本論文は、2024 年 9 月公開済みの論文 "IoT アプリケーションへの自動ソフトウェア分割適用," の採録版のコピーである。

This paper is a copy of the accepted article "IoT application adopting for automatic software division," published in September 2024.

公開済み論文の書誌情報は、下記の通りである。
山登庸次, "IoT アプリケーションへの自動ソフトウェア分割適用," IECC 2024, pp.6-10, 2024 年 7 月

The bibliographic information of the published paper is as follows:
Y. Yamato, "IoT application adopting for automatic software division," 2024 6th International Electronics Communication Conference, pp.6-10, July 2024.

本論文が最初に公開された ACM は、本論文のプレプリントサーバ登録を認めているため、Jxiv において公開することについて許可されている。

The Association for Computing Machinery (ACM), where this paper was first published, accepted the registration of this paper on preprint servers, and therefore the permisson to publish it on Jxiv was given.

"(c) ACM 2024. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in ACM, https://dl.acm.org/doi/10.1145/3686625.3686627"

IoT application adopting for automatic software division

YOJI YAMATO*

*Network Service Systems Laboratories, NTT Corporation, Japan

yoji.yamato@ntt.com

We have proposed the concept of environment-adaptive software that automatically converts and operates program code so that it can appropriately utilize the environment in which it is placed. This paper applies an automatic dividing method to IoT services as an element of environment-adaptive software, thereby making it easier for users to customize their own services. We confirm that the sample application can be automatically divided 300 lines out of the total 480 lines and easily to add and change functions for the right materials in the right places.

CCS CONCEPTS • Software and its engineering • Software creation and management • Software development technique

Additional Keywords and Phrases: Environment-adaptive software, IoT services, Function addition, Automatic division, Dynamic analysis

1 INTRODUCTION

Recent progress in AI (Artificial Intelligence) has led to heterogeneous hardware such as not only CPUs (Central Processing Units), but also GPUs (Graphics Processing Units), FPGAs (Field Programmable Gate Arrays), and IoT (Internet of Things) devices ([1]-[10]) now being used in many applications. Amazon and Microsoft [11] provide and utilize cloud technology (e.g., [12]-[17]) for GPU and FPGA processing. However, to use heterogeneous hardware, a program that is aware of the hardware characteristics is required, which is a high barrier for most engineers. Knowledge of CUDA (Compute Unified Device Architecture) [18] is required for GPUs, OpenCL (Open Computing Language) [19] for FPGAs, and IoT PF (Platform) knowledge is often required for IoT devices.

To make better use of heterogeneous hardware, we believe that we need a PF that allows even ordinary software engineers without advanced knowledge to make the most of it. A PF analyzes software whose processing is written in the same way as simple programs, converts it appropriately, configures it in accordance with the environment in which it is applied (GPU, FPGA, IoT devices, etc.), and operates it in a way that adapts to the environment. In the future, this will be required to be done automatically by the PF.

Therefore, we proposed the concept of environment-adaptive software that automatically converts and operates code normally intended for the CPU so that it can appropriately utilize the environment in which it is placed. [20]-[24] proposed a method for automatically offloading code to GPUs and FPGAs as environment-adaptive software elements. In this year for IoT devices, we have also proposed a method for automatically building IoT services by allowing users to select a basic service, specify their own processing, and specify the IoT devices to be used. In addition, to allow users to add and change the processing they want to perform even in general-purpose programs, by dividing the application on the basis of related processing and localizing changes on the basis of the division boundaries, the right material can be used in the right place. This makes it easier to add and change services.

However, the automatic division method verification has been limited to C language computational applications so far. Therefore, in this paper, by applying the automatic division method to IoT services, we will make it easier for users to customize their own IoT services, which previously required the provider to provide basic services in advance. We will automatically divide the temperature display sample application and confirm that it can be easily changed to operate in the right place.

2 EXISTING TECHNOLOGIES

2.1 Commercial technologies and environment-adaptive software

NVIDIA provides CUDA in the GPGPU (General-Purpose GPU) environment, which uses GPUs for general calculations [25]. OpenCL is a specification that commonly handles heterogeneous hardware such as FPGAs and GPUs. To easily use GPU or other heterogeneous hardware, there are efforts to create binary files for GPU etc. processing by specifying lines for GPU etc. processing with directives, such as OpenMP (Open Multi-Processing) [26] and OpenACC (Open Accelerators) [27] specifications. There are compilers such as gcc and PGI [28] that interpret and execute the specifications. Although these methods make it possible to use heterogeneous hardware, performance is currently difficult to improve due to the effects of data copying. GPUs and FPGAs are more complex because they have different memories, and manual tuning using OpenCL and CUDA is required to improve performance. Among them, we propose automatic offloading using GA (Genetic Algorithm), which is an evolutionary computing method.

Heterogeneous hardware is also used in IoT devices, which often have terminals with small amounts of resources such as memory, rather than accelerators such as GPUs. The M2M platform oneM2M has been standardized as an IoT standard. Sensing data from IoT sensors is aggregated by IoT GW (Gateway) and sent to the cloud using protocols such as MQTT (Message Queue Telemetry Transport) and HTTP. Common processing such as data aggregation and storage is performed by IoT PF on the cloud, and processing is performed, and the calculation results are often displayed on a cloud server to users such as corporate executives. As for IoT PF, major cloud providers Amazon and Microsoft provide IoT PF such as AWS IoT and Azure IoT in the cloud, which has become close to the de facto standard.

Previously, we proposed the overall picture shown in Fig. 1 as a process for environment-adaptive software. Processing of environment-adaptive software is as follows: Steps 1-6 are code analysis, code conversion according to the deployment environment, resource amount adjustment, deployment location adjustment, verification, and start of application operation. In Step 7, after the application starts operating, it

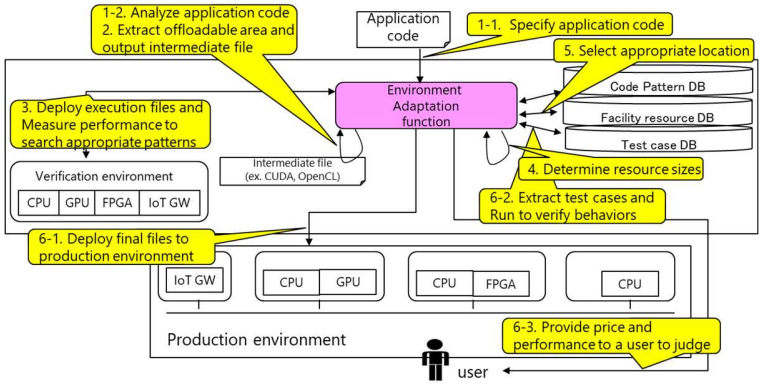


Figure 1: Overview of environment-adaptive software.

analyzes the actual usage characteristics and performs the necessary reconfiguration. We have verified the flow of Steps 1-7 using both accelerators such as GPUs and FPGAs, and low-resource devices such as IoT devices. Furthermore, to increase the number of applications that can be code analyzed, we propose an automatic division method that localizes and facilitates changes by dividing the application on the basis of related processes when the user wants to add or change functions of applications.

2.2 Summarize the issues of this paper

We clarify the issues of this paper. We previously proposed the concept of environment-adaptive software, including a method that automatically offloads programs to accelerators such as GPUs and FPGAs, and an IoT adaptation method where users can automatically build IoT services by specifying the basic service, the computational processing code they want to process, and IoT devices. We have also verified a method that makes it easier for users to add or change functions by automatically dividing general-purpose programs on the basis of the code processing relationships and localizing changes.

However, automatic division of general-purpose programs has so far only been verified for C language calculation applications, and new services such as IoT that are expected to be used by many users have not been verified and their effectiveness has not been demonstrated. Therefore, this paper aims at automatic division in IoT services and slightly changes the method on the basis of considerations unique to IoT. We implement and verify that we can analyze with IoT services that are also used in Azure IoT PF.

3 APPLYING GENERAL-PURPOSE PROGRAM AUTOMATIC DIVISION TO IOT SERVICES

3.1 Automatic division method for IoT

When we consider adding and changing unique processing that the user wants to perform to the program, for applications with a certain number of lines, changes can have a wide-ranging impact, so additional changes require a lot of work to check to see if related functions are affected. Therefore, by dividing the application into related processes and localizing changes on the basis of the division boundaries, additional changes can be made easily.

Among the analysis methods, there is a static analysis method that looks at relationships such as function calls and writes in source code without actually running the application. Static analysis first allows us to understand the calling relationships between functions. Next, we can also figure out whether certain functions write to the same data. In general, functions that have a calling relationship or that use them to write to the same data are closely related, so they need to be grouped when they are divided.

There is a dynamic analysis method that uses sample test cases to actually run an application and view information such as execution logs. Dynamic analysis can extract the calling relationships between functions that are actually used in sample test cases specified by the user. Furthermore, in the case of applications that use databases (DB), it is possible to extract the DB access log, discover data access instructions that are being executed continuously, and find the program range that is being executed as a series of processes. Furthermore, even when accessing a file rather than a DB, we can find the program range that is being executed as a series of processes from the file access log.

When GPU automatic offloading is used for environment-adaptive software, static analysis can be used to determine whether calculation processing can be performed by the GPU, but the performance when processing with the GPU cannot usually be known unless it is actually measured, so we combine static analysis and dynamic analysis to search for appropriate offload parts automatically. Dynamic analysis was an important element to measure performance by actually running sample test cases used by users and to respond differently to each individual user. Therefore, we decided to use a method that combines static analysis and dynamic analysis to respond to individual users when automatically dividing IoT services. Since IoT services always require the aggregation of IoT data, especially when analyzing programs, processes that involve communication should be divided into separate groups and placed in separate casings.

The operation of the proposed method is shown in Fig. 2. We statically and dynamically analyze IoT service program codes and divide them on the basis of information about call relationships.

We suppose that file A defines functions a1, a2, a3, etc., and file B defines functions b1, b2, b3, etc.

In static analysis, if function a1 called from the Main function calls b2, AB is counted as once. The method counts it for all files. As a result, AB: 20, AC: 30, BA: 10, etc. are the static analysis results.

In dynamic analysis, as a result of performing a certain number of sample tests, if function a1 called from the Main function calls b2 10 times, AB will add 10 times. It is counted by the number of sample tests conducted. As a result, AB: 10, AC: 20, etc. are the dynamic analysis results.

As a result of dynamic analysis, files called at least once are considered to be in the same group. As a result of static analysis, files called three or more times are classified as the same group. As a result of static analysis, all files called twice or less are placed in the same group. As a result of static analysis, files that communicate even once are placed in a separate group.

By doing this, files that have a calling relationship in the sample test case specified by the user during dynamic analysis will always be in the same group. In addition, when all files are analyzed using static analysis, files called a certain number of times (three or more times) will be in the same group. Files for functions that are Stand Alone-like and have little interaction with other functions will be in the remaining group. Also, if there is communication related to IoT data aggregation, which is a feature of IoT, it will be in a separate group. By automatically dividing the application on the basis of calling relationships, the user can reduce the scope of confirmation when adding functions.

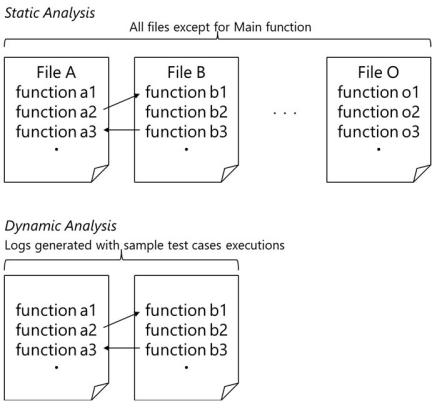


Figure 2: Analysis of automatic program division of IoT services.

3.2 Implementation

The targets for automatic division are C language and Python applications, Clang is used to analyze C language, and ast is used to analyze Python. The implementation to analyze the application is done in Python 3. The implementation receives source code files and sample test case information as input and outputs group information for static analysis and group information for dynamic analysis as output. Group information consists of divided groups and files belonging to them in the form of i (A, B, C), ii (D, E). In addition, the fixed number of times the sample test case was executed was set to 10 times.

4 EVALUATION

4.1 Evaluation application and evaluation conditions

The evaluation target is a Python application that is a basic IoT service that collects IoT data from environmental sensors, the IoT GW sends it in bulk to the IoT PF, and the results are displayed on the web on the IoT PF.

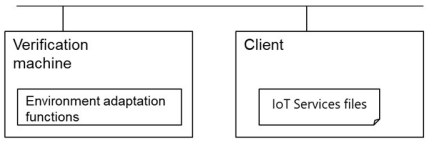
Sensors: Omron 2JCIE-BU01. These collect environmental information every minute: temperature, humidity, illuminance, atmospheric pressure, noise, 3-axis acceleration, eTVOC, discomfort index, heatstroke alert level, and vibration information.

IoT GW: Armadillo-IoT G3L.

IoT PF: Azure IoT. Data processing is performed via the REST API of Azure IoT Hub, and the results are passed to the same Azure Virtual Machine.

Program added by the user after automatic division: It calculates and outputs the average values of temperature and humidity.

Thus, divided groups for static analysis and dynamic analysis are displayed for IoT services, so we can check the division status and count and compare the total number of lines in the original application and the number of lines in the division application. Functionality is added to the divided application by using existing technology to build an IoT service that displays the average values of temperature and humidity on the web, and the display is confirmed manually.



Name	Hardware	CPU	RAM	GPU	OS
Verification Machine	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX (32 Core)	64GB	NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory : GDDR6 11GB)	Ubuntu 16.04.6 LTS
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro

Figure 3: Evaluation environment.

Application	Divided group #	total code #	target group code #
Show temperature	2	480	300

Figure 4: An example of automatic division using the proposed method.

Figure 3 shows the evaluation environment. The environment adaptation described in this paper does not speed up the process, so the tool can be run on any machine, but static analysis and dynamic analysis take a long time, so running on a machine with sufficient specifications is desired.

4.2 Result and Discussions

Figure 4 shows the number of divided groups, the number of lines in the entire application, and the number of lines in the groups to which new functions are added to the application when the temperature display application is automatically divided using the proposed method.

First, looking at the divided groups, the part that collects data from Omron sensors and the part that accumulates and displays sensor data are divided, and the functions can be placed separately in IoT GW and IoT PF. The temperature display application is divided into two parts, with a total of 480 lines, and the new function addition group has 300 lines. The automatic division reduces the range to be checked when the user adds unique processing such as calculating the average value of temperature or humidity, making it easier to add and change functions.

The automatic building of IoT services using IoT devices using our previous environment-adaptive software has been based on the premise that the operator has prepared the basic services in advance. This time, by making it possible to apply automatic division technology to IoT services, we have enabled users to add desired functions to IoT services that do not have basic services. With automatic division, IoT GW and IoT PF can be appropriately divided and placed, so we believe that the right materials can be used in the right places.

We consider costs. When thinking about adding functionality to IoT services, we used a Python application as a sample. The number of lines to check when adding a function has been reduced to 2/3. We believe that the impact confirmation operation will be lower and the modification cost can be reduced.

In this analysis, we combined static analysis, which has been commonly used, with dynamic analysis to respond to each user. The analysis itself uses function calling relationships, and we have confirmed that this

is sufficient for division. More detailed analysis becomes possible with additional information such as writing to the same data, access logs of databases and files, and continuous execution.

5 RELATED WORK

Regarding offloading to GPU, there are existing studies such as [29][30][31]. [29] used metaprogramming and JIT (Just In Time) compilation for GPU offloading of C++ expression templates, whereas [30][31] worked on GPU offloading using OpenMP. However, few methods automatically convert existing code for GPUs, which we are aiming for, without introducing a new development model or manually inserting directives.

There are [32][33][34] and other studies on FPGA offloading. [32] proposed a way to offload nested loops to the FPGA. [34] used a CPU-FPGA hybrid machine to speed up programs in slightly modified standard C language. These methods require manual addition of instructions, such as parts to be parallelized using OpenMP or other specifications.

[35][36][37][38] are IoT frameworks. [35] specializes in the development method of microservices and improves efficiency. [36] is a framework for smart government to improve efficiency. The IoT framework increases productivity, but it requires specialized knowledge according to specialized methods and targets, similar to development when using IoT PF of a major cloud provider. .

[39][40][41][42] are studies on automatic configuration in IoT. Implementation of Home GW is being considered for automatically setting MQTT devices and industrial IoT environments. Since the number of IoT devices, such as sensors and actuators, is much larger than before, the reduction of setting operation by automatic configuration has a great advantage. However, although the automatic configuration is being considered, there is no attempt to automatically build an IoT service that incorporates the processing that the user wants to conduct.

[43] analyzes large-scale business applications, divides them into functions, and visualizes them. Although the target is a large-scale business application, the aim of analyzing and dividing the application is similar to that of our technology. The technology performs clustering by understanding the call destination of the program and the group of programs to be written and weighting the relationships. Furthermore, if access logs to the DB can be used, the range to be executed as a series of processes is found and the clustering results are corrected. The difference is that it targets large-scale business applications such as ERP (Enterprise Resource Planning) and uses a lot of information such as DB access logs for dividing. Our technology uses simple information for analysis so that even small-scale applications that do not use a DB or applications that do not require persistence can be divided and have functionalities added.

Regarding offloading to GPUs and FPGAs, the mainstream is to manually add instructions for OpenMP, OpenCL, etc. and offload accordingly, and there are almost no methods to automatically offload existing code. The same is true for IoT adaptation, and although frameworks and automatic configuration have been researched, there are no methods that can be easily used even by users without knowledge of IoT PF, etc. Regarding the addition of functions by users to general-purpose programs other than IoT, there have been studies on large-scale business applications, but no methods have been developed to facilitate the addition of functions to small-scale applications used by general users.

6 CONCLUSION

In this paper, we applied the automatic division method to IoT services as an element of environment-adaptive software that automatically adapts software to the environment and operates applications, so that users can easily customize their own services.

The proposed method grasps the calling relationships between files that contain functions by analyzing both the static state in which the program is not running and the dynamic state in which it is running. It automatically proposes divisions between groups of files that have no calling relationship and are not related and divisions between groups of files that communicate with each other. With the proposed method, the number of lines in the divided file group is smaller than that in the original file group, and the scope of checking the change impact when adding original processing that the user wants to perform becomes smaller. As a sample application, we applied the proposed method to an IoT service that displays temperature, and it is divided into IoT GW and IoT PF, allowing the use of the right materials in the right places. The new function addition group has 300 lines out of the total 480 lines, making it easy to add and change functions.

In the future, we will apply the proposed automatic program division method to other types of services and evaluate its effectiveness with various types of services.

REFERENCES

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

[2] H. Noguchi, et al., "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.

[3] Y. Yamato, et al., "Proposal of Real Time Predictive Maintenance Platform with 3D Printer for Business Vehicles," International Journal of Information and Electronics Engineering, Vol.6, No.5, pp.289-293, 2016.

[4] Y. Yamato, "Proposal of Automatic GPU Offloading Method from Various Language Applications," The 9th International Conference on Information and Education Technology (ICIET 2021), pp.400-404, Mar. 2021.

[5] Y. Yokohata, et al., "Service Composition Architecture for Programmability and Flexibility in Ubiquitous Communication Networks," IEEE International Symposium on Applications and the Internet Workshops (SAINT 2006), pp.142-145, Jan. 2006.

[6] H. Sunaga, et al., "Service Delivery Platform Architecture for the Next-Generation Network," ICIN 2008, Oct. 2008.

[7] M. Takemoto, et al., "Service Elements and Service Templates for Adaptive Service Composition in a Ubiquitous Computing Environment," The 9th Asia-Pacific Conference on Communications (APCC 2003), Vol.1, pp.335-338, Sep. 2003.

[8] Y. Yokohata, et al., "Context-Aware Content-Provision Service for Shopping Malls Based on Ubiquitous Service-Oriented Network Framework and Authentication and Access Control Agent Framework," IEEE Consumer Communications and Networking Conference (CCNC 2006), pp.1330-1331, Jan. 2006.

[9] Y. Nakano, et al., "Method of Creating Web Services from Web Applications," IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07), pp.65-71, June 2007.

[10] Y. Yamato, et al., "Study of Service Processing Agent for Context-Aware Service Coordination," IEEE International Conference on Service Computing (SCC 2008), pp.275-282, July 2008.

[11] Y. Yamato, et al., "Context-Aware Ubiquitous Service Composition Technology," The IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006), pp.51-61, Apr. 2006.

[12] H. Noguchi, et al., "Device Identification Based on Communication Analysis for the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2019.2910848, Apr. 2019.

[13] H. Noguchi, et al., "Distributed Search Architecture for Object Tracking in the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, Oct. 2018.

[14] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA'14), 2014.

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

[16] Y. Yamato, "Automatic Verification for Plural Virtual Machines Patches," The 7th International Conference on Ubiquitous and Future Networks (ICUFN 2015), pp.837-838, July 2015.

[17] Y. Yamato, et al., "Fast Restoration Method of Virtual Resources on OpenStack," IEEE Consumer Communications and Networking

Conference (CCNC 2015), pp.607-608, Jan. 2015.

[18] J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, ISBN: 0131387685, 2010.

[19] J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science \& engineering, Vol.12, No.3, 2010.

[20] 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.

[21] 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.

[22] 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.

[23] Y. Yamato, "Proposal of Automatic Offloading for Function Blocks of Applications," The 8th IIAE International Conference on Industrial Application Engineering 2020 (ICIAE 2020), pp.4-11, Mar. 2020.

[24] Y. Yamato, "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.

[25] 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.

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

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

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

[29] J. Chen, et al., "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.

[30] C. Bertolli, et al., "Integrating GPU support for OpenMP offloading directives into Clang," ACM Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM'15), Nov. 2015.

[31] S. Lee, et al., "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.

[32] Cheng Liu, et al., "Automatic nested loop acceleration on fpgas using soft CGRA overlay," Second International Workshop on FPGAs for Software Programmers (FSP 2015), 2015.

[33] L. Sommer, et al., "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.

[34] A. Putnam, et al., "CHiMPS: A C-level compilation flow for hybrid CPU-FPGA architectures," IEEE 2008 International Conference on Field Programmable Logic and Applications (FPL 2008), pp.173-178, Sep. 2008.

[35] L. Sun, et al., "An open IoT framework based on microservices architecture," China Communications, Vol.14, No.2, pp.154-162, 2017.

[36] B. W. Wirtz, et al., "An integrative public IoT framework for smart government," Government Information Quarterly, Vol.36, No.2, pp.333-345, 2019.

[37] M. Alshehri, "Blockchain-assisted internet of things framework in smart livestock farming," Internet of Things 22, Elsevier, pp.100739, 2023.

[38] G. Singh, and S. Jaspreet "A Fog Computing based Agriculture-IoT Framework for Detection of Alert Conditions and Effective Crop Protection," 2023 5th International Conference on Smart Systems and Inventive Technology (ICSSIT), IEEE, 2023.

[39] S. M. Kim, et al., "IoT home gateway for auto-configuration and management of MQTT devices," In 2015 IEEE Conference on Wireless Sensors (ICWiSe), pp.12-17, Aug. 2015.

[40] J. M. Gutierrez-Guerrero and J.A. Holgado-Terriza, "Automatic configuration of OPC UA for Industrial Internet of Things environments," Electronics, Vol.8, No.6, pp.600, 2019.

[41] H. Hajizadeh, M. Nabi and K. Goossens, "Decentralized Configuration of TSCH-Based IoT Networks for Distinctive QoS: A Deep Reinforcement Learning Approach," IEEE Internet of Things Journal, 2023.

[42] S. Khoun, H. Chemali and N. Kerkar, "Alternative technique of messaging through a new configuration of IOT devices," International Conference on Pioneer and Innovative Studies, Vol.1, 2023.

[43] M. Kamimura, et al., "Extracting Candidates of Microservices from Monolithic Application Code," 2018 25th Asia-Pacific Software Engineering Conference (APSEC), pp.571-580, 2018.