Journal of Computational Physics 229 (2010) 8364–8383

Contents lists available at ScienceDirect

Journal of Computational Physics

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

# High order finite volume methods on wavelet-adapted grids with local time-stepping on multicore architectures for the simulation of shock-bubble interactions

Babak Hejazialhosseini, Diego Rossinelli, Michael Bergdorf, Petros Koumoutsakos *

*Chair of Computational Science, ETH Zürich, CH-8092, Switzerland*

ARTICLE INFO

*Article history:*
Received 2 February 2010
Received in revised form 21 June 2010
Accepted 15 July 2010
Available online 4 August 2010

*Keywords:*
Wavelets
Adapted grid
Local time-stepping
Multiphase flow
Multicore architectures

ABSTRACT

We present a space–time adaptive solver for single- and multi-phase compressible flows that couples average interpolating wavelets with high-order finite volume schemes. The solver introduces the concept of wavelet blocks, handles large jumps in resolution and employs local time-stepping for efficient time integration. We demonstrate that the inherently sequential wavelet-based adaptivity can be implemented efficiently in multicore computer architectures using task-based parallelism and introducing the concept of wavelet blocks. We validate our computational method on a number of benchmark problems and we present simulations of shock-bubble interaction at different Mach numbers, demonstrating the accuracy and computational performance of the method.



## 1. Introduction

The efficient implementation of computational methods on parallel computer architectures has enabled unprecedented large scale flow simulations using billions of computational elements [1,2]. Such massively parallel simulations usually benefit from the regularity of structured grids that translate into effective computer implementations by means of data parallelism. Even with the ever increasing capabilities of computer architectures, further methodological developments are necessary in order to address problems of engineering interest such as the flow around an aircraft wing or cavitation phenomena in turbomachinery. These methodological developments must include adaptive methods that can dynamically adjust the resolution of the computational elements according to the phenomena they aim to resolve. Methods such as Adaptive Mesh Refinement (AMR) [3,4] or wavelet-based multiresolution techniques [5–9] have been developed to this end. Wavelets (see the recent review [10] and references there in) have been implemented successfully for the simulation of conservations laws in computational fluid dynamics. In addition to providing a framework for solving partial differential equations, they can be adapted to multiresolution geometrical representations using level sets [11]. More recently an adaptive wavelet collocation method was applied to simulations of shocks interacting with interfaces [12] demonstrating how the wavelet coefficients can be used to detect the emergence of localized structures.

Spatially adaptive grids can be further enhanced by their coupling with local time-stepping integration schemes (LTS). These schemes exploit the locality of the time step stability condition: coarser elements can be integrated with larger time steps than the finer ones thus needing less integration steps. Local time-stepping schemes have been shown to speedup the

* Corresponding author. Tel.: +41 1 632 5258; fax: +41 1 632 1703.
*E-mail addresses:* babak.hejazi@inf.ethz.ch (B. Hejazialhosseini), diegor@inf.ethz.ch (D. Rossinelli), bergdorf@inf.ethz.ch (M. Bergdorf), petros@ethz.ch (P. Koumoutsakos).

0021-9991/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2010.07.021

computation by one or more orders of magnitude [13] depending on the number of blocks at each level of resolution. Presently the formulation of LTS schemes is limited to second order accuracy. We propose LTS schemes where each computational element of the grid is considered as an independent entity. This formulation allows the development of higher order time-steppers that are in line with the higher order allowed by the spatial multiresolution capabilities of our method.

The data structures associated with adaptive methods need to be managed efficiently to exploit the modern multicore computer architectures with an ever increasing number of cores. Wavelets, while efficient in providing spatial adaptivity, do not result in highly parallel algorithms, due to their inherently sequential, hierarchical, nested data structure. This limits their effective implementation on multicore architectures and affects the development of high performance, per-thread-based software that is independent from a specific hardware architecture. This limitation however can be relaxed by employing *task-based parallelism* [14]. Because of the various types of multicore architectures, their large heterogeneity and their difference in the number of cores, it is desirable that the simulation software abstracts from a specific architecture. Such abstractions can be achieved by means of standard parallel libraries such as OpenMP (versions prior to 3.0) [15] or HMPP [16]. Those libraries work on the basis of a per-thread specification that may enable an acceptable scaling in case of ideal parallel grain size and trivial data parallelism. In case of non-uniform grain size or non-trivial data (e.g. recursive) parallelism those libraries present two shortcomings: they could suffer from bad scaling and they do not allow a clear expression of the parallelism itself. Recent advances in multithreading software technology embodied in Cilk [17], Intel Threading Building Blocks [18] and other libraries [19] allow to expose parallelism on a per-task basis which is better suited for non-trivial data parallelism. In this fashion, good scaling is achieved by specifying parallel tasks that are dynamically scheduled over the physical threads. The benefits of the dynamic scheduler are twofold: it can efficiently handle nested-parallel load-unbalanced tasks using a strategy defined as *Work Stealing* [20,21] and it automatically finds the optimal grain size for a given set of parallel tasks, achieving a good scaling among different hardware architectures.

In this work we combine wavelet-based adaptive grid methods, with finite volume methods [22] and LTS schemes and implement them efficiently on multicore computer architectures for the simulation of multi-phase flows. We use the problem of shock-bubble interaction as a validation testbed for our computational methods. Shock-bubble interactions have been studied experimentally by Haas and Sturtevant [23]. Simulations of shock-bubble interactions often treat the two phases as gases of different density while the viscosity of the fluid as well as the surface tension of the bubble are neglected. The development of a cascade of scales as a result of non-linear wave interactions and instabilities along with the lack of physical diffusion present significant challenges to computational methods [24–26]. The proposed wavelet-adapted grids aim to capture this cascade of scales. In order to effectively represent these grids in parallel computer architectures, we design a simple data structure to overcome the inherent fine granularity of the methods associated with single grid points in wavelet-adaptive grids. The LTS algorithms are reformulated such that they can be efficiently incorporated with generic adaptive grids. We demonstrate how these considerations contribute to the speedup and parallel scaling of our simulation software.

The paper is structured as follows: In Section 2 we present the governing equations of multi-phase compressible flows and explain the numerical method regardless of any adaptivity in Section 3. We then use a wavelet-based multiresolution analysis to introduce an adaptive grid and implement our numerical methods on these dynamically evolving grids in Section 4. We introduce a data structure referred to as blocks to reduce the granularity of the method and make it suitable for task parallelism. Finally, we describe our simple yet efficient algorithm for local time-stepping and demonstrate the results of validation tests and simulations of shock-bubble interaction at different Mach numbers along with performance analysis in Section 5. Finally, we include the basis of our wavelets-driven approach in the Appendix.

## 2. Governing equations

We consider an inviscid, compressible, two phase flow described by the Euler equations as a one fluid model [27]:

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \\
&\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p \mathbb{I}) = \mathbf{0} \\
&\frac{\partial (\rho E)}{\partial t} + \nabla \cdot ((\rho E + p)\mathbf{u}) = 0
\end{aligned} \tag{1}$$

with $\rho$ being the density, $\mathbf{u}$ the velocity vector, $p$ the pressure and $E$ the total energy of the fluid per unit mass. Furthermore we assume that the two phases follow the ideal gas equation of state,

$$p = (\gamma - 1)\rho\left(E - \frac{1}{2}|\mathbf{u}|^2\right) \tag{2}$$

with $\gamma$ being the ratio of specific heats of each phase.

The interface is represented by a level set function [28], $\phi$, such that $\phi < 0$ embodies fluid 1 whereas fluid 2 lies in $\phi \geqslant 0$. Material properties (here only $\gamma$) are then identified using $\phi$, thus coupling the evolution of the interface to Eq. (1) through Eq. (2). The evolution of interface is governed by a linear advection equation of the form,

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0 \tag{3}$$

which is solved only close to the interface with $|\nabla\phi| = 1$ as the level set function is chosen to be a signed distance function. The signed distance function property is often violated during the evolution of the interface, deteriorating the accuracy of approximating the spatial derivatives of $\phi$ needed in Eq. (3). In order to maintain this regularity, we solve the re-initialization equation,

$$\frac{\partial\phi}{\partial t} + sgn(\phi_0)(|\nabla\phi| - 1) = 0 \tag{4}$$

as proposed by Sussman et al. [29] until steady-state. Moreover, we employ the sub-cell correction introduced by Russo and Smereka [30] to avoid dislocating the interface ($\phi = 0$) while re-initializing the level set field.

## 3. Numerical methods for uniform computational grids

The governing flow Eq. (1) can be cast into a vector form:

$$\mathbf{q}_t + \nabla\cdot\mathbf{f}(\mathbf{q}) = \mathbf{0} \tag{5}$$

where $\mathbf{q}(x,t) = (\rho, \rho\mathbf{u}, \rho E)^T$, with initial conditions $\mathbf{q}(\mathbf{x},0) = \mathbf{q}_0(\mathbf{x})$ and appropriate boundary conditions. The integral form of (5), written in 1D as

$$\oint (\mathbf{q}dx - f(\mathbf{q})dt) = 0 \tag{6}$$

is used as a starting point for the finite volume discretization. If the computational domain is uniformly discretized by finite volumes, then cell averages $\{\mathbf{q}_i\}$ at time $t = t_n$ and the flux

$$\mathbf{F}_{i\pm1/2} = \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} \mathbf{f}\Big(\mathbf{q}_{i\pm1/2}(t)\Big)dt \tag{7}$$

determine the new solution at time $t_{n+1} = t_n + \Delta t$, $\mathbf{q}^{n+1}$. In order to avoid the expensive Riemann solver [31], $\mathbf{F}_{i\pm1/2}$ is approximated by a numerical flux $\widehat{\mathbf{F}}^n_{i\pm1/2}$. The new values of $\{\mathbf{q}_i^{n+1}\}$ are found after one simulation step, by evaluating the numerical fluxes and perform a time integration for all the averages:

$$\mathbf{q}_i^{n+1} = \mathbf{q}_i^n - \frac{\Delta t}{\Delta x}\left(\widehat{\mathbf{F}}_{i+1/2} - \widehat{\mathbf{F}}_{i-1/2}\right) \tag{8}$$

Since $\widehat{\mathbf{F}}_{i+1/2}$ and $\widehat{\mathbf{F}}_{i-1/2}$ depend on the local cell neighbors of $\mathbf{q}_i^n$, the simulation step formulated in (8) can be seen as a nonlinear uniform filtering at the location of $\mathbf{q}_i^n$. There have been various formulations for these numerical fluxes namely Roe [32], Lax-Friedrichs [33] and HLL [34]. In this paper, we use the HLLE [35] flux which is capable of correctly resolving isolated shocks and rarefaction waves.

The exact Riemann solver and its approximate versions require a reconstruction step which provides them with the left and the right states as well as the exact and approximate characteristic velocities on the cell interfaces. A broad range of reconstruction-evolution methods has been developed to make this reconstruction high-order as well as oscillation-free e.g. TVD/MUSCL [36], PPM [37], ENO [38], etc. Besides, it has also been shown that the reconstruction of conserved quantities, $\mathbf{q}$, leads to oscillations in the pressure close to the contact discontinuity and violate the zero jump conditions of velocity and pressure across the interface generating spurious wiggles on the interface [39]. In 2D, this can be overcome by reconstructing the primitive quantities $\mathbf{u} = (\rho, u, v, p)$, based on which the conserved quantities and fluxes are then calculated [40,41]. We use fifth-order WENO scheme [42,43] to this end. Total Variation Diminishing (TVD) Runge–Kutta scheme of order two is used as the time-stepper [44]. The interface evolution Eq. (3) is made suitable for our generic conservative flux-based solver using the method in [45]. Since the flow of interest is dominated by discontinuities, Eq. (4) is solved at every time step to maintain the regularity of the level set field, $\phi$.

We use a mollified Heaviside function to retrieve the heat capacity ratio ($\gamma$) of the phases from the level set field, $\phi$,

$$H_\epsilon(\phi) = \begin{cases} 0 & \phi < -\epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right) & |\phi| \leqslant \epsilon \\ 1 & \phi > \epsilon \end{cases} \tag{9}$$

$$\gamma(\phi) = \gamma_1 H_\epsilon(\phi) + \gamma_2(1 - H_\epsilon(\phi)) \tag{10}$$

where the mollification length, $\epsilon$, is set to the smallest possible spatial resolution and is kept constant during the simulation.

## 4. Wavelet-based adaptive computational grids

In the first part of this section we briefly introduce the way wavelets are used to represent adaptive computational grids. It is followed by a description of the ghost reconstruction. We then introduce the wavelet blocks, a key aspect of the present work, and discuss their advantages. Finally, we present the local time-stepping schemes in conjunction with wavelet blocks.

### 4.1. Wavelet-based adaptive grids

We use biorthogonal wavelets to build up a multiresolution analysis of the flow field. Biorthogonal wavelets are characterized by symmetric and smooth scaling functions [46]. Moreover, they have two (in general different) pairs of scaling and wavelet functions for analysis (performed by the Fast Wavelet Transform or FWT) and synthesis (performed by the inverse FWT). A function of interest can therefore be decomposed into scaling coefficients and detail coefficients at different levels of resolution, i.e. an *MRA* of the function. One can discard the detail coefficients that, compared to a threshold, do not carry significant information. The result of this thresholding is a compressed representation of the original field. One can then retain only the *active scaling coefficients* on different levels of resolution i.e. a spatially-adapted grid. This grid can in turn be coupled to finite volume or finite-difference schemes. In order to avoid any modification to these schemes due to different length scales in their stencils, we temporarily introduce the *ghosts* whenever a jump in resolution happens. The following sections are largely based on wavelet transforms and adaptive grid representations in terms of scaling coefficients. We refer the reader to the Appendix for a detailed discussion on how wavelets are used to represent an adaptive computational grid.

### 4.2. Ghosts

After performing an MRA to a flow quantity of interest, we assume to have a wavelet-adapted grid made of active scaling coefficients $c_i^l$ with $l$ being the level of resolution and $i$ the position of the scaling coefficient in the index space. We then discretize the differential operators by applying standard finite volume or finite-difference schemes on the active coefficients. Such operators can be viewed as (non-linear) filtering operations on uniform resolution grid points, formally:

$$F(c_k^l) = \sum_{j=s_f}^{e_f-1} c_{k+j}^l \beta_j, \quad \beta_j^l \text{ function of } \{c_m^l\} \tag{11}$$

where $\{s_f, e_f - 1\}$ is the support of the filter, and the filter coefficients are $\{\beta_j\}$.

In order to perform uniform resolution filtering we need to temporarily introduce artificial auxiliary grid points, the so-called *ghosts*. These ghosts are necessary in order to ascertain that for every grid point $k$ in the adapted grid $\mathcal{G}$, its neighborhood $[k - k_f, k + k_f]$ contains its stencil elements, either as active grid points $k'$ in $\mathcal{G}$ or ghosts $g$. Using this set of ghosts, which can be either precomputed and stored or computed on the fly, we are now able to apply the filter $F$ to all points $k$ in $\mathcal{G}$. The ghosts are constructed from the active scaling coefficients as a weighted average $g_i^l = \sum_l \sum_j w_{ijl} c_j^l$, where the weights $w_{ijl}$ are provided by the FWT and the inverse FWT. It is convenient to represent the construction of a ghost as $g_i = \sum_j w_{ij} p_j$, where $i$ is the identifier for the ghost $g$ and $j$ represents the identifier for the source point $p$ which is an active scaling coefficient in the grid. Calculation of the weights $\{w_{ij}\}$ is done by traversing a graph associated with the FWT and the inverse wavelet transform (Fig. 1). This operation can be expensive for several reasons: firstly, if the graph contains loops, we need to solve a linear system of equations to compute $\{w_{ij}\}$. Fig. 1 (left) shows this issue for a two-resolutions grid associated with the third-order b-spline wavelets. According to the one-level inverse wavelet transform, the evaluation of the ghost $g_A$ consists of a weighted average of the points $\{c_{-3}^0,\ c_{-2}^0,\ c_{-1}^0,\ g_B\}$, where $g_B$ is a secondary ghost. The value of $g_B$ is obtained from the points $\{c_0^1,\ c_1^1,\ c_2^1,\ g_A\}$, according to the one-level FWT. As $g_A$ depends on $g_B$ and vice versa, one has to solve the following linear system to find $g_A$:

$$g_A = \frac{1}{4}c_{-3}^0 + \frac{3}{4}c_{-2}^0 + \frac{3}{4}c_{-1}^0 + \frac{1}{4}g_B \tag{12}$$

$$g_B = \frac{3}{4}c_0^1 + \frac{3}{4}c_1^1 - \frac{1}{4}c_2^1 - \frac{1}{4}g_A \tag{13}$$

For any order of b-spline wavelets, one can find $g_A$ by introducing a vector of ghost values $\mathbf{g}$ and having $g_A$ as the first component of $\mathbf{g}$ (i.e. $g_A = \mathbf{e}_1^T \cdot \mathbf{g}$). The remaining components are secondary ghosts involved in the calculation of $g_A$. It holds:

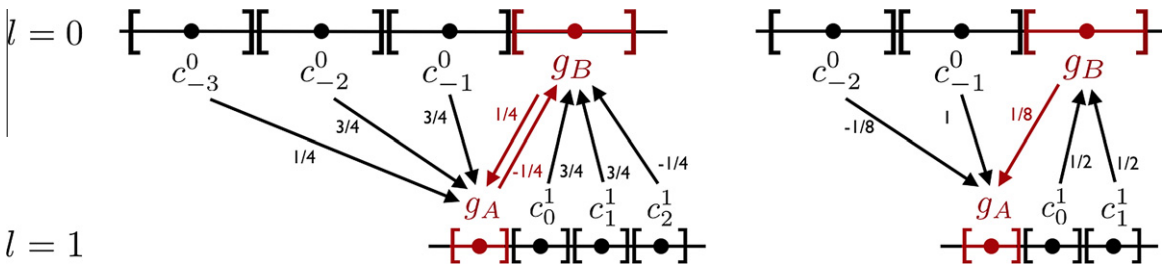


**Fig. 1.** Graph of the contributions for the reconstruction of the ghost $g_A$, with a resolution jump of 1 for the case of third-order b-spline wavelets (left) and third-order average interpolating wavelets (right). The arrows, and their associated weights, denote contributions from the grid points to the ghosts $g_A$ and $g_B$. Both wavelets need the secondary ghost $g_B$ to evaluate $g_A$, but average interpolating wavelets are more efficient as the evaluation of $g_B$ does not depend on $g_A$ (i.e. there is no loop in the graph).

$$\mathbf{g} = \mathbf{W}_{ghosts} \cdot \mathbf{g} + \mathbf{W}_{points} \cdot \mathbf{k} \tag{14}$$

where the matrices $\mathbf{W}_{ghosts}$ and $\mathbf{W}_{points}$ are specific to the wavelet type, and $\mathbf{k}$ is a vector containing all the grid point values involved in the reconstruction of $g_A$. The entry $(\mathbf{W}_{ghosts})_{ij}$ contains the weight that the ghost $g_i$ receives from the ghost $g_j$, whereas the entry $(\mathbf{W}_{points})_{ij}$ contains the weight that $g_i$ receives from the point $p_j$. The ghost $g_A$ can be found with an expensive matrix inversion:

$$g_A = \mathbf{e}_1^T \cdot (\mathbf{I} - \mathbf{W}_{ghosts})^{-1} \mathbf{W}_{points} \cdot \mathbf{k} \tag{15}$$

Furthermore, the evaluation of the ghosts is costly in areas of the grid with resolution jumps. The number of secondary ghosts involved in the reconstruction of $g_A$, and therefore the cost of inverting $(\mathbf{I} - \mathbf{W}_{ghosts})$, grows exponentially with the resolution jump. Fig. 1 (right) shows the evaluation of $g_A$ with the use third-order average interpolating wavelets. By virtue of their property, the ghost reconstruction is straightforward and leads to efficient reconstruction formulae. Since the evaluation of $g_B$ does not involve $g_A$, we have

$$g_A = -\frac{1}{8} c_{-2}^0 + c_{-1}^0 + \frac{1}{8}\left(\frac{1}{2} c_0^1 + \frac{1}{2} c_1^1\right). \tag{16}$$

In this work we use fifth-order average interpolating wavelets, which have similar efficient reconstruction formulae but involve more points.

### 4.3. Wavelet blocks

The wavelet-adapted grids for finite-differences/volumes are often implemented with quad-tree or oct-tree structures whose leaves are single scaling coefficients. The main advantage of such fine-grained trees is the high compression rate by thresholding individual detail coefficients. On the other hand, the drawback of this approach is the large amount of sequential operations they involve and the number of indirections (or instructions) necessary to access a group of elements. Even in cases where we only compute without changing the structure of the grid, these grids already perform a great number of neighborhood look-ups. In addition, operations like refining or coarsening single grid points have to be performed and those operations are relatively complex and strictly sequential. In order to decrease the amount of sequential operations per grid point, we simplify the data structure by decreasing the granularity of the method at the expense of a reduction in the compression rate. Hence, we introduce the concept of blocks of scaling coefficients whose size are one or two orders of magnitude larger in each direction i.e. in 2D the granularity of a block is 3 orders of magnitude coarser than a single scaling coefficient. The grid is then represented with a tree which contains blocks as leaves (Fig. 2).

A block consists of a predefined number of scaling coefficients in each dimension, denoted as $s_{block}$. All the blocks contain the same number of scaling coefficients. The scaling coefficients contained in one block have the same level. In the physical space, the blocks have varying size and therefore different resolution. We note that none of the blocks overlap with any other block in the physical space because blocks are the leaves of the tree. Blocks can be split and doubled in each direction and they can be collapsed into one coarser block.

The introduction of blocks has several computational advantages over the adapted grids with fine-grained trees. The first benefit is that tree operations are now accelerated as they can be performed in $\log_2(N^{1/D}/s_{block})$ operations instead of $\log_2(N^{1/D})$, where $N$ is the total number of active coefficients, $D$ the dimensions in consideration. The second benefit is that blocks have random access: their scaling coefficients can be retrieved in $O(1)$ read accesses. As a consequence, the third advantage is the reduction of the sequential operations involved in processing a local group of scaling coefficients. We consider the cost $c$ (in terms of memory accesses) of filtering a grid point with a finite difference scheme of size $w_{stencil}$ in each direction. In a

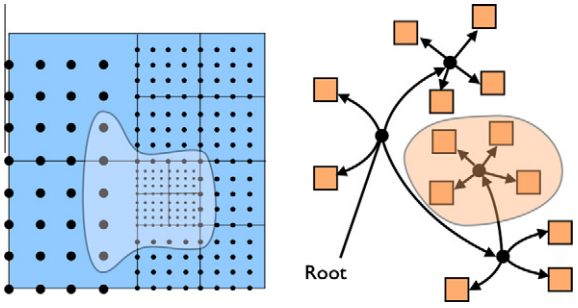


**Fig. 2.** Representation of the blocks in physical space (left), and the associated tree with the blocks as leaves (right). Collapsing four blocks (right, shaded region with light orange) into a coarser one requires ghosts, since the coarser block is obtained by filtering the four blocks (one-level FWT). The computation of the ghosts involves scaling coefficients also from the neighbor blocks, which can have different resolutions (left, shaded region with light blue). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

uniform resolution grid, it holds $c = w_{stencil}D$. For a grid represented by a fine-grained tree, the number of accesses is proportional to $c = w_{stencil}D\log_2(N^{1/D})$. Using the wavelet blocks approach, we assume that $s_{block}$ is roughly one order of magnitude larger than $w_{stencil}$. The ratio of ghosts needed per grid point in order to perform the filtering for a block, is:

$$r = \frac{(s_{block} + w_{stencil})^D - s_{block}^D}{s_{block}^D} \approx D\frac{w_{stencil}}{s_{block}} \tag{17}$$

Therefore, the number of accesses for filtering one grid point is:

$$c = (1-r)w_{stencil}D + rw_{stencil}D\left(\log_2\left(N^{1/D}/s_{block}\right)\right) = w_{stencil}D + w_{stencil}Dr\left(\log_2\left(N^{1/D}/s_{block}\right) - 1\right) \tag{18}$$

In order to improve the efficiency of finding the neighbors of a block, we constrain the neighbors to be only the adjacent ones. Because of this constraint, the highest jump in resolution allowed (between two adjacent blocks), $L_j$, is bounded by $\log_2(s_{block}/w_{stencil})$. All the results presented in this work were obtained with $L_j = 2$.

### 4.4. Local time-stepping (LTS) schemes

Space-adapted grids enable considerable performance improvement because of their ability in adjusting the resolution of the computational elements to the emerging scales. An additional substantial speedup can be achieved by coupling space-adapted grids with local time-stepping (LTS) integration schemes. These schemes exploit the locality of the time step imposed by stability conditions: coarser elements can be integrated with larger time steps than the finer ones thus demanding fewer integration steps.

Local time-stepping schemes have been shown to speedup the computation by one order of magnitude or more [47,13,48,49] depending on the number of blocks at each level of resolution. The current issue with LTS schemes is that their formulation is limited to second order accuracy: up to present-day, to the best of our knowledge, no third-order (or higher) accurate LTS has been presented. A main characteristic of these algorithms is that they consist of monolithic and recursive sequences of repeated operations which may be difficult to implement in modern computer architectures. Within those operations, three distinct actions can be identified: the evaluation of the right hand side, the traversing of the grid and the update of the computational elements. Efficient implementation of algorithms on modern hardware- such as multicore architectures, GPUs and FPGAs- can be another major source of performance improvement. In order to be efficiently implemented, these algorithms should be formulated as a collection of compute intensive tasks that are data parallel (and possibly fine-grained). Efficient memory representation of data requires a simple access pattern that avoids multiple memory indirections.

We propose a new perspective on LTS schemes that, in combination with a block-based representation, is efficiently executed on multicore machines. We associate a reconstruction function to every computational element which reconstructs the value of that element in the physical time (see the following subsection). The computation of the right hand side is therefore not performed by considering directly the value of computational elements, but their time-reconstructed values.

Because the operations needed to integrate a grid point cannot be performed in a row, we need to introduce a state for the grid point. We then use a state diagram to identify the next operation needed in the time integration of the grid point. In the new formulation we relax, or almost eliminate, the coupling between the evaluation of the right hand side, the update and the marching through the elements. This approach enables us to identify the compute intensive parts of the LTS scheme and therefore accelerate them. The compute intensive part includes the reconstruction of values and ghosts in time and the evaluation of the spatial derivatives.

As we would like to abstract from any specific space-adaptive technique, we assume that the solution is represented on an adapted grid $\mathcal{G} = \{\mathbf{p_i}\}_{\mathbf{i}=0}^{\mathbf{N}}$, where $\mathbf{p_i}$ are generic computational elements. For convenience we will refer to them as grid points, that in our context correspond to scaling coefficients representing cell averages.

#### 4.4.1. Reconstruction functions

Each grid point consists in three variables $\mathbf{p}_i = \{\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i\}$. For each grid point $\mathbf{p}_i$ we define a reconstruction function $\mathbf{R}_i(t)$, which represents the solution vector $\mathbf{u}_i$ at a given time $t$:

$$\mathbf{u}_i(t) := \mathbf{R}_i(t) = t\mathbf{a}_i + \mathbf{b}_i \tag{19}$$

To compute the evolution of $\mathbf{u}_i$ (given $d\mathbf{u}_i/dt = \mathbf{f}_i$) at time $t$ we evaluate the right hand side $\mathbf{f}_i$ using time-reconstructed values of the grid points (as in Fig. 3 with $t = t^\star$). Whenever we need to compute the right hand side, it is stored in $\mathbf{c}_i$. The right hand side of $\mathbf{u}_i$ is computed as:

$$\mathbf{c}_i := \left.\frac{d\mathbf{u}_i}{dt}\right|_t = \mathbf{f}_i|_t \approx \mathbf{D}_i\left(\{\mathbf{R}_j(t)\}_{j\in N_i}\right), \tag{20}$$

where $N_i$ is the set of neighbor points needed to compute the right hand side at point $\mathbf{p_i}$, and $\mathbf{D}_i$ can be a finite-difference/volume scheme, or another discretization of the right hand side with a stencil size of $N_i$.

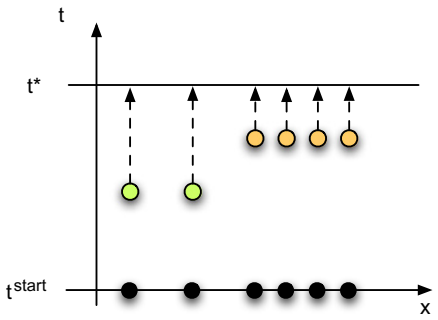


**Fig. 3.** Grid points of different length/time scale (denoted in black), have a different state (orange and green) and have been updated at two different times (vertical position of the colored circles). All the points are used for the computation of the right hand side at time $t^*$, which is performed by evaluating the reconstruction functions associated with each grid points (dashed arrows) and then by applying the finite volume scheme in space. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

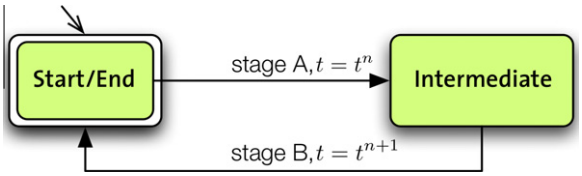


**Fig. 4.** State diagram used for a LTS scheme based on Euler and TVD RK2 time-stepper. The only accepting state is "Start/End". State transitions are allowed only at specific times: $t^n$ and $t^{n+1}$.

*4.4.2. Single grid point integration*

We define $\Delta t(\mathbf{p}_i)$ and $\Delta x(\mathbf{p}_i)$ to be the time and the length scale of the grid point $\mathbf{p}_i$. In the case that the grid is used to solve for advection, we have $\Delta t(\mathbf{p}_i) = \text{CFL} \cdot \Delta x(\mathbf{p}_i)/\|\mathbf{v}_{max}\|_\infty$, where CFL is the CFL number and $\|\mathbf{v}_{max}\|_\infty$ is the maximal speed in the field.

As the steps involved in the integration of a grid point may not be executed in a row, a state diagram is essential to recognize what are the missing operations to complete the integration for each individual grid point. These operations are grouped into "stages" and the operations within a stage are executed in a row. As illustrated in Fig. 4, for the first and second order time integration schemes, the grid point state starts/ends at a state denoted by "Start/End". The state diagram has another state denoted by "Intermediate". Since in this work the grid points within a block have the same time scale, it is enough to track a single state per block.

Stages trigger state transitions that change the values of $\mathbf{a}_i$ and $\mathbf{b}_i$ and must be called at specific times e.g. at time $t^n$ and $t^{n+1}$ (for stage A and stage B respectively). The right hand side must be evaluated as in Eq. (20) and stored in $\mathbf{c}_i$ before calling any stage, except for stage B in the Euler LTS scheme. For convenience, from here on we write **a**, **b**, **c** instead of $\mathbf{a}_i$, $\mathbf{b}_i$, $\mathbf{c}_i$.

*4.4.3. Marching through the grid points*

To time-integrate the whole grid, we let the grid points traverse their state diagrams multiple times. The number of times that a grid point will go through its state diagram depends on its time scale $\Delta t(\mathbf{p})$ (or the level of resolution it belongs to). We first partition the grid points according to their time scale into subsets. In this work we group blocks according to their level of resolution into subsets $\{\mathcal{G}_{level}\}_{level=0}^{N}$ because all the grid points within a block have the same time scale. We also force $\Delta t(\mathbf{p})$ to take only values of the form $\Delta t(\mathbf{p}) = K^{-level}\Delta T$, where $\Delta T$ is the coarsest time scale of the grid and $K \geqslant 1$ is a fixed integer number imposed by the stability condition of the underlying PDE. Simulating advection phenomena would require $K = 2$, whereas for diffusion problems $K = 4$.

Algorithm 1 shows how the LTS scheme recursively marches through the grid points on different levels of resolution and updates them. The meaning of line 2 is that we apply *stage* (which could be stage A or stage B) to all the points belonging to $\mathcal{G}_{\text{level}}$. In order to call these stages we need to pass $t$ and $\Delta t$ (computed on line 1).

At every time integration step, by calling LTS(0, 0, "A") the LTS starts iterating through the grid points and perform an integration step of $\Delta T$, by taking different smaller time steps $\Delta t$. We note that in every time integration step, $0 \leqslant t \leqslant \Delta T$ i.e. $t^n$ and $t^{n+1}$ are mapped to 0 and $\Delta T$. If there is only one subgroup of time scales, we recover the original global time-stepping scheme.

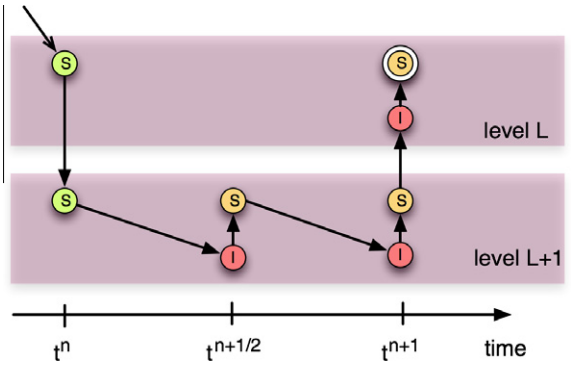


**Fig. 5.** Marching graph of Algorithm 1 through a two-level grid for LTS-based Euler/TVD RK2 steppers. An empty arrow indicates the starting node, whereas black arrows show the chronological steps in the marching, and white-stroked node denotes the end node. To get to a red node, one must always first evaluate the right hand side. To get to an orange node, one must evaluate the right hand side only for the RK2-based LTS. To get to green nodes, no evaluation of the right hand side is needed. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

**Algorithm 1**: [LTS (t, level, *stage*)]

1: $\Delta t = K^{-\text{level}}\Delta T$
2: apply *stage* with parameters $t$ and $\Delta t$ to the grid points in $\mathcal{G}_{\text{level}}$
3: *if stage* is "A" **then**
4: **for** $i = 0$ to $K - 1$ **do**
5: LTS($t + i\Delta t/K$, level + 1, "A")
6: **end for**
7:
8: LTS($t + \Delta t$, level, "B")
9: **end if**

Fig. 5 shows the marching sequence of the Algorithm 1 for $K = 2$ and a grid consisting of two subgroups of grid points at level $L$ and $L + 1$ (coarse and fine points) from $t^n$ to $t^{n+1}$.

*4.4.4. Local time-stepping using Euler scheme*

We introduce the operations associated with each state transition, starting from the LTS scheme based on the Euler time-stepper. As every point could be integrated asynchronously we need to find the necessary operations to integrate an individual point from $t = t^n$ to $t = t^{n+1}$. For this purpose, we consider the explicit, forward Euler time-stepping scheme

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t \mathbf{f}^n \tag{21}$$

We start with the state "Start/End" with $\mathbf{a} = \mathbf{0}$ and $\mathbf{b} = \mathbf{u}^n$ where $\mathbf{u}^n$ is the solution vector from the previous step. Then, the right hand side is computed at $t = t^n$ and it is stored into $\mathbf{c}$. We therefore call stage A and trigger the state transition from "Start/End" to "Intermediate", which modifies $\mathbf{a}$ and $\mathbf{b}$ (and thus $\mathbf{R}$). Stage A has the following postconditions on the new reconstruction function, denoted as $\mathbf{R}^{new}(t) = t\mathbf{a}^{new} + \mathbf{b}^{new}$

$$\begin{cases} \mathbf{R}^{new}(t^{n+1}) = \mathbf{u}^h + \Delta t \mathbf{f}^n \\ \frac{d}{dt}\mathbf{R}^{new} = \mathbf{f}^n \end{cases} \tag{22}$$

From these conditions we can infer the new values $\mathbf{a}^{new}$ and $\mathbf{b}^{new}$

$$\begin{cases} \mathbf{a}^{new} = \mathbf{f}^n \\ \mathbf{b}^{new} = \mathbf{u}^n + \Delta t \mathbf{f}^n - (t^n + \Delta t)\mathbf{a}^{new} \\ \qquad = \mathbf{u}^n - t^n \mathbf{a}^{new} \end{cases} \tag{23}$$

Considering that $\mathbf{c}$ contains the current evaluation of the right hand side (i.e. $\mathbf{f}^n$), we summarize stage A with the following instructions:

$\mathbf{a} \leftarrow \mathbf{c}$
$\mathbf{b} \leftarrow \mathbf{b} - t\mathbf{a}$

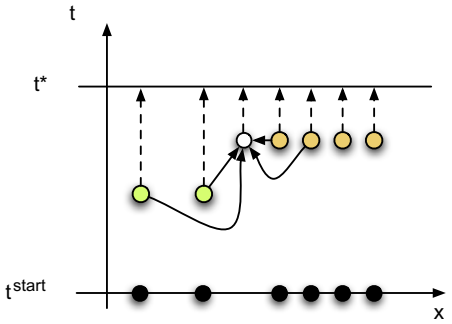


**Fig. 6.** The reconstruction of a ghost (white point) requires a tensorial weighted average: the ghost is first reconstructed in space (solid arrows), and then it is reconstructed in time (dashed arrow). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In stage B, we do not need a new right hand side evaluation, therefore we do not use **c**. We just update **b** with its reconstructed value and we reset **a** = **0**.

The instructions of stage B, that is called at $t = t^{n+1}$, read:

$$\mathbf{b} \leftarrow \mathbf{R}(t)$$
$$\mathbf{a} \leftarrow \mathbf{0}$$

States and transition stages for this scheme is shown in Fig. 4.

*4.4.5. Local time-stepping based on second order TVD Runge–Kutta scheme*

We design an LTS scheme of second order accuracy starting from the explicit forward TVD RK2 time-stepping:

$$\begin{aligned}\mathbf{u}^{*} &= \mathbf{u}^{n} + \Delta t \mathbf{f}^{\mathbf{n}} \\ \mathbf{u}^{n+1} &= \mathbf{u}^{n} + \frac{1}{2}\Delta t\left(\mathbf{f}^{n} + \mathbf{f}^{\star}\right)\end{aligned} \tag{24}$$

where $\mathbf{f}^{\star} = \mathbf{f}(\mathbf{u}^{\star})$. Stage A is identical to the one of the Euler-based LTS and it is called at time $t = t^{n}$, with of the following instructions:

$$\mathbf{a} \leftarrow \mathbf{c}$$
$$\mathbf{b} \leftarrow \mathbf{b} - t\mathbf{a}$$

Before calling stage B, the right hand side $\mathbf{f}^{\star}$ is computed at $t = t^{n+1}$ and stored into **c**. The postconditions of stage B on the new reconstruction function are:

$$\begin{cases}\mathbf{R}^{new}(t^{n+1}) = \mathbf{u}^{n} + \frac{1}{2}\Delta t\left(\mathbf{f}^{n} + \mathbf{f}^{\star}\right) \\ \frac{d}{dt}\mathbf{R}^{new} = \mathbf{0}\end{cases} \tag{25}$$

It holds:

$$\begin{cases}\mathbf{a}^{new} = \mathbf{0} \\ \mathbf{b}^{new} = \mathbf{u}^{n} + \frac{1}{2}\Delta t(\mathbf{f}^{n} + \mathbf{f}^{\star}) \\ \qquad = \mathbf{u}^{n} + \frac{1}{2}\Delta t\left(\mathbf{f}^{n} + \mathbf{f}^{\star}\right) - \left(\mathbf{u}^{n} + \frac{1}{2}\Delta t\mathbf{f}^{\mathbf{n}}\right) + \mathbf{R}\left(t - \frac{1}{2}\Delta t\right) \\ \qquad = \mathbf{R}\left(t - \frac{1}{2}\Delta t\right) + \frac{1}{2}\Delta t\mathbf{f}^{\star}\end{cases} \tag{26}$$

where $\mathbf{R}(t)$ is the reconstruction function prior to stage B. Stage B consists of the instructions:

$$\mathbf{b} \leftarrow \mathbf{R}\left(t - \frac{1}{2}\Delta t\right) + \frac{1}{2}\Delta t\mathbf{c}$$
$$\mathbf{a} \leftarrow \mathbf{0}$$

States and transition stages for this scheme is shown in Fig. 4. We note that the marching algorithm (Algorithm 1) together with the stages of this second order LTS scheme are mathematically equivalent to the second order accurate LTS scheme presented in [13].

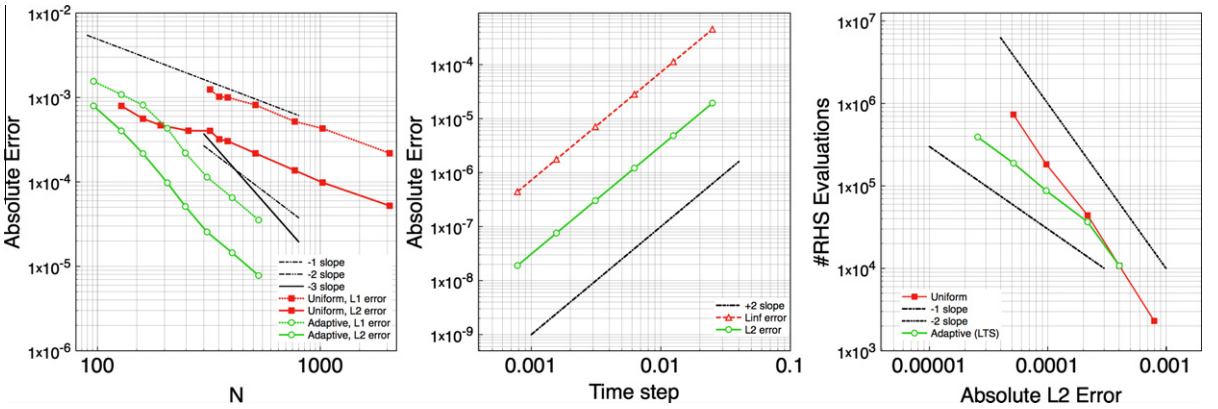


**Fig. 7.** $L_1$ and $L_2$ norm error of density vs. number of grid points ($N$, uniform and adaptive) for Sod shock tube problem (left), $L_2$ and $L_\infty$ norm error of second order Runge–Kutta LTS scheme vs. the time step for the linear advection problem of a Gaussian pulse (middle), Number of right hand side evaluations of global (uniform) time-stepping and LTS (adaptive) vs. the absolute $L_2$ error.

#### 4.4.6. Combining wavelet blocks and LTS schemes

Using our block-based representation, the creation of the ghosts involves a more expensive reconstruction, as it is done in both space and time:

$$\mathbf{g}_i(t) = \sum_j w_{ij}\mathbf{R}_j(t) = \sum_j w_{ij}\mathbf{b}_j + t\sum_j w_{ij}\mathbf{a}_j = \mathbf{b}_i^{ghost} + t\mathbf{a}_i^{ghost} \tag{27}$$

As also illustrated in Fig. 6, ghosts can be first reconstructed in space by creating $\mathbf{a}_i^{ghost}$ and $\mathbf{b}_i^{ghost}$. We note that $\mathbf{c}_i^{ghost}$ is not needed as we never evaluate the right hand side at the location of the ghosts.

## 5. Results

In this section we perform validations of the presented computational method on benchmark problems and present simulations of shock-bubble interaction in 2D for various Mach numbers.

### 5.1. 1D validations – The shock tube problem

We perform simulations of the Sod shock tube problem with the initial condition.

- $\rho = 1$, $u = 0$, $p = 1$ for $x < 0.5$
- $\rho = 0.125$, $u = 0$, $p = 0.1$ for $x \geqslant 0.5$

and $\gamma = 1.4$ up to the final time of 0.2. In Fig. 7 (left) it can be observed that the present numerical method when using uniform grids is not worse than any other finite volume method in capturing discontinuities i.e. it has the expected first order convergence and because of the spatial adaptivity, the order in $L_2$ norm is improved to about 3. The order of convergence in $L_1$ norm is improved from 1(uniform grid) to 2 (adaptive grid). It should be noted that adaptivity in a finite volume method does not improve the errors localized on the discontinuities (and therefore the $L_\infty$ norm error). However, this allows for grid compression where the detail coefficients are small enough so that we can achieve the same $L_2$ (or $L_1$) norm error as in the uniform case with fewer grid points. To demonstrate the order of our LTS algorithm, we advect a Gaussian density pulse [50] with a constant advection velocity. Fig. 7 (middle) shows the expected second order convergence. The algorithmic improvement of the LTS is presented in the Fig. 7 (right) and it is clear that to achieve a certain error (here in $L_2$ norm), we need fewer right hand side evaluations in our LTS than in the global (uniform) time-stepping. The difference in cost in terms of right hand side evaluations between the two solvers becomes larger for smaller errors i.e. by using an adaptive grid with more levels of resolution.

### 5.2. 2D shock-bubble interaction – validations and simulations at various mach numbers

We simulate the interaction of shock waves with three different Mach numbers, $M = 1.22$, 3 and 6, in air with a cylindrical bubble of helium. Initial condition is such that the bubble is in mechanical (pressure) and thermodynamical (temperature) equilibrium with the surrounding air. Jump conditions across the shock wave are obtained from the pressure and density ratios across a normal shock with a known Mach number given by:

**Table 1**
Wavelet detail thresholding results.

| $\epsilon$ | No. of blocks | $L_{max}$ | No. of blocks at $L_{max}$ |
|---|---|---|---|
| 0.5 | 28 | 3 | 16 |
| 0.1 | 585 | 6 | 8 |
| 0.05 | 1640 | 8 | 10 |
| 0.01 | 2327 | 8 | 60 |
| 0.005 | 9082 | 8 | 400 |

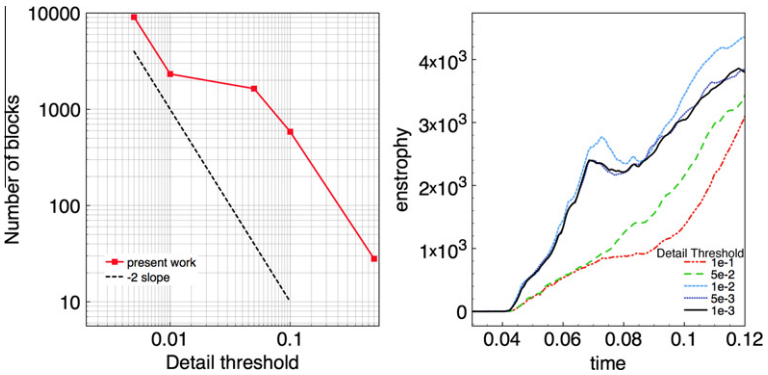


**Fig. 8.** Scaling of number of blocks in the adaptive grid with the wavelet detail coefficient threshold, $\epsilon$ (left), temporal evolution of enstrophy for different values of $\epsilon$ (right).

$$\frac{p_2}{p_1} = 1 + \frac{2\gamma}{\gamma+1}\left(M_1^2 - 1\right)$$
$$\frac{\rho_2}{\rho_1} = \frac{M_1^2}{1 + \frac{\gamma-1}{\gamma+1}\left(M_1^2 - 1\right)} \tag{28}$$

As an example, the initial condition for a Mach 3 shock wave can be written as:

- helium bubble: $\gamma = 1.677$, $\rho = 0.138$, $u = 0$, $v = 0$, $p = 1$,
- pre-shock air: $\gamma = 1.4$, $\rho = 1$, $u = 0$, $v = 0$, $p = 1$,
- post-shock air: $\gamma = 1.4$, $\rho = 3.857$, $u = 2.629$, $v = 0$, $p = 10.333$.

The helium bubble of initial radius 0.1 is placed at $(x,y) = (0.5,0.5)$ in a periodic computational domain of $[0,1] \times [0,1]$. The initial level set field is therefore $\phi = \sqrt{(x-0.5)^2 + (y-0.5)^2} - 0.1$ where $\phi < 0$ denotes helium and $\phi \geqslant 0$ is considered as air. The pre-shock air region is defined by $0.25 < x < 1$ (not including the helium bubble) and as a result, $0 < x < 0.25$ will be the post-shock air region.

Non-dimensional time is defined as $\tilde{t} = (T - T_{\text{impact}})Mc_s/r$ where $T$, $T_{\text{impact}}$, $M$, $c_s$ and $r$ are respectively the physical time, the physical time at which the shock impinges the bubble, the Mach number of the shock, the speed of sound in the surrounding air and the initial radius of the helium bubble. Thus $\tilde{t} = 0$ corresponds to the impact time and $\tilde{t} = 1.0$ is approximately the time by which the outside shock has traveled a distance equal to the radius $r$. In our simulations, the number of grid points in each block is set to 32 per direction and we impose the maximum level of resolution to 8, therefore allowing for an effective grid of 8192 points in each direction. Time step is chosen in accordance to Section 4.4 with CFL = 0.25.

*5.2.1. Effects of thresholding the detail coefficients*

A study is done on thresholding the wavelet detail coefficient, $\epsilon$, for the shock-bubble interaction at $M = 3.0$ and values of $\epsilon$ between 0.5 and 0.005. In Table 1, we present the total number of blocks in the adaptive grid as well as the maximum level of resolution ($L_{max}$) reached at $\tilde{t} = 1.0$ and the number of blocks at $L_{max}$. Fig. 8 (left) demonstrates that for this specific problem, the total number of blocks scales quadratically with the threshold. In Fig. 8 (right), the enstrophy, $\mathcal{E} = \int_A |\nabla u|^2 dA$, is plotted for different values of detail thresholds against time. We notice that for thresholds larger than 0.05, the vorticity generation is drastically reduced since the numerical diffusion of the coarse grid reduces the amplitude of especially the pressure gradients whereas for smaller values of thresholds, the solution seems to tend towards a unique one. This can be noticed in Fig. 9, where the density field as well as the adaptive grid is presented for $\epsilon = 0.5$, 0.1, 0.05, 0.01 at $\tilde{t} = 1.0$ to show the effect of smaller thresholds on the emergence of more smaller scales in the flow.

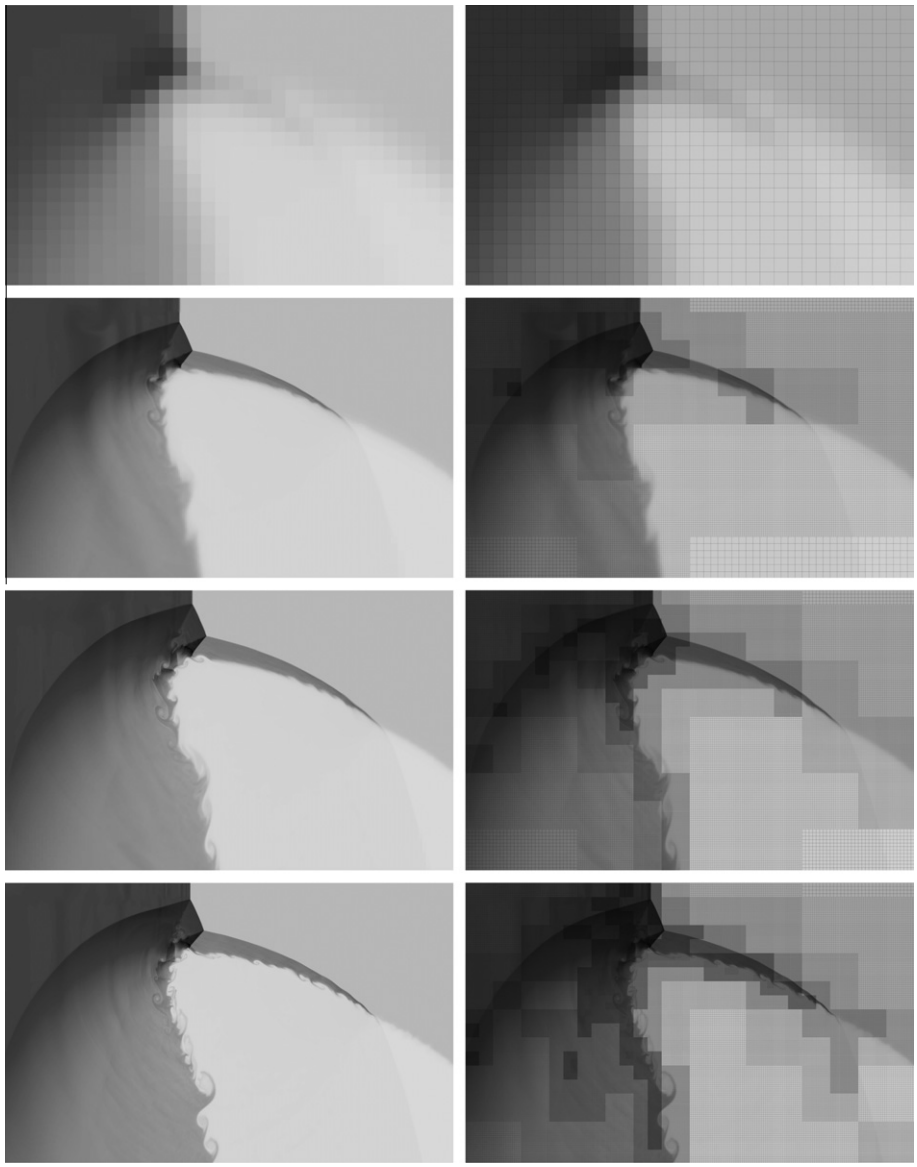

**Fig. 9.** $\epsilon$ = 0.5, 0.1, 0.05, 0.01 (top to bottom). Density field (left) and the adaptive grid (right) close to the bubble.

The reason why there is a loss of enstrophy at around time = 0.07 is due to the grid compression and the fact that detail coefficients are not computed based on the vorticity in our simulations. We have also noticed that after $t > 0.15$ (not shown here), the enstrohpy starts to decay since no significant vorticity generation takes place any longer and more and more energy is dissipated by numerical diffusion and grid coarsening. It should be noted that as we solve the Euler equations of compressible flows, the more we reduce the detail threshold and allow for refinement, the more details i.e. smaller flow structures are generated until the point where numerical diffusion prohibits the emergence of smaller details. Therefore, the reason we converge to a unique solution by reducing the detail threshold is the fact that maximum level of resolution is fixed to 8 and the grid will be saturated in the limit of $\varepsilon \to 0$.

*5.2.2. M = 1.2*

In Fig. 10, we present vorticity and density fields from the simulation at $M = 1.2$. At $\tilde{t} = 0.5$, the original shock wave outside the bubble is connected to the first reflection which is moving away from the surface to the upstream. The speed of sound inside the bubble is higher than that of the surrounding air and the refracted wave travels faster in helium. Having exited from the right half of the bubble, the refracted wave connects to the shock-reflection wave structure outside the bubble creating the so-called twin reflection-refraction system [51]. This happens around $\tilde{t} = 1.0$ when the outside shock has

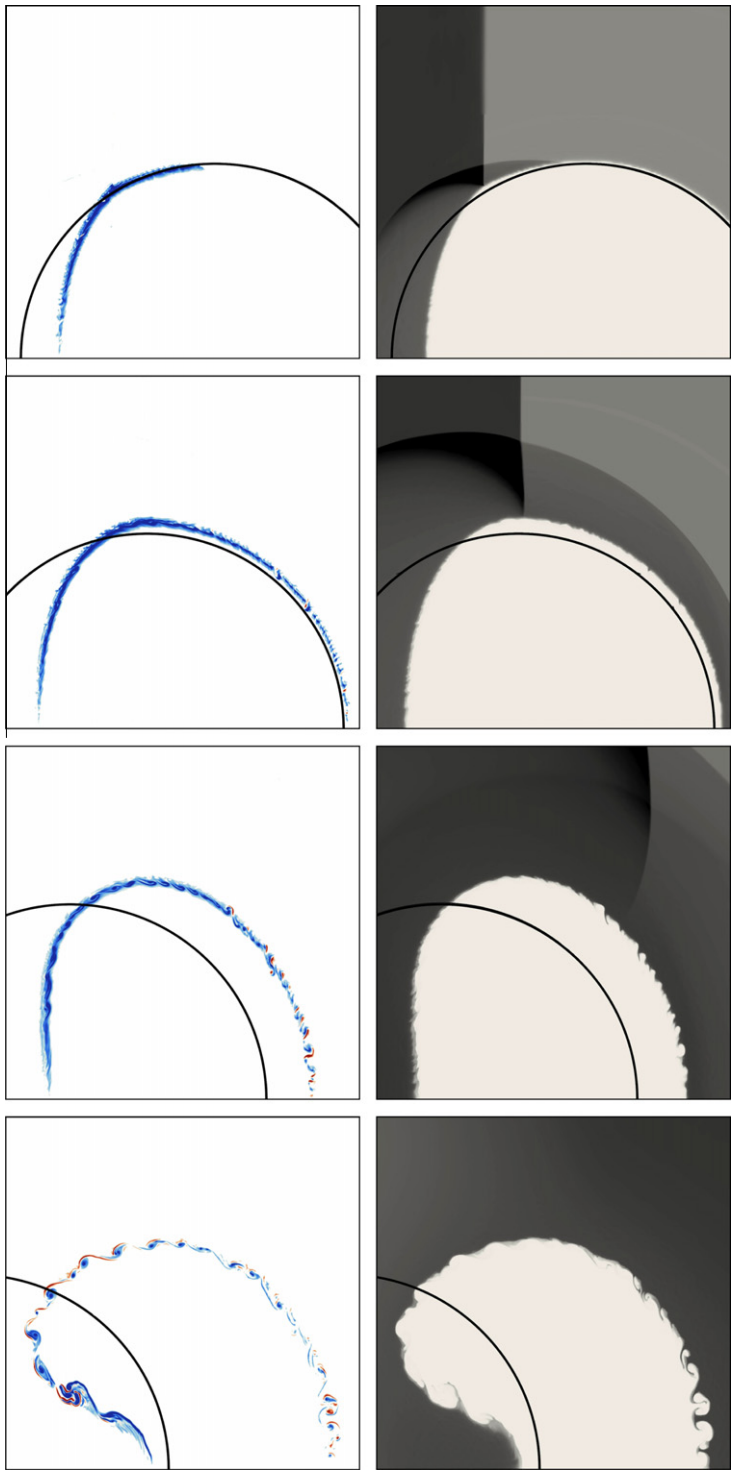

**Fig. 10.** Density (right) and vorticity (left) fields for $M = 1.2$ test case at $\tilde{t} = 0.5$, 1.0, 2.0 and 4.0 (top to bottom). blue/red: positive/negative vorticity, white/black: low/high density. Black solid line shows the initial location of air/helium interface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

traveled about a distance equal to the radius of the bubble. From time $\tilde{t} = 3.0$ to $\tilde{t} = 4.0$, we can observe the development of mushrooms close to the centerline on the downstream side of interface followed by roll-ups over the entire surface better shown in the vorticity plot.

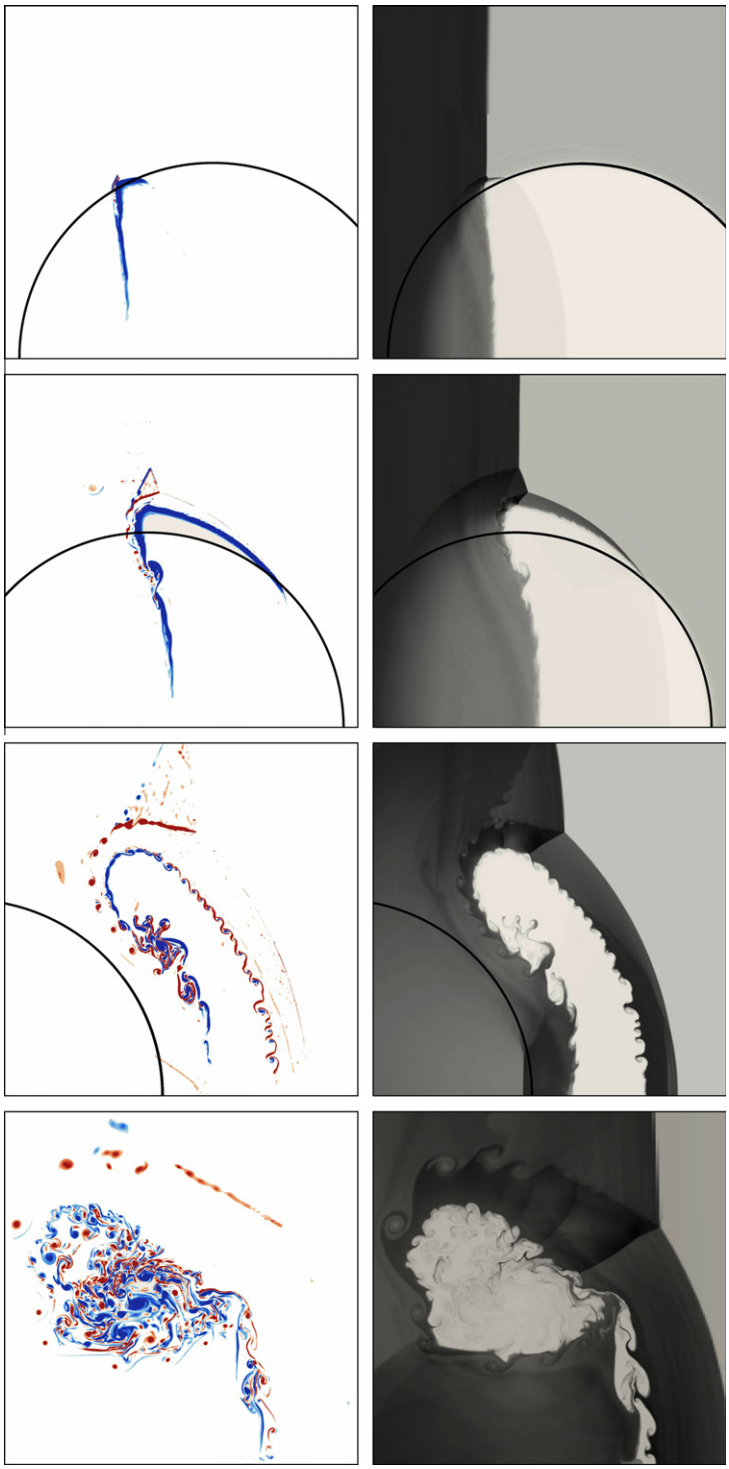

**Fig. 11.** Density (right) and vorticity (left) fields for $M = 3.0$ test case at $\tilde{t} = 0.5$, 1.0, 2.0 and 4.0 (top to bottom). blue/red: positive/negative vorticity, white/black: low/high density. Black solid line shows the initial location of air/helium interface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

### 5.2.3. $M = 3.0$

In Fig. 11, we present vorticity and density fields from the simulation at $M = 3.0$. The transmitted shock inside the bubble is more pronounced than that of the $M = 1.2$ case and is now visible in the density plots at $\tilde{t} = 0.5$ and 1.0. The upstream interface of the bubble is more flattened as the incident shock is stronger than that of the $M = 1.2$ case. A major difference

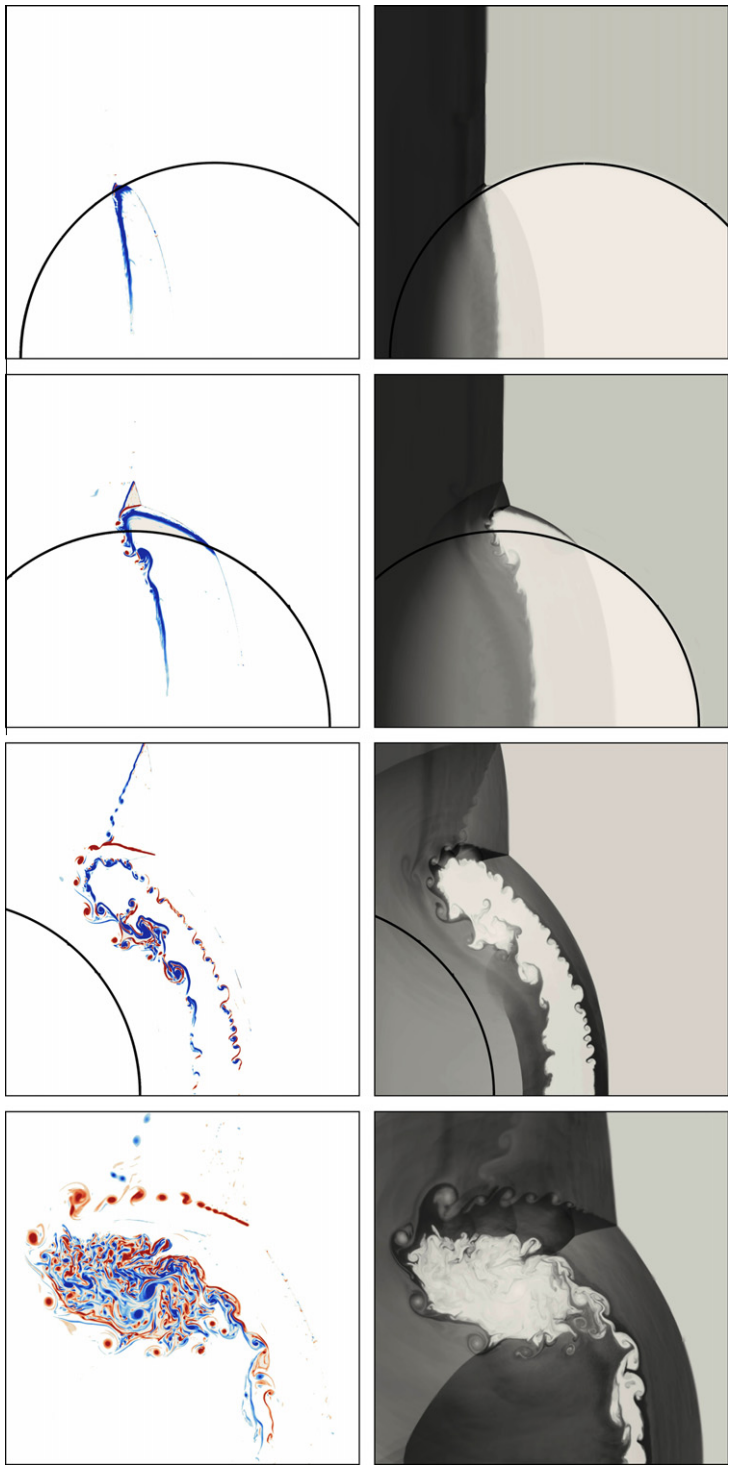

**Fig. 12.** Density (right) and vorticity (left) fields for $M = 6.0$ test case at $\tilde{t} = 0.5$, 1.0, 2.0 and 4.0 (top to bottom). blue/red: positive/negative vorticity, white/black: low/high density. Black solid line shows the initial location of air/helium interface. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

in this Mach number is that the initially reflected wave and the moving shock outside the bubble together with the refracted shock compose a *mach reflection* structure which is already captured at $\tilde{t} = 0.6$, half the time as reported in [52]. The slip-stream after the mach reflection is easier to spot in the vorticity images at $\tilde{t} = 2.0$ and 4.0 where vortices of negative sign

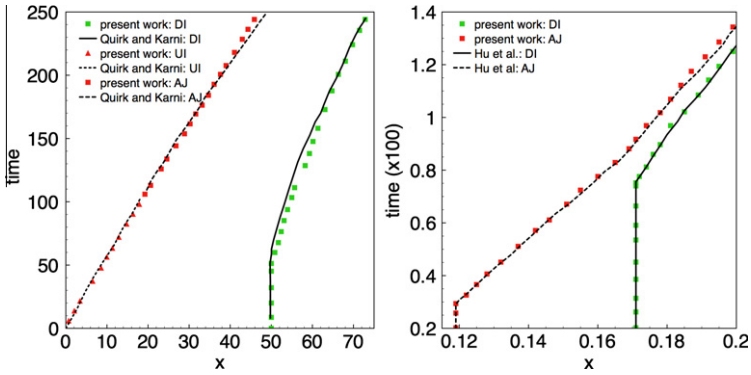


**Fig. 13.** Space–time diagrams of air jet (AJ) and downstream interface (DI) for $M = 1.2$ (left) and $M = 6$ (right). Scales on the axes correspond to those of the references.

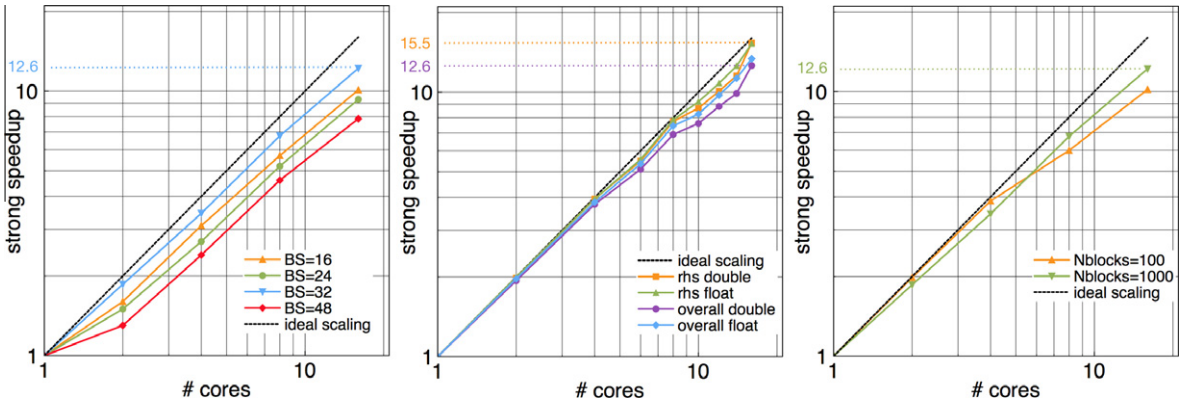


**Fig. 14.** Strong scaling versus number of cores for different number of points per block (left), strong scaling measured for the computation of right hand side and for the overall time step using single and double precision versus number of cores (middle), strong scaling for different number of blocks in the grid versus number of cores (right).

are created and carried downwards to the interior of the bubble and interact with the originally deposited positive vorticity on the interface of the bubble. Another important feature, which to our knowledge has not been reported for the shock-bubble interaction, is the second triple point which emerges along with the first one and is responsible for the creation of positive vorticity. The two triple points diverge further as they move to the downstream, one moving upward and the other downward. A very thin layer of vorticity is also noticed in the images at $\tilde{t} = 0.5$ and 1.0 on the refracted wave which has now partially exited from the downstream interface of the bubble. Close to the centerline of the bubble on the upstream side, another wave structure is formed as the reflection of the transmitted wave from the downstream interface exits the upstream interface shown at $\tilde{t} = 2.0$.

*5.2.4. M = 6.0*

In Fig. 12, we present vorticity and density fields from the simulation at $M = 6.0$. The major features of the flow do not dramatically differ from those of the $M = 3.0$ case as also noted by Bagabir and Drikakis [52]. The transmitted shock is stronger than before and therefore results in some vorticity generation are visible inside the bubble at $\tilde{t} = 0.5$ and 1.0. While the two triple points are at the same distance as the $M = 3.0$ case at $\tilde{t} = 1.0$, they stay closer to each other at the later times e.g. $\tilde{t} = 2.0$ compared to those of the $M = 3.0$ case. The Mach reflection structure also stays closer to the surface of the bubble. The structure on the upstream side of the bubble from the internal reflection of the transmitted wave is also stronger in this case and remains closer to the upstream interface of the bubble as can be observed in the images at $\tilde{t} = 2.0$ and 4.0. We notice that as we increase the Mach number, the volume of the bubble becomes smaller such that at $\tilde{t} = 4.0$, it is 87.2, 36.6 and 26.7% of the initial volume respectively for $M = 1.2$, 3.0 and 6.0.

We compare the velocities obtained by the present method, of the upstream and downstream interface as well as the air jet head in the space–time diagrams of Fig. 13 for $M = 1.2$ (left) and $M = 6$ (right), with the results of [25,26]. The small discrepancy visible in the downstream interface at $M = 1.2$ is probably due to the fact that in [25], the helium is assumed to be contaminated by air (28% in mass) which reduces the specific heat ratio inside the bubble leading to a smaller velocity for the transmitted shock i.e. the first wave to interact with the downstream interface. This difference however starts to vanish as

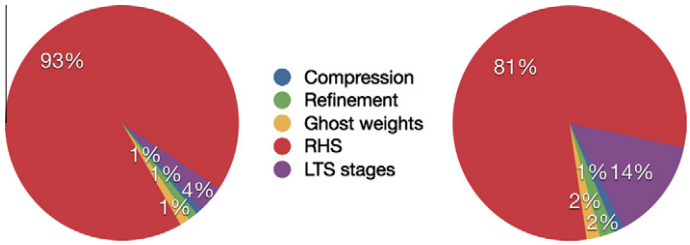


**Fig. 15.** Percentage of different stages in one simulation time step for the execution on 1 core (left) and 16 cores (right).

the outside shock (not shown here) also passes over the downstream interface (around $t = 125$). The upstream interface is no longer plotted after the air jet develops around $t = 100$ for $M = 1.2$. The velocities at $M = 6.0$ case are in very good agreement with those of Hu et al. [26] and since the shock is stronger in this Mach number, the upstream interface and the air jet head are close to each other.

### 5.3. Performance tests

The present method was implemented in a computational framework with C++ using generic programming and object oriented concepts [14]. The Intel's Threading Building Blocks (TBB) [18] library was used to map logical tasks to physical threads. This library allows for specifying task patterns and enables the user to easily express nested parallelism inside tasks. Furthermore, we took extra care to the compute intensive parts of the code by replacing C++ instructions with Intel SSE intrinsics (through C++ template specialization).

The compute nodes used in this work i.e. multi-threaded nodes have 4 quad-core AMD Opteron 8380 processors ("Shanghai" core, 2.5 GHz, 6 MB L3-cache) with 32 GB of RAM.

The simulation of shock-bubble interaction at $M = 3$ (presented in Fig. 11) was performed on one of these nodes using about 40 times less grid points than required by a uniform grid at maximum level of resolution of 8 with $s_{block} = 32$ (i.e. using only 1200 grid points per dimension instead of 8192).

We used the PAPI library [53] to measure the FLOPS of the shock-bubble interaction benchmark with a maximum level of resolution of 7 during the first LTS steps of the simulation. It consists of a total of $5700 \cdot 10^9$ floating point operations. Around $5200 \cdot 10^9$ of these operations were performed for the evaluation of the right hand side. Another $400 \cdot 10^9$ are used in the LTS stages, meaning that the refinement and compression stages are not compute intensive as they took only 1% of the floating point computation. For the right hand side computation, the GFLOPS achieved on one core is 2.19 whereas on 16 cores is 33.16 using double precision. For single precision, the GFLOPS reached on one core is 3.86 compared to 58.81 on 16 cores. We investigate the effect of the number of grid points in each block on the scaling of our simulation software while running on up to 16 cores of a computing node. It can be deduced from Fig. 14 (left) that a block size of 32 gives the best scaling for this set of simulations. In Fig. 14 (middle) we present the speedup measured for the right hand side computation and for the overall time step using single and double precision. The right hand side computation shows a speedup of 15.5 over 16 for both float and double precision executions. The overall speedup achieved by using float is slightly better than that achieved from a double precision execution. Furthermore, we show how the parallel efficiency improves as the number of blocks are increased in Fig. 14 (right). The increase in the number of blocks is achieved by reducing the wavelet detail threshold and not by reducing the block size. An extensive performance study on the type of wavelets, block size and number of blocks can be found in [14]. Moreover, we have noticed that the LTS provides us with a speedup of about 24 in the simulation of shock-bubble interaction with around 1000 blocks in the adaptive grid and therefore, compared to a single core execution of the same adaptive simulation with global time-stepping, we have achieved a speedup of 288 by incorporating parallelism and LTS.

In Fig. 15 we show the percentage of execution time spent in different stages of one simulation time step. The execution time is divided into five categories namely compression, refinement, ghost weights, right hand side computation and the LTS stages. Right hand side and LTS stages represent the computing part of the solver. Compression consists of thresholding the detail coefficients and collapsing the associated blocks. Refinement is responsible for splitting the blocks where new scales are expected to emerge. In the category "ghost weights" we compute and store the necessary weights needed to reconstruct the ghosts ($w_{ij}$ as in Section 4.2). We also observe that the scalability of the LTS stages is slightly less than that of the others. We attribute this to the low computing intensity of the operations involved in those stages. As expected, computing the right hand side takes the major part of the computation time in both single core and 16-core executions. Given this and the fact that this part of the computation is highly compute intensive (91% of the FLOPS), we deduce that by further increasing the number of cores we could still substantially reduce the execution time of the solver.

We compared the solver presented in this work to its single core adaptive "plain-C++" version that does not employ LTS schemes. We observed that with the use of SSE intrinsics we get an overall improvement of 1.8X. Another 12X is achieved by employing task-based parallelism on 16 cores and 24X is the gain in performance provided by the algorithmic improvement of LTS schemes. By combining the three techniques we arrive at an overall improvement of 518X.

## 6. Conclusions

We presented a wavelet-based space–time adaptive finite volume solver for single- and multi-phase compressible flows. The method was validated on 1D benchmark problems and on high resolution simulation of shock-bubble interaction at different Mach numbers. The solver has enabled to resolve the behavior of deposited vorticity on the bubble interface, revealing the subsequent instabilities on the surface and the detailed vortical structures generated by the wave structures around the interface of the bubble.

The solver couples the space-adapted grids with high-order finite volume schemes. Furthermore, the solver is time adaptive as it efficiently performs LTS schemes (first and second order accurate) by exploiting a straightforward technique presented in this work. The employment of LTS schemes changes the complexity of the solver to a great extent by drastically reducing the number of flux evaluations needed to reach a desired simulation time. By introducing the wavelet blocks, we have demonstrated that it is possible to exploit the spatial and temporal adaptivity without jeopardizing the computational efficiency on multicore architectures. We demonstrate the change in performance based on the size of the block and the number of cores. We obtain the best performances in terms of scaling using a block size of 32 which gave a strong speedup of about 12 over 16 cores. We observe that the performance differences between different block sizes are independent of the number of cores, implying that good efficiency is expected also for more than 16 cores.

Present work involves the simulations of 3D flows with complex, deforming boundaries.

## Appendix A. Wavelet-based adaptivity

Biorthogonal wavelets can be used to construct multiresolution analysis (MRA) of the quantities of interest and they are combined with finite difference/volume approximations to discretize the governing equations. Biorthogonal wavelets are a generalization of orthogonal wavelets and they can have associated scaling functions that are symmetric and smooth [46]. Biorthogonal wavelets introduce two pairs of functions $\phi$, $\psi$ for the synthesis, and $\tilde{\phi}$, $\tilde{\psi}$ for analysis.

Given a signal in the physical space, the functions $\tilde{\phi}$, $\tilde{\psi}$ are used in the *forward* wavelet transform, where one computes the wavelet coefficients. The functions $\phi$, $\psi$ are used in the *inverse* wavelet transform, where one reconstructs the signal in the physical space from the wavelet coefficients. The functions $\phi$, $\psi$, $\tilde{\phi}$, $\tilde{\psi}$ introduce four refinement equations:

$$\phi(x) = \sum_m h_m^S \phi(2x+m), \quad \psi(x) = \sum_m g_m^S \phi(2x+m) \tag{29}$$

$$\tilde{\phi}(x) = \sum_m h_m^A \tilde{\phi}(2x+m), \quad \tilde{\psi}(x) = \sum_m g_m^A \tilde{\phi}(2x+m) \tag{30}$$

For the case of average interpolating wavelets, $\tilde{\phi}(x) = T(x/2)/2$, $\psi(x) = -T(x) + T(x-1)$ where $T$ is the "top-hat" function. Because of their average-interpolating properties, the functions $\tilde{\psi}$ and $\phi$ are not known explicitly in analytic form.

The forward wavelet transform computes two types of coefficients: the scaling $\{c_k^l\}$ and detail coefficients $\left\{d_k^l\right\}$. From the scaling and details coefficients of $f$ obtained in the forward transform, we can reconstruct $f$ as follows:

$$f = \sum_k c_k^0 \phi_k^0 + \sum_{l=0}^{L} \sum_k d_k^l \psi_k^l \tag{31}$$

where $\phi_k^l = \phi(2^l x - k)$ and $\psi_k^l(x) = \psi(2^l x - k)$.

If $f$ is uniformly discretized in space with cell averages $\{f_i\}$, we can find $\{c_k^0\}$, $\left\{d_k^l\right\}$ by first considering the finest scaling coefficients to be $c_k^L = f_k$ and then perform the full Fast Wavelet Transform (FWT), by repeating the following step:

$$c_k^l = \sum_m h_{2k-m}^A c_m^{l+1} \tag{32}$$

$$d_k^l = \sum_m g_{2k-m}^A c_m^{l+1} \tag{33}$$

for $l$ from $L-1$ to 0. To reconstuct $\{f_i\}$ we use the fast inverse wavelet transform, that repeats the following step for $l$ from 0 to $L-1$

$$c_k^{l+1} = \sum_m h_{2m-k}^S c_m^l + \sum_m g_{2m-k}^S d_m^l \tag{34}$$

### A.1. Active scaling coefficients

Using the FWT we can decompose functions into scaling and detail coefficients, resulting in an MRA of our data. We can now exploit the scale information of the MRA to obtain a compressed representation by keeping only the coefficients that carry significant information

$$f_{\geqslant \epsilon} = \sum_k c_k^0 \phi_k^0 + \sum_l \sum_{k:|d_k|>\epsilon} d_k^l \psi_k^l \tag{35}$$

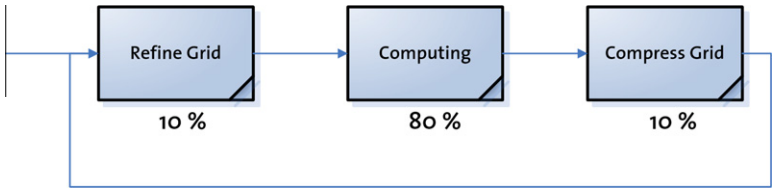


**Fig. 16.** In order to capture the emerging small scales, refinement of the grid is performed before each simulation step. In the computing stage, the solver evaluates the right hand side of the PDE and evolves the solution updating the grid points. The compression step discards the negligible grid points by looking at the new detail coefficients. It is desired that the computing stages takes most of the execution time (e.g. around 80%).

where $\varepsilon$ is called detail threshold and is used to truncate terms in the reconstruction. The scaling coefficients $c_k^l$ needed to compute $d_k^l : \left|d_k^l\right| > \epsilon$, and the coefficients at coarser levels needed to reconstruct $c_k^l$ are the active scaling coefficients. The pointwise error introduced by this thresholding is bounded by $\epsilon$. Each scaling coefficient has a physical position and therefore the above compression results in an adapted grid $\mathcal{G}$, where each grid point will represent an active scaling coefficient, representing a physical quantity.

We then discretize the differential operators by applying standard finite volume or finite-difference schemes on the active coefficients. Such operators can be viewed as (non-linear) filtering operations on uniform resolution grid points, formally:

$$F(c_k^l) = \sum_{j=s_f}^{e_f-1} c_{k+j}^l \beta_j, \quad \beta_j^l \text{ function of } \{c_m^l\} \tag{36}$$

where $\{s_f, e_f - 1\}$ is the support of the filter, and the filter coefficients are $\{\beta_j\}$. In order to avoid any modification to the finite volume or finite-difference schemes, we construct a locally uniform grid by means of the ghosts. The details of the ghost reconstruction is described in Section 4.2.

### A.2. Block splitting and collapsing in 1D

The adaptation of the grid is achieved by performing elementary operations on the blocks: block splitting to locally refine the grid, and block collapsing to coarsen the grid. Block splitting and block collapsing is triggered by logic expressions based on the thresholding of detail coefficients residing inside the block. In the one dimensional case, when we decide to collapse some blocks into one, we just have to replace the data with the scaling coefficient at the coarser level

$$c_k^l = \sum_m h_{2k-m}^A c_m^{l+1} k \in \left\{ \frac{i_b}{2} \cdot s_{block}, \ \left(\frac{i_b}{2} + 1\right) \cdot s_{block} - 1 \right\} \tag{37}$$

with $i_b$ being the index of the block. Note that $k$ is inside the block but $m$ can be outside the block. If we consider the block collapse as the key operation for compressing, we can consider the block splitting as the key to capture smaller emerging scales. In the case where we split one block into two blocks, we perform one step of the inverse wavelet transform on the block $(i_b, l_b)$

$$c_k^{l_b+1} = \sum_m h_{2m-k}^S c_m^{l_b}, \quad k \in \{2i_b \cdot s_{block}, \ 2(i_b+1) \cdot s_{block} - 1\} \tag{38}$$

In our work, grid adaptation is performed in two steps: before the computing stage we refine the grid in order to allow for the emergence of new smaller scales [54] and after the computation we apply the compression based on thresholding to retain only the blocks with significant details. It is also desirable that most of the execution time is spent in solving the PDE and not in refining/compressing the grid (Fig. 16).

## References

[1] P. Chatelain, A. Curioni, M. Bergdorf, D. Rossinelli, W. Andreoni, P. Koumoutsakos, Billion vortex particle direct numerical simulation of aircraft wakes, Computer Methods in Applied Mechanics and Engineering 197 (13–16) (2008) 1296–1304.
[2] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, L.C. Wilcox, Parallel scalable adjoint-based adaptive solution of variable-viscosity stokes flow problems, Computer Methods in Applied Mechanics and Engineering 198 (21–26) (2009) 1691–1700. Advances in Simulation-Based Engineering Sciences – Honoring J. Tinsley Oden.
[3] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, Journal of Computational Physics 53 (3) (1984) 484–512.
[4] F. Miniati, P. Colella, Block structured adaptive mesh and time refinement for hybrid, hyperbolic plus N-body systems, Journal of Computational Physics 227 (1) (2007) 400–430.
[5] A. Harten, Adaptive multiresolution schemes for shock computations, Journal of Computational Physics 115 (2) (1994) 319–338.
[6] O. Vasilyev, S. Paolucci, A dynamically adaptive multilevel wavelet collocation method for solving partial differential equations in a finite domain, Journal of Computational Physics 125 (2) (1996) 498–512.
[7] R. Abgrall, A. Harten, Multiresolution representation in unstructured meshes, SIAM Journal on Scientific Computing 35 (6) (1998) 2128–2146.
[8] N.K.R. Kevlahan, O.V. Vasilyev, An adaptive wavelet collocation method for fluid-structure interaction at high reynolds numbers, SIAM Journal on Scientific Computing 26 (6) (2005) 1894–1915.

[9] Qianlong Liu, O. Vasilyev, A Brinkman penalization method for compressible flows in complex geometries, Journal of Computational Physics 227 (2) (2007) 946–966.
[10] K. Schneider, O.V. Vasilyev, Wavelet methods in computational fluid dynamics, Annual Review of Fluid Mechanics 42 (1) (2010) 473–503.
[11] M. Bergdorf, P. Koumoutsakos, A Lagrangian particle-wavelet method, Multiscale Modeling and Simulation 5 (3) (2006) 980–995.
[12] J.D. Regele, O.V. Vasilyev, An adaptive wavelet-collocation method for shock computations, International Journal of Computational Fluid Dynamics 23 (7) (2009) 503–518.
[13] M.O. Domingues, S.M. Gomes, O. Roussel, K. Schneider, Space-time adaptive multiresolution methods for hyperbolic conservation laws: Applications to compressible euler equations, Second Chilean Workshop on Numerical Analysis of Partial Differential Equations, Concepcion, Chile, January 16–19, 2007, Applied Numerical Mathematics 59 (9) (2009) 2303–2321.
[14] D. Rossinelli, M. Bergdorf, B. Hejazialhosseini, P. Koumoutsakos, Wavelet-based adaptive solvers on multi-core architectures for the simulation of complex systems, in: Euro-Par'09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 721–734.
[15] Board, O.A.R.: Openmp application program interface, Technical report, OpenMP Architecture Review Board, October, 2007.
[16] F. Bodin, S. Bihan, Heterogeneous multicore parallel programming for graphics processing units, Scientific Programming 17 (4) (2009) 325–336.
[17] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, Y. Zhou, Cilk – An efficient multithreaded runtime system, 5Th ACM Sigplan Symposium On Principles And Practice Of Parallel Programming, Santa Barbara, Ca, July 19–21, 1995, Sigplan Notices 30 (8) (1995) 207–216.
[18] G. Contreras, M. Martonosi, Characterizing and improving the performance of Intel threading building blocks, in: 2008 IEEE International Symposium on Workload Characterization (IISWC), Piscataway, NJ, USA, IEEE, 2008, pp. 57–66.
[19] D. Leijen, W. Schulte, S. Burckhardt, The design of a task parallel library, Acm Sigplan Notices 44 (10) (2009) 227–241.
[20] R.D. Blumofe, C.E. Leiserson, Scheduling multithreaded computations by work stealing, Journal of the ACM 46 (5) (1999) 720–748.
[21] A. Robison, M. Voss, A. Kukanov, Optimization via reflection on work stealing in TBB, in: 2008 IEEE International Symposium on Parallel & Distributed Processing, vols 1–8, 345 E 47TH ST, New York, NY 10017 USA, IEEE, IEEE, 2008, pp. 598–605.
[22] R. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge University Press, 2002.
[23] J. Haas, B. Sturtevant, Interaction of weak shock-waves with cylindrical and spherical gas inhomogeneities, Journal of Fluid Mechanics 181 (August) (1987) 41–76.
[24] J. Picone, J. Boris, Vorticity generation by shock propagation through bubbles in a gas, Journal of Fluid Mechanics 189 (April) (1988) 23–51.
[25] J. Quirk, S. Karni, On the dynamics of a shock-bubble interaction, Journal of Fluid Mechanics 318 (Jul) (1996) 129–163.
[26] X.Y. Hu, B.C. Khoo, N.A. Adams, F.L. Huang, A conservative interface method for compressible flows, Journal of Computational Physics 219 (2) (2006) 553–578.
[27] A. Prosperetti, G. Tryggvason (Eds.), Computational Methods for Multiphase Flow, Cambridge University Press, Cambridge, 2007.
[28] S. Osher, J. Sethian, Fronts propagating with curvature-dependent speed – Algorithms based on Hamilton–Jacobi formulations, Journal of Computational Physics 79 (1) (1988) 12–49.
[29] M. Sussman, P. Smereka, S. Osher, A level set approach for computing solutions to incompressible 2-phase flow, Journal of Computational Physics 114 (1) (1994) 146–159.
[30] G. Russo, P. Smereka, A remark on computing distance functions, Journal of Computational Physics 163 (1) (2000) 51–67.
[31] E.F. Toro, Rieman Solvers and Numerical Methods for Fluid Dynamics, Springer-Verlag, Berlin, 1999. Includes references and index.
[32] P. Roe, Approximate Riemann solvers, parameter vectors and difference-schemes, Journal of Computational Physics 43 (2) (1981) 357–372.
[33] P. Lax, Weak solutions of nonlinear hyperbolic equations and their numerical computation, Communication on Pure and Applied Mathematics 7 (1) (1954) 159–193.
[34] A. Harten, P. Lax, B. Van Leer, On upstream differencing and Godunov-type schemes for hyperbolic conservation laws, SIAM Review 25 (1) (1983) 35–61.
[35] B. Einfeldt, On Godunov-type methods for gas-dynamics, SIAM journal on numerical analysis 25 (2) (1988) 294–318.
[36] W. Anderson, J. Thomas, B. Vanleer, Comparison of finite volume flux vector splittings for the euler equations, AIAA Journal 24 (9) (1986) 1453–1460.
[37] P. Colella, P. Woodward, The piecewise parabolic method (PPM) for gas-dynamical simulations, Journal of Computational Physics 54 (1) (1984) 174–201.
[38] A. Harten, B. Engquist, S. Osher, S. Chakravarthy, Uniformly high order accurate essentially non-oscillatory schemes.3. (Reprinted from Journal of Computational Physics, vol 71, pg 231, 1987), Journal of Computational Physics 131 (1) (1997) 3–47.
[39] R. Abgrall, How to prevent pressure oscillations in multicomponent flow calculations: A quasi conservative approach, Journal Of Computational Physics 125 (1) (1996) 150–160.
[40] E. Johnsen, T. Colonius, Implementation of WENO schemes in compressible multicomponent flow problems, Journal of Computational Physics 219 (2) (2006) 715–732.
[41] A. Mignone, T. Plewa, G. Bodo, The piecewise parabolic method for multidimensional relativistic fluid dynamics, Astrophysical Journal Supplement Series 160 (1) (2005) 199–219.
[42] X. Liu, S. Osher, T. Chan, Weighted essentially nonoscillatory schemes, Journal of Computational Physics 115 (1) (1994) 200–212.
[43] G. Jiang, C. Shu, Efficient implementation of weighted ENO schemes, Journal of Computational Physics 126 (1) (1996) 202–228.
[44] J. Williamson, Low-storage Runge–Kutta schemes, Journal of Computational Physics 35 (1) (1980) 48–56.
[45] R. Saurel, R. Abgrall, A simple method for compressible multifluid flows, SIAM Journal on Scientific Computing 21 (3) (1999) 1115–1145.
[46] A. Cohen, I. Daubechies, J. Feauveau, Biorthogonal bases of compactly supported wavelets, Communication on Pure and Applied Mathematics 45 (5) (1992) 485–560.
[47] M.O. Domingues, S.M. Gomes, O. Roussel, K. Schneider, An adaptive multiresolution scheme with local time stepping for evolutionary pdes, Journal Of Computational Physics (2008).
[48] J.M. Alam, N.K.R. Kevlahan, O.V. Vasilyev, Simultaneous space–time adaptive wavelet solution of nonlinear parabolic differential equations, Journal of Computational Physics 214 (2) (2006) 829–857.
[49] O. Roussel, K. Schneider, A. Tsigulin, H. Bockhorn, A conservative fully adaptive multiresolution algorithm for parabolic pdes, Journal of Computational Physics 188 (2) (2003) 493–523.
[50] A. Calder, B. Fryxell, T. Plewa, R. Rosner, L. Dursi, V. Weirs, T. Dupont, H. Robey, J. Kane, B. Remington, R. Drake, G. Dimonte, M. Zingale, F. Timmes, K. Olson, P. Ricker, P. MacNeice, H. Tufo, On validating an astrophysical simulation code, Astrophysical Journal Supplement Series 143 (1) (2002) 201–229.
[51] L. Henderson, P. Colella, E. Puckett, On the refraction of shock-waves at a slow fast gas interface, Journal of Fluid Mechanics 224 (March) (1991) 1–27.
[52] A. Bagabir, D. Drikakis, Mach number effects on shock-bubble interaction, Shock Waves 11 (3) (2001) 209–218.
[53] S. Browne, J. Dongarra, N. Garner, G. Ho, P. Mucci, A portable programming interface for performance evaluation on modern processors, International Journal of High Performance Computing Application 14 (3) (2000) 189–204.
[54] J. Liandrat, P. Tchamitchian, Resolution of the 1D regularized Burgers equation using a spatial wavelet approximation, Technical Report 90-83, 1CASE, December 1990, NASA Contractor Report 18748880.