Mantle Convection on GPU Using the Rayleigh-Benard Paradigm

Contributors

David A. Sanchez

MN Supercomputing Inst.,
University of Minnesota,
Minneapolis MN, USA

Dr. Dave Yuen

MN Supercomputing Inst.,
University of Minnesota,
Minneapolis MN, USA

Chris Gonzalez

Department of Geology,
University of Minnesota,
Minneapolis MN, USA

Dr. Grady Wright

Dept. of Math,
Boise State University,
Boise Idaho, USA

Greg A. Barnett

Dept. of Applied Math,
University of CO, Boulder
Boulder CO, USA

Introduction

Although considered over-hyped by many scientists, the price, performance, and ubiquity of GPU have contributed to its status as the rising star in the computational world. However, the generality of this performance is doubted by many and must be verified. Also notably, the areas in which prejudice would typically assert the dominance of GPU must be thoroughly investigated. To that end we implement an example from GPU’s own playground– linear algebra.

Preview

- We implemented 2D and 3D Rayleigh-Benard convection on GPU by utilizing a 2nd-order finite difference method.
- On a single Tesla C2070:
  - In single precision, we hit 535 GFLOP/s for 2D
  - In single precision, we hit 100 GFLOP/s for 3D

Outline

- **Model**
- Implementation
- Performance
- Analysis

Model: Assumptions

- We fix for all time $T_{top} = 0$, $T_{bottom} = 1$
- We take the limit of the Prandtl # to infinity
  - Ignore effects of thermal diffusion
- We assume the Boussinesq approximation
  - Ignore density variations, except bouyancy
- Freeslip sides
  - Velocities normal to sides are zero at sides
- Uniform, Cartesian geometry
- Fluid has constant viscosity

Model: Variables

- Rayleigh Number (Ra)
  - Describes how heat is transported within the fluid
  - Higher Ra promotes more vigorous convection
  - Such convection requires greater numerical accuracy
  - 2D max tested $6x10^{10}$
  - 3D max tested 10^7

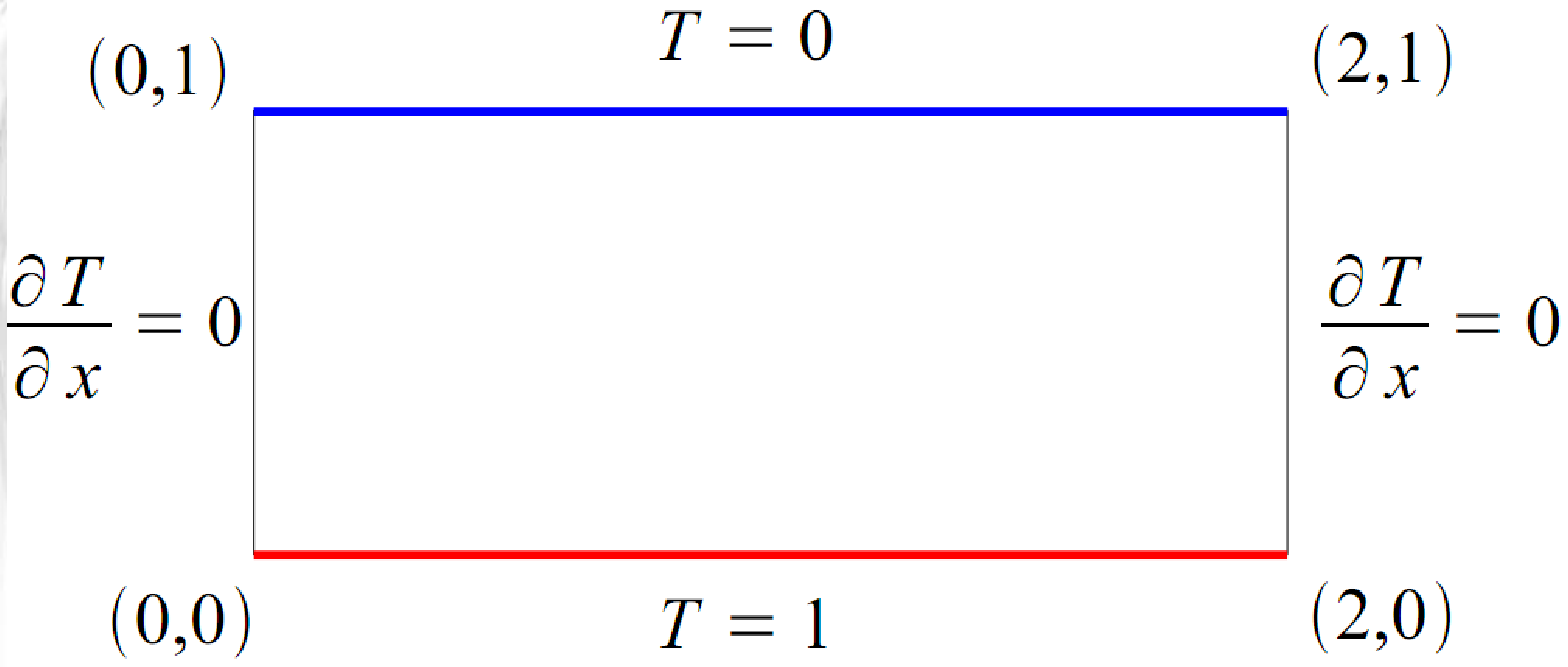
Model: Variables

- Aspect ratio
  - 2D tested: 1:1, 1:2, 1:3
  - 3D tested: 1:1:1, 1:2:2, 1:3:3
  - Greatly influences convective behavior
  - For fixed Ra, varying aspect ratio can throw the system from equilibrium into chaos

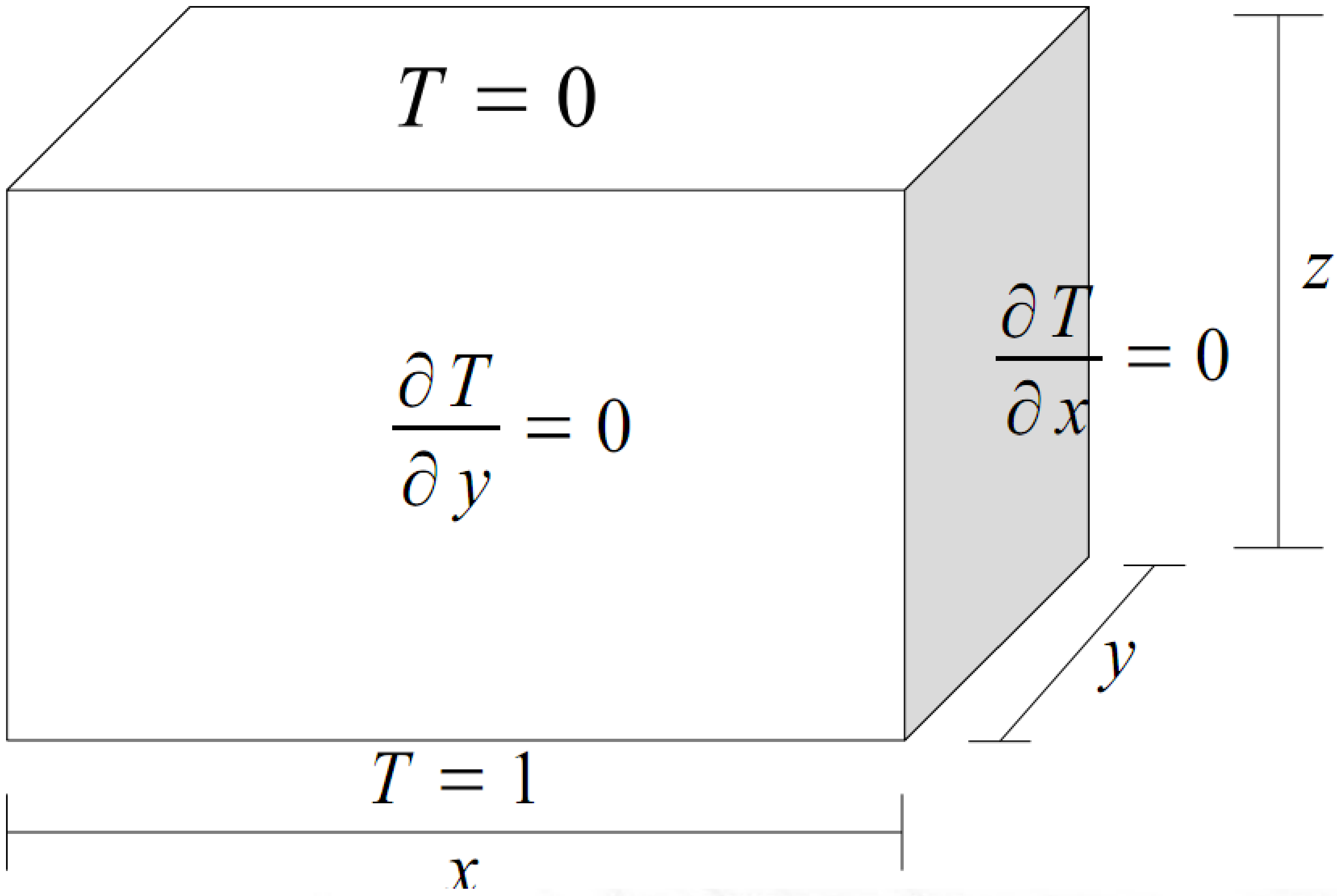
Model: Variables

- Precision
  - Single
  - Double
- This is usually held fixed for a given model, but we briefly examine effects due to change of precision
  - Some of these effects are difficult to predict a priori

Model: 2D Pictorial Model


$T = 0$
$(0,1)$
$(2,1)$
$\frac{\partial T}{\partial x} = 0$
$\frac{\partial T}{\partial x} = 0$
$(0,0)$
$T = 1$
$(2,0)$

Model: 3D Pictorial Model


$T = 0$
$\frac{\partial T}{\partial y} = 0$
$\frac{\partial T}{\partial x} = 0$
$T = 1$
x
y
z

Model: Application

- Generically, Rayleigh-Benard convection is a simple model describing the behavior of a basally heated fluid
- This model has applications to fluid mechanics
  - This is an often-encountered system
    - Plenty of benchmarks
    - Plenty of prior work
- Not quite a passable model for the contemporary Earth’s mantle, especially at high Ra
  - More suitable for magma oceans

Model: Equations

- Energy: $\frac{\partial T}{\partial t} = \nabla^2 T - \vec{v} \cdot \nabla T$

- Momentum: $$\nabla^2 \omega = \mathrm{Ra}\, T$$ $$\nabla^2 \psi = \omega$$

- Stated in this way, velocity looks like:

$$u = \frac{\partial^2 \psi}{\partial x\, \partial z},\ v = \frac{\partial^2 \psi}{\partial y\, \partial z},\ w = -\frac{\partial^2 \psi}{\partial x^2} - \frac{\partial^2 \psi}{\partial y^2}$$

Model: Numerical

- Domain spatially discretized via method of lines
  - Using a 2nd order finite-difference scheme
- Timestepping is with Runge-Kutta 3
  - Allows for much-needed variable timestepping
    - Domain-wide, convective velocity is non-uniform
    - Duration-wide, convective velocity is non-uniform
  - Additionally, must solve momentum equations 3x per timestep

Model: Architectural Concerns

- Classically, parallel machines are “good” at different types of problems than conventional ones
- CPU implementations might perform miserably on an exact problem, but may be able to perform better on a lower-FLOP algorithm which progresses more rapidly
- Generically, machine-to-machine benchmarks must take on qualitative properties if these concerns are to be addressed (though the victor is often still obvious)

Model: Architectural Concerns II

- Despite what was said in the previous slide, exposing underlying architectural differences is much easier beginning from congruent algorithms
- The search for better algorithms is motivated by understanding what is wrong with implemented ones, so these differences are important to identify
  - We don’t always do these searches when we should—they might be a lot of work

Model: Solution

- The next slide is of a computational flowchart for the 3D solution, although the 2D solution follows closely
- 1 and 2 in the next slide are the coupled Poisson equations we will discuss shortly, once these are solved, velocity can be found easily and used to update the array

Model: Computational Flowchart

$$\begin{tikzcd}
 & \boxed{T^{k+1} = T^k + \frac{\Delta t}{4}(q_1 + 3q_3)} \arrow[dl] & \\
\boxed{q_1 = g(T^k)} \arrow[r] \arrow[dr, shift right] & \boxed{q_2 = g(T^k + \frac{\Delta t}{3} q_1)} \arrow[r] \arrow[d, shift right] & \boxed{q_3 = g(T^k + \frac{2\Delta t}{3} q_2)} \arrow[ul] \arrow[dl, shift right] \\
 & \text{function } y = g(S) \arrow[ul, shift right] \arrow[u, shift right] \arrow[ur, shift right] &
\end{tikzcd}$$

```
function $y = g(S)$
1. Solve $\nabla^2 \omega = \mathrm{Ra}\ S$.
2. Solve $\nabla^2 \psi = \omega$.
3. Compute $\vec{u} = (u, v, w)$:
    $u = \frac{\partial^2 \psi}{\partial x \partial z}, v = \frac{\partial^2 \psi}{\partial y \partial z}, w = -\frac{\partial^2 \psi}{\partial x^2} - \frac{\partial^2 \psi}{\partial y^2}$.
4. Compute $y = \nabla^2 S - \vec{u} \cdot \nabla S$.
end
```

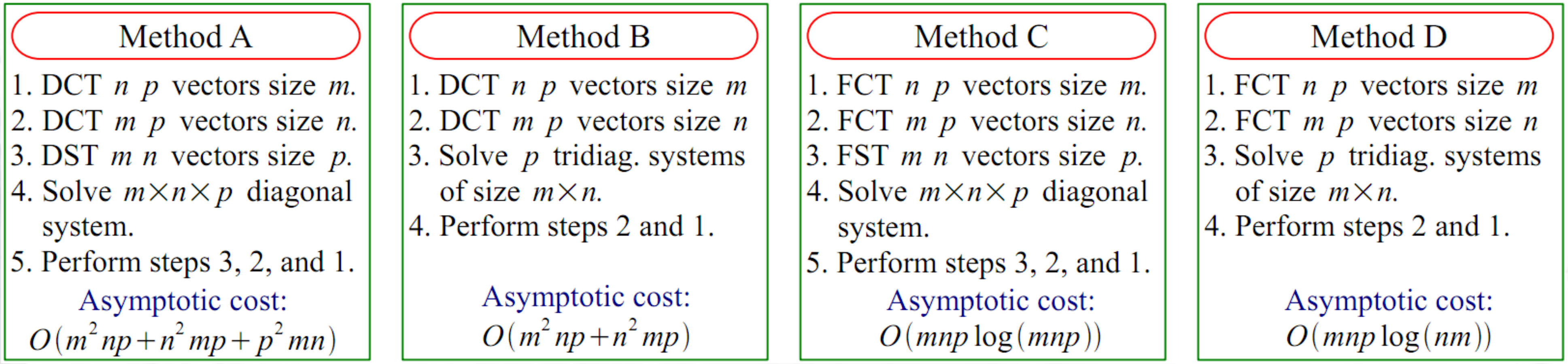
Outline

- Model
- **Implementation**
- Performance
- Analysis

Implementation: Poisson Eqns

- We have coupled Poisson equations, which require a suitable solution
- Generically, this should be done with something like a DFT (often an FFT-based method)
- cuFFT would work, however…

Implementation



Method A

1. DCT n p vectors size m.
2. DCT m p vectors size n.
3. DST m n vectors size p.
4. Solve $m \times n \times p$ diagonal system.
5. Perform steps 3, 2, and 1.

Asymptotic cost: $O(m^2 np + n^2 mp + p^2 mn)$

Method B

1. DCT n p vectors size m
2. DCT m p vectors size n
3. Solve p tridiag. systems of size $m \times n$.
4. Perform steps 2 and 1.

Asymptotic cost: $O(m^2 np + n^2 mp)$

Method C

1. FCT n p vectors size m.
2. FCT m p vectors size n.
3. FST m n vectors size p.
4. Solve $m \times n \times p$ diagonal system.
5. Perform steps 3, 2, and 1.

Asymptotic cost: $O(mnp \log(mnp))$

Method D

1. FCT n p vectors size m
2. FCT m p vectors size n
3. Solve p tridiag. systems of size $m \times n$.
4. Perform steps 2 and 1.

Asymptotic cost: $O(mnp \log(nm))$

- We tested methods for solving
  - Within our scale, A is fastest—not FFT-based
  - Computationally, swap FFT for GEMM
- FCT and FST above are based on FFT
- Sensitive results need constant supervision:
  - Is this faster in CUDA 3.2 with BLAS speedup?
  - Faster heterogeneously (CPU+GPU with MAGMA)?

Implementation

- To do everything in the flowchart from before, we need only a small repertoire of operations
  - Scalar-matrix multiplication
  - Matrix-matrix addition
  - These matrices come from the xy, xz, and yz sheets of the 3D array
  - For the 2D problem, change “matrix” to “vector” above
  - Finally, we need matrix-matrix multiply

Implementation

- Computationally, the various sheets in the array look like strided vectors of some variety, so the only BLAS routines we need are:
  - axpy ($\mathbf{y}=\alpha\mathbf{x}+\mathbf{y}$)
  - scal ($\mathbf{x}= \alpha\mathbf{x}$)
  - gemm ($C = \alpha AB + \beta C$)
- CUBLAS could accommodate this need
  - So could MAGMA, which uses CUBLAS and a CPU-BLAS for CPU-GPU computing (not discussed in this talk)

Implementation: GPU concerns

- It’s easy to get lazy and forget that manycore systems don’t scale monotonically with problem size!
- This is especially easy to do when utilizing high-level abstractions, such as BLAS packages
- These “gotchas” may significantly impact performance

Implementation: example

- As a part of the finite-difference stencil, we iterate through the xy-sheets A_i of the array. We want to make a new array whose interior xy-sheets $B_i = A_{i-1} + A_{i+1}$
- It’s easy to just say “for all i, $B_i = A_{i-1} + A_{i+1}$”
- But if we think of A_i as a vector $\mathbf{v}_i$ with a given stride k AND the concatenation $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ also has stride k, we gain from doing the entire operation at once, on the concatenation of vectors, rather than iteratively through the list of vectors

Implementation: Considerations

- Because global memory transactions are so expensive, it’s economical to compute certain quantities every time they are needed
- For example, our GEMM takes a temperature array and a fixed array as input. We could improve performance by determining the elements of this array on every timestep instead of pulling it from memory

Implementation: Caveats

- The GPU software landscape is changing rapidly
  - Accordingly, benchmark landscape changes too
    - MAGMA 1.0 was released on Dec 9th
    - CUDA 3.2 improved our runtime by a factor of two
      - On Fermi
- This is just a part of life at the cutting edge
- After a few years hopefully either,
  - Better support (providing this is our job)
  - New hardware to work with

Outline

- Model
- Implementation
- **Performance**
- Analysis

Performance: 2D Single Precision

- All results Tesla C2070
  - For the problems it could accommodate, faster 2D performance on the GTX 480—tiny global memory heavily restricted the domain size
- Max 535 GFLOP/s for full code
  - Computing, iterating, and saving
  - Close to SGEMM limit on C2070
- Min 0.14 us/timestep×gridpoint

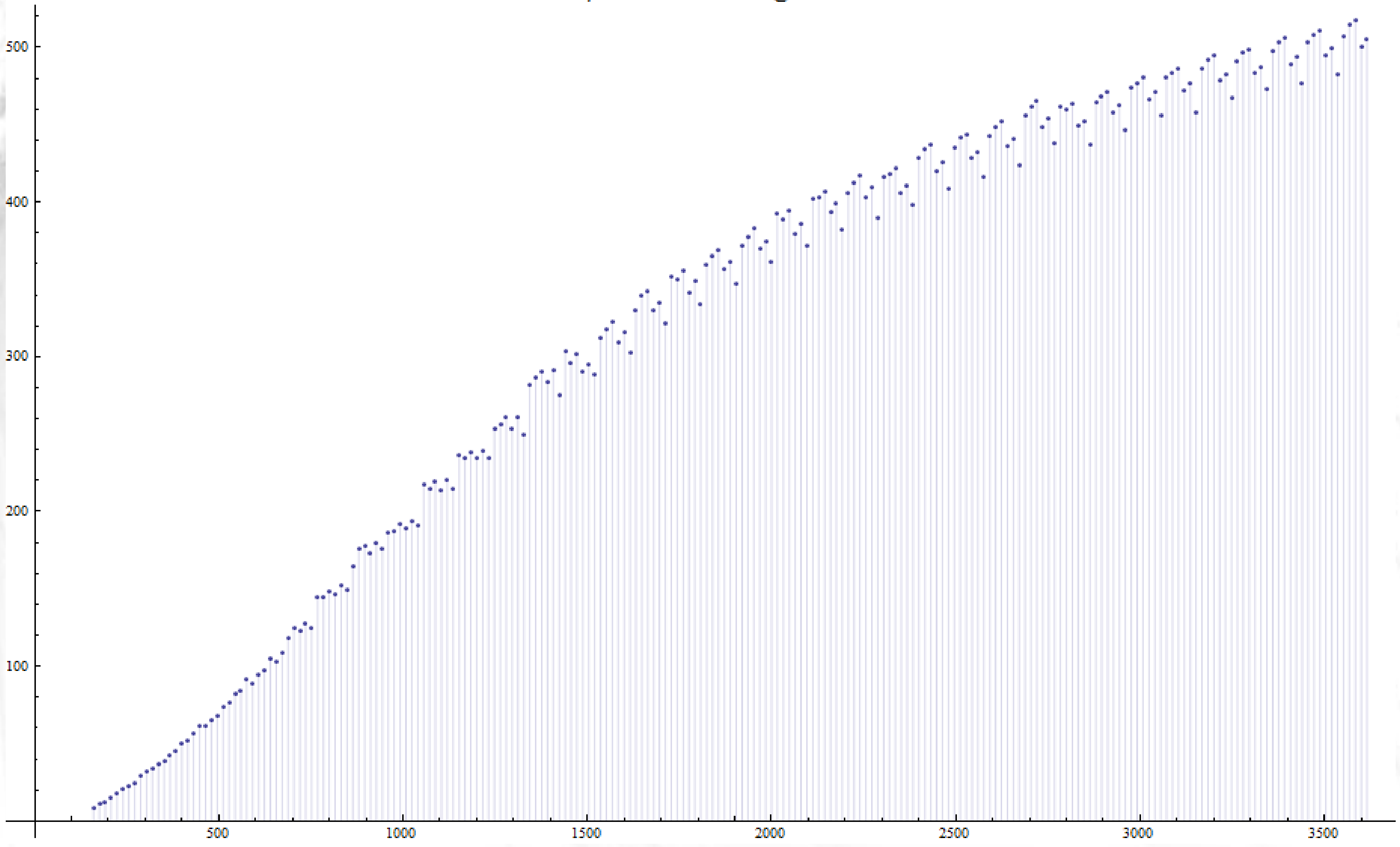
Performance: 2D Double

- Max 270 GFLOP/s for full code
- Given that only smaller problems can be tackled in double precision, performance slightly better than 50% single precision’s
  - Due to more efficient memory pipelining

Performance: 2D Remarks

- Predictably, square matrices out-performed non-square matrices with comparable number of elements
- Non-monotonic performance profile exposes scaling irregularities. This is due to GPU processor use during GEMM—on non-optimal matrix dimensions, many computing agents are completely idle
- The next slide showcases this visually—check out the regularity of peaks and valleys!

GFLOP/s vs side length 2D 1:1



Performance: 3D Single Precision

- Reaches 100 GFLOP/s for 1:1:1
- Reaches 85 GFLOP/s for 1:3:3
- Why is this so low? The matrix-matrix multiplies are maximally 450x450 matrices, which does not allow sufficient saturation of GPU computing elements
- How does the aspect ratio interact? By increasing the size of the matrices, but also significantly increasing the proportion of less-efficient, non-GEMM operations

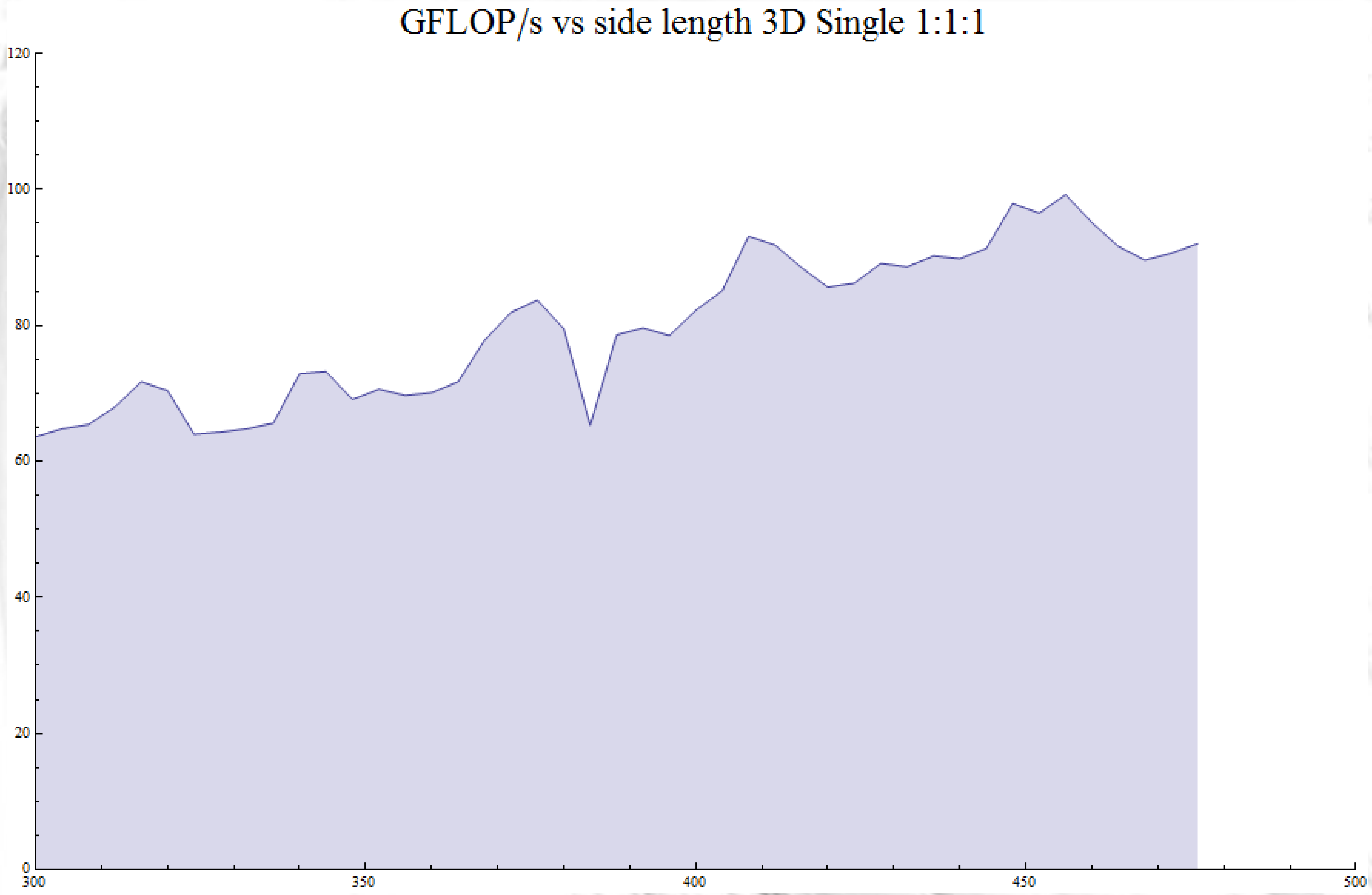
Performance: 3D remarks

- Although dominated by matrix-matrix multiplies (as is 2D) performance is much worse. Why is that?
- Each GEMM is on much, much smaller matrices—for single precision, 450x450 vs 2000x2000. When the 2D matrices are also taken down to this scale, the performance is comparable.
- Excellent candidate for upscaling to 360 GPU

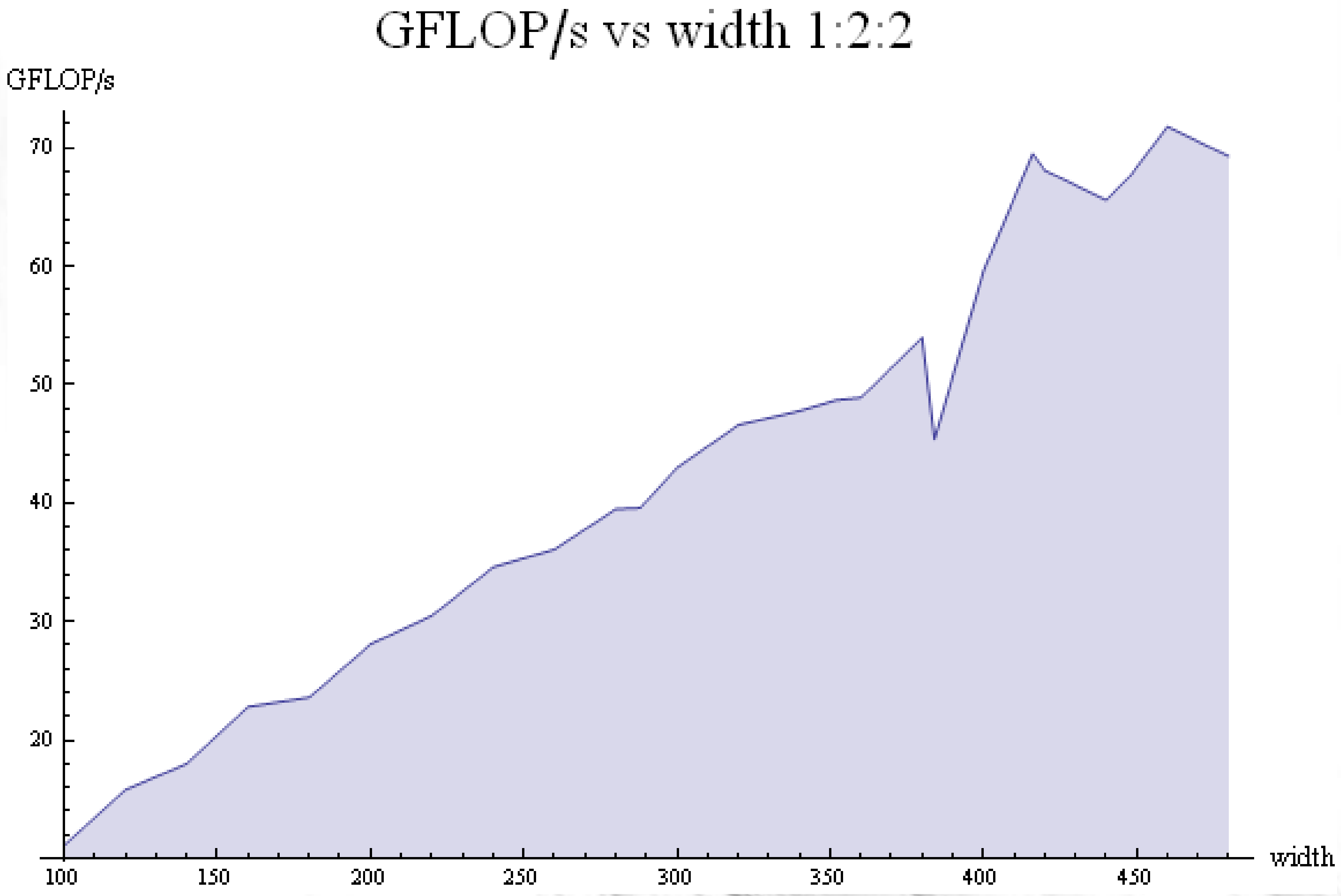
Performance: Scaling

- Predictably, 3D doesn't scale the same as 2D code by side length.
- Double-precision code compares more favorably to single-precision code for these runs
- Even in 3D, when there is more computational noise per timestep, properly saturating the GPU can contribute markedly to performance
- The following are two graphs of 1:1:1 and 1:2:2 GFLOP/s performance, then a third graph of wall-clock time per timestep*gridpoint

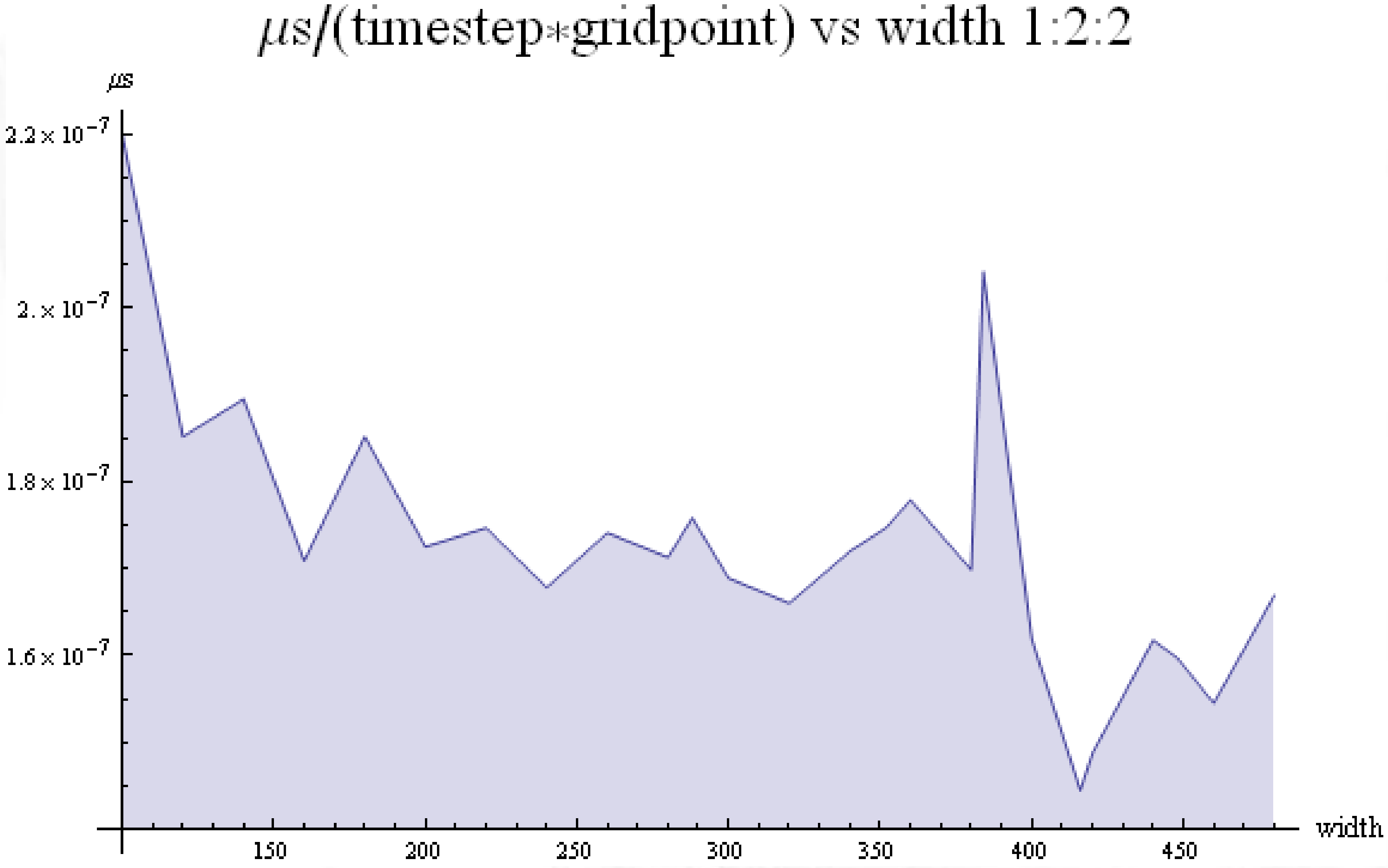
Performance: Scaling SP



Performance: Scaling DP



Performance: Scaling DP



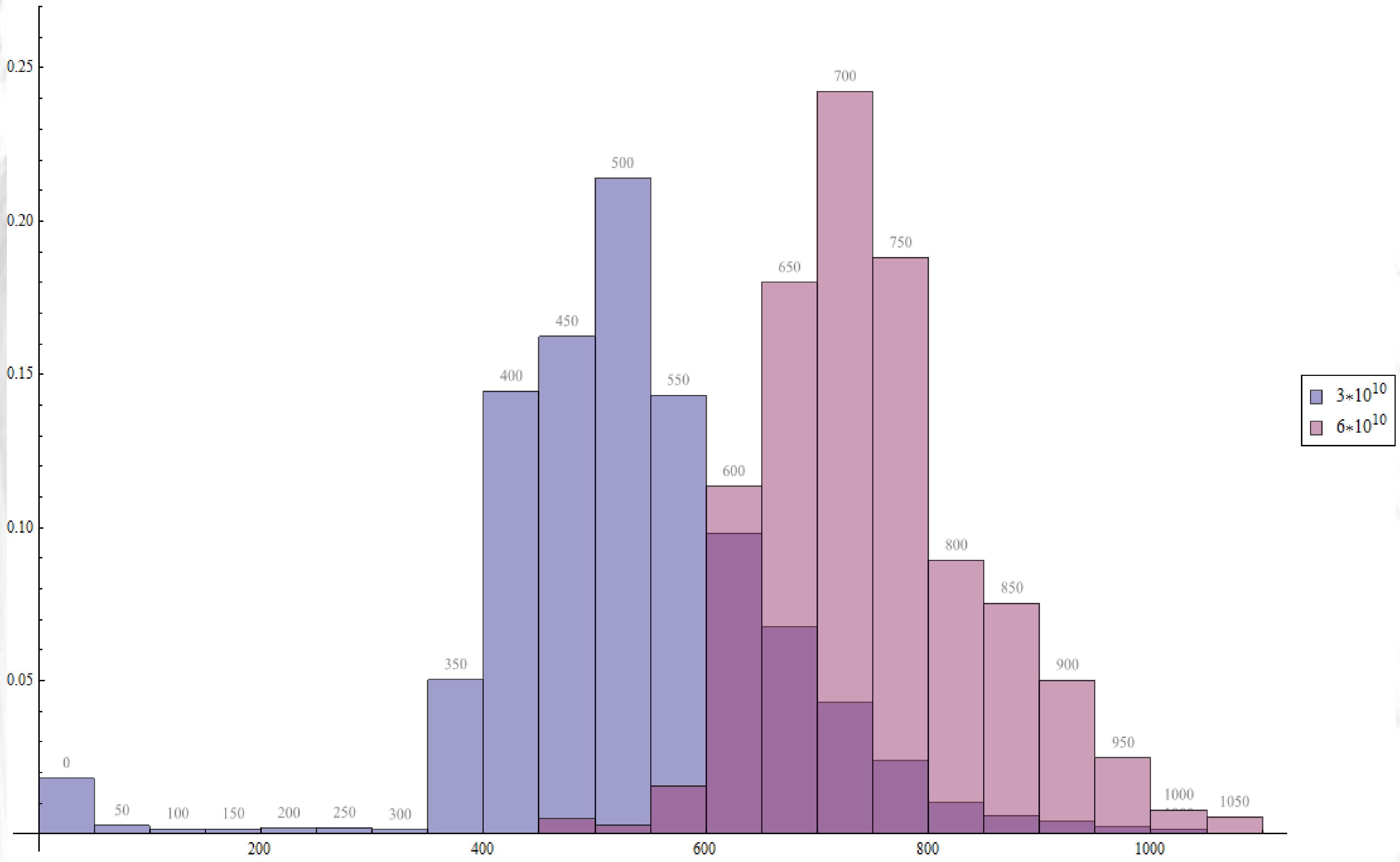
Outline

- Model
- Implementation
- Performance
- **Analysis**

Analysis: 2D

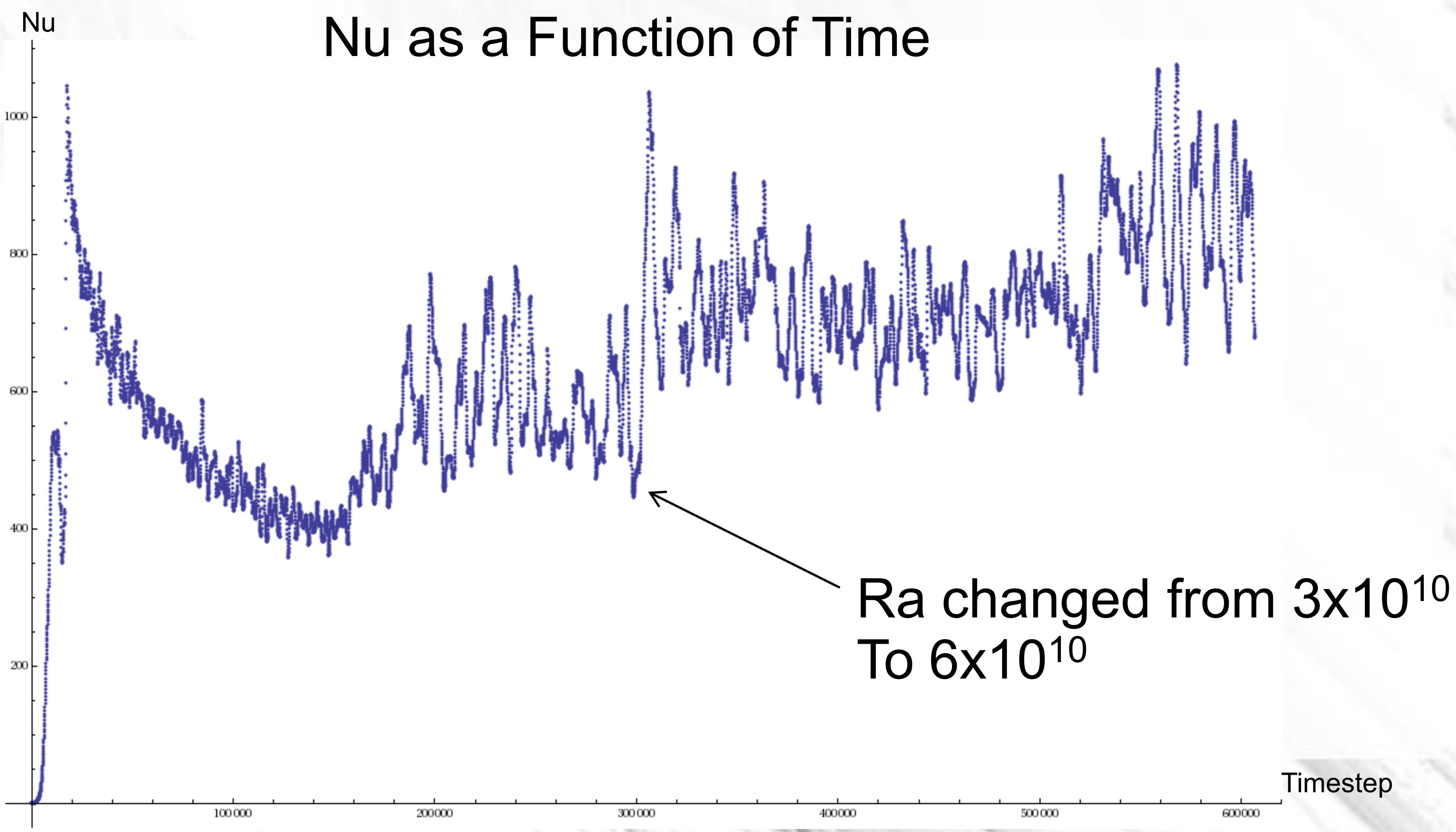
- Flow-reversal at high Ra
  - Characterization and enumeration of these reversals is a necessary next-step in our analysis
- Appropriate relationship with the Nusselt and Rayleigh numbers
- Increased Rayleigh number relates to a general increase in Nusselt number, indicating more turbulent flow

Nusselt Number



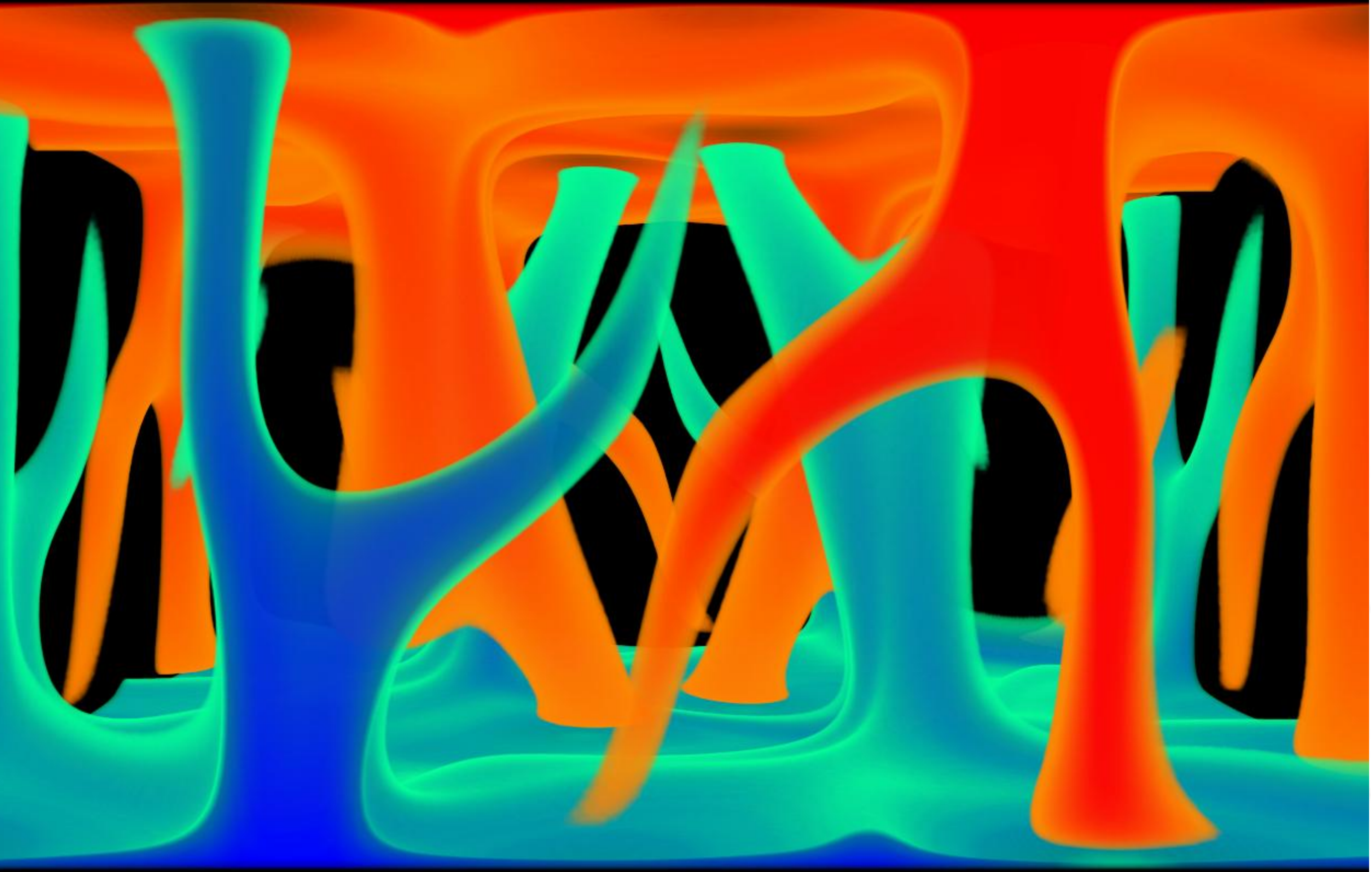
Analysis: 2D

- Also predictably, perturbing Ra mid-run raises the mean value of the Nusselt number



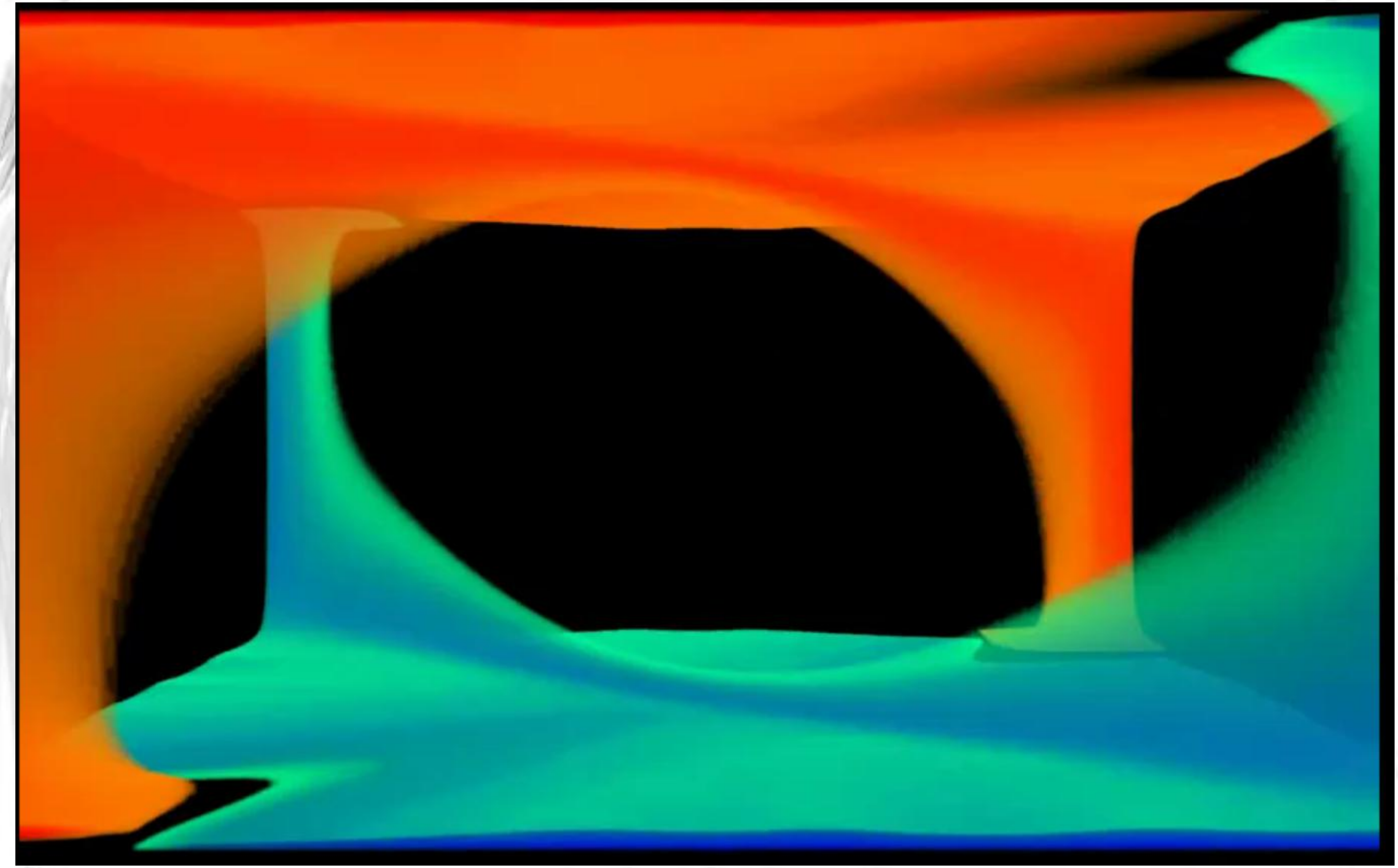
Analysis: 3D

- Agreement in Nu with Zhong(2005)
- Quasi-stable at Ra=10^6, 10^7
- Following is a picture from early on in the model’s evolution



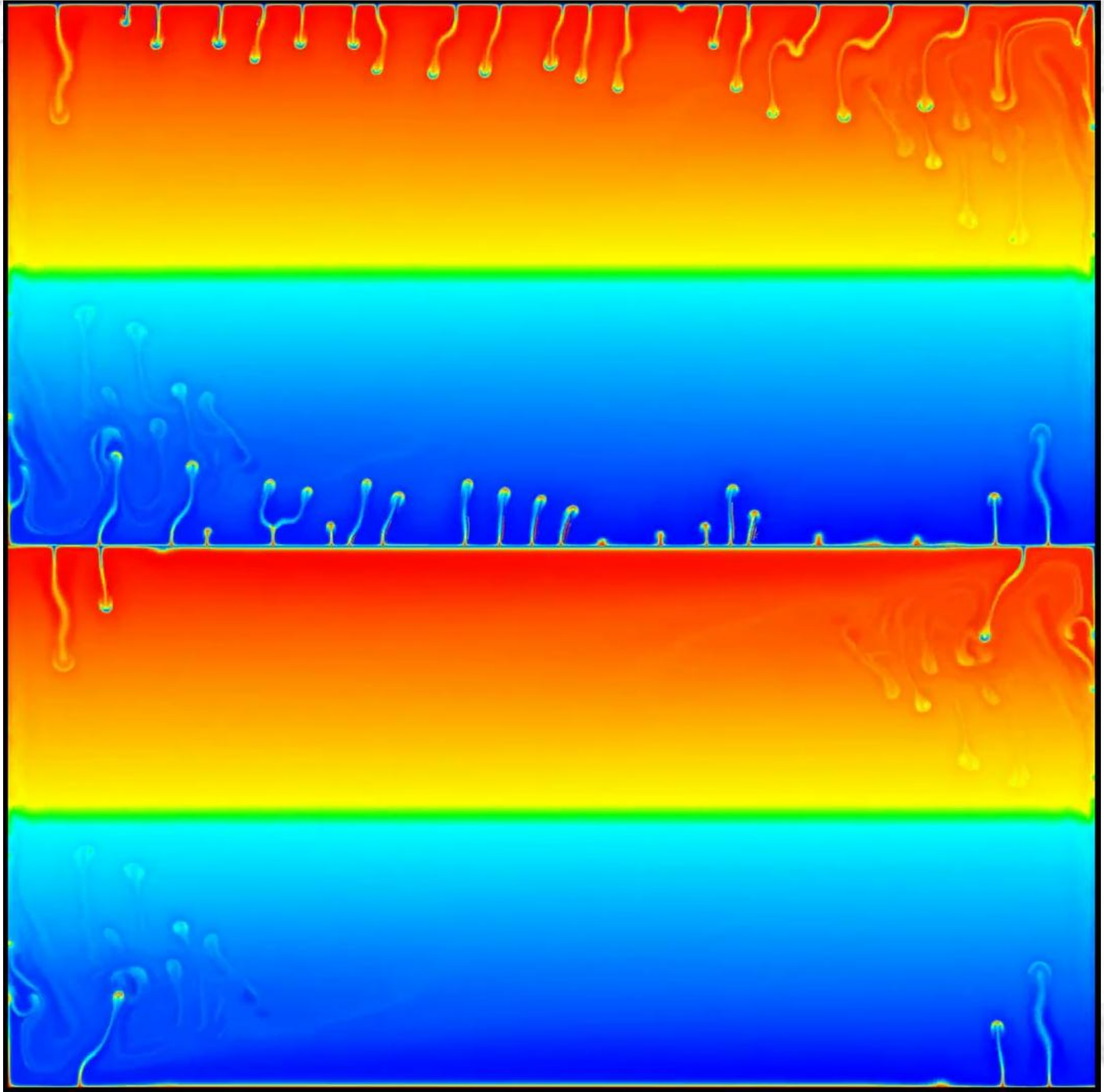
Analysis

- 3D Ra=10^6,10^7 runs quickly achieve a quasi-stability, as shown in the next slide
- This can be overcome by utilizing more GPU



Performance: 2D SPvsDP

- We’ve also investigated divergence of trajectories in both single- and double-precision 2D runs. The following image is at a low timestep, at which point the trajectories have obviously diverged.
  - Non-dimensional times at this point agree—this divergence is not due to timestepping
  - Single-precision is on the top half of the slide
- Both runs at Ra=10^9



Analysis: 2D SPvsDP

- Despite this, a surprising amount of agreement in various structures that arise
  - Number of plumes
  - Mean temperature
  - Max x- and z- speed
  - To check: number of flow-reversals

Analysis: Future

- With 360 GPU, we can do 3000x3000x1500
  - High Ra, more efficient GEMM, faster speed
- 3 GPU to explore 3D flow-reversal
  - Wavelet analysis on vorticity field
  - Visualize more fields (velocity, vorticity, etc)
- Implement on CPU with ATLAS, MKL, or other BLAS

Thank you

Many videos are available at

www.youtube.com/sanc0174

Please send comments and questions to

sanc0174@umn.edu