GPU コードから分散ファイルシステムにアクセスする試み

松宮 遼　髙橋 一志　大山 恵弘

我々は分散ファイルシステム上のファイルを GPU コードで読み書きする API を開発している．この API を開発する上で，いくつかの課題が浮上している．GPU コードから CPU コードの呼び出す方法が通常の GPU プログラミング環境には存在しないこと，PCI Express のバス速度が低速であること，クライアント内で余分なメモリコピーが発生することである．本稿は，それらの課題に対して我々が取った対策について論じるものである．

We are developing an API to operate files on a distributed file system from GPU code. Several issues are found during this development; the limitation of bus speed of PCI Express, no method to call CPU code from GPU code in standard GPU software development kits, and extra data copies in the client. This paper describes our solutions to these issues.

1 はじめに

TSUBAME-KFC [5] のように，Graphic Processing Unit (GPU) を積極的に利用したクラスタマシンがある．そのようなマシンは，気象シミュレーション [20] から深層学習 [4] に至るまで幅広く利用されており，それらの分野で扱われる問題規模は日々増大している．事実，Microsoft では 10〜100 TB もの入力データを用いた深層学習を行っている [3]．

巨大なデータを扱う場合，ファイルは ext4 や btrfs [17] といったローカルファイルシステムではなく，Gfarm [24] や HDFS [21]，GlusterFS [16]，PVFS [2] といった分散ファイルシステムで管理することが一般的である．分散ファイルシステムの一部は，CPU で実行されるコード (CPU コード) からファイルの読み書きを行うためのクライアント API を独自に提供している．しかしそれらのクライアント API では，GPU で実行されるコード (GPU コード) からファイルの読み書きを直接行うことができない．GPU 上でファイルの読み書きを行おうとした場合，計算機はファイルの内容を一度マザーボード上のメインメモリ (ホストメモリ) 上に転送し，その後，メインメモリ上のデータを GPU 上の VRAM (デバイスメモリ) に転送することが一般的である．このデータ転送処理はソースコードを複雑にする．その上でプログラムの最適化を行うことにより，ソースコードはより複雑となる．

そこで我々は GPU コード上で分散ファイルシステム上のファイルを読み書きする機構を API として提供することを提案し，実装を行っている．提案機構の実装を行う上で，いくつかの課題が浮かび上がった．本稿では，これらの課題を取り上げた上で，我々が取ろうとしている解決策について述べる．本稿で取り上げる課題と解決策とは，以下の 3 つである．

- 近年の GPU は PCI Express (PCIe) によって接続されていることが多い．PCIe のバス速度は，GPU コアからデバイスメモリに対するバス速度に比べて十分に低速である．我々はデバイスメ

Attempting to access a distributed file system from GPU code.

Ryo Matsumiya, 電気通信大学, The University of Electro-Communications.

Kazushi Takahashi and Yoshihiro Oyama, 電気通信大学 / 科学技術振興機構 CREST, The University of Electro-Communications / CREST, Japan Science and Technology Agency.

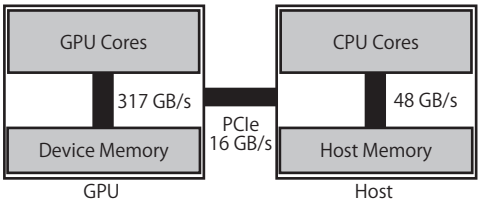

図 1　デバイスメモリとホストメモリの関係,
値は本稿で述べる実験環境の理論値.

モリ上にファイルの内容をキャッシュさせるようにした.

- 一般的な GPU プログラミング環境は，CPU コードから GPU コードを呼び出す機構のみが備わっており，GPU コードから CPU コードを呼び出す機構は，現状備わっていない．しかしながら，分散ファイルシステムサーバへの接続を行う時は CPU コードが実行される必要がある．したがって，我々は GPU コードから CPU コードを呼び出す Remote Procedure Call (RPC) 機構の実装を行った.
- GPU が読み書きを行うデータについては，通常，ホストメモリとデバイスメモリの間でメモリコピーが発生する．我々は，ファイルの読み込みを行う際，GPUDirect RDMA [11] を用いることでホストメモリとデバイスメモリのコピーを削減した．メモリコピーの削減により，ファイルの読み込み速度が向上することを確認した.

本稿の構成を述べる．2 章では背景となる GPU について説明する．3 章では我々が開発している API の設計と実装について，4 章では我々が開発している API の現在の性能に関する実験について述べる．5 章では我々が開発している API の移植性や，実験で得られた知見についての議論を行う．6 章で関連研究について説明し，7 章でまとめと今後の課題について述べる.

2 GPU

2.1 GPU のメモリ階層

図 1 で示した通り，最近の GPU は PCIe によって接続されている．つまり，CPU コアがデバイスメモリにアクセスする場合や，GPU コアがホストメモリにアクセスする場合は，PCIe のバスを経由する必要がある．このアクセス速度は，CPU コアがホストメモリにアクセスする速度や，GPU コアがデバイスメモリにアクセスする速度に比べて十分に遅い．従って，GPU を用いたプログラムでは，デバイスメモリのみを使って処理を行うことが一般的である.

```
__kernel plus_kernel(a[], b[], c[])
{
  c = a + b;
}

plus(a[], b[], c[])
{
  d_a = gpuMalloc(sizeof(a));
  d_b = gpuMalloc(sizeof(b));
  d_c = gpuMalloc(sizeof(c));
  memcpyHtoD(d_a, a, sizeof(a));
  memcpyHtoD(d_b, b, sizeof(b));
  kernelLaunch(plus_kernel, d_a, d_b,
      d_c);
  memcpyDtoH(c, d_c, sizeof(c));
  gpuFree(d_a);
  gpuFree(d_b);
  gpuFree(d_c);
}
```

図 2　GPU による一般的な行列加算プログラムの
擬似コード

2.2 GPU のプログラミングモデル

CUDA [9] や OpenCL [14] といった一般的な GPU プログラミング環境でのプログラミングモデルについて述べる．GPU を用いた一般的な行列加算プログラムの例を擬似コードとして図 2 に示す．関数 `plus_kernel()` は 2 つの行列 `a`，`b` を受け取り, その和を行列 `c` に書き込むものである．関数 `plus_kernel()` は GPU 上で処理される．GPU 上で処理される関数はカーネル関数とよばれる．関数 `plus()` は，関数 `plus_kernel()` を GPU 上で実行するための処理を行う関数で，CPU 上で実行される．関数 `plus()` ではまず，関数 `gpuMalloc()` を用いて，GPU の処理に必要なメモリ領域をデバイスメモリ上に確保する．確保されたメモリ領域には関数 `memcpyHtoD()` によっ

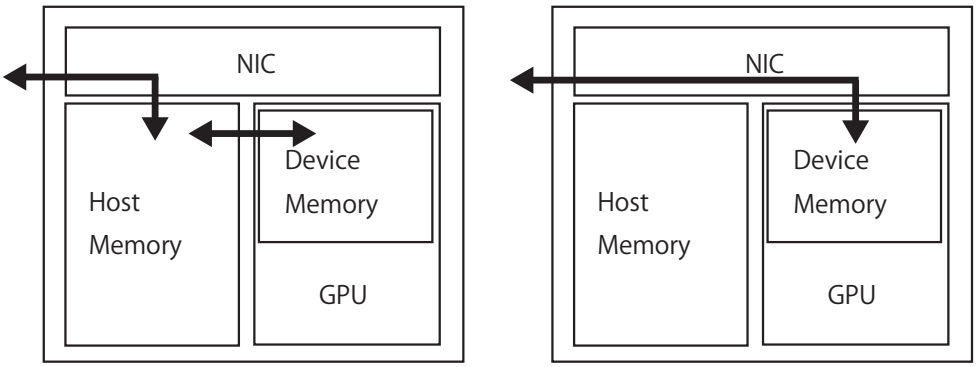

図 3　(左)**GPUDirect RDMA** を利用していない場合のデータのフロー
(右)**GPUDirect RDMA** を利用した場合のデータのフロー

て，ホストメモリ上にある入力データがコピーされる．カーネル関数の呼び出しは，通常の関数の呼び出しとは異なる．例では関数 `kernelLaunch()` を用いて関数 `plus_kernel()` を呼び出している．カーネル関数の実行結果はデバイスメモリ上に保存され，カーネル関数の実行が終わった後にホストメモリに転送される．例では，関数 `memcpyDtoH()` を用いて，カーネル関数によって書き込まれた行列の和を，ホストメモリ上にコピーしている．コンパイラは，関数に与えられているアノテーションをもとに CPU コードか GPU コードのどちらにコンパイルするかを決定する．例では，`__kernel` アノテーションを関数 `plus_kernel()` に与えることで，GPU コードとしてコンパイルするようにコンパイラに指示している．GPU コードから CPU コードの関数を呼び出す機構は現在のところ標準では実装されていない．

2.3　GPUDirect RDMA

ネットワーク上のデータをデバイスメモリに転送する，あるいはデバイスメモリ上のデータをネットワーク上のノードに転送する場合，図 3 の左で示すように，ホストメモリを一度経由する必要がある．GPUDirect RDMA と呼ばれる NVIDIA 製の一部の GPU に実装されている機能を用いることで，図 3 の右で示すようにホストメモリを経由することなく，デバイスメモリの内容をネットワーク上のノードから読み書きすることができる．GPUDirect RDMA では，GPUDirect RDMA に対応した NVIDIA 社製の GPU とネットワーク環境が必要となる．また，GPUDirect RDMA を用いて一度に転送できるデータサイズには限度がある [10]．そのため，GPUDirect RDMA を用いたプログラムの開発者はそのデータサイズを考慮しなければならない．

3　設計と実装

3.1　API の仕様

本研究では，分散ファイルシステムの Gfarm に対して我々のチームが開発している RDMA 実装 [18] を対象とし，GPU コード上で Gfarm 上のファイルを読み書きする API を実装した．API を実行するマシンは，NVIDIA 社製の，GPUDirect RDMA に対応した GPU を搭載したもので，NVIDIA 社公式の GPU プログラミング環境である CUDA によってプログラムが開発されていると仮定する．

CUDA では多くの場合 C あるいは C++でソースコードを記述する．そこで我々が提供する API は C 言語によるものである．Gfarm では POSIX-like の API を libgfarm として公式に提供している．我々の API も POSIX-like のものとした．現在までに実装が完了しており，API で提供している関数の一覧を表 1 および表 2 に示す．表 1 は CPU コード上で呼び出される関数であり，表 2 は GPU コード上で呼び出される関数である．

開発中の API を用いて，行列加算プログラムのカー

表 1　CPU コードで実行する API 関数の一覧

関数名	説明
ginitialize	API の内部処理に必要なメモリ領域の確保と Gfarm サーバへの接続を行う.
ghandle	カーネル関数の実行を開始した後に呼び出す. この関数が呼び出される前に実行を開始したカーネル関数が全て終了するまで待機する.
gfinalize	ginitialize で確保したメモリ領域の解放と Gfarm サーバとの接続の切断を行う.

表 2　GPU コードで実行する API 関数の一覧

関数名	説明
gopen	与えられた URL に対応する Gfarm 上のファイルを開き，ファイルの識別子を返す.
gwrite	与えられたファイルオフセットに対してファイルの書き込みを行う.
gread	与えられたファイルオフセットに対するファイルの読み込みを行う.
gclose	与えられた識別子に対応するファイルを閉じる.
gfstat	与えられた識別子に対応するファイルのサイズを返す.

```
__kernel plus_kernel(a[], b[], c[])
{
  int fd_a, fd_b, fd_c;
  gopen("File_a", O_RDONLY, &fd_a);
  gopen("File_b", O_RDONLY, &fd_b);
  gopen("File_c", O_WRONLY, &fd_c);
  gread(fd_a, a, 0, gfstat(fd_a));
  gread(fd_b, b, 0, gfstat(fd_b));
  c = a + b;
  gwrite(fd_c, c, 0, gfstat(fd_c));
  gclose(fd_a);
  gclose(fd_b);
  gclose(fd_c);
}
```

図 4　開発中の API を用いた行列加算プログラムの
カーネル関数擬似コード

ネル関数を記述すると図 4 のようになる．このカーネル関数では，まず，File_a，File_b，File_c というファイル名のファイルを関数 gopen() で開く．入力値の記述されたファイル File_a，File_b の内容を関数 gread() でファイルの先頭 (オフセットが 0) からファイルサイズ分だけ読み込む．ファイルサイズは関数 gfstat() にて取得する．その後行列の加算処理を行い，結果を関数 gwrite() を用いてファイル File_c に書き込む．最後に 3 ファイル全てを閉じる．

3.2　キャッシュの実装

2.1 節で述べたように，最近の GPU は PCIe で接続されている．GPU コアからホストメモリへのバス速度は，GPU コアからデバイスメモリへのバス速度に比べて遅い．

そこで我々はデバイスメモリ上にファイルの内容をキャッシュさせるようにした．我々が実装したキャッシュ機構は Linux カーネルで利用されているページキャッシュと同様に，固定長のものである．キャッシュ戦略は，実装が容易である先入れ先だし (FIFO) 戦略を取った．

3.3　RPC 機構の実装

ネットワーク機器の制御を行うには，OS が提供しているシステムコールを使う必要がある．つまり，提案機構の実現のためには GPU コードから CPU コードを呼び出す RPC 機構を実装する必要がある．しかしながら，2.2 節で述べたように，GPU コードから CPU コードの関数を呼び出す RPC 機構は標準では実装されていない．

そこで我々は GPU コードから CPU コードの関数を呼び出す RPC 機構を実装した．この RPC 機構は GPUfs [22,23] や GPUnet [8] で用いられているものと同様のものである．

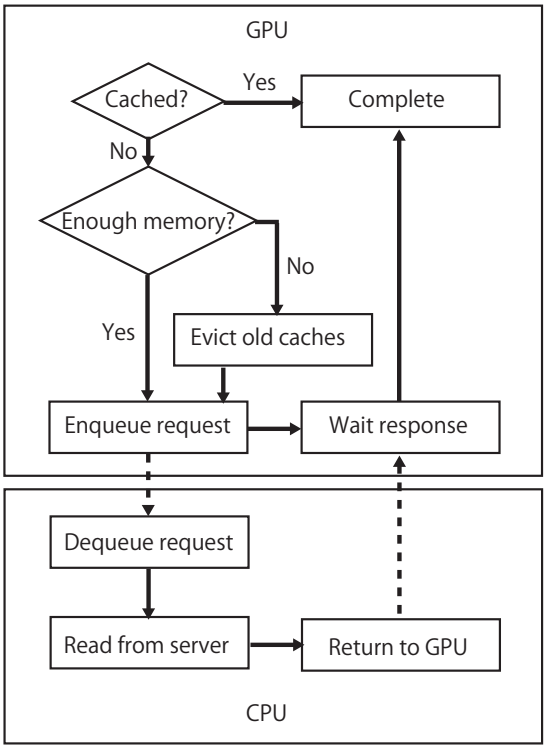

図 5　関数 gread() の処理の概略図

RPC 機構はキューにより構成されている．キューは 1 つのプロセスにつき 1 つずつ，ホストメモリ上に存在する．GPU が API に対してどのような命令を行ったかという情報をキューに追加する．CPU はキューに命令が追加されたことを確認すると，その命令に応じた処理を行い，結果を GPU に返す．GPU は CPU から結果が返ってくるまで処理をブロックする．

図 5 は関数 gread() の処理の概略である．まず，キャッシュがデバイスメモリ上に存在するかを確認する．もしキャッシュされていれば，キャッシュから読み込む．キャッシュされていなかった場合は，キャッシュ領域の空き容量を確認する．キャッシュ領域の空き容量が，読み込みを行うサイズよりも小さかった場合はキャッシュの掃き出しを行う．キャッシュ領域の空き容量が十分ならば，キューに読み込みのリクエストを追加し，CPU からの応答を待つ．CPU はキューからリクエストを取り出し，サーバにリクエストを送信する．サーバはデバイスメモリにファイルの内容を転送する．データの転送が終わると，それをクライアントの CPU に通知する．クライアントの CPU は通知を受け取ると，GPU に処理を返す．GPU は転送されたデータを読み込み，処理を完了する．

3.4　GPUDirect RDMA の利用

我々が開発している API では，ファイルの内容を転送するのに，2.3 節にて説明した GPUDirect RDMA を利用する．しかし，GPUDirect RDMA を用いてデバイスメモリ上のデータを他ノードに送信するとバンド幅が非常に小さい場合があることが，いくつかの既存研究により示されている [12, 15]．そこで我々はファイルシステム上のファイルの内容を読み込む時にのみ GPUDirect RDMA を利用するようにした．キャッシュの書き戻しを行うとき，CPU は GPUDirect RDMA を利用せず，一度ホストメモリ上にキャッシュの内容をコピーしてからサーバにアップロードする．

4　実験

我々の API で発生するオーバーヘッドと，GPUDirect RDMA の効果を検証するために実験を行った．

4.1　実験環境

実験は，クライアント 1 台と分散ファイルシステムサーバ 1 台から構成された InfiniBand FDR 4X によるネットワーク上で行われた．クライアントの CPU は Intel Core i7-4770K，メモリサイズは 32 GB，GPU は NVIDIA Quadro M6000，OS は CentOS 7，InfiniBand の NIC は Mellanox ConnectX-3 である．サーバの CPU は Intel Xeon E5-2665，メモリサイズは 32 GB，OS は CentOS 7，InfiniBand の NIC は Mellanox ConnectX-3 であり，HDD は SAS 6 Gbps で 15000 rpm のものを使用した．

本稿において No API は，我々の開発している API を利用せず，RDMA 版 Gfarm のクライアント API を用いて実装した場合を表す．Without GDR は我々の開発している API で GPUDirect RDMA を利用しなかった場合，Proposal が GPUDirect RDMA を利用した場合をそれぞれ表す．

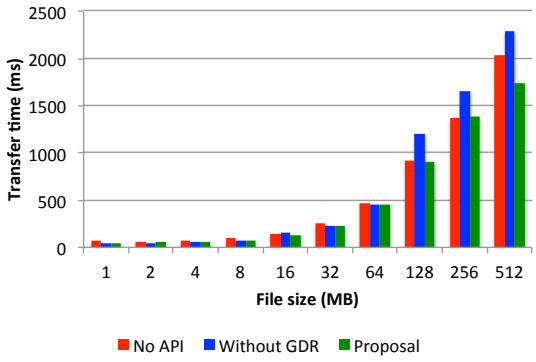

図 6　シーケンシャルリードにおけるデータ転送時間

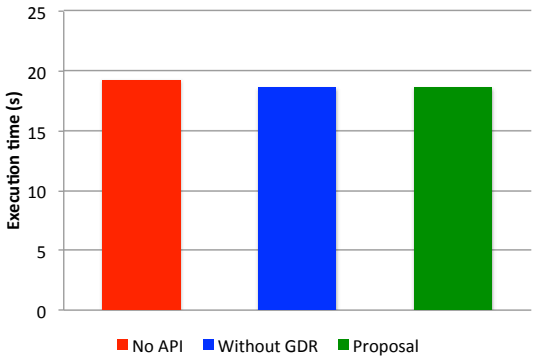

図 7　文字列マッチングプログラムの実行時間

No API と Without GDR におけるデバイスメモリでのキャッシュ領域の大きさを 2 GB とした．しかし，2.3 節にて説明したように，GPUDirect RDMA で使用できるデバイスメモリ領域の大きさには制限があるため，Proposal についてはキャッシュ領域の大きさを 64 MB とした．また実験はサーバのページキャッシュをクリアした上で行っている．

4.2　シーケンシャルリード

Gfarm 上のファイルに対してシーケンシャルリードを行い，サーバ上のディスクからデバイスメモリに，Gfarm 上のファイルの内容が転送されるまでの総時間を図 6 に示す．GPUDirect RDMA を利用しなかった場合，我々の開発している API を利用しなかった場合に比べて転送時間は最悪 1.3 倍となる．しかしながら，GPUDirect RDMA を利用した場合の転送時間は，我々の開発している API を利用しなかった場合と比較して，最悪でも 1.01 倍，最高では 0.69 倍となる．

4.3　文字列マッチング

我々の開発している API を利用した場合と，利用しなかった場合で文字列マッチングを行うプログラムをそれぞれ作成し，実行時間を測定した．これは検索する単語が記載されている辞書ファイルと，検索対象とするファイル名が列挙されている検索対象リストファイルをまず読み込む．そして検索対象リストファイルをもとに，検索対象ファイルを読み込む．辞書ファイルに記載されている全ての単語について，検索対象ファイルに含まれている数を出力するものである．1 単語につき 1 スレッド割り当てられ，各スレッドは割り当てられている単語が検索対象ファイル上に存在する数をカウントする．

本実験で用いた辞書 †1 は 58000 単語，ファイルサイズにして 1.8 MB 掲載されており，Shakespere の作品集 †2 5.2 MB 分のテキストファイルを検索対象とした．検索対象リストファイルのサイズは 16 B である．

結果を図 7 に示す．出力ファイルの大きさは 5.5 MB となった．我々の開発している API では，GPUDirect RDMA の利用を問わず，API を利用する前に比べて 0.97 倍の実行速度を得ることができた．

5　議論

我々の開発している API を利用せず，libgfarm などの CPU コード向けの API のみを利用する場合，GPUDirect RDMA を利用することは不可能である．その結果，クライアントの内部で最低でも 1 回のメモリコピーが発生する．4.2 節では，我々の開発している API により，Gfarm 上のファイルの内容を転送する時間が短縮されることがあるということを示している．また，アプリケーション全体を通しても，我々

†1　http://www.mieliestronk.com/wordlist.html
†2　http://www.gutenberg.org/ebooks/100

の開発している API がアプリケーションの実行時間に対して影響が小さい場合があることを 4.3 節では示している．

近年のクラスタマシンは，GPU ではなく，Xeon Phi [6] を搭載する場合がある．Xeon Phi も，GPU と同様にデバイスメモリを内蔵しており，GPUDirect RDMA と同等のデータ転送機構を持っている．したがって，本稿で論じた手法は，Xeon Phi に移植することが可能である．

提案システムでは，GPUDirect RDMA に対応した環境を想定している．しかしながら，GPUDirect RDMA に対応していない環境でも，クライアント内のメモリコピーを削減する手法が提案されている [1]．この手法は Gdev [7] などの NVIDIA 非公式の GPU デバイスドライバを利用する必要があるため本研究では取り入れていない．この手法を用いることで，GPUDirect RDMA に対応していない環境でも，GPUDirect RDMA を導入したときと同様の効果が得られると考えられる．

6 関連研究

本研究と特に密接に関連した研究として GPUfs [22,23] が挙げられる．GPUfs はローカルファイルシステムを対象とし，GPU コードからファイルの読み書きを行えるようにしたものである．本研究の対象は分散ファイルシステムである．また，GPUfs の研究では，本稿で行ったクライアント内でのメモリコピーについての議論を行っていない．

GPUDirect RDMA を導入したシステムソフトウェアがいくつか発表されている．Oden ら [13] は，InfiniBand の Verbs API を GPU コードから呼び出すためのライブラリを開発した．Potluri ら [15]，Venkatesh ら [25]，Shi ら [19] は，GPU が搭載されたマシンのクラスタを仮定し，GPUDirect を用いることで，MPI で行われるノード間通信の高速化を試みている．GPUnet [8] は，UNIX におけるソケット通信を GPU で実現させるためのライブラリである．GPUnet では，GPUDirect RDMA が有効な環境では GPUDirect RDMA を利用して通信を行うようにしている．本研究は，GPUDirect RDMA を分散ファイルシステムのクライアントで利用した初の試みである．

7 まとめと今後の課題

我々は分散ファイルシステムの Gfarm を対象とし，GPU コードでファイルの読み書きを行う API を開発している．現在まで，GPU コードから CPU コードを呼び出す RPC 機構およびデバイスメモリ上でのキャッシュ機構の実装を行った．また，GPUDirect RDMA を利用することで，ファイル読み込み時にサーバと GPU 間で発生する通信時間を削減することに成功し，GPUDirect RDMA が分散ファイルシステムのクライアントでは有効に機能することを示した．

現在の API ではファイルの作成や削除といった，ファイルシステムの操作で必須となる一部の処理が未実装である．これらの機能の実装をまず完了させる必要がある．またキャッシュに関しても改善の余地は多い．FIFO 戦略ではなく，広く利用されている Least-Recently-Used 戦略の導入を検討することや，デバイスメモリだけでなく，ホストメモリも用いたキャッシュ機構の導入が考えられる．これらの実装を行った上で，より詳細な評価実験を行うことを予定している．

謝辞 本研究を行うにあたって，筑波大学の建部修見教授及び HPCS 研究室ビッグデータグループの方々より有益な助言を頂いた．また本研究は科学技術振興機構 CREST の研究課題「ポストペタスケールデータインテンシブサイエンスのためのシステムソフトウェア」の支援を受けている．

参 考 文 献

[1] Anh, N. V., Fujii, Y., Iida, Y., Azumi, T., Nishio, N., and Kato, S.: Reducing Data Copies between GPUs and NICs, *the Proceedings of the IEEE International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA '14)*, 2014.

[2] Carns, P. H., LigonIII, W. B., Ross, R. B., and Thakur, R.: PVFS: A Parallel File System for Linux Clusters, *the Proceedings of the 4th Annual Linux Showcase and Conference (ALS '00)*, 2000.

[3] Chilimbi, T., Suzue, Y., Apacible, J., and Kalya-

naraman, K.: Project Adam: Building an Efficient and Scalable Deep Learning Training System, *the Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*, 2014.

[4] Coates, A., Huval, B., Wang, T., Wu, D., Catanzaro, B., and Andrew, N.: Deep learning with COTS HPC systems, *the Proceedings of the 30th International Conference on Machine Learning (ICML '13)*, 2013.

[5] Endo, T., Nukuda, A., and Matsuoka, S.: TSUBAME-KFC: A Modern Liquid Submersion Cooling Prototype towards Exascale Becoming the Greenest Supercomputer in the World, *the Proceedings of the 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS '14)*, 2014.

[6] Intel: Xeon Phi™ Product Family, `http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html`.

[7] Kato, S., McThrow, M., Maltzahn, C., and Brandt, S.: Gdev: First-Class GPU Resource Management in the Operating System, *the Proceedings of the USENIX Annual Technical Conference (USENIX ATC '12)*, 2012.

[8] Kim, S., Huh, S., Hu, Y., Zhang, X., Witchel, E., Wated, A., and Silberstein, M.: GPUnet: Networking Abstractions for GPU Programs, *the Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*, 2014.

[9] NVIDIA Corporation: CUDA Zone, `http://developer.nvidia.com/cuda-zone`.

[10] NVIDIA Corporation: Developing a Linux Kernel Module using GPUDirect RDMA, `http://docs.nvidia.com/cuda/gpudirect-rdma/`.

[11] NVIDIA Corporation: NVIDIA GPUDirect, `http://developer.nvidia.com/gpudirect`.

[12] NVIDIA Corporation: NVIDIA/gdrcopy - GitHub, `https://github.com/NVIDIA/gdrcopy`.

[13] Oden, L., Froning, H., and Pfreundt, F.-J.: Infiniband-Verbs on GPU: A Case Study of Controlling an Infiniband Network Device from the GPU, *the Proceedings of the 4th International Workshop on Accelerators and Hybrid Exascale Systems (AsHES '14)*, 2014.

[14] OpenCL - Khronos Group: Khronos Group, `https://www.khronos.org/opencl/`.

[15] Potluri, S., Hamidouche, K., Venkatesh, A., Bureddy, D., and Panda, D. K.: Efficient Inter-node MPI Communication using GPUDirect RDMA for InfiniBand Clusters with NVIDIA GPUs, *the Proceedings of the 42nd International Conference on Parallel Processing (ICPP '13)*, 2013.

[16] RedHat, Inc.: Write once, read everywhere - Gluster, `http://www.gluster.org/`.

[17] Rodeh, O., Bacik, J., and Mason, C.: BTRFS: The Linux B-Tree Filesystem, *ACM Transactions on Storage (TOS)*, Vol. 9, No. 9, 2013.

[18] Sasaki, S., Takahashi, K., Oyama, Y., and Tatebe, O.: RDMA-based Direct Transfer of File Data to Remote Page Cache, *the Proceedings of the 2015 IEEE International Conference on Cluster Computing (CLUSTER '15)*, 2015.

[19] Shi, R., Potluri, S., Hamidouche, K., Perkins, J., Li, M., Rossetti, D., , and Panda, D. K.: Designing Efficient Small Message Transfer Mechanism for Inter-node MPI Communication on InfiniBand GPU Clusters, *the Proceedings of the 21st International Conference on High Performance Computing (HiPC '14)*, 2014.

[20] Shimokawabe, T., Aoki, T., Muroi, C., Ishida, J., Kawano, K., Endo, T., Nukuda, A., Maruyama, N., and Matsuoka, S.: An 80-Fold Speedup, 15.0 TFlops Full GPU Acceleration of Non-Hydrostatic Weather Model ASUCA Production Code, *the Processing of the 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10)*, 2010.

[21] Shvachko, K., Kuang, H., Radia, S., and Chansler, R.: The Hadoop Distributed File System, *the Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST '10)*, 2010.

[22] Silberstein, M., Ford, B., Keidar, I., and Witchel, E.: GPUfs: Integrating a File System with GPUs, *the Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13)*, 2013.

[23] Silberstein, M., Ford, B., Keidar, I., and Witchel, E.: GPUfs: Integrating a File System with GPUs, *ACM Transactions on Computer Systems (TOCS)*, Vol. 32, No. 1, 2014.

[24] Tatebe, O., Hiraga, K., and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol. 28, No. 3, Ohmsha, Ltd. and Springer, 2010.

[25] Venkatesh, A., Subramoni, H., Hamidouche, K., and Panda, D. K.: A High Performance Broadcast Design with Hardware Multicast and GPUDirect RDMA for Streaming Applications on Infiniband Clusters, *the Proceedings of the 21st International Conference on High Performance Computing (HiPC '14)*, 2014.