# Addressing Multiphysics and Hardware Complexity in a HPC Environment

Tony Saad and James C. Sutherland
*University of Utah*

Workshop on Software Development Environments for High-Performance Computing
October 18, 2015 • San Francisco, CA

DOE Awards
DE-NA0002375
DE-NA-000740

NSF PetaApps award 0904631

Matt Might

James Sutherland

Hao Hou

Michael Ballantyne

Christopher Earl

Tony Saad

Abhishek Bagusetty

DOE Awards
DE-NA0002375
DE-NA-000740



National Science Foundation
WHERE DISCOVERIES BEGIN

NSF PetaApps award 0904631

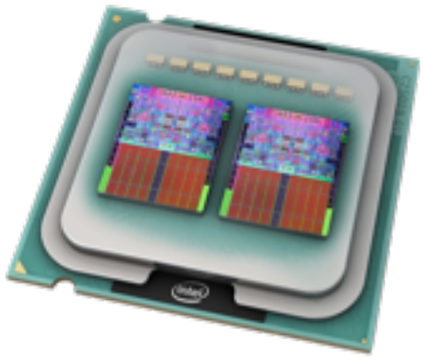
# Three Challenges

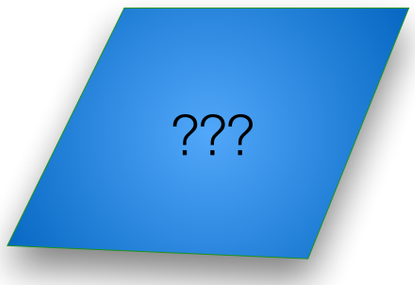
# Three Challenges

# 1. Hardware Complexity

# Three Challenges

# 1. Hardware Complexity



# Three Challenges

## 2. Programming Complexity

# Three Challenges

# 2. Programming Complexity

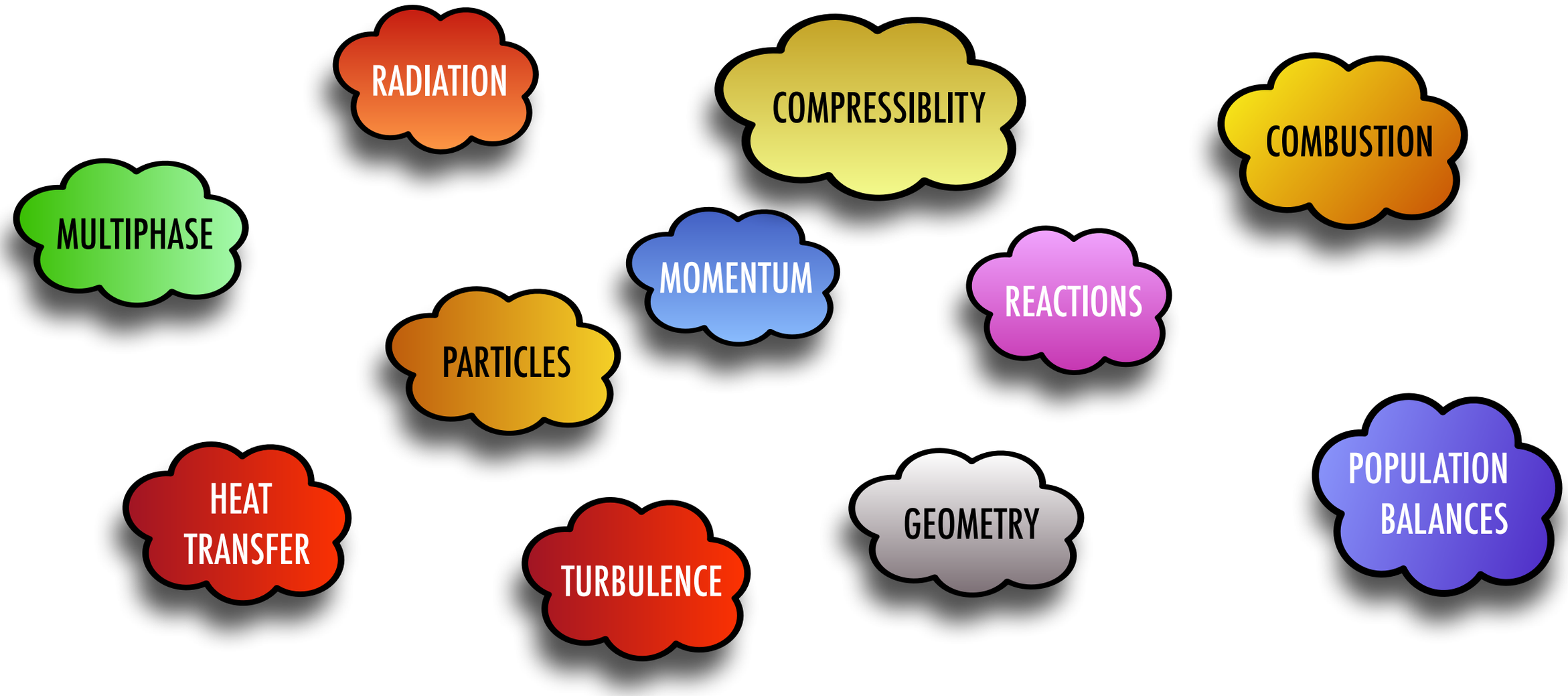
```
for (int i = 0; i<nx; ++i) {
  for (int j=0; j<ny; ++j) {
    for(int k=0;k<nz; ++k){
      rhs =  (phi[i+1, j, k] - 2* phi[i,j,k] + phi[i-1,j,k])/dx2
           + (phi[i, j+1, k] - 2* phi[i,j,k] + phi[i,j-1,k])/dy2
           + (phi[i, j, k+1] - 2* phi[i,j,k] + phi[i,j,k-1])/dz2;
    }
  }
}
```

# Three Challenges

## 3. Multiphysics Complexity

# Three Challenges

# 3. Multiphysics Complexity



# Three Challenges

1. Hardware Complexity
2. Programming Complexity
3. Multiphysics Complexity

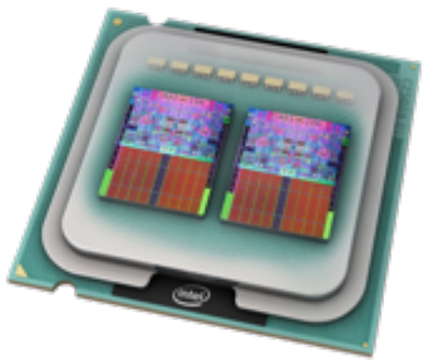
# Three Solutions

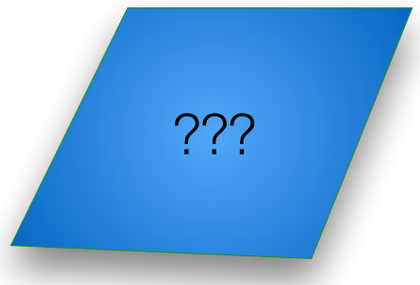
write one code
execute
on arbitrary architectures

write one code

execute

on arbitrary architectures





# write expressive syntax

$$\nabla \cdot (k \nabla \phi)$$

# write expressive syntax

$$\nabla \cdot (k \nabla \phi)$$

```
div( k * grad(phi) )
```

# write expressive syntax

$$\nabla \cdot (k \nabla \phi)$$

```
div( k * grad(phi) )
```

# that is as fast as nested loops

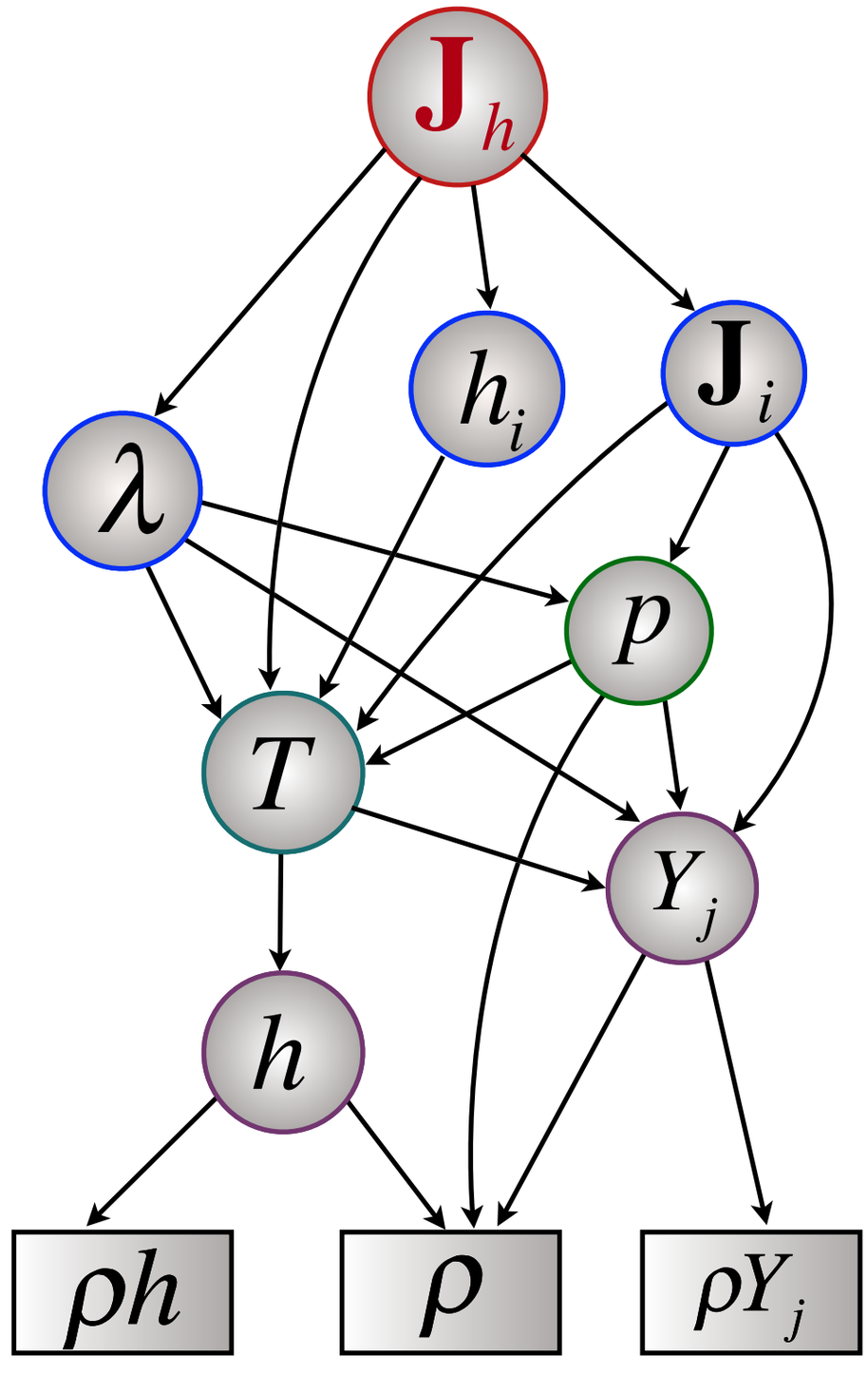
# that is as fast as nested loops

```
div( k * grad(phi) )
```

```
for (int i = 0; i<nx; ++i) {
  for (int j=0; j<ny; ++j) {
    for(int k=0;k<nz; ++k){
      rhs =  (phi[i+1, j, k] - 2* phi[i,j,k] + phi[i-1,j,k])/dx2
           + (phi[i, j+1, k] - 2* phi[i,j,k] + phi[i,j-1,k])/dy2
           + (phi[i, j, k+1] - 2* phi[i,j,k] + phi[i,j,k-1])/dz2;
    }
  }
}
```

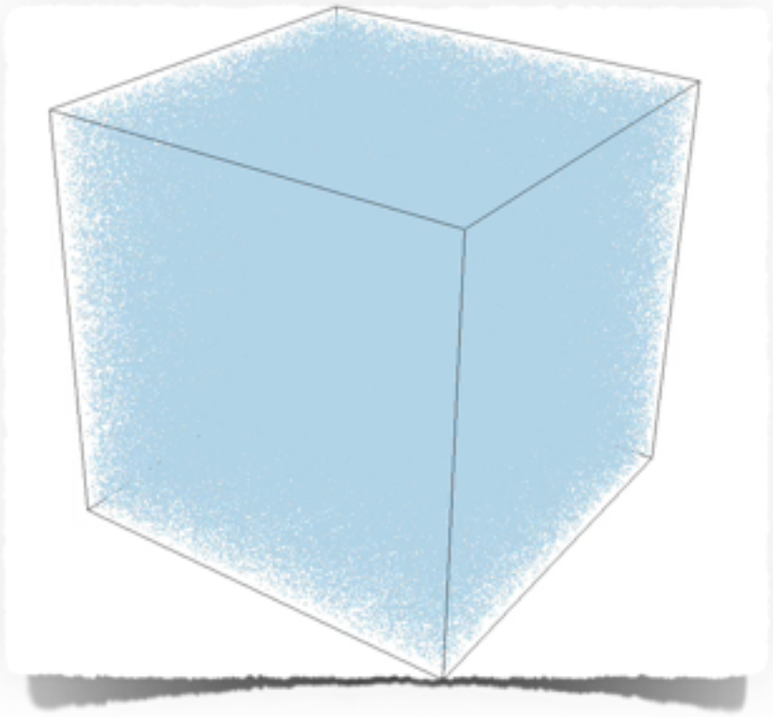
# Tame multi-physics complexity

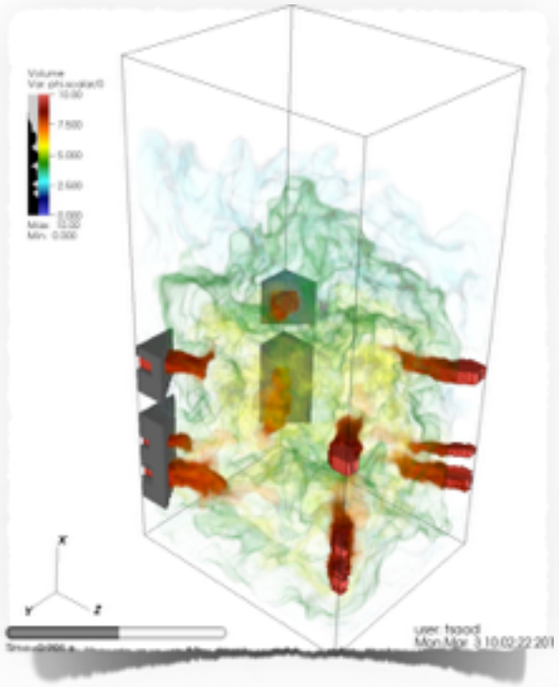
# Tame multi-physics complexity

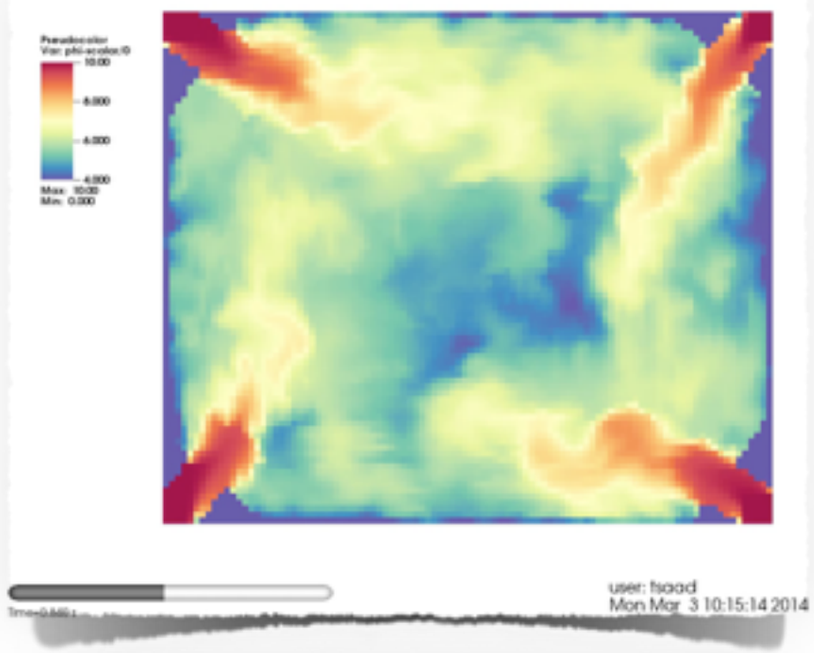


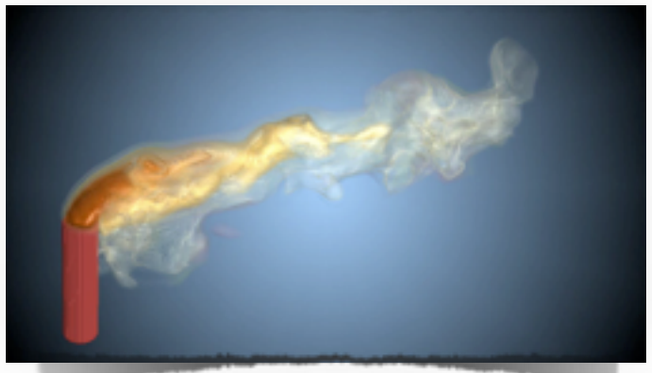
# Wasatch (*et al.*)

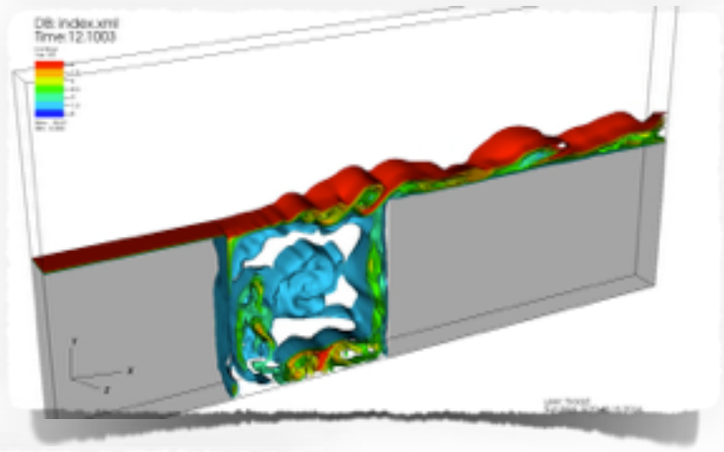
Wasatch is
a multi-architecture
CFD
and multi-physics code
that enables fast deployment
of physics models
and PDEs

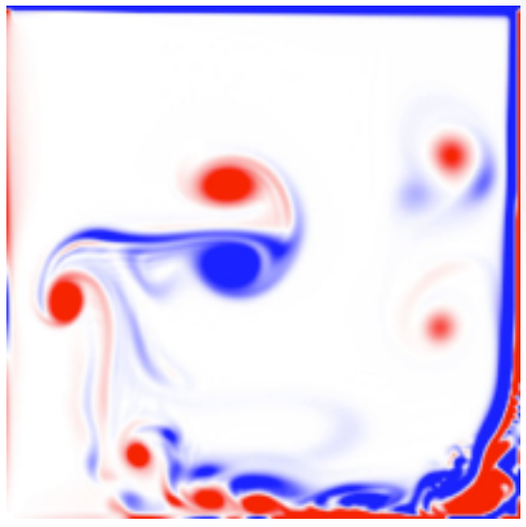


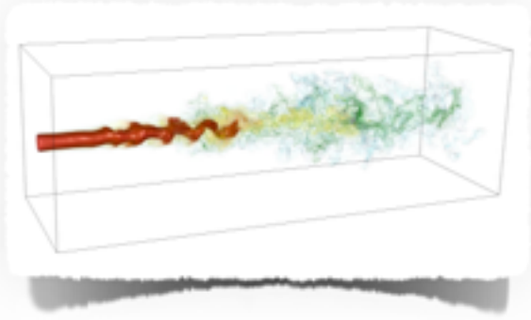

Volume
Var: phi-scalar0
10.00
7.500
5.000
2.500
0.000
Max: 10.00
Min: 0.000
X
Y
Z
user: hoad
Mon Mar 3 10:02:22 201

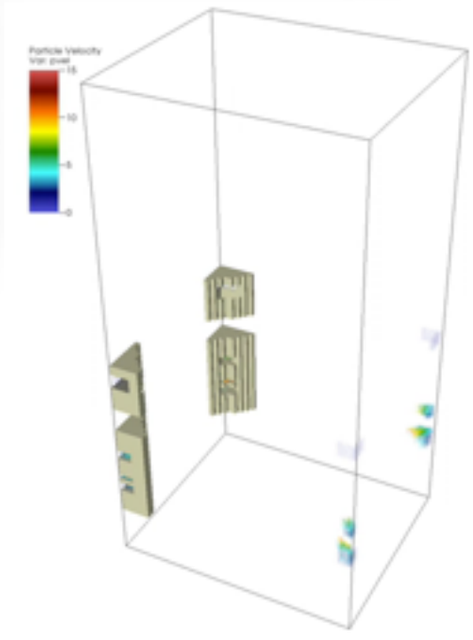

Pseudocolor
Var: phi-scalar0
10.00
6.000
4.000
2.000
Max: 10.00
Min: 0.000
user: hoad
Mon Mar 3 10:15:14 2014




DB: index.xml
Time: 12.1003

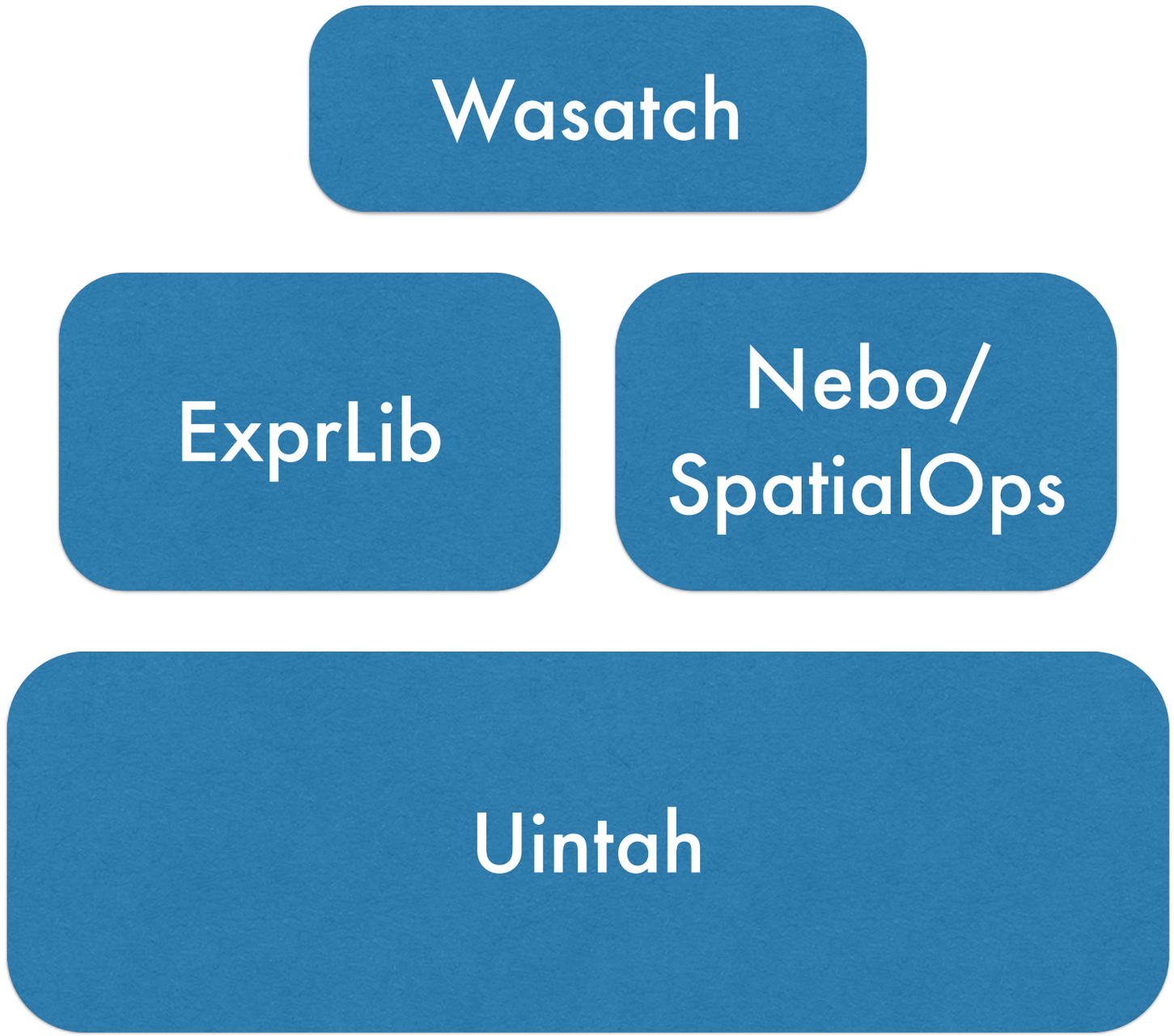





Particle Velocity
Var: pVel
15
10
5
0

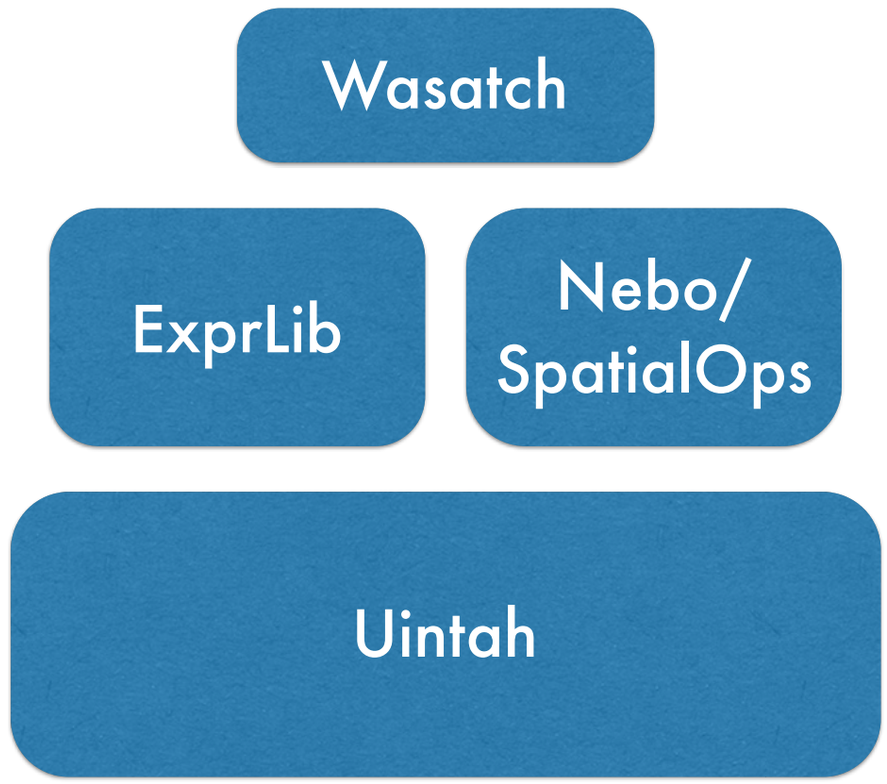
# Can’t do it alone…

# Life in an HPC Ecosystem



# Uintah

- Task based parallel infrastructure
- Provides parallel IO
- Grid specification
- Domain decomposition



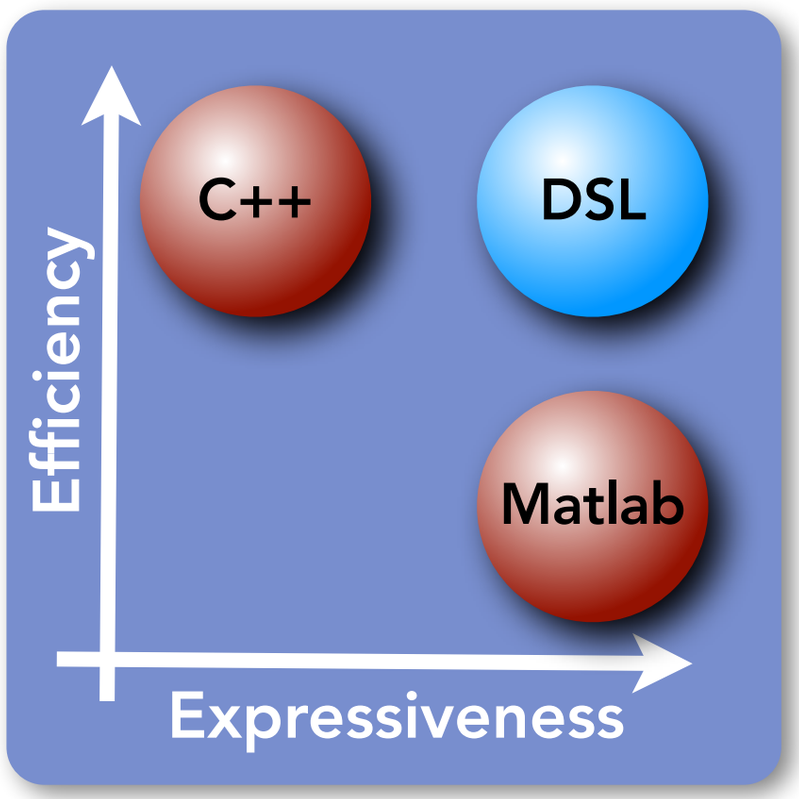
# Nebo DSL

Field & stencil operations:

(and much more)

```
rhs <<= -divOpX( xConvFlux + xDiffFlux )
        -divOpY( yConvFlux + yDiffFlux )
        -divOpZ( zConvFlux + zDiffFlux );
```

Can “chain” stencil operations.

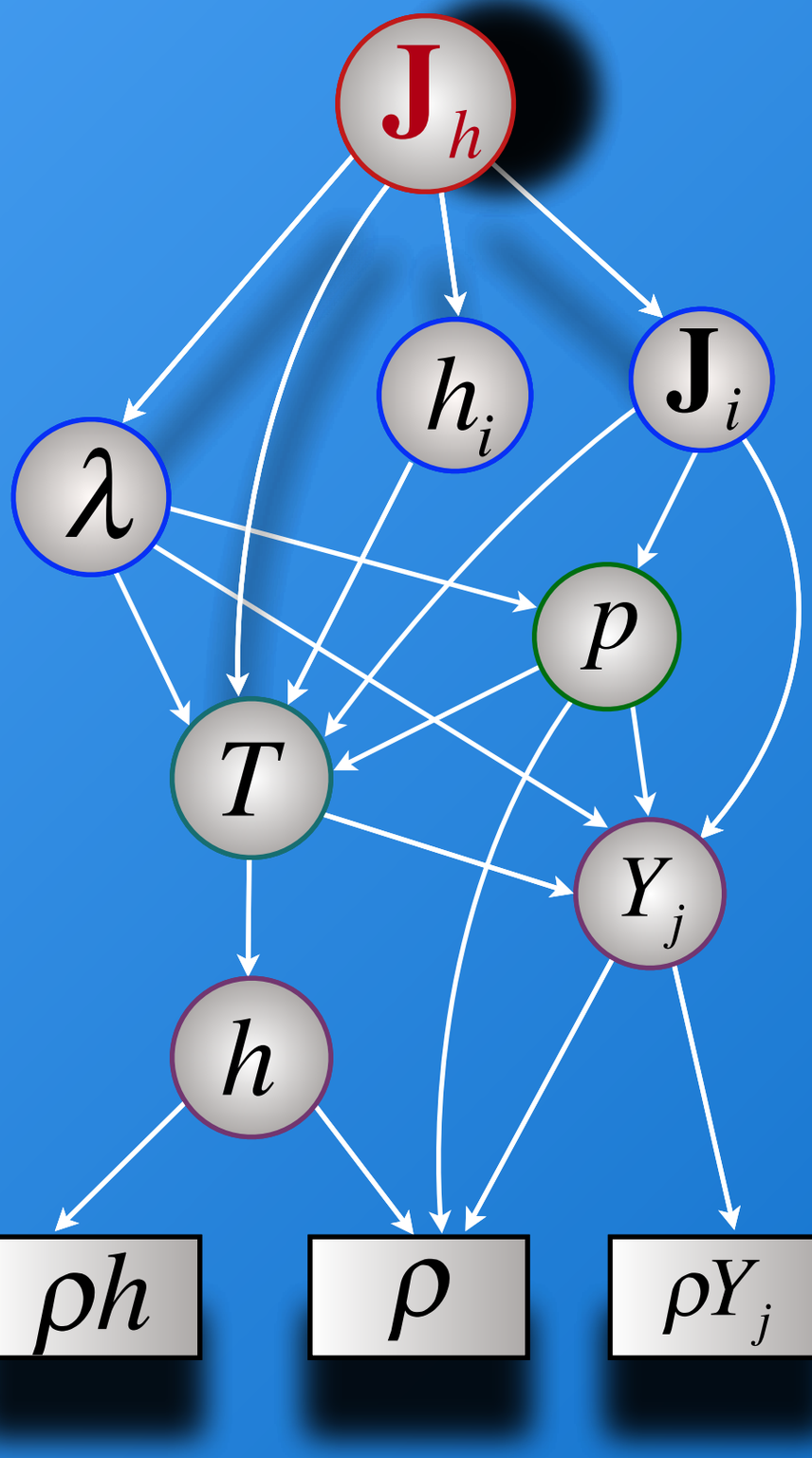



- 70+ natively supported discrete operators, easily define new ones
- Masks - allow operations on field subsets
- CPU (serial, multicore), GPU, Xeon Phi backends
- conditional operations (vectorized ‘if’)

**Software Stacks for Next-Gen Production Codes @ 11 AM**

# ExprLib

$$\mathbf{J}_h = -\lambda \nabla T + \sum_{i=1}^{n_s} h_i \mathbf{J}_i$$




## MODEL A

$$\lambda = \lambda_0 = \text{const}$$

$$\mathbf{J}_i = -\sum_{j=1}^{n_s} D \nabla Y_j$$

$$h_i = h_i(T)$$

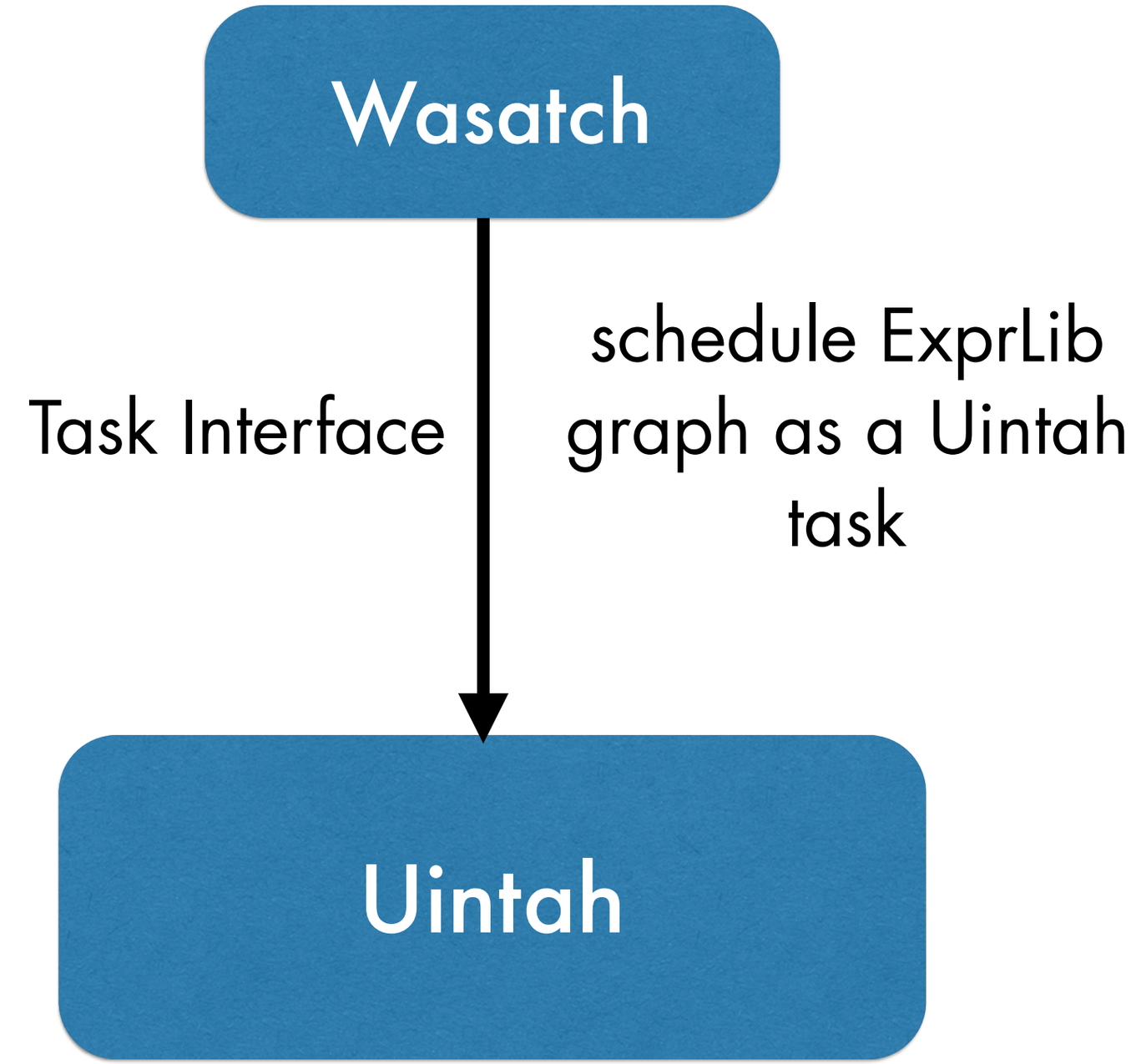
## MODEL B

$$\lambda = \lambda(T, p, Y_i)$$

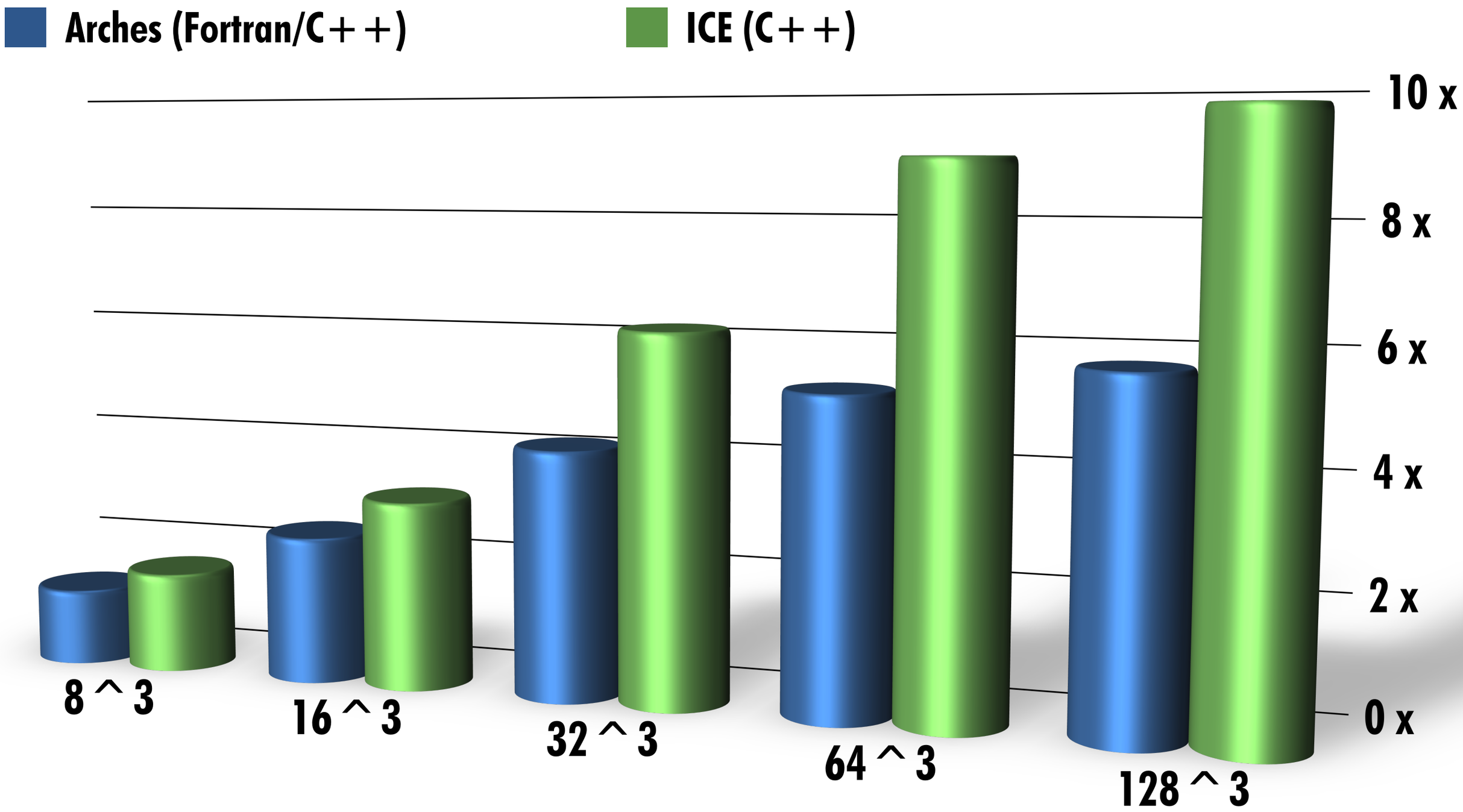
$$\mathbf{J}_i = -\sum_{j=1}^{n_s} D_{ij}(T, p, Y_k) \nabla Y_j - D_i^T(T, p, Y_k) \nabla T$$

$$h_i = h_i(T)$$


Notz, P. K., Pawlowski, R. P., and Sutherland, J. C., "Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE software," Vol. IV, 2010.


Wasatch
Task Interface
schedule ExprLib graph as a Uintah task
Uintah

# Speedup using Nebo DSL Syntax compared to traditional iterative/ijk loops in Uintah



# In Practice

# Solve

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

# 1. Define expressions (easy)

# 1. Define expressions (easy)

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

Convective Flux

Diffusive Flux

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

# 2. Define input spec (easy)

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

# 2. Define input spec (easy)

```
<TransportEquation equation="generic">
  <SolutionVariable>phi</SolutionVariable>
  <DiffusiveFlux direction="X" coefficient="0.4"/>
  <DiffusiveFlux direction="Y" coefficient="0.2"/>
  <DiffusiveFlux direction="Z" coefficient="0.1"/>

  <ConvectiveFlux direction="Y" method="SUPERBEE">
    <AdvectiveVelocity>
        <NameTag name="v"/>
    </AdvectiveVelocity>
  </ConvectiveFlux>

</TransportEquation>
```

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

# 3. Implement expressions (easy)

# 3. Implement expressions (easy)

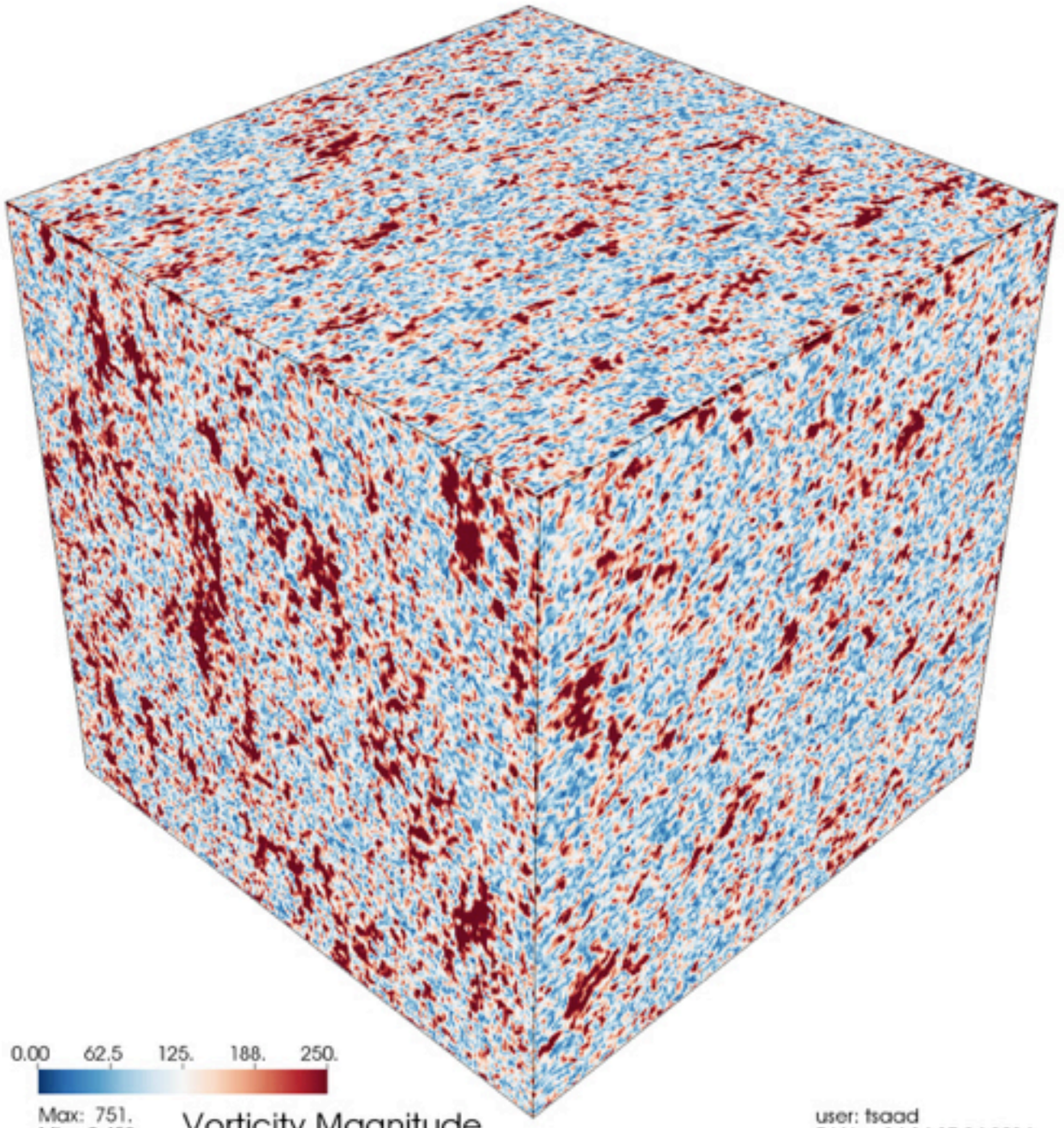
$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

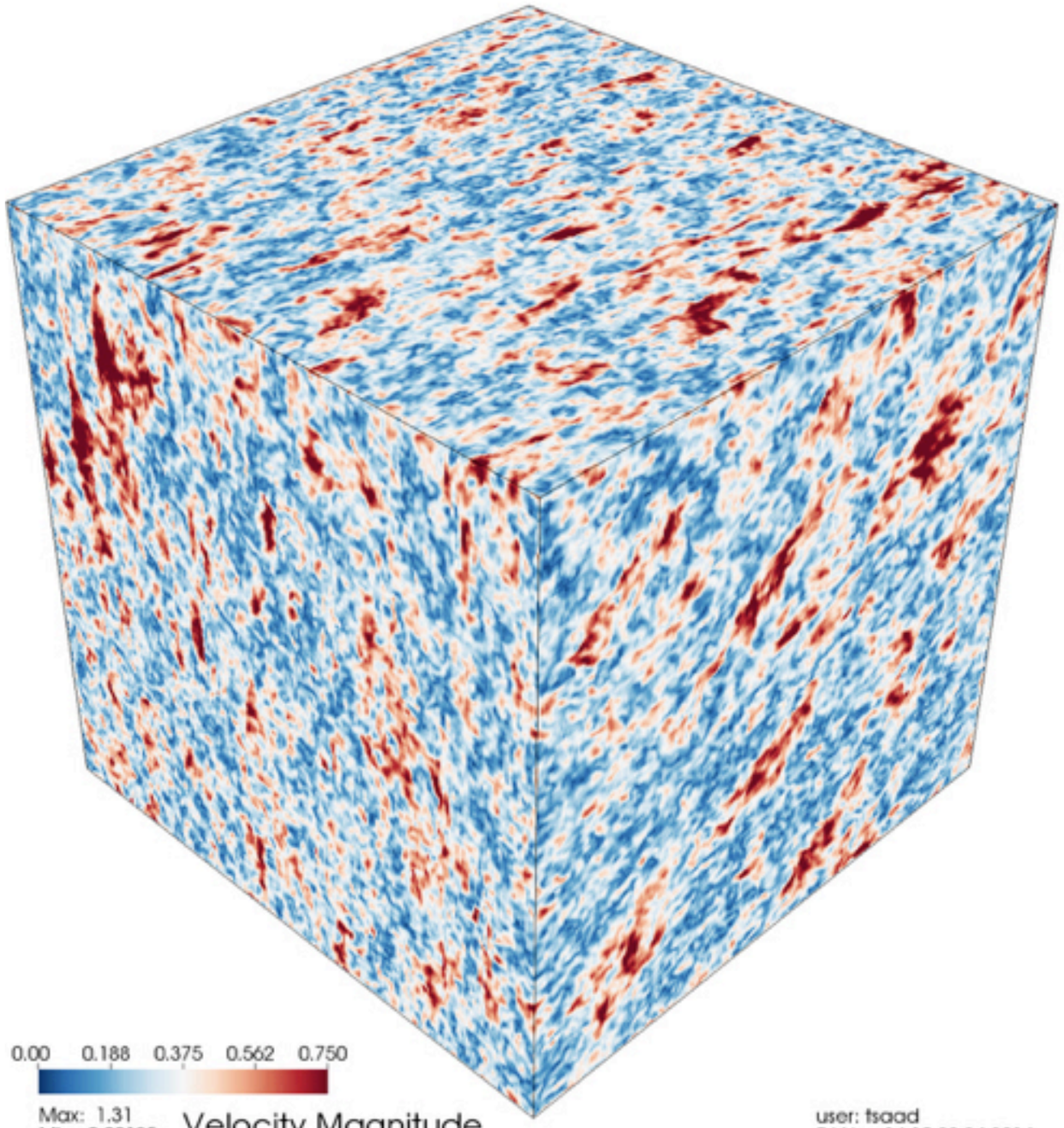
```
cflux <<= div(u*interp(phi))

dflux <<= div(interp(k)*grad(phi))
```

Interaction with computer science

$$\frac{\partial \phi}{\partial t} = -\nabla \cdot \mathbf{u}\phi - \nabla \cdot (k \nabla \phi)$$

# 4. Test, verify, validate

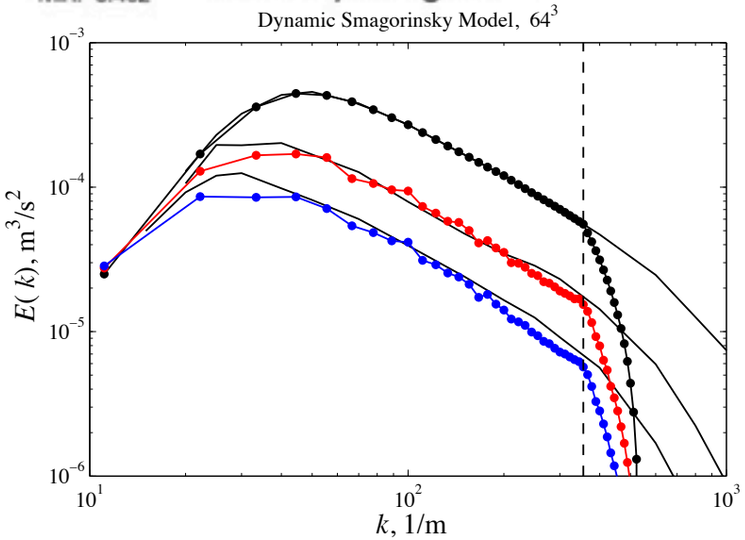


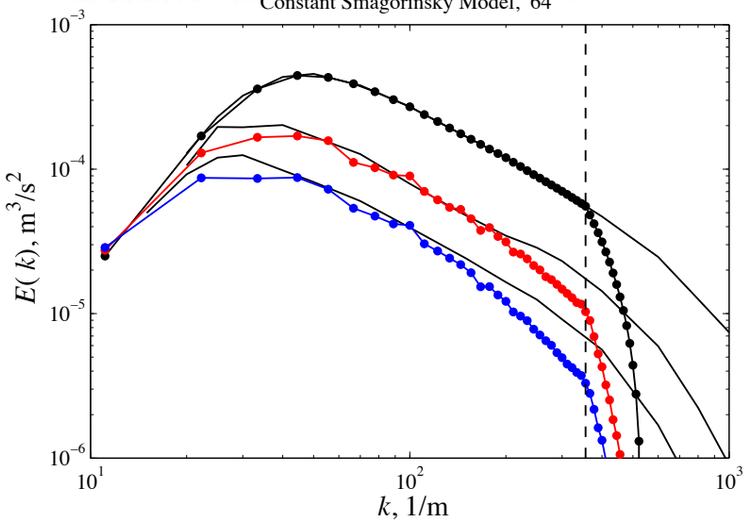

user: tsaad
Fri Nov 14 14:17:24 2014

user: tsaad
Fri Nov 14 15:00:24 2014

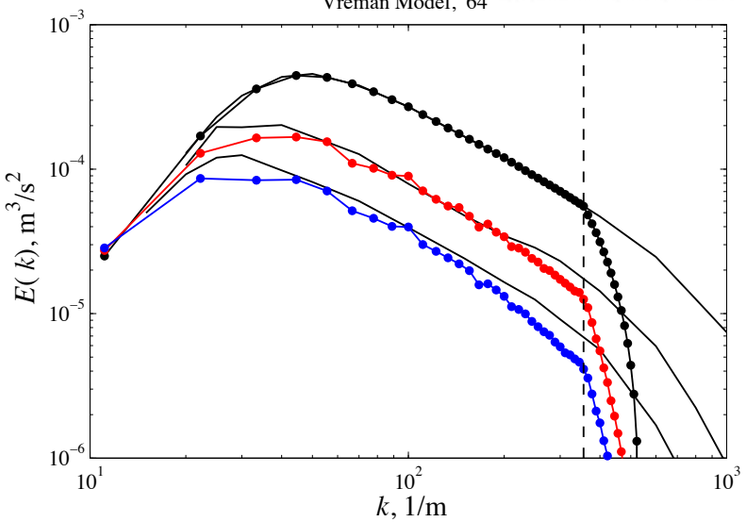
Dynamic Smagorinsky Model, $64^3$



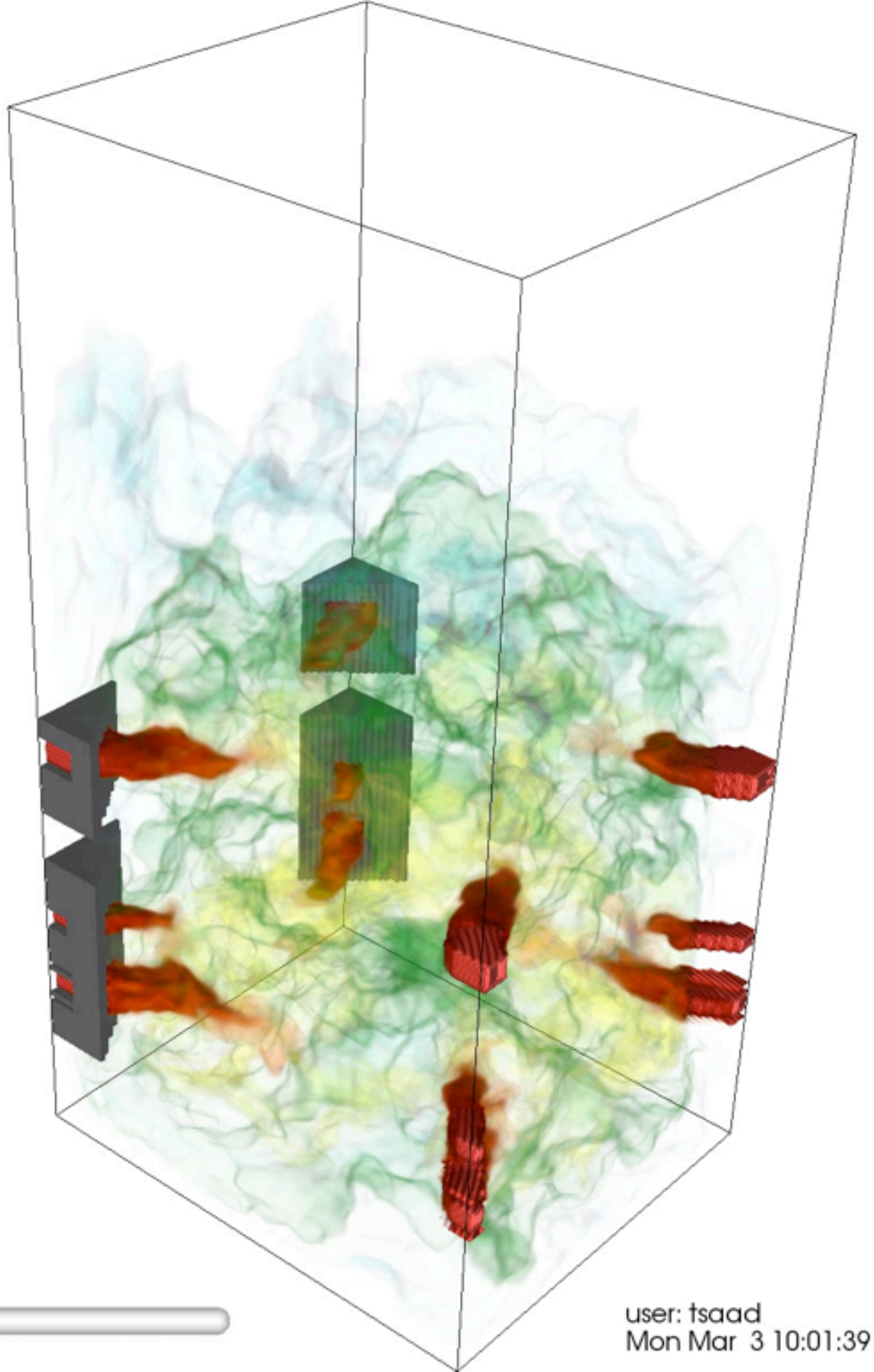

Constant Smagorinsky Model, $64^3$




Vreman Model, $64^3$



Volume
Var: phi-scalar/0
10.00
7.500
5.000
2.500
0.000
Max: 10.00
Min: 0.000



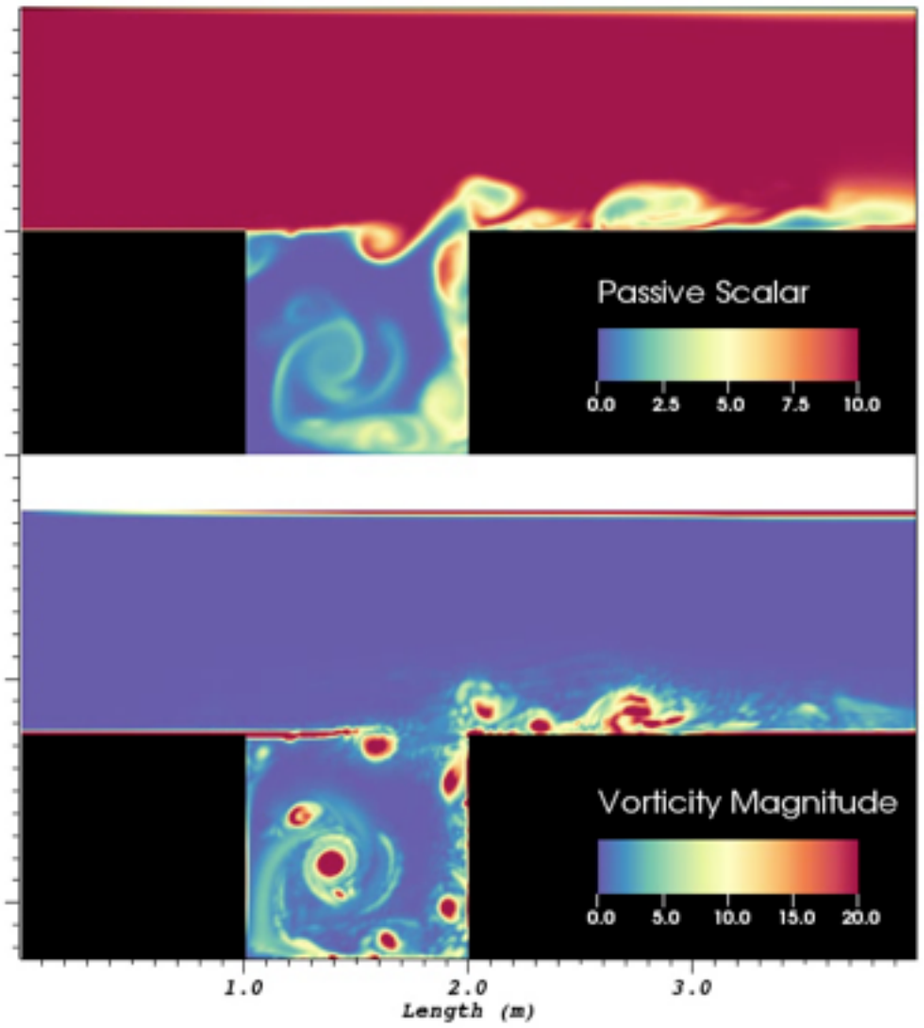
Time=0.000 s

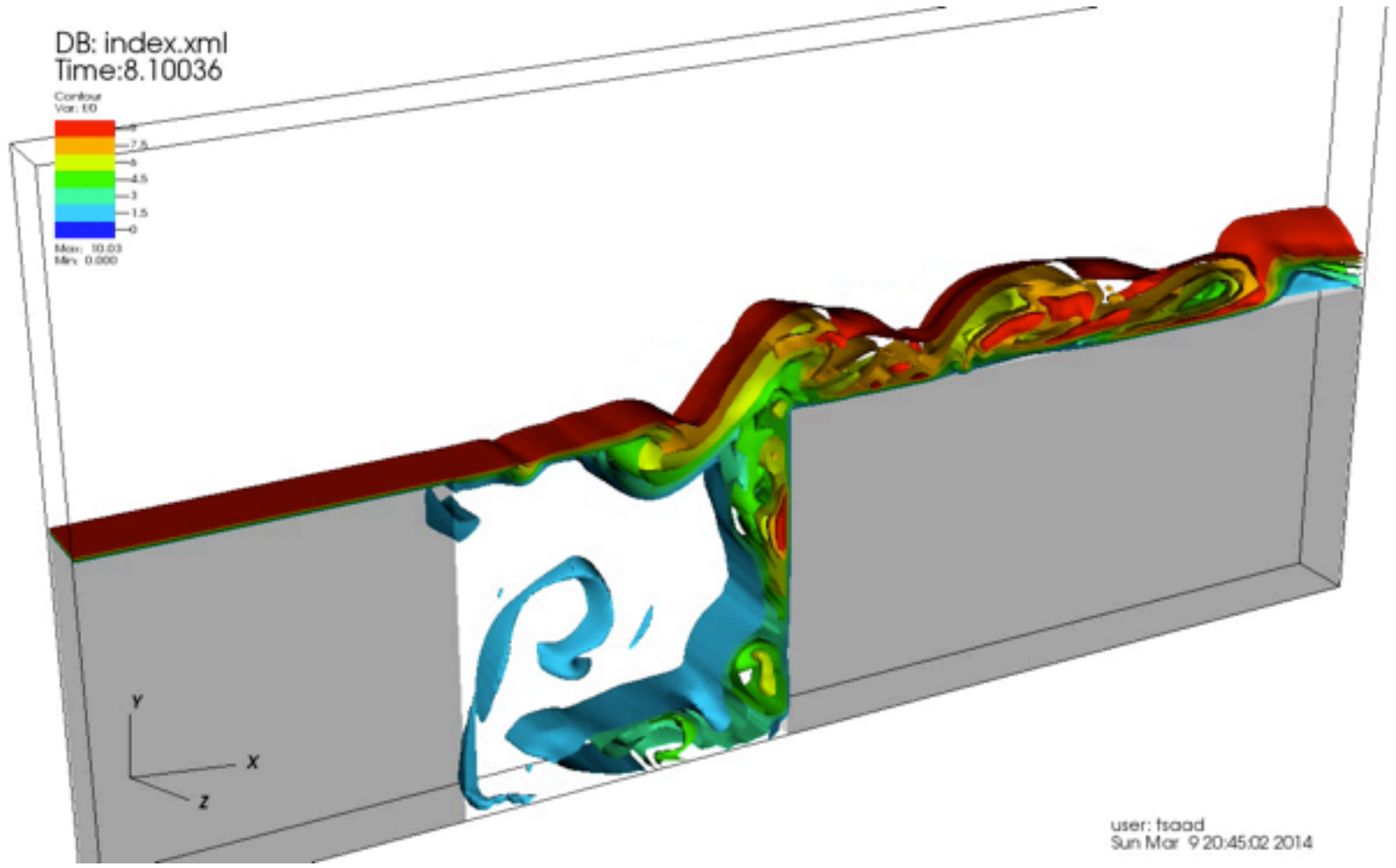
user: tsaad
Mon Mar 3 10:14:12 2014

X
Y
Z

Time=0.781 s

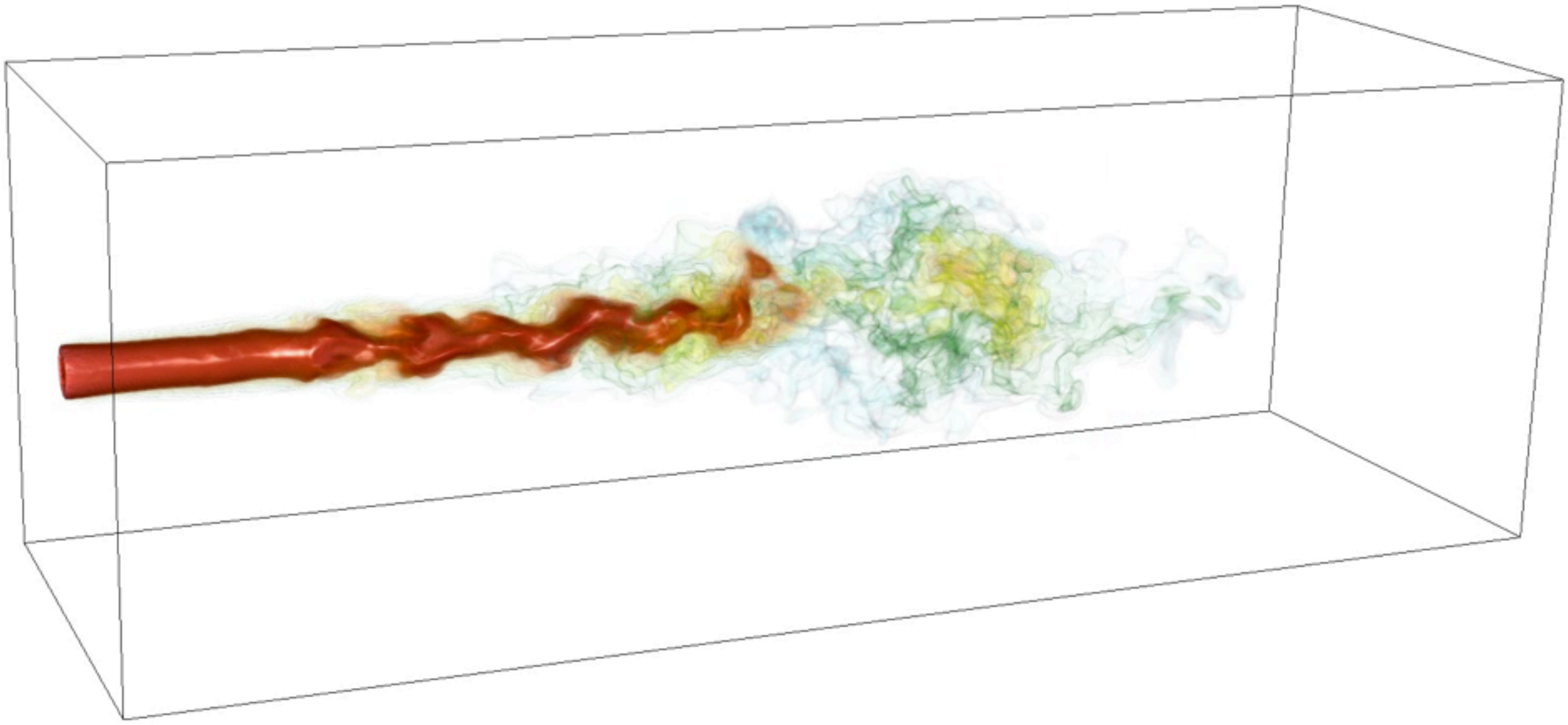
user: tsaad
Mon Mar 3 10:01:39 201

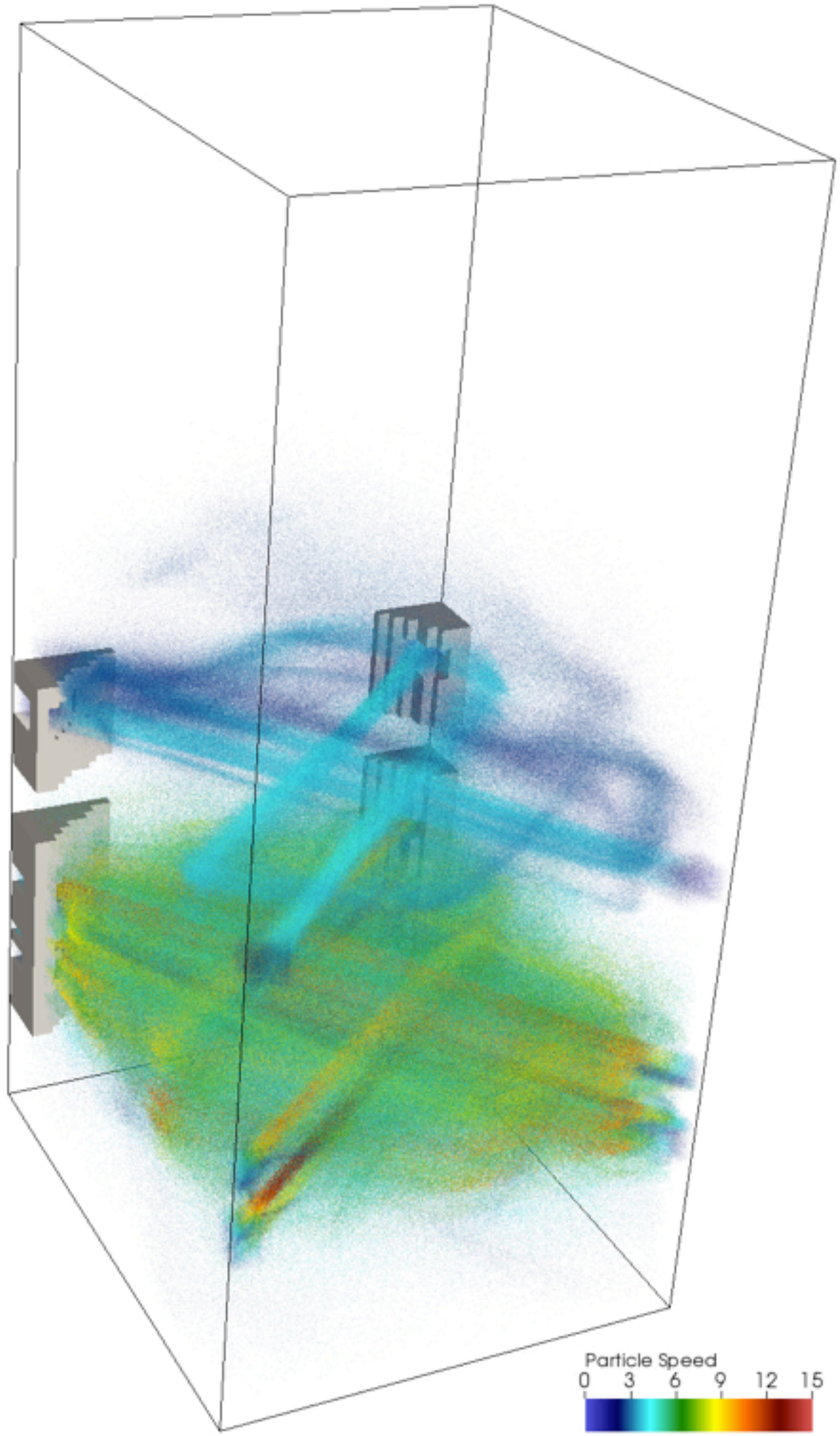


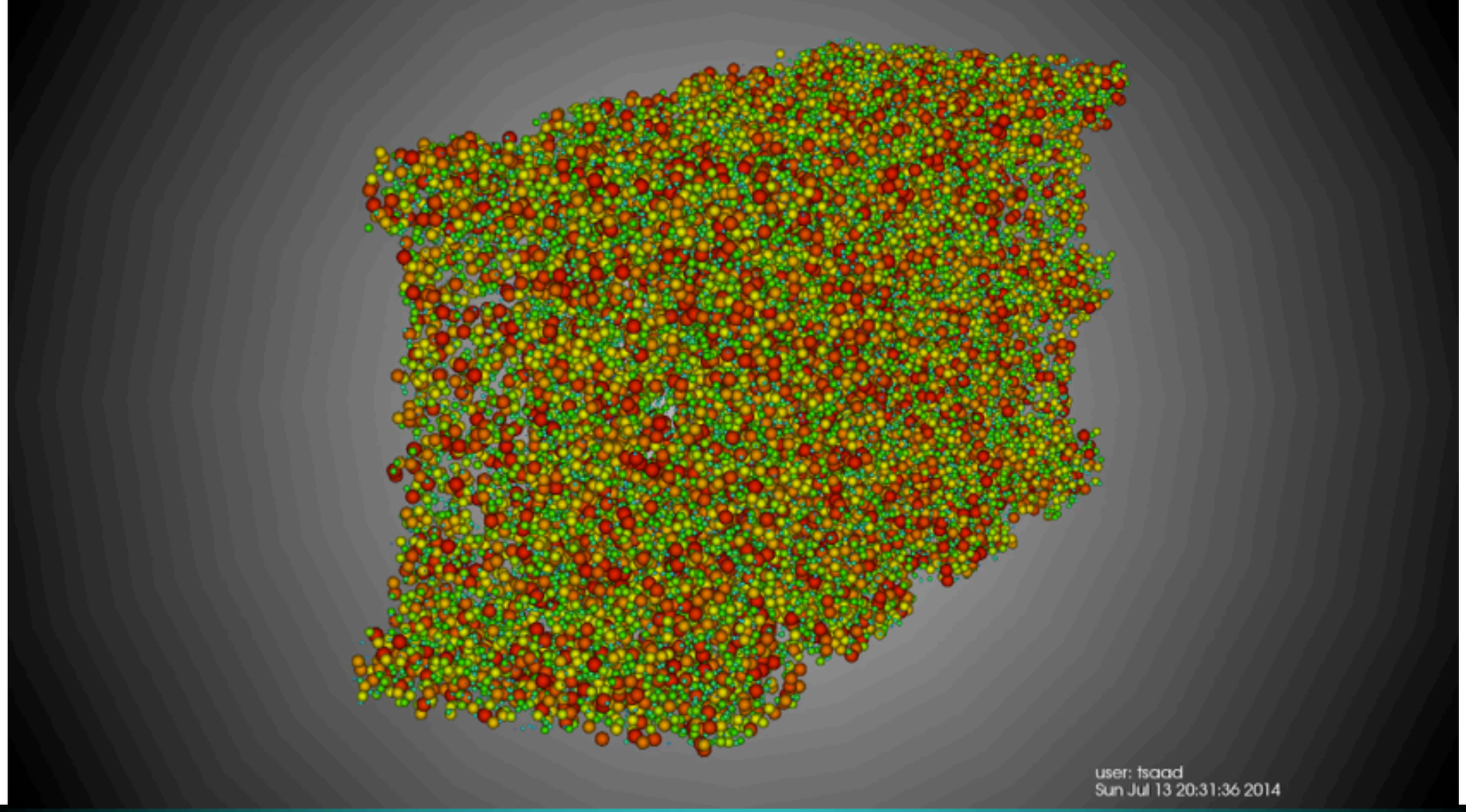



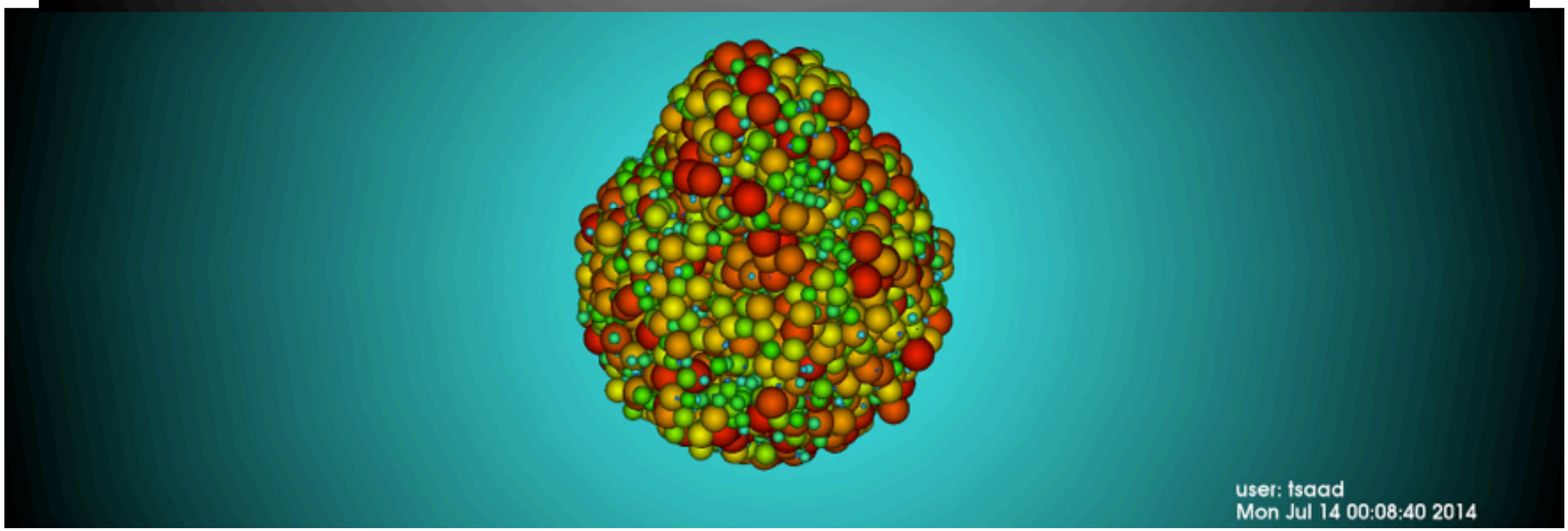

Time=8.10036 s

user: tsaad
Sun Mar 9 20:39:13 20



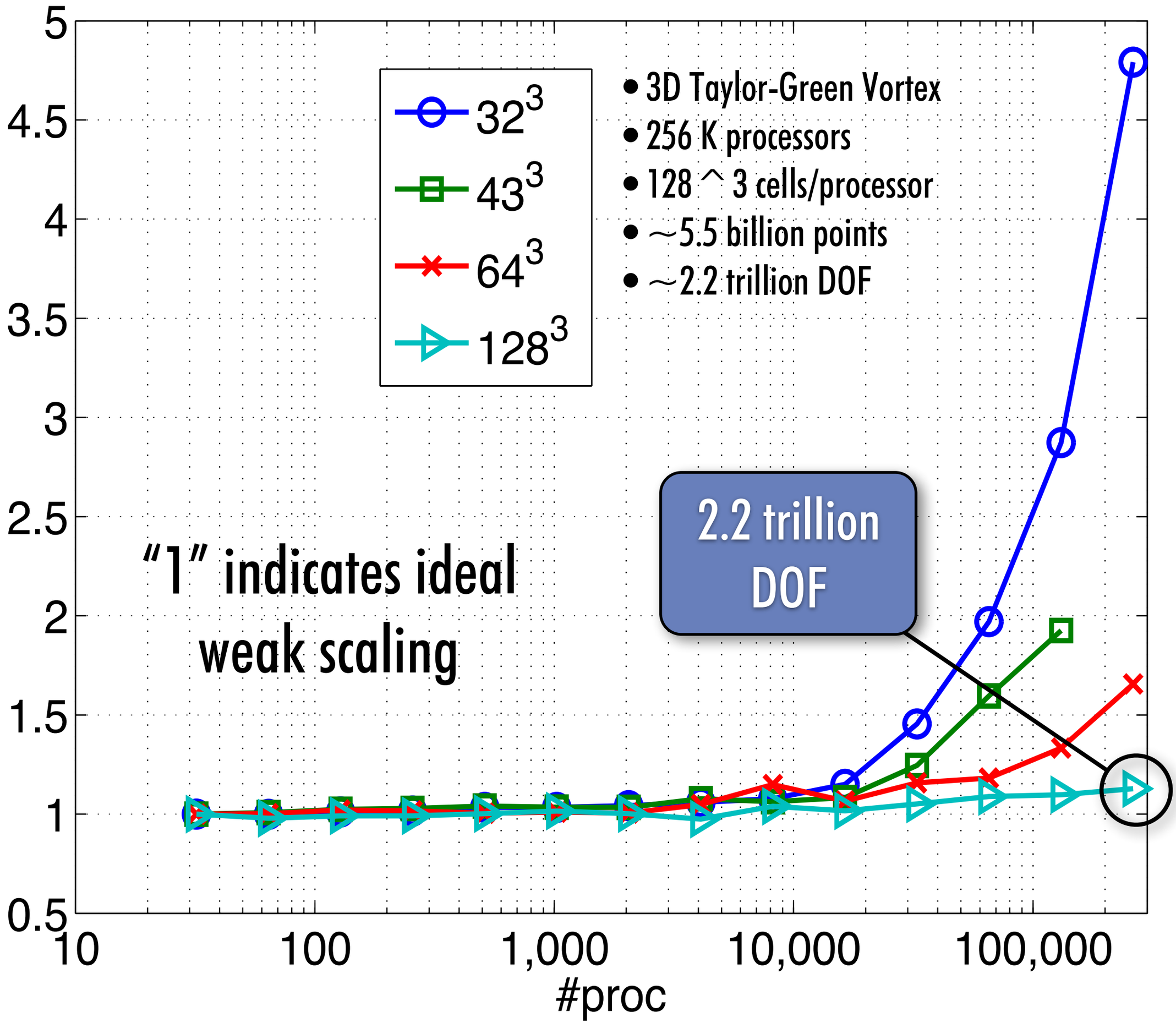

Particle Speed
0 3 6 9 12 15


user: tsaad
Sun Jul 13 20:31:36 2014


user: tsaad
Mon Jul 14 00:08:40 2014

# Weak scaling on Titan

Scalability

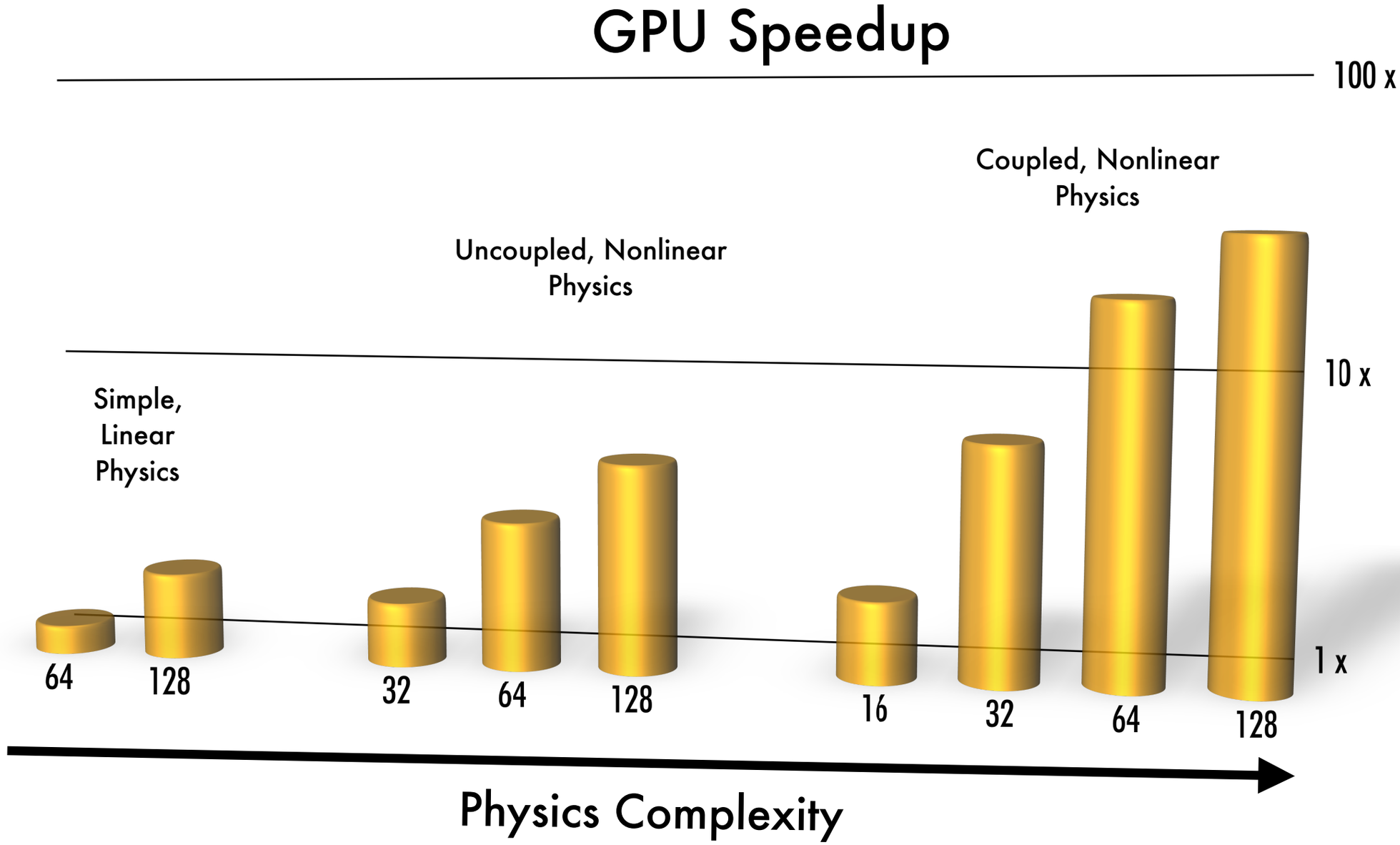


# GPU Scaling

$$\frac{\partial \phi_i}{\partial t} + \nabla \cdot \mathbf{u}\phi_i = -\nabla \cdot \Gamma \nabla \phi_i + S; \quad i = 0,1,\cdots$$

$$S = \begin{cases} 0 & \text{Simple physics} \\ e^{\phi_i} & \text{Moderate} \\ \sum_j e^{\phi_j} & \text{Coupled} \end{cases}$$

# Single-Node GPU Speedup



# Scaling on Titan

# Simple Physics (Scalar Transport without Source Terms)

10 Equations  20 Equations

Weak Scaling - Scalar Transport, No Source Terms

Mean time per timestep

10s
1s
0.1s
0.01s

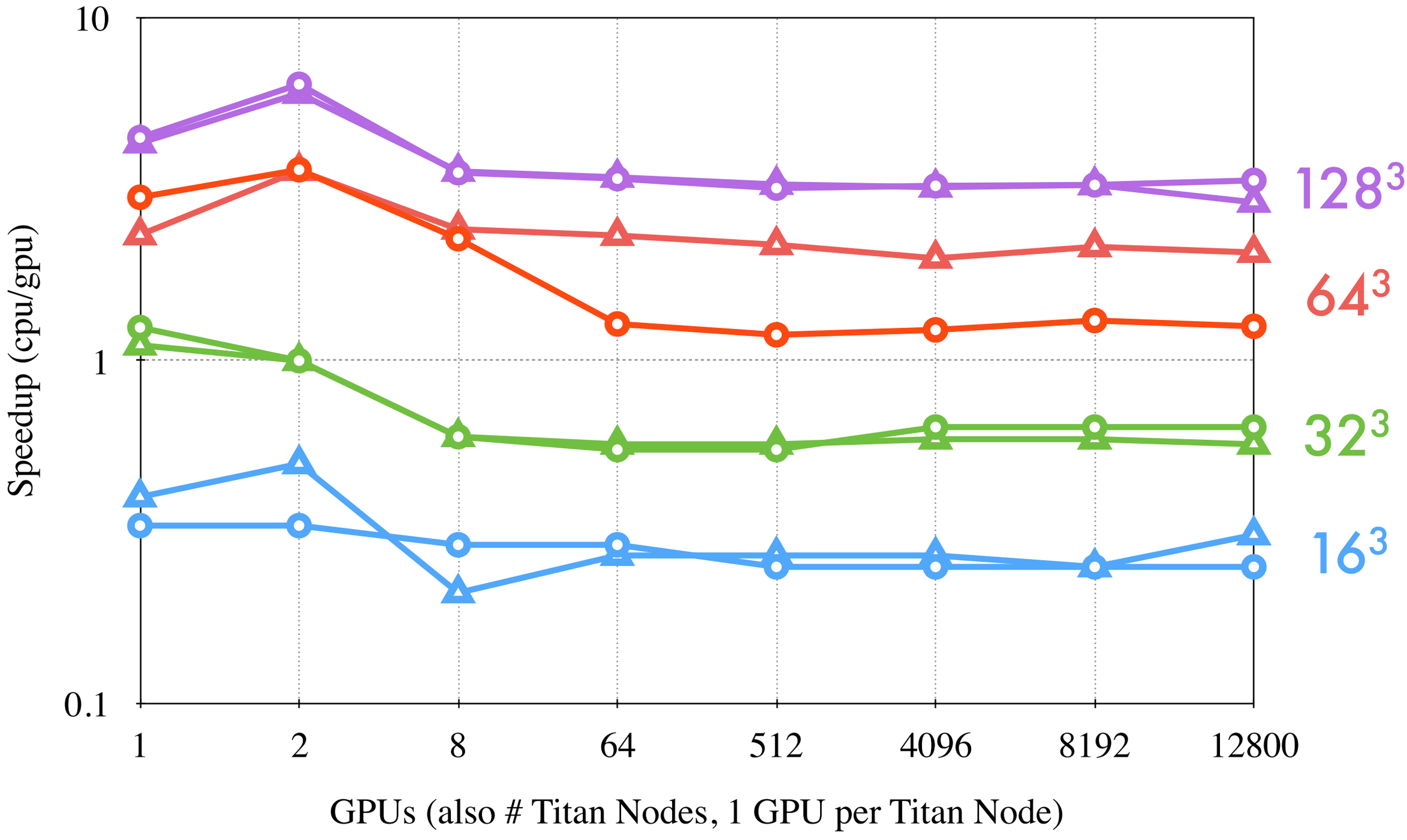
$128^3$
$64^3$
$32^3$
$16^3$

1 2 8 64 512 4096 8192 12800

GPUs (also # Titan Nodes, 1 GPU per Titan Node)

10 Equations  20 Equations

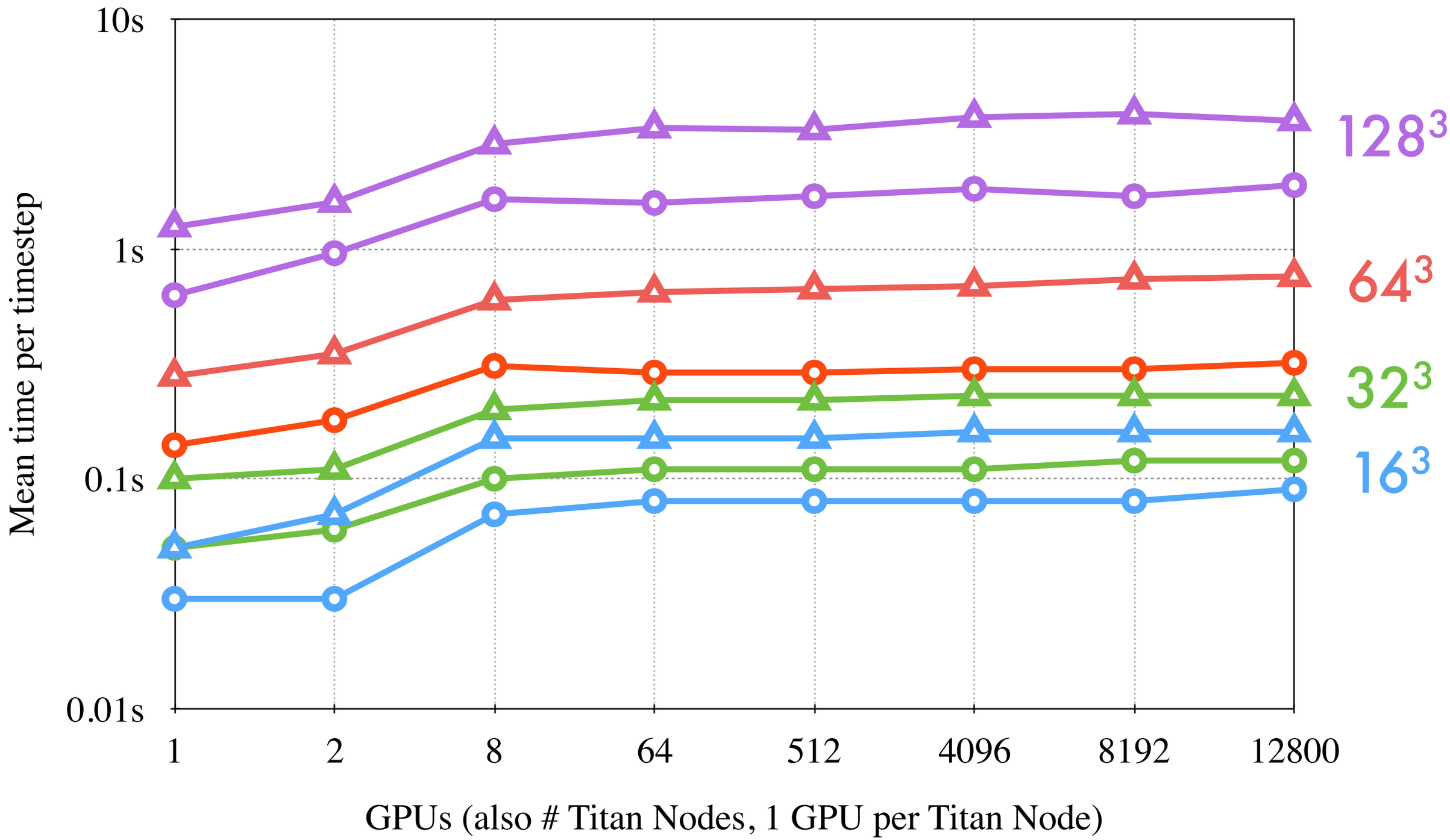
Speedup - Scalar Transport, No Source Terms



# Slightly Complex Physics (Scalar Transport with Uncoupled Source Terms)

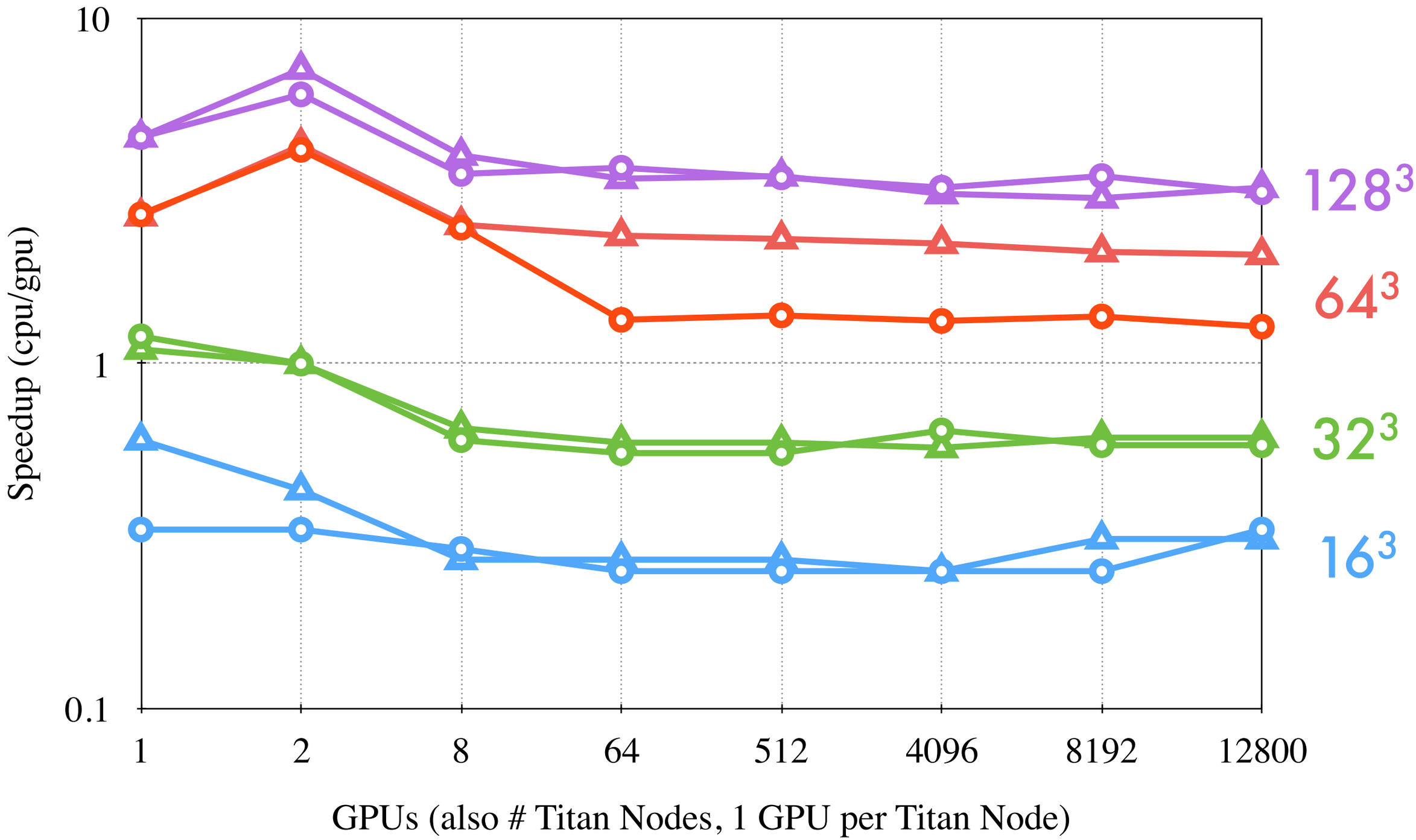
10 Equations  20 Equations

Weak Scaling - Scalar Transport, Uncoupled Source Terms



10 Equations  20 Equations

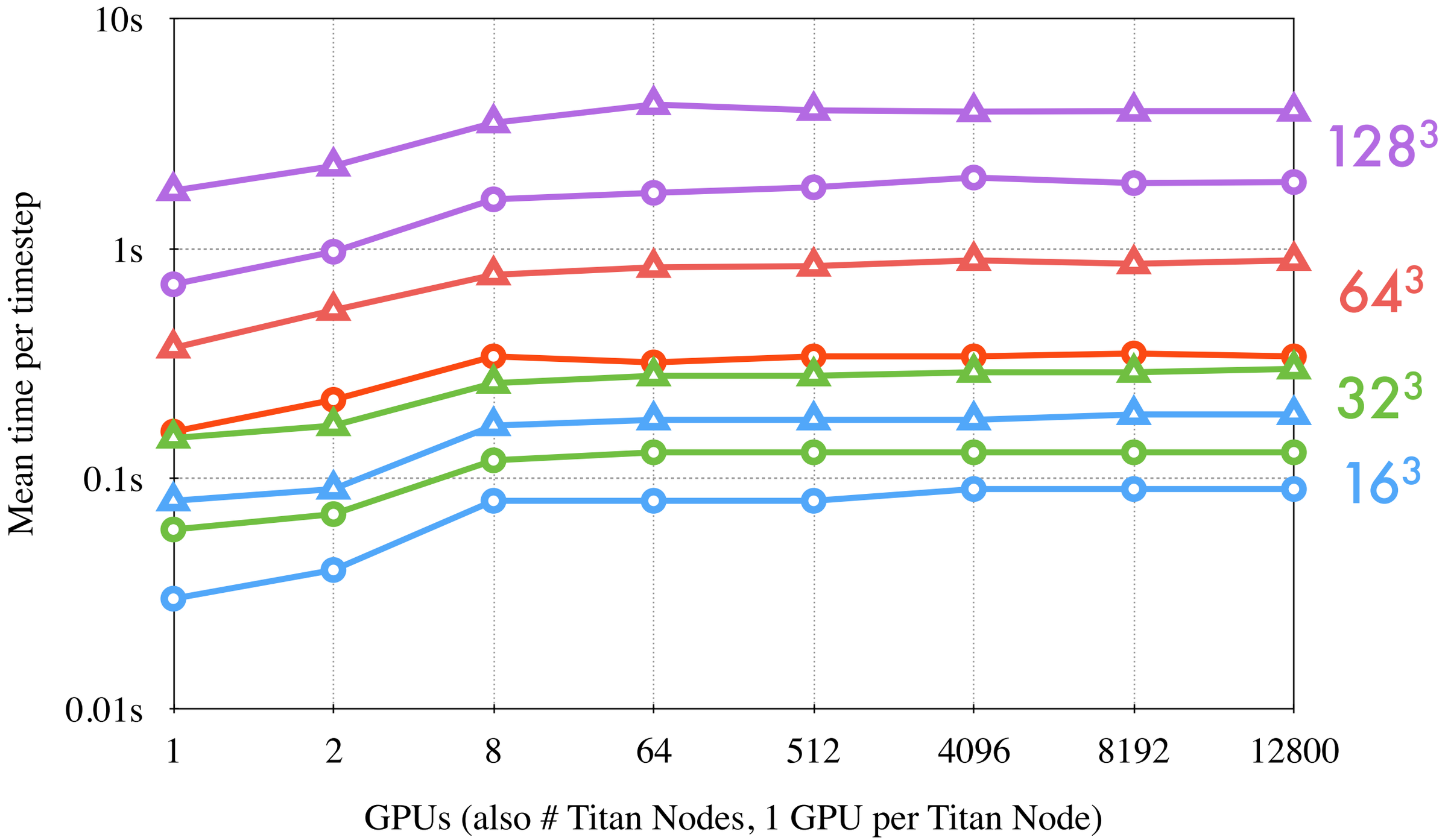
Speedup - Scalar Transport, No Source Terms



# Somewhat Complex Physics (Scalar Transport with Coupled Source Terms)

10 Equations  20 Equations

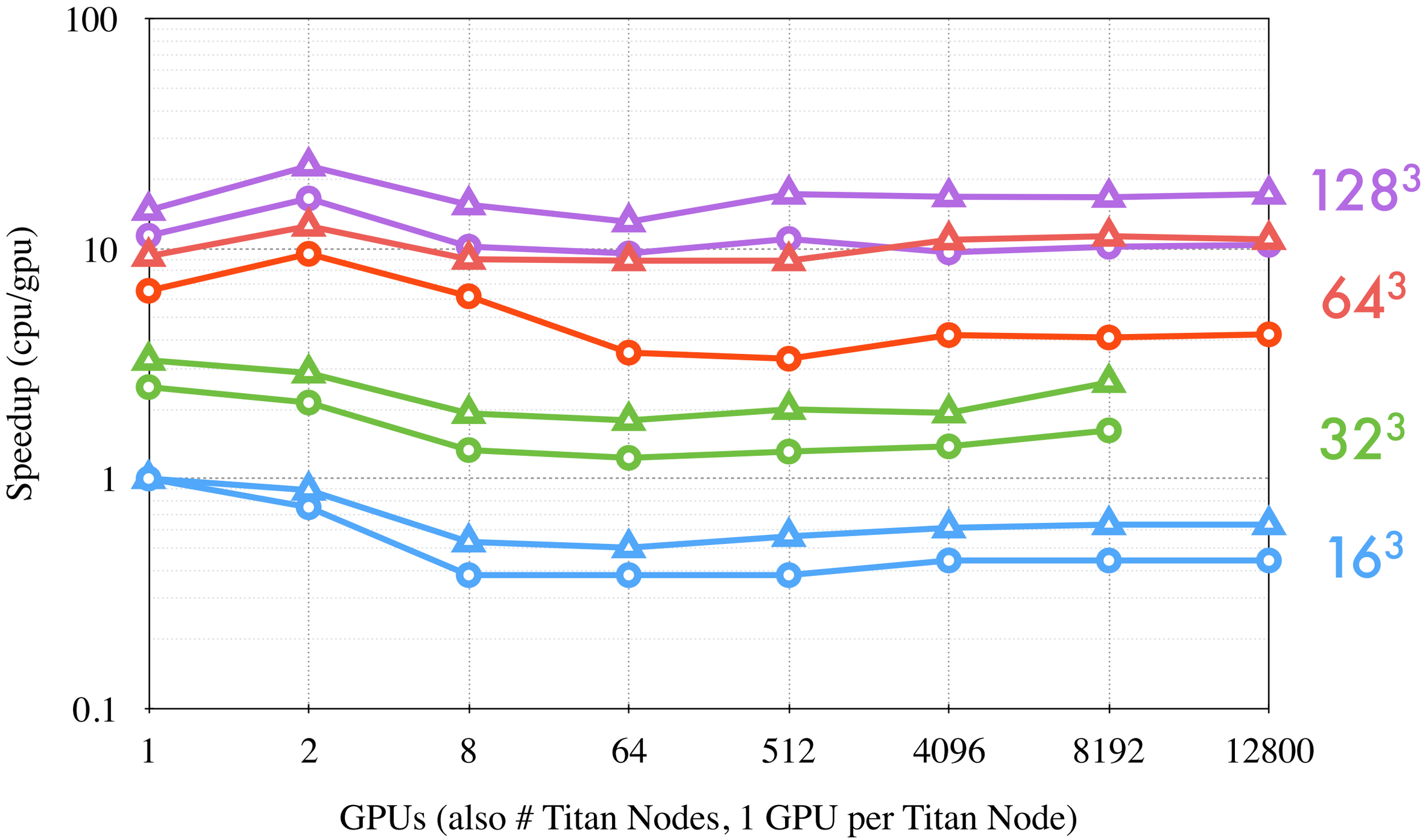
Weak Scaling - Scalar Transport, Coupled Source Terms



10 Equations

20 Equations

Speedup - Scalar Transport, Coupled Source Terms



# Q & A

http://software.crsim.utah.edu

Tony Saad: tony.saad@utah.edu
James C. Sutherland: james.sutherland@chemeng.utah.edu