OpenCCL™: A Proposed Open Standard and Reference Implementation

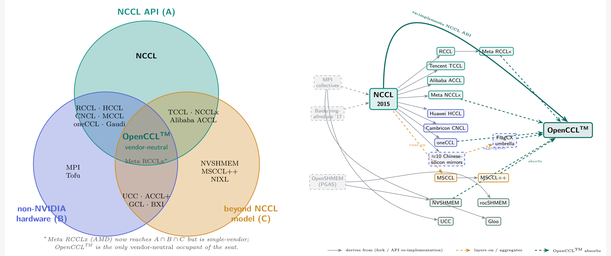
Unifying *CCLs for Performance, Portability, and Productivity

Anthony Skjellum[1] Purushotham V. Bangalore[2] Thomas Hines[1] Kayla Humphrey[1] Javontae Martin[1] Amr Akmal Abouelmagd[1] Riley Shipley[1] Rajat Bisht[1] Zulkar Nine[3]
[1]ASCEND and Dept. of Computer Science, Tennessee Technological University [2]Dept. of Computer Science, University of Alabama [3]Arkansas Tech University

GitHub

Latest poster

*CCL Field — Motivating OpenCCL™ Based on State of the Art



No vendor-neutral library has all three properties

A diverse set of libraries—OpenCCL™ will unify

The problem: vendor lock-in and limited creativity for applications and the community

State of the art:

- *CCL — the communication layer under a distributed AI framework: point-to-point, collective, accelerator-optimized.
- NCCL [1], the de-facto CCL, modeled on MPI: NVIDIA's GPU APIs, ported as AMD's RCCL [2], forked by vendors (Meta's NCCLx/RCCLx [3]).
- **NCCL is a toolkit, not a specification** — thin semantics, much unsaid, **not vendor-neutral.**

Key issues addressed:

- **One vendor sets the pace** — the API advances when NVIDIA chooses; others reimplement or fork. Supports InfiniBand.
- New fabrics, accelerators, workloads — all lose to baseline-NCCL portability.
  - Extension libraries: good ideas, no broad platform support.
  - No community path from best practice to supported feature.

OpenCCL™ follows MPI's proven approach

- MPI-1.0 (1993–94) replaced many incompatible message-passing dialects with one open standard [4] and an open reference implementation, MPICH [5].
- Applications became portable, reusable libraries burgeoned, and software investments survived generations of machines while vendors specialized behind a stable interface.
- MPI is an evolving standard, decades of fielded experience, dedicated contributors, ubiquitous.
- **OpenCCL™ seeks to follow the successful model of the MPI Forum and the MPICH reference implementation.**

Status of Implementation and Support



- **Current test platforms include:** LLNL Tuolumne and Matrix, Purdue Anvil, UIUC Delta, and DGX Spark Cluster.
- **Multi-node on AMD over Slingshot-11**: on **LLNL Tuolumne** (MI300A), RAPIDS native **CXI** path
- **NVIDIA's** `nccl-tests`, no source changes: `all_reduce_perf` passes its data checks on 4 H100s, one rank per GPU.
- **PyTorch** [6] training end-to-end on three parallelism strategies: **data-parallel** (ranks bit-identical after the gradient allreduce, **over** `verbs`), **sharded** (FSDP), and **tensor-parallel** via **Megatron-LM**'s parallel layers.
- **Collectives that reduce on the GPU**, over a **CUDA-IPC** transport added through the plugin ABI on NVIDIA targets.
- **GPUDirect, measured:** on H100/InfiniBand, one rank per GPU, no MPS.

Acknowledgments and AI Disclosure

Support is acknowledged from the ASCEND Center at Tennessee Technological University; the U.S. National Science Foundation (OAC-2514054, CSSI; CNS-2450093, CIRC); and the U.S. Department of Energy NNSA Predictive Science Academic Alliance Program (PSAAP-IV, Award DE-NA-0004267). This work uses NCSA Delta through the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (**ACCESS**) program, supported by U.S. NSF grants #2138259, #2138286, #2138307, #2137603, and #2138296; the specific ACCESS allocation is cited on the final poster.

AI disclosure. Claude (Opus 4.8, then Opus 5) wrote the code, tests, and diagrams; the authors set every requirement, judged every claim, and revised all drafted wording and content. Strong design inputs were used from the outset. Authors do not use AI for work we could not do ourselves by hand, and students were guided by the professors to ensure appropriate learning. We used AI for editorial and "google-like" purposes too.

OpenCCL™ Design Goals and Focus

Features:

- **Fast transfers, with the middle layer removed (now):** OpenCCL™ codes to RAPIDS, lightly abtracting the fabric's own API — `libcxi` on Slingshot-11, `ibverbs/mlx5` on InfiniBand— rather than through a portability layer such as libfabric. RAPIDS focuses on low cost of portability+GPU integration and high performance, but requires resource awareness.
- **State-of-the-art collective algorithms (in progress):** Importing the best ideas from NCCL, NCCLx, MSCCL, MPI Advance, etc.

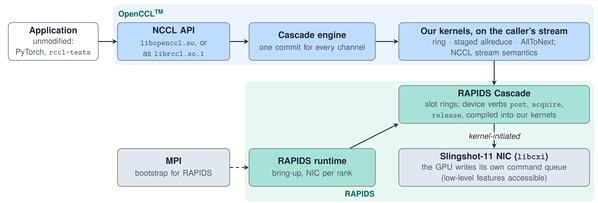
Targets and performance goals:

- **Slingshot: outperform NCCL and RCCL.**
- **InfiniBand: performance that competes where NCCL and RCCL are already tuned**, plus an extension surface they do not offer.
- **Newer fabrics: the way NCCL functionality is delivered** — a drop-in replacement on Cornelis and other networks.

AI-enabled middleware design for HPC

- OpenCCL™ is built by plan-heavy, **prompt-driven engineering** followed by programmer optimization/refactoring.
- Authors supply design, functional and non-functional requirements (a "mixture of human middleware designers"), and drive iteration/incrementation.
- **Design documents** are written and updated as we build, so each increment refines a recorded design driven by performance and functionality goals.
- Experience with actual low-level behavior is captured as tests run on diverse HPC systems.
- **Codesign across two codebases**: OpenCCL™ and RAPIDS.
- **Configurability by design, not by build accident.** A site picks its stack rather than accepting one. Example: PyTorch on a CPU-only system.
- **Taxonomical study and documentation** of **24+** *CCL variants worldwide, identifying unique features and commonality as input toward a standards-driven, performance-portable programming model on these and future diverse platforms.

RAPIDS and OpenCCL™ Codesign



- **Division of labor.** RAPIDS supplies bring-up, persistent flow-controlled channels, and operations; OpenCCL™ supplies the NCCL API (and more), the collective algorithms, and their kernels.
- **Codesign is bilateral.** Our suite and site runs are Cascade's key consumer. OpenCCL™ has motivated features and functions in RAPIDS, and vice versa as greater functionality and performance has evolved through the test-driven design and development process.

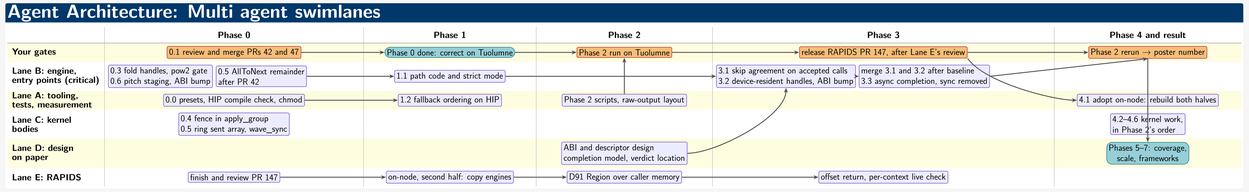
RAPIDS — Enabling Lower Middleware

The primitives, split on trigger domain, not on layers:

	Plenum	Cascade	Mobius	Herald
trigger	host	**kernel**	host	host
on-node	shared mem	**GPU copy**	shared mem	shared mem
off-node	mpi, cxi	**cxi only**	mpi, cxi, verbs, ...	mpi, cxi
built	always	**opt-in**	yes	always
API	experimental	**stable**	stable	stable

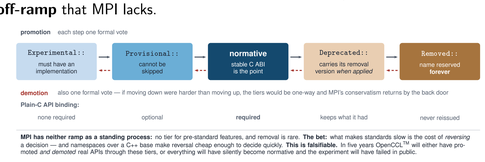
- **Cascade is the primary RAPIDS path on Slingshot-11**, working and **kernel initiated** on `cxi`.
- **RAPIDS codes to the fabric's lowest-level API and/or lower**, `libcxi` on Slingshot-11, not to a portability layer. *Persistent* transfers set up once, reused.
- **Fabric portability layers — diminishing value.** Abstractions that limit access to low-level features are bypassed by RAPIDS. Libfabric, in particular, offers little value since it hides network-specific features.
- **AI-generated refinements** to OpenCCL™ enables use of diverse RAPIDS on various networks that exploit the low-level features of each product.

Agent Architecture: Multi agent swimlanes

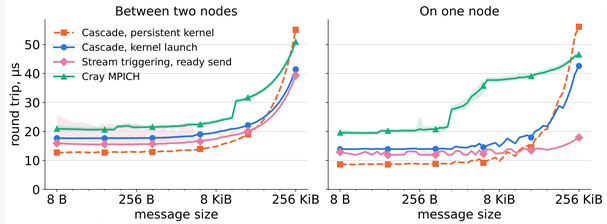


Middleware Standardization Lifecycle

Agile in concept, unlike the MPI Forum: as a standard, OpenCCL™ will have an **on-ramp** and an **off-ramp** that MPI lacks.



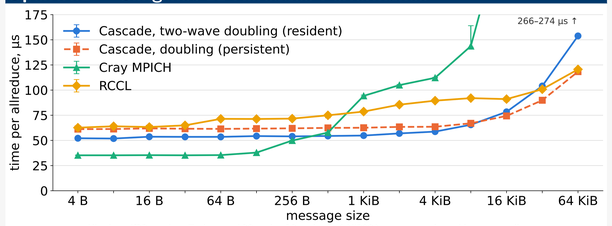
RAPIDS Baseline Performance vs. Cray MPICH on Slingshot-11



GPU-to-GPU ping-pong between two MI300A on Tuolumne, each side's add in the round trip; median of 1,000 iterations × 3 runs, band 10th–90th percentile. On one node both Cascade kernels copy by one wave.

- **Cascade offers superior performance** compared to Cray MPICH.
- **RAPIDS Cascade is the basis** for OpenCCL™ on Slingshot.
- The **Rally benchmark** has a trivial kernel between ping and pong.
- Notice the **poor tuning of Cray MPICH on-node** (above 1 KB).

OpenCCL™ Slingshot-11 Allreduce Performance



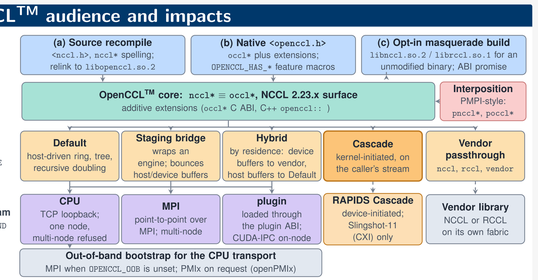
Allreduce (f32 sum) on Tuolumne: 128 ranks, 32 nodes × 4 MI300A, a compute kernel between calls. Slowest rank over 10,000 calls; median of 3 runs, whiskers min to max.

- **Except below 256 bytes, OpenCCL™ already outperforms** RCCL and MPI on Slingshot-11 (best configurations used for RCCL and MPI).
- These are only **preliminary results** for OpenCCL™.
- There is a **trivial kernel between each Allreduce** (keeps MPI honest).

Supporting MSCCL's AllToNext

- **MSCCLang's AllToNext** [8], implemented as an OpenCCL™ feature: `occlAllToNext`. Rank i sends to rank $i{+}1$; a shift, not a chain.
- **Their optimized on-node/off-node algorithm**, also implemented: scatter at the node boundary, cross nodes in parallel, gather (`occlAllToNextNodeAware`), beside naive and pipelined variants.
- **One schedule, four collectives**: AllToNext, AllToPrev, AllRingNext, AllRingPrev, through `occlShift`.
- **Runs on RAPIDS Cascade** as a device kernel, and on the portable host path.
- Performance measurements in progress.

OpenCCL™ audience and impacts



- **Target audience**
  - Scalable AI application developers
  - Companies offering products in the competitive heterogeneous accelerator ecosystem
  - People programming on various heterogeneous HPC+AI products
- **Masquerade** — build the NCCL/RCCL soname (`libnccl.so.2` / `librccl.so.1`): binary drop-in for a prebuilt client, and an *unmodified* one made instrumentable by Caliper or Vernier [12]. Read Llama's real collective pattern, then tune.

Control stability or choose newest

A vendor can change or break its interface at any time. For fast-moving AI work that can be acceptable; for long-lived and costly software it is a risk. OpenCCL™ offers a choice:

- Choose the **most-stable** or the **newest** level.
- **Experimental:** for pre-standard features — see the lifecycle figure.
- OpenCCL™ can be built to deliver application-specific profiles, with the goal of greater, focused optimization.
- A program needs its vendor library's whole *link-time* symbol set, not the part it calls: PyTorch references 45 NCCL entry points. OpenCCL™ supplies the rest as declined stubs — loads, and reports what it omits.

Related Work

- **Schedule languages and synthesizers**: MSCCLang [8], TACCL [13]. Algorithms, not a runtime; OpenCCL™ executes schedules of this kind.
- **GPU-initiated communication**: NVSHMEM [14], MSCCL++ [15], DeepEP [16]. NVIDIA-only; RAPIDS Cascade brings kernel initiation to Slingshot and AMD.
- **Stream-triggered MPI** [17]: CPU-free MPI GPU communication; the baseline in our RAPIDS charts.
- **Vendor** *CCLs: oneCCL [18] and others. NCCL-shaped, one vendor each.
- **Unifying layers**: FlagCX [7] routes between *CCLs and inherits their thin semantics.
- **HPC collective layers**: UCC/UCX [19]. No NCCL ABI; a framework needs a separate backend.
- **Locality-aware algorithms**: MPI Advance [11]. Node-aware, hierarchical collectives for MPI.
- **Inference data movement**: NIXL [9], Mooncake [10]. Point-to-point KV-cache transfer; complementary.
- **NCCL analyzed**: Hu et al. [20] document the protocols and algorithms NCCL leaves unspecified.

Selected references

[1] NVIDIA. NCCL. github.com/NVIDIA/nccl.
[2] AMD. RCCL (ROCm). github.com/ROCm/rccl.
[3] Meta. torchcomms: NCCLX/RCCLX/CTran. github.com/meta-pytorch/torchcomms.
[4] MPI Forum. MPI: A Message-Passing Interface Standard, v4.0. 2021. mpi-forum.org.
[5] Gropp, Lusk, Doss, Skjellum. A high-performance, portable implementation of the MPI standard. Parallel Computing, 1996.
[6] Li et al. PyTorch Distributed: Data-Parallel Training. VLDB, 2020.
[7] BAAI/FlagOpen. FlagCX. github.com/FlagOpen/FlagCX.
[8] Cowan et al. MSCCLang. ASPLOS, 2023.
[9] NVIDIA. NIXL: Inference Xfer Library. github.com/ai-dynamo/nixl.
[10] Qin et al. Mooncake: KVCache-centric LLM serving. USENIX FAST, 2025.
[11] Bienz, Schafer, Skjellum. MPI Advance: Open-Source Message Passing Optimizations. EuroMPI, 2023.
[12] Dominguez-Trujillo et al. Monitoring Comm. Patterns in Vernier. SC'25 ProTools, 2025.
[13] Shah et al. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. USENIX NSDI, 2023.
[14] NVIDIA. NVSHMEM. developer.nvidia.com/nvshmem.
[15] Microsoft. MSCCL++. github.com/microsoft/msccl.
[16] DeepSeek-AI. DeepEP. github.com/deepseek-ai/DeepEP.
[17] Bridges et al. Co-Design and Evaluation of a CPU-Free MPI GPU Communication Abstraction and Implementation. IEEE Cluster, 2026, to appear.
[18] Intel. oneCCL. github.com/uxlfoundation/oneCCL.
[19] UCF Consortium. UCC: Unified Collective Communication. github.com/openucx/ucc.
[20] Hu et al. Demystifying NCCL. arXiv:2507.04786, 2025.