An Efficient GEMM Acceleration Method for LLM Inference with Variable-Length Sequences

Yu Zhang
yuzhang0722@163.com
School of Compute Science and Engineering
South China University of Technology
GuangZhou, Guangdong, China

Lu Lu*
lul@scut.edu.cn
School of Compute Science and Engineering
South China University of Technology
Guangzhou, Guangdong, China
Pengcheng Laboratory
Shenzhen, Guangdong, China

Abstract

This work proposes an efficient GEMM acceleration method for LLM inference with variable-length sequences. First, a fused parallel prefix scan design is developed to capture the matrix dimension distribution. Second, an efficient various-size tile kernel is implemented based on Matrix Core, with an analysis of the hardware resource requirements in the computation process. Third, a hardware-aware tiling algorithm is designed to select the optimal tiling scheme based on thread parallelism and hardware resources. The experimental results show that the proposed approach achieves performance improvements of 3.10× and 2.99× (up to 4.44× and 4.27×) over hipBLAS and rocBLAS.

Keywords

LLM inference, Batch GEMM, Matrix Core, Tiling algorithm

1 Introduction

With LLM demonstrating impressive performance across various NLP domains, they have become capable of handling complex text-processing tasks such as machine translation [1], code generation [2], and human-computer interaction [3]. In previous studies, the batch-processed sequence is usually extended to a uniform length using a zero-padding method to avoid length inconsistencies between input sequences. In this way, multiple input sequences are mapped to the same length by setting max sequence length. Zero-padding and matrix computation in LLM inference are shown schematically in Fig. 1. Unfortunately, this method can lead to inefficiencies in computation and unnecessary memory usage during the batch inference process, which can impede the deployment of LLM.

2 Design and Methodology

2.1 Fused parallel prefix scan design

The padding mask matrix is a crucial component in the LLM inference process, particularly within the SA module. This matrix regulates the model's access to input sequences, ensuring that specific rules are followed during inference. The padding mask matrix indicates the length of each sequence, where 1 signifies valid data and 0 denotes a padding element. Our motivation is to determine the valid sequence length distribution (val_seq array) of the input sequences using the padding mask matrix representation. To avoid

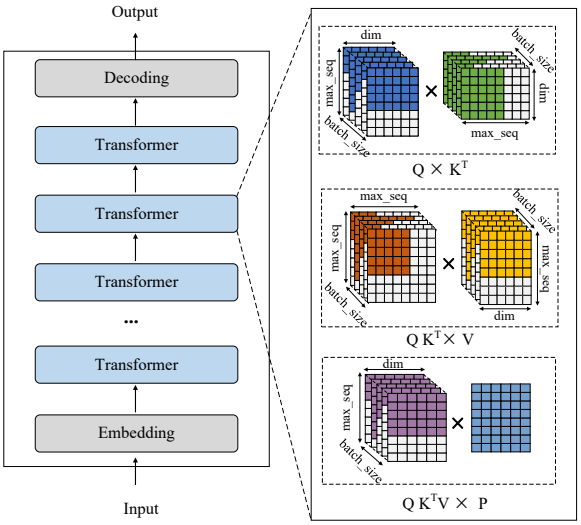


Figure 1: Illustration of batch GEMM in SA and MLP modules during LLM inference

invalid computation and bandwidth consumption, the matrix is segmented according to the val_seq array. Wavefront-level parallel accumulation with hardware intrinsics (e.g., __shfl_up) minimizes synchronization overhead, while kernel fusion reduces context-switching latency.

2.2 Variable-sized kernel design and performance analysis

In this section, we design five tiles with the specific parameter list shown in Table 1. Fig. 2 shows four categories of hardware resources (LDS, VGPR, SGPR, SCR) during kernel execution along with the corresponding performance data for each tile.Larger-size tiles tend to provide better intra-tile parallelism. As the tile size increases, the demand for hardware resources during the execution of each tile also increases. The performance of the "256*256*32" tile exhibits a significant, cliff-like drop compared to other tiles rather than an expected upward trend. This is primarily attributed to its heavy reliance on SCRs – hardware resources employed to manage register overflow. When a thread's workload exceeds the on-chip registers' capacity, the compiler temporarily utilizes SCRs to store overflow data. Accessing data from SCRs is significantly

*Corresponding author.

slower than accessing data directly from registers. Consequently, the "256*256*32" tile demonstrates poor performance in practical calculations.

Table 1: The parameter design choice in variable-sized tile

Index	Tile	T_m	T_n	T_k	W_m	W_n	W_k
0	small	64	64	32	1	1	8
1	small-medium	64	128	32	1	2	8
2	medium	128	128	32	2	2	8
3	medium-large	128	256	32	2	4	8
4	large	256	256	32	4	4	8

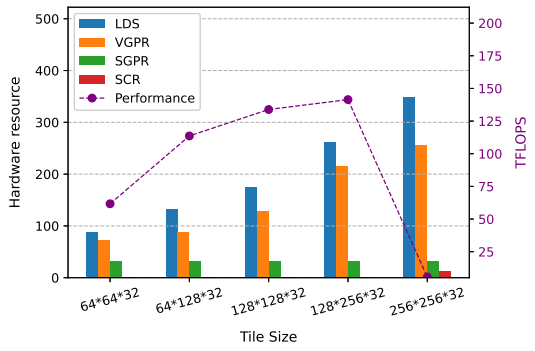


Figure 2: The variable-sized tile kernel performance and hardware resource overhead

2.3 Hardware-aware tiling algorithm

In this section, we utilize two primary criteria—Thread-Level Parallelism (TLP) performance and Hardware Resource Limitation (HRL)—to guide the selection of an optimal tiling strategy. In thread-level parallelism, TLP is used to quantize the number of threads performing computation tasks in parallel, which is usually related to tile size and the number of threads in the workgroup. For a given batch matrix and tiling scheme, the TLP can be computed as:

$$TLP = \sum_i \phi\left(\frac{M_i \times N_i}{T_{mi} \times T_{ni}}\right) \times T_{wavefront} \quad (1)$$

HRL is used to quantize the hardware resource requirements of threads during kernel execution, which involves the hardware resource requirements (LDS and VGPR) based on the tiling scheme. HRL can be computed as:

$$HRL = LDS_{used} + VGPR_{used} \quad (2)$$

3 Evaluation

In the comparison experiments, we evaluate two implementations (hipBLAS [4] and rocBLAS [5]) on MI210. Table 2 lists the experimental platform's key components and software versions. The experimental results in Fig. 3 show that the proposed method has a speedup of 3.10× and 2.99× (up to 4.44× and 4.27×) then hipBLAS and rocBLAS, respectively.

Table 2: The configuration of experimental platform

Experimental Platform	
CPU	AMD EPYC 7663 @3.50 GHz
GPU	AMD Instinct MI210 GPUs, 64GB RAM
ROCm	ROCm 6.1
Python	Python 3.9.19
OS	Ubuntu 22.04

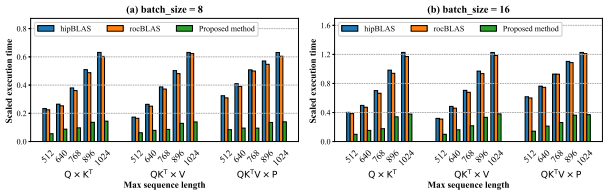
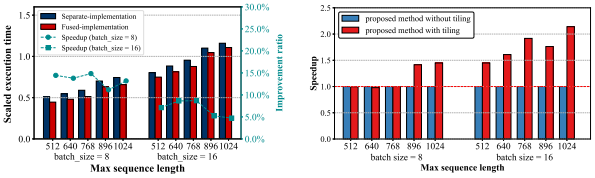


Figure 3: The average latency of $Q \times K^T$, $QK^T \times V$ and $QK^T V \times P$ functions. (average length = 0.6 × max sequence length)



(a) Fused kernel implementation (b) Tiling algorithm

Figure 4: The performance benefits of fused kernel implementation and tiling algorithm

Fig. 4 (a) show the proposed method with the fused-implementation has 1.16× and 1.06× performance improvement, respectively. As shown in Fig. 4 (b), the proposed tiling algorithm can achieve an average performance improvement of 1.17× and 1.78× when the batch size is 8 and 16, respectively, which proves the effectiveness of the proposed tiling algorithm.

Acknowledgments

This work was supported by the Natural Science Foundation of Guangdong Province (2024A1515010204) and the Technological Research Project of Southern Power Grid Company (ZBKJXM20232483).

References

[1] Dehong Gao, Kaidi Chen, Ben Chen, Huangyu Dai, Linbo Jin, Wen Jiang, Wei Ning, Shanqing Yu, Qi Xuan, Xiaoyan Cai, et al. 2024. LLMs-based machine translation for E-commerce. *Expert Systems with Applications* 258 (2024), 125087.

[2] Jianxun Wang and Yixiang Chen. 2023. A review on code generation with llms: Application and evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*. IEEE, 284–289.

[3] Dhruval Kenal Kothari and Owen Noel Newton Fernando. 2024. Enhancing Human-Computer Interaction through AI: A Study on ChatGPT in Educational Environments. In *2024 IEEE Conference on Artificial Intelligence (CAI)*. IEEE, 500–503.

[4] AMD ROCm™ Software. 2025. ROCm BLAS marshalling library. https://github.com/ROCm/hipBLAS.

[5] AMD. 2025. Next generation BLAS implementation for ROCm platform. https://github.com/ROCm/rocBLAS.