Firedrake: Re-imagining FEniCS by Composing Domain-specific Abstractions

Florian Rathgeber[1], Lawrence Mitchell[1], David Ham[1,2], Michael Lange[3], Andrew McRae[2], Fabio Luporini[1], Gheorghe-teodor Bercea[1], Paul Kelly[1]

[1] Department of Computing, Imperial College London [2] Department of Mathematics, Imperial College London [3] Department of Earth Science & Engineering, Imperial College London

> *The FEniCS Project is a collection of free software for automated, efficient solution of differential equations.*
>
> — *fenicsproject.org*

Firedrake

> *Firedrake is an automated system for the portable solution of partial differential equations using the finite element method (FEM).*
>
> — *firedrakeproject.org*

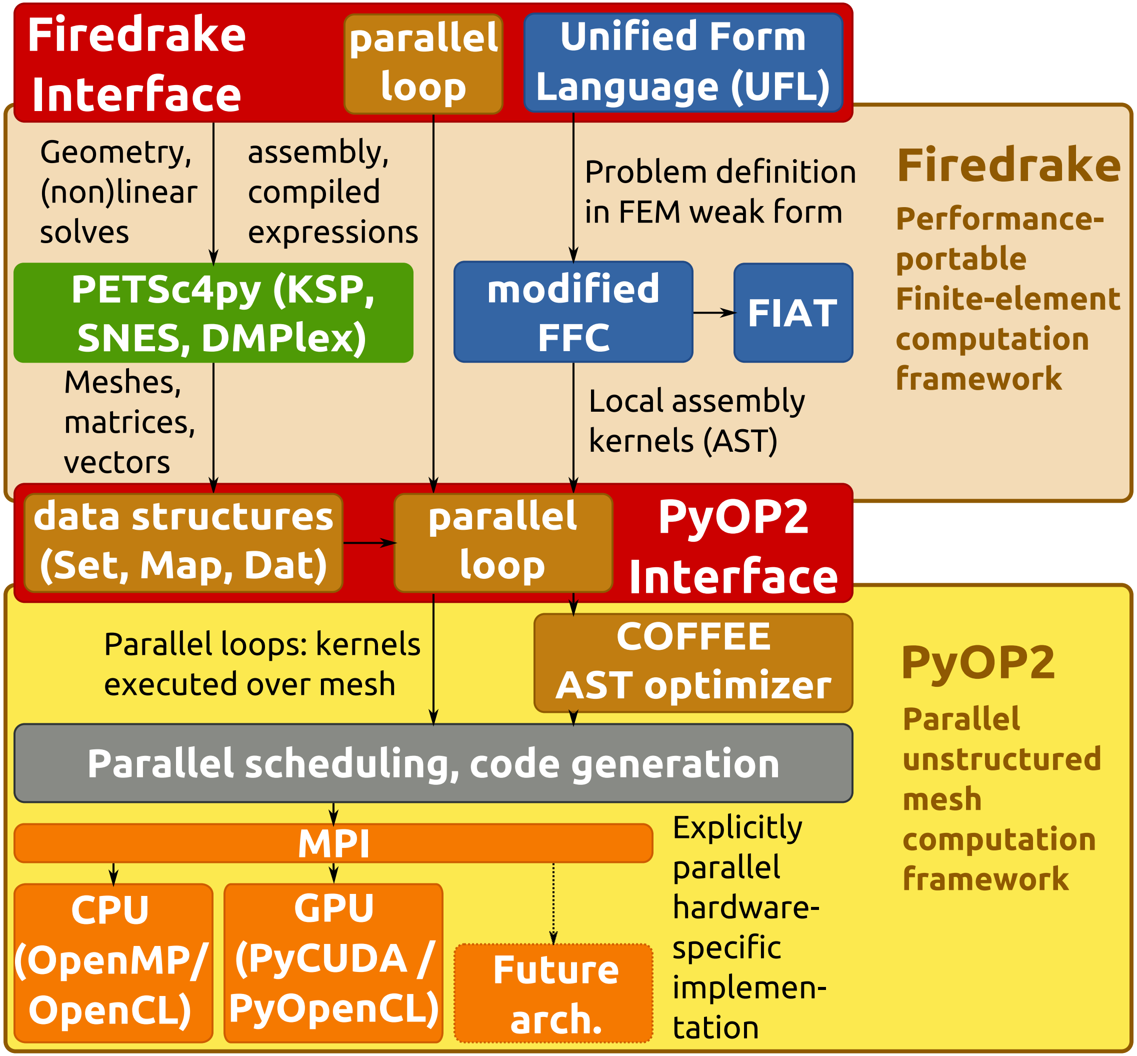
Firedrake

> Firedrake is an automated system for the portable solution of partial differential equations using the finite element method (FEM).
>
> — *firedrakeproject.org*

Two-layer abstraction for FEM computation from high-level descriptions:

- Firedrake: a portable finite-element computation framework
  Drive FE computations from a high-level problem specification
- PyOP2: a high-level interface to unstructured mesh based methods
  Efficiently execute kernels over an unstructured grid in parallel

The Firedrake/PyOP2 tool chain



Parallel computations on unstructured meshes with PyOP2

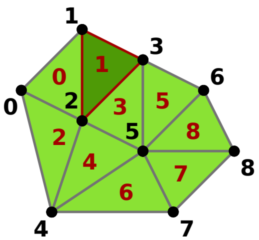
Scientific computations on unstructured meshes

- Independent *local operations* for each element of the mesh described by a *kernel*.
- *Reductions* aggregate contributions from local operations to produce the final result.

PyOP2

A domain-specific language embedded in Python for parallel computations on unstructured meshes or graphs.

Unstructured mesh



PyOP2 Sets:
nodes (9 entities: 0-8)
elements (9 entities: 0-8)
PyOP2 Map elements-nodes:
elem_nodes = [[0, 1, 2], [1, 3, 2], ...]
PyOP2 Dat on nodes:
coords = [..., [.5,.5], [.5,-.25], [1,.25], ...]

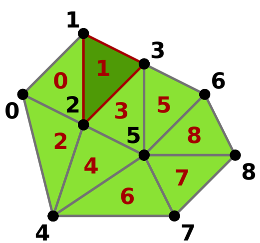
Scientific computations on unstructured meshes

- Independent *local operations* for each element of the mesh described by a *kernel*.
- *Reductions* aggregate contributions from local operations to produce the final result.

PyOP2

A domain-specific language embedded in Python for parallel computations on unstructured meshes or graphs.

Unstructured mesh



PyOP2 Sets:
nodes (9 entities: 0-8)
elements (9 entities: 0-8)
PyOP2 Map elements-nodes:
elem_nodes = [[0, 1, 2], [1, 3, 2], ...]
PyOP2 Dat on nodes:
coords = [..., [.5,.5], [.5,-.25], [1,.25], ...]

PyOP2 Data Model

Mesh topology

- `Sets` – cells, vertices, etc
- `Maps` – connectivity between entities in different sets

Data

- `Dats` – Defined on sets (hold pressure, temperature, etc)

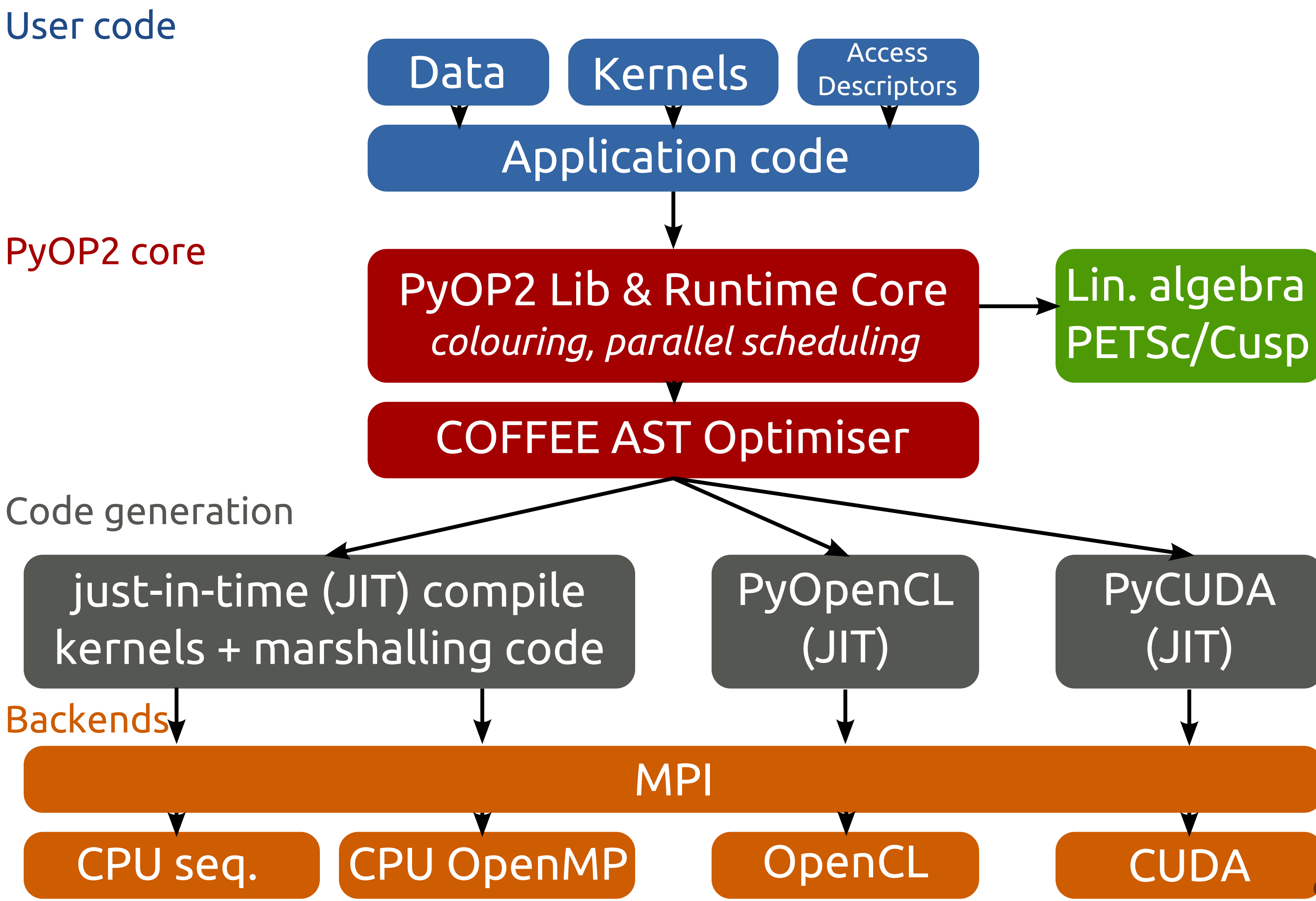
Kernels / parallel loops

- Executed in parallel on a set through a parallel loop
- Read / write / increment data accessed via maps

Linear algebra

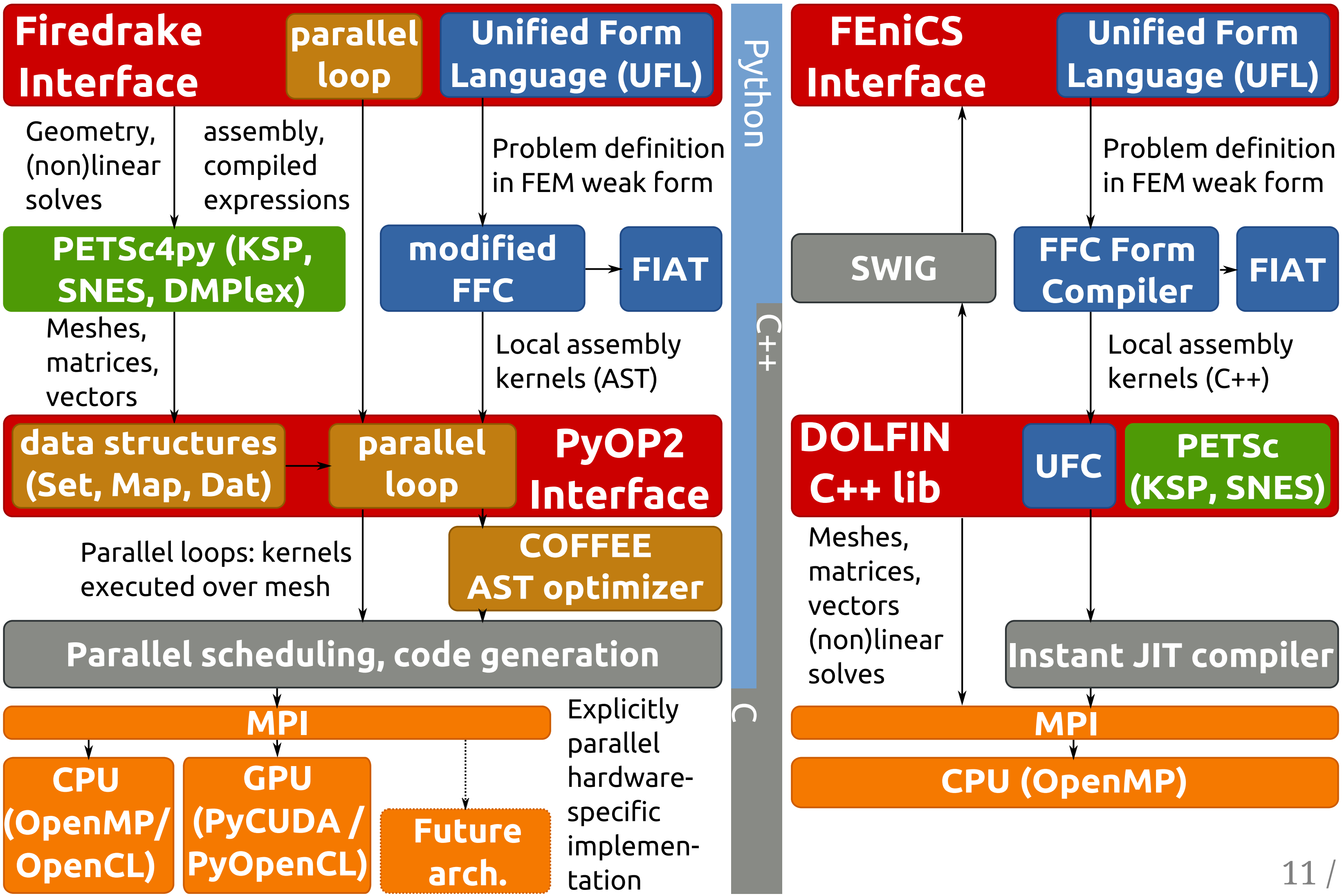
- Sparsities defined by mappings
- Matrix data on sparsities
- Kernels compute a local matrix – PyOP2 handles global assembly

PyOP2 Architecture



Finite-element computations with Firedrake

Firedrake vs. DOLFIN/FEniCS tool chains



Firedrake concepts

Function

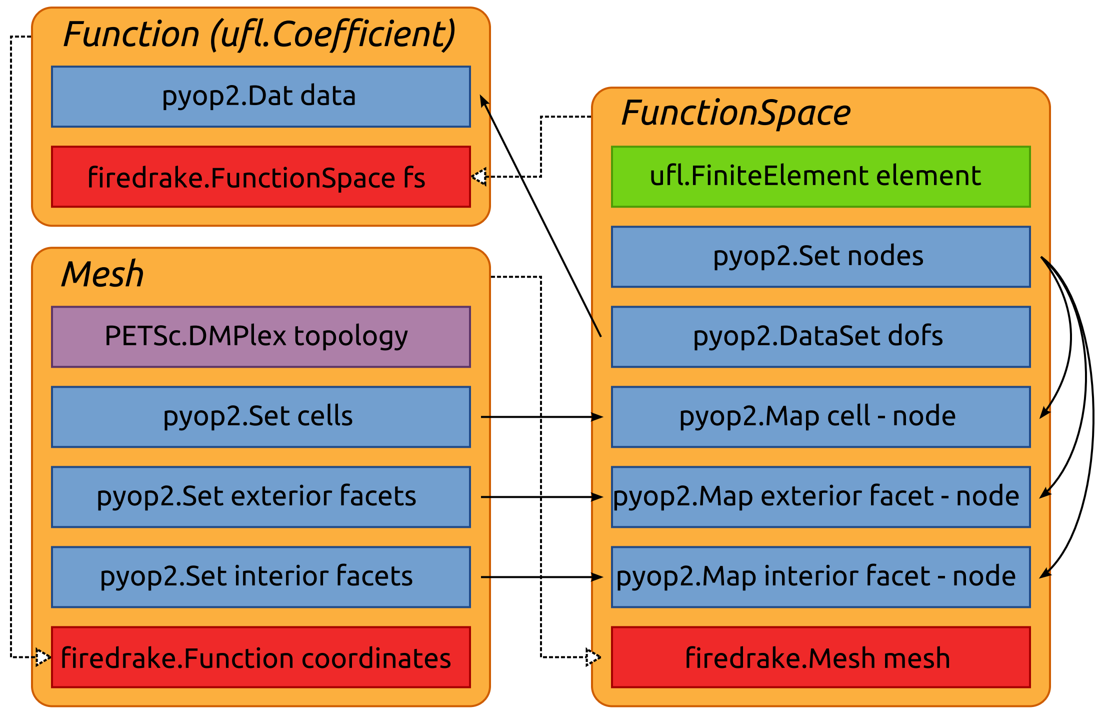
Field defined on a set of degrees of freedom (DoFs), data stored as PyOP2 `Dat`

FunctionSpace

Characterized by a family and degree of FE basis functions, defines DOFs for function and relationship to mesh entities

Mesh

Defines abstract topology by sets of entities and maps between them (PyOP2 data structures)



Driving Finite-element Computations in Firedrake

Solving the Helmholtz equation in Python using Firedrake:

$$\int_\Omega \nabla v \cdot \nabla u - \lambda v u \, dV = \int_\Omega v f \, dV$$

```python
from firedrake import *

# Read a mesh and define a function space
mesh = Mesh('filename')
V = FunctionSpace(mesh, "Lagrange", 1)

# Define forcing function for right-hand side
f = Expression("- (lmbda + 2*(n**2)*pi**2) * sin(X[0]*pi*n) * sin(X[1]*pi*n)",
               lmbda=1, n=8)

# Set up the Finite-element weak forms
u = TrialFunction(V)
v = TestFunction(V)

lmbda = 1
a = (dot(grad(v), grad(u)) - lmbda * v * u) * dx
L = v * f * dx

# Solve the resulting finite-element equation
p = Function(V)
solve(a == L, p)
```

Behind the scenes of the solve call

- Firedrake always solves nonlinear problems in resdiual form `F(u;v) = 0`
- Transform linear problem into residual form:

  ```
  J = a
  F = ufl.action(J, u) - L
  ```

  - Jacobian known to be `a`
  - **Always** solved in a single Newton (nonlinear) iteration
- Use Newton-like methods from PETSc SNES
- PETSc SNES requires two callbacks to evaluate residual and Jacobian:
  - evaluate residual by assembling residual form

    ```
    assemble(F, tensor=F_tensor)
    ```

  - evaluate Jacobian by assembling Jacobian form

    ```
    assemble(J, tensor=J_tensor, bcs=bcs)
    ```

Applying boundary conditions

- Always preserve symmetry of the operator
- Avoid costly search of CSR structure to zero rows/columns
- Zeroing during assembly, but requires boundary DOFs:
    - negative row/column indices for boundary DOFs during addto
    - instructs PETSc to drop entry, leaving 0 in assembled matrix

Applying boundary conditions

- Always preserve symmetry of the operator
- Avoid costly search of CSR structure to zero rows/columns
- Zeroing during assembly, but requires boundary DOFs:
  - negative row/column indices for boundary DOFs during addto
  - instructs PETSc to drop entry, leaving 0 in assembled matrix

Preassembly

```
A = assemble(a)
b = assemble(L)
solve(A, p, b, bcs=bcs)
```

Applying boundary conditions

- Always preserve symmetry of the operator
- Avoid costly search of CSR structure to zero rows/columns
- Zeroing during assembly, but requires boundary DOFs:
    - negative row/column indices for boundary DOFs during addto
    - instructs PETSc to drop entry, leaving 0 in assembled matrix

Preassembly

```
A = assemble(a)  # A unassembled, A.thunk(bcs) not yet called
b = assemble(L)
solve(A, p, b, bcs=bcs)
```

Applying boundary conditions

- Always preserve symmetry of the operator
- Avoid costly search of CSR structure to zero rows/columns
- Zeroing during assembly, but requires boundary DOFs:
  - negative row/column indices for boundary DOFs during addto
  - instructs PETSc to drop entry, leaving 0 in assembled matrix

Preassembly

```
A = assemble(a)  # A unassembled, A.thunk(bcs) not yet called
b = assemble(L)
solve(A, p, b, bcs=bcs)  # A.thunk(bcs) called, A assembled
```

Applying boundary conditions

- Always preserve symmetry of the operator
- Avoid costly search of CSR structure to zero rows/columns
- Zeroing during assembly, but requires boundary DOFs:
    - negative row/column indices for boundary DOFs during addto
    - instructs PETSc to drop entry, leaving 0 in assembled matrix

Preassembly

```
A = assemble(a)  # A unassembled, A.thunk(bcs) not yet called
b = assemble(L)
solve(A, p, b, bcs=bcs)  # A.thunk(bcs) called, A assembled
# ...
solve(A, p, b, bcs=bcs)  # bcs consistent, no need to reassemble
```

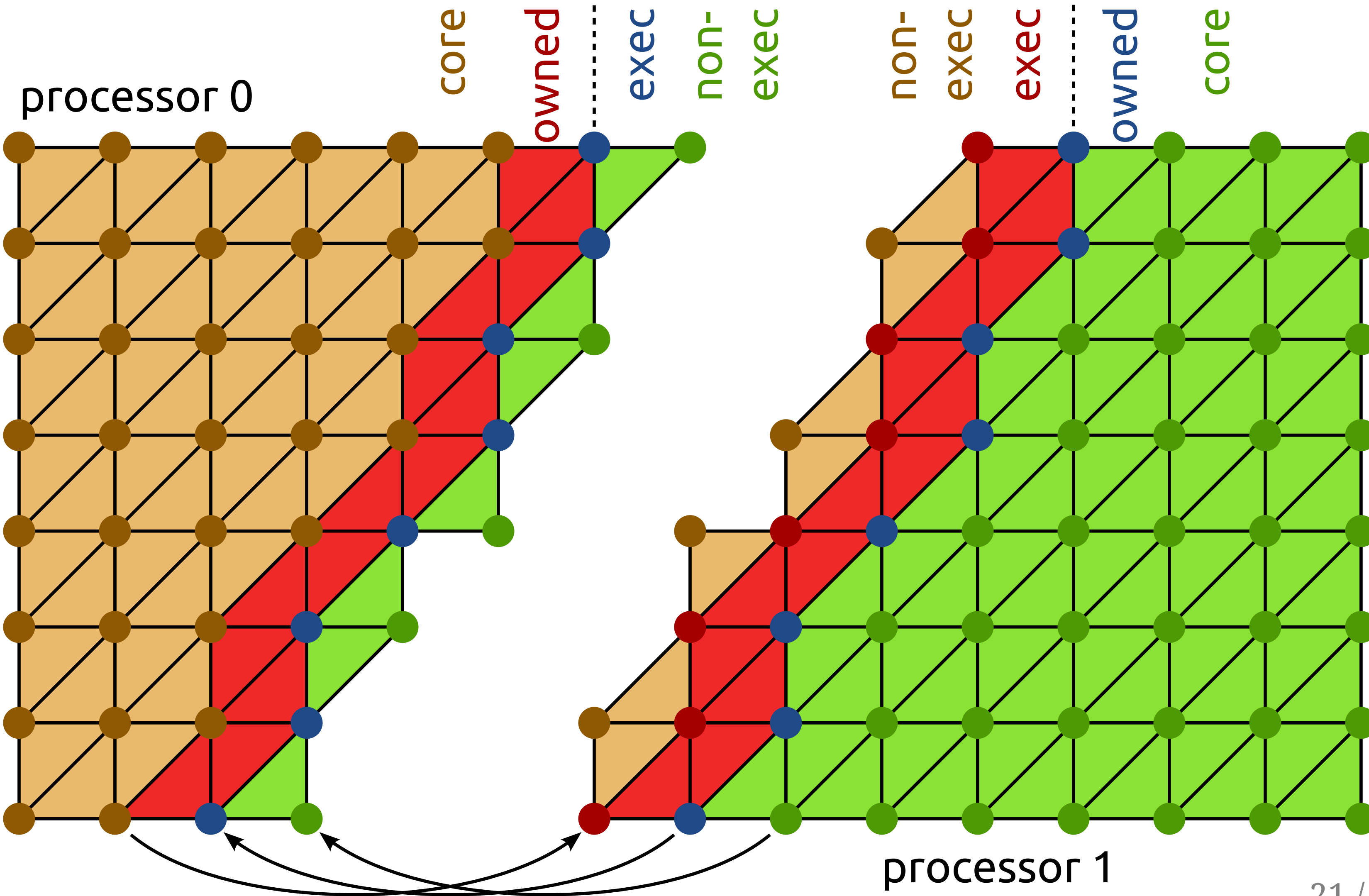
Applying boundary conditions

- Always preserve symmetry of the operator
- Avoid costly search of CSR structure to zero rows/columns
- Zeroing during assembly, but requires boundary DOFs:
    - negative row/column indices for boundary DOFs during addto
    - instructs PETSc to drop entry, leaving 0 in assembled matrix

Preassembly

```
A = assemble(a)  # A unassembled, A.thunk(bcs) not yet called
b = assemble(L)
solve(A, p, b, bcs=bcs)  # A.thunk(bcs) called, A assembled
# ...
solve(A, p, b, bcs=bcs)  # bcs consistent, no need to reassemble
# ...
solve(A, p, b, bcs=bcs2)  # bcs differ, reassemble, call A.thunk(bcs2)
```

Distributed Parallel Computations with MPI



Benchmarks

Hardware

- Intel Xeon E5-2620 @ 2.00GHz (Sandy Bridge)
- 16GB RAM

Compilers

- Intel Compilers 14.0.1
- Intel MPI 3.1.038
- Compiler flags: -O3 -xAVX

Software

- DOLFIN 389e0269 (April 4 2014)
- Firedrake 570d999 (May 13 2014)
- PyOP2 e775c5e (May 9 2014)

Problem setup

- DOLFIN + Firedrake: RCM mesh reordering enabled
- DOLFIN: quadrature with optimisations enabled
- Firedrake: quadrature with COFFEE loop-invariant code motion enabled

Poisson benchmark

preassembled system

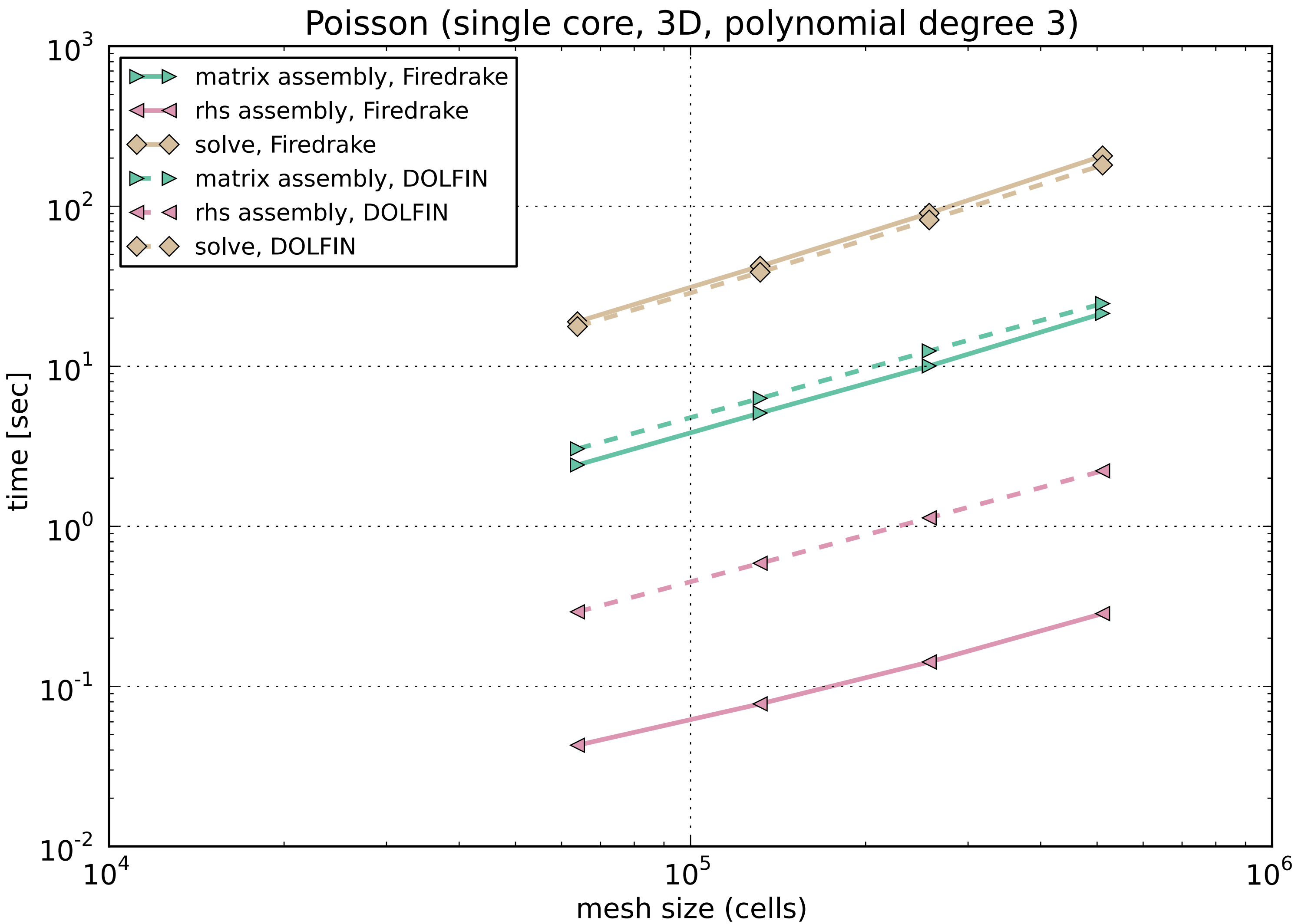
Solver

CG

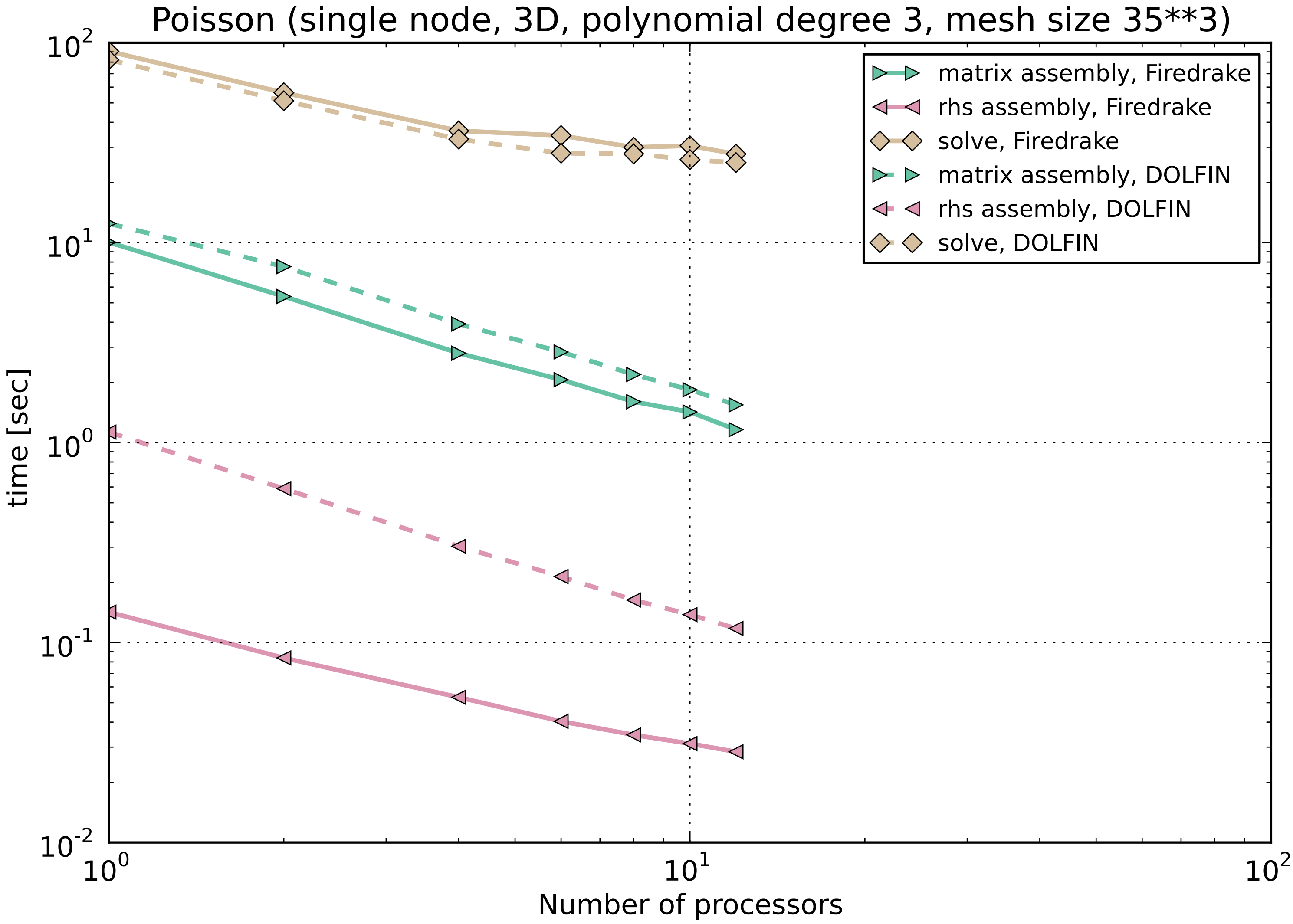
Preconditioner

Hypre Boomeramg

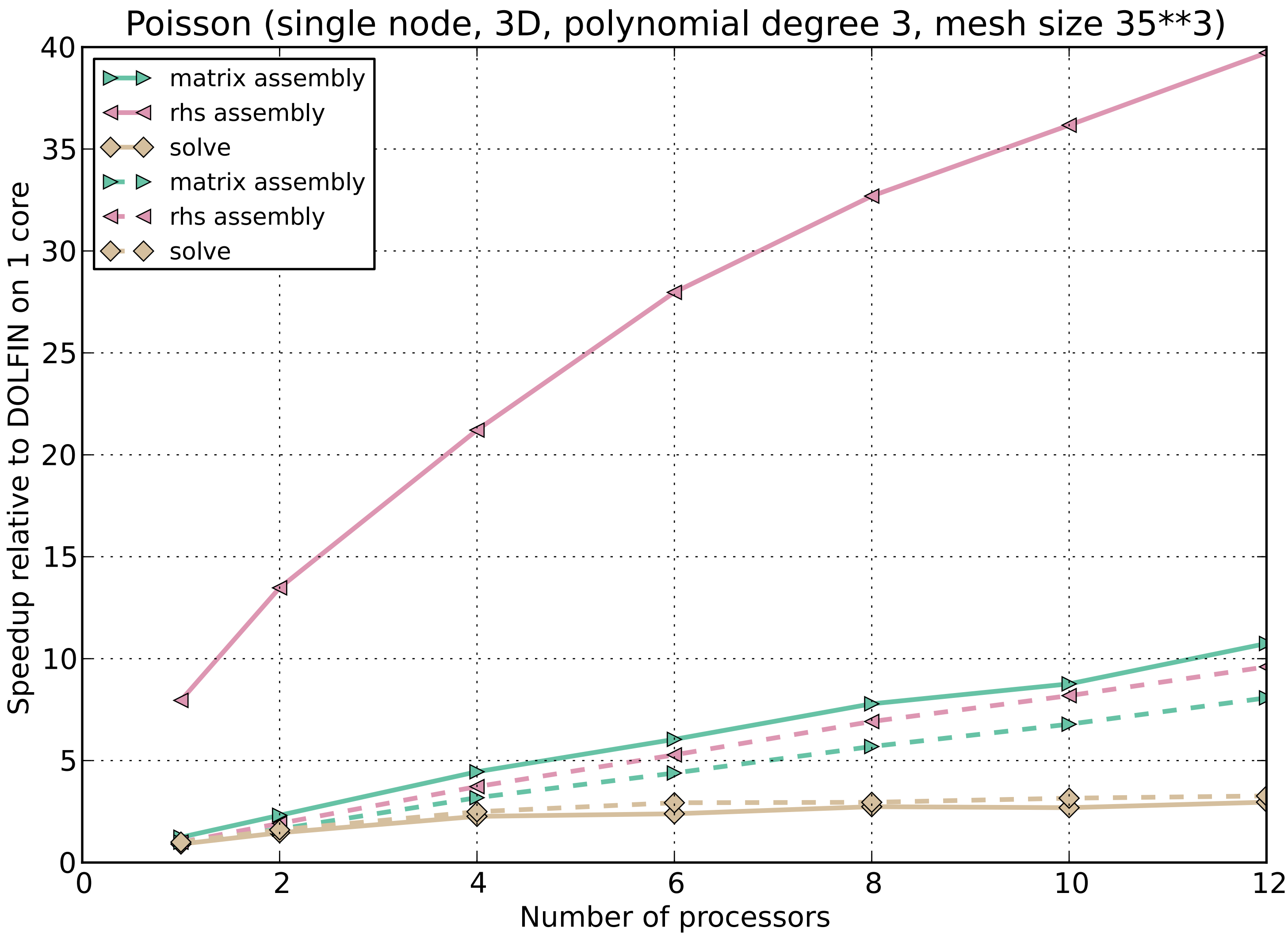
```
V = FunctionSpace(mesh, "Lagrange", degree)

# Dirichlet BC for x = 0 and x = 1
bc = DirichletBC(V, 0.0, [3, 4])

# Test, trial and coefficient functions
u = TrialFunction(V)
v = TestFunction(V)
f = Function(V).interpolate(Expression(
        "10*exp(-(pow(x[0] - 0.5, 2) + \
        pow(x[1] - 0.5, 2)) / 0.02)"))
g = Function(V).interpolate(Expression("sin(5*x[0])"))

# Bilinear and linear forms
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds

# Pre-assemble and solve
u = Function(V)
A = assemble(a, bcs=bc)
b = assemble(L)
bc.apply(b)

solve(A, u, b, solver_parameters=params)
```



solid: Firedrake, dashed: DOLFIN



solid: Firedrake, dashed: DOLFIN



solid: Firedrake, dashed: DOLFIN

Incompressible Navier-Stokes benchmark (Chorin's method)

preassembled system

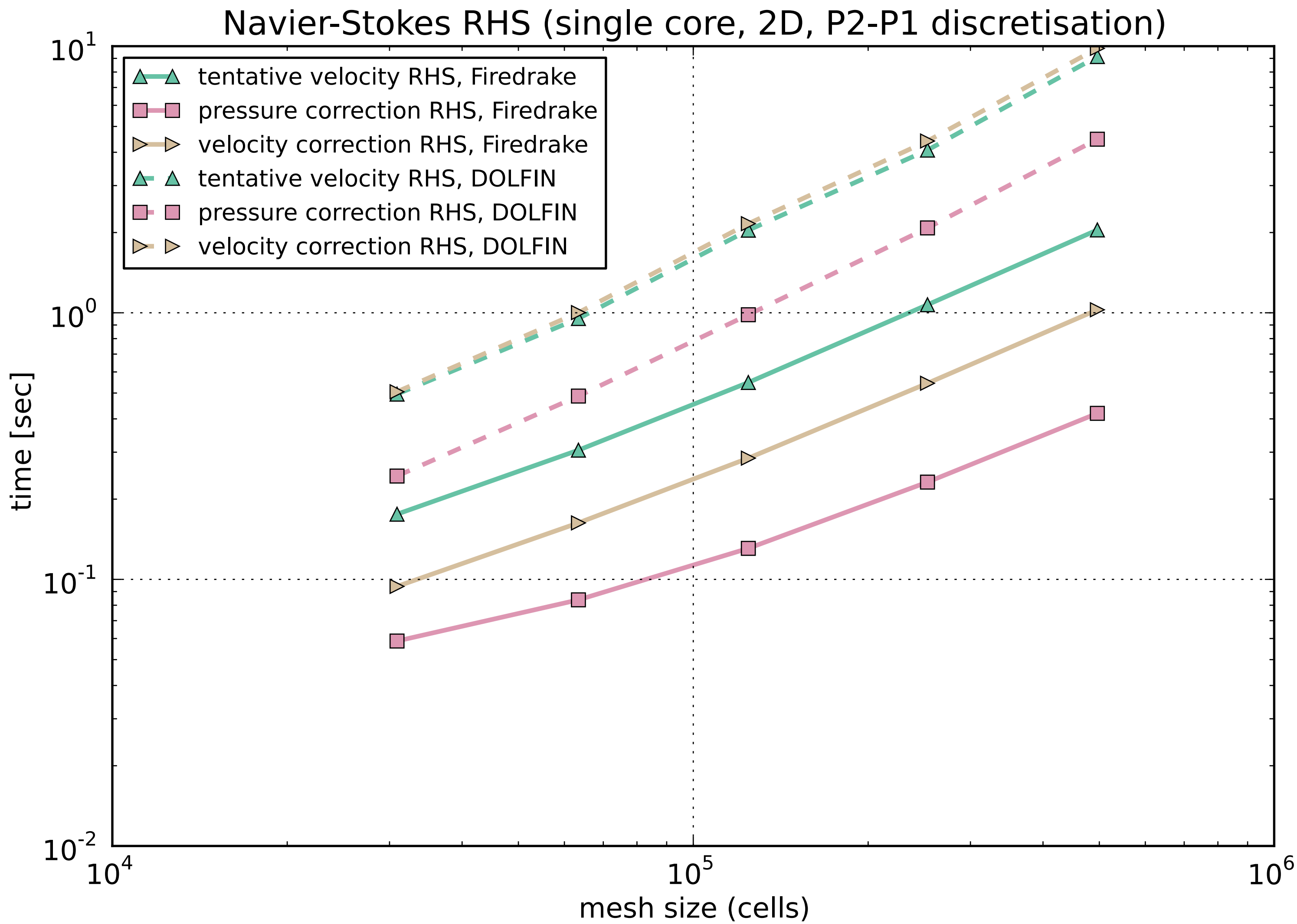
Solver

- GMRES for tentative velocity + velocity correction
- CG for pressure correction

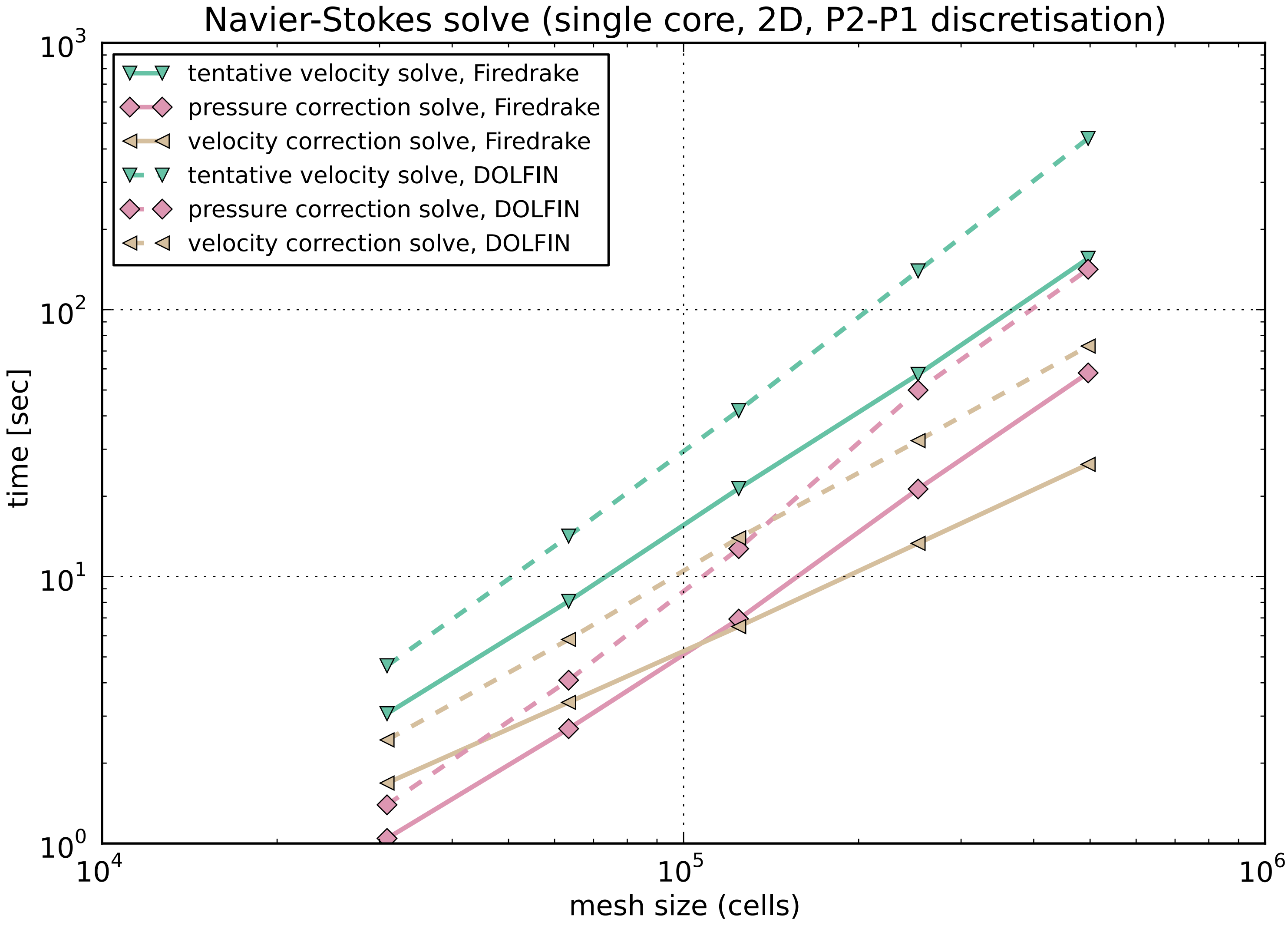
Preconditioner

- block-Jacobi
- ILU block preconditioner

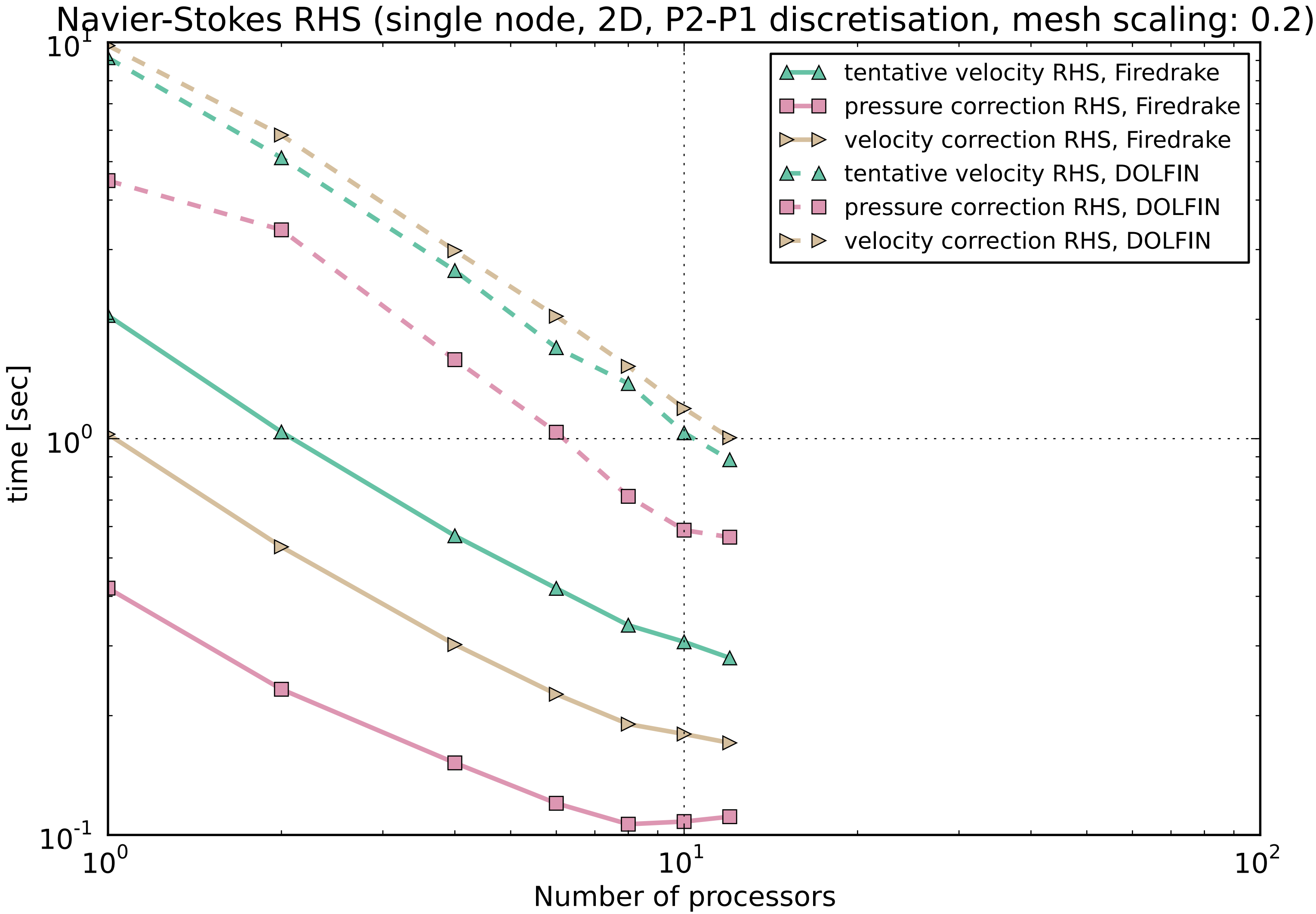
```
V = VectorFunctionSpace(mesh, "Lagrange", 2)
Q = FunctionSpace(mesh, "Lagrange", 1)
u, p = TrialFunction(V), TrialFunction(Q)
v, q = TestFunction(V), TestFunction(Q)

dt = 0.01
nu = 0.01
p_in = Constant(0.0)

noslip = DirichletBC(V, Constant((0.0, 0.0)), (1, 3, 4, 6))
inflow = DirichletBC(Q, p_in, 5)
outflow = DirichletBC(Q, 0, 2)
bcu = [noslip]
bcp = [inflow, outflow]

u0, u1, p1 = Function(V), Function(V), Function(Q)
k = Constant(dt)
f = Constant((0, 0))

# Tentative velocity step
F1 = (1/k)*inner(u - u0, v)*dx + inner(grad(u0)*u0, v)*dx + \
    nu*inner(grad(u), grad(v))*dx - inner(f, v)*dx
a1, L1 = lhs(F1), rhs(F1)

# Pressure update
a2 = inner(grad(p), grad(q))*dx
L2 = -(1/k)*div(u1)*q*dx

# Velocity update
a3 = inner(u, v)*dx
L3 = inner(u1, v)*dx - k*inner(grad(p1), v)*dx
```



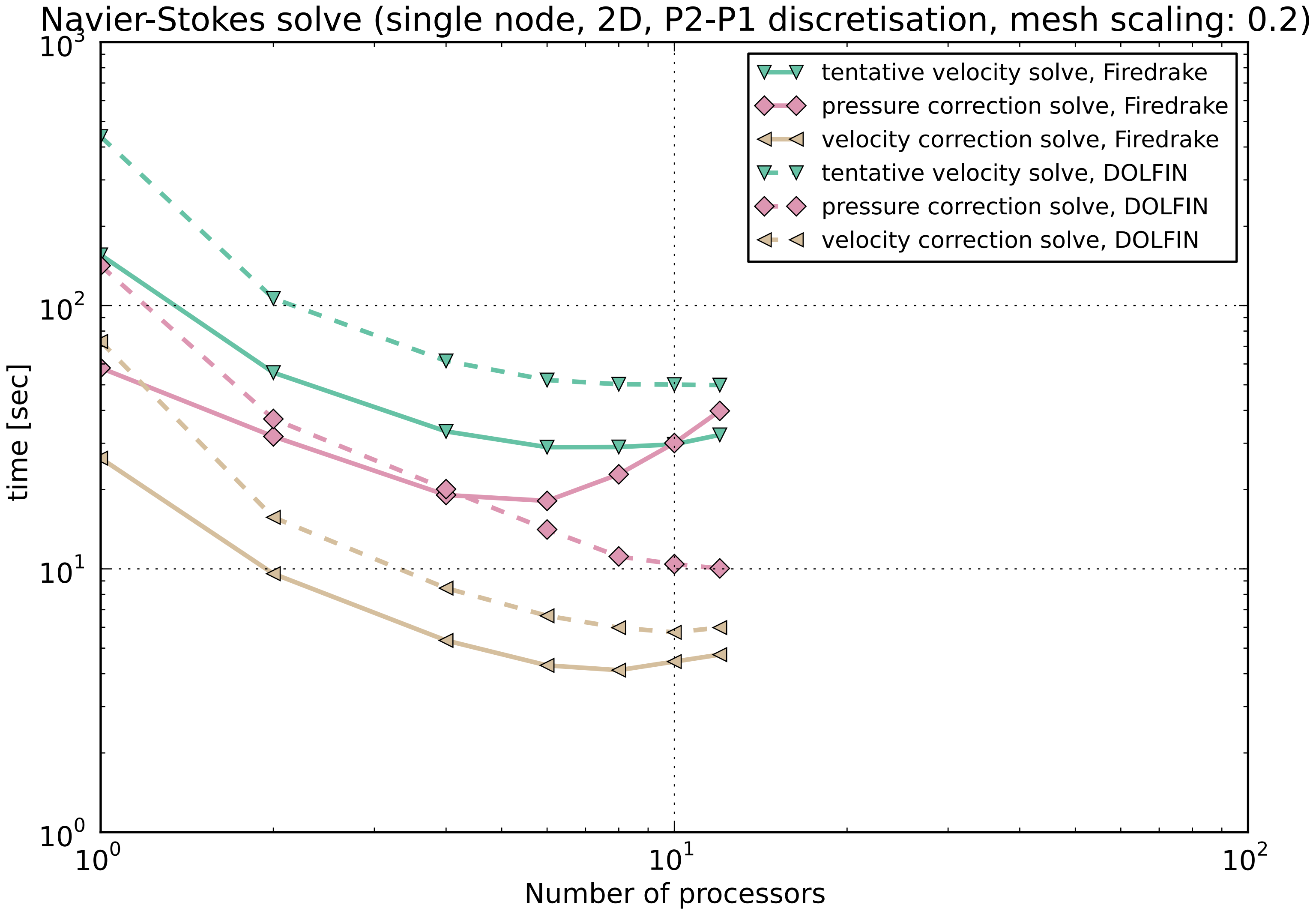
solid: Firedrake, dashed: DOLFIN



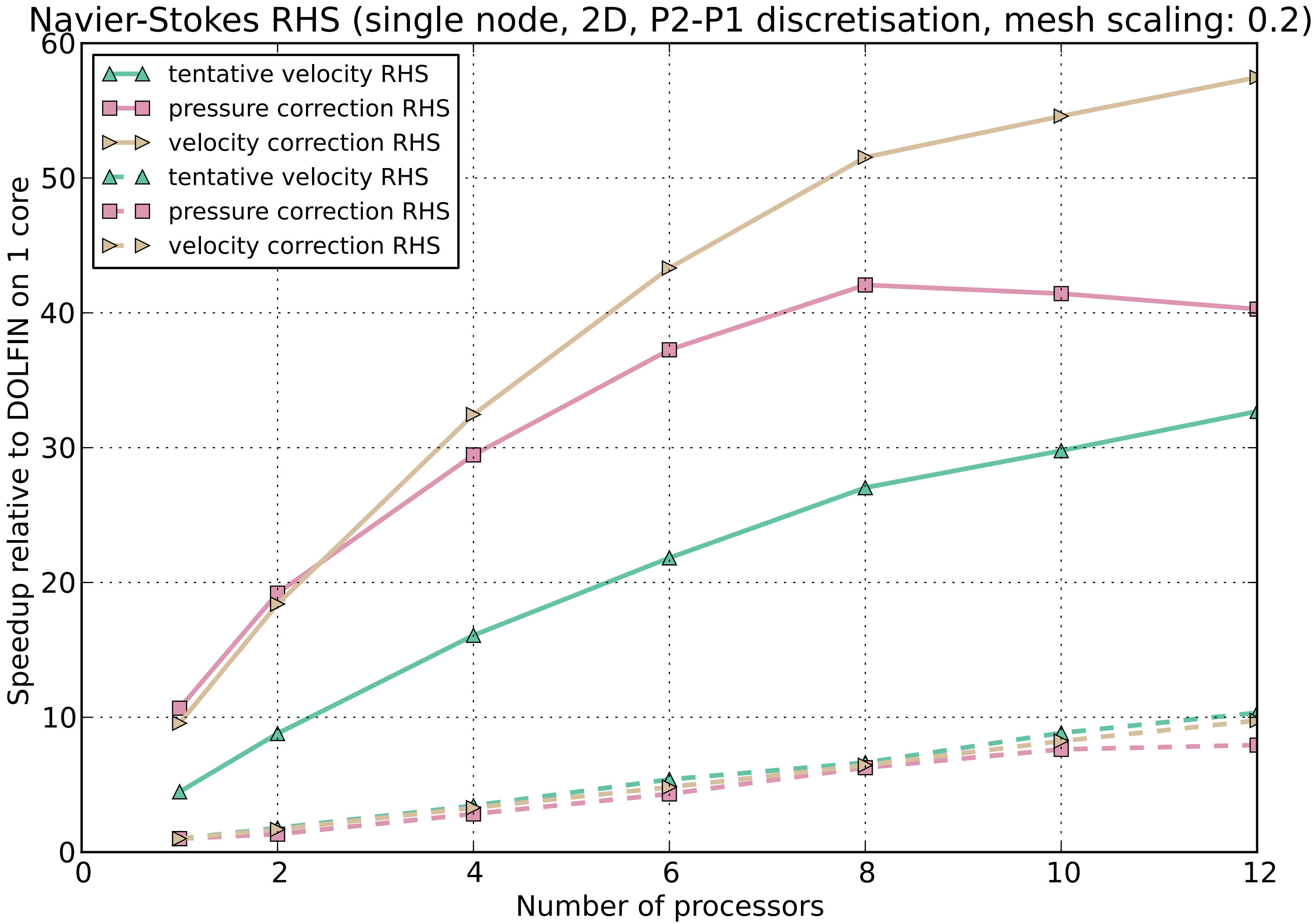
solid: Firedrake, dashed: DOLFIN



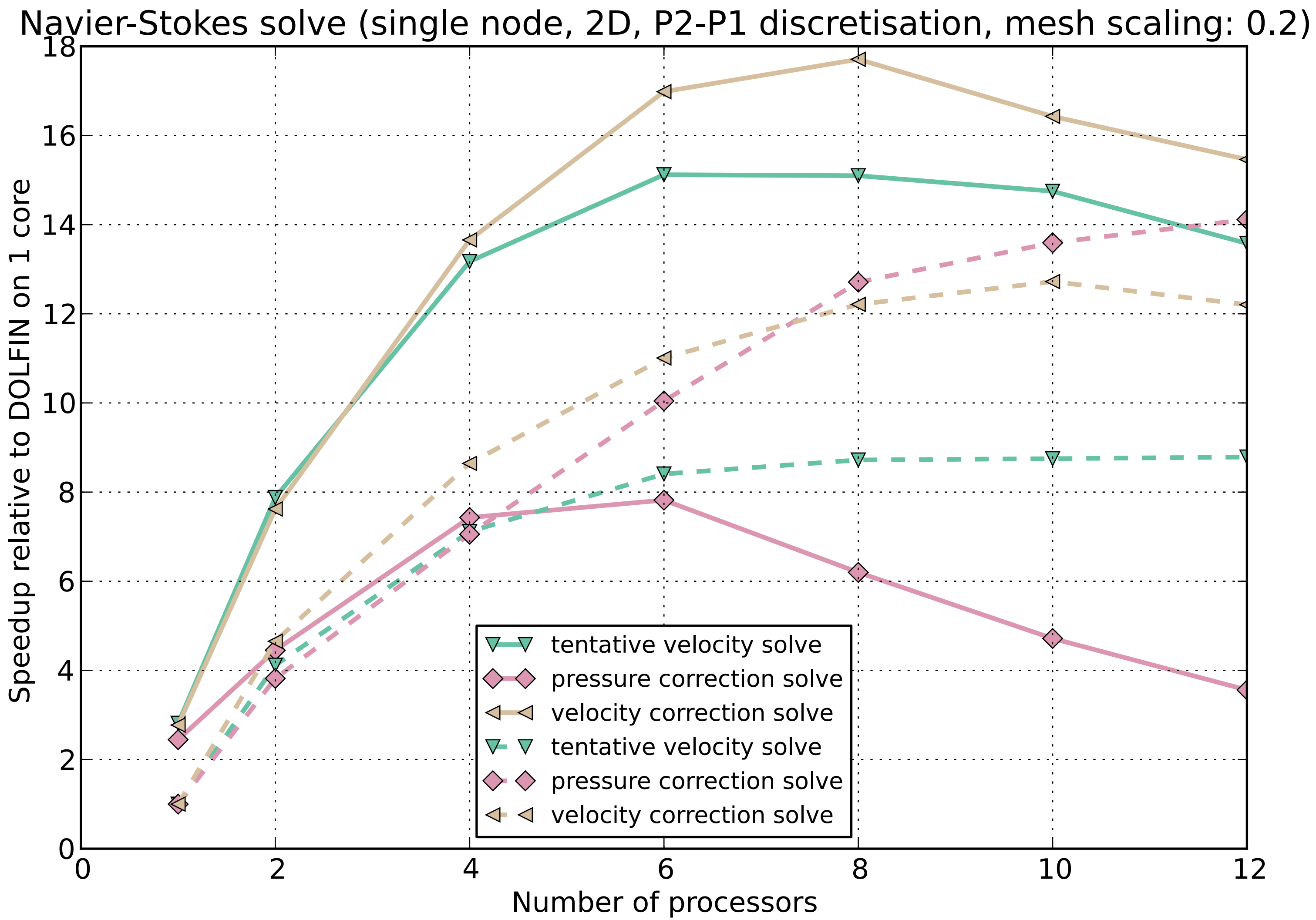
solid: Firedrake, dashed: DOLFIN



solid: Firedrake, dashed: DOLFIN



solid: Firedrake, dashed: DOLFIN



solid: Firedrake, dashed: DOLFIN

Summary and additional features

Summary

- Two-layer abstraction for FEM computation from high-level descriptions
  - Firedrake: a performance-portable finite-element computation framework
    Drive FE computations from a high-level problem specification
  - PyOP2: a high-level interface to unstructured mesh based methods
    Efficiently execute kernels over an unstructured grid in parallel
- Decoupling of Firedrake (FEM) and PyOP2 (parallelisation) layers
- Firedrake concepts implemented with PyOP2/PETSc constructs
- Portability for unstructured mesh applications: FEM, non-FEM or combinations
- Extensible framework beyond FEM computations (e.g. image processing)

Summary and additional features

Summary

- Two-layer abstraction for FEM computation from high-level descriptions
  - Firedrake: a performance-portable finite-element computation framework
    Drive FE computations from a high-level problem specification
  - PyOP2: a high-level interface to unstructured mesh based methods
    Efficiently execute kernels over an unstructured grid in parallel
- Decoupling of Firedrake (FEM) and PyOP2 (parallelisation) layers
- Firedrake concepts implemented with PyOP2/PETSc constructs
- Portability for unstructured mesh applications: FEM, non-FEM or combinations
- Extensible framework beyond FEM computations (e.g. image processing)

Preview: Firedrake features not covered

- Automatic optimization of generated assembly kernels with COFFEE (Fabio's talk)
- Solving PDEs on extruded (semi-structured) meshes (Doru + Andrew's talk)
- Building meshes using PETSc DMPlex
- Using fieldsplit preconditioners for mixed problems
- Solving PDEs on immersed manifolds
- ...

Thank you!

Contact: Florian Rathgeber, @frathgeber, f.rathgeber@imperial.ac.uk

Resources

- **PyOP2** https://github.com/OP2/PyOP2
  - *PyOP2: A High-Level Framework for Performance-Portable Simulations on Unstructured Meshes* Florian Rathgeber, Graham R. Markall, Lawrence Mitchell, Nicholas Loriant, David A. Ham, Carlo Bertolli, Paul H.J. Kelly, WOLFHPC 2012
  - *Performance-Portable Finite Element Assembly Using PyOP2 and FEniCS* Graham R. Markall, Florian Rathgeber, Lawrence Mitchell, Nicolas Loriant, Carlo Bertolli, David A. Ham, Paul H. J. Kelly , ISC 2013
- **Firedrake** https://github.com/firedrakeproject/firedrake
  - *COFFEE: an Optimizing Compiler for Finite Element Local Assembly* Fabio Luporini, Ana Lucia Varbanescu, Florian Rathgeber, Gheorghe-Teodor Bercea, J. Ramanujam, David A. Ham, Paul H. J. Kelly, submitted
- **UFL** https://bitbucket.org/mapdes/ufl
- **FFC** https://bitbucket.org/mapdes/ffc

This talk is available at http://kynan.github.io/fenics14 (source)

Slides created with remark