Parallelized Level-Set Velocity Extension Algorithm for Nanopatterning Applications

Michael Quell
Christian Doppler Laboratory for High Performance TCAD
Institute for Microelectronics, TU Wien
Wien, Austria
quell@iue.tuwien.ac.at

Alexander Toifl
Christian Doppler Laboratory for High Performance TCAD
Institute for Microelectronics, TU Wien
Wien, Austria
toifl@iue.tuwien.ac.at

Andreas Hössinger
Silvaco Europe Ltd.
Cambridge, United Kingdom
andreas.hoessinger@silvaco.com

Siegfried Selberherr
Institute for Microelectronics, TU Wien
Wien, Austria
selberherr@tuwien.ac.at

Josef Weinbub
Christian Doppler Laboratory for High Performance TCAD
Institute for Microelectronics,TU Wien
Wien, Austria
weinbub@iue.tuwien.ac.at

***Abstract*—We present a parallelized algorithm for accelerating the velocity extension calculations in a level-set method, which is essential for surface velocity based topography simulations, such as etching or deposition simulations for nanopatterning applications. The proposed algorithm improves the prevailing fast marching method by optimizing the heap data structure and efficiently reordering the calculations. We implemented the algorithm into Silvaco's Victory Process simulator, which is utilized for evaluating our algorithm with a three-dimensional simulation of an ion beam etching process used for spin-transfer torque magnetoresistive random access memory devices. Our results show a significant serial speed-up by a factor of at least 1.4 and a total speed-up by a factor of up to 8 using 8 threads for the velocity extension.**

I. Introduction

Modern semiconductor devices are progressively employing non-planar geometries, e.g., high-aspect ratio pillars, cavities or fins. The thus necessary fabrication processes are based on complex patterning techniques on the nanoscale, essentially requiring high control over geometry parameters. For example, the devices proposed in the field of emerging memory techologies [1] particularly demand optimized nanopatterning to enable a small feature size and high density memory cells in order to replace conventional complementary metal-oxide semiconductor (CMOS)-based random access memory (RAM) [2], [3].

Process simulations employing the level-set method [4], [5], [6] enable highly robust and accurate simulations of the temporal evolution of the wafer surface during such fabrication processes of semiconductor devices. The level-set method naturally handles topological changes in three dimensions and allows for the simulation of a variety of processing techniques via a general velocity function.

A level-set topography simulation involves several steps which are summarized in Fig. 1(a). Starting with the construction of the level-set function from the initial geometry, the surface (front) velocity has to be calculated according to a process-specific physical model. Subsequently, the surface velocity is extended to the computational domain (velocity extension), which allows to move the surface according to the velocity by employing a time integration scheme for the level-set field. After the time stepping the level-set field is converted to an explicit mesh, which enables the visualization of the resulting surface.

The velocity extension step, which has to be performed in every time step, significantly contributes to the overall runtime. In this work we present a shared-memory parallelized velocity extension algorithm which considerably reduces the simulation runtime. The proposed algorithm was implemented into Silvaco's Victory Process simulator [7], which we utilize here to test our algorithm by simulating an ion beam etching (IBE) step required for the production of spin-transfer torque magnetoresistive RAM (STT-MRAM) devices [8].

II. Velocity Extension

The level-set method implicitly represents the wafer surface Γ as the zero level-set of the function $\phi(\vec{x}, t)$ and the time evolution is determined by the level-set equation

$$\frac{\partial \phi}{\partial t} + V|\nabla \phi| = 0. \tag{1}$$

V is a velocity field which is defined on the entire computational domain, modeling the motion of the surface. However, most process-specific physical models only provide the surface velocity V_{surface} on Γ. In order to solve (1), V_{surface} has to be extended to the entire computational domain. We employ

978-1-7281-0940-4/19/$31.00

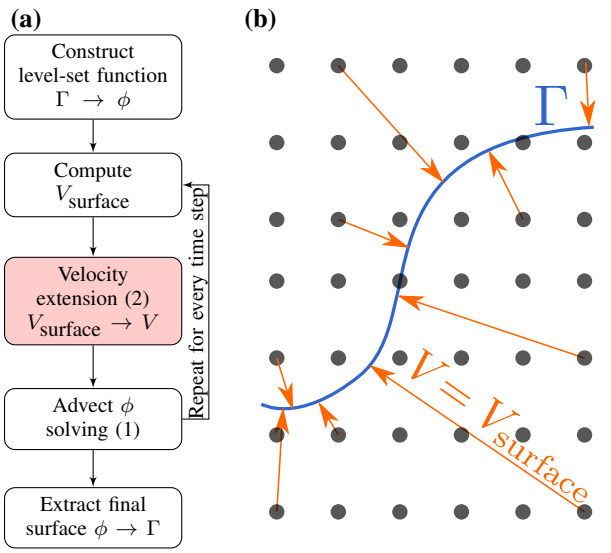

Fig. 1: **(a)** Flow diagram of the computational tasks in a level-set simulation. **(b)** Velocity extension according to (2) yields for every grid point the velocity of the closest surface point. Thus the velocity along the orange arrows is constant.

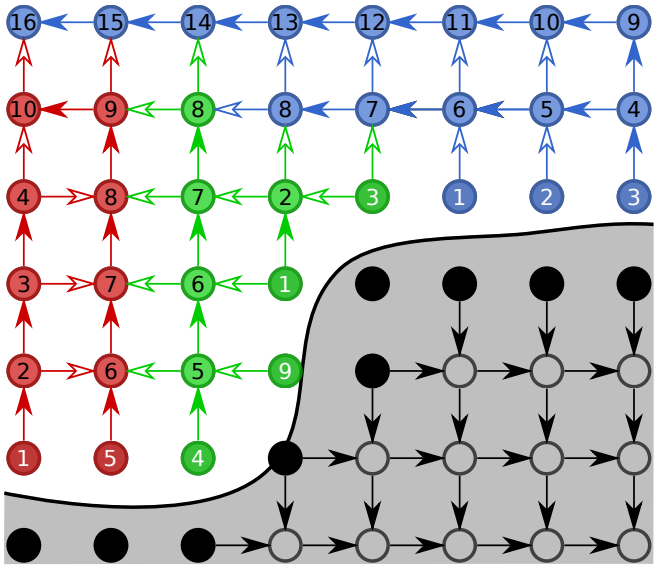

Fig. 2: Processing order of the grid points for one side of the surface by the proposed algorithm employing three threads (red, green, and blue). Grid points with a white number next to the interface are the grid points which constitute the *runs*. Filled arrows show successful updates of the neighbor, while bare arrows show that the neighboring grid point either had an unprocessed upwind neighbor or has been already processed by another thread.

the signed-distance property preserving approach, which is characterized by constant velocity values along the surface normals [9]. This is shown for selected grid points in Fig. 1(b). The associated partial differential equation

$$\nabla\phi \cdot \nabla V = 0, \text{ and } V\big|_{\Gamma} = V_{\text{surface}} \tag{2}$$

is typically solved with the fast marching method (FMM) [4]. As a prerequisite for the FMM the velocity is set on the grid points in direct proximity to the surface [9]. The surface divides the set of grid points into two subsets, allowing for the subsequent independent application of the FMM for both sides.

The FMM processes the grid points in ascending order by their level-set value, enabling the usage of an upwind difference scheme to approximate the gradients in (2). Thus, only information from grid points which are closer to the surface is utilized, resulting in a one pass algorithm. In particular, the processing of a grid point consists of two tasks: (a) calculate the velocity and (b) insert all unprocessed neighbors into the minimum heap. To track the processing order of the grid points, a global minimum heap data structure sorted by the level-set values is used. Consequently, the FMM is an inherently serial algorithm.

III. Parallelized Velocity Extension

Our proposed algorithm overcomes the limitation of the FMM by executing a modified FMM for every grid point in direct proximity to the surface. We refer to the execution of the modified FMM based on one of those grid points and the corresponding minimum heap (from the modified FMM) as a *run*. The *runs* are processed one after another, which effectively reorders the calculations and yields a speed-up by reducing the effective heap sizes.

The introduction of the non-global minimum heaps creates cases in which grid points have unprocessed upwind neighbors. These cases need special treatment: We remove grid points which have an unprocessed upwind neighbor from the current minimum heap without processing it. Nevertheless, it is ensured that all grid points are processed, because once the formerly unprocessed upwind neighbor is processed (by a different *run*) the removed grid point is visited again, allowing for the application of the upwind scheme.

In summary, some of the grid points are visited several times by our algorithm (depending on the number of upwind neighbors). However, the velocity is calculated only once, using the same upwind neighbors as in the FMM. Consequently, we ensure that calculated extended velocities are identical compared to the FMM.

The *runs* enable an efficient approach to parallelize the velocity extension algorithm, because they are dynamically assigned to different threads in parallel. Once the currently assigned *run* is completed (i.e., its minimum heap is empty) a new unprocessed *run* is assigned. The dynamic assignment of the *runs* to the threads is necessary as the associated computational load varies strongly between them. Fig. 2 shows the processing order of the grid points for an exemplary surface topology. One side of the surface is processed by three threads.

Each thread is assigned a *run* (in Fig. 2, a grid point with a white number) and creates its own minimum heap containing

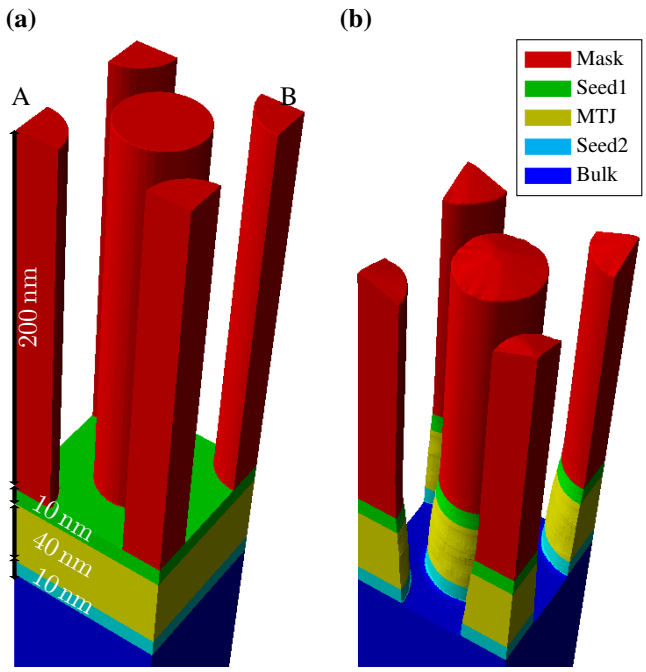

Fig. 3: Initial wafer topography consisting of the magnetic tunnel junction (MTJ) and the associated seed layers deposited on the substrate: **(a)** prior to the IBE process and **(b)** final pillar topography for the STT-MRAM device.

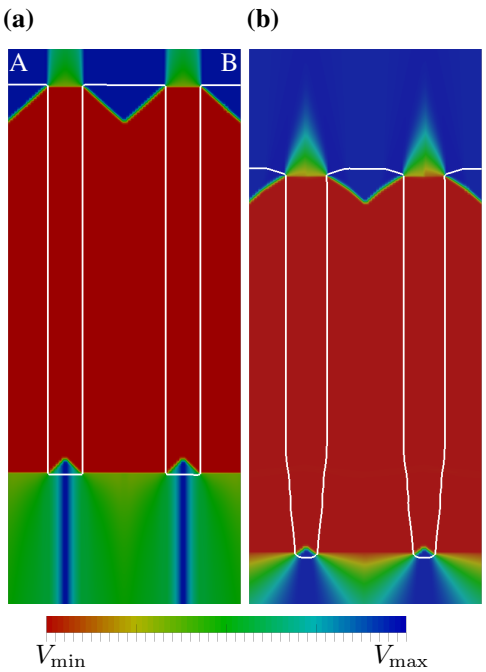

Fig. 4: Extended velocity field in the plane containing points A and B in Fig. 3 **(a)** prior to and **(b)** after the IBE process. The velocity is constant along the surface normals as imposed by (2). The white line depicts the wafer surface.

only this grid point. Then the thread processes its minimum heap analogously to the traditional FMM implementation with the previously described modification, while taking care of unprocessed upstream grid points. Once the minimum heap is empty, a new *run* is assigned. The blue thread in Fig. 2 demonstrates the varying load: The first and second *run*, (blue grid point 1 and 2) only processes a single grid point, while the third *run* (blue grid point 3) processes 14 grid points in total.

The proposed algorithm does not use explicit synchronization during the processing of the *runs*. This is a trade-off between the cost of the explicit synchronization and avoiding redundant operations, i.e., computing the velocity of a grid point more than once. It is not an issue if the velocity for a grid point is computed more than once, because the algorithm always enforces the usage of the same upwind neighbors for all threads. Thus, for any grid point the computational result is the same for every thread. The redundant operations may occur along the border of two regions processed by different threads (cf. bare arrows pointing to a differently colored grid point in Fig. 2). Nevertheless, the accuracy of the final extended velocity field does not depend on the choice between explicit and not explicit synchronization, as both threads use the same upwind neighbors.

IV. Results and Discussion

We employ the proposed algorithm for simulating an IBE process which is essential for the fabrication of STT-MRAM devices. The considered structure consists of a magnetic tunnel junction (MTJ) and the associated seed layers which have been deposited on a substrate. IBE allows to fabricate an array of MTJ pillars (shown in Fig. 3a and Fig. 3b). We apply the proposed velocity extension algorithm to the IBE step. The extended velocity fields for the initial and final surface are shown in Fig. 4(a) and Fig. 4(b) for a vertical plane slicing the computational domain diagonally (containing Points A and B in Fig. 3(a). The velocity is constant along the surface normals. Furthermore, the corners, where the velocity abruptly changes and consequently the solution is discontinuous, are well resolved.

We assessed the implementation of the algorithm (for implementation details see [11]) on a compute node of the Vienna Scientific Cluster 3 (two Intel Xeon E5-2650v2 Ivy Bridge EP processors, 64 GB main memory) [10], by comparing the runtimes for a single velocity extension step. The code uses C++11 and was compiled with gcc-7.3 with `-O3` optimization.

Fig. 5 shows that the serial runtime for the velocity extension is reduced by a factor of 1.4 (1.5) for a grid resolution of $2\,\mathrm{nm}$ ($0.5\,\mathrm{nm}$). The runtime reduction is mainly caused by the splitting of the global heap into smaller heaps, resulting in less time-consuming sorting operations. The cost of visiting grid points twice is negligible, because the velocity is calculated only once.

Furthermore, we evaluated the parallelized algorithm (implemented with OpenMP 4.5), which has a parallel efficiency of approximately $60\,\%$ ($66\,\%$) for 8 threads and thus results

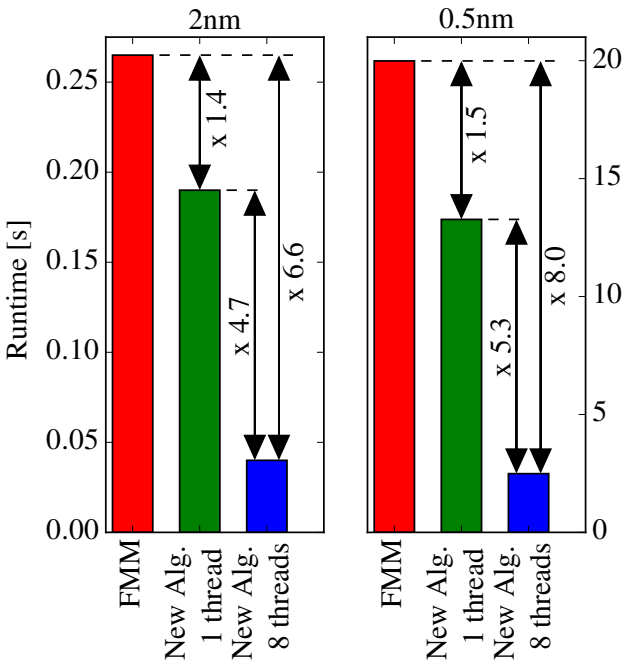

Fig. 5: Runtime of the velocity extension for a single time-step measured on a compute node of the Vienna Scientific Cluster 3. The proposed algorithm is compared to the original algorithm (FMM) for a spatial resolution of $2\,\mathrm{nm}$ and $0.5\,\mathrm{nm}$.

in a parallel speed-up of 4.7 (5.3). Each of the 8 threads is assigned to a separate core. In our implementation, the grid points are accessed using OpenMP atomic operations, which is required to enforce a consistent view of the memory between the threads.

Avoiding explicit synchronization leads to less than $1\,\%$ ($0.1\,\%$) of the grid points processed twice (see Section III). The lower percentage of the redundantly processed grid points in the higher resolution simulation (i.e. the $0.5\,\mathrm{nm}$ case) is caused by a square-cube law of grid points on the borders between threads and total processed grid points. The grid points on the borders between threads scale by a power of two and the total processed grid points by a power of three, therefore, their ratio decreases with increasing size.

The serial and parallel speed-up combined yield a total runtime reduction of the velocity extension step by a factor of 6.6 (8.0) for 8 threads, which demonstrates the excellent performance of our approach. Our algorithm results in the same velocity field as the reference FMM implementation, which we confirm by the L^∞-Norm (maximum over the velocity value differences for all grid points) yielding 0.

V. Summary

We have presented a parallelized velocity extension algorithm improving on the serial fast marching method, by efficiently reordering the calculations. This is achieved by replacing the global heap data structure associated with the fast marching method with many non-global minimum heaps (e.g., unprocessed upwind neighbors) The efficient velocity algorithm enables level-set topography simulation of complex three-dimensional non-planar geometries as they commonly appear in nanopatterning applications. We demonstrate the capability of the proposed algorithm with the velocity extension step in an ion beam etching process of a spin-transfer torque magnetoresistive memory device. Our implementation does not show any difference in the resulting velocity field compared to the reference fast marching method. Furthermore, we have investigated the serial and parallel speed-up, where the combined speed-up is in the range from 6.6 to 8.0 for 8 threads.

Acknowledgment

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 computational results presented have been achieved using the Vienna Scientific Cluster (VSC).

References

[1] S. Bhatti, R. Sbiaa, A. Hirohata, H. Ohno, S. Fukami, and S. N. Piramanayagam, "Spintronics Based Random Access Memory: A Review," *Materials Today*, vol. 20, no. 9, pp. 530–548, 2017.

[2] D. Apalkov, B. Dieny, and J. M. Slaughter, "Magnetoresistive Random Access Memory," *Proceedings of the IEEE*, vol. 104, no. 10, pp. 1796–1830, 2016.

[3] V. D. Nguyen, P. Sabon, J. Chatterjee, L. Tille, P. V. Coelho, S. Auffret, R. Sousa, L. Prejbeanu, E. Gautier, L. Vila, and B. Dieny, "Novel Approach for Nano-Patterning Magnetic Tunnel Junctions Stacks at Narrow Pitch: A Route Towards High Density STT-MRAM Applications," in *IEEE International Electron Devices Meeting (IEDM)* , pp. 38.5.1–38.5.4, 2017.

[4] J. A. Sethian, "A Fast Marching Level Set Method for Monotonically Advancing Fronts." *Proceedings of the National Academy of Sciences*, vol. 93, pp. 1591–1595, 1996.

[5] J. A. Sethian, "Evolution, Implementation, and Application of Level Set and Fast Marching Methods for Advancing Fronts," *Journal of Computational Physics*, vol. 169, no. 2, pp. 503–555, 2001.

[6] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, 2011.

[7] "Silvaco Victory Process," https://www.silvaco.com/products/tcad.html, (accessed March 27, 2019).

[8] M. Gajek, J. J. Nowak, J. Z. Sun, P. L. Trouilloud, E. J. OSullivan, D. W. Abraham, M. C. Gaidis, G. Hu, S. Brown, Y. Zhu, R. P. Robertazzi, W. J. Gallagher, and D. C. Worledge, "Spin Torque Switching of 20nm Magnetic Tunnel Junctions with Perpendicular Anisotropy," *Applied Physics Letters*, vol. 100, no. 13, p. 132408, 2012.

[9] D. Adalsteinsson and J. A. Sethian, "The Fast Construction of Extension Velocities in Level Set Methods," *Journal of Computational Physics*, vol. 148, no. 1, pp. 2–22, 1999.

[10] "Vienna Scientific Cluster 3," http://vsc.ac.at/systems/vsc-3/, (accessed June 18, 2019).

[11] M. Quell, P. Manstetten, A. Hössinger, S. Selberherr, and J. Weinbub, "Parallelized Construction of Extension Velocities for the Level-Set Method," in *International Conference on Parallel Processing and Applied Mathematics (PPAM)*, Springer LNCS, 2019.