SUGISAKA MASANORI

Journal of Robotics, Networking and Artificial Life
Vol. 10(2); September (2023), pp. 105–114
ISSN (Online): 2352-6386; ISSN (Print): 2405-9021
https://alife-robotics.org/jrnal.html

Research Article

Optimizing a Field-Programmable Gate Array Object Detection System Considering Processing System and Programmable Logic Load Balance

Yusuke Watanabe[1], Hakaru Tamukoh[2]

[1]CRAFT WORK Co., Ltd, 5F OS Bldg., 3-5-15 Shibasaki-cho, Tachikawa, Tokyo, 190-0023, Japan, Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, 2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan
[2]Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, 2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan

ARTICLE INFO

Article History

Received 28 November 2022
Accepted 13 November 2023

Keywords

FPGA
Zynq
PL
PS
Multi-threading
System Performance

ABSTRACT

A field-programmable gate array (FPGA) device with a Zynq architecture integrates a processing system (PS) and programmable logic (PL) into a single chip. Although the PL performance is typically considered, the PS load cannot be completely ignored. In this study, using an FPGA board with a Zynq architecture, the conditions under which an object detection system performs the best, while considering the PS and PL load balance, are explored.

1. Introduction

In robotics, the demand for running neural networks such as object detection, on a low energy consumption device is extremely high. Consequently, the relationship between neural networks and energy consumption is being studied extensively [1], [2]. Field programmable gate arrays (FPGAs) meet these requirements as they are well known for their low energy consumption [3], [4] Owing to recent advancement in FPGAs, architectures such as Zynq, which featured the integration of the software programmability of a processing system (PS) and the hardware programmability of programmable logic (PL) into a single device have become popular.

When FPGA devices are employed, focus is typically placed on the PL performance regarding system performance [5], [6], however, the PS performance cannot be ignored completely. Although numerous studies on PS and PL communication have been conducted [7], [8], [9]. PS load was hardly considered. Currently, numerous studies highly focus on PL and neural networks [10], [11], [12]. The larger a neural network in a system becomes, the larger both its PS and PL load tend to be. Therefore, in this study, the impact of the PS load on system performance, including PS and PL communication, is investigated by measuring the execution time. Considering both the PS and PL performances concurrently based on the results, this study aims to improve the overall system performances to ensure that the system is more optimized than those focusing only on either PS or PL performance optimization. This is an extended paper submitted to the 2023 International Conference on Artificial Life and Robotics [13].

Corresponding author's E-mail: watanabe.yusuke898@mail.kyutech.jp, tamukoh@brain.kyutech.jp URL: http://www.lsse.kyutech.ac.jp/english/

2. Method

The object detection system used in this study to investigate the influence of PS load on PL and system performances is shown in Fig. 1. Herein, the object detection in PL is implemented based on [14] and [15], and performs both convolution and max pooling operations which are widely used in neural networks.

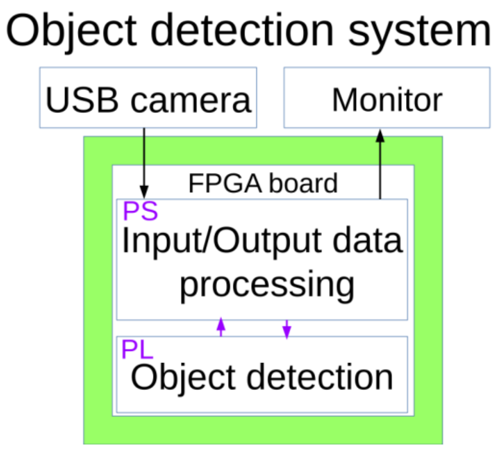

Fig. 1 Proposed object detection system on an FPGA board

The details of the layers in the neural network are based on Tiny YOLOv2 [16] and are listed in Table 1. From left to right of the columns in the table, Layer represents the type of layer, Input width and height represent the height and width of input feature maps to the layer, respectively. Input channel represents the number of input feature maps to the layer, and Conv. and Max pool. in the Layer column represent the convolution and max pooling layer, respectively. From the top to the bottom of the rows in the Layer column, input data to the PL in the system are passed and processed sequentially.

Table 1. Layers for object detection in PL

No.	Layer	Input width	Input height	Input channel
1	Conv.	384	288	3
2	Max pool.	384	288	16
3	Conv.	192	144	16
4	Max pool.	192	144	32
5	Conv.	96	72	32
6	Max pool.	96	72	64
7	Conv.	48	36	64
8	Max pool.	48	36	128
9	Conv.	24	18	128
10	Max pool.	24	18	256
11	Conv.	12	9	256
12	Max pool.	12	9	512
13	Conv.	12	9	512
14	Conv.	12	9	1,024
15	Conv.	12	9	512

Following our previous work [17], we implement the neural network for object detection in PL in C++ to achieve optimal performance. Channel dimensional bitwise operations and UNROLL pragmas are adopted, along with PIPELINE pragmas for further improvement [18]. Consequently, approximately 3,049,770 minimum clock cycles and 5,153,242 maximum clock cycles are estimated. Table 2 presents the total resource usage of the PL, where in the Resource row, BRAM denotes block random-access memory, DSP denotes digital signal processing, FF denotes flip-flop, and LUT denotes a lookup table, respectively.

Table 2. Resource usage in PL

Resource	BRAM	DSP	FF	LUT
Usage	1,574	1	151,079	163,491
Available	1,824	2,520	548,160	274,080

The system is run on an FPGA board with a Zynq architecture, to which a USB camera and monitor are directly connected. An image file and USB camera image are used as the input data for the proposed system. The PS load in the system is varied by switching between the two input methods. After receiving the input data, the system detects the objects and passes an output image on which bounding boxes are drawn to the monitor. The execution time of both object detection in PL and inference in the application are measured to evaluate the performance of the proposed system. Inference time represents the period from obtaining the input image data to displaying the output image data.

An experiment containing six configurations is conducted, and the execution time is collected and compared to evaluate the impact of the PS load on the object detection system performance.

3. Experiment

A Zynq UltraScale+ MPSoC ZCU102 evaluation board, Logicool C270 HD WEBCAM and BenQ GW2480T are employed as the experimental environment, and an experiment executing the following applications is conducted.

(i) FPGA application using a sub-thread for output images.
(ii) C++ and FPGA applications.
(iii) FPGA application using only the main thread.
(iv) FPGA application using a sub-thread for input images.
(v) FPGA application using three threads.
(vi) FPGA application using over four threads.

The FPGA applications in all cases are generated using the AMD Xilinx tool, and completely the same object detection in PL and other operations in PS are performed, as shown in Fig. 1. Although operations in the PL are uniform across all applications, operations in the PS vary in each application. In addition to switching between the two input methods, the number of threads in the PS is increased to change the PS load as required. For execution time, both CPU time and wall time are measured using the C library clock() and C library clock_gettime() functions, respectively. CPU time refers to the time when the CPU is busy processing the program's instructions, whereas wall time refers to the time elapsed during the measurement. The OpenCV imshow() function is used to display output images from the PS in the system.

Fig. 2 represents an overview of the processing of the

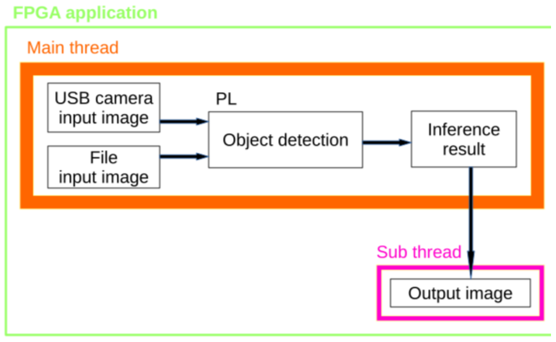

Fig. 2 Overview of the case one application

FPGA application executed in the case one. The two input methods are switched each other by changing the arguments passed to the application. The pthread_create() function in the POSIX thread libraries is called to display an output image in a sub-thread, and all sub-threads are created using the pthread_create() function.

Fig. 3 represents an overview of the case two. The C++

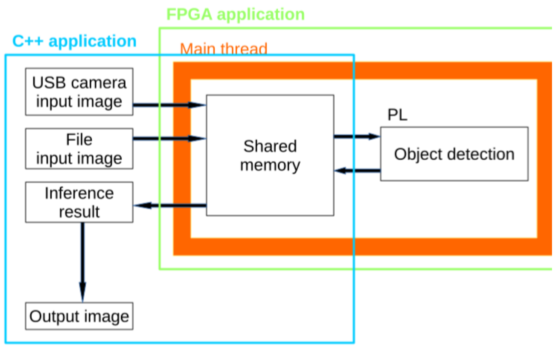

Fig. 3 Overview of the case two application

application generated by Cmake is executed to automatically execute the FPGA application in it. Shared memory is used to pass data between the C++ and FPGA applications. Note that regardless of the input method, the input data to the FPGA application are passed through the shared memory, therefore, only the image file is employed to measure the execution time. In the case two, the execution time of the FPGA application, which is the time from the beginning of the FPGA application to immediately before the end of the FPGA application, is also measured.

Fig.4 shows an overview of the case three, which is almost similar to that of the case one. However, it differs from the case one application in that the function for displaying an output image is called in the main thread rather than in a sub-thread.

Fig. 5 shows an overview of the case four. Obtaining input images from the USB camera is executed in a sub-thread. In the case four and other cases using a sub-thread to obtain the input data, the input data are not obtained from the image file because the data acquisition from the

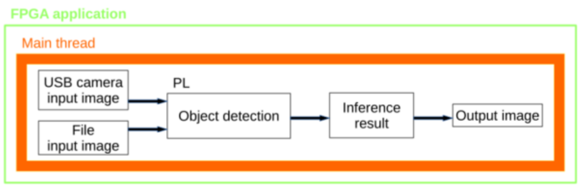

Fig. 4 Overview of the case three application

file is excessively light to properly evaluate the system performances.

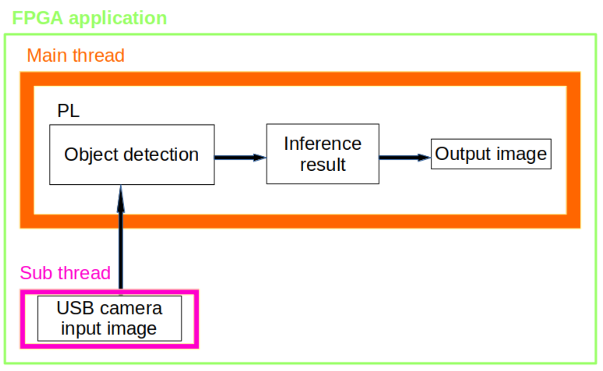

Fig. 6 Overview of the case four application

Fig. 6 shows an overview of the case five. A sub-thread is created for obtaining input images, as in the case four application, and an additional sub-thread is created for displaying the corresponding output images, as in the case one application. These two sub-threads are not the same thread but the different ones.

The case one, three, four, and five aim to determine whether processing the input or output images in a sub-thread is better for system performances.

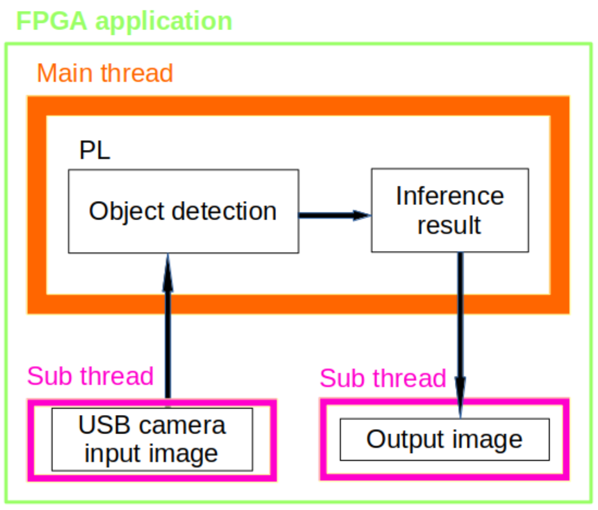

Fig. 7 Overview of the case five application

Fig. 7 represents an overview of the application in the case six. The application is basically the same as that in the case five. Infinitely idle sub-threads are created in addition to the threads in the case five. A state of infinitely idle refers to when a C while loop infinitely keeps running with nothing to do. The case six aims to elucidate the effect of the number of threads on the system performances. Although the total number of threads could be easily varied, a minimum of 10, 100, 1,000, and 10,000 are chosen to investigate the PS load of threads to ease in comparing the execution time. The proposed system can have a maximum of 25,961 threads.

Each application in the six cases is executed ten times, and the execution time is recorded. When the USB camera is used to obtain input data, the time when any

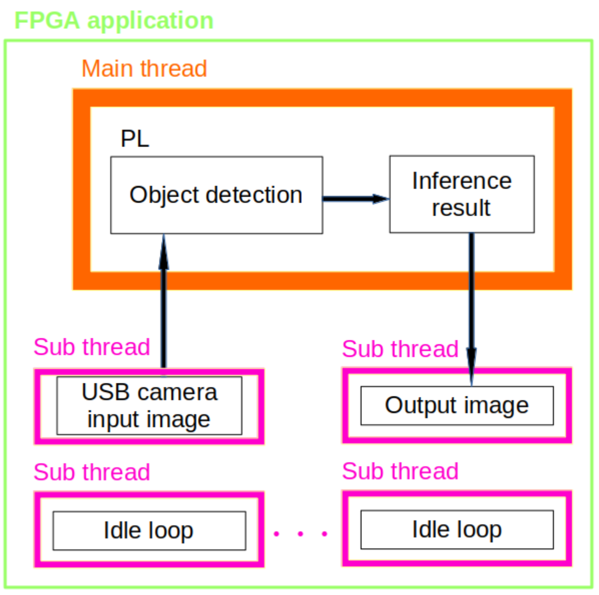

Fig. 5 Overview of the case six application

object is detected is regarded as the execution time. A specific image file is prepared such that fixed objects are constantly detected by the system, to acquire input data from it when required.

4. Results

Table 3 to Table 14 represent the experimental results, where Min. denotes the minimum execution time and Max. denotes the maximum time, and Ave. denotes the meantime of ten measurements. The time is shown in seconds. Fig. 8 and Fig. 10 respectively represent figures of average inference CPU and wall times. Fig. 9 and Fig. 11 respectively represent figures of average PL CPU and wall times.

Table 3. CPU time in the case one

Image input			Camera	File
CPU time (s)	Inference	Min.	1.344	5.978 x 10^{-1}
		Ave.	1.348	5.988 x 10^{-1}
		Max.	1.350	5.997 x 10^{-1}
	PL	Min.	1.042 x 10^{-1}	1.016 x 10^{-1}
		Ave.	1.046 x 10^{-1}	1.028 x 10^{-1}
		Max.	1.080 x 10^{-1}	1.048 x 10^{-1}

Table 4. Wall time in the case one

Image input			Camera	File
Wall time (s)	Inference	Min.	1.215	5.136 x 10^{-1}
		Ave.	1.219	5.145 x 10^{-1}
		Max.	1.221	5.161 x 10^{-1}
	PL	Min.	5.231 x 10^{-2}	5.271 x 10^{-2}
		Ave.	5.232 x 10^{-2}	5.273 x 10^{-2}
		Max.	5.233 x 10^{-2}	5.281 x 10^{-2}

Table 3 lists the CPU times required for the case one, and Table 4 lists the wall times in the case one. Evidently from Table 3 and Table 4, the PS load of the USB camera was higher than that of the image file because of the inference time difference, and the execution time in PL was almost similar in both input methods. The case one in the Fig. 8 and Fig. 10 shows the inference time difference between the USB camera and image file, while the case one in the Fig. 9 and Fig. 11 shows the similarity of the execution time in PL. These results indicate no impact of the PS load on the PL execution.

Table 5. CPU time in the case two

Image input			File
CPU time (s)	Inference	Min.	5.797
		Ave.	5.808
		Max.	5.817
	FPGA application	Min.	4.071
		Ave.	4.073
		Max.	4.074
	PL	Min.	5.285 x 10^{-2}
		Ave.	5.294 x 10^{-2}
		Max.	5.301 x 10^{-2}

Table 5 lists the CPU times required for the case two, here, the wall time is not presented because both the CPU and wall time were almost similar. As shown in Table 5, the CPU times in the PL were shorter than those in the case one application, although the inference CPU times were longer, and the wall times in the PL were almost similar. A comparison between the case one and two in Fig. 9 shows the shorter CPU times of the case two in the PL. A comparison between the case one and two in Fig. 11 shows the similarity of the wall times in the PL.

Table 6. CPU time in the case three

Image input			Camera	File
CPU time (s)	Inference	Min.	1.258 x 10^{-1}	5.067 x 10^{-1}
		Ave.	1.279 x 10^{-1}	5.076 x 10^{-1}
		Max.	1.285 x 10^{-1}	5.082 x 10^{-1}
	PL	Min.	5.218 x 10^{-2}	5.262 x 10^{-2}
		Ave.	5.227 x 10^{-2}	5.272 x 10^{-2}
		Max.	5.231 x 10^{-2}	5.281 x 10^{-2}

Table 7. Wall time in the case three

Image input			Camera	File
Wall time (s)	Inference	Min.	1.209	5.129 x 10^{-1}
		Ave.	1.213	5.142 x 10^{-1}
		Max.	1.236	5.153 x 10^{-1}
	PL	Min.	5.231 x 10^{-2}	5.272 x 10^{-2}
		Ave.	5.232 x 10^{-2}	5.275 x 10^{-2}
		Max.	5.232 x 10^{-2}	5.282 x 10^{-2}

Table 6 lists the CPU times required for the case three, and Table 7 lists the wall times in the case three. Evidently, both the CPU and wall times of the image file input were almost similar. Both the inference and PL CPU times in the case three were shorter than those of the case one, and the wall times were identical to those of the case one application. Comparisons between the case one and three in Fig. 8 and Fig. 9 show the shorter CPU times of the case three. Comparisons between the case one and three in Fig. 10 and Fig. 11 show the similarity of the wall times. Based on these results, an output image should be displayed in the main thread to reduce CPU time when the main thread is used to obtain the input data.

Table 8. CPU time in the case four

Image input			Camera
CPU time (s)	Inference	Min.	2.398 x 10^{-1}
		Ave.	2.458 x 10^{-1}
		Max.	2.519 x 10^{-1}
	PL	Min.	1.041 x 10^{-1}
		Ave.	1.043 x 10^{-1}
		Max.	1.044 x 10^{-1}

Table 9. Wall time in the case four

Image input			Camera
Wall time (s)	Inference	Min.	1.206 x 10^{-1}
		Ave.	1.242 x 10^{-1}
		Max.	1.385 x 10^{-1}
	PL	Min.	5.231 x 10^{-2}
		Ave.	5.237 x 10^{-2}
		Max.	5.242 x 10^{-2}

Table 8 lists the CPU times required for the case four, and Table 9 lists the wall times in the case four. Evidently from Table 6 and Table 8, although the CPU times in the case four were longer than those in the case three, the inference wall times were shorter. Comparisons between the case three and four in Fig. 8 and Fig. 9 show the longer CPU times of the case four. A comparison between the case three and four in Fig. 10 shows the shorter wall time of the case four for inference. Evidently from Table 7 and Table 9, the wall times in the PL were almost similar. A comparison between the case three and four in Fig. 11 shows the similarity of the wall times in the PL. Based on these results, the input images from the USB camera should be acquired in a sub-thread to save inference wall time.

When we pay attention to the PL time, we realize the following points.

(i) The PL CPU times in both the case one and four are almost similar, evidently from Table 3 and Table 8. A comparison between the case one and four in Fig. 9 shows the similarity of the CPU times in the PL.

(ii) The PL CPU times in both the case two and three are almost similar, evidently from Table 5 and Table 6. A comparison between the case two and three in Fig. 9 shows the similarity of the CPU times in the PL.

(iii) The PL CPU times in both the case one and four are longer than those in both the case two and three. A comparison among the case one, two, three, and four in Fig. 9 shows the longer CPU times of the case one and four in the PL.

(iv) The number of threads in both the case one and four is more than that in both the case two and three.

These four points suggest that the number of threads affected the PL CPU time.

Table 10 and Table 12 respectively list the CPU times required for the case five and six, and Table 11 and Table 13 respectively list the wall times in the case five and six. Table 12 and Table 13 list the execution time corresponding to the total number of threads in each column. Regarding the execution time when the number of threads is 1,000, no objects are detected because the PS load could be confirmed without detecting any object in the system. Furthermore, the execution time for 10,000 threads could not be acquired owing to module errors. Additionally, the execution time of eight and nine threads were measured to determine the conditions under which the PL CPU time starts affecting the PL wall time.

Evidently from Table 8 and Table 10, the CPU times for both inference and PL in the case five were longer than those in the case four. Comparisons between the case four and five in Fig. 8 and Fig. 9 show the longer CPU times of the case five. Evidently from Table 9 and Table 11, the PL wall times in both the case four and five were almost similar, whereas the infere*nce* wall times in the case five were slightly shorter than those in the case four. A comparison between the case four and five in Fig. 11 shows the similarity of the wall times in the PL, whereas a comparison between the case four and five in Fig. 10 shows the slightly shorter wall time of the case five for inference. These results, together with the results from the case one to five, suggest that if focus is placed on the wall time, the object detection system performs best when it deals with both input and output data in sub-threads. Fig. 10 also shows the case five brings the shortest wall time in all cases for inference.

Table 10. CPU time in the case five

Image input			Camera
CPU time (s)	Inference	Min.	3.524×10^{-1}
		Ave.	3.569×10^{-1}
		Max.	3.608×10^{-1}
	PL	Min.	1.543×10^{-1}
		Ave.	1.569×10^{-1}
		Max.	1.638×10^{-1}

Table 11. Wall time in the case five

Image input			Camera
Wall time (s)	Inference	Min.	1.176×10^{-1}
		Ave.	1.184×10^{-1}
		Max.	1.192×10^{-1}
	PL	Min.	5.233×10^{-2}
		Ave.	5.235×10^{-2}
		Max.	5.239×10^{-2}

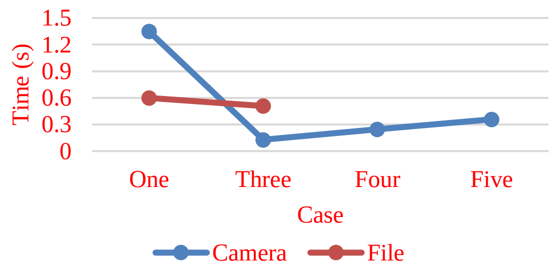

Fig. 8. Average inference CPU time

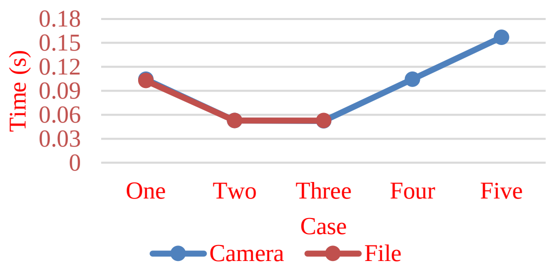

Fig. 9. Average PL CPU time

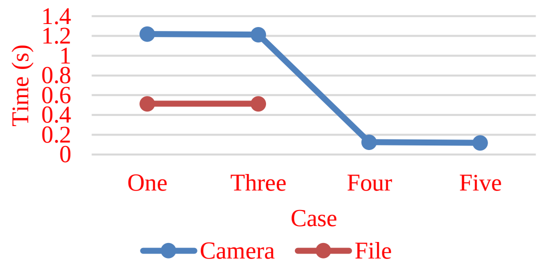

Fig. 10 Average inference wall time

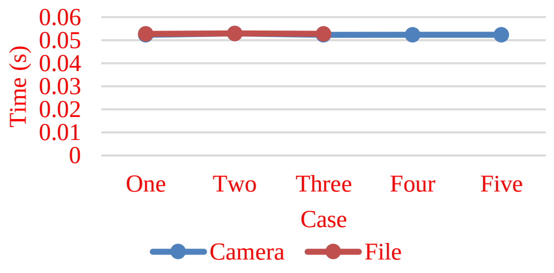

Fig.11 Average PL wall timre

Table 12. CPU time in the case six

Image input			Camera				
The number of threads			8	9	10	100	1,000
CPU time (s)	Inference	Min.	6.072 x 10^{-1}	5.910 x 10^{-1}	7.074 x 10^{-1}	6.493	6.490 x 10^{1}
		Ave.	7.171 x 10^{-1}	7.992 x 10^{-1}	8.479 x 10^{-1}	6.975	6.828 x 10^{1}
		Max.	7.774 x 10^{-1}	9.479 x 10^{-1}	9.779 x 10^{-1}	7.469	7.189 x 10^{1}
	PL	Min.	1.762 x 10^{-1}	1.842 x 10^{-1}	1.838 x 10^{-1}	4.057 x 10^{-1}	3.992
		Ave.	1.913 x 10^{-1}	1.962 x 10^{-1}	2.107 x 10^{-1}	4.201 x 10^{-1}	4.026
		Max.	2.220 x 10^{-1}	2.171 x 10^{-1}	2.381 x 10^{-1}	4.417 x 10^{-1}	4.072

Table 13. Wall time in the case six

Image input			Camera				
The number of threads			8	9	10	100	1,000
Wall time (s)	Inference	Min.	1.734 x 10^{-1}	1.841 x 10^{-1}	1.957 x 10^{-1}	1.642	1.625 x 10^{1}
		Ave.	2.040 x 10^{-1}	2.233 x 10^{-1}	2.315 x 10^{-1}	1.763	1.710 x 10^{1}
		Max.	2.193 x 10^{-1}	2.659 x 10^{-1}	2.618 x 10^{-1}	1.886	1.800 x 10^{1}
	PL	Min.	5.231 x 10^{-2}	5.231 x 10^{-2}	5.232 x 10^{-2}	1.017 x 10^{-1}	9.982 x 10^{-1}
		Ave.	5.428 x 10^{-2}	5.470 x 10^{-2}	5.818 x 10^{-2}	1.050 x 10^{-1}	1.006
		Max.	6.335 x10^{-2}	6.186 x10^{-2}	6.648 x10^{-2}	1.098 x10^{-1}	1.018

Table 14 presents the relationship between the number of threads in the PS and the execution time in the PL. The time in Table 14 represents the meantime in the PL for the USB camera input. Evidently, the longer the PL CPU time, the longer the PL wall time. Moreover, if the PL CPU time became over approximately 0.19 seconds, the PL wall time also started increasing. If the PL CPU time was below approximately 0.19 seconds, the PL wall time was kept at almost the constant execution time.

Table 14. Relationship between the number of threads and the execution time in PL

Case	The number of threads	Mean CPU time in PL (s)	Mean wall time in PL (s)
Three	1	5.227 x 10^{-2}	5.232 x 10^{-2}
Four	2	1.043 x 10^{-1}	5.237 x 10^{-2}
Five	3	1.569 x 10^{-1}	5.235 x 10^{-2}
Six	8	1.913 x 10^{-1}	5.428 x 10^{-2}
	9	1.962 x 10^{-1}	5.470 x 10^{-2}
	10	2.107 x 10^{-1}	5.818 x 10^{-2}
	100	4.201 x 10^{-1}	1.050 x 10^{-1}
	1,000	4.026	1.006

5. Discussion

The PS load does not affect the PL execution, particularly in terms of the wall time, unless the PS load exceeds the specific CPU time in the PL. Although switching methods to obtain data from the USB camera to the file and vice versa does not affect the CPU time in the PL, the number of threads in the PS directly impacts the CPU time in the PL. Even if the CPU time in the PL increases as the number of threads in the PS increases, the wall time in the PL does not change until the CPU time in the PL reaches a specific value. After the CPU time in the PL exceeds the specific time, the wall time in the PL also increases as the number of threads increases.

For the proposed system, when the total number of threads in an application approached approximately eight, the CPU time in PL became approximately 0.19 seconds and the wall time in PL also began increasing. Further, the proposed system performed best when input data were acquired in a sub-thread, and output images were displayed in a sub-thread.

6. Conclusion

The system performance of the proposed system can be improved by considering both the PS and PL independently because the PS load, except for factors to increase the CPU time in the PL such as the number of threads, does not affect the wall time in the PL. When dealing with multi-threading, because the PS load is likely to degrade the PL performance, we need to be careful particularly of the PS load to prevent the system performance deterioration.

In the near future, we will integrate our neural networks and artificial intelligence circuits into a PL [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29] and then connect it to a home service robot via a PS by utilizing the knowledge of this study [30], [31], [32], [33] to realize a high-performance and low-power intelligent robot system.

References

1. C. Profentzas, M. Almgren and O. Landsiedel, "Performance of deep neural networks on low-power IoT devices", Proceedings of the Workshop on Benchmarking Cyber-Physical Systems and Internet of Things, pp. 32-37, 2021.
2. L. Caballero, Á. Perafan, M. Rinaldy and W. Percybrooks, "Predicting the Energy Consumption of a Robot in an Exploration Task Using Optimized Neural Networks", Electronics, vol. 10, no. 8, pp. 920, 2021.
3. M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno and P. H. Jones, "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels", 2019 IEEE International Conference on Embedded Software and Systems (ICESS), pp. 1-8, 2019.
4. H. Nakahara and T. Sasao, "A High-speed Low-power Deep Neural Network on an FPGA based on the Nested RNS: Applied to an Object Detector ", 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2018.
5. L. Mo, C. Wu, L. He and G. Chen, "Layout driven FPGA packing algorithm for performance optimization", IEICE Electronics Express, vol. 14, no. 11, pp. 1-10, 2017.
6. T. Nguyen, C. MacLean, M. Siracusa, D. Doerfler, N. J. Wright and S. Williams, "FPGA-based HPC accelerators: An evaluation on performance and energy efficiency", Concurrency and Computation: Practice and Experience, vol. 34, no. 20, 2021.
7. S. Ramagond, S. Yellampalli and C. Kanagasabapathi, "A review and analysis of communication logic between PL and PS in ZYNQ AP SoC", 2017 International Conference on Smart Technologies for Smart Nation (SmartTechCon), pp. 946-951, 2017.
8. A. Rios-Navarro, R. Tapiador-Morales, A. Jimenez-Fernandez, C. Amaya, C. Amaya, T. Delbruck and A. Linares-Barranco, "Performance evaluation over HW/SW co-design SoC memory transfers for a CNN accelerator", 2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO), pp. 1-4, 2018.
9. C. Concatto, A. Attwood, J. A. Pascual, M. Ashworth, J. Navaridas, M. Luján and J. Goodacre, "Enabling shared memory communication in networks of MPSoCs ", Concurrency and Computation: Practice and Experience, vol. 31, no. 21, 2018.
10. C. Wang and Z. Luo, "A Review of the Optimal Design of Neural Networks Based on FPGA", Applied Sciences, vol. 12, no. 21, 2022.
11. S. S. Lingala, S. Bedekar, P. Tyagi, P. Saha and P. Shahane, "FPGA Based Implementation of Neural Network", 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), pp. 1-5, 2022.
12. J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang and H. Yang, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network", Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '16), pp. 26-35, 2016.
13. Y. Watanabe and H. Tamukoh, "Impact of PS load on FPGA object detection system performance", Proceedings of International Conference on Artificial Life and Robotics, pp. 415-419, 2023.
14. M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, " Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1", 2016.

15. H. Nakahara, H. Yonekawa, H. Iwamoto and M. Motomura, " A Batch Normalization Free Binarized Convolutional Deep Neural Network on an FPGA (Abstract Only)", Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17), pp. 290, 2017.
16. J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger", 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517-6525, 2017.
17. Y. Watanabe and H. Tamukoh, " Influence of FPGA Implementation Methods in High-Level Synthesis", 2021 International Conference on Artificial Life and Robotics (ICAROB 2021), vol. 26, pp. 10-11, 2021.
18. J. Licht, M. Besta, S. Meierhans and T. Hoefler. " Transformations of High-Level Synthesis Codes for High-Performance Computing ", IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 5, pp. 1014-1029, 2021.
19. S. Hori and H. Tamukoh, "An Implementation Method Using Cut-Off Bits for Restricted Boltzmann Machines Without Random Number Generators", IEEE Access, vol. 10, pp. 42791-42801, 2022.
20. Y. Tanaka and H. Tamukoh, " Reservoir-based convolution ", Nonlinear Theory and Its Applications, IEICE, vol. 13, no. 2, pp. 397-402, 2022.
21. N. Fuengfusin and H. Tamukoh, " INT8 Activation Ternary or Binary Weights Networks", Proceedings of International Conference on Artificial Life and Robotics, vol. 27, pp. 599-603, 2022.
22. N. Fuengfusin and H. Tamukoh, " Mixed-precision weights network for field-programmable gate array ", PLoS ONE, vol. 16, no. 5, 2021.
23. N. Fuengfusin and H. Tamukoh, " A Sub-Model Detachable Convolutional Neural Network", Journal of Robotics, Networking and Artificial Life, vol. 8, no. 1, pp. 52-55, 2021.
24. N. Fuengfusin and H. Tamukoh, "Network with Sub-networks: Layer-wise Detachable Neural Network " , Journal of Robotics, Networking and Artificial Life, vol. 7, no. 4, pp. 240-244, 2021.
25. Y. J. Yeoh, T. Morie and H. Tamukoh, "An Efficient Hardware-Oriented Dropout Algorithm " , Neurocomputing, vol. 427, pp. 191-200, 2021.
26. Y. Tanaka, T. Morie and H. Tamukoh, "An amygdala-inspired classical conditioning model on FPGA for home service robots", IEEE Access, vol. 8, pp. 212066-212078, 2020.
27. I. Kawashima, T. Morie and H. Tamukoh, " FPGA implementation of hardware-oriented chaotic Boltzmann machines", IEEE Access, vol. 8, pp. 204360-204377, 2020.
28. K. Honda and H. Tamukoh, "A Hardware-Oriented Echo State Network and its FPGA Implementation", Journal of Robotics, Networking and Artificial Life, vol. 7, no. 1, pp. 58-62, 2020.
29. A. Suzuki, T. Morie and Hakaru Tamukoh, "A shared synapse architecture for efficient FPGA implementation of autoencoders", PLoS ONE, vol. 13, no. 3, 2018.
30. T. Ono, D. Kanaoka, T. Shiba, S. Tokuno, Y. Yano, A. Mizutani, I. Matsumoto, H. Amano and H. Tamukoh, "Solution of World Robot Challenge 2020 Partner Robot Challenge (Real Space)", Advanced Robotics, vol. 36, no. 17-18, pp. 870-889, 2022.
31. Y. Yoshimoto and H. Tamukoh, "FPGA Implementation of a Binarized Dual Stream Convolutional Neural Network for Service Robots", Journal of Robotics and Mechatronics, vol. 33, no. 2, pp. 386-399, 2021.
32. Y. Ishida, T. Morie and H. Tamukoh, " A hardware intelligent processing accelerator for domestic service robots", Advanced Robotics, vol. 34, no. 14, pp. 947-957, 2020.
33. Y. Ishida and H. Tamukoh, " Semi-Automatic Dataset Generation for Object Detection and Recognition and its Evaluation on Domestic Service Robots ", Journal of Robotics and Mechatronics, vol. 32, no. 1, pp. 245-253, 2020.

Authors Introduction

Mr. Yusuke Watanabe

He received his B.Eng. and M.Eng. degrees from Waseda University, Japan, in 2007 and 2009. He is pursuing his Ph.D. degree in Kyushu Institute of Technology. He is working for CRAFT WORK Co., Ltd. His research interest includes digital hardware design, soft-computing, and home service robots.

Mr. Hakaru Tamukoh

He received his B.Eng. degree from Miyazaki University, Japan, in 2001. He received his M.Eng. and Ph.D. degrees from Kyushu Institute of Technology, Japan, in 2003 and 2006, respectively. He was a postdoctoral research fellow at Kyushu Institute of Technology, from 2006 to 2007. He was an assistant professor at Tokyo University of Agriculture and Technology, from 2007 to 2013. He is currently a professor in the graduate school of Life Science and Systems Engineering, Kyushu Institute of Technology, Japan. His research interest includes digital hardware design, soft-computing, and home service robots. He was the author of works that won the Best Paper Award at IJCNN 2019, the Best Live Demonstration Award at ISCAS 2019, the Best Paper Award at ICONIP 2013. He is a member of IEEE, IEICE, JNNS.