Implementation and evaluation of the Level Set method: Towards efficient and accurate simulation of wet etching for microengineering applications

C. Montoliu, N. Ferrando, M. A. Gosálvez, J. Cerdá, R. J. Colom

Abstract

The use of atomistic methods, such as the Continuous Cellular Automaton (CCA), is currently regarded as a computationally efficient and experimentally accurate approach for the simulation of anisotropic etching of various substrates in the manufacture of Micro-electro-mechanical Systems (MEMS). However, when the features of the chemical process are modified, a time-consuming calibration process needs to be used to transform the new macroscopic etch rates into a corresponding set of atomistic rates. Furthermore, changing the substrate requires a labor-intensive effort to reclassify most atomistic neighborhoods. In this context, the Level Set (LS) method provides an alternative approach where the macroscopic forces affecting the front evolution are directly applied at the discrete level, thus avoiding the need for reclassification and/or calibration. Correspondingly, we present a fully- operational Sparse Field Method (SFM) implementation of the LS approach, discussing in detail the algorithm and providing a thorough characterization of the computational cost and simulation accuracy, including a comparison to the performance by the most recent CCA model. We conclude that the SFM implementation achieves similar accuracy as the CCA method with less fluctuations in the etch front and requiring roughly 4 times less memory. Although SFM can be up to 2 times slower than CCA for the simulation of anisotropic etchants, it can also be up to 10 times faster than CCA for isotropic etchants. In addition, we present a parallel, GPU-based implementation (gSFM) and compare it to an optimized, multi-core CPU version (cSFM), demonstrating that the SFM algorithm can be successfully parallelized and the simulation times consequently reduced, while keeping the accuracy of the simulations. Although modern multicore CPUs provide an acceptable option, the massively parallel architecture of modern GPUs is more suitable, as reflected by computational times for gSFM up to 7.4 times faster than for cSFM.

Keywords: Level Set Method, Sparse Field Method, Anisotropic wet chemical etching, Microengineering, Cellular Automata, Micro-electro-mechanical systems, MEMS, KOH, TMAH, Parallel computing, GPU, CUDA

1. Introduction

Anisotropic wet chemical etching is one of the most popular bulk micromachining methods for the fabrication of Micro-Electro-Mechanical Systems (MEMS). This process is technologically relevant due to the low cost and ability to generate smooth/flat surfaces and release suspended structures. Nevertheless, the resulting three-dimensional shapes are difficult to predict due to multiple dependencies, such as the crystal orientation of the surface [1, 2, 3, 4, 5, 6], the composition of the applied etchant —e.g. potassium hydroxide (KOH) [7, 8], tetramethylammonium hydroxide (TMAH) [9, 10] or various other solutions [11]— the etchant concentration [4, 6], the temperature [2, 4, 6] or the use of additives, such as Triton X-100 [12, 13] or isopropyl alcohol [4, 8, 14]. Accordingly, an important effort has been dedicated through the years to model accurately the process.

The first simulators were based on geometrical models focused on describing the etching front as a set of moving facets [15, 16]. This posed various challenges, such as the efficient implementation of the intersection of two or more planes in 3D and, correspondingly, the description of wafer perforation. On the other hand, the use of atomistic approaches, such as Cellular Automata (CA) [17, 18, 19, 20, 21, 22, 23, 24] and Kinetic Monte Carlo (KMC) [25, 26, 27, 28], became gradually successful. By treating the evolving surface as a collection of atomistic sites within a lattice, the surface atoms are progressively removed according to simple microscopic rules, thus resulting in the emergence of the neighbor sites into the surface. This enables a natural description of topological changes in the front (e.g. coalescing/splitting of contiguous regions) without any further computational/programming effort. While KMC has been traditionally used to model the surface roughness on different substrate orientations [26, 27], there exist successful examples of its use for microstructure prediction [29, 28]. In turn, although CA is best suited for modeling the formation of 3D microstructures, it has also been used to describe a wide range of surface morphologies [30].

The first CA models were based on stochastic approaches
[17], eventually giving way to various deterministic procedures [18, 19, 20]. By taking into account several physical phenomena, such as the step flow nature of wet etching [5, 22], the Continuous CA (CCA) method has demonstrated to be the most accurate, simulating reliably wet etching for a wide range of MEMS structures and conditions [21, 31]. In addition, algorithmic accelerations for sequential and parallel computational environments have been presented [23, 24] and commercial simulators, such as IntelliEtch [32], are able to execute the CCA calculations on Graphics Processing Units (GPUs), drastically reducing the computation times to just a few seconds per simulation. Another advantage over the classical geometrical methods is the possibility of calibrating the atomistic model parameters by using a reduced dataset of experimental etch rates [33]. Originally requiring careful supervision by the user, the calibration process has been recently automated by using an evolutionary algorithm for both KMC [28] and CCA [6, 34]. Despite the good results, new calibrations are needed each time the experimental conditions are changed, requiring several hours or even days for calibration completion. In this context, the search of an alternative method, which is simultaneously capable of using the experimental data without prior calibration while remaining computationally-efficient, has the potential to influence significantly the future of MEMS design.

In this study we consider the use of the Level Set (LS) method, introduced in the 80's for capturing moving fronts [35] and developed further for related problems, such as image reconstruction [36], image segmentation [37], moving liquid interfaces [38, 39] as well as chemical etching [40, 41, 42, 43] and Deep Reactive Ion Etching (DRIE)/plasma etching [44, 45]. While LS has been previously used to simulate anisotropic etching [46, 47, 48, 49], the computational efficiency and relative accuracy in comparison to the existing methods have not been addressed before. In addition, there is some freedom to perform the LS calculations over the whole domain or just in the vicinity of the evolving front [50]. Thus, the computational efficiency and relative accuracy of different approaches need to be compared to the benchmark results by the CCA method.

Based on the Sparse Field Method (SFM) [50], which concentrates the processing effort on the active points around the evolving front, we present an efficient sequential implementation of the LS method and compare it to an equivalent sequential implementation of the CCA method. This way, we evaluate the viability of LS as a simulation tool for MEMS design in practice. In addition, two parallel versions of the SFM are developed: (i) cSFM, which takes advantage of the multicore nature of modern Central Processing Units (CPUs) and the Single Instruction Multiple Data (SIMD) functionality, enabling the computation of the same instruction over multiple data in a single clock cycle; and (ii) gSFM, which benefits from the massively parallel architecture of modern Graphics Processing Units (GPUs), enabling the simultaneous processing of many thousands of threads and, thus, reducing drastically the computational time. Both cSFM and gSFM implementations are compared in terms of their computational efficiency.

The rest of the paper is organized as follows. The LS method and the numerical techniques needed to adapt it to anisotropic etching are presented in Section 2, including the resulting algorithm in detail. This is followed by the results from several examples in Section 3, demonstrating the reliability of our LS simulations by direct comparison to experimental microphotographies. This also includes a comparison of the computational time, memory usage and relative accuracy between LS and CCA. Finally, the parallel cSFM and gSFM implementations are presented and compared in Section 4.

2. Level Set Method applied to anisotropic etching

The LS method was introduced by Osher and Sethian in order to track moving fronts [35]. The main idea consists in embedding the front, \( x(t) \), inside a signed distance function, \( \phi \), such that the distance right at the front at any time is zero by construction: \( \phi(x(t), t) = 0 \). This enables to apply to the function \( \phi \) itself the partial derivatives in time and space required to describe the propagation of the front. In the specific case where the front evolves in the normal direction, using the chain rule the propagation equation (or LS equation) is:

\[
\phi_t + R(x(t), t) |\nabla \phi| = 0 ,
\]

where \( R(x(t), t) \) is defined over the whole domain and denotes the velocity of a point in the normal direction, as determined by the physical laws of the process being simulated. The LS equation can also be written in Hamilton-Jacobi form:

\[
\phi_t + H(\nabla \phi(x(t), t)) = 0 ,
\]

where \( H = R(x(t), t) |\nabla \phi| \) is the Hamiltonian. For anisotropic etching, where the rate of propagation is a function of the local normal vector, the Hamiltonian depends on the geometry of the front (non-convex Hamiltonian), in which case there are several techniques to discretize Eq. 2, such as the Lax-Friedrichs (LF) [51], Roe-Fix [52] or Godunov [53] schemes. Based on central difference derivatives, LF is the simplest scheme that preserves monotonicity. Applied to Eq. 2, one obtains the following space-time discretization:

\[
\phi_t^{n+1} - \phi_t^n = \Delta t \left[ H(\beta_{x}^{n}, \beta_{y}^{n}, \beta_{z}^{n}) - \alpha_{x} (\alpha_{y} \beta_{y}^{n} + \alpha_{z} \beta_{z}^{n}) + \alpha_{y} (\alpha_{x} \beta_{x}^{n} + \alpha_{z} \beta_{z}^{n}) + \alpha_{z} (\alpha_{x} \beta_{x}^{n} + \alpha_{y} \beta_{y}^{n}) \right],
\]

\[
\beta_{x}^{n} = \frac{1}{2} \left[ (a_{i,j,k}^{n})^n \pm (a_{i,j,k}^{n})^n \right],
\]

\[
(a_{i,j,k}^{n})^n = \frac{\phi_{t}^{n} - \phi_{t}^{n-1}}{\Delta t},
\]

\[
\alpha_{i} = \max \left| \frac{\partial H}{\partial \phi} \right| ,
\]

where \( l = x, y, z \) and the operation \((i, j, k) \pm 1 \) only affects the index \( i, j \) or \( k \) corresponding to the selected dimension \( l \). Although \( \Delta t \) is the spatial resolution of the grid in a particular dimension, from now on the grid resolution will be considered to be the same in all dimensions and referred to as \( \Delta x \). Finally, the \( \alpha_{i} \) coefficients are the artificial viscosity factors inherent to the LF method and \( \alpha_{x} \) is an overall viscosity coefficient which we introduce to guarantee the stability of the front.
by modifying simultaneously the three \( \alpha_i \) values. Larger \( \alpha_F \)
results in smoother surfaces while smaller \( \alpha_F \) generates sharp
features/discontinuities, which should be handled carefully to
avoid unrealistic and/or diverging solutions. We have found
that \( \alpha_F = 0.48 \) allows the front to converge with every tested
etchant.

In order to apply the LF scheme, it is necessary to express
the viscosity factors of Eq. 6 in terms of \( R(x(t), t) \). In practice,
the experimental etch rates are constant in time, depending on
local the crystallographic orientation of the surface. Corre-
spondingly, the etch rates are organized as a 180x45 matrix,
\( R(\theta, \Phi) \), as a function of the azimuth (\( \Phi \)) and latitude (\( \theta \)) of
1810 normal vectors in spherical coordinates with a resolution
of 2 degrees [6]. Thus, applying the chain rule to Eq. 6 we
obtain:

\[
\alpha_l = \max \left| \frac{\partial R_l(\phi)}{\partial R} \Delta^2 \phi^l \right| + R(\theta, \Phi) \cdot N_l
\]

where \( N_l \) is the \( l \)-component of the unit normal vector with \( l, p \)
and \( q \) any of \( x, y \) and \( z \), such that \( p \neq l, q \neq l \) and \( p \neq q \).
Here, \( \frac{\partial R_l(\phi)}{\partial R} \Delta^2 \phi^l \) denotes the variation of the etch rate with respect
to the \( l \)-component of the normal vector, which is numerically
determined by discretizing each component with a resolution of
\( \Delta N_l = 0.01 \):

\[
\frac{\partial R_l(\phi)}{\partial R} = \frac{\Phi(N_3{\Delta N_l}N_p N_q) - \Phi(N_3 - \Delta N_l N_p N_q)}{2 \Delta N_l}
\]

It is important to notice that the cost to evaluate Eq. 3 has
four contributions: (i) the computation of the spatial derivatives
according to Eqs. 4 and 5, which involves the processing of
neighbor \( \phi \) values, (ii) the update of the Hamiltonian, which
requires determining the value of the etch rate corresponding
to the normal vector, (iii) the determination of the artificial viscos-
cities according to Eq. 7, which requires searching for the
maximum value over the whole domain, and (iv) the update of
the new \( \phi \) values. As described below, contribution (iii) repres-
sents the largest computational burden for the implementation
of the SFM method, taking about 55% of the total time required
to evaluate Eq. 3, while the other contributions represent: (i)
20-28% (ii) 11-13% and (iv) 7-12%.

2.1. Sparse Field Method

Due to the need to update the value of \( \phi \) at every point of
the grid, the LS method has a computational cost of \( O(N^3) \),
where \( N \) is the number of grid points in one dimension. In order
to reduce the cost Adalsteinsson et al. introduced the Narrow
Band (NB) method [54], which updates only the points located
within \( k \) layers counted from the front itself, thus reducing the
computational cost to \( O(kN^2) \). However, this approach is not
completely optimal because more points are still updated than
strictly necessary and, in addition, the signed distance function
needs to be rebuilt periodically every time the front reaches the
\( k \)-th layer. The Sparse Field Method (SFM) was introduced by
Whitaker [50] in order to reduce the active region to only the
strictly necessary points while efficiently updating the signed
distance in each time step [55].

Based on the use of backward and forward first order deriva-
tives, the SFM focuses on maintaining only 3 groups of points:

\[
\begin{align*}
\text{those at the front (L0), at distance } \Delta x \text{ from the front (Lp1), and at distance } -\Delta x \text{ (Ln1). The rest of the points} \\
\text{are not included in any list since they do not need to be up-
}\text{dated. The active points are labelled as 1, 0 or -1, depending on} \\
\text{whether they belong to Lp1, L0 or Ln1, respectively, while the}\text{rest of the points are labelled as 2 or -2, depending on whether} \\
\text{they are located below or above the front. If higher k-order} \\
\text{derivatives are used (e.g. to increase the accuracy), the points} \\
\text{at distance } \leq | \pm (k)\Delta x | \text{ from the front should be maintained} \\
\text{in similar lists. Fig. 1 shows a two-dimensional example of a} \\
\text{wet etching interface and the necessary lists. For each time} \\
\text{step, only the values of } \phi \text{ for the points in L0 are updated using} \\
\text{Eq. 3. For the points in Lp1 (Ln1), we search for the nearest} \\
\text{point belonging to L0 and add (subtract) } \Delta x \text{ to the found dis-
}\text{tance value. After propagating the front, the neighbouring points} \\
\text{that are not included in any list are added to the Lp1 or Ln1 list,} \\
\text{depending on the sign of their distance value. This is described} \\
\text{in detail in Section 2.2.}
\end{align*}
\]

2.2. Pseudocode: SFM implementation

The pseudocode of our SFM implementation is as follows:

1. Load the mask pattern and apply it to the substrate. This
simulates the lithographic process used to mask the wafer.
2. Choose the substrate type (silicon or quartz), crystallo-
graphic orientation, grid dimensions, etchant concentra-
tion, temperature and the etching time (\( t_{\text{max}} \)), in minutes).

\[
\text{The depth of the grid is calculated to ensure that the front} \\
\text{does not reach the bottom of the grid during } t_{\text{max}}:\n\]

\[
\text{Num}_c = 4 + t_{\text{max}} \cdot R(\theta = 0, \Phi = 0)
\]

where \( R(\theta = 0, \Phi = 0) \) is the vertical etch rate. To reduce
memory, dynamic allocation and an octree data structure
can be used [23].
3. The three-dimensional grid is built and the exposed surface
described.
4. Calculate the time step \( \Delta t \) according to the Courant-
Friedrichs-Levy (CFL) condition [56]. To avoid front diver-
gence we use: \( \Delta t = \frac{0.3 \Delta x}{\max(\partial R/\partial \Phi)} \).
5. Calculate the signed distance $\phi$ for the points located near the front and assign a constant value to all other points.
6. Add the points to the lists $L0$, $Lp1$ or $Ln1$, depending on the distance value.
7. Main loop:
   1) For every point in $L0$, determine:
      i. The forward and backward derivatives using Eq. 5.
      ii. The normal components $N_l = \phi_l/|\nabla \phi|$, where $l = x$, $y$ and $z$.
      iii. The etch rate (convert the cartesian components to spherical coordinates and access the corresponding matrix element).
      iv. The maximum value according to Eq. 7.
   2) Define temporary lists $S0$, $Sp1$ and $Sn1$.
   3) For every point $(i, j, k)$ in $L0$:
      • Update $\phi$ using Eq. 3.
      • If $\phi_{i,j,k}^{n+1} < -0.5\Delta x$: remove $(i, j, k)$ from $L0$; add it to $Sn1$.
      • If $\phi_{i,j,k}^{n+1} > 0.5\Delta x$: remove $(i, j, k)$ from $L0$; add it to $Sp1$.
   4) For every point $(i, j, k)$ in $Lp1$:
      i. Find the neighbor $(i_n, j_n, k_n)$ from $L0$ that has the smallest distance value $\phi_{i_n,j_n,k_n}^{n+1}$. Then update:
         $\phi_{i,j,k}^{n+1} = \phi_{i_n,j_n,k_n}^{n+1} + \Delta x$.
      ii. If $|\phi_{i,j,k}^{n+1}| \leq 0.5\Delta x$: remove $(i, j, k)$ from $Lp1$; add it to $S0$.
      iii. If no $L0$ neighbor is found, remove $(i, j, k)$ from $Lp1$.
   5) For every point $(i, j, k)$ in $Ln1$:
      i. Find the neighbor $(i_n, j_n, k_n)$ from $L0$ that has the maximum distance value $\phi_{i_n,j_n,k_n}^{n+1}$. Then update:
         $\phi_{i,j,k}^{n+1} = \phi_{i_n,j_n,k_n}^{n+1} - \Delta x$.
      ii. If $|\phi_{i,j,k}^{n+1}| \leq 0.5\Delta x$: remove $(i, j, k)$ from $Ln1$; add it to $S0$.
      iii. If no $L0$ neighbor is found, remove $(i, j, k)$ from $Ln1$.
   6) For every point in $Sp1$:
      i. Add it to $Lp1$.
      ii. Change the state to 1.
      iii. Remove it from $Sp1$.
   7) For every point in $Sn1$:
      i. Add it to $Ln1$.
      ii. Change the state to -1.
      iii. Remove it from $Sn1$.
   8) For every point in $S0$:
      i. Add it to $L0$.
      ii. Change the state to 0.
      iii. Remove it from $S0$.
   9) For every point $(i, j, k)$ in $L0$:
      i. Search for neighbor points $(i_{np}, j_{np}, k_{np})$ with state $= 2$ and:
         • Change the state of $(i_{np}, j_{np}, k_{np})$ to 1.
         • Update: $\phi_{i_{np},j_{np},k_{np}}^{n+1} = \phi_{i,j,k}^{n+1} + \Delta x$.
   10) Update the time: $t^{n+1} = t^n + \Delta t$. If $t^{n+1} < t_{max}$, go to step 7.
8. Extract the final surface from $\phi$ (zero level set) and visualize it. We use the marching cubed method [57] to solve this step.

3. Results

This section presents an extensive collection of results. For the LS approach, the SFM implementation is used. For comparison, we use the Constant-Time-Stepping (CTS) implementation of the Continuous Cellular Automaton (CCA) presented in [23] in order to simulate the same experiments. The parameters used in the LS simulations are as follows: $Numt = Eq. 9$, $\Delta t = \frac{\Delta k}{\max(4k_{\phi}^{0},4\phi_{\phi}^{0})}$, $\alpha_{F} = 0.48$ and $\Delta N_{F} = 0.01$. Both SFM and CCA implementations are written using sequential Java, similar data structures and single precision, since it has been demonstrated that this level of precision is sufficient to produce accurate results [24]. The testing machine consists on an Intel Core i7 at 2.8 GHz with 4 GB of RAM using 64 bit Windows-based
server Java Virtual Machine (version 1.7.0.03). The purpose of this section is to compare the computational cost, memory use and accuracy of similar implementations of both methods.

3.1. Orientation-dependence of the etch rate

From an experimental point of view the traditional way to represent the anisotropy of an etchant is to show a stereographic projection of the unit sphere while presenting the etch rates at different locations (θ, Φ) using isolines and/or a collection of colors. Since a sphere contains all possible substrate orientations, it provides a complete representation of the etchant anisotropy in a compact way. The procedure originated in early experimental reports, where real hemispherical samples were actually etched [7, 10]. By mechanically probing the hemisphere surface before and after etching at different latitude/longitude locations, the etched distance is determined as a function of the orientation (θ, Φ), thus obtaining the etch rate by dividing by the etch time. Since the procedure can be easily mimicked computationally, in this section we compare the simulated etch rate distributions to those obtained in an extensive, experimental study by Gosálvez et al. [6], where ~ 8100 etch rates were determined for each of a total of 33 different etchants.

Fig. 2 compares the experimental and simulated stereographic projections for four different etching conditions, including a) KOH 40 wt% at 70°C, b) TMAH 25 wt% at 80°C, c) TMAH 25 wt% + Triton 0.1 v/v at 80°C and d) an isotropic etchant. There is much similarity between the experimental and SFM-simulated etch rate distributions, confirming the correctness of our implementation. In comparison, the results from the atomistic CCA method seem more noisy in all the cases, especially for the isotropic etchant. For very long etch times, Fig. 3 shows the actual shape evolution simulated with the SFM for typical spheres etched in the same etchants. As expected, the lower the local etch rate the sharper the corresponding protruding region, which reflects the underlying symmetry of the etch rate: two-, four- and six/three-fold around the {110}, {100} and {111} orientations, respectively.

3.2. Grid resolution

The resolution of the grid used to represent the substrate affects the final accuracy of the simulated microstructures. Typically, the larger the number of grid points the better the accuracy, a phenomenon that is also observed in the atomistic models. As an example, Fig. 4 shows how the number of grid points affects a particular length measurement in both the SFM and CCA simulated fronts for the same etching process: the under-etching of the convex corners of a square-shaped mask pattern on a 128x128 µm² substrate etched in KOH 40 wt% at 70°C for 30 minutes. As shown in Fig. 5, the measured width of the obtained mesa structure decreases as the number of grid points is increased (voxels in SFM and Unit Cells (UCs) in CCA, each cell containing 4 silicon atoms), following a negative exponential behavior that essentially saturates at the 512x512 grid size. Considering a discrepancy of 2% with respect to this value,
the fitted exponentials indicate that the minimal grid resolution to achieve this error is 68062 voxels and 28589 UCs for SFM and CCA, respectively. The fitted exponentials indicate that the minimal grid resolution to achieve this error is 68062 voxels and 28589 UCs for SFM and CCA, respectively. Nevertheless, the discrepancy between SFM and CCA for 512x512 grid points is only 3.8%, thus concluding that the SFM method seems suitable for the simulation of anisotropic etching of real structures.

3.3. Etching of real structures

We now compare the relative performance of the SFM and CCA methods in terms of computational time, memory use and relative error with respect to the experiments. For this purpose, we consider several microengineering structures obtained by the etching of patterned Si[100] wafers in KOH 30 wt% at 80°C [58]. The SFM/CCA-simulated etchant is actually KOH 35 wt% at 80°C, which has a very similar etch rate distribution [6]. Fig. 6 compares all the etched shapes. From left to right, the different columns correspond to (i) the used mask patterns, (ii) the experimental results [58], (iii) the SFM-simulated fronts and (iv) the CCA-based results. The grid sizes, chosen for each simulation to satisfy the 2% error described in the context of Fig. 5, are collected in Table 1, which also presents the etched depth, memory use and computational time for both computational methods. According to the figure, the simulated microstructures by both SFM and CCA are very similar to the experimental shapes. This is confirmed by the values of the simulated etched depths, which are in close proximity to the experimental values. These results demonstrate the reliability of the SFM simulator, achieving similar accuracy and producing less noise than the CCA model.

As shown in Table 1, the required computational times for both methods are within the same order of magnitude for all tests, although CCA can be up to 2 times faster. The larger computational cost for SFM is assigned primarily to the need to determine the maximum in Eq. 7 over all active points, which represents 36-42% of the simulation time. Comparatively, the computation of the spatial derivatives by Eqs. 4 and 5 takes 14-22% of the simulation time. This involves on average more operations than the processing of the atomistic neighborhood in the CCA. Finally, the task of updating the content of the lists (i.e., steps 4) through 9) of the SFM algorithm) is also relevant, representing 13-26% of the whole time. In turn, the reported main memory refers to the main variables stored during the whole simulation, which directly reflects the grid size. Additionally, the memory referred to as lists corresponds to the points stored in the SFM lists and similar structures for the CCA model. The lists use much less memory than the grid since only the necessary information to access to the main grid is stored. Although the SFM implementation typically requires a finer grid, Table 1 shows that the CCA method needs to store more atoms, requiring between 3.7 and 4.8 times more memory.

3.4. Isotropic etching

Isotropic etching is used as a complementary process in connection with anisotropic etching and/or DRIE [59]. For isotropic etchants, the etch rate remains essentially constant along any direction. This behavior can be tested by using a mask with a small circular opening. If the etching process is truly isotropic, a perfect hemispherical cavity will be developed.
Figure 6: Comparison between experimental and simulated etch structures: (left column) applied masks, (center-left) experiments [58], (center-right) SFM results, (right) CCA results.
into the substrate after prolonged etching. Fig. 7(a) compares the results of such a computational experiment for the SFM and CCA methods. It is concluded that the CCA results are noisier and less isotropic, which is consistent with the etch rate distribution for the isotropic etchant shown in Fig. 2. A similar conclusion is obtained from Fig. 7(b), where a square mask pattern leads to the formation of a mesa structure. In both examples the grid size has been chosen by the same criteria explained in the context of Fig. 5, leading to a surface grid of 261 voxels for SFM and 169×169 UCs for CCA.

An interesting feature of the SFM method applied to isotropic etching is that it is not necessary to search the values for the artificial viscosity of Eq. 3 since the etch rate is always constant and, thus, α1 = 1 in all directions. The spherical coordinates of the normal vector are not required either, thus resulting in a faster algorithm for isotropic etchants. This is reflected in the simulation times shown in Fig. 7, where SFM is 10.5 and 4.5 times faster than CCA, respectively, while the memory used by SFM remains lower, as for the anisotropic examples.

4. Exploratory parallel implementations

Although our SFM implementation is able to achieve similar accuracy as the CCA method with less noise in the etch front and roughly 4 times less memory, the computational performance of SFM remains up to 2 times slower than that of CCA for the anisotropic simulations, as reflected in Table 1. To accelerate the performance we present in this Section two parallel-oriented versions of the SFM: (i) gSFM, a truly parallel implementation based on the Nvidia CUDA architecture of modern GPUs, which enables the simultaneous processing of thousands of threads and, thus, substantial savings in the overall computational time, and (ii) cSFM, a parallel-oriented version based on the multi-threading capabilities of modern multicore CPUs, enhanced by the use compilation options, such as the Streaming SIMD Extension (SSE) instruction set, which optimizes the computation of the same instruction over multiple data in a single clock cycle for single/multi-core CPUs. Although the LS method has been successfully parallelized previously, leading to substantially faster performance [60, 37], such procedures have never been applied to the simulation of wet etching for improved MEMS design.

4.1. GPU implementation: gSFM

The use of linked lists does not fit well the GPU computing philosophy. Thus, we have developed a SFM implementation without this data structure. When launching a GPU algorithm (or kernel), several hundreds/thousands of threads are executed in parallel, each targeting a portion of the whole task. In our implementation, each thread is in charge of evaluating all the grid points (voxels) of a vertical column, as recommended by previous CUDA optimizations for similar data access patterns [61]. In order to mimic the SFM behavior, each thread keeps track of the upper and lower spatial boundaries of the active region of a column, thus processing only a few voxels instead of the whole column. This requires the use of an auxiliary state variable (aux_state) to avoid modifying the current state until required. In addition, only the points labelled as state=0 (or state = ±1, depending on the algorithm step) are processed, while the points labelled as state = ±2 are ignored. Using 32-
bit single precision, this implementation fully reproduces the results from the sequential, linked-list version (SFM).

In the CUDA approach the threads are grouped into thread blocks and the kernel launches the execution of a grid of thread blocks, assigning each block to a different Streaming Multiprocessor (SM). The GPU used in this study is the Nvidia GeForce 560 GTX, with 336 cores. Since the multiprocessors execute in parallel groups of 32 threads, called warps, the best performance is obtained when the thread blocks size is a multiple of the warp size. We conclude after some tests that 256 threads per block is the optimal choice. Because the work space is 2D (i.e. a collection of vertical columns in 3D) the block size is chosen to be 16x16 threads and the grid of thread blocks launched by each kernel is 2D. The actual grid dimensions depend on the size of the simulated substrate according to:

\[ \text{Grid}_x = [(\text{substrate}_x + 15)/16] \]  
\[ \text{Grid}_y = [(\text{substrate}_y + 15)/16] \]

where substrate\(_x\) and substrate\(_y\) are the number of voxels, which define the planar size of the substrate, and \([x]\) denotes the nearest integer towards infinity. This scheme guarantees the execution of one thread per column.

A very important aspect of any CUDA implementation is the need to optimize the memory bandwidth by performing memory coalescing, i.e. encouraging/promoting that contiguous threads access contiguous memory locations during read/write operations. This is easily obtained when updating the values of \(\phi\), since each thread only needs to access its memory position. However, when each thread needs to access the neighboring \(\phi\) values to calculate the spatial derivatives, memory coalescing is better achieved by directly using the texture memories of the GPU, since these are designed for exploiting memory access locality. With the chosen thread memory order, we find that this memory offers good performance for neighborhood access. In addition, the values of the etch rates \(R(\theta, \Phi)\) are also stored in the texture memory to improve memory coalescing when calculating the local velocity of the front.

Based on the algorithm shown in Section 2.2, our gSFM implementation makes use of 5 kernels:

- **Kernel1** (Corresponds to step (7.1) of section 2.2)
  For every column voxel between upper and lower boundaries labelled as state = 0, determine:
  1. The forward and backward derivatives using Eq. 5.
  2. The normal components \(N_l = \phi_l/|\nabla \phi|\) where \(l = x, y\) and \(z\).
  3. The etch rate (convert the cartesian components to spherical coordinates and access the corresponding matrix element).
  4. The maximum value according to Eq. 7.

- **Kernel2** (corresponds to step (7.3) of section 2.2)
  For every column voxel between upper and lower boundaries labelled as state = 0:
  1. Update \(\phi\) using Eq. 3.
  2. If \(\phi^{n+1}_{i,j,k} < -0.5\Delta x\) label (i, j, k) with aux_state = -1.
  3. If \(\phi^{n+1}_{i,j,k} > 0.5\Delta x\) label (i, j, k) with aux_state = +1.

- **Kernel3** (corresponds to steps (7.4)-(7.5) of section 2.2)
  For every column voxel between upper and lower boundaries:
  - If (i, j, k) is labelled as state = 1
    1. Find the neighbor \((i_n, j_n, k_n)\) with state = 0 that has the smallest distance value \(\phi^{n+1}_{i_n,j_n,k_n}\). Then update: \(\phi^{n+1}_{i,j,k} = \phi^{n+1}_{i_n,j_n,k_n} + \Delta x\).
    2. If \(|\phi^{n+1}_{i,j,k}| \leq 0.5\Delta x\): label (i, j, k) with aux_state = 0.
    3. If no state = 0 neighbor is found, label (i, j, k) with state = 2 and aux_state = 2.
  - If (i, j, k) is labelled as state = -1
    1. Find the neighbor \((i_n, j_n, k_n)\) with state = 0 that has the maximum distance value \(\phi^{n+1}_{i_n,j_n,k_n}\). Then update: \(\phi^{n+1}_{i,j,k} = \phi^{n+1}_{i_n,j_n,k_n} - \Delta x\).
    2. If \(|\phi^{n+1}_{i,j,k}| \leq 0.5\Delta x\): label (i, j, k) with aux_state = 0.
    3. If no state = 0 neighbor is found, label (i, j, k) with state = -2 and aux_state = -2.

- **Kernel4** (corresponds to step (7.6)-(7.8) of section 2.2)
  For every column voxel between one position above upper boundary and one position below lower boundary:
  1. Update state = aux_state.
  2. If any voxel labelled with state = 0 is above (below) the upper (lower) boundary, increment (decrement) it in one voxel.

- **Kernel5** (corresponds to step (7.9) of section 2.2)
  For every column voxel between upper and lower boundaries:
  - Search for neighbor points \((i_{hp}, j_{hp}, k_{hp})\) with state = 2 and:
    1. Label \((i_{hp}, j_{hp}, k_{hp})\) with state = 1 and aux_state = 1.
    2. Update: \(\phi^{n+1}_{i,j,k} = \phi^{n+1}_{i_{hp},j_{hp},k_{hp}} + \Delta x\).
  - Search for neighbor points \((i_{hp}, j_{hp}, k_{hp})\) with state = -2 and:
    1. Label \((i_{hp}, j_{hp}, k_{hp})\) with state = -1 and aux_state = -1.
    2. Update: \(\phi^{n+1}_{i,j,k} = \phi^{n+1}_{i_{hp},j_{hp},k_{hp}} - \Delta x\).

Fig. 8 shows the workflow of the implemented gSFM algorithm. All the steps performed by the CPU are identical to those in the sequential algorithm of Section 2.2, except that the addition/removal of voxels to/from the SMF lists occurs by modifying the corresponding labels of state and aux_state of each voxel. In order to search the maximum value of Eq. 7 over every voxel with state = 0, each thread searches the maximum within a grid column and this value is passed to the CPU.
Thus, in each iteration the CPU has to find the maximum value among the maxima found by each thread. Once obtained, the three maxima $\alpha_1$, $\alpha_2$, and $\alpha_3$ are returned to the GPU, which then continues with the next step. Although there is room for further optimization, the time required by the CPU to find the maximum value over a two dimensional grid is negligible while the time spent doing the transfer from the GPU to the CPU is only about 2.5% of the total GPU time. To minimize data transfer from the CPU to the GPU, all the model information is stored in the GPU, in such a way that data transfers take place only at the beginning and end of the simulation, as shown by steps 5 and 7 in Fig. 8. At the beginning, the CPU sends the following data to the GPU: (i) the current values of the signed distance function $\Phi$, (ii) the etch rates $R(\theta, \Phi)$, (iii) the current state of the voxels and (iv) the mask pattern. At the end, the GPU copies the final values of $\Phi$ to the CPU.

4.2. CPU implementation: cSFM

Since the benchmarks of this study are performed on a quad-core 2.8 GHz Intel Core i7, we consider the possibility to optimize the performance by taking advantage of specific properties of this CPU. This includes the use of (a) Intel’s Streaming SIMD Extensions (SSE) for improved data transfer within the microprocessor and (b) multi-threading options due to the multicore nature of the microprocessor.

The SSE instruction set enables applying the same instruction to multiple data objects in one clock cycle. Based on the use of Microsoft Visual Studio 2010 in this study, the code is automatically optimized by generating SSE2 instructions using the /arch:SSE2 compilation option. On the other hand, there are different choices to take advantage of the multicore architecture, such as openMP directives or the Open Source POSIX Threads (Pthreads) library recently adapted to Windows O.S. Based on the use of the Pthreads library in this study, we avoid data conflict by enforcing that (i) each thread creates a sub-list of SFM lists and (ii) access to each list is exclusive for the creator. As an example, if two threads ($i = 0$ and $i = 1$) are created, the following lists are generated: $L0_{\text{sub}}[i]$, $Lp_{1_{\text{sub}}}[i]$, $Ln_{1_{\text{sub}}}[i]$, $S0_{\text{sub}}[i]$, $Sp_{1_{\text{sub}}}[i]$ and $Sn_{1_{\text{sub}}}[i]$. Nevertheless, barriers are required after steps 7.1), 7.2) and 7.4) through 7.9) of the main loop of the algorithm shown in Section 2.2 to ensure that all the values of the global variables, such as the distance function, are updated correctly. Taking advantage of both the SSE optimization and multi-threading, the cSFM implementation with 8 threads is about 3.9 – 4.9 times faster than the unoptimized, sequential code for the simulation of anisotropic etching.

4.3. Results

This section compares the computational performance of the cSFM and gSFM implementations. The cSFM program is developed in C while the gSFM code is implemented in CUDA C. Both implementations are executed on a 2.8 GHz Intel Core i7 with 4 GB of RAM and an Nvidia GeForce 560 GTX using single precision (32 bits). In the compilation of the cSFM code the following options have been selected: /arch:SSE2, /fp:fast, /Ox and /Ot. In addition, the best performance is achieved when creating 8 execution threads.

The comparison is made by considering the systems shown in Figs. 6-7. Since both versions implement the SFM the results are identical to those shown in both figures. In addition, the memory use by cSFM is also the same than that presented in Table 1 and Fig. 7. On the other hand, gSFM does not use any linked lists but requires storing the auxiliary state variable, the maximum alpha values of each column as well as the upper and lower boundaries of each thread, implying an small increase of the main memory usage. In terms of computational performance, Table 2 collects the corresponding data for the cSFM/gSFM results of Figs. 6-7. The table demonstrates that the GPU algorithm runs up to 7.4 times faster than the CPU version. Although the results confirm that current CPUs stand as an acceptable platform to perform this type of wet etching simulations, the massively parallel architecture of current GPUs provides better computational efficiency with only a slightly larger use of memory. In fact, gSFM is expected to become increasingly more efficient when simulating larger substrates, since the massively parallel architecture of a GPU is progressively more suitable when more and more threads are created, in contrast to the multi-core CPU. Nevertheless, both implementations of the SFM algorithm produce simulated results within a few seconds and combine both the computational efficiency and the algorithmic accuracy required for the realistic simulation of wet etching fronts for improved MEMS design.

5. Conclusions

Based on the Level Set (LS) method, a simulator of anisotropic wet chemical etching is developed, aiming at versatile design of Micro-electro-mechanical systems (MEMS). The
Table 2: Performance comparison between the cSFM and gSFM implementations for the systems shown in Figs. 6-7. The etched shapes of both implementations are identical to those presented in Figs. 6-7.

<table>
<thead>
<tr>
<th>Simulation time (s)</th>
<th>Memory usage (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cSFM</td>
</tr>
<tr>
<td>cSFM</td>
<td>2.7</td>
</tr>
<tr>
<td>gSFM</td>
<td>0.37</td>
</tr>
<tr>
<td>cSFM</td>
<td>0.84</td>
</tr>
<tr>
<td>gSFM</td>
<td>4.2</td>
</tr>
<tr>
<td>cSFM</td>
<td>2.8</td>
</tr>
<tr>
<td>gSFM</td>
<td>0.77</td>
</tr>
<tr>
<td>cSFM</td>
<td>2.15</td>
</tr>
<tr>
<td>gSFM</td>
<td>1.63</td>
</tr>
</tbody>
</table>

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions. This work has been supported by the Spanish FPI-MICINN BES-2011-045940 grant and the Ramón y Cajal Fellowship Program by the Spanish Ministry of Science and Innovation. Also, we acknowledge support by the JAE-Doc grant form the Junta para la Ampliación de Estudios program co-funded by FSE and the Professor Partnership Program by NVIDIA Corporation.

References


R. Courant, K. Friedrichs, H. Lewy, Über die partiellen differenzen- gleichungen der mathematischen physik, Mathematische Annalen 100 (1928) 32–74.


Simode, collection of examples (c), gesselschaft fur mikroelektronikanwendung chemnitz mbh.


R. Courant, K. Friedrichs, H. Lewy, Über die partiellen differenzen- gleichungen der mathematiscn physik, Mathematische Annalen 100 (1928) 32–74.


Simode, collection of examples (c), gesselschaft fur mikroelektronikanwendung chemnitz mbh.


14