一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
ICM2024-13(2024-07)

# アプリケーション自動分割のIoTサービス適用確認

山登庸次†

† NTT ネットワークサービスシステム研究所，東京都武蔵野市緑町 3-9-11
E-mail: †yoji.yamato@ntt.com

**あらまし**　私はプログラムコードを，配置先環境を適切に利用できるように，変換等を自動で行い動作させる，環境適応ソフトウェアのコンセプトを提案してきた．本稿は，環境適応ソフトウェアの要素として，IoT サービスにも自動分割方式を適用することで，ユーザが独自カスタマイズを容易化できるようにする．サンプルアプリケーションを自動分割し，適材適所動作に容易に変更できることを確認する．
**キーワード**　環境適応ソフトウェア，IoT サービス，機能追加，自動分割，動的分析

# IoT service adopting confirmation of application automatic split

Yoji YAMATO†

† Network Service Systems Laboratories, NTT Corporation, 3-9-11, Midori-cho, Musashino-shi, Tokyo
E-mail: †yoji.yamato@ntt.com

**Abstract**　We have proposed the concept of environment-adaptive software that automatically converts and operates program code so that it can appropriately utilize the environment in which it is placed. This paper applies an automatic dividing method to IoT services as an element of environment-adaptive software, thereby making it easier for users to customize their own services. We confirm that the sample application can be automatically divided and easily changed to put the right people in the right places.
**Key words**　Environment-adaptive software, IoT services, Function addition, Automatic division, Dynamic analysis

## 1. はじめに

AI（Artificial Intelligence）進歩も有り，CPU（Central Processing Unit）だけでなく，GPU（Graphics Processing Unit），FPGA（Field Programmable Gate Array）や IoT（Internet of Things）機器（[1]- [11] 等）等ヘテロジニアスなハードウェアが，多くのアプリケーションに用いられるようになっている．Amazon 社や Microsoft 社 [12] は，クラウド技術使い（例えば [13]- [17]），GPU や FPGA 処理を提供，利用している．しかし，ヘテロジニアスなハードウェアを利用するためには，ハードウェア特性を意識したプログラムが必要となり，大半の技術者にとって壁が高い．GPU では CUDA（Compute Unified Device Architecture）[18]，FPGA では OpenCL（Open Computing Language）[19]，IoT 機器では IoT PF（Platform）知識が必要となってくることが多い．

ヘテロジニアスなハードウェアをより活用していくためには，高度な知識を持たない通常のソフトウェア技術者でも，それらを最大限に活用できるようにする PF が必要と考える．技術者が簡易プログラムと同様の手法で処理を記述したソフトウェアを，分析して，適用する環境（GPU，FPGA，IoT 機器等）にあわせて，適切に変換，設定を行い，環境に適応した動作をさせることを，自動で行うことが今後求められていく．

そこで，私は，通常 CPU 向けコードを，配置先環境を適切に利用できるよう，変換等を自動で行い動作させる，環境適応ソフトウェアのコンセプトを提案している．環境適応ソフトウェア要素として，コードを，GPU，FPGA に自動オフロードする方式を提案している [20]- [24]．IoT 機器向けには，ユーザは基本サービスを選びユーザ独自処理と利用 IoT 機器を指定すれば，IoT サービスを自動構築する方式も提案している．また，汎用的プログラムでも，ユーザが行いたい処理を追加変更できるようにするため，アプリケーションを関連処理に基づいて分割して，分割境界を元に変更を局所化することで，適材適所での利用できるようにし，サービス追加変更の容易化している．

しかし，これまでの自動分割方式検証は，C 言語の計算系のアプリケーションに限られていた．そこで，本稿では，IoT サービスにも自動分割方式を適用することで，これまでは，事前に事業者の基本サービス提供が必要だった IoT サービスでも，ユーザが独自カスタマイズを容易化できるようにする．温

度表示のサンプルアプリケーションを自動分割し，適材適所動作に容易に変更できることを確認する．

## 2. 既存技術

### 2.1 市中技術とこれまでの環境適応ソフトウェア

GPU を一般的計算にも用いる GPGPU（General Purpose GPU）[25] の環境に NVIDIA は CUDA を提供している．FPGA，GPU 等ヘテロジニアスなハードウェアを共通的に扱う仕様として OpenCL がある．容易に GPU 等を用いるため，ディレクティブで GPU 処理等行う行を指定して，GPU 処理等のバイナリファイルを作成する取り組みがあり，OpenMP（Open MultiProcessing）[26] や OpenACC（Open Accelerators）[27] 等仕様と，それを解釈実行する gcc や PGI [28] 等コンパイラがある．これらにより，ヘテロジニアスなハードウェアを利用はできても，データコピーの影響で，性能改善は容易でないのが現状である．GPU や FPGA の際はメモリも異なるためより複雑で，性能改善には OpenCL や CUDA を駆使した手動チューニングが必要となる．その中で，著者は進化計算手法である遺伝的アルゴリズム（GA）を用いた自動オフロードを提案している．

GPU 等のアクセラレータでなく，メモリ等のリソース量が少ない端末が多い IoT 機器も，ヘテロジニアスなハードウェアの利用である．IoT に関する標準としては，M2M プラットフォームの oneM2M が標準化されている．IoT センサからセンシングデータを IoT GW で集約し MQTT（Message Queue Telemetry Transport）や HTTP 等のプロトコルでクラウドに送り，データ集計や保持等共通的処理はクラウド上 IoT PF が行い，加工等処理を行って，その結果を企業幹部等のユーザにクラウドサーバで表示する形が多い．IoT PF としては，大手クラウド事業者の Amazon や Microsoft が，AWS IoT や Azure IoT 等の IoT PF をクラウドの中で提供しており，デファクトスタンダードに近くなっている．

以前に著者は，環境適応ソフトウェアの処理として，図 1 の全体像を提案した．環境適応ソフトウェアの処理は，Step1-6 は，コード分析し，配置環境に応じたコード変換，リソース量調整，配置場所調整，検証の一連を行い，アプリケーション運用を開始する，Step7 は，アプリケーションの運用開始後に，実際の利用特性等を分析して，必要な再構成を行う．Step1-7 の流れを，GPU や FPGA 等のアクセラレータ，IoT 機器等の少リソース端末の両方で検証してきた．更に，コード分析できるアプリケーションを増やすため，ユーザが行いたい処理を追加変更する際に，アプリケーションを関連処理に基づいて分割して，変更を局所化して容易化する，自動分割方式を提案している．

### 2.2 本稿の課題の整理

本稿の課題を整理する．著者は環境適応ソフトウェアのコンセプトを以前提案し，GPU や FPGA 等アクセラレータにプログラムを自動オフロードする方式や，ユーザは基本サービスと処理したい計算処理コードと IoT 機器を指定すれば，自動でサービス化する IoT 適応方式を検証してきた．また，汎用的プログラムの，コードの処理関係を元に自動分割して，変更を局所化することで，ユーザが機能追加や変更をしやすくする方式を検証してきた．

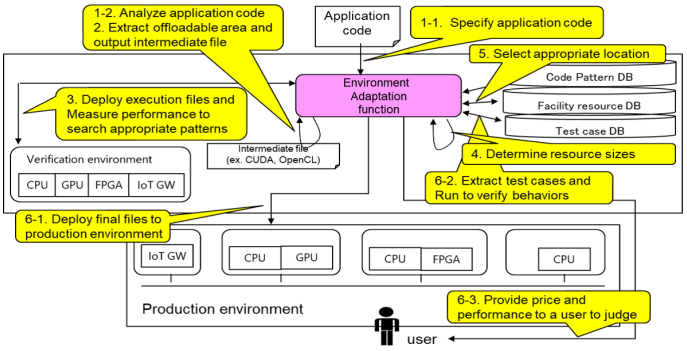



図 1　環境適応ソフトウェアの全体像

しかし，汎用プログラムの自動分割はこれまで，C 言語の計算系アプリケーションの検証だけであり，多くのユーザが使うと思われる IoT 等の新たなサービスは検証されていなく，有効性が示されていなかった．そこで，本稿は，IoT サービスでの自動分割を狙いとし，IoT ならではの考慮事項を踏まえて自動方式を微変更して，Azure IoT PF でも使われる IoT サービスでも分析できるように実装し，検証する．

## 3. 汎用的プログラム自動分割の IoT 適用

### 3.1 IoT 向け自動分割方式

プログラムに，ユーザが行いたい独自処理を追加変更することを考える．それなりに行数があるアプリケーションは，変更影響が広範囲に及ぶため，追加変更には関連する機能に影響がないかのチェックに大きな稼働がかかる．そこで，アプリケーションを関連する処理で分割して，分割境界を元に変更を局所化することで，追加変更を容易化する．

分析手法で，アプリケーションを実際には動かさずに，ソースコードの関数の呼び出しや書き込み等のリレーションを見る，静的分析手法がある．静的分析では，まず，関数同士の呼び出し関係が把握できる．次に，ある関数が同じデータに書き込む関数かどうかも把握できる．一般に，呼び出し関係があったり，同じデータに書き込むでそれを使う関数は，関連が深いため，分割する際にはグループ化する必要がある．

サンプルテストケースを用いてアプリケーションを実際に動かし，実行されたログ等の情報を見る，動的分析手法がある．動的分析では，関数同士の呼び出し関係でも，ユーザが指定するサンプルテストケースで実際に使われる呼び出し関係が抽出できる．さらに，データベース（DB）を用いるアプリケーションの場合，DB アクセスログを抽出して，連続的に実行しているデータアクセス命令を発見し，一連処理として実行しているプログラム範囲を見つけることができる．また，DB でなくファイルアクセスの場合でも，ファイルアクセスログから，一連処理として実行しているプログラム範囲を見つけることができる．

環境適応ソフトウェアの，GPU 自動オフロードの際，GPU で計算処理可能かは静的分析で分かるが，GPU 処理した際の

性能は実際に測定しないと分からないのが通常であり，静的分析と動的分析を組合わせて，オフロード部を自動探索していた．動的分析は，ユーザが使うサンプルテストケースを実際に動かし性能測定するため，個々のユーザ毎に異なる対応をするために重要な要素であった．そこで，IoT サービスの自動分割でも，個々のユーザに対応するために，静的分析と動的分析を組合わせた方式をとることにする．IoT サービスでは，IoT データの集約が必ず必要となる特徴となるため，特にプログラムを分析する際に，通信が行われる処理の場合は，必ず別グループに分割して，別筐体となってよいようにする．

提案方式の動作は図 2 のようになる．IoT サービスのプログラムコード群を静的分析，動的分析し，呼出関係がある情報をもとに分割を行う．

ファイル A に関数 a1,a2,a3„, ファイル B に関数 b1,b2,b3„, が定義されているとする．

静的分析では，Main 関数相当から呼ばれた先の関数 a1 が b2 を呼んでいる場合に，AB が一回と加算する．それを全ファイルに対してカウントする．その結果，AB：20，AC：30，BA：10 等が分析結果になる．

動的分析では，サンプル試験を一定数実施した結果，Main 関数から呼ばれた先の関数 a1 が b2 を 10 回呼んでいる場合に，AB が 10 回と加算する．それをサンプル試験実施分だけカウントする．その結果，AB：10，AC：20 等が分析結果になる．

動的分析の結果，1 回でも呼び出しがあるファイル群は同じグループとする．静的分析の結果，3 回以上呼び出しがあるファイル群は同じグループとする．静的分析の結果，2 回以下呼び出ししかないファイル群は全て同じグループとする．静的分析の結果，通信が 1 回でも行われるファイル群は別グループとする．

このようにすることで，動的分析でユーザが指定するサンプルテストケースで呼び出し関係があるファイル群は必ず同じグループとなる．また，静的分析で全ファイルを分析した際に一定度の回数である 3 回以上呼び出し関係があるファイル群も同じグループになる．Stand Alone 的で他の関数とあまり関わりがない関数のファイルは残りのグループになる．また，IoT で特徴となる IoT データ集約に関わる通信がある場合は別グループになる．呼び出し関係に基づいて自動分割することで，ユーザが機能追加する際の確認範囲を小さくすることができる．

### 3.2 実　　装

自動分割の対象は C 言語と Python のアプリケーションとし，C 言語の解析には Clang，Python の解析には ast を用いる．アプリケーションを解析する実装は，Python 3 で行う．実装は，入力として，ソースコードファイル群と，サンプルテストケース情報を受け，出力として，静的分析の場合のグループ情報と，動的分析の場合のグループ情報を出力する．グループ情報は，イ（A,B,C），ロ（D,E）のような形で，分割されたグループとそれに所属するファイル群からなる．また，サンプルテストケースを実行する一定回数は 10 回とした．

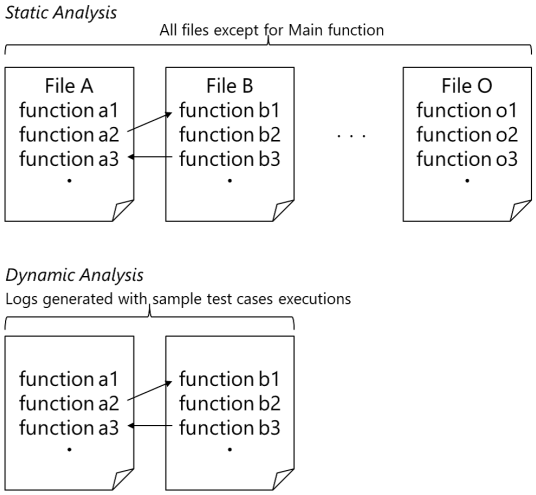
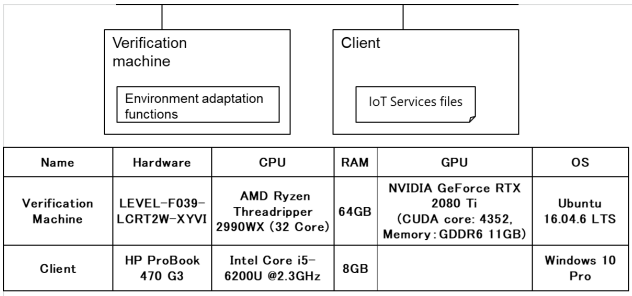



図 2　IoT サービスのプログラム自動分割時の分析




| Name | Hardware | CPU | RAM | GPU | OS |
|---|---|---|---|---|---|
| Verification Machine | LEVEL-F039-LCRT2W-XYVI | AMD Ryzen Threadripper 2990WX (32 Core) | 64GB | NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory: GDDR6 11GB) | Ubuntu 16.04.6 LTS |
| Client | HP ProBook 470 G3 | Intel Core i5-6200U @2.3GHz | 8GB | | Windows 10 Pro |

図 3　評 価 環 境

## 4. 評　　価

### 4.1 評価対象と評価手法

評価対象は，Python アプリケーションで，IoT データを環境センサから集め，IoT GW が一括して IoT PF に送信し，IoT PF で結果を Web 表示する基本的な IoT サービスとする．

センサ：Omron 2JCIE-BU01 から 1 分毎に環境情報の温度，湿度，照度，気圧，騒音，3 軸加速度，eTVOC，不快指数，熱中症警戒度，振動情報を収集．

IoT GW：Armadillo-IoT G3L.

IoT PF：Azure IoT．データ処理は，Azure IoT Hub の REST API を介して行われ，結果等は同じ Azure の VM に渡される．

自動分割された後，ユーザが追加するプログラム：温度，湿度の平均値を計算して出力．IoT サービスに対して，静的分析，動的分析の場合の分割グループが表示されるので，分割状況を確認し，元々のアプリケーションの全行数と，分割されたアプリケーションの行数をカウントし比較する．分割されたアプリケーションに機能追加して，温度，湿度の平均値が Web 表示されるサービス化は，従来技術で行い，画面確認は目視で行う．

評価環境を図 3 に示す．本稿の環境適応は高速化するわけではないので，ツールはどのマシンで動かしても良いとは言えるが，静的分析，動的分析は長時間がかかるため，十分なスペックのマシンでの動作が必要である．

| Application | Divided group # | total code # | target group code # |
|---|---|---|---|
| Show temperature | 2 | 480 | 300 |

図 4　提案方式での自動分割例

### 4.2　結果と考察

図 4 は，温度表示アプリケーションを提案方式で自動分割した際の，分割グループ数と，新機能をアプリケーションに追加する先のグループの行数と，アプリケーション全体の行数を示している．

まず，分割グループを見ると，Omron センサからデータ収集する部分と，センサデータを蓄積して表示する部分は分割されており，IoT GW へと IoT PF へで，分離して機能配置することでが出来る．温度表示アプリケーションは 2 つに分割され，全体 480 行に対して，新機能追加グループは 300 行となっている．自動で分割がされ，ユーザが温度や湿度の平均値計算等の独自処理を追加した際の確認する範囲が小さくなり，機能追加変更が容易になっていることがわかる．

著者の環境適応ソフトウェアでのこれまでの IoT 機器を用いたサービス自動構築は，事前に基本となるサービスを事業者が準備していることが前提だった．今回，自動分割技術を IoT サービスに適用できるようにすることで，基本サービス等ない IoT サービスにユーザが行いたい機能追加をできるようにした．自動分割では，IoT GW と IoT PF で適切に分割して配置することができるため，適材適所利用もできると考える．

コストを考察する．IoT サービスに機能追加することを考えた際に，今回 Python のアプリケーションをサンプルに利用した．機能追加する際に確認する行数が 2/3 になった．影響確認稼働が低くなり，改造コストを抑えることが出来ると考える．

今回分析には，ユーザ毎の対応を行うために，従来も良く使われていた静的分析に，動的分析を組合わせる方式を行った．分析自体は，関数の呼び出し関係を用いており，それで十分分割できることを確認したが，同じデータへの書き込み，DB やファイルのアクセスログ，連続実行等，更なる情報により，より詳細な分析が可能となる．

## 5. 関 連 研 究

GPU へのオフロードについては，[29] [30] [31] 等の既存研究があげられる．[29] は，C++ expression template の GPU オフロードのため，メタプログラミングと JIT コンパイル利用をあげており，[30] [31] は OpenMP を使った GPU へのオフロードに取り組んでいる．新たな開発モデルや指示句の手動挿入等を行わず，著者が狙う様な既存コードを自動で GPU 向けに変換する研究は少ないと言える．

FPGA のオフロードに関しては [32] [33] [34] 等の研究がある．[32] は，ネストされたループを FPGA にオフロードする方法を提案している．[34] は，CPU-FPGA ハイブリッドマシンを使用して，わずかに変更された標準 C 言語でプログラムを高速化している．これらの方法では，OpenMP または他の仕様を使用して並列化するパーツなどの命令を手動で追加する必要がある．

[35] [36] は，IoT フレームワークに関する研究である．[35] はマイクロサービス開発手法に特化したフレームワークで，効率を高めている．[36] はスマート行政と適用ターゲットに特化し，効率を高めている．IoT フレームワークは生産性等高くする効果はあるが，特化した手法や対象に応じた，専門の知識が必要になる等があり，大手クラウド業者の IoT PF を用いた際の開発と同様の課題があると言える．

[37] [38] は，IoT における自動設定の研究である．MQTT デバイスの自動設定を Home GW の実施，産業用 IoT 環境の自動設定等が検討されている．IoT 機器は，センサ，アクチュエータ等，機器数がこれまでよりもはるかに大規模になるため，自動設定による設定稼働削減は大きなメリットがある．しかし，設定の自動化は検討されているが，IoT サービス自体を自動構築する試みは見当たらない．

[39] は，大規模業務アプリケーションを分析して機能ごとに分割，可視化する技術である．対象が大規模業務アプリケーションであることは異なるが，アプリケーションを分析して，分割する狙いは類似である．記載技術は，プログラムの呼び出し先及び書き込むプログラム群を把握し関係を重みづけしてクラスタリングを行う．更に，データベースへのアクセスログを利用できる場合，一連の処理として実行すべき範囲を見つけ，クラスタリング結果を補正する．ERP 等の大規模業務アプリケーションが対象で，DB アクセスログ等多くの情報を分割に利用している点が異なる．本技術は，DB を使わないような小規模だったり永続性が不要なアプリケーションでも分割して機能追加できるよう，シンプルな情報を分析に用いている．

GPU，FPGA へのオフロードに関しては，OpenMP や OpenCL 等の指示を手動追加し，それに従ってオフロードするのが主流で，既存コードを自動でオフロードする様な取組みはほとんどない．IoT に関してもそれは同様であり，フレームワークや自動設定の研究はあるが，IoT PF 等の知識が無いユーザでも容易に利用できる取り組みは無い．IoT 以外の汎用的プログラムへのユーザの機能追加に関しては，大規模業務アプリケーションでの検討はされているが，一般ユーザが使うような小規模アプリケーションでの機能追加の容易化は取組みがない．

## 6. ま と め

本稿では，私が提案している，ソフトウェアを環境に自動適応させアプリケーションを運用する環境適応ソフトウェアの要素として，IoT サービスにも自動分割方式を適用することで，ユーザが独自カスタマイズを容易化できるようにする．

提案方式では，プログラムを動作させない静的状態と動作させる動的状態の両分析から，関数が含まれるファイル同士の呼び出し関係を把握し，呼び出し関係が無く関連が無いファイル群同士の分割及び通信があるファイル群同士の分割を自動提案する．提案方式により，分割ファイル群の行数が，元のファイル群の行数に比べて小さくなり，ユーザが行いたい独自処理を

追加する際の変更影響確認範囲が小さくなる．サンプルアプリケーションとして，温度表示する IoT サービスに提案方式を適用し，IoT GW と IoT PF に分割され適材適所利用ができ，全体 480 行に対して新機能追加グループは 300 行となり，機能追加変更が容易になっていることがわかる．

今後は，提案したプログラム自動分割方式を，IoT サービス以外にも適用し，様々な種類でも有効性を確認する．

## 文　献


[1] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," Technical report of General Electric (GE), 2012.

[2] H. Noguchi, et al., "Autonomous Device Identification Architecture for Internet of Things," 2018 IEEE 4th World Forum on Internet of Things (WF-IoT 2018), pp.407-411, Feb. 2018.

[3] Y. Yamato, et al., "Study of Service Processing Agent for Context-Aware Service Coordination," IEEE International Conference on Service Computing (SCC 2008), pp.275-282, July 2008.

[4] M. Takemoto, et al.,"Service-composition method and its implementation in service-provision architecture for ubiquitous computing environments," IPSJ Journal, Vol.46, No.2, pp.418-433, Feb. 2005.

[5] Y. Yamato, et al., "Method of Service Template Generation on a Service Coordination Framework," 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov. 2004.

[6] Y. Yamato, et al., "Ubiquitous Service Composition Technology for Ubiquitous Network Environments," IPSJ Journal, Vol.48, No.2, pp.562-577, Feb. 2007.

[7] Y. Yamato, et al., "Development of Service Control Server for Web-Telecom Coordination Service," IEEE International Conference on Web Services (ICWS 2008), pp.600-607, Sep. 2008.

[8] Y. Yamato, et al., "Study and Evaluation of Context-Aware Service Composition and Change-over Using BPEL Engine and Semantic Web Techniques," IEEE Consumer Communications and Networking Conference (CCNC 2008), pp.863-867, Jan. 2008.

[9] H. Sunaga, et al., "Ubiquitous Life Creation through Service Composition Technologies," World Telecommunications Congress 2006 (WTC 2006), May 2006.

[10] M. Takemoto, et al., "Service Elements and Service Templates for Adaptive Service Composition in a Ubiquitous Computing Environment," The 9th Asia-Pacific Conference on Communications (APCC 2003), Vol.1, pp.335-338, Sep. 2003.

[11] H. Noguchi, et al., "Distributed Search Architecture for Object Tracking in the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, Oct. 2018.

[12] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA'14), 2014.

[13] O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, No.3, 2012.

[14] Y. Yamato, "Automatic Verification for Plural Virtual Machines Patches," The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, July 2015.

[15] Y. Yamato, et al., "Evaluation of Agile Software Development Method for Carrier Cloud Service Platform Development," IEICE Transactions on Information & Systems, Vol.E97-D, No.11, pp.2959-2962, Nov. 2014.

[16] Y. Yamato, "Automatic verification technology of software patches for user virtual environments on IaaS cloud," Journal of Cloud Computing, Springer, 2015, 4:4, DOI: 10.1186/s13677-015-0028-6, Feb. 2015.

[17] Y. Yamato, et al., "Fast Restoration Method of Virtual Resources on OpenStack," IEEE Consumer Communications and Networking Conference (CCNC 2015), pp.607-608, Jan. 2015.

[18] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, ISBN: 0131387685, 2010.

[19] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, 2010.

[20] Y. Yamato, "Study and Evaluation of Automatic GPU Offloading Method from Various Language Applications," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1971666, Sep. 2021.

[21] Y. Yamato, "Automatic Offloading Method of Loop Statements of Software to FPGA," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1916020, Apr. 2021.

[22] Y. Yamato, "Improvement Proposal of Automatic GPU Offloading Technology," The 8th International Conference on Information and Education Technology (ICIET 2020), pp.242-246, Mar. 2020.

[23] Y. Yamato, "Proposal of Automatic Offloading for Function Blocks of Applications," The 8th IIAE International Conference on Industrial Application Engineering 2020 (ICIAE 2020), pp.4-11, Mar. 2020.

[24] Y. Yamato, "Study and Evaluation of Improved Automatic GPU Offloading Method," International Journal of Parallel, Emergent and Distributed Systems, Taylor and Francis, DOI: 10.1080/17445760.2021.1941010, June 2021.

[25] J. Fung and M. Steve, "Computer vision signal processing on graphics processing units," 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 5, pp.93-96, 2004.

[26] T. Sterling, et al., "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.

[27] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, 2012.

[28] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, 2010.

[29] J. Chen, et al., "Automatic offloading C++ expression templates to CUDA enabled GPUs," 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pp.2359-2368, May 2012.

[30] C. Bertolli, et al., "Integrating GPU support for OpenMP offloading directives into Clang," ACM Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM'15), Nov. 2015.

[31] S. Lee, et al., "OpenMP to GPGPU: a compiler framework for automatic translation and optimization," 14th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP'09), 2009.

[32] Cheng Liu, et al., "Automatic nested loop acceleration on fpgas using soft CGRA overlay," Second International Workshop on FPGAs for Software Programmers (FSP 2015), 2015.

[33] L. Sommer, et al., "OpenMP device offloading to FPGA accelerators," 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors

(ASAP 2017), pp.201-205, July 2017.
[34] A. Putnam, et al., "CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures," IEEE 2008 International Conference on Field Programmable Logic and Applications (FPL 2008), pp.173-178, Sep. 2008.
[35] L. Sun, et al., "An open IoT framework based on microservices architecture," China Communications, Vol.14, No.2, pp.154-162, 2017.
[36] B. W. Wirtz, et al., "An integrative public IoT framework for smart government," Government Information Quarterly, Vol.36, No.2, pp.333-345, 2019.
[37] S. M. Kim, et al., "IoT home gateway for auto-configuration and management of MQTT devices," In 2015 IEEE Conference on Wireless Sensors (ICWiSe), pp.12-17, Aug. 2015.
[38] J. M. Gutierrez-Guerrero and J.A. Holgado-Terriza, "Automatic configuration of OPC UA for Industrial Internet of Things environments," Electronics, Vol.8, No.6, pp.600, 2019.
[39] M. Kamimura, et al., "Extracting Candidates of Microservices from Monolithic Application Code," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp.571-580, 2018.