Institut de Mathématiques de Toulon

INTERNSHIP PROJECT

STUDY OF DAM BREAK PROBLEM WITH FOCUS ON ADAPTIVE MESH REFINEMENT

Author: Josef Musil
Supervisor: Fréderic Golay
Year: 2019

Abbreviations

Symbol	Description	Unit
$\mathbf{u}$	velocity vector	$[m/s]$
u	velocity component in x-direction	$[m/s]$
v	velocity component in y-direction	$[m/s]$
w	velocity component in z-direction	$[m/s]$
ρ	density	$[kg/m^3]$
ρ_W	density of water phase	$[kg/m^3]$
ρ_A	density of air phase	$[kg/m^3]$
p	pressure	$[Pa]$
T	thermodynamic temperature	$[K]$
α, φ	volume fraction function	$[-]$
μ	kinematic viscosity	$[m^2/s]$
σ	surface tension	$[N/m]$
κ	surface curvature	$[m^{-1}]$
g	gravitational acceleration	$[m/s^2]$
$\mathbf{g}$	vector of gravitational acceleration	$[m/s^2]$
t	time	$[s]$
x, y, z	spatial coordinates	
$(\cdot)_\xi$	partial derivative with respect to arbitrary variable	
c_0	artificial speed of sound	$[m/s]$
s	entropy function	$[-]$
ϕ	entropy flux	$[-]$
S	numerical entropy production	$[J/kg]$
h	height of free surface	$[m]$

Introduction

The computational fluid dynamics (CFD) is scientific-engineering discipline primarily focused on solving partial differential equations (PDE) of fluid mechanics by means of computational machines. This discipline has developed as a consequence of fact that majority of these equations are impossible to solve with analytic approach. In order to get some form of approximation of the solution, given partial differential equation (or equations) which describes certain physical phenomenon has to be re-expressed in suitable form for computation and then solved numerically. There have been developed several approaches to handle this task. From perspective of this work, these can be distinguished as mesh-based methods and mesh-free methods. The latter involving large number of other subsequent methods, i.e. smoothed particle hydrodynamics (one of the earliest), will not be further discussed.

This work will focus solely on finite volume method (FVM), mesh-based method which is considered to be widely used for solving problems in industrial sector. Furthermore, attention will be paid to technique called Adaptive Mesh Refinement (AMR), which serves to effectively refine computational mesh in areas where physical phenomena need to be captured with more precision and, otherwise, in order to maintain overall number of cells within reasonable limit, coarse areas where steep spatial gradients or large temporal changes of physical quantities are not present.

At first in this work will be provided a brief summary/review of current state of art of AMR methods. The review will be mostly organized as a collection of ideas and facts borrowed from scientific literature and articles which already cover this topic both in wide and deep. Afterwards, performance of three numerical solvers which use AMR technique will be tested on 2D dam-break problem without obstacle. Comparison with experimental results will be also made in order to validate numerical models with physical reality. The main focus of this work is to provide some explanation and tutorial how to set-up, run (and possibly post-process) aforementioned computational cases of dam-break problem.

1. Adaptive mesh refinement

The adaptive mesh refinement method (AMR) is a process where cells of computational mesh are split into smaller non-overlapping, disjunct cells or on the contrary they are merged together, see Figure 1 and Figure 2. This process serves to compute development of given physical system with more spatial (and also temporal) resolution in region of higher interest (i.e. relatively high gradients of physical quantities), or otherwise save computational effort by lowering the resolution in regions where our system is developing slowly or small gradients are present only. Within numerical simulation, the mesh refinement/coarsening usually takes place (if prescribed) before advancing in time to the next time step.

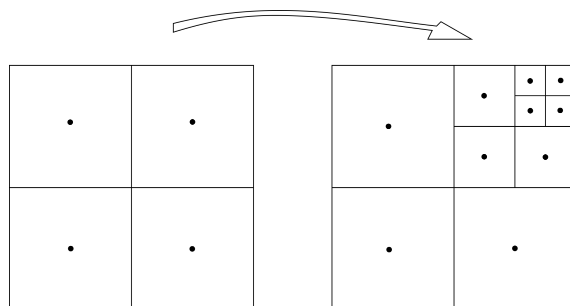

Figure 1: Two layer mesh refinement of Cartesian mesh.

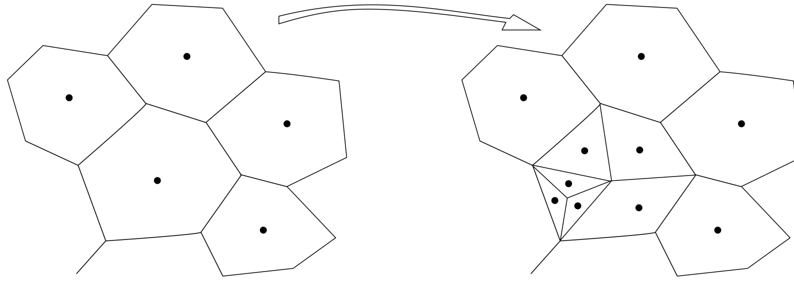

Figure 2: Two layer mesh refinement of unstructured, polyhedral mesh.

On the Figures 1 and 2 there are shown two distinct types of meshes, namely structured and unstructured. During the refinement/coarsening procedure some interpolation technique has to be introduced while assigning field values in new - splited/merged cells in order maintain spatial accuracy of computation and other important qualities, e.g. conservation of mass.

On unstructured mesh, this task largely depends on geometrical configuration of (in general) polyhedral cells. From that configuration, determined by position of cells vertices and mass center, coefficients for interpolation are obtained and fields values in new cells can be assigned afterwards. This whole task is harder do on unstructured meshes, because cells labeling is un-ordered and requires list of cells connectivity (changing at each re-meshing). Unlike the Cartesian meshes.

On Cartesian meshes, it is much easier to propose such interpolation algorithm since the geometrical configuration of mesh cells (ordering) is much simpler. Therefore there can be introduced algorithms that take advantage of such cells ordering. These algorithms (also data structures) are called *quadtree* of *octree*, depending whether one is dealing with 2D or 3D geometry, respectively.

1.1 Quadtree/Octree family of algorithms

In this section, there will be given brief description of quadtree algorithm. The octree algorithm can be than considered as straightforward extension of quadtree in three spatial dimensions. The literature dealing with comprehensive description of these algorithms is vast, see e.g. [1], [2] and [3]. Note that quadtree algorithm is used not only in computational fluid dynamics but also for example in image storage and compression, effective storing of curves, storing sparse data etc.

In terms of data structure a quadtree is a tree-based hierarchical structure which stores information of computational cell's size and connectivity. The algorithm and data structure can be demonstrated on the most simple case of square computational domain, $\Omega = [0, 1] \times [0, 1]$, as follows.

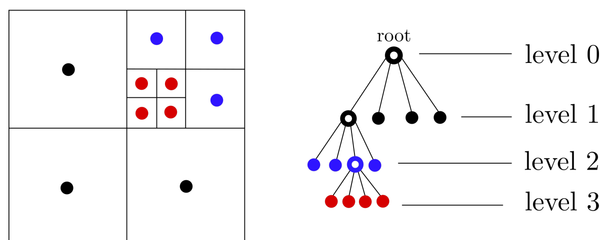

Figure 3: Cartesian mesh (left) with example of refinement history represented by quadtree data structure (right). Empty symbols in quadtree represent *parents* and full symbols represent *leaves*.

The attributes of computational cells can be stored in quadtree structure in several different ways, usually depending on suitable tools of chosen programming language. For example in `C/C++` language the structure can be represented by using a linked list. With using pointers a parent cell is connected with its *child* cells (which could be another, lower level, parents or leaves). The example of a such structure is described as follows. Numbering system for child cells in the each parent cell is depicted on Figure 4.

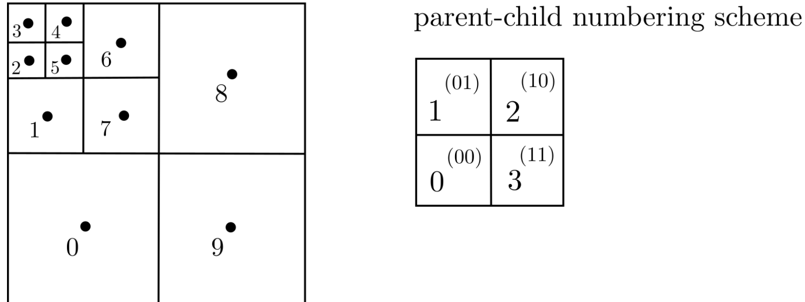

Figure 4: Refined mesh with cell labels (left). Scheme for walking through the cells on particular refinement level (right).

The cell labeling is organized as follows:

I. On each level, cell numbering continues with bottom-left cell and then clockwise (according to the scheme on Figure 4)

II. If current cell is further refined, labeling continues on next level

III. If bottom-right cell is reached (labeled) on given level, labeling continues on the previous level according I. and II. until 1st level cells are all labeled.

Furthermore, the quadtree is constructed via pointers. Each parent points to its children and its neighbors. If node in tree has no children (NULL pointer) then it is called *leaf* and represent corresponding computational cell. The cells can also point to any other attributes, such as geometrical data and physical quantities.

From the perspective of numerical computation the construction of a good numerical scheme plays important role here. In majority of codes it is required that refinement of neighboring cells differ at most of one level. If there is difference in refinement, the numerical flux needs to be designed to obey mass conservation. This is mostly done through interpolation of coarser cells to finer ones and vice-versa which can be described as auxiliary ghost cells at the coarse-fine cells interface, see Figure 5.

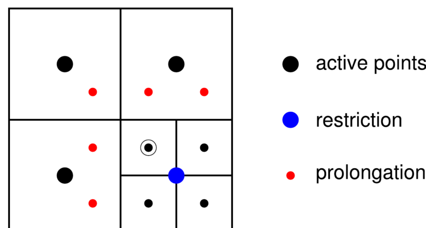

Figure 5: Example of interpolation procedure (Basilisk). At first, blue value is constructed from four black values by simple averaging. (Interpolation from fine-to-coarse is often referred as *restriction*). Secondly, red values are interpolated using active points and blue one (coarse-to-fine interpolation is here referred as *prolongation*). Finally, numerical scheme for black cell in circle can be constructed (here 3x3 stencil, hence 2nd order spatial accuracy). Picture is taken from [4].

So far the quadtree algorithm was here demonstrated on rectangular computational domain and each *leaf* of the tree here represented actual computational cell. This does not need to be the case in general. Some works employ so called *block-based* AMR technique for Cartesian meshes, e.g. [5] and [6], and some use curvilinear, structured, body-fitted meshes, e.g. [7], [8], with further improvements of allowing the mesh to adapt anisotropically (quadtree is heavily modified in this case).

In the block-based technique the leaf in quadtree (or any other tree-structure) now represent block of predefined number of cells, thus leading to much lighter tree structure than cell-based approach, but, with losing the possibility to adapt each cell independently. When arbitrary block is refined resp. coarsened, the new blocks resp. block contain(s) the same amount of original cells, e.g. 8×8-block is refined into four 8×8-blocks (see Figure 6, 3rd block).

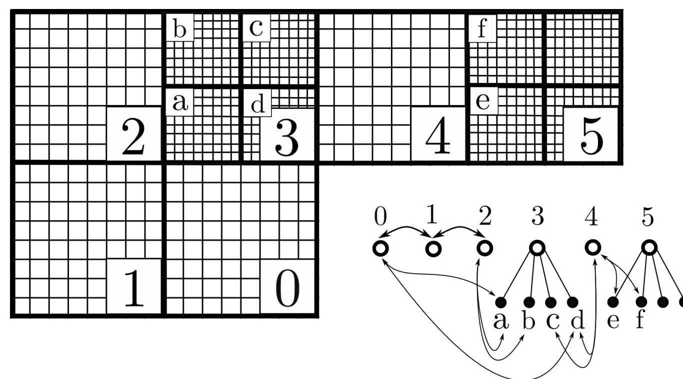

Figure 6: Example of computational domain divided into 6 initial (bold lines) blocks, each treated with its own quadtree, block-based structure. The additional treatment of communication through block boundaries is expressed by arrows. This approach enables to treat more complex geometries.

As mentioned above, the block-based approach can be also used to deal with more complex computational domains with curved boundaries. This technique is employed in [9].

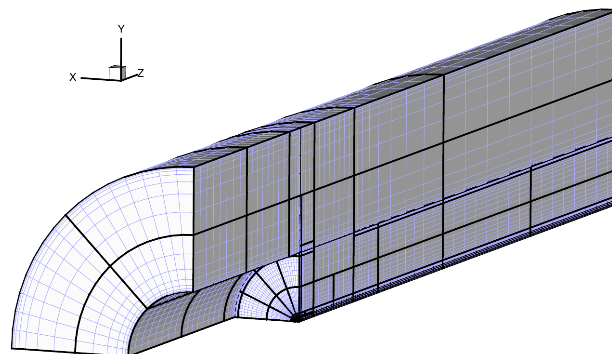

Figure 7: Body-fitted adapted mesh after several re-finements. Grid blocks are shown with boldlines. Picture is taken from [10].

2. Computational case

Throughout the rest of this work there will be studied two-dimensional dam-break problem without the obstacle from perspective of several numerical solvers which are implemented in different software. In all cases the geometry, initial conditions and initial mesh are set the same, according to the experimental setting in [11]. Figure 8 shows initial setting with water column, represented by red color situated in left of a computational domain.

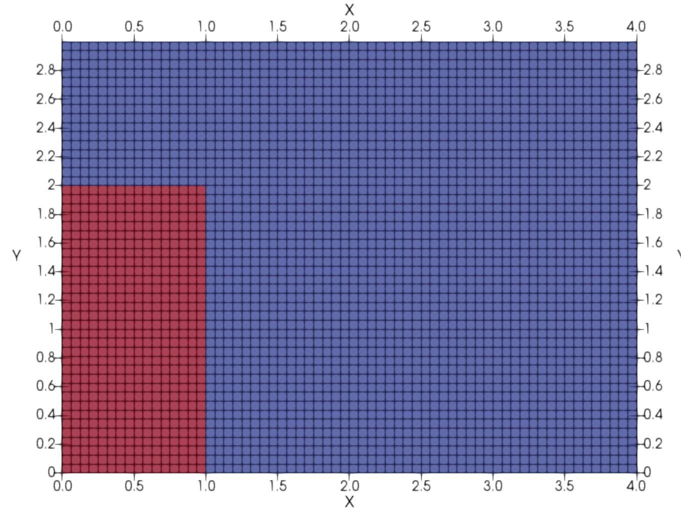

Figure 8: Dam-break case study, initial set-up.

The computational domain is given by $\Omega = \{[x, y] \in \mathbb{E}_2,\ 0 \leq x \leq 4, 0 \leq y \leq 3\}$. The domain is discretised in Cartesian way with 64×48 quadrilateral cells and during the execution, in of all the solvers the level of maximal refinement is set to 4. It means that the minimum size of smallest computational cell is 16 times less than the original/initial one. Boundary conditions are considered as free-slip walls with total pressure set to be zero at the top wall. Specific realization of these boundary conditions is software-dependent. Initial velocity is set as zero in whole domain (or some negligible value is prescribed to the water column at the start in some cases). All the numerical simulations are executed up to time $T = 0.8s$ and the results then compared also with experimental data [11].

3. BBAMR

In this section there is described numerical solver BB-AMR (Block-Based Adaptive Mesh Refinement) developed by T.Altazin, M.Ersoy, F.Golay, D.Sous and L.Yushchenko. The software is written in Fortran 90 programming language. More detailed description of the solver can be found in [5].

The aim of the solver is to provide computationally efficient tool for solving hyperbolic systems with adaptive mesh refinement based on (unstructured) block computational mesh. Here, the mesh refinement is based on numerical density of entropy production function which provides quite general approach for indicating numerical errors in all hyperbolic problems without shock waves. In order to maintain efficiency of the solver, phase interface sharpening is handled by adding specially designed source terms to right hand side of governing equations (3.1, 3.4 and 3.2) which are then solved with fractional inner time step procedure.

3.1 Governing equations

The governing equations which describe compressible low Mach two-phase flows in BBAMR are presented in this section.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{3.1}$$

$$\frac{\partial \varphi}{\partial t} + \mathbf{u} \cdot \nabla \varphi = 0 \tag{3.2}$$

$$\rho = \varphi \rho_A + (1 - \varphi)\rho_W \tag{3.3}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) + \nabla p = \rho \mathbf{g} \tag{3.4}$$

The equation (3.1) represents mass conservation law. Here ρ and $\mathbf{u}$ denotes for density and velocity field respectively. The two-phase model of mixture of water and air is here managed by advection of coloring function, equation (3.2). Furthermore, density field ρ is defined by equation (3.3) as a linear combination of water density ρ_W and air density ρ_W. The equation (3.4) represent conservation of momentum. Here, p stands for pressure and $\mathbf{g}$ for gravitational acceleration.

For the purpose of closing the system of equations (3.1-3.4), the artificial pressure law is introduced here

$$p = c_0^2 \Big(\rho - \big(\varphi \rho_A + (1 - \varphi)\rho_W \big) \Big) + p_0 \tag{3.5}$$

Here constant c_0 refers to artificial speed of sound, chosen as a compromise between the limits of compressible effects, the rate of numerical diffusion and a reasonable CFL constraint. The value was here chosen as $c_0 = 20\ m/s$, [5],[12].

The refinement criterion is determined by numerical entropy production inequality

$$S := \frac{\partial s}{\partial t} + \nabla \cdot \boldsymbol{\psi} \leq 0 \tag{3.6}$$

where (s, ψ) denotes convex entropy-entropy flux pair (thus the sign in the inequality is the opposite than physical entropy) and depend on conservative variables. Here, in the

context of two-phase compressible flows, the entropy is expressed as

$$s = \tfrac{1}{2}\rho\mathbf{u}^2 + c_0^2\rho\ln\rho - c_0^2(\rho_W - \rho_A)\varphi \tag{3.7}$$

and the entropy flux is given by

$$\boldsymbol{\psi} = \left(\tfrac{1}{2}\rho|\mathbf{u}|^2 + c_0^2\rho(\ln\rho + 1)\right)\mathbf{u} \tag{3.8}$$

Mesh refinement parameters are then based on comparison of ratio between local entropy production (production within given block represented by corresponding initial cell) and global entropy production (in whole computational domain) with pre-defined threshold values:

- if $\dfrac{S_{block}}{S_{global}} \leq \alpha_{min}$ the mesh in corresponding block is refined
- if $\dfrac{S_{block}}{S_{global}} \geq \alpha_{max}$ the mesh in corresponding block is coarsened.

Here $0 \leq \alpha_{min} \leq \alpha_{max} \leq 1$ represents user-defined threshold values for refinement/coarsening the mesh.

3.2 Numerical schemes

The system of equations (3.1, 3.2 and 3.4) is discretized by finite-volume method and after that solved numerically. The semi-discretized form of aforementioned equations reads as follows

$$\frac{\partial \mathbf{w}_k(t)}{\partial t} + \frac{1}{|C_k|}\sum_a |\partial C_{k/a}|\mathbf{F}\big(\mathbf{w}_k(t), \mathbf{w}_a(t), \mathbf{n}_{k/a}\big) = 0 \tag{3.9}$$

where $\mathbf{w}_k(t)$ denotes spatial mean value in arbitrary computational cell C_k of a vector of conservative variables, $\mathbf{w}(t,\mathbf{x}) = (\rho, \rho\mathbf{u}, \rho\varphi)^T$. Additionally, $|C_k|$ denotes N-dimensional volume of computational cell C_k and $|\partial C_{k/a}|$ denotes $(N-1)$-dimensional volume of $\partial C_{k/a}$ face between cells k and a where N represents spatial dimension of computational problem. Numerical flux of conservative variables at the interface k/a is denoted by $\mathbf{F}\big(\mathbf{w}_k(t), \mathbf{w}_a(t)\big)$ and similarly $\mathbf{n}_{k/a}$ stands for for unit normal vector on $C_{k/a}$ pointing from k to a.

The time-step integration is realized by second order Adams-Bashforth scheme without local time stepping. Hence temporally discretized equation (3.9) reads as

$$\begin{aligned}\mathbf{w}_k(t_{n+1}) = \mathbf{w}_k(t_n) &- \frac{\delta t_n}{|C_k|}\sum_a |\partial C_{k/a}|\mathbf{F}^n \\ &- \frac{\delta t_n^2}{2\delta t_{n-1}|C_k|}\Big(\sum_a |\partial C_{k/a}|\mathbf{F}^n - \sum_a |\partial C_{k/a}|\mathbf{F}^{n-1}\Big) + \mathbf{S}\end{aligned} \tag{3.10}$$

Here δt_i denotes $i-$th timestep and $\mathbf{F}^i = \mathbf{F}\big(\mathbf{w}_k(t_i), \mathbf{w}_a(t_i), \mathbf{n}_{k/a}\big)$. On right-hand side of equation (3.10) there is added extra source term $\mathbf{S} = \big(S_c, S_c(\rho_A - \rho_W), S_c\mathbf{u}(\rho_A - \rho_W)\big)^T$ in contrast of equation (3.9) serving as a phase-interface sharpening mechanism in region where both water and air phases coexist. Definition of the parameter S_c can be found in

[5]. The numerical flux $\mathbf{F}^i$ is computed using second order MUSCL scheme, for details see [13]. The same discretization method as described by equation (3.10) was used for entropy production (3.6).

3.3 Setting the case

The setting of the computational case will be described in this part. Once the BBAMR software is compiled the input file has to be created. We suppose that input file `damBreak.inp` is located in `$HOME/BBAMR/exec`. The file is listed in Appendix section, here. Most of the important parameters used `damBreak.inp` file are explained within comments in the file. Some parameters are referenced with double hash symbol, i.e. `NRMA #NRMA#`. This is because we want to manage these values by shell script.

The following bash script (presumed to be located in `$HOME/BBAMR/exec/run_dam.sh`) is here also needed to manage the code run with AMR procedure and adds some output, useful for post-processing. Note that the code here (as well as in other case) is executed in parallel, on 4 cores (`nproc=4`).

```
#bin.sh
start=`date +%s`
nc=1
nproc=4
N=100

cp dam.inp bbamr.inp
mkdir -p Dam_data/case_00$nc

sed -i 's/#NRMA#/4/g'          bbamr.inp # max refinement level
sed -i 's/#NRMA_init#/4/g'     bbamr.inp # max_initial refinement level
sed -i 's/#VCDE#/7.5/g'        bbamr.inp # lower treshold for refinement
sed -i 's/#VCRA#/9/g'          bbamr.inp # upper treshold for refinement
sed -i 's/#TFIN#/1.0/g'        bbamr.inp # final time

./convert_mai
./bbamr2tecamr
./bbamr2tec
mv tecamr.dat  Dam_data/case_00$nc/dam_amr_0.dat
mv tecplot.dat Dam_data/case_00$nc/dam_tec_0.dat

for i in $(seq 1 $N)
do
    mpirun -np $nproc ./bbamr
    ./bbamr2tecamr
    mv tecamr.dat Dam_data/case_00$nc/dam_amr_$i.dat
    ./bbamr2tec
    mv tecplot.dat Dam_data/case_00$nc/dam_tec_$i.dat
    ./amr
    mv amr.dat Dam_data/case_00$nc/dam_bloc_$i.dat
    end=`date +%s`
    runtime=$(($end-$start))
    printf "\n"
    printf "Computation time: "
    echo "Computation time: $runtime "
    printf "\n\n"
done

rm solbin_* bbamr.inp
```

Finally, to run the code (in parallel, with AMR), user just need to execute bash script and redirect the output to the log file by

```
sh run_dam.sh > log
```

3.4 Post-processing the numerical results

In order to post-process the output file (`log`) of the numerical computation, namely, to get the dependency of number of computational cells and computational time on simulation real time, there is provided small shell script (`postProc_BBAMR.sh`) to handle this task.

```
grep -i -B 16 'Average' log_1 > foo
awk '{for(i=1;i<=NF;i++)if($i=="T=")print $(i+1)}' foo > foo1
awk '{for(i=1;i<=NF;i++)if($i=="Average")print $(i+2)}' foo > foo2
cat foo2 | awk '{ sum+=$1;print sum}' > summed
awk '{for(i=1;i<=NF;i++)if($i=="Nombre" && $(i+3)=="cellules,")print $(i+7)}' log_1
    > foo3
paste foo1 summed foo3 | column -s $'\t' -t > postProc.dat
sed -i '1s/^/0.00000E+00  0.000  3072\n/' postProc.dat
rm foo foo1 foo2 foo3 summed
```

The processed results can be displayed for example by `gnuplot` software. Furthermore, the other numerical results are outputted by BBAMR software in the form suitable for ParaView processing (Tecplot files).

4. OpenFOAM

In this section there will be provided short description of OpenFOAM solver used for numerical computation of two-phase VOF mathematical model based on the Navier Stokes equations for two incompressible, isothermal immiscible fluids. The OpenFOAM solver (`interIsoFoam`) solves the Navier Stokes equations for two incompressible, isothermal immiscible fluids. That means that the material properties are constant in the region filled by one of the two fluid except at the interphase. The term expressing dependency field of a mixture is similar as in the BBAMR case. But here the coloring function (denoted by $\alpha(\mathbf{x}, t)$) describes density field in exactly opposite way, i.e. $\alpha = 1$ where only water phase is present, $\alpha = 0$ for only air phase.

4.1 Governing equations

Governing equations of complete mathematical model in detailed form are presented hereby.

$$\nabla \cdot \mathbf{u} = 0 \tag{4.1}$$

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \mathbf{u}) = 0 \tag{4.2}$$

$$\rho = \alpha \rho_W + (1 - \alpha)\rho_A \tag{4.3}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \left[\mu\Big(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\Big)\right] - \mathbf{x} \cdot \mathbf{g} \nabla \rho - \sigma \kappa \nabla \alpha \tag{4.4}$$

Here $\mathbf{u}$ denotes velocity vector, α is phase fraction field (coloring function), ρ is density, p is pressure, μ represents kinematic viscosity, g denotes gravitational acceleration, σ is surface tension and $\kappa = -\nabla \cdot (\nabla \alpha |\nabla \alpha|^{-1})$ is surface curvature. Density field is represented in the same fashion as in the BBAMR software, by linear combination of water density and air density, ρ_W and ρ_A respectively.

In order to obtain proper coupling of velocity and pressure fields while solving these equations numerically, there is employed PISO algorithm where equations for velocity field are coupled with appropriately modified Poisson equation for pressure and then are solved simultaneously with the interface advection equation. For more details on implementation of PISO algorithm in OpenFOAM see e.g. [14].

4.2 Numerical schemes

There are two techniques handling with phase interface transport in OpenFOAM. First is called MULES - Multi-dimensional Universal Limiter for Explicit Solution and it uses advantage of modified advection term, in equation 4.1

$$\nabla \cdot (\alpha \mathbf{u}) \quad \xrightarrow{\text{modif.}} \quad \nabla \cdot (\alpha \mathbf{u}) + \nabla \cdot \Big(\alpha(1-\alpha)\mathbf{u}_r\Big)$$

where $\mathbf{u}_r = \mathbf{u}_{water} - \mathbf{u}_{air}$ is velocity vector of relative movement of phases. The additional term is called *compression term* and serves to compress/sharpen phase interface, smeared by numerical diffusion. This techinque is employed in **interFoam** solver. More detailed description can be found in e.g. [15], [16].

The second technique how to deal with the phase interface sharpening is based on explicit geometrical reconstruction and advection of the phase interface. At first the phase interface is reconstructed using volume fraction field and then, with using velocity field, the interface is advected through each computational cell. This all is implicitly coupled

with momentum equation through iteration loops ensuring conservation of mass. This technique is is employed in **interIsoFoam** solver and the algorithm itself is named as **isoAdvector**. For more information see original article [17] or tutorial paper [18].

The numerical schemes used for discretization of spatial terms in equations (4.2) and (4.4) solved within the PISO algorithm are specified in file `system/fvSchemes` which is enclosed in Appendix, here. All the spatial terms are of second order of accuracy. Integration in time is performed by Crank-Nicolson scheme of second order accuracy. More detailed description of used numerical schemes can be found in e.g. [14] or OpenFOAM code documentation, [19].

4.3 Setting the case

This section provide some more detailed description of setting up and running 2D dam-break with AMR computational case in OpenFOAM [20] in order to make it more accessible to person who is less familiar (or possibly not at all) with this software.

First of all, in the case of not having OpenFOAM installed, the user is pointed to follow the instructions on: `https://www.openfoam.com` or `https://www.openfoam.org`. Hereafter, there will be presented step-by-step setting in version OpenFOAM-v1812 in the rest of this section. The simplest way how to proceed with setting up the 2D dam-break case is to copy and modify an already existing tutorial case. This can be done in Linux terminal as follows.

```
1 mkdir $HOME/OpenFOAM/my_dam
2 cp -r $FOAM_TUTORIALS/multiphase/interIsoFoam/damBreak/
3       $HOME/OpenFOAM/my_dam
```

Now, we should have a new folder named `my_dam` and within it another folder, `damBreak`, copied from OpenFOAM tutorials.

Next, let us look onto default computational mesh in this tutorial case. The mesh itself is here created by using `blockMesh` utility and then can be displayed with ParaView visulization software, which can be compiled to cooperate with OpenFOAM as a third-party visualization software. By executing following commands in terminal

```
1 cd $HOME/OpenFOAM/my_dam/damBreak
2 blockMesh
3 paraFoam
```

we find out that our mesh deals with 2D dam-break problem **with obstacle** composed as structured, rectangular mesh, see Figure 9

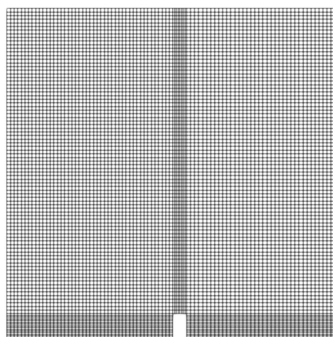

Figure 9: Structured, rectangular mesh created in OpenFOAM tutorial case - 2D dam-break problem with obstacle.

Note: If `paraFoam` command does not work and still ParaView is installed on your computer and works (can be launched by executing `paraview`) there exist a trick to load OpenFOAM case by creating empty file called e.g. damBreak.foam by

```
touch $HOME/OpenFOAM/my_dam/damBreak/damBreak.foam
```

and then open this file in ParaView. In order to create Cartesian mesh we have to modify source file `blockMeshDict`, needed by `blockMesh` utility to construct new mesh. The file is located at

```
cd $HOME/OpenFOAM/my_dam/damBreak/system
```

and after modification should read as described in Appendix, here. The structure and input definitions in `blockMeshDict` file can be found in various examples through scientific literature, code documentation and OpenFOAM tutorials, i.e. see [21],[22]. The output mesh **without obstacle** after re-running `blockMesh` command should now look like the one on Figure 10.

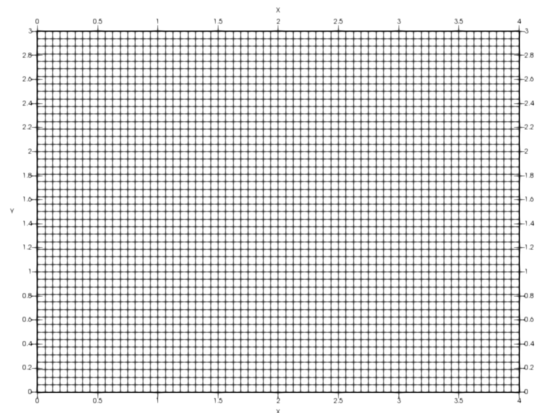

Figure 10: Structured, rectangular mesh for 2D dam-break problem.

Now, since the goal of the numerical computation is to test and compare adaptive mesh refinement (AMR) we proceed with setting up this feature. Unfortunately, there is implemented only AMR in 3D sense in OpenFOAM, which means that only Octree algorithm in 3D cases is supported (in version OpenFOAM-v1812). In order to run AMR in 2D there was used Quadtree mesh-refinement algorithm developed for OpenFOAM by Luca Cornolti [23], see also works [24]. It can be downloaded and compiled as described below.

```
cd $HOME/OpenFOAM/my_dam/damBreak
git clone https://github.com/krajit/dynamicRefine2DFvMesh.git
cd dynamicRefine2DFvMesh/src
wmake
```

Now, we should be able to use this AMR algorithm in our simulation. Moreover, if user is interested in AMR performance, few lines of code which output time consumption of the re-meshing procedure can be added to files `dynamicRefine2DFvMesh.H` and `dynamicRefine2DFvMesh.C` in folder `/src`. Changes in original files are marked by green color.

- `$HOME/OpenFOAM/my_dam/damBreak/dynamicRefine2DFvMesh/src/dynamicRefine2DFvMesh.H`

```
...
65  //- Protected cells (usually since not hexes)
66  PackedBoolList protectedCell\_;
67  //- Time spent performing interface advection
68  scalar remeshingTime_;
69
...
174 //- Update the mesh for both mesh motion and topology change
175 virtual bool update();
176 scalar remeshingTime() const
177 return remeshingTime_;
178
...
```

- `$HOME/OpenFOAM/my_dam/damBreak/dynamicRefine2DFvMesh/src/dynamicRefine2DFvMesh.C`

```
...
873  dynamicRefine2DFvMesh::dynamicRefine2DFvMesh(const IOobject& io)
874  :
875  dynamicFvMesh(io),
876  meshCutter_(*this),
877  dumpLevel_(false),
878  nRefinementIterations_(0),
879  protectedCell_(nCells(), 0),
880  remeshingTime_(0)
881
882  {
883
...
1030 bool dynamicRefine2DFvMesh::update()
1031 {
1032 //create time counter
1033 scalar remeshingTime_start = time().elapsedCpuTime();
1034
...
1237 topoChanging(hasChanged);
1238 //print out global nuber of cells after refienment/un-refinement
1239 Info << "GlobalCells " << globalData().nTotalCells() << endl;
1240 // compute time spent with adaptive mesh refinement
1241 remeshingTime_ += (time().elapsedCpuTime() - remeshingTime_start);
1242 Info << "Mesh refinement: time consumption = "
1243      << scalar( 100*remeshingTime_/( time().elapsedCpuTime() + SMALL ) )
1244      << "%" << endl;
1245
1246 return hasChanged;
...
```

And, if above proposed performance check modification is done, we need to recompile the 2D AMR code again.

```
1 wmake
```

Now we have to set some parameters of AMR in file `dynamicMeshDict`. So change folder using the terminal

```
cd $HOME/OpenFOAM/my_dam/damBreak/constant
```

and here edit aforementioned file as follows.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

//dynamicFvMesh   dynamicRefineFvMesh;
dynamicFvMesh   dynamicRefine2DFvMesh;

dynamicRefine2DFvMeshCoeffs{

nBufferLayersR  2;

// How often to refine
refineInterval  1;

// Field to be refinement on
field           alpha.water;

// Refine field inbetween lower..upper
lowerRefineLevel 0.1;
upperRefineLevel 0.9;

// If value < unrefineLevel unrefine
unrefineLevel   10;

// Have slower than 2:1 refinement
nBufferLayers   1;

// Refine cells only up to maxRefinement levels
maxRefinement   4;

// Stop refinement if maxCells reached
maxCells        150000;

// Flux field and corresponding velocity field. Fluxes on changed
// faces get recalculated by interpolating the velocity. Use 'none'
// on surfaceScalarFields that do not need to be reinterpolated.
correctFluxes
(
    (phi none)
    (nHatf none)
    (rhoPhi none)
    (alphaPhi none)
    (ghf none)
    (phi0 none)
    (dVf_ none)
);

// Write the refinement level as a volScalarField
dumpLevel       true;
};


// ************************************************************************* //
```

Coefficients in this file are of the following meaning

`refineInterval`	specifies how often mesh refinement should be performed
`field`	specifies which field the dictionary shall use to determine mesh refinements,fields can be scalar or vector, here is used coloring function named alpha.water, one field can be specified
`lowerRefineLevel`, `upperRefineLevel`	these specify the limits of when to trigger mesh refinement or coarsening
`unrefineLevel`	specifies the max number of times the cells can be coarsened, typically set to be a large number of levels
`maxRefinement`	maximum number of layers of refinement that a cell can experience
`nBufferLayers`	specifies how many layers the mesh must hold a cell size before proceeding to the next level of refinement (or coarsening)
`maxCells`	refinement will not exceed this maximum number of cells
`correctFluxes`	list of fields that require flux correction, for each pair, specify a flux field and corresponding velocity field
`dumpLevel`	writes the refinement level for each cell as a field

Description of above mentioned coefficients is taken from [25] where user can found more detailed explanation.

Also, some changes need to be made in solver settings. All of them are listed in Appendix. Precise meaning of parameters/coefficients in those files are out of the scope of this work and their meaning can be found in various OpenFOAM documentation, [19], [26].

Now we are ready to run the case. All the other dictionary files i.e. defining boundary conditions, physical and turbulence properties do not need to be modified. They should be already fine from tutorial setting. Executing the solver could be done either manually by executing commands

```
cd $HOME/OpenFOAM/my_dam/damBreak
cp -r 0.orig 0
blockMesh
decomposePar
setFields
mpirun -np 4 interIsoFoam > log
```

or by running shell scripts `./Allrun` or `./Allrun-parallel` also in `damBreak/` folder.

Note: In case of running on single core, skip `decomposePar` command and for solver executing just run

```
interIsoFoam > log
```

4.4 Post-processing the numerical results

Most of the post-processing procedure is generally done in some convenient visualization software along with extracting the necessary information from data files. There is provided interface for reading OpenFOAM data as done in Subsection 4.3. There also exist variety of useful tools, compiled with OpenFOAM, dealing with data extraction. These tools can be executed during the numerical computation as well as after it.

In order to compare numerical simulations of 2D dam-break cases between each other and also with experiment we need

I. evolution of number of cells in computational domain during time

II. relation between CPU time and physical time

III. qualitative comparison of free surface

IV. quantitative comparison of free surface

At first, the author did not found how to extract total number of cells by means of OpenFOAM tools, therefore there is used an advantage of additional output coded in source file `dynamicRefine2DFvMesh.C` in 4.3. For generating desired data file is then used simple bash script executed in `/damBreak` folder.

```
awk '{for(i=1;i<=NF;i++)if($i=="Time" && $(i+1)=="=")print $(i+2)}' \
log.interIsoFoam > foo1
awk '{for(i=1;i<=NF;i++)if($i=="GlobalCells")print $(i+1)}' \
log.interIsoFoam > foo2
paste foo1 foo2 | column -s $'\t' -t > globalCells.dat
rm foo1 foo2
```

Produced output, `globalCells.dat`, can be then displayed by i.e. `gnuplot`

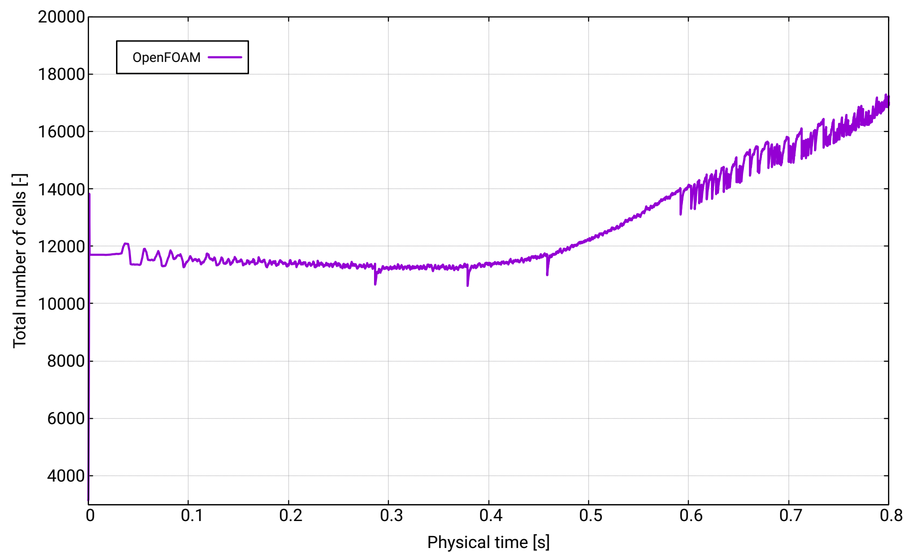

Figure 11: Evolution of number of computational cells through physical time.

Secondly, time evolution of numerical computation is obtained with help of OpenFOAM following command.

```
foamLog log.interIsoFoam
```

This produces couple of files consisting of solver performance data, e.g. residuals. The file of our interest is `logs/clockTime_0` which is displayed by `gnuplot` as follows.

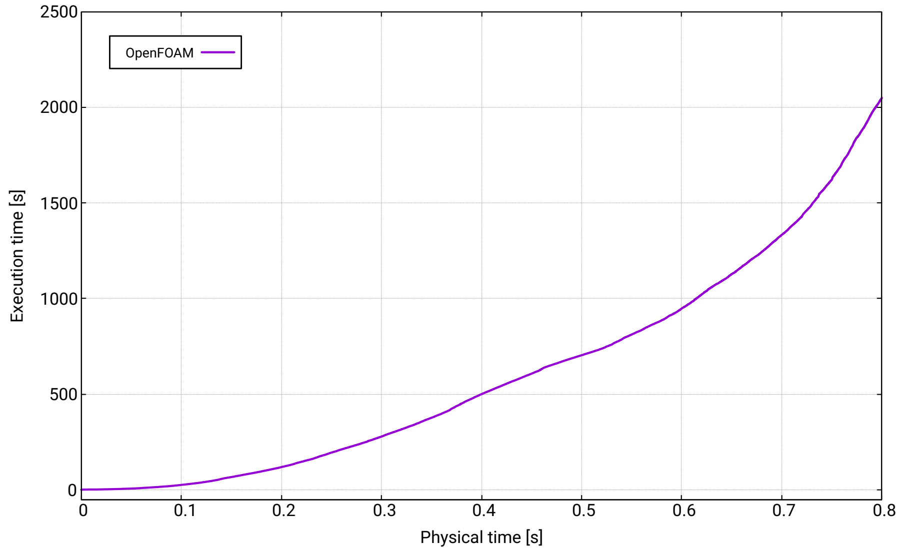

Figure 12: Dependency of execution time on physical time.

The last major task in our post-processing procedure is to compare free-surface shape and position throughout all three computational software. This is a task that could be done in ParaView and will not be explained here any further. Nevertheless, the output should by similar with following figures.

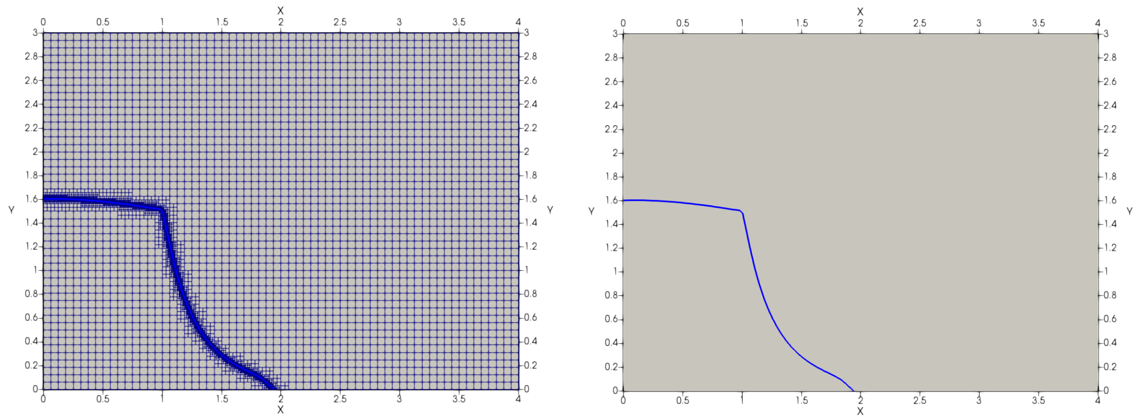

Figure 13: Pictures of free-surface contour of 2D dam-brak problem. With resp. without mesh cells on the left resp. right picture.

5. Basilisk

This section deals with numerical computation of 2D dam-break problem in software Basilisk which is free software program for the solution of partial differential equations on adaptive Cartesian meshes. Complete information about the software can be found on the home page http://basilisk.fr/. The unique feature of Basilisk is the adaptive mesh refinement algorithm based on wavelet-estimated discretization error which provide a generic approach to grid refinement based on estimation and reduction of numerical error [27]. Additional information on mesh refinement can be found in source code http://basilisk.fr/src/grid/tree-common.h#151. Major drawback of Basilisk software is that it supports only Cartesian meshes due to the nature of used AMR technique. Nevertheless there is a lot of effort made by developers to provide some programming techniques in order to overcome this (i.e. immersed boundary, fictious domain - Lagrange multipliers, etc.).

5.1 Governing equations and numerical schemes

The full form of governing equations for two-phase VOF model solved within Basilisk software is the same as in OpenFOAM. Here we recall the equations 4.1-4.4 again

$$\nabla \cdot \mathbf{u} = 0 \tag{5.1}$$

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha \mathbf{u}) = 0 \tag{5.2}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla p + \nabla \cdot \left[\mu\Big(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\Big)\right] - \mathbf{x} \cdot \mathbf{g} \nabla \rho - \sigma \kappa \nabla \alpha \tag{5.3}$$

In the Basilisk software, these equations are solved by using second order incremental approximate projection method on staggered grids, first introduced by Bell, Collela and Glaz in [28]. The phase interface advection equation 5.2 is solved by conservative VOF method for Cartesian grids described in [29]. This method efficiently reconstructs phase interface at any cell using only information of volume fraction function from directly neighbouring cells. The reconstruction is done by geometrical means, without any expensive iterative-base algorithms. Advection is then realized in sequential fashion in each direction with subsequently corrected interfaces. The method is of second order accuracy in space and time. Only restriction is here posed to Courant number.

$$\Delta t \left(\left|\frac{u}{\Delta x}\right| + \left|\frac{v}{\Delta y}\right| + \left|\frac{w}{\Delta z}\right| \right) < \frac{1}{2}$$

5.2 Setting the case

First of all, software Basilisk is quite easy to download and compile while users' system is equipped with all the necessary libraries, see http://basilisk.fr/src/INSTALL. Hence throughout this work is used common visualization software ParaView, user is encouraged to also download scripts for saving the computation results in VTK-format. The links to the codes are http://basilisk.fr/sandbox/cselcuk/output_vtu_foreach.h and http://basilisk.fr/sandbox/cselcuk/save_data.h

Once the Basilisk is installed one only needs to prepare C code for particular computational case. Considering the Basilisk is compiled in `$HOME/basilisk/src` user can make his/her own folder here

```
mkdir $HOME/basilisk/src/myCases
```

and here create file damBreak.c described in Appendix, here. Now if the file is prepared, we can compile and run the code in parallel (using 4 cores)

```
CC99='mpicc -std=c99' qcc -Wall -O2 -D_MPI=4 damBreak.c -o dam -lm
mpirun -np 4 ./dam
```

On our screen should appear window with performance statistics

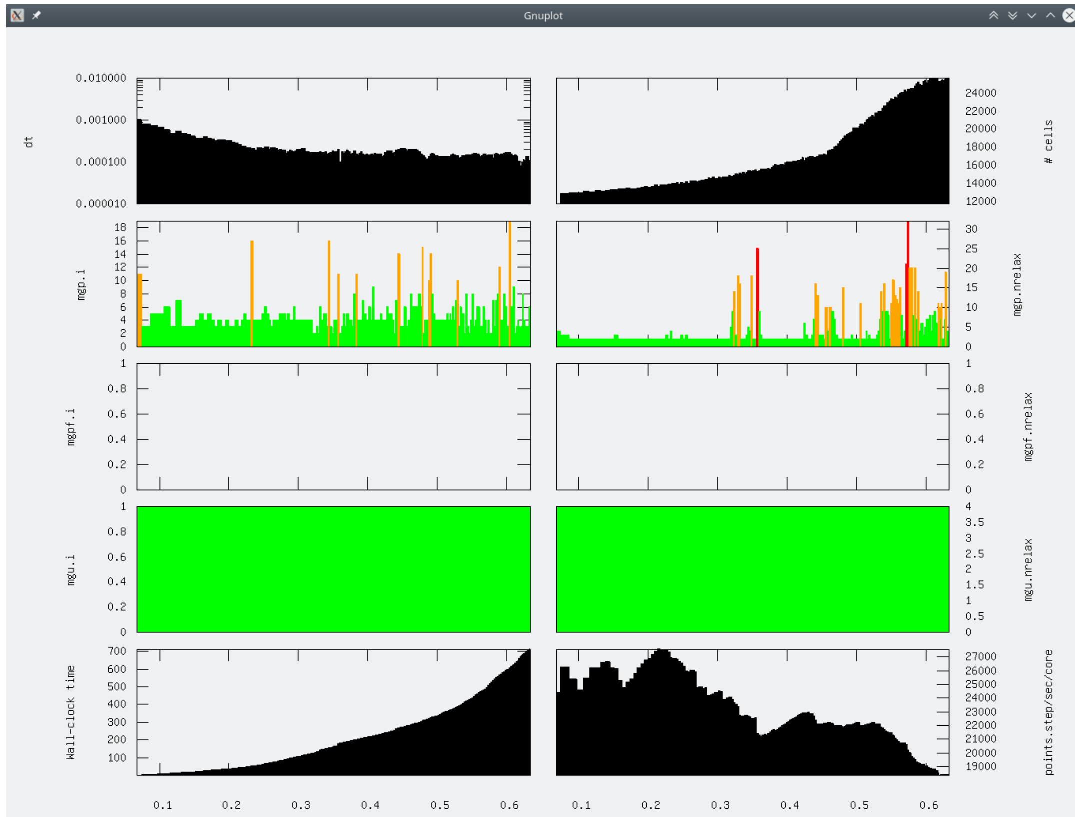

Figure 14: Performance statistics of Basilisk computation.

which will be also saved into file `prefs`, helpful for future analysis. According to the output VTK files, they can be loaded into ParaView as time series. Note that there need to be loaded `*.pvtu` files when running the case in parallel.

Results

The following section shows results of 2D dam-break problem computational cases solved with BBAMR, OpenFOAM and Basilisk software. Computation was performed in parallel mode, using 4 cores of Intel® Core i5-6300HQ processor.

On Figures 15 and 16 there are graphs comparing execution time with physical time and number of computational cells with physical time respectively.

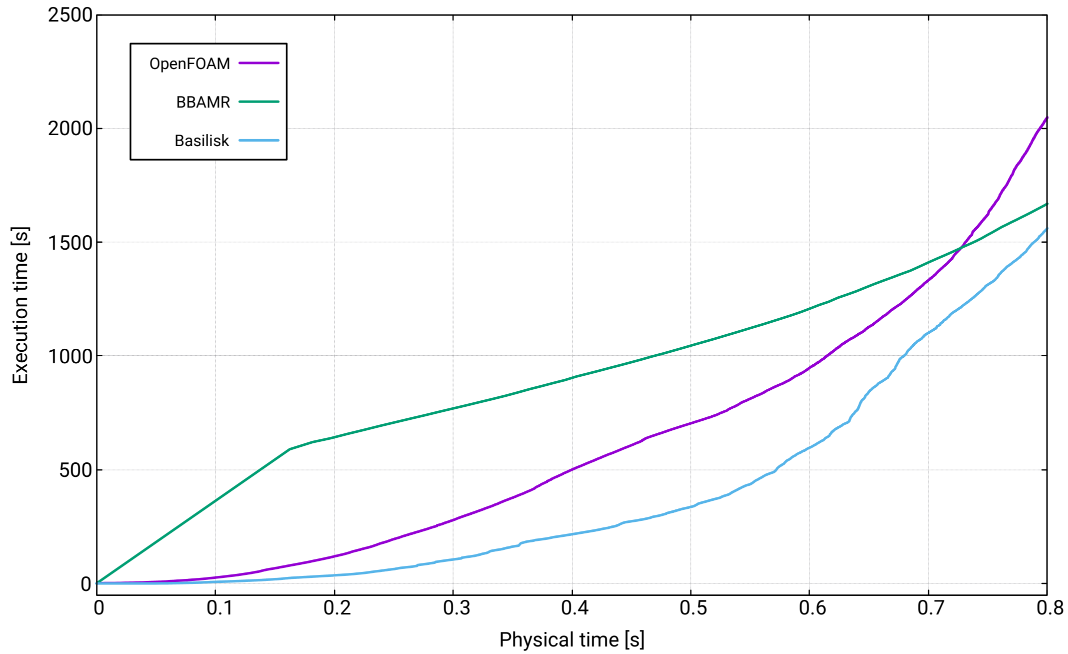

Figure 15: Comparison of execution times of used solvers.

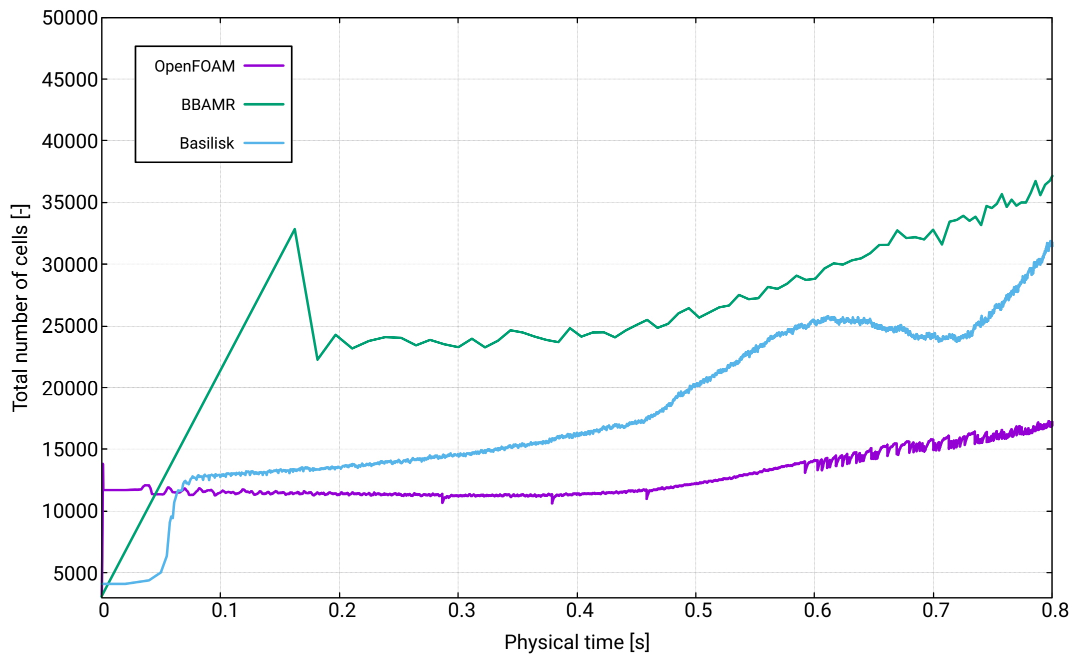

Figure 16: Comparison of total number of cells of used solvers.

In the interval $0 < T[s] < 0.2$ one can observe some discrepancies between total number of cells on Figure 16 which, in the case of BBAMR solver leads to worse performance in terms of execution time (Figure 15). This phenomenon is caused by less appropriate setting of initial refinement in BBAMR solver. The initial refinement covers wide area in the neighborhood of phase interface, which subsequently leads to computation on finer mesh for a longer time because the criterion for coarsening is not met at the start of computation but later, approximately in $T = 0.16s$.

In the case of Basilisk software, there is apparent greater growth of number of computational cells in $0.5 < T[s] < 0.6$ caused by creation and separation of water droplets in the front part of a moving wave, see also left picture of bottom row in Figure 18.

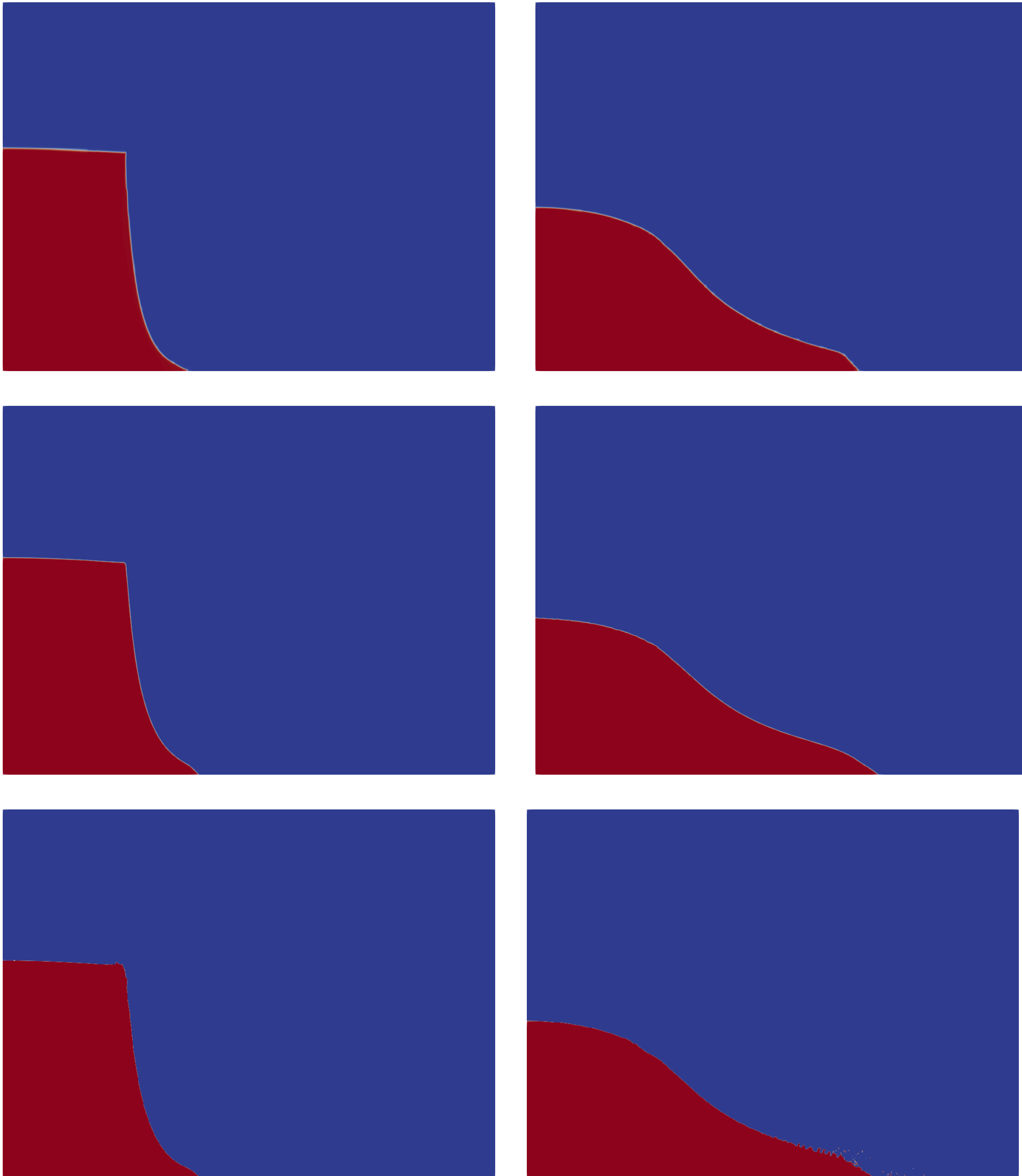

Figure 17: Pictures of density fields of 2D dam-break problem. From the top to the bottom, there are displayed BBAMR, OpenFOAM and Basilisk software. Left column represents time $T = 0.24s$, right column $T = 0.48s$.

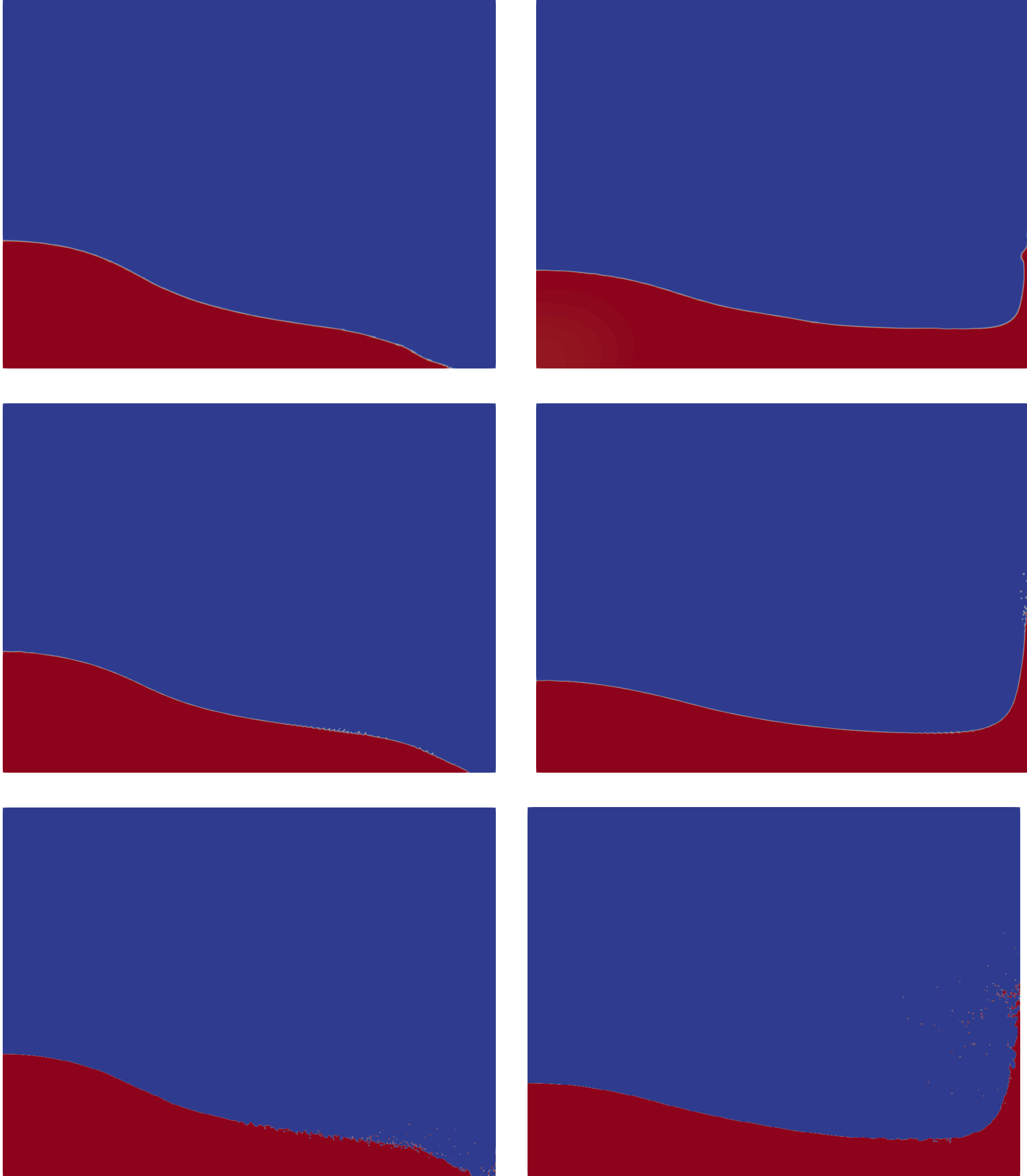

Figure 18: Pictures of density fields of 2D dam-break problem. From the top to the bottom, there are displayed BBAMR, OpenFOAM and Basilisk software. Left column represents time $T = 0.64s$, right column $T = 0.80s$.

The series of pictures on Figures 17 and 18 show several time snapshots of evolving density/coloring function for all the computational cases. All of the solvers tend to give the same qualitative solutions with respect to large scale dynamics, however on the smaller scale, they differ by the ability to capture the formation of water droplets on the free surface which are possibly created by instabilities generated by steep gradients of coloring function and velocity (figures of velocity fields are omitted here). This might be the effect of different (or absent, in the case of BBAMR) treatment of surface tension together with the ability of mesh adaptivity to follow the phase interface. Following the rows from top to bottom on Figure 18 one can observe there is a tendency of water phase to behave like viscosity is decreasing here.

Pictures on Figure 19 show evolution of computational mesh. Here is clearly visible that in the case of BBAMR solver (upper row) there are more cells situated near the phase interface (as mentioned above) which could increase computational cost significantly. However, due to its simplicity in numerical schemes (e.g. explicit time stepping) and hyperbolic-problems-built solver, BBAMR is still able to slightly outperform other two numerical solvers.

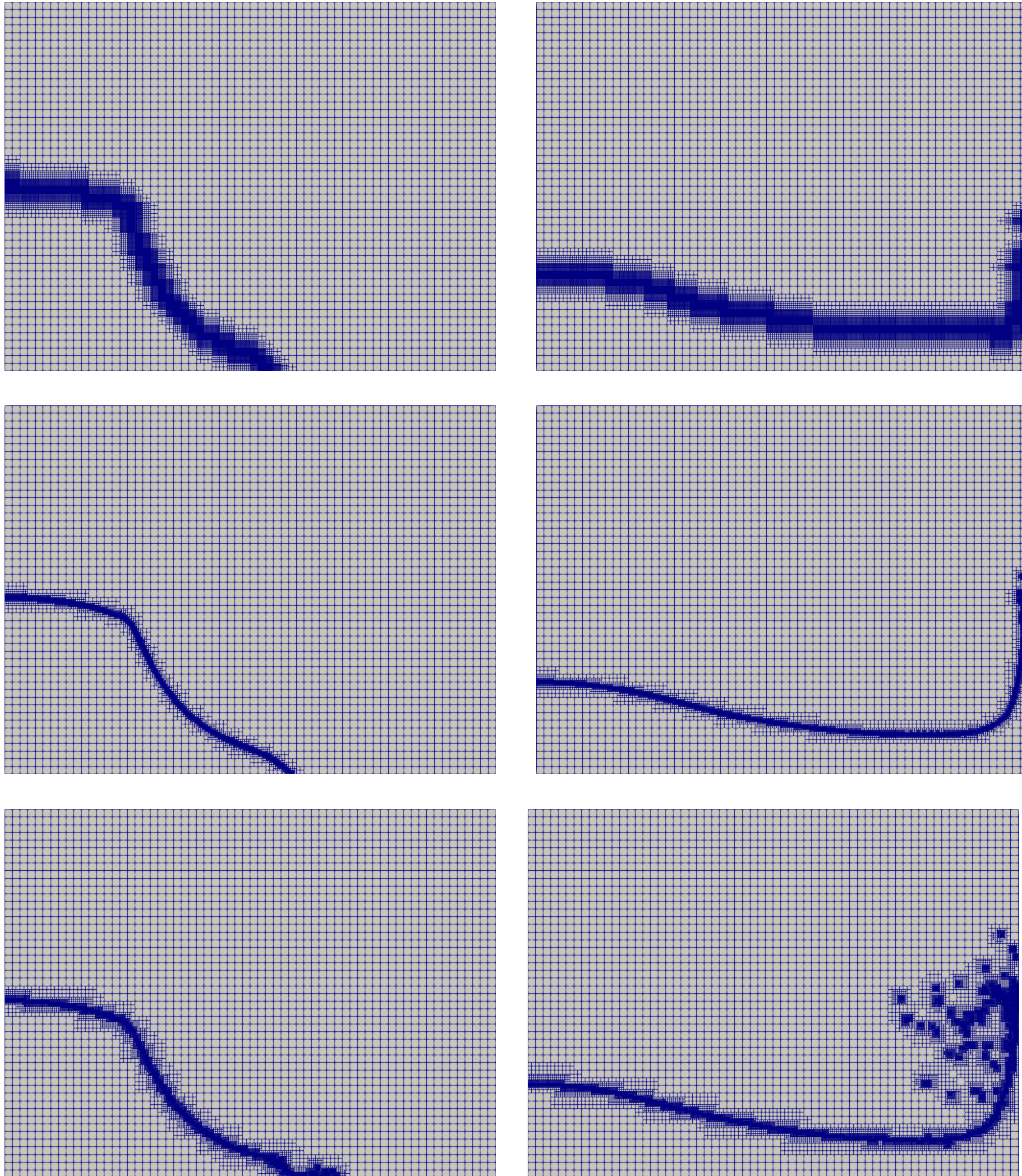

Figure 19: Pictures of adaptive mesh evolution through time. From the top to the bottom, there are displayed BBAMR, OpenFOAM and Basilisk software. Left column represents time $T = 0.40s$, right column $T = 0.80s$.

In the case of OpenFOAM (middle row) it appears that the free surface is followed by refined mesh quite well and only refined area is present in the vicinity of the free surface which leads to lowest amount of computational cells within the domain. On the other hand, the Basilisk (bottom row) is able to capture smaller scale dynamics very efficiently

and, in the final part of droplets presence (and thus more computational cells), the slope of computational time cost in Figue 15 seems to be on par with OpenFOAM solver.

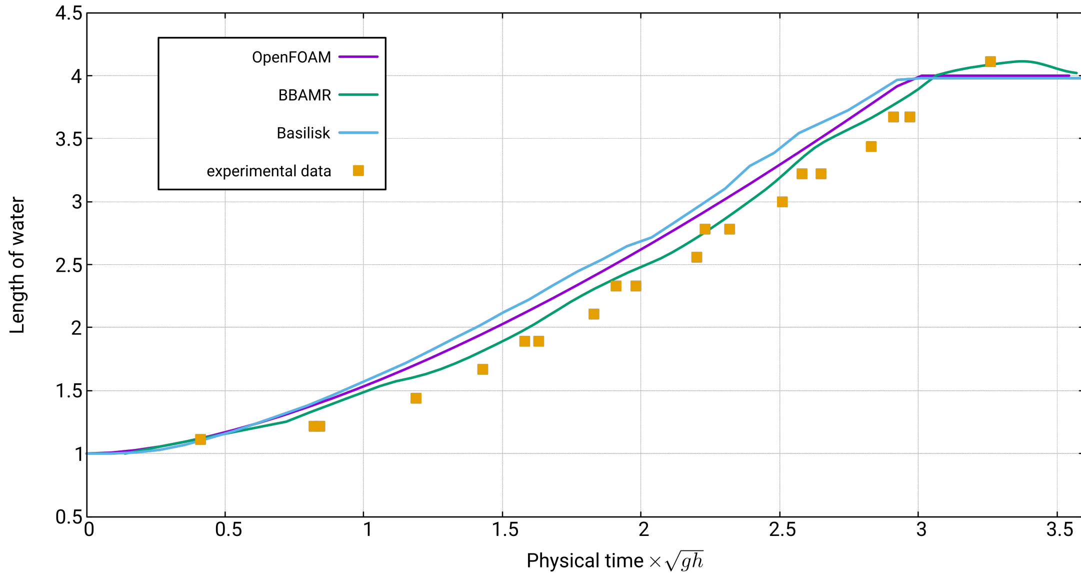

Figure 20: Length of water-wetted part of **bottom** wall during the water column collapse. Comparison with experimental data of Martin and Moyce [11].

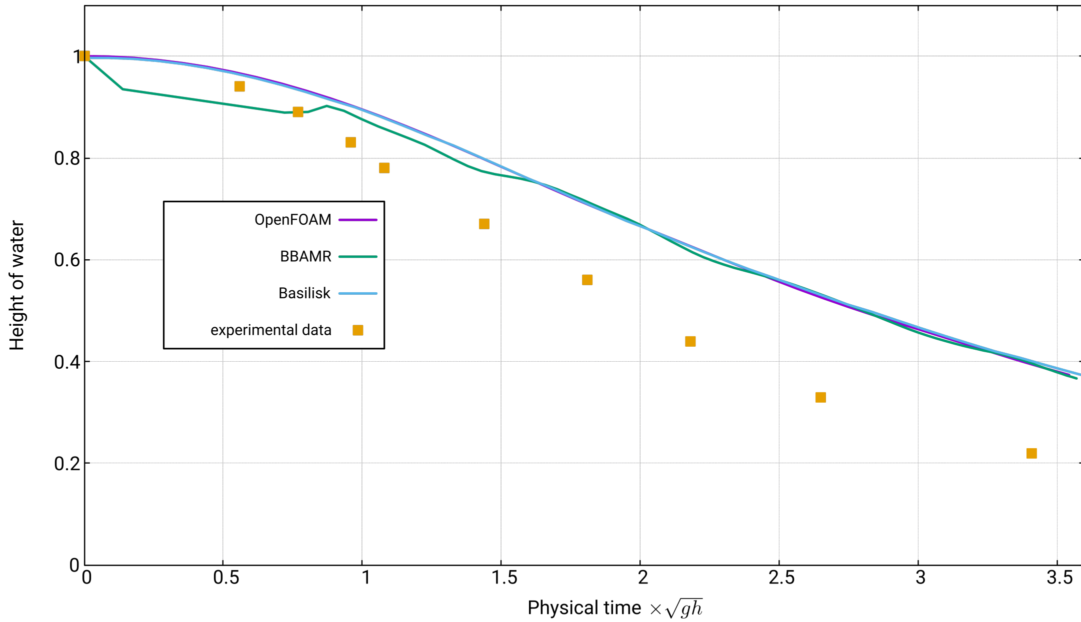

Figure 21: Length of water-wetted part of **left** wall during the water column collapse. Comparison with experimental data of Martin and Moyce [11].

The graphs on Figures 20 and 21 compare numerical data with experimental measurements of water-wetted part of bottom and left side of experimental apparatus respectively. The geometrical configuration of the apparatus is matching with computational domain in sense of dimensionless parameters. On Figure 20, the numerical data correspond acceptably with experiment however on Figure 21 one can detect considerable transition of the two sets of data.

Conclusion

In this work comparison of three different numerical software/solvers was done on the 2D dam-break case. The goal of the work was to study performance of numerical solvers with different implementation of adaptive mesh refinement techniques. Furthermore, description of settings of these numerical solvers is presented with aim to serve as tutorial for engineering students.

At first, a brief description of mesh refinement algorithms was done with particular focus on Quadtree algorithm commonly used in CFD software. Secondly, 2D dam-break computational case, used throughout the rest of the work, was described.

Afterwards, there follows the main part of the work which is organized as series of computational case descriptions regarding to the three numerical software, namely BBAMR, OpemFOAM and Basilisk. The case description is consists of governing equations, numerical schemes, settings of the solver and possibly post-processing.

The final part is devoted to comparison of results. There are compared relevant parameters of computation results such as evolution of number of computational cells, computational cost, qualitative character etc. Finally, confrontation of all the solvers with experimental results was carried out.

The results of numerical software comparison can be summarized as follows. The BBAMR software offers efficient and computationally cheap method for solving two phase flows with advantage of using block-based adaptive mesh refinement algorithm with light tree-structure. The solver is designed to solve only hyperbolic equations (at least for two-phase flows) where mesh-adapting criterion, the local numerical entropy production, naturally corresponds with this type of equations and provide more general option comparing to other physically based criteria, e.g. gradients of density, velocity, etc. The block-based algorithm usually reduce the computational time in contrast with cell-based one, nevertheless, the setting of initial mesh (blocks) and mesh adaptation criteria can be very problem-dependent. For example, when the cell refinement is prescribed to follow phase interface, the refined area can be wider due to the larger initial blocks. The hyperbolic solvers also deal with only inviscid flows. But the dynamics of the flow on larger scales is not affected much by viscosity and is captured well by BBAMR, plus, the viscous terms can be partly recovered as numerical viscosity of the chosen numerical scheme.

The OpenFOAM two-phase solver, interIsoFoam, equipped with 2D adaptive meshing algorithm, uses cell-to-cell connectivity data structure. This results in mesh adaptation more locally based than block-based adaptation, but with additional complexity in data tree. Considering the adaptation criterion based on phase interface, there are much less refined cells comparing to the BBAMR (Figure 16), and they are distributed only in the neighborhood of phase interface. However, the more the cell refinement is focused on the phase interface the more frequent re-meshing is needed in order to capture evolving free-surface. Since mesh adaptation in OpenFOAM is more time consuming than BBAMR, the goal here is find optimal amount of cells around interface. One of the solver's advantage is that it deals with viscous flows with the possibility of including surface tension term or various turbulence models.

The last tested software was Basilisk. There was used numerical solver for two-phase viscous fluids with option of surface tension. The mesh adaptation, based on quadtree data structure is here implemented very efficiently enabling to follow phase-interface to the relatively small flow structures (water droplets). Considering the complexity of flow of produced solution, the solver tends to outperform OpenFOAM and BBAMR in terms of computational time. The drawback of the solver is that only simple geometries are supported up to now and tools for dealing with more complex ones are still being developed.

References

[1] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, Mar 1974.

[2] M. Berg, de, O. Cheong, M.J. Kreveld, van, and M.H. Overmars. *Computational geometry : algorithms and applications*. Springer, Germany, 3rd ed edition, 2008.

[3] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129 – 147, 1982.

[4] Stéphane Popinet. A quadtree-adaptive multigrid solver for the serre–green–naghdi equations. *Journal of Computational Physics*, 302:336–358, 2015.

[5] Thomas Altazin, Mehmet Ersoy, Frédéric Golay, Damien Sous, and Lyudmyla Yushchenko. Numerical investigation of bb-amr scheme using entropy production as refinement criterion. *International Journal of Computational Fluid Dynamics*, 30(3):256–271, 2016.

[6] T. Gombosi, Darren Zeeuw, Kenneth Powell, Aaron Ridley, Igor Sokolov, Q. Stout, and Gábor Tóth. *Adaptive Mesh Refinement for Global Magnetohydrodynamic Simulation*, volume 615, pages 247–274. 01 2003.

[7] Jason ZX Zheng. *Block-based adaptive mesh refinement finite-volume scheme for hybrid multi-block meshes*. PhD thesis, 2012.

[8] Michael Williamschen. Parallel anisotropic block-based adaptive mesh refinement algorithm for three-dimensional flows. 2013.

[9] L. Freret, L. Ivan, Hans De Sterck, and Clinton P. T. Groth. High-order finite-volume method with block-based amr for magnetohydrodynamics flows. *Journal of Scientific Computing*, 79:176–208, 2019.

[10] Lucie Freret and Clinton P Groth. Anisotropic non-uniform block-based adaptive mesh refinement for three-dimensional inviscid and viscous flows. In *22nd AIAA Computational Fluid Dynamics Conference*, page 2613, 2015.

[11] John Christopher Martin, William James Moyce, JC Martin, WJ Moyce, William George Penney, AT Price, and CK Thornhill. Part iv. an experimental study of the collapse of liquid columns on a rigid horizontal plane. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 244(882):312–324, 1952.

[12] Frédéric Golay and Philippe Helluy. Numerical schemes for low mach wave breaking. *International Journal of Computational Fluid Dynamics*, 21(2):69–86, 2007.

[13] Mehmet Ersoy, Frédéric Golay, and Lyudmyla Yushchenko. Adaptive multiscale scheme based on numerical density of entropy production for conservation laws. *Open Mathematics*, 11(8):1392–1415, 2013.

[14] F Moukalled, L Mangani, M Darwish, et al. The finite volume method in computational fluid dynamics. *An advanced introduction with OpenFoam® and Matlab®. Nueva York: Springer. Recuperado de http://www. gidropraktikum. narod. ru/Moukalled-et-al-FVM-OpenFOAM-Matlab. pdf*, 2016.

[15] S Márquez Damián. An extended mixture model for the simultaneous treatment of short and long scale interfaces. *Doktorarbeit. Universidad Nacional Del Litoral. Facultad de Ingenieria y Ciencias Hidricas*, 2013.

[16] S Márquez Damián. Description and utilization of interfoam multiphase solver.

[17] Johan Roenby, Henrik Bredmose, and Hrvoje Jasak. A computational method for sharp interface advection. *Royal Society open science*, 3(11):160405, 2016.

[18] E. Olsson. A description of isoadvector - a numerical method for improved surface sharpness in two-phase flows. *In Proceedings of CFD with OpenSource Software, Edited by Nilsson. H.*, 2017.

[19] OpenCFD Ltd (ESI Group). Openfoam - documentation. 26.6. 2019. https://www.openfoam.com/documentation/.

[20] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Comput. Phys.*, 12(6):620–631, November 1998.

[21] Chris Greenshields. Openfoam v6 user guide: 5.3 mesh generation - blockmesh. 26.6. 2019. https://cfd.direct/openfoam/user-guide/v6-blockmesh/.

[22] OpenFOAMWiki. Blockmesh. 26.6. 2019. https://openfoamwiki.net/index.php/BlockMesh.

[23] Luca Cornolti. Adaptive mesh refinement in openfoam-v1812 for 2-dimensional problems. 26.6. 2019. https://github.com/krajit/dynamicRefine2DFvMesh.

[24] Jonas Karlsson. Implementing anisotropic adaptive mesh refinement in openfoam. Master's thesis, Chalmers University of Technology, University of Gothenburg, 2012. http://publications.lib.chalmers.se/records/fulltext/174173/174173.pdf.

[25] OpenFOAMWiki. Parameter definitions - dynamicrefinefvmesh - openfoamwiki. 26.6. 2019. https://openfoamwiki.net/index.php/Parameter_Definitions_-_dynamicRefineFvMesh.

[26] OpenCFD Ltd (ESI Group). Openfoam: Api guide: Openfoam : Open source cfd. 26.6. 2019. https://www.openfoam.com/documentation/guides/latest/api/.

[27] J. Antoon van Hooft, Stéphane Popinet, Chiel C. van Heerwaarden, Steven J. A. van der Linden, Stephan R. de Roode, and Bas J. H. van de Wiel. Towards adaptive grids for atmospheric boundary-layer simulations. *Boundary-Layer Meteorology*, 167(3):421–443, Jun 2018.

[28] John Bell, Phillip Colella, and Harland Glaz. A second-order projection method for the navier–stokes equations. *Journal of Computational Physics*, 85:257–283, 12 1989.

[29] Gabriel Weymouth and Dick K.-P. Yue. Conservative volume-of-fluid method for free-surface simulations on cartesian-grids. *J. Comput. Physics*, 229:2853–2865, 04 2010.

Appendix

There are listed source files which contain input parameters set in each particular simulation performed by the software.

BBAMR

Input file located in `$HOME/BBAMR/exec/damBreak.inp`. Description of some parameters is given in section 3.3.

```
!------------------------------------------------------------------------
! Data file in EOLENS format
! Each block of data is characterized in the first line by a header (4 characters)
! then a code (1 integer) and an argument (1 integer)
! format A4,2I5
! PHYS MESH INIT COND NUME
!------------------------------------------------------------------------

!------------------------------------------------------------------------
! Reading physical parameters
! Header PHYS
! 1 keyword (4 characters) and a real
! for now we have the same values for all domains
!------------------------------------------------------------------------
! MODE : physical model
!         0/ Euler
!         1/ Navier-Stokes
!         2/ K-epsilon
!         3/ isothermal two-phases
!         4/ two-phases 1 energy
!         5/ two-phases 2 energies
!         6/ granular
!        10/ Shallow water                              (default 0.)
! GPES : Gravity                                        (default 9.81)
! GAMA : Parameter gama of the pressure law             (default 1.1)
! PINF : Parameter Pinf of the pressure law             (default -0.9996363636D5)
! VIMU : Dynamical viscosity mu            (Pa s)       (default -0.890d-3)
! ZETA : Volumic viscosity zeta            (Pa s)       (default 0.)
! CSVC : Specific heat at constant volume (J/kg/K)      (default 4180.)
! TREF : Temperature of reference                       (default 299.9999130d0)
! CONT : Thermal conductivity (W/m K)                   (default 0.6071)
! CONV : Computation of convective terms 0/no 1/yes     (default 1.)
! PREF : isothermal pressure of reference               (default 1.d5)
! CSOA : Air sound velocity (isothermal)                (default 20.)
! CSOW : Water sound velocity (isothermal)              (default 20.)
! REFA : Isothermal density reference for Air           (default 1.)
! REFW : Isothermal density reference for Water         (default 1000.)
! PIFA : Parameter pinf for air                         (default 0.)
! PIFW : Parameter pinf for water                       (default 0.)
! GMAA : Parameter gamma for air                        (default 1.4)
! GMAW : Parameter gamma for water                      (default 1.1)
! CONS : non-conservative computation 0/cons 1/non-cons (default 0)
! SHAR : isothermal interface sharpening parameter
!        recommended value 0.1                          (defaut 0)
!------------------------------------------------------------------------
PHYS 0
MODE 3.
CONS 1
SHAR 0.1

!------------------------------------------------------------------------
! Reading the mesh
! Header  MESH
! code 1 -> creation of a mesh
!           argument: 0 type "1D shock tube"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
! next line: nb_dom,nx
!
!           argument: 1 type "2d box"
! next line: xmin,xmax,ymin,ymax,zmin,zmax
```

```
61 ! next line: nb_dom,nx,ny,iaxi
62 !
63 !           argument: 2 type "3d box"
64 ! next line: xmin,xmax,ymin,ymax,zmin,zmax
65 ! next line: nx,ny,nz
66 !
67 ! code 2 -> Reading an ascii file format Fluent 6
68 !           argument: Dimension of the problem
69 !                     0/axi, 1/1D, 2/2D, 3/3D
70 ! next line: file name
71 !------------------------------------------------------------------------
72 MESH    1    1
73 0. 4. 0. 3. -0.001 0.001
74 1 64 48 0
75 !------------------------------------------------------------------------
76 ! Lecture des conditions initiales (Si icode_MESH>0)
77 ! Entete INIT
78 ! code 0 -> Definition pour un tube a chocs
79 !           variables primitives a gauche rho,u,p si x<0 (3 reels)
80 !           variables primitives a droite rho,u,p si x>0 (3 reels)
81 !           argument: 0
82 ! code 1 -> Definition constante par domaine
83 !           argument: nombre de domaines
84 !           pour chaque domaine, on ecrit sur une ligne
85 !           nvar valeurs (variables primitives)
86 ! code 2 -> Lecture sur un fichier , nvar valeurs par cellule
87 !           argument: 0
88 ! code 3 -> Definition par une fonction utilisateur
89 !           (routine ../uti/fct_uti.f90)
90 !           argument: Numero de la fonction
91 ! code 4 -> Definition pour le BB-AMR
92 !           argument: Numero de la fonction utilisateur (si <0)
93 !                     (subroutine ../uti/amr_ini.f90)
94 !                     ou nombre de zones si argument >0
95 !           puis, si argument >0, pour chaque zone
96 !           xmin,xmax,ymin,ymax,zmin,zmax
97 !           nvar valeurs (variables primitives)
98 !------------------------------------------------------------------------
99 INIT    4    2
100 -1. 4.1 -1. 3.1 -1. 1.
101 1. 0. 0. 0.  1.e5 0.
102 -1. 1. -1. 2. -1. 1.
103 1000. 0. 0. 0.  1.e5 0.
104
105 !------------------------------------------------------------------------
106 ! Reading boundary condition (if icode_MESH>0)
107 ! (maximum 10 for the moment)
108 ! Header COND
109 !
110 ! code 0 -> Description in the case  "shock tube" : Nothing
111 !
112 ! code 1 -> Description in the case  "2d box" (at least 1 line per type) :
113 !           kind of boundary condition in xmin, xmax, ymin, ymax
114 !
115 ! code 2 -> Description in the case  "3d box" (at least 1 line per type) :
116 !           kind of boundary condition in xmin, xmax, ymin, ymax, zmin,zmax
117 !
118 ! code 3 -> Definition per zone for  Fluent mesh
119 !           argument: number of zones defining a boundary condition
120 !           number of zone, kind of boundary condition
121 !
122 ! kind of boundary condition:
123 ! 0 outlet (we copy)
124 ! 1 miror
125 ! 2 wall (in the case of vicous model)
126 ! 3 Dirichlet condition (if 1.e20 then copy), then next line nvar values
127 ! 4 user's function , then next line function number
128 ! 5 wall law (for turbulent flow)
129 ! 6 Periodicity (the second time, 3 real values defining the translation vector are
      added)
130 ! 7 Dirichlet condition on conservative variables  (if 1.e20 then copy),
131 !   then next line nvar values
132 ! 999 Non used
```

```
133 !------------------------------------------------------------------------
134 COND    1    0
135 1
136 1
137 1
138 1
139 !------------------------------------------------------------------------
140 ! Reading mesh parameters per Block AMR (BB-AMR)
141 ! in the case of BB-AMR, the block mesh is represented by th domain << 0 >>.
142 ! Header BLOC
143 !
144 ! code 0 -> default value
145 !
146 ! 1 keyword (4 characters) and a real
147 ! NBDS : number of domains                          (default 001.)
148 ! NRMA : Maximum refinement level                   (default 000.)
149 ! VCDE : Mesh coarsening parameter          0<..<1 (default 0.002)
150 ! VCRA : Mesh refinement parameter          0<..<1 (default 0.02)
151 ! TFIN : Final time of simulation
152 ! CCBL : CFL Condition on blocks before remeshing
153 ! FDRA : Function defining the mesh refinement to be applied
154 ! NZRA : Number of zones where initial blocks are refined (default 1.)
155 !        then for each zone
156 !        nrb,nx,ny,nz,xmin,xmax,ymin,ymax,zmin,zmax
157 !------------------------------------------------------------------------
158 BLOC 0
159 NBDS 4
160 NRMA #NRMA#
161 VCDE #VCDE#
162 VCRA #VCRA#
163 TFIN #TFIN#
164 CCBL .8
165 NZRA 3
166 0            1 1 1 -1. 4.10 -1. 3.10 -1. 1.
167 #NRMA_init# 1 1 1 -1. 1.21 -1. 2.21 -1. 1.
168 0            1 1 1 -1.  .79 -1. 1.79 -1. 1.
169 !------------------------------------------------------------------------
170 ! Reading numerical parameter
171 ! Header  NUME
172 ! 1 keyword (4 characters) then a real
173 ! code 0 -> default value
174 !
175 ! TMIN : lower bound of computation time            (default 0.)
176 ! TMAX : upper bound of computation time            (default 0.)
177 ! CCFL : cfl condition                              (default 0.9)
178 ! NPAS : time step number                           (default 0.)
179 ! FLUX : kind of numerical flux
180 !        0/rusanov
181 !        1/godunov
182 !        2/ Flux centered
183 !        3/ VF-Roe                                  (default 1.)
184 ! PRET : ordre of time discretization 1 ou 2        (defaut 1.)
185 ! PREE : ordre of space discretization 1 ou 2       (defaut 1.)
186 ! LIMI : kind of limiter  0/Barth, 1/Cartesian      (default 0.)
187 ! PART : Number of domains to be created (without AMR!!) (default 0.)
188 ! SAVE : kind of backup, three-digit number         (default 001.)
189 !        hundred : shock tube backup 0/no 1/yes
190 !        decade  : MEDIT backup      0/no 1/yes
191 !        unit    : binary backup     0/no 1/yes
192 ! SOND : number of probe  < 10                     (defaut 000.)
193 !        then coordinates x,y,z of the probe (1 probe per line)
194 ! METH : time integration method
195 !        1/ RK1 ou RK
196 !        2/ Adams Bashforth
197 !        3/ Adams Bashforth pas de temps local      (default 001.)
198 !------------------------------------------------------------------------
199 NUME 0
200 TMAX 0.1
201 TMIN 0.
202 TMAX 0.1
203 CCFL .5
204 PREE 2.
205 PRET 2.
```

```
206 LIMI 0.
207 METH 2
208 SAVE 001.
```

OpenFOAM

The location of OpenFOAM input files is hereafter referred to the case's main directory, `$HOME/OpenFOAM/my_dam/damBreak/`, meaning that e.g `system/blockMesh` file has following full path `$HOME/OpenFOAM/my_dam/damBreak/system/blockMesh`.

- `system/blockMeshDict`

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  v1812                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      blockMeshDict;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16 
17 scale   1;
18 
19 vertices
20 (
21     (0  0  0)
22     (4  0  0)
23     (4  3  0)
24     (0  3  0)
25     (0  0  1)
26     (4  0  1)
27     (4  3  1)
28     (0  3  1)
29 
30 );
31 
32 blocks
33 (
34     hex (0 1 2 3 4 5 6 7) (64 48 1) simpleGrading (1 1 1)
35 );
36 
37 edges
38 (
39 );
40 
41 boundary
42 (
43     atmosphere
44     {
45         type wall;
46         faces
47         (
48             (3 7 6 2)
49         );
50     }
51     leftWall
52     {
53         type wall;
54         faces
55         (
56             (0 4 7 3)
```

```
57            );
58        }
59        rightWall
60        {
61            type wall;
62            faces
63            (
64                (2 6 5 1)
65            );
66        }
67        lowerWall
68        {
69            type wall;
70            faces
71            (
72                (1 5 4 0)
73            );
74        }
75        frontAndBack
76        {
77            type empty;
78            faces
79            (
80                (0 3 2 1)
81                (4 5 6 7)
82            );
83        }
84    );
85
86    mergePatchPairs
87    (
88    );
89
90    // ************************************************************************* //
```

Some of the `blockMeshDict`parameters are described here.

`scale`	ascribes physical size in meters to distance defined by vertices
`vertices`	Cartesian coordinates of vertices (implicitly labeled from zero)
`blocks`	definition of blocks each consisting of 8 vertices, followed by number of cells in x,y,z directions and `simpleGrading` tells here to divide cells equidistantly in each direction
`edges`	don't need to be defined here (allows curved edges, etc.)
`boundary`	user named boundary patches with definition of type (no boundary conditions yet) and list of faces consisted of labels of vertices. Here `type empty` means that computation will be done in 2D sense (although mesh is 3D every time)

- `system/controlDict`

Note: In this file there need to be included line adding our newly compiled library - `libdynamicRefine2D.so`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

libs ("libdynamicRefine2D.so");

application     interIsoFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         1.0;

deltaT          0.0001;

writeControl    adjustableRunTime;

writeInterval   0.02;

purgeWrite      0;

writeFormat     ascii;

writePrecision  7;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           0.8;
maxAlphaCo      0.8;
maxDeltaT       0.1;


// ************************************************************************* //
```

- `system/fvSchemes`

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicolson 0.9;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    div(rhoPhi,U)   Gauss upwind;
    div(phi,alpha)  Gauss vanLeer;
    div(phirb,alpha) Gauss linear;
    div(phi,k)      Gauss upwind;
    div(phi,omega)  Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

wallDist
{
    method meshWave;
}


// ************************************************************************* //
```

- system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        isoFaceTol      1e-10;
        surfCellTol     1e-6;
        nAlphaBounds    3;
        snapTol         1e-12;
        clip            true;
        writeSurfCells  false;
        writeBoundedCells false;
        writeIsoFaces   false;

        nAlphaCorr      1;
        nAlphaSubCycles 2;
        cAlpha          1;
    }

    p_rgh
    {
        solver          GAMG;
        tolerance       1e-08;
        relTol          0.01;
        smoother        DIC;
        cacheAgglomeration no;
    }

    p_rghFinal
    {
        $p_rgh;
        relTol          0;
        tolerance       1e-9;
    }

    "pcorr.*"
    {
        $p_rghFinal;
        tolerance       1e-08;
    }

    U
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-07;
        relTol          0;
        nSweeps         1;
    }

}

PIMPLE
{
```

```
71      momentumPredictor no;
72      nCorrectors       3;
73      nNonOrthogonalCorrectors 0;
74      correctPhi        yes;
75
76      pRefPoint         (0.01 2.98 0.001);
77      pRefValue         0;
78  }
79
80
81  // ************************************************************************* //
```

- `system/decomposeParDict`

Note: Here we set parameters for parallel computation. Parameter **`coeffs`** here define splitting our computational domain according prescribed number of divisions. Method **`simple`** defines the division of domain will be done in equidistant fashion. The computation can be also still started on single core, without using this dictionary file.

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  v1812                                 |
5  |   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     location    "system";
14     object      decomposeParDict;
15 }
16 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
17
18 numberOfSubdomains  4;
19
20 method  simple;
21
22 coeffs
23 {
24     n   (2 2 1);
25 }
26
27 // ************************************************************************* //
```

- `system/setFieldsDict`

Note: In this file is set the initial field of coloring function - alpha.water as left-corner-situated $1m \times 2m$ block.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1812                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-100 -100 -100) (1.0 2.0  100);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
);


// ************************************************************************* //
```

Basilisk

Basilisk's input file.

- `$HOME/basilisk/src/myCases/damBreak.c`

```c
#include "common.h"
// 2D Cartesian grid
#include "grid/quadtree.h"
// embeded boundaries, in order to work
// with non-square geometry of computational grid
#include "embed.h"
// Navier-Stokes, VOF formulation
#include "navier-stokes/centered.h"
#include "two-phase.h"
#include "navier-stokes/conserving.h"
// surface tension
#include "tension.h"
// statistics of computation performance
#include "navier-stokes/perfs.h"
// compute with total pressure, p <-- p-rgh
#include "reduced.h"
// save data to VTK, for ParaView
#include "save_data.h"
#include "utils.h"

// maximal level of refinement
#define MAXLEVEL 10
// minimal level of refinement
#define MINLEVEL 6

double endTime = 1.0;

int main() {

    L0 = 4.;              // original domain length
                          // (now still square domain, i.e. 4x4 m)
    origin (0., 0.);
    N = 64;               // number of initial cells in x and y
                          // direction

    rho1 = 1000.;         // water density
    rho2 = 1.0;           // air   density
    mu1 = 1./1000.;       // water viscosity
    mu2 = 1.81/100000.;   // air   viscosity

    u.n[bottom] = dirichlet(0.); // full-slip boundary condition
    u.t[bottom] = neumann(0.);   // full-slip boundary condition
    u.n[embed] = neumann(0.);    // this replaces top boundary -->
    u.t[embed] = neumann(0.);    // --> homogeneous Neumann

    run();
}

event init (t = 0)
{
    // here is defined initial refinement of
    // initial free surface
    double Dref = 0.05;
    refine( ((x > 1-Dref/2. && x < 1+Dref/2. && y < 2+Dref/2.) \
    || (y > 2-Dref/2. && y < 2+Dref/2. && x < 1+Dref/2.)) && level<MAXLEVEL );

    // gravitational acceleration
    const face vector g[] = {0,-9.81};
    // all volumetric forces are denoted as -a- by default
    a = g;
    // alpha is equal to 1/rho
    alpha = alphav;

    // new field phi, needed only for masking-out
    // upper part of computational domain,
    // now it is supposed to be rectangular domain, 4x3m
```

```
67     vertex scalar phi[];
68     foreach_vertex()
69     {
70         phi[] = intersection (3.0 - y, HUGE);
71     }
72     boundary ({phi});
73     fractions (phi, cs, fs);
74 
75     // setting initial condition for coloring (phase fraction) function
76     foreach_vertex()
77     {
78         phi[] = min(2.0 - y, 1.0 - x);
79     }
80     fractions (phi, f);
81 
82     // initial condition for velocity (tiny movement of water column to right)
83     foreach()
84     {
85         u.x[] =    f[] * (1e-8);
86         u.y[] =    0;
87     }
88 
89 }
90 //-----------------------------------------------------------------------------//
91 // print some computation statistics into separate window
92 void mg_print (mgstats mg)
93 {
94     if (mg.i > 0 && mg.resa > 0.)
95         fprintf (stderr, "#   %d %g %g %g\n", mg.i, mg.resb, mg.resa,
96                 exp (log (mg.resb/mg.resa)/mg.i));
97 }
98 
99 
100 //-----------------------------------------------------------------------------//
101 // print some log in terminal
102 event logfile2 (i+=50)
103 {
104     if (i == 0)
105     fprintf (ferr,"t dt mgp.i mgpf.i mgu.i grid->tn perf.t perf.speed\n");
106     fprintf (ferr, "%g %g %d %d %d %ld %g %g\n",t, dt, mgp.i, mgpf.i, mgu.i, grid->
        tn, perf.t, perf.speed);
107 //     fprintf (ferr, "# refined %d cells, coarsened %d cells\n", s.nf, s.nc);
108 }
109 
110 //-----------------------------------------------------------------------------//
111 // print specified fields into VTK files
112 event data_out (t += 0.02; t <= endTime)
113 {
114     char name[80] = "damBreak_data/data";
115     scalar * list = {p, rho};
116     vector * vlist = {u};
117     save_data(list, vlist, i, t, name);
118 }
119 
120 //-----------------------------------------------------------------------------//
121 // adaptive mesh refinement
122 event adapt(i++){
123     scalar g[];
124     foreach()
125     {
126         // define field to be refined
127         // only in the neighbourhood of free surface
128         if( f[] > 0.05 && f[] < 0.95) { g[] = 1.0 + noise(); }
129         else {g[] = 0.;}
130     }
131     // update boundary conditions
132     boundary({g});
133     // define wavelet transformation criteria (lower and upper error tresholds)
134     adapt_wavelet({g},(double[]){0.001,0.01},minlevel = MINLEVEL,maxlevel =
        MAXLEVEL);
135 }
```