A GPU-accelerated Boundary Element Method and Vortex Particle Method

Mark J. Stock, Ph.D. and Adrin Gharakhani, Sc.D.

Applied Scientific Research
Santa Ana, CA

Funding: AMRDEC SBIR Phase I Grant,
NSF grant via Teragrid, and NCRR award

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Outline

- Motivation
  - computational and numerical
- Method
  - Lagrangian Vortex Particle Methods
  - Boundary Element Method
  - GPU implementation
- Performance

Definitions

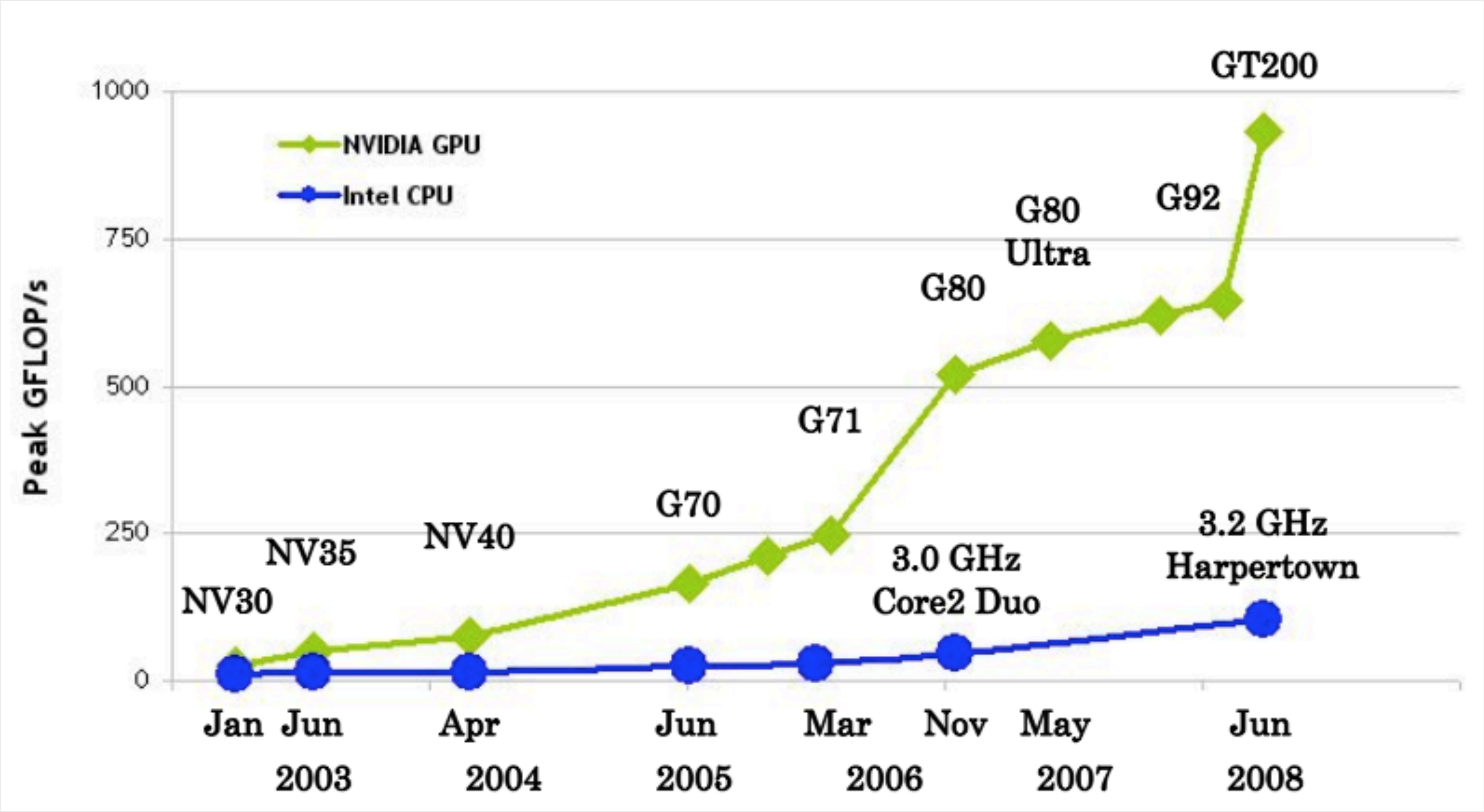
- GPU - Graphics Processing Unit
- Flop - Floating point operation
- GFlop/s - Billion Flops per second

Two Nvidia 9800GX2 GPUs
> 1 TFlop/s

[Tom's Hardware]

Motivation

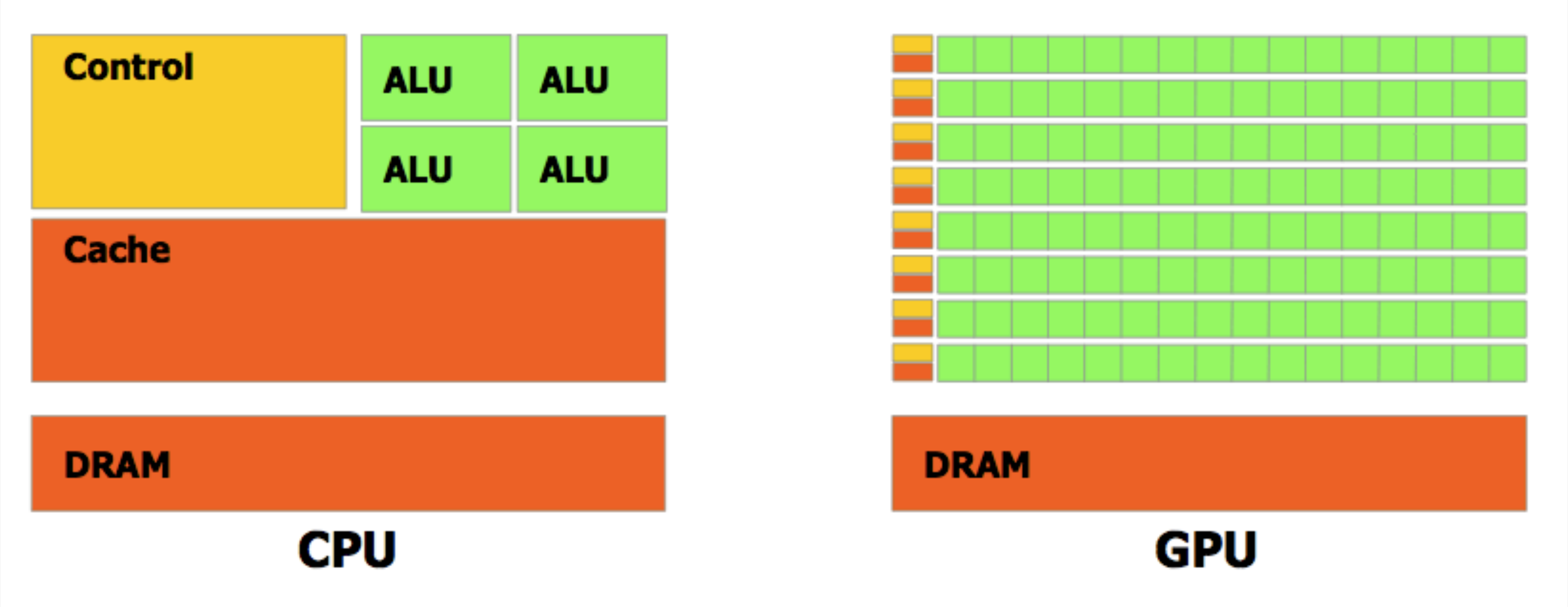
- GPU performance doubles ~1-2 years
- CPU performance doubles ~2-3 years



NVIDIA CUDA Programming Guide

Motivation

- GPU is in sweet spot for numerical computation



NVIDIA CUDA Programming Guide

Eulerian CFD

- Advantages
  - High-order methods
  - Body-fitted grids with high aspect ratio
  - Generality and history

- Disadvantages
  - Numerical diffusion
  - Gridding effort
  - Start-up
  - Far-field BCs

Lagrangian CFD

- Advantages
  - Numerical diffusion can be explicit (not limited by CFL)
  - Lower dimension grids
  - Efficient use of computational elements/far-field BCs automatic
  - Continuity preserved by construction
  - No start-up problems

- Disadvantages
  - Computationally expensive
  - Largely incompressible
  - Spatial adaptation difficult

Lagrangian vortex methods

Lagrangian vortex methods

- Navier-Stokes equations in vorticity

$$\frac{D\omega}{Dt} = \omega \cdot \nabla u + \nu \nabla^2 \omega$$

- Discretized onto Lagrangian particles

$$\bar{\omega}(\bar{x}, t) = \sum_{i=1}^{N_v} \bar{\Gamma}_i(t)\, \phi_\sigma(\bar{x} - \bar{x}_i)$$

$$\bar{\Gamma}_i = \bar{\omega}_i \, \Delta V_i$$

Lagrangian vortex methods

- **Helmholtz integral formula**

$$
\begin{aligned}
\vec{u}(\vec{x}) = \vec{U}_\infty \quad & + \quad \nabla \times \int_V \vec{\omega}(\vec{x}')\, G(\vec{x},\vec{x}')\, dV(\vec{x}') \\
& - \quad \nabla \int_V \theta(\vec{x}')\, G(\vec{x},\vec{x}')\, dV(\vec{x}') \\
& + \quad \nabla \times \int_S \big(\vec{\gamma}(\vec{x}') + \hat{n}(\vec{x}') \times \vec{u}(\vec{x}')\big) G(\vec{x},\vec{x}')\, dS(\vec{x}') \\
& - \quad \nabla \int_S \big(\hat{n}(\vec{x}') \cdot \vec{u}(\vec{x}')\big) G(\vec{x},\vec{x}')\, dS(\vec{x}')
\end{aligned}
$$

Lagrangian vortex methods

- **Inviscid flow**

$$\frac{\partial \bar{x}_i}{\partial t} = \bar{u}_i$$

$$\frac{\partial \bar{\Gamma}_i}{\partial t} = \bar{\Gamma}_i \cdot \nabla \bar{u}_i$$

- **Viscous diffusion via VRM (Shankar and van Dommelen, 1996)**

$$\frac{\partial \bar{\Gamma}_i}{\partial t} = \nu \nabla^2 \bar{\Gamma}_i$$

- **Anisotropic SGS dissipation (Cottet *et al*. 1996, 2003)**

Boundary element method

- **BEM solves for unknown surface vortex sheet strength**

$$\frac{1}{2}\vec{\gamma}(\vec{x}) \times \hat{n}(\vec{x}) + \int_{\partial\Omega} \vec{\gamma}(\vec{x}') \times K(\vec{x} - \vec{x}')\, d\vec{x}' = \vec{U}_{slip}(\vec{x})$$

- **Right-hand-side vector is**

$$\vec{U}_{slip}(\vec{x}) = \vec{U}_{\infty}(\vec{x}) - \int_{\Omega} \vec{\omega}(\vec{x}') \times K(\vec{x} - \vec{x}')\, d\vec{x}'$$

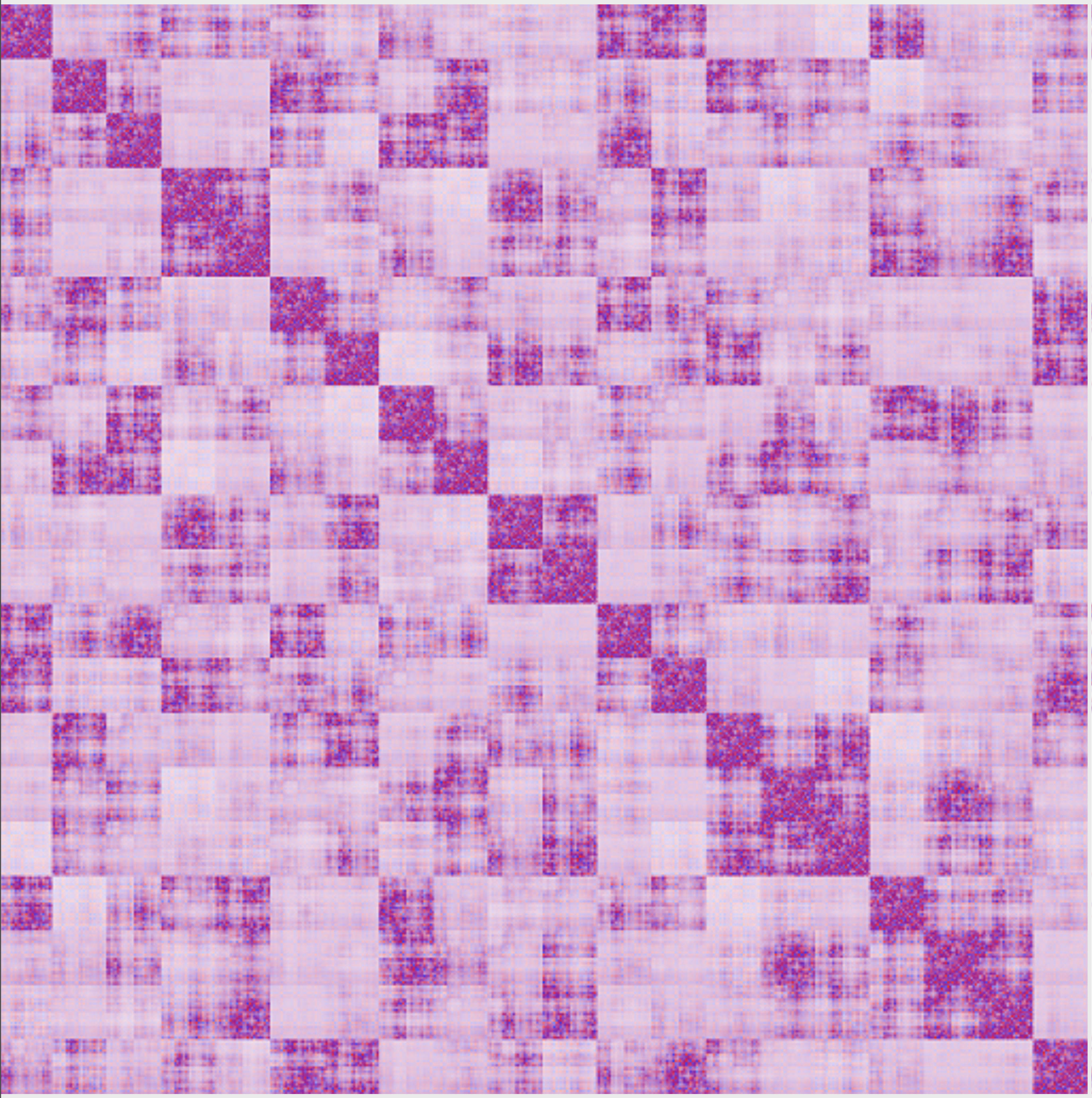
Algorithm - Time step

1) **Solve BEM for unknown surface vortex strengths γ**
2) **Compute u and ∇u on all particles using VPM**
3) **Advect particles and apply stretch**
4) **Repeat (1) - (3) for 2nd order advection**
5) **Split elongated particles, merge close particles**
6) **Create new particles above panels with $\Gamma = \gamma$ * (area)**
7) **Use VRM to diffuse vorticity**

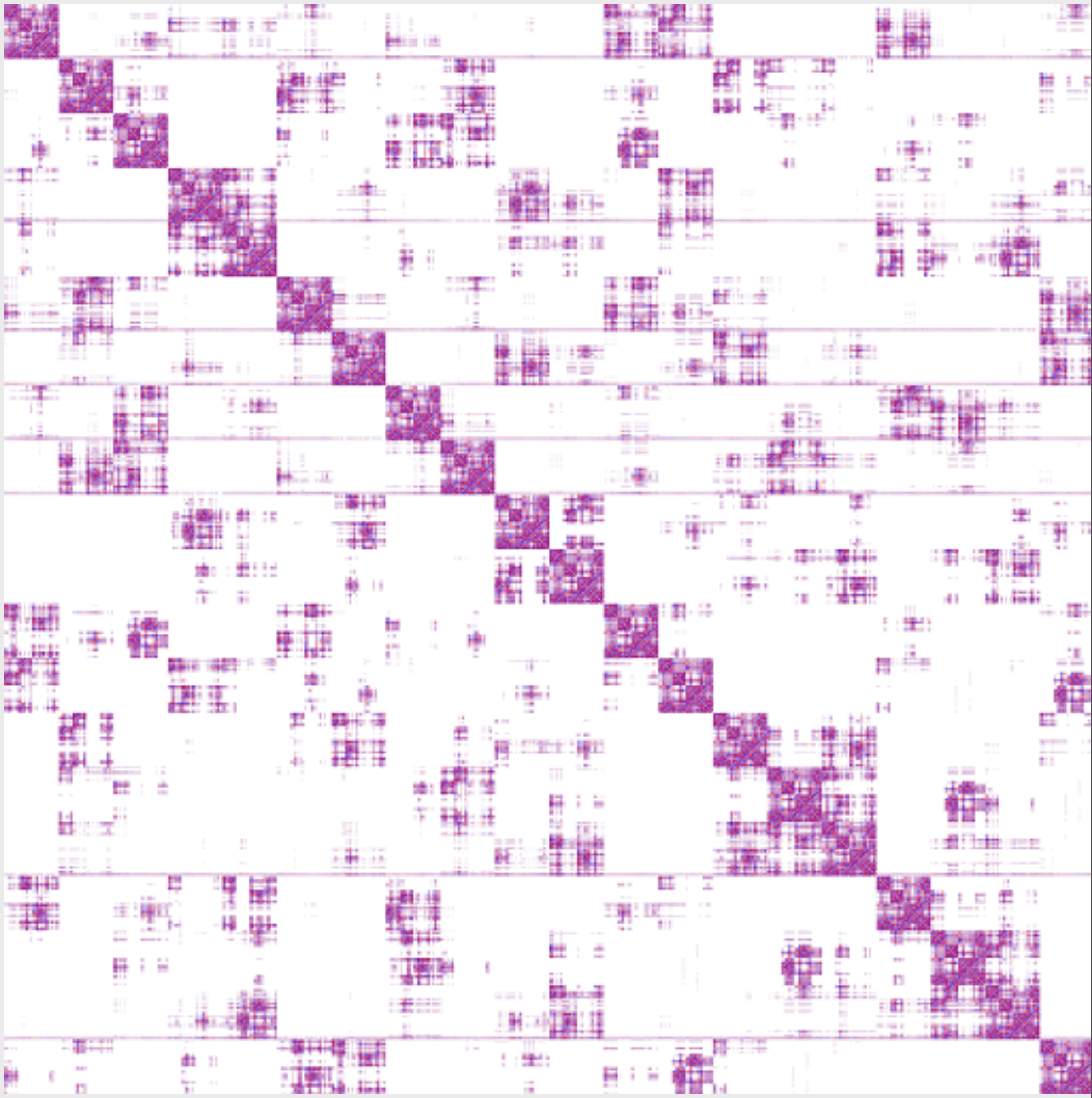
Algorithm - BEM

1) **Solve BEM for unknown surface vortex strengths γ**
   a) **Transform surface elements to position**
   b) **Re-make tree structure if necessary**
   c) **Find interaction lists for each leaf box**
   d) ***Create and save sparse component of influence matrix - GPU***
   e) **Remove any particles close to boundary**
   f) ***Compute RHS using VPM - GPU***
   g) **Iterate using GMRES until converged:**
      i) **Using previous solution, compute multipole moments**
      ii) ***Compute far-field influence using multipoles - GPU***
      iii) **Compute near-field using sparse matrix multiply**

Algorithm - BEM



Direct influence matrix



Sparse influence matrix

Algorithm - VPM

2) **Compute u and ∇u on all particles**
   a) **Distribute particles across all processors (weighted ORB)**
   b) **Build tree structure (VAM-Split k-d trees)**
   c) **Re-order particles for coherency**
   d) **Compute multipole moments (real spherical, cast to cart.)**
   e) ***Compute far-field influence using multipoles - GPU***
   f) ***Compute near-field using Biot-Savart summations - GPU***
   g) **Add free stream velocity**

Algorithm - GPU routine

1. **Fortran calls C host code**
2. **Split into threads, one for each GPU**
   a. **Loop over kernel blocks if problem is too large**
      i. **Pack arrays if necessary**
      ii. **Copy data to GPU**
      iii. **Define problem size (thread blocks, threads per block)**
      iv. ***Call GPU kernel***
      v. **Pull results from GPU to main memory, free GPU memory**
3. **Return to Fortran code**

Algorithm - GPU kernel

1. **Enter CUDA device kernel, one thread per target**
2. **One thread per block reads source box pointers to register**
3. **Use thread and block ID to determine target index**
4. **Load target position into register, zero target u and ∇u**
5. **Loop over all source boxes**
   a. **Each thread loads one source element's position and strength into thread block's shared memory**
   b. **Loop over the array of source elements**
      i. **Calculate the element-element influence**
      ii. **Accumulate u and ∇u**
6. **Write the result back to GPU memory**

Performance

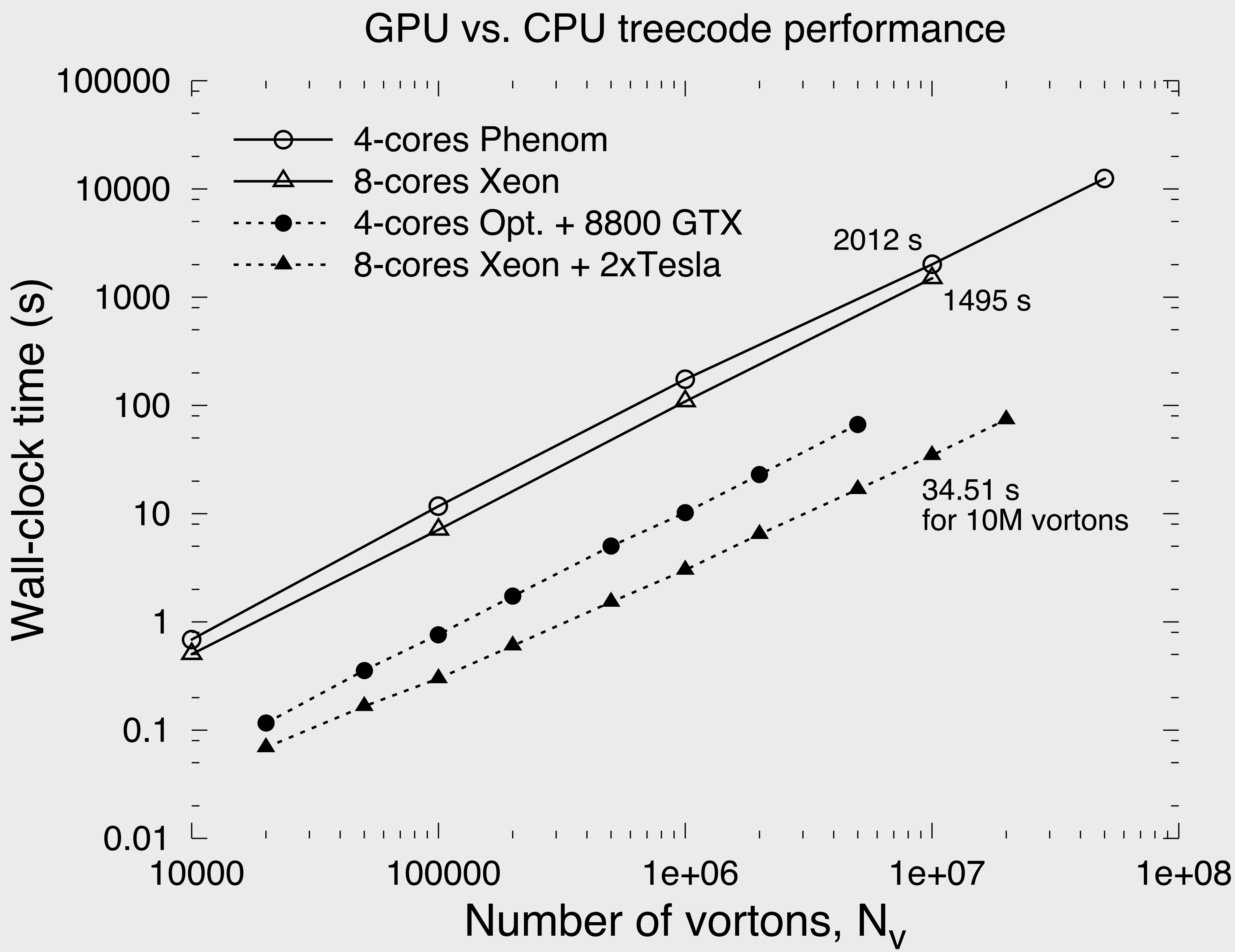
VPM performance

- Test problem is of random vortons in a cube
  - Cube size set as if uniform particles had proper overlap
  - random locations throughout the cube
  - random strengths
- Solve for and save velocities and 9-component vel. gradients
  - to estimated RMS error of 2.e-4 (max <2.e-3)
  - 7 (GPU) or 9 (CPU) levels of multipole moments
  - 128 or 196 (GPU) or 64 (CPU) vortons per tree leaf node

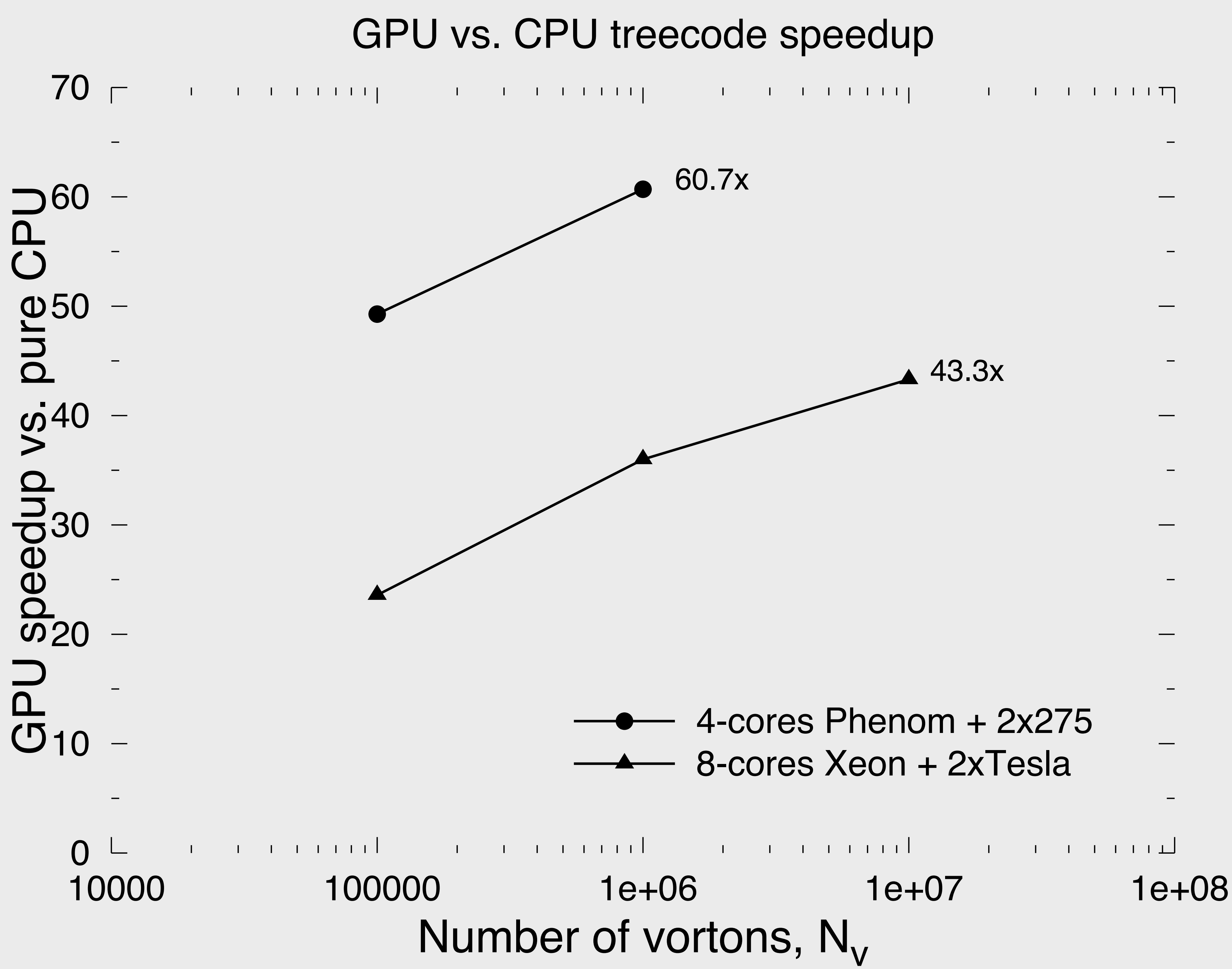
VPM performance

- Gumerov and Duraiswami (2008)
  - Ported FMM to GPU
  - 72x speedup from single-core to 8800 GTX for singular particles
  - Used such large leaf boxes that scaling was only $N^{1.15}$
- Yokota *et al*. (2009)
  - Ported FMM to cluster of GPUs
  - 60x speedup from single-core to 8800 GTS for Gaussian parts
  - Again, scaling did not reach O(N)
  - 60% parallel efficiency on 32 nodes for 10M particles

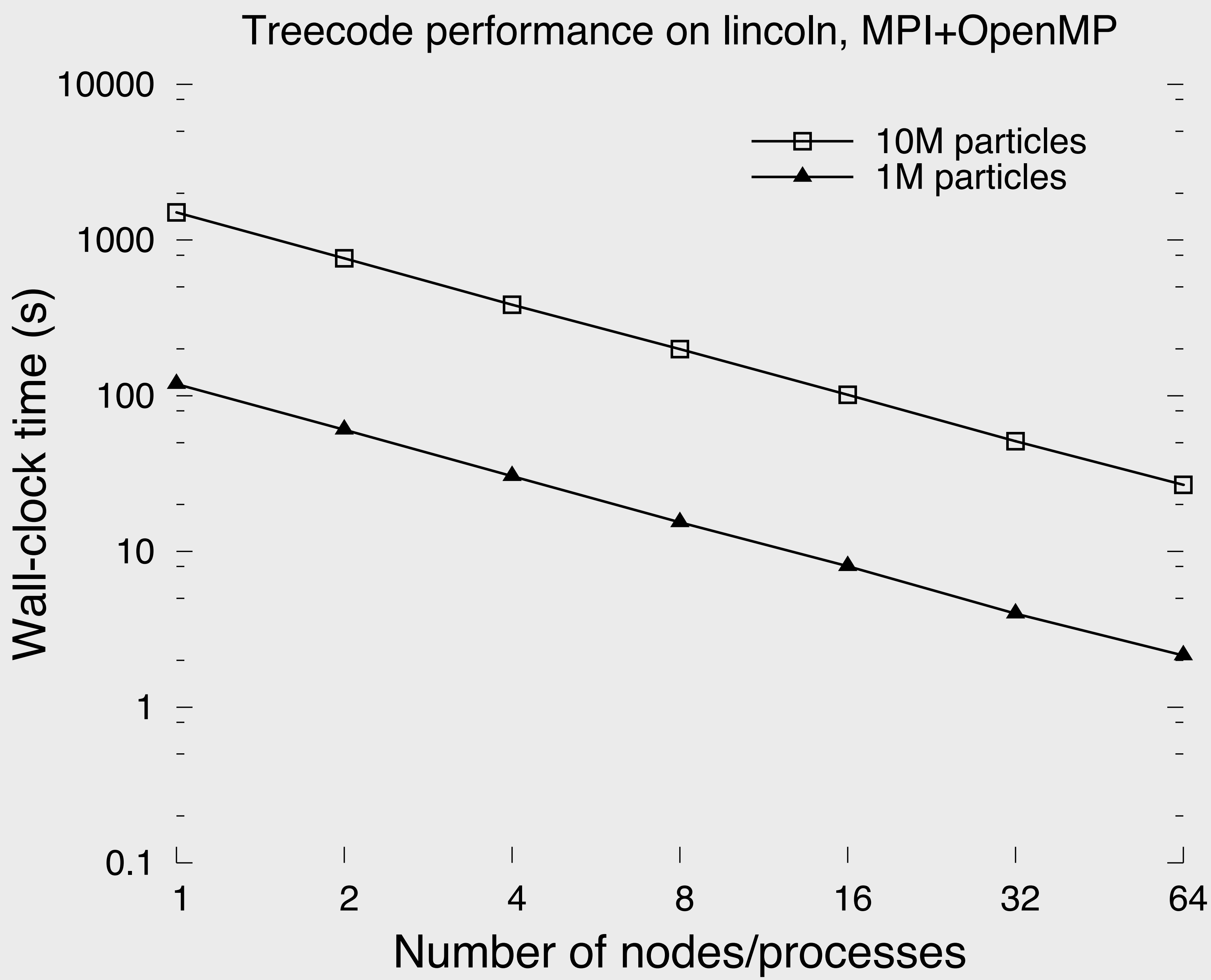
VPM performance



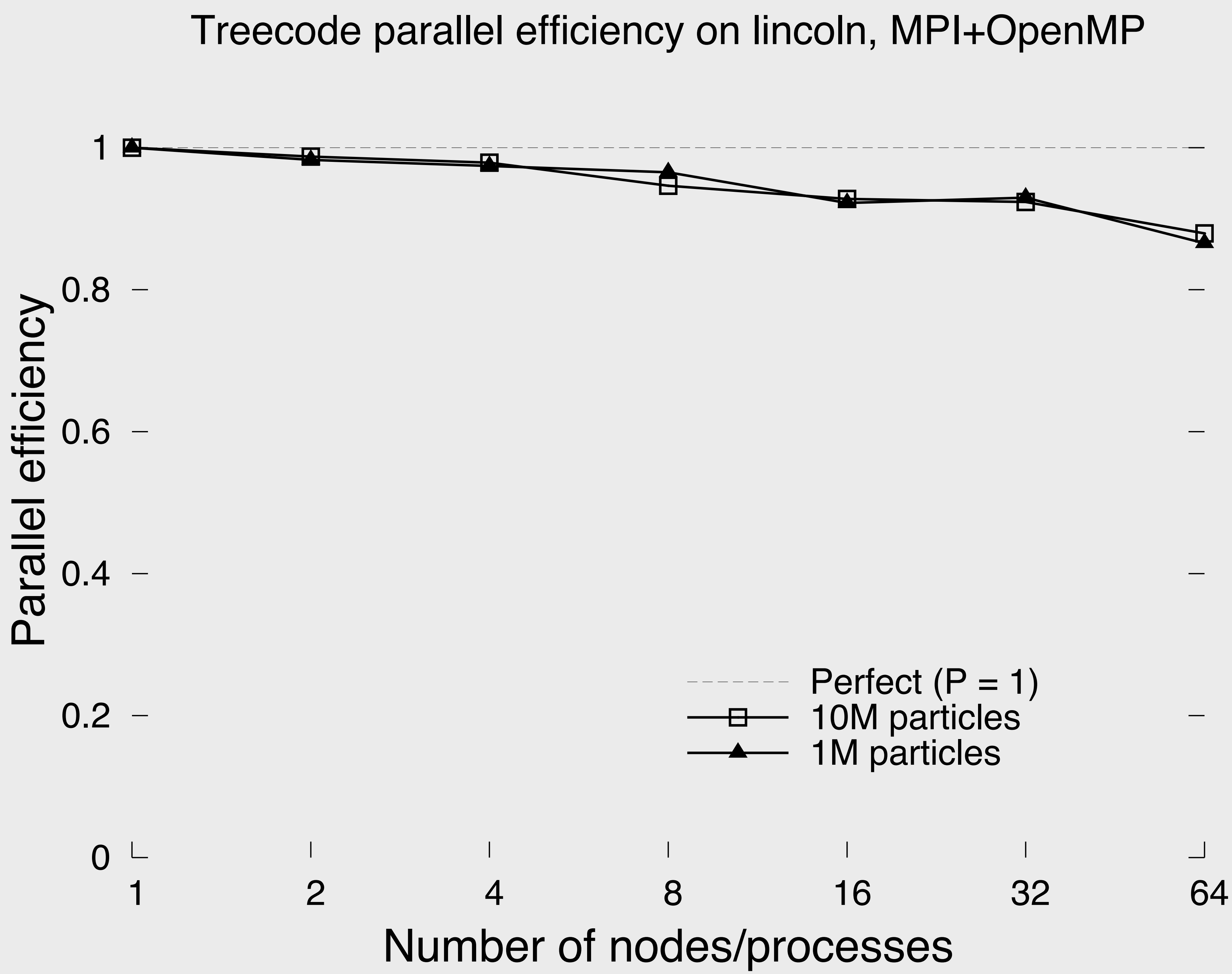
VPM performance



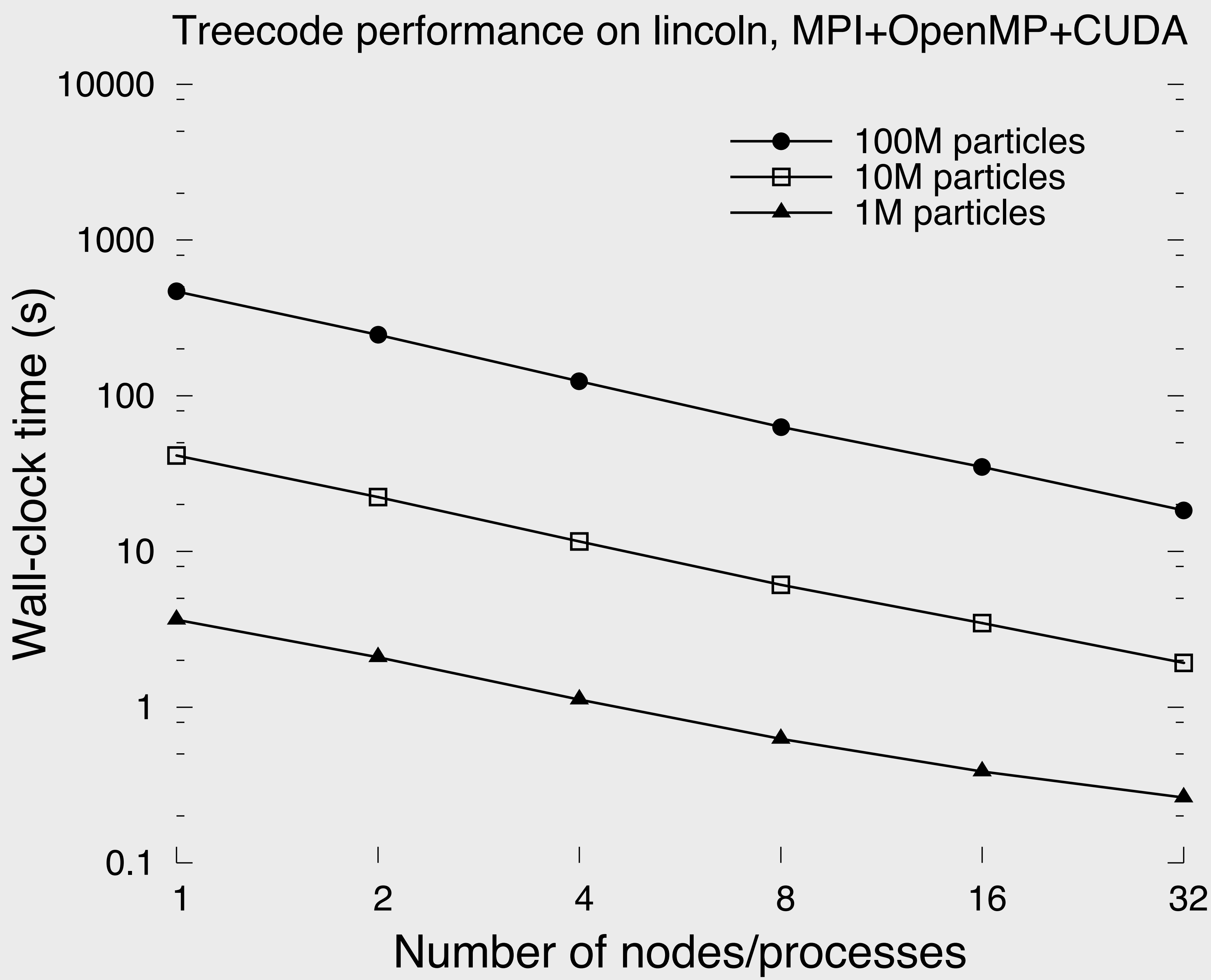
Parallel VPM performance



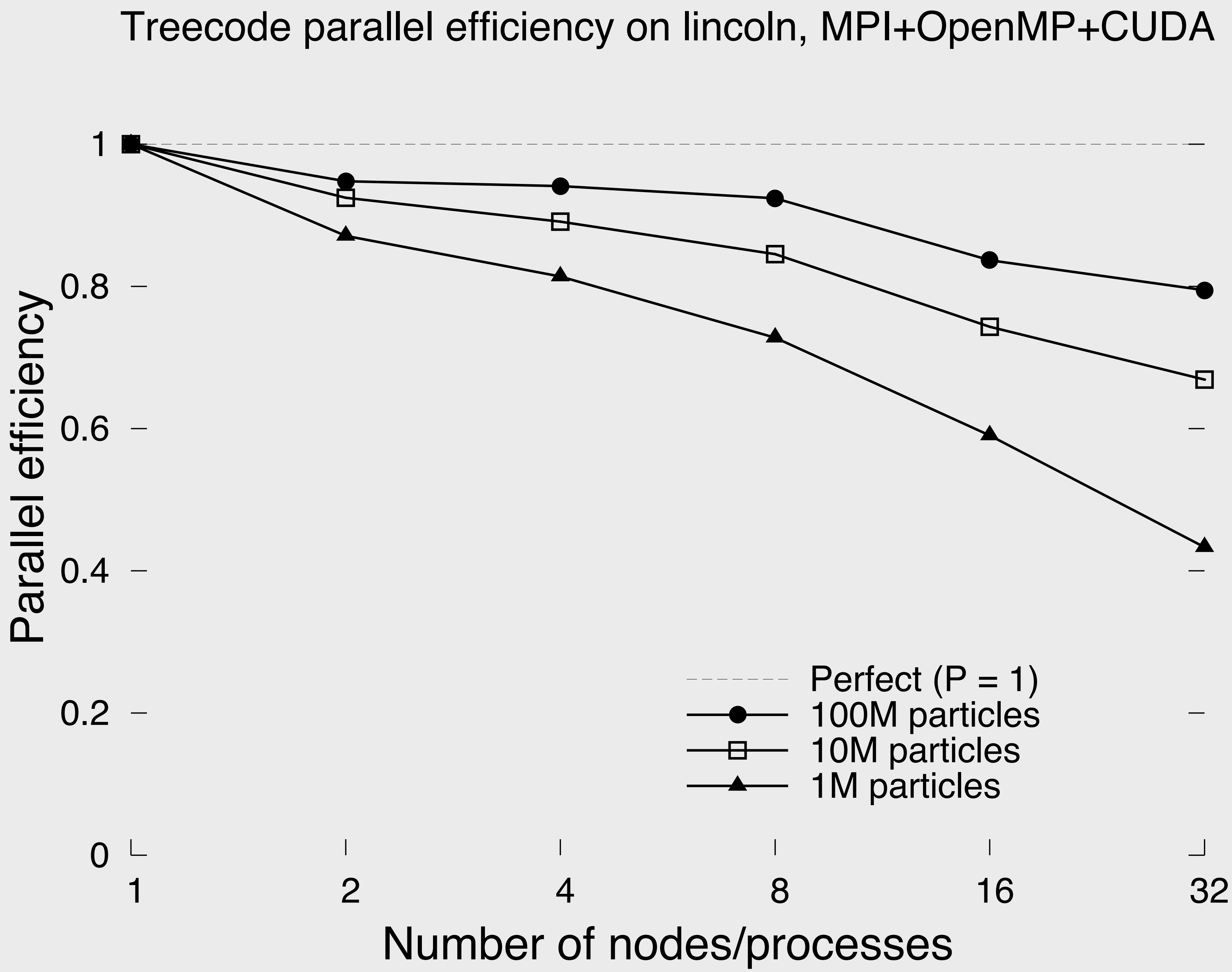
Parallel VPM performance



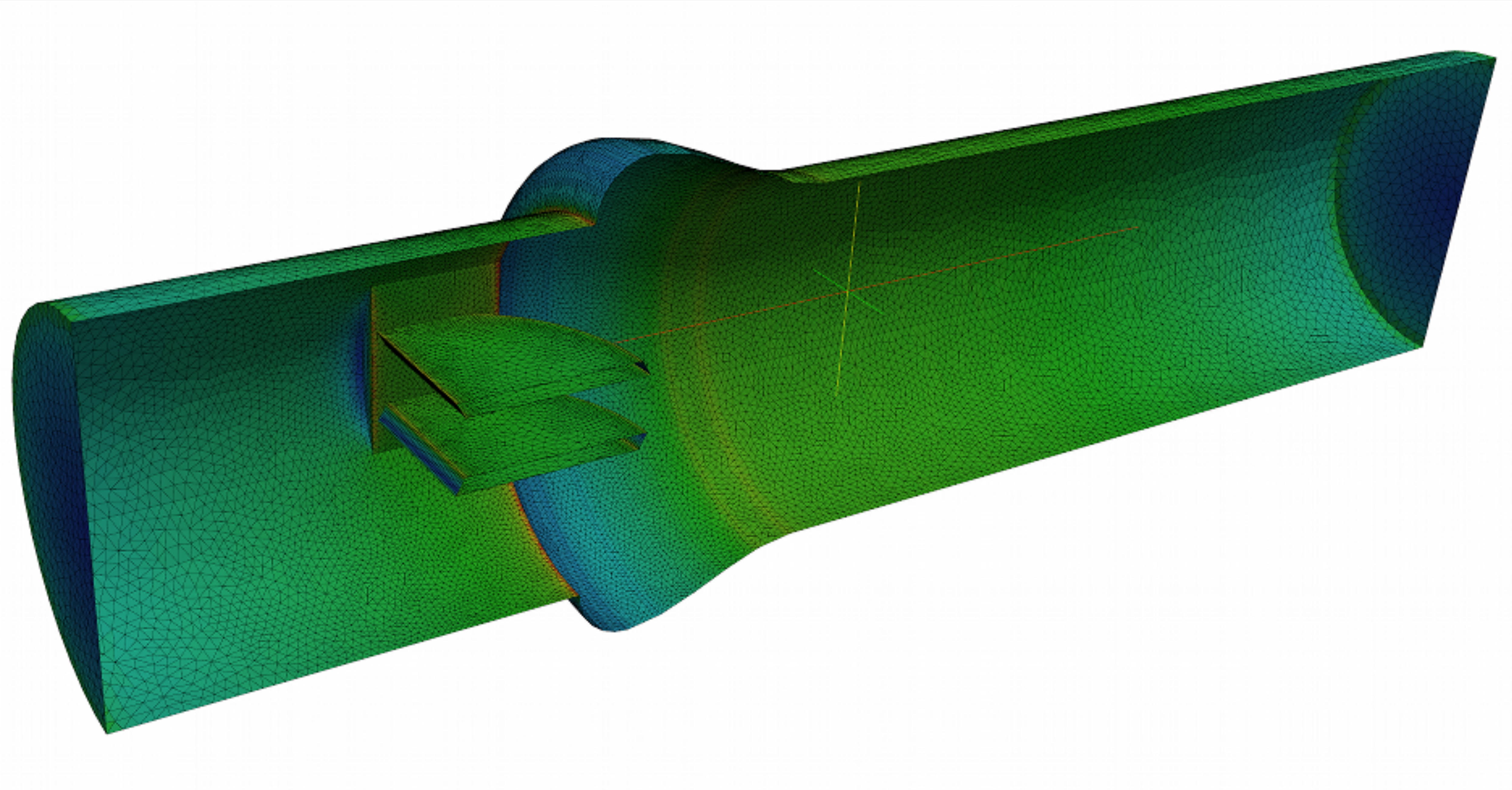
Parallel VPM performance



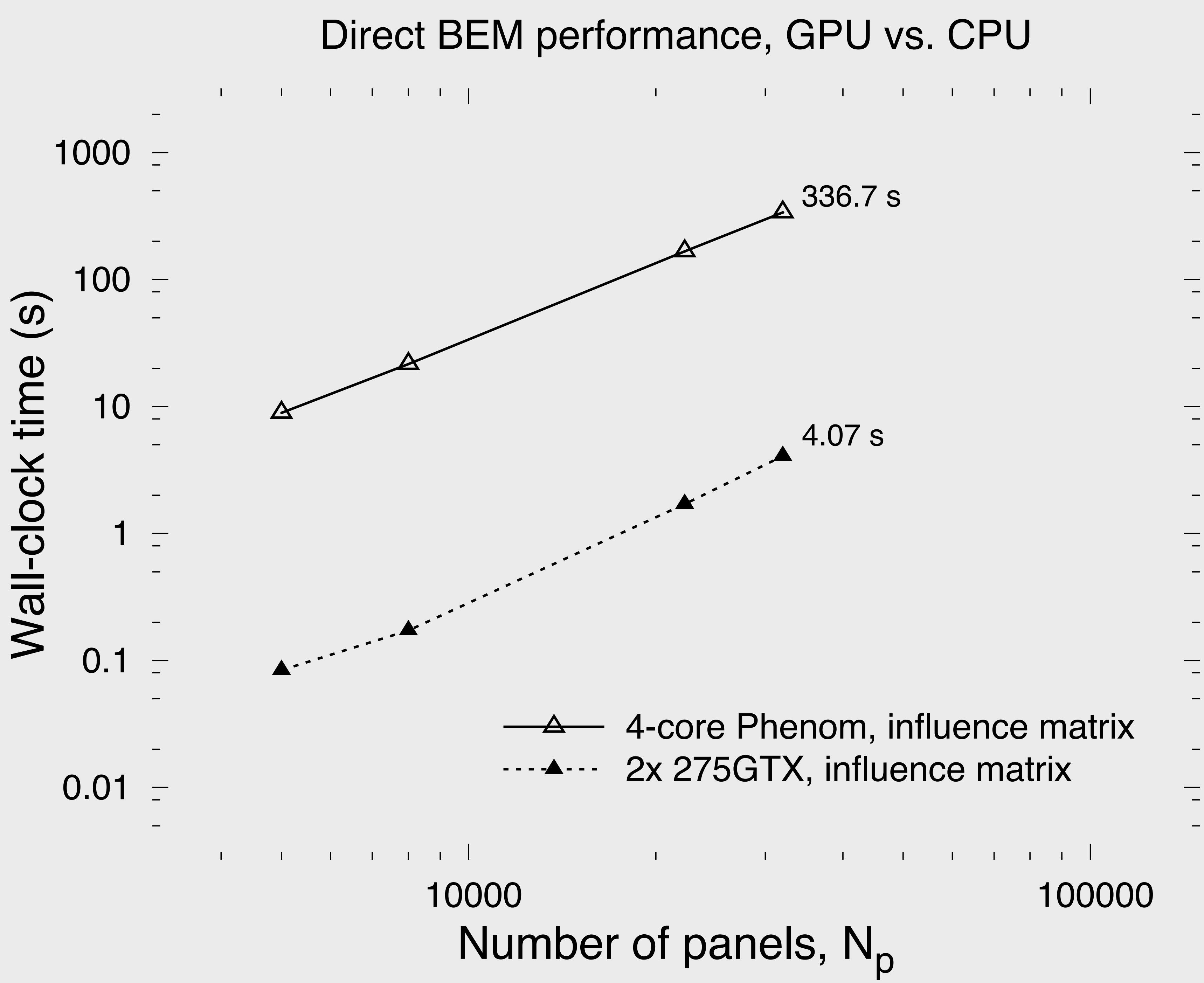
Parallel VPM performance



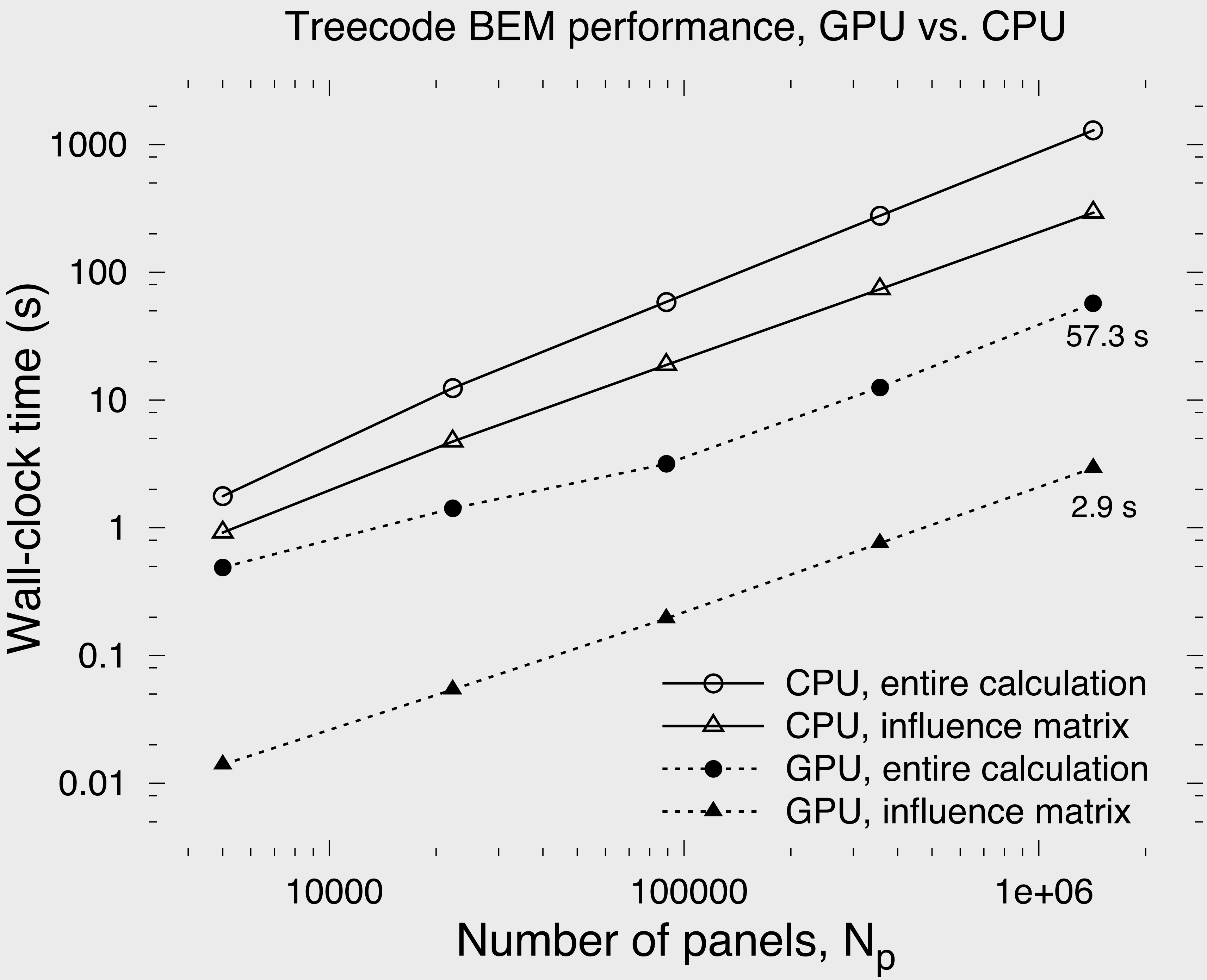
BEM performance



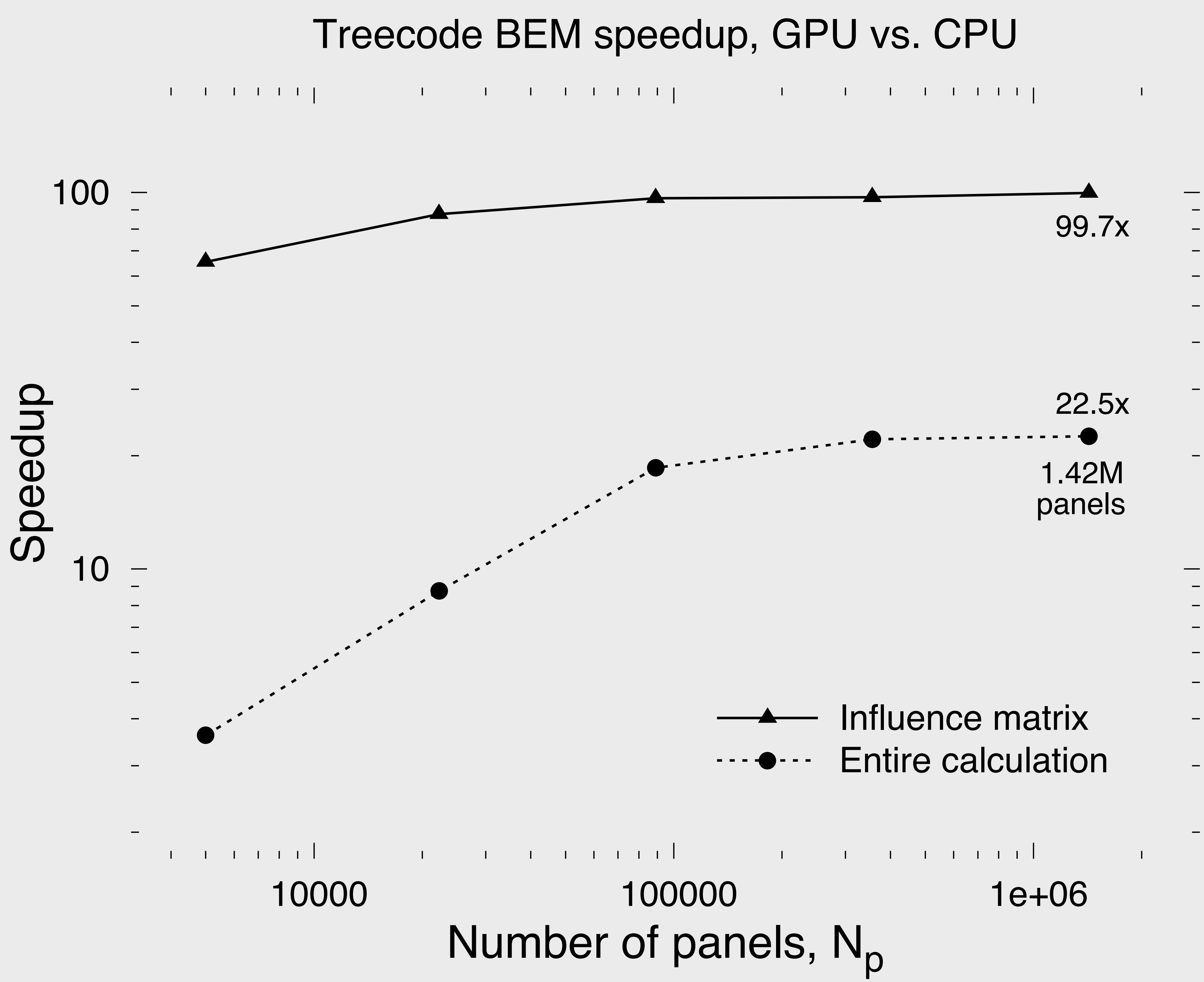
BEM performance



BEM performance



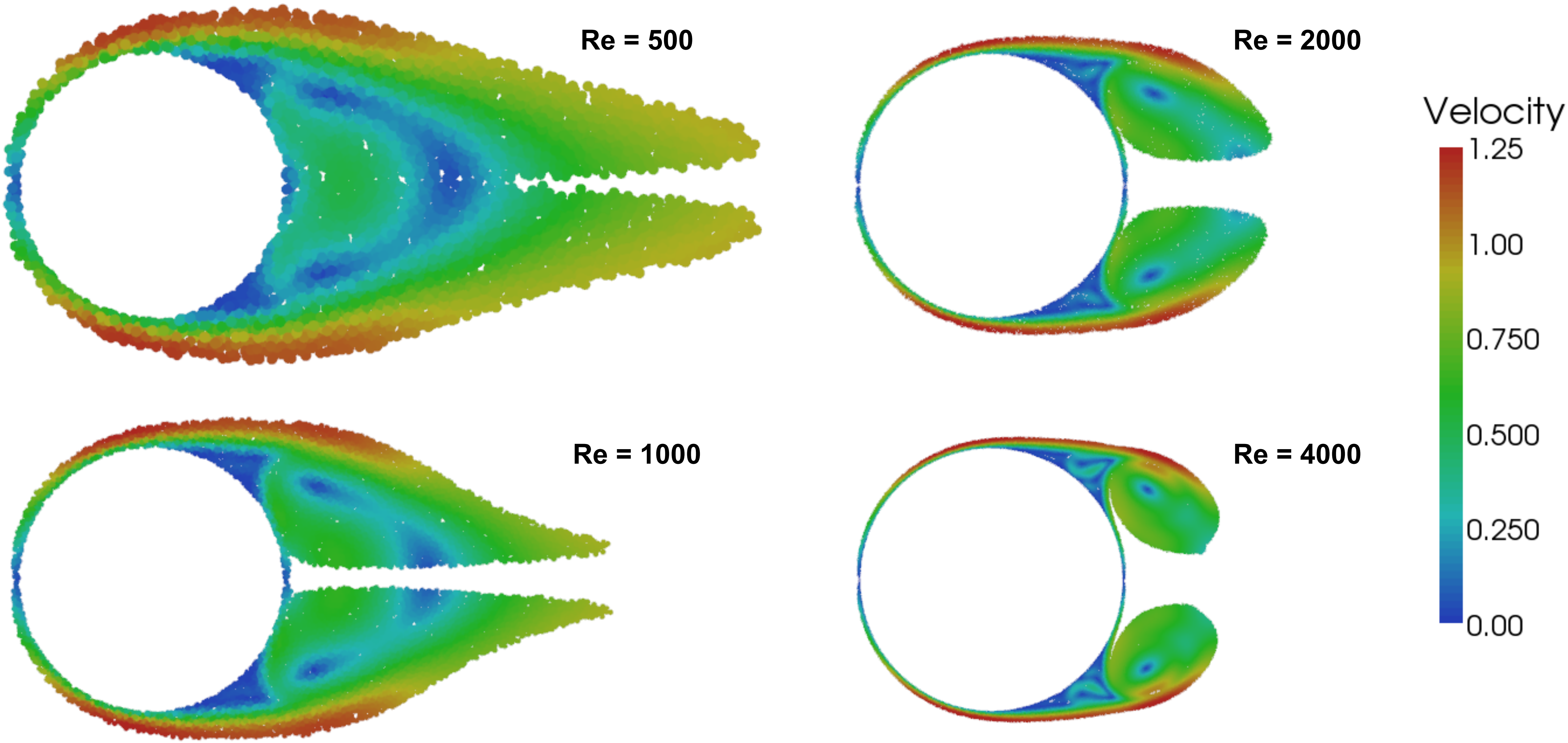
BEM performance



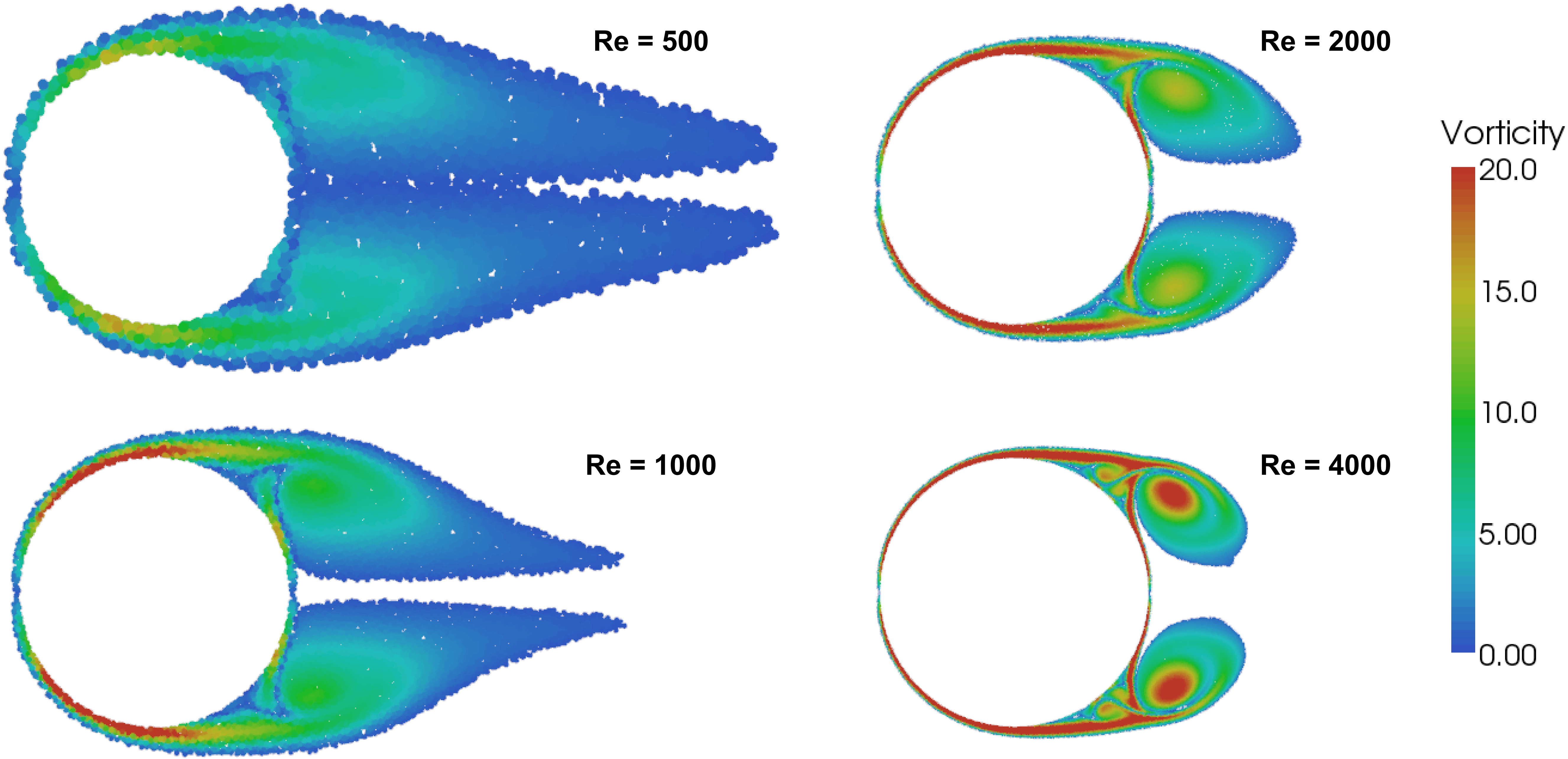
Dynamic simulations

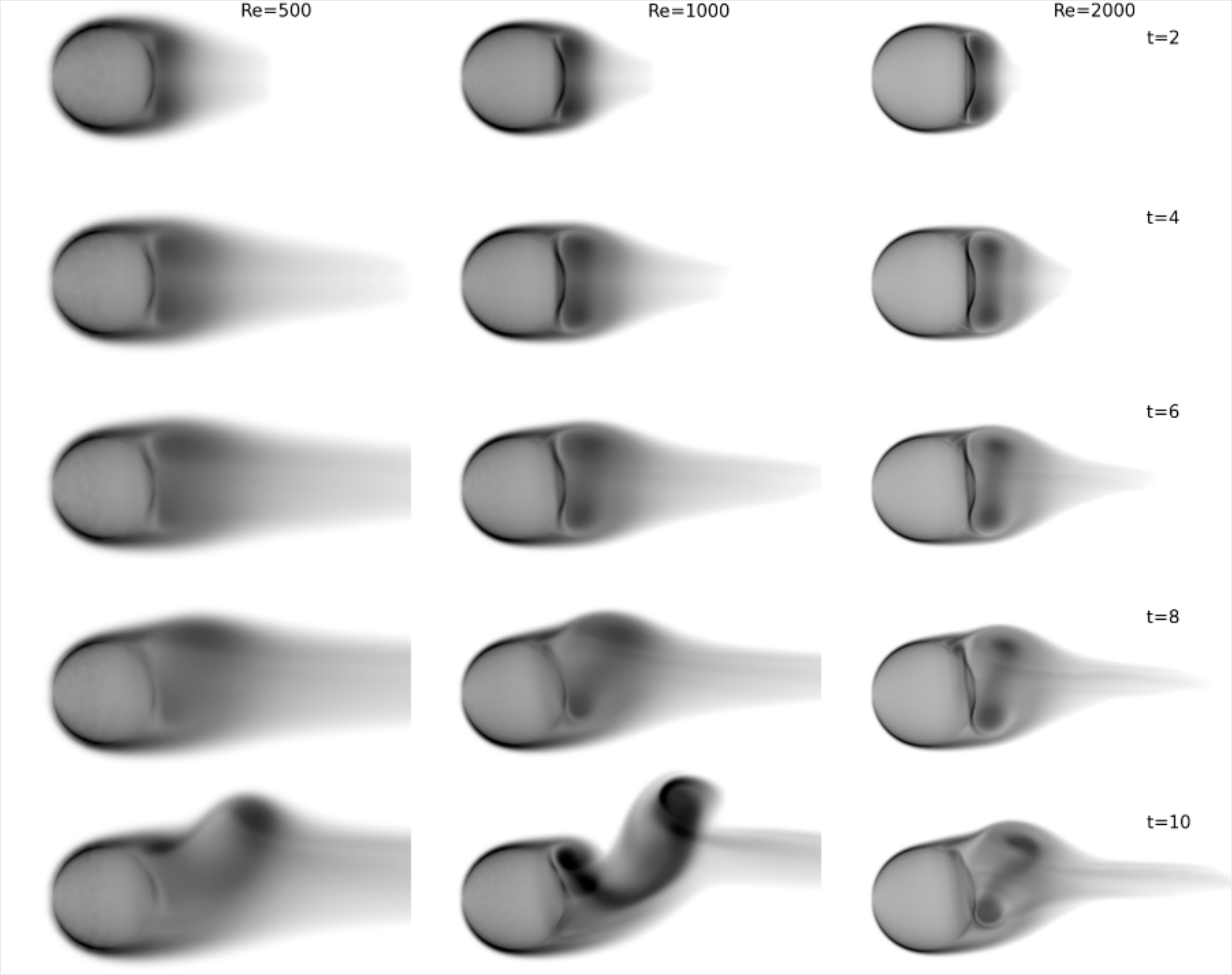
- Spheres at Re = 500, 1000, 2000, 4000
- 4-Bladed isolated rotor in forward flight

Sphere flow at various Re_D



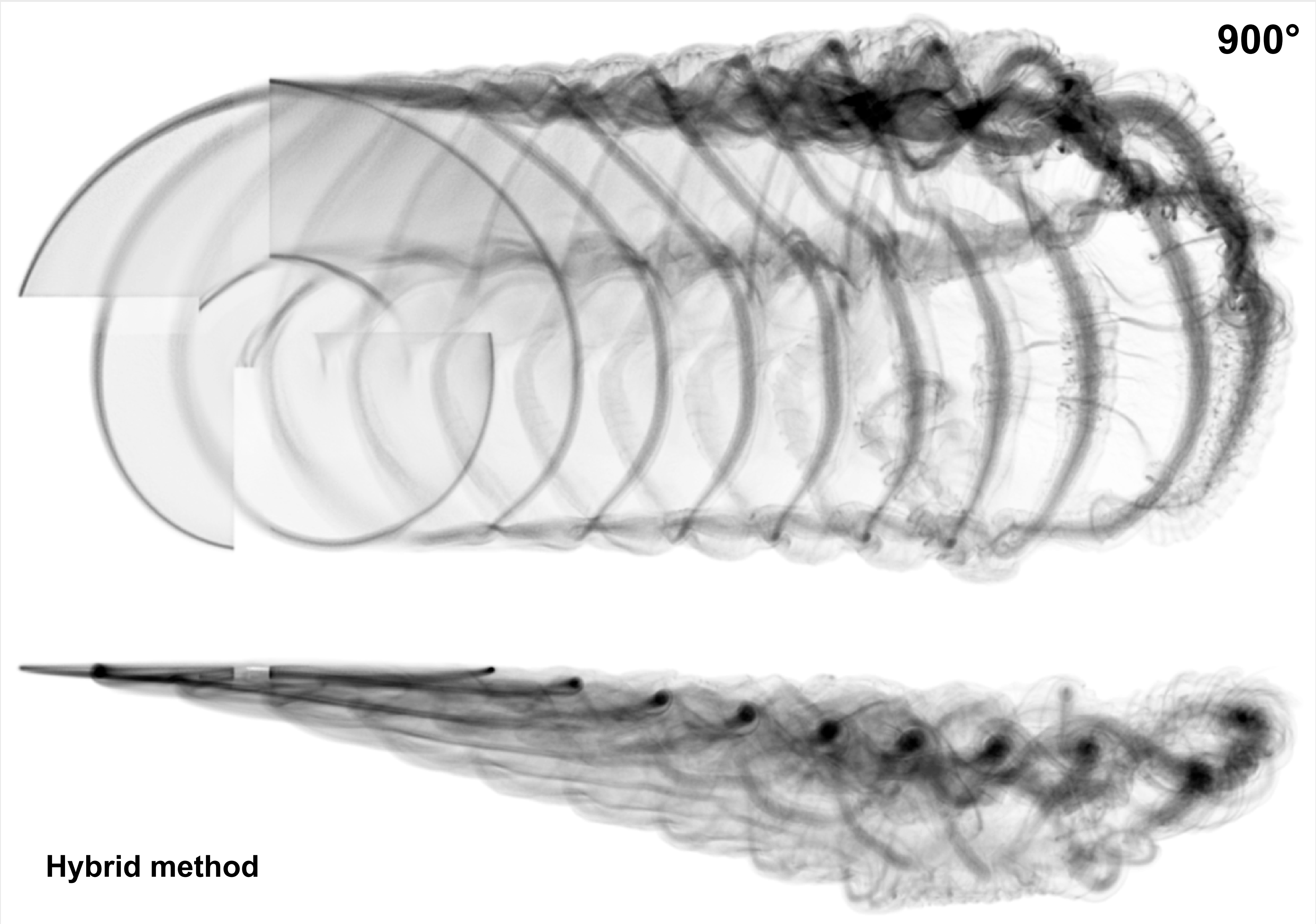
Sphere flow at various Re_D



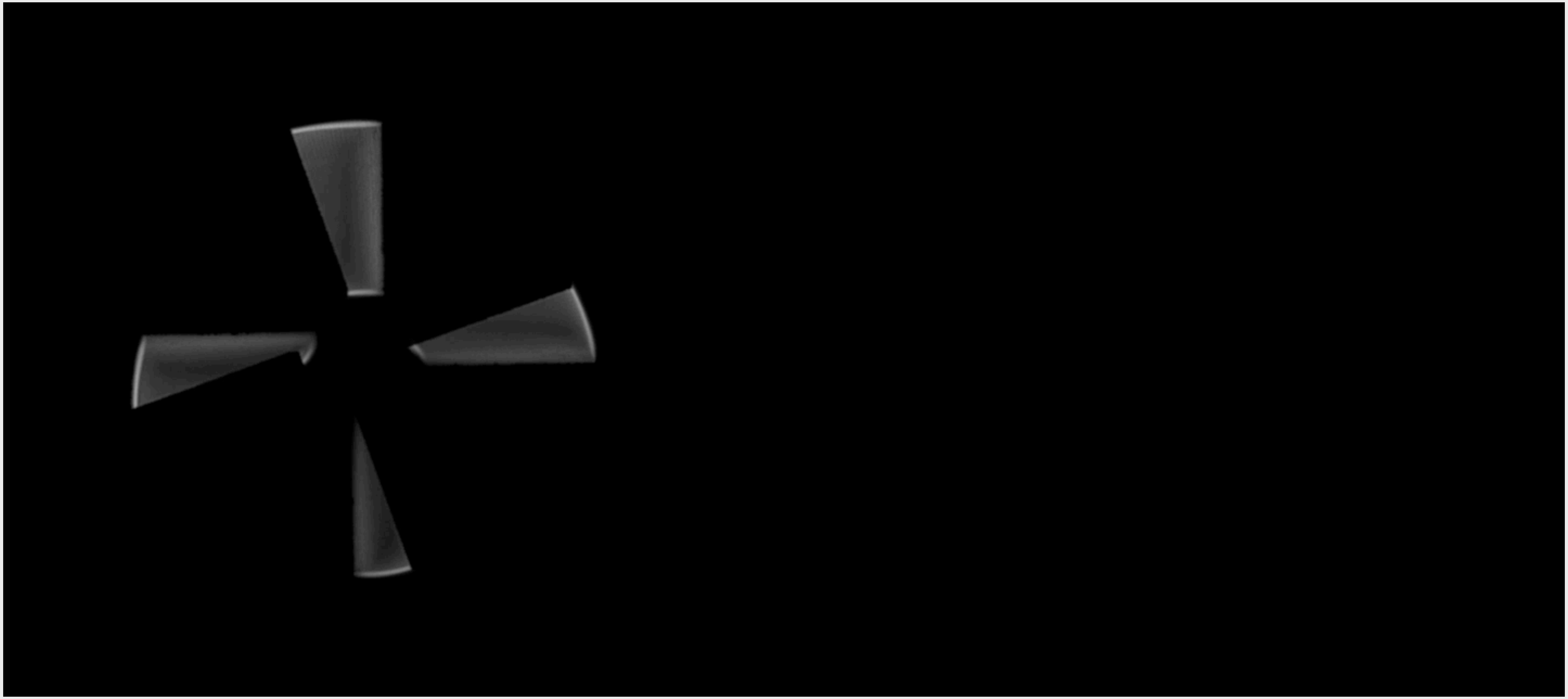


Re=500
Re=1000
Re=2000
t=2
t=4
t=6
t=8
t=10

4-Bladed isolated rotor



4-Bladed isolated rotor



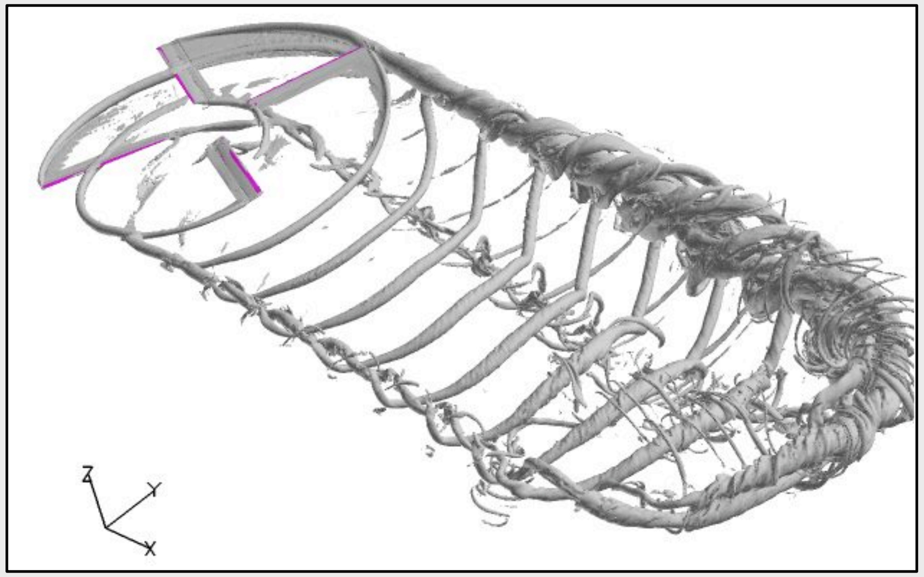
4-Bladed isolated rotor

Conclusions

- First GPU-accelerated *treecode* BEM
- *Treecode* BEM shows *20.6x* speedup from quad-core to single GPU.
- 100M vortex particles solved in 18.4s on 64 GPUs with 80% parallel efficiency
- *43x* speedup for treecode Biot-Savart solver vs. *8-core* CPU

Thank you

- AIAA
- AMRDEC, NSF, NCRR
- You, for your attention



- Please see our other paper: “Modeling Rotor Wakes with a Hybrid OVERFLOW-Vortex Method on a GPU cluster” AIAA-2010-4553

`http://www.applied-scientific.com/`