C. R. Mecanique 339 (2011) 194–207

Contents lists available at ScienceDirect

Comptes Rendus Mecanique

www.sciencedirect.com

High Performance Computing / Le Calcul Intensif

Parallel simulation of multiphase flows using octree adaptivity and the volume-of-fluid method

Simulation parallèle adaptative octree d'écoulements multiphasiques par suivi d'interface de type volume de fluide

Gilou Agbaglah[a,b], Sébastien Delaux[c], Daniel Fuster[a,b], Jérôme Hoepffner[a,b], Christophe Josserand[a,b], Stéphane Popinet[a,b,c], Pascal Ray[a,b], Ruben Scardovelli[d], Stéphane Zaleski[a,b,*]

[a] *Univ. Paris 06, UMR 7190, institut Jean-Le-Rond-d'Alembert, 75005 Paris, France*
[b] *CNRS, UMR 7190, institut Jean-Le-Rond-d'Alembert, 75005 Paris, France*
[c] *National Institute of Water and Atmospheric Research, Private Bag 14-901, Wellington, New Zealand*
[d] *DIENCA - Lab. di Montecuccolino, Università degli Studi di Bologna, Via dei Colli 16, 40136 Bologna, Italy*

A R T I C L E   I N F O

Article history:
Available online 14 January 2011

Keywords:
Computer science
Two-phase flow
Parallel computing
Octree
Volume-of-fluid
Droplet impact
Atomisation

Mots-clés :
Informatique
Algorithmique
Écoulement diphasique
Calcul parallèle
Octree
Volume de fluid
Impact de goutte
Atomisation

A B S T R A C T

We describe computations performed using the Gerris code, an open-source software implementing finite volume solvers on an octree adaptive grid together with a piecewise linear volume of fluid interface tracking method. The parallelisation of Gerris is achieved by domain decomposition. We show examples of the capabilities of Gerris on several types of problems. The impact of a droplet on a layer of the same liquid results in the formation of a thin air layer trapped between the droplet and the liquid layer that the adaptive refinement allows to capture. It is followed by the jetting of a thin corolla emerging from below the impacting droplet. The jet atomisation problem is another extremely challenging computational problem, in which a large number of small scales are generated. Finally we show an example of a turbulent jet computation in an equivalent resolution of 6×1024^3 cells. The jet simulation is based on the configuration of the Deepwater Horizon oil leak.

© 2010 Published by Elsevier Masson SAS on behalf of Académie des sciences.

R É S U M É

Nous décrivons des simulations réalisées avec le code Gerris, un logiciel libre qui implémente des méthodes de résolution de type volume fini sur un maillage adaptatif hiérarchique octree, et une méthode de suivi en volume avec construction d'interface affine par morceaux. La parallélisation de Gerris est obtenue par une décomposition en domaine. Nous montrons des exemples des capacités de cette approche sur plusieurs types de problèmes. L'impact d'un goutte sur une couche du même liquide provoque la formation d'une mince couche d'aire sous la goutte à l'instant de l'impact qui peut être capturée par la méthode adaptative. Il est suivi par le jaillissement d'une mince corolle par dessous la goutte impactante. Le problème de l'atomization est également un défi important pour le calcul intentsif, dans lequel un grand nombre de structures de petite échelle sont produites. Finalement nous montrons un exemple de simulation de jet turbulent avec une résolution

* Corresponding author at: CNRS, UMR 7190, institut Jean-Le-Rond-d'Alembert, 75005 Paris, France.
E-mail addresses: s.popinet@gmail.com (S. Popinet), zaleski@dalembert.upmc.fr (S. Zaleski).

1631-0721/$ – see front matter © 2010 Published by Elsevier Masson SAS on behalf of Académie des sciences.
doi:10.1016/j.crme.2010.12.006

équivalente de 6×1024^3 cellules. Cette configuration est basée sur celle de la fuite de pétrole brut de Deepwater Horizon.

1. Introduction

Fluid mechanics, and especially turbulent and multiphase flow, seem to require a never-ending wealth of computing resources. In many cases, however, regions of interest in a flow are localised in specific areas, such as boundary layers, turbulent jets and wakes, contact lines, or the high curvature regions of fluid interfaces. Moreover, in open flow problems, accuracy and physical realism often require the use of domains that are large compared to the typical scales of the flow. For example, to obtain the Stokes flow solution around a diameter-D sphere to 1% accuracy, a region of volume of order $10^6 D^3$ must be computed. Both phenomena strongly encourage investigators to opt for variable grid sizes that adapt to the problem to be solved. Indeed adaptive meshes can lead to several orders of magnitude gains in runtime depending on the problem [1,2] compared to uniform Cartesian grids. Unstructured grids made of cells of arbitrary polygonal shape are enjoying a wide popularity but an interesting alternative is the use of grids that retain some of the features of Cartesian grids. Researchers in multiphase flow often use block adaptive mesh refinement that superpose patches of Cartesian grids [3,4]. An alternative is the use of quad/octree grids. Quad/octrees are very simple, but flexible data structures which are well suited for efficient adaptive mesh generation (Fig. 1). They have been used for multiphase flow by several groups [5–7].

Among the advantages of octrees is the fact that an existing spatial discretisation can easily be refined or coarsened locally, in a robust manner and with very little computational overhead (typically less than a few percent of the total runtime when done at every timestep). Moreover quad/octrees directly include a multiscale (multigrid) representation of the solution which permits a very simple implementation of the multigrid methods, resulting in an efficient algorithm for the resolution of spatially-implicit problems (such as the Poisson and Helmholtz problems) [5]. Yet another advantage of octree grids is that they share with Cartesian grid a relative ease of implementation of modern interface-tracking methods such as high-accuracy piecewise-linear volume-of-fluid methods or level-set methods. There is in some cases no logical impossibility to implement these methods on unstructured grids with polygonal cells but some computations are more complex. In other cases, such as the use of the very efficient height-function methods in volume-of-fluid, the adaptation to unstructured grids is impossible.

In this article we attempt to discuss the potential use of such methods for high-performance computations. We thus describe some of the issues involved in the parallelisation of fluid solvers on octree grids in one particular example, the Gerris Flow solver [5,2]. We then describe results obtained in three typical problems. The droplet impact problem exemplifies the difficulties that arise where a very small region of the flow is of particular interest, such as the region of reconnection of the liquid masses. The atomisation simulations show another example that represents a difficult challenge, due to the formation of numerous regions containing small scales in the form of ligaments and droplets, and to the requirement of dealing with air–water properties to compare the results to experiment. We finally show a three-dimensional simulation of the gulf oil leak that is emblematic of the potential manifold usages of fluid-mechanical computations.

2. Numerical method and the Gerris code

The simulations presented below are made possible by the development and use of a specific code, Gerris [8], as well as continuous development of methods for direct two phase-flow simulation such as volume-of-fluid [9]. Gerris is a framework to solve partial differential equations adaptively on quad/octree meshes. It is freely available as open source software [8]. Gerris provides a range of finite-volume discretisation techniques which allow to solve: advection–diffusion equations, Poisson and Helmholtz equations and systems of conservation laws. When combined this allows the solution of a range of problems: Euler, Stokes, Navier–Stokes, Saint-Venant, electrohydrodynamics, etc. In what follows we show results where we resolve the Navier–Stokes equations for incompressible one or two-phase flow with constant surface tension. For the direct numerical simulation of multiphase flow our group has developed a range of methods of the volume-of-fluid type. The simulations reported in this paper use a volume-of-fluid method of the piecewise linear type, in which the interface segments are reconstructed using the mixed-Young's-centered (MYC) approximation [10]. While not the most accurate for very fine grids, the MYC approximation is easily implemented when using only information from the nearest-neighbour cells, an important advantage when domain-decomposition on octrees is used. Advection is performed using the "Lagrangian-explicit" method first published by Li [11] and discussed in [12,13]. While not volume-conserving to machine accuracy, it has very good volume and mass conservation properties.

Surface tension is a vexing question in multiphase flow. As density ratio and viscosity ratios become very large or very small, the simulations become increasingly difficult with standard methods [9]. The problem stems from an unbalance between surface tension and pressure forces that is resolved in so-called "balanced-force" algorithm [14].

The simulations presented in this article also use the multiphase capabilities in Gerris which include: geometric volume-of-fluid methods for interface tracking as well as accurate representation of surface tension terms using the height function method [2,1].

The main characteristic of Gerris is the use of quadtrees (octrees in 3D) for adaptive spatial discretisation.

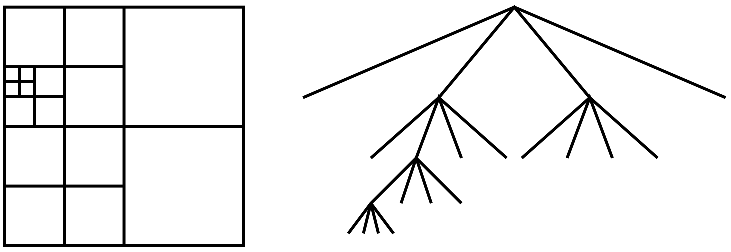
Fig. 1. Quadtree discretisation (left) and corresponding logical structure (right).

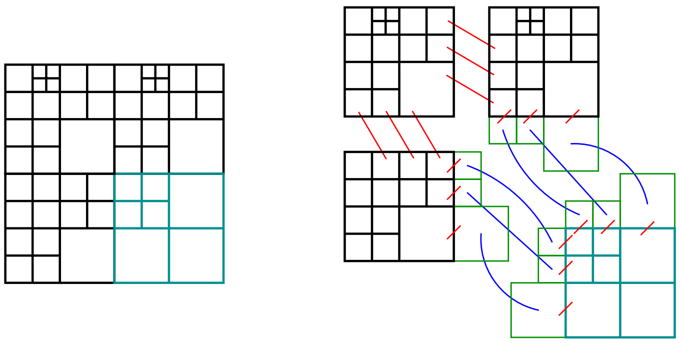
Fig. 2. Sketch of parallel domain decomposition on two processors within Gerris. Left: leaf cells coloured according to processor number. Right: corresponding structure. The four quadtrees making up the domain are shown, together with "stitches" (red), ghost cells (green), and parallel boundary conditions (blue).

3. Parallelisation and adaptivity

In this section we will describe how adaptivity and parallelism are combined within Gerris. Many algorithmic combinations are possible and the final choice involves a compromise between efficiency, degree of complexity and difficulty of implementation. The two primary issues to solve when performing this combination are: 1) preserving the simplicity and efficiency of the multilevel implicit solver in parallel, 2) insuring a balanced workload among processors when the number of cells varies during the simulation due to adaptivity (load-balancing).

3.1. Fine-grain versus coarse-grain parallelism

The natural option for parallelism in Gerris is through domain decomposition (rather than *procedural parallelism*): the global simulation mesh is split into as many subdomains as processors and each processor performs the same instructions as the others but only on its own subdomain (this is known as a Single Instruction, Multiple Data model of parallelism). Of course, the values of quantities required at the boundaries of the subdomains need to be synchronised when the inner values change. This involves communications between processors, which need to be minimised to obtain good parallel performance.

Fine-grain domain decomposition would consider boundaries between subdomains following the boundaries of cells on the finest level of the octree without restrictions. While this would allow complete flexibility in how the (finest) cells are distributed between processors, this would also lead to a complicated octree structure. For example children of non-leaf cells could belong to different processors, so that even a simple tree traversal would involve communications between processors (and thus parallel overhead). As described previously, the natural multilevel nature of the octree is very useful to easily implement multigrid methods and it also would be nice to be able to preserve this simplicity in parallel.

The option we have chosen during the initial design of Gerris, is to allow only *coarse-grain parallelism* where the smallest parallel subdomain is an entire quad/octree. The simulation domain is then tiled using multiple quad/octrees (i.e. a *forest* of octrees) which are connected through their common boundaries (Fig. 2). Two cases occur:

1. The two neighbouring quadtrees belong to the same processor: in that case they are "stitched" together simply by assigning the matching memory pointers in the fully-threaded-tree data structure. They in effect become a single data structure and all operations on quadtrees in the code are carried out as if a single quadtree were concerned.
2. The two neighbouring quadtrees belong to different processors: a "ghost layer" is created on the boundary of each quadtree. Ghost layers are used elsewhere in Gerris to define boundary conditions (such as symmetry, periodicity, etc.). They are implemented as a standard quadtree "stitched" to the boundary of the domain quadtree, but for which only the layer of cells in contact with the boundary is kept (i.e. the boundary quadtree is "flattened" in the direction

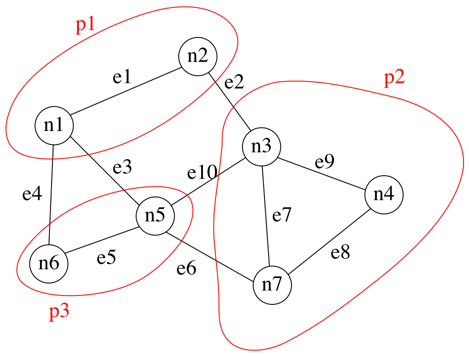

Fig. 3. Example of graph partitioning. Nodes and edges can have an associated weight. The subgraphs p1, p2, p3 are a partition of the initial graph. The goal is to obtain a balanced partition (each subgraphs have comparable total weight) while minimising the number (or total weight) of edges cut.

normal to the boundary). As in standard ghost layer implementations, discretisation operators carry out unmodified across the boundary and the appropriate boundary condition is defined by setting the values in the ghost cells. In the case which concerns us here (a "parallel boundary"), applying the boundary condition just means swapping the values in the ghost layers between the two processors sharing the boundary. Note that this is very similar to what is required for a *periodic* boundary condition, and indeed, within the object-oriented framework of Gerris, the parallel boundary (`GfsBoundaryMPI`) is a descendant of the periodic boundary (`GfsBoundaryPeriodic`) where swapping is performed using `MPI_Send` and `MPI_Recv` functions rather than simple memory assignments.

The main advantage of this approach is that parallelism just becomes another type of boundary condition. As all boundary conditions are applied as appropriate within the solution algorithm (i.e. whenever quantities have been updated in the bulk of the domain), consistency in the parallel case will automatically be guaranteed. From an implementation point of view, the serial and the parallel version of the code are strictly identical and the parallel code is restricted to the implementation of buffer swapping in `GfsBoundaryMPI`. This greatly simplifies the maintenance, portability and debugging of the parallel version.

A problem we have not discussed yet is how to perform the domain decomposition itself. While in some cases (such as illustrated in Fig. 2) the decomposition can be trivial, it is clear that the general problem of how to optimally partition a domain between p processors is non-trivial. A formal description of the problem is best articulated using the concept of *connected graph*. A graph is only concerned with the topological description of a set of *nodes* connected with *edges*. The nodes (and the edges) can also have associated weights to form a *weighted graph*. If we think of the nodes of the graph as (for example) our finest discretisation elements and of the edges as the connections (i.e. neighbouring relationship) between these elements, the domain decomposition problem is identical to a *partitioning* of the associated graph. The general goal is to partition the graph into p sub-graphs (of equal weight) while minimising the number (or weight) of edges being cut (Fig. 3). This will guarantee a balanced load between processors as well as minimising the cost of communications (i.e. boundary condition synchronisation) between processors. Graph partitioning is a difficult problem (it can be shown to be NP-complete) and polynomial-time (in p) algorithms are not known for the general case (nodes of degree larger than two), however several heuristic algorithms which work well in practice have been developed. To perform the initial partition, we use the generic graph partitioning algorithms which we have implemented in the GTS library [15] (note that many other libraries also implement graph partitioning algorithms such as METIS, JOSTLE, etc.). Because of our choice of coarse-grain parallelism, the nodes in our graph are entire quadtrees, their associated weight is the total number of cells they contain. The edges in the graph are the parallel boundaries (`GfsBoundaryMPI`). The associated weight is the number of neighbouring cells on the boundary. While several algorithms are available in GTS which can be used transparently, the quality of the initial partition is not very important in practice, since the partition usually quickly changes due to adaptivity and load-balancing (see the next section). We have found that a simple heuristic "bubble partitioning" algorithm gives good results in practice [16].

An important consequence of opting for coarse-grain parallelism is that to have a chance of obtaining a decent (load-balanced) partition of the domain between processors, the number of "grains" (i.e. connected quadtrees) needs to be much larger than the number of processors. By construction the coarsest level of approximation of the solution corresponds to a discretisation containing only the root cell of each quadtree in the forest. This will be the toplevel of the multigrid hierarchy used to solve linear problems (such as the Poisson equation required to enforce incompressibility). Thus, partitioning the domain on many processors will have the side effect of decreasing the depth of the traversal of the multigrid hierarchy: the size of the linear problem to be solved on the coarsest grid will thus increase. This could in turn affect the rate of convergence of the multigrid solver. In practice we have found that, provided at least 5 levels of coarsening are kept between the finest and the coarsest discretisation in the multigrid hierarchy, the convergence rate of the multigrid is not affected significantly. Maintaining 5 levels of coarsening is usually not a problem. For example a problem with 10 levels of refinement (i.e. 1024^3 equivalent maximum resolution) could be solved using a maximum of $(2^{10-5})^3 = 32\,768$ connected octrees. This would be suitable for load-balanced parallelism on, say, 1024 processors which would be reasonable given the size of the problem.

3.2. Load balancing

Combining adaptivity and parallelism makes the use of *load-balancing* techniques mandatory in order to maintain good parallel efficiency. Indeed, depending on the type of solutions sought, the number of discretisation elements can vary greatly during the course of a simulation. When nothing is done, even if the initial domain decomposition is optimal, the number of discretisation elements per processor will not remain identical between processors. In the worst cases this can lead to most processors waiting for a single processor (the most heavily-laden one) with parallel efficiencies dropping to zero. It is thus necessary to periodically *rebalance* the domain decomposition between processors.

At first, load-balancing could be seen as just another instance of the graph-partitioning problem which needs to be solved to obtain the initial domain decomposition. A simple way to perform load-balancing would then be to, for example: 1) save the simulation, 2) repartition it using the initial graph-partitioning algorithm, 3) restart the simulation. This approach has several drawbacks however. For example, the initial graph partitioning algorithm does not generally need to operate on the fully-resolved mesh: in the algorithm described in the previous section, the algorithm in effect operates on a much condensed version of the full graph. This means that graph partitioning can easily be done on a single processor, even for large problems (once fully discretized). In the case of load balancing, graph partitioning needs to be done frequently on larger graphs which makes parallel graph partitioning algorithms mandatory. These algorithms are often more complex than their serial counterparts.

Another specificity of load balancing compared to initial graph partitioning is that, if it is performed often enough, it should only require small changes to the partitioning to maintain good balance. This means that the graph partitioning technique needs to be able to make good use of a good *initial guess* for the optimal partition. Furthermore, once a better partition has been computed, rebalancing will require shifting discretisation elements between processors. This will have an associated communication cost which could be large if the better partition is very different from the initial partition. Minimising this additional parallel cost will also need to be taken into account when computing the optimised partition.

The load-balancing algorithm we have chosen for Gerris takes into account these imperatives: 1) given a good initial partition it converges quickly towards an improved partition, 2) it runs in parallel and minimises global parallel communications, 3) it minimises the changes required to obtain the improved partition. The algorithm is split into two sub-problems:

(i) given an initial weighted graph G, compute the *balancing fluxes* $F_{i,j}$ on the edges of G so that the weighted graph G' obtained when moving $F_{i,j}$ elements from node i to node j of G is better balanced (i.e. the weights of each node are close) than G. Do this while also minimising $\sum |F_{i,j}|$ i.e. the weight (i.e. total number of elements) which need to be moved between processors,

(ii) once the balancing fluxes are known, choose which elements to move between nodes so as to minimise communications between processors.

First of all note that the first subproblem is already a special case of the general graph partitioning problem: there is no reason *a priori* to restrict the transfer of elements to nodes sharing an edge, although it seems reasonable to do so if the initial graph is close to being balanced. With this restriction, the solution to subproblem one must verify for each i

$$\sum_{j,(i,j)\in E} F_{i,j} = w_i - \bar{w} \tag{1}$$

where E is the set of edges of G, w_i are the node weights (i.e. number of elements per node or load) and $\bar{w}$ is the average weight (i.e. the desired value for a balanced graph). Given that a connected graph always has more edges than vertices, the linear system defined by (1) is under-constrained (i.e. there are an infinitely many ways of exchanging weights between nodes in G which lead to a balanced graph). To constrain the system, we add the requirement that the solution must also minimise $\boldsymbol{F}^T\boldsymbol{F}$ i.e. the norm of the total weight displaced between nodes must also be minimised. This turns (1) into a classical constrained least-square optimisation problem which can be solved by introducing a Lagrange multiplier $\boldsymbol{\lambda}$ such that

$$\boldsymbol{\Delta}_G \boldsymbol{\lambda} = \boldsymbol{b} \tag{2}$$

with $b_i = w_i - \bar{w}$ and $\boldsymbol{\Delta}_G$ is the *Laplacian operator* on G such that

$$\Delta_G^{i,j} \equiv \begin{cases} -1 & \text{if } (i,j) \in E \\ 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ |E| & \text{if } i = j \end{cases}$$

It is then trivial to demonstrate that the solution of (1) which minimises $\boldsymbol{F}^T\boldsymbol{F}$ is

$$F_{i,j} = \lambda_i - \lambda_j$$

It is clear that this solution is closely related to the solution of a diffusion equation on graph G [17]. As noted by previous authors [18], the balancing fluxes F can be interpreted as the time-integrated value of diffusive fluxes over the course of relaxation toward the equilibrium state (for which all nodes have the same weight).

Computing the balancing fluxes thus requires the solution of the linear system (2). This system is definite positive (and also diagonally dominant) so that a wide range of iterative solution techniques could be used. As stated earlier, we would like to use a load-balancing algorithm which is itself parallelised. To reach this goal we simply use Jacobi iterations which are performed in parallel according to the following algorithm:

```
1. get the total number of elements $p\bar{w}$ where $p$ is the number of processors (requires global communication)
2. set $\lambda^0$ to zero
3. get the set $\mathcal{N}$ of neighbouring processors
4. Jacobi iteration
    i. for each processor $i$ in $\mathcal{N}$ send $\lambda^n$ and get $\lambda_i^n$ (local communication)
    ii. set $\lambda^{n+1}$ to $(w-\bar{w})/\sum\lambda_i^n$
    iii. if $|\lambda^{n+1}-\lambda^n| > \epsilon$ repeat 4
```

Note that only step 1 requires global communication so that the constrained least-square optimisation problem algorithm should easily scale to large numbers of processors. The convergence rate of Jacobi iterations could be an issue for large p, although this is helped in practice by the fact that the initial solution is close to convergence (i.e. the right-hand side in (1) is usually small). We have not encountered cases where more than 20 iterations where necessary to obtain convergence, even for $p > 1024$. The cost of solving (1) was thus negligible in most cases. A notable exception however was the case of weakly connected graphs, such as encountered when partitioning a long domain into a chain of subdomains (i.e. the degree of each graph vertex is two). In this particular case Jacobi iterations do not converge (note that Jacobi iterations are only guaranteed to converge for *strictly* diagonally-dominant systems which is not the case here): this is not very surprising given that the spectral radius of the Laplacian operator on a graph is closely related to the degree of the vertices [17]. This marginal case was corrected by using the recent modification of the standard Jacobi iteration of Johnson et al. [19] which guarantees convergence even for marginally diagonally-dominant systems.

Once the balancing fluxes $F_{i,j}$ are known, one needs to select which discretisation elements are going to be exchanged between processors to satisfy these fluxes. In our case, discretisation elements are entire quadtrees which may themselves contain a variable number of cells (i.e. have different weights). The problem is then to choose which quadtrees to exchange between pairs of neighbouring processors so that the balancing fluxes are most closely approximated. Several other constraints can be added such as: minimise the total length/area of the boundaries between processors (to try to minimise communication costs), minimise the aspect ratio of the subdomains (because compact subdomains will tend to have a smaller surface area, but also because this can improve multigrid convergence, etc.). This problem is analogous to a well-known problem in combinatorial optimisation called the *knapsack problem*: given a bag with a set carrying capacity and a set of p items of different weights, choose the subset of items which maximises the weight carried [20]. This problem is also NP-complete which indicates that a heuristic algorithm is likely to be the only practical solution (more so in our case where additional constraints are added). The simple heuristic algorithm we have chosen is a *greedy approximation algorithm* with the following properties: given a flux $F_{i,j}$ to satisfy,

1. the only quadtrees considered for transfer are those belonging to processor i and having a common parallel boundary with a quadtree on processor j,
2. of these quadtrees select by priority the quadtree which:
    a) has the maximum number of parallel boundaries in common with j,
    b) in case of a tie, select the quadtree whose weight is closest to $F_{i,j}$
3. subtract the weight of the chosen quadtree from $F_{i,j}$ and repeat for the remaining quadtrees.

Conditions 1) and 2.a) will tend to maintain compact subdomains and condition 2.b) will tend to minimise the number of quadtrees to transfer to achieve balance. While this algorithm could be improved, we have found that it performs quite well in practice.

An example of a parallel load-balanced adaptive simulation is given in Figs. 4 and 5. Fig. 4 depicts the atomisation of a pulsed liquid jet injected into a quiescent, lighter phase. The interface between the two phases is described using a geometric VOF method (the VOF interface reconstruction is used directly in Fig. 4 to display the interface); surface tension is added using an accurate, balanced CSF formulation [2]. The mesh is adapted according to the local vorticity and curvature of the interface. All the parameters necessary to reproduce this simulation are available on the Gerris web site [21]. We will not discuss here the physical results of this simulation as this has been the topic of several previous papers by our group [22,1]. The simulation is run in parallel on four processors on a workstation. In Fig. 4 the interface between the two phases is coloured according to which processor the corresponding cell belongs to. The "granularity" of the partitioning is apparent on the figure.

Fig. 5 illustrates the evolution with time of the number of cells per processor. Initially, only the area very close to the injection point needs to be resolved, so that the number of cells is small. The simulation domain is made of a $8 \times 8 \times 24$

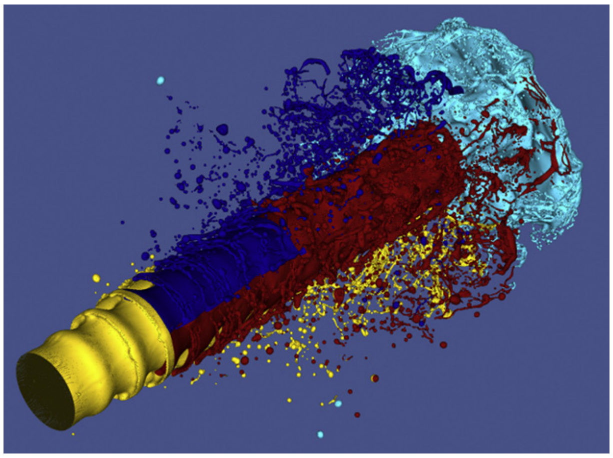

Fig. 4. Load-balanced domain decomposition for jet atomisation.

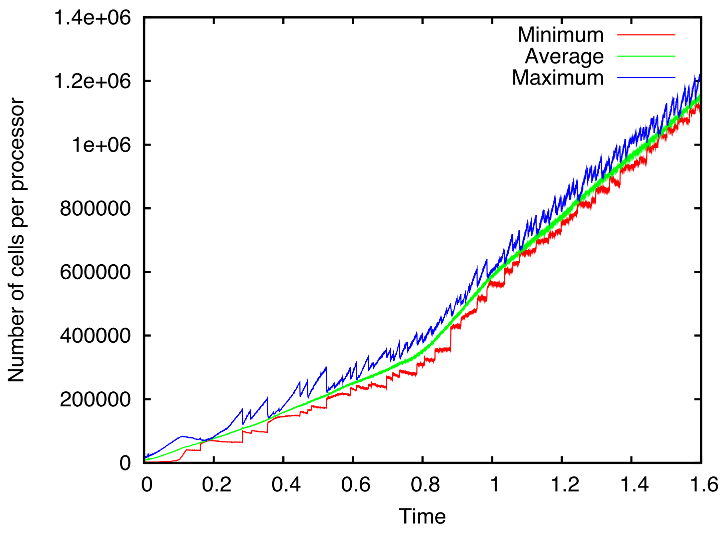

Fig. 5. Number of elements per processor as a function of time. Parallel adaptive simulation on four processors with load-balancing.

array of connected octrees. With this level of granularity, most of the cells are initially contained within a single octree (close to the injection point) so that load-balancing is not possible. This is reflected in Fig. 5 by the red curve staying close to zero for $t < 0.1$. As the simulation progresses, the total number of cells increases and good load-balancing can be maintained. Note that the curved is "jagged" rather than smooth. The main reason for this is that we chose to rebalance only when the difference between the minimum and maximum number of elements per processor exceeds 10% of the total number of cells. This allows to avoid "oscillatory" rebalancing which would not improve overall balance while degrading performance due to frequent communications between processors. Note that the initial peaks (up to $t = 0.6$) are due to the granularity constraint on achievable balance rather than to the 10% threshold. With this technique, the cost of load-balancing is negligible compared to the total simulation cost. Note also that while the simulation is relatively large (about 5 million cells overall at $t = 1.6$) it is much smaller than a non-adaptive uniform Cartesian grid with a comparable maximum resolution. Such a simulation would require about $3 \times 512^3 \approx 400$ million cells. The adaptive simulation runs in a few days on 4 processors of a standard workstation.

3.3. *Visualisation*

An important issue when running large parallel simulations is how to visualise (and/or post process) the results. Very often for large problems solved in parallel, the memory required to store the results is much larger than the total memory available on a single processor of the parallel machine or even that available on a powerful workstation. In that case visualisation of the results cannot be done using standard serial tools and must itself be done in parallel.

A related problem is that it is often useful to visualise the results during the simulation run itself, for example in order to check that a possibly-long simulation is proceeding as expected. This “online visualisation” can also be useful to generate animations which would be too expensive (in term of storage requirements) if done in a post-processing stage. This is not just for show: animations can be a very useful visualisation and interpretation tool (as demonstrated for example by the popularity of high-speed cameras in experimental physics).

Both these issues are addressed within GfsView, the visualisation tool associated with the Gerris solver. GfsView was designed to make the most of the quad/octree data structure. While this data structure was chosen primarily for its flexibility and efficiency in term of adaptive mesh refinement, it is also very useful to perform efficient multiscale visualisation of the results. For example “occlusion culling” i.e. removing parts of the scene which are outside the field-of-view can be performed very efficiently using a simple traversal of the octree. When this is combined with “level-of-detail” culling i.e. removing details which would be smaller than a single pixel, which can also be done very efficiently using the natural multilevel description encoded by the octree, this gives a very powerful multiscale interactive framework for visualisation. Note that more classical tools which are also freely available (e.g. Visit, Paraview, etc.) do not generally know about and take advantage of quad/octrees in this way.

GfsView was conceived from the start with a clear separation (separate libraries) between visualisation itself (performed using OpenGL commands coupled with the Gerris library) and user interface. As a result, it was simple to provide two separate interfaces: a standard interactive graphical user interface (with windows, buttons, mouse interaction etc.) and a “batch mode” interface which takes visualisation parameters defined using the graphical interface and renders (static) images/models in a variety of formats. This decoupling is very useful to be able to generate “online” visualisations/animations while the simulation runs, without any user input. Furthermore, most large parallel systems do not allow an interactive mode of operation, so that the only way to also use these systems for visualisation is through such a batch-mode interface.

Another important advantage of the decoupling between user-interface and visualisation is that this greatly facilitates the implementation of the *parallel rendering* necessary to be able to visualise large parallel simulations. Each processor generates the rendering for the entire field of view but only using the fraction of the simulation it owns. This is done locally, in parallel, using the standard OpenGL calls as implemented in the GfsView library. The result of this first phase is a set of p images which are encoded classically using (red, green, blue) triplets for each pixel of the image. An additional field is added which is the *depth* of the pixel: this value is provided directly by the so-called OpenGL *depth buffer*. The p fractional images are then gathered on a single processor to be merged into the single final image. This is done simply by setting the (red, green, blue) values of each pixel of the final image to the (r, g, b) value of the corresponding topmost pixel (according to its depth) of the p fractional images. While this image recomposition is not itself done in parallel, its cost is negligible compared to the rendering cost on each processor, so that the overall algorithm has an excellent parallel efficiency.

3.4. Scalability

Finally we conclude this section by an evaluation of the scalability of the algorithms we discussed on a large, modern supercomputer. The system we used is made of 32 IBM Power 575 supercomputing nodes with 32 4.7 GHz POWER6 processor cores per node (giving a total of 1024 processors). The benchmark case consists of running 100 timesteps of a 3D Navier–Stokes lid-driven cavity problem. As parallel efficiency is typically affected by problem size, three mesh sizes were considered: 32^3, 64^3 and 128^3. The number of processors used to solve the problem was varied from 1 to 512. Note that on 512 processors, for the smallest grid size, each processor would only use $32^3/51 = 64$ cells, while for the largest grid size this value would raise to 4096 (which is still a very small problem). Fig. 6 summarises the runtimes obtained as functions of the number of processors used and of problem size. The straight lines are obtained by least-square fitting in log–log space. The exponents are a measure of parallel efficiency. For all problem sizes and for up to 32 processors, the runtimes decrease close to linearly with the number of processors. This illustrates good initial scalability. For the smallest problem size (and in some respect also for the medium problem size), scalability deteriorates for large numbers of processors. For the largest problem size however, scalability is close to perfect up to 512 processors. As noted above, the smallest problem size is a very small problem to solve on 512 processors. As parallel efficiency can be expected to decrease linearly with decreasing spatial resolution, the fact that the runtime on 512 processors for the 32^3 problem is the lowest of all runtimes also reflects the overall excellent scalability of the algorithms (as well as the quality of the machine the problem is running on).

4. Droplet impact

In this section we investigate the emblematic question of the droplet impact which is ubiquitous in many problems involving strong deformation of liquid surfaces. Among numerous applications, droplet impact is crucial for ink-jet printing [23], combustion [24] and in environmental contexts such as for instance rain droplet erosion [25]. When the droplet velocity is high enough, the impact of a droplet leads to the formation of a very thin liquid jet and to the break-up of small daughter droplets. Fig. 7 shows the complexity of such dynamics in a 3D numerical simulation of a droplet impact on a thin liquid film using Gerris.

Considering incompressible and viscous fluids, droplet impact on a liquid surface is characterized by the liquid and surrounding-gas properties (densities ρ_l and ρ_g, dynamical viscosities μ_l and μ_g and surface tension γ), the droplet di-

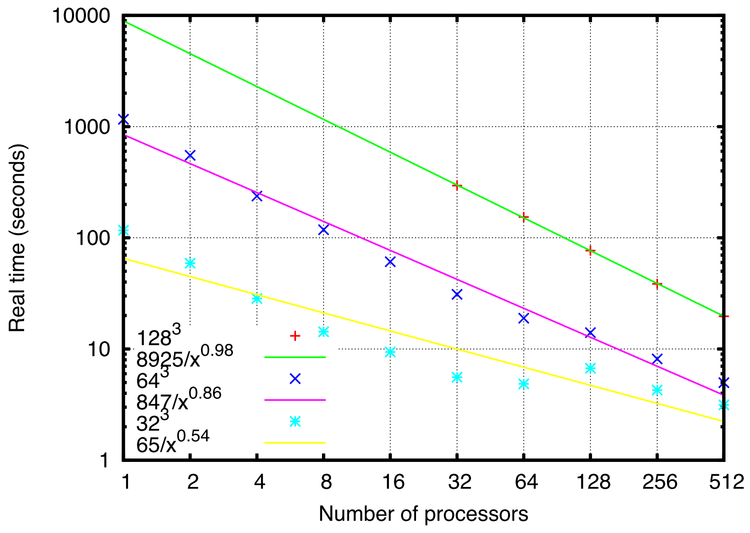

Fig. 6. Total computation time as a function of the number of processors for the lid-driven cavity parallel benchmark. IBM supercomputer.

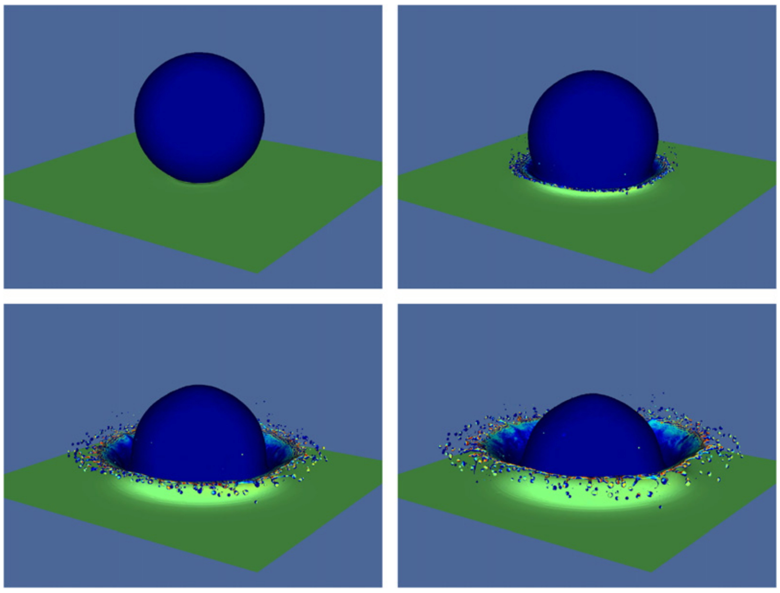

Fig. 7. Snapshot of a 3D numerical simulation of droplet impact on a thin liquid film using Gerris. The fluids in the film and in the droplet are the same but have been coloured differently in the simulation to illustrate their distribution along the dynamics.

ameter D and the liquid-layer thickness H and by the droplet U (one can neglect in general the gravity during the droplet impact dynamics). Therefore the droplet impact problem has five dimensionless parameters. Three of them are the density, viscosity and aspect ratios (ρ_l/ρ_g, μ_l/μ_g and D/H respectively). The last two dimensionless numbers correspond to the ration of inertia to viscous forces (Reynolds number) and inertia to capillary forces (Weber number):

$$\mathrm{Re} = \frac{\rho_l U D}{\mu_l} \quad \text{and} \quad \mathrm{We} = \frac{\rho_l U^2 D}{\gamma}$$

Further on, we address the different challenges of numerical simulation of droplet impact.

4.1. Numerical challenges

The main numerical challenges to face when computing droplet impact come from the formation of rapid and small structures. Such striking effects can be observed in Fig. 8, where the numerical simulation of a droplet impacting on a liquid film is shown in axisymmetric configuration. Notice that this formally 2D computation allows for a precision that cannot be

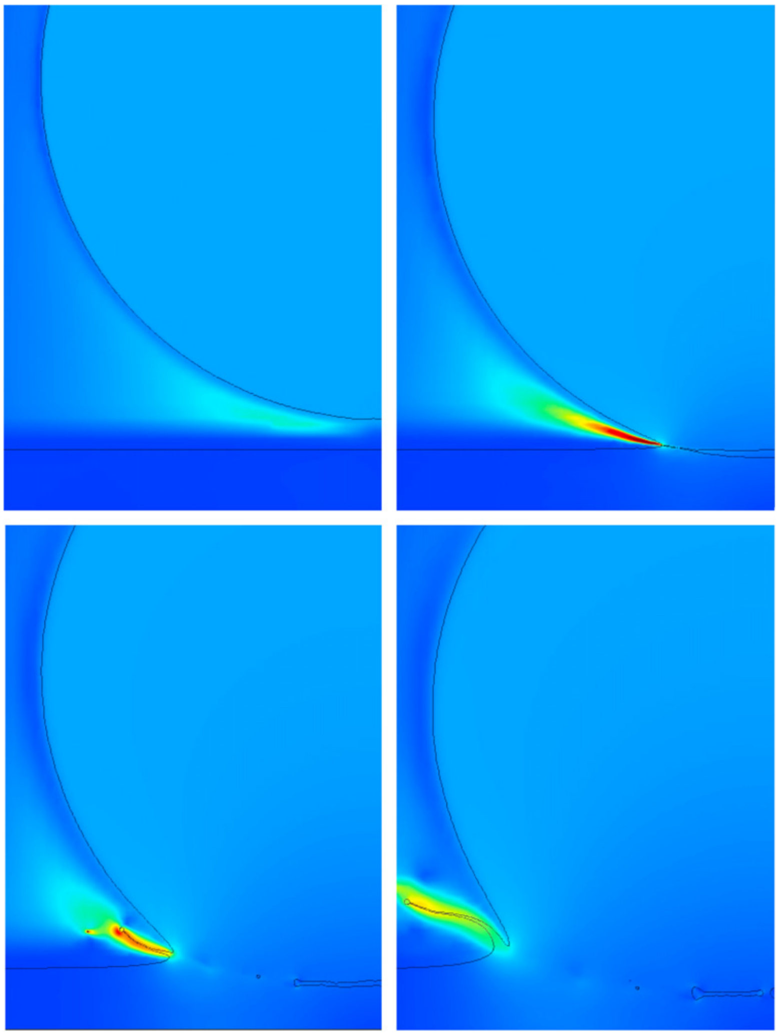

Fig. 8. Snapshot of 2D axisymmetric numerical simulation of a droplet impacting on a thin liquid film using Gerris. The colour map indicates the velocity norm.

reached nowadays in 3D calculations as the one shown above. Fig. 8 illustrates the tremendous complexity of such situations where various small-scale and rapidly deforming features appear simultaneously. Indeed, we first observe that the interface deforms beneath the droplet. This is due to the high pressure in the air layer below. The air layer further evolves to a very small thickness while the contact of the droplet with the liquid film is delayed. The connection between the droplet and the liquid layer is eventually made at a finite radius and a gas bubble is entrapped. Following this liquid merging, a very rapid and thin liquid jet is radially ejected. The jet eventually breaks into small droplets.

To assess quantitatively the numerical challenges, we will discuss below two characteristic features of droplet impact that need to be properly accounted for in the computations, namely the air cushioning layer and the liquid jet. One should however keep in mind that other numerical issues have to be considered: the thinning of the liquid jet as it expands and the droplet break-up from the expanding corolla to cite only two of the important effects.

4.2. Air cushioning beneath the droplet

The fact that the viscosity of the gas beneath the droplet becomes relevant at small time and space scales around impact is related to the experimental observation that a small gas bubble [26] is entrapped below the droplet. Numerically, entrapment of a gas bubble has been observed since the early two-phase impact simulations. However in these early simulations the radius of the gas bubble was of the order of the mesh size, and the bubble was under-resolved [27,28]. As shown on Fig. 8, one needs to resolve numerically the small thickness of the gas layer. For such a thin and elongated geometry, the gas dynamics are dominated by viscous forces and can be described by the lubrication equation where the film thickness $h(r,t)$ is coupled with the pressure in the gas layer $p(r,t)$

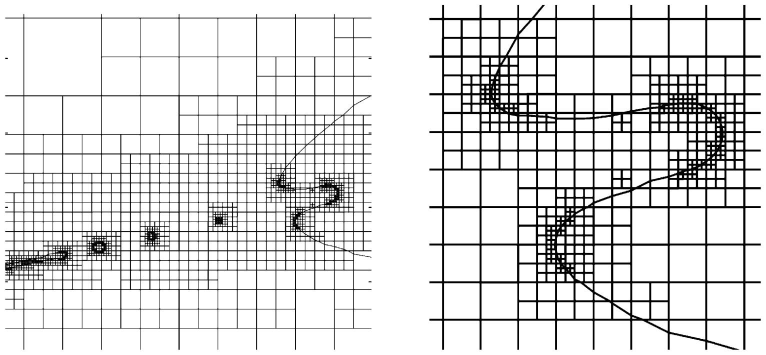

Fig. 9. Typical quad-tree grid used to describe the formation of the jet after the droplet has merged with the liquid layer. The criterion for refinement is the curvature of the interface.

$$\partial_t h = \frac{1}{12r\mu_g}\partial_r\left(rh^3\partial_r p\right)$$

Using this lubrication equation, one can estimate the typical gas layer thickness h_c due to air cushioning. Introducing the radial length $r_c = \sqrt{Dh_c}$, we obtain easily that the liquid pressure beneath the droplet is dominated by inertia effects:

$$\rho_l \partial_{t^2}^2 h \sim \rho_l \frac{U^2}{h_c} \sim \frac{p}{r_c}, \quad \text{giving } p \sim \rho_l U^2 \frac{r_c}{h_c}$$

Considering high Weber numbers so that we can neglect surface tension at this stage, the gas pressure is then the same than the liquid pressure at the interface and the lubrication equation gives:

$$U \sim \frac{h_c^3 p}{r_c^2 \mu_g} \sim U \cdot \frac{\rho_l U D}{\mu_g}\left(\frac{h_c}{D}\right)^{3/2}$$

from which we deduce the scaling for the typical gas layer [29], defining the Stokes number $\mathrm{St} = \frac{\mu_g}{\rho_l U D} = \frac{\mu_g}{\mu_l \mathrm{Re}}$:

$$h_c \sim \mathrm{St}^{2/3} D$$

To model correctly the gas cushioning during droplet impact, one needs to have the smaller grid size to be below or of the order of h_c. For instance in the case of a water droplet ($D \sim 2$ mm) in air, falling at $U \sim 5\ \mathrm{m\,s^{-1}}$ the ratio between the largest scale to describe (D) and the smallest one (h_c) is 10^{-4} which imposes 13 refinement levels at a minimum.

4.3. Jet formation

When the droplet and the liquid layer have connected, the large pressure peak in the reconnection zone leads to the formation of a thin liquid jet. This jet moves rapidly outwards. (See Fig. 9 for an example.) Again, to capture such fast and small-scale phenomena, the mesh needs to adapt to the smallest length scales: the thicknesses of the jet and gas layer, the radius of curvature of the interface and the curvature of streamlines caused by vorticity. As an example we show on Fig. 9 a case where the adaptation on the curvature is given a large weight. More accurate results can be obtained when more adaptation is done on the vorticity, but also at a higher computational cost. Of course it is not possible to capture the smallest length scales very close to the time of reconnection, as the curvature diverges at that point, but it is hoped that these length scales are captured a sufficiently short time after reconnection. It has been shown recently [28,30], that the jet emerges as a balance between the high pressure due to the impact with the viscous forces inside the jet, leading to a typical jet thickness l_c scaling like:

$$l_c \sim \sqrt{\frac{\mu_l t_c}{\rho_l}}$$

where t_c is the delay time of impact due to the air cushioning. One can in fact deduce from the previous estimate this time and the radius r_c at which the droplet and the liquid layer connect, entrapping an air bubble:

$$t_c \sim \frac{h_c}{U} \sim \frac{\mathrm{St}^{2/3} D}{U} \quad \text{and} \quad r_c \sim \sqrt{Dh_c} \sim \mathrm{St}^{1/3} D$$

Finally, we obtain the following scaling for the minimal liquid jet thickness to model:

$$l_c \sim D \cdot \mathrm{Re}^{-1/2} \cdot \mathrm{St}^{1/3}$$

which gives again $l_c/D \sim 10^{-4}$ for the typical water–air case and a minimum number of refinement levels of thirteen.

Table 1
Simulation conditions for the analysis of atomisation processes.

	U (m/s)	ρ (kg/m^3)	μ (Pa s)	σ (N/m)
Air	20	1.2	$1.7 \cdot 10^{-5}$	0.069
Water	0.173	1000	10^{-3}	

4.4. Discussion

We have shown using scaling arguments that the apparently simple case of droplet impact is actually very demanding in terms of mesh refinement. These fine meshes are necessary to achieve a correct description of the thin air and liquid layer formed by the impact. In axisymmetric coordinates, such precision can reasonably be reached using quadtree adaptive mesh refinement while an equivalent uniform Cartesian grid would be already at the limit of the capabilities of a single workstation. In three dimensions, such refinement will be done using parallel computations and efficient load balancing. Load balancing will be particularly difficult to achieve since the region containing small scales — the high refinement region — is very small compared to the size of the simulation domain. Thus in order to have a sufficiently large number of "grains" or octree in the high refinement region a very large number of grains will be needed overall. We thus encounter in a pronounced way the issue of high granularity discussed in Section 3.1.

5. Atomisation

The use of quadtree and octree adaptive mesh refinement (AMR) in conjunction with the robust Navier–Stokes solver and numerical methods for interfaces implemented in Gerris has recently made it possible to perform simulations in conditions typically used in atomisation laboratory experiments. The large density and viscosity ratios, the large momentum ratio between the gas and the liquid, and the multiple scales playing a rôle in the process makes this type of simulations a challenging problem to test the capabilities of a code able to handle multiple phases. As an example of these capabilities, typical conditions used in the experiments performed by Ben Rayana [31] have been tested. The most relevant parameters are given in Table 1. For more details, the reader is referred to [1]. A visualisation of this two-dimensional simulation may be obtained from http://www.lmm.jussieu.fr/~zaleski/Movies/fancyjet.mov. While a two-dimensional simulation allows us to have a much larger degree of accuracy even with a given computational cost, it is also justified in the particular experimental setup. Indeed in the experiment a planar sheet is produced from an elongated rectangular nozzle. Experimental observations near the nozzle show noisy two-dimensional waves. Further downstream, the large scale structures can still be approximated by two-dimensional rollers.

This example is especially interesting due to the disagreement found between the experiments carried out by Ben Rayana [31] and the theoretical predictions provided by the linear theory for viscous flows [32]. A factor of three difference is found, with the experimentally observed wavelength three times larger than the theoretical one. An alternative theory by the authors of Ref. [33] gives a factor of three error in the other direction. We note that in [31] the analysis of the error has been improved compared to the relatively crude analysis in [32]. A possible reason of the discrepancies between the theory and experiments can be found when the mechanism inducing the wave in the system is contemplated. Whereas the linear theory assumes that the flow perturbations are very small near the injection region, and growing exponentially as they propagate downstream, the simulations reported in Fuster et al. [1] show how a large amplitude wave is generated immediately behind the injector and is then advected downstream.

Fig. 10 depicts the void fraction field in the whole simulation domain. As the simulation is aimed at capturing the apparition of waves in the primary atomisation zone, a large level of refinement is applied near the injector (with a minimum cell size of 10 μm), whereas further downstream the level of refinement is decreased. The use of combined criteria, on curvature and vorticity for the quad-tree Adaptive Mesh Refinement allows to capture accurately the evolution of the instabilities at the interface as well as the flow patterns in the gas (Fig. 11). The main mechanisms leading to the appearance of waves in the primary atomisation region are captured and the frequency of the wave generation in this region is in quantitative agreement, within 20% with the experimental observations of Ben Rayana [31]. While this does not constitute a definitive cross-verification of experiment and theory/numerics, it is an important step in that direction.

6. Turbulent jet: oil leak in the Gulf of Mexico

On April 20th, 2010, the explosion and sinking of the Deepwater Horizon oil rig in the Gulf of Mexico started a massive oil leak that could not be controlled before several months had passed. Understanding the turbulent flow is of help in estimating the flow rate, and thus, the amount of oil released in the ocean [34]. The dynamics of the plume generated by a small horizontal tube in the ocean is a challenging problem for the numerical simulation as widely different spatial scales are involved in the process. On one side, the high-Reynolds-number turbulence of the flow at the large velocities encountered at the exit of the tube induces very small structures in the plume. On another side, a characteristic length of the order of the radius of curvature of the plume is important in order to relate, for example, this parameter with the outgoing flow. Thus, the simulation of these type of problems can be seen as an another example where the use of octree AMR and dynamic balance constitutes a powerful tool to reproduce complex phenomena.

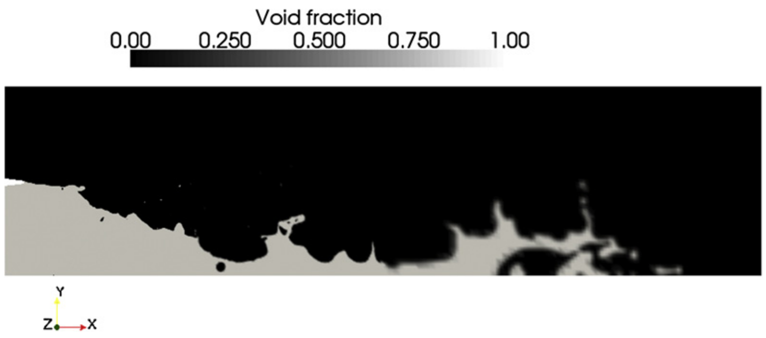

Fig. 10. Void fraction field in the whole simulation domain (8 × 2 cm). Near the injector, a large level of refinement is used in order to capture the effect of the small disturbances on the primary atomisation zone. The right boundary of the domain is placed further downstream in order to avoid any influence of the boundary conditions on the flow near the injector. In this region, a low level of refinement is used which has related a significant save of computational time.

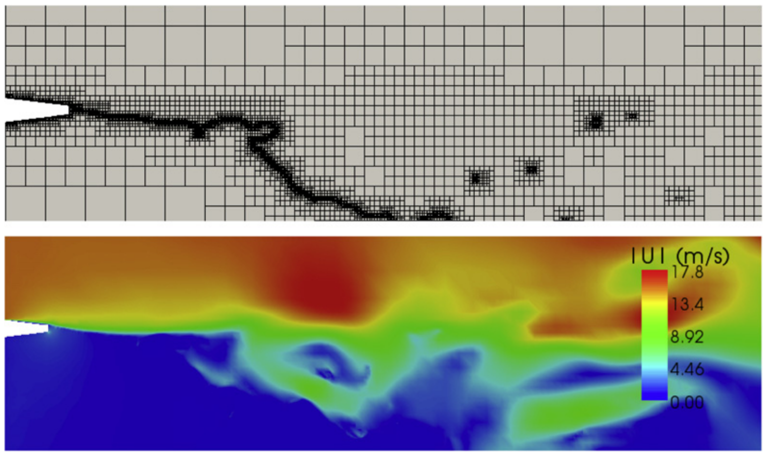

Fig. 11. Adaptive mesh (top) and velocity field (modulus) of the flow near the injector. The mesh is adapted at the interface for an accurate representation of the interface. The important velocity gradients induced in the gas phase are also accurately captured by applying dynamically adaptive criteria.

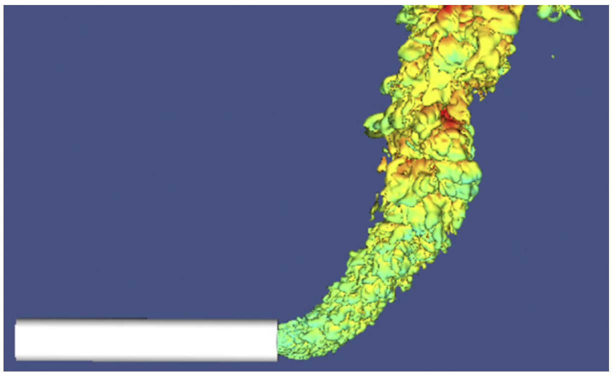

Fig. 12. Three-dimensional representation of the water/oil interface of a plume generated by the oil leak. Fr = 10. In colours, the norm of the vertical velocity at the interface. The AMR techniques implemented in Gerris allow capturing the small structures generated by the turbulence of the flow at the tube exit while keeping a relatively low total number of nodes in all the computational domain when compared with an uniform mesh.

In absence of viscosity (Re = ∞), the Froude number defined as

$$\mathrm{Fr} = \frac{U_0^2}{gD} \frac{\rho_w}{\rho_w - \rho_{\mathrm{oil}}} \tag{3}$$

is the only dimensionless number controlling the curvature of the plume, where U_0 is the velocity at the tube exit, D is the diameter of the tube, g is the gravity, and ρ_w and ρ_{oil} are respectively, the density of the water and the oil. Fig. 12 and the animation at http://www.dalembert.upmc.fr/ijlrda/images/stories/level10Fr10.mov contains a three-dimensional representation of the water/oil interface for $\mathrm{Fr} = 10$. Taking the diameter of the tube D as a reference distance, a domain of $40 \times 60 \times 20$ is simulated with a minimum mesh size of 0.0195. The domain is initially constructed with 6×20^3 cubic boxes, then these boxes are further subdivided up to 10 levels of refinement, an equivalent 6×1024^3 resolution. Two criteria are used to dynamically adapt the mesh: the norm of the vorticity and an a posteriori error estimate on the prediction of the concentration field. These criteria are used as a cost function for adaptation. The a posteriori error is estimated by computing the norm of the Hessian matrix of the given field, estimated using third-order-accurate discretisation operators. The use of the adaptive techniques reduces the number of cells compared to the equivalent Cartesian grid simulation by three orders of magnitude, making possible to perform this type of simulations in a cluster of 32 cpus in 50 hours of cpu time (the simulation time is $35.7\sqrt{D/g}$).

7. Conclusion

We have introduced the implementation of the volume-of-fluid method in Gerris and described the parallelisation technique used. The method is still being developed but is powerful enough to attack extremely challenging problems such as the small-scale structures created a very short time after droplet impact. Simulations of atomisation in a set-up closely matching that of the Grenoble experiment have been performed and show quantitative agreement with experimental results. Finally a turbulent three-dimensional variable-density single-phase-flow example is shown. We expect the set of methods described in this paper to prove useful in the resolution of problems of even larger complexity in the future.

References

[1] D. Fuster, G. Agbaglah, C. Josserand, S. Popinet, S. Zaleski, Numerical simulation of droplets, bubbles and waves: state of the art, Fluid Dynam. Res. 41 (2009) 065001.
[2] S. Popinet, An accurate adaptive solver for surface-tension-driven interfacial flows, J. Comput. Phys. 228 (16) (2009) 5838–5866.
[3] M. Sussman, A.S. Almgren, J.B. Bell, P. Colella, L.H. Howell, M.L. Welcome, An adaptive level set approach for incompressible two-phase flows, J. Comput. Phys. 148 (1999) 81–124.
[4] R. Lebas, T. Menard, P.A. Beau, A. Berlemont, F.X. Demoulin, Numerical simulation of primary break-up and atomization: DNS and modelling study, Int. J. Multiphase Flow 35 (2009) 247–260.
[5] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries, J. Comput. Phys. 190 (2) (2003) 572–600.
[6] G. Agresar, J.J. Linderman, G. Tryggvason, K.G. Powell, An adaptive, Cartesian, front-tracking method for the motion, deformation and adhesion of circulating cells, J. Comput. Phys. 43 (1998) 346–380.
[7] F. Gibou, L. Chen, D. Nguyen, S. Banerjee, A level set based sharp interface method for the multiphase incompressible Navier–Stokes equations with phase change, J. Comput. Phys. 222 (2007) 536–555.
[8] S. Popinet, The Gerris flow solver, http://gfs.sf.net, 2010.
[9] G. Tryggvason, R. Scardovelli, S. Zaleski, Direct Numerical Simulations of Gas–Liquid Multiphase Flows, Cambridge University Press, 2011, in press.
[10] E. Aulisa, S. Manservisi, R. Scardovelli, S. Zaleski, Interface reconstruction with least-squares fit and split advection in three-dimensional Cartesian geometry, J. Comput. Phys. 225 (2007) 2301–2319.
[11] J. Li, Calcul d'interface affine par morceaux (piecewise linear interface calculation), C. R. Acad. Sci. Paris Sér. IIb (Paris) 320 (1995) 391–396.
[12] R. Scardovelli, S. Zaleski, Interface reconstruction with least-square fit and split Lagrangian–Eulerian advection, Int J. Numer. Meth. Fluids 41 (2003) 251–274.
[13] E. Aulisa, S. Manservisi, R. Scardovelli, S. Zaleski, A geometrical area-preserving volume of fluid advection method, J. Comput. Phys. 192 (2003) 355–364.
[14] M.M. Francois, S.J. Cummins, E.D. Dendy, D.B. Kothe, J.M. Sicilian, M.W. Williams, A balanced-force algorithm for continuous and sharp interfacial surface tension models within a volume tracking framework, J. Comput. Phys. 213 (1) (2006) 141–173.
[15] S. Popinet, The GNU triangulated surface library, http://gts.sf.net, 2010.
[16] R. Diekmann, R. Preis, F. Schlimbach, C. Walshaw, Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM, Parallel Comput. 26 (12) (2000) 1555–1581.
[17] J.E. Boillat, Load balancing and Poisson equation in a graph, Concurrency: Practice and Experience 2 (4) (1990) 289–313.
[18] Y.F. Hu, R.J. Blake, D.R. Emerson, An optimal migration algorithm for dynamic load balancing, Concurrency: Practice and Experience 10 (6) (1998) 467–483.
[19] J.K. Johnson, D. Bickson, D. Dolev, Fixing convergence of Gaussian belief propagation, in: IEEE International Symposium on Information Theory (ISIT) 2009.
[20] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer-Verlag, 2004.
[21] S. Popinet, Atomisation of a pulsed liquid jet, http://gfs.sourceforge.net/examples/examples/atomisation.html, 2009.
[22] D. Fuster, A. Bagué, T. Boeck, L. Le Moyne, A. Leboissetier, S. Popinet, P. Ray, R. Scardovelli, S. Zaleski, Simulation of primary atomization with an octree adaptive mesh refinement and VOF method, Int. J. Multiphase Flow 35 (6) (2009) 550–565.
[23] J.L. Zable, Splatter during ink jet printing, IBM J. Res. Development 21 (4) (1977) 315–320.
[24] P. Einsenklam, Atomisation of liquid fuel for combustion, J. Inst. Fuel 0 (1961) 130–143.
[25] O. Planchon, E. Mouche, A physical model for the action of raindrop erosion on soil microtopography, Soil Sci. Soc. Am. J. 74 (2010) 1092–1103.
[26] S.T. Thoroddsen, T.G. Etoh, K. Takehara, Air entrapment under an impacting drop, J. Fluid Mech. 478 (2003) 125–134.
[27] V. Mehdi-Nejad, J. Mostaghimi, S. Chandra, Air bubble entrapment under an impacting droplet, Phys. Fluids 15 (1) (2003) 173–183.
[28] C. Josserand, S. Zaleski, Droplet splashing on a thin liquid film, Phys. Fluids 15 (2003) 1650.
[29] S. Mandre, M. Mani, M.P. Brenner, Precursors to splashing of liquid droplets on a solid surface, Phys. Rev. Lett. 102 (2009) 134502.
[30] S.D. Howison, J.R. Ockendon, J.M. Oliver, R. Purvis, F.T. Smith, Droplet impact on a thin fluid layer, J. Fluid. Mech. 542 (2005) 1–23.
[31] F. Ben Rayana, Étude des instabilités interfaciales liquide-gaz en atomisation assistée et tailles de gouttes, PhD thesis, Institut National Polytechnique de Grenoble, 2007.
[32] T. Boeck, S. Zaleski, Viscous versus inviscid instability of two-phase mixing layers with continuous velocity profile, Phys. Fluids 17 (2005) 032106.
[33] P. Marmottant, E. Villermaux, On spray formation, J. Fluid Mech. 498 (2004) 73–111.
[34] T.J. Crone, M. Tolstoy, Magnitude of the 2010 Gulf of Mexico oil leak, Science 330 (6004) (2010) 634.