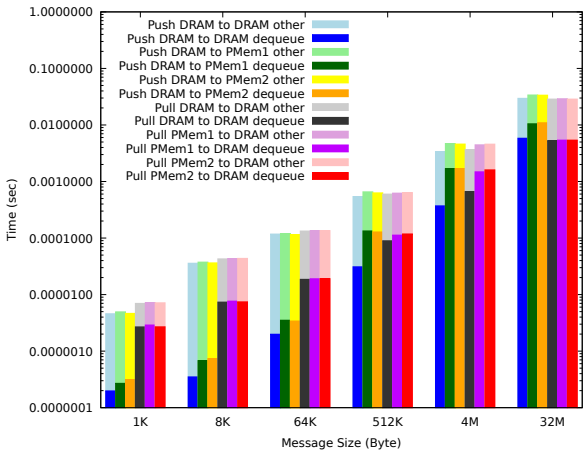

図 11　出力側プロセスの所要時間

一方、[3] では、RDMA Write とローカルメモリへの書き込みが競合した場合に、ローカルメモリの書き込み性能が低下する現象が報告されている。本稿のメッセージキューイングシステムでは、Push 型でキュー領域を PMem1 もしくは PMem2 に確保した場合に、入力側プロセスからキュー領域への RDMA Write と出力側プロセスの計算で競合が発生すると考えられる。そこで、各方式での出力側プロセスの計算時間を図 12 に示す。図からわかる通り、Push 型、Pull 型、どちらも、DRAM と PMem1、PMem2 での有意な性能差は見られない。すなわち、今回の実験では、RDMA Write とローカルメモリへの書き込みの競合による性

能低下は確認できなかった。

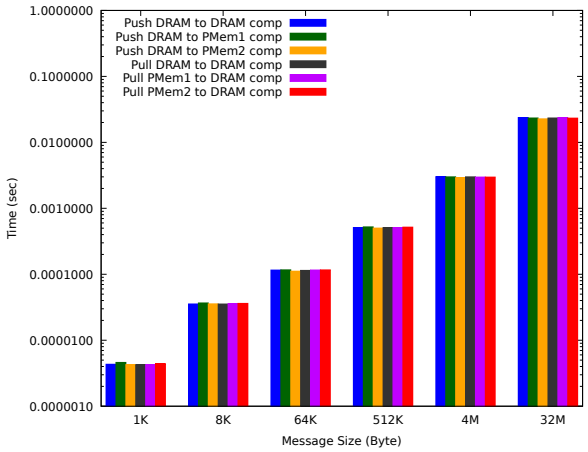

図 12　出力側プロセスの計算時間

5　むすび

SSD より高速で DRAM より大容量化が期待できる新しい記憶装置 NVDIMM の代表的な製品である PMem について、まず基本性能を計測した。その結果、ローカルの読み書き性能は DRAM より大幅に劣るものの、RDMA 通信の性能に近いことを確認した。これにより、RDMA 通信を利用したアプリケーションでは、DRAM を PMem に置き換えることによる性能低下が抑制できると期待できる。そこで、RDMA 通信を用いたメッセージキューイングシステムにおいて、キュー領域を PMem に確保した場合と DRAM に確保した場合の性能を比較した結果、全体として性能差が小さく、十分実用性が期待できることが分かった。

また、メッセージキューイングシステムの実装手段としては、メッセージサイズが小さい場合は、通信隠蔽効果により Push 型が Pull 型に対して有利であった。しかしメッセージサイズが大きい場合、出力側プロセスの Dequeue におけるメモリコピーに要する時間が影響し、Pull 型に対する優位性が見られなくなり、特に PMem にキューを配置する実装では、Push 型と Pull 型の優劣が入れ替わることが分かった。なお、既存研究で報告されていた、PMem への RDMA write とローカルメモリへの書き込みの競合による性能劣化については、今回の実験では確認できなかった。

参考文献

[1] Kalia, A. et al, ”Using RDMA efficiently for key-value services”, Proceedings of SIGCOMM’14, pp. 295-306, Aug. 2014. DOI: 10.1145/2619239.2626299

[2] Hirofuchi, T. et al, ”A Prompt Report on the Performance of Intel Optane DC Persistent-Memory Module”, IEICE Trans. Inf. and Syst., vol. E103-D, no. 5, pp. 1168-1172, 2020. DOI: 10.1587/transinf.2019EDL8141

[3] Oe, K., ”Analysis of Interference between RDMA and Local Access on Hybrid Memory System”, Aug. 2020. arXiv:2008.12501v1

[4] Weiland, M. et al, ”An early evaluation of Intel’s optane DC persistent memory module and its impact on high-performance scientific applications”, SC’19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Nov. 2019. DOI: 10.1145/3295500.3356159

[5] Module DIMM Population for Intel Optane Persistent Memory. https://www.intel.com/content/www/us/en/support/articles/000032932/memory-and-storage/data-center-persistent-memory.html

[6] Quick Start Guide: Provision Inte Optane DC Persistent Memory https://software.intel.com/content/www/us/en/develop/articles/quick-start-guide-configure-intel-optane-dc-persistent-memory-on-linux.html

[7] Programming Persistent Memory https://software.intel.com/content/www/us/en/develop/topics/persistent-memory.html

[8] STREAM Benchmark http://www.cs.virginia.edu/stream/ref.html