A Detailed Analysis of LLM Execution on IMAX3 and Initial Evaluation of IMAX4 Prototype for Server Environment

Takuto ANDO Yu ETO Yasuhiko NAKASHIMA

Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma, Nara, 630-0192 Japan
ando.takuto.aq5@naist.ac.jp eto.yu.ew0@naist.ac.jp nakashim@is.naist.jp

Abstract— In this paper, we present a detailed analysis of the IMAX3 CGRA-based accelerator and the IMAX4 prototype with an upgraded host CPU. To address the host CPU bottlenecks of its predecessor, IMAX3, IMAX4 incorporates a server-oriented Intel Xeon processor and PCIe Gen5 connectivity, realizing IMAX scalability. We implemented and evaluated the IMAX4 prototype using microbenchmarks and the LLaMA3 8B quantized model. The results demonstrate significantly reduced host-side overheads and improved data transfer compared to IMAX3. While performance characteristics vary by quantization, IMAX4 demonstrates significant potential by shifting bottlenecks from the host to data pathways, highlighting the viability of CGRA for server-based LLM acceleration.

I. Introduction

In recent years, AI (Artificial Intelligence) technology has made remarkable progress and is widely applied in a variety of fields, such as image recognition and natural language processing. In particular, LLMs (Large Language Models) have the ability to engage in natural dialogue with humans and to perform sophisticated tasks such as sentence generation, translation, and question-and-answer sessions. Therefore, LLMs have attracted a great deal of attention not only in academia but also in industry [1, 2].

Currently, LLM computation is mainly performed using CPUs (Central Processing Units) and GPGPUs (General-purpose Graphics Processing Units). Shehabi et al. suggest that the energy consumption of data centers in the U.S. in 2028 will account for 6.7 % to 12 % of the estimated power consumption [3]. It is becoming difficult to achieve high performance per power with existing CPU and GPU architectures, and it is essential to develop new accelerator technologies more energy efficient. With this background, we proposed a CGRA (Coarse Grained Reconfigurable Array)-based accelerator architecture, IMAX3 (In-Memory Accelerator eXtension 3), which aims to achieve both high power efficiency and program flexibility. IMAX is an accelerator that features a linear structure of alternating cache memory and processing units to maximize LMM (Local Memory Module) utilization.

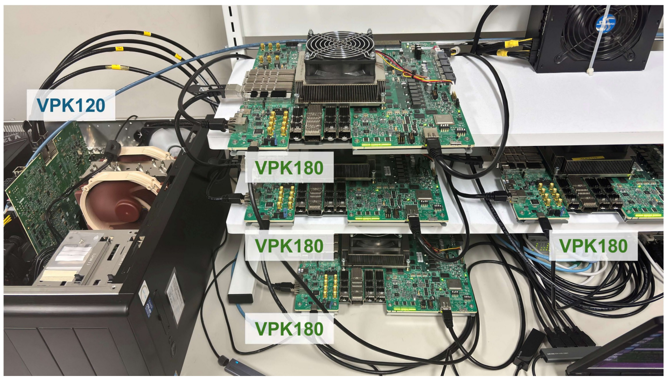


Fig. 1. IMAX4 Prototype using Intel Xeon processor

Eto et al. have evaluated LLaMA2 7B-based LLMs using IMAX3 on an FPGA and found that it has the potential to demonstrate good power efficiency compared to CPUs and GPUs for 1-thread and 2-threads [4]. However, the 2-core ARM CPU host system of IMAX3 had performance limitations in terms of controlling complex applications like LLMs, supplying data to the IMAX, and scalability in multi-threaded processing. These issues have been a major challenge in deploying the IMAX architecture for server environments requiring higher-performance computation. In this work, we propose an IMAX4 prototype as shown in Fig. 1 that upgrades the host CPU to an Intel Xeon server-oriented processor and utilizes one VPK120 with four VPK180 boards.

The objectives of this work are twofold. First, to identify the bottleneck of IMAX3 through a detailed runtime analysis using the more recent and generic LLaMA3 model. The second is to obtain initial insight into the proposed IMAX4 prototype with an updated host CPU to confirm its basic operation and evaluate its preliminary performance.

The main contributions of this paper are as follows:

- We designed and implemented a server-oriented IMAX4 prototype. This system features a redesigned host system, transitioning from the conventional 2-core ARM CPU to an 16-core Intel Xeon processor, and employs a PCIe Gen5 interface. This design, based on a multi-lane IMAX architecture, also ensures future scalability.

- We conducted a detailed evaluation and analysis of the execution performance in the LLaMA3 8B quantized model on IMAX4. The performance characteristics of IMAX3 and IMAX4 in LLM were clarified through a detailed analysis of end-to-end latency and matrix product processing time.

- We quantified the performance improvement of IMAX by host CPU enhancements. Through comparison of IMAX3 and detailed processing time breakdown analysis, we identified current performance-limiting factors.

The rest of this paper is organized as follows. Section II provides related work on LLM accelerators and the previous work on LLM execution using the IMAX3. Section III describes the architecture of the IMAX4 prototype. Section IV presents the results of our performance evaluations, including fundamental characterization via microbenchmarks and LLM execution evaluations using LLaMA3. Finally, Section V concludes our work.

II. Related Work

A. *LLM Accelerators*

Currently, GPUs are widely utilized for LLM learning and inference. However, the deployment costs and power consumption of high-performance GPUs remain high. Therefore, there is growing interest in alternative architectures that offer greater power efficiency, especially for LLM inference in specific applications.

In this background, research and development of accelerators using reconfigurable devices such as FPGAs (Field Programmable Gate Arrays) and CGRAs are actively being pursued. For example, Xu et al. proposed "LlamaF," an FPGA-based accelerator specialized for Llama2 models, demonstrating efficient execution in embedded environments [5]. Additionally, several works have reported on accelerating the attention mechanism and Feed-Forward networks, key components of Transformer models on FPGAs [6]. While these works attempt to achieve high efficiency by optimizing for specific operations, challenges remain in terms of versatility, scalability to large-scale models, and efficient coordination with host systems. This work focuses on CGRA-based accelerators, which offer a superior balance of architectural flexibility and power efficiency.

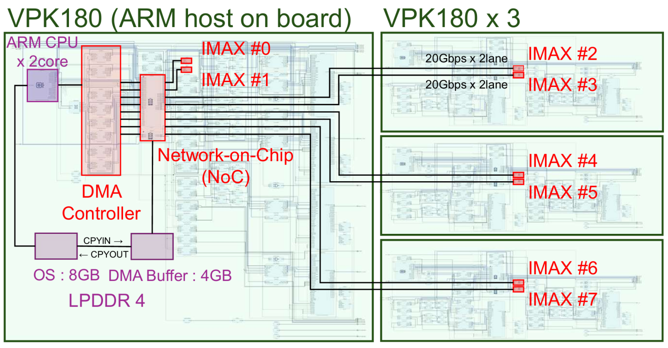


Fig. 2. IMAX3 configuration

B. *IMAX Architecture and LLM on IMAX3*

CGRA has the potential to combine the efficiency and flexibility of ASICs, but it has faced challenges such as scalability, compilation time, and loop implementation complexity [7]. To address these challenges, we proposed IMAX3 [8]. The basic design of IMAX is based on a linear array structure that alternates processing units and cache memory, aiming to thoroughly utilize LMM and improve efficiency of irregular memory accesses. The IMAX architecture features a multi-lane configuration consisting of multiple compute lanes to increase parallel processing power. As shown in Fig. 2, IMAX3 implements this 8-lane configuration on an AMD Versal VPK180x4. These lanes are allocated according to the thread count of the application, allowing for parallel execution.

In the previous work by Eto et al., LLaMA2 7B-based LLMs were evaluated on IMAX3. It was shown to have the potential to exhibit good power efficiency compared to existing CPUs and GPUs during 1-thread and 2-threads operation [4]. However, the onboard ARM Cortex-A72 dual-core processor used for host control lacked processing power. This limitation increased the overhead, especially during program control and multi-threaded operation on the host side, and significantly limited the overall system performance. Therefore, the current IMAX3 has an 8-lane configuration, but this number of lanes cannot be fully utilized.

This work conducts a more detailed analysis of existing IMAX3 and identifies its bottlenecks through an evaluation with the LLaMA3 8B-based LLM, the successor to the LLaMA2 7B model. Furthermore, as a step towards resolving these identified issues, the work aims to introduce the preliminary development of the IMAX4 prototype, targeted for high-performance server environments, and to indicate future pathways for architectural enhancements and subsequent evaluations.

III. IMAX4 Prototype

In this section, we proposed the design and implementation of IMAX4 as a new accelerator prototype for high-performance and scalable LLM execution in a server environment.

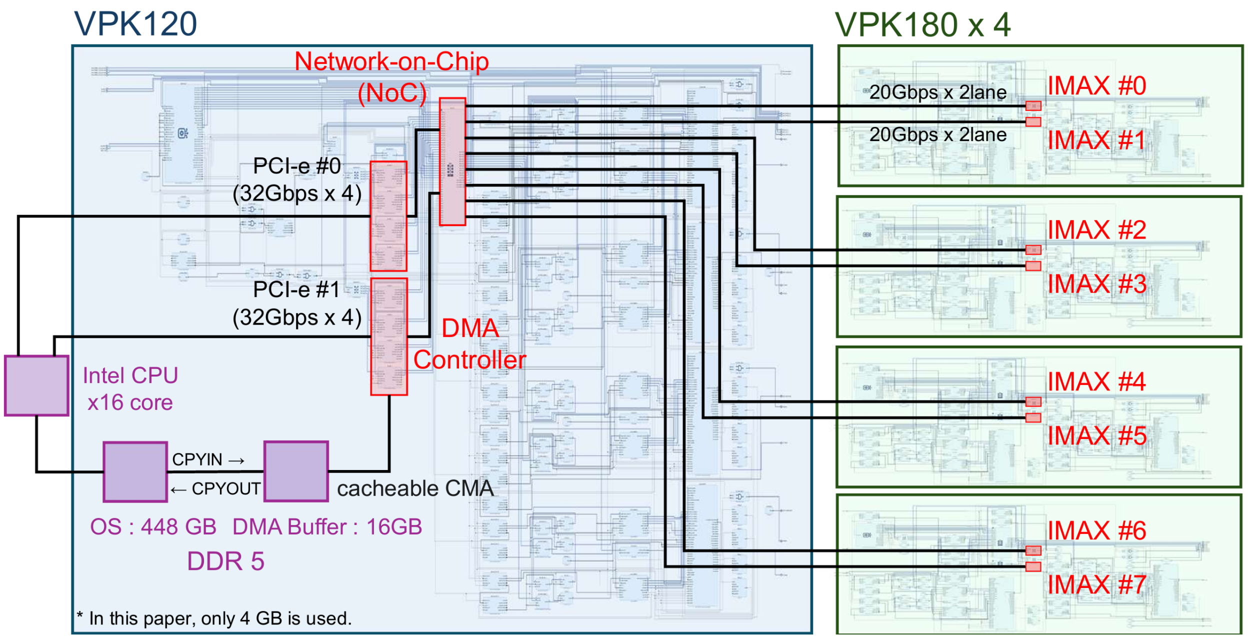


Fig. 3. IMAX4 configuration

A. *IMAX4 System Overview*

Fig. 3 shows the system configuration of IMAX4. It consists of the host processor, an Intel Xeon, one AMD Versal VPK120 evaluation board that functions as a PCIe bridge, and four AMD Versal VPK180 evaluation boards implemented with IMAX. The most significant change in IMAX4 is the revamped host CPU. The on-board 2-core Arm Cortex-A72 processor used in IMAX3 was replaced by a server-class 16-core Intel Xeon processor. This change significantly increases the host-side computing power and reduces the overhead associated with complex control flows and data pre-processing and post-processing during LLM execution.

In addition, the AMD Versal VPK180 used in IMAX3 and the IMAX4 prototype lacks a direct PCIe Gen5 interface. Therefore, the AMD Versal VPK120 evaluation board is used as a PCIe bridge. The VPK120 is connected to the host system via PCIe Gen5 for high-speed data relay and control with the downstream VPK180 group. Furthermore, a 16 GB DMA buffer supporting cacheable CMA was allocated to bolster the data supply infrastructure. While this work uses only 4 GB for the convenience of the experiment, the full 16 GB capacity will be utilized in future implementations to maximize data transfer performance.

B. *Data Transfer Infrastructure*

IMAX4 features a significantly enhanced host-accelerator data transfer infrastructure, crucial for maximizing performance in high-bandwidth applications like LLMs. IMAX3 employed an ARM SoC host with a 32-bit LPDDR4 memory interface (4 Gbps/lane), yielding a theoretical maximum bandwidth of 128 Gbps. Data supply to IMAX3 relied on this LPDDR4 interface and a single internal SoC NoC data path, limiting overall system throughput.

In contrast, IMAX4 significantly improves data transfer capability by adopting PCIe Gen5 for its Intel Xeon processor connection. This configuration was chosen over an alternative that uses PCIe Gen4 x16 lanes. Consequently, we prioritized the higher per-lane transfer rate of PCIe Gen5 to ensure a simple bus architecture.

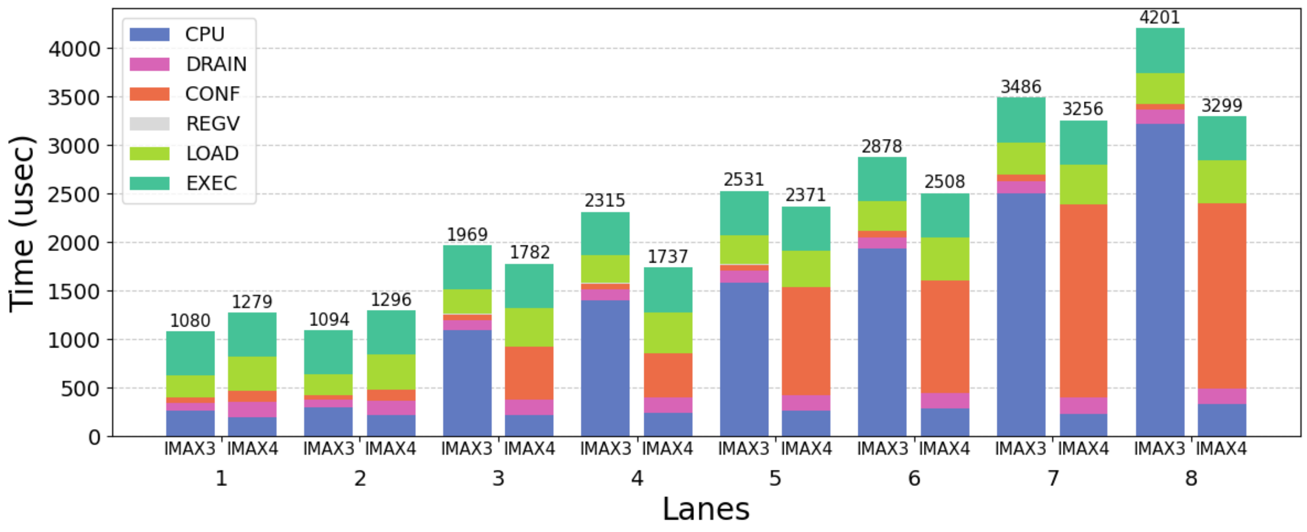
Furthermore, IMAX4 leverages increased parallel data transfer capability through its multiple PCIe bus lines. Unlike IMAX3 single data path, IMAX4 assigns its two PCIe Gen5 links to distinct groups. One link (PCI-e#0) serves units 0-3, and another (PCI-e#1) serves units 4-7. This allows parallel data supply to, and result retrieval from, multiple IMAX lanes, significantly reducing data transfer overhead and boosting overall accelerator utilization efficiency.

IV. Experiments and Results

This experiment aims to identify the bottlenecks of IMAX3 through detailed analysis and to clarify the improvements made in IMAX4. To achieve this, we first conducted a basic evaluation using microbenchmarks. Subsequently, we performed an application-level evaluation using LLMs.

A. *Fundamental Evaluation by Microbenchmark*

The primary objective of this microbenchmark is to evaluate the data transfer performance to the 512KB LMM integral to each IMAX computation unit. To fully utilize the LMM capacity, the benchmark processes a total of 512 KB of data per lane, configured as three 128 KB input arrays (A, B, C) and one 128 KB output array (D). During the evaluation, each IMAX lane asynchronously executes a FMA (Fused Multiply - Add) operation, which can be expressed as $D = C + A \times B$. The execution time of a process in IMAX3 is determined by the total time of



CPU:	CPU processing	**DRAIN**:	Cache ⇒ Main memory transfer	**CONF**:	IMAX command transfer
REGV:	Register initialization	**LOAD**:	Main memory ⇒ Cache transfer	**EXEC**:	Burst calculation

Fig. 4. Processing time breakdown comparison of IMAX3 and IMAX4

the following components: CPU, DRAIN, CONF, REGV, LOAD, and EXEC. A detailed description of these components is shown in Fig. 4. To ensure robust measurements, each program execution was performed 10 times, and the average processing time for each component was recorded and analyzed. Fig. 4 shows the time for each execution phase by number of lanes. In IMAX3, the CPU processing time increased with increasing host processing load with the number of lanes. On the other hand, IMAX4 significantly reduced the CPU, especially for three or more lanes. However, the CONF of IMAX4 was longer than that of IMAX3. This is considered to be partly due to the low-frequency operation of PIO at 100 MHz to avoid DMAC and PIO contention on the NoC.

Next, we compare the effective data transfer rates during 1-lane execution, as detailed in Table I. IMAX3 achieved an effective LOAD transfer rate of approximately 13.9 Gbps and a DRAIN rate of approximately 13.0 Gbps. In contrast, IMAX4 effective LOAD rate was approximately 8.7 Gbps, and its DRAIN rate was approximately 6.9 Gbps. While IMAX3 rates demonstrated a degree of efficiency relative to the AXI interface theoretical peak bandwidth to the FPGA-internal LMM (approximately 37.1 Gbps for a 145 MHz, 256 bit interface), IMAX4 rates were notably lower. These results confirm the presence of a more significant bottleneck in IMAX4 data transfer path compared to IMAX3.

Overall, IMAX4 resolved the control overhead issues of IMAX3 by improving host CPU processing capability. However, it also revealed new challenges, such as PIO transfer overhead and a data transfer rate that is lower than both AXI theoretical performance and IMAX3.

TABLE I
Data Transfer Time and Effective Bandwidth Comparison

Transfer Type	Device	Time [μs]	Rate [Gbps]
LOAD : 384KB	IMAX3	220.3	13.9
	IMAX4	351.8	8.7
DRAIN : 128KB	IMAX3	78.7	13.0
	IMAX4	149.1	6.9

B. *LLM Execution Evaluation using LLaMA3*

To assess practical LLM acceleration of IMAX, we compared IMAX3 and IMAX4 using the llama.cpp framework [9] with the LLaMA3 8B Q3_K_S and Q8_0 quantization models. Evaluations consistently used ”Hello” as the input prompt, 32 predicted tokens, and a seed value of 1. Primary metrics were end-to-end latency and matrix multiplication function processing time on IMAX and CPU. It is important to note that the matrix multiplication function processing time, a key metric, encompasses data handling and control overhead, not solely raw IMAX computation time. Llama.cpp executes matrix products on IMAX cores via SIMD instructions after LMM data transfer, with quantized data processed through an optimized data flow.

Table II shows the evaluation results for the Q3_K_S quantization model, and Table III for the Q8_0 quantization model. First, we check the performance characteristics of IMAX3. For a relatively smaller model such as Q3_K_S, IMAX3 latency was relatively small during multi-threaded execution, achieving 49.2 seconds at 2 lanes. However, as a limitation of the host CPU, IMAX3 exhibits very long CPU processing time for both the Q3_K_S and Q8_0 models.

Next, we examine the performance of IMAX4. For the Q3_K_S model, the latency for IMAX4 at 1-lane execution was 172.0 s, which was greater than that of IMAX3.

TABLE II
PERFORMANCE COMPARISON OF IMAX3/4
ON Q3_K_S LLM INFERENCE

Configuration		Latency [s]		Processing Time [s]	
Device	Lanes			IMAX	CPU
IMAX3	1	88.6	-	64.3	20.4
IMAX4	1	172.0	-	164.1	7.0
IMAX3	2	49.2	[1] −44%	33.3	12.3
IMAX4	2	87.6	−49%	82.6	4.0
IMAX3	3	56.3	−36%	25.9	24.7
IMAX4	3	252.3	+47%	248.4	2.8
IMAX3	4	60.3	−32%	21.6	31.9
IMAX4	4	296.3	+72%	292.5	2.2
IMAX3	5	65.1	−26%	20.2	35.3
IMAX4	5	354.7	+106%	350.9	2.0
IMAX3	6	67.1	−24%	17.5	39.2
IMAX4	6	322.4	+87%	318.2	1.9
IMAX3	7	69.4	−22%	16.9	40.0
IMAX4	7	387.3	+125%	383.7	1.7
IMAX3	8	70.2	−20%	14.4	42.6
IMAX4	8	350.9	+103%	347.3	1.6

TABLE III
PERFORMANCE COMPARISON OF IMAX3/4
ON Q8_0 LLM INFERENCE

Configuration		Latency [s]		Processing Time [s]	
Device	Lanes			IMAX	CPU
IMAX3	1	1799.0	-	231.3	1462.4
IMAX4	1	221.3	-	205.1	15.3
IMAX3	2	1700.0	[1] −5.5%	553.4	1033.0
IMAX4	2	112.3	−49%	103.4	7.9
IMAX3	3	1638.6	−8.9%	692.1	830.4
IMAX4	3	187.8	−15.4%	180.5	5.9
IMAX3	4	1599.1	−11.1%	725.3	752.0
IMAX4	4	200.8	−9.5%	195.0	4.4
IMAX3	5	1579.1	−12.2%	725.1	730.1
IMAX4	5	189.0	−14.5%	184.0	3.8
IMAX3	6	1569.2	−12.8%	723.6	720.9
IMAX4	6	164.2	−25.8%	159.1	3.3
IMAX3	7	1559.5	−13.3%	725.7	703.1
IMAX4	7	217.0	−1.8%	212.2	3.0
IMAX3	8	1577.0	−12.3%	725.0	703.0
IMAX4	8	209.7	−5.4%	205.3	2.7

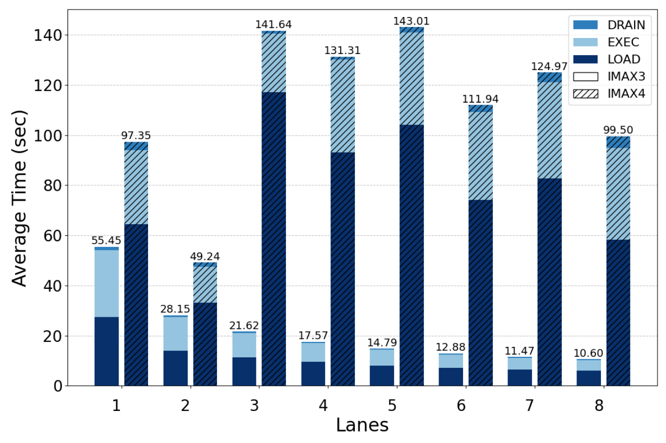


Fig. 5. Performance plot of IMAX4 and IMAX3 for Q3_K_S model inference

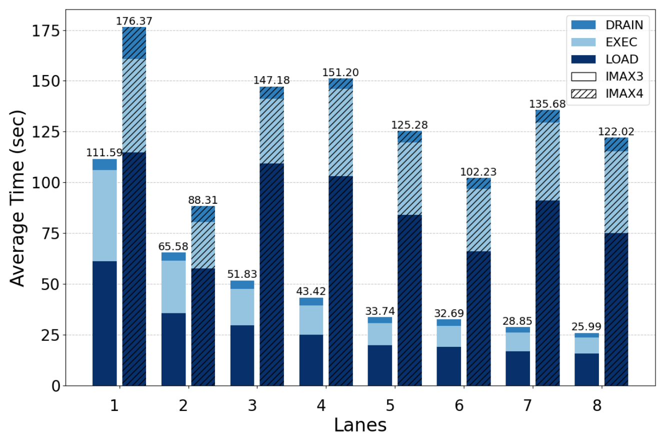


Fig. 6. Performance plot of IMAX4 and IMAX3 for Q8_0 model inference

Furthermore, as the number of lanes increased, the latency for IMAX4 tended to increase significantly more than that of IMAX3. Conversely, the host CPU processing time for IMAX4 was reduced compared to IMAX3, nearly eliminating the host-side computational bottleneck in IMAX4. With the Q8_0 model, IMAX4 achieved substantially lower latency than IMAX3 across all lane counts. While IMAX3 confirmed the impact of CPU multithreading overhead, there was no significant change in the IMAX core processing time for IMAX4, even when increasing lanes from one to eight. This outcome indicates that the enhanced host system of IMAX4 effectively functioned and overcame the host-side bottlenecks present in IMAX3. Additionally, regarding the impact of varying execution lane counts on IMAX4 with the Q8_0 model, latency significantly decreased from 221.3 s at 1 lane to 112.3 s at 2 lanes. However, from 3 lanes onwards, latency either plateaued or slightly increased. Similarly, the IMAX core processing time also remained nearly constant or showed a slight increase with additional lanes, failing to achieve ideal linear scaling. This behavior may be attributed to potential bottlenecks within the DMA and PIO data transfer paths.

[1]Percentage changes in latency compared to 1-lane configuration of the same device.

C. *Detailed Analysis of Processing Time in LLM Execution*

In this subsection, we first analyze the data transfer and processing times for IMAX. Fig. 5 and 6 show the LOAD, EXEC, and DRAIN times in IMAX3 and IMAX4 by number of lanes during LLaMA3 8B quantized model

TABLE IV
Performance comparison of IMAX3 and IMAX4 for CPYIN A, CPYIN B, and CPYOUT operations

Configuration		Operation Time [s]		
Device	Threads	CPYIN A	[2]CPYIN B	CPYOUT
IMAX3	1	1092.9	0	1.4
IMAX4	1	17.4	0	0.3
IMAX3	8	350.9	0.3	1.6
IMAX4	8	3	0	0.1

inference. In IMAX3, the processing time generally decreases and saturates as the number of lanes increases in both models, although it is limited by the host bottleneck. On the other hand, the processing time of the IMAX4 was reduced up to two lanes, but the performance improvement tended to plateau or worsen after three lanes. Although the host CPU bottleneck was eliminated, data transfer inside the FPGA emerged as a new major factor. Detailed analysis and improvement are future tasks.

To further analyze the host CPU-induced bottlenecks in IMAX3, we measured the data copy times between DDR memory and the DMA buffer during Q8_0 model execution. Specifically, we measured CPYIN A for model weight transfer, CPYIN B for input data transfer, and CPYOUT for retrieving computation results. Table IV shows that CPYIN A, the model weight transfer, consumed the most processing time. For IMAX3, copying CPYIN A required an exceptionally long 1092.9 s for 1-thread execution and 350.9 s even with 8-threads. In contrast, IMAX4 reduced this time to 17.4 s and 3 s, respectively. This substantial improvement is a direct result of the high processing capability of the Intel Xeon host and the optimized data transfer pathways in IMAX4.

V. Conclusion

In this work, we first identified the host CPU as the bottleneck for LLaMA3 on IMAX3. Next, we implemented and evaluated an IMAX4 prototype with the host system updated to an Intel Xeon processor. As a result of the experiments, IMAX4 significantly outperformed IMAX3 in end-to-end latency on a large model such as the LLaMA3 8B Q8_0. These results indicate that IMAX4's architectural improvements are effective in improving LLM execution performance in a server environment.

On the other hand, the effective data transfer rate to the LMM on IMAX4 is lower than the theoretical performance of the AXI interface and IMAX3. In future work, we aim to establish a competitive CGRA-based accelerator against GPGPUs by identifying and optimizing the data transfer performance bottleneck.

[2]This value is displayed as 0 seconds for clarity, but the actual duration is a few microseconds.

Acknowledgment

This work was supported by the JST-ALCA-Next Program (Grant Number JPMJAN23F4) and JSPS KAKENHI (Grant No. 22H00515). We also acknowledge the activities of VDEC, The University of Tokyo, in collaboration with NIHON SYNOPSYS G.K.

References

[1] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. LLaMA: Open and Efficient Foundation Language Models, 2023.

[2] Meta. Introducing Meta Llama 3: The most capable openly available LLM to date. Meta AI Blog, 2024, `https://ai.meta.com/blog/meta-llama-3/`, 2024. Accessed on May 11, 2025.

[3] A. Shehabi, S. Smith, A. Hubbard, A. Newkirk, N. Lei, M. A. B. Siddik, et al. 2024 United States Data Center Energy Usage Report. Technical Report LBNL-2001637, Lawrence Berkeley National Laboratory, 2024. Retrieved from `https://escholarship.org/uc/item/32d6m0d1`.

[4] Y. Eto and Y. Nakashima. Implementation and Performance Analysis of LLaMA on a CGLA. In *International Conference on Intelligent Systems and Networks (ICISN)*, 2025.

[5] H. Xu, Y. Li, and S. Ji. LlamaF: An Efficient Llama2 Architecture Accelerator on Embedded FPGAs. In *2024 IEEE 10th World Forum on Internet of Things (WF-IoT)*, pp. 1–7, 2024.

[6] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding. FTRANS: Energy-Efficient Acceleration of Transformers using FPGA, 2020.

[7] C. Torng, P. Pan, Y. Ou, C. Tan, and C. Batten. Ultra-Elastic CGRAs for Irregular Loop Specialization. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 412–425, 2021.

[8] T. Akabe, V. Trung Duong LE, and Y. Nakashima. IMAX: A Power-Efficient Multilevel Pipelined CGLA and Applications. *IEEE Access*, Vol. 13, pp. 31899–31911, 2025.

[9] G. Gerganov. GitHub-ggerganov/llama.cpp: LLM inference in C/C++. `https://github.com/ggerganov/llama.cpp`, 2023. Accessed on May 11, 2025.