Job Pipeline Mechanism of GPU Node for Multi-node Edge Computing (MEC) Systems

TAIKI MIYAKAWA†1 LI YANZHI†1
MIDORI SUGAYA†1

Abstract: A Multi-node Edge Computing (MEC) system that integrates hardware with different accelerators, such as GPUs and FPGAs, has been proposed for high-performance computing applications with low power consumption. Generally, a GPU node that connects to the MEC system provides a huge computation resource for AI applications. However, the current implementation of the MEC system executes jobs serially, which causes GPU utilization to decrease when processing multiple jobs. Moreover, there is a lack of a mutual-execution mechanism that avoids resource competition on the GPU, which causes abnormal program termination and may result in incorrect processing. To improve the efficiency and reliable execution of the multiple jobs on the GPU node, we propose a job pipeline mechanism that receives multiple jobs from the requester and executes them as node programs parallelly cooperate with the CPU. Also, we propose a mechanism that provides mutual exclusion to avoid abnormal termination of the jobs. In the evaluation, we found the efficient use of the GPUs of the system and avoid abnormal termination despite the multiple job assignment.

Keywords: GPGPU, Parallel processing, Multi-node

1. Introduction

Japan's Fifth Science and Technology Basic Plan proposed Society 5.0 in 2016 [1]. In Society 5.0, servers would process large amounts of data and make it possible to host applications that require low-latency and high-capacity computing resources that can satisfy the advanced AI processing to support various activities of the human-centric society.

To satisfy the requirement for the Society 5.0 applications, multi-node computing systems utilizing high-performance computing are discussed [2, 3]. As one of the multi-node computing systems, Multi-node Edge Computing (MEC) system is highly expected, which integrates high-performance hardware with different accelerators, such as GPUs and FPGAs, for utilizing the Society 5.0 applications with low latency. Within the system, each of the multiple computers (nodes) is equipped with an accelerator, which performs a large amount of computation by taking advantage of the accelerator's characteristics. These nodes are connected through a network and form one extensive system. The system has high computational power and responsiveness and is well suited for edge computing, which provides high processing power as required by Society 5.0 applications [4].

2. Related Studies and Problems

In this section, we first describe the Graphics Processing Units (GPUs) containing multiple arithmetic cores with a simple structure. Then we describe the problems that need to be solved.

2.1 A multiple GPGPU cores servers for the MEC node

Figure 1 illustrates the multi-node mechanism that integrates the multiple GPUs and FPGAs, and middleware that Li et al. developed for the MEC system [5]. In this system, there is a job allocation server that accepts the job from the client and allocates the job to the appropriate node that is organized as the GPUs and FPGAs. Within this node, GPU provides a huge computation resource that can be executed using high computation jobs such as machine learning.

GPUs were originally developed for image processing. General-Purpose computing on Graphics Processing Units (GPGPU) is a technology that applies this technology to general-purpose computing. The simple structure of GPUs makes them easier to scale up compared to general-purpose CPUs. Various applications such as machine learning have been accelerated by GPGPU. Efforts are also being made to make GPGPU processing as efficient as possible by optimizing algorithms for parallel data processing. Moreover, GPU memory (VRAM) uses high-bandwidth memory, such as GDDR6X and HBM3. Under these structures, exact instructions are executed for multiple data in parallel.

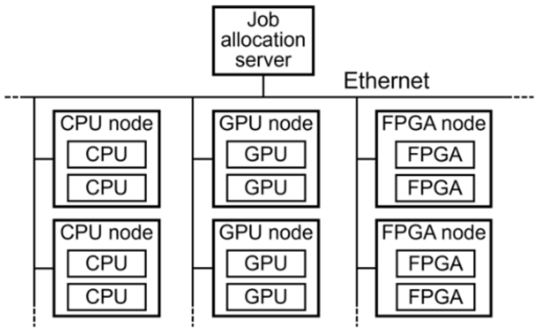

Figure 1: Multi-node computing system [5]

2.2 Problems

As we described above, the multiple GPU provides huge computing power for the MEC system. However, there are following two problems to support the MEC system. In this section, we describe the detail of the problems.

- **Inefficient job allocation**

Figure 2 shows the conventional implementation of the GPU node client program in the MEC system. Inside the GPU node, the job executor handles the assigned job from the job allocation server. The job executor selects one of the application programs to execute according to the job context. The application program

†1 Shibaura Institute of Technology

calls the GPU library to issue a GPU kernel. When the GPU kernel finishes the calculation, the application program collects the calculation result and posts it to the job executor. When the job executor obtains the result from the application program, it posts it to the job allocation server and waits for the next job assignment.

Figure 3 shows the job processing flow of the conventional implementation of the middleware. In the figure, two jobs are sent from the client. Through the network, the required data of Job 1 is downloaded to the GPU and executed, then the result of the job is uploaded to the client. The execution of the network proceeded on the CPU.

As shown in Figure 3, there is an idle time in GPU activity after Job 1's execution is finished and until the next, Job 2, is downloaded from the network. During the execution on the network, the GPU node is idle and does not use the GPU resource efficiently.

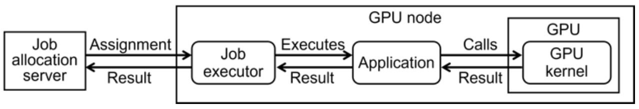

Figure 2: The implementation of the GPGPU node in the MEC system

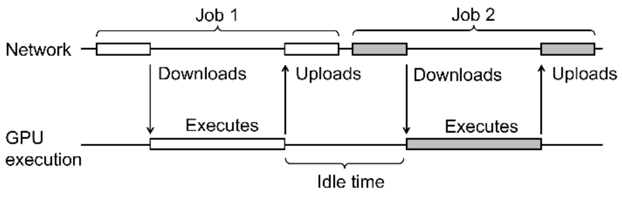

Figure 3: Conventional job processing flow

- **Abnormal termination during parallel execution**

Another problem is that the GPU does not have a mutual-exclusion mechanism, which causes abnormal program termination and may result in incorrect processing. For example, when programs are being executed in parallel, VRAM may run out and no more space can be allocated for new programs. In such cases, the program may terminate abnormally [6].

3. Proposed Design and Implementation

To solve the problems described in the previous section, we propose an efficient job allocation mechanism for GPU for MEC Systems. To achieve this, we firstly propose a mechanism to pipeline the jobs to improve the processing efficiency on GPU. Second, we propose a mechanism to suppress abnormal termination.

3.1 Improvement of processing efficiency

There is idle time on GPU after the execution and waiting for the next job. We consider that the pipeline mechanism is suitable to solve the problem that can execute the second job suspension on the CPU and download it quickly after the execution on the GPU. We illustrate the pipeline mechanism to the GPU node in Figure 4. It shows the pipeline job processing flow that deletes the idle time on the GPU. To support the download of the job suspension mechanism, it is possible to achieve the purpose.

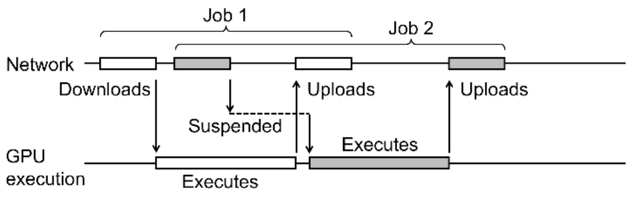

Figure 4: Proposed job processing flow

In this study, we developed the spawn and suspension mechanism on the CPU that makes it possible to execute the jobs on GPU parallelly. To achieve this, we use the fork() and waitpid() system calls in the implementation. Figure 4 illustrates that the execution of each program reduces the unused time on the GPU and improves utilization efficiency.

3.2 Suppression of abnormal termination

We propose a mechanism to avoid abnormal termination. Wang et al. mentioned the problem that conflicting demands for GPU resources (like VRAM) can cause applications to crash. We had the same problem that executing multiple applications parallelly without any kind of resource managing mechanisms caused the application's abnormal termination [6]. To solve the problem, we implemented an exclusive lock on the GPU using flock (file lock). It guarantees only one execution to the GPU. This makes it possible to suspend the execution of the process which has a lock until the process has finished and released the exclusive lock. When the exclusion lock is released, execution resumes.

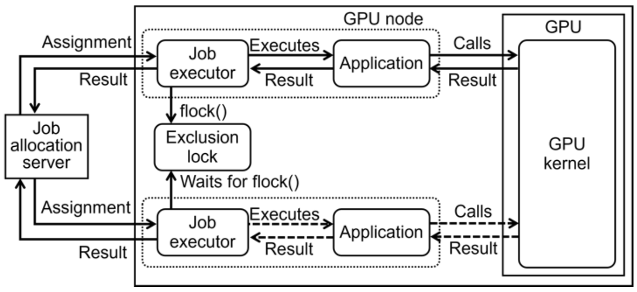

Figure 5: Proposed implementation of GPU node in multi-node computing systems

4. Evaluation experiment

We set up the evaluation experiment to compare the conventional system that assigns only one job at a time and the proposed system that applies the proposed design described in Section 3.

Table 1 shows the configuration of the system used in the experiment. NVIDIA® Jetson Xavier NX™ Development Kit was used in this experiment. The small size and low power consumption make it ideal as an edge device, and it can run programs written in CUDA, OpenCL, and other languages.

Table 1: Experimental system configuration

Component	**Specification**
CPU	NVIDIA Carmet ARM®v8.2, 6 Cores
GPU	NVIDIA Volta™ architecture
Main memory	8GB LPDDR4X
OS	NVIDIA Jetson Linux 32.7.1 (Jetpack 4.6.1)
Ethernet	1000BASE-T Ethernet

To evaluate the system in this experiment, we performed two experiments for two types of tasks.

4.1 Experiment 1: Image processing task

In Experiment 1, we used an image super-resolution program based on a convolutional neural network. The super-resolution program takes image data as input, performs super-resolution processing, and outputs the resulting image data.

In the evaluation, the five jobs were allocated consecutively for each experiment and measured the time from issuing the first job to all the results returned. We performed ten experiments and obtained the average and maximum values. We performed this experiment using images of different resolutions and compared the results between the conventional and proposed systems.

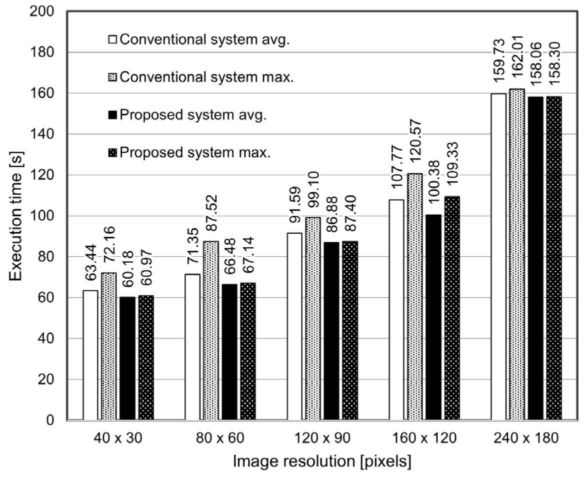

Figure 6: Experiment 1 results

Figure 6 shows the experimental results. The results show that the proposed system reduced the average execution time by approximately 1 - 7% and the maximum by approximately 2 - 23% compared to the conventional system. Moreover, because no abnormal termination of the program occurred during the experiment, it is considered that the suppression of abnormal termination proposed in 3.2 above is effective.

4.2 Experiment 2: Video processing task

In Experiment 2, to clarify the relationship between the file size of the job and the performance of the proposed method, we measured the time of video transcoding of FFmpeg. The video processing task requires transferring the video file, which takes longer than the image processing task performed above due to the size of the files. We thought that the performance of the video processing task can be improved much more by the proposed system.

Five jobs were issued consecutively per experiment. Time measurements and statistical methods are the same as in Experiment 1. This experiment was performed using videos of different resolutions and compared the conventional and the proposed systems.

Figure 7 shows the experimental results. The results show that the proposed system reduced the average execution time by approximately 6 - 24% and the maximum by 4 - 54% compared to the conventional system. The percentage of improvement tended to be higher at lower resolutions.

In some areas of the results, such as the maximum value of 640 x 360 pixels, significant differences were observed. In the current system, there is sometimes a delay in job allocation. In the conventional system, there was a period with no processing between the time a job was finished and the time a new job was allocated. On the other hand, the proposed system processed other jobs during the delay in job assignment, mitigating the delay.

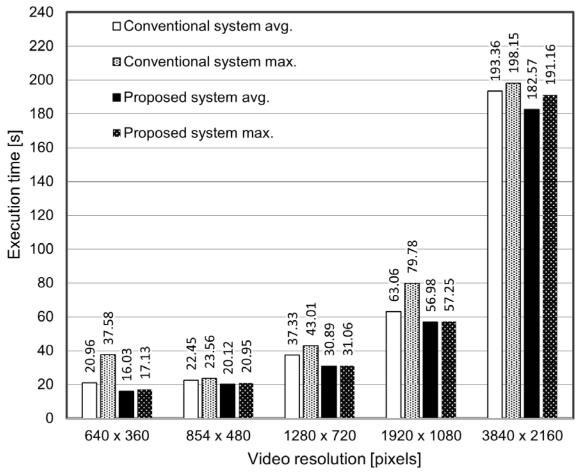

Figure 7: Experiment 2 results

5. Conclusion

We proposed to improve the utilization efficiency of GPU nodes in a multi-node computing system by parallel processing. Evaluation experiments have shown that parallel processing can reduce the time required to complete a job and improve utilization efficiency.

However, the proposed method has several problems to be addressed. The first is to improve the method. The current system has a fixed number of parallel executions. We would like to develop a method to dynamically change these limitations and improve the system to process applications more efficiently.

Another problem is that we need to perform additional evaluations of the system using other indicators. We would like to evaluate the system using multiple indicators, such as the average response time of each job or power consumption.

After these efforts, some problems need to be addressed to develop the system into a platform. In the current system, only the processing efficiency of individual nodes was considered. We would like to propose a method to increase the processing efficiency of the entire system by scheduling according to the conditions of other nodes.

We would also like to propose methods to improve processing efficiency in systems that include accelerators other than GPUs, such as FPGAs, ASICs, and SmartNICs (including DPUs and IPUs), by taking advantage of the characteristics of each accelerator.

Acknowledgments This research was supported by JST, CREST, and JPMJCR19K1.

References

[1] Society 5.0, Science and Technology Policy, Cabinet Office, Japan, https://www8.cao.go.jp/cstp/society5_0/ (July 24, 2022.)

[2] Zhe Fan, Feng Qiu, A. Kaufman and S. Yoakum-Stover, "GPU Cluster for High Performance Computing," SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, 2004, pp. 47-47, doi: 10.1109/SC.20 04.26.

[3] Miho Yamakura, Ryousei Takano, Akram Ben Ahmed, Midori Sugaya, Hideharu Amano "A Multi-tenant Re-source Management System for Multi-FPGA Systems", IEICE Trans. Information and Systems, Vol.E104-D, No.12, Dec. 2021.

[4] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," in IEEE Internet of Things Journal, vol. 4, no. 5, pp. 1125-1142, Oct. 2017, doi: 10.1109/JIOT.2017.2683200.

[5] Li Yanzhi and Midori Sugaya, "Resource management system for mixed multi-FPGA and GPGPU environments" in SWoPP2022, Jul. 2022.

[6] Kaibo Wang, Xiaoning Ding, Rubao Lee, Shinpei Kato, Xiaodong Zhang, GDM: device memory management for gpgpu computing, ACM SIGMETRICS Performance Evaluation Review, Volume 42, Issue 1, June 2014.