Study of program placement optimization in mixed environments in environmental adaptation

Yoji Yamato[a,*]

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

*Corresponding Author: yoji.yamato@ntt.com

Abstract

In recent years, the number of applications using heterogeneous hardware such as FPGAs and GPUs has been increasing. However, to fully utilize them, an understanding of the hardware is necessary, which is a high barrier. Based on this background, we have proposed environment-adaptive software that automatically converts code written by programmers for normal CPUs according to the deployment environment, enabling high performance operation. This paper targets automatic offloading to appropriate hardware in a mixed environment that contains normal CPUs, multi-core CPUs, FPGAs, GPUs, and quantum computers. We confirm how much the performance of normal CPUs can be achieved with each hardware, and based on cost performance, we determine the appropriate hardware for offloading and perform automatic offloading. We confirm that the proposed method can automatically offload by measuring the processing time using actual heterogeneous hardware.

Keywords: Environment-Adaptive Software, Automatic Offloading, Quantum Computer, Mixed Environment, Hardware Selection.

1. Introduction

It is said that there are some aspects of Moore's Law that expect CPUs (Central Processing Units) to become faster over time that are no longer compatible with it. As a result, in addition to normal CPUs, the use of heterogeneous hardware such as multi-core CPUs, FPGAs (Field Programmable Gate Arrays), and GPUs (Graphics Processing Units) is increasing in applications. For example, Microsoft has stated the use of FPGAs for search (1), and Amazon provides FPGAs and GPUs as cloud (e.g., (2)-(6)) instances (7). In addition to FPGAs and GPUs, the number of IoT devices is also increasing (e.g., (8)-(16)), and new types of hardware such as DPUs (Data Processing Units) and quantum computers are also emerging.

However, in order to accelerate on heterogeneous hardware, it is necessary to configure and write code with consideration for the characteristics of the hardware, and programming skills using C language extensions such as OpenCL (Open Computing Language) (17), OpenMP (Open Multi-Processing) (18), and CUDA (Compute Unified Device Architecture) (19) are desirable. Therefore, for many programmers who have become accustomed to using scripting languages such as PHP and Python, the skill hurdle is high.

Heterogeneous hardware systems such as FPGAs, GPUs, and quantum computers can also reduce power consumption and are expected to develop in the future, but the skill hurdle is high to use them. Therefore, a new platform that can fully utilize heterogeneous hardware without the hurdles will be important, which converts and configures software written by programmers in the same way as usual, and adapts it to the environment (FPGA, GPU, quantum computer, etc.) where it will be deployed.

Therefore, we have proposed environment-adaptive software that automatically converts existing software code for FPGAs and GPUs and automatically sets the amount of resources so that the code can be used in the deployment environment, thereby accelerating applications. As environment-adaptive software elements, we are evaluating methods to automatically offload loop statements and function blocks of existing code to FPGAs, GPUs, quantum computers, etc. (20)-(27).

However, our verification so far has focused mainly on automating offloading to individual hardware such as FPGAs, GPUs, and quantum computers, and appropriate

offloading to a mixed environment has not been considered. For example, even if GPU offloading provides 10 times the performance and FPGA offloading provides 20 times the performance, if the monthly cloud usage fee for an FPGA instance is three times that of a GPU instance, GPU offloading may be appropriate.

This paper focuses on automatic offloading to appropriate hardware in a mixed environment that includes normal CPUs, multi-core CPUs, FPGAs, GPUs, and quantum computers. First, we analyze the existing application to be offloaded, find the offloadable parts, and attempt to offload to each hardware using the existing proposed method. We confirm how many times the performance of a normal CPU can be achieved with each hardware, and based on the cost performance determined from the cloud usage fee for that hardware, we determine the appropriate hardware for offloading and automatically offload it. We confirm that the proposed method can automatically offload by measuring the processing time using actual heterogeneous hardware: AMD Ryzen, Intel Stratix, NVIDIA Geforce, and Microsoft Azure Quantum (28).

2. Existing Technologies

2.1 Market technologies

NVIDIA has proposed CUDA for development of GPGPU (General Purpose GPU). CUDA is targeted at GPU, but OpenCL has been proposed for handling heterogeneous hardware such as GPU, multi-core CPU, and FPGA, and various companies provide tools for it. CUDA and OpenCL require C language extension program writing, which is difficult (e.g., releasing and copying data in memory between kernels such as FPGA and CPU host).

Unlike CUDA and OpenCL, there are compilers that specify the part to be processed on a directive basis and create binary files for GPUs and so on according to the directive, in order to easily use heterogeneous hardware. Specifications include OpenACC (9) and OpenMP, and compilers include PGI compiler (30) and gcc.

GPU, multi-core CPU, and FPGA offloading is possible using CUDA, OpenCL, OpenACC, OpenMP, etc. However, even if processing is performed, it is difficult to speed up. For example, Intel compiler (31) is used for multi-core CPU parallelization. This parallelizes the parts of the loop that can be processed in parallel. However, simply parallelizing the loop does not speed up the process mainly for memory usage problems. When speeding up the process using a GPU, CUDA experts perform tuning work and use a compiler to search for suitable parallel processing parts. As an automated search method, the author proposes offloading using evolutionary computing.

Quantum computers are computers that apply quantum mechanics. There has been an increase in research and development of quantum computers using the versatile quantum gate method. Many companies, including Microsoft, Amazon, IBM, and Google, are working on this. Azure Quantum and AWS Braket are provided as functions of Microsoft's Azure clouds and Amazon's AWS clouds, respectively, and allow for time-based use of the cloud, with no initial costs for quantum computers. However, the current situation is that quantum algorithms for solving prime factorization and knapsack problems and so on using quantum computers are devised by experts and implemented one by one, and they are not easily accessible to general users.

2.2 Outline of environment-adaptive software

We have proposed environment-adaptive software with the processing overview shown in Figure 1. The environment-adaptive function provided by the operator is at the core, and the environment-adaptive software works by linking the production environment, verification environment, equipment resource DB, test case DB, and code pattern DB.

Step 1: Analysis of user-provided code. Step 2: Extraction of offloadable parts. Step 3: Search for appropriate offload parts. Step 4: Adjustment of production resource amount. Step 5: Adjustment of production deployment location. Step 6: Production deployment of binary files and verification. Step 7: Reconfiguration during production operation.

User provided code is analyzed, and before production operation begins, in Steps 1-6, code conversion, resource

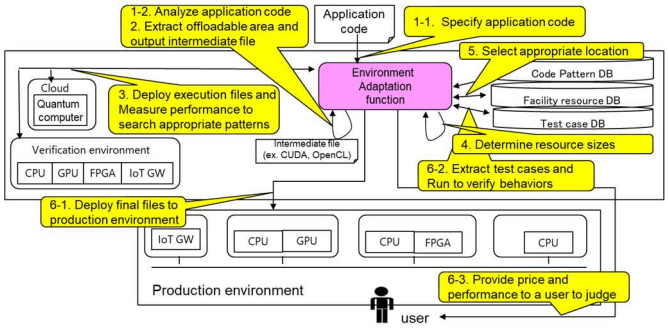


Fig. 1. Processing outline of environment-adaptive software

adjustment, placement location adjustment, and operation verification are performed through verification environment performance measurement tests. In Step 7, after production operation begins, actual usage data is analyzed, and reconfiguration is performed if it is clear that a configuration change is appropriate.

2.3 Problems in this paper

Now, we summarize the problems. Acceleration using heterogeneous hardware requires manual work by an expert. We have previously proposed environment-adaptive software and realized an automatic offloading method for FPGAs, GPUs, and quantum computers. However, until now the aim was to automate offloading for individual hardware, and appropriate offloading in a mixed environment had not been fully considered. For example, even if a GPU provides 10 times the performance and an FPGA provides 20 times the performance, if the usage fee for FPGAs is three times that of GPUs, then GPUs would be appropriate. Therefore, this paper targets automatic offloading to appropriate hardware in an environment that contains normal CPUs, multi-core CPUs, FPGAs, GPUs, and quantum computers. We propose an appropriate automatic offloading method for a mixed environment and confirm the proposed method using a real heterogeneous hardware mixed environment.

3. Offloading to a mixed environment including quantum computers

Offloading to individual hardware is attempted using the previous proposed method. First, we use a syntax analysis library such as Clang (32) to understand the program structure, and then attempt to offload it to a quantum computer. In quantum computer offloading, function block offloading is performed, so after the quantum computer, a search is performed to see if there is a library equivalent that can be used with multi-core CPUs, GPUs, and FPGAs. If found, the function block offloading is performed and performance is measured. Regardless of whether function block offloading is possible or not, after the function block offloading attempt, loop statement offloading is attempted in the order of multi-core CPUs, GPUs, and FPGAs, and performance is measured with the final solution of evolutionary computation. Since performance measurement results for all patterns are obtained, the cost performance is calculated based on the usage fee for each hardware, and the appropriate hardware is selected.

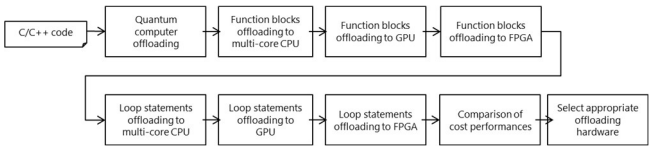


Fig. 2. An appropriate automatic offloading method for mixed environments

The details of the proposed method are explained using Figure 2.

3.1 Function block offloading trial

Function block offloading trials are performed for each hardware, including quantum computers. We use similarity detection tools such as Deckard (33) to discover whether there are any functional blocks that can be offloaded to each hardware. A similarity detection tool is a tool that makes it possible to detect similar code, such as code that has been copied and then modified. Similarity detection uses similarities in abstract syntax trees, vocabulary, lines, program dependency graphs, fingerprints and so on. This time, we use a tool that confirms abstract syntax trees.

When a function block that can be offloaded is found, it is replaced with a processing part (equivalent to a library) that uses the hardware corresponding to that code registered in the code pattern DB, and offloaded. When using a quantum computer, it is based on a quantum algorithm and is implemented using one of the following. Qiskit (34) is a quantum computer framework open sourced by IBM, Cirq is a quantum computer framework open sourced by Google, and Q¥# is a programming language for quantum computers proposed by Microsoft. When using a multi-core CPU instead of a quantum computer, it is replaced with an OpenMP file that specifies the multi-core CPU library, when using a GPU, it is replaced with an OpenACC file that specifies the GPU library, and when using an FPGA, it is replaced with an OpenCL file that corresponds the FPGA IP core.

When an offload function block is found and replaced with a processing part, the processing time when offloaded and the processing time when executed on a normal CPU are measured again, and the performance improvement degree when offloaded is obtained.

3.2 Loop statement offloading trial

Loop statement offloading trials are performed for each hardware. The structure of the for statement in the user

application is understood using a syntax analysis library. For multiple for statements, patterns are created for whether or not to offload to hardware, and a fast offload pattern is found through repeated performance measurements in the verification environment. This is common to all hardware. The verification order may change, but the basic order is multi-core CPU, GPU, and FPGA. This is because FPGAs in particular take a long time from compilation to actual device measurement, so if reasonably good results are obtained in multi-core CPU and GPU offloading trials that take a similar amount of processing time, they can be skipped.

In trials in the verification environment, for multi-core CPUs and GPUs, methods such as (20) are used to measure multiple patterns of whether or not for statements can be offloaded, using GA, an evolutionary computing method, to find a fast pattern. Based on methods such as (22), FPGA narrows down candidate loop statements based on arithmetic intensity, loop count, and resource amount analyzed using the ROSE framework (35), and then measures the performance of multiple patterns in the verification environment to find a high-speed offload pattern. The final solution is an OpenMP file for multi-core CPUs, an OpenACC file for GPUs, and an OpenCL file for FPGAs.

Performance measurements are repeated along the way, and the processing time when offloaded with the final solution and the processing time when executed on a normal CPU are measured again to obtain the degree of performance improvement when offloading.

3.3 Hardware selection based on cost performance

There are cases where offloading is not possible because a replaceable part cannot be found, or where offloading does not improve performance, up to seven patterns are available, including offloading function blocks to quantum computers, multi-core CPUs, GPUs, and FPGAs, and offloading loop statements to multi-core CPUs, GPUs, and FPGAs. Basically, function block offloading replaces a dedicated processing part (equivalent to a library), so the degree of speedup is often large. In (25), when the CPU's Fourier transform processing was replaced with the GPU library cuFFT, the performance was 730 times higher. Therefore, if function block offloading is possible, the selection of hardware corresponding to the processing part is a strong candidate. On the other hand, if function block offloading is not possible and only loop statements are offloaded, it is necessary to decide which hardware is appropriate for offloading.

Therefore, we propose a method to determine using the cost performance based on user requests. The degree of improvement compared to a normal CPU is measured for seven offloading patterns. Here, when processing with a quantum computer, most problems cannot be solved by a CPU in the first place, and a normal CPU will time out, so there is no need to calculate the degree of improvement. Specifically, the degree of improvement for each offloading pattern is divided by the offloading hardware usage fee (monthly usage fee for cloud instances) to determine the cost performance. If the cost performance is higher than that of a normal CPU and is the highest across all patterns, that hardware is appropriate, so it is selected as the system and proposed to the user.

For example, if a VM with a normal CPU costs 60 USD/Month, a VM with a GPU costs 200 USD/Month, and a VM with an FPGA costs 700 USD/Month, and an application achieves 10 times the performance of a normal CPU with GPU offloading and 20 times the performance of a normal CPU with FPGA offloading, the cost performance is calculated. In this case, GPU offloading is determined to be appropriate, so it is proposed to the user.

4. Evaluation

The effectiveness of the method is evaluated by confirming that four applications specified by the user can be automatically offloaded to a mixed environment of quantum computers, multi-core CPUs, GPUs, and FPGAs in an appropriate manner and that cost-effective hardware is selected.

4.1 Evaluation conditions

(a) Evaluated targets

There are four applications.

The evaluation target is an eigenvalue analysis problem. For high-dimensional complex matrices, there is no calculation procedure that can accurately express eigenvalues with a finite number of algebraic operations. For this reason, iterative methods have traditionally been used for numerical analysis of eigenvalue problems. This can be accelerated by utilizing the superposition property of quantum states of quantum computers. The essence of the problem is to find the eigenvalues of the associated matrix of a general n-th order algebraic equation, and each problem is determined by differences in parameters. The eigenvalue analysis problem used is (36). As (36) shows,

the solution of eigenvalue analysis problems on quantum computers is also implemented in Qiskit using quantum algorithms.

There is NAS.BT (37), a block diagonal solver calculation. A block diagonal solver is a numerical solution method for partial differential equations. There are various types and implementations of numerical calculations. We use NAS.BT (NASA block triangular solver) (37) as an example of a medium-scale numerical calculation application with more than 100 loop statements. The parameters are CLASS A settings, grid size is 64*64*64, number of iterations is 200, and time step is 0.0008.

Himeno Benchmark (38) is a benchmark software used to measure the performance of incompressible fluid analysis, and solves the Poisson equation using the Jacobi iteration method. It is frequently used for manual acceleration on accelerators, and is used to confirm that the proposed automatic method can also accelerate it. The data size used is LARGE (512*256*256).

MRI-Q (39) is an MRI image processing software that calculates the Q matrix that represents the scanner settings for calibration. MRI-Q is used in 3D MRI reconstruction algorithms in non-Carthean space. In many fields such as IoT, image processing is often required for automatic monitoring of camera images, and offloading is important to improve the throughput of image processing. MRI-Q performs 3D MRI image processing, and the processing time is measured using data of size 64*64*64.

(b) Evaluation method

The user requests offloading of four applications. When the cloud platform receives the request, it analyzes the application and offloads function blocks and loop statements to the quantum computer, multi-core CPU, GPU, and FPGA.

Usage fees: 90 USD/hour for quantum computers, 60 USD/month for normal CPU VMs, 140 USD/month for multi-core CPU VMs, 200 USD/month for GPU VMs, and 700 USD/month for FPGA VMs. Usage fees are based on market clouds.

Function block offloading trials will be performed as follows.

Function blocks: Eigenvalue analysis problems, NAS.BT, Himeno benchmark, MRI-Q.

Function block discovery method: Function blocks contained in the code are matched with the code contained in the code pattern DB using the similarity detection tool Deckard.

Loop offloading trials are performed as follows.

A genetic algorithm is used for multi-core CPUs and GPUs.

Offloading targets and number of loop statements: eigenvalue analysis problem 0, NAS.BT 120, Himeno benchmark 13, MRI-Q 12.

Number of individuals M: Less than the number of loop statements (NAS.BT 20, Himeno benchmark 10, MRI-Q 10)

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

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

The shorter the processing time, the higher the goodness of fit. Also, by raising it to the (-1/2) th power, the goodness of fit of a specific individual with a short processing time is prevented from becoming too high, narrowing the search range. Also, if the performance measurement does not end within the fixed time of 3 minutes, a timeout is triggered and the goodness of fit is calculated with the processing time set to the infinity.

Selection: Roulette selection. However, the best-fit genes in a generation are preserved in the next generation without crossover or mutation, so elite preservation is also performed.

Crossover rate Pc: 0.9

Mutation rate Pm: 0.05

For FPGAs, measurements are made after narrowing down the results.

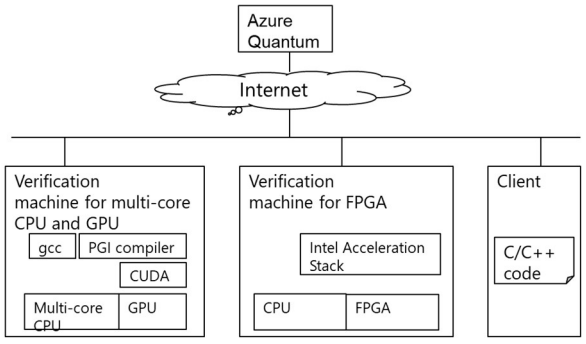
Narrowing down by arithmetic intensity: Narrow down to the top 5 loop statements in arithmetic intensity analysis.

Narrowing down by resource efficiency: Narrow down to the top 3 loop statements in resource efficiency analysis (select the top 3 loop statements with the highest arithmetic intensity/resource amount).

Number of offload patterns actually measured: 4 (The first time, the top 3 loop statement offload patterns are measured, and the second time, the combination pattern of the two loop statement offloads that had the highest performance in the first time are measured.

Performance measurement: Processing time is measured with the sample parameters unchanged.

Under the above conditions, a function block offload trial and a loop statement offload trial are performed, the



Name	Hardware	CPU	RAM	GPU/FPGA	OS	gcc	CUDA toolkit	PGI Compiler	Intel Acceleration Stack
Verification machine for multi-core CPU and GPU	LEVEL-F039-LCRT2W-XYVI	AMD Ryzen Threadripper 2990WX (32 Core)	32GB *2	NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory: GDDR6 11GB)	Ubuntu 16.04.6 LTS	10	10.1	19.1	
Verification machine for FPGA	Dell PowerEdge R740	Intel(R) Xeon Bronze 3206R *2	32GB *4	Intel PAC D5005 (Intel Stratix 10 GX FPGA)	Cent OS 7.9				2
Quantum computer	Azure Quantum								
Client	HP ProBook 470 G3	Intel Core i5-6200U @2.3GHz	8GB		Windows 10 Pro				

Fig. 3. Performance measurement environment

processing time during offload processing is measured, and the measurement results and usage fees for the four applications are presented to the user.

(c) Evaluation environment

The quantum computer used is Azure Quantum, and processing is performed via the Internet through a Qiskit implementation for quantum computers. The multi-core CPU used is an AMD Ryzen Threadripper 2990WX (32 cores), and OpenMP processing uses gcc 10.1. The GPU used is an NVIDIA GeForce RTX 2080 Ti (CUDA core: 4352, Memory: GDDR6 11GB), and OpenACC processing uses PGI compiler 19.10 and CUDA Toolkit 10.1. Ryzen and GeForce are installed on the same node. The FPGA used is an Intel PAC D5005 (Intel Stratix 10 GX FPGA). The compilation machine is a DELL EMC PowerEdge R740 (CPU: Intel Xeon Bronze 3206R *2, RAM: 32GB RDIMM * 4), and OpenCL is compiled with Intel Acceleration Stack 2.0. The evaluation environment and specifications are shown in Figure 3. Here, the C/C++ language applications used by the user is specified from the client notebook PC, and the verification machine uses Deckard to match and search the DB on the same node. The processing time is measured by the verification machine and appropriate hardware is determined and deployed in a production environment used by actual users.

4.2 Results

Figure 4 shows the processing time, performance improvement, and cost performance when the normal CPU VM is set as 1 when four applications are offloaded to a mixed environment using the proposed method. The four

Applications	Offloading destination	Performance improvement	Processing time	Cost performance [/Offloading VM fee /normal CPU VM fee]
Eigenvalue solvers	Quantum computer		15.7 sec	
NAS.BT	multi-core CPU	5.39	24.1 sec	2.31
Himeno benchmark	GPU	22	1.87 sec	6.59
MRI-Q	FPGA	11.8	2.21 sec	1.01

Fig. 4. Results of offloading 4 applications to a mixed environment

applications select hardware with high cost performance.

First, eigenvalue analysis problems are often unsolvable by normal CPUs, but by offloading to a quantum computer, the processing is completed in about 15 seconds. Next, in the loop offload trial of NAS.BT, a multi-core CPU was automatically selected as the appropriate offload destination, and the performance was more than 5 times that of a normal CPU and the cost performance was 2.3 times that of a normal CPU. Next, in the loop offload trial of Himeno benchmark, a GPU was automatically selected as the appropriate offload destination, and the performance was more than 20 times that of a normal CPU and the cost performance was 6.6 times that of a normal CPU. Next, in the loop offload trial of MRI-Q, an FPGA was automatically selected as the appropriate offload destination, and the performance was more than 11 times that of a normal CPU and the cost performance was 1.01 times that of a normal CPU (generally, FPGA instances cost about 10 times the usage fee of a normal CPU VM, and the cost performance is not so high).

The time it takes to offload is completed in seconds or minutes for a function block offloading trial, but it takes time for a loop statement offloading trial. In the case of a multi-core CPU or GPU, it depends on the number of measurements in GA, but in the case of NAS.BT and other cases where there are many for statements, it takes about 8 hours. In the case of FPGA, it takes about 8 hours to compile OpenCL and perform one measurement, so the more measurements there are, the longer it takes. In addition, because Azure is used as an external service, when the user, not the service provider, contracts with Azure, an examination is required, so a certain amount of time and operation is required before the contract can be made.

5. Conclusions

In this paper, we proposed a method to automatically

offload user-provided applications to an environment that includes a mixture of normal CPUs, multi-core CPUs, GPUs, FPGAs, and quantum computers as part of the environment-adaptative software we have proposed.

The automatic offload trial for each individual piece of hardware uses the previous method proposed by the author. First, the user application is analyzed. Based on syntax tree similarity using Deckard etc., function block offload trials are performed on multi-core CPUs, GPUs, FPGAs and quantum computers, and the number of times faster it is compared to a normal CPU is obtained for each piece of hardware. Next, regardless of whether function block offload is possible, loop statement offload trials using evolutionary computing are performed on multi-core CPUs, GPUs and FPGAs. The number of times faster is divided by the monthly usage fee for each piece of hardware to find cost performance. The application is automatically offloaded to the hardware with the best cost performance, which is higher than that of a normal CPU.

In this verification, four user applications - eigenvalue analysis problems, NAS.BT, Himeno benchmark, and MRI-Q - were analyzed and performance measured, and the effectiveness of the method was confirmed by automatically offloading them to a quantum computer, multi-core CPU, GPU, and FPGA, respectively.

References

(1) A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," Proceedings of the 41th Annual International Symposium on Computer Architecture (ISCA'14), pp.13-24, June 2014.

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

(3) 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.

(4) Y. Yamato, "Optimum Application Deployment Technology for Heterogeneous IaaS Cloud," Journal of Information Processing, Vol.25, No.1, pp.56-58, 2017.

(5) Y. Yamato, "Cloud Storage Application Area of HDD-SSD Hybrid Storage, Distributed Storage and HDD Storage," IEEJ Transactions on Electrical and Electronic Engineering, Vol.11, Issue.5, pp.674-675, DOI:10.1002/tee.22287, Sep. 2016.

(6) 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.

(7) AWS EC2 web site, https://aws.amazon.com/ec2/instance-types/

(8) M. Hermann, et al., "Design Principles for Industrie 4.0 Scenarios," Rechnische Universitat Dortmund. 2015.

(9) 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.

(10) M. Takemoto, et al.,"Service-composition method and its implementation in service-provision architecture for ubiquitous computing environments," IPSJ Journal, Vol.46, No.2, pp.418-433, Feb. 2005.

(11) H. Noguchi, et al., "Distributed Search Architecture for Object Tracking in the Internet of Things," IEEE Access, DOI: 10.1109/ACCESS.2018.2875734, 2018.

(12) Y. Yamato, et al., "Ubiquitous Service Composition Technology for Ubiquitous Network Environments," IPSJ Journal, Vol.48, No.2, pp.562-577, Feb. 2007.

(13) 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.

(14) Y. Yamato, et al., "Method of Service Template Generation on a Service Coordination Framework," 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Nov. 2004.

(15) 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, pp.289-293, 2016.

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

(17) J. E. Stone, et al., "OpenCL: A parallel programming standard for heterogeneous computing systems," Computing in science & engineering, Vol.12, No.3, pp.66-73, 2010.

(18) T. Sterling, et al., "High performance computing : modern systems and practices," Cambridge, MA : Morgan Kaufmann, ISBN 9780124202153, 2018.

(19) J. Sanders and E. Kandrot, "CUDA by example : an introduction to general-purpose GPU programming," Addison-Wesley, 2011.

(20) Y. Yamato, "Proposal and evaluation of GPU offloading parts reconfiguration during applications operations for environment adaptation," Journal of Network and Systems Management, Springer, DOI: 10.1007/s10922-023-09789-2, Nov. 2023.
(21) Y. Yamato, "Study and Evaluation of Automatic Offloading Method in Mixed Offloading Destination Environment," Cogent Engineering, Taylor and Francis, Vol.9, Issue 1, DOI: 10.1080/23311916.2022.2080624, June 2022.
(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) 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.
(24) 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.
(25) 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.
(26) 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.
(27) 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.
(28) Azure quantum website, https://azure.microsoft.com/products/quantum/
(29) S. Wienke, et al., "OpenACC-first experiences with real-world applications," Euro-Par 2012 Parallel Processing, pp.859-870, 2012.
(30) M. Wolfe, "Implementing the PGI accelerator model," ACM the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.43-50, Mar. 2010.
(31) E. Su, et al., "Compiler support of the workqueuing execution model for Intel SMP architectures," In Fourth European Workshop on OpenMP, Sep. 2002.
(32) Clang website, http://llvm.org/
(33) Deckard website, http://github.com/skyhover/Deckard
(34) Qiskit website, https://www.ibm.com/quantum/qiskit
(35) ROSE framework website, http://rosecompiler.org/ROSE_HTML_Reference/index.html
(36) Eigenvalue Solver website, https://learning.quantum.ibm.com/course/variational-algorithm-design/instances-and-extensions
(37) NAS.BT website, https://www.nas.nasa.gov/publications/npb.html
(38) Himeno benchmark web site, http://accc.riken.jp/en/supercom/
(39) MRI-Q website, http://impact.crhc.illinois.edu/parboil/