Vol.2019-OS-145 No.16
2019/3/1

マルチFPGAクラウドシステムにおける資源管理の検討

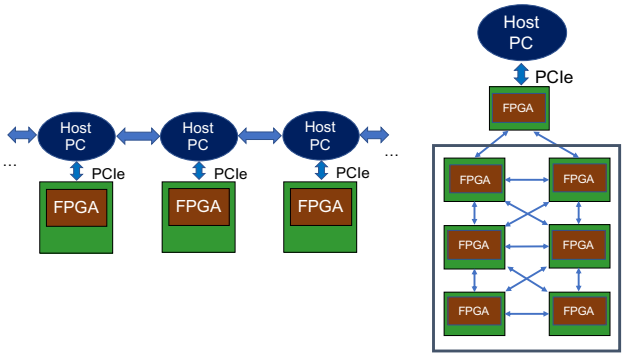
高野 了成[1,a)]　池上 努[1]　鯉渕 道紘[2]　胡 曜[2]　弘中 和衛[3]　天野 英晴[3]　工藤 知宏[4]

概要：ビッグデータ解析や機械学習処理においてFPGAのクラウド利用が進んでいる．我々は，コスト性能比のよいFPGAと高速ネットワークを結合することで，複数のFPGAをまとめて巨大なアクセラレータとして仮想化するクラウドシステムFlow-in-Cloudの研究開発を進めている．本稿では，マルチFPGAを単一システムとして管理し，プログラミングするための課題と解決策について議論する．

1. はじめに

FPGAはプログラム可能な集積回路であり，今日では組み込み機器からクラウドデータセンタに至るまで，幅広い領域で利用されている．さらに2014年頃より，FPGAの性能向上，高位合成技術の進展等の要因から，ビッグデータ解析や機械学習のアクセラレータとしてFPGAを利用する事例が多数報告されるようになり[1], [2]，さらにFaaS (FPGA as a Service) と呼ばれる商用クラウドサービスも始まっている．一方で，アクセラレーションが必要な処理の複雑化・大規模化は，FPGAの大規模化を上回るペースで進展しており，複数のFPGAをまとめて巨大なFPGAとして扱うクラウド技術が早晩求められる．

FPGAそのものを大規模化する代わりに，複数のFPGAを利用することで大規模な問題にアプローチするマルチFPGAシステムの形態を図 **1** に示す．Amazon EC2 F1インスタンスに代表される既存のFaaSクラウドでは，図1(a)のようにホストPCにPCIeバス経由でハイエンドFPGAカードが接続されており，ユーザはホストPC（もしくは仮想マシン）経由でFPGAカードを排他的に利用できる．基本的にはFPGA間の直接的な通信は想定されておらず，ここでは疎結合型と呼ぶ．一方，我々が提案しているFlow-in-Cloud [3]では，図1(b)で示すように，ミドルレンジのFPGAを高速シリアルリンクで密結合することで，仮想的に巨大なFPGAを構築することを目指している．なお，検索エンジンBingで採用されているCatapult [1]は，FPGA間を2Dトーラスで接続することで，複数のFPGAをまたがったパイプライン処理を可能にしており，両者のハイブリッドと見なすことができる．以降では，マルチFPGAとは密結合型を暗黙的に意味することにする．



(a) 疎結合型　　(b) 密結合型

図 1　マルチFPGAシステムの形態

マルチFPGAを採用したクラウドサービスには，（1）複数のユーザからFPGAプールを安全に共有できるマルチテナントの実現，（2）高位合成言語で設計されたジョブの複数FPGAへの配置・実行，（3）データセンタ規模のスケーラビリティが必要になるが，我々の知る限りそのようなシステムはまだ存在しない．そこで，我々はFlow-in-Cloudプロジェクトにおいて開発しているFiCスイッチボードを複数用いたマルチFPGA環境を題材に，マルチFPGAクラウドシステムが要求する資源管理システムについて検討を行った．本稿では，その検討結果を受けて開発を進めている，マルチFPGA資源管理システムFiC-RMの概要と設計について報告する．FiC-RMはマルチテナント，マルチFPGAに対応した資源管理システムであり，Vivado HLSで設計されたアプリケーションをFiCスイッチボードへ配置・実行できる．さらにFiCスイッチボード間を高

1 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)
2 国立情報学研究所
National Institute of Informatics (NII)
3 慶應義塾大学
Keio University
4 東京大学
The University of Tokyo
a) takano-ryousei@aist.go.jp

© 2019 Information Processing Society of Japan

速シリアルリンクを介して接続し，スケーラブルかつ柔軟なネットワークを構築できる．

本稿の構成は次のとおりである．2 節では，我々が最終目標としているアクセラレータクラウド Flow in Cloud (FiC) の構想を概略する．3 節では，FiC プロジェクトの一環として開発しているマルチ FPGA 資源管理システム FiC-RM について述べ，4 節で，現在構築中のテストベッドと実験について報告する．5 節では，資源ディスアグリゲーションおよびマルチ FPGA システムに関する関連研究について言及する．最後に 6 節でまとめと今後の予定を述べる．

2. Flow-in-Cloud: アクセラレータクラウド

2.1 概要

ポストムーア時代に向けて，特定用途に特化したアクセラレータの需要が創出されると，ユーザの要求に応じて，多種多様なアクセラレータの柔軟な組み合わせを提供するクラウドシステムが必要になる．

アクセラレータは，一般的に PCIe カードのような形態で実装されるだけでなく，データの転送や前後処理を担うホスト PC が必要となるので，それら単体でシステムを構築することは困難である．既存のクラウドデータセンタアーキテクチャでは，図 1(a) でも示したように，アクセラレータを増やすにはホスト PC 単位で増やす必要がある．使用頻度が低いアクセラレータのためにホスト PC を待機させる必要があるなど，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ので，設備投資と運用コストの両面から非効率である．一方，我々が提案しているアクセラレータクラウドでは，**図 2** に示すように，計算ノード（ホスト PC）とアクセラレータを物理的に分離し，アクセラレータをプール化する．その上で，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．

我々は，アクセラレータクラウドの概念を実証するために，Flow in Cloud (FiC) システムの開発を進めている [3]．FiC システムでは，計算ノードとアクセラレータを搭載したアクセラレータノードが分離され，計算ノードと資源プール間は FiC ネットワークと呼ぶ高速ネットワークで密接続されている．FiC システムのオペレーティングシステムに相当し，全体の資源管理を担う FlowOS は，アプリケーションに必要なアクセラレータの組合せ（メタアクセラレータ）を資源プールから動的に構成し，計算ノードに接続する．FiC システムは，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メタアクセラレータを備えた計算機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．

FiC ネットワークは複数の FiC スイッチノード（詳細は

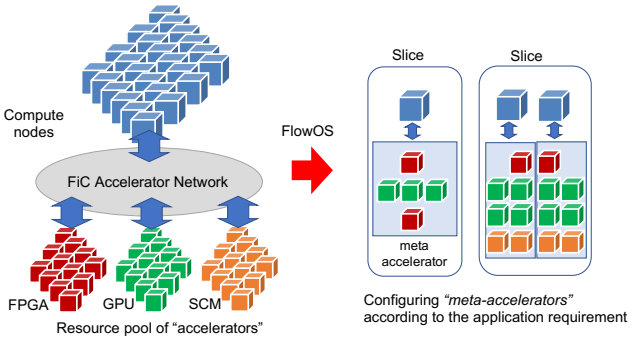


図 2 Flow-in-Cloud (FiC) が実現するアクセラレータクラウド

2.3 にて後述）から構成される．また，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，波長多重技術を活用してファイバあたりの通信を多重化できるが，電気ネットワークを採用する現在の FiC スイッチでは，その代わりに STDM (Static Time Division Multiplexing) 方式を採用する．FiC ネットワークの物理トポロジはメッシュ，トーラス，ツリー，ランダムネットワークなど，後述するルーティングテーブルを適切に設定することで，自由に再構成できる．

2.2 FlowOS

FlowOS は，アクセラレータクラウドの資源であるアクセラレータ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，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 [4]．

まず最初に FlowOS における用語を定義する．

- **タスク**: 一つのアクセラレータノード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**: アクセラレータノードの集合であり，FlowOS はスライス単位で資源を動的に確保・解放する．
- **ジョブ**: アプリケーションの実行単位であり，一つ以上のフローから構成され，それぞれが対応するスライス上で実行される．

FlowOS はタスクのフローグラフを記述・実行するための Python ライブラリである FlowOS Job と，資源管理を行う FlowOS-RM [5] から構成される．FlowOS-RM はクラスタ管理ミドルウェアである Apache Mesos と連携することで，ユーザの要求に応じて，Mesos が管理する計算ノード群を確保し，それぞれに必要なアクセラレータをホットプラグし，提供する．3 節で述べる FiC-RM は FlowOS-RM

© 2019 Information Processing Society of Japan

の主要コンポーネントの一つであり，FiC ネットワークの資源管理を担当する．

アプリケーションの開発から実行の流れを以下に示す．ユーザはアプリケーションを複数のタスクに分割し，処理の流れをフローとして記述する．各タスクをまず Pure Python で実装・検証した後，タスク単位でその処理をアクセラレータにオフロードするといったインクリメンタルな開発が可能である．ユーザがジョブを実行する際は，アプリケーションや必要なライブラリ一式をまとめたコンテナを準備し，FlowOS-RM にジョブとして投入する．FlowOS-RM は計算ノードとアクセラレータノードの確保や，ネットワークの設定を行い，ジョブを計算ノード上で起動する．FlowOS Job ランタイムでは，フローグラフから実行に必要な資源を解析し，スライスの生成を FlowOS-RM に要求する．ジョブの実行が終了すると，確保していたスライスは開放される．

2.3 FiC スイッチ

FiC ネットワークを構成する FiC スイッチの概要を述べる．FiC スイッチは Accelerator in Switch [6] のように通信中に FPGA を用いて一部の演算処理をオフロードすることが可能な構成になっている．したがって，FiC ネットワークはそれ自身が計算能力を有するマルチ FPGA システムと考えることができる．本節以降では，マルチ FPGA システムに必要な説明に言及する．詳細は論文 [3], [7] を参照されたい．

FiC スイッチボードには，**図 3** に示すように，FPGA (Xilinx Kintex UltraScale XCKU095)，DDR4 メモリ，8 対の双方向 10Gbps シリアルリンク，及び制御用の Rasberry Pi 3（以下，「RasPi」と記す）が搭載されている．FPGA は，STDM スイッチロジックが実装されたスタティック領域と，アプリケーションロジックを実装できるパーシャルリコンフィグレーション可能な領域（以下，「PR 領域」と記す）から構成される．PR 領域の変更はスタティック領域を停止することなく実行可能であるので，STDM スイッチの動作を停止することなく，アプリケーションロジックを入れ替えることができる．また，RasPi の GPIO を経由して，コンフィグレーションデータの書き込みや，ルーティングテーブル等，小規模なデータの授受が可能である．

FiC スイッチの 8 つあるコネクタには，それぞれフラットケーブル（Samtec FireFly）が接続され，1 コネクタを 1 ポートと呼ぶ．各ケーブルは 4 本の双方向シリアルリンクが束ねられている．このシリアルリンクは Xilinx のシリアル転送用 IP Aurora に接続され，9.9Gbps のシリアル転送を行う．シリアル転送は FiC スイッチ内でパラレルに展開され，ECC などの処理を経て，最終的に 170 ビット幅のデータが 2 クロックに 1 回，転送される．スタティック領域の動作周波数は 100MHz でなので，実効転送速度

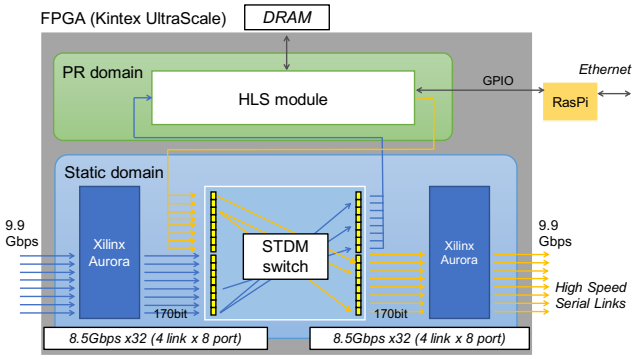


図 3　FiC スイッチボードの FPGA 構成

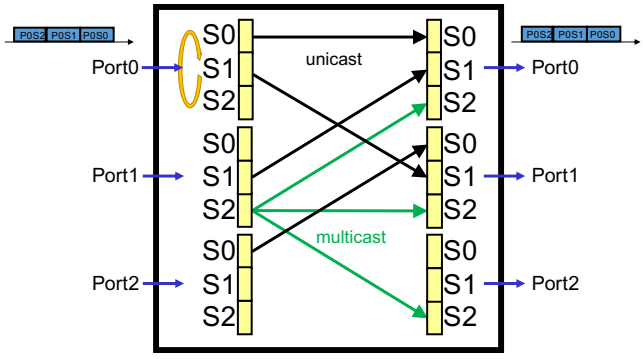


図 4　STDM (Static Time Division Multiplex) スイッチ

は 8.5Gbps となる．170 ビットの内訳は，MSB 側から有効ビット (1)，宛先スロット ID (16)，予約領域 (25)，ペイロード (128) である．予約領域は，シーケンス番号などに利用されることを想定している．

STDM 方式では，各々のリンクを複数のスロットに時分割するが，データはあらかじめ設定されたルーティングテーブル情報にしたがって静的に経路制御される．**図 4** に STDM スイッチの動作を示す．実際の FiC スイッチボードでは，8 ポート，4 リンク，そしてスロット数はプログラマブルであるが，ここでは説明のため 3 ポート，1 リンク，3 スロットとする．それぞれのポートから入ってきたデータの宛先はスロット毎に決まる．例えばポート 0 に入ってきたデータは，いったん FIFO に格納され，スロット 0 のタイミングになったタイミングでポート 0 へ，次のスロット 1 のタイミングでポート 1 へと転送される．スロット 2 は宛先が設定されていないので無視される．また，ポート 1 のスロット 2 では，複数のスロットを宛先に設定することで，マルチキャスト（ブロードキャスト）を行っている．深層学習を複数のアクセラレータノードにまたがって実装する場合など，ブロードキャストが必要になるが，本 STDM スイッチはその機能を効率的にサポートすることができる．

上記の転送ルールは各スイッチ内のルーティングテーブルによって決定される．STDM スイッチはスロット数が増えれるほど，それぞれのポートが転送を開始するまでの待

© 2019 Information Processing Society of Japan

ち時間が長くなり，レイテンシが悪化する．そこで与えられたワークロードの中で，スロット数を最小に抑えるようにルーティングテーブルを設定することも可能である [8]．このルーティングテーブルは RasPi から書き換え可能になっており，変更の都度，ハードウェアの再構成を行う必要性はない．

2.4 HLS モジュール

FiC システムでは，アプリケーションロジックを高位合成ツール Vivado HLS を用いて設計する．つまり C/C++ 言語を用いて FPGA の RTL コードを生成できる．HLS モジュールのサンプルコードを下記に示す．これは自分が受け取った整数値をインクリメントしてバケツリレーするプログラムである．

```
1  void hls_entry (
2      /* FiC switch board ID */
3      char id,
4      /* mapping table of slot ID from flow ID */
5      unit8 flow2slot[16],
6      /* RasPi I/O */
7      uint4 raspi_input, uint4 raspi_output,
8      /* STDM switch I/O */
9      uint169 input[1],  uint169 output[1])
10 {
11 #pragma HLS INTERFACE axis port=id
12 #pragma HLS INTERFACE axis port=flow2slot
13 #pragma HLS INTERFACE axis port=raspi_input
14 #pragma HLS INTERFACE axis port=raspi_output
15 #pragma HLS INTERFACE axis port=input
16 #pragma HLS INTERFACE axis port=output
17 #pragma HLS PIPELINE
18
19     uint128 tmp;
20     int fid = 0;
21
22     tmp = input[0];
23     tmp += 1;
24     output[0] = tmp | flow2slot[fid]<<153;
25     return;
26 }
```

hls_entry 関数は HLS モジュールのトップレベル関数であり，各種 ID，STDM スイッチと RasPi との I/O に関するパラメータが引数として渡される．例えば，169 ビット整数[*1]の配列 input[] は STDM スイッチからの入力，配列 output[] は出力と接続されている[*2]．配列の各要素はリンクに対応する．また，HLS モジュールでは，スロット ID を直接コーディングする代わりに，フロー ID という抽象化された識別子を用いる．後述する FiC-RM は，アプリケーションが必要とするタプル＜フロー ID，送信元論理ノード ID，送信先論理ノード ID ＞のリストから，フロー ID とスロット ID，論理ノード ID と物理ノード ID の割当，ルーティングテーブルの生成を実行し，FiC ネットワークを設定する．HLS モジュールでは，実行時に配列 flow2slot を参照することで，フロー ID からスロット ID を取り出すことができる．

3. マルチ FPGA 管理システム FiC-RM

3.1 概要

ユーザが設計した HLS モジュールを実行可能な FiC スイッチノードから構成される FiC ネットワークは，一種のマルチ FPGA システムと考えることができる．つまり，ここでは FiC スイッチをアクセラレータノードとみなす．FiC-RM では，**図 5** に示すように，（1）複数のユーザから FPGA プールを安全に共有できるマルチテナントの実現，（2）高位合成言語で設計されたジョブの複数 FPGA への配置・実行，（3）データセンタ規模のスケーラビリティの実現を目的とする．なお，各スライスは排他的であり，FiC スイッチの PR ドメインに複数のスライスのタスクが共存することはない．ただし，STDM スイッチは共有資源であり，PR ドメインの利用者に関わらず，データはルーティングされる．

ジョブは FlowOS Job で記述された Python スクリプトと，そのスクリプトから呼び出される Vivado HLS で設計された一つ以上のタスクから成り，一つ以上の FiC スイッチにまたがって配置される．なお，現時点では一つの FiC スイッチに配置されるタスクは一つに制限する．FPGA の資源利用率を上げるには，複数のタスクを融合して一つの大きなタスクとして HLS モジュールを生成する必要があるが，現時点では手作業で実施するしかない．タスクの融合や分割の自動化支援は今後の課題である．

FiC-RM の構成は**図 6** に示す通りであり，API サーバと，FiC スケジューラ，FiC モニタなどのサービス群と，各種構成情報を保持するデータベースから成る．

- **API サーバ**：アプリケーションに対する REST API を提供する．下記のスケジューラやモニタ等のサービス実体に処理を仲介する．テナント間の隔離を保証するために，アプリケーションから FiC スイッチを直接操作することを許さず，必ず API サーバを介して操作する必要がある．

[*1] FiC ネットワークで送受信されるデータは 170 ビット幅であるが，HLS モジュールには MSB の有効ビットを除く 169 ビットデータが授受される．

[*2] 現時点の FiC スイッチでは，仮想的なポート-1 の入出力が HLS モジュールに固定的に接続されており，これを前提にスケジューリングが行われる．

© 2019 Information Processing Society of Japan

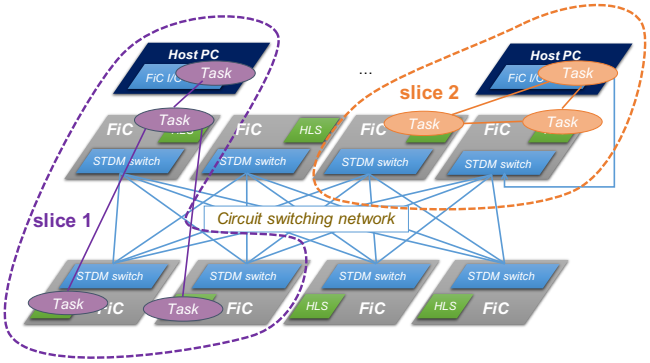


図 5　FiC スイッチを用いたマルチ FPGA クラウド

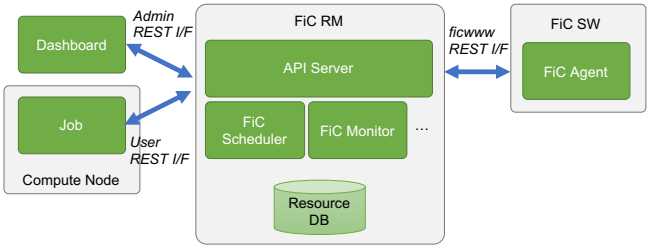


図 6　FiC-RM の概要

- **FiC スケジューラ**：アプリケーションからの要求に応じて，タスクの割当と，STDM スイッチのルーティングテーブルの生成を担う．
- **FiC モニタ**：FiC エージェントを介した FiC スイッチの制御，および FiC スイッチ（FiC エージェント）とアプリケーションの死活監視を行う．
- **FiC エージェント**：RasPi 上に常駐し，FiC スイッチを制御する REST API を提供する．

FiC-RM はジョブの投入を受け付けると，FiC スケジューラに資源割当を問い合わせし，その結果を受けて，FiC エージェント経由でコンフィグレーションデータを FiC ボードの PR 領域に書き込み，スイッチにルーティングテーブルを設定する．さらに，ジョブ実行中に計算ノードと FiC スイッチ間で RasPi を介したデータ転送も可能である．**表 1** に FiC-RM API サーバが提供する REST API を示す．本 API はジョブから呼び出されるユーザ API と，ダッシュボードなどの管理者ツールが使用する管理者 API に大別できる．

スライス作成時は，下記に示す JSON 形式でスライスの仕様を記述し，所定の URI に POST する．スライスの作成に成功するとスライス ID が返ってくるので，ジョブではこれを用いて，状態の取得や，データ送受信を実行する．

この例では，LeNet5 を実行する二つのタスク（`lanet5a`, `lanet5b`）から構成されるスライスの作成を要求している．タスクフローの定義 `flow_graph` では，ノード `src_node_id` からノード `dst_node_id` に向けて片方向の通信を定義している．この通信路を識別するフロー ID（`flow_id`）は 0 である．なお，`src_node_id` と `dst_node_id` への FiC ノード

表 1　FiC-RM REST API 一覧

Path	Method	説明
ユーザ API		
/slice	POST	スライスの確保
/slice/<slice-id>/start	PUT	ジョブの実行開始
/slice/<slice-id>/status	GET	ジョブの情報の取得
/slice/<slice-id>/<node-id>	PUT	ノードへのデータ送信
/slice/<slice-id>/<node-id>	GET	ノードからのデータ受信
/slice/<slice-id>	DELETE	スライスの開放
管理者 API		
/admin/slice	GET	スライスリストの取得
/admin/slice/<slice-id>	GET	スライス情報の取得
/admin/node	GET	ノードリストの取得
/admin/snode/<node-id>	GET	ノード情報の取得
/admin/slice	DELETE	スライスの強制開放

の割り当て，および `flow_id` へのスロット ID の割り当ては，3.2 節で示す FiC スケジューラが実行する．タスクの定義 `task_list` では，各ノードに配備するコンフィグレーションデータを定義している．なお，現在の実装では PR 領域を書き換える際には，一度クリア用のコンフィグレーションデータを書き込んだ後，新たに実行するコンフィグレーションデータを書き込む必要があるので，クリア用のコンフィグレーションデータの記載も必要である．

```
{
  "flow_graph": {
    "edge_list": [
      {
        "src_node_id": 1,
        "dst_node_id": 2,
        "flow_id": 0
      }
    ],
  }
  "task_list": [
    {
      "node_id": 1,
      "application_partial_bin": {
        "mode": "sm16pr",
        "bitname": "lenet5a",
        "bitstream": "lenet5a.base64",
      },
      "application_clear_bin": {
        "bitstream": "lenet5a.clear.base64"
      }
    },
    {
      "node_id": 2,
      "application_partial_bin": {
```

© 2019 Information Processing Society of Japan

```
        "mode": "sm16pr",
        "bitname": "lenet5b",
        "bitstream": "lenet5b.base64",
      },
      "application_clear_bin": {
        "bitstream": "lenet5b.clear.base64"
      }
    }
  ],
}
```

3.2 FiC スケジューラ

投入されたジョブはジョブキューに格納され，「実行可能になり次第」，原則，ジョブの到着順に実行される．ただし，前のジョブの開始時間に影響することなく実行できる場合には，後のジョブを先に実行することを許可する．いわゆる，並列計算機のスケジューラで一般的に用いられているバックフィリングを採用している．ジョブが「実行可能な状態」とは，必要ノード数を確保し，そのノード間通信をサーキットスイッチネットワークが実行できる状態を指す．運用に関しては，あるジョブが完了した場合，後継のジョブは，そのジョブが利用していた通信資源を利用可能である．したがって，スケジューラは，ノードとネットワークの両方の資源管理を行うことが必要となる．

この中で，我々が取り組む課題は以下である．現実には，サーキットスイッチネットワークが提供できるサーキット数は限られる．具体的には，1 つのリンクを通過できる出発地／目的地の通信対の数が制限される．そのため，(1) 1 つのジョブをできるだけ隣接したノード群に割り当てるジョブマッピングと (2) ルーティング，トポロジを工夫することにより，1 つのリンクを通過できる出発地／目的地の通信対の最大数を抑制する最適化が重要となる．(2) の技術については文献 [8] で開発済みである．今後，(1) に関してジョブマッピングとバックフィリングアルゴリズムの改良を行い，ノードとネットワークの両方に関して高い利用効率を実現するよう設計中である．

なお，複数の FPGA を用いた既存の計算システム [9] では通常，パケット通信を前提としており，ネットワークの混雑状況により通信遅延のゆらぎが極めて大きい．一方，本システムの研究では，マルチ FPGA から構成される仮想 FPGA に要求される決定的な動作を実現するために，通信遅延が相応に予測可能なサーキットスイッチネットワークを採用した．

3.3 FiC エージェント

前述したとおり FiC スイッチボードには FPGA の制御用に RasPi が搭載されている．RasPi 上には Linux OS（Raspbian OS）が動作しており，FiC エージェントは UNIX プロセスとして実装されている．FiC エージェントは，**表 2** に示す ficwww REST API [10] を提供すると共に，当該 API に対応する FiC スイッチへの操作を実行する．

表 2 FiC エージェント REST API 一覧

Path	Method	説明
/fpga	POST	FPGA のコンフィグレーション
/switch	POST	ルーティングテーブルの設定
/hls	POST	HLS モジュールへの制御・データ転送（reset, start, receive4, send4）
/status	GET	FiC ボード状態の取得（dipsw, led, link, pwrok, done, bitname, conftime）
/regread	POST	FiC レジスタの読み込み
/regwrite	POST	FiC レジスタへの書き込み

4. テストベッドの構築と実証実験

現在，FiC スイッチボード 6 枚から構成されるテストベッドを構築し（**図 7**），深層学習を用いた画像識別 [11] などのアプリケーションを用いて，マルチ FPGA のフィージビリティを検証している．

資源管理システムとの連携の観点では，FlowOS から 1 台の計算ノードと複数枚の FiC スイッチを制御し，LeNet5 を用いた手書き文字認識など 2 種類のタスクを，STDM スイッチの動作を止めることなく，再構成するデモンストレーションに成功している [12]．しかし，本稿執筆時点では，FiC-RM の実装が完了していないので，ルーティングテーブルの設定は自動化されておらず，ネットワークの設定には制約があった．また，計算ノードから FiC スイッチへデータを供給するには，RasPi を経由するしか手段がなく，大規模なデータを用いた実験が困難であった．そこで，計算ノードを FiC ネットワークに直接接続するための FiC I/O カード（Xilinx Kintex UltraScale KCU1500 がベースの PCIe カード）を実装し，その動作を確認中である．

今後は本テストベッドに FiC-RM を組み込み，クラウドシステムとしての実証を行う予定である．さらに，FiC ネットワーク本来の Accelerator in Switch としての利用に止まらず，マルチ FPGA システムとして本格的に活用することを想定して，DSP を強化して計算能力を増強した FPGA を搭載した FiC スイッチボードを増設する予定である．

5. 関連研究

資源ディスアグリゲーション技術は，多種多様なアクセラレータを効率的に活用できるため，ポストムーア時代の計算機システムアーキテクチャとして注目されている．このような背景から，Intel Rack Scale Design [13]，HP The Machine [14]，IBM Composable System [15]，UCB

© 2019 Information Processing Society of Japan

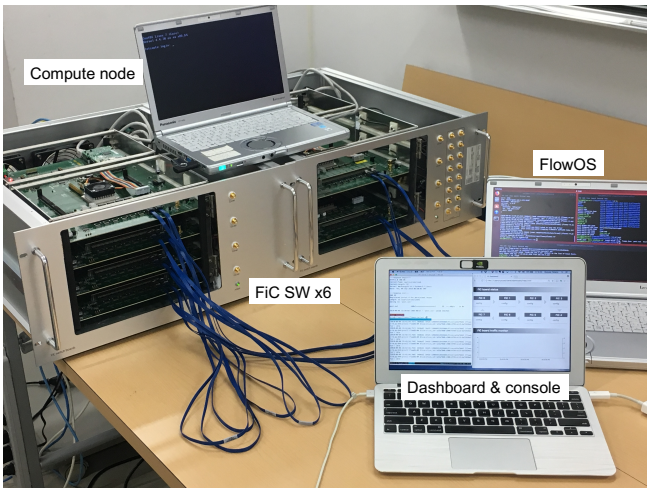


図 7　FiC テストベッドの外観

FireBox [16]，EU dReDBox [17] などのシステムが提案されている．これらの研究では，主にハードウェアレベルでどのように資源ディスアグリゲーションを実現するかに焦点が当てられており，Gen-Z [18] や光ネットワーク等の次世代インタコネクト技術を用いて，いかにラックやデータセンター規模の資源管理を実現するか議論されている．また，LegoOS [19] は，ディスアグリゲーションされたハードウェア資源上で動作する一種の分散 OS である．LegoOS は Splitkernel アーキテクチャを採用することで，プロセスとメモリを分離し，ディスアグリゲートされたハードウェアコンポーネントをまたがってプロセスを構築できる．

FPGA を採用した IaaS である FaaS (FPGA as a Service) クラウドサービスとしては，Amazon，Alibaba，Baidu がすでに提供を開始している他，Microsoft は自社の検索サービスや機械学習処理を効率化するために FPGA を活用している（Catapult [1]，Brainwave [9]）．今後の FPGA クラウドの研究開発の方向性として，複数の FPGA を一つのシステムとして管理するマルチ FPGA と，オーバレイ技術等を用いて，物理 FPGA に依存しないより抽象度の高いコンフィグレーションデータを実行可能にする FPGA 仮想化の検討が進んでいる [20]．マルチ FPGA システムとしては，前述の Catapult の他，欧州で始まった EuroEXA [21] プロジェクトが注目を集めている．一方，FPGA 仮想化を支援するシステムソフトウェアとしては，AmorphOS [22] や ReconOS [23] などが提案されている．FiC はマルチ FPGA システムであり，Catapult 等の既存システムとの違いは，近年成熟しつつある高位合成によるプログラミングを採用している点である．

アプリケーションフレームワークの観点からは，深層学習のモデル記述，学習，推論までを容易に実行するために，TensorFlow，PyTorch，Chainer 等の深層学習フレームワークが開発されている．さらに最近は，学習環境と推論環境の分離や，Google TPU や PFN MN-Core 等，様々な実行環境へのデプロイを目的に，TensorFlow XLA，TVM [24]，Chainer-X といった実行デバイスを選択可能にする提案され始めている．FlowOS では，深層学習をメインターゲットとしつつ，より汎用なタスクフローの記述を指向している．

6. まとめと今後の予定

本稿ではマルチ FPGA クラウドシステム FiC-RM の構想と現状について報告した．高水準言語でプログラム可能で，汎用かつ柔軟性のあるマルチ FPGA クラウドシステムは我々の知る限り前例がない．また，我々の最終目標は，異種アクセラレータを統合したアクセラレータクラウドの実現であるが，本研究はその一里塚として位置づけている．本稿執筆時点では，FiC-RM の基本設計を終え，各コンポーネントの実装を開始したところである．当面は FPGA のみのホモジニアスなシステム構成を前提に，4 節で紹介したテストベッド上で各種アプリケーションを用いた FiC-RM の動作確認を行う予定である．

謝辞　本研究は，NEDO 委託事業「高効率・高速処理を可能とする AI チップ・次世代コンピューティングの技術開発（旧：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．

参考文献

[1] Putnam, A., Caulfield, A., Chung, E., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gray, J., Haselman, M., Hauck, S., Heil, S., Hormati, A., Kim, J.-Y., Lanka, S., Peterson, E., Smith, A., Thong, J., Xiao, P. Y., Burger, D., Larus, J., Gopal, G. P. and Pope, S.: A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services, *41st International Symposium on Computer Architecture* (2014).

[2] Ouyang, J., Lin, S., Qi, W., Wang, Y., Yu, B. and Jiang, S.: SDA: Software-defined accelerator for large-scale DNN systems, *2014 IEEE Hot Chips 26 Symposium (HCS)*, pp. 1–23 (online), DOI: 10.1109/HOTCHIPS.2014.7478821 (2014).

[3] 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).

[4] 高野了成，池上　努，須崎有康，田中　哲，広渕崇宏：ベアメタルクラウド向けアクセラレータシステムを実現する Flow OS の提案，情報処理学会研究報告 2017-HPC-163，pp. 1–7 (2018).

[5] Takano, R., Suzaki, K. and Koie, H.: FlowOS-RM: Disaggregated Resource Management System, *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (Poster)* (2018).

[6] Tsuruta, C., Kaneda, T., Nishikawa, N. and Amano, H.: Accelerator-in-Switch: a framework for tightly coupled switching hub and an accelerator with FPGA, *International Conference on Field-Programmable Logic and*

© 2019 Information Processing Society of Japan

Applications (FPL) (2017).
[7] 畔上佳太, 武者千嵯, 弘中和衛, Ahmed, A. B., 天野英晴: マルチ FPGA ボード間通信を行うスイッチの開発, 電子情報通信学会技術研究報告 118(215), pp. 55–59 (2018).
[8] Hu, Y. and Koibuchi, M.: Optimizing Slot Utilization and Network Topology for Communication Pattern on Circuit-switched Parallel Computing Systems, Vol. E102-D, No. 02, pp. 247–260 (2019).
[9] Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., Heil, S., Patel, P., Sapek, A., Weisz, G., Woods, L., Lanka, S., Reinhardt, S., Caulfield, A., Chung, E. and Burger, D.: A Configurable Cloud-Scale DNN Processor for Real-Time AI, *45th International Symposium on Computer Architecture* (2018).
[10] ficwww: `https://github.com/hungalab/ficwww` (2019).
[11] 武者千嵯, Ahmed, A. B., 工藤知宏, 天野英晴: FPGA スイッチボードへの深層学習アプリケーションの実装, 電子情報通信学会技術研究報告 118(340), pp. 33–38 (2018).
[12] Flow-in-Cloud project demonstration: `https://youtu.be/0JE5BYft700` (2019).
[13] Intel Rack Scale Design: `https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design-overview.html` (2018).
[14] Hewlett Packard Labs: The Machine: Our vision for the future computing, `https://www.labs.hpe.com/the-machine` (2018).
[15] Chung, I.-H., Abali, B. and Crumley, P.: Towards a Composable Computer System, *The International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 18)*, pp. 137–147 (2018).
[16] Asanovic, K.: Firebox: A hardware building block for 2020 warehouse-scale computers (2014).
[17] Katrinis, K., Syrivelis, D., Pnevmatikatos, D., Zervas, G., Theodoropoulos, D., Koutsopoulos, I., Hasharoni, K., Raho, D., Pinto, C., Espina, F., Lopez-Buedo, S., Chen, Q., Nemirovsky, M., Roca, D., Klos, H. and Berends, T.: Rack-scale disaggregated cloud data centers: The dReDBox project vision, *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 690–695 (2016).
[18] Gen-Z Consortium: `https://genzconsortium.org` (2019).
[19] Shan, Y., Huang, Y., Chen, Y. and Zhang, Y.: LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation, *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 69–87 (2018).
[20] Vaishnav, A., Pham, K. D. and Koch, D.: A Survey on FPGA Virtualization, *International Conference on Field-Programmable Logic and Applications (FPL)* (2018).
[21] EuroExa Project: `https://euroexa.eu/` (2019).
[22] Khawaja, A., Landgraf, J., Prakash, R., Wei, M., Schkufza, E. and Rossbach, C. J.: Sharing, Protection, and Compatibility for Reconfigurable Fabric with AmorphOS, *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 107–127 (2018).
[23] Wiersema, T., Bockhorn, A. and Platzner, M.: Embedding FPGA Overlays into Configurable Systems-on-Chip: ReconOS meets ZUMA, *IEEE International Conference on Reconfigurable Computing and FPGAs*, pp. 1–6 (2014).
[24] Chen, T., Moreau, T., Jiang, Z., Zheng, L., Tong, S. J., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C. and Krishnamurthy, A.: TVM: An Automated End-to-End Optimizing Compiler for Deep Learning, *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 69–87 (2018).

© 2019 Information Processing Society of Japan