Exploiting Intel Advanced Matrix Extensions (AMX) for Large Language Model Inference

Hyungyo Kim, *Student Member, IEEE*, Gaohan Ye, Nachuan Wang, Amir Yazdanbakhsh, *Member, IEEE*, and Nam Sung Kim, *Fellow, IEEE*

***Abstract*—The ever-increasing number of parameters in Large Language Models (LLMs) demands many expensive GPUs for both inference and training. This is because even such a high-end GPU such as NVIDIA A100 can store only a subset of parameters due to its limited memory capacity. To reduce the number of required GPUs, especially for inference, we may exploit the large memory capacity of (host) CPU to store not only all the model parameters but also intermediate outputs which also require a substantial memory capacity. However, this necessitates frequent data transfers between CPU and GPU over the slow PCIe interface, creating a bottleneck that hinders the accomplishment of both low latency and high throughput in inference. To address such a challenge, we first propose CPU-GPU cooperative computing that exploits the Advanced Matrix Extensions (AMX) capability of the latest Intel CPU, codenamed Sapphire Rapids (SPR). Second, we propose an adaptive model partitioning policy that determines the layers of a given LLM to be run on CPU and GPU, respectively, based on their memory capacity requirement and arithmetic intensity. As CPU executes the layers with large memory capacity but low arithmetic intensity, the amount of data transferred through the PCIe interface is significantly reduced, thereby improving the LLM inference performance. Our evaluation demonstrates that CPU-GPU cooperative computing, based on this policy, delivers 12.1× lower latency and 5.4× higher throughput than GPU-only computing for OPT-30B inference when both CPU-GPU and GPU-only computing store the model in CPU memory.**

***Index Terms*—Large language models, advance matrix extensions, cooperative heterogeneous computing.**

I. LLM Inference With Limited GPU Resources

TRANSFORMER enables massively parallel training and an ability to learn long range dependencies [1]. Since its advent, transformer-based language models have been pushing their performance by scaling the model size [2]. Now the number of parameters (or weights) reaches several hundreds of billions to trillions, earning their name of large language models (LLMs). Among various transformer-based LLMs, decoder-only LLMs have been widely adopted for natural language processing (NLP) due to its exceptional ability to generalize across different NLP tasks [3] exemplified by GPT, Llama, PaLM, Falcon, and OPT.

Manuscript received 4 March 2024; revised 22 April 2024; accepted 3 May 2024. Date of current version 28 May 2024. This work was supported in part by IBM IIDAI and Google. *(Corresponding author: Nam Sung Kim.)*

Hyungyo Kim, Gaohan Ye, Nachuan Wang, and Nam Sung Kim are with the University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA (e-mail: hyungyo2@illinois.edu; gaohany2@illinois.edu; nachuan3@illinois.edu; nskim@illinois.edu).

Amir Yazdanbakhsh is with Google Deepmind, Mountain View, CA 94043 USA (e-mail: ayazdan@google.com).

Digital Object Identifier 10.1109/LCA.2024.3397747

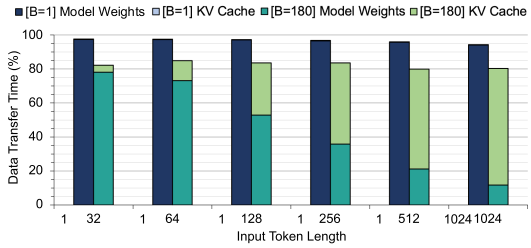

Fig. 1. The data transfer time (`memcpy` time) of model weights and KV caches between the CPU and the A100 GPU during OPT-30B model inference, generating 32 output tokens with varying input token lengths for batch sizes of $B = 1$ and $B = 180$.

However, deploying such LLMs requires many expensive GPUs to accommodate the entire model parameters and intermediate outputs, such as KV cache and activation, with the aggregated memory capacity of these GPUs. For instance, Megatron-Turing Natrual Language Generation (NLG) model with 540 B parameters requires at least thirteen NVIDIA A100 GPUs, each with 80 GB memory. Therefore, it is imperative to reduce the number of required GPUs for wider and more sustainable deployment of such massive LLMs

Toward such a goal, especially for inference, we may leverage the large memory capacity of the (host) CPU to store not only all the parameters but also intermediate outputs which also require substantial memory space [4], [5], [6]. In such a setup, the forward path is executed layer by layer via sequentially loading the parameters, KV cache, and activation of each single layer into the GPUs. However, this necessitates frequent data transfers between the CPU and the GPU over the slow PCIe interface, creating a bottleneck that hinders the accomplishment of both low latency and high throughput in LLM inference. Fig. 1 presents the percentage time consumed for `memcpy` between an Intel 4th-generation Xeon Scalable Processor (codenamed Sapphire Rapids (SPR)) and an NVIDIA A100 GPU. We run the OPT-30B model inference with FlexGen for various input token lengths and a 32 output token length. Although only 30% of the model weights are offloaded to the CPU memory for a batch size of 1, more than 94% of the time is spent for transferring the parameters. This underscores data transfer over the slow PCIe interface as the performance bottleneck.

To reduce the cost of data transfers, the latest throughput-optimized inference approach using a single GPU batches as many input tokens as possible once a single layer is loaded into the GPU memory [6]. The key innovation of such an approach is to sequentially iterate through the process of loading a batch into the GPU memory, computing a single layer, and then transferring its KV cache and activation back to the CPU memory. As such,

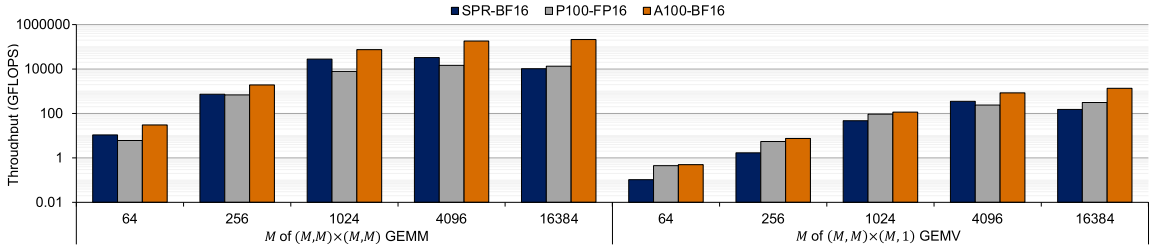

Fig. 2. General matrix multiplication (GEMM) and general matrix-vector multiplication (GEMV) throughput of Sapphire Rapids (SPR), P100, and A100 GPU for varying sizes of half precision (BF16/FP16) square matrix operands.

the cost of transferring parameters is amortized across input tokens in a batch. However, as the batch size and/or the input token length increases, the KV cache size also grows, eventually surpassing the model size. That is, the transfer of KV cache emerges as the primary bottleneck. For example, with a batch size of 180 and an input token length of 1024, the KV cache size for the OPT-30B model reaches around 240 GB, i.e., four times larger than the memory capacity to store parameters. As the input token length becomes longer, a larger percentage of the time is spent for transferring the KV cache (Fig. 1). Consequently, even with a notably large batch size, the data transfer time remains persistently high, ranging from 80% to 85%. Lastly, the parameters and KV cache can be quantized to reduce data transfers or even fit them into the limited GPU memory, but this comes at the cost of lower inference accuracy.

Recent work has proposed using the CPU to run the attention scoring layer such that the large KV cache transfer over the PCIe interface can be obviated [6]. However, the traditional CPU's low compute capability requires a significant amount of time to execute the attention scoring layer for long input tokens, giving rise to either negligible improvement or even notable degradation in performance. To address the limited compute capability of the CPU in the prior work, this work proposes CPU-GPU cooperative computing that leverages the powerful capability of Advanced Matrix Extensions (AMX) in the latest Intel CPUs. Subsequently, this work proposes an adaptive model partitioning policy that determines the layers to be run on the CPU and the GPU, respectively after holistically examining the memory capacity requirement and arithmetic intensity of each LLM layer. The CPU with AMX can execute layers with higher arithmetic intensity than the traditional CPU, without being a performance bottleneck. As such, it allows us to offload more layers to the CPU, thereby offering less data transfers over the PCIe interface and higher inference performance. Our evaluation demonstrates that CPU-GPU cooperative computing, based on this policy, delivers 12.1× lower latency and 5.4× higher throughput than GPU-only computing with data offloading for OPT-30B inference.

II. Intel Advacned Matrix Extensions (AMX)

To enhance the compute capability of the CPU for ML applications, Intel introduced AMX in 2021, extensions to the x86 instruction set architecture along with a necessary hardware support. AMX supports SIMD operations for matrix multiplications with 8-bit integer (INT8) and 16-bit brain floating point (BF16) data types. AMX was first supported in 2023 by the SPR CPU.

The Intel AMX accelerator architecture consists of two core elements: (1) a 2-D array of registers (tiles) that store sub-arrays of matrix operands and (2) a dedicated accelerator, the tile matrix multiply unit (TMUL), designed to operate on these tiles [7]. TMUL leverages parallel architecture to accelerate matrix operations. The CPU dispatches AMX instructions, such as tile load/store and accelerator commands to the multi-cycle AMX units. The accelerator's memory accesses remain coherent with the CPU's memory accesses.

Fig. 2 presents the throughput of general matrix multiplication (GEMM) and general matrix-vector multiplication (GEMV) offered by a dual-socket, 80-core SPR server and two NVIDIA GPUs, P100 and A100. The data type is half-precision floating point (BF16/FP16) and the matrix dimension M ranges from $M = 64$ to 16384 covering the GEMM/GEMV dimensions of the smallest OPT model (OPT-125 M) to the largest (OPT-175B). At a matrix dimension of $M = 1024$, SPR achieves a remarkable GEMM throughput of 32.7 TFLOPS, 15% of A100's peak throughput (212 TFLOPS). SPR outperforms P100 for GEMM throughput across different dimensions and achieves 2.2× higher peak throughput. However, for memory-bound GEMV operations, SPR achieves 2.4× and 4.6× lower throughput than A100 and P100, respectively, primarily due to the SPR's lower memory bandwidth.

The v3.1 results of running the MLPerf benchmark suite demonstrate that SPR achieves 7.8% and 32.9% per-device-throughput of H100 and A100, respectively, in a 99.9%-accuracy-achieving offline inference scenario with GPT-J 6B model [8]. Despite falling short of the performance exhibited by the high-end GPUs, we find that the commendable compute throughput and the CPU's large memory capacity can effectively assist the high-end GPUs for memory-capacity-bound operations. That is, the CPU can be effectively used to enhance the LLM inference performance when the number of accessible GPUs is limited.

III. Model Partitioning Between CPU and GPU

A. Arithmetic Intensity of LLM Layers

The text generation process of decoder-based LLMs consists of two stages: (1) prefill and (2) decoding stages. Fig. 3 depicts the two stages with the OPT model, a prominent open-source decoder-only LLM. The prefill stage serves as the initial phase where the first output token is generated with the entire input token sequence fed into the model. KV matrices corresponding to each input token are computed and cached during this stage. The decoding stage follows, where only the single output token generated from the previous iteration is fed to the model for the prediction of a subsequent token. Thus, the parameters of the linear layers are used for the forward path of a single token

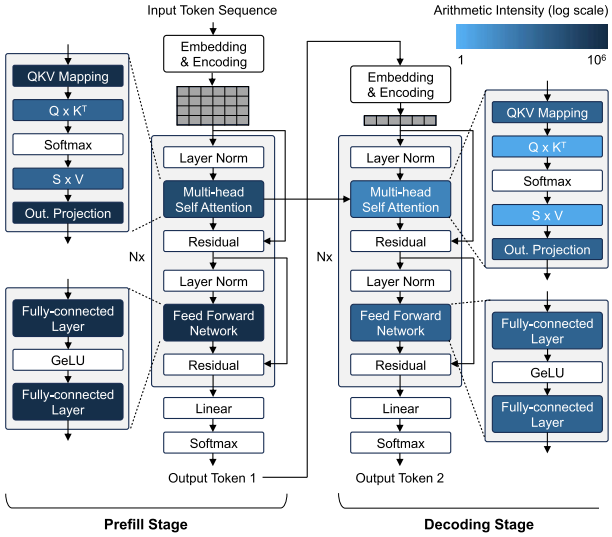

Fig. 3. The OPT model inference data flow. The heatmap depicts the ARI of the prefill and decoding stages for an OPT-30B model with dimension $d_{model} = 7168$, input sequence length $L = 512$ and batch size $B = 180$.

TABLE I
ARITHMETIC INTENSITY OF LAYERS. 'ATTENTION SCORING' DENOTES FOR $Q \times K^\top$, $S \times V$ LAYERS. $d_{model} = 7168$ FOR THE OPT-30B MODEL

Stage	Compute Layers	Arithmetic Intensity
Prefill	QKV Mapping	$6(1/d_{model} + 3/BL)^{-1}$
	Attention Scoring	L
	Out. Projection	$2(1/d_{model} + 1/BL)^{-1}$
	FC Layers	$8(1/d_{model} + 4/BL)^{-1}$
Decoding	QKV Mapping	$6(1/d_{model} + 3/B)^{-1}$
	Attention Scoring	$2(1 + 1/L)^{-1}$
	Out. Projection	$2(1/d_{model} + 1/B)^{-1}$
	FC Layers	$8(1/d_{model} + 4/B)^{-1}$

(possibly batched). For the attention scoring layer ($Q \times K^\top$ and $S \times V$ layers), KV vectors from the QKV mapping layer are concatenated with the cached KV matrices, eliminating redundant computations at the cost of memory space. Since KV matrices are unique to each input token within the batch, they are not shared across the batch.

This distinct compute aspect between the two stages and between the linear layers and the attention scoring layer creates a wide range of arithmetic intensity (ARI) for the operations involved in the inference as visually represented with a heat map in Fig. 3. The ARI of each layer in each stage can be derived as a function of B, L, and d_{model} as summarized in Table I, where B, L, and d_{model} denote batch size, input token length and dimension of the model, respectively. Since the non-matrix operations (Softmax, Layer Norm, and Residual) present low ARI, they are often fused with matrix operations to reduce the data movement. Therefore, we omit the ARI analysis of non-matrix operations.

In the prefill stage, the ARI of the linear layers (QKV mapping, output projection, and FC layers) ranges from hundreds to tens of thousands across different input token lengths and batch sizes. The ARI of the attention scoring layer in the prefill stage also exhibits high ARI as the ARI scales with L. However, it is orders of magnitude lower than that of the linear layers in the prefill stage for batched cases. Note that the operands of the attention scoring layer (KV matrices) during this stage are generated from the previous layer, obviating the data transfer over the PCIe interface if both the layers are computed on the same device.

In the decoding stage, meanwhile, operations generally exhibit low ARIs. The ARI of the linear layers roughly scales with B, thus, the batch size determines whether these operations become bounded by data transfers. Lastly, the ARI of the attention scoring layer is capped to 2. That is, their operations are bounded by data transfers.

B. ARI-Based Model Partitioning

Considering that lower ARI operations incur a higher overhead of data transfers, it is more effective to compute low ARI operations with the CPU with AMX if the operands resides in the CPU memory. The condition, which determines whether to offload a given layer to the CPU can be formulated by comparing the latency of two options:

$$\frac{dN_l}{BW_C} + \frac{C_l}{TH_C} \leq \frac{dN_l}{BW_P} + \frac{C_l}{TH_G}, \tag{1}$$

where d, N_l, and C_l denote the size of the operand data type in Bytes, the number of parameters of l-th operands, and the number of arithmetic operations, respectively. BW_C and BW_P denote the CPU's memory bandwidth and the PCIe bandwidth in GB/s, respectively. TH_C and TH_G denote the compute throughput of the CPU and the GPU for a given data type in GFLOPS, respectively. Here, the latency of the GPU memory is ignored as that of data transfers latency is dominated by that of the PCIe interface. By dividing (1) by N_l and substituting $\text{ARI}_l = C_l/N_l$, the range of ARI values that determine whether to offload a given layer to the CPU can be derived as:

$$\text{ARI} \leq d \left(\frac{1}{BW_P} - \frac{1}{BW_C}\right) \left(\frac{1}{TH_C} - \frac{1}{TH_G}\right)^{-1}. \tag{2}$$

The ARI of each layer can be compared against the RHS threshold to determine which compute device to use between the CPU and the GPU. We make an exception for the attention scoring layer in the prefill stage, since we can compute the attention scoring layer where we computed the previous QKV mapping to leverage the KV cache locality. Specifically for OPT-30B and the SPR-A100 system deployed in Section IV, three model partitioning policies are derived from (2) when the operands (i.e., parameters, KV cache, and activation) reside in the CPU memory. (Policy 1) The CPU computes all the layers for a small batch size and a short input token length. (Policy 2) The CPU computes only the layers in the decoding stage for a small batch size but a long input token length. (Policy 3) The CPU computes only the attention scoring layer in the decoding stage for large batch size. The policy is determined dynamically upon the arrival of requests, according to the batch size and input token length.

IV. EVALUATION

We set up a system based on two Intel Xeon Platinum 8460H SPR CPUs, each with 40 cores, 8 DDR5-4800 channels, and 384 GB DRAM capacity, to evaluate the performance of inference. We also use the Intel Extension for Pytorch (IPEX)and FlexGen [6] for inference based on SPR and A100, respectively.

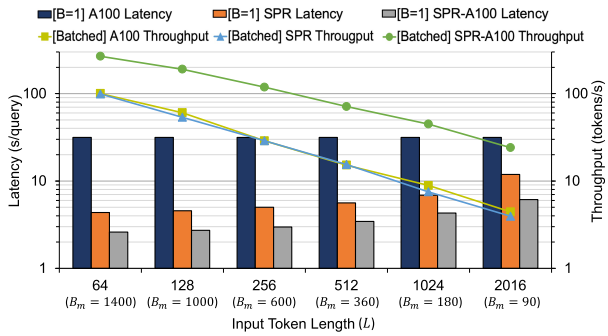

Fig. 4. Inference performance of SPR, A100, and SPR-A100 for the OPT-30B model. The latency and throughput are measured for $B = 1$ and a maximum-batch case, respectively. The maximum batch size B_m for each input token length is set such that the KV cache and parameter fit into the CPU memory.

We evaluate the OPT-30B model that is the largest model supported by IPEX at the moment. The latency of CPU-GPU cooperative inference (denoted by SPR-A100) with each partitioning policy is calculated as follows:

$$T_{P1} = \alpha\big(T_{A100} - T_{PCIe,A100}\big) + (1-\alpha)T_{SPR},$$
$$T_{P2} = T_{P1} + (1-\alpha)\big(T_{P,A100} - T_{P,SPR}\big),$$
$$T_{P3} = T_{P2} + (1-\alpha)\big(T_{lin,D,A100} - T_{lin,D,SPR}\big), \quad (3)$$

where T_{Pi} denotes the inference latency of policy i, α denotes the portion of parameters stored in the A100 memory, T_{A100}, T_{SPR} denote the total inference latency of A100 and SPR, respectively, $T_{PCIe,A100}$ denotes the `memcpy` time of A100, $T_{P,A100}$, $T_{P,SPR}$ denote the prefill stage latency of A100 and SPR, respectively, and $T_{lin,D,A100}$, $T_{lin,D,SPR}$ denote the latency of the linear layers in the decoding stage of A100 and SPR, respectively. $T_{P,A100}$ and $T_{P,SPR}$ are provided by both frameworks and $T_{lin,D,A100}$ and $T_{lin,D,SPR}$ are obtained by measuring the decoding stage latency with $L = 1$ input. Note that T_{A100}, $T_{P,A100}$, and $T_{lin,D,A100}$ are measured for the case where all the data is offloaded to the host CPU memory. SPR-A100 follows the overlapping schedule of FlexGen. We leave the full system integration and evaluation as future work.

The performance of SPR-, A100-, and SRP-A100-based inference is evaluated for two cases: online (latency-sensitive) and offline (throughput-optimized) cases. Latency (s/query) of an unbatched ($B = 1$) inference is measured as the performance metric for the online case, whereas the per-token throughput (tokens/s) of maximally-batched inference is measured for the offline case. For the online case, 30% of the parameters are offloaded to the CPU memory for inference of both A100 and SPR-A100, as they are the maximum number of parameters that can fit into the A100's 40 GB memory for $B = 1$. For the offline case, parameters, KV cache, and activation are entirely offloaded to the CPU memory. The maximum batch size of each input for the offline case are set such that the parameters, KV cache, and the activation fit into the host CPU memory, as shown in Fig. 4. The number of output tokens is set to 32 for all evaluation. Furthermore, the latency model deployed in (1) was validated through a comparison between the calculated model inference latency and the measurement results across the data points depicted in Fig. 4. The normalized mean squared errors of the latency model are $2.9\text{e}{-3}$ and 0.18 for SPR and A100, respectively.

For the online case, SPR-A100 offers $12.1\times$ and $1.9\times$ lower latency than A100 and SPR, respectively. Policy 1 is deployed for $L = 64$ to $L = 1024$, while Policy 2 is deployed for $L = 2016$ for SPR-A100. The latency from using A100 remains consistently high across L since the parameter transfer over the PCIe bottlenecks the performance, even when only 30% of the parameters are offloaded to the CPU memory. SPR exhibits lower latency than A100 as there is not data transfers through the PCIe interface. Yet, the latency significantly increases with larger L as the number of compute operations for the prefill stage increases quadratically with L. With Policy 2 being deployed for SPR-A100 for $L = 2016$, the compute-heavy prefill stage is computed by A100, leading to a relatively lower increase in latency.

For the offline case, SPR-A100 gives up to $5.4\times$ and $6.1\times$ higher throughput than A100-only and SPR-only, respectively. Policy 3 is deployed for $L = 64$, while Policy 2 is deployed for $L = 128$ to $L = 2016$ for SPR-A100. The number of operations for the liner layers in the decoding stage increases with batch size B, underscoring the compute throughput difference between SPR and A100. Thus, SPR-A100 shifts to Policy 3, leveraging the high compute throughput of A100 for the linear layers in the decoding stage.

V. Conclusion and Future Work

This work proposed CPU-GPU cooperative computing, exploiting AMX in the latest Intel CPU for efficient LLM inference with a limited number of GPUs, along with a model partitioning policy. We showed that it delivers $12.1\times$ lower latency and $5.4\times$ higher throughput than computing only with a GPU for inference of the OPT-30B model.

References

[1] A. Vaswani et al., "Attention is all you need," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[2] J. Kaplan et al., "Scaling laws for neural language models," 2020, arXiv: *2001.08361*.
[3] T. Wang et al., "What language model architecture and pretraining objective works best for zero-shot generalization?," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 22964–22984.
[4] B. Pudipeddi, M. Mesmakhosroshahi, J. Xi, and S. Bharadwaj, "Training large neural networks with constant memory using a new execution algorithm," 2020, arXiv: *2002.05645*.
[5] R. Y. Aminabadi et al., "DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2022, pp. 1–15.
[6] Y. Sheng et al., "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proc. Int. Conf. Mach. Learn.*, 2023, pp. 31094–31116.
[7] Intel Architecture Instruction Set Extensions and Future Features. Mar. 2023. Accessed: 2024. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/774990/intel-architecture-instruction-set-extensions-programming-reference.html
[8] MLPerf Inference: Datacenter Benchmark Suite Results. Accessed: 2024. [Online]. Available: https://mlcommons.org/benchmarks/inference-datacenter/