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

Yan Guimarães [1] Felipe Tomé [2†] Julian Samaroo [2†]

[1]University of Brasilia [2]Massachusetts Institute of Technology [†]Advisor

Motivation

- High-performance computing (HPC) and scientific research increasingly depend on distributed computing frameworks that can handle complex, large-scale workloads and are both scalable and productive for developers.
- The Julia language's Dagger.jl framework leverages Directed Acyclic Graphs (DAGs) to schedule and execute distributed tasks, relying on standard TCP communication. While this model is well-suited for cloud and local clusters, it becomes a limiting factor in HPC clusters that are optimized for protocols like Message Passing Interface (MPI).

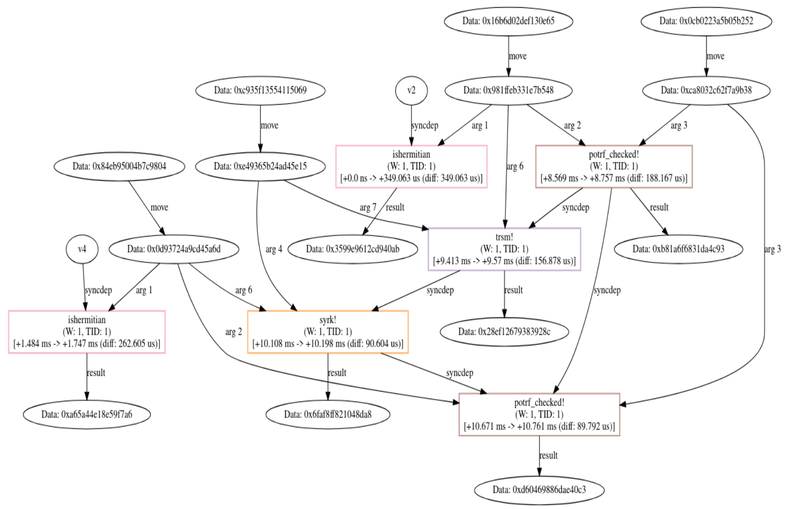

Figure 1. Dynamic Acyclic Graph (DAG) representing the data dependencies of a small (4x4 matrix with 2x2 tiles) Cholesky decomposition.

Key Question

- How can Dagger's communication layer be extended to achieve scalability on high-performance clusters without compromising ease of use?

Solution Approach

We introduce an MPI-based backend for Dagger, which enhances the communication layer and integrates with the scheduler, enabling task graphs to execute across multiple MPI ranks with just a single line of code.

```
julia> using Dagger

julia> Dagger.accelerate!(:mpi)
Dagger.MPIAcceleration(MPI.Comm(Ptr{Nothing} @0x00000001592806c8))

julia> rand(Blocks(2, 2), 4, 4)
4x4 DMatrix{Float64} with 2x2 partitions of size 2x2:
 0.54052   0.625893  0.78572   0.861409
 0.600908  0.791648  0.26342   0.833294
 0.370986  0.208938  0.14419   0.0591259
 0.349641  0.882112  0.176859  0.193161
```

Figure 2. Julia REPL output of a Dagger.jl computation with MPI acceleration, demonstrating block-partitioned random matrix construction.

Results Summary

- On AWS, MPI execution time increases with processor count while TCP remains stable. In contrast, on Aurora, MPI slightly outperforms TCP at lower scales and maintains competitive execution times at larger scales.
- Communication overhead is the primary bottleneck reducing MPI efficiency at larger scales, particularly evident on AWS.
- The MPI implementation convincingly demonstrates Dagger's ability to execute workloads across HPC interconnects with competitive performance on Aurora, validating its integration into high-performance systems, though further optimization is needed to enhance scalability at larger node counts.

Future Work

- Introduce collective operations, including remote memory access (RMA), to reduce communication overhead.
- Decentralize scheduling to eliminate coordinator bottlenecks.
- Expand the use of in-place data transfers for efficient communication.

Experimental Setup

- **Benchmark:** Cholesky decomposition (dense matrices).
- **Systems:** AWS r5d.24xlarge (96 CPUs, 768 GB RAM) and Aurora exascale supercomputer (81 nodes, 1 MPI rank/node, HPE Slingshot interconnect).
- **Scaling Models:** Assessed using strong scaling (fixed workload) and weak scaling (workload proportional to processors).

Execution Time vs Number of Processes

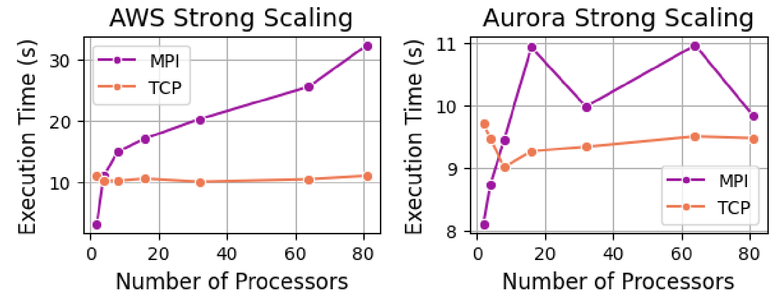

Figure 3. Strong scaling using MPI and TCP backends on AWS and Aurora with Float32 precision.

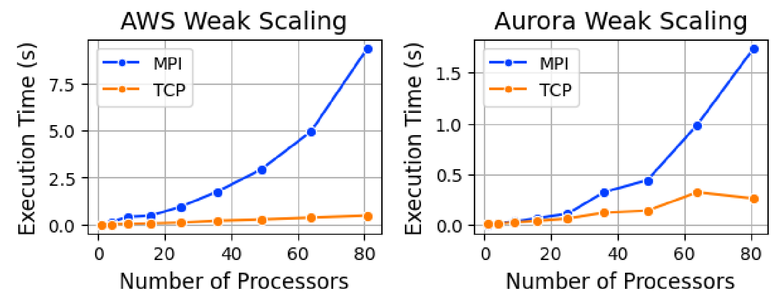

Figure 4. Weak scaling using MPI and TCP backends on AWS and Aurora with Float32 precision.