Vol.2022-HPC-183 No.24
2022/3/18

計算ノード上の不揮発性メモリを用いたMPI-IOバーストバッファの設計

平賀 弘平[1,a)]　建部 修見[1,b)]

概要：本研究では，スーパコンピュータの計算ノード上にある不揮発性メモリ (PMem) を活用するための，MPI-IO のデバイス抽象化レイヤー ADIO の設計を行う．これまでノードローカル PMem の利用方法として，アドホック分散ファイルシステムが提案されているが，本研究では MPI-IO から直接計算ノード上の PMem を並列ファイルシステムのバーストバッファとして活用する方法を提案する．PMem 実デバイスを用いた IOR による N-1 書き込み性能評価では，シングルノードではローカルストレージと比較して 4.39 倍の 10.35 GiB/s の書き込みバンド幅を達成した．マルチノードでは，4 ノードを用いた場合 38.18 GiB/s の合計書き込みバンド幅を達成し，スケーラブルなバンド幅を示した．

1. はじめに

深層学習がもたらした機械学習 AI のブレークスルーにより急速に進歩しているデータサイエンス分野では，データを分析する学問的性質上ストレージ性能の向上はますます重要となっている．近年コモディティで利用可能になった新しい高速ストレージデバイスに，不揮発性メモリ（PMem）がある．PMem はシステムの電源を落としてもデータを保持する不揮発なストレージデバイスだが，HDD や SSD とは全くデバイス特性が異なっている [1].

PMem は，従来の HDD や NVMe SSD と同様に，単なる高速なブロックデバイスとしてファイルシステムから POSIX インタフェースを介して利用できるが，より最適化されたデバイス利用方法として，SNIA NVM Programming Model（NPM）[2] が標準化されている．NPM では，PMem デバイスを User 空間の仮想アドレスに直接 mmap して，PMem をメモリとして扱う．CPU のキャッシュライン上にあるデータはユーザが明示的に PMem デバイスに Flush する必要があり，Kernel を経由することなくキャッシュラインを Flush する操作がサポートされている．これらは，DAX（Direct Access）と呼ばれており，従来のファイルシステムとは完全に異なるインターフェースとなる．従って既存のアプリケーションは，NPM に対応するようにアプリケーションコードを修正するか，DAX を活用するストレージシステムを利用しなければならないという問題がある．これらのストレージシステムは NPM を既存の POSIX インターフェース等に変換する役割を持っているが，プロセス間通信のコストや，インターフェース変換のため追加のオーバーヘッドが発生する．本研究では，HPC 分野で既に広く使用されている集団 I/O ライブラリである MPI-IO を対象とし，MPI-IO 内部から NPM を用いた PMem アクセスを行うことで，PMem を利用するための追加の管理プロセスやストレージ管理スタックを増やさずに直接 PMem を利用して，デバイスの性能を最大限発揮できるようにする．MPI-IO を使用する既存アプリケーションは，コードの変更なしに PMem への最適化された集団 I/O を行えるようになる．

大規模なデータサイエンスでは，スーパーコンピュータ（スパコン）が利用される．従来のスパコンにおけるストレージシステム構成は，複数のストレージ専用の計算機から構成される並列ファイルシステムが一般的に利用されているが，ストレージデバイスをストレージ専用ノードではなく，アプリケーションが実行される計算ノード上に配置する構成が増えている．このシステム構成は，ストレージの利用に関する通信が最小限になることと，他のジョブの影響を受けない専有的な利用が可能という利点から，スパコン全体の I/O 性能をスケールさせるために有用と考えられている．本研究では，PMem を計算ノード上に配置したシステム構成（ノードローカル PMem）を対象とし，ノードローカル PMem を活用するストレージシステムを提案する．

ノードローカルストレージを活用する手法として，これまでにノードローカルバーストバッファに関する研究がな

[1] 筑波大学計算科学研究センター
[a)] hiraga@ccs.tsukuba.ac.jp
[b)] tatebe@cs.tsukuba.ac.jp

されている．バーストバッファは，HPC アプリケーションにおける I/O が，アプリケーション実行のある一定の区間でバースト的に行われる特徴に着目し，一旦高速なストレージに書き込みのバッファリングを行い，バックグラウンドで低速なストレージにフラッシュすることで，I/O 全体のスループットを改善するストレージシステムのコンポーネントである．本研究では，ノードローカル PMem をバーストバッファとして活用する手法を提案する．

1.1 貢献

本研究は，計算ノード上に不揮発性メモリを配置したスーパーコンピュータにおける，大規模データサイエンスアプリケーションのストレージ性能の向上を目的とする．以下に貢献を挙げる．

- **ノードローカル PMem をバーストバッファとして活用するストレージシステムの設計．**PMem のデバイス特性を活かした I/O 最適化のため，PMem を単にローカルファイルシステムとして利用するのではなく，NPM を用いて Byte-addressable な Memory として活用する．
- **MPI-IO によるデータサイエンスアプリケーションから使いやすいライブラリの設計．**既に広く利用されている MPI-IO のインターフェースで，直接計算ノード上の PMem を扱えるようにする．アプリケーションコードの変更は必要なく，追加の I/O 用 daemon プロセスのようなものも必要ない．集団 I/O アクセスにおいて，POSIX インターフェースより最適化が可能で，ライブラリ内から直接 PMem デバイスを扱い，ストレージ管理コストを最小化する．
- **性能評価により，PMem をローカルファイルシステムとして利用するよりも高い性能を実現する**ことを示す．

2. 関連研究

HPC における並列アプリケーションのファイルアクセスパターンには，Single Shared File（N-1）と，File per Process（N-N）がある [3], [4]．どちらのアクセスパターンも重視されており，IO-500 ベンチマーク [5] では IOR-easy が N-N，IOR-hard が N-1 の性能評価を行っている．大規模なアプリケーションでは N-1 の利用率が高いという調査結果がある [6]．一般的に N-1 アクセスの方が N-N に比べて性能がスケールしづらい．これはストレージシステム内部でデータブロックにアクセスする際にロック競合が発生するためである．この問題に対処するため，既存研究では N-1 アクセスをストレージシステム内部で N-N アクセスに変換する手法が提案されている．PLFS [3]，BurstFS [4] は，各プロセスの N-1 ファイル書き込みを N-N のログベースの書き込みに変換している．MPI-IO/Gfarm [7] は，分散ファイルシステム Gfarm [8] のための MPI-IO を提案している．MPI-IO/Gfarm では，MPI-IO の MPI_File_set_view の中で N-1 を N-N に変換する view を構築し，各プロセスの I/O を，Gfarm 上のの N-N ファイルの I/O へ変換する．杉原らは，N-1 を N-N に変換する機構として，*Sparse Segments* [9] を提案した．この手法は N-1 ファイルを open する際に，MPI の各プロセス毎に個別にファイルを作成し，そこにアプリケーションの I/O 要求を転送する．実際に書き込むファイルのオフセットまでシークして書き込むことで，データが無い場所は hole となる．これにより，元の N-1 ファイルのオフセット情報を保持した N-N ファイルを Gfarm 上に作成する．

計算ノード上のローカルストレージを活用する既存研究としては，アドホック分散ファイルシステムが挙げられる．アドホック分散ファイルシステムはジョブが割り当てられている間のみ利用する分散ファイルシステムで，計算ノードのローカルストレージを集約して共有ファイルシステムとして利用可能になる．Gfarm/BB [10]，CHFS [11], GekkoFS [12], BurstFS [4], BeeOND [13] 等がある．CHFS を除いてこれらは基本的に従来のストレージデバイスを対象としたストレージシステムである．本研究は分散ファイルシステムを構築せずに，MPI-IO から直接ノードローカル PMem を扱えるようにする．

3. ノードローカル PMem バーストバッファの設計

本章ではノードローカル PMem をバーストバッファとして活用するストレージシステムの設計を示す．スパコンにおける共有ファイルシステムへの集団 I/O 性能向上を目的とし，特に HPC 分野で広く利用されている MPI-IO から透過的に扱えるようにする．まずはじめに，システム全体の概要と，想定する計算機システム構成と課題設定について 3.1 節で述べる．次に PMem 上の不揮発データ構造 *pmembb* について 3.2 節で述べる．最後に pmembb を MPI-IO から利用する設計について 3.3 節で述べる．

3.1 概要

図 1 に提案ストレージシステム全体の概要図を示す．本研究が想定する計算機システム構成は，アプリケーションの MPI 並列プロセスが実行される計算ノードと，計算の入力データや結果の保存等のためのグローバルファイルシステムから成る．計算ノードはユーザジョブが割り当てられている間，専有的に利用ができる．グローバルファイルシステムは，計算ノードの割り当てとは無関係に共有利用できるため，他のジョブの影響を受ける．各計算ノードはノードローカル PMem を持つ．ノードローカル PMem の利用方法にはいくつか選択肢があり，

- POSIX インターフェースによるローカルファイルシ

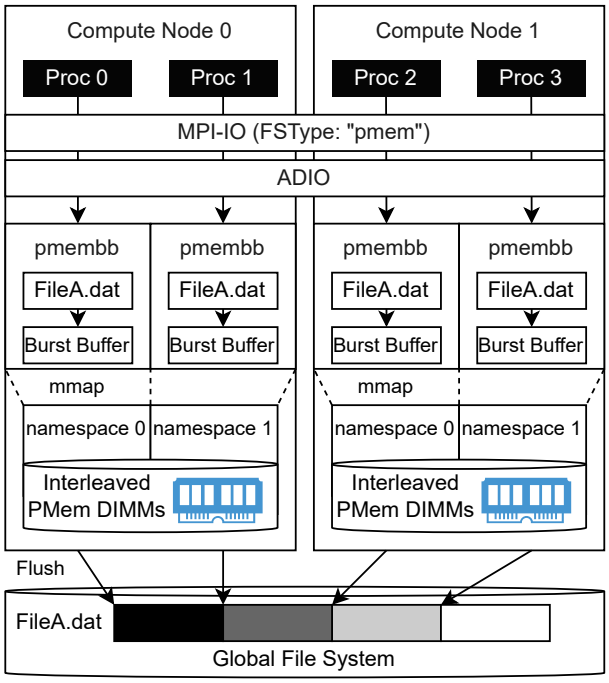

図 1 システム概要図
Fig. 1 System Overview

ステムとして利用

- fsdax モード
- devdax モード

が想定される．POSIX インターフェースは，ノードローカル PMem を単なる高速なブロックデバイスとして利用する方法である．fsdax モードと devdax モードは，SNIA NPM に基づくインターフェースでノードローカル PMem を利用する方式で，fsdax は fsdax 対応のローカルファイルシステムを介して PMem にアクセスし，Kernel のストレージスタックを回避する．devdax モードは PMem の DAX キャラクターデバイスに直接アクセスする．提案システムでは基本的には devdax モードの利用を想定する．devdax が利用できない場合は fsdax に fallback する．

提案ストレージシステムは，ノードローカル PMem をグローバルファイルシステムの Write back キャッシュのように利用するための仮想デバイス層を，MPI-IO 内部のデバイス抽象化層 ADIO [14] を利用して実現する．アプリケーションの各 MPI プロセスは，グローバルファイルシステム上のファイル（図 1 の FileA.dat）を，MPI-IO を介して集団 Open する．この時，ノードローカル PMem バーストバッファを利用することを明示するため，アプリケーションはファイルパスの先頭に **pmem:** プレフィックスをつけて Open する．そして，グローバルファイルシステム上のファイルに対する MPI-IO による集団 I/O リクエストを，ADIO の仮想デバイスを通してノードローカル PMem 上のバーストバッファに転送する．

PMem 対応 ADIO 仮想デバイスは，ノードローカル

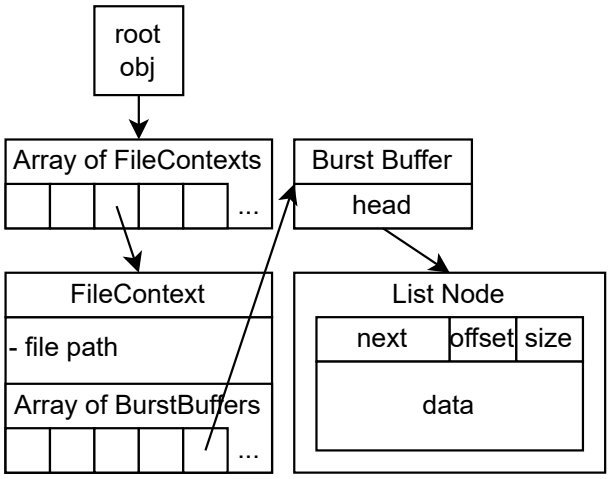

図 2 pmembb 不揮発データ構造
Fig. 2 The pmembb nonvolatile data structure

PMem を NPM から扱うため *pmembb* library を利用する．pmembb は PMem を DAX モードで mmap するための抽象データ構造 *pmembb pool* を持つ．pmembb の詳しい解説は 3.2 節で述べる．ノードローカル PMem は予めプロセス数分 ndctl namespaces に分割されており，pmembb pool は，namespace で分割された devdax 上に作成される．

ファイル Open の際に，pmembb 内部に Open したファイルに対応するバーストバッファが新規に作成される．ファイルにデータを write すると，ノードローカル PMem バーストバッファへ書き込まれる．グローバルファイルシステム上のファイルへの書き込みデータ反映は，明示的に Flush を行うまで遅延される．読み込みは現在のところはグローバルファイルシステムから直接読み込んでいる．バーストバッファ内にデータがある場合は，一旦すべてのデータをグローバルファイルシステムに Flush を行ってから，読み込みを行う．本研究では，まずはファイルの集団書き込みに焦点を当て，ノードローカル PMem の活用により書き込みの高速化を行う．

3.2 pmembb: PMem 上の不揮発データ構造

本節では，ノードローカル PMem 上のバーストバッファの不揮発データ構造を扱うためのライブラリである **pmembb** について述べる．pmembb ライブラリは，SNIA NPM により mmap された PMem を，pmembb pool として管理する．各 pmembb pool は mmap した PMem 領域の不揮発なヒープ領域をもち，そこからメモリ alloc や free を行う．また，不揮発ヒープ領域内のデータ構造の Root Object を指し示す特別なポインタを管理しており，そこからすべてのバーストバッファを列挙できるようになっている．

pmembb は，グローバルファイルシステム上のファイル毎に，バーストバッファを作成，列挙，書き込み，フラッシュを行えるようになっている．ファイルとバーストバッ

ファの対応付けは，グローバルファイルシステムのファイルパスを key とした Key Value 形式で保持している．ファイル毎にバーストバッファを管理するのは，バーストバッファ内に書き込んだデータを後でグローバルファイルシステム上の対応するファイルに反映するために必要となる．バーストバッファは，Log-structured 形式のデータ構造で，データの追記書き込みと，古い書き込みデータから順に列挙と削除する機能を持つ．PMem は小さいランダムな書き込みは苦手としている [1] ため，Log-structured 形式で書き込みリクエストを管理すると，ファイルへの書き込みをアクセスパターンによらずに PMem 上でシーケンシャルな書き込みに変換できるため，PMem のデバイス性能を引き出すのに適していると考える．

図 2 に pmembb の不揮発データ構造を示す．*Root Obj* は，*File Context* の配列へのポインタが格納されている．File Context は，グローバルファイルシステム上にあるファイルと対応するバーストバッファを紐付けるためのデータ構造で，ファイルパスと，バーストバッファのポインタの配列を持つ．各 File Context は，複数の Burst Buffer を持つことができる．pmembb pool の Open の際に File Context の配列をすべて列挙し，ファイルパスを key，File Context のポインタを value とした Runtime Key-Value を DRAM 上に作成する．これにより，以降のファイル Open の際の File Context の lookup を高速化する．

Burst Buffer は単方向リンクリスト構造になっており，先頭ノードへのポインタと，List Node から構成される．各 List Node は，アプリケーションからの書き込みリクエスト **pwrite** で書き込まれるデータを Log-structured 形式で保持する．新しい書き込みリクエストは，リストの末尾に追加される．従って先頭ノードは時間的に一番古い書き込みリクエストとなる．各 List Node は，次の List Node へのポインタ *next*，グローバルファイルシステム上のファイルの書き込み位置を示す *offset*，可変長の書き込みデータ *data* と，その *size* から構成される．next, offset, size はそれぞれ 16 bytes, 8 bytes, 8 bytes で，合計 32 bytes のメタデータ領域を持つ．メタデータ領域は List Node の初期化時に書き込み，その後キャッシュライン上のデータが PMem にフラッシュされて永続化が保証されるまで待つ．データ領域は，*non-temporal store* 命令を用いて，キャッシュに載せずに直接 PMem に書き込む．

データ構造内のポインタの付け替え操作やオブジェクトの alloc，free 等は，ソフトウェアトランザクショナルメモリを用いてアトミックに行う．これは，ポインタ付け替え中に計算機の電源が喪失した場合に，Root Object から辿れないオブジェクトが発生して，Persistent なメモリリークになってしまうのを防ぐために必要である．

Burst Buffer からグローバルファイルシステムへ書き込みデータを Flush する際は，リストを先頭から順番に列挙して書き込むことで，元の書き込み順を保持したまま Flush できる．Flush が完了した後，List Node を Burst Buffer から削除する．Flush の途中に計算機の電源が喪失した場合については，復旧後に Burst Buffer 内に残っているデータをもう一度 Flush して対応する．Flush が完全に完了するまでは List Node は PMem 上に保持され，List Node 削除操作の Power-fail Atomicity はソフトウェアトランザクショナルメモリによって保証される．さらに Flush は冪等な操作なので，同じデータを複数回 Flush しても問題ない．

排他制御に必要な Mutex オブジェクトやキャッシュデータ等は，PMem 上に持たずに揮発性の Runtime なデータとして DRAM 上に持ち，pool の Open の時や各オブジェクトへの初回アクセス時に初期化する．

3.3 MPI-IO-PMEM: ノードローカル PMem をバーストバッファとする MPI-IO

本節では，前節までに述べた pmembb を用いて，MPI-IO から直接ノードローカル PMem をバーストバッファとして利用する MPI-IO 実装，MPI-IO-PMEM の設計について述べる．

3.3.1 N-1 to N-N write アクセスパターン変換

MPI-IO-PMEM は，グローバルファイルシステム上のファイルに対する N-1 write アクセスを，ノードローカル PMem 上のバーストバッファへの N-N write アクセスに変換する．各アプリケーションプロセスは，それぞれ専有の pmembb pool を持つ．そして，グローバルファイルシステムの単一共有ファイルを集団 Open し，ファイルへの書き込みを，自身が専有する pmembb pool 内のバーストバッファに Log-strucutred 形式で追記し，永続化する．プロセスごとに並列にログ形式でノードローカル PMem に保存するので，N-N パターンになる．

MPI-4.0 の標準 [15] によると，MPI-IO のノンアトミックモードにおける，複数のプロセスから集団 Open したファイルの一貫性は，各プロセスのファイルアクセス範囲がオーバラップし，少なくとも 1 つが書き込みアクセスである場合はコンフリクトとなり，コンフリクトがある場合は逐次一貫性を保証しなくて良いという規定になっている．アトミックモードを必要とする HPC アプリケーションは稀であるため [16]，提案システムはアトミックモードはサポートしない．一貫性が必要となる場合は，*MPI_File_sync* を利用する．以下に変換手順の詳細を述べる．

最初に，MPI-IO-PMEM は MPI の各プロセスに専有の pmembb pool を割り当てる．アプリケーションのジョブが計算ノードに割り当てられる際に，1 ノードあたりの MPI プロセス数（ローカルランクサイズ）でノードローカル PMem を分割し，pmembb pool として初期化する．ノードローカル PMem の分割は，ndctl Linux utility の namespace 機能を利用するが namespace の利用が難しい

場合は，fsdax を用いて割り当てを行う．その場合は，MPI hints で指定する fsdax 上の pool path に，MPI ローカルランク番号をサフィックスとして付与したファイル名を pool path とする．MPI プロセスと pmembb pool の割り当ては，MPI のローカルランクに基づいて一対一で行う．

MPI-IO-PMEM はファイル Open の際にプロセスに割り当てられた pmembb pool に新規のバーストバッファを作成する．各 MPI プロセスは N-1 write リクエストをプロセス毎に作成したバーストバッファへ転送する．その際にグローバルファイルシステム上のファイルの書き込みオフセット情報を一緒に書き込むことで保持する．

3.3.2 SetInfo

MPI-IO-PMEM を使用する最に必要な情報（e.g. pmembb pool path）は，MPI hints を通して指定する．アプリケーションコード内から，*MPI_Info_set* 等を通じて指定するか，環境変数で指定する．基本的にはユーザが直接指定する必要はなく，システム管理者やジョブスケジューラの機能でグローバルなシステムヒントとして与える．以下に MPI hints の例を示す．

MPIO_PMEM_POOL_LIST
: CSV 形式で複数の pool path を指定する

MPIO_PMEM_POOL_PER_RANK
: enable or disable. pool path の末尾に，Local rank 番号のサフィックスをつけるかどうか

MPIO_PMEM_POOL_SIZE
: pool の大きさ．pool 初回作成時にのみ使用する

3.3.3 Open

Open では，まずグローバルファイルシステム上のファイルを Open する．次に MPI Local Rank に基づいて自プロセスに割り当てられた pmembb pool を Open する．既に pool を Open 済みの場合はそれを使用する．pool の Open 回数を参照カウントとして，カウンターに記録しておく．fsdax モードで pool がまだ作成されていない場合は，新規に pmembb pool を作成する．そして，pmembb pool 内に新規のバーストバッファを作成する．以降の Write や Flush は，この新規に作成したバーストバッファに対して行われる．

3.3.4 Write

Open 以降の Write 関数では，ファイルに対する I/O 要求をインターセプトし，各プロセスに割り当てられた pmembb pool 上のバーストバッファに対して書き込みを行う．アプリケーションからのファイルに対する書き込みリクエストは，ファイルのグローバルな byte オフセット位置に対する可変長の書き込みとして行われるが，そのデータを可変長のまま，オフセット情報付きで，バーストバッファに Log-structured 方式で追記する．

3.3.5 Flush

Flush では，バーストバッファから書き込みリクエストを取り出して，グローバルファイルシステム上のファイルに書き込みを行う．バーストバッファ内のデータはアプリケーションプロセスが書き込みを行った順序が保持されたまま並んでいるので，先頭から順に取り出すことで，ファイルへの書き込み順序を維持する．グローバルファイルシステムにデータを反映し終わったら，バーストバッファ上から書き込みデータを削除する．Flush の最適化として，まずバーストバッファから 1 度に取り出す書き込みリクエスト数の上限を決める．これにより，Flush 中の DRAM 使用量を抑えられる．次に，書き込みリクエストの offset 位置でソートを行う．2 つのリクエストの書き込み区間が重なっている場合は，後からきたリクエストで上書きになるように区間位置と長さを調節する．その後，offset の小さい順に書き込みリクエストを列挙して，書き込み区間が連続しているリクエストを 1 つにまとめてグローバルファイルシステム上のファイルに Flush する．

3.3.6 Read

Read ではまず Flush を呼び，バーストバッファ上のデータをすべてグローバルファイルシステムに Flush する．その後，グローバルファイルシステムから直接データを読む．

3.3.7 Close

Close では，バーストバッファを close する．ユーザは Close の前にバーストバッファ内に残っている書き込みデータを明示的に Flush するか，アプリケーション終了後，ジョブスケジューラの epilogue 等で，pool 内のデータを Flush するユーティリティプログラムに任せるか選択できる．pool の Open 回数のカウンターをデクリメントして，すべてのファイルが Close されたら pool を Close する．そして，グローバルファイルシステム上のファイルも Close する．

4. 実装

4.1 MPI-IO-PMEM

MPI-IO のインターフェースでノードローカル PMem をバーストバッファとして扱う MPI-IO の実装，MPI-IO-PMEM について述べる．MPI-IO-PMEM は，ADIO [14] を利用して実装している．ADIO は，並列 I/O インターフェスを移植可能かつ効率的に実装するための抽象化層であり，ADIO の規定するインターフェースに沿って特定のファイルシステムに最適化した I/O を実装すると MPI-IO から利用可能になる．提案システムは，ADIO から pmembb library を通してノードローカル PMem にアクセスを行う．ADIO は，MPI-IO の実装の一つである MPICH 内の ROMIO [17] から呼び出される．

4.2 pmembb library

ノードローカル PMem を SNIA NPM を利用してバーストバッファとするライブラリ，pmembb library の実装につ

いて述べる．pmembb は，the Persistent Memory Development Kit（PMDK）[18] の，libpmemobj-cpp を用いて実装している．このライブラリは PMem 内に不揮発なデータ構造を作成するために必要となる，PMem 内の不揮発ヒープからのメモリ割り当て，開放機能や，ソフトウェアトランザクショナルメモリを用いた，Power-fail Atomicity を保証するデータ構造の更新機能を提供している．提案システムでは，図 2 で示すデータ構造の，ポインタ付け替えや，各配列の確保やサイズ変更に PMDK のトランザクション機能を使用している．新たな List Node の確保には，PMDK の *Atomic operation* 機能を利用している．これは，1. PMem 内の不揮発ヒープからのメモリ割り当て，2. 割り当てたメモリ領域の初期化処理，3. 割り当てたメモリのアドレスを保存する PMem 上のポインタ変数の更新，の 3 つの処理をアトミックに行う機能で，通常のトランザクションと混ぜて使用できない代わりに，通常のトランザクションよりも軽量に動作する．

5. 評価

本節では，これまでに述べた MPI-IO-PMEM について，以下の評価を行う．

- ローカルファイルシステムとの N-1 書き込み性能比較
- シングルノードでの N-1 書き込み性能
- マルチノードでの合計 N-1 書き込み性能のスケーラビリティ

ローカルファイルシステムとの比較評価については，POSIX ローカルファイルシステムと MPI-IO-PMEM の N-1 書き込み性能の比較を行う．これは，アプリケーションが計算ノードローカルのストレージを利用する方法として，ローカルファイルシステムを利用するのは一般的に行われており，ノードローカル PMem においても POSIX ファイルシステムをマウントして利用すれば，既存のアプリケーションコードを変更することなく利用できるためである．

シングルノードでの評価については，シングルノード上で書き込みの並列度と書き込みリクエストのサイズを変化させた場合の，MPI-IO-PMEM の性能変化をより詳細に評価する．

マルチノードでの評価は，MPI-IO-PMEM のスケーラビリティを評価する．

5.1 評価環境とベンチマーク

性能評価に使用した計算機の仕様を**表 1** に示す．ノードローカル PMem は，物理コア数と同数の 16 namespaces に均等に分割する．そして分割した PMem は devdax モードで pmembb の pool として使用する．比較対象として，ノードローカル PMem を XFS+dax モードでマウントした POSIX ローカルファイルシステムを使用する．また，PMem namespace を sector モードに設定し，dax マウントオプションを指定しない XFS も比較する．データの Flush については，ノードローカル PMem 上に書き込んだデータが永続化されるまでを評価し，グローバルファイルシステムへの反映は行わない．

表 1 評価環境
Table 1 Evaluation Environment.

CPU	Xeon Gold 5218 2.30GHz
Sockets/node	1
Cores/socket	16
Num of Nodes	4
Kernel	Linux 3.10.0-1160.36.2.el7.x86_64
OS	CentOS Linux release 7.9.2009 (Core)
PMem	Intel Optane DC 128 GB * 6 slots
Network	InfiniBand HDR100

ベンチマークは，IOR [19] ベンチマークの single-shared-file モードを用いて，N-1 アクセスを行う．書き込みサイズは各 IOR プロセスあたり 10 GiB の Weak scaling 評価を行う．MPI-IO-PMEM の評価では，IOR の MPI-IO api（`-a MPIIO`）を使い，MPI-IO インターフェースを通してファイル書き込みを行う．POSIX ローカルファイルシステムの評価では，IOR POSIX api を使用する．Flush については，MPI-IO-PMEM の評価では IOR の fsync オプション（`-e`）を付与していない．これは，MPI-IO-PMEM は各書き込みリクエスト単位で PMem にキャッシュライン上のデータがフラッシュされるのを待っており，もしくは non-temporal store 命令によって，キャッシュを経由せずにデータをフラッシュしているのでデータの永続性は保証されているためである．POSIX ファイルシステムについては，`-e` オプションを使用し，データが PMem デバイスにフラッシュされ永続化されるのを待つ．

5.2 ローカルファイルシステムとの N-1 書き込み性能比較

図 3 に，シングルノード IOR の N-1 書き込みの評価結果を示す．横軸の Transfer size は，IOR の `-t` オプションで指定されるクライアントが発行する書き込みリクエスト 1 回あたりの書き込みデータサイズを表す．縦軸は合計書き込みバンド幅を示す．提案システムの MPI-IO-PMEM は，ローカルファイルシステムと比べて，すべての Transfer size において高い性能示した．XFS+dax のピーク性能は，8 MiB Transfer size の 2.36 GiB/s であった．dax 機能無しの XFS のピーク性能は，8 MiB Transfer size の 1.19 GiB/s であった．MPI-IO-PMEM のピーク性能は，8 MiB Transfer size の 10.35 GiB/s で，XFS+dax と比べて 4.39 倍 N-1 書き込み性能が向上した．

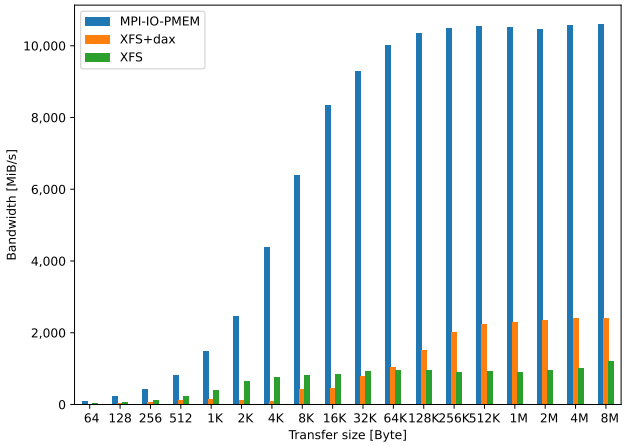

図 3 IOR N-1 書き込み，シングルノード（8 プロセス）
Fig. 3 IOR N-1 Write on single node (8 procs)

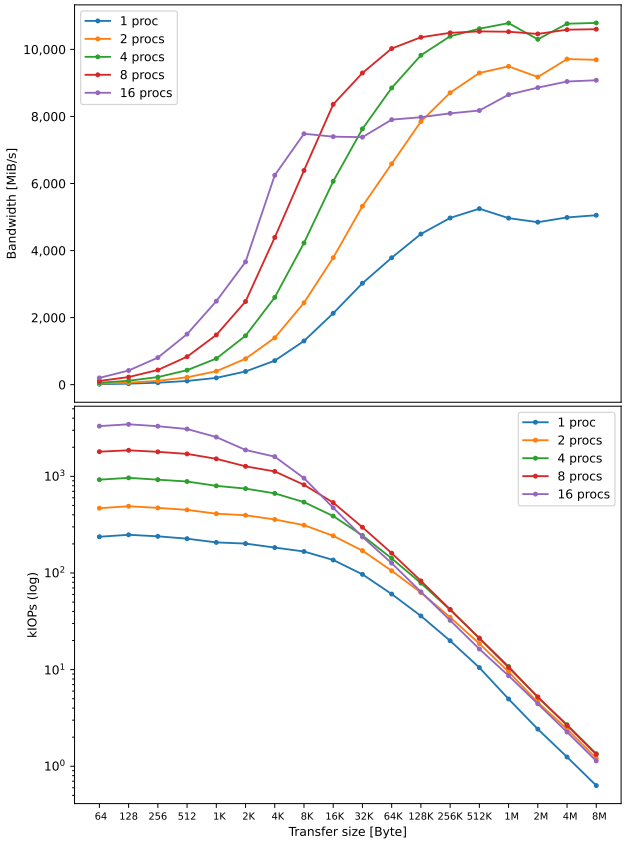

図 4 MPI-IO-PMEM IOR N-1 書き込み，シングルノード
Fig. 4 MPI-IO-PMEM IOR N-1 Write on single node

5.3 MPI-IO-PMEM のシングルノードでの N-1 write 性能評価

図 4 に，MPI-IO-PMEM を用いてシングルノード上で IOR を実行し，N-1 書き込みを行った場合の性能を示す．上グラフは縦軸がバンド幅を表し，下グラフは縦軸が kIOPs で，1 秒あたりの書き込みリクエスト発行を表す．横軸はどちらのグラフも Transfer size を示す．IOR のプロセスを，1 プロセスから 16 プロセスまで変化させ，書き込み並列度が変わった場合のバンド幅と kIOPs の変化を評価した．

4 プロセスから，8 MiB Transfer size で並列書き込みを行った場合が最もバンド幅が高く，10.5 GiB/s であった．1 プロセスの時と比較すると，4.94 GiB/s から 2.13 倍の性能向上となっている．8 プロセスの場合は，4 プロセスと同程度のピーク性能が得られており，Transfer size が 16 KiB–256 KiB の場合は，最も性能が良い．16 プロセスに増やした場合は，Transfer size が 8KiB 以下の場合は最も良い性能を示すが，16KiB 以上では 4 プロセスや 8 プロセスに比べて性能向上がゆるやかで，ピーク性能も 8 MiB Transfer size の時の 8.87 GiB/s となり，0.84 倍と低い性能を示した．これに関して，Optane PMem への並列書き込み性能は並列度を上げすぎるとむしろ減少するという報告 [1] があるため，PMem のデバイス特性に起因すると考えられる．

Transfer size が 4 KiB 以下の場合の kIOPs に注目すると，プロセス数を増やし並列度を上げた分だけ性能がスケールしている．一方で Transfer size を 64 Byte から 128 Byte に増加しても，kIOPs の値はほとんど変化していない．このことから，Transfer size が小さい場合，データを PMem へ転送する操作より，pmembb の List Node をアロケーションするコストの方が支配的になっており，アロケーションのサイズを多少大きくしても kIOPs はあまり低下しないため，アロケーション回数を抑える最適化を行えば，Transfer size が小さい場合の更なる書き込み性能向上ができると考えられる．

5.4 MPI-IO-PMEM のマルチノードでの N-1 write スケーラビリティ評価

本評価は，IOR のプロセスを複数ノードに分散し，MPI-IO-PMEM の N-1 書き込みのスケーラビリティを評価する．**図 5** に評価結果を示す．上グラフは縦軸がバンド幅を表し，下グラフは縦軸が kIOPs である．横軸はどちらのグラフも Transfer size を示す．IOR は 1 ノードあたり 8 プロセス起動し，1–4 ノードに変化させた場合の合計バンド幅と kIOPs を評価した．

合計バンド幅のピーク性能は，1–4 ノードの場合にそれぞれ 10.79 GiB/s，19.96 GiB/s，29.73 GiB/s，38.18 GiB/s で，並列化効率は，2 ノード 92.5 %，3 ノード 91.8 %，4 ノード 88.4 %であった．MPI-IO-PMEM によって，N-1 書き込みがノードローカル PMem の N-N 書き込みに変換され，高いバンド幅で性能がスケールした．

6. まとめ

本研究では，計算ノード上の不揮発性メモリを MPI-IO から直接活用するためのシステムである MPI-IO-PMEM と，PMem 内の不揮発バーストバッファデータ構造 pmembb の設計を行った．MPI-IO のデバイス抽象化レイヤー ADIO

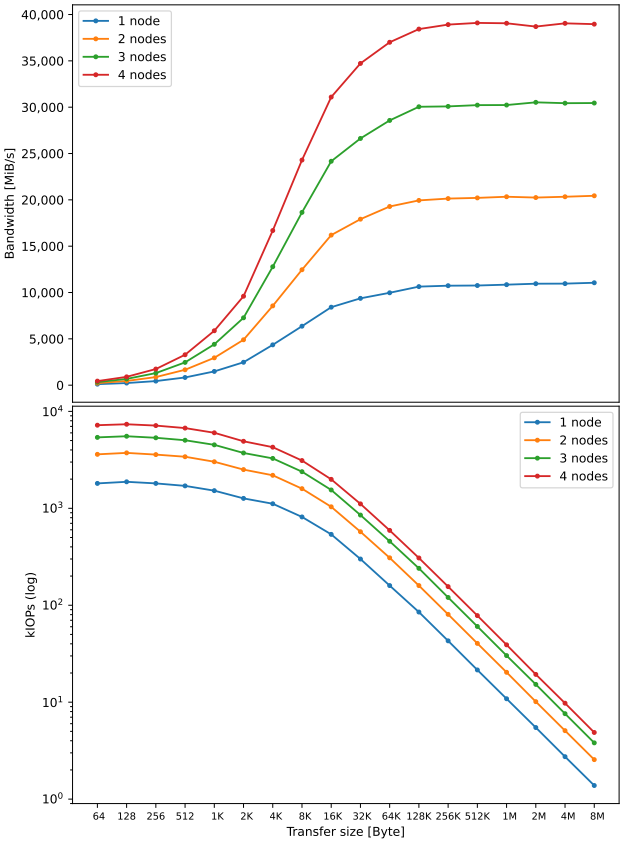

図 5 MPI-IO-PMEM IOR 複数ノードから N-1 書き込みした場合の合計バンド幅．8 procs/node

Fig. 5 MPI-IO-PMEM IOR Aggregate bandwidth when writing N-1 from multiple nodes. 8 procs/node

から SNIA NPM を用いて直接ノードローカル PMem を扱い，ノードローカルバーストバッファを構築する．IOR による N-1 書き込み性能評価では，シングルノードでは XFS+dax のローカルストレージと比較して 4.39 倍の 10.35 GiB/s の合計書き込みバンド幅を達成した．マルチノードでの N-1 書き込み性能は，4 ノードを用いた場合，38.18 GiB/s を達成し，スケーラブルなバンド幅を示した．

今後はより大規模環境でのスケーラビリティ調査を行いたいと考えている．また，バーストバッファからの直接データを読み込むための設計，実アプリケーション評価を行いたいと考えている．

謝辞 本研究の一部は，筑波大学研究基盤支援プログラム（A タイプ），筑波大学計算科学研究センターの学際共同利用プログラム（Cygnus），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）および富士通研究所との共同研究の助成を受けたものです．

参考文献

[1] Yang, J., Kim, J., Hoseinzadeh, M., Izraelevitz, J. and Swanson, S.: An Empirical Guide to the Behavior and Use of Scalable Persistent Memory, *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 169–182 (2020).

[2] SNIA: NVM Programming Model v1.2, `https://www.snia.org/sites/default/files/technical-work/npm/release/SNIA-NVM-Programming-Model-v1.2.pdf` (2017).

[3] Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M. and Wingate, M.: PLFS: A Checkpoint Filesystem for Parallel Applications, *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–12 (2009).

[4] Wang, T., Mohror, K., Moody, A., Sato, K. and Yu, W.: An Ephemeral Burst-Buffer File System for Scientific Applications, *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 807–818 (2016).

[5] IO500 Foundation: IO500, `https://io500.org/`.

[6] Carns, P., Harms, K., Allcock, W., Bacon, C., Lang, S., Latham, R. and Ross, R.: Understanding and Improving Computational Science Storage Access through Continuous Characterization, *ACM Transactions on Storage*, Vol. 7, No. 3, pp. 8:1–8:26 (10 月 1, 2011).

[7] Kimura, H. and Tatebe, O.: MPI-IO/Gfarm: An Optimized Implementation of MPI-IO for the Gfarm File System, *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 610–611 (2011).

[8] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, pp. 257–275 (2010).

[9] Sugihara, K. and Tatebe, O.: Design of Locality-Aware MPI-IO for Scalable Shared File Write Performance, *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, New Orleans, LA, USA, IEEE, pp. 1080–1089 (2020).

[10] Tatebe, O., Moriwake, S. and Oyama, Y.: Gfarm/BB — Gfarm File System for Node-Local Burst Buffer, *Journal of Computer Science and Technology*, Vol. 35, No. 1, pp. 61–71 (2020).

[11] Tatebe, O., Obata, K., Hiraga, K. and Ohtsuji, H.: CHFS: Parallel Consistent Hashing File System for Node-Local Persistent Memory, *International Conference on High Performance Computing in Asia-Pacific Region*, HPCAsia2022, New York, NY, USA, Association for Computing Machinery, pp. 115–124 (1 月 7, 2022).

[12] Vef, M.-A., Moti, N., Süß, T., Tocci, T., Nou, R., Miranda, A., Cortes, T. and Brinkmann, A.: GekkoFS - A Temporary Distributed File System for HPC Applications, *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 319–324 (2018).

[13] Pacific Teck Limited: BeeOND, `https://www.beegfs.io/wiki/BeeOND`.

[14] Thakur, R., Gropp, W. and Lusk, E.: An Abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces, *Proceedings of 6th Symposium on the Frontiers of Massively Parallel Computation (Frontiers '96)*, Annapolis, MD, USA, IEEE, pp. 180–187 (1996).

[15] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard Version 4.0, `https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf` (2021).

[16] Wang, C., Mohror, K. and Snir, M.: File System Semantics Requirements of HPC Applications, *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*,

HPDC '21, New York, NY, USA, Association for Computing Machinery, pp. 19–30 (6 月 21, 2021).

[17] Thakur, R., Gropp, W. and Lusk, E.: On Implementing MPI-IO Portably and with High Performance, *Proceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems*, IOPADS '99, New York, NY, USA, Association for Computing Machinery, pp. 23–32 (5 月 1, 1999).

[18] Scargall, S.: *Programming Persistent Memory*, Apress, Berkeley, CA, first edition (2020).

[19] Loewe, W., McLarty, T. and Morrone, C.: HPC IO Benchmark Repository, `https://github.com/hpc/ior` (2003).