JAX & OpenXLA DevLab, Fall 2025

Sharding the Sphere with JAX

Richard Loft
AreandDee LLC
November 18, 2025

Outline

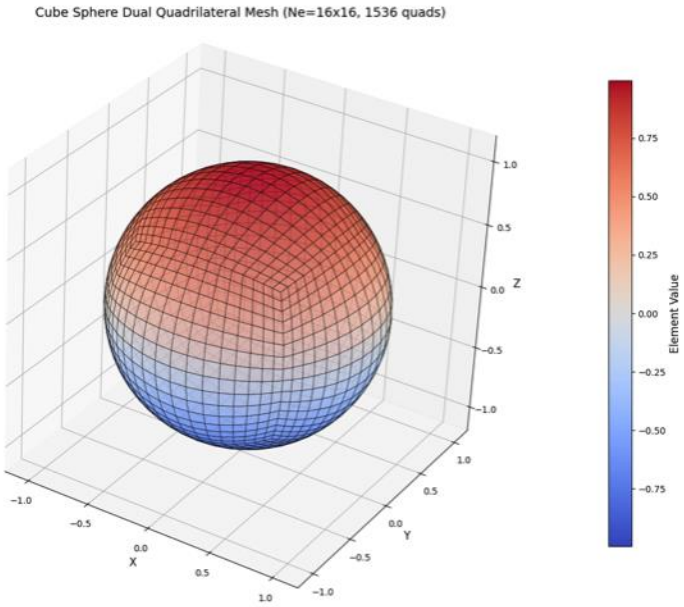
- Project Overview
- Parallel Code Deep Dive
- Cool Videos

Project Overview: Global Atmospheric Dynamics

- **The Atmospheric Modeling Bottleneck**
  - Doubling resolution → 8× compute cost.
- **Tensor-Train (TT)**
  - TT compresses N×N fields → $O(dNr^2)$, $r << N$
  - Numerics operates directly on compressed cores.
  - Scientists at LANL (Danis et al., 2024):
    - Applied TT to Cartesian2D, SWEs
    - 124× speedup, accuracy preserved
- **Our Research Questions**
  - Can TT be applied to global atmospheric equations?
  - Can we use JAX and TT to help unify the HPC and AI ecosystems?

Our Numerical Approach

- **Finite Volume (PLR) Method**
  - A solid, 2nd Order method
- **Cubed-Sphere**
  - 6, TT-friendly 2D tiles
- **JAX Ecosystem**
  - Portable
  - Built-in parallelism (sharding)
  - Checkpoint/restart (Orbax).

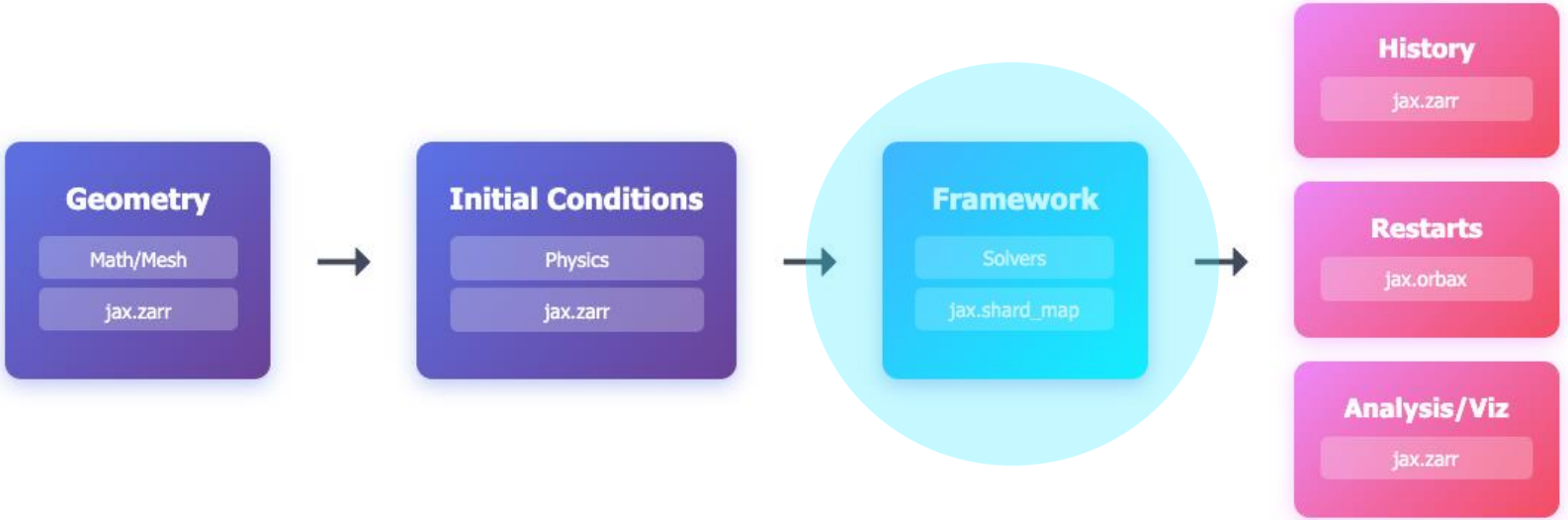


Pros and Cons of Tensor Train

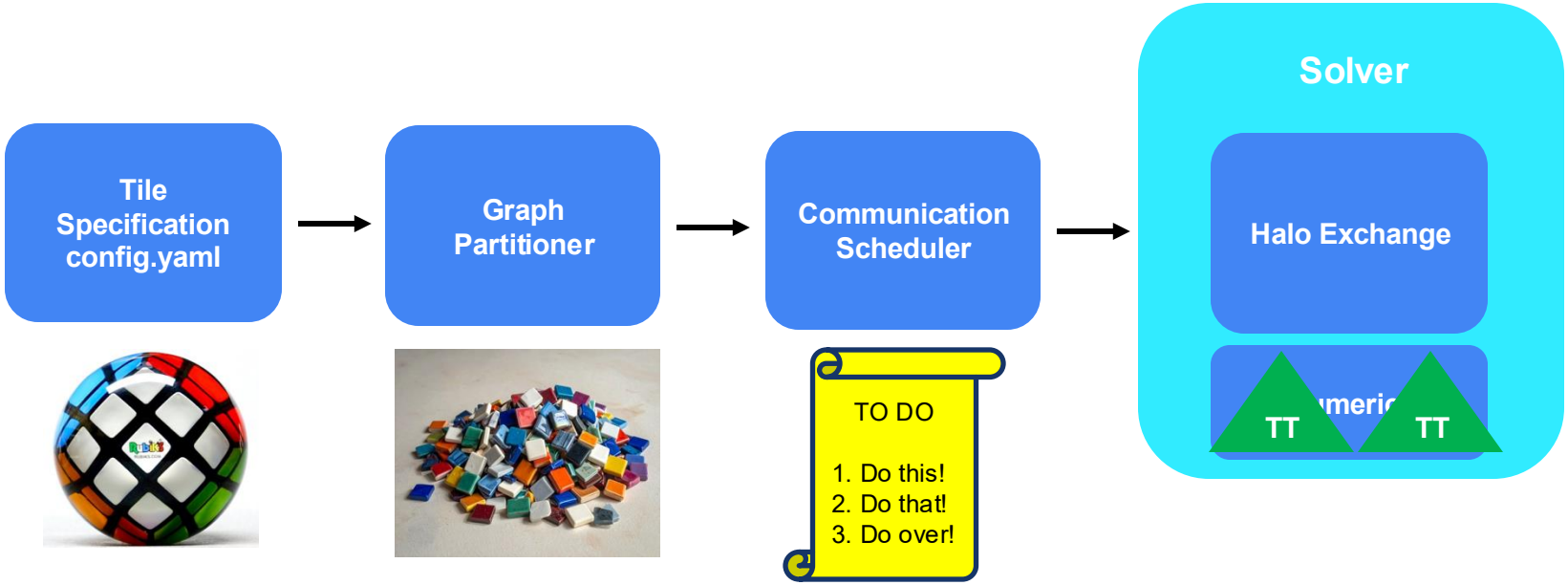
- **Are atmospheric flows TT compressible?**
  - Large scales should be TT compressible
  - Smaller scales (<10 km) could be challenging!
- **Computational Advantages**
  - memory-bound numerics → compute-bound TT
  - Ideal for TPU/GPU devices.

JAXStream Framework

Atmospheric Simulation Pipeline



Zooming in on the Solver
… and the parallelization pipeline



Sharding…

- **Parallelism**
  - Controlled from .yaml file
  - Tiling will allow more flexible device counts.
- **`setup_sharding`:**
  - **`jax.devices()`** — "what devices do I have?"
- **Sharding**
  - **`Mesh(devices,'tiles')`**
    - create a device mesh named 'tiles'
  - **`P('tiles',)`**
    - Partition the 1st array dimension sequentially along the 'tiles' axis
- Jax enforces **`num_tiles%num_devices = 0`**

Setup_sharding

```
def setup_sharding(self):
    print(f"    total tiles: {num_tiles} (6 faces × {tiles_per_edge}² tiles/face)")
    print(f"    tiles per device: {num_tiles / num_devices:.1f}")

    if device_type == 'cpu':
        # Create virtual devices for CPU testing
        os.environ['XLA_FLAGS'] = f'--xla_force_host_platform_device_count={num_devices}'
        print(f"  Device type: CPU (virtual devices)")
        print(f"  XLA_FLAGS set: {num_devices} virtual CPU devices")
    else:
        print(f"  Device type: GPU")
        print(f"  Requested devices: {num_devices}")

    devices = jax.devices()[:num_devices]
    print(f"  Available devices: {len(jax.devices())}")
    print(f"  Using devices: {devices}")

    # Create mesh: 1D array of devices mapped to 'tiles' axis
    self.mesh = Mesh(devices, ('tiles',))
    self.sharding = NamedSharding(self.mesh, P('tiles'))

    print(f"  Mesh created: {num_tiles} tiles across {len(devices)} device(s)")
    print(f"  Sharding strategy: PartitionSpec('tiles') on axis 0")
    if num_tiles > len(devices):
        print(f"  Note: Multiple tiles per device (serial execution)")
    print(f"{'='*70}\n")
```

Communication Schedule

- **Why Schedules?**
  - Avoids Deadlocks and Race Conditions
- **The Problem:**
  - There are 12 edges on a cube
  - Some (4) edges need transposition and/or reversal.
  - Must avoid simultaneous read/write conflicts to the same device.
- **For 6 devices, 4 stages are required.**
  - No device appears twice in the same communication stage.
- **General solution:**
  - Scalable edge coloring algorithm.

```python
def create_communication_schedule():
    """
    12 buffer swaps in 4 non-blocking stages.

    Format: ((face_a, edge_a), (face_b, edge_b), operations)

    Returns:
        Tuple of 4 stages, each containing 3 edge pairs
    """
    return (
        # Stage 0
        (
            ((0, "N"), (1, "N"), "R"),
            ((3, "E"), (4, "W"), "N"),
            ((2, "S"), (5, "E"), "TR")
        ),
        # Stage 1
        (
            ((0, "E"), (4, "N"), "T"),
            ((2, "E"), (3, "W"), "N"),
            ((1, "S"), (5, "N"), "N")
        ),
        # Stage 2
        (
            ((0, "W"), (2, "N"), "TR"),
            ((1, "W"), (4, "E"), "N"),
            ((3, "S"), (5, "S"), "R")
        ),
        # Stage 3
        (
            ((0, "S"), (3, "N"), "N"),
            ((1, "E"), (2, "W"), "N"),
            ((4, "S"), (5, "W"), "T")
        )
    )
```

The Halo Exchange Factory

No recompilation during timestepping!

Warding off Jax JIT compilations:

- **`functools.partial`**
  - freezes static arguments
  - (face IDs, edges, N)
- Each **`exchange_fn`** compiles once at initialization

Why Two JITs?

- **Without the second JIT:**
  - 12 separate JIT-compiled functions
  - Python loop overhead between each call
  - Less optimization across boundaries
- **With the second JIT (composed):**
  - JAX sees the entire sequence
  - Can optimize across function boundaries
  - Eliminates Python loop overhead

Freeze!

Level 1

Level 2

```python
def make_halo_exchange(schedule, N):

    exchange_functions = []

    # Pre-compile each exchange in schedule
    for stage_idx, stage in enumerate(schedule):
        for (face_a, edge_a), (face_b, edge_b), operations in stage:
            # Use partial to bake in static arguments
            exchange_fn = partial(
                exchange_edge_pair,
                face_a=face_a, edge_a=edge_a,
                face_b=face_b, edge_b=edge_b,
                operations=operations, N=N
            )

            # JIT compile once
            exchange_fn_jit = jax.jit(exchange_fn)
            exchange_functions.append(exchange_fn_jit)

    # Return composed function that applies all exchanges
    def cubesphere_halo_exchange(field_ghosts):
        """Apply all pre-compiled exchanges."""
        for exchange_fn in exchange_functions:
            field_ghosts = exchange_fn(field_ghosts)
        return field_ghosts

    # JIT compile the composed function for additional optimization

    return jax.jit(cubesphere_halo_exchange)
```

Core method of the Factory, `exchange_edge_pair`

Key Features of Edge Swapper:

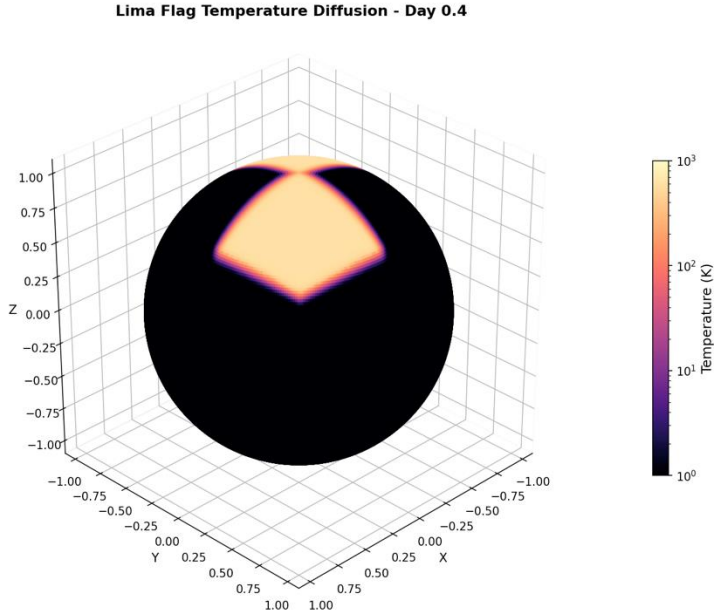
- **Bidirectional:**
  - Both edges exchange simultaneously!
- In-place update using JAX's `.at[].set()` syntax
- Handles transpose with simple array operation.
- Flatten/reshape for compatibility with sharding.

What is `.at[].set()` and why is it needed?

- **JAX Arrays Are Immutable**
  - Consider `arr[0] = 5`
  - `arr.at[0].set[5]`
  - returns a **new** copy of `arr`, **with** `arr[0]=5`
- **Required for Multi-Device Sharding**
  - **Devices can work with their own private copy**
  - **No conflicts!**
- **Enables Compiler Optimizations**
  - Copy fusing, reordering, elimination and tracking

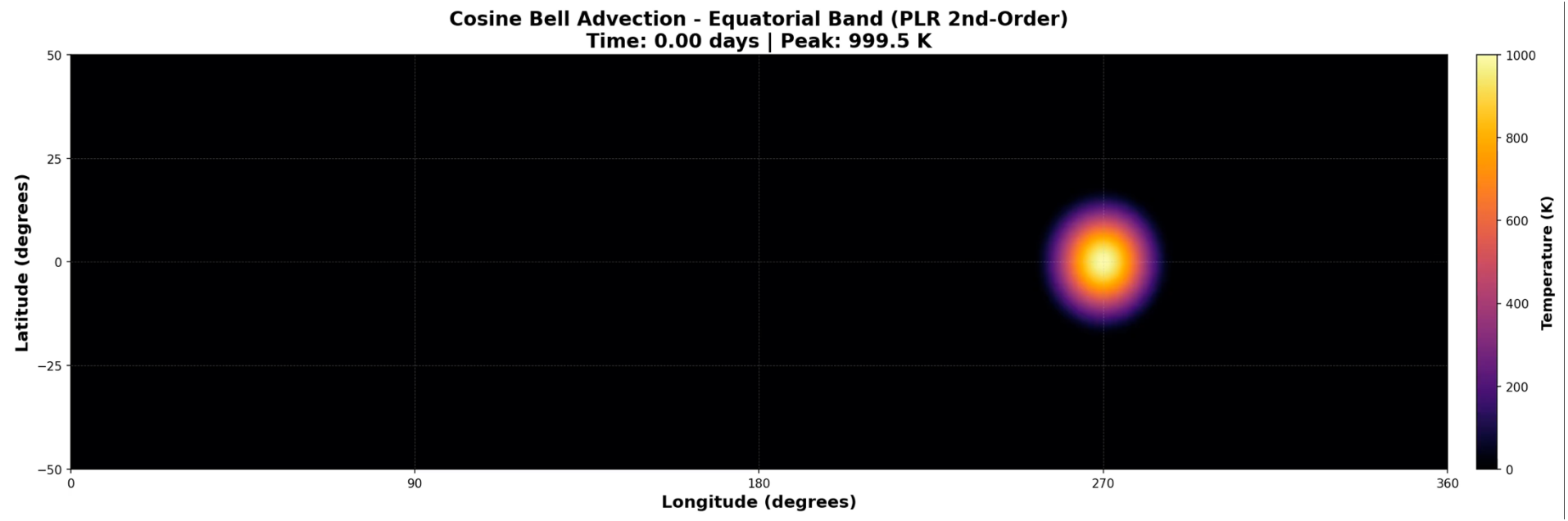
```python
@jax.jit
def exchange_edge_pair(field_ghosts, face_a, edge_a, face_b, edge_b,
                       operations, N):
    """Swap data between two edges"""

    # Extract edge data from interior boundaries
    data_a = extract_boundary_data(field_ghosts[face_a], edge_a, N)
    data_b = extract_boundary_data(field_ghosts[face_b], edge_b, N)

    # Apply rotation if needed (90° = reverse order)
    data_to_b = apply_operations(data_a, operations)
    data_to_a = apply_operations(data_b, operations)

    # Write to ghost cells using .at[].set() syntax
    field_ghosts = field_ghosts.at[face_b].set(
        set_ghost_data(field_ghosts[face_b], edge_b, data_to_b, N))
    field_ghosts = field_ghosts.at[face_a].set(
        set_ghost_data(field_ghosts[face_a], edge_a, data_to_a, N))

    return field_ghosts
```

Thermal Diffusion Visualization…



Proof that sharding works:
a checkerboard heat source on the top panel!

Tracer Advection on the sphere…



Proof that sharding works with winds!

References

- Finite Volume: Putman, W. M., & Lin, S.-J. (2007). Finite-volume transport on various cubed-sphere grids. Journal of Computational Physics, 227(1), 55–78. https://doi.org/10.1016/j.jcp.2007.07.022

- Tensor Train: Danis, M. E., Truong, D. P., DeSantis, D., Petersen, M., Rasmussen, K. Ø., & Alexandrov, B. S. (2024). High-order tensor-train finite volume method for shallow water equations. arXiv preprint arXiv:2408.03483. https://arxiv.org/abs/2408.03483

Thank you!

Learning more

Access the example code at github/areanddee/???

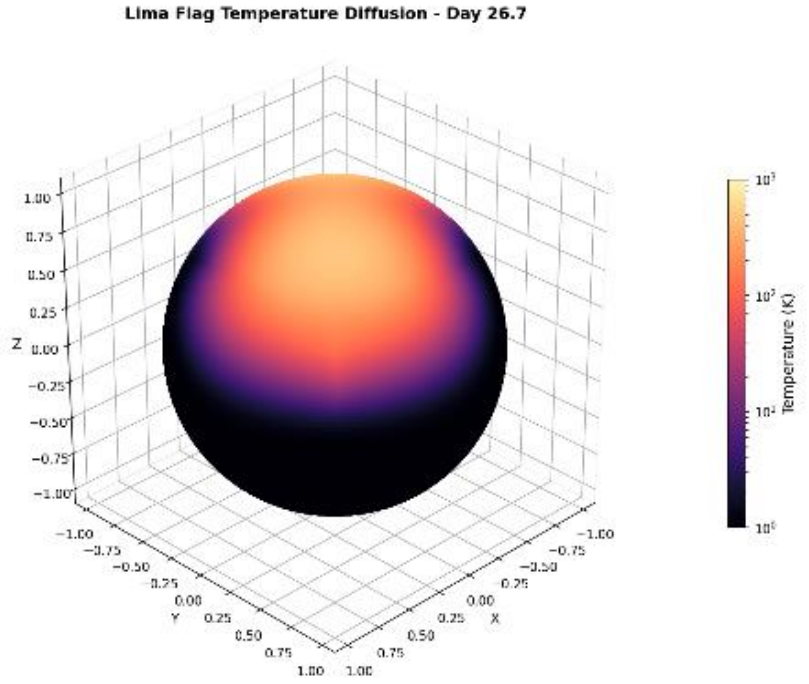
To contribute to JAXStream email us at contact@areanddee.com

This project was generously funded by a Google Gift to the Colorado State University Foundation.

Richard D. Loft
richard.d.loft@areanddee.com

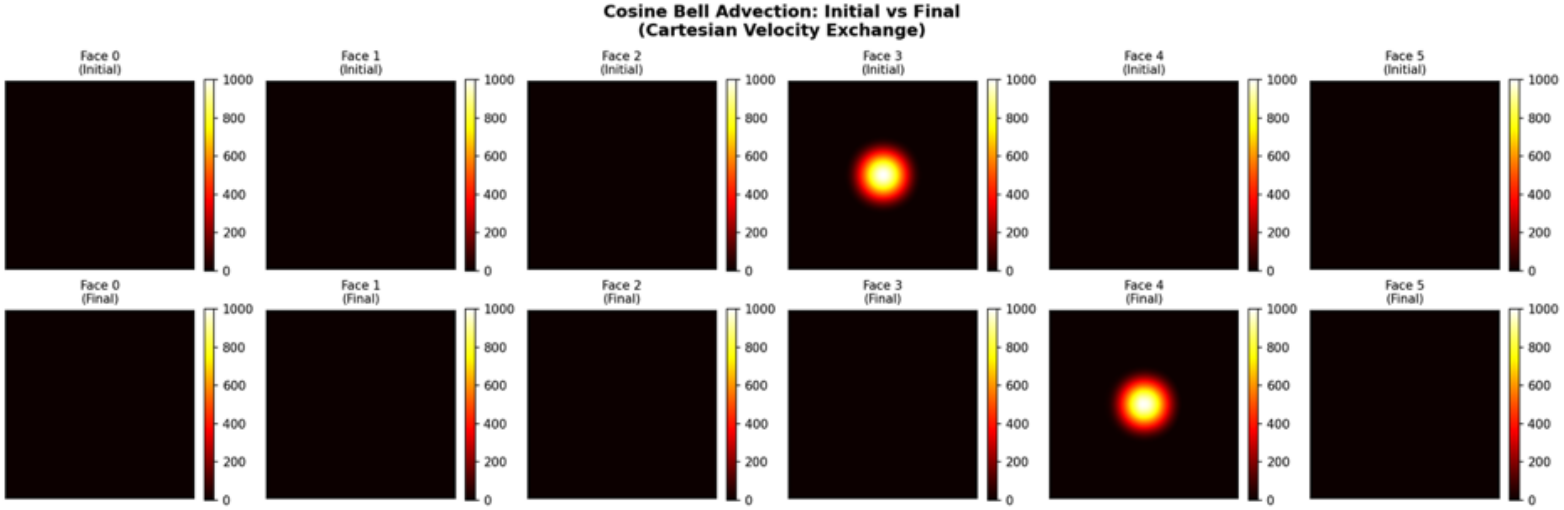
Backup Slides

Thermal Diffusion Visualization…



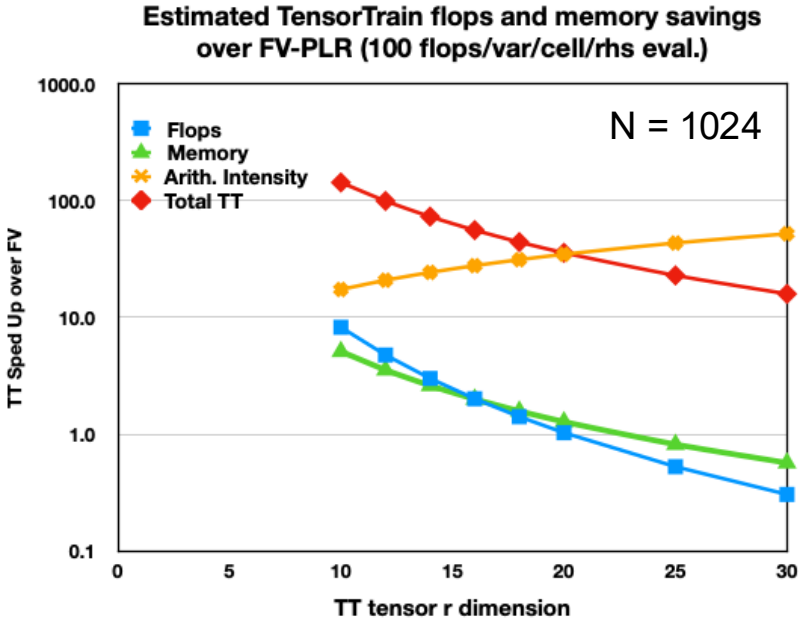
Proof that sharding works:
a checkerboard heat source on the top panel!

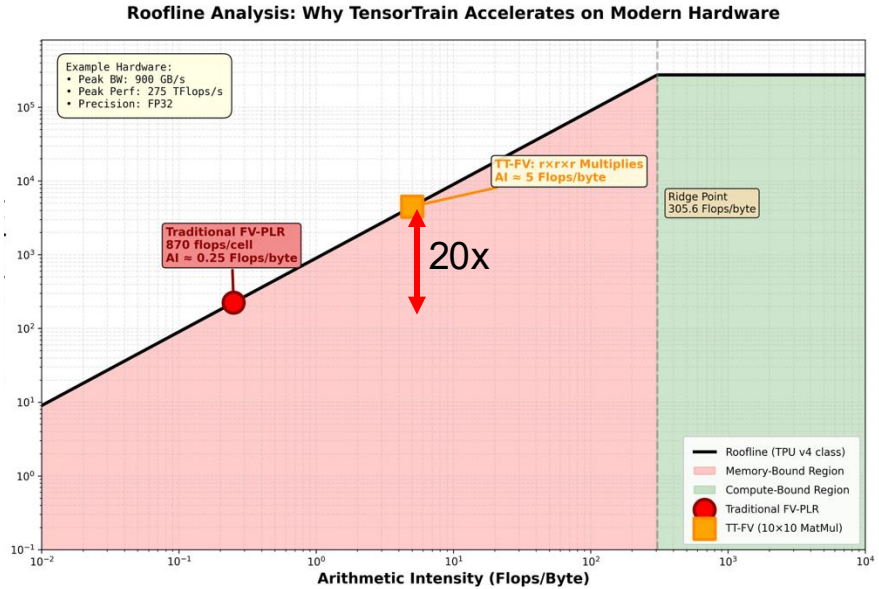
Tracer Advection on the sphere…



Proof that sharding works with winds!

TensorTrain: the math behind the speedup…





Two keys to TT performance: fewer flops (left) are performed faster (right)

Deck template

Speaker Instructions & Next Steps

JAX & OpenXLA DevLab, Fall 2025

OpenXLA

Timeline and Key Dates

- Now: Use the template in this deck, and start drafting your talk.
- Week of October 12th: You will receive calendar invites for your session, your roundtable, and a deck review.
- Week of November 10th. Deck reviews with DevRel & OPM.
- November 18 and 19th: **Present!** The DevLab will be in SVL MP6. You can find the most up to date agenda here.
- November 18: **Staff your roundtable!** All sessions will have a roundtable in the afternoon. This is an informal opportunity for you to chat with attendees about your talk. No extra preparation needed. Just sit at the table with your name on it, and chat with folks.

Speaker Instructions

Session Goals

- The best sessions are educational. You can share product updates and new features, or discuss your research. **Tutorials and walkthroughs are fantastic**. You can also share opportunities for collaboration.

Format

- You can find the agenda here: https://rsvp.withgoogle.com/events/devlab-fall-2025
- Your speaking slot is 15 mins, 30 mins, or 60 mins. We encourage you to use the time flexibly. For instance, you could plan a 45-minute presentation followed by 15 minutes for Q&A. If you're not sure how long your session is, contact edgarchen@google.com
- **Googlers**: All of the code you present should be **external friendly**, so users can follow along without access to a corporate account. You should present using https://colab.research.google.com/ (public colab, using a browser logged into a **gmail account**), not an internal version logged into your Google account.
- You can present a mix of slides and code. If you present code, please be sure to include a link to your code on GitHub on your sides, so users can find it easily.
- All sessions must be reviewed before the DevLab.
- Feel free to use your own laptop for presenting.