KTH ROYAL INSTITUTE
OF TECHNOLOGY

FEniCS HPC: An automated predictive high-performance framework for multiphysics simulations

Niclas Jansson

Acknowledgments

FEniCS-HPC

- Prof. Johan Hoffman (KTH, BCAM, Adaptive)
- Asst. Prof. Johan Jansson (KTH, BCAM)
- Dr. Aurélien Larcher (NTNU, Adaptive)
- Rodrigo Vilela de Abreu (Adaptive)
- Jeannette Hiromi Spühler (KTH, Adaptive)
- Niyazi Cem Degirmenci (BCAM)
- Dr. Van Dang Nguyen (KTH)
- Massimiliano Leoni (KTH)
- Ezhilmathi Krishnasamy (BCAM)
- Frida Svelander (KTH)

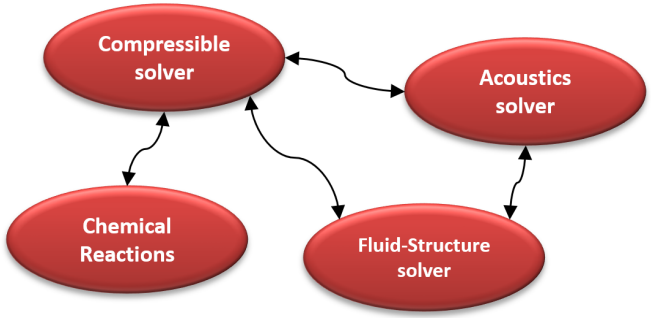
Challenges for Industrial Applications

Mesh generation for complex geometries

- High resolution
- Unstructured meshes
- Optimal element placement
- Memory footprint

Multiphysics problems

- Coupling of different simulation frameworks
- Different representation (Euler/Lagrange)
- Different data structures
- Difficult to achieve good performance



Automated scientific computing

The FEniCS project

- Seeks to automate the scientific software process
- High level language input of equation and discretization method

$$\Delta u = f \quad \text{in}\,\Omega$$

$$\int_\Omega \Delta u\, v\, dx = \int_\Omega f\, v\, dx$$

$$\int_\Omega \nabla u \nabla v\, dx = \int_\Omega f\, v\, dx + \int_\Gamma \frac{\partial u}{\partial \nu}\, v\, ds$$

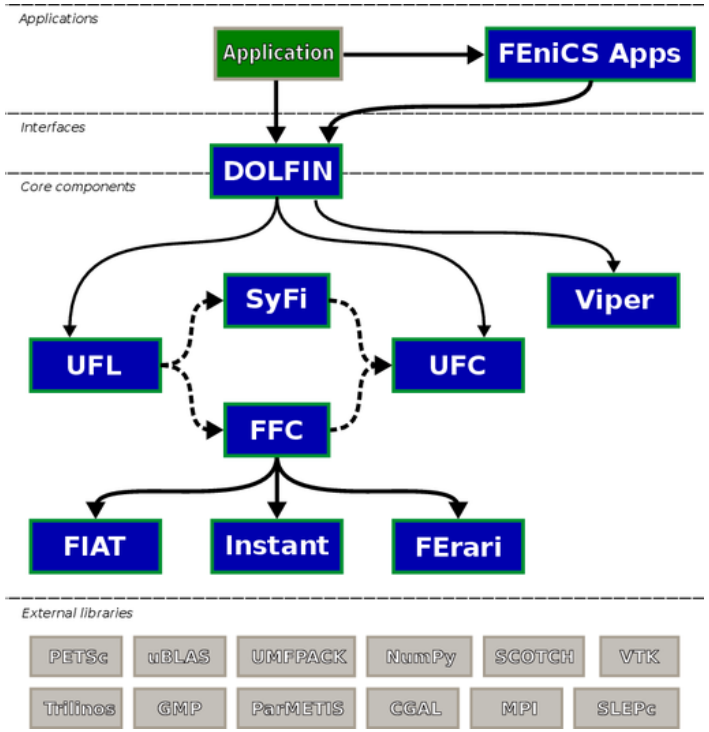
```
element = FiniteElement("Lagrange","triangle",1)

v = TestFunction(element)
u = TrialFunction(element)
f = Function(element)
g = Function(element)

a = dot(grad(v), grad(u))*dx
L = v*f*dx + v*g*ds
```

- Low-level (C++) generated by the FEniCS compiler

Automated scientific computing



DOLFIN Dynamic Object-oriented Library for FINite element computation

FFC FEniCS Form Compiler

FIAT FInite element Automatic Tabulator

UFL Unified Form Language

Automated scientific computing

Poisson's equation

$$
\begin{aligned}
-\nabla^2 u &= f, \quad \text{in}\,\Omega \\
u &= 0, \quad \text{on}\,\Gamma_D \quad \{(0,y)\cup(1,y)\subset\partial\Omega\} \\
\nabla u\cdot n &= g, \quad \text{on}\,\Gamma_N \quad \{(x,0)\cup(x,1)\subset\partial\Omega\}
\end{aligned}
$$

Find $u \in V$ such that

$$a(u,v) = L(v) \quad \forall v \in V,$$

where,

$$
\begin{aligned}
a(u,v) &= \int_\Omega \nabla u\,\nabla v\,dx \\
L(v) &= \int_\Omega f\,v\,dx + \int_{\Gamma_N} \frac{\partial u}{\partial \nu}\,v\,ds
\end{aligned}
$$

Automated scientific computing

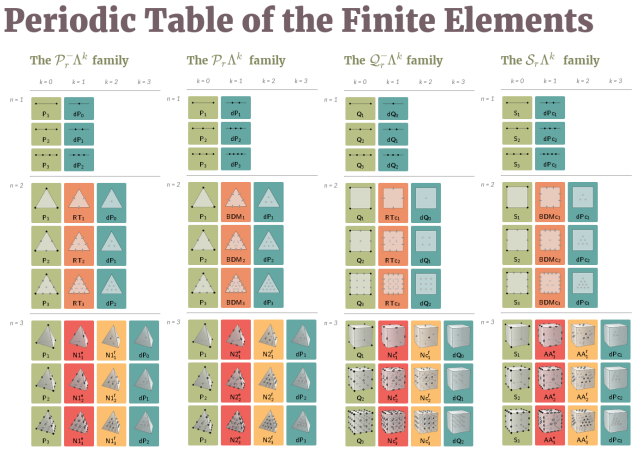
Unified Form Language

UFL provides a language for expressing $a(u, v)$ and $L(v)$ (tree like)

```
V = FunctionSpace(mesh, "Lagrange", 1)
u = TrialFunction(V)
v = TestFunction(V)
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)")
g = Expression("sin(5*x[0])")
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds
```

- Easy to define large and complex equations
- No need to change a large amount of code when changing an equation or element type
- UFL implements several different kinds of elements, easy to change

```
V = FunctionSpace(mesh, "Lagrange", 2)
V = FunctionSpace(mesh, "Lagrange", 4)
```



- https://femtable.org

Automated scientific computing

FInite element Automatic Tabulator

Generates the finite element space to evaluate on the reference cell and possibly the mapping

FEniCS Form Compiler

- Loops on the tree and inserts the FIAT contribution
- Spits out code using snippets (provided by consumers)

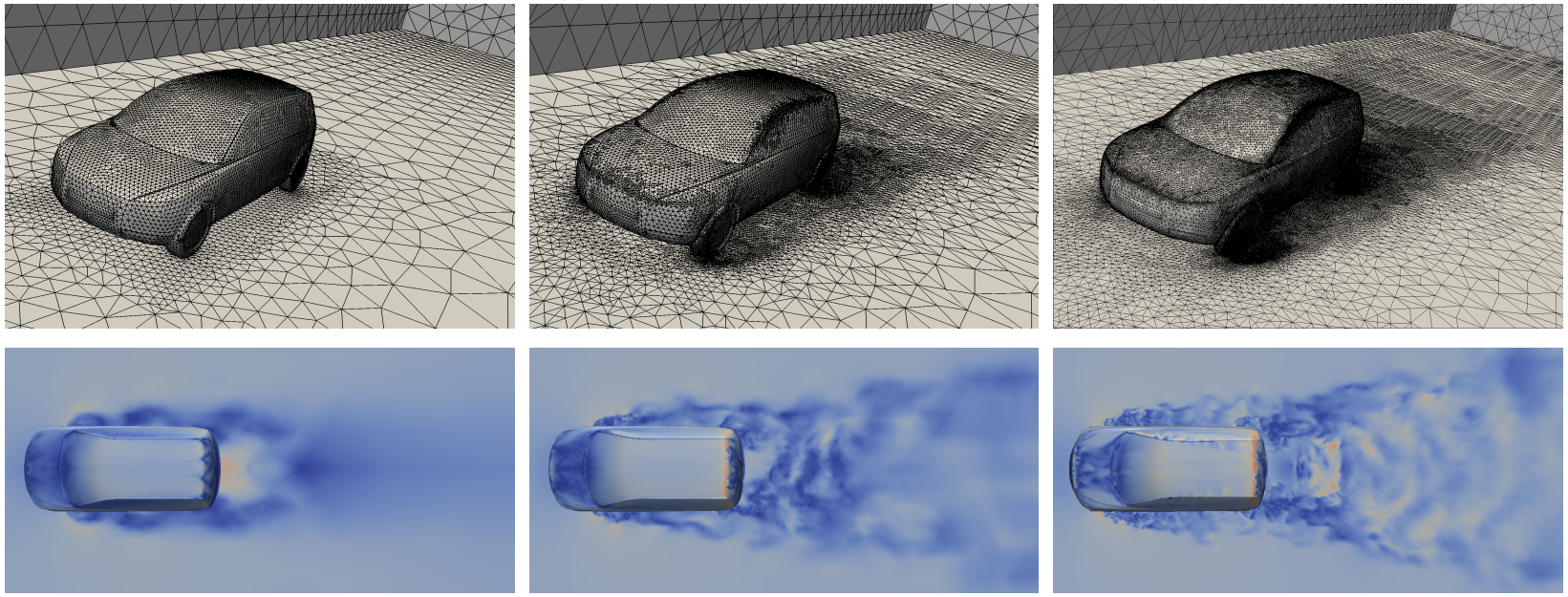
FEniCS Code Generation

- Everything is implemented as python modules
- Great for “std” FEniCS (JIT of equations, solvers in python)
- Not so great for certain arch. BG/L, K computer...
- Code generation only concerns evaluation of element integrals
- Parallelization of discretization and linear solvers left to consumers

FEniCS-HPC

- Code generation: FIAT (FErari) FFC UFC UFL
- DOLFIN-HPC
  - HPC branch of FEniCS's FEM framework DOLFIN
  - Portable: MIPS, PowerPC, SPARC, ia64, x86, ...
  - Hybrid parallelization (MPI + X)
  - Fully distributed mesh
  - Parallel I/O (MPI I/O)
  - Parallel adaptive mesh refinement
  - Predictive (multi constrained) dynamic load balancer
  - Insitu processing (libsim/VisIt)
  - External libraries for linear algebra (e.g. PETSc)
- Unicorn
  - A posteriori error estimation based on adjoint problems
  - Parameter-free turbulence modeling
  - Fluid-structure interaction
  - ALE moving meshes

Adaptive error control



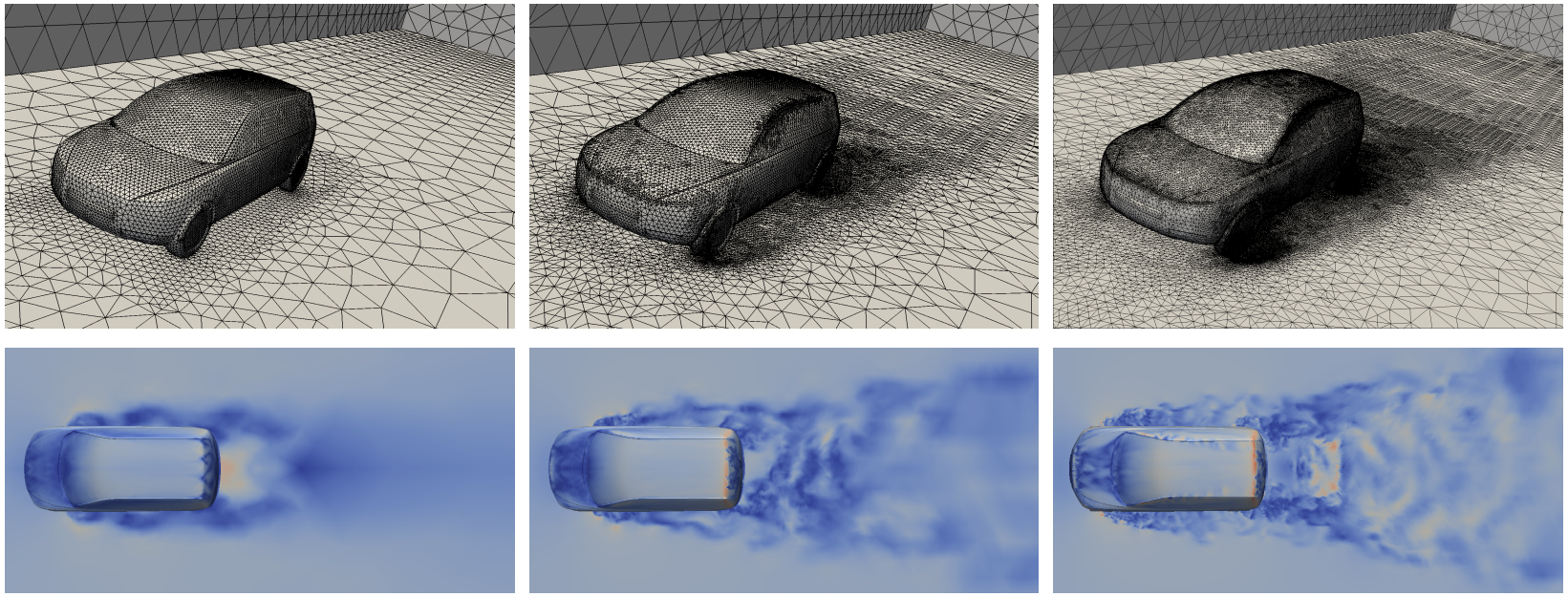
A posteriori error estimate

$$|M(\hat{u}) - M(\hat{U})| \leq \sum_{n=1}^{N} \Big[\int_{I_n} \sum_{K \in \mathcal{T}_n} |R_1(\hat{U})|_K \cdot \omega_1 \, dt + \int_{I_n} \sum_{K \in \mathcal{T}_n} |R_2(U)|_K \, \omega_2 \, dt \Big]$$

$$R_1(\hat{U}) = \dot{U} + (U \cdot \nabla) U + \nabla P - \nu \Delta U - f$$

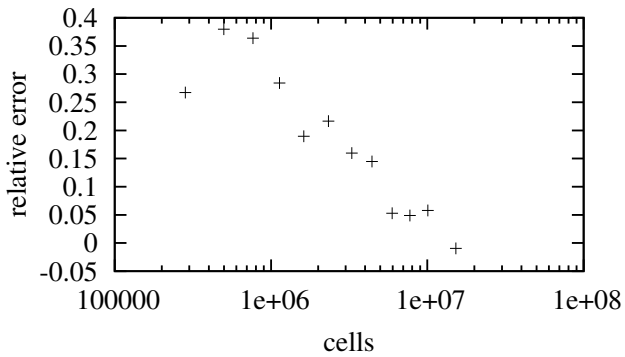
$$R_2(U) = \nabla \cdot U$$

Adaptive error control



Adaptive algorithm

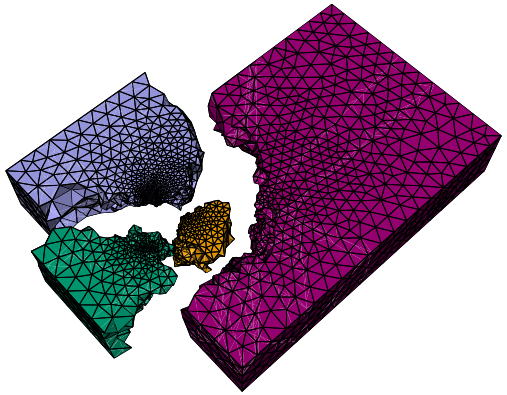
1 Solve the flow problem
2 Compute *a posteriori* errors $\mathbf{e}_K$
3 If $\mathbf{e}_K$ is less than a given tolerance, stop
4 Refine elements K where $\mathbf{e}_K$ is large, restart



Parallelization

Element based decomposition

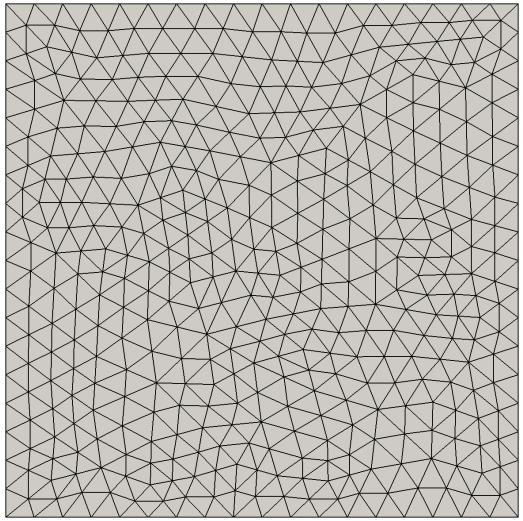
- Dual graph based partitioning
- Overlap represented as ghosted entities
- Low data dependency during FE assembly
- Fully distributed mesh
  - Single input file
  - All pre/post proc. steps performed in parallel
  - Entire problem nor mesh rep. on a single core

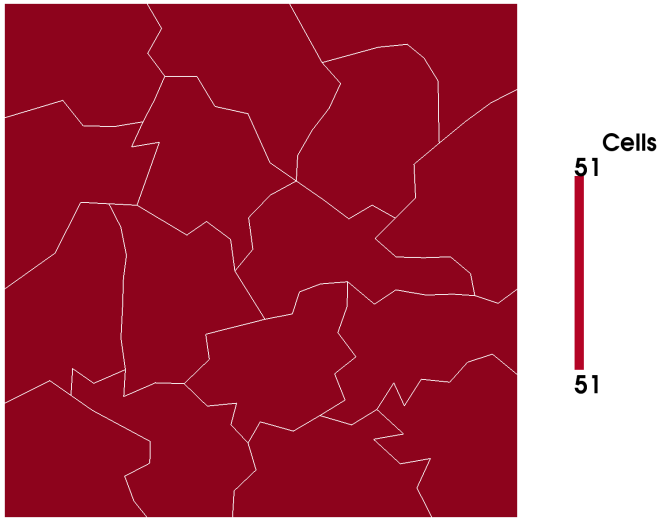


Parallel Mesh Refinement

Recursive Rivara Longest Edge Bisection

- Two phase process, local refinement and propagation
- PEs can change state between “active” and “idle”
- Non-centralized termination detection algorithm

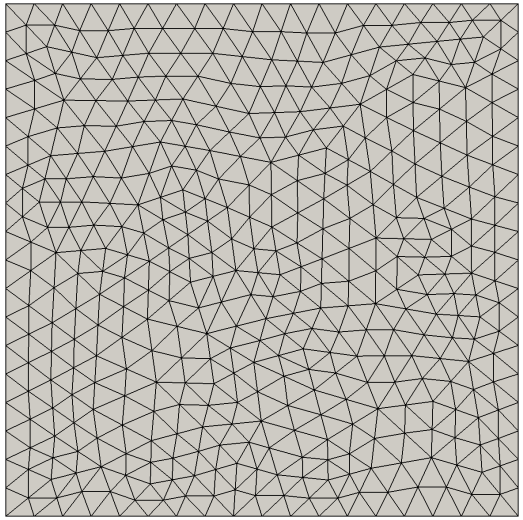


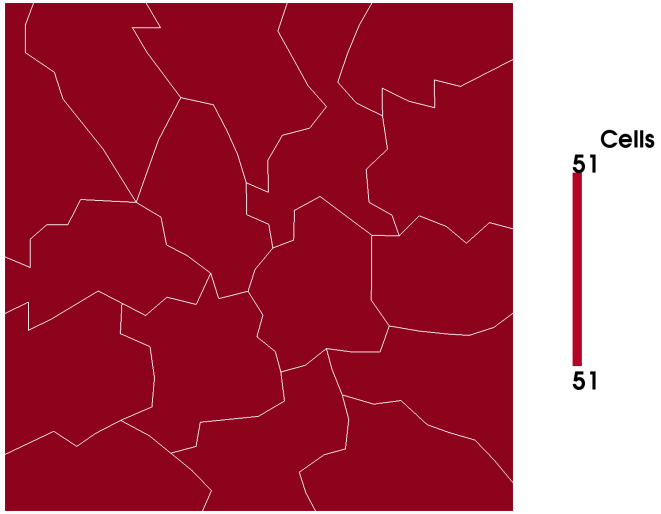


Parallel Mesh Refinement

Predictive Dynamic Load Balancer

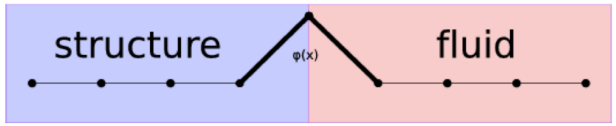
- Scratch and remap approach
- Weighted dual graph (predicted future workload)
- Longest-edge propagation paths to unroll recursion



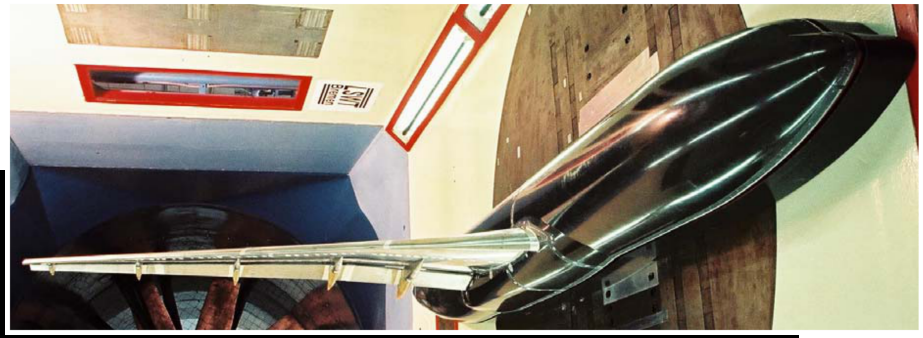


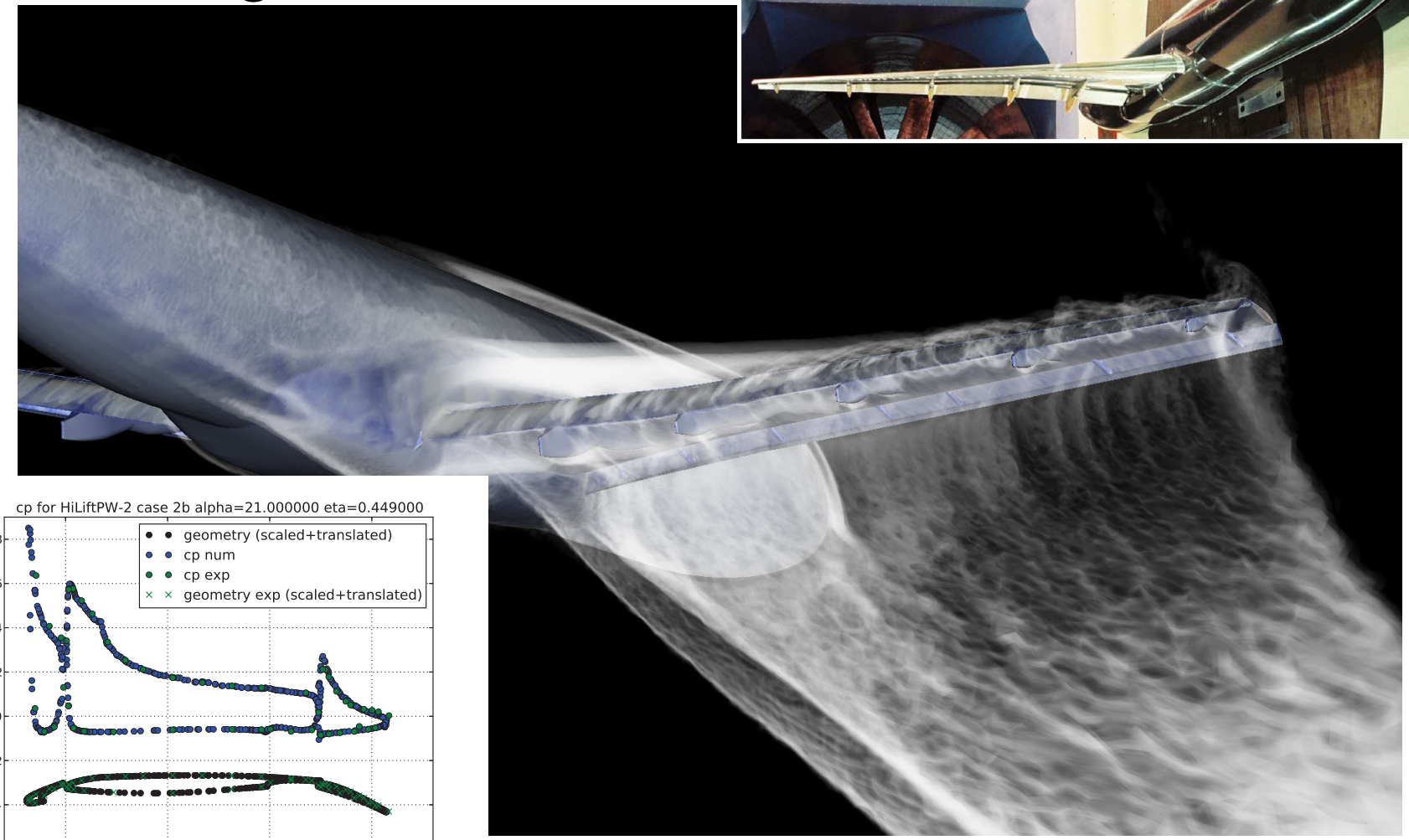
Fluid Structure Interaction

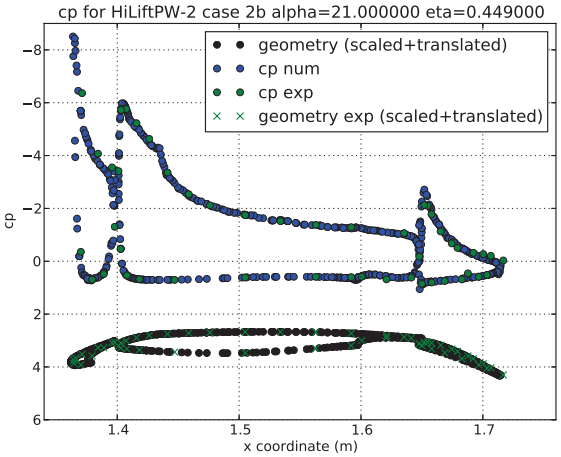
- Phase function $\varphi(x)$ marking fluid/solid
- Ekional equations for contact modeling
- Everything implemented in the form files



High Lift Prediction

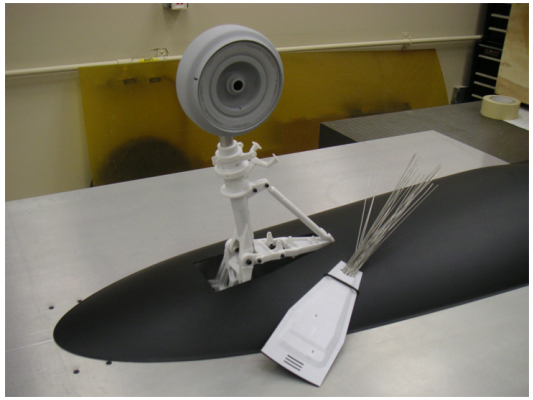






Aeroacoustic sources for a landing gear

A benchmark problem in connection with the 18th AIAA/CEAS Aeroacoustics Conference.

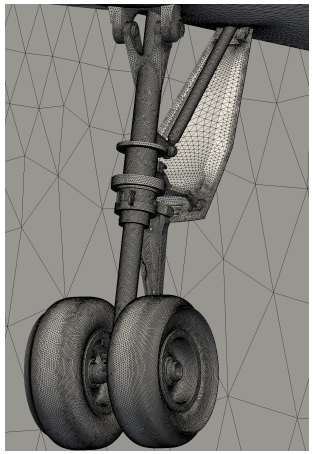


- Gulfstream G550 nose landing gear
- Initial coarse mesh 3.6M elements
- Final mesh 23.8M elements
- Ref. target: Lighthill's source term

Computational resources

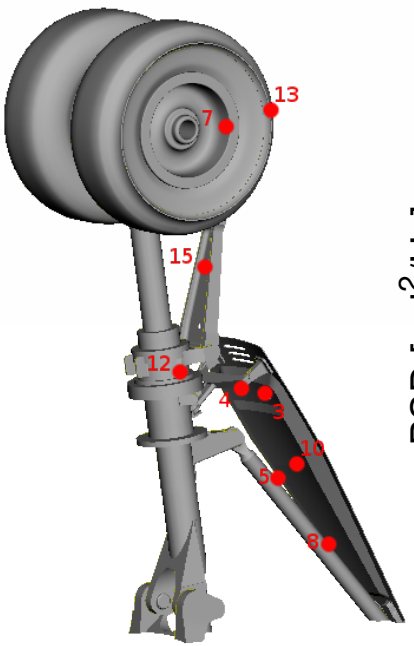
Computed on Lindgren a Cray XE6

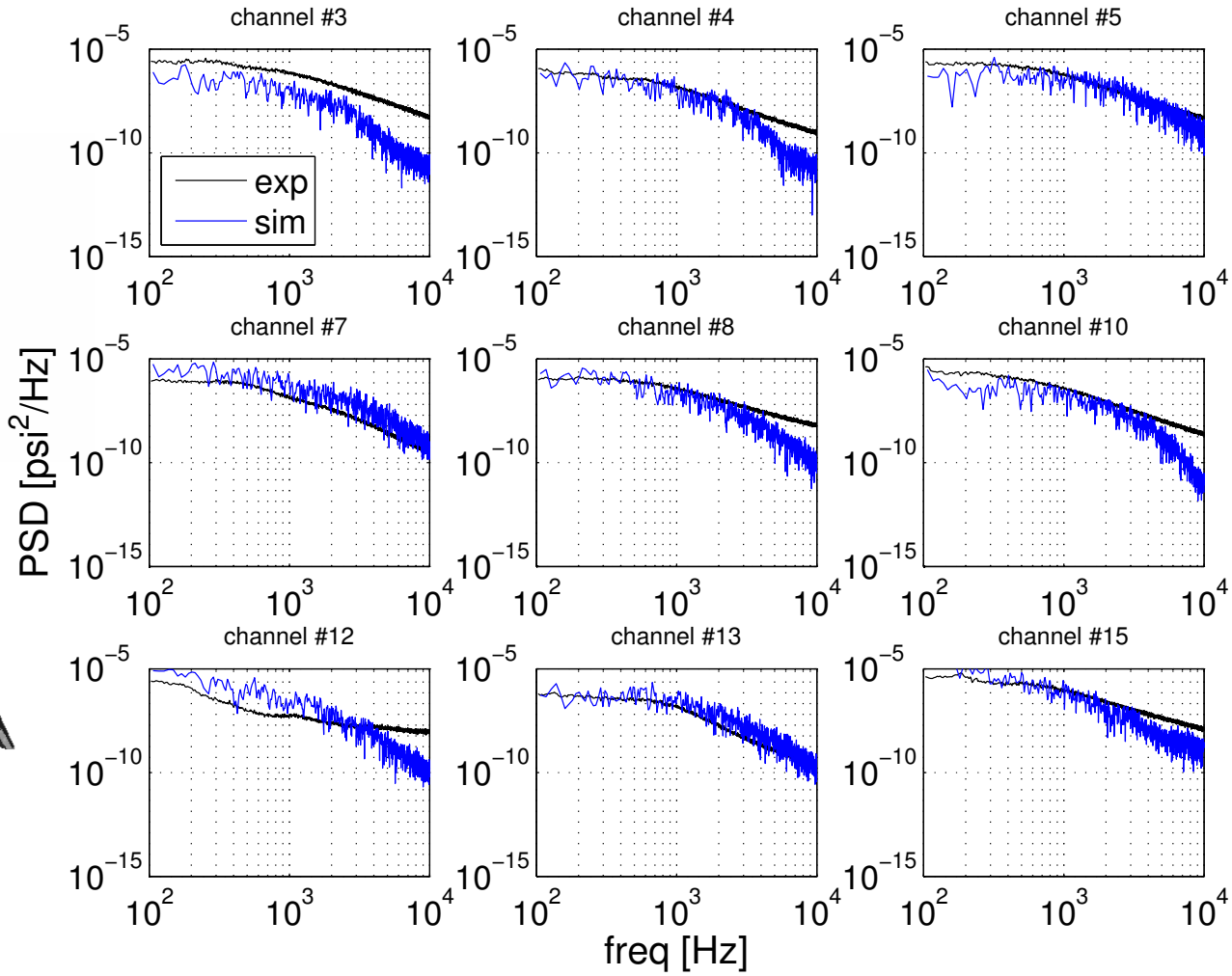
- 900k core hours for the adaptive process (various #PEs)
- 600k core hours for the final run (2304 PEs)
- 1.5M core hours in total (NASA used approx. 1.7M)
- 1.4TB of raw simulation data (final run)



Aeroacoustic sources for a landing gear

Near Field, dynamic pressure





Adaptive FEM for noise

Incompressible Navier-Stokes + Lighthill's wave eq.

$$\dot{u} + (u \cdot \nabla)u + \nabla p - \nu \nabla \cdot \epsilon(u) = f \tag{1}$$

$$\nabla \cdot u = 0$$

$$\ddot{\rho} - c_0^2 \Delta \rho = \nabla \cdot (\nabla \cdot T). \tag{2}$$

Adaptive algorithm

1. For mesh $\mathcal{T}_n$: compute Navier-Stokes and adjoint.
2. For mesh $\mathcal{T}_n$: compute Lighthill's equation and adjoint.
3. Mark ca. 10% of the elements with highest "error indicator" for refinement for each equation.
4. Generate the refined mesh $\mathcal{T}_{n+1}$, and goto 1.

Adaptive FEM for noise

What advantages?

- Fully automated, solution-based mesh generation.
- Final mesh has “optimal” size, avoids “over refinement”.
- No need for a “mesh study”.
- No need to choose porous or solid surfaces for integration.
- Less I/O operations.
- No pre-processing of acoustic sources.

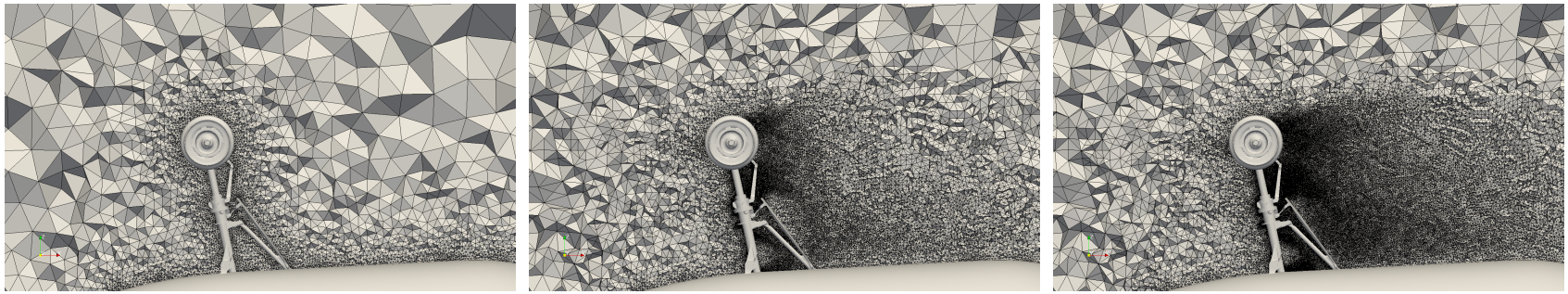


Figure: Initial mesh, and mesh after 4 and 6 adaptive refinements.

Adjoint Solution I, flow

Cells marked, Navier-Stokes equations only

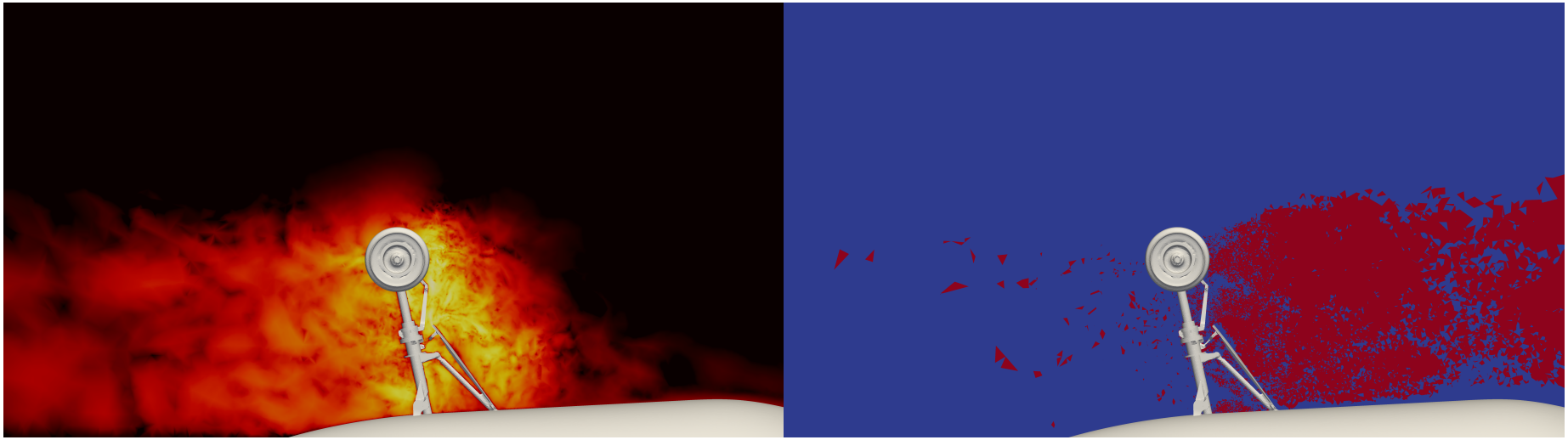


Figure: Snapshot of adjoint velocity (*left*), and cells marked for refinement (*right*).

Adjoint Solution II, flow + wave

Cells marked, Navier-Stokes + Lighthill

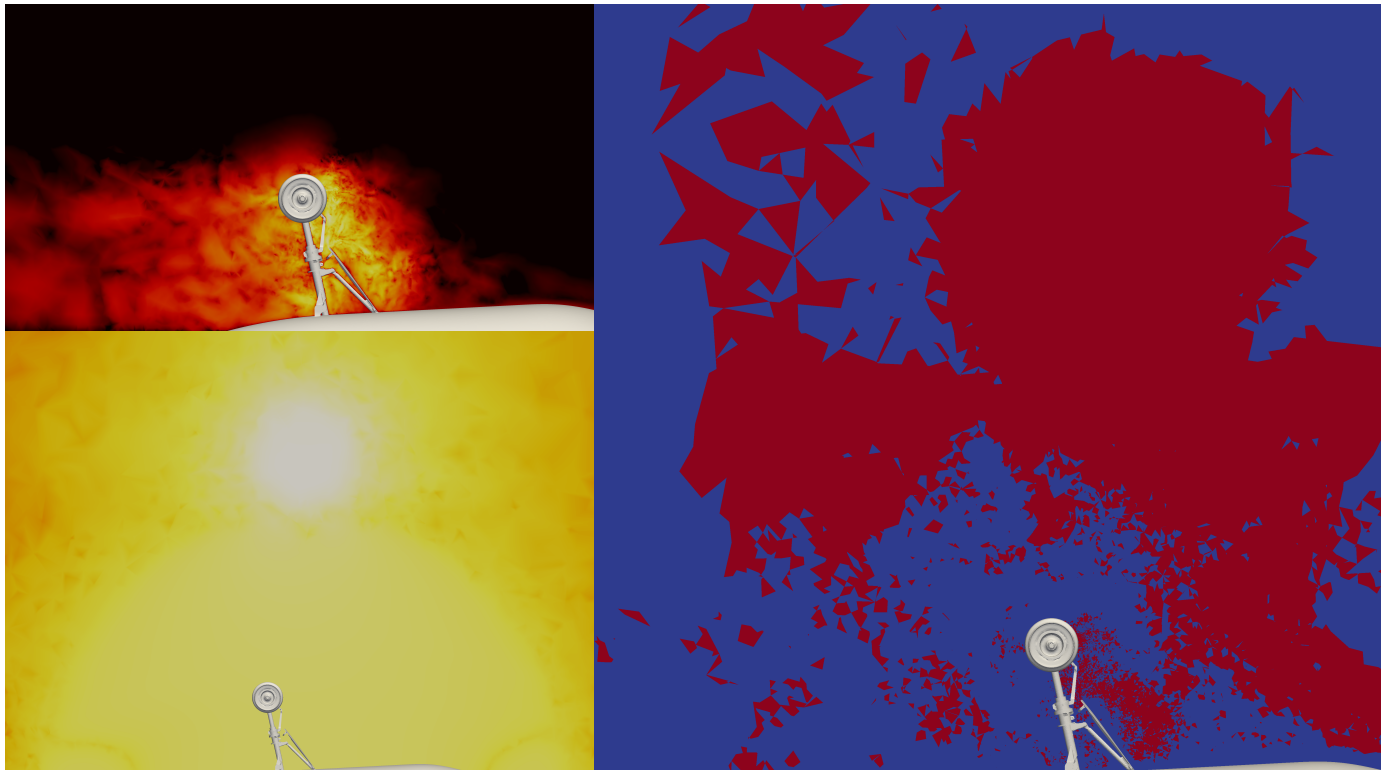


Figure: Snapshot of adjoint velocity (*upper left*), and adjoint density (*lower left*), and cells marked for refinement (*right*).

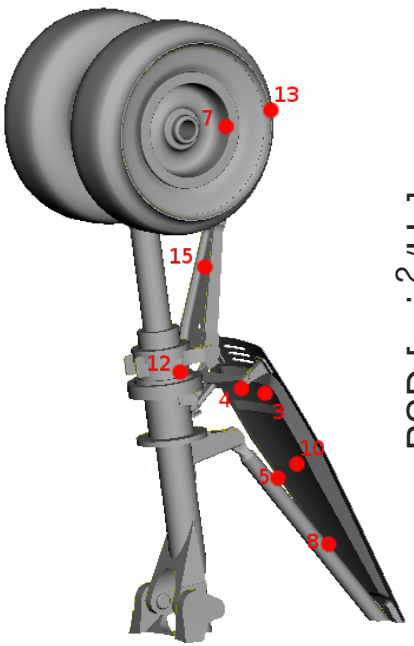
Aeroacoustic sources for a landing gear

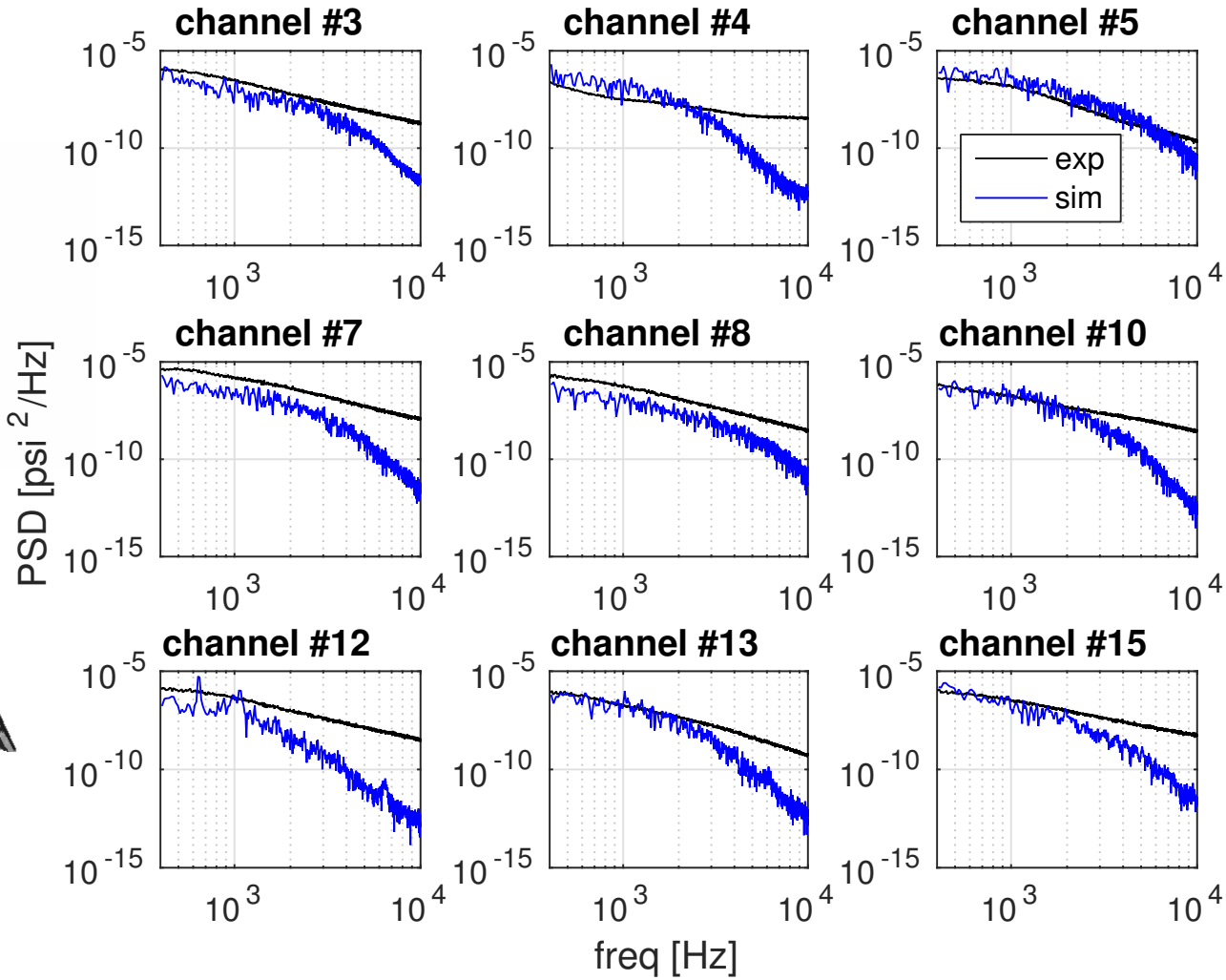
- New simulations for the 22th AIAA/CEAS Aeroacoustics Conference.
- UFAFF wind tunnel configuration
- Computed on Beskow a Cray XC40 system at PDC/KTH.
- Used resources:

	T_{sim}	T_{wall}	# *cores*	*core-hours*
start-up	0.24s	2 days	4,096	$\sim 200,000$
sampling	0.19s	1.5 days	4,096	$\sim 150,000$
adaptive ref	-	-	160-1,024	$\sim 100,000$

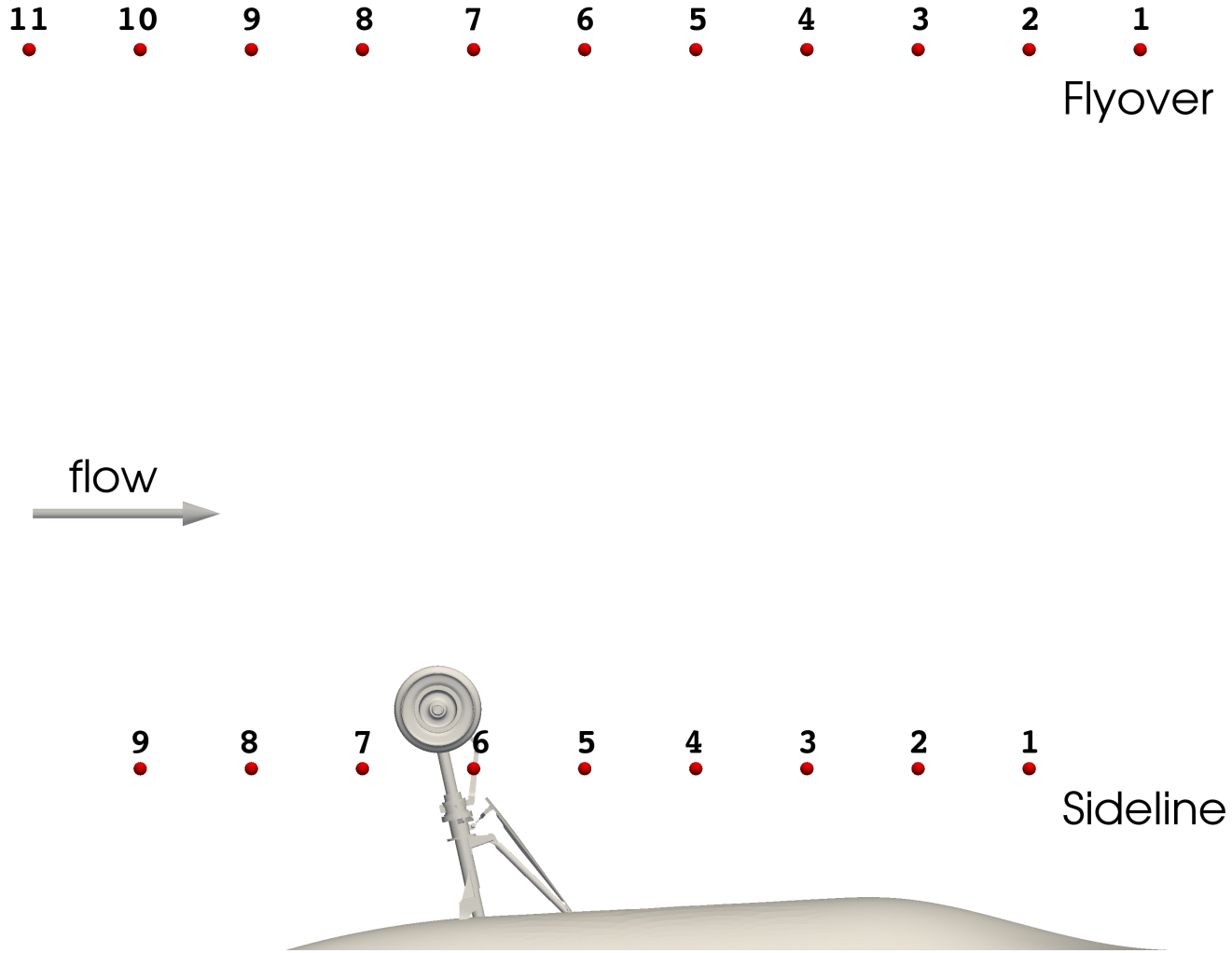
- Mesh sizes
  Coarsest mesh: 200K vertices (1M cells)
  Finest mesh: 12M vertices (66M cells)

Near Field, dynamic pressure





Far Field, sound pressure



Far Field, sound pressure

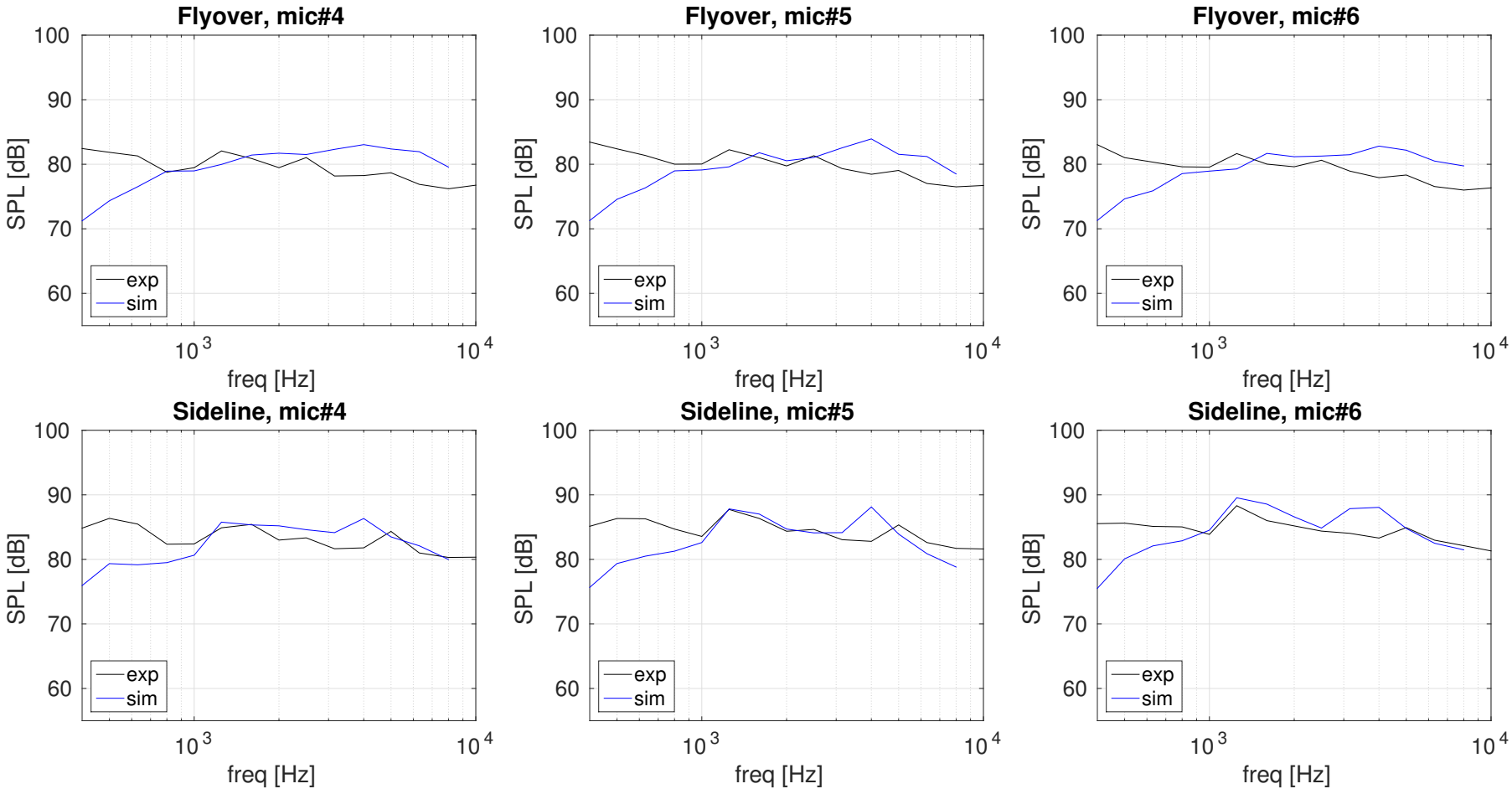
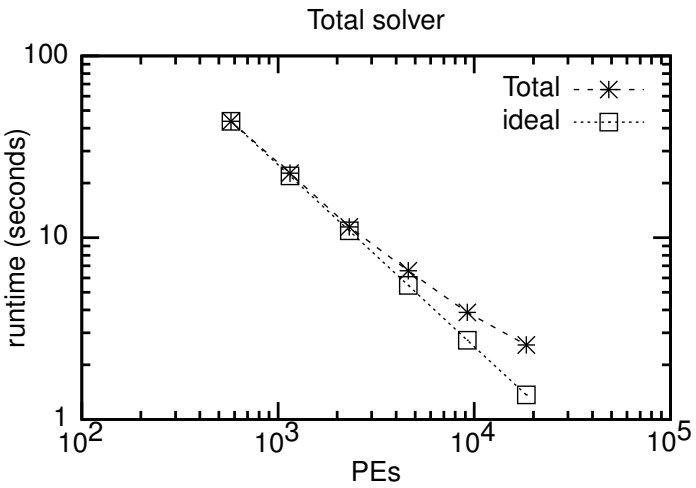
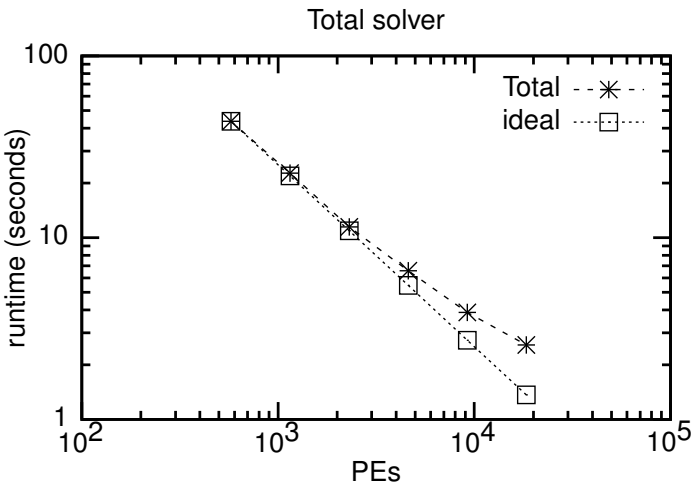


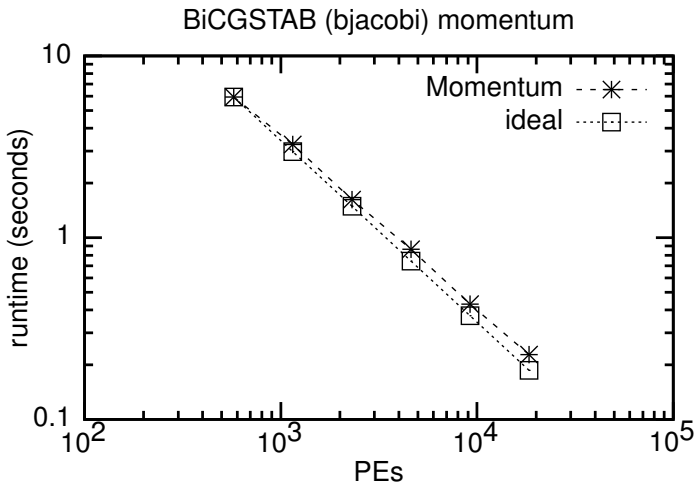
Figure: Far field Sound Pressure Level (SPL) for selected microphones.

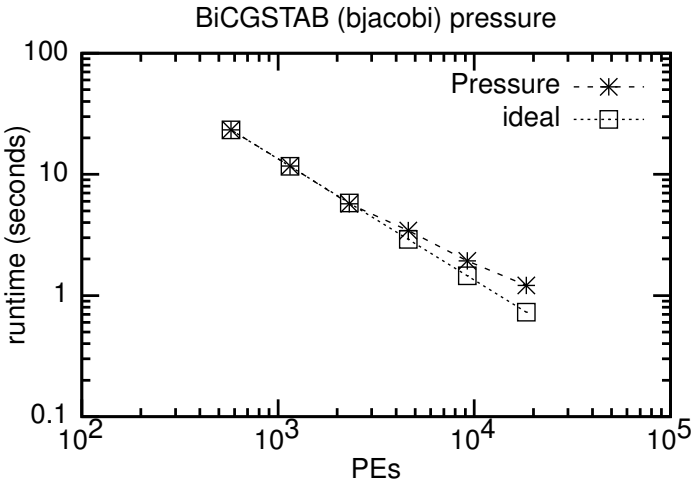
Strong scaling (DLR F11)



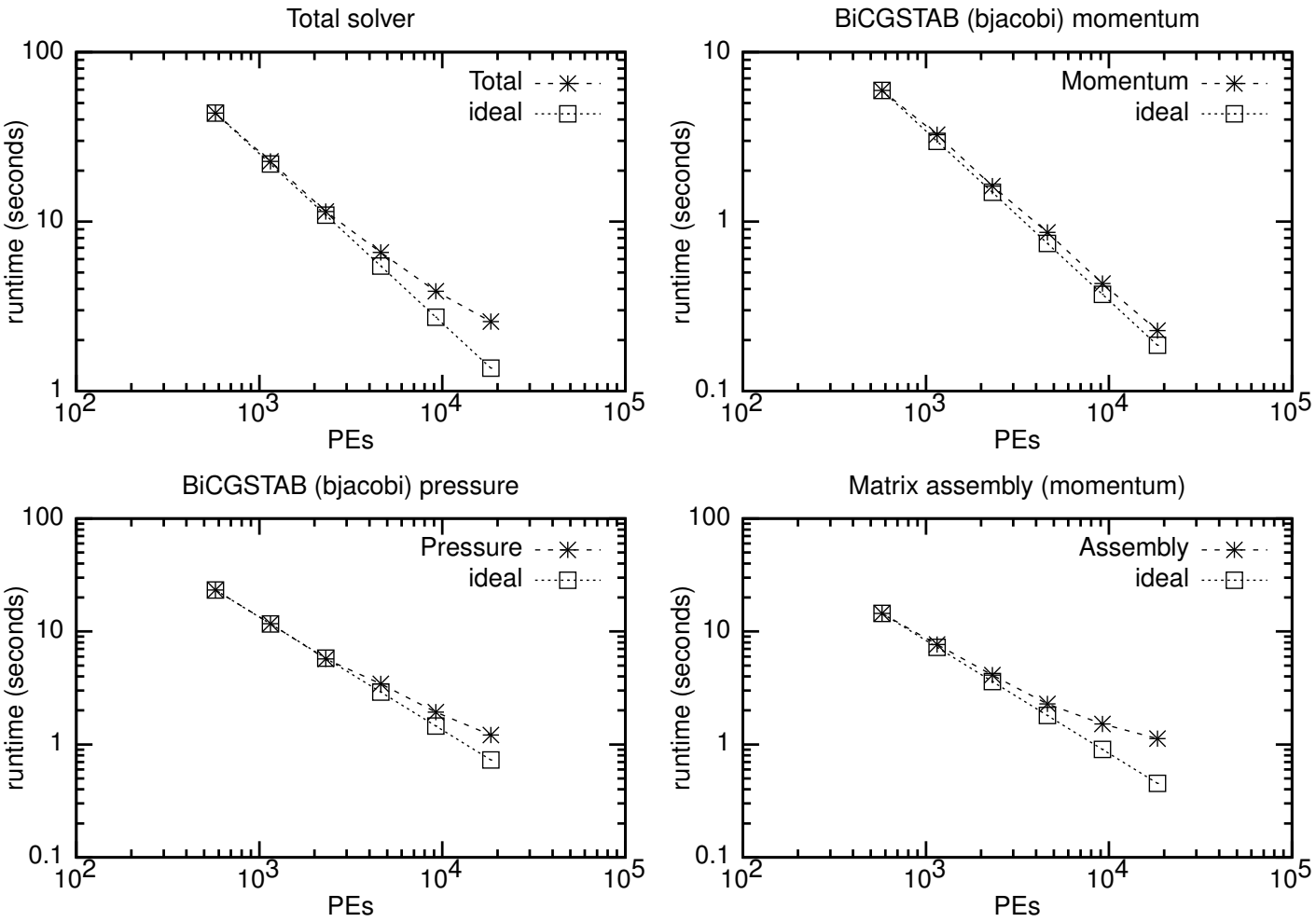
Strong scaling (DLR F11)







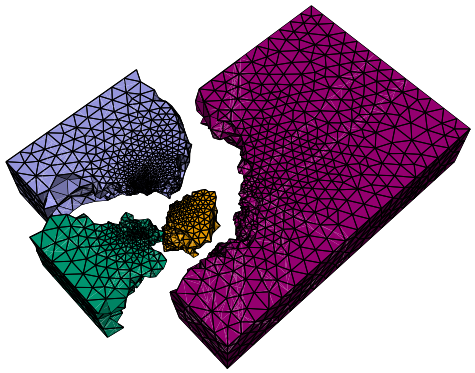
Strong scaling (DLR F11)



Large Scale Sparse Matrix Assembly

Compute and add element stiffness matrices A_{ij}^k to global matrix A

for $k \in \mathcal{T}$
  for $i = 1 \ldots n$
    for $j = 1 \ldots n$
      $A_{\mathcal{I}_{(i)},\mathcal{I}_{(j)}} += A_{ij}^k$



- Two phase assembly process
- Message matching is expensive on large #PEs
- One sided communication?
- PGAS languages?

Hybrid MPI+PGAS

Combining PGAS languages with legacy MPI codes

A path forward for MPI only codes

- Too costly to rewrite a large code base
- Rewrite latency sensitive kernels in PGAS

JANPACK

A sparse matrix library written in Unified Parallel C

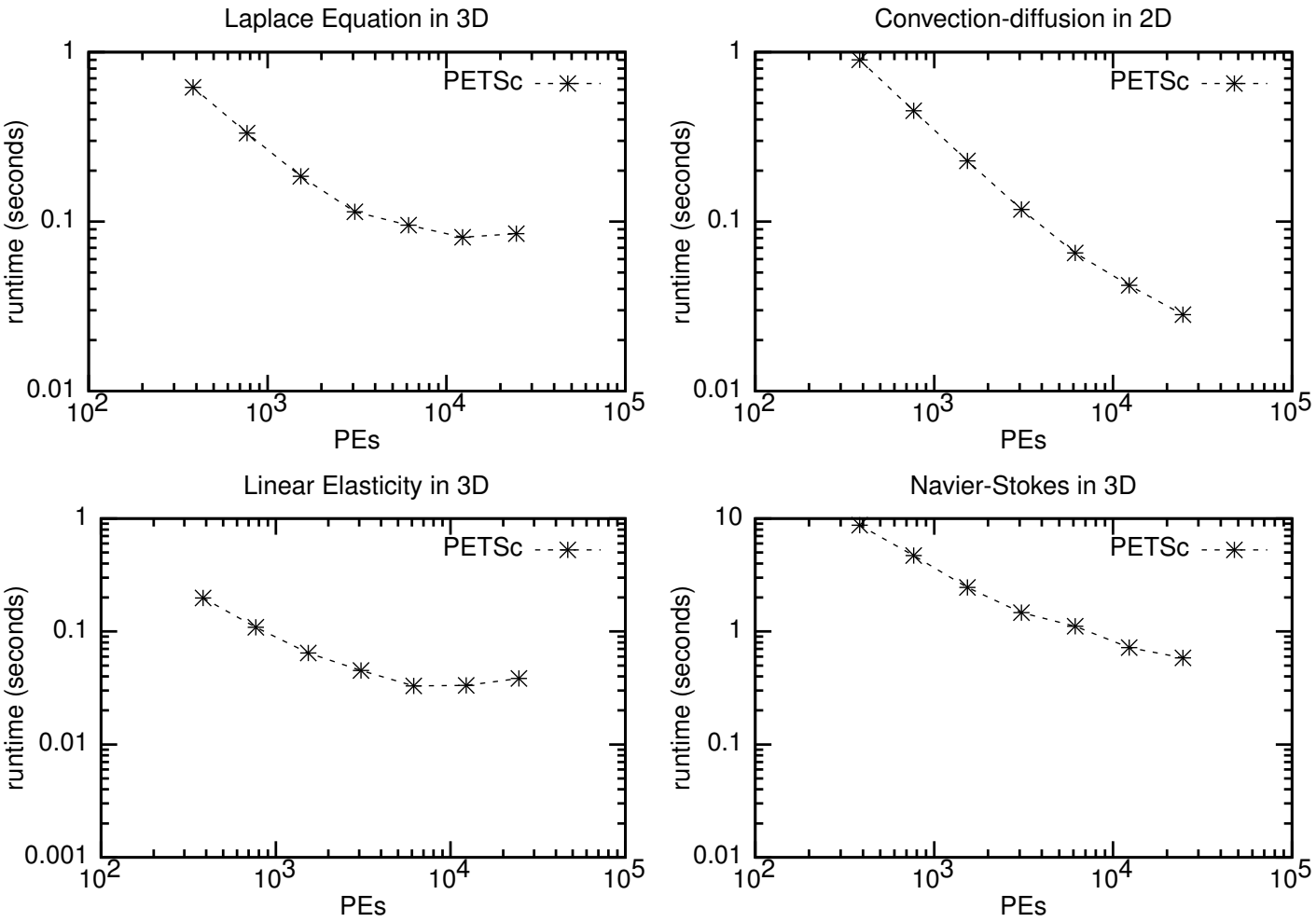
- General, but influenced by our FEM needs
  - Row based data distribution
  - CRS like representation
- Algebraic multigrid
- Krylov solvers (CG (std., pipe., async.), BiCGSTAB (std., ro.))
- Preconditioners (block Jacobi, D-ILU, ILU(0), AMG)

Large Scale Sparse Matrix Assembly

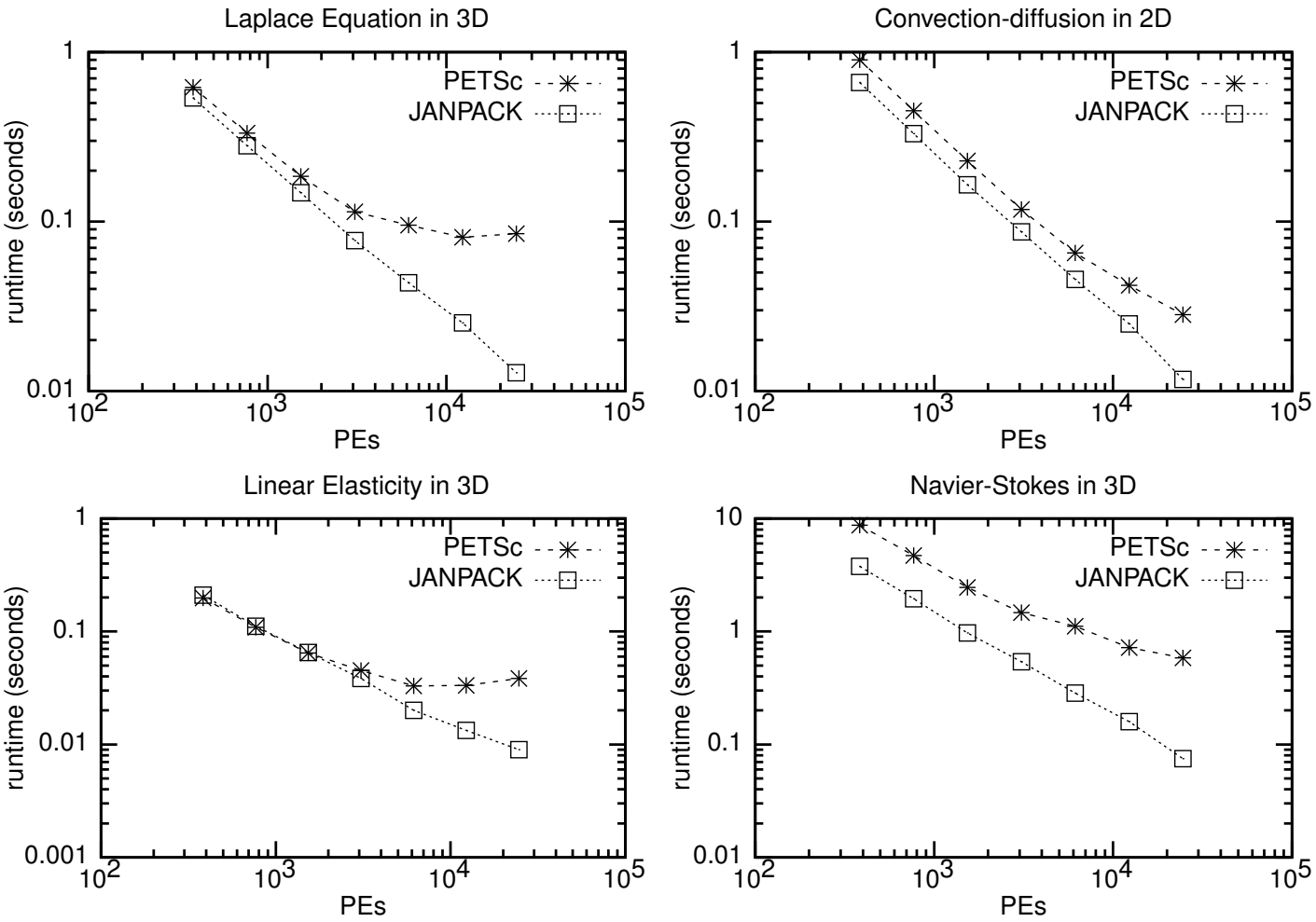
Benchmark suite

- Four different problems with various characteristics
  - 3D Laplace , 317M elements
  - 2D Convection-diffusion, 214 M elements
  - 3D Linear Elasticity, 14M elements
  - 3D Navier-Stokes (Momentum part), 645M elements
- Compute FEM integrals in DOLFIN
- Compare two different LA backends
  - PETSc (MPI)
  - JANPACK (UPC)
- Measure reassembly times
- Each reassembly was repeated 100 times
- Cray XC40 Hornet (HLRS)

Sparse matrix assembly



Sparse matrix assembly



Summary

FEniCS HPC – Automated scientific computing

- Easy expression of complex multiphysics problems
  - Code generation
- General and portable FEM kernel
- Automatic “optimal” mesh generation through adjoints
  - Adaptive algorithm with error control in output of interest
  - Adjoints equations from code generation