A NUMA Aware Compiler Framework for Large Scale Mathematical Reasoning Inference on PCIe Based Multi Accelerator Systems

JooHyoung Cha
UST
South Korea
jh.cha@etri.re.kr

Yongin Kwon
ETRI
South Korea
yongin.kwon@etri.re.kr

Abstract

Mathematical reasoning workloads proof search, program verification, equation solving, code as proof traces, and tool augmented LLM pipelines demand long context decoding, speculative/beam search, and mixture of experts, which induce frequent collectives under model/tensor parallelism. In commodity dual socket NUMA servers with PCIe interconnects, non uniform link bandwidth/latency and host mediated cross socket routes make these collectives the bottleneck, inflating end to end latency that matters for interactive theorem proving and education at scale. We present a NUMA aware compiler framework for large scale math reasoning inference. The system profiles compute and memory paths, learns a latency bandwidth cost model for hierarchical collectives, and jointly optimizes data, model, and tensor partitioning along with device memory placement under static feasibility constraints. Using MLIR/TOSA templates, it emits host and accelerator code with explicit comm–compute overlap and schedule shaping via ring, tree, and hybrid schemes, without relying on vendor specific fabrics. We target math AI pipelines such as LLMs with solver or tool use and prover trace generation, and We outline ablations to isolate how profiling, static analysis, schedule choice, and overlap affect throughput and p95 latency.

1 Introduction

Math AI at scale. Recent LLM advances have unlocked automated and assisted mathematical reasoning: multi step chain of thought, self verification and solver calls, program synthesis for proofs, and interaction with proof assistants. These pipelines mix long context decoding, speculative/beam search, and mixture of experts (MoE) gating, often exceeding the capacity/latency targets of a single accelerator and requiring model/tensor parallelism at inference time.

Systems gap. While GPU centric collective communication libraries (CCL) perform well on uniform, high bandwidth fabrics, many lab and production deployments for math AI run on PCIe based multi-accelerator servers with dual socket non-uniform memory access (NUMA) topologies, where inter socket traffic traverses CPU fabrics. The resulting non-uniform bandwidth/latency and host mediated transfers make collective communication (including KV cache exchange, expert routing, and partial result aggregation) a dominant bottleneck especially harmful for interactive theorem proving and classroom settings that are latency sensitive.

Key observation. Effective math reasoning inference on NUMA systems requires *hierarchical* schedules (complete fast intra socket collectives before a minimized inter socket step) and *overlap aware* execution (interleave comm with compute). Because decoding strategies (e.g., speculative

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025.

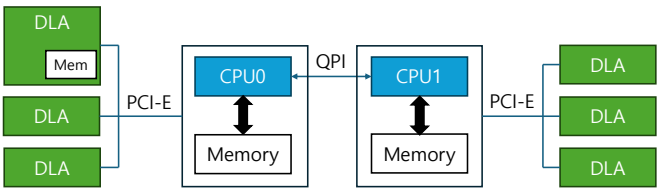

Figure 1: Typical topology of a PCIe based multi DLA NUMA system.

or beam search) change traffic patterns and tensor shapes over time, the *schedule shape*, *chunking*, *overlap policy*, and *NUMA aware placement* must be co-designed.

This work. We introduce a compiler assisted framework that is NUMA aware by construction and tailored to mathematical reasoning inference on PCIe/QPI-class servers. The framework:

- calibrates a portable latency bandwidth cost model for collectives and NUMA paths,
- **co-optimizes** partitioning and device memory placement under memory/concurrency limits,
- generates MLIR (TOSA) templates with explicit communication, compute, and synchronization,
- prunes infeasible configurations via static analysis (e.g., OOM, DMA limits), and
- runs feedback driven search to emit overlapped host/accelerator code with ring, tree, and hybrid schedules.

2 Background and Motivation

2.1 Collectives on NUMA topologies

Figure 1 shows a common two socket NUMA system with six deep learning accelerators (DLA). Intra socket transfers may leverage peer to peer direct memory access (DMA), while inter socket traffic typically traverses QPI with CPU involvement [1]. This asymmetry necessitates *hierarchical* schedules that (1) complete *intra node* collectives first and (2) perform a minimized *inter node* step, often with smaller payloads. Typical operations include BROADCAST, REDUCE, ALLREDUCE, GATHER, and SCATTER. Prior work has extensively studied optimization of collective operations in MPI !and hierarchical fabrics [2, 3, 4], and more recent work explores topology aware or template guided AllReduce in deep learning systems [5, 6, 7].

2.2 Parallelism and overlap

Data parallelism replicates the model and communicates gradients/activations; model/tensor parallelism partitions layers/weights and typically induces more frequent exchanges. Effective systems overlap collective stages with computation to hide latency. Practical distributed training frameworks such as Horovod [8], BytePS [9], and large scale model parallel systems like Megatron LM [10] all demonstrate the importance of minimizing communication overheads and overlapping communication with computation.

3 System Model and Problem Formulation

Let devices (DLAs/CPUs) be nodes $\mathcal{A}$ and directed links be $\mathcal{E}$. Each link $e \in \mathcal{E}$ has bandwidth BW_e and latency α_e. Transferring a message of size sz over e costs

$$T_e(\mathsf{sz}) \;=\; \alpha_e \;+\; \frac{\mathsf{sz}}{\mathsf{BW}_e}. \tag{1}$$

This latency bandwidth formulation is standard in the analysis of collective algorithms [4, 2, 3]. For a path P (e.g., a host mediated inter socket route), $T_P(\mathsf{sz}) = \sum_{e \in P} T_e(\mathsf{sz})$. For a collective schedule $\mathcal{S}$ (e.g., RING, TREE, hierarchical RING$\rightarrow$TREE), the total is

$$T_{\mathcal{S}}(\mathbf{m}) \;=\; \sum_{k} \max_{p \in \mathcal{P}_k} T_p\big(\mathsf{sz}_k\big), \tag{2}$$

where stage k transmits chunk sz_k over a set of concurrent paths $\mathcal{P}_k$.

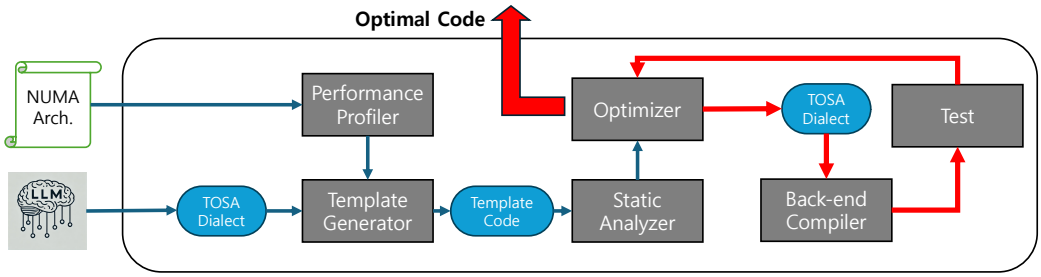

Figure 2: Proposed compiler framework: profiling, MLIR/TOSA based template generation, static feasibility analysis, and feedback driven optimization.

Objective. Given a model graph, partitioning choice π (data/model/tensor), placement ρ (device/memory/NUMA node), and a schedule $\mathcal{S}$ for each collective, minimize wall-clock latency $T_{\text{end-to-end}}(\pi, \rho, \mathcal{S})$ subject to memory, DMA, and stream concurrency constraints.

4 Framework Overview

Figure 2 outlines the compiler based framework. It consists of a NUMA aware profiler, a template generator, a static analyzer, and an optimizer.

NUMA aware Performance Profiler. We calibrate $\{\alpha_e, \text{BW}_e\}_{e \in \mathcal{E}}$ using microbenchmarks covering DMA sizes/patterns (H2D, D2H, D2D), and measure compute kernels (GEMM, nonlinears, reduce). Inter socket routes are characterized via QPI. The profiler yields per-link fits for (1), which feed both analysis and search.

Template Generator (MLIR/TOSA). Using MLIR [11] and the TOSA dialect [12], we lower models to an IR that *explicitly* marks (i) collective ops and (ii) buffer locations (host vs. DLA memory). The generator emits host/DLA code stubs with repeatable inference loops, communication/computation streams, and barriers. Partitioning knobs include: (a) weights resident on host vs. DLA, (b) tensor shard sizes and NUMA node affinity for data parallelism, (c) tensor dimensions and nodes for model parallelism, (d) redundancy level for replicated weights. Similar ideas of exposing collectives as first class IR constructs have appeared in DSL based approaches such as MSCCLang [13], and template guided synthesis techniques like TACCL [7] also motivate our template driven design.

Static Analyzer. Given a parameter tuple $(\pi, \rho, \mathcal{S})$, the analyzer checks: memory capacity per NUMA node/DLA; DMA engine limits; alignment and maximum transfer sizes; stream counts; and synchronization feasibility. It returns tightened domains to the optimizer, eliminating infeasible or clearly dominated regions.

5 Optimization Details

We cast schedule/placement selection as constrained blackbox optimization with fast model based estimates and measured feedback, following prior work that has explored automated synthesis and search of collective algorithms [6, 7, 5].

5.1 Parameterization

- **Collective schedule** per phase: RING, TREE, or hierarchical (RINGintra node, TREEinter node).
- **Chunking**: number/size of pipeline chunks per tensor.
- **Overlap policy**: degree of comm/compute overlap (streams, prefetch distance).
- **Placement**: NUMA node for inputs/weights/activations; replica vs. shard decisions.
- **Partitioning**: data vs. model/tensor parallel split ratios.

Algorithm 1 Feedback driven schedule/placement search

Initialize policy π_θ with cost model priors
for $t = 1, \ldots, T$ **do**
  Sample $(\pi, \rho, \mathcal{S}) \sim \pi_\theta(\cdot \mid \text{features})$
  if STATICANALYZE rejects $(\pi, \rho, \mathcal{S})$ **then continue**
  end if
  Generate code via MLIR templates; compile (host/DLA)
  Run inference micro batches; measure latency L_t, tokens/s R_t
  Update $\pi_\theta \leftarrow$ IMPROVE$(\pi_\theta; -L_t$ or $R_t)$
end for
return best configuration

5.2 Searcher

We use a lightweight reinforcement learning loop (e.g., contextual bandit) with state features derived from (i) per-link α/BW, (ii) current partition sizes, (iii) model layer types, and (iv) analyzer derived headroom (memory, streams). The action is the parameter tuple; the reward is negative latency (or tokens/s). A fitted model (e.g., linear/UCB or shallow network) warm starts from the analytic cost model to reduce trials. This feedback driven approach is inspired by recent collective synthesis systems that combine analytic models with empirical search to converge to efficient schedules [6, 7].

6 Evaluation Plan

We outline an evaluation methodology; concrete numbers depend on hardware availability.

- **Platforms:** Dual socket NUMA server with six DLAs over PCIe (as in Figure 1); DRAM backed accelerators (no HBM) to stress PCIe/QPI paths.
- **Workloads:** LLM inference (decoder only, FP16/bfloat16), plus operator level microbenchmarks (GEMM + ALLREDUCE/BROADCAST). To contextualize our workloads, we follow prior distributed DL benchmarks that evaluate collectives in the context of large scale training and inference, such as Horovod [8], BytePS [9], and Megatron LM [10].
- **Baselines:** NCCL [14], oneCCL [15], RCCL [16] where applicable; vendor SDK default schedules; naive host staged collectives. These are compared against practical distributed training frameworks (e.g., Horovod and BytePS) that have become common baselines in recent evaluations [8, 9].
- **Metrics:** End to end latency per token and tokens/s; p95 latency; PCIe/QPI utilization; CPU involvement overhead; memory footprint.
- **Ablations:** (A1) cost model warm start vs. random; (A2) hierarchical vs. flat schedules; (A3) overlap on/off; (A4) placement only vs. placement+partitioning; (A5) static analyzer enabled/disabled.
- **Reporting:** For each workload, report best configuration and budget (number of trials), with Pareto curves (latency vs. memory).

7 Conclusion and Future Work

We introduced a NUMA aware, compiler based framework that co-optimizes partitioning, placement, and collective schedules for inference on PCIe-/QPI-class multi accelerator systems. By combining calibrated cost models, MLIR/TOSA code generation, static feasibility checks, and feedback driven search, our approach improves efficiency without relying on vendor specific interconnects. Although demonstrated on decoding centric workloads, the methodology also applies to latency sensitive domains such as mathematical reasoning with long context decoding and KV cache traffic.

Future work includes adding runtime adaptivity under dynamic contention, extending support to more accelerators and interconnects, and co-designing with math AI decoding strategies. We also plan to incorporate energy/cost objectives and lightweight verification of collectives, making large scale reasoning pipelines more practical on commodity NUMA servers.

Acknowledgement

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP)grant funded by the Korea government (MSIT) (No.RS-2024-00459797, No.RS-2023-00277060, No.RS-2025-02217404, No.RS-2025-02214497, No.RS-2025-02216517)

References

[1] Intel. https://www.intel.com/content/www/us/en/io/quickpath-technology/quickpath-technology-general.html.

[2] Rolf Rabenseifner. Optimization of collective reduction operations. In *Computational Science – ICCS 2004*, volume 3036 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2004.

[3] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *The International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[4] Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2):117–124, 2009.

[5] Minsik Cho, Ulrich Finkler, Mauricio Serrano, David Kung, and Hillery Hunter. Blueconnect: Decomposing all-reduce for deep learning on heterogeneous network hierarchy. In *Proceedings of the 2nd SysML Conference (now MLSys)*, 2019.

[6] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '21)*, 2021.

[7] Aashaka Shah, Abhinav Jangda, Binyang Li, Caio Rocha, Changho Hwang, Jithin Jose, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. Taccl: Guiding collective algorithm synthesis using traffic-optimized templates. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023.

[8] Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in tensorflow. *arXiv preprint arXiv:1802.05799*, 2018.

[9] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 463–479, 2020.

[10] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[11] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14. IEEE, 2021.

[12] LLVM. https://mlir.llvm.org/docs/dialects/tosa/.

[13] Meghan Cowan, Saeed Maleki, Madanlal Musuvathi, Olli Saarikivi, and Yifan Xiong. Mscclang: Microsoft collective communication language. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23)*, 2023.

[14] NVIDIA. https://github.com/nvidia/nccl.

[15] Intel. https://github.com/oneapi-src/oneccl.

[16] AMD. https://github.com/rocm/rccl.