Parallel stochastic methods for PDE based grid generation

Alexander Bihlo† and Ronald D. Haynes‡

†*Department of Mathematics and Statistics, McGill University, 805 Sherbrooke W.,*
Montréal (QC), H3A 2K6, Canada
 E-mail*: abihlo@math.mcgill.ca*

‡*Department of Mathematics and Statistics, Memorial University of Newfoundland,*
St. John's (NL), A1C 5S7, Canada
 E-mail*: rhaynes@mun.ca*

The efficient generation of meshes is an important step in the numerical solution of various problems in physics and engineering. Here we are interested in situations where global mesh quality and coupling to the physical solution is important. Hence we consider PDE based mesh generation and present a method for the construction of adaptive meshes in two spatial dimensions using domain decomposition that is suitable for an implementation on parallel computing architectures. The method uses the stochastic representation of the exact solution of a linear mesh generator of Winslow type to find the points of the adaptive mesh along the sub-domain interfaces. The meshes over the single sub-domains can then be obtained completely independently of each other using the probabilistically computed solutions along the interfaces as boundary conditions for the linear mesh generator. Further improvements through the use of interpolation along the sub-domain interfaces and smoothing of mesh candidates are discussed. Various examples of meshes constructed using this stochastic–deterministic domain decomposition technique are shown and compared to the respective single domain solutions using a representative mesh quality measure.

1 Introduction

Grid adaptation is seen as an integral component of the numerical solution of many partial differential equations (PDEs). Here we are interested in the calculation of an adaptive grid automatically tuned to the underlying solution behaviour. The grid is found by solving a mesh PDE which is (often) coupled to the physical PDE of interest. Recently this general approach has shown great promise, solving problems in meteorology [3], relativistic magnetohydrodynamics [10], groundwater flow [14], semiconductor devices [27], and viscoelastic flows [28], to name just a few. A thorough, recent overview of PDE based mesh generation may be found in [12]. Such grid calculations potentially add a burdensome overhead to the solution of the physical model. Here we present an efficient, parallel strategy for the solution of the mesh PDE based on a stochastic domain decomposition method recently proposed by Acebrón et al. [1]. In many situations it is mesh quality, not an extremely accurate solution of the mesh PDE, which is important. It is for this reason the stochastic domain decomposition approach is a viable alternative for mesh generation.

PDE based mesh generation may be classified by the type of PDE, elliptic or parabolic, linear or nonlinear, to be solved for the mesh. The *equipotential* method of mesh generation in 2D was first presented, to the best of our knowledge, by Crowley [5]. The mesh lines in the physical co-ordinates x and y are the level curves of the potentials ξ and η satisfying Laplace's equations

$$\nabla^2 \xi = 0, \qquad \nabla^2 \eta = 0, \tag{1}$$

and appropriate boundary conditions which ensure grid lines lie along the boundary of the domain. The mesh transformation, giving the physical co-ordinates $x(\xi, \eta)$ and $y(\xi, \eta)$ in the physical domain Ω_p, can be found either by (inverse) interpolation of the solution of (1) onto a (say) uniform (ξ, η) grid or as Winslow [25] showed, by directly solving the inverse equations of (1), namely

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = 0, \quad \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = 0, \tag{2a}$$

where

$$\alpha = x_\eta^2 + y_\eta^2, \quad \beta = x_\xi x_\eta + y_\xi y_\eta \quad \text{and} \quad \gamma = x_\xi^2 + y_\xi^2. \tag{2b}$$

System (2) can be solved directly for the mesh transformations $x(\xi, \eta)$ and $y(\xi, \eta)$ using a uniform grid for the variables ξ and η belonging to the artificial computational domain Ω_c.

In this paper, we will exclusively work with mesh generators in the physical space, i.e. yielding $\xi = \xi(x, y)$ and $\eta = \eta(x, y)$. As mentioned, a convenient way to invert the mesh transformation from the physical space Ω_p to the computational space Ω_c is through *interpolation*. That is, once the numerical solution yielding $\xi_{ij} = \xi(x_i, y_j)$, $\eta_{ij} = \eta(x_i, y_j)$ is obtained one obtains the values for $x_{ij} = x(\xi_i, \eta_j)$, $y_{ij} = y(\xi_i, \eta_j)$ from two-dimensional interpolation. An alternative would be to numerically solve the hodograph transformations

$$\begin{pmatrix} x_\xi & x_\eta \\ y_\xi & y_\eta \end{pmatrix} = \frac{1}{J} \begin{pmatrix} \eta_y & -\xi_y \\ -\eta_x & \xi_x \end{pmatrix},$$

where $J = \xi_x \eta_y - \xi_y \eta_x$. As this leads to another system of PDEs, we choose the interpolation method to obtain the physical mesh lines. In practice, this inversion to the physical co-ordinates is not necessary. Instead one could transform the physical PDE of interest into the computational co-ordinate system.

Dvinksy [6] discusses the existence and uniqueness of such mesh transformations in the context of one–to–one harmonic maps. He demonstrates that the solutions to (1), and hence (2), will be well–defined if the (ξ, η) domain is convex. Since we construct this domain ourselves this condition can always be satisfied.

Godunov and Prokopov [8], Thompson et al. [22,23] and Anderson [2], for example, add terms to (1) and (2) to better control the mesh distribution and quality. Winslow [26] generalizes (1) by adding a diffusion coefficient $D(x, y) > 0$ depending on the gradient or other aspects of the solution. This gives the linear elliptic mesh generator

$$\nabla \cdot (D\nabla \xi) = 0 \quad \text{and} \quad \nabla \cdot (D\nabla \eta) = 0, \tag{3}$$

of interest in this paper. The diffusion coefficient D characterizes regions where additional mesh resolution is needed. An example for a mesh in both computational and physical co–ordinates is depicted in Figure 2 on page 7. This figure shows that the grids in the computational and physical co–ordinates behave inversely; the areas of grid concentration of the mesh in the computational co–ordinates coincide with the areas of de-concentration in the physical space and vice versa.

There has been recent work to parallelize nonlinear PDE based mesh generation using on Schwarz based domain decomposition approaches. In [7], Haynes and Gander propose and analyze classical, optimal and optimized Schwarz methods in one spatial dimension. A numerical

study of classical and optimized Schwarz domain decomposition for 2D nonlinear mesh generation has been presented in [9]. Recently, in [4], a *monolithic* domain decomposition method, simultaneously solving a mesh generator similar to (4) coupled to the physical PDE, was presented for a shape optimization problem. The authors used a overlapping domain decomposition approach to solve the coupled problem. Here we focus on the linear mesh generation problem only, detailing the effect of our stochastic domain decomposition method on the generated meshes.

This paper is organized as follows. In Section 2 we review the necessary background material on the stochastic interpretation of solutions of linear elliptic boundary value problems and their relation to numerical grid generators. We also briefly discuss the techniques required to achieve such stochastic solutions numerically. We explain how to couple the stochastic solution of linear elliptic mesh generators with a domain decomposition approach to obtain a scalable version of the algorithm. Section 3 illustrates our parallel grid generation strategy. The effect of smoothing the probabilistically computed interface solutions and the sub-domain solutions is demonstrated. Section 4 is devoted to further examples of grids computed using stochastic domain decomposition. Section 5 contains the conclusions of the paper as well as thoughts for further research directions.

2 Winslow mesh generation using a stochastic domain decomposition method

In this section, following [1], we describe how to generate adaptive meshes by solving Winslow's mesh generator (3) using a non–overlapping domain decomposition and a stochastic representation of the solution along the artificial interfaces.

2.1 Background

For concreteness, we will consider a two-dimensional linear grid generator of Winslow type

$$
-\nabla \cdot \left(\frac{1}{w} \nabla \xi \right) = 0, \quad -\nabla \cdot \left(\frac{1}{w} \nabla \eta \right) = 0,
$$
$$
\xi|_{\partial\Omega_p} = f(x,y), \quad \eta|_{\partial\Omega_p} = g(x,y),
$$
(4)

where $w = w(x,y) > 0$ is a strictly positive weight function [12, 25]. This grid generator finds the stationary solution of spatially dependent diffusion processes, yielding the computational coordinates $\xi = \xi(x,y)$ and $\eta = \eta(x,y)$ in terms of the physical coordinates x and y. For the sake of simplicity, we restrict ourselves to the case of rectangular physical and computational domains. As mentioned, Dvinksy [6], then guarantees a well–posed mesh generation problem.

Equations (4) form a system of two decoupled linear elliptic PDEs. It is well known that for such boundary value problems the solution can be written using methods of stochastic calculus [16, 18]. In the present case, this solution is conveniently derived from the expanded form of system (4), which reads

$$
-\frac{1}{w}\nabla w \cdot \nabla \xi + \nabla^2 \xi = 0, \quad -\frac{1}{w}\nabla w \cdot \nabla \eta + \nabla^2 \eta = 0.
$$
(5)

The solution of system (4) has the stochastic representation

$$
\xi(x,y) = \mathrm{E}[f(\mathbf{X}(\tau))], \quad \eta(x,y) = \mathrm{E}[g(\mathbf{X}(\tau))],
$$
(6a)

where $\mathbf{X}(t) = (x(t), y(t))^{\mathrm{T}}$ satisfies, in the Îto sense, the stochastic differential equation

$$d\mathbf{X}(t) = -\frac{1}{w}\nabla w\,dt + d\mathbf{W}(t). \tag{6b}$$

In the stochastic solution (6), $\mathrm{E}[\cdot]$ denotes the expected value, τ is the time when the stochastic path starting at the point (x, y) first hits the boundary of the physical domain Ω_{p} and $\mathbf{W}$ is the standard two-dimensional Brownian motion [1, 16, 18].

As noted in [1], seeking the numerical solution of system (4) using the probabilistic solution at *all* the grid points of interest is generally too expensive, especially when compared to direct or iterative (deterministic) methods to solve the linear system of equations resulting from the discretization of (4). Efficiency is indeed a crucial factor for grid adaptation, since the grid is computed in addition to the solution of the physical PDE. This is particularly important for time-dependent problems where (4) has to be solved in combination with a system of PDEs at every time step.

To address the expense of the stochastic approach, the key idea put forward in [1] is to use the probabilistic solution (6) in the context of domain decomposition. The probabilistic solution is used only to obtain the boundary conditions at the sub-domain interfaces, see Figure 1.
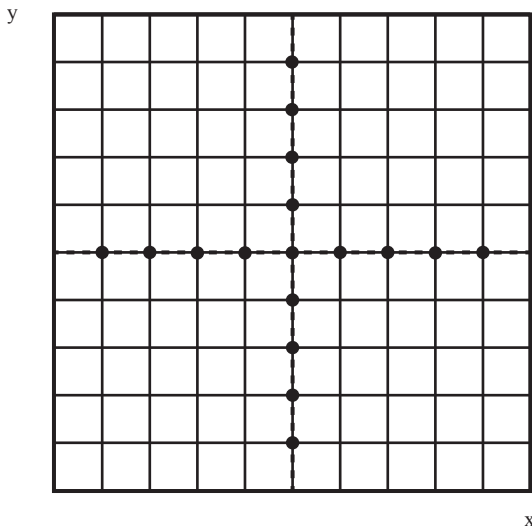

Figure 1. Domain decomposition using the stochastic approach. The probabilistic solution (6) of the linear mesh generator (4) is used to compute the exact solution $\xi(x, y)$, $\eta(x, y)$ at the interface points (solid dots). The solution on the sub-domains is computed in parallel using a suitable single domain solver with the interface solutions serving as Dirichlet boundary values.

This approach to domain decomposition has the advantage of being fully parallelizable, as each sub-domain can be assigned to a single core and thus the grids at the various sub-domains can be computed completely independently of each other. Once the interface values are obtained with sufficient accuracy the solutions on the sub-domains are only computed once — no iteration is needed in this domain decomposition approach. In addition, the two-dimensional random walks required to solve the stochastic differential equations (6b) at each interface point can be done independently and efficiently using, for example, an implementation on a graphics processing unit (GPU). We stress here that several Monte-Carlo techniques have already been successfully realized on GPUs, e.g. for problems in molecular dynamics, statistical physics and financial mathematics (see [20, 24] for examples), leading to a massive speed-up (often several orders of magnitude) compared to the conventional CPU solution. The perspective of compati-

bility with the principles of GPU programming is therefore a highly desirable feature of a new numerical algorithm.

2.2 Implementation details

We now describe the solution procedure for solving the system of stochastic differential equations (6b). This system of equations is particularly simple in that it is decoupled, i.e. the equations for $x(t)$ and $y(t)$ can be solved independently of each other and hence this solution is fully parallelizable. The components of the two-dimensional Brownian motion $\mathrm{d}\mathbf{W}$ can be realized as $\mathrm{d}W_i = 2\sqrt{\Delta t}\,N(0,1)$, for $i = 1, 2$, where $N(0,1)$ is a normally distributed random number with zero mean and variance one [16]. A simple method for integrating the stochastic differential equations (6b) is to use the Euler–Maruyama method [17] with a time step Δt.

The crucial part of solving (6) is the correct determination of the *exit time* τ, i.e. the time when the stochastic process starting at (x, y) first hits one of the boundaries of Ω_p. One of the simplest possibilities to estimate the exit time is to integrate (6b) using the Euler–Maruyama method and to determine whether the process leaves the domain at the end of the time step. The problem with this approach is that one cannot determine whether the true stochastic process already exited the domain during the time step. It is possible that the true process started within the domain, left it, but returned to the domain before the end of the time step. This problem is inherent in all linear time stepping methods. Indeed, we found the Euler–Maruyama method led to unacceptably large errors in the fully probabilistically generated grid lines close to the boundaries.

To address this, we implemented the approach proposed in [15], based on *exponential time-stepping*. Unlike the Euler–Maruyama scheme, which uses a constant time step Δt, this approaches chooses Δt as an exponentially distributed random variable. That is, each time step is determined as an independent realization of a distribution with the probability density function $\lambda \exp(-\lambda t)$, where $\lambda > 0$ is a constant parameter. The expected value of an exponentially distributed random variable X is then $\mathrm{E}[X] = 1/\lambda$, i.e. as $\lambda \to \infty$, $\Delta t \to 0$.

The central idea behind exponential time-stepping is that it is possible to carry out the boundary test for the exit time by both explicitly checking if the process has exited the domain and also using the conditional probability that the process reached a boundary between $\mathbf{X}(t)$ and $\mathbf{X}(t+\delta t)$ [15]. Exponential time-stepping works because this conditional probability is a function of the boundary itself. It was shown in [15] that for several classes of stochastic differential equations this conditional probability can be either computed analytically or approximated numerically with high efficiency. System (6b) constitutes a process of the *additive noise* class — a deterministic system (the first term in (6b)) that is superimposed by constant noise (the second term in (6b)). For such classes of problems, the precise implementation of the exponential time-stepping method is presented in Section 4.2 of [15]. The use of exponential time-stepping leads to meshes that are sufficiently accurate throughout the whole domain. In particular, unlike the Euler–Maruyama results, no problem occurs in the grid lines near the boundaries.

The system of stochastic differential equations (6b) is solved many times, as one needs enough samples to approximate the expected value in (6a) through the arithmetic mean. Indeed the main problem with Monte-Carlo techniques is the rather slow convergence rate. The error in estimating the expected value with the sample mean, using pseudo–random numbers, is known to be of the order of $N^{-1/2}$, where N is the number of samples, see e.g. [21].

The main problem with pseudo-random numbers is that a significant fraction of them tend to cluster in localized regions of the sampling space, while other parts of the domain are under-

5

sampled. This variation in the point density is measured through the *discrepancy*. Sequences of numbers that fill the sampling space while reducing the discrepancy are called quasi-random numbers. They can lead to a significant speed-up of the Monte-Carlo method, yielding convergence rates of order N^{-1} [21]. The downside of quasi-random numbers is that they cannot be generated in a fully parallel way. Due to their inherent correlation it is necessary to first generate a sequence and then re-order the single elements of this sequence to break the correlation amongst the elements. This procedure is not a parallel operation.

Despite the superior convergence properties of Monte-Carlo techniques employing quasi-random numbers, we choose to use pseudo–random numbers in what follows. Given the relative low accuracy needed when solving the mesh PDE, we find that the use of a moderate number of Monte-Carlo simulations ($N \approx 10^4$) with pseudo–random numbers gives sufficiently accurate meshes. Moreover, as shown in the next section, applying a smoothing procedure to the computed mesh or simply using Monte-Carlo solutions only along the subdomain interfaces has the potential to further reduce the number of Monte-Carlo simulations needed to obtain a good quality mesh. At the same time, avoiding the quasi-random numbers prevents the introduction of a bottleneck in the otherwise fully parallel algorithm of stochastic domain decomposition for grid generation.

3   A numerical case study

In this section we illustrate the approach described in the previous section, taking care to detail the effect of each algorithmic choice on the resulting mesh. We illustrate the successive stages of our implementation by solving the Winslow mesh generator (4) with $w = 1/\rho$ where ρ is the monitor function

$$\rho = 1 + R \exp(-50(x - 3/4)^2 - 50(y - 1/2)^2 - 1/2). \tag{7}$$

In all examples we choose $R = 15$. In the physical co-ordinates, this monitor function will concentrate the mesh near $x = 3/4$ and $y = 1/2$. It is instructive to note that due to the relation $w = 1/\rho$, the region where ρ attains its maximal values is where the mesh in the *physical space* is the sparsest. We begin by showing the mesh obtained by solving (4) on single domain, displaying the mesh in both computational and physical co-ordinates. The entire mesh is then recomputed using the stochastic method. The effect of the number of Monte–Carlo simulations is shown. We then compute the mesh stochastically only along the artifical interfaces; the rest of the mesh is computed using a domain decomposition approach with deterministic sub–domain solves. We finally show how smoothing can be incorporated to obtain quality meshes while keeping the number of Monte–Carlo simulations small.

3.1   Mesh quality measures

The effects of the different steps of the proposed algorithm on the meshes can be quantified by introducing a *mesh quality measure*. Several mesh quality measures have been proposed in the literature to quantify properties of adaptive meshes. These measures usually assess mesh regularity, the degree of adaptivity to the numerical solution, the regularity of mesh elements in an appropriate metric (defined through the monitor function) or they quantify equidistribution [12].

In the present case, we restrict ourselves to a comparison of the mesh quality of the domain decomposition solution with the mesh quality of the single domain mesh. We do not aim to

evaluate the absolute mesh quality of the single domain solution of the linear mesh generator (4) here, but rather we are interested in estimating how well the domain decomposition solution approximates the quality of the single domain grid. This is an important task because a mesh quality measure can yield a *stopping criteria* for the probabilistic algorithm as we are not concerned with finding a perfect numerical solution of the mesh generator (4).

For the sake of simplicity, we will exclusively work with the *geometric mesh quality measure* defined by

$$Q(K) = \frac{1}{2} \frac{\mathrm{tr}(J^{\mathrm{T}}J)}{\sqrt{\det(J^{\mathrm{T}}J)}}, \tag{8}$$

where J is the Jacobian of the transformation $x = x(\xi, \eta)$, $y = y(\xi, \eta)$ and K is a mesh element in Ω_{c}, see [12] for more details. This measure has the property $Q(K) \geq 1$ with $Q(K) = 1$ for an equilateral mesh element only. Probabilistically computed meshes that have not yet converged usually show several kinks in mesh lines and thus feature grid cells that are far away from equilateral as compared to the meshes obtained from a deterministic algorithm. Hence $Q(K)$ is an appropriate measure for our purposes.

3.2 Single domain reference mesh

We first solve system (4) with monitor function (7) on the entire domain $\Omega_{\mathrm{p}} = [0, 1] \times [0, 1]$. This mesh will serve as the reference solution for the stochastically generated meshes which follow.

System (4) is discretized with centered finite differences on a uniformly spaced grid in the physical (x, y) co–ordinates and the resulting system is solved using a Jacobi iteration. We use the Dirichlet boundary conditions $\xi(0, y) = 0$, $\xi(1, y) = 1$, $\eta(x, 0) = 0$ and $\eta(x, 1) = 1$. This ensures grid lines on the boundary of the unit square. The values of $\xi(x, 0)$, $\xi(x, 1)$, $\eta(0, y)$ and $\eta(1, y)$ are obtained from solving the one-dimensional forms of the mesh generator (4).
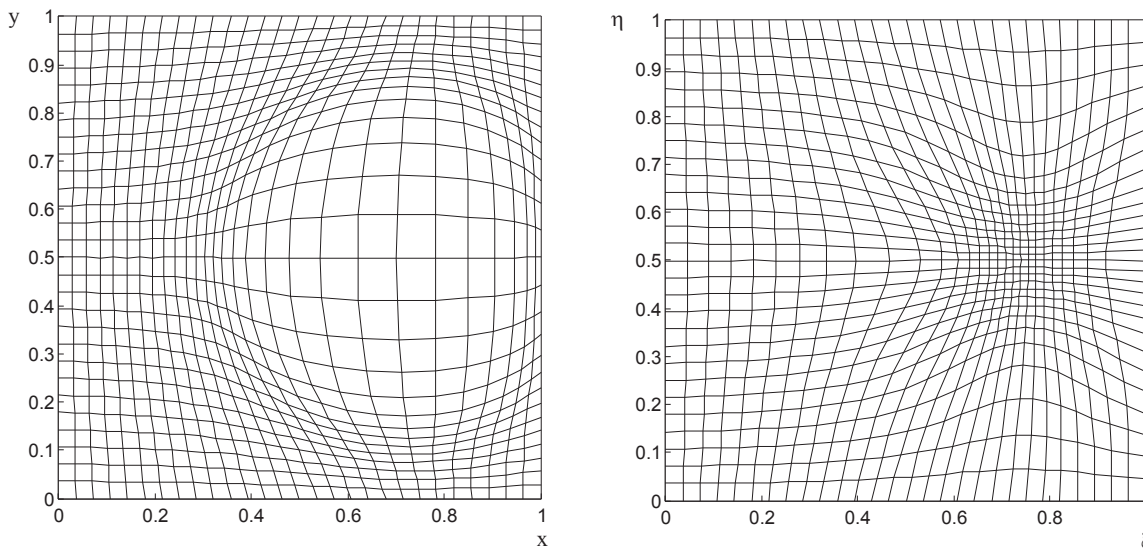

Figure 2. **Left:** Numerical solution of the Winslow-type grid generator (4) for the monitor function (7) over the physical space $\Omega_{\mathrm{p}} = [0, 1] \times [0, 1]$. **Right:** The physical mesh lines obtained from the grid on the left using linear interpolation.

Figure 2 (left) shows the numerical approximation to $\xi(x, y)$ and $\eta(x, y)$ obtained by solving (4) using the monitor function (7) on $\Omega_{\mathrm{p}} = [0, 1] \times [0, 1]$ with 29×29 uniformly chosen grid

points. On the right of Figure 2 we depict the physical mesh lines obtained by interpolation of the mesh on the left onto uniformly spaced (ξ, η) co–ordinates. Here, as expected, the regions of mesh concentration indeed coincide with the maximal values of ρ, near $x = 3/4$ and $y = 1/2$.

3.3 Probabilistically computed mesh

It is instructive to display the result that is achieved from solving the Winslow-type mesh generator (4) for *all* the grid points using the stochastic solution (6). The result of such fully probabilistically computed solutions is displayed in Figure 3.
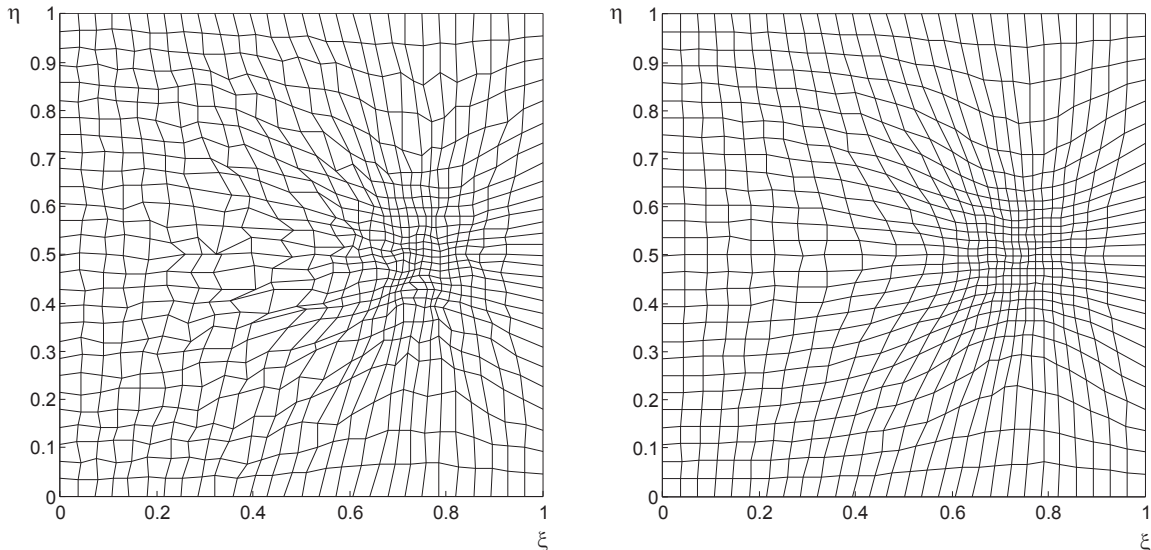

Figure 3. **Left:** Grid obtained from the probabilistic solution (6) using $\lambda = 1000$ and $N = 1000$ Monte-Carlo simulations at each point to estimate the expected value. **Right:** Same as left, but using $N = 10000$ Monte-Carlo simulations.

We choose the exponential time stepping parameter $\lambda = 1000$ for both runs. The difference between Figure 3 (left) and Figure 3 (right) is the number of Monte–Carlo simulations used. On the left only $N = 1000$ Monte-Carlo simulations were used to approximate the expected values in the solution (6a). On the right we use $N = 10000$ Monte-Carlo runs. Comparing to the reference solution on the right of Figure 2, the difference in the results is dramatic. This illustrates the well–known problem that a high number of simulations is needed to obtain the solution of a PDE with sufficient accuracy from a stochastic representation. As discussed in Section 2, the theoretical explanation for this finding is the slow convergence rate of Monte-Carlo methods, which is proportional to $N^{-1/2}$. It is mainly this slow convergence which has prevented the wider use of Monte-Carlo techniques in the numerical solution of PDEs. Here we suggest that the relative lower accuracy requirement for the mesh makes the stochastic formulation, coupled with the domain decomposition approach to follow, an appealing strategy.

To quantify the above findings, we compute the geometric mesh quality measure (8) for both the deterministic mesh and the two meshes found by the stochastic algorithm. As $Q = Q(K)$ is a function of each mesh element K, we only list the maximum and the mean of Q over the domain, denoted by $Q_{\max}$ and $Q_{\mean}$ for the deterministic solution and by $Q_{\max}^{S}$ and $Q_{\mean}^{S}$ for the stochastic solution. Comparisons of the stochastic solution and the deterministic solution are given by the ratios $R_{\max} = Q_{\max}/Q_{\max}^{S}$ and $R_{\mean} = Q_{\mean}/Q_{\mean}^{S}$. We also present l_{∞}, a measure of how well ρ is captured by the generated mehses. More precisely, we compute the

linear interpolant of ρ onto the grids obtained using the deterministic and the stochastic solver yielding ρ_{interp} and $\rho^{\text{S}}_{\text{interp}}$, respectively. The l_∞ error is then computed as the maximal difference between ρ_{interp} and $\rho^{\text{S}}_{\text{interp}}$, i.e. $l_\infty = \max(|\rho_{\text{interp}} - \rho^{\text{S}}_{\text{interp}}|_i)$, where i runs through all grid points. We have chosen this error measure as it allows us to compare the maximal deviation of the interpolated ρ on the stochastically generated mesh to interpolated ρ on the deterministically generated mesh, and hence provides an additional measure of convergence for the Monte-Carlo simulation.

Table 1. Mesh quality for grid adapting to monitor function (7). $\lambda = 1000$.

	N	l_∞	$Q^{\text{S}}_{\text{max}}$	$Q^{\text{S}}_{\text{mean}}$	R_{max}	R_{mean}
$Q_{\text{max}} = 1.8$	1000	2.23	3.85	1.22	0.47	0.95
$Q_{\text{mean}} = 1.16$	10000	1.33	1.9	1.18	0.95	0.98

It can be seen from Table 1 that the probabilistically computed mesh obtained with $N = 1000$ Monte-Carlo simulations yields relatively poor mesh quality measures when compared to the mesh obtained from the deterministic algorithm. If we increase the number of Monte-Carlo simulations to $N = 10000$ the mesh quality measures found are already reasonably close to the deterministic case.

3.4   Smoothing of the mesh

The problem with the stochastic representation of the solution of the linear mesh generator (6) is that usually a very large number of Monte-Carlo simulations is needed to obtain a reasonably accurate numerical approximation. On the other hand, such a highly accurate numerical solution might not be absolutely necessary to obtain a good quality mesh from the grid generator (4). Instead, we show that applying a smoothing operation to a lower accuracy solution of (4) can yield a high-quality grid.

A type of smoothing algorithm that appears particularly well-suited for this type of problem is *anisotropic diffusion*, also referred to as *Perona–Malik diffusion*,

$$
\begin{aligned}
\xi_t - \nabla \cdot (c_\xi \nabla \xi) = 0, & \quad c_\xi = \exp(-||\nabla \xi||^2/k^2), \\
\eta_t - \nabla \cdot (c_\eta \nabla \eta) = 0, & \quad c_\eta = \exp(-||\nabla \eta||^2/k^2),
\end{aligned}
\tag{9}
$$

where k is an arbitrary constant [19]. In image processing, smoothing of this kind is used to remove noise without significantly affecting the edges of an image. This smoothing operation is suitable for grids generated with the Monte-Carlo technique as the error in approximating the solution to (4) will appear as superimposed random noise over the required adapted mesh (the underlying signal). Anisotropic diffusion of the above form will remove this superimposed noise while still accurately preserving the regions of grid concentration.

We now apply anisotropic diffusion to the fully probabilistically computed mesh displayed in Figure 3 (left). For this purpose we discretize (9) with a forward in time and centered in space (FTCS) scheme. It is usually sufficient to numerically integrate (9) only a few steps starting with a grid computed using the Monte-Carlo technique to yield a smooth mesh. Consequently, diffusive smoothing is computationally a rather cheap operation. The result of integrating (9) to $t = 5\Delta t$ starting with the mesh shown on the left of Figure 3 using a FTCS discretization with $k = 1000$ and $\Delta t = 10^{-4}$ is shown in Figure 4.
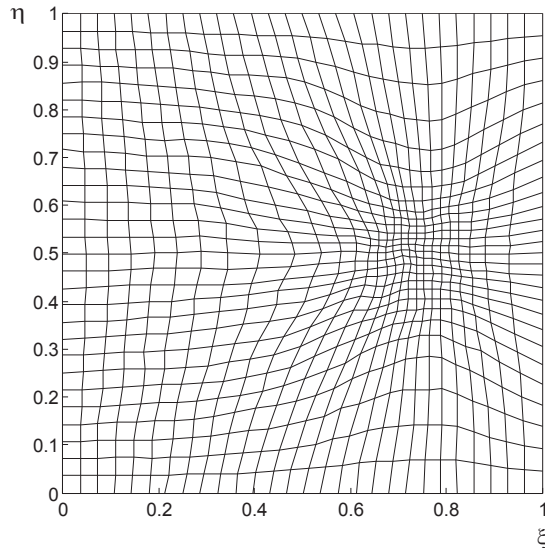
Figure 4. Grid obtained from the probabilistic solution (6) using $N = 1000$ Monte-Carlo simulations at each point to estimate the expected value. After the computation, we apply anisotropic diffusion of the form (9) to the mesh in the computational space. Eq. (9) was discretized using a FTCS scheme with $\Delta t = 10^{-4}$ and $k = 1000$. We integrated (9) up to $t = 5\Delta t$.

As can be seen by comparing with the single domain solution shown in the right of Figure 2, a few smoothing operations are able to almost completely eliminate the 'wiggles' that are typical of the approximate solution computed using Monte-Carlo techniques. At the same time, the grid is still properly concentrating near the maximum values of the monitor function (7), i.e., as expected, anisotropic diffusion properly preserves the signal in the Monte-Carlo solution.

To confirm these qualitative findings we once again compute the geometric mesh quality measure for the probabilistically computed and smoothed mesh. Here it is particularly instructive to monitor the mesh quality as a function of the number of total smoothing steps n (quantified through the final integration time $t = n\Delta t$). To have a proper comparison, we also smoothed the deterministic solution with the respective number of smoothing steps, which leads to the different absolute mesh quality measures for $Q_{\max}$ and Q_{mean} in Table 2.

Table 2. Mesh quality for grids adapting to monitor function (7), smoothed using (9) for $N = 1000$ and $\lambda = 1000$.

n	l_∞	$Q_{\max}$	Q_{mean}	$Q_{\max}^{\text{S}}$	$Q_{\text{mean}}^{\text{S}}$	$R_{\max}$	R_{mean}
1	2.25	1.77	1.16	2.45	1.2	0.71	0.97
2	1.83	1.77	1.16	2.44	1.2	0.72	0.97
5	1.72	1.66	1.15	1.83	1.18	0.91	0.98
10	1.19	1.54	1.15	1.57	1.16	0.98	0.99

3.5 Domain decomposition solution

Unless a large number of cores, equal to the total number of mesh points, is available, the stochastic solution of (4) at all points would still be quite expensive. In the context of domain decomposition we only evaluate the stochastic form of the solution at a few mesh points along the domain interfaces and then compute the solution deterministically at all the remaining grid points. As will be shown below, the meshes obtained from this stochastic–deterministic domain

10

decomposition technique are usually much smoother than the meshes shown in Figure 3. Thus, fewer (or no) smoothing sweeps will be required.

We now determine solution of the mesh generator (4) with the monitor function (7) using the domain decomposition technique outlined in Section 2. For the sake of simplicity we restrict ourselves to the case of four square sub-domains. The probabilistic solutions at the interfaces are computed using $\lambda = 10000$ and $N = 10000$.
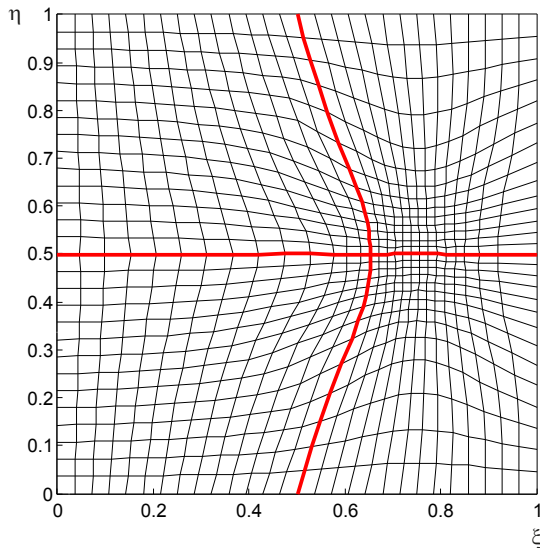

Figure 5. Domain decomposition solution for the mesh generator (4) with monitor function (7).

As can be seen from Figure 5 the stochastic–deterministic approach to solve (4) almost perfectly reproduces the single domain solution shown in Figure 2 (right). This finding is also confirmed with the geometric mesh quality measure, having values $Q_{\max}^{\mathrm{DD}} = 1.80$ and $Q_{\mathrm{mean}}^{\mathrm{DD}} = 1.16$ for the domain decomposition solution which give rise to the ratios $R_{\max} = 1$ and $R_{\mathrm{mean}} = 1$ when compared to the single domain solution.

No smoothing was applied to obtain the mesh shown in Figure 5. We should nevertheless like to stress that the smoothing operation proposed in Section 3.4, if needed, can be also achieved in parallel, as (9) can be applied on each sub-domain separately. That is, equation (9) yields a local smoothing operation and does not dilute the parallel nature of the proposed algorithm.

The only complication is how to smooth the sub-domain interfaces as they serve as natural (Dirichlet) boundaries for the smoothing operation (9). Ways to overcome this difficulty include a pre-smoothing of the interface to ensure that it is sufficiently accurate before being used as boundary conditions for the sub-domain solves or possibly apply a second smoothing cycle with shifted sub-domains that have the original interfaces in their interior. We will present an example for the first possibility in Section 4.

3.6   Interpolation along the interface

If the number of compute cores is limited, a promising approach to further reduce the cost of the probabilistic part of the domain decomposition algorithm is to avoid computing the solution stochastically at all the grid points along the interface. Instead we obtain the stochastic solution at only a few points and approximate the solution at the remaining points using interpolation. This was suggested in [1], however, we found the meshes obtained by this approach to be quite

sensitive to the location of these points.

A short study in this *optimal placement problem* is reported in Figure 6. In Figure 6 (left) we computed the probabilistic solution at seven equally spaced points along each of the dividing lines. Linear interpolation is used to obtain the remaining interface points. It is clearly visible that this procedure does not give the mesh obtained by domain decomposition in Figure 5. The problem is that local maxima and minima in the monitor function is missed and hence proper transitions between regions of grid concentration and de-concentration are not captured. We also tested more sophisticated interpolation strategies such as splines, cubic interpolation and Chebychev interpolation, but the results obtained are nearly identical. We conclude that, unlike the results reported in [1], the placement of points is crucial if interpolation is used to approximate the interface conditions for the domain decomposition for mesh generation.
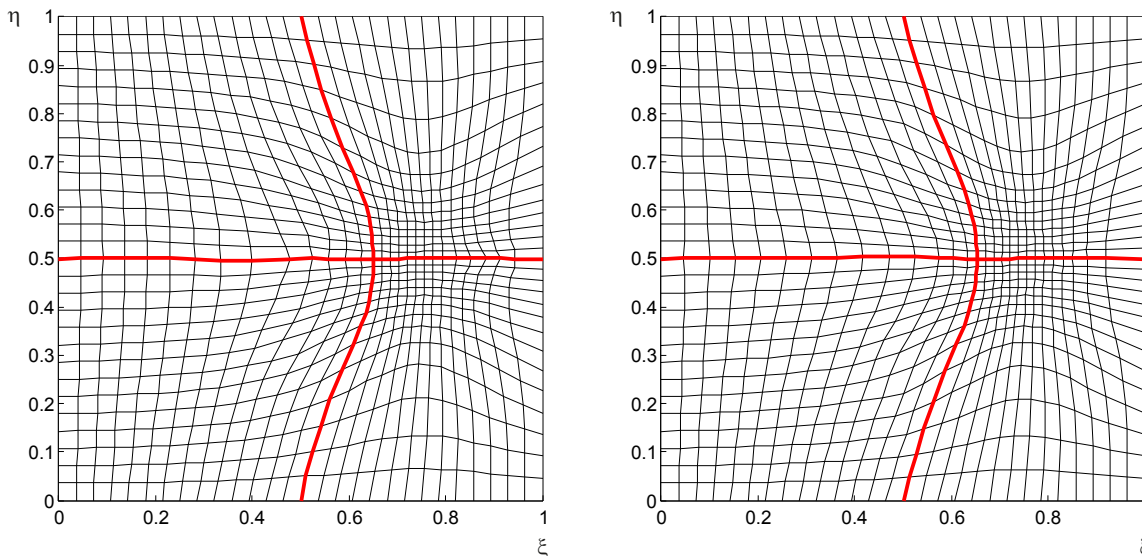

Figure 6. **Left:** Domain decomposition solution for the mesh generator (4) with monitor function (7). The solution was evaluated at seven equidistant points along each dividing line probabilistically. Linear interpolation was used to obtain the remaining interface points. **Right:** Same as left, but solution was evaluated probabilistically at the maxima and minima of the first and second derivatives of the monitor function at the interface. This gave seven points where the mesh was computed with Monte-Carlo simulations. Linear interpolation was used to obtain the remaining interface points.

In Figure 6 (right) we determined the points along the dividing lines where the solution is computed probabilistically based on the properties of the monitor function ρ. In particular, we required placement of the points near the maxima and minima of ρ_x and ρ_{xx} along the horizontal dividing line and ρ_y and ρ_{yy} along the vertical dividing line. This leads to a total of seven points along each dividing line where the solution has to be computed using the Monte-Carlo technique. Note that these points are not equally spaced as they were in the previous case. The remaining interface points are then approximated using linear interpolation.

The solution found using this rather simple placement criteria gives a mesh very close to the solution shown in Figure 5 but at only a fraction of the computational cost, or cores, required by the original stochastic domain decomposition solution. Smoothing of the mesh shown could be used to further improve the smoothness in the transition of the grid lines across the interfaces.

These qualitatively visible differences in Figure 6 are not clearly reflected in our chosen mesh quality measure, which yields essentially the same results ($R = 1$) for both equally and

'optimally' placed interfaced points. On the other hand, the l_∞ error found when using equally spaced points, $l_\infty = 4.91$, is more than five times as large as the l_∞ error obtained when we strategically place the points where the stochastic solution is obtained, $l_\infty = 0.92$. This shows a larger deviation from the single domain solution.

The one drawback of this placement approach is that it is not obvious apriori how to determine how many and at what locations the solution is best determined probabilistically. For large meshes in real-world applications, a possible trade-off would be to place the points near the most pronounced features of the monitor function only and smooth the resulting meshes over a few cycles.

4   Further examples

Having explained the technique and the issues involved, in this section we present additional examples to demonstrate the meshes generated using our stochastic domain decomposition algorithm on well known test problems.

We first choose the monitor function

$$\rho = \sqrt{1 + \alpha(R^2 \exp(R(x-1))^2 \sin(\pi y)^2 + (1 - \exp(R(x-1)))^2 \cos(\pi y)^2 \pi^2)} \qquad (10)$$

with $\alpha = 0.7$ and $R = 15$. This monitor function was used in [9,13] to generate meshes using a nonlinear mesh generator.
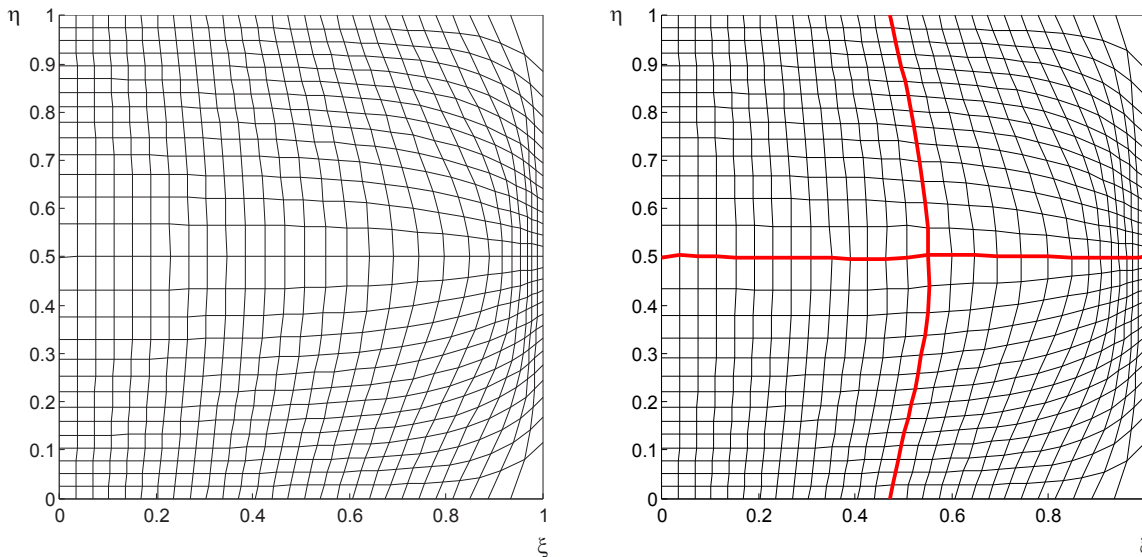

Figure 7. **Left:** Single domain solution of the mesh generator (4) using the monitor function (10). **Right:** The associated domain decomposition solution. The solution along the interfaces was computed using (6) with $N = 10000$ Monte-Carlo simulations and $\lambda = 10000$ as the parameter of the exponential distribution.

Figure 7 (left) displays the single domain mesh obtained by solving Winslow's generator (4) using a Jacobi iteration. The corresponding domain decomposition solution with four subdomains is depicted in Figure 7 (right). Here, we used $\lambda = 10000$ and $N = 10000$ to compute the solution probabilistically along all the points on the interfaces. No smoothing was applied to the final mesh.

The mesh quality measures for the domain decomposition solution are $Q_{\max}^{\mathrm{DD}} = 1.69$ and $Q_{\mathrm{mean}}^{\mathrm{DD}} = 1.14$, respectively, with corresponding ratios $R_{\max} = 0.99$ and $R_{\mathrm{mean}} = 1$ compared to

the single domain solution. That is, in terms of the geometric mesh quality measure, the single domain solution is almost perfectly approximated.

The next example uses the monitor function

$$\rho = \frac{1}{1 + \alpha \exp(-R(y - 1/2 - 1/4\sin(2\pi x))^2)} \tag{11}$$
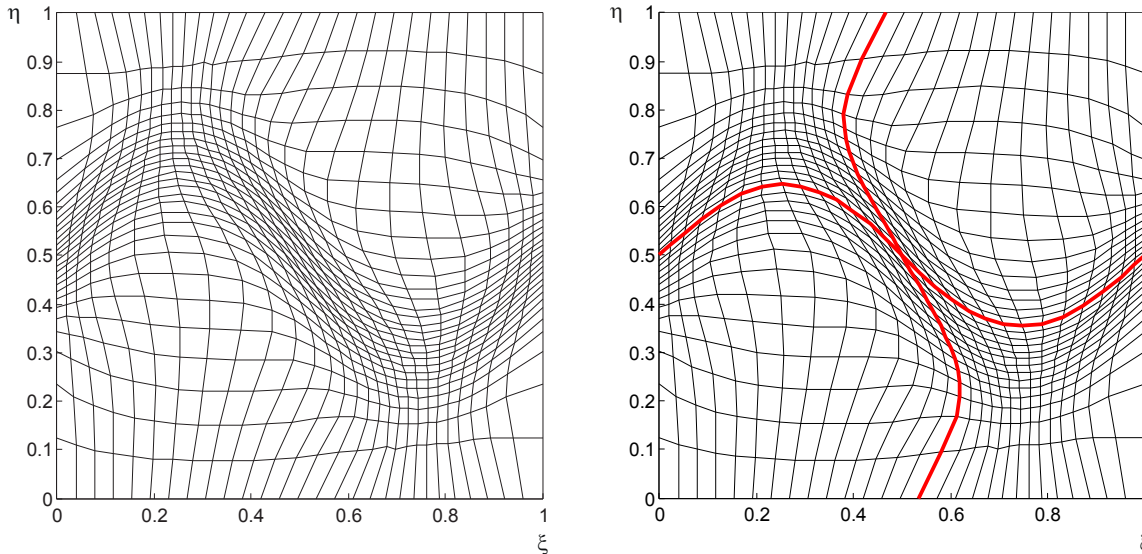
with $\alpha = 10$ and $R = 50$.

Figure 8. **Left:** Single domain solution of the mesh generator (4) using the monitor function (11). **Right:** The associated domain decomposition solution. The solution along the interfaces was computed using (6) with $N = 10000$ Monte-Carlo simulations and $\lambda = 10000$ as the parameter of the exponential distribution.

The single domain solution for this example is depicted in Figure 8 (left). The mesh reflects both the large scale features (the sinusoidal wave) and smaller scale variations (across the wave) in the monitor function. In Figure 8 (right) we show the domain decomposition solution using four sub-domains. The parameters for the stochastic solver again were $\lambda = 10000$ and $N = 10000$. We used the stochastic solver to compute all the points along the interfaces. No smoothing is applied to the final mesh. It can be seen in Figure 8 (right) that the domain decomposition solution has several kinks which are slightly more pronounced than in the single domain mesh.

We now present the results of four different versions of the domain decomposition solution in Figure 9 and Figure 10. In Figure 9 we contrast the effect of global smoothing (left) versus local smoothing on each of the sub-domains (right). As was indicated in Section 3 the global smoothing is a sequential operation whereas smoothing on the sub-domains can be carried out in parallel. The global smoothing operation leads to a mesh that varies very smoothly throughout the whole domain. The locally smoothed mesh is much improved as well (also compare the mesh quality measures reported in Tables 3 and 4) and has less kinks when compared to the original domain decomposition solution shown in Figure 8 (right). The remaining discontinuities could be further reduced by applying a second smoothing cycle over shifted sub-domains, i.e. by re-assigning the sub-domains in such a manner that the original interfaces lie within the new sub-domains and then re-apply the smoothing. Once again, this would not sacrifice the overall parallel nature of the proposed algorithm.
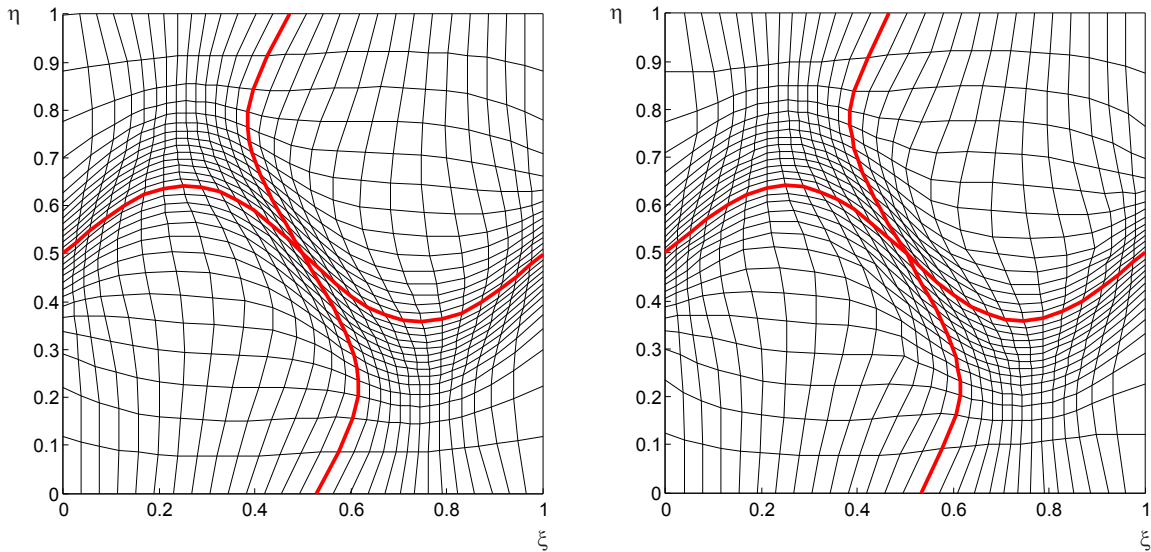
14

Figure 9. **Left:** Domain decomposition solution of the mesh generator (4) using the monitor function (11). Five smoothing cycles using (9) with $\Delta t = 10^{-4}$ and $k = 1000$ were applied on the whole domain. **Right:** Smoothing is applied on each of the sub-domains separately.
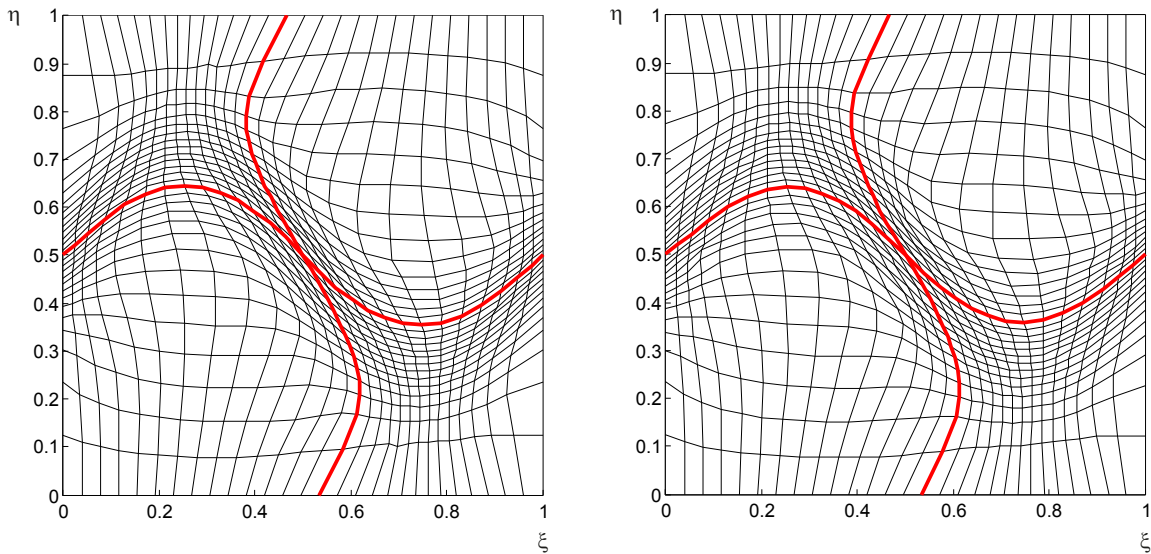

Figure 10. **Left:** Domain decomposition solution of the mesh generator (4) using the monitor function (11). The probabilistically computed interface solution was smoothed using weighted linear least squares and a second degree polynomial model. **Right:** Same as right, but additionally five smoothing cycles using (9) with $\Delta t = 10^{-4}$ and $k = 1000$ were applied on each of the sub-domains separately.

In Figure 10 we demonstrate the effect of pre-smoothing the stochastically computed interface solution. In Figure 10 (left), the values of ξ and η have been smoothed using weighted linear least squares and a second degree polynomial model (provided by the `Matlab` function `smooth`) before using the interface solution as the boundary values for the solver on each of the sub-domains. In general we see that only a few interface solutions are required; smoothing of the interface is much cheaper than smoothing the whole domain or the sub-domains. In Figure 10 (right), we demonstrate the effect of both smoothing the interface and applying the diffusive smoothing (9) on each of the sub-domains.

15

We now present the results for the geometric mesh quality measure for the different variants of the stochastic–deterministic domain decomposition algorithm discussed above. In particular, the geometric mesh quality measures for the grids computed using monitor function (11) are determined for two series of numerical experiments. In Table 3, we keep the number of Monte-Carlo runs constant, $N = 10000$, and vary the parameter of the exponential distribution λ; in Table 4 we vary N and keep $\lambda = 5000$ constant.

Table 3. Mesh quality for grids adapting to monitor function (11) with $N = 10000$ Monte–Carlo simulations.

	λ	l_∞	$Q_{\max}^{\mathrm{DD}}$	$Q_{\mathrm{mean}}^{\mathrm{DD}}$	$R_{\max}$	R_{mean}
No smoothing $Q_{\max} = 3.86$ $Q_{\mathrm{mean}} = 1.67$	1000	1.18	4.44	1.76	0.87	0.95
	2000	0.91	4.29	1.72	0.90	0.97
	5000	0.49	3.98	1.7	0.97	0.98
	10000	0.4	3.9	1.69	0.99	0.99
Global smoothing $Q_{\max} = 3.6$ $Q_{\mathrm{mean}} = 1.62$	1000	1.04	3.96	1.71	0.91	0.95
	2000	0.64	3.79	1.67	0.95	0.97
	5000	0.45	3.67	1.64	0.98	0.99
	10000	0.44	3.67	1.62	0.98	1
Sub-domain smoothing $Q_{\max} = 3.6$ $Q_{\mathrm{mean}} = 1.62$	1000	1.36	4.24	1.69	0.85	0.96
	2000	0.92	3.83	1.65	0.94	0.98
	5000	0.75	3.79	1.62	0.95	1
	10000	0.74	3.67	1.62	0.98	1
Interface smoothing $Q_{\max} = 3.86$ $Q_{\mathrm{mean}} = 1.67$	1000	1.40	4.89	1.76	0.79	0.95
	2000	0.82	4.54	1.73	0.85	0.97
	5000	0.5	4.39	1.69	0.88	0.99
	10000	0.41	4.29	1.69	0.9	0.99
Sub-domain and interface smoothing $Q_{\max} = 3.6$ $Q_{\mathrm{mean}} = 1.62$	1000	1.52	4.29	1.69	0.84	0.96
	2000	1.1	4.8	1.65	0.75	0.98
	5000	0.8	4.04	1.62	0.89	1
	10000	0.75	4.09	1.62	0.88	1

Tables 3 and 4 illustrate that both the value of λ, the exponential time-stepping parameter, and the number of Monte-Carlo simulations N affect the quality of the resulting mesh. The larger λ and N, the better the domain decomposition mesh approximates the quality of the single domain mesh. This is expected since as $\lambda \to \infty$ and $N \to \infty$ the numerical approximation of the stochastic representation of the solution to the linear mesh generator (4) approaches the deterministic solution and thus converges to the single domain result.

From Tables 3 and 4 it can be seen that the meshes obtained using domain decomposition with probabilistically computed solution along the interfaces give good quality meshes (compared to the single domain solution) with very few Monte-Carlo simulations (low N) at relatively coarse mean time steps (low λ). Global and local smoothing on the sub-domains can bring the mesh quality of the domain decomposition solution slightly closer to the single domain case, but these improvements are quite minor for the present example. Pre-smoothing of the interface does not

Table 4. Mesh quality for grids adapting to monitor function (11) with $\lambda = 5000$.

	N	l_∞	$Q_{\max}^{\text{DD}}$	$Q_{\text{mean}}^{\text{DD}}$	$R_{\max}$	R_{mean}
No smoothing $Q_{\max} = 3.86$ $Q_{\text{mean}} = 1.67$	1000	0.74	5.44	1.72	0.71	0.97
	2000	0.66	3.99	1.69	0.97	0.99
	5000	0.4	3.9	1.69	0.99	0.99
	10000	0.45	3.9	1.72	0.99	0.97
Global smoothing $Q_{\max} = 3.6$ $Q_{\text{mean}} = 1.62$	1000	0.762	3.79	1.65	0.95	0.98
	2000	0.47	3.83	1.64	0.94	0.99
	5000	0.36	3.75	1.64	0.96	0.99
	10000	0.44	3.67	1.62	0.98	1
Sub-domain smoothing $Q_{\max} = 3.6$ $Q_{\text{mean}} = 1.62$	1000	1.15	4.5	1.64	0.8	0.99
	2000	0.94	4	1.64	0.89	0.99
	5000	0.82	3.96	1.62	0.91	1
	10000	0.74	3.67	1.62	0.98	1
Interface smoothing $Q_{\max} = 3.86$ $Q_{\text{mean}} = 1.67$	1000	1.47	4.82	1.7	0.8	0.98
	2000	0.95	4.95	1.7	0.78	0.98
	5000	0.59	4.95	1.7	0.78	0.98
	10000	0.41	4.29	1.69	0.9	0.99
Sub-domain and interface smoothing $Q_{\max} = 3.6$ $Q_{\text{mean}} = 1.62$	1000	1.1	6	1.65	0.6	0.98
	2000	0.95	4.86	1.64	0.74	0.99
	5000	0.82	4.5	1.64	0.8	0.99
	10000	0.75	4.01	1.62	0.88	1

yield any improvement in the case of the monitor function (11); neither does pre-smoothing of the interface combined with smoothing on the sub-domains, at least not in the measure $Q(K)$.

The above results confirm for the present example that the probabilistic approach to domain decomposition can yield good quality meshes at relatively cheap computational costs. Different ways of smoothing have the potential to improve the mesh quality of domain decomposition grids further but ultimately may not be necessary at all.

5 Conclusions and outlook

In this paper we proposed a new method for computing PDE based adaptive meshes using a stochastic domain decomposition approach. The main idea derives from the study proposed in [1] to use probabilistic domain decomposition to solve linear boundary value problems. As various types of adaptation strategies fit into this class of problems, this probabilistic take on domain decomposition is readily applicable to the mesh generation problem. The main motivation behind this work is to find an algorithm for the construction of meshes that is fully parallelizable as it allows one to determine the single sub-domain solutions without information exchange across the interface of neighboring sub-domains. In addition, the relative low accuracy requirement for mesh generation suggests that quality meshes can be found with only a moderate

number of Monte–Carlo simulations along the interface and hence this mesh generation problem is well suited to this probabilistic domain decomposition approach.

We have restricted ourselves to a simple linear mesh generator of Winslow type. Despite being amongst the simplest PDE based mesh generators, we found the Winslow potential system suitable to demonstrate several typical issues to be addressed when using a combined probabilistic–deterministic solver. These issues include ways to smooth the mesh in order to avoid an excessive number of Monte-Carlo simulations as well as the optimal placement problem to determine the locations at the sub-domain interfaces where the solution is needed probabilistically. The former issue can be conveniently tackled using anisotropic diffusion to smooth out kinks that are inevitable due to the local nature of the Monte-Carlo simulations while properly preserving the regions of grid concentration and de-concentration. The latter problem of computing only few points along the sub-domain interfaces and hence further reducing the number of Monte-Carlo simulations needed is more difficult and ultimately boils down to a trade–off between the required grid smoothness and available computational resources. We found that placing the points near the maxima and minima of the first and second derivatives of the monitor function along the interfaces yields a sufficiently smooth mesh that can be determined at a fraction of the computational efficiency incurred if all interface points are obtained probabilistically. More sophisticated criteria for this optimal placement problem will be investigated in the future.

A more challenging problem is the generalization of the proposed method to nonlinear mesh generators. It is well known that linear mesh generators of Winslow type work reasonably well for problems for which isotropic mesh generation suffices. In turn, if more control over the grid adaptation is required, one usually has to resort to nonlinear mesh generators, e.g. those that are based on equidistribution and alignment, see e.g. [11,12]. The problem is that for nonlinear partial differential equations, very few stochastic solution representations are known. As the existence of a stochastic representation is at the heart of the proposed algorithm, it is crucial to either find a nonlinear mesh generator for which such a stochastic solution exists or to modify the problem so that it fits again into the realm of linear partial differential equations. The latter possibility may be realized through an appropriate linearization of the nonlinear mesh equations. This is the subject of current investigations.

A further crucial step to refine the proposed algorithm will be the development of a *stopping criteria* for the Monte-Carlo simulations. The task of generating meshes by solving a system of PDEs is different from the problem of obtaining numerical solutions of physical PDEs. For the former problem the mesh PDE is usually only a means to realize underlying computational grids with specific properties (e.g. isotropic vs. anisotropic adaptation). This means that it is usually not necessary to solve the mesh generation system with high accuracy, i.e. one can stop the solution procedure once a suitably smooth mesh is obtained. The derivation of proper stopping criteria for the mesh generation process will be another important source for improving the computational cost of the proposed algorithm. The main idea is to continuously evaluate the quality of the current candidate mesh and to stop or continue the solution procedure depending on whether a specified mesh quality threshold is reached. This strategy is fully compatible with the probabilistic computation of the interface solution due to the additive properties of the expected value.

As mentioned, the standard way to speed up Monte-Carlo simulations is to use quasi-random numbers instead of pseudo-random numbers. Using quasi-random numbers was a crucial factor in [1] to obtain more accurate numerical solutions using the probabilistic domain decomposition method. In this study we decided to use pseudo-random numbers only, as quasi-random numbers

cannot be computed in parallel. Introducing this potential bottleneck in the algorithm does not appear justified for the mesh generation problem since a good quality mesh can be found without running the stochastic simulation to convergence. We reserve the more careful study of this issue for future work.

In addition, it is necessary to do a careful comparision between this probabilistic DD approach and the more standard application of DD for mesh generation, cf. [7, 9]. The candidates for comparison include an optimized Schwarz approach, using DD as a preconditioner for the solving (3) or in a Krylov–Newton–Schwarz framework for nonlinear mesh generation.

Acknowledgements

References

[1] Acebrón J.A., Busico M.P., Lanucara P. and Spigler R., Domain decomposition solution of elliptic boundary-value problems via monte carlo and quasi-monte carlo methods, *SIAM J. Sci. Comput.* **27** (2005), 440–457.

[2] Anderson D.A., Equidistribution schemes, poisson generators, and adaptive grids, *Applied Mathematics and Computation* **24** (1987), 211–227.

[3] Budd C.J., Cullen M.J.P. and Walsh E.J., Monge–Ampère based moving mesh methods for numerical weather prediction, with applications to the Eady problem, *J. Comput. Phys.* **236** (2013), 247–270.

[4] Chen R. and Cai X.C., Parallel One-Shot Lagrange–Newton–Krylov–Schwarz Algorithms for Shape Optimization of Steady Incompressible Flows, *SIAM Journal on Scientific Computing* **34** (2012), B584–B605.

[5] Crowley W.P., An 'Equipotential' Zoner on a Quadrilateral Mesh, *Tech. rep.*, 1962.

[6] Dvinsky A.S., Adaptive grid generation from harmonic maps on Riemannian manifolds, *J. Comput. Phys.* **95** (1991), 450–476.

[7] Gander M.J. and Haynes R.D., Domain decomposition approaches for mesh generation via the equidistribution principle, *SIAM J. Numer. Anal.* **50** (2012), 2111–2135.

[8] Godunov S.K. and Prokopov G.P., The utilization of movable grids in gas dynamic calculations, *Ž. Vyčisl. Mat. i Mat. Fiz.* **12** (1972), 429–440.

[9] Haynes R.D. and Howse A.J.M., Generating equidistributed meshes in 2d via domain decomposition, *LNCSE, DD21* (2012), 776–797.

[10] He P. and Tang H., An adaptive moving mesh method for two-dimensional relativistic magnetohydrodynamics, *Computers and Fluids* **60** (2012), 1–20.

[11] Huang W., Variational mesh adaptation: isotropy and equidistribution, *J. Comput. Phys.* **174** (2001), 903–924.

[12] Huang W. and Russell R.D., *Adaptive Moving Mesh Methods*, Springer, New York, 2010.

[13] Huang W. and Sloan D.M., A simple adaptive grid method in two dimensions, *SIAM J. Sci. Comput.* **15** (1994), 776–797.

[14] Huang W. and Zhan X., Adaptive moving mesh modeling for two dimensional groundwater flow and transport, Amer. Math. Soc., Providence, RI, pp. 239–252, 2005.

[15] Jansons K.M. and Lythe G.D., Exponential timestepping with boundary test for stochastic differential equations, *SIAM J. Sci. Comput.* **24** (2003), 1809–1822.

[16] Karatzas I. and Shreve S.E., *Brownian motion and stochastic calculus*, vol. 113 of *Graduate Texts in Mathematics*, Springer, New York, 1991.

[17] Kloeden P.E. and Platen E., *Numerical solution of stochastic differential equations*, vol. 23 of *Application of Mathematics*, Springer Verlag, Berlin, 1992.

[18] Øksendal B., *Stochastic Differential Equations: An Introduction with Applications*, Springer, Heidelberg, 2010.

[19] Perona P. and Malik J., Scale-space and edge detection using anisotropic diffusion, *IEEE Trans. PAMI* **12** (1990), 629–639.

[20] Preis T., Virnau P., Paul W. and Schneider J.J., GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model, *J. Comput. Phys.* **228** (2009), 4468–4477.

[21] Press W.H., Teukolsky S.A., Vetterling W.T. and Flannery B.P., *Numerical recipes 3rd edition: The art of scientific computing*, Cambridge University Press, Cambridge, UK, 2007.

[22] Thompson J.F., Thames F.C. and Mastin C.W., Automatic numerical generation of body-fitted curvilinear coordinate system for field containing any number of arbitrary two-dimensional bodies, *J. Comput. Phys.* **15** (1974), 299–319.

[23] Thompson J.F., Warsi Z.U.A. and Mastin C.W., *Numerical grid generation: Foundations and applications*, North-Holland Publishing Co., New York, 1985.

[24] van Meel J.A., Arnold A., Frenkel D., Portegies Zwart S.F. and Belleman R.G., Harvesting graphics power for md simulations, *Molecular Simulation* **34** (2008), 259–266.

[25] Winslow A.M., Numerical solution of the quasilinear Poisson equation in a nonuniform triangle mesh, *J. Comput. Phys.* **1** (1966), 149–172.

[26] Winslow A.M., *Adaptive-mesh zoning by the equipotential method*, *Tech. Rep. UCID-19062*, Lawrence Livermore National Lab., CA (USA), 1981.

[27] Yuan Y., The characteristic finite element alternating-direction method with moving meshes for the transient behavior of a semiconductor device, *Int. J. Numer. Anal. Model.* **9** (2012), 86–104.

[28] Zhang Y. and Tang T., Simulating three-dimensional free surface viscoelastic flows using moving finite difference schemes, *Numer. Math. Theory Methods Appl.* **4** (2011), 92–112.