Available online at www.sciencedirect.com
SCIENCE DIRECT®
Real-Time Imaging 10 (2004) 41–51
www.elsevier.com/locate/rti

# High performance cluster computing with 3-D nonlinear diffusion filters

Andrés Bruhn[a,*], Tobias Jakob[b], Markus Fischer[b], Timo Kohlberger[c], Joachim Weickert[a], Ulrich Brüning[b], Christoph Schnörr[c]

[a] *Mathematical Image Analysis Group, Department of Mathematics and Computer Science, Saarland University, Building 27.2 Room 24, Saarbrücken 66041, Germany*
[b] *Computer Architecture Group, Department of Mathematics and Computer Science, University of Mannheim, Mannheim 68131, Germany*
[c] *Computer Vision, Graphics and Pattern Recognition Group, Department of Mathematics and Computer Science, University of Mannheim, Mannheim 68131, Germany*

**Abstract**

This paper deals with parallelization and implementation aspects of partial differential equation (PDE)-based image processing models for large cluster environments with distributed memory. As an example we focus on nonlinear diffusion filtering which we discretize by means of an additive operator splitting (AOS). We start by decomposing the algorithm into small modules that shall be parallelized separately. For this purpose image partitioning strategies are discussed and their impact on the communication pattern and volume is analyzed. Based on the results we develop an algorithmic implementation with excellent scaling properties on massively connected low-latency networks. Test runs on two different high-end Myrinet clusters yield almost linear speedup factors up to 209 for 256 processors. This results in typical denoising times of 0.4 s for five iterations on a $256 \times 256 \times 128$ data cube.


*MSC:* 68T45; 49K20; 65K10; 35J60; 65Y05; 65N06

## 1. Introduction

Partial differential equations (PDEs) form the basis of a number of recent methods in the fields of image processing and computer vision. In this paper we focus on nonlinear diffusion techniques that allow to denoise images while preserving edges. This property makes them useful for various restoration and segmentation purposes. Nonlinear diffusion models were first introduced by a work of Perona and Malik [1]. After some years their original model was improved by Catté et al. [2] from both a theoretical and practical viewpoint, and anisotropic extensions with a diffusion tensor [3] followed.

Many nonlinear diffusion algorithms are based on the simplest numerical scheme, an explicit finite difference discretization. While such schemes are easy to implement, they require small time steps for stability reasons. Hence, many iterations are needed to reach some interesting diffusion time, and the entire procedure is relatively inefficient. This has triggered a number of researchers to look for alternative algorithmic realizations of nonlinear diffusion filtering and related variational approaches.

These alternatives include three-level methods [4], semi-implicit approaches [2,5] and their multiplicative [6] or additive operator splitting variants [7], multigrid methods [8], finite element techniques [9–11], finite and complementary volume methods [5], numerical schemes with wavelets as trial functions [4,12], pseudo-spectral methods [4], lattice Boltzmann techniques [13], and stochastic simulations [14]. Approximations in graphics hardware have been considered in [15], and realizations on analog hardware are discussed in [16,17]. Related variational approaches have been treated using linearizations by means of auxiliary variables [18]. Also here it is possible to use adaptive finite elements [19] and to consider analog hardware realizations [20]. Parallel implementations on shared memory systems are studied in [21,22].

In the present paper, we focus on additive operator splitting (AOS) schemes for nonlinear diffusion filters.

*Corresponding author.
*E-mail address:* bruhn@mia.uni-saarland.de (A. Bruhn).

1077-2014/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rti.2003.12.002

These specific semi-implicit schemes have been first introduced to image analysis in [7]. Since then, they have been used for medical imaging problems [23], for regularization methods [24], image registration [25] and for optic flow computation [26]. Recently, also a number of active contour approaches [27–32] made use of these splitting techniques. The basic idea behind AOS schemes is to decompose a multi-dimensional problem into one-dimensional (1-D) ones that can be solved very efficiently. The final multi-dimensional solution is then approximated by averaging the 1-D solutions. AOS schemes inherit a number of favorable properties from their original continuous diffusion process and reveal linear complexity [7]. Their usefulness has also been shown in a number of other applications ranging from Navier–Stokes equations [33,34] to sandpile growth simulations [35]. In fact, it seems that Navier–Stokes equations have constituted one of their historically first application domains.

The rising popularity of AOS schemes has soon triggered first parallel implementations for diffusion filtering [36]. At that time, however, the development of network architectures did not allow the efficient use of distributed memory systems for such communication-intensive problems. For this reason these approaches generally stayed confined to systems based on *shared* memory. In recent years a rapid progress in this sector changed the situation completely. High-performance cluster systems with massively connected low-latency networks were built throughout the world. There are two reasons for this development: Firstly, cluster systems are much more attractive to customers, since they are less expensive than the shared memory systems. This has increased their availability for research purposes. Secondly, the number of processors is not limited by such severe hardware restrictions than in the case of shared memory systems, thus allowing larger scaling possibilities. In order to exploit this potential, parallelization approaches must fit the underlying network topology.

The goal of the present paper is to show that a 3-D nonlinear diffusion process can be parallelized in such way, that it reveals excellent scaling properties regarding both computation and communication on a *distributed* memory system. To this end the use of a parallel repartitioning strategy is proposed, that maintains the computational efficiency from the sequential setting while limiting the therefore required communication at the same time.

The paper is organized as follows. In Section 2, a review on diffusion filtering and the AOS scheme is given. Furthermore, a modular decomposition before parallelization is shown. In Section 3, partitioning and communication models are discussed. Relevant parallelization and implementation details of our approach are explained in Section 4, while Section 5 deals with communication costs. In Section 6, results obtained from test runs on two high-performance cluster systems are presented. The summary in Section 7 concludes our paper. A preliminary shorter version of this paper has been presented at a symposium [37]. In the current version more algorithmic details are presented, theoretical bounds for the communication volume are given and additional experiments are performed.

## 2. Nonlinear isotropic diffusion

### 2.1. Continuous model

In the following, we give a short review of the nonlinear diffusion model of Catté et al. [2]. A gray value image $f$ is considered as a function from a given domain $\Omega \subset \mathbb{R}^m$ into $\mathbb{R}$. In our case we have $m \in \{2, 3\}$, which corresponds to 2-D and 3-D images. The basic nonlinear diffusion problem then reads:

Find a function $u(x, t)$: $\Omega \times \mathbb{R}_0^+ \to \mathbb{R}$ that solves the diffusion equation

$$\partial_t u = \operatorname{div}(g(|\nabla u_\sigma|^2)\nabla u) \quad \text{on} \quad \Omega \times \mathbb{R}_0^+ \tag{1}$$

with $f$ as initial value.

$$u(x, 0) = f(x) \quad \text{on} \quad \Omega \tag{2}$$

and reflecting boundary conditions

$$\partial_n u = 0 \quad \text{on} \quad \partial\Omega \times \mathbb{R}_0^+, \tag{3}$$

where $u_\sigma = K_\sigma u$ denotes the convolution of $u$ with a Gaussian of standard deviation $\sigma$, $n$ is a normal vector perpendicular to $\partial\Omega$, and the diffusivity $g$ is a non-negative decreasing function with $g \in C^\infty[0, \infty)$. In this paper, a diffusivity function originally proposed by Perona and Malik [1] is used. It is given by

$$g(|\nabla u_\sigma|^2) := \frac{1}{1 + |\nabla u_\sigma|^2/\lambda^2}, \tag{4}$$

where $\lambda$ is a contrast parameter. The solution $u(x, t)$ is a family of images, where the diffusion time $t$ acts as a scale parameter. An example illustrating the performance of this diffusion filter in 2-D is given in Fig. 1. For a detailed description of nonlinear diffusion filter design the interested reader may refer to [3].

### 2.2. Additive operator splitting

Since such nonlinear diffusion equations cannot be solved analytically, numerical approximations are required. In [7] a finite difference scheme based on an AOS technique is used for this purpose. This AOS technique forms the basis of our parallelization efforts. It is an extension of the semi-implicit scheme for nonlinear

(a) (b)

Fig. 1. (a) Test image with a gray scale range [0, 255] degraded by Gaussian noise with standard deviation $\sigma_n = 30$. (b) Image denoised by the nonlinear diffusion filter, 5 iterations with $\sigma = 0.01$, $\lambda = 2.5$ and $\tau = 20$.

diffusion filtering and can be described as

$$u^{k+1} = \frac{1}{m} \sum_{l=1}^{m} (I - m\tau A_l(u_\sigma^k))^{-1} u^k, \tag{5}$$

where $u^k$ is a vector with the gray values at all pixels as components. The iteration index $k$ refers to the diffusion time $t = k\tau$ where $\tau$ is the time step size. The tridiagonal matrix $A_l$ is a discretization of the divergence expression along the $l$th coordinate axis. Let $N_l(i)$ denote the set of neighbors of pixel $i$ in direction of axis $l$, let $h_l$ be the corresponding grid size and let $g_i^k$ stand for the evaluated diffusivity at pixel $i$ of the pre-smoothed image $u_\sigma^k$, then the matrix $A_l(u_\sigma^k)$ is given by

$$(A_l(u_\sigma^k))_{ij} := \begin{cases} -\sum_{n \in N_l(i)} \dfrac{g_i^k + g_n^k}{2h_l} & \text{if } j = i, \\ \dfrac{g_i^k + g_j^k}{2h_l} & \text{if } j \in N_l(i), \\ 0 & \text{else.} \end{cases} \tag{6}$$

In each iteration step, the AOS method requires the solution of $m$ tridiagonal linear systems of equations. Each system describes diffusion along one coordinate direction and may even be decomposed into smaller tridiagonal systems. The final result at the next time level is obtained by averaging these 1-D diffusion results.

Such a splitting into 1-D diffusion processes offers significant computational advantages: The corresponding tridiagonal systems can be solved in linear complexity by means of the so-called Thomas algorithm [38], a specific variant of the Gaussian algorithm; see [7,39] for further details.

Typical AOS schemes are one order of magnitude more efficient than simple diffusion algorithms. Although they are stable for all time step sizes $\tau$ one usually limits the step size for accuracy reasons. Hence, the scheme is applied in an iterative way in order to reach some interesting stopping time.

### 2.3. Algorithmic decomposition

The following algorithmic steps can easily be derived from the iteration instruction for the AOS Scheme (5).

(1) Perform a Gaussian pre-smoothing of $u$ using $u_\sigma^k = K_\sigma u^k$.
(2) Compute derivatives $|\nabla u_\sigma^k|^2$ and diffusivity values $g(|\nabla u_\sigma^k|^2)$.
(3) Set up and solve all $m$ tridiagonal systems $(I - m\tau A_l(u_\sigma^k))u_l^{k+1} = u^k \quad (l = 1, \ldots, m)$. Average the computed results: $u^{k+1} = (1/m) \sum_{l=1}^{m} u_l^{k+1}$.

## 3. Parallelization models

The following parallelization models are based on image partitioning. This allows parallel execution of fast sequential algorithms instead of applying slower parallel variants to the complete image domain. Since the unbalanced case shall not be discussed here, images are assumed to be dividable in partitions of equal size.

### 3.1. Communication models

A large part of image processing algorithms consist of neighborhood operations. This raises problems at partition boundaries, since required information is missing. Let us now discuss two communication models to handle this problem: repartitioning and boundary communication.

*Repartitioning*: The basic idea of the repartitioning strategy is to find an appropriate partitioning for each operation, such that the problem of missing neighborhood information does not occur. Therefore, partitions have to be relocated and reshaped by means of communication. In many cases this communication involves data exchanges between all processes, the so-called *all-to-all communication*.

For large partition numbers such a connection–intensive communication pattern makes high demands on the network topology. Whether the network can satisfy these demands or not is reflected in a scaling of bandwidth or a rise of communication time. As long as highly parallel pairwise disjoint communication is possible and the network latency is small compared to the time required for data transfer, the total bandwidth increases almost linearly with the number of processors. However, this effect is only present for massively connected low-latency networks. The use of other network types results in performance breakdowns, either because of topological limitations (e.g. token rings or computing grids) or relatively high latency times (e.g. Gigabit Ethernet).

Taking a closer look at the total communication volume the importance of this bandwidth scaling property becomes obvious. Since non–overlapping partitions are used, each pixel is sent and received by no more than one process. Thus, the communication behavior imposes a limit to the total communication volume that is given by the image size. The required neighborhood and the number of processors can only affect the total communication volume within this scope as shown in Fig. 2. Hence, each scaling of bandwidth is passed on to the communication time.

*Boundary communication*: Keeping existing partitions, the second communication model simply exchanges the missing neighborhood information. One should note that this implies a dependency of the total communication volume on two unknowns: The number of partitions as well as the boundary size.

For moderate values of both parameters, the communication is limited to its adjacent segments. In this case the total communication volume may drop significantly beyond that of a repartitioning strategy (cf. Fig. 2). Moreover such a simple communication pattern has a second advantage. Since it makes less demands on the network topology than the previously discussed all–to–all communication, also weakly connected cluster systems do benefit from a bandwidth scaling effect. Even with respect to high-latency networks this strategy is favorable due to its rather large message size that results from the limited communication pattern.

However, larger boundary sizes and partition numbers do change the situation completely. Then boundary-volume ratios deteriorate, communication patterns may require extensions to further partitions and finally an inefficient parallelization remains. This is reflected in the enormous communication volumes shown for larger boundary sizes in Fig. 2. In the worst case the communication volume is only limited by $(n-1)$ times the image size, where $n$ is the number of partitions. Hence boundary exchange does only address operations that require information from a small neighborhood.

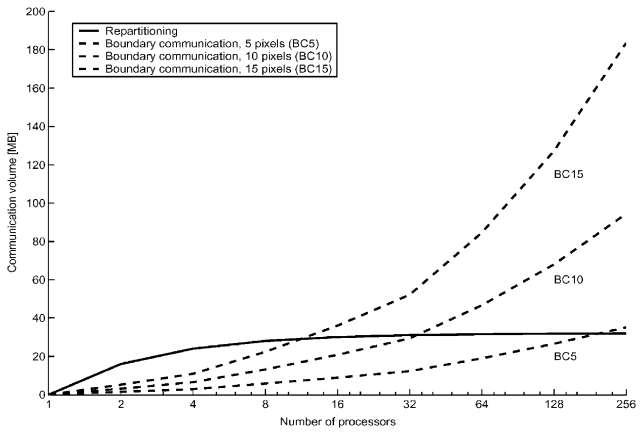


Fig. 2. Impact of boundary size and processor number on the exchanged data volume for different communication models by the example of a separable 3-D convolution. Data volumes refer to a 32-bit data set of size $256 \times 256 \times 128$.

In parallelized software for scientific simulations, boundary exchange is a frequently used communication model. Since the underlying theory allows a free scaling of the problem size, the loss of efficiency can be circumvented by a finer sampling of the continuous model. Thus, along with an increased accuracy, larger partition sizes and, therefore, better scaling properties for larger cluster systems are obtained. In the area of image processing such possibilities are not given. Since the accuracy is determined at the moment of image acquisition, a subsequent change of the problem size by means of interpolation is only of limited use.

### 3.2. Partition models

In addition to the communication models appropriate image partitioning strategies have to be chosen. In general, cuboid partitions are preferred since they can be realized with commonly used data structures and are easier to handle. There are two partitioning models that result in such cuboid partitions.

*Slice partitioning*: As the name anticipates the main idea of this strategy is to partition an image along one single direction. Thus no further boundaries arise. Operations that are separable or do not require neighborhood information from all directions can exploit this property. An example is given in Fig. 3(a), where a data cube divided in four slice partitions is shown.

However, there are also two disadvantages of this strategy. Firstly, the maximum number of partitions is limited by the number of pixels in the direction of partitioning. Secondly, slices have an evidently bad boundary-volume ratio. While the first drawback is only relevant for small image sizes, the second one has no relevance if repartitioning is applied.

*Mesh partitioning*: This strategy focuses on partitioning an image along all directions. Thus the largest

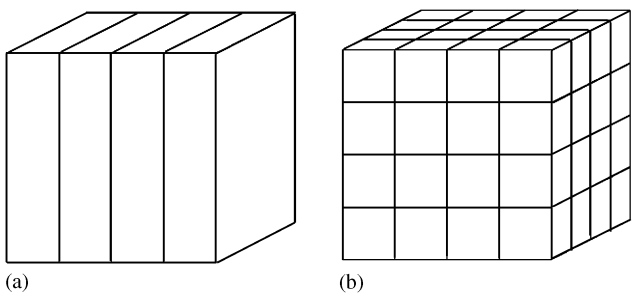


Fig. 3. Partition models. (a) Slice partitioning; (b) mesh partitioning.

theoretical scalability is achieved, since the maximum number of partitions is only limited by the total number of pixels. Its main disadvantage is the occurrence of boundaries in all directions. In our case this drawback is quite severe, since the performance of certain operations lives on their separability property.

A special case of mesh partitioning is cube-like partitioning. Thereby an image is partitioned in such a way, that the sum of all partition boundaries is minimized. Obviously, this partition strategy should be used when it comes to the exchange of boundary information. An example for mesh partitioning is shown in Fig. 3(b).

## 4. Algorithmic details

The basic idea of our parallelization approach is the use of two alternating slice partitions as shown in Fig. 4.

In order to obtain a balanced problem with equal partition sizes and equal execution times, at least two out of three data dimensions ($X$, $Y$) are required to be a multiple of the processor number. However, there are no restrictions regarding the size of the third dimension ($Z$).

In the remainder of this section all parallelized modules are discussed in detail. A global overview of the computation and communication steps resulting from this parallelization is given in Fig. 5. It illustrates one iteration of the final algorithm.

*Module* 1: Gaussian convolution. For optimal performance, the Gaussian convolution is implemented exploiting separability and symmetry. The required 1-D convolution masks are obtained by sampling the continuous Gaussian

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2} \tag{7}$$

and truncating it at three times the standard deviation. Each mask is then renormalized such that its weights sum up to 1.

Moreover, data access patterns are optimized with respect to an efficient use of the available cache resources. This allows for an extensive use of precached data. As a consequence, significantly smaller runtimes are already achieved in the sequential setting.

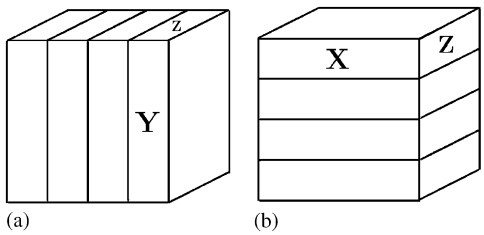


Fig. 4. Slice partitions. (a) $YZ$ partitioning; (b) $XZ$ partitioning.

The parallelization is performed by combining the alternating slice partitions with a repartitioning strategy. Thus, Gaussian convolution in two out of three directions can be performed without communication effort (Fig. 6(a)). Only smoothing in the third direction requires a previous repartitioning step (Fig. 6(b)). The main advantage of this implementation is the independence of its communication effort on the value for the standard deviation $\sigma$. Since no boundary exchange takes place, the algorithm achieves equal performance for any amount of pre-smoothing.

*Module* 2: Derivatives and diffusivity. Derivatives within the diffusivity are computed using central differences. Since this uses stencils of type $(1/2h)(-1, 0, 1)$, where $h$ denotes the grid size, the boundary size is limited to 1. Besides, the computation of the diffusivity values demands matching partitions for all derivatives. Both aspects favor a boundary exchange strategy. Although cube-like partitioning would be desirable, a change of the partition model at the cost of two repartitioning steps is obviously not profitable. Hence, the alternating slice partitions are combined with boundary communication.

Since parallelism is achieved via image partitioning, derivatives are computed sequentially for all directions. Again, the exchange of neighborhood information takes place after the computation for two out of three directions is completed (Fig. 7(a) and (b)). Finally, the diffusivity values are computed in place based on the diffusivity function given in Eq. (4).

*Module* 3: Diffusion and AOS. As discussed before, AOS offers parallelism on two different levels. First, it allows to decouple the diffusion processes for each direction (coarse grain parallelism). For the same reason as in the case of the derivative computation, this property will not be exploited directly for parallelization purposes. Of major importance is the fact, that the huge linear tridiagonal equation systems for each diffusion direction can be decomposed into many small independent equation systems of same style (mid-grain parallelism). Since each of these smaller systems corresponds to the diffusion process along one complete image line, the use of mesh partitions makes no sense. Instead, it seems to be desirable to combine the alternating slice partitions once more with a repartitioning strategy. This allows the application of fast sequential solvers such as the Thomas algorithm [7,38]. By performing an $LR$ decomposition, a forward substitution as well as a backward substitution step, it solves tridiagonal systems very efficiently. Because of the sequential nature of these substitution steps, special variants for a boundary exchange strategy could not have been developed without loss of parallelism and performance.

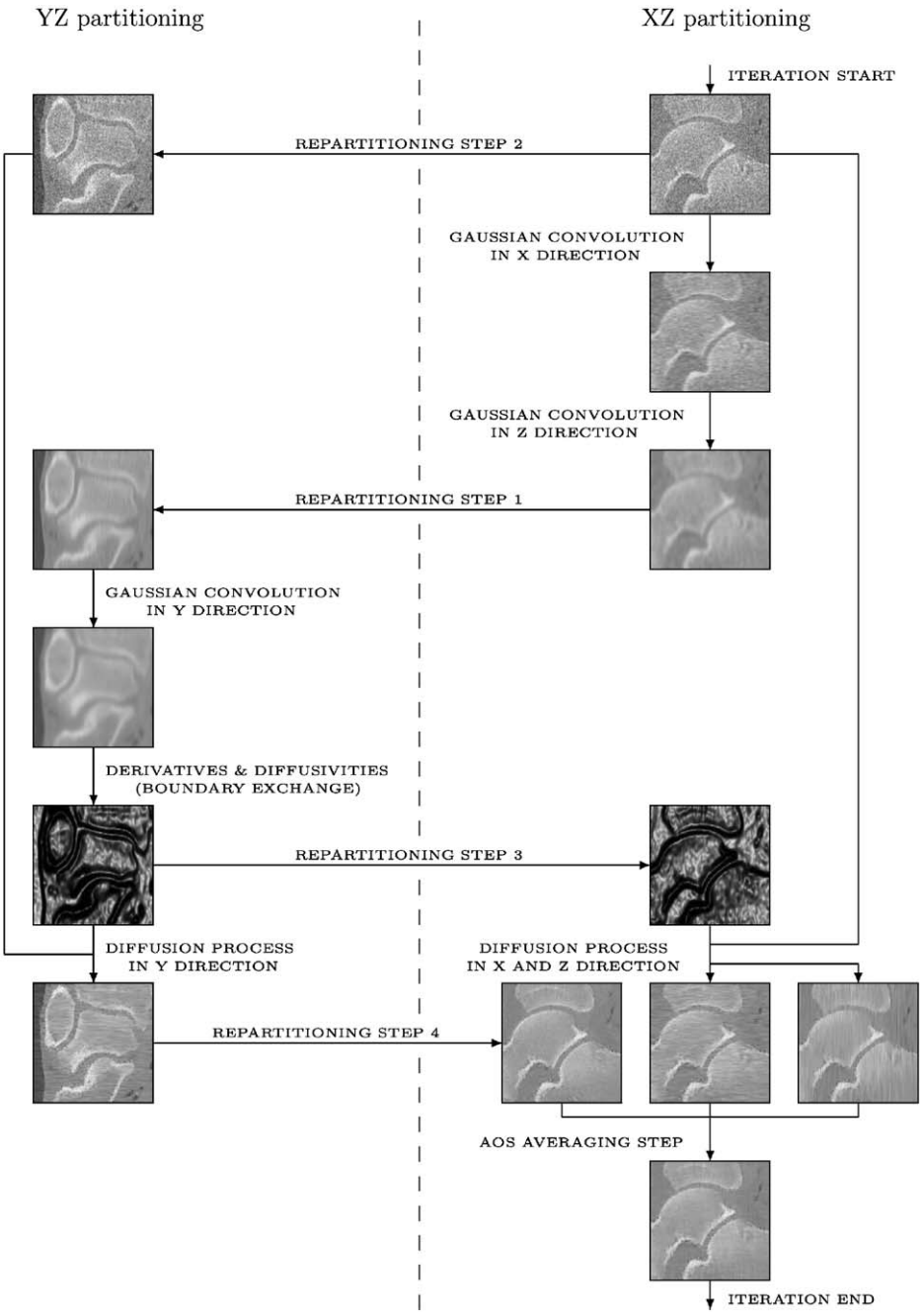


Fig. 5. Flow diagram illustrating one iteration step of the implemented algorithm. Column headings give information on the direction of data partitioning and the cut direction of presented slices for intermediate results. Computation and communication steps are symbolized by arrows.

However, even in the case of repartitioning the parallelization effort is large: In order to set up and solve the tridiagonal equation system for one direction, matching slice partitions for the image $u^k$ *and* and the diffusivity values $g(|\nabla u_\sigma^k|^2)$ are required. Moreover, two of this partition pairs ($XZ$ and $YZ$) are needed to cover all three diffusion directions. Therefore, not only the image has to be repartitioned (Fig. 8(a)), but also the diffusivity data (Fig. 8(b)). Finally, combining the results of all three diffusion processes—the averaging step in the AOS scheme—requires a third repartitioning.

## 5. Communication costs

Let $p$ be the number of processors and let $n_x \geqslant n_y \geqslant n_z$ be the dimensions of the data set in direction $x, y$ and $z$, with $n_x$ and $n_y$ being a multiple of $p$. Then the total number of pixels sent and received during one

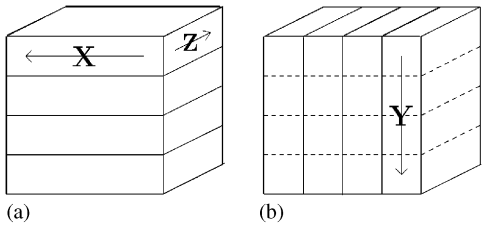


Fig. 6. Partitioning scheme for Module 1. (a) Data cube before repartitioning step 1. (b) Data cube after repartitioning step 1. Arrows show the directions in which Gaussian convolutions are performed. Solid lines are boundaries of current partitions while dashed lines refer to boundaries of previously used partitions.

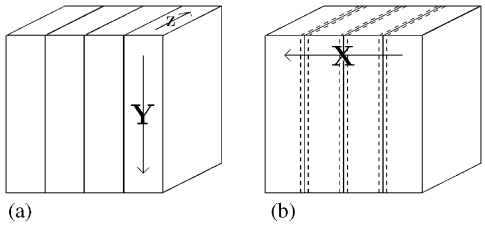


Fig. 7. Partitioning scheme for Module 2. (a) Data cube before boundary exchange. (b) Data cube after boundary exchange. Arrows show the directions in which derivatives are computed. Solid lines are boundaries of current partitions while dashed lines refer to boundaries that have been exchanged.

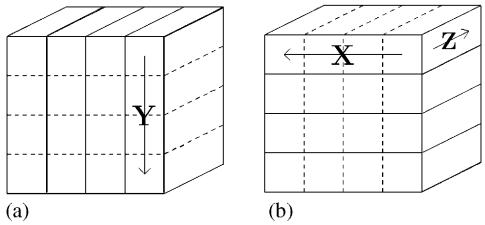


Fig. 8. Partitioning scheme for Module 3. (a) Pair *YZ*: repartitioned image (repartitioning step 2) matching diffusivity from in Fig. 7(b). (b) Pair *XZ*: repartitioned diffusivities (repartitioning step 3) matching image from Fig. 6(a). Arrows show the directions in which diffusion is computed. Solid lines are boundaries of current partitions while dashed lines refer to boundaries of previously used partitions.

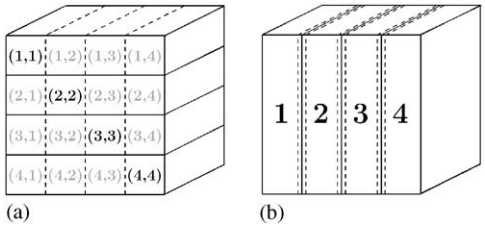


Fig. 9. Communication pattern. (a) Repartitioning strategy. Entries $(s,r)$ denote communication with sender $s$ and receiver $r$. Blocks on the main diagonal do not require communication. (b) Boundary communication.

repartitioning step is given by

$$v_{rep}(p) = \underbrace{(p^2-p)\left(\frac{n_x}{p}\frac{n_y}{p}n_z\right)}_{\text{messages}\times\text{message size}} = \left(1-\frac{1}{p}\right)n_x n_y n_z < n_x n_y n_z. \quad (8)$$

Fig. 9(a) illustrates this equation by an example where four processors have been used. While there are $p^2 = 16$ data packets of size $\frac{1}{16}(n_x n_y n_z)$ in total, only $p^2 - p = 12$ of them are exchanged between the processors.

For the boundary communication, the number of exchanged pixels can be calculated in a straightforward way:

$$v_{bc}(p) = \underbrace{((p-2)\cdot 2 + 2\cdot 1)(n_x n_z)}_{\text{messages}\times\text{message size}} = 2n_x(p-1)n_z < 2n_x n_y n_z. \quad (9)$$

Due to the fixed boundary size of one pixel, slices with a volume of $n_x n_z$ are sent to the left and right neighbor as shown in Fig. 9(b). One should note that two processors—the first and the last—have only one neighbor and thus contribute only one slice to the communication volume.

Since one iteration consists of four repartitioning steps and one boundary communication step, the total communication volume sums up to

$$v_{tot}(p) = 4\left(1-\frac{1}{p}\right)n_x n_y n_z + 2n_x(p-1)n_z < 6n_x n_y n_z \quad (10)$$

Thus, the total communication volume in each iteration is limited by six times the image size.

## 6. Results

For performance evaluation a C/C++ implementation based on the platform independent inter-process communication standard message passing interface (MPI) was developed. Thereby the original MPI routines for collective communication were replaced with appropriate sequences of non-blocking send and receive operations. Preliminary test runs showed that this substitution leads to significantly shorter communication times and thus improves the overall scalability.

The presented results have been obtained by the application of 10 AOS iterations to a 32-bit medical data set of size $256 \times 256 \times 128$. Required computation and communication times were measured using the MPI function `MPI_wtime`. Basis for the determination of speedup factors and the theoretical maximum is the runtime of the optimized sequential code for the iteration loop on a single CPU.

### 6.1. ScoreIIIe cluster (Tsukuba)

Our first test run has been performed on the Score IIIe cluster of the real world computing partnership (RWCP) at the Tsukuba Research Center in Japan. Running a modified Linux 2.4 SMP Kernel it consists of 524 nodes with two PIII 933 MHz processors each. Focusing on distributed memory systems only one CPU per node has been considered at a time. The cluster is fully connected to a CLOS network and uses 64 Bit/66 MHz Myrinet2000 network interfaces in the slower 33 MHz mode. Due to its performance it is ranked 131th in the June 2003 TOP 500 list of supercomputers.

As one can see from Fig. 10 considerations regarding the parallelization for a specific network architecture do pay off. The obtained results demonstrate an excellent, almost linear scaling behavior up to 256 nodes with a top speedup of 209. This equals 82% of the theoretical maximum. The corresponding runtimes divided in computation and communication effort can be found in Table 1. Thereby up to 1.83-GB data are sent and received in a single second. These numbers show the importance of a sophisticated algorithm design that allows bandwidth scaling up to a large number of processors.

This scaling property is reflected in the percental distribution, that shows only a moderate increase of the communication part. In particular the bandwidth doubling from 8 to 16 and from 64 to 128 processors should be noted. This effect is directly connected to the hierarchical structure of the underlying CLOS network. Each time a doubling of the number of processors requires the extension of the actually used network to a higher hierarchy level, the available bandwidth scales perfectly. And even in the case of 256 processors the communication ratio does hardly exceed one quarter of the runtime.

### 6.2. HeLiCS cluster (Heidelberg)

The second test run took place on the HeLiCS cluster of the Interdisciplinary Center for Scientific Computing at University of Heidelberg in Germany. It consists of 256 nodes each running a Debian Woody Kernel 2.4.21 on two 1.4 GHz Athlon MP processors. Again, only one processor per node has been used at a time. This cluster is also connected via a CLOS network, but allows the 64 Bit/66 MHz Myrinet2000 interfaces to be run at the

Table 1
Runtimes on the Score IIIe cluster for 10 iterations divided in computation and communication times

| CPUs | Runtime (s) | Comp. (s) | Comm. (s) | Comp. (%) | Comm. (%) |
|---|---|---|---|---|---|
| 1 | 212.741 | 212.741 | 0.000 | 100.000 | 0.000 |
| 2 | 114.625 | 106.205 | 8.420 | 92.654 | 7.346 |
| 4 | 57.534 | 52.221 | 5.513 | 90.766 | 9.234 |
| 8 | 29.401 | 26.123 | 3.278 | 88.851 | 11.149 |
| 16 | 15.065 | 13.471 | 1.594 | 89.420 | 10.580 |
| 32 | 7.731 | 6.753 | 0.978 | 87.350 | 12.650 |
| 64 | 4.029 | 3.333 | 0.696 | 82.725 | 17.275 |
| 128 | 1.894 | 1.550 | 0.344 | 81.837 | 18.163 |
| 256 | 1.017 | 0.745 | 0.272 | 73.255 | 26.745 |

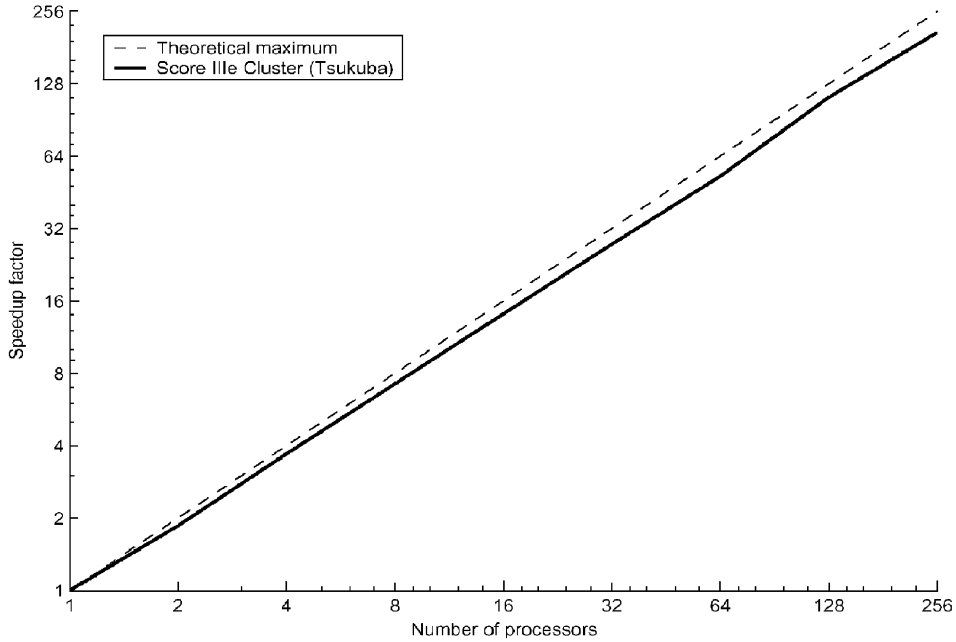


Fig. 10. Speedup chart, ScoreIIIe cluster.

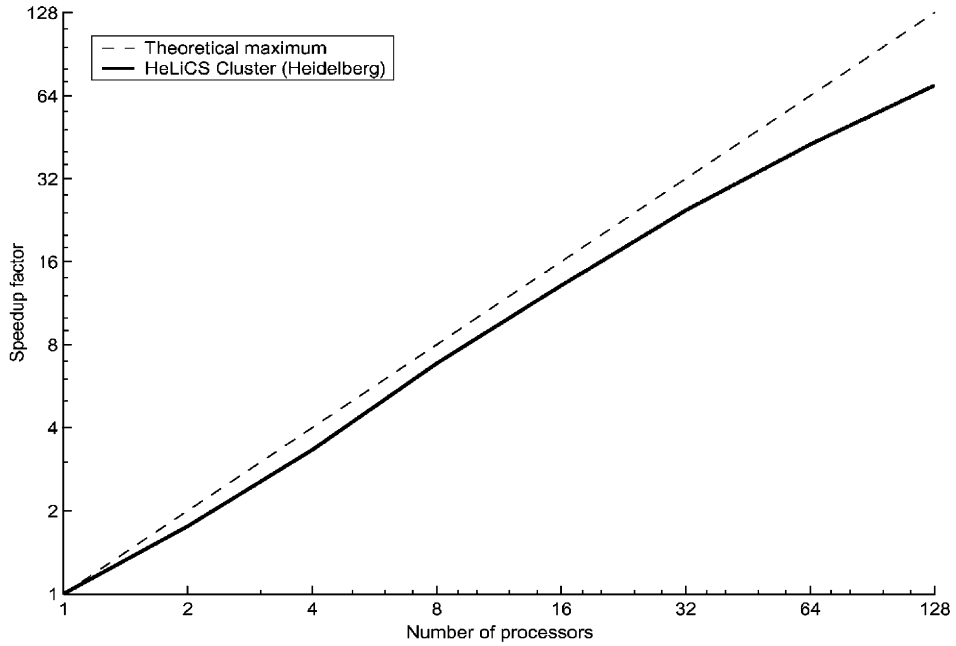


Fig. 11. Speedup chart, HeLiCS cluster.

Table 2
Runtimes on the HeLiCS cluster for 10 iterations divided in computation and communication times

| CPUs | Runtime (s) | Comp. (s) | Comm. (s) | Comp. (%) | Comm. (%) |
|---|---|---|---|---|---|
| 1 | 57.991 | 57.911 | 0.000 | 100.000 | 0.000 |
| 2 | 32.985 | 28.288 | 4.697 | 85.760 | 14.240 |
| 4 | 17.348 | 13.645 | 3.703 | 78.645 | 21.355 |
| 8 | 8.476 | 6.855 | 1.621 | 80.876 | 19.124 |
| 16 | 4.426 | 3.420 | 1.006 | 77.275 | 22.725 |
| 32 | 2.360 | 1.708 | 0.652 | 72.376 | 27.624 |
| 64 | 1.362 | 0.857 | 0.505 | 62.912 | 37.088 |
| 128 | 0.831 | 0.470 | 0.361 | 56.524 | 43.477 |

full frequency of 66 MHz. In the June 2003 TOP 500 list the HeLiCS cluster is ranked 80th.

Fig. 11 shows once more a very good scaling behavior with speedup factors up to 70 for 128 processors. Since network resources had to be shared with other parallel programs during our test runs, this even constitutes a lower limit for the actual scaling performance of the algorithm on this cluster.

The corresponding runtimes divided in computation and communication effort are presented in Table 2. As one can see, already 128 nodes of the HeLiCS cluster are sufficient to outperform the results obtained by 256 nodes of the ScoreIIIe cluster. With a runtime of 0.831 s this equals an overall improvement of 18%. The percental distribution shows that this gain is directly connected to the scaling of the computation time. However, the bandwidth scaling effect for an increasing number of processors is less distinct due to the competition for network resources. But even in this case, the results were considerably well, as we have seen.

An example for the quality of the parallelized nonlinear diffusion algorithm is given in Fig. 12. Although the medical data set was severely degraded by Gaussian noise of standard deviation $\sigma_n = 30$, the algorithm is still able to recover basic structures. Performing just five iterations, runtimes of 0.4 s are obtained on the HeLiCS cluster. This results in typical processing rates of more than two data cubes per second.

## 7. Summary and conclusions

The goal of this paper was to show how to design PDE-based image processing algorithms for high-performance cluster systems. This was done by the example of nonlinear diffusion. Based on an AOS scheme, we first performed a decomposition into modules. Then parallelization strategies suitable for a high-performance low-latency network were discussed. We saw that in this case a repartitioning approach is favorable for the majority of operations. Moreover, we noticed that this strategy should be combined with slice partitioning for optimal performance. Test runs with our implementation on two high-end cluster system yielded speedup factors of up to 209 for 256 nodes, proving its excellent scalability.

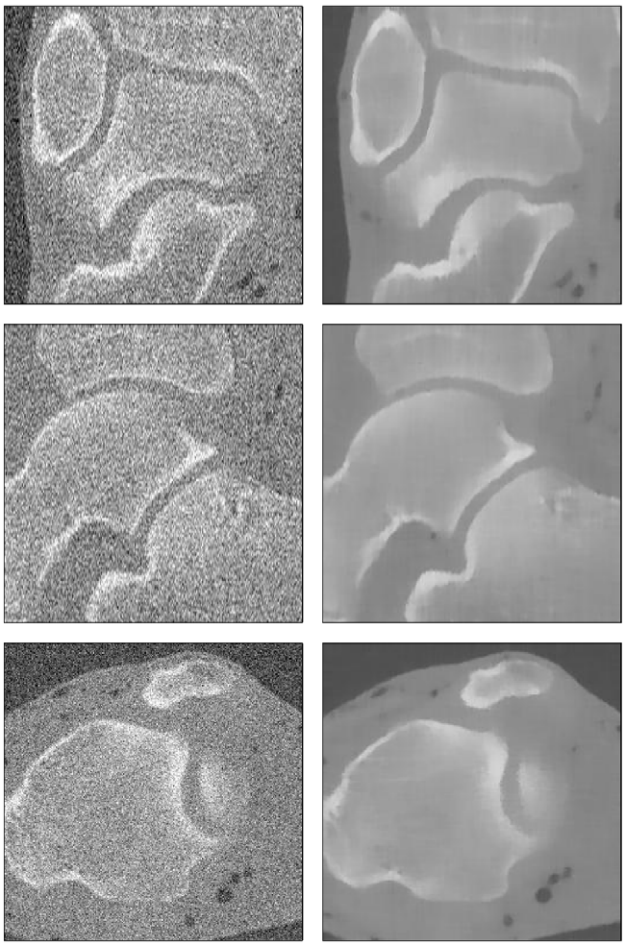

Fig. 12. CT scan of a human foot area with gray scale range $[0, 255]$. The data size is $256 \times 256 \times 128$. From top to bottom: Slices in $XY$, $XZ$ and $YZ$ direction. Left: Data set degraded by Gaussian noise with standard deviation $\sigma_n = 30$. Right: Data set denoised by the implemented algorithm, 5 iterations with $\sigma = 0.5$, $\lambda = 2.5$ and $\tau = 20$.

## Acknowledgements

Our research has been partly funded by the Deutsche Forschungsgemeinschaft (DFG) under the project SCHN 457/4-1. This is gratefully acknowledged. We also thank Wiro Niessen at Utrecht University Hospital for providing the 3-D trabecular bone data set.

## References

[1] Perona P, Malik J. Scale space and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence 1990;12:629–39.

[2] Catté F, Lions P-L, Morel J-M, Coll T. Image selective smoothing and edge detection by nonlinear diffusion. SIAM Journal on Numerical Analysis 1992;32:1895–909.

[3] Weickert J. Anisotropic diffusion in image processing. Stuttgart: Teubner; 1998.

[4] Fröhlich J, Weickert J. Image processing using a wavelet algorithm for nonlinear diffusion. Technical Report 104, Laboratory of Technomathematics, University of Kaiserslautern, Germany, March 1994.

[5] Handlovičová A, Mikula K, Sgallari F. Variational numerical methods for solving diffusion equation arising in image processing. Journal of Visual Communication and Image Representation 2002;13(1):217–37.

[6] Weickert J, ter Haar Romeny BM, Lopez A, van Enk WJ. Orientation analysis by coherence-enhancing diffusion. In: Proceedings of the 1997 Real World Computing Symposium, Tokyo, Japan, January 1997. p. 96–103.

[7] Weickert J, ter Haar Romeny BM, Viergever MA. Efficient and reliable schemes for nonlinear diffusion filtering. IEEE Transactions on Image Processing 1998;7(3):398–410.

[8] Acton ST. Multigrid anisotropic diffusion. IEEE Transactions on Image Processing 1998;7(3):280–91.

[9] Bänsch E, Mikula K. A coarsening finite element strategy in image selective smoothing. Computation and Visualization in Science 1997;1:53–61.

[10] Kačur J, Mikula K. Solution of nonlinear diffusion appearing in image smoothing and edge detection. Applied Numerical Mathematics 1995;17:47–59.

[11] Preußer T, Rumpf M. An adaptive finite element method for large scale image processing. In: Nielsen M, Johansen P, Olsen OF, Weickert J. editors. Scale-space theories in computer vision, Lecture notes in computer science, vol. 1682. Berlin: Springer; 1999. p. 223–34.

[12] Fontaine FL, Basu S. Wavelet-based solution to anisotropic diffusion equation for edge detection. International Journal of Imaging Systems and Technology 1998;9:356–68.

[13] Jawerth B, Lin P, Sinzinger E. Lattice Boltzmann models for anisotropic diffusion of images. Journal of Mathematical Imaging and Vision 1999;11:231–7.

[14] Ranjan US, Ramakrishnan KR. A stochastic scale space for multiscale image representation. In: Nielsen M, Johansen P, Olsen OF, Weickert J. editors. Scale-space theories in computer vision. Lecture notes in computer science, vol. 1682. Berlin: Springer; 1999. p. 441–6.

[15] Rumpf M, Strzodka R. Nonlinear diffusion in graphics hardware. In: Proceedings of the Joint Eurographics—IEEE TCVG Symposium on Visualization, Ascona, Switzerland, Springer, May 2001.

[16] Gijbels T, Six P, Van Gool L, Catthoor F, De Man H, Oosterlinck A. A VLSI-architecture for parallel non-linear diffusion with applications in vision. In: Proceedings of the IEEE Workshop on VLSI Signal Processing, La Jolla, USA, vol. 7. 1994. p. 398–407.

[17] Perona P, Malik J. A network for multiscale image segmentation. In: Proceedings of the IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 1998. p. 2565–8.

[18] Charbonnier P, Blanc-Féraud L, Aubert G, Barlaud M. Deterministic edge-preserving regularization in computed imaging. IEEE Transactions on Image Processing 1997;6(2): 298–311.

[19] Schnörr C. A study of a convex variational diffusion approach for image segmentation and feature extraction. Journal of Mathematical Imaging and Vision 1998;8(3):271–92.

[20] Wiehler K, Heers J, Schnörr C, Stiehl H-S, Grigat R-R. A 1D analog VLSI implementation for non-linear real-time signal preprocessing. Real-Time Imaging 2001;7(1):127–42.

[21] Heers J, Schnörr C, Stiehl H-S. Investigation of parallel and globally convergent iterative schemes for nonlinear variational image smoothing and segmentation. In: Proceedings of the 1998 IEEE International Conference on Image Processing, vol. 3. Chicago, IL, October 1998. p. 279–83.

[22] Heers J, Schnörr C, Stiehl HS. Globally-convergent iterative numerical schemes for non-linear variational image smoothing and segmentation on a multi-processor machine. IEEE Transactions on Image Processing 2001;10(6):852–64.
[23] Rexilius J. Anisotrope nichtlineare Diffusion für die Bildverarbeitung. Technical Report B-99-07, Institute of Mathematics and Computer Science, Medical University of Lübeck, Germany, December 1999.
[24] Weickert J. Efficient image segmentation using partial differential equations and morphology. Pattern Recognition 2001;34(9):1813–24.
[25] Fischer B, Modersitzki J. Fast diffusion registration. In: Nashed MZ, Scherzer O, editors. Inverse problems, image analysis, and medical imaging, Contemporary mathematics, vol. 313. Providence, RI: AMS; 2002. p. 117–27.
[26] Weickert J, Schnörr C. Variational optic flow computation with a spatio-temporal smoothness constraint. Journal of Mathematical Imaging and Vision 2001;14(3):245–55.
[27] Goldenberg R, Kimmel R, Rivlin E, Rudzsky M. Fast geodesic active contours. IEEE Transactions on Image Processing 2001;10(10):1467–75.
[28] Kühne G, Weickert J, Beier M, Effelsberg W. Fast implicit active contour models. In: Van Gool L. editor. Pattern recognition, Lecture notes in computer science, vol. 2449. Berlin: Springer; 2002. p. 133–140.
[29] Malladi R, Ravve I. Fast difference schemes for edge enhancing Beltrami flow. In: Heyden A, Sparr G, Nielsen M, Johansen P. editors. Computer vision—ECCV 2002, Lecture Notes in Computer Science, vol. 2350. Berlin: Springer; 2002. p. 343–57.
[30] Paragios N, Mellina-Gottardo O, Ramesh V. Gradient vector flow fast geodesic active contours. In: Proceedings of the Eighth International Conference on Computer Vision, vol. 1. Vancouver, Canada: IEEE Computer Society Press; 2001. p. 67–73.
[31] Weickert J. Applications of nonlinear diffusion in image processing and computer vision. Acta Mathematica Universitatis Comenianae 2001;70(1):33–50.
[32] Weickert J, Kühne G. Fast methods for implicit active contour models. In: Osher S, Paragios N, editors. Geometric level set methods in imaging, vision and graphics. New York: Springer; 2003. p. 43–58.
[33] Lu T, Neittaanmäki P, Tai X-C. A parallel splitting up method and its application to Navier–Stokes equations. Applied Mathematics Letters 1991;4(2):25–9.
[34] Temam R. Sur l'approximation de la solution des équations de Navier–Stokes par la méthode de pas fractionnaires (I). Archive for Rational Mechanics and Analysis 1969;32:135–53.
[35] Herbe C. Numerical methods for nonlinear diffusion models of sandpile growth. Master's thesis, Department of Mathematics, University of Kaiserslautern, Germany, January 1999.
[36] Weickert J, Zuiderveld KJ, ter Haar Romeny BM, Niessen WJ. Parallel implementations of AOS schemes: a fast way of nonlinear diffusion filtering. In: Proceedings of the 1997 IEEE International Conference on Image Processing, vol. 3. Santa Barbara, CA, October 1997. p. 396–9.
[37] Bruhn A, Jacob T, Fischer M, Kohlberger T, Weickert J, Brüning U, Schnörr C. Designing 3-D nonlinear diffusion filters for high performance cluster computing In: Van Gool L. editor. Pattern recognition, Lecture notes in computer science, vol. 2449. Berlin: Springer; 2002. p. 290–7.
[38] Thomas LH. Elliptic problems in linear difference equations over a network. Technical Report, Watson Scientific Computing Laboratory, Columbia University, New York, NJ, 1949.
[39] Schwarz HR. Numerical analysis: a comprehensive introduction. New York: Wiley; 1989.