PARALLEL IMPLEMENTATIONS OF THE FAST SWEEPING METHOD

HONGKAI ZHAO*

Abstract. The fast sweeping method is an efficient iterative method for hyperbolic problems. It combines Gauss-Seidel iteration with alternating sweeping ordering. In this paper several parallel implementations of the fast sweeping method are presented. These parallel algorithms are simple and efficient due to the causality of the underlying partial different equations. Numerical examples are used to verify our algorithms.

AMS subject classifications. 65N06, 65N12, 65N15, 35L60.

Keywords. Hamilton-Jacobi equation, Eikonal equation, characteristics, viscosity solution, upwind difference, Courant-Friedrichs-Levy (CFL) condition, Gauss-Seidel iteration, domain decomposition.

1. Introduction. The fast sweeping method is an iterative method that combines Gauss-Seidel iteration with alternating sweeping ordering. It was first introduced in [1] from the stochastic control point of view and then in [10] from partial differential equation point of view. For Eikonal type of equations it is fully analyzed and shown in [9] that the number of iteration is finite and is independent of mesh size. The fast sweeping method has been developed recently in [4] for unstructured meshes. The method has also been extended to more general Hamilton-Jacobi equations [6, 2, 3] and high-order methods [8].

The convergence mechanism for fast sweeping method is quite different from the general framework of iterative methods. For most iterative methods, the convergence is due to some contraction property of the iteration. The fact that the fast sweeping method can converge in a finite number of iterations, e.g., for Eikonal equations which is a nonlinear hyperbolic boundary value problem, is because of the causality of the partial differential equation: information propagates along characteristics. With a systematic alternating ordering strategy all directions of characteristics can be divided into a finite number of groups and each group of directions is covered simultaneously by one of the orderings. Moreover, any characteristic can be covered by a finite number of orderings [9]. From the discrete point of view, the discretized system of nonlinear equations can be put in a triangular form if all nodes are ordered in an increasing order according to their values, i.e., if we interpret the viscosity solution to the Eikonal equation as the first arrival time for some expanding wave front earlier arrival time determines later arrival time. A triangular system can be solved one equation by one equation in one sweep. Of course the problem is we do **not** know the solution so there is no such ordering a priori. That is why systematic alternating orderings are needed to cover all directions of causality blindly. In other words, with an appropriate upwind scheme that captures information propagation along characteristics correctly, we can characterize all nodes into a few groups. All nodes in each group have a similar dependence pattern on their neighbors. For example, using the monotone upwind difference scheme (2.2) on a rectangular grid in two dimensions for an Eikonal equation, almost all grid points can be divided into simply connected regions. In each region the value at a grid point depends on two of its neighbors in the following four ways: (1) left and down neighbors; (2) left and up neighbors; (3) right and down neighbors; (4) right and up neighbors. Using Gauss-Seidel iteration each connected region can be covered by one of the orderings simultaneously when the ordering is in the upwind direction of the dependence pattern. Grid points at or near intersections of characteristics

*Department of Mathematics, University of California, Irvine, CA 92697-3875; zhao@math.uci.edu.
Work partially supported by Sloan Fundation, NSF DMS0513073, ONR grant N00014-02-1-0090 and DARPA grant N00014-02-1-0603.

may have more complicated dependence on their neighbors when certain symmetry exists, e.g., equal distance points, these points may cause a few more iterations. However, even without these few extra iterations the accuracy is not going to be affected ([9]).

Since the fast sweeping method is an iterative method, a natural question is: can the fast sweeping method be parallelized? The answer is yes and simple. This is again due to the causality of the partial differential equation. We will present a few possible versions of parallel implementations of the fast sweeping method in section 2 and show their efficiencies in section 3 using numerical examples. Note that a parallel algorithm based on fast marching method (expanding front) was proposed in [7]. Although the above algorithm can achieve the optimal complexity $O(n/p)$, where n is the total number of grid points and p is the number of processors, the algorithm works for a special discretization scheme. In addition a load balancing algorithm is needed at each stage. Our domain decomposition based parallel algorithms is very simple to implement and work for more general discretizations.

2. Parallel Algorithms For The Fast Sweeping Method. For simplicity we restrict our discussions to the computation for Eikonal equation in two dimensions on rectangular grids. Extensions to unstructured meshes and higher dimensions is straightforward. Assume we want to compute the positive viscosity solution of the following boundary value problem

$$\begin{array}{ll} |\nabla u(\boldsymbol{x})| = f(\boldsymbol{x}) & \boldsymbol{x} \in R^2 \\ u(\boldsymbol{x}) = 0 & \boldsymbol{x} \in \Gamma \subset R^2, \end{array} \tag{2.1}$$

where $f(\boldsymbol{x}) > 0$. Denote $\boldsymbol{x}_{i,j}$ to be a grid point in the computational domain Ω, h to be the grid size and $u^h_{i,j}$ to be the numerical solution at $\boldsymbol{x}_{i,j}$. First we give a brief description of the fast sweeping algorithm [9].

Discretization: At interior grid points Godunov type of upwind scheme is used to discretize the partial differential equation (2.1) which results in the following system of nonlinear equations

$$\begin{array}{l} [(u^h_{i,j} - u^h_{xmin})^+]^2 + [(u^h_{i,j} - u^h_{ymin})^+]^2 = f^2_{i,j} h^2 \\ i = 2, \ldots, I-1, \ j = 2, \ldots, J-1, \end{array} \tag{2.2}$$

where $u^h_{xmin} = \min(u^h_{i-1,j}, \ u^h_{i+1,j}), \ u^h_{ymin} = \min(u^h_{i,j-1}, u^h_{i,j+1})$ and $(x)^+ = \begin{cases} x & x > 0 \\ 0 & x \le 0 \end{cases}$. Appropriate conditions are used at the boundary of the computational domain. For example, one sided difference can be used at the outflow boundary, e.g., at a left boundary point $\boldsymbol{x}_{1,j}$, a one sided difference is used in the x direction,

$$[(u^h_{1,j} - u^h_{2,j})^+]^2 + [(u^h_{1,j} - u^h_{ymin})^+]^2 = f^2_{1,j} h^2.$$

Initialization: To enforce the boundary condition, $u(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \Gamma \subset R^n$, assign exact values or interpolated values at grid points in or near Γ. These values are fixed in later calculations. Assign large positive values at all other grid points. These values will be updated later.

Gauss-Seidel iterations with alternating orderings: At each grid $\boldsymbol{x}_{i,j}$ whose value is not fixed during the initialization, compute the solution, denoted by $\overline{u}$, of (2.2) from the current values of its neighbors $u^h_{i\pm1,j}, u^h_{i,j\pm1}$ and then update $u^h_{i,j}$ to be the

smaller one between $\overline{u}$ and its current value, i.e., $u_{i,j}^{new} = \min(u_{i,j}^{old}, \overline{u})$. We sweep the whole domain with four alternating orderings repeatedly,

$$\begin{array}{ll} (1)\ i = 1 : I, j = 1 : J & (2)\ i = I : 1, j = 1 : J \\ (3)\ i = I : 1, j = J : 1 & (4)\ i = 1 : I, j = J : 1 \end{array}$$

One of the crucial point in the algorithm is that we update the value at a point if and only if the new value is smaller than the old value. This updating rule comes from the fact that it is better if it is smaller due to the causality of the partial differential equation, i.e., the physical interpretation of first arrival time. When a grid value reaches the smallest possible value it is the true value and it will not be changed in later iterations. This simple causality also allows us to develop the following parallel algorithms for the fast sweeping method easily.

2.1. Parallel Sweeping. We parallelize the fast sweeping algorithm by implementing sweeping with different orderings in parallel, i.e., we assign different sweepings to different CPUs and compute simultaneously. For example, in two dimensions, we start with the same initial guess and assign the four Gauss-Seidel iterations with different orderings to four CPUs. After the first iteration, e.g., the four sweepings are finished, we get four solutions $u_1^{(1)}, u_2^{(1)}, u_3^{(1)}, u_4^{(1)}$ from the four different computations. We take minimum value of these four solutions at each grid point

$$u^{(1)} = \min\{u_1^{(1)}, u_2^{(1)}, u_3^{(1)}, u_4^{(1)}\} \tag{2.3}$$

and use this as our updated solution in the computation domain and start the second iteration using four CPUs for different orderings as in the first iteration. This process is continued until convergence.

Again, all directions of characteristics will be covered by one of the sweeping orderings. Taking the minimum among the four simultaneously computed solutions will guarantee the updated solution will get better after each iteration at all grid points. In another interpretation, we can view the original version of fast sweeping method is multiplicative or sequential for the four different orderings while this new version is additive or parallel for the four different orderings.

2.2. Domain Decomposition. In a standard domain decomposition setup we divide the whole computation domain into subdomains and assign the computation (using the fast sweeping method) in different subdomains to different CPUs. The two crucial issues for a domain decomposition method are:

1. how to communicate between computations in different subdomain?
2. is overlapping between adjacent subdomains needed?

The answer to the second question is no. We only require two adjacent subdomains to share a common boundary. Again due to the simple causality of the partial differential equation the communication between adjacent subdomains are extremely simple. We call the shared boundary between adjacent subdomains an internal boundary. At internal boundaries we have multiple values from those subdomains that share those internal boundaries. The communication among the subdomains is to take the minimum value at the internal boundaries after each update. In another word, we use the simple causality for first arrival time, the smaller one is the better one. By this updating rule we always have a uniquely defined solution in the whole domain and the solution is monotonically decreasing at all grid points. This simple communication between adjacent subdomains guarantees we only pass correct information from one subdomain to its adjacent subdomains.

Of course there could be several different implementations for this domain decomposition algorithm.

- We can iterate the fast sweeping method in each subdomain until it converges before we communicate between subdomains. However, this may cost unnecessary extra computations since information from other subdomains has not passed through yet. Another extreme is to exchange information after one sweeping in each subdomain. However, one sweeping in a particular order may not be enough to propagate useful information in all directions in each subdomain and may cost too many unnecessary communications. A good compromise would be to finish four different sweepings in each subdomain before exchange of information at internal boundaries.
- We can have additive version, i.e., computations in subdomains are done simultaneously, or multiplicative version, i.e., computations in subdomains are done sequentially. For the multiplicative version, again we can sweep the subdomains in different orderings which may accelerate information propagation in different directions through subdomains and hence improve the convergence speed.
  Here we give a concrete example to describe these two versions. Suppose the whole computation domain Ω is decomposed into four subdomains, $\Omega_i, i = 1, 2, 3, 4$, as shown in Fig. 2.1. First we initialize the solution as in the fast sweeping algorithm described above. For the additive version, we compute the solution in each subdomain simultaneously using the fast sweeping algorithm. After the computation is completed in each subdomain we have two values at $\Gamma_{12}, \Gamma_{13}, \Gamma_{24}, \Gamma_{34}$ and four values at the point O. We just define the updated solution at these internal boundaries to be the minimum one from all possible values. Then we start next iteration in all subdomains. This procedure is repeated until convergence. For the multiplicative version it is even simpler since the most updated value at each grid point is always used. For example, after the computation is finished in Ω_1, the updated values at the internal boundary Γ_{12} shared by Ω_1 and Ω_2 will be involved and updated during the following computation in Ω_2.

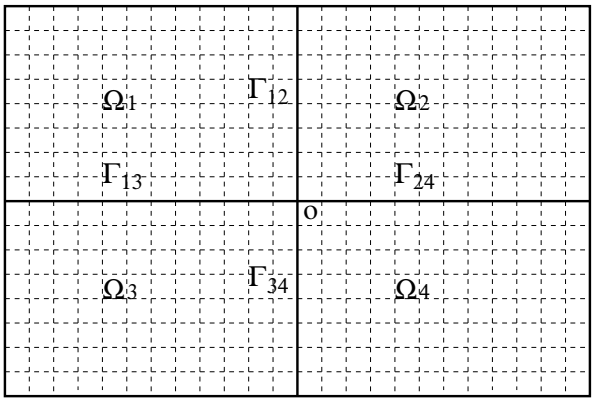

FIG. 2.1. *An example of domain decomposition*

Use the same arguments from [9] we can easily see

1. The above parallel algorithms will converge since the scheme is monotone and the solution at all grid points are decreasing and getting better with iterations.

2. The above parallel algorithms converges in a finite number of iterations and is independent of grid size since the fast sweeping algorithm in each subdomain converges in a finite number of iterations and information propagates through all subdomains in a finite number of communications through internal boundaries.

Remark1: Since information may need to propagate through subdomains, the number of iterations may depend on the number of subdomains for the additive version. For a fixed domain decomposition configuration the number of iterations needed for convergence is independent of grid size because the convergence of fast sweeping method in each subdomain is independent of grid size. For the multiplicative version if we sweep the subdomains in different orderings the number of iterations is also independent of the number of subdomains.

Remark 2: Our parallel algorithms are based on the hyperbolic nature and its causality which is totally different from elliptic problems. These algorithms work for Hamilton-Jacobi equations where solution is always increasing (or decreasing) along characteristics.

3. Numerical Examples. In this section we use numerical tests to verify the proposed parallel algorithms. The discretization is as described in (2.2). The numerical errors are the same as those shown in [9]. Here we only want to demonstrate the convergence behavior of different parallel algorithms.

We present four examples to test our algorithms. Contour plots of the solutions are shown in Figure 3.1. All computation domains are unit square $\Omega = [0,1] \times [0,1]$.

Example 0: Compute the distance function to the center point (0.5,0.5).

Example 1: Compute the distance function to 100 random points in the unit square, see Fig. 3.1(a).

Example 2: Compute the distance function to four circles, centered at (0.2, 0.2), (0.2, 0.8), (0.8, 0.2), and (0.8, 0.8) with radius 0.15, see Fig. 3.1(b).

Example 3: Compute the distance function to the lower-left corner with five ring obstacles, see Fig. 3.1(c). In the five rings the propagation speed is zero, i.e., $f(x,y) = 1$ outside the rings and $f(x,y) = \infty$ inside the rings (in actual implementation those grid points inside the rings are explicitly marked and their values are never updated).

Example 4: Compute the solution to a shape from shading problem in [5], see Fig. 3.1(d). In the Eikonal equation (2.1) we have

$$f(x,y) = 2\pi\sqrt{[\cos(2\pi x)\sin(2\pi y)]^2 + [\sin(2\pi x)\cos(2\pi y)]^2}. \tag{3.1}$$

and values are prescribed at five isolated points.

$$u(\frac{1}{4},\frac{1}{4}) = u(\frac{3}{4},\frac{3}{4}) = u(\frac{1}{4},\frac{3}{4}) = u(\frac{3}{4},\frac{1}{4}) = 1, u(\frac{1}{2},\frac{1}{2}) = 2.$$

$u(x,y) = 0$ is prescribed at the boundary of the unit square. The solution for this problem is the shape function, which has the brightness $I(x,y) = 1/\sqrt{1+f(x,y)^2}$ under vertical lighting. See [5] for details.

First we show the convergence results for the original fast sweeping method for reference. Table 3.1 gives the number of sweeping needed for convergence to machine zero on different grid size. Here each sweeping is one iteration that goes through all grid points in the computation domain with a particular ordering. We see a finite number of iterations independent of mesh size for each example when the grid is refined. Here are a few remarks to clarify some of the examples. For Example 0, the distance function to a single point, exact four sweepings are needed. For Example 1, the distance to 100

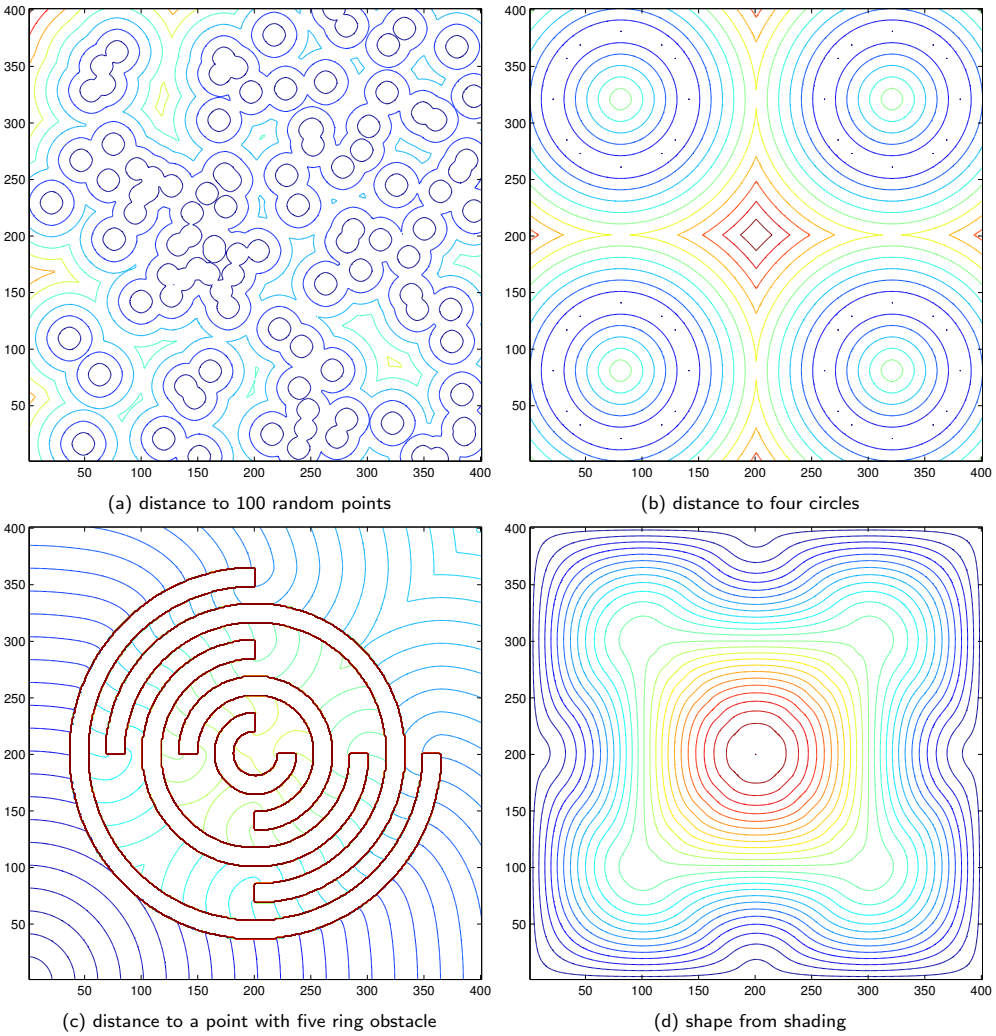

FIG. 3.1.

random points, more than four sweepings are needed for full convergence. However as explained in [9], after four sweepings the solution is as accurate as the fully converged solution. Interactions of characteristics at shocks may need more iterations and minor modifications to settle down. For Example 3, distance to a point with five ring obstacles, the number of sweepings settles down after the complicated geometric configuration can be resolved by the grid.

Now we present the number of iterations needed for convergence for different parallel algorithms. Table 3.2 shows the results for the parallel sweeping algorithm described in Section 2.1. In the table one iteration includes four sweepings through the computation domain with different orderings in parallel. A communication among the four sweepings, i.e., taking the minimum values, are done after each iteration. The numerical results clearly show that the number of iterations are independent of mesh size. For Example 0, the distance to a single point, each sweeping gets the correct solution in a corresponding quadrant independent of other sweepings, e.g., grid points in the first quadrant get the

Number of sweepings for the original fast sweeping algorithm.

	example 0	example 1	example 2	example 3	example 4
h=1/100	4	8	4	18	8
h=1/200	4	8	4	18	8
h=1/300	4	9	4	16	8
h=1/400	4	8	4	16	8
h=1/500	4	9	4	16	8

TABLE 3.1

correct value in the sweeping with ordering $i = 1 : I, j = 1 : J$ since they all depend on their left and down neighbors. After four sweepings are finished a single communication among them will get the correct solution in the whole domain. In general if the characteristics are straight lines, sweepings with different orderings are almost independent of each other and the parallel sweeping algorithm should be as efficient as the the original fast sweeping algorithm. However if the characteristics are curved then different orderings implemented sequentially may propagate information faster on a curved characteristics than different orderings implemented in parallel.

Number of iterations for the parallel sweeping algorithm. Each iteration includes four sweepings of different orderings in parallel.

	example 0	example 1	example 2	example 3	example 4
h=1/100	1	4	2	8	3
h=1/200	1	4	2	9	3
h=1/300	1	4	2	8	3
h=1/400	1	4	2	8	3
h=1/500	1	4	2	8	4

TABLE 3.2

Table 3.3 shows the convergence results for the multiplicative domain decomposition algorithm described in Section 2.2, i.e., computations in the subdomains are done successively. In our numerical tests four sweepings with different orderings are performed in each subdomain before moving to the next subdomain. The communication among subdomains are only through the shared internal boundaries. In the table one iteration includes one pass through all subdomains and the computation in each subdomain is composed of four sweepings. First we fix the ordering of the subdomains, i.e., the order in which the subdomains are went through in each iteration is the same. Table 3.3 (a) and (b) show the results for 2x2 and 5x5 subdomains respectively. To converge in the whole computation domain information needs to propagate both in subdomains and among subdomains. More subdomains may result in more iterations. However, using the sweeping idea we can also sweep the subdomains in different orderings in different iterations. In this way information also propagates through subdomains efficiently. Table 3.3 (c) compared the multiplicative domain decomposition algorithm on Example 3 with and without different orderings for the subdomains. When sweeping the subdomains with different orderings the number of iterations is almost independent of the number of subdomains. While the number of iterations increases when the number of subdomains increases with the ordering of subdomains fixed.

Finally we show the convergence results for the additive domain decomposition algorithm, i.e., computations in the subdomains are done in parallel. In our numerical tests

Number of iterations for multiplicative domain decomposition algorithm. One iteration contains one pass through all subdomains and the computation in each subdomain includes four sweepings.

	example 0	example 1	example 2	example 3	example 4
h=1/100	2	4	2	9	4
h=1/200	2	4	2	9	4
h=1/300	2	4	2	9	4
h=1/400	2	4	2	9	4
h=1/500	2	4	2	9	4

(a) The computation domain is decomposed into 2x2 subdomains

	example 0	example 1	example 2	example 3	example 4
h=1/100	6	4	4	12	6
h=1/200	6	4	4	12	6
h=1/300	6	4	4	12	7
h=1/400	6	4	4	12	7
h=1/500	6	4	4	12	7

(b) The computation domain is decomposed into 5x5 subdomains

	2x2	4x4	8x8	16x16
use different ordering of the subdomains	11	12	12	13
use fixed ordering of the subdomains	9	12	16	27

(c) The computation domain is decomposed into different number of subdomains. The grid size is h=1/400.

TABLE 3.3

four sweepings with different orderings are performed in each subdomain. The communication among subdomains are only through the shared internal boundaries as described in 2.2. In the table one iteration includes simultaneous computations in all subdomains and the computation in each subdomain is composed of four sweepings. Since computations in all subdomains are done in parallel there is no ordering for the subdomains. Table 3.4 (a) and (b) shows the results for 2x2 and 5x5 subdomains respectively. We see that more subdomains may need more iterations to converge.

REFERENCES

[1] M. Boué and P. Dupuis. Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control. *SIAM J. Numer. Anal.*, 36(3):667–695, 1999.

[2] C. Kao, S. Osher, and J. Qian. Lax-friedrichs sweeping scheme for static hamilton-jacobi equations. *J. Comput. Phys.*, 2004.

[3] C. Kao, S. Osher, and R. Tsai. Fast sweeping methods for hamilton-jacobi equations. *SINUM*, 2005.

[4] J. Qian, Y. Zhang, and H. Zhao. Fast sweeping methods for eikonal equations on triangulated meshes. *submitted.*, 2005.

[5] E. Rouy and A. Tourin. A viscosity solution approach to shape-from-shading. *SIAM J Num Anal*, 1992.

[6] R. Tsai, L. Cheng, S. Osher, and H. Zhao. Fast sweeping algorithms for a class of hamilton-jacobi equations. *SINUM*, 2003.

[7] John N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 1995.

[8] Y. Zhang, J. Qian, and H. Zhao. High order fast sweeping methods for static hamilton-jacobi

Number of iterations for the additive domain decomposition algorithm. One iteration contains computations in all subdomains in parallel and the computation in each subdomain includes four sweepings.

	example 0	example 1	example 2	example 3	example 4
h=1/100	4	4	2	13	5
h=1/200	4	4	2	13	5
h=1/300	4	4	2	13	5
h=1/400	4	5	2	13	5
h=1/500	4	5	2	13	5

(a) The computation domain is decomposed into 2x2 subdomains

	example 0	example 1	example 2	example 3	example 4
h=1/100	6	5	4	22	7
h=1/200	6	5	4	23	7
h=1/300	6	5	4	23	7
h=1/400	6	6	4	23	7
h=1/500	6	6	4	23	7

(b) The computation domain is decomposed into 5x5 subdomains

TABLE 3.4

equations. *Journal of Scientific Computing*, to appear.

[9] H. Zhao. Fast sweeping method for eikonal equations. *Math. Comp.*, 2005.

[10] H. Zhao, S. Osher, B. Merriman, and M. Kang. Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. *Computer Vision and Image Understanding*, 80(3):295–319, 2000.