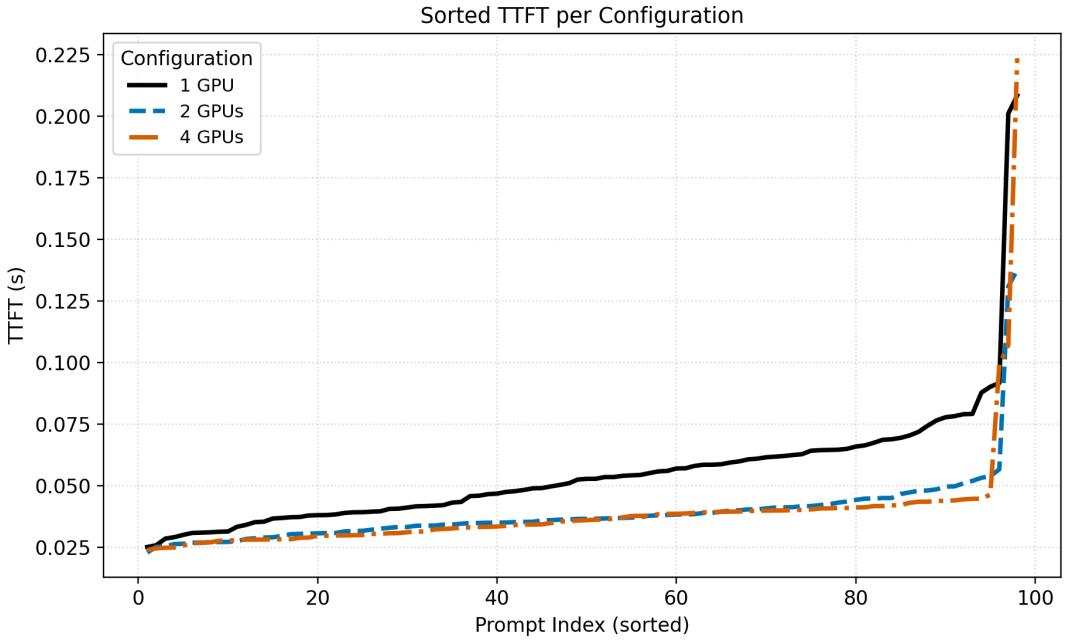
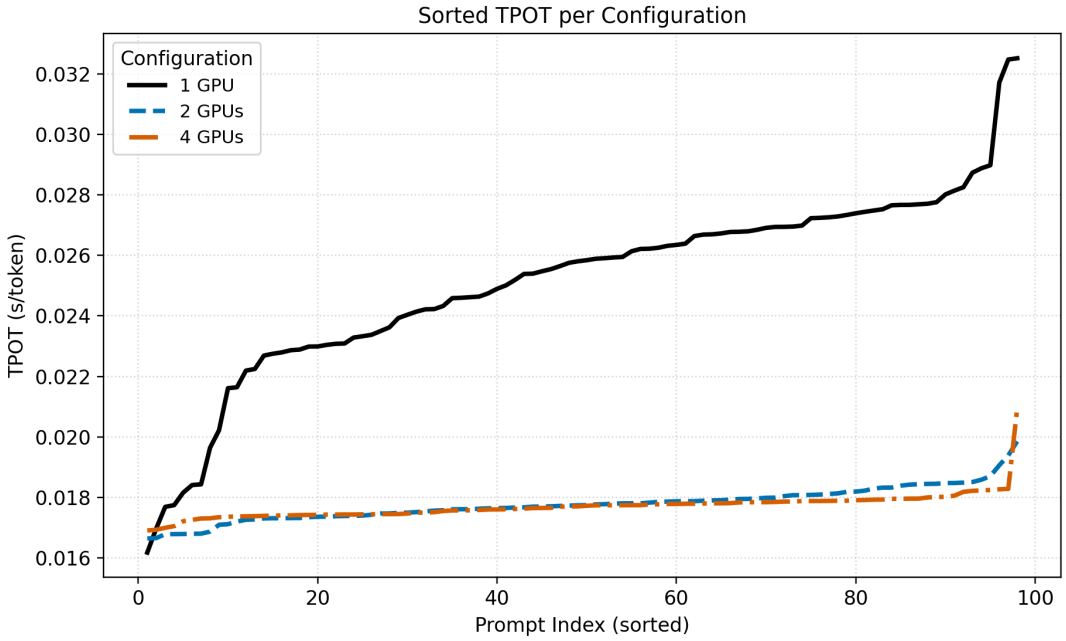
Figure of TTFT and TPOT discussion





The evaluation of TPOT (time per output token) and TTFT (time to first token) across different GPU configurations using vLLM with Llama-3.1-8B shows that a single GPU suffers from significantly higher latency, with both TPOT and TTFT increasing sharply as prompt length grows, making it impractical for real-time or long-context workloads.

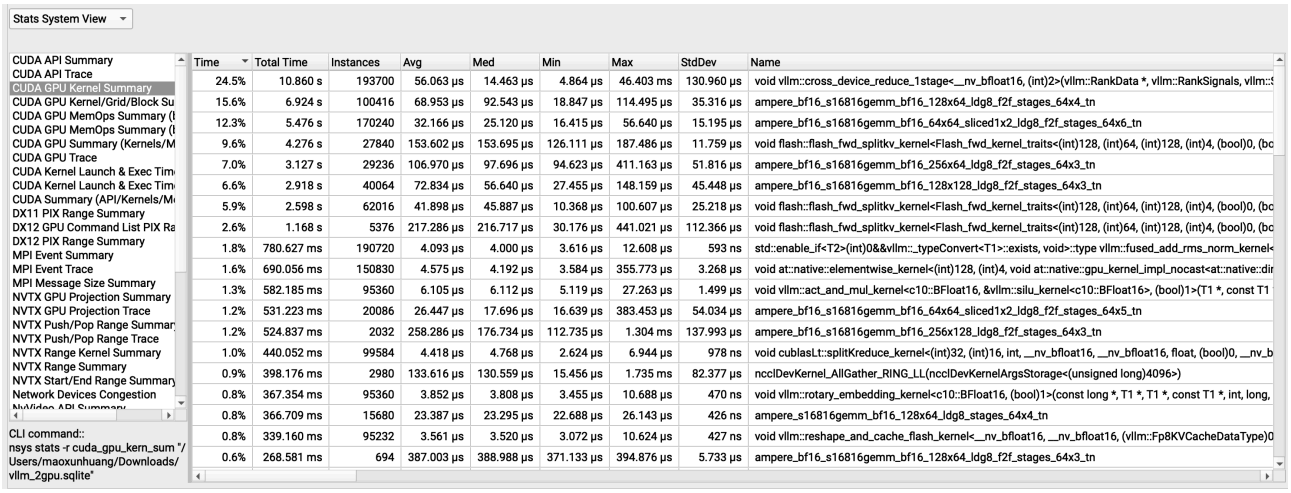
In contrast, scaling to two GPUs provides a substantial improvement, flattening TPOT around 0.018–0.020 s/token and keeping TTFT consistently low, demonstrating that tensor parallelism is highly effective at this scale.

Moving from two to four GPUs yields only marginal gains, with slight stability improvements for very long prompts but limited benefits for short and medium ones, reflecting the diminishing returns caused by communication overhead.

Overall, two GPUs emerge as the most cost-effective configuration, offering near-optimal throughput and latency, while four GPUs are only advantageous for serving extremely long prompts or high-throughput scenarios.

An interesting observation from the results is that when the prompt length becomes very long, the curves for 2 GPUs and 4 GPUs cross, indicating that 2 GPUs can actually achieve lower TTFT and TPOT than 4 GPUs in this regime. This suggests that the communication and synchronization overhead introduced by scaling across more devices may outweigh the computational benefits, making 2 GPUs more efficient than 4 GPUs for extremely long prompts.

Table with Nsight profiling results showing top 3 kernel names and times.



Stats System View

CUDA API Summary
CUDA API Trace
CUDA GPU Kernel Summary
CUDA GPU Kernel/Grid/Block Su
CUDA GPU MemOps Summary (I
CUDA GPU MemOps Summary (I
CUDA GPU Summary (Kernels/M
CUDA GPU Trace
CUDA Kernel Launch & Exec Tim
CUDA Kernel Launch & Exec Tim
CUDA Summary (API/Kernels/M
DX11 PIX Range Summary
DX12 GPU Command List PIX Ra
DX12 PIX Range Summary
MPI Event Summary
MPI Event Trace
MPI Message Size Summary
NVTX GPU Projection Summary
NVTX GPU Projection Trace
NVTX Push/Pop Range Summar
NVTX Push/Pop Range Trace
NVTX Range Kernel Summary
NVTX Range Summary
NVTX Start/End Range Summary
Network Devices Congestion

CLI command::
nsys stats -r cuda_gpu_kern_sum "/
Users/maoxunhuang/Downloads/
vllm_2gpu.sqlite"

Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Name
24.5%	10.860 s	193700	56.063 µs	14.463 µs	4.864 µs	46.403 ms	130.960 µs	void vllm::cross_device_reduce_1stage<__nv_bfloat16, (int)2>(vllm::RankData *, vllm::RankSignals, vllm::S
15.6%	6.924 s	100416	68.953 µs	92.543 µs	18.847 µs	114.495 µs	35.316 µs	ampere_bf16_s16816gemm_bf16_128x64_ldg8_f2f_stages_64x4_tn
12.3%	5.476 s	170240	32.166 µs	25.120 µs	16.415 µs	56.640 µs	15.195 µs	ampere_bf16_s16816gemm_bf16_64x64_sliced1x2_ldg8_f2f_stages_64x6_tn
9.6%	4.276 s	27840	153.602 µs	153.695 µs	126.111 µs	187.486 µs	11.759 µs	void flash::flash_fwd_splitkv_kernel<Flash_fwd_kernel_traits<(int)128, (int)64, (int)128, (int)4, (bool)0, (bo
7.0%	3.127 s	29236	106.970 µs	97.696 µs	94.623 µs	411.163 µs	51.816 µs	ampere_bf16_s16816gemm_bf16_256x64_ldg8_f2f_stages_64x3_tn
6.6%	2.918 s	40064	72.834 µs	56.640 µs	27.455 µs	148.159 µs	45.448 µs	ampere_bf16_s16816gemm_bf16_128x128_ldg8_f2f_stages_64x3_tn
5.9%	2.598 s	62016	41.898 µs	45.887 µs	10.368 µs	100.607 µs	25.218 µs	void flash::flash_fwd_splitkv_kernel<Flash_fwd_kernel_traits<(int)128, (int)64, (int)128, (int)4, (bool)0, (bo
2.6%	1.168 s	5376	217.286 µs	216.717 µs	30.176 µs	441.021 µs	112.366 µs	void flash::flash_fwd_splitkv_kernel<Flash_fwd_kernel_traits<(int)128, (int)64, (int)128, (int)4, (bool)0, (bo
1.8%	780.627 ms	190720	4.093 µs	4.000 µs	3.616 µs	12.608 µs	593 ns	std::enable_if<T2>(int)0&&vllm::_typeConvert<T1>::exists, void>::type vllm::fused_add_rms_norm_kernel<
1.6%	690.056 ms	150830	4.575 µs	4.192 µs	3.584 µs	355.773 µs	3.268 µs	void at::native::elementwise_kernel<(int)128, (int)4, void at::native::gpu_kernel_impl_nocast<at::native::dir
1.3%	582.185 ms	95360	6.105 µs	6.112 µs	5.119 µs	27.263 µs	1.499 µs	void vllm::act_and_mul_kernel<c10::BFloat16, &vllm::silu_kernel<c10::BFloat16>, (bool)1>(T1 *, const T1
1.2%	531.223 ms	20086	26.447 µs	17.696 µs	16.639 µs	383.453 µs	54.034 µs	ampere_bf16_s16816gemm_bf16_64x64_sliced1x2_ldg8_f2f_stages_64x5_tn
1.2%	524.837 ms	2032	258.286 µs	176.734 µs	112.735 µs	1.304 ms	137.993 µs	ampere_bf16_s16816gemm_bf16_256x128_ldg8_f2f_stages_64x3_tn
1.0%	440.052 ms	99584	4.418 µs	4.768 µs	2.624 µs	6.944 µs	978 ns	void cublasLt::splitKreduce_kernel<(int)32, (int)16, int, __nv_bfloat16, __nv_bfloat16, float, (bool)0, __nv_b
0.9%	398.176 ms	2980	133.616 µs	130.559 µs	15.456 µs	1.735 ms	82.377 µs	ncclDevKernel_AllGather_RING_LL(ncclDevKernelArgsStorage<(unsigned long)4096>)
0.8%	367.354 ms	95360	3.852 µs	3.808 µs	3.455 µs	10.688 µs	470 ns	void vllm::rotary_embedding_kernel<c10::BFloat16, (bool)1>(const long *, T1 *, T1 *, const T1 *, int, long,
0.8%	366.709 ms	15680	23.387 µs	23.295 µs	22.688 µs	26.143 µs	426 ns	ampere_s16816gemm_bf16_128x64_ldg8_stages_64x4_tn
0.8%	339.160 ms	95232	3.561 µs	3.520 µs	3.072 µs	10.624 µs	427 ns	void vllm::reshape_and_cache_flash_kernel<__nv_bfloat16, __nv_bfloat16, (vllm::Fp8KVCacheDataType)0
0.6%	268.581 ms	694	387.003 µs	388.988 µs	371.133 µs	394.876 µs	5.733 µs	ampere_bf16_s16816gemm_bf16_128x64_ldg8_f2f_stages_64x3_tn

From the Nsight Systems statistics, the top three CUDA kernels consuming the most execution time are: **vllm::cross_device_reduce_1stage (24.5%)**, which handles cross-device all-reduce communication for tensor parallelism, and two BF16 GEMM compute kernels, **ampere_bf16_s16816gemm_bf16_128x64 (15.6%)** and **ampere_bf16_s16816gemm_bf16_64x64** (12.3%), which correspond to matrix multiplications in the transformer layers. This indicates that a large portion of runtime is split between communication overhead from multi-GPU synchronization and heavy linear algebra operations, reflecting the typical bottlenecks in large-scale LLM inference.

Identification (Screenshot) of the All-reduce kernel responsible for tensor parallel communication during multi-GPU inference

On NVIDIA GPUs, AllReduce is typically implemented by **NCCL (NVIDIA Collective Communications Library)**.

Difficulties or surprising results

One surprising result from this assignment was the observation that **two GPUs occasionally outperformed four GPUs** in both TTFT and TPOT when handling very long prompts. Intuitively, adding more GPUs should always reduce computation time, but the profiling results revealed that the **communication and synchronization overhead** introduced at larger scales can offset these gains. Another difficulty was correctly setting up Nsight Systems to capture meaningful CUDA events, since improper configuration initially produced empty traces. Once resolved, the profiling made it clear that performance bottlenecks are not only in the compute-intensive GEMM kernels

but also in the **all-reduce communication kernel**, highlighting the challenges of balancing computation and communication efficiency in large-scale LLM inference