

Jxiv

[ジェイカイク / dʒéikaiv]

Title	
Author(s)	
Citation	<p>Journal title (Repository name etc.), Volume, Issue, Pages (Article number) etc. ・ ジャーナル名 (刊行物・サイト名) ・ 巻号 ・ ページ (その他論文番号等) :</p> <p>・ DOI (URL)</p> <p>Publication Date: yyyy/mm/dd ・ 出版日 : 年 月 日</p> <p>Publisher ・ 出版者 :</p>
Declaration	<p>This preprint is the _____ of the above. ・ 本プレプリントは、上記論文の _____ である。</p> <p>All necessary permissions from the publisher have ・ ジャーナル (出版者) から必要な許諾を been obtained not been obtained 得ている 得ていない</p>
Notes	

ヘテロデバイス自動オフロード時の電力使用量削減評価

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, 少コア CPU だけでなく, GPU, FPGA, メニーコア CPU 等のヘテロなデバイスを利用したシステムが増えている. しかし, これらの利用には, CUDA 等のハードウェアを意識した技術仕様の理解が必要であり, ハードルは高い. これらの背景から, 私は, プログラマーが CPU 向けに開発したソースコードを, 適用される環境に応じて, 自動で変換し, リソース量等を設定して, 高性能, 省電力で運用可能とする環境適応ソフトウェアのコンセプトを提案しており, 今までに, GPU, FPGA への自動オフロードでの性能向上を検証してきた. 本稿では, 今まで未検証だった, 自動オフロード時の電力使用量を確認し, 環境適応により低電力で運用可能であることを検証する. 既存アプリケーションの自動オフロード時の Watt と処理時間 Sec の積を, CPU のみで処理する場合と比較する.

キーワード 環境適応ソフトウェア, GPGPU, 自動オフロード, 省電力, 進化的計算

Power Reduction of Automatic Heterogeneous Device Offloading

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 In recent years, utilization of heterogeneous hardware other than small core CPU such as GPU, FPGA or many core CPU is increasing. However, when using heterogeneous hardware, barriers of technical skills such as CUDA are high. Based on that, I have proposed environment-adaptive software that enables automatic conversion, configuration, and high performance and low power operation of once written code, according to the hardware to be placed. I also have verified performance improvement of automatic GPU and FPGA offloading so far. In this paper, I verify low power operation with environment adaptation by confirming power utilization after automatic offloading. I compare Watt*seconds of existing applications after automatic offloading with the case of CPU only processing.

Key words Environment Adaptive Software, GPGPU, Automatic Offloading, Low Power, Evolutionary Computation.

1. はじめに

近年, CPU の半導体集積度が 1.5 年で 2 倍になるというムーアの法則が減速するのではないかとされている. そのような状況から, CPU だけでなく, GPU (Graphics Processing Unit) や FPGA (Field Programmable Gate Array) 等のデバイスの活用が増えている. 例えば, Microsoft 社は FPGA を使って Bing の検索効率を高めるといった取り組みをしており [1], Amazon 社は, FPGA, GPU 等をクラウド技術を用いて (例えば, [2]-[9]) インスタンスとして提供している [10]. また, IoT 機器利用のシステム (例えば, [11]-[18]) 等も増えている.

しかし, 少コアの CPU 以外のデバイスをシステムで適切に活用するためには, デバイス特性を意識した設定やプログラム

作成が必要であり, OpenMP (Open Multi-Processing) [19], OpenCL (Open Computing Language) [20], CUDA (Compute Unified Device Architecture) [21] といった知識や IoT 機器向けの組み込みシステムの知識が必要になってくるため, 大半のプログラマーにとっては, スキルの壁が高い.

GPU や FPGA, メニーコア CPU 等を活用するシステムは増えていくと予想されるが, それらを活用するには, 技術的壁がある. そこで, そのような壁を取り払い, 少コアの CPU 以外のデバイスを十分利用できるようなするため, プログラマーが処理ロジックを記述したソフトウェアを, 配置先の環境 (GPU, FPGA, メニーコア CPU 等) にあわせて, 適応的に変換, 設定し, 環境に適合した動作をさせるような, プラットフォームが求められている.

1995年に登場したJava [22] は一度記述したコードを、別メーカーのCPUでも動作可能にし、環境適応のパラダイムシフトを起こした。しかし、移行先での性能や電力使用量は、適切であるとは限らなかった。そこで、私は、一度記述したコードを、配置先の環境に存在するGPUやFPGA、メニーコアCPU等を利用できるように、変換、リソース設定等を自動で行い、アプリケーションを高性能かつ低電力で動作させることを目的とした、環境適応ソフトウェアを提案している。あわせて、環境適応ソフトウェアの要素として、アプリケーションコードのループ文や機能ブロックを、GPU、FPGAに自動オフロードする方式を提案し性能向上を評価している [23] [24] [25]。しかし、今までは、自動オフロード時の処理時間のみ評価しており、電力使用量については評価されていなかった。

本稿は、通常のCPU向けプログラムを、GPU等のデバイスにオフロードし高性能化させる際に、電力使用量も考慮に入れる方式を提案し、電力使用量低減を、既存アプリケーションのオフロードを通じて確認する。合わせて、GPU、FPGA等の混在環境の際に、オフロード時の電力使用量も踏まえて、適切なオフロード先を自動選択する手法も提案する。

本稿の構成は以下の通りである。2節で、既存技術を概説する。3節で、電力使用量を考慮したGPU、FPGAへの自動オフロード手法、混在環境でのオフロード先選択手法を提案する。4節で、既存アプリケーションオフロードを通じて、GPU自動オフロード時の電力使用量を評価する。5節でまとめを行う。

2. 既存技術

2.1 市中技術

環境適応ソフトウェアとしては、Javaがある。Javaは、仮想実行環境であるJava Virtual Machineにより、一度記述したJavaコードを再度のコンパイル不要で、異なるメーカー、異なるOSのCPUマシンで動作させている (Write Once, Run Anywhere)。しかし、移行先で性能が十分かはわからず、移行先でのデバッグや性能に関する稼働が大きかった (Write Once, Debug Everywhere)。

GPUを画像処理以外にも使うGPGPU (General Purpose GPU) (例えば [26]) を行うための環境にCUDAがある。また、GPUだけでなく、FPGA、メニーコアCPU等のヘテロなデバイスを同様に扱うための仕様としてOpenCLがある。CUDA、OpenCLは、C言語の拡張を行いプログラムを行う形だが、プログラムの難度は高い。

CUDAやOpenCLに比べて、より簡易にヘテロデバイスを利用するため、指示行ベースで、並列処理等を行う箇所を指定して、指示行に従ってコンパイラが、GPU等に向けて実行ファイルを作成する技術がある。仕様としては、OpenACC [27] やOpenMP等、コンパイラとしてPGIコンパイラ [28] やgcc等がある。

CUDA、OpenCL、OpenACC、OpenMP等の技術を用いることで、GPUやFPGA、メニーコアCPUへオフロードすることは可能になってきている。しかしオフロード自体は行えるようになって、高速化や省電力化には課題がある。例えば、

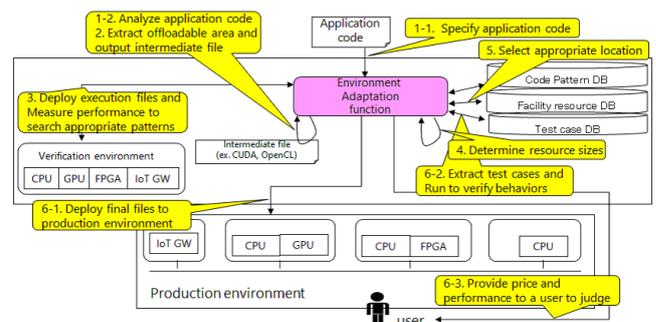


図1 環境適応ソフトウェアのフロー

マルチコアCPU向けに自動並列化機能を持つコンパイラとして、Intelコンパイラ [29] 等がある。これらは、ループ文中で並列処理可能な部分を抽出して、並列化している。しかし、メモリ処理等の影響で単に並列化しても性能がでないことも多い。GPUやFPGA等で高速化するには、OpenCLやCUDAの技術者がチューニングを繰り返している。

このため、スキルが乏しい技術者が、GPUやFPGA、メニーコアCPUを活用してアプリケーションを高速化、省電力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箇所探索の試行錯誤等の稼働が必要だった。

2.2 環境適応処理のフロー

ソフトウェアの環境適応を実現するため、著者は図1の処理フローを提案している [31]。環境適応ソフトウェアは、環境適応機能を中心に、検証環境、商用環境、テストケースDB、コードパターンDB、設備リソースDBの機能群が連携することで動作する。

- Step1 コード分析：
- Step2 オフロード可能部抽出：
- Step3 適切なオフロード部探索：
- Step4 リソース量調整：
- Step5 配置場所調整：
- Step6 実行ファイル配置と動作検証：
- Step7 運用中再構成：

現状を整理する。ヘテロなデバイスに対するオフロードは手動での取組みが主流である。著者は以前に環境適応ソフトウェアのコンセプトを提案し、自動オフロードを検討している。GPU等に処理をオフロードする際に、並列処理箇所探索を自動化するため、進化計算を用いた手法を提案しているが、処理時間の短縮のみの評価であり [31]、電力使用量の削減は未評価であった。そのため、本稿では、GPU等のオフロードデバイスに自動オフロードした際の、電力使用量削減を評価する。

3. 電力を考慮したGPU、FPGAへの自動オフロード

著者は、環境適応ソフトウェアのコンセプトを具体化するために、これまでに、プログラムのループ文のGPU自動オフロード、FPGA自動オフロード、プログラムの機能ブロックの自動オフロードや、多様言語、混在環境での自動オフロードを提案

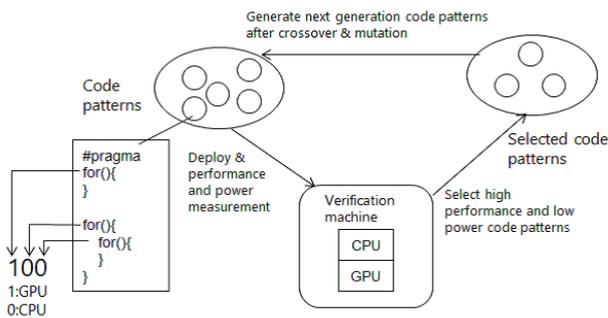


図 2 電力を考慮した GPU 自動オフロード手法

してきた。これらの要素技術検討も踏まえて、本節の、3.1, 3.2 では、電力使用量を考慮したループ文の GPU, FPGA 自動オフロード技術を提案する、3.3 では、移行先が混在環境での適切なオフロード先選択技術を提案する。

3.1 ループ文の GPU 自動オフロード

処理時間短縮を目的にしたループ文の GPU 自動オフロード手法については、[30] [31] で提案している。

まず、共通的な課題として、コンパイラがこのループ文は GPU で並列処理できないという制限を見つけることは可能だが、このループ文は GPU の並列処理に適しているという適合性を見つけることは難しいのが現状である。実際に GPU にオフロードすることでどの程度の性能、電力消費量になるかは、実測してみないと予測は難しい。そのため、このループを GPU にオフロードするという指示を手動で行い、測定を試行錯誤することが行われている。著者はそれを踏まえ、GPU にオフロードする適切なループ文の発見を、進化計算手法である遺伝的アルゴリズム (GA) [32] で自動的に行うことを以前提案した。プログラムから、最初に並列可能ループ文のチェックを行い、並列可能ループ文群に対して、GPU 実行の際を 1, CPU 実行の際を 0 と値を置いて遺伝子化し、検証環境で反復測定し適切なパターンを探索している。ここで、検証環境測定で短時間処理できるパターンを高い適合度の遺伝子とするが、検証環境測定では電力使用量も合わせて測定し、低電力なパターンを高い適合度とする処理を新たに加える。例えば、 $(\text{処理時間})^{-1/2} \times (\text{電力使用量})^{-1/2}$ のように、短処理時間、低電力使用量ほど適合度が高くなるよう設定する (図 2 参照)。

[31] では、ネストループ文中での利用変数について、ループ文を GPU 処理する際に、ネスト下位で CPU-GPU 転送が行われると下位ループ毎に転送が行われ効率的でないため、上位で CPU-GPU 転送が行われても問題ない変数については、上位で一括転送することを提案している。ネストだけでなく、複数ファイル定義の変数も、GPU 処理と CPU 処理が入れ子にならず、CPU 処理と GPU 処理が分けられる変数については、一括して転送する。

ループ文の GPU オフロードについては、電力使用量を適合度に含める進化計算手法と、CPU-GPU 転送の低減により、自動での高速化、低電力化を行う。

3.2 ループ文の FPGA 自動オフロード

著者は、ループ文の FPGA 自動オフロード手法についても、提案している [25]。

FPGA で、処理時間が長時間かかる特定ループ文を FPGA にオフロードして高速化することを考えた際に、どのループをオフロードすれば高速になるかの予測は難しいため、GPU 同様検証環境で性能測定を自動で行うことを提案している。しかし、FPGA は、OpenCL をコンパイルして実機で動作させるまでに数時間以上かかるため、GPU 自動オフロードでの GA を用いて何回も反復して性能測定することは、処理時間が膨大となり行う事はできない。

そこで、FPGA にオフロードする候補のループ文を絞ってから、性能測定試行を行う形をとる。具体的には、まず、発見されたループ文に対して、ROSE フレームワーク [33] 等の算術強度分析ツールを用いて算術強度が高いループ文を抽出する。更に、gcov や gprof 等のプロファイリングツールを用いてループ回数が多いループ文も抽出する。算術強度やループ回数が多いループ文を候補として、OpenCL 化を行う。OpenCL 化時には、CPU 処理のプログラムを、カーネル (FPGA) とホスト (CPU) に、OpenCL の文法に従って分割する。算術強度やループ回数が多いオフロード候補ループ文に対して、作成した OpenCL をプレコンパイルして、リソース効率が高いループ文を見つける。これは、コンパイルの途中で、作成する Flip Flop や Look Up Table 等のリソースは分かるため、利用するリソース量が十分少ないループ文に更に絞り込む。候補ループ文が幾つか残るため、それらを用いて性能や電力使用量を実測する。選択された単ループ文に対して FPGA 実機で動作するようコンパイルして測定し、更に高速化できた単ループ文に対してはその組み合わせのパターンも作り 2 回目の測定をする。検証環境で測定された複数パターンの中で、短時間かつ低電力のパターンを解として選択する。短時間かつ低電力は、GPU の時と同様の式でスコア付けすればよい (図 3 参照)。

ループ文の FPGA 自動オフロードについては、算術強度やループ回数、リソース量を用いて候補ループ文を絞り込んでから、検証環境での測定を行い、低電力パターンの評価値を高めることで、自動での高速化、低電力化を行う。

3.3 混在環境での自動オフロード

著者は、GPU, FPGA, メニーコア CPU が移行先として混在している中で、高性能な移行先を選択しオフロードする技術も検討している。

検証する順番として、メニーコア CPU 向けループ文オフロード、GPU 向けループ文オフロード、FPGA 向けループ文オフロードで検証し、高性能となるパターンを探索していく。自動オフロードでは、パターンの探索は、できるだけ安価で短時間に行う事が期待される。そこで、検証時間がかかる FPGA は最後とし、前の段階で十分ユーザ要件を満足するパターンが見つかっていれば、FPGA 検証は行わない事とする。GPU とメニーコア CPU に関しては、価格的にも検証時間的にも大きな差はないが、メモリも別空間となりデバイス自体が異なる GPU に比して、メニーコア CPU の方が、通常 CPU との差は

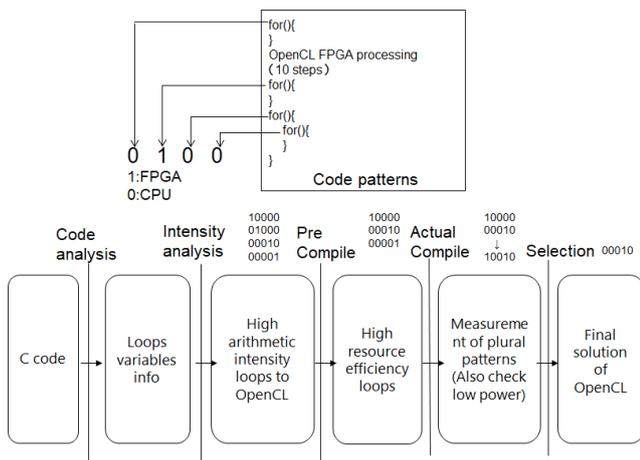


図 3 電力を考慮した FPGA 自動オフロード手法

小さいため、検証順はメニーコア CPU を先として、メニーコア CPU で十分ユーザ要件を満足するパターンが見つかったら、GPU 検証は行わない事とする。

ここで、3つの移行先を検証し、高速な移行先を自動選択するのが既存方式であるが、本稿では検証環境での実測を通じて短処理時間だけでなく低電力の移行先も、自動選択の候補となるようにする。例えば、 $(\text{処理時間})^{-1/2} \times (\text{電力使用量})^{-1/2}$ のように、短処理時間、低電力使用量なほどスコアが高くなるように評価式を設定すれば良い。

典型的データセンタのコストとして、ハードウェアや開発費等の初期費用が全体コストの 1/3、電力や保守等の運用コストが 1/3、サービスオーダ等のその他費用が 1/3 の例がある。この場合、例えば処理時間が 1/5 になり、CPU と GPU 合わせてもハードウェア台数が半減となれば初期費用も低減される。電力使用量半減も運用コスト低減につながる。ただし、運用コストは電力以外要素も多く、電力使用量半減が運用コスト半減になるわけでない。また、ハードウェア価格も、導入する GPU、FPGA サーバ数によりボリューム割引等があり、事業者毎に異なる。そのため、評価式は事業者毎に異なる設定とする必要がある。

このように、本稿では、処理時間だけでなく、電力使用量も考慮して、適切なオフロード先を自動選択する。

4. 評価

ループ文の GPU、FPGA への自動オフロードは [31] 等で既に評価しているため、本稿では、以前論文の実装ツールをベースに、測定パターンの評価値を定める際に、低電力な程評価値が高くなる様な改造を加えてオフロードを行い、オフロードによって、処理時間低減とともに、低電力化が出来ていることを確認する。

4.1 評価条件

4.1.1 評価対象

評価対象は、GPU オフロードで、流体計算の姫野ベンチマークとする。

姫野ベンチマーク [34] は、非圧縮流体解析の性能測定に用い

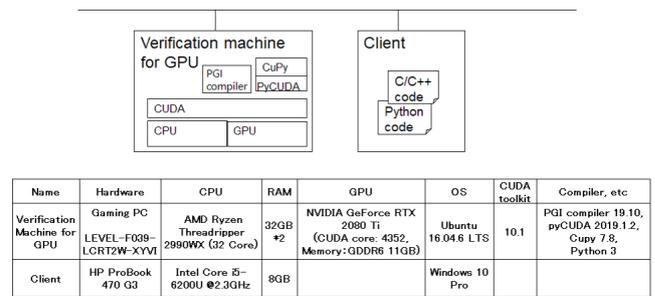


図 4 検証環境

られるベンチマークソフトで、ポアソン方程式をヤコビ反復法で解いている。GPU での手動高速化に頻繁に利用されており、自動でも高速化と低電力化ができることの確認のため評価対象とする。姫野ベンチマークは C 言語や Fortran もあるが、今回は電力測定のためある程度処理時間がかかる Python を用いることとし、処理ロジックを Python で記述した。用いるデータは Large で $512 \times 256 \times 256$ のグリッドで計算する。CPU 処理は、Python の Numpy [35] で処理され、GPU 処理は Numpy Interface で入力され GPU に処理オフロードする Cupy ライブラリ [36] を用いて処理される。

4.1.2 評価手法

対象のアプリケーションのコードを入力し、移行先の GPU に対して、Clang [37] 等の解析ライブラリで認識されたループ文のオフロードを試行してオフロードパターンを決めるが、その際に、処理時間と電力使用量を測定する。最終的に定まったオフロードパターンについて、電力使用量の時間変化を取得し、オフロードせず全て通常 CPU で処理する場合に比した電力使用量改善を確認する。

GPU での GA の条件は以下で行う。

オフロード対象ループ文数：姫野ベンチマーク 13

個体数 M：ループ文数以下とする。姫野ベンチマーク 12

世代数 T：ループ文数以下とする。姫野ベンチマーク 12

適合度： $(\text{処理時間})^{-1/2} \times (\text{電力使用量})^{-1/2}$ 処理時間と電力使用量が低い程高適合度になる。また、 $(-1/2)$ 乗とすることで、処理時間が短い特定の個体の適合度が高くなり過ぎて、探索範囲が狭くなるのを防ぐ。また、性能測定が一定時間 (3分) で終わらない際はタイムアウトさせ、処理時間 10000 秒とする。

選択：ルーレット選択。ただし、世代での最高適合度遺伝子は交叉も突然変異もせず次世代に保存するエリート保存も合わせて行う。

交叉率 P_c : 0.9

突然変異率 P_m : 0.05

4.1.3 評価環境

利用する GPU として GeForce RTX 2080 Ti (CUDA core: 4352, Memory : GDDR6 11GB) を用いる。GPU 処理は Cupy7.8 と PyCUDA2019.1.2 を用いる。電力使用量は、GPU 搭載 PC にインストールした NVIDIA ツールの nvidia-smi コマンド [38] で GPU 電力を、s-tui コマンド [39] で CPU 電力を測定する。評価環境とスペックを図 4 に示す。

Gaming PC (only CPU use)			Gaming PC (CPU and GPU use)			
Watt * Sec	4077		Watt * Sec	2071		
processing time (sec)	153		processing time (sec)	19		
time	CPU Watt		time	CPU Watt	GPU Watt	Sum Watt
15:04:07	31.2		15:03:15	28.9	27.6	56.5
15:04:08	24.2		15:03:16	31.5	27.6	59.1
15:04:09	31.0		15:03:17	29.7	27.6	57.3
15:04:10	22.9	execution period of Himeno benchmark	15:03:18	23.4	41.6	65.0
15:04:11	25.7		15:03:19	23.9	66.7	90.6
15:04:12	26.9		15:03:20	24.2	81.9	106.1
15:04:13	25.0		15:03:21	22.3	91.3	113.6
15:04:14	23.8		15:03:22	25.1	91.1	116.2

図5 GPU オフロード時電力使用量(姫野ベンチマーク)

4.2 結果と考察

図5はGPUに姫野ベンチマークをオフロードした際の、Wattと時間を記載している。図より、全てCPU処理する場合と比較して、処理時間は153秒から19秒に短縮されているが、電力はCPUのみの場合の27W程度からCPUとGPUが使われ109W程度となっていることがわかる。その結果、Watt*secは、CPUのみ処理の場合の4080Watt*secから、GPUにオフロードした場合で1/2の2070Watt*secとなった事が分かる。

IoT等で多くのユーザが利用すると想定されるアプリケーションとして、流体計算の姫野ベンチマークの高速化と低電力化を確認した。GPUにオフロードした際は、Watt自体は増えるが、短時間化の効果が高く全体としてWatt*secが減っている。FPGAオフロードの際の電力使用量は今回測定にはないが、GPU、FPGA混在環境ならば、オフロード時の性能と電力から適切なオフロード先を選択する。

5. まとめ

本稿では、私が提案している、ソフトウェアを配置先環境に合わせて自動適応させGPU、FPGA等を適切に利用して、アプリケーションを高性能、低電力で運用するための環境適応ソフトウェアの要素として、電力使用量を考慮したオフロード手法を提案し評価した。

GPU、FPGA自動オフロード時に検証環境で実測する際に、処理時間とともに電力使用量を取得し、短時間かつ低電力なパターンを高い適合度とすることで、自動でのコード変換に低電力化を盛り込む。GPU、FPGA等が混在している場合は、単体の移行先移行を試行し、短時間かつ低電力な移行先を選択するようにすることで、自動選択を行う。流体計算のGPU自動オフロードを通じて、高性能化と共に低電力化を確認し、方式の有効性確認した。

今後は、より多くのアプリケーションでの電力使用量低減を検証する。また、事業者のコスト構造の具体例を参考に、短処理時間化と低電力化の評価式を検討する。

文献

[1] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," ISCA'14, pp.13-24, 2014.
 [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 Restoration Method of Virtual Resources on OpenStack," 12th Annual IEEE Consumer Com-

munications and Networking Conference (CCNC 2015), pp.607-608, Jan. 2015.
 [4] Y. Yamato, "Automatic system test technology of virtual machine software patch on IaaS cloud," IEEJ Transactions on Electrical and Electronic Engineering, Vol.10, Issue.S1, pp.165-167, Oct. 2015.
 [5] Y. Yamato, "Proposal of Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," 2016 6th International Workshop on Computer Science and Engineering (WCSE 2016), pp.34-37, June 2016.
 [6] Y. Yamato, "Server Structure Proposal and Automatic Verification Technology on IaaS Cloud of Plural Type Servers," International Conference on Internet Studies (NETs2015), July 2015.
 [7] 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.
 [8] 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, July 2015.
 [9] Y. Yamato, et al., "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Transactions on Cloud Computing, DOI: 10.1109/TCC.2015.2481392, Sep. 2015.
 [10] AWS EC2 web site, <https://aws.amazon.com/ec2/instance-types/>
 [11] M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Technische Universitat Dortmund. 2015.
 [12] Y. Yamato, et al., "Proposal of Shoplifting Prevention Service Using Image Analysis and ERP Check," IEEJ Transactions on Electrical and Electronic Engineering, Vol.12, Issue.S1, pp.141-145, June 2017.
 [13] Y. Yamato, "Proposal of Vital Data Analysis Platform using Wearable Sensor," 5th IIAE International Conference on Industrial Application Engineering 2017 (ICIAE2017), pp.138-143, Mar. 2017.
 [14] Y. Yamato, et al., "Security Camera Movie and ERP Data Matching System to Prevent Theft," IEEE Consumer Communications and Networking Conference (CCNC 2017), pp.1021-1022, Jan. 2017.
 [15] Y. Yamato, "Experiments of posture estimation on vehicles using wearable acceleration sensors," The 3rd IEEE International Conference on Big Data Security on Cloud (Big-DataSecurity 2017), pp.14-17, May 2017.
 [16] Y. Yamato, et al., "Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles," International Journal of Information and Electronics Engineering, Vol.6, No.5, pp.289-293, Sep. 2016.
 [17] Y. Yamato and M. Takemoto, "Method of Service Template Generation on a Service Coordination Framework," 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov. 2004.
 [18] Y. Yamato, et al., "Analyzing Machine Noise for Real Time Maintenance," 2016 8th International Conference on Graphic and Image Processing (ICGIP 2016), Oct. 2016.
 [19] T. Sterling, et al., "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.
 [20] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, pp.66-73, 2010.
 [21] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011
 [22] J. Gosling, et al., "The Java language specification, third

edition," Addison-Wesley, 2005. ISBN 0-321-24678-0.

- [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, "Improvement Proposal of Automatic GPU Offloading Technology," The 8th International Conference on Information and Education Technology (ICIET 2020), pp.242-246, Mar. 2020.
- [25] 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.
- [26] 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.
- [27] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par Parallel Processing, 2012.
- [28] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.
- [29] E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.
- [30] Y. Yamato, et al., "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2018.2872545, Sep. 2018.
- [31] Y. Yamato, "Study of parallel processing area extraction and data transfer number reduction for automatic GPU offloading of IoT applications," Journal of Intelligent Information Systems, Springer, DOI:10.1007/s10844-019-00575-8, Aug. 2019.
- [32] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.
- [33] Rose compiler framework web site, <http://rosecompiler.org/>
- [34] Himeno benchmark web site, <http://acc.riken.jp/en/supercom/>
- [35] Numpy web site, <https://numpy.org/>
- [36] Cupy web site, <https://cupy.dev/>
- [37] Clang web site, <http://llvm.org/>
- [38] NVIDIA system management interface web site, <https://developer.nvidia.com/nvidia-system-management-interface>
- [39] s-tui web site, <https://github.com/amanusk/s-tui>