Parallel Velocity Extension for Level-Set-Based Material Flow on Hierarchical Meshes in Process TCAD

Michael Quell, Vasily Suvorov, Andreas Hössinger, and Josef Weinbub, *Senior Member, IEEE*

***Abstract*—The level-set method is widely used for high-accuracy 3-D topography simulations in process technology computer-aided design (TCAD) because of its robustness to topological changes introduced by the involved complicated physical phenomena. Particularly challenging are material flow processes, such as oxidation, reflow, and silicidation, as these require the solution of intricate physical models and the extension of the model-dependent velocity fields to the entire simulation domain at every time step to accurately compute the advection. This velocity extension, thus, introduces yet another computational burden at every time step, which is significant when considering that high-accuracy material flow simulations can easily require several hundred time steps and are applied multiple times in cutting-edge fabrication processes of integrated circuits. In this work, a shared-memory parallel scalar and vector velocity extension algorithm for level-set-based material flow simulations on hierarchical meshes is introduced, allowing to further reduce the turnaround time of TCAD workflows. The performance is evaluated by investigating a representative material flow simulation of 3-D thermal oxidation of silicon. A parallel speedup of 7.1 for the vector-valued extension and 6.6 for the scalar-valued extension is achieved for ten threads; the latter outperforms a previous approach by up to 60%.**

***Index Terms*—Hierarchical meshes, level-set method, material flow process, parallel algorithm, process technology computer-aided design (TCAD), shared-memory, thermal oxidation, velocity extension.**

I. Introduction

SINCE the first introduction of the level-set method [1], it has rapidly evolved into one of the core methods for implementing high-accuracy 3-D simulations in the field of process technology computer-aided design (TCAD) [2]–[7]. The key advantage is the ability to implicitly represent materials and describe their evolution without self-intersections in three dimensions. In particular, etching, deposition, and material flow processes (e.g., oxidation, reflow, and silicidation) benefit from the method's inherent robustness with respect to topological changes [8]–[15]. However, the latter are particularly challenging as they require, aside from the already computationally demanding solution procedures of the physical models, the computation of physical model-specific velocity fields for the entire simulation domain to correctly describe the involved advection. Cutting-edge integrated circuits typically require hundreds of process steps of which many necessary high-accuracy material flow steps can each easily require several hundred time steps: a computationally inefficient single advection step, thus, significantly increases the overall turnaround time of entire TCAD workflows. Further increasing the challenge is the fact that modern fabrication processes involve intricate 3-D geometries and processing conditions; both require high accuracies and, thus, in general, introduce a significant computational burden on the simulation.

A key approach to balance the need for high accuracy with practical simulation run-times is to optimize the data structure [16]. When considering finite-difference discretization-based solution schemes (favorable for level-set methods), a particularly attractive approach is the use of hierarchical meshes [17]–[19], which represents a stack of Cartesian discretizations (meshes based on cuboid cells) with different resolutions. Hierarchical meshes allow to locally increase the mesh resolution (key to high simulation accuracy) in areas of particular interest, such as corners and edges of the level-set interfaces. However, more importantly and as previously indicated, the involved algorithms of the level-set advection step need to be efficiently parallelized, considering the broad availability of shared-memory parallel computing platforms. Recent advances focused on redistancing (for reviews see [16], [20]), particularly considering hierarchical meshes [21]–[23]), and the flux calculation step by, for instance, ray-tracing on explicit interfaces [24], [25] or by importance sampling based on machine learning [26].

In this work, a novel parallel velocity extension algorithm for hierarchical meshes is presented to further accelerate the turnaround time of material flow process TCAD simulations. Contrary to previous work where the focus was solely on scalar velocity fields and single Cartesian meshes [27], [28], here,

Manuscript received February 26, 2021; revised April 23, 2021; accepted May 29, 2021. Date of publication June 18, 2021; date of current version October 22, 2021. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. The review of this article was arranged by Editor S. M. Cea. *(Corresponding author: Michael Quell.)*

Michael Quell and Josef Weinbub are with the Christian Doppler Laboratory for High Performance TCAD, Institute for Microelectronics, TU Wien, 1040 Vienna, Austria (e-mail: michael.quell@tuwien.ac.at; josef.weinbub@tuwien.ac.at).

Vasily Suvorov and Andreas Hössinger are with Silvaco Europe Ltd., Saint Ives PE27 5JL, U.K. (e-mail: vasily.suvorov@silvaco.com; andreas.hoessinger@silvaco.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TED.2021.3087451.

Digital Object Identifier 10.1109/TED.2021.3087451

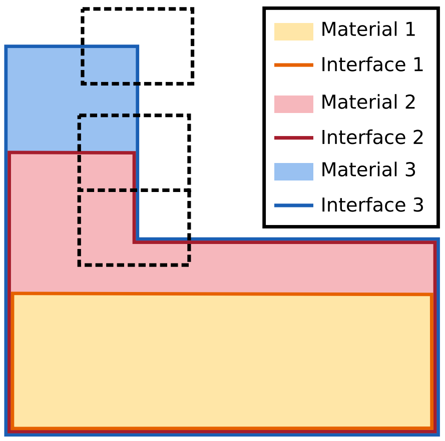

Fig. 1. Example multilayer structure for three materials (colored areas). The three level-set functions (outline of the colored areas) represent the interfaces. The dashed boxes show regions of special interest (corners), where meshes with a higher spatial resolution are required.

we introduce a new parallel algorithm for extending scalar and vector velocity fields to hierarchical Cartesian meshes.

The developed algorithm is evaluated by investigating a material flow process simulation of 3-D thermal oxidation of silicon [29]. This is a representative application case as it jointly involves a scalar and a vector velocity field. Other material flow process simulations only require either of those: thermal oxidation of silicon, thus, represents the most challenging scenario from a computational point of view and is, therefore, perfectly suited to evaluate the performance of our parallel velocity extension algorithm.

This work is organized as follows. Section II provides a short overview of the material handling and the relation to the velocity field. To provide context, Section III summarizes the modeling aspects of thermal silicon oxidation, whereas the actual evaluation benchmark is specified in Section IV. The main contribution of this work, the parallel velocity extension algorithm for hierarchical meshes, is introduced in Section V and evaluated with respect to parallel performance and the silicon oxidation benchmark in Section VI.

II. Material Description and Velocity Field

Materials (e.g., silicon and silicon oxide) are represented in a multilayer structure in level-set process TCAD (see Fig. 1). The multilayer structure consists of a set of orientable interfaces Γ_i represented by level-set functions

$$\phi_i(\vec{x}, t) \begin{cases} < 0, & \vec{x} \text{ inside} \\ = 0, & \vec{x} \text{ on } \Gamma_i \\ > 0, & \vec{x} \text{ outside.} \end{cases} \tag{1}$$

Typically, materials are enclosed by two level-set functions. If a level-set function fulfills $\phi(\vec{x}, t) = \pm\text{dist}(\vec{x}, \Gamma)$, the level-set function has the signed-distance property. The signed-distance property normalizes the gradient of ϕ, which is beneficial for numerical stability. The materials are evolved by advecting each ϕ driven by a velocity field, which is determined by a physical model, describing the involved physical processes (e.g., etching, deposition, and material flow). The velocity field may either be a scalar field, a vector field, or a combination of both, depending on the underlying physical model.

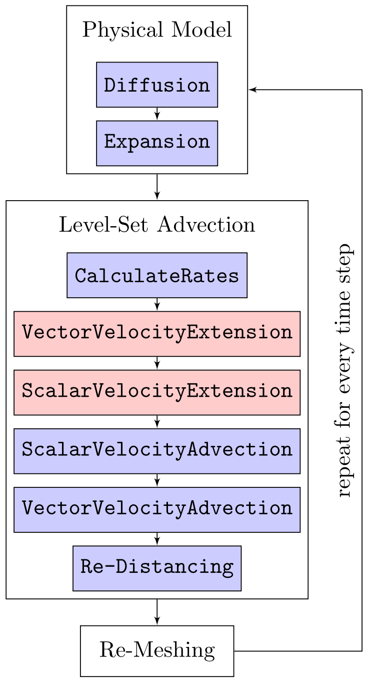

Fig. 2. Simulation flow of a typical level-set-based thermal oxidation simulation. The focus of this work is on the two velocity extension steps shown in red.

III. Thermal Oxidation Simulation

A typical thermal process simulation, which is considered as evaluation target in this work, consists of three main parts: 1) the physical model; 2) the level-set advection; and 3) the remeshing, as discussed in detail in the following sections. The simulation flow-graph with the different steps is depicted in Fig. 2.

A. Physical Model

This section provides a brief summary of the involved modeling aspects for describing thermal oxidation to establish a relation between the physical models and the level-set required velocity field. An extended description of the modeling is provided, for instance, in [10].

The thermal oxidation simulation deals with two physical problems: 1) transport and reaction of the oxygen (diffusion) and 2) volume expansion due to the chemical conversion (e.g., silicon to silicon dioxide) accompanied by material flow (displacement) of all materials above the reactive material (reaction).

Problem 1, the oxidant diffusion through the oxide, is mathematically described by a Poisson equation

$$\frac{\partial}{\partial x_i}\left(D\frac{\partial C}{\partial x_i}\right) = 0 \tag{2}$$

$$-D\frac{\partial C}{\partial \vec{n}}\bigg|_{\text{Si/SiO}_2} = kC, \quad -D\frac{\partial C}{\partial \vec{n}}\bigg|_{\text{SiO}_2\text{/Si}} = h(C_0 - C) \tag{3}$$

with C being the oxidant concentration, D the diffusion coefficient, k the reaction rate, h the gas-phase mass-transfer coefficient, C_0 the equilibrium concentration in the oxide, and $\vec{n}$ the normal to the corresponding interface. This gives a reaction rate at the silicon interface, which is ultimately transformed to a scalar velocity field v at the gas interface.

Problem 2, the volume expansion from the chemical reaction, causes a displacement of materials, which is mathematically described by a creeping flow

$$\frac{\partial S_{ij}}{\partial x_i} = 0 \tag{4}$$

with $S_{ij} = -p \cdot \delta_{ij} + \sigma_{ij}$ denoting the Cauchy stress tensor, p the pressure, and δ_{ij} the Kronecker delta. The shear tensor σ_{ij} uses the Maxwell viscoelastic fluid model. Further simplifications (detailed in [10]) yield the systems of Stokes equations

$$\mu \Delta \vec{v} = \nabla p \tag{5}$$
$$\nabla \cdot \vec{v} = 0 \tag{6}$$

with $\vec{v}$ being the desired vector velocity field and μ the dynamic viscosity.

B. Level-Set Advection

In general, the level-set advection handles the evolution of interfaces (representing multiple materials) based on the solutions of both (in the here considered case) physical problems, i.e., the scalar and vector velocity fields. As the first step, the velocities from the physical models are transferred to *cross points* (i.e., points[1] where a line connecting two neighboring points of the mesh intersects an interface) for each interface. The *cross points* itself are not part of a mesh; their sole purpose is data exchange with the physical model (see Fig. 2: `CalculateRates`).

Based on the calculated *cross point* velocities, the velocities of both physical problems are extended from all the *cross points* to the *interface points* (i.e., points of the mesh, which have a neighboring point on the other side of the interface)[2] In a final velocity extension step (and the focus of this work), the velocities are extended from the *interface points* to the entire hierarchical mesh (see Fig. 2: `ScalarVelocityExtension` and `VectorVelocityExtension`). The details of the velocity extension step are discussed in Section V.

After the velocity extensions, the level-set equations for the scalar velocity case (see Fig. 2: `ScalarVelocityAdvection`)

$$\frac{\partial}{\partial t}\phi(\vec{x}, t) = |\nabla \phi(\vec{x}, t)| v \tag{7}$$

and for the vector velocity case (see Fig. 2: `VectorVelocityAdvection`)

$$\frac{\partial}{\partial t}\phi(\vec{x}, t) = \nabla \phi(\vec{x}, t) \cdot \vec{v} \tag{8}$$

are solved.

[1]We consider points to be cell-centered.

[2]The extension from the *cross point* velocities to the *interface point* velocities is straightforward [30] and negligible with respect to computational effort and, thus, not further considered in this work.

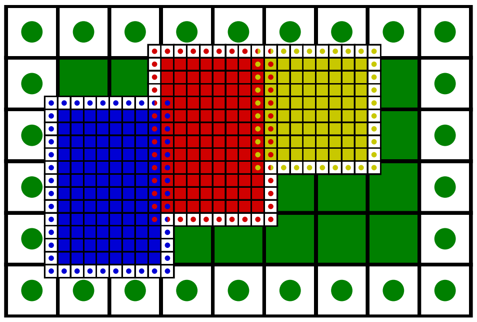

Fig. 3. 2-D schematic example of a mesh hierarchy: the coarsest mesh (green) covers the entire computational domain, and its ghost points (dots in empty cells) are determined by the domain boundary conditions. On the finer level, three exemplary meshes with arbitrary sizes and positions (depicting a remeshing outcome; see Section III-C), offering a four-times increased spatial resolution than the base mesh, are shown (blue, red, and yellow). The respective ghost points (in empty cells) are set by interpolation from the coarser mesh, and the background-colored ghost points are part of overlapping areas between meshes and, thus, have to be synchronized (see Section V-B).

During the advection of the level-set equations, the signed-distance property of the level-set functions is distorted [31] and, thus, must be restored by a redistancing step (see Fig. 2: `Redistancing`) [21]–[23].

C. Remeshing

The remeshing part (see Fig. 2: `Remeshing`) identifies interface regions, which requires a higher spatial resolution and places the meshes within the mesh hierarchy accordingly. As previously indicated, hierarchical meshes consist of levels of different spatial resolutions (see Fig. 3). In this work, we consider a mesh to represent a 3-D Cartesian-discretized rectangular domain with surrounding *ghost points* (ghost layer). The *ghost points* are used to set the boundary conditions or to exchange data with neighboring meshes. On the coarsest level (Level 0), a single mesh covers the entire computational domain. A finer level, such as Level 1, has, e.g., four times the spatial resolution than its corresponding coarser level. This scheme is extendable to several levels, depending on the need to balance accuracy with performance. Meshes on the same level must not overlap (except for their *ghost points*). In this work, the remeshing is automatically handled by the used simulation tool, Silvaco's *Victory Process* [18] (see Section VI), and the therein implemented remeshing algorithm [10].

IV. Thermal Oxidation Benchmark

The computational domain has a length of 1.6 μm and a width of 0.8 μm (see Fig. 4). The domain boundary conditions are symmetric.

The structure is vertically organized as follows (from bottom to top): 0.3-μm bulk silicon (silicon), L-shape 0.02-μm padding silicon dioxide (SiO_2), 0.1-μm buffer poly silicon (polysilicon), and 0.15-μm hard mask silicon nitride (Si_3N_4). A 15-min thermal oxidation process at a temperature of 1000 °C is considered. Fig. 4 shows the topography before and Fig. 5 after the oxidation process.

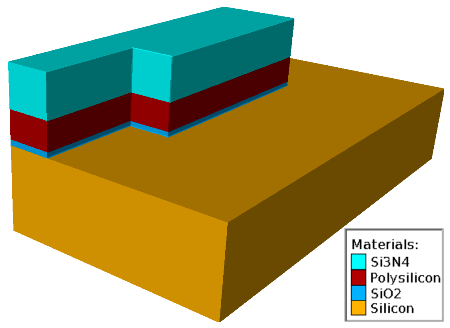

Fig. 4. Device structure before the considered thermal oxidation benchmark process.

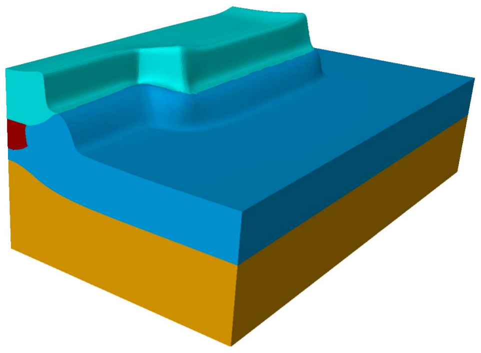

Fig. 5. Device structure after the considered thermal oxidation benchmark process.

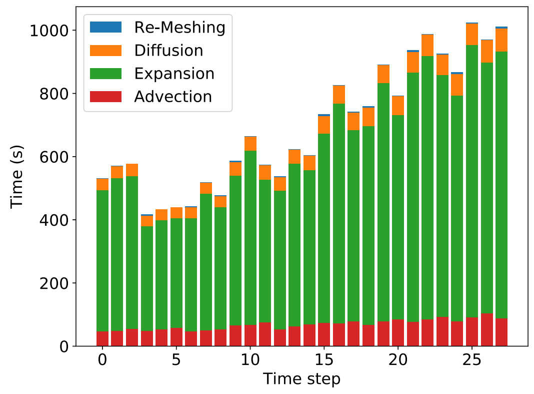

Fig. 6. Serial run-time contributions according to Fig. 2 for all time steps of the considered thermal oxidation benchmark process.

A. Serial Run-Time Analysis

As reference, the total run-time for a serial simulation shown for individual time steps is presented in Fig. 6. The main contributions to the run-time are (ordered): 1) `Expansion` (see Section III-A); 2) `Level-Set Advection` (see Section III-B); and 3) `Diffusion` (see Section III-A). `Remeshing` (see Section III-C) has negligible impact on the overall run-time of the thermal oxidation simulation. Indeed, the results show that the majority of computational time is spent in solving the physical models (in this case, primarily `Expansion`). Dedicated acceleration methods for solving the physical models have already been developed but are not the topic of this work. Instead, this work focuses on further increasing the efficiency of the level-set advection step, in particular, the velocity extension. As discussed before

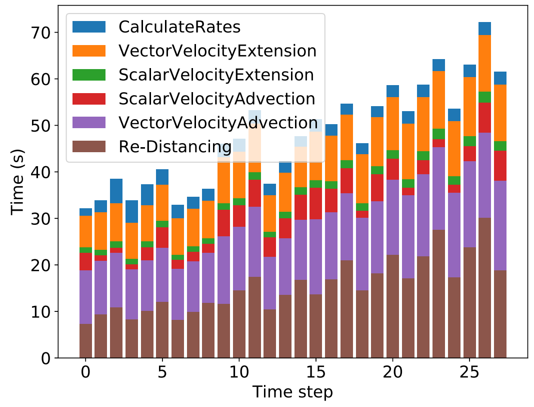

Fig. 7. Serial run-time contributions according to Fig. 2 for all time steps of the individual `Level-Set Advection` steps representing the *status quo*.

(see Section I), the fact that a typical high-accuracy material flow process is applied multiple times during cutting-edge fabrication processes and each material flow process itself can easily require several hundred time steps necessitates increasing the efficiency of all involved algorithms.

As `Level-Set Advection` consists of several algorithmic steps (see Fig. 2), Fig. 7 shows a detailed overview of the individual run-time contributions. `Vector VelocityExtension` is placed third and uses the original serial algorithm presented in [30] based on the fast marching method (FMM), using a single global heap covering all meshes on the same level. `ScalarVelocityExtension` is considerable faster than its vector counterpart as its implementation is based on the improvements to the velocity extension on a single mesh from [27].

The depicted results show that there is considerable potential for further performance improvements, as discussed in the remainder of this work.

V. Parallel Velocity Extension on Hierarchical Meshes

This section introduces the main contribution of this work, i.e., a parallel velocity extension algorithm for hierarchical meshes. The velocity extension is performed according to the partial differential equation

$$\nabla\phi(\vec{x},t)\cdot\nabla v(\vec{x},t)=0 \tag{9}$$

which minimizes the distortions of the signed-distance property [30]. In the case of a vector-valued velocity, (9) is solved for each component of the vector as in the scalar case. This extension is equivalent to a constant extension of the velocity along with interface normal vectors. Equation (9) is discretized using a first order upwind scheme, as proposed in [30].

Higher order schemes, as presented in [32] and [33], are not considered because of their higher computational cost and due to the fact that the overall simulation accuracy is typically limited by the solution of the physical model and not from the accuracy of the velocity extension. Alternatively, an approach based on the fast sweeping method has been developed [34], which, however, requires 2^d iterations (with d referring to the

Algorithm 1 ExtensionHierarchical

```
 1: setBoundaryConditionsOnLevel 0
 2: for all Levels do                                  ▷ From coarsest to finest
 3:     for all Meshes on Level do                     ▷ Parallel region
 4:         WQ ← InitialPoints                         ▷ Create task
 5:         ExtensionMesh(Mesh,WQ)
 6:     Synchronization                                ▷ Barrier
 7:         ▷ Red lines are removed in the proposed algorithm,
    as data exchange is now handled in Algorithm 2.
 8:     WQ ← Exchanged ghost points
 9:     while WQ ≠ ∅ do
10:         for all Meshes on Level do                 ▷ Parallel region
11:             ExtensionMesh(Mesh,WQ)                 ▷ Create task
12:         Synchronization                            ▷ Barrier
13:         WQ ← Exchanged ghost points
14:         setBoundaryConditionsOnNextLevel
```

Algorithm 2 ExtensionMesh(Mesh, WQ)

```
 1: while WQ not empty do
 2:             ▷ Blue lines are added in the proposed algorithm
 3:     if WQ.length > limit then
 4:         $WQ_1$,$WQ_2$ ← Split WQ
 5:         ExtensionMesh(Mesh, $WQ_1$)                ▷ Create Task
 6:         WQ ← $WQ_2$
 7:     for all neighboring points [np] of WQ.front do
 8:         if ExtensionPoint(np) then
 9:             WQ.push(np)
10:             if Overlap(np) then
11:                 EQ.push(neighboring mesh, np)  ▷ EQ gathers
    overlapping points in one local queue per neighboring mesh
12:     for all neighboring meshes [nm] of Mesh do
13:         ExtensionMesh(nm, EQ(nm))                  ▷ Create Task
```

number of spatial dimensions) per point: in comparison, our approach processes each point at most d times.

Extensions using the biharmonic method, as presented in [35], are not possible in the here considered case. Such a method requires the velocity to be fully available on one side of the interface, whereas we consider the case of the velocity being only available at the interface. Approaches based on projecting points directly onto the interface, as in [36] and [37] for redistancing, are embarrassingly parallel (i.e., parallelizable with little or no effort: straightforward independent computations and very low communication needed [38]), but suffer from inaccuracies, especially at sharp corners.

In what follows, the three involved algorithmic steps of the developed parallel velocity extension are discussed in detail: Algorithm 1: extension on the hierarchical levels (`ExtensionHierarchical`); and Algorithm 2: extension on a single mesh (`ExtensionMesh`) and extension on a single point (`ExtensionPoint`). The latter has been introduced in detail in [27] and is, thus, not further discussed here.

A. Extension on the Hierarchical Levels

Algorithm 1 processes the levels of the hierarchical mesh in top-down order, starting from the coarsest mesh covering the full computational domain, down to the finest level covering only regions with the highest interest (e.g., corners). This order is necessary as, on the finer levels, the boundary conditions are defined by the solutions on the coarser levels via interpolation (see Fig. 3).

For every mesh on a level, a parallel task is created (line 4), which, in our case, represents an independent set of computations, along with a dedicated work queue (WQ): WQ is a *first-in–first-out* queue that tracks the points from which the velocity has to be extended. A WQ is initialized with `InitialPoints` of the given mesh, consisting of points for which the velocity is known either because they are *interface points* (see Section III-B) or because they belong to a part of a ghost layer, which is interpolated from a coarser level. In the second step, `ExtensionMesh` (see Section V-B) is executed (line 5), which processes its WQ until it is empty. Global synchronization is necessary before continuing to ensure proper computation (line 6). The original implementation based on the FMM (see Section IV-A) performs the steps highlighted in red (lines 7–13), which are not required anymore as data exchange is now handled in Algorithm 2 to avoid global synchronizations, which allows higher parallel efficiency. Points that have been computed in overlap areas (ghost points) are exchanged between the meshes on the same level and added to the WQs of the neighboring meshes. Next, `ExtensionMesh` is restarted again for all meshes in parallel. The process of restarting, synchronizing, and exchanging is concluded when all WQs is empty. Finally, the boundary conditions are set on the next finer level via interpolation; this step is also required in the new algorithm (line 14).

B. Extension On a Single Mesh

The velocity extension operating on a single mesh (see Algorithm 2) is the key contribution of this work. The main advantages over the previous approach [27] (changes indicated by the blue colored lines in Algorithm 2) are the integrated support to directly exchange data with neighboring meshes without the necessity of global synchronization and splitting the WQ to enable higher parallel efficiency.

The algorithm receives a WQ of points for which the velocities are known. If the queue contains more than `limit` elements, the WQ is split into two WQs pushing the first half into a new WQ named WQ1 (lines 3 and 4). The newly created WQ1 is then processed in a recursive call to Algorithm 2 (line 5). The recursive call is put into a new task, which is executed by the next available thread, allowing for parallelism. The second half WQ2 is renamed to WQ and the algorithm continues (line 6). Splitting the WQ in the proposed manner preserves the spatial locality of the WQs as points closely positioned within a WQ are typically also close in space: newly inserted points most likely have a common neighbor with the previously inserted point as their insertion is triggered by the same point. Setting the `limit` with a central processing unit's (CPU's) core cache size in mind allows for performance optimizations. In locally concave regions, the WQ typically grows in size, while, in locally convex regions, the WQ shrinks.

For all neighboring points of the current front point (the point that just has been extracted from the queue) of the WQ, `ExtensionPoint` is executed to compute the extended velocity (line 8). The `ExtensionPoint` may *fail* (i.e., return `false`) if, e.g., another task was faster in computing the velocity, or an upwind neighboring point of the considered neighboring point has not been computed yet. The extended discussion on the conditions for success or failure is given in [27]. In the case of success, the neighboring point is pushed onto the WQ (line 9). In addition, we added a check if the neighboring point is in an overlap area with a neighboring mesh. Considering a 3-D domain, a point can be assigned to up to three overlap areas. If so, the neighboring point is also pushed to an exchange queue (EQ; lines 10 and 11): The EQ collects all points to be exchanged with a specific neighboring mesh. Once the WQ is empty, for each nonempty EQ, a recursive call to Algorithm 2 with the EQ's corresponding mesh is created in a new task, allowing further parallel execution (lines 12 and 13).

The collection of points of the overlap areas in an EQ and using them in a single recursive call per neighboring mesh reduces the task creation overhead by avoiding scenarios where a task with a WQ consisting only of a single point is encountered. This strategy allows the algorithm to propagate the solution to all meshes without the necessity of mesh activation strategies used, e.g., in [39] and [40] or global synchronization (see Section V-A).

VI. Parallel Performance Results and Analysis

The performance is measured on a workstation equipped with an Intel Xeon E5-2680 v2 CPU offering ten physical cores (2.8–3.6 GHz) and 226 GB of the main memory. The CPU offers a 32-kB instruction and data L1-cache and 256-kB L2-cache on each core, and a 25.6-MB L3-cache is shared between the cores. To evaluate the impact of the accelerated algorithm on an entire simulation, the developed velocity extension algorithm has been integrated into Silvaco's Victory Process [18] (a 3-D process TCAD simulator) using C++ and OpenMP for shared-memory parallelization. The developed algorithm has been devised with a parallel task concept in mind, which, thus, maps directly to the task mechanism provided by OpenMP. OpenMP tasks allow implementing dynamic load-balancing by using a thread pool. However, the load-balancing is limited if the number of tasks is in the order of the used number of threads. Therefore, a large number of tasks are desired for high parallel efficiency.

The mesh hierarchy of the considered benchmark case consists of a single Level 0 mesh with $40 \times 80 \times 40 \mathrel{\hat{=}} 128\,000$ points, which does not change during the simulation. On Level 1, there are initially 18 meshes with a total of 536 704 points. The `Remeshing` procedure adapts the hierarchical meshes every third time step (empirically chosen, shown to offer a practical balance between accuracy and performance). The number of meshes on Level 1 nearly doubles during the simulation to 34, and the total number of points increases to

TABLE I
Evolution of Number of Meshes and Total Number of Points on Level 1 Over All Time Steps

Time step	Number of meshes	Number of points
0	18	536 704
3	22	492 544
6	18	503 808
9	20	620 544
12	17	566 592
15	28	683 840
18	18	774 016
21	30	764 992
24	33	868 928
27	34	969 536

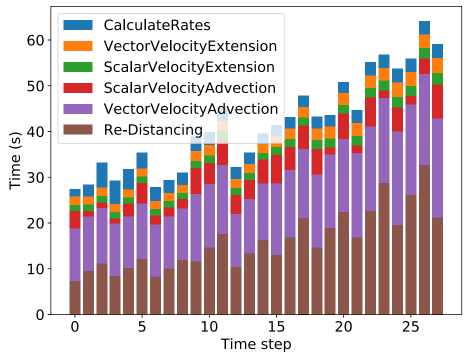

Fig. 8. Serial run-time contributions for all time steps of the individual `Level-Set Advection` steps considering the proposed changes.

969 536 (see Table I) due to an overall increase in complexity of the geometry. However, temporary reductions in mesh numbers are caused by Victory Process' internal remeshing logic (see Section III-C), as evolved topographies allow for remeshing optimizations (e.g., mesh merges). Comparing the number of points to the serial run-time of the different time steps (see Fig. 8) shows that the run-time linearly depends on the number of points.

A comparison between Fig. 7 (conventional velocity extension; see Section IV) and Fig. 8 (proposed algorithm; see Section V) shows that the run-time spent on the vector velocity extension is significantly reduced in the case of serial execution. The improvement is due to using a mesh-local WQ instead of using a global heap of the FMM-based implementation and, thereby, reducing the computational complexity from $\mathcal{O}(N \log N)$ to $\mathcal{O}(N)$. The complexity reduction comes from replacing the heap (priority queue) with a *first-in–first-out* queue [27]. In addition, the mesh-local approach simplifies the neighboring point computations as only those from the same mesh are considered, yielding better cache locality. The serial run-time attributed to the scalar velocity extension is barely affected as its implementation already uses a mesh-local queue.

Increasing the number of threads to ten (fully allocating the CPU's physical cores) yields the run-times per time step shown in Fig. 9, which, aside from the acceleration of the velocity extension, also shows the parallelization impact of the other involved algorithms (although not the focus of this work) to provide an overview of the entire workflow. The total run-time of the `Level-Set`

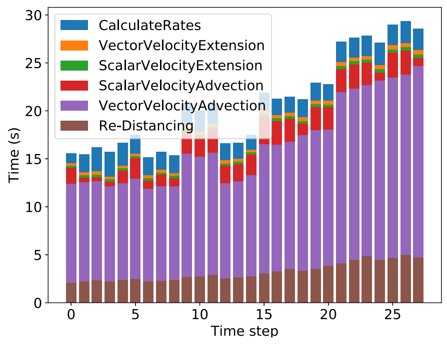

Fig. 9. Parallel run-time contributions using ten threads for all time steps of the individual `Level-Set Advection` steps considering the proposed changes.

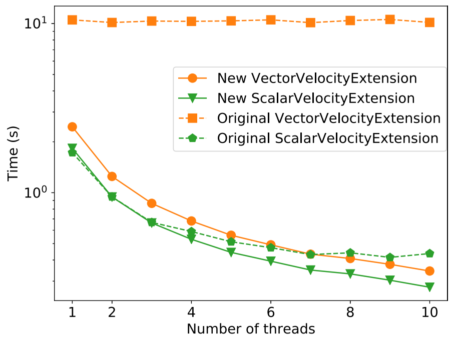

Fig. 10. Average run-time of the velocity extensions. The dashed lines show data from the original implementation of the scalar and vector velocity extensions.

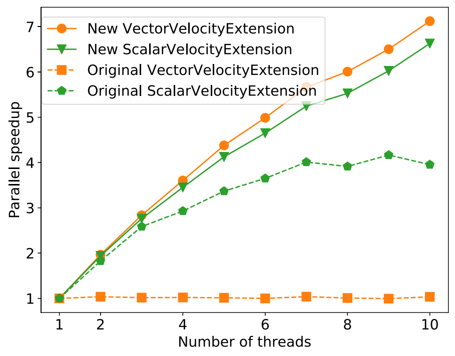

Fig. 11. Average parallel speedup of the velocity extensions. The dashed lines show data from the original implementation of the scalar and vector velocity extensions.

`Advection` is reduced by a factor of 2.4. In this parallel scenario, `VectorVelocityAdvection` dominates the overall run-time per time step.

For a detailed analysis of the parallelization efficiency of the proposed algorithm for the velocity extension, the run-time has been averaged over all time steps and plotted over the number of threads (see Fig. 10).

The run-time of the original `VectorVelocity-Extension` does not change with the number of threads because it is not parallelized. The still visible minor variations of the run-time for different thread numbers are caused by measurement noise. More interestingly, the run-time of the original `ScalarVelocityExtension` algorithm for a single thread is 4% faster than our proposed algorithm.

This slowdown is due to a compromise. There is an additional check if a point is in an overlap area after its velocity has been computed. This check is performed on all points, while the original algorithm would only iterate over points in the overlap area once its computation on a mesh is finished. Sticking with the original algorithm's approach would require every thread working on the same mesh to iterate over all points in the overlap area creating an even bigger slowdown. This shortcoming is easily compensated by better parallelization.

Fig. 11 shows the parallel speedup, which, for `Scalar VelocityExtension`, reaches 6.6 for ten threads, compared to 4.1 of the original implementation, corresponding to a 60% increase. The original `ScalarVelocityExtension` parallel speedup flattens beginning with four threads, as the implicit load-balancing from the thread pool deteriorates due to insufficiently available tasks. The number of tasks created between two global synchronizations in the original algorithm equals the number of meshes, thus ranging from 17 to 34 and, therefore, limiting performance. The workload per mesh differs as they are not equally sized. A naive estimation of the workload per mesh from the mesh size is not feasible, as single points in the ghost layer can have a large influence on how many points are potentially to be computed before the next synchronization. Depending on the time step and the total number of threads, a thread may receive merely a single task, e.g., there are 18 meshes in the first step (see Table I), but ten threads are available in the thread pool, and two threads would only have a single task.

The `VectorVelocityExtension` achieves a parallel speedup of 7.1 for ten threads. The improved parallel speedup for the vector velocity extension over the scalar case is caused by the three times higher computational load for each point due to the considered 3-D problem. While doing three times as many computations for each point, the serial run-times for the scalar and vector extensions differ only by 34%. This shows that most of the run-time is spent on the checks and ordering of the points, instead of actual computations.

VII. Conclusion

We introduced a shared-memory parallel scalar and vector velocity extension algorithm for level-set-based material flow simulations on hierarchical meshes. The performance was evaluated by investigating a representative simulation of 3-D thermal oxidation of silicon. A parallel speedup of 7.1 and 6.6 is achieved for the vector and scalar velocity extension using ten threads, respectively. The parallel speedup of the scalar velocity is a 60% improvement compared to the original implementation.

References

[1] 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, doi: 10.1016/0021-9991(88)90002-2.

[2] D. Adalsteinsson and J. A. Sethian, "A fast level set method for propagating interfaces," *J. Comput. Phys.*, vol. 118, no. 2, pp. 269–277, May 1995, doi: 10.1006/jcph.1995.1098.

[3] D. Adalsteinsson and J. A. Sethian, "A level set approach to a unified model for etching, deposition, and lithography II: Three-dimensional simulations," *J. Comput. Phys.*, vol. 122, no. 2, pp. 348–366, Dec. 1995, doi: 10.1006/jcph.1995.1221.

[4] D. Adalsteinsson and J. A. Sethian, "A level set approach to a unified model for etching, deposition, and lithography," *J. Comput. Phys.*, vol. 138, no. 1, pp. 193–223, Nov. 1997, doi: 10.1006/jcph.1997.5817.

[5] O. Ertl and S. Selberherr, "A fast level set framework for large three-dimensional topography simulations," *Comput. Phys. Commun.*, vol. 180, no. 8, pp. 1242–1250, Aug. 2009, doi: 10.1016/j.cpc.2009.02.002.

[6] S. M. Cea *et al.*, "Process modeling for advanced device technologies," *J. Comput. Electron.*, vol. 13, no. 1, pp. 18–32, 2014, doi: 10.1007/s10825-013-0491-6.

[7] M. Upadhyaya *et al.*, "Level-set multilayer growth model for predicting printability of buried native extreme ultraviolet mask defects," *J. Vac. Sci. Technol. B, Nanotechnol. Microelectron., Mater., Process., Meas., Phenomena*, vol. 33, no. 2, Mar. 2015, Art. no. 021602, doi: 10.1116/1.4913315.

[8] J. A. Sethian and D. Adalsteinsson, "An overview of level set methods for etching, deposition, and lithography development," *IEEE Trans. Semicond. Manuf.*, vol. 10, no. 1, pp. 167–184, Feb. 1997, doi: 10.1109/66.554505.

[9] D. Wheeler, D. Josell, and T. P. Moffat, "Modeling superconformal electrodeposition using the level set method," *J. Electrochem. Soc.*, vol. 150, no. 5, p. C302, 2003, doi: 10.1149/1.1562598.

[10] V. Suvorov, A. Hössinger, Z. Djurić, and N. Ljepojevic, "A novel approach to three-dimensional semiconductor process simulation: Application to thermal oxidation," *J. Comput. Electron.*, vol. 5, no. 4, pp. 291–295, 2006, doi: 10.1007/s10825-006-0003-z.

[11] A. Auge, A. Weddemann, B. Vogel, F. Wittbracht, and A. Hütten, "A level set based approach for modeling oxidation processes of ligand stabilized metallic nanoparticles," *Appl. Phys. Lett.*, vol. 96, no. 9, Mar. 2010, Art. no. 093111, doi: 10.1063/1.3353957.

[12] C. Montoliu, N. Ferrando, M. A. Gosálvez, J. Cerdá, and R. J. Colom, "Implementation and evaluation of the Level Set method: Towards efficient and accurate simulation of wet etching for microengineering applications," *Comput. Phys. Commun.*, vol. 184, no. 10, pp. 2299–2309, Oct. 2013, doi: 10.1016/j.cpc.2013.05.016.

[13] X. Klemenschits, S. Selberherr, and L. Filipovic, "Modeling of gate stack patterning for advanced technology nodes: A review," *Micromachines*, vol. 9, no. 12, p. 631, Nov. 2018, doi: 10.3390/mi9120631.

[14] A. Toifl *et al.*, "The level-set method for multi-material wet etching and non-planar selective epitaxy," *IEEE Access*, vol. 8, pp. 115406–115422, 2020, doi: 10.1109/ACCESS.2020.3004136.

[15] A. Toifl, F. Rodrigues, L. F. Aguinsky, A. Hössinger, and J. Weinbub, "Continuum level-set model for anisotropic wet etching of patterned sapphire substrates," *Semicond. Sci. Technol.*, vol. 36, Apr. 2021, Art. no. 045016, doi: 10.1088/1361-6641/abe49b.

[16] F. Gibou, R. Fedkiw, and S. Osher, "A review of level-set methods and some recent applications," *J. Comput. Phys.*, vol. 353, pp. 82–109, Jan. 2017, doi: 10.1016/j.jcp.2017.10.006.

[17] R. Nourgaliev, S. Wiri, N. Dinh, and T. Theofanous, "On improving mass conservation of level set by reducing spatial discretization errors," *Int. J. Multiphase Flow*, vol. 31, no. 12, pp. 1329–1336, 2005, doi: 10.1016/j.ijmultiphaseflow.2005.08.003.

[18] *Silvaco Victory Process*. Accessed: Feb. 5, 2021. [Online]. Available: https://silvaco.com/tcad/victory-process-3d/

[19] M. Adams *et al.*, "Chombo software package for AMR applications—Design document," Lawrence Berkeley Nat. Lab., Berkeley, CA, USA, Tech. Rep. LBNL-6616E, 2015.

[20] R. I. Saye and J. A. Sethian, "A review of level set methods to model interfaces moving under complex physics: Recent challenges and advances," *Handbook Numer. Anal.*, vol. 21, pp. 509–554, Jan. 2020, doi: 10.1016/bs.hna.2019.07.003.

[21] G. Diamantopoulos, A. Hössinger, S. Selberherr, and J. Weinbub, "A shared memory parallel multi-mesh fast marching method for re-distancing," *Adv. Comput. Math.*, vol. 45, no. 4, pp. 2029–2045, 2019, doi: 10.1007/s10444-019-09683-z.

[22] M. Quell, G. Diamantopoulos, A. Hössinger, S. Selberherr, and J. Weinbub, "Parallel correction for hierarchical re-distancing using the fast marching method," in *Advances in High Performance Computing, Studies in Computational Intelligence*, vol. 902. Cham, Switzerland: Springer, 2021, pp. 438–451, doi: 10.1007/978-3-030-55347-0_37.

[23] M. Quell, G. Diamantopoulos, A. Hössinger, and J. Weinbub, "Shared-memory block-based fast marching method for hierarchical meshes," *J. Comput. Appl. Math.*, vol. 392, Aug. 2021, Art. no. 113488, doi: 10.1016/j.cam.2021.113488.

[24] P. Manstetten, J. Weinbub, A. Hössinger, and S. Selberherr, "Using temporary explicit meshes for direct flux calculation on implicit surfaces," *Procedia Comput. Sci.*, vol. 108, pp. 245–254, Jan. 2017, doi: 0.1016/j.procs.2017.05.067.

[25] P. Manstetten, L. Gnam, A. Hössinger, S. Selberherr, and J. Weinbub, "Sparse surface speed evaluation on a dynamic three-dimensional surface using an iterative partitioning scheme," in *Computational Science—ICCS* (Lecture Notes in Computer Science), vol. 10860. Cham, Switzerland: Springer, 2018, pp. 694–707, doi: 10.1007/978-3-319-93698-7_53.

[26] A. Scharinger, P. Manstetten, A. Hössinger, and J. Weinbub, "Generative model based adaptive importance sampling for flux calculations in process TCAD," in *Proc. Int. Conf. Simul. Semiconductor Processes Devices (SISPAD)*, Oct. 2020, pp. 39–42, doi: 10.23919/SISPAD49475.2020.9241615.

[27] M. Quell, P. Manstetten, A. Hössinger, S. Selberherr, and J. Weinbub, "Parallelized construction of extension velocities for the level-set method," in *Parallel Processing and Applied Mathematics* (Lecture Notes in Computer Science), vol. 12043. Cham, Switzerland: Springer, 2020, pp. 348–358, doi: 10.1007/978-3-030-43229-4_30.

[28] M. Quell, A. Toifl, A. Hossinger, S. Selberherr, and J. Weinbub, "Parallelized level-set velocity extension algorithm for nanopatterning applications," in *Proc. Int. Conf. Simulation Semiconductor Processes Devices (SISPAD)*, Oct. 2019, pp. 1–4, doi: 10.1109/SISPAD.2019.8870482.

[29] D. Guoy *et al.*, "3-D simulation of silicon oxidation: Challenges, progress and results," in *Proc. Int. Conf. Simulation Semiconductor Processes Devices (SISPAD)*, Sep. 2013, pp. 196–199, doi: 10.1109/SISPAD.2013.6650608.

[30] D. Adalsteinsson and J. A. Sethian, "The fast construction of extension velocities in level set methods," *J. Comput. Phys.*, vol. 148, no. 1, pp. 2–22, Jan. 1999, doi: 10.1006/jcph.1998.6090.

[31] M. F. Trujillo, L. Anumolu, and D. Ryddner, "The distortion of the level set gradient under advection," *J. Comput. Phys.*, vol. 334, pp. 81–101, Apr. 2017, doi: 10.1016/j.jcp.2016.11.050.

[32] D. L. Chopp, "Another look at velocity extensions in the level set method," *SIAM J. Sci. Comput.*, vol. 31, no. 5, pp. 3255–3273, Jan. 2009, doi: 10.1137/070686329.

[33] F. de Gournay, "Velocity extension for the level-set method and multiple eigenvalues in shape optimization," *SIAM J. Control Optim.*, vol. 45, no. 1, pp. 343–367, Jan. 2006, doi: 10.1137/050624108.

[34] G. F. Ouyang, Y. C. Kuang, and X. M. Zhang, "A fast scanning algorithm for extension velocities in level set methods," *Adv. Mater. Res.*, vol. 328, no. 1, pp. 677–680, 2011, doi: 10.4028/www.scientific.net/AMR.328-330.677.

[35] T. J. Moroney, D. R. Lusmore, S. W. McCue, and D. L. McElwain, "Extending fields in a level-set method by solving a biharmonic equation," *J. Comput. Phys.*, vol. 343, pp. 170–185, Aug. 2017, doi: 10.1016/j.jcp.2017.04.049.

[36] B. Lee, J. Darbon, S. Osher, and M. Kang, "Revisiting the redistancing problem using the Hopf–Lax formula," *J. Comput. Phys.*, vol. 330, pp. 268–281, Feb. 2017, doi: 10.1016/j.jcp.2016.11.005.

[37] M. Royston *et al.*, "Parallel redistancing using the Hopf–Lax formula," *J. Comput. Phys.*, vol. 365, pp. 7–17, Jul. 2018, doi: 10.1016/j.jcp.2018.01.035.

[38] M. Herlihy and N. Shavit, *The Art of Multiprocessor Programming, Revised Reprint*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012.

[39] J. Yang and F. Stern, "A highly scalable massively parallel fast marching method for the Eikonal equation," *J. Comput. Phys.*, vol. 332, pp. 333–362, Mar. 2017, doi: 10.1016/j.jcp.2016.12.012.

[40] J. Yang, "An easily implemented, block-based fast marching method with superior sequential and parallel performance," *SIAM J. Sci. Comput.*, vol. 41, no. 5, pp. C446–C478, Jan. 2019, doi: 10.1137/18M1213464.