

# ストレージ上のデータベースに対するCPUとGPUを併用した基本処理の実行方式に関する予備実験

三浦 優也<sup>1,a)</sup> 小沢 健史<sup>1,b)</sup> 合田 和生<sup>1,c)</sup>

**概要:** GPUはCPUに比べて大きな帯域幅と高い計算速度を有しており、データ処理の文脈においても有効活用される動きが高まっている。従来、ストレージ上のデータをGPUで処理するためにはストレージ-CPUメモリ間とCPUメモリ-GPUメモリ間のデータ転送が必要であったが、GPUDirect Storageの登場により、ストレージ-GPUメモリ間の直接のデータ転送が可能になった。しかし、GPUDirect Storageを用いたGPU直接I/Oの定量的な性能については未だ明らかになっていない。本論文では、GPU直接I/Oの基本的な性能を測定するマイクロベンチマークの結果を示し、CPU I/Oの性能と比較する。また、CPU I/O、GPU間接I/O、GPU直接I/Oを用いて実装したTPC-H問合せ実験の結果を示し、それぞれ比較する。

## 1. はじめに

ビッグデータの台頭とストレージのコストの低下により、世界中で生み出されるデータの総量はここ数十年間で指数関数的に増加している。データを管理するためのデータベースシステムは、身近にあって欠かせない基盤であるが、このかつてないデータ量の増加に対応することが求められる。今でもプロセッサやストレージの処理速度は年々向上しているものの、ムーアの法則は物理的制約によって限界を迎えてきている。データベースを高速化する方法として、ハードウェア単体の処理能力を上げるスケールアップ設計では限界があるといえる。

同時に、GPUやNVMe SSDといった、データベースシステムの歴史に比べれば新しいハードウェア技術が登場してきているのも事実である。こうした新しいハードウェア技術は、新しいパフォーマンス体系、コスト体系、計算モデルを有している。データベースシステムのようなソフトウェアとハードウェアが密接に関係する技術は、新しいハードウェア技術を用いた再設計によって、さらなる効率の向上を見込むことができる。

2021年、NVIDIAがGPUDirect Storage [1]の提供を開始した。従来、GPUでストレージ上のデータを処理するためには、ストレージ-CPUメモリ間とCPUメモリ-GPU

メモリ間の2回のデータ転送が必要だった。GPUDirect Storageの登場によって、ストレージ-GPUメモリ間の直接のデータ転送を行うことが可能になる。このGPU直接I/Oによって、データベースシステムの設計に新たな選択肢が加わる。

しかし、GPU直接I/Oの定量的な性能については、未だ明らかになっていない。GPU直接I/Oを活用してデータベースシステムを設計するために、その基本的な性能を知ることは欠かせない。本稿では、GPU直接I/Oの基本的な性能を測定するマイクロベンチマークの結果を示し、CPU I/Oの性能と比較する。また、CPU I/O、GPU間接I/O、GPU直接I/Oを用いて実装したTPC-H問合せ実験の結果を示し、それぞれ比較する。

本論文の構成は以下の通りである。第2章では、GPUによるデータ処理やGPU直接I/Oを紹介する。第3章では、CPU I/OとGPU直接I/Oの基本的な性能を測定するマイクロベンチマークの結果を示す。第4章では、CPU I/O、GPU間接I/O、GPU直接I/Oを用いた実装によるTPC-H問合せ実験の結果を示す。第5章では、GPUやGPUDirect RDMAを用いたデータベースシステムの研究を紹介し、第6章において本論文をまとめる。

## 2. GPUとI/O

### 2.1 GPUデータ処理

Graphics Processing Unit (GPU)は元来、画像処理に特化したプロセッサである。CPUと比べると、より小型で定形処理に特化したコアを多数搭載しており、高い並列

<sup>1</sup> 東京大学 生産技術研究所  
Institute of Industrial Science, The University of Tokyo  
a) ymiura@tkl.iis.u-tokyo.ac.jp  
b) ozawa@tkl.iis.u-tokyo.ac.jp  
c) kgoda@tkl.iis.u-tokyo.ac.jp

性能を誇る。そのため、コア間でタスクを分割して処理することさえできれば、CPU よりも格段に高い計算速度と帯域幅を発揮する。

画像処理でない分野において当初は、タスクをグラフィックシェーダーの形で GPU に与えることで GPU を活用しようという試みがあった。その後、NVIDIA が 2007 年に統合開発環境 CUDA [2] を発表したことなどにより、GPU を用いた汎用計算 (general-purpose computing on GPUs, GPGPU) が、特に高性能計算 (high-performance computing, HPC) や機械学習の分野を筆頭として盛んになってきた。

こうした流れを受けて、データベースシステムの研究においても GPU を活用する動向が見られはじめた。しかし、GPU をデータベースシステムの設計に活用する際には、以下のような制約が存在する。

- (1) GPU が扱うデータが GPU のメモリ内に収まる必要がある。GPU のメモリ容量は CPU に比べて小さいので、どのデータを GPU のメモリに配置し、CPU のメモリに退避するか戦略を立てる必要がある。
- (2) GPU でストレージ上のデータを扱うために CPU のメモリを介する必要がある。CPU のメモリと GPU のメモリとの間でのデータ転送にはコストがかかるため、これを極力避ける必要がある。

こうした制約を踏まえ、近年の GPU を用いたデータベースシステムの研究としては、CPU と GPU のヘテロジニアスな環境を意識したものが多い。

## 2.2 GPU 直接 I/O

GPUDirect Storage [1] は NVIDIA が 2021 年に提供を開始した技術で、GPU メモリと NVMe ストレージとの直接のデータのやりとりを追加のハードウェアなしに可能にするものである。図 1 は従来の GPU 間接 I/O と、GPUDirect Storage による GPU 直接 I/O との違いを示している。ストレージから GPU メモリにデータを読み込むとき、従来は図 1(a) のように、ストレージから CPU メモリにデータを読み込み、さらに CPU メモリから GPU メモリにデータを転送する必要があった。GPUDirect Storage の登場によって、図 1(b) のように、ストレージから CPU メモリを介さずに GPU メモリにデータを直接読み込むことができる。GPU メモリからストレージへの書き込みについても同様に CPU メモリを介さずに行うことができる。

GPU 直接 I/O は、データベースシステムの設計に新しい方針をもたらす。従来の GPU を用いたデータベースシステムは、CPU が I/O を行い、GPU が演算処理を行うものである。ここで GPU 直接 I/O の登場によって、GPU が I/O も演算処理も行う設計、あるいは GPU が I/O を行い CPU が演算処理を行うという従来とは逆の設計を選択することも可能となる。

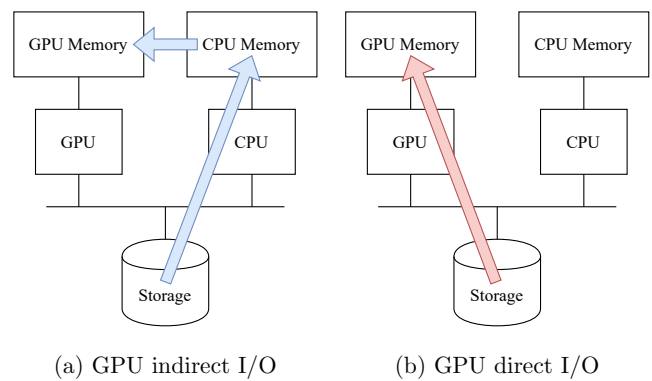


図 1: 2 種類の GPU I/O 方式の比較

Fig. 1 Comparison of two types of GPU I/O methods

しかしながら、GPU 直接 I/O がどのような恩恵をもたらすのかは現在、明らかではない。具体的には、GPU 直接 I/O が、性能、速度、効率といった面で CPU I/O や GPU 間接 I/O を置き換えることができるものなのかということである。これに関する定量的な研究は、著者らの知る限りにおいて確認されていない。GPU 直接 I/O をデータベースシステムの設計に活用するにあたっては、こうした基礎的な性能を明らかにすることが重要である。

## 3. 小規模環境におけるマイクロベンチマーク実験

著者らは、CPU I/O と GPU 直接 I/O の基本的な性能を測定する実験を行った。

### 3.1 マイクロベンチマークの実装

GPU 直接 I/O を発行する方法は GPUDirect Storage の cuFile API によって定義されており、現時点では以下の 3 種類が存在する。

**同期 I/O** CPU で GPU I/O を発行した後、その I/O が完了するのを待つ。cuFile API ではホスト関数 cuFileRead と cuFileWrite を用いて実現することができる。

**非同期 I/O** CPU で GPU I/O を発行すると、その I/O がキューに入れられる。CPU はその I/O の完了を待たずに次の処理に移る。結果は CUDA Stream API によって得ることができる。cuFile API ではホスト関数 cuFileReadAsync と cuFileWriteAsync を用いて実現することができるが、本稿執筆時点ではこれらのインターフェイスは実装されていない。

**バッチ I/O** CPU で複数の GPU I/O を同時に発行することができる。発行された I/O はキューに入れられ、適切な順番で実行される。cuFile API ではホスト関数 cuFileBatchIOSubmit を用いて実現することができるが、本稿執筆時点ではベータ版として提供されている。

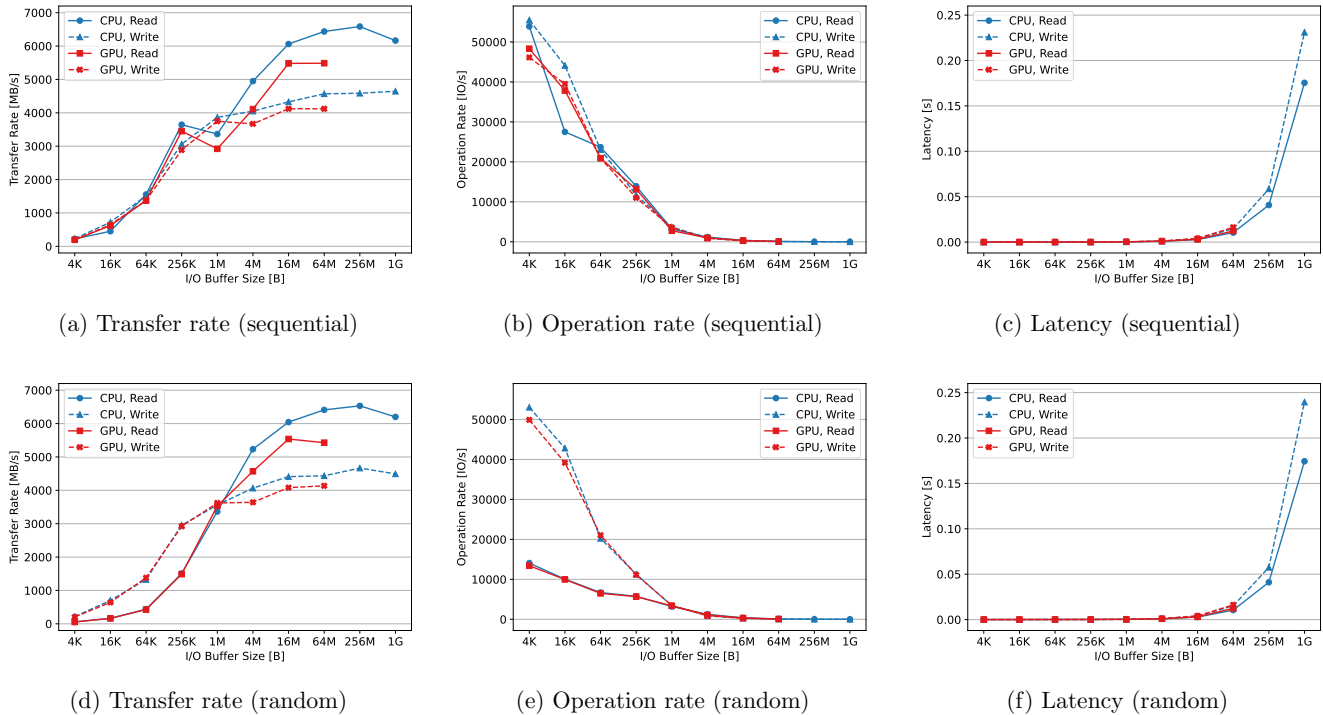


図 2: シングルスレッドにおける CPU I/O と GPU 直接 I/O の性能  
Fig. 2 Performances of single-threaded CPU I/O and GPU direct I/O

本実験では、GPUDirect Storage を用いた GPU 直接 I/O の特性を知るために、最も原始的な操作である同期 I/O 命令を取り上げて、その性能を測定した。比較対象としては、CPU の同期 I/O 命令である `pread`, `pwrite` を用いた。

### 3.2 実験環境

上述のマイクロベンチマークを、表 1 に示す環境で行った。

ワークステーション	HP Z4 G4 Workstation
CPU	Intel® Xeon® W-2245 CPU @ 3.90 GHz
メモリ	システムメモリ 64 GiB
NVMe SSD	WD_BLACK AN1500
GPU	NVIDIA RTX A4000
OS	Ubuntu 22.04
ドライバ	Mellanox OFED 5.8
ライブラリ	CUDA 11.7

表 1: 実験環境

Table 1 Experimental environment

### 3.3 シングルスレッド実験

図 2 は、GPU 直接 I/O と CPU I/O とで同等の I/O 性能が得られたことを示している。図 2(a) はストレージ上のデータに対して I/O バッファの大きさを変えてシーケンシャルアクセスを行ったときの転送レート (MB/s) を

示している。CPU I/O は read と write とともに、I/O バッファを大きくするにつれて転送レートが上昇していくが、I/O バッファの大きさが 64 MB 程度になると飽和する。GPU I/O も同様に 64 MB 程度で飽和が見られ、上昇の傾向も CPU I/O と近い。GPU I/O で 256 MB 以降のデータがないのは、GPU のメモリ上にその大きさのバッファを確保できなかったためである。図 2(b) は I/O 発行レート (IO/s)、図 2(c) はレイテンシ (s) を示している。これらについても GPU 直接 I/O と CPU I/O とで同等の数値が得られた。同様に図 2(d)、図 2(e)、図 2(f) も、ランダムアクセスを行ったときの転送レート (MB/s)、I/O 発行レート (IO/s)、レイテンシ (s) が GPU 直接 I/O と CPU I/O とで同等の性能であることを示している。

### 3.4 マルチスレッド実験

図 3 は、CPU マルチスレッド下で I/O を多重に発行しても、GPU 直接 I/O と CPU I/O とで同等の性能が得られたことを示している。図 3(a)、図 3(b) はストレージ上のデータに対して複数の CPU スレッドから 1 MB 固定のシーケンシャルな I/O を発行したときの転送レート (MB/s)、レイテンシ ( $\mu$ s) を示している。CPU I/O と GPU 直接 I/O のどちらについても、read 性能は 4 スレッド程度、write 性能は 2 スレッド程度で飽和している。飽和したときの数値は CPU I/O と GPU 直接 I/O とで同等である。同様に図 3(c) と図 3(d) は 4 KB 固定のランダム

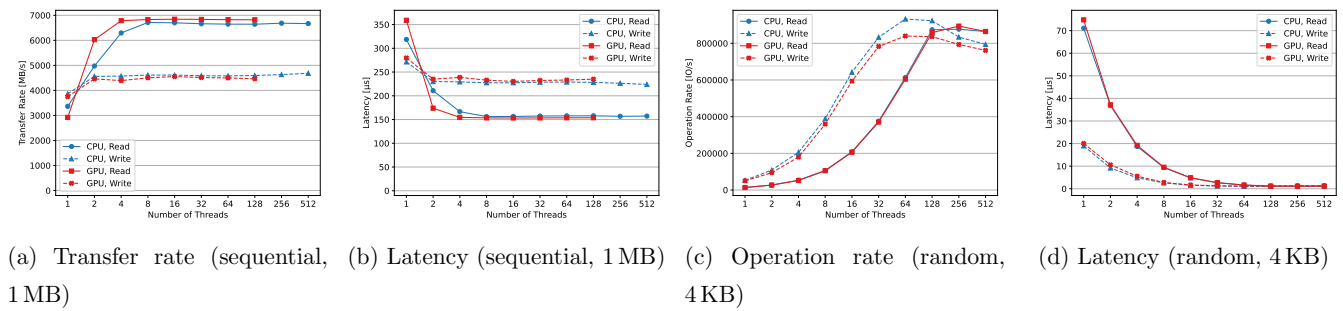


図 3: マルチスレッドにおける CPU I/O と GPU 直接 I/O の性能  
Fig. 3 Performances of multi-threaded CPU I/O and GPU direct I/O

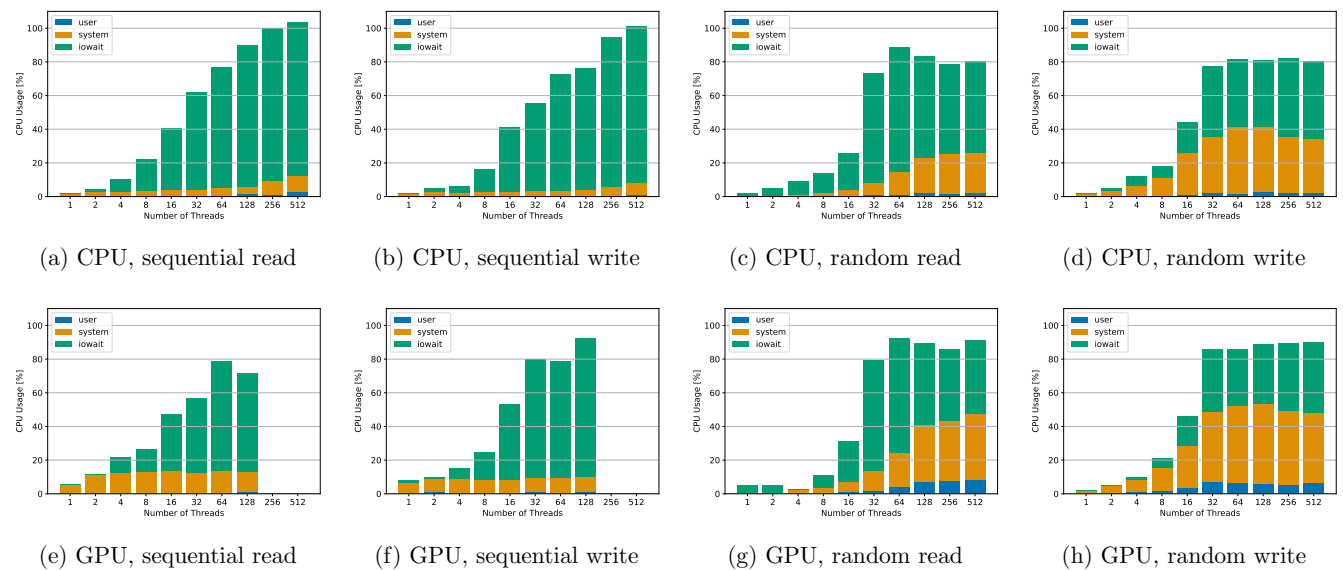


図 4: マルチスレッドにおける CPU I/O と GPU 直接 I/O の CPU 使用率  
Fig. 4 CPU usage of multi-threaded CPU I/O and GPU direct I/O

な I/O を発行したときの転送レート (MB/s) とレイテンシ ( $\mu\text{s}$ ) である。

一方で図 4 は、GPU 直接 I/O が CPU I/O よりも多くの CPU 計算資源を消費したことを示している。図 4(a), 図 4(b), 図 4(c), 図 4(d) は、比較対象となる CPU I/O を用いてシーケンシャルアクセスとランダムアクセスを行ったときの、CPU スレッド数ごとの CPU 使用率である。とくにシーケンシャルアクセスにおいては、ユーザー時間とシステム時間はスレッド数を増やしても低いままであり、CPU 時間のほとんどは I/O 待ち時間として消費された。それに対して、図 4(e), 図 4(f), 図 4(g), 図 4(h) は GPU 直接 I/O を行ったときの CPU スレッド数ごとの CPU 使用率である。全体の CPU 時間は CPU I/O のものと同じ傾向を示しているが、シーケンシャルアクセスとランダムアクセスの両方について、ユーザー時間とシステム時間が占める割合が大きくなっている。

## 4. 小規模環境における TPC-H 問合せ実験

著者らは、CPU I/O, GPU 間接 I/O, GPU 直接 I/O をそれぞれ用いて、TPC-H の問合せを模した処理を実装し、その実行性能を測定する実験を行った。

### 4.1 GPU 直接 I/O を用いた問合せ実行器の試作

本実験は、表 1 に示したものと同一環境下で行った。実行する問合せとして、TPC-H ベンチマーク [3] の Query 6 を選択した。Query 6 の内容は図 6 の通りである。さらに TPC-H の dbgen を用いて、scale factor = 100 のデータを生成した。Query 6 の実行には LINEITEM 表が必要であり、このとき生成された LINEITEM 表の行数は 59,986,052 行であった。

また、問合せ処理を実行するために、以下の 3 種類の実装を試作した。

#### CPU I/O, CPU 実行

A1: pread を用いて CPU メモリにページを読み込む。

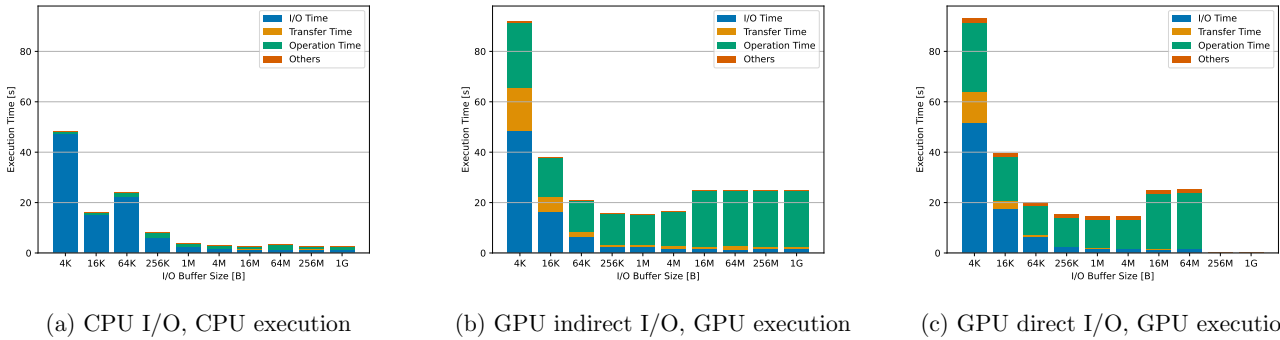


図 5: 3 種類の実装における Query 6 の実行時間  
Fig. 5 Execution time of Query 6 in three implementations

```
select
  sum(l_extendedprice * l_discount) as revenue
from
  lineitem
where
  l_shipdate >= date '1994-01-01'
  and l_shipdate < date '1994-01-01' + interval
    '1' year
  and l_discount between .06 - 0.01 and .06 +
    0.01
  and l_quantity < 24;
```

図 6: 実験に用いた Query 6  
Fig. 6 Query 6 used in the experiment

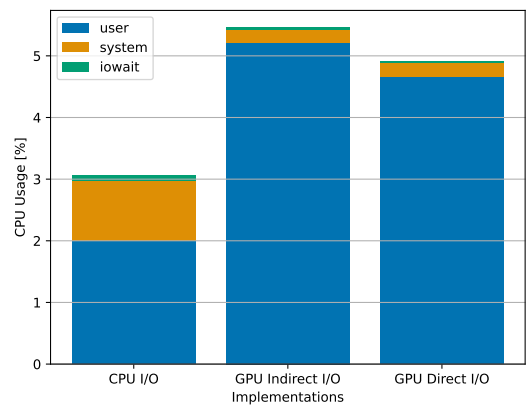


図 7: 3 種類の実装における Query 6 実行時の CPU 使用率  
Fig. 7 CPU usage during Query 6 execution in three implementations

A2: CPU で問合せ処理を実行する。

**GPU 間接 I/O, GPU 実行**

- B1: pread を用いて CPU メモリにページを読み込む。
- B2: cudaMemcpy を用いて CPU メモリから GPU メモリにページを転送する。
- B3: GPU で問合せ処理を実行する。
- B4: cudaMemcpy を用いて GPU メモリから CPU メモリに結果を転送する。

**GPU 直接 I/O, GPU 実行**

- C1: cuFileRead を用いて GPU メモリにページを読み込む。
- C2: GPU で問合せ処理を実行する。
- C3: cudaMemcpy を用いて GPU メモリから CPU メモリに結果を転送する。

ただし、CPU 側と GPU 側のいずれの処理も、シングルスレッド処理として実装した。

**4.2 Query 6 の実行性能**

図 5 は、各実装における問合せ実行時間の内訳を示している。各実装において、問合せ実行時間のうち A1, B1, C1 を I/O 時間、B2, B4, C3 を転送時間、A2, B3, C2 を演算時間と定めている。図 5(a) は比較対象となる CPU I/O, CPU 実行の実装における I/O バッファの大きさ

ごとにかかった実行時間である。I/O バッファの大きさによらず、I/O 時間がほとんどを縮めていて、演算時間の割合はわずかであった。図 5(b) も比較対象となる GPU 間接 I/O, GPU 実行の実行時間である。これは図 5(a) に転送時間を加え、処理時間を GPU 実行の時間に置き換えたものであるが、どちらも I/O 時間に匹敵する時間がかかった。図 5(c) は、GPU 直接 I/O, GPU 実行におけるの実行時間を示している。これは図 5(b) の転送時間を減らし、I/O 時間を GPUDirect Storage の処理時間に置き換えたものであるが、ページの転送がなくなったにもかかわらず、結果を返す転送処理だけでオーバーヘッドがかかった。また、I/O 時間、演算時間、その他初期化等にかかる時間が若干増えたことにより、全体の実行時間としては GPU 間接 I/O とほぼ変わらなくなった。他に注目すべき点として、図 5(b) と図 5(c) の両方で、バッファの大きさが 1MB 以下のときはバッファが大きくなるにつれて実行時間が減少したが、4MB を超えると逆に実行時間が増加した。これは、本実験に用いた GPU の L2 キャッシュの大きさが 4MB であることと関わっていると考えられる。

図 7 は、各実装における問合せ実行時の CPU 使用率を

示している。GPU 間接 I/O と GPU 直接 I/O では、CPU I/O の場合に比べてシステム時間が減ったもののユーザー時間が増え、全体の CPU 時間としては上昇した。GPU 直接 I/O では、GPU 間接 I/O の場合に比べて少しだけユーザー時間が減少した。

## 5. 関連研究

### 5.1 GPU を用いたデータベースシステム

Yogatama らが提案する Mordred [4] は、CPU と GPU 間のデータ配置と問合せ実行を工夫することにより、全体のトラフィックを最適化し、CPU と GPU の並列性を高めた。データ配置ポリシーとしては、GPU へのキャッシュを細かい粒度に分け、コストモデルに基づいた計算を行い、GPU にキャッシュされていてほしいデータを高い精度で GPU に配置した。異種問合せ実行としては、細かく分けたデータがどう配置されているかによって問合せ実行を分割し、CPU と GPU の両方を活用した。

他にも近年の研究では、処理配置 [5] [6]、問合せコンパイラ [7] [8]、問合せ実行 [9] [10] などに注目した手法がある。共通する傾向としては、CPU と GPU が協力するヘテロジニアスなデータベースシステムを意識した研究が多い。

### 5.2 GPUDirect RDMA を用いたデータベースシステム

GPUDirect Storage を用いたデータベースシステムは著者らの知る限りにおいて存在しないが、同じく NVIDIA が提供する技術である GPUDirect RDMA [11] を用いた研究がいくつか存在する。GPUDirect RDMA は、GPU メモリ上の remote direct memory access (RDMA) を可能にする技術である。Wang ら [12] は、resource description framework (RDF) の問合せを並列で高速に実行するシステムを GPUDirect RDMA を用いて実装した。Guo ら [13] は、GPUDirect RDMA を用いて GPU クラスタ上で動作する高速なハッシュ結合とソートマージ結合アルゴリズムを提案した。

## 6. おわりに

本稿では、GPU 直接 I/O の基本的な性能を測定するマイクロベンチマークの結果を示し、CPU I/O の性能と比較した。また、CPU I/O、GPU 間接 I/O、GPU 直接 I/O を用いて実装した問合せ処理について、その性能を測定するベンチマークの結果を示し、それぞれ比較した。

今後の展望としては、GPU 直接 I/O を、GPU の演算能力を有利に活かす並列処理とともに用いて、CPU I/O と CPU 処理や GPU 間接 I/O と GPU 処理といった組み合わせの場合と比較していきたい。また、GPU 直接 I/O を活かすためのデータベースシステムの設計についても論じていきたい。

**謝辞** 本研究の一部は、日本学術振興会科学研究費補助金基盤研究 (B) JP20H04191 の助成を受けたものである。

## 参考文献

- [1] NVIDIA GPUDirect Storage. <https://docs.nvidia.com/gpudirect-storage/index.html>. Accessed: 2022-12-3.
- [2] CUDA C++ Programming Guide. <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html>. Accessed: 2022-10-25.
- [3] TPC-H Homepage. <https://www.tpc.org/tpch/>. Accessed: 2022-12-4.
- [4] Bobbi W Yogatama, Weiwei Gong, and Xiangyao Yu. Orchestrating data placement and query execution in heterogeneous CPU-GPU DBMS. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 15, No. 11, pp. 2491–2503, July 2022.
- [5] Sebastian Breß, Henning Funke, and Jens Teubner. Robust Query Processing in Co-Processor-accelerated Databases. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16*, pp. 1891–1906, New York, NY, USA, June 2016. Association for Computing Machinery.
- [6] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. Adaptive work placement for query processing on heterogeneous computing resources. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 10, No. 7, pp. 733–744, March 2017.
- [7] Henning Funke, Sebastian Breß, Stefan Noll, Volker Markl, and Jens Teubner. Pipelined Query Processing in Coprocessor Environments. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18*, pp. 1603–1618, New York, NY, USA, May 2018. Association for Computing Machinery.
- [8] Periklis Chrysogelos, Manos Karpathiotakis, Raja Appuswamy, and Anastasia Ailamaki. HetExchange: encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 12, No. 5, pp. 544–556, January 2019.
- [9] Johns Paul, Shengliang Lu, Bingsheng He, and Chiew Tong Lau. MG-Join: A Scalable Join for Massively Parallel Multi-GPU Architectures. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD '21*, pp. 1413–1425, New York, NY, USA, June 2021. Association for Computing Machinery.
- [10] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. Triton Join: Efficiently Scaling to a Large Join State on GPUs with Fast Interconnects. In *Proceedings of the 2022 International Conference on Management of Data, SIGMOD '22*, pp. 1017–1032, New York, NY, USA, June 2022. Association for Computing Machinery.
- [11] Developing a Linux Kernel Module using GPUDirect RDMA. <https://docs.nvidia.com/cuda/gpudirect-rdma/index.html>. Accessed: 2022-10-26.
- [12] Siyuan Wang, Chang Lou, Rong Chen, and Haibo Chen. Fast and concurrent RDF queries using RDMA-assisted GPU graph exploration. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '18*, pp. 651–664, USA, July 2018. USENIX Association.
- [13] Chengxin Guo, Hong Chen, Feng Zhang, and Cuiping

Li. Distributed Join Algorithms on Multi-CPU Clusters with GPUDirect RDMA. In *Proceedings of the 48th International Conference on Parallel Processing*, No. Article 65 in ICPP 2019, pp. 1–10, New York, NY, USA, August 2019. Association for Computing Machinery.