J Comput Electron (2014) 13:877–884
DOI 10.1007/s10825-014-0604-x

# Accelerated redistancing for level set-based process simulations with the fast iterative method

**Josef Weinbub · Andreas Hössinger**

Published online: 14 August 2014


**Abstract** The finite iterative method is compared to an industry-hardened fast marching method for accelerating the redistancing step essential for Level Set-based process simulations in the area of technology computer-aided design. We discuss our implementation of the finite iterative method and depict extensions to improve the method for process simulations, in particular regarding stability. Contrary to previously published work, we investigate real-world structures with varying resolutions, originating from the area of process simulation. The serial execution performance as well as error norms are used to compare our approach with an industry-hardened fast marching method implementation. Parallel scalability is discussed based on a shared-memory OpenMP implementation. We show that our approach of the finite iterative method is an excellent candidate for accelerating Level Set-based process simulations, as it offers considerable performance gains both in serial and parallel execution mode, albeit being inferior with respect to accuracy.

**Keywords** Finite iterative method · Redistancing · OpenMP · Process simulation

## 1 Introduction

A key ability of technology computer-aided design (TCAD) process simulation is to simulate changes in the topography and topology of the device structure through etching and deposition processes or through thermal treatment [1]. Predicting the topography evolution requires a method capable of describing geometric deformations over time. In this respect it is primarily important to trace the motion of interfaces or particularly of the surface of the device structure, giving rise to a so-called boundary tracking problem (also frequently referred to as boundary evolution), of which the Level Set (LS) [2] method is a widely applied technique.

The LS method has been successfully applied over the years both in academia [3–6] as well as in industry applications, such as Silvaco's Victory Process [7]. A central aspect of applying the LS method to keep track of the moving boundary is the so-called *redistancing* step, recomputing the distances between the evolving surfaces and the grid points. The widely applied method to implement this approach is to utilize the fast marching method (FMM). However, the FMM does not favor parallel execution nor large problem sizes in a three-dimensional setting [8]. These facts increase simulation times of the ever-growing simulation challenges—both in size and complexity—of today's TCAD process simulations.

To remedy this problem, this work investigates the application of the fast iterative method (FIM) to the field of TCAD process simulation. In particular, we discuss a shared-memory OpenMP parallelized implementation of the FIM and apply it to a set of real-world surface evolution cases, offering different problem sizes typical to the area of process simulation. We compare serial run-time performance and accuracy with an industry-hardened implementation of the FMM and investigate parallel scalability.

In Sect. 2 we provide a short overview of the LS method and the redistancing step. In Sect. 3 our approach for a shared-memory implementation of the FIM is discussed. In Sect. 4 execution performance and quality is compared to a reference industry implementation of the FMM.

J. Weinbub (✉)
Institute for Microelectronics, TU Wien, Vienna, Austria
e-mail: weinbub@iue.tuwien.ac.at

A. Hössinger
Silvaco Europe Ltd., St Ives, Cambridge, UK
e-mail: andreas.hoessinger@silvaco.com

## 2 Background

This section introduces the underlying key topics and gives an overview of already conducted research work. We sketch the LS method and elaborate on the necessity to redistance the grid relative to the evolved surface. The discussion is followed by introducing the principles of the FMM as well as the FIM and an overview of other redistancing methods not considered in this work.

### 2.1 Level set method

Within the concept of the LS method a *surface* (or more generally an *interface*) $\mathcal{S} = \partial\mathcal{M}$ of a region $\mathcal{M}$ is implicitly described by a continuous function $\Phi(\mathbf{x})$. The surface is described as $\Phi(\mathbf{x}) = 0$ and the interior of the region is identified by $\Phi(\mathbf{x}) \leq 0$, $\mathbf{x} \in \mathcal{M}$. Boundary tracking relates to an initial surface $\mathcal{S}(t = 0)$, where the evolution of this surface until the end of the process time $\mathcal{S}(t_p)$ is specified. This specification is achieved by calculating surface velocities $V_S(\mathbf{x}_S)$ of the surface $\mathcal{S}(t = t_i)$ in the normal direction at every time step and by moving the surface accordingly.

Using the implicit representation of a surface $\mathcal{S}(t)$, the time evolution of the surface driven by a scalar velocity field $V(\mathbf{x})$, which can be obtained from $V_S(\mathbf{x}_S)$ by velocity extension [9], can be described by the LS equation:

$$\frac{\partial\Phi}{\partial t} + V(\mathbf{x})\|\nabla\Phi\| = 0$$

However, the signed distance function $\Phi(\mathbf{x}, t)$ gets distorted when solving the LS equation for moving the interfaces within the implicit geometry representation. In order to reconstruct the signed distance function property, redistancing algorithms must be applied while solving the boundary tracking problem. Redistancing means to solve the Eikonal equation [10] :

$$\|\nabla\Phi(\mathbf{x})\| = f(\mathbf{x}), \quad \Phi|_{\partial\mathcal{M}} = 0$$

with a speed function $f(\mathbf{x}) = 1$ and with a fixed set of boundary points. The solution of the Eikonal equation yields the the signed distance field, holding the signed distances of any point $\mathbf{x}$ to the surface $\mathcal{S}$.

The most popular method for solving the Eikonal equation is the already indicated FMM [11].

### 2.2 Fast marching method

Initially the FMM was developed for a rectangular orthogonal mesh [11], while later it was extended to other mesh types, such as unstructured meshes [12] and nested Cartesian meshes [13]. The FMM is a Dijkstra-type method, as the principle is similar to Dijkstra's shortest-path graph algorithm [14,15]. The order of which the wave-front passes through the grid nodes of the simulation domain determines the order of computing the solution updates on the respective nodes. The set of nodes being currently updated within an iteration is denoted as *narrow band*, being typically implemented as a heap data structure. This heap data structure represents a computationally significant part of this method, allowing to identify the point with the smallest signed distance from a list of available points. However, the heap imposes a bottle neck, as the causality principle has to be uphold, impeding reasonable parallel scalability [8].

Despite this fact parallel implementations of the FMM have been investigated. Typical approaches to parallelize the FMM are based on the domain decomposition method, where the simulation domain is decoupled into distributed sub-domains [16,17]. Each sub-domain hosts its own heap data structure. Additionally, so-called ghost layers are utilized at the interfaces to adjacent sub-domains, ensuring solution continuity. The scalability of the reported domain decomposition algorithm is strongly dependent on load balancing with respect to the number of mesh points on each side of the evolving interface. Therefore, such a domain decomposition approach is reasonable for small-scale parallelization albeit requiring significant development overhead.

A non-domain decomposition approach based on a shared-memory programming model and Cartesian meshes has been investigated [18]. In contrast to domain decomposition approaches the interfaces are distributed. A shared data structure is utilized—governed by a mutex lock—to satisfy the causality principle. Mutex locks, however, are well-known to impede scaling behavior. The approach does not tackle the load-balancing problem satisfactorily, resulting in limited scalability. For instance, a part of the interface assigned to a thread might leave the simulation domain, leaving the thread jobless.

The FMM has been investigated with respect to parallelization based on accelerators [19,20]. However, induced by the sequential nature of the FMM the reported results underline that the FMM method is not particularly suited for parallelization let alone large-scale parallelization on graphics adaptors.

### 2.3 Fast iterative method

The FIM has been originally implemented for parallel execution on Cartesian meshes [8] and later extended to triangular surface meshes [21]. This method relies on a modification of a label correction scheme coupled with an iterative procedure for the mesh point update.

The inherent high degree of parallelism is due to the ability of updating the mesh points in parallel, supporting a *single instruction, multiple data* parallel execution model. Therefore, the FIM favors implementations on highly parallel accelerators, such as graphics adaptors [20,22].

Although the FIM has been originally developed for large-scale parallelism on graphics adaptors, the inherent support for parallelism obviously also favors small-scale parallel shared-memory implementations.
This aspect is of particular interest to this work, as shared-memory implementations, e.g., based on OpenMP, are easier to develop, to maintain, and to deploy compared to accelerator-based implementations.

Investigations on shared-memory approaches have been conducted, discussing generic test cases for two- and three-dimensional problems [23,24]. Our work will continue this previous work towards actual applications by investigating the applicability of the FIM to real-world problems in the area of process TCAD.

### 2.4 Other Eikonal solution methods

Aside of the FMM and the FIM other methods to solve the Eikonal equation are available. In particular, the fast sweeping method (FSM) [25] has been developed to enable parallel redistancing, albeit favoring small-scale shared-memory parallelization. The FMM, FIM, and the FSM have been compared in the context of computer tomography [20]. The original FSM has been extended with respect to parallel scalability on shared-memory architectures [26]. In our work we will not investigate the FSM but focus entirely on the FIM. The FIM's inherent support for large-scale parallelization on accelerators, such as graphics adaptors, offers excellent potential for future extensions towards accelerator-based implementations.

Other available methods to solve the Eikonal equation are the algebraic Newton method, brute force redistancing, and via a reformulation to a hyperbolic problem [27]. The algebraic Newton method cannot be seen as a truly general method to compute distance functions for arbitrary distance fields, as the distance fields must be smooth. If in fact the distance field is non-smooth, as would be the case, for instance, at corners of a square, the method would fail. Brute force redistancing has a complexity of $\mathcal{O}(N \cdot M)$, $N$ being the number of mesh points and $M$ being the number of zero distance segments. The brute force algorithm thus exhibits a poor scaling behavior due to quadratic computational costs for high numbers of interface segments. Also, this particular method requires an explicit representation of the interface. It's extraction is non-negligible and thus introduces considerable computational overhead. Finally, the Eikonal equation can be reformulated into a hyperbolic partial differential equation, enabling the solution via parallelized discretization schemes, such as the finite element method [28,29]. However, such an approach suffers from problems regarding interface movement due to numerical truncation errors [30]. Because of the mentioned issues, the algebraic Newton, brute force redistancing, and hyperbolic reformulation methods are not considered in this work.

## 3 Our approach

Our implementation is based on a C++ OpenMP approach on top of regular Cartesian grids. We use a similar approach for a shared-memory FIM implementation as reported in [23,24], albeit using the original FIM/FSM approach to solve the Eikonal equation [8,25], being based on a Godunov upwind difference scheme. In the following we focus on our extensions which are essential to make the FIM a viable candidate for redistancing the signed distance fields of LS-based process simulations. As already indicated, LS-based simulations yield a description of the evolving surface.

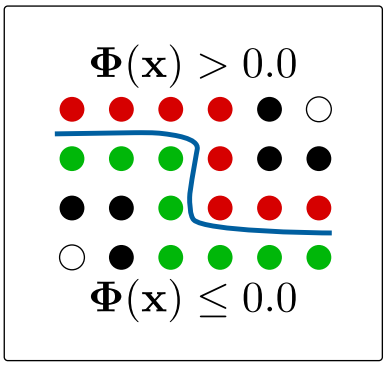


**Fig. 1** A discrete grid stores signed distance values relative to a surface (*blue line*). For the forward and backward direction all source values with positive (*red nodes*) and negative or zero signed (*green nodes*) distances, respectively, are used as initial condition. A so-called active list is generated consisting of nodes (*black nodes*) adjacent to nodes for which the distances are already known, i.e., in the beginning the neighbors of the initial source node set. For the nodes in the active list the signed distances are computed within an iteration. The iterative scheme gradually processes all unprocessed nodes (*circles*), however, the algorithm must not advance into the opposite region, to ensure that the interior and the exterior region is successfully identified (Color figure online)

**Algorithm 1** Forward/Backward initialization

```
1: General: X: set of grid nodes, L: active list,
2:    S: set of source nodes, F: set of source distances,
3:    U: set of distances, #: comment, nb: neighbor
4: for all x ∈ X do
5:    U(x) = ∞
6: end for
7: for all x ∈ S do
8:    if x > 0.0 then {#Backward x ≤ 0.0}
9:       U(x) = F(x)
10:      for all adjacent x_nb of x do
11:         if x_nb ∉ S then {#Direction}
12:            add x_nb to L
13:         end if
14:      end for
15:   end if
16: end for
```

**Algorithm 2** Stabilized and direction-aware FIM

```
1: while L ≠ ∅ do
2:   L_New = ∅
3:   for all x ∈ L in parallel do
4:     p = U(x)
5:     q = solveEikonal(x)
6:     if |p − q| < ε then
7:       U_New(x) = q
8:       for all adjacent x_nb of x do
9:         if x_nb ∉ L then
10:          if x_nb ∉ S then {#Direction}
11:            p = U(x_nb)
12:            q = solveEikonal(x_nb)
13:            if p − q > ε then {#Stabilization}
14:              U_New(x_nb) = q
15:              add x_nb to L_New
16:            end if
17:          end if
18:        end if
19:      end for
20:    else
21:      if q < U_New(x) then
22:        add x to L_New
23:      end if
24:      U_New(x) = q
25:    end if
26:  end for
27:  U = U_New
28:  L = L_New
29: end while
```

More concretely, this description is in fact represented by a signed distance field of two node sets surrounding the actual surface.

In our LS-based context, we need to redistance the interior ($\Phi(\mathbf{x}) \leq 0$) and exterior ($\Phi(\mathbf{x}) > 0$) of the problem domain. Therefore, we need to perform two passes to compute the entire signed distance field, one in forward direction (i.e. exterior) and one in backward direction (i.e. interior). To this end, the original FIM algorithm must be extended to avoid advancing into the wrong direction. This is done by evaluating whether a new node to be inserted into the *active list* (holding the narrow band) is part of the source node set (Fig. 1).

Algorithm 1 depicts the initialization of both passes, considering the forward/backward scenario as well as direction handling. Algorithm 2 introduces our adapted redistancing algorithm, which is based on the approach introduced in [24]. Aside from ensuring the correct advancement of the active list (Line 10) we adapted the original condition $p > q$ for the neighbor nodes to a numerically more stable version, being $p - q > \varepsilon$ (Line 13). We experienced convergence problems of the iterative scheme when using the original condition, which we attribute to numerical insufficiencies of the discretization scheme.

## 4 Results

To investigate the applicability of the FIM to the area of process simulation, we use four different structures for which we compute the signed distance fields (Figs. 2, 3, 4, 5 show the structures and the isosurfaces of the signed distance fields). For the presented results we used $\varepsilon = 10^{-10}$. In Table 1 we compare our approach of the FIM with an industry-hardened FMM implementation on a 64-bit Linux workstation, powered by an Intel i5-3570 (four cores, 3.4

**Table 1** Comparison of serial execution performance as well as evaluation of the $l_1$ ($\|u\|_1$), $l_2$ ($\|u\|_2$), and $l_\infty$ ($\|u\|_\infty$) error norms based on four real-world structures, each evaluated in two different grid size resolutions

| Structure | Size | FMM Forward (s) | FIM Forward (s) | FIM Forward* (s) | FMM Backward (s) | FIM Backward (s) | FIM Backward* (s) | FMM Total (s) | FIM Total (s) | FIM Total* (s) | Speedup* | $\|\mathbf{u}\|_1$ ($\cdot 10^{-3}$) | $\|\mathbf{u}\|_2$ ($\cdot 10^{-3}$) | $\|\mathbf{u}\|_\infty$ ($\cdot 10^{-2}$) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2D Trench | 37.8M | 18 | 10 | 15 | 19 | 10 | 15 | **37** | 20 | **30** | **1.2** | **7.6** | **87.5** | 2.7 |
| 2D Trench | 2.4M | 0.9 | 0.9 | 1.4 | 1 | 0.7 | 1.1 | **1.9** | 1.6 | **2.5** | **0.8** | 0.0062 | 2.5 | 1.3 |
| 3D Trench | 38.4M | 25 | 11.9 | 17.9 | 23 | 10.6 | 15.9 | **48** | 22.5 | **33.8** | **1.4** | 0.0046 | 25.7 | 5.5 |
| 3D Trench | 4.8M | 2.4 | 0.8 | 1.2 | 2.2 | 0.8 | 1.2 | **4.6** | 1.6 | **2.4** | **1.9** | 0.012 | 3.4 | 4.9 |
| Multiple Lines | 47.6M | 44 | 27.2 | 40.8 | 24 | 9.5 | 14.3 | **68** | 36.7 | **55.1** | **1.2** | 1 | 31.6 | 3.8 |
| Multiple Lines | 4.8M | 3.1 | 1.7 | 2.6 | 1.7 | 0.8 | 1.2 | **4.8** | 2.5 | **3.8** | **1.3** | 0.034 | 5.9 | **6.6** |
| Filled Trench | 70M | 57 | 14.6 | 21.9 | 31 | 7.6 | 11.4 | **88** | 22.2 | **33.3** | **2.6** | 0.7 | 2.2 | 3.5 |
| Filled Trench | 15M | 10 | 3.2 | 4.8 | 6 | 1.6 | 2.4 | **16** | 4.8 | **7.2** | **2.2** | 5.3 | 72.6 | 5.6 |

For all but one test case our FIM implementation outperforms the reference FMM code. However, the error norms indicate a noticeable difference between the computed results

* Indicates a manual adjustment with a factor of 1.5 to model the overhead of the nested grid data structure used by the reference FMM implementation, which we approximate with 50 %

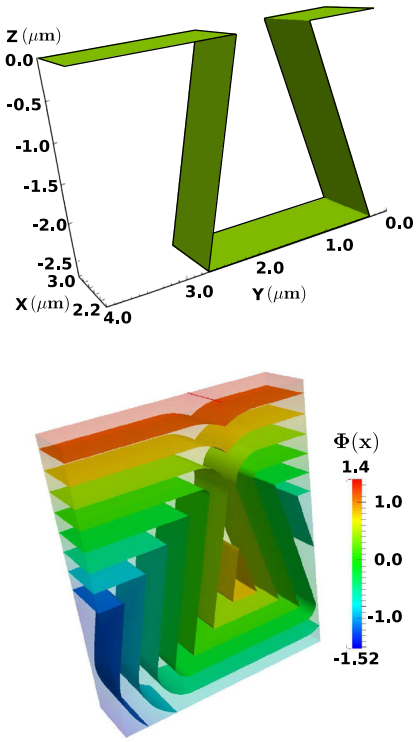


**Fig. 2** 2D TRENCH structure/signed distance field

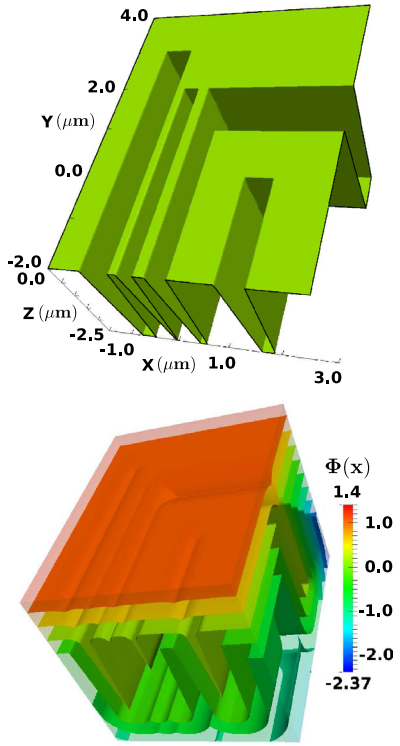


**Fig. 4** MULTIPLE LINES structure/signed distance field

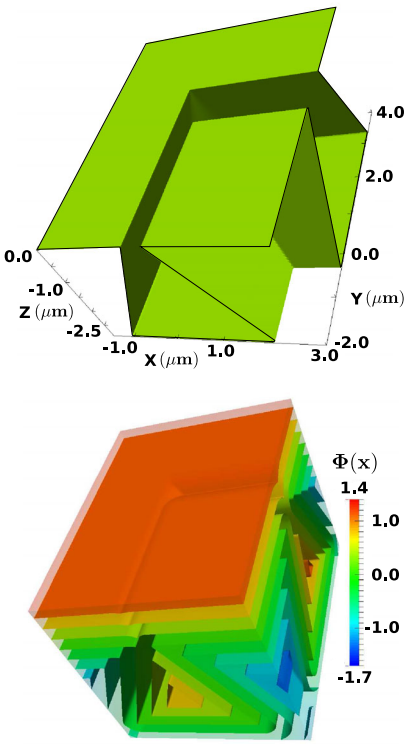


**Fig. 3** 3D TRENCH structure/signed distance field

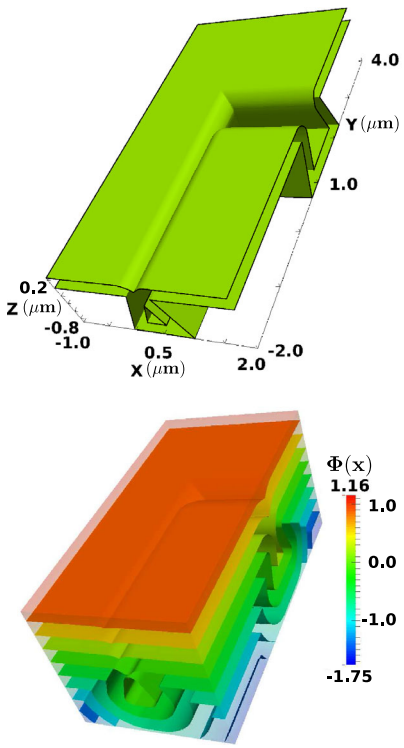


**Fig. 5** FILLED TRENCH structure/signed distance field

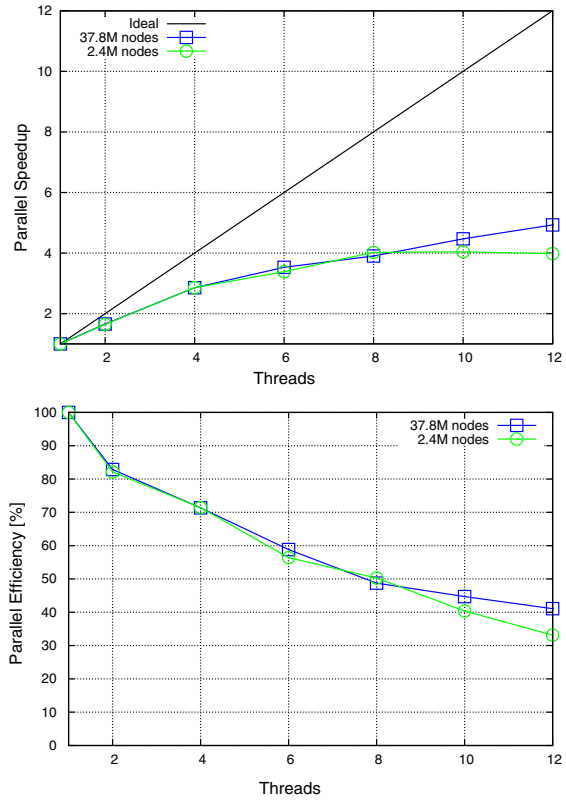


**Fig. 6** 2D TRENCH parallel scaling/efficiency

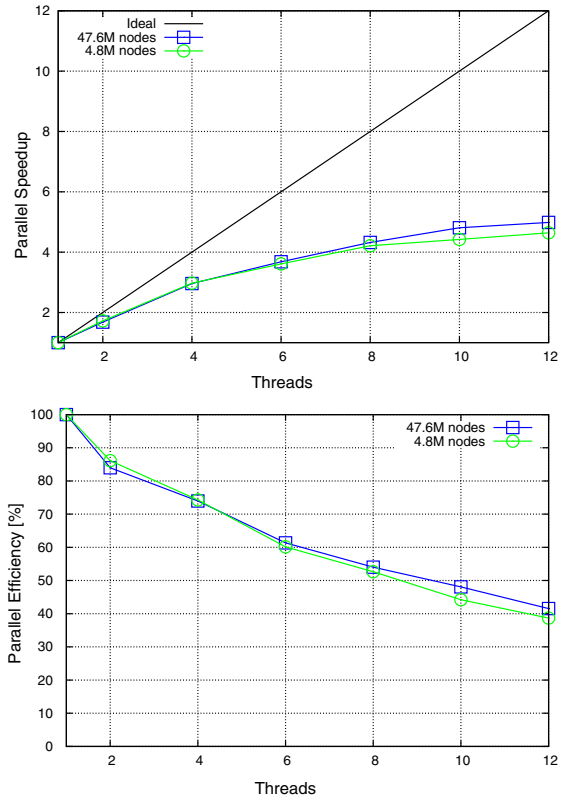


**Fig. 8** MULTIPLE LINES parallel scaling/efficiency

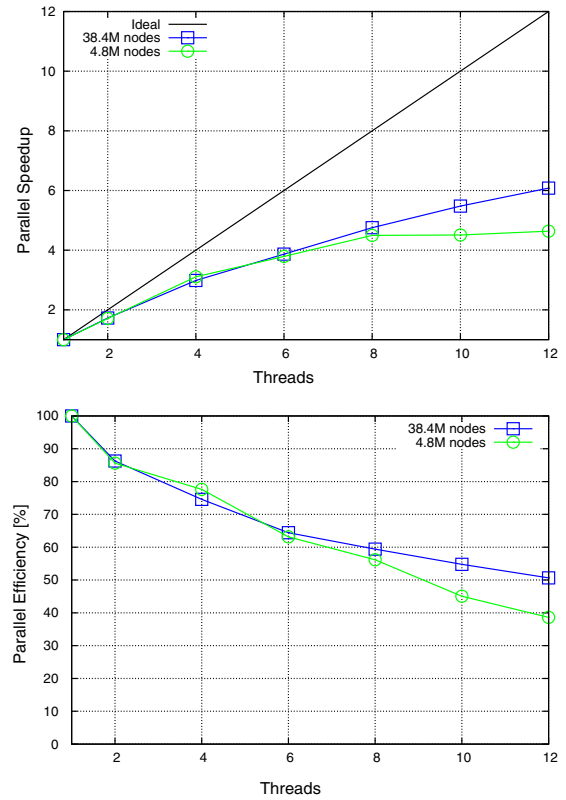


**Fig. 7** 3D TRENCH parallel scaling/efficiency

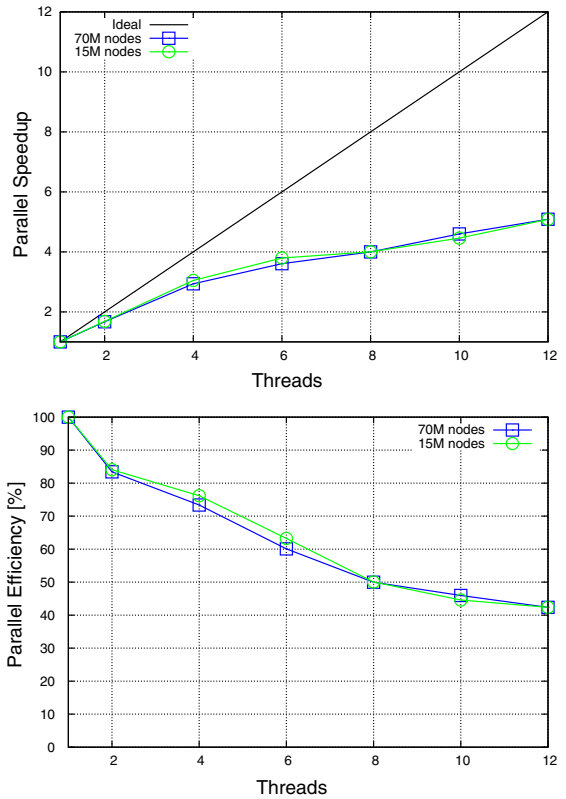


**Fig. 9** FILLED TRENCH parallel scaling/efficiency

GHz). The reference implementation is implemented on top of a nested grid data structure—offering flexibility albeit introducing run-time overhead—where the base mesh level is used. The reference implementation is tightly interwoven within an industry process simulator, thus we cannot directly compare the results. However, we estimate the run-time overhead of the nested grid data structure to be at most 50 %, therefore we manually adjust the execution-times of our standalone FIM application with a factor of 1.5 to allow for a reasonable comparison. For all but one test case our FIM implementation outperforms the reference FMM implementation, even for large problem sizes (speedup of 2.6 for the FILLED TRENCH structure with 70 M grid nodes). However, the error norms indicate noticeably differences in the computed results even for $\varepsilon < 10^{-10}$ ($\|\mathbf{u}\|_\infty = 6.6$ for the MULTIPLE LINES structure with 4.8M grid nodes). We attribute this to the utilized difference operator, which is proposed in the original work of the FSM [25] and the FIM [8], respectively.

In Figs. 6, 7, 8, 9 we present the parallel scalability of our OpenMP-parallelized FIM implementation on a 64-bit Linux cluster node, powered by an AMD Opteron 6348 (12 cores, 2.8 GHz). Increasing the problem size improves parallel scalability for larger numbers of threads, being particular important for the ever-increasing drive towards simulating high-resolution problems. For twelve threads we get 40 % efficiency for most of the problems.

For the important thread range between four and eight threads (most of today's single-user workstations offer four- to eight-cores), the efficiency is between 50 and 80 %, which is fairly reasonable considering the fact that we investigate three-dimensional real-world structures.

However, it is interesting to note that the parallel scalability is not only influenced by the problem size, but also by the complexity of the given problem. The intricate high-resolution FILLED TRENCH structure offers 70 M nodes and a parallel efficiency of 40 % for twelve threads as compared to the high-resolution mesh of the 3D TRENCH structure which offers 38.4M nodes and a parallel efficiency of 50 %. Comparing the input structures reveals that the FILLED TRENCH structure offers two surface areas, requiring the FIM more iterations to converge, which also impedes parallel efficiency due to an increased number of operations on shared-memory.

## 5 Conclusion

We compared an OpenMP-parallelized implementation of the FIM with an industry-hardened implementation of the FMM in the context of TCAD process simulation. Our extensions relative to the already available approaches have been outlined and our general algorithm has been discussed. Four process simulation problem structures, each offering two different grid resolutions, have been used to evaluate the performance and the quality of our implementation. Our FIM implementation offers excellent serial execution performance, however, the computed signed distance fields differ noticeably from the reference results ($\|\mathbf{u}\|_\infty = 6.6$ for the MULTIPLE LINES structure with 4.8 M grid nodes). Parallel execution behavior is good for four to eight threads, and reasonable for twelve threads and large problem sizes (i.e. $\geq$5 M grid nodes). Overall, we conclude that despite the identified accuracy shortcomings, our FIM implementation is an excellent approach for increasing the performance of LS-based process simulations in the area of TCAD by leveraging the parallel execution potential of today's multi-core workstations.

Future work will aim for increasing the accuracy of our FIM implementation by investigating higher-order difference schemes. Different parallelization approaches will be analyzed to further improve parallel efficiency. The applicability of using the FSM as well as an accelerator-based implementation of the FIM for the field of process TCAD will be investigated.

**Acknowledgments** This work has been supported by the Austrian Science Fund (FWF) through the Grant P23296. The authors thank Florian Dang from the Université de Versailles, France for valuable discussions concerning the FIM.

## References

1. Suvorov, V., Hössinger, A., Djurić, Z., Ljepojevic, N.: A novel approach to three-dimensional semiconductor process simulation: application to thermal oxidation. J. Comput. Electron **5**(4), 291 (2006). doi:10.1007/s10825-006-0003-z
2. Osher, S., Sethian, J.A.: Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. J. Comput. Phys. **79**(1), 12 (1988). doi:10.1016/0021-9991(88)90002-2
3. Ertl, O., Selberherr, S.: A fast level set framework for large three-dimensional topography simulations. Comput. Phys. Commun. **180**(8), 1242 (2009). doi:10.1016/j.cpc.2009.02.002
4. Radjenović, B., Radmilović-Radjenović, M.: 3D simulations of the profile evolution during anisotropic wet etching of silicon. Thin Solid Films **517**(14), 4233 (2009). doi:10.1016/j.tsf.2009.02.007
5. Filipovic, L., Selberherr, S.: A method for simulating atomic force microscope nanolithography in the level set framework. Microelectron. Eng. **107**, 23 (2013). doi:10.1016/j.mee.2013.02.083
6. Montoliu, C., Ferrando, N., Gosálvez, M., Cerdá, J., Colom, R.: Level set implementation for the simulation of anisotropic etching. J. Micromech. Microeng. **23**(7), 075017 (2013). doi:10.1088/0960-1317/23/7/075017
7. Silvaco.: Victory Process (2014). http://www.silvaco.com/products/tcad/process_simulation/victory_process/victory_process.html
8. Jeong, W.K., Whitaker, R.T.: A fast iterative method for Eikonal equations. SIAM J. Sci. Comput. **30**(5), 2512 (2008). doi:10.1137/060670298
9. Adalsteinsson, D., Sethian, J.: The fast construction of extension velocities in level set methods. J. Comput. Phys. **148**(1), 2 (1999). doi:10.1006/jcph.1998.6090

10. Mauch, S.: A fast algorithm for computing the closest point and distance transform. Tech. Rep. 077. California Institute of Technology (2000)
11. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proc. Natl Acad. Sci. **93**(4), 1591 (1996)
12. Sethian, J.A., Vladimirsky, A.: Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes. Proc. Natl Acad. Sci. **97**(11), 5699 (2000). doi:10.1073/pnas.090060097
13. Zhu, Y.: Adaptively Refined Meshes for Level Set Function. University of British Columbia, Tech. rep (2004)
14. Dijkstra, E.: A note on two problems in connexion with graphs. Numer. Math. **1**, 269 (1959)
15. Sethian, J.A.: Evolution, level set and fast marching methods for advancing fronts. J. Comput. Phys. **169**(2), 503 (2001). doi:10.1006/jcph.2000.6657
16. Herrmann, M.: A domain decomposition parallelization of the fast marching method, annual research briefs. Center for Turbulence Research, pp. 213–225 (2003)
17. Tugurlan, M.C.: Fast Marching Methods—Parallel Implementation and Analysis. Ph.D. thesis, Louisiana State University (2008)
18. Breuß, M., Cristiani, E., Gwosdek, P., Vogel, O.: An adaptive domain-decomposition technique for parallelization of the fast marching method. Appl. Math. Comput. **218**(1), 32 (2011). doi:10.1016/j.amc.2011.05.041
19. Gunnarsson, J.: Algorithms for representation of 3d regions in radiotherapy planning software. Master's thesis, Uppsala Universitet (2013)
20. Li, S., Mueller, K., Jackowski, M., Dione, D., Staib, L.: Physical-space refraction-corrected transmission ultrasound computed tomography made computationally practical. Lect. Notes Comput. Sci. **5242**, 280–288 (2008). doi:10.1007/978-3-540-85990-1_34
21. Fu, Z., Jeong, W.K., Pan, Y., Kirby, R., Whitaker, R.T.: A fast iterative method for solving the Eikonal equation on triangulated surfaces. SIAM J. Sci. Comput. **33**(5), 2468 (2011). doi:10.1137/100788951
22. Jeong, W.K., Whitaker, R.T.: A fast Eikonal equation solver for parallel systems. In Proceedings of the SIAM Conference on Computational Science & Engineering (CSE) (2007)
23. Dang, F., Emad, N., Fender, A.: A fine-grained parallel model for the fast iterative method in solving eikonal equations. In Proceedings of the International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC) (2013), pp. 152–157. DOI:10.1109/3PGCIC.2013.29
24. Dang, F., Emad, N.: Multi-level parallel upwind finite difference scheme for anisotropic front propagation. In Proceedings of the International Meeting on High Performance Computing for Computational Science (VECPAR) (2014)
25. Zhao, H.: A fast sweeping method for Eikonal equations. Math. Comput **74**(250), 603 (2005). doi:10.1090/S0025-5718-04-01678-3
26. Detrixhe, M., Gibou, F., Min, C.: A parallel fast sweeping method for the Eikonal equation. J. Comput. Phys. **237**, 46 (2013). doi:10.1016/j.jcp.2012.11.042
27. Hysing, S.R., Turek, S.: The Eikonal equation: numerical efficiency vs. algorithmic complexity on quadrilateral grids. In: Proceedings of Algoritmy, pp. 22–31 (2005)
28. Sussman, M., Fatemi, E.: An efficient. Interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. SIAM J. Sci. Comput. **20**(4), 1165 (1999). doi:10.1137/S1064827596298245
29. Ausas, R.F., Dari, E.A., Buscaglia, G.C.: A geometric mass-preserving redistancing scheme for the level set function. Int. J. Numer. Methods Fluids **65**(8), 989 (2011). doi:10.1002/fld.2227
30. Losasso, F., Fedkiw, R., Osher, S.: Spatially adaptive techniques for level set methods and incompressible flow. Comput. Fluids **35**(10), 995 (2006). doi:10.1016/j.compfluid.2005.01.006