A Massive Parallel Fast Marching Method

Petr Kotas[1], Roberto Croce[2], Valentina Poletti[2], Vit Vondrak[1], and Rolf Krause[2]

[1] Department of Applied Mathematics, VSB-Technical University of Ostrava, CZ 708 33 Ostrava-Poruba, Czech Republic {petr.kotas, vit.vondrak}@vsb.cz
[2] Institute of Computational Science, University of Lugano, CH-6904 Lugano, Switzerland {roberto.croce, rolf.krause, valentina.poletti}@usi.ch

1 Introduction

In this paper we present a novel technique based on domain decomposition which enables us to perform the fast marching method (FMM) [Sethian(1996)] on massive parallel high performance computers (HPC) for given triangulated geometries. The FMM is a wildely used numerical method and one of the fastest serial state-of-the-art techniques for computing the solution to the Eikonal equation.

For clarification we define an open set $\Omega = \Omega_I \cup \Omega_E \cup \Gamma \subset \mathbb{R}^2$(or $\mathbb{R}^3$) where Ω_I is the interior and Ω_E the exterior of the domain enclosed by Γ and the bounding box itself, as shown in Fig. 1. Then the resulting problem for the Eikonal equation reads as the following boundary value formulation

$$\begin{aligned} |\nabla T(x)|F(x) &= 1 \quad \text{for } x \in \Omega, \\ T|_\Gamma &= 0, \\ T(x) > 0 \quad &\text{for } x \in \Omega_E \quad \text{and} \quad T(x) < 0 \quad \text{for } x \in \Omega_I. \end{aligned} \tag{1}$$

with $F(x)$ as speed function. The solution to this problem with $F(x) = 1$ leads to the well-known signed distance function T with respect to Γ.

Signed distance functions are indispensable in varied fields such as seismic imaging, approximation of geodesic distances and computational fluid dynamics [Sethian(1999)]. Hence, finding a fast solution method for the Eikonal equation is of high interest. Several attempts lead to different approaches for a fast and reliable solver. Among various techniques are iterative schemes, expanding box schemes, expanding wavefront schemes, and sweeping schemes [Jeong and Whitaker(2007)].

Though the FMM is a very efficient algorithm of complexity $O(N\ log\ N)$, its major drawback is that it cannot be easily parallelized due to its inherently serial nature. Attempts to parallelize it either modify its underlying scheme, losing some of its agility, or have limited scalability. Nevertheless various authors tried to achieve a faster and more efficient scheme.

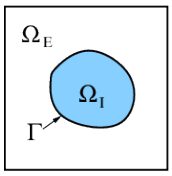

Fig. 1. Set-up of the Eikonal equation problem.

In [Weber et al.(2008)Weber, Devir, Bronstein, Bronstein, and Kimmel] a modification of the FMM algorithm is introduced to make it parallelizable. The implementation relies heavily on memory shareability, and the maximal number of processes is limited by the size of the updating stencil. Other authors have relied on different schemes all together, such as level set methods or variations thereof [Herrmann(2003), Jeong and Whitaker(2007), Tugurlan(2008)]. The speed-up gained by the scalability of these methods comes at the loss of serial algorithmic efficiency as the complexity of the underlying algorithms is higher.

In this paper we will present a parallel algorithm for the computation of the FMM on distributed memory machines via MPI. Another strength of our parallel algorithm is that inter-processsor communication does not exist during the parallel FMM computation, each core is basically computing independently the level-set function on its subdomain. This is possible, because each subdomain computes accurate boundary values for its local dataset before starting the parallel FMM.

The remainder of this paper is organized as follows: The sequential fast marching method is shortly explained in section 2 and our extensions towards a Massive Parallel Fast Marching Method (MPFMM) are presented in section 3. Finally, we present several numerical results in section 4 which investigate the performance of the new MPFMM-algorithm. We conclude with some remarks in section 5. Finally, notice that a more extensive version of this paper can be found as ICS-preprint [Croce et al.(2013)Croce, Kotas, Krause, and Poletti].

2 Sequential Fast Marching Method (SFMM)

The Fast Marching Method [Sethian(1996), Sethian(1999)] is designed to efficiently solve the Eikonal equation (1). To do so, the FMM uses the first order Godunov scheme to aproximate the gradient term $|\nabla T(x)|$, thus the Eikonal equation is given as

$$F(x)\begin{bmatrix}\max(D_{ijk}^{-x}T(x),-D_{ijk}^{+x}T(x),0)^2+\\ \max(D_{ijk}^{-y}T(x),-D_{ijk}^{+y}T(x),0)^2+\\ \max(D_{ijk}^{-z}T(x),-D_{ijk}^{+z}T(x),0)^2\end{bmatrix}^{\frac{1}{2}}=1 \tag{2}$$

where D_{ijk}^{-} is the first order backward and D_{ijk}^{+} the first order forward finite difference operator. The equation (2) utilizes the upwind technique for approx-

Algorithm 1 Sequential Fast Marching Method (SFMM).

1: Compute the distance $T(x)$ to all node values that are directly adjacent to the interface and tag them as *accepted*. Tag all nodes adjacent to these *accepted* nodes as *narrow* nodes and all others as *away* nodes.
2: Compute $T(x)$ of all *narrow* nodes via equation (1), treating $T(x)$ in any adjacent *narrow* or *away* node as ∞. Set the loop index $n = 1$.
3: **repeat**
4: Mark as *accepted* the *narrow* node i, j, k with the smallest $T(x)$ value, denoted by $T(x)^n = T(x)_{i,j,k}$.
5: Mark all *away* nodes adjacent to $T(x)_{i,j,k}$ as *narrow*.
6: Recompute the $T(x)$ values of all *narrow* nodes adjacent to $T(x)_{i,j,k}$ by equation (1), treating $T(x)$ in any adjacent *narrow* or *away* node as ∞.
7: Set $n = n + 1$.
8: **until** All nodes are taged as *accepted*

imating the Eikonal equation (1). This works, because the front Γ propagates forward and visits each cell only once.

In the core of the FMM there are three lists preserving the state of each cell in the computation domain. Nodes marked as *accepted* are nodes for which the singed distance function has already been computed. Nodes within the vicinity of known nodes are marked *narrow band* and are updated according to equation (2). Finally, nodes with unknown distance are marked as *away*. The complete and easy to follow description of FMM algorithm is given by Sethian in his book [Sethian(1999)].

3 Massive Parallel Fast Marching Method (MPFMM)

A few existing strategies for parallelizing the FMM exist, with varying level of success. The simplest approach is to split the interface in two disjoint regions and to run the FMM on both of them at once. This method however lacks larger parallelism and it cannot achieve ideal load-balancing. To overcome this problem another natural approach of decomposing the computational region into a group of sub regions was studied in [Tugurlan(2008)]. This implementation utilizes the strategy of overlapping domain decomposition. To exchange the boundary information an iterative update strategy is used. The overall algorithm is designed in an iterative fashion, since after the boundary update, the FMM needs to be re-run on each sub-domain. This process repeats until convergence is achieved. Finally, methods decomposing the computational domain in such a way that each sub-domain contains part of the initial interface are shown for instance in [Núnez(2011)]. With this strategy global minima need to be exchanged in each FMM run, therefore the algorithm is not entirely parallel in nature. Furthermore, none of the existing methods is able to provide reasonable scalability and performance needed by large datasets.

In our approach we use domain decomposition with a combination of exact boundary conditions on each sub-domain. Our decomposition scheme does not require the initial interface to be present in each sub-domain, however the exact distance between the initial interface and the sub-domain boundaries is

Algorithm 2 Parallel Fast Marching Method (MPFMM).

1: Divide the given gridded domain into N sub-domains $D_0, D_1, ...D_N$
2: On each domain D_i chose initial points on the local narrow band around the geometry and on the domain boundary.
3: Load balance initial data points among domains.
4: Compute the initial data in parallel on each domain using closest point projection to triangulated geometry.
5: Compute the SDF in parallel on each domain using the SFMM algorithm

necessary. This property allows us to loosen the strict limit on scalability that the above mentioned methods possess and allow even very large computational domains to be processed.

The features of our algorithm can be summarized as:

- Easy to implement since it is the SFMM with according boundary conditions on the subdomains and narrow band
- The algorithm works for massive parallel computations
- Excellent FMM-speedup on fine grids since no communication is needed
- Parallelization improves accuracy, for as the number of processes increases, the number of points computed directly with closest point projection increases.
- The parallel algorithm works also for second order schemes

The parallelization of the entire algorithm basically consists of the parallelization of its two main subroutines, i.e.

1. narrow band initialization
2. ghostcell boundary data computation

4 Numerical experiments

In the following section we designed a series of test cases to exploit the numerical features of our algorithm. We show numerically the performance and scalability of our new massive parallel fast marching algorithm, as well as to check the accuracy of the computed signed distance function. In particular we will show that it maintains first order error, as can be easily deduced from the type of evolution scheme we use.

4.1 MPFMM performance for "analytical circle/sphere"

At first, we investigate the parallel error propagation of our MPFMM algorithm. We make use of a 2D circle with its center $x_c = (0, 0, 0)$, radius $R = 3.0$, and set in a computational domain with the size $[-10, 10] \times [-10, 10]$. The signed distance function for this geometry can be computed analytically via the following equation

$$T(\mathbf{x}) = R_0 - \|\mathbf{x} - \mathbf{x_c}\|, x \in R^N. \tag{3}$$

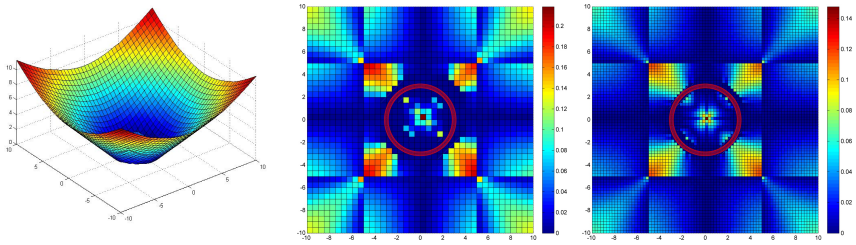

Fig. 2. On the left: parallel signed distance computation on 16 cores for a circle. In the middle and right: error-evolution on a 41×41 and a 81×81 grid.

This simplifies both the initialization procedure and the computation of the error. We used two different grid resolutions: 41×41 and 81×81 grid cells on a uniform grid with grid cell sizes $dx = 0.48780$ and $dx = 0.24691$. The computation is performed on 16 cores. Figure 2 shows the error distribution through the global domain subdivided into 16 subdomains.

As expected, the MPFMM algorithm aggregates the error in diagonal direction. This is because the fast marching method computes the discrete gradient in the horizontal and vertical coordinate directions. The maximum global error on each grid is 0.21 and 0.14. This is less than the grid sizes $dx = 0.48780$ and $dx = 0.24691$. Thus this experiment shows that the MPFMM algorithm maintains first order accuracy on all local subdomains.

In order to further exploit the nice initialization properties of the analytical sphere, we set up the 3D problem described in [Herrmann(2003)]. In this problem, the sphere is located inside a unit cube, with $\mathbf{x_c} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ and radius $R_0 = 0.25$. The signed distance function in this problem, is defined similarly to equation (3), thus we provide the initial data for the MPFMM algorithm using this equation. Again, we make use of two grid sizes: $192 \times 192 \times 192$ and $384 \times 384 \times 384$. We run our algorithm using up to 8192 parallel cores. Figure 3, shows the MPFMM algorithm's super linear scalability over all processor ranges. This is due to the logarithmic complexity of the sequential Fast Marching, which therefore scales logarithmically. It is worth noting that the only limiting factor in the number of subdomains on which the MPFMM algorithm can be parallelized on is the number of global grid cells. This does not present a problem particularly when dealing with larger domains. Thus it is a highly scalable algorithm for computing the signed distance function on large domains.

4.2 MPFMM for triangulated surfaces

Here we show some numerical tests targeting the overall performance of the MPFMM algorithm. Thus we run our MPFMM algorithm together with the data initialization routine. We run our algorithm on two different benchmark geometries, each of which is composed of a different number of triangles:

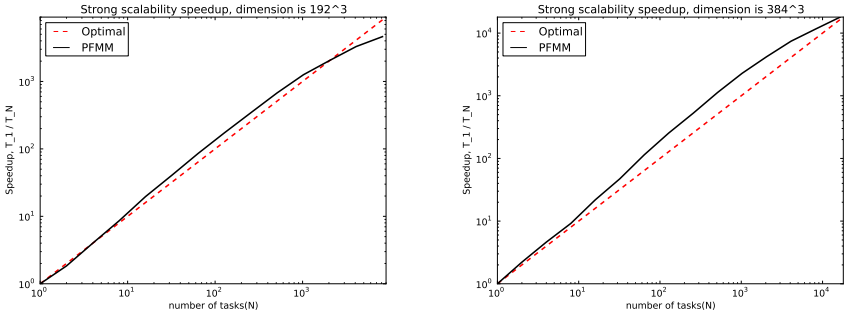

Fig. 3. Speedup for "analytical sphere" for up to 8192 cores (left 192^3, right 384^3).

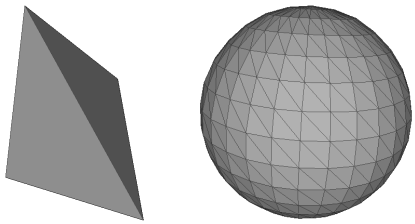

Fig. 4. A tetrahedron consisting of 4 triangles (left) and a triangulated sphere consisting of 840 triangles (right) are used for our speedup investigations.

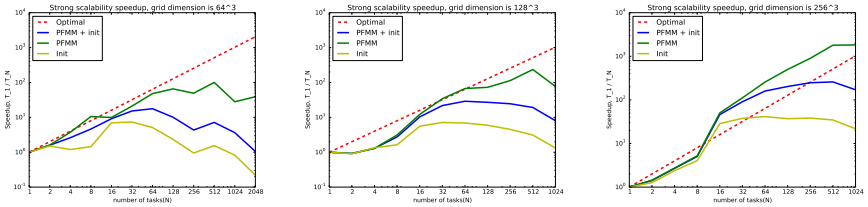

Fig. 5. Speedup for up to 2048 cores performed for the triangulated tetrahedron with three grid-resolutions: 64^3 (left) and 128^3 (middle) and 256^3 (right) gridcells.

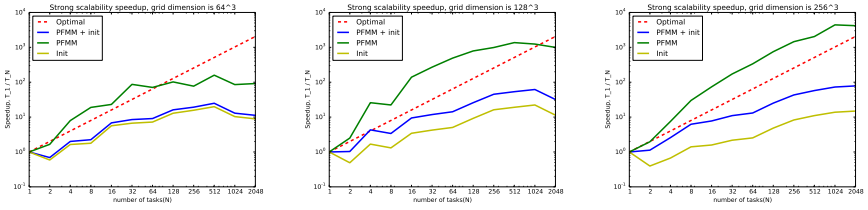

Fig. 6. Speedup for up to 2048 cores performed for the triangulated sphere with three grid-resolutions: 64^3 (left) and 128^3 (middle) and 256^3 (right) gridcells.

- tetrahedron (Figure 4) consisting of 4 triangles,
- sphere (Figure 4) comprising 840 triangles.

We run our algorithm on the benchmark geometries using three different meshes: 64^3 and 128^3 and 256^3. With this set up we can investigate the performance of all the important algorithmic parts.

In the first test we compute the signed distance function on a tetrahedron. This shows the performance of the MPFMM algorithm paired with the initialization routine for a very simple initialization. Due to the simple nature of the geometry, we can easily deduce the performance of the MPFMM with an accelerated search algorithm for the closest triangle. Such results are shown in Figure 5.

In the second test, we compute the signed distance function on a triangulated sphere in order to investigate the performance of the MPFMM on larger triangulated meshes. Results depicted in Figure 6 show that the MPFMM maintains good scalability and performance in this case as well.

Both tests show similar scaling properties and performance, suggesting that the latter are maintained through larger meshes. However, the algorithm performs more poorly for the smaller mesh of size 64^3. This is due to the fact that the ratio of set-up time to PFMM-computation time is higher, as the number of degrees of freedom is too small to obtain reasonably efficient parallel computation. These benchmark tests therefore show that scaling is limited only on the lower size of the mesh.

5 Concluding Remarks

In this paper we presented a parallel algorithm for the fast marching method. We investigated several massive parallel FMM-computations (MPFMM) for simple geometries with respect to their speedup behaviour on up to 2048- and 8192 cores respectively. As expected, the parallel FMM-speedup scales optimally for fine grid resolutions and the numerical results show an according global signed distance function. However, the parallel boundary value initialization could still be improved by storing the geometry information in a tree and use a triangle search with a special partition on the tree, instead of distributing the entire geometry to each process. Furthermore, we showed that the order of convergence is conserved for the parallel computations.

6 Acknowledgments

This result/work/publication was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070) and the project of major infrastructures for research, development and innovation of Ministry of Education, Youth and Sports with reg. num. LM2011033.

References

[Croce et al.(2013)Croce, Kotas, Krause, and Poletti] R. Croce, P. Kotas, R. Krause, and V. Poletti. A new massive parallel fast-marching algorithm for signed distance computations with respect to complex triangulated surfaces. Preprint 201314, Institute of Computational Science, Università della Svizzera Italiana, December 2013.

[Herrmann(2003)] M. Herrmann. A domain decomposition parallelization of the fast marching method. Technical report, Center for Turbulence Research, 2003.

[Jeong and Whitaker(2007)] W. Jeong and R.T. Whitaker. A fast iterative method for a class of hamilton-jacobi equations on parallel systems. *Sch. Comput. Univ. Utah*, 84112(2):1–4, 2007.

[Núnez(2011)] L.A.Z. Núnez. Parallel implementation of fast marching method. Technical report, Massachusetts Institute of Technology, 2011.

[Sethian(1996)] J.A. Sethian. A fast marching method for monotonically advancing fronts. *Proc. Natl. Acad. Sci. United States Am.*, 93(4):1591–1595, 1996.

[Sethian(1999)] J.A. Sethian. *Level set methods and fast marching methods.* Cambridge University Press, 1999. ISBN 9780521645577.

[Tugurlan(2008)] M.C. Tugurlan. *Fast marching methods - Parallel implementation and analysis.* Dissetation, Louisiana State University, 2008.

[Weber et al.(2008)Weber, Devir, Bronstein, Bronstein, and Kimmel] O. Weber, Y.S. Devir, A.M. Bronstein, M.M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Trans. Graph.*, 27(4):104, 2008.