Mesh Coarsening for Fast Simulation on Low Resource Machine

Gan Luo and Zhiguo Zhang*

School of Data and Computer Science, Sun Yat-Sen University, Guangzhou 510006, China

*Corresponding author

Abstract. Mesh-based application uses a mesh to discretely represent a real physical system and then calls appropriate algorithm over the mesh to solve equation for simulation. Mesh-based application is usually developed by domain programmers and it runs with very large mesh on super computer. It takes a long time to run with such a large mesh on low resource machine. In this paper, we propose an algorithm designed for coarsening two-dimensional quadrilateral mesh aiming to reduce the size of the mesh so that the application could run with the smaller coarse mesh on low resource machine in a short time. This makes it more convenient for domain programmers to develop and debug the application on low resource machine. The algorithm first generates cells of the coarse mesh by agglomerating cells of the original mesh. Then it constructs internal edges, boundary edges, nodes and all mappings of the coarse mesh. At last, the algorithm constructs datasets of the coarse mesh according to the datasets of the original mesh. The experimental result shows that it takes much less time to run application over the coarse mesh produced by our algorithm, and the error of fast simulation results is within the acceptable range.

Keywords: Mesh-based application; mesh coarsening; fast simulation; quadrilateral mesh.

1. Introduction

Mesh-based application applies a mesh to discretely represent a real physical system and calls appropriate algorithm over the mesh to solve the equation for simulation. Mesh-based applications are usually developed by domain programmers and run on super computer since the meshes used would be extremely large. It takes a long time to run such an application on low resource machine, which is inconvenient for domain programmers to develop and debug the application. To solve the problem, a mesh coarsening algorithm is needed. The algorithm should construct a small coarse mesh based on the large original mesh. Existing coarse mesh construction algorithms operate on the dual graph of the mesh and obtain the cells of the coarse mesh by using a variety of agglomerating techniques [1]. Unfortunately, these algorithms fail to construct internal edges, boundary edges, nodes, all mappings and datasets of the coarse mesh. We propose an algorithm designed for coarsening large two-dimensional quadrilateral meshes. The algorithm first generates cells of the coarse mesh by agglomerating cells of the original mesh. Then it constructs internal edges, boundary edges, nodes and all mappings of the coarse mesh. At last, the algorithm constructs datasets of the coarse mesh according to the datasets of the original mesh. Thus our algorithm can construct an integral coarse mesh, which can be an input mesh of the application.
2. Related Works
In the field of mesh coarsening, a large amount of work has been done to construct coarse mesh cells. The basic idea behind the existing algorithms is to start from a particular vertex of the dual graph and fuse together some of its adjacent vertices into a new coarse mesh cell. There are many ways to select vertices to be fused together. Guillard [1] have proposed to select vertices based on connectivity of the dual graph, while Mavriplis [2] proposed that it can be done so that the aspect ratio of the coarse cell is locally optimized. Besides, some computationally efficient algorithms for solving certain optimization problem are developed. These algorithms are based on multilevel paradigm that has been found to be quite effective in solving the related problem of graph and mesh partitioning. Moulitsas et al. [3] proposed a robust multilevel algorithm for generating coarse mesh from the original mesh. The algorithm is implemented in routines of MGridGen and ParMGridGen library.

However, the coarse mesh cells constructed by the algorithms mentioned above are likely to be adjacent to more than four cells while there is a requirement that the resulting coarse mesh should be a quadrilateral mesh as well. Moreover, those algorithms only construct cells of the coarse mesh, without constructing internal edges, boundary edges, nodes, mappings and datasets of the coarse mesh. It is because the goal of those algorithm are different from ours. For example, the goal of the multilevel algorithm [3] is to build coarse meshes for geometric multigrid solvers while our algorithm aims to reduce the size of the original mesh so that the mesh-based application could run with the coarse mesh on low resource machine in a short time. The input mesh of an application should be integral, which means that all sets, mappings and datasets necessary for running application over the mesh. Unfortunately, there are few algorithms designed for accomplishing that. Therefore our work mainly focuses on constructing those components of the coarse mesh.

3. Mesh Coarsening Algorithm
In this section we describe the main steps of the mesh coarsening algorithm. An integral mesh is composed of sets, mappings and datasets. Sets represent basic component elements of a mesh. There are four sets in a two-dimensional mesh, which are cells, nodes, internal edges and boundary edges, denoted by $S_c, S_n, S_e, S_b$ respectively. Each element of a certain set has a unique index ranging from 0 to $n - 1$, where $n$ is the number of elements of the set. Mappings are used to specify topology of the mesh. A mapping maps each element of a certain set to a fixed number of elements of another set if the ‘from set’ element is connected to those ‘to set’ elements physically. The fixed number is the dimension of the mapping. There are five mappings in a two-dimensional mesh: ‘cells to nodes’, ‘internal edges to nodes’, ‘internal edges to cells’, ‘boundary edges to nodes’, ‘boundary edges to cells’, denoted by $M_{c\rightarrow n}, M_{e\rightarrow n}, M_{e\rightarrow c}, M_{b\rightarrow n}, M_{b\rightarrow c}$ respectively. Dataset is defined on set according to the practical needs and has physical meaning. If a dataset of n-dimension is defined on a set, then there is data of n-dimension defined on each element of the set.

The goal of our algorithm is to construct all sets $S_c', S_n', S_e', S_b'$, mappings $M_{c\rightarrow n}', M_{e\rightarrow n}', M_{e\rightarrow c}', M_{b\rightarrow n}', M_{b\rightarrow c}'$ and datasets of the coarse mesh using sets $S_c, S_n, S_e, S_b$, mappings $M_{c\rightarrow n}, M_{e\rightarrow n}, M_{e\rightarrow c}, M_{b\rightarrow n}, M_{b\rightarrow c}$ and datasets of the original mesh. The algorithm first constructs coarse mesh cells $S_c'$ by agglomerating original mesh cells, ensuring that each coarse mesh cell is adjacent to no more than four cells. Then the algorithm constructs internal edges $S_e'$, boundary edges $S_b'$, nodes $S_n'$ and all mapping $M_{c\rightarrow n}', M_{e\rightarrow n}', M_{e\rightarrow c}', M_{b\rightarrow n}', M_{b\rightarrow c}'$ of the coarse mesh. At last, based on the results of the previous steps, the algorithm constructs datasets of the coarse mesh according to the datasets of the original mesh.

3.1. Construction of Coarse Mesh Cells
We first construct cells of the coarse mesh by agglomerating multiple original mesh cells into a coarse mesh cell. Each constructed coarse mesh cell should be adjacent to no more than four cells since the coarse mesh is also a quadrilateral mesh. Suppose we are clear about the structure of the original mesh, then we can directly determine which original mesh cells are agglomerated into a coarse mesh cell, ensuring that the requirement is satisfied. We use an array named part to give the corresponding
relationship between the original mesh cells and the coarse mesh cells. The size of the array part is equal to the number of cells in the original mesh. Specifically, part[i] stores the index of a coarse mesh cell that the original mesh cell i belongs to after coarsening.

```
|    | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|----|---|---|---|---|---|---|---|
| 0  |  6|  1| 10|  4|  5|  6|  7|
| 1  |  8|  9| 10| 11| 12| 13| 14|
```

Figure 1. An example process of constructing coarse mesh cells.

Figure 1 shows an example process of constructing coarse mesh cells. In this example, the original mesh cells 0, 1, 4, 5 are agglomerated into coarse mesh cell 0, so the values of part[0], part[1], part[4], part[5] are all 0.

3.2. Construction of Internal Edges, Boundary Edges, Nodes and All Mappings

In this step, we first construct the corresponding dual graph of the coarse mesh. In the dual graph, a vertex represents a cell of the coarse mesh. Two vertices of the dual graph are adjacent to each other and connected by an edge $E'$ if the corresponding two cells share an internal edge $e'$ in the mesh. Therefore each internal edge of the mesh corresponds to an edge of the dual graph. There are $n$ vertices in the dual graph, where $n$ is the number of cells in the coarse mesh. Traverse the mapping $M_{e'\rightarrow e}$ of the original mesh. For each internal edge $e$, it is mapped to two original mesh cells $c_1$ and $c_2$. Check the array part to see which coarse mesh cells they belong to. If they belong to two different coarse mesh cells $c_1'$ and $c_2'$, the corresponding two vertices $v_1'$ and $v_2'$ are adjacent to each other. Create an edge $E'$ connecting the two vertices in the dual graph if it is not created before.

Based on the dual graph, we first construct the set of internal edges $S_e'$ and the mapping $M_{e'\rightarrow e}$ of the coarse mesh. Traverse the edge set of the dual graph. For each edge $E'$, create the corresponding internal edge $e'$ of the coarse mesh and numbering it. The edge $E'$ connects two vertices $v_1'$ and $v_2'$, which represent the two coarse mesh cells $c_1'$ and $c_2'$ respectively. Add the mapping information $e'\rightarrow c_1'$ and $e'\rightarrow c_2'$ (we use the notation $\rightarrow$ to denote mapping relationship, similarly hereinafter) to $M_{e'\rightarrow e}$.

Next we construct the set of boundary edges $S_b'$ and the mapping $M_{b'\rightarrow b}$ of the coarse mesh. In dual graph of the coarse mesh, a vertex is adjacent to four vertices at most. Suppose that a vertex is adjacent to $n$ vertices, where $n$ is not more than four. The corresponding cell is adjacent to $n$ cells, so there are $n$ internal edges and $4-n$ boundary edges connected to the cell. Traverse the vertex set of the dual graph. For each vertex $v'$, determine the number $n$ of its adjacent vertices. Suppose that $v'$ represents the coarse mesh cell $c'$. Create $4-n$ boundary edges, numbering them and add the mapping information $b_1'\rightarrow c'$, ..., $b_{4-n}'\rightarrow c'$ to $M_{b'\rightarrow b}$. Note that if a vertex has four neighboring vertices, then no boundary edge is created.

![Diagram](image1.png)

Figure 2. An example process of constructing dual graph and coarse mesh internal edges and boundary edges.

The next step is constructing the set of nodes $S_n'$ and the mappings $M_{c'\rightarrow c}$, $M_{e'\rightarrow e}$, $M_{b'\rightarrow b}$ of the coarse mesh. We classify nodes into three types based on the connection relationship between nodes and elements of other sets.
(I) The first type of node is connected to only one cell and multiple boundary edges, but it is not connected to any internal edge. Traverse the vertex set of the dual graph. For each vertex \( v' \), determine the number \( n \) of its adjacent vertices. Suppose that \( v' \) represents the coarse mesh cell \( c' \). If \( n \) is 1, then \( c' \) is adjacent to one cell and connected to three boundary edges and two nodes of type I. Check the mapping \( M_{b \rightarrow c} \) constructed before to determine the three boundary edges \( b_1', b_2', \) and \( b_3' \) which are mapped to \( c' \). Create two nodes \( n_1' \) and \( n_2' \) and numbering them. Add the mapping information \( 'c' \rightarrow n_1' \) and \( 'c' \rightarrow n_2' \) to \( M_{c \rightarrow n} \), \('b_1' \rightarrow n_1', 'b_2' \rightarrow n_1', 'b_3' \rightarrow n_2' \), and \('b_3' \rightarrow n_3' \) to \( M_{b \rightarrow n} \). If \( n \) is 2, then \( c' \) is adjacent to two cells and connected to two boundary edges and one node of type I. Check the mapping \( M_{b \rightarrow c} \) to determine the two boundary edges \( b_1' \) and \( b_2' \) which are mapped to \( c' \). Create a node \( n' \) and numbering it. Add the mapping information \( 'c' \rightarrow n_1' \) to \( M_{c \rightarrow n} \), \('b_1' \rightarrow n_1' \) and \('b_2' \rightarrow n_1' \) to \( M_{b \rightarrow n} \).

(ii) The second type of node is connected to three boundary edges, internal edges and multiple cells. In order to construct coarse mesh nodes of type II, we have to find the corresponding nodes of type II in the original mesh. In the original mesh, a node of type II is connected to two boundary edges which are mapped to two cells. If the two cells belong to two different coarse mesh cells, then the original mesh node is the target node. Traverse the boundary edges set \( S_b \) of the original mesh. For each boundary edge \( b \) , it is mapped to two nodes \( n_1 \) and \( n_2 \) by \( M_{b \rightarrow n} \) and is mapped to a cell \( c \) by \( M_{b \rightarrow c} \). Check the array \( \text{part} \) to determine the coarse mesh cell \( c' \) that \( c \) belongs to. Build relationships between \( n_1 \) and \( c_1' \), \( n_2 \) and \( c_2' \). When the loop is over, all boundary nodes of the original mesh are related to two coarse mesh cells while all internal nodes are not related to any coarse mesh cell. Traverse the nodes set \( S_e \) of the original mesh. For each node \( n \), if it is related to two different coarse mesh cells, then it is a target node. Create a node \( n' \) of the coarse mesh and numbering it. Check the mapping \( M_{b \rightarrow c} \) to determine which coarse mesh boundary edges are mapped to the two cells. For each of the two cells, if the cell is connected to one boundary edge \( b' \), then add the mapping information \( 'b' \rightarrow n_1' \) to \( M_{b \rightarrow n} \). If the cell is connected to multiple boundary edges, then choose a boundary edge \( b' \) whose mapping table \( M_{b \rightarrow n} \) is not full yet and add the mapping information \( 'b' \rightarrow n_1' \) to \( M_{b \rightarrow n} \). Then check the mapping \( M_{e \rightarrow n} \) to determine which internal edges are mapped to \( n \). For each of these internal edges, it is mapped to two cells by \( M_{e \rightarrow c} \). If the two cells belong to different coarse mesh cells \( c_1' \) and \( c_2' \), which means that they are connected to \( n_1' \) in the coarse mesh, then record the connection information. The mapping \( M_{e \rightarrow c} \) to determine the internal edge \( e' \) which is mapped to \( c_1' \) and \( c_2' \). The coarse mesh internal edge \( e' \) is connected to \( n_1' \) as well and record the connection information. When the loop is over, all cells and internal edges connected to \( n_1' \) in the coarse mesh have been found. Suppose there are \( m \) cells \( c_1', ..., c_m' \) connected to \( n_1' \), then there must be \( m - 1 \) internal edges \( e_1', ..., e_{m-1}' \) connected to \( n_1' \). Add the mapping information \( 'c_1' \rightarrow n_1', 'c_{m}' \rightarrow n_1' \) to \( M_{e \rightarrow n} \), and \('e_1' \rightarrow n_1', 'e_{m-1}' \rightarrow n_1' \) to \( M_{e \rightarrow n} \).

(iii) The third type of node is connected to internal edges and cells, but it is not connected to any boundary edge. In order to construct coarse mesh nodes of type III, we also have to find the corresponding nodes in the original mesh. In the original mesh, a node of type III \( n \) is not connected to any boundary edge, but it’s connected to more than two cells. Each of those cells belongs to a coarse mesh cell \( c' \) and we say that \( n \) is related to \( c' \). If \( n \) is related to more than two different coarse mesh cells, then it is the target node. Based on the idea, we traverse \( S_n \) of the original mesh. For each cell \( c \) , it is mapped to four nodes \( n_1, n_2, n_3 \) and \( n_4 \) by \( M_{c \rightarrow n} \). Check the array \( \text{part} \) to determine the coarse mesh cell \( c' \) that \( c \) belongs to. Build relationships between \( n_1 \) and \( c_1' \), \( n_2 \) and \( c_2' \), \( n_3 \) and \( c_3' \), \( n_4 \) and \( c_4' \). When the loop is over, all nodes of the original mesh are related to one or multiple coarse mesh cells. Traverse the set of nodes \( S_n \) of the original mesh. For each node \( n \), if it is related to more than two different coarse mesh cells and is not connected to any
boundary edge, then it is a target node. Create a node \( n' \) of the coarse mesh and numbering it.

Suppose \( n \) is related to \( m \) coarse mesh cells \( c'_1, \ldots, c'_m \), where \( m \) is larger than 2. Then \( n' \) is connected to the \( m \) coarse mesh cells. Add the mapping information \( c'_1 \mapsto n', \ldots, c'_m \mapsto n' \) to \( M_{c \to n} \). Then we check the mapping \( M_{c \to c} \) to determine several coarse mesh internal edges each of which is mapped to \( c'_1 \) and \( c'_j \), where \( c'_1, c'_j \in \{ c'_1, \ldots, c'_m \} \). Note that the number of such internal edges must be \( m \) as well. The \( m \) internal edges \( e'_1, \ldots, e'_m \) are connected to \( n' \) in the coarse mesh. Add the mapping information \( e'_1 \mapsto n', \ldots, e'_m \mapsto n' \) to \( M_{e \to n} \).

\[ \text{Figure 3. An example of construction result of the three types of nodes.} \]

We have constructed the three types of nodes and their related mappings so far. Combine the results and we get \( S_n, M_{c \to n}, M_{e \to n} \) and \( M_{b \to n} \) of the coarse mesh.

3.3. Generation of Coarse Mesh Datasets

In this step, we generate datasets of the coarse mesh. For each element of the coarse mesh, we first find a group of corresponding elements of the original mesh, and then process data defined on those original mesh elements to get the data defined on the coarse mesh element. For a coarse mesh cell \( c' \), the original mesh cells that belong to \( c' \) are its corresponding cells. For a coarse mesh internal edge \( e' \), it is mapped to two cells \( c'_1 \) and \( c'_j \) by \( M_{c \to c} \). If an original mesh internal edge \( e \) is mapped to two cells that belong to \( c'_1 \) and \( c'_j \) respectively, then \( e \) is a corresponding edge of \( e' \). For a coarse mesh boundary edge \( b' \), it is mapped to a cell \( c' \) by \( M_{b \to c} \). If an original mesh boundary edge \( b \) is mapped to a cell that belongs to \( c' \), then \( b \) is a corresponding boundary edge of \( b' \). For a coarse mesh node \( n' \) of type I, it is connected to only one cell \( c' \). If an original mesh node of type I is connected to a cell that belongs to \( c' \), then it is a corresponding node of \( n' \). If there are multiple original mesh nodes satisfying the requirement, then choose one of them as the corresponding node. For a coarse mesh node of type II or III, its corresponding original mesh node is the target node found in the previous step. Now each coarse mesh element corresponds to one or multiple original mesh elements. The data defined on each coarse mesh element is generated by processing data defined on the corresponding original mesh elements. The way of processing each dataset is defined by developers, which can be summation, averaging or so on. Till now we have constructed all sets \( S_n, S_c, S_e \) and \( S_b \), mappings \( M_{c \to n}, M_{e \to n}, M_{b \to n} \) and \( M_{c \to c} \) and datasets of the coarse mesh and the algorithm is over.

4. Experiments

Our experiment focused on evaluating the time reduction of running application over the constructed coarse mesh and the quality of fast simulation result. The mesh-based application used in the experiment is Airfoil [4], which is an industrial representative CFD application benchmark, written using OP2’s API [5]. The original mesh is an example mesh of Airfoil.
Table 1. Number of elements in each set of the original mesh and the three coarse meshes.

| Mesh/Set  | Node     | Cell     | Internal Edge | Boundary Edge |
|-----------|----------|----------|---------------|---------------|
| Original  | 721801   | 720000   | 1438600       | 2800          |
| C2        | 180901   | 180000   | 359300        | 1400          |
| C4        | 45451    | 45000    | 89650         | 700           |
| C8        | 11476    | 11250    | 22325         | 350           |

We called the algorithm to construct three coarse meshes of different size. For each coarse mesh Cn, the algorithm constructs each cell by agglomerating exactly n × n original mesh cells, where n is a fixed number. The three coarse meshes constructed in the experiment are denoted by C2, C4, C8 respectively. From Table 1, the size of each of the three coarse meshes is reduced to different extent compared with the original mesh.

There are multiple versions of Airfoil. We ran each version of Airfoil over the original mesh and the three coarse meshes and recorded the time consumed respectively. From Table 2 we can see that for each version of Airfoil, the time consumed by running it over the three coarse meshes is greatly reduced compared with that over the original mesh.

Table 2. Time consumed by running each version of Airfoil over each mesh (in seconds).

| Mesh/Version | Airfoil_Seq | Airfoil_MPI | Airfoil_OpenMP | Airfoil_MPI_OpenMP |
|--------------|-------------|-------------|----------------|--------------------|
| Original Mesh| 142.224927  | 138.432006  | 90.309896      | 81.306900          |
| C2           | 34.455395   | 34.215024   | 19.540705      | 16.965625          |
| C4           | 8.496579    | 8.479568    | 1.399623       | 1.457946           |
| C8           | 2.126921    | 2.104961    | 0.569514       | 0.517687           |

Next we evaluated the quality of result produced by running the sequential version of Airfoil over the three coarse meshes. The simulation result is a four-dimensional dataset named q defined on the set of cells of the input mesh. We define q_o and q_cn as the resulting datasets produced by running Airfoil over the original mesh and the coarse mesh Cn respectively. We also define q_o(c, n) as the nth dimension of the data defined on the original mesh cell c and define q_cn(c', n) as the nth dimension of the data defined on the coarse mesh cell c'. Our goal is to get resulting data defined on each original mesh cell by conducting fast simulation over the coarse mesh, but the resulting dataset q_cn is defined on the set of cells of the coarse mesh. Therefore we should generate a dataset named q_co defined on the set of cells of the original mesh based on q_cn. Suppose the original mesh cell c belongs to the coarse mesh cell c' after coarsening, then we let q_co(c; n) = q_cn(c', n). The quality of result produced by running Airfoil over a coarse mesh is evaluated by error between q_co(c; n) and q_o(c, n) for all dimensions and all cells.

Table 3. Proportion of data in each error range of each dimension of each resulting dataset(%).

| Dataset/Dim/Error Range | 0% - 1% | 1% - 10% | 10% - 100% | > 100% |
|-------------------------|--------|----------|------------|-------|
| q_c2                    | 0      | 93.9161  | 6.0839     | 0     |
|                         | 1      | 85.2461  | 14.7464    | 0.0072| 0.0003|
|                         | 2      | 92.1456  | 2.8292     | 4.6497| 0.3756|
|                         | 3      | 92.4667  | 7.5333     | 0     | 0     |
| q_c4                    | 0      | 96.7942  | 3.2058     | 0     | 0     |
|                         | 1      | 89.6264  | 10.3553    | 0.0164| 0.0019|
|                         | 2      | 93.9294  | 1.6386     | 4.4294| 0.0025|
|                         | 3      | 96.0061  | 3.9939     | 0     | 0     |
| q_c8                    | 0      | 96.6781  | 3.3219     | 0     | 0     |
|                         | 1      | 89.8181  | 10.1328    | 0.0442| 0.0050|
|                         | 2      | 93.8475  | 1.4167     | 4.7225| 0.0133|
|                         | 3      | 95.7297  | 4.2703     | 0     | 0     |
We first ran the sequential version of Airfoil over the original mesh to get the dataset \( q_o \). For each coarse mesh \( Cn \), we ran the same version of Airfoil over it to get the dataset \( q_cn \) and generated \( q_co \) based on \( q_cn \). Then we calculated error between \( q_co(c, n) \) and \( q_o(c, n) \) for all dimensions and all cells. For each dimension \( d \), we calculated the proportion of data in each error range. The result is shown in Table 3. From Table 3, we can see that for each dimension of each resulting dataset \( q_cn \), there is over 85% of data whose error is less than 1%. Therefore, we can draw a conclusion that the result of running Airfoil over the coarse mesh produced by our algorithm is of good quality.

### 5. Conclusion

Our algorithm can construct coarse mesh of correct structure and complete information, which can be used as input mesh of mesh-based application directly. The size of the coarse mesh reduces greatly and the time consumed by running application over the coarse mesh greatly reduces, which is convenient for domain programmers to develop and debug the mesh-based application on low resource machine. Moreover, the error of fast simulation result is small and within acceptance range. Specifically, it takes over 75% less time to get the resulting data while the error of more than 85% resulting data is less than 1%. Our main future work will be to parallelize the algorithm and improve the algorithm to coarsen more general mesh.

### Acknowledgement

This research was financially supported by the National Key R&D Program of China (2017YFB0202400, 2017YFB0202403).

### References

[1] H. Guillard. Node–nested multigrid with delaunay coarsening. Technical Report No 1898, INRIA–Sophia Antipolis, France, 1993.

[2] V. Venkatakrishnan and D.J. Mavriplis. Agglomeration multigrid for the three–dimensional euler equations. Technical Report 94-5, Institute for Computer Applications in Science and Engineering NASA Langley Research Center, 1994.

[3] I. Moulitsas and G. Karypis. Multilevel algorithms for generating coarse grids for multigrid methods. In Proceedings of the 2001 ACM/IEEE conference on Supercomputing.

[4] M. Giles, G. Mudalige and I. Reguly. OP2 Airfoil Example. Available on WWW at URL https://op-dsl.github.io/docs/OP2/airfoil-doc.pdf. 2013.

[5] G.R. Mudalige, I. Reguly, M.B. Giles, C. Bertolli and P.H.J. Kelly. OP2: An Active Library Framework for Solving Unstructured Mesh-based Applications on Multi-Core and Many-Core Architectures. In Proceedings of Innovative Parallel Computing (InPar), 2012, pp.1-12, 13-14 May 2012. (https://doi.org/10.1109/InPar.2012.6339594)