Optimizing Real-Time Object Detection in a Multi NPU System with Double Buffering and Queue-Based Processing

Sehyeon Oh
University of Science and Technology

Yongin Kwon
Electronics and Telecommunications Research Institute

Jemin Lee*
Electronics and Telecommunications Research Institute

Abstract

Real-time object detection demands high throughput and low latency, necessitating the use of hardware accelerators. In this paper, we construct a real-time object detection system based on YOLOv3 utilizing Neubla's Antara NPU and propose two approaches for performance optimization. First, we ensure the continuity of NPU inference by allowing the CPU to perform preprocessing in advance through double buffering. Second, in a multi-NPU environment, we distribute tasks among NPUs through queue-based processing and analyze the performance limits using Amdahl's law. Experimental results demonstrate that compared to a CPU-only environment, applying the NPU in single buffering improved throughput by 2.13 times, double buffering by 3.35 times, and in a multi-NPU environment by 4.81 times. Latency decreased by 1.6 times in single and double buffering, and by 1.18 times in the multi-NPU environment. The accuracy remained consistent, with 31.4 mAP on the CPU and 31.8 mAP on the NPU.

1 Introduction

Recent advancements in artificial intelligence have increased demand for real-time video analysis and object detection applications [1, 2]. However, as deep learning models become more complex, traditional CPU or GPU-based systems struggle to meet real-time processing requirements [3]. To address these challenges, specialized hardware accelerators, such as Google's TPU [4], NVIDIA's Tensor Core [5], and Intel's Nervana NNP [6] have been developed. These accelerators enhance computational performance and energy efficiency through specialized architectures. NPUs, offering excellent energy efficiency and processing speed, play a crucial role in high-performance deep learning applications [7, 8].

However, in real-time object detection systems, effectively utilizing the NPU requires cooperation and task synchronization with the CPU. These systems typically follow sequential stages of preprocessing, inference, and postprocessing. During preprocessing, the input data is transformed, followed by the NPU performing inference, and finally, the results are interpreted in the postprocessing stage. However, frequent data transmission and synchronization between the CPU and NPU can lead to performance bottlenecks. In particular, if each processor must wait for the other to complete its task before proceeding, latency increases, and system resources remain idle. This issue can reduce the overall system throughput and make it difficult to achieve real-time performance.

In this paper, to address these challenges, we propose a real-time object detection system based on YOLOv3 utilizing Neubla's Antara NPU, with performance optimization achieved through a double buffering technique and a queue-based processing scheme for multi-NPU environments. The double buffering technique mitigates data processing bottlenecks by allowing the NPU to continue inference while the CPU performs preprocessing, alternating between two buffers. This enables parallel task execution between the CPU and NPU, maximizing system performance. The queue-based processing scheme uses input and output queues to distribute tasks evenly across NPUs in a multi-NPU environment. The CPU preprocesses images captured from the camera and stores them in the input queue, from which the NPUs retrieve data and perform inference. The inference results are then stored in the output queue, where the CPU processes and visualizes them. This approach reduces waiting times between the CPU and NPU and significantly improves throughput in multi-NPU environments.

The experimental results showed that, compared to a CPU-only system, the Antara NPU system with single buffering achieved approximately 2 times higher throughput, while the double buffering system achieved about 3.4 times higher throughput. Latency in both single and double buffering systems was reduced by approximately 1.6 times compared to the CPU. In a multi-NPU environment, adding NPUs increased throughput by about 4.6 times compared to the CPU-only system; however, performance did not increase linearly due to CPU resource limitations. Additionally, inference using

*Corresponding Author: leejaymin@etri.re.kr

quantized models on the NPU maintained stable accuracy, with a slight increase from 31.4 to 31.8 mAP.

2 Architecture and Execution Flow of Antara NPU

We utilize the Antara NPU, developed by Neubla [9] in South Korea. This hardware accelerator is optimized for real-time deep learning applications such as image classification, object detection, and super-resolution. It is designed to deliver high computational performance and efficient memory utilization. The Antara NPU features a 32x48 MAC array, supporting Int8/FP8 data types, and achieves 6 TOPS of computational power with 1.5MB of on-chip SRAM to maximize bandwidth efficiency between computation and memory. Additionally, its 3,000 MAC units provide fast and accurate inference performance. The specifications are summarized in Table 1.

Table 1: Neubla's HW Specifications.

Feature	Specification
Performance	6 TOPS
Embedded SRAM	1.5MB
# of MACs	3K (1.5K x 2)
TX Engine	32 x 48 MAC array
Data Type	Int-8 or FP-8

As shown in Figure 1, the internal architecture of the Antara NPU consists of Neural Engines, shared SRAM, a DMA engine, and DRAM. Data is transferred from the host system to the DRAM on the Antara board via the PCIe interface, enabling the real-time processing of large-scale data. Additionally, the shared SRAM allows for efficient data exchange between the Neural Engines and the DMA engine, optimizing memory bandwidth efficiency. This architecture minimizes latency while providing the high throughput required for real-time applications, ensuring that the Antara NPU delivers the performance necessary for high-performance real-time deep learning applications.

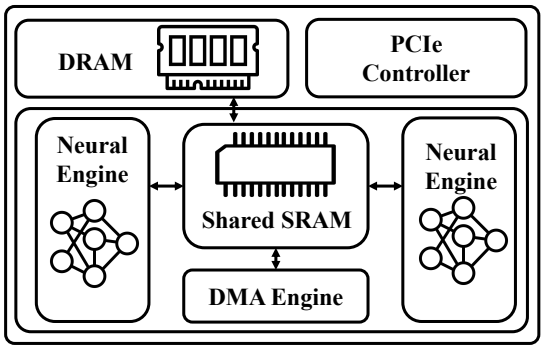

Figure 1: Neubla's Antara NPU Architecture

As shown in Figure 2, The software stack includes a dedicated compiler that partitions the workload between the CPU and the NPU, enabling the execution of deep learning models on the Antara NPU. This compiler analyzes the model to divide it into parts that run on the CPU and the NPU, generating target binary files for the NPU execution. During this process, the CPU still handles preprocessing and postprocessing tasks, making efficient scheduling at the application level critical for coordinating tasks between the CPU and NPU. Our approach focuses on optimizing this scheduling to minimize idle time between CPU and NPU operations, thereby maximizing throughput. Further details on the pipeline optimization techniques are provided in the subsequent sections.

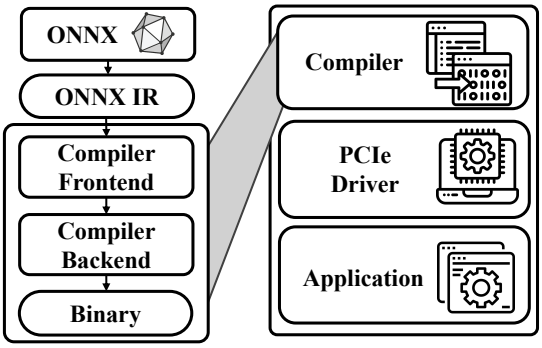

Figure 2: Neubla's Antara NPU Software Stack

3 Best Practices for Efficient Object Detection on Antara NPU

To maximize the performance of real-time object detection systems, we conducted a thorough analysis and optimization of three key computational stages: preprocessing, inference, and postprocessing. In a typical processing flow, the CPU preprocesses input images before sending them to the NPU for inference, and then handles postprocessing of the results to provide the final output. In the initial implementation, a single-buffering approach was used to sequentially execute tasks between the NPU and CPU. As shown in Figure 3 (a), the CPU completes preprocessing before sending data to the NPU, and once the NPU finishes inference, the CPU performs postprocessing. However, in this approach, if the CPU does not prepare the next input data before the NPU completes its inference, the NPU remains idle, potentially creating performance bottlenecks. This reduces system throughput and prevents the NPU from being fully utilized.

To address this performance bottleneck, a double-buffering technique was introduced to enable parallel processing between the CPU and NPU. This approach alternates between two buffers: while the CPU preprocesses input data in one buffer, the NPU performs inference on the data in the other buffer. This allows both the CPU and NPU to operate simultaneously, minimizing NPU idle time and significantly improv-

ing system processing speed. Particularly, in time-sensitive applications such as real-time object detection, this method reduces latency and contributes to performance optimization. As shown in Figure 3 (b), double buffering maximizes resource utilization and effectively increases throughput.

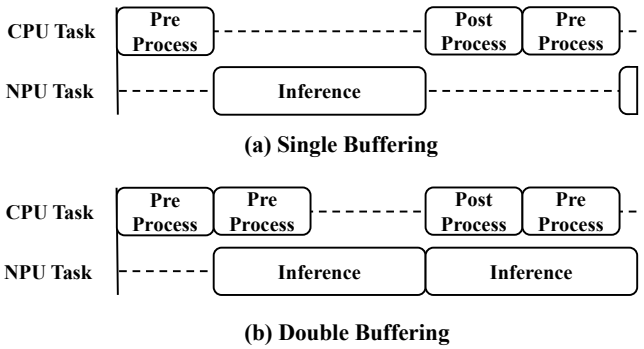

Figure 3: Single buffering with sequential operation and double buffering for parallel processing, where the timing differences between CPU and NPU tasks in each method are presented.

To achieve performance scaling in a multi-NPU environment, we implemented a queue-based processing strategy, as shown in figure 4. The CPU captures images from the camera in real-time and stores them in the input queue. Subsequently, the CPU handles preprocessing and postprocessing, while the NPU performs inference. These three stages are combined into a single task for each image until processing is complete. Each task retrieves data from the input queue, processes it, and stores the final output in the output queue, after which the CPU visualizes the results. This queue-based approach efficiently distributes the workload across the NPUs, maximizing the utilization of system resources. As each NPU processes data independently, increasing the number of NPUs can lead to an increase in system throughput. However, this performance gain becomes limited as the number of NPUs increases, due to constraints on system resources such as CPU, memory bandwidth, and I/O. Therefore, while the system can scale efficiently up to a certain point, it is important to recognize that performance may not improve proportionally with the addition of more NPUs [10].

Figure 4: Data Distribution through Input and Output Queues in a Multi-NPU System.

4 Experiment

4.1 Environment Setup

To quantitatively evaluate the performance of a real-time object detection system and compare the performance differences across various hardware configurations, we designed experiments using an Intel Core i7-12700 CPU, an NVIDIA GeForce GTX 1060 6GB GPU, and a Neubla Antara NPU. The experiments were conducted based on the YOLOv3 608×608 model, which was utilized in its uint8 quantized version. Input data was provided in real time via a Logitech Brio 4K Pro camera, and all software was configured to run in a consistent environment to ensure the reliability of the experiments. The operating system used was Ubuntu 22.04, and inference on the CPU and GPU was performed using ONNX Runtime 1.12.1. For the GPU, inference was executed in an environment with CUDA 11.8 and cuDNN 8.9. For the NPU, inference was conducted using binaries generated through the Neubla compiler.

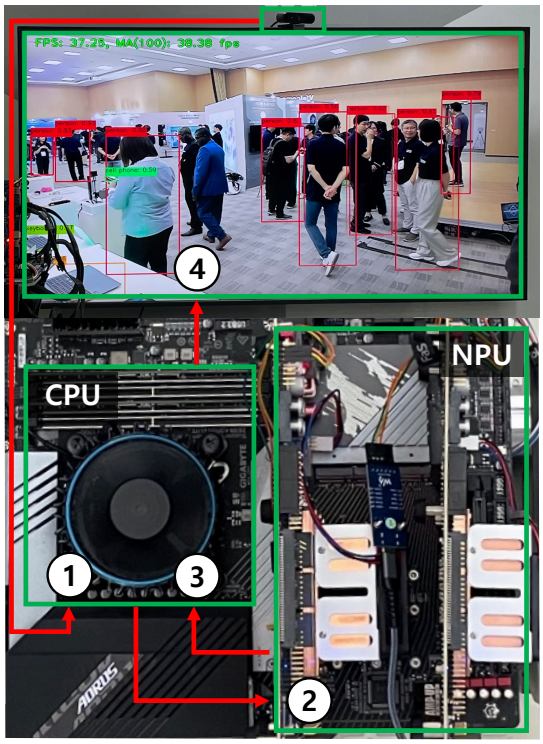

Figure 5: Four-stage execution flow: (1) Data acquisition and preprocessing, (2) Inference, (3) Postprocessing, and (4) Real-time result output with object detection and performance metrics.

The overall execution flow is shown in Figure 5. This flow consists of four stages: ❶ **Data Acquisition and Preprocessing**, where real-time video data is collected from the camera, and the CPU performs preprocessing steps such as

image resizing and data normalization. ❷ **Inference**, during which the preprocessed data is sent to the NPU or GPU for inference using the deep learning model. ❸ **Postprocessing**, in which the inference results are returned to the CPU, where tasks such as drawing bounding boxes around detected objects and organizing class information are performed. ❹ **Result Output**, where the postprocessed results are displayed on the screen in real-time, with detected objects indicated by bounding boxes and performance metrics such as Frames Per Second (FPS) provided.

For performance evaluation, we measured the mean Average Precision (mAP) using the COCO 2017 validation dataset [11]. In addition, we assessed throughput (FPS) and latency for each hardware configuration to comprehensively analyze the performance of the real-time system. This allowed us to compare the inference performance across CPU, GPU, and NPU environments, providing a quantitative understanding of the performance differences between the hardware components.

4.2 Evaluation

This section evaluates the performance differences across various hardware configurations and provides a comprehensive analysis of the throughput (FPS), latency, and accuracy for each configuration. The performance comparison is summarized in Table 2. The CPU and GPU configurations showed similar throughput, with 11.14 FPS and 11.17 FPS, respectively, but differed in terms of latency. The CPU recorded a latency of 99.73 ms, whereas the GPU had a higher latency of 128.22 ms. This discrepancy is likely due to the memory transfer and computational overhead associated with the GPU. The accuracy (mAP) was identical for both configurations, with a score of 31.4 mAP.

When using the NPU, the performance in the single buffering (SB) configuration shows that preprocessing takes 7.15ms, inference takes 25.76ms, and postprocessing takes 8.05ms, resulting in a total processing time of 40.96ms. Based on this, the calculated FPS is 24.41, which represents approximately 2.13 times the performance improvement compared to using the CPU alone. In the case of double buffering (DB), since each stage is processed in parallel, the overall processing time is determined by the longest stage, which is the inference time of 25.76ms. Therefore, the calculated FPS is 38.79, which is close to the actual measured value of 38.38 FPS. This indicates that double buffering improves throughput by approximately 3.35 times compared to CPU-only usage, and by 1.58 times compared to single buffering. In terms of latency, the measured latency for double buffering is 37.77ms, which shows little difference from the 38.34ms observed in single buffering. This demonstrates that double buffering increases throughput without significantly increasing latency.

In a multi-NPU environment, we conducted experiments using two NPUs and optimized data distribution through a queue-based processing method. As a result, the throughput increased to 55.04 FPS. However, the improvement was sublinear, indicating that the CPU's preprocessing and postprocessing tasks became bottlenecks. Analysis showed that approximately 63% of the total workload is parallelizable, primarily corresponding to the inference stage. Applying Amdahl's Law [12], which relates the proportion of parallelizable tasks to overall performance improvement, we calculated the theoretical speedup using the formula $S = \frac{1}{(1-P)+\frac{P}{N}}$, where $P = 0.63$ and $N = 2$. Substituting these values, we obtain a speedup of 1.46. Therefore, the expected FPS is 56.03, which closely matches the actual measured value of 55.04 FPS. This demonstrates that even with multiple NPUs, performance improvement is limited by the non-parallelizable portions of the workload and CPU resource constraints.

The accuracy difference between hardware configurations was minimal, with both the CPU and GPU recording 31.4 mAP, and the NPU slightly higher at 31.8 mAP. This small variation may be attributed to differences in how quantized models are executed across hardware [13]. Notably, the NPU maintained model accuracy while improving processing speed and reducing latency.

Table 2: Performance comparison across different hardware configurations. SB and DB denote Single Buffering and Double Buffering, respectively.

HW Config.	Throughput (FPS)	Latency (ms)	Accuracy (mAP)
CPU	11.14	99.73	31.4
GPU	11.17	128.22	31.4
NPU (SB)	24.41	38.34	31.8
NPU (DB)	38.38	37.77	31.8
NPUx2 (DB)	55.04	81.88	31.8

5 Conclusion

In this paper, we propose two approaches to optimize the performance of a real-time object detection system utilizing the Neubla Antara NPU. First, by employing double buffering, the CPU performs preprocessing in advance, ensuring continuity in NPU inference. Second, in a multi-NPU environment, a queue-based processing scheme is introduced to distribute tasks across NPUs, and the performance limits are analyzed using Amdahl's Law. Experimental results show that, compared to the CPU-only system, the throughput increased by 2.13 times with single buffering, 3.35 times with double buffering, and 4.81 times in the multi-NPU environment. Latency was reduced by 1.6 times with single and double buffering, and by 1.18 times in the multi-NPU setup. Accuracy remained consistent, with 31.4 mAP on the CPU and 31.8 mAP on the NPU.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2024-00459797, Development of ML compiler framework for on-device AI) and (No.RS-2023-00277060, Development of open edge AI SoC hardware and software platform).

References

[1] Xiongwei Wu, Doyen Sahoo, and Steven CH Hoi. Recent advances in deep learning for object detection. *Neurocomputing*, 396:39–64, 2020.

[2] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 111(3):257–276, 2023.

[3] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[4] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[5] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[6] Brian Hickmann, Jieasheng Chen, Michael Rotzin, Andrew Yang, Maciej Urbanski, and Sasikanth Avancha. Intel nervana neural network processor-t (nnp-t) fused floating point many-term dot product. In *2020 IEEE 27th Symposium on Computer Arithmetic (ARITH)*, pages 133–136. IEEE, 2020.

[7] Kyuho J Lee. Architecture of neural processing unit for deep neural networks. In *Advances in Computers*, volume 122, pages 217–245. Elsevier, 2021.

[8] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020.

[9] Hyeryeong Kang. Hanwha establishes semiconductor company neubla... advancing in system semiconductors. https://www.sedaily.com/NewsView/26250GNIMV, 2022.

[10] Mohammed A Noaman Al-hayanni, Fei Xia, Ashur Rafiev, Alexander Romanovsky, Rishad Shafik, and Alex Yakovlev. Amdahl's law in the context of heterogeneous many-core systems–a survey. *IET Computers & Digital Techniques*, 14(4):133–148, 2020.

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[12] Mark D Hill and Michael R Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, 2008.

[13] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.