Predictive Many-core Allocation Method for Edge Off-Road Services

Masato Fukui† Yoichi Ishiwata‡ Takeshi Ohkawa§ Midori Sugaya†

Abstract: In recent years, edge computing has been attracting attention for the purpose of offloading advanced processing by IoT and robot services. It needs to efficiently provide services to multiple robotics services while dynamically switching abundant computational resources such as many-core resources. In Many-core research, many studies have been conducted to increase the degree of parallelism of tasks to improve responsiveness and computational efficiency. However, there is still not enough discussion on how to effectively allocate resources in a way that is suitable for integrated services and to achieve both efficiency and responsiveness. In this study, we construct an efficient resource allocation prediction formula as a study of an efficient Many-core allocation method for edge offload. In addition, as a specific service, we decided to evaluate the offload of AI processing of communication robots. In the construction of the prediction formula, it was confirmed that sufficient processing performance and responsiveness can be obtained with the minimum core by automatically calculating the optimum number of cores from the actual application operation.

Keywords: Many-core, Edge computing, Off-Road

1. Introduction

In recent years, robots equipped with advanced AI have widely spread [1]. Among them, service robots are expected to realize advanced interaction with humans using AI [2]. These robots support people not only at home but also at educational institutions, watching over the elderly, and providing medication support. In the further, it is widely spread.

These high-performance communication robots require advanced computational processing to achieve more advanced human robot interaction such as to understand the complex emotions and behaviors of humans. Unibo provides a personal interaction mechanism through offloading (load distribution) the calculation process of image data on the cloud [2]. This is because the robot itself is not always equipped with a high-performance computer, and it is necessary to reduce the load of advanced calculation processing. In addition, if the use of a large amount of data such as images that increases due to the recognition processing of moving objects by robots, it is predicted that data transfer to the cloud will become a responsive bottleneck [3].

As a solution for reducing computational power, edge computing has been proposed as a technology for supporting those high-performance required service [4]. In edge computing, a server with abundant computing resources such as manycore server is installed near the robot device. It needs to efficiently provide services to multiple robotics while dynamically switching abundant computational resources. In many-core research, several studies have been conducted to increase the degree of parallelism of tasks to improve responsiveness and computational efficiency [6-8].

However, there is still not enough discussion on how to effectively allocate resources in a way that is suitable for integrated sophisticated multiple robotic interactive services with satisfying their requirement of responsiveness.

In addition, empirical research on edge servers that considers the use and responsiveness of multiple data has not been sufficiently discussed. In this study, we firstly developed an efficient resource allocation mechanism for considering the actual robotic service and its requirement with the many-core edge server, and propose a prediction formula as a study of an efficient many-core allocation method for the edge offload. We consider the sufficient processing performance and responsiveness can be obtained with the minimum core by automatically calculating the optimum number of cores from the actual application operation.

The structure of this paper is as follows. Section 2 describes related research and issues. Section 3 describes the proposal, Section 4 describes the experiment for verification, Section 5 describes the creation and verification of the estimation formula based on the experiment, and Section 6 describes the summary and future issues.

2. Related research

In edge computing, research is being conducted to improve items such as Latency, Bandwidth, Availability, Energy, Secure, and Privacy with the aim of satisfying a large amount of resource demands around us [5]. Latency is especially important for robots to provide interaction services with humans. The most efficient method for processing multiple parallel processes at high speed is to perform advanced parallel processing such as machine learning with multiple cores to improve responsiveness.

Ogoshi et al. Introduced an edge server between the IoT device and the cloud. By offloading processing from an IoT device with limited computer resources to an edge server, they have realized efficient operation of DNN model learning in deep learning using a huge amount of learning data [6]. However, research is limited to a specific AI model, and a general-purpose model for the many-core server that considers the resource management of them has not yet been sufficiently proposed.

† Department of Information Science and Engineering, College of Engineering, Shibaura Institute of Technology
‡ VA Linux Systems Japan K K
§ Tokai University

Oka et al. Introduced parallelism at the local task level into the task-driven coarse-grained parallel processing of java's Fork / Join Framework for environments equipped with manycore processors. As a result, they succeeded in shortening the execution time of parallel processing on the manycore processor [7]. Also, Takamaeda et al. Have proposed a method to reduce the latency caused by inter-core communication in manycore processors. In this research, parallelism between processing and communication is detected by hardware. In this way, the processing can be executed without waiting for the transfer to be completed, so the stall time due to communication can be reduced by up to 40% [8]. However, sufficient research has not been conducted on the application of methods focusing on off-road processing of advanced robot processing.

3. Proposal

To achieve the service with minimum response time, we propose an efficient resource allocation mechanism for the many-core edge server, and to achieve them, propose a prediction formula as a study of an efficient many-core allocation method for edge offload.

In developing the prediction formula, we assumed actual robotic service that operates as shown in Figure. 1. As a specific example of advanced interaction, we assume a robot that uses human related information such as brain waves, which is expected to develop in recent years, to grasp human emotions by processing using machine learning in the future. This robot system is requested to return a response within 1 second [9] after receiving the data. During the 1 second, the system should (3) analyze the data, (4) modeling with machine-learning (5) selects the control operation based on the model, (6) instruction for the robot. Generally, this operation requires the several steps, it is necessary to execute the application composed program with parallel manner.

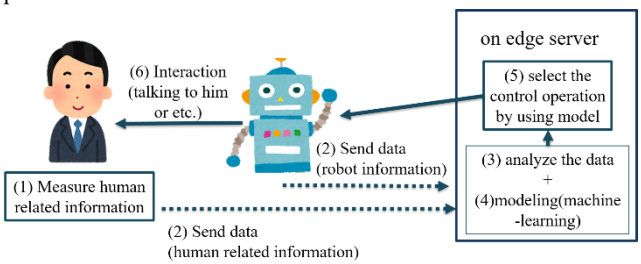

Figure 1: advanced interaction system overview

The purpose of this study is to realize load offload by manycore. However, it is not clear how to allocate resources to manycore with considering supporting interactive service for the robotics. From the experiences of the robotics application development [10], we consider the following points to satisfy the allocation mechanism. (1) The resource requirement of the robotics service totally depends on each different robotic service. (2) Generally, the robotic service is composed of the several types of execution components such as streaming data treatment, and control calculations, and signal processing. To achieve the purpose to suitable allocation with satisfying the response time requirement, we consider dynamically assign the core satisfying the request of the robotics service (Figure 2). To achieve them, we consider some predictive method to know how much cores should be assigned to the robotics service.

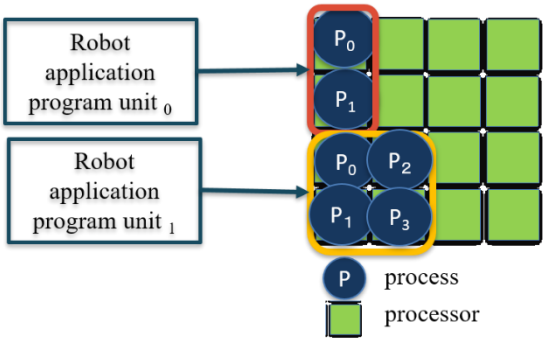

Figure 2: Application-dependent dynamic allocation image

To clarify the predictive allocation for the service, and to make some models for them, we conducted an experiment that has a purpose to investigate the relationship between the execution time when the processor affinity was set in the multithreaded program and the multi process program, and the number of allocated processors, and clarified the relationship.

4. Experiments

To achieve the proposed idea, we conducted two set of experiments. To know the appropriate assignment for the cores, we firstly consider the benchmark program that is constitute by threads and processes. We planned the two experiments by threads and process as follows.

4.1 Experiment 1: Multi thread

To execute the component robotics application parallels, we consider developing a program that execute the number of pthreads and evaluated the execution time of that program.

Since threads will achieve the parallel execution on the manycore with the right-weight manner. In each thread, 10 million for() loops were executed to derive the pi using the Monte Carlo method. In addition, there is no communication or dependency between each thread. Allocate available processors by using the sched_setaffinity () system call for the first process that issues the pthread_create() library call. The processor was assigned to the thread group. The group by utilizing the fact that the thread created from the first process is also executed by the same processor.

4.2 Experiment 2: Multi process

Inside each process, 10 million for() loops were executed to derive the pi using the Monte Carlo method. In addition, there is no dependency such as inter-process communication in each process. In the experiment, the processes with parent-child relationship were set as process groups, and processors were assigned to the process groups. When a fork() system call is issued in a process for which processor affinity is set using the sched_setaffinity () system call. A child process is created from

one of the parent in the process group, the child process inherits the processor allocation of the parent process. Using this, a processor was assigned to the process group.

4.3 Experimental Environment

We set up the experimental environment based on the manycore environment. The Intel Xeon Gold 6230 (20Core, 2.1GHz, 27.5MB cache) has 20 cores and we use the two CPU that means 40 cores for the experiment. To avoid the effect of the cache and hyper-threading, we disabled the cache and hyper-threading respectively. To evaluate the execution time, we use the UNIX time command to measure the execution time of the developed benchmark program.

Table 1 Machine Spec

OS	Ubuntu 18.04 LTS
CPU	Intel Xeon Gold 6230 (20Core, 2.1GHz, 27.5MB cache) x 2
memory	DDR4-2933 REG ECC 16G x 12

4.4 Result

4.4.1 Result: Experiment 1

The results of varying the number of allocated processors and increasing the number of threads in stages at this time are shown below. First, the results were plotted on a 3d graph. (Figure 3) The execution time is the z label, the number of processors is the x label, and the number of threads is the y label. The two-dimensional graphs Figure 4 and 5 also show execution time. In this graph the horizontal axis is the number of threads.

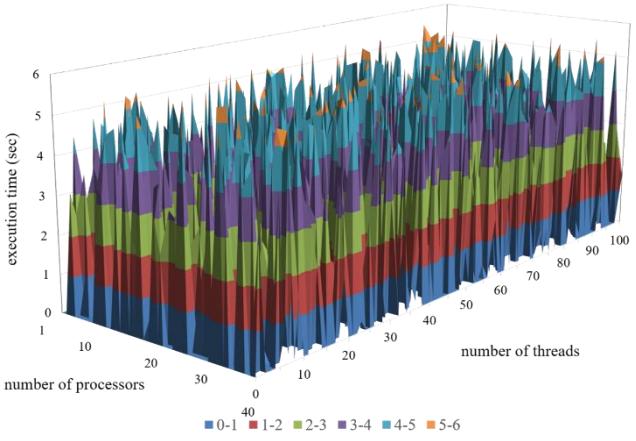

Figure 3: 3D graph of the execution time of the increased number of threads and processors

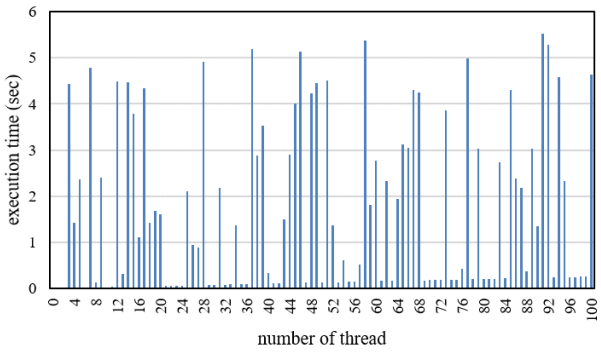

Figure 4: Execution time of the threads runs on one processor

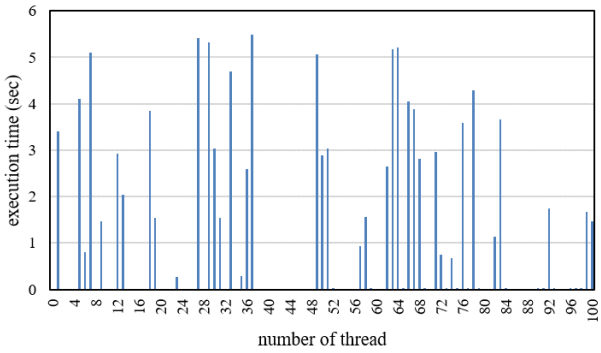

Figure 5: Execution time of the threads runs on 40 processors

4.4.2 Result: Experiment 2

The results of changing the number of allocated processors and gradually increasing the number of processes at this time are shown below. First, the results were plotted on a 3d graph. (Figure 6) The execution time is the z label, the number of processors is the x label, and the number of processes is the y label. The two-dimensional graphs Figure 7, 8 and 9 also show execution time. In this graph the horizontal axis is the number of threads.

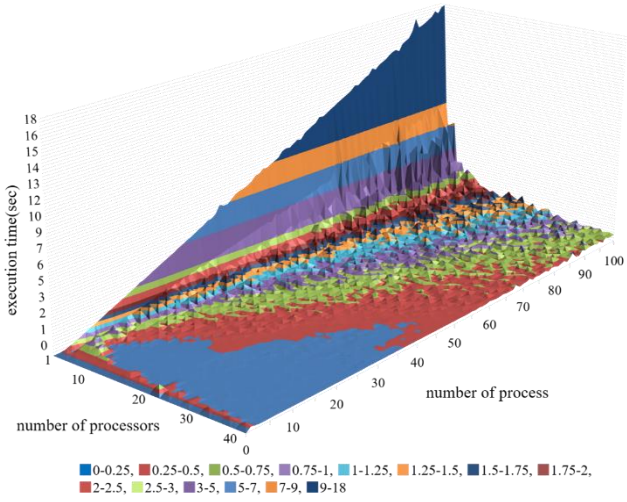

Figure 6: 3D graph of the execution time of the increased number of process and processors

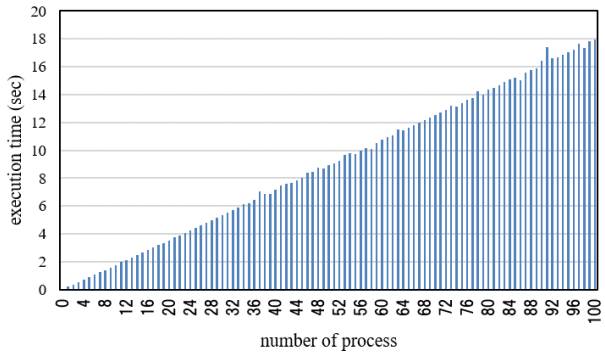

Figure 7: Execution time of the process runs on 1 processor

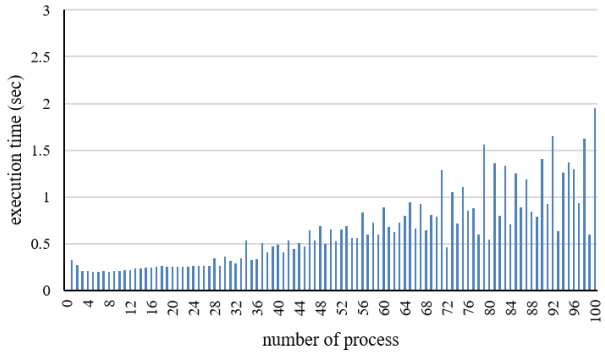

Figure 8: Execution time of the process runs on 20 processors

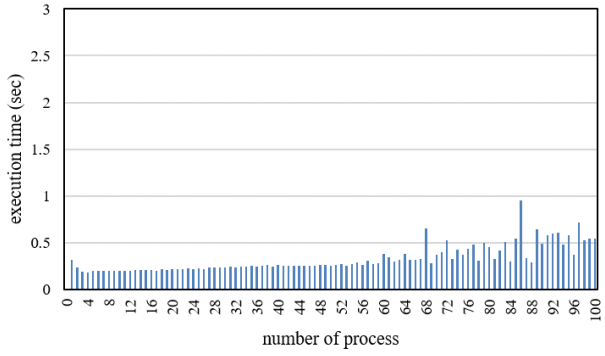

Figure 9: Execution time of the process runs on 40 processors

4.5 Discussion

Since the threads and processes created by performing pthread_create() or fork() in Experiment 1 and Experiment 2 perform the same processing, the number of threads or the number of processes is considered to be equivalent to the computational load. However, looking at 4.3.1, when threads were adopted, there was no correlation with execution time when the number of threads was increased. Also, when we checked which processor each thread was assigned to when executing the benchmark program, we confirmed that it was assigned to the same processor consecutively.

On the other hand, when a computational load is applied as a multi-process program, the relationship between the increase in the number of processes and the execution time is increasing proportionally. If the number of allocated processors exceeds the number of processes, it is possible to perform processing in a certain period. In addition, when comparing the cases where the same amount of calculation is performed, there is an inverse relationship between the calculation time and the case where the processor allocation is large and the case where the processor allocation is small.

For these reasons, it is difficult to estimate the execution time for the computational load when using threads. On the other hand, in the case of a process, a correlation was found, so we selected a target for a multi-process program and created and verified a prediction model of execution time.

5. Resource allocation predictive model

5.1 Creating a predictive model

Based on the above results, the relationship between the number of processes that one processor can execute. As we described in the 4.5, if the number of allocated processors exceeds the number of processes, it is possible to perform processing in a certain period. Therefore, it is not necessary to consider the case. To avoid the case, we omit the case of data from the original evaluation result data. Based on the remaining data, we develop an execution time calculation formula that was derived by multiple regression analysis. The result is shown in Equation 1. Table 2 shows the standard deviation, t-value, and p-value at this time. From the values, this formula is considered reliable.

Formula 1 estimation of program execution time

$$(execution\ time) T_{exec} = 0.11 \frac{Process}{Core} + 0.18$$

Table 2: result of multiple regression analysis

Process/Core: Number of processes processed per processor		
standard deviation	t-value	p-value
0.0064	17.172	2.798

Based on the formula, we considered that the accuracy of the estimation formula was sufficiently high and generalized as the predictive formula. To generalized the formula, we use α and β, instead of the variables 0.18 and 0.12 that are determined by the program in the multiple regression analysis respectively. Finally, we set the formula 2 for estimating the execution time of the entire program with an arbitrary number of jobs can be expressed.

Formula 2: estimation of software execution time

$$(execution\ time)\ T_{exec} = \sum_{i=0}^{jobs} \left(\alpha_i \times \frac{Process_i}{Core_i} + \beta_i \right)$$

In this equation, α and β are constants determined by the program, with the number of jobs included in the program as "*jobs*".

5.2 Model validation

In order to verify the created model, we assumed a software component named A that consists of the three software programs (a)(b)(c). The A is internally composed of programs that apply to (a) and (c). It is assumed that the requirement of the software component must be completed within 1 second. In order to find out how many processors should be assigned to this software to execute it in order to satisfy the requested response time, it was calculated by applying it to formula 2.

Table 3 programs composed of the software component A

program	Number of process	α	β
(a)	3	0.2	0.1
(b)	2	0.5	0.2
(c)	2	0.3	0.3

Using the formula 2, we solve the formula 3 for x_p, where the number of processors to allocate is x_p.

Formula 3 Apply to calculate the execution time with the parameters from the (a)(b)(c) program assignment.

$$1 = \left(0.2 \times \frac{3}{x_p} + 0.1 \right) + \left(0.5 \times \frac{2}{x_p} + 0.2 \right) + \left(0.3 \times \frac{2}{x_p} + 0.3 \right)$$

$$\therefore x_p > 5.5$$

From this formula, the execution time of the software component is less than 1 second in the case of executing on 6 processors or more.

6. Conclusion

In this study, we have constructed an efficient resource allocation prediction formula as a study of an efficient many-core allocation method for edge offload server. We developed an execution time prediction formula based on the measurement of the benchmark program. This can be used to calculate the minimum number of cores to run an application. Currently, the target of the prediction formula is a benchmark program, so this formula may not be applicable. In the future, we will apply the method to a practical robot service that consist of the application and consider applying a prediction formula to assign multiple cores dynamically for the requested applications.

Reference

[1] “Real world feedback and robots”, https://www.soumu.go.jp/ict_skill/pdf/ict_skill_1_4.pdf, (ref 2020/08/05)

[2] “Unibo | The world's first partner robot to learn individuality "unibo" | Unirobot Corporation”, https://www.unirobot.com/unibo/ , (ref 2020/08/05)

[3] Weisong Shi, “Edge Computing: Vision and Challenges”, IEEE INTERNET OF THINGS JOURNAL, VOL. 3, NO. 5, 2016

[4] Kashif Bilala,b, Osman Khalidb, Aiman Erbada, Samee U. Khan, “Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers,” Computer Networks Volume 130, pp.94-120, 15 January 2018.

[5] Weisong Shi, “Edge Computing”, PROCEEDINGS OF THE IEEE , Vol107, No. 8,2019

[6] Junpei Ogoshi ,Naofumi Hama ,Nobukazu Kondo, “Proposal and evaluation of DNN model operation method by cloud edge cooperation”, Proceedings of the 80th National Convention, 2018

[7] Hiroki Oka , Akimasa Yoshida, ” Task-driven coarse-grained parallel processing with local task co-execution on manycore”, Information Processing Society of Japan Journal Computing System Vol.12 No.3 1‐13, 2019

[8] Shinya Takamaeda Yoichi Mori Kise Kenji,”Examination of inter-core communication latency concealment method in manycore processor”, Proceedings of the 72nd National Convention, 2010

[9] Takuma Sumiya Yutaka matsubara Miyuki Nakano Midori Sugaya, “IXM:Rapid Inter-Process Communication Middlewarefor Robotics Software”, Information Processing Society of Japan Journal, Vol. 58-10, 2017.

[10] Masato Fukui Yoichi Ishiwata Takeshi Ohkawa Midori Sugaya, “Consideration of edge server that supports robot interaction”, Cloud Network Robot Study Group (CNR), 2020