LLM4GV: An LLM-based Flexible Performance-Aware Framework for GEMM Verilog Generation

Dingyang Zou[1], Gaoche Zhang[1], Kairui Sun[2], Zhe Wen[2], Meiqi wang[2], Zhongfeng Wang[1,2]
[1] *Nanjing University, Nanjing, China;* [2] *Sun Yat-sen University, Shenzhen, China*
{*dyzou, gczhang*}*@smail.nju.edu.cn,* {*sunkr, wenzh9*}*@mail2.sysu.edu.cn*
wangmq53@mail.sysu.edu.cn, zfwang@nju.edu.cn

***Abstract*—Advancements in AI have increased the demand for specialized AI accelerators, with design for general matrix multiplication (GEMM) module being crucial but time-consuming. While large language models (LLMs) show promise for automating GEMM design, challenges arise from GEMM's vast design space and performance requirements. Existing LLM-based frameworks for RTL code generation often lack flexibility and performance awareness. To overcome the challenges, we propose LLM4GV, a multi-agent LLM-based framework that integrates hardware optimization techniques (HOTs) and performance modeling, improving correctness and performance of the generated code over prior works.**

I. Introduction

Recent AI advancements have increased the demand for specialized AI accelerators (AIA) to efficiently run models [1]. GEMM is a key module in AIA, and optimizing its design is vital for performance. However, designing tailored GEMM variants is complex and requires extensive hardware expertise. This highlights the need to automate the GEMM design process. Given the powerful capabilities of LLMs to generate high-quality content from human instructions and prompts, we propose leveraging LLMs to automate GEMM module design for AIA through in-context learning.

Many studies have proposed frameworks using LLMs for RTL code generation, but three challenges remain for GEMM module generation. First, GEMM's complexity makes prompt creation for LLMs labor-intensive, with limited code correctness. Second, most approaches rely on manually designed templates with minor LLM adjustments, restricting flexibility for diverse GEMM variants. Third, previous LLM-based frameworks struggle to evaluate the performance of complex modules like GEMM with large design space.

To tackle these challenges, this paper proposes a multi-agent LLM-based framework, LLM4GV, for generating high-quality Verilog code for GEMM. This framework enables the LLM to generate different variants of the GEMM module by enabling it to learn hardware optimization techniques (HOTs). Additionally, it incorporates performance awareness through modeling the design space and utilizing a tailored performance model for the LLM (LLMPM).

II. LLM4GV Framework

A. Multi-Agent Framework

To address the first challenge, we propose the LLM4GV multi-agent framework to decompose the complex task of generating GEMM modules, ensuring correctness and automation. The framework includes three agents: the top control agent (TCA), the performance model agent (PMA), and the code generation agent (CGA). By efficiently breaking down tasks, the agents enhance specific capabilities, leading to improved code correctness. The TCA manages the entire process through instruction-feedback control, directing code generation and ensuring correct feedback at each step. The PMA, using LLMPM, helps the LLM identify the optimal design from a large space. The CGA consists of sub-agents that generate code for sub-modules and handle module instantiation to connect them into the complete module.

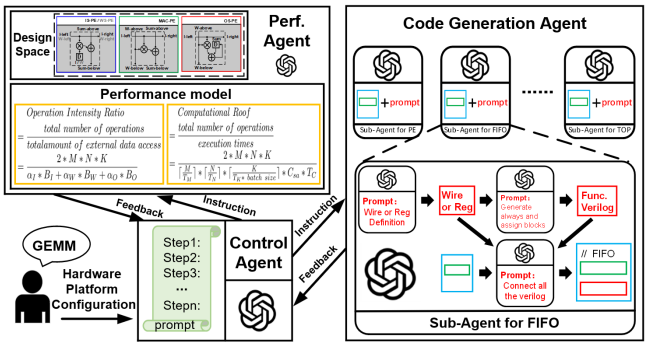

Fig. 1. Overall architecture of LLM4GV.

B. Teaching LLM Hardware Optimization Techniques

To address the second challenge, we teach the LLM how to use HOTs to design GEMM module variants based on a template. Due to token length limitations in LLMs, it's impractical to provide large resources like a hardware design book. Instead, we teach the LLM about the changes in Verilog code after applying HOTs. We construct a template-prompt incorporating CGA architecture, dividing the module into two parts based on HOTs' impact on code generation. For less-impacted parts, we use a template engineering method, making minor adjustments to the parameters of the highly parameterized template. For more-impacted parts, where errors are more likely to occur, we use prompt engineering to generate the critical code and connect it to the template via the module connection component in CGA or replace template code segments that are incorrect. The prompts for these operations are embedded in the TCA.

C. Teaching LLM to Be Aware of Performance for GEMM

To address the third challenge, we enhance LLM's performance-awareness with the proposed LLMPM (Fig. 2). We first decouple the GEMM design space using parameters that fully describe it, then model the performance of GEMM variants in this space. The design space is decoupled based on two principles: (1) parameters are independent, and (2) changing a parameter significantly affects GEMM performance. We

First author and second author contribute equally to this work. Corresponding authors: Meiqi Wang, Zhongfeng Wang.

978-3-9826741-0-0/DATE25/© 2025 EDAA

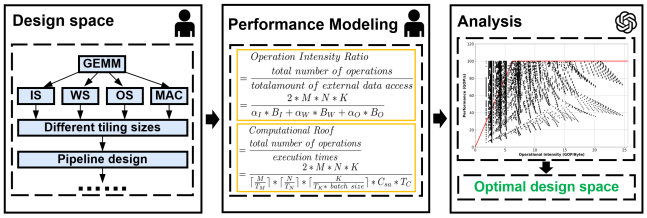

Fig. 2. Overview of teaching LLM to be aware of performance for GEMM.

design the performance model to cover the modeling from microarchitecture to system level to make LLM fully aware of performance. At the microarchitecture level, we establish the relationship between throughput and design parameters (e.g., bit width, pipeline stages) by analyzing array critical paths and incorporating DC synthesis data. At the system level, inspired by the roofline model [6], we use operation intensity (OI) and throughput as metrics. The relationship between OI and parameters like matrix dimensions, GEMM array size, and pipeline stages is shown in Fig. 2. The LLMPM is integrated into the framework via in-context learning with prompts and an internal code interpreter.

III. Experimental Results

In this section, we demonstrate the effectiveness of our LLM4GV framework for automatically generating the optimal Verilog code for GEMM. We use GPT-4 as the default baseline and evaluate the correctness and performance of the Verilog code generated by the frameworks with Synopsys Design Compiler and AMD (formerly Xilinx) Vivado.

To validate the effectiveness of our multi-agent framework with the proposed template and critical code prompt (TCCP) in code generation correctness for variants of GEMM, we design an ablation and comparative experiment with the following baseline and variants: GPT-4 (baseline), GPT-4 + function description prompt (FP), and GPT-4 + FP + TCCP. We also add the open-source LLMs-based frameworks CodeV-7B [4] and VeriGen-16B [3] as references for comparison. Table. I summarizes the quantitative evaluation of both syntax and functionality correctness of different LLMs and different methods. We show the number of syntax-correct and functionality-correct codes for each of the five codes, respectively.

To validate the effectiveness of the proposed LLMPM, we design an comparative experiment with the following baselines: GPT-4 + performance-aware prompt (PAP), GPT-4 + performance model (PM) in [5], Manual design, GPT-4 + LLMPM (ours). As shown in Fig. 3(a), we utilize the three commonly used neural networks as baselines to evaluate the performance of the GEMM code. To validate that our method can achieve the best performance under different hardware resource constraints and system requirements, we visualize the trade-off between DSPs and latency based on VGG-16 with an input resolution of 224*224*3, as shown in the Fig. 3(b). The results show that the code generated by our proposed framework demonstrates significant advantages in both correctness and performance.

IV. Conclusion

Recent works on LLMs for Verilog code generation show great potential but suffer from a lack of automation, correctness, flexibility, and awareness of performance when supporting GEMM modules. To address these problems, we propose a flexible performance-aware framework called LLM4GV, which utilizes in-context learning based on GPT-4 to generate the optimized Verilog code for the GEMM module with a large design space according to system requirements. In future work, we plan to develop a well-annotated, high-quality Verilog dataset of the GEMM module with this framework and use it to fine-tune an open-source LLM.

TABLE I
The Syntax and Functionality Correctness Verification for Different LLMs and Different Methods

GEMM module	VeriGen-16B [3]		CodeV-7B [4]		GPT-4 (baseline)		GPT4+FP		**GPT-4+FP +TCCP (ours)**	
	synt.*1	func.*2	synt.	func.	synt.	func.	synt.	func.	synt.	func.
PE Array-1	5	0	5	0	4	0	3	1	5	5
PE Array-2	5	0	5	0	2	0	3	0	5	5
PE Array-3	0	0	4	0	4	0	4	2	5	5
DataReOrganize-1	5	0	2	0	5	0	2	0	5	4
DataReOrganize-2	5	0	5	0	5	0	2	0	4	4
DataReOrganize-3	5	0	5	0	5	2	3	0	4	3
FIFO	0	0	1	0	3	0	1	0	5	5
Requantization	0	0	5	0	3	0	5	1	5	5
success rate	62.5%	0%	80%	0%	77.5%	5%	57.5%	10%	**95%**	**90%**

*1: The number of codes successfully compiled by DC in five trials.
*2: The number of codes that passed functional testing in five trials.

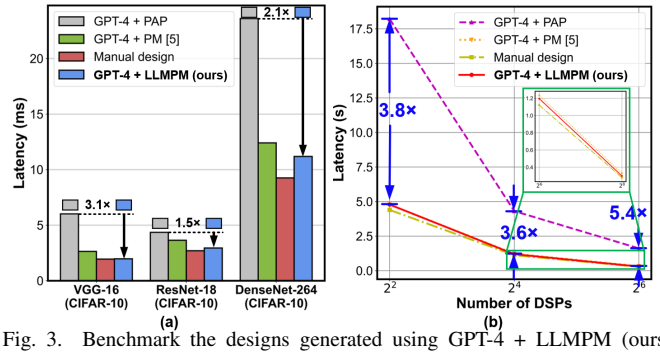

Fig. 3. Benchmark the designs generated using GPT-4 + LLMPM (ours) parameters against those generated using GPT-4 + PAP, GPT-4 + PM [5], and manually design. (a) For different models. (b) Under certain resource constraints.

V. Acknowledgement

This work was supported in part by the National Key R&D Program of China under Grant 2022YFB4400600 and the National Natural Science Foundation of China under Grant 62404256.

References

[1] W. Li, A. Hu, N. Xu and G. He, "Quantization and Hardware Architecture Co-Design for Matrix-Vector Multiplications of Large Language Models," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 71, no. 6, pp. 2858-2871, June 2024.

[2] Fu, Yonggan, et al. "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models." 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD). IEEE, 2023.

[3] Thakur, Shailja, et al. "Verigen: A large language model for verilog code generation." ACM Transactions on Design Automation of Electronic Systems 29.3 (2024): 1-31.

[4] Zhao, Yang, et al. "CodeV: Empowering LLMs for Verilog Generation through Multi-Level Summarization." arXiv preprint arXiv:2407.10424 (2024).

[5] C. Zhang, et al. "Optimizing FPGA-based accelerator design for deep convolutional neural networks," 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Monterey, CA, USA, 2015, pp. 161–170.

[6] S. Williams, A. Waterman, and D. Patterson. Roofline: An insightful visual performance model for multicore architectures. Commun. ACM, 52(4):65–76, Apr. 2009.