# Flexible, Scalable Mesh and Data Management using PETSc DMPlex

Michael Lange
Department of Earth Science
and Engineering
Imperial College London, UK
m.lange@imperial.ac.uk

Matthew G. Knepley Computation Institute University of Chicago, USA knepley@gmail.com Gerard J. Gorman
Department of Earth Science
and Engineering
Imperial College London, UK
g.gorman@imperial.ac.uk

## **ABSTRACT**

Designing a scientific software stack to meet the needs of the next-generation of mesh-based simulation demands, not only scalable and efficient mesh and data management on a wide range of platforms, but also an abstraction layer that makes it useful for a wide range of application codes. Common utility tasks, such as file I/O, mesh distribution, and work partitioning, should be delegated to external libraries in order to promote code re-use, extensibility and software interoperability. In this paper we demonstrate the use of PETSc's DMPlex data management API to perform mesh input and domain partitioning in Fluidity, a large scale CFD application. We demonstrate that raising the level of abstraction adds new functionality to the application code, such as support for additional mesh file formats and mesh reordering, while improving simultation startup cost through more efficient mesh distribution. Moreover, the separation of concerns accomplished through this interface shifts critical performance and interoperability issues, such as scalable I/O and file format support, to a widely used and supported open source community library, improving the sustainability, performance, and functionality of Fluidity.

#### Keywords

Mesh, topology, partitioning, renumbering, Fluidity, PETSc

#### 1. INTRODUCTION

Scalable file I/O and efficient domain topology management present important challenges for many scientific applications if they are to fully utilise future exascale computing resources. Although these operations are common to many scientific codes they have received little attention in optimisation efforts, resulting in potentially severe performance bottlenecks for realistic simulations that require and generate large data sets. Moreover, due to a multitude of formats and a lack of convergence on standards for mesh and output data in the community there is only limited interoperability and very little code reuse among scientific applications for

common operations, such as reading and partitioning input meshes. Thus developers are often forced to create custom I/O routines or even use application-specific file formats, which further limits application portability and interoperability.

Designing a scientific software stack to meet the needs of the next-generation of simulation software technologies demands, not only scalable and efficient algorithms to perform data I/O and mesh management at scale, but also an abstraction layer that allows a wide variety of application codes to utilise them and thus promotes code reuse and interoperability. Such an intermediate representation of mesh topology has recently been added to PETSc [3], a widely used scientific library for the scalable solution of partial differential equations, in the form of the DMPlex data management API [14].

In this paper we demonstrate the use of PETSc's DMPlex API to perform mesh input and domain topology management in Fluidity [17], a large scale CFD application code that already uses the PETSc library as its linear solver engine. By utilising DMPlex as the underlying mesh management abstraction we not only add support for new mesh file formats, such as Exodus II [20], CGNS [18], Gmsh [9], Fluent Case [2] and MED [1], to Fluidity, but also enable the use of domain decomposition methods, data migration, and mesh renumbering techniques at run-time. Moreover, the separation of concerns allows PETSc parallel data management and HDF5 support to be independently optimized for the target platform, removing this complexity from the application code. Our refactoring of Fluidity provides significant performance benefits due to improved cache locality and mesh distribution during simulation initialisation, which we demonstrate with performance benchmarks performed on Archer, a Cray XC30 architecture.

## 2. BACKGROUND

The key challenge in designing software for large scale systems lies in the composition of abstractions and the definition of clearly defined yet flexible interfaces between them. Code reuse and inter-disciplinary cooperation necessitate deeper software stacks and thus configuration and extensibility will play a key role in designing the software stack of the future [5]. In this paper we therefore focus on the interaction between applications and their supporting libraries to provide the infrastructure for efficient data management at exascale.

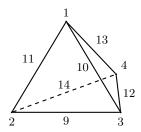
# 2.1 Fluidity

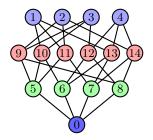
The primary user application in our work is Fluidity, an open source unstructured finite element code that uses mesh adaptivity to accurately represent a wide range of scales in a single numerical simulation without the need for nested grids. Fluidity is used in a number of different scientific areas including geophysical fluid dynamics, computational fluid dynamics, ocean modelling and mantle convection. Fluidity implements various finite element and finite volume discretisation methods and is capable of solving solving the Navier-Stokes equation and accompanying field equations in one, two and three dimensions.

Previous optimisation efforts have highlighted that file I/O, in particular during model initialisation, presents a severe performance bottleneck when running on large numbers of processes [11]. The primary reasons for this are a off-line domain partitioning and the need to store each partition using a file-per-process strategy.

#### 2.2 DMPlex

PETSc's ability to handle unstructured meshes is centred around DMPlex, a data management object that encapsulates the topology of unstructured grids to provide a range of functionalities common to many scientific applications. As shown in Figure 1, DMPlex stores the connectivity of the associated mesh as a layered directed acyclic graph (DAG), where each layer (stratum) represents a class of topological entities [14, 16]. This flexible yet efficient representation provides an abstract interface for the implementation of mesh management and manipulation algorithms using dimension-independent programming.





Vertex and edge numbering

Topological connectivity

Figure 1: DAG-based representation of a single tetrahedron in DMPlex.

DMPlex stores data by associating data with points in the DAG, allowing an arbitrary data size for each point. This can be efficiently encoded using the same AIJ data structure used for sparse matrices. This scheme is general enough to encompass any discrete data layout over a mesh. The association with points also means that data can be moved using the same set of scalable primitives that are used for mesh distribution.

DMPlex's internal representation of mesh topology also provides an abstraction layer that decouples the mesh from the underlying file format and thus allows support for multiple mesh file formats to be added generically. At the time of writing DMPlex is capable of reading input meshes in Exodus II, CGNS, Gmsh, Fluent-Case and MED formats. Moreover, DMPlex provides output routines that generate

solution output in HDF5-based XDMF format, while also storing the DMPlex DAG connectivity alongside the visual-isable solution data to facilitate checkpointing [3].

In addition to a range of I/O capabilities DMPlex also provides parallel data marshalling through automated parallel distribution of the DMPlex [15] and the pre-allocation of parallel matrix and vector data structures. Mesh partitioning is provided via internal interfaces to several partitioner libraries (Chaco, Metis/ParMetis) and data migration is based on PETSc's internal Star Forest communication abstraction (PetscSF) [3]. Additionally, DMPlex is designed to provide the connectivity data and grid hierarchies required by sophisticated preconditioners, such as geometric multigrid methods and "Fieldsplit" preconditioning for multi-physics problems, to speed up the solution process [4, 6].

## 2.3 Mesh Reordering

Mesh reordering techniques represent a powerful performance optimisation that can be utilised to increase cache coherency of the matrices required during the solution process [10, 12, 21]. The well-known Reverse Cuthill-McKee (RCM) algorithm, which can be used to reduce the bandwidth of CSR matrices, is implemented in PETSc allowing DMPlex to compute the required permutation of mesh entities directly from the domain topology DAG. The resulting permutation can then be applied to any discretisation derived from the stored mesh topology to improve the cache coherency of the associated CSR matrices.

## 3. FLUIDITY-DMPLEX INTEGRATION

Initial mesh input has been a scalability bottleneck in Fluidity due to the off-line mesh partitioning step. As illustrated in Figure 2a, the current preprocessor module uses Zoltan [8], which use ParMetis [13] for graph partitioning, to partition and distribute the initial simulation state to the desired number of processes before writing the partitioned mesh and data to disk, allowing the main simulation to read the pre-partitioned data in a parallel fashion.

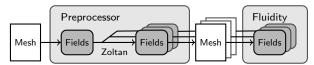
Fluidity's parallel mesh initialisation routines, however, rely on a file-per-process I/O strategy that require large numbers of individual files when running the application at scale. This has been shown to put significant pressure on the metadata servers in distributed filesystems, such as Lustre or PVFS, which ultimately has a detrimental effect on scalability when using sufficiently large numbers of processes [11].

# 3.1 Parallel Simulation Start-up

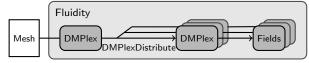
One of the objectives of this work, in addition to enhacing functionality and usability, is to alleviate Fluidity's start-up bottleneck by utilising DMPlex's mesh distribution capabilities to perform mesh partitioning at run-time. For this purpose, as shown in Figure 2b, a DMPlex topology object is created from the initial input mesh and immediately partitioned and distributed to all participating processes, allowing Fluidity's initial coordinate field to be derived from the DMPlex object in parallel. From the initial coordinate mesh all further discretisations and fields in the simulation state are then derived using existing functionality.

Using DMPlex as an intermediate representation for the underlying mesh topology has the following advantages:

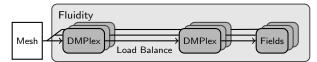
- Run-time mesh distribution and load balancing removes the need to store the partitioned mesh on disk, thus removing two costly I/O operations and reducing Fluidity's disk space requirements.
- Communication volume during startup is reduced, since only the topology graph is distributed. This is in contrast to the preprocessor, which partitions and distributes a fully allocated Fluidity state with multiple fields.
- Support for multiple previously unsupported mesh file formats is inherited from DMPlex, increasing application interoperability.



(a) Original Fluidity start-up based on off-line pre-processing.



(b) DMPlex-based start-up where an initial DMPlex object is distributed at run-time.



(c) Potential future workflow, where DMPlex performs the initial mesh read in parallel before a parallel load balancing step.

Figure 2: Workflow diagram for Fluidity simulation startup.

A key point to note about the DMPlex-based mesh initialisation approach is that by delegating the initial mesh read to PETSc any mesh format reader added to DMPlex in the future will automatically be inherited by the application code. Moreover, future performance optimisations, such as parallisation of the initial mesh file read, will also be available in Fluidity without any further changes to the application. Such an envisaged scenario is shown in Figure 2c, where an already parallel DMPlex object is created from the initial file, followed by a load balancing step before deriving the parallel Fluidity state.

## 3.2 Mesh Renumbering

One of the key components of the DMPlex integration is the derivation of Fluidity's initial coordinate mesh object from the distributed DMPlex, which includes the derivation of the data mapping required for halo exchanges in parallel. DMPlex is able to provides such a mapping from local nonowned degrees-of-freedom (DoFs) to their respective remote owners. However, since Fluidity halo objects require remote non-owned DoFs in the solution field to be located contiguously at the end of the solution vector ("trailing receives"

assumption), a node permutation is required when deriving Fluidity data structures from the mapping provided by DMPlex.

As a consequence, further node ordering permutations may be applied during the derivation of the initial field discretisation, such as the RCM renumbering provided by DMPlex. An optional renumbering step can be performed locally after the initial mesh distribution and added to the mesh initialisation routine with very little programming effort. As a result of Fluidity's run-time derivation of field discretisations from the underlying coordinate mesh, the RCM data layout of the initial reordering is inherited by all fields in the simulation, as shown in Figure 3. Moreover, as new mesh renumbering schemes are incorporating into PETSc, they will be automatically available to the application code.

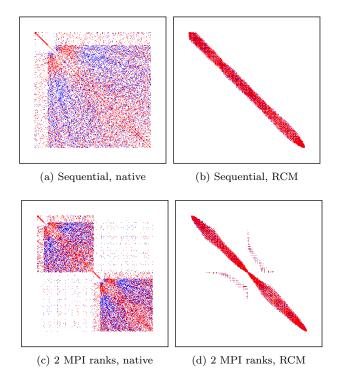


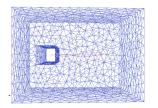
Figure 3: Effects of RCM reordering on the structure of the assembled pressure matrix.

## 4. RESULTS

The following benchmark tests were performed on the UK national supercomputer, a Cray XE30 with 4920 nodes connected via an Aries interconnect and a parallel Lustre filesystem  $^1$ . Each node consists of two 2.7 GHz, 12-core Intel E5-2697 v2 (Ivy Bridge) processors with 64GB of memory.

The benchmark runs simulate the flow past a square cylinder for 10 timesteps using a  $P_1^{\rm DG}-P_2$  discretisation [7], where a second-order pressure field is solved using Fluidity's multigrid algorithm and paired with a discontinuous first order velocity field that uses a GMRES solver with SOR preconditioning. The mesh used has been generated with the Gmsh mesh generator [9] and is shown in Figure 4.

<sup>1</sup>http://www.archer.ac.uk/



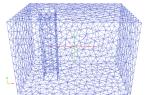


Figure 4: Three-dimensional benchmark mesh used to model flow past a cylinder.

#### 4.1 Mesh Initialisation

Figure 5 shows a comparison of the simulation start-up cost between the DMPlex-based implementation and the original preprocessor approach on 4 nodes (96 cores) with increasing mesh sizes up to approximately 3 million elements (weak scaling). The original start-up cost is hereby quantified as the sum of preprocessor and simulation initialisation times.

A clear improvement in overall start-up performance is shown in Figure 5a, although no significant improvement in direct file I/O, as shown in Figure 5b, can be determined. This is unsurprising, as file I/O is still completely dominated by the initial sequential read, although potential gains can be expected at larger scales due to removing the intermediate reads and writes of the partitioned mesh.

As highlighted in Figure 5c, the majority of the observed overall performance gains can be attributed to significantly improved mesh distribution via DMPlex. It is important to note here that DMPlex partitions and migrates only the mesh topology graph and its associated coordinate values, in contrast to the original preprocessor module that distributes fully assembled fields. As a result less data needs to be communicated during the mesh migration phase, resulting in significantly increased start-up performance.

#### 4.2 Mesh Renumbering

The overall simulation performance, including the effects of the mesh reordering derived from DMPlex, are evaluated in Figure 6. This benchmark compares the performance of both implementations by running 10 timesteps of the full simulation using a mesh with approximately 3 million elements on up to 96 cores. The results, shown in Figure 6a, indicate a consistent performance improvement of the DMPlex-based model with native mesh ordering over the preprocessor approach that increases with growing numbers of processes due to a smaller start-up overhead.

The effect of RCM mesh reordering is best demonstrated by analysing the two most expensive components of the simulation: the pressure field solve (Figure 6b) and the assembly of the velocity matrix (Figure 6c). Both components exhibit significant performance increases with RCM reordering on small numbers of processors that diminish as the simulation approaches the strong scaling limit. However, the benefits for overall simulation performance (Figure 6c) with RCM reordering decrease between 24 and 96 processes due to the fixed-cost start-up overhead of generating the permutation outweighing the solver and assembly benefits.

## 5. DISCUSSION

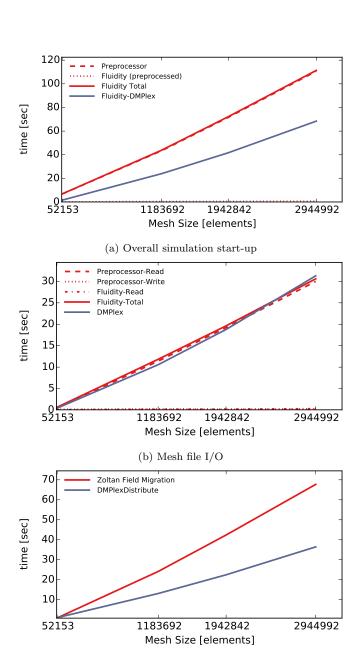
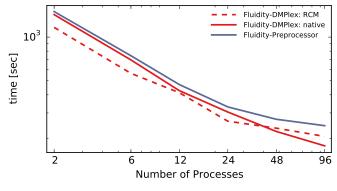
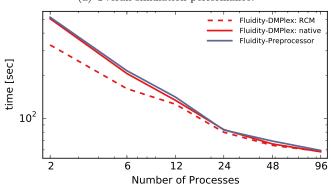


Figure 5: Comparison of total start-up cost between preprocessor and DMPlex-based approaches.

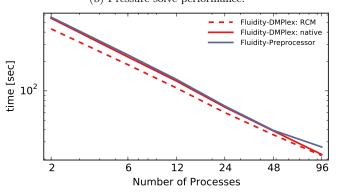
(c) Mesh distribution



(a) Overall simulation performance.



(b) Pressure solve performance.



(c) Velocity assembly performance.

Figure 6: Full simulation performance for 10 timesteps on approximately 3 million elements.

Achieving scalable performance with production-scale scientific applications on future exascale systems requires appropriate levels of abstraction across the entire software stack. In this paper we report progress on the integration of DM-Plex, a library-level domain topology abstraction, with the application code Fluidity in order to delegate a set of common mesh and data management tasks to a widely used library. We highlight the increased interoperability achieved through the inheritance of new mesh file format readers and demonstrate improved model initialisation performance through run-time mesh distribution routines provided by DMPlex.

The key benefit of the restructured model initialisation workflow, however, lies in the fact that responsibility for supporting various mesh file formats and optimising mesh file I/O now lies with the underlying library. This entails that any future extensions, such as new file formats or parallel mesh reader implementations, are automatically inherited by Fluidity, as well as other applications using PETSc, such as Firedrake [19] where we have also employed these abstractions. Moreover, by utilising a centralised mesh management API other types of mesh-based performance optimisations become available to the application, as highlighted by the seamless addition of the RCM renumbering feature.

#### 5.1 Future Work

A key contribution of this work lies in the fact that it enables future extensions and optimisations. Most crucially perhaps is the development of a fully parallel mesh input reader in PETSc in order to overcome the remaining sequential bottleneck during model initialisation. This change, however, requires a new default mesh format for Fluidity due to the inherently sequential nature of the Gmsh file format, which again highlights the need for abstraction when optimising mesh management.

In addition to the optimisation of mesh input and model initialisation, further integration of DMPlex throughout Fluidity is desirable to utilise DMPlex's advanced I/O features, such as the HDF5-based Xdmf output format. For this purpose closer integration is required, where additional discretisation data needs to be passed to PETSc to perform all the necessary field I/O. Moreover, DMPlex's mesh and data distribution utility may also be used to provide load balancing after mesh adaptation.

# 6. ACKNOWLEDGMENTS

This work was supported by the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk), and the Intel Parallel Computing Center program through grants to both the University of Chicago and Imperial College London. We would also like to thank Frank Milthaler for providing the test configurations used for benchmarking.

## 7. REFERENCES

- Med data model. http://www.code-aster.org/outils/med/.
- [2] ANSYS. FLUENT reference manual, 2015. Software Release Version 6.3.

- [3] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.
- [4] J. Brown, M. Knepley, D. May, L. McInnes, and B. Smith. Composable Linear Solvers for Multiphysics. In Parallel and Distributed Computing (ISPDC), 2012 11th International Symposium on, pages 55–62, June 2012.
- [5] J. Brown, M. G. Knepley, and B. F. Smith. Run-time extensibility and librarization of simulation software. *IEEE Computing in Science and Engineering*, 2015.
- [6] P. Brune, M. Knepley, and L. Scott. Unstructured Geometric Multigrid in Two and Three Dimensions on Complex and Graded Meshes. SIAM Journal on Scientific Computing, 35(1):A173-A191, 2013.
- [7] C. J. Cotter, D. A. Ham, and C. C. Pain. A mixed discontinuous/continuous finite element pair for shallow-water ocean modelling. *Ocean Modelling*, 26(1):86–90, 2009.
- [8] K. D. Devine, E. G. Boman, R. T. Heaphy, U. V. Çatalyürek, and R. H. Bisseling. Parallel hypergraph partitioning for irregular problems. SIAM Parallel Processing for Scientific Computing, February 2006.
- [9] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.
- [10] F. Günther, M. Mehl, M. Pögl, and C. Zenger. A Cache-Aware Algorithm for PDEs on Hierarchical Data Structures Based on Space-Filling Curves. SIAM Journal on Scientific Computing, 28(5):1634–1650, 2006.
- [11] 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.
- [12] G. Haase, M. Liebmann, and G. Plank. A Hilbert-order multiplication scheme for unstructured sparse matrices. *International Journal of Parallel*, *Emergent and Distributed Systems*, 22(4):213–220, 2007.
- [13] G. Karypis and V. Kumar. ParMETIS: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Department of Computer Science, University of Minnesota, 1997. http://www.cs.umn.edu/metis.
- [14] M. G. Knepley and D. A. Karpeev. Mesh Algorithms for PDE with Sieve I: Mesh Distribution. Sci. Program., 17(3):215–230, Aug. 2009.
- [15] M. G. Knepley, M. Lange, and G. J. Gorman. Unstructured overlapping mesh distribution in parallel. Submitted to ACM TOMS, 2015.
- [16] A. Logg. Efficient representation of computational meshes. *International Journal of Computational* Science and Engineering, 4:283–295, 2009.
- [17] M. D. Piggott, G. J. Gorman, C. C. Pain, P. A. Allison, A. S. Candy, B. T. Martin, and M. R. Wells.

- A new computational framework for multi-scale ocean modelling based on adapting unstructured meshes. International Journal for Numerical Methods in Fluids, 56(8):1003–1015, 2008.
- [18] D. Poirier, S. R. Allmaras, D. R. McCarthy, M. F. Smith, and F. Y. Enomoto. The cgns system, 1998. AIAA Paper 98-3007.
- [19] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. Kelly. Firedrake: automating the finite element method by composing abstractions. Submitted to ACM TOMS, 2015.
- [20] L. A. Schoof and V. R. Yarberry. EXODUS II: a finite element data model. Technical Report SAND92-2137, Sandia National Laboratories, Albuquerque, NM, 1994.
- [21] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-oblivious Mesh Layouts. ACM Trans. Graph., 24(3):886–893, July 2005.