Rendiconti di Matematica, Serie VII
Volume 29, Roma (2009), 29–50

A level-set algorithm for front propagation in the presence of obstacles

M. FALCONE – C. TRUINI

Dedicated to Umberto Mosco

ABSTRACT: *We present an algorithm for tracking the monotone propagation of an interface in a domain where obstacles are present, i.e. where some regions inside the domain cannot be crossed by the interface. The evolution is computed via the level-set method solving a stationary Hamilton-Jacobi equation. A semi-Lagrangian method based on a bilinear spatial recostruction is used to compute the viscosity solution and to obtain the interface. Transparent boundary conditions are implemented in a very simple way which writes them as an homogeneous Dirichlet boundary condition. As a result, the algorithm produces a global solution and it is easy to implement even when the obstacles have rather complicated boundaries. Numerical tests in 2D and 3D illustrate the main features of this method.*

1 – Introduction

The level-set method is a clever and rather simple way to describe an interface separating two or more regions with different physical phases. As it is

KEY WORDS AND PHRASES: *Front propagation – Level-set method, Hamilton-Jacobi equations – Semi-Lagrangian schemes – Obstacles and state constraints – Transparent boundary conditions*

A.M.S. CLASSIFICATION: 65N12, 65M10, 49L25.

This work has been partially supported by MURST, Progetto Nazionale "Calcolo Scientifico: Modelli e Metodi Numerici Innovativi". We also wish to thank the CASPUR Consortium for its technical support.

well known, the method describes the evolution of the front by a continuous representation function $u(x,t)$, which is positive in the domain Ω_t corresponding to one of the phases, negative outside that domain and changes sign accross the interface. A comprehensive introduction to the level-set method as well as several applications and references can be found in [24].

Starting from the first paper by Osher and Sethian [21], the level-set method has been extensively used to track numerically the evolution of interfaces in a variety of different situations. Whenever, the interface evolution is simply driven by a given vectorfield plus a normal velocity the level set method leads to a nonlinear first order PDE. More complicated types of evolution consider the normal velocity as a function of the curvature and/or of other geometric parameters of the interface and this leads to second order nonlinear PDE problems.

Let us consider here the typical model problem for an interface which evolves in the normal direction driven by a (given) scalar velocity $c(x)$. This problem leads to the following first order Hamilton–Jacobi equation,

$$\begin{cases} u_t + c(x)|\nabla u| = 0, & \text{in } \mathbb{R}^n \times \mathbb{R} \\ u(x,0) = u_0(x) & \text{in } \mathbb{R}^n \end{cases} \tag{1.1}$$

where the initial condition u_0 must be the representation function of the initial position of the front $\Gamma_0 = \partial\Omega_0$, which is the only initial datum. The above problem can be drastically simplified when the evolution is monotone (increasing or decreasing), i.e. when either $\Omega_t \subset \Omega_{t+s}$ or the reverse inclusion are satisfied. For monotone types of evolution, it has been proved in [14] that equation (1.1) can be replaced by the following stationary equation,

$$\begin{cases} c(x)|\nabla T| = 1, & \text{in } \mathbb{R}^n \setminus \Omega_0 \\ T(x) = 0 & \text{in } \Omega_0 \end{cases} \tag{1.2}$$

where T represents the time of transfer of a point $x \in \mathbb{R}^n \setminus \Omega_0$ to Ω_0 by an appropriate dynamics. In fact, the link between the two problems is simple: the unique viscosity solution of (1.1) is $v(x,t) = T(x) - t$, where T is the viscosity solution of (1.2). In terms of level-sets the interface corresponds to the 0-level set of u and to the t-level set of T. It is worth to note that the second problem is easier to solve since it does not require the additional computation of u_0, this computation would require in fact the solution of another Hamilton-Jacobi equation of type (1.2) to compute the (signed) distance function to Ω_0. Moreover, the knowledge of T gives a global description of the interface at any time t. The stationary approach introduced in [14] and extended to anisotropic evolutions in [30] relies on the link between the propagation of fronts and the minimum time problem of control theory. We should also mention that the link between stationary and evolutive problems has been analyzed also in other papers, e.g. we refer to [20] for a general level set formulation for the solution of the Dirichlet

problem for Hamilton–Jacobi equations. As far as numerics is concerned, the most popular methods are certainly based on finite differences and often include recent developments such as the fast marching method to save CPU time (see e.g. [25] and [26]). We will introduce here a local version of the fully discrete semi-Lagrangian scheme (SL scheme in the sequel) for the stationary problem (1.2) studied in [9] and [2].

A first goal of this paper is to extend the stationary approach to the numerical approximation of the evolution of interfaces in the presence of obstacles. The difficulty is to include obstacles in the model and to adapt the SL scheme in order to obtain a simple implementation of boundary conditions as well as accurate results. An obstacle is, in our approach, a subdomain which cannot be crossed by the interface: the front can only "touch" the obstacle and proceed going around it. Naturally this process modifies the front propagation after the first time of contact between the front and the obstacle. Which kind of boundary conditions is more appropriate to describe the front-obstacle interaction? As we will see in the following sections we need to implement transparent boundary condition, since standard Dirichlet or Neumann type boundary conditions modify the front also inside the domain of computation. The interesting point is that the implementation of transparent boundary conditions can be obtained rather easily for the SL scheme adopted in this paper.

The second goal is to present a parallel algorithm based on the SL method. To this end we first write a local version of the scheme on a structured grid where the solution at one node just depends on the informations at the neighbouring nodes (this is usually not necessary for SL schemes). Then we construct the parallel algorithm by a domain decomposition strategy, i.e. we split the domain of computation Ω into D physical subdomains. Naturally, the subdomains have internal boundaries (and cross points) so we need to modify the serial algorithm in order to make the informations flow between the subdomains during the computation. Inside the subdomains we apply the local version of the serial algorithm. This approach allowed us to compute with a reasonable CPU time the solutions of 3D model problems which have a huge number of nodes.

In order to give some background to the interested reader, we recall that the problem of an efficient implementation of boundary conditions for finite difference schemes has been addressed by Aslam, Bdzil and Scott Stewart in [1]. They have modified the original scheme proposed in [21] including three types of boundary conditions: reflecting, non-reflecting (or continuation) and angle (sonic or subsonic) boundary conditions. The non-reflecting (transparent) boundary condition are applied there in a way which is different from ours since they use quadratic extrapolation. It is also worth to mention that SL schemes are extentions of the Courant–Isaacson–Rees method [7] for conservation laws. They allow large time–steps and can guarantee high accuracy, properties which made them very popular in the Numerical Weather Prediction community (see the survey paper [28] for more informations). In the framework of (first order)

front propagation problems they have been studied in [9], [13]. SL schemes for general problems with convex hamiltonians have been developed and analysed in [11]. It is interesting to note that, in that case, the schemes can be interpreted as a discrete version of the Hopf representation formula for the exact solution. High–order accurate versions of those schemes have been studied in [11], [17] (see also [10] and the references therein). To complete the scenario, we should also mention the applications of SL methods to curvature dependent front propagation problems given in [31] and [12]. Although this paper deals with a scheme written for a simple cartesian grid, it is worth to mention that SL schemes may also be used on unstructed meshes as it has been done in [23] (see also [27] for a finite difference scheme on unstructured meshes for similar problems).

The paper is organized as follows.
In Section 2 we give some background, the basic assumptions and results on the SL algorithm for the front propagation problem without obstacles. In Section 3 we introduce the problem with obstacles and we discuss the implementation of transparent boundary conditions. Section 4 is devoted to the fully discrete scheme in $\mathbb{R}^2$. Section 5 deals with the parallel version of the scheme on a MIMD (Multiple Instruction Multiple Data) architecture. Finally, in Section 6 we present several experiments in $\mathbb{R}^2$ and $\mathbb{R}^3$.

2 – Preliminary results and assumptions

Following [9], let us recall the basic results on a SL scheme for a monotone evolution without obstacles, i.e. for the Dirichlet problem (1.2). To this end it is useful to remind that (1.2) can be written as

$$\begin{cases} H(x, DT(x)) = 0, & \text{in } \mathbb{R}^n \setminus \Omega_0 \\ T(x) = 0 & \text{in } \Omega_0 \end{cases} \tag{2.1}$$

where

$$H(x, Du) \equiv \max_{a \in B(0,1)} \{c(x) DT(x) \cdot a\} - 1 \tag{2.2}$$

In the sequel we will always assume that the normal velocity $c : \mathbb{R}^n \to \mathbb{R}$ is strictly positive and that there exists a constant L such that

$$|c(x) - c(y)| \le L|x - y|, \text{ for any } x, y \in \mathbb{R}^n \setminus \Omega_0; . \tag{2.3}$$

$$|c(y)| \le L(1 + |y|), \text{ for any } y \in \mathbb{R}^n \setminus \Omega_0; . \tag{2.4}$$

We also define, for $x \in \mathbb{R}^n$ and $a \in B(0, 1)$,

$$b(x, a) \equiv -c(x)a \ . \tag{2.5}$$

Naturally, we compute the solution in an open n-rectangular domain $Q \subset \mathbb{R}^n$ containing Ω.

Let us construct a uniform cartesian grid in Q which (for simplicity) has the same step size in every direction, i.e. $\Delta_1 x = \cdots = \Delta_n x = k$. Let us denote by X the set of its nodes x_i, $i \in I \equiv \{1, ..., N\}$ and by $\mathcal{S}$ the set of all cells S_j, $j \in \{1, ..., L\}$, i.e. $Q = \bigcup_j S_j$. By the change of variable $w(x) \equiv 1 - \exp(-T(x))$ we can trasform (1.2) into a fixed point problem for w and get the fully discrete scheme for w:

$$(2.6) \qquad \begin{cases} w(x_i) = \min\limits_{a \in B(0,1)} [\beta w(x_i + hb(x_i, a))] + 1 - \beta & \text{for } i \in I_{in} \\ w(x_i) = 0 & \text{for } i \in I_0 \ . \end{cases}$$

where $\beta \equiv \mathrm{e}^{-h}$ and

$$(2.7) \qquad \begin{aligned} &I_0 \equiv \{\ i \in I \ :\ x_i \in \Omega\ \} \\ &I_{in} \equiv \{\ i \in I \setminus I_0 \ :\ \exists\ a \in B(0,1) \text{ such that } x_i + hb(x_i, a)\ \in Q\ \} \end{aligned}$$

Note that for $0 \le T < +\infty$, $w \in [0, 1[$. The value $w(x_i + hb(x_i, a))$ which appear on the right-hand side must be computed by a local reconstruction which uses the values sitting on the nodes. A typical choice is to use linear interpolations in the cells, this produces a monotone first order accurate fully discrete scheme. In the scheme we must introduce a "generalized" bondary condition to compute $w(x_i + hb(x_i, a))$ whenever $x_i + hb(x_i, a) \notin Q$. For the moment, let us just set $w(z) \equiv 1$, for any $z \notin Q$. We will explain and comment that condition in the next section.

The algorithm computes the solution via the fixed point iteration (2.6) which we can be written as

$$W^{p+1} = F(W^p), \quad p = 0, 1, 2, \ldots$$

where $F : \mathbb{R}^N \to \mathbb{R}^N$ is defined componentwise by the right-hand side of (2.6).

The following result summarizes the properties of the scheme (2.6).

THEOREM 2.1. *Assume* (2.3), (2.4) *then:*

a) the approximation scheme (2.6) *admits a unique (piecewise linear) solution* $w : Q \to [0, 1]$.

b) choosing

$$(2.8) \qquad W^0 = \begin{cases} 0 & \text{for } i \in I_0 \\ 1 & \text{for } i \in I \setminus I_0 \end{cases}$$

the fixed point iteration produces a monotone decreasing sequence converging to the fixed point.

The following result gives an $a-priori$ estimate for the L^∞ error of the approximation of v and of the Hausdorff distance between the approximate and exact interface (cfr. [9] for the proof):

COROLLARY 2.2. *Let* (2.3), (2.4) *hold true,* Ω_0 *have a piecewise regular boundary and let* $k \leq C\,h$ *, for some positive constant. The approximate solution converges uniformly in* Q *to the viscosity solution of* (1.2)*. Moreover, if*

$$C \equiv \min_{x \in Q \setminus \Omega_0} c(x) \tag{2.9}$$

there exist two positive constant C_1 *and* C_2 *such that, for* h *and* k *sufficiently small,*

$$\| v - w \|_\infty \leq C_1 h + C_2 k \tag{2.10}$$

and

$$d_H(\Omega_t, \widehat{\Omega}_t) \leq C_1 h + C_2 k. \tag{2.11}$$

where d_H *denotes the standard Hausdorff distance between two sets and* $\widehat{\Omega}_t$ *is the level set of the approximate solution.*

3 – Obstacles and transparent boundary conditions

Let us consider an evolution driven by a normal velocity $c(x)$ in a domain where obstacles are present. We will assume that the obstacles are closed disjoint subsets O_i, $i = 1, \ldots, M$, inside our domain of computation Q and we will denote by $\mathcal{O}$ their union, i.e. $\mathcal{O} = \bigcup_{i=1}^{M} O_i$. An *obstacle* is a subdomain which cannot be crossed by the front, i.e. when the front touches an obstacle it is forced to get around it. It is important to note that c will not vanish on the boundary $\partial\mathcal{O}$ so that the front continues its evolution slipping around the obstacle boundary. Which kind of boundary condition should we impose there? A classical boundary condition which is usually adopted in fluid dynamics to describe this type of situations is the homogeneous Neumann boundary condition

$$\frac{\partial u}{\partial \eta} = 0 \text{ on } \partial\mathcal{O} \tag{3.1}$$

where $\eta(x)$ is the (exteriour) normal direction to the boundary of $\mathcal{O}$. That condition will not be appropriate for our problem since it will force the front to be orthogonal to the obstacles boundaries. In fact, the Neumann condition

$$\frac{\partial u}{\partial \eta} = \nabla u(x,t) \cdot \eta(x) = 0 \tag{3.2}$$

means that the front is orthogonal to $\partial\mathcal{O}$ since the front Γ_t is the 0-level set of u, i.e.

$$\Gamma_t \equiv \{x \in Q : u(x,t) = 0\}. \tag{3.3}$$

Any other Neumann boundary condition will force the front to have a predetermined angle with the normal to the obstacle and this does not seem adequate for our problem since the front can hit the obstacle with any angle. In this respect, the situation is different from the detonation shock dynamics problem described in [1] where an angle boundary condition is applied at all the "internal boundary nodes" which have a subsonic interaction. The same remarks apply to the boundary conditions on ∂Q. A correct solution to this problem can be obtained introducing transparent boundary conditions on the obstacle boundaries *and* on the boundary of the domain of computation. The use of trasparent boundary condition is typical in other evolutive problems, e.g. the wave propagation problem, and usually requires the solution of a differential equation at the boundary (cfr. [8]). In [1] a non-reflecting (transparent) boundary condition is applied by using quadratic extrapolation. Both methods can be difficult to apply when the geometry of the boundaries is rather complex. Moreover, they are more expensive if compared to our method. Now we will introduce easy-to-use boundary conditions for our front propagation problem motivating them by a control theoretical interpretation.

As we mentioned in the introduction, the relation between front propagation in a homogeneous region and the minimum time problem has been analyzed in [14] and [9]. It has been shown that the characteristics of the front problem are the optimal trajectories of the corresponding minimum time problem.

Let us define

$$D = Q \setminus (\mathcal{O} \cup \Omega_0), \tag{3.4}$$

When there are no obstacles the trajectories can go everywhere in Q, but when obstacles are present in the domain of computation the trajectories cannot cross $\mathcal{O}$ and must remain in D. The problem with obstacles corresponds then to the minimum time problem with state constraints where the state of the problem must stay in D, for any $t \geq 0$. This suggests which condition has to be imposed on their boundaries. In fact, the set of admissible controls (i.e. of admissible directions which are allowed for the front propagation) at a point $x \in \partial\mathcal{O}$ is reduced to the directions pointing inward $\overline{D}$. The usual definition of viscosity solution has to be modified to characterize the solution to this problem and get uniqueness. We have to look for a *state constrained viscosity solution* which means that we have to solve

$$H(x, DT) \leq 0, \quad x \in D \tag{3.5}$$

$$H(x, DT) \geq 0 \quad x \in \overline{D} \tag{3.6}$$

where H has been definined in (2.2) and both conditions must be understood in the viscosity sense. Naturally, the above conditions must be coupled with the homogeneous Dirichlet boundary condition on Ω_0. The definitions (3.5)-(3.6) mean that a state constrained viscosity solution is a viscosity subsolution in D and a viscosity supersolution in $\overline{D}$. They were first introduced by Soner [29] (see also [6]). Note that the above definition keeps all the directions in the unit ball and requires an inequality to be satisfied at the obstacle boundary. More recently, Ishii and Koike [18] proposed a new definition which is closer to our approach since it reduces the space of admissible controls taking into account the constraints and requiring an equality at the obstacle boundary. That idea was introduced, for numerical purposes, in [5]. It is interesting to note that also the new definition allows to get a uniqueness result for the solution.

Now let us construct a discrete scheme for the minimum time problem with state constraits which will allow us to solve the interface problem with obstacles whenever the evolution is monotone.

Let $h = \Delta t$ be the time step (for the discrete characteristics) and k be the (uniform) mesh size. Let us reduce the discrete set of admissible control at every point x to

$$A_h(x) \equiv \{a \in B(0,1) : x + hc(x)a \in \overline{D}\} \tag{3.7}$$

The corresponding fully discrete scheme is

$$w(x_i) = \min_{a \in A_h(x_i)} [\beta w(x_i + hc(x_i)a)] + 1 - \beta \,, \qquad \text{for } x_i \in D \tag{3.8}$$

Following the arguments in [5], one can prove that there exists a unique piecewise linear solution w (3.8) and that w converges uniformly to the "constrained" viscosity solution v of the constrained problem in $\overline{D}$ for h and k tending to 0. Note that the direct application of the above formulation would require the preliminary computation of the admissible control set $A_h(x_i)$ at every node of the grid. This can be rather expensive particularly when the grid has a large number of nodes (tipically in 3D problems). However, we can get around this difficulty and obtain the same result just leaving $a \in B(0,1)$ provided we modify the algorithm so that the minimum *cannot* be attained at controls which do *not* satisfy the constraints, i.e. for $a \in B(0,1) \setminus A_h(x_i)$. This can be easily obtained just setting in (3.8)

$$w(x_i + c(x_i)a) = w_{max}, \text{ when } x_i + c(x_i)a \in \mathcal{O} \tag{3.9}$$

where w_{max} is the positive constant

$$w_{max} = \max_{x \in D} w(x) \,. \tag{3.10}$$

Since, $0 \leq w \leq 1$ it is sufficient to set $w_{max} = 1$ and

$$w(x) \equiv 1 \quad \text{for any } x \in \mathcal{O}. \tag{3.11}$$

That simple choice produces an important reduction of the storage requirements. Moreover, the algorithm will *not* modify the internal values (at the nodes $x_i \in D$) imposing a boundary condition. In fact, at the nodes close to the boundaries where $x_i + hc(x_i)a \notin D$ the algorithm performs an automatic up-wind correction taking into account only the informations inside D since the values corresponding to points inside $\mathcal{O}$ are bigger than those values (and the algorithm drops them out searching for the minimum). As we said, the computation is made only at the internal nodes. The scheme is consistent with the physics since setting the value $w = 1$ at all the nodes $x_i \in \mathcal{O}$ corresponds to set $c(x) \equiv 0$ in $\mathcal{O}$, so that the front cannot cross the obstacle. However, the normal velocity inside D can be strictly positive up to the boundary of the obstacle.

4 – The SL fully discrete scheme on a structured grid

Let us describe the SL scheme for (2.6) on a structured grid. The extension to unstructured grid would require more computational effort in order to locate the roots of the characteristics on the simplices. An implementation on unstructured grid can be found in [23].

According to what we have seen in the preceeding section, the scheme will really compute only on the nodes in I_{in}, i.e.

$$w(x_i) = \min_{a \in B(0,1)} [\beta w(x_i + hb(x_i, a))] + 1 - \beta \,, \qquad i \in I_{in}. \tag{4.1}$$

Moreover, on the nodes x_i such that $i \in I_0$ (i.e. for $x_i \in \Omega_0$) we always apply the Dirichlet boundary condition $w(x_i) = 0$ and whenever the point $x_i + hb(x_i, a) \notin (Q \setminus \mathcal{O}) \cup \Omega_0$ we set $w(x_i + hb(x_i, a)) = 1$.

The scheme will use a variable time step which will depend on the velocity at the point x_i and on the space step k,

$$h_i = \frac{k}{c(x_i)}. \tag{4.2}$$

This will guarantee that the root of the characteristic starting at x_i will always belong to one of the four neighbouring cell having a node at x_i. In fact, substituting (2.5) and (4.2) in the equation (4.1) we have:

$$w(x_i) = \min_{a \in B(0,1)} [\beta w(x_i - ka)] + 1 - \beta \qquad i \in I_{in} \tag{4.3}$$

so that $x_i - ka$, for $a \in B(0, 1)$, spans the circle of radius k centered at x_i

From now on we will always use two indices for the nodes since this notation makes easier to write the 2D scheme. Let $x_{ij} \equiv (x_i, y_j)$ be a node of the grid, the corresponding circle will be

$$x_{i,j} - ka = (x_i - kcos(\theta), y_j - k\sin(\theta)) \qquad \theta \in (0, 2\pi] \tag{4.4}$$

To compute the minimum we will use a discrete version of the unit ball $B(0,1)$ taking into account only 36 directions,

$$\theta_l = \theta_0 + \frac{2\pi}{36} l \qquad l = 1, ..., 36 \qquad \theta_0 = 0. \tag{4.5}$$

The values $w(x_{i,j} - ka)$ are computed by a bilinear interpolation on the values at the nodes. Let us show how one can easily compute the coefficients $\lambda_{i,j}(\theta)$ of this representation

$$w(x_i - k\cos\theta, y_j - k\sin\theta) = \sum_{i,j} \lambda_{i,j}(\theta) w(x_{i,j}) \tag{4.6}$$

Let G be a grid cell with vertices at $x_{i,j}$, $x_{i+1,j}$, $x_{i,j+1}$, $x_{i+1,j+1}$ and let $f_{i,j}$, $f_{i+1,j}$, $f_{i,j+1}$, $f_{i+1,j+1}$ be the corresponding values of a function f at the vertices. The bilinear interpolation computes the value of f at every point $(x, y) \in G$ by the simple definition

$$f(x, y) = axy + bx + cy + d \tag{4.7}$$

where the coefficients a, b, c and d can be determined solving the linear system 4×4 which corresponds to the four height conditions at the four vertices of the cell. Those values can also be written as linear combinations of the values of f at the vertices of the cell, i.e.

$$f(x, y) = \lambda_{i,j} f_{i,j} + \lambda_{i+1,j} f_{i+1,j} + \lambda_{i,j+1} f_{i,j+1} + \lambda_{i+1,j+1} f_{i+1,j+1} \tag{4.8}$$

where the $\lambda_{i,j}$ coefficients are given by

$$\lambda_{i,j} = (x_{i+1} - x)(y_{i+1} - y)/C \tag{4.9}$$

$$\lambda_{i+1,j} = (x - x_i)(y_{i+1} - y)/C \tag{4.10}$$

$$\lambda_{i,j+1} = (x_{i+1} - x)(y - y_i)/C \tag{4.11}$$

$$\lambda_{i+1,j+1} = (x - x_i)(y - y_i)/C \tag{4.12}$$

and $C = (x_{i+1} - x_i)(y_{i+1} - y_i)$ is the area of the cell, i.e. $C = k^2$ for our uniform grid. In practice, it is useful to write the λ_{ij} coefficients as functions of the θ angle:

$$
\begin{aligned}
\lambda_{i,j} &= (k - k\cos\theta)(k - k\sin\theta)/k^2 = (1 - \cos\theta)(1 - \sin\theta) \\
\lambda_{i+1,j} &= (k\cos\theta)(k - k\sin\theta)/k^2 = \cos\theta(1 - \sin\theta) \\
\lambda_{i,j+1} &= (k - k\cos\theta)(k\sin\theta)/k^2 = \sin\theta(1 - \cos\theta) \\
\lambda_{i+1,j+1} &= (k\cos\theta)(k\sin\theta)/k^2 = \sin\theta\cos\theta.
\end{aligned}
\tag{4.13}
$$

Substituting in (4.3) we get a new form for the fully discrete scheme:

$$
w(x_{i,j}) = \beta \min_{\theta\in(0,2\pi]} \left[\sum_{r,s}^{-1,0,1} \lambda_{i+r,j+s}(\theta) w(x_{i+r,j+s}) \right] + 1 - \beta \tag{4.14}
$$

The iterative scheme corresponding to (4.14) can be finally written componentwise as

$$
U_{i,j}^{p+1} = \beta \min_{\theta\in(0,2\pi]} \left[\sum_{r,s}^{-1,0,1} \lambda_{i+r,j+s}(\theta) U_{i+r,j+s}^{p} \right] + 1 - \beta. \tag{4.15}
$$

This is the "local" version of the iterative scheme (2.6). In fact, the solution at the $(p+1)$-th iteration at the node x_{ij}, is expressed just in terms of the eight values of the p-th iteration at the nodes whose distance to the central node is lower than $\sqrt{2}k$. This property, which is not necessary to make the SL method converge, is particularly useful for the construction of a parallel version of the algorithm since it minimizes the communication loads between the processors as we will see in the next section.

5 – Parallel Algorithm

We will start from the local version (4.15) to construct a parallel version of the algorithm based on a domain decomposition. An introduction to numerical methods for parallel machines can be found in [19]. Actually, the algorithm has been implemented on SIMD (Single Instruction Single Data) and MIMD (Multiple Instruction Multiple Data) machines with SM (Shared Memory) or DM (Distributed Memory) at the CASPUR Consortium (cfr. [16] for a 2D implementation without obstacles on a SIMD machine).

A typical difficulty which is encountered in the simulation of interface problems is to obtain a precise description of the interface even in the presence of singularities, cusps and topological changes. This often requires to use meshes with a large number of nodes and produces long computations and/or large memory storage requirements (this is particularly true when dealing with 3D problems). A parallel version of the algorithm can be very useful to obtain accurate solutions with low CPU times.

We will describe here the implementation on a MIMD-DM architecture since this is the most flexible, cheap and, perhaps, most popular architecture. In fact, a MIMD-DM machine can be "constructed" just connecting several UNIX workstation on a network by means of a communication software (typically PVM, MPI or OpenMP). The memory is distributed due to the fact that every processor has its own memory, so a crucial point for the algorithm is to minimize the number of (unavoidable) communications among the processors. We adopt a domain decomposition strategy which splits Q into D subdomains Q_d, $d = 1, \ldots, D$, assigning the computation on the nodes inside every subdomain to a single processor P_d (cfr. [22] for a comprehensive introdution to this topic). The decomposition creates internal boundaries and can also have crossing points (i.e. points which belong to more than $n + 1$ subdomains in dimension n). Usually we divide a rectangle Q into rectangles and a cube into cubes. This decomposition is rather efficient since the ratio between the surface and the volume is minimal and this balances the work required to the processors. This type of domain decomposition is called "chess board" decomposition. The domain Q is cut in every direction: the optimal choice is obtained when $D \equiv q^n$, where n is the space dimension and $q \in \mathbb{N}$, e.g. for $n = 3$ this gives $D = 1, 8, 27, \ldots$.

We exploit in full the local version of the algorithm where the value at the node x_{ij} at the $(p + 1)$-th iteration can be computed just knowing the values at the p-th iteration on the nodes wich are "first neighbours" of x_{ij}. If the first neighbours are strictly inside the domain Q_d all the informations are stored in the memory of the processor P_d, whereas if $x_{ij} \in \partial Q_d \cap \partial Q_g$ those informations must flow between the processors P_d and P_g. The values of the solution on the nodes belonging to the internal boundaries must be communicated to all the processors computing x_{ij}. This allows to minimize the communication load between the processors since only the boundary values are sent out through the network to the neighbouring processors. Our algorithm belong to the class SPMD (Single Program Multiple Data), which means that every processor executes the same program on different data sets. We have adopted the parallelization protocol MPI but the same approach can be used with other protocols like PVM or OpenMPI.

The algorithm has two main parts: the Master program and the SPMD program. The *Master program* manages the initialization of the data and the job assignments to the processors. This part is executed by a single processor so it is serial. It is preliminary with respect to the real parallel computation since

the Master program makes a link between every processor and every subdomain, send a copy of the program to be executed to every processor and send the data corresponding to every subdomain to its processor. The *SPMD Program* includes the numerical scheme and the communication commands. This is the parallel part since the same program is running on every processor and the informations are sent to the D processors which constitute the virtual parallel machine. The Master will be in stand-by during the execution of the SPMD program on the processors, it just waits for the results of the parallel part of the algorithm. Every processor P_d sets its own initial condition in its domain Q_d according to the rule

$$W_{ij}^0 = \begin{cases} 0 & \text{for any } x_{ij} \in \Omega_0 \\ 1 & \text{for any } x_{ij} \in Q_d \setminus \Omega_0 \end{cases}$$

As we have already seen, in order to compute the solution U^{p+1} at the node $x_{i,j}$ at the $(p+1)$-th iteration the algorithm needs the values of U^p at the 8 neighbouring nodes to $x_{i,j}$ (in $\mathbb{R}^2$) and at the 26 neighbouring nodes $x_{i,j,m}$ (in $\mathbb{R}^3$). Every processor has to send at every iteration the values corresponding the boundary nodes of its subdomain to the "neighbouring" nodes. Let us see how the procedure works in the 2D case.

For example, the P_3 processor must send the value at its node $x_{1,1}$ to the P_1 processor, the vector $U_{1,i}$ to P_2 and so on. When P_1 receives from P_3, the value $U_{1,1}$ copies it on $U_{maxn,maxn}$ (in Figure 1 this is indicated by a small circle). P_2 must receive from P_3 the values $U_{1,i}$ and copies these values on $U_{maxn,i}$ (in Figure 1 this is indicated by the line of small circles). Every processor computes the

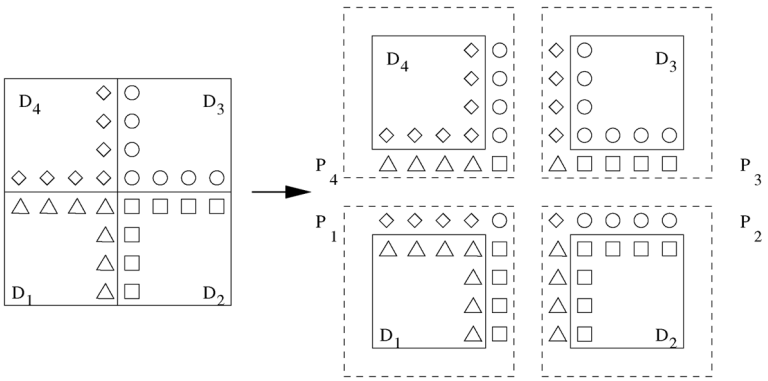

Fig. 1: Domain decomposition and mapping of the nodes on internal boundaries

solution on the nodes $x_{1,1}, \cdots, x_{maxn-1,maxn-1}$, but this computation requires an additional frame $x_{0,0}, \cdots, x_{maxn,maxn}$ which is filled with the values received by the "neighbouring" processors. Naturally the $3D$ algorithm has the same structure but its more difficult to handle since more values (the facets) have to be

sent around. Note that the SPMD Program which computes the (local solution) in every subdomain Q_d coincides with the serial algorithm (4.15) described in the previous section.

In order to show the speed-up, the efficiency and the communication load of the algorithm we made a numerical simulation tracking the evolution of a sphere centered at the origin which grows with speed 1.

The domain Q is a cube and we have compared the results for several domain decompositions where the number of subdomains (and processors) is

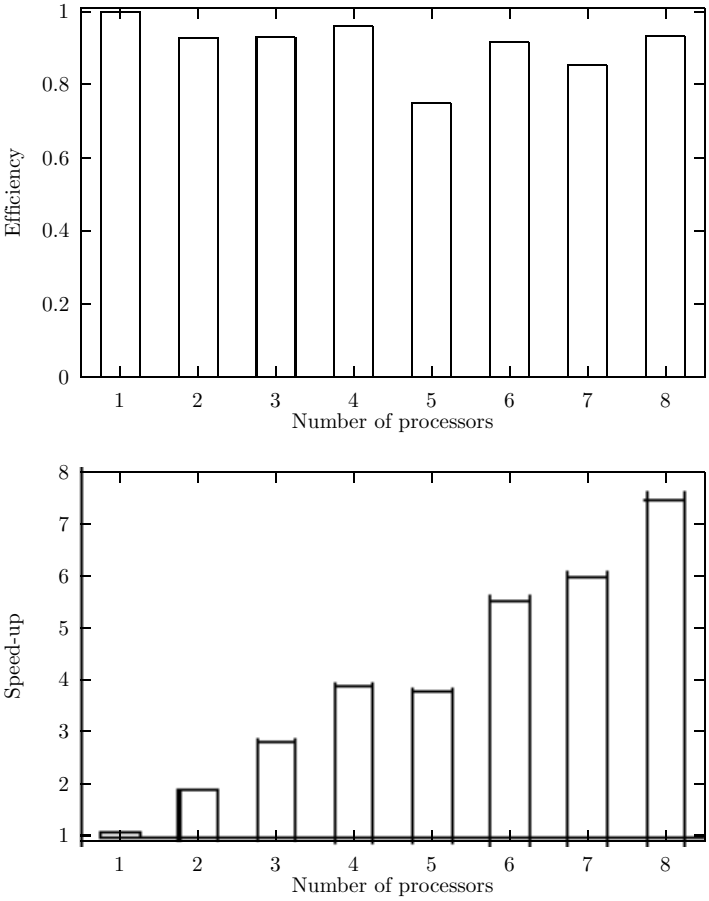

Fig. 2: Efficiency and speed-up of the parallel algorithm

$D = 1, \cdots, 8$. When $D = 8$ the cube is divided into cubes of equal volume, whereas for $D = 3, 5, 7$ we get a non uniform distribution of the nodes among

the processors. The case $D = 3, 5, 7$ is unbalanced in the computation load *and* in the communication load (see Figure 2).

Figure 3 shows that the amount of data, measured in bytes, that a processor must send is minimum for $D = 8$.

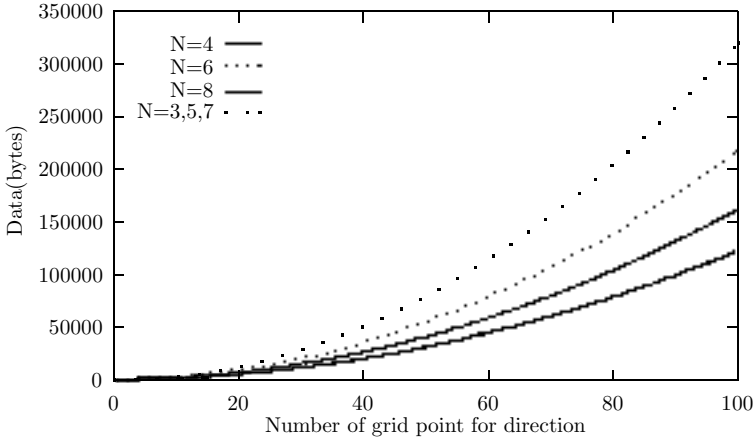

Fig. 3: Data sent by every processor at every iteration

6 – Numerical experiments

This section is devoted to the numerical experiments in $\mathbb{R}^2$ and $\mathbb{R}^3$. The first set of tests is without obstacles in order to show the accuracy of the scheme and its stability face to topological changes. The second set of test deals with obstacles to show how the schemes behaves when transparent boundary conditions are imposed. In all the tests the solution of the stationary problem (1.2) is computed only once and *all* the front configurations are obtained from that information. In fact, we can get the front Γ_t at time t for any $t \leq T_{max}$ just drawing the corresponding level set, i.e. $\Gamma_t = \{x \in Q : T(x) = t\} = \{x \in Q : v(x,t) = 1 - \exp(-t)\}$.

The runs have been made on the Compaq AlphaServer ES40 (CPU EV6 - 500MHz) at the CASPUR consortium, unless a different indication is given.

6.1 – 2D Tests

6.1.1 – Tests on free evolution

The problem is considered in $Q = [-2, 2]^2$, with a space step $k = 0.0314$ (128×128 grid), a normal speed $c \equiv 1$ and a time step $h = 0.0314$. The solution

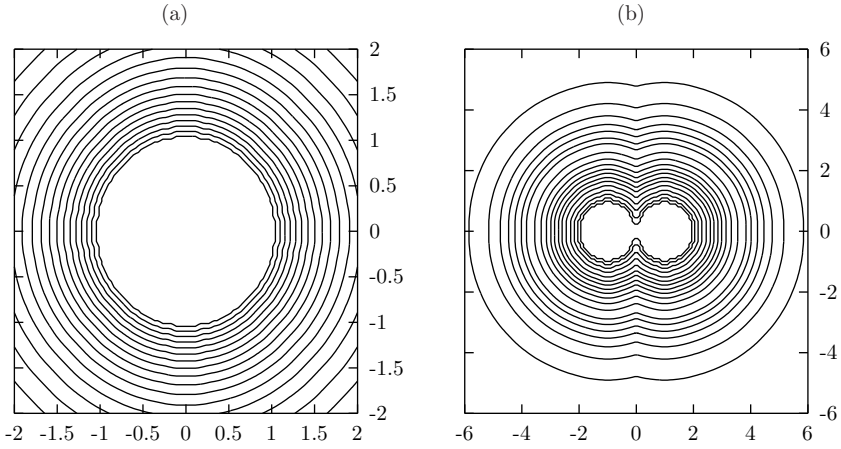

Fig. 4: Evolution of a circular front and merging of two circular fronts

is obtained after 80 iterations with a CPU time of 4.3 seconds. The initial front configuration is a circle of radius $r = 1$, centered at (0,0). The solution is shown in Figure 4a. The circular front evolves in the normal direction keeping its circular shape. Table 1 shows the error (in the max norm) between the exact and the approximate solution on grids with an increasing number of nodes N.

Nodes	$\Delta x = \Delta t$	Max Error	Iter	Nodes $\in \Omega_0$	CPU Time
32^2	0.1290323	0.1174259	27	188	0.26 sec
64^2	6.3492067E-02	5.2653372E-02	44	788	0.9 sec
128^2	3.1496063E-02	3.0788243E-02	80	3168	4.3 sec
256^2	1.5686275E-02	1.4527082E-02	145	12796	24 sec
512^2	7.8277886E-03	7.7226162E-03	271	51288	2 min 39 sec
1024^2	3.9100684E-03	3.8992167E-03	518	205508	20 min 34 sec

Table 1. Errors in the max norm

The second example is computed in $Q = [-6, 6]^2$ with a space step $k = 0.094$ (128×128 grid), speed $c \equiv 1$. The initial front configuration is a disconnected domain made by two circles of radius $r = 1$, the first is centered at $A \equiv (-1, 0)$ whereas the second is centered at $B \equiv (1, 0)$.

The solution is shown in Figure 4b. As it is expected, the two circular fronts evolve in the normal direction until they merge in a single front. Table 2 shows the L^∞ error for an increasing number of grid nodes.

Nodes	$\Delta x = \Delta t$	Max error	Iter	Nodes $\in \Omega_0$	CPU Time
32^2	0.3870968	0.2906976	26	44	0.28 sec
64^2	0.1904762	0.1689198	48	168	1.1 sec
128^2	9.4488189E-02	8.6824715E-02	87	708	5.3 sec
256^2	4.7058824E-02	4.5691967E-02	165	2832	31 sec
512^2	2.3483366E-02	2.3140967E-02	319	11400	3 min 34 sec
1024^2	1.1730205E-02	1.1644483E-02	618	45640	26 min 38 sec

Table 2. Errors for the merging of two circular fronts

6.1.2 – Tests on front propagations with obstacles

Let us consider a front evolution starting from a circle of radius $r = 0.1$, velocity $c(x) \equiv 1$ in $Q = [-2, 2]^2$. In these tests some obstacles $\mathcal{O}$ are present in Q and the transparent boundary conditions are imposed on $\partial\mathcal{O}$ and ∂Q as we have explained in Section 3. In Figure 5 a, b, c, d one can see the fronts at different times for four different types of obstacles: a C-shaped obstacle, a rectangle, a diamond and a set of randomly distributed rectangles. Initially the front evolves keeping its circular shape, then it hits the obstacle which has a sharp corner and starts to get around the obstacles closing on itself after the obstacle. Note that in the limit for t going to $+\infty$ the shape must always be a circle.

As one can see the treatment of transparent boundary conditions is rather effective since the front propagation is neither deviated nor reflected by the obstacle boundaries. This is particularly evident in the test with the C-shaped obstacle where the front hits the lower left vertex and then proceeds on the left and bottom sides with two different angles with respect the normal to the obstacle. The same happens in the rectangular obstacle test. The diamond test show a different phenomenon. The front hits the obstacle, flows around it and closes back after the obstacle tending again to a circular shape configuration. The same happens in the last and more complicated example where many rectangular obstacles are randomly distributed in the domain of computation. Our implementation of the boundary conditions allows to treat very complex boundaries in a simple way.

Obstacle Shape	Nodes $\in \Omega_0$	iter	CPU Time
C-shape	30	167	6.1 sec
Rectangle	30	158	6.0 sec
Diamond	30	161	6.3 sec
Random Rectangles	332	112	4 min 4 sec

Table 3. CPU time for the tests with obstacles.

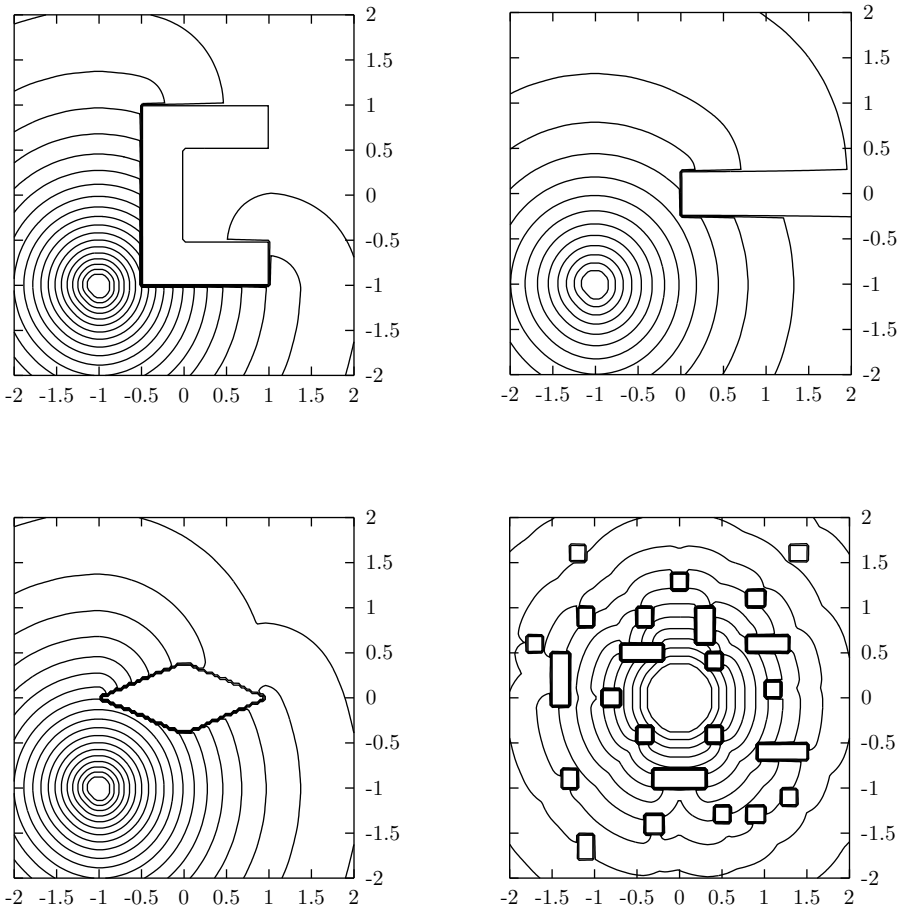

Fig. 5: Front propagation in the presence of obstacles in $\mathbb{R}^2$

Table 3 shows the CPU times and the number of fixed point iterations needed to converge. Table 4 shows the errors in the max-norm for the test with a rectangular obstacle. Since the exact solution is not available, the errors were computed comparing the approximate solution to the approximate solution obtained on the grid with 801^2 nodes (last row in the table).

Nodes	$\Delta x = \Delta t$	Iter	Nodes $\in \Omega_0$	CPUTime	Max error
101^2	0.04	127	21	3.1 sec	0.02426
201^2	0.02	239	77	21.2 sec	0.01452
401^2	0.01	457	313	2 min 32 sec	0.00486
801^2	0.005	806	1253	20 min 26 sec	–

Table 4: Errors for the rectangular obstacle test.

6.2 – 3D Tests

In $\mathbb{R}^3$ we considered evolutions in $Q = [-3,3]^3$ with 100^3 nodes in the grid. Figure 6 shows the evolution of a torus centered at the origin with internal radius $R = 0.8$ and circular section of radius $r = 0.1$. The speed is constant $c = 1$ and the pictures describe the interface at different times, $t = 0, .174, .385, .635, 1.021, 2.040$. Table 4 shows the CPU time for an increasing numeber of nodes on an IBM Power3.

Nodes	$\Delta x = \Delta t$	Iter	Nodes $\in \Omega_0$	CPU Time
32^3	0.19354	31	240	2 min 12 sec
64^3	9.5238E-02	56	2016	31 min 40 sec
80^3	7.5949E-02	69	4128	1h 16 min 09 sec
90^3	6.7416E-02	77	5824	2h 02 min 07 sec
100^3	6.0606E-02	84	8032	3h 01 min 58 sec

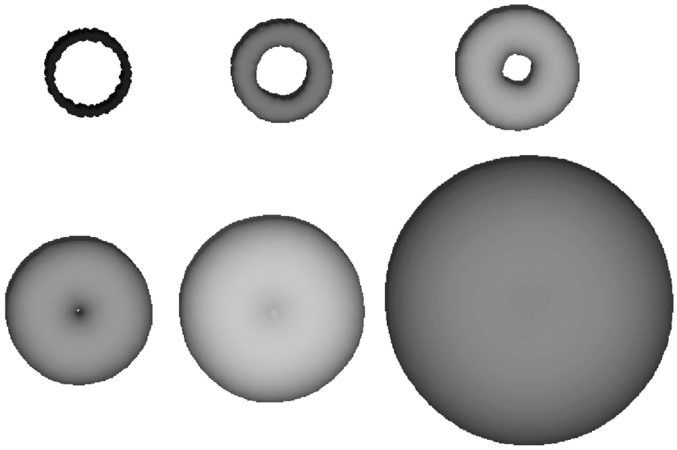

Fig. 6: Evolution of a torus

Figure 7 shows the evolution of a sphere centered at the origin and initial radius $r = 0.4$. The speed is constant $c = 1$ and an n-rectangular obstacle lies inside the domain of computation $Q = [-3,3]^3$, discretized with 100^3 nodes, $dx = .0606$. The pictures show the position of the interface at different times, $t = 0, .635, 1.078, 1.386, 1.7141, 2.525$. The global computing time needed to complete the 98 fixed point iterations has been of 51 min 15 sec.

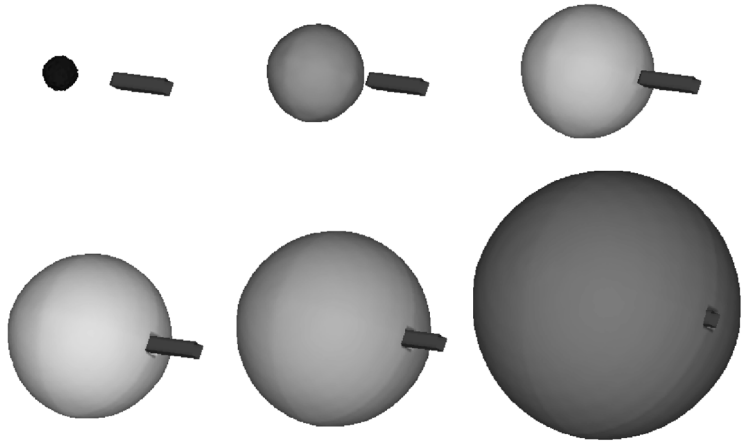

Fig. 7: Spherical interface evolving around a polyhedral obstacle

REFERENCES

[1] T.D. Aslam – J.B. Bdzil – D. Scott Stewart: *Level set methods applied to modeling detonation shock dynamics*, J. Comput. Phys., **126** (1996), 390-409.

[2] M. Bardi – M. Falcone: *An approximation scheme for the minimum time function*, SIAM J. Control Optim.,**28** (1990) 950-965.

[3] M. Bardi – I. Capuzzo Dolcetta: *Optimal control and viscosity solutions of Hamilton–Jacobi–Bellman equations*, Birkhäuser, 1997.

[4] G. Barles: *Solutions de viscositè des equations d'Hamilton–Jacobi*, Springer–Verlag, 1998.

[5] F. Camilli – M. Falcone: *Approximation of optimal control problems with state constraints: estimates and applications*, in B.S. Mordukhovic, H.J. Sussman (eds.), "Nonsmooth analysis and geometric methods in deterministic optimal control", IMA Volumes in Applied Mathematics 78, Springer Verlag, 1996, 23-57.

[6] I. Capuzzo Dolcetta – P.L. Lions: *Hamilton-Jacobi equations with state constraints*, Trans. Amer. Math. Soc., **318** (1990), 643-683.

[7] R. Courant – E. Isaacson – M. Rees: *On the solution of nonlinear hyperbolic differential equations by finite differences*, Comm. Pure Appl. Math. **5** (1952), 243-255.

[8] B. Engquist – R. Clayton: *Absorbing boundary conditions for acoustic and elastic wave equations*, Bull. Seismol. Soc. Amer., **67** (1977), 1529-1540.

[9] M. Falcone: *The minimum time problem and its applications to front propagation*, in G. Buttazzo e A. Visintin (eds) "Motion by mean curvature and related topic", (Trento, 1992), 70–88, de Gruyter, Berlin, 1994.

[10] M. Falcone: *Numerical solution of Dynamic Programming equations*, Appendix A in [3].

[11] M. Falcone – R. Ferretti: *Semi-Lagrangian schemes for Hamilton-Jacobi equations, discrete representation formulae and Godunov methods*, Journal of Computational Physics, **175** (2002), 559-575.

[12] M. Falcone – R. Ferretti: *Consistency of a large time–step scheme for mean curvature motion*, in F. Brezzi, A. Buffa, S. Corsaro, A. Murli (eds), "Numerical Mjathematics and Advanced Applications-ENUMATH 2001", Springer Verlag, 2003, 495-502.

[13] M. Falcone – T. Giorgi: *An approximation scheme for evolutive Hamilton–Jacobi equations*, in W.M.McEneaney, G. Yin, Q. Zhang (eds), "Stochastic analysis, Control, optimization and applications: a volume in honor of W.H. Fleming", Birkhäuser, 1999, 289-303.

[14] M. Falcone – T. Giorgi – P. Loreti: *Level sets of viscosity solutions and applications*, SIAM J. Appl. Math., **54** (1994), 1335-1354.

[15] M. Falcone – Ch. Makridakis (eds): *Numerical Methods for Viscosity Solutions and Applications*, World Scientific, Singapore, 2001.

[16] M. Falcone – P. Lanucara – F. Massaioli – M. Rosati – C. Truini: *The flame front propagation problem on the SIMD architecture QUADRICS*, E. Hollander, G.R. Joubert, F.J. Peters and D. Trystram (eds.), Parallel Computing: State-of-the-Art and Perspectives , Elsevier, 1996, 85-92.

[17] R. Ferretti: *Convergence of semi–Lagrangian approximations to convex Hamilton–Jacobi equations under (very) large Courant numbers*, SIAM J. Num. Anal., **40** (2003), 2240-2253.

[18] H. Ishii – S. Koike: *A new formulation of state constraint problems for first order PDEs*, SIAM J. Control and Optimization, **34** (1996), 544-571.

[19] T.G. Lewis – H.E. Rewini: *Introduction to parallel computing*, Prentice-Hall International Eds, New Jersey, 1992.

[20] S. Osher: *A level set formulation for the solution of the Dirichlet problem for Hamilton–Jacobi equations*, SIAM J. Math. Anal., **24** (1993), 1145-1152.

[21] S. Osher – J.A. Sethian: *Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations*, J. Comput. Phys. **79** (1988), 12-49.

[22] A. Quarteroni – A. Valli: *Domain decomposition methods for partial differential equations*, Oxford University Press, 1999.

[23] M. Sagona – A. Seghini: *An adaptive scheme for the Shape-from-Shading problem*, in [15], 197-219.

[24] J.A. Sethian: *Level Set Method. Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science*, Cambridge Monographs on Applied and Computational Mathematics, vol. 3, Cambridge University Press, Cambridge, 1996.

[25] J.A. Sethian: *Fast marching methods*, SIAM Review **41** (1999), 199-235.

[26] J.A. Sethian – A. Vladimirsky: *Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms*, SIAM J. Numer. Anal., **41** (2003), 325-363.

[27] J.A. Sethian – A. Vladimirsky: *Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes*, Proc. Natl. Acad. Sci. USA **97** (2000), 5699-5703.

[28] A.N. Staniforth – J. Côté: *Semi–Lagrangian integration schemes for atmospheric models – A review*, Mon. Wea. Rev., **119** (1991), 2206-2223.

[29] H.M. Soner: *Optimal control problems with state-space constraints I and II*, SIAM J. Control and Optimization, **24** (1986), 551-561 and 1110-1122.

[30] P. Soravia: *Generalized motion of a front propagating along its normal direction: a differential game approach*, Nonlinear Anal. TMA, **22** (1994) 1247-1262.

[31] J. Strain: *Semi-Lagrangian methods for level set equations*, J. Comput. Phys. **151** (1999), 498-533.

Lavoro pervenuto alla redazione il 1 dicembre 2008
ed accettato per la pubblicazione il 2 febbraio 2009.
Bozze licenziate il 18 marzo 2009

Indirizzo degli Autori:

M. Falcone – Dipartimento di Matematica – Sapienza Università di Roma – P. Aldo Moro, 2 – 00185 Roma, Italy
E-mail: falcone@mat.uniroma1.it

C. Truini – CASPUR, Consortium for Supercomputing Applications in Research – Sapienza Università di Roma – P. Aldo Moro, 2 – 00185 Roma – Italy
E-mail: truini@caspur.it