CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* (2010)
View this article online at wileyonlinelibrary.com. DOI: 10.1002/cpe.1639

# Wavelet-adaptive solvers on multi-core architectures for the simulation of complex systems

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

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

SUMMARY

We build wavelet-based adaptive numerical methods for the simulation of advection-dominated flows that develop multiple spatial scales, with an emphasis on fluid mechanics problems. Wavelet-based adaptivity is inherently sequential and in this work we demonstrate that these numerical methods can be implemented in software that is capable of harnessing the capabilities of multi-core architectures while maintaining their computational efficiency. Recent designs in frameworks for multi-core software development allow us to rethink parallelism as task-based, where parallel tasks are specified and automatically mapped onto physical threads. This way of exposing parallelism enables the parallelization of algorithms that were considered inherently sequential, such as wavelet-based adaptive simulations. In this paper we present a framework that combines wavelet-based adaptivity with the task-based parallelism. We demonstrate the promising performance obtained by simulating various physical systems on different multi-core architectures using up to 16 cores. Copyright © 2010 John Wiley & Sons, Ltd.

Received 16 November 2009; Revised 1 May 2010; Accepted 30 May 2010

KEY WORDS: wavelets; multi-core; task-based parallelism

## 1. INTRODUCTION

The physically accurate simulation of advection-dominated processes is a challenging computational problem. Advection is the main driver in the simulation of fluid motion in computational fluid dynamics (CFD), or in the animation of complex geometries using level sets in computer graphics. The difficulty arises from the simultaneous presence of a broad range of length scales, e.g. fine geometric features of a 3D model and their non-linear interactions, e.g. in compressible

*Correspondence to: Petros Koumoutsakos, Chair of Computational Science, ETH Zürich, Zürich CH-8092, Switzerland.
†E-mail: petros@ethz.ch

Contract/grant sponsor: Publishing Arts Research Council; contract/grant number: 98-1846389

flows. Presently, the workhorse approach in simulating multiscale flow phenomena such as turbulent flows are Direct Numerical Simulations (DNS) [1] which use large uniform grids to resolve all the present scales within the flow. Large DNS calculations are performed on massively parallel computing architectures and compute the evolution of tens of billions of unknowns [2, 3].

DNS is not a viable solution for the simulation of flows of engineering interest [4] albeit the continuous increase in available high performance computers. Adaptive simulation techniques, such as Adaptive Mesh Refinement (AMR) [5], or multiresolution techniques using Wavelets [6–8] have been introduced to locally adjust the resolution of the computational elements to the different length scales emerging in the flow field. Wavelets have been employed largely for conservation laws but they have also been recently coupled to level sets for geometry representation [9].

In order to create simulation tools that go beyond the state-of-the-art, these adaptive numerical methods must be complemented with massively parallel and multi-level parallelism. Hardware-accelerated wavelet transforms have a long history. The first GPU-based 2D FWT was introduced by Hopf and Ertl in 2000 [10]. Since then, many different parallel implementations have been proposed and evaluated [11] both on multi-core architectures such as the Cell BE [12] and accelerators such as GPUs [13]. These efforts, however, have been restricted to the *full* FWT, where one computes *all* the detail coefficients and performs the subsequent signal processing in the wavelet space.

Wavelet-based adaptive partial differential equation (PDE) solvers can be very different than signal processing techniques, where full FWTs are performed. First of all, it is not known in advance how the solution will evolve, therefore, full time-space FWTs cannot be employed. Another difference is that nonlinear operators involving finite difference stencils (for example the convection term in the fluid flow equations) need to be efficiently performed, which is not always possible when working directly in the wavelet space. For this reason sometimes it is meaningful to keep the representation of the solution in the physical space, and use the detail coefficients only to readapt the grid. In this approach not every detail coefficient is needed but only the finest ones. When the solution is represented in the physical space, a third difference is introduced: the choice of wavelets is limited to the biorthogonal ones associated with symmetric and smooth scaling functions (for synthesis). Conventional choices are average interpolating wavelets, interpolating wavelets (first and second generation) or B-splines. The fourth difference is that the grid has to undergo the refinement and compression very frequently as extra care is needed to capture all the emerging scales.

The effective parallel implementation of adaptive wavelet-based methods is hindered by their inherently sequential-nested structure. This difficulty limits their effective implementation on multi-core and many-core architectures and affects the development of per-thread-based software that can have both high performance and abstracts from a specific hardware architecture. An alternative approach for the effective implementation of wavelets for flow simulations is to specify parallel tasks instead of threads and then let an external library map logical tasks to physical threads according to the specific hardware architecture. Another emerging need in developing simulation software is the necessity to specify more than one granularity level for parallel tasks in order to combine multi-core computing with many-core accelerators such as GPUs.

In this paper we present multiresolution wavelet-based computational methods designed for different multi-core architectures. The resulting computational framework is flexible and can be used to simulate different phenomena, such as transport of interfaces, reaction–diffusion problems and compressible flows.

The paper is organized as follows: first we briefly introduce wavelets followed by the presentation of our algorithms for the discretization of PDEs. We discuss how the framework can be



DOI: 10.1002/cpe

employed for multi-core and SMP machines. The performance of the proposed methodology is then demonstrated on computations of level set-based advection of interfaces, two-dimensional multiphase compressible flows and reaction–diffusion problems.

## 2. WAVELET-BASED ADAPTIVE GRIDS

Biorthogonal wavelets can be used to construct multiresolution analysis (MRA) of the quantities being represented 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 [14]. Biorthogonal wavelets introduce two pairs of functions, $\phi$, $\psi$ for synthesis, and $\tilde{\phi}$, $\tilde{\psi}$ for analysis.
There are four refinement equations:

$$\phi(x)=\sum_m h_m^S \phi(2x+m), \quad \psi(x)=\sum_m g_m^S \phi(2x+m), \tag{1}$$

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

Given a function $f$, we compute $c_k^0=\langle f,\tilde{\phi}_k^0\rangle$, $d_k^l=\langle f,\tilde{\psi}_k^l\rangle$ where $\{\tilde{\phi}_k^0\}$ are the analysis scaling functions at the coarsest resolution and $\{\tilde{\psi}_k^l\}$ are the analysis wavelet functions at all the levels. The reconstruction of $f$ follows:

$$f=\sum_k c_k^0 \phi_k^0+\sum_{l=0}^{\infty}\sum_k d_k^l \psi_k^l. \tag{3}$$

The Fast Wavelet Transform (FWT) uses the filters $h_n^A$, $g_n^A$ for analysis whereas $h_n^S$, $g_n^S$ are used in the inverse FWT for synthesis,

$$c_k^l=\sum_m h_{2k-m}^A c_m^{l+1}, \quad d_k^l=\sum_m g_{2k-m}^A c_m^{l+1}, \tag{4}$$

$$c_k^{l+1}=\sum_m h_{2m-k}^S c_m^l+\sum_m g_{2m-k}^S d_m^l. \tag{5}$$

### 2.1. 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 (details):

$$f_{\geq\varepsilon}=\sum_k c_k^0 \phi_k^0+\sum_l \sum_{k:|d_k|\geq\varepsilon} d_k^l \psi_k^l, \tag{6}$$

where $\varepsilon$ is the compression threshold and is used to truncate relatively insignificant terms in the reconstruction. The scaling coefficients $c_k^l$ needed to compute $d_k^l$ such that $|d_k^l|\geq\varepsilon$, and



DOI: 10.1002/cpe

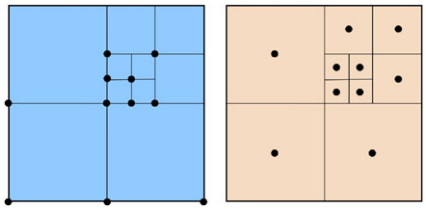


Figure 1. Different types of 2D adapted grids with the same number of active coefficients, for vertex–centered (left) and cell-centered (right) displaced scaling coefficients.

the coefficients at coarser levels needed to reconstruct $c_k^l$ are called active scaling coefficients. The pointwise error introduced by this thresholding is bounded by $\varepsilon$. Each scaling coefficient has a physical position. Therefore the above compression results in an adapted grid $\mathscr{K}$, where each grid node is an active scaling coefficient (Figure 1). We then discretize the differential operators by applying standard finite-volume or finite-difference schemes on the active coefficients. One way to retain simple operations is to first create a local, uniform resolution neighborhood for a grid point, and then apply the scheme on it. Such operators can be viewed as (non-linear) filtering operations on uniform resolution grid points, formally:

$$F(\{c_{k'}^l\})_k = \sum_{j=s_f}^{e_f-1} c_{k+j}^l \beta_j, \quad \beta_j \text{ function of } \{c_{k'}\} \tag{7}$$

where $\{s_f, e_f-1\}$ is the support of the filter in the index space and $e_f - s_f$ is the number of non-zero filter coeffcients $\{\beta_j\}$. In order to perform uniform resolution filtering we need to temporarily introduce artificial auxiliary grid points, the so-called 'ghosts'. We need to ascertain that for every grid point k in the adapted set $\mathscr{K}$, its neighborhood $[k-s_f, k+e_f-1]$ is filled with either other points $k'$ in $\mathscr{K}$ or ghosts $g$. Using this set of ghosts, which can be precomputed and stored or computed on the fly (see Section 2.1), we are now able to apply the filter F to all the points $k$ in $\mathscr{K}$. A ghost is 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 refinement equations (1). 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 and $j$ represents the identifier for the source point $p_j$ which is an active scaling coefficient in the grid. Calculation of the weights $\{w_{ij}\}$ is done by traversing a dependency graph associated with the refinement equations (see Figure 2). This operation can be expensive for two reasons: first, if the dependency graph has loops, we need to solve a linear system of equations to compute $\{w_{ij}\}$ and second, the complexity of calculating the values $\{w_{ij}\}$ scales with the number of dependency edges, i.e. it grows exponentially with the jump in level between a point $k$ and its (non-ghost) neighbors in the adapted grid. The wavelets incorporated into the present framework are based on subdivision schemes and either interpolate the function values or their averages [15]. Due to their construction, these wavelets do not explicitly exist in an analytic form. The primal scaling function can, however, be constructed by a recursive scheme imposed by its refinement equation. By virtue of their interpolatory property, the ghost reconstruction is straightforward (the ghost dependency graphs do not contain loops) and leads to very efficient reconstruction formulae.



DOI: 10.1002/cpe

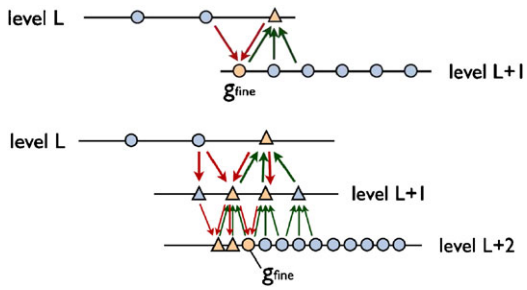



Figure 2. Dependency graph for the ghost reconstruction $g_{\text{fine}}$ with a jump in resolution of 1 (top) and 2 (bottom), for the case of a linear, non-interpolating, biorthogonal wavelet.The ghost generates secondary (triangles), temporary ghosts to calculate the needed weights and source points. The unknown ghosts are the ones with double arrows.

### 2.2. *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 which can be achieved when thresholding individual detail coefficients. The drawback on the other hand is the large amount of sequential operations they involve and the number of indirections (or read instructions) they need in order 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 such as refining or coarsening single grid points have to be performed and those operations are relatively complex and strictly sequential. To expose more parallelism and to decrease the amount of sequential operations per grid point, the key idea is to simplify the data structure. We address this issue by decreasing the granularity of the method at the expense of a reduction in the compression rate. We introduce the concept of block of grid points, which has a coarser granularity by 1 or 2 orders of magnitude with respect to the number of scaling coefficients in every direction, i.e. in 3D, the granularity of the block is coarser by 3 to 5 orders of magnitude with respect to a single scaling coefficient.

*2.2.1. Structure of the wavelet blocks.* A block contains scaling coefficients which reside on the same level and every block contains the same number of scaling coefficients. The grid is then represented with a tree which contains blocks as leaves (see Figure 3). The blocks are nested so that every block can be split and doubled in each direction and blocks can be collapsed into one. Blocks are interconnected through the ghosts. In the physical space the blocks have varying size and therefore different resolutions. The idea of introducing the intermediate concept of a block provides a series of benefits. The first benefit is that tree operations are now accelerated as they can be performed in $\log_2(N^{1/D}/s_{\text{block}})$ operations instead of $\log_2(N^{1/D})$, where $N$ is the total number of active coefficient, $D$ the dimensions in consideration and $s_{\text{block}}$ is the block size. The second benefit is that the random access at elements inside a block can be extremely efficient because the block represents the atomic element. Another very important 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 access) of filtering per grid point with a filter of size $w_{\text{stencil}}D$



DOI: 10.1002/cpe

($c = w_{\text{stencil}} D$). In a uniform resolution grid the data access operations to perform the computation is proportional to $w_{\text{stencil}} D$. For a grid represented by a fine-grained tree, the number of accesses is proportional to $c = w_{\text{stencil}} D \log_2(N^{1/D})$, due to the sequential access of the tree. Using the wavelet blocks approach and assuming that $s_{\text{block}}$ is roughly 1 order of magnitude larger than $w_{\text{stencil}}$, the ratio of ghosts needed per grid point in order to perform the filtering for a block is:

$$r = \frac{(s_{\text{block}} + w_{\text{stencil}})^D - s_{\text{block}}^D}{s_{\text{block}}^D} \approx D \frac{w_{\text{stencil}}}{s_{\text{block}}}. \tag{8}$$

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

$$\begin{aligned} c(w_{\text{stencil}}) &= (1-r) w_{\text{stencil}} D + r w_{\text{stencil}} D(\log_2(N^{1/D}/s_{\text{block}})) \\ &= w_{\text{stencil}} D + w_{\text{stencil}} D r(\log_2(N^{1/D}/s_{\text{block}}) - 1). \end{aligned} \tag{9}$$

We note that none of the blocks overlap with any other block in the physical space as the blocks are the leaves of the tree. 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 additional condition $s_{\text{block}} \geq w_{\text{stencil}} 2^{L_j}$ appears, where $w_{\text{stencil}}$ is the filter size and $L_j$ is the maximum jump in resolution.

*2.2.2. Local compression and refinement of the grid.* The re-adaptation of the grid is achieved by performing elementary operations on the blocks: block splitting to locally refine the grid and blocks collapsing to coarsen the grid. Block splitting and block collapsing are triggered by logic expressions based on the thresholding of detail coefficients residing inside the block. In one dimension, to compute the detail coefficients of the block $i_b$ at level $l_b$, we perform one step of the FWT:

$$d_k^{(l_b-1)} = \sum_m g_{2k-m}^A c_m^{l_b}, \quad k \in \left\{ \frac{i_b \cdot s_{\text{block}}}{2}, \frac{(i_b+1) \cdot s_{\text{block}}}{2} - 1 \right\}. \tag{10}$$

Note that $k$ is inside the block but $m$ can be outside the block (see Figure 3). When we decide to collapse some blocks into one, we just have to replace the data with the scaling coefficient at the coarser level:

$$c_k^{l_b} = \sum_m h_{2k-m}^A c_m^{l_b+1}, \quad k \in \left\{ \frac{i_b}{2} \cdot s_{\text{block}}, \left( \frac{i_b}{2} + 1 \right) \cdot s_{\text{block}} - 1 \right\}. \tag{11}$$

The complexity of a single FWT step, which consists of computing all the details and the scaling coefficients, is approximately $D \cdot (c \cdot s_{\text{block}}/2)^D$ with $c = c(w_{\text{stencil}})$ defined as in Equation (9) and $w_{\text{stencil}}$ is the maximum stencil size of the FWT filters. If we consider the block collapse as the key operation for compression, 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 $(i_b, l_b)$:

$$c_k^{l_b+1} = \sum_m h_{2m-k}^S c_m^{l_b}, \quad k \in \{2i_b \cdot s_{\text{block}}, 2(i_b+1) \cdot s_{\text{block}} - 1\}. \tag{12}$$



DOI: 10.1002/cpe

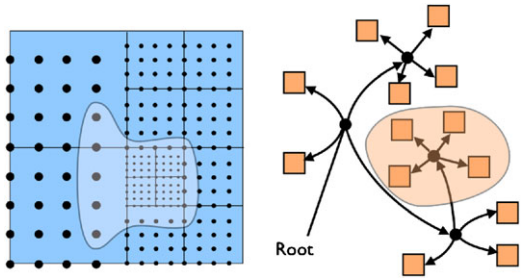



Figure 3. The masked (left) region identifies the source points of the grid used to construct the ghosts, which are used to collapse/split the blocks in the tree (right-masked region).

*2.2.3. Local time-stepping schemes.* Small spatial scales are often associated with small scales in time, especially if the equation under investigation contains nonlinear terms. Our block-adapted grid can exploit local time-stepping (LTS) schemes [16–19]. Depending on the case, this can alone accelerate the simulation by 1 or 2 orders of magnitude. There are, however, two shortcomings in exploiting different time scales. The first drawback comes from the fact that the processing of the grid blocks has to be grouped by their time scales (limiting the degree of parallelism) and no more than one group can be processed in parallel. The second drawback is the increased complexity of reconstructing ghosts, i.e. when dealing with different time scales, ghosts have to be reconstructed also in time.

## 3. WAVELET ALGORITHMS FOR MULTI-CORE COMPUTING

Our first objective in designing the framework is to expose enough load-balanced parallel tasks so that the number of tasks is greater than the number of cores. The second objective is to express nested parallelism inside the parallel tasks, so that the idling cores could help in processing the unfinished tasks. Once these are met, we expect an external library to map our logical tasks into physical threads based on the specific multi-core architecture. Grid adaptation is performed in two steps: before the computing stage we refine the grid to allow for the emergence of new smaller scales [20] and after the computation we apply the compression based on thresholding to retain only the blocks with significant details. It is also desired that most of the execution time is spent in solving the PDE and not in refining/compressing the grid (Figure 4). Figure 5 shows the program flow for the refinement stage. An important aspect of the framework is its ability to retain control over the maximum number of scaling coefficients and therefore, the memory; instead of keeping the threshold fixed, we can also control the number of scaling coefficients with an upper bound by changing the compression threshold.

### *3.1. Producing many parallel tasks*

Owing to the design of the framework, the number of parallel tasks scales with the number of blocks. Parallel tasks can be grouped into *elementary* and *complex* tasks. A task of the *elementary* type operates exclusively on data inside the block, i.e. it is purely local. These tasks are inherently load-balanced, since every task operates on the same number of points. A *complex* task, however,



DOI: 10.1002/cpe

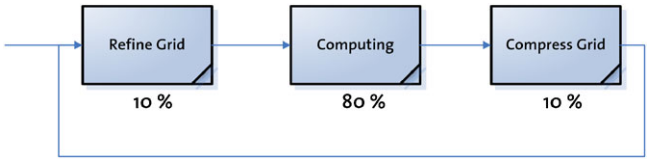



Figure 4. In order to capture the emerging small scales, refinement of the grid is performed before each simulation step. In the computing stage, the solver evaluates the right-hand side of the PDE and evolves the solution updating the grid points. The compression step discards the negligible grid points by looking at the new detail coefficients. It is desired that the computing stages takes most of the execution time (e.g around 80%).

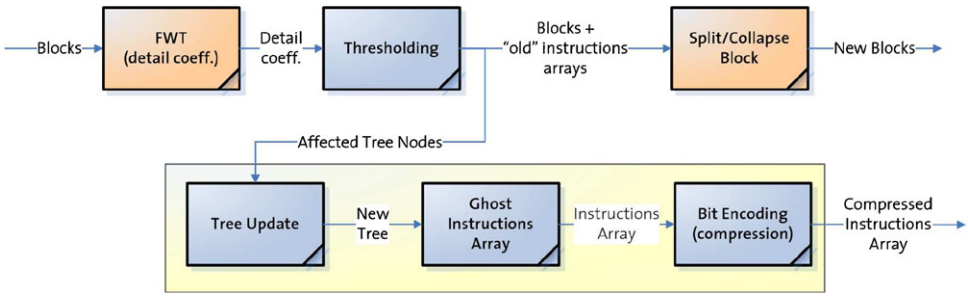



Figure 5. Stages of refinement/compression of the grid. The regions that are surrounded by a box are performed sequentially but executed in parallel for different blocks. The 'FWT' and 'Split/Collapse' stages need to be optimally parallelized as they involve intensive processing of the data. The other stages do not involve compute intensive operations.

involves the reconstruction of ghosts. Block splitting, the FWT, or evaluating finite-difference schemes are complex tasks. The cost of reconstructing a ghost is not constant and can be expensive as it depends on the structure of the tree. The efficiency of the ghost reconstruction is, therefore, paramount for load balance.

### *3.2. Fast ghost reconstruction*

We recall that ghosts are computed as weighted averages of the scaling coefficients in the neighborhood. We call $n_i$ the number of source points/weights that we need to construct the ghost $g_i$ from:

$$g_i = \sum_j w_{ij} p_j, \quad n_i = |\{j \in \mathbb{Z} : w_{ij} \neq 0\}|. \tag{13}$$

We can split the ghost construction into two stages: find the contributing source points/weights and evaluate the weighted average. The first stage will produce an array of weights and indices associated with grid points, which is successively sent as input to the second stage where the array will be used to produce the weighted summation. We call this array the instruction array. We note immediately that the first stage is expensive because it needs recursive data structures. If we assume that we have to construct the same ghost several times, it is meaningful to store the instruction arrays and successively perform only the second stage, which is faster. For the ghosts of



DOI: 10.1002/cpe

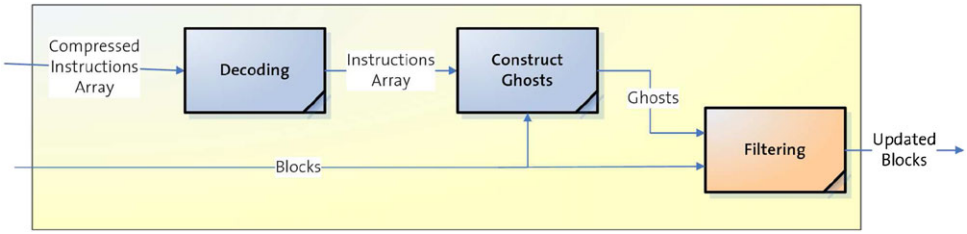



Figure 6. Different stages of a complex parallel task for filtering a block. The Filtering stage needs to be optimally parallelized as it involves intensive processing of the data.

a block, we can reuse the instruction arrays as long as the neighboring blocks remain unchanged. The main shortcoming of this strategy is its memory consumption, as the instruction arrays tend to be large. However, the entries of the instruction arrays of two close ghosts are likely to be very similar. This coherence between instruction arrays allows us to compress them before they are saved. To compress them, we re-organize every instruction array of a block into streams of $w_{ij}$s, $j$s and $n_i$s, and then pass them to an encoder (Figure 5), using either quantization-based encoding or Huffman encoding. We noticed that the overall compression factor of the instruction arrays varies between 1.5 and 5 with a compression factor of 10–20 for the $n_i$. Figure 6 summarizes the use of ghosts in complex parallel tasks for filtering.

## 4. RESULTS

In this section we explain the software and hardware architectures we used to perform our simulations and the performance of our algorithms on a set of benchmark problems. We then present the study on the effect of different block sizes, types of wavelets, and the number of threads.

*Software details.* The framework was written in C++ using generic programming and object-oriented concepts. In our framework, Intel Threading Building Blocks (TBB) [21] library was used to map logical tasks to physical threads. This library completely fulfills the requirements mentioned in Section 3, as it allows for specifying task patterns and enables us to easily express nested parallelism inside tasks. Furthermore TBB provides a very useful set of templates for programming in parallel which are independent of the simulated phenomena. Most of the TBB calls were to `parallel_for`, which exploits recursive task-based parallelism. We employed the `auto_partitioner` for an automatic grain size through the 'reflection on work stealing' [22], to dynamically change the grain size of tasks. In fact, the most efficient grain size varies with respect to the block size and the wavelet order but also with the number of cores.

*Computer architectures.* Simulations were done on the ETH-Zürich central cluster, 'Brutus', a heterogeneous system with a total of 2200 processors in 756 compute nodes. The compute nodes that meet our needs, i.e. multi-threaded nodes are the eight so-called *fat* nodes each with eight dual-core AMD Opteron 8220 CPU's (2.8 GHz) and 64–128 GB of RAM. These nodes are interconnected via a Gigabit Ethernet backbone. The Intel C++ compiler v10.1.015 (64-bit) was used along with the Intel TBB library (64-bit) to build the three different test cases used in this work. For some of



DOI: 10.1002/cpe

the cases, we have also used an Intel Xeon 5150 machine, with two dual core processors with a core speed of 2.66 GHz and 4GB RAM.

### *4.1. Benchmark problems*

*4.1.1. Two-dimensional multiphase compressible flows.* A common benchmark problem for compressible flow solvers consists of the interaction of a shock wave inside air with a cylindrical bubble of helium (see Figure 8, top left). The shock deforms the bubble due to baroclinic vorticity generation and leads to instabilities and emergence of fine structures on the helium/air interface. The governing equations are those of compressible inviscid flows,

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= 0, \\ \frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + p \mathbb{I}) &= \mathbf{0}, \\ \frac{\partial \rho E}{\partial t} + \nabla \cdot ((\rho E + p)\mathbf{u}) &= 0, \end{aligned} \tag{14}$$

with $\rho$, $\mu$, $\mathbf{u}$, $p$, and $E$ being the fluid density, viscosity, velocity, pressure, and total energy per unit mass, respectively. This system of equations is closed with an equation of state of the form $p = p(\rho, E)$. The initial condition is set to that of a Mach 6 shock,

- *State 1* (*post-shock air*): $\gamma = 1.4$, $\rho = 5.268$, $u = 5.752$, $v = 0$, $p = 41.833$
- *State 2* (*pre-shock air*): $\gamma = 1.4$, $\rho = 1$, $u = 0$, $v = 0$, $p = 1$
- *State 3* (*helium bubble*): $\gamma = 1.677$, $\rho = 0.138$, $u = 0$, $v = 0$, $p = 1$

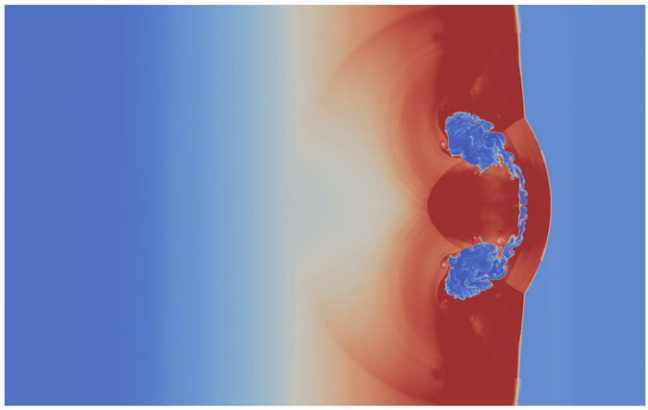
as denoted in Figure 8. A level set description (see Section 4.1.2) is also employed to track the air–helium interface. In this test case, each grid point stored the values $\rho$, $\rho u$, $\rho v$, $\rho E$, and $\phi$. The simulation was performed using a maximum effective resolution of $8192 \times 8192$ grid points with fifth-order average interpolating wavelets, a block size of 32, and a maximum jump in resolution of 2. Derivatives and numerical fluxes were calculated using fifth-order WENO [23] and HLLE [24] schemes, respectively. Time stepping was achieved through an LTS version of second-order TVD Runge–Kutta scheme. The compression/refinement criteria were based on the wavelet coefficients of the interface location and the magnitude of the density (see Figures 7 and 8).

Figure 9 (left) demonstrates the strong speedup achieved for block sizes ranging from 16 to 48 and fifth-order average interpolating wavelets in the shock bubble simulation on 1–16 cores with a fixed $\varepsilon_{\text{compression}}$ and $\varepsilon_{\text{refine}}$. We notice that the optimal block size for this problem is 32 which gives a speedup of 12.2 over 16. In Figure 9 (right), one can notice that a better strong speedup is made possible by using more blocks on higher number of cores, whereas the speedup is higher for less number of blocks while running on less than four cores.

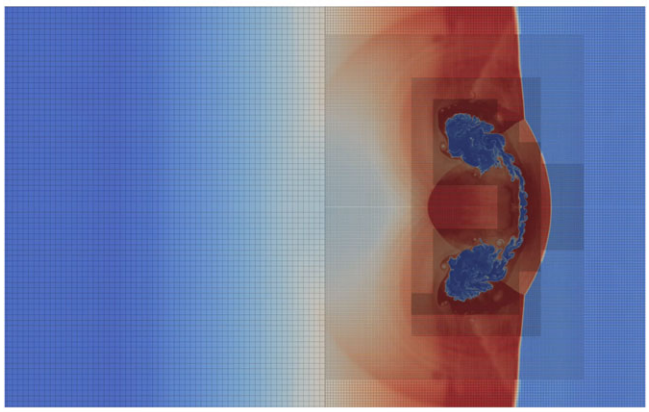
*4.1.2. Three-dimensional advection of level sets.* We simulated the advection of level sets in 3D based on the advection equation $(\partial \phi / \partial t) + \mathbf{u} \cdot \nabla \phi = 0$. Derivatives and numerical fluxes were constructed using a fifth-order WENO scheme [23] and the Godunov flux, respectively. Time integration was performed with an LTS version of the first-order Euler scheme, where given a



DOI: 10.1002/cpe



(a)



(b)

Figure 7. Small scales in the density field (top) of the helium bubble interacting with the Mach 6 shock are captured by the adaptive solver (bottom).

fixed CFL number, each block is updated based on its own time step, i.e. for a block at level $l$ we used a time step proportional to $2^{-l}$. CFL number is defined as:

$$\mathrm{CFL} = \Delta t \frac{|u|_{\max}}{\Delta x_{\mathrm{local}}} \tag{15}$$

with $\Delta x_{\mathrm{local}} = 2^{-l}/\mathrm{blocksize}$. Solving the level set advection equation often requires extra treatment to maintain the regularity of the level set field and we use the re-initialization equation $(\partial\phi/\partial t) + \mathrm{sgn}(\phi_0)(|\nabla\phi| - 1) = 0$, proposed by Sussman *et al.* [25] to this end with $\phi_0$ being the initial condition. For efficient level set compression we employed the detail-clipping trick introduced in [9]. As initial interface we used a dragon and a horse illustrated in Figure 10. Simulations were carried out using four different types of wavelets: third- and fifth-order average interpolating wavelets, second- and fourth-order interpolating wavelets, and for different block sizes ranging from 16 to 32 as well as for a range of cores from 1 to 16, while $\varepsilon_{\mathrm{compression}}$ and $\varepsilon_{\mathrm{refine}}$ were



DOI: 10.1002/cpe

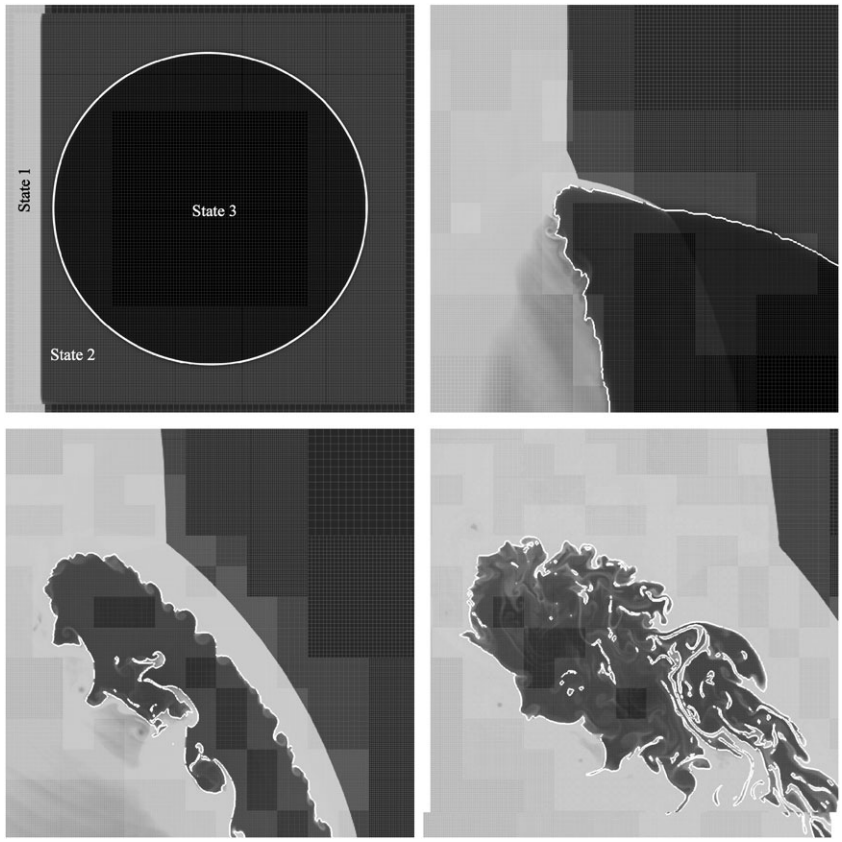



Figure 8. Initial condition (top left) and zooms of the simulation to the high resolution region close to the bubble at different times (chronologically: top right, bottom). White/black depict high/low density, whereas the interface of the bubble is depicted in white.

kept fixed. We measured the time spent in the computing stage and Figure 11 shows the strong efficiency obtained from these measurements. The largest decrease in performance is from 8 to 16 cores. In the case of 16 cores, the best result was achieved using a block size of 16 with the interpolating wavelet of order 4. With this settings we achieved a strong efficiency of 0.8, with a strong speedup of 12.75 with 16 cores. While with the fourth-order interpolating wavelets we achieved a monotonic decreasing efficiency, the efficiency obtained with average interpolating wavelets of order 3 and 5 was better with 8 cores than with 4. We observe that the combination of fourth-order interpolating wavelets with a block size of 16 also achieved the best timings (see Figure 11). Furthermore, we note that the lines do not cross, i.e. the change in the number of cores does not affect the ranking of the computing time for the different wavelets and block size. We also note that the second fastest setting was the third-order average interpolating wavelets with a block size of 16, which did not show a good efficiency (0.72 with 16 cores). Regarding the time spent per block versus the block size, the best timing was achieved by the third-order average interpolating wavelets. Theoretically, the time spent per block should increase roughly by a factor



DOI: 10.1002/cpe

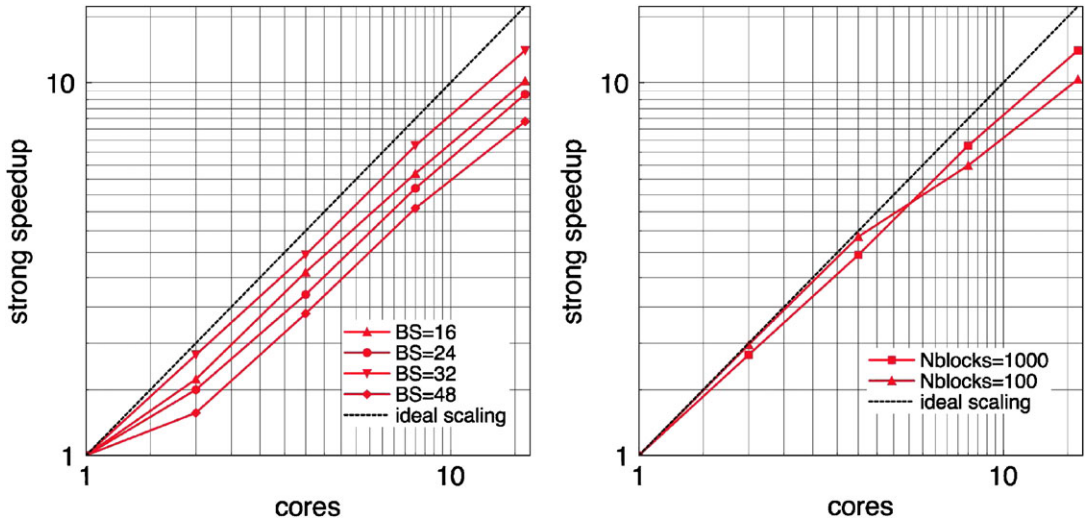



Figure 9. Strong scaling for different block sizes (left) and for different number of blocks in the adaptive grid (right).

of 8 between a block size of 16 and 32. We measure an increase by a factor of about 7 for all three wavelets, meaning that for a block size of 16, the 'pure computing' time per block was about 85% of the total time spent per block. We must, however, note that the time spent per block also includes some synchronization mechanism, which scales with the number of blocks. The latter is high in the case of the third-order average interpolating wavelets with a block size of 32. Figure 12 depicts the time spent versus the block size with one core (left) and 16 cores (middle). We can see that with the third-order wavelet we have a substantial decrease in performance with a block size of 32; this is due to the high number of blocks (Figure 12, right) and the cost associated with the reconstruction of the ghosts at the coarser resolutions.

*4.1.3. Three-dimensional reaction–diffusion problems.* We have also simulated the Gray-Scott [26] model, where the chemical reactions for two diffusing species, $U$ and $V$, are given as:

$$U + 2V \to 3V,$$
$$V \to P.$$

The corresponding PDEs are

$$\frac{\partial u}{\partial t} = d_u \Delta u - uv^2 + F(1-u), \tag{16}$$

$$\frac{\partial v}{\partial t} = d_v \Delta v + uv^2 - (F+\kappa)v, \tag{17}$$



DOI: 10.1002/cpe

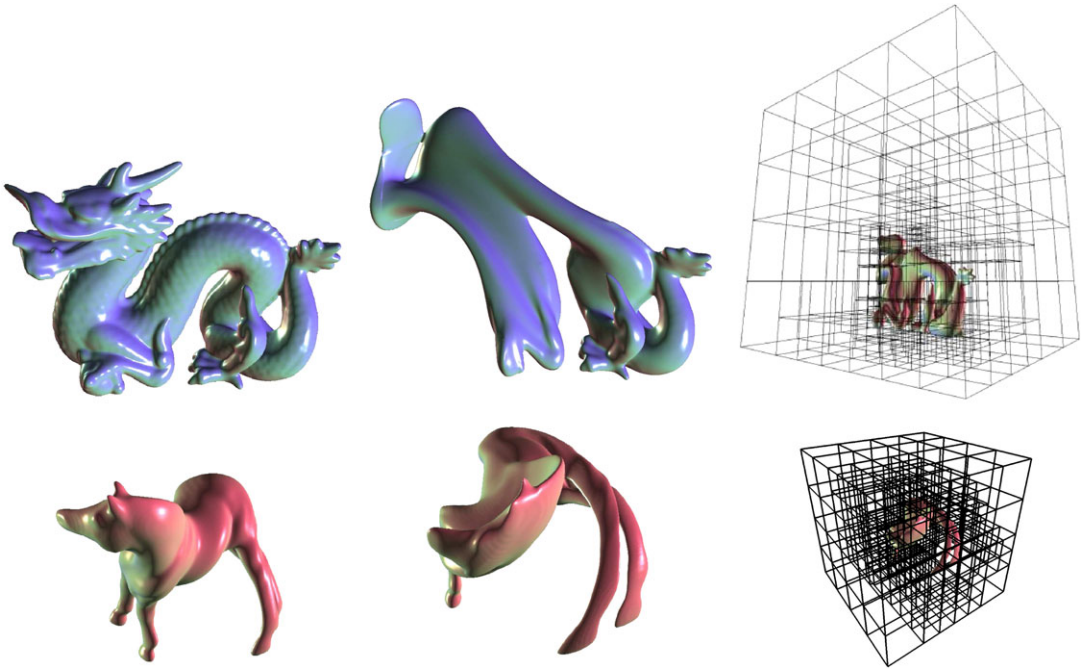


Figure 10. Initial interface (left), advected with a velocity field (middle) of a vortex ring (top) or a sphere transform (bottom), and the distribution of the blocks in the grid (right).

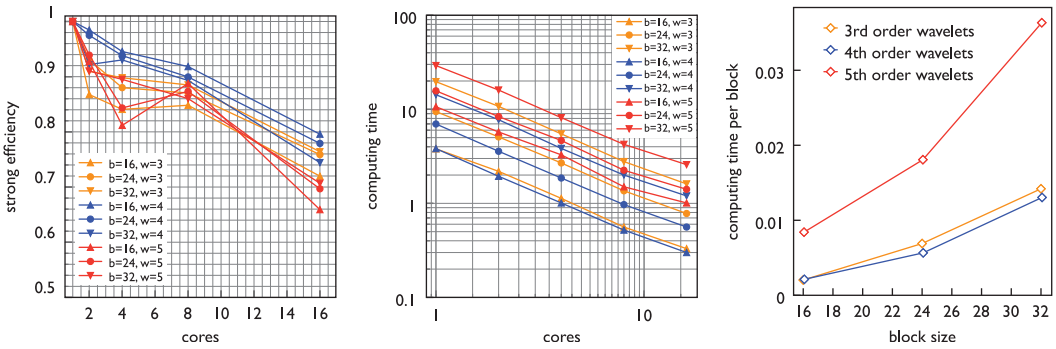



Figure 11. Efficiency (left), computing times (middle) and time spent per block (right) for different interpolating/average interpolating wavelets and block sizes.

where $\kappa$ is a reaction rate for the second chemical reaction, $F$ is a dimensionless feed rate, $d_u$ and $d_v$ diffusion coefficients, $u = u(\mathbf{x}, t)$ represents the concentration of species $U$, and $v = v(\mathbf{x}, t)$ represents the concentration of species $V$. We chose a case with $F = 0.04$ and $\kappa = 0.06$ in a cubic



DOI: 10.1002/cpe

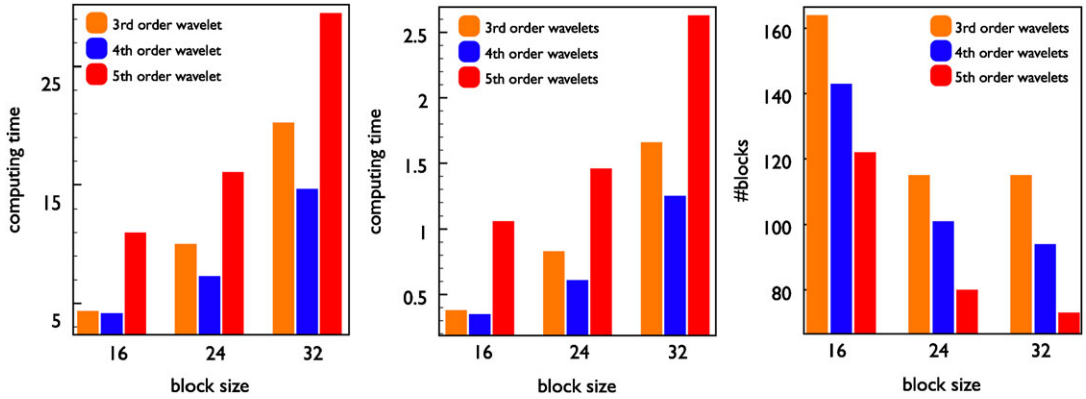



Figure 12. Computing time versus different block sizes by using one core (left) and 16 cores (middle) and the number of blocks versus different block sizes (right).

domain of $[-0.5; 0.5]^3$ and the following initial concentration

$$u_0(x) = \begin{cases} 1 & \text{if } \|\mathbf{x}\|_2 < 0.1, \\ 1/2 & \text{otherwise,} \end{cases} \tag{18}$$

$$v_0(x) = \begin{cases} 1/4 & \text{if } \|\mathbf{x}\|_2 < 0.1, \\ 3/4 & \text{otherwise,} \end{cases} \tag{19}$$

perturbed with white noise of a magnitude of $10^{-2}$. The diffusion rates were set to $d_u = 10^{-6}$ and $d_v = 2 \times 10^{-6}$. The simulation was done from time $t = 0$ to $t = 0.2$ (Figure 13) with second-order finite difference schemes for the diffusion term. For this simulation we used an Euler local time-stepper. Each grid point was a six-component floating point vector to encode $u_i$, $v_i$, $\mathrm{d}u_i/\mathrm{d}t$, $\mathrm{d}v_i/\mathrm{d}t$ and two temporary variables used for the multi-level time integration. We imposed a maximum level of 4 and we tried block sizes up to $32^3$. This means that the simulation could reach a full resolution of $512^3$ scaling coefficients and a maximum memory size of 3 GB during the computation and 1GB as simple field representation. Considering the fact that diffusion effects have a time scale of $\delta t \sim \delta x^2$, for a block at level $l$ we used a time step proportional to:

$$\delta t = \frac{1}{8} d_v \frac{2^{-2l}}{s_{\text{block}}}, \tag{20}$$

where $s_{\text{block}}$ is the block size. With the LTS scheme we achieved an acceleration factor of more than 30. We run the simulation on an Intel Xeon machine and on one fat SMP node of Brutus. On the Intel Xeon machine the total simulation took approximately 100 CPU hours and 350 MB while we used the fourth-order interpolating wavelets, whereas in the case of the third-order average interpolating wavelets, the simulation took roughly 186 CPU hours and 850 MB. This means that the interpolating wavelet of order 4, which is definitely more expensive than the third-order average interpolating wavelet, had a more compact representation of the concentration and thus skipping the processing of many blocks.



DOI: 10.1002/cpe

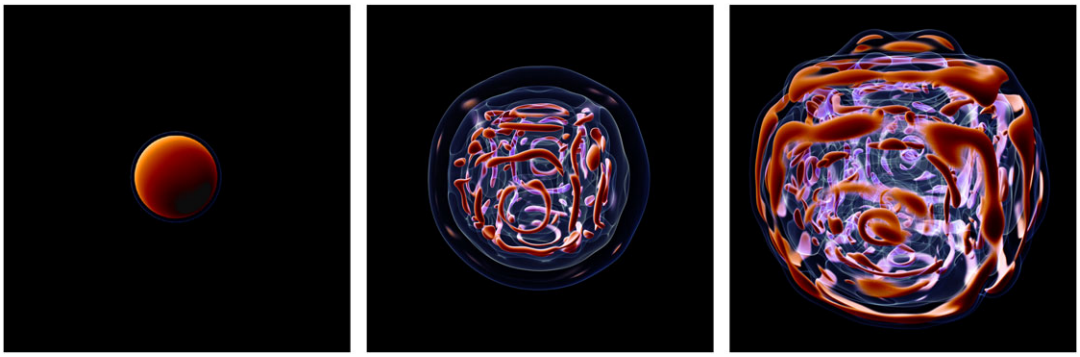


Figure 13. Density field of $v \in [0.3; 0.5]$ for the simulation of the Gray–Scott system at $t = 0$ (left), $t = 0.1$ (middle) and $t = 0.2$ (right).

## 5. CONCLUSIONS

We presented a wavelet-based framework for adaptive multiresolution simulations of time-dependent PDEs. The present framework demonstrates that it is possible to develop space–time adaptive simulation tools that can exploit the new multi-core technologies without degrading the parallel scaling. We have also shown the change in performance and compression rate based on the choice of the wavelets, the size of the blocks, and the number of cores. We achieved the best performances in terms of scaling and timings using a block size of 16; the best wavelet was the fourth-order interpolating one, which gave a strong speedup of 12.75 of 16 cores. We observed that bigger block sizes can be efficiently coupled with high-order wavelets, whereas smaller block sizes are more efficient when combined with low-order wavelets. We have also seen that the performance differences between combinations of block sizes and wavelets are independent of the number of cores, meaning that good efficiency is expected also for more than 16 cores. The current work includes an extension of the framework to support particle-based methods [27] and the incorporation of multiple GPUs as computing accelerators. On the application level we focus on extending the two-dimensional flow simulations to three dimensions and combining them with reaction–diffusion processes and penalization techniques for adaptive simulations of combustion and fluid–structure interaction.

## REFERENCES

1. Ishihara T, Gotoh T, Kaneda Y. Study of high reynolds number isotropic turbulence by direct numerical simulation. *Annual Review of Fluid Mechanics* 2009; **41**(1):165–180.
2. Yokokawa M, Uno A, Ishihara T, Kaneda Y. 16.4 tflops dns of turbulence by fourier spectral method on the earth simulator. *Supercomputing '02*: *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society Press: Los Alamitos, CA, U.S.A., 2002; 1–17.
3. Chatelain P, Curioni A, Bergdorf M, Rossinelli D, Andreoni W, Koumoutsakos P. Billion vortex particle direct numerical simulation of aircraft wakes. *Computer Methods in Applied Mechanics and Engineering* 2008; **197**(13–16):1296–1304.
4. Jiménez J. Computing high-reynolds-number turbulence: Will simulations ever replace experiments? *Journal of Turbulence* 2003; **4**. *Fifth International Symposium on Engineering Turbulence Modelling and Measurements*, Mallorca, Spain, 16–18 September 2002.



DOI: 10.1002/cpe

5. Berger MJ, Oliger J. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics* 1984; **53**(3):484–512.
6. Harten A. Adaptive multiresolution schemes for shock computations. *Journal of Computational Physics* 1994; **115**(2):319–338.
7. Kevlahan NKR, Vasilyev OV. An adaptive wavelet collocation method for fluid–structure interaction at high reynolds numbers. *SIAM Journal on Scientific Computing* 2005; **26**(6):1894–1915.
8. Liu Q, Vasilyev O. A Brinkman penalization method for compressible flows in complex geometries. *Journal of Computational Physics* 2007; **227**(2):946–966.
9. Bergdorf M, Koumoutsakos P. A lagrangian particle-wavelet method. *Multiscale Modeling and Simulation* 2006; **5**(3):980–995.
10. Hopf M, Ertl T. Hardware accelerated wavelet transformations. *Second Joint Eurographics and IEEE TCVG Symposium on Visualization*, Amsterdam, Netherlands, 29–31 May 2000; 93–103.
11. Yang LH, Misra M. Coarse-grained parallel algorithms for multi-dimensional wavelet transforms. *Journal of Supercomputing* 1998; **12**(1–2):99–118.
12. Bader DA, Agarwal V, Kang S. Computing discrete transforms on the cell broadband engine. *Parallel Computing* Mar 2009; **35**(3):119–137.
13. Tenllado C, Setoain J, Prieto M, Pinuel L, Tirado F. Parallel implementation of the 2D discrete wavelet transform on graphics processing units: Filter bank versus lifting. *IEEE Transactions on Parallel and Distributed Systems* 2008; **19**(3):299–310.
14. Cohen A, Daubechies I, Feauveau J. Biorthogonal bases of compactly supported wavelets. *Communication on Pure and Applied Mathematics* 1992; **45**(5):485–560.
15. Donoho D. Interpolating wavelet transforms. *Technical Report*, Preprint Department of Statistics, Stanford University, 1992.
16. Domingues MO, Gomes SM, Roussel O, Schneider K. An adaptive multiresolution scheme with local time stepping for evolutionary pdes. *Journal of Computational Physics* 2008; **227**(8):3758–3780.
17. Domingues MO, Gomes SM, Roussel O, Schneider K. 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, 16–19 January 2007.
18. Alam JM, Kevlahan NKR, Vasilyev OV. Simultaneous space–time adaptive wavelet solution of nonlinear parabolic differential equations. *Journal of Computational Physics* 2006; **214**(2):829–857.
19. Roussel O, Schneider K, Tsigulin A, Bockhorn H. A conservative fully adaptive multiresolution algorithm for parabolic pdes. *Journal of Computational Physics* 2003; **188**(2):493–523.
20. Liandrat J, Tchamitchian P. Resolution of the 1D regularized Burgers equation using a spatial wavelet approximation. *Technical Report 90-83*, NASA Contractor Report 18748880, 1CASE, December 1990.
21. Contreras G, Martonosi M. Characterizing and improving the performance of Intel threading building blocks. *2008 IEEE International Symposium on Workload Characterization* (*IISWC*). IEEE: Piscataway, NJ, U.S.A., 2008; 57–66.
22. Robison A, Voss M, Kukanov A. Optimization via reflection on work stealing in TBB. *2008 IEEE International Symposium on Parallel and Distributed Processing*, vol. 1–8. IEEE: New York, NY, U.S.A., 2008; 598–605.
23. Harten A, Engquist B, Osher S, Chakravarthy S. Uniformly high order accurate essentially non-oscillatory schemes. *Journal of Computational Physics* 1997; **131**(1):3–47. (Reprinted from *Journal of Computational Physics* 1987; **71**:231.)
24. Wendroff B. Approximate riemann solvers, godunov schemes and contact discontinuities. *Godunov Methods*: *Theory and Applications*, Toro EF (ed.). London Math Soc, Kluwer Academic/Plenum Publisher: New York, NY, U.S.A., 2001; 1023–1056.
25. Sussman M, Smereka P, Osher S. A level set approach for computing solutions to incompressible 2-phase flow. *Journal of Computational Physics* 1994; **114**(1):146–159.
26. Reynolds W, PonceDawson S, Pearson J. Self-replicating spots in reaction-diffusion systems. *Physical Review E* 1997; **56**(1, Part A):185–198.
27. Koumoutsakos P. Multiscale flow simulations using particles. *Annual Review of Fluid Mechanics* 2005; **37**: 457–487.



DOI: 10.1002/cpe