Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 947 – 955

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

A new parallel 3D front propagation algorithm for fast simulation of geological folds

Tor Gillberg[a,c,*], Mohammed Sourouri[a,b], Xing Cai[a,b],

[a]*Computational Geoscience, CBC, Simula Research Laboratory, P.O. Box 134, 1325 Lysaker, Norway*
[b]*Department of Informatics, University of Oslo, P.O. Box 1080 Blindern, 0316 Oslo, Norway*
[c]*Kalkulo AS, P.O. Box 134, 1325 Lysaker, Norway*

Abstract

We present a novel method for 3D anisotropic front propagation and apply it to the simulation of geological folding. The new iterative algorithm has a simple structure and abundant parallelism, and is easily adapted to multithreaded architectures using OpenMP. Moreover, we have used the automated C-to-CUDA source code translator, Mint, to achieve greatly enhanced computing speed on GPUs. Both OpenMP and CUDA implementations have been tested and benchmarked on several examples of 3D geological folding.

Keywords: Static Hamilton-Jacobi equations, front propagation, automated C-to-CUDA code translation

1. Introduction

The arrival time of a propagating front is often described by non-linear static Hamilton-Jacobi equations. Advanced numerical algorithms are needed to efficiently compute solutions to those equations. It is therefore a challenge to implement fast solvers, especially for large 3D simulations. Solution algorithms are often divided into two groups, Front Tracking methods and Sweeping methods. Front Tracking methods [1, 2, 3] update node values in a strictly increasing order, and thus mimic a front expanding from the initial object Γ_0. These algorithms are sequential by construction, since the front passes only one node at a time. Front tracking methods for anisotropic propagation are known as Ordered Upwind Methods. These methods are complicated, and some must be simplified for implementation. Moreover, they often assume prior knowledge of the degree of anisotropy in the problem, see for instance [4, 5, 6]. Sweeping methods [7] compute the solution from a distance perspective by iterating over directions. This makes them faster than Front Tracking methods on simple problems [8]. In the cases with complex geometries or velocities that force the characteristics to be curved, the Sweeping methods are slow because many iterations are needed before convergence [9]. Since the iteration order is predetermined, Sweeping methods [10] are readily parallelized. However, the parallelism of the traditional Sweeping algorithms is limited, and the parallel speedup is therefore modest [11]. By an alternative formulation of the stencil and iteration order, abundant parallelism can be obtained as shown with

*Corresponding author
Email addresses: torgi@simula.no (Tor Gillberg), mohamso@ifi.uio.no (Mohammed Sourouri), xingca@simula.no (Xing Cai)

doi:10.1016/j.procs.2012.04.101

the Parallel Marching Method (PMM) [12]. Early iterative algorithms trace the front in specific patterns such as expanding boxes [13], or by updating all nodes until convergence [14]. These can be made parallel as described in [15]. There are also a few algorithms that use concepts from both Front Tracking and Sweeping methods [16, 17, 18, 9].

To our knowledge, only two methods for front propagation have been successfully implemented on graphics processing units (GPUs), namely the Fast Iterative Method [17] and the PMM [12]. Of these two methods only the Fast Iterative Method is applicable in 3D, since the PMM was created for computing geodesic distances on surfaces. In this article, we present a new 3D algorithm with abundant parallelism, making the algorithm suitable for both multicore CPU and GPU architectures. Since we use the idea of an alternative stencil formulation from the PMM, we refer to our new algorithm as the 3D PMM. The 3D PMM has a highly parallel structure as nodes on an entire surface (planar cut of the 3D volume) can be updated in parallel. Moreover, thanks to the automated C-to-CUDA translator Mint [19], we can maintain an annotated serial C code for our 3D PMM method, without having to do low-level CUDA programming by hand. In comparison, the more general Fast Iterative method has a more intricate algorithmic structure, making CUDA programming much more involved.

1.1. Simulation of Folds and other Applications

Over the past several years, Statoil and Kalkulo have developed a novel paradigm for highly interactive modelling of complicated geological scenarios and processes. This methodology describes present-day geology as the realization of a series of geological events and processes along a geological timeline [20, 21]. Many processes rely on relevant surfaces and their corresponding metric property fields or maps like distances, gradients etc. [22]. These distance maps are described by the viscosity solution to the static Hamilton-Jacobi equation:

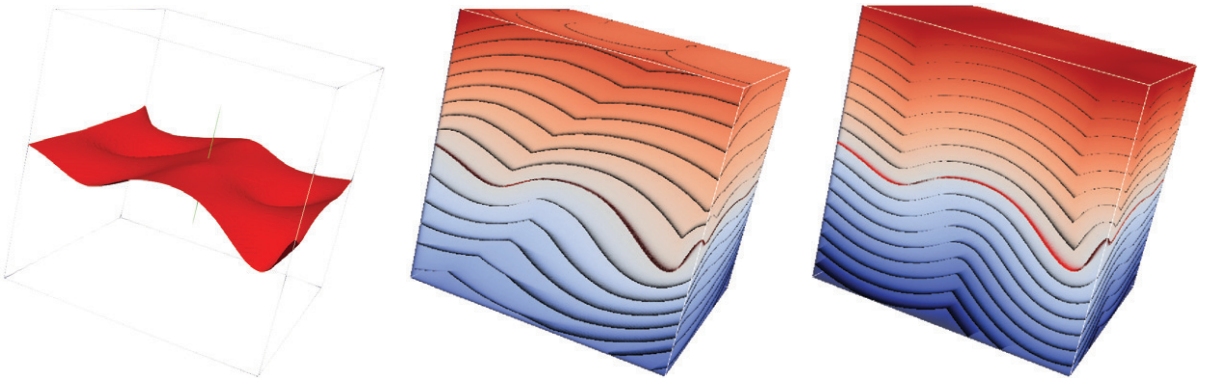
$$F\|\nabla T(\mathbf{x})\| + \psi\,(\mathbf{a}\cdot\nabla T(\mathbf{x})) = 1, \tag{1}$$

$$\text{given}\quad T = t_0 \text{ on } \Gamma_0. \tag{2}$$

Here, Γ_0 is an initial horizon, and $\mathbf{a}$ marks the axial direction of the fold. Other folded layers are implicitly given as iso-surfaces of T, that is, the position of a front propagating from Γ_0 at different times. This equation can replicate all traditional folding classes, as defined by [23], by altering the size and sign of F, ψ and direction of $\mathbf{a}$. For details and derivation of this system we refer to [22]. The same equation also describes the first arrival of a wave in a media in motion [24]. Figure 1(a) shows an example of an initial surface Γ_0. From this surface Figures 1(b) and (c) show two simulations of folds, created with different parameter choices. When $\psi \neq 0$, the front propagation is of the anisotropic type. In the special case where $\psi = 0$, (1) reduces to the isotropic eikonal equation, which is solved in many applications [1] and is isotropic in the sense that the velocity is independent of direction. When $F = 1$, the viscosity solution to the eikonal equation is the minimal Euclidean distance from Γ_0. The concept of characteristic curves or ray-paths is important in front propagations [22, 25]. These are curves along which a particle on the front is transported, and can also be interpreted as curves defining the shortest distance (fastest path) to Γ_0.

When modelling folds, the dip isogons, and thus the characteristic curves [22], are in general linear. For isotropic problems with linear characteristics Sweeping algorithms converge quickly [9], which motivates us to investigate related algorithms for the simulation of geologically folded structures. A powerful laptop is sufficient for an interactive geological modelling application in 2D, but the computational requirement is vastly higher in 3D. Other geological applications where the simulation of a propagating front is central include reservoir simulations [26] and simulation of seismic traveltimes [25, 27]. In seismic applications, front propagation solvers are used to simulate entire first arrival traveltime fields. By repeating the simulation from reflecting surfaces, multiple reflected traveltimes can be computed with front propagation solvers [25, 28]. Front propagation is also heavily used in medical imaging [11, 29, 30]. In these applications, the computational speed remains often a challenge. A faster solver would allow for more interactive applications, potentially leading to faster medical diagnoses and faster seismic processing. An interactive geological modelling application allows users to test many geological scenarios faster, leading to a better understanding of the inner earth. One method to achieve faster solvers is by making use of the powerful computational resources available in GPUs.

In this paper, we present a novel 3D front propagation algorithm, as well as a numerical stencil for solving (1). We also show how to parallelize the algorithm for both multicore CPU and GPU architectures. The parallelized codes are tested on several examples of geological folding, for which the parallelized codes scale well.



(a) Initial surface Γ_0 (b) Anisotropic propagation (c) Isotropic propagation

Figure 1: (a) An initially given surface Γ_0. (b) A folded 3D volume with $F = 1, \psi\mathbf{a} = \frac{1}{2}(-1, 1, 1)$, simulated from Γ_0. (c) A folded 3D volume from the same initial surface, but this time with $F = 1, \psi = 0$, resulting in isotropic front propagation and an Euclidean distance field.

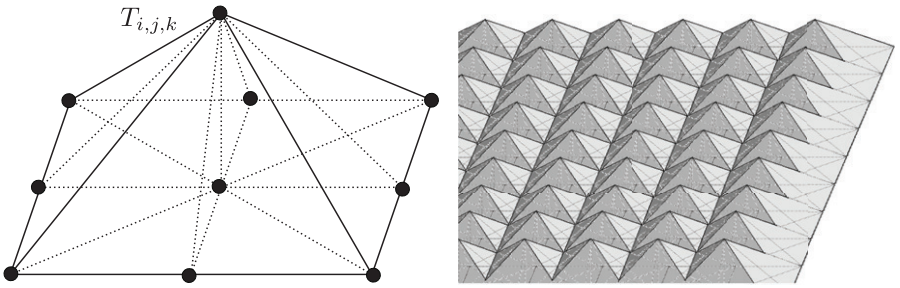
2. The 3D Parallel Marching Method

Consider a 3D box grid with nodal values $T_{i,j,k}$ where $(1, 1, 1) \leq (i, j, k) \leq (n_x, n_y, n_z)$, and with a spacing of (dx, dy, dz). In this paper we assume that values at the nodes closest to Γ_0 are given, and all the other nodes are initially set to an infinite value. (Efficient methods for initializing such values are outside the scope of this paper.) In every iteration, a smaller T value is a better approximation, since we solve for the minimal distance (the first time of arrival). It is of great importance that a new approximation is not too small, since such values are never corrected. A methodology for assuring such a discretization of (1) is presented in Appendix A. The PMM iterates through the grid in axial directions, and computes new distance values based on nodal values along the iteration direction. In the x-direction, the 3D volume is first iterated in the increasing order of the i index, and then in the decreasing order of the i index. The same sub-sweeps are also repeated in the y- and z-directions. We refer to such a full iteration as a *sweep*, which consists of 6 *sub-sweeps* of the 3D domain. Pseudocode for the sub-sweeps for the x-direction is given below.

for $i = 2, \ldots, n_x$ **do**
  for all $j = 1, \ldots, n_y$ **do**
    for all $k = 1, \ldots, n_z$ **do**
      Update $T_{i,j,k}$ using values $T_{i-1,j\pm a,k\pm b}, a \in \{0, 1\}, b \in \{0, 1\}$
    end for
  end for
end for
for $i = n_x - 1, \ldots, 1$ **do**
  for all $j = 1, \ldots, n_y$ **do**
    for all $k = 1, \ldots, n_z$ **do**
      Update $T_{i,j,k}$ using values $T_{i+1,j\pm a,k\pm b}, a \in \{0, 1\}, b \in \{0, 1\}$
    end for
  end for
end for

We remark that $T_{i,j,k}$ is computed using nine nodes in the previously updated plane. The form of the update stencils are illustrated in Figure 2(a), where the sub-sweep is in the direction of the pyramid top. Every approximation's update step includes a significant amount of computations, as shown in Appendix A.

Since there are no internal dependencies between nodes on the same update plane, all nodes in the plane can be computed simultaneously. Figure 2(b) shows a plane of stencil shapes that can be solved in parallel. In the pseudocode this corresponds to computing the two inner loops in each sub-sweep entirely in parallel. Because of the simplicity of this parallelism, the algorithm is easily parallelized using OpenMP. For the OpenMP parallelization, the parallel



(a) One stencil shape (b) Independent updates

Figure 2: (a) Nodes used when computing the generalized distance in the upward direction. (b) Illustration of the fine-grained parallel feature of the algorithm. All nodes in one plane can be computed simultaneously since they have no internal dependencies, making the algorithm suitable for parallel architectures.

region which encapsulates all the sweeps, is declared using **#pragma omp parallel**. Inside each of the six sub-sweeps, the two innermost nested loops are parallelized by adding **#pragma omp for**.

As shown in the next section, such a straightforward OpenMP parallelization achieves good speedup on multicore CPUs. Still, the OpenMP implementation is not sufficient for the application to be interactive for large grid sizes. This can be remedied by porting the algorithm to a GPU. To avoid manual GPU programming, we have made use of the automated C-to-CUDA source code translator Mint, freely available at https://sites.google.com/site/mintmodel/. Mint takes as input annotated serial C code and generates (optimized) CUDA code. The needed Mint pragmas are very similar to the OpenMP pragmas, except for two additional pragmas: **#pragma mint copy(T,toDevice,nx,ny,nz)** and **#pragma mint copy(T,fromDevice,nx,ny,nz)**, for transferring data between the host CPU and the device GPU. In Listings 1 and 2, we show two CUDA code segments that are automatically generated by Mint.

```
1  for (int SweepNbr = 0; SweepNbr < nbrSweeps; SweepNbr++) {
2
3    // x-direction: sweep from bottom to top
4    for (i = 1; i < 400; ++i) {
5      int num2blockDim_6_1527 = (400) % 16 == 0?(400) / 16 : (400) / 16 + 1;
6      int num1blockDim_6_1527 = (400) % 16 == 0?(400) / 16 : (400) / 16 + 1;
7      dim3 blockDim_6_1527(16,16,1);
8      dim3 gridDim_6_1527(num1blockDim_6_1527,num2blockDim_6_1527);
9
10     mint_6_1527<<<gridDim_6_1527,blockDim_6_1527>>>(DXYP,DXZP,DYZP,dev_1_T,i,tnew,st,xt,yt,txy,xnt,
           ynt,txm,tym,txnyn,F,ax,ay,az,dzz,dxx,dyy);
11     cudaThreadSynchronize();
12   }
13
14   // x-direction: sweep from top to bottom
15   // ...
16
17   // y-direction: sweep from bottom to top
18   // ...
19   // y-direction: sweep from top to bottom
20   // ...
21
22   // z-direction: sweep from bottom to top
23   // ...
24   // z-direction: sweep from top to bottom
25   // ...
26 }
```

Listing 1: The main computational body of the `Sweep` function after automated Mint translation from C to CUDA; The number of nodes in each spatial direction is 400.

```
1  __global__ void mint_6_1527(double DXYP,double DXZP,double DYZP,cudaPitchedPtr dev_1_T,int i,double
       tnew,double st,double xt,double yt,double txy,double xnt,double ynt,double txm,double tym,double
       txnyn,double F,double ax,double ay,double az,double dzz,double dxx,double dyy)
2  {
3    double *T = (double *)dev_1_T.ptr;
4    int _width = dev_1_T.pitch / sizeof(double );
5    int _slice = dev_1_T.ysize * _width;
6    int _p_j;
```

```
 7   int _p_k;
 8 {
 9     int _upperb_y = 400;
10     int _upperb_x = 400;
11     int _idx = threadIdx.x + 1;
12     int _gidx = _idx + blockDim.x * blockIdx.x;
13     int _idy = threadIdx.y + 1;
14     int _gidy = _idy + blockDim.y * 1 * blockIdx.y;
15 {
16       if (_gidy >= 1 && _gidy <= 400) {
17             if (_gidx >= 1 && _gidx <= 400) {
18         // the same computations as in the original C code
19             }
20       }
21 }
22 }
23 }
```

Listing 2: The CUDA kernel function `mint_6_1527` that is automatically generated by Mint; for the purpose of sweeping one yz-plane.

3. Results

In this section we present numerical results from an example of simulating a folded volume. From the same initially given surface as in Figure 1, we ran 8 sweeps on the three uniform grids with a total of $160^3, 320^3$ and 400^3 nodes. After 8 sweeps the solution has converged sufficiently. In these computations $\psi a = (-0.34, 0.4, 0.7)$, $F = 1.1$, and the domain has length 10 in x, y and z directions. We have measured the computational time for the OpenMP code using one node on the NERSC Cray XE6 "Hopper" supercomputer. Each node is equipped with two twelve-core AMD 'Magny-Cours' 2.1 GHz processors. The Mint-translated CUDA code for the same problem was executed using a Nvidia GeForce GTX 590 card. Table 1 shows elapsed times for the three grids on 1, 2, 4, 8, 16 and 24 CPU cores, as well as for the GPU. The time to transfer data to and from the GPU are included in the reported times. Both the CPU and GPU executables were compiled with the `-O3` flag, using `nvcc 4.0, V0.2.1221` and `gcc v4.3.4` respectively.

In Appendix A we present conditions that reduce the number of unnecessary computations in the update step. If all conditions are used, the number of branches increases. The update scheme already has many branches, as is indicated of profiling of the code. The profiling also indicate that the registers are under high pressure. Therefore, we have tried to formulate the update step to reduce unnecessary branching and register use. Several experiments was run with different update conditions, showing that the computational time is reduced the most when all conditions in Appendix A are used. This result holds for both the multicore and GPU versions of our code. Both the CPU and GPU codes was tested with both single and double precision. The difference between the single and double precision solutions is very small. Therefore, when modelling folds the gain in accuracy might not be worth the associated cost in computational time. Further investigation is needed before making any conclusion in this matter. In the geological modelling software, the derivative of the computed solution is used in post processes. This puts extra demand for high accuracy of the computed distance field. Thus, future research will be focused on extending the discretization to higher order schemes.

The code scales well on the multicore CPU, with near-linear speedup (1.9x) measured when conducting a strong scaling study up to 16 cores. Beyond 16 cores, the speedup drops to 1.3x. This drop in speedup is possibly due to the underlying NUMA (Non Uniform Memory Architecture) architecture on Hopper. With a more careful distribution of threads and data, we might be able to reduce the challenges NUMA imposes on the performance.

For the largest grid of 400^3 nodes, the GPU needs **25.28** seconds to perform 8 sweeps using single precision. As comparison, 24 CPU cores need **177.86** seconds to perform the 8 sweeps. When double precision is used on the GPU, the time usage is **57.42** seconds, more than **4** times faster than using 24 CPU cores (**240.05** seconds). It can also be seen in Table 1 that the speed advantage of GPU computations increases with the grid size.

At the moment of writing, a GeForce GTX 590 GPU costs around $500 USD, while one AMD 'Magny-Cours' 2.1 GHz costs more than $1000 USD. Depending on the grid size and precision, using a GPU will deliver 2-7x the performance, while costing[1] four times less than a 24-core CPU node. This makes the results even more impressive.

[1] Other system parts such as memory, motherboard etc., are not taken into account.

Table 1: Computational times for three grids with a total of N nodes using single (top table) and double precision. t_i is the CPU time for i cores. The speedup factors S_1 and S_{24} are calculated using the running time from 1 core and 24 cores (the highest number of CPU cores available). The speedup factor increases as N increases, possibly due increased computational complexity that amortises the data transfer cost from the CPU to the GPU. The data in Tabular (a) are single precision results while data in Tabular (b) are double precision results.

N	t_1	t_2	t_4	t_8	t_{16}	t_{24}	t_{GPU}	S_1	S_{24}
160^3	194.17	100.66	52.62	27.20	14.07	10.22	2.43	79.9x	4.2x
320^3	1795.86	822.48	423.87	219.39	112.57	81.40	14.84	121.0x	5.4x
400^3	3543.14	1628.12	853.50	430.59	223.55	177.86	25.28	140.1x	7.0x

(a) Computational times (t) and GPU speedup (S) for single precision

N	t_1	t_2	t_4	t_8	t_{16}	t_{24}	t_{GPU}	S_1	S_{24}
160^3	217.54	112.85	58.93	30.44	15.73	11.40	2.43	89.5x	2.4x
320^3	2016.50	911.12	472.84	245.08	125.08	90.07	31.09	64.8x	2.9x
400^3	3886.31	1799.22	928.89	481.99	246.80	240.05	57.42	67.6x	4.1x

(b) Computational times (t) and GPU speedup (S) for double precision

4. Discussions

Mint has been shown to deliver good performance on 3D finite difference codes [19]. Our algorithm is a 3D finite difference solver, but a non-traditional one. The grid is iterated in specific orders and a non-traditional stencil is used. Mint has delivered a surprisingly good GPU performance for our 3D PMM. A detailed comparison of the Mint-translated code with a hand-coded CUDA version would be an interesting investigation. We have experimented with the formulation of the update step, to search for an efficient formulation. Those optimization investigations indicate that the high number of branches introduced from conditioning the update computations, reduces the computational speed. Nevertheless, profiling has indicated some further optimization possibilities, for both CPU and GPU implementations. For instance, the current register use is very extensive. A better use of the registers might improve both implementations.

An interesting algorithmic extension is to try ideas from [10], in which approaches for parallelization of the otherwise sequential Fast Sweeping Method are presented. One of the suggested ideas there is to sweep the domain in different directions at the same time on copies of the data structure, and then synchronize the results. Sub-sweeps in different directions can for instance be computed on several GPUs simultaneously. The approach of performing full sweeps of subdomains from the same paper may increase the convergence rate of the algorithm. Weber et al. [12] present a variant of the 2D PMM method to make it run faster on a GPU. Similar extensions in 3D are possible, and might assure good reuse of transferred data.

Although an $O(N)$ method [12], the PMM method often needs more Sweeps than the Fast Sweeping method to converge. Therefore, the algorithm is not suitable for applications with strongly curved characteristics, such as seismic data processing. For such applications the updates need to be ordered somehow. The Fast Iterative Method is one approach to this, but the ordering is too weak for complicated problems [17]. A related method exists for sequential 2D code on isotropic examples [9], in which subdomains are swept in a specified order. This idea can be extended to 3D and parallelized for GPUs by sweeping a list of subdomains simultaneously, using one streaming multiprocessor each, on which the streaming vector processors make use of the parallelism of the 3D PMM method. Furthermore, the subdomains can be ordered using an approach similar to that of [16] to ensure a stronger ordering, and convergence also on anisotropic problems. Such an algorithm will be efficient also on problems with bending characteristics.

5. Conclusion

Simulating a propagating front is a computationally challenging problem, especially in 3D applications. Simulations are needed in several applications, where the solution is needed within a few seconds for the software to be used in an interactive manner. In 3D, sequential algorithms are only applicable on small grid sizes. We have presented a simple 3D Parallel Marching Method and applied it to the simulation of geological folding. The algorithm can be easily implemented on parallel architectures. Numerical experiments using OpenMP show near-linear scaling

on multicore CPUs. Using the automated C-to-CUDA code translator Mint, we obtained a CUDA implementation without manual GPU programming. The GPU implementation runs approximately 2.4-7 times faster than the fastest multi-threaded version on 24 CPU cores, giving hope to compute large 3D grids interactively in the future.

6. Acknowledgments

The presented work was funded by Statoil through its Academia Program, and the Research Council of Norway under grant 202101/I40. The work has been conducted at Kalkulo AS, a subsidiary of Simula Research Laboratory.

Appendix A. Conditional Upwind Approximations

As in most front propagation methods, it is of great importance that approximations are computed from upwind values. Upwind values are values that are passed by the front. Monotonic convergence is a fundamental property for convergence toward the viscosity solution for most algorithms [17, 1, 14]. A too small approximation will not be increased, since that would contradict the monotonicity assumption. Therefore, one must assure that the computed value uses solution values that are upwind from the updated node. We assert this by computing the characteristic curve of the approximation, and make sure that it is embedded in the convex hull of the nodes used in the computations. If it is not, the new approximation is rejected. A similar approach for isotropic problems are presented in [31], where the entrance point is used to find the rays in a seismic processing setting. From [22] we have the characteristics $x(s)$ to (1)

$$\frac{\partial x}{\partial s} = F\nabla T + \psi \mathbf{a}|\nabla T|. \tag{A.1}$$

Consider the stencil shape as given in Figure 2(a), where a new solution is sought for the pyramid-top node value $T_{i,j,k}$, and the nine nodes in the lower plane all have the third coordinate as $k-1$. The nodes on the lower plane are divided in eight groups of three nodes that form trirectangular tetrahedras with $T_{i,j,k}$ as apex. Similarly, 16 groups of two nodes forming right-angled triangles are created (dotted lines on the lower plane) as well as nine groups of one node each. From each of these groups solution estimates are created. If $T_{i,j,k}$ is bigger than the smallest acceptable of these approximations, we update the value at (i, j, k) with the new estimate. With estimates of the gradient of T and equation (A.1), the entrance point in the lower plane is given as

$$x_e = -dz\frac{F\frac{\partial T}{\partial x} + \psi \mathbf{a}_x|\nabla T|}{F\frac{\partial T}{\partial z} + \psi \mathbf{a}_z|\nabla T|}, \quad \text{and} \quad y_e = -dz\frac{F\frac{\partial T}{\partial y} + \psi \mathbf{a}_y|\nabla T|}{F\frac{\partial T}{\partial z} + \psi \mathbf{a}_z|\nabla T|}. \tag{A.2}$$

Assume the nodes $T_{i,j,k-1}, T_{i+1,j,k-1}$ and $T_{i+1,j+1,k-1}$ are three nodes in a trirectangular shape. With these nodes we estimate the partial derivatives in x and y direction with

$$\frac{\partial T}{\partial x} = \frac{T_{i+1,j,k-1} - T_{i,j,k-1}}{dx}, \quad \text{and} \quad \frac{\partial T}{\partial y} = \frac{T_{i+1,j+1,k-1} - T_{i+1,j,k-1}}{dy}. \tag{A.3}$$

Using these two estimates, we directly discretize (1) and get $\frac{\partial T}{\partial z}$ as the solution of a second degree polynomial. From (A.2) we get the entrance coordinates x_e, y_e, and the solution estimate $T^{new}_{i,j,k} = T_{i,j,k-1} + dz\frac{\partial T}{\partial z}$ is accepted if

$$0 < \min x_e, y_e, \quad y_e dx < x_e dy, \quad \text{and} \quad x_e < dxx. \tag{A.4}$$

That is to assure the entrance point is within the convex hull of the nodes used in the stencil. The remaining stencils using three values, are identical up to a rotation and reflection.

For the two node group using $T_{i,j,k-1}$ and $T_{i+1,j,k-1}$ we estimate $\frac{\partial T}{\partial x}$ with $\frac{T_{i+1,j,k-1} - T_{i,j,k-1}}{dx}$. That the characteristic curve to $T_{i,j,k}$ cut the line segment between $(i, j, k-1)$ and $(i+1, j, k)$ is to say that $y_e = 0$ of (A.2), that is

$$\frac{\partial T}{\partial y} = \frac{-\psi a_y|\nabla T|}{F}. \tag{A.5}$$

Together with (1) we get $\frac{\partial T}{\partial z}$ as the solution to a second degree polynomial. If the entrance point x_e from (A.2) satisfies $0 < x_e < dx$, we accept the new approximation. The remaining 15 stencils using two values are identical up to a rotation and reflection.

When only the value at $T_{i,j,k-1}$ is used, the characteristic curve must go from $(i, j, k-1)$ through (i, j, k). That is $x_e = 0, y_e = 0$ in (A.2), resulting in

$$F\frac{\partial T}{\partial x} + \psi a_x|\nabla T| = 0, \quad \text{and} \quad F\frac{\partial T}{\partial y} + \psi a_y|\nabla T| = 0. \tag{A.6}$$

The traveltime solution for this system is easiest found using the group velocity v_G, that is the velocity in the direction of motion [7, 3]. In our case we have the the group velocity in the z-direction as $v_G = F\frac{\frac{\partial T}{\partial z}}{|\nabla T|} + a_z$, and the arrival time to $T_{i,j,k} = T_{i,j,k-1} + \frac{dz}{v_G}$. For a general point with value $T_{l,m,n}$ at the distance $\mathbf{x} = (x, y, z)$ from index (i, j, k) the corresponding solution is

$$T_{i,j,k} \quad = \quad T_{l,m,n} + \frac{\mathbf{x}\cdot\mathbf{x}}{\mathbf{a}\cdot\mathbf{x} + F\sqrt{(1-\frac{|\mathbf{a}|^2}{F^2})|\mathbf{x}|^2 + \frac{(\mathbf{a}\cdot\mathbf{x})^2}{F^2}}}. \tag{A.7}$$

In 2D this result is the same as the analytical solution of a point source as shown in [24].

Reducing the number of redundant computations

In isotropic front propagations, only strictly smaller nodes should be used for creating solution estimates with upwind stencils [1]. For a general anisotropic problem the same principle does not hold. However, at least one of the used nodes must be smaller than the old estimate for there to be a possibility of an acceptable solution estimate [16]. In the above stencil formulation this correspond to $T^{old}_{i,j,k} > \min_{(a,b)\in\{(0,0),(1,0),(1,1)\}} T_{i+a,j+b,k-1}$ in the three node case, $T^{old}_{i,j,k} > \min_{a\in\{0,1\}} T_{i+a,j,k-1}$ for the two node case, and $T^{old}_{i,j,k} > T_{i,j,k-1}$ for the one node case.

Moreover, we can derive the following condition for the characteristics entrance point x_e and y_e to be grater than 0

$$n_x \quad = \quad \frac{\frac{\partial T}{\partial x}}{|\nabla T|} < \frac{a_x}{F}, \quad \text{and} \quad n_y = \frac{t_y}{|\nabla T|} < \frac{a_y}{F}. \tag{A.8}$$

If $a_x < 0$ then we must have $\frac{\partial T}{\partial x} < 0$, otherwise the solution will not be accepted, and hence we need not to create the estimate. The corresponding argument holds for the y_e entrance point.

Remark; Signed Distance

When simulating folds, one must distinguish between the inside and outside of the structure. In order for the fold to be consistent, the axial direction **a** is negative on the inside, and positive on the outside. The same holds for the solution T. Accordingly, the 3D PMM adjusts the initialisation step slightly. The initialised data is set to $-\infty$ on the inside, and $+\infty$ on the outside. The sign of $T_{i,j,k}$ during the update step is saved locally, and the update is performed with absolute values of the nodes. If a new solution is found, it is saved with the same sign as the previous approximation was.

References

[1] J. A. Sethian, Level Set Methods and Fast Marching Methods, Cambridge University Press, 1999.
[2] F. Qin, Y. Luo, K. B. Olsen, W. Cai, G. T. Schuster, Finite-difference solution of the eikonal equation along expanding wavefronts, Geophysics 57 (3) (1992) 478–487.
[3] Y. Wang, T. Nemeth, R. Langan, An expanding-wavefront method for solving the eikonal equations in general anisotropic media, Geophysics 71 (5) (2006) T129.
[4] E. Cristiani, A fast marching method for Hamilton-Jacobi equations modeling monotone front propagations, Journal of Scientific Computing 39 (2) (2009) 189–205.
[5] J. Sethian, A. Vladimirsky, Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms, SIAM Journal on Numerical Analysis 41 (1) (2004) 325–363.
[6] K. Alton, Dijkstra-like ordered upwind methods for solving static Hamilton-Jacobi equations, Ph.D. thesis (2010).

[7] J. Qian, Y.-T. Zhang, H.-K. Zhao, A fast sweeping method for static convex hamilton-jacobi equations, J. Sci. Comput. 31 (1-2) (2007) 237–271.
[8] P. A. Gremaud, C. M. Kuster, Computational study of fast methods for the eikonal equation, SIAM Journal on Scientific Computing 27 (6) (2006) 1803–1816.
[9] A. Chacon, A. Vladimirsky, Fast two-scale methods for eikonal equations, SIAM Journal on Scientific Computing 34 (2) (2012) A547–A578.
[10] H.K.Zhao, Parallel implementations of the fast sweeping method, Journal of Computational Mathematics 25 (4) (2007) 421 – 429.
[11] S. Li, K. Mueller, M. Jackowski, D. Dione, L. Staib, Physical-space refraction-corrected transmission ultrasound computed tomography made computationally practical, in: Medical Image Computing and Computer-Assisted Intervention - MICCAI 2008, Vol. 5242 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 280–288.
[12] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, R. Kimmel, Parallel algorithms for approximation of distance maps on parametric surfaces, ACM Transactions on Graphics 27 (4) (2008) 1–16.
[13] J. Vidale, Finite-difference calculation of travel times, Bulletin of the Seismological Society of America 78 (6) (1988) 2062–2076.
[14] E. Rouy, A. Tourin, A viscosity solutions approach to shape-from-shading, SIAM J. Numer. Anal. 29 (3) (1992) 867 – 884.
[15] P. Podvin, I. Lecomte, Finite difference computation of traveltimes in very contrasted velocity models: a massively parallel approach and its associated tools, Geophysical Journal International 105 (1991) 271–284.
[16] T. Gillberg, A Semi-Ordered Fast Iterative Method (SOFI) for Monotone Front Propagation in Simulations of Geological Folding, in: MODSIM2011, 19th International Congress on Modelling and Simulation, 2011, pp. 631–647.
[17] W.-K. Jeong, R. T. Whitaker, A fast iterative method for eikonal equations, SIAM Journal on Scientific Computing 30 (5) (2008) 2512–2534.
[18] S. Bak, J. McLaughlin, D. Renzi, Some Improvements for the Fast Sweeping Method, SIAM Journal on Scientific Computing 32 (2010) 2853–2874.
[19] D. Unat, X. Cai, S. Baden, Mint: Realizing CUDA performance in 3D stencil methods with annotated C, in: Proceedings of the 25th International Conference on Supercomputing (ICS'11), ACM Press, 2011, pp. 214–224.
[20] S. A. Petersen, Ø. Hjelle, Earth recursion, an important component in shared earth model builders, EAGE 70th Conference & Exhibition, Extended Abstracts.
[21] A. Tveito, A. M. Bruaset, O. Lysne (Eds.), Simula Research Laboratory – by thinking constantly about it, Springer, 2010, Ch. Turning Rocks into Knowledge.
[22] Ø. Hjelle, S. A. Petersen, A Hamilton-Jacobi framework for modeling folds in structural geology, Mathematical Geosciences 43 (7) (2011) 741–761.
[23] J. G. Ramsay, Folding and fracturing of rocks, McGraw-Hill, New York and London, 1967.
[24] E. Kornhauser, Ray Theory for Moving Fluids, The Journal of the Acoustical Society of America 25 (5) (1953) 945–949.
[25] N. Rawlinson, M. Sambridge, Multiple reflection and transmission phases in complex layered media using a multistage fast marching method, Geophysics 69 (5) (2004) 1338–1350.
[26] I. Berre, K. H. Karlsen, K.-A. Lie, J. R. Natvig, Fast computation of arrival times in heterogeneous media, Computational Geosciences 9 (4) (2005) 179–201.
[27] A. M. Popovici, J. A. Sethian, 3-D imaging using higher order fast marching traveltimes, Geophysics 67 (604, Issue 2).
[28] J.-W. Huang, G. Bellefleur, Joint transmission and reflection traveltime tomography using the fast sweeping method and the adjoint-state technique, Geophysical Journal International 188 (2) (2012) 570–582.
[29] W.-K. Jeong, P. T. Fletcher, R. Tao, R. Whitaker, Interactive visualization of volumetric white matter connectivity in DT-MRI using a parallel-hardware Hamilton-Jacobi solver, IEEE Transactions on Visualization and Computer Graphics 13 (2007) 1480–1487.
[30] Q. Lin, Enhancement, extraction, and visualization of 3d volume data [elektronisk resurs], Ph.D. thesis, Linköping University, Institute of Technology (2003).
[31] J. Zhang, Y. Huang, L.-P. Song, Q.-H. Liu, Fast and accurate 3-D ray tracing using bilinear traveltime interpolation and the wave front group marching, Geophysical Journal International 184 (3) (2011) 1327–1340.