The Lattice Boltzmann Method in Kokkos

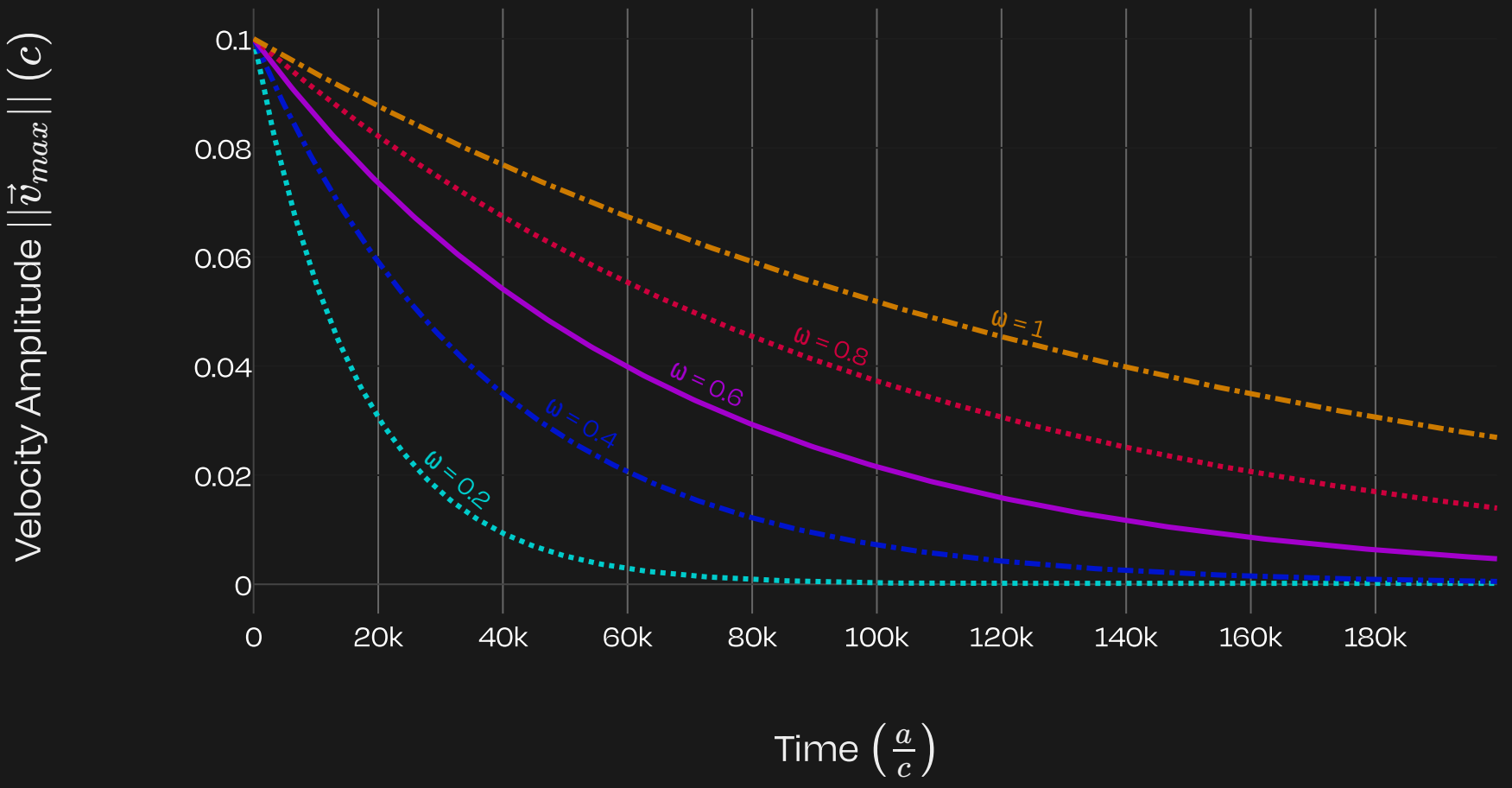
Performance Lessons Learned

Julian Karrer

Correctness

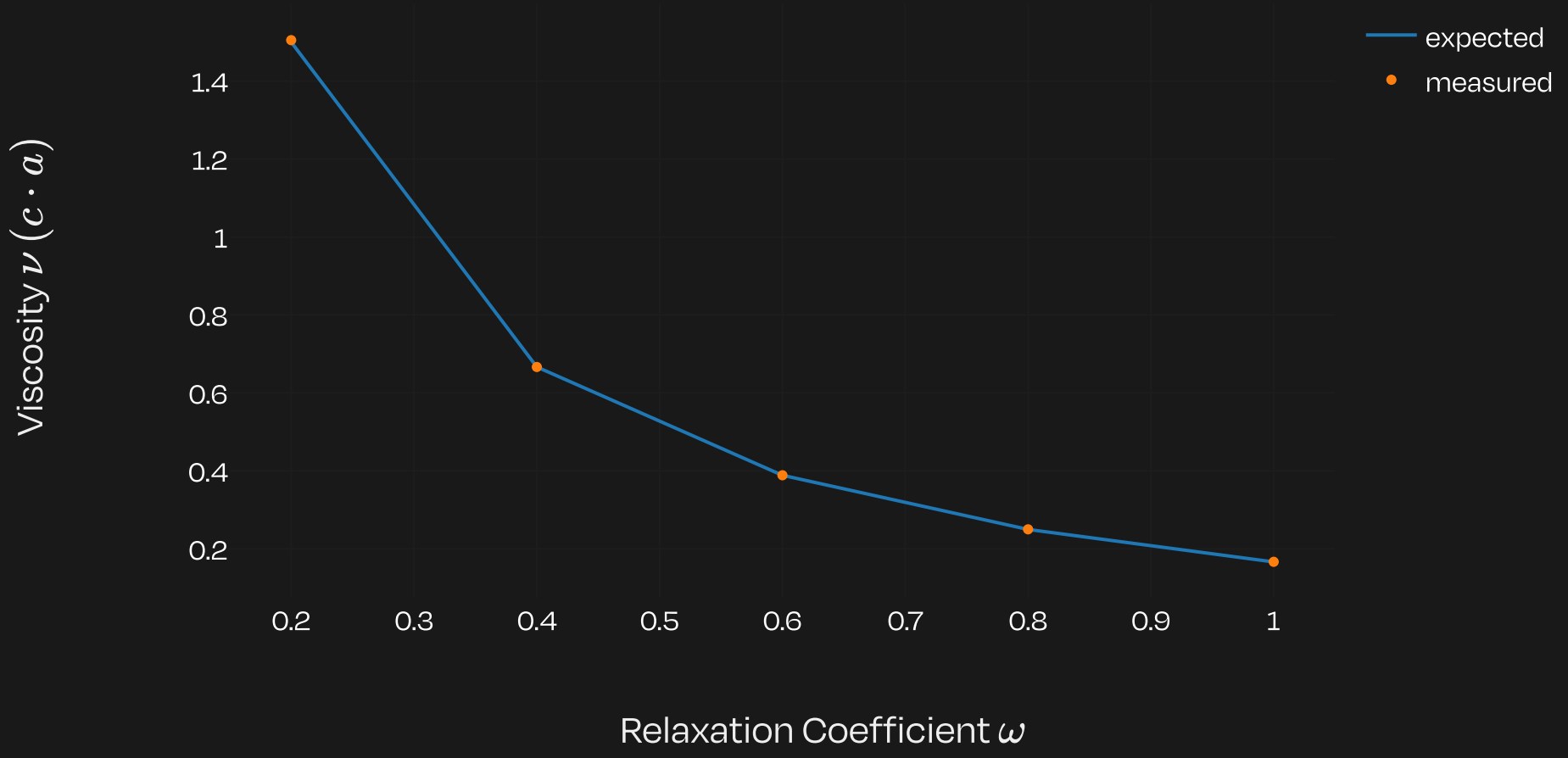
$1920 \times 1080\,\text{Domain}\,, \omega = 1.7, \vec{u}_x^{\text{inlet}} = 0.05$

Shearwave decay



200k time steps of D2Q9 shearwave decay with $u_x(t = 0) = 0.1$, *[Math Processing Error]* on 1000×1000 nodes

Shearwave decay



200k time steps of D2Q9 shearwave decay with $u_x(t=0) = 0.1$, *[Math Processing Error]* on 1000×1000 nodes

1. Hand-Tuning and Precision

Micro-optimizations

1. Manual Loop-Unrolling
2. Common Subexpression Elimination
3. Arithmetic Simplification
4. Read/Write reordering

NO significant difference ($\pm\sigma$) for any tested GPU

Precision

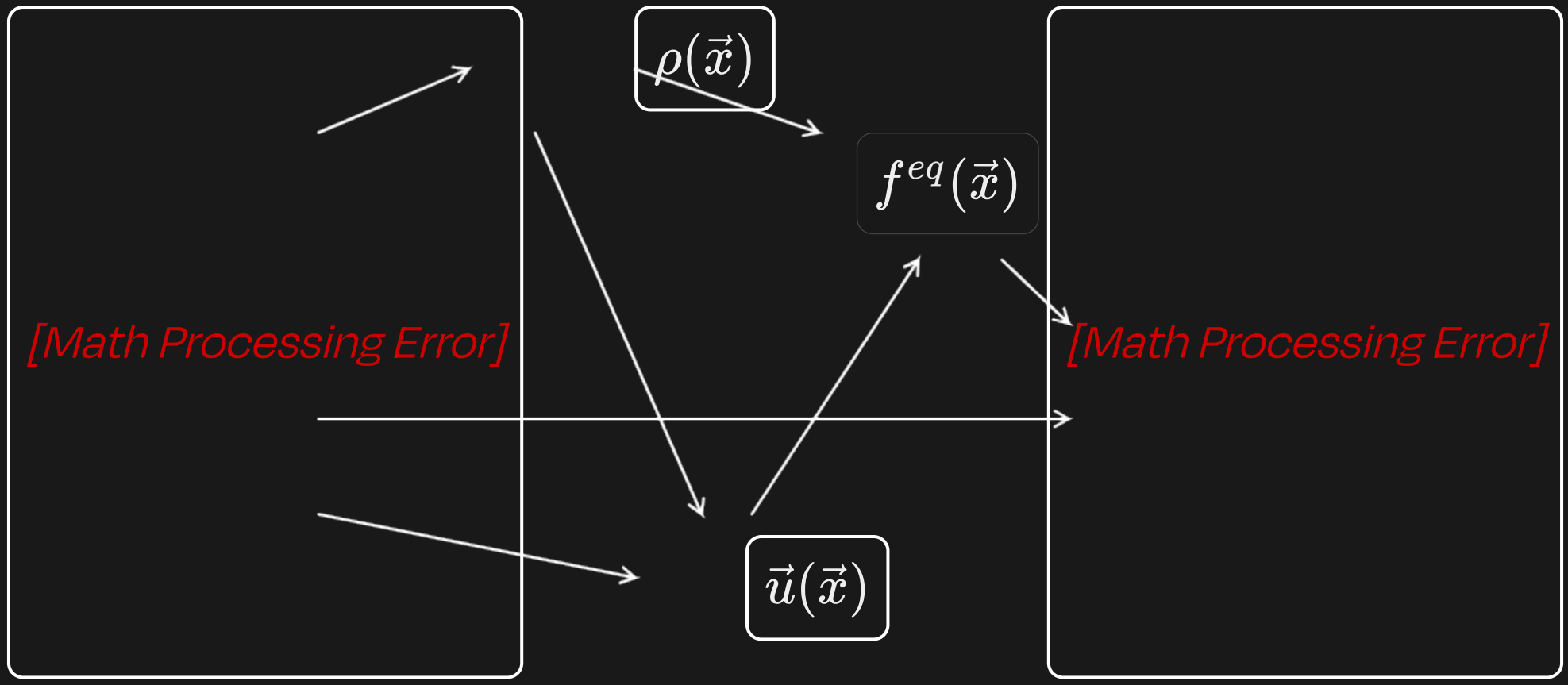
- Use `float` instead of `double` if possible!

GPU	relative Speedup for **`float`**
3060Ti	$+102\%$
MI300A	$+100\%$
H100	$+101\%$

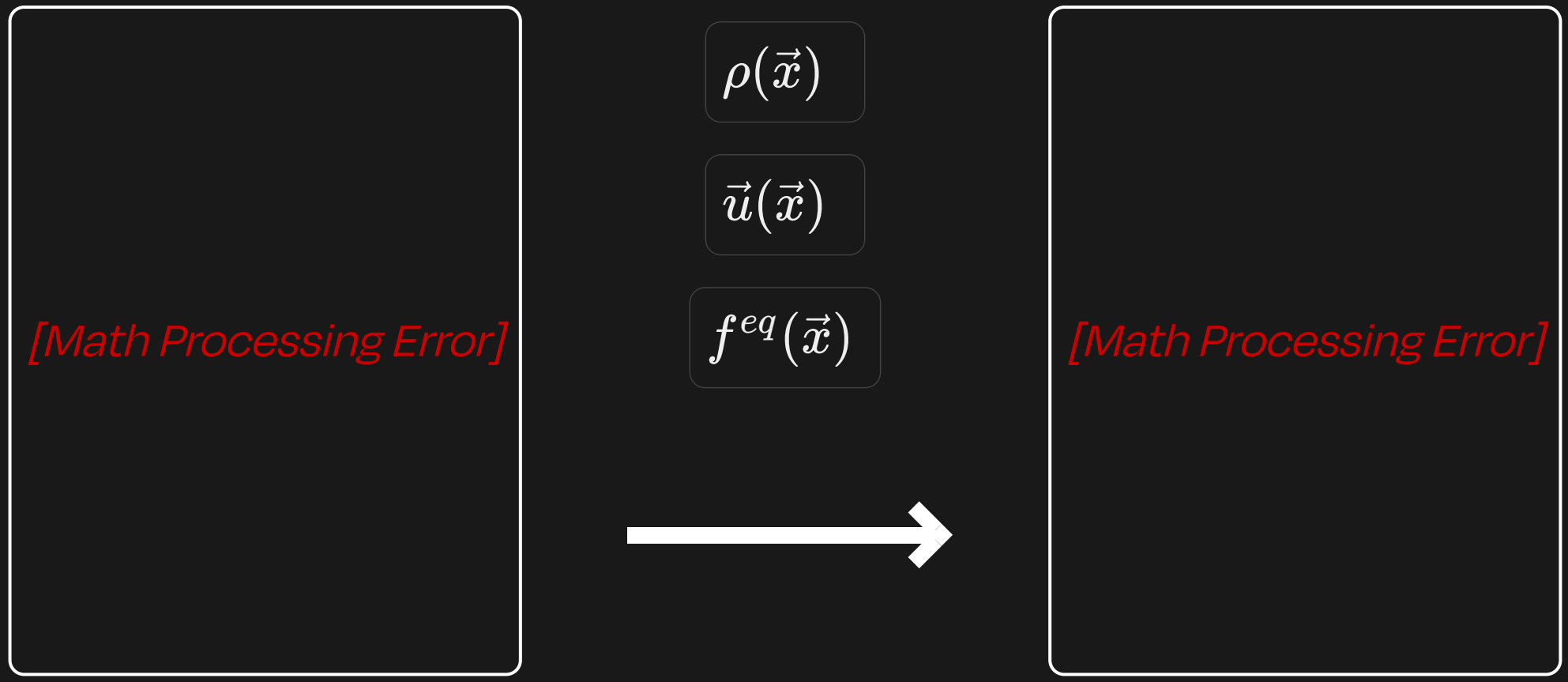
- Factor of two for variety of hardware

2. Efficiently use Memory Bandwidth

Data Dependencies

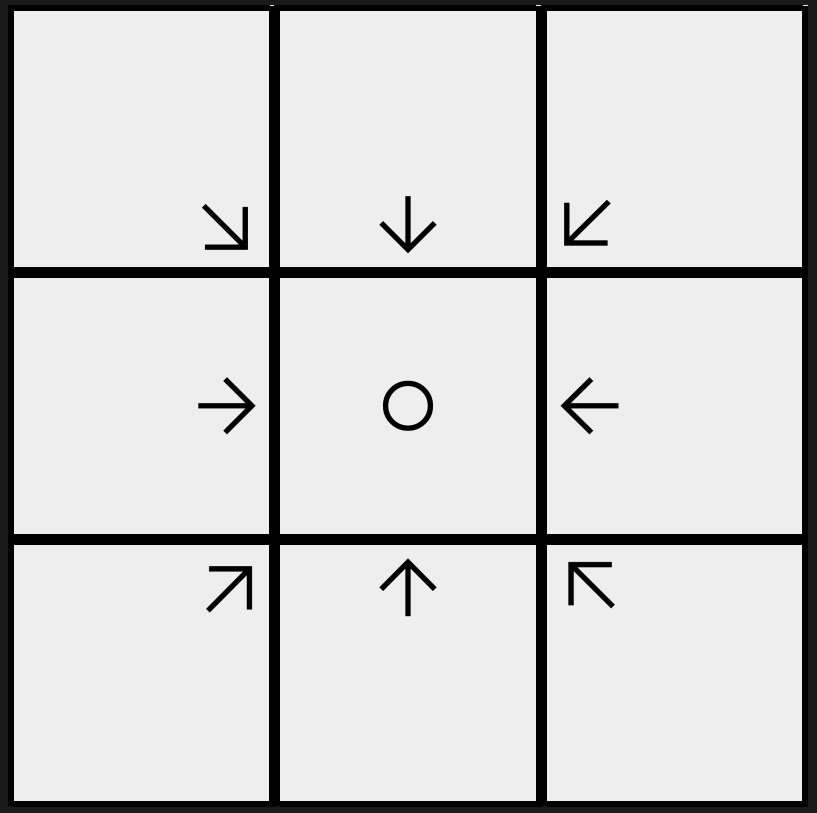


Data Dependencies



Fused Stream-Collide

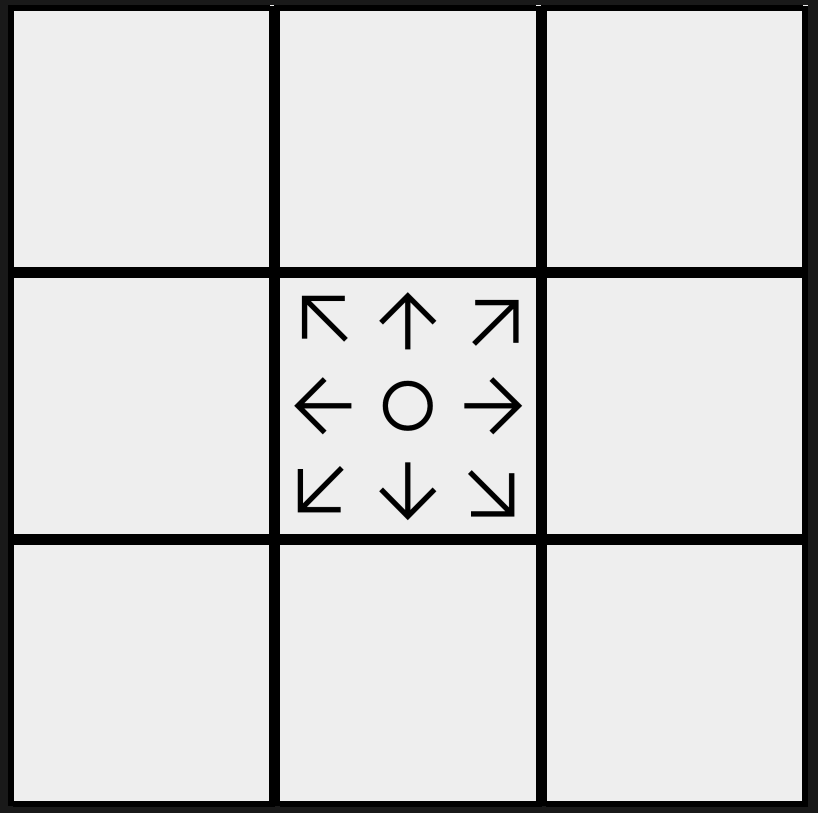
Pull



Read from neighbours

Write locally

Push



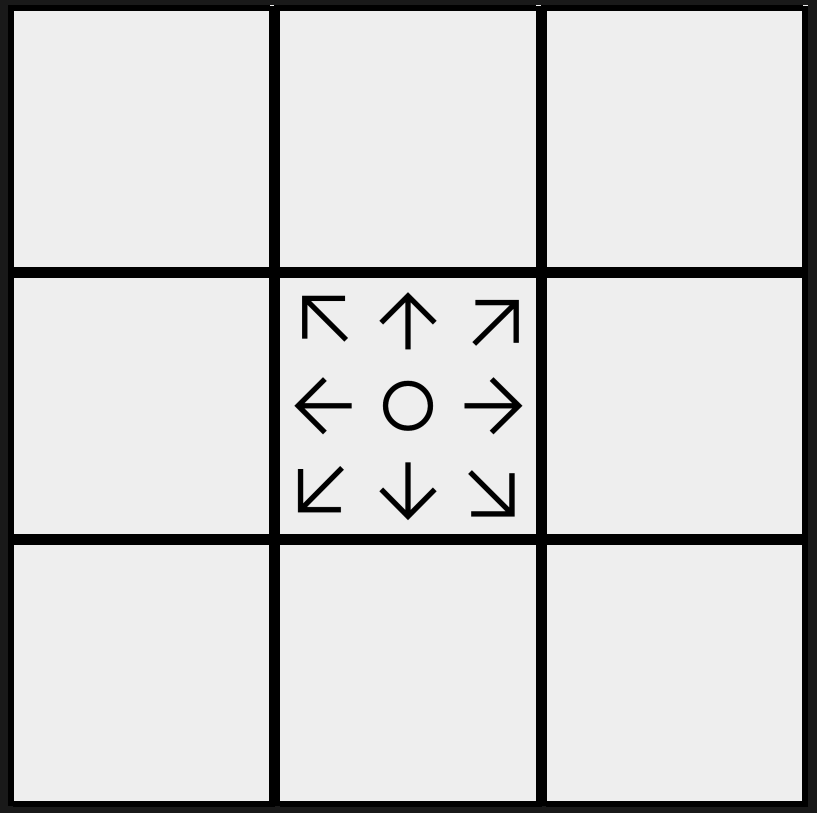
Read locally

Write to neighbours

[Wittmann et al. 2012]

Fused Stream-Collide

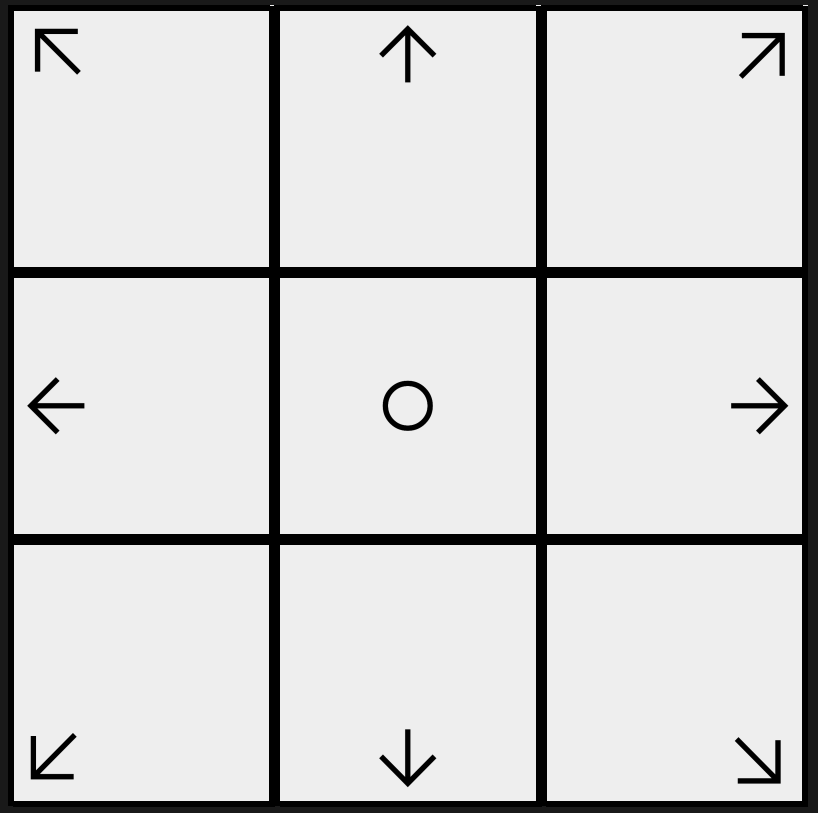
Pull



Read from neighbours

Write locally

Push

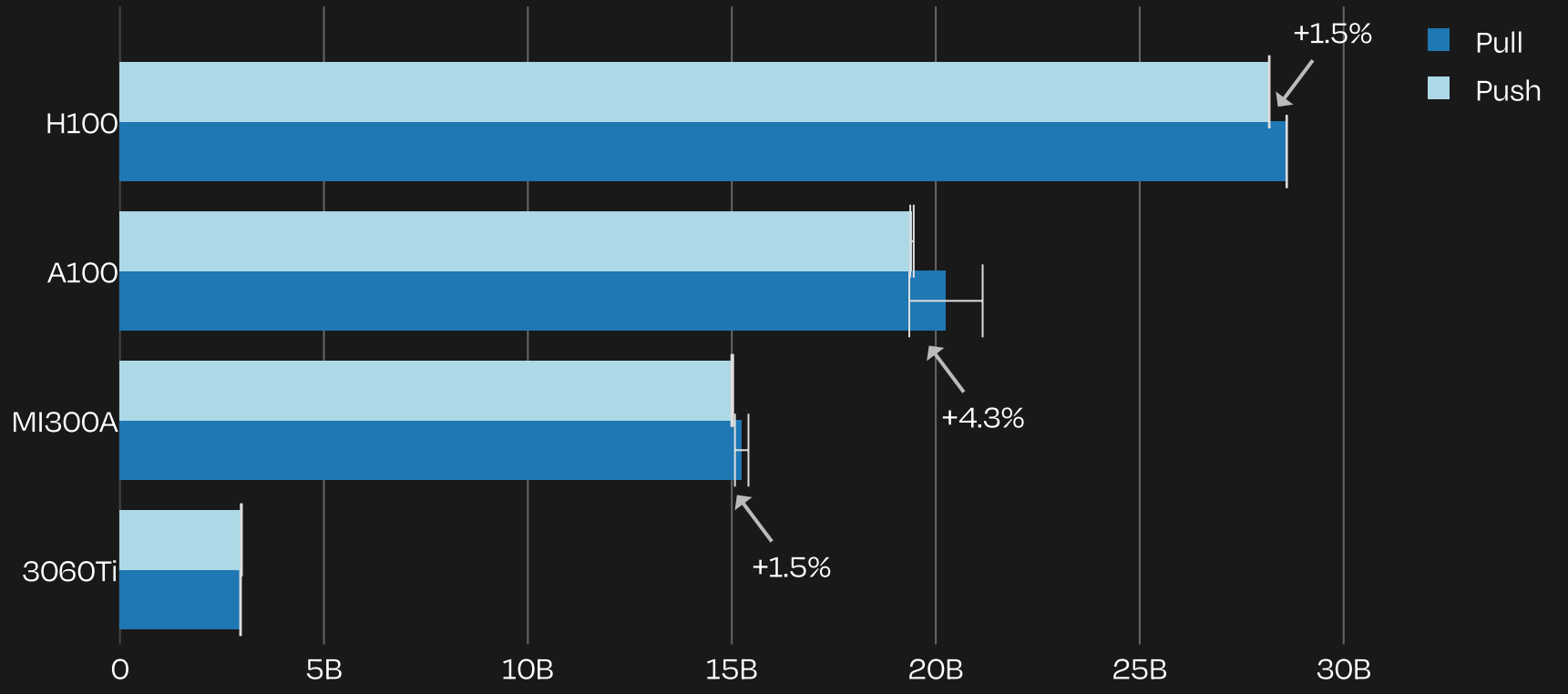


Read locally

Write to neighbours

[Wittmann et al. 2012]

Results - Push vs. Pull



Lattice Updates per Second *[Math Processing Error]*

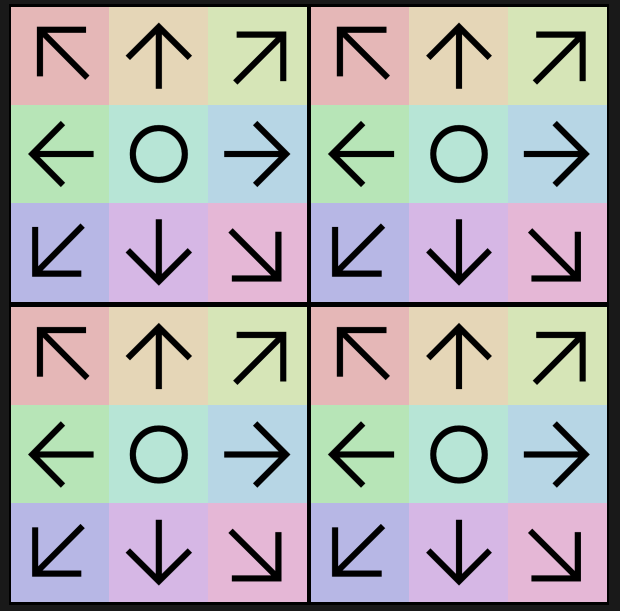
Peak performance: 28.6 BLUPS (H100), 22.0 BLUPS(A100)

A100 and H100 (32000 × 32000), 3060Ti (3000 × 3000), 100 steps, ≥ 5 repeats, D2Q9 Shearwave Decay

3. Use Coalescing Memory Accesses

Memory Layout

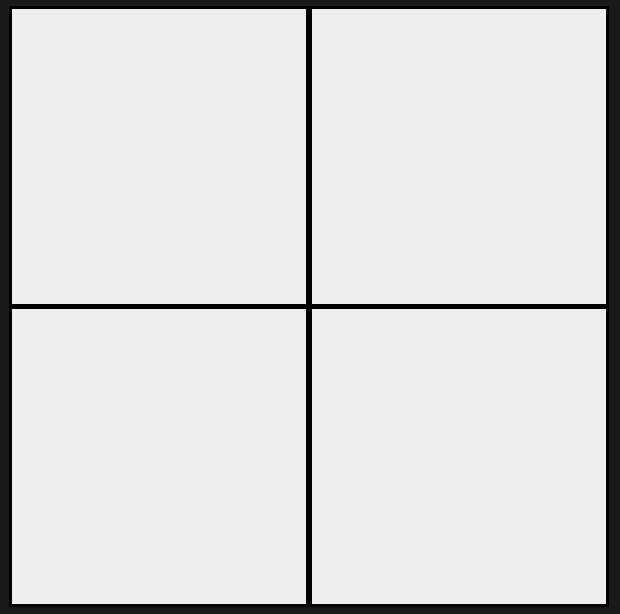
Array of Structs - uncoalesced



[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

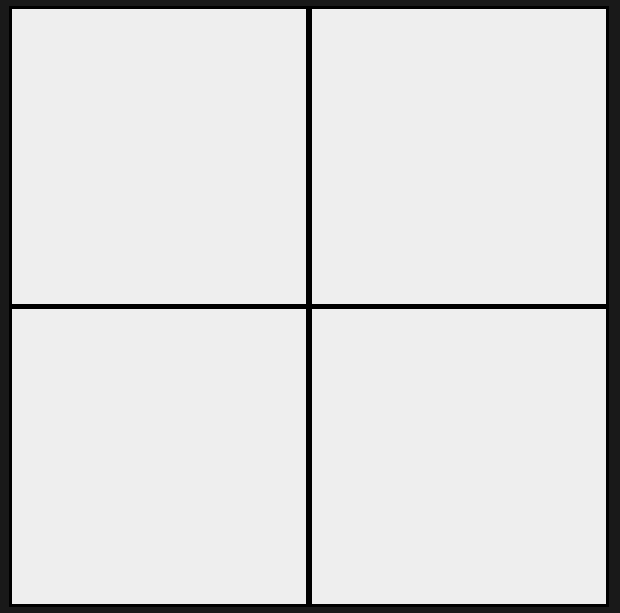
Array of Structs - uncoalesced



[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

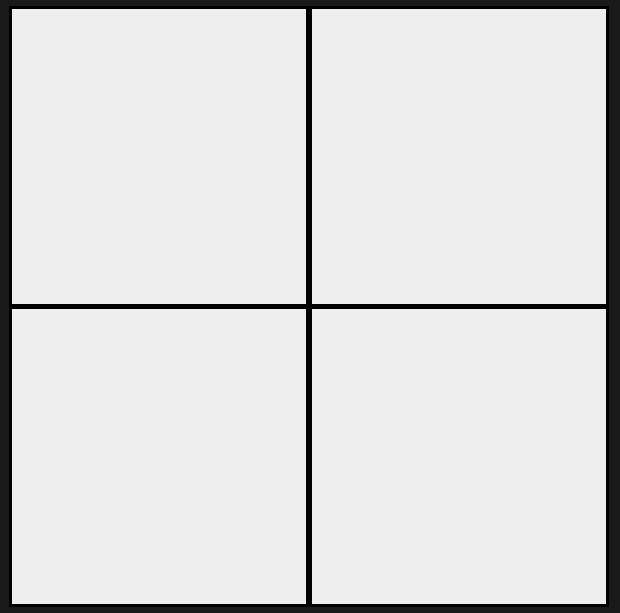
Array of Structs - uncoalesced

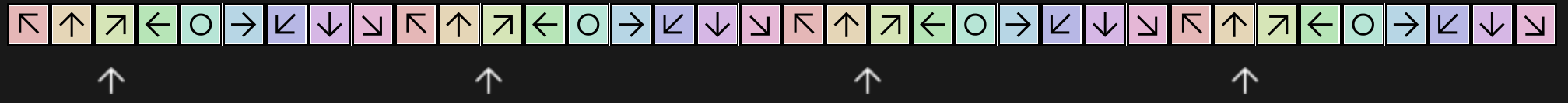


[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

Array of Structs - uncoalesced

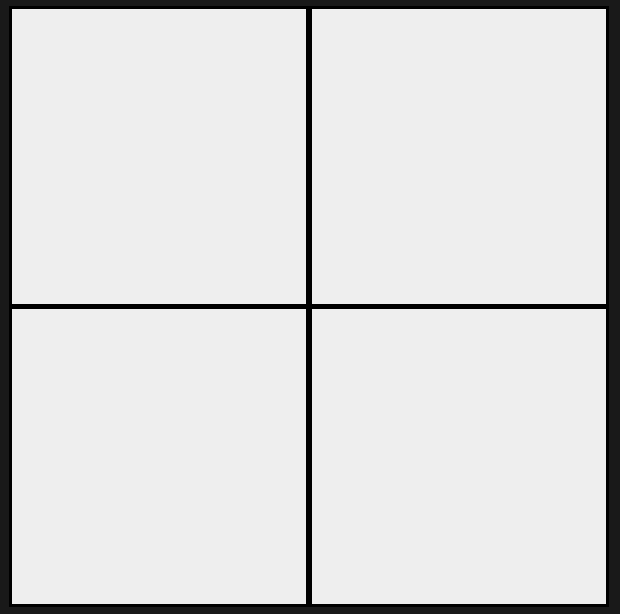


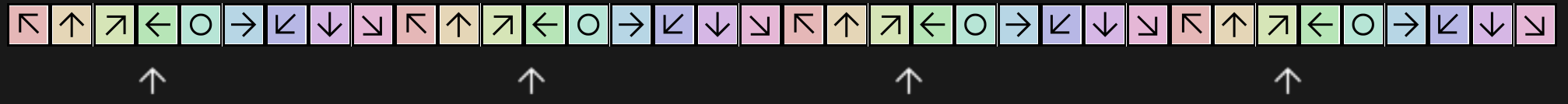


[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

Array of Structs - uncoalesced

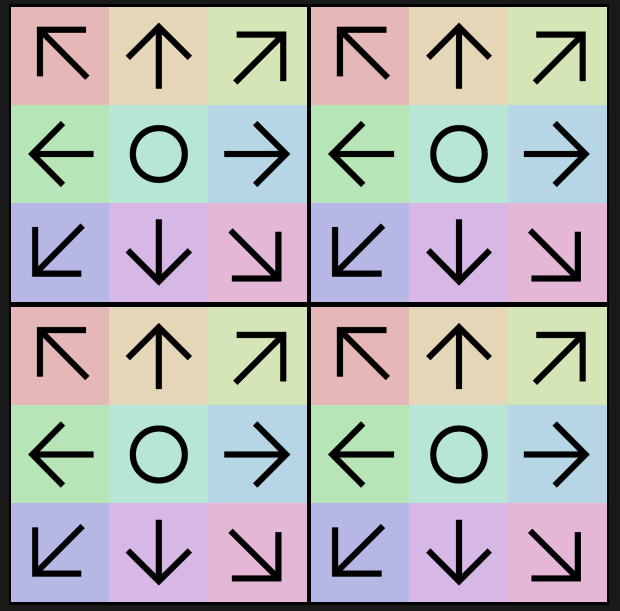




[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

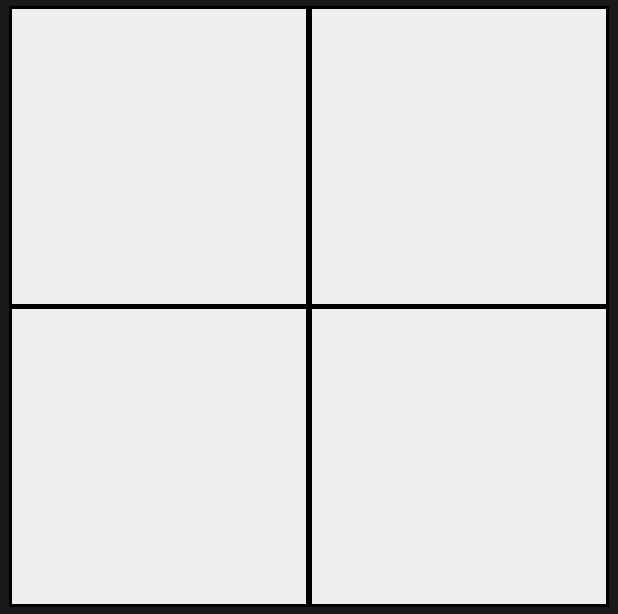
Struct of Arrays - coalesced

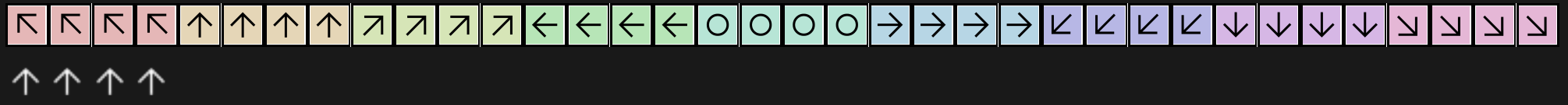


[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

Struct of Arrays - coalesced

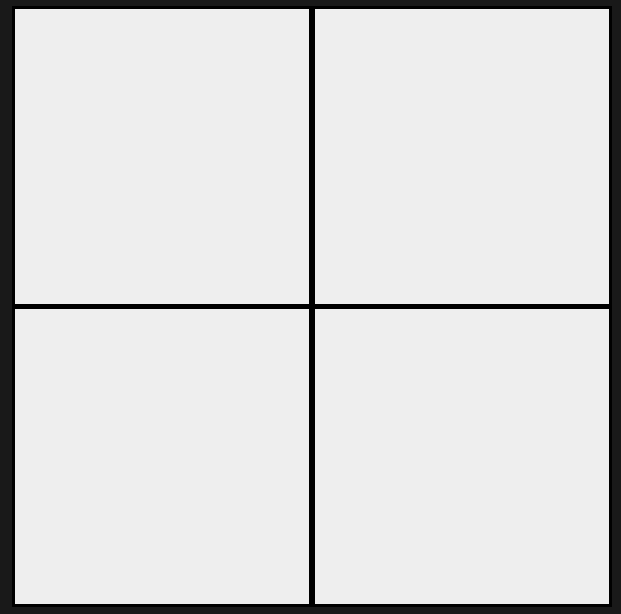


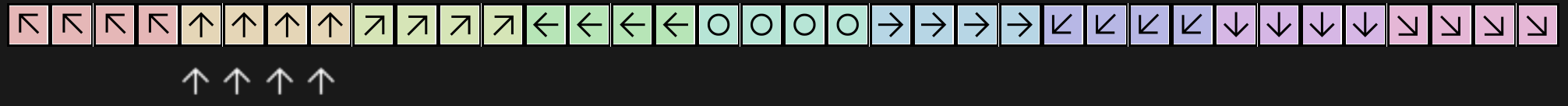


[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

Struct of Arrays - coalesced

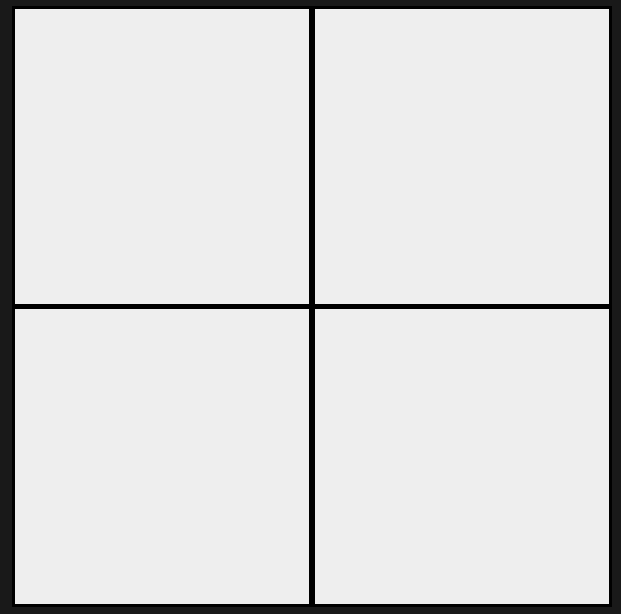


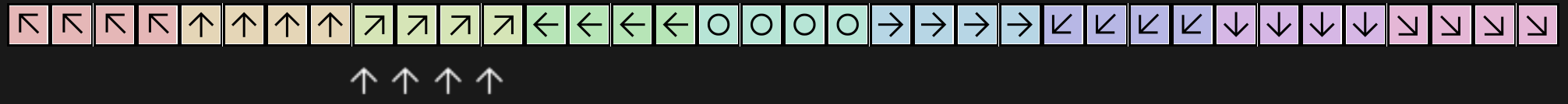


[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

Struct of Arrays - coalesced

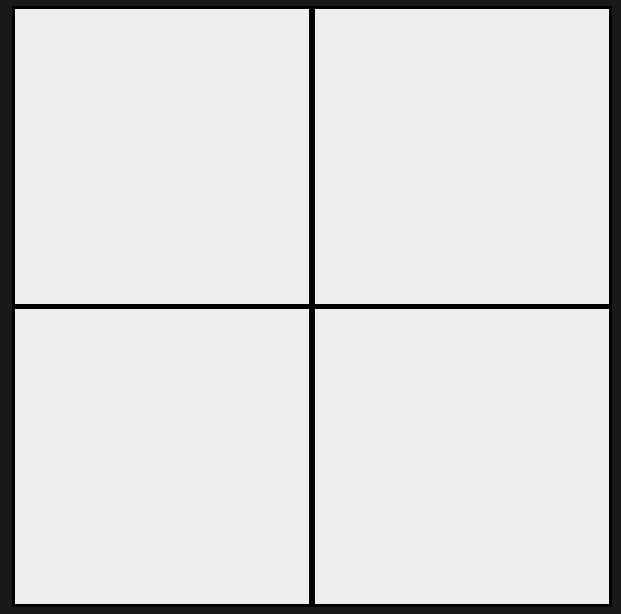


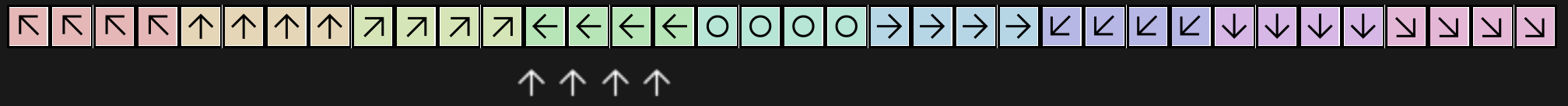


[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Memory Layout

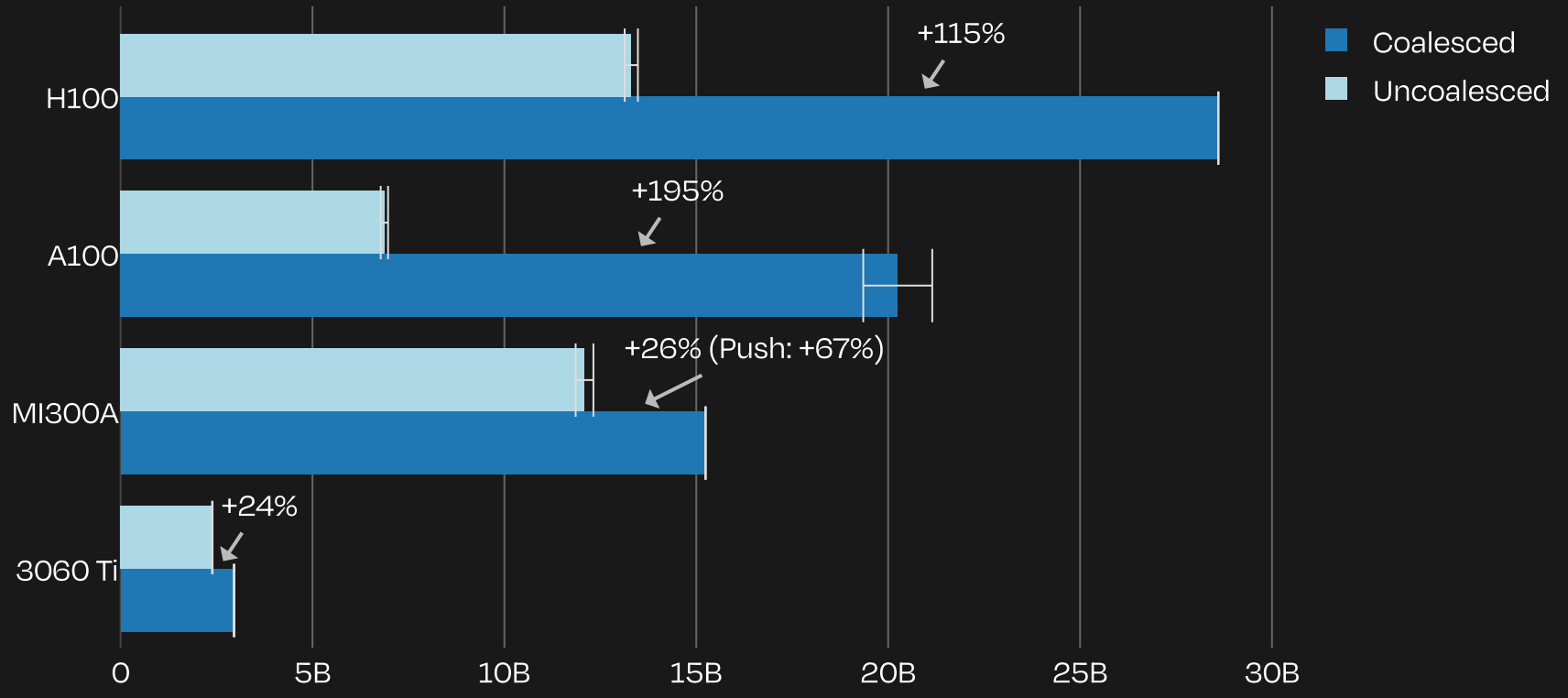
Struct of Arrays - coalesced





[Navarro-Hinojosa et al. 2018, Chapter 3.2]

Results - Coalescing Accesses

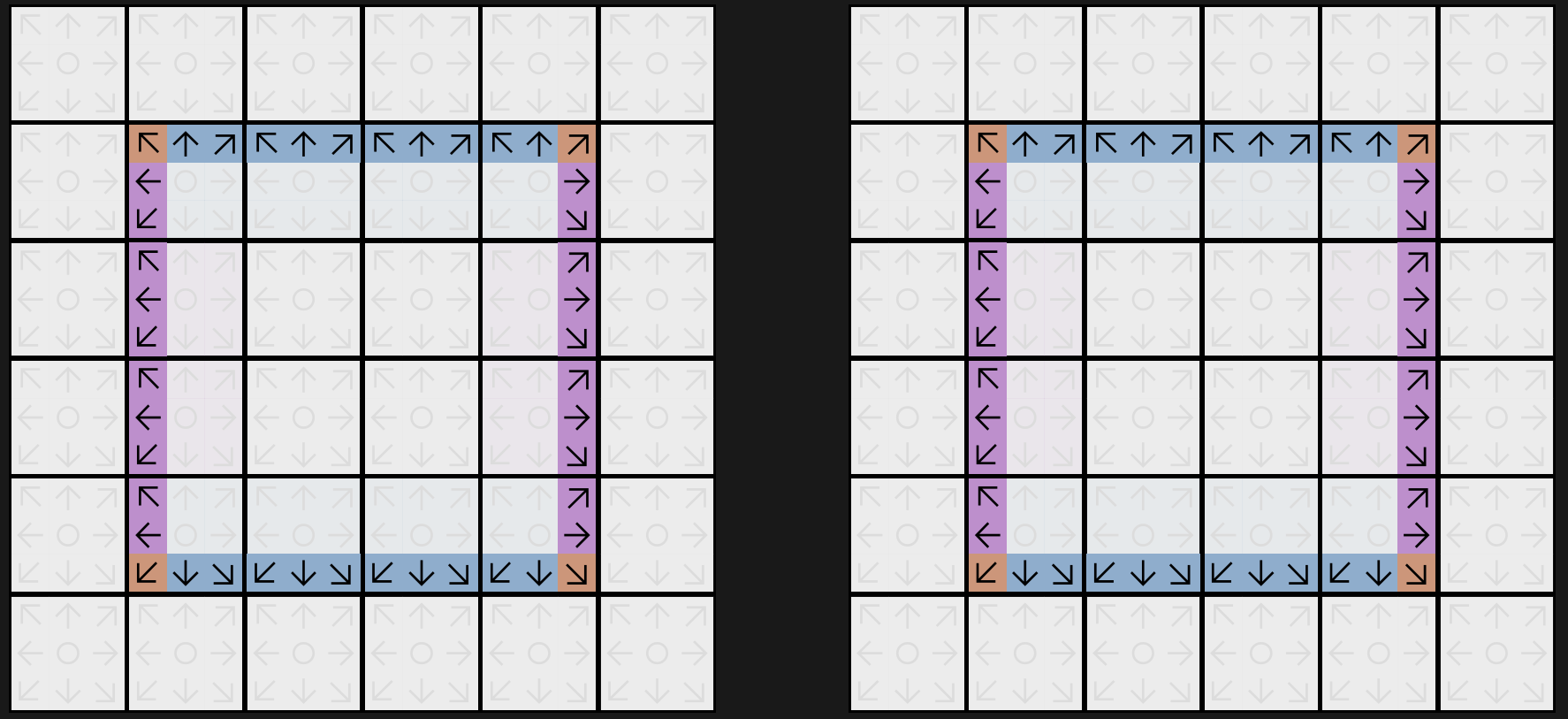


Lattice Updates per Second *[Math Processing Error]*

Pull scheme, A100 and H100 (32000 × 32000), 3060Ti (3000 × 3000), 100 steps, ≥ 5 repeats, D2Q9 Shearwave Decay

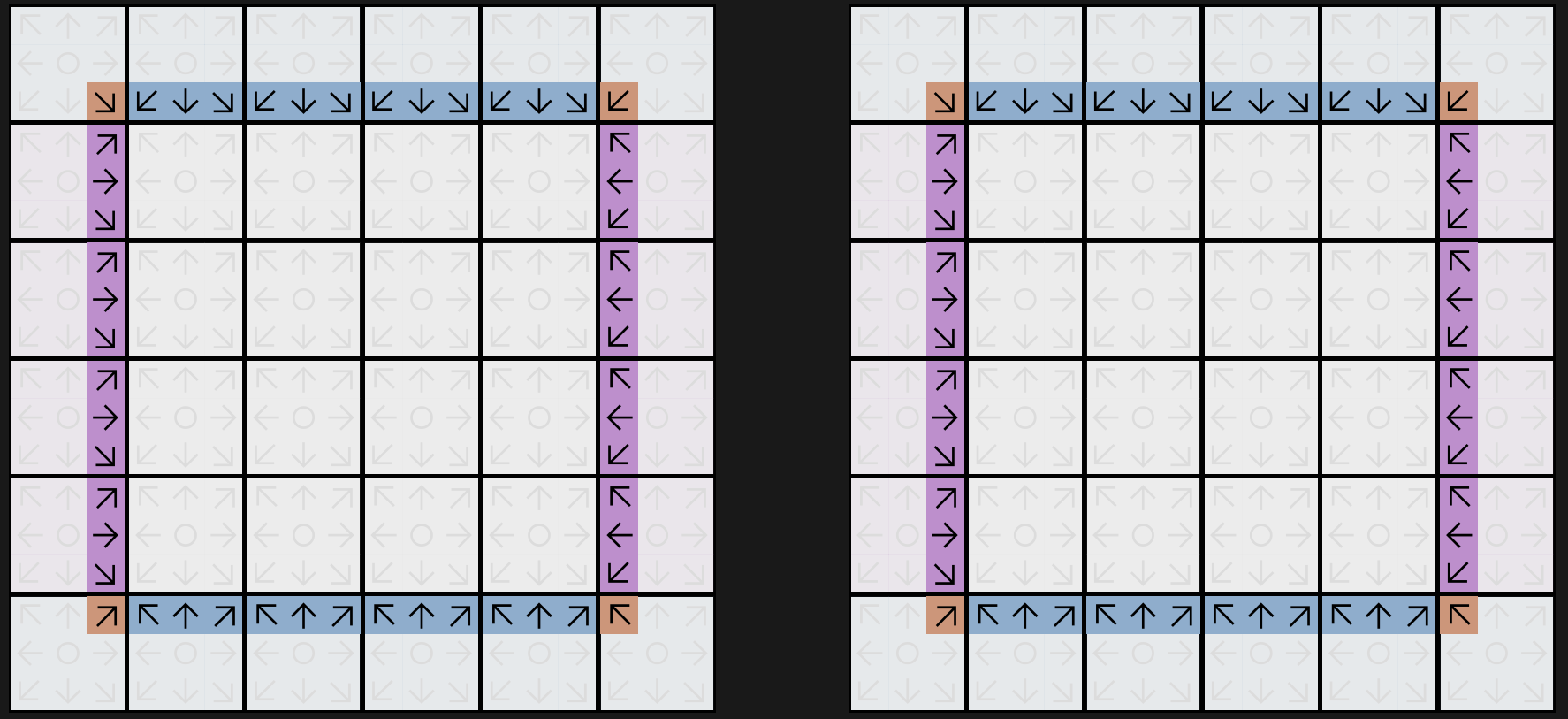
4. Work while communicating

MPI Communication



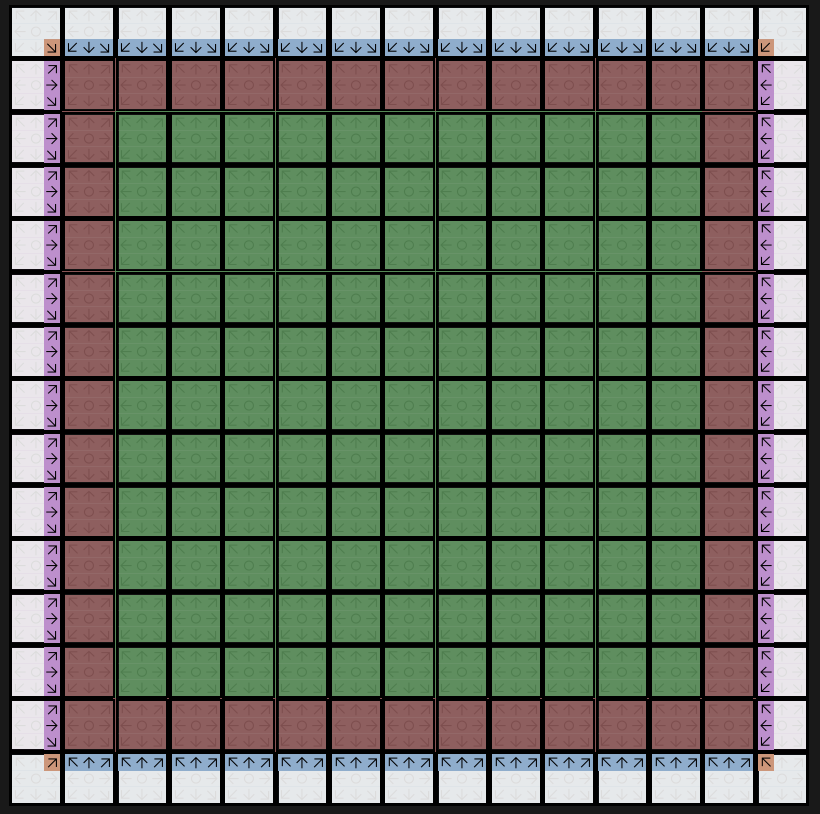
Communicate each border and corner asynchronously

MPI Communication



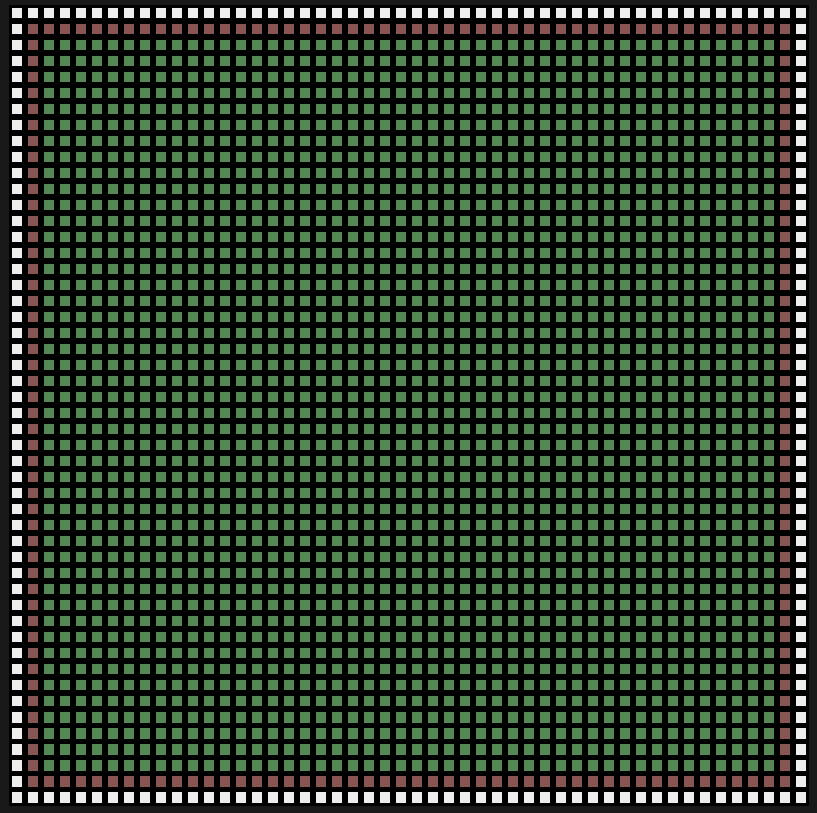
Communicate each border and corner asynchronously

MPI Communication



$\mathcal{O}(N)$ halo nodes $\ll \mathcal{O}(N^2)$ inner nodes

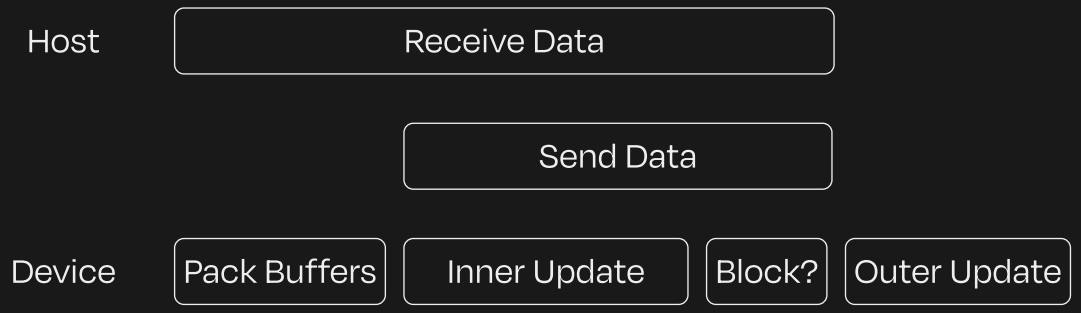
MPI Communication



$\mathcal{O}(N)$ halo nodes $\ll \mathcal{O}(N^2)$ inner nodes

MPI Communication

1. Post all `IRecv`
2. Pack Buffers and `ISend` them
3. Work on inner nodes during data transfer!
4. Block only if transfer not yet completed
5. Work on outer nodes[1]



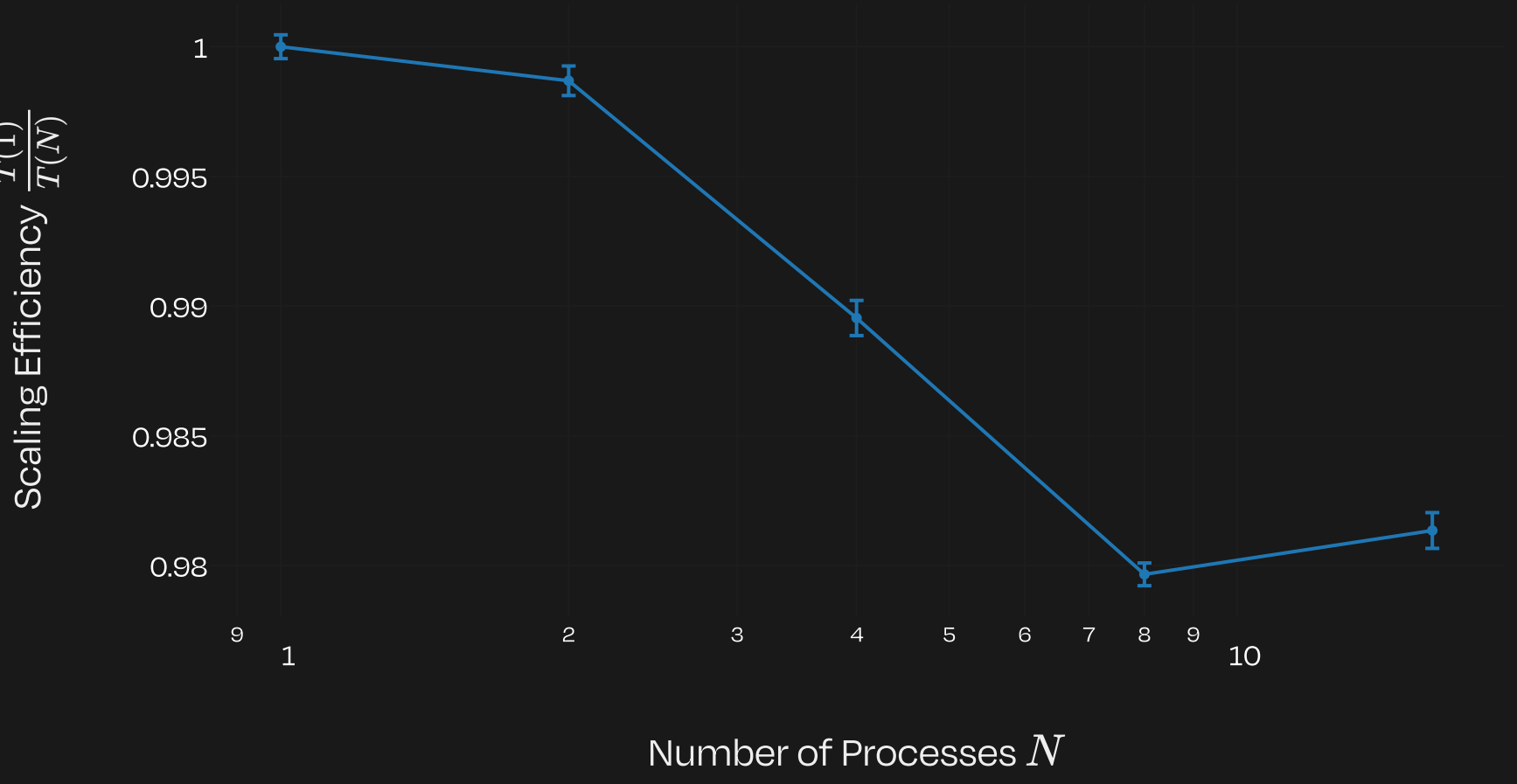
1. The Kokkos Lectures Module 6, Sandia National Laboratories, 2020

MPI Communication

1. Post all `IRecv`
2. Pack Buffers and `ISend` them
3. Work on inner nodes during data transfer!
4. Block only if transfer not yet completed
5. Work on outer nodes[1]

1. The Kokkos Lectures Module 6, Sandia National Laboratories, 2020

Results - Weak Scaling



Nvidia A100 GPUs, 32000 × 32000 nodes per process, 100 steps, 5 repeats, D2Q9 Shearwave Decay

Thank you for your attention?

Bonus: Taichi

Taichi Features

- Embedded in Python
- Portable, fast, statically typed
- Layout- and Dimension-independent Code
- GUI, LAS, Solvers etc. included
- Differentiable
- Only $\sim 15\%$ performance penalty[1]

1. 10^4 steps, 1000^2 nodes of D2Q9-PBC on 3060Ti compared to peak of Kokkos (2503 MLUPS vs. 2956 MLUPS)

Full example

```python
# Imports
import taichi as ti
import taichi.math as tm

# Start Taichi
ti.init(arch=ti.gpu)

# Define constants
NY = 800
NX = 800
Q = 9
OMEGA = 1.7
RHO = 1.
U_0 = 0.1
w = [4/9, 1/9, 1/9, 1/9, 1/9, 1/36, 1/36, 1/36, 1/36]
cx = [0,1,0,-1,0,1,-1,-1,1]
cy = [0,0,1,0,-1,1,1,-1,-1]
refl = [0,3,4,1,2,7,8,5,6]

# Define fields
buf = ti.field(dtype=ti.f32, shape=(NX,NY,Q,))
pixels = ti.Vector.field(n=3, dtype=ti.f32, shape=(NX,NY))
```

$\leq$ 50 Lines of Code, including GUI

Full example

Thank you for your attention!