Immediate Communication for Distributed AI Tasks

Jihao Xin*†
jihao.xin@kaust.edu.sa
KAUST

Seongjong Bae*
sjbae1999@snu.ac.kr
Seoul National University

KyoungSoo Park
kyoungsoo93@snu.ac.kr
Seoul National University

Marco Canini
marco@kaust.edu.sa
KAUST

Changho Hwang
changhohwang@microsoft.com
Microsoft Research

1 Introduction

Recent advancements in AI, driven by large models like ChatGPT [18] and SORA [19], have led to significant computational challenges. Scaling these models often requires multi-GPU or multi-node systems [2, 14], utilizing parallelization strategies like tensor parallelism [25] to handle the computational load. For example, the Llama 3.1-405B model training uses 16,000 H100 GPUs [16]. However, distributed computation introduces communication as a major bottleneck, accounting for up to 80% of the execution time, as seen with the Llama 2-7B model [1]. As demonstrated in [3], scaling the Llama 2-13B [27] training from 8 to 1,024 GPUs drastically reduces Model FLOPs Utilization (MFU) from 47% to 4% due to communication overhead. This highlights a key issue: despite advancements in hardware capabilities, the hardware (particularly GPUs), often remains underutilized due to the introduced communication overhead.

To increase the MFU, previous studies have explored the potential of pipelining computation with communication [20, 22, 28, 30] or data loading [9] to boost hardware utilization. However, these strategies mainly focus on overlapping computation operators with independent communication operators. If dependencies exist, for example, in the inference phase, both computation and communication are on the critical path, where the inter-operator overlapping is unfeasible.

Recognizing this opportunity, we introduce **DistFuse**, a system that facilitates fine-grained overlapping, even in the presence of dependencies. At its core, DistFuse aims to coordinate computation and communication such that GPU initiates communication as soon as a part of the data is ready instead of waiting for the entire data.

We conduct a proof-of-concept experiment to show the performance gain by applying DistFuse with Llama 3-70B's inference on a single node that can hide up to 44.3% communication latency. Our current prototype focuses on LLM tasks, but the core concept of immediate communication is versatile and can be applied to other scenarios such as convolutional models. Given the increasing prevalence of large model workloads in data centers and the growing demand for efficient communication, we anticipate significant performance gains through our technique. In addition, we are applying our approach to vLLM [13]. Only by substituting the individual kernel without overlapping, we can achieve 5.2% to 11.1% speedup in Llama 3's end-to-end inference latency. The speedup is attributed to better spatial locality and more efficient block usage than the original vLLM implementation, and we are working on the overlapping integration.

2 Background

Distributed LLM. We briefly introduce the Llama 3-70B model architecture and how communication happens. LLMs primarily rely on General Matrix Multiplication (GeMM) operations, with model weights and activations distributed across multiple GPUs due to memory limitations. Figure 1 shows the Llama 3-70B model inference workflow, exemplifying that GeMM is the dominating operator. Each colored box represents a GeMM operation, while the dashed boxes indicate that GeMM is distributed across GPUs, where an All-reduce or All-gather is performed after the GeMM.

Inter-operator Overlapping. Existing studies mainly focus on computation and communication overlapping across independent operators, such as gradient calculations and communications in adjacent layers. A common solution is the use of execution schedulers [8, 15, 20, 22], which introduces lightweight layers between ML frameworks and the execution engine. On the other hand, compilation-based methods [4, 9, 11, 23, 24, 26, 29] enhance execution speed by optimizing model operational graphs after training. Among related works, CoCoNet [10] and T3 [21] are the most closely aligned with fine-grained intra-operator overlapping. CoCoNet introduces an additional scheduler and requires users to code in a DSL, while T3 depends on specialized hardware. In contrast, DistFuse can be seamlessly integrated into frameworks like PyTorch without user intervention or hardware support, offering a more practical and accessible solution. And we aim to achieve significant speedup by leveraging caching effects through overlapping dependent kernels.

GPU Communication. Traditionally, communication between GPUs involves the CPU and uses the PCIe interface, resulting in kernel overhead. Nvidia's GPUDirect technology enables GPUs to access peer memory using a unified memory address space without involving the CPU. For local GPUs, data is transferred directly via NVLink; for remote machines, GPUDirect leverages RDMA to facilitate efficient data transfer, reducing the latency by bypassing the CPU and

*Both authors contributed equally to this research.
†This work is primarily completed during internship at Microsoft Research.

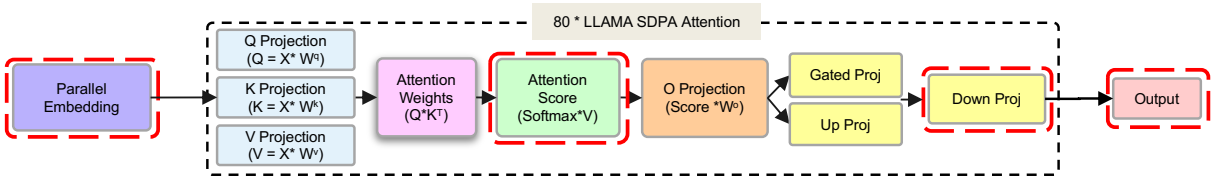

Figure 1. Llama 3-70B Model Architecture. Dashed boxes indicate distributed GeMM.

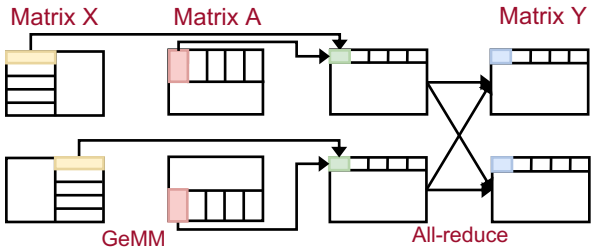

Figure 2. Concept of Fused Tile-wise GeMM & All-reduce of $X \cdot A = Y$ on 2 GPUs

enabling a more flexible programming model. At the software level, customizing the All-reduce kernel with NCCL [6] has been challenging due to its latency and complexity, while recent solutions like TensorRT-LLM [7] offer more portable communication CUDA kernels that we can adapt.

3 Immediate Communication

The core idea of DistFuse is to **trigger communication immediately, rather than waiting for the entire computation to finish**. In this paper, we use distributed GeMM with All-reduce as an example to illustrate our idea due to its prevalence in today's LLM applications. GeMM operations can involve very large dimensions. For example, the Down Projection GeMM in Llama 3-70B has dimensions [M=1,024, N=8,192, K=28,672], and the operation is usually spread across multiple GPUs using Tensor Parallelism. Each GPU handles a portion of the input, performs local computations, and then runs an All-reduce to communicate the results after all local computations are done. The inter-operator overlapping cannot be applied here due to dependency.

We observe that, although GPU utilizes the SIMD (Single Instruction Multiple Data) programming model, the hardware does not compute all data synchronously. In practice, some outputs are generated earlier than others. Leveraging this, DistFuse starts the communication as soon as parts of the data are ready, to enable fine-grained overlapping with remaining computation. To further elaborate on the details, the following section outlines the two main challenges we encounter and the strategies we employ to overcome them.

3.1 Granularity

The first challenge is determining the optimal data size to trigger communication. While triggering communication for each data element is straightforward and could maximize latency hiding, it introduces significant initiating communication overhead and requires rewriting the computation

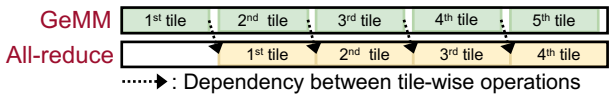

Figure 3. Overlapping at Fused Tile-wise GeMM & All-reduce Pipelining within a SM

kernels. To address this, we propose triggering communication for each tile. A "tile" refers to a sub-matrix of the original GeMM. Figure 2 illustrates the tile-wise distributed GeMM followed by a tile-wise All-reduce.

The tile-wise method is widely used for GeMM in off-the-shelf libraries, and we adopt the tile-wise GeMM kernel from CUTLASS [5], one of the open-sourced back-ends of PyTorch. However, tile-wise operation is rarely applied to communication libraries like NCCL, which uses contiguous buffers for sequential communication. Therefore, we develop a new tile-wise communication library for DistFuse based on GPUDirect P2P access. The tile-wise All-reduce splits the original All-reduce into multiple collective calls and handles non-contiguous memory. Additionally, we ensure the tile widths align with the GPU cache line size (128 bytes for A100 GPUs) to reduce access overhead. By adopting this pattern, we verify our tile-wise All-reduce achieves performance on par with NCCL's optimized All-reduce kernels.

By applying tile-wise GeMM and All-reduce, we achieve overlapping across tiles in a Streaming Multiprocessor (SM). Tile tasks are distributed among SMs; each SM completes the GeMM operation for a tile and then begins its communication phase while starting the next tile's GeMM, creating an operation pipeline. Traditional All-reduce waits until all GeMM finishes, but as shown in Figure 3, our tile-wise All-reduce transmits partial results as soon as the first tile is done. Tile-wise overlapping requires each SM to be allocated with more than one tile. For instance, on an Nvidia A100 GPU with 108 SMs, overlapping occurs only when more than 108 CTAs are launched. DistFuse by default launches one dedicated CTA (Cooperative Thread Array) per tile for large GeMM, and applies Split-K [12] to split a tile into multiple CTAs, balancing the workload across SMs.

3.2 Hardware Scheduling

The second is an engineering challenge: the compiler often fails to generate a binary that executes operations simultaneously, even if the operations are obviously parallel and use different types of hardware resources. The most straightforward method to achieve tile-wise overlapping is to fuse GeMM and All-reduce into a single kernel, where each CTA

GeMM Size [M,N,K]	PyTorch GeMM	PyTorch All-reduce	DistFuse GeMM	DistFuse All-reduce	PyTorch	DistFuse w/o Overlap	DistFuse
[1024,8192,8192]	166.0us	221.0us	168.9us	234.5us	387.0us(100%)	403.4us(95.9%)	333.8us(115.9%)
[1024,8192,28112]	534.5us	221.0us	536.6us	234.5us	764.5us(100%)	771.1us(99.1%)	621.6us(123.0%)

Table 1. Llama 3 Attention block's Distributed GeMM Layers. [M, N] is the output size, K is the middle dimension.

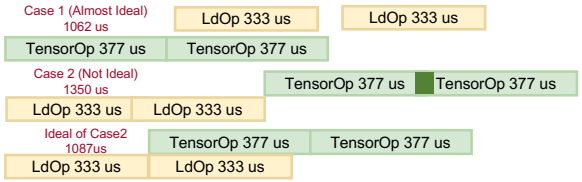

Figure 4. Random Starting of Overlapping[1].

first computes a GeMM tile and then performs an All-reduce. However, this approach has inherent issues.

To illustrate the issue, we conduct the following experiments. We first define two operations: TensorOps (Matrix Multiply-Accumulate instructions using Tensor Cores) and LdOps (global memory reads). Although these operations can overlap when executed separately, fusing them into one kernel revealed inconsistencies. We test two cases: **Case 1** where each CTA executes TensorOps followed by LdOps, and **Case 2** where the order of TensorOps and LdOps is reversed. There are no data dependencies between TensorOps and LdOps, and we have enabled all compiler optimizations. Ideally, with two fused operators launched per SM, both cases can achieve certain overlapping. However, as shown in Figure 4, only Case 1 achieves ideal overlapping, while Case 2 results in no overlap. We confirm this issue consistently exists in other tasks and hardware (tested with V100, A100, and H100). We check the compiled SASS code and suspect that this is rooted in the compiler and hardware scheduler which is transparent to end-users.

To overcome this limitation, we propose adding an **explicit synchronization barrier** for each tile and launching GeMM and All-reduce in two streams with a shared buffer. Once a GeMM tile is ready, it will set the corresponding flag in the buffer, while the All-reduce will busy-wait until the flag is set. This empirical solution consistently ensures the desired overlapping for all cases and hardware we tested.

3.3 Preliminary Evaluation

Model. We test DistFuse on Llama 3-70B's inference (Figure 1), one of the most popular publicly accessible LLMs. Our parallelism follows the strategy specified in Meta's source code [17]. Within each attention block, both the Attention Score and the Down Projection layers employ distributed GeMM, utilizing row parallel GeMM in conjunction with All-reduce. Additionally, the initial embedding and final output layer utilize column parallel GeMM coupled with All-gather.
Hardware We conduct our evaluation in one ASUS ESC N4A-E11 server that runs Ubuntu 22.04 with CUDA 12.1, and PyTorch 2.1.2, equipped with 4 NVIDIA A100-40GB GPUs. GPUs are P2P connected by NVLink.
Speedup. Two layers in Llama 3's attention block require communication, with 8.43% to 25.05% of inference latency spent on it. Table 1 compares DistFuse and PyTorch for these layers, with the first row showing the Attention Score layer and the second the Down Projection layer. Our evaluation confirms that DistFuse's tile-wise GeMM kernel matches the performance of PyTorch's standard GeMM kernel. While the tile-wise All-reduce introduces around a 6% overhead[2], it remains competitive with PyTorch's NCCL [6] backend. Taking PyTorch as the baseline, we observe a 2% performance degradation in our kernel due to implementation inefficiency without overlapping. However, enabling overlapping results in a 20.5% speedup. Overall, DistFuse achieves a 44.3% reduction in All-reduce communication latency.
Integration to vLLM. We are integrating our idea into vLLM and have implemented DistFuse's tile-wise All-reduce. We observe 5.2% to 11.1% end-to-end latency speedup on Llama 3-70B. We believe this is because vLLM optimizes kernels independently and accesses data linearly during All-reduce, whereas our tile-wise All-reduce preserves the spatial locality of GeMM. We expect further speedups from DistFuse's overlapping, with the current improvement as a lower bound, and caching effects from avoiding global memory access may even exceed the speedup from overlapping.

4 Discussion and Future Work

The current experiments demonstrate that fine-grained kernel fusion significantly enhances the performance of attention blocks even within a single-node environment. However, Llama 3 inference with NVLink does not present a significant communication bottleneck scenario. Moving forward, we plan to extend our evaluation to both the training process and large-scale, multi-node experiments. We also aim to go beyond the GeMM & All-reduce operators, where immediate communication could offer significant benefits. Besides, using the current implementation is incomplete which requires manual configurations, which is inefficient for real-world deployment. We plan to develop DistFuse as a comprehensive framework, which can automatically identify overlapping opportunities and select the most appropriate tile size based on a model execution plan. Additionally, we aim to explore finer-grained overlapping by co-designing computation and communication kernels, enabling higher MFU from instruction-level parallelism.

[2]This is an implementation issue that can be further improved instead of a fundamental overhead from the immediate communication methodology.

Acknowledgments

We appreciate the invaluable comments by anonymous reviewers of HotInfra 2024. This work was supported in part by the Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (IITP 2022-0-00531, Development of In-Network Computing Techniques for Efficient Execution of AI Applications). This publication is based upon work supported by the King Abdullah University of Science and Technology (KAUST) Office of Research Administration (ORA) under Award No. ORA-2021-CRG9-4382.

References

[1] Adel Nabli, Louis Fournier, Pierre Erbacher, Louis Serrano, Eugene Belilovsky, Edouard Oyallon. 2024. ACCO: Accumulate while you Communicate, Hiding Communications in Distributed LLM Training. arXiv:2406.02613 [cs.LG] https://arxiv.org/abs/2406.02613

[2] Alexander Borzunov, Max Ryabinin, Artem Chumachenko, Dmitry Baranchuk, Tim Dettmers, Younes Belkada, Pavel Samygin, and Colin A Raffel. 2024. Distributed Inference and Fine-tuning of Large Language Models Over The Internet. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*.

[3] Qiaoling Chen, Qinghao Hu, Guoteng Wang, Yingtong Xiong, Ting Huang, Xun Chen, Yang Gao, Hang Yan, Yonggang Wen, Tianwei Zhang, and Peng Sun. 2024. AMSP: Reducing Communication Overhead of ZeRO for Efficient LLM Training. arXiv:2311.00257 [cs.DC] https://arxiv.org/abs/2311.00257

[4] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: an automated end-to-end optimizing compiler for deep learning. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation (OSDI)*.

[5] NVIDIA Corporation. 2024. *CUTLASS: CUDA Templates for Linear Algebra Subroutines*. https://github.com/NVIDIA/cutlass

[6] NVIDIA Corporation. 2024. *NCCL: Optimized primitives for collective multi-GPU communication*. https://github.com/NVIDIA/nccl

[7] NVIDIA Corporation. 2024. *A TensorRT Toolbox for Optimized Large Language Model Inference*. https://github.com/NVIDIA/TensorRT-LLM

[8] Ana Klimovic. Foteini Strati, Xianzhe Ma. 2024. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications. In *Proceedings of the European Conference on Computer Systems (EuroSys)*.

[9] Guyue Huang, Yang Bai, Liu Liu, Yuke Wang, Bei Yu, Yufei Ding, and Yuan Xie. 2023. ALCOP: Automatic Load-Compute Pipelining in Deep Learning Compiler for AI-GPUs. In *Proceedings of the Machine Learning and Systems (MLSys)*.

[10] Abhinav Jangda, Jun Huang, Guodong Liu, Amir Hossein Nodehi Sabet, Saeed Maleki, Youshan Miao, Madanlal Musuvathi, Todd Mytkowicz, and Olli Saarikivi. 2022. Breaking the computation and communication abstraction barrier in distributed machine learning workloads. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[11] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. 2019. TASO: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

[12] Andrew Kerr, Duane Merrill, Julien Demouth, and John Tran. 2017. *CUTLASS: Fast Linear Algebra in CUDA C++*. https://developer.nvidia.com/blog/cutlass-linear-algebra-cuda/

[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

[14] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*. 583–598.

[15] Kshiteej Mahajan, Ching-Hsiang Chu, Srinivas Sridharan, and Aditya Akella. 2023. Better Together: Jointly Optimizing ML Collective Scheduling and Execution Planning using SYNDICATE. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[16] META. 2024. *Introducing Llama 3.1: Our most capable models to date*. https://ai.meta.com/blog/meta-llama-3-1/

[17] META. 2024. *meta-llama/llama3*. https://github.com/meta-llama/llama3

[18] OpenAI. 2024. *ChatGPT*. https://openai.com/chatgpt/

[19] OpenAI. 2024. *SORA*. https://openai.com/sora/

[20] Kazuki Osawa, Shigang Li, and Torsten Hoefler. 2023. PipeFisher: Efficient Training of Large Language Models Using Pipelining and Fisher Information Matrices. In *Proceedings of the Machine Learning and Systems (MLSys)*.

[21] Suchita Pati, Shaizeen Aga, Mahzabeen Islam, Nuwan Jayasena, and Matthew D. Sinclair. 2024. T3: Transparent Tracking & Triggering for Fine-grained Overlap of Compute & Collectives. arXiv:2401.16677 [cs.AR]

[22] Yanghua Peng, Yibo Zhu, Yangrui Chen, Yixin Bao, Bairen Yi, Chang Lan, Chuan Wu, and Chuanxiong Guo. 2019. A generic communication scheduler for distributed DNN training acceleration. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*.

[23] Amit Sabne. 2020. XLA: Compiling machine learning for peak performance. *Google Research*. https://research.google/pubs/pub50530

[24] Yining Shi, Zhi Yang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Ziming Miao, Yuxiao Guo, Fan Yang, and Lidong Zhou. 2023. Welder: Scheduling Deep Learning Memory Access via Tile-graph. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[25] 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).

[26] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*.

[27] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288 [cs.CL]

[28] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[29] Hongyu Zhu, Ruofan Wu, Yijia Diao, Shanbin Ke, Haoyu Li, Chen Zhang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Wei Cui, Fan Yang, Mao Yang, Lidong Zhou, Asaf Cidon, and Gennady Pekhimenko. 2022. ROLLER: Fast and Efficient Tensor Compilation for Deep Learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[30] Yonghao Zhuang, Hexu Zhao, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, and Hao Zhang. 2023. On optimizing the communication of model parallelism. In *Proceedings of the Machine Learning and Systems (MLSys)*.