Improving Throughput-oriented Generative Inference with CPUs

Daon Park
daon@csap.snu.ac.kr
Seoul National University
Seoul, Republic of Korea

Sungbin Jo
goranmoomin@snu.ac.kr
Seoul National University
Seoul, Republic of Korea

Bernhard Egger
bernhard@csap.snu.ac.kr
Seoul National University
Seoul, Republic of Korea

ABSTRACT

Despite recent attempts to reduce the number of parameters of large language models (LLMs), their parameter data is still too large to fit into a single GPU. With the emergence of throughput-oriented tasks, high-throughput generative inference frameworks for LLMs on a single commodity GPU leverage GPU, DRAM, and NVMe to run inference on large models with terabytes of data. Our analysis of the technique shows that the runtime is dominated by data transfers of the weights, leading to a low utilization of both the GPU and the CPU. In this paper, we increase the throughput and decrease the total latency of state-of-the-art frameworks by including the CPU as a compute device and overlapping computations on the CPU with GPU data transfers. Our work shows a promising improvement of around 40% in throughput and total latency, with potential room for further improvements.

CCS CONCEPTS

• **Computing methodologies → Parallel algorithms**; **Natural language generation**.

KEYWORDS

Large language models, latency reduction, CPU offloading

ACM Reference Format:
Daon Park, Sungbin Jo, and Bernhard Egger. 2023. Improving Throughput-oriented Generative Inference with CPUs. In *14th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '23), August 24–25, 2023, Seoul, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3609510.3609815

1 INTRODUCTION

Today, deep learning is applied to a wide variety of tasks [2, 4, 9]. One of the most researched and recognized fields is natural language processing [6, 10, 11]. Recent large language models (LLMs) demonstrate excellent performance across a wide range of tasks [5, 15]. The quest for better and better results has lead to an explosion of the parameter data required by state-of-the-art LLMs [8, 14]. Attempts to reduce the model size by using an extremely high number of batches per iteration [16] resulted in a smaller number of parameters while maintaining a certain level of accuracy. Nevertheless, such models are still too large to fit into a single GPU. For example, to accommodate the 700 GB of GPT-3's parameters [6], eight NVIDIA H100 GPUs are required. Systems that fit eight high-end GPUs are not easily available to individuals and also companies. Putting the high-memory requirements and the cost aside, accommodating the entire model into a number of GPUs and running them in a model-parallel and pipeline-parallel fashion is suitable for latency-sensitive tasks. Indeed, many the tasks such as data wrangling, information extraction, and benchmarks are throughput-oriented and feed millions of tokens in batches to the models. A high numbers of batches increases the overall throughput at the expense of latency.

Recent studies attempt to balance the latency-throughput trade-off by reducing the resource requirements for LLM inference with a throughput-oriented process. Several existing throughput-oriented frameworks, such as ZeRO-Inference [3], HuggingFace Accelerate [1], and FlexGen [13], leverage the memory hierarchy and increase the number of batches during inference to improve the throughput. ZeRO-Inference transfers weigths of the model to the system DRAM or NVMe and utilizes other optimizations to minimize the latency of transferring layer weights from the DRAM or NVMe memory to the GPU memory. HuggingFace Accelerate focuses on training methods and efficiently accumulates gradients without affecting the process. HuggingFace Accelerate also supports inference using the same technique used during training, where a hook mechanism sequentially moves the next layer's weights into the GPU memory, much like the process in ZeRO-Inference. However, ZeRO-Inference and HuggingFace Accelerate do not achieve acceptable throughput on a single GPU, especially at small batch sizes. FlexGen improves the aforementioned frameworks by expanding the tensor placement and computation delegation search space, while also allowing manual configuration options of the ratios for weights, key-value cache, and intermediate activation.

Existing throughput-oriented frameworks focus on hiding the data transfer time with layer prefetches and efficient GPU I/O computation, but do not exploit the opportunity to use the CPUs for computation. While GPUs are orders of magnitudes faster than CPUs during computation, the weights of LLMs are so large that the weight transfers often dominate the execution time of LLM inference on single commodity GPU system. In this paper, we add employ the CPU as an additional compute device to seize opportunities of overlapping communication with computations to improve inference latency. The main idea is to execute the first half of a decoder layer's computations on the CPUs while the weights for the second half of the decoder layer are transferred to the GPU, thereby overlapping computation with data transfers and reducing the weight transfer overhead. To support efficient offloading of some part of the workload to the CPU, we need to consider the different characteristics of the workload for different batch sizes and different phases of LLM inference. We carefully determine which data should remain in DRAM and is processed by the CPUs while we send the remainder of the data to the GPUs and show that the latency of throughput-oriented workload with the OPT-30B model [18] can be reduced by up to 40%.

The remainder of this paper is organized as follows. Section 2 discusses the large language models and related work. The CPU offloading technique is presented in Section 3 and evaluated in Section 4. Section 5, finally, concludes this paper.

2 BACKGROUND AND RELATED WORK

Large language models. Most of today's LLMs are based on the decoder architecture [17], which is comprised of layers of layer normalization, multi-head attention, another layer normalization, and a feed-forward network. As shown in Figure 1, the majority of the workload is attributed to matrix multiplications in the multi-head attention and the feed-forward network. All matrix multiplications with a single arrow as an input require a weight tensor. Multi-head attention and the feed-forward network require four and two weight tensors, respectively. The multi-head attention weights are in the shape of (*embedding dimension, embedding dimension*), and the feed-forward network has two weights composed of (*embedding dimension, intermediate dimension*) and (*intermediate dimension, embedding dimension*). These weights are not small in size. In the OPT-30B model [18], the embedding dimension is set to 7168 and the intermediate dimension to 28672, making a single layer contain around 616 million parameters and require 1.2GB of memory with FP16 precision.

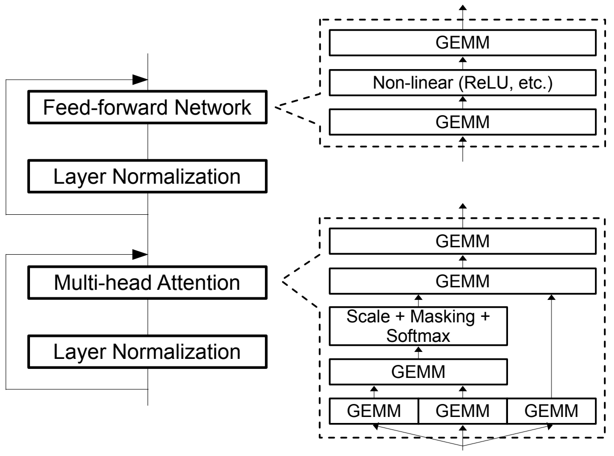


Figure 1: The OPT model decoder layer architecture.

All weights are reused in every iteration. The user can decide the number of iterations that take place by specifying how many output tokens the model should generate. One output token is generated per iteration, and the generated token is concatenated with the input tokens and fed into the network as input for the next iteration. This means that there are redundant computations along the way since many of the inputs are reused in future iterations. The key-value cache technique can be used to remove such redundant computations [7]. Multi-head attention is comprised of three matrix multiplications at the start, and the outcomes of these computations are named query, key, and value respectively. The first iteration computes the matrix multiplications with all input tokens and creates the initial key and value matrices called the *key-value cache.* After the key-value cache has been created, it can be reused in the next iteration, thus requiring only the output token as an input. The output token will create another set of key and value matrices which are concatenated to the previous key-value cache. The first iteration that creates the key-value cache is called the *prefill phase*, and the following iterations are called the *generation phase*. The resulting key-value cache has a dimension of (*batches, # of input tokens + # of generated tokens, embedding dimension*) for both key and value matrices. The key-value cache is not small in size. For example, with 512 input tokens and 32 iterations of the OPT-30B model using FP16 precision, the key-value cache requires around 16MB per batch and multi-head operation. Since there is multi-head attention in every layer, a single batch requires around 750MB of key-value cache memory in the OPT-30B model.

FlexGen. Existing throughput-oriented works such as ZeRO-Inference [3] and HuggingFace Accelerate [1] support offloading large models to DRAM and NVMe. However, inefficient I/O scheduling and tensor placement hurt throughput performance. FlexGen [13] alleviates these shortcomings by formally defining a search space of possible offloading strategies that considers the computation schedule, tensor placements, and computation delegation. FlexGen leverages the zigzag scheduling methodology, which does not finish the entire inference process to completion, but rather divides the batches into smaller ones and runs inference one small batch at a time. For example, if there are 100 batches to compute in total and the user decides to run 10 batches at a time, other related works finish the inference process of 10 batches first, then compute the next 10 batches, and so on. On the other hand, FlexGen's zigzag scheduling transfers the weights of an operation (multi-head attention or feed-forward network) first, computes the results for all 100 batches by keeping 10 batches at a time within the GPU, and then proceeds forward. Using the zigzag scheduling technique, FlexGen tries to find optimal weight, intermediate activation, and key-value cache tensor placements for optimal performance. In addition, FlexGen lets the user choose the weight, key-value cache, and intermediate activation tensor placements during inference so that the user can try out different combinations of tensor placements, and also configure how many batches at a time can be placed in VRAM.

3 CPU OFFLOADING

Motivation. Figure 2 plots the latency breakdown of pure data transfers, overlapping transfers with computation, and pure computation for the OPT-30B model using FlexGen's double-buffering technique with single pipeline and the hardware configuration given in Section 4. PRE and GEN denote prefill and generation, respectively, and the number of batches is shown within the bracket. The prefill and generation phases exhibit different characteristics. Both are dominated by the data transfer overhead when the batch size is small. Only the prefill phase is quickly dominated by the computation latency as the number of batches increases. The time when computation can be overlapped with data transfers is shown in gray. We observe that the GPU computation latency is too short to hide the data transfer latency during the generation phase, leading to the generation phase being dominated by the latency of the data transfers.

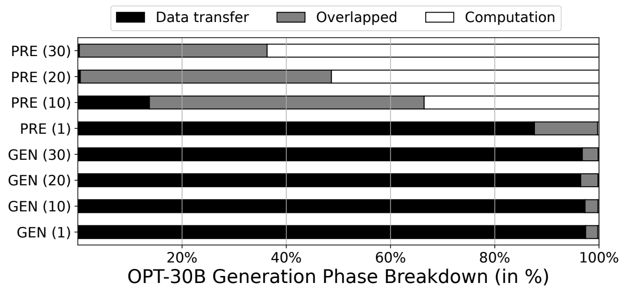


Figure 2: FlexGen OPT-30B workload breakdown. PRE and GEN denotes prefill and generation respectively, and the number in the bracket denoted number of batches.

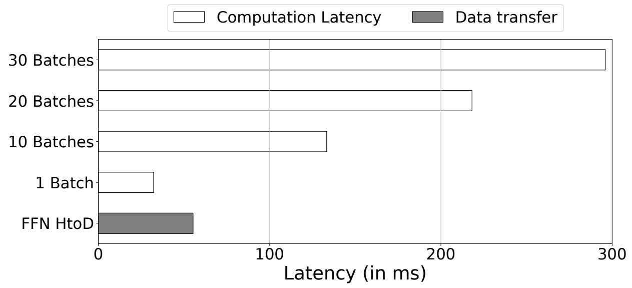
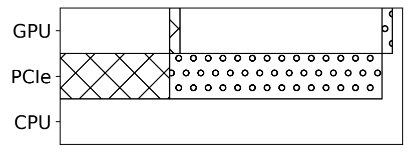


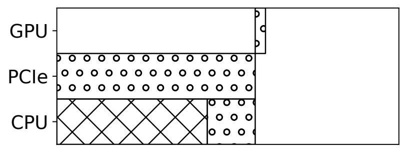
Figure 3: Latency comparison between multi-head attention computation latency with CPUs and feed-forward network weight transfer latency.

The observations of FlexGen's latency breakdown provide the motivation for this work, namely, to offload parts of the computation to the CPUs to decrease the latency of the generation phase. Since the weights of the key-value cache are not small and increase with the number of batches, we selected multi-head attention to be offloaded to the CPUs to decrease the data transfer overhead and the VRAM requirements to store the key-value cache.

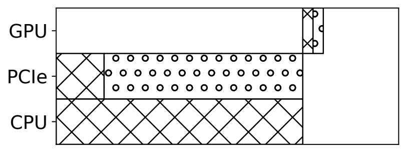
The amount of computation offloaded to the CPUs must be carefully chosen since the CPUs are orders of magnitudes slower than the GPU. An inefficient work



(a) GPU-only workload in FlexGen.



(b) Small-batch overlap scenario.



(c) Large-batch overlap scenario.

Figure 4: Gantt charts comparing GPU-only workload scenarios with CPU-GPU overlapping workload scenarios. Crosses represent the weight transfers and the computations of multi-head attention, and circles represent the weight transfers and the computations of the feed-forward network.

division could end up being worse than the original implementation. To this end, we compare the CPU computation latency of the multi-head attention against the feed-forward network weight transfers using the same hardware settings as described in Section 4. Figure 3 shows that the multi-head attention latency executed on the CPUs is smaller than the feed-forward network data transfers latency only for a small number of batches. Thus, to overlap computations with communication as much as possible, it is imperative to offload some feed-forward network workload to the CPUs during small-batch inference, and vice versa in large-batch inference.

Implementation. Figure 4 shows the possible two scenarios of overlapping CPU computations and data transfers compared to the original workload carried out by FlexGen. The core idea is to overlap the computation of multi-head attention in the CPUs with the data transfer of the feed-forward network to the GPU since the multi-head attention workload is smaller than that of the feed-forward network's. In a small-batch scenario, the multi-head attention computation in the CPU can be faster than the feed-forward weight transfer latency, so we can push the overlap further by breaking up the first matrix multiplication of the feed-forward network and computing it as well during the data transfer. The weights are divided along the columns, and the output computed in the CPU is concatenated with the output computed in the GPU. Breaking up and computing parts of the feed-forward network in CPUs can result in smaller data transfers to the GPU and also fewer computations inside the GPU.

However, in large-batch scenarios, the multi-head attention computation can take longer than the data transfers of the feed-forward network, which would result in the CPU performance becoming the bottleneck. Instead of breaking up the feed-forward network, we must divide the multi-head attention layer workload, send part of the multi-head attention weights in addition to the feed-forward networks', and overlap the computation with the added data transfers as well. In this work, we send the weights used in the last or the last two matrix multiplications of the multi-head attention to the GPU depending on the workload, and overlap communication with computation as much as possible without having the CPU computation become the bottleneck and increase GPU idle time.

The workload can be divided at different points depending on the size of the input in the generation phase, but not in the prefill phase. The generation phase using the key-value cache technique only takes one token per batch as an input, whereas the initial input consists of all input tokens making the computations heavier in the prefill phase. For example, FlexGen's benchmark' input token length is 512, i.e., the amount of computations in the prefill phase is 512 times larger than in the generation phase. Applying the offloading techniques of the generation to the prefill phase would lead to a severe drop in throughput, and thus we choose not to offload any work to the CPUs in the prefill phase.

Another consideration is that executing the prefill phase in the GPU may trigger data type conversions if the CPUs do not support FP16 precision since the model weights are stored in FP16 precision. We try to first reduce this cost by keeping all model weights in FP16 and whenever a multi-head attention operation finishes its prefill phase, the weights that are required in the CPU computations are converted from FP16 to FP32. This procedure is also applied to the newly-generated key-value cache so that it can be later used during the generation phase with the CPU. Note that this limitation can be avoided with a modern CPUs that support FP16.

4 EVALUATION

Configuration. We have implemented the presented technique into FlexGen [13]. FlexGen is based on PyTorch [12]. We use an NVIDIA RTX 2080 Ti with an Intel Core i5-10400 CPU for this evaluation. The workload is FlexGen's default workload with 512 input tokens

Batch Size	FlexGen (0-100)		FlexGen (20-80)		CPU Offload	
	Throughput (tokens/s)	Latency PRE + GEN (s)	Throughput (tokens/s)	Latency PRE + GEN (s)	Throughput (tokens/s)	Latency PRE + GEN (s)
1	0.204	5.1 + 151.6	0.213	4.8 + 145.1	0.328	5.3 + 92.1
10	1.711	8.2 + 178.8	1.825	8.1 + 167.2	1.928	12.3 + 153.6
20	2.931	12.4 + 205.9	3.106	12.4 + 193.6	2.818	19.7 + 207.4
30	3.827	18.5 + 232.3	4.011	18.5 + 220.8	3.761	25.2 + 230.1

Table 1: Throughput and latency comparison between FlexGen and CPU-offload-enabled inference.

and 32 output tokens generated per batch. We chose the OPT-30B model to test the presented technique because the model does not fit into the GPU's VRAM and thus leaves room for optimizations by overlapping the host-to-device data transfers with CPU computation in the inference process.

Results. Table 1 shows the result of running 1 to 30 batches for two different FlexGen memory configuration and our work. (0-100) and (20-80) denote the configuration used during the FlexGen run and refer to the percentage of the weights being pinned in GPU and CPU, respectively. Our work, regardless of the batch size, pins all tensors (weights, key-value cache, and intermediate activation) in the DRAM. We measure the throughput and the latency of running a certain batch size. The latency columns show the latencies of the prefill and the generation phase.

We observe a latency increase in all prefill phases when using the CPU offloading technique caused by the data type conversion of the computed key-value cache. Larger batches create a bigger key-value cache, thus the gap between FlexGen's prefill phase and ours approach increases linearly. Overall, we observe a 40% improvement with a batch size of 1 thanks to offloading the multi-head attention and parts of the feed-forward network to the CPUs. However, with larger batches, multi-head attention computations are only partially overlapped with data transfers due to CPU performance limitations and result in similar performance of the presented approach to FlexGen. We believe that 18% reduced PCIe bandwidth throughput caused by memory bandwidth contention is the root cause of this problem, prolonging both the data transfers and the CPU computation latency. Improving the PCIe bandwidth usage and CPU inference latency would result in higher throughput in huge batches as well, and we leave this for future work.

5 CONCLUSION

Large language models are widely applied due to their state-of-the-art performance, and their tasks may require them to be both latency-sensitive and throughput-oriented. Recent throughput-oriented frameworks on a single commodity GPU are dominated by data transfers during their inference process, therefore presenting an opportunity to overlap CPU computations with GPU data transfers. In this paper, we showed that efficiently overlapping CPU computations with host-to-device data transfers can increase overall throughput. Whilst our work shows a promising improvement of around 40% for small batches, there are some areas where we can further improve to gain more performance with larger batches, and potentially for pipelined workloads within a single-GPU as well.

REFERENCES

[1] [n. d.]. HuggingFace Accelerate. https://huggingface.co/docs/accelerate/index

[2] David Ahmedt-Aristizabal, Mohammad Ali Armin, Simon Denman, Clinton Fookes, and Lars Petersson. 2021. Graph-based deep learning for medical diagnosis and analysis: past, present and future. *Sensors* 21, 14 (2021), 4758.

[3] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, et al. 2022. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale. *arXiv preprint arXiv:2207.00032* (2022).

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).

[5] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* (2021).

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860* (2019).

[8] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378* (2023).

[9] Wookey Lee, Jessica Jiwon Seong, Busra Ozlu, Bong Sup Shim, Azizbek Marakhimov, and Suan Lee. 2021. Biosignal sensors and deep learning-based speech recognition: A review. *Sensors* 21, 4 (2021), 1399.

[10] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[11] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[13] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E Gonzalez, et al. 2023. High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865* (2023).

[14] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[15] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[16] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[18] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).