Imperial College London

Flexible, Scalable Mesh and Data Management using PETSc DMPlex

M. Lange[1] M. Knepley[2] L. Mitchell[3] G. Gorman[1]

[1]AMCG, Imperial College London
[2]Computational and Applied Mathematics, Rice University
[3]Computing, Imperial College London

June 24, 2015

Motivation

Mesh management

- Many tasks are common across applications:
  Mesh input, partitioning, checkpointing, ...
- File I/O can become severe bottleneck!

Mesh file formats

- Range of mesh generators and formats
  Gmsh, Cubit, Triangle, ExodusII, Fluent, CGNS, ...
- No universally accepted format
  - Applications often "roll their own"
  - No interoperability between codes

Motivation

Interoperability and extensibility

- Abstract mesh topology interface
  - Provided by a widely used library[1]
  - Extensible support for multiple formats
  - Single point for extension and optimisation
  - Many applications inherit capabilities

- Mesh management optimisations
  - Scalable read/write routines
  - Parallel partitioning and load-balancing
  - Mesh renumbering techniques
  - *Unstructured mesh adaptivity*

Finding the right level of abstraction

[1] J. Brown, M. Knepley, and B. Smith. Run-time extensibility and librarization of simulation software. *IEEE Computing in Science and Engineering*, 2015

Imperial College London

Motivation

Unstructured Mesh Management

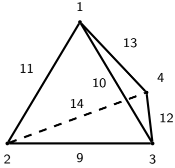
Parallel Mesh Distribution

Fluidity

Firedrake

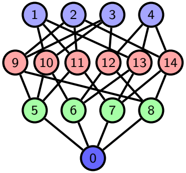
Summary

Unstructured Mesh Management



DMPlex - PETSc's unstructured mesh API[1]

- Abstract mesh connectivity
  - Directed Acyclic Graph (DAG)[2]
  - Dimensionless access
  - Topology separate from discretisation
  - Multigrid preconditioners
- **`PetscSection`**
  - Describes irregular data arrays (CSR)
  - Mapping DAG points to DoFs
- **`PetscSF`:** Star Forest[3]
  - One-sided description of shared data
  - Performs sparse data communication

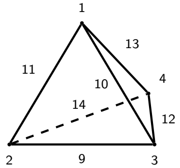


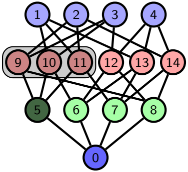
[1] M. Knepley and D. Karpeev. Mesh Algorithms for PDE with Sieve I: Mesh Distribution. *Sci. Program.*, 17(3):215–230, August 2009

[2] A. Logg. Efficient representation of computational meshes. *Int. Journal of Computational Science and Engineering*, 4:283–295, 2009

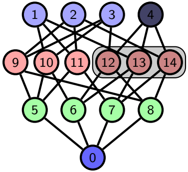
[3] Jed Brown. Star forests as a parallel communication model, 2011

Unstructured Mesh Management

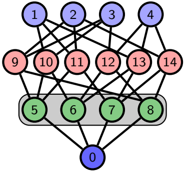




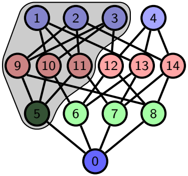
$cone(5) = \{9, 10, 11\}$



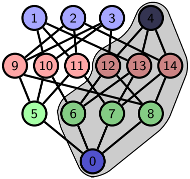
$support(4) = \{12, 13, 14\}$



$stratum_{height}(1) = \{5, 6, 7, 8\}$



$closure(1) = \{1, 2, 3, 5, 9, 10, 11\}$

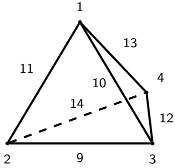


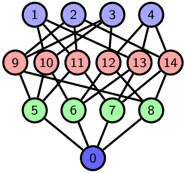
$star(14) = \{0, 4, 6, 7, 8, 12, 13, 14\}$

Unstructured Mesh Management

DMPlex - PETSc's unstructured mesh API[1]

- Input: ExodusII, Gmsh, CGNS, Fluent-CAS, MED, ...
- Output: HDF5 + Xdmf
  - Visualizable checkpoints
- Parallel distribution
  - Partitioners: Chaco, Metis/ParMetis
  - Automated halo exchange via PetscSF
- Mesh renumbering
  - Reverse Cuthill-McGee (RCM)



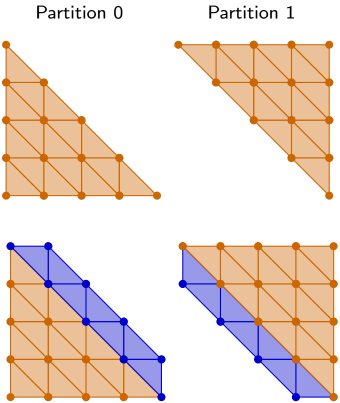


[1] M. Knepley and D. Karpeev. Mesh Algorithms for PDE with Sieve I: Mesh Distribution. *Sci. Program.*, 17(3):215–230, August 2009

Parallel Mesh Distribution

DMPlexDistribute[1]

- Mesh partitioning
  - Topology-based partitioning
  - Metis/ParMetis, Chaco
- Mesh and data migration
  - One-to-all and all-to-all
  - Data migration via SF
- Parallel overlap computation
  - Generic N-level point overlap
  - FVM and FEM adjacency



[1] M. Knepley, M. Lange, and G. Gorman. Unstructured overlapping mesh distribution in parallel. *Submitted to ACM TOMS*, 2015

Parallel Mesh Distribution

```
def DMPlexDistribute(dm, overlap):

  # Derive migration pattern from partition
  DMLabel partition = PetscPartition(partitioner, dm)
  PetscSF migration = PartitionLabelCreateSF(dm, partition)

  # Initial non-overlapping migration
  DM dmParallel   = DMPlexMigrate(dm, migration)
  PetscSF shared = DMPlexCreatePointSF(dmParallel, migration)

  # Parallel overlap generation
  DMLabel overlap    = DMPlexCreateOverlap(dmParallel, N, shared)
  PetscSF migration = PartitionLabelCreateSF(dm, overlap)
  DM dmOverlap       = DMPlexMigrate(dm, migration)
```

- Two-phase distribution enables parallel overlap generation

Parallel Mesh Distribution

```
def DMPlexMigrate(dm, sf):

  if (all-to-all):
    # Back out original local numbering
    old_numbering = DMPlexGetLocalNumbering(dm)
    new_numbering = SFBcast(sf, old_numbering)
    dm.cones      = LToGMappingApply(old_numbering, dm.cones)
  else:
    new_numbering = LToGMappingCreateFromSF(sf)

  # Migrate DM
  DMPlexMigrateCones(dm, sf, new_numbering, dmTarget)
  DMPlexMigrateCoordinates(dm, sf, dmTarget)
  DMPlexMigrateLabels(dm, sf, dmTarget)
```

- Generic migration for one-to-all and all-to-all

Parallel Mesh Distribution

```
def DMPlexCreateOverlap(dm, N, sf):
  # Derive receive connections
  for leaf, root in sf:
    adjacency = DMPlexGetAdjacency(dm, leaf)
    DMLabelSetValue(label, adjcency, root.rank)

  # Derive send connections
  for root, rank in DMPlexDistributeOwnership(dm, sf):
    adjacency = DMPlexGetAdjacency(dm, root)
    DMLabelSetValue(label, adjcency, rank)

  # Add further overlap levels
  for ol = 1 to N:
    for point, rank in DMLabelGetValues(label, rank):
      adjacency = DMPlexGetAdjacency(dm, point)
      DMLabelSetValue(label, adjcency, rank)
```

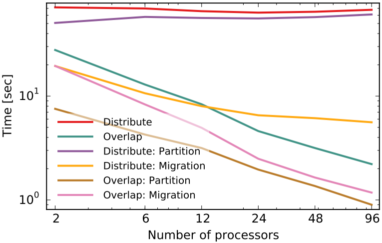
- Generic parallel N-level overlap generation
- Each rank computes local contribution to neighbours

Parallel Mesh Distribution

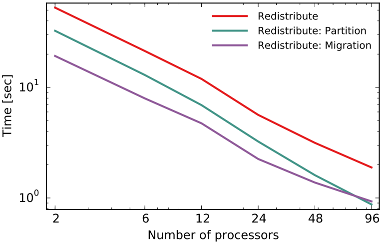
Strong scaling performance:

- Cray XE30 with 4920 nodes; 2×12-core E5-2697 @2.7GHz[1]
- 128^3 unit cube with ≈ 12 mio. cells

One-to-all distribution



All-to-all redistribution



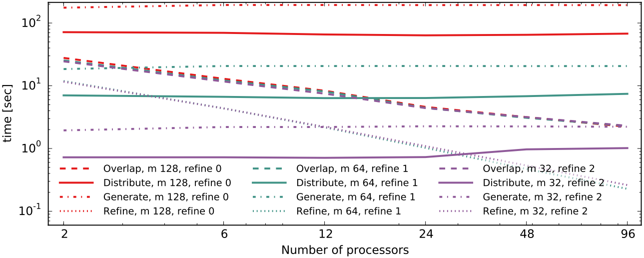
- Remaining bottleneck: Sequential partitioning

[1] ARCHER: www.archer.ac.uk

Parallel Mesh Distribution

Regular parallel refinement:

- Distribute coarse mesh and refine in parallel
- Generate overlap on resulting fine mesh

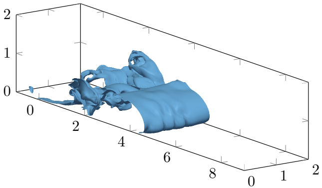


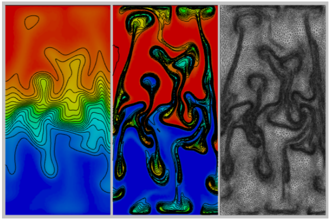
Fluidity

Fluidity

- Unstructured finite element code
- Anisotropic mesh adaptivity
- Uses PETSc as linear solver engine
- Applications:
  - CFD, geophysical flows, ocean modelling, reservoir modelling, mining, nuclear safety, renewable energies, etc.

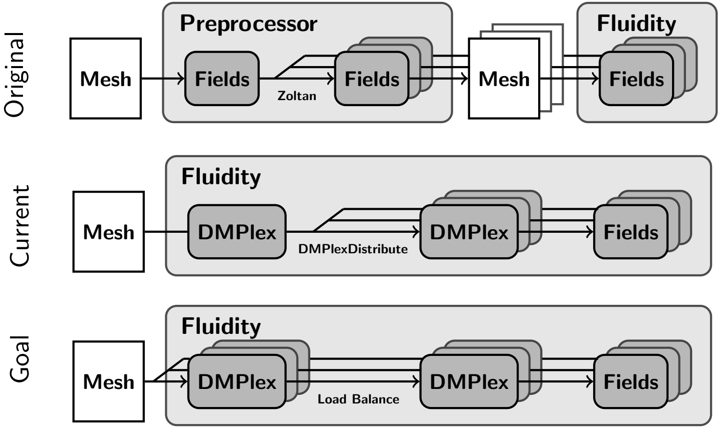
Bottleneck: Parallel pre-processing[1]





[1] X. Guo, M. Lange, G. Gorman, L. Mitchell, and M. Weiland. Developing a scalable hybrid MPI/OpenMP unstructured finite element model. *Computers & Fluids*, 110(0):227 – 234, 2015. ParCFD 2013

Fluidity - DMPlex Integration



Fluidity

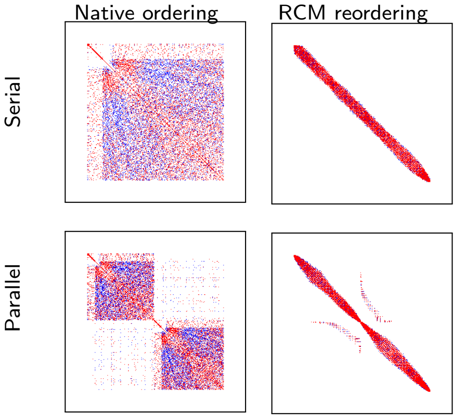
Fluidity - DMPlex Integration

- Delegate mesh input to DMPlex
  - More formats and improved interoperability
  - Potential performance improvements
  - Maintained by third-party library
- Domain decomposition at run-time
  - Remove pre-processing step
  - Avoid unnecessary I/O cycle
  - Topology partitioning reduces communication
- Replaces existing mesh readers
  - Build and distribute DMPlex object
  - Fluidity initialisation in parallel
  - Only one format-agnostic reader method

Fluidity - DMPlex Integration

Mesh reordering

- Fluidity Halos
  - Separate L1/L2 regions
  - “Trailing receives”
  - Requires permutation
- DMPlex provides RCM
  - Locally generated as a permutation
  - Combine with halo reordering
- Fields inherit reordering
  - Better cache coherency



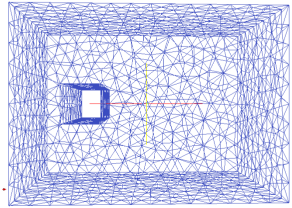
Fluidity - Benchmark

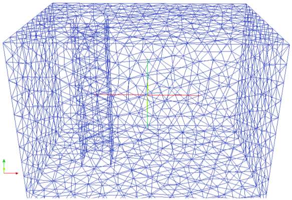
Archer

- Cray XC30
- 4920 nodes (118,080 cores)
- 12-core E5-2697 (Ivy Bridge)

Simulation

- Flow past a square cylinder
- 3D mesh, generated with Gmsh

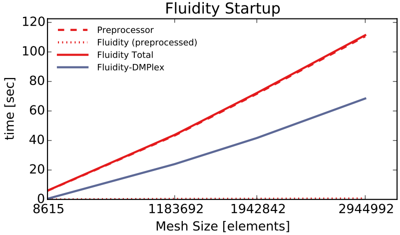


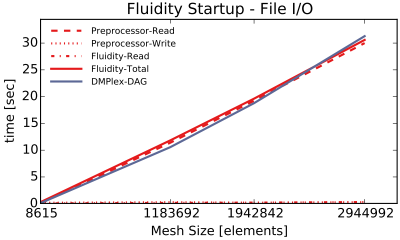


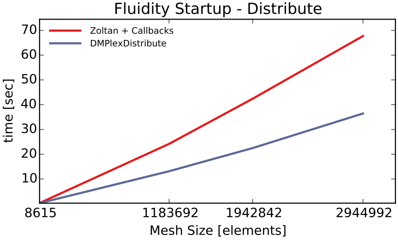
Fluidity - Results

Simulation Startup on 4 nodes

- Runtime distribution wins
- Fast topology distribution
- No clear I/O gains
  - Gmsh does not scale



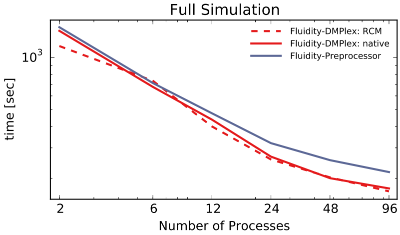


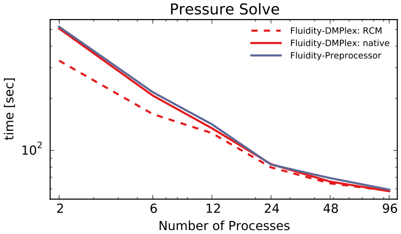


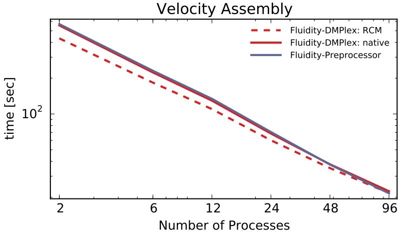
Fluidity - Results

Simulation Performance

- Mesh with ∼2 mio elements
- Preprocessor + 10 timesteps
- RCM brings improvements
  - Pressure solve
  - Velocity assembly







Firedrake

Firedrake - Automated Finite Element computation[1]

- Re-envision FEniCS[2]

$$
\begin{aligned}
\phi^{n+1/2} &= \phi^n - \frac{\Delta t}{2} p^n \\
p^{n+1} &= p^n + \frac{\int_\Omega \nabla \phi^{n+1/2} \cdot \nabla v \, \mathrm{d}x}{\int_\Omega v \, \mathrm{d}x} \quad \forall v \in V \\
\phi^{n+1} &= \phi^{n+1/2} - \frac{\Delta t}{2} p^{n+1}
\end{aligned}
$$

where

$$
\begin{aligned}
\nabla \phi \cdot n &= 0 \text{ on } \Gamma_N \\
p &= \sin(10\pi t) \text{ on } \Gamma_D
\end{aligned}
$$

```
from firedrake import *
mesh = Mesh("wave_tank.msh")
V = FunctionSpace(mesh, 'Lagrange', 1)
p = Function(V, name="p")
phi = Function(V, name="phi")
u = TrialFunction(V)
v = TestFunction(V)
p_in = Constant(0.0)
bc = DirichletBC(V, p_in, 1)
T = 10.
dt = 0.001
t = 0
while t <= T:
    p_in.assign(sin(2*pi*5*t))
    phi -= dt / 2 * p
    p += assemble(dt * inner(grad(v), grad(phi))*dx) \
         / assemble(v*dx)
    bc.apply(p)
    phi -= dt / 2 * p
    t += dt
```

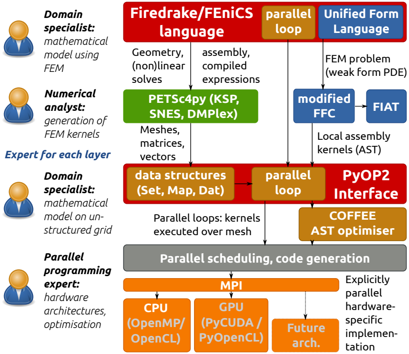
[1] F. Rathgeber, D. Ham, L. Mitchell, M. Lange, F. Luporini, A. McRae, G. Bercea, G. Markall, and P. Kelly. Firedrake: Automating the finite element method by composing abstractions. *Submitted to ACM TOMS*, 2015

[2] A. Logg, K.-A. Mardal, and G. Wells. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012

Firedrake

Firedrake - Automated Finite Element computation

- Implements UFL[1]
  - Outer framework in Python
  - Run-time C code generation
  - PyOP2: Assembly kernel execution framework
- Domain topology from DMPlex
  - Mesh generation and file I/O
  - Derive discretisation-specific mappings at run-time
  - Geometric Multigrid

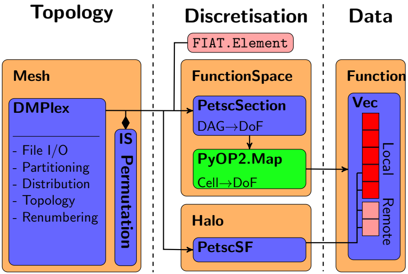


[1] M. Alnæs, A. Logg, K. Ølgaard, M. Rognes, and G. Wells. Unified Form Language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 40(2):9, 2014

Firedrake

Firedrake - Data structures

- DMPlex encodes topology
  - Parallel distribution
  - Application ordering
- Section encodes discretisation
  - Maps DAG to solution DoFs
  - Generated via FIAT element[1]
  - Derives PyOP2 indirection maps for assembly
- SF performs halo exchange
  - DMPlex derives SF from section and overlap

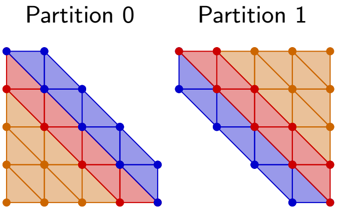


[1] R. Kirby. FIAT, A new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software (TOMS)*, 30(4):502–516, 2004

Firedrake

PyOP2 - Kernel execution

- Run-time code generation
  - Intermediate representation
  - Kernel optimisation via AST[1]
- Overlapping communication
  - Core: Execute immediately
  - Non-core: Halo-dependent
  - Halo: Communicate while computing over core
  - Imposes ordering constraint

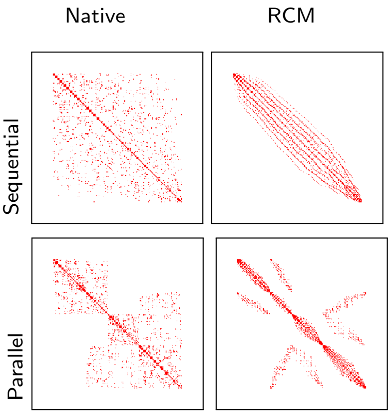


[1] F. Luporini, A. Varbanescu, F. Rathgeber, G.-T. Bercea, J. Ramanujam, D. Ham, and P. Kelly. Cross-Loop Optimization of Arithmetic Intensity for Finite Element Local Assembly. *Accepted for publication, ACM Transactions on Architecture and Code Optimization*, 2015

Firedrake

Firedrake - RCM reordering

- Mesh renumbering
  - Improves cache coherency
  - Reverse Cuhill-McKee (RCM)
- Combine RCM with PyOP2 ordering[1]
  - Filter cell reordering
  - Apply *within* PyOP2 classes
  - Add DoFs per cell (closure)



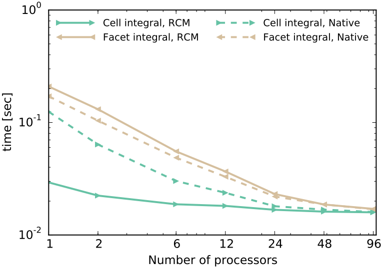
[1] M. Lange, L. Mitchell, M. Knepley, , and G. Gorman. Efficient mesh management in Firedrake using PETSc-DMPlex. *Submitted to SISC Special Issue*, 2015

Firedrake Performance

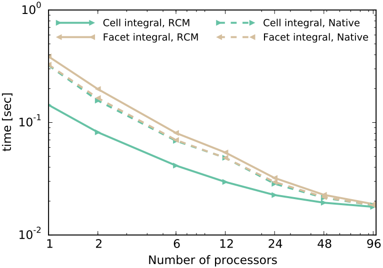
Indirection cost of assembly loops:

- Cell integral: `L = u * dx`
- Facet integral: `L = u('+')* dS`

100 loops on P_1



100 loops on P_3

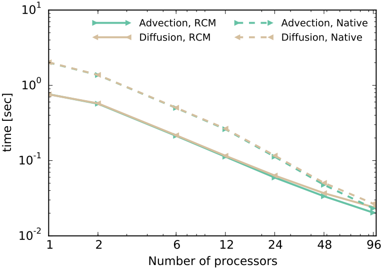


Firedrake Performance

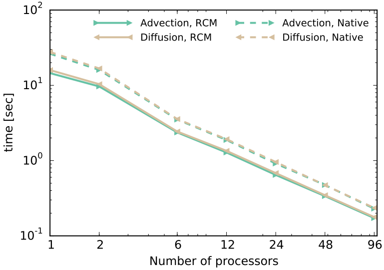
Advection-diffusion:

- Eq: $\frac{\partial c}{\partial t} + \bigtriangledown \cdot (\vec{u}c) = \bigtriangledown \cdot (\overline{\overline{\kappa}} \bigtriangledown c)$
- L-shaped mesh with ≈ 3.1 mio. cells

Matrix assembly on P_1



Matrix assembly on P_3

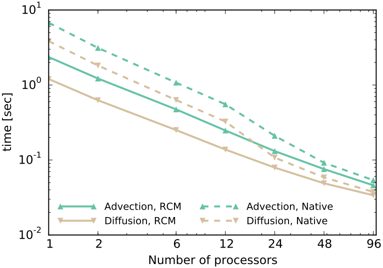


Firedrake Performance

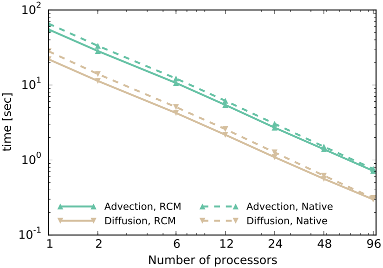
Advection-diffusion:

- Eq: $\frac{\partial c}{\partial t} + \nabla \cdot (\vec{u} c) = \nabla \cdot (\overline{\overline{\kappa}} \nabla c)$
- L-shaped mesh with $\approx$ 3.1 mio. cells

RHS assembly on P_1



RHS assembly on P_3

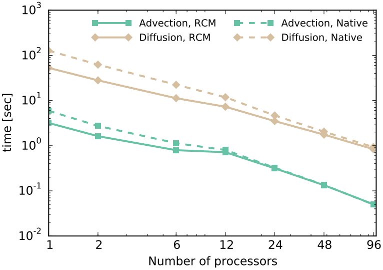


Firedrake Performance

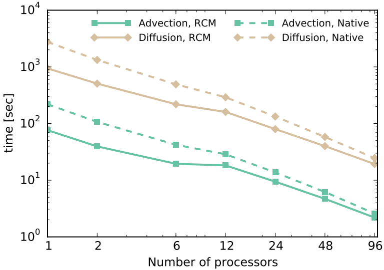
Advection-diffusion:

- Advection solver: CG + Jacobi
- Diffusion solver: CG + HYPRE BoomerAMG

Solve on P_1



Solve on P_3



Imperial College London

Summary

DMPlex mesh management

- Unified mesh reader/writer interface
  - Improves compatability and interoperability
  - Delegate I/O and it's optimisation
- Improved **`DMPlexDistribute`**
  - Run-time domain decomposition
  - Scalable overlap generation
  - All-to-all load balancing
- Firedrake FE environment
  - DMPlex as topology abstraction
  - Derive indirection maps for assembly
  - Compact RCM renumbering

Future work

- Parallel mesh file reads
- Anisotropic mesh adaptvity

Thank You

https://fluidityproject.github.io

www.firedrakeproject.org

http://www.archer.ac.uk/

EPSRC
Engineering and Physical Sciences Research Council

FEniCS '15
29 June - 1 July 2015
Imperial College London

Intel PCC

Platform for Research in Simulation Methods
PRISM

FEniCS project