jh230027

次世代演算加速装置とそのファイル IO に関する研究

塙 敏博（東京大学）

概要

GPU クラスタにおいて，実アプリケーションを効率よく実行するためには GPU に対するデータの入出力の考慮が必要であり，データ転送と演算のオーバラップ，転送レイテンシの短縮を工夫する必要がある．そこで本研究では，GPU 上データの直接ファイル IO である GPUDirect Storage (GDS)，あるいは計算とファイル IO のオーバラップの両者を容易に取り扱い可能にする手法を確立し，様々なファイル入出力特性を持つ実アプリケーションにおいて GPU-ファイル IO 間の処理を効率化することを目的とする．今年度は，N 体シミュレーションに基づくベンチマーク `h5gds` を作成した．また HDF5 の既存の GDS プラグイン vfd-gds を拡張し，IO の性質に基づいて GDS に限らず最適なファイル転送を実現した．その結果，既存手法と比較してローカルファイルシステムにおいては最大 4.33 倍，リモートファイルシステムにおいては最大 6.09 倍のバンド幅向上が得られた．

1 共同研究に関する情報

1.1 共同研究を実施した拠点名

- 東京大学 情報基盤センター
- 名古屋大学 情報基盤センター
- 大阪大学 サイバーメディアセンター
- mdx

1.2 課題分野

- 大規模計算科学課題分野

1.3 共同研究分野 (HPCI 資源利用課題のみ)

- 超大規模情報システム関連研究分野

1.4 参加研究者の役割分担

塙 敏博 (東大・代表): 全体取りまとめ, GPU 通信，機械学習
建部 修見 (筑波大・副代表): ストレージ技術，機械学習
三木 洋平 (東大): 宇宙物理コード
佐藤 拓人 (筑波大): City-LES コード
星野 哲也 (名古屋大): GPU プログラミング
河合 直聡 (名古屋大): 数値アルゴリズム
中島 研吾 (東大): 計算科学アプリケーション
住元 真司 (東大): ストレージ技術
山崎 一哉 (東大): 計算科学アプリケーション
富永 瑞己 (東大): ストレージ技術
小林 諒平 (筑波大): GPU 通信，IO
藤田 典久 (筑波大): GPU 通信，IO
朴 泰祐 (筑波大): GPU 通信，IO
平賀 弘平 (筑波大): ストレージ技術、機械学習
小山 創平 (筑波大): ストレージ技術、機械学習

2 研究の目的と意義

HPC システムにおいて，アクセラレータとして GPU が広く利用されているが，実アプリケーションでは本質的に GPU に対するデータ

の入出力の考慮が必要であり，データ転送と演算のオーバラップ，転送レイテンシの短縮を工夫する必要がある．近年 NVIDIA によって，GPUDirect RDMA をファイル IO に拡張した GPUDirect Storage (GDS) が提供されるようになり、IO オーバヘッドの短縮が期待される．本研究は，GPU 上データの直接ファイル IO，あるいは計算とファイル IO のオーバラップの両者を容易に取り扱い可能にする手法を確立し，様々なファイル入出力特性を持つ実アプリケーションにおいて GPU-ファイル IO 間の処理を効率化することを目的とする．さらにこの成果を，最先端共同 HPC 基盤施設 (JCAHPC) の次期システム “Oakforest-PACS II (OFP-II)” の設計に反映し，OFP-II および次世代 GPU クラスタにおいて利用可能な，ファイル IO を考慮したベンチマークを実装する．

3 当拠点公募型研究として実施した意義

本研究では，計算科学の実アプリにおけるファイル IO の観点で，宇宙物理コードの GOTHIC，気象コードの City-LES を対象にし，各分野の専門家を交えて研究を実施した．さらに PyTorch などの機械学習フレームワークも対象にしている．本研究で利用した NVIDIA A100 GPU は，Wisteria/BDEC-01 Aquarius，mdx，SQUID で採用されている．当初は，Wisteria/BDEC-01 Aquarius および不老 Type II において GDS の利用を想定していたが，GPU ドライバやストレージの制約等により実現できなかった．一方で，mdx においては，VM を採用しており，ハードウェアとしては抽象化され，必ずしもデバイスに対して最適な構成であるか不可視な部分はあるものの，必要に応じた OS や所望のドライバ構成を取ることができ，実際に性能の改善が見られることも確認できた．本研究では，別途 AMD EPYC Milan CPU, NVIDIA A100 80GB PCIe GPU, H100 80GB PCIe GPU を搭載する GPU サーバ，および筑波大学計算科学研究センターに導入されている Intel Sapphire Rapids CPU, NVIDIA H100 80GB PCIe GPU を搭載する Pegasus システムも使用して実施している．

4 前年度までに得られた研究成果の概要

前年度は，N 体シミュレーション，都市気候シミュレーション，機械学習トレーニングのそれぞれについて GDS の利用について実装検討，一部については予備評価を行った．その結果，GDS は多くの場合に高い性能を示し，mdx では VM 環境であっても GDS が非 GDS に対して最大 1.67 倍の性能を示した．

5 今年度の研究成果の詳細

5.1 N 体シミュレーション

HDF5 から GDS を用いたファイル入出力性能を測定するベンチマークプログラム `h5gds` を実装した．本ベンチマークプログラムでは N 体シミュレーションにおける典型的なデータ構造を想定し，GPU メモリ上に確保された粒子位置と質量を格納する `float4` 型の配列，粒子速度の x 成分と y 成分を格納する `float2` 型の配列，粒子速度の z 成分を格納する `float` 型の配列，粒子 ID を格納する `uint64_t` 型の配列のデータ入出力を `H5Dread_multi()` 及び `H5Dwrite_multi()` で行った際の実行時間を測定する．また，`h5gds` については投稿論文が採択された後に公開する予定である．

データ入出力方式については asis と hyperslab の 2 つのモードがある．Asis モードにおいては，N 体計算時に用いる `float4` 型や `float2` 型の配列データをそのまま入出力する．

Hyperslab モードは，実際のシミュレーション実行及びポストプロセスでの解析処理における利便性を考慮し，HDF5 の hyperslab 機能（MPI における派生データ型に類似の機能）を用いて，粒子位置と質量を格納する `float4` 型の配列を位置データを格納する $N \times 3$ 成分の配列と粒子質量の配列に分離して出力する実装である．Hyperslab モードにおいては，粒子位置を格納する $N \times 3$ 要素の `float` 型の配列，粒子速度を格納する $N \times 3$ 要素の `float` 型の配列，粒子質量を格納する N 要素の `float` 型の配列，粒子 ID を格納する N 要素の `uint64_t` 型の配列が出力ファイルに書き込まれ，またこのファイルを GPU から読み出すこととなる．

図 1 に，筑波大学計算科学研究センターが運用する Pegasus 上での性能測定結果を示す．性能評価に使用した GPU は NVIDIA H100 PCIe，データ入出力先の NVMe SSD は Micron 7400 MTFDKCB3T2TFC である．ソフトウェア環境としては，CUDA 12.1, GDS 1.6.0.25, HDF5 1.14.1-2, HDF5 Nvidia GPUDirect Storage VFD 1.0.2 を用いた．Asis モードのデータ出力時においては，$N \gtrsim 10^5$ においては GDS を用いた方が太いバンド幅が得られることが確認された．一方で，$N \lesssim 10^5$ およびデータ入力時においては常に GDS を無効化した互換モード使用時の方が高性能だった．

Hyperslab モードの性能は asis モードに比べて極めて低く，また粒子数 N に対する依存性もほぼないことが分かった．CPU のみを用いて同じ処理を実装した際には，hyperslab モードの性能は asis モードよりも若干低いものの，粒子数 N に対する依存性は共通であったことから，HDF5 Nvidia GPUDirect Storage VFD の実装が最適でないことが主たる要因であると考えられる．

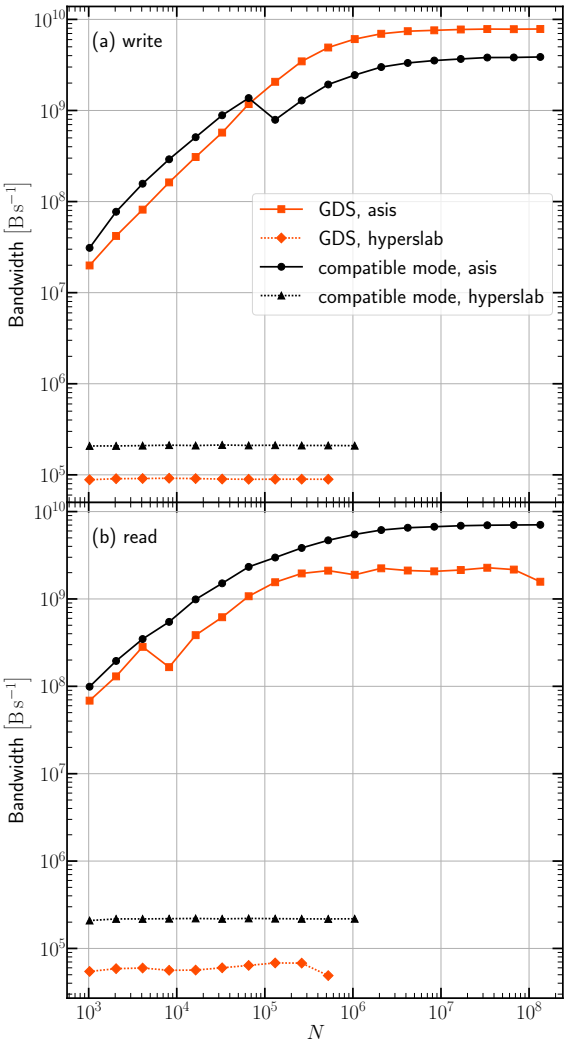


図 1: GDS 使用の有無によるデータ入出力性能の比較．上段にデータ書き出し時のバンド幅を，下段にはデータ読み込み時のバンド幅を，それぞれ粒子数 N の関数として示した．赤が GDS 使用時，黒が GDS 無効化時の性能である．実線が asis モード，点線が hyperslab モードの性能である．

大阪大学サイバーメディアセンターが運用する SQUID 上で `h5gds` を動作させたところ，GDS を無効化する compatible mode では正常に動作したものの，GDS を有効化した native mode では動作しなかった．これは，GDS を動作させるために必要な nvidia-fs がシステムにインストールされていなかったためである．

5.2 都市気象シミュレーション

今年度は, 昨年度までテストプログラムで実装していた出力のオーバーラップを CityLES(CPU-ver) を簡単化したベンチマークに実装した．また, フラグ管理の容易さ等を考慮して `task` 文を用いた実装に関しても検討した．ラックマウントサーバ (研究室所有) を用いて小規模な問題で予備調査を行った．試験計算は, 格子点数 128^3 および 256^3 の領域を対象とし, 100 ステップの流体計算と, 10 ステップごとの出力を行うワークロードを設定した．実行は MPI 4 プロセス, OMP5 スレッドで行った．試験計算においては, 計算・出力がオーバーラップすることを確認できたものの, オーバーラップによって性能が劣化する結果が得られた．これには複数の理由が考えられる．

一つは, 出力をオーバーラップさせるために, 流体計算部においても出力用のスレッドを 1 スレッド確保しなければならず, 流体計算の性能が劣化したことである．オーバーラップを止めた従来のベンチマークを用いて実行時間を計測したところ, 256^3 ケースで平均 5.1 sec 程度の性能劣化が確認された．これは, 出力オーバーラップのために流体計算開始時から出力用のスレッドを確保しているために起こっているため, スレッドの割り当てを実行状況によって変更する等で回避可能と思われる．もうひとつは, 出力部のコール (task 文によるタスク並列) のために `taskprivate` 節を用いて配列値を保持した上で task を生成するため, このオーバーヘッドが大きくなることである．実際に task 生成部の実行時間のみを取り出すと, 1 回の出力あたり 128^3 ケースでは 0.4sec, 256^3 では 3.5 sec 程度のオーバーヘッドであった．いずれの問題についても, 問題サイズが大きくなる, すなわち全体実行時間に占める出力の時間の割合が大きくなることで顕在化しにくくなる可能性は高いと考えているものの, 回避のため, 例えば `taskprivate` による出力値の保持ではなく, 別途出力バッファを確保する等の方法を検討している．

大規模な問題でのテスト実行のため, オーバーラップを施したベンチマークを試験的に Wisteria/BDEC-01 Odyssey にて実行を試みた．オーバーラップを施したベンチマークは流体計算部と出力部の両方で同時に MPI 通信を発行する可能性がある．そのため, 流体計算部と出力部でそれぞれの MPI コミュニケータを用意している．複数スレッドで同時に MPI 通信を発行するためには, `mpi_thread_multiple` がサポートされている必要があるが, 富士通 MPI ではサポートされておらず, 実現できなかった．様々なアーキテクチャに対応できるように CPU 版をベースに出力オーバーラップを行う場合は, より細かな MPI 通信の制御が必要な可能性があるだろう．

5.3 機械学習

機械学習フレームワークにおける GDS 利用に向けた評価環境を構築した．

NVIDIA によって公開されている Data Loading Library (DALI: `https://github.com/NVIDIA/DALI`) は、データの読み込みおよび前処理を行うライブラリであり，画像、ビデオ、オーディオ等のデータセットを効率よく行う機能を提供する．PyTorch, Tensorflow など機械学習向けのフレームワークでは，データローダやデータイテレータを備えており, 通常,

データロードとそれに伴うデコードや画像の整形などの処理は CPU で行われる．DALI はこれらの機能をオーバーライドして，GPU 上にオフロードする．さらに、DALI は、入力パイプラインのスループットを向上させるため，プリフェッチ、並列実行、バッチ処理などの機能を持ち，ユーザからは透過的に処理が行われる．DALI は TensorFlow, PyTorch, MXNet, PaddlePaddle をサポートしており，学習や推論のワークフローに対して高い移植性を備えている．この中で，GDS は DALI のデータローダにおいて用いられている．

一方，KvikIO（`https://github.com/rapidsai/kvikio`）は，NVIDIA によって提供されているライブラリであり，cuFile によって提供される GDS の機能を，C++ や Python から容易に使えるようにしたものである．KvikIO では，ホストおよび GPU から，GDS の利用可能の有無に関わらずシームレスに動作する機構を提供している．DALI とは独立に開発されているものと考えられ，両者を組み合わせてより高い性能を実現できる可能性がある．

最新の MLPerf 3.1 においては，音声認識 RNN-T の training においてのみ DALI が使用されており，以前のバージョンよりも対応が減っているようである．実際に mdx を使って環境を構築し，GDS の有効/無効での性能を比較したが，結果として顕著な差が見られなかった．

図 1 でわかるように，読み出しの場合にファイル転送サイズが大きくなると，GDS の方が却って性能が低下しており，training においては大量のデータセットを読み出すことから，GDS があまり有効ではないものと思われる．

一方，KvikIO は，HPC Training の deepCam において適用された例があることはわかった．今後は KvikIO にも調査を広げていきたいと考えている．

5.4 HDF5 向け GDS プラグインの拡張

昨年度までに，GDS が高い性能を示す場合があることは分かったが，常に性能が高いわけではない．様々な条件のもとで測定を行った結果，ファイル読み出し/書き込みの転送の向き，ローカル NVMe SSD/InfiniBand 経由の Lustre ファイルシステムといった種類，ファイル転送サイズ，IO のスレッド数，といったパラメータに依存して，どの転送方法が最善か細かく変動することがわかった．

しかし，実際に GDS を使うためには，リスト 1 をリスト 2 のように書き換える必要がある．ユーザのプログラム中で各条件を判断し GDS とその他の手法を切り替えるのは困難である．そのため，ユーザプログラムでの記述は抽象化しておき，ライブラリなどにより吸収するのが望ましい．

HDF5（Hierarchical Data Format 5）は，HPC のアプリケーションで広く使用されているミドルウェアライブラリであり，データ管理や高度な IO パフォーマンスを提供している．VFD (Virtual File Driver) によりプラグインを提供する仕組みが存在する．GDS に対応したものとして HDF5 GPUDirect Storage VFD が存在しているが，常に GDS を使うような設計がされており，効率が良いと言えない．そこで我々は，VFD の中でファイルアクセスパターンに応じた転送方式の切り替えを行うことで，常に最適な性能が得られるように拡張することとした．このアプローチにより、以下のメリットが得られる．

- 抽象性：ユーザは、転送方式の切替、転送長、スレッド数などの設定に関して意識することなく、VFD 内部でこれらの処理が

自動的に行われる。VFD がこれらのパラメータを自動的に高速なファイル IO になるように調整し、ユーザは単にヒントを与えるだけで、高速なファイル IO が実現可能である．

- 拡張性：VFD を HDF5 アプリケーションにリンクするだけで，アプリケーションの既存の構造や主要なコードに手を加えることなく、VFD を改良することができる．そのため、VFD の改良やアップデートを容易に導入でき、拡張性・柔軟性がある．

改良した VFD では，現時点ではユーザがファイルサイズとファイルシステムなどのヒントを与えることにより，これらの調整を抽象化し，高速なファイル IO を実現することを目標に設計する．将来的にはこれらを完全に自動化することを目標にする．

gdsio 等の測定結果を元に，以下のように VFD プラグインを実現した．

- GDS・CPU コピーによる転送方式が高速になる条件の特定：
  Pegasus 上でアプリケーションベンチマークを実行し、ファイル IO が高速になるときの転送方式や IO パラメータを特定した．その中で、ファイルサイズおよびファイルシステムによって，ファイル IO の性能が大きく異なることを発見した．表 1 は、GDS と CPU コピーによる転送方式の結果を比較し、高速なときの転送方式と IO パラメータをまとめたものである．例えば，ファイルサイズが 3 MB 以下の場合は，CPU コピーによる転送方式が GDS より高速な傾向があり，ファイルサイズが 10 MB 以上の場合は，スレッド数を上げた GDS が高速な傾向が得られた．
- ユーザフレンドリな HDF5 プラグインの導入：
  ユーザが VFD に対してマニュアルでヒントを与えることで，容易にファイル IO を高速化できるようなフレームワークを提案・実装した．これにより，アプリケーションのコードを大きく変更することなく、ファイル IO を高速化することができた．結果として，図 2a,2b に示すように既存手法と比較したときにローカルファイルシステムにおいては最大 4.33 倍，リモートファイルシステムにおいては最大 6.09 倍のバンド幅向上がみられ，ファイル IO の高速化に寄与した．

```
int fd = open(...)
void *sysmem_buf, *gpumem_buf;
sysmem_buf = malloc(buf_size);
cudaMalloc(gpumem_buf, buf_size);
pread(fd, sysmem_buf, buf_size);
cudaMemcpy(sysmem_buf, gpumem_buf, buf_size
    ↪ , H2D);
...
```

リスト 1: POSIX API によるプログラム記述例

```
CUFileHandle_t *fh;
CUFileDescr_t desc;
void *gpumem_buf;
int fd = open(file_name, O_DIRECT,...)
desc.type=CU_FILE_HANDLE_TYPE_OPAQUE_FD;
desc.handle.fd = fd;
cuFileHandleRegister(&fh, &desc);
cudaMalloc(&gpumem_buf, buf_size);
cuFileRead(&fh, gpumem_buf, buf_size, …);
...
```

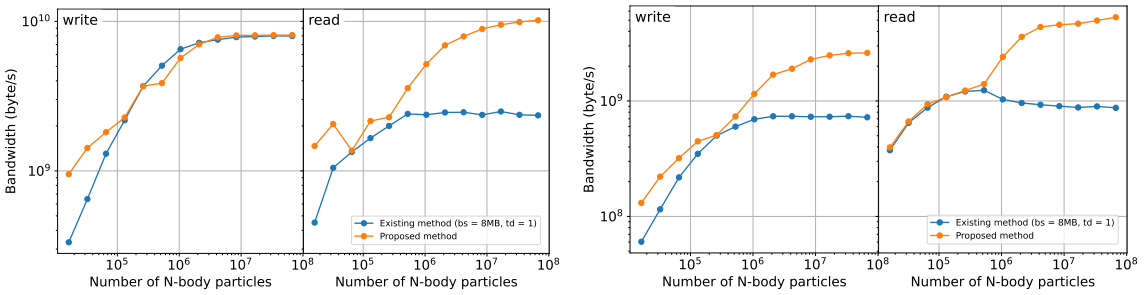
リスト 2: cuFile API によるプログラム記述例 ([1] を修正)

参考文献

[1] NVIDIA Magnum IO GPUDirect Storage Overview Guide, https://docs.nvidia.com/gpudirect-storage/overview-guide/index.html

表 1: バンド幅が高速なときの転送方式および IO パラメータ (Pegasus)

N (size)	アクセス 種類	Local FS (NVMeSSD)	Remote FS (Lustre FS)
16384–65536 (579 KB–2.3 MB)	write read	CPU copy, Block size: 4KB CPU copy, Block size: 4KB	CPU copy, Block size: 4KB GDS, Block size: 4MB, Thread: 1
131072-262144 (4.6 MB–9.1 MB)	write read	GDS, Block size: 4MB, Thread: 1 GDS, Block size: 16MB, Thread: 1	CPU copy, Block size: 4KB GDS, Block size: 4MB, Thread: 1
524288-67108864 (19 MB–2.3 GB)	write read	GDS, Block size: 16MB, Thread: 1 GDS, Block size: 16MB, Thread: 8	GDS, Block size: 16MB, Thread: 8 GDS, Block size: 16MB, Thread: 8



(a) ローカルファイルシステムにおける性能比較　(b) リモートファイルシステムにおける性能比較

図 2: 提案手法と既存の GDS VFD のファイル IO 性能比較

6 今年度の進捗状況と今後の展望

6.1 *N* 体シミュレーション

今年度目標としていたスパコン調達における性能評価試験のためのベンチマークテストの作成については，`h5gds` の実装によって達成できたと言える．この `h5gds` については，現在投稿中の論文が採択された際に速やかに公開できるように公開準備を進めている．また，HDF5 の hyperslab 機能と GDS を組み合わせた際には IO 性能が極端に低下することが確認された．Hyperslab と GDS は実アプリケーションの動作において共存できるべき機能であるため，今後の VFD などの改善によって解決していく必要のある課題である．

6.2 都市気象シミュレーション

今年度はベンチマークソフトを対象に計算-出力のオーバーラップを実装し，限定的ではあるが性能の測定を行った．その結果，単なるオーバーラップではオーバーヘッドの増大や計算性能の劣化等によって必ずしも性能向上が見込めない可能性があることが明らかとなった．今後はオーバーヘッドの軽減や，様々なアーキテクチャへの対応を見越した MPI の制御等を検討する必要があるだろう．また，GPU 版への実装は未着手であるため，これの実装を進める予定である．

6.3 機械学習

今年度は DALI を用いた MLPerf training において GDS の効果を調査したが，ほとんど効果が見られないことが分かった．来年度は KvikIO についても調査し，機械学習における GDS の効果的な利用法を検討する予定である．

6.4 HDF5 向け GDS プラグインの拡張

今年度は HDF5 に GDS 機能を含めファイルアクセス性能を最適化する vfd-gds の拡張を

実現した．今後は，GDS および CPU コピーによる転送方式の非同期・マルチスレッド化を含めた最適化の余地がある．また，スレッド数や転送長，ファイルサイズなどをより広範な範囲で設定し，実アプリケーションを実行したときのファイル IO 特性を調査する必要がある．これらの課題に取り組むことで，更なるファイル IO の高速化が期待できる．

本プラグインにおける最適パラメータは，システムのアーキテクチャが異なれば変わるはずである．2025 年 1 月から運用開始される JCAHPC の Miyabi (OFP-II) においても，パラメータの確認および実際にユーザにも提供する予定である．他のシステムについても GDS が利用可能な環境に対してパラメータ同定および導入に協力したいと考えている．

7　研究業績一覧（発表予定も含む）

学術論文 （査読あり）

- 富永 瑞己, 塙 敏博, 三木 洋平,「GPU 直接 IO を用いたファイル IO の高速化」, xSIG 2024, 2024 年 8 月 (発表予定) (Best Master's Student Award 受賞)

国際会議プロシーディングス （査読あり）

- M. Tominaga, T. Hanawa, and Y. Miki, "Toward Optimizing File IO on GPU Clusters," GPU Technical Conference 2024, Poster, Mar. 2024.

国際会議発表 （査読なし）

国内会議発表 （査読なし）

- 富永 瑞己, 塙 敏博, 三木 洋平, 「GPU クラスタにおけるファイル IO 性能評価」、情報処理学会ハイパフォーマンスコンピューティング研究会, 2023 年 8 月
- 塙 敏博, 他,「GH200 における予備性能評価」, 情報処理学会ハイパフォーマンスコンピューティング研究会 (SWoPP2024), 2024 年 8 月 (発表予定)

公開したライブラリ等

その他（特許，プレス発表，著書等）