Flash Communication: Reducing Tensor Parallelization Bottleneck for Fast Large Language Model Inference

Qingyuan Li[1] **Bo Zhang**[1] **Liang Ye**[1] **Yifan Zhang**[1] **Wei Wu**[1] **Yerui Sun**[1] **Lin Ma**[1] **Yuchen Xie**[1]

Abstract

The ever-increasing sizes of large language models necessitate distributed solutions for fast inference that exploit multi-dimensional parallelism, where computational loads are split across various accelerators such as GPU clusters. However, this approach often introduces significant communication overhead, especially on devices with limited bandwidth. In this paper, we introduce *Flash Communication*, a novel low-bit compression technique designed to alleviate the tensor-parallelism communication bottleneck during inference. Our method substantially boosts intra-node communication speed by more than **3**× and reduces the *time-to-first-token* by **2**×, with nearly no sacrifice in model accuracy. Extensive experiments on various up-to-date LLMs demonstrate the effectiveness of our approach.

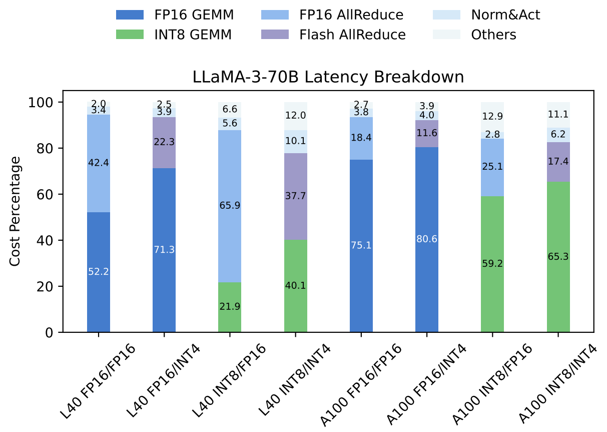


Figure 1. Prefill cost breakdown of LLaMA-3-70B operations with and without Flash Communication, as measured by NSys (NVIDIA, 2024b). Tested on 4×L40/A100 GPUs (TP=4) with a batch size of 8, each with 1024 input and 64 output tokens. NCCL (NVIDIA, 2024d)'s Ring All-Reduce is applied. The notion of x-ticks (e.g. L40 FP16/FP16) denotes GPU type, model weight precision, and communication precision, respectively.

1. Introduction

To date, the number of parameters of large language models has tremendously increased. For instance, GPT-3 (Brown et al., 2020) has 175B, DeepSeek V2 (Liu et al., 2024) utilizes 236B, LLaMA-3 (Dubey et al., 2024) reaches 450B. Their enormous sizes create big challenges for both training and inference.

To tackle the scaling difficulties of large language models, the research community now resorts to multiple parallelism strategies across a large group of computing accelerators. Since previous parallelism methods focus on resolving the training challenges, we quickly review these methods for a background check. Particularly, *data parallelism* (Dean et al., 2012; Ben-Nun & Hoefler, 2019) is first introduced to allocate the training samples onto multiple GPUs where each GPU retains a duplicate of the model and processes its own given batch of samples. Synchronization is hence required at the end of each iteration to update the model parameters. In the LLM era, ZeRO (Rajbhandari et al., 2020) and FSDP (Zhao et al., 2023) renovate data parallelism by sharding models on all devices but virtually rendering a whole model on a single device through All-Gather communication. In contrast, *pipeline parallelism* partitions sequential layers onto different GPUs where point-to-point communication is adopted to transmit activation and gradients. However, it creates data dependency which leads to substantial GPU idle time, called *bubbles*. To improve GPU utilization, GPipe (Huang et al., 2019) schedules micro-batches in a pipeline with forward passes and then followed by backward passes. PipeDream (Harlap et al., 2018) proposes one-forward one-backward (1F1B) to further reduce the bubble ratio. Megatron-LM (Narayanan et al., 2021) advances PipeDream by allowing each device to perform computation for multiple non-contiguous subsets of layers. Another dimension to split the model is *tensor parallelism* which splits the tensors of each layer and performs All-Reduce to aggregate the activation and gradients from all devices. Megatron-LM (Shoeybi et al., 2019; Narayanan

[*]Equal contribution [1]Meituan. Correspondence to: Qingyuan Li <liqingyuan02@meituan.com>.

Preprint.

et al., 2021) is such an example that devises delicate conjugate tensor slicing schemes (e.g. column-wise first and row-wise next for the MLP layer) to remove unnecessary synchronization inside a transformer block.

With the rise of LLMs developed for the long-context scenario, *sequential parallelism* (Korthikanti et al., 2023) is proposed to divide activation of LayerNorm and Dropout layers in the sequence dimension as they are sequence-independent. It also jointly combines with tensor-parallelism by replacing two All-Reduce operations with one All-Gather and one Reduce-Scatter to merge the communication cost. However, self-attention and MLP layers are left untouched for sequential parallelism. In this regard, *context parallelism* (NVIDIA, 2024b) is designed to separate all layers in sequence dimension. To break the sequence dependency in self-attention, Ring Attention (Liu et al., 2023) applies blockwise self-attention and feedforward (Dao, 2023; Liu & Abbeel, 2023) in a distributed environment with point-to-point communication. On top of this, Deepspeed-Ulysses (Jacobs et al., 2023) exchanges point-to-point communication for All2All for faster speed.

Another emerging direction is sparse architectures represented by mixture-of-experts models (Jiang et al., 2024; Team, 2024; Liu et al., 2024). *Expert parallelism* (Fedus et al., 2022) parallelizes the experts on different GPUs which requires All2All communication. Deepspeed-MoE (Rajbhandari et al., 2022) propose hierarchical All2All communication to reduce the number of communication hops.

As large language models continue to scale up, modern frameworks like DeepSpeed (Microsoft, 2024), and Megatron (NVIDIA, 2024a) tend to make joint use of the aforementioned parallelism to accelerate the training process. Nevertheless, they easily meet communication bottlenecks as they require many collective operations. This overhead grows as the model becomes larger.

Meanwhile, the communication bottleneck is also pronounced when serving large language models in the cloud. Constrained by strict service level objectives (SLOs), multiple parallelism schemes are adopted to speed up the inference, where tensor parallelism is the most popular option among all. Besides, due to the lower bandwidth of inference GPUs, communication can account for more than half of the prefill inference cost where an 80-layer LLaMA-3-70B (Dubey et al., 2024) carries out 160 all-reduce operations at each forward pass on 4×L40 GPUs, as shown in Figure 1. Therefore, to enable a faster speed, we must make efficient use of limited intra-node bandwidth.

In this work, we design a novel technique to reduce the communication cost introduced by tensor parallelism without substantially sacrificing accuracy. Our contributions are,

1. Through detailed measurements, we unveil the communication bottleneck problem that also recurs in large language model (LLM) inference. For instance, communication can account for up to 65% of the total latency on NVIDIA L40 GPUs (Fig. 1).

2. We design an efficient communication mechanism called *Flash Communication*, which applies low-bit fine-grained quantization on activations to reduce communication volume and employs a *two-step all-reduce* strategy to minimize communication hops.

3. We implement a fused CUDA kernel called Flash All-Reduce to perform Flash Communication, achieving up to a 2× reduction in time-to-first-token (TTFT) on NVIDIA L40 GPUs. Even on A100 GPUs with higher communication bandwidth, we observe notable latency reductions, demonstrating the effectiveness of our method.

2. Related Work

Before diving into the investigated problem, we cover some fundamental knowledge required for discussion in Appendix A. We suggest that readers without prior experience quickly review the content.

Communication efficiency is crucial to distributed training and serving, as it directly affects the total processing time and cost. Several techniques have been proposed to optimize communication in distributed training in recent years, including topology optimization, pipeline optimization, and compression methods.

2.1. Topology Optmization

Topology optimization adjusts communication patterns to match the physical topology of hardware to reduce communication latency/hops, mainly ring-based and tree-based. Ring-All-Reduce (Baidu Research, 2024) organizes workers in a ring topology so that the overall communication latency is constant regardless of the number of workers. Say a worker transmits data of volume M to a group of $N-1$ workers, the total communication volume is $2M(N-1)/N$, which is approximately $2M$ when $N >> 1$. However, it doesn't take the physical topology into account, where intra- and inter-node communication have different bandwidths. Hence the average speed depends largely on the lowest bandwidth in such a strategy. Hierarchical Ring-All-Reduce (Jia et al., 2018) highlights the importance of hierarchical structures in managing overheads, which employs three-phase all-reduce for separate intra- and inter-node communication. Later, 2D-Torus (Mikami et al., 2018) organizes GPUs in a 2D-grid of (X, Y) so that the inter-node horizontal communication volume is X times smaller than that of hierarchical

Ring-All-Reduce. NCCL (NVIDIA, 2019) introduces double binary trees (Sanders et al., 2009) provides logarithmic latency by reducing hops from $2(N-1)$ to $2log(N)$. However, it is more prone to result in suboptimal bandwidth utilization, as only a subset of nodes are engaged in any given communication step.

With the rise of sparse architectures like mixture-of-experts, All2All collective operation is common for communication in expert parallelism. DeepSpeed-MoE (Rajbhandari et al., 2022) and HetuMoE (Nie et al., 2022) both utilize a scalable hierarchical scheme to reduce all-to-all communication hops for faster speed.

Besides, NVLink SHARP (NVIDIA, 2023) is a hardware improvement that offloads collective operations from GPUs to the network devices, hence eliminating the need to send data multiple times between endpoints.

2.2. Pipeline Optimization

Pipelining optimization aims to maximize resource utilization with optimized scheduling strategies, mainly by overlapping computation with communication. Domino (Wang et al., 2024) breaks data dependency in Tensor-parallelism by splitting activations row-wisely and weights column-wisely into smaller independent parts. FLUX (Chang et al., 2024) divides computation and communication operations into much finer-grained operations and later merges them in a larger kernel to effectively hide communication. DistributedGEMM (Hassani et al., 2024) provides an implementation based on CUTLUSS (NVIDIA, 2024f) using P2P communication. ScMoE (Cai et al., 2024) implements a shortcut-connected MoE architecture to effectively decouple communication from its conventional sequence, allowing for a substantial overlap.

2.3. Communication Compression

Compression techniques like sparsification and quantization are proposed to balance communication reduction with acceptable performance degradation. Sparse Communication (Aji & Heafield, 2017) observes that gradient updates are mostly close to zero and maps them directly to zero to only exchange sparse matrices among distributed nodes. In contrast, DISCO (Qin et al., 2024) aims to achieve sparse communication by gradually pruning the network to generate sparse features. QSDP (Markov et al., 2023) remove FSDP's communication bottleneck by performing both gradient and weight quantization. ZeRO++ (Wang et al., 2023) applies All-Gather with blockwise weight quantization and an All2All-based gradient quantization to reduce the communication volume when collecting weights and gradients.

3. Method

3.1. Motivation

Tensor Parallelism (TP) is now supported in almost all mainstream inference frameworks like TensorRT-LLM (NVIDIA, 2023), vLLM (Kwon et al., 2023), SGLang (sgl-project, 2024), and LMDeploy (LMDeploy, 2023), becoming the most adopted scheme in LLM inference. However, TP comes at a non-negligible cost due to heavy communication, which in the case of larger language models creates an excessive communication overhead. For example, the communication overhead of LLaMA-3-70B on L40 GPUs easily meets the bottleneck as the input token length increases, shown by the cost breakdown of LLaMA-3-70B operations in Figure 2. Although high-end training-purpose accelerators like NVIDIA A100 where GPUs are connected through NVLink (NVIDIA, 2024e), the communication overhead still reaches a notable 20%. We can easily conclude that TP communication is the inference bottleneck.

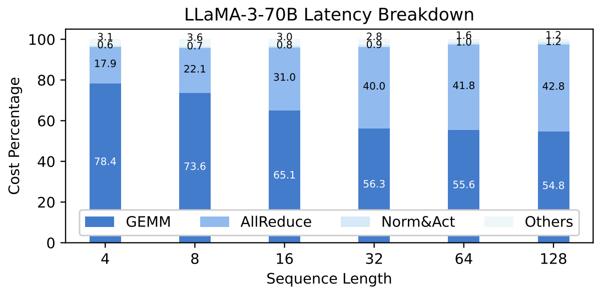


Figure 2. Prefill cost breakdown of LLaMA-3-70B operations at various sequence lengths. Tested on 4×L40/A100 GPUs (TP=4) with a batch size of 8.

For communication optimization, one can think of overlapping the communication with computation to hide overhead but it requires sophisticated design with scheduling which is harder to implement in any given inference framework, for which reason NanoFlow (Zhu et al., 2024) invents a new serving framework to circumvent the difficulty. On the contrary, compression is a handy option but none of the above-mentioned methods investigated communication quantization for LLM inference. It could be due to the activation quantization challenge posed at the inference stage, which is pointed out by LLM.int8() (Dettmers et al., 2022) and SmoothQuant (Xiao et al., 2024) that activations are harder to quantize because of outliers. The communication quantization method for training doesn't suffer from this problem as it only quantizes weights and gradients (Wang et al., 2023). Besides, during training, communication quantization degradation can be compensated by further learning. Whereas at inference, the quantization loss is nearly irreversible.

At the same time, the existing Ring All-Reduce operation adopted in tensor parallelism remains a bottleneck at inference since it is inclined to be constrained by lower bandwidth. Furthermore, to integrate quantization with Ring All-Reduce, also shown by ZeRO++ (Wang et al., 2023), it requires N times of sequential quantization and dequantization in a complete Reduce-Scatter, which worsens the latency.

The above issues call for a delicate orchestration of the quantization approach and a better All-Reduce scheme.

3.2. Flash Communication

Motivated by the above, we approach the inference challenges in the distributed scenario with quantization and topology optimization. In the paper, we specifically examine tensor parallelism as it is the most popular paradigm.

Take tensor parallelism in LLaMA-3 (Dubey et al., 2024) as an example, the tensors of QKV projections are first sliced column-wisely and then the output projection row-wisely. After that, an All-Reduce operation is required to collect activations on each device, shown in Fig. 3. Similarly in the feedforward network, the gate projection and the up projection are split by column and the down projection by row, then another All-Reduce is needed to sum up the activations from both devices. To reduce the communication volume, we are left to compress the activation from the output projection and the down projection.

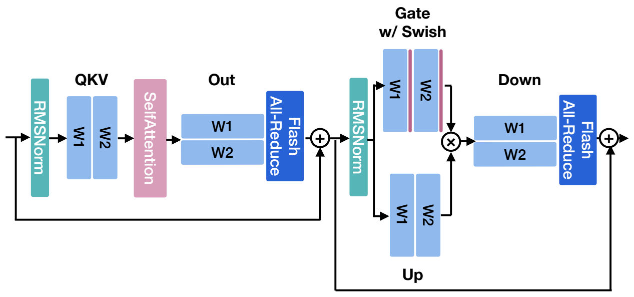


Figure 3. Tensor parallelism for a LLaMA-3 transformer block. Our Flash All-Reduce is applied to speed up communication.

3.2.1. Quantization Challenge

To obtain an optimal trade-off between accuracy and latency, we choose to apply low-bit quantization. From Fig. 4, we observe that fine granularity is necessary since per-token quantization at larger block sizes suffers from performance collapse in terms of C4 perplexity, albeit the asymmetric version is relatively better.

However, we discover that it is non-trivial to apply low-bit

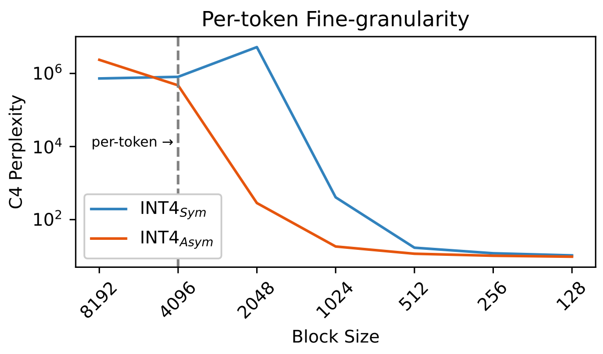


Figure 4. Activation quantization with various block sizes of LLaMA-3-8B on C4. Starting from 4096 (the length of hidden dimension), the granularity becomes finer till 128.

activation quantization in this scenario. To investigate the quantization sensitivity, we calculate the layerwise mean squared errors (MSE) before and after activation quantization on LLaMA-3-8B, as depicted in Figure 5. We find that the down projection d_{proj} is much harder to quantize than the output projection o_{proj}, as the former MSEs are quite distinct even on a logarithmic scale. This phenomenon is also discovered by (Li et al., 2023; Ashkboos et al., 2023; Yu et al., 2024).

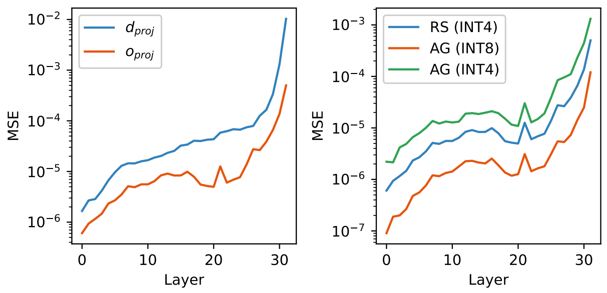


Figure 5. Left: Comparison of o_{proj} and d_{proj} All-Reduce Quantization MSE. Right: MSE of quantization before Reduce-Scatter (RS) vs. All-Gather (AG).

Besides, an All-Reduce comprises a pair of Reduce-Scatter and All-Gather operations, where the quantization corresponding to each operation exhibits different levels of difficulty, see Fig. 5 right. This is as expected since the quantization before Reduce-Scatter only introduces rounding errors while in the case of All-Gather, it includes both rounding and accumulated errors. Alternatively, we could use a higher precision for the quantization before All-Gather to improve accuracy.

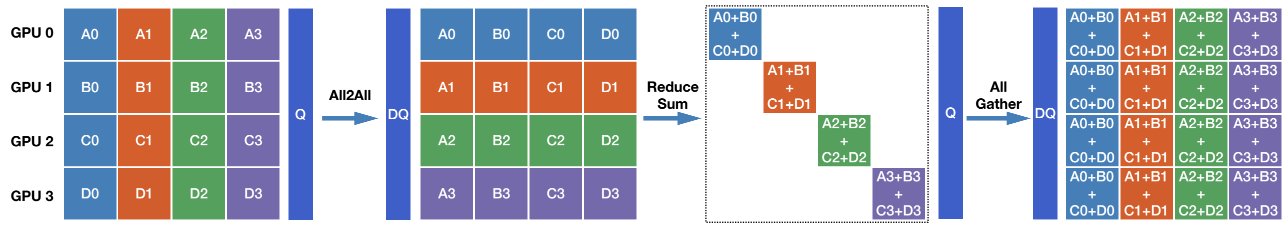


Figure 6. Flash communication's *two-step All-Reduce*. The communication volume is quantized and dequantized only twice.

3.2.2. Algorithm

Considering the above issues, we design a two-step quantization strategy to replace vanilla Ring All-Reduce, as portrayed in Fig. 6. Its integration with tensor parallelism is shown in Fig.3. We name our overall strategy as *two-step All-Reduce*.

Fig. 6 illustrates how Flash Communication works. First, we divide the computation volume (activation) on each GPU by the number of ranks. After fine-grained quantization on activation, we perform All2All communication so that each device receives its computation load for reduction. After on-device reduction, the sum is again quantized to speed up the transmission. We then perform All-Gather to collect all results and dequantization to recover float values on each device. This two-step workflow is also formulated by Alg. 1.

Algorithm 1 Flash All-Reduce

```
Input: Communication volume $M$, world size $N$, chunk size $C$, quantization bit-width $b$, group size $g$
Output: Reduced sum $S^{dq}$
Divide $M$ into $T = \lceil M/C \rceil$ chunks.
for $1 \leq i \leq T$ do
    // Quantize volume to obtain zeros and scales
    $M_i^q$, $z_i$, $s_i$ = FinegrainedQuantize($M_i$, $b$, $g$);
    // Each device sends and receives volume from others
    All2All($M_i^q$, $z_i$, $s_i$, $N$);
    for $1 \leq j \leq N$ do
        $M_{i_j}^{dq}$ = Dequantize($M_{i_j}^q$, $z_{i_j}$, $_{i_j}$);
    end for
    $S_i$ = ReduceSum($M_{i_0}^{dq}$, $M_{i_1}^{dq}$, $\cdots$, $M_{i_N}^{dq}$);
    $S_i^q$, $z_i^s$, $s_i^s$ = FinegrainedQuantize($S_i$, $b$, $g$);
    // Each device collects the reduced sum from others
    All-Gather($S_i^q$, $z_i^s$, $s_i^s$, $N$);
    for $1 \leq j \leq N$ do
        $S_{i_j}^{dq}$ = Dequantize($S_i^q$, $z_i^s$, $s_i^s$);
    end for
end for
```

Table 1. Comparison of Ring All-Reduce vs. Flash All-Reduce

Method	Ring All-Reduce	Flash All-Reduce
Total Volume	$2M(N-1)/N$	$2M(N-1)/N$
Reduce Step	$N-1$	1
Reduce-Scatter	M/N	$M(N-1)/N$
Gather Step	$N-1$	1
All-Gather	M/N	$M(N-1)/N$
QDQ Step	N	2

3.2.3. Kernel Design

For efficiency, we implement a fused Flash All-Reduce kernel to encompass all the above collective communication operations and quantization steps. Compared with Ring All-Reduce in Table 1, Flash All-Reduce cuts quantization-dequantization steps from N to 2, and Reduce/Gather steps from $N-1$ to 1. Although the size of total volumes remains the same, each of our volumes is quantized to lower bits, substantially reducing the amount of data to transmit. We summarize three key aspects in designing our kernel below.

Fast Fine-grained Quantization The total communication volume M for each rank is divided into T chunks for transmission. Given a chunk size C, we draw how GPU threads are organized in parallel to process the chunk information in Fig. 7. A chunk is split into N blocks and each block corresponds to 32 warps, where each warp is a collection of 32 threads and each thread can process eight FP16 elements. Take our asymmetric quantization with a group size of 128 as an example, we perform quantization on each group of 128 elements using 16 threads. Specifically, we leverage the CUDA API function `__shfl_xor_sync` to iteratively exchange information among these warp threads to achieve max/min reduction efficiently.

Fast Communication. Instead of calling All2All primitive, we utilize GPU peer direct memory access from CUDA Run-

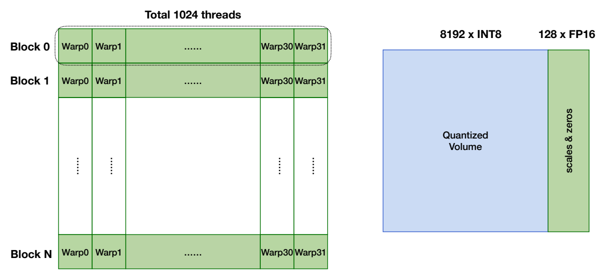


Figure 7. Thread mapping of fast fine-grained quantization.

time API (NVIDIA Corporation, 2024) to transmit quantized volume, where we can directly fetch data from different ranks, substantially boosting the communication speed.

Fast Dequantization. Once quantization volumes are received, we have to dequantize them into FP16 for sum reduction. As naïve INT4 to FP16 conversion incurs overhead, we utilize the dequantization layout in (Kim et al., 2022). To coordinate its ordering online, we also employ fast INT4 packing (LMDeploy, 2023), as illustrated in Fig. 8. Given that two 32-bit unsigned integers `U0` and `U1` holding 4 INT4-quantized activations (each stored in the lower 4 bits out of 8 bits) to transmit, we first perform the right shift by 12 bits, and then apply bitwise OR to itself. Later we select the target bits from these two integers with CUDA Math API `__byte_perm`. In this way, we can pack 8 4-bit integers in a convenient order to dequantize. Next we apply `lop3.b32`[1] to perform logical operation `(0xF0 & 0xCC)|0xAA` on the packed variable with mask `0x000F000F` and `0x64006400`, then we subtract it with `0x64006400`, which effectively represents `W1` and `W0` in FP16. The dequantization can be performed iteratively by varying the masks for the rest INT4 integers.

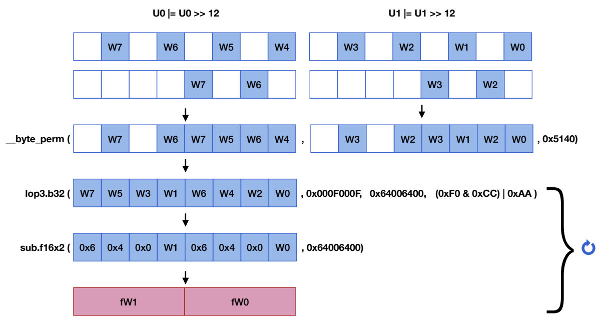


Figure 8. Fast INT4 packing and dequantization. For simplicity, only the dequantization of W1 and W0 is shown. An empty square means zero.

[1] https://docs.nvidia.com/cuda/parallel-thread-execution/#logic-and-shift-instructions-lop3

INT6 Quantization. Given that quantization prior to All-Gather leads to greater loss, as illustrated in Figure 5 (left), we opt for an INT8 bit-width for All-Gather operations and maintain INT4 for ReduceSum, effectively creating an INT6 solution. As later shown in Table 3, the INT6 configuration strikes a commendable balance between performance and communication efficiency.

4. Experiments

4.1. Setup

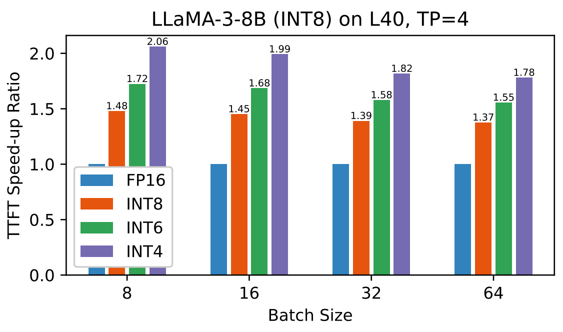
Unless otherwise noted, we use an input token length of 1024 and an output token length of 64 for the inference measurement. Latencies are tested on NVIDIA L40 and A100 SXM GPUs. The baseline uses FP16 for communication.

4.2. Accuracy Comparison

FP16 Weights. We evaluate the accuracy of LLaMA-2 and LLaMA-3 models on PIQA (Bisk et al., 2020), ARC$_C$ and ARC$_E$ (Clark et al., 2018), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021) in various communication quantization bit widths, shown in Table 2. In all cases, asymmetric INT8 quantization obtains the best accuracies. Asymmetric INT4 is also better than symmetric INT4. C4 (Raffel et al., 2020) and WikiText (Merity et al., 2016) results are shown in Table 6 of Appendix B.1.

Table 2. Accuracy of LLaMA models with various communication quantization strategies. All model weights are kept in FP16 precision. Prec: Communication precision. All INT quantization is asymmetrical. HS: HellaSwag, WG: WinoGrande.

Model	Prec	PIQA	ARC$_C$	ARC$_E$	HS	WG	Avg
2-7B	FP16	79.11	46.33	74.58	76.01	69.30	64.83
	INT8	79.11	45.99	74.75	76.10	69.06	64.77
	INT6	78.78	45.99	74.79	75.75	68.75	64.68
	INT4	78.02	45.82	74.49	75.63	67.25	64.23
2-13B	FP16	80.52	49.15	77.48	79.39	72.14	67.83
	INT8	80.69	49.06	77.61	79.34	71.59	67.78
	INT6	79.98	49.32	77.06	79.17	71.11	67.11
	INT4	79.60	48.21	76.89	78.92	71.90	67.04
2-70B	FP16	82.70	57.34	81.02	83.80	77.98	72.43
	INT8	82.75	57.68	80.93	83.80	77.98	72.47
	INT6	82.48	57.00	80.85	83.77	76.87	72.23
	INT4	82.92	57.25	80.43	83.60	77.27	71.99
3-8B	FP16	80.79	53.41	77.69	79.16	72.77	68.64
	INT8	80.58	52.47	77.40	79.09	73.09	68.45
	INT6	79.98	51.37	77.65	78.73	73.16	68.06
	INT4	80.09	51.02	75.84	78.11	70.48	66.92
3-70B	FP16	84.55	64.33	85.86	84.89	80.35	75.25
	INT8	84.55	63.91	85.82	84.90	80.82	75.27
	INT6	84.11	61.69	85.44	84.87	80.35	74.76
	INT4	83.13	61.35	83.33	84.69	78.93	73.82



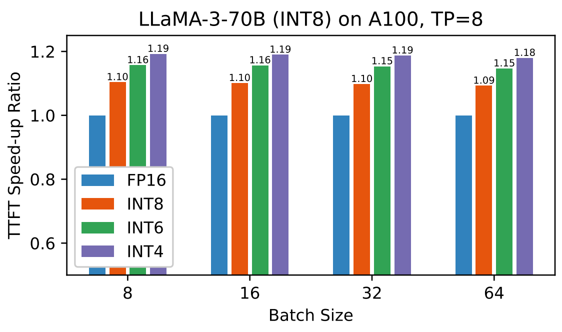


Figure 9. TTFT speed-up ratio of 8-bit LLaMA-3-8B under various communication quantization bit widths on L40 with TP=4 (left), and 8-bit LLaMA-3-70B on A100 with TP=8 (right).

INT8 Weights. As model weights are quantized, the impact of communication overhead becomes more pronounced. This observation motivates us to explore communication quantization in this context. We begin by quantizing the weights of the LLaMA model using Smoothquant (Xiao et al., 2024). Subsequently, we implement fine-grained communication quantization to assess its impact on performance as detailed in Table 3. Results on C4 (Raffel et al., 2020), WikiText-2 (Merity et al., 2016) is shown in Appendix B.1.

Table 3. Accuracy of LLaMA models with various communication quantization strategies (denoted as 'Comm Prec'). All model weights are quantized into INT8 precision with SmoothQuant ($\alpha = 0.85$ except 0.9 for LLaMA2-70B). Prec: Communication precision. All INT quantization is asymmetrical.

Model	Prec	PIQA	ARC$_C$	ARC$_E$	HS	WG	Avg
2-7B	FP16	79.00	46.16	74.24	75.89	68.75	68.81
	INT8	79.27	45.65	74.37	75.89	68.51	68.74
	INT6	78.56	45.39	74.75	75.75	68.67	68.62
	INT4	77.53	45.31	74.20	75.51	68.59	68.23
2-13B	FP16	80.25	49.49	77.27	79.37	71.59	71.59
	INT8	80.41	49.32	77.53	79.21	72.22	71.74
	INT6	80.09	49.40	77.02	79.14	71.74	71.48
	INT4	79.11	49.06	76.30	78.89	71.27	70.93
2-70B	FP16	83.08	57.94	80.98	83.54	77.66	76.64
	INT8	83.08	57.76	80.68	83.55	78.14	76.64
	INT6	82.81	57.94	80.85	83.78	76.80	76.44
	INT4	82.92	56.83	80.35	83.41	78.14	76.33
3-8B	FP16	80.36	51.96	77.69	78.71	73.48	72.44
	INT8	80.09	52.90	77.57	78.79	73.09	72.49
	INT6	80.03	52.56	79.12	78.38	71.82	72.38
	INT4	78.94	50.17	77.10	77.87	70.56	70.93
3-70B	FP16	84.44	63.99	85.56	84.55	79.72	79.65
	INT8	83.84	63.48	85.56	84.67	79.79	79.47
	INT6	83.57	61.26	83.67	84.66	80.58	78.75
	INT4	82.70	61.60	83.29	84.51	77.82	77.98

4.3. Latency and Throughput Performance

Fig. 9 illustrates weight-quantized LLaMA-3-8B and LLaMA-3-70B's TTFT comparison with and without Flash communication. The lowest quantization bit yields the most gain, i.e. **2.06**× and **1.19**× on L40 and A100 respectively. More measurements are listed in Appendix C.

5. Ablation Study

5.1. Integer vs. Low-bit Float

Table. 4 shows the difference between INTx and FPx communication quantization. In general, INT8 performs comparably with FP8, while the asymmetric version of INT8 is the best among all. FP6 is a mixed version of FP8 (Micikevicius et al., 2022) and FP4 (Rouhani et al., 2023), which is a fair comparison with similarly mixed INT6.

Table 4. LLaMA models' C4 Perplexity with INTx vs FPx quantization with a group size of 128. Weights are in FP16.

Prec	2-7B	2-13B	2-70B	3-8B	3-70B
FP8$_{Sym}$	6.98	6.47	5.52	8.90	6.75
INT8$_{Sym}$	6.98	6.47	5.52	8.90	6.74
INT8$_{Asym}$	**6.98**	**6.47**	**5.52**	**8.89**	**6.74**
FP6$_{Sym}$	7.09	6.52	5.56	9.22	6.87
INT6$_{Sym}$	7.15	6.57	5.59	9.48	6.94
INT6$_{Asym}$	**7.08**	**6.51**	**5.55**	**9.20**	**6.86**
FP4$_{Sym}$	7.24	6.60	5.61	9.72	7.07
INT4$_{Asym}$	7.50	6.71	5.69	10.51	7.30
INT4$_{Asym}$	**7.21**	**6.58**	**5.60**	**9.68**	**7.04**

5.2. Naïve vs. Rotation-based Quantization

As previously shown in Fig. 4, the C4 perplexity of coarse quantization suffers performance collapse, while fine-grained quantization gradually resolves the problem as the group size increases. We investigate whether the Hadamard transform popularized by QuaRot (Ashkboos et al., 2024) could alleviate the issue in the coarse setting in Table 5. It turns out that Hadamard transform with coarse quantization readily performs well, however, it loses its advantage in finer granularity cases as compared with naïve asymmetric quantization. Besides, it doesn't exhibit gains on FP8.

Table 5. LLaMA models' C4 perplexity with communication quantization (naïve vs. rotation) at various granularity levels and precision formats. Asymmetric quantization is used for INT4 and symmetric for FP8. HT: Hadamard Transform

MODELS	GROUP	INT4	W/ HT	FP8	W/ HT
3-8B	8192	2363367.8	10.61	8.91	8.96
	2048	284.56	10.11	8.91	8.96
	128	9.68	**9.67**	**8.90**	8.94
3-70B	8192	7417.79	7.86	6.75	6.81
	1024	10.47	7.76	6.75	6.81
	128	**7.04**	7.64	**6.75**	6.82
2-7B	8192	47601520	7.86	7.01	7.01
	1024	8.78	7.35	6.98	7.01
	128	**7.21**	7.24	**6.98**	7.01
2-13B	8192	306.39	6.80	6.47	6.49
	1024	7.43	6.68	6.47	6.49
	128	**6.58**	6.62	**6.47**	6.49
2-70B	8192	49.96	5.71	5.52	5.54
	1024	6.17	5.67	5.52	5.54
	128	**5.60**	5.64	**5.52**	5.54

5.3. Flash All-Reduce vs. Ring All-Reduce

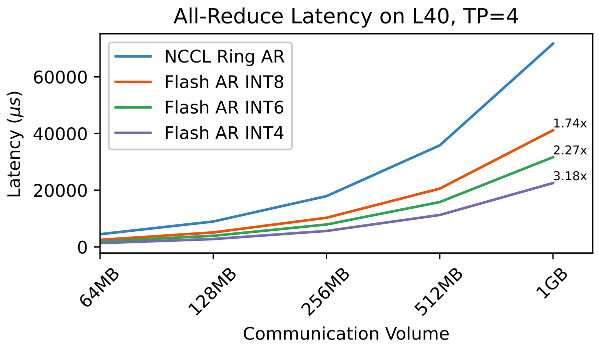


Figure 10. Flash Communication's All-Reduce (Flash AR) Performance compared with NCCL's ring version. NCCL's latency is tested with `nccl-test` (NVIDIA, 2016-2024).

Assembling several boosting techniques, the speed of our Flash All-Reduce kernel surpasses that of Ring All-Reduce by a large margin. Fig. 10 exhibits the latency measurement given a certain amount of communication volume. When the communication volume becomes obvious (e.g. larger than 64MB), our quantized All-Reduce is crucial to reduce the cost, where the INT4 version brings at most **3.18**× kernel latency reduction. Noticeably, the mixed precision version INT6 obtains a good trade-off between INT8 and INT4.

We further show that the number of streaming processors (SMs) matters in Fig. 11. When the communication volume is of small size, a smaller number of SMs is beneficial as less kernel launch and inter-block synchronization overhead is produced. When the volume gets larger, more SMs are required for calculation. A configuration of 48 SMs strikes a better balance between communication and computation.

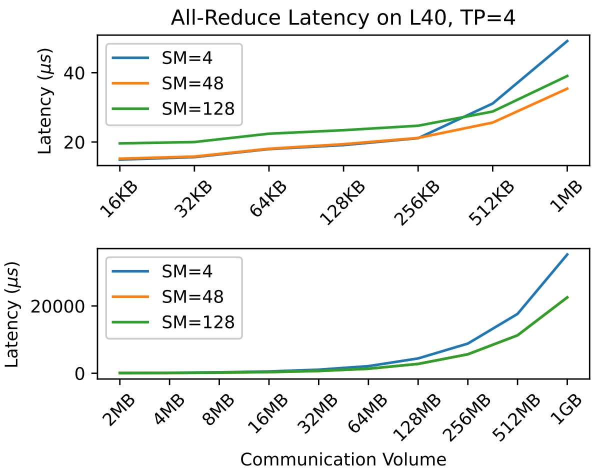


Figure 11. The number of SMs affects the communication latency at different sizes of communication volume. SM=48 is similar to SM=128 in larger volumes.

6. Conclusion

Our work presents a novel technique to reduce the communication volume associated with tensor parallelism while maintaining accuracy. Our key contributions include a comprehensive analysis that reveals the communication bottleneck in Large Language Model (LLM) inference, the design of a fast communication mechanism known as Flash Communication, and the demonstration of its implementation, which has been shown to achieve up to a 2× TTFT reduction. Flash Communication employs fine-grained quantization on activations and a two-step All-Reduce strategy to decrease communication volumes significantly. We have conducted extensive experiments on NVIDIA L40 and A100 GPUs across various configurations and with several state-of-the-art LLMs, which have consistently demonstrated the effectiveness of our approach. These findings address a critical challenge in parallel computing and pave the way for more efficient and scalable LLM inference.

References

Aji, A. F. and Heafield, K. Sparse communication for distributed gradient descent. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2017. doi: 10.18653/v1/d17-1045. URL http://dx.doi.org/10.18653/v1/D17-1045.

Ashkboos, S., Markov, I., Frantar, E., Zhong, T., Wang, X., Ren, J., Hoefler, T., and Alistarh, D. Towards end-to-end 4-bit inference on generative large language models. *arXiv preprint arXiv:2310.09259*, 2023.

Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Cameron, P., Jaggi, M., Alistarh, D., Hoefler, T., and Hensman, J. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*, 2024.

Baidu Research. baidu-allreduce: A C++ library demonstrating ring allreduce and ring allgather techniques. GitHub repository, 2024. URL https://github.com/baidu-research/baidu-allreduce.

Ben-Nun, T. and Hoefler, T. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys (CSUR)*, 52(4):1–43, 2019.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. 2020.

Cai, W., Jiang, J., Qin, L., Cui, J., Kim, S., and Huang, J. Shortcut-connected expert parallelism for accelerating mixture-of-experts, 2024. URL https://arxiv.org/abs/2404.05019.

Chang, L.-W., Bao, W., Hou, Q., Jiang, C., Zheng, N., Zhong, Y., Zhang, X., Song, Z., Yao, C., Jiang, Z., Lin, H., Jin, X., and Liu, X. Flux: Fast software-based communication overlap on gpus through kernel fusion, 2024. URL https://arxiv.org/abs/2406.06858.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Dettmers, T., Lewis, M., Belkada, Y., and Zettlemoyer, L. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL https://arxiv.org/abs/2208.07339.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Facebook. Gloo: Collective communications library with various primitives for multi-machine training. https://github.com/facebookincubator/gloo, 2024. Accessed: 2024-11-23.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G., and Gibbons, P. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

Hassani, A., Isaev, M., McDonald, N., Ren, J., Thakkar, V., Wu, H., and Shi, H. Distributed gemm, 2024. URL https://blog.shi-labs.com/distributed-gemm-88be6a481e2b. Accessed: 2024-12-04.

Hidayetoglu, M., de Gonzalo, S. G., Slaughter, E., Surana, P., Hwu, W.-m., Gropp, W., and Aiken, A. Hiccl: A hierarchical collective communication library. *arXiv preprint arXiv:2408.05962*, 2024.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

IEEE. IEEE Standard for Floating-Point Arithmetic. Technical Report IEEE 754-1985, Institute of Electrical and Electronics Engineers, New York, NY, 1985. URL https://standards.ieee.org/ieee/754/6210/.

Jacobs, S. A., Tanaka, M., Zhang, C., Zhang, M., Song, S. L., Rajbhandari, S., and He, Y. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.

Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., Casas, D. d. l., Hanna, E. B., Bressand, F., et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Kim, Y. J., Henry, R., Fahim, R., and Awadalla, H. H. Who says elephants can't run: Bringing large scale moe models into cloud scale production. *arXiv preprint arXiv:2211.10017*, 2022.

Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., and Catanzaro, B. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Li, Q., Zhang, Y., Li, L., Yao, P., Zhang, B., Chu, X., Sun, Y., Du, L., and Xie, Y. Fptq: Fine-grained post-training quantization for large language models. *arXiv preprint arXiv:2308.15987*, 2023.

Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., Dengr, C., Ruan, C., Dai, D., Guo, D., et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Liu, H. and Abbeel, P. Blockwise parallel transformers for large context models. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8828–8844. Curran Associates, Inc., 2023.

Liu, H., Zaharia, M., and Abbeel, P. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

LMDeploy. Lmdeploy: A toolkit for compressing, deploying, and serving llm. `https://github.com/InternLM/lmdeploy`, 2023.

Markov, I., Vladu, A., Guo, Q., and Alistarh, D. Quantized distributed training of large models with convergence guarantees, 2023. URL `https://arxiv.org/abs/2302.02390`.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Micikevicius, P., Stosic, D., Burgess, N., Cornea, M., Dubey, P., Grisenthwaite, R., Ha, S., Heinecke, A., Judd, P., Kamalu, J., et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Microsoft. DeepSpeed: Deep learning optimization library. GitHub repository, 2024. URL `https://github.com/microsoft/DeepSpeed`.

Microsoft. Microsoft collective communication library (msccl). `https://github.com/microsoft/msccl`, 2024. Accessed: 2024-11-23.

Mikami, H., Suganuma, H., Tanaka, Y., Kageyama, Y., et al. Massively distributed sgd: Imagenet/resnet-50 training in a flash. *arXiv preprint arXiv:1811.05233*, 2018.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021.

Nie, X., Zhao, P., Miao, X., Zhao, T., and Cui, B. Hetumoe: An efficient trillion-scale mixture-of-expert distributed training system, 2022. URL `https://arxiv.org/abs/2203.14685`.

NVIDIA. NCCL Tests. `https://github.com/NVIDIA/nccl-tests`, 2016-2024. Accessed: 2024-12-06.

NVIDIA. Massively scale your deep learning training with nccl 2.4. `https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4/`, 2019. Accessed: 2024-11-23.

NVIDIA. NVIDIA Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) v2.6.1 Release Notes. Technical Report Revision 2.6.1, NVIDIA, 2023. URL `https://docs.nvidia.com/networking/display/sharpv261/release+notes`. Last updated on May 23, 2023.

NVIDIA. TensorRT-LLM. GitHub repository, 2023. URL `https://github.com/NVIDIA/TensorRT-LLM`.

NVIDIA. Megatron-LM: Ongoing research training transformer models at scale. GitHub repository, 2024a. URL `https://github.com/NVIDIA/Megatron-LM`.

NVIDIA. NVIDIA Nsight Systems. Web Page, 2024b. URL https://developer.nvidia.com/nsight-systems.

NVIDIA. Collective operations. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html, 2024a. Accessed: 2024-11-23.

NVIDIA. Context parallelism overview. https://docs.nvidia.com/megatron-core/developer-guide/latest/api-guide/context_parallel.html, 2024b. Accessed: 2024-11-23.

NVIDIA. Nvidia l40: Delivering unprecedented visual computing performance for the data center. https://images.nvidia.cn/content/Solutions/data-center/vgpu-L40-datasheet.pdf, 2024c. Accessed: 2024-11-23.

NVIDIA. Optimized primitives for collective multi-gpu communication. https://github.com/NVIDIA/nccl, 2024d. Accessed: 2024-11-23.

NVIDIA. Nvlink & nvswitch for advanced multi-gpu communication. https://www.nvidia.com/en-us/data-center/nvlink/, 2024e. Accessed: 2024-11-26.

NVIDIA. Cuda templates for linear algebra subroutines, 2024f. URL https://github.com/NVIDIA/cutlass. Accessed: 2024-12-04.

NVIDIA Corporation. Peer Device Memory Access. Technical report, NVIDIA, 2024. URL https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__PEER.html. Accessed: 2024-12-04.

Qin, M., Sun, C., Hofmann, J., and Vucinic, D. Disco: Distributed inference with sparse communications. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2432–2440, 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–16. IEEE, 2020.

Rajbhandari, S., Li, C., Yao, Z., Zhang, M., Aminabadi, R. Y., Awan, A. A., Rasley, J., and He, Y. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

Rouhani, B. D., Zhao, R., More, A., Hall, M., Khodamoradi, A., Deng, S., Choudhary, D., Cornea, M., Dellinger, E., Denolf, K., et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Sanders, P., Speck, J., and Träff, J. L. Two-tree algorithms for full bandwidth broadcast, reduction and scan. *Parallel Computing*, 35(12):581–594, 2009.

Sergeev, A. and Balso, M. D. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*, 2018.

sgl-project. SGLang: A Fast Serving Framework for Large Language Models and Vision Language Models. GitHub repository, 2024. URL https://github.com/sgl-project/sglang. Accessed: 2024-12-01.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Team, Q. Qwen1.5-moe: Matching 7b model performance with 1/3 activated parameters", February 2024. URL https://qwenlm.github.io/blog/qwen-moe/.

Wang, G., Qin, H., Jacobs, S. A., Holmes, C., Rajbhandari, S., Ruwase, O., Yan, F., Yang, L., and He, Y. Zero++: Extremely efficient collective communication for giant model training. *arXiv preprint arXiv:2306.10209*, 2023.

Wang, G., Zhang, C., Shen, Z., Li, A., and Ruwase, O. Domino: Eliminating communication in llm training via generic tensor slicing and overlapping, 2024. URL https://arxiv.org/abs/2409.15241.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. Smoothquant: Accurate and efficient post-training quantization for large language models, 2024. URL https://arxiv.org/abs/2211.10438.

Ying, C., Kumar, S., Chen, D., Wang, T., and Cheng, Y. Image classification at supercomputer scale. *arXiv preprint arXiv:1811.06992*, 2018.

Yu, M., Wang, D., Shan, Q., and Wan, A. The super weight in large language models. *arXiv preprint arXiv:2411.07191*, 2024.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, T., Lin, Z., Yang, G., and Sa, C. D. Qpytorch: A low-precision arithmetic simulation framework, 2019.

Zhao, Y., Gu, A., Varma, R., Luo, L., Huang, C.-C., Xu, M., Wright, L., Shojanazeri, H., Ott, M., Shleifer, S., et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Zhu, K., Zhao, Y., Zhao, L., Zuo, G., Gu, Y., Xie, D., Gao, Y., Xu, Q., Tang, T., Ye, Z., Kamahori, K., Lin, C.-Y., Wang, S., Krishnamurthy, A., and Kasikci, B. Nanoflow: Towards optimal large language model serving throughput, 2024. URL https://arxiv.org/abs/2408.12757.

A. Background

A.1. GPU Topology

Modern inference GPUs are connected via various hardware configurations. Figure 12 shows a typical simplified physical topology where every node contains 8 GPUs. Every two adjacent GPUs are connected via a PCIe of 64GB/s bandwidth (NVIDIA, 2024c). Cross-GPU communication may take several different paths, e.g. GPU 0 to 1 has the shortest route, but from GPU 0 to 4 it has to go across two NUMA (Non-uniform memory access) nodes. Cross-node communication relies on NICs (Network interface cards) that transmit data via Ethernet, whose bandwidth is usually 100Gbps.

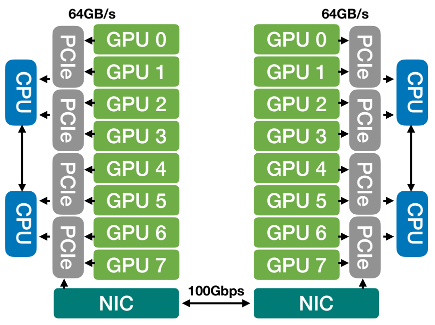


Figure 12. Physical topology of two NVIDIA 8× L40 GPU nodes connected for inference. Each node has 8 GPUs interconnected with PCI Switches and 2 NUMA nodes. For simplicity, NIC is shown to only connect with the last PCI.

For high-performance large-scale training, high-end GPUs like A100 SXM GPUs enjoy a much wider bandwidth due to the combination use of NVLink (NVIDIA, 2024e), NVSwitch, and InfiniBand NIC. Each GPU in the same node directly connected with all other GPUs could reach 600GB/s via NVSwitch, and inter-node communication could have a bandwidth of 200Gbps. These advanced hardware configurations tremendously accelerate the training of large language models.

Warp scheduling is critical for GPU utilization. L40 is shipped with 142 streaming multiprocessors (SM) while A100 has 108. A warp consists of a group of 32 threads, which is the minimum scheduling unit for SM. Multiple warps can be simultaneously executed on an SM.

A.2. Collective Communication

Libraries. Due to the hierarchical design of GPU clusters, collective communication methods are crucial in distributed training and inference. To synchronize the workloads across GPUs, communication libraries like NCCL (NVIDIA, 2024d), MSCCL (Microsoft, 2024), HiCCL (Hidayetoglu et al., 2024), Gloo (Facebook, 2024), and Horovod (Sergeev & Balso, 2018) are developed to provide efficient collective communication operations for a group of devices. These libraries usually hide the physical topology and organize GPUs in a ring (Mikami et al., 2018; Jia et al., 2018; Ying et al., 2018) or a tree. In ring-based topology, GPUs are connected hand by hand to create a logical circle, which maximizes the utilization of the full bandwidth. In contrast, tree-based topology, especially double binary tree (Sanders et al., 2009), guarantees logarithmic communication hops. Therefore, it is more beneficial to use ring-based communication for intra-GPUs and a tree-based approach for inter-GPU clusters.

Operations. Collective operations such as broadcast, aggregation (Reduce/All-Reduce/Reduce-Scatter), collection (Gather/All-Gather), and All2All are shipped out-of-box in most collective communication libraries. For instance, NCCL provides a series of such collective operations (NVIDIA, 2024a) where each rank processes or transmits the same amount of data. Reduce-Scatter sums data across nodes and then scatters the result to corresponding nodes. All-Gather collects data from all nodes to all nodes. All-Reduce is a many-to-many reduce operation, where the same reduce operation is applied on all nodes. All2All exchanges data between all nodes, with each node sending and receiving an equal amount of data. All2All can be implemented with multiple point-to-point communication operations.

A.3. Quantization Fundamentals

Quantization is a mapping from floating numbers to integers. We utilize asymmetric quantization which is formulated below,

$$s = \frac{X_{max} - X_{min}}{2^n - 1}, z = \lceil \frac{-X_{min}}{s} \rceil \tag{1}$$

$$Q(X) = clamp(\lceil X/s \rceil + z, 0, 2^n - 1) \tag{2}$$

where X_{max} and X_{min} denotes the maximum and minimum value of X, n is the quantization bit-width, s is called the scale and z the zero point. $Q(x)$ quantizes float X to integer to the target bitwidth.

Symmetric quantization is formulated as follows,

$$s = \frac{|X|_{max}}{2^{n-1} - 1} \tag{3}$$

$$Q(X) = clamp(\lceil X/s \rceil, -2^{n-1}, 2^{n-1} - 1) \tag{4}$$

IEEE 754 standards for FP16. IEEE 754 (IEEE, 1985) FP16 includes 16 bits in total, which comprises 1 bit for the sign (S), 5 bits for the exponent (E), and 10 bits for the mantissa or fraction (F). The bias for the exponent is 15, which means that the actual exponent value must be added to 15 to get the stored exponent value. Also, notice there's an assumed leading 1 in the fractional part.

FP8 and FP4 Format. FP8 (Micikevicius et al., 2022) format is designed to advance FP16 with two encodings, E4M3 (4-bit exponent and 3-bit mantissa) and E5M2 (5-bit exponent and 2-bit mantissa). E5M3 follows IEEE 754 conventions. FP4 (Rouhani et al., 2023) is of E2M1. For quantization to FP4, we utilize QPyTorch (Zhang et al., 2019) for simulation.

B. Additional Experiments

B.1. C4 and WikiText

The C4 and WikiText perplexity of FP16-weight LLaMA models is given in Table 6 while the INT8-weight version is shown in Table 7. Both communication volumes are quantized with Flash communication.

Table 6. LLaMA models' perplexity of C4 (upper rows) and WikiText2 (lower rows) with fine-grained communication quantization with a group size of 128.

MODEL	INT8$_{Asym}$	FP8	INT6$_{Asym}$	INT4$_{Asym}$
3-8B	8.89	8.90	9.20	9.68
3-70B	6.74	6.75	6.85	7.04
2-7B	6.98	6.98	7.07	7.21
2-13B	6.47	6.47	6.50	6.58
2-70B	5.52	5.52	5.55	5.60
3-8B	6.14	6.15	6.37	6.70
3-70B	2.86	2.87	3.00	3.21
2-7B	5.47	5.48	5.55	5.66
2-13B	4.88	4.89	4.93	4.99
2-70B	3.32	3.32	3.35	3.40

C. Additional Latency Measurements

We list the latency measurements of LLaMA models under various configurations (weight precision, tensor parallelism, GPU cards) in the following Fig. 13 and Fig. 14.

Table 7. The 8-bit LLaMA models' perplexity of C4 (upper rows) and WikiText2 (lower rows) with fine-grained communication quantization with a group size of 128.

MODEL	INT8$_{Asym}$	FP8	INT6$_{Asym}$	INT4$_{Asym}$
LLAMA-2-7B	7.00	7.01	7.10	7.24
LLAMA-2-13B	6.51	6.51	6.55	6.63
LLAMA-2-70B	5.54	5.54	5.57	5.62
LLAMA-3-8B	9.01	9.02	9.33	9.85
LLAMA-3-70B	6.82	6.83	6.95	7.13
LLAMA-2-7B	5.50	5.51	5.59	5.69
LLAMA-2-13B	4.92	4.92	4.97	5.03
LLAMA-2-70B	3.35	3.35	3.39	3.43
LLAMA-3-8B	6.25	6.26	6.49	6.84
LLAMA-3-70B	2.96	2.97	3.13	3.32

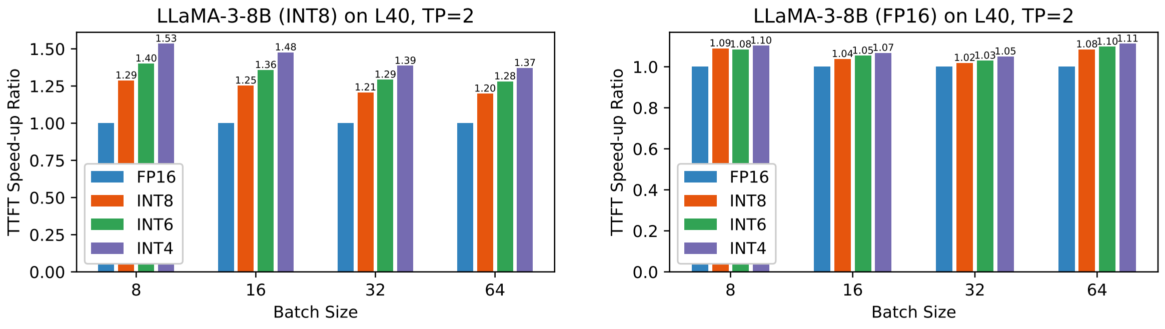


Figure 13. TTFT speed-up ratio of LLaMA-3-8B (INT8 vs. FP16) under various communication quantization bit widths on L40 with TP=2.

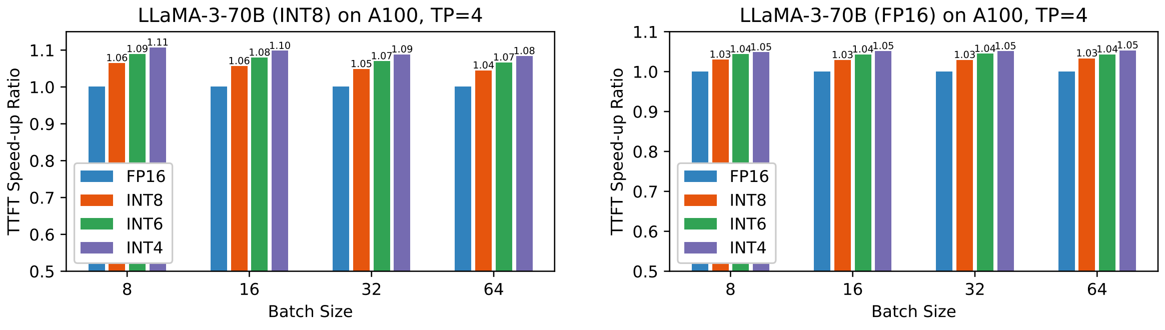


Figure 14. TTFT speed-up ratio of LLaMA-3-70B (INT8 vs. FP16) under various communication quantization bit widths on A100 with TP=4.