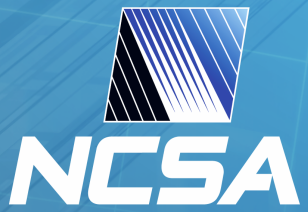
Doubling the Whammy: Relating Parallel Particle Advection State-of-the-Art to In Situ Processing

Rob Sisneros, Dave Pugmire



National Center for Supercomputing Applications
University of Illinois at Urbana–Champaign

This Talk

- Not in situ, motivated by in situ (classify *that*, Hank)
- In situ motivation
- Progress improving parallel particle advection

The Argument for In Situ

- Larger machines make larger datasets
- Data is *already* hard if not possible to move let alone store

- The only possibility for full scale analysis is to bypass the storage subsystem

- Operate on the data in situ


MISSION ACCOMPLISHED

… Kind of

Hank Doesn't Like In Situ

- How do we figure out what to do in situ before the simulation runs?
- What's the new cost for being wrong?
- Is there really enough memory for us to take some from the simulation?
- What about some proposed very low memory per core machines?
- After integrating with a simulation, how much of the HPC workflow should we be concerned with? Resiliency? Checkpointing?
- Wait – how the hell do even integrate with simulation codes?

Some opinions

Stated as fact

All things in situ represent the core of current HPC visualization research

- Does this mean we have addressed these issues? No
- What the hell is all the research then? Lots of ad hoc solutions
- Are we at least trying to answer these questions? Not really. (We've kind of begrudgingly agreed this is somehow the future and in the meantime continue to publish our ad hoc approaches)

But Wait, There's More

It gets worse

Broader View of HPC Visualization Research

- Summary: we want to enable visualization capabilities on HPC platforms through improved algorithms, etc.
- Moving even non-in situ visualization to HPC resources adds many of the difficulties of in situ deployment (or similar ones)

- Ways we improve performance make in situ deployment harder (or impossible)

Our favorite problem child

Particle Advection

- Particle advection is a foundational visualization algorithm
  - Used for streamlines, pathlines, streamsurfaces, pathsurfaces, poincare analysis, FTLE, etc.
- Difficult to efficiently parallelize due to load imbalance
  - Data decomposition
  - Vector field characteristics
  - Seeding strategies
  - Number of seeds
  - Computational resources

Improvements

- Parallelizing over data domains vs. over seeds
- Hybrid parallelism approaches
- Load balancing/rebalancing
  - Workflow estimations
  - Per round dynamic
- Data partitioning
  - Create optimal domain geometry
  - Static/dynamic

What works in situ?

NCSA

Improvements In Situ

- Parallelizing over data domains vs. over seeds
- Hybrid parallelism approaches
- Load balancing/rebalancing
  - Workflow estimations
  - Per round dynamic
- Data partitioning
  - Create optimal domain geometry
  - Static/dynamic

Improvements In Situ

- Parallelizing over data domains vs. over seeds
- Hybrid parallelism approaches
- Load balancing/rebalancing
  - Workflow estimations
  - Per round dynamic
- Data partitioning
  - Create optimal domain geometry
  - Static/dynamic

This Work

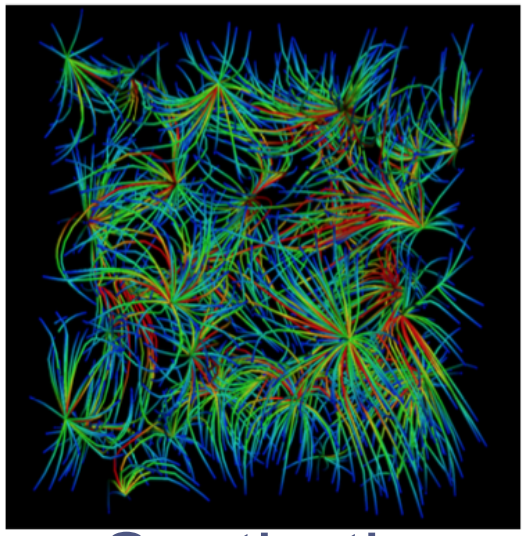
- Go back to (and stay) where improvements impact practical use: parallelize over data (POD) algorithm
- Poke around for opportunities through analyzing parameter sweep results

- Getting very practical
  - Improvements to particle advection are slow to make it into production
  - POD is the primary algorithm offered by large scale visualization suites and used at HPC centers

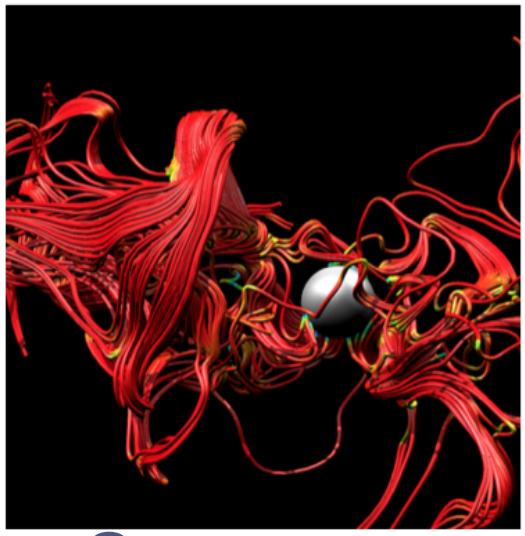
Parameter Sweep

- Advection Rounds
  - Number of advection steps: 5 values
  - Number of particles per round: 18 values
- Seeding
  - Number of seeds: 50K, 250K, 500K
  - Strategy: whole, medium, small
- Communication: synchronous, asynchronous
- Parallelism
  - Moderate: 64
  - Larger: 512, 1024
- Four datasets

Datasets

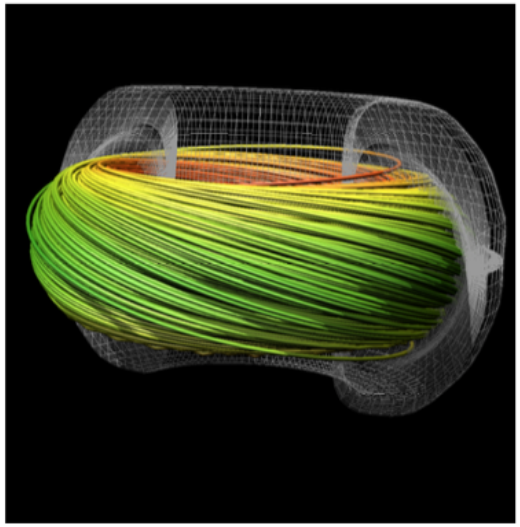


Synthetic

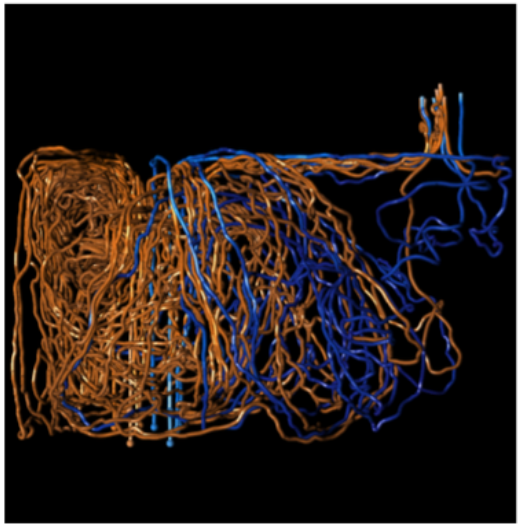


Supernova

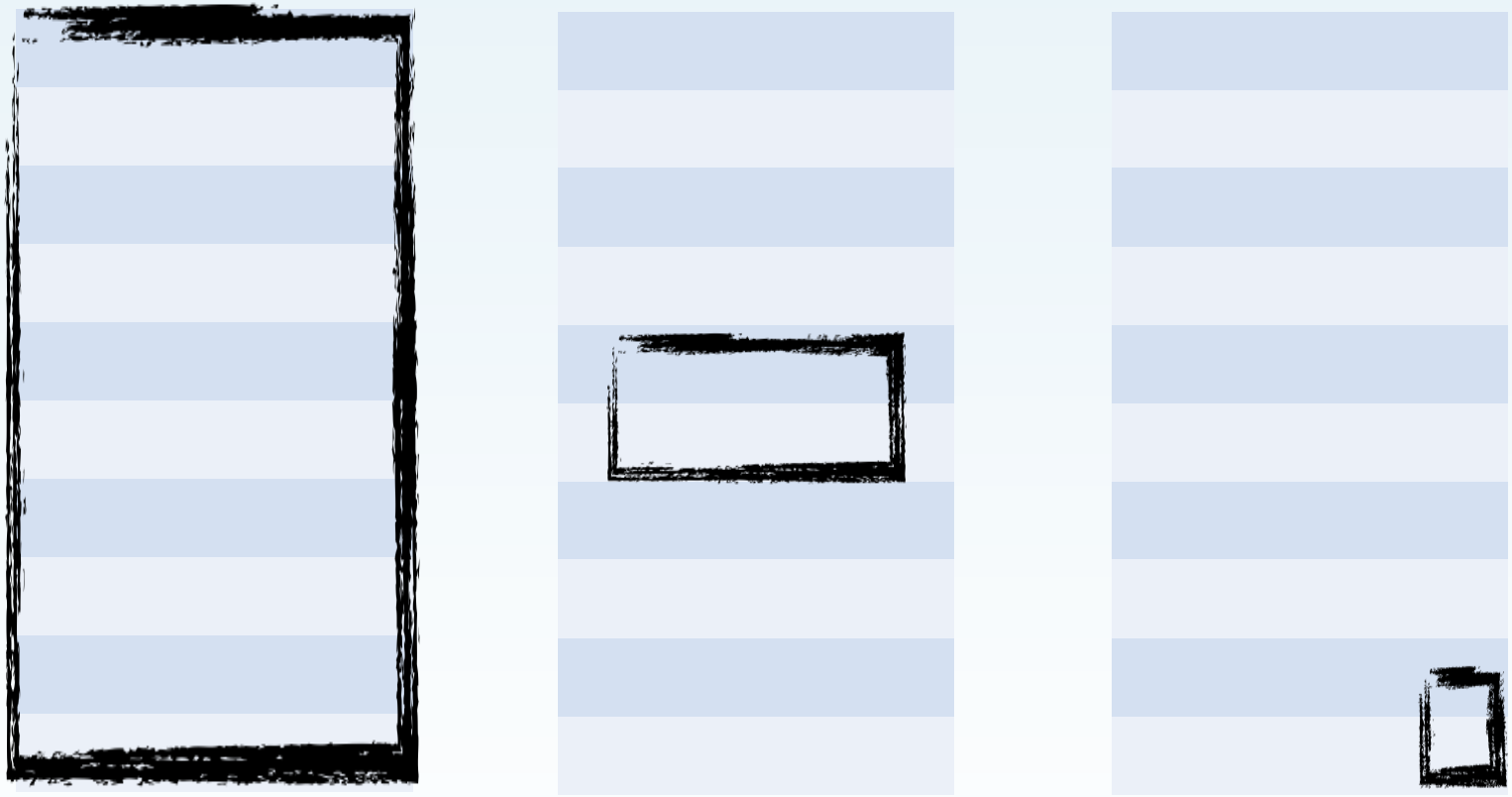
Fusion



Thermal



Seeding Strategies



Whole Medium ~20% Small ~ 5%

The Grand Total

- 4 datasets
- 3 seeding strategies
- 3 seeding levels
- 3 levels of concurrency
- 2 algorithms
- 90 parameter pairs

4*3*3*3*2*90 = 19,440 tests

NCSA

Experimental Setup

- Run on Rhea cluster at OLCF using VisIt
  - Rhea: 512 node, two 8-core Intel Xeon, 64 GB per node
- Used a production version of VisIt (2.9)
  - Minor modifications to POD algorithm to support additional knobs
  - Added more timers, counters, etc.
  - Ignored I/O and rendering time

NCSA

Early Results

Highlighting the Unhighlightable

- For “particles per round” the worst possible parameter is the DEFAULT

- For “advection steps” the default is not the worst possible, but is nothing to celebrate

Qualifying the Runs

- F(x,y) = Fastest time at configuration (x,y)
- T(x,y) = Time at configuration (x,y)
- Bin the times into 4 groups:

$$
\begin{array}{rrll}
\textit{Good}: & T_{(x,y)} \leq & 1.15 * F_{(x,y)} & \\
\textit{Decent}: & 1.15 * F_{(x,y)} \leq & T_{(x,y)} & < 1.5 * F_{(x,y)} \\
\textit{Poor}: & 1.5 * F_{(x,y)} \leq & T_{(x,y)} & < 2 * F_{(x,y)} \\
\textit{Abysmal}: & T_{(x,y)} > & 2 * F_{(x,y)} &
\end{array}
$$

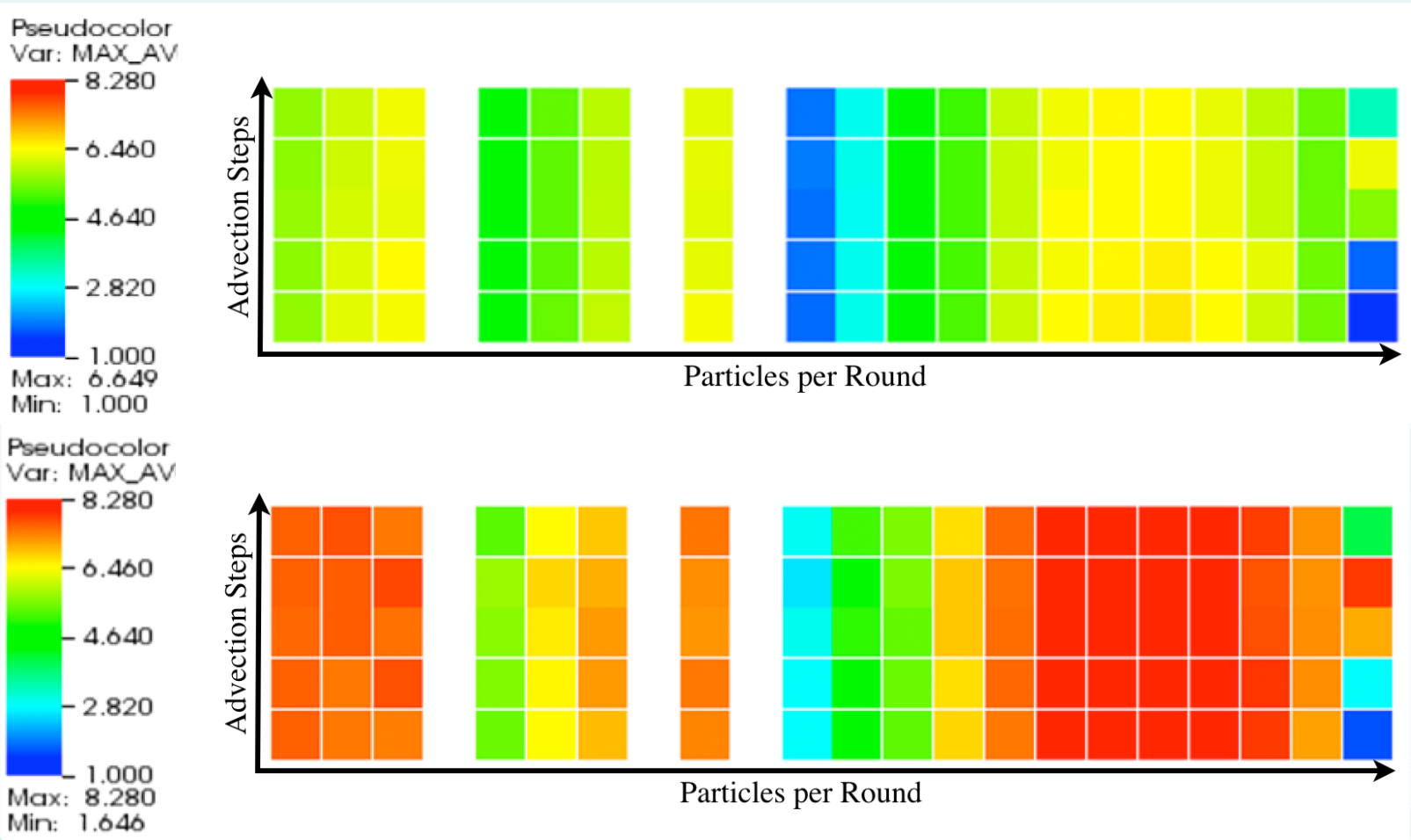
Default vs. Everything Else

	Good	Decent	Poor	Abysmal
Average	42.46%	41.80%	10.03%	5.72%
Default	33.96%	25.47%	20.75%	19.81%
Increase	8.5%	16.33%	10.72%	14.09%

Point #1

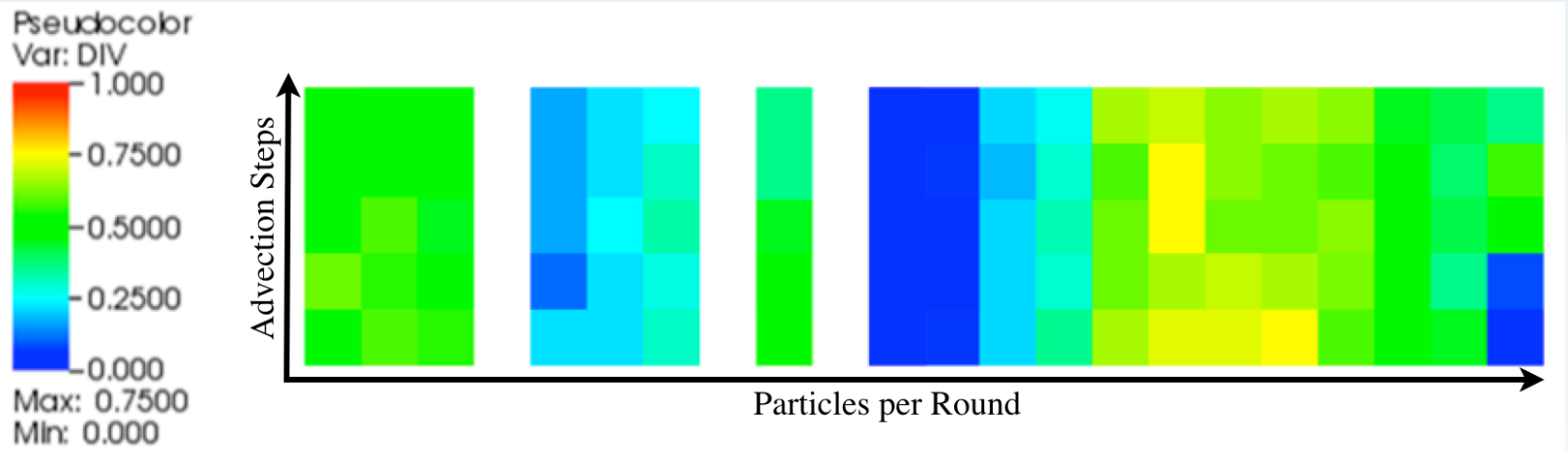
- Updating terrible default settings save HPC resources

No Brainer – Synchronous vs. Asynchronous

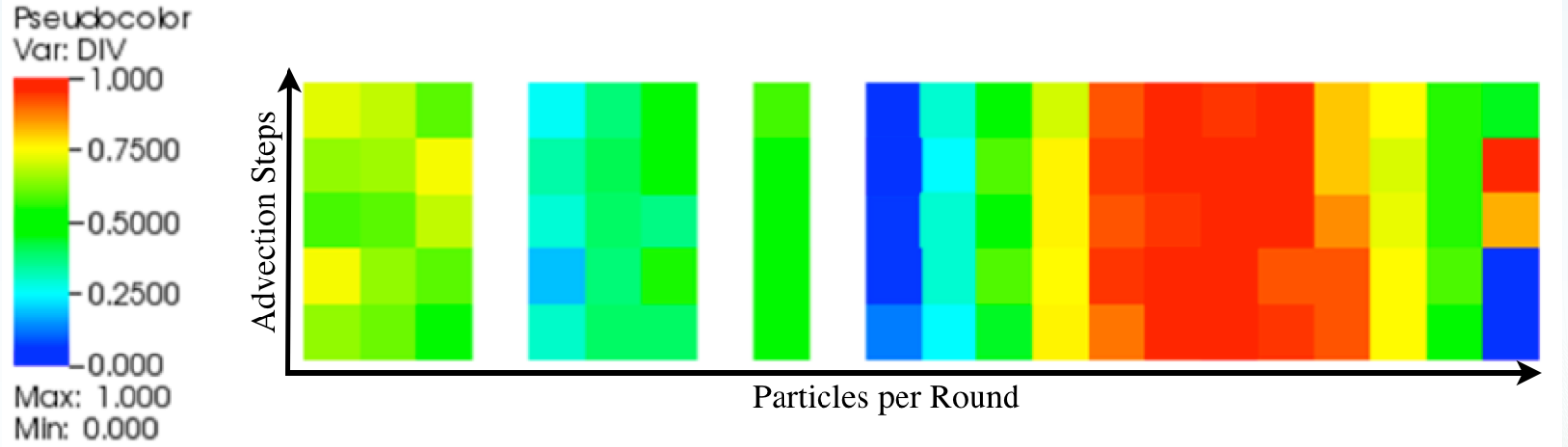


but

250K Seeds – Synchronous vs. Asynchronous

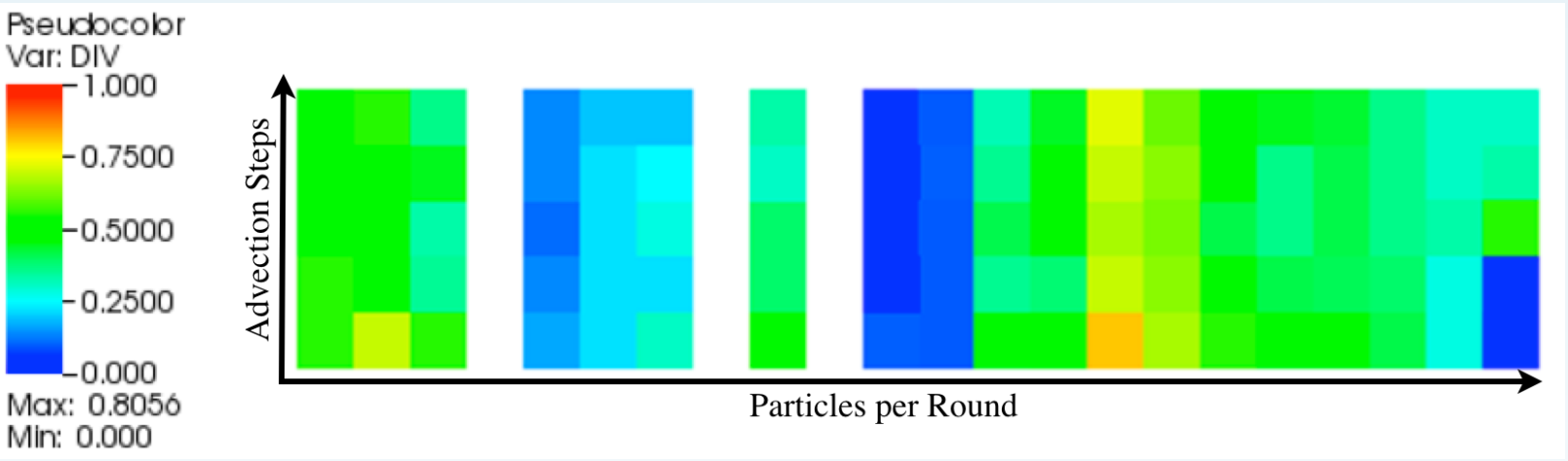


Sync

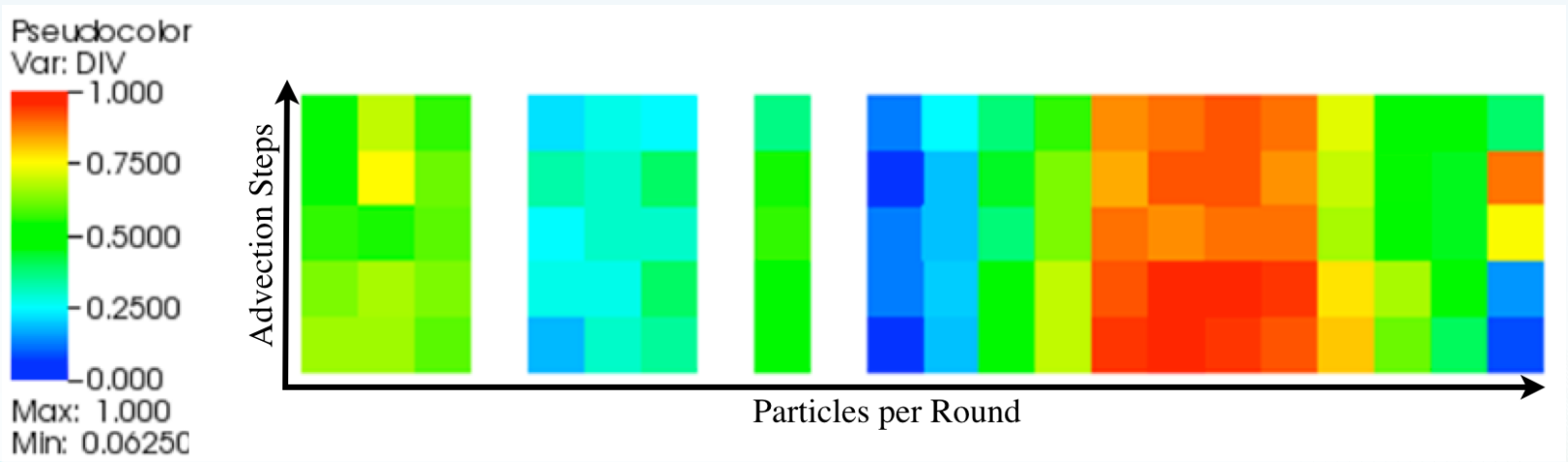


Async

Med. Strategy– Synchronous vs. Asynchronous



Sync



Async

Point #2

- The best performing parameters are similar between synchronous and asynchronous communication

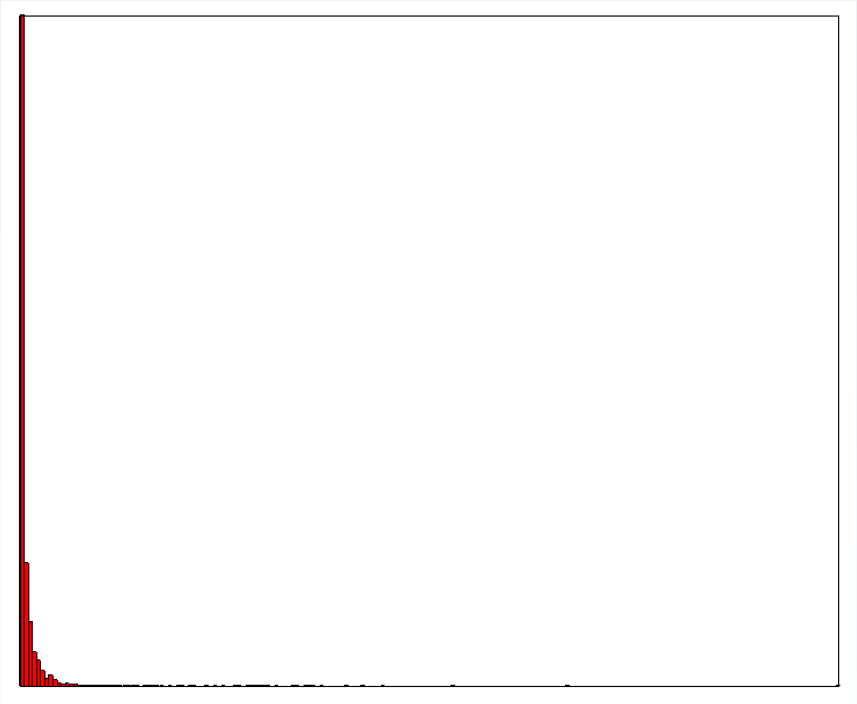
- Asynchronous communication is appropriate to use alone for testing, drastically reducing time required

Measuring Runs

- We are evaluating this algorithm because it is susceptible to load imbalance
- We know load imbalance directly affects the number of rounds necessary for completing advection

- However, the straight runtime has no discernible relationship to number of advection rounds

Runtime Distribution



Runtime Alternative: Imbalance Metric

$$\left(\frac{coreCount}{coreCount-1}\right) \times \left(\frac{waitTime}{workTime}\right)$$

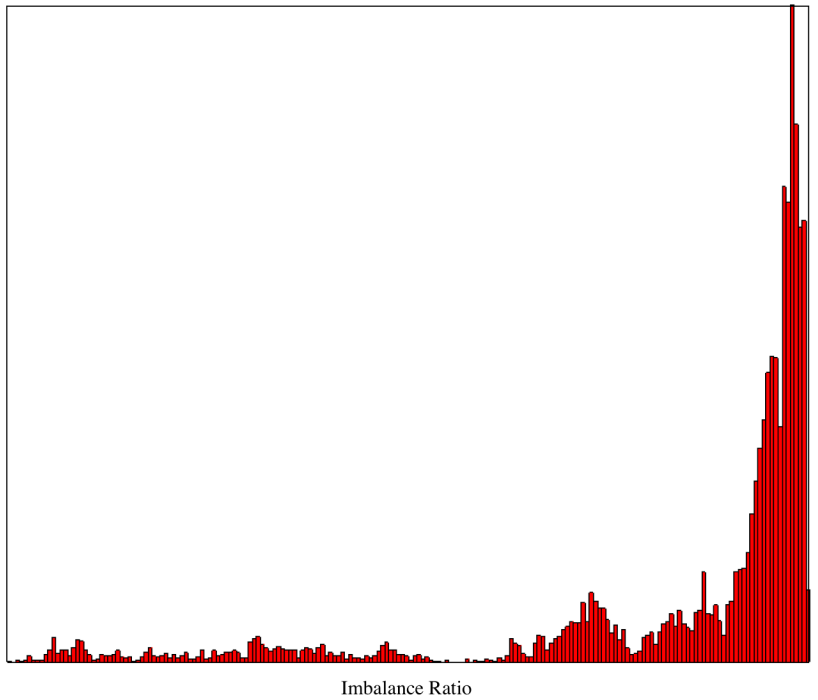
- Average advection time: *adT*
- Average communication time: CT
- Total time to advect particles: $coreCount * adT$
- Total communication $coreCount * CT$
- Best case scenario
  - Each core is performing the average adT
  - No waiting
  - Ratio of wait to work is 0

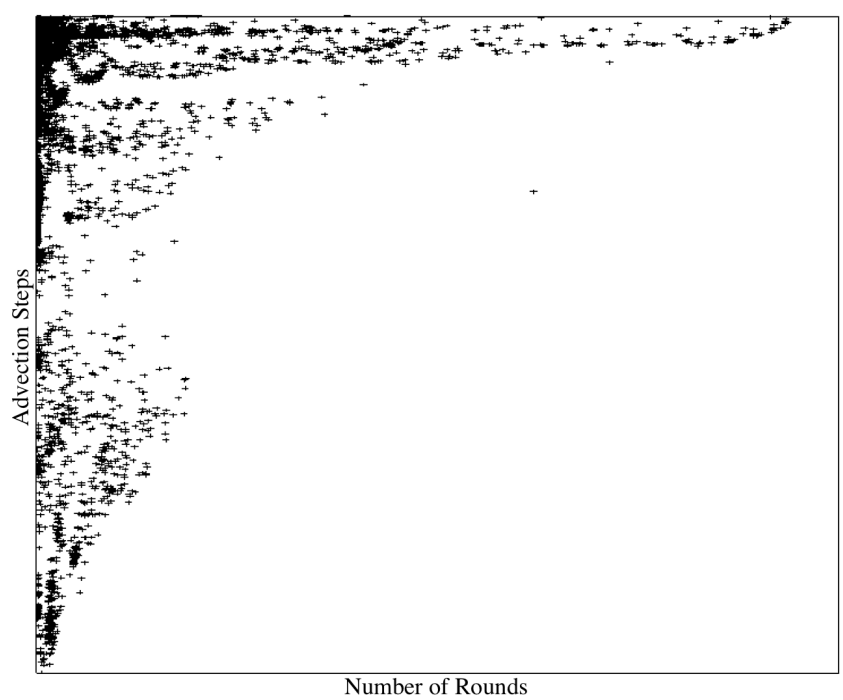
Worst Case Scenario

$$\left(\frac{coreCount}{coreCount-1}\right)\times\left(\frac{waitTime}{workTime}\right)$$

- Serial
- One processor at work each round then all communicated to another
- Total time: $worstCase = coreCount * adT + coreCount * CT$
- Average wait time: $\frac{worstCase*(coreCount-1)}{coreCount}$
- Ratio of wait to work: $\frac{coreCount-1}{coreCount}$
- Worst case ratio is maximum ratio value, dividing by this value keeps imbalance metric in range [0,1]

Imbalance





Points #3 and #4

- Dynamic per-round setting of maxICs is an area requiring further investigation – there is a nice division between the good and bad performing parameters
- Also, there are areas that definitively do not require additional investigation
  - Parameters, e.g. maxSteps
  - Datasets, e.g. synthetic
  - Leaving these out makes for more efficient parameter sweeps

Conclusion

- Generally, if in situ is the future, our research should not be at odds with it

- Specifically
  - Default settings of HPC Algorithms should ***not*** be tuned to terrible
  - There are efficient ways to implement a testing framework to tune defaults to at least half terrible