2010 12th IEEE International Conference on High Performance Computing and Communications

Hybrid Distributed-/Shared-Memory Parallelization For Re-Initializing Level Set Functions

Oliver Fortmeier and H. Martin Bücker
Institute for Scientific Computing,
Center for Computational Engineering Science (CCES),
RWTH Aachen University,
52056 Aachen, Germany
{*fortmeier,buecker*}*@sc.rwth-aachen.de*

***Abstract*—The ever-increasing power of high-performance computers and advances in numerical techniques make possible the realistic study of two-phase flow problems in three spatial dimensions. Unfortunately, today, there is often still a gap between the design of numerical algorithms and the characteristics of the hardware on which the algorithms are executed. For the solution of a particular subproblem of a two-phase flow problem, we develop a numerical algorithm that aims to match the architecture of a cluster of nodes with multi-core chips. The algorithm is concerned with the re-initialization of level set function used to keep track of the interface between two phases of a fluid. It consists of a hybrid MPI/OpenMP parallelization strategy, using a domain decomposition approach on the outermost level of parallelization. On the inner level, a parallel region handles an individual subdomain. So, a domain decomposition approach based on MPI is combined with an OpenMP approach leading to a hybrid distributed-/shared-memory parallelization. Numerical experiments show that using such a hybrid strategy scales better than a pure MPI parallelization on two different Xeon-based clusters of quad-core processors using up to 1024 cores.**

***Keywords*-hybrid parallelization, two-phase flow problems, unstructured grids, finite elements, computational fluid dynamics, DROPS**

I. Introduction

Three-dimensional flow problems are ubiquitous in scientific computing on high-performance computers. In various flow problems, there are two spatial regions each of which is characterized by a different physical property. Often, these properties are uniformly distributed in space. Examples of such two-phase flow problems include climate systems separating clouds from air, oil slicks in coastal waters, or systems containing gas and liquid. The aim of an interdisciplinary team of researchers from engineering, mathematics and computer science at RWTH Aachen University is to analyze the behavior of multi-phase flow problems using different approaches [1]. In particular, one is interested in a drop levitated in a fluid of different density [2] as well as in a fluid located on walls of a tube that flows as a continuous film downwards driven by gravity [3]. In both applications, different flow phenomena in the vicinity of the interface between the phases are studied.

A mathematical model to describe two-phase flows is based on a so-called level set approach [4]. Here, a scalar function, called level set function, is used to divide the computational domain into two parts, where each part represents one of the two phases. In this approach, it is crucial that the level set function is close to a signed distance function, i.e., the value of the level set function indicates the distance to the interface between the two phases. While evolving in simulation time, this distance property gets lost. To recover the property, two-phase flow solvers typically include a re-initialization algorithm to retrieve the signed distance property.

Various sequential approaches are commonly applied for this task. Three examples includes the fast marching method [5], the fast sweeping method [6], and the solution of the underlying Eikonal equation [7]. In [8], a comparison of these sequential approaches is given. Parallel algorithms to re-initialize level set functions are rarely investigated in the open literature. Often, the parallel algorithms require structured grids as described for the fast sweeping method in [9] and for the fast marching method in [10]. In [11], the level set function is discretized on an adaptive refined Cartesian grid, whereas the flow is discretized on an unstructured grid. In contrast, our approach uses the same unstructured grid to discretize the flow and the level set function.

The recent parallel algorithm [12] for re-initializing level set functions is based on a domain decomposition strategy. This algorithm uses the same unstructured grid for the level set function as well as for the flow solution. In this present note, we extend this domain decomposition parallelism by an additional level of shared-memory parallelism. The main advantage of extending the algorithm is given by the opportunity to use a hybrid distributed-/shared-memory parallelization. This allows to cope with the architecture of most of state-of-the-art high-performance clusters where each compute node consists of a multi-core chip.

The outline of this note is as follows. In Sec. II, we briefly describe the level set approach modeling two-phase flows. The hybrid parallel algorithm for re-initializing level

978-0-7695-4214-0/10 $26.00 © 2010 IEEE
DOI 10.1109/HPCC.2010.64

set functions is presented in Sec. III. We show detailed performance results using up to 1 024 cores in Sec. IV before concluding the note in Sec. V.

II. Level Set Approach for Two-Phase Flow

The level set approach is a technique for the solution of two-phase flow problems thoroughly described in [4], [13]. Let $\Omega \subset \mathbb{R}^3$ denote a computational domain. The level set function is a scalar-valued function $\varphi : \Omega \times [0, \tau_e] \to \mathbb{R}$ used to split the computational domain over time $\tau \in [0, \tau_e]$ into two subdomains $\Omega_1(\tau)$ and $\Omega_2(\tau)$, where $\Omega_1(\tau)$ exemplarily represents an oil phase such as a drop, and $\Omega_2(\tau)$ the second phase, e.g., the surrounding water phase. These subdomains are characterized by

$$\begin{aligned} \Omega_1(\tau) &:= \{\, \vec{x} \in \Omega \mid \varphi(\vec{x}, \tau) < 0 \,\} \text{ and} \\ \Omega_2(\tau) &:= \{\, \vec{x} \in \Omega \mid \varphi(\vec{x}, \tau) > 0 \,\}. \end{aligned}$$

That is, depending on the sign of $\varphi(\vec{x}, \tau)$ the point $\vec{x} \in \Omega$ is located at time τ in $\Omega_1(\tau)$ or $\Omega_2(\tau)$. This representation allows to describe the interface $\Gamma_\varphi(\tau)$ between both phases implicitly by the root of the level set function φ, in formula

$$\Gamma_\varphi(\tau) := \{\, \vec{x} \in \Omega \mid \varphi(\vec{x}, \tau) = 0 \,\}.$$

In Fig. 1, this situation is illustrated for a one-dimensional example. Here, φ decomposes the domain $\Omega \subset \mathbb{R}$ into Ω_1 and Ω_2.

Using the level set function φ, a two-phase flow problem involving the velocity field $\vec{u}(\vec{x}, \tau)$ and the pressure $p(\vec{x}, \tau)$ in a domain Ω can be mathematically modeled by the following set of equations

$$\rho(\varphi)\left(\frac{\partial \vec{u}}{\partial \tau} + (\vec{u} \cdot \nabla)\vec{u}\right) = -\nabla p + \rho(\varphi)\vec{g} + \operatorname{div}(\mu(\varphi)\vec{D}(\vec{u})) + f_{\Gamma_\varphi}, \tag{1}$$

$$\operatorname{div} \vec{u} = 0, \tag{2}$$

$$\frac{\partial}{\partial \tau}\varphi + \vec{u} \cdot \nabla\varphi = 0 \tag{3}$$

with suitable boundary and initial conditions. Here, equations (1)–(2) are the Navier–Stokes equations, whereas (3) describes the evolution of φ in time. In these equations, $\vec{D}$ denotes the viscous stress tensor and $\vec{g}$ the external gravity force. The term f_{Γ_φ} is the so-called "continuum surface force term" [14], [15], describing the surface tension at the interface Γ_φ. To distinguish between the different material properties of the two phases, the density ρ and the viscosity μ depend on the level set function. That is, the material properties at a point $\vec{x} \in \Omega$ and time τ are determined according to the sign of $\varphi(\vec{x}, \tau)$.

For simulating two-phase flow problems, it is crucial that the implicitly given interface Γ_φ can be accurately resolved numerically. Therefore, the level set function should be close to a signed distance function satisfying the distance property

$$\|\nabla\varphi\|_2 = 1, \tag{4}$$

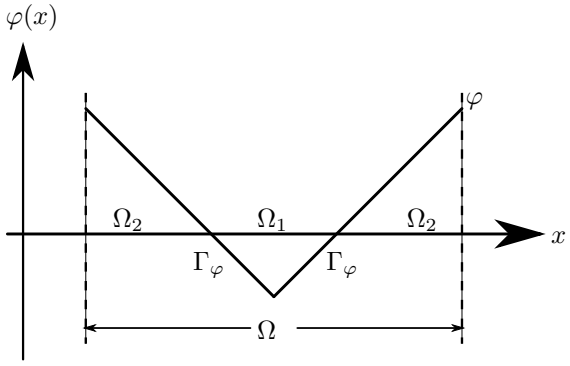


Figure 1. One-dimensional example of a decomposed domain $\Omega \subset \mathbb{R}$ by a level set function φ.

where $\|\cdot\|_2$ denotes the Euclidian norm. That is, the absolute value of $\varphi(\vec{x}, \tau)$ gives the distance of $\vec{x}$ to Γ_φ at time τ, and the sign indicates the phase in which $\vec{x}$ is located. If φ is close to a signed distance function the numerical algorithms can determine the position of Γ_φ accurately. However, while evolving in simulation time the property of the signed distance function may be lost. In the next section, we present a hybrid parallel re-initialization algorithm for reconstructing a signed distance function without moving its root set, i.e. Γ_φ.

III. Hybrid Parallel Algorithm

We introduce some notations and the serial algorithm before describing the hybrid parallel re-initialization algorithm with two levels of parallelism in Sec. III-B. The underlying parallel algorithm with a single level of parallelism obtained from a domain decomposition strategy is presented and analyzed in [12].

A. Re-Initialization Algorithm

Let $\mathcal{T}$ denote a tetrahedral triangulation of the three-dimensional domain Ω and $\mathcal{V}$ the set of vertices in $\mathcal{T}$ where the degrees of freedom to represent φ are located. Given a numerically disturbed level set function $\tilde{\varphi}(\vec{x})$, the objective of the re-initialization algorithm is to determine a new level set function $\varphi(\vec{x})$. The re-initialized level set function φ better recaptures the distance property (4) at each vertex $u \in \mathcal{V}$ without changing the root of $\tilde{\varphi}$. To this end, we distinguish between frontier and off-site vertices. The "frontier vertices," denoted by $\mathcal{F}$, are located on tetrahedra intersected by Γ_φ. The remaining vertices $\mathcal{S} := \mathcal{V} \setminus \mathcal{F}$ are called "off-site vertices." An example triangulation $\mathcal{T}$ with an interface Γ_φ is illustrated in Fig. 2. For the sake of simplicity, the example is given in two dimensions. In this figure, the interface Γ_φ is given by a dashed line, the frontier vertices are marked by $\bullet$, and the off-site vertices are marked by $\circ$.

Computing the unsigned distance $d(\Gamma_\varphi, u)$ between a vertex $u \in \mathcal{V}$ and the interface Γ_φ differs for frontier and

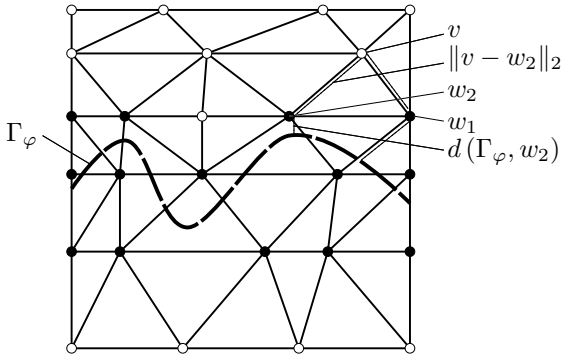


Figure 2. Triangulation $\mathcal{T}$ of a computational domain Ω with a given interface Γ_φ where the frontier vertices $\mathcal{F}$ are marked by $\bullet$ and the off-site vertices $\mathcal{S}$ by $\circ$.

off-site vertices. The algorithm outlined in Alg. 1 consists of the following three steps:

1) Determine all frontier vertices $\mathcal{F}$ and compute the distance $d(\Gamma_\varphi, w)$ of each $w \in \mathcal{F}$.
2) Determine $d(\Gamma_\varphi, v)$ of each $v \in \mathcal{S}$.
3) Determine the value of $\varphi(u)$ for each vertex $u \in \mathcal{V}$, i.e., assigning the signed distance.

Algorithm 1: Algorithm to re-initialize a level set function.

```
1  d(Γ_φ, u) ← ∞ for all u ∈ V
2  foreach t ∈ T do                                   // FRONT
3      if t is intersected by Γ_φ then
4          d(Γ_φ, w) ← min(d(Γ_φ, w), P(t, w)) for each
           corner vertex w ∈ t
5          F ← F ∪ {w} for all corner vertices w ∈ t
6  S ← V \ F
7  KD ← Build k-d tree representing F
8  foreach v ∈ S do                                   // OFFSITE
9      Compute N(m, v) with KD
10     minDist ← ∞
11     foreach y ∈ N(m, v) do                         // MIN
12         minDist ← min{minDist, d(Γ_φ, v, y)}
13     d(Γ_φ, v) ← minDist
14 foreach u ∈ V do                                   // SIGN
15     φ(u) ← d(Γ_φ, u) · sign(φ̃(u))
```

The first step (lines 2–6 of Alg. 1) is accomplished by a loop over all tetrahedra. If a tetrahedron $t \in \mathcal{T}$ is intersected by the root of the level set function, all corner vertices w of t are characterized as frontier vertices. The distance $d(\Gamma_\varphi, w)$ for $w \in \mathcal{F}$ is performed by a local projection $P(t, w)$ on an intersected tetrahedron t involving basic geometrical calculations. This projection is described in [13]. All remaining vertices are marked as off-site vertices.

Afterwards, in the second step (lines 7–13), the distance between each off-site vertex $v \in \mathcal{S}$ and Γ_φ is determined. Therefore, we first define the distance $d(\Gamma_\varphi, v, w)$ of an off-site vertex v via a frontier vertex w by

$$d(\Gamma_\varphi, v, w) := \|v - w\|_2 + d(\Gamma_\varphi, w). \tag{5}$$

Using this definition, we determine the distance $d(\Gamma_\varphi, v)$ between $v \in \mathcal{S}$ and Γ_φ by

$$d(\Gamma_\varphi, v) := \min\{\, d(\Gamma_\varphi, v, w) \mid w \in \mathcal{F} \,\}. \tag{6}$$

Here, the distance of an off-site vertex is given by the shortest distance via all frontier vertices. To determine the minimum in (6) efficiently, we do not search for the minimum over all frontier vertices but only over the m nearest neighbors of v in the set $\mathcal{F}$. The set of m nearest neighbors of v is denoted by $\mathcal{N}(m, v) \subset \mathcal{F}$ where m is a user-given parameter. Note that determining (6) by using only the nearest neighbor does not give accurate results. For instance, in Fig. 2, the nearest neighbor of v in $\mathcal{F}$ is the vertex w_1. However, with respect to the distance defined in (5), the distance of v to the interface is shorter via w_2 than via w_1. Bentley [16] developed k-d trees (k-dimensional trees), which allow to perform a nearest neighbor search of a given point v in a set of points $\mathcal{F}$ efficiently. To this end, the set of frontier vertices $\mathcal{F}$ is represented by a tree data structure. Overall, the second step of the re-initialization algorithm consists of generating the set of m nearest neighbors $\mathcal{N}(m, v)$ for each vertex $v \in \mathcal{S}$ followed by determining

$$d(\Gamma_\varphi, v) := \min\{\, d(\Gamma_\varphi, v, y) \mid y \in \mathcal{N}(m, v) \,\}.$$

The final and third step (lines 14–15) consists of determining the value of the re-initialized level set function $\varphi(u)$ for each vertex $u \in \mathcal{V}$. The value of $\varphi(u)$ is determined by the product of the distance $d(\Gamma_\varphi, u)$ and the sign of $\tilde{\varphi}(u)$, in formula

$$\varphi(u) := d(\Gamma_\varphi, u) \cdot \operatorname{sign}(\tilde{\varphi}(u)).$$

B. Hybrid Parallel Re-Initialization Algorithm

In this section, we combine a distributed-memory parallelization with a shared-memory parallelization leading to a hybrid parallel algorithm for re-initializing level set functions. A domain decomposition strategy, i.e., distributing the tetrahedra of $\mathcal{T}$ among processes, enables the distributed-memory parallelization. The shared-memory parallelization additionally distributes the computational work of the loops beginning in lines 2, 8, and 14 of Alg. 1. In the remainder, we use the labels `FRONT`, `OFFSITE`, and `SIGN` for these loops as given in Alg. 1.

The distributed-memory parallelization of the numerical solution of (1)–(3) is presented in detail in [12], [17], [18] and is implemented using the message passing interface (MPI). Decomposing the tetrahedra of $\mathcal{T}$ among processes

leads to a decomposition of the vertices $\mathcal{V}$ and, in particular, of the off-site vertices $\mathcal{S}$ and frontier vertices $\mathcal{F}$. Let $\mathcal{T}^p$, $\mathcal{V}^p$, $\mathcal{S}^p$ and $\mathcal{F}^p$ denote the restriction of these sets to a process p, respectively. Then, each process has to determine the value of $\varphi(u)$ for each vertex $u \in \mathcal{V}^p$. However, for computing the nearest neighbor set $\mathcal{N}(m, v)$ in line 9 of Alg. 1, each process p needs to access the "global" set $\mathcal{F}$ to build a k-d tree representing $\mathcal{F}$. Note that using on each process only a k-d tree representing the "local" set $\mathcal{F}^p$ does not yield accurate results because the nearest neighbor w of a vertex $v \in \mathcal{S}^p$ must not be located on the same process p. Creating the k-d tree in parallel and performing a parallel nearest neighbor search is beyond the scope of this paper and is currently investigated in an ongoing work. For creating the "global" k-d tree, an additional communication step is necessary to gather the set $\mathcal{F}$ from the "local" sets $\mathcal{F}^p$. After this gathering step, each process stores the global set $\mathcal{F}$ and is capable of building the k-d tree. Note that this gathering also takes special care of frontier vertices located at process boundaries.

The shared-memory parallelization distributes the computational work arising in the subdomains. To this end, the loops `FRONT`, `OFFSITE`, and `SIGN` are parallelized by using the OpenMP standard.

- **Parallelization of loop FRONT.** Most of the computational work within step one, i.e., initializing the frontier vertex set and the distance of its vertices, can be performed in parallel. Hence, the loop over all tetrahedra can be distributed among threads. However, in general, a frontier vertex $w \in \mathcal{F}$ is located at several tetrahedra. If two threads handle two different tetrahedra but assign $d(\Gamma_\varphi, w)$ for the same vertex w, then a data race occurs. Hence, assigning $d(\Gamma_\varphi, w)$ to a frontier vertex has to be executed by only one thread at any time.
- **Parallelization of loop OFFSITE.** Computing the distance $d(\Gamma_\varphi, v)$ for off-site vertices in lines 9–13 of Alg. 1 does not depend on computing any other distance $d(\Gamma_\varphi, v')$ for all $v' \in \mathcal{V} \setminus \{v\}$. Therefore, no data dependencies exist for different loop indices and this loop can be parallelized.
- **Parallelization of loop SIGN.** Determining the value of $\varphi(u)$ as the signed distance $\pm d(\Gamma_\varphi, u)$ does not depend on any other assignment in that loop. Hence, the computational work of this loop body is straightforward distributed among threads.

Note that searching the minimum in line 11 of Alg. 1 labeled by `MIN` can be also parallelized by OpenMP leading to an additional level of shared-memory parallelism. However, numerical experiments not reported here indicate that such an approach involving nested shared-memory parallelization is not competetive in terms of execution time. Indeed, the smallest execution times are gained if all available threads

Algorithm 2: Hybrid parallel algorithm to re-initialize a level set function.

```
1  d(Γ_φ, u) ← ∞ for all u ∈ V
2  foreach {T^1, ..., T^p} do in parallel          // MPI
3      foreach t ∈ T^p do in parallel              // OpenMP
4          if t is intersected by Γ_φ then
5              d(Γ_φ, w) ← min(d(Γ_φ, w), P(t, w)) for
               each corner vertex w ∈ t
6              F^p ← F^p ∪ {w} for all corner vertices
               w ∈ t
7  F ← Gather F^p
8  S^p ← V^p \ F
9  KD ← Build k-d tree representing F
10 foreach {S^1, ..., S^P} do in parallel          // MPI
11     foreach v ∈ S^p do in parallel              // OpenMP
12         Compute N(m, v) with KD
13         minDist ← ∞
14         foreach y ∈ N(m, v) do
15             minDist ← min{minDist, d(Γ_φ, v, y)}
16         d(Γ_φ, v) ← minDist
17 foreach {V^1, ..., V^P} do in parallel          // MPI
18     foreach u ∈ V^p do in parallel              // OpenMP
19         φ(u) ← d(Γ_φ, u) · sign(φ̃(u))
```

are used to execute the loop `OFFSITE` in a single level of shared-memory parallelism rather than two levels of shared-memory parallelism. The high-level representation depicted in Alg. 2 summarizes the hybrid MPI/OpenMP parallel algorithm. Here, the labels `MPI` and `OpenMP` indicate the distributed- and shared-memory parallelization, respectively.

IV. Numerical Results

All experiments are performed at RWTH Aachen University on two clusters consisting of Xeon-based quad-core processors. One is based on "Harpertown" and one on "Nehalem" processors. Each node of the cluster has two sockets, where on each socket a quad-core processor is placed. The nodes are connected via an InfiniBand network, the executable is built by the Intel compiler (version 11.1), and SUN's implementation of MPI (version 8.2) is used to implement the communication among processes.

The re-initialization algorithm given in Alg. 2 is implemented within the parallel C++ two-phase flow solver DROPS [13], [17], [19], [20], [18]. To demonstrate performance results of the parallel re-initialization algorithm, our test scenario is a conical measurement cell taken from [21]. In Fig. 3, a vertical slice through the rotationally symmetric cell with a levitated droplet is depicted. This cell is used to study the behavior of levitated oil drops [2] in water. The cell is discretized by a tetrahedral grid which is locally

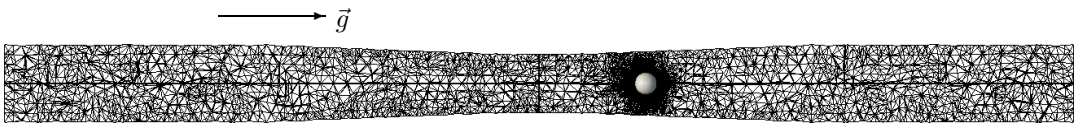
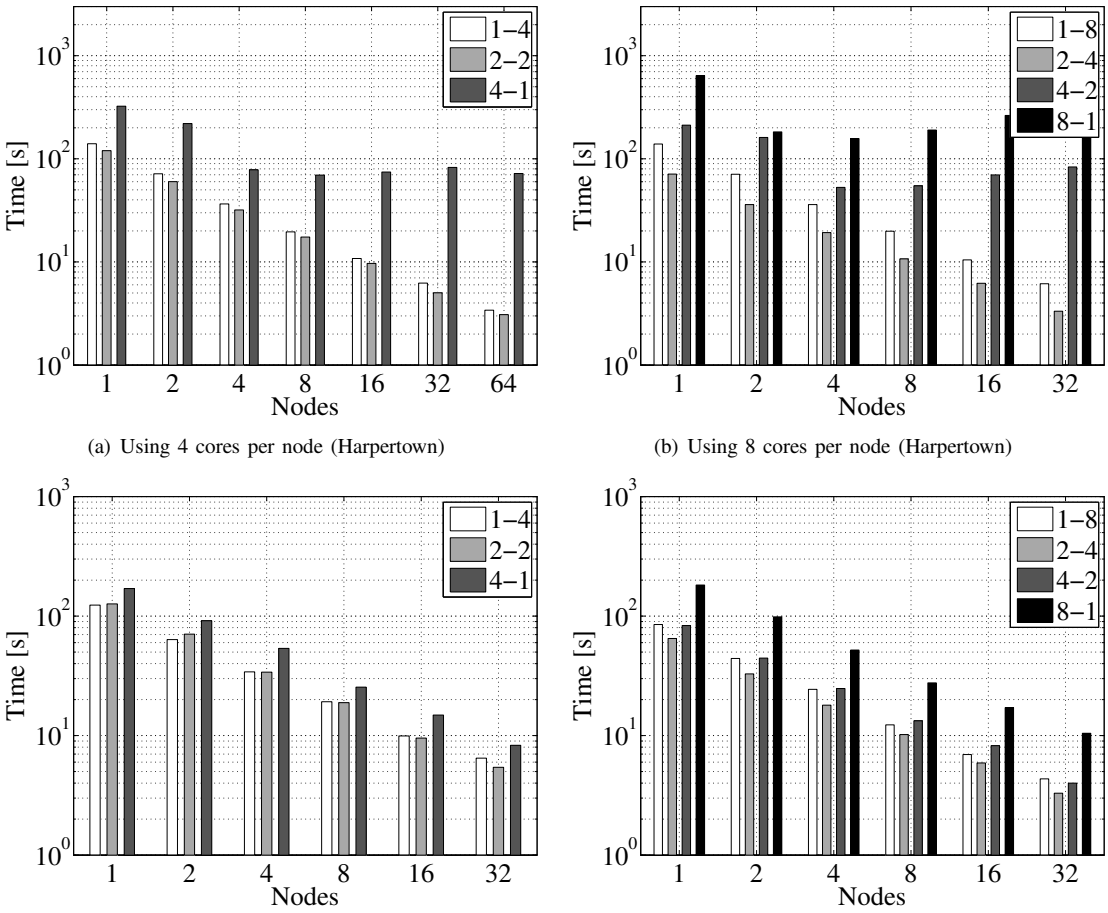


Figure 3. Vertical slice through the computational domain Ω and a drop described by φ.



(a) Using 4 cores per node (Harpertown)

(b) Using 8 cores per node (Harpertown)

(c) Using 4 cores per node (Nehalem)

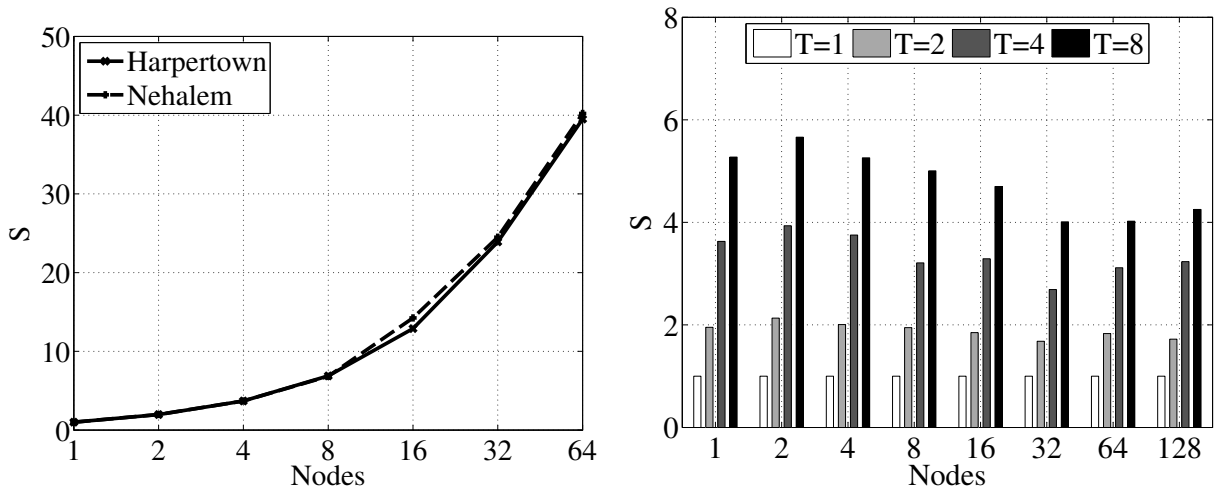
(d) Using 8 cores per node (Nehalem)

Figure 4. Execution time of carrying out the loop `OFFSITE` (lines 10–16 of Alg. 2) on different numbers of nodes.

refined in the vicinity of the surface of the drop. The 2 733 447 tetrahedra lead to 3 152 034 degrees of freedom to represent φ. Here, $|\mathcal{F}| = 155\,962$ vertices are located at intersected tetrahedra. In all experiments, the number of nearest neighbors is set to $m = 1\,000$. The implementation of the k-d trees is based on Kennel's library [22]. To determine a suitable decomposition of the tetrahedra, the graph partitioning library ParMetis [23] is used. In this scenario, over 99 % of the sequential execution time for re-initializing level set functions is performed in the body of the loop `OFFSITE`, corresponding to lines 10–16 of Alg. 2.

Recall that the algorithm includes two levels of parallelism: the distributed-memory parallelization (lines 2, 10, and 17) and the shared-memory parallel regions (lines 3, 11, and 18). We use different schemes to place MPI processes and OpenMP threads on the eight cores per node. We denote the schemes by P-T. Here, P denotes the number of MPI processes placed on each node. For each such process, T denotes the number of threads used to execute the parallel regions of the shared-memory parallelization. Thus, if n nodes are used, the total number of cores executing the algorithm is given by $n \cdot P \cdot T$. For instance, if using the 2-4 scheme, two processes are placed on each node and each process spawns four threads leading to eight threads per node. The total number of MPI processes p is given by $p = P \cdot n$. The execution time of the algorithm carried out on n nodes using the P-T scheme is denoted by $t^{P\text{-}T}(n)$.

First, we investigate which P-T scheme exploits the hardware best. In Fig. 4, we present the execution time on various number of compute nodes to determine $d\,(\Gamma_\varphi, v)$ for



(a) $S^{2\text{-}1}_{\text{MPI}}$ using the scheme 2-1 on different numbers of nodes

(b) $S^{1\text{-}T}_{\text{OpenMP}}$ for different number of Nehalem nodes

Figure 5. Speedup of the hybrid parallel re-initialization algorithm (Alg. 2).

all $v \in \mathcal{S}$, i.e., performing lines 7–16 of Alg. 2. In Fig. 4(a) and (c), only each second core of a node is used whereas, in Fig. 4(b) and (d), all cores of a node are involved in the computation. In Fig. 4(a) and (b), the execution times on a cluster consisting of Harpertown processors is presented whereas, in Fig. 4(c) and (d), a cluster of Nehalem processors is used. Although the clock cycle of both processors is approximately equal to 3 Ghz, the execution time on the Harpertown cluster is larger because the connection between the memory and the Nehalem's cores is better than the connection to the Harpertown's cores. If considering the pure MPI parallelization, i.e., the scheme 4-1 in both left figures and 8-1 in both right figures, the algorithm scales well on up to 4 nodes of the Harpertown cluster and up to 32 nodes of the Nehalem cluster. An explanation is given by the InfiniBand network connecting the nodes. The Nehalem cluster uses a newer generation of an InfiniBand network than the Harpertown cluster. This newer network is faster and, in contrast to the Harpertown cluster, the network interface cards can serve the eight MPI processes on a node better. All figures demonstrate that exclusively using an MPI parallelization yields the largest execution times on a fixed number of nodes because the shared memory can not be exploited. For instance, in Fig. 4(d), switching from a hybrid MPI/OpenMP parallelization to a pure MPI parallelization increases the runtime on four nodes from $t^{1\text{-}8}(4) = 24.8\,\text{s}$ to $t^{8\text{-}1}(4) = 52.1\,\text{s}$. In particular, on four nodes, the serial runtime is reduced by a factor of 18.4 by using the hybrid parallel approach whereas the corresponding factor is 8.6 using a pure MPI approach. In general, for both types of processors, the smallest execution time is obtained if placing one process and four threads on each socket, i.e., using strategy 2-4. The Nehalem processors provides simultaneous multithreading (SMT) which allows to place up to 16 threads on one node. However, our experiments show that enabling this technology increases the runtime on one node by a factor 1.06 compared to only using eight threads.

Finally, we investigate the speedup of Alg. 2. Concerning the MPI parallelization we define the speedup $S^{P\text{-}T}_{\text{MPI}}(n)$ using n nodes and the scheme P-T by

$$S^{P\text{-}T}_{\text{MPI}}(n) := \frac{t^{P\text{-}T}(1)}{t^{P\text{-}T}(n)}.$$

For a fixed number of nodes n, the speedup with respect to OpenMP using T threads is defined by

$$S^{P\text{-}T}_{\text{OpenMP}}(n) := \frac{t^{P\text{-}1}(n)}{t^{P\text{-}T}(n)}.$$

In Fig. 5, the speedup of the re-initializing algorithm is depicted. The speedup $S^{2\text{-}1}_{\text{MPI}}$ is illustrated in Fig. 5(a) for a varying number of nodes. This figure demonstrates that the parallel algorithm scales well on both clusters using a pure MPI parallelization with two processes per node. That is, on both clusters, the execution time of two MPI processes on one node is reduced by a factor of about 40 if using 64 nodes. In Fig. 5(b), the speedup $S^{1\text{-}T}_{\text{OpenMP}}$ with respect to a varying number of threads is shown for different numbers of Nehalem nodes. This figure illustrates that the shared-memory parallelization is capable of decreasing the runtime by a factor larger than 4 for all nodes using $T = 8$ cores.

V. Conclusions

Today's high-performance computing platforms are increasingly built by connecting nodes consisting of multi-core chips. Most likely, the total number of threads summed over all nodes that is available on future platforms will easily exceed the number of independent tasks that follow from

using a single level of parallelism. A hierarchy of multiple levels of parallelism seems a promising option to cope with this large number of threads. This has an immediate implication to the design of algorithms which will have to offer these multiple levels of parallelism. We propose a novel algorithm for the re-initialization of level set functions on an unstructured grid that consists of two levels of parallelism. We investigate the performance of that algorithm by combining a domain decomposition approach based on MPI with an additional level of parallelism obtained from an OpenMP approach. Numerical experiments are carried out on two clusters whose nodes consist of Xeon-based quad-core processors of type Harpertown or Nehalem. These results indicate that it is currently difficult to determine the "best" technique from the rich set of parallelization strategies that results from mapping the hierarchy of parallel tasks to cores in a different way. However, it is evident from our performance results that, for the particular re-initialization algorithm, a hybrid MPI/OpenMP approach is superior to a parallelization technique purely based on MPI.

ACKNOWLEDGMENT

The development of the parallel finite element solver DROPS was supported by the "Deutsche Forschungsgemeinschaft" (DFG) in the SFB 540 "Model-based Experimental Analysis of Kinetic Phenomena in Fluid Multi-phase Reactive Systems," and is being developed in a collaboration with the chair for numerical mathematics (LNM) at RWTH Aachen University. We would like to thank Alin Bastea and the HPC group of the Center for Computing and Communication for helping us to gather and analyze all performance data.

REFERENCES

[1] W. Marquardt, "Model-based experimental analysis of kinetic phenomena in multi-phase reactive systems," *Trans. Inst. Chem. Eng.*, vol. 83, no. A6, pp. 561–573, 2005.

[2] E. Gross-Hardt, A. Amar, S. Stapf, A. Pfennig, and B. Blümich, "Flow dynamics inside a single levitated droplet," *Ind. & Eng. Chem. Res.*, vol. 1, pp. 416–423, 2006.

[3] S. Groß, M. Soemers, A. Mhamdi, F. Al-Sibai, A. Reusken, W. Marquardt, and U. Renz, "Identification of boundary heat fluxes in a falling film experiment using high resolution temperature measurements," *Int. J. Heat Mass Tran.*, vol. 48, pp. 5549–5562, 2005.

[4] M. Sussman, P. Smereka, and S. Osher, "A level set approach for computing solutions to incompressible two-phase flow," *J. Comput. Phys.*, vol. 114, no. 1, pp. 146–159, 1994.

[5] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," in *Proc. Natl. Acad. Sci. of the USA*, vol. 93, no. 4, 1996, pp. 1591–1595.

[6] H.-K. Zhao, "A fast sweeping method for Eikonal equations," *Math. Comput.*, vol. 74, no. 250, pp. 603–627, 2004.

[7] S. Osher and J. A. Sethian, "Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulations," *J. Comput. Phys.*, vol. 79, no. 1, pp. 12–49, 1988.

[8] S.-R. Hysing and S. Turek, "The Eikonal equation: Numerical efficiency vs. algorithmic complexity on quadrilateral grids," in *17th Conference on Scientific Computing (Algoritmy 2005), Vysoké Tatry, Podbanské, Slovakia*, A. Handlovicova, Z. Kriva, K. Mikula, and D. Sevcovic, Eds., 2005, pp. 22–31.

[9] H. Zhao, "Parallel implementations of the fast sweeping method," *J. Comp. Math.*, vol. 25, no. 4, pp. 421–429, 2007.

[10] M. C. Tugurlan, "Fast marching methods—parallel implementation and analysis," Ph.D. dissertation, Department of Mathematics, Louisiana State University, USA, 2008.

[11] M. Herrmann, "A parallel Eulerian interface tracking/Lagrangian point particle multi-scale coupling procedure," *J. Comput. Phys.*, vol. 229, no. 3, pp. 745–759, 2010.

[12] O. Fortmeier and H. M. Bücker, "Parallel re-initialization of level set functions on distributed unstructured tetrahedral grids," RWTH Aachen University, Aachen, Preprint of the Institute for Scientific Computing RWTH–CS–SC–10–03, 2010, submitted for publication.

[13] S. Groß, V. Reichelt, and A. Reusken, "A finite element based level set method for two-phase incompressible flows," *Comput. Vis. Sci.*, vol. 9, no. 4, pp. 239–257, 2006.

[14] J. U. Brackbill, D. B. Kothe, and C. Zemach, "A continuum method for modeling surface tension," *J. Comput. Phys.*, vol. 100, no. 2, pp. 335–354, June 1992.

[15] S. Groß and A. Reusken, "Finite element discretization error analysis of a surface tension force in two-phase incompressible flows," *SIAM J. Numer. Anal.*, vol. 45, no. 4, pp. 1679–1700, 2007.

[16] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[17] S. Groß and A. Reusken, "Parallel multilevel tetrahedral grid refinement," *SIAM J. Sci. Comput.*, vol. 26, no. 4, pp. 1261–1288, 2005.

[18] O. Fortmeier, T. Henrich, and H. M. Bücker, "Modeling data distribution for two-phase flow problems by weighted graphs," in *23rd Workshop on Parallel Sytems and Algorithms, Hannover, Germany, Februar 12, 2010*, M. Beigl and F. J. Cazorla-Almeida, Eds. VDE, 2010, pp. 31–38.

[19] O. Fortmeier and H. M. Bücker, "A parallel strategy for a level set simulation of droplets moving in a liquid medium," in *9th International Meeting on High Performance Computing for Computational Science (VECPAR'10), Berkeley, CA, USA, June 23–25, 2010*, 2010, accepted for publication.

[20] C. Terboven, A. Spiegel, D. an Mey, S. Groß, and V. Reichelt, "Experiences with the OpenMP parallelization of DROPS, a Navier-Stokes solver written in C++," in *OpenMP Shared Memory Parallel Programming, Proceedings of the International Workshops IWOMP 2005 and IWOMP 2006, Eugene, OR, USA, June 1–4, 2005, and Reims, France, June 12–15, 2006*, ser. Lecture Notes in Computer Science, M. S. Mueller, B. M. Chapman, B. R. de Supinski, A. D. Malony, and M. Voss, Eds., vol. 4315. Berlin: Springer, 2008, pp. 95–106.

[21] E. Gross-Hardt, E. Slusanschi, H. M. Bücker, A. Pfennig, and C. H. Bischof, "Practical shape optimization of a levitation device for single droplets," *Optimization and Engineering*, vol. 9, no. 2, pp. 179–199, 2008.

[22] M. B. Kennel, "KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space," 2004, http://arxiv.org/abs/physics/0408067v2.

[23] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *J. Parallel Distrib. Comput.*, vol. 48, no. 1, pp. 71–95, 1998.