Evaluation of GPU Logic Reconfiguration after Service Start

Yoji Yamato
Network Service Systems Labs
NTT Corporation
Tokyo, Japan
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 CUDA, 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 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 GPU is reconfigured into other loops or applications offloading according to the usage trends.**

Keywords—environment adaptive software, automatic offloading, GPGPU, reconfiguration during operation

I. INTRODUCTION

In recent years, it is said that Moore's law, which states that the degree of semiconductor integration of CPUs will double in 1.5 years, will slow down. Under such circumstances, not only CPUs but also devices such as FPGAs (Field Programmable Gate Arrays) and GPUs (Graphics Processing Units) are being increasingly used. For example, Microsoft is making efforts to improve the search efficiency of Bing using FPGA [1], and Amazon provides FPGA, GPU as instances [2] using Cloud technologies (e.g., [3]-[9]). In addition, small devices such as IoT devices (e.g., [10]-[18]) are also increasing.

However, in order to properly utilize devices other than CPUs with a small number of cores in the system, it is necessary to make settings and program coding that are conscious of device characteristics. For example, knowledge such as OpenMP (Open Multi-Processing) [19], OpenCL (Open Computing Language) [20], CUDA (Compute Unified Device Architecture) [21] for GPGPU (General Purpose GPU) [22] is required. Even if we use high level description techniques such as [23]-[26], performance improvement needs much skills.

It is expected that the number of systems that utilize devices such as GPUs, FPGAs, and multi-core CPUs other than few core CPUs will increase in the future, but there are high technical barriers to making the best use of them. Therefore, in order to remove such a barrier and make it possible to fully use devices other than few core CPUs, the software in which the programmer describes the processing logic is adapted to the environment (FPGA, GPU, multi-core CPU or so on) of the deployment destination. Therefore, there is a need for a platform that can be adaptively converted, configured, and operated according to the environment.

I proposed environment-adaptive software so that the code once written can be used with the GPU, FPGA, multi-core CPU or so on that exist in the environment of the deployment destination. The environment-adaptive software automatically performs conversion, resource setting and so on to operate the application with high performance. At the same time, I have also proposed and evaluated methods of automatically offloading loop statements and function blocks of C language program code to GPU and FPGA as elements of environment-adaptative software [27]-[32].

However, my verification has so far only performed adaptation processing such as conversion before the start of operation, and has not considered re-adapting according to the actual usage characteristics in the production environment during operation. We will consider an example of image processing. Before the start of operation, the logic was built so that GPU processing would be performed on the premise that there would be a lot of classification processing. However, when analyzing the actual number of requests 3 months after the start of operation, object detection process may be larger. In this case, it is better to change the logic that performs GPU processing.

This paper focuses on the reconfiguration of GPU logic, while reconfiguring the applications according to usage characteristics after the start of operation. There is no example of reconfiguring GPU logic according to usage characteristics during operation using GPU for application acceleration in the production cloud (AWS GPU instance or so on). GPU reconfiguration has a big impact, I think. First, the existing application is automatically offloaded to the GPU and the operation is started. I propose a reconfiguration method which analyzes the usage characteristics, suggests to the user to reconfigure the GPU logic to another offload, and changes it with a shorter break time. The effectiveness of the proposed method is evaluated by the performance improvement and the break time through GPU reconfiguration during operation.

II. GPU RECONFIGURATION DURING OPERATION

A. Review of Automatic GPU Offload before Operation Start

I review the automatic GPU offload method verified in my previous papers. There are two main features for automation. The point that loop statement extraction suitable for GPU is performed using the genetic algorithm (GA) [33] is proposed in

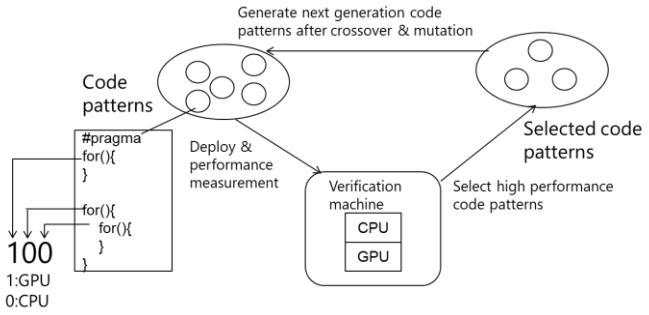


Fig. 1. Automatic GPU offload method for loop statements.

[27] (see, Fig. 1), and the point to suppress GPU-CPU data transfer, the variables used in the nested loop statement are transferred on the upper loop as much as possible is proposed in [32]. Based on these two ideas, I have confirmed that the automatic speedup is several times higher even in medium sized applications with more than 100 loop statements.

B. Basic Policy for GPU Reconfiguration during Operation

By the method in A, the loop statements suitable for the GPU can be automatically offloaded to the GPU in the application specified by the user. After offloading to the production environment used by the user, the actual performance and price in the production environment are confirmed, and the user starts using the application. However, the performance optimization test case (item for performance measurement when comparing performance with multiple offload patterns) used for offload in previous subsection uses the assumed usage data specified by the user because the operation is not started. Therefore, there is a possibility that the data will be significantly different from the data actually used during operation.

Therefore, in this subsection, I study that if the usage pattern after the start of operation is different from the initial assumption and the performance is improved by offloading another logic to the GPU, the GPU logic should be reconfigured with less influence on the user. Reconfiguration may change to different loop statements offload in the same application, or it may change to a different application offload.

GPU offload logic reconfiguration requires changing the executed OpenACC, but there are multiple ways to do it. First, there is a method of stopping current OpenACC and starting new OpenACC on the running machine. OpenACC is stopped and started in a short time, and the break time is about several seconds. Next, there is a method of newly building the machine itself that executes new OpenACC, then switching the routing to the new machine at the timing when all the current OpenACC processing is completed, and then stopping the current running machine. Routing switching is short and there is almost no break time. Depending on the degree of user influence, the GPU logic reconfiguration method may be selected, but in either case, there will be a slight break time, and changes to another logic will require an operation confirmation test, therefore, I think reconfiguration should not be done frequently. I set restrictions such as proposing only when the effect is above the threshold.

The reconfiguration process begins with an analysis of request trends over a period of long term, such as one month. Request trends are analyzed to see if there is high processing load than the currently offloaded application. Next, for applications with high processing load, GPU offload optimization trials are performed in the verification environment using data that is actually used for production environment instead of initial assumed usage data. Then, it is judged whether the new offload pattern found by the verification has a sufficiently higher improvement effect than the current offload pattern, depending on if it is above or below the threshold. If the effect exceeds the threshold, the user is proposed to reconfigure. After the approvals, the production environment is reconfigured, but the user influence is suppressed as much as possible.

C. Method Proposal of GPU Reconfiguration during Operation

Based on the basic policy, this subsection proposes a specific reconfiguration method. The reconfiguration method consists of 6 steps, and each step is explained in detail.

1. Analyze the production request data history for a certain period (long term), identify multiple applications with high processing time load, and acquire representative data when using those applications.

1-1. Calculate the actual processing time and the total number of uses from each application usage history for a certain period. However, for application that is GPU offloaded, the processing time if not offloaded is calculated. From the test history of the assumed usage data before the operation, calculate the improvement coefficient by (actual processing time when only CPU processing is performed)/(actual processing time when GPU is used). Next, the sum of the values obtained by multiplying the actual processing time by the improvement coefficient is used as the total processing time for comparison.

1-2. Compare the total actual processing time for all applications.

1-3. Sort by the total actual processing time, and identify multiple applications with high processing time load.

1-4. Acquire request data for a certain period (short term) of high load applications, arrange the data size for each fixed size, and create a frequency distribution.

1-5. Select one of the actual request data corresponding to the Mode class of the data size frequency distribution and select it as the representative data.

2. Offload patterns are extracted through verification environment measurement to speed up test cases of production representative data in multiple high load applications.

2-1. In high load applications, count the number of for statements by parsing such as Clang [34], and use the GPU compiler function to find and remove for statements that cannot be processed by the GPU.

2-2. Create a certain number of gene patterns with the number of remaining for statements as the gene length, 1 means to GPU computation, 0 means to CPU execution, and corresponding OpenACC directives are added. Directives are both GPU computations such as #pragma acc kernels and data processing such as #pragma acc data copy.

2-3. Compile the OpenACC file corresponding to the genepattern in the verification environment machine, measure the performance in the test case of the production representative data, and set the higher goodness of fit for faster patterns.

2-4. Depending on the goodness of fit, GA processing such as crossover is performed, next-generation gene patterns are created, and GA processing is repeated for a certain number of generations, and the final highest performance pattern is determined a solution.

3. The processing time with production representative data of the current offload pattern and the extracted new offload patterns is measured, and the performance improvement effect based on the frequency of production use is obtained.

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

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

4. Reconfiguration proposal is judged based on whether the performance improvement effect of the new offload pattern is higher than that of the current offload pattern.

4-1. Calculate high load applications (3-2)/(3-1), check if it is above the threshold, then propose a reconfiguration if it is above, and do nothing if it is below.

5. Propose GPU reconfiguration to the contracted user and get an OK/NG approval.

6. Start new OpenACC in a production environment and reconfigure. There are two methods of GPU reconfiguration.

6-1. Stop the old OpenACC and start the new OpenACC.

6-1'. A new machine that processes the new OpenACC is built, and the routing is changed so that the new processing is sent to the new machine.

III. EVALUATION

A. Evaluation Condition

1) Evaluated applications

The evaluated applications are mainly neural networks, Fourier transforms, and fluid calculations that are expected to be used by many users on GPUs.

Darknet [35] is a neural network framework that can perform various processing such as classification, detection and nightmare, but this time, speeds up of object detection and image modification, which are the basic processing are confirmed. There are various types of neural networks, and there are many libraries for GPU, but as an example, I use Darknet which is written in C language. In the offload verification before operation, detection processing (image detection) is used in the sample application installed in Darknet.

The Fourier transform process is used in various situations of monitoring in IoT such as analysis of vibration frequency. NAS.FT [36] is one of the open source applications for FFT (Fast Fourier Transform) processing. When considering an application that transfers data from a device to a network in IoT, it is expected that the device will perform primary analysis such as FFT processing and send it to reduce network costs. For offload verification before operation, the sample test case attached to NAS.FT is used. The basic data of Class A is that the grid size is 256*256*128 and the number of iterations is 6.

Himeno Benchmark [37] is a benchmark software used to measure the performance of incompressible fluid analysis, and solves the Poisson equation solution by the Jacobi iterative method. It is frequently used for manual speedup on GPU, and is used in this time to confirm that speedup can be done automatically. The basic data used for offload is LARGE (512*256*256 grid size).

NAS.BT [38] for block diagonal solver computation and MRI-Q [39] for 3D MRI image processing are run on the same machine and receive execution requests.

2) Evaluation method

Assuming two users, user A specifies Darknet detection and user B specifies NAS.FT for automatic offloading before operation.

The following conditions are performed in an initial offloading.

Number of loop statements: Darknet 171, NAS.FT 81, Himeno 13, NAS.BT 120, MRI-Q 16

Number of individuals M: Less than the number of loop statements (Darknet 30, NAS.FT 30, Himeno 10, NAS.BT 30, MRI-Q 10)

Number of generations T: less than the number of loop statements (Darknet 20, NAS.FT 20, Himeno 10, NAS.BT 20, MRI-Q 10)

Goodness of fit: (processing time)$^{-1/2}$.

The shorter the processing time, the higher the goodness of fit. The (-1/2) power prevents the goodness of fit of a particular individual with a short processing time from becoming too high. It prevents the search area from becoming too narrow.

Selection: Roulette selection. To keep (not crossed or mutated) best goodness of fit gene in a generation, elite selection to next generation is also applied.

Crossover rate Pc: 0.9

Mutation rate Pm: 0.05

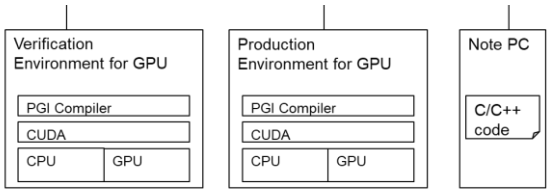
The operational conditions for reconfiguration are as follows.

Request load:

User A: Darknet detection 16 req/h, nightmare 20 req/h, NAS.FT 4 req/h, Himeno 3 req/h, NAS.BT 2 req/h, MRI-Q 1

req/h. Darknet detection and nightmare (image modification) processing uses equipped 3 images with 111 KB, 170 KB, and 374 KB. These data are requested in a ratio of 3:5:2. Data of NAS.FT, Himeno, NAS.BT and MRI-Q are same as sample data.

User B: Darknet detection 4 req/h, nightmare 3 req/h, NAS.FT 20 req/h, Himeno 30 req/h, NAS.BT 2 req/h, MRI-Q 1 req/h. In NAS.FT, Class W, A, and B sizes of sample data are requested in a ratio of 3:5:2. In Himeno, M, L, and XL sizes of



Name	Hardware	CPU	RAM	GPU	OS	CUDA toolkit	PGI Compiler
Verification Environment for GPU	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX	64GB	NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory: GDDR6 11GB)	Ubuntu 16.04.6 LTS	10.1	19.10
Production Environment for GPU	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX	64GB	NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory: GDDR6 11GB)	Ubuntu 16.04.6 LTS	10.1	19.10
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro		

Fig. 2. Evaluation environment.

sample data are requested in a ratio of 2:5:3. The data of Darknet detection and nightmare, NAS.BT and MRI-Q are same as sample data.

Long term during load analysis: 2 hours

Short term during representative data selection: 1 hour

Number of high load applications: 2

Performance improvement effect threshold: 2.0

During the reconfiguration, the performance improvement effect and the processing time of each step are obtained.

3) Evaluation environment

NVIDIA GeForce RTX 2080 Ti is used as the evaluation GPU. The machine equipped with GeForce RTX 2080 Ti is Iiyama LEVEL-F039-LCRT2W-XYVI. GPU control uses PGI compiler 19.10 and CUDA toolkit 10.1. By adding the \#pragma directives to the C language program according to the OpenACC syntax, GPU offload processing is performed, and reconstruction to another OpenACC program is also processed by the PGI compiler. Figure 2 shows the evaluation environment.

B. Reults

Figure 3 shows the degree of improvement in processing time before and after reconfiguration of users A and B, and the total processing time (corrected for improvement coefficient) for a certain period related to it.

Before the reconfiguration of user A, 4 for statements of Darknet are offloaded, the degree of improvement in the assumed data before operation is 2.92, and the loads are 16 req/h for detection processing and 20 req/h for nightmare processing after the start of operation. 5,960 seconds that can be calculated by the total actual processing time of the request*2.92 is the total corrected processing time. As a result of the calculation, Darknet and NAS.FT are 2 high load applications. Among them, after the start of operation, nightmare processing of Darknet was used many times in production environment, thus new offload patterns for nightmare processing using representative data are searched, found a new pattern that offloads 13 for statements, and reduced the total processing time which multiplies the number of production uses. The degree of improvement is 96 seconds/h for Darknet before reconfiguration and 1,850 seconds/h for Darknet after reconfiguration.

User A	Offload application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	Darknet (4 loops)	96.0 sec/h	5,960 sec
After reconfiguratoin	Darknet (13 loops)	1,850 sec/h	5,960 sec

User B	Offload application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	NAS.FT	308 sec/h	2,420 sec
After reconfiguratoin	Himeno benchmark	1,180 sec/h	2,790 sec

Fig. 3. Performance improvement through proposed reconfiguration.

Before the reconfiguration of user B, NAS.FT was offloaded, and the degree of improvement in the assumed data was 2.54. 2,420 seconds that can be calculated by the total actual processing time of the request*2.54 is the total corrected processing time. As a result of the calculation, Himeno and NAS.FT are 2 high load applications. By searching for offload patterns using representative data after the start of operation and multiplying the number of production uses, the degree of improvement in processing time reduction is 308 seconds/h in NAS.FT before reconfiguration and 1,180 seconds/h in Himeno after reconfiguration.

From Fig. 3, the improvement threshold value 2.0 is checked, and in the case of user A, a reconfiguration with offload loop statement change of Darknet is proposed, and in the case of user B, a reconfiguration with offload application change from NAS.FT to Himeno is proposed.

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 representative data selection, less than a day for improvement effect calculation, and less than 1 second for reconfiguration. For offload trials before the operation and new offload pattern trials during operation, it takes about 7 hours to measure the performance of GAs of several tens of generations, so it takes less than a day if there are three high load applications. However, most of the processing such as analysis, including the trial of the new offload pattern, is performed in the background during the operation in the production environment, so there is no user influence. Only, regarding the conduction of production environment reconfiguration, the application has a break time, but it takes less than 1 second even if current OpenACC is stopped and new OpenACC is started, and it was confirmed

that there is almost no effect. It is also possible to prepare OpenACC on the new machine and switch the routing.

It was confirmed that the GPU processing application in operation was reconfigured to a different loops or to a different application offload according to the usage characteristics of each user. It was shown that the degree of performance improvement increased above the threshold through the reconfiguration and the break time was sufficiently short.

IV. Conclusion

In this paper, as the elemental technology, I proposed a GPU reconfiguration method during that reconfigures the appropriate GPU logic during operation according to the usage characteristics after the application operation starts.

Before starting operation, one of application loop statement is automatically offloaded to the GPU. In the proposed method, the applications with the high CPU processing time load are acquired from the actual request data at regular intervals, and the corresponding representative test case are also acquired. Next, the new offload patterns that speed up representative test cases are extracted through trial measurements in the verification environment. This is almost the same as offload before the start of operation. Next, the processing time of the current offload pattern and the new offload patterns are measured, and the processing time improvement based on the frequency of production use are calculated. Here, if the new offload pattern has an effect greater than the threshold of the current pattern, the implementation proposes to the user to conduct reconfiguration. After obtaining the user's consent, the implementation stops OpenACC in the production environment and reconfigure GPU logic by starting the new OpenACC. The experiment showed that the GPU logic was reconfigured to offload other loop statements or other applications by reconfiguring the application that was automatically offloaded to the GPU during operation according to the usage characteristics. The reduction in processing time was improved by reconfiguration, and at the same time, reconfiguration was performed with a short break time, and the effectiveness of the 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, "Server Structure Proposal and Automatic Verification Technology on IaaS Cloud of Plural Type Servers," International Conference on Internet Studies (NETs2015), July 2015.

[6] 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.

[7] 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.

[8] 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.

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

[11] 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.

[12] 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.

[13] 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.

[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. 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.

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

[22] 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.

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

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

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

[35] Darknet website, https://pjreddie.com/darknet/

[36] NAS.FT website, https://www.nas.nasa.gov/publications/npb.html

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

[38] NAS.BT website, https://www.nas.nasa.gov/publications/npb.html

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