Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

Load balanced 2D and 3D adaptive mesh refinement in OpenFOAM

Daniel Rettenmaier [a,c,*], Daniel Deising [d,f], Yun Ouedraogo [e], Erion Gjonaj [e], Herbert De Gersem [e,b], Dieter Bothe [d], Cameron Tropea [c], Holger Marschall [d,b,*]

[a] *Graduate School Computational Engineering, Technical University Darmstadt, Dolivostr. 15, 64293 Darmstadt, Germany*
[b] *Centre for Computational Engineering, Technical University Darmstadt, Dolivostr. 15, 64293 Darmstadt, Germany*
[c] *Institute of Fluid Mechanics and Aerodynamics, Technical University Darmstadt, Alarich-Weiss-Str. 10, 64287 Darmstadt, Germany*
[d] *Institute for Mathematical Modeling and Analysis, Technical University Darmstadt, Alarich-Weiss-Str. 10, 64287 Darmstadt, Germany*
[e] *Institute for Theory of Electromagnetic Fields, Technical University Darmstadt, Schloßgartenstr. 8, 64289 Darmstadt, Germany*
[f] *ENGYS UK, Studio 20, Royal Victoria Patriotic Building, John Archer Way, London SW18 3SX, United Kingdom of Great Britain and Northern Ireland*

ARTICLE INFO

Article history:
Received 13 August 2018
Received in revised form 29 July 2019
Accepted 29 August 2019

Keywords:
Dynamic load balancing
Adaptive mesh refinement
2D
3D
OpenFOAM

ABSTRACT

An object-oriented approach to load-balanced adaptive mesh refinement in two and three dimensions is presented for OpenFOAM, a mature C++ library for computational fluid dynamics. Such a high-performance computing technique is mandatory for computational efficiency in cases of moving regions of interest for distributed-memory parallel computer architectures where domain decomposition is applied. Moving regions of interest are dynamically deforming and migrating through the domain during simulations and require high spatial resolution of solution features. We present software design and code structure, and detail on our achievements with respect to software usability for engineering applications, as well as on numerous developments and improvements necessary for high stability and runtime performance.

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_143
Legal Code License	GNU General Public License (GPL)
Code versioning system used	git
Software code languages, tools, and services used	OpenFOAM-dev https://github.com/OpenFOAM/OpenFOAM-dev
Compilation requirements, operating environments & dependencies	Compilation within OpenFOAM-dev on Linux
If available Link to developer documentation/manual	Documentation included within the computable document
Support email for questions	rettenmaier@gsc.tu-darmstadt.de, ouedraogo@temf.tu-darmstadt.de, marschall@mma.tu-darmstadt.de

1. Introduction

The parallel computational simulation of transient transport processes in science and engineering frequently poses a major challenge, that is to use computational resources efficiently while at the same time maintaining high accuracy of the numerical solution throughout the computation. Transport processes often involve moving regions of interest, which migrate through the domain and deform dynamically over time. There is a demand to fully resolve fine transient solution features within these regions in order to decrease numerical errors. Examples in engineering applications are flame fronts, detonation shocks or fluid interfaces [1].

Nowadays, Computational Fluid Dynamics (CFD) software typically relies on distributed-memory parallel computer architectures for all medium and large scale computations. In modern CFD simulation software, the commonly underlying Finite Volume Method (FVM) is required to support dynamic unstructured computational meshes of general topology to cope with

* Corresponding authors.
E-mail addresses: danielrettenmaier@gmail.com (D. Rettenmaier), marschall@tu-darmstadt.de (H. Marschall).

https://doi.org/10.1016/j.softx.2019.100317

complex solution domains of varying shape. Under these demands it has become established practice to utilize a collocated (pseudo-staggered) variable arrangement on the computational mesh, where the values of transport variables are stored in cell centers (volume fields), while the cell-face centers hold information on the fluxes and interpolated cell values (surface fields). Parallelization is accomplished almost exclusively in domain decomposition mode, i.e. a large loop over cell-faces of the mesh is split up and processed by many processor cores, which are referred to as processors in the following.

Several commercial and academic CFD solver libraries already feature Adaptive Mesh Refinement (AMR) [2–6] as well as Dynamic Load Balancing (DLB) [7–9]. To the list of mentioned DLB implementations, this work is of importance as it is the first stable implementation for OpenFOAM. Simulations using the AMR and DLB framework made available here have already been employed in [10] and [11].

The implementation of AMR and DLB depends on the underlying data structure and parallelization strategy. OpenFOAM (Open Field Operation and Manipulation) is a comprehensive open source C++ library for computational continuum mechanics, including CFD [2,12–14]. It exhibits a rigorous and efficient object-oriented approach to domain decomposition parallelism. It distinguishes itself from other open source CFD software by a modular code structure, where modules (linear solvers, interpolation schemes, physical models, etc.) are implemented following the Strategy Design Pattern. Maintainability is achieved by generic programming: heavy use of templates enables operations on different data types using the same methods and without code duplication. Along with operator overloading, this allows to devise tailored top-level CFD code, which closely mimic the mathematical language in continuum modeling, i.e. partial differential equations. Further flexibility is achieved using the Runtime Type Selection (RTS) mechanism: following the Factory Design Pattern, OpenFOAM allows instantiation of an object in the class hierarchy at runtime of the software on the user side, by changing entries in a configuration file (dictionary). Such strict distinction between interface and library use is held up throughout the library in order to hide implementation details for the sake of usability and readability on top-level.

As for parallelism, communication details are isolated from library use at top-level by a software interface layer which works as parallel communication wrapper with a standard interface so as to allow every top-level code to be written without any specific parallelization requirements. This way, the same lines of code operate in serial and parallel execution. Inter-processor communication is established as a boundary condition, meaning each cell is uniquely allocated to separate processors in a zero-halo-layer approach, without duplicating cell data next to processor boundaries.

Many developments in OpenFOAM are community-driven which greatly contributed to the success of the framework. The work exemplifies such an effort by revising and building upon community contributions on two-dimensional and axisymmetric cases [15] and on DLB [16]. Enhancements to AMR and DLB are consolidated here into one unified framework, made available to the community. The benefit is an effectively reduced computational effort on small and medium sized engineering problems, solvable on modern desktop computers, where AMR and DLB are as well imperative. Since the here presented AMR and DLB framework benefits from many OpenFOAM libraries, the parallel performance of the AMR and DLB algorithms cannot be tested independently of OpenFOAM. Thus, exhaustive parallel efficiency and scaling tests are out-of-scope in this work, particularly on many-core architectures for larger scales. Recently, however, promising results have been shown by Phuc et al. using several billion cells on a statically refined mesh [17].

To effectively tackle the above challenges we use and significantly enhance the OpenFOAM C++ library. This contribution bases on the following objectives:

- to set out the main ingredients of AMR and DLB in the object-oriented fabric of OpenFOAM software, and
- to provide an implementation to the community readily usable for a wide range of available solvers in recent OpenFOAM releases (version 4.x and later).

2. Adaptive mesh refinement

To dynamically obtain high accuracy only where necessary, the h-adaptivity approach is used in AMR. Thus far, OpenFOAM has only supported AMR for hexahedral cells in 3D. Recent work to extend AMR for arbitrary polyhedral cells has been presented in [18,19], implemented only in *foam-extend-4.1*, which has yet to find its way into OpenFOAM, and in [20], whose code is currently not freely available. The polyhedral cell refinement decomposes cells into tetrahedra, which introduces parasitic currents in the invariant mode of 2D and 2.5D cases. Therefore, a true 2D and 2.5D refinement is still useful in many use-cases.

2.1. Octree cell refinement

In 3D cases, hexahedral cells are split using an octree structure where a so-called refinement level is associated with each cell. A cell is refined by cutting it into eight child cells with 36 faces of which 12 are internal to the parent cell (see Fig. 1(a)). Neighbors to the refined cells will change from hexahedral to polyhedral cells with more than six faces. The pseudo-staggered approach in OpenFOAM relies on two locations for defining mesh related fields: volume fields represent values on cell centers and surface fields on face centers. The handling of surface fields in combination with AMR suffers from three major problems in the current OpenFOAM versions: the lack of consistent interpolation between values of refined and unrefined cells, wrong face addressing of newly created faces during refinement, and sign-flipping of non-flux surface fields. Solutions for this problems are outlined in the following.

Mapping is the interpolation of fields between parent and refined child cells. Mapping of cell-centered volume fields is implemented in a straightforward and conservative manner. In a cell refinement step, child cells receive the cell-centered value of the parent cell; in a cell unrefinement step, the volume average of child cells' cell-centered values is set on the parent cell-center. However, mapping of surface fields during the refinement step is more demanding and critical since most solvers in OpenFOAM are flux based and fluxes are represented as surface fields. Values at refined faces are set with the value of their corresponding parent face, called master face, which is the face of the parent cell that is split into four new faces. New faces internal to the parent cells are however not related to any master face. A better value interpolation onto those faces is realized in this work by arithmetically averaging the value of the four adjacent master faces, as shown for one internal face in Fig. 1(a). The mapping of surface fields onto new internal faces has been implemented in the `dynamicRefineFvMesh` class (see Fig. 2).

An implementation error dating back to 1.x OpenFOAM versions in the mesh cutter class `hexRef8` at the initialization of new faces has been corrected, which invalidated the addressing of newly created internal faces, resulting in seemingly random values on new faces in the whole simulation domain.

The sign of fluxes depends on the face normal direction that is shared by up to two adjacent cells. This direction may change due to topological changes during refinement or unrefinement depending on a formal ownership. Thus a sign-flip of the flux

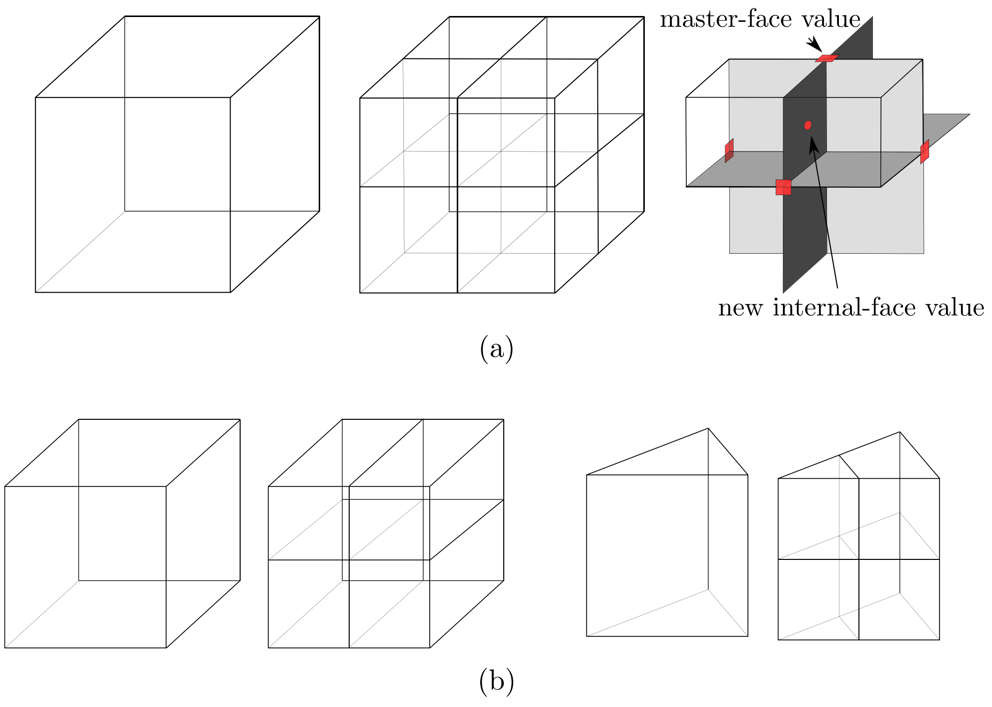

Fig. 1. (a) Octree refinement of a hexahedral cell creates 12 internal faces. Values of surface fields (round symbol) are interpolated from the values of the adjacent master faces (squared symbols) which is exemplarily shown for one internal face. (b) Quadtree refinement of hexahedral cells and prism cells. The invariant direction is normal to the page.

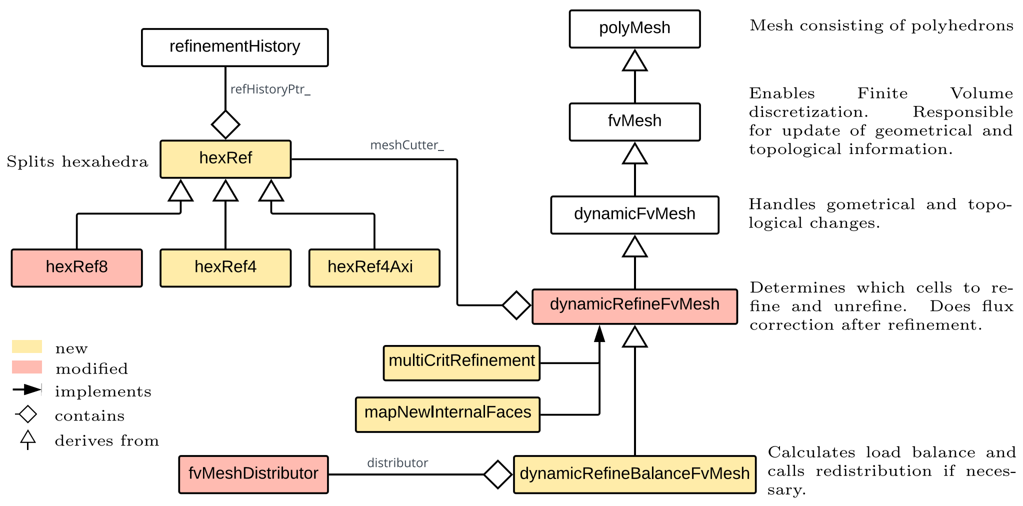
refinementHistory
refHistoryPtr_
Splits hexahedra
hexRef
meshCutter_
hexRef8
hexRef4
hexRef4Axi
polyMesh
Mesh consisting of polyhedrons
fvMesh
Enables Finite Volume discretization. Responsible for update of geometrical and topological information.
dynamicFvMesh
Handles geometrical and topological changes.
dynamicRefineFvMesh
Determines which cells to refine and unrefine. Does flux correction after refinement.
multiCritRefinement
mapNewInternalFaces
new
modified
implements
contains
derives from
fvMeshDistributor
distributor
dynamicRefineBalanceFvMesh
Calculates load balance and calls redistribution if necessary.

Fig. 2. UML diagram showing the newly implemented classes for dynamic load balancing and 2D as well as 2.5D mesh refinement in yellow and already existing classes with necessary changes in red. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

values becomes necessary. Sign-flipping is done for all types of surface fields, even though they may not represent fluxes. Wrong sign-flipping has been corrected in numerous places within the OpenFOAM library.

Incompressible flow solvers in OpenFOAM rely on conservative flux fields, corresponding to a divergence free velocity field. Due to the nonconservative mapping of surface fields, the flux field needs to be corrected after each AMR step (see Appendix A). Using the mentioned correction of face addressing, the enhanced surface field mapping and the correction in sign-flipping (see commit 1cef6f9) significantly reduces the necessary correction to ensure a divergence free field, and improves runtime performance, accuracy and stability [10].

The relation between child and parent cells is maintained in form of a `refinementHistory`, which also stores the refinement level for each cell. Cells are only unrefined if the criteria for unrefinement is matched in all sibling cells, which is taken care of in the `dynamicRefineFvMesh` class. Otherwise, the level of cells at the border of the refined volume may switch back and forth between two levels at each refinement and unrefinement step, unnecessarily consuming computational resources.

2.2. Quadtree refinement

For 2D cases, OpenFOAM uses 3D meshes, with the additional constraint of a single cell layer in the invariant direction. At runtime, the boundary type `empty` is automatically detected as an indication of a 2D case in order to reduce the number of dimensions in which to solve. Octree refinement breaks the invariance condition since it introduces cells in the invariant

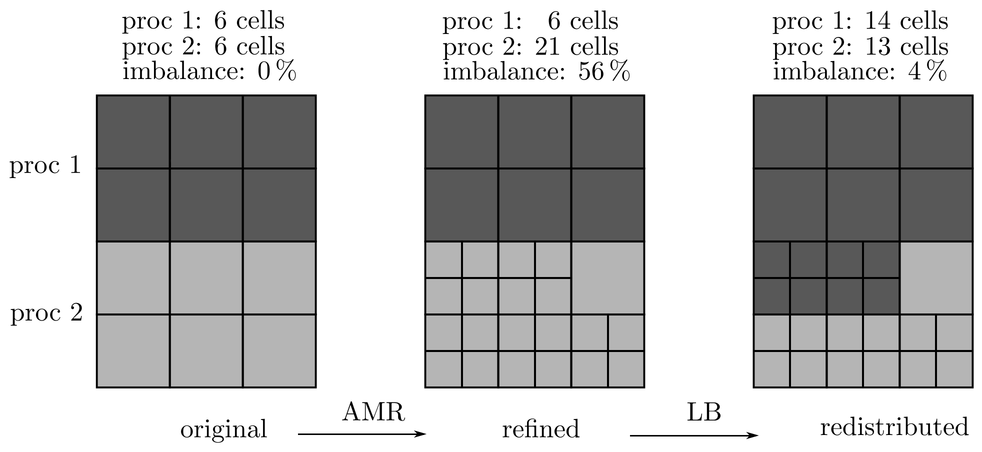

Fig. 3. The load balancing redistribution after mesh refinement.

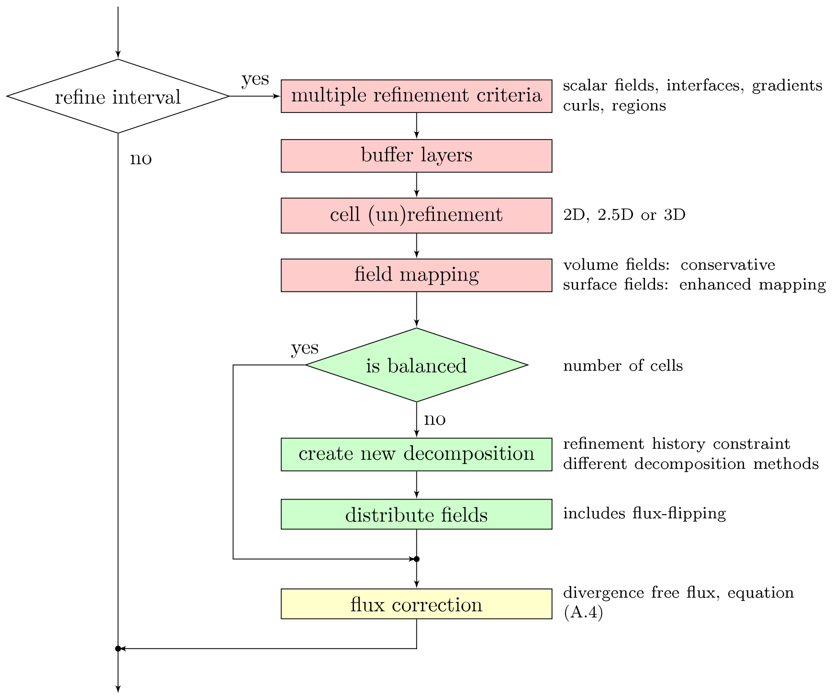
refine interval
yes
no
multiple refinement criteria
scalar fields, interfaces, gradients curls, regions
buffer layers
cell (un)refinement
2D, 2.5D or 3D
field mapping
volume fields: conservative
surface fields: enhanced mapping
is balanced
yes
no
number of cells
create new decomposition
refinement history constraint
different decomposition methods
distribute fields
includes flux-flipping
flux correction
divergence free flux, equation (A.4)

Fig. 4. Adaptive mesh refinement, dynamic load balancing and flux correction if current simulation time matches with the refine interval.

direction. For two-dimensional cases, the standard octree refinement must therefore be replaced by a quadtree refinement. The implementation of quadtree refinement is based on the octree refinement, with the difference that internal cells are not split to introduce new cells in the invariant direction. The two faces on the `empty` boundaries are split at their center into four new faces each. This step introduces new points at the middle of all boundary edges, used to split faces in the acceptable direction. Cell refinement therefore produces four child cells and introduces four internal faces in the parent cell (see Fig. 1(b)).

For axisymmetric cases, the implementation uses another type of boundary, the `wedge` type. The mesh still has a thickness of one cell but takes the form of a wedge with one edge along the axis of symmetry. Quadtree refinement can be applied as described above for hexahedral cells. However, special care needs to be taken in the refinement of prism cells at the axis. Triangular faces of those cells are detected at refinement to be split into triangular and quadrangular faces, so that prism cells are refined into two prisms and two hexahedra (see Fig. 1(b)).

The extension of the 3D mesh cutter to 2D and axisymmetric cases is based on work by [15], which is further abstracted and refined in this work. In order to limit code duplication, the `hexRef8` class is replaced with a base `hexRef` class with three derived classes, `hexRef8`, `hexRef4` and `hexRef4Axi`, which implement octree refinement, quadtree refinement and quadtree refinement including prisms on the axis, respectively (see Fig. 2). The `hexRef` classes make use of the runtime selection mechanism so that the `dynamicRefineFvMesh` selects the appropriate *n*-tree refinement based on the number of dimensions of the mesh and the number of dimensions of the solution. The number of dimensions is also used at runtime to determine whether prism cell refinement is allowed.

2.3. Combining multiple refinement criteria

For greater flexibility, OpenFOAM's single criterion refinement was extended to multiple different criteria. The criteria include uniquely selectable fields, their gradients and curls, interfaces, geometrical features such as boxes or domain boundaries as well as the maximum and minimal refinement levels on each individual criterion. All settings are modifiable at runtime and the use exemplarily demonstrated in tutorials of the repository. Fig. 2 shows the multi-criterion refinement class as a modular plug-in to `dynamicRefineFvMesh`.

2.4. Buffer layers

Buffer layers are the number of cell layers between two refinement levels. In meshes with more than two refinement levels it is important to ensure a smooth transition between the different levels in order to sufficiently decrease discretization errors due to mesh skewness at refinement transitions and to provide a buffer between two refinement levels for the computed flow to adapt to the new mesh level. The one-irregularity constraint restricts the refinement level difference to neighboring cells to one, so the minimum number of buffer layers is one. However, more buffer layers are recommended. The implementation of this feature is described in Appendix B.

3. Dynamic load balancing

In OpenFOAM, parallelism is based on domain decomposition. The simulation domain is decomposed into sub-domains, each being assigned to a different processor. Whenever AMR is used on a transient problem in parallel, the changing load on processors may significantly reduce the efficient use of computational resources. The processor with the largest load becomes a bottleneck when processed data is required by other waiting processors. Several methods to estimate the load balance can be thought of. Here, the load balance is calculated as the maximum difference between the number of cells on a sub-domain to the average number of cells of all sub-domains. This difference is divided by this same average. This approach assumes for each cell the same computational cost which is not generally true. An example of load imbalance is given in Fig. 3. After a refinement of a set of cells, the domain distribution is imbalanced by 56%, clearly a significant amount. After redistribution, a load imbalance of only 4% is achieved.

Several DLB algorithms based on OpenFOAM libraries are already in use, such as the one found in [21] for version 2.1.x or another in [22] based on the solution by Voskuilen [16] for version 2.3.x, which also serves as a base for this work. The algorithms of all these authors rely on the redistribution functionality of OpenFOAM, which exchanges cells and their field values between processors given an old and a new domain decomposition. This algorithm needs only minor changes as outlined by Voskuilen to enable redistribution of fields with physical dimensions and we further add to that list with changes regarding surface field flipping and we implement additional functionality to allow for the combination with 2D refinement.

Apart from Voskuilen's list, a problem was found in `fvMeshDistribute` related to the flux-flipping issue explained in Section 2.1, where non-flux surface fields are flipped while redistributing the cells on different processors. Wrongly flipped fields cause solver crashes. Furthermore, the default mapping of values on boundaries during an AMR mesh update is not sufficient in combination with DLB. An application oriented solution is described in Appendix C.

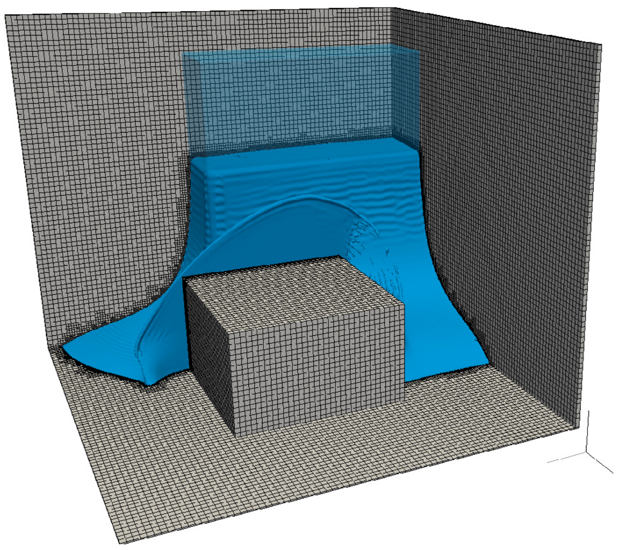
Fig. 5. Simulation result of a dam break with two level refinement at the interface showing $t = 0.24$ s and the initial state.

3.1. Domain decomposition strategy

Both structured and unstructured domain decomposition strategies are available in OpenFOAM. Representative examples are the `simple` structured decomposition method, which subdivides the mesh to a user-defined number of sub-domains in each direction, and the unstructured `scotch` (`ptscotch` in parallel) decomposition, which aims at minimizing the size of inter-processor boundaries (see OpenFOAM User Guide [23] and [24]). The `scotch` (`ptscotch`) decomposition tends to generate new domains from scratch at redistribution, resulting in a large quantity of cell data exchange between processors, and a large memory consumption.

Regardless of the chosen decomposition strategy, refined sibling cells should not be assigned to different sub-domains to preserve unrefinement capabilities. In [21], this limitation was overcome by introducing a new `clustered` decomposition method. However, since OpenFOAM version 4.x, decomposition constraints allow selecting the `refinementHistory` class in the user-specified dictionary `decomposeParDict` to enforce that cell families remain on the same sub-domain, allowing the use of any decomposition strategy while keeping unrefinement capabilities. The process of AMR, DLB and flux correction is summarized in Fig. 4. Unit test-cases in the repository compare the results with and without the changes on flux-flipping during AMR and DLB as well as the enhanced mapping algorithm for surface fields in simple setups of only up to 16 cells.

4. Results

Two typical cases are presented to highlight the performance gains and reduction in necessary computational resources due to AMR and DLB in 2D and 3D. The 3D case is a dam break with obstacle and the 2D case a capillary rise. Both cases are calculated with 2.5 GHz "Intel® Xeon® E52680 v3" processors, using the `interDyMFoam` solver, which captures the interface using an algebraic Volume of Fluid method [25–27]. The main advantage of the introduced method is the reduced number of cells, while maintaining parallel efficiency by re-balancing the computational load between processors.

The simulation of dam break flow is used to understand catastrophic dam-break incidents, promote dam safety and also used

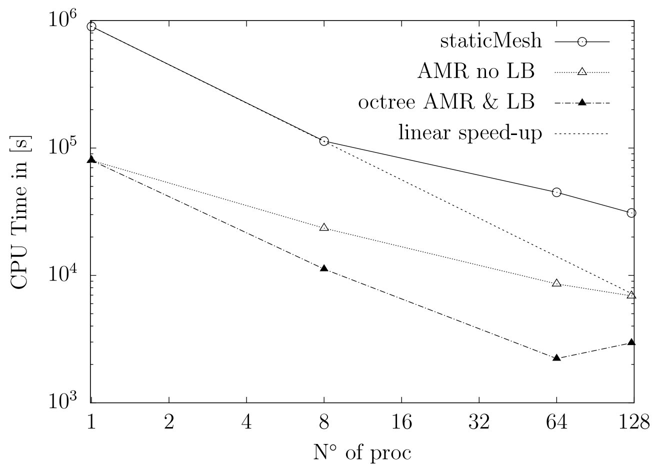

Fig. 6. 3D dam-break case showing the runtime comparing an uniform mesh without refinement as reference with adaptive mesh refinement in two levels with and without dynamic load balancing. The static mesh matches the resolution of the finest refined cells. Using octree-AMR provides a speed-up related to the reduction in cells and DLB helps to maintain the speed-up for an increasing number of processors.

as a validation case in many CFD solvers [5,28,29]. The domain consists of a cube with a base length of 1 meter with a centered obstacle measuring 0.4 m×0.4 m×0.25 m. The water is initialized as a rectangle measuring 0.6 m × 0.19 m × 0.75 m as shown in Fig. 5. Two meshes are compared, one with a uniform static mesh with a spacial resolution of $3.06 \cdot 10^{-3}$ m, resulting in 33.6 million cells and the second one with two level AMR around the interface, matching the static mesh resolution and resulting in a total cell count only 2.6 million cells. The simulation is done with and without DLB. The maximum allowed imbalance is set to 20%. Between the results with and without AMR no difference is visible since the equally resolved interface dominates the flow field.

The speed-up of the simulation for the three different runs is shown in Fig. 6. A speed-up of one order of magnitude is achieved using AMR and DLB for 8, 16 and 125 processors. Because of communication overhead for increasing numbers of processors, only sub-linear speed-up is achieved. Note that the optimal number of processors is smaller for a setup with less cells using AMR.

To outline the necessity of DLB when using AMR, simulations are performed with and without DLB. Simulation times are halved for 125 processors when using DLB compared to a non-balanced simulation. It can be argued that for highly dynamic cases, where the region of interest migrates through the entire domain, the bottleneck originating from load imbalance may even result in lower parallel performance than with the statically refined case.

A two-dimensional case simulating capillary rise between plates is presented in Appendix D. As in the dam-break case it shows significant speed-up and demonstrates the importance of a quadtree over an octree refinement in 2D cases.

Simulation results of an industrial show-case on car aerodynamics are compared in Appendix E using AMR and AMR with LB.

Tutorials for both cases as well as an axisymmetric capillary rise in a tube are appended to the repository. These tutorial test cases are limited to the use of `interDyMFoam`, whereas the AMR and DLB functionality can be applied to the majority of flow solvers in OpenFOAM using FVM.

5. Summary and outlook

An object-oriented approach to load-balanced 2D and 3D adaptive mesh refinement in OpenFOAM has been set out. We present software design, code structure and developments with respect to software usability for engineering applications. We report on several library improvements necessary for stability and runtime performance.

In detail, 2D adaptive mesh refinement is devised for meshes of hexahedral topology in OpenFOAM 4.x and 5.x, which has supported only 3D refinement so far. Our 2D mesh refinement includes refinement rules for both classical 2D and axisymmetric meshes using the mesh cutter class of OpenFOAM, which has been further abstracted. The developments are based on code of Baniabedalruhman used in [15], which has been further enhanced and used for abstraction into mesh cutter classes to avoid code duplication. We have developed a modular framework to support multiple refinement criteria. The dynamic load balancing code by Voskuilen [16] has been enhanced and combined with our developments for adaptive mesh refinement. Significant speed-up has been achieved and demonstrated. It is shown that the performance gain when reducing the overall cell count by adaptive mesh refinement is preserved only in combination with dynamic load balancing. The presented developments are of direct relevance to OpenFOAM's transient top-level solvers for incompressible flow and can be readily used due to its modular software design.

On the short to medium term, development will be devoted to further abstraction, separating the dynamic load balancing functionality from the specific use case of adaptive mesh refinement, i.e. enabling load balanced domain decomposition during simulation along with any desired dynamic mesh class available in OpenFOAM.

Acknowledgments

All authors thank Tyler Voskulien and Ahmad Baniabedalruhman for providing open access to their code and Andrea Montorfano and Stephan Batzdorf for fruitful discussions, as well as Engys Ltd. for their permission to show an industrial test-case. The work is supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universität Darmstadt. The authors also acknowledge support from the Collaborative Research Centre, Interaction between Transport and Wetting Processes SFB 1194 (D.R., H.M., D.B., C.T.) and Collaborative Research Center Transregio, Droplet Dynamics Under Extreme Ambient Conditions SFB-TRR75 (Y.O., E.G., H.D.G.). Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

Appendix A. Flux correction in OpenFOAM

The `interFoam` solver family in particular relies on a divergence free volumetric flux field that corresponds to a divergence free velocity field. Due to non-conservative mapping of surface fields, the flux field needs to be corrected after each AMR step. This correction applies on the entire flux field even in cells not changed by the AMR mesh update. Good initial fluxes ensure convergence and minimizes the correction. To start with, a flux estimate F_f^*

$$F_f^* = \mathbf{u}_f \cdot \mathbf{S}_f \,, \tag{A.1}$$

is calculated extrapolating the flux of the last time step $F_f^{(-1)}$ with the cell face-interpolated velocity field $\mathbf{u}_{f,I}^{(-1)}$

$$\mathbf{u}_f = \left(\mathbf{I} - \mathbf{n}_f \otimes \mathbf{n}_f\right)(\mathbf{u}_{f,I}^{(-1)}) + \frac{F_f^{(-1)}}{|\mathbf{S}_f|}\mathbf{n}_f \,. \tag{A.2}$$

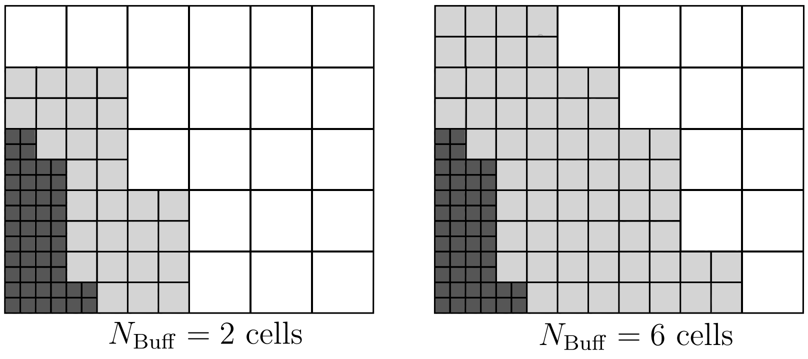

Fig. A.7. Example of two buffer layer enforcing a minimum of two (left) and six (right) buffer cells between two different cell layers.

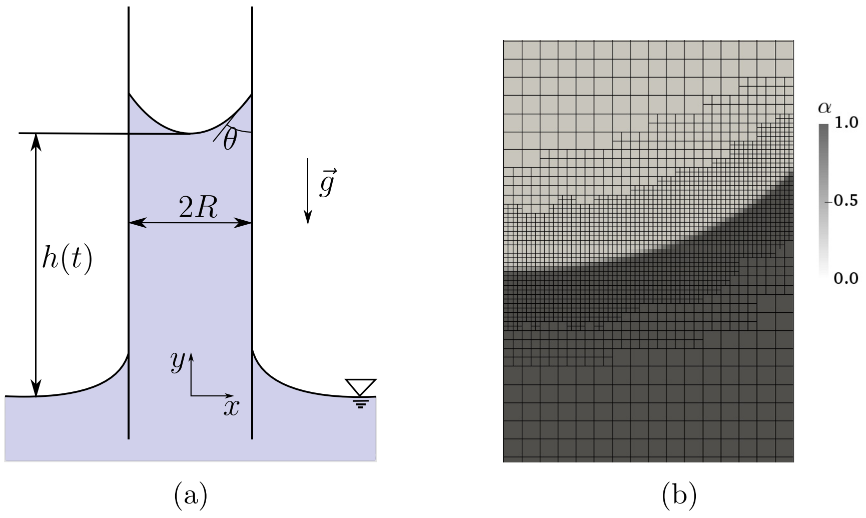

Fig. A.8. (a) Capillary rise setup. (b) Two level refinement at the interface and a contact angle of 40°. For better visualization the resolution of the half width is reduced to $R = 64$. The volume fraction α indicates which cells are filled with liquid.

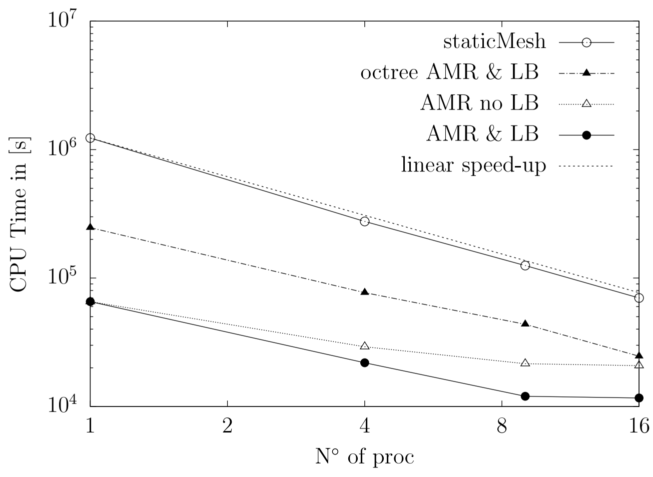

Fig. A.9. 2D capillary rise between plates case showing the speed-up when using AMR in two levels with and without DLB versus uniform mesh without refinement, matching the resolution of the finest refined cells. Using octree-AMR provides an speed-up related to the reduction in cells which is even further reduced by the quadtree AMR. DLB helps to maintain the speed-up for an increasing number of sub-domains.

Hereby $\mathbf{S}_f$ denotes the surface area vector and $\mathbf{n}_f = \mathbf{S}_f/|\mathbf{S}_f|$ the face normal. Face-centered values are marked with the subscript f. Using this flux estimate F_f^*, the pressure Laplace equation is solved iteratively

$$\sum_f \left[\frac{1}{A_f} \nabla_f \, p_{\text{corr}}^n \, |\mathbf{S}_f| \right] = \sum_f F_f^* \,. \tag{A.3}$$

In the above equation, A_f is the central coefficient of the discretized momentum equation and ∇_f the surface normal gradient. Finally, the estimated flux is updated by the pressure correction term to obtain the divergence free flux

$$F_f^n = F_f^* - \frac{1}{A_f} \nabla_f \, p_{\text{corr}}^n \, |\mathbf{S}_f| \,. \tag{A.4}$$

Using the different improvements namely the addressing fix, the enhanced surface field mapping and the correction in sign-flipping, as mentioned in Section 2 significantly reduces the necessary correction iteration to ensure a divergence free field, while improving solver performance, robustness and accuracy [10].

Appendix B. Buffer layer implementation

The buffer layers are the number of cell layers between two refinement levels. In meshes with more than one cell refinement level it is important to ensure a smooth transition between the

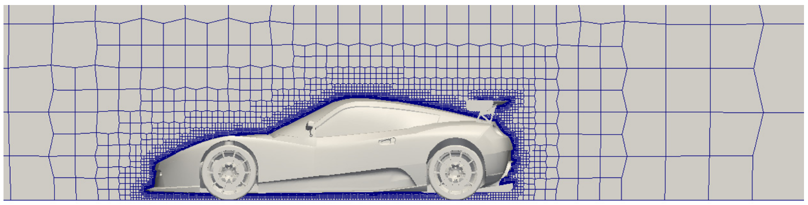

Fig. A.10. Start mesh configuration using 7 refinement layer.
Source: Courtesy of Engys Ltd.

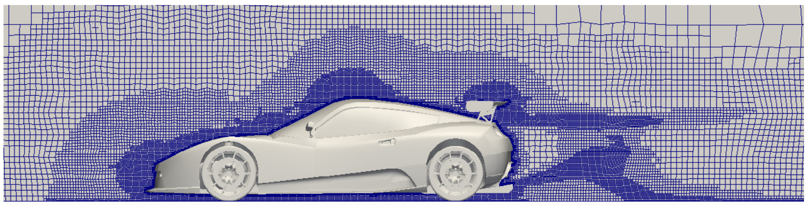

Fig. A.11. Final mesh configuration in steady-state.

different levels in order to sufficiently decrease discretization errors due to mesh skewness and to provide a buffer between two refinement levels for the physical properties to adapt to the new mesh level. Which is a trade-off between computational costs due to mesh size and additional overhead necessary to allow the flow field to adapt to topological changes.

The calculation of the target refinement level for each cell is performed as follows. Each cell holds the information of its current and target refinement level. In a recursive manner and beginning with the highest refinement level, cells with this level are marked to preserve their refinement level. Then all adjacent cells of this preserved-set are found, as often as the user set number of buffer layers indicates. Therefore, the selected neighboring cells are added to a neighbors-set of cells, to repeat the search of neighbors using the updated neighbors-set. The newly selected neighbors receive the current refinement level of the preserved-set decremented by one as target refinement level. The whole procedure is repeated by merging the neighbors-set to the preserved-set searching for new neighbor cells and setting their target level to the highest refinement level decremented by the number of performed recursions, which are counted beginning with one.

Neighbor cells are those that share a cell corner point instead of faces. The choice of points for propagation compared to faces is motivated by the larger directional preference exhibited by face based propagation.

Fig. A.7 shows examples of refined meshes using respectively two and six buffer layers. The number of effective buffer layers may exceed the specified number by one, since refining a cell creates two cells in each refined direction. Although one buffer layer (1:1) is enforced by the refinement engine, a minimum of two layers (2:1) is recommended to ensure a smooth field transition.

Appendix C. Boundary condition initialization

During DLB, stability issues appear using boundaries that hold a value such as for general Neumann or Dirichlet boundary conditions. Similarly to the mapping issue explained in Section 2.1, a proper mapping of values from parent to child cells is required. At boundaries this mapping is not provided by default in the basic boundary conditions such as for Neumann boundary conditions (`fixedGradientFvPatchField`) from which contact angle boundary class (`alphaContactAngleFvPatchScalarField`) is derived. This contact angle boundary condition is necessary for the capillary rise test case in Section 4. During refinement and DLB, the boundary field size is updated, but new faces lack proper initialization. This issue leads to either incorrect gradient values or to exceptions. To prevent interruptions in our specific application, new faces are assigned with a zero normal gradient boundary condition. This corresponds to a contact angle of 90°. Using an appropriate refinement-criteria, we recommend to ensure, that cells containing part of the three-phase contact line are never refined nor unrefined to make sure that the boundary condition acts as expected. There is no guard implemented to help detect or prevent such (un)refinement. However, treating the two-phase interface with the highest level of refinement is sufficient to avoid this issue.

Appendix D. Case description of capillary rise between two plates

The capillary rise between plates is a two-dimensional case used in literature as a benchmark for curvature calculation and validation for contact angle models [30,31] since an analytic solution for prescribed constant contact angles is known [32]. The liquid–gas interface is initialized horizontally between the plates and rises or falls over time depending on the applied contact angle, here chosen as 40°. The material parameters of air and the liquid are set as presented in [30] With the surface tension $\sigma = 72.75\,\mathrm{mN/m}$ and the kinematic viscosity of $\eta = 1.5 \cdot 10^{-5}\,\mathrm{m^2/s}$ and $\eta = 10 \cdot 10^{-5}\,\mathrm{m^2/s}$ as well as the density $\varrho = 1.19\,\mathrm{m^3/kg}$ and $\varrho = 998\,\mathrm{m^3/kg}$, respectively. In order to cover the rise of the two-phase interface, the entire domain is covered by a fine static mesh. As only the interface evolution requires high accuracy (see Fig. A.8(b)), this test case is a good candidate to apply AMR and DLB.

The static discretization of the domain (see Fig. A.8(a)) uses a uniform hexahedral mesh resolving the half width of $R = 0.625\,\mathrm{mm}$ between the two plates with 128 cells and the height of $17R$ with 2176 cells, in total $5.6 \cdot 10^5$ cells. The dynamically refined mesh is set with two refinement levels around the interface, matching the resolution of the static mesh at the interface. The resulting cell number is $4.3 \cdot 10^4$ cells.

As for the dam break case in Section 4, the speed-up with 2D-AMR achieved for the capillary rise case is around one order of magnitude, which directly corresponds to the reduction in

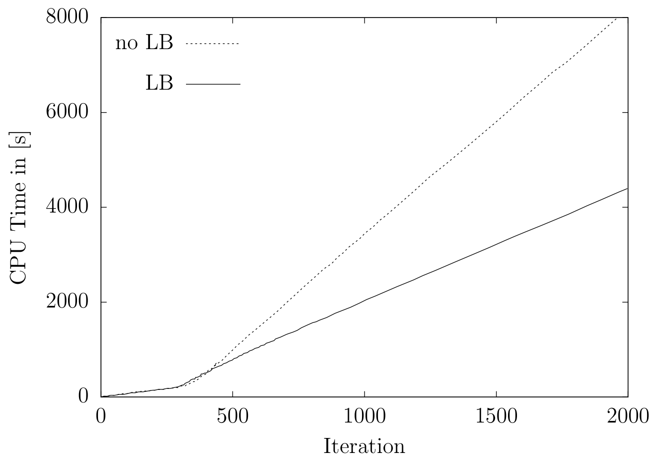

Fig. D.12. Simulation time for AMR vs AMR with LB (LB start at $t = 300$ s). *Source:* Courtesy of Engys Ltd.

cell count (see Fig. A.9). AMR with DLB still produces a two-fold increase in performance compared to simple AMR. To point out the benefit of having a true two-dimensional refinement in two-dimensional cases, additional simulations are performed using octree refinement instead of quadtree refinement. In this case, octree refinement is about five times slower than quadtree refinement, due to the larger number of child cells per parent as well as the introduction of parasitic flow in the invariant direction.

Appendix E. Multi-criterion adaptive mesh refinement and dynamic load balancing applied in an industrial case

By courtesy of Engys Ltd., we showcase the AMR capabilities for a more complex and realistic application, a typical car aerodynamic simulation (steady-state Reynolds-Averaged Navier Stokes on 48 processor) was performed for the record configuration of the ERA electric car in Nürburgring. Simulated is a symmetric half model with 7 refinement levels and minimum wall-normal cell height of 8 mm (see start configuration in Fig. A.10). The refinement in this case is based on pressure and velocity gradient values from an engineering guess. The criteria using the pressure gradient are:

- $30\,\mathrm{m/s^2}$ for refinement level 4,
- $150\,\mathrm{m/s^2}$ for refinement level 5,
- $600\,\mathrm{m/s^2}$ for refinement level 6.

Additional a criteria using the velocity gradient is applied with

- 10 1/s for refinement level 4,
- 50 1/s for refinement level 5,
- 100 1/s for refinement level 6.

The resulting steady-state mesh configuration is shown in Fig. A.11. In the presented setup, the final mesh size using the AMR approach (2.6 Mio cells) is less than 50% of the standard best-practices stationary mesh refinement for car aerodynamic simulations (5.7 Mio cells), leading to a significant speed-up of the time-to-solution. It is further shown, that usage of additional dynamic load balancing leads to a further 50% decrease of the time-to-solution (cf. Fig. D.12) and hence is essential to obtain maximum speed-up performance.

References

[1] Berger MJ, Colella P. Local adaptive mesh refinement for shock hydrodynamics. J Comput Phys 1989;82(1):64–84. http://dx.doi.org/10.1016/0021-9991(89)90035-1.

[2] Jasak H. Error analysis and estimation for the finite volume method with applications to fluid flows [Ph.D. thesis], Imperial College of Science; 1996.

[3] Vuong A-V, Boschert S, Simeon B. Adaptive finite volume methods for interfacial flows. 2008, Available at http://www-m2.ma.tum.de/homepages/simeon/publica.html. [Last Accessed June 2018].

[4] Schwing AM, Nompelis I, Candler GV. Implementation of adaptive mesh refinement in an implicit unstructured finite-volume flow solver. In: 21st AIAA Computational fluid dyn. conference. San Diego, CA: American Institute of Aeronautics and Astronautics; 2013, http://dx.doi.org/10.2514/6.2013-2446.

[5] Fondelli T, Andreini A, Facchini B. Numerical simulation of dam-break problem using an adaptive meshing approach. Energy Procedia 2015;82:309–15. http://dx.doi.org/10.1016/j.egypro.2015.12.038.

[6] Adams M, Schwartz PO, Johansen H, Colella P, Ligocki TJ, Martin D, et al. Chombo software package for AMR applications – design document. Technical report LBNL-6616E, Berkeley National Laboratory; 2015, URL https://escholarship.org/uc/item/5cs5d1sq.

[7] Flaherty J, Loy R, Shephard M, Szymanski B, Teresco J, Ziantz L. Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws. J Parallel Distrib Comput 1997;47(2):139–52. http://dx.doi.org/10.1006/jpdc.1997.1412.

[8] Load balancing computational Fluid Dyn. calculations on unstructured grids. In: Vandriessche R, and Roose D, editors. AGARD, Special course on parallel comput. in CFD; 1995. (see N96-16247 04-60). URL http://adsabs.harvard.edu/abs/1995scpc.agar.....V.

[9] Misaka T, Sasaki D, Obayashi S. Adaptive mesh refinement and load balancing based on multi-level block-structured Cartesian mesh. Int J Comput Fluid Dyn 2017;1–12. http://dx.doi.org/10.1080/10618562.2017.1390085.

[10] Deising D, Bothe D, Marschall H. Direct numerical simulation of mass transfer in bubbly flows. Comput Fluids 2018. http://dx.doi.org/10.1016/j.compfluid.2018.03.041.

[11] Gurumurthy VT, Rettenmaier D, Roisman IV, Tropea C, Garoff S. Computations of spontaneous rise of a rivulet in a corner of a vertical square capillary. Colloids Surf A 2018;544:118–26. http://dx.doi.org/10.1016/j.colsurfa.2018.02.003.

[12] Weller HG, Tabor G, Jasak H, Fureby C. A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput Phys 1998;12(6):620–31. http://dx.doi.org/10.1063/1.168744.

[13] Jasak H. OpenFOAM: Open source CFD in research and industry. Int J Nav Archit Ocean Eng 2009;1(2):89–94. http://dx.doi.org/10.2478/IJNAOE-2013-0011.

[14] Moukalled F, Mangani L, Darwish M. The finite volume method in computational fluid dyn, vol. 113. Springer Int. Publishing; 2016, http://dx.doi.org/10.1007/978-3-319-16874-6.

[15] Baniabedalruhman A. Dynamic meshing around fluid-fluid interfaces with applications to droplet tracking in contraction geometries [Ph.D. thesis], Michigan Technological University; 2015, URL https://digitalcommons.mtu.edu/etds/1005.

[16] Voskuilen T. Mesh balancing. GitHub Repository, GitHub; 2014, [Last Accessed July 2018].

[17] Van Phuc P, Chiba S, Minami K. Large scale transient CFD simulations for buildings using OpenFOAM on a world's top-class supercomputer. In: 4th OpenFOAM user conference, Cologne, Germany; 2016.

[18] Meredith K, Zhou X, Wang Y. Towards resolving the atomization process of an idealized fire sprinkler with VOF modeling. In: ILASS Europe. 28th European conference on liquid atomization and spray systems. Editorial Universitat Politècnica de València; 2017, p. 257–64. http://dx.doi.org/10.4995/ilass2017.2017.5014.

[19] Meredith K, Vukčević V. Resolving the near-field flow patterns of an idealized fire sprinkler with VOF modeling and adpative mesh refinement. In: 13th OpenFOAM workshop in Shanghai China; 2018.

[20] Joshi SV. Adaptive mesh refinement in openfoam with quantified error bounds and support for arbitrary cell-types [Ph.D. thesis], Institut für Informatik, Technische Universität München; 2016.

[21] Batzdorf S. Heat transfer and evaporation during single drop impingement onto a superheated wall [Ph.D. thesis], Institute of Technical Thermodynamics, Technical University Darmstadt; 2015, URL http://tuprints.ulb.tu-darmstadt.de/id/eprint/4542.

[22] Mooney K. Implementation of a moving immersed boundary method on a dynamically refining mesh with automatic load balancing. In: 10th OpenFOAM workshop. Ann Arbor MI; 2015.

[23] Direct CFD. OpenFOAM user guide. 2016.

[24] Chevalier C, Pellegrini F. PT-scotch: A tool for efficient parallel graph ordering. Parallel Comput 2008;34(6):318–31. http://dx.doi.org/10.1016/j.parco.2007.12.001, Parallel matrix algorithms and applications. URL http://www.sciencedirect.com/science/article/pii/S0167819107001342.

[25] Hirt C, Nichols B. Volume of Fluid (VOF) method for the dynamics of free boundaries. J Comput Phys 1981;39(1):201–25. http://dx.doi.org/10.1016/0021-9991(81)90145-5.

[26] Ubbink O. Numerical prediction of two fluid systems with sharp interfaces [Ph.D. thesis], Department of Mechanical Engineering, Imperial College of Science, Technology & Medicine; 1997, URL http://hdl.handle.net/10044/1/8604.

[27] Muzaferija S, Periĉ M. Computations of free-surface flows using the Finite-Volume Method and moving grids. Numer Heat Transfer B 1997;32:369–84. http://dx.doi.org/10.1080/10407799708915014.

[28] Biscarini C, Di Francesco S, Manciola P. CFD modelling approach for dam break flow studies. Hydrol Earth Syst Sci 2010;14(4):705. http://dx.doi.org/10.5194/hess-14-705-2010.

[29] Gada VH, Tandon MP, Elias J, Vikulov R, Lo S. A large scale interface multi-fluid model for simulating multiphase flows. Appl Math Model 2017;44:189–204. http://dx.doi.org/10.1016/j.apm.2017.02.030.

[30] Fath A, Bothe D. Direct Numerical simulations of thermocapillary migration of a droplet attached to a solid wall. Int J Multiph Flow 2015;77:209–21. http://dx.doi.org/10.1016/j.ijmultiphaseflow.2015.08.018.

[31] Xu H, Guetari C. The use of CFD to simulate capillary rise and comparison to experimental data. In: 2004 international ANSYS conference. Pittsburgh; 2004.

[32] Fries N, Dreyer M. An analytic solution of capillary rise restrained by gravity. J Colloid Interface Sci 2008;320(1):259–63. http://dx.doi.org/10.1016/j.jcis.2008.01.009.