Productive Scalable Distributed Task Scheduling Using an MPI-based Backend for Dagger

Yan Guimarães*
University of Brasilia
Brasilia, Federal District, Brazil
yan.guimaraes@aluno.unb.br

Felipe Tomé (advisor)
Massachusetts Institute of Technology
Cambridge, MA, USA
felipe0@mit.edu

Julian Samaroo (advisor)
Massachusetts Institute of Technology
Cambridge, MA, USA
jpsamaroo@mit.edu

Abstract

Distributed computing frameworks are vital for managing complex workloads in high-performance computing and scientific research. Julia's Dagger.jl supports task-based parallelism using TCP communication, suitable for cloud and local environments. However, TCP limits performance on modern HPC systems with low-latency, high-bandwidth interconnects. We introduce MPIAcceleration, an MPI-based backend replacing TCP with MPI-aware task placement and data movement. We benchmarked MPIAcceleration against TCP using parallel Cholesky decomposition on an AWS r5d.24xlarge cloud instance and Aurora exascale supercomputer at Argonne National Laboratory. Results show MPI successfully enables Dagger on HPC interconnects, narrowing the performance gap with TCP on Aurora compared to cloud environments.

CCS Concepts

• **Software and its engineering** → *Software libraries and repositories*; *Software development techniques*; • **Computing methodologies** → **Parallel algorithms**; **Distributed computing methodologies**.

Keywords

HPC, Julia, Dagger, MPI, TCP

ACM Reference Format:
Yan Guimarães, Felipe Tomé (advisor), and Julian Samaroo (advisor). 2025. Productive Scalable Distributed Task Scheduling Using an MPI-based Backend for Dagger. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis. (SC '25).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Extreme scale simulations in the exaflop era are heavily based on distributed frameworks that balance scalability, productivity, and the ability to exploit cutting-edge hardware; some of the known examples in this landscape are Legion [3] and Iris [4]. The modern parallel programming language Julia addresses this need with `Dagger.jl` [2], a parallel computing framework that uses Directed Acyclic Graphs (DAGs) to efficiently schedule and execute distributed tasks. Dagger builds on Julia's native `Distributed.jl` module, which establishes communication between processes, called workers, through standard TCP, is a design choice that, while convenient and portable, becomes a limiting factor in high-performance computing (HPC) clusters equipped with specialized low-latency interconnects such as Infiniband [6] or HPE Slingshot [5] . To extend Dagger's productivity benefits, we introduce MPIAcceleration, a strategic extension to Dagger with an MPI-based backend. MPI can extend the scope of Dagger workflows to scale properly on HPC clusters, leveraging the network optimizations already integrated by the various MPI vendors.

2 Methodology

2.1 MPI scheduler implementation

Dagger's MPIAcceleration integrates seamlessly with the existing scheduler, enabling task graphs across MPI ranks with minimal changes through a single line: **`Dagger.accelerate!(:mpi)`**. When enabled, each MPI rank is mapped into Dagger's Processor/Memory Space model with rank-aware placement, ensuring tasks execute where their data resides and minimizing communication overhead. Remote data transfers occur transparently, returning handles on appropriate ranks.

The communication layer utilizes nonblocking MPI with cooperative progress. Asynchronous sends/receives are polled while yielding, overlapping computation with communication. For large numeric payloads (dense arrays and sparse formats), in-place transfers avoid unnecessary copies. Deterministic message tags keep channels disambiguated, and lightweight safeguards ensure that multiple receivers do not race on the same message.

2.2 Experimental setup

We evaluated the performance of Dagger's MPIAcceleration against the Dagger's TCP backend using parallel Cholesky decomposition as a benchmark. Experiments were conducted on a high-memory cloud instance (AWS r5d.24xlarge with 96 CPUs and 768 GB RAM) and the Aurora exascale supercomputer at Argonne National Laboratory [1] . Aurora consists of over 10,600 nodes interconnected by an HPE Slingshot high-bandwidth fabric [5]. For scalability testing, we used 81 nodes with one MPI rank per node to deliberately enforce inter-node communication, leveraging MPI's specialization in efficient distributed messaging. This setup allows evaluation of Dagger's MPI and TCP backends under realistic multi-node, low-latency HPC conditions and contrasts with the local, single-node network environment where communication overheads differ markedly.

*Main author contributed to this research.

SC '25, St. louis, MO

ACM ISBN 978-1-4503-XXXX-X/2025/11
https://doi.org/XXXXXXX.XXXXXXX

3 Evaluation

Two types of scaling experiments were conducted to thoroughly evaluate the system's performance:

- **Strong scaling** assessed the performance of a fixed 18000 x 18000 matrix workload across multiple processors to test the framework's capacity to distribute a constant workload among the cores.
- **Weak scaling** evaluated performance with a workload that increased in size proportionally to the number of processors, ranging from 1 to 81, to understand how performance scaled as both the workload and computing capacity expanded together.

3.1 Execution Time vs Number of Processes

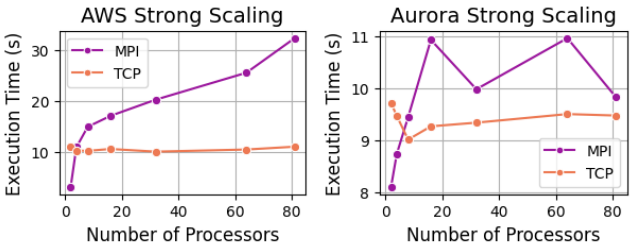

Figure 1: Strong scaling using MPI and TCP backends on AWS and Aurora with Float32 precision.

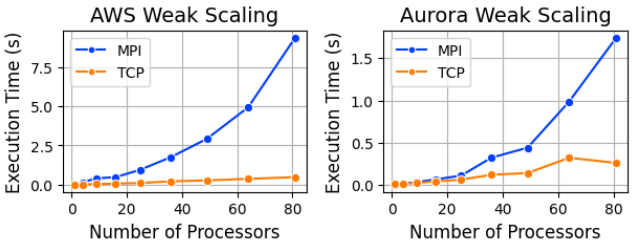

Figure 2: Weak scaling using MPI and TCP backends on AWS and Aurora with Float32 precision.

For strong scaling **Figure 1**, MPI execution time on AWS increases linearly with processor count, while TCP exhibits relatively flat performance within single-node limits. On Aurora, MPI exhibits superior performance at lower processor counts and maintains execution times significantly closer to TCP at scale, demonstrating the benefit of Slingshot's low-latency interconnect. For weak scaling **Figure 2**, MPI time grows proportionally with processes due to inter-node communication costs, while TCP remains nearly constant.

3.2 GFLOPS vs Number of Processes

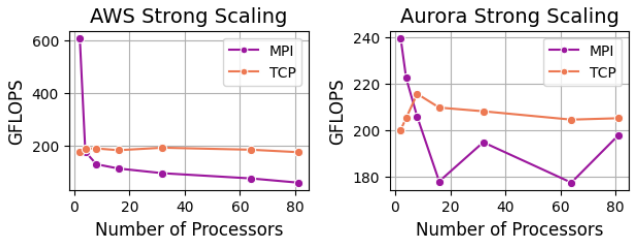

Figure 3: Strong scaling using MPI and TCP backends on AWS and Aurora with Float32 precision.

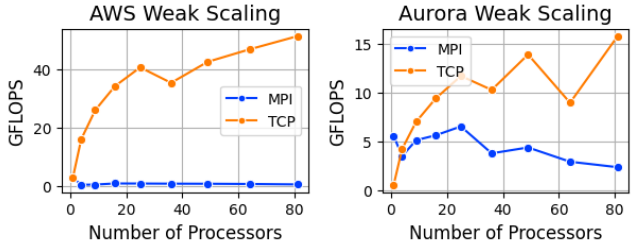

Figure 4: Weak scaling using MPI and TCP backends on AWS and Aurora with Float32 precision.

GFLOPS measurements in **Figure 3** and **Figures 4** highlight throughput differences. On AWS, TCP consistently outperforms MPI as the processor count increases. In contrast, on Aurora, the performance gap narrows, especially at moderate scales. Although TCP achieves higher overall throughput, MPI remains competitive in multi-node HPC environments, aligning with the integration's primary goal.

4 Conclusion

The MPI implementation demonstrates Dagger's ability to execute workloads across HPC interconnects, achieving competitive performance with TCP on Aurora. Nonetheless, the communication layer requires further optimization for larger node counts. Future work opportunities include collective-based distribution (e.g., Bcast/Scatterv) using Remote Memory Access (RMA) to reduce communication overhead. Additionally, hierarchical scheduling and a broader use of in-place data transfers will be essential to realize MPI's potential and strengthen Dagger's position as a productive yet high-performance framework for modern scientific computing.

Acknowledgments

This research used resources of the Argonne Leadership Computing Facility, a U.S. Department of Energy (DOE) Office of Science user facility at Argonne National Laboratory, and is based on research supported by the U.S. DOE Office of Science-Advanced Scientific Computing Research Program, under Contract No. DE-AC02-06CH11357. It was also supported by the Google Summer of Code

(GSoC) 2025. Combined with the invaluable guidance from my mentors, Felipe Tomé and Julian Samaroo, this effort was instrumental in the successful completion of the project. I also sincerely thank Larissa Guimarães and Gabriel Manso for their helpful conversations and suggestions, which significantly improved the structure and clarity of this work.

References

[1] Benjamin S. Allen et al. 2025. Aurora: architecting argonne's first exascale supercomputer for accelerated scientific discovery. *arXiv preprint arXiv:2509.08207*, (Sept. 2025). https://arxiv.org/abs/2509.08207.

[2] Rabab Alomairy, Felipe Tome, Julian Samaroo, and Alan Edelman. 2024. Dynamic task scheduling with data dependency awareness using julia. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE.

[3] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2012. Legion: expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (SC '12). IEEE Computer Society Press, Salt Lake City, Utah, USA, (Nov. 2012), 1–11. ISBN: 978-1-4673-0806-9. doi:10.1109/SC.2012.71.

[4] Jungwon Kim, Seyong Lee, Beau Johnston, and Jeffrey S. Vetter. 2021. Iris: a portable runtime system exploiting multiple heterogeneous programming systems. In *Proceedings of the 25th IEEE High Performance Extreme Computing Conference* (HPEC '21). IEEE.

[5] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. 2020. An in-depth analysis of the slingshot interconnect. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*. Vol. 2020-November. IEEE Computer Society, (Nov. 2020). ISBN: 9781728199986. doi:10.1109/SC41405.2020.00039.

[6] Jiuxing Liu Balasubramanian Chandrasekaran Jiesheng Wu Weihang Jiang Sushmitha Kini Weikuan Yu Darius Buntinas Peter Wyckoff and D K Panda. 2003. Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics ¡. Tech. rep.