Procedia Computer Science

Volume 29, 2014, Pages 1859–1869

ICCS 2014. 14th International Conference on Computational Science

Fast iterative method in solving eikonal equations : a multi-level parallel approach

Florian Dang[12] and Nahid Emad[23]

[1] Silkan. Meudon-la-Forêt, France
florian.dang@silkan.com
[2] Laboratoire PRiSM. Université de Versailles, France
[3] Maison de la Simulation. CNRS, Saclay, France
nahid.emad@uvsq.fr

Abstract

The fast marching method is widely used to solve the eikonal equation. By introducing a new way of managing propagation interfaces which avoid the use of expensive data structures, the fast iterative method reveals to be a faster variant with a higher parallel potential compared to the fast marching method. We investigate in this paper a multi-level parallel approach for the fast iterative method which is well fitted for today heterogenous and hierarchical architectures. We show experiment results which focus on the fine-grained parallel level of the algorithm and we give a performance analysis.

Keywords: eikonal equation, fast iterative method, multi-level parallelism, Hamilton-Jacobi equations, fast marching method, parallel upwind finite difference

1 Introduction

The eikonal equation, a broader class of Hamilton-Jacobi (HJ) equations, arises in countless applications such as image theory, geoscience, computer vision, path planning, financial risk, etc... Image segmentation [7], shape-from shading problems [16, 13], first-arrival seismic travel-times [14], brain connectivity mapping [15] can be computed with the solutions of the eikonal equation. Methods for solving these equations are complex, require a good knowledge of the numerical computing and parallelization techniques. In this intensive computing era, there is currently a lack of parallel solvers for scientists who want to compute efficiently the solutions of the eikonal equations. A large amount of numerical applications recquires a lot of computing power and take time even on high performance computers. The fast marching method (FMM) is well-known to solve these kind of problems. According to Sethian in its 1999 book [17], it is "the most used nowadays and the optimal way to solve Hamilton-Jacobi equations". However, the method does not take advantage of parallel environment (see section 2). The fast iterative method (FIM) is an interesting alternative to the FMM which offers a better parallel potential.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.170

We investigate this point in this paper and we propose multilevel parallel strategies for the FIM. Our work is involved in the realization of a reusable parallel numerical library for solving HJ equations. Firstly we present briefly mathematical background of the HJ equations, numerical methods to solve them. We present the fast iterative method and its specifications compared to the original fast marching method. Then, we present our coarse-grained and fine-grained parallel strategy for the FIM. Finally we focus on the fine-grained approach implementation and we show data size and parallel scalability.

2 Background

HJ equations take the following form :

$$H(x, u, \nabla u, \nabla^2 u) = 0, x \in \Omega \subset \mathbb{R}^n \tag{1}$$

The Hamiltonian H is a continuous scalar function defined on $\Omega \times \mathbb{R} \times \mathbb{R}^N \times S_n(\mathbb{R})$. $S_n(\mathbb{R})$ denotes the vector space of $n \times n$ symmetric matrices, ∇u is the gradient and $\nabla^2 u$ is the Hessian of u. Considering static HJ equations and given $H(x, \nabla u) = c(x).|\nabla u| - 1$ we get back the eikonal equation. The eikonal equation is widely used to simulate the propagation of a wave-front . The viscosity solution of equation 1 existence and uniqueness are shown in [3, **?**]. We will focus in our paper on equation 2. We describe the first arrival times of a moving interface :

$$\begin{cases} c(x).|\nabla u| & = 1, x \in \Omega \subset \mathbb{R}^n, c(x) > 0 \\ u(x) & = \phi(x) \text{ where } \phi : \Gamma \subset \partial\Omega \to \mathbb{R} \end{cases} \tag{2}$$

where Ω is an open subset of $\mathbb{R}^n$, c a positive speed function, and Φ is a known function describing a surface Γ.

Updating solutions is done with a finite upwind difference scheme to discretize convex Hamiltonians. In this paper, we use the upwind Godunov scheme (below for the two-dimensional case at vertex (i, j)) :

$$\max(D^x_- u_{i,j}, -D^x_+ . u_{i,j}, 0)^2 + \max(D^y_- u_{i,j}, -D^y_+ u_{i,j}, 0)^2 = \frac{1}{c^2_{i,j}} \tag{3}$$

where $D^x_\pm$ and $D^y_\pm$ define four differentiation operations :

$$\begin{cases} D^x_- u_{i,j} = \frac{u_{i,j} - u_{i-1,j}}{h}, & D^x_+ . u = \frac{u_{i+1,j} - u_{i,j}}{h} \\ D^y_- . u = \frac{u_{i,j} - u_{i,j-1}}{h}, & D^y_+ u_{i,j} = \frac{u_{i,j+1} - u_{i,j}}{h} \end{cases} \tag{4}$$

with $(x_i, y_j) = (i.\Delta x, j.\Delta y)$ and $u_{ij} = u(x_i, y_j)$.

During the last decades, several numerical methods have been proposed to solve directly the eikonal equation. We remark two efficient methods which are widely used : fast marching methods (FMM) [17, 18] and fast sweeping methods (FSM) [19, 12]. Sethian's FMM is characterized by tracking the expanding wavefront scheme using a heap data structure. The wavefronts expand in the order of the causality given by the equation and the speed function. Its worst-case complexity is $O(N.log(N))$. The FSM, proposed by Zhao solves the eikonal equation using directional sweeps, within a Gauss-Seidel update scheme. The complexity of this method is also $O(N.log(N))$. The reader can refer to [8] for a computational study between these two methods ; to works on parallel FSM in [20, 5] and on parallel FMM in [2, 9, 1]. Note that our results in section 4 show a better parallel scalability compared to the previous mentionned papers. We focus on FMM-like methods (especially the FIM) in this paper.

The fast marching method

The FMM is closely related to Dijkstra's algorithm which is used to compute the shortest path on a network [6]. The principle is to evaluate the solution values at grid vertices in the order in which the wavefront passes trough the grid vertices. In Sethian's algorithm the FMM, we divide the grid into three regions [18] (fig. 1) : the accepted vertices or frozen vertices (FZ vertices), the narrow band vertices (NB vertices) and the far away vertices (FA vertices).

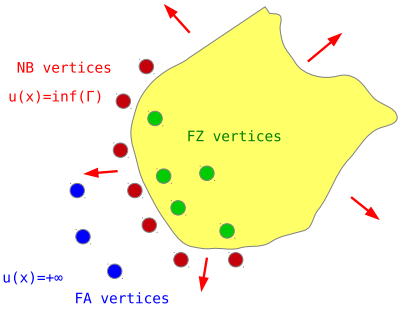

Figure 1: FMM three regions

The accepted vertices have already been reached by the front. Their solution has been computed and the value will not change in the future. The narrow band vertices are composed of the neighbors of the accepted vertices. These are the vertices where the computation actually takes place and they might be updated at the following iterations. Finally, the far away vertices have never been reached by the front yet. We suggest the reader to refer to Sethian work [17, 18] for details about the algorithm. The method is a one-pass method, each point being touched only once. This corresponds to a computational cost of $O(N)$. FMM uses a heap data structure to store the narrow band vertices and computing the minimum value at each iteration makes the global algorithm complexity up to $O(N.log(N))$. The FMM is an efficient method to solve eikonal equation on sequential architectures. However, managing the heap structure is a hindrance for performance causing bottlenecks since the heap has to be updated whenever a new vertex in the narrow band added. The causality principle forbids the simultaneously update of several points at the same time. A basic heap management has a complexity of $O(N(\log(N))$. In our implementation, we use a min heap structure which reduce the cost to $\log(N)$. Even with this improvement, on parallel architectures of p processors, we would have a $\log(N) - \log(p)$ complexity. Given a large scale problem, it would still be insignifiant.

The fast iterative method

Jeong and Whitaker [11] underline the fact that we have to design fast parallel algorithms for solving the eikonal equation on parallel architectures. Therefore, the authors have proposed the Fast Iterative Method (FIM) to solve the eikonal equation efficiently on Graphical Processing Units (GPUs). The FIM is a variant of the FMM which keeps the idea of the narrow band management by using a list called the active list instead of an expensive heap data structure. In this paper, we prefer to refer to the narrow band rather than the active list. The FIM is even faster compared to the sequential FMM and according to the authors, "The FIM algorithm should scale well on various parallel architectures, e.g., multi-core processors, shared memory

multiprocessor machines, or cluster systems". To our knowledge this point has not been verified yet. The FIM is built such that if follows three main points : there is no particular update order ; the FIM uses separate, heterogeneous data structure for sorting and the FIM permits multiple points to be updated simultaneously. The reader can refer to the original paper [10] for more details.

3 Multilevel parallel approach for the FIM

It is interesting to study works on parallel FMM [9, 1, 2] although they do not fit for parallel FIM. In each subsection we analyze different parallel strategies used for the FMM.

Coarse-grained parallelism

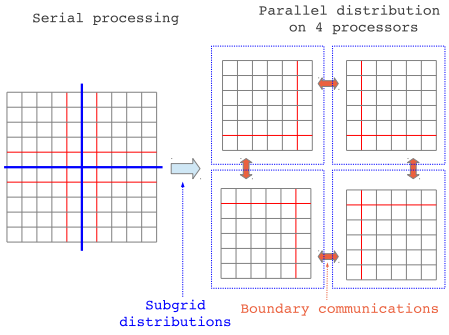

Figure 2: Domain decomposition

Domain decomposition In [9], the author proposed a domain decomposition model for the FMM. The whole computational grid Ω is divided between p processors, giving each processor access to only its own sub-domain Ω_k where k is the process rank, and use message passing strategy to communicate between different processors. Drawbacks of these strategies are that each sub-grids Ω_k has to compute its smallest close value u^k_{min} for the FMM and has to share ghost vertices between other neighboring sub-domains leading to boundary communications (fig. 2). Some sub-domains can also have few work to do, when the narrow band does not contain much vertices. Boundaries synchronizations, rollback operations can reveal to be costly and hard to manage in some situations.

Master-worker model Porting this decomposition on the FIM would give the same drawbacks. We have looked for other solutions in order to manage the narrow band efficiently. An other way to avoid any complex synchronizations is to share a global narrow band among all processes. A dedicated process would manage the narrow band. However, this strategy is not recommended since working processes need either to communicate new narrow band vertices everytime one is added in their subdomain or to communicate local new narrow band array at the end of the iteration to a dedicated process. The first case is obviously subject to huge communications bottlenecks and in the second case, we would have to synchronize the global narrow band with local narrow bands and distribute back new local narrow bands to worker

processes which is costly. We propose a different approach based on a master-worker distribution, using local narrow bands and synchronizing only the narrow band vertices which are in ghost areas to the corresponding neighbor process. The method has the benefits to be reusable since we do not have to change the FIM algorithm main loop. Let p be the number of processes, P_0 the master process and $P_{i\in[1;p-1]}$ a worker process. The grid G is decomposed into $nblocks$ $G_{k\in[0;nblocks]}$ where $nblocks > 2(p-1)$ for load-balancing purposes. Indeed, we have to avoid collective, blocking communications since our problem is dynamic (a subdomain can have few works to do compared to an other one).

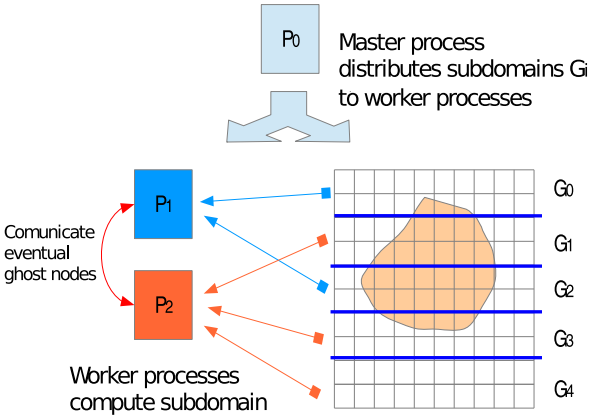

Figure 3: Load-balanced coarse-grained FIM model

The model illustrated on fig. 3 with 3 processors works as the following. The master process P_0 gives some subdomains (G_0 and G_1) to compute to working process. As soon as one working process has finished (computing work is not static) it sends its work back to the master process which give immediatly an other work to compute (G_2 on P_1 and G_3, G_4 on P_2). We remark that subdomain G_4 is given to processor P_2 since P_2 might finish to compute its two first subdomains before P_1 (less narrow band vertices to check). If a new narrow band vertex is a ghost vertex then we have to communicate it to the corresponding neighbor process. We present the distributed FIM model algorithm 1 below for one iteration :

Algorithm 1: Distributed fast iterative method Main loop

mymasterProcess P_0 `master()` P_0 sends subdomains $G_{k\in[1;p-1]}$ to compute to worker process $P_{i\in[1;p-1]}$
while $k < nblocks$ *i.e. there are still subdomains not computed yet* **do**
  P_0 send subdomain G_k to compute to available P_i
  Increment k
myworkerProcesses $P_{i\in[1;p-1]}$
`worker()` P_i computes work G_k
mymainloopMain Loop `Main loop()` of algorithm 2 where NB_i and u_i are local to the process P_i.
if *vertices in* $NewNB_i$ *are in ghost zones* **then**
  P_i sends $NewNB_i$ ghost vertices to corresponding neighbor process $P_{i-1} or P_{i+1}$

This strategy has the benefit to minimize global communications, permitting asynchronous

communications thus giving good parallel scalability.

Fine-grained parallelism

Narrow band partitioning An other method is proposed in [1] and [2] where the decomposition is an adaptive domain-decomposition which is more focused on the narrow band management. The initial front is partitioned at the initialization and then each processor solves a sub-problem independently. However, the method still requires some synchronizations between processors, when some vertices from a sub partition can overlap in other sub partitions for instance. One option is to redivide the narrow band vertices again.

Applying this strategy to the FMM is costly since we generally want to minimize synchronizations the most at a fine-grained level. Changing the algorithm and the method itself can be a good idea in order to be efficient in parallel computing. The active list is more fitted for parallel purpose but still requires a careful partitioning. We cannot compute the vertices in the active list independently following the original algorithm. Jeong and Whitaker has parallelized the method on GPUs [10] using block-based update scheme which can be compared to a domain decomposition. They mention their synchronizations with reduction calls. We propose in the next subsection a different parallel model.

FIM fine-grained implementation First work on fine-graind parallel FIM is available in our paper [4]. We have refined our model since and introduce a modified main-loop (one iteration) algorithm for shared-memory FIM in algorithm (2). We have seen that in [10], the authors propose a parallel method targeting GPUs by managing block of actives lists. We instead use a modified NB/active list partitioning for multi-core processing. To limit data transmissions, we do not use a matrix flags for the status of the grid vertices. Local FIM on particular sub-domain or sub-front use minimal tests on grid vertices values to determine their state. We also manage to make vertices computation independent in every sub active lists. Given an initial front Γ_0 partitioned in $p > 1$ subsets Γ_0^k where k represents the rank of the subset, we have to arrange data that at each iteration t, every Γ_t^i can be solve independently and verify : $\forall i \in [[0; p]], \begin{cases} \bigcup_{k=1}^{p} \Gamma_t^k & = \Gamma_t \\ \bigcap_{k=1}^{p} \Gamma_t^k & = \emptyset \end{cases}$.

Load balancing is a real concern in [1, 2] where the authors have proposed to choose the sets in such a way that the emerging wave fronts ideally cover nearly the same portions of the computational domain. However, this assumes that we know the behavior of the solution. One possible solution is to use a hierarchical domain decomposition scheme such as kd-trees to assign neighboring regions to the initial subsets. It is also mentioned that during simulations where unbalance can become larger (one sub-front can move ahead of another), we can redivide all the narrow band vertices rebalancing the jobs. The authors omitted this option completely as they mentioned since it is costly. In our shared-memory algorithm 2, we have decided to use this option and compensate the potential lack of performance by differing from the algorithm proposed in [2] where every thread manages a local narrow band and solves its own sub-front. We decide in our strategy to share the narrow band NB among every thread avoiding potential synchronization needs and combine sub-fronts. Thus we do not have to split domains at the initialization in such way to obtain a good load-balancing. Load balancing would be done at every iteration. In order to ensure data coherency, if two different threads p_k and p_l want to write on the same value u_{ij}, we put well placed critical section and we ensure that u_{ij} take the minimum value between $u_{ij}^{p_1}$ and $u_{ij}^{p_2}$. This method allows us to limit rollback operations and the use of any hierarchical domain decomposition. Even though critical sections can be

costly we minimize their impacts by well placing them in the lowest level possible. We can see their impacts in the next section 4. This method has the merit to be more direct forward to implement since we do not have to be as much concerned about synchronizations possible issues and offers a dynamic load-balancing.

Algorithm 2: Modified loop iteration for the fine-grained parallel FIM

```
mymainloopFunction fine-grained main loop()
Clear NewNB
while NB ≠ ∅ do
    foreach x ∈ NB in parallel do
        p_outer ← u(x)
        q_outer ←solution of eq. 3 at x
        if |p_outer − q_outer| < ε then
            foreach neighbor x_neighbor of x do
                if x_neighbor ∉ NB then
                    p ← u(x_neighbor)
                    q ← solution of eq. 3 at x_neighbor
                    if q < p then
                        u(x_neighbor) ← q
                        Add x_neighbor to NewNB ;        /* critical section */
        else
            if q_outer < NewU(x) then
                Add x_neighbor to NewNB ;                /* critical section */
            NewU(x) = q_outer
    NB = NewNB
    u = NewU
```

4 Experiments

We propose 3 case tests to illustrate our work running on a 2D cartesian mesh. The first test is a basic test which can be compared with results in [2], composed of one single circle initial front at the center. The second setup is more complex, composed of two circles and an obstacle. The last setup simulates a more dynamic and realistic environment with 32 random seed initial fronts (fig 4).

We are running the tests at HPC@LR Montpellier Resource Center. We use two different shared memory systems. The first one is composed of 2 processors Intel Xeon X5650 at 2.66 GHz with 6 physical cores each. Hyper-threading at HPC@LR is deactivated hence 12 logical cores in total are available. The second system is a SMP node composed of 8 Intel Xeon E7-8860 processors with 10 logical cores each. This multiprocessor computer has then 80 logical cores in total. We use OpenMP as the shared memory multiprocessing programming API. The results obtained in parallel are the same as the sequential simulations with the same number of iterations achieved.

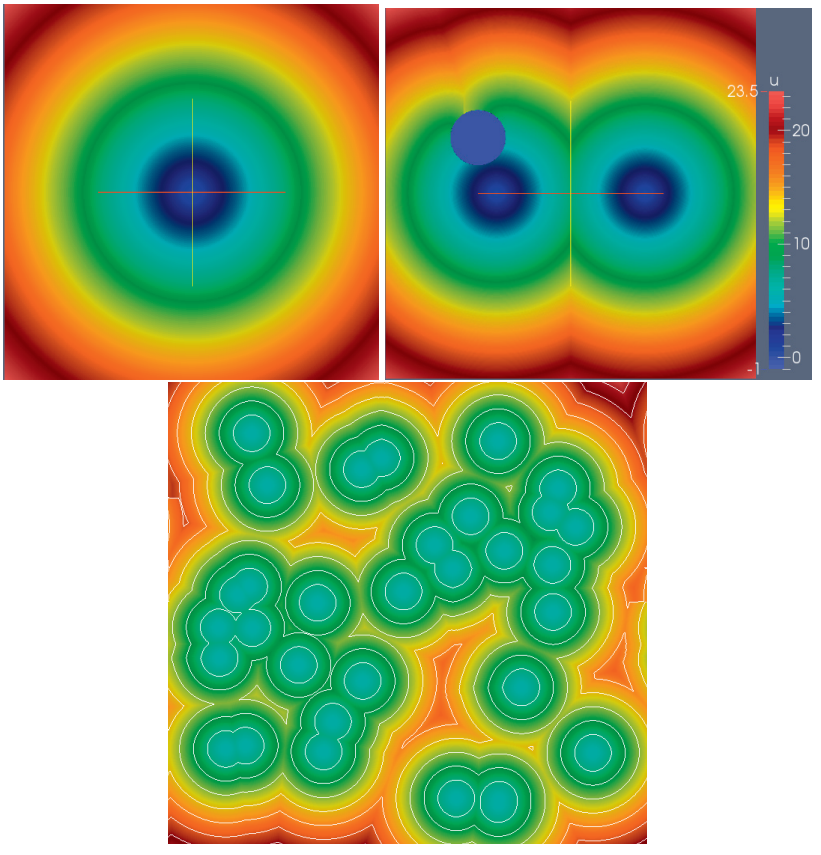

Figure 4: Center test, obstacle test and random test

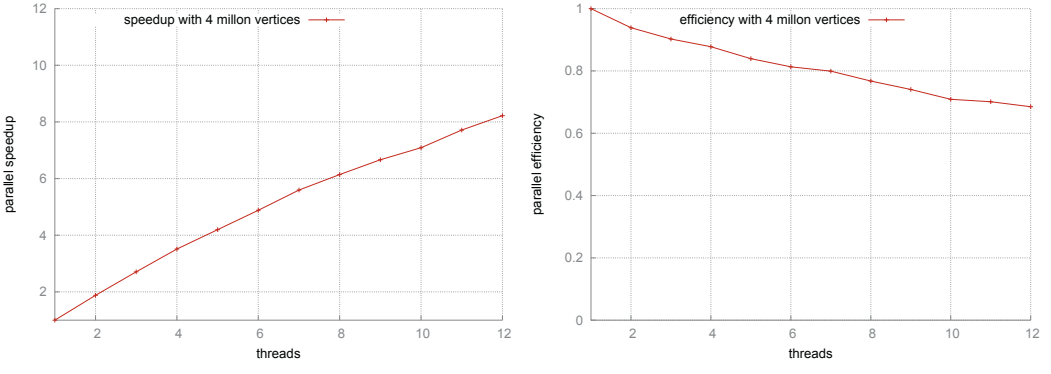

Figure 5: Parallel FIM speedup and efficiency for the center test

On figure 5, we run a first experiment with 4 million vertices at HPC@LR to observe parallel scalability. We remark that without hyper-threading we have a smooth parallel speedup up to 12 threads where the low efficiency loss is due to the critical sections mentioned previously in section 3.

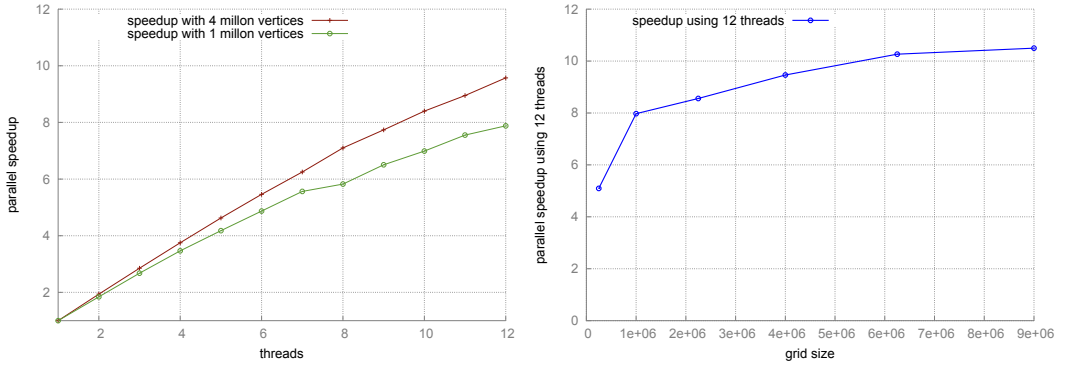

Figure 6: Parallel FIM speedup and datasize scalability the obstacle test

On the second test (fig. 6) the parallel scalability is better compared to the first test. This is due to the narrow band size which is more important (more initial fronts) thus giving a more nteresting speedup. We also monitor the parallel data-size scalability on different problem size. The second graph on figure 6 which represent the data size on axis x and parallel speedup using 12 threads on axis y show that working on larger problem size gives a better parallel speedup. Large scale applications should greatly benefits from this behavior.

We now observe the last test (random seeds) behaviour and we also use the SMP parallel system up to 80 cores for this case (7). In comparison, for the parallel FMM in [2], where this test offers the most interesting results, with 2 million points grid size, the parallel speedup obtained on 16 cores is 8.65 (speedup is 7.2 for 12 cores). For other tests, the parallel speedup is less interesting and stays the same above 8 cores. For the parallel FSM in [5], the parallel speedup reaches for a 4 million points grid size (in 3D) 15 on 30 cores and 17 for a 32 million points grid size. On 12 cores we can observe a speedup around 8.

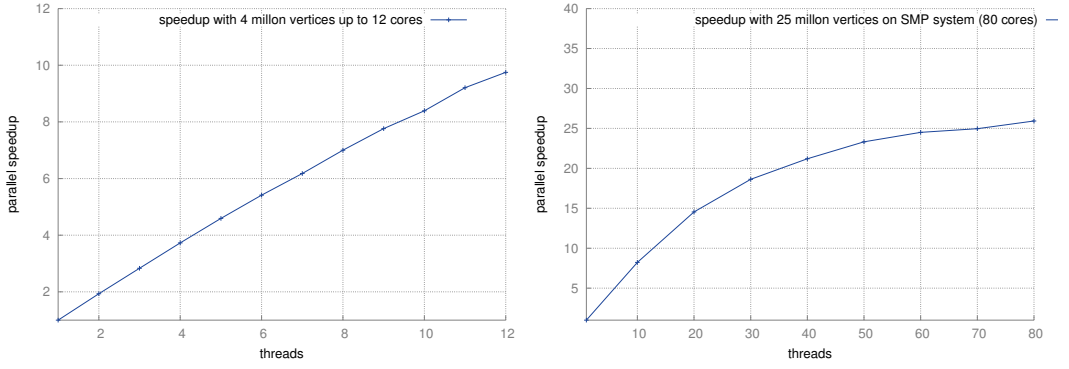

Figure 7: Parallel FIM speedup for the random seeds test on two different parallel systems

In our experiments, the random seeds test also offers a better parallel scability compared

to other case tests, reaching a speedup of 10 when using 12 cores. Indeed, the numerous initial fronts imply a wider narrow band node. The computing flow is then more important. We verify now our parallel algorithm on a larger scale system such as the SMP node composed of 80 cores. We obtain a smooth speedup which should be more interesting for larger scale applications. We do not have the same efficiency due to the fact that eight processors are interconnected together, synchronizing the work at each iteration requires more communications.

5 Conclusion

In this paper we propose two parallel levels for the fast iterative method (FIM) in solving eikonal equations : a coarse-grained approach and a fine-grained approach. The coarse-grained approach is based on subdomain decompositions. Load balancing is assured by a master/worker model and the model can work with asynchronous communications. The fine-grained approach is based on narrow bands partitioning. At every iteration, our algorithm checks the whole narrow band state, divides it and give partitioned narrow bands to different processors. This allows to have a good load- balancing, limit synchronizations issues and we do not have to prepare splitting domains at initialization. Our parallel fine-grained behaviour is closer to the results in the parallel FSM shown in [5] both in parallel scalability and data-size scalability compared to the results in the parallel FMM [2]. Furthermore, the results are generally more interesting, scales relatively well on new multiprocessors system and our parallel strategy fits better in a realistic setup (with several initial fronts).

We have proposed two parallel levels for the recent FIM algorithm. We reinforce the idea that the FIM is particularly fitted for parallel computing and our fine-grained parallel model works great on large scale applications. We are currently implementing the coarse-grained parallel model on distributed systems. Combining these multilevel parallelism would offer interesting hybrid computations which would fit extreme scale computing.

Acknowledgment

The authors would like to thank the Montpellier HPC@LR Center for giving us the opportunity to run our simulations.

References

[1] Michael Breuss, Emiliano Cristiani, Pascal Gwosdek, and Oliver Vogel. A domain-decomposition-free parallelisation of the fast marching method, 2009. Universitat des Saarlandes, preprint.

[2] Michael Breuss, Emiliano Cristiani, Pascal Gwosdek, and Oliver Vogel. An adaptive domain-decomposition technique for parallelization of the fast marching method. *Applied Mathematics and Computation*, 118(1):1–206, 2011.

[3] Michael G. Crandall and Pierre-Louis Lions. Viscosity solutions of Hamilton-Jacobi-Bellman equations. *Transactions of the American Mathematical Society Vol. 277 No.1*, pages 1–42, 1983.

[4] Florian Dang, Nahid Emad, and Alexandre Fender. A fine-grained parallel model for the fast iterative method in solving eikonal equations. In *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2013)*, Paris, France, October 2013.

[5] Miles Detrixhe, Frédéric Gibou, and Chohong Min. A parallel fast sweeping method for the eikonal equation. *Journal of Computational Physics*, 237(0):46–55, March 2013.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269271, 1959.

[7] Nicolas Forcadel, Carole Le Guyader, and Christian Gout. Generalized fast marching method : applications to image segmentation. *Numerical Algorithms Vol.48 Issue 1-3*, pages 189–211, 2008.

[8] Pierre A. Gremaud and Chrospher M. Kuster. Computational study of fast methods for the eikonal equation. *SIAM J. Sci. Comput. 27*, pages 1803–1816, 2006.

[9] M. Herrman. A domain decomposition parallelization of the fast marching method and applications to image segmentation. Annual research briefs, Center for Turbulence Research, Center for Turbulence Research, 2003. Published: Center for Turbulence Research, Annual Research Briefs.

[10] Won-Ki Jeong and Ross T. Whitaker. A fast iterative method for a class of Hamilton-Jacobi equations on parallel systems. Technical Report UUCS-07-010, University of Utah, April 2007.

[11] Won-Ki Jeong and Ross T. Whitaker. A fast iterative method for eikonal equations. *SIAM J. Sci. Comput. Vol.30 No.5*, pages 2512–2534, 2008.

[12] Chiu-Yen Kao, Stanley Osher, and Yen-Hsi Tsai. Fast sweeping methods for static Hamilton-Jacobi-Bellman equations. *SIAM J. Numer. Anal.*, 74(250):2612–2632, 2005.

[13] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds, 95. *Proc. Natl. Acad. Sci. USA*, pages 8431–8438, 1998.

[14] Peter G. Lelièvre, Colin G. Farquharson, and Charles A. Hurich. Computing first-arrival seismic traveltimes on unstructured 3-d tetrahedral grids using the fast marching method. *Geophys. J. Int. (2011) 184*, pages 885–896, 2010.

[15] Emmanuel Prados, Christophe Lenglet, Nicolas Wotawa, Rachid Deriche, Olivier Faugeras, and Stefano Soatto. Control theory and fast marching techniques for brain connectivity mapping. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume*, page 10761083, 2006.

[16] E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM J. Numer. Anal. 29*, pages 867–884, 1992.

[17] J. A. Sethian. Fast marching methods. *SIAM Review Vol. 41 No. 2*, pages 199–235, 1999.

[18] J. A. Sethian. *Level Set Methods and Fast Marching Method*. Cambridge University Press, 1999.

[19] Hongkai Zhao. A fast sweeping method for eikonal equations. *Mathematics of Computation Vol.74 No.250*, pages 603–627, 2005.

[20] Hongkai Zhao. Parallel implementations of the fast sweeping method. *Journal of Computational Mathematics*, 25(4):421–429, 2007.