Orrery: Modeling and Simulation of GPU-direct MPI Communication with SimGrid

Ahmad Moh'd Saleh A. Belbeisi
Technical University of Munich
Munich, Germany
ahmad.belbeisi@tum.de

Amir Raoofy
Leibniz Supercomputing Centre
Munich, Germany
amir.raoofy@lrz.de

Josef Weidendorfer
Dresden University of Technology
Dresden, Germany
josef.weidendorfer@tu-dresden.de

Majid Salimi Beni
Vienna University of Technology
Vienna, Austria
majid.salimibeni@tuwien.ac.at

***Abstract*—Simulation of HPC applications is an essential tool for analyzing system design trade-offs and enabling effective hardware–software co-design. As modern supercomputers increasingly rely on GPU acceleration, MPI implementations have evolved to support GPU-aware communication, enabling direct GPU data transfers. Modeling these mechanisms is therefore critical for simulating HPC applications on current systems. In this paper, we introduce Orrery, an interposition layer built on SimGrid's SMPI to model GPU-aware MPI communication using existing collective implementations and flow-based simulation. Orrery intercepts GPU memory allocations, kernel launches, data transfers, and MPI operations, and models their execution in SimGrid according to target system specifications, enabling simulation without requiring physical GPUs. We validate Orrery using CUDA-aware MPI microbenchmarks and a Jacobi solver on up to 32 nodes of the MUSICA supercomputer and demonstrate that it reproduces key GPU-aware MPI mechanisms.**

***Index Terms*—GPU-aware MPI, CUDA-aware MPI, Simulation, Performance Modeling, SimGrid, Collective Communication, GPU Communication.**

I. Introduction

GPUs are widely used in modern HPC systems, and, e.g., in the latest TOP500 ranking (June 2026) [1], four of the top five supercomputers are equipped with GPUs. Consequently, MPI implementations have evolved to support GPU-aware communication, enabling direct interaction between GPU memory and distributed communication mechanisms. As GPU-accelerated systems become increasingly prevalent, accurate performance modeling and simulation of these architectures are essential for analyzing, understanding, and optimizing HPC applications, as well as for guiding the design of future systems. Simulators leveraging novel techniques play a critical role in evaluating the performance and scalability of applications and algorithms:

Simulation enables evaluation of system configurations and application designs, allowing system architects and application developers to rapidly evaluate algorithm and hardware scalability and compare alternative approaches before costly deployments on large-scale systems.

Simulating modern GPU-accelerated HPC applications requires capturing and modeling kernel executions, data paths and data movements, as well as the support for GPU-aware communication mechanisms, often hidden behind the MPI interface. In particular, modeling GPU-aware MPI, empowered by peer-to-peer GPU communications [2] that is often used in modern GPU-accelerated systems, becomes increasingly important for simulating many scientific applications. SimGrid [3], [4] provides a strong foundation for this goal through its execution-driven model, detailed network abstraction, and SMPI support for simulating MPI applications; however, SMPI has been designed for CPU-based MPI execution and lacks native support for representing GPU memory management, GPU kernel execution, and GPU-aware MPI communication.

This work addresses this gap by extending SimGrid with GPU-aware communication capabilities, in particular targeting a CUDA-aware platform with GPUDirect RDMA [5] transports. We introduce *Orrery*[1], an interposition layer that enables the online simulation of CUDA-aware MPI applications. Without modifying SimGrid, *Orrery* intercepts and models GPU memory management operations like allocation and data transfer calls, kernel launches, and GPU-aware MPI communications, allowing for leveraging existing SimGrid and SMPI simulation engines for GPU-enabled HPC applications.

The main contributions of this work are as follows:

C1: We present Orrery, an interposition layer built on SMPI, and its first implementation, the first GPU-aware extension in SimGrid ecosystem. that enables online simulation of CUDA-aware MPI applications.

C2: We introduce two execution modes enabling applications to be simulated regardless of whether GPU devices are physically present on the simulation platform (Twin mode) or not (Skeleton mode).

C3: We provide an initial validation of Orrery on up to 32 nodes (128 GPUs) of the MUSICA supercomputer, using CUDA-enabled MPI micro-benchmarks and a GPU-enabled Jacobi solver application.

[1]Orrery and the calibration artifacts are publicly available under the MIT license at https://github.com/ahmadbelb/orrery.

II. Background and Related Work

SimGrid: SimGrid is an open-source framework for simulating distributed systems through analytical models of computation, communication, and I/O over user-defined virtual platforms. Its SMPI module enables execution-driven simulation of MPI applications by redirecting MPI operations to SimGrid's implementation, where each rank is represented as a lightweight simulation actor. Communication operations are simulated using SimGrid's flow-level network models and SMPI's implementations of the collective algorithms of MPI libraries, such as the ones implemented in Open MPI [6], while computations are represented by either native kernel execution and scaling their measured durations to the simulated platform, or by replacing them with virtual delays (advancing per rank virtual clocks). This approach enables lightweight and scalable simulation of MPI applications on a single machine without requiring access to the target system [6].

GPU-aware MPI: Unlike traditional MPI, GPU-aware MPI accepts device pointers as communication buffers, allowing data to remain on the GPU unless staging through host memory is enabled [2].

GPU-aware MPI implementations typically determine the memory type of communication buffers and select an appropriate data path, including peer-to-peer transfers over PCIe or NVLink within a node, GPUDirect RDMA across nodes [5], or staged transfers through pinned host memory when direct GPU communication is unavailable.

Related Work: Prior work has explored simulation approaches for large-scale HPC systems, primarily focusing on CPU-based communication and network behavior. Trace-driven simulators capture, parse, and replay application executions under different system configurations; for example, LogGOPSim [7] and PHANTOM [8] provide MPI-oriented communication models, while ATLAHS [9] enables evaluation of distributed workloads through a common representation and multiple simulation backends. However, trace-based approaches capture only a specific execution instance, requiring new traces for different configurations and limiting scalability studies [9], [10], while trace collection and storage can become costly at large scale [9], [11]. Execution-driven simulators, such as SimGrid/SMPI [6] and xSim [12], address these limitations by executing applications against a modeled platform, while SST/macro [13] and BigSim [14] combine both approaches. With the rise of GPU-accelerated systems, simulators such as ASTRA-sim [15]–[17] and SimAI [18] have extended simulation toward distributed AI workloads, through specialized GPU communication tools such as NCCL [19]. None of these approaches support GPU-aware MPI: execution-driven simulators lack device-memory awareness, while AI-oriented simulators primarily target NCCL-based communications rather than MPI.

III. Simulating GPU-aware MPI with Orrery

Figure 1 presents the overall architectural design of Orrery. Orrery acts as an interposition layer that interfaces with SMPI to support GPU-aware MPI applications without modifying the underlying SimGrid framework. An application leveraging GPU-aware MPI ① runs unmodified, while Orrery ② intercepts relevant GPU (CUDA) runtime and MPI calls. GPU memory allocations, data transfers, kernel executions, and communication operations are captured and modeled as a target machine description representing the bandwidth and latencies of operations and the data paths ③. This design allows using the existing SimGrid simulation cores: The resulting execution costs are charged to each rank's virtual clock within SimGrid ④, which continues to provide the underlying process scheduling and network simulation.

A. Orrery Interposition

SMPI's standard build process targets host-based MPI applications and cannot compile or link device code. Orrery extends the build process by using the vendor toolchain (`nvcc` in our prototype) for device code compilation and final linking while retaining SMPI's MPI interface. The interposition layer and cost model (Sec. III-B) are compiled with the application into a shared object loaded by `smpirun`, making interception a link-time property rather than a runtime preload mechanism. To intercept CUDA runtime operations, Orrery provides CUDA wrappers bound at the application binary interface (ABI): link-time symbol binding redirects calls such as `cudaMalloc` to Orrery without parsing or transforming the source. MPI interception leverages the standard MPI Profiling Interface (PMPI) mechanism [20]: Orrery provides weak MPI wrappers, while `PMPI_*` symbols are resolved by SMPI. Therefore, both call paths follow application → Orrery wrappers → SMPI/SimGrid, enabling CUDA operations and MPI communication to be captured and modeled in a unified simulation environment.

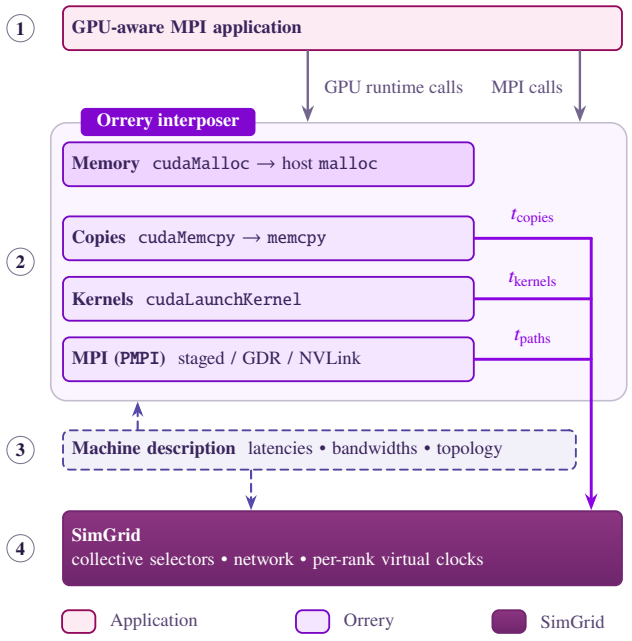


Fig. 1: Orrery overview, depicting how it intercepts GPU runtime and MPI calls to leverage SimGrid's simulation engine.

B. Modeling GPU-direct P2P, Collectives, and Kernels

Each intercepted operation is assigned a duration derived from the machine description and added to the rank's virtual clock. Data movement follows the standard latency–bandwidth model established by Hockney and LogGP [21], [22]. The *charging structure* defines which costs a GPU-aware call incurs, which rank is charged, and whether each cost is applied per call or per message. A call's cost is composed of recurring terms. The central term is the *device leg*: the first or last segment between GPU memory and the network interface. Charged once per device buffer on the owning rank, its cost follows the selected transport:

$$t_{\mathrm{leg}}(n) = \begin{cases} t_{\mathrm{d2h}}(n) + t_{\mathrm{h2d}}(n) & \text{Host staging} \\ \alpha_{\mathrm{gdr}} & \text{GPUDirect RDMA} \\ t_{\mathrm{peer}}(n) & \text{NVLink / PCIe peer} \\ 0 & \text{Host-resident buffer,} \end{cases} \tag{1}$$

where t_{d2h}, t_{h2d}, and t_{peer} follow piecewise-linear latency–bandwidth models, while α_{gdr} represents the per-message overhead of direct NIC access to device memory.

Calls combine device legs with network time. A point-to-point transfer pays one leg per endpoint. Collectives (e.g., broadcast, allgather) pay one leg per device buffer involved—one for broadcast and two for allgather—plus a fixed per-call overhead. Operator-based collectives (allreduce, reduce) are also covered [20]; in the case of Open MPI's CUDA component [23], operands are staged to host memory, reduced on the host, and copied back, replacing device legs with host bounce and reduction costs:

$$\begin{aligned} T_{\mathrm{p2p}} &= t_{\mathrm{leg}}(n_{\mathrm{s}}) + T_{\mathrm{net}} + t_{\mathrm{leg}}(n_{\mathrm{r}}), \\ T_{\mathrm{coll}} &= t_{\mathrm{leg}}(n_{\mathrm{s}}) + t_{\mathrm{leg}}(n_{\mathrm{r}}) + t_{\mathrm{call}} + T_{\mathrm{net}}, \\ T^{\mathrm{op}}_{\mathrm{coll}} &= \underbrace{2\,t_{\mathrm{d2h}} + t_{\mathrm{h2d}}}_{\text{host bounce}} + t_{\mathrm{reduce}} + t_{\mathrm{call}} + T_{\mathrm{net}}. \end{aligned} \tag{2}$$

Here n_{s} and n_{r} denote send and receive payload sizes. t_{call} represents the per-call overhead, while $t_{\mathrm{reduce}} = (P-1)\,n/(P\,\beta_{\mathrm{op}})$ models the reduce-scatter arithmetic floor, where P is the communicator size and β_{op} is the host reduction rate [24]. T_{net} is provided by SimGrid: Orrery accounts for GPU-side costs and invokes `PMPI_*`, while SimGrid models the communication algorithm on the declared topology.

Where needed, a kernel launch is modeled as a fixed overhead plus device work:

$$T_{\mathrm{kernel}} = t_{\mathrm{launch}} + \frac{Nf}{R}, \quad T_{\mathrm{compute}} = \sum_{\ell} T^{(\ell)}_{\mathrm{kernel}} + \sum_{s} t^{(s)}_{\mathrm{sync}}, \tag{3}$$

where $N = G \times B$ is the total thread count, f the per-thread work, and R the achieved device rate. t_{launch} and t_{sync} are measured launch and synchronization overheads, with ℓ and s indexing kernel launches and synchronization calls.

C. Machine Description and Model Calibration

The machine description provides the parameters of the cost model (Eqs. 1–3). Since achieved performance often differs from theoretical peaks, inflating the error [25], each parameter is calibrated using a dedicated micro-benchmark on the reference cluster. Table I summarizes the probes used in these micro-benchmarks and their target parameters.

TABLE I: Calibration probes and collected model parameters.

Probe	Yields	Symbol
GPU-GPU intra-node	link latency, bandwidth	t_{peer}
GPU-GPU inter-node	wire latency, bandwidth	T_{net}
MPI API overhead	per-call library cost	α_{gdr}
Host–device copy	copy latency, banded bandwidth	t_{d2h}, t_{h2d}
Empty kernel launch	launch, sync overheads	t_{launch}, t_{sync}
Kernel compute	per-kernel compute rate	R
`MPI_Op` time	host operation (e.g., reduction) time	β_{op}

D. Execution Modes

Orrery provides two execution modes, following SMPI's treatment of computation phases. In both modes, intercepted operations are evaluated by the cost model and their durations are added to the rank's virtual clock. SimGrid handles scheduling and contention: Orrery determines the operation duration, while SimGrid determines its execution.

The modes differ only in how wrappers handle operations before charging:

Skeleton mode does not require GPUs: memory allocations use an internal registry, and kernels and data transfers are modeled with cost models. However, the device buffers contain no output, and no kernel is launched. This mode uses all Equations 1-3 for modeling.

Twin mode requires a physical GPU for GPU kernel execution (does not use Equation 3), and forwards each call to the CUDA runtime and charges the measured duration. This is done by leveraging `-Bsymbolic` to resolve the name to the wrapper; the real entry point comes from `dlsym(RTLD_NEXT)`.

IV. Experimental Results

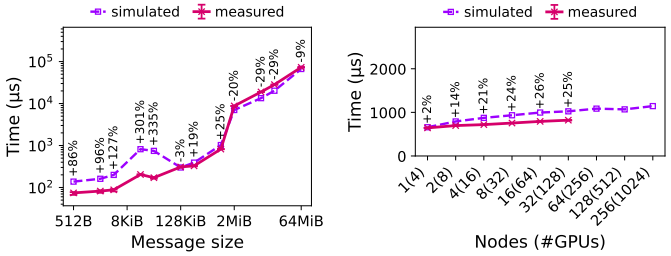
A. Experimental Setup

Reference measurements are collected on 32 nodes of the MUSICA supercomputer, each equipped with four fully connected NVIDIA H100 GPUs via NVLink 4.0 and connected by 200 Gbit/s InfiniBand. The system runs Open MPI 4.1.6 with UCX 1.15, CUDA support, and GPUDirect RDMA.

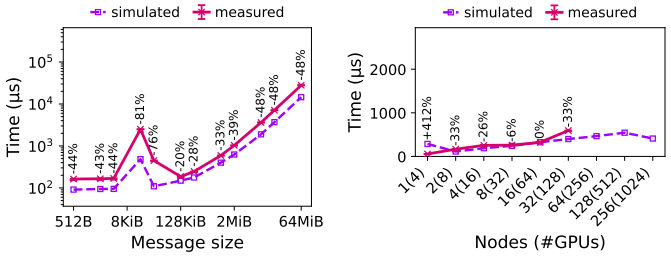
Simulations use skeleton mode on a workstation with a 4-core Intel Xeon W-2125 (4.0 GHz), 32 GB memory, and a 5 GB Quadro P2000 GPU, running SimGrid 4.1.1.

B. Micro-benchmark Evaluations

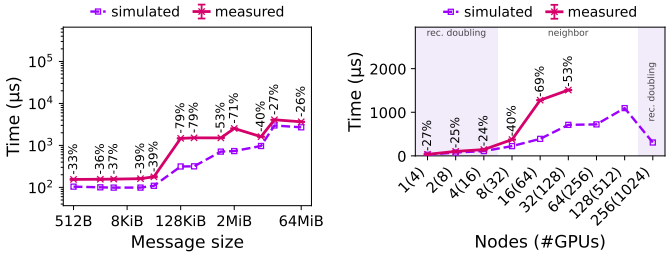
Figure 2 compares skeleton-mode simulation with micro-benchmark measurements. The left column varies message size at 128 GPUs: Orrery qualitatively reproduces GPU-direct MPI communication behavior on different collective operations and message sizes, with an average error of 60%. The right column scales a fixed 1 MiB message size up to 1024 GPUs for the simulator: The simulator can qualitatively



(a) Allreduce. (b) Allreduce scaling.

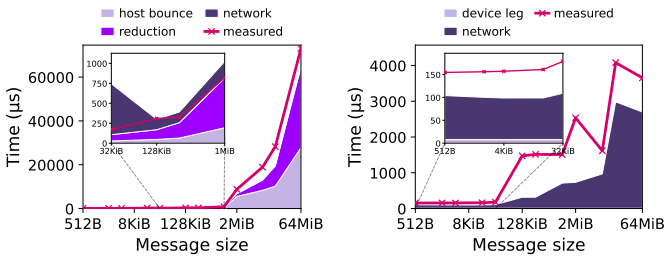


(c) Bcast. (d) Bcast scaling.



(e) Allgather. (f) Allgather scaling.

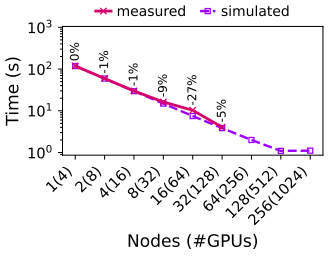
Fig. 2: Micro-benchmarks, measured versus simulated: The left column (a, c, and e) shows results on 32 nodes (128 GPUs); the right column (b, d, and f) shows strong scaling with a fixed message size of 2^{20} B, from 1 node to 32 nodes. Relative simulation error is annotated next to each point, and the shaded regions in (f) highlight the collective algorithm selected by Open MPI at each scale.



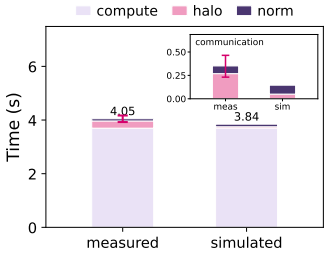
(a) Allreduce.

(b) Allgather.

Fig. 3: Simulated collective micro-benchmarks at 32 nodes (128 GPUs), decomposed into the costs the interposer charges.



(a) Strong scaling.



(b) Decomposed runtimes.

Fig. 4: Jacobi with GPUDirect RDMA. (a) Measured versus simulated, measured to 32 nodes (128 GPUs), simulated beyond. (b) The 128-GPU run split into the solver's phases; the measured bar's range is the spread across ranks. Inset: the two communication phases, magnified.

reproduce scalability, and we can also see clearly the effect of Open MPI's algorithm selection for Allgather (Fig. 2f) where performance improves at 1024 GPUs with choosing the recursive doubling algorithm.

Figure 3 provides a more detailed view with simulated time broken down into cost components.

For Allreduce (3a), host bounce and reduction time grow with the message size because the reduction operations are performed on the host, and data must be copied back and forth between host and device. Allgather (3b) and Bcast (not shown here) have no reduction and no such bottleneck.

C. Jacobi Solver

Figure 4 shows a Jacobi solver [26], simulated in skeleton mode with every kernel launch modeled using Eq. 3. On different numbers of nodes (Fig. 4a), the relative error of the simulation is at most 27%, and we can see how increasing the number of nodes in the simulator to 256 improves the performance. To further analyze the application, Fig. 4b decomposes the 128-GPU run into solver phases. The simulated and measured total run times differ by about 5%, as compute dominates and its simulation error is only 0.1%. The inset highlights the communication phases: 0.35 s measured versus 0.14 s simulated, with halo exchange undercharged (-81%) and norm calculation overcharged ($+16\%$).

V. Discussion

The errors in the simulation arise in part from calibration mismatches: the calibration relies on isolated two-GPU device-buffer measurements, whereas real applications involve different buffer types and communication patterns. In addition, collective operations introduce concurrent transfers that are not captured by the current calibration benchmarks and probes (listed in Table I). Part of the error is also structural: the cost model is built from simple latency–bandwidth terms in the style of Hockney [21] and LogGP [22], which abstract away effects such as protocol switches, memory registration, and contention among concurrent transfers, so a residual error may remain even with well-tuned parameters. Improving accuracy requires independent measurements for host-buffer

transfers and concurrent communication scenarios, as well as accounting for network and memory contention.

Two limitations remain in our current prototype: First, skeleton mode relies on a single measured kernel rate, which is sufficient for memory-bound workloads but cannot distinguish computation and memory effects or generalize across kernels and architectures. Second, Orrery currently models operations sequentially, so it cannot capture computation-communication overlap in asynchronous applications.

VI. Conclusion and Future Directions

We introduced Orrery, an interposition layer that extends SimGrid's SMPI to GPU-aware MPI applications, modifying neither the simulator nor the application code. It captures GPU runtime calls (e.g., CUDA calls) and charges each from a calibrated cost model, exposing the device-side cost of a GPU-aware MPI implementation to the simulation framework.

Several extensions can broaden Orrery's applicability. Starting with validation on full applications. The interception approach can be extended beyond CUDA to other runtimes such as HIP and SYCL, which would allow validating the cost model on GPUs from other vendors, while additional and automated calibration [25] can improve model accuracy. Twin mode enables validation across real platforms, and the machine description allows exploration of future GPU systems, topologies, and extending previous MPI algorithm selection studies to be GPU-aware [27]. Such modeling can also evaluate emerging approaches, such as in-network computing, as in [28].

As the GPU communication stack moves further off the host, we envision Orrery setting the foundations for modeling GPU-aware communication based on GPUDirect RDMA in the well-established community-driven SimGrid simulator to better evaluate machines, libraries, and algorithms, including those that do not yet exist.

Acknowledgements

The computational results presented in this work have been achieved using the Austrian Scientific Computing (ASC) infrastructure (cluster: MUSICA) under project ID 201169. Development and initial validation were carried out on systems in the BEAST (Bavarian Energy Architecture & Software Testbed) test environment at the Leibniz Supercomputing Centre.

References

[1] J. Dongarra, "TOP500 list," 2026, [Accessed 19-08-2026]. [Online]. Available: https://www.top500.org/lists/top500/

[2] H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda, "Gpu-aware MPI on RDMA-enabled clusters: Design, implementation and evaluation," *IEEE Trans. Parallel Distributed Syst.*, vol. 25, no. 10, pp. 2595–2605, 2014, doi: 10.1109/TPDS.2013.222.

[3] H. Casanova, A. Legrand, and M. Quinson, "Simgrid: A generic framework for large-scale distributed experiments," in *Tenth International Conference on Computer Modeling and Simulation (UKSim 2008)*, 2008, pp. 126–131, doi: 10.1109/UKSIM.2008.28.

[4] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Lowering entry barriers to developing custom simulators of distributed applications and platforms with SimGrid," *Parallel Comput.*, vol. 123, p. 103125, 2025, doi: 10.1016/J.PARCO.2025.103125.

[5] NVIDIA Corporation, "GPUDirect RDMA Documentation," https://docs.nvidia.com/cuda/gpudirect-rdma/, 2026, accessed: Apr. 22, 2026.

[6] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI applications: The SMPI approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 8, pp. 2387–2400, Aug. 2017, doi: 10.1109/TPDS.2017.2669305.

[7] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim: simulating large-scale applications in the LogGOPS model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC 2010, Chicago, Illinois, USA, June 21-25, 2010*, S. Hariri and K. Keahey, Eds. ACM, 2010, pp. 597–604, doi: 10.1145/1851476.1851564.

[8] J. Zhai, W. Chen, and W. Zheng, "PHANTOM: predicting performance of parallel applications on large-scale parallel machines using a single node," in *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2010, Bangalore, India, January 9-14, 2010*, R. Govindarajan, D. A. Padua, and M. W. Hall, Eds. ACM, 2010, pp. 305–314, doi: 10.1145/1693453.1693493.

[9] S. Shen, T. Bonato, Z. Hu, P. Jordan, T. Chen, and T. Hoefler, "ATLAHS: an application-centric network simulator toolchain for AI, HPC, and distributed storage," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2025, St. Louis, MO, USA, November 16-21, 2025*. ACM, 2025, pp. 349–367, doi: 10.1145/3712285.3759838.

[10] T. Cornebize and A. Legrand, "Simulation-based optimization and sensibility analysis of MPI applications: Variability matters," *J. Parallel Distributed Comput.*, vol. 166, pp. 111–125, 2022, doi: 10.1016/J.JPDC.2022.04.002.

[11] C. Wang, P. Balaji, and M. Snir, "Pilgrim: scalable and (near) lossless MPI tracing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021, doi: 10.1145/3458817.3476151.

[12] S. Böhm and C. Engelmann, "xSim: The extreme-scale simulator," in *2011 International Conference on High Performance Computing & Simulation, HPCS 2011, Istanbul, Turkey, July 4-8, 2011*, W. W. Smari and J. P. McIntire, Eds. IEEE, 2011, pp. 280–286, doi: 10.1109/HPCSIM.2011.5999835.

[13] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. R. Mayo, "A simulator for large-scale parallel computer architectures," *Int. J. Distributed Syst. Technol.*, vol. 1, no. 2, pp. 57–73, 2010, doi: 10.4018/JDST.2010040104.

[14] G. Zheng, G. Kakulapati, and L. V. Kalé, "BigSim: A parallel simulator for performance prediction of extremely large parallel machines," in *18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA*. IEEE Computer Society, 2004, doi: 10.1109/IPDPS.2004.1303013.

[15] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-SIM: enabling SW/HW co-design exploration for distributed DL training platforms," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2020, Boston, MA, USA, August 23-25, 2020*. IEEE, 2020, pp. 81–92, doi: 10.1109/ISPASS48437.2020.00018.

[16] W. Won, T. Heo, S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "ASTRA-sim2.0: modeling hierarchical networks and disaggregated systems for large-model training at scale," in *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2023, Raleigh, NC, USA, April 23-25, 2023*. IEEE, 2023, pp. 283–294, doi: 10.1109/ISPASS57527.2023.00035.

[17] W. Won, J. Yoo, T. Ta, M. Dey, A. Balogh, P. Datta, F. Eris, C. Green, W. Liu, C. Man, K. Mandal, A. Rai, V. Ramakrishnaiah, R. Shah, D. Sidler, H. Sikhwal, H. Wu, T. Krishna, and B. M. Beckmann, "Astra-sim 3.0: Next-level distributed machine learning simulations via high-fidelity GPU and infrastructure modeling," *CoRR*, vol. abs/2606.10440, 2026, doi: 10.48550/ARXIV.2606.10440.

[18] X. Wang, Q. Li, Y. Xu, G. Lu, D. Li, L. Chen, H. Zhou, L. Zheng, S. Zhang, Y. Zhu, Y. Liu, P. Zhang, K. Qian, K. He, J. Gao, E. Zhai, D. Cai, and B. Fu, "SimAI: Unifying architecture design and performance tuning for large-scale large language model training with scalability and precision," in *22nd USENIX Symposium on Networked Systems Design and Implementation, NSDI 2025, Philadelphia, PA, USA, April 28-30, 2025*, T. A. Benson and R. N. Mysore, Eds. USENIX Association, 2025, pp. 541–558. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai

[19] Z. Hu, S. Shen, T. Bonato, S. Jeaugey, C. Alexander, E. Spada, J. Dinan, J. R. Hammond, and T. Hoefler, "Demystifying NCCL: an in-depth analysis of GPU communication protocols and algorithms," in *IEEE Symposium on High-Performance Interconnects, HOTI 2025, San Jose, CA, USA, August 20-22, 2025*. IEEE, 2025, pp. 48–59, doi: 10.1109/HOTI66940.2025.00024.

[20] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard, Version 5.0," 2025. [Online]. Available: https://www.mpi-forum.org/docs/mpi-5.0/mpi50-report.pdf

[21] R. W. Hockney, "The communication challenge for MPP: Intel Paragon and Meiko CS-2," *Parallel Comput.*, vol. 20, no. 3, pp. 389–398, 1994, doi: 10.1016/S0167-8191(06)80021-9.

[22] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGP in theory and practice - an in-depth analysis of modern interconnection networks and benchmarking methods for collective operations," *Simul. Model. Pract. Theory*, vol. 17, no. 9, pp. 1511–1521, 2009, doi: 10.1016/J.SIMPAT.2009.06.007.

[23] The Open MPI Project, "Open MPI v4.1.6: the coll/cuda component," Source code, https://github.com/open-mpi/ompi/blob/v4.1.6/ompi/mca/coll/cuda/coll_cuda_module.c, 2023, accessed: Aug. 24, 2026.

[24] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, 2005, doi: 10.1177/1094342005051521.

[25] J. McDonald, Y. C. Wong, K. Mehta, F. Suter, R. F. da Silva, L. Pottier, E. Deelman, and H. Casanova, "Determining levels of detail for simulators of parallel and distributed computing systems via automated calibration," in *Proceedings of the SC '25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC Workshops 2025, St Louis, MO, USA, November 16-21, 2025*. ACM, 2025, pp. 1452–1463, doi: 10.1145/3731599.3767698.

[26] A. Herten, L. Oden, S. Garcia de Gonzalo, J. Kraus, M. Hrywniak, D. Appelhans, and L. Morgenstern, "Efficient distributed GPU programming for exascale," Jun. 2026, version 10.0-isc26, file `03-H_Multi_GPU_Parallelization/solutions/jacobi.cu`, doi: 10.5281/zenodo.5745504.

[27] M. S. Beni, B. Cosenza, and S. Hunold, "MPI collective algorithm selection in the presence of process arrival patterns," in *IEEE International Conference on Cluster Computing, CLUSTER 2024, Kobe, Japan, September 24-27, 2024*. IEEE, 2024, pp. 108–119, doi: 10.1109/CLUSTER59578.2024.00017.

[28] A. M. S. A. Belbeisi, M. S. Beni, T. Erbesdobler, E. Saleh, M. Tovey, A. Raoofy, and J. Weidendorfer, "Simulating MPI collectives on Tofino smart switches in SimGrid," in *Proceedings of the 23rd ACM International Conference on Computing Frontiers, CF 2026, Catania, Italy, May 19-21, 2026*. ACM, 2026, pp. 118–121, doi: 10.1145/3801487.3801809.