Toward efficient GPU-accelerated *N*-body simulations

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

Applied Scientific Research
Santa Ana, CA

Funding: NASA SBIR Phase I Grant
and NCRR SBIR Phase II Grant

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Definitions

- GPU - Graphics Processing Unit
- GPGPU - General-Purpose programming for GPUs
- Flop - Floating point operation
- GFlop/s - Billion Flops per second

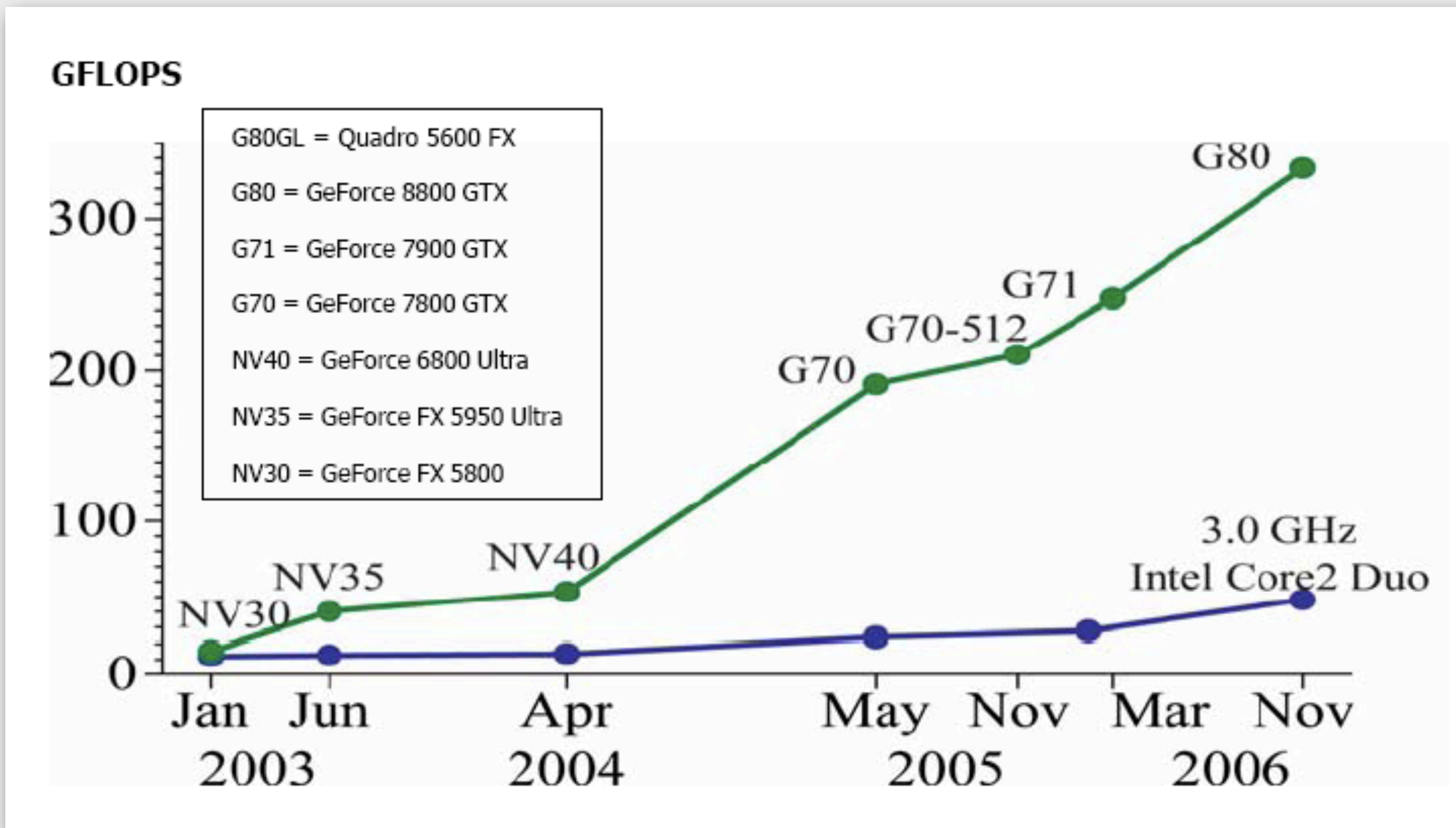
Two Nvidia 9800GX2 GPUs
> 1 TFlop/s

[Tom's Hardware]

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Motivation

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

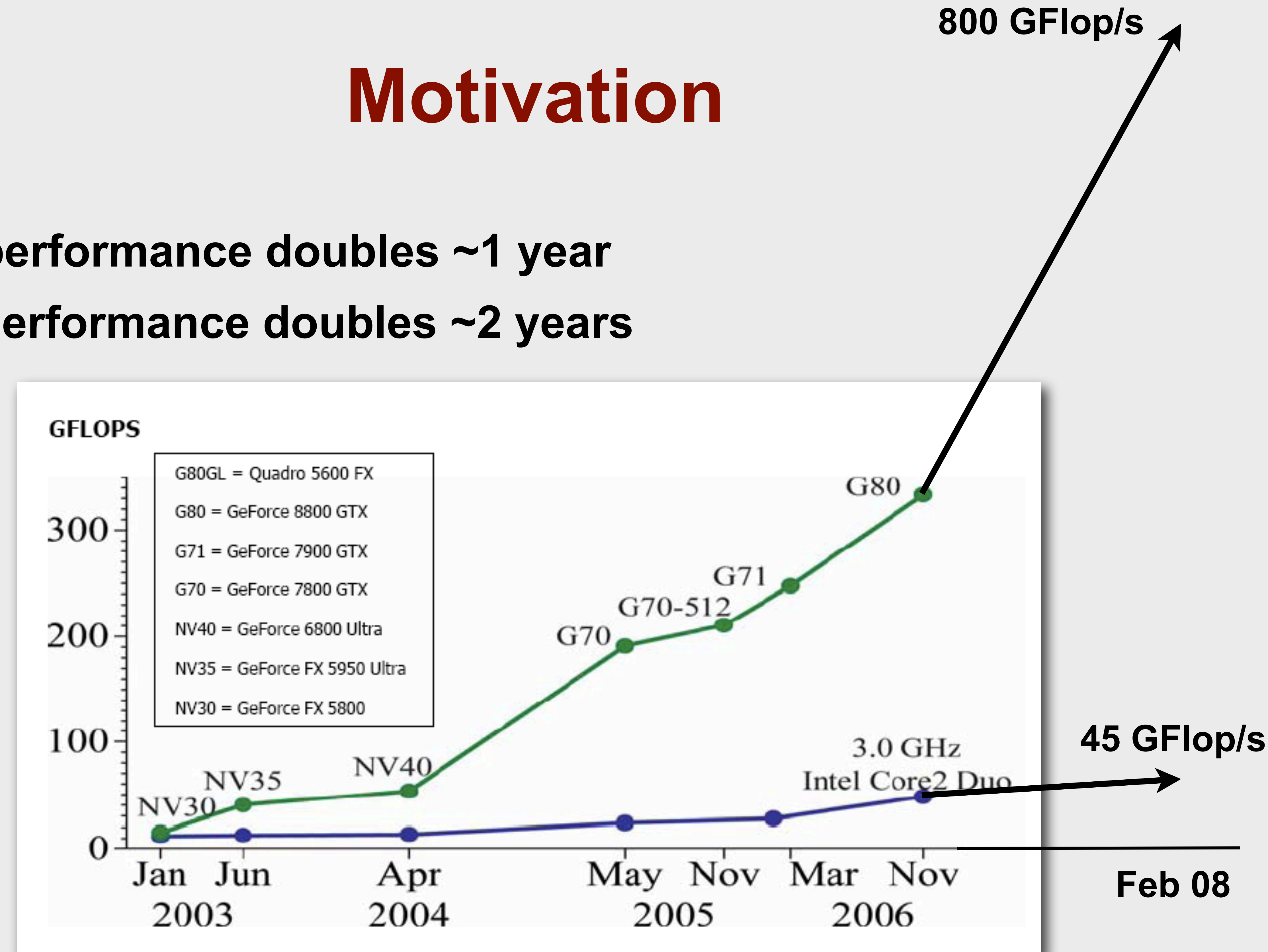


NVIDIA CUDA Programming Guide

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Motivation

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



NVIDIA CUDA Programming Guide

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Lagrangian vortex methods

- **Navier-Stokes equations in vorticity**

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

- **Discretized onto Lagrangian particles**

$$\begin{aligned} \bar{\omega}(\bar{x}, t) &= \sum_{i=1}^{N_v} \bar{\Gamma}_i(t)\, \phi_\sigma(\bar{x} - \bar{x}_i) \\ \bar{\Gamma}_i &= \bar{\omega}_i\, \Delta V_i \end{aligned}$$

Why vortex methods?

- *Grid-free* in the fluid domain (only mesh surfaces)
- *Not limited* by CFL convective instability (time steps 10x)
- Computational elements needed only where vorticity present
- *Free* from numerical diffusion (must be explicitly added)
- *Continuity conserved* by construction

CPU-direct method

- **Perform N^2 Biot-Savart evaluations**

$$
\begin{aligned}
\bar{u}_i\left(\bar{x}_i\right) &= \sum_{j=1}^{N_v} K_\sigma\left(\bar{x}_j - \bar{x}_i\right) \times \bar{\Gamma}_j + U_\infty \\
K_\sigma(\bar{x}) &= K(\bar{x}) \int_0^{|\bar{x}|/\sigma} 4\pi\, g(r)\, r^2 dr \\
K(\bar{x}) &= -\frac{\bar{x}}{4\pi|\bar{x}^3|} \\
g(r) &= (3/4\pi)\exp\left(-r^3\right)
\end{aligned}
$$

- **3-component u and 9-component ∇u is 72 Flops**

CPU-direct method

- Perform N^2 Biot-Savart evaluations

$$\bar{u}_i\left(\bar{x}_i\right) = \sum_{j=1}^{N_v} K_\sigma\left(\bar{x}_j - \bar{x}_i\right) \times \bar{\Gamma}_j + U_\infty$$

CPU-direct method

- **Perform N^2 Biot-Savart evaluations**

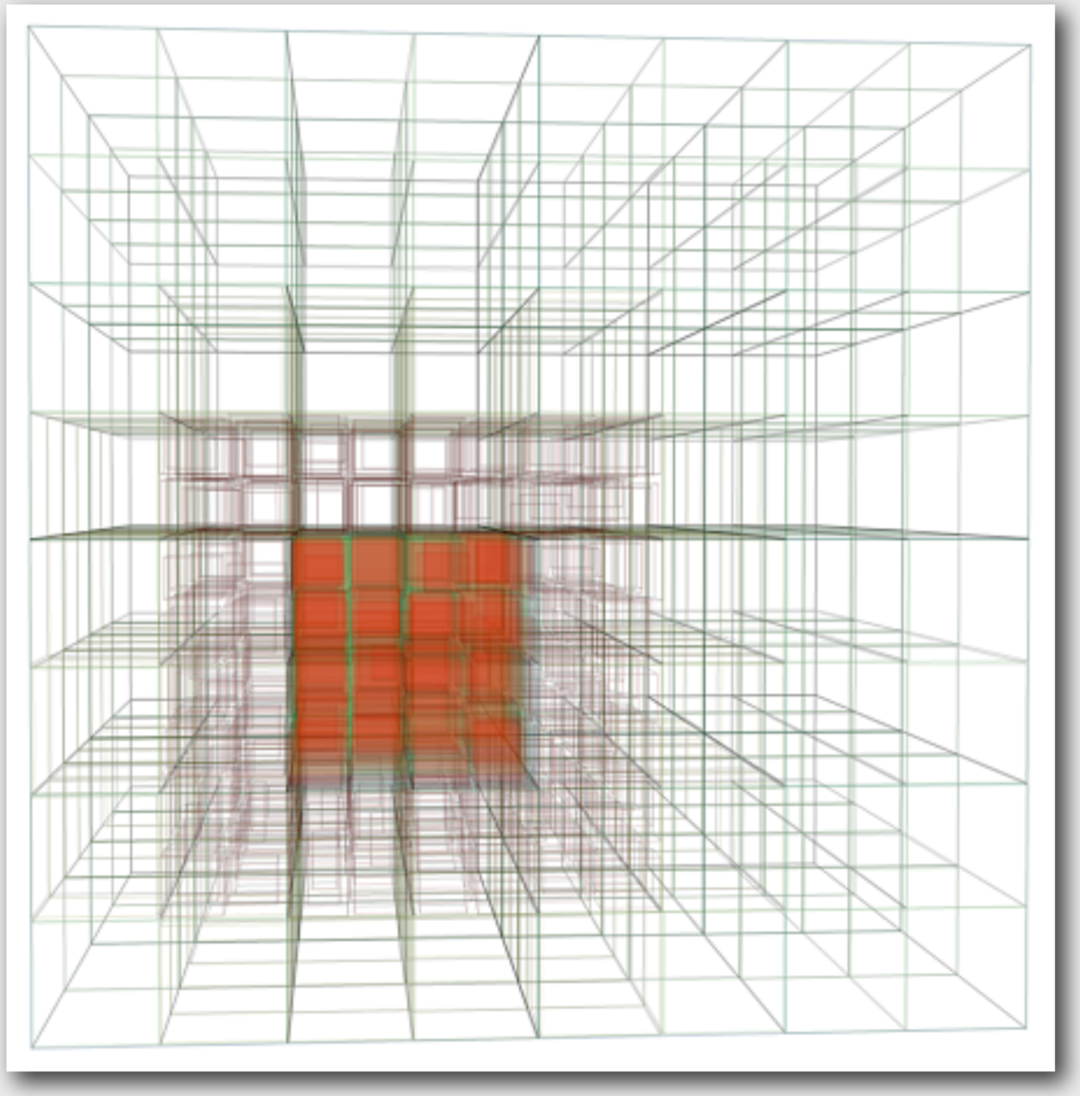
$$\begin{aligned}
\bar{u}_i\left(\bar{x}_i\right) &= \sum_{j=1}^{N_v} K_\sigma\left(\bar{x}_j - \bar{x}_i\right) \times \bar{\Gamma}_j + U_\infty \\
K_\sigma(\bar{x}) &= K(\bar{x}) \int_0^{|\bar{x}|/\sigma} 4\pi\, g(r)\, r^2 dr \\
K(\bar{x}) &= -\frac{\bar{x}}{4\pi|\bar{x}^3|} \\
g(r) &= (3/4\pi)\exp\left(-r^3\right)
\end{aligned}$$

- **3-component u and 9-component ∇u is 72 Flops**

Applied Scientific Research
Engineering and Software Development Consultants

Faster summations

- Appel (1985),
  Barnes and Hut *Treecode* (1986),
  Greengard-Rokhlin *FMM* (1987)
- Principle: use simplified representation for faraway groups of particles
- Application: if box is too close, examine box's children, repeat...
- O(N) to O(N log N)



Our fast multipole treecode

1) **Create tree structure**

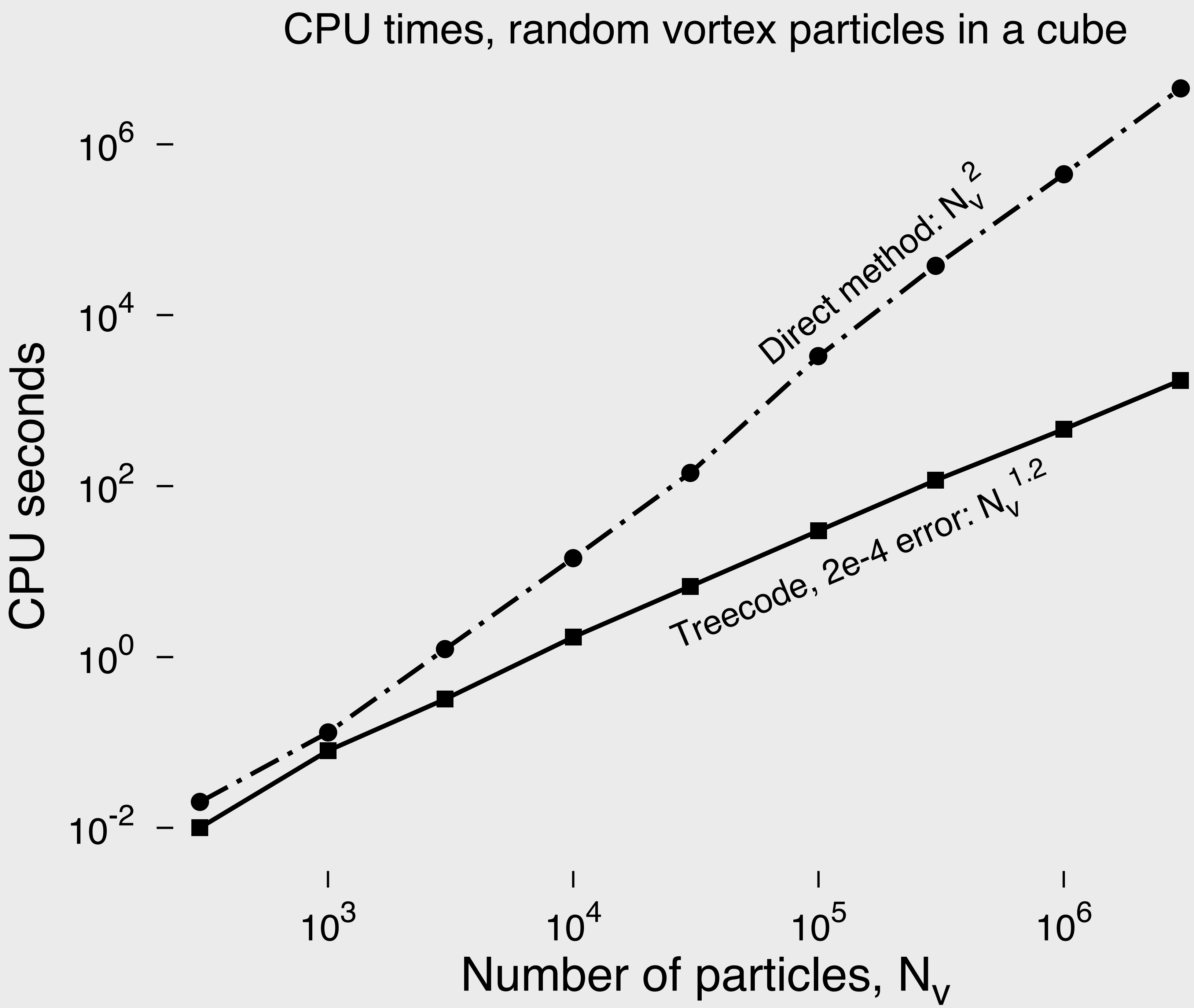
2) **Compute multipole moments for each box**

3) **For each particle:**
   - **Recurse through tree starting at root**
   - **Calculate influence from source box using either multipole multiplication or Biot-Savart, as necessary**
   - **Add freestream**

Our fast multipole treecode

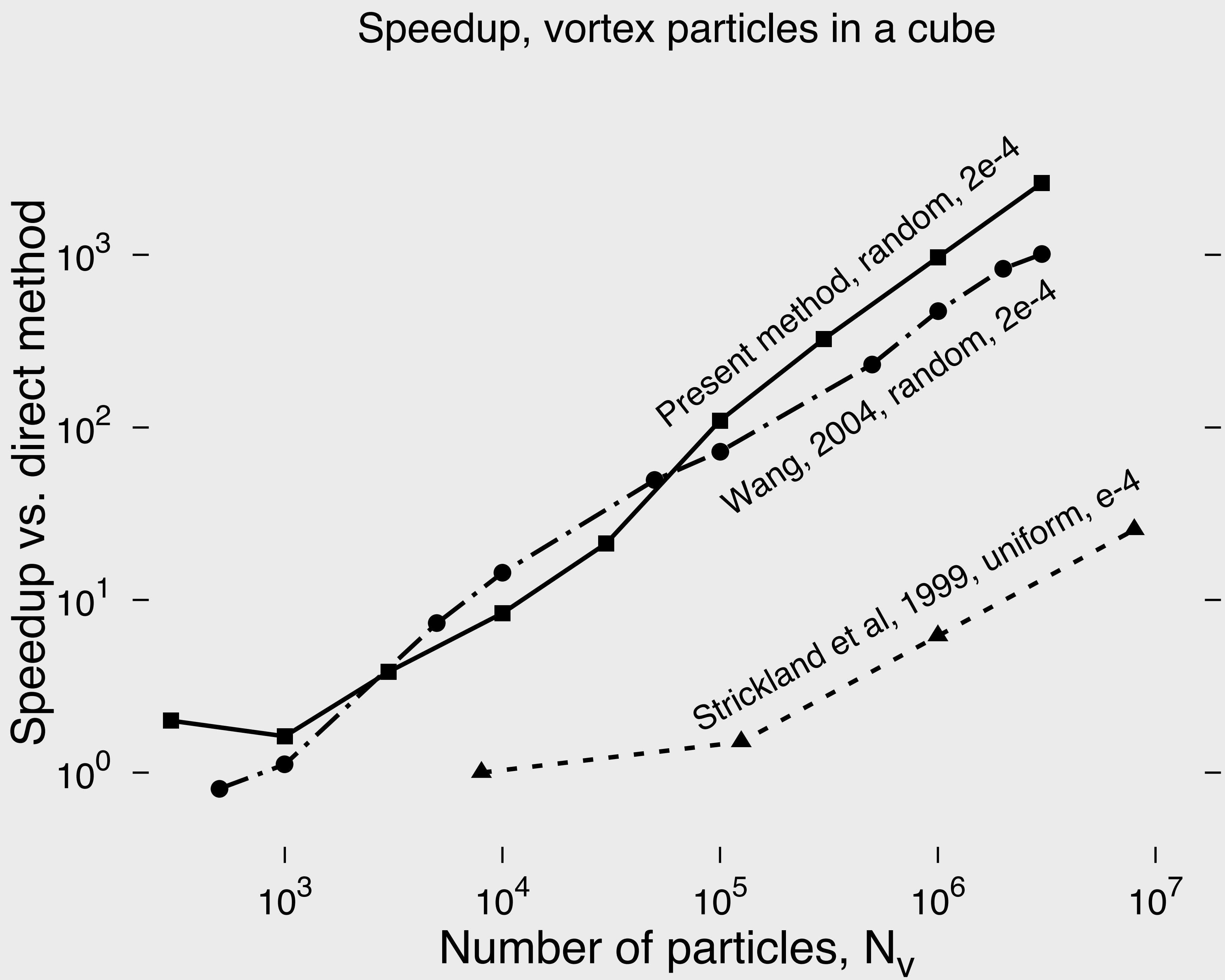
- VAMSplit k-d trees (White and Jain, 1996)
- Only real multipole moments (Wang, 2004)
- 7 or 9 orders of multipole moments
- Barnes-Hut box-opening criteria, with extensions
  Barnes and Hut (1986), Warren and Salmon (1994)
- Interaction lists made uniquely for each particle

Serial performance



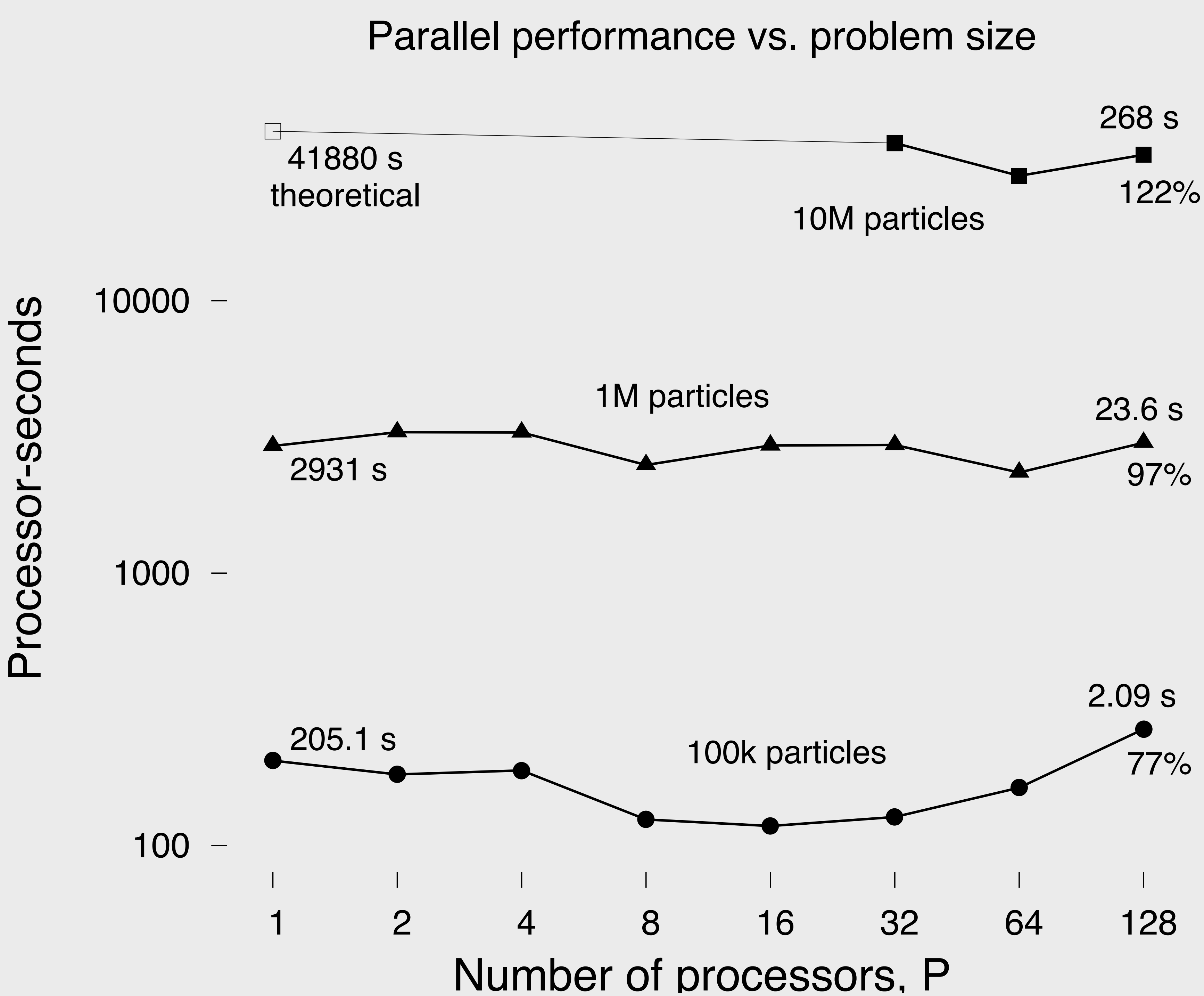
APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Serial performance



Applied Scientific Research
Engineering and Software Development Consultants

Parallel performance



APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

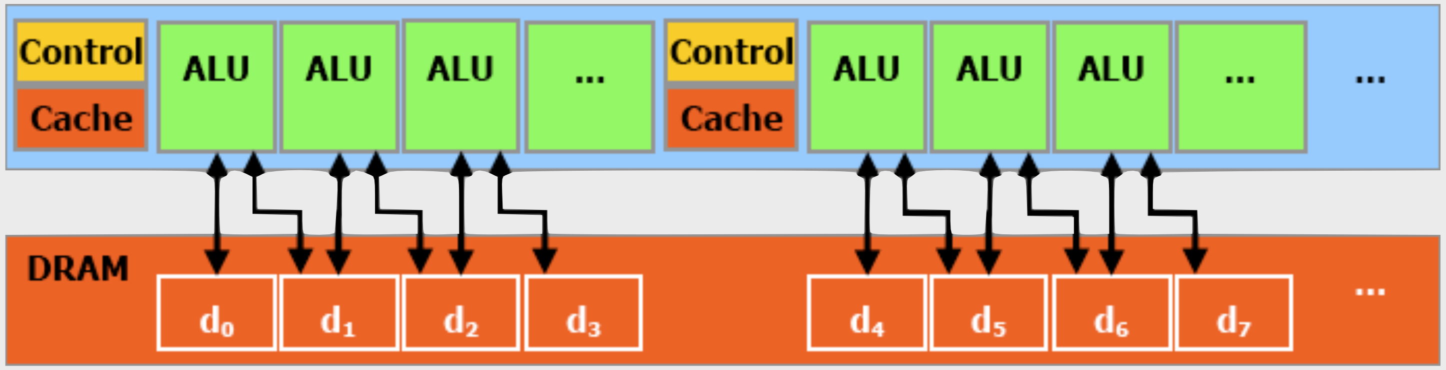
GPU-direct method

- BrookGPU (Buck *et al* 2004)
  - Looks like C language, converted to Cg
  - Define *streams* of data
  - OpenGL driver on Linux returns one set of `float4` per kernel

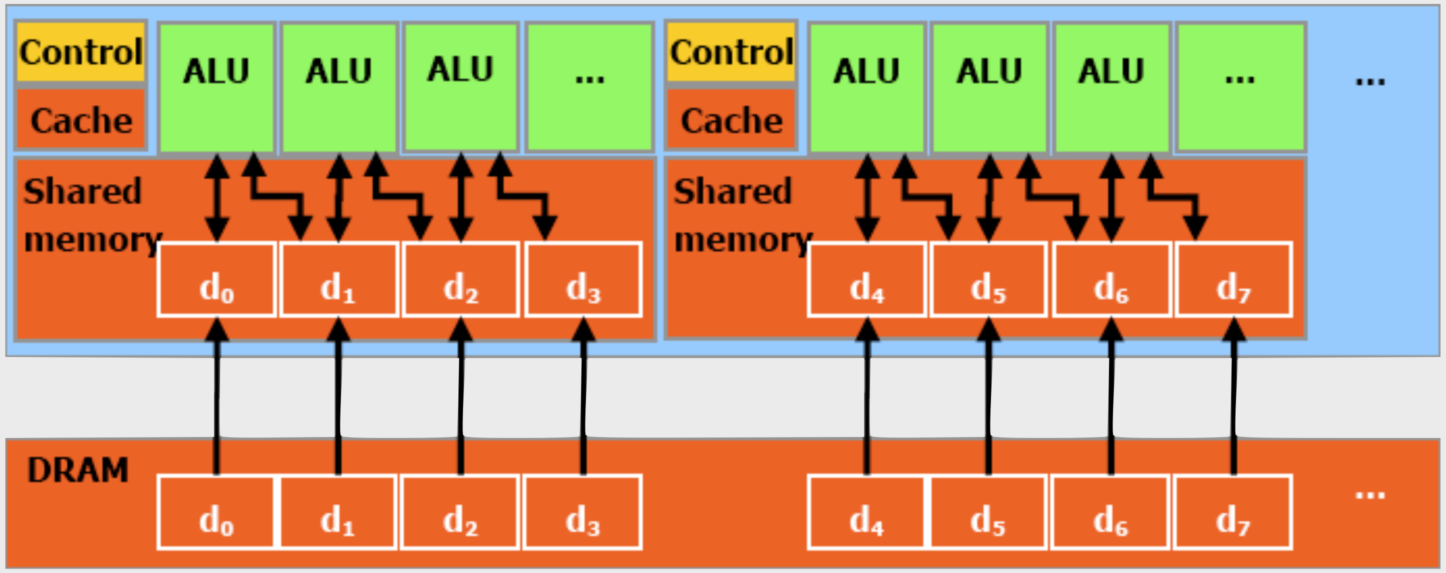
- CUDA: Compute Unified Device Architecture (NVIDIA, 2007)
  - C-like syntax, compiled by nvcc
  - Explicit control of memory on device
  - Kernels have few limits

GPU-direct method

- NVIDIA 8800 GTX has 8 *multiprocessors*, each with:
  - 16 *processing elements* (PEs)
  - 8192 registers
  - 16 banks of 16kB *shared memory*



Without shared memory



With shared memory

GPU-direct method

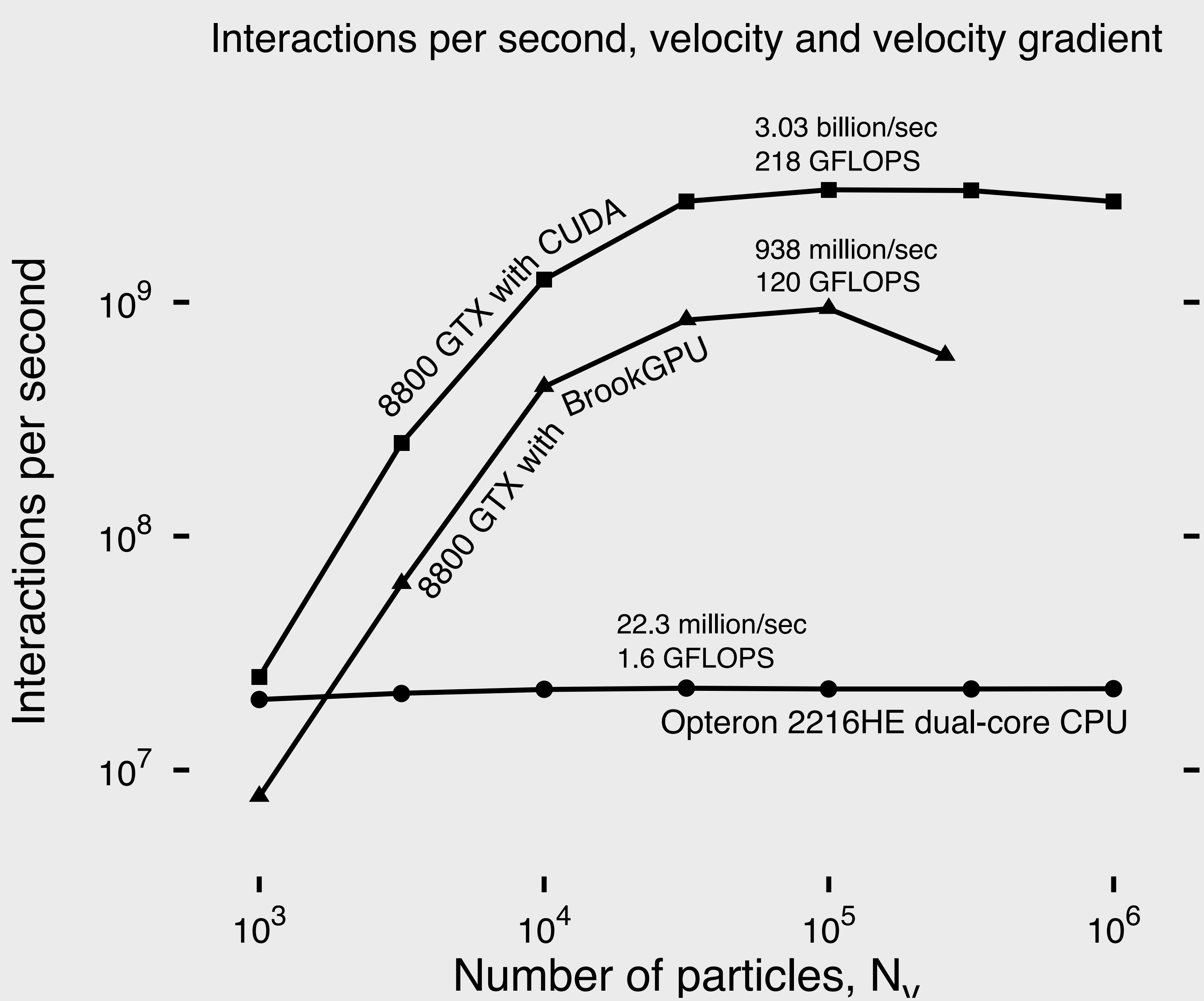
1) Put all particles and field points on GPU main memory

2) Start one thread per field point (8 x 64 = 512 at a time)

3) For each group of 64 threads:
   - Load 64 source particles from main to shared memory
   - Calculate influences using Biot-Savart
   - Repeat; when done, write resulting u and ∇u to main GPU memory

4) Read all u and ∇u back to CPU memory

GPU-direct method



APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

GPU treecode

1) **Create tree structure (CPU, single-thread)**

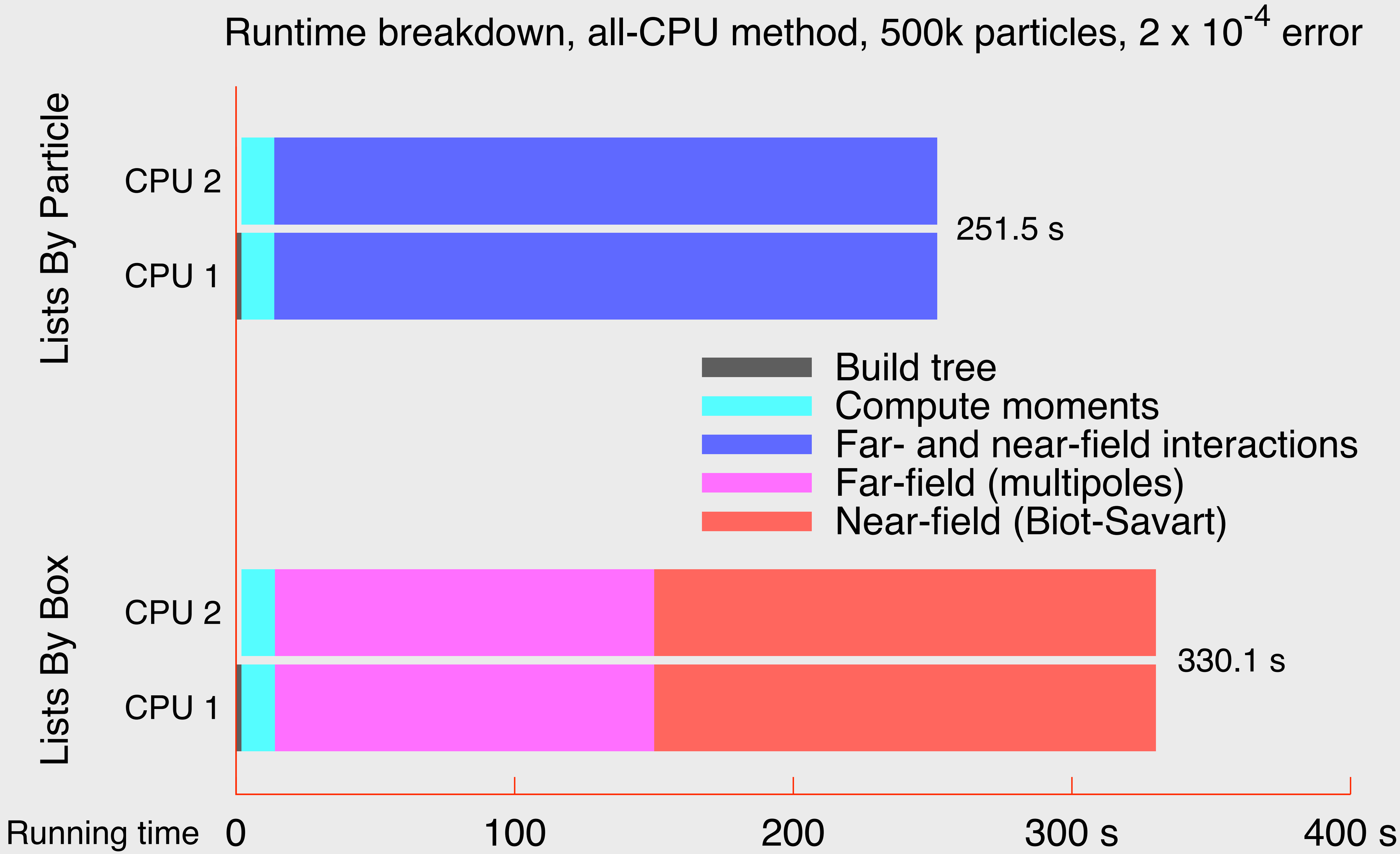
2) **Compute multipole moments for each box (CPU, multithread)**

3) **Determine all interaction lists (CPU, single-thread)**

4) **Calculate all influences from *far-field* using multipole mult. (GPU)**

5) **Calculate all influences from *near-field* using Biot-Savart (GPU)**

GPU treecode



Applied Scientific Research
Engineering and Software Development Consultants

GPU-direct method -- far-field

1) **Put all particles, field points, and interaction lists on GPU**

2) **Start one thread per field point (8 x 64 = 512 at a time)**

3) **For each group of 64 threads:**
   - **Iterate through that group's *far-field* interaction list**
   - **Load *210 multipole moments* from GPU main to shared memory**
   - **Calculate influences using *multipole multiplication***
   - **Repeat; when done, write resulting u and ∇u to GPU main mem.**

4) **Read all u and ∇u back to CPU main memory**

GPU-direct method -- near-field

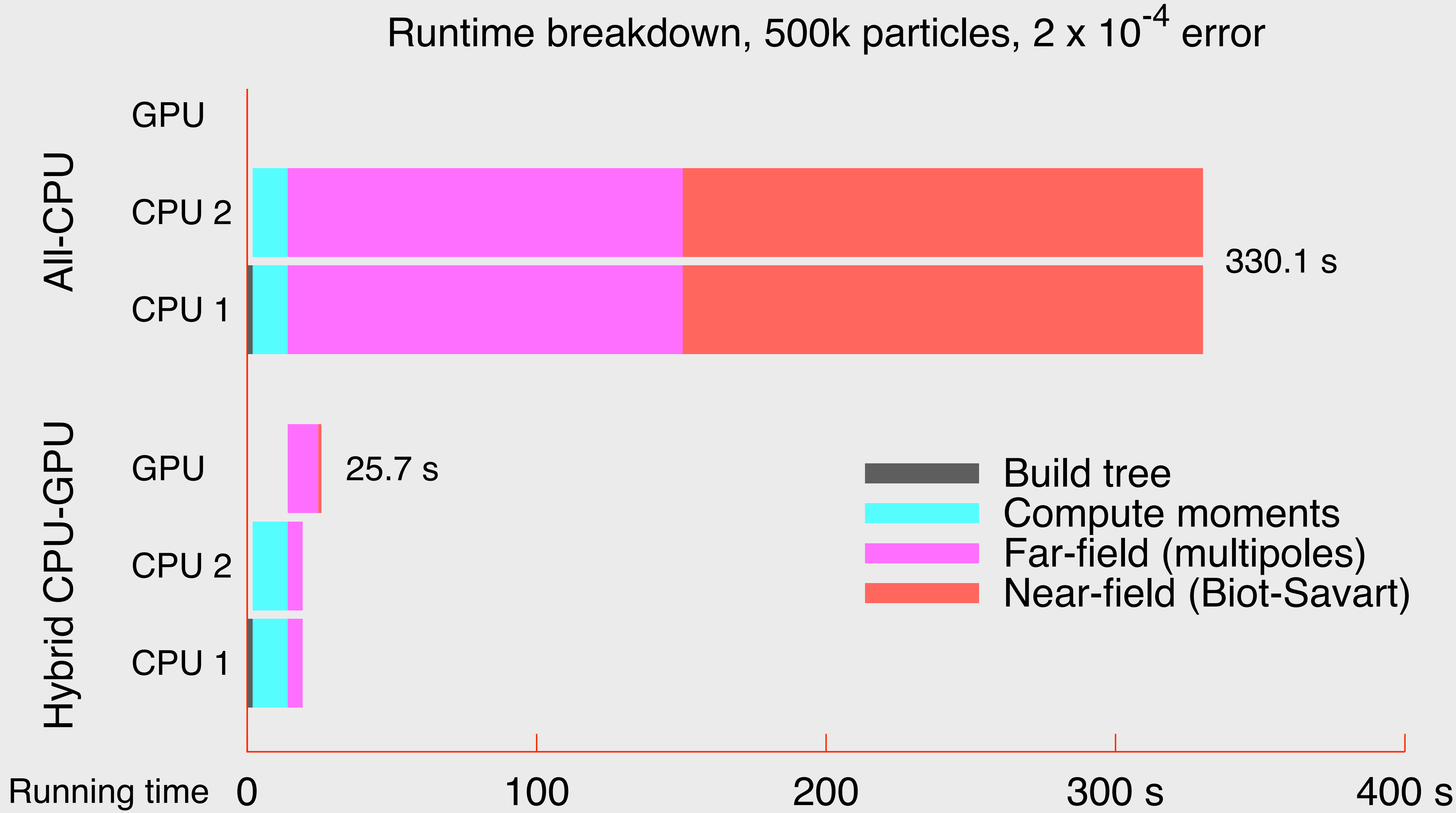
1) **Put all particles, field points, and interaction lists on GPU**

2) **Start one thread per field point (8 x 64 = 512 at a time)**

3) **For each group of 64 threads:**
   - **Iterate through that group's *near-field* interaction list**
   - **Load *64 source particles* from GPU main to shared memory**
   - **Calculate influences using *Biot-Savart***
   - **Repeat; when done, write resulting u and ∇u to GPU main mem.**

4) **Read all u and ∇u back to CPU main memory**

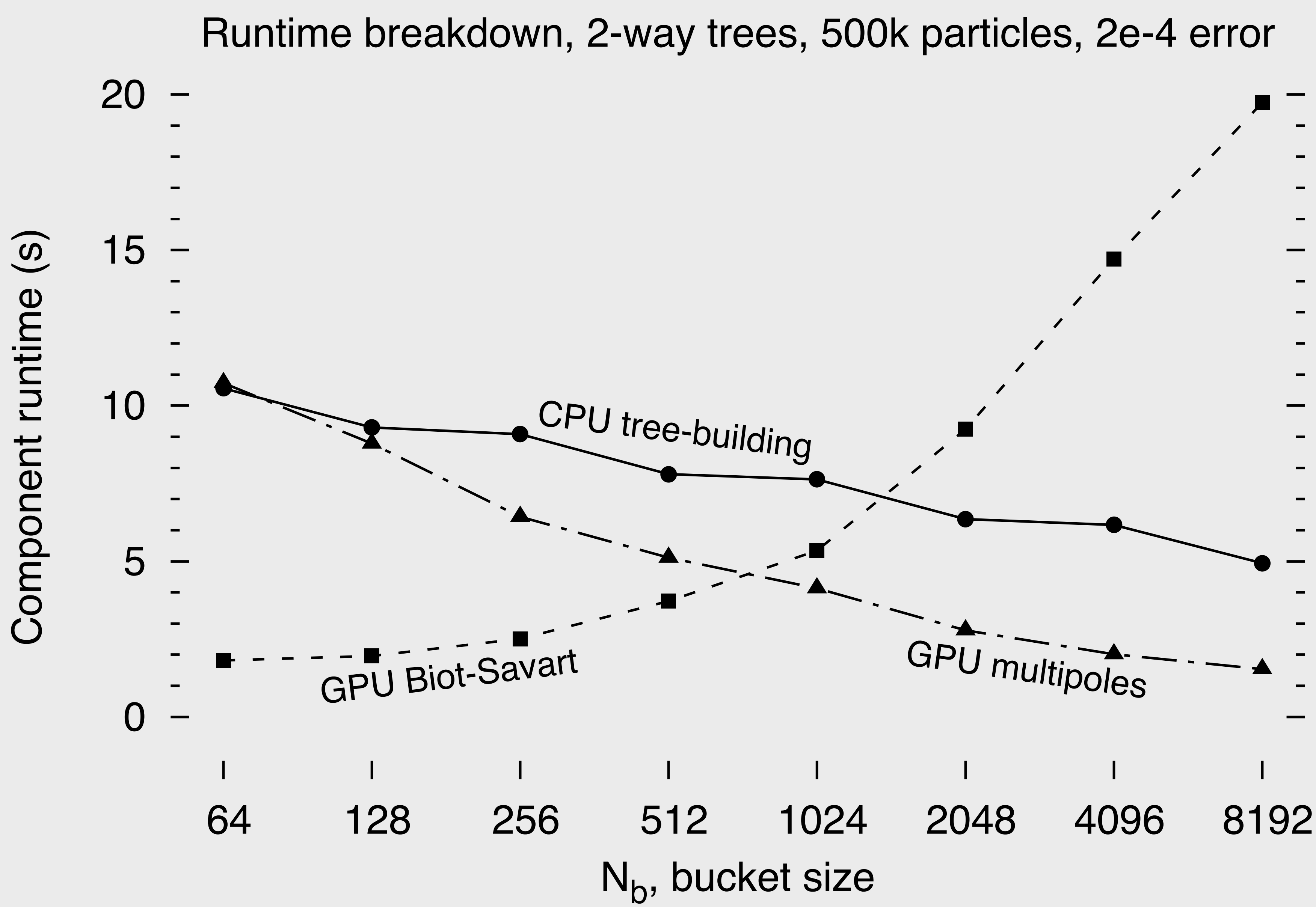
GPU treecode



APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

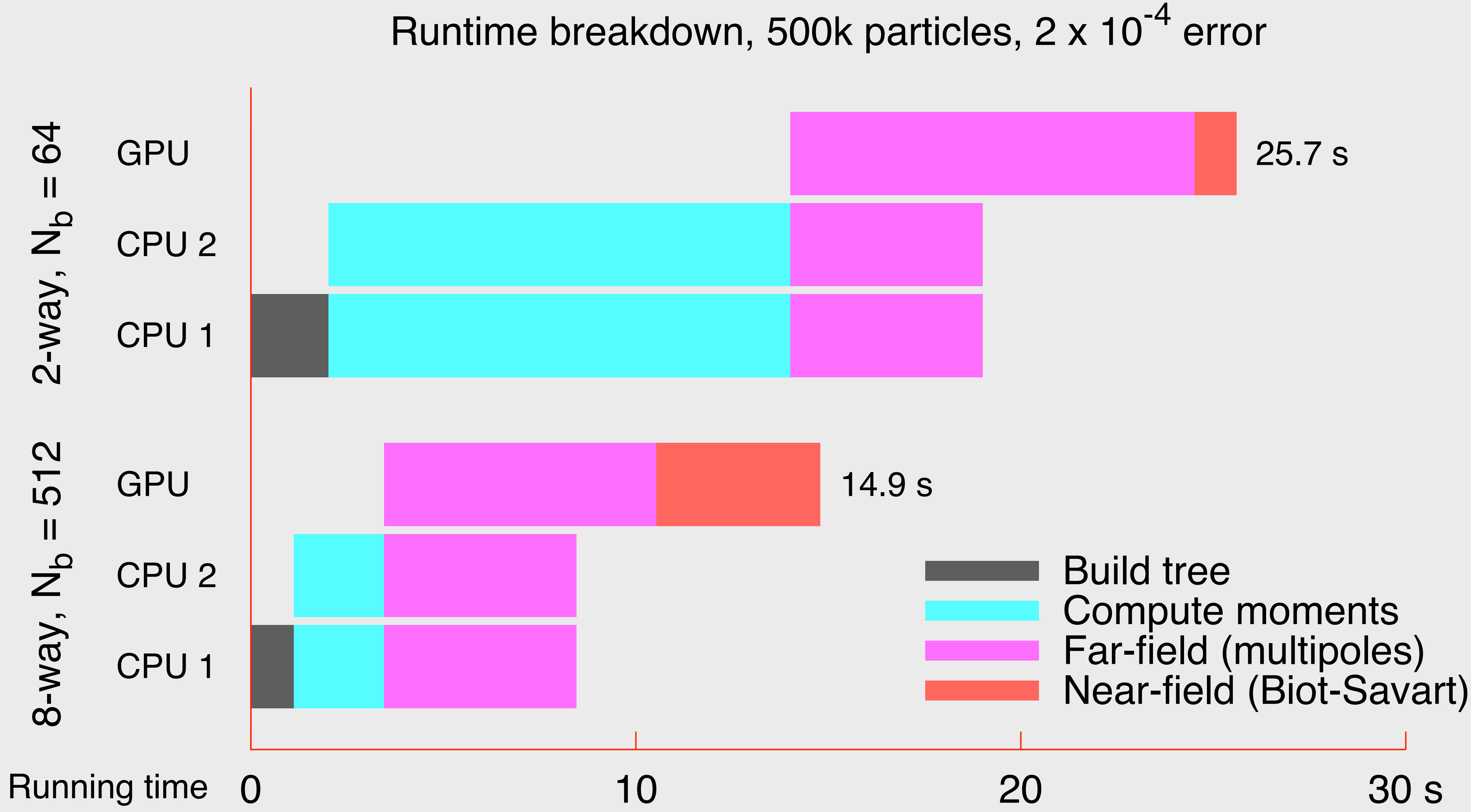
GPU treecode

- Changing *bucket size* shifts work between near- and far-field



APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

GPU treecode



Applied Scientific Research
Engineering and Software Development Consultants

Best performance, 500k particles, 2×10^{-4} mean velocity error

	Total time	Tree-building	Multipole moments	Velocity solution
CPU direct	11242 s	-	-	11242 s
CPU treecode	251.5 s	2.0 s	11.7 s	237.8 s
GPU direct	88.7 s	-	-	88.7 s
GPU treecode	14.9 s	1.1 s	2.3 s	11.4 s

Dynamic simulation step

- **Inviscid Flow - Lagrangian Vortex Element Method**

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

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

- **Diffusion - Vorticity Redistribution** (Shankar and van Dommelen, 1996)

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

Dynamic simulation step

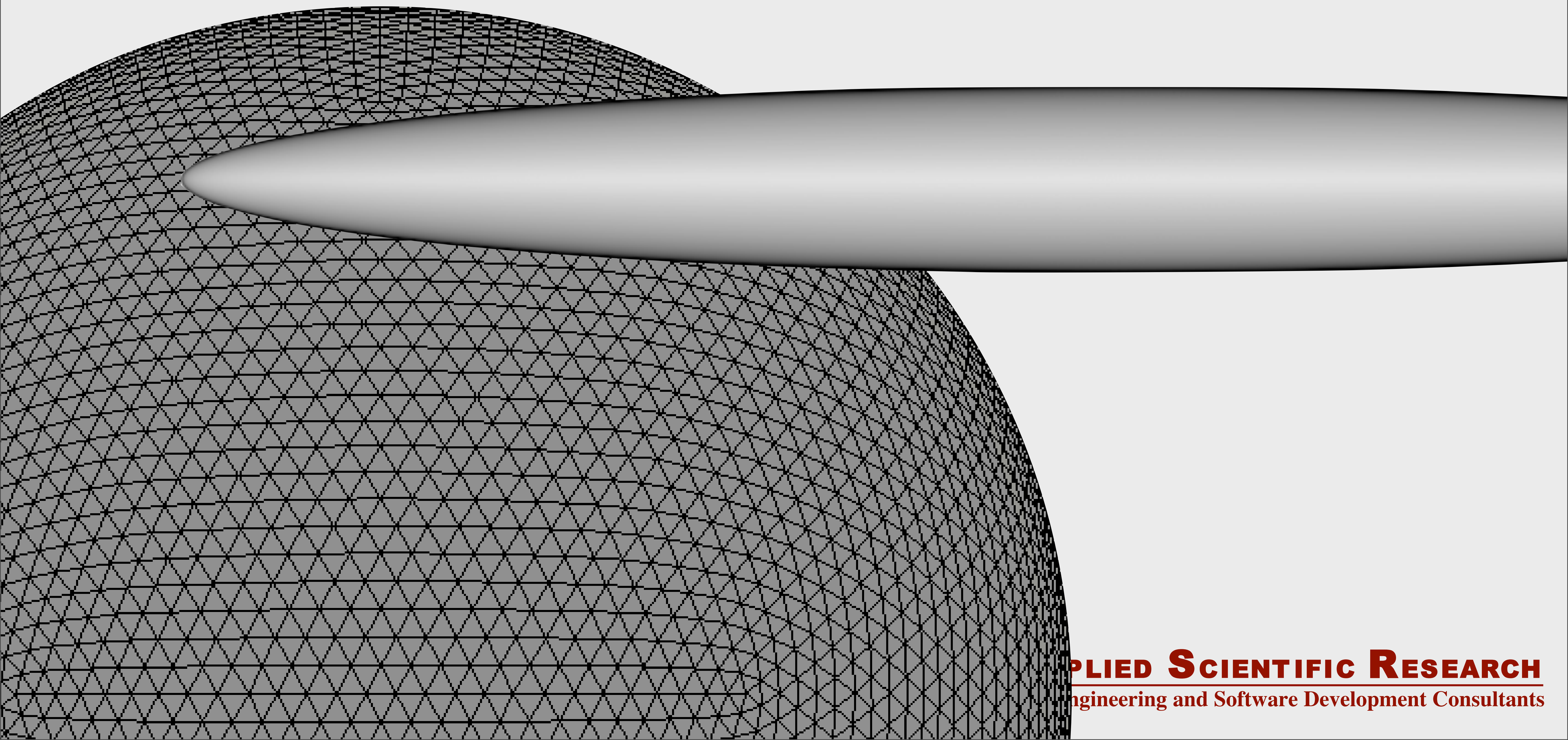
- **Wall B.C. satisfied by generating vorticity at the wall -**
  Boundary Element Method

$$\frac{1}{2}(\bar{\gamma}A)_i \times \hat{n} + \sum_{m \neq i}^{N_v} (\bar{\gamma}A)_m \times \int_{S_i} K(\bar{x} - \bar{x}_m)\, d\bar{x} = \bar{Q}_s$$

$$\bar{Q}_s = \int_{S_i} \bar{U}_\infty(\bar{x})\, d\bar{x} - \sum_{j=1}^{N_v} \bar{\Gamma}_j \times \int_{S_i} K(\bar{x} - \bar{x}_j)\, d\bar{x}$$

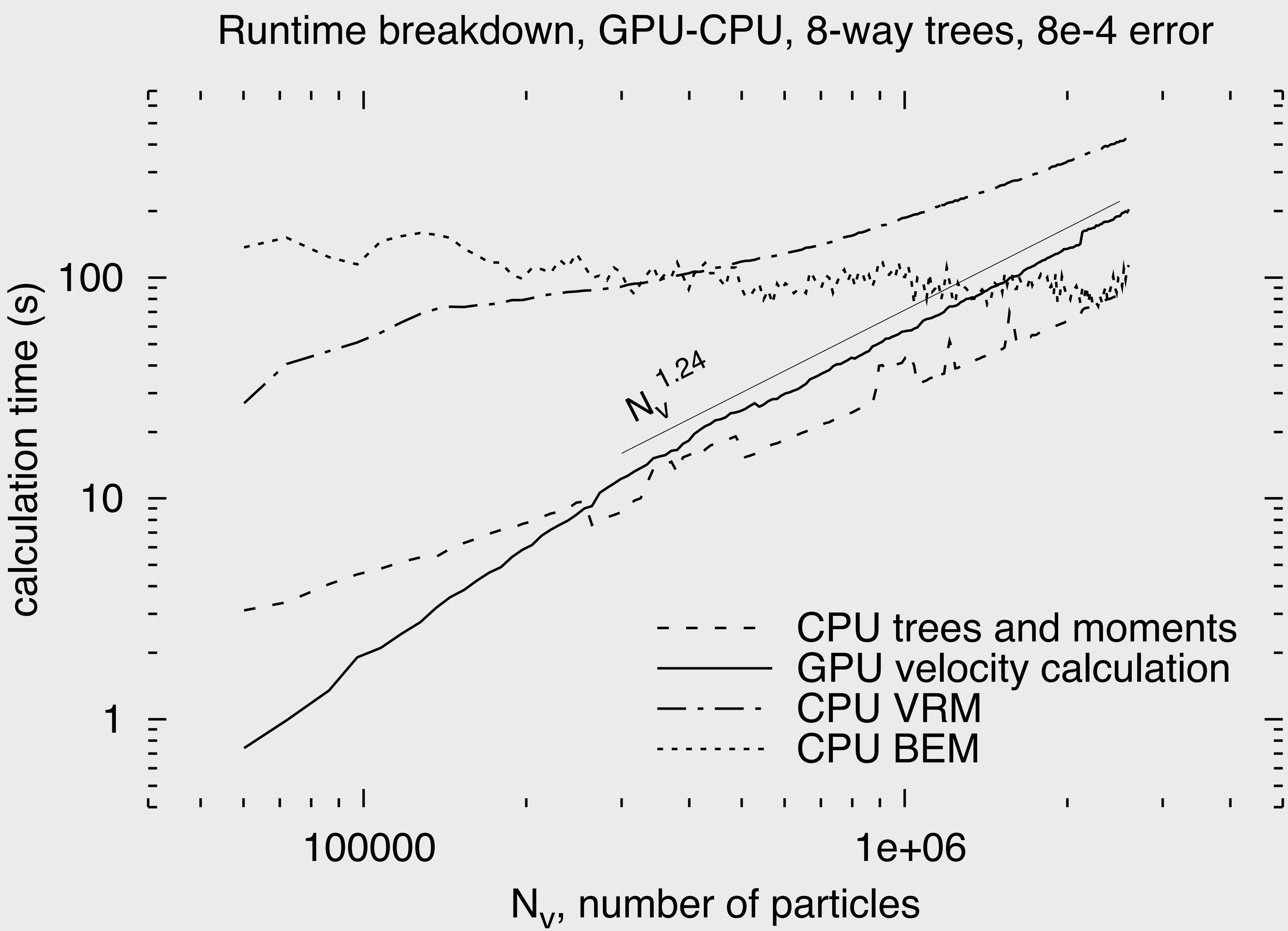
Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

SURFACE MESH



PLIED SCIENTIFIC RESEARCH
ngineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°



APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°

Re_D=1000

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60° Re_D=1000

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

Applied Scientific Research
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°

Re_D=1000

APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60° $Re_D=1000$

Applied Scientific Research
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

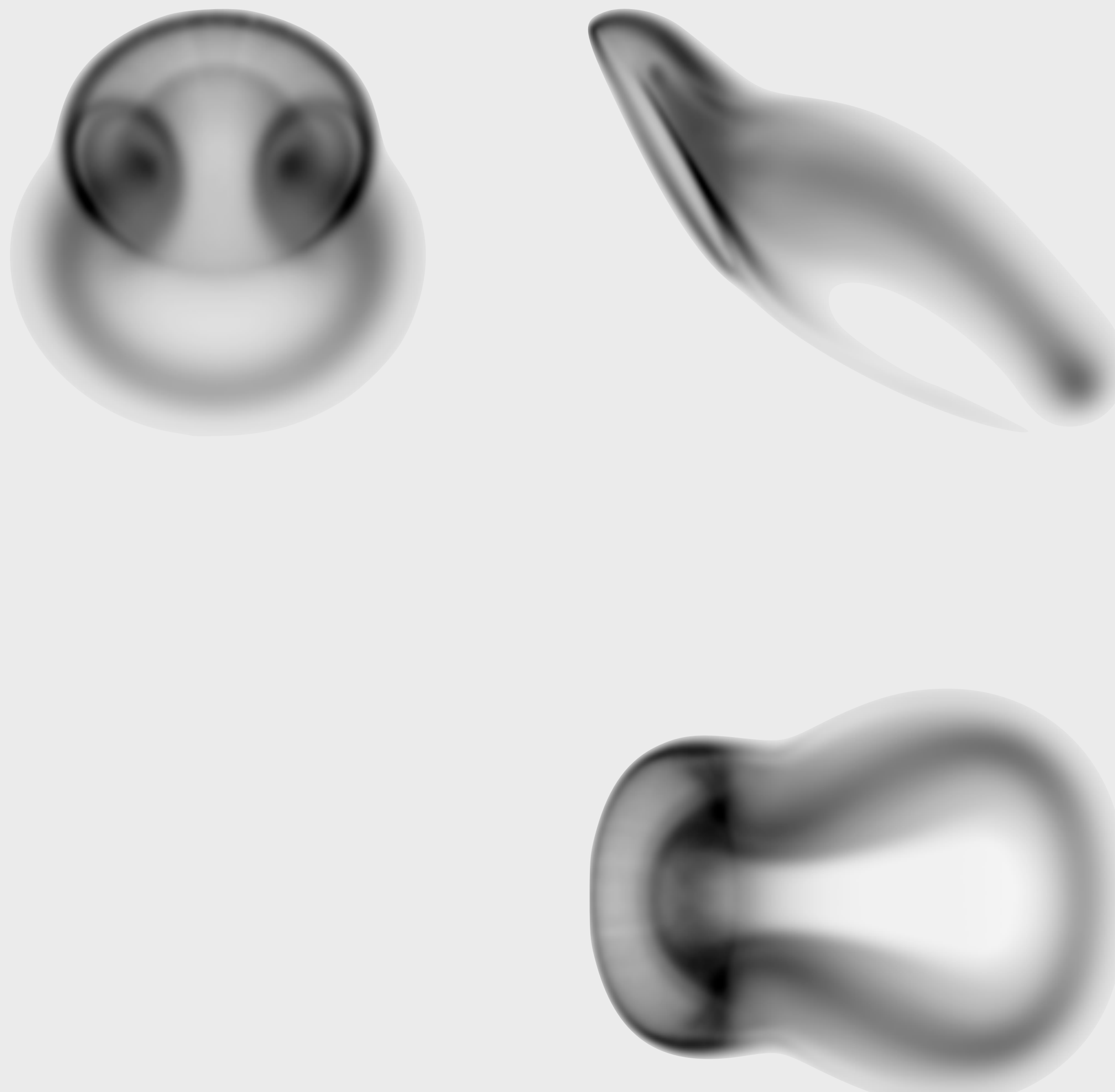
Applied Scientific Research
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

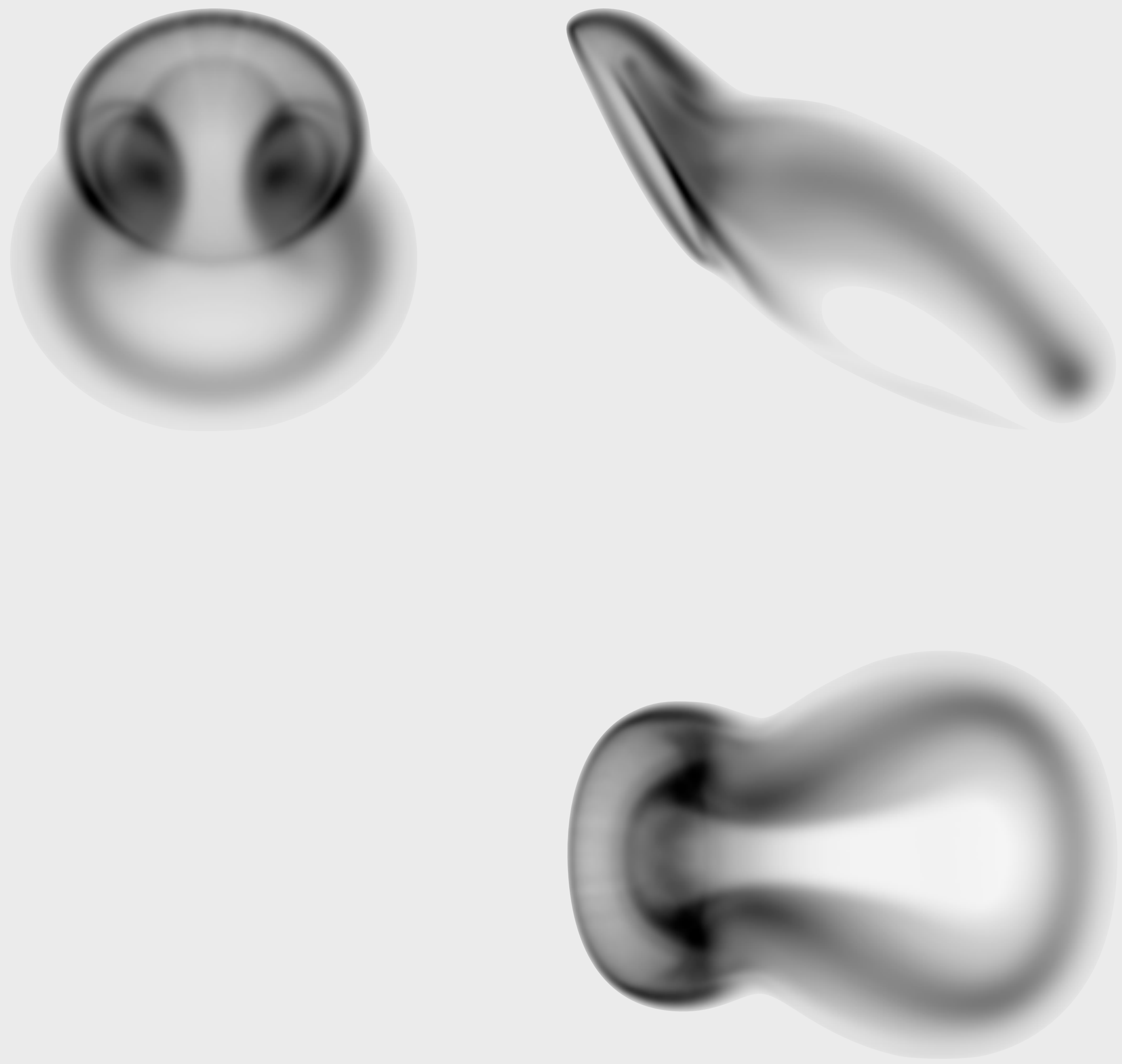
Applied Scientific Research
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



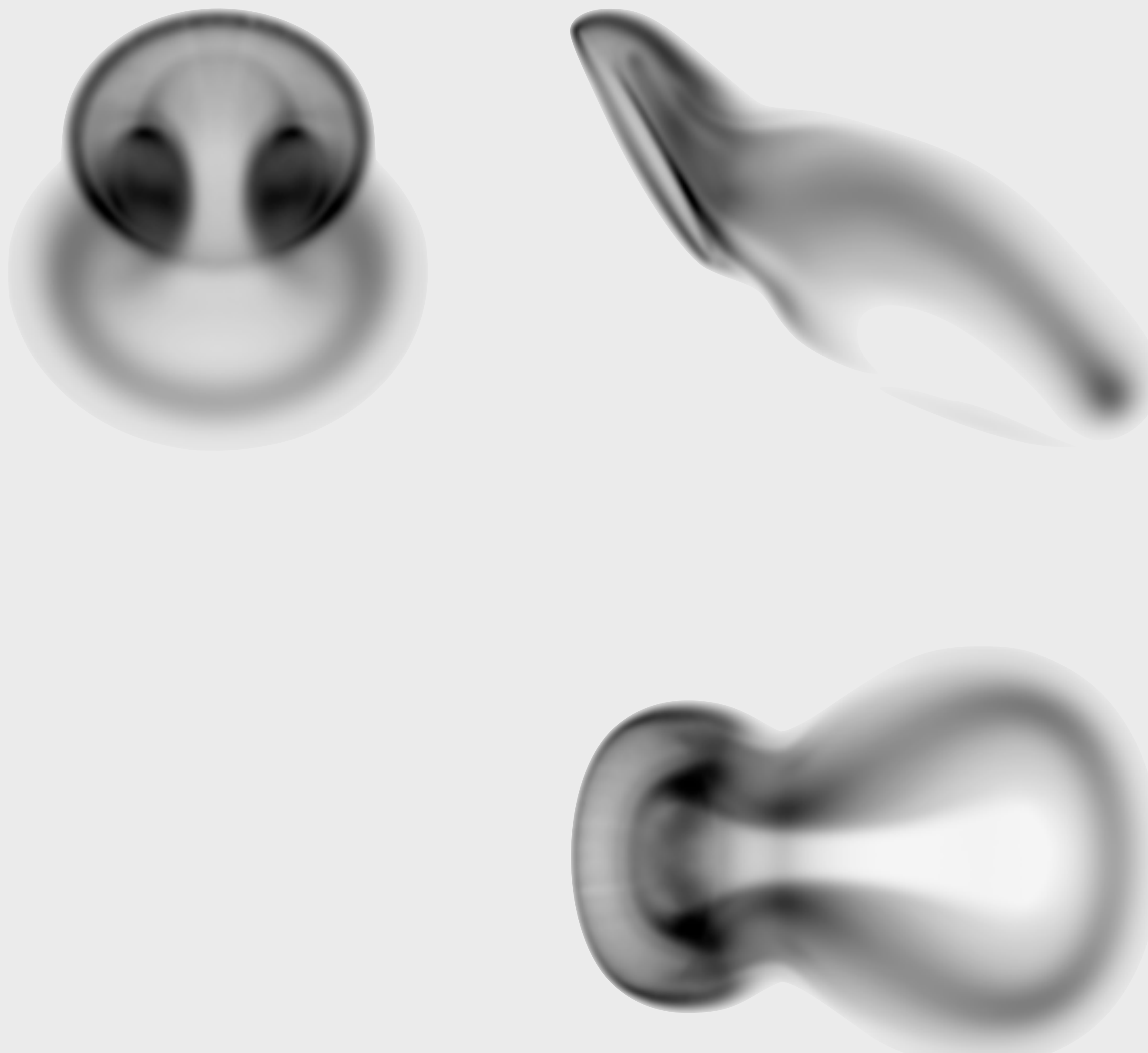
APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



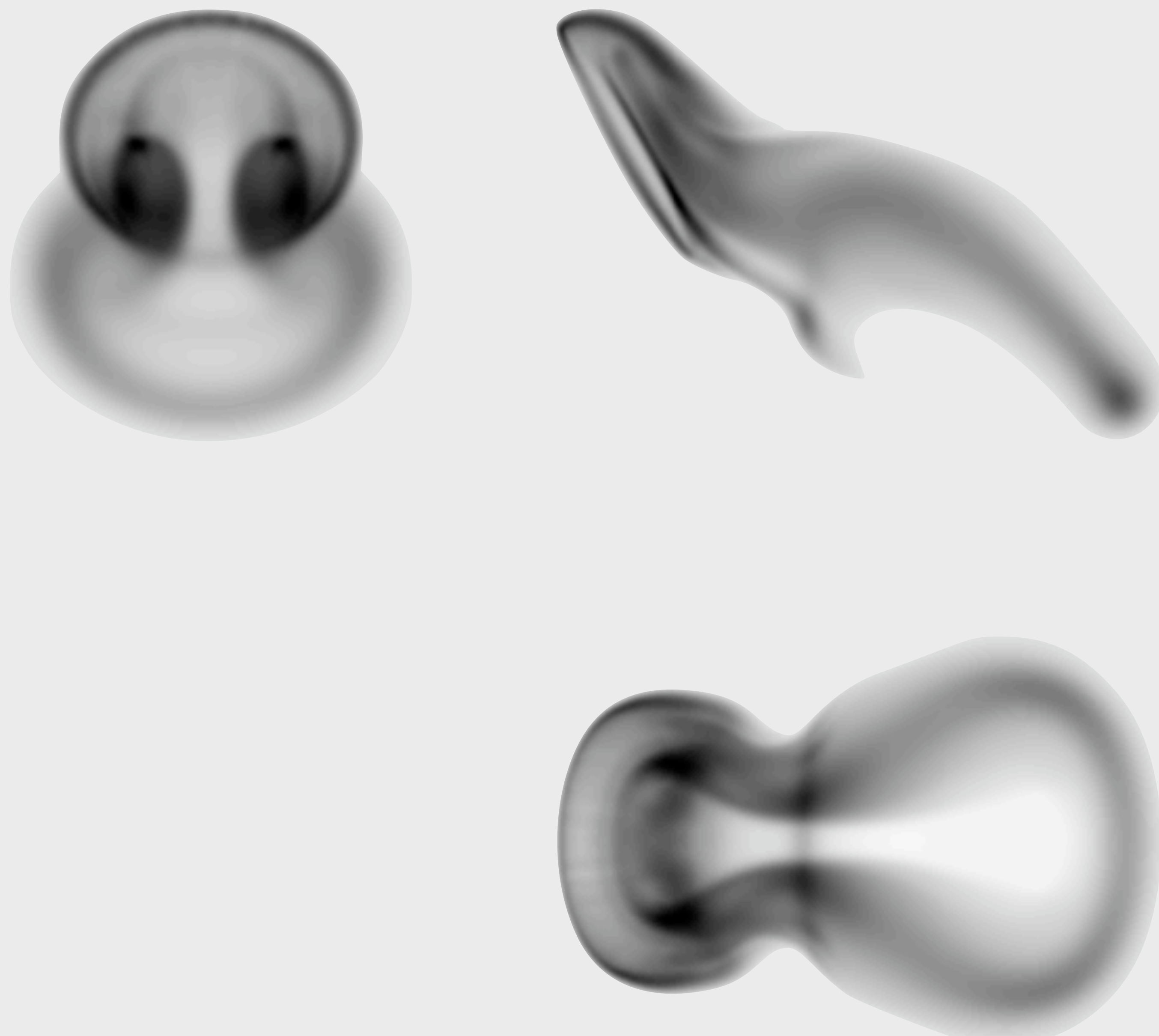
APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



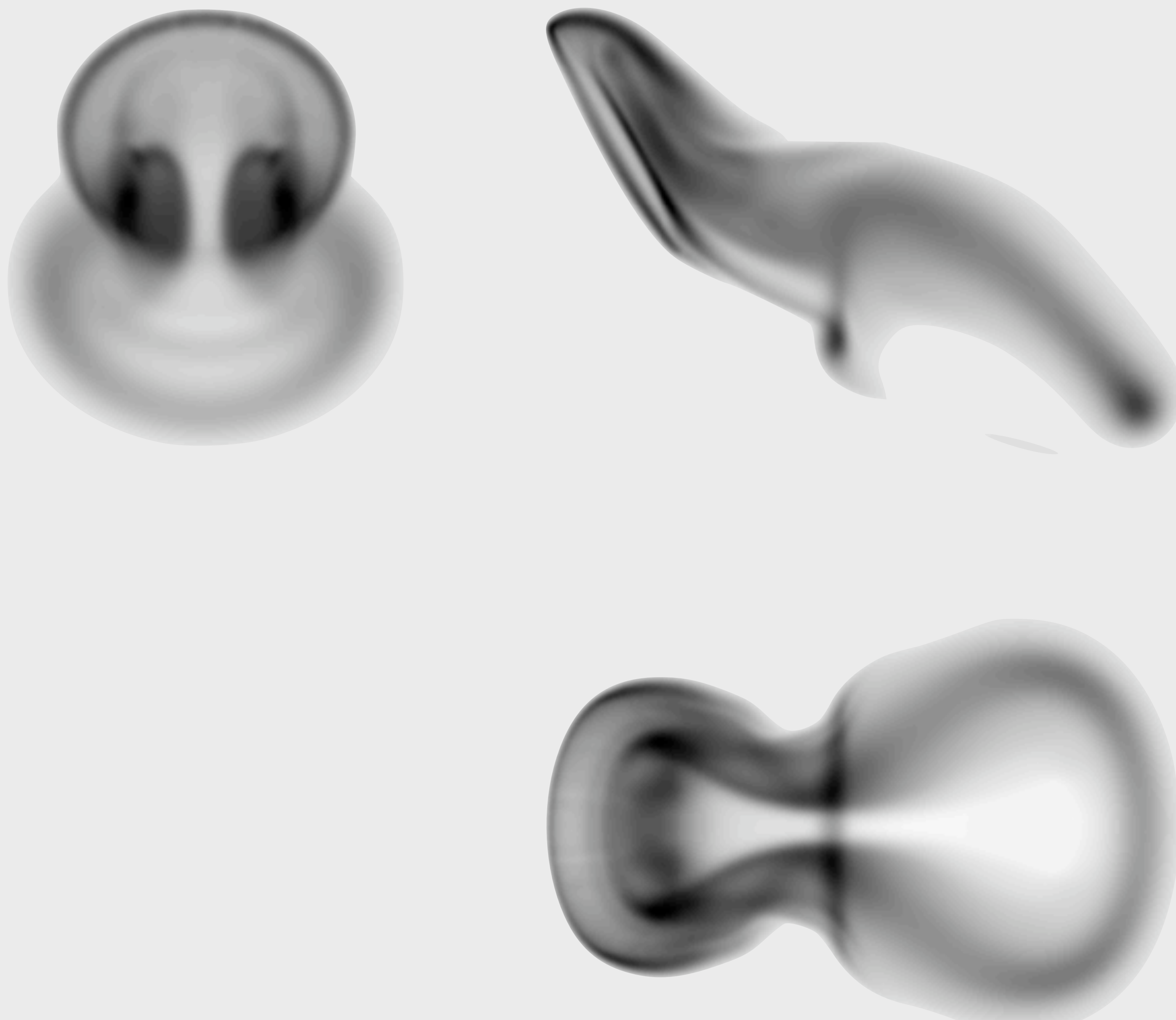
Applied Scientific Research
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



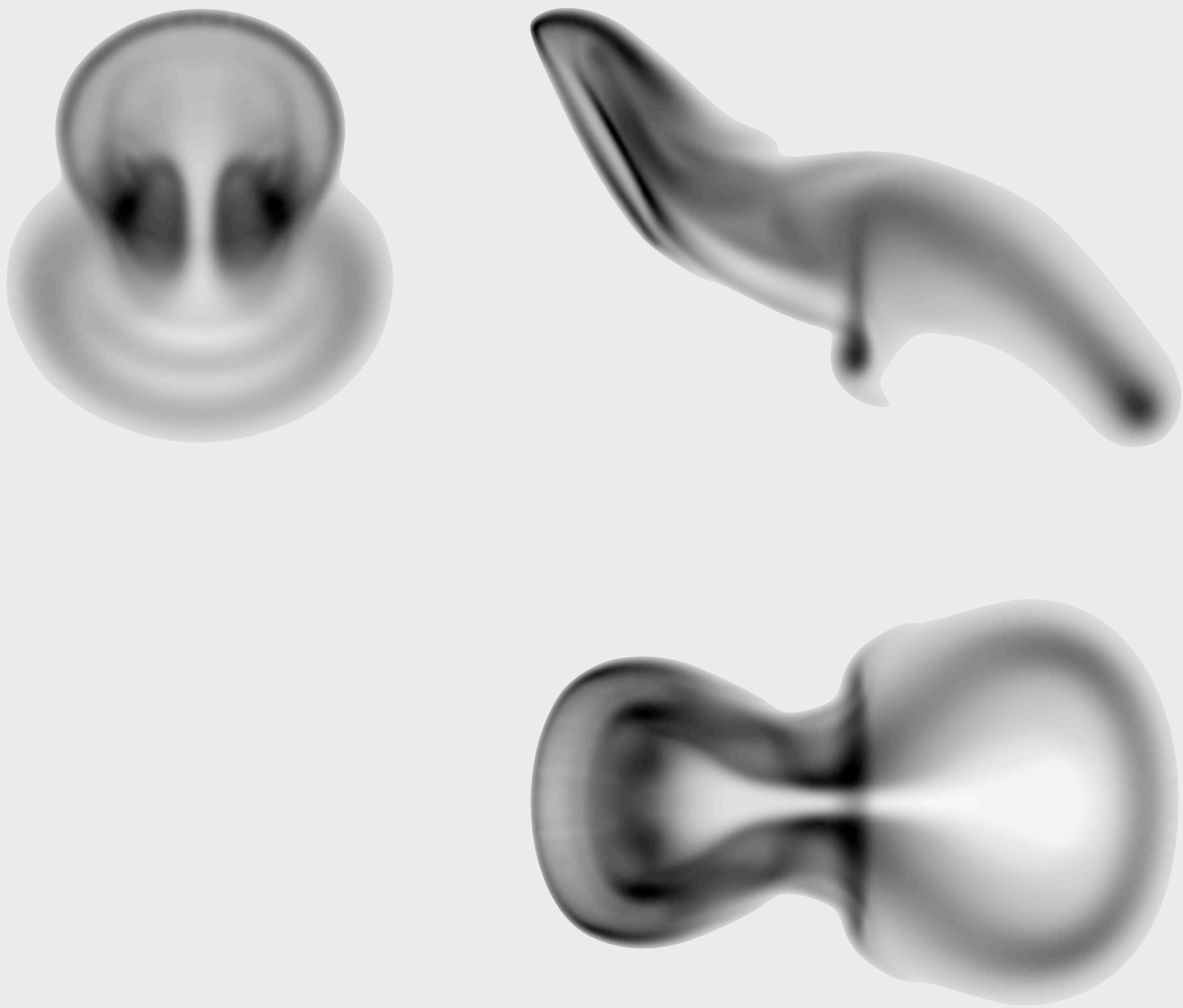
AfR APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



Flow over 10:10:1 ellipsoid at 60°

Re_D=1000

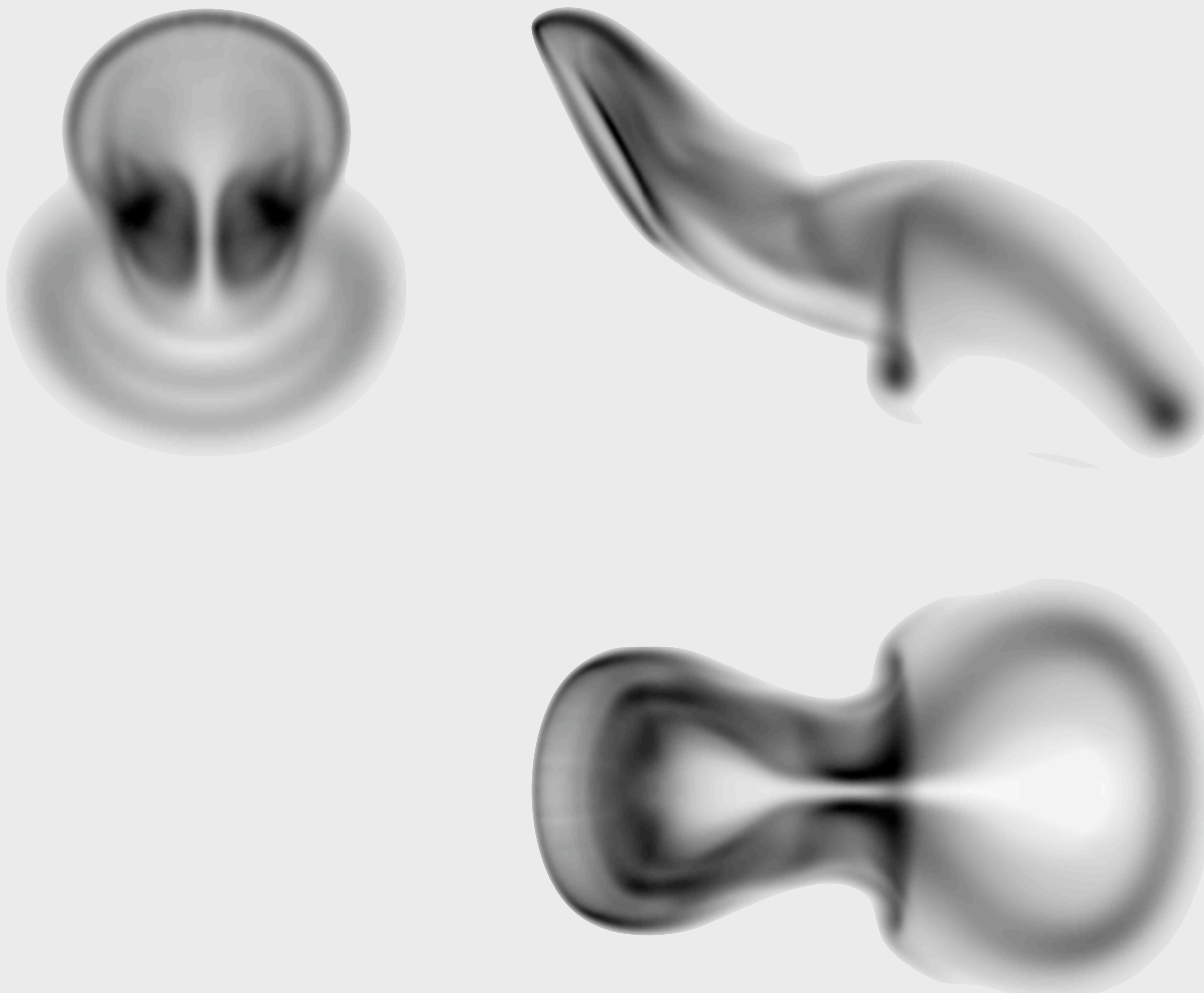


Applied Scientific Research

Engineering and Software Development Consultants

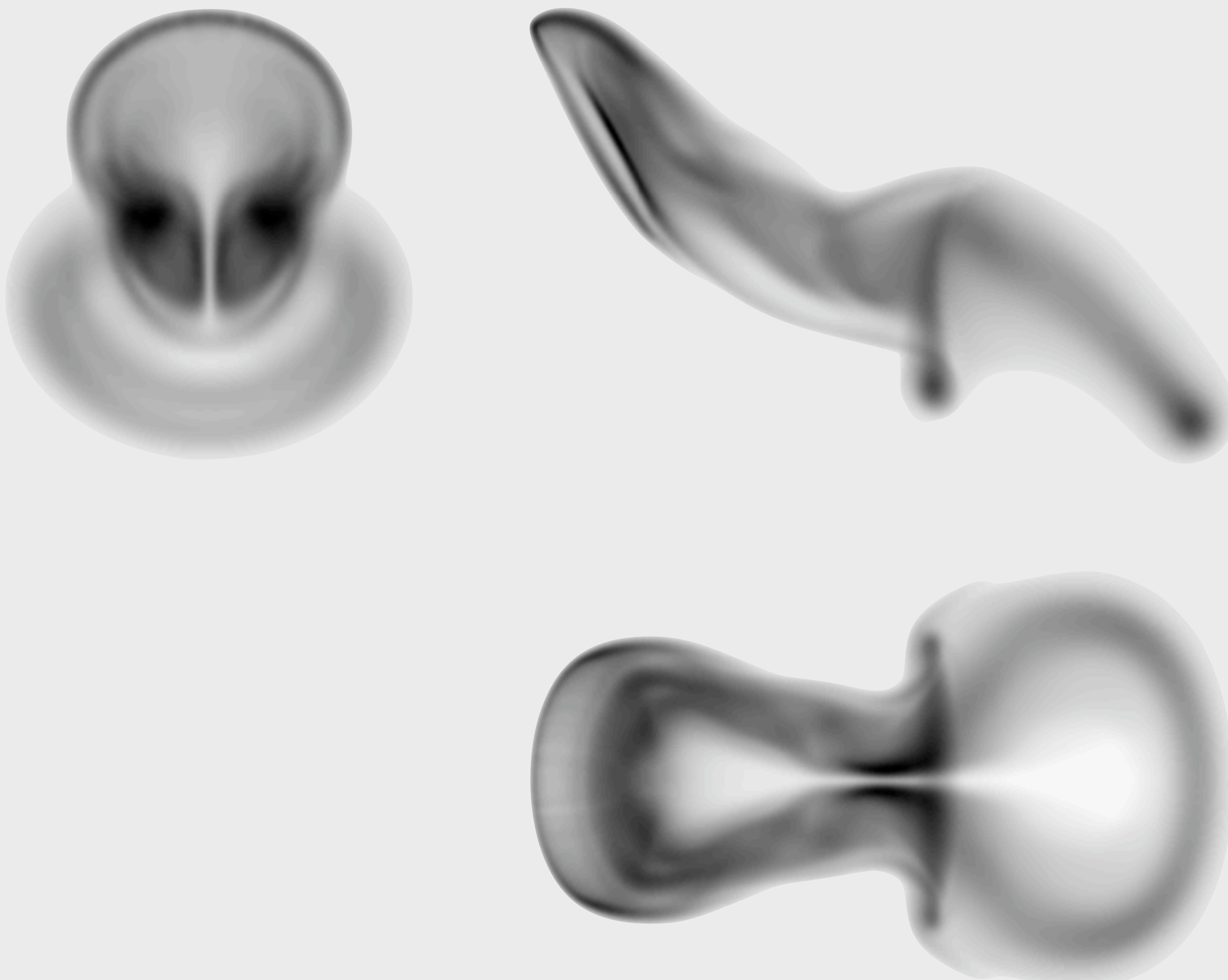
Flow over 10:10:1 ellipsoid at 60°

$Re_D=1000$



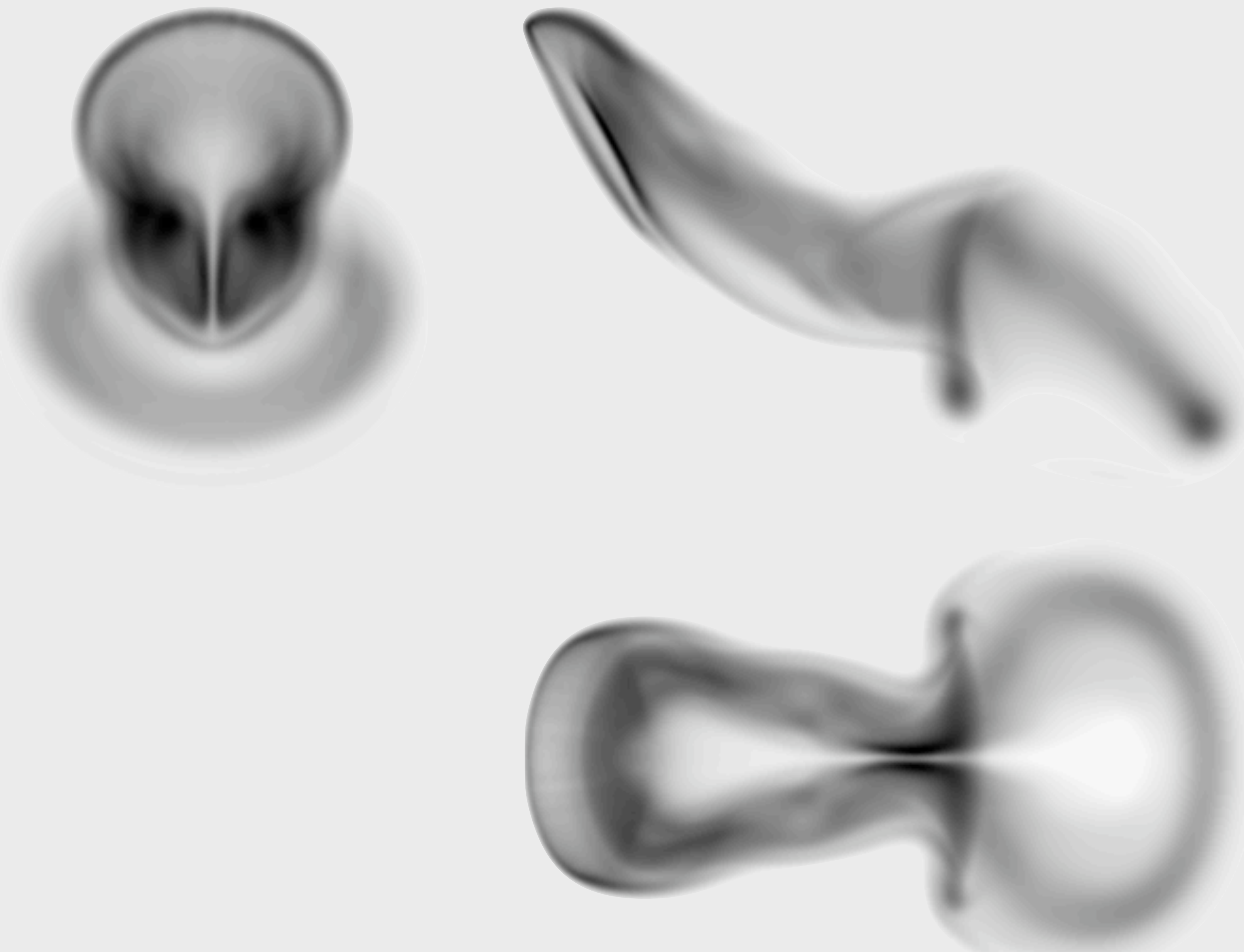
APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

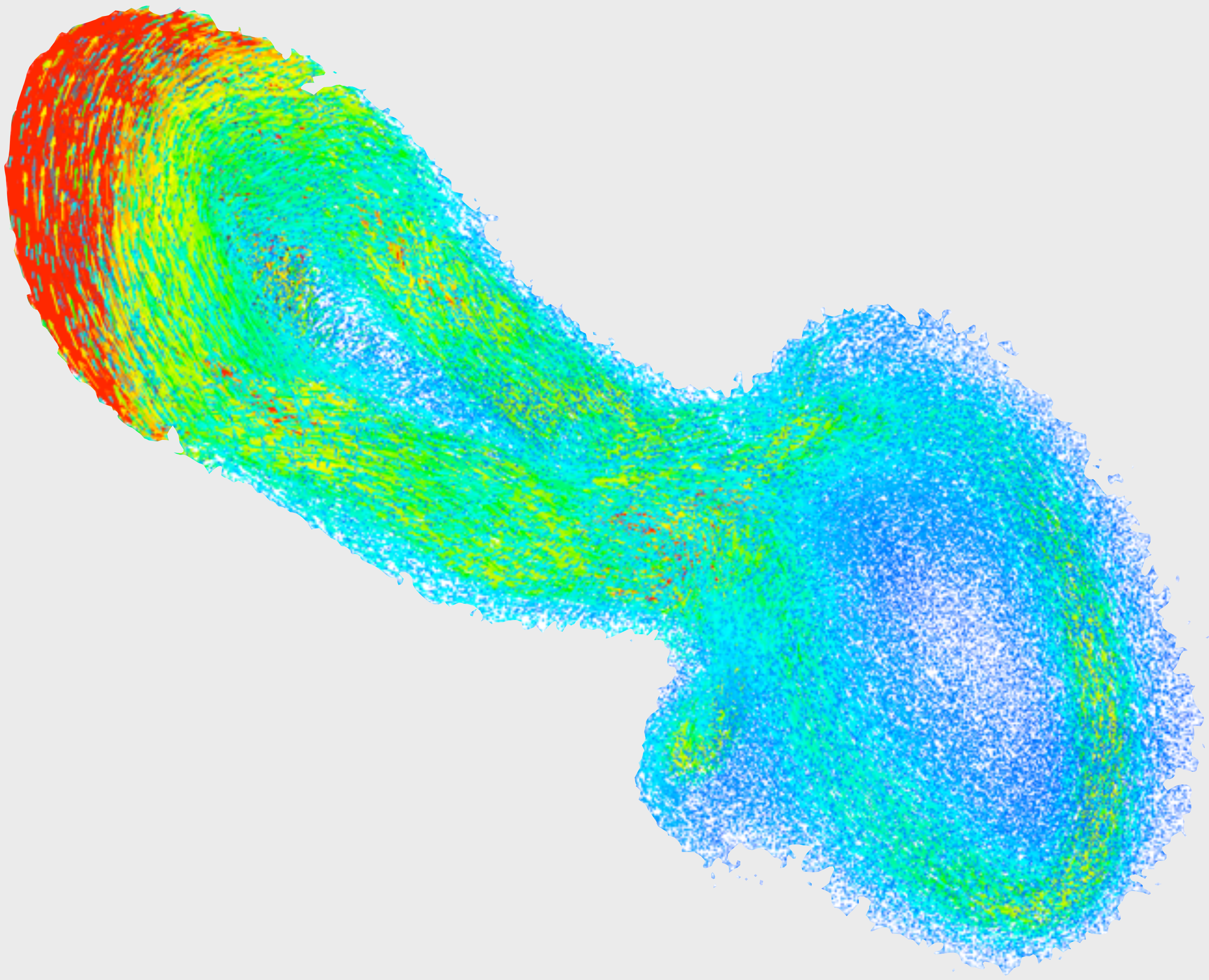


APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000

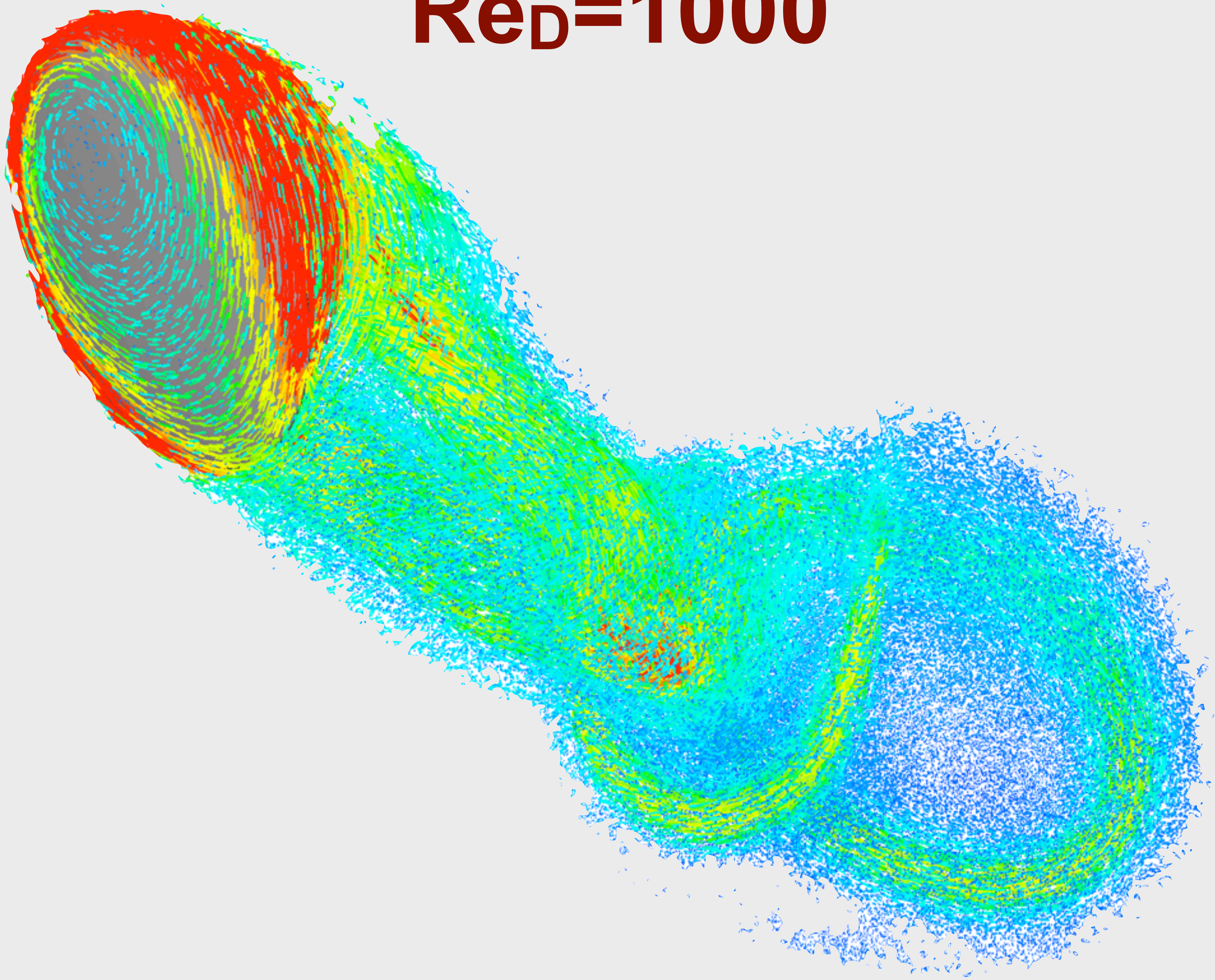


Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



APPLIED SCIENTIFIC RESEARCH
Engineering and Software Development Consultants

Flow over 10:10:1 ellipsoid at 60°
Re_D=1000



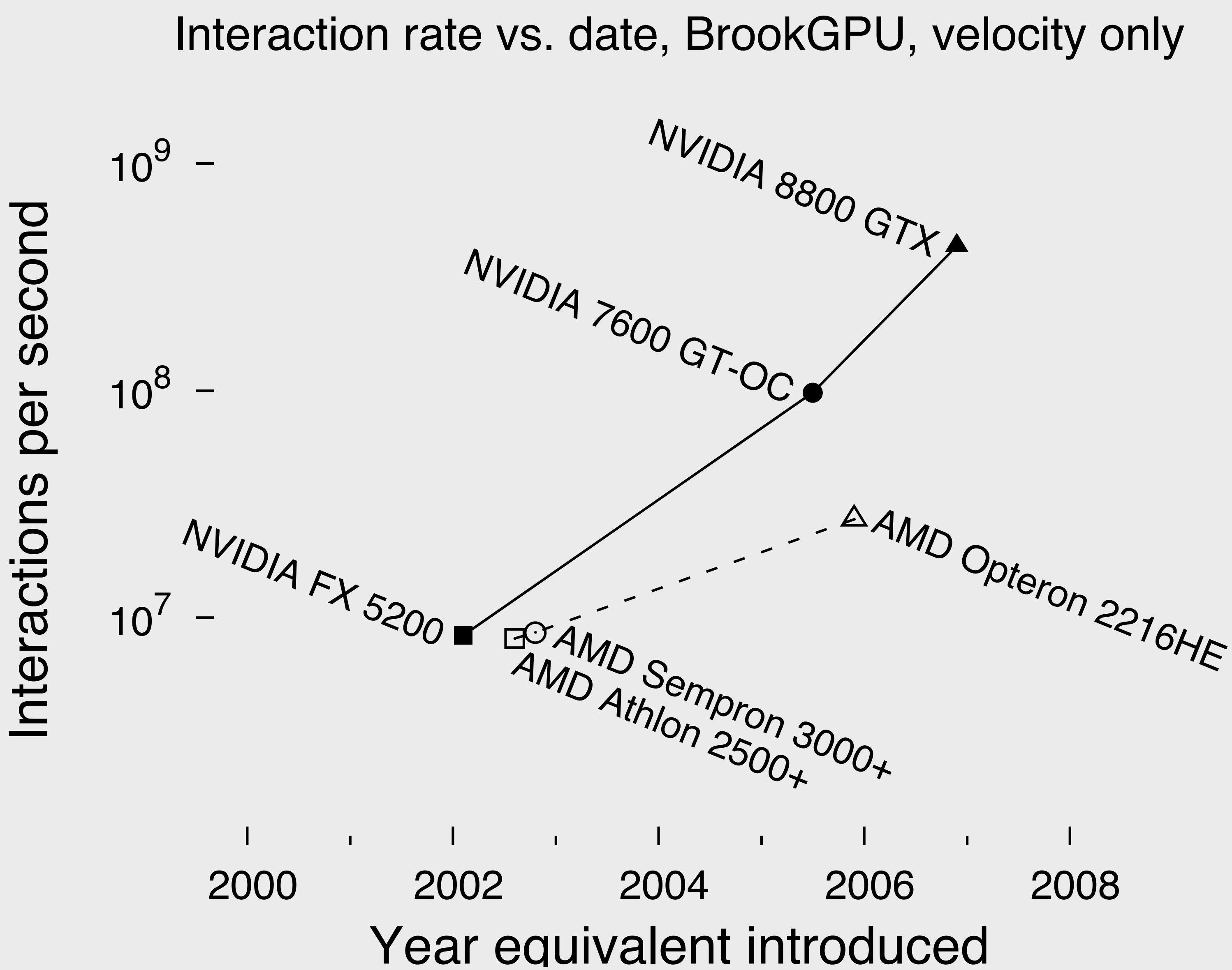
Applied Scientific Research
Engineering and Software Development Consultants

Conclusions

- Vortex particle methods adapt to hybrid CPU-GPU systems
- 140x speedup for direct summations on GPU
- 10x-15x speedup for GPU treecode velocity-finding
- 3.4x speedup for full dynamic timestep

Conclusions

- **GPU performance has increased rapidly...and should continue**



Future Work

- Reorganize to more efficiently use CPU & GPU
- Put VRM diffusion onto GPU
- Test on distributed-memory CPU-GPU clusters

Future Work

- Reorganize to more efficiently use CPU & GPU
- Put VRM diffusion onto GPU
- Test on distributed-memory CPU-GPU clusters

Thank you for your attention! Any questions?

mstock@applied-scientific.com