Procedia Computer Science

Volume 80, 2016, Pages 2271–2275

ICCS 2016. The International Conference on Computational Science

Comparison of the Parallel Fast Marching Method, the Fast Iterative Method, and the Parallel Semi-Ordered Fast Iterative Method

Josef Weinbub[1] and Andreas Hössinger[2]

[1] Christian Doppler Laboratory for High Performance TCAD, Institute for Microelectronics, TU Wien, Wien, Austria
weinbub@iue.tuwien.ac.at

[2] Silvaco Europe Ltd., St Ives, Cambridge, United Kingdom
andreas.hoessinger@silvaco.com

Abstract

Solving the eikonal equation allows to compute a monotone front propagation of anisotropic nature and is thus a widely applied technique in different areas of science and engineering. Various methods are available out of which only a subset is suitable for shared-memory parallelization, which is the key focus of this analysis. We evaluate three different approaches, those being the recently developed parallel fast marching method based on domain decompositioning, the inherently parallel fast iterative method, and a parallel approach of the semi-ordered fast iterative method, which offers increased stability for variations in the front velocity as compared to established iterative methods. We introduce the individual algorithms, evaluate the accuracy, and show benchmark results based on a dual socket Intel Ivy Bridge-EP cluster node using C++/OpenMP implementations. Our investigations show that the parallel fast marching method performs best in terms of accuracy and single thread performance and reasonably well with respect to parallel efficiency for up to 8-16 threads.

Keywords: Parallel fast marching method, fast iterative method, semi-ordered fast iterative method, parallel algorithm, eikonal equation, OpenMP

1 Introduction

Simulating an expanding front is a fundamental step in many computational science and engineering applications, such as image segmentation [6], brain connectivity mapping [12], medical tomography [11], seismic wave propagation [13], geological folds [9], semiconductor process simulation [17], and computational geometry [15]. In general, an expanding front originating from a start position Γ is described by its first time of arrival T to the points of a domain Ω. This problem can be described by solving the eikonal equation [14], which for n spatial dimensions reads: $\|\nabla T(\mathbf{x})\|_2 \, F(\mathbf{x}) = 1 \quad \mathbf{x} \in \Omega \subset \mathbb{R}^n \, , \, T(\mathbf{x}) = g(\mathbf{x}) \quad \mathbf{x} \in \Gamma \subset \Omega$.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.408

$T(\mathbf{x})$ is the unknown solution (i.e. first time of arrival), $g(\mathbf{x})$ are boundary conditions for Γ, and $F(\mathbf{x})$ is a positive speed function, with which the interface information propagates in the domain. Generally speaking, isosurfaces to the solution represent the position of the front at a given time, and can thus be regarded as the geodesic distance relative to Γ. If the velocity $F = 1$, then the solution $T(\mathbf{x})$ represents the minimal Euclidian distance from Γ to $\mathbf{x}$.
In this work, we compare three relevant methods for solving the eikonal equation in a shared-memory parallel setting, for which an overview is given in the following sections. A domain decomposition-based parallel approach of the fast marching method (FMM) (cf. Section 2), the inherently parallel fast iterative method (FIM) (cf. Section 3), and a parallel approach of the semi-ordered fast iterative (SOFI) method (cf. Section 4). Section 5 compares implementations based on C++ and OpenMP of those three methods with respect to execution speed, parallel speedup, and accuracy.

2 Parallel Fast Marching Method

The original FMM [14] is inherently sequential (due to the fact that identifying the smallest solution value is a global process) and previous attempts to parallelize it have been unsatisfactory [2][11]. However, recently a shared-memory parallelization method for the FMM has been developed which is used in this work [18]. The computational domain is partitioned into equal parts (i.e. subgrids) via a block-partitioning scheme; each thread is responsible for a specific subgrid. A single ghost layer is used in all interior spatial decomposition directions to ensure that the parallel algorithm properly computes the entire domain; the updates at the *inner* boundary cells (i.e. boundary cells introduced by the partitioning scheme) are forwarded to the neighboring threads allowing the local solutions to influence the global solution process. By using a domain decomposition technique, which is due to the use of ghost zones to be considered an *overlapping* domain decomposition approach, each thread (and thus partition) has its own minimum heap data structure for finding the smallest value within the individual subgrid's `BAND` set. Thereby the primary reason for the FMM to not favor parallelization is eliminated, as no synchronized access has to be implemented. The eikonal equation is thus solved by the threads on their individual subgrid, including the ghost cells, which gives rise to parallelism. Details of this method are presented in [18].

3 Fast Iterative Method

The FIM was originally implemented for parallel execution on Cartesian meshes [10] and later extended to triangular surface meshes [7]. Due to its inherent parallel nature no parallel extension was required, contrary to the FMM (cf. Section 2) and the SOFI method (cf. Section 4). The FIM relies on a modification of a label correction scheme coupled with an iterative procedure for the mesh node update. In particular, there is no fixed node update order and several nodes can be updated simultaneously, enabling parallelism. More concretely, the inherent high degree of parallelism is due to the ability of processing all nodes of an active list (i.e. narrow band) in parallel, thus efficiently supporting a single instruction, multiple data parallel execution model. Therefore, the FIM is suitable for implementations on highly parallel accelerators, such as graphics adapters [10][11]. The iterative nature of the algorithm requires the use of an error threshold; the smaller the threshold the more accurate the method computes the solution by simultaneously requiring more iterations. Details of this algorithm can be found in the original work [10]. Although FIM has been primarily investigated regarding fine-grained parallelism on accelerators, investigations on shared-memory approaches have also been conducted [4][5][17].

4 Parallel Semi-Ordered Fast Iterative Method

Although the FIM provides superior parallel performance to other available methods (in most cases), its performance is problem dependent. Complex speed functions tend to significantly increase the solution time. To overcome this shortcoming, the SOFI method has been developed [8]; SOFI is based on both the FIM as well as on the Two-Queue method [1]. SOFI enforces an ordering to get the iterative behavior closer to front tracking methods, i.e., fast marching and wavefront tracking methods, in turn offering an increased stability, when faced with intricate speed functions. Front tracking methods inherently favor sequential execution, therefore parallel scalability is by definition inferior to that of the FIM. Rather than computing all *active* nodes in parallel (as is the case with the FIM), the SOFI method *pauses* some of the awaiting updates according to a cutoff criterion based on statistical in-situ analysis of the solution values. Details can be found in the original work [8]. Although originally developed as a serial algorithm, the SOFI method has been extended to support shared-memory parallelism [16]: The parallel approach achieves - albeit using straightforward parallelization techniques via partitioning the active nodes' iteration space and thread-synchronized data structures - reasonable performance and is thus investigated in this work.

5 Benchmarks

In this section, we compare the introduced methods with respect to execution performance (serial and parallel), parallel efficiency, and accuracy. Our benchmarks use established synthetic problem cases [3][4][9] and cover different three-dimensional problems with varying problem sizes (100^3 and 200^3 Cartesian cube grids using **0.0** and **1.0** for lower and upper bounds, respectively), speed functions as described in [16], and multiple-source configurations, i.e., 100 source nodes randomly spread over the entire simulation domain. The entire simulation domains are computed. The benchmarks have been carried out on a dual socket machine, equipped with two eight-core Intel Xeon E5-2650v2 (Ivy Bridge-EP) and 64 GB of DDR3 1866 ECC main memory. The Intel C++ compiler version 16 has been used as well as 64-bit floating point precision (i.e. `double`) and - for the FIM - an error threshold of $\varepsilon = 10^{-12}$ has been utilized. The fastest execution times out of five repetitions have been recorded and used for the investigation.

Figure 1-3 compare the strong scaling results of the parallel FMM and the FIM for both grid setups. The execution times are shown in logarithmic scale to increase the identifiability of the presented data sets, as the timings for the various setups would otherwise potentially be indistinguishable from each other. As can be seen from the execution times the FIM struggles in general with complicated speed functions whereas the parallel FMM and SOFI method fare significantly better. The parallel efficiency improves for increased problem sizes, which is due to the increased workload per thread. For the 100^3 problem size, good scalability is achieved for up to eight threads (efficiency of around 90%) by all methods. In turn, for the 200^3 case, all methods perform well for up to 16 threads. A tendency for super-linear scaling for the parallel FMM is identified which is due to the decomposition scheme: For one thread, the required additional decomposition logic results in a disproportionate overhead which is alleviated for higher thread numbers. However, for more than eight threads, the book keeping required for handling the larger thread numbers counters this effect again. Table 1(a,b) provides a detailed comparison of the serial execution times (i.e. with one thread) of the different benchmark setups. The results show that the parallel FMM and the SOFI method alternate in claiming the single thread performance lead, albeit the SOFI method being disproportionately slower ($\geq 55\%$) than the FMM for the F_{osc} speed function.

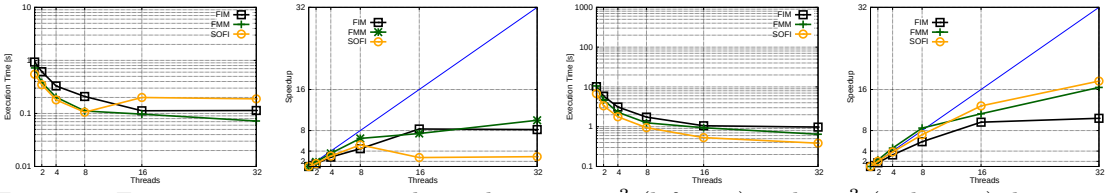

Figure 1: F_{const} execution times and speedups on 100^3 (left two) and 200^3 (right two) domain

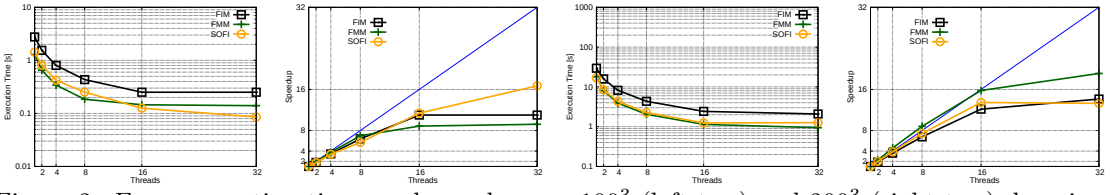

Figure 2: F_{check} execution times and speedups on 100^3 (left two) and 200^3 (right two) domain

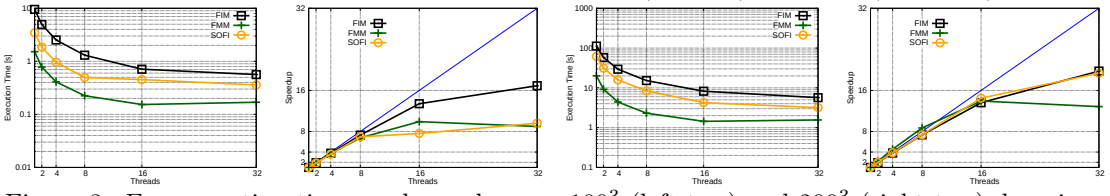

Figure 3: F_{osc} execution times and speedups on 100^3 (left two) and 200^3 (right two) domain

	F_{const}	F_{check}	F_{osc}
FMM	0.715878	1.28597	1.51159
FIM	0.931384	2.76385	9.53287
SOFI	0.548253	1.42799	3.44997

(a) 100^3 Execution Times

	F_{const}	F_{check}	F_{osc}
FMM	10.4973	17.9843	20.0329
FIM	10.1596	29.4321	112.024
SOFI	6.79996	16.7754	61.6414

(b) 200^3 Execution Times

	$\mathbf{L_1}$	$\mathbf{L_2}$	$\mathbf{L_\infty}$
FMM	$8 \cdot 10^{-3}$	$7.3 \cdot 10^{-5}$	$1 \cdot 10^{-3}$
FIM	$17 \cdot 10^{-3}$	$31 \cdot 10^{-5}$	$15 \cdot 10^{-3}$
SOFI	$17 \cdot 10^{-3}$	$31 \cdot 10^{-5}$	$15 \cdot 10^{-3}$

(c) Error Norms

Table 1: Comparison of serial execution times (i.e. single threaded) between the parallel FMM, the FIM, and the parallel SOFI method for different speed functions for the 100^3 (a) and 200^3 (b) test case. (c) The L_1, L_2, and L_∞ norms of the parallel FMM, the FIM, and the parallel SOFI method are compared for the single source, 100^3 test case using a constant speed function.

Overall, the FMM offers the best serial execution performance over the entire spectrum, considering the relatively small shortfall to the SOFI method ($< 35\%$) when the FMM is indeed slower. On the contrary, the FIM offers inferior serial execution performance. Table 1(c) shows the rounded L_1, L_2, and L_∞ error norms of the individual methods for a single source problem (i.e. a single source node located at $\mathbf{0.0}$) with constant speed for a 100^3 grid. As can be seen from the results, the parallel FMM offers the highest accuracy and is thus the clear favorite for accuracy-focused applications.

6 Conclusion

Three relevant eikonal equation solver algorithms have been investigated with a focus on three-dimensional problem cases and shared-memory parallelization: The domain decomposition parallel FMM, the FIM, and the parallel SOFI method. An introduction to each method has been given followed by a description of the benchmark and comparison setups. The serial and parallel execution times as well as accuracy have been shown and discussed. Overall, the parallel FMM has been shown to perform best. Future work will extend the investigations beyond synthetic benchmarks and will focus on real-world problem cases arising from surface evolution simulation techniques.

Acknowledgments. The financial support by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development is gratefully acknowledged. The presented computational results have been achieved using the Vienna Scientific Cluster (VSC).

References

[1] Stanley Bak et al. Some Improvements for the Fast Sweeping Method. *SIAM J.Sci.Comput.*, 32(5):2853–2874, 2010. DOI: 10.1137/090749645.

[2] Michael Breuß et al. An Adaptive Domain-Decomposition Technique for Parallelization of the Fast Marching Method. *Appl.Math.Comput.*, 218(1):32–44, 2011. DOI: 10.1016/j.amc.2011.05.041.

[3] Adam Chacon and Alexander Vladimirsky. Fast Two-Scale Methods for Eikonal Equations. *SIAM J.Sci.Comput.*, 34(2):A547–A578, 2012. DOI: 10.1137/10080909X.

[4] Florian Dang and Nahid Emad. Fast Iterative Method in Solving Eikonal Equations: A Multi-level Parallel Approach. *Proc.Comput.Sci.*, 29:1859–1869, 2014. DOI: 10.1016/j.procs.2014.05.170.

[5] Florian Dang et al. A Fine-Grained Parallel Model for the Fast Iterative Method in Solving Eikonal Equations. In *Proc. of 3PGCIC*, pages 152–157, 2013. DOI: 10.1109/3PGCIC.2013.29.

[6] Nicolas Forcadel et al. Generalized Fast Marching Method: Applications to Image Segmentation. *Numer.Algorithms*, 48(1-3):189–211, 2008. DOI: 10.1007/s11075-008-9183-x.

[7] Zhisong Fu et al. A Fast Iterative Method for Solving the Eikonal Equation on Triangulated Surfaces. *SIAM J.Sci.Comput.*, 33(5):2468–2488, 2011. DOI: 10.1137/100788951.

[8] T. Gillberg. A Semi-Ordered Fast Iterative Method (SOFI) for Monotone Front Propagation in Simulations of Geological Folding. In *Proc. of MODSIM*, pages 631–647, 2011.

[9] Tor Gillberg et al. Parallel Solutions of Static Hamilton-Jacobi Equations for Simulations of Geological Folds. *J.Math.Indus.*, 4(10):1–31, 2014. DOI: 10.1186/2190-5983-4-10.

[10] Won-Ki Jeong and Ross T. Whitaker. A Fast Iterative Method for Eikonal Equations. *SIAM J.Sci.Comput.*, 30(5):2512–2534, 2008. DOI: 10.1137/060670298.

[11] Shengying Li et al. Physical-Space Refraction-Corrected Transmission Ultrasound Computed Tomography Made Computationally Practical. In *Lect.Notes.Comput.Sc.*, volume 5242, pages 280–288, 2008. DOI: 10.1007/978-3-540-85990-1_34.

[12] Emmanuel Prados et al. Control Theory and Fast Marching Techniques for Brain Connectivity Mapping. In *Proc. of CVPR*, volume 1, pages 1076–1083, 2006. DOI: 10.1109/CVPR.2006.89.

[13] Nick Rawlinson and Malcolm Sambridge. Multiple Reflection and Transmission Phases in Complex Layered Media Using a Multistage Fast Marching Method. *Geophy.*, 69(5):1338–1350, 2004. DOI: 10.1190/1.1801950.

[14] James A. Sethian. A Fast Marching Level Set Method for Monotonically Advancing Fronts. *P.Natl.A.Sci.*, 93(4):1591–1595, 1996.

[15] James A. Sethian. *Level Set Methods and Fast Marching Methods.* Cambridge University Press, 1999. ISBN: 978-0521645577.

[16] Josef Weinbub et al. Shared-Memory Parallelization of the Semi-Ordered Fast Iterative Method. In *Proc. of HPC*, pages 217–224, 2015.

[17] Josef Weinbub and Andreas Hössinger. Accelerated Redistancing for Level Set-Based Process Simulations with the Fast Iterative Method. *J.Comp.Elect.*, 13(4):877–884, 2014. DOI: 10.1007/s10825-014-0604-x.

[18] Josef Weinbub and Andreas Hössinger. Shared-Memory Parallelization of the Fast Marching Method Using an Overlapping Domain-Decomposition Approach. In *Proc. of HPC*, 2016.