even though the use of adaptive grids can dramatically reduce the computational cost, performing high-resolution three dimensional calculations of complex interfacial problems, e.g. crystal growth in binary alloys [42], could be prohibitively expensive or even impossible on a serial machine. In this paper we extend the level-set technology on Quad-/Octree by proposing parallel algorithms for distributed memory machines using a domain decomposition technique.

One of the main challenges in parallelizing algorithms on adaptive grids is handling the grid itself. One option is to replicate the entire grid on each process and use off-the-shelf graph partitioners, e.g. ParMetis [25] or Zoltan [9], for load balancing and domain decomposition. For instance, this was the approach originally taken by the `deal.II` library [7]. This approach, however, is only scalable to a few hundred processes at best and is limited by the size of the grid itself that can fit in memory. Moreover, the use of a general-purpose graph partitioner adds extra overhead that can limit the overall scalability even further. Interestingly, tree-based grids have nice spatial ordering that naturally leads to the concept of space-filling curves (SFCs) that can be efficiently exploited for parallel load balancing [5, 14].

The idea of using SFCs for parallel partitioning of Quad-/Octrees is not new in itself and has been used by many researchers. For instance, `Dendro` [45] uses a Morton curve (also known as Z-curve) for traversing the leaves of an Octree for indexing and load balancing and has been scaled up to 62,000 cores [12]. Dendro [35] is another example of an Octree code in which similar ideas are used for parallel partitioning and development of a parallel geometric multigrid that has been scaled up to about 32,000 cores [36]. More recently, authors in [13] extended these ideas to a collection, or a "forest", of Octrees that are connected through a common, potentially unstructured, hexahedral grid. This forest is then partitioned in parallel using a global Morton curve. The implementation of these algorithms, which were shown to scale to more than 200,000 cores, is publicly available through a simple API provided by the `p4est` library [1]. In fact the algorithms presented in this paper are implemented on top of the `p4est` library and we do not discuss any algorithm that is already covered in [13]. We also make use of the popular PETSc [6] library for linear algebra and its parallel primitives, such as parallel ghosted vector and scatter/gather operations, which simplifies the implementation and delivers good performance.

Parallel level-set algorithms can be categorized into two groups: parallel advection algorithms and parallel reinitialization algorithms. Eulerian advection schemes can easily be parallelized but unfortunately are limited by the CFL condition, which could be very restrictive for adaptive grids. Semi-Lagrangian methods combine the unconditional stability of Lagrangian methods and the ease of use of Eulerian grids and have been successfully used for advecting the level-set function on tree-based grids [26, 27, 28]. However, parallelizing the semi-Lagrangian algorithm in a domain decomposition context is not an easy task. The reason for this is twofold. First, depending on the CFL number, the departure points may end up outside the ghost region and in remote processes that are potentially far away. This requires a very dynamic and nonuniform communication pattern which is complicated to implement. For an adaptive grid, the situation is even more complicated due to the asymmetric nature of the communication (c.f. section 3). Second, load balancing could be an issue for large CFL numbers and nonuniform velocity fields, due to clustering of departure points, which can thus considerably restrict the scalability of the algorithm. Both of these problems, of course, could be avoided by choosing CFL $\leq 1$ but that would defeat the purpose of using the semi-Lagrangian algorithm in the first place.

Nonetheless, several parallel semi-Lagrangian algorithms have been proposed. A simple domain decomposition technique was used in [43] where the width of the ghost layer is fixed based on the maximum CFL number to ensure that all the departure points are covered by the ghost layer. Good scalings were reported for small CFL numbers (CFL $\leq 2$). However, for large CFL numbers, this leads to a large volume of communication that can limit the scalability. In [17] the authors propose a more sophisticated domain decomposition approach which uses a "dynamic ghost layer". Here the width of the ghost layer is dynamically determined at runtime based on information from previous time steps. Unfortunately, this approach also suffers from excessive communication overhead at large numbers of processes. More recently, the authors in [48] used a domain decomposition strategy on a cubed sphere but with a single layer of ghost nodes. Interpolation on remote processes is then handled by sending query points to the corresponding process and asking for the interpolated result. This approach seems to provide good scalability for transporting a single tracer up to about 1000 cores for CFL $\sim 10$. At higher CFL numbers, the method begins to loose scalability due to an increase in communication volume. Finally, note that although we are mainly interested in parallel semi-Lagrangian methods, one could resort to finite difference or finite element discretization