アプリケーションループ文のFPGA自動オフロード手法の検討

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.wa@hco.ntt.co.jp

あらまし　近年，GPUやFPGA等のヘテロなハードウェアの利用が増えており，IoTデバイスも増えている．しかし，ヘテロなハードウェアの活用には，技術的ハードルが高い．そこで，私は，アプリケーションを一度書けば，配置される場所の環境に応じて，GPUやFPGA，IoTデバイスを扱えるようにコードが変換，自動設定され，高い性能で運用できる，環境適応ソフトウェアを提案しており，GPUへの自動オフロードについては一部実現してきた．本稿では，GPUの次に市場が大きいFPGAへの自動オフロードの第一ステップとして，アプリケーションソフトウェアの適切なループ文の自動抽出手法について検討する．
キーワード　環境適応ソフトウェア, FPGA，自動オフロード，性能

Study of Automatic FPGA Offloading for Loop Statements of Applications

Yoji YAMATO†

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

Abstract　Recently, heterogeneous hardware such as GPU and FPGA is used in many systems and also IoT devices are increased repidly. However, to utilize heterogeneous hardware, the hurdles are high because of much technical skills. I have proposed environment adaptive software to operate an once written application with high performance by automatically converting the code and configuring setting so that we can utilize GPU, FPGA and IoT devices in the location to be deployed and I have also achieved automatic GPU offloading partly. In this paper, I study a method of FPGA offloading which automaticaly extracts appropriate loop statements of application software.
Key words　Environment Adaptive Software, FPGA, Automatic Offloading, Performance.

1. はじめに

近年，ムーアの法則が終焉し，CPUが1.5年で2倍の密度になることが期待できなくなるのではと言われている．その状況も踏まえて，CPUだけでなく，GPU（Graphics Processing Unit）やFPGA（Field Programmable Gate Array）といったヘテロジニアスなハードウェアをシステムに用いることが増えている．例えば，Amazonは，GPU，FPGAのインスタンス[1]をクラウド技術（例えば，[2]-[14]）を使って提供しており，MicrosoftはFPGAを使ってData Centerの効率を高めている[15].

しかし，ヘテロなハードウェアをアプリケーションで適切に活用するためには，ハードウェアに合わせたプログラミングや設定が必要で，CUDA（Compute Unified Device Architecture）[16], OpenCL（Open Computing Language）[17]といった技術知識が求められ，多くのプログラマーにとってハードルは高い．

IoT（Internet of Things）技術（例えば，[18]-[21]）の進展に伴い，IoTデバイスは増加を続けており，2020年には数百億，2020年代後半には兆に達するデバイスがネットワークに繋がると言われている．IoTの応用分野は，製造，流通，医療，農業等多岐に渡り，サービス連携技術等[22]-[31]を用いて，製品の製造過程を可視化するといった，多彩なアプリケーションが出ている．

IoTアプリケーションで，IoTデバイスのより細かい制御を行う際は，組込みソフトウェアの知識やアセンブリ等が必要になる．ゲートウェイ（GW）で，複数のIoTデバイスを集約して制御することも数多くされているが，ゲートウェイのリソースは限られるため，利用する環境に応じた設計が必要である．

整理すると，GPUやFPGA等のヘテロなハードウェア，多くのIoTデバイスを活用したアプリケーションの期待は高まっているが，それらの活用にはハードルが高いのが現状である．そこで，そのような現状を打破するため，ヘテロなハードウェアや，IoTデバイスを容易に活用できるようにするため，アプ

リケーションプログラマーは処理したいロジックだけを書き，使う先の環境（ヘテロハードウェアや利用する IoT デバイス）に合わせて，ソフトウェアが適応して，環境にあった動作をすることが，将来的に求められると考える．

1995 年に登場した Java [32] は，一度書いたソフトウェアを，別のマシンでも動作できるようにする，環境適応のパラダイムシフトを起こした．しかし，移行した先での性能については，特に考慮がされていなかった．私は，一度書いたソフトウェアを，配置先環境で GPU や FPGA，IoT デバイス等を活用できるように，コード変換等を自動で行う事で，高性能でアプリケーションを動作させることを目標にした，環境適応ソフトウェアを提案しており，更に，アプリケーションソフトウェアの GPU 自動オフロードを一部実現している．本稿では，GPU の次に市場が大きい FPGA への自動オフロードの第一ステップとして，アプリケーションソフトウェアの適切なループ文の自動抽出手法について検討する．検討手法での実際のアプリケーションソフトウェアでの評価は別稿で報告する．

2. 既存関連技術

環境適応ソフトウェアという点では，Java が挙げられる．Java は，Java Virtual Machine という仮想実行環境を用いて，一度書いたソフトは，再度のコンパイル不要で，異なる OS のマシンでも動作可能にしている（Write Once, Run Anywhere）．しかし，移植した先で，期待する性能が出るかは考慮がされておらず，性能チューニングや，移植した先でのデバッグ等の稼働は小さくないことが課題であった（Write Once, Debug Everywhere）．

GPU の計算能力を画像処理以外にも使う GPGPU（General Purpose GPU）（例えば [33]）のための開発環境 CUDA が発展している．CUDA は GPGPU 向けの開発環境だが，GPU, FPGA, メニーコア CPU 等のヘテロハードウェアを統一的に扱うための標準規格として OpenCL やその SDK [34] [35] も登場している．CUDA や OpenCL では，C 言語の拡張によるプログラミングを行うが，GPU 等のデバイスと CPU の間のメモリコピー，解放等を記述する必要があり，記述の難度は高い．

簡易にヘテロハードウェアを利用するため，ディレクティブベースで，並列処理すべき個所を指定し，ディレクティブに従いコンパイラがデバイス向けコードに変換する技術が有る．技術仕様として，OpenACC [36] 等，コンパイラとして PGI コンパイラ [37] 等がある．OpenACC は C/C++向けであるが，IBM の Java JDK [38] は，ラムダ記述に従い GPU オフロード処理が可能である．

このように，CUDA, OpenCL，OpenACC 等の技術により，GPU や FPGA へのオフロードが可能になっている．しかし，ヘテロハードウェア処理は行えるようになっても，高速化には課題が多い．マルチコア CPU 向けに，例えば，Intel コンパイラ [39] 等の自動並列化機能を持つコンパイラがある．自動並列化時は，プログラム上の for 文等の並列可能部を抽出するが，GPU や FPGA を用いる場合は，CPU-GPU や FPGA メモリ間のデータ転送オーバヘッドのため性能が出ないことも多い．GPU や FPGA を用いて高速化する際は，スキル者が，CUDA や OpenCL でのチューニングや，PGI コンパイラ等で適切な並列処理部を探索することが必要になっている．

このため，スキルが無いユーザが GPU や FPGA を使ってアプリケーションを高性能化することは難しいし，自動並列化技術等を使う場合も並列処理可否の試行錯誤や高速化できない場合があった．

IoT デバイスに関しては，計算リソース等が限られている IoT デバイスでは，細かい制御を行う際は，アセンブリ等の組み込みソフトウェアの知識が必要になるのが現状である．Rasberry Pi 等のシングルボードコンピュータでは，リソース量は限られるものの，Linux や Java 等が動作するため，Rasberry Pi を GW として複数の IoT デバイスからデータを収集したり制御したりする等の自由度が開発者に出てくる．しかし，IoT デバイスを何台収容するかや，IoT デバイスとシングルボードコンピュータでどのように処理を分担するか等は，アプリケーション，利用形態によって異なり，環境に合わせた設計が必要である．

3. ループ文の FPGA 自動オフロード手法の検討

3.1 環境適応ソフトウェアの処理フロー

環境適応を実現するため，以下の処理フローを提案している（図 1 参照）．環境適応ソフトウェアは，環境適応機能，テストケース DB，コードパターン DB，設備リソース DB，検証環境，商用環境からなる機能が連携して実現される．

Step1 コード分析：

ユーザは動作させたいアプリケーションコードと利用を想定したテストケース，要望する性能とコストを，環境適応機能に指定する．環境適応機能は，アプリケーションのコードを分析する．分析では，ループ文や変数の参照関係，処理する機能ブロック（FFT 処理）等，コードの構造を把握する．

Step2 オフロード可能部抽出：

環境適応機能は，アプリケーションコードの並列処理可能なループ文や FFT 処理の機能ブロック等，オフロード可能な処理をコードパターン DB を参照して特定し，オフロード先に応じた中間言語（OpenCL 等）を抽出する．なお，中間言語抽出は一度で終わりでなく，適切なオフロード領域探索のため，実行試行して最適化するため反復がされる．

Step3 適切なオフロード部探索：

次に，環境適応機能は，検証用環境として，GPU や FPGA，IoT デバイス用 GW 等を備えた検証用環境に，中間言語から導かれる実行ファイルをデプロイする．配置したファイルを起動し，想定するテストケースを実行して，オフロードした際の性能を測定する．ここで，GPU や FPGA 等オフロード先に応じて，この性能測定結果を用いて，より適切なオフロードとするため，Step2 の中間言語抽出のステップに戻り，別パターンの抽出を行い，性能測定を試行する．検証環境での性能測定を繰り返し，最終的にデプロイするコードパターンを決定する．

Step4 リソース量調整：

Step3 でコードパターンを決定後は，環境適応機能は，適切なリソース量の設定を行う．Step3 の検証環境性能測定で取得される，想定するテストケースの処理時間の中で，CPU 処理時間と GPU 等の CPU 以外ハードウェアの処理時間を分析し，適切なリソース比（CPU と GPU 等ハードウェアの確保するリソースの比．例：vCPU コア 4：仮想 GPU1 が適切等）を定める．次に，ユーザの要望する性能，コストと適切なリソース比を鑑みて，実際に確保するリソース量を定める（例：vCPU コア 8，仮想 GPU2 が，性能，コストを満たす量）．

Step5 配置場所調整：

Step3 で定めたコードパターンの実行ファイルを，Step4 で定めたリソース量を確保して配置する際に，環境適応機能は，性能，コストが適切になる場所を計算し配置先を決める．実行するアプリケーションの想定するテストケースの特性，設備リソース DB の情報から，性能とコストが適切になる配置場所を計算する．例えば，IoT カメラの画像情報を分析して不審者を見つける処理を，0.5sec 以内の遅延で行いたいような場合は，IoT カメラに近いエッジサーバを特定して，配置する．ここで，配置したい場所には，リソース量制限から，必要なリソースを確保できない場合等は，リソース量や場所を再調整するため，Step4 に処理を戻す場合がある．

Step6 実行ファイル配置と動作検証：

Step5 で定めた商用環境配置場所に，Step3 で定めたコードパターンの実行ファイルを，Step4 で定めたリソース量を確保して配置すると，期待通りの動作となるかを，環境適応機能は，動作検証試験を行う．試験は自動試験技術（[40] [41] 等）を活用する．ユーザが指定した想定テストケースや，テストケース DB に保持されているアプリケーションリグレッションテストケースを用いて，動作検証する．この際に，想定テストケースの商用環境での実際の性能を，確保した全リソースのスペックやコストも含めて，ユーザに提示し，ユーザにサービス開始判断をもらい，OK の場合にアプリケーションの運用を開始する．

Step7 運用中再構成：

Step6 で開始したアプリケーション運用にて，リクエスト特性変化等で当初期待していた性能が出ない場合に，環境適応機能は，ソフトウェア設定，ソフトウェア/ハードウェア構成を再構成する．ソフトウェア設定とは，リソース量や配置場所の再変更を意味しており，例えば，CPU と GPU の処理時間バランスが悪い場合に，リソースの比を変更したり，リクエスト量が増え応答時間が劣化してきた場合に，リソースの比はキープして量を増やす．あるいは，配置する場所を別のクラウドに変えるなどである．ソフトウェア/ハードウェア構成とは，コード変換から行い，GPU であればオフロード処理するロジックを変更したり，FPGA のようにハードウェアロジックを運用中に変更できる場合はハードウェアロジックを再構成することを意味している．例えば，後者で，SQL DB と No SQL の DB を両運用している場合に，元々 SQL リクエストが多かったが，NoSQL リクエストが一定よりも増えてきた場合に，NoSQL をアクセラレートする FPGA にロジックを再構成する．

ここで，Step1-7 で，環境適応に必要な，コード変換，リソース量調整，配置場所調整，運用中再構成を一括して行う処理フローを説明したが，行いたい処理だけ切出すことも可能である．例えば，FPGA 向けにコード変換だけ行いたい場合は，Step1-3 だけ行い，環境適応機能や検証環境等必要な部分だけ利用すれば良い．

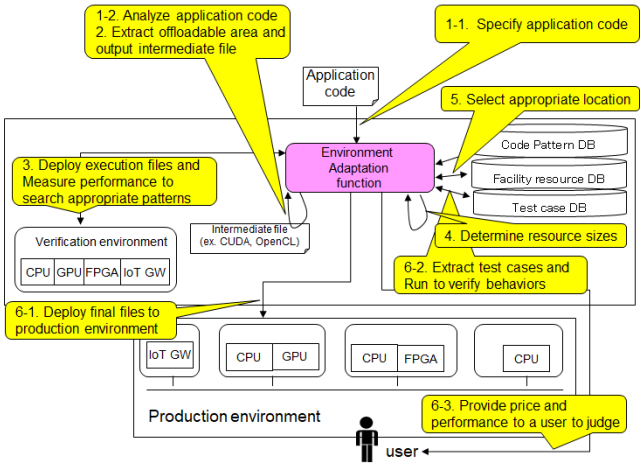

図 1　環境適応ソフトウェア処理フロー

3.2　FPGA 自動オフロード手法検討

Step1 のコード分析については，Clang [42] 等の構文解析ツールを用いて，アプリケーションコードの分析を行う．コード分析は，オフロードするデバイスを想定した分析が必要になるため，一般化は難しいが，ループ文や変数の参照関係等のコードの構造を把握したり，機能ブロックとして FFT 処理を行う機能ブロックであることや，FFT 処理を行うライブラリを呼び出している等を把握する．機能ブロックの判断は，Machine が自動判断する事は難しいが，CCFinderX [43] 等の類似コード検出ツールを用いて類似度判定等で把握する．また，Clang は C/C++向けツールであるが，解析する言語に合わせたツールを選ぶ必要がある．

Step2-3 は処理をオフロードする，GPU，FPGA，IoT デバイス制御用 GW 等，オフロード先に合わせた検討が必要となる．一般に，性能に関しては，最大性能になる設定を一回で自動発見するのは難しいため，オフロードパターンを何度か性能測定を検証環境で繰り返し試行し，高速化できるパターンを見つけることを行う．ここでは，アプリケーションソフトウェアのループ文の FPGA 向けオフロード手法について説明する．

アプリケーションは，多種多様だが，映像分析のための画像処理（[44] [45] 等），センサデータ分析のための機械学習等，計算量が多いアプリケーションでは，繰り返しが多い．そこで，アプリケーションのループ文を FPGA に自動でオフロードする事での高速化を行う．

しかし，2 節で記載の通り，高速化には適切な並列処理やパイプライン処理が必要である．特に，FPGA を使う場合は，CPU と FPGA 間のメモリ転送のため，データサイズやループ回数が多くないと性能が出ないことが多い．また，メモリプロセスの持ち方やメモリデータ転送のタイミング等により，並列，パイプラインで高速化できる個々のループ文の組み合わせが，最

速とならない場合等がある（例：10 個の for 文で，1 番，5 番，10 番の 3 つが CPU に比べて高速化出来る場合に，1 番，5 番，10 番の 3 つの組み合わせが最速になるとは限らない等）．

GPU の場合は，OpenACC を用いて#pragma acc kernels のディレクティブ記述で，指定したループ文の GPU 実行をしたり，CUDA でより細かい指定が出来た．FPGA では，CUDA に近い OpenCL を用いた指定や，より抽象的な高位合成ツールを用いた指定が可能である．OpenCL では，以下の 10 ステップの記載が必要である．デバイスの準備，カーネルの準備〔オンラインコンパイル〕，デバイスメモリの割り当て，ホストからデバイス方向のデータ転送，カーネル関数の引数設定，カーネル関数の実行，デバイスから方向のデータ転送，デバイスメモリの解放，カーネルの解放，その他オブジェクト（デバイス）の解放．高位合成ツールは提供ベンダにより仕様は大きく異なるが，例えば，Xilinx 社の vivado HLS では，OpenACC と類似の#pragma HLS PIPELINE や#pragma HLS UNROLL 等のディレクティブで FPGA 処理を指定できる．

ここで，GPU では，ループ文をオフロードする際は，並列処理できるかが厳密にチェックされ，ループに依存関係がある場合は，コンパイルエラーが出ていた．一方，FPGA では，並列で出来ない際も，FPGA 実行環境側でパイプライン処理として実行することで高速化することも期待できるため，並列処理とパイプライン処理を組み合わせることで，実際にオフロード出来るループ数が GPU に比べて増える可能性が高い．

著者の以前の研究である [46] は，CPU 向け汎用プログラムから，GPU に向けて，自動で適切なオフロード領域を抽出するため，並列可能ループ文群に対して遺伝的アルゴリズム（GA）を用いて検証環境で性能検証試行を反復し適切な領域を探索する．遺伝子の部分の形で，処理パターンを保持し組み換えていくことで，取り得る膨大なパターンから，効率的に高速化可能なパターン探索を狙っている．GA は，生物の進化過程を模倣した組合せ最適化手法の一つで，GA のフローチャートは，初期化→評価→選択→交叉→突然変異→終了判定である [47]．

なお，FPGA ではプログラムサイズやコンパイル環境によっては，コンパイルして FPGA 実機で動作させるのに長時間がかかる場合がある．そのため，エミュレータ環境を用いた性能簡易試験なども併用を検討する．

また，[46] では，オフロードに適切なループ文探索はできるが，ループ毎に CPU と GPU のデータ転送が発生する場合等効率的でない場合や，オフロードしても高速にならない場合もある等，高速化できるアプリケーションが限られていた．そのため，非効率なデータ転送を低減するため，明示的指示を用いたデータ転送指定を，GA での並列処理の抽出と合わせて行う事を更に行う．提案方式は，GA で生成された各個体について，ループ文の中で利用される変数データの参照関係を分析し，ループ毎に毎回データ転送するのでなくループ外でデータ転送して良いデータについては，ループ外でのデータ転送を指定する．

データ転送の種類は，CPU から FPGA へのデータ転送，及び，FPGA から CPU へのデータ転送がある．

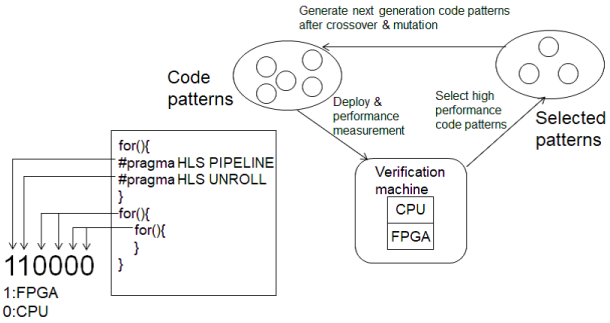

図 2　ループ文 FPGA 自動オフロード手法（高位合成ツール利用時）

・CPU から FPGA へのデータ転送．

CPU プログラム側で定義した変数と FPGA プログラム側で参照する変数が重なる場合は，CPU から FPGA へのデータ転送が必要として，データ転送指定を行う．データ転送を指定する位置は，FPGA 処理するループ文かそれより上位のループ文で，該当変数の設定，定義を含まない最上位のループとする．データ転送指示行の挿入位置は，for, do, while 等のループの直前に行う．

・FPGA から CPU へのデータ転送．

FPGA プログラム側で設定した変数と CPU プログラム側で参照する変数とが重なる場合は，FPGA から CPU へのデータ転送が必要として，データ転送指定を行う．データ転送を指定する位置は，FPGA 処理するループ文か，それより上位のループ文で，該当変数の参照，設定，定義を含まない最上位のループとする．データ転送指示行の挿入位置は，for, do, while 等のループの直前に行う．

このように，出来るだけ上位のループでデータ転送を一括して行うように，データ転送を明示的に指示することで，ループ毎に毎回データを転送する非効率な転送を避けることができる．

更に，gcov [48]，gprof [49] 等のプロファイリングツールを用いて，ループ回数を事前にチェックし，FPGA オフロードを試行するかどうかの振り分けをしても良い．

まとめると，FPGA 向けループ文オフロードは，GA 等を用いた適切なオフロード部の探索（図 2 参照）と，変数参照関係を用いたデータの一括転送により，自動での高速化を狙う．

4. ま と め

本稿では，提案のアプリケーションを環境に適応させ GPU や FPGA 等を適切に活用し高性能にアプリケーションを動作させるための環境適応ソフトウェアの新要素技術として，ソフトウェアループ文の FPGA 自動オフロード手法を検討した．

検討した FPGA 自動オフロードの手法は，ソフトウェアを分析してループ文を検知し，適切なループ文抽出のため，ループ文のオフロードパターンを遺伝子にマッピングし，遺伝アルゴリズムを用いて，高速なオフロードパターンを，検証環境での試験を通じて抽出する．GPU と近い方法での抽出を検討するが，FPGA はパイプライン処理，並列処理等より細かいチューニング性を確認する．また，FPGA 実機へのコンパイルは長時

間かかることが多いため，検証途中段階ではエミュレータの利用等も検討する．

今後は，検討手法での FPGA 自動オフロードをオープンソースソフトウェアを用いて検証し，別稿で報告する．

文　献

[1] AWS web, https://aws.amazon.com/ec2/instance-types/

[2] O. Sefraoui, et al., "OpenStack: toward an open-source solution for cloud computing," International Journal of Computer Applications, Vol.55, 2012.

[3] Y. Yamato, et al., "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Transactions on Cloud Computing, Sep. 2015.

[4] Y. Yamato, et al., "Software Maintenance Evaluation of Agile Software Development Method Based on OpenStack," IEICE Transactions on Information & Systems, Vol.E98-D, No.7, pp.1377-1380, July 2015.

[5] Y. Yamato, "Automatic verification technology of software patches for user virtual environments on IaaS cloud," Journal of Cloud Computing, Springer, 2015, 4:4, Feb. 2015.

[6] Y. Yamato, "Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," Journal of Information Processing, Vol.25, No.1, pp.56-58, Jan. 2017.

[7] Y. Yamato, "Performance-Aware Server Architecture Recommendation and Automatic Performance Verification Technology on IaaS Cloud," Service Oriented Computing and Applications, Springer, Nov. 2016.

[8] Y. Yamato, "Cloud Storage Application Area of HDD-SSD Hybrid Storage, Distributed Storage and HDD Storage," IEEJ Transactions on Electrical and Electronic Engineering, Vol.11, pp.674-675, 2016.

[9] Y. Yamato, "Use case study of HDD-SSD hybrid storage, distributed storage and HDD storage on OpenStack," 19th International Database Engineering & Applications Symposium (IDEAS15), pp.228-229, 2015.

[10] Y. Yamato, "Server Selection, Configuration and Reconfiguration Technology for IaaS Cloud with Multiple Server Types," Journal of Network and Systems Management, Springer, Aug. 2017.

[11] 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.

[12] Y. Yamato, et al., "Development of Resource Management Server for Production IaaS Services Based on OpenStack," Journal of Information Processing, Vol.23, pp.58-66, 2015.

[13] Y. Yamato, "Server Structure Proposal and Automatic Verification Technology on IaaS Cloud of Plural Type Servers," International Conference on Internet Studies (NETs2015), July 2015.

[14] Y. Yamato, "OpenStack Hypervisor, Container and Baremetal Servers Performance Comparison," IEICE Communication Express, Vol.4, No.7, pp.228-232, July 2015.

[15] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," ISCA'14, pp.13-24, 2014.

[16] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011

[17] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.

[18] M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[19] Y. Yamato, et al., "Predictive Maintenance Platform with Sound Stream Analysis in Edges," Journal of Information Processing, Vol.25, pp.317-320, 2017.

[20] Tron project web site, http://www.tron.org/

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

[22] Y. Yamato, et al., "Context-aware Ubiquitous Service Composition Technology," The IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006), pp.51-61, Apr. 2006.

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

[24] Y. Nakano, et al., "Method of creating web services from web applications," IEEE SOCA 2007, pp.65-71, June 2007.

[25] H. Sunaga, et al., "Ubiquitous Life Creation through Service Composition Technologies," World Telecommunications Congress 2006 (WTC2006), May 2006.

[26] H. Sunaga, et al., "Service Delivery Platform Architecture for the Next-Generation Network," ICIN2008, 2008.

[27] Y. Yamato, et al., "Development of Service Control Server for Web-Telecom Coordination Service," IEEE ICWS 2008, pp.600-607, Sep. 2008.

[28] Y. Yokohata, et al., "Service Composition Architecture for Programmability and Flexibility in Ubiquitous Communication Networks," IEEE International Symposium on Applications and the Internet Workshops (SAINTW'06), 2006.

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

[30] Y. Yamato and H. Sunaga, "Context-Aware Service Composition and Component Change-over using Semantic Web Techniques," IEEE ICWS 2007, pp.687-694, July 2007.

[31] Y. Yokohata, et al., "Context-Aware Content-Provision Service for Shopping Malls Based on Ubiquitous Service-Oriented Network Framework and Authentication and Access Control Agent Framework," IEEE CCNC 2006, pp.1330-1331, 2006.

[32] J. Gosling, et al., "The Java language specification, third edition," Addison-Wesley, 2005. ISBN 0-321-24678-0.

[33] K. Shirahata, et al., "Hybrid Map Task Scheduling for GPU-Based Heterogeneous Clusters,"IEEE CloudCom, 2010.

[34] Altera SDK web site, https://www.altera.com/products/design-software/embedded-software-developers/opencl/documentation.html

[35] Xilinx SDK web site, https://japan.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/lyx1504034296578.html

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

[37] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.

[38] K. Ishizaki, "Transparent GPU exploitation for Java," CANDAR 2016, Nov. 2016.

[39] E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.

[40] Jenkins web site, https://jenkins.io/

[41] Selenium web site, https://www.seleniumhq.org/

[42] Clang website, http://llvm.org/

[43] CCFinder web site, http://www.ccfinder.net/

[44] OpenCV web site, http://opencv.org/

[45] imageJ web site, https://imagej.nih.gov/ij/docs/concepts.html

[46] Y. Yamato, et al., "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, Sep. 2018.

[47] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.

[48] gcov website, http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[49] gprof website, http://sourceware.org/binutils/docs-2.20/gprof/