Computer Physics Communications 220 (2017) 205–211

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Suppressing correlations in massively parallel simulations of lattice models

Jeffrey Kelling [a,*], Géza Ódor [b], Sibylle Gemming [c,d]

[a] Helmholtz-Zentrum Dresden-Rossendorf, Department of Information Services and Computing, Bautzner Landstraße 400, 01328 Dresden, Germany
[b] Institute of Technical Physics and Materials Science, Centre for Energy Research of the Hungarian Academy of Sciences, P.O. Box 49, H-1525 Budapest, Hungary
[c] Helmholtz-Zentrum Dresden-Rossendorf, Institute of Ion Beam Physics and Materials Research, Bautzner Landstraße 400, 01328 Dresden, Germany
[d] Institute of Physics, TU Chemnitz, 09107 Chemnitz, Germany

ARTICLE INFO

Article history:
Received 19 May 2017
Received in revised form 9 July 2017
Accepted 11 July 2017
Available online 27 July 2017

Keywords:
Lattice Monte Carlo
Kardar–Parisi–Zhang
GPU
Autocorrelation

ABSTRACT

For lattice Monte Carlo simulations parallelization is crucial to make studies of large systems and long simulation time feasible, while sequential simulations remain the gold-standard for correlation-free dynamics. Here, various domain decomposition schemes are compared, concluding with one which delivers virtually correlation-free simulations on GPUs. Extensive simulations of the octahedron model for 2 + 1 dimensional Kardar–Parisi–Zhang surface growth, which is very sensitive to correlation in the site-selection dynamics, were performed to show self-consistency of the parallel runs and agreement with the sequential algorithm. We present a GPU implementation providing a speedup of about 30× over a parallel CPU implementation on a single socket and at least 180× with respect to the sequential reference.

1. Introduction

Lattice Monte Carlo (MC) simulations are employed widely to out-of-equilibrium problems [1]. Examples range from growth processes, such as non-equilibrium surface growth [2] or domain growth after phase separation [3], to evolutionary game theory [4]. Simulations of such processes must reproduce the physical kinetics in real systems. This is fundamentally different from equilibrium problems, where it is admissible to change the kinetics to speed up the relaxation, for example by cluster algorithms [5,6]. Out-of-equilibrium simulations are not at liberty to apply such optimizations.

In practice, the most efficient way to perform lattice MC simulations on a bipartite lattice is checkerboard, or sub-lattice parallel [7], updates, which fit the definition of a stochastic cellular automaton (SCA) [8] since the dynamics considers each site as strictly independent from all other sites on the same sub-lattice. Algorithms of this type can be parallelized very efficiently on many architectures, including GPUs [9–12]. However, in this scheme, the selection of lattice sites is correlated, which influences the kinetics [7]. As we have shown recently [13], this artificial dynamics can indeed affect the dynamical universality class of lattice gas models.

Markov chain MC models, like the Metropolis algorithm [14] and surface growth models [15–18], are usually defined as a series of single-particle updates. To leave the update attempts uncorrelated they must be performed in a random-sequential (RS) fashion, which cannot be parallelized by definition. As an approximation domain decomposition (DD) can be used, where random site selection is restricted to a local domain per parallel worker [19,20].

Here we review different DD schemes for parallel GPU viable for implementations of lattice MC and present one which is virtually free of correlations. For this purpose we consider the 2 + 1-dimensional octahedron model [18] for Kardar–Parisi–Zhang (KPZ) surface growth [21] with RS dynamics and compare autocorrelations for the various types of DD.

In Section 2 we define the octahedron model and introduce the basics of KPZ surface growth and aging. The section concludes with a brief summary of properties of the GPU architecture most relevant to this work. In Section 3, the considered DD schemes are introduced, their impact on simulation results is presented in Section 4, which ends with a comparison of the performance achieved by our implementations. We conclude in Section 5.

2. Models and methods

2.1. Octahedron model for ballistic deposition

The octahedron model is illustrated in Fig. 1: Octahedra are deposited or removed at eligible sites with probabilities p and q,

* Corresponding author.
E-mail address: j.kelling@hzdr.de (J. Kelling).

http://dx.doi.org/10.1016/j.cpc.2017.07.010
0010-4655/

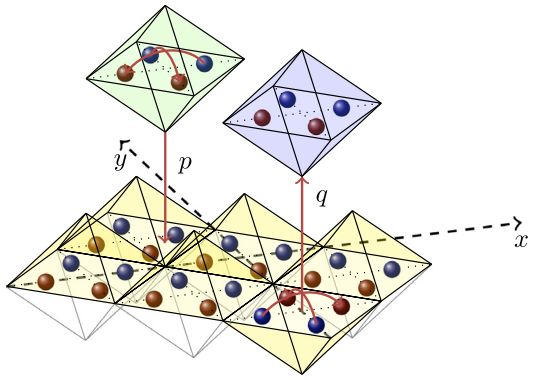

Fig. 1. Illustration of the octahedron model. The letters p and q label deposition and removal processes, respectively. The edges of the octahedra are encoded as up and down slopes which form a lattice gas with two species of particles drawn as red and blue balls, respectively. The red curved arrows connected to the update processes mark the directions of particle exchange in the corresponding lattice gas updates. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

respectively. The slopes of the octahedra edges are mapped to a lattice gas with the Kawasaki [22] update rules

$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \underset{q}{\overset{p}{\rightleftharpoons}} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}, \tag{1}$$

which amount to the exchange of dimers along the bisection of the Cartesian x and y axes. The peaks of the octahedra define the height profile of a two-dimensional surface growing through random deposition of particles.

In the case $p = 1, q = 0$, which we focus on, the growth of the height is described by the KPZ equation [21]:

$$\partial_t h(\mathbf{x}, t) = \sigma \nabla^2 h(\mathbf{x}, t) + \xi(\nabla h(\mathbf{x}, t))^2 + \eta(\mathbf{x}, t) + v, \tag{2}$$

where σ models a surface tension smoothening the surface in competition with a local growth velocity ξ and an uncorrelated, zero-mean Gaussian white noise η roughening it. The roughness of the surface is defined as:

$$W^2(L, t) = \frac{1}{L^2} \sum_{i,j}^{L} h_{i,j}^2(t) - \left(\frac{1}{L} \sum_{i,j}^{L} h_{i,j}(t) \right)^2, \tag{3}$$

where L denotes the lateral size of the system. It grows following a Family–Vicsek law [23],

$$W(L, t) \propto L^{\alpha} f(t/L^z),$$

with the dynamical exponent z and the universal growth function:

$$f(u) \propto \begin{cases} u^{\beta} & \text{if } u \ll 1 \\ \text{const.} & \text{if } u \gg 1. \end{cases} \tag{4}$$

The case $u \ll 1$ is called the growth regime characterized by the growth exponent β has been determined numerically in many studies [13,19,24–26], with the currently most precise estimate being $\beta = 0.2414(2)$, which is based on simulations using the implementation we are presenting here [13].

Out-of-equilibrium systems like KPZ surfaces age and can thus be best characterized by two-time quantities like autocorrelation functions:

$$C(t, s) = \langle \phi(t, \mathbf{r}) \phi(s, \mathbf{r}) \rangle - \langle \phi(t, \mathbf{r}) \rangle \langle \phi(s, \mathbf{r}) \rangle ,$$

where s denotes the waiting time, which must be in the aging regime $\tau_{\text{micro}} \ll s,\ t - s \ll L^z$. One expects the scaling law:

$$C \propto s^{-b} (t/s)^{-\lambda/z}, \tag{5}$$

which defines the autocorrelation and aging exponents λ and b, respectively. The autocorrelation can be calculated for slope variables, i.e. the lattice gas:

$$C_s(t, s) = \left\langle n(t; \vec{r}) n(s; \vec{r}) \right\rangle - \left\langle \overline{n}(t; \vec{r}) \right\rangle \left\langle \overline{n}(s; \vec{r}) \right\rangle ,$$

as well as the surface heights:

$$\begin{aligned} C_h(t, s) &= \left\langle h^2(t, \mathbf{r}), h^2(s, \mathbf{r}) \right\rangle - \langle h(t, \mathbf{r}) \rangle \langle h(s, \mathbf{r}) \rangle \\ &\propto s^{-b_h} \cdot (t/s)^{-\lambda_h/z}. \end{aligned} \tag{6}$$

For $t = s$, one finds:

$$C_h(t = s, s) = W^2(L \to \infty, s) \propto s^{-b_h},$$

and, after comparing to Eq. (4):

$$b_h = -2\beta, \tag{7}$$

which must be satisfied in the $L \to \infty, s \to \infty$ limit and thereby fixes the aging exponent b_h in our study.

The simulation time is measured in complete sweeps of the lattice, Monte Carlo step (MCS).

In order to characterize the finite-time behavior of quantities following power-laws, i.e. W and C, we calculate effective exponents as follows:

$$e_{\text{eff}} \left(\frac{t_i - t_j}{2} \right) = \frac{\ln E(t_i) - \ln E(t_j)}{\ln(t_i) - \ln(t_j)}, \tag{8}$$

where $E \equiv W, C$; $e_{\text{eff}} \equiv \beta_{\text{eff}}, \lambda_{\text{eff}}$ and $t_i > t_j$.

2.2. Parallel implementation

Modern computing architectures, such as GPUs, contain large numbers of elements executing operations in parallel. The general concept of program units executing on these elements may be called *workers*, it is those the computational load needs to be distributed among. In this work we apply DD to parallelize lattice MC algorithms. While this confines each worker to a small region of the lattice, site selection must remain random within this region to keep updates uncorrelated. The random memory access patterns produced this way constitute the main obstacle for an efficient implementation, thus we shall briefly recall the most important concepts of GPU architecture and its memory hierarchy, before we introduce the DD schemes we consider. Here, only a bird's eye view of or CUDA [27] implementation can be provided, which shares many technical details with earlier implementations [20].

For the purposes of this work, we view GPUs as an assembly of vector processors or *compute units* sharing *global device memory* and each having its own fast on-chip memory, which we call *local memory*. This constitutes a two-layered architecture, which DD schemes must be designed around.

1. Work must be distributed at the *device-layer* among compute units. Since these share the global memory, we assume, that the state of the whole lattice is stored there.
2. Each compute unit executes blocks of worker threads in lockstep. Within such a block, work must be distributed among the workers, which gives rise to the *block-layer*. All workers in a block share access to their compute unit's local memory.

In lattice models, divergence of code paths between workers can usually be avoided easily. A more relevant problem is, that accesses of workers to the global memory must be coalesced to be efficient, which strongly discourages random access patterns. For this reason device-layer domains are loaded into local memory before updates are performed. Accesses to local memory can be random as long as bank conflicts are avoided, which can be done by padding multidimensional data by one word.

The size of the local memory, currently 48 kB on NVIDIA devices, imposes constraints on the DD. Foremost, device-layer domains

must fit into this, including any required border regions. High occupation of the device can only be reached if a large number of threads, possibly 1024, workers are executing per block, this means each device-layer domain must decompose into as many block-layer domains, making the latter necessarily rather small.

Our simulations obey periodic boundary conditions. System sizes and the length of simulation runs are chosen such that finite-size effects are negligible.

3. Domain decomposition schemes

DD schemes shall first be treated in a generic fashion. When the given problem consists in distributing simulation lattice among multiple *workers* which are to perform full updates asynchronously, the solutions can be formulated independent of the parallel architecture. The application of these schemes at device and block layer will be detailed below.

3.0.1. Dead border (DB)

In the dead border (DB) DD scheme, used by the GPU implementation of the octahedron model in [19,28], the lattice decomposes into tiles, where the rim of each is kept inactive during asynchronous updates (*dead border*), so that no update of the site in the active part of a tile would affect or be affected by the state of site in a neighboring tile. If a border site is selected the update is not carried out. After the asynchronous interval t_{async}, the origin of the tiling is moved randomly before the next asynchronous updates, displacing the dead borders so that former inactive border sites may be updated and propagate changes between tiles which were separated before. Since not all sites are active during a sweep of the lattice, a time step under DB is a little shorter than a whole sweep:

$$1\text{MCS}_{DB} = 1/(N_{\text{latticesites}} - N_{\text{bordersites}})\text{MCS}. \tag{9}$$

In general, sites on the rim of tiles in all directions need to belong to the dead border. In the octahedron model on a square lattice, where only the links between sites carry state (the slope), only one rim in each spatial direction needs to be inactive. This is because, for each direction one slope is encoded on-site (to the left neighbor), hence only the other one must be retrieved from the right neighbor. Fig. 2(a) illustrates DB DD in 2d.

3.0.2. Double tiling (DT)

Parallel implementations of lattice Metropolis Monte Carlo methods [14,29] use double tiling (DT) for DD [20], this scheme is illustrated in 2d in Fig. 2(b). Here, the system is decomposed into tiles, which are split into two sub-tiles in each spatial direction, creating 2^d sets of non-interacting domains, where d is the dimension. These domain sets are updated in a random order, which is a permutation shuffled uniformly at each MCS, synchronization occurs after completing each sweep of a domain-set. Sub-tiles do not comprise inactive sites, thus updates of lattice sites on the rim of a sub-tile will affect sites in neighboring sub-tiles which are at that time inactive.

In DT borders of DD domains always remain at the same place, which enables higher performance, because, not having to deal with arbitrary decomposition origins, memory alignment can be controlled and the amount of data that needs to be exchanged between workers is reduced. The disadvantage is that it allows errors of the effective fixed boundary condition approximation to accumulate which can be mediated in models with explicitly thermally activated dynamics. This scheme is widely used for example in a parallel implementation of the n-fold way algorithm [30].

3.0.3. DT DD with random origin (DTr)

In a model where the only source of randomness is the random selection of lattice sites, i.e. without explicitly thermally activated dynamics, such as the octahedron model, site selection must not be biased. Thus, a DD scheme which allows imbalances in the site-selection to accumulate at specific places (sub-tile boundaries), however slightly, cannot be expected to perform well in general. The accumulation can be removed by randomly moving the decomposition origin like with DB (see Fig. 2(c)), which was also done in off-lattice simulations before [31]. In the DT DD with random origin (DTr) scheme restricting the random origin to a coarse grid is not necessary in any case, since there effectively are dead borders of the same width as the decomposition domain, which easily provides enough padding to avoid having words shared between workers.

3.1. Application of DD on GPUs

In a GPU implementation DD must be applied both at device and block layer, where any combination of the schemes mentioned above is possible in principle.

At device layer, implementations copy the to-be-updated decomposition domains into local memory, including all bordering region which affect or are affected by updates. Here, thread blocks assume the role of workers. Synchronization occurs after a sweep of the domains has been completed. Communication of the boundaries is implemented by copying the data back to global memory and loading back the new domain configuration prior to the next sweep. It is not possible to keep non-shared parts of domains in local memory, because it is not persistent over device synchronization events (i.e. device kernel invocations).

In the implementation of the octahedron model, each 32-bit word encodes 4×4 lattice sites, thus, freely picking a random origin would result in words becoming shared between neighboring tiles. For reasons of performance, the early implementation [19] restricted moving of the DD origin in such a way, that the borders would always be located at the edge of 32-bit words. Essentially, the origin was moved randomly on a coarse grid with steps of 4×4 lattice sites. This variation will henceforth be labeled coarse dead border (cDB). DB was also implemented without restriction of the DD origin to a coarse grid, by increasing the width of the dead boarder to five lattice sites. This padding ensures that if a word is shared between tiles, the left tile will not need to write to it, because all lattice sites encoded in this word, belonging to the left tile, will be inactive and will not have an active neighbor.

Since local memory is shared between threads, the DD at block layer is implemented logically, all threads access the domains assigned to them directly. Due to the large number of threads and the limited amount of local memory, block layer domains are rather small. The smallest size we consider here is 8×8 lattice sites. Performing a complete sweep of 64 updates on these would allow fixed-boundaries errors to become too large. However, since groups of threads are executing in lockstep and accesses to global memory do no occur while updates are performed, synchronization of threads is cheap. This allows collective single-hit updates to be performed, where each domain only receives one random update before synchronization, which practically eliminates fixed-boundary effects. Here, DT schemes pick a domain set for each update at random, because using a permutation would not only be computationally more expensive, but would also decrease site-selection noise more than necessary at this level.

When using schemes with random decomposition origin (DB or DTr), the origin is moved without restrictions and thus words are likely to be shared between neighboring thread cells (TCs). Atomic operations are therefore used to update the domain in shared memory.

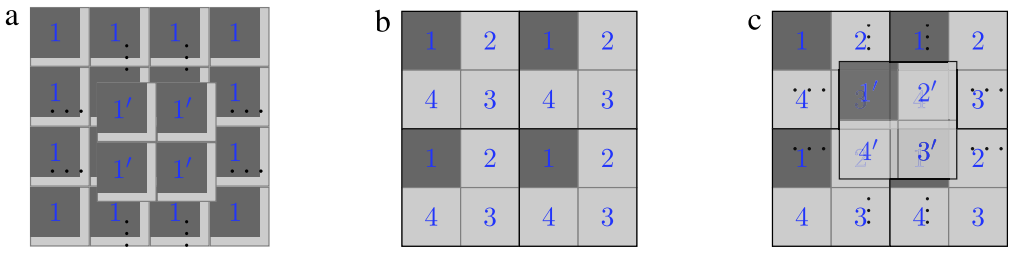

Fig. 2. 2d realizations of DD schemes that have been evaluated for parallel implementations of the (2 + 1)d octahedron model. All of them could be straightforwardly applied in setups of arbitrary dimension. Dark areas indicate regions being updated concurrently while the light areas are inactive, acting as buffers. Numbers indicate a (randomized) sequence of synchronous steps in which the asynchronous domain updates are performed. Domains labeled with primed numbers illustrate a possible randomized decomposition after moving the decomposition origin: $O \to O'$. Domains calculated based on a random origin will always wrap around the system (periodic boundary conditions (PBC)), even if PBC are not used in the simulation. Dead border (DB) (a) is displayed for the special case of the octahedron model (and other slope-based surface growth models) where updates can affect neighboring cells only in positive x or y direction. In general, all cell edges would be dead borders.

In the present work, single-hit DB is implemented using delayed borders. This means, that updates hitting the border are not discarded, but delayed until all bulk updates are completed. Updates to the corners of cells, i.e. those which affect two borders, are carried out last. For the octahedron model, where either $p > 0$ or $q > 0$, but not both, which is the usual case when simulating the KPZ universality class, updates can never be allowed for two nearest neighbor (NN) sites at the same time, because slopes are restricted to ± 1. Thus, in this case, delayed borders are not required to avoid conflicts between updates and updates ignoring borders are completely equivalent to updates with delayed borders. Hence, in the present work, all simulations stated as using DB at block-level are actually ignoring borders at block-level if only one of p and q is finite.

3.1.1. Notation for DD configuration

We refer to combinations of device and block layer DD by concatenating the acronyms of the schemes used at both layers, putting the device layer scheme first. The most interesting combinations discussed later are DTrDT—double tiling with random origin at device and double tiling at block level—and DTrDB—double tiling with random origin at device and dead border at block level. We only consider single-hit updates at block layer.

The size and shape of thread cells are given in the notation `TC=log_2(x), log_2(y)`, where x and y are lateral sizes of the cells in words (4 lattice sites). For DT and DTr, the smallest possible tile size is `TC=1,1` (8 × 8 sites), because the domains are split in half to form sub-tiles and the implementation requires active regions to be at least the size of one word. DB would technically allow `TC=0,0`, but here we consider decomposition domains not smaller than 8 × 8 lattice sites.

4. Results

4.1. Correlation artifacts of DD schemes

In this section we discuss the correlation artifacts of DD schemes. Aligned with this two-layered GPU architecture, we will discuss schemes at device layer first and then move on to the block layer.

4.1.1. Device-layer DD

In an aging study published in [28] it was found, that the GPU implementation of the octahedron model with RS dynamics used therein exhibits an asymptotic autocorrelation function deviating from the sequential CPU reference implementation. This parallel implementation uses cDB, where the DD origin is only moved on a coarse grid with 4 × 4 lattice-site units. Fig. 3 compares autocorrelation functions of both heights (a) and slopes (b) resulting when cDB is employed at device-layer with other schemes and

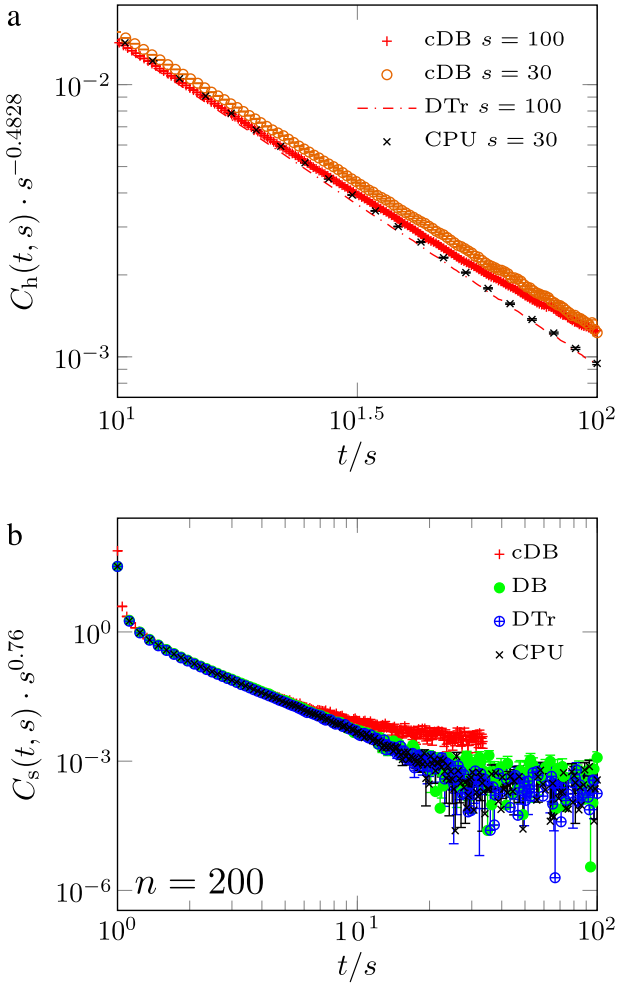

Fig. 3. Autocorrelations under RS dynamics with different device-layer DDs. (a) Autocorrelation of heights (data from [28]). The system sizes are $L = 2^{15}$, $L = 2^{16}$ and $L = 2^{13}$ in the cDB, the DTr and the sequential (CPU) runs, respectively. Sample sizes are $n = 696, 830, 71, 4367$, as they appear in the legend from top to bottom. (b) Autocorrelation of slopes for different types of device-layer DD compared to a sequential simulation at $L = 2^{13}$. All sample sizes are $n = 200$, so noise levels can be compared visually. The presented parallel runs use single-hit DT at block-layer.

sequential simulation results. For the slopes one clearly observes convergence of the autocorrelation function to a finite value. The autocorrelation of heights under cDB can also be seen to deviate from the power law (PL) laid-out by the sequential reference. A finite asymptotic limit has not been reached in any of these simulations due to the short time-scale considered.

Intuitively, a finite asymptotic value for the autocorrelation function could mean that there is some pattern imprinted on the system during its evolution. When the DD origin is only shifted on a coarse grid, only lattice sites which lie on edges of this coarse grid can become borders. These sites are thus updated less frequently than the remaining sites, which never become border-sites. Hereby, two types of sites are defined in the system evolving at different rates. Since these borders are only a single lattice site wide and device-layer domains are very large, the effect of this is apparently too small to be observed in the kinetics of surface roughening or steady state properties. It is, however, strong enough to imprint a persistent pattern onto the surface, which can be observed in the autocorrelation functions.

A straightforward way to solve this problem, would be to not shift the DD origin on a coarse grid but allow arbitrary coordinates, which does indeed eliminate the observed correlations. For the present bit-coded implementation, unrestricted DB requires borders which are five inactive lattice sites wide. Moving the origin freely ensures that all lattice sites are updated with the same frequency, when sufficiently long times are considered. Ideally, border sites are only inactive for one asynchronous update sweep at a time (t_{async}). But, after moving the DD origin randomly, the old and new borders will necessarily intersect at a grid of points, which are then inactive for $2t_{\text{async}}$. Due to the wide borders, these intersections cover patches of 5×5 lattice sites. Thus, the wide borders are producing locally varying update frequencies at short time scales. This manifests as additional noise in the autocorrelation functions and possibly other observables.

All curves presented in Fig. 3(b) are averaged over the same number $n = 200$ of runs. At late times ($t/s > 30$) autocorrelation signals have decayed below the noise-level of the respective sample, so the variance can be compared. Reducing t_{async} decreases adverse effects of DD borders and thus also the additional noise caused by wide borders. The data shown for DB stems from simulation using $t_{\text{async}} = 0.5$ MCS. Even larger noise is observed for $t_{\text{async}} = 1$ MCS.

This study shows double tiling (DT) DD with random origin (DTr) to be superior to the other presented schemes: It appears to neither introduce correlation nor additional noise, compared with a sequential simulation. It has only the disadvantage of using smaller active domains for asynchronous update sweeps, but this influence is negligible for the large domain sizes usually used at device-layer and it serves to keep update frequencies homogeneous. For this reason it may even be preferable over DB with a border-width of one lattice site, since intersections of thin borders could still cause tiny imbalances. Using DT without randomly moving the DD origin results in similar correlations as exhibited by cDB.

As the goal of this work is present a virtually correlation-free approach, all further considerations are based on DTr as device layer DD.

4.1.2. Block-layer DD

The observed changes in autocorrelation functions with block-layer DD are so small, that they could not be resolved by most sequential simulations. At the same time sequential and parallel simulations may differ by small corrections, not affecting any universal properties. Thus, even if a small difference between parallel and sequential results could be resolved, it would be unclear if this was a cause for concern. For these reasons, comparisons with sequential simulations would not be constructive at this point.

In simulations using DD, the size and shape of domains remain free parameters, where the exact sequential behavior corresponds to the limit of infinite domain size. This view suggests checks for self-consistency as a viable method for this analysis.

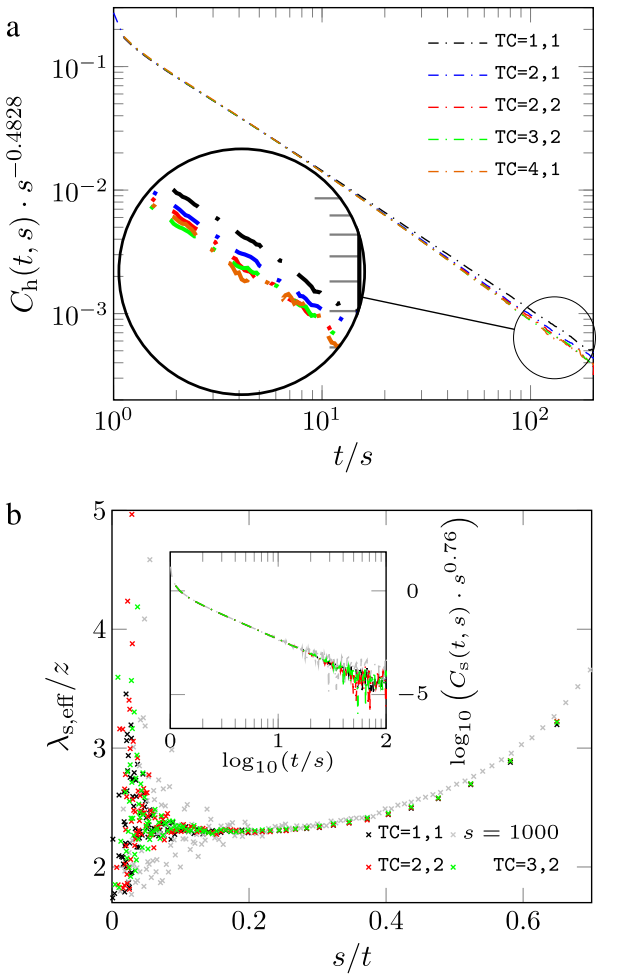

Fig. 4. This figure presents the effects of different TC configurations on RS autocorrelation results when using DT for DD at block-layer (DTr at device level and single-hit DT at block level (DTrDT)). (a): Comparison of different TC volumes and shapes for waiting time $s = 100$. (b) Comparison of autocorrelation (AC) of slopes for different TC configurations at $s = 100$ and, for the smallest configuration TC=1,1, $s = 1000$. The system size for TC=2,1 is $L = 2^{17}$, all other system sizes are $L = 2^{16}$. Sample sizes vary: $n_{\text{TC=1,1}} \geq 38$; $n_{\text{TC=2,1}} \geq 120$; $n_{\text{TC=2,2,T4,3}} \geq 71$; $n_{\text{TC=2,2,T5,4}} \geq 20$; $n_{\text{TC=3,1}} \gtrsim n_{\text{TC=1,3}} \geq 78$; $n_{\text{TC=3,2}} \geq 28$ and $n_{\text{TC=4,1}} \geq 30$. Statistical 1σ-errors are below 5×10^{-4} for all data points.

Fig. 4(a) shows autocorrelation functions for heights when using DTrDT for different block-layer domain configurations. Different TC configurations appear to show a trend in the autocorrelation functions at late times:

$$C_{\text{TC=1,1}} > C_{\text{TC=2,1}} > C_{\text{TC=2,2}} \gtrsim C_{\text{TC=3,2}} \approx C_{\text{TC=4,1}}.$$

For the most part, the differences are barely significant, but quite clear between the configuration with the smallest volume block-layer domains, TC=1,1, and the others. If lateral cell dimensions dominated, the configuration TC=4,1 would be expected to give more similar results, instead it much better agrees with TC=3,2, which features block-layer domains with the same volume ($V_{\text{TC=4,1}} = V_{\text{TC=3,2}} = 16 \cdot 2^5 = 256$ lattice sites).

Contrary to the these observations for the autocorrelation of heights, the autocorrelation of slopes is identical in all these simulations. This is shown in Fig. 4(b) by overlaying the effective exponents $\lambda_{\text{s,eff}}/z$, corresponding to the autocorrelation functions of the slopes shown in the inset.

The reason DTrDT produces some small correlated noise may be found in the grid-like site-selection pattern at block-layer, where the coordinates for collective update attempts are restricted to the

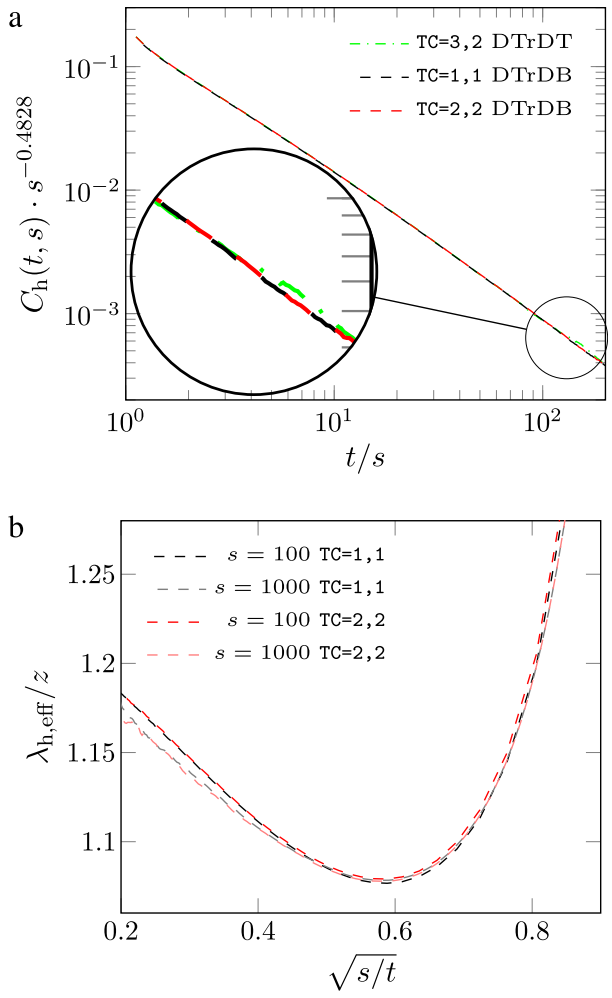

Fig. 5. Comparison of autocorrelation results in RS simulations using DB for DD at block-layer (DTr at device level and single-hit DB at block level (DTrDB)). (a) Autocorrelation functions for different TC sizes for waiting time $s = 100$. Results using DTrDT with TC configuration TC=3,2 are given for comparison. (b) Local slope analysis of DTrDB results for $s = 100$ and 1000. All system sizes are $L = 2^{16}$. Sample sizes are: $n_{\mathrm{TC}=1,1} \geq 1044$; $n_{\mathrm{TC}=2,2} \geq 708$ and (DTrDT) $n_{\mathrm{TC}=3,2} \geq 28$. Samples sizes of DTrDB runs are much larger because these stem from production runs which where used to extract final results.

selected set of active domains, which resembles a grid. This grid-pattern cannot be eliminated by randomly moving the DD origin at the block-layer after each collective update (DTrDTr). We also observed, that DTr at device level and single-hit DTr at block level (DTrDTr) do not substantially improve the results obtained using DTrDT, which is why it was not investigated in further detail.

The grid can be eliminated by turning back to the DB scheme at block layer (DTrDB). Fig. 5 shows autocorrelation data for surface heights from DTrDB simulations using TC=1,1 and TC=2,2. Here, the autocorrelation functions are in perfect agreement. Panel (b) shows local slope analyses for autocorrelation functions with waiting times $s = 100$ and 1000 from these simulations, which also agree almost perfectly. The data presented for DTrDB is taken from production runs, which are further analyzed in the next section. The curves are much smoother because of the larger sample size afforded.

Fig. 5(a) also shows a curve from DTrDT simulations with TC=3,2 for comparison. A good agreement cannot be denied, suggesting, that at this block-layer domain size, the DTrDT simulations are sufficiently converged with respect to the autocorrelation of heights.

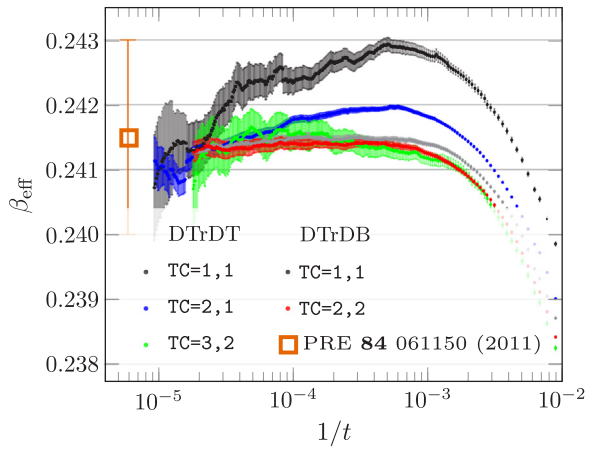

Fig. 6. Effective exponents for selected DD configurations. System sizes are $L_{\mathrm{DTrDT,TC=2,1}} = 2^{17}$ others $L = 2^{16}$. Sample sizes are: $n_{\mathrm{DTrDT,TC=1,1}} \geq 85$, $n_{\mathrm{DTrDT,TC=2,1}} \geq 396$, $n_{\mathrm{DTrDT,TC=3,2}} \geq 89$, $n_{\mathrm{DTrDB,TC=1,1}} \geq 1107$ and $n_{\mathrm{DTrDB,TC=2,2}} \geq 708$. Propagated 1σ error bars are attached to the effective exponents. The orange square shows the asymptotic exponent $\beta = 0.2415(15)$ obtained in Ref. [19].

4.2. Corrections to scaling

The roughness of a KPZ surface is expected to grow with a power law. The asymptotic growth exponent in $2 + 1$ dimensions is only known numerically.

Eq. (4) allows for a non-universal, constant prefactor to the growth-law. Such model-dependent parameters are affected by DD. One aspect we wish to point out is, that the size of decomposition domains influences the amplitude of the noise parameter η in Eq. (2), due to the variance of the rate at which each site is selected for updates decreasing with the size of the smallest decomposition domains. In other words, the system is sampled more smoothly with fine DD, which slightly inhibits the roughening of the surface, even when updates are uncorrelated, leading to a smaller prefactor to the growth law for finer DD. As a consequence, data from simulations with different DD cannot be averaged. Furthermore, in studies where absolute values of the roughness are relevant, such as finite-size scaling, the coarseness of the DD must be kept fixed.

Fig. 6 shows effective scaling exponents for a selected set of DD configurations. All curves suggest about the same asymptotic value β, but they differ at finite times, i.e. in corrections to scaling. Because only the asymptotic scaling exponent β is a universal property, while $\beta_{\mathrm{eff}}(t)$ depends on the model for finite times, this does not mark the result of any of these simulations wrong. However, the effective exponents from DTrDB runs show a plateau over almost two decades, suggesting that corrections at late times are very small.

The figure also shows the asymptotic value $\beta = 0.2415(15)$ from Ref. [19], which is based on GPU simulations using cDB at device and single-hit DB at block level. This value agrees well with the data from DTr implementations despite the presence of correlations.

4.3. Performance

The performance of the two most interesting implementations for future applications on NVIDIA Kepler-generation GPUs is compared in Fig. 7. Both use DTr at device-layer and differ in the type of block-layer DD employed: DT and DB. On a GTX Titan Black GPU, the performance drops after the first ∼100 MCS, because the device clocks down under load. This leads to a measured sustained performance which is lower than the peak. The performance on

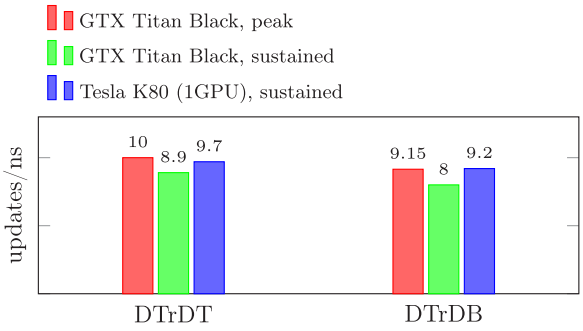

Fig. 7. Performance of the DTrDT and DTrDB variations of the RS implementation of the octahedron model on GPU. For K80, the sustained performance equals the peak performance. Benchmarks were performed for systems of lateral size 2^{16}.

a Tesla K80 is about constant. The DTrDB variant is consistently slower, by about ten percent.

A sequential implementation on an Intel i7-4930k CPU delivers 0.055 update attempts per ns for a system of lateral size $L = 2^{12}$, a size where the whole system fits into the L3 cache. The performance is less for larger systems, which do not fit into L3 cache. Running multiple independent runs on the same device is also likely to reduce performance. A parallel implementation, using DTr, running twelve threads, performs 0.28 update attempts per ns. This leads to a speedup factor of about 30 for the GPU code over a single-socket CPU.

5. Summary and conclusions

In this work we compared GPU implementations of the octahedron model for surface growth using different DD approaches. We were able to identify and eliminate the cause of correlations in a previously published GPU simulation [28]. Our solution applies the DT DD with random origin (DTr) scheme at device layer, which can be expected to perform well in simulations for any coarse decomposition with full sweeps between synchronizations. Apart from an obvious reduction in the site-selection noise, the results are indistinguishable from sequential runs on CPU up to the accuracy these can provide.

To go beyond this precision, we checked for artifacts due to fine DD at block layer by testing the self-consistency of GPU simulation results. We found, that using single-hit dead border (DB) at block level gives self-consistent results for both autocorrelation and roughness growth. The differences at finite times, we saw when using single-hit double tiling (DT) for this DD layer, do not allow a firm conclusion that universal properties could be affected. However, it is advantageous to exclude any finite-time corrections, which are artifacts of the simulation method applied. For these reasons we conclude, that the combined DTrDB scheme is the best one to be used in similar lattice MC studies. Our implementation of the octahedron model using this scheme delivers about nine update attempts per ns on high-end Kepler-generation NVIDIA GPUs.

The code used in this work, together with code used in [12,13] can be found at https://github.com/jkelling/CudaKpz.

Acknowledgments

Support from the Hungarian research fund OTKA (Grant No. K109577), the Initiative and Networking Fund of the Helmholtz Association via the W2/W3 Programme (W2/W3-026) and the International Helmholtz Research School NanoNet (VH-KO-606) is acknowledged. We gratefully acknowledge computational resources provided by the HZDR computing center, NIIF Hungary and the Center for Information Services and High Performance Computing (ZIH) at TU Dresden.

References

[1] Géza Ódor, Rev. Modern Phys. 76 (2004) 663–724.
[2] Joachim Krug, Adv. Phys. 46 (2) (1997) 139–282.
[3] S. Majumder, S.K. Das, Phys. Rev. E 81 (5) (2010) 050102.
[4] Matjaž Perc, Eur. J. Phys. 38 (4) (2017) 045801.
[5] Robert H. Swendsen, Jian-Sheng Wang, Phys. Rev. Lett. 58 (1987) 86–88.
[6] U. Wolff, Phys. Rev. Lett. 62 (1989) 361.
[7] N. Rajewsky, L. Santen, A. Schadschneider, M. Schreckenberg, J. Stat. Phys. 92 (1–2) (1998) 151–194.
[8] Stephen Wolfram, Rev. Modern Phys. 55 (1983) 601–644.
[9] Henrik Schulz, Géza Ódor, Gergely Ódor, Máté Ferenc Nagy, Comput. Phys. Comm. 182 (7) (2011) 1467–1476.
[10] Martin Weigel, J. Comput. Phys. 231 (8) (2012) 3064–3082.
[11] Andrea Pagnani, Giorgio Parisi, Phys. Rev. E 92 (2015) 010101.
[12] Jeffrey Kelling, Géza Ódor, Sibylle Gemming, 2016 IEEE International Conference on Intelligent Engineering Systems, 2016. INES '16, IEEE, 2016.
[13] Jeffrey Kelling, Geza Odor, Sibylle Gemming, Dynamical universality classes of simple growth and lattice gas models, ArXiv e-prints, January 2017.
[14] N. Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, Edward Teller, J. Chem. Phys. 21 (1953) 1087–1092.
[15] Michael Plischke, Zoltán Rácz, Phys. Rev. Lett. 53 (1984) 415–418.
[16] Paul Meakin, P. Ramanlal, L.M. Sander, R.C. Ball, Phys. Rev. A 34 (1986) 5091–5103.
[17] Jin Min Kim, J.M. Kosterlitz, Phys. Rev. Lett. 62 (1989) 2289–2292.
[18] Géza Ódor, B. Liedke, K.-H. Heinig, Phys. Rev. E 79 (021125) (2009) 021125.
[19] Jeffrey Kelling, Géza Ódor, Phys. Rev. E 84 (2011) 061150.
[20] Jeffrey Kelling, Géza Ódor, Máté Ferenc Nagy, H. Schulz, K. Heinig, Eur. Phys. J.: Spec. Top. 210 (2012) 175–187. http://dx.doi.org/10.1140/epjst/e2012-01645-8.
[21] Mehran Kardar, Giorgio Parisi, Yi-Cheng Zhang, Phys. Rev. Lett. 56 (1986) 889–892.
[22] K. Kawasaki, Phys. Rev. 145 (1) (1966) 224–230.
[23] F. Family, T. Vicsek, J. Phys. A 18 (2) (1985) L75.
[24] Jeffrey Kelling, Géza Ódor, Sibylle Gemming, Phys. Rev. E 94 (2016) 022107.
[25] Timothy Halpin-Healy, Phys. Rev. E 88 (2013) 042118.
[26] Bruce M. Forrest, Lei-Han Tang, Phys. Rev. Lett. 64 (1990) 1405–1408.
[27] NVIDIA, NVIDIA CUDA Programming Guide, 8.0 edition, 2016.
[28] Géza Ódor, Jeffrey Kelling, Sibylle Gemming, Phys. Rev. E 89 (2014) 032146.
[29] M.E.J. Newman, G.T. Barkema, Monte Carlo Methods in Statistical Physics, 2002 ed., Oxford University Press, 1999.
[30] Y. Shim, J.G. Amar, Phys. Rev. B 71 (12) (2005) 125432.
[31] Joshua A. Anderson, Eric Jankowski, Thomas L. Grubb, Michael Engel, Sharon C. Glotzer, J. Comput. Phys. 254 (2013) 27–38.