Parallel redistancing using the Hopf-Lax formula

Michael Royston, Andre Pradhana, Byungjoon Lee, Yat Tin Chow, Wotao Yin, Joseph Teran, Stanley Osher

University of California Los Angeles

Abstract

We present a parallel method for solving the eikonal equation associated with level set redistancing. Fast marching [1, 2] and fast sweeping [3] are the most popular redistancing methods due to their efficiency and relative simplicity. However, these methods require propagation of information from the zero-isocontour outwards, and this data dependence complicates efficient implementation on today's multiprocessor hardware. Recently an interesting alternative view has been developed that utilizes the Hopf-Lax formulation of the solution to the eikonal equation [4, 5]. In this approach, the signed distance at an arbitrary point is obtained without the need of distance information from neighboring points. We extend the work of Lee et al [4] to redistance functions defined via interpolation over a regular grid. The grid-based definition is essential for practical application in level set methods. We demonstrate the effectiveness of our approach with GPU parallelism on a number of representative examples.

Keywords:

level set methods, eikonal equation, Hamilton Jacobi, Hopf-Lax

1. Introduction

The level set method [6] is a powerful technique used in a large variety of problems in computational fluid dynamics, minimal surfaces and image processing [7]. In general these methods are concerned with the transport evolution $\frac{\partial \phi}{\partial t} + \mathbf{v} \cdot \nabla \phi$ of a level set function $\phi : \mathbb{R}^n \to \mathbb{R}$ in velocity field $\mathbf{v}$. While the essential property of ϕ is typically the location of its zero iso-contour

($\Gamma = \{\mathbf{x} \in \mathbb{R}^n | \phi(\mathbf{x}) = 0\}$), in practice many applications additionally require that it be a signed distance function ($|\nabla\phi| = 1$). This property will not generally be preserved by the transport evolution, but it can be recovered without modifying the location of the zero isocontour. This process is commonly referred to as redistancing or reinitialization [8, 9, 10].

We describe the redistancing problem in terms of the signed distance function ϕ that is obtained from an arbitrary non-signed-distance level set function ϕ^0 (while preserving the zero isocontour of ϕ^0). Mathematically, the process obeys the eikonal equation as

$$|\nabla\phi(\mathbf{x})| = 1 \tag{1}$$
$$\text{sgn}(\phi(\mathbf{x})) = \text{sgn}(\phi^0(\mathbf{x})).$$

There is extensive previous work related to the solution of (1). The most commonly used methods are the fast marching method (FMM) [1, 2] and the fast sweeping method (FSM) [3]. First proposed by Tsitsiklis [1] using optimal control, the fast marching method was independently developed by Sethian in [2] based on upwind difference schemes. It is similar to Dijkstra's method [11] for finding the shortest path between nodes in a graph. The fast marching method uses upwind difference stencils to create a discrete data propagation consistent with the characteristics of the eikonal equation. Sorting is used to determine a non-iterative update order that minimizes the number of times that a point is utilized to create a strictly increasing (or decreasing) propagation. The operation count is O(Nlog(N)) where N is the number of grid points and the log(N) term is a consequence of the sorting. Fast sweeping is similar, but it uses a simpler propagation. Rather than using the optimal update ordering that requires a heap sort, a Gauss-Seidel iterative approach is used with alternating sweep directions. Typically, grid axis dimensions are used as the sweep directions. For $\mathbb{R}^n$ it only requires 2^n propagation sweeps of updating to properly evaluate every point.

Notably, both the FMM and FSM approaches create data flow dependencies since information is propagated from the zero isocontour outwards and this complicates parallel implementation. Despite this, various approaches have achieved excellent performance with parallelization. The Gauss-Seidel nature of FSM makes it more amenable to parallelization than FMM. Zhao initially demonstrated this in [12] where each sweep direction was assigned to an individual thread with the final updated nodal value being the minimum nodal value from each of the threads. This method only allowed for a low number of threads and further scaling was achieved by splitting the individual sweeps into subdomain sweeps with a domain decomposition approach. However, this strategy can require more sweep iterations than the original serial FSM and the required iterations increase with the number of domains which reduces parallel efficiency. Detrixhe et al [13] developed a parallel FSM that scales in an arbitrary number of threads without requiring more iterations than in the serial case. Rather than performing grid-axis-aligned Gauss-Seidel sweeps, they use Cuthill-McKee ordering (grid-diagonal) to decouple the data dependency. Since the upwind difference stencil only uses grid axis neighbors, nodes along a diagonal do not participate in the same equation and can thus be updated in parallel trivially. They extended these ideas to hybrid distributed/shared memory platforms in [14]. They use a domain decomposition strategy similar to Zhao [12] to divide the grid among available compute nodes and a fine grained shared memory method within each subdomain that utilizes their approach in [13] to achieve orders of magnitude performance increases.

FMM is more difficult to implement in parallel, however even Tsitsiklis [1] developed a parallel FMM algorithm using a bucket data structure. A number of approaches use domain decomposition ideas similar to Zhao [12] and Detrixhe et al [14] to develop parallel FMM [15, 16, 17, 18]. In these approaches the grid is typically divided into disjoint sub grids with a layer of ghost nodes continuing into adjacent neighbors. Each sub grid is updated in parallel with the FMM update list typically leading to rather elaborate communication between threads.

Jeong et al [17] developed the fast iterative method (FIM), which is a parallel approach using domain decomposition but with a looser causal relationship in the node update list to localize updates for Single Instruction Multiple Data (SIMD) level parallelism. Simplifications to the update list in FMM improve parallel scaling, but tend to increase the number of worst case iterations. Dang et al [19] extended FIM to a coarse/fine-grained approach based on domain decomposition with load balancing via master/worker model that allowed for efficient performance on heterogeneous platforms.

Recently an interesting alternative to FMM and FSM has been proposed. Darbon and Osher [5] and Lee et al [4] utilize the Hopf-Lax formulation of the solution to the Hamilton-Jacobi form of the eikonal equation. Notably, the signed distance at an arbitrary point is obtained without the need of distance information from neighboring points. This allows for the solution at any given point in any order and prevents the need for communication across cores which greatly simplifies parallel implementation. Furthermore, this inherently allows for updates done only in a narrow band near the zero-isocontour. FSM must solve over the entire domain, and while FMM can be done in a narrow band, FMM methods are generally more difficult to implement in parallel. These aspects make the Hopf-Lax approaches in [4, 5] very compelling for parallel architectures. In this paper, we extend the work of Lee et al to handle functions defined via interpolation over a regular grid. Lee et al demonstrated compelling results with abstractly defined functions. However, treatment of grid-based functions is essential for practical application in level set methods. We demonstrate the effectiveness of our approach with Graphics Processing Unit (GPU) parallel implementation.

2. Method

Following Lee et al [4] we use the Hamilton-Jacobi formulation of the eikonal equation (1)

$$\begin{aligned} &\frac{\partial}{\partial t}\tilde{\phi}(\mathbf{x},t) + \|\nabla\tilde{\phi}(\mathbf{x},t)\|_2 = 0 \\ &\tilde{\phi}(\mathbf{x},0) = \phi^0(\mathbf{x}) \end{aligned} \tag{2}$$

for $\mathbf{x} \in \mathbb{R}^n, t > 0$. We assume that ϕ^0 is constructed such that

$$\begin{cases} \phi^0(\mathbf{x}) < 0 & \mathbf{x} \in \Omega \backslash \partial\Omega \\ \phi^0(\mathbf{x}) > 0 & \mathbf{x} \in (\mathbb{R}^n \backslash \Omega) \\ \phi^0(\mathbf{x}) = 0 & \mathbf{x} \in \partial\Omega \end{cases}$$

for some set $\Omega \subset \mathbb{R}^n$. As in Lee et al. [4] we assume that the set Ω is closed and non empty. Isocontours of the time dependent solution $\tilde{\phi}$ progress from the boundary $\partial\Omega$ in its normal direction at a rate of 1. To know the distance to the boundary, we simply need to know at which time $\hat{t}$ the zero-isocontour of $\tilde{\phi}$ has progressed to the point $\mathbf{x}$. In other words, the signed distance ($\phi(\mathbf{x})$) from the point $\mathbf{x}$ to the boundary $\partial\Omega$ is given by the time $\hat{t}$ with $\tilde{\phi}(\mathbf{x},\hat{t}) = 0$: $\phi(\mathbf{x}) = \hat{t}$. Note that we only consider the case here of positive ϕ since the case of negative ϕ is trivially analogous.

As in Lee et al [4], we treat the problem as root finding and use the secant method. However, unlike Lee et al we are specifically interested in redistancing grid based functions. Thus we assume that the initial function is defined in terms of its interpolated values from grid nodes as $\phi^0(\mathbf{x}) = \sum_{\mathbf{i}} \phi^0_{\mathbf{i}} N_{\mathbf{i}}(\mathbf{x})$ where the function $N_{\mathbf{i}}$ is the bilinear interpolation kernel associated with grid node $\mathbf{x_i}$ and $\phi^0_{\mathbf{i}} = \phi^0(\mathbf{x_i})$. Also, when we solve for the redistanced values, we do so only at grid nodes (i.e. we solve for $\phi_{\mathbf{i}} = \phi(\mathbf{x_i}) = \hat{t}$). Thus the secant method only requires the evaluation of the function $\tilde{\phi}(\mathbf{x_i}, t^k)$ for iterative approximation $t^k \to \hat{t}$. We next discuss the practical implementation of the secant method and evaluation of $\tilde{\phi}(\mathbf{x_i}, t^k)$ for grid based data.

2.1. Secant method for roots of $\tilde{\phi}(\mathbf{x_i}, \hat{t}) = 0$

In order to use the secant method to solve for the root in this context we use the iterative update

$$t^{k+1} = t^k - \tilde{\phi}(\mathbf{x_i}, t^k) \frac{t^k - t^{k-1}}{\tilde{\phi}(\mathbf{x_i}, t^k) - \tilde{\phi}(\mathbf{x_i}, t^{k-1})}. \tag{3}$$

The initial guess t^0 can either be set from neighboring nodes that have been recently updated, or generally from a priori estimates of the distance (see Section 2.3). However, when no such information is possible or when it would negatively effect parallel performance we use $t^0 = 0$. We set $t^1 = t^0 + \epsilon$ where ϵ is proportionate to the grid cell size.

The main concern with using the secant method in this context is that while $\tilde{\phi}(\mathbf{x_i}, t)$ is monotonically decreasing in t, it is not strictly monotone. This means that there can be times t where $\frac{d}{dt}\tilde{\phi}(\mathbf{x_i}, t) = 0$. For example, if the minimum of $\phi^0(\mathbf{x_i})$ over the ball centered at $\mathbf{x_i}$ of radius t is in the interior of the ball (at a point of distance s from $\mathbf{x_i}$), then $\frac{d}{dt}\tilde{\phi}(\mathbf{x_i}, r) = 0$ for $s \leq r \leq t$ (see Section 2.2). The secant method is not well defined if we have iterates with equal function values. To compensate for this, if secant would divide by zero, and we have not already converged, we simply increase or decrease $t^{k+1} = t^k \pm \Delta t_{\max}$ in the correct direction. The correct direction is trivial to find, because if $\tilde{\phi}(\mathbf{x_i}, t^k) > 0$ then we need to increase t^k. Otherwise we need to decrease t^k. In practice, we use $\Delta t_{\max}$= $5\Delta x$ where Δx is the grid cell size

Another issue is that errors in the approximation of $\tilde{\phi}(\mathbf{x_i}, t^k)$ can lead to more secant iterations. This can be reduced by solving for $\tilde{\phi}(\mathbf{x_i}, t^k)$ to a higher tolerance. However, requiring more iterations to approximate $\tilde{\phi}(\mathbf{x_i}, t^k)$ more accurately can be more costly than just using more secant iterations with a less accurate (but more efficient) approximation to $\tilde{\phi}(\mathbf{x_i}, t^k)$. We discuss the process and cost of solving for $\tilde{\phi}(\mathbf{x_i}, t^k)$ in Section 2.2. Our modified secant algorithm is summarized in Algorithm 2.1.

Algorithm 1 Modified Secant Method

while $|\tilde{\phi}(\mathbf{x_i}, t^{k+1})| > \epsilon$ **do**
  $\Delta t = -\tilde{\phi}(\mathbf{x_i}, t^k) \frac{t^k - t^{k-1}}{\tilde{\phi}(\mathbf{x_i}, t^k) - \tilde{\phi}(\mathbf{x_i}, t^{k-1})}$
  if $|\Delta t| > tol$ **then**
    if $\tilde{\phi}(\mathbf{x_i}, t^k) > 0$ **then**
      $\Delta t = \Delta t_{\max}$
    else
      $\Delta t = -\Delta t_{\max}$
    end if
  end if
  $t^{k+1} = t^k + \Delta t$
end while

2.2. Hopf-Lax Formula for $\tilde{\phi}(\mathbf{x_i}, t^k)$

As in Lee et al [4] we obtain the solution of Equation 2 with the Hopf-Lax formula

$$\tilde{\phi}(\mathbf{x_i}, t^k) = \mathtt{min}_{\mathbf{y} \in \mathbb{R}^n} \left\{ \phi^0(\mathbf{y}) + t^k H^* \left(\frac{\mathbf{x_i} - \mathbf{y}}{t^k} \right) \right\}$$

where H^* is the Fenchel-Legendre Transform of $H = \| \cdot \|_2$

$$H^*(\mathbf{x}) = \begin{cases} 0 & \|\mathbf{x}\|_2 \leq 1 \\ \infty & otherwise \end{cases}$$

or equivalently

$$\tilde{\phi}(\mathbf{x_i}, t^k) = \min_{\mathbf{y} \in B(\mathbf{x_i}, t^k)} \phi^0(\mathbf{y}) \tag{4}$$

where $B(\mathbf{x_i}, t^k)$ is the ball of radius t^k around grid node $\mathbf{x_i}$. Thus the problem of evaluating $\tilde{\phi}(\mathbf{x_i}, t^k)$ amounts to finding the minimum of the initial ϕ^0 over a ball. While Lee et al [4] use Split Bregman iteration to solve this, we instead simply use projected gradient descent. We used a few hundred projected gradient iterations in practice since this was faster than Split Bregman in parallel implementations due to its relative simplicity. Using $\mathbf{y}_k^0$ as an initial guess for

the argmin of ϕ^0 over the ball $B(\mathbf{x_i}, t^k)$, we iteratively update the approximation from

$$\tilde{\mathbf{y}}_k^{j+1} = \mathbf{y}_k^j - \gamma \nabla \phi^0(\mathbf{y}_k^j) \tag{5}$$

$$\mathbf{y}_k^{j+1} = \texttt{PROJ}_{B(\mathbf{x_i}, t^k)}(\tilde{\mathbf{y}}_k^{j+1}) \tag{6}$$

where

$$\texttt{PROJ}_{B(\mathbf{x_i}, t^k)}(\mathbf{y}) = \begin{cases} \mathbf{y} & \|\mathbf{x_i} - \mathbf{y}\|_2 \leq t^k \\ \mathbf{x_i} - t^k \frac{\mathbf{x_i} - \mathbf{y}}{\|\mathbf{x_i} - \mathbf{y}\|_2} & otherwise \end{cases}$$

In practice, we set the step size γ equal to the grid spacing Δx. Note that the gradients $\nabla \phi^0(\mathbf{y}_k^j)$ are computed using the bilinear interpolation kernels $N_{\mathbf{l}}(\mathbf{x})$ as $\nabla \phi^0(\mathbf{y}_k^j) = \sum_{\mathbf{l}} \phi^0_{\mathbf{l}} \nabla N_{\mathbf{l}}(\mathbf{y}_k^j)$. We emphasize that for efficiency the sum over $\mathbf{l}$ can be taken over only the four grid nodes surrounding the cell that the argument $\mathbf{y}_k^j$ is in. We further note that the index for the cell containing the argument $\mathbf{y}_k^j$ can be found in constant time using floor($\frac{y_{\alpha k}^j}{\Delta x}$) where $y_{\alpha k}^j$ are the components of $\mathbf{y}_k^j$. In general, ϕ^0 is a non-convex function defined from the grid interpolation and projected gradient descent will only converge to a local minimum. We illustrate this in Figure 1. Failure to converge to a global minimum can lead to large errors in the approximation of $\tilde{\phi}(\mathbf{x_i}, t^k)$. While it is impractical to ensure we achieved a global minimizer, it is possible to find multiple local minimizers increasing the probability we find a global minimizer. We solve (4) multiple times with different initial guesses y_k^0 and then take the minimum over these solutions to come up with a final answer that is likely to be close to a global minimzer. We found in practice, on the order of one guess per grid cell in the ball $B(\mathbf{x_i}, t^k)$ is sufficient to find a global minimizer. For problems without many local extrema the number of initial guesses can be reduced. In general when finding $\tilde{\phi}(\mathbf{x_i}, t^k)$ we use as initial guesses $\texttt{PROJ}_{B(\mathbf{x_i}, t^k)}(\mathbf{y}_{k-1})$ where

$$\mathbf{y}_{k-1} = \underset{\mathbf{y} \in B(\mathbf{x_i}, t^{k-1})}{\operatorname{argmin}} \phi^0(\mathbf{y})$$

is the argmin of ϕ^0 used in the previous secant iteration as well as a small number of random points in $B(\mathbf{x_i}, t^k)$. We use this strategy because $\texttt{PROJ}_{B(\mathbf{x_i}, t^k)}(\mathbf{y}_{k-1})$

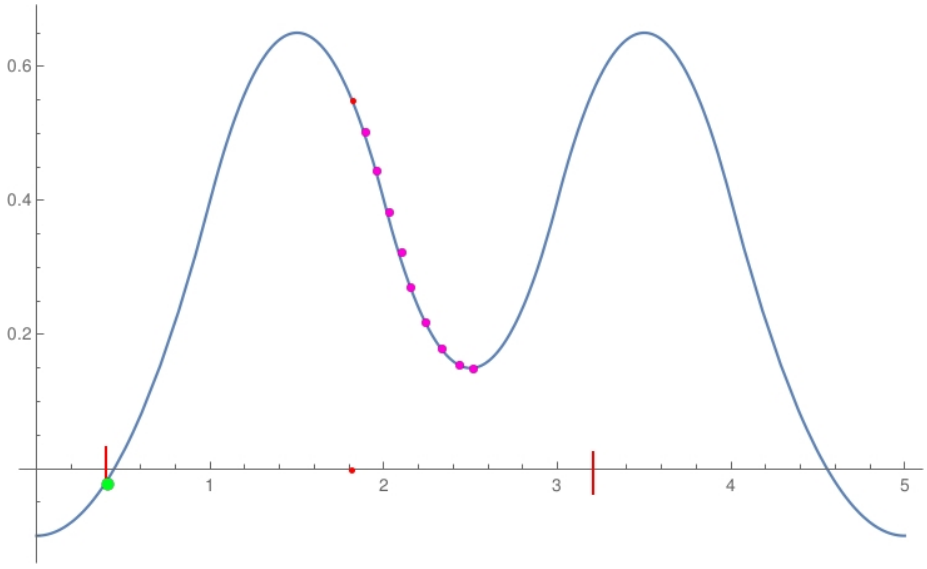

Figure 1: The two vertical lines are the boundary of minimization. Grid node $x_i = 1.8$ is in the middle of the region, and also the starting guess for projected gradient descent. The sequence of points leading off to the right represent the subsequent steps of gradient descent. These points converge to the incorrect argmin $x = 2.5$. The correct solution is at $x = 0.4$. In order to converge to this point, the initial guess would have to be less than 1.25

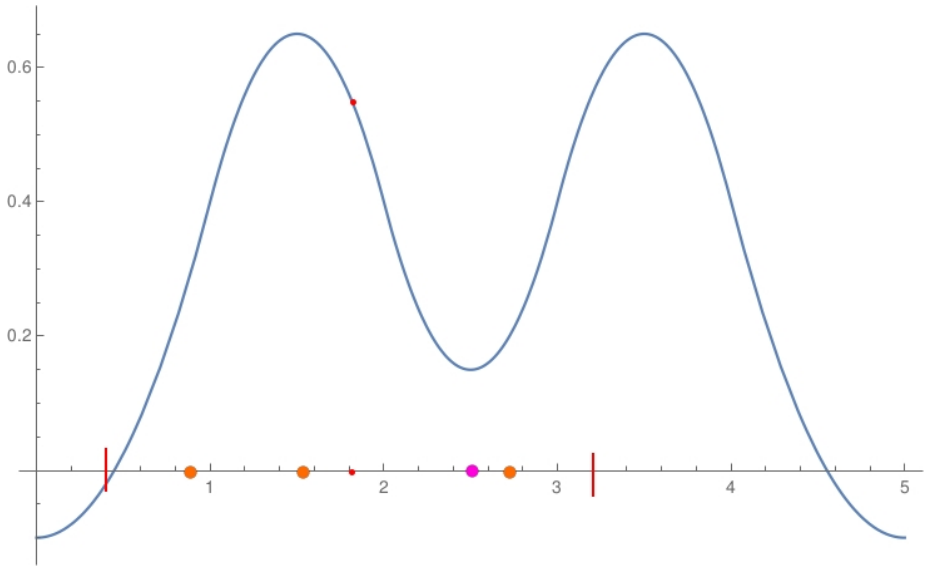

Figure 2: The figure illustrates representative random initial guesses used in solving for $\tilde{\phi}(\mathbf{x}_i, t^k)$. In addition we use an initial guess equal to the minimizer computed in the previous secant iteration shown in magenta.

tends to be a good guess for the global minimum. In general, it is very likely that at the next step, the minimum will either be the same point, or along the boundary. Therefore, we prioritize random initial guesses near the boundary of the ball. In fact, for $t^{k-1} < t^k$ we know that the argmin will be at a distance s from $\mathbf{x_i}$ with $t^{k-1} \leq s \leq t^k$ so in theory we should only sample in the annulus. However, in practice we do not have full confidence in the argmin attained at iteration $k-1$ since our procedure is iterative. Allowing for initial guesses at some locations closer to $\mathbf{x_i}$ than t^{k-1} admits the possibility of finding a more accurate argmin. Thus, we found that skewing the sampling density to be higher towards the boundary of the ball struck a good balance between efficiency and accuracy. We illustrate this process in Figure 4.

Failure to find the global minimum over the ball can cause unpredictable behavior in the secant iteration for $\hat{t}$. This includes false positives where a t^k is incorrectly interpreted as a root. However, continuing to run secant to a fixed large number of iterations usually corrects for this. In general, there

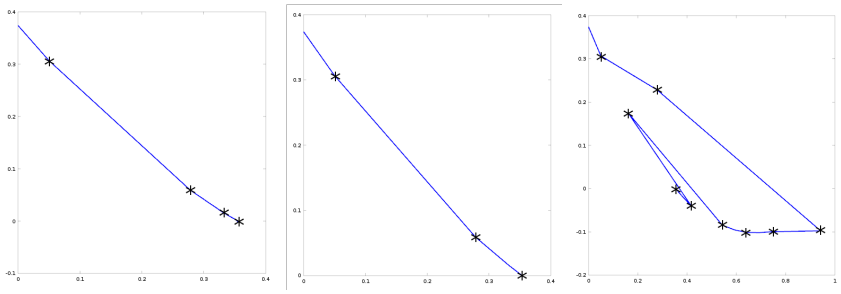

Figure 3: The plots above are the points $(\tilde{\phi}(\mathbf{x_i}, t^k), t^k)$ found when running our algorithm with different choices of random guess and gradient descent iterations on circle initial data. The left most plot was run with 100 random guesses, and 1 gradient descent iteration. The middle plot was run with 1 random guess, and 100 gradient descent iterations. The right plot was run with 1 random guess and 5 gradient descent iterations. Note that in all cases, the correct root was found.

is a tradeoff between the number of initial guesses and iterations of projected gradient and the number of secant iterations. We illustrate this in Figure 3 which shows the path to convergence for a few choices of iteration parameters. When $\tilde{\phi}(\mathbf{x_i}, t^k)$ is solved with high accuracy, the secant iteration converges with minimal overshoot in 7 iterations. When $\tilde{\phi}(\mathbf{x_i}, t^k)$ is not solved to high accuracy, secant overshoots by a large margin, and takes 16 iterations to converge, but notably still converges. However because each iteration is cheaper, the total runtime is lower to reach the same convergence for $\hat{t}$. In practice we found that a few hundred projected gradient iterations combined with our initial guess sampling strategy struck a good balance between accuracy and efficiency.

2.3. Computing in a narrow band

In many applications, only data close to the interface is needed. Since each grid node can be solved for independently, the Hopf-Lax approach naturally lends itself to narrow banding strategies for these applications. We provide a narrow banding strategy based on a coarse initial grid computation followed by a fine grid update in the narrow band. We first redistance the function on the coarse grid, then we interpolate values from the coarse grid to the fine grid. We then only recompute values on the fine grid that are smaller than a threshold

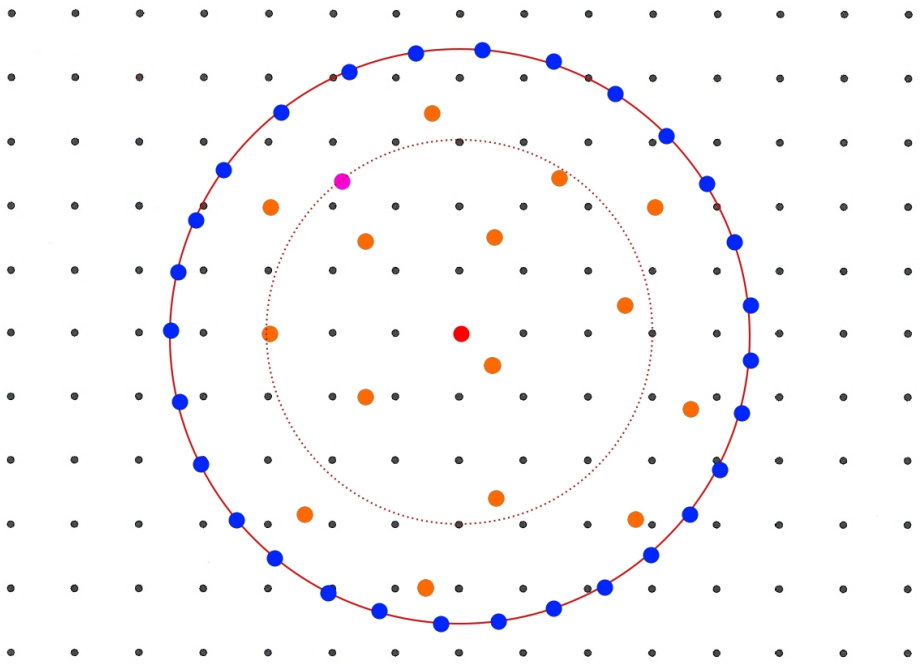

Figure 4: In this image, the red dot in the center is $\mathbf{x_i}$, the solid red line represents the ball of radius t_k and the dotted line represents the ball of radius t_{k-1}. The magenta point was the approximate argmin $\mathbf{y}_{k-1}$ of ϕ^0 over the ball of radius t_{k-1}. Since it is unlikely for the minimizer to be inside of t_{k-1} we use coarse (random) grid initial guesses in the interior. However, since it is possible that expanding t will move the minimizer to a different location we take a large number of initial guesses along the boundary of t_k

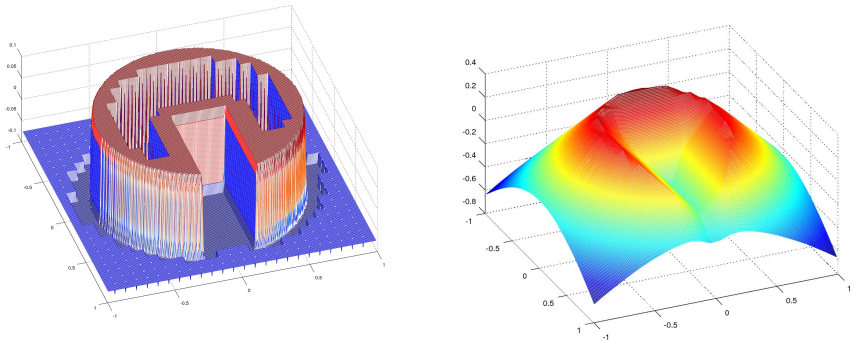

Figure 5: A coarse grid 8X smaller than the fine grid was solved for initially. Using those values the fine grid was only solved on cells where the distance to the boundary could be less than 0.1 represented as the solid areas of the left image. In the right image those coarse areas are defined from bilinear interpolation. This coarse/banding approach provided approximately a 2.5 times increase in performance.

value and we use the value interpolated from the coarse nodes as an initial guess t_0 for the computation on the fine grid. As an example see Figure 5.

2.4. Computing geometric quantities

The Hopf-Lax formulation naturally allows us to compute normals ($\mathbf{n} = \nabla\phi$) and curvatures ($\nabla \cdot \mathbf{n}$) at the grid nodes. As pointed out in Lee et al [4], as the argmin $\mathbf{y}^k$ from Equation 5 is iterated to convergence, it approaches the closest to point to the grid node $\mathbf{x_i}$ on the zero isocontour of ϕ^0. Therefore, recalling that when t^k has converged (within a tolerance) to the root $\hat{t}$ of $\tilde{\phi}(\mathbf{x}, \hat{t}) = 0$, t^k is approximately the distance to the zero isocontour, we can compute the unit normal at the grid node from

$$\mathbf{n}(\mathbf{x_i}) = \frac{\mathbf{x_i} - \mathbf{y}^k}{t^k}.$$

Notably, this approximation is very close to the exact signed distance function with zero isocontour determined by the bilinearly interpolated ϕ^0. It is as accurate as the argmin $\mathbf{y}^k$ so it essentially only depends on the accuracy of the secant method. We get this very accurate geometric information for free. Moreover, the curvature ($\nabla \cdot \mathbf{n}$) can be computed accurately by taking a numerical

divergence of $\mathbf{n}(\mathbf{x_i})$ that would have accuracy equal to the truncation error in the stencil (since no grid-based errors are accumulated in the computation of $\mathbf{n}(\mathbf{x_i})$).

3. Results

All of the following results were run on an Intel 6700k processor with an Nvidia GTX 1080. The domain for each problem was set to be $[0,1]X[0,1]$ and was run on a 512X512 grid. To ensure efficient performance on the GPU, both projected gradient descent and the secant method were run for a fixed number of iterations rather than to a specific tolerance. All timings listed in this section are averages over 5 seperate runs, with the exception of the Vortex problem which is already an average. This is due to the fact that we noticed in practice variations of up to 10% in the individual runtimes of a given problem

Problem	*num_secant*	num_rand	num_proj	Timing(ms)
Circle	10	5	100	47.567
Two Points	10	5	100	45.199
Vortex(Per Frame)	10	5	200	73.248
Square	10	4	200	67.582
Sine	10	5	200	71.429

Figure 6 shows initial data ϕ^0 with a zero-isocontour given by a circle with radius .25. Figure 7 shows a more complicated test. The zero-isocontour is a square bounded between $[.25, .75]$ in x and y. The corners present rarefaction fans, and the inside needs to handle sharp corners along the diagonals. Because of these difficulties (especially the sharp gradient in our original interpolated ϕ^0), more work is needed in resolving the projected gradient descent step to ensure quick convergence of secant method. The zero-isocontour shown in Figure 8 is the union of two overlapping circles. Like in Figure 6 the gradient is fairly smooth and thus requires less computation to successfully converge in gradient descent. In Figure 9 we demonstrate our algorithm with a large number of local minima. This problem requires more projected gradient iterations than

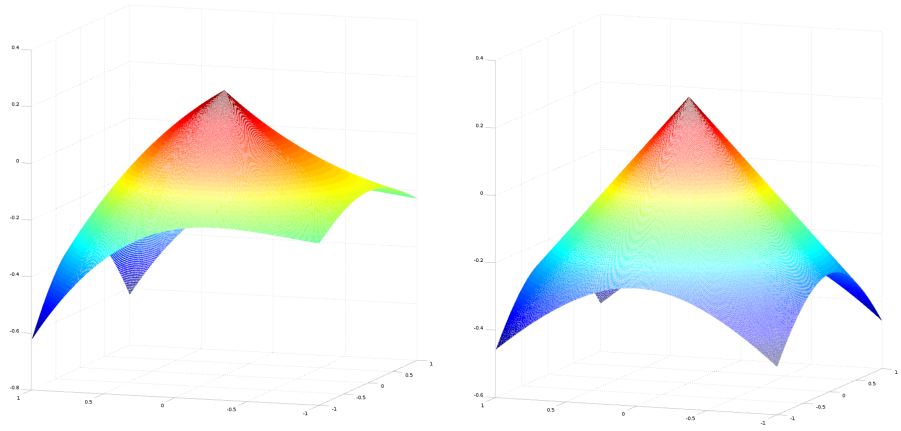

Figure 6: Scaled circle: the initial data is $\phi^0 = exp(x) * (.125 - (.5 - x)^2 + (.5 - y)^2)$. The zero level set is a circle of radius .25 centered around (.5,.5)

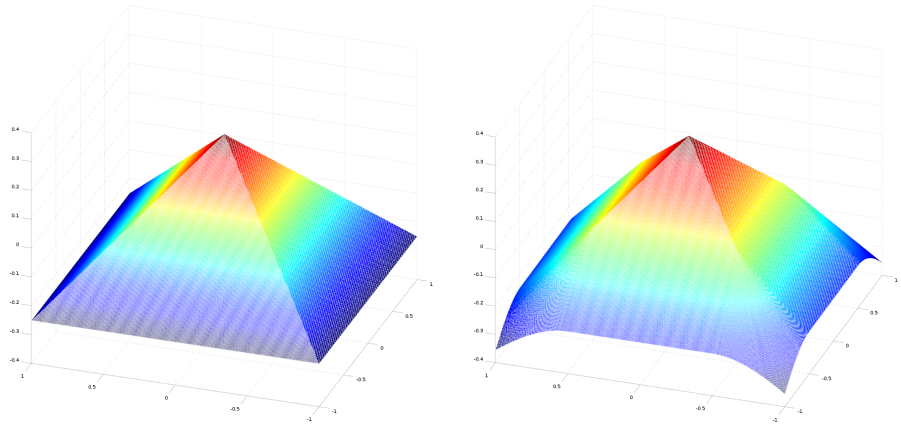

Figure 7: Square: the initial data is $\phi^0 = \texttt{min}(.25 - |x - .5|, .25 - |y - .5|)$. The zero level is a square with side length .5 centered around (.5,.5)

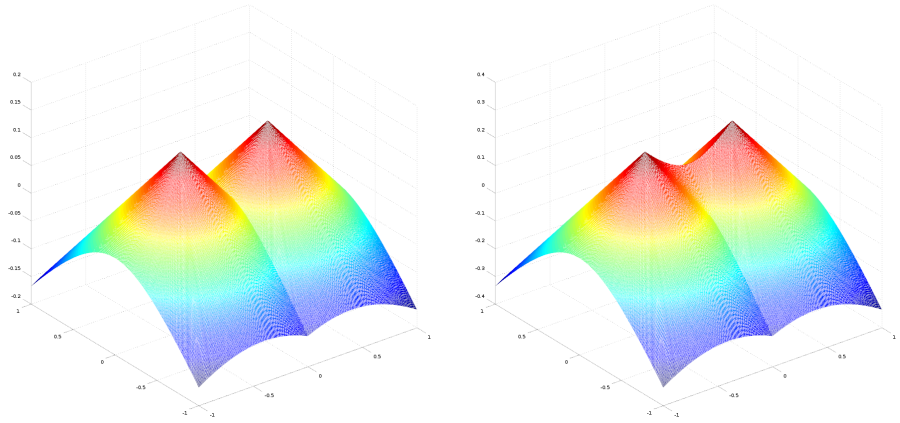

Figure 8: Union of circles: the initial data is $\phi^0 = \texttt{max}((.25 - \|(.3,.5) - (x,y)\|_2), (.25 - \|(.7,.5) - (x,y)\|_2))$. The zero level set is a union of two circles both with radius .25, one centered at (.3,.5) and the other centered at (.7,.5)

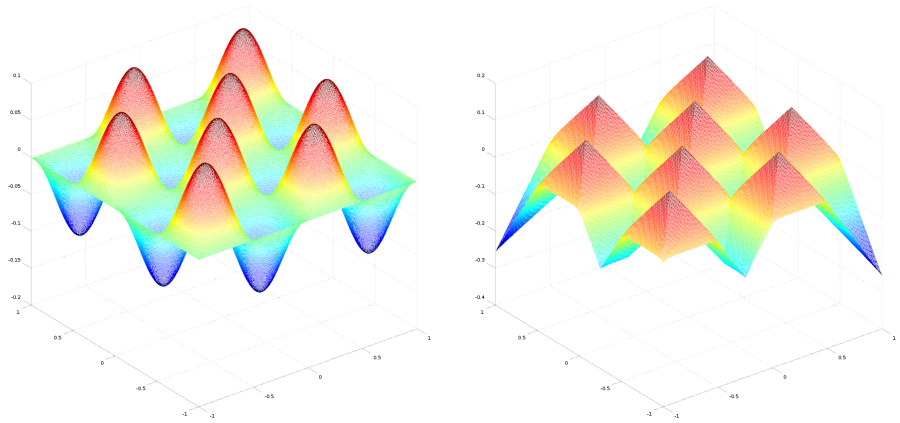

Figure 9: Many local minima: the objective function is $\phi^0 = \texttt{sin}(4 * \pi * x) * \texttt{sin}(4 * \pi * y) - .01$. The zero level set is a group of rounded squares that almost touch

Size	Timing(ms)	average L2 error
32X32	3.303	$6.67 * 10^{-05}$
64X64	3.392	$1.5917 * 10^{-05}$
128X128	4.477	$3.8723 * 10^{-06}$
256X256	17.8400	$9.7391 * 10^{-07}$
512X512	67.533	$2.4624 * 10^{-07}$
1024X1024	274.216	$6.4343 * 10^{-08}$
2048X2048	1185.87	$2.3843 * 10^{-08}$

Table 1: Timing and error at different grid resolutions using a square as our zero level set. The average L2 error is calculated by calculating the L2 distance between the computed answer and the analytic solution

the simpler examples.

Figure 11 shows our method being used in a level set advection scheme using a simple vortex test. Like previous problems it was run on a 512X512 grid. For this problem the average time per frame for redistancing was 73.248ms.

3.1. Scaling

The results in table 1 was run with the square given in Figure 7 with the same parameters. The poor scaling at the low resolutions is due to not using all of the threads possible on the GPU.

For Table 2 we ran our algorithm on the initial data in Figure 6 with a 1024X1024 grid. The problem was broken up into sub domains and each domain was run separately on the GPU. The performance results are shown in Figure 10. The scaling is nearly optimal, but breaks down at high number of GPUs when we include the time it takes to transfer the data to the GPU. The transfer time takes approximatly 1.2 ms. If we ignore the time it takes to transfer data to the GPU we get a result that is close to being perfectly parallel. We also show this in Figure 10.

# GPUs	Total (w/)	per GPU (w/)	Total (w/o)	per GPU (w/o)
1	125.10	125.10	124.05	124.05
2	126.25	63.13	124.76	62.38
4	130.17	32.54	124.92	31.23
8	139.26	17.41	131.10	16.39
16	149.26	9.33	133.24	8.33
32	167.97	5.25	133.07	4.1585
64	206.07	3.22	134.85	2.11

Table 2: Timing in ms showing scaling in number of GPUs. The first two columns show the time it takes to run our problem when we include the timing cost of transferring the grid data to the GPU (approximatly 1.2ms), while the last two columns show the scaling without the cost of transferring data

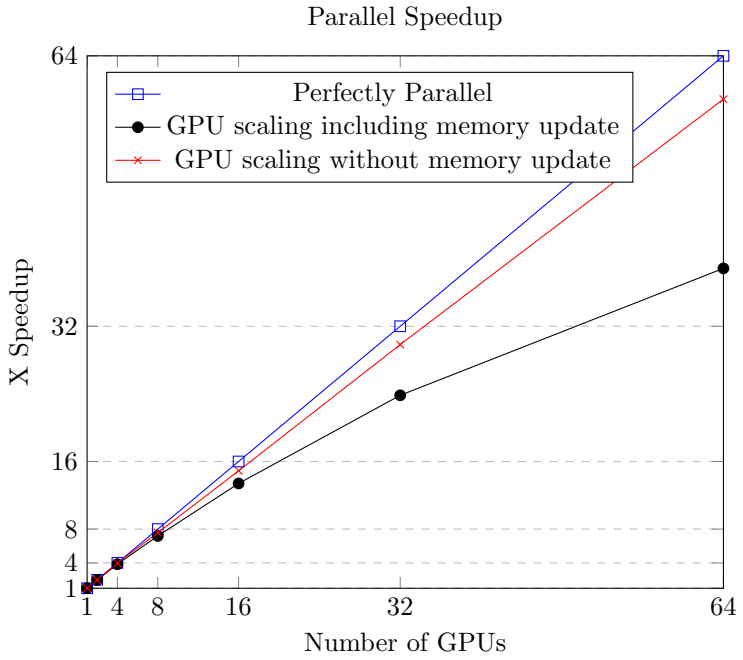

Figure 10: Parallel speed up is plotted both with and without including the cost of updating memory on the GPU. With 64 GPUs the memory update can take up to 33% of the runtime. However without the memory update (I.e. if the data is already on the GPU) the method scales simply.

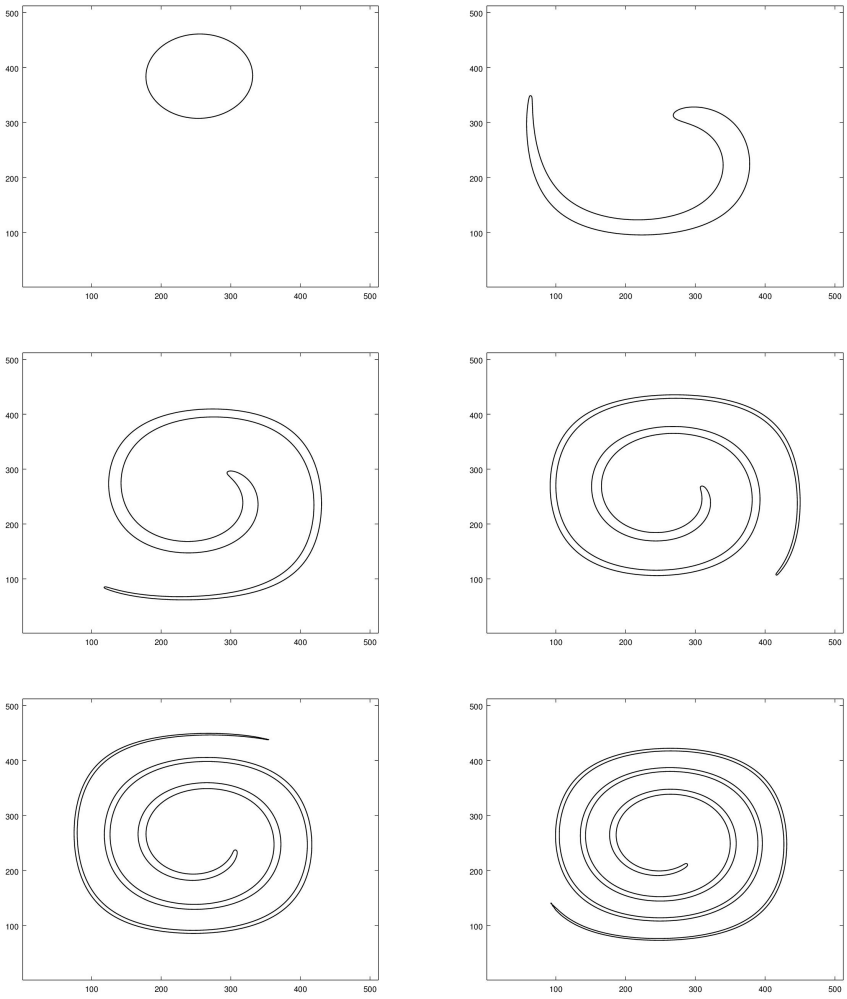

Figure 11: Practical application: vortex advection test at t = 0,1,2,3,4,5

Acknowledgements

The authors were partially supported by ONR grants N000141410683, N000141210838, N000141712162, N000141110719, N000141210834, DOE grant DE-SC00183838, DOD grant W81XWH-15-1-0147, NSF grant CCF-1422795 and an Intel STC-Visual Computing Grant (20112360) as well as a gift from Disney Research.

[1] J. N. Tsitsiklis, Efficient algorithms for globally optimal trajectories, IEEE Trans Auto Cont 40 (9) (1995) 1528–1538.

[2] J. A. Sethian, A fast marching level set method for monotonically advancing fronts, Proc Nat Acad Sci 93 (4) (1996) 1591–1595.

[3] H. Zhao, A fast sweeping method for eikonal equations, Math Comp 74 (250) (2005) 603–627.

[4] B. Lee, J. Darbon, S. Osher, M. Kang, Revisiting the redistancing problem using the hopf–lax formula, J Comp Phys 330 (2017) 268–281.

[5] J. Darbon, S. Osher, Algorithms for overcoming the curse of dimensionality for certain hamilton–jacobi equations arising in control theory and elsewhere, Res Math Sci 3 (1) (2016) 19.

[6] S. Osher, J. A. Sethian, Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations, J Comp Phys 79 (1) (1988) 12–49.

[7] S. Osher, R. P. Fedkiw, Level set methods and dynamic implicit surfaces, Applied mathematical science, Springer, New York, N.Y., 2003.
URL `http://opac.inria.fr/record=b1099358`

[8] M. Sussman, P. Smereka, S. Osher, A level set approach for computing solutions to incompressible two-phase flow, J Comp Phys 114 (1) (1994) 146–159.

[9] G. Russo, P. Smereka, A remark on computing distance functions, J Comp Phys 163 (1) (2000) 51–67.

[10] M. G. Crandall, P.-L. Lions, Two approximations of solutions of hamilton-jacobi equations, Math Comp 43 (167) (1984) 1–19.

[11] E. W. Dijkstra, A note on two problems in connexion with graphs, Numer Math 1 (1) (1959) 269–271.

[12] H. Zhao, Parallel implementations of the fast sweeping method, J Comp Math (2007) 421–429.

[13] M. Detrixhe, F. Gibou, C. Min, A parallel fast sweeping method for the eikonal equation, J Comp Phys 237 (2013) 46–55.

[14] M. Detrixhe, F. Gibou, Hybrid massively parallel fast sweeping method for static hamilton–jacobi equations, J Comp Phys 322 (2016) 199–223.

[15] M. Herrmann, A domain decomposition parallelization of the fast marching method, Tech. rep., DTIC Document (2003).

[16] J. Yang, F. Stern, A highly scalable massively parallel fast marching method for the eikonal equation, J Comp Phys 332 (2017) 333–362.

[17] W.-k. Jeong, R. Whitaker, et al., A fast eikonal equation solver for parallel systems, in: SIAM Conf Comp Sci Eng, Citeseer, 2007.

[18] M. Breuß, E. Cristiani, P. Gwosdek, O. Vogel, An adaptive domain-decomposition technique for parallelization of the fast marching method, App Math Comp 218 (1) (2011) 32–44.

[19] F. Dang, N. Emad, Fast iterative method in solving eikonal equations: a multi-level parallel approach, Proc Comp Sci 29 (2014) 1859–1869.