Title	Study and Evaluation of FPGA Reconfiguration during Service Operation for Environment-Adaptive Software
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/10.1080/17445760.2023.2242639 Publication Date: yyyy/mm/dd ・出版日：2023 年 8 月 3 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 3rd Aug., 2023, available online: https://doi.org/10.1080/17445760.2023.2242639."

ORIGINAL PAPER

Study and evaluation of FPGA reconfiguration during service operation for environment-adaptive software

Yoji Yamato[a]

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

ARTICLE HISTORY
Compiled June 20, 2023

ABSTRACT
In recent years, more applications have been using not only CPUs with few cores but also heterogeneous hardware such as FPGAs, GPUs, and multi-core CPUs. However, to make full use of these, it is necessary to have technical skills for using hardware such as OpenCL, and this is a high barrier. To solve this problem, we have developed environment-adaptive software that enables high-performance operation by automatically converting application code written for normal CPUs by engineers in accordance with the deployed environment and setting appropriate amounts of resources. We also have verified its elemental technologies that automatically offload to FPGA and GPU. Until now, we only considered conversions and settings before the start of operation. In this paper, we verify that the logic is reconfigured in accordance with the usage characteristics during operation. Especially for FPGA logic, the example of reconfiguration during operation is not a commercial cloud, and this reconfiguration has a large impact, so we work on FPGA logic reconfiguration this time. We propose a FPGA reconfiguration method during operation and confirm that the application running on the FPGA is reconfigured into another application in accordance with the usage characteristics. Through confirmation, performance improvement and break time are checked, and the method is shown to be effective.

KEYWORDS
Environment Adaptive Software; Automatic Offloading; FPGA; Reconfiguration during Operation; Cost Performance

1. Introduction

In accordance with the deceleration prediction of Moore's Law, the use of a multi-core CPUs (Central Processing Unit), GPU (Graphics Processing Unit), and FPGA (Field Programmable Gate Array) has been increasing. 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]-[6]). In addition, the use of small devices such as IoT devices is increasing (e.g., [7]-[14]) as heterogeneous hardware.

However, to efficiently use heterogeneous hardware that is not a single-core CPU, programs need to be created and set in accordance with the hardware specification, which is a high barrier for most engineers. High knowledge and skill of OpenMP

CONTACT Yoji Yamato Email: yoji.yamato@ntt.com

(Open Multi-Processing) [15] for multi-core CPU, CUDA (Compute Unified Device Architecture) [16] for GPU, OpenCL (Open Computing Language) [17] for FPGA, assembly for IoT devices, or so on are often required.

To increase the use of heterogeneous hardware, we think that a platform is needed that enables even ordinary engineers without high-level skills or knowledge to make the best use of them. A platform analyzes software that describes processing with the same logic as a normal CPU, appropriately converts and sets it in accordance with 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 adaptation.

Therefore, we have developed environment-adaptive software that automatically converts, sets resources, determines the placement and other settings of the program code once written for normal CPU so that the GPU, FPGA, and 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, we have developed 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 [18]-[22].

However, our 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 there have been no studies on whether it is reconfigured in accordance with changes in usage characteristics after the start of operation. For example, before the start of operation, the logic for accelerating SQL processing was configured with FPGA on the premise that there were many SQL queries, but half a year after the start of operation, NoSQL queries greatly increased, so the logic for accelerating NoSQL processing was better to reconfigure the FPGA.

This paper focuses on the reconfiguration of FPGA logic, while reconfiguring software in accordance with the usage characteristics during operation. Except for special applications such as the circuit reconfiguration of an artificial satellite while using FPGA for accelerating normal applications, there is no example in the commercial cloud, which reconfigures FPGA logic in accordance with 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 changing the FPGA logic to another program, and examine and evaluate a method for reconfiguration with less user impact. The effectiveness of the proposed method is evaluated through the FPGA configuration of the existing application and the reconfiguration during operation. This paper extends and improves a short paper of the international conference CANDAR 2022 by a proposing, implementing, evaluating, and discussing the methods.

The structure of this paper is as follows. Section 2 outlines the existing technology and explains our previously proposed environment-adaptive software and the position of the reconfiguration during operation. In Section 3, we propose a method for reconfiguring FPGA logic in accordance with the usage characteristics after the start of operation. Section 4 explains the implementation of the proposed method. In Section 5, we evaluate the proposed FPGA reconfiguration method through offloading of existing applications and reconfiguration after the start of operation. Section 6 discusses related research. The paper is summarized in Section 7.

2. Existing technologies

2.1. Technologies in the market

GPGPU (General Purpose GPU), which uses the simple computing power of GPU for general calculation [23], has become widely used in recent years. NVIDIA provides CUDA as an environment for that purpose. OpenCL is a specification that handles heterogeneous hardware such as FPGAs and GPUs in common, not limited to GPUs, and many vendors have been supporting OpenCL. OpenCL and CUDA write programs using C language extensions. As an extended description, the transfer of memory information between the FPGA called the kernel and the CPU called the host is described, but it is said that more knowledge of hardware is required more for the original C language.

To make it easy to use heterogeneous hardware such as GPU even if we do not understand the specifications of OpenCL and CUDA, there is a technology to specify the part to perform GPU or multi-core CPU with a directive. The compiler creates binary files for GPUs and multi-core CPUs on the basis of the directives. There are compilers such as gcc and PGI [24] that interpret and execute specifications such as OpenMP and OpenACC [25].

By using OpenCL, CUDA, OpenMP, OpenACC, and so on, FPGA, GPU, and multi-core CPU can be used. However, even if that hardware can be used, the application performance is not easy to improve. For example, there is an Intel compiler [26], which automatically distributes processing to multiple cores of the multi-core CPU. At the time of automation, the Intel compiler finds a loop that can be processed in parallel in the loop of the program and lets multiple cores perform the processing. However, in many cases, the performance does not improve even if the loop is simply processed by multiple cores due to data copying or the like. It is more complicated because the memory is different for GPU and FPGA instead of multi-core CPU. Thus, to improve application performance, it may be necessary to tune by making full use of OpenCL and CUDA or search for appropriate GPU or other processing part using gcc with trial and error. Therefore, performance improvement using heterogeneous hardware requires high-level technical skills or trial-and-error searching.

As a GPU offload for loop statements, we have developed an offload using GA (genetic algorithm) [27], which is an evolutionary computation method, as an effort to automate the search for appropriate GPU processing parts in all loop statements. In FPGA, an application compile takes a long time and cannot be measured many times, so the loop statements that are candidates for offload are narrowed down on the basis of the arithmetic intensity of loop and the FPGA resource usage rate. For after narrowing down the candidate loop statements, we have developed a method to search for an appropriate offload pattern by making the loop statement with OpenCL and measuring it.

2.2. Environment-adaptive software

Previously, we proposed the overall processing in Fig. 1 as the processing of environment-adaptive software. The environment-adaptive software is processed by coordinating the production environment, verification environment, code pattern DB, facility resource DB, test case DB, and so on as platform functions, centering on the environment adaptation function provided by the operator.

Step 1: Code analysis

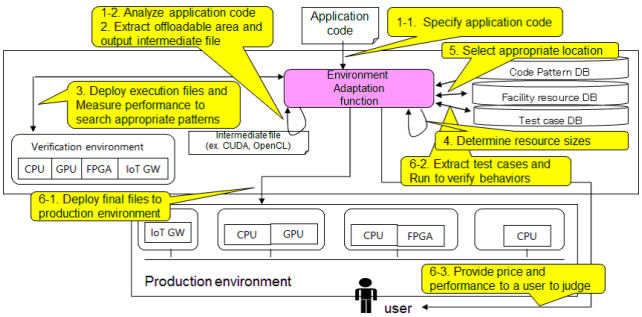


Figure 1. Overview of environment-adaptive software

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

Here, Steps 1-6 perform code conversion, resource-amount adjustment, placement-location adjustment, and verification, which are required before the application operation starts. In Step 7, after the application operation starts, the actual production usage characteristics are analyzed and appropriate reconfiguration is performed. The targets of the reconfiguration are code conversion, resource-amount adjustment, and placement-location adjustment as before the operation start.

We summarize the targets. Manual efforts are the mainstream for offloading to heterogeneous hardware. We previously proposed the concept of environment-adaptive software and verified the automatic GPU and FPGA offload method such as loop statements, but all of them were before the start of operation, and the reconfiguration after the start of operation, which corresponds to Step 7, has not been considered. Therefore, in this paper, we will focus on the method of reconfiguring FPGA offload logic in accordance with the usage characteristics during operation. There are many targets for reconfiguration, such as GPU, FPGA offload logic, resource amount, placement location, or so on, but FPGA logic is targeted in this paper. This is because there is no example of reconfiguring FPGA logic in accordance with usage characteristics even in a commercial cloud, thus it will have a large impact.

3. FPGA reconfiguration during operation

We have developed automatic GPU and FPGA offloading of loop statements and automatic adjustment of resource amounts and placements so far, to embody the concept of environment-adaptive software.

On the basis of these elemental technologies, 3.1 reviews the automatic FPGA offload of the loop statements before the operation starts. In 3.2, the basic policy for reconfiguration is considered to achieve an appropriate configuration according to the usage characteristics. In 3.3, we will examine the specific method of the FPGA reconfiguration during operation.

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

We review the automatic FPGA offload method of the loop statement that was verified in our previous paper [22].

This method first analyzes the source code to offload and grasps the information of the loop statement and variables.

Next, the method focuses on the candidate loops for trying the FPGA offload. The arithmetic intensity can be an indicator of whether the loop statement has an offload effect to judge. The arithmetic intensity is an index that increases when the number of calculations is large and decreases when the data size is large, and a high value may increase the load of CPU processing. Therefore, the arithmetic intensity of the loop statement is analyzed, and offload loop candidates are narrowed down by using intensity values. ROSE Framework [28] was used for arithmetic intensity analysis. In addition, loop statements with many loops also increase the load of CPU processing. The number of loops is analyzed by the profiler, and offload loop candidates are narrowed down using loop numbers. Gcov was used to analyze the number of loops.

It is a problem that the resources are excessively consumed when they are processed with FPGA, even for loop statements with high arithmetic intensity and loop numbers. Next, we check to calculate the amount of resources when the loop statement is processed on FPGA. When compiling to FPGA, the loop statement description is converted from high level languages such as OpenCL to hardware level language such as Hardware Declaration Language (HDL), and wiring circuits are performed on the basis of hardware level language. At this time, wiring processing takes a lot of time, but it does not take much time until HDL level. At the HDL level, the FPGA usage resource is known, so the amount of resource usage is known in a short time. By OpenCL language of the offload candidate loop statement, we can calculate the amount of resources, thus, the arithmetic intensity and the amount of resource usage are determined. Resource efficiency is the arithmetic intensity/resource-amount usage. In this method, a loop statement with high resource efficiency is further narrowed down as an offload candidate. Here, when a loop statement is converted to OpenCL language, the program of CPU processing is divided into a kernel (FPGA) and host (CPU) in accordance with the grammar of OpenCL.

Next, since some high resource efficient loop statements are narrowed down, patterns to measure performance are created using them. The method creates a certain number of offload patterns with narrowed loop statements and their combination and compiles to run them with FPGA. Finally, in the verification environment, the performance of multiple patterns is measured, and the highest speed pattern is selected as a solution.

In this way, for the automatic FPGA offload of the loop statement, high-speed patterns are explored through verification environment measurement by focusing on loop statements with high arithmetic intensity and loop numbers (see Fig. 2). At the time of the GPU, the combination was performed in most loop statements by GA, and 1000 scales were measured to explore the optimal pattern. In FPGA, it takes more than 6 hours to compile, so the number of measurements is narrowed down.

3.2. Basic policy for FPGA reconfiguration during operation

By using the method in 3.1, the application specified by the user can automatically offload the loop statements suitable for FPGA to FPGA.

After the offload to the production environment used by the user, the user can check the actual performance and price in the production environment and will start

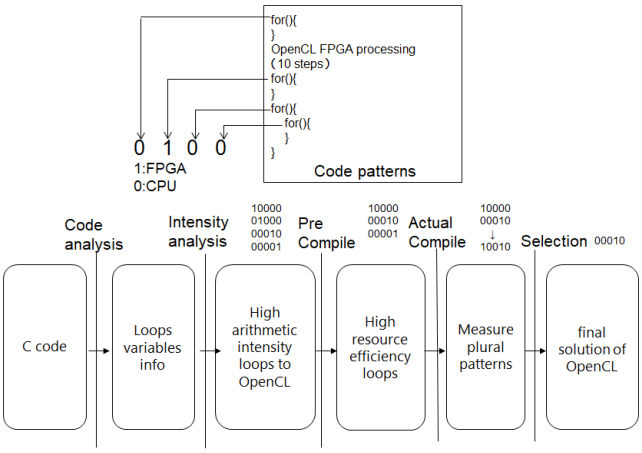


Figure 2. Automatic FPGA offload of loop statements

using the application or not. However, the performance optimization test case used in 3.1 (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. This may be greatly different from the data that will be used after the operation starts.

Therefore, in 3.2, 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, reconfiguration is considered to have a low user impact on FPGA logic. The reconfiguration may be changed to a different loop statement offload in the same application or may be changed to offload different applications.

There are two types of FPGA reconfiguration: dynamic and static. The former changes the circuit configuration while running the FPGA, and the time for rewriting is msec order. The latter 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, both methods have a break time and require a test for operation confirmation after reconfiguration. Therefore, we do not think that FPGA 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 the 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 in which the load is higher than or the same as 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 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 compared with the threshold value. If the improvement exceeds the threshold, a reconfiguration is proposed to the user. If the user accepts, the production environment is reconfigured. During reconfiguration, the production environment is reconfigured to reduce user impact as much as possible.

3.3. Method proposal of FPGA reconfiguration during operation

On the basis of the basic policy in 3.2, 3.3 proposes a concrete reconfiguration method. The reconfiguration method consists of six steps, and each step is explained in detail. Step 1 is particularly complicated, so we will add a supplementary explanation of 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 times with all applications are compared.

1-3. On the basis of 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 are created in accordance with the data size.

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 a verification environment measurement, which speeds up the test case of production representative data for multiple high load applications.

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

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

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

2-4. The highest speed offload pattern in four 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 are calculated on the basis of the frequency of production use.

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 whether 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 value is

checked to see if it 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 responds with OK or not OK.

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 with other applications that are processed in CPU only. When choosing representative data, the average data size may vary significantly from the actual production data, so we use the mode value of the data size.

4. Implementation

4.1. Tools to use

We explain the implementation to evaluate the effectiveness of the proposed technology. To evaluate the effectiveness of the FPGA reconfiguration, the targets are the applications of the C/C ++ language, and the FPGA uses Intel PAC D5005 (Intel Stratix 10 GX FPGA). The machine equipped with FPGA is Dell EMC Poweredge R740 (CPU: Intel Xeon Bronze 3206R * 2, RAM: 32GB RDIMM * 4).

For FPGA processing, I use Intel Acceleration Stack Version 2.0 (Intel FPGA SDK for OpenCL, Intel Quartus Prime). The Intel Acceleration Stack can conduct a High Level Synthesis (HLS), which interprets #pragma for Intel in addition to the standard OpenCL commands by coordinating two pieces of software. OpenCL code that describes the kernel program processed with FPGA and the host program processed by CPU is analyzed by Intel Acceleration Stack, and it outputs information such as the amount of resources and performs circuit setting work of FPGA, so that the OpenCL program can be run in FPGA.

For C/C++ language syntax analysis, we use LLVM/Clang 6.0 syntax analysis library (Python binding of libclang) [29].

In FPGA offloading, arithmetic intensity and the number of loops are used to narrow down candidates for statements. ROSE Compiler Framework 0.9 [28] is used to analyze arithmetic intensity, and gcov profiler is used to analyze the number of loops.

The implementation is implemented in Perl 5 and Python 3, and the following processing is performed. Perl focuses on the performance measurement of the offload pattern, and Python focuses on other processing such as syntax analysis.

4.2. Each step implementation point

Before the operation starts, the FPGA offload is performed with the same operation as the previous implementation tool [22]. The newly added reconfiguration method described in 3.3 is implemented to operate six steps in the configuration implementation in order: 1: load analysis, 2: new offload pattern extraction, 3: performance improvement calculation, 4: reconfiguration proposal judgment, 5: reconfiguration approval,

and 6: production reconfiguration.

Among them, the contents determined when implementation, which was not determined as a method, will be explained while supplementing the points.

First, in the load analysis in Step 1, the request data for a certain period (long term) is analyzed to determine the applications with the high load. Then, the actual request data for a certain period (short term) of the high load applications are selected as the representative data and used when extracting new offload patterns. Here, the number of high load applications is set by the operator. The operator also sets a certain period. A long span of one month or more is assumed for a long term and a short span such as 12 hours is assumed for a short term. In the request data analysis, the actual processing time and the number of times of use of the application are calculated and are acquired by the Linux Time command. Since the Time command logs the actual processing time of the application, the value can be calculated by the number of logs and summed actual processing time.

Next, in the representative data selection in Step 1, the actual request data for a certain period (short term) of high load applications is arranged for each fixed size to create a frequency distribution. In addition, the class number is determined by the Sturges formula. The Sturges formula is appropriate to set the number of classes to $1+\log_2 n$ when the application is used n times. After determining the number of classes and selecting the Mode class, one representative data needs to be selected from Mode class. When selecting representative data, the implementation selects the data whose data size is closest to the central value of the class as the representative data.

In Step 2, using the selected production representative data, FPGA offload is performed for the application with the high load by the same processing as before the start of operation. It is different from before the start of operation in that the test case used for performance measurement uses production representative data instead of assumed usage data.

In Step 3, the improvement effect needs to be seen when the production environment is reconfigured to the new offload pattern. Since it has not been proposed to the user for reconfiguration, it has to be verified in the verification environment server, but the improvement of each process is measured using the representative data of production use, and the degree of improvement of the whole is calculated by using the frequency of production use. By calculating, we will compare the effect when the production environment is reconfigured.

In Step 4, the proposal for reconfiguration is judged. No proposals are made if the improvement is below the threshold. Frequent proposals are inconvenient for the user, so by setting the threshold for improving the effect to a value sufficiently larger than one time, it is possible to suppress the frequent occurrence of proposals and leave a case of truly effective reconfiguration. The threshold can be set variably, but this time 2.0 is set.

In Step 5, at the time of the reconfiguration proposal, the price and the improvement effect in the verification environment after the reconfiguration are given, and the reconfiguration is proposed to the contract user. From this, the contract user can judge whether it is better to reconfigure or not.

In Step 6, when the reconfiguration is performed, the circuits are rewritten using OpenCL static reconfiguration, and an about one-second break time occurs. If we want to reduce the break time to the order of msec, we can use the dynamic reconfiguration such as the dynamic partial reconfiguration function of Intel FPGA.

5. Evaluation

5.1. Evaluation conditions

5.1.1. Evaluated applications

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

The tdFIR (time-domain Finite-Impulse Response filter) 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 [30] 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 to reduce network costs. Therefore, we think that the automatic speed-up of signal processing in FPGA has a wide range of applications.

MRI-Q [31] 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 needs 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 [32] for uncompressed fluid analysis, Symm (Symmentry matrix manipulation) [33] for symmetric matrix calculation, and DFT (Discrete Fourier Transform) [34] for discrete Fourier transform calculation are run on the same server and execution requests are received.

5.1.2. Evaluation methods

We evaluated 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, 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.

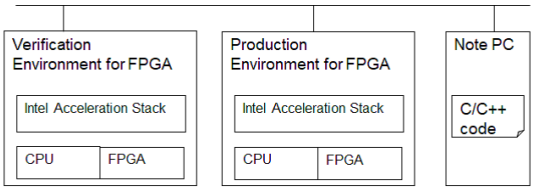
Narrow down of Arithmetic Intensity: Narrowing down to the top four loop statements in arithmetic intensity analysis

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

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

The operational conditions for FPGA reconfiguration are as follows.

Request frequency: tdFIR 200 req/h, MRI-Q 10 req/h, Himeno 3 req/h, Symm 2 req/h, DFT 1 req/h requests are applied for 3 hours.



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	

Figure 3. Evaluation environments

Types of data: For tdFIR and MRI-Q, three types of data are prepared. In tdFIR, 162 KB, 2.06 MB and 33.0 MB of sample data are used at a ratio of 75:120:5. In MRI-Q, 32*32*32 sample data, 64*64*64 sample data and double size of 64*64*64 sample data in which 64*64*64 data is copied and added are used at a ratio of 3:5:2. Himeno, Symm, and DFT only request the equipped sample data.

Long term during load analysis: 3 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.

5.1.3. 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 in accordance with the OpenCL syntax, FPGA offload processing is performed by OpenCL, and reconfiguration to another OpenCL program is also processed by Intel Acceleration Stack.

Figure 3 shows the evaluation environment and specifications. Here, the notebook 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 notebook 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.

5.2. Results

Figure 4 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

	Application	Improvement of processing time	Summation of processing time
Before reconfiguratoin	tdFIR	77.4 sec/h	644 sec
After reconfiguratoin	MRI-Q	238 sec/h	746 sec

Figure 4. Comparison of performance improvement through reconfiguration of the proposed method

the reconfiguration, the degree of improvement in the assumed data before the start of operation was 4.12, and the load of 200 req/h was applied after the start of operation. The total corrected processing time is 644 seconds, which was calculated from the total actual processing time of the request * 4.12, and the total number of uses is 600. Next, MRI-Q has a load of 10 req/h after the start of operation. The total corrected processing time is 746 seconds of the total actual processing time of the request, and the total number of uses is 30. These two are the applications with the high load. tdFIR and MRI-Q search for new offload patterns using production representative data. The processing time for one time in the new offload pattern is reduced from 0.511 to 0.124 seconds for tdFIR and from 26.0 to 2.21 seconds for MRI-Q. By multiplying the number of production uses, the processing time is reduced. The degree of performance improvement is 77.4 seconds/hour for tdFIR and 238 seconds/hour for MRI-Q.

From Fig. 4, the change from tdFIR offload to MRI-Q offload increases the performance improvement by 3.1 times and exceeds 2.0, so reconfiguration is proposed to the user.

The request analysis is small because only three hours of data is analyzed at this time, but it will take longer in proportion to the size. This time, it takes about only few seconds for request analysis and for production representative data selection, about one day for improvement effect calculation, and 2.7 seconds for reconfiguration. Regarding the trial of the new offload patterns searches during operation, and the time to compile the new offload pattern in the production environment, since one FPGA compilation takes six hours or more, the number of measurements is four, which takes more than a day for one application. Most 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 two seconds, 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.

We evaluated the FPGA reconfiguration in accordance with 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 the break time was sufficiently short.

5.3. Discussion

In our previous FPGA offload of loop statement, the performance is measured in the verification environment and the high-speed pattern is automatically searched for. By using Intel Arria 10 FPGA, tdFIR and MRI-Q can achieve several times higher performance automatically. Since FPGAs require programs that accord with the hardware characteristics and manual design is the mainstream, automatic offload can be said to be a major advance. This time, it can be said that resource efficiency of

FPGA is further advanced because the amount of resources is limited by reconfiguring to more appropriate logic in accordance with the usage characteristics during operation as well as before the start of operation.

The cost is considered. FPGA boards such as Intel Stratix cost about 3,000 USD each. Therefore, looking only at the hardware price, the price of a machine equipped with FPGA is usually twice that of a machine with only a CPU. However, in general, the cost of a data center is less than 1/3 of the cost of system development such as hardware and cloud, the operation cost of electricity cost and maintenance/operation system is more than 1/3, and the service order and other costs are about 1/3. Therefore, even if the hardware price itself is two times higher, this technology can improve the performance of loop statement processing, which used to take time by CPU processing, by two times or more and reconfigure it into appropriate logic. We think the cost effectiveness is good.

The time required for reconfiguration is considered. The analysis takes almost no time, but the time to search for new offload patterns varies greatly depending on the number of performance measurements. This is because it takes a long time to compile with FPGA, and the current situation is that it takes more than six hours * total number of measurements. However, this measurement time is the time required in the verification environment and is irrelevant to users in the production environment, so the effect can be said to be small. The only thing that affects the user is the break time when reconfiguring the production environment, but since it is about two seconds, the effect is still small. In summary, the verification time of one day or more and the break time of about two seconds are considered to be acceptable considering that the cost effectiveness will be improved by the reconfiguration.

The FPGA offload pattern search time can be shortened by preparing multiple FPGA servers and performing measurements in parallel. In addition, although the offload candidate loop statements are narrowed down on the basis of the arithmetic intensity and resource usage, it is also effective to reduce the number of narrowed-down measurements by the number of loops. The FPGA offload can be further speeded up by changing the logic to consider the algorithm of the entire function, rather than simply offloading the loop statement. For example, tdFIR loop statement offload improves performance by several times with Intel FPGA, but when the entire tdFIR function is offloaded to logic with OpenCL implemented with optimized algorithms, Intel Arria 10 FPGA improves performance by 21 times. Therefore, we will consider using such a function block offload together with the loop statement.

This time, we focused on the reconfiguration of FPGA logic in the reconfiguration during operation. Before starting operation, environment-adaptive software has verified automatic conversion to multi-core CPU, GPU, FPGA, optimization of processing resource amount, and optimization of placement location. Therefore, during operation reconfiguration, other optimizations such as GPU logic and the amount of resources can be possible, not just FPGA logic. We will also consider these various types of reconfigurations and aim to improve cost performance by reconfiguration during operation as a whole.

6. Related work

There is a method to improve resource efficiency on the cloud using OpenStack [35]. Our proposed method can also be said to be a resource efficiency technology by reconfiguration including cloud, but our method especially targets FPGA reconfiguration

in accordance with the usage characteristics during operation. This can be said to be a new attempt to reconfigure offload logic automatically.

As for the development environment of OpenCL that controls FPGA, Altera before it was acquired by Intel was released in the same way as Xilinx. For example, the Altera SDK for OpenCL [36] consists of the OpenCL C Compiler and the OpenCL Runtime Library, which describes the processing in OpenCL and enables FPGA processing. However, it is difficult to automatically offload certain logic to the FPGA to improve performances. On the other hand, many well-known processing patterns such as FFT (Fast Fourier Transform) may be implemented in OpenCL. Therefore, to utilize them, it is considered to prepare a replacement OpenCL in the DB for the well-known pattern, and when using the well-known processing, the method is studied that replaces the processing with the replacement OpenCL and offloads it.

There is a lot of research on FPGA offload [37][38][39][40]. Liu et al. [37] suggested a method for offloading nested loops into the FPGA and showed that the nested loops can be manually offloaded for an additional 20 minutes. Alias et al. [38] proposed a method for HLS tools to specify C language code, loop tie, and so on to configure FPGA when using Altera HLSC2H. The method proposed by Sommer et al. [39] can be used to interpret OpenMP code and perform FPGA offload. Putnum et al. [40] used a CPU-FPGA hybrid machine to accelerate the program in a slightly modified standard C language. These methods require manual addition of instructions such as parts to be parallelized using OpenMP or other specifications. Therefore, there is almost no research on automatically offloading or reconfiguring existing code to FPGA as in this paper.

SYCL [41] is a single-source programming model on heterogeneous hardware. In OpenCL, the host code and kernel code are written separately, but in SYCL, they can be written in a single source. DPC++ [42] is Intel's SYCL compiler. Both OpenCL and SYCL require new programs to use heterogeneous hardware. SYCL improves over OpenCL because it targets a single code that runs on multiple pieces of hardware, such as CPU and GPU or FPGA. However, the new SYCL single code needs to be written by the programmer. Therefore, the target is different from that in this paper, which does not require the manual creation of new code.

OpenCL basically controls kernel processing within a node, but when processing with an FPGA of multiple nodes, it is normal to perform processing between nodes using MPI (Message Passing Interface). However, MPI requires a high level of knowledge like OpenCL. Therefore, the MPI description itself does not need to be described, and MPI processing technology that uses the device of a different node by showing the device as a local node has come out [43]. When processing with FPGAs of multiple nodes, we are also considering the use of these MPI processing technologies.

As mentioned above, there are many works to speed up by offloading to FPGA, but the mainstream approach is to manually add instructions such as which part to parallelize and offload in accordance with it like OpenMP. There are few methods to automatically offload existing code. In addition, others are only considering conversion for offloading to FPGA before the start of operation. There is no study to improve cost performance by reconfiguring the FPGA logic to a new FPGA logic during operation in accordance with the usage characteristics as targeted in this paper.

7. Conclusion

We previously proposed environment-adaptive software for automatically adapting software to the deployment destination environment and appropriately using GPU, FPGA, and so on to operate applications with high performance. In this paper, as an element of it, we have proposed an FPGA reconfiguration method that reconfigures the appropriate FPGA logic during operation in accordance with 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 is calculated on the basis of the frequency of production use. 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 was automatically offloaded to the FPGA, and the FPFA 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 2.7 seconds. Thus, the proposed method was shown to be effective.

In the future, we will further expand the scope of reconfiguration during operation, and consider reconfiguring not only FPGA logic but also GPU logic and multi-core CPU logic, as well as other settings such as adjusting resource amount and placement. With appropriate reconfiguration during operation, we will evaluate cost performance improvement.

Disclosure statement

The author declares no conflicts of interest associated with this 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, "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.

[5] 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.

[6] 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.

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

[8] H. Noguchi, T. Demizu, N. Hoshikawa, M. Kataoka and Y. Yamato, "Autonomous Device Identification Architecture for Internet of Things," 2018 IEEE 4th World Forum on Internet of Things (WF-IoT 2018), pp.407-411, Feb. 2018.

[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] H. Noguchi, M. Kataoka and Y. Yamato, "Device Identification Based on Communication Analysis for the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2019.2910848, Apr. 2019.

[12] Y. Yamato, Y. Fukumoto and H. Kumazaki, "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] 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. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.

[17] 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.

[18] 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.

[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[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] 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.
[26] 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.
[27] J. H. Holland, "Genetic algorithms," Scientific american, Vol.267, No.1, pp.66-73, 1992.
[28] ROSE compiler framework web site, http://rosecompiler.org/
[29] Clang website, http://llvm.org/
[30] Time domain finite impulse response filter web site, http://www.omgwiki.org/hpec/files/hpec-challenge/tdfir.html
[31] MRI-Q website, http://impact.crhc.illinois.edu/parboil/
[32] Himeno benchmark web site, http://accc.riken.jp/en/supercom/
[33] Polybench symm website, https://web.cse.ohio-state.edu/ pouchet.2/software/polybench/
[34] DFT website, http://programming.blogo.jp/c/fourier_transform
[35] 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.
[36] Altera SDK for OpenCL web site, https://www.altera.com/products/design-software/embedded-software-developers/opencl/documentation.html
[37] 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.
[38] 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.
[39] 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.
[40] 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.
[41] SYCL web site, https://www.khronos.org/sycl/
[42] DPC++ web site, https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-library.html#gs.flx6xq
[43] A. Shitara, T. Nakahama, M. Yamada, T. Kamata, Y. Nishikawa, M. Yoshimi and H. Amano, "Vegeta: An implementation and evaluation of development-support middleware on multiple opencl platform," IEEE Second International Conference on Networking and

Computing (ICNC 2011), pp.141-147, 2011.