Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 12, Number 1, pages 1–5, January 2023

Evaluation of FPGA reconfiguration after service launch

Yoji Yamato
NTT Network Service Systems Laboratories
NTT Corporation
Musashino-shi, Tokyo 180–8585, Japan
Email: yoji.yamato.wa@hco.ntt.co.jp

***Abstract*—In order to make full use of heterogeneous hardware, it is necessary to have a technical skill of hardware such as OpenCL, and the current situation is that the barrier is high. Based on this background, I have proposed environment-adaptive software that enables high-performance operation by automatically converting application code written for normal CPUs by engineers according to the deployed environment and setting appropriate amount of resources. Until now, I only considered conversions and settings before the start of operation. In this paper, I verify that the logic is reconfigured according to the usage characteristics during operation. I confirm that the application running on the FPGA is reconfigured into another application according to the usage characteristics.**

***Index Terms*—Environment Adaptive Software, Automatic Offloading, FPGA, Reconfiguration during Operation, Cost Performance.**

I. Introduction

From the deceleration prediction of Moore's Law, a multi-core CPU, GPU (Graphics Processing Unit) or FPGA (Field Programmable Gate Array) increases, which is not only increasing the degree of semiconductor integration and the number of clocks of one core CPU (Central Processing Unit) . This type of heterogeneous hardware has come to be used for normal application operation. Microsoft is making efforts such as searching with FPGA [1], and Amazon provides FPGA and GPU instances [2] using cloud technologies ([3]-[8]). In addition, the use of small devices such as IoT devices is increasing (e.g., [9]-[18] as heterogeneous hardware.

However, in order to efficiently use heterogeneous hardware that is not a single core CPU, it is necessary to create and set programs according to the hardware specification, which is a high barrier for most engineers. High knowledge and skill of OpenMP (Open Multi-Processing) [19] for multi-core CPU, CUDA (Compute Unified Device Architecture) [20] for GPU to achieve GPGPU (General Purpose GPU) [21], OpenCL (Open Computing Language) [22] for FPGA, assembly for IoT devices, or so on are often required. Even if we use high level description techniques such as [23]-[25], performance improvement needs much skills.

In order to increase the use of heterogeneous hardware, we think that a platform that enables even ordinary engineers without high knowledge to make the best use of them is necessary. A platform analyzes software that describes processing with the same logic as a normal CPU, appropriately converts and sets it according to the environment of the deployment destination (multi-core CPU, GPU, FPGA, etc.), and adapts to the environment. In the future, these platforms will be required to perform the adaption.

Therefore, I have proposed environment-adaptive software which automatically converts, sets resources, determines the placement and others of the program code once written for normal CPU so that the GPU, FPGA, multi-core CPU that exist in the environment of the placement destination can be used, and makes the applications high performances. At the same time, as elements of environment-adaptive software, I have proposed and evaluated a method of automatically offloading loop statements and functional blocks of codes to GPUs and FPGAs, and a method of appropriately assigning the amount of processing resources such as GPUs [26]-[31].

However, my environmental adaptation has been based on the premise that adaptation processing such as conversion is performed before the start of operation of the application, and it was not studied that it will be reconfigured according to changes in usage characteristics after the start of operation.

This paper focuses on the reconfiguration of FPGA logic, while reconfiguring software according to the usage characteristics during operation. Except for special applications such as the circuit reconfiguration of artificial satellite while using FPGA for accelerating normal applications, there is no example in the commercial cloud, which reconfigures FPGA logic according to usage characteristics during application operation. FPGA reconfiguration during application operation is difficult and effective, we think. First, we offload a normal CPU program to the FPGA and start operation, analyze the request characteristics, propose to change the FPGA logic to another program, and examine and evaluate a method for reconfiguration with less user effect. The effectiveness of the proposed method is confirmed through the FPGA configuration of the existing application and the reconfiguration during operation.

II. FPGA reconfiguration during operation

A. Review of the automatic FPGA offload method before operation start

I review the automatic FPGA offload method of the loop statement that has been verified by my previous papers [31] simply.

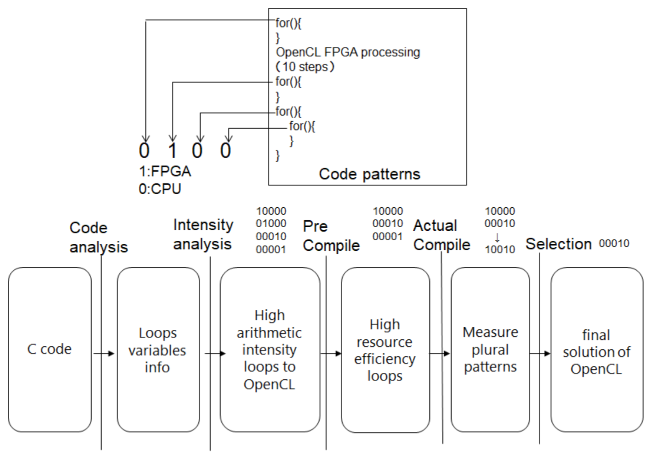


Fig. 1. Automatic FPGA offload of loop statements

For the automatic FPGA offload of the loop statement is made some offload patterns and exploring high speed patterns through Clang [32] analysis and verification environment measurement by focusing on loop statements with high arithmetic intensity and loops numbers (See, Fig. 1). At the time of the GPU, the combination was performed in most loop statements by Genetic Algorithms [33], and the measurement of 1000 times scale was performed to explore the optimal pattern. In FPGA, it takes more than 6 hours to compile, so the number of times measurements is narrowed down.

B. Basic policy for FPGA reconfiguration during operation

By II.A method, the application specified by the user can automatically offload the loop statements suitable for FPGA to FPGA.

After offload to the production environment used by the user, the user can check the actual performance and price in the production environment, and the user will start or not using the application. However, the performance optimization test case used in II.A (the item to be measured when comparing the performance in multiple offload patterns) uses the assumed usage data specified by the user before the operation starts. It is possible that it will be greatly separated from the data that will be used after the operation starts.

Therefore, in II.B, the usage after the operation starts is different from the initial assumption, and the performance may be improved by offloading other logic to FPGA. At this time, it is considered to be reconfigured with a low user impact of FPGA logic. The reconfiguration may be changed to a different loop statement offload in the same application, or may be changed to offload of different applications.

There are two types of FPGA reconfiguration: dynamic reconfiguration and static reconfiguration. The former is a technology that changes the circuit configuration while running the FPGA and the time for rewriting is msec order, the latter is a technology that stops the FPGA and then changes the circuit configuration, and the break time is about 1 second. Depending on the degree of user impact of the break time, we can select a reconfiguration method provided by FPGA vendors. However, either method has a break time, and it requires a test for operation confirmation after reconfiguration, therefore, I do not think that it should be reconfigured frequently. To restrict frequent reconfiguration, reconfigurations are only proposed when the improvement effect is higher than the threshold.

The reconfiguration processing begins with an analysis of request tendency for a certain period such as one month. The functions of the proposed method analyze the request tendency and understand whether there are applications which load are higher or the same than that of the currently offload application. Next, applications with high load are executed in the verification environment for the FPGA offload optimization using data actually used in production use instead of the assumed usage data. Here, it is determined whether the new offload pattern found by verification has a much higher improvement effect than the current offload pattern or less comparing the threshold value. If the improvement exceeds the threshold, a reconfiguration is proposed to the user. After the user acceptance, the production environment is reconfigured. At the time of reconfiguration, it is reconfigured to reduce user impact as much as possible.

C. Method proposal of FPGA reconfiguration during operation

Based on the basic policy of II.B, II.C proposes a concrete reconfiguration method. The reconfiguration method consists of 6 steps, and each step is explained in detail. In particular, step 1 is complicated, so I will add supplementary explanation it at the end.

1. For a certain period of time (long term), production request data is analyzed, and multiple applications with high processing time load are identified and production representative data when using the applications are acquired.

1-1. The actual processing time and the number of usage times from each application usage history for a certain period are calculated.

However, for an application that is offloaded to FPGA, the processing time is calculated assuming that it is not offloaded. From the test history in the assumed usage data before the start of operation, (actual processing time with CPU processing only)/(actual processing time with FPGA offload) is calculated to set the improvement coefficient. Next, the total processing time is used for comparing to calculate the sum value of (the improvement coefficient)*(the actual processing time).

1-2. The total actual processing time with all applications are compared.

1-3. Based on the total actual processing time with all applications, multiple applications with high load are identified.

1-4. Request data for a certain period (short term) of the high load applications are obtained. The distributions of request data according to the data size are created.

1-5. From the actual request data corresponding to the Mode value of the data size distribution, one data is selected as the production representative data.

2. Offload patterns are extracted through verification environment measurement, which speeds up the test case of production representative data for multiple high load applications.

2-1. 4 for statements with high arithmetic intensity are selected for each high load application.

2-2. 4 OpenCL with high arithmetic intensity loop are created and pre-compiled. Resource usage of each OpenCL are showed, then 3 OpenCL with high values of arithmetic intensity/resource usage are selected.

2-3. 3 OpenCL is measured with production representative data. Then, an OpenCL that combines 2 for statements with top 2 performances is created and additionally measured the performance as well.

2-4. The highest speed offload pattern in 4 measurements is selected as a final solution for each high load application.

3. The processing time of the current offload pattern and the extracted multiple new offload patterns are measured using production representative data, and the performance improvement effects based on the frequency of production use are calculated.

3-1. Calculation with the current offload pattern (actual processing time reduction in verification environment)*(frequency of production use).

3-2. Calculation with multiple new offload patterns (actual processing time reduction in verification environment)*(frequency of production use).

4. The reconfiguration proposal is determined by the performance improvement effect of the new offload pattern is more than the threshold of the current offload pattern.

4-1. (3-2)/(3-1) of each high load application is calculated, and calculated values are checked if the value is higher than the threshold. If the value is more than the threshold, the reconfiguration is proposed, and nothing is done if the value is below the threshold.

5. A reconfiguration of FPGA is proposed to the contract user and the user responses OK or NG.

6. A static reconfiguration is conducted by starting new OpenCL in a production environment.

6-1. New offload pattern compilation.

6-2. Stop operation of the current offload pattern.

6-3. Start the operation of the new offload pattern.

The method selects high load applications in step 1. For the current FPGA offload application, the processing time is calculated by applying the improvement coefficient to calculate where it is not offloaded to compare other applications which are processed CPU only. When choosing a representative data, the average data size may vary significantly from the actual production data, I use the mode value of the data size.

III. Evaluation

A. Evaluation conditions

1) Evaluated applications: The evaluated applications are mainly signal processing and image processing, which are expected to be used by many users in FPGAs.

The time-domain finite-impulse response filter (tdFIR) for signal processing is a type of filter that cuts off the output when an impulse function is input to the system in a finite time. There are various implementations, but the C code of [34] is used. When considering an application that transfers signal data from a device to a network in IoT or other situations, it is assumed that the data will be sent to the cloud after processing signals such as filters in order to reduce network costs. Therefore, I think that the automatic speed-up of signal processing in FPGA has a wide range of applications.

MRI-Q [35] is an MRI image processing that calculates the Q-matric that represents the scanner settings for calibration. MRI-Q is used in a 3D MRI reconstruction algorithm in non-Cartesian space. In IoT or other situations, image processing is often required for automatic monitoring of camera images, and performance of image processing throughput is required to be enhanced. In the performance measurement at the time of offload pattern extraction, MRI-Q performs 3D MRI image processing and depends on the data size, but in the assumed usage, the processing time is measured using 64*64*64 size data.

In addition, the Himeno benchmark [36] for uncompressed fluid analysis, Symm (Symmentry matrix manipulation) [37] for symmetric matrix calculation, and DFT (Discrete Fourier transform) [38] for discrete Fourier transform calculation are run on the same server and execution requests are received.

2) Evaluation methods: I confirmed the proposed method. Before the start of operation, the user specifies the offload of tdFIR and automatically offloads it to FPGA. In the production environment, only tdFIR is offloaded to FPGA, and MRI-Q, Himeno Benchmark, Symm and DFT are run by CPU only processing. A request load is applied to the production environment server for a certain period of time, the requests are analyzed, a reconfiguration to a new offload pattern with a high performance improvement effect is proposed and the reconfiguration is performed after user approval.

The conditions for FPGA offload are as follows.

Offload target: Number of loop statements. tdFIR 6, MRI-Q 16, Himeno 13, Symm 9, DFT 10.

Narrow down of Arithmetic Intensity: Narrowing down to the top 4 loop statements in arithmetic intensity analysis of ROSE framework [39]

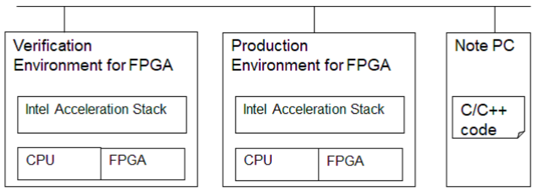
Narrow down of resource efficiency: Narrow down to the top 3 loop statements in resource efficiency analysis

Number of measured offload patterns: 4 (The first measurement measures 3 offload patterns with high resource efficiency, and the second measurement measures the combination pattern of the 2 loop statement offloads that are high performance in the first measurement.)

The operational conditions for FPGA reconfiguration are as follows.

Request frequency: tdFIR 300 req/h, MRI-Q 10 req/h, Himeno 3 req/h, Symm 2 req/h, DFT 1 req/h requests are applied for 2 hours.

Types of data: 3 types of data are prepared. Small size of sample data, large size of sample data, and double of large data which copies and adds large sample data once. In tdFIR and MRI-Q, requests for small, large and double large sizes



Name	Hardware	CPU	RAM	FPGA	OS	Intel Acceleration Stack
Verification Environment for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	32GB *4	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	CentOS 7.9	2.0
Production Environment for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	32GB *4	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	CentOS 7.9	2.0
Note PC	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro	

Fig. 2. Evaluation environments

are requested at a ratio of 3:5:2. Himeno, Symm and DFT are only requested the data which are same with equipped sample data.

Long term during load analysis: 2 hours

Short term when selecting representative data: 1 hour

Number of high load applications: 2

Threshold of performance improvement effect: 2.0

During the reconfiguration, the performance improvement effect and the processing time of each step associated with the reconfiguration are acquired.

3) Evaluation environments: Intel FPGA PAC D5005 (Intel Stratix 10 GX FPGA, Logic Element 2,800,000) is used as the evaluation FPGA. The server equipped with Intel FPGA PAC D5005 is DELL EMC PowerEdge R740 (CPU: Intel Xeon Bronze 3206R * 2, RAM: 32GB RDIMM * 4). Intel Acceleration Stack Version 2.0 is used for FPGA control. By dividing the C language program into a kernel program and a host program according to the OpenCL syntax, FPGA offload processing is performed by OpenCL, and reconfiguration to another OpenCL program is also processed by Intel Acceleration Stack.

Figure 2 shows the evaluation environment and specifications. Here, the note PC specifies the application code to be offloaded, extracts the offload pattern through performance measurement in the verification environment, and then deploys it to the production environment. Execution requests are made to the production environment applications from the note PC periodically. The production environment requests are analyzed, new offload patterns are extracted using the verification environment, and after user confirmation, the production environment is reconfigured into the new offload pattern.

B. Results

Figure 3 shows the degree of improvement in the processing time of the offload application before and after the reconfiguration and the total processing time (corrected for the improvement coefficient) for a certain period. First, tdFIR was offloaded before the reconfiguration, the degree of improvement in the assumed data before the start of operation was 2.07, and the load of 300 req/h was applied after the start of operation. 159 seconds which calculated from the total actual processing time of the request * 2.07 is the total corrected processing time, and 600 is the total number of uses. Next, MRI-Q has a load of 10 req/h after the start of operation. 549 seconds of the total actual processing time of the request is the total processing time, and 20 is the total number of uses. These two are the applications with the high load. These two searches for new offload patterns using production representative data. The processing time for one time in the new offload pattern is reduced from 0.266 seconds to 0.129 seconds for tdFIR and from 27.4 seconds to 2.23 seconds for MRI-Q. By multiplying the number of production uses, the processing time is reduced. The degree of performance improvement is 41.1 seconds/h for tdFIR and 252 seconds/h for MRI-Q.

	Application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	tdFIR	41.1 sec/h	159 sec
After reconfiguratoin	MRI-Q	252 sec/h	549 sec

Fig. 3. Comparison of performance improvement through reconfiguration of the proposed method

From Fig. 3, the change from tdFIR offload to MRI-Q offload increases the performance improvement by 6.1 times and exceeds 2.0, so reconfiguration is proposed to the user.

The size of the request analysis is small because only several hours data is analyzed in this time, but it will take longer in proportion to the size. This time, it takes about only 1 second for request analysis and production representative data selection, about 1 day for improvement effect calculation, and about 1 second for reconfiguration. Regarding the offload trial before the start of operation, the trial of the new offload patterns searches during operation, and the time to compile the new offload pattern one the production environment, since one FPGA compilation takes 6 hours or more, the number of measurements is 4 takes more than a day for one application. Most of the processing such as analysis, including the trial of the new offload patterns searches, is performed in the background during application operation in the production environment, so there is no user impact. The only thing that can be confirmed is that the break time of the application is necessary for production reconfiguration. However, the static reconfiguration of OpenCL takes about only 1 second, and there is almost no effect. If a shorter break time is required, it is possible to use the dynamic reconfiguration function of FPGA vendors.

I confirmed the FPGA reconfiguration according to the usage characteristics during operation by changing from tdFIR offload to MRI-Q offload during operation. Through the reconfiguration, the degree of performance improvement increased above the threshold value, and it was shown that the break time was sufficiently short.

IV. Conclusion

In this paper, as an element of it, I have proposed an FPGA reconfiguration method that reconfigures the appropriate FPGA logic during operation according to the usage characteristics after the application operation starts.

Before starting operation, the application loop statement is automatically offloaded to the FPGA. In the proposed method, the applications with large CPU processing times are analyzed from the actual request data at regular intervals, and the corresponding representative test cases are gathered. Next, the offload patterns that speed up representative test cases are extracted through trial measurement in the verification environment for large load applications. This is almost the same as offload before the start of operation. Next, the processing time of the current offload pattern and the extracted new offload pattern are measured, and the processing time improvement based on the frequency of production use is calculated. Here, if the new offload pattern has an effect greater than the threshold of the current offload pattern, our method proposes to the users to carry out reconfiguration. Once user consent is obtained, our method reconfigures FPGA logic using OpenCL reconfiguration in a production environment. The application that was automatically offloaded to the FPGA, and the FPGA logic was reconfigured during operation to another application in the experiment. The reduction in processing time was improved by reconfiguration, and reconfiguration was performed with a short interruption time of about 1 second. Thus, the effectiveness of the proposed method was confirmed.

References

[1] 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), pp.13-24, June 2014.

[2] AWS EC2 web site, https://aws.amazon.com/ec2/instance-types/

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

[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, et al., "Fast Restoration Method of Virtual Resources on OpenStack," IEEE Consumer Communications and Networking Conference (CCNC2015), Las Vegas, pp.607-608, Jan. 2015.

[6] Y. Yamato, "Automatic verification for plural virtual machines patches," The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, Sapporo, July 2015.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[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, 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.

[16] Y. Yamato, et al., "Analyzing Machine Noise for Real Time Maintenance," 2016 8th International Conference on Graphic and Image Processing (ICGIP 2016), Oct. 2016.

[17] Y. Yamato, "Experiments of posture estimation on vehicles using wearable acceleration sensors," The 3rd IEEE International Conference on Big Data Security on Cloud (BigDataSecurity 2017), pp.14-17, May 2017.

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

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

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

[21] 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.

[22] 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.

[23] S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.

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

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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, 2019.

[31] 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.

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

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

[34] Time domain finite impulse response filter web site, http://www.omgwiki.org/hpec/files/hpec-challenge/tdfir.html

[35] MRI-Q website, http://impact.crhc.illinois.edu/parboil/

[36] Himeno benchmark web site, http://accc.riken.jp/en/supercom/

[37] Polybench symm website, https://web.cse.ohio-state.edu/ pouchet.2/software/polybench/

[38] DFT website, http://programming.blogo.jp/c/fourier_transform

[39] ROSE compiler framework web site, http://rosecompiler.org/