Inference Performance Optimization for Large Language Models on CPUs

Pujiang He [1] **Shan Zhou** [1] **Wenhuan Huang** [1] **Changqing Li** [1] **Duyi Wang** [1] **Bin Guo** [1] **Chen Meng** [1] **Sheng Gui** [1] **Weifei Yu** [1] **Yi Xie** [1]

Abstract

Large language models (LLMs) have shown exceptional performance and vast potential across diverse tasks. However, the deployment of LLMs with high performance in low-resource environments has garnered significant attention in the industry. When GPU hardware resources are limited, we can explore alternative options on CPUs. To mitigate the financial burden and alleviate constraints imposed by hardware resources, optimizing inference performance is necessary. In this paper, we introduce an easily deployable inference performance optimization solution aimed at accelerating LLMs on CPUs. In this solution, we implement an effective way to reduce the KV cache size while ensuring precision. We propose a distributed inference optimization approach and implement it based on oneAPI Collective Communications Library. Furthermore, we propose optimization approaches for LLMs on CPU, and conduct tailored optimizations for the most commonly used models. The code is open-sourced at https://github.com/intel/xFasterTransformer.

1. Introduction

Large language models (LLMs) based on the Transformer Vaswani et al. (2017) architecture have garnered profound technical attention globally and achieved remarkable accomplishments (Touvron et al., 2023) , (Zhang et al., 2022), (Yang et al., 2023), (Wu et al., 2023), (Bai et al., 2023). Their robust comprehension and generation capabilities are profoundly changing applications of artificial intelligence (Thirunavukarasu et al., 2023). However, the practical deployment of LLMs is significantly hindered by the high cost and resource limitations of hardware (Zhao et al., 2023). Therefore, deploying of LLM with good performance for practical applications has become a trending topic in industry, which helps land LLMs in more practical applications. When GPU hardware resources are limited, we can explore alternative options on CPUs.

Optimizing the practical deployment performance of LLMs necessitates effectively leveraging hardware capabilities for cost efficiency and improved inference performance, which, in turn, demands that developers possess a strong understanding of both hardware and software. To tackle the practical deployment challenge, we propose an easy and efficient solution designed to facilitate easy deployment on CPUs. Deploying on CPUs offers the advantage of being unrestricted by VRAM size, preventing KV cache overflow, and enabling the processing of extremely long-context support (Liu et al., 2023). Furthermore, deployment on CPUs can enhance system resource utilization, making multitasking more efficient.

The paper introduces the solution for efficient deployment and inference performance optimization for LLMs on CPUs. The solution supports for widely used LLMs, and experiment results show the proposed solution has good inference performance scalability on CPUs. We have established a repository where we curate relevant optimization with real-time updates. The main contributions are as follows:

- We propose new LLM optimize solutions on CPUs, such as SlimAttention. We conduct individual optimizations for LLM operations and layers, and the support extends to widely used LLMs, encompassing Qwen, Llama, ChatGLM , Baichuan, and Opt series.
- We implement an effective way to reduce the KV cache size and ensure precision. This approach allows for a more efficient use of memory without significantly compromising the quality of the model's output.
- We design a distributed inference optimization solution for LLMs on CPUs, facilitating the attainment of necessary scalability and efficient low-latency inference.

[1]Intel Corporation, Shanghai, China. Correspondence to: Pujiang He <pujiang.he@intel.com>, Shan Zhou <shan.zhou@intel.com>, Wenhuan Huang <wenhuan.huang@intel.com>.

Published at ICML 2024 Workshop on Foundation Models in the Wild. Copyright 2024 by the author(s).

2. Approach

In this section, three main optimization approaches are proposed and more details could be referred to the following sections.

2.1. LLM Optimization

To improve the inference performance, we proposed individual optimize solutions for each LLM operations and layers. Taking attention layer as an example. Since the resources consumed by the attention mechanism are directly proportional to the square of the sequence length from theoretical, optimizing attention is particularly important for long sequence inputs.

A new approach we called SlimAttention shown in Figure 1 is proposed. SlimAttention is essentially a one-dimensional decomposition of the score between query and key. In terms of computation sequence, it involves first calculating a horizontal score, followed by applying softmax to the score. The resulting softmax is then multiplied by the corresponding values, producing a portion of the output. This process is repeated for the next block using the same score buffer. In theory, each thread needs only to maintain a buffer of the block size, thereby reducing memory usage and enhancing computational efficiency in practice. And FlashAttention (Dao et al., 2022) is a solution initially introduced on GPUs, shown in Figure 2. Essentially, it involves a two-dimensional decomposition of the score. In the computational process, each thread only needs to maintain a single tile. However, as a tile doesn't encompass all the data in the softmax direction, there is a need for iterative corrections to the final results during the calculation.

Compared with FlashAttention, SlimAttention entails no redundant computations but does necessitate a larger intermediate buffer.

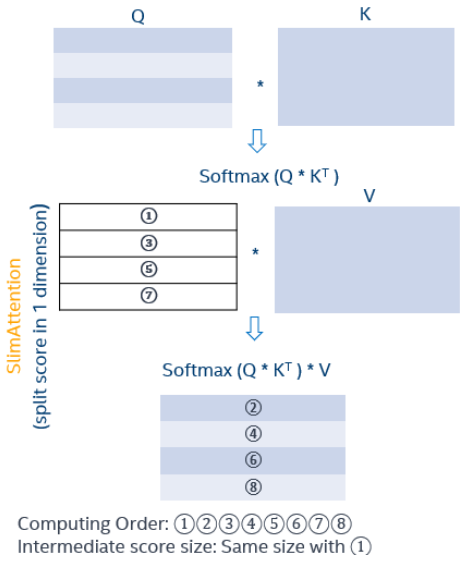

Figure 1: Slim Attention

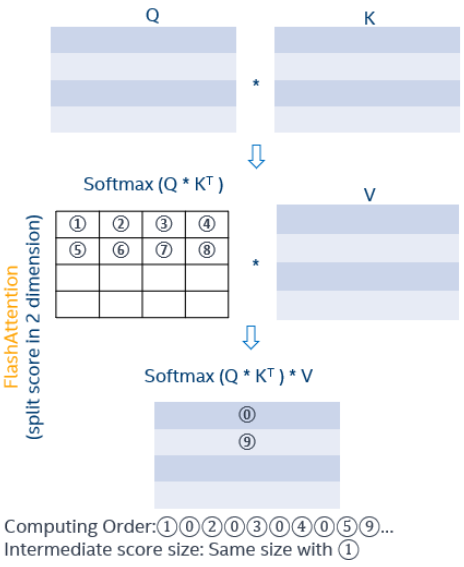

Figure 2: Flash Attention

2.2. Effective KV cache Optimization

Typical LLM architectures are predominantly decoder-only structures. Within these models, computation is primarily divided into two categories: attention (MHA/GQA) and matrix multiplication (MatMul), the latter often including post-operations that can be fused into the MatMul process. During the generation of the first token, both attention and MatMul operations are compute-bound. However, for the generation of subsequent tokens (beyond the first token), the attention typically exhibits a pattern of general matrix-vector multiplication (gemv), which is bound in memory bandwidth.

The volume of the key-value (KV) cache accessed during the generation of a single token can be calculated as follows:

$$2b(L_i + L_o)ln_{head}s_{head}s_d \quad (1)$$

where the b is batch size, L_i and L_o are input sequence length and output sequence length, n_{head}, s_{head} l, an ds_d respectively represents head number, head size, layer, and size of the data type, and factor of 2 accounts for both the key and value components.

Consider the Llama2-7B model as an illustrative example, which is characterized by the following parameters: head_number=32, head_size=128, and layers=32. Assuming a batch size of 256, an input sequence length of 1024, an output sequence length of 1024, and the use of FP16/BF16 data types for both weights and the KV cache, the maximum KV cache required is approximately 128GB. In contrast, the weight data that needs to be accessed is only around 14GB. This comparison underscores the significance of the KV cache size during inference with large batch sizes.

To enhance performance by effectively reducing the KV cache size, we have implemented an INT8 KV cache approach. To ensure that the INT8 representation closely approximates higher precision data types, we maintain a unique scale for each token and each head as shown in Figure 3. In this method, each head has a unique scale value, which could regulate the KV cache in a fine grained pattern, while with acceptable storage increasing. This approach allows for a more efficient use of memory without significantly compromising the quality of the model's output. Moreover, we engineered a custom kernel capable of supporting MatMul operations with hybrid data types. This kernel is adept at handling INT8 data, which it dynamically converts to FP32 during execution, as shown in Figure 4. This conversion process is integral to leveraging the Fused Multiply-Add(FMA) instructions facilitated by the AVX512 instruction set, a feature of the most recent x86 architectures. The conversion from INT8 to FP32 is a two-step process: initially, the _mm512_cvtepi8_epi32 intrinsic function (Intel, 2024) transforms INT8 into INT32, followed by the _mm512_cvtepi32_ps function, which then converts the INT32 data into FP32 format. This approach ensures efficient utilization of the available instruction set for optimized computational throughput.

	Head 0	Head 1	Head 2	Head 3
Sequence position 0	Scale_00	Scale_01	Scale_02	Scale_03
Sequence position 1	Scale_10	Scale_11	Scale_12	Scale_13
Sequence position 2	Scale_20	Scale_21	Scale_22	Scale_23
Sequence position 3	Scale_30	Scale_31	Scale_32	Scale_33
Sequence position 4	Scale_40	Scale_41	Scale_42	Scale_43
Sequence position 5	Scale_50	Scale_51	Scale_52	Scale_53

Figure 3: Example of maintained unique scale.

2.3. Distributed Inference Optimization

To enhance distributed inference performance, we present a solution for optimizing distributed inference for LLMs on CPUs. This solution is implemented using the oneAPI Collective Communications Library (oneCCL).

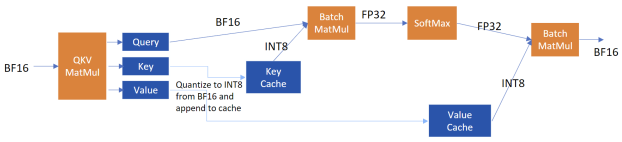

Figure 4: Custom kernel workflow.

In the proposed solution, our solution broadcasts token IDs instead of broadcasting the values of the Embedding part obtained based on token IDs. We perform the reduction after each worker computes the top-k, rather than directly reducing the logits of all tokens. The implementation is shown in Figure 5 .

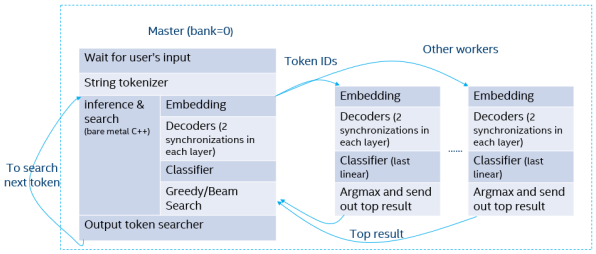

Figure 5: Distributed inference based on oneCCL.

Moreover, an aggressive optimization approach was proposed for reduce data copying since we found that when the computation module and communication module interact, data copying is often involved in practice. This involves the computation module, during its last operation before communication, directly writing the results to the location of the communication module, achieving a zero-copy implementation which shown in Figure 6.

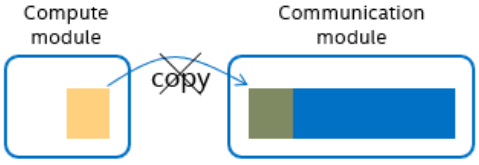

Figure 6: A zero-copy implementation

Table 1: Hardware configuration

HEADINGS	CONFIG
CPU Model	Intel Xeon CPU 8563C
Sockets	2
Cores per Socket	52
Base Frequency	2.6GHz
All-core Max Frequency	3.1GHz

Table 2: Next token generation latency of Llama2-70B with the proposed distributed solution

SOCKET	LATENCY
2	249.7 ms
8	87.7 ms

3. Experiment Results

3.1. Configuration

We conducted experiments on the Intel® Xeon® CPU 8563C, and detailed information of one bare-mental machine is provided in Table 1. Note that each machine has 2 sockets.

3.2. Performance

We measure the throughput and latency of next token generation on Intel® Xeon® CPU 8563C by adopting the proposed solution.

Table 2 shows next token generation latency results of the Llama2 (Touvron et al., 2023) with 70B parameters under a model configuration with input tokens = 1024, output tokens = 128, batch size = 1. The latency result is better when it is lower. With 4 bare-mental machines (8 sockets) running the proposed distributed solution, it could bring 2.85X performance gain for Llama2-70B compared with 1 bare-mental machine (2 sockets).

Table 3 shows the performance comparison of SlimAttention and FlashAttention on 1 socket. In table 3, 1st column is input length, 2nd column is the performance of FlashAttention while 3rd column is the performance of the proposed SlimAttention. Note that performance refers to the average cost time (ms) of each attention layer during the generation of the first token with configuration of Llama2-7B model when batch size = 1. It shows the proposed SlimAttention approach could achieve better performance on CPU.

Table 4 shows throughput without first token of Llama2-7B on 1 socket, under a model configuration with input tokens = 148, output tokens = 198, and the batch size is 256/512. Throughput result is better when it is higher. The proposed solution demonstrates its potential to enhance throughput effectively.

INPUT	FLASH	PROPOSED SLIM
256	10.85	1.10
512	27.95	6.60
1024	61.57	16.02
2048	176.36	96.65
4096	540.14	392.80

Table 3: Performance comparison between FlashAttention and SlimAttention for different input token sizes.

Table 4: Throughput without first token of Llama2-7B

BATCH SIZE	LATENCY
256	796.9 tokens/s
512	853.6 tokens/s

4. Conclusion and Future Work

We presented an end-to-end LLM inference accelerating solution including distributed inference optimization, effective KV cache optimization, and individual LLM optimization. We showcased the versatility across a range of popular LLMs and the performance superiority over the open-source solution on CPUs. In our forthcoming research, we intend to broaden our study to include a wider variety of CPUs, particularly those with resource constraints. Our primary focus will be on enhancing performance for larger batch sizes and exploring effective deployment serving solutions. Concurrently, we aim to adapt our solution to accommodate the latest trending models, such as the mixture of experts (MoE) models. Our goal is to offer a practical alternative to existing GPU solutions.

References

Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

Intel, R. Intrinsics guide, 2024. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., and Liang, P. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Thirunavukarasu, A. J., Ting, D. S. J., Elangovan, K., Gutierrez, L., Tan, T. F., and Ting, D. S. W. Large language models in medicine. *Nature medicine*, 29(8):1930–1940, 2023.

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

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

Wu, T., He, S., Liu, J., Sun, S., Liu, K., Han, Q.-L., and Tang, Y. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023.

Yang, A., Xiao, B., Wang, B., Zhang, B., Bian, C., Yin, C., Lv, C., Pan, D., Wang, D., Yan, D., et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023.

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

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.