GLITCHES: GPU-FPGA LLM Inference Through a Collaborative Heterogeneous System

Fan Yang*,1,2, Xinhao Yang*,1, Hongyi Wang[1], Zehao Wang[1], Zhenhua Zhu[1], Shulin Zeng[1], Yu Wang[1†]
[1]Dept. of EE, BNRist, Tsinghua University, [2]SenseTime Inc.
[†]Corresponding author: yu-wang@tsinghua.edu.cn

***Abstract*—Large language models (LLMs) demonstrate strong capabilities across various tasks. However, in latency-sensitive scenarios, a small batch or even one batch is usually required. This leads to the prefill and the decode stage of LLM inference being computational and memory bottlenecks, respectively. Therefore, it is difficult for a homogeneous FPGA or GPU system to simultaneously address different computational bottlenecks in different stages of LLM inference, resulting in long prefill latency on FPGAs and low utilization during the decode stage on GPUs.**

This paper proposes GLITCHES, GPU-FPGA LLM inference through a collaborative heterogeneous system. In this paper, we analyze the different characteristics of GPUs and FPGAs and employ GPUs for the prefill stage and FPGAs for the decode stage, leveraging the strengths of GPUs and FPGAs. Based on HBM profiling results, we apply the data prefetching technique to further improve the off-chip memory bandwidth utilization during the decode computations on FPGAs. Experiments demonstrate that a GLITCHES heterogeneous LLM inference system with an A100 GPU and seven U280 FPGAs achieves a 1.28/1.34 times improvement in system throughput and a 2.38/1.90 times improvement in cost efficiency compared to a homogeneous system with 8-card A100/V100S GPUs.

***Index Terms*—Large language model, Heterogeneous system**

I. Introduction

Large language models (LLMs), such as ChatGPT, have gained significant attention over the past few years and have demonstrated strong capabilities across a variety of tasks [1]–[3]. In latency-sensitive scenarios, like real-time chatbots [4], [5] or code completion applications [6]–[8], it is crucial to minimize the inference latency to guarantee user experience [9]. Therefore, small batch inference, even with only one batch, is commonly used in these scenarios. Numerous inference frameworks [10]–[12] and hardware designs [13], [14] have emerged to accelerate LLM inference.

LLMs are based on the transformer architecture [15], and the inference process can be divided into two stages: prefill and decode. The prefill stage understands the input prompts and extracts the information, while the decode stage generates the response token by token. However, from the computational perspective, the prefill and decode stages are intrinsically different. The prefill stage processes all input tokens simultaneously. Thus, the computation mainly consists of matrix-matrix multiplication operations, with computing being the performance bottleneck. Meanwhile, the decode stage processes only one token at a time (with batch size one), and its computation mainly consists of matrix-vector multiplications, which leads to severe bandwidth bottlenecks.

The different characteristics of the prefill and decode stages lead to inefficiency in LLM inference on existing hardware platforms like GPUs or FPGAs, failing to fully exploit the performance of the hardware platforms. For GPUs, the matrix-matrix multiplication in the prefill stage can effectively utilize the high computing power and parallelism of modern GPUs. However, the decode stage is severely limited by off-chip memory bandwidth [16], resulting in extremely low utilization of the GPU computing power, with only 0.19% on the A100 GPU. At the same time, the power consumption of GPUs is usually high, with a typical power of 256.6 W for the LLM of inference on an NVIDIA A100 GPU. For FPGAs with specific hardware architectures, current FPGAs equipped with high-bandwidth memory (HBM) (e.g., Xilinx Alveo U280 and Versal VHK158) have been proven to have the potential to outperform GPUs in the decode stage with significant cost efficiency and energy efficiency [14]. However, due to the limited computational resources and frequency of FPGAs, FPGAs encounter computing bottlenecks in the prefill stage, resulting in long first token latency [13], [16]. For example, the latency to pre-fill 1536 tokens on the U280 FPGA is about 5 seconds, 28.44 times that of an A100 GPU. In addition, we found that the HBM bandwidth in current FPGA-based LLM accelerators can be additionally developed to further enhance the inference performance of the decode stage.

To address these challenges, we propose a heterogeneous system consisting of FPGAs and GPUs called GLITCHES to realize end-to-end LLM inference with higher performance compared to a homogeneous system. The key contributions of this paper are as follows:

- Leveraging the distinct characteristics of GPUs and FPGAs, we propose a heterogeneous LLM inference system that maps prefill computations to GPUs and decode computations to FPGAs, enhancing system throughput and cost efficiency by up to 1.34 and 2.39 times, respectively.
- Based on the HBM profiling results on FPGAs, we propose a data prefetching and memory access merging technique, improving the decode stage inference performance by up to 1.20 times on FPGAs.
- Based on the throughput of GPUs and FPGAs, we propose a scalable heterogeneous system solution for multiple GPUs and FPGAs within a single node, enabling data transmission during LLM inference.

*Both authors contributed equally to this work.

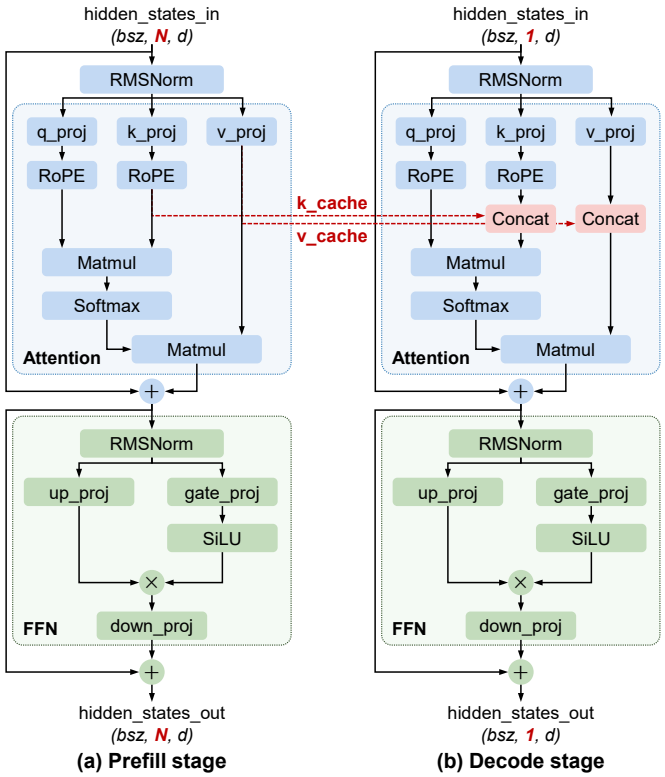

Fig. 1. Structure of a transformer block in the LLaMA series of LLMs during (a) the prefill stage and (b) the decode stage.

II. Background and Related Work

A. Background

Transformer-based LLMs. Popular LLMs today are commonly built on decoder-only transformer architecture [15], consisting of multiple transformer blocks. Input prompts are first converted to tokens and then embedded into input hidden states and fed to transformer blocks. The hidden states, after being processed sequentially by all the transformer blocks, go through a linear layer called lm_head to get the $logits$ of the next token. By sampling the distribution of $logits$, the index of the next token can be obtained.

Taking the popular LLaMA series of LLMs [17] as an example, the structure of each transformer block is shown in Fig. 1. Each transformer block contains a self-attention block and a feed-forward network (FFN). The self-attention block first normalizes the input hidden state and then obtains the query (Q), key (K), and value (V) by linear projection layers. The rotary position embedding (RoPE) is then applied to Q and K, then the self-attention computation is performed within each attention head, as shown in Eq. 1.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{1}$$

Notably, the attention computation grows quadratically with the input token length N, so the computation increases significantly when the input token is longer. The FFN consists of several linear projection layers, nonlinear activation functions, and element-wise computation to obtain the output hidden states.

As shown in Fig. 1(a), in the prefill stage, LLMs process all the input tokens, computing the corresponding K cache and V cache to obtain the $logits$ of the first generated token. At this point, the input shape of each transformer block is (bsz, N, d), where bsz stands for the batch size, N stands for the input token length, and d represents the hidden dimension size, so the computations in the prefill stage are matrix-matrix multiplications, where the matrix size is determined by N.

As shown in Fig. 1(b), the input shape of each transformer block is $(bsz, 1, d)$ since the token generated in the previous prefill or decode is used as an input during this round of decode. In each transformer block, the K and V of the current token are concatenated with the KV cache, and then self-attention computation is performed. Consequently, in the decode stage, we have $N = 1$, and computations are memory-bound matrix-vector multiplications.

B. Related Work

GPU-based Inference Engines. GPUs have now become the most popular hardware for LLM inference. Many GPU-based inference engines have emerged, such as FlashAttention [18], FlashDecoding [19], DeepSpeed [11], FlexGen [20], TensorRT-LLM [12], vLLM [10], and FlashDecoding++ [21]. These works improve the performance of LLM inference by optimizing computational graphs, attention and FFN kernels, etc., to fully utilize the hardware resources on GPUs. However, the decode stage is heavy memory bottlenecked in small batch scenarios, leading to low utilization and difficulty in leveraging the high computing power of modern GPUs.

FPGA-based Accelerators. FPGA-based LLM accelerators can be divided into single-FPGA based [14], [16] and multi-FPGA designs [13], [22]. These works design specialized hardware architectures for efficient LLM inference, focusing on accelerating the decode stage. Most of them design efficient matrix-vector multiplication architectures, utilizing the multi-channel HBMs on FPGAs for higher off-chip memory bandwidth as much as possible. However, due to the limited DSP resources on FPGAs, the peak computing power of current HBM-equipped FPGAs struggles to meet the computational demands of the prefill stage for long input scenarios (e.g., longer than 2k tokens), resulting in very long first token latency for FPGA-based LLM inference.

III. System Architecture of GLITCHES

A. Challenge

The prefill and decode stages of LLM inference exhibit distinctly different computational characteristics. Prefill computations are matrix-matrix multiplications with high data reuse, demanding massive computing power. Decode computations are matrix-vector multiplications, which have very poor data reuse and require high bandwidth and little computing power.

Existing single hardware platforms, whether GPUs or FPGAs, struggle to achieve high performance and high utilization across both the prefill and decode stages, leading to long latency or low efficiency. GPUs are highly underutilized during the decode stage due to the low computational demands. As

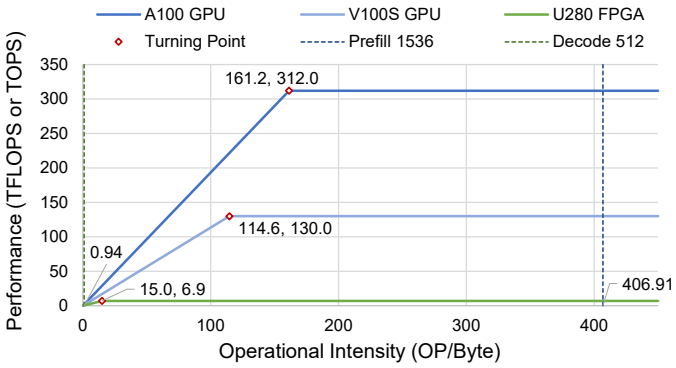

Fig. 2. The roofline model for processing prefill and decode stages by A100, V100S GPU and U280 FPGA, where the operational intensity of prefill and decode takes the values on GPU platforms.

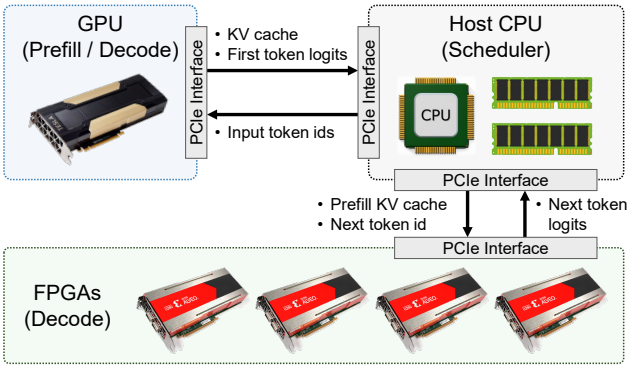

Fig. 3. The overall architecture of GLITCHES, including host CPU, GPU and multiple FPGAs

shown in Tab. I and Tab. II, when processing the prefill of 1536 tokens for LLaMA2-7B on an NVIDIA A100 GPU, the prefill latency is 175.85 ms. However, during the decode of 512 tokens, the average time for decode each token is 24.26 ms. Compared to the prefill stage, the operational intensity in the decode stage is reduced by 432.88 times, while the latency is only reduced by 7.25 times. This indicates that the GPU tensor core is significantly underutilized in the decode stage, at only 0.19%.

Although FPGAs show better performance than V100S in the decode stage, FPGAs are significantly slower than GPUs in the prefill stage due to the limited peak computing power. For example, the prefill latency of U280 FPGA is as long as 5 seconds, which is 28.44 times that of A100 and 12.54 times that of V100S. Such a long first token latency may lead to a degraded user experience.

Therefore, using only GPUs for LLM inference leads to significant computing power waste during the decode stage, while using only FPGAs results in excessive first token latency in the prefill stage, thus diminishing the user experience.

B. Insight

By leveraging the strengths of GPUs and FPGAs, it is expected to maximize throughput by assigning different computational tasks to different hardwares. Based on the data in Tab. I and Tab. II, we can analyze the three hardware platforms by the roofline model, as shown in Fig. 2. Regarding operational intensity, the prefill stage is far beyond the roofline turning point of all three hardware platforms, so GPUs with higher peak performance should be employed to reduce the prefill latency for a better user experience. The operational intensity of the decode stage is much lower than the roofline turning point of all the three hardware platforms, severely limited by the memory access, so that even with far inferior peak computing power, FPGAs can still achieve comparable performance to GPUs in decode computations. The roofline model indicates that GPUs are better suited for compute-intensive tasks, whereas FPGAs are relatively more efficient for memory-intensive tasks.

Therefore, by combining the preferences of both GPU and FPGA platforms, we can assign prefill computations to GPUs and decode computations to FPGAs, thereby achieving higher overall system throughput and efficiency.

C. Solution

To address these problems, we propose GLITCHES, a heterogeneous GPU-FPGA system designed to effectively support LLM inference, capable of interconnecting multiple GPUs and FPGAs through the PCIe interface. By handling prefill computations on GPUs, decode computations on FPGAs, and transferring the KV cache during prefill from GPUs to FPGAs, we achieve collaborative LLM inference between the GPUs and FPGAs.

The overall system architecture is shown in Fig. 3. The system comprises a host CPU, GPUs, and multiple FPGAs. Since each GPU and FPGA needs to perform the LLM inference process independently, the whole model weights need to be stored on each GPU and FPGA.

In GLITCHES, when the CPU receives a LLM inference request, it assigns the corresponding prefill computation to the GPU. The GPU processes the prefill stage inference using complete FP16 precision. Each self-attention layer during the prefill stage computes K and V matrices as KV cache. These FP16-format KV caches are first stored in the GPU's memory, then quantized and transferred back to the host CPU's memory via the PCIe interface. Once the host CPU receives the KV cache data, it forwards it directly to a specific address in the HBM of the specified FPGA via the PCIe interface. This HBM address is reserved during the offline FPGA instruction compilation, specifically for storing the KV cache generated during the prefill stage.

As shown in Fig. 4, the prefill computations for multiple transformer blocks on the GPU and KV cache transmissions occur in a pipeline, so most of the data transmission latency can be overlapped. Once the GPU finishes the prefill stage inference, the $logits$ results generated by the prefill stage are also sent back to the CPU memory, and then the first generated token index is sampled from $logits$. The CPU transmits the sampled token indexes to the FPGA where the KV cache data is located, and ensures that all of the KV cache data has been transferred to the HBM of this FPGA, then raises the `start` register and launches the decode computation of the FPGA.

TABLE I

THE COMPUTATION, MEMORY ACCESS, POWER, AND LATENCY OF LLAMA2-7B INFERENCE ON GPUS AND FPGAS WITH PREFILL AND DECODE TOKEN LENGTHS OF [1536, 512], RESPECTIVELY. THE GPU IMPLEMENTATION IS BASED ON HUGGINGFACE (FP16), WHILE THE FPGA IMPLEMENTATION USES THE FLIGHTLLM [14] (APPROXIMATELY W4A8)

	Prefill (1536 tokens)					Decode (512 tokens, per token on average)				
Hardware Platform	Computation (GFLOPs or GOPs)	Memory Access (GB)	Operational Intensity (OP/Byte)	Power (W)	Latency (ms)	Computation (GFLOPs or GOPs)	Memory Access (GB)	Operational Intensity (OP/Byte)	Power (W)	Latency (ms)
A100 GPU	21137.01	48.38	406.91	256.6	175.85	14.16	14.08	0.94	167.3	24.26
V100S GPU	21137.01	48.38	406.91	239.8	398.80	14.16	14.08	0.94	222.5	29.52
U280 FPGA	21137.01	21.29	924.47	46.0	5001.20	14.16	3.96	3.33	46.0	21.50

TABLE II

THE BANDWIDTH UTILIZATION, COMPUTING UNIT UTILIZATION, ENERGY EFFICIENCY, AND COST EFFICIENCY OF LLAMA2-7B INFERENCE ON GPUS AND FPGAS WITH PREFILL AND DECODE TOKEN LENGTHS OF [1536, 512], RESPECTIVELY

	Parameters			Prefill (1536 tokens)				Decode (512 tokens, per token on average)			
Hardware Platform	Peak Perf. (TFLOPS or TOPS)	Band-width (GB/s)	Cost ($)	Band-width Util.	Compute Util.	Energy Efficiency (token/s/W)	Cost Efficiency (token/s/$)	Band-width Util.	Compute Util.	Energy Efficiency (token/s/W)	Cost Efficiency (token/s/$)
A100 GPU	312[1]	1935	17000	14.22%	38.52%	2.22E-02	3.35E-04	29.99%	0.19%	2.46E-01	2.42E-03
V100S GPU	130[1]	1134	12000	10.70%	40.77%	1.05E-02	2.09E-04	42.06%	0.37%	1.52E-01	2.82E-03
U280 FPGA	6.91[2]	460	8000	0.93%	61.15%	4.35E-03	2.50E-05	40.08%	9.53%	1.01E+00	5.81E-03

[1] Peak computing power of tensor core in GPUs.
[2] The peak INT8 computing power of the U280 FPGA is 24.5 TOPS, but the peak computing power of FlightLLM is only 6.91 TOPS.

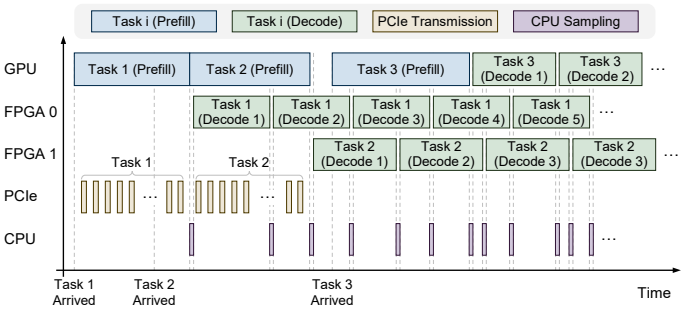

Fig. 4. An example of the timeline of a GLITCHES system running with a GPU and two FPGAs. The scheduler distributes the decode tasks to the idle FPGAs, and the GPU will handle the decode tasks when all FPGAs are busy.

Upon receiving the `start` signal, the FPGA initiates the decode stage inference based on the KV cache computed by the GPU. The final *logits* output from each decode iteration is transmitted back to the host CPU via PCIe, where the next token index is sampled and sent back to the FPGA for further decode processing until a special stop token is encountered or the maximum token length is reached. Once the FPGA starts to process the decode stage, no further data exchange between the FPGA and GPU occurs, so the GPU does not need to retain a copy of the KV cache and can release the corresponding memory for other computations.

The number of cards for FPGAs and GPUs is determined by the ratio of prefill and decode throughput requirements. Since the decode latency usually accounts for most of the end-to-end inference, a single GPU for prefill computation can typically match the multiple FPGAs for decode computations. To avoid leaving the GPU idle, if all other FPGAs are busy, the GPU can act as an FPGA substitute to assist the decode computation if no other prefill tasks are available.

We designed a scheduler on the host side to manage the resources of GPUs and FPGAs for multiple inference requests. The scheduler distributes decode tasks to different FPGAs based on a first-come-first-served strategy. When all FPGAs are busy, new decode tasks will be processed on the GPU. In this case, the KV cache data computed by the GPU in the prefill stage does not need to be transferred to the host CPU but remains in the GPU memory for subsequent decode computations. If the incoming inference request exceeds the GPU memory, the task will not be executed immediately; instead, it will enter a task queue, waiting for the GPU to finish the previous tasks and free enough memory.

Additionally, GLITCHES supports further scale-up. There are typically Ethernet interfaces on FPGAsthat can connect to external Ethernet cables. This allows KV cache data to be transferred over the network between multiple nodes, enabling multi-node scaling and larger interconnections. For heterogeneous multi-GPU and multi-FPGA systems (e.g., different models of GPUs or FPGAs), a more complex and accurate scheduler can be trained to achieve resource scheduling for the heterogeneous GPUs and FPGAs. However, this will be explored in future work and will not be discussed in detail in this paper.

IV. DATA PREFETCHING FOR FPGAS

GLITCHES applies FlightLLM as the baseline design for decode computation on FPGAs. Although FlightLLM leverages the HBM on U280 FPGA to provide high bandwidth for the decode stage, the HBM bandwidth utilization could still be further improved to get better decode performance.

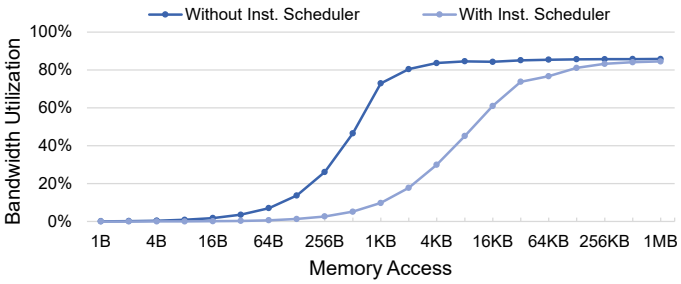

Fig. 5. HBM bandwidth utilization with different amount of memory data access on U280 FPGA with/without instruction scheduler.

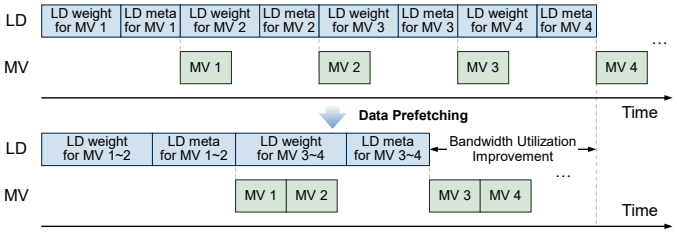

Fig. 6. Data prefetching merges multiple fine-grained memory access instructions into a single instruction with larger memory access volume, improving bandwidth utilization and reducing memory access time.

A. Challenge

FlightLLM is an instruction-driven FPGA-based accelerator, accessing HBM through load (`LD`) and store (`ST`) instructions, sending read and write requests to the datamover. Therefore, the memory access performance depends not only on the HBM itself but also on the overhead of the instruction scheduler inside the accelerator. For instance, if a large number of `LD` or `ST` instructions with small memory access volume are frequently sent, the instruction decode and issue latency may become the performance bottleneck, preventing timely read/write requests to the HBM, resulting in worse memory access performance than expected. Consequently, existing direct modeling for HBM on FPGAs (Shuhai [23], etc.) cannot accurately model the instruction decode and issue latency and the instruction pipeline in the instruction scheduler. In scenarios with a large number of fine-grained memory access instructions, this may lead to large deviations in memory access performance evaluation.

FlightLLM reduces the volume of model parameters by grouped mixed-precision quantization, thus enhancing the inference performance. However, grouped mixed-precision quantization introduces additional quantization parameters, such as zero points and scaling factors. Although these quantization meta-data are relatively small compared to the weights (about 4$\sim$6% of the weights), they need to be stored in different on-chip buffers from the weights in the FPGA accelerator. This leads to a large number of fine-grained `LD` instructions to load both weights and quantization meta-data from HBM to the on-chip buffers, resulting in low HBM bandwidth utilization, only around 40%.

B. Insight

To provide an accurate performance evaluation of the HBM and optimize the HBM bandwidth, we develop a profiler for instruction-based accelerators. The profiler generates a series of memory access instructions and measures the latency, obtaining actual HBM performance with the instruction scheduler overhead. As shown in Fig. 5, the profiling results demonstrate that the additional overhead for instruction decode and issue in the instruction scheduler considerably degrades the performance for small volume memory access instructions. Therefore, merging small data memory access requests into larger requests (e.g., >32KB) can significantly improve bandwidth utilization.

C. Solution

To minimize the fine-grained memory access requests, GLITCHES applies the data prefetching technique to improve HBM bandwidth utilization. As shown in Fig. 6, we prefetch the weights and quantization meta-data required for the following multiple matrix-vector multiplication (`MV`) instructions in advance, thereby merging the fragmented memory access requests. Although prefetching does not reduce the overall memory access volume, it reduces the number of memory access instructions and increases the memory access volume per instruction. Therefore, data prefetching not only lowers the extra overhead of instruction decode and issue in the instruction scheduler, but also helps HBM provide higher bandwidth by sending coarse-grained memory access requests.

Although prefetching can improve memory bandwidth utilization, the prefetched data size should not be excessively large or small. When the prefetched data size is too large, it is challenging to pipeline between memory access and computations to overlap the latency. When the prefetched data size is too small, no substantial bandwidth utilization improvement can be achieved. GLITCHES utilizes an accurate performance simulator (with an error <5%) to find the optimal data prefetching ratio for each layer in the network to achieve the best end-to-end performance.

V. Evaluations

A. Evaluation Setup

In GLITCHES, the GPUs used are the NVIDIA V100S and A100, which are commonly used for LLM inference, and the FPGAs used are the Xilinx Alveo U280 with HBM. The peak performance, bandwidth, and price of these hardware platforms are shown in Tab. II. The GPUs are running the LLaMA2-7B model from HuggingFace at FP16 accuracy, while the FPGAs are running a quantized LLaMA2-7B (about 4-bit weights and 8-bit activations, similar to FlightLLM).

In the data transmission latency experiment, we repeatedly measure the transmission latency of the KV cache 50 times to derive the average transmission latency and its standard deviation. The GPU to FPGA latency is obtained by adding the latency from GPU to CPU and from CPU to FPGA. The performance of the FPGA comes from a cycle-accurate simulator based on FlightLLM running at 225MHz, with data prefetching technique applied.

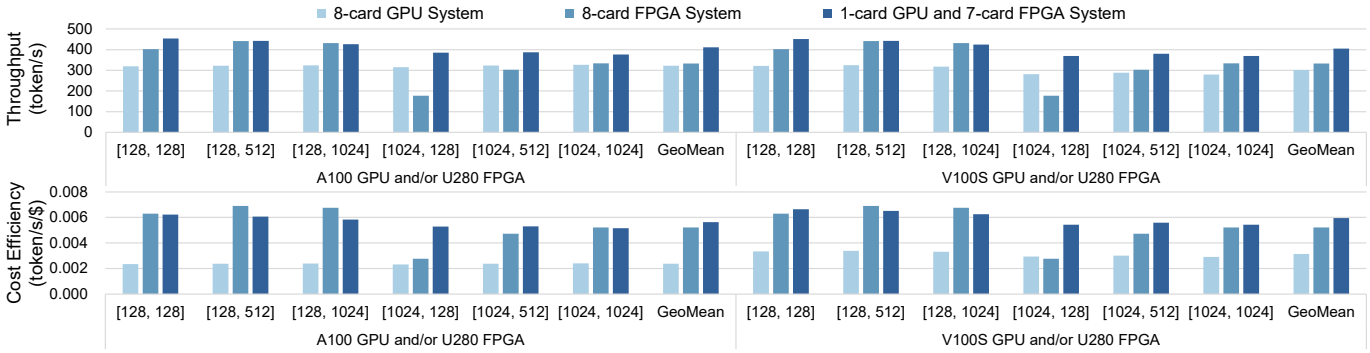

Fig. 7. System throughput (token/s) and cost efficiency (token/s/$) of all-GPU, all-FPGA, and GPU-FPGA collaborative heterogeneous system in GLITCHES under different number of input and output tokens. (e.g., [128, 512] stands for prefill 128 tokens and decode 512 tokens)

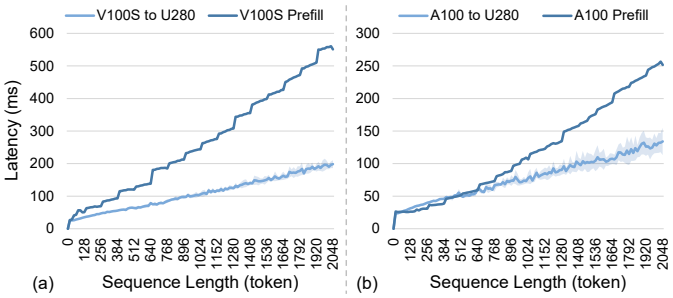

Fig. 8. For (a) V100S and (b) A100 GPU, the KV cache transmission latency from GPU memory to HBM on U280 FPGA and the prefill latency for different sequence lengths, where the light-colored area represents the standard deviation of the data transmission latency.

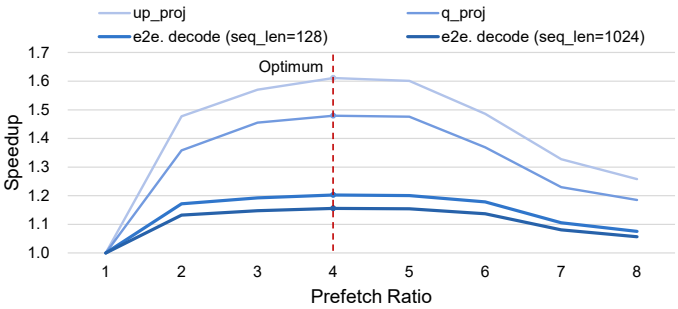

Fig. 9. The speedup of up_proj, q_proj linear layer and end-to-end decode in LLaMA2-7B with different prefetch ratios. The optimal performance is achieved when the prefetch ratio is four.

B. Evaluation Results

Data transmission latency. The volume of the KV cache grows linearly with sequence length N in LLMs. To evaluate the performance of GLITCHES, we measure the latency of KV cache data transmission from GPU memory to the HBM on FPGAs. As shown in Fig. 8, in most cases (especially with longer input tokens), even considering the standard deviation of the data transmission latency fluctuations, the data transmission latency is less than the GPU prefill latency. The reason is that the prefill computation grows quadratically with sequence length N. Since the data transmission is spread out over the computation of each transformer block, the data transmission latency can be easily overlapped by the prefill computation, introducing little additional latency.

Data prefetching. Based on FlightLLM, we evaluate the performance improvement brought by data prefetching on U280 FPGA, and the results are shown in Fig. 9. The prefetch ratio M indicates that each memory access instruction provides the required data for the subsequent M computational instructions. The performance is optimal when the prefetch ratio is four. For example, in the q_proj linear layer, the single memory access volume of weight data increased from 8KB to 32KB, and the single memory access volume of quantization metadata increased from 256B to 1KB. Therefore, data prefetching improved the end-to-end decode performance by 1.20 times and 1.16 times when the sequence length is 128 and 1024, respectively. Prefetching accelerates smaller sequence lengths more significantly because the memory access volume of the attention computation is smaller in these cases, while prefetching greatly increases bandwidth utilization.

System throughput and Cost Efficiency. We simulate a single-node 8-card heterogeneous system for GLITCHES based on A100 and V100S GPU. We compare GLITCHES to a GPU-only and FPGA-only homogeneous system and measure the system throughput and cost efficiency (throughput per dollar) under different prefill and decode token lengths. As shown in Fig. 7, with an A100 GPU and seven U280 FPGAs, GLITCHES improves the average system throughput by 1.28/1.23× and cost efficiency by 2.38/1.08× compared to an 8-card A100/U280-only system. In comparison, with a V100S GPU and seven U280 FPGAs, GLITCHES achieves an average throughput of 1.34/1.21× and cost efficiency of 1.90/1.14× compared to an 8-card V100S/U280-only system, effectively improving the system throughput and cost efficiency.

VI. Conclusions

In this paper, we propose GLITCHES, GPU-FPGA LLM inference through a collaborative heterogeneous system. To address the challenges of long prefill latency on FPGAs and low utilization in the decode stage on GPUs, GLITCHES employs a heterogeneous system using GPUs for the prefill stage and FPGAs for the decode stage. GLITCHES also introduces the data prefetching technique to improve the bandwidth utilization on FPGAs further. Experiments show that the GLITCHES heterogeneous system with a GPU and seven FPGAs improves system throughput by 1.28/1.34× and cost efficiency by 2.38/1.90× compared to an 8-card A100/V100S homogeneous GPU system.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (No. 62325405, 62104128, U21B2031, 62204164), the National Key R&D Program of China (2023YFB4502200), Tsinghua EE Xilinx AI Research Fund, Tsinghua-Meituan Joint Institute for Digital Life, and Beijing National Research Center for Information Science and Technology (BNRist).

REFERENCES

[1] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[2] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, "Large language models in medicine," *Nature medicine*, vol. 29, no. 8, pp. 1930–1940, 2023.

[3] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, pp. 1–32, 2024.

[4] S. K. Dam, C. S. Hong, Y. Qiao, and C. Zhang, "A complete survey on llm-based ai chatbots," *arXiv preprint arXiv:2406.16937*, 2024.

[5] S. Chen, M. Wu, K. Q. Zhu, K. Lan, Z. Zhang, and L. Cui, "Llm-empowered chatbots for psychiatrist and patient simulation: application and evaluation," *arXiv preprint arXiv:2305.13614*, 2023.

[6] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[7] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an llm to help with code understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[8] D. Guo, C. Xu, N. Duan, J. Yin, and J. McAuley, "Longcoder: A long-range pre-trained language model for code completion," in *International Conference on Machine Learning*. PMLR, 2023, pp. 12 098–12 107.

[9] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Barnes, and A. Mian, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.

[10] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[11] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[12] N. Vaidya, F. Oh, and N. Comly, "Optimizing inference on large language models with nvidia tensorrt-llm, now publicly available.[online], 2023."

[13] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 616–630.

[14] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, "Flightllm: Efficient large language model inference with a complete mapping flow on fpga," *arXiv preprint arXiv:2401.03868*, 2024.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[16] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.

[17] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[18] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[19] T. Dao, D. Haziza, F. Massa, and G. Sizov, "Flash-decoding for long-context inference," 2023.

[20] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[21] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, H. Dong, and Y. Wang, "Flashdecoding++: Faster large language model inference on gpus," *arXiv preprint arXiv:2311.01282*, 2023.

[22] Y. Gao, J. C. Vega, and P. Chow, "The feasibility of implementing large-scale transformers on multi-fpga platforms," *arXiv preprint arXiv:2404.16158*, 2024.

[23] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on fpgas," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 111–119.