Vol.2018-HPC-163 No.19
2018/3/1

アクセラレータクラウドを実現するシステムソフトウェアFlowOSの提案

高野 了成[1,a)]　池上 努[1]　須崎 有康[1]　田中 哲[1]　広渕 崇宏[1]

概要：データセンター向け計算機システムとして、ラックスケールアーキテクチャ等、論理的な計算システム構成と物理的な資源を分離したシステムが提案されている。本稿では、ラックスケールアーキテクチャを拡張し、エネルギー効率の高い人工知能処理基盤を実現するため、用途ごとに様々な種類のアクセラレータやストレージクラスメモリを動的に組み合わせるアクセラレータクラウドシステム Flow in Cloud (FiC) と、FiC の制御・資源管理を行う分散ミドルウェア FlowOS を提案する。FiC は GPU や FPGA といった AI エンジンをサーキットスイッチネットワークを介して直結し、アプリケーションの各処理ステージごとに最適な資源を割り当てることが可能である。FlowOS は FiC の資源管理を担うとともに、デバイス間通信を配列データの転送として抽象化し、処理のフローに集中したプログラミングを可能にする。

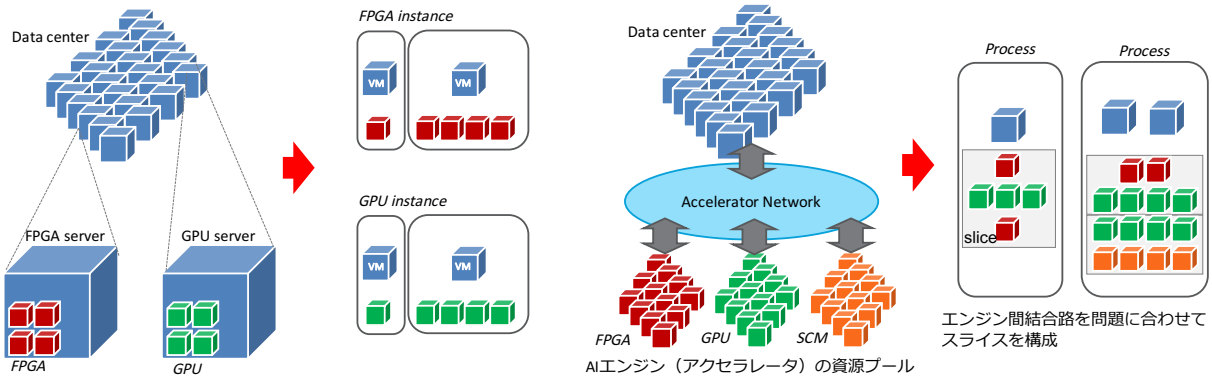
1. はじめに

インターネットの普及や各種センサーの発展によるデータ収集の高性能化により、データセンターに蓄積されるデータの量・速度・種類は増加の一途を辿っている。このようなビッグデータの利活用が社会課題解決やビジネス創出の成否の鍵を握るという認識が広まり、より有益な情報を高速かつ高効率に抽出するデータ処理基盤への期待が高まっている。このような背景の中、クラウド及び高性能計算（HPC）のそれぞれの分野で培われてきたハードウェアとソフトウェア技術が融合し、HPE The Machine [1]、Intel Rack Scale Architecture [2]、UCB FireBox [3]、NEC RDStore [4] 等、データセンタースケールのデータ処理基盤システムが提案されている [5]。さらにデータセンターを取り巻く状況は、ポストムーア及びポストノイマンの２つの潮流により直近 5〜10 年のスパンで変革が加速され、激変すると考える。まず CMOS プロセスの微細化が電力限界と経済限界により行き止まりを迎え、所謂ムーアの法則が 2025 年頃に終焉を迎える（ポストムーア） [6]。これに対して特定用途の性能向上に特化したデバイス・回路の開発や、３次元積層によるチップの全体性能向上が図られている。一方で、深層学習に代表される人工知能技術がテクノロジードライバとなり、NVIDIA GPU における Tensor Core、富士通 Deep Learning Unit (DLU)、Google Tensor Processing Unit (TPU)、IBM TrueNorth、D-Wave の量子アニーリングマシン等のポストノイマン型アクセラレータの開発が加速している。さらにデータセンターの大きなアプリケーションの一つになりつつある分散深層学習の分野では、大規模 HPC システムを活用して学習を高性能化するために、データ並列学習の並列度を高める取り組みが急速に進んでいる [7], [8], [9]。

これらのヘテロジニアスコンピューティング、深層学習の大規模並列化といった技術動向は、今後のコンピュータアーキテクチャのみならず、データセンターアーキテクチャの設計にも大きな影響をもたらす。つまり、ホモジニアスな構成のサーバを疎に結合して資源の全体最適を図るというシステムから、ヘテロジニアスな機能を持つサーバもしくはコンポーネントをアプリケーションに合わせて適材適所で密に結合するシステムへと変革が起きると考える。その結果、データセンターネットワークに求められる広帯域と低遅延への要求はますます高くなる。これに対して、広域網で利用されている波長多重や多値変調をデータセンタ内ネットワークに適用することで、ファイバあたり数 10 Tbps 級の帯域が 2030 年頃には実現可能であるという試算が示されている [10][11]。

我々は、このような用途特化型演算ハードウェアと超広帯域通信を組み合わせた新しいコンピューティングパラダイムである「フローセントリックコンピューティング」を提案し [12]、その実証に向けて異種 AI エンジン統合クラウド（**Flow in Cloud**）の開発に着手した [13][14]。さらにシステムソフトウェアの観点からは、システム全体の資

[1] 国立研究開発法人 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST)
[a)] takano-ryousei@aist.go.jp



(a) 既存データセンタ　　(b) 提案システム

図 1　クラウドとヘテロジニアスコンピューティング：既存データセンタ (a) と提案システムを採用したアクセラレータクラウド (b) との比較

源利用効率や性能を最適化するシステム化技術の創出、及びそれを具現化したデータセンタワイドなミドルウェアの実現が課題となる。我々は、異種の AI エンジンが連なるフローに着目して資源管理を行う点から、この分散ミドルウェアを **FlowOS** と呼び、研究開発を進めている。

本論文の構成は次のとおりである。2 節では、アクセラレータクラウドの一例として我々が開発を進めている Flow in Cloud (FiC) の概要について述べる。3 節では、アクセラレータクラウドを実現するための分散ミドルウェア FlowOS を紹介する。4 節では、FiC の模擬環境とその予備評価について述べる。最後に 5 節でまとめと今後の予定について言及する。

2. アクセラレータクラウド

我々が提案するアクセラレータクラウドの概要を**図 1** に示す。既存クラウドデータセンタにおいて GPU や FPGA 等のアクセラレータをサービスとして提供する場合、物理的にはアクセラレータを組み込んだ PC サーバ群を準備し、利用者の要求に応じてアクセラレータを PCI パススルーで接続した仮想マシンインスタンスを提供する。アクセラレータクラウドでは、PC サーバからアクセラレータを分離してプール化し、専用のアクセラレータネットワークを介して PC サーバと接続する。

既存クラウドデータセンタアーキテクチャ **図 1a** では、アクセラレータを増やすには PC サーバ単位で増やす必要があり、有効利用できないハードウェア資源や無駄な電力を消費する。一方、アクセラレータクラウド **図 1b** では、必要なときに、必要なだけ、必要な種類のアクセラレータをアプリケーションの問題に合わせて動的に組み合わせた計算環境を構成し、利用者に提供する。さらにこのような計算環境の構築をソフトウェアで集中制御することで、システム全体の利用率や可用性の向上が期待できる。以下、本提案で利用するアクセラレータのことを AI エンジンと呼ぶ。

さらに既存のシステムでは、GPU や FPGA は PCI バス経由でホスト PC と接続されるアクセラレータの形態を取っているため、デバイス間で直接データ交換を行うことは基本的に考慮されておらず、ホスト PC のメインメモリを介した通信になる。NVIDIA は GPU 間のインタコネクト技術として NVLink を提案しているが、あくまでノード内での規模の Point-to-Point 接続技術でありスケーラビリティは考慮されていない。我々は、PCI バスやホスト PC を経由するオーバーヘッドを除去し、多数の異種 AI エンジンの柔軟な組み合わせを実現するために、AI エンジンを搭載した基板 (AI エンジンノード) と、それらをつなぐネットワークから構成される Flow in Cloud (FiC) と呼ぶアクセラレータクラウドを開発している。FiC は、計算機仮想化技術を用いることなく、アプリケーションに応じた柔軟なアクセラレータの組合せをオンデマンドで構成する、ベアメタルクラウドの一形態と捉えることができる。

個々の AI エンジンノードは、GPU や FPGA などの AI エンジンに加え、DRAM、ネットワーク I/O、制御用 CPU から構成される。制御用 CPU は演算を実行するものではなく、あくまで GPU や FPGA の設定を行うための補助的なものを想定している。AI エンジン間のネットワーク (FiC ネットワーク) は複数のスイッチノードから構成され、将来の光ネットワークの導入も見越して、パケットスイッチではなく、サーキットスイッチネットワークを前提とする。光ネットワークの場合は、光の速度のままでスイッチングを行える利点があるが、電気ネットワークの場合でも実装が簡単になるという利点もある。一方、通信の開始前に AI エンジンノードの端点同士で接続を確立する必要があることを意味し、これを効率的に利用するには、後述する FlowOS のようなシステムソフトウェアとの連携が必要になる。

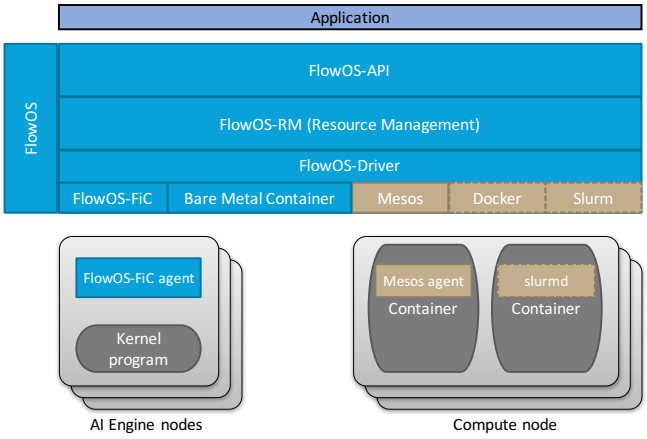


図 2 FlowOS の構成

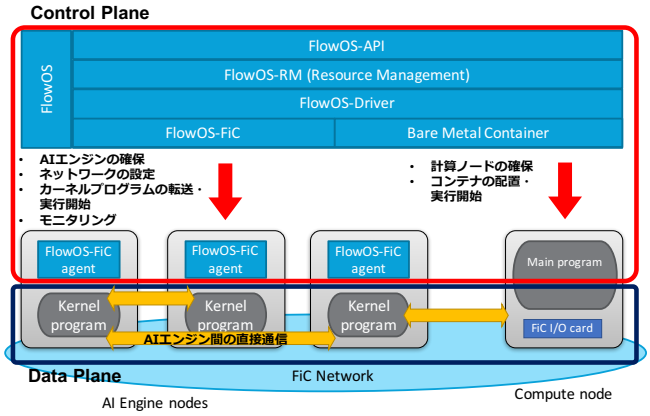


図 3 FlowOS の設計：制御プレーンとデータプレーンの分離

3. FlowOS

3.1 FlowOS の概要

FlowOS は、アクセラレータクラウド（FiC）の資源である AI エンジンノードとスイッチノードを統一的に管理し、利用者に対して、論理的なアプリケーション実行環境であるスライスを提供する分散ミドルウェアである。

まず最初に FlowOS における用語を定義する。プロセスとスライスの関係は図 1b も参照されたい。

- **タスク**: 一つの AI エンジンで動作する処理の単位。名前、入出力パラメータ、処理の実体から構成される。処理の実体は C++、Python、CUDA、OpenCL、HDL 等、目的と手段に応じて使い分けることができる。
- **フロー**: 処理の流れに応じてタスクを繋げたもの。前段のタスクの出力を後段のタスクの入力に繋げていくことで、フローを構築する。
- **スライス**: フローを実行するための計算資源（AI エンジンの集合）であり、FlowOS はスライス単位で資源を動的に確保・解放する。なお、一度作成されたスライスに含まれるアクセラレータ数・種類は不変である。
- **プロセス**: 一つのアプリケーションの実行単位であり、アプリケーションは一つ以上のフローから構成され、一つ以上のスライスで実行される。

FlowOS の主要な機能は以下のとおりである。

- AI エンジンの資源管理
- プロセスのライフサイクル管理
- プログラミング環境の提供

図 2 に FlowOS の構成を示す。FlowOS は主に FlowOS-API、FlowOS-RM (Resource Management)、FlowOS-Driver の 3 層から構成される。

既存のクラスタ管理ミドルウェアは、各計算ノード上に Linux 等フル機能の OS が動作することを前提としている。一方、FiC では、アプリケーションの実行効率を追求し、AI エンジンがデータ処理に集中するために、可能な限り外部からの干渉を抑制する必要がある。そこで**図 3** に示すように、OS の構造を制御プレーンとデータプレーンに分離し、性能と機能性の両立を図るという着想に至った。データプレーンは、アプリケーション実行に特化した環境であり、ホスト CPU を介さず AI エンジン間で直接データの交換が可能である。一方、制御プレーンは、資源管理、ユーザ管理に必要な機能性を提供し、FlowOS と各 AI エンジンノードに搭載された制御用 CPU 上で動作する FlowOS agent が連携することで動作する。

3.2 FlowOS-API

FiC では必然的にヘテロジニアスな資源を組み合わせたプログラミングが必要になるが、デバイス間の通信は規格が乱立しており、統一化が困難であることに加え、通信効率が悪い。また、プログラマが手動で AI エンジンを管理する必要がある。そこで、FlowOS-API ではデバイス間通信を配列データの転送として抽象化し、処理のフローに集中したプログラミングを可能にする。さらに、後述する FlowOS-RM と連携して、処理内容やモニタリングに応じて、処理に最適なスライスを構築する。

FlowOS-API は Python ライブラリとして実装しており、以下に、FlowOS-API を用いた単純なアプリケーション例を示す。4~10 行目で 2 つのタスク `add_scalar_def` と `sub_scalar_def` を定義する。13~17 行目で前述のタスクを利用してフローを定義する。19 行目でフローに基づいてスライスを確保し、21 行目でフローを実行する。ここに示した API はプリミティブとなる低水準 API であるが、実際のアプリケーションは、例えば機械学習向けの高水準 API を利用して記述することになる。

```
1 import flowos as fo
2
3 # タスクの定義
4 add_scalar_def = {'task_name': "add_scalar",
5   'iport_nametypes': (fo.int64, fo.int64),
```

```
  'oport_nametypes': fo.int64,
  'fun': add_scalar_fun}
add_scalar = fo.task(add_scalar_def)
    :
sub_scalar = fo.task(sub_scalar_def)

# フローの定義
a = fo.flow_input('a', fo.int64)
b = fo.flow_input('b', fo.int64)
c = fo.flow_input('c', fo.int64)
d = sub_scalar(add_scalar(a, b), c)
o = fo.flow_output(d)

s = fo.slice(o) # スライスの確保

vs = s.run(1, 2, 3) # フローの実行
print(vs) # [0]

s.free() # スライスの解放
```

3.3 FlowOS-RM

FlowOS において、システム全体の資源管理を行い、アプリケーションからの要求に対してスライスを割り当てるのが、FlowOS-RM である。具体的には、前述のプログラム例ではタスクをどの AI エンジンノードで実行するか明示的に指定していないが、そのスケジューリングを行うことが FlowOS-RM の役割である。FlowOS-RM はタスクを AI エンジンノードにデプロイし、AI エンジンノード間の経路を疎通ためにスイッチノードを設定する。さらにプロセスのモニタリング機能を提供し、AI エンジンの利用率を最大化できるように、自動及び手動によるスケジューリング最適化の手段を提供する。なお、FiC システムにおけるスケジューリングに際しては、下記の点に配慮する必要がある。

- FPGA の再構成可能なデバイスではあるが、論理合成、ビットストリーム転送から再構成までを含めると CPU におけるプログラムロードと同程度のレイテンシは期待できない。したがって、各 AI エンジンにどのようなプログラムがデプロイされているかも含めて、スケジューリングする必要がある。
- ネットワークトポロジを考慮して、レイテンシの少ない経路を選択する。
- スイッチノードに搭載された FPGA を用いたインネットワークプロセッシング（集約などの簡単な処理を想定）を考慮する。

アクセラレータ以外の PC サーバに関しても FlowOS 単独で資源管理を実現するが、既存のデータセンタをアクセラレータクラウド化する場合や、既存の Hadoop や Spark、MPI といった各種フレームワークと共存したい場合は、クラスタ資源管理システムの Apache Mesos [15] やバッチシステムである Slurm などと連携可能できることが望ましい。FlowOS-RM を Apache Mesos のフレームワークとして実行する手段をオプション機能として提供している。

3.4 FlowOS-Driver

FlowOS-Driver は、スライスを構成する資源の確保・解放や、タスクのデプロイ等、AI エンジンに依存する操作・アクセスの抽象化層である。ここでは FlowOS-Driver の具体例として、Bare-Metal Container (BMC) と FlowOS-FiC について述べる。

Bare-Metal Container (BMC) [16] は、コンテナとネットワークブート、リモート電源制御を組み合わせることで、アプリケーションに最適化したユーザランドと OS カーネルを任意に組み合わせてデプロイする仕組みである。一方、Docker 等のコンテナは、アプリケーション実行環境の可搬性の点では有効な技術であるが、仮想マシンとは異なり、利用者が OS カーネルを選択することはできない。FlowOS では、スライスに PC サーバ (CPU) を含む場合、BMC ドライバを利用して実行環境をデプロイする。

次に FlowOS-FiC ドライバの概要について述べる。工藤らの研究チームでは、FiC ネットワークを構築するための FiC スイッチボードを開発している [13]。FiC スイッチボードは、FPGA (Xilinx Kintex Ultrascale XCKU095)、DDR4 メモリスロット、8 対の 10Gbps シリアルリンク、及び制御用の Rasberry Pi 3（以下、「RasPi」と記す）ボードから構成される。FiC スイッチボード上にはシリアルリンクのサーキットスイッチロジック以外に、ユーザロジックを書き込める領域があり（パーシャルリコンフィグレーション可能）、AI エンジンとしても機能する。また、RasPi の GPIO と FPGA が連結されており、コンフィグレーションデータの書き込みや、小規模なデータの授受が可能である。

FlowOS-FiC ドライバは、RasPi 上の Linux で動作する FlowOS-FiC agent と連携し、次の機能を提供する。

- コンフィグレーションデータのカタログ管理
- コンフィグレーションデータのデプロイ
- コンフィグレーションデータの書き込み
- スイッチングテーブルの読み書き
- その他のデータ読み書き（GPIO ピンの読み書き）

FlowOS-FiC ドライバは FlowOS アプリケーションからの要求に応じて、コンフィグレーションデータのデプロイや書き込みを行う他に、単独のコマンドとして動作可能な設計としている。

4. FlowOS の試作と予備評価

4.1 FiC 模擬環境の構築

FiC のハードウェアと並行して FlowOS の開発を進めるために模擬環境を構築した。模擬環境では、AI エンジンノードとして PCI デバイスを用い、アプリケーションの要求に応じて、PC サーバと PCI デバイスの接続を動的に変更する方針とした。これでは AI エンジン間を直接接続するという FiC 本来の目的は達成できないが、FlowOS の機能面を検証するには十分である。例えば、GPU を利用するアプリケーションに対しては GPU クラスタをスライスとして提供し、GPU は不要だが高速なストレージが必要なアプリケーションに対しては NVMe を搭載したクラスタをスライスとして提供することができる。

PC サーバと PCI デバイスをディスアグリゲーションする技術として、ExpEther [17][18] や rCUDA [19] が存在する。ExpEther は PCI Express バスを一般的な Ethernet 上に拡張できる PCI Express over Ethernet 技術であり、ホスト PC は ExpEther ホストバスアダプタを利用して、PCI Express 規格準拠デバイスをネットワーク越しに結合することが可能になる。一方、rCUDA は、対応デバイスは NVIDIA 製 GPU に限られるが、CUDA ライブラリを仮想化することで、リモート PC の GPU デバイスをネットワーク越しにあたかもローカルに存在するかのように利用できる技術である。以下では、ExpEther と rCUDA について、性能面と機能面について検証する。

実験環境では 40 ギガビット Ethernet スイッチを介して、4 台の PC サーバと 4 台の ExpEther IO ユニットが接続されている。ExpEther IO ユニットには、合計でそれぞれ 4 基の NVIDIA P100 GPU、及び Intel NVMe SSD が収納されている。PC サーバの諸元を**表 1** に示す。なお、PCI デバイスのホットプラグを安定化するために、カーネル起動時オプションとして、`pci=pcie_bus_safe` を付与している。本オプションの副作用により PCIe パケットのペイロードが小さくなる。

4.2 ExpEther と rCUDA の比較

ExpEther は IO ユニットに格納された PCI デバイスを Ethernet 経由で透過的に PC サーバに接続するハードウェア技術である。OS からは専用のソフトウェアをインストール等することなく、通常の PCI デバイスとして認識される。したがって、本プロジェクトで想定している深層学習フレームワーク等は問題なく動作する。一方、rCUDA はソフトウェア技術であり、専用のハードウェアは不要であるが、CUDA との互換性が十分ではないので、模擬環境目的で使用するには制限がある。例えば、論文執筆現時点で cuDNN に完全対応していない他、Chainer や DeepBench

表 1 実験環境の諸元

Compute Node	
CPU	10-core Intel Xeon E5-2630v4/2.2GHz
M/B	Supermicro X10SRG-F
Memory	128 GB DDR4-2133
Network	ExpEther 40G HBA
InfiniBand	Mellanox ConnectX-2 (QDR)
Disk	Intel SSD 480 GB
Resource Pool	
GPU	NVIDIA P100 x4
NVMe	Intel SSD 750 x4
Software	
OS	CentOS 7
Kernel	Linux 3.10.0-514.26.2.el7.x86_64
	NVIDIA CUDA 8.0
	rCUDA v16.11.04.02

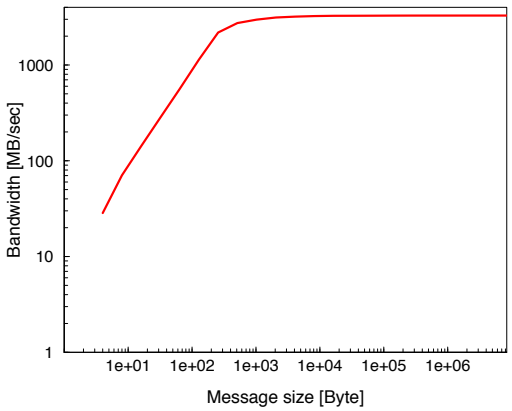


図 4 ib_write_bw

等を正常実行することはできなかった。

性能面での比較を行うため、CUDA 付属の bandwidthTest プログラム、nbody プログラム、及び DeepBench を実行し、比較した。

rCUDA では、CUDA 関数を呼び出すクライアントと、CUDA 関数を GPU 上で実行するサーバの間を TCP/IP によるソケット通信もしくは InfiniBand の RDMA 通信を用いて接続する。予備実験として、本環境における InfiniBand QDR での通信性能を測定した。結果を**図 4** に示す。メッセージサイズが 1 KB より大きい場合は約 3.3 GB/sec の性能が安定して出ている。

bandwidthTest により、CPU と GPU メモリ間の転送帯域を 4 パターン（local、ExpEther、rCUDA(RDMA)、rCUDA(TCP)）で測定した。local では、PC サーバの PCIe Gen3 スロット（x16 レーン）に直接 GPU が挿入されている。rCUDA に関しては、クライアント・サーバ間の通信路として前述の InfiniBand QDR を使用し、プロトコルとしては RDMA と TCP/IP の 2 パターンを計測した。なお、40G ExpEther HBA 及び InfiniBand HBA は x8 レーンの PCIe Gen3 スロットに挿入されている。今回はホストメモ

リとしてページロックメモリを割り当てたので、OS でのページングは発生しない。実験結果を**図 5** に示す。ピーク転送帯域（host to device）は、local で 11.7 GB/sec であるのに対して、ExpEther では 2.2 GB/sec、rCUDA(RDMA) で 3.1 GB/sec、rCUDA(TCP) で 1.5 GB/sec と、40G Ethernet もしくは InfiniBand QDR の帯域の制限により、悪化する。また、1 KB 以下の小さいメッセージ転送は ExpEther が速いが、100 KB 以上の大きなメッセージ転送では rCUDA(RDMA) が速い。これは PCIe パケットのペイロードは小さく、大きなメッセージ転送のオーバヘッドが大きいことが理由だと考える。

表 2 に DeepBench [20] に含まれる行列積 (GEMM) と畳み込み演算 (Convolution) の学習のベンチマーク結果を示す。これらは深層学習で頻出するカーネル演算である。GEMM のカーネルは M=5124、N=9124、K=2560、畳み込みの入力サイズは W=224、H=224、C=3、N=16 とした。計算インテンシブであり、ホスト・デバイス間の通信の影響は無視できるので、local と ExpEther に明確な性能差はない。なお、rCUDA ではエラーにより実行できなかった。

続いて、1 台の PC サーバから複数の GPU を用いるケースとして、nbody の性能を比較した。パラメータ N は 256000 とし、小数点演算の精度として FP32 と FP64 を用いた。なお、FP32 の方が相対的に通信性能の影響が大きくなる。結果を**図 6** に示す。1 GPU での性能低下は 表 2 同様に無視できる。一方、2 GPU になると GPU 間通信が発生するため、local と ExpEther には図 5 に示したメモリ転送性能差が原因となって明かな性能低下が生じる。特に FP32 の場合は、GPU 数を増やしても逆に性能が低下している。この性能低下の原因については今後さらに追求する予定である。

最後に fio ベンチマークを用いて、NVMe のストレージ性能を測定した。4KB のランダム読書きの IOPS を**図 7** に示す。2 箇所で ExpEther の性能低下が確認できた。書き込みでスレッドが少ない場合、最大 33%の性能低下が見られる。これはスレッドが少ないとリクエストキューが埋まらず十分な性能が出ていないものと考える。また、読み込みでスレッド数が多い場合、最大 16%の性能低下が見られる。後者に関しては、`pci=pcie_bus_safe` を指定しなければ、性能低下は起こらないので、小さいペイロードサイズに起因していると考える。

5. まとめと今後の予定

ポストムーア時代における継続的な計算機システムの発展の鍵は、用途特化型演算ハードウェアと超広帯域通信の組合せである。異種のハードウェアを多数組み合わせたシステムの可能性を引き出すためには、アプリケーションやアルゴリズムに対して、これらの資源を、ネットワークやストレージまで含めて、どのように組み合わせるべきか、システム全体の構成を最適に制御するアーキテクチャが必要であり、このような議論や開発は世界的にも端緒についたところである。

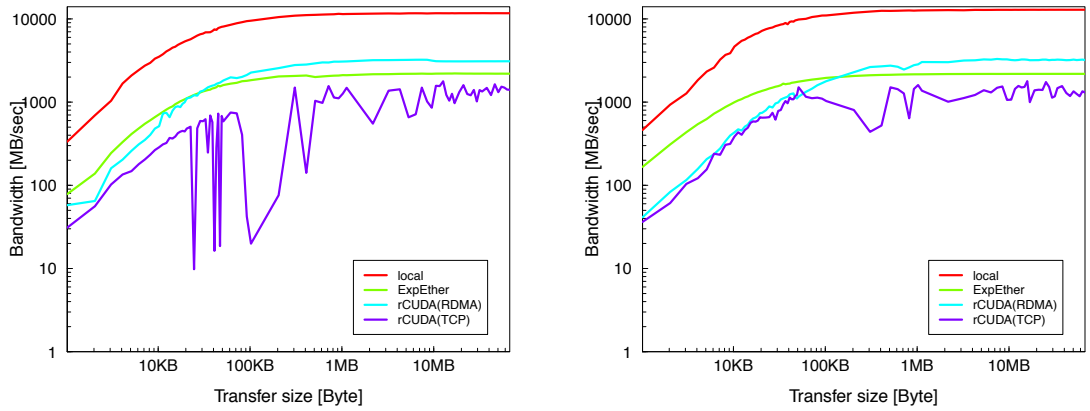
このような背景のなか、本論文では、人工知能アプリケーションの実行ステージごとに異種のアクセラレータ（AI エンジン）を適材適所で組み合わせることができるアクセラレータクラウドの概要と、これを実現するハードウェアアーキテクチャ Flow in Cloud (FiC)、及び分散ミドルウェア FlowOS について述べた。

我々は、FiC のハードウェアの開発と並行して、汎用 PC サーバと ExpEther を用いた模擬環境を構築し、FlowOS の試作システムを開発中である。模擬環境の構築技術として、ExpEther と rCUDA を検討し、性能面で劣る場合もあるが、汎用性と互換性の面から ExpEther を採用した。現時点で、FlowOS-API、FlowOS-RM、及び Bare-Metal Container ドライバの基本機能を実装した。今後は、FlowOS-API を用いた機械学習アプリケーションの実装と並行して、FlowOS-FiC ドライバの実装を進め、アクセラレータクラウドのコンセプトの有効性を実証する予定である。

謝辞 本研究は、NEDO 委託事業「IoT 推進のための横断技術開発／省電力 AI エンジンと異種エンジン統合クラウドによる人工知能プラットフォーム」での研究成果の一部を用いている。また、FiC のハードウェア開発を担当し、定期的に議論して頂いた東京大学 工藤知宏教授、慶應義塾大学 天野英晴教授に大いに感謝する。

参考文献

[1] HPE The Machine: https://www.labs.hpe.com/the-machine (2018).
[2] Intel Rack Scale Architecture: https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html (2018).
[3] Asanovic, K.: Firebox: A hardware building block for 2020 warehouse-scale computers (2014).
[4] 吉川隆士, 菅 真樹, 高橋雅彦, 宮川伸也, 飛鷹洋一, 阿部晋樹：スケールアップにより多種多様なコンピューティングを実現する Resource Disaggregated Platform, NEC 技報 Vol.67, No.2, pp. 69–74 (2015).
[5] Barroso, L. A., Clidaras, J. and Hölzle, U.: *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition* (2013).
[6] International Roadmap for Devices and Systems (IRDS): http://irds.ieee.org/reports (2016).
[7] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. and He, K.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, *arXiv preprint arXiv:1706.02677* (2017).
[8] You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J. and Keutzer, K.: ImageNet Training in Minutes, *arXiv preprint arXiv:1709.05011* (2017).
[9] Akiba, T., Suzuki, S. and Fukuda, K.: Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes, *arXiv preprint arXiv:1711.04325* (2017).



(a) Host to Device (b) Device to Host

図 5 bandwidthTest の比較

表 2 DeepBench の比較 [msec]

	local (FP32)	local (FP16)	ExpEther (FP32)	ExpEther (FP16)
GEMM	29.121	15.424	29.156	15.427
Convolution	3.846	6.842	3.847	6.842

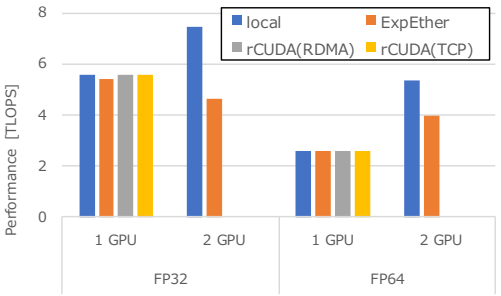


図 6 nbody の比較 [TFLOPS]

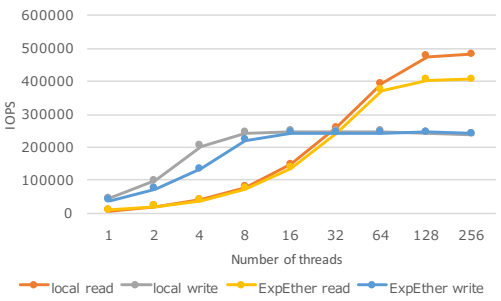


図 7 fio (4KB random read/write IOPS) の比較

[10] Kudoh, T., Ishii, K. and Namiki, S.: Current status and future prospects of optical communications technology and possible impact on future BDEC ssytems, *4th Big Data Extreme Computing Workshop 2016* (2016).

[11] Inoue, T., Kurosu, T., Ishii, K., Kuwatsuka, H. and Namiki, S.: Exabit Optical Network Based on Optical Comb Distribution for High-Performance Datacenters: Challenges and Strategies, *Frontiers in Optics 2015 OSA Technical Digest, FTh3C.3* (2015).

[12] Takano, R. and Kudoh, T.: Flow-centric computing leveraged by photonic circuit switching for the post-moore era, *10th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pp. 1–4 (2016).

[13] 工藤知宏，高野了成，天野英晴，鯉渕道紘，松谷宏紀，塙 敏博，田浦健次朗，並木 周，池上 努，須崎有康，田中 哲，赤沼領大：データの流れに着目した異種エンジン統合クラウドシステム Flow In Cloud，電子情報通信学会技術研究報告 117(153)，pp. 1–5 (2017).

[14] 高野了成，池上 努，須崎有康，田中 哲，広渕崇宏：異種 AI エンジン統合クラウドの実現に向けたシステムソフトウェア Flow OS の構想，情報処理学会研究報告 2016-OS-139 (5)，pp. 1–7 (2017).

[15] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S. and Stoica, I.: Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center., *NSDI*, Vol. 11, pp. 22–22 (2011).

[16] Suzaki, K., Koie, H. and Takano, R.: Bare Metal Container, *18th IEEE International Conference on High Performance Computing and Communications (HPCC)*, pp. 25–36 (2016).

[17] Suzuki, J., Hidaka, Y., Higuchi, J., Yoshikawa, T. and Iwata, A.: ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform, *14th IEEE Symposium on High-Performance Interconnects (HOTI)*, pp. 45–51 (2006).

[18] ExpEther Consortium: http://www.expether.org (2016).

[19] Reaño, C. and Silla, F.: A Performance Comparison of CUDA Remote GPU Virtualization Frameworks, *IEEE International Conference on Cluster Computing* (2015).

[20] Baidu Research: DeepBench, https://github.com/baidu-research/DeepBench (2018).