MAXIMIZE GPU EFFICIENCY WITH AMD EPYC™ HIGH-FREQUENCY PROCESSORS

WHITEPAPER | 2025

In the world of AI and machine learning, running training and inference workloads at scale requires significant computational resources. For many workloads GPUs have become essential in the development and deployment of large AI models. Data center operators of all sizes and scale need to ensure that their investments in GPUs are returning maximum value. Increasing the utilization and efficiency of GPUs should be an essential priority for anyone using or supporting GPU based AI workloads.

While GPUs have been top of mind for AI applications handling the heavy lifting for training and inference, CPUs play an essential role in orchestrating the overall workload and have significant impact over the efficiency of GPUs. Host CPUs perform crucial tasks like pre- and post-processing of data and managing data movement. Inference requests using a Large Language Model (LLM) are typically executed by a GPU-based system containing a host CPU that manages incoming requests or model queries, also known as prompts. Some of the most common examples of applications using LLM inference include sentiment analysis, language translation, content creation, summarization, and question-answer chatbots. Each of these applications have different structures of their prompt and response and coordinating the processing of each request to reduce idle time on the GPUs is the core function of a host CPU.

In this blog, we will explore how the host CPU in a GPU-based system can significantly impact the overall performance and cost-efficiency of LLM inference. We will focus on the key functions of the CPU during inference, show how host CPU performance can reduce end-to-end latency, and quantify the benefits up to 24% and an average improved latency of 9% with AMD Instinct MI300 and 8% with Nvidia H100 respectively of using AMD EPYC™ high-frequency CPUs as the host processor.

Host CPU Involvement in LLM Inference

In the world of AI and machine learning, running training and inference workloads at scale requires significant computational resources. For many workloads GPUs have become essential in the development and deployment of large AI models. Data center operators of all sizes and scale need to ensure that their investments in GPUs are returning maximum value. Increasing the utilization and efficiency of GPUs should be an essential priority for anyone using or supporting GPU based AI workloads.

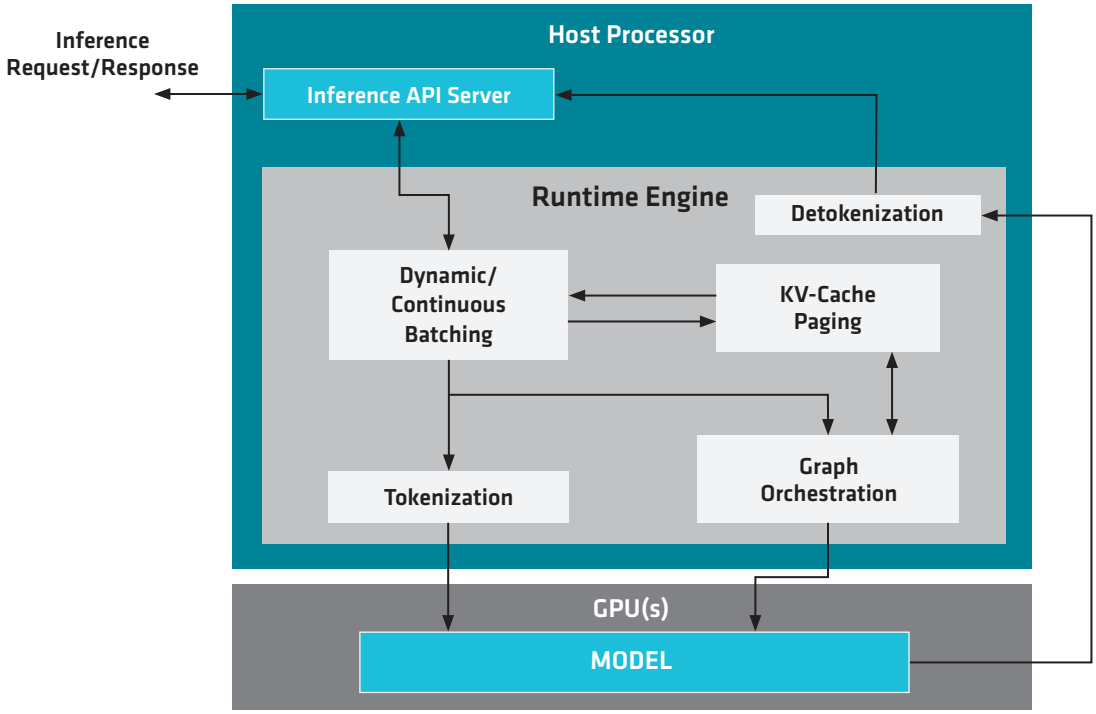

Figure 1: Generalized Host CPU Fundamental Functional Block

Figure 1 illustrates the fundamental flow of an inference request in a GPU-based system with the model (shown in blue) running on the GPU and the other functions running on the host CPU. The following describes the fundamental and optional functions that are employed in providing an inference service.

Inference API Server:
The Inference API server processes an incoming request and forwards it to the Runtime Engine. The Runtime Engine generates a response on completion that is sent back to the requester. The actions of the Inference API can largely be decoupled from the Runtime Engine using queues at the interface between the two. This is an especially important function when there are multiple concurrent users to service, ie multiple prompts, multiple concurrent models, and or multiple GPUs operating with a shared host CPU. The API ensures that the response to a prompt is returned to the appropriate requester.

Runtime Engine:
The Runtime Engine is represented by the functions in the orange box in Figure 1. The Runtime Engine within the CPU performs critical resource management functions, such as dynamic batching and K-V cache paging, to ensure that the GPU's compute efficiency and memory usage are optimized. It must also manage orchestration tasks such as kernel-launch and synchronization across multiple GPUs. These tasks can be on the critical path and directly impact the end-to-end latency of inference requests. Additionally, as model architectures evolve to include elements like data dependent control flow, there will be more pressure on the CPU response time.

Draft Model Execution (optional):
For certain applications, a draft model may be used to speculate in advance of running full inference and make early predictions.These early predictions allow the primary LLM to be run with higher batch sizes, effectively reducing the number of token generation steps on that model. In this mode of operation, the draft model can be executed on the host CPU to allow the GPU compute and memory resources to be dedicated to the primary LLM.

Pre-Processing:
For some applications, pre-processing involves running smaller models like Sentence BERT to generate embeddings for retrieval systems, known as "prompt engineering." Once pre-processed, the prompts within a batch are tokenized and prepared for execution on the GPU(s) during the main inference phase. The execution of the embedding model and the subsequent vector database similarity search by the retriever could be on the inference-critical path. The overhead of this action can benefit from a high-performance host CPU.

Post-Processing:
Once the GPU completes the model execution, the CPU finalizes the response by managing token sampling and performing other output processing tasks such as formatting, error handling, or visualization of the response data to present the user.

ML Ops:
In a production environment where an inference request makes a choice between multiple models (See https://arxiv.org/pdf/2405.07518), model loading time is especially important. A CPU with good IPC, memory and IO bandwidth can significantly improve the performance of model loading.

Each of these functions that the host CPU performs has the potential to materially impact the efficiency of the GPU(s) performing inference and therefore the total response time of an inference request.

The Role of High-Frequency CPUs

While some of the actions listed in the previous section can run concurrently with GPU inference execution, others – such as kernel launch, tokenization, dynamic batching, draft model execution, data dependent control flow, synchronization, and more etc. – can reside in the latency-critical path. When performed in the latency critical path, the efficiency and speed at which these functions perform their task can become an important factor in the total inference response time. These host CPU actions become more critical when inference runtimes are latency constrained.

Our previous internal studies have included extensive measurements of host CPU activity for inference and training workloads across Nvidia H100 and AMD Instinct™ MI300 based systems. We have collected these profiles running TRT-LLM (H100 system) and vLLM for inference. For training, we collected profiles running JAX, Pytorch, and Megatron-LM frameworks.

These profiles have informed us that single-thread performance of the host CPU is more critical than throughput to mitigate potential overhead from host CPU activity.

As such, AMD offers a series of high frequency parts with core counts ranging from 16 to 64 enabling users to effectively address host processor requirements while providing control over system specifications and costs. For large GPU systems, AMD and AI ecosystem partners recommend a system based on AMD EPYC™ 9575F with 64 cores which has a TDP range of 320-400 Watts and a maximum core frequency (Fmax) of 5GHz to help improve overall end-to-end inference performance.

Host Sensitivity - Test Setup and Results

To demonstrate the performance impact of the host CPU on the overall system performance, which we refer to as "host sensitivity" we conducted a study. We lay out the details of the experiment performed to highlight the benefits of AMD EPYC™ 9575F as a host CPU in a GPU based system. Our study focused primarily on measuring end-to-end latency of inference using prompt length and output length combinations that are representative of chatbot, content-creation, summarization and translation inference tasks. We chose batch sizes of 32 and 1024 as representatives for online and offline inference, respectively.

We did not implement an inference server, a RAG pipeline, multi-model COE inference or a draft model for speculation in this study. This study was performed using FP8 checkpoints of popular open-source models for inference.

TABLE 1: HOST SENSITIVITY EXPERIMENT SETUP

RUNTIME: VLLM :0.7	MODELS:	HOST CPU:	SYSTEMS:

• Continuous batching enabled
• Num-scheduler-steps = 1
• (prompt length, output tokens)
  • Chatbot = (128,128)
  • Content Creation = (128,1024/2048)
  • Summarization = (1024/2048, 128)
  • Translation = (1024/2048, 1024/2048)
• Batch Sizes = [32, 1024]
• Tokenization = offline
• Detokenization = online
	• Llama3.1-70B-Instruct (FP8) ; Parallelism (Tensor) = 8
	• Llama3.1-8B-Instruct (FP8); Parallelism (Tensor) = 1
	• Mixtral 8x7B-Instruct (FP8); Parallelism (Tensor) = 8
		• AMD EPYC™ 9575F with a TDP of 320-400 W; Fmax=5 Ghz
		• Intel Xeon 8592+ with a TDP of 350 W; Fmax=3.9 Ghz
			• 8x AMD Instinct™ MI300 - Device 74a1-XGMI-192GB-750W; ROCm™ 6.3.0-39
			  • Host OS: 9575F – Ubuntu 24.04 LTS; 8592+ 24.04.1 LTS
			• 8x H100 NVIDIA H100-80GB-HBM3-700W; Cuda version 12.6
			  • Host OS: 9575F- Ubuntu 22.04.4 LTS; 8592+ - Ubuntu 22.04.5 LTS

Host Sensitivity - Results

The following tables show the performance advantage of a EPYC 9575F host CPU over Xeon 8592+ in 8x AMD Instinct™ MI300x based and 8x Nvidia H100 based GPU systems on a variety of inference tasks. For every test case, we performed three runs and collected measurements for each run. The median of those measurements is reported in the following tables.

TABLE 2: HOST COMPARISON: 8x AMD INSTINCT™ MI300x GPU BASED SYSTEM

MODEL	TASK	BATCH SIZE	INPUT TOKENS	OUTPUT TOKENS	LATENCY IMPROVEMENT AMD EPYC™ 9575F/XEON 8592+
Llama-3.1-8B-Instruct-FP8	Chatbot	32	128	128	1.06x
		1024	128	128	1.05x
	Content Creation	32	128	1024	1.07x
		1024	128	1024	1.03x
	Summarization	32	1024	128	1.05x
		1024	1024	128	1.03x
	Translation	32	128	1024	1.05x
		1024	1024	1024	1.03x

TABLE 2 CONT-D: HOST COMPARISON: 8x AMD INSTINCT™ MI300x GPU BASED SYSTEM					
MODEL	TASK	BATCH SIZE	INPUT TOKENS	OUTPUT TOKENS	LATENCY IMPROVEMENT AMD EPYC™ 9575F/XEON 8592+
Llama-3.1-70B-Instruct-FP8	Chatbot	32	128	128	1.13
		1024	128	128	1.08
	Content Creation	32	128	1024	1.10
		1024	128	1024	1.05
	Summarization	32	1024	128	1.08
		1024	1024	128	1.03
	Translation	32	1024	1024	1.10
		1024	1024	1024	1.05
Mixtral 8x7B-Instruct-FP8	Chatbot	32	128	128	1.19
		1024	128	128	1.15
	Content Creation	32	128	1024	1.13
		1024	128	1024	1.16
	Summarization	32	1024	128	1.24
		1024	1024	128	1.11
	Translation	32	1024	1024	1.11
		1024	1024	1024	1.15

TABLE 3: HOST COMPARISON 8X NVIDIA H100 GPU BASED SYSTEM					
MODEL	TASK	BATCH SIZE	INPUT TOKENS	OUTPUT TOKENS	LATENCY IMPROVEMENT AMD EPYC™ 9575F/XEON 8592+
Llama-3.1-8B-Instruct-FP8	Chatbot	32	128	128	1.12x
		1024	128	128	1.18x
	Content Creation	32	128	2048	1.09x
		1024	128	2048	1.09x
	Summarization	32	2048	128	1.05x
		1024	2048	128	1.04x
	Translation	32	2048	2048	1.05x
		1024	2048	2048	1.04x
Llama-3.1-70B-Instruct-FP8	Chatbot	32	128	128	1.04x
		1024	128	128	1.11x
	Content Creation	32	128	2048	1.04x
		1024	128	2048	1.09x
	Summarization	32	2048	128	1.01x
		1024	2048	128	1.03x
	Translation	32	2048	2048	1.02x
		1024	2048	2048	1.05x
Mixtral 8x7B-Instruct-FP8	Chatbot	32	128	128	1.09x
		1024	128	128	1.20x
	Content Creation	32	128	2048	1.10x
		1024	128	2048	1.19x
	Summarization	32	2048	128	1.06x
		1024	2048	128	1.05x
	Translation	32	2048	2048	1.08x
		1024	2048	2048	1.13x

CONCLUSION

The above tables clearly show that inference latency is impacted by the choice of host CPU. The higher frequency and exceptional single-thread performance of AMD EPYC™ 9575F helps reduce the CPU overhead associated with performing an offline or online inference task.

Across the three representative models and the four inference tasks in our experiment, AMD EPYC™ 9575F provides an average performance improvement of 9% on the 8x AMD Instinct™ MI300 and 8% on the 8x Nvidia H100 GPU based systems. For the chatbot use case, a significant portion of the end-to-end latency is spent in setting up batching and memory management of the GPU device. For the cases involving many token generation steps such as content-creation and translation, detokenization was observed to be significant contributor to overall latency.

Our future work is projected to include a detailed breakdown of the host activity across these inference cases. We will also study the host sensitivity for an inference serving workload with latency constraints.

AUTHORS:

Ram Sivaramakrishnan: Fellow Systems Design Eng. AI Architect Server Solutions.
ram.sivaramakrishnan@amd.com

Matt Ouellette: Director Product Development Eng. AIG-AI Product Mgt.
matt.ouelette@amd.com

Ajith Sirra: MTS Product Application Eng. AIG-AI Product Mgmt.
ajith.sirra@amd.com

Shubin Zhao: SMTS Software Systems Design Eng. DCGPU Perf Eng.
shubin.zhao@amd.com

Danyang Zhang: SMTS Software Development Eng. AIG-AI Product Mgt.
danyang.zhang@amd.com

Jeremy Arnold: PMTS Software System Design Eng. DCGPU Perf Eng.
jeremy.arnold@amd.com

Mary Cirino: MTS Software Development Eng. DCGPU Perf Eng.
mary.cirino@amd.com