Understanding Performance Implications of LLM Inference on CPUs

Seonjin Na[1] Geonhwa Jeong[1] Byung Hoon Ahn[2] Jeffrey Young[1] Tushar Krishna[1] Hyesoon Kim[1]

[1]*Georgia Institute of Technology* [2]*University of California, San Diego*

***Abstract*—The remarkable performance of LLMs has led to their application in a wide range of fields, with data centers utilizing expensive accelerators such as GPUs and TPUs to support LLM inference and training. However, these costly accelerators face challenges with memory capacity due to the large size of LLMs and Key-Value (KV) cache during inference. To address memory capacity issues of accelerators such as GPUs/TPUs, offloading-based LLM inference has been proposed to store model weights, activations, and KV cache in CPU memory. This approach, however, often incurs significant performance degradation in LLM inference in terms of latency and throughput as the offloaded data must be transferred back and forth over the PCIe bus, which has a lower bandwidth compared to memory.**

This study explores new opportunities for leveraging CPUs in LLM inference. Recent CPUs are equipped with dedicated accelerators for efficient matrix computations and have extended ISAs to support training and inference of new AI models. They support larger memory sizes than most GPUs, allowing for the direct computation of large models and KV caches without offloading. Additionally, recent CPUs are often equipped with DDR and HBM memory, which provides options for optimizing for either memory capacity or bandwidth. This study provides a detailed analysis of LLM inference performance on the latest CPUs equipped with these advanced features. Based on our experimental results, we propose potential optimization strategies tailored to enhance the performance of LLM inference on CPUs.

***Index Terms*—Large Language Model (LLM), Offloading-based LLM Inference, LLM Inference on CPU, Intel AMX**

I. Introduction

Transformer-based Large Language Models (LLMs) demonstrate exceptional performance on a wide range of tasks and have initiated a new era in the field of generative AI. These advanced models are now broadly applied in numerous areas, including text generation [5], [50], [57], machine translation [4], etc. To meet the substantial compute demands of LLMs, companies are introducing specialized units in their existing processors, such as NVIDIA's Tensor Cores [33] in GPUs, Intel's TMUL unit [38] in CPUs, or even custom offload accelerators tailored for their LLMs, such as Meta's MTIA [13] or Google's TPU [24]. Many modern data centers are increasingly equipped with these accelerators to execute LLM training and inference efficiently [17].

Despite these advancements, serving LLMs remains expensive, primarily due to their high memory requirements [28], [45], [49]. The remarkable performance of LLMs is driven by their large model sizes, as dictated by scaling laws, enabling them to understand complex and lengthy contexts.

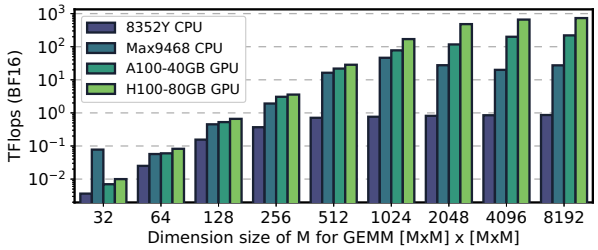


Fig. 1: General Matrix Multiplication (GEMM) throughput comparison across CPUs and GPUs with varying matrix dimensions.

For instance, one of the most recent LLMs, such as OPT-175B [57], requires 350GB of memory to load the weights with the FP16 data type, or the equivalent of at least five NVIDIA 80GB H100 GPUs. Furthermore, models used in industry, such as ChatGPT-4 [1] are known to be even larger [1], [7]. In fact, a recent report from Google suggests that scaling the sequence length to more than a million tokens may unlock even more in-context learning possibilities [47].

In addition to large model sizes, a commonly used optimization technique in LLM inference, KV caching, presents significant memory challenges. KV caching enhances inference performance by storing key and value vectors of previously generated tokens, which avoids repetitive computations during autoregressive generation. However, the size of the KV cache increases linearly with both sequence length and batch size. Recent studies have not only focused on supporting longer sequence lengths to enable models to understand and generate more complex outputs [12], [30], but have also explored increasing batch sizes to efficiently handle multiple user requests and maximize hardware utilization [28], [45], [56]. As a result, the memory demands for KV caching are becoming increasingly problematic. For example, OPT-66B [57] with a sequence length of 4096 and a batch size of 32 requires 288GB of memory for KV caching. This growing memory requirement for KV caching poses a significant challenge for LLM inference systems.

To address the growing memory requirements of LLMs and KV caching that exceed GPU capacity, prior studies have proposed *offloading-based LLM inference* serving techniques [45], [49], [58]. In offloading-based LLM inference serving systems, weights, activations, and KV caches are stored in the larger CPU memory and loaded from it during computation. Although offloading-based systems enable executing LLM inference with a limited GPU memory capacity, they introduce

new performance problems. These systems need to transfer offloaded model weights, activations, and KV caches from CPU memory to the GPU on demand via the slow PCIe bus during LLM inference, leading to significant performance degradation as shown in several previous studies [37], [49].

To circumvent these issues, this study explores the new opportunity of leveraging CPUs as compute units for LLM inference based on the following insights. First, CPU vendors are incorporating *dedicated matrix multiplication accelerators* along with the necessary ISA support such as Intel Advanced Matrix Extensions (AMX) [38]. Figure 1 shows the general matrix multiplication (GEMM) throughput comparison results across different matrix dimensions on 4th generation Intel CPUs (Max 9468) that support Intel AMX, 3rd generation CPUs (8352Y) that do not, and GPUs (A100 and H100). The graph shows that although the overall throughput is still lower compared to GPUs due to their specialized hardware and instruction capabilities, the CPU with dedicated matrix units (MAX 9468) has significant potential when considering the hardware cost[1].

Additionally, CPU memory typically offers greater capacity than GPU memory and can be further expanded using memory extension technologies such as CXL [34], unlike GPUs with fixed memory sizes. Moreover, some of the latest Intel CPUs are equipped with both DDR memory and HBM memory, enabling CPUs to exploit larger memory capacity and higher memory bandwidth. In order to address the memory capacity challenges posed by large model sizes and KV caches in LLM inference, the large memory capacity of CPUs and the high bandwidth of HBM provides an opportunity to exploit CPUs for LLM inference.

The contributions of this paper are as follows:

- The identification of challenges for LLM inference and opportunities for acceleration using new CPU-based platforms.
- The first extensive performance characterization of LLM inference on the latest Intel CPUs.
- A comprehensive performance comparison of the latest CPUs with state-of-the-art GPUs for LLM inference with various LLMs and configurations.
- Discussion of potential optimizations to better support LLM inference on CPUs based on characterization insights across platforms.

II. Background

A. Large Language Model Architecture

Recent large language models (LLMs) [5], [36], [50], [57] are *decoder-only transformer models* trained to generate subsequent tokens for a given input sequence in an auto-regressive manner. As shown in Figure 2, the decoder-only transformer model includes multiple decoder blocks, and each of these decoder blocks comprises several components that work together to process the input data. The *Multi-Head Attention* is a critical layer that captures the relationships among the input tokens through the attention mechanism. Each head within the multi-head attention focuses on various aspects of the input sequence, enabling the model to better grasp the context and dependencies between tokens. The *Feed-Forward Network (FFN)* after the self-attention layer applies non-linear transformations to refine the sequence representations. The FFN consists of two linear transformations separated by a non-linear activation function which helps capture complex patterns in the data. The outputs from these subsequent transformer layers are then projected through a linear layer and a softmax function to generate the final token predictions.

[1]Although calculating the exact cost for each machine is not straightforward due to the various factors affecting the overall cost, using the listing price of each processor as a proxy shows that Intel MAX 9468 [21] is 3x cheaper than NVIDIA H100-80GB [41]

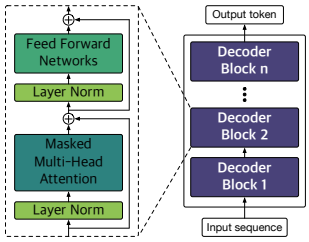


Fig. 2: Overview of Transformer-based LLM architecture.

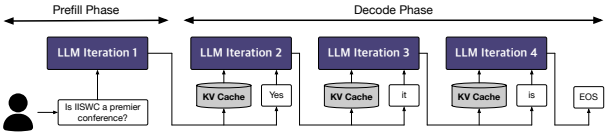


Fig. 3: Overview of the Prefill and Decode phases in LLM inference.

B. LLM Inference

LLM inference typically consists of two phases: the *Prefill Phase* and the *Decode Phase*, as shown in Figure 3. During the prefill phase, the model processes all input prompts from the user and produces a new token used for initial input for the decode phase. This phase involves computing the hidden representations for the entire input sequence simultaneously, which is usually *computationally intensive*, especially for large models and long sequences. The prefill phase typically makes the system *compute-bound* due to the high volume of parallel processing required. In contrast, the decode phase generates one token at a time, using the previously generated token as input for the next step. This phase continues iteratively until a predefined sequence length is reached or an end-of-sequence (EOS) token is produced. Despite processing only one token per step, the decode phase demands *substantial I/O operations* making it *memory-bound*. Importantly, the *KV cache*, which has become a de-facto optimization for the decode phase to avoid recalculating key/value vectors in every iteration, requires a large memory space of $2B\ (BF16) * 2\ (Key/Value) * n_{layers} * d_{model} * n_{seq} * n_{batch}$, where n_{layers} is the number of layers, d_{model} is the hidden dimension of the model, n_{seq} is the sequence length, n_{batch} is the batch size. Therefore, efficient memory management is crucial to handle these KV caches and the LLM weights. Understanding these phases and their computational demands is essential for optimizing LLM inference performance.

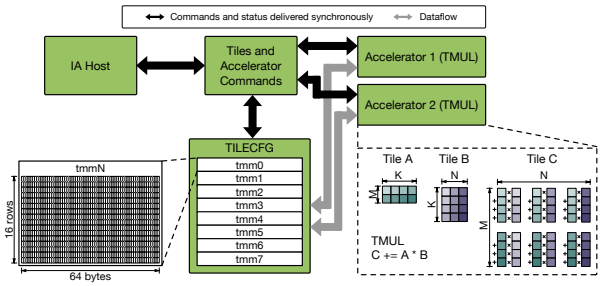


Fig. 4: Overview of Intel AMX architecture [9].

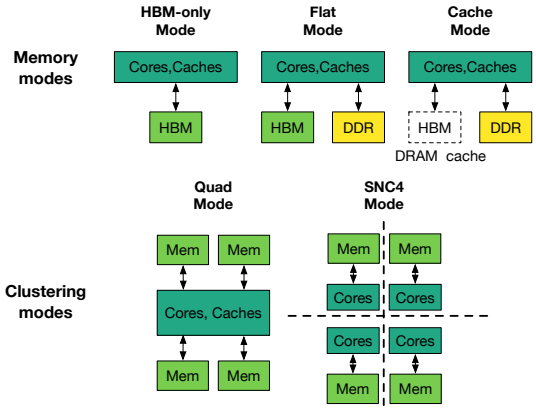


Fig. 5: Different memory and clustering modes in SPR Max CPU servers [38].

C. LLM Inference Key Metrics

There are various metrics used to compare the LLM inference performance [28], [49], [58] such as the time to first token (TTFT), the time per output token (TPOT), end-to-end (E2E) latency, and system throughput in terms of the number of generated tokens per second. The importance of each metric varies depending on the use case. For a real-time chatbot service, TTFT is crucial, as users expect quick responses for a seamless experience. In real-time translation service used in live media broadcasts, a slight delay at the start might be acceptable, but TPOT must be low to keep up with the natural pace of speech. If one is conducting research that requires batch processing of text data for sentiment analysis, the priority is to complete the entire job as quickly as possible, meaning higher system throughput is preferred over faster processing of individual texts. Therefore, in this work, instead of focusing on a specific metric, we characterize various workloads and evaluate different HWs using these three metrics as further explained in Section IV-A.

D. Matrix Multiplication Accelerators on CPUs

To improve the performance of machine learning applications on CPUs, hardware vendors have incorporated dedicated accelerators into their processors and introduced instruction set architecture to support it. Examples include Intel Advanced Matrix Extensions (AMX) [38], IBM Power CPU Matrix Multiply Assist (MMA) [10], and ARM Matrix Extension (ME) [18]. These extensions are designed to enable efficient matrix multiplication, one of the key operations in machine learning. In this paper, we conduct performance characterization of LLM inference on the latest Intel CPUs. Intel AMX is supported by Intel Xeon processors based on the Sapphire Rapids (SPR) architecture [38] and introduces two main components: Tile and Tile Matrix Multiply Unit (TMUL). As shown in Figure 4, Tile is defined as a 2-dimensional register with a size of 1KB, consisting of 16 rows of size of 64 bytes. Therefore, each tile can store 32 elements for the 16-bit brain floating point (BF16) data type and 64 elements for the 8-bit integer (INT8) data type. TMUL is a hardware unit that accelerates matrix-multiply computation on tiles and supports both BF16 and INT8 data types [22], [23], [38].

E. NUMA Architecture

Non-Uniform Memory Access (NUMA) architecture is a design widely used in modern data centers to improve performance and scalability by denoting lower-bandwidth remote memory regions across sockets. In recent Intel CPUs, NUMA accesses can refer either to inter-socket memory accesses that use Intel's UPI (Ultra Path Interconnect) or a remote domain within a socket (usually denoting levels of cache sharing). Intel's Sapphire Rapids Max Series CPUs support multiple NUMA options via different memory and clustering modes as described in Figure 5.

HBM can operate in three different memory modes: (1) HBM-only, (2) Flat, and (3) Cache, each with distinct benefits. HBM-only mode offers maximum bandwidth and lowest latency for workloads that fit within the HBM capacity, but is limited by the HBM size. Flat mode combines HBM and DDR as separate NUMA nodes, providing greater memory capacity and flexibility but requiring software to manage memory allocation. Cache mode uses HBM as a cache for DDR, simplifying implementation without software changes but may lead to sub-optimal performance.

These SPR CPUs also support two different clustering modes: (1) *Quadrant* (quad) and (2) *Sub-NUMA Clustering-4 (snc)*. *Quadrant* mode presents a single address space (NUMA node) to software, requiring no NUMA awareness, and is ideal for applications sharing large data structures across all cores. *Sub-NUMA Clustering-4* mode, the default, divides each CPU into four sub-NUMA clusters, seen as separate NUMA nodes but offers higher bandwidth and lower latency. Used together, these memory and clustering modes can provide flexible options to optimize performance for various high-demand computational tasks, including LLM inference.

III. Challenges and Opportunities in LLM

Despite LLM's excellent performance across various domains, serving LLMs for inference presents significant challenges, primarily due to the memory required to store the models and the memory consumption to store KV cache.

High memory requirement in LLM inference: Figure 6 illustrates the memory footprint required to store the OPT [57] and LLaMA [50] models based on their scale. As shown in Figure 6, models with a large number of parameters, such as LLaMA2-70B, have a substantial memory footprint for storing model weights. Even the latest GPUs such as NVIDIA A100

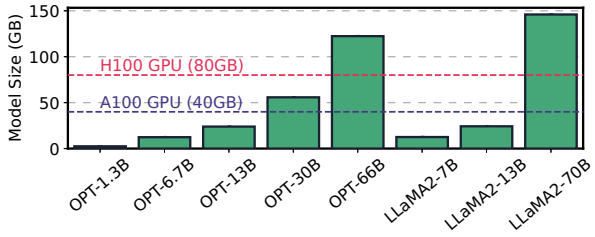
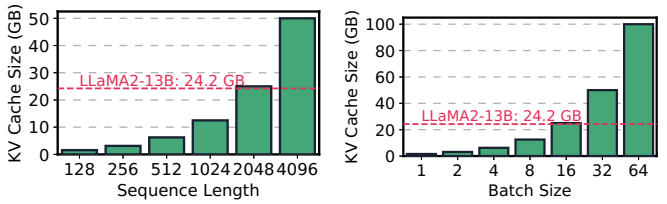


Fig. 6: Memory footprint required to store the parameters of each model using the FP16 data type.

with 40GB or NVIDIA H100 with 80GB memory cannot fit these models into a single GPU. For example, loading the LLaMA2-70B model onto GPUs requires at least two H100 GPUs. Practical industry LLMs such as GPT-3 175B [5] require over 320GB to solely store the parameters, necessitating at least five H100 GPUs.



(a) KV cache size for various sequence length (batch size=16)

(b) KV cache size for various batch size (seq. length=2048)

Fig. 7: KV cache memory footprint for different sequence lengths and batch sizes in LLaMA2-13B. The dotted line represents the size of LLaMA2-13B model.

Recent LLM inference serving systems exploit KV caching to reduce computation during the decode phase. While KV caching can make the output token generation faster, the memory consumption by the KV cache becomes significant when using larger batch sizes for throughput or generating longer sequences. Figure 7 shows the memory footprint required to store the KV cache for LLaMA2-13B with different sequence lengths and batch sizes. The figure illustrates that the memory capacity for the KV cache increases linearly with both the sequence lengths and batch sizes, eventually surpassing the model size when using large batch sizes and sequence lengths. Recent inference serving systems including TorchServe [8] and NVIDIA Triton Server [52] support batched LLM inference to efficiently utilize hardware resources and handle multiple user requests. Additionally, various prior studies have been proposed to support longer contexts in the latest LLMs and recent chatbot services; for instance, ChatGPT-4.0 [1], and Gemini [47], now support longer sequence lengths, such as 32K length. Hence, the memory required for the KV cache often exceeds the model size, imposing a significant burden on GPU or other accelerator memory.

Offloading-based LLM inference systems: To support LLM inference when model sizes and KV caches exceed a single GPU's memory capacity, recent frameworks such as FlexGen [49] and DeepSpeed [45] provide offloading-based inference, storing model weights, activations, and KV caches in CPU memory. However, these systems face performance degradation since model weights, activations, and KV caches stored on CPU must be transferred via slow PCIe interconnect during computation.

Opportunities of CPUs for LLM inference: As explained in Section II-D, the latest CPUs can provide advanced accelerators that create new opportunities to leverage CPUs for LLM inference serving. The latest CPUs include dedicated matrix multiplication accelerators, similar to NVIDIA's Tensor core [33], as well as fast access to large DDR5 DRAM memory capacities and High Bandwidth Memory (HBM) on recent Sapphire Rapids CPUs [38], [46]. DRAM capacity on these platforms can also be further expanded using recent technologies such as CXL [34], and CCIX [26]. Therefore, in scenarios where larger models such as LLaMA2-70B shown in Figure 6 or extensive KV cache sizes exceed the memory capacity of a single GPU as shown in Figure 7, the large memory capacity of the CPU offers an opportunity to utilize CPUs for LLM inference.

IV. Characterizing LLM Inference on CPUs

A. Characterization Methodology

	CPU 1 (ICL CPU)	CPU 2 (SPR CPU)
Generation	IceLake (ICL)	Sapphire Rapids (SPR)
CPU	Xeon 3rd 8352Y	Xeon 4th Max 9468
Core Frequency	2.20 GHz	2.10 GHz
Compute Throughput (BF16)	18.0 TFLOPS (AVX-512)	25.6 (AVX-512) / 206.4 (AMX) TFLOPS
# of cores (per socket) / sockets	32 / 2	48 / 2
L1D / L2 Cache (per core)	48 KB / 1.25 MB	48 KB / 2 MB
L3 Cache	48 MB	105 MB
CPU Memory	DDR4 256 GB	DDR5 512 GB, HBM 128 GB
Memory Bandwidth[2]	156.2 GB/s	DDR5 233.8 GB/s, HBM 588 GB/s

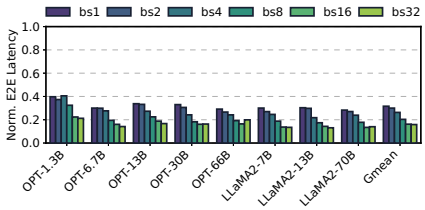
TABLE I: Evaluation Setup for CPU Servers.

Experimental setup: In this section, we conducted our experiments using two different CPUs, an Intel 3rd generation and a 4th generation CPU, to analyze the performance of common LLMs with recent advanced CPU features, including Intel AMX. The detailed CPU server configurations are shown in Table I. To measure the throughput and latency of LLM inference on CPUs, we use Intel Extension for Pytorch v2.3 [19] that provides optimizations for Intel CPUs.

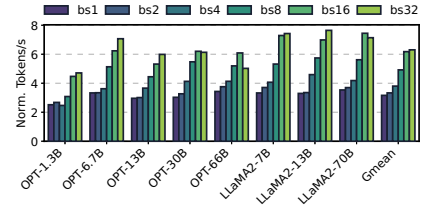
Models: We use recent open-sourced representative LLM families, OPT [57] and LLaMA-2 [50] with different model sizes. We evaluate OPT models with 1.3B, 6.7B, 13B, 30B, and 66B parameters and LLaMA-2 models with 7B, 13B, and 70B parameters. For all our experiments, we set an input sequence length of 128 and an output sequence length of 32 with varying batch sizes, ranging from 1 to 32. In Section V, we also explore the impact of longer input sequence lengths.

Metrics: To measure the latency and throughput of LLM inference, we employed various metrics widely used in previous studies [28], [49], [58] as different metrics are prioritized based on the use cases as mentioned in Section

[2]Measured on a single socket using the STREAM benchmark [35].

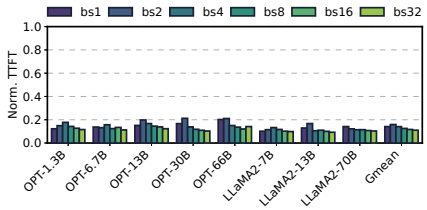


(a) End-to-End latency comparison

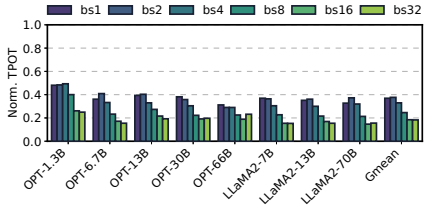


(b) End-to-End throughput comparison

Fig. 8: Latency and throughput comparison of Intel ICL and SPR CPUs in LLM inference.

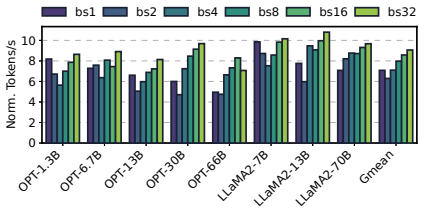


(a) Prefill phase latency comparison

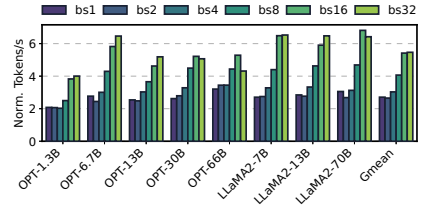


(b) Decode phase latency comparison

Fig. 9: Latency comparison of Intel ICL and SPR CPUs for prefill and decode phases.



(a) Prefill phase throughput comparison



(b) Decode phase throughput comparison

Fig. 10: Throughput comparison of Intel ICL and SPR CPUs for prefill and decode phases.

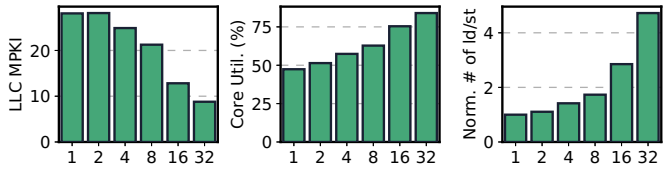


Fig. 11: Comparison of various hardware performance counters, including LLC misses per kilo instruction (MPKI), for LLaMA2-13B model inference on SPR CPU with different batch sizes. The number of load/store instructions is normalized to that of batch size 1.

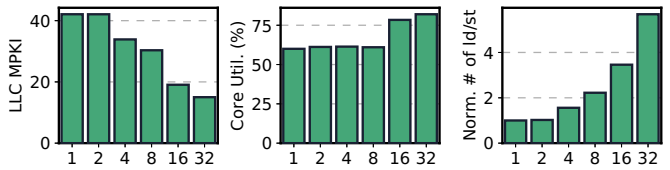


Fig. 12: Comparison of various hardware performance counters, including LLC misses per kilo instruction (MPKI), for OPT-66B model inference on SPR CPU with different batch sizes. The number of load/store instructions is normalized to that of batch size 1.

II-C. We use three metrics to measure LLM inference latency: (1) end-to-end latency (E2E latency), which is the total time taken to generate the entire output sequence. (2) the time to first token (TTFT), which indicates the time to generate the first token during the prefill phase, and (3) the time per output token (TPOT), which is the average time taken to generate subsequent tokens during the decode phase. To measure the overall LLM inference throughput, we use tokens generated per second, defined as the total number of generated tokens divided by the end-to-end latency. Similarly, the throughput for both the prefill phase and the decode phase is also measured in terms of tokens generated per second. We also report hardware performance counters, such as cache miss rates and UPI utilization, using Linux perf [11] and Intel VTune profiler [20] to better understand the results.

B. Performance of LLM Inference on CPUs

To understand the performance of LLM inference on CPUs, we first perform empirical studies using two different CPU servers and analyze the performance impact of recently introduced features in recent Intel Sapphire Rapids CPUs such as AMX.

Performance comparison between Xeon IceLake And Sapphire Rapids CPUs: In this experiment, the Xeon 3rd generation ICL 8352Y CPU (*referred to as ICL CPU*) was configured with 32 cores, while the Xeon 4th generation SPR Max 9468 CPU (*referred to as SPR CPU*) was configured with 48 cores. The SPR CPU was set to *Quadrant*, *Flat* memory mode to achieve optimal performance in this context. Figure 8 shows the end-to-end latency and throughput comparison results for LLM inference on the ICL and SPR CPUs with varying batch sizes from 1 to 32. *Each bar is normalized to the results of the ICL CPU.* As seen in Figure 8, the SPR CPU consistently shows reduced latency and improved throughput compared to the ICL CPU across all LLMs and batch sizes. On average, the SPR CPU achieves an end-to-end latency reduction in the range of 68.4% to 84.1% compared to the ICL CPU. Additionally, the token generation throughput of the SPR CPU is improved by 3.2 to 6.3×. These normalized results highlight the performance benefits gained from the use of both the matrix multiplication accelerator and high-bandwidth memory on the SPR Max CPU.

Figures 9 and Figure 10 show the comparison of latency and throughput during the prefill and decode phases of LLM inference on the ICL and SPR CPUs, respectively. As shown in Figure 9, TTFT during the prefill phase decreased by an average of 84.1% to 89%. In the decode phase, TPOT is reduced from 62.3% to 81.7% on average. The SPR CPU shows throughput improvement ranging from 6.3 to 9.1× in the prefill phase and an increase in the range of 2.7 to 5.5× in the decode phase compared to the ICL CPU. The significant reduction in latency and improvement in throughput during the prefill phase is due to AMX support on the SPR Max CPU, while the throughput improvement in the memory-bound decode phase is made possible by the higher memory bandwidth provided by HBM.

Figure 11 and Figure 12 compare various hardware performance counters during the inference of LLaMA2-13B and OPT-66B models as the batch size increases. With larger

batch sizes, both models exhibit a decrease in LLC MPKI and an increase in core utilization, indicating a shift towards a more compute-bound execution. This trend highlights the significant performance gap between ICL and SPR CPUs at larger batch sizes, driven by Intel AMX support and the high memory bandwidth of HBM.

When using a batch size of 32, end-to-end latency is reduced by 84.1%, and throughput increases by 6.3× compared to the ICL CPU.

Key Finding#1: With AMX support, larger cores and cache, and HBM integration, the SPR Max CPU significantly reduces latency and increases throughput for BF16 LLM inference compared to the ICL CPU.

Performance impact of SPR CPU server configurations: As discussed in Section II-E, when using the SPR CPU series servers, different settings can be configured for memory and clustering modes, which could change the performance a lot. Since we employ DDR5 memory in our server setup, excluding the HBM-only mode, there are four possible combinations of memory and cluster modes: (1) quad_cache, (2) quad_flat, (3) snc_cache, and (4) snc_flat.

To evaluate the impact of these different server configurations on LLM inference latency and throughput, we conducted experiments using each configuration. We utilized a single socket with 48 cores to avoid performance degradation due to inter-socket communication. Linux *numactl* [25] was used to appropriately bind the memory nodes and cores of the NUMA node. In flat mode, memory allocation prioritized HBM memory, with DDR memory being used only when the allocation exceeded 64GB, as each socket has 64GB of HBM. For snc mode, each of the four NUMA nodes within the socket was bound to 12 physical cores.

Figure 13 presents the comparison of various LLM inference metrics, averaged across all workloads and batch sizes, normalized to the quad_cache configuration. As shown in Figure 13, overall, quad clustering mode showed better latency and throughput compared to snc mode. Although snc mode theoretically provides better performance by localizing data within each NUMA domain, our results suggest that when data allocation is not properly managed, performance can degrade due to inefficient memory access and increased inter-core communication. This indicates potential for further software optimization to fully exploit snc mode.

Additionally, explicitly leveraging the HBM (flat mode) resulted in better performance. This improvement can be attributed to the decode phase being more memory-bound compared to the prefill phase, making the utilization of higher memory bandwidth crucial for enhancing performance. In this context, effectively utilizing HBM's memory bandwidth can significantly contribute to improved performance. Figure 15 shows the results of various hardware performance counters when running LLaMA2-13B LLM inference with a batch size of 8 across different server configurations. LLaMA2-13B is selected for this analysis as it effectively demonstrates the performance trends observed across different configurations. The figure shows snc mode suffered from performance degradation due to frequent remote cache accesses to other NUMA nodes. In terms of memory mode, flat mode slightly outperformed cache mode by leveraging HBM's higher bandwidth more effectively.

In conclusion, considering all the metrics, we found that the quad_flat mode configuration delivered the best performance.

Key Finding#2: Proper memory and clustering configurations are essential for optimizing performance. The *Flat* memory mode with *Quadrant* clustering offers the best latency and throughput for LLM inference.

Performance impact of the number of cores: Figure 14 shows the comparison results in terms of various metrics for LLM inference when varying the number of cores. Each metric result is averaged across all evaluated LLMs and different batch sizes from 1 to 32, normalized to the results using 12 cores. As shown in Figure 14, using a larger number of cores does not consistently demonstrate the best results for all metrics. For instance, the end-to-end latency was lowest with 48 cores, achieving a 59.8% reduction compared to using 12 cores. 1.8× improvement over 12 cores. When analyzing each phase of LLM inference, i.e. the prefill and decode phases, we observe the following: In the prefill phase, using 48 cores reduced latency by 65.9% compared to using 12 cores, showing the best performance. In the decode phase, using 48 cores achieved a 54.6% reduction in latency, which was the best result. For the overall throughput considering both the prefill and decode phases, using 48 cores improved performance by 2.2× and 1.7×, respectively, compared to using 12 cores.

Figure 16 shows the comparison of various metrics such as physical core utilization, UPI utilization, etc. with varying numbers of cores during LLaMA2-7B model inference. Overall, increasing the number of cores resulted in better performance in terms of latency and throughput compared to using fewer cores. However, as shown in Figure 16, using 96 cores led to poor performance due to the need for inter-socket communication via Intel UPI, negatively impacting both latency and throughput. In summary, we found that using 48 cores is the best configuration when considering all latency and throughput-related metrics, resulting in a 59.8% reduction in latency and a 1.8× improvement in overall throughput compared to using 12 cores.

Considering the performance impact of the SPR CPU NUMA configurations and the number of CPU cores, we use the *quad_flat* configuration with 48 cores for the Intel SPR CPU results in the subsequent sections.

Key Finding#3: Using 48 SPR cores with HBM maximizes core utilization and minimizes inter-socket communication, resulting in the best performance across models.

V. Performance Comparison with GPUs

In this section, we compare the performance result of CPUs with that of GPUs for various LLMs.

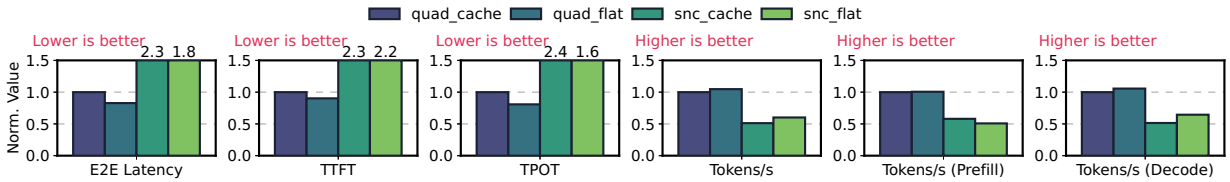


Fig. 13: Normalized latency and throughput metrics for different SPR CPU server configurations. Each result is normalized to quad_cache configuration. Each metric is averaged across all evaluated LLMs and batch sizes from 1 to 32.

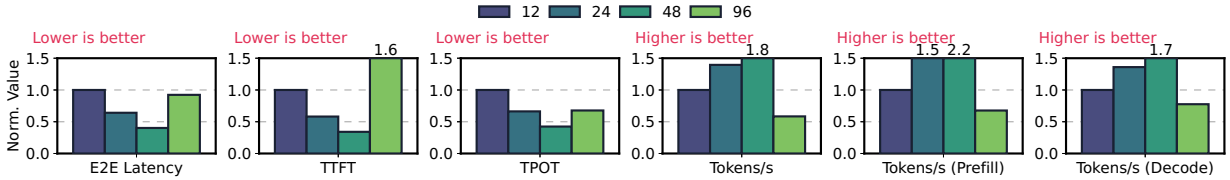


Fig. 14: Normalized latency and throughput metrics for different core configurations. Each result is normalized to 12 core configuration. Each metric is averaged across all evaluated LLMs and batch sizes from 1 to 32.

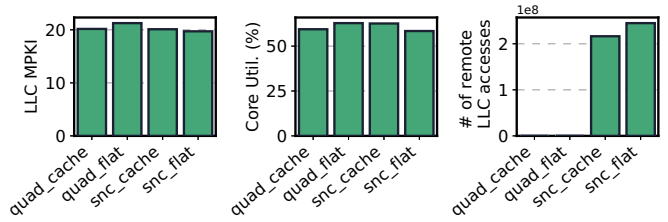


Fig. 15: Comparison of LLC misses per kilo instruction (MPKI), core utilization, and the number of remote LLC accesses for LLaMA2-13B model with batch size 8 with different server configurations.

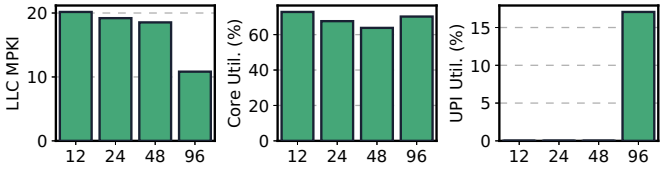


Fig. 16: Comparison of LLC misses per kilo instruction (MPKI), core utilization, UPI utilization for LLaMA2-7B model with batch size 8 as core count increases.

A. Evaluation Methodology

Experimental setup: To compare the LLM inference performance on CPUs and GPUs with various compute capabilities and memory sizes, we use two distinct server-class GPUs: the NVIDIA A100 GPU with 40GB memory and the NVIDIA H100 GPU with 80GB memory. Table II shows the detailed GPU server configurations. To measure the performance of LLM inference on GPUs, we employ FlexGen [49], a state-of-the-art offloading-based LLM inference engine designed to achieve high throughput even with limited GPU memory. FlexGen enables GPU to offload model weights, activations, and KV cache to CPU memory. In these experiments, we set the input sequence length to 128 and the output sequence length to 32.

B. End-to-End Performance

Figure 17 illustrates the end-to-end latency and token generation throughput comparison of LLM inference with a batch size of 1 on both CPUs and GPUs for OPT, and LLaMA-2 models. Each result is normalized to the result of the SPR Max CPU. As shown in the Figure 17, for smaller models that fit into GPU memory such as OPT-1.3B, OPT-6.7B, LLaMA2-7B, OPT-13B, LLaMA2-13B, GPUs outperform the SPR Max CPU in terms of both latency and throughput. For example, with the A100 GPU, the OPT-13B model showed a reduction in end-to-end latency by 65.5% compared to the CPU, while the H100 GPU showed a reduction of 72.8% for the OPT-13B model. In terms of throughput, the A100 GPU demonstrated an improvement of 2.9×, and the H100 GPU showed an improvement of 3.7× over the CPU.

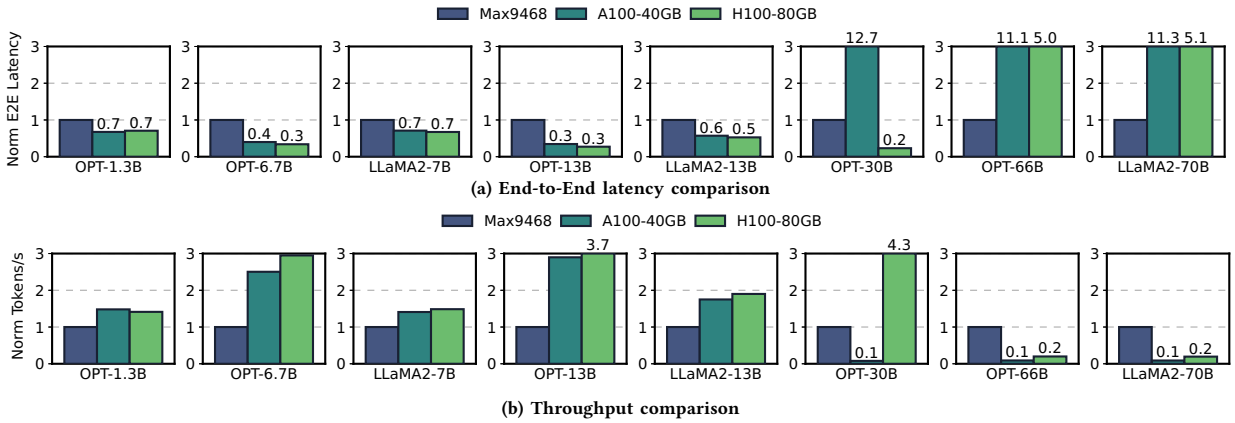
However, for models that exceed GPU memory, such as OPT-30B (in the case of the A100), OPT-66B, and LLaMA2-70B, the CPU outperforms GPUs in both latency and throughput. For instance, while the H100 GPU could accommodate the entire OPT-30B model and perform better than the CPU, the A100 GPU needs to offload model weights and activations on CPU memory, which must then be loaded on demand over the PCIe bus. In this scenario, the CPU reduced latency by 92.1% and improved throughput by 12.7× compared to the A100 GPU. Furthermore, compared to the H100 for OPT-66B, the SPR CPU showed an 80.1% reduction in latency and 5× improvement in throughput.

To analyze how much time GPUs spend on data loading over the PCIe bus when using offloading-based LLM inference methods, we break down the execution time for large models. Figure 18 (a) shows the execution time breakdown for the A100 GPU running the OPT-30B model, and Figure 18 (b)

	GPU 1	GPU 2
GPU	NVIDIA A100	NVIDIA H100
Number of SMs	108	132
Compute Throughput (BF16)[3]	312 TFLOPS	756 TFLOPS
L1 / L2 Cache	192 KB / 40 MB	256 KB / 50MB
GPU Memory	40 GB	80 GB
Memory Bandwidth[4]	1299.9 GB/s	1754.4 GB/s
CPU-GPU Interconnect	PCIe 4.0, 64 GB/s	PCIe 5.0, 128 GB/s

TABLE II: Evaluation Setup for GPU Servers.

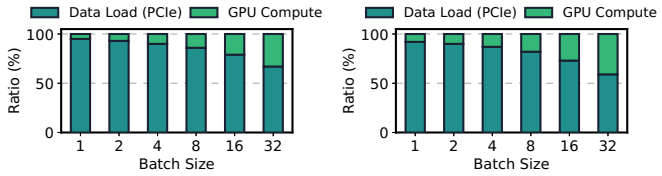
[3]TFLOPS values are for dense operations without sparsity.
[4]Measured using the STREAM benchmark [35].



(a) End-to-End latency comparison

(b) Throughput comparison

Fig. 17: LLM end-to-end inference and throughput comparison of Max9468 CPU and GPUs (A100 and H100) for batch size=1. Each result is normalized to SPR Max 9468 CPU.



(a) OPT-30B on A100 GPU

(b) OPT-66B on H100 GPU

Fig. 18: GPU execution time breakdown during LLM inference for larger models (OPT-30B and OPT-66B) on A100 and H100 GPUs.

presents the same breakdown for the H100 GPU running the OPT-66B model, both using offloading-based LLM inference with batch sizes ranging from 1 to 32. As shown in Figure 18, the A100 GPU spends between 67% and 95% of its total execution time on data loading over the PCIe bus, while the H100 GPU spends between 59% and 92% of its execution time on data loading when running the OPT-66B model. However, FlexGen's zig-zag block scheduling technique [49], which overlaps data transfer with computation, reduces the time spent on data loading via the PCIe bus as the batch size increases. Consequently, for smaller models that do not require offloading, the high compute throughput of GPUs leads to a wider performance gap between the CPU and GPU, as illustrated in Figure 19. In contrast, for larger models that require offloading, although the CPU still outperforms the GPU, the performance gap narrows due to the efficiency of the scheduling technique.

We also note that new Grace-Hopper Superchip would see lower overheads for offloading from DRAM to the integrated H100 due to its higher NVLink bandwidth (900 GB/s versus PCIe 5.0's 128 GB/s), albeit at a cost of ∼4x of the SPR CPU and DDR5 [40].

Key Finding#4: Overall, GPUs outperform CPUs in LLM inference, but AMX-enabled CPUs can achieve lower latency and higher throughput for larger models requiring offloading.

C. Sensitivity to Sequence Length

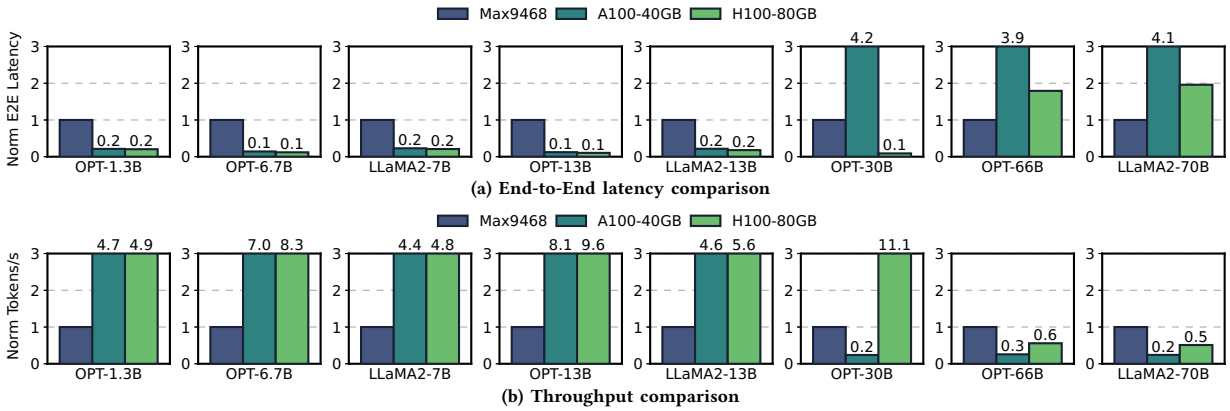
Figure 20 shows the comparison of LLM inference latency and throughput between CPUs and GPUs across varying sequence lengths. The x-axis represents the number of input tokens. In all experiments, we set the number of output tokens to 32 and increased the input prompt sizes from 128 to 1024. As the number of input tokens increases, GPU latency and throughput remain stable, while the SPR Max 9468 CPU shows more variability. This is due to the CPU's lower compute throughput and memory bandwidth, resulting in less favorable performance scalability. Interestingly, for larger models such as LLaMA2-70B, the CPU outperforms the GPU in both latency and throughput across all sequence lengths. This is primarily due to the significant time spent on data loading via the PCIe bus when the batch size is set to 1, as shown in Figure 18.

Similarly, Figure 21 compares the performance of CPUs and GPUs at different sequence lengths with a batch size of 16. As the batch size increases to 16, the performance gap between CPUs and GPUs widens, particularly for smaller models. For larger models such as LLaMA2-70B, we observed that at sequence lengths of 256 or more, the H100 GPU—even when using offloading-based LLM inference—achieves lower latency compared to the CPU. This is because, at these longer sequence lengths, the CPU's LLM inference throughput continues to decline, resulting in lower performance than the H100. However, in the case of the A100 GPU, the CPU outperforms the GPU across all sequence lengths. This demonstrates that lower PCIe bandwidth significantly degrades the performance of offloading-based LLM serving systems.

Key Finding#5: For larger batch sizes, GPUs outperform CPUs in small models. Even in larger models that require offloading, CPUs may underperform at longer sequence lengths due to lower compute throughput.

VI. Potential Optimizations for LLM Inference on CPUs

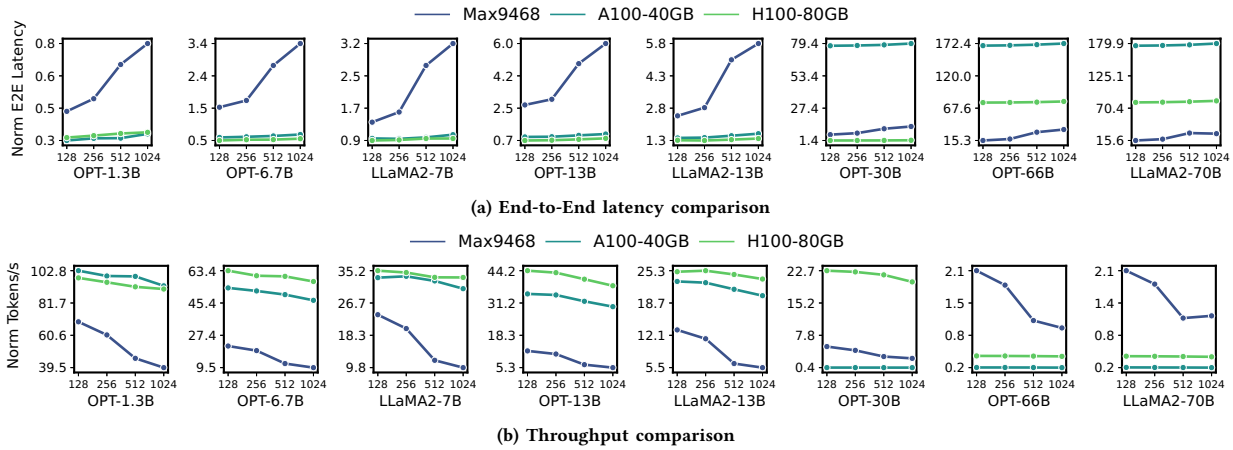
NUMA-aware designs: In Section IV, we compared LLM inference performance on SPR CPU with various NUMA configurations based on memory and clustering modes. Our results indicate that exposing a single NUMA node per socket and using HBM explicitly showed the best performance. For large models, even with both DDR and HBM memory on



(a) End-to-End latency comparison

(b) Throughput comparison

Fig. 19: LLM inference and throughput comparison of Max9468 CPU and GPUs (A100 and H100) for batch size=16. Each result is normalized to SPR Max 9468 CPU.



(a) End-to-End latency comparison

(b) Throughput comparison

Fig. 20: LLM inference and throughput comparison of Max9468 CPU and GPUs (A100 and H100) for batch size=1.

a single socket, memory capacity can become insufficient, necessitating the use of memory from other sockets.

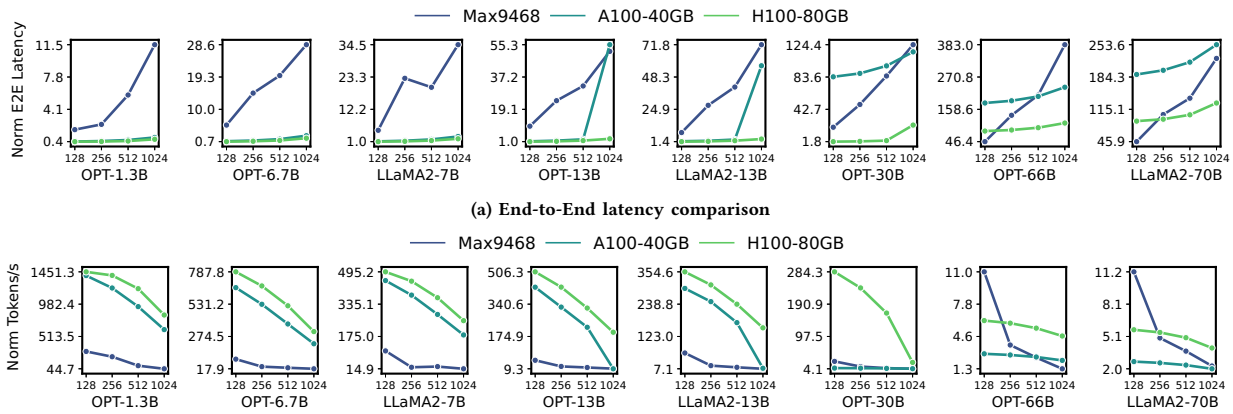
Recent studies [27], [32], [54] found that not all activations in LLMs are equally important; certain activations are more critical. Leveraging this insight can significantly aid NUMA node data placement. By placing the important activations (hot data) in HBM and local DDR memory and storing less critical activations (cold data) in remote DDR memory in other sockets, a NUMA-aware data placement can enhance LLM inference performance on CPUs while reducing the negative impacts of remote memory accesses.

CPU-GPU Hybrid Execution: Our performance comparison results with GPUs in Section V demonstrated that SPR Max CPU can outperform GPUs for models larger than the GPU memory size, particularly when the batch size and sequence length are not too large. FlexGen [49] typically underutilizes CPU computation resources, using them only for attention score calculations. However, our evaluation suggests that exploiting CPU computation resources can benefit large models that would require large amounts of PCIe data transfer. Therefore, exploiting the CPU for layer computations in such scenarios with small batch sizes or designing a cooperative execution model that partitions layers between the CPU and GPU can significantly improve LLM inference latency and throughput. For smaller models, while CPU performance may not surpass that of the GPU, leveraging CPU computation resources can enhance overall hardware utilization in data centers where GPU resources are fully occupied. This approach can also reduce time-to-first-token (TTFT), thereby improving user experience.

VII. Related Work

A. Performance Characterization of Deep Learning Workloads

Many recent research works have tried to characterize deep learning workloads [17], [29], [43], [44], [51]. A significant portion of the research is focused on GPUs, with several studies concentrating on training [17], [44], [51] and others extending the scope to inference while still emphasizing GPUs [29], [43]. There have also been efforts to understand deep learning inference on CPUs. DLsim [6] proposed a simulation infrastructure for deep learning workloads. Recent work [42] performed large-scale workload characterization on Facebook cloud for both CPUs and GPUs, but its workloads are limited to recommendation, computer vision, and GRU/LSTMs based models which have different characteristics compared to the transformer-based LLMs. This work aims to better understand LLM inference on CPUs equipped with specialized



(a) End-to-End latency comparison

(b) Throughput comparison

Fig. 21: LLM inference and throughput comparison of Max9468 CPU and GPUs (A100 and H100) for batch size=16.

hardware accelerators and instructions for efficient matrix multiplications.

B. Deep Learning Acceleration on CPUs

There have been previous studies to optimize the performance of deep learning workloads on CPUs [14]–[16], [31], [48]. NeoCPU [31] proposes convolutional neural network (CNN) inference acceleration techniques tailored for CPUs using graph-level joint optimizations. Graphite [15] proposes software-hardware co-design to optimize graph neural network (GNN) training and inference on CPUs. To support the key operations in deep learning such as matrix multiplication, and convolution, LIBXSMM [14], [16] proposes Just-In-Time (JIT) compilation-based kernels optimized for Intel Xeon processors while RASA [23] and VEGETA [22] introduce dense and sparse matrix multiplication support through matrix engines. Recent work [48] proposes weight-only quantization with negligible accuracy loss and provides optimized kernels for the latest CPUs to enable efficient LLM inference on CPUs.

C. LLM Inference Optimizations

Efficient LLM Inference Batching: To improve the resource utilization during LLM inference, recent serving systems have devised various batching mechanisms [2], [28], [39], [56]. FasterTransformer [39] processes a batch of requests concurrently for better resource utilization. Orca [56] introduced iteration-level scheduling which allows the scheduler to dynamically create batches for better utilization. vLLM [28] introduces paged attention that allows the system to batch more sequences together. Sarathi-Serve [2] builds on chunked-prefill in Sarathi [3] that enables dynamically batching without stalling ongoing decode phase.

Offloading-based LLM Inference: Recent LLM serving systems use offloading-based LLM inference serving to enable resource-constrained GPUs to serve larger models exceeding GPU memory size [45], [49], [53]. By storing model parameters and activations in CPU memory, those systems can reduce the peak GPU memory usage during LLM inference. Although offloading methods provide an opportunity to support larger models, it incurs high performance overhead due to the frequent data transfer via PCIe bus as also shown in this work. To reduce the performance overhead in offloading-based LLM inference, FlexGen [49] proposes a zig-zag scheduling mechanism to increase the system throughput and minimize total execution time given resource constraints.

VIII. Conclusion

In this work, we provide a comprehensive characterization of LLM inference on the latest CPUs equipped with matrix multiplication accelerators and HBM. We demonstrate the potential computational capabilities of the latest Intel CPUs featuring AMX and HBM. Our findings highlight the importance of various NUMA server configurations and the appropriate number of cores for optimizing LLM inference performance. Furthermore, by comparing the latest CPU with server-class GPUs, we discover that CPU can outperform GPUs when using larger models exceeding GPU memory with all batch sizes when the sequence length is small. Based on these extensive experimental results, we provide insights and discuss potential optimizations for efficient CPU-based LLM inference. We believe our work opens new opportunities for exploiting CPUs as computation units for LLM inference, rather than merely as offloading devices.

IX. Acknowledgement

This work was supported in part through research infrastructure and services provided by the Rogues Gallery testbed [55] hosted by the Center for Research into Novel Computing Hierarchies (CRNCH) at Georgia Tech and was partially supported by NSF PPOSS 2119523 and Intel. The Rogues Gallery testbed is primarily supported by the National Science Foundation (NSF) under NSF Award Number #2016701.

We would like to thank Christopher J. Hughes at Intel Labs for his valuable feedback, and Aaron Jezghani and Sterling Peet at Georgia Tech for their help with the server environment setup. We also thank the anonymous reviewers for their feedback on improving the paper.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee, "Taming throughput-latency tradeoff in llm inference with sarathi-serve," *arXiv preprint arXiv:2403.02310*, 2024.

[3] A. Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, "Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills," *arXiv preprint arXiv:2308.16369*, 2023.

[4] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen *et al.*, "Deep Speech 2: End-to-end speech recognition in English and Mandarin," in *International Conference on Machine Learning (ICML)*, 2016.

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[6] Z. Chishti and B. Akin, "Memory system characterization of deep learning workloads," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 497–505.

[7] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[8] P. T. Contributors, "TorchServe," https://pytorch.org/serve/, 2020.

[9] I. Cooperation, "Intel architecture instruction set extensions programming reference," *Intel Corp., Mountain View, CA, USA, Tech. Rep*, pp. 319 433–030, 2016.

[10] J. P. de Carvalho, J. E. Moreira, and J. N. Amaral, "Compiling for the ibm matrix engine for enterprise workloads," *IEEE Micro*, vol. 42, no. 5, pp. 34–40, 2022.

[11] A. C. De Melo, "The new linux'perf'tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[12] Y. Ding, L. L. Zhang, C. Zhang, Y. Xu, N. Shang, J. Xu, F. Yang, and M. Yang, "Longrope: Extending llm context window beyond 2 million tokens," *arXiv preprint arXiv:2402.13753*, 2024.

[13] A. Firoozshahian, J. Coburn, R. Levenstein, R. Nattoji, A. Kamath, O. Wu, G. Grewal, H. Aepala, B. Jakka, B. Dreyer *et al.*, "Mtia: First generation silicon targeting meta's recommendation systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[14] E. Georganas, S. Avancha, K. Banerjee, D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on simd architectures," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 830–841.

[15] Z. Gong, H. Ji, Y. Yao, C. W. Fletcher, C. J. Hughes, and J. Torrellas, "Graphite: optimizing graph neural networks on cpus through cooperative software-hardware techniques," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 916–931.

[16] A. Heinecke, G. Henry, M. Hutchinson, and H. Pabst, "Libxsmm: accelerating small matrix multiplications by runtime code generation," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 981–991.

[17] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo *et al.*, "Characterization of large language model development in the datacenter," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 709–729.

[18] D. A. Iliescu and F. Petrogalli, "Arm scalable vector extension and application to machine learning," *Retrieved October*, 2018.

[19] Intel, "Intel Extension for PyTorch," https://github.com/intel/intel-extension-for-pytorch, 2020.

[20] ——, "VTune Profiler," https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html, 2020.

[21] ——, "4th generation xeon cpu max 9468 processor," https://www.intel.com/content/www/us/en/products/sku/232596/intel-xeon-cpu-max-9468-processor-105m-cache-2-10-ghz/specifications, 2023.

[22] G. Jeong, S. Damani, A. R. Bambhaniya, E. Qin, C. J. Hughes, S. Subramoney, H. Kim, and T. Krishna, "Vegeta: Vertically-integrated extensions for sparse/dense gemm tile acceleration on cpus," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 259–272.

[23] G. Jeong, E. Qin, A. Samajdar, C. J. Hughes, S. Subramoney, H. Kim, and T. Krishna, "Rasa: Efficient register-aware systolic array matrix engine for cpu," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 253–258.

[24] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–14.

[25] A. Kleen, "A numa api for linux," *Novel Inc*, 2005.

[26] D. Koenen and J. Defilippi, "Ccix: a new coherent multichip interconnect for accelerated use cases," 2017.

[27] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, W. Leiserson, S. Moore, N. Shavit, and D. Alistarh, "Inducing and exploiting activation sparsity for fast inference on deep neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5533–5543.

[28] 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.

[29] M. LeMay, S. Li, and T. Guo, "Perseus: Characterizing performance and cost of multi-tenant serving for cnn models," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020, pp. 66–72.

[30] B. Lin, T. Peng, C. Zhang, M. Sun, L. Li, H. Zhao, W. Xiao, Q. Xu, X. Qiu, S. Li *et al.*, "Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache," *arXiv preprint arXiv:2401.02669*, 2024.

[31] Y. Liu, Y. Wang, R. Yu, M. Li, V. Sharma, and Y. Wang, "Optimizing {CNN} model inference on {CPUs}," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 1025–1040.

[32] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re *et al.*, "Deja vu: Contextual sparsity for efficient llms at inference time," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 137–22 176.

[33] S. Markidis, S. W. Der Chien, E. Laure, I. B. Peng, and J. S. Vetter, "Nvidia tensor core programmability, performance & precision," in *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*. IEEE, 2018, pp. 522–531.

[34] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "Tpp: Transparent page placement for cxl-enabled tiered-memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 742–755.

[35] J. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," *http://www. cs. virginia. edu/stream/*, 2006.

[36] A. Meta, "Introducing meta llama 3: The most capable openly available llm to date, 2024," *URL https://ai. meta. com/blog/meta-llama-3/. Accessed on April*, vol. 26, 2024.

[37] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi *et al.*, "Specinfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 932–949.

[38] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula *et al.*, "Sapphire rapids: The next-generation intel xeon scalable processor," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 44–46.

[39] NVIDIA, "FasterTransformer," https://github.com/NVIDIA/FasterTransformer, 2021.

[40] ——, "Nvidia grace hopper superchip architecture in-depth," https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/, 2022.

[41] ——, "Nvidia's h100 ai gpus cost up to four times more than amd's competing mi300x," https://www.tomshardware.com/tech-industry/artificial-intelligence/nvidias-h100-ai-gpus-cost-up-to-four-times-more-than-amds-competing-mi300x-amds-chips-cost-dollar10-to-dollar15k-apiece-nvidias-h100-has-peaked-beyond-dollar40000, 2023.

[42] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur *et al.*, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv preprint arXiv:1811.09886*, 2018.

[43] P. Patel, E. Choukse, C. Zhang, Í. Goiri, B. Warrier, N. Mahalingam, and R. Bianchini, "Characterizing power management opportunities for llms in the cloud," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 207–222.

[44] H. Qi, L. Dai, W. Chen, Z. Jia, and X. Lu, "Performance characterization of large language models on high-speed interconnects," in *2023 IEEE Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2023, pp. 53–60.

[45] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[46] I. Z. Reguly, "Comparative evaluation of bandwidth-bound applications on the intel xeon cpu max series," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1236–1244.

[47] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. Lillicrap, J.-b. Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[48] H. Shen, H. Chang, B. Dong, Y. Luo, and H. Meng, "Efficient llm inference on cpus," *arXiv preprint arXiv:2311.00502*, 2023.

[49] 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.

[50] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[51] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia, "Characterizing deep learning training workloads on alibaba-pai," in *2019 IEEE international symposium on workload characterization (IISWC)*. IEEE, 2019, pp. 189–202.

[52] Z. Wang, Y. Wei, M. Lee, M. Langer, F. Yu, J. Liu, S. Liu, D. G. Abel, X. Guo, J. Dong *et al.*, "Merlin hugectr: Gpu-accelerated recommender system training and inference," in *Proceedings of the 16th ACM Conference on Recommender Systems*, 2022, pp. 534–537.

[53] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[54] H. Xia, Z. Zheng, Y. Li, D. Zhuang, Z. Zhou, X. Qiu, Y. Li, W. Lin, and S. L. Song, "Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity," *arXiv preprint arXiv:2309.10285*, 2023.

[55] J. S. Young, J. Riedy, T. M. Conte, V. Sarkar, P. Chatarasi, and S. Srikanth, "Experimental insights from the rogues gallery," in *2019 IEEE International Conference on Rebooting Computing (ICRC)*, Nov 2019, pp. 1–8.

[56] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[57] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[58] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.