Title	Study and Evaluation of Optimum Location Deployment for Environment Adaptive Applications
Author(s)	Yoji Yamato
Citation	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）： International Journal of Parallel, Emergent and Distributed Systems ・DOI (URL) https://doi.org/10.1080/17445760.2022.2088749 Publication Date: yyyy/mm/dd ・出版日：2022 年 6 月 20 日 Publisher ・出版者：Taylor and Francis
Declaration	This preprint is the Author Accepted Manuscript(AAM) of the above. ・本プレプリントは、上記論文の (受理済)著者最終稿(AAM) である。 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
Notes	"This is an Accepted Manuscript of an article published by Taylor & Francis in International Journal of Parallel, Emergent and Distributed Systems, on 20th Jun., 2022, available online: https://doi.org/10.1080/17445760.2022.2088749."

ORIGINAL PAPER

Study and evaluation of optimum location deployment for environment adaptive applications

Yoji Yamato[a]

[a]Network Service Systems Laboratories, NTT Corporation, 3-9-11 Midori-cho, Musashino-shi, Tokyo 180-8585, Japan

ARTICLE HISTORY
Compiled May 20, 2022

ABSTRACT
Heterogeneous hardware other than a small-core central processing unit (CPU) such as a graphics processing unit (GPU), field-programmable gate array (FPGA), or many-core CPU is increasingly being used. However, to use heterogeneous hardware, programmers must have sufficient technical skills to utilize OpenMP, CUDA, and OpenCL. On the basis of this, I have proposed environment-adaptive software that enables automatic conversion, configuration, and high performance operation of once written code, in accordance with the hardware. However, although it has been considered to convert the code according to the offload devices, there has been no study where to place the offloaded applications to satisfy users' requirements of price and response time. In this paper, as a new element of environment-adapted software, I examine a method to calculate appropriate locations using linear programming method. I confirm that applications can be arranged appropriately through simulation experiments when some conditions such as application type and users' requirements are changed.

KEYWORDS
Environment Adaptive Software; Automatic Offloading; Optimum Placement; Linear Programming; User Requirements

1. Introduction

As Moore's Law slows down, a central processing unit's (CPU's) transistor density cannot be expected to double every 1.5 years. To compensate for this, more systems are using heterogeneous hardware, such as graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and many-core CPUs. For example, Microsoft's search engine Bing uses FPGAs [1], and Amazon Web Services (AWS) provides [2] GPU and FPGA instances using cloud technologies (e.g., [3]-[7]). Systems with Internet of Things (IoT) devices are also increasing (e.g., [8]-[14]).

However, to properly utilize devices other than small-core CPUs in these systems, configurations and programs must be made that consider device characteristics, such as Open Multi-Processing (OpenMP) [15], Open Computing Language (OpenCL) [16], and Compute Unified Device Architecture (CUDA) [17]. In addition, embedded software skills are needed for controlling the details of IoT devices. Therefore, for most programmers, skill barriers are high.

CONTACT Yoji Yamato Email: yoji.yamato.wa@hco.ntt.co.jp

The expectations for applications using heterogeneous hardware are becoming higher; however, the skill hurdles are currently high for using them. To surmount these hurdles, application programmers should only need to write logics to be processed, and then software should adapt to the environments with heterogeneous hardware to make such hardware easy to use. Java [18], which appeared in 1995, caused a paradigm shift in environment adaptation that enables software written once to run on another CPU machine. However, no consideration was given to the application performance at the porting destination.

Therefore, I previously proposed environment-adaptive software that effectively runs once-written applications by automatically executing code conversion and configurations so that GPUs, FPGAs, many-core CPUs, and so on can be appropriately used in deployment environments. For an elemental technology for environment-adaptive software, I also proposed a method for automatically offloading loop statements and function blocks of applications to GPUs or FPGAs [19]-[23].

This paper is for optimizing the applications placement when a normal CPU program is offloaded to a device such as a GPU, to meet the user's cost requirements and the response time requirements. I propose a method to calculate appropriate locations using a linear programming method. I demonstrate that applications can be arranged appropriately through simulation experiments when conditions such as application type and users' requirements are changed. This paper extends and improves a short paper presented at the International Conference ICIAE by adding evaluation results, discussion of results, and related work.

The rest of this paper is organized as follows. In Section 2, I review technologies on the market and our previous proposals. In Section 3, I propose an appropriate placement method when the application is automatically converted in accordance with the use case. In Section 4, I evaluate the proposed method through simulation experiments. In Section 5, I describe related work, and in Section 6, I conclude the paper.

2. Existing technologies

2.1. Technologies on the market

Java is one example of environment-adaptive software. In Java, using a virtual execution environment called Java Virtual Machine, written software can run even on machines that use different operating systems (OSes) without more compiling (Write Once, Run Anywhere). However, whether the expected performance could be attained at the porting destination was not considered, and there was too much effort involved in performance tuning and debugging at the porting destination (Write Once, Debug Everywhere).

CUDA is a major development environment for general purpose GPUs (GPGPUs) (e.g., [24]) that use GPU computational power for more than just graphics processing. To control heterogeneous hardware uniformly, the OpenCL specification and its software development kit (SDK) are widely used (e.g., [25]). CUDA and OpenCL require not only C language extension but also additional descriptions such as memory copy between GPU or FPGA devices and CPUs. Because of these programming difficulties, there are few CUDA and OpenCL programmers.

For easy heterogeneous hardware programming, there are technologies that specify parallel processing areas by specified directives, and compilers transform these specified parts into device-oriented codes on the basis of directives. Open accelera-

tors (OpenACC) [26] and OpenMP are examples of directive-based specifications, and the Portland Group Inc. (PGI) compiler [27] and gcc are examples of compilers that support these directives.

In this way, CUDA, OpenCL, OpenACC, OpenMP, and others support GPU, FPGA, or many-core CPU offload processing. Although processing on devices can be done, sufficient application performance is difficult to attain. For example, when users use an automatic parallelization technology, such as the Intel compiler [28] for multi core CPUs, possible areas of parallel processing such as "for" loop statements are extracted. However, naive parallel execution performances with devices are not high because of overheads of CPU and device memory data transfer. To achieve high application performance with devices, CUDA, OpenCL, or so on needs to be tuned by highly skilled programmers, or an appropriate offloading area needs to be searched for by using the OpenACC compiler or other technologies. As an effort to automate trial and error of parallel processing area search, I propose automatic GPU offloading using an evolutionary computation method.

Regarding applications placement, there is research on optimizing the inserted position of VN (Virtual Network) for a group of servers on the network as an effective use of network resources [29]. In [29], the optimum placement of VN is determined in consideration of communication traffic. However, it is for single-resource of virtual networks, with the aim of reducing carrier facility costs and overall response time, conditions such as the processing time of individual users' applications, and requirements of the cost and response time are not taken into consideration.

2.2. Previous proposals

To adapt software to an environment, I previously proposed environment-adaptive software [23], the processing flow of which is shown in Figure 1. The environment-adaptive software is achieved with an environment-adaptation function, test-case database (DB), code-pattern DB, facility-resource DB, verification environment, and production environment.

Step 1: Code analysis
Step 2: Offloadable-part extraction
Step 3: Search for suitable offload parts
Step 4: Resource-amount adjustment
Step 5: Placement-location adjustment
Step 6: Execution-file placement and operation verification
Step 7: In-operation reconfiguration

In Steps 1-7, the processing flow conducts code conversion, resource-amount adjustment, placement-location adjustment, and in-operation reconfiguration for environment adaptation. However, only some of the steps can be selected. For example, if we only want to convert code for a GPU, FPGA, or many-core CPU, we only need to conduct Steps 1-3.

I will summarize this section. Because most offloading to heterogeneous devices is currently done manually, I proposed the concept of environment-adaptive software and automatic offloading to heterogeneous devices. However, after automatic conversion, the adjustment of the appropriate location of the offloaded application has not been examined (corresponding to Step 5). Therefore, in this paper, I will consider a method for efficiently arranging automatically converted applications by satisfying the user's cost and response time requirements.

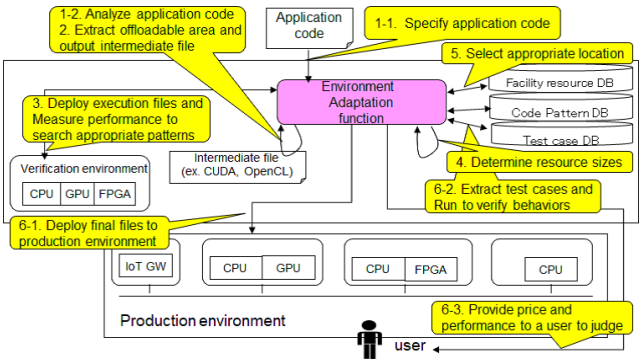

Figure 1. Processing flow of environment adaptive software

3. Appropriate placement of applications

To embody the concept of environment-adaptive software, I have proposed automatic GPU and FPGA offloading of program loop statements and function blocks, offloading from various languages, offloading to mixed devices environment and automatic adjustment of the offloading devices resource amounts. Based on these elemental technology studies, Section 3.1 outlines automatic GPU offloading technology for loop statements as an example, and 3.2 discusses consideration points for deploying applications. In 3.3, I formulate a linear programming method for proper placement of applications.

3.1. Automatic GPU offloading of loop statements

There are many cases where a program running on a normal CPU is speeded up by offloading it to a device such as a GPU or FPGA, but there are few cases where it is automatically conducted. For automatic GPU offloading of loop statements, I have proposed several methods [19][23].

First, as a basic problem, the compiler can find the limitation that this loop statement cannot be processed in parallel on the GPU, but it is difficult to find out whether this loop statement is suitable for parallel processing on the GPU. Loops with a large number of loops are generally said to be more suitable, but it is difficult to predict how much performance will be achieved by offloading to the GPU without actually measuring it. Therefore, it is often the case that the instruction to offload this loop to the GPU is manually given and the performance measurement is tried. On the basis of that, [19] proposes automatically finding an appropriate loop statement that is offloaded to the GPU with a genetic algorithm (GA)[30], which is an evolutionary computation method. From a general-purpose program for normal CPUs that does not assume GPU processing, the proposed method first checks the parallelizable loop statements. Then for the parallelizable loop statements, it sets 1 for GPU execution and 0 for CPU execution. The value is set and geneticized, and the performance verification trial is repeated in the verification environment to search for an appropriate area. By narrowing down to parallel processing loop statements and holding and recombining parallel processing patterns that can be accelerated in the form of gene parts, patterns that can be efficiently accelerated are explored from the huge number of parallel processing patterns (see, Figure 2).

[23] proposes transferring variables efficiently. Regarding the variables used in the

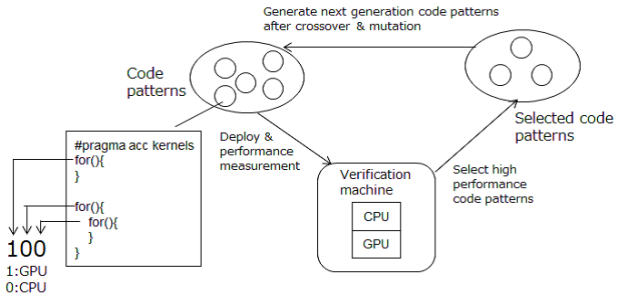

Figure 2. Automatic GPU offload of loop statements

nested loop statement, when the loop statement is offloaded to the GPU, the variables that do not have any problem even if CPU-GPU transfer is performed at the upper level are summarized at the upper level. This is because if CPU-GPU transfer is performed at the lower level of the nest, the transfer is performed at each lower loop, which is inefficient. I also proposed a method to further reduce CPU-GPU transfers. Specifically, for not only nesting but also variables defined in multiple files, GPU processing and CPU processing are not nested, and for variables where CPU processing and GPU processing are separated, the proposed method specifies to transfer them in a batch.

Regarding GPU offload of loop statement, automatic offload is possible by optimization using the evolutionary computation method and reduction of CPU-GPU transfer.

Using a similar method, I have proposed automatic conversion of loop statements for FPGA and automatic offloading of function blocks. I also proposed a method of automatically adjusting the resource amounts of the offload destination device in order to operate with good cost performance after automatic code conversion.

3.2. Consideration points to place applications

By methods such as 3.1, normal CPU programs can be automatically converted into offload devices such as GPUs and FPGAs. In this subsection, I will consider placing the application in an appropriate location after the program conversion.

In method 3.1, multiple offload patterns are repeatedly tried in the verification environment, and the appropriate offload pattern is selected. Therefore, the converted application is measured with the processing time, data amount, bandwidth used, calculation resource amount, and so on in each offload pattern, including the finally selected offload pattern. Those measured values when each application is offloaded are used in the proper placement calculation.

Conventionally, applications have been placed in the cloud, and data collected by IoT devices and the like has been transferred to a cloud server, where the data has been aggregated and analyzed. However, with the keywords edge computing and fog computing, there are increasing attempts to speed up the response time of applications by processing executions in the user environment or network edges that require real-time response.

In this paper as well, I will consider applications on the premise that they can be placed not only in the cloud but also at the network edge and user edge. However, at the network edge and user edge, the server concentration is lower than in the

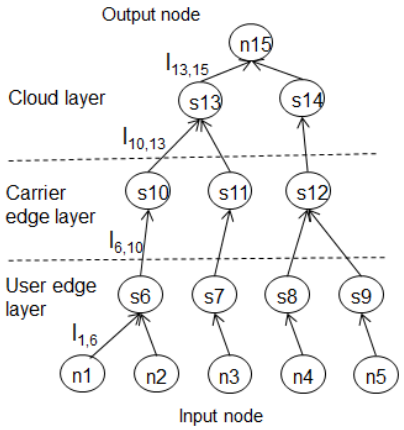

Figure 3. Network topology example

cloud and distributed, so the cost of computing resources is higher than in the cloud. Generally, the price of hardware such as CPU and GPU is constant regardless of the location. However, in a data center that operates the cloud, it is possible to monitor and control the air conditioning of the aggregated servers collectively, so the operating cost is cheaper.

For example, as a simple topology of the computational node link, Figure 3 can be considered. Fig. 3 shows the topology used in an example where an IoT device that collects data in a user environment sends data to the user edge, data is sent to the cloud via the network edge, and the analysis results are viewed by managers. Computational nodes are divided into three types: CPU, GPU, and FPGA. A node equipped with a GPU or FPGA also has a CPU, but it is provided as a GPU instance or an FPGA instance that includes CPU resources by virtualization technology.

Applications are located on the cloud, network edge, and user edge, and the closer to the user environment, the lower the response time, but the higher the cost of computational resources. In this paper, I will deploy the converted application for GPU and FPGA, but when deploying, the user can make two types of requests. The first is a cost requirement, which specifies an acceptable price for operating the application, for example, operating it within 50 USD per month. The second is a response time request, which specifies an allowable response time when operating an application, for example, returning a response within 10 seconds.

In the conventional facility design, as considered in [29], for example, the location where the server accommodating the virtual network is placed is decided by analyzing at the long-term trend such as the amount of traffic increase.

On the other hand, this paper's target has two features. The first is that the application to be placed is not statically determined but is automatically converted for GPU and FPGA, and the pattern suitable for the usage pattern is extracted through actual measurement through GA or other methods, so the application code and performance can change dynamically. For example, if the same Fourier transform program is used with large data for user A and small data for user B, the loop statements offloaded to the GPU might be different and the performance might be 10 times higher than only CPU processing for A and 5 times higher for B. Second, it is necessary to not only reduce carrier facility costs and overall response times but also meet individual user requirements for costs and response times, and application placement policies can

change dynamically.

On the basis of these two features, when there is a placement request from the user, the application placement in this paper is to sequentially place the converted applications on the server at that time to meet the user's request. If the cost performance does not improve even after converting the application, my method places applications without conversion. For example, a GPU instance costs twice as much as a CPU instance, and if the converted application does not improve performance more than twice, it is better to place an application without conversion. Also, if the computing resources and bandwidth have already been used up to the upper limit, they cannot be placed on that server.

3.3. Linear programming equations for appropriate application placement

In this subsection, I formulate a linear programming equation for calculating the appropriate location of the application. The parameters are shown in Figure 4.

Here, since the cost of devices and links, the upper limit of calculation resources, and the upper limit of bandwidth depend on the server and network prepared by the operator, the parameter values are set in advance by the operator. The amount of computational resources, bandwidth, data capacity, and processing time used by the application when offloaded are determined by the measurements in the final selected offload pattern in the verification environment during automatic conversion. These values of the application are automatically set by the environment adaptation function.

The objective function and constraints change depending on whether the user request is a cost request or a response time request. If the request requires the placement price within a month due to cost requirements, the minimization of the response time in (1) becomes the objective function, and the cost within (2) becomes one of the constraints. The constraint conditions that the resource upper limit of the server in (3) and (4) are not exceeded are also added. If the response time request requires the application to be placed within a number of seconds, the objective function is to minimize the cost of (5) corresponding to (2). The response time of (6) corresponding to (1) is one of the constraints within a number of seconds, and the constraints of (3) and (4) are also added.

(1) and (6) are equations for calculating the response time R_k of the application k, the objective function in the case of (1), and the constraint condition in the case of (6). (2) and (5) are equations for calculating the price P_k for operating the application k, the constraint condition in the case of (2), and the objective function in the case of (5). (3) and (4) are constraint conditions for setting the upper limit of the calculated resource and the communication band, which is calculated including the application placed by other users and checks the resource upper limit from being exceeded due to the application placement of the new user.

The linear programming equations (1)-(4) and (3)-(6) can be applied to conditions of network topology, conversion application type (cost and performance compared to only CPU processing), user requirements, and existing applications. By deriving a solution with a linear programming solver such as GLPK (Gnu Linear Programming Kit) or CPLEX (IBM Decision Optimization), an appropriate application location can be calculated. By sequentially performing the actual placement for multiple users after the appropriate placement calculation, multiple applications are placed on the basis of the requests of each user.

a_i: Device usage cost
b_j: Link usage cost
C_i^d: Device calculation resource limit of #i
C_j^l: Link bandwidth limit of #j
C_k: Data size of #k application
$A_{i,k}^d$: Whether to use of #k application on #i device
$A_{j,k}^l$: Whether to use of #k application on #j link
B_k^d: Calculation resource of #k application
B_k^l: Bandwidth usage of #k application
$B_{i,k}^p$: Processing time of #k application on #i device

Figure 4. Parameters of linear programming equations

$$R_k = \sum_{i \in Device} (A_{i,k}^d \cdot B_{i,k}^p) + \sum_{j \in Link} (A_{j,k}^l \cdot \frac{C_k}{B_k^l}) \tag{1}$$

$$\sum_{i \in Device} a_i (\frac{A_{i,k}^d \cdot B_k^d}{C_i^d}) + \sum_{j \in Link} b_j (\frac{A_{j,k}^l \cdot B_k^l}{C_j^l}) \leq P_k \tag{2}$$

$$\sum_{k \in App} (A_{i,k}^d \cdot B_k^d) \leq C_i^d \tag{3}$$

$$\sum_{k \in App} (A_{j,k}^l \cdot B_k^l) \leq C_j^l \tag{4}$$

$$P_k = \sum_{i \in Device} a_i (\frac{A_{i,k}^d \cdot B_k^d}{C_i^d}) + \sum_{j \in Link} b_j (\frac{A_{j,k}^l \cdot B_k^l}{C_j^l}) \tag{5}$$

$$\sum_{i \in Device} (A_{i,k}^d \cdot B_{i,k}^p) + \sum_{j \in Link} (A_{j,k}^l \cdot \frac{C_k}{B_k^l}) \leq R_k \tag{6}$$

4. Evaluation

On the basis of the equations of the linear programming, I demonstrate that multiple applications are properly arranged by using the solver GLPK by changing some conditions.

4.1. Evaluation method

4.1.1. Evaluated applications

The evaluated applications are the Fourier transform and image processing, which are expected to be used by many users.

The Fast Fourier Transform (FFT) is used in various situations of monitoring in IoT such as analysis of vibration frequency. NAS.FT [31] is one of the open source applications for FFT processing. It calculates the 2,048 * 2,048 size of the built-in sample test. 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.

MRI-Q [32] computes a matrix Q, representing the scanner configuration for calibration, used in 3D MRI reconstruction algorithms in non-Cartesian space. In an IoT environment, image processing is often necessary for automatic monitoring from camera videos, and performance enhancements are requested in many cases. MRI-Q is a C language application and during application performance measurement, MRI-Q executes 3D MRI image processing to measure processing time using 64*64*64 size sample data.

From my previous automatic GPU and FPGA offloading methods [19][20], NAS.FT can be accelerated by GPU, and MRI-Q can be accelerated by FPGA. Performance improvements compared with CPU are 5 and 7 times, respectively.

4.1.2. Experimental conditions

The topology for arranging applications is composed of 3 layers as shown in Fig. 3, with 5 sites in cloud layers, 20 sites in carrier edge layers, 60 sites in user edge layers, and 300 input nodes. Assuming an application of IoT, IoT data is collected from the input node at the user edge, and analysis processing is performed at the user edge or carrier edge or cloud in accordance with the response time requirements of the application.

All the servers to be analyzed are the assets held by a single operator, and the upper limit and price of the server and link are decided by the operator. In this evaluation experiment, the author decided on the following policy. In terms of servers, there are 8 CPU servers, 4 GPU servers with 16GB RAM and 2 FPGA servers in the cloud, 4 CPU servers, 2 GPU servers with 8GB RAM, and 1 FPGA server in the carrier edge, and 2 CPU servers and 1 GPU server with 4GB RAM in the user edge. Regarding server cost, for CPU, GPU, FPGA servers, it is assumed that 6,000, 12,000, and 14,400 USD will be collected in one year as the standard price of the servers in the cloud. When all resources of one server will be used (when using 16GB RAM for GPU server), the monthly fee is 500, 1,000, and 1,200 USD. Due to the aggregation effect, the monthly fee is 1.25 and 1.5 times that of the cloud, assuming that the carrier edge and user edge will be expensive.

For links, a bandwidth of 100 Mbps is secured between the cloud and the carrier edge, and a bandwidth of 30 Mbps is secured between the carrier edge and the user edge. For the link cost, referring to the price of OCN Mobile One Full MVNO for IoT services (data transfer amount up to 500MB costs 5 USD per month, up to 1GB costs 8 USD per month, etc.), and I set prices that 100Mbps link fee is 80 USD per month, 30Mbps link fee is 50 USD per month.

As the resource used by the application, the value when actually offloaded to GPU or FPGA is used for the processing time. NAS.FT uses GPU 1GB RAM, usage band

2Mbps, transfer data amount 0.2MB, and processing time 5.8 seconds. MRI-Q uses 10 % of the FPGA server (the number of Flip Flop and Look UP Table used is the FPGA resource usage), the usage band is 1 Mbps, the transfer data amount is 0.15 MB, and the processing time is 2.0 seconds.

The experiment deploys up to 1,000 applications on the basis of user requirements with set parameter values. The application is an IoT application and is supposed to analyze the data generated from the input node. Each input node generates placement requests randomly. Request number to place the applications is 1000 times at a ratio of NAS.FT:MRI-Q = 3:1.

As a user request, a price condition or a response time condition is selected for each application when requesting placement. In the case of NAS.FT, the monthly upper limit of 70, 85 or 100 USD is selected for the price, and the 6, 7 or 10 second upper limit is selected for the response time. In the case of MRI-Q, the monthly upper limit of 125 or 200 USD is selected for the price, and the 4 or 8 second upper limit is selected for the response time. There are three patterns as variations of user requests.

Pattern 1 (P1): 6 types of requests by 1/6 for NAS.FT and 4 types of requests by 1/4 for MRI-Q.

Pattern 2 (P2): It selects the condition with the lowest price as the upper limit (initially 70 USD for NAS.FT, 125 USD for MRI-Q), and if there is no vacancy, the next cheapest price condition is selected.

Pattern 3 (P3): It selects the condition that the shortest time is the upper limit (initially 6 seconds for NAS.FT, 4 seconds for MRI-Q), and if there is no vacancy, the next shortest time condition is selected.

4.1.3. Experimental tool

The placement is performed by a simulation experiment using the solver GLPK 5.0. It will be a simulation using tools to simulate a large-scale network layout. In actual use, when an application offload request comes in, an offload pattern is created by repeated performance tests using a verification environment, an appropriate amount of resources is determined on the basis of the performance test results in the verification environment. Then appropriate placement is determined using GLPK or other solvers to meet user requests. After production placement, a normal confirmation test and performance test are automatically performed, the result and price are presented to the user, and use is started after the user makes a judgment.

4.2. Results

In Fig. 5, the average price and the number of application placements are taken for three patterns, and in Fig. 6, the average response time and the number of application placements are taken for three patterns.

It was found that patterns 2 and 3 fills in order from the cloud and from the edge, respectively. In pattern 1, when various requests come in, they are arranged so as to satisfy the user requirements.

Regarding Fig. 5, in pattern 2, up to about 400 applications are placed in the cloud and the average price remains the lowest, but when the cloud is filled, it will gradually increase. In pattern 3, NAS.FT is placed from the user edge and MRI-Q is placed from the carrier edge, so the average price is high, and as the edges are filled, applications are also placed in the cloud, so the average price becomes low. For pattern 1, the average price is in the middle of patterns 2 and 3 and is arranged in accordance with

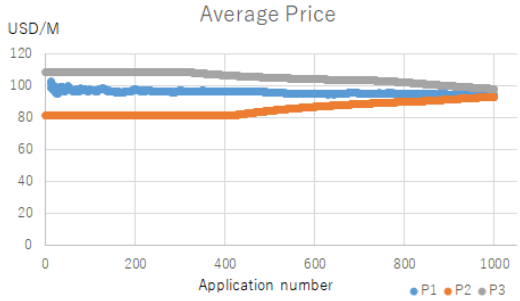

Figure 5. Average price change with the number of application placements

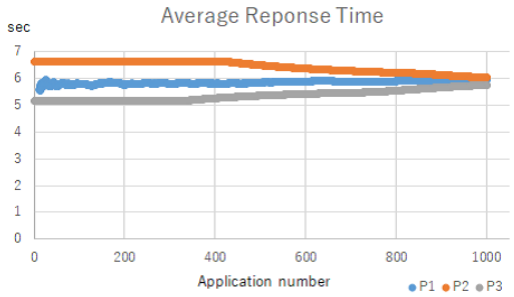

Figure 6. Average response time change with the number of application placements

the user's request, so the average price is appropriately reduced compared with pattern 3, which initially fills the edge.

Regarding Fig. 6, in pattern 2, up to about 400 applications are placed in the cloud and the average response time remains the highest, but when the cloud is filled, it gradually decreases. In pattern 3, NAS.FT is placed from the user edge and MRI-Q is placed from the carrier edge, so the average response time is the shortest, but as the number increases, applications are also placed in the cloud, so the average response time becomes higher. Regarding pattern 1, the average response time is between patterns 2 and 3 and is arranged in accordance with the user's request, so the average response time is appropriately reduced compared with pattern 2, which initially fills the cloud.

4.3. Discussion

The offload of the loop statement to the GPU in the previous research is a method of measuring the performance of multiple offload patterns in a verification environment and making it a high-speed pattern. For example, even Darknet, which is a large-scale application with more than 100 for statements, automatically offloads to the GPU and has been tripled in speed. However, not all users can use GPUs and FPGAs abundantly in the user environment, so it was necessary to be able to use GPUs and FPGAs resources appropriately in response to user price requests among virtualized resources. The proposed method automatically converts to GPU and FPGA, and after satisfying price conditions and response time conditions, resources in appropriate locations can

be used, so good cost performance can be achieved.

Regarding the offloading effect with hardware price, GPU or FPGA board price is about several thousand USD. Therefore, a server with a GPU or FPGA board price is about two times as much as that for only CPU. In data center systems such as a cloud systems, initial cost of hardware, development, and verification costs are about a 1/3 of the total cost; electricity power and operation/maintenance costs are more than 1/3; and other expenses, such as service orders, are less than 1/3. Therefore, I think automatically converting and placing application appropriately with performances more than three times higher will have a sufficiently positive effect even though the hardware price is about two times higher.

My previous research on environment-adaptive software has been based on the premise of offloading on a server on the cloud, and even if the throughput is improved by offloading to a GPU or FPGA instance on the cloud, response time could not be improved enough because of transfer time to the cloud. The method proposed this time is environmental adaptation using resources of the entire network including not only the cloud but also the network edge and user edge, and applications with severe response times respond by automatically arranging them on the edge according to the users' request. In contrast, for applications with severe cost requirements, a server on the cloud may be used.

Another paper describes the optimization of the amount of resources. It calculates the appropriate resource ratio on the basis of the test case execution time in the verification environment and automatically sets the resource amount within the user price condition so that no devices becomes a bottleneck in CPU, GPU, and FPGA. By combining the optimization of the resource amount and the optimization of the placement of the method proposed this time, the converted application can be provisioned with better cost performance on the basis of the user's request.

5. Related work

Wuhib et al. studied resource management and effective allocation [33] on the OpenStack cloud. My method is a network wide resource management and effective allocation method including the cloud, but it focuses on appropriate offloading on heterogeneous hardware resources.

Regarding the optimal use of resources existing on the network, there is research to optimize the inserted position of VN (Virtual Network) for the servers on the network [29][34]. In these studies, the optimal placement of VN is determined in consideration of communication traffic. The main target of these studies is facility design, which is designed by looking at the amount of traffic increase. Specifically, the processing time and cost for each application, which is different for each user, and various placement environments such as edges and cloud servers are not taken into consideration. The purpose of this paper is to convert different applications for each user and arrange them appropriately in accordance with the user request as an new element of environment-adaptive software.

Some studies focused on offloading to GPUs [35][36][37]. Chen et al. [35] used metaprogramming and just-in-time (JIT) compilation for GPU offloading of C++ expression templates. Bertolli et al. [36] and Lee et al. [37] investigated offloading to GPUs using OpenMP. There have been few studies on automatically converting existing code into a GPU without manually inserting new directives or a new development model, which I target. The method of [38][39] searches for GPU offloading areas

and also uses GA to search automatically. However, its target is specific applications for which many GPU-based methods have been researched to accelerate, and a huge number of tunings is needed.

There have been many studies on FPGA offloading [40][41][42][43]. Liu et al. [40] proposed a method of offloading nested loops to an FPGA and found that nested loops can be offloaded with an additional 20 minutes of manual work. Alias et al. [41] proposed a method with which an HLS tool configures an FPGA by specifying C language code, loop tiling, and so on when using Altera HLS C2H. The method proposed by Sommer et al. [42] can be used to interpret OpenMP code and perform FPGA offloading. Putnum et al. [43] used a CPU-FPGA hybrid machine to accelerate a program with a slightly modified standard C language. These methods require manually adding instructions such as which parts to parallelize using OpenMP or other specifications.

There are many works to speed up by offloading to GPU, FPGA, many-core CPU, but efforts are needed such as adding instructions of OpenMP manually that specify parts to parallelize and offload. There are few works to automatically offload the code of existing applications that users use differently. In addition, only the conversion for offloading to GPU, FPGA, and many-core CPU in the typical usage of the application is examined, and the application location optimization for each user is not considered like this paper.

6. Conclusion

For a new element of my environment-adaptive software, in this paper, to respond to the user's cost request and response time request when automatically offloading to GPU or other devices, I proposed an application placement optimization method. Environment adaptive software adapts applications to the environments to use heterogeneous hardware such as GPUs and FPGAs appropriately.

The proposed method works after the program is converted and the amount of assigned resources is determined so that it can be processed by an offload device such as GPU. In the proposed method, first, the data capacity, the amount of calculation resources, the bandwidth, and the processing time of the application are set from the data of the performance test performed in the verification environment at the time of program conversion. Appropriate placement of applications is calculated on the basis of the linear programming formula from the values set for each converted application and the values set by operators such as the cost of servers and links set in advance. When deploying an application, one is a constraint and the other is an objective function based on a user-specified price or response time request. An appropriate allocation is calculated by the linear programming solver, and the proposed method presents the price and so on to the user when the resource is allocated at the calculated location, and the production use is started after the user consents.

For the applications automatically offloaded to GPU and FPGA, the price condition and response time condition requested by the user, the number of application placements were changed, then the appropriate placement was calculated by the proposed method, and the effectiveness of the method was demonstrated. In the future, I will consider not only calculating the proper placement at the beginning of actual use but also reconfiguring when there is a more proper placement or converted offload pattern even during operation.

Disclosure statement

The author declares no conflicts of interest associated with this manuscript.

Data availability statement

The author confirms that the data supporting the findings of this work are available within the manuscript.

Notes on contributors

Yoji Yamato received a B.S. and M.S. in physics, and a Ph.D. in general systems studies from the University of Tokyo in 2000, 2002, and 2009. He joined NTT in 2002, where he has been conducting developmental research on a cloud computing platform, an IoT platform and a technology of environment adaptive software. Currently, he is a distinguished researcher of NTT Network Service Systems Laboratories. Dr. Yamato is a senior member of IEEE and IEICE, and a member of IPSJ.

References

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao and D. Burger, "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, M. Aissaoui and M. Eleuldj, "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, "Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," Journal of Information Processing, Vol.25, No.1, pp.56-58, 2017.

[6] 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.

[7] Y. Yamato, Y. Nishizawa, S. Nagao and K. Sato, "Fast and Reliable Restoration Method of Virtual Resources on OpenStack," IEEE Transactions on Cloud Computing, DOI: 10.1109/TCC.2015.2481392, Sep. 2015.

[8] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

[9] 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.

[10] Y. Yamato, Y. Fukumoto and H. Kumazaki, "Security Camera Movie and ERP Data Matching System to Prevent Theft," IEEE Consumer Communications and Networking Conference (CCNC 2017), pp.1021-1022, Jan. 2017.

[11] Y. Yamato, Y. Fukumoto and H. Kumazaki, "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.
[12] Y. Yamato, Y. Fukumoto and H. Kumazaki, "Analyzing Machine Noise for Real Time Maintenance," 2016 8th International Conference on Graphic and Image Processing (ICGIP 2016), Oct. 2016.
[13] 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.
[14] P. C. Evans and M. Annunziata, "Industrial Internet: Pushing the Boundaries of Minds and Machines," Technical report of General Electric (GE), Nov. 2012.
[15] T. Sterling, M. Anderson and M. Brodowicz, "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.
[16] J. E. Stone, D. Gohara and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.
[17] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.
[18] J. Gosling, B. Joy and G. Steele, "The Java language specification, third edition," Addison-Wesley, 2005. ISBN 0-321-24678-0.
[19] Y. Yamato, T. Demizu, H. Noguchi and M. Kataoka, "Automatic GPU Offloading Technology for Open IoT Environment," IEEE Internet of Things Journal, DOI: 10.1109/JIOT.2018.2872545, Sep. 2018.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] Xilinx SDK web site, https://japan.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/lyx1504034296578.html
[26] S. Wienke, P. Springer, C. Terboven and D. an Mey, "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.
[27] M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.
[28] E. Su, X. Tian, M. Girkar, G. Haab, S. Shah and P. Petersen, "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.
[29] C. C. Wang, Y. D. Lin, J. J. Wu, P. C. Lin and R. H. Hwang, "Toward optimal resource allocation of virtualized network functions for hierarchical datacenters," IEEE Transactions on Network and Service Management, Vol.15, No.4, pp.1532-1544, 2018.
[30] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.
[31] NAS.FT website, https://www.nas.nasa.gov/publications/npb.html
[32] MRI-Q website, http://impact.crhc.illinois.edu/parboil/
[33] F. Wuhib, R. Stadler, and H. Lindgren, "Dynamic resource allocation with management objectives - Implementation for an OpenStack cloud," In Proceedings of Network and service management, 2012 8th international conference and 2012 workshop on systems

virtualiztion management, pp.309-315, Oct. 2012.
[34] K. Kawashima, T. Otoshi, T. Ohshita and M. Murata, "Dynamic placement of virtual network functions based on model predictive control", IEEE/IFIP International Workshop on Analytics for Network and Service Management, Apr. 2016.
[35] J. Chen, B. Joo, W. Watson III and R. Edwards, "Automatic offloading C++ expression templates to CUDA enabled GPUs," 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, pp.2359-2368, May 2012.
[36] C. Bertolli, S. F. Antao, G. T. Bercea, A. C. Jacob, A. E. Eichenberger, T. Chen, Z. Sura, H. Sung, G. Rokos, D. Appelhans and K. O'Brien, "Integrating GPU support for OpenMP offloading directives into Clang," ACM Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM'15), Nov. 2015.
[37] S. Lee, S.J. Min and R. Eigenmann, "OpenMP to GPGPU: a compiler framework for automatic translation and optimization," 14th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP'09), 2009.
[38] Y. Tomatsu, T. Hiroyasu, M. Yoshimi and M. Miki, "gPot: Intelligent Compiler for GPGPU using Combinatorial Optimization Techniques," The 7th Joint Symposium between Doshisha University and Chonnam National University, Aug. 2010.
[39] Bobby R. Bruce and J. Petke, "Towards automatic generation and insertion of OpenACC directives," RN, 18.04: 04. 2018.
[40] Cheng Liu, Ho-Cheung Ng and Hayden Kwok-Hay So, "Automatic nested loop acceleration on fpgas using soft CGRA overlay," Second International Workshop on FPGAs for Software Programmers (FSP 2015), 2015.
[41] C. Alias, A. Darte and A. Plesco, "Optimizing remote accesses for offloaded kernels: Application to high-level synthesis for FPGA," 2013 Design, Automation and Test in Europe (DATE), pp.575-580, Mar. 2013.
[42] L. Sommer, J. Korinth and A. Koch, "OpenMP device offloading to FPGA accelerators," 2017 IEEE 28th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2017), pp.201-205, July 2017.
[43] A. Putnam, D. Bennett, E. Dellinger, J. Mason, P. Sundararajan and S. Eggers, "CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures," IEEE 2008 International Conference on Field Programmable Logic and Applications, pp.173-178, Sep. 2008.