Implementing cellular sheaf optimization using ParitionedArrays

Getting results to accompany Tyler's new papers

Sam Cohen

Distributed Optimization with Sheaf Homological Constraints

Jakob Hansen
Department of Mathematics
University of Pennsylvania
Philadelphia, Pennsylvania
jhansen@math.upenn.edu

Robert Ghrist
Department of Mathematics
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, Pennsylvania
ghrist@math.upenn.edu

***Abstract*—In this paper we introduce a new class of local linear operators on graphs, the *sheaf Laplacians*, which provide drop-in replacements for graph Laplacians in distributed algorithms. These operators can enforce more general constraints on data distributed in a network than those given by the graph Laplacian. The constraints for such optimization problems can be framed in the context of *sheaf cohomology*, leading to a description of this framework as distributed optimization with homological constraints. We formulate a representative problem, elucidate its solution with sheaf Laplacians, and give illustrative examples of potential applications.**

***Index Terms*—optimization, distributed systems, spectral graph theory, algebraic topology, cellular sheaf**

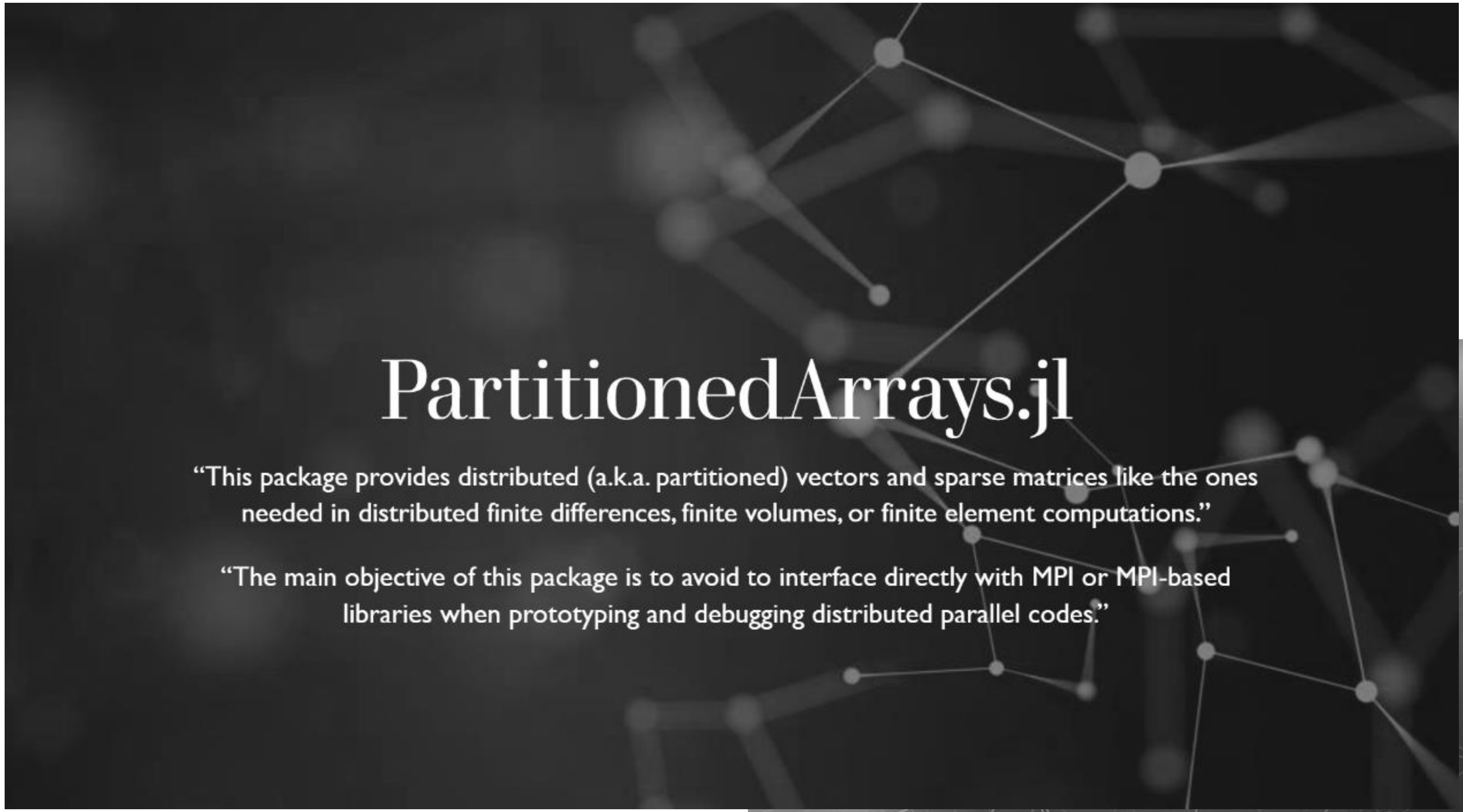
I. Introduction

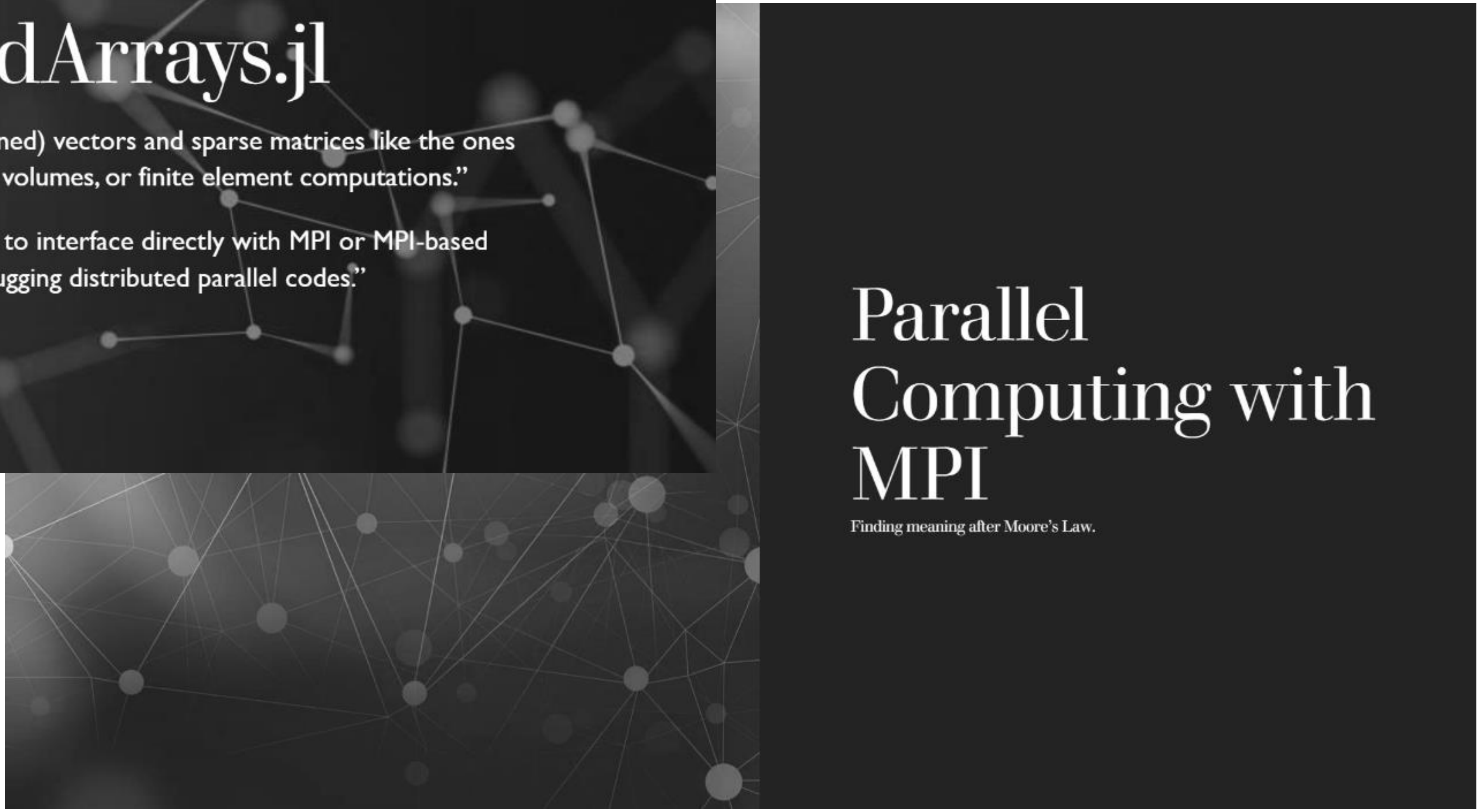
Distributed optimization is a broad field with applications in machine learning, systems control, sensor networks, and many other areas. Algorithms for distributed optimization typically act over a network modeled as a graph, with each node both performing local optimization over its own

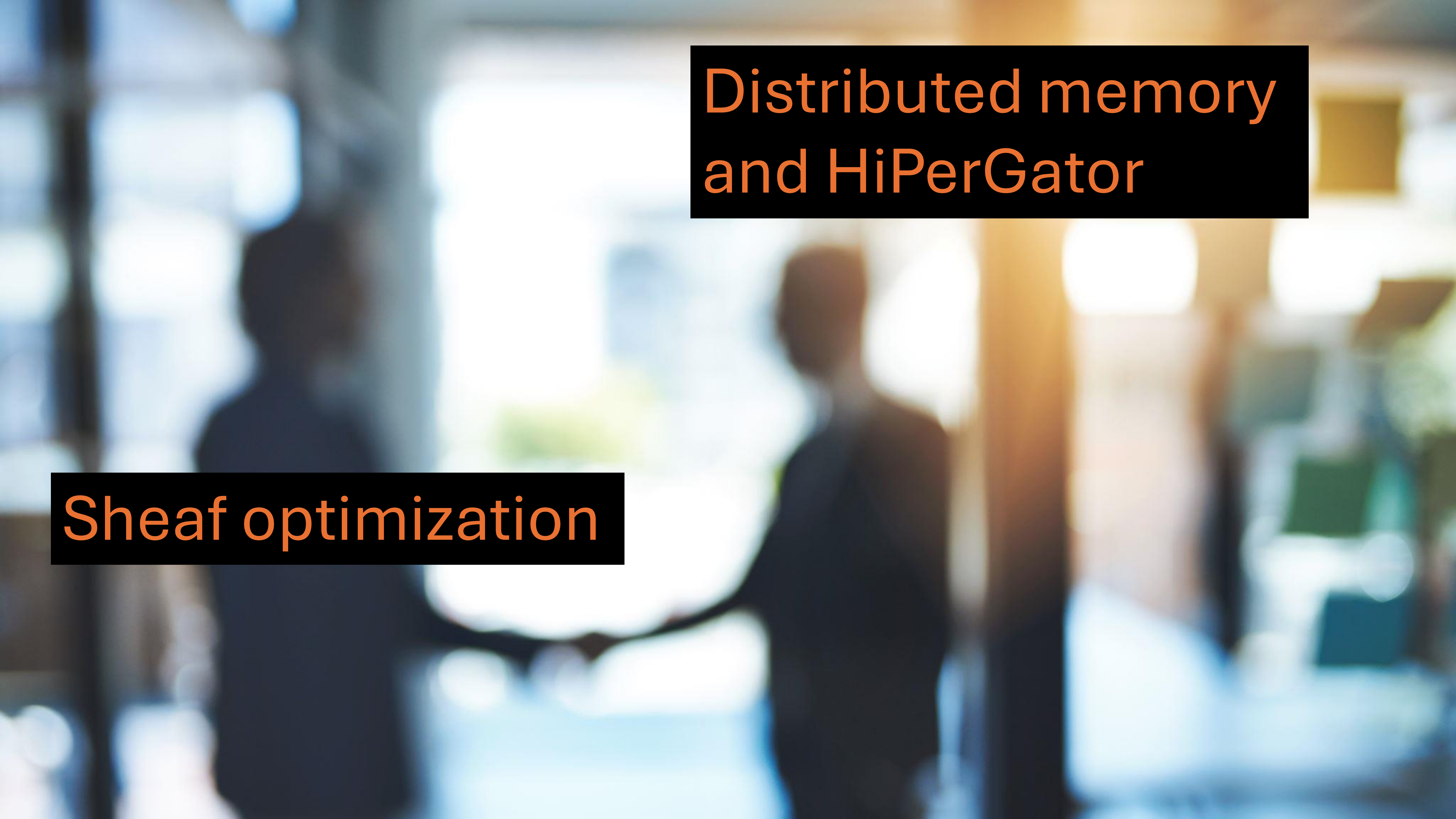
II. Distributed Optimization

Distributed optimization has an illustrious history spanning a diversity of models and approaches. Some of the earliest work is due to Tsitsiklis and collaborators, studying asynchronous methods where all agents know the entire objective function [3]. More recent work has applied other notions from convex optimization such as primal-dual methods and the alternating direction method of multipliers [4], [5]. Other work has studied situations where agents have only partial knowledge of the objective function [6], [7]. Algorithms for these problems often involve a combination of a local optimization process (e.g., gradient descent) and a global consensus process (e.g., graph diffusion). These processes can be implemented and integrated in different ways, and the ultimate goals and assumptions of these algorithms vary. The function to be optimized may be known to all agents in the network, or it may be broken up into a sum of

```
# Dr. Fairbanks' approach: Separate vertex and edge workers

mutable struct SheafVertex
    adj::Vector{}  # It's actually a Vector{SheafEdge} but this was causing a circular dependency problem. Use Vector{Int} instead?
    f::Function    # Objective function at vertex
    x::Vector  # Primary variables
    z::Vector  # Dual variables

    # x_update::Vector   # This would be if we wanted to separate the compute and update steps, as above
    # z_update::Vector
end

struct SheafEdge    # Note-- not mutable!
    src::SheafVertex         # How to separate out src, target? Also, should src be an actual sheafnode, not just an int?
    tgt::SheafVertex
    left::Array
    right::Array
end

function SheafVertex(f::Function, x::Vector, z::Vector)
    return SheafVertex([], f, x, z)
end

function add_edge!(u::SheafVertex, v::SheafVertex, u_map::Array, v_map::Array)
    edge = SheafEdge(u, v, u_map, v_map)
    push!(u.adj, edge)
    push!(v.adj, edge)
end

# "Asks the shared edge" to calculate the local update to the primal x variable
function xLaplacian(v::SheafVertex, e::SheafEdge)
    if v == e.src
        return -2 * e.left' * (e.left * e.src.x - e.right * e.tgt.x) - e.left' * (e.left * e.src.z - e.right * e.tgt.z)
    else
        return 2 * e.right' * (e.left * e.src.x - e.right * e.tgt.x) + e.right' * (e.left * e.src.z - e.right * e.tgt.z)
    end
end
```


PartitionedArrays.jl
“This package provides distributed (a.k.a. partitioned) vectors and sparse matrices like the ones needed in distributed finite differences, finite volumes, or finite element computations.”
“The main objective of this package is to avoid to interface directly with MPI or MPI-based libraries when prototyping and debugging distributed parallel codes.”


Parallel Computing with MPI
Finding meaning after Moore’s Law.



Distributed memory and HiPerGator

Sheaf optimization

Goals for the next 2-5 months

1. Implement the Hansen & Ghrist paper using PartitionedArrays
2. Run on HiPerGator and get results
3. Implement Tyler’s new ADMM paper on top of system from (1)
4. Run on HiPerGator and get results
5. Implement Tyler’s new Newton’s Method paper
6. Run on HiPerGator and get results
7. Work with Tyler to produce at least 1 relevant application
   - Candidates: multitask learning, domain decomposition, multi-agent control
8. Publish in ICML (due Jan 1) with fallback NeurIPS (due May), or potentially with LICS and CDC (March)

The road ahead

Longer-term goals

1. Implement new optimization algorithms as Tyler creates them
2. Help create those new optimization algorithms
3. Implement sheaf optimization using other models of distributed computing (ex. shared memory) and benchmark
4. Develop lots of applications with good documentation so that other people can actually use sheaf optimization
5. Research theoretically optimal ways to exploit sheaf/graph structures when partitioning on a GPU cluster (my master’s thesis? 👀)
6. Learn a ton about category theory, distributed computing, and optimization along the way! 🎉 🏫

Thanks for listening!