A comparison of Nek5000 and OpenFOAM for DNS of turbulent channel flow

Michael A. Sprague
Michael.A.Sprague@nrel.gov
Scientific Computing
National Renewable Energy Lab

Nek5000 Users Meeting
Argonne National Lab
10 December 2010

Matthew Churchfield, Avi Purkayastha, Patrick Moriarty, Sang Lee

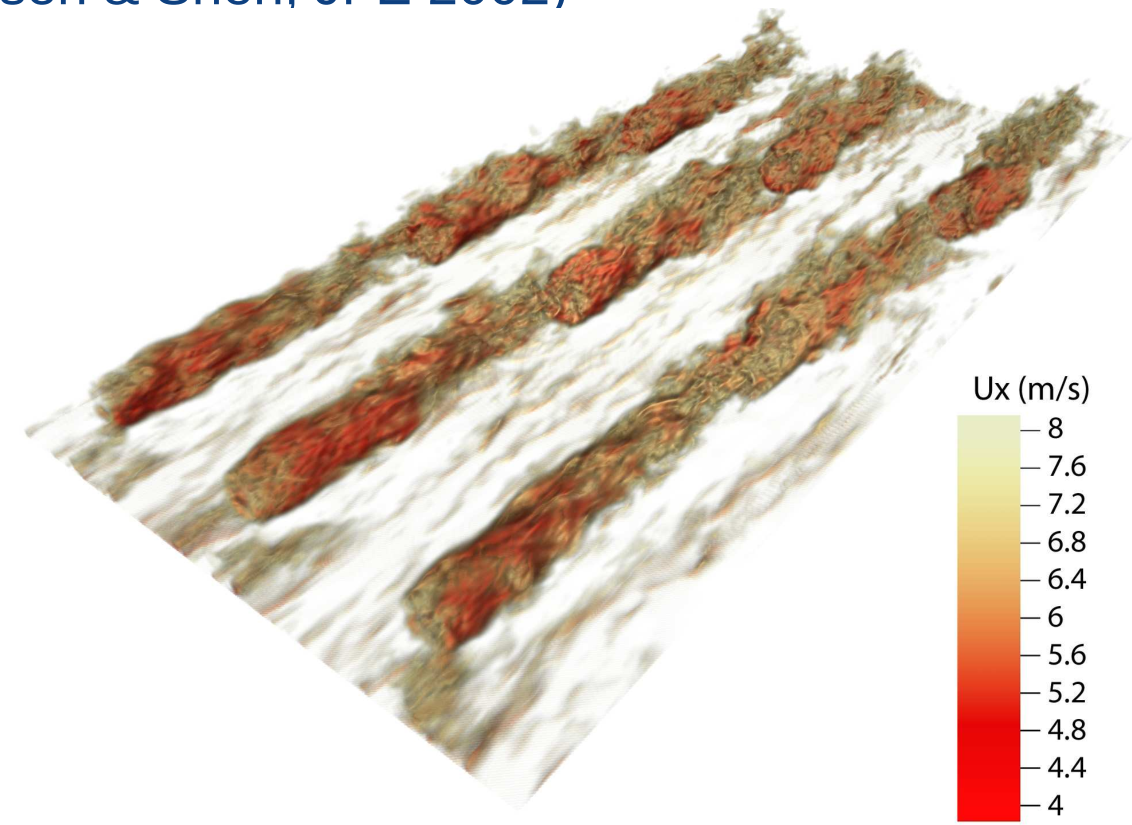
Motivating Problem

NREL LDRD Project (PI-Moriarty):

Wind Turbine Array Fluid Dynamic and Aero-elastic-simulations

- Couple Weather Research and Forecasting (WRF) code to OpenFOAM
- Couple OpenFOAM to NREL's aeroelastic design code, FAST
  - Actuator-line method (Sorensen & Shen, JFE 2002)

Example OpenFOAM simulation of 3×3 turbine array (M. Churchfield):



Motivating Questions

Question 1: Why might one use a high-order code like Nek5000 instead of a low-order code like OpenFOAM?

— Difficult to answer in general – may be problem specific, and may have many competing facets

— We focus on two sub-questions for a straight-forward DNS problem:
  - For a given spatial grid, how much more accurate is the high-order code?
  - For a given accuracy level, which method provides shorter time to solution?

Question 2: How do Nek5000 and OpenFOAM scale?

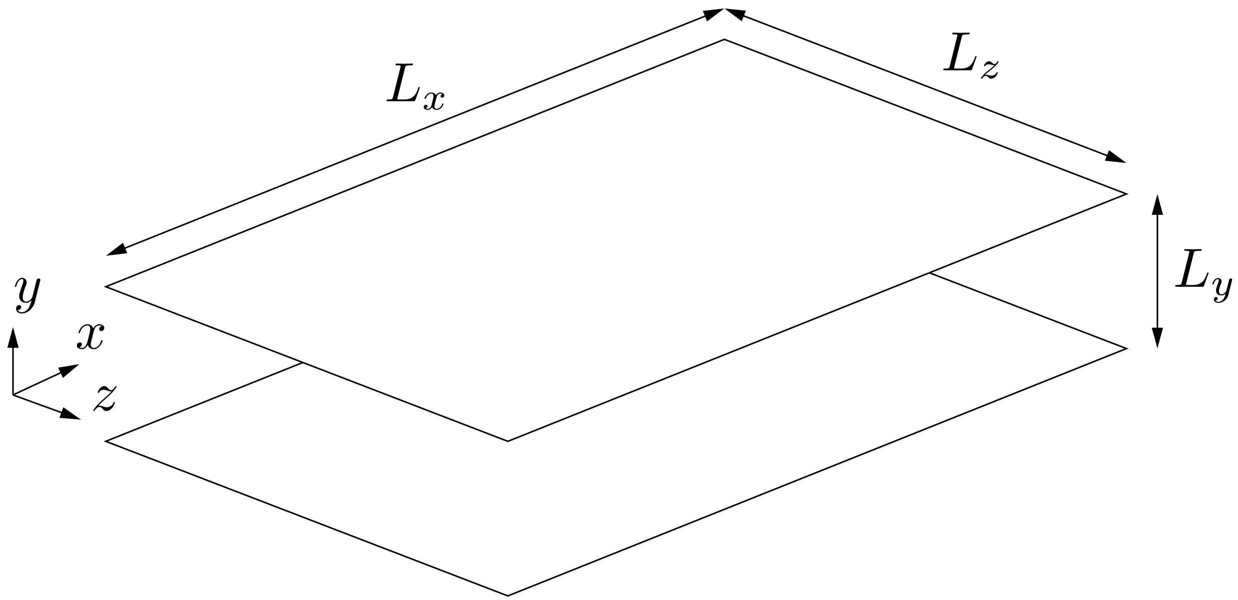
Test problem: DNS of turbulent channel Flow

The Competitors

1. Nek5000 – http://nek5000.mcs.anl.gov
   - Open-source spectral finite-element code
   - Written in Fortran77 & C with MPI communication
2. OpenFOAM – http://www.openfoam.com
   - Open-source finite-volume code
   - OpenFOAM: Open Field Operation and Manipulation
   - Written in C++ with MPI communication

KMM Channel Flow – Problem Description

- Incompressible viscous flow driven by a constant pressure gradient between two infinite parallel plates (Kim, Moin & Moser, JFM 1987)
- Computational domain: $(L_x, L_y, L_z) = (4\pi\delta, 2\delta, 2\pi\delta)$
  - no-slip boundary conditions at $y = \pm\delta$
  - periodic boundary conditions in x and z directions
- Reynolds number: $Re = \frac{U_m \delta}{\nu} = 2800$,
  U_m is the bulk mean velocity

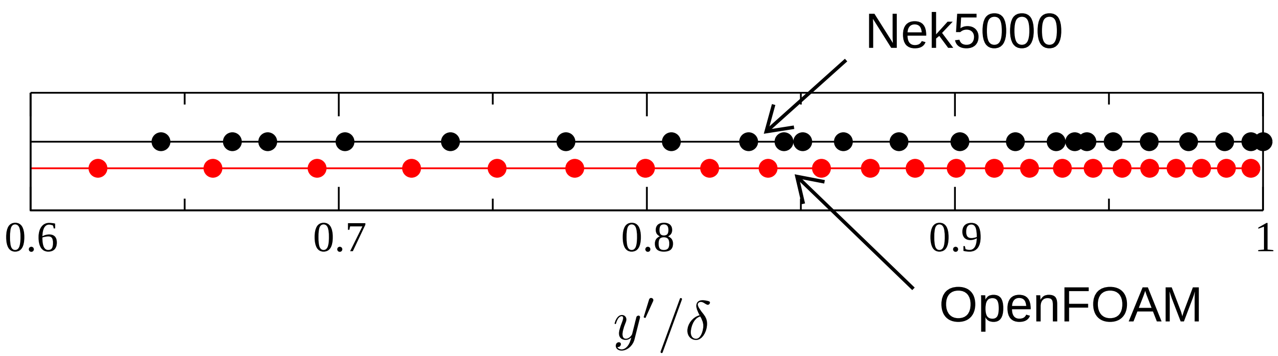


Spatial Grids

	identifier	$n_{e,x} \times n_{e,y} \times n_{e,z}$	$n_x \times n_y \times n_z$
	low	$7 \times 4 \times 6$	$50 \times 29 \times 43$
Nek5000	med	$14 \times 9 \times 11$	$99 \times 64 \times 78$
$(N = 7)$	high	$27 \times 18 \times 23$	$190 \times 127 \times 162$
	low	$48 \times 32 \times 40$	$48 \times 32 \times 40$
OpenFOAM	med	$96 \times 64 \times 80$	$96 \times 64 \times 80$
	high	$192 \times 130 \times 160$	$192 \times 130 \times 160$
KMM (1987)			$192 \times 129 \times 160$

Element sizes uniform in x and z; for y, equally-spaced-element boundaries mapped as $y'/\delta = \mathrm{sgn}(y)\,\{1 - \sinh[3.25(1 - \mathrm{sgn}(y)y/\delta)]/\sinh(3.25)\}$

Representative Grids:



Time Integration

- **Nek5000:**
  - Integrated with $C \approx 2$ (Courant number) and constant Δt
    - third-order accurate
- **OpenFOAM:**
  - PISO (Pressure-Implicit Splitting Operation) method; integrated with $C \approx 1$ and variable Δt
    - second-order accurate
- Time steps for simulations:

grid resolution	Nek5000 Δt_{const}	OpenFOAM Δt_{avg}
low	0.070	0.050
med	0.034	0.049
high	0.017	0.030

Linear-System Solvers

- **Nek5000:**
  - Pressure solve: GMRES; residual tolerance 10^{-5}
  - Velocity solve: Preconditioned CG; residual tolerance 10^{-8}
- **OpenFOAM:**
  - Pressure solve: diagonal incomplete Cholesky-preconditioned CG; divergence tolerance 10^{-6}
  - Velocity solve: diagonal incomplete LU-preconditioned bi-conjugate gradient solver; residual tolerance 10^{-8}

Initialization & Statistics Gathering

- Initial condition:

$$
\begin{aligned}
u/U_m &= 5(1-y^4)/4 + 0.3\cos(12z)e^{0.5-32.4(1-|y|)^2}(1-|y|) \\
v/U_m &= 0 \\
w/U_m &= 21.6\sin(12x)e^{-32.4(1-|y|)^2}(1-|y|)
\end{aligned}
$$

- Statistically steady turbulence established over $0 \leq (t\,U_m/\delta) \leq 200$
- Statistics gathered over $200 \leq (t\,U_m/\delta) \leq 700$

Computational Resources

Red Mesa Computational Cluster

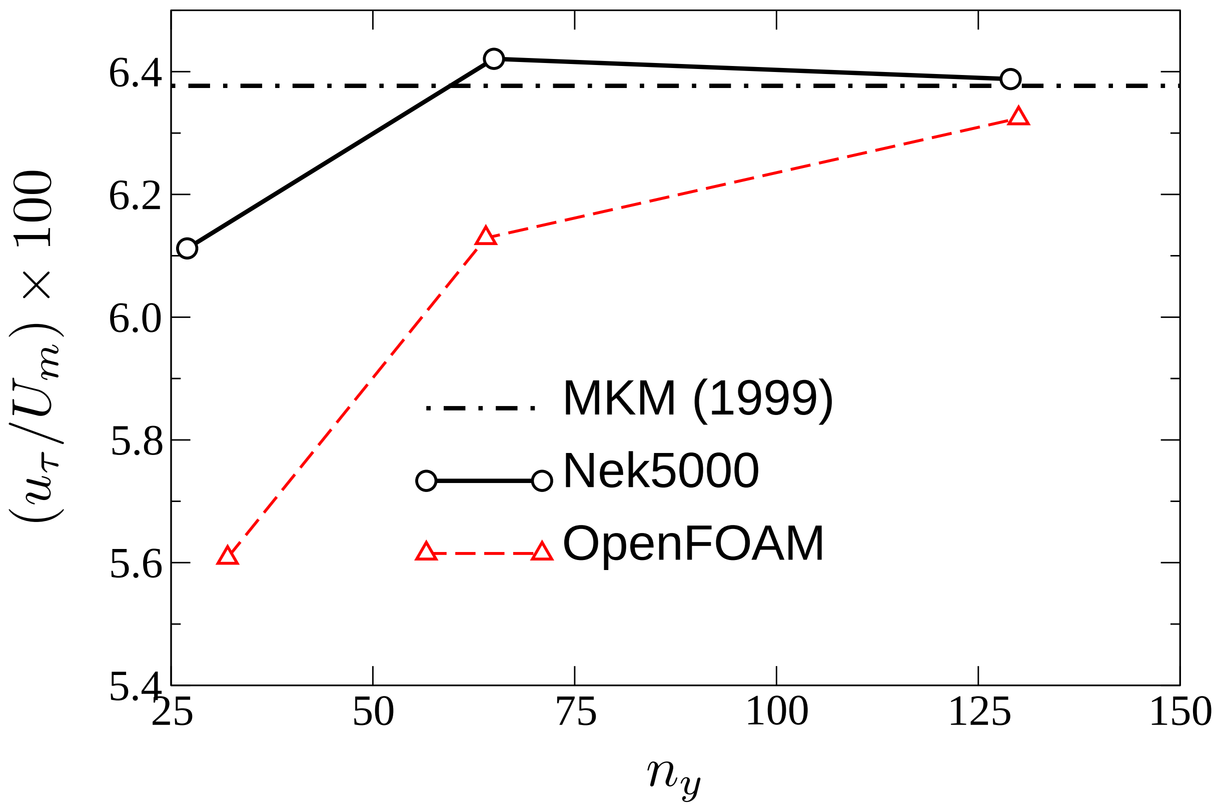
— NREL's 15,360 core HPC system

— Intel Xeon 5570 "Nehalem" cores

— Quad-Data-Rate Infiniband interconnect arranged in a 3D torus topology

— Lustre file system with about 1 petabyte of total storage

Results: Friction Velocity

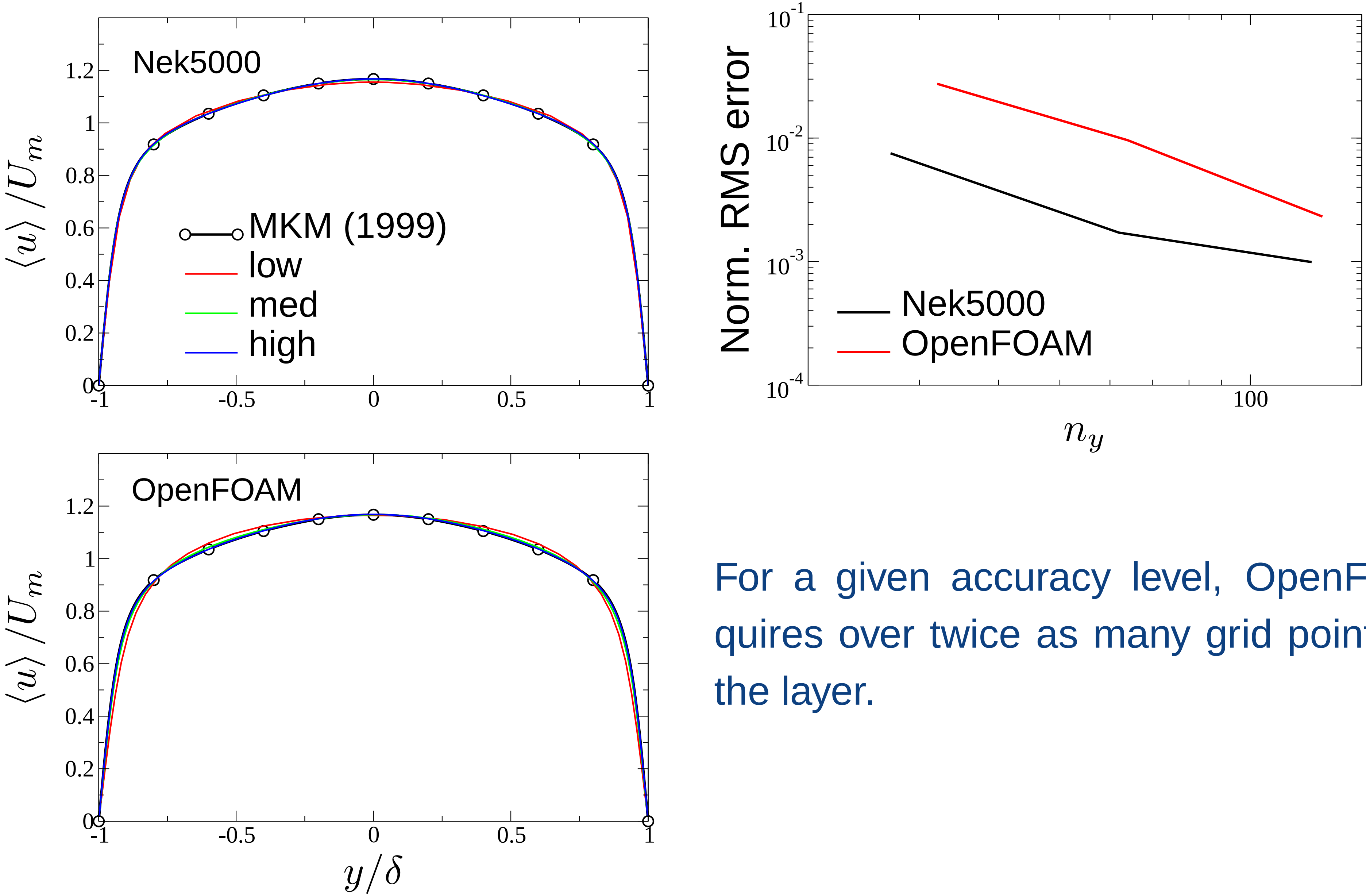


$$u_\tau = \sqrt{\nu \left. \frac{\partial \langle u \rangle}{\partial y} \right|_{y=-\delta}}$$

n_y: # grid points across layer

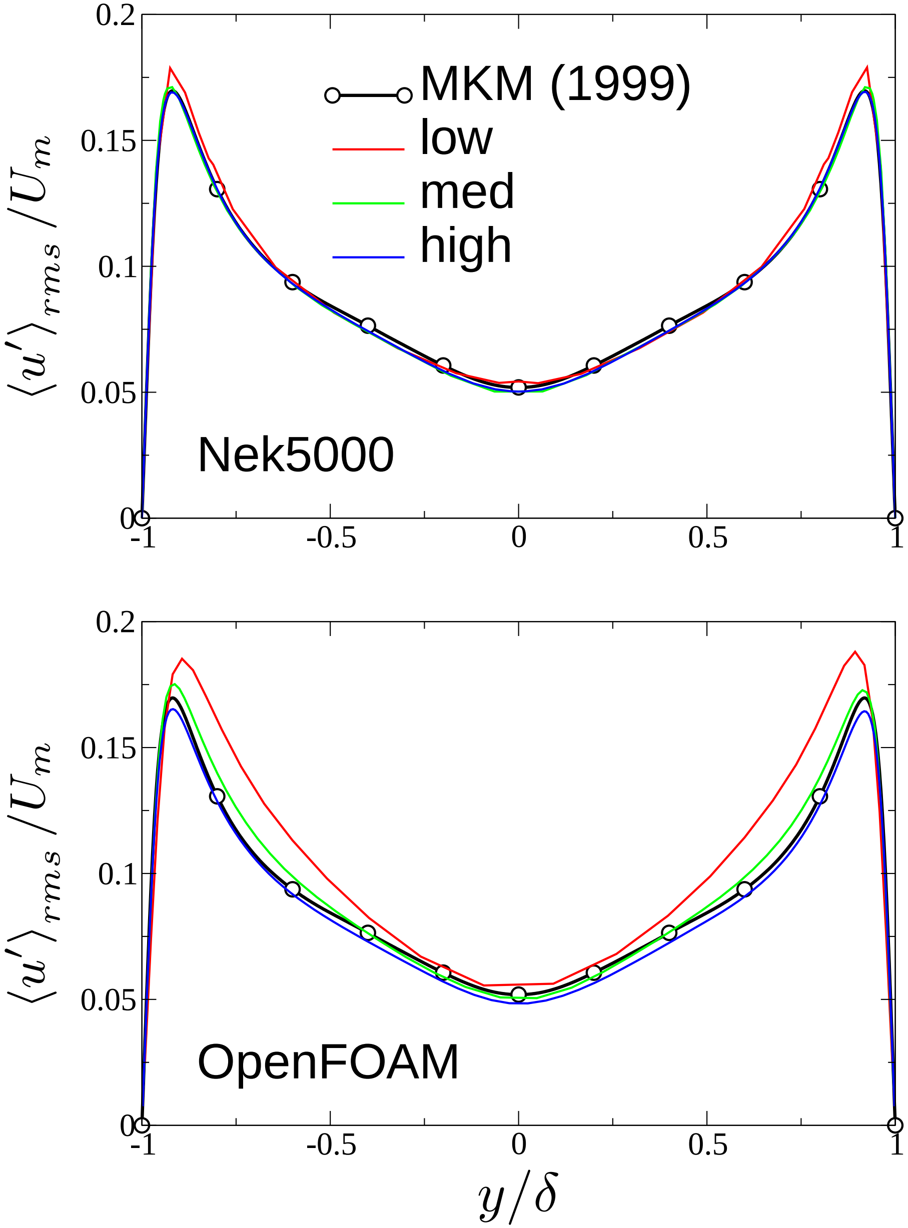
- Benchmark results: Moser, Kim & Mansour, 1999 *Physics of Fluids*
- For a given accuracy, OpenFOAM needs approximately twice as many grid points across layer
  $\rightarrow$eight-times as many grid points in 3-D

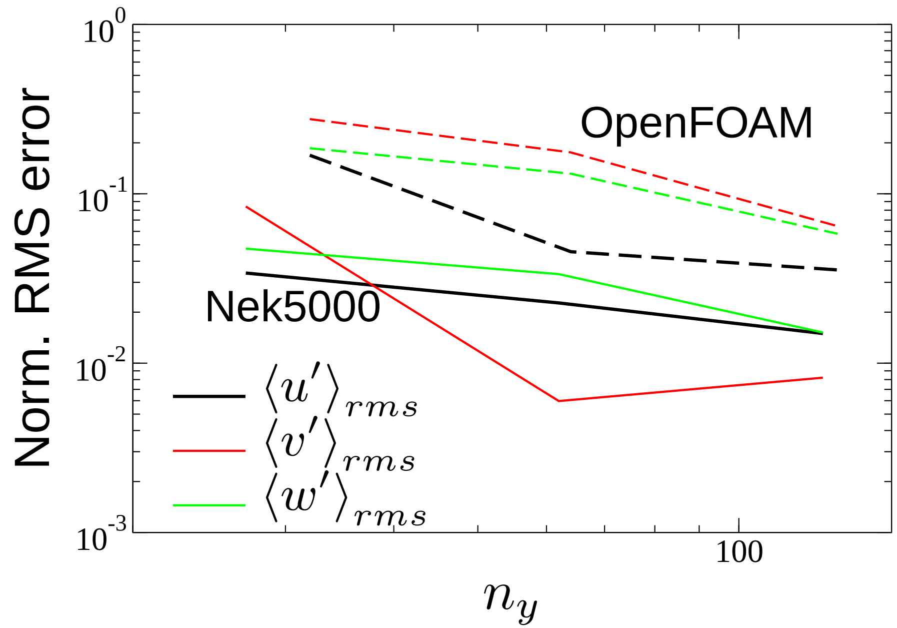
Results: Mean-Velocity Profiles



For a given accuracy level, OpenFOAM requires over twice as many grid points across the layer.

Results: RMS-Velocity Profiles





Here, better accuracy of Nek5000 even more pronounced.

Scaling Study: Model Parameters

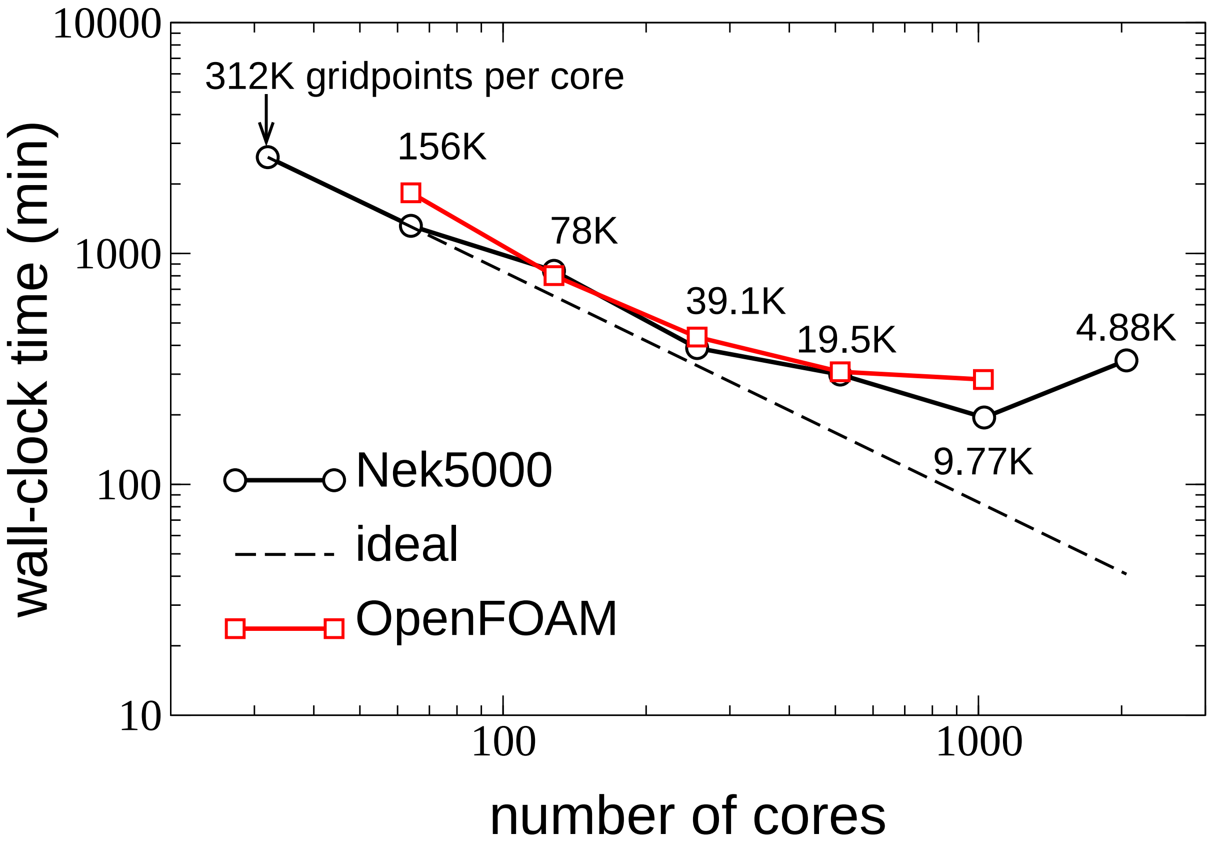
	$n_x \times n_y \times n_z$	grid pts.	$\Delta t\, U_m/\delta$	C	# time steps
Nek5000	$253 \times 183 \times 218$	$\approx 10^7$	0.012	≈ 2	8333
OpenFOAM	$262 \times 178 \times 218$	$\approx 10^7$	0.018 avg	≈ 1	≈ 5430

— Simulations run for $200 \leq (t\, U_m/\delta) \leq 300$

— I/O minimized; statistics gathering disabled

— number of cores: 32 – 2048

Results: Wall-Clock Time



— Nek5000 and OpenFOAM exhibit similar scaling and wall-clock times for the same number of gridpoints

— Good scaling exhibited when gridponts-per-core $\gtrsim 50K$

Summary & Conclusions

- For a given number of gridpoints, Nek5000 (with $N = 7$) is significantly more accurate than OpenFOAM
  - OpenFOAM requires over eight times the number of gridpoints to achieve similar accuracy
- For a given number of gridpoints, Nek5000 and OpenFOAM show similar strong scaling and wall-clock times
  - For a given accuracy, time-to solution is significantly less for Nek5000
- OpenFOAM linear-system solvers are too inefficient
  - Better options in the *Extended Project* version of OpenFOAM http://www.extend-project.de/

Comments on Visualization

Python scripts for converting Nek5000 .fld/.fXXX files to .vtk/.vtu

– work in progress; uses tvtk wrapper

– “wrapped” Fortran for speed

Mayavi2 (built on python) for visualization

http://code.enthought.com/projects/mayavi/