A structure-exploiting numbering algorithm for finite elements on extruded meshes, and its performance evaluation in Firedrake

Gheorghe-Teodor Bercea[1], Andrew T. T. McRae[2,3,4], David A. Ham[3], Lawrence Mitchell[1,3], Florian Rathgeber[1,5], Luigi Nardi[1], Fabio Luporini[1], and Paul H. J. Kelly[1]

[1]Department of Computing, Imperial College London, London, SW7 2AZ, United Kingdom
[2]The Grantham Institute, Imperial College London, London, SW7 2AZ, United Kingdom
[3]Department of Mathematics, Imperial College London, London, SW7 2AZ, United Kingdom
[4]Department of Mathematical Sciences, University of Bath, Bath, BA2 7AY, United Kingdom
[5]European Centre for Medium-Range Weather Forecasts (ECMWF), Reading, RG2 9AX, United Kingdom

Correspondence to: Gheorghe-Teodor Bercea (gb308@doc.ic.ac.uk)

1 Responses to reviewers

1.1 Reviewer 1 comments

Responses to individual comments:

Comment 1: *Data structures for layered meshes have been considered and implemented before - certainly in single-purpose codes, but also in frameworks (DUNE's prism- grid module, e.g.); I am aware that providing a survey of respective approaches to grid numbering in such packages might be impossible to do, but I think a general discussion on what options actually exist (and what implications resp. choices might have) when designing the numbering scheme could make the paper stronger.*

Answer: We have added some additional discussion in the introduction around alternative approaches to this same problem.

Comment 2: *This is a bit related to the choice of title: at first reading I found myself expecting such a discussion; however, the paper clearly focuses on the approach followed in Firedrake (which is fine in itself, but a bit in contrast to the generic title and the abstract).*

Answer: The paper describes a numbering algorithm which is completely generic to finite element approaches. However, it is true that the evaluation of the approach is in Firedrake. We have therefore reworked the title to explicitly mention that the performance evaluation is in Firedrake. In addition we have added some mention of Firedrake as the tool for performance evaluation in the abstract.

Comment 3: *You chose to number the DOFs of an entity contiguously, such that all DOFs of an edge (or cell) would be contiguous in memory. However, for low order methods and when the key design goal is to allow vectorization, you might want to strictly keep a stride-1 access on corresponding DOFs in layers - effectively this would mean exchanging the l and d2 loops in Alg. 1. In any case, this choice depends on the type of operations we expect in simulations (whether we are strongly memory or compute bound, what the memory access patterns are, etc.), so a discussion on this would be interesting.*

Answer: We acknowledge that the ordering within each entity column is not unique. However, we do not see an obvious advantage to interchanging as suggested here. We have commented on this in section 3.2.

Comment 4: *As far as I got it, your concept of a stencil goes beyond the strict notion typically used for finite difference methods on structured grids: your stencils may also include element-local operations in finite-element- type methods (requiring a cell and its faces, edges, vertices) or also a face-based flux operation as in finite volume methods (which might require a face and its two adjacent cells). In any case, you might explain this in a bit more detail, and maybe state one or two examples.*

Answer: The reviewer is entirely correct. We have made our definition of a stencil explicit in a new section 2.4.

Comment 5: *What kind of unstructured mesh did you actually use for your results? You discuss in the paper that having a structured mesh as base mesh is advantageous for performance. Hence you might even explicitly address this issue by comparing results for a structured mesh (stored in an unstructured way) and one (or more?) typical unstructured meshes from applications.*

Answer: In the problem setup, we have described how we generate the base mesh, using Gmsh. Although the domain of computation is regular, the mesh itself is unstructured.

We believe the comparison to a topologically structured mesh is outwith the scope of the paper. In particular, the comments relative to structured base meshes are in place to indicate that we are explicitly depriving ourselves of these advantages, since we are aiming for an iteration algorithm that gives good performance irrespective of the base domain. Our results demonstrate that, for layered meshes, a reasonable base numbering (obtained in our case via RCM) is sufficient to obtain performance close to hardware bounds at significantly fewer layers than are scientifically interesting.

Comment 6: *As my only major suggestion, I would like to encourage you to switch from GFlop/s to GB/s in all performance plots: as you are in a memory-bound regime and the numbering scheme primarily addresses achievable "valuable bandwidth", "GB/s" would be the natural metric.*

Answer: We have clarified in section 4.3 that the relevant bound is, in fact, operation count, and not bandwidth. We were ourselves surprised by this conclusion, however the performance results support this hypothesis. As such, GFlop/s is the correct metric.

Comment 7: *You might check whether having a log-scale for x-axes makes the results for few layers better visible*

Answer: We tried this, but it did not create a more useful figure.

Comment 8: *It would be helpful to a add a sketch for illustration of the indexing scheme defined in Eq. (5)*

Answer: We agree and have taken the opportunity to do so as Figure 3.

Comment 9: *I was wondering what kind of stencil a DG0xDG0 discretization would produce for the residual; aren't all accesses element-local then? In general, would it make sense to add a table (or similar) that describes which entities are accessed for the various discretisations?*

Answer: This is correct, and Figure 5 shows the degrees of freedom and therefore entities which are accessed in each case. Additionally, we have fixed the suggested typos.

1.2 Reviewer 2 comments

Responses to individual comments:

Comment 1: *I assume that λ defined as number of extruded layers stands for the number of vertices of a vertical mesh, rather than the number of segments in the vertical. In this case, looking at Equation 5: d_2 can be either 0 (for vertices) or 1 (for segments). Then the statement $0 \leq l \leq \lambda - d_2$ will let λ go out of bounds, unless the second $\leq$ symbol is replased by $<$. Similar observation for l-loop in Algorithm 3. In case this assumption is wrong, it should be made more clear what is meant with layers.*

Answer: λ is in fact the number of segments in the vertical. We have explicitly defined λ accordingly and ensured that the usage is consistent throughout the manuscript.

Comment 2: *The title suggests that besides a memory layout for function spaces, also a numbering strategy in the horizontal would be discussed.*

Answer: We do not think that the title implies this, however, we have changed the title in response to reviewer 1. We hope that this title is less ambiguous in what it is suggesting the contributions of the paper are. We feel that the contributions of the paper are expressly stated and the title does not read contrary to those claims.

Comment 3: *Looking at Figure 3., how is the numbering of $n + \#$ related to $m + \#$, and possibly nodes internal to the triangle? Are $m + \#$ and $n + \#$ related to different function spaces?*

Answer: The nodes internal to the triangle would be numbered in a similar way, but we feel that this would unnecessarily complicate the figure. Because the horizontal mesh is unstructured, there is no simple relationship between m and n, however our results show that using a suitable ordering of the horizontal mesh (such that m and n are typically "close") is important for performance. A given function space will have degrees of freedom associated with one or more entity types, for example a continuous cubic space in the horizontal combined with a continuous linear mesh in the vertical would have the degrees of freedom shown in the figure, plus one degree of freedom per horizontal facet.

Comment 4: *Given the title I would have expected to see discussed what would be the impact (on e.g. performance) to number $n + \#$ (as in Figure 3) first in vertical fashion (zig-zag up-down, rather than right-left).*

Answer: Reviewer 1 also had a similar comment. We acknowledge that the ordering in each entity column is not unique. We do not see an obvious advantage to zig-zag up-down as opposed to left-right numbering. We have added some text to section 3.2 in this vein. We note that we do not believe the title claims that this is the best numbering algorithm on extruded meshes, merely an approach which works well.

Comment 5: *It would help to see a visualisation of a practical numbering for a mesh of a few triangles and few levels, for a few function function space configurations, besides Algorithm 1.*

Answer: We do not believe this would add clarity to the paper. For example, the newly added Figure 3 merely numbers the topological entities on a single extruded triangle and is already complex.

Comment 6: *In Algorithm 1, first "dofsfs" is assigned with round brackets, later with square brackets.*

Answer: Thanks, fixed.

Comment 7: *In Algorithm 1, the second last line makes reference to l, which is invalid outside the vertical loop.*

Answer: Thanks, fixed.

Comment 8: *In Algorithm 1, perhaps exchanging loops l and d_2 can avoid the last 2 lines by looping l in $0, 1, ..., \lambda - 1 - d_2$? (possibly without changing the resulting order)*

Answer: Unfortunately this would change the resulting numbering.

Comment 9: *In Algorithm 3, can I assume that for a single DG-DG cell-entity, $(dof_0, dof_1, ..., dof_{k-1})$ are contiguous?*

Answer: Yes.

Comment 10: *In Algorithm 3, subscript fs in $Lfs(v)$ missing*

Answer: Thanks, fixed.

Comment 11: *In Algorithm 3, d_2 unassigned, should be referenced as subscript of V?*

Answer: Fixed by making d_2 an explicit input to the algorithm.

Comment 12: *In Figure 5, almost none of the results have reached as mentioned the "performance plateau" with 100 layers for the badly ordered mesh. It would be interesting to see how many layers are required for all discretizations to reach this plateau.*

Answer: The reviewer is correct that in the badly ordered case almost none of the results have reached the performance plateau, we have reworded the discussion to address this. We do not believe that extending the experiment until we achieve some plateau in the "bad" case would be of practical interest since typical simulations have tens, not hundreds of layers in the vertical.

Comment 13: *Is there an expected benefit in going higher order to reach the plateau with less layers?*

Answer: There is indeed an expected benefit in going to high order, we have added a note to this effect at the end of the discussion.

A structure-exploiting numbering algorithm for finite elements on extruded meshes~~which avoids the unstructured mesh penalty~~, and its performance evaluation in Firedrake

Gheorghe-Teodor Bercea[1], Andrew T. T. McRae[2,3,4], David A. Ham[3], Lawrence Mitchell[1,3], Florian Rathgeber[1,5], Luigi Nardi[1], Fabio Luporini[1], and Paul H. J. Kelly[1]

[1]Department of Computing, Imperial College London, London, SW7 2AZ, United Kingdom
[2]The Grantham Institute, Imperial College London, London, SW7 2AZ, United Kingdom
[3]Department of Mathematics, Imperial College London, London, SW7 2AZ, United Kingdom
[4]Department of Mathematical Sciences, University of Bath, Bath, BA2 7AY, United Kingdom
[5]European Centre for Medium-Range Weather Forecasts (ECMWF), Reading, RG2 9AX, United Kingdom

Correspondence to: Gheorghe-Teodor Bercea (gb308@doc.ic.ac.uk)

Abstract. We present a generic algorithm for numbering and then efficiently iterating over the data values attached to an extruded mesh. An extruded mesh is formed by replicating an existing mesh, assumed to be unstructured, to form layers of prismatic cells. Applications of extruded meshes include, but are not limited to, the representation of 3D high aspect ratio domains employed by geophysical finite element simulations. These meshes are structured in the extruded direction. The algorithm presented here exploits this structure to avoid the performance penalty traditionally associated with unstructured meshes. We evaluate ~~our algorithm~~ the implementation of this algorithm in the Firedrake finite element system on a range of low compute intensity operations which constitute worst cases for data layout performance exploration. The experiments show that having structure along the extruded direction enables the cost of the indirect data accesses to be amortized after 10-20 layers as long as the underlying mesh is well-ordered. We characterise the resulting spatial and temporal reuse in a representative set of both continuous-Galerkin and discontinuous-Galerkin discretisations. On meshes with realistic numbers of layers the performance achieved is between 70% and 90% of a theoretical hardware-specific limit.

Keywords: extruded meshes, code generation, finite elements, locality optimisation

1 Introduction

In the field of numerical simulation of fluids and structures, there is traditionally considered to be a tension between the computational efficiency and ease of implementation of structured grid models, and the flexible geometry and resolution offered by unstructured meshes.

In particular, one of the grand challenges in simulation science is modelling the ocean and atmosphere for the purposes of predicting the weather or understanding the Earth's climate system. The current generation of large-scale operational atmosphere and ocean models almost all employ structured meshes (Slingo et al., 2009). However, requirements for geometric flexibility as well as the need to overcome scalability issues created by the poles of structured meshes has led in recent years

to a number of national projects to create unstructured mesh models (Ford et al., 2013; Zängl et al., 2015; Skamarock et al., 2012).

The ocean and atmosphere are thin shells on the Earth's surface, with typical domain aspect ratios in the thousands (oceans are a few kilometres deep but thousands of kilometres across). Additionally the direction of gravity and the stratification of the ocean and atmosphere create important scale separations between the vertical and horizontal directions. The consequence of this is that even unstructured mesh models of the ocean and atmosphere are in fact only unstructured in the horizontal direction, while the mesh is composed of aligned layers in the vertical direction. In other words, the meshes employed in the new generation of models are the result of extruding an unstructured two-dimensional mesh to form a layered mesh of prismatic elements.

This layered structure was exploited in Macdonald et al. (2011) to create a numbering for a finite volume atmospheric model such that iteration from one cell to the next within a vertical column required only direct addressing. They show that when only paying the price of indirect addressing on the base mesh there is less than 5% performance difference between two implementations of an atmospheric model which treat the same icosahedral mesh first as fully structured and then as partially structured (extruded). One of the caveats of that comparison is that the underlying mesh is fully structured in both cases which presents an advantage to the indirect addressing scheme which is not present for more general unstructured meshes.

Exploiting the anisotropic nature of domains has seen various software developments in various fields. For example `p6est` (Isaac et al., 2015) and (Isaac, 2015, §2.3), a package for 2+1 dimensional adaptive mesh refinement, was developed to maintain columnwise numbering for numerical reasons in ice sheet modelling, but does not support general unstructured base meshes. The DUNE-PrismGrid module (Gersbacher, 2012) provides extruded meshes for any base DUNE grid, but does not describe a degree of freedom numbering or provide detailed performance characteristics of the iteration on extruded meshes. The Model for Prediction Across Scales (MPAS) uses a column innermost numbering for their C-grid atmospheric and ocean model (Koziel et al., 2015). Their implementation is limited to the single discretisation employed by that model.

A key motivation for this work was to provide an efficient mechanism for the implementation of the layered finite element numerics which have been adopted by the UK Met Office's Gung Ho programme to develop a new atmospheric dynamical core. The algorithms here have been adopted by the Met Office for this purpose (Ford et al., 2013). While geophysical applications motivate this work, the algorithms and their implementation in Firedrake (Rathgeber et al., 2015) are more general and could be applied to any high aspect ratio domain.

1.1 Contributions

- We generalize the numbering algorithm in Macdonald et al. (2011) to the full range of finite element discretisations.
- We demonstrate the effectiveness of the algorithm with respect to absolute hardware performance limits.

2 Unstructured Meshes

In this section we briefly restate the data model for unstructured meshes introduced in Logg (2009); Knepley and Karpeev (2009). In Section 2.2 we rigorously define a *mesh*, explain mesh topology, geometry and numbering. In Section 2.3 we explain how data may be associated with meshes.

2.1 Terminology

When describing a mesh, we need some way of specifying the neighbours of a given entity. This is always possible using *indirect addressing* in which the neighbours are explicitly enumerated, and sometimes possible with *direct addressing* where a closed form mathematical expression suffices.

In what follows we start with a *base mesh* which we will *extrude* to form a mesh of higher topological dimension. Due to geophysical considerations, we refer to the plane of the base mesh as the *horizontal* and to the layers as the *vertical*.

We will also employ the definition of a *graph* as a set V and a set E of edges where each edge represents the relationships between the elements of the set V.

2.2 Meshes

A mesh is a decomposition of a simulation domain into non-overlapping polygonal or polyhedral cells. We consider meshes used in algorithms for the automatic numerical solution of partial differential equations. These meshes combine topology and geometry. The topology of a mesh is composed of mesh entities (such as vertices, edges, cells) and the adjacency relationships between them (cells to vertices or edges to cells). The geometry of the mesh is represented by coordinates which define the position of the mesh entities in space.

Every mesh entity has a topological dimension given by the minimum number of spatial dimensions required to represent that entity. We define D to be the minimum number of spatial dimensions needed to represent a mesh and all its entities. A vertex is representable in zero-dimensional space, similarly an edge is a one-dimensional entity and a cell a D-dimensional entity. In a two-dimensional mesh of triangles, for example, the entities are the vertices, edges and triangle cells with topological dimensions 0, 1 and 2 respectively. The minimum number of geometric dimensions needed to represent the mesh and all its entities is $D = 2$.

A mesh can be represented by several graphs. Each graph consists of a multi-type set V and a typed adjacency relationship Adj_{d_1,d_2} between d_1- and d_2-typed elements in V. The type of an entity in V is simply its dimension. The adjacency graphs will always map from a set of uniform dimension to a set of uniform dimension. Attaching types to elements of V enables graphs to capture the relationships between different mesh entities, for example cells and vertices, edges and vertices.

We write V_d to mean the set of mesh entities of topological dimension d where $0 \leq d \leq D$:

$$V_d = \{(d, i) \mid 0 \leq i \leq N_d - 1\}, \tag{1}$$

where N_d is the number of entities of dimension d. The set V is then simply the union of the V_ds:

$$V = \bigcup_{0 \leq d \leq D} V_d. \tag{2}$$

Every mesh entity has a number of adjacent entities. The mesh-element connectivity relationships are used to specify the way mesh entities are connected. For a given mesh of topological dimension D there are $(D+1)^2$ different types of adjacency relationships. To define the mesh, only a minimal subset of relationships from which all the others can be derived is required. For example, as shown in Logg (2009), the complete set of adjacency relationships may be derived from the cell-vertex adjacency.

We write

$$\mathrm{Adj}_{d_1,d_2}(v) = (v_1, v_2, \ldots, v_k), \tag{3}$$

to specify the entities $v_1, v_2, \ldots, v_k \in V_{d_2}$ adjacent to $v \in V_{d_1}$.

In a mesh with a very regular topology, there may be a closed form mathematical expression for the adjacency relationship $\mathrm{Adj}_{d_1,d_2}(v)$. Such meshes are termed *structured*. However since we are also interested in supporting more general *unstructured meshes*, we must store the lists of adjacent entities explicitly.

2.3 Attaching data to meshes

Every mesh entity has a number of values associated with it. These values are also known as *degrees of freedom* and they are the discrete representation of the continuous data fields of the domain. As the degrees of freedom are uniquely associated with mesh entities, the mesh topology can be used to access the degrees of freedom local to any entity using the connectivity relationships.

A *finite element discretisation* associates a number of degrees of freedom with each entity of the mesh. A *function space* uses the discretisation to define a numbering for all the degrees of freedom. Multiple different function spaces may be defined on a mesh and each function space may have several data fields associated with it. In the case of a triangular mesh for example, a piecewise linear function space will associate a degree of freedom with every vertex of the mesh while a cubic function space will associate one degree of freedom with every vertex, two degrees of freedom with every edge and one degree of freedom with every cell. In the former case there will be three degrees of freedom adjacent to a cell, and a total of ten in the latter case.

The data associated with the mesh also needs to be numbered. The choice of numbering can have a significant effect on the computational efficiency of calculations over the mesh (Günther et al., 2006; Lange et al., 2015; Yoon et al., 2005).

2.4 Kernels and stencils

The most common operation performed on meshes is the local application of a function or *kernel* while traversing, or *iterating* over a homogeneous subset of mesh entities. ~~When iterating over a specific mesh entity type, the kernel often consists of a stencil-like operation accessing nearby~~ The kernel is executed once for each such mesh entity and acts on the degrees of freedom

in a *stencil* composed of the mesh entities adjacent to the the iterated entity. For example, ~~in~~ a finite element ~~simulation over a triangle mesh, when iterating over a cell, the kernel might require the~~ operator evaluating an integral over the domain would iterate over the mesh cells and access data through a stencil comprising the degrees of freedom on ~~the vertices, edgesand the interior of the triangle~~that cell and its adjacent facets, edges, and vertices. For a more in-depth discussion on the construction of stencils on unstructured meshes, the reader is referred to Logg (2009) and Knepley and Karpeev (2009). In theory, this requires cell-to-facets, cell-to-edges, and cell-to-vertices adjacency relationships (cell-to-cell is implicit). In practice the three different relationships may be composed into a single adjacency relationship which references the data associated with all the different adjacent entity types.

In the unstructured case, we store an explicit list (also known as a *map*) $L(e)$ for each type of stencil operation which given a topological entity e returns the set of degrees of freedom in the stencil at that entity.

3 Extruded Meshes

In Section 3.1 we introduce extruded meshes and in Section 3.2 we show how the entities and the data are to be numbered. In Section 3.3 we present the extruded mesh iteration algorithm and the offset computation for the direct addressing scheme along the vertical direction.

3.1 Definition of an Extruded Mesh

An extruded mesh consists of a base mesh which is replicated a fixed number of times in a layered structure[1]. A mesh of topological dimension D becomes an extruded mesh of topological dimension $D + 1$.

The mesh definition can be extended to include extruded meshes. Let mesh $M = (V, \mathrm{Adj})$ be a non-extruded mesh where Adj stands for all the valid adjacency relationships of M. An extruded mesh which has M as the base mesh can be defined as a triple $(V^{\mathrm{extr}}, \mathrm{Adj}^{\mathrm{extr}}, \lambda)$ where $\mathrm{Adj}^{\mathrm{extr}}$ is the set of valid adjacency relationships and $\lambda \in \mathbb{N}^+$ is the number of ~~layers of the extruded mesh .~~ intervals over which the mesh is extruded. This implies that there are $\lambda + 1$ vertices in the extruded direction. Before we can define V^{extr} and $\mathrm{Adj}^{\mathrm{extr}}$ several concepts have to be introduced.

3.1.1 Tensor product cells

The effect of the extrusion process on the base mesh can always be captured by associating a line segment with the vertical direction. We write D_b for the topological dimension of the base mesh while the topological dimension of the vertical mesh is always equal to 1.

As a consequence, the cells of the extruded mesh are prisms formed by taking the tensor product of the base mesh cell with the vertical line segment. For example, each triangle ~~becoems~~ becomes a triangular prism. The construction of tensor product cells and finite element spaces on them is considered in more detail in McRae et al. (2016).

[1] For ease of exposition, we discuss the case where each mesh column contains the same number of layers, however this is not a limitation of the method and algorithms presented here

3.1.2 Extruded Mesh Entities

The extrusion process introduces new types of mesh entities reflecting the connectivity between layers. The pairs of corresponding entities of dimension d in adjacent layers are connected using entities of dimension $d+1$. In a triangular mesh for example (Fig. 1), the corresponding vertices are connected using vertical edges, edges contained in each layer are connected by quadrilateral facets and the 2D triangle faces are connected by a 3D triangular prism (Fig. 2).

Figure 1. Extruded mesh entities belonging to the base mesh to be extruded (left to right): vertices, horizontal edges, horizontal facets.

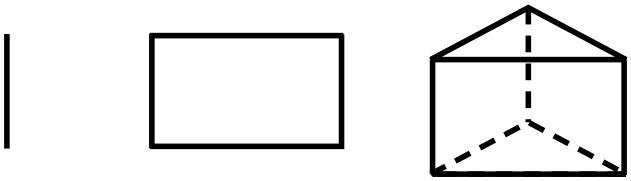


Figure 2. Mesh entities used in the extrusion process to connect entities in Fig. 1 (left to right): vertical edges, vertical facets, 3D cells.

The topological dimension on its own is no longer enough to distinguish between the different types of entities and their orientation. Instead entities are characterised by a pair composed of the horizontal and vertical dimensions. In the case of a 2D triangular base mesh the set of dimensions is $\{0,1,2\}$. The line segment of the vertical can be described by the set of dimensions $\{0,1\}$. The Cartesian product of the two sets yields a set of pairs (4) which can be used to uniquely identify mesh entities.

$$\{(0,0),(0,1),(1,0),(1,1),(2,0),(2,1)\} \tag{4}$$

We refer to the components of each pair as the *horizontal* and *vertical* dimension of the entity respectively. Table 1 shows the mapping between the mesh entity types and their descriptor.

3.1.3 Extruded Mesh Entity Numbering

We write V_{d_1,d_2} to denote the set of topological entities which are the tensor product of entities of dimensions d_1 in the horizontal and d_2 in the vertical ($0 \leq d_1 \leq D_b$ and $0 \leq d_2 \leq 1$):

$$V_{d_1,d_2} = \{((d_1,d_2),(i,l)) \mid 0 \leq i \leq N_{d_1} - 1,\ 0 \leq l \leq \lambda - d_2\}, \tag{5}$$

where N_{d_1} is the number of entities of dimension d_1 in the base mesh and λ is the number of ~~extruded layers~~edges in the extruded direction. The subtraction of d_2 from ~~the number of layers~~ λ accounts for the fencepost error caused by the fact that there is always one fewer edge than vertex in the vertical direction.

The complete set of extruded mesh entities is then

$$V^{\text{extr}} = \bigcup_{\substack{0 \leq d_1 \leq D_b \\ 0 \leq d_2 \leq 1}} V_{d_1,d_2}. \tag{6}$$

These entities are the drawn for the case of an extruded triangle in Fig. 3.

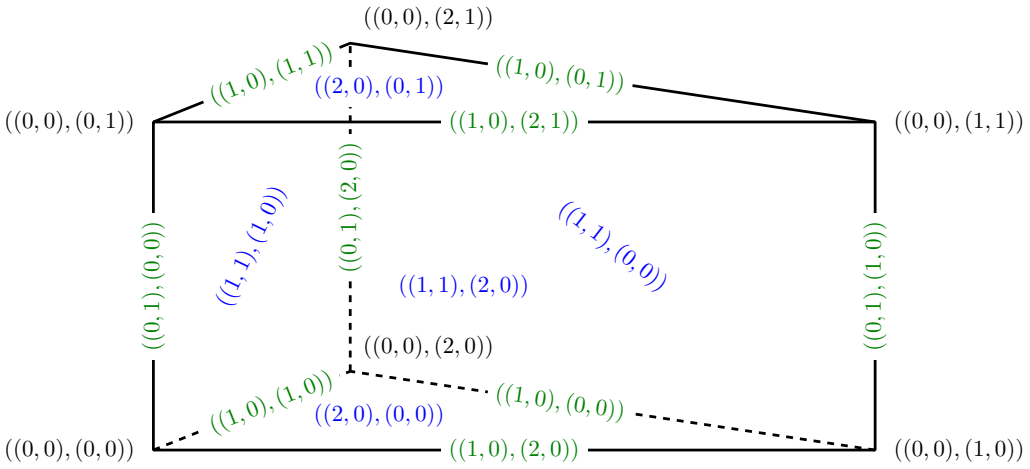


Figure 3. Numbering of the topological entities of an extruded cell for the case of an extruded triangle. The cell itself has numbering $((2,1),(0,0))$ (not shown), the other entities are numbered as shown with vertices in black, edges in green, and faces in blue.

Similarly we must extend the indexing of the adjacency relationships, writing:

$$\text{Adj}^{\text{extr}}_{(d_1,d_2),(d_3,d_4)}(v) = (v_1, v_2, \ldots, v_k), \tag{7}$$

where $v \in V_{d_1,d_2}$ and $v_1, v_2, \ldots, v_k \in V_{d_3,d_4}$.

3.2 Attaching data to extruded meshes

Identically to the case of non-extruded meshes, function spaces over an extruded mesh associate degrees of freedom with the (extended) set of mesh entities. A constant number of degrees of freedom is associated with each entity of a given type.

If we can arrange that the degrees of freedom are numbered such that the vertical entities are "innermost", it is possible to use direct addressing for the vertical part of any mesh iteration, significantly reducing the computational penalty introduced by using an indirectly addressed, unstructured base mesh. Algorithm 1 implements this "vertical innermost" numbering algorithm. The critical feature of this algorithm is that degrees of freedom associated with vertically adjacent entities have adjacent global numbers. The outcome of this vertical numbering is shown in Fig. 4. The global numbering algorithm is orthogonal to any base mesh decomposition strategy used to support execution on distributed memory parallel systems. The numbering order within each entity column is not unique, for example one could interchange the l and d_2 loops in Algorithm 1. However, our choice maximises cache-line usage on a per-element basis.

Algorithm 1 Computing the global numbering for degrees of freedom on an extruded mesh

Input: V : the set of base mesh entities

Input: λ : the number of ~~layers~~vertical intervals

Input: $\delta((d_1,d_2))$: the number of DoFs associated with each (d_1,d_2) entity

Output: $\mathsf{dofs_{fs}}$: the degrees of freedom associated with each entity

```
c ← 0 {Loop over base mesh entities}
for (d1, i) in V do
    {Loop over layers}
    for l in {0, 1, ..., λ − 1} do
        {Number the horizontal layer, then the connecting entity above it}
        for d2 in {0, 1} do
            {Assign the next δ((d1, d2)) global DoF numbers to this entity}
            dofs_fs((d1, d2), (i, l)) ← c, c + 1, ..., c + δ((d1, d2)) − 1
            c ← c + δ((d1, d2))
        end for
    end for
    {Number the top horizontal layer of this column}
    ~~dofs_fs[((d1, 0), (i, l)] ← c, c + 1, ..., c + δ((d1, 0)) − 1~~ dofs_fs((d1, 0), (i, λ)) ← c, c + 1, ..., c + δ((d1, 0)) − 1
    c ← c + δ((d1, 0))
end for
```

3.3 Iterating over extruded meshes

Iterating over the mesh and applying a kernel to a set of connected entities (stencil) is the key operation used in mesh-based computations.

The global numbering of the degrees of freedom allows stencils to be calculated using a direct addressing scheme when accessing the degrees of freedom of vertically adjacent entities. We assume that the traversal of the mesh occurs over a set of mesh entities which is homogeneous (a set containing only cells for example). Degrees of freedom belonging to vertically

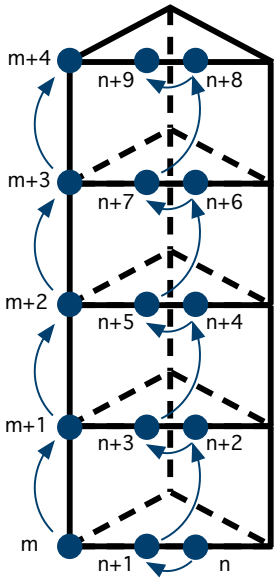


Figure 4. Vertical numbering of degrees of freedom (shown in filled circles) associated with vertices and horizontal edges. Only one set of vertically aligned degrees of freedom of each type is shown. The arrows outline the order in which the degrees of freedom are numbered.

adjacent entities, accessed by two consecutive kernel applications on the same column, have a constant offset between them. The offset is given by the sum of degrees of freedom attached to the two vertically adjacent entities contained in the stencil:

$$\delta((d,0)) + \delta((d,1)) \tag{8}$$

Let S be the stencil of a kernel which needs to access the values of the degrees of freedom of a field f defined on a function space fs. Let $L_{\mathsf{fs}}(v) = (\mathsf{dof}_0, \mathsf{dof}_1, ..., \mathsf{dof}_{k-1})$ be the list of degrees of freedom of the stencil for an input entity $v \in V_{d_1,d_2}$.

The lists of degrees of freedom accessed by S could be provided explicitly for all the input entities v. Using the previous result we can instead reduce the number of explicitly provided lists by a factor of λ. For each column we visit, the only explicit accesses required are the ones to the degrees of freedom at the bottom of the column. The degrees of freedom identifiers for the rest of the stencil applications in the same column can be obtained by adding a multiple of the constant vertical offset to each degree of freedom in the bottom explicit list.

For a given stencil function S an offset can be computed for each degree of freedom in the corresponding explicit list L_{fs}. As the ordering of the degrees of freedom in the stencil is fixed (by consistent ordering of mesh entities) the vertical offset only needs to be computed once for a particular function space fs.

The algorithm for computing the vertical offset is presented in Algorithm 2. Note that since the offset for two vertically aligned entity types is the same, only the base mesh entity type is considered.

Algorithm 2 Computation of vertical offsets

Input: k : number of degrees of freedom accessed by stencil function S

Input: $E_S(i)$: the base mesh entity type of the i-th degree of freedom accessed by S

Input: $\delta((d_1,d_2))$: the number of DoFs associated with each (d_1,d_2) entity

Output: $\mathsf{offset}_{S,\mathsf{fs}}$: the vertical offset for function space fs given stencil S

for i in $\{0,1,...,k-1\}$ **do**

$d \leftarrow E_S(i)$

$\mathsf{offset}_{S,\mathsf{fs}}(i) \leftarrow \delta((d,0)) + \delta((d,1))$

end for

If $(\mathsf{dof}_0, \mathsf{dof}_1, ..., \mathsf{dof}_{k-1})$ is the explicit list of degrees of freedom for the initial layer to which the stencil can be applied, then the list of degrees of freedom for the n^{th} application of the stencil along the vertical ~~$(n < \lambda)$~~ is given by:

$$(\mathsf{dof}_0 + n \times (\mathsf{offset}_{S,f}(0)), ..., \mathsf{dof}_{k-1} + n \times (\mathsf{offset}_{S,f}(k-1))) \tag{9}$$

Algorithm 3 shows the iteration algorithm working for a single field f on a function space fs. The stencil function S is applied to the entities of each column in turn. Each time the algorithm moves on to the next vertically adjacent entity, the indices of the degrees of freedom accessed are incremented by the vertical offset $\mathsf{offset}_{S,\mathsf{fs}}$. The algorithm is also applicable to stencil functions of multiple fields defined on the same function space since the data associated with each field is accessible using the same set of degree of freedom numbers. The extension to fields from different function spaces just requires explicit lists L_{fs} for each space.

4 Performance Evaluation

In this section, we test the hypothesis that iteration exploiting the extruded structure of the mesh amortizes the unstructured base mesh overhead of accessing memory through explicit neighbour lists. We also show that the more layers the mesh contains, the closer its performance is to the hardware limits of the machine.

We validate our hypotheses in the Firedrake finite element framework (Rathgeber et al., 2015). Although we restrict our performance evaluation to examples drawn from finite element discretisations, the algorithms we have presented can be applied to any mesh-based discretisation.

In Section 4.1 we describe the design of the experiments undertaken. The hardware platforms and the methodology used are described in Section 4.2 followed by results and discussion in Sections 4.3 and 4.4 respectively.

4.1 Experimental Design

The design space to be explored is parameterized by number of layers and the manner in which the data is associated with the mesh and therefore accessed. In establishing the relationship between the performance and the hardware we examine performance on two generations of processors and varying process counts.

Algorithm 3 Iteration of a stencil function over an extruded mesh

Input: V: iteration set of base mesh entities
Input: d_2 : the dimension of vertical iteration entities
Input: λ : the number of vertical intervals
Input: S: stencil function to be applied to the degrees of freedom of field f
Input: L_{fs}: set of explicit lists of degrees of freedom for function space fs
Input: $\mathsf{offset}_{S,\mathsf{fs}}$: the vertical offset for function space fs given stencil S
  for v in V **do**
    ~~$(\mathsf{dof}_0, \mathsf{dof}_1, ..., \mathsf{dof}_{k-1}) \leftarrow L(v)$~~ $(\mathsf{dof}_0, \mathsf{dof}_1, ..., \mathsf{dof}_{k-1}) \leftarrow L_{\mathsf{fs}}(v)$
    for l in $\{0, 1, ..., \lambda - d_2\}$ **do**
      $S(f(\mathsf{dof}_0), f(\mathsf{dof}_1), ..., f(\mathsf{dof}_{k-1}))$
      for j in $\{0, 1, ..., k-1\}$ **do**
        $\mathsf{dof}_j \leftarrow \mathsf{dof}_j + \mathsf{offset}_{S,\mathsf{fs}}(j)$
      end for
    end for
  end for

4.1.1 Choosing the computation

Numerical computations of integrals are the core mesh iteration operation in the finite element method. We focus on residual (vector) assembly for two reasons. First, in contrast to Jacobian assembly, there are no overheads due to sparse matrix insertion; the experiment is purely a test of data access via the mesh indirections. Second, residual evaluation is the assembly operation with the lowest computational intensity and therefore constitutes a worst-case scenario for data layout performance exploration.

Since we are interested in data accesses, we choose the simplest non-trivial residual assembly operation:

$$I_1 = \int_\Omega f v \, \mathrm{d}x, \quad \forall v \in V \tag{10}$$

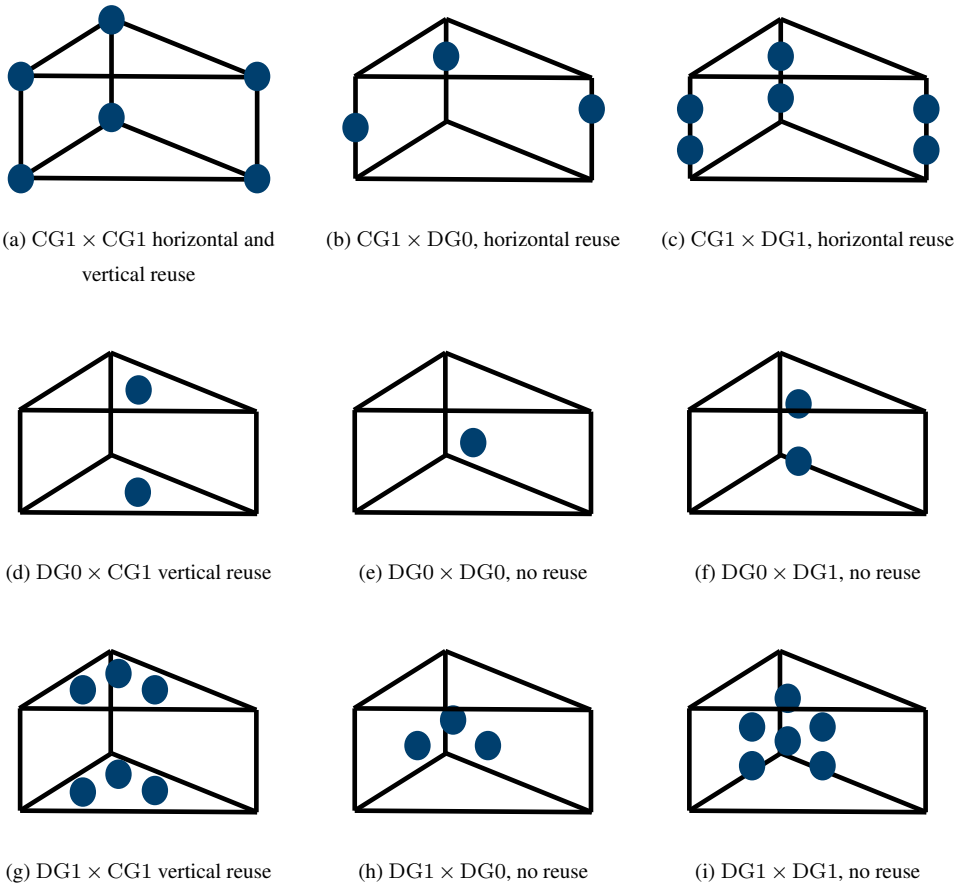
for f in the finite element space V. For this study we choose $\Omega = [0, 1]^3$ to be the unit cube. The base mesh is generated in an unstructured manner using Gmsh (Geuzaine and Remacle, 2009), and then extruded to form a three-dimensional domain.

In addition to the output field I_1 and the input field f this computation accesses the coordinate field, $\boldsymbol{x}$. Regardless of the choice of V, we always represent $\boldsymbol{x}$ by a d-vector at each vertex of the d-dimensional mesh.

4.1.2 Choosing the discretisations

The construction of a wide variety of finite element spaces on extruded meshes was introduced in McRae et al. (2016). This enables us to select the horizontal and vertical data discretisations independently.

For the purposes of data access, the distinguishing feature of different finite element spaces is the extent to which degrees of freedom are shared between adjacent cells.



(a) CG1 × CG1 horizontal and vertical reuse

(b) CG1 × DG0, horizontal reuse

(c) CG1 × DG1, horizontal reuse

(d) DG0 × CG1 vertical reuse

(e) DG0 × DG0, no reuse

(f) DG0 × DG1, no reuse

(g) DG1 × CG1 vertical reuse

(h) DG1 × DG0, no reuse

(i) DG1 × DG1, no reuse

Figure 5. Tensor product finite elements with different data layout and cell-to-cell data re-use.

We choose a set of finite element spaces spanning the combinations of horizontal and vertical reuse patterns found on extruded meshes: horizontal and vertical reuse, only horizontal, only vertical, or no reuse at all.

We employ low order continuous and discontinuous discretisations (abbreviated as *CG* and *DG* respectively) in both the horizontal and vertical directions.

The set of discretisations is $A = \{\mathrm{CG1}, \mathrm{DG0}, \mathrm{DG1}\}$ where the number indicates the degree of polynomials in the space. We examine all pairs of discretisations $(h, v) \in A \times A$. Since the cells of the base mesh are triangles, the extruded mesh consists of triangular prisms. Fig. 5 shows the data layout of each of these finite elements.

Both Firedrake and our numbering algorithm support a much larger range of finite element spaces than this. However, the more complex and higher degree spaces will result in more computationally intensive kernels but not materially different data reuse. The lowest order spaces are the most severe test of our approach since they are more likely to be memory bound.

4.1.3 Layer count and problem size

We vary the number of layers between 1 and 100. This is a realistic range for current ocean and atmosphere simulations. The number of cells in the extruded mesh is kept approximately constant by shrinking the base mesh as the number of layers increases. The mesh size is chosen such that the data volume far exceeds the total last level cache capacity of each chosen architecture (L3 cache in all cases). This minimizes caching benefits and is therefore the strongest test of our algorithms. The overall mesh size is fixed at approximately 15 million cells which yields a data volume of between 300 and 840 MB depending on discretisation.

4.1.4 Base mesh numbering

The order in which the entities of the unstructured mesh are numbered is known to be critical for data access performance. To characterize this effect and distinguish it from the impact of the number of layers, we employ two variants of each base mesh. The first is a mesh for which the traversal is optimised using a reverse Cuthill-McKee ordering (Lange et al., 2015). The second is a *badly* ordered mesh with a random numbering. This represents a pathological case for temporal locality.

4.2 Experimental Setup

The specification of the hardware used to conduct the experiments is shown in Table 2. Following Ofenbeck et al. (2014) we disable the Intel turbo boost and frequency scaling. This is intended to prevent our performance results from being subject to fluctuations due to processor temperature.

The experiments we are considering are run on a single two-socket machine and use MPI (Message Passing Interface) parallelism. The number of MPI processes varies from one up to 2 processes per physical core (exploiting hyperthreading). We pin the processes evenly across physical cores to ensure load balance and prevent process migration between cores.

The Firedrake platform performs integral computations by automatically generating *C* code. The compiler used is GCC version 4.9.1 (`-O3 -march=native -ffast-math -fassociative-math`). We also assessed the performance of the Intel C Compiler version 15.0.2 (`-O3 -xAVX -ip -xHost`), however we only report results from GCC in this paper since the performance of the Intel compiler was inferior.

4.2.1 Runtime, data volume, bandwidth and FLOPs

Runtime is measured using a nanosecond precision timer. Each experiment is performed ten times and we report the minimum runtime. Exclusive access to the hardware has been ensured for all experiments.

~~Different discretisations lead to different data volumes due to the way data is shared between cells. DG based discretisations require the movement of larger data volumes while CG discretisations lead to smaller volumes due to data reuse. To evaluate the impact of different data volumes we compare the valuable bandwidth with the achieved STREAM bandwidth.~~

We model the data transfer from main memory to CPU assuming a perfect cache: each piece of data is only loaded from main memory once. We define the *valuable data volume* as the total size of the input, output and coordinate fields. This gives

a lower bound on the memory traffic to and from main memory. The valuable data volume divided by the runtime yields the *valuable bandwidth*.

~~The~~ Different discretisations lead to different data volumes due to the way data is shared between cells. DG based discretisations require the movement of larger data volumes while CG discretisations lead to smaller volumes due to data reuse.

To evaluate the impact of different data volumes we compare the valuable bandwidth with the maximum bandwidth achieved for the STREAM triad benchmark (McCalpin, 1995)~~is~~, shown in Table 3. The valuable bandwidth achieved as percentage of STREAM bandwidth ~~achieved by the valuable bandwidth~~ shows how prone the code is to becoming bandwidth bound as ~~the~~ its floating point performance ~~of the payload~~ is improved.

The floating point operations – adds, multiplies and, on Haswell, fused multiply-add (FMA) operations – are counted automatically using the Intel Architecture Code Analyzer (Intel, 2012) whose results are verified with PAPI (Mucci et al., 1999) which accesses the hardware counters.

4.2.2 Theoretical performance bounds

The performance of the extruded iteration depends on the efficiency of the generated finite element kernel (payload) code which for some cases may not be ~~vectorized~~ vectorised (as outlined in Luporini et al. (2015)) or may not have a perfectly balanced number of floating point additions and multiplications. ~~Kernel~~ A discussion of kernel code optimality is outside the scope of this paper.

To a first approximation the performance of a numerical algorithm will be limited by either the memory bandwidth or the floating point throughput. The STREAM benchmark provides an effective upper bound on the achievable memory bandwidth. The floating point bounds employed are based on the theoretical maximum given the clock frequency of the processor.

The Intel architectures considered are capable of executing both a floating point addition and a floating point multiplication on each clock cycle. The Haswell processor can execute a fused multiply-add instruction (FMA) instead of either an addition or multiplication operation.

The achievable FLOP rate may therefore be as much as twice the clock rate depending on the mix of instructions executed. The achievable speed-up over the clock rate, f_b, for the Sandy Bridge platform is therefore bounded by the balance factor

$$f_b = 1 + \frac{\min(\text{add FLOPs}, \text{multiplication FLOPs})}{\max(\text{add FLOPs}, \text{multiplication FLOPs})}, \tag{11}$$

while for Haswell it is bounded by

$$f_b = 1 + \frac{\min(\text{add FLOPs}, \text{multiplication FLOPs}) + k}{\max(\text{add FLOPs}, \text{multiplication FLOPs}) + k}, \tag{12}$$

where k is half the number of FMAs.

4.2.3 Vectorization

The processors employed support 256-bit wide vector floating point instructions. The double precision FLOP rate of a fully ~~vectorized~~ vectorised code can be as much as four times ~~the clock rate~~that of an unvectorised code. GCC automatically

~~vectorized~~ vectorised only a part of the total number of floating point instructions. The ratio between the number of vector (packed) floating point instructions and the total number of floating point instructions (scalar and packed) characterizes the impact of partial vectorization on the floating point bound through the vectorization factor

$$f_v = 1 + (4-1) \times \frac{\text{vector FLOPs}}{\text{total FLOPs}}. \tag{13}$$

To control the impact of the kernel computation (payload) on the evaluation, we compare the measured floating point throughput with a theoretical peak which incorporates the payload instruction balance and the degree of vectorization. Let c be the number of active ~~CPU physical~~ physical CPU cores during the computation of interest. The ~~base (theoretical)~~ theoretical base floating point performance B_c is the same for all discretisations and assumes one floating point instruction per cycle for each ~~(active)~~ active physical CPU core. The peak theoretical floating point throughput P_d is different for each discretisation d as it depends on the properties of the payload and is given by

$$P_d = B_c \times f_b \times f_v. \tag{14}$$

4.3 Experimental Results

4.3.1 Percentage of theoretical performance

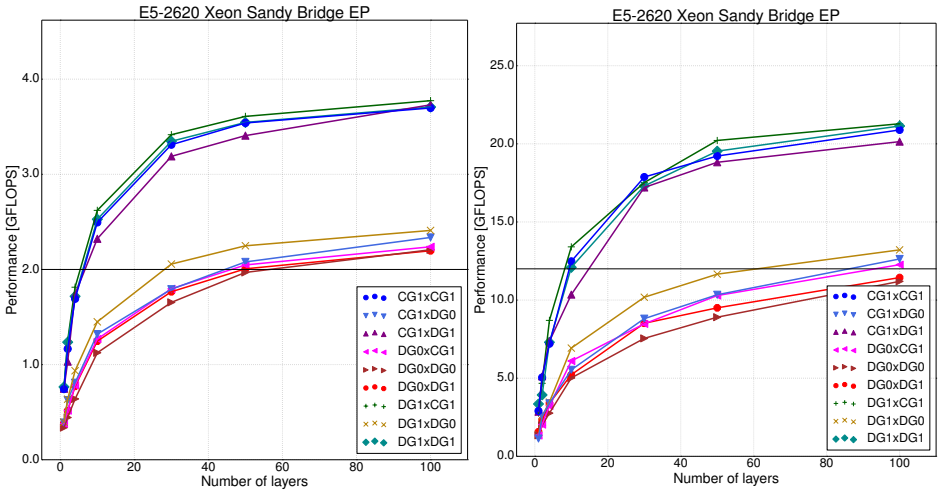
For the Sandy Bridge and Haswell architectures, the best performance is achieved in the 100-layer case run with 24 and 32 processes respectively (hyperthreading enabled). The results in Tables 4 and 5 show percentages of the STREAM bandwidth and the theoretical floating point throughput which incorporates the instruction balance and vectorization factors.

On Sandy Bridge, the proportion of peak theoretical floating point throughput is between 71 and 85%, while on Haswell it is between 71 and 92%. In contrast, the proportion of peak bandwidth achieved varies between 7 and 51% on Sandy Bridge and 9 and 75% on Haswell. The higher, and much more consistent peak FLOP results lead us to the conclusion that we are in an operation- rather than bandwidth-limited regime. The performance figures are therefore presented with respect to this metric.

4.3.2 Amortizing the cost of indirect accesses

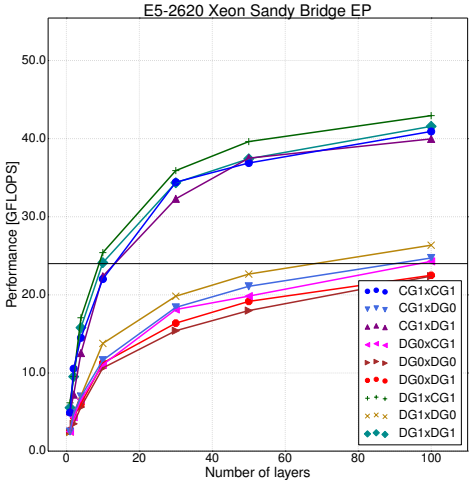
When the base mesh is well ordered (Fig. 7), the number of layers required to reach a performance plateau is between 10 and 20 for all discretisations. When the base mesh is badly ordered (Fig. 6) the ~~required number of layers can be as large as fifty or more. For example, in the case of discretisations employing DG0 either horizontally or vertically, the FLOP rate plateau is~~ plateau is frequently not reached even ~~at a hundred layers.~~ with 100 layers. A striking feature of Figs. 6 and 7 is that cases in which the local kernel calculations are identical produce very similar achieved FLOP rates, despite having different data sharing patterns. This supports the hypothesis that the results are operation bound.

4.3.3 ~~Percentage of theoretical performance~~

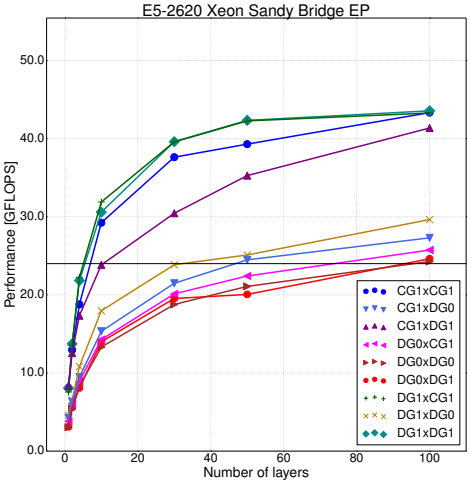


(a) Sandy Bridge, 1 process, $c = 1$

(b) Sandy Bridge, 6 processes, $c = 6$

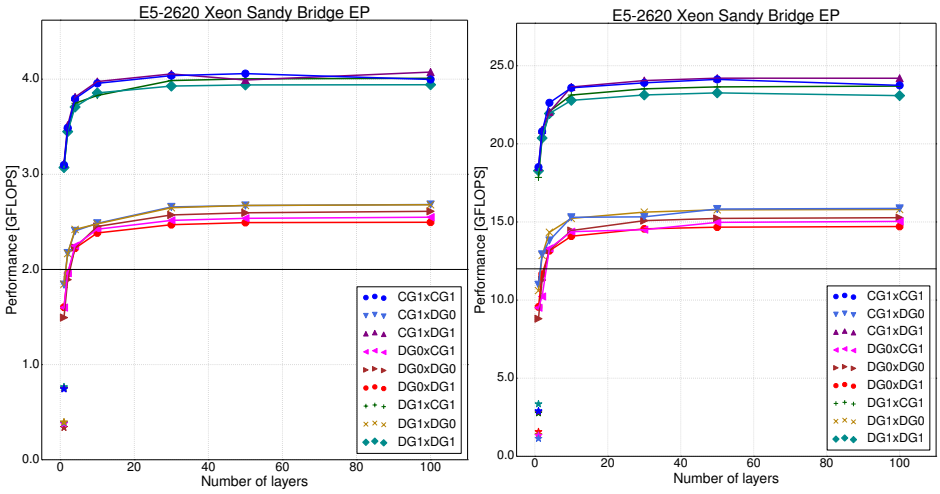


(c) Sandy Bridge, 12 processes, $c = 12$



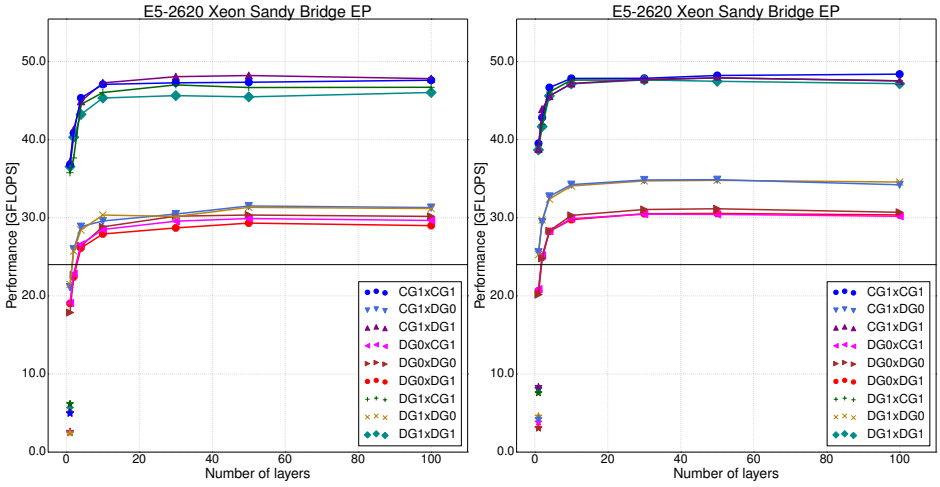
(d) Sandy Bridge, 24 processes, $c = 12$

Figure 6. Performance of the I integral computation with varying number of layers and number of processes on a badly-ordered base mesh. The horizontal line is the base FLOP throughput for $f_b = f_v = 1$ and the number of physical cores used.



(a) Sandy Bridge, 1 process, $c = 1$

(b) Sandy Bridge, 6 processes , $c = 6$



(c) Sandy Bridge, 12 processes, $c = 12$

(d) Sandy Bridge, 24 processes, $c = 12$

Figure 7. Performance of the I integral computation with varying number of layers and number of processes on a well-ordered base mesh. The star-shaped markers show the performance of the 1-layer badly-ordered mesh for comparison. The horizontal line is the base FLOP throughput for $f_b = f_v = 1$ and the number of physical cores used.

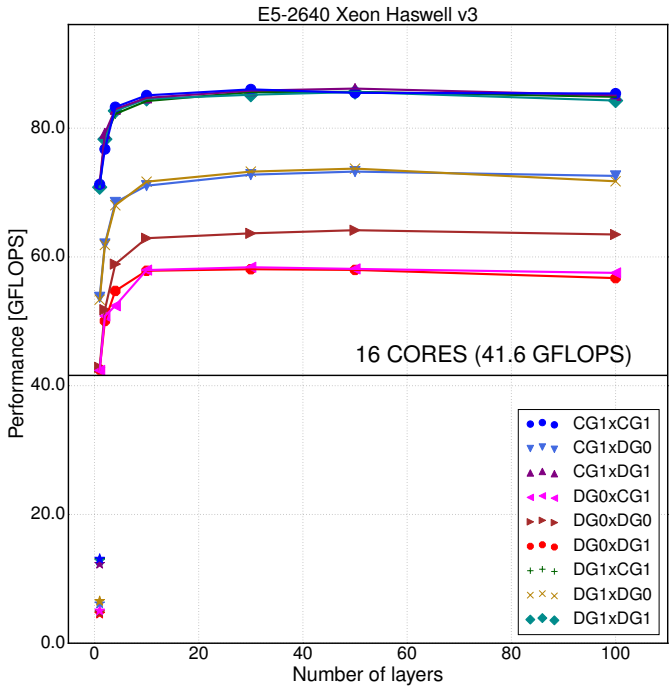


Figure 8. Performance of the I integral computations on different data discretisations with varying number of layers on the Haswell architecture for a well-ordered base mesh. The star-shaped markers show the performance of the 1-layer badly-ordered mesh for comparison. The horizontal line is the base FLOP throughput for $f_b = f_v = 1$ and the number of physical cores used.

~~For the Sandy Bridge and Haswell architectures, the best performance is achieved in the 100-layer case run with 24 and 32 processes respectively (hyperthreading enabled). The results in Tables 4 and 5 show percentages of the STREAM bandwidth and the theoretical floating point throughput which incorporates the instruction balance and vectorization factors.~~

4.4 Discussion

The performance of the extruded mesh iteration is constrained by the properties of the mesh and the kernel computation. The total number of computations is based on the number of degrees of freedom per cell. The range of discretisations used in this paper (Fig. 5) leads to four cases: one, two, three or six degrees of freedom per cell. In compute bound situations, discretisations with the same number of computations have the same performance (Fig. 8).

4.4.1 Temporal locality

The numbering algorithm ensures good temporal locality between vertically aligned cells. Any degrees of freedom which are shared vertically are reused when the iteration algorithm visits the next element. The reuse distance along the vertical is therefore minimal.

For CG discretisations, where degrees of freedom are shared horizontally with other vertical columns, the overall performance depends on the ordering of cells in the base mesh. Assuming a perfect ordering of the base mesh, the numbering algorithm ensures a minimal reuse distance while guaranteeing a minimum number of indirect accesses and satisfying all the previously introduced spatial and temporal locality requirements.

Figures 7 and 6 demonstrate the combined impact of horizontal mesh ordering and extrusion. In the extreme case the flop rate increases up to 14 times between the badly ordered single-layer case and the 100 layer well ordered case. This is consistent with the widely held belief that unstructured mesh models are an order of magnitude slower than structured mesh models.

The difference between well- and badly-ordered mesh performance outlines the benefits responsible for the boost in performance. Horizontal data reuse dominates performance for low number of layers while spatial locality and vertical temporal locality (ensured by the numbering and iteration algorithms) are responsible for most of the performance gains as the number of layers increases.

We note, once again, that these results are for the lowest order spaces which represent a worst case. Higher-order methods both access more contiguous data in each column and require many more FLOPs. As a result, we would expect to reach performance plateaus at lower numbers of layers.

5 Conclusions

In this paper we have presented efficient, locality-aware algorithms for numbering and iterating over extruded meshes. For a sufficient number of layers, the cost of using an unstructured base mesh is amortized. Achieved performance ranges from 70% to 90% of our best estimate for the hardware's performance capabilities and current level of kernel optimisation. Benefits of

spatial and temporal locality vary with number of layers: as the number of layers is increased the benefits of spatial locality increase while those of temporal locality decrease.

This paper employed two simplifying constraints: that there are a constant number of layers in each column, and that the number of degrees of freedom associated with each entity type is a constant. These assumptions are not fundamental to the numbering algorithm presented here, or to its performance. We intend to relax those constraints as they become important for the use cases for which Firedrake is employed.

The current code generation scheme can be extended to include inter-kernel vectorization (an optimisation mentioned in Meister and Bader (2015)) for the operations which cannot be ~~vectorized~~ vectorised at intra-kernel level. The efficiency of such a generic scheme applicable to different data discretisations is currently being explored.

In future work we intend to generalize some of the optimisations which extrusion enables for both residual and Jacobian assembly: inter-kernel optimisations, grouping of addition of contributions to the global system and exploiting the vertical alignment at the level of the sparse representation of the global system matrix. In addition to the CPU results presented in this paper, we also plan to explore the performance portability issues of extruded meshes on Graphical Processing Units and Intel Xeon Phi accelerators.

6 Team list

The Firedrake project contributors: Gheorghe-Teodor Bercea, George Boutsioukis, Colin J. Cotter, Patrick E. Farrell, Simon W. Funke, David A. Ham, Miklós Homolya, Christian Jacobs, Anna Kalogirou, Paul H.J. Kelly, Michael Lange, Nicolas Loriant, Fabio Luporini, Graham R. Markall, Andrew T.T. McRae, Lawrence Mitchell, Eike H. Müller, Florian Rathgeber, Asbjørn Nilsen Riseth, Francis P. Russell, Kaho Sato.

The Software Performance Optimisation (SPO) group members: Gheorghe-Teodor Bercea, David A. Ham, Miklós Homolya, Paul H.J. Kelly, Michael Lange, Fabio Luporini, Lawrence Mitchell, Luigi Nardi, Emanuele Rossini.

7 Code availability

The packages used to perform the experiments have been archived using Zenodo: Firedrake (Mitchell et al., 2016), PETSc (Smith et al., 2016), petsc4py (Dalcin et al., 2016), FIAT (Rognes et al., 2016), UFL (Alnæs et al., 2016), FFC (Logg et al., 2016), PyOP2 (Rathgeber et al., 2016) and COFFEE (Luporini et al., 2016). The source code repositories as well as the archived versions are publicly available.

8 Data availability

The scripts used to perform the experiments as well as the results are archived using Zenodo: Sandy Bridge ~~(Bercea, 2016d) and Haswell (Bercea, 2016c). The~~ (Bercea, 2016c) and Haswell (Bercea, 2016b). The meshes used in the experiments are available also (Bercea, 2016a). The archives are publicly available.

Author contributions. Gheorghe-Teodor Bercea designed the generalized extrusion algorithm, performed the extension of the Firedrake and PyOP2 packages to support extruded meshes, the performance evaluation and the preparation of the graphs and tables. Andrew T. T. McRae extended components of the Firedrake toolchain to support the finite element types used in the experiments, and made minor contributions to the extruded mesh iteration functionality. David A. Ham was the proponent of a generalized extrusion algorithm. Lawrence Mitchell, Florian Rathgeber and Fabio Luporini developed related features and framework improvements in Firedrake, PyOP2 and COFFEE. Luigi Nardi is responsible for the use of the floating point balance metric. David A. Ham and Paul H. J. Kelly are the principal investigators for this paper. Gheorghe-Teodor Bercea prepared the manuscript with contributions from all the authors. All authors contributed with feedback during the paper's write-up process.

Acknowledgements. This work was supported by an Engineering and Physical Sciences Research Council prize studentship [Ref. 1252364], the Grantham Institute and Climate-KIC, the Natural Environment Research Council [grant numbers NE/K006789/1, NE/K008951/1, and NE/M013480/1] and the Department of Computing, Imperial College London. The authors would like to thank J. (Ram) Ramanujam at Louisiana State University for the insightful discussions and feedback during the writing of this paper. We are thankful to Francis Russell at Imperial College London for the feedback on this paper.

References

Alnæs, M. S., Logg, A., Wells, G., Mitchell, L., Rognes, M. E., Homolya, M., Bergersen, A., Ring, J., Ham, D. A., chrisrichardson, Mardal, K.-A., Blechta, J., Rathgeber, F., Markall, G., Li, L., Cotter, C. J., McRae, A. T. T., mliertzer, maxalbert, Airaksinen, T., and Hake, J.: UFL: The Unified Form Language, doi:10.5281/zenodo.47713, 2016.

Bercea, G.-T.: Unstructured meshes for extrusion article, doi:10.5281/zenodo.61819, http://dx.doi.org/10.5281/zenodo.61819, 2016a.

Bercea, G.-T.: Data and plot scripts for Haswell experiments, doi:10.5281/zenodo.61919, http://dx.doi.org/10.5281/zenodo.61919, 2016b.

Bercea, G.-T.: Data and plot scripts for Sandy Bridge experiments Data and plot scripts for Sandy Bridge experiments, doi:10.5281/zenodo.61920, http://dx.doi.org/10.5281/zenodo.61920, 2016c.

Bercea, G.-T.: Extrusion Performance: Sandy Bridge Performance, doi:10.5281/zenodo.48638, http://dx.doi.org/10.5281/zenodo.48638, 2016d.

Bercea, G.-T.: Extrusion Performance: Haswell Performance, doi:10.5281/zenodo.48642, http://dx.doi.org/10.5281/zenodo.48642, 2016e.

Dalcin, L., Mitchell, L., Brown, J., Farrell, P. E., Lange, M., Smith, B., Karpeyev, D., nocollier, Knepley, M., Ham, D. A., Funke, S. W., Ahmadia, A., Hisch, T., Homolya, M., Alastuey, J. C., Riseth, A. N., Wells, G., and Guyer, J.: Petsc4py: The Python interface to PETSc, doi:10.5281/zenodo.47714, 2016.

Ford, R., Glover, M., Ham, D., Maynard, C., Pickles, S., and Riley, G.: Gung Ho phase 1 computational science recommendations, Tech. rep., The Met Office, 2013.

Gersbacher, C.: The Dune-PrismGrid Module, in: Advances in DUNE: Proceedings of the DUNE User Meeting, Held in October 6th–8th 2010 in Stuttgart, Germany, edited by Dedner, A., Flemisch, B., and Klöfkorn, R., pp. 33–44, Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-28589-9_3, 2012.

Geuzaine, C. and Remacle, J.-F.: Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, International Journal for Numerical Methods in Engineering, 2009.

Günther, F., Mehl, M., Pögl, M., and Zenger, C.: A Cache-Aware Algorithm for PDEs on Hierarchical Data Structures Based on Space-Filling Curves, SIAM J. Sci. Comput., 28, 1634–1650, doi:10.1137/040604078, http://dx.doi.org/10.1137/040604078, 2006.

Intel: Intel Architecture Code Analyzer, https://software.intel.com/en-us/articles/intel-architecture-code-analyzer, 2012.

Isaac, T.: Scalable, adaptive methods for forward and inverse problems in continental-scale ice sheet modeling, Ph.D. thesis, University of Texas at Austin, 2015.

Isaac, T., Stadler, G., and Ghattas, O.: Solution of Nonlinear Stokes Equations Discretized By High-Order Finite Elements on Nonconforming and Anisotropic Meshes, with Application to Ice Sheet Dynamics, SIAM Journal on Scientific Computing, 37, B804–B833, doi:10.1137/140974407, 2015.

Knepley, M. G. and Karpeev, D. A.: Mesh Algorithms for PDE with Sieve I: Mesh Distribution, Scientific Programming, 17, 215–230, doi:10.3233/SPR-2009-0249, 2009.

Koziel, S., Leifsson, L., Lees, M., V.Krzhizhanovskaya, V., Dongarra, J., Sloot, P. M., Sarje, A., Song, S., Jacobsen, D., Huck, K., Hollingsworth, J., Malony, A., Williams, S., and Oliker, L.: International Conference On Computational Science, ICCS 2015 Parallel Performance Optimizations on Unstructured Mesh-based Simulations, Procedia Computer Science, 51, 2016 – 2025, doi:http://dx.doi.org/10.1016/j.procs.2015.05.466, http://www.sciencedirect.com/science/article/pii/S1877050915012740, 2015.

Lange, M., Mitchell, L., Knepley, M., and Gorman, G.: Efficient mesh management in Firedrake using PETSc-DMPlex, To appear in SIAM Journal on Scientific Computing, 2015.

Logg, A.: Efficient Representation of Computational Meshes, International Journal of Computational Science and Engineering, 4, 283–295, doi:10.1504/IJCSE.2009.029164, 2009.

Logg, A., Alnæs, M. S., Rognes, M. E., Wells, G., Ring, J., Mitchell, L., Hake, J., Homolya, M., Rathgeber, F., Luporini, F., Markall, G., Bergersen, A., Li, L., Ham, D. A., Mardal, K.-A., Blechta, J., Bercea, G.-T., Airaksinen, T., Schlömer, N., Langtangen, H. P., Cotter, C. J., Skavhaug, O., Hisch, T., mliertzer, Haga, J. B., and McRae, A. T. T.: FFC: FEniCS Form Compiler, doi:10.5281/zenodo.47761, 2016.

Luporini, F., Varbanescu, A. L., Rathgeber, F., Bercea, G.-T., Ramanujam, J., Ham, D. A., and Kelly, P. H. J.: Cross-Loop Optimization of Arithmetic Intensity for Finite Element Local Assembly, ACM Trans. Archit. Code Optim., 11, 57:1–57:25, doi:10.1145/2687415, 2015.

Luporini, F., Mitchell, L., Homolya, M., Rathgeber, F., Ham, D. A., Lange, M., Markall, G., and Russell, F.: COFFEE: A Compiler for Fast Expression Evaluation, doi:10.5281/zenodo.47715, 2016.

Macdonald, A. E., Middlecoff, J., Henderson, T., and Lee, J.-L.: A General Method for Modeling on Irregular Grids, Int. J. High Perform. Comput. Appl., 25, 392–403, doi:10.1177/1094342010385019, 2011.

McCalpin, J. D.: Memory Bandwidth and Machine Balance in Current High Performance Computers, IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pp. 19–25, 1995.

McRae, A. T. T., Bercea, G.-T., Mitchell, L., Ham, D. A., and Cotter, C. J.: Automated generation and symbolic manipulation of tensor product finite elements, To appear in SIAM Journal on Scientific Computing, 2016.

Meister, O. and Bader, M.: 2D adaptivity for 3D problems: Parallel SPE10 reservoir simulation on dynamically adaptive prism grids, Computational Science at the Gates of Nature, 9, 101–106, doi:10.1016/j.jocs.2015.04.016, 2015.

Mitchell, L., Ham, D. A., Rathgeber, F., Homolya, M., McRae, A. T. T., Bercea, G.-T., Lange, M., Cotter, C. J., Jacobs, C. T., Luporini, F., Funke, S. W., Büsing, H., Kärnä, T., Kalogirou, A., Rittich, H., Mueller, E. H., Kramer, S., Riseth, A. N., and Markall, G.: Firedrake: an automated finite element system, doi:10.5281/zenodo.47717, 2016.

Mucci, P. J., Browne, S., Deane, C., and Ho, G.: PAPI: A Portable Interface to Hardware Performance Counters, in: In Proceedings of the Department of Defense HPCMP Users Group Conference, pp. 7–10, 1999.

Ofenbeck, G., Steinmann, R., Caparros, V., Spampinato, D. G., and Püschel, M.: Applying the roofline model, in: 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 76–85, doi:10.1109/ISPASS.2014.6844463, 2014.

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., McRae, A. T. T., Bercea, G.-T., Markall, G. R., and Kelly, P. H. J.: Firedrake: automating the finite element method by composing abstractions, Submitted to ACM TOMS, 2015.

Rathgeber, F., Mitchell, L., Luporini, F., Markall, G., Ham, D. A., Bercea, G.-T., Homolya, M., McRae, A. T. T., Dearman, H., Jacobs, C. T., gbts, Funke, S. W., Sato, K., and Russell, F.: PyOP2: Framework for performance-portable parallel computations on unstructured meshes, doi:10.5281/zenodo.47712, 2016.

Rognes, M. E., Logg, A., Homolya, M., Ham, D. A., Schlömer, N., Blechta, J., McRae, A. T. T., Bergersen, A., Cotter, C. J., Ring, J., Mitchell, L., Wells, G., Kirby, R., Rathgeber, F., Li, L., Alnæs, M. S., and mliertzer: FIAT: The Finite Element Automated Tabulator, doi:10.5281/zenodo.47716, 2016.

Skamarock, W. C., Klemp, J. B., Duda, M. G., Fowler, L. D., Park, S.-H., and Ringler, T. D.: A multiscale nonhydrostatic atmospheric model using centroidal Voronoi tesselations and C-grid staggering, Monthly Weather Review, 140, 3090–3105, doi:10.1175/MWR-D-11-00215.1, 2012.

Slingo, J., Bates, K., Nikiforakis, N., Piggott, M., Roberts, M., Shaffrey, L., Stevens, I., Vidale, P. L., and Weller, H.: Developing the next-generation climate system models: challenges and achievements, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 367, 815–831, doi:10.1098/rsta.2008.0207, 2009.

Smith, B., Balay, S., Knepley, M., Brown, J., McInnes, L. C., Zhang, H., Brune, P., sarich, stefanozampini, Karpeyev, D., Dalcin, L., markadams, Minden, V., VictorEijkhout, vijaysm, tisaac, and Rupp, K.: Petsc: Portable, Extensible Toolkit for Scientific Computation, doi:10.5281/zenodo.47718, 2016.

Yoon, S.-E., Lindstrom, P., Pascucci, V., and Manocha, D.: Cache-oblivious Mesh Layouts, ACM Trans. Graph., 24, 886–893, doi:10.1145/1073204.1073278, http://doi.acm.org/10.1145/1073204.1073278, 2005.

Zängl, G., Reinert, D., Rípodas, P., and Baldauf, M.: The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core, Quarterly Journal of the Royal Meteorological Society, 141, 563–579, doi:10.1002/qj.2378, 2015.

Table 1. Topological dimensions of extruded mesh entities. D_b denotes the topological dimension of the base mesh.

Mesh entity	*Dimensions*
Vertex	$(0, 0)$
Vertical Edge	$(0, 1)$
Horizontal Edge	$(1, 0)$
Vertical Facet	$(D_b - 1, 1)$
Horizontal Facet	$(D_b, 0)$
Cell	$(D_b, 1)$

Table 2. Hardware used.

Name	Intel Sandy Bridge	Intel Haswell
Model	Xeon E5-2620	Xeon E5-2640 v3
Frequency	2.0 GHz	2.6 GHz
Sockets	2	2
Cores per socket	6	8
Bandwidth per socket	42.6 GB/s	56.0 GB/s

Table 3. Maximum STREAM triad ($a_i = b_i + \alpha c_i$) performance achieved by varying the number of MPI processes from one to twice the number of physical cores.

Platform	STREAM bandwidth
Intel Sandy Bridge	$55.3\ GB/s$
Intel Haswell	$80.2\ GB/s$

Table 4. Percentage of STREAM bandwidth and theoretical throughput achieved by the computation of integral I over 100 layers on Sandy Bridge with 24 MPI processes.

Discretisation	f_b	f_v	P_d (%)	Bandwidth (%)
CG1 $\times$ CG1	1.7	1.58	73.45	7.092
CG1 $\times$ DG0	1.81	1.0	78.96	14.70
CG1 $\times$ DG1	1.7	1.58	73.03	10.50
DG0 $\times$ CG1	1.65	1.0	76.01	27.86
DG0 $\times$ DG0	1.5	1.0	85.14	34.86
DG0 $\times$ DG1	1.65	1.0	75.45	45.68
DG1 $\times$ CG1	1.7	1.58	73.20	24.60
DG1 $\times$ DG0	1.81	1.0	78.93	50.98
DG1 $\times$ DG1	1.7	1.58	71.78	44.37

Table 5. Percentage of STREAM bandwidth and theoretical throughput achieved by the computation of integral I over 100 layers on Haswell with 32 MPI processes.

Discretisation	f_b	f_v	P_d (%)	Bandwidth (%)
CG1 $\times$ CG1	1.76	1.61	72.43	9.015
CG1 $\times$ DG0	1.97	1.0	88.57	21.92
CG1 $\times$ DG1	1.76	1.61	72.20	13.39
DG0 $\times$ CG1	1.87	1.0	73.94	38.74
DG0 $\times$ DG0	1.66	1.0	91.93	53.10
DG0 $\times$ DG1	1.87	1.0	72.89	63.11
DG1 $\times$ CG1	1.76	1.61	71.99	31.19
DG1 $\times$ DG0	1.97	1.0	87.55	75.17
DG1 $\times$ DG1	1.76	1.61	71.50	56.98