Hindawi Publishing Corporation
International Journal of Biomedical Imaging
Volume 2011, Article ID 680765, 16 pages
doi:10.1155/2011/680765

Research Article

Numerical Solution of Diffusion Models in Biomedical Imaging on Multicore Processors

Luisa D'Amore,[1] Daniela Casaburi,[2] Livia Marcellino,[3] and Almerico Murli[1,2]

[1] *University of Naples Federico II, Complesso Universitario M.S. Angelo, Via Cintia, 80126 Naples, Italy*
[2] *SPACI (Southern Partnership of Advanced Computing Infrastructures), c/o Complesso Universitario M.S. Angelo, Via Cintia, 80126 Naples, Italy*
[3] *University of Naples Parthenope, Centro Direzionale, Isola C4, 80143 Naples, Italy*

Correspondence should be addressed to Luisa D'Amore, luisa.damore@unina.it

Received 1 April 2011; Revised 16 June 2011; Accepted 24 June 2011

Academic Editor: Khaled Z. Abd-Elmoniem

In this paper, we consider nonlinear partial differential equations (PDEs) of diffusion/advection type underlying most problems in image analysis. As case study, we address the segmentation of medical structures. We perform a comparative study of numerical algorithms arising from using the semi-implicit and the fully implicit discretization schemes. Comparison criteria take into account both the accuracy and the efficiency of the algorithms. As measure of accuracy, we consider the Hausdorff distance and the residuals of numerical solvers, while as measure of efficiency we consider convergence history, execution time, speedup, and parallel efficiency. This analysis is carried out in a multicore-based parallel computing environment.

1. Introduction

High-quality images are crucial to accurately diagnose a patient or determine treatment. In addition to requiring the best images possible, safety is a crucial consideration. Many imaging systems use X-rays to provide a view of what is beneath a patient's skin. X-ray radiation levels must be kept at a minimum to protect both patients and staff. As a result, raw image data can be extremely noisy. In order to provide clear images, algorithms designed to reduce noise are used to process the raw data and extract the image data while eliminating the noise. In video imaging applications, data often have to be processed at rates of 30 images per second or more. Filtering noisy input data and delivering clear, high-resolution images at these rates require tremendous computing power. This gave rise to the need of developing high-end computing algorithms for image processing and analysis which are able to exploit the high performance of advanced computing machines.

In this paper, we focus on the computational kernels which arise as basic building blocks of the numerical solution of medical imaging applications described in terms of partial differential equations (PDEs) of parabolic/hyperbolic type. Such PDEs arise from the scale-space approach for description of most inverse problems in imaging [1]. One of the main reasons for using PDEs to describe image processing applications is that PDE models preserve the intrinsic locality of many image processing operations. Moreover, we can rely on standard and up-to-date literature and software about basic computational issues arising in such case (such as the construction of suitable discretization schemes, the availability of a range of algorithmic options, and the reuse of software libraries that allow the effective exploitation of high-performance computing resources). Finally, PDEs appear to be effectively implemented on advanced computing environments [2].

We consider two standard discretization schemes of nonlinear time-dependent PDEs: semi-implicit scheme and fully implicit scheme [3]. The former leads to the solution of a linear system at each time (scale) step, while the computational kernel of the fully implicit scheme is the solution of a nonlinear system, to be performed at each time (scale) step. Taking into account that we aim to solve such problems on parallel computer in a scalable way, in the first case, we

use, as linear solver, Krylov iterative methods (GMRES) with algebraic multigrid preconditioners (AMG) [4, 5]. Regarding the fully implicit scheme, we use the Jacobian-Free Newton-Krylov (JFNK) method as nonlinear solver [6].

In recent years, multicore processors are becoming dominant systems in high-performance computing [7]. We provide a multicore implementation of numerical algorithms arising from using the semi-implicit and the implicit discretization schemes of nonlinear diffusion models underlying most problems in image analysis. Our implementation is based on parallel PETSc (Portable Extensible Toolkit for Scientific Computation) computing environment [8]. Parallel software uses a distributed memory model where the details of intercore communications and data managements are hidden within the PETSc parallel objects.

The paper is organized as follows. In Section 2, an overview of the PDE model equation used in describing some of inverse problems in imaging applications will be given. Then, the segmentation problem of medical structures is discussed. Numerical approach will be introduced in Section 3. Section 4 is devoted to the discussion of numerical algorithms based on semi-implicit and implicit numerical schemes. In Section 6, we describe the experiments that we carried out to show both the accuracy and the performance of these algorithms, while Section 7 concludes the work.

2. Diffusion Models Arising in Medical Imaging

The task in medical imaging is to provide in a noninvasive way information about the internal structure of the human body. The basic principle is that the patient is scanned by applying some sort of radiation and its interaction with the body is measured. This result is the data, whose origin has to be identified. Hence, we face an inverse problem. Most medical imaging problems lead to ill-posed (inverse) problems in the sense of Hadamard [9–11]. A standard approach for dealing with such intrinsic instability is to use additional information to construct families of approximate solution. This principle characterizes regularization methods that, starting from the milestone Tikhonov regularization [12], are now one of the most powerful tools for solution of inverse ill-posed problems. In 1992, Rudin et al. introduced the first nonquadratic regularization functional (i.e., the total variation regularization) [13] to denoise images. Moreover, the authors derive the Euler-Lagrange equations as a time-dependent PDE. In the same years Perona and Malik introduced the first nonlinear multiscale analysis [14].

Scale-space theory has been developed by the computer vision community to handle the multiscale nature of image data. A main argument behind its construction is that if no prior information is available about what are the appropriate scales for a given data set, then the only reasonable approach for a vision system is to represent the input data at multiple scales. This means that the original image $u(\mathbf{x}), \mathbf{x} \in \Re^2$ should be embedded into a one-parameter family of derived images, in which fine-scale structures are successively suppressed:

$$SS_\tau : \tau \in \Re \longrightarrow u(\mathbf{x}, \tau). \tag{1}$$

A crucial requirement is that structures at coarse scales in the multiscale representation should constitute simplifications of corresponding structures at finer scales—they should not be accidental phenomena created by the method for suppressing fine-scale structures. A main result is that if rather general conditions are imposed on the types of computations that are to be performed, then convolution by the Gaussian kernel and its derivatives is singled out as a canonical class of smoothing transformations [15, 16].

A strong relation between regularization approaches and the scale-space approach exists via the Euler-Lagrange equation of regularization functionals: it consists of a PDE of parabolic/hyperbolic (diffusion/advection) type [17], defined as follows.

Nonlinear Diffusion Models. Let $\mathbf{x} = (x, y) \in \Re^2$ and $u(\mathbf{x}, \tau)$, defined in $[0, T] \times \Omega$, be the scale-space representation of the brightness function image $u(\mathbf{x})$ defined in $\Omega \subset \Re^2$ describing the real (and unknown) object and $u_0(\mathbf{x})$ the observed image (the input data). Let us consider the following PDE problem:

$$\begin{aligned} \frac{\partial u(\mathbf{x}, \tau)}{\partial \tau} &= |\nabla u| \nabla \cdot \left(g(u) \frac{\nabla u}{|\nabla u|} \right) \quad \tau \in [0, T],\ (x, y) \in \Omega \\ u(0, \mathbf{x}) &= u_0(\mathbf{x}) \quad \tau = 0,\ (x, y) \in \Omega. \end{aligned} \tag{2}$$

$[0, T]$ is the scale (time) interval; $g(v)$ is a nonincreasing real valued function (for $v > 0$) which tends to zero as $v \to \infty$. Initial and boundary conditions will be provided according to the problem to be solved (denoising, segmentation, deblurring, registration, and so on).

Equations in (2) describe the motion of a curve (a moving front) with a speed depending on a local curvature. Such equations, known as level set equations, were first introduced in [18]. The original idea behind the level set method was a simple one. Given an interface Γ in $\Re^n$ of codimension one (i.e., its dimension is $n - 1$), bounding an (perhaps multiply connected) open region Ω, we wish to analyze and compute its subsequent motion under a velocity field $\vec{v}$. This velocity can depend on position, time, the geometry of the interface (e.g., its normal or its mean curvature), and the external physics. The idea, as devised in 1988 by Osher and Sethian, is merely to define a smooth (at least Lipschitz continuous) function $\phi(x, t)$, that represents the interface Γ as the set where $\phi(x, t) = 0$. Thus, the interface is to be captured for all later time, by merely locating the set $\Gamma(t)$ for which ϕ vanishes. The motion is analyzed by convecting the ϕ values (levels) with the velocity field v. This elementary equation is

$$\frac{\partial \phi}{\partial t} + v \cdot \nabla \phi = 0. \tag{3}$$

Actually, only the normal component of v is needed:

$$v_N = v \cdot \frac{\nabla \phi}{|\nabla \phi|}, \tag{4}$$

and the motion equation becomes

$$\frac{\partial \phi}{\partial t} + v_N \cdot |\nabla \phi| = 0. \tag{5}$$

Taking into account that the mean curvature of $\Gamma(t)$ is

$$\text{cur} = -\nabla \cdot \left(\frac{\nabla \phi}{|\nabla \phi|} \right), \tag{6}$$

equation (5) describes the motion of $\Gamma(t)$ under a speed v_N proportional to its curvature cur (Mean Curvature Motion, MCM equation) [18–20]. This basic model has received a lot of attention because of its geometrical interpretation. Indeed, the level sets of the image solution or level surfaces in 3D images move in the normal direction with a speed proportional to their mean curvature. In image processing, equations like (5) arise in nonlinear filtration, edge detection, image enhancement, and so forth, when we are dealing with geometrical features of the image-like silhouette of object corresponding to level line of image intensity function. Finally, the level set approach instead of explicitly following the moving interface itself takes the original interface Γ and embeds it in higher dimensional scalar function u, defined over the entire image domain. The interface Γ is now represented implicitly as the zeroth level set (or contour) of this function, which varies with space and time (scale) using the partial differential equation in (2), containing terms that are either hyperbolic or parabolic. The theoretical study of the PDE was done by [21] which proved existence and uniqueness of viscosity solutions.

2.1. A Case Study: Image Segmentation. In this paper, we use equations (2) for image segmentation. The task of image segmentation is to find a collection of nonoverlapping subregions of a given image. In medical imaging, for example, one might want to segment the tumor or the white matter of a brain from a given MRI image.

The idea behind level set (also known implicit active contours, or implicit deformable models) for image segmentation is quite simple. The user specifies an initial guess for the contour, which is then moved by image-driven forces to the boundaries of the desired objects. More precisely, the input to the model is a user-defined point-of-view u_0, centered in the object we are interested in segmenting. The output is the function $u(\mathbf{x}, \tau)$. Function $u(\mathbf{x}, \tau)$ in (2) is the *segmentation function,* u_0 represents the initial *contour (initial state of the segmentation function)*, and the image to segment is I^0. Moreover, as proposed in [22], instead of following evolution of a particular level set of u, the PDE model follows the evolution of the entire surface of u under speed law dependent on the image gradient, without regard to any particular level set. Suitably chosen, this flow sharpens the surface around the edges and connects segmented boundaries across the missing information. In [22, 23], the authors formalized such model as the *Riemannian mean curvature flow* where the variability in the parameter ϵ also improves the segmentation process and provides a sort of regularization. Thus, (2) becomes

$$\begin{aligned} \frac{\partial u(\mathbf{x}, \tau)}{\partial \tau} &= \sqrt{\epsilon^2 + |\nabla u|^2} \nabla \cdot \left(g(|\nabla I^0|) \frac{\nabla u}{\sqrt{\epsilon^2 + |\nabla u|^2}} \right) \\ &\qquad \tau \in [0, T],\ \mathbf{x} = (x, y) \in \Omega \\ u(\mathbf{x}, 0) &= u_0(\mathbf{x}) \quad \tau = 0,\ \mathbf{x} = (x,\ y) \in \Omega \\ u(\tau,\ \mathbf{x}) &= 0 \quad \tau \in [0, T],\ \mathbf{x} = (x, y) \in \partial\Omega \end{aligned} \tag{7}$$

accompanied with initial condition u_0 and zero Dirichlet boundary conditions. Regarding u_0, it is usually defined as a circle completely enclosed inside the region that one wish to segment.

The term $g(v)$, called *edge detector*, is a nonincreasing real function such that $g(v) \to 0$ while $v \to \infty$, and it is used for the enhancement of the edges. Indeed, it controls the speed of the diffusion/regularization: if ∇u has a small mean in a neighborhood of a point $\mathbf{x}$, this point $\mathbf{x}$ is considered the interior point of a smooth region of the image and the diffusion is therefore strong. If ∇u has a large mean value on the neighborhood of $\mathbf{x}$, $\mathbf{x}$ is considered an edge point and the diffusion spread is lowered, since $g(v)$ is small for large v. A popular choice in nonlinear diffusion models is the Perona and Malik function [14]: $g(v) = 1/(1 + v^2/\beta)$, $\beta > 0$. In many models, the function $g(|\nabla u|)$ is replaced by its smoothed version $g(|\nabla G_\sigma * u|)$, where G_σ is a smoothing kernel, for example, the Gauss function, which is used in presmoothing of image gradients by the convolution. For shortening notations, we will use abbreviation

$$g = g(|\nabla G_\sigma * u|). \tag{8}$$

In conclusion, we use the *Riemannian mean curvature flow*, as model equation of the segmentation of medical structures: given I_0, the initial image and u_0 equals to a circle contained inside an object of the image I_0, we are interested in segmenting, we compute $u(\mathbf{x}, \tau)$ by solving (7). The level sets of $u(\mathbf{x}, \tau)$, at steady state, provide approximations of the contour to detect.

3. Numerical Schemes

Nonlinear PDE in (7) can be expressed in a compact way as

$$\frac{\partial u(x, y, \tau)}{\partial \tau} = F[u(x, y, \tau, \nabla u(x, y, \tau), I_0)], \tag{9}$$

where

$$F = \sqrt{\epsilon^2 + |\nabla u|^2} \nabla \cdot \left(g(|\nabla I^0|) \frac{\nabla u}{\sqrt{\epsilon^2 + |\nabla u|^2}} \right). \tag{10}$$

Scale Discretization. That is discretization with respect to τ. If $[0, T]$ is the scale interval and *nscales* is the number of scale steps, we denote by τ_i the ith scale-step for all $i = 1, \ldots, nscales$, so that $\tau_{i+1} = \tau_i + \Delta\tau$, where $\Delta\tau = T/nscales$ is the step-size.

Using the Euler forward finite difference scheme to discretize the scale derivative on the left hand side of (9), we get

$$\frac{u(x, y, \tau_i) - u(x, y, \tau_{i-1})}{\Delta\tau} = F[u(x, y, \tau)] \tag{11}$$

or, equivalently

$$u(x, y, \tau_i) = u(x, y, \tau_{i-1}) + \Delta\tau \cdot F[u(x, y, \tau)]. \tag{12}$$

Let us denote as $u_i = u(x, y, \tau_i)$, $i = 1, \ldots, nscales$, the function u evaluated at τ_i. Equation (12) is rewritten as

$$u_i = u_{i-1} + \Delta\tau \cdot F(u(x, y, \tau)) \equiv G[u(x, y, \tau)]. \tag{13}$$

Depending on the collocation value, used to evaluate $u(x, y, \tau)$ with respect to the parameter τ, inside the F function on the right hand side of (12) three iterative schemes derive:

(i) *explicit scheme*: $u_i = G[u_{i-1}]$, that is, the function F is evaluated at $u_{i-1} = u(x, y, \tau_{i-1})$;

(ii) *semi-implicit scheme*: $u_i = G[u_{i-1}, u_i]$, that is, we use u_i to discretize the numerator $|\nabla u|$ of the fraction $\nabla u/\sqrt{\epsilon^2 + |\nabla u|^2}$. Other quantities are evaluated at u_{i-1};

(iii) *implicit scheme*: $u_i - G[u_{(i)}] = 0$, that is, the function F is evaluated at u_i.

In summary, the difference between the semi-implicit and the implicit scheme relies on the scale discretization of the term $|\nabla u|$ at the numerator of $\nabla u/\sqrt{\epsilon^2 + |\nabla u|^2}$ inside the function F. This term controls the diffusion process, and it plays the role of *edge-enhancement*. If we consider the three-dimensional (3D) domain $\Omega_T = \Omega \times [0, T]$, the semi-implicit scheme employs a sort of 2D + 1 discretization of Ω_T proceeding along *nscales* two dimensional (2D) slices each one obtained at $\tau \equiv \tau_i$, while the fully implicit scheme uses a fully 3D discretization of Ω_T. This difference suggests that the fully implicit scheme may provide a more accurate edge detection than the semi-implicit scheme. This difference is highlighted by considering their discretization errors.

Space Discretization. That is discretization with respect to (x, y). If Ω is the space domain, we introduce a rectangular uniform grid on Ω consisting of $N_x \times N_y$ (for simplicity we assume that Ω is a rectangular of dimension 1×1; this means that $h_x = 1/N_x$ and $h_y = 1/N_y$), nodes $(x_i, y_j) = (l\Delta x, m\Delta y)$, $l = 1, \ldots, N_x$, $m = 1, \ldots, N_y$, and we use finite volumes to discretize the partial derivatives of u, as in [24, 25].

Scale-Space Discretization. Let

$$u_i^{l,m} = u(x_l, y_m, \tau_i) \in \Re^{N_x \times N_y \times nscales} \tag{14}$$

be the vector obtained from the scale-space discretization of the function u, we have the following iteration formulas.

(i) Explicit scheme:

$$\frac{u_i^{l,m} - u_{i-1}^{l,m}}{\Delta\tau} = \sqrt{\epsilon^2 + \left|\nabla u_{i-1}^{l,m}\right|^2}\nabla \cdot \left(g(|\nabla I^0|)\frac{\nabla u_{i-1}^{l,m}}{\sqrt{\epsilon^2 + \left|\nabla u_{i-1}^{l,m}\right|^2}}\right) \Longleftrightarrow \tag{15}$$
$$u_i^{l,m} = \left(I + \Delta\tau[A]_{i-1}^{l,m}\right)u_{i-1}^{l,m} \quad \forall i = 1, 2, \ldots, N_E,$$

where, for each i, the matrix $[A]_{i-1}^{l,m} \in \Re^{N_x^2 \times N_y^2}$ and $I \in \Re^{N_x^2 \times N_y^2}$ is the unit matrix, while N_E is the scale steps number.

(i) Semi-implicit scheme:

$$\frac{u_i^{l,m} - u_{i-1}^{l,m}}{\Delta\tau} = \sqrt{\epsilon^2 + \left|\nabla u_{i-1}^{l,m}\right|^2}\nabla \cdot \left(g(|\nabla I^0|)\frac{\nabla u_i^{l,m}}{\sqrt{\epsilon^2 + \left|\nabla u_{i-1}^{l,m}\right|^2}}\right) \Longleftrightarrow$$
$$u_i^{l,m} = u_{i-1}^{l,m} + \Delta\tau[A]_{i-1}^{l,m}u_i^{l,m} \Longleftrightarrow$$
$$\left(I + \Delta\tau[A]_{i-1}^{l,m}\right)u_i^{l,m} = u_{i-1}^{l,m} \quad \forall i, i = 1, \ldots, N_{\mathrm{SI}} \tag{16}$$

where, for each i, the matrix $[A]_{i-1}^{l,m} \in \Re^{N_x^2 \times N_y^2}$ and $I \in \Re^{N_x^2 \times N_y^2}$ is the unit matrix and N_{SI} is the scale steps number.

(ii) Fully-implicit scheme:

$$\frac{u_i^{l,m} - u_{i-1}^{l,m}}{\Delta\tau} = \sqrt{\epsilon^2 + \left|\nabla u_i^{l,m}\right|^2}\nabla \cdot \left(g(|\nabla I^0|)\frac{\nabla u_i^{l,m}}{\sqrt{\epsilon^2 + \left|\nabla u_i^{l,m}\right|^2}}\right) \Longleftrightarrow \tag{17}$$
$$u_i^{l,m} = u_{i-1}^{l,m} + \Delta\tau[A]_i^{l,m}\left(u_i^{l,m}\right) \quad \forall i = 1, \ldots, N_I,$$

where N_I is the scale steps number and $[A]_i^{l,m}$, for each i, is a nonlinear vector operator on $\Re^{N_x^2 \times N_y^2}$, depending on $u_i^{l,m}$.

In particular, we apply the Crank-Nicholson scheme [3] which uses the average of the forward Euler method at step $i-1$ and the backward Euler method at step i:

$$\begin{aligned}\frac{u_i^{l,m}-u_{i-1}^{l,m}}{\Delta\tau} &= \frac{1}{2}\left[\sqrt{\epsilon^2+\left|\nabla u_i^{l,m}\right|^2}\nabla\right.\\ &\quad\left.\cdot\left(g(|\nabla I^0|)\frac{\nabla u_i^{l,m}}{\sqrt{\epsilon^2+\left|\nabla u_i^{l,m}\right|^2}}\right)\right]+\frac{1}{2}\cdots\\ &\quad\cdots\left[\sqrt{\epsilon^2+\left|\nabla u_{i-1}^{l,m}\right|^2}\nabla\right.\\ &\quad\left.\cdot\left(g(|\nabla I^0|)\frac{\nabla u_{i-1}^{l,m}}{\sqrt{\epsilon^2+\left|\nabla u_{i-1}^{l,m}\right|^2}}\right)\right],\\ &\qquad\forall i=1,\ldots,N_I\\ \Longleftrightarrow\ u_i^{l,m} &= u_{i-1}^{l,m}+\Delta\tau\left[B_{i-1}^{l,m}u_{i-1}^{l,m}+[A]_i^{l,m}\left(u_i^{l,m}\right)\right],\\ &\qquad\forall i=1,\ldots,\mathrm{N}_I,\end{aligned}\tag{18}$$

where $B_{i-1}^{l,m}$ is a matrix on $\Re^{N_x^2\times N_y^2}$, depending on $u_{i-1}^{l,m}$, and $[A]_i^{l,m}$ is a nonlinear vector operator on $\Re^{N_x^2\times N_y^2}$, depending on $u_i^{l,m}$.

4. Algorithms and Their Computational Complexity

The effectiveness of these schemes depends on a suitable balance between accuracy (scale-space discretization error), number of flop/s per iteration (algorithm complexity), and the total execution time needed to reach a prescribed accuracy (software performance).

Let us denote the discretization error E_d. It is

$$E_d = O\left(h_x^p\right)+O\left(h_y^p\right)+O(\Delta\tau^q).\tag{19}$$

Explicit scheme is accurate at the first order both with respect to scale and space, that is, $p=q=1$; anyway, it is the one straightforwardly computable. The computational kernel is a matrix-vector product, at every scale step. This scheme requires very small time steps in order to be stable (CFL (Courant-Friedrich-Levy) condition that guarantees the stability of the evolution), and its use is limited rather by its stability than accuracy. This constraint is practically very restrictive, since it typically leads to the need for a huge amount of iterations [3]. Semi-implicit scheme is absolutely stable for all scale steps. The accuracy, in terms of discretization error with respect to both scale and space, is of the first order, because $p=1$, $q=2$ [24, 25]. Crank-Nicholson provides a discretization error of second order, that is, $p=q=2$, but it requires extra computations, leading to a nonlinear system of equations, at every time step, while stability is ensured for all scale steps [3]. In the following, we collect these results:

$$\begin{aligned}&\text{explicit scheme: } p=q=1,\\ &\text{semi-implicit scheme: } p=1,\ q=2,\\ &\text{implicit scheme: } p=q=2.\end{aligned}\tag{20}$$

Then, the fully implicit scheme provides an order of accuracy greater than that provided by the others. This difference may be important in those applications of image analysis where the edges are fundamental to recognize some pathologies.

Algorithm complexity of these schemes depends on the choice of the numerical solver. Concerning the semi-implicit scheme, we employ Krylov subspace methods, which are the most effective approaches for solving large linear systems [5]. In particular, we use Generalized Minimal RESidual method (GMRES) equipped with Algebraic multigrid (AMG) preconditioner. Such techniques are convenient because they require as input only the system matrix corresponding to the finest grid. In addition, they are suitable to implement in a parallel computing environment. For the fully implicit scheme we use the Jacobian Free Newton Krylov Method (JFNK) [6]. JFNK methods are synergistic combinations of Newton-type methods for superlinearly convergent solution of nonlinear equations and Krylov subspace methods for solving the Newton correction equations. The link between the two methods is the Jacobian-vector product, which may be probed approximately without forming and storing the elements of the true Jacobian.

Let us briefly describe the numerical algorithms that we are going to implement, which are based on the semi-implicit and the implicit discretization schemes, together with their complexity.

Algorithm SI (Semi-Implicit Scheme). For all $i=1,\ldots,N_{\mathrm{SI}}$ solution of

$$\left(I+\Delta\tau[A]_{i-1}^{l,m}\right)u_i^{l,m}=u_{i-1}^{l,m}\Longleftrightarrow HS_{i-1}^{l,m}u_i^{l,m}=u_{i-1}^{l,m},\tag{21}$$

with respect to $u_i^{l,m}$. $HS_{i-1}^{l,m}$, for each i, is a matrix $\in\Re^{N_x^2\times N_y^2}$. As space derivative we use the 2nd order finite covolume discretization scheme (see [24, 25] for convergence, consistence and stability). By this way, matrix $[A]_{i-1}^{l,m}$ is a block pentadiagonal matrix with tridiagonal blocks along the main diagonal and diagonal blocks along the upper and lower diagonals.

Algorithm I (Implicit Scheme). For all $i=1,\ldots,N_I$ solution of

$$\begin{aligned}u_i^{l,m} &= u_{i-1}^{l,m}+\Delta\tau\left[B_{i-1}^{l,m}u_{i-1}^{l,m}+[A]_i^{l,m}\left(u_i^{l,m}\right)\right]\\ &\Longleftrightarrow HI_i^{l,m}\left(u_i^{l,m},u_{i-1}^{l,m}\right)=0,\end{aligned}\tag{22}$$

with respect to $u_i^{l,m}$. $HI_i^{l,m}$, for each i, is a nonlinear vector operator on $\Re^{N_x^2\times N_y^2}$.

Compute $r_0 = P(b - Ax_0)$, $\beta := \|r_0\|_2$ and $v_1 := r_0/\beta$
Define the $(m+1) \times mH_k = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$. Set $H_m = 0$.
for $k = 1$ to m do
  Compute $w_j := PAv_j$
  For $i = 1$ to k do:
  $h_{ij} := (w_j, v_i)$
  $w_j := w_j - h_{ij}v_i$
end for
  $h_{j+1,j} = \|w_j\|_2$. If $h_{j+1,j} = 0$ set $m := j$ and go to 12
  $v_{j+1} = w_j/h_{j+1,j}$
end for
Compute y_m the minimiser of $\|\beta e_1 - H_m y\|_2$
Set $x_m := x_0 + V_m y_m$.

Algorithm 1: Preconditioned GMRES for solving a linear system $Ax = b$. Input: A (matrix coefficient), b (right hand side), P (preconditioner). Output: x_m, approximate solution at the mth step. For all iteration, a matrix-vector product is required.

Algorithm SI. For each scale step, to solve the linear system (21), we employ GMRES iterative method (see Algorithm 1). Computational kernel of GMRES is a matrix-vector product. Taking into account the structure of the coefficient matrix (we assume that $N_x = N_y = N$, then $h = 1/N = h_x = h_y$), the computational cost of GMRES is

$$T_{\text{GMRES}}(N^2) = O\left(k^{\text{SI}}_{\text{GMRES}} \cdot 5N^2\right), \tag{23}$$

where $k^{\text{SI}}_{\text{GMRES}}$ is the maximum iterations of GMRES (over the scale steps). Computational complexity of Algorithm SI is

$$T_{\text{SI-GMRES}}(N^2) = O\left(N_{\text{SI}} \cdot k^{\text{SI}}_{\text{GMRES}} \cdot 5N^2\right). \tag{24}$$

Algebraic multigrid (AMG) method follows the main idea of (geometric) multigrid (MG), where a sequence of grids is constructed from the underlying geometry with corresponding transfer operators between the grids [26]. The main idea of MG is to remove the smooth error, that cannot be eliminated by relaxation on the fine grid, by coarse-grid correction. The solution process then as usual consists of presmoothing, transfer of residuals from fine to coarse grids, interpolation of corrections from coarse to fine levels, and optional postsmoothing. In contrast to geometric multigrid, the idea of AMG is to define an artificial sequence of systems of equations decreasing in size. We call these equations *coarse-grid* equations. The interpolation operator $P^{l,m}_{lv}$ and the restriction operator $R^{l,m}_{lv}$ define the transfer from finer to coarser grids and vice versa. Finally, the operator on the coarser grid at level $lv + 1$ is defined by

$$A^{l,m}_{lv+1} = R^{l,m}_{lv} A^{l,m}_{lv} P^{l,m}_{lv}. \tag{25}$$

The AMG method consists of two main parts, the setup phase and the solution phase. During the setup phase, the coarse-grids and the corresponding operators are defined. The solution phase consists of a multilevel iteration. The number of recursive calls, which is the number of levels lv, depends on the size and structure of the matrix. For our case, we use the V-cycle pattern with the FALGOUT-CLJP coarse grid selection [27]. Looking at the Algorithm SI in Table 1, the preconditioner P is just A_{lv+1}.

Table 1: Comparisons between two algorithms: $E_d = O(10^{-1})$.

Algorithm	$\Delta\tau$	Number of scale steps	d_H	Execution time (secs)
SI	0.16	3	0.9492	9.262
I	0.4	1	0.9498	7.675

Computational cost of each iteration of GMRES is that of the AMG preconditioner plus the matrix-vector products:

$$T_{\text{AMG+GMRES}}(N^2) = O\left(k^{\text{SI}}_{\text{GMRES}}\left(\underbrace{T_{\text{AMG}}(lv)}_{lv \cdot N^2} + 5N^2\right)\right), \tag{26}$$

then, we get:

$$T_{\text{SI+AMG+GMRES}}(N^2) = O\left(N_{\text{SI}} k^{\text{SI}}_{\text{GMRES}} lv N^2\right). \tag{27}$$

Following picture shows a schematic description of Algorithm SI that emphasizes its main steps and the most time consuming operation, that is, the matrix vector products needed at each step of GMRES.

Algorithm I. For each scale step, to solve the nonlinear equations (22), we employ the Jacobian-Free Newton-Krylov (JFNK) method. JFNK is a nested iteration method consisting of at least two and usually four levels. The primary levels, which give the method its name, are the loop over the Newton method:

$$HI^{l,m}_i\left(u^{l,m}_{n+1}\right) = 0 \Longleftrightarrow HI^{l,m}_{\text{i}}\left(u^{l,m}_n\right) + J_i\left(u^{l,m}_n\right)\left(u^{l,m}_{n+1} - u^{l,m}_n\right) = 0, \tag{28}$$

and the loop building up the Krylov subspace out of which each Newton step is drawn:

$$J_i\left(u^{l,m}_n\right)\delta u^{l,m}_n = -HI^{l,m}_i\left(u^{l,m}_n\right), \qquad u^{l,m}_{n+1} = u^{l,m}_n + \delta u^{l,m}_n, \tag{29}$$

```
for i = 1 to N_SI do
   for k = 1 to k_GMRES do
      for lv = 1 to levels_AMG do
         matrix-vector products involving HS_i^{l,m}, A_{lv+1}^{l,m} and u_i^{l,m}(k, lv)
end for
```

Algorithm 2: Sketch of Algorithm SI.

```
for i = 1 to N_I do
   for n = 1 to N_New do
      for k = 1 to k_GMRES do
         evaluate HI_i^{l,m}, u_i^{l,m}(n, k)
end for
```

Algorithm 3: Sketch of Algorithm I.

Outside of the Newton loop, a globalization method is often required. We use line search.

Forming each element of J which is a matrix of dimension $N^2 \times N^2$ requires taking derivatives of the system of equations with respect to u. This can be time consuming. Using the first order Taylor series expansion of $HI_i^{l,m}(u_n^{l,m} + \rho v)$, it follows that

$$J_i\left(u_n^{l,m}\right)\delta u_n^{l,m} = \frac{\left[HI_i^{l,m}\left(u_n^{l,m} + \rho\delta u_n^{l,m}\right) - HI_i^{l,m}\left(u_n^{l,m}\right)\right]}{\rho} + O(\rho^2), \tag{30}$$

where ρ is a perturbation. JFNK does not require the formation of the Jacobian matrix; it instead forms a result vector that approximates this matrix multiplied by a vector. This *Jacobian-free* approach has the advantage to provide the quadratic convergence of Newton method without the costs of computing and storing the Jacobian. Conditions are provided on the size of ρ that guarantee local convergence.

Algorithm 3 shows a schematic description of Algoritm I that emphasizes its main steps and the most time consuming operation, that is, evaluations of the nonlinear operator $HI_i^{l,m}$ at each innermost step.

Algorithm complexity of JFNK is

$$T_{\text{JFNK}}(N^2) = O\left(N_{\text{New}} k^I_{\text{GMRES}}\left[f + O(N^2)\right]\right), \tag{31}$$

where N_{New} is the maximum number of Newton's steps, over the scale steps, k^I_{GMRES} is the maximum number of GMRES iterations (computed over Newton's steps and scale steps)), f is the number of evaluations of $HI_i^{l,m}$. Finally, we get

$$T_{I+\text{JFNK}}(N^2) = O\left(N_I N_{\text{New}} k^I_{\text{GMRES}}\left[f + O(N^2)\right]\right). \tag{32}$$

A straightforward comparison between the algorithm complexity of these algorithms shows that Algorithm SI asymptotically seems to be comparable with respect to Algorithm SI. Of course, the performance analysis must also take into account the efficiency of these two schemes in a given computing environment. Next section describes the PETSc-based implementation of these algorithms that we have developed in a multicore computing environment.

5. The Multicore-Based Implementation

The software has been developed using the high-level software library PETSc (Portable Extensible Toolkit for Scientific Computations) (release 3.1, March 2010) [8]. PETSc provides a suite of data structures and routines as building blocks for the implementation of large-scale codes to be used in scientific applications modeled by partial differential equation. PETSc is flexible: its modules, that can be used in application codes written in Fortran, C, and C++, are developed by means of object-oriented programming techniques.

The library has a hierarchical structure: it relies on standard basic computational (BLAS, LAPACK) and communication (MPI) kernels and provides mechanism needed to write parallel application codes. PETSc transparently handles the moving of data between processes without requiring the user to call any data transfer function. This includes handling parallel data layouts, communicating ghost points, gather, scatter and broadcast operations. Such operations are optimized to minimize synchronization overheads.

Our parallelization strategy is based on domain decomposition: in particular, we adopt the *row-block data distribution*, which is the standard PETSc data distribution. Row-block data distribution means that blocks of dimensions $N^2/p \times N$ of contiguous rows of matrices of dimension $N^2 \times N^2$ are distributed among contiguous processes. By the same way, vectors of size N are distributed among p processors as blocks of size N/P. Such partitioning has been chosen because overheads, due to redistribution before the solution of the linear systems, are avoided. Further, row-block data distribution introduces a coarse grain parallelism which is best oriented to exploit concurrency of multicore multiprocessors because it does not require a strong cooperation among computing elements: each computing element has to locally manage the blocks that are assigned to it.

The computing platform that we consider is made of 16 blades (1 blade consisting of 2 quad core Intel Xeon E5410@2.33 GHz) Dell PowerEdge M600, equipped with IEEE double precision arithmetic. Because high performance technologies can be employed in medical applications only to the extent that the overall cost of the infrastructure is

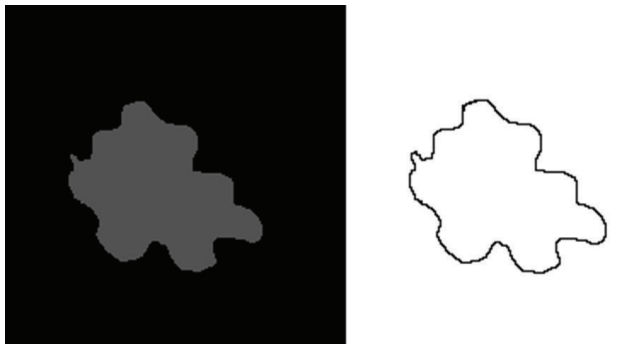

Figure 1: Test 1. Image size: 840 × 840, simulated image and its contour to compute.

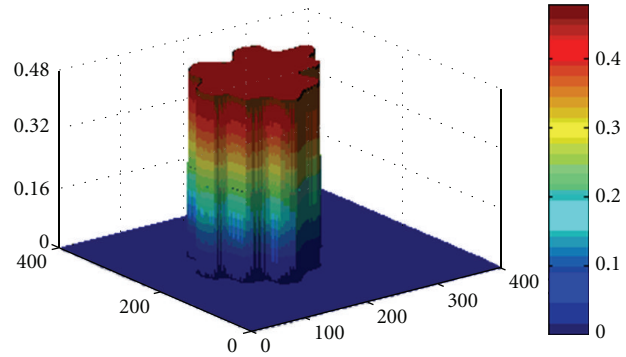

Figure 2: Test 1: The 3D visualization of the segmentation function $u(\tau, \mathbf{x})$ at steady state $T = 0.4$.

affordable, and because we consider single images of medium size, we show results obtained by using 1 blade, that is, we run the parallel algorithms on up to $p = 8$ cores of a single blade. Of course, in case of multiple images or sequences of images, the use of a greater number of cores may be interesting.

6. Experiments

In this section we present and discuss computational results obtained by implementing Algorithm I and Algorithm SI in a multicore parallel computing machine. Before illustrating experimental results, let us briefly describe the choice of

(1) test images,

(2) comparison criteria,

(3) parameters selection.

(1) Test Images. we have carried out many experiments in order to analyze the performance of these algorithms. Here we show results concerning the segmentation of a (malignant) melanoma (see Figure 7) [28]. Epiluminescence microscopy (ELM) has proven to be an important tool in the early recognition of malignant melanoma [29, 30]. In ELM, halogen light is projected onto the object, thus rendering the surface translucent and making subsurface structures visible. As an initial step, the mask of the skin lesion is determined by a segmentation algorithm. Then, a set of features containing shape and radiometric features as well as local and global parameters is calculated to describe the malignancy of the lesion. In order to better validate computed results and to analyze the software performance, we first consider a synthetic test image simulating the object we are interested in segmenting (see Figure 1).

(2) Comparison Criteria. We compare the algorithms using the following criteria:

(a) distance from original solution. As measure of the difference between two curves, we use the Hausdorff distance measured between the computed curve and the original one. It is well known that the Hausdorff distance is a metric over the set of all closed bounded sets (see [31]), here we restrict ourselves to finite point sets because that is all that is necessary for segmentation algorithms [32]. Given two finite point sets C_1 and C_2, the Hausdorff distance d_H between the sets C_1 and C_2 is defined as follows:

$$d_H(C_1, C_2) = \max\{h(C_1, C_2), h(C_2, C_1)\},$$
$$h(C_1, C_2) = \max_{a \in C_1} \min_{b \in C_2} \|a - b\|, \tag{33}$$

where $\|\cdot\|$ is the euclidean norm. It identifies the point $a \in C_1$ that is fastest from any point of C_2 and vice versa, then it keeps the maximum,

(b) efficiency: execution time of (serial) algorithms,

(c) convergence history: behavior of residuals and iteration numbers of inner solvers,

(d) Parallel performance: execution time, speedup, and efficiency versus cores number.

(3) Parameters Selection. We set $K = 1.0$ and $\epsilon = 1.0$. Let us explain how we select the values of the scale step size and the number of scale steps. Regarding $\Delta\tau$, its value is chosen according to that required to E_d. Taking into account that, in Algorithm SI, E_d is accurate at the first order with respect to $\Delta\tau_{\mathrm{SI}}$, while it is accurate at the second order with respect to $\Delta\tau_I$, in Algorithm I, by requiring that the discretization error is about the same, we get

$$E_d = O(\Delta\tau_{\mathrm{SI}}) = O(\Delta\tau_I^2) \Longrightarrow \Delta\tau_{\mathrm{SI}} = \sqrt{\Delta\tau_I}. \tag{34}$$

Finally, in Algorithm SI, the stopping criterion of the linear solver (GMRES) uses the tolerance

$$\mathrm{TOL} = 10^{-10}, \tag{35}$$

while the preconditioner AMG uses the tolerance $\mathrm{TOL} = 10^{-7}$ and 25 as maximum number of AMG-levels. In the Algorithm 2, the stopping criterion of the nonlinear solver uses the tolerance

$$\mathrm{TOL} = 10^{-10}. \tag{36}$$

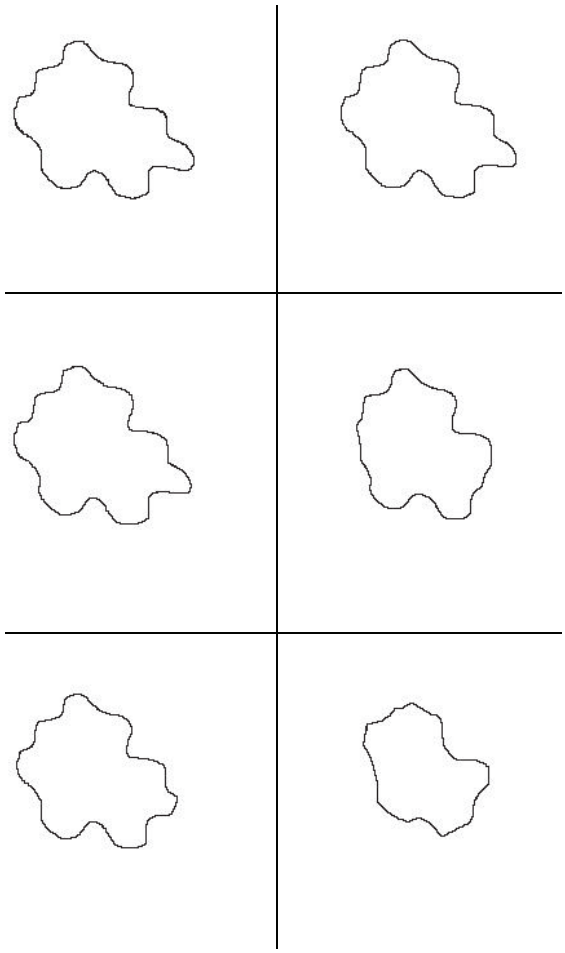

Figure 3: Test 1: Comparisons between segmentation results. On the left Algorithm I. On the right Algorithm SI. First row: $\Delta\tau_I = 0.4$, $\Delta\tau_{\mathrm{SI}} = 0.16$. Second row: $\Delta\tau_I = 0.04$, $\Delta\tau_{\mathrm{SI}} = 0.16$. Third row: $\Delta\tau_I = 0.004$, $\Delta\tau_I = 0.0016$.

Regarding the number of scale steps (N_{SI} and N_I), taking into account that

$$N_{\mathrm{SI},I} = \frac{T}{\Delta\tau_{\mathrm{SI},I}}, \tag{37}$$

its choice depends on $\Delta\tau_{\mathrm{SI},I}$ and on the value of $\tau \equiv T$, that is, the value of the scale parameter corresponding to steady state of the segmentation function $u(\mathbf{x}, \tau)$, solution of the PDE model. To check the steady state, we require that the residuals, corresponding to different scale steps, reach the tolerance

$$\mathrm{TOL} = 10^{-9}. \tag{38}$$

We found that this corresponds to $T = 0.4$ (see Figure 2) for Test 1 and to $T = 2$ for Test 2.

Test 1: Synthetic Image. In Tables 1, 2, and 3, we show the Hausdorff distance and execution time by requiring that discretization error is of the first, second, and third order, that is, $E_d = O(10^{-1})$, $E_d = O(10^{-2})$, and $E_d = O(10^{-3})$,

Table 2: Comparisons between two algorithms: $E_d = O(10^{-2})$.

Algorithm	$\Delta\tau$	Number of scale steps	d_H	Execution time (secs)
SI	0.016	25	0.6412	61.487
I	0.04	10	0.9498	71.67

Table 3: Comparisons between two algorithms: $E_d = O(10^{-3})$.

Algorithm	$\Delta\tau$	Number of scale steps	d_H	Execution time (secs)
SI	0.0016	250	0.5993	356.926
I	0.004	100	0.9492	356.335

respectively. Hence, we get the following values of the scale step size:

$$\begin{aligned} \Delta\tau_I = 0.4 &\Longrightarrow \Delta\tau_{\mathrm{SI}} = 0.16, \\ \Delta\tau_I = 0.04 &\Longrightarrow \Delta\tau_{\mathrm{SI}} = 0.016, \\ \Delta\tau_I = 0.004 &\Longrightarrow \Delta\tau_{\mathrm{SI}} = 0.0016. \end{aligned} \tag{39}$$

Moreover, concerning the number of scale steps, it follows that

$$\begin{aligned} E_d = O(10^{-1}) &\Longrightarrow N_I = \frac{0.4}{0.4} = 1, \quad N_{\mathrm{SI}} = \mathrm{int}\left[\frac{0.4}{0.16}\right] = 3, \\ E_d = O(10^{-2}) &\Longrightarrow N_{\mathrm{SI}} = \frac{0.4}{0.04} = 10, \quad N_{\mathrm{SI}} = \frac{0.4}{0.016} = 25, \\ E_d = O(10^{-3}) &\Longrightarrow N_{\mathrm{SI}} = \frac{0.4}{0.04} = 100, \quad N_{\mathrm{SI}} = \frac{0.4}{0.016} = 250. \end{aligned} \tag{40}$$

Note that while in the first case, that is, if we require $E_d = O(10^{-1})$, these two algorithms are quite numerically equivalent, both in terms of execution time and of the computed result; as discretization error decreases, Algorithm I appears to be more robust than Algorithm SI, in the sense that Algorithm I reaches the steady state with high accuracy (the Hausdorff distance is of 95%), while the computed results of Algorithm SI are less accurate, even though the execution time of Algorithm I sometimes slightly increases. These results suggest that if it needs to get an accurate and reliable result, Algorithm I should be preferable. Figure 3 show segmentation results. Finally, note that the execution time of these two algorithms asymptotically is the same, as stated by the analysis of computational cost carried on in Section 4.

Convergence History. Convergence history is illustrated by showing the behavior of relative residuals versus the scale steps (see Figures 4, 5, and 6), and by reporting iteration number of the GMRES and of Newton's method, respectively, (see Tables 4, 5, 6, and 7). We consider $\Delta\tau_{\mathrm{SI}} = 0.16$, 0.016, and $\Delta\tau_I = 0.04$.

In the following, we show results corresponding to the segmentation of a melanoma (see Figure 7). As expected, because this is a real image, the steady state is reached at a

Table 4: Convergence history of Algorithm SI. $\Delta\tau_{\mathrm{SI}} = 0.16$. First column reports the scale step number, second column reports the number of GMRES iterations at each scale step, and last column reports the maximum number of AMG levels at each GMRES iteration.

i	k^i_{gmres}	lv^i
1	8	6
2	6	5
3	5	3

Table 5: Convergence history of Algorithm SI. $\Delta\tau_{\mathrm{SI}} = 0.016$. On the first column, we denote the scale step number, and the symbol $p - q$ is used to denote the steps ranging from the p-th until to the q-th. Second column reports the number of GMRES iterations at each scale step, third column reports the maximum number of AMG levels at each GMRES iteration.

i	k^i_{gmres}	lv^i
1	5	6
2-3	5	5
4	5	4
5–7	4	4
8-8	4	3
10–13	3	3
14–20	3	2
21-22	3	1
23–25	2	1

Table 6: Convergence history of Algorithm SI. $\Delta\tau_I = 0.4$. First column reports the scale step number, second column reports the number of Newton's steps at each scale step, and last column reports the number of GMRES iterations at each Newton's step.

i	N^i_{New}	k^i_{GMRES}
1	10	9, 9, 8, 8, 8, 7, 7, 7, 6, 6

Table 7: Convergence history of Algorithm SI. $\Delta\tau_I = 0.04$. First column reports the scale step number, second column reports the number of Newton's steps at each scale step, and last column reports the number of GMRES iterations at each Newton's step.

i	N^i_{New}	k^i_{GMRES}
1	10	9, 9, 8, 8, 7, 7, 7, 7, 6, 6
2	10	9, 8, 8, 8, 7, 7, 7, 6, 6, 6
3	9	9, 8, 8, 7, 7, 6, 6, 6, 6
4	9	8, 8, 8, 7, 7, 6, 6, 6, 5
5	9	8, 8, 7, 7, 7, 6, 6, 5, 5
6	7	8, 7, 7, 7, 6, 5, 5
7	7	7, 7, 7, 7, 6, 5, 5
8	6	7, 7, 6, 6, 6, 5
9	6	7, 7, 6, 6, 5, 5
10	5	7, 6, 6, 5, 4

scale greater than that of the synthetic test image, that is, $T = 2$, thus both algorithms require a greater number of scale steps to the reach the steady state. Tables 8, 9, and 10, and Figure 8, compare results of Algorithm SI and Algorithm I.

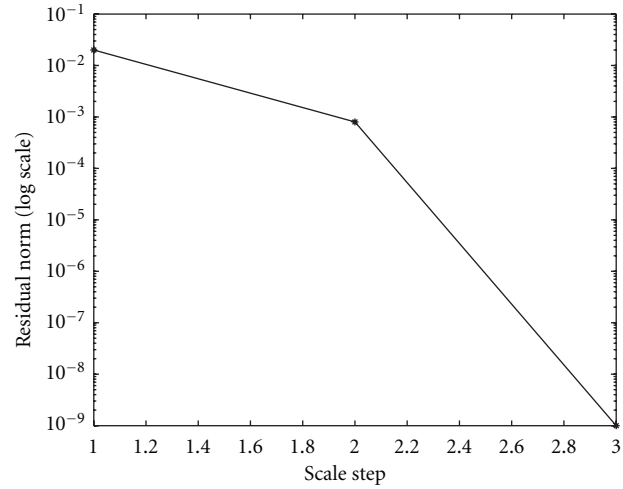

Figure 4: Algorithm SI. Behavior of the relative residual $\|r_i\|_2/\|r_0\|_2$ versus 3 scale steps. $\Delta\tau_{\mathrm{SI}} = 0.16$.

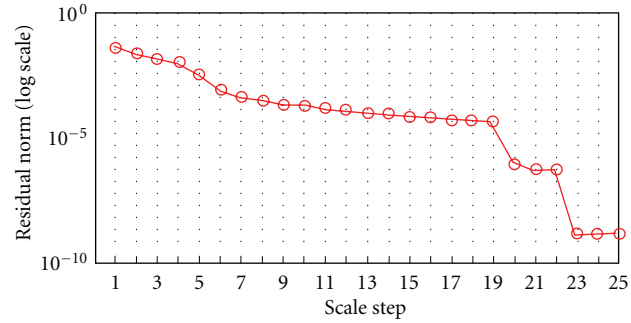

Figure 5: Algorithm SI. Behavior of the relative residual $\|r_i\|_2/\|r_0\|_2$ versus 25 scale steps. $\Delta\tau_{\mathrm{SI}} = 0.016$.

Table 8: Test 2: Comparisons between two algorithms: $E_d = O(10^{-1})$.

Algorithm	$\Delta\tau$	Number of scale steps	Execution time (secs)
SI	0.16	13	41.125
I	0.4	5	47.893

Table 9: Test 2: Comparisons between two algorithms: $E_d = O(10^{-2})$.

Algorithm	$\Delta\tau$	Number of scale steps	Execution time (secs)
SI	0.016	125	417.56
I	0.04	50	455.33

Table 10: Test 2: Comparisons between two algorithms: $E_d = O(10^{-3})$.

Algorithm	$\Delta\tau$	Number of scale steps	Execution time (secs)
SI	0.0016	1250	4828.5
I	0.004	500	4663.7

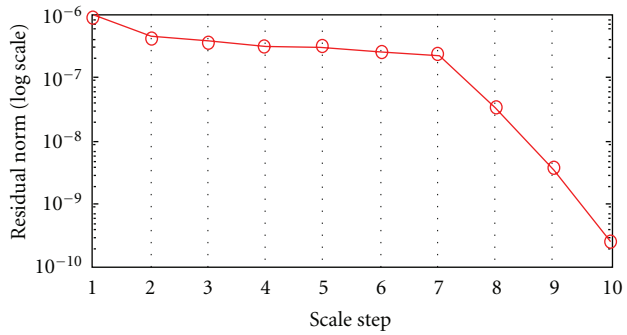

FIGURE 6: Algorithm I. Behavior of the relative residual $\|r_i\|_2/\|r_0\|_2$ versus 10 scale steps. $\Delta\tau_I = 0.04$.

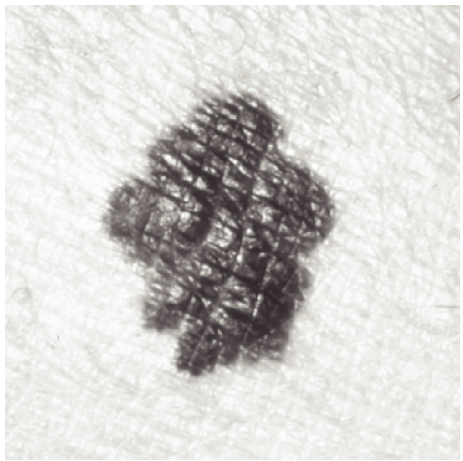

FIGURE 7: Test 2: ELM of a melanoma. Image size is 840×840.

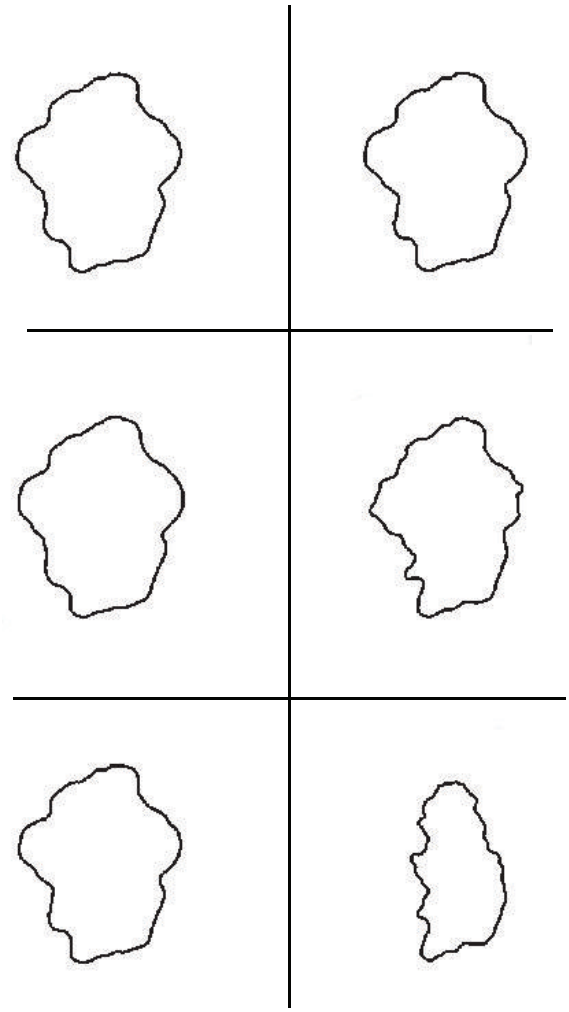

FIGURE 8: Test 2: Comparisons between segmentation results. On the left Algorithm I. On the right Algorithm SI. First row:, $\Delta\tau_I = 0.4$, $\Delta\tau_{\mathrm{SI}} = 0.16$. Second row: $\Delta\tau_I = 0.04$, $\Delta\tau_{\mathrm{SI}} = 0.16$. Third row: $\Delta\tau_I = 0.004$, $\Delta\tau_I = 0.0016$.

Parallel Performance. We show the performance of the multicore-based parallel algorithms and their scalability as the number of cores increases. We run the parallel algorithms using up to $p = 8$ cores of the parallel machine.

Following Figures report execution time, speedup, and efficiency of Algorithm SI, at scale step size $\Delta\tau = 0.16$ (i.e., $E_d = (10^{-1})$ (i.e., Figures 9, 10, and 11), then same results are shown at scale step size $\Delta\tau = 0.016$, corresponding to $E_d = O(10^{-2})$ (i.e., Figures 12, 13, and 14).

Finally, we report execution time, speedup, and efficiency of Algorithm I, at scale step $\Delta\tau = 0.4$ (i.e., Figures 15, 16, and 17) and $\Delta\tau = 0.04$ (i.e., Figures 18, 19, and 20), respectively. Note that parallel efficiency of both algorithms always is, at least, of 60% and, on average, of about 80%. In particular, parallel efficiency of Algorithm I is about 90%. Execution time of both algorithms reduces to about 2 seconds on eight cores in the first case (i.e., $E_d = (10^{-1})$, and to about 10 seconds in the second case ($E_d = O(10^{-2})$). This means that, in a multicore computing environment, both algorithms provide the requested solution within a response time that can be considered quite acceptable in medical imaging applications and, in particular, that Algorithm I is competitive with Algorithm SI.

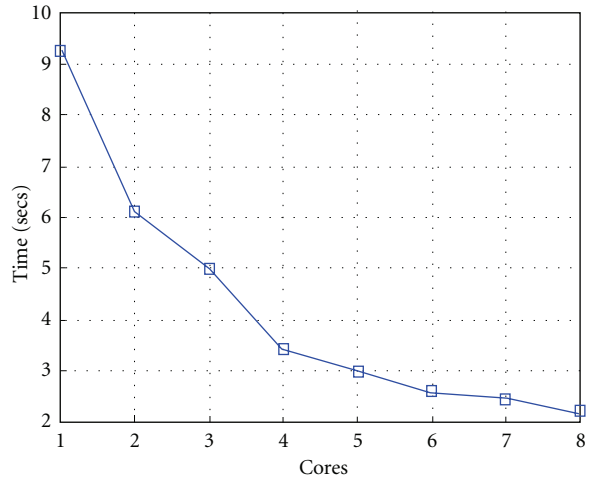

FIGURE 9: Test 1: Algorithm SI: Total execution time versus the number of cores. $\Delta\tau = 0.16$. 3 scale steps.

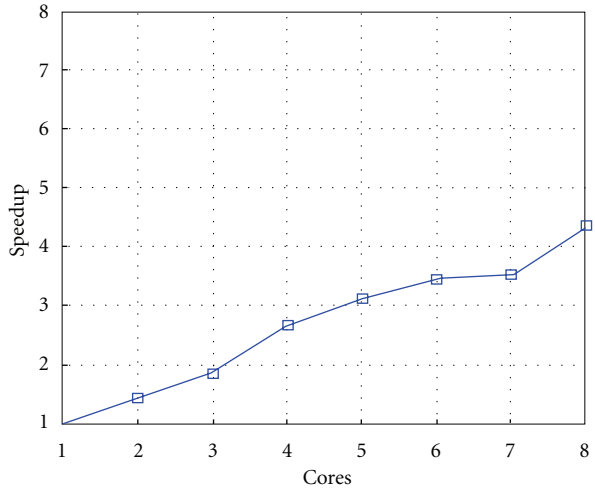

Figure 10: Test 1: Algorithm SI: Speedup versus the number of cores. $\Delta\tau = 0.16$. 3 scale steps.

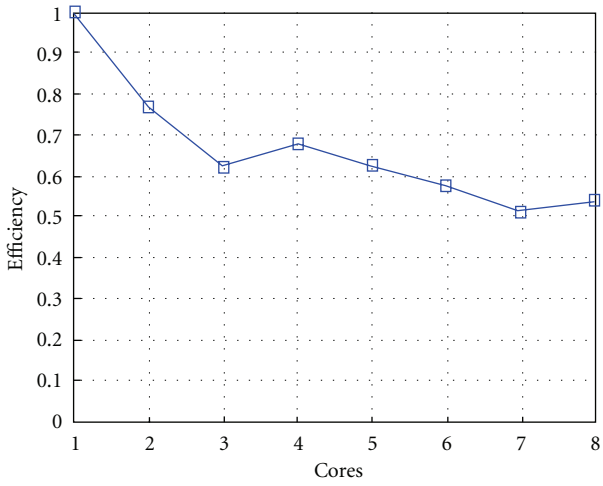

Figure 11: Test 1: Algorithm SI: Parallel efficiency versus the number of cores. $\Delta\tau = 0.16$. 3 scale steps.

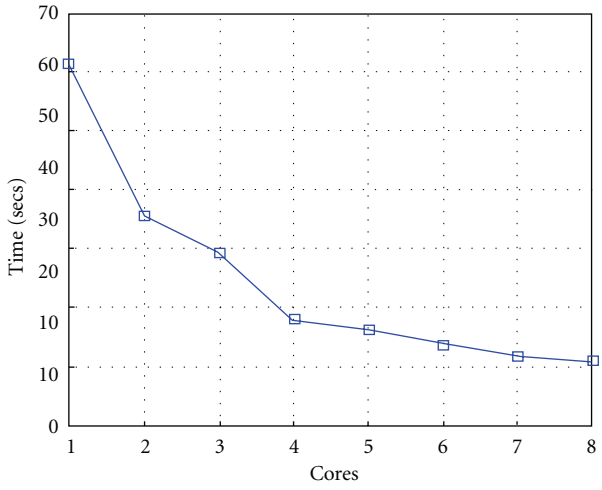

Figure 12: Test 1: Algorithm SI: Total execution time versus the number of cores. $\Delta\tau = 0.016$.

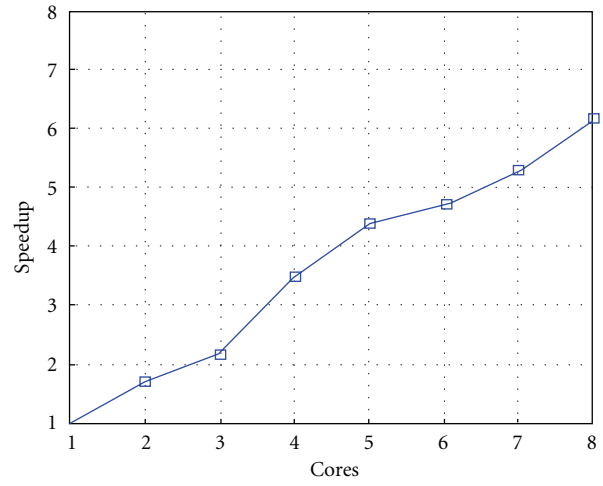

Figure 13: Test 1: Algorithm SI: Speedup versus the number of cores. $\Delta\tau = 0.016$.

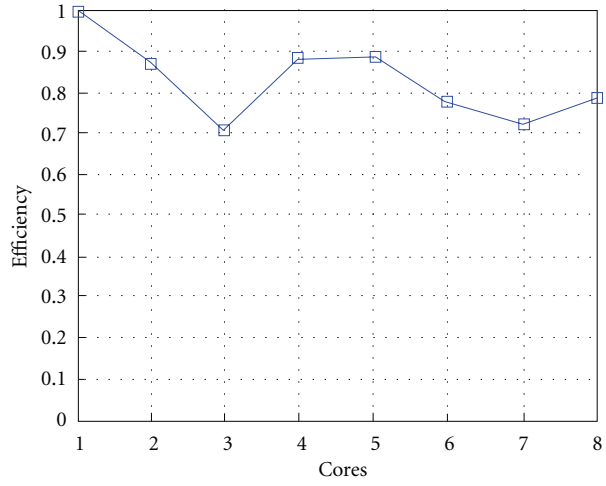

Figure 14: Test 1: Algorithm SI: Parallel efficiency versus the number of cores. $\Delta\tau = 0.016$.

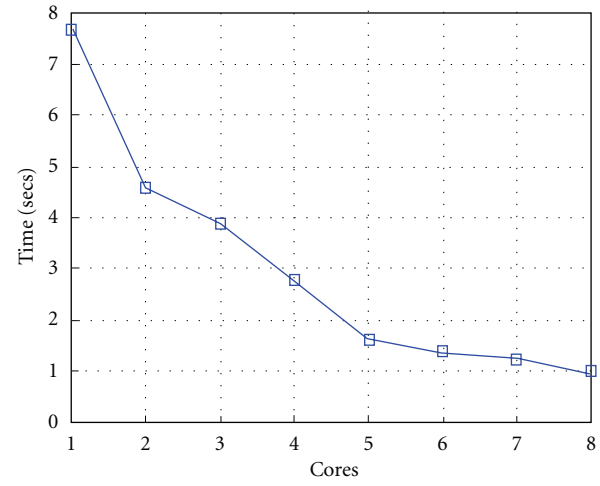

Figure 15: Test 1: Algorithm I: Total execution time versus the number of cores. $\Delta\tau_I = 0.4$.

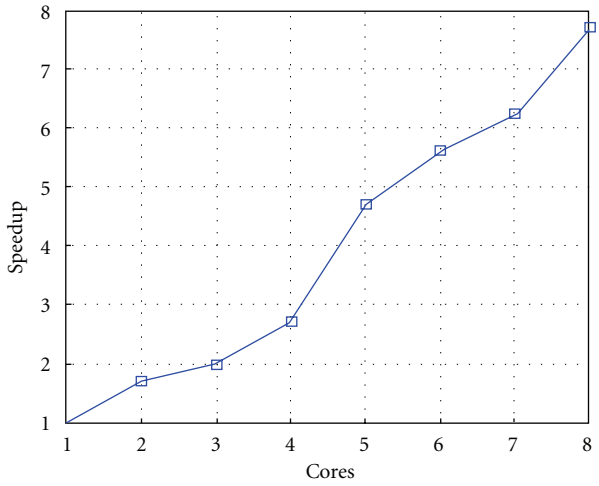

Figure 16: Test 1: Algorithm I: Speedup. $\Delta\tau_I = 0.4$.

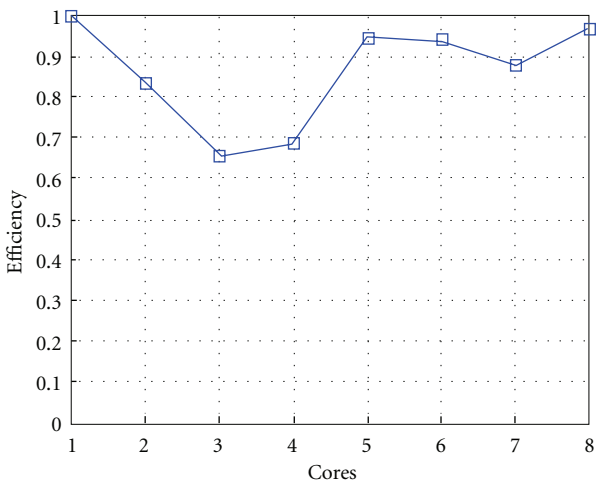

Figure 17: Test 1: Algorithm I: Efficiency. $\Delta\tau_I = 0.4$.

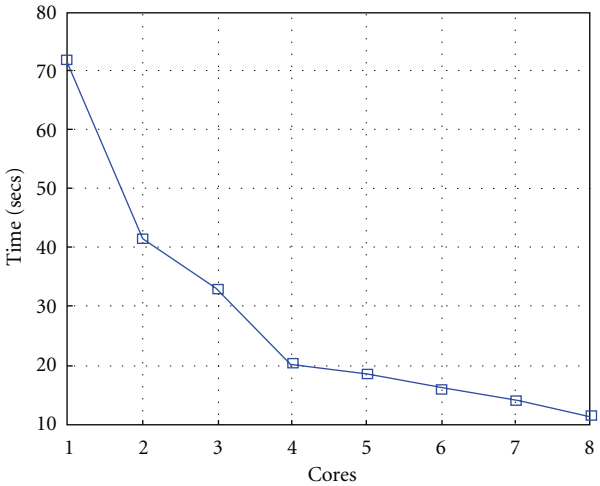

Figure 18: Test 1: Algorithm I: Total execution time versus the number of cores. $\Delta\tau_I = 0.04$.

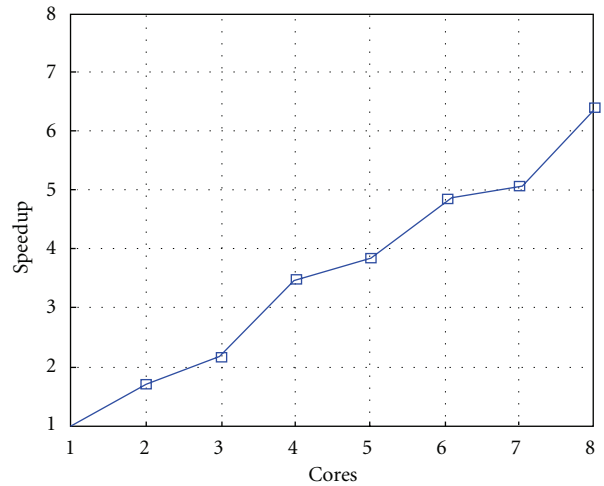

Figure 19: Test 1: Algorithm I: Speedup. $\Delta\tau_I = 0.04$.

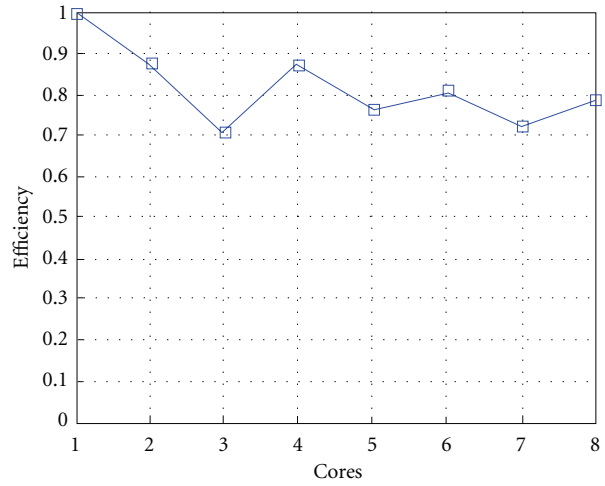

Figure 20: Test 1: Algorithm I: Efficiency. $\Delta\tau_I = 0.04$.

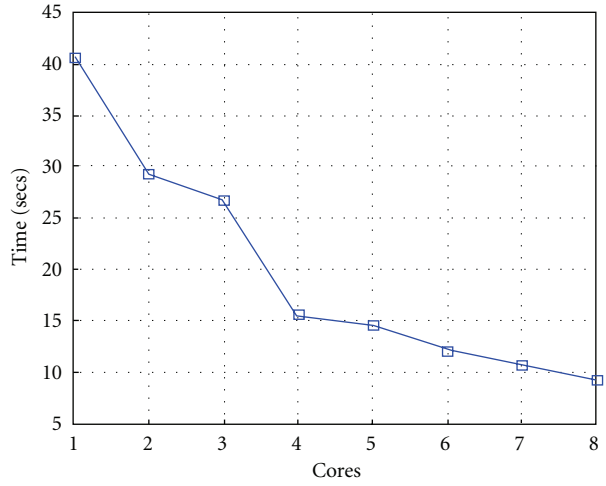

Figure 21: Test 2: Algorithm SI: Total execution time versus the number of cores. $\Delta\tau_I = 0.16$.

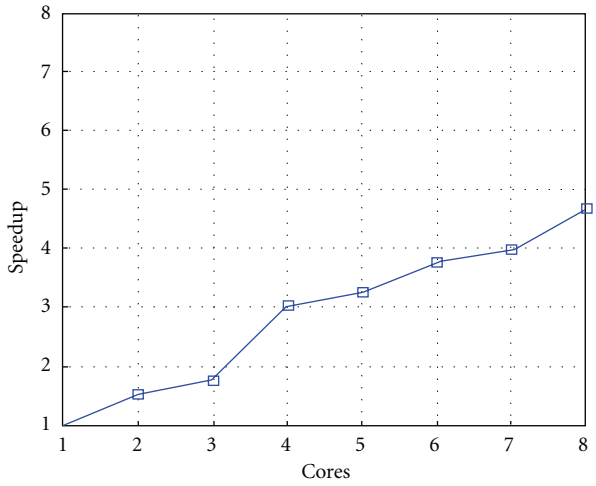

Figure 22: Test 2: Algorithm SI: Speedup. $\Delta\tau_I = 0.16$.

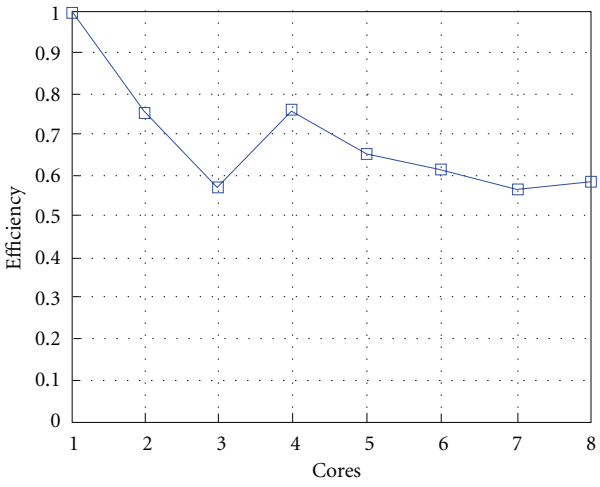

Figure 23: Test 2: Algorithm SI: Efficiency. $\Delta\tau_I = 0.16$.

Figures 21, 22, 23, 24, 25, and 26 show time, speedup, and parallel efficiency of two algorithms in case of Test 2. We only consider the case of $\Delta\tau_{\mathrm{SI}} = 0.16$ and $\Delta\tau_I = 0.4$.

Finally, we show results on scalability of parallel algorithms. Let $T_p(N)$ be the execution time of the parallel algorithm running on p cores for segmenting an image of size $N^2 \times N^2$. We measure the scalability of these algorithms by measuring $T_p(N)$ as N varies, once p is fixed, and by measuring $T_p(N)$ as N and p grow. We note that, in case of Algorithm SI, the scaling factor is

$$\frac{T_p(2N)}{T_p(N)} \simeq 4.2, \tag{41}$$

while, for Algorithm I, we get as scaling factor

$$\frac{T_p(2N)}{T_p(N)} \simeq 2.3. \tag{42}$$

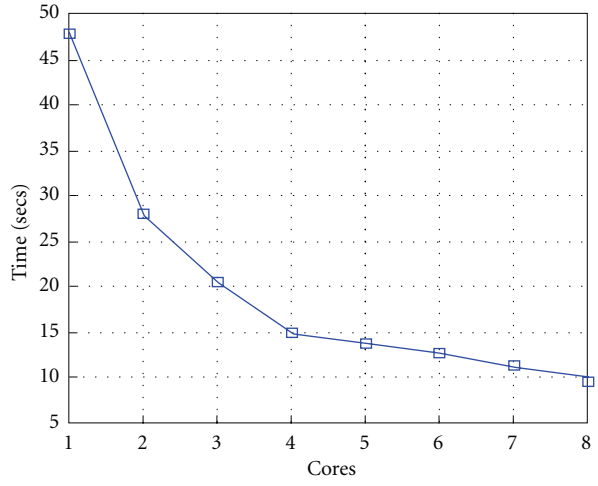

Figure 24: Test 2: Algorithm I: Total execution time versus the number of cores. $\Delta\tau_I = 0.4$.

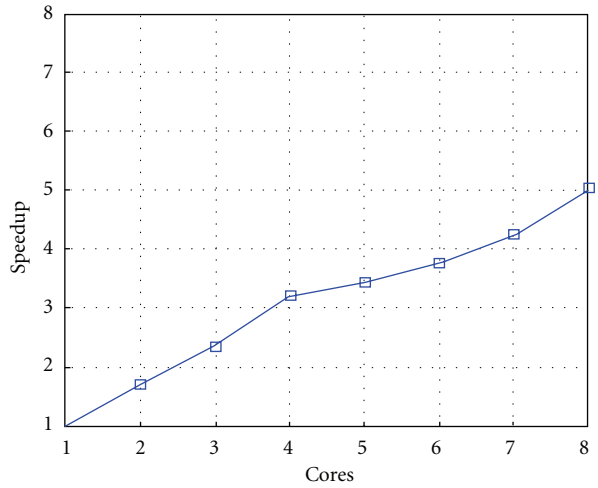

Figure 25: Test 2: Algorithm I. Speed up. $\Delta\tau_I = 0.4$.

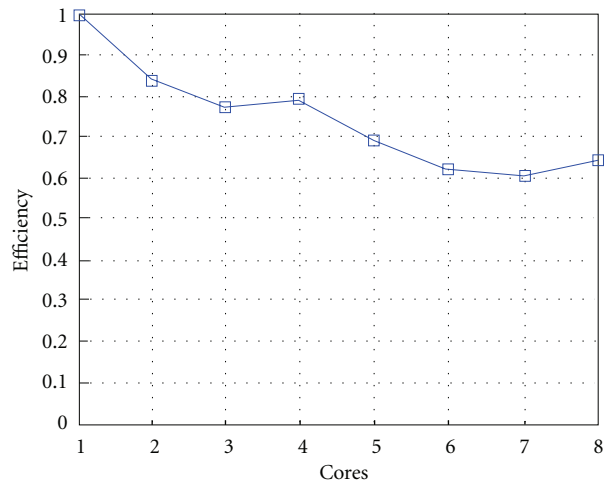

Figure 26: Test 2: Algorithm I. Efficiency. $\Delta\tau_I = 0.4$.

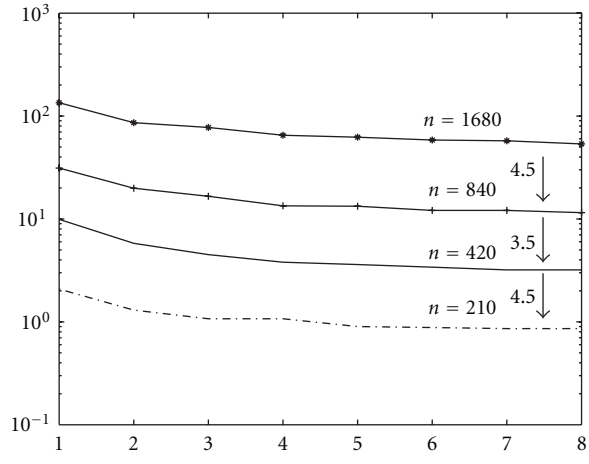

Figure 27: Scalability of parallel Algorithm SI, as N is fixed and p varies, and $N = 210, 420, 840, 1680$. Each line refers to the execution time of the algorithm at a fixed value of N.

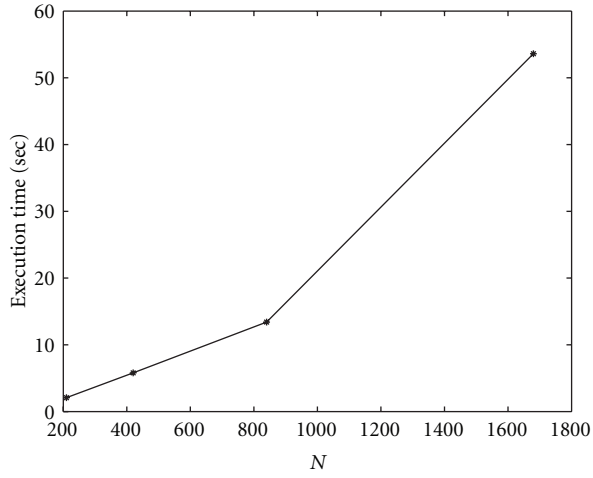

Figure 28: Scalability of parallel Algorithm SI, as N and p vary. Each point of the graph refers to the execution time of the parallel semi-implicit algorithm at ($p = k \cdot p_1$, $N = k \cdot N_1$), where $p_1 = 1$, $N_1 = 210$, and $k = 1, 2, 3, 4$.

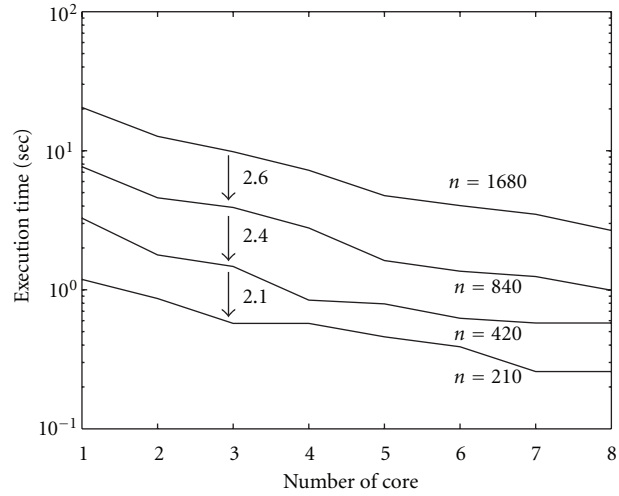

Figure 29: Scalability of parallel Algorithm I, as N is fixed and p varies, and $N = 210, 420, 840, 1680$. Each line refers to the execution time of the algorithm at a fixed value of N. $\Delta\tau_I = 0.4$.

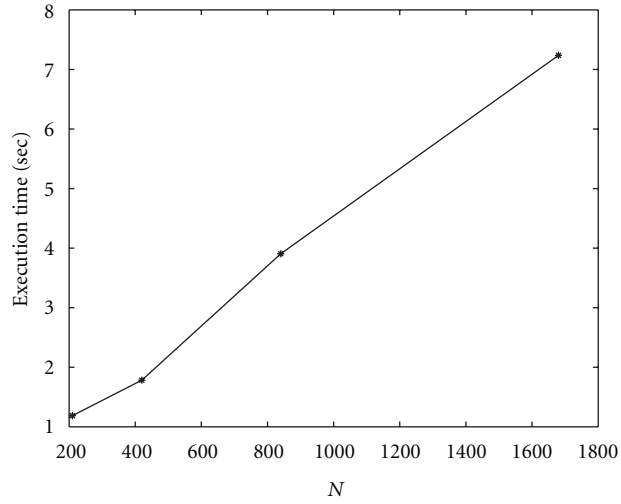

Figure 30: Scalability of parallel Algorithm I, as N and p vary. Each point of the graph refers to the execution time of the parallel implicit algorithm at ($p = k \cdot p_1$, $N = k \cdot N_1$), where $p_1 = 1$, $N_1 = 210$ and $k = 1, 2, 3, 4$. $\Delta\tau_I = 0.4$.

This means that Algorithm I scales better than Algorithm SI. In Figures 27 and 28 (for Algorithm SI) and Figure 29 and 30 (for Algorithm SI), we report $T_p(N)$ as N and p varies. In particular, each point of the graph refers to the execution time of the parallel algorithm at ($p = k \cdot p_1$, $N = k \cdot N_1$), where $p_1 = 1$, $N_1 = 210$, and $k = 1, 2, 3, 4$.

7. Conclusions

A straightforward comparison between the semi-implicit and the fully implicit discretization schemes of nonlinear PDE of parabolic/hyperbolic type states that fully implicit discretization usually leads to too expensive algorithms. In this paper, we provide a multicore implementation of two numerical algorithms arising from using these two discretization schemes: semi-implicit (Algorithm SI) and fully implicit (Algorithm I). Taking into account that we aim to solve such problems on parallel computer in a scalable way, in the first case, we use, as linear solver, Krylov iterative methods (GMRES) with algebraic multigrid preconditioners (AMG). Regarding the fully implicit scheme, we use the Jacobian-Free-Newton Krylov (JFNK) method as nonlinear solver.

We compare these two algorithms using different metrics measuring both the accuracy and the efficiency. We note that if we require that the discretization error E_d is $E_d = O(10^{-1})$, these two algorithms are quite numerically equivalent, both in terms of execution time and of the computed result; while, as discretization error decreases, Algorithm I appears to be more robust than Algorithm SI, in the sense that

Algorithm I reaches the steady state with high accuracy (the Hausdorff distance is of 95%), while the computed results of Algorithm SI are less accurate, even though the execution time of Algorithm I sometimes slightly increases. These results suggest that if it needs to get accurate and reliable results, Algorithm I should be preferable.

The parallel efficiency of both algorithms always is, at least, of 60% and, on average, of about 80%. In particular, parallel efficiency of Algorithm I is of about 90%. Execution time of both algorithms reduces to about 2 seconds on eight cores if $E_d = (10^{-1})$ and to about 10 seconds if $E_d = O(10^{-2})$. This means that, in a multicore computing environment, Algorithm I is competitive with Algorithm SI.

In conclusion, our results suggest that if it is required high accuracy of the computed solution in a suitable turnaround time, using a multicore computing environment fully implicit scheme provides an accurate and reliable solution within a response time of few seconds, quite acceptable in medical imaging applications, such as computer-aided-diagnosis.

References

[1] G. Aubert and P. Kornprobst, *Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations*, vol. 147 of *Applied Mathematical Sciences*, Springer, 2nd edition, 2006.

[2] http://www.cs.purdue.edu/research/cse/pses.

[3] J. W. Thomas, *Numerical Partial Differential Equations*, vol. 22 of *Text in Applied Mathematics*, Springer, 1995.

[4] Y. Saad and M. Schultz, "GMRES: a generelized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, vol. 7, pp. 856–869, 1986.

[5] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd edition, 1993.

[6] D. A. Knoll and D. E. Keyes, "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, vol. 193, no. 2, pp. 357–397, 2004.

[7] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy, "The impact of multicore on computational science software," *CTWatch Quarterly*, vol. 3, no. 1, 2007.

[8] http://www.mcs.anl.gov/petsc/petsc-as/index.html.

[9] M. Bertero and P. Boccacci, *Introduction to Inverse Problems in Imaging*, IOP Publishers, Bristol, UK, 1998.

[10] A. K. Louis, "Medical imaging: state of the art and future development," *Inverse Problems*, vol. 8, no. 5, pp. 709–738, 1992.

[11] W. Rundell and H. Eng, *Inverse Problems in Medical Imaging & Nondesctructive Testing*, Springer, 1997.

[12] A. N. Tikhonov and V. Y. Arsenin, *Solutions of Ill-Posed Problems*, John Wiley & Sons, New York, NY, USA, 1977.

[13] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, no. 1–4, pp. 259–268, 1992.

[14] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[15] J. J. Koenderink, "The structure of images," *Biological Cybernetics*, vol. 50, no. 5, pp. 363–370, 1984.

[16] T. Lindeberg, *Scale-Space Theory in Computer Vision*, Springer, 1994.

[17] O. Scherzer and J. Weickert, "Relations between regularization and diffusion filtering," *Journal of Mathematical Imaging and Vision*, vol. 12, no. 1, pp. 43–63, 2000.

[18] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

[19] J. A. Sethian, *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision and Materials Science*, Cambridge University Press, Cambridge, UK, 1999.

[20] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, 2003.

[21] L. C. Evans and J. Spruck, "Motion of level sets by mean curvature I," *Transaction of the American Mathematical Society*, vol. 330, no. 1, 1992.

[22] A. Sarti and G. Citti, "Subjective surface and Riemannian mean curvature flow graphs," *Acta Mathematica Universitatis Comenianae*, vol. 70, no. 1, pp. 85–104, 2001.

[23] R. Malladi and J. A. Sethian, "Level set methods for curvature flow, image enchancement, and shape recovery in medical images," in *Visualization and Mathematics*, H. C. Hege and K. Polthier, Eds., pp. 329–345, Springer, Heidelberg, Germany, 1997.

[24] N. J. Walkington, "Algorithms for computing motion by mean curvature," *SIAM Journal on Numerical Analysis*, vol. 33, no. 6, pp. 2215–2238, 1996.

[25] A. Handlovivcov, K. Mikula, and F. Sgallari, "Semi-implicit complementary volume scheme for solving level set like equations in image processing and curve evolution," *Numerishe Mathematik*, vol. 93, no. 4, pp. 675–695, 2003.

[26] J. W. Ruge and K. Stüuben, "Algebraic multigrid methods (AMG) applied to fluid flow problems," *Frontiers in Applied Mathematics*, 1986.

[27] V. E. Henson and U. M. Yang, "BoomerAMG: a parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.

[28] H. Ganster, A. Pinz, R. Röhrer, E. Wildling, M. Binder, and H. Kittler, "Automated melanoma recognition," *IEEE Transactions on Medical Imaging*, vol. 20, no. 3, pp. 233–239, 2001.

[29] H. Pehamberger, A. Steiner, and K. Wolff, "In vivo epiluminescence microscopy of pigmented skin lesions. I. Pattern analysis of pigmented skin lesions," *Journal of the American Academy of Dermatology*, vol. 17, no. 4, pp. 571–583, 1987.

[30] F. Nachbar, W. Stolz, T. Merkle et al., "The ABCD rule of dermatoscopy: high prospective value in the diagnosis of doubtful melanocytic skin lesions," *Journal of the American Academy of Dermatology*, vol. 30, no. 4, pp. 551–559, 1994.

[31] A. Csazar, *General Topology*, Adam HIlger, Bristol, UK, 1978.

[32] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 850–863, 1993.