PROGRESS IN APPLIED SCIENCE AND TECHNOLOGY
Faculty of Science and Technology
Rajamangala University of Technology Thanyaburi
Prog Appl Sci Tech. 2024; 14(2):44-54
ISSN (Print): 2730-3012
ISSN (Online): 2730-3020
https://ph02.tci-thaijo.org/index.php/past

Research Article

Received: March 01, 2024
Revised: June 12, 2024
Accepted: Augst 26, 2024

DOI: 10.60101/past.2024.252733

Video Analytic for Human Management and Security and FPGA Accelerated High Concurrency

Boonchom Sudjit[1*], Somrak Petchartee[2] and Maneesha Perera[3]
[1] *Faculty of Engineering and Architecture, Rajamangala University of Technology Tawan-ok, Bangkok 10330, Thailand.*
[2] *Digital Innovation Center Brownien Laboratory, NT Telecom Public Company Limited 10210, Thailand*
[3] *Research Assistance, Brownien Laboratory, NT Telecom Public Company Limited 10210, Thailand*
*E-mail: boonchom_su@rmutto.ac.th

Abstract

This paper explores the use of video analytics by leveraging accelerated FPGA technology in combination with high-performance computing, specifically utilizing two Xilinx Alveo U50Lv cards and one U55C card. While many applications exist for motion analysis and detection in videos, the use of FPGAs in this context remains relatively scarce. FPGAs offer significant advantages in terms of energy efficiency and throughput. We present results demonstrating the parallelism capabilities in terms of the number of threads within a single Docker container that shares stack memory, as well as across multiple Docker containers. When operating within a single Docker process, the application shares the same memory space and resources, making it ideal for tasks that require efficient communication or data sharing. In contrast, running in multiple containers isolates processes, each with its own environment, and can significantly increase the number of threads. Our findings show that the combination of these techniques offers optimal performance for video analytics.

Keywords: Video Analytic, FPGA Accelerated High Concurrency, Deep Neural Network

1. Introduction

Video Analytics is a concept that has been gaining interest in both the academic and industrial sectors. Continuous improvements in video analytic have being used as key features in integrated systems. The main goal of video analytic is to automatically detect temporal events in a video. Usually, video analytic is perform on real-time videos where the main focus is on the movement of objects in a specific area.

Video analytic is used in many industries such as healthcare, transportation, retail, and security. This study mainly focuses on video analytic for Human Management and Security. Video Analytic for Human Management and Security is a system for evaluating and analyzing images captured by IP surveillance cameras. According to the defined functions, the system will send an alert as soon as an occurrence is identified. This will be a mechanism that will assist in the facilitation process. and improve surveillance efficiency Investigate numerous occurrences, which will not only improve surveillance efficiency but will also reduce the cost of managing the security system in the long term. The defined functions for this study will be introduced under the features of AI of this paper.

Deep Neural Network (DNN) which is a subsection of Artificial Intelligence (AI) is used to train video analyzing systems and make it possible for these systems to identify objects and track their movements. Due to the

complexity of computations and logic in neural networks, a Field-programmable gate array (FPGA) chip-based system was used in this study. The implementation and the speed-up process of FPGAs will be discussed in detail in the latter part of this paper.

FPGAs are comprised of programmable logic elements systematically arranged in a two-dimensional grid architecture. These configurable regions encompass look-up tables (LUTs), which are capable of realizing specified truth tables. Recently, FPGAs have been augmented with various hard intellectual property (IP) modules, such as Intelligent Processing Units (IPUs),

Deep Processing Units (DPUs), DDR/HBM controllers, and additional platform-specific IPs interspersed among the programmable logic domains.

2. Application: Internal use in Company with 100 Cameras

Proposed system was implemented at NT Telecom Head Office that includes 3 main buildings namely, Auditorium building, Office building, and Vehicle Park Building with 100 CCTV cameras. Since the number of CCTV cameras are relatively high, using and managing them to their maximum potential is challenging. There-fore, it is necessary to install a high-performance CCTV system. Here, we have implemented a CCTV system of Pelco brand.

Control room of the CCTV cameras are located at the Car Park building. We directed the live video stream from Car Park building to the Brownien Lab Research and Development Center via fiber optics (Figure 1).

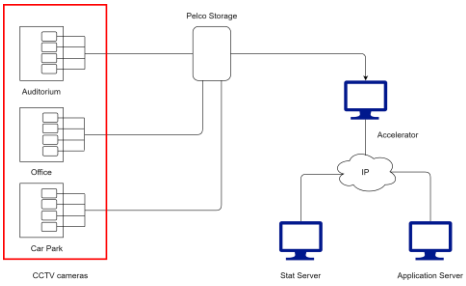

Figure 1 Network Configuration

We implemented the system in Python and integrated it with a Xilinx Deep Processing Unit (DPU) on a server equipped with two Xilinx Alveo U50Lv cards and one U55C card. This setup is capable of achieving a maximum design frequency of 1,000 images per second. The server is powered by a 128-core AMD EPYC 7V13 CPU operating at 2.45GHz.

3. Features of AI

3.1 Tripwire

Tripwire is nothing but a virtual line which is used in areas that are needed to be guarded and alert the user of any trespassers. In order to detect trespassers accurately using AI, first we trained the body's data and created the virtual line that prevents trespassing. Whenever a person crosses this virtual line, there will be an alert.

3.2 Enter and Exit

Enter and Exit is much similar to Tripwire technique. According to the direction the body moves, we can detect entering exiting. This feature also uses a virtual line to detect and alerts when a person enters or exits a forbidden area (i.e. Fire exit door) as in Figures 2-3.

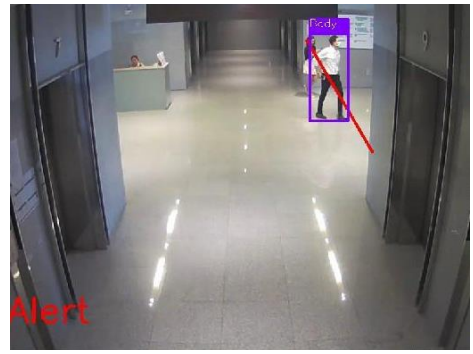

Figure 2 Alert when a person is crossing the virtual line

Figure 3 A person entering a restricted area

3.3 Body Detection

AI virtually detects a person walking in the place of interest and sends the body value to the API system. API system interprets the detection that was captured by each camera. This result conveys the number of people in a particular area as in Figure 4.

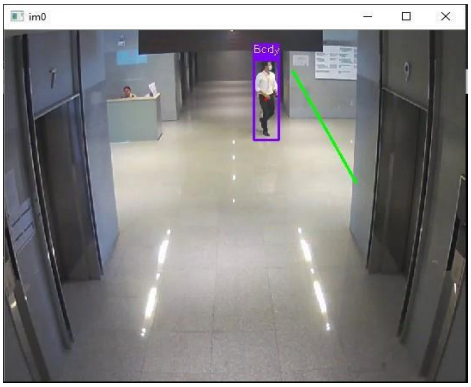

Figure 4 Count of human bodies in a monitored area

3.4 Head Detection

The head detection feature counts the number of heads that pass a certain area. Counting the number of heads will reveal the number of people in that particular area (i.e., an elevator). Obtaining the headcount is done by cropping a person's head virtually as in Figure 5. However, there are few possibilities that AI confuses anything black to be a head because head cropping starts with black hair. To resolve this issue, the crop-ping will be done from head to the neck.

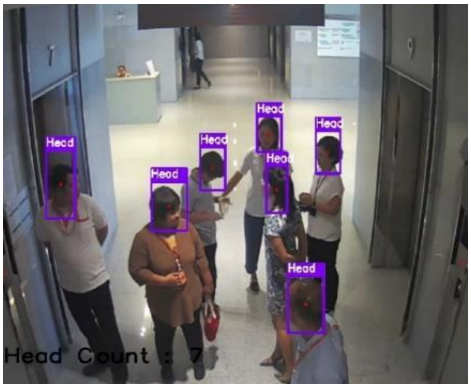

Figure 5 Count of human bodies in a monitored area

3.5 Man down Detection

This feature helps to detect people who need assistant in case of emergencies. From this study, we realized in order to detect man down situations, it is required to crop the image of the person's body, buttocks, or both knees, landing on the ground as in Figure 6. Whenever a man down situation occurs, the security personnel will be notified and will assist the person according to the emergency (i.e., contact emergency vehicle, provide CPR).

Figure 6 Detection of a person who has fallen

3.6 Loitering

Loitering is a feature that is used to detect suspicious activities. Loitering hap-pens when a person stays within the area of interest for a longer time. This will trigger the alarm and security personals will take actions with using the surveillance video as in Figure 7. In this study we identified 12 loitering patterns. However, due to the different intensities of the sunlight, the detection was not up to the standards. To resolve this problem, the training was done in diverse lighting conditions.

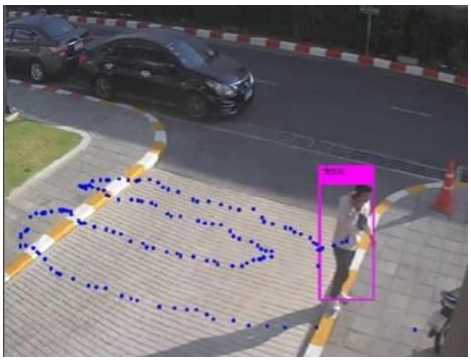

Figure 7 Detection of a suspicious movement of a person

The movement of the person is measured from the centroid point to each point along X and Y axis. Different algorithms were used to calculate the movement in degrees according to the walking pattern as shown in Figure 8.

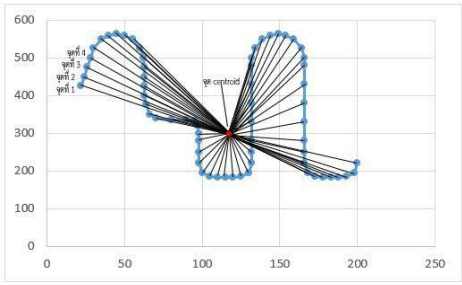

Figure 8 The x y graph shows the use of centroid points to measure degrees.

3.7 Fire and Smoke Detection

Fire and smoke detection is one of the commonly used features in video analytic. In case of a fire, captured footage will be processed using AI and AI will detect the smoke and flame which triggers the fire alarm. To be able to correctly detect the smoke and flame, it is required to train the model with an image file and a text file that does not contain any label data (Figure 9).

Figure 9 Detection of fire

3.8 Unattended Object Detection

Unattended objects can be an act of terrorism or an act of forgetfulness. This feature will detect an object that has been left in a certain area for a long time unattended and the body data of the person who left the object as in Figures 10-11. Security personal will be notified and actions will be taken according to the situation.

Figure 10 A person entering and leaving an object

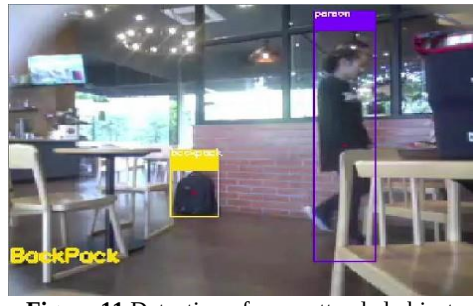

Figure 11 Detection of an unattended object

4. Architecture

Concurrency occurs when multiple tasks are progressing at the same time. Generally, concurrency is achieved by interconnecting two or more processors. High concurrency is aimed at enhancing the performance of tasks (i.e., video analyzing and detection) to obtain fast and accurate results. This is known as High-Performance Computing (HPC) (1). HPC has many benefits compared to the traditional computing. HPC often integrated with AI and as a result of this integration, capabilities of researches are widening exponentially. High concurrent AI computing includes training and testing AI algorithms and optimizing their performances using different architectures. Training AI models is a multi-staged and iterative task. Based on its local data collection, each iterative step calculates parameters. This iterative parameter update process continues until the target model accuracy is reached (2). The main benefit of concurrency is that many tasks cab be progressed at the same time which leads to fast obtaining of the updated parameters. Computed nodes communicate with each other and synchronizes to complete a given task. Therefore, communication and synchronizations are key factors that should be included when using concurrency. Novel contributions of HPC and AI

integration have generated many promising results. These studies prove the efficiency of the integration. Daeyong Jung et al. (3) developed a management approach for data synchronization and heterogeneous cluster computing with multiple compute nodes. Various Windows Sub-system for Linux were used in the memology (WSL) package. For Son of Grid Engine, the cluster scheduler engine was employed (SGE). The manager synchronizes with the Linux daemon under specified conditions and the Manager supervises the jobs that synchronize the data in SGE. Sangdo Lee et al. (4) investigated previous cyber-attack incidents and hacking efforts that targeted nuclear power plants. They recommended a security measure based on the study to protect against previous attack incidences and the mechanism of large data exploding in the attacks. Yin et al. (5) addressed the issue of increased operating speed and regular engines in finite-state automata-based deep packet inspection systems in order to reduce memory usage. The authors looked at the limitations of using finite automata as well as its key feature. They enhanced the non-deterministic finite automata (NFA) based on the analysis to reduce memory use by lowering the conversion edge. Hwang et al. (6) presented the importance of AI for the Internet of things (IoT) in preserving huge data in a variety of domains, including smart transportation, health- care, large-scale deployments, infrastructure management, and smart homes manufacturing, home automation and etc. They proposed a network clock paradigm for time awareness in IoT and cloud integration AI. The suggested system was put to the test using a MICAz-compatible test platform. Sensor nodes, and the results of the experimental assessment revealed that the any AI application can use IoT devices to supply and maintain a standard timestamp. Above mentioned studies successfully performed the relevant task and obtained efficient results which proves the consistency of HPC in AI. However, there are several drawbacks in HPC and AI clusters due to the network. Network plays a major role in distributed compute and storage architecture. Due to their iterative, intensive, frequent, and synchronous communication, distributed AI and HPC systems will aggravate network infrastructure necessity. The performance of distributed systems is heavily influenced by network infrastructure bandwidth, latency, congestion management, dependability, and a variety of other factors (7). Some of these factors will be discussed briefly.

Network latency. Network latency is the time that it takes data or requests to transmit from the source to the destination. There are two types: Static Latency and Dynamic Latency. Static latency is associated with switch forwarding latency and optical /electrical transmission latency while Dynamic latency is associated with queuing delay, flow completion time, tail latency, packet drop, burst and microbursts. Incast and Bursts. The frequent incast traffic patterns created by AI and HPC distributed systems place strain on network infrastructure, producing congestion and substantial latency spikes. As for the bursts, they occur due to the communication between AI layers in order to obtain the task completion. These bursts fill the free space in net-work node queues and increases the queuing time which leads to the communication latency. Flow Completion Time (FCT) and Tail Latency. Since bursts increases the queue time, the time that task to be completed will also increase. Some tasks depend on the results of previous tasks and since the completion of the previous tasks are delayed, tail latency will be created. Tail latency is the latency of the fraction of the flows that take the longest to complete. In a nutshell, HPC and AI are systems that require a lot of compute and storage. While concurrency allows these algorithms to reach their full potential, it also puts a lot of strain on network operations. In order to meet the needs of HPC and AI clusters, network infrastructure must expand beyond traditional architecture.

4. Methodology

The extensive utilization of various artificial intelligence (AI) applications, particularly deep neural networks (DNNs), underscores the necessity for specialized hardware solutions. Conventional computer systems face significant challenges in terms of data processing speed and scalability. Domain-specific designs provide superior energy efficiency and performance improvements compared to general-purpose processors, with the general aim of improving throughput and energy efficiency.

Consequently, leading players in the data center market are progressively incorporating FPGA System-on-Chip (SoC) into their infrastructure to address sophisticated AI applications. Notable companies such as

Microsoft, Amazon Web Services, and Baidu are expanding their AI and high-performance application capabilities across hundreds of thousands of Intel and Xilinx FPGA devices (8, 9).

5. Result: FPGA Speeding up Processing

As implementations of AI grows larger, the complexity of computations grows too. As a result, the necessity to speed up microprocessors has become prominent. Compared to the traditional GPU, FPGA has an advantage in speeding up the process in computing. FPGAs have achieved large speedups for a wide variety of application. This section aims to discuss the contribution of FPGAs. This study uses a HPC system which allows multiple microprocessors and multiple FPGAs to be embedded to the system to enhance the overall performance. FPGA accommodates a variety of parallel computer applications and can be implemented in a single clock cycle (10). Moreover, re- programmable FPGAs provide on-chip capability for a variety of applications. The memory access bandwidth of co-processor logic is not limited by the number of I/O pins available in the devices due to the presence of on-chip memory (11 – 19).

Tobias (20) introduced a methodology for partitioning Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) across multiple FPGAs; however, this approach lacks generalizability across diverse workloads. Alternative methodologies, such as those discussed in (21), exploit latency insensitivity to facilitate the partitioning of designs across multiple FPGAs. These approaches, however, necessitate that the user manually define module-to-FPGA mappings at the Register-Transfer Level (RTL) and conduct simulation-based evaluations. The integration of High-Level Synthesis (HLS) frontends with automatically partitioned designs could significantly broaden FPGA adoption among users with diverse technical backgrounds.

An example of utilizing the Yolo model within the Xilinx toolkit can be observed through the segmentation of the model's operations across different hardware components. The first segment, responsible for accessing images, operates on the processor. The second segment, which handles image processing, runs on the Program Logic (PL). The third segment operates in the Deep Learning Processor Unit (DPU), which is also developed on the PL, as illustrated in Figure 12.

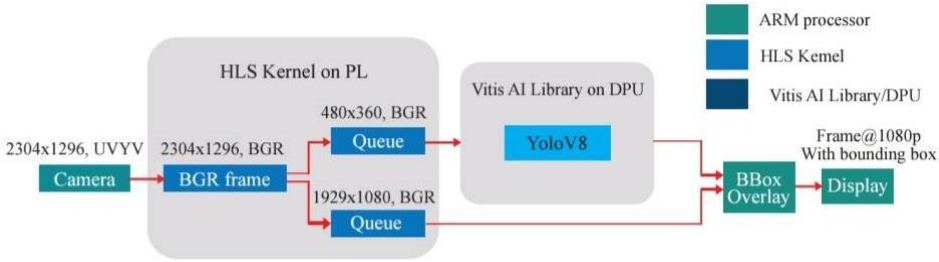

Figure 12 Yolov8 Implementation on FPGA

5.1 FPGA vs. GPU

When implementing FPGA and GPU in Binary Convolutional Neural Network separately, we can witness FPGA outperforming GPU in every aspect. In terms of Through-put/FPS (Figure 13) shows that FPGAs offer more promising results than GPU across the board. Furthermore, in terms of Energy Efficiency/W, FPGAs achieve more compelling results compared to GPU.

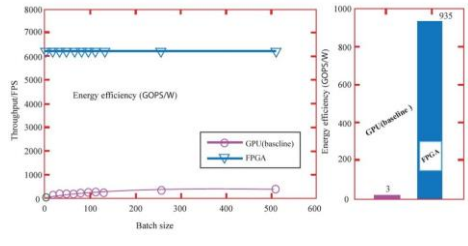

Figure 13 FPGA vs. GPU
Throughput and energy efficiency comparison with GPU and FPGA implementations (Binary Convolution) (22)

5.2 Speed-up Process

Customized, application specific accelerator cores are common in FPGA designs. Compared to the regular cores, Accelerator cores are largely synchronous circuits with a strict clock period timing limitation. They are, on the other hand, nearly often built by a system designer for a specific application and are utilized to set the system apart from other systems. A designer should be able to swiftly and simply develop high-performance custom cores, conduct a design space exploration of viable designs, and integrate them into their systems in a short amount of time.The foundational implementation facilitates the exchange of messages between the global memory of interconnected FPGAs arranged in a ring topology via PCIe. Additionally, it utilizes the Xilinx SDK to read from and write to a memory buffer that represents the message, thereby enabling communication between the FPGA and the host system.

We implemented the system in Python and integrated it with a Xilinx Deep Processing Unit (DPU) on a server equipped with two Xilinx Alveo U50Lv cards and one U55C card. This setup is capable of achieving a maximum design frequency of 1,000 images per second. The server is powered by a 128-core AMD EPYC 7V13 CPU operating at 2.45GHz.

6. Speed Verify and Improve

Indeed, executing a quantized model on an FPGA (Field-Programmable Gate Array) can lead to significant performance improvements, especially for applications requiring low latency and high throughput. FPGAs are particularly well-suited for running quantized models due to their reconfigurable nature and efficient handling of parallel tasks. FPGAs can be configured to perform specific operations in an optimized manner. For quantized models, this means that the computations can be executed very efficiently, often outperforming general-purpose CPUs and always even GPUs. Quantized models use lower precision arithmetic (like INT8), which FPGAs can handle more efficiently than floating-point operations. This leads to faster computations and reduced power consumption as in Figure 14.

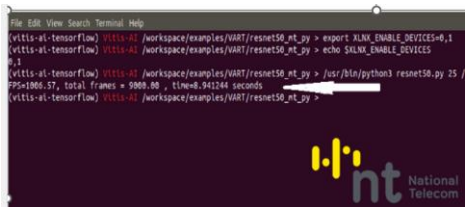

Figure 14 Inference Rate Test

Table 1 Under 250, 500 and 1,000 Test Image VS Number of Thread, With 1,000 Class Model

Thread	Count	Second	FPS
1	1000	7.7	129.9
2	500	7.9	126.6
3	250	5.6	131.6
3	1000	8.3	120.5

Table 1 presents the results of AI inference speed tests, highlighting variations in performance across different configurations. The tests were conducted with varying numbers of threads and test images, measuring the time taken (in seconds) and the resulting frames per second (FPS) for inference. Thread 1: With 1,000 test images, the inference time was 7.7 seconds, resulting in an FPS of 129.9. Thread 2: For 500 test images, the inference time was 7.9 seconds, achieving an FPS of 126.6. Thread 3: When processing 250 test images, the inference time increased to 5.6 seconds, with an FPS of 131.6. Additionally, when the number of test images was raised to 1,000, the inference time was 8.3 seconds, resulting in an FPS of 120.5. These results indicate how the number of threads and test images impact the inference speed and efficiency of the AI system. Notably, increasing the number of test images does not linearly affect the inference time and FPS, suggesting complex interactions between the system's workload and its performance capabilities.

FPGAs excel in parallel processing, allowing them to execute multiple operations of a quantized model simultaneously. This is particularly advantageous for the parallel nature of neural network computations. In Figure 15, A speed testing a model with 1,000 classes on 1,000 images using one terminal. When using a CPU to manage the execution of tasks on an FPGA and to retrieve results, the orchestration typically involves a certain number of software threads. These threads are responsible for handling the communication between the CPU and FPGA, including sending data to the FPGA

for processing, initiating the execution of the model on the FPGA, and receiving the processed results back. Managing these threads effectively is crucial for optimal performance, particularly for inference tasks in deep learning. The number of threads can impact the efficiency of the system. Too few threads might underutilize the FPGA's capabilities, while too many threads can lead to overhead and contention, potentially slowing down the system.

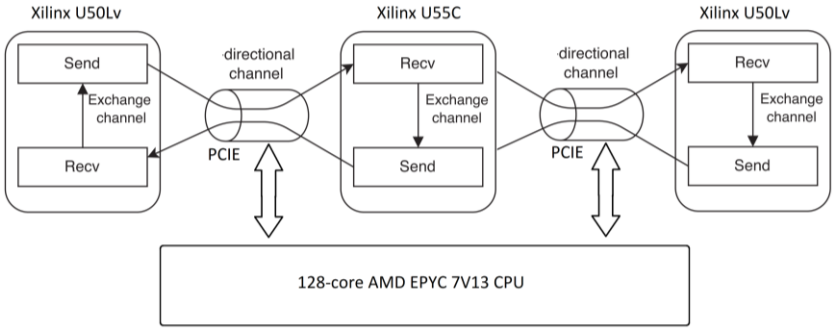

Figure 15 Simplified diagram of the system architecture

Table 2 Under 1,000 Test Image VS Number of docker container, With 1,000 Class Model

Terminal	Terminal	FPS/terminal	FPS
1	7.6	131.5789474	131.6
2	8.1	123.4567901	246.9
3	8.3	120.4819277	361.4

Table 2, a speed test on a model with 1,000 classes, using 1, 2, and 3 terminals to test 250 images per terminal. The number of threads can impact the efficiency of the system. Too few threads might underutilize the FPGA's capabilities, while too many threads can lead to overhead and contention, potentially slowing down the system.

7. AI Model Theory and Contribution

Figure 16, there was an issue or error encountered during the process of quantization that affected the computational graph of the model. Quantization is the process of reducing the precision of the weights and activations of a neural network, typically from floating-point to lower-bandwidth integers, for example, from 32-bit floating-point to 8-bit integers. This process is crucial for deploying models on resource-constrained devices like mobile phones or embedded systems, as it reduces the model size and increases inference speed while aiming to maintain accuracy.

Figure 16 Broke Graph during Quantization

Graphs in the context of deep learning models refer to a structured representation of the model itself, or to the use of graph-based data structures in deep learning algorithms as in Figure 17.

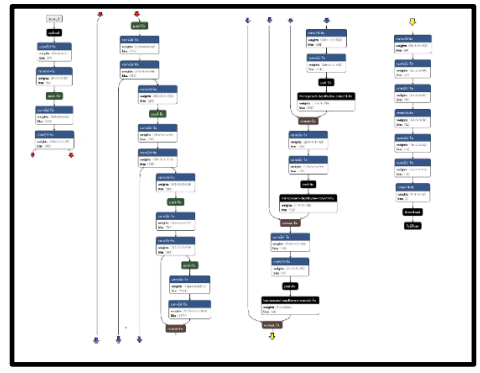

Figure 17 The inherently possesses a network structure

We encountered a situation where the computational graph of a deep learning model broke during quantization, possibly due to the absence of a library or specific functionality needed to handle the quantization process correctly. This issue can occur when the quantization library or the method used does not fully support all the operations or layers in your model. Carefully review the model to identify which specific operations or layers are not supported by the quantization library.

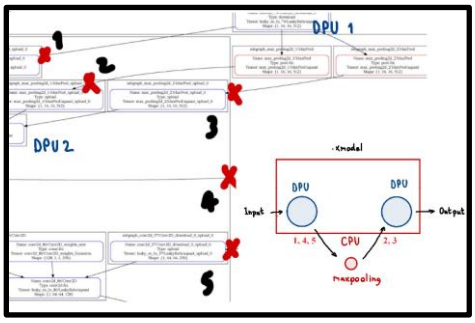

Figure 18 FPGA and CPU execution with Data Transfer and Synchronization.

If a small part of the model is causing issues, consider partially quantizing the model. Quantize only the layers that are supported as in Figure 18, leaving the problematic layers in their original precision. This approach can still offer some benefits in terms of model size and performance, albeit less than full quantization. Using a fallback to partial quantization and processing unsupported operations on the CPU is a practical approach when dealing with a model where not all components can be quantized. This strategy allows you to leverage the benefits of quantization for the parts of the model that are compatible while still maintaining the overall functionality by executing the unsupported operations on the CPU. Profile the model to identify bottlenecks. In some cases, the CPU-bound operations might become the bottleneck, and further optimization might be necessary, such as parallelizing certain computations or optimizing the CPU-bound code.

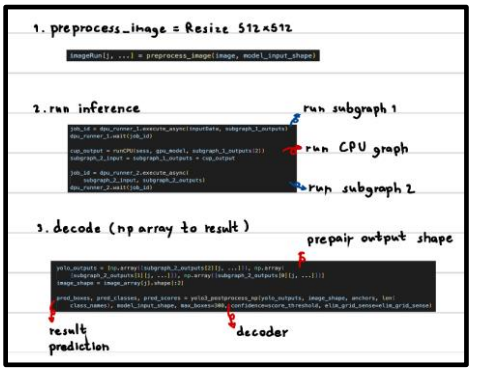

Figure 19 FPGA and CPU execution with Data Transfer and Synchronization.

In Figure 19, implement efficient data transfer mechanisms between the CPU and FPGA. This step is crucial to minimize the performance hit due to data transfer between different processing units. Ensure proper synchronization between the operations running on different hardware. The output from one part of the model should seamlessly feed into the next, maintaining the integrity and flow of the computation. The effective-ness of this approach can depend on the specific hardware capabilities and the nature of the model. Some devices might have powerful CPUs that can handle the non-quantized operations without significantly impacting overall performance. Create a hybrid execution plan where the quantized and non-quantized parts of the model are handled differently. The quantized parts can run on specialized hardware (like a FPGA or TPU), optimized for low-precision computations, while the non-quantized parts run on the CPU. Running multiple Docker containers to parallelize the execution of the same quantized model on an FPGA can indeed increase throughput and efficiency, especially under a Linux environment. This approach leverages containerization to isolate and manage multiple instances of the model running concurrently. Here are some key points to consider when implementing this strategy: Each Docker container acts as an isolated environment, running its own instance of the application (in this case, the inference model). This isolation helps in managing dependencies and ensuring that each instance runs in a consistent environment. By running multiple containers, you can effectively parallelize the workload. Each container can handle separate inference

requests, thereby increasing the overall throughput of the system. The degree of parallelization will depend on the FPGA's capabilities and how well it handles concurrent execution. It's important to manage the resources (like memory and compute time) allocated to each container to ensure efficient operation. The FPGA must be capable of handling the combined load from all containers. Overloading the FPGA could lead to reduced performance or system instability. Some FPGAs support virtualization, allowing them to be shared among multiple applications or containers. This can be a complex process but is beneficial for maximizing resource utilization. Ensure that your FPGA and its driver support such virtualization. In summary, using Docker containers to parallelize the execution of a quantized model on an FPGA can significantly enhance throughput and efficiency. However, it requires careful planning in terms of resource allocation, load balancing, container management, and ensuring that the FPGA can handle the concur-rent execution load. This approach is particularly useful in scenarios where high scalability and isolation between different inference tasks are required. Finally, the results from improving the AI model in the experiment inspired researchers to realize the effort and capability required to completely convert a model trained on a 32-bit GPU to work on an 8-bit FPGA. In cases where certain layers of the model cannot be converted, it will be necessary for those layers to be processed on the CPU instead. This presents a significant challenge.

Acknowledgements

This research was made possible with the support of the Research and Development Department at Brownien Laboratory, and we extend our gratitude for their invaluable contributions. We also acknowledge the use of resources provided by NT Tele-com PCL, which were instrumental in the completion of this study. Additionally, this project was supported by FPGA hardware (U55C) provided by Xilinx, whose technological assistance was crucial to our research endeavors.

Declaration of Conflicting Interests

The authors declared that they have no conflicts of interest in the research, authorship, and this article's publication.

References

1. Yi G, Loia V. High-performance computing systems and applications for AI. 2019.
2. Ferro M, Kloh VP, de Paula FB, Schulze B. Artificial Intelligence and High Performance Computing Convergence. 2019.
3. Jung D, Lee D, Kim M, Kim J. Efficient data synchronization method on integrated computing environment. 2018
4. Lee S, Huh JH. An effective security measures for nuclear power plant using big data analysis approach. 2018.
5. Yin C, Wang H, Yin X, Sun R, Wang J. Improved deep packet inspection in data stream detection. 2018.
6. Hwang S. A network clock model for time awareness in the Internet of things and artificial intelligence applications. 2019.
7. Hu S, Zhu Y, Cheng P, Guo C, Tan K, Padhye J, et al. Deadlocks in Datacenter Networks: Why Do They Form, and How to Avoid Them. Microsoft, Hong Kong University of Science and Technology.
8. Alveo U55C High Performance Compute Card. Available from: https://www.xilinx.com/products/boards-and-kits/alveo/u55c.html#specifications.
9. Keller AM, Wirthlin MJ. Impact of Soft Errors on Large-Scale FPGA Cloud Computing. In: ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA). New York, NY, USA: Association for Computing Machinery; 2019. p. 272–81. Available from: https://doi.org/10.1145/3289602.3293911.
10. Buell D, El-Ghazawi T, Gaj K, Kindratenko V. High-Performance reconfigurable computing. IEEE Computer Society. 2007 Mar.
11. Altera Cooperation White Paper. Accelerating high performance computing with FPGAs. 2007 Oct.
12. Kumbhar DD, Je Y, Hong S, Lee D, Kim H, Kwon MJ, et al. Molecularly Reconfigurable Neuroplasticity of Layered Artificial Synapse Electronics. Adv Funct Mater. 2024;2311994.
13. Liu X, Wang R, Shi C, Zou C, Zhu W. Computing Acceleration to Genome-Wide Association Study Based on CPU/FPGA Heterogeneous System. ACM SIGAPP Appl Comput Rev. 2024;23(4):16-26.

14. Li Y, Han D, Cui M, Yuan F, Zhou Y. RESNETCNN: An abnormal network traffic flows detection model. Comput Sci Inf Syst. 2023;(00):4-4.
15. Hussen N, Elghamrawy SM, Salem M, El-Desouky AI. A Fully Streaming Big Data Framework for Cyber Security based on Optimized Deep Learning Algorithm. IEEE Access. 2023.
16. Sultan MT, El Sayed H. QoE-aware analysis and management of multimedia services in 5G and beyond heterogeneous networks. IEEE Access. 2023.
17. Zhang L, Yang J, Wang T, Sun Z, Sun K, Zeng J. Event Building Algorithm in a Distributed Stream Processing Data Acquisition Platform: D-Matrix. IEEE Trans Nucl Sci. 2023;70(2):105-12.
18. Jayashankara M, Shah P, Sharma A, Chanak P, Singh SK. A Novel Approach for Short-Term Energy Forecasting in Smart Buildings. IEEE Sens J. 2023;23(5):5307-14.
19. Potnurwar AV, Bongirwar VK, Ajani S, Shelke N, Dhone M, Parati N. Deep Learning-Based Rule-Based Feature Selection for Intrusion Detection in Industrial Internet of Things Networks. Int J Intell Syst Appl Eng. 2023;11(10s):23-35.
20. Alonso T, Petrica L, Ruiz M, Petri-Koenig J, Umuroglu Y, Stamelos I, et al. Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds through Automatic Partitioning. ACM Trans Reconfigurable Technol Syst. 2021 Dec;15(2).
21. Agiakatsikas D, Foutris N, Sari A, et al. Evaluation of Xilinx Deep Learning Processing Unit under Neutron Irradiation. In: 21st European Conference on Radiation and Its Effects on Components and Systems (RADECS); 2021 Sep 13-17.
22. Li Y, Liu Z, Xu K, Yu H, Ren F. A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks. ACM J Emerg Technol Comput Syst. 2018;14(2):1-16.

