Deep Learning Workload Analysis for Efficient Resource Allocation

Sayaka Takayama
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku
Tokyo, Japan
Email: sayaka-t@ogl.is.ocha.ac.jp

Takashi Shiraishi
Fujitsu Laboratories Ltd.
4-1-1 Kamikodanaka, Nakahara-ku
Kawasaki, Kanagawa, Japan
Email: shiraishi-ten@fujitsu.com

Shigeto Suzuki
Fujitsu Laboratories Ltd.
4-1-1 Kamikodanaka, Nakahara-ku
Kawasaki, Kanagawa, Japan
Email: shigeto.suzuki@fujitsu.com

Masao Yamamoto
Fujitsu Laboratories Ltd.
4-1-1 Kamikodanaka, Nakahara-ku
Kawasaki, Kanagawa, Japan
Email: masao.yamamoto@fujitsu.com

Yukihiro Watanabe
Fujitsu Laboratories Ltd.
4-1-1 Kamikodanaka, Nakahara-ku
Kawasaki, Kanagawa, Japan
Email: watanabe.y@fujitsu.com

Masato Oguchi
Ochanomizu University
2-1-1 Otsuka, Bunkyo-ku
Tokyo, Japan
Email: oguchi@is.ocha.ac.jp

***Abstract*—In recent years, with the prosperity of deep learning, Graphics Processing Units (GPUs) have become popular as hardware accelerators specialized for this purpose. However, compared to CPUs, which are general-purpose computing resources, GPUs are very scarce and valuable resources. Therefore, in this paper, we would like to consider some control that reduces GPU resource waste by determining GPU allocation based on the difference in application performance when using different GPUs. As a basic study, we evaluate the performance of 9 types of benchmarks executed on the framework using GPU and compare the performance when changing machine conditions. From this examination, it is judged whether the above control is possible. In addition, we estimate how much performance improvement can be expected by preferentially allocating GPUs with high performance to workloads that have a large impact on GPU performance using the data we collected. From this estimate, it is found that GPU priority control can reduce the total execution time by 8.24%.**

Keywords–Workload analysis; MLPerf; Zabbix; Deep learning.

I. INTRODUCTION

Graphics Processing Units (GPUs), processors designed for 3D graphics applications that require enormous computational processing, possess a large number of computing cores and memory that enables high-speed communication, and are good at parallel processing. With the prosperity of deep learning, GPUs became popular as specialized hardware accelerators. However, compared with Central Processing Units (CPUs), which are general-purpose computing resources, GPUs are very scarce and valuable computing resources. Also, deep learning does not fit well with current configuration practices and deployment models, which assume a static allocation of GPUs for each user or framework regardless of utilization, performance and scalability [1].

Therefore, we would like to consider GPU allocation control based on the difference in application performance when using different GPUs. As a basic study, we evaluated the performance of 9 benchmarks executed on a framework that uses GPUs and compared the performance on machines of different generations. From this examination, it was judged whether the above control is possible.

Also, assuming that a physical machine is fully used by one application in Docker environments, the job execution time can be determined by allocating a high-performance GPU preferentially to a workload that has a large impact on GPU performance. We estimated whether this would lead to improvement using the data we collected. As a result, it was found that the total execution time was reduced by 8.24% when GPUs were assigned according to the difference in job performance due to the difference in GPUs, compared to when GPUs were evenly assigned.

The reminder of the paper is organized as follows.

In Section II, related works about operational methods based on resource performance differences and application characteristics are introduced. The overview of the experiment for performance evaluation and comparison of each benchmark is proposed in Section IV. In Section IV, the impact of hardware on the execution of AI applications, focusing on CPUs, GPUs, and memory are described. Finally, concluding remarks are provided in Section V.

II. RELATED WORK

Operational methods based on resource performance differences and application characteristics have already been proposed. Scheduling based on the impact of power variability for specific applications, taking into account performance and power consumption variations that occur during the manufacturing process of the CPU, is the largest compared to modern scheduling policies used in production clusters. The job turnaround time has been reduced by 31% and the power supply has been reduced by up to 5.5% [2].

Also, efficiency is an important consideration for large-scale High Performance Computing centers with a wide range of different applications and heterogeneous infrastructures. For the purpose of optimizing the usage rate and job waiting time of a cluster, a method for executing job simulation of a Portable Batch System (PBS) based cluster using a historical workload has been proposed [3].

Most of the data contained in Facebook are sent to the machine learning pipeline, and the system is selectively used for training using both GPUs and CPUs and real-time inference using CPUs. Additionally, in real-time reasoning, the required resources are different because of the size of input data, and the importance of feature analysis of AI workload

 ISBN: 978-1-61208-787-0

behavior based on machine learning is discussed in [4]. A toolkit called FBLearner has been developed for the purpose of simplifying the task of using Facebook machine learning and is processed by the CPU server and the GPU server with different resource designs according to the features of machine learning. FBLearner consists of three tools focused on different parts of the machine learning pipeline: FBLearner Feature Store, FBLearner Flow, and FBLearner Predictor. By utilizing an internal job scheduler, it allocates the resources on a shared pool of GPUs and CPUs and schedules jobs. Most of Facebook's machine learning training is performed through FBLearner.

GPU virtualization is a method for effectively using limited GPU resources. Bitfusion FlexDirect [1] is a virtualization layer that supports management of computing resources by combining multiple CPUs and GPUs into a single elastic cluster. FlexDirect is designed so that multiple workloads can be executed in parallel by slicing the GPU into a virtual GPU of any size, and achieves a significant reduction in GPU resources compared to conventional GPU solutions. Singularity [5], a Linux container for HPC developed by Lawrence Berkeley National Laboratory, implements the ability to share GPUs among multiple applications running in a virtual environment. A job scheduler is used for resource management.

CPUs and servers with automatic workload management functions to analyze the bottlenecks and allocate resources and distribute the connections have already been developed. We introduce two examples of tools and products that perform tuning based on workload. Intel's CPU architecture "Haswell" [6] executes DynamicVoltage and Frequency Scaling that switches the voltage and operating frequency according to the load on a CPU core or cluster basis. NVIDIA's "HGX-2" [7] is a server for the GPU neck jobs that has many processors and is specialized for AI workloads. Unlike a general IA server, HGX-2 can use up to 16 GPUs in a single server.

We analyze the characteristics of workload as a basic study of efficient operation technology of GPU resources.

III. Overview of the Experiment

In this study, performance evaluation and comparison of each benchmark are performed using MLPerf to analyze the hardware information at the time of the execution of application, which is a representative type of AI. We use MLPerf's nine benchmarks for performance evaluation. Zabbix [8] is used to acquire information. Information such as CPU, GPU, memory, and I/O is acquired at the time of benchmark execution by these commands for feature analysis for each benchmark. Information acquisition is performed at one minute intervals. Table IV shows the measurement conditions for each benchmark. The learning accuracy is changed to 15 only for the Recurrent Neural Network (RNN) translator (RT).

In particular, feature analysis is conducted focusing on CPU, GPU, and memory utilization. Table II shows the experimental environment.

An outline of the software used in this research is provided below.

A. MLPerf

MLPerf is an AI benchmark supported by companies, such as Google, Intel, Baidu, and NVIDIA. It aims to build a common set of benchmarks that enable the machine learning field to measure system performance for both training and inference for a variety of environments ranging from mobile devices to cloud services. This benchmark includes Image Classification (IC), Single Stage Detector (SSD), Object Detection (OD), Recommendation (RM), Sentiment Analysis (SA), Rnn Translator (RT), Translation (TL), Speech Recognition (SR), and Reinforcement (RI).

TABLE I. MEASUREMENT CONDITIONS OF EACH BENCHMARK

Benchmark	Epoch (step, iteration)	SEED	Job Time
IC	53200 (step)	1	3:01:13
SSD	11	1	3:09:12
OD	25000 (iteration)	3	2:23:26
RM	6	1	1:09:34
SA	100	1	1:22:50
RT	1	1	2:48:18
TL	17200 (step)	1	2:59:55
SR	1	1	10:53:32
RI	4	1	5:46:00

TABLE II. ENVIRONMENT

OS	ubuntsu 16.04
Server	FUJITSU Primergy RX2540 M4
CPU	Intel Xeon Skylake 2 sockets 20 cores 2.4GHz Gold 6148 150W
GPU	NVIDIA Tesla V100 16GB
Storage M2.SSD	290GB read 0.87GB/s write 1.75GB/s
Memory	192GB DDR4 2666MHz
Python	3.50
CUDA	9.2

1) Time-series Data: The following introduces the hardware time-series data when MLPerf is run.

IPMI can obtain infrastructure information such as power and temperature. Figure 1 shows the time-series data for the CPU temperature. The time-series data for the CPU and GPU power consumption is shown in Figure 2. Perf can obtain the CPU utilization and memory usage that can be acquired by OS, and nvidia-smi can acquire GPU utilization. These pieces of information can be utilized for static optimal design of device resources. The obtained time-series CPU and GPU utilization data are shown in Figure 3. The CPU and GPU memory utilization time-series data are illustrated in Figure 4. Iostat can obtain how much Input and Output occurs on the disk. The results of the time-series data for the disk I/O access are shown in Figure 5. PMU can obtain the number of instructions and cache miss rate from CPU event information. These pieces of information can be used for dynamic frequency design. The time-series data of the memory intensive index in Figure 6 uses the values obtained by the following equation.

$$intensive = \frac{(local + remote)}{inst} \times 100$$

In the above equation, "local" means the number of times that the local DRAM is accessed, and "remote" means the number of time that remote DRAM is accessed because of the failure to access L3 cache memory. "inst" indicates the number of instructions.

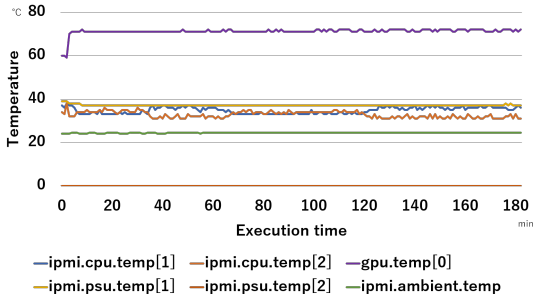

Figure 1. Temperature during the IC processing.

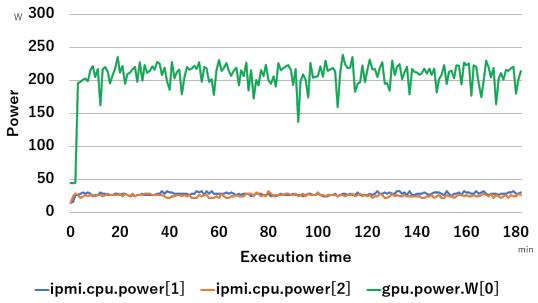

Figure 2. CPU and GPU power consumption during IC processing.

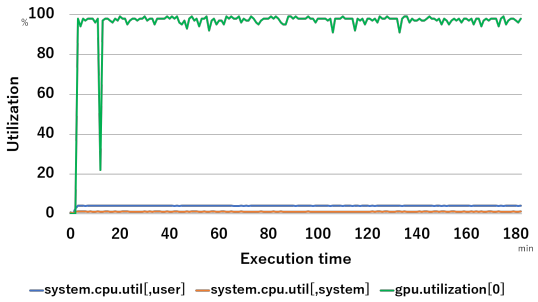

Figure 3. CPU and GPU utilization during IC processing.

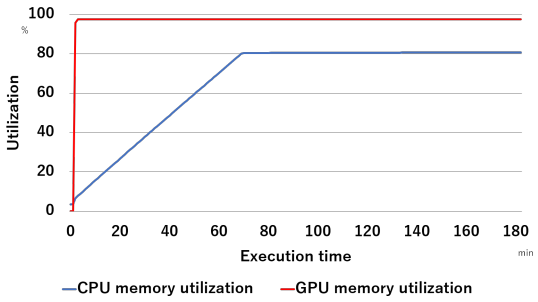

Figure 4. CPU and GPU memory utilization during IC processing.

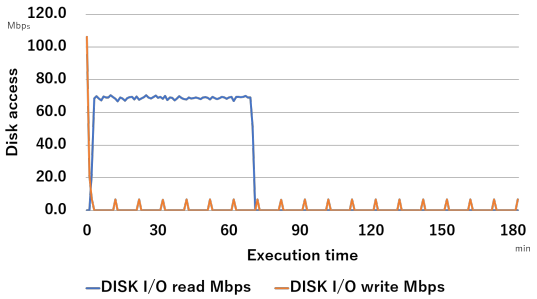

Figure 5. Disk I/O access during IC processing.

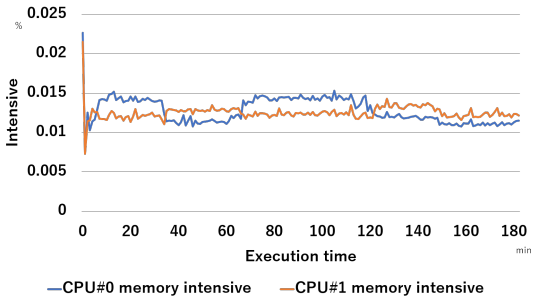

Figure 6. Intensive memory use during IC processing.

B. *Zabbix*

Zabbix is a flexible monitoring software and can add monitoring targets by using templates. We analyze the information that can be acquired by IPMI, Perf, nvidia-smi, and Performance Monitoring Unit (PMU) [9], which is a performance monitoring mechanism of the Intel CPU. PostgreSQL is used for data storage. We build the environment shown in Figure 7.

Since acquiring information on the benchmark execution server will influence the results, the server used for acquiring information and the server used for the execution of the benchmark are separated.

IV. Experimental Results

In this section, we investigate the impact of hardware on the execution of AI applications, focusing on CPUs, GPUs, and memory.

A. *Server*

We compare the results by running MLPerf on different generation servers. These servers are similar but have different performance CPUs. First, we measure the speed of disk access in the environment of Table II. Table III shows the maximum speed of disk access for each benchmark execution. The maximum disk access speed of FUJITSU Primer RX2540 M4 used in the experiment is 0.87 GB/s for reading and 1.75 GB/s for writing.

Since the maximum value of the disk access speed at the time of benchmark execution is sufficient for the server disk performance and the time required for disk access is extremely short compared to the job time, it is possible that the disk performance difference has a small impact on the benchmark.

We consider the change in the job execution time when using the server with the different CPU. For comparison, we

 ISBN: 978-1-61208-787-0

use two servers with different specifications. the specifications of the servers used for CPU comparison are shown in Table IV.

We compared IC, SSD, OD, RM, SA, RT, TL, and RI. Table V lists the comparison results of the job times required when executing MLPerf on the environments of Table IV. We also obtained CPI information, which represents the number of the clocks required to execute one instruction, but we did not observe any major changes with the change of the server.

Figures 8 - 15 show the time-series data for the clock frequency of each thread when each benchmark is executed on each server.

The time-series data of the clock frequency for “IC” are shown in Figure 8. On Skylake, two threads had a high clock frequency and their values changed alternately. The time-series data of the clock frequency for the “SSD” are illustrated in Figure 9. There were no large differences between these graphs. The results of the time-series data for clock frequency of “OD” are shown in Figure 10. This benchmark also showed no notable change. The time-series data for the clock frequency of “RM” are shown in Figure 11. Compared with Skylake, the clock frequency on Haswell tended to be high overall. Figure 12 shows the time-series data for the clock frequency of “SA”. In “SA”, only a specific thread had a high clock frequency, and this was considered to be a specification that only some threads were used when the frequency decreased. The results of the time-series data for the clock frequency of “RT” are shown in Figure 13. Only the clock frequency of a specific thread remained high on Skylake. The time-series data for the clock frequency of ”TL” are shown in Figure 14. On Skylake, some Specific threads showed noticeable changes, but on Haswell such changes were not observed. Figure 15 shows the time-series data for the clock frequency of “RI”. The trends of the changes were similar in Skylake and Haswell, but their periods were different.

Comparing the results obtained on Skylake and Haswell, it is observed that differences in the maximum values and other details are present, but the clock frequency shows some similar changes along the time series. We concluded that the impact of the CPU on the performance of AI applications is small.

From these results, differences in servers and CPUs are considered to have little impact on AI application performance.

B. GPU

This section introduces the analysis results on how GPU performance affects the performance of each AI application.

The average GPU utilization divided by the average CPU utilization is illustrated in Figure 16. Additionally, we note that the average utilizations of GPU and CPU per socket are shown in Table VI.

Large differences were observed in processor utilization for the different applications in the family of benchmarks. Table 16 and Table VI show that the overall application tends to be GPU-necked, and that translation-based applications require a particularly large amount of GPUs. On the other hand, there are also applications in which CPU performance is considered to be important, such as RM and SSD.

We compare the results obtained by running MLPerf with different generation GPUs. the specifications of the servers used for GPU comparison are shown in Table VII.

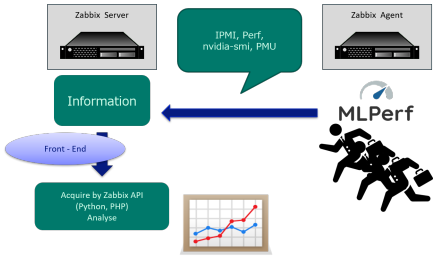

Figure 7. Zabbix environment for acquiring information.

TABLE III. DISK ACCESS SPEED

Benchmark	Read (MB/s)	Write (MB/s)
IC	73.92	111.32
SSD	6.75	10.96
OD	4.66	1.25
RM	2.67	29.26
SA	9.82	0.01
RT	29.50	0.06
TL	26.14	93.90
SR	7.10	8.05
RI	22.13	0.75

TABLE IV. ENVIRONMENTS FOR CPU COMPARISON

	Environment 1	Environment 2
OS	CentOS Linux release 7.5.1804 (Core)	ubuntsu 16.04
Server	FUJITSU Primergy CX400 M1	FUJITSU Primergy RX2540 M4
CPU	Intel Xeon Haswell 2 sockets 14 cores 2.6GHz E5-2697 145W	Intel Xeon Skylake 2 sockets 20 cores 2.4GHz Gold 6148 150W
GPU GPU	NVIDIA Tesla P100 16GB	NVIDIA Tesla P100 16GB
Storage HDD	270GB read 0.21GB/s write 1.07GB/s	290GB read 0.87GB/s write 1.75GB/s
Memory	256GB DDR4 2133MHz	192GB DDR4 2666MHz
Python	3.50	3.50
CUDA	9.2	9.2

TABLE V. COMPARISON OF THE JOB TIME - CPU

Benchmark	Haswell	Skylake	Skylake / Haswell
IC	4:36:10	4:33:04	0.99
SSD	3:56:42	3:12:18	0.81
OD	2:59:34	2:57:51	0.99
RM	1:12:00	1:09:07	0.96
SA	2:00:12	1:48:14	0.90
RT	4:06:41	4:05:28	1.00
TL	4:37:47	4:29:21	0.97
SR	-	15:07:34	-
RI	6:28:00	6:08:37	0.95

 ISBN: 978-1-61208-787-0

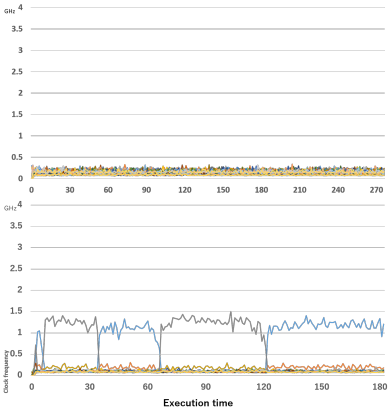

Figure 8. Time-series data for the clock frequency of IC (above:Haswell below:Skylake).

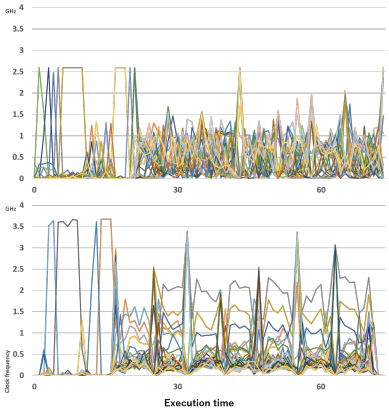

Figure 11. Time-series data for the clock frequency of RM (above:Haswell below:Skylake).

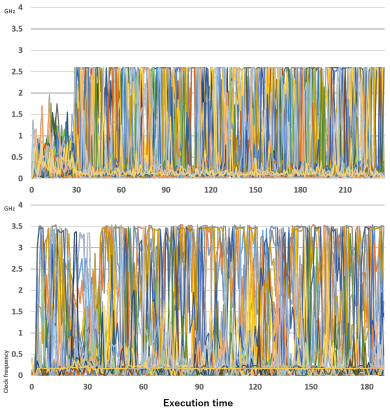

Figure 9. Time-series data for the clock frequency of SSD (above:Haswell below:Skylake).

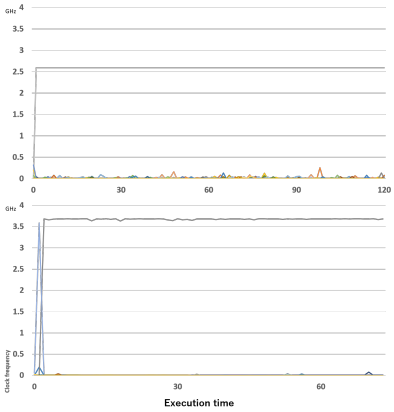

Figure 12. Time-series data for the clock frequency of SA (above:Haswell below:Skylake).

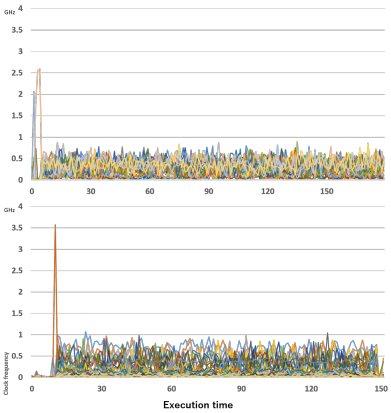

Figure 10. Time-series data for the clock frequency of OD (above:Haswell below:Skylake).

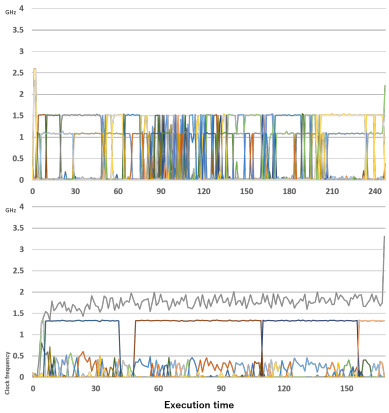

Figure 13. Time-series data for the clock frequency of RT (above:Haswell below:Skylake).

 ISBN: 978-1-61208-787-0

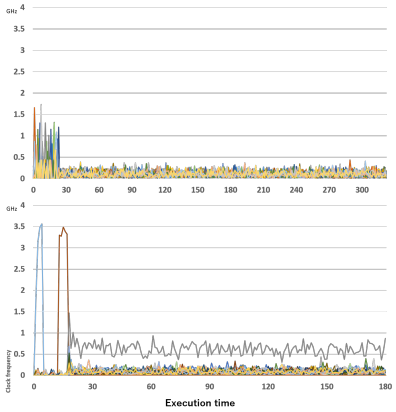

Figure 14. Time-series data for the clock frequency of TL (above:Haswell below:Skylake).

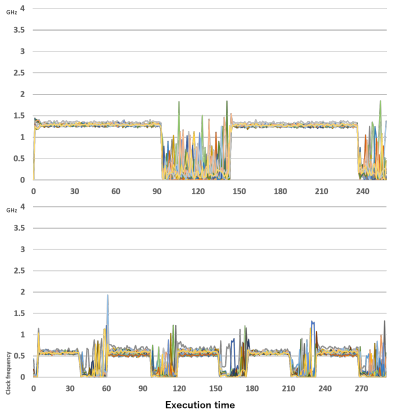

Figure 15. Time-series data for the clock frequency of RI (above:Haswell below:Skylake).

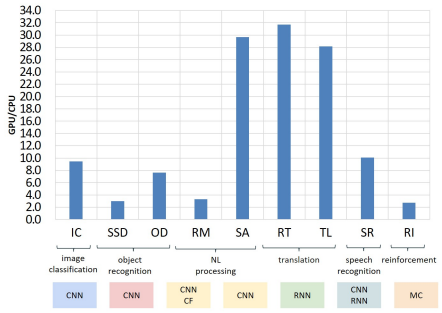

Figure 16. Average GPU/CPU utilization.

TABLE VI. GPU AND CPU AVERAGE UTILIZATION

Benchmark	CPU (%)	GPU (%)
IC	5.1	95.4
SSD	9.9	59.8
OD	4.9	74.5
RM	6.7	44.3
SA	1.4	82.0
RT	1.5	95.5
TL	1.5	83.9
SR	3.2	65.1
RI	11.4	62.7

TABLE VII. ENVIRONMENTS FOR GPU COMPARISON

	Environment 1	Environment 2
OS	ubuntsu 16.04	ubuntsu 16.04
Server	FUJITSU Primergy RX2540 M4	FUJITSU Primergy RX2540 M4
CPU	Intel Xeon Skylake 2 sockets 20 cores 2.4GHz Gold 6148 150W	Intel Xeon Skylake 2 sockets 20 cores 2.4GHz Gold 6148 150W
GPU	NVIDIA Tesla P100 16GB	NVIDIA Tesla V100 16GB
Storage HDD	290GB read 0.87GB/s write 1.75GB/s	290GB read 0.87GB/s write 1.75GB/s
Memory	192GB DDR4 2666MHz	192GB DDR4 2666MHz
Python	3.50	3.50
CUDA	9.2	9.2

TABLE VIII. GPU SPEC

	P100	V100
Core	3584	5120
MHz	1300	1455
FP16	18.636	119.19
FP32	9.318	14.90
FP64	4.659	7.45
Memory Bandwidth	720	900

Table VIII shows the specifications of the GPUs V100 and P100 used in this experiment.

Table IX lists the comparison of job times obtained when running each benchmark on different GPUs. Compared to the average GPU utilization data presented in Table VI, the change in the job time when changing the GPU is larger for the benchmarks with higher average GPU utilization.

The present result suggested that a job with a high GPU utilization shows a high job performance improvement effect due to changes in the GPU performance, and the difference in the job performance due to the changes in the CPU performance is small.

C. Memory

Figure 17 shows the maximum value of the memory utilization of each benchmark. Compared with the memory utilization of a CPU, the memory use of a GPU is remarkably large in this experiment. This is because the capacity of GPU memory is insufficient for the data size required by applications. Additionally, both the class “IC” and the class “TL” have high GPU utilization but show a large difference in

TABLE IX. COMPARISON OF JOB TIME - GPU

Benchmark	P100	V100	V100 / P100
IC	4:33:04	3:01:13	0.66
SSD	3:12:18	3:09:12	0.98
OD	2:57:51	2:23:26	0.81
RM	1:09:07	1:09:34	1.01
SA	1:48:14	1:22:50	0.76
RT	4:05:28	2:48:18	0.68
TL	4:29:21	2:59:55	0.67
SR	15:07:34	10:53:32	0.72
RI	6:08:37	5:46:00	0.93

 ISBN: 978-1-61208-787-0

the CPU memory utilization. This difference is caused by the data set type.

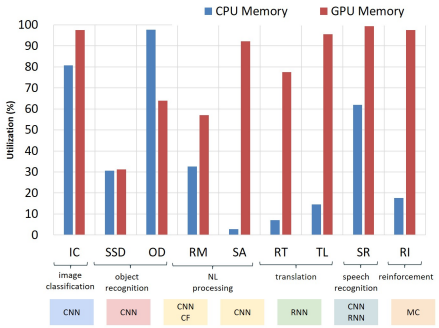

Figure 17. Memory utilization.

In this experimental environment and experimental conditions, the amount of total memory is sufficient for the operation of the application, and the effect on job performance is considered to be small.

D. GPU Priority Assignment

From the results of comparing job execution times when changing the GPU in Table VIII, we estimate the improvement in job time by GPU priority assignment.

In a Docker environment, it is assumed that one application fully uses a physical machine. Each machine that handles nine MLPerf benchmarks is assigned one of 10 P100 and 10 V100 as GPU resources, and process the same number of each benchmark in this assumption. Table X shows the comparison of the total execution time when a new GPU is preferentially assigned to a benchmark where the improvement in job execution time due to GPU performance was significant (Proposed Method) and the time when a GPU is evenly assigned (Evenly Assigned).

TABLE X. COMPARISON OF TOTAL EXECUTION TIME

Number of Jobs	Proposed Method (h:m:s)	Evenly Assigned (h:m:s)	Reduce (%)
200	74:30:00	81:11:27	8.241
400	149:00:00	162:22:54	8.241
600	223:30:00	243:34:21	8.241
800	298:00:00	324:45:48	8.241
1000	372:30:00	405:57:15	8.241
10000	3724:42:00	4059:32:33	8.248

Regardless of the number of jobs to be processed, improvement in job performance was confirmed at an almost constant rate. In the research works on task scheduling algorithm, the previous study [10] has shown that overall execution time was reduced by 1.2% - 8% over one-step and predictive average interval scheduling policies based on more accurate information prediction. Also, it was observed that the proposed algorithm resulted in 9.47% in terms of overall execution time of task completion in previous reseach [11]. These research works focus on the task scheduling of different target from ours.

From the simulation results, the control method proposed in this study is expected to reduce the total execution time by 8.24%, which can lead to efficient operation of limited GPU resources.

However, from the viewpoint of fairness of allocation among applications, the order of job processing should not necessarily be determined only by the type of benchmark, the order of job processing should not be determined only by the type of benchmark.

V. Conclusion and Future Work

In this paper, the efficient operation technology of GPU resources has been investigated using the difference of job performance caused by the difference of GPU performance. As a basic study, we evaluated the characteristics of 8 types of benchmarks executed on a framework that uses GPUs and compared performance on machines of different generations.

As a result of job performance analysis when GPUs of different generations were used, it was found that there is a big difference in job performance caused by the difference of GPU performance for each benchmark.

Assuming a Docker environment where old GPUs and new GPUs coexist, the execution time of the case where GPUs were evenly assigned was estimated and compared with that of the case where new GPUs were preferentially assigned to benchmarks with large differences in job performance due to differences in GPUs. The total execution time was reduced by 8.24%. This suggests that the above control leads to efficient operation of limited GPU resources. In the future, we would like to construct a system that actually controls the operation of the GPU.

VI. ACKNOWLEDGMENTS

This work is partially based on a collaborative research agreement between Ochanomizu University and Fujitsu Laboratories.

References

[1] bitfusion, “Bitfusion flexdirect virtualization technology white paper,” bitfusion, Tech. Rep., November 2017.

[2] Chasapis et al., “Power efficient job scheduling by predicting the impact of processor manufacturing variability,” in Proceedings of the ACM International Conference on Supercomputing. ACM, 2019, pp. 296–307.

[3] G. Zitzlsberger, B. Jansík, and J. Martinovič, “Job simulation for large-scale pbs based clusters with the maui scheduler,” Big Data Analytics, Data Mining and Computational Intelligence 2018, Theory and Practice in Modern Computing 2018, p. 137.

[4] K. Hazelwood et al., “Applied machine learning at facebook: A datacenter infrastructure perspective,” IEEE HPCA2018, 2018, pp. 620–629.

[5] L. B. N. Laboratory, “Singularity,” https://singularity.lbl.gov/, retrieved: April, 2020.

[6] Intel, “Haswell,” https://ark.intel.com/ja/products/codename/42174/Haswell.html, retrieved: April, 2020.

[7] NVIDIA, “HGX-2,” https://www.nvidia.com/en-us/data-center/hgx/, retrieved: April, 2020.

[8] Zabbix, “Zabbix,” https://www.zabbix.com/documentation/4.2/manual, retrieved: April, 2020.

[9] Intel, Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3B :Order Number 253669-067US., 2018, pp. 19.3–19–24, 19.46–19.58, https://software.intel.com/en-us/articles/intel-sdm.

[10] L. Yang, J. M. Schopf, and I. Foster, “Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments,” in Proceedings of the 2003 ACM/IEEE conference on Supercomputing. ACM, 2003, p. 31.

 ISBN: 978-1-61208-787-0

[11] Y. K. Ashish Kumar Mishra, Dhirendra K. Yadav and N. Jain, “Improving reliability and reducing cost of task execution on preemptible vm instances using machine learning approach,” The Journal of Supercomputing, vol. 75, no. 4, 2019, pp. 2149–2180.

 ISBN: 978-1-61208-787-0