Exploiting intensive multithreading for the efficient simulation of 3D seismic wave propagation

Fabrice Dupros, Hideo Aochi, Ariane Ducellier
BRGM
BP 6009, 45060 Orléans Cedex 2, France
(f.dupros, h.aochi, a.ducellier)@brgm.fr

Dimitri Komatitsch
Université de Pau, CNRS and
INRIA Sud-Ouest Magique-3D,
Laboratoire de Modélisation et d'Imagerie en Géosciences,
Pau, France.
Also at Institut universitaire de France, Paris, France.
dimitri.komatitsch@univ-pau.fr

Jean Roman
ScAlApplix Project
INRIA Bordeaux Sud-Ouest
LaBRI UMR 5800
University of Bordeaux 1
33405 Talence Cedex, France
roman@labri.fr

Abstract

Parallel computing is widely used for large scale three-dimensional simulation of seismic wave propagation. One particularity of most of these simulations is to consider a finite computing domain whereas the physical problem is unbounded. Additional numerical conditions are then required to absorb the energy at the artificial boundaries, which introduces a different formulation and a load-imbalance. In the context of finite difference method, we study the use of thread overloading approach to alleviate the imbalance. We introduce a mixed-hybrid parallel implementation based on a classical cartesian partitioning at the MPI level and a self-scheduling algorithm at the thread level to handle more than 700 threads on 8 processors. We demonstrate the efficiency of our methodology on an example of regional modeling performed on 80 processors.

1. Introduction

One major goal of strong motion seismology is the estimation of damage in future earthquake scenarios. Recent advances in high-performance computing technologies make realistic simulation of seismic wave propagation feasible on a regional scale at relatively high frequencies. Different numerical methods have been successfully developed to model the propagation of seismic waves, for instance finite-element methods [1], spectral and pseudo-spectral method [2], or spectral-elements methods [3]. One of the most widely-used technique is the finite-difference method (FDM) because of its simplicity and numerical efficiency [4]. A review of finite-difference methods for the seismic wave equation can be found for instance in [5].

Another requirement in the case of unbounded physical domain is the definition of a finite computational domain with artificial conditions to absorb the outgoing energy. Several approaches are available. We select the Convolutional Perfectly Matched Layer (CPML) [6] which is an improvement of the PML method described for example in [7] for the elastodynamics equation. A common feature of these formulations is to introduce different computational loads in the computing domain, especially at the lateral and bottom edges of the three-dimensional geometry.

Concerning parallel computing, Message Passing Interface (MPI) has become a successful parallel programming environment for distributed and shared memory architectures such as clusters of SMP nodes, mainly due to its simplicity and portability. In this context, mapping of structured computational grids on a set of processors is generally performed by defining a cartesian grid of processors. In our case, this simple partitioning leads to poor scalability. Due to the absorbing boundary formulation, this naive decomposition leads to subdomains with a different CPU load and inhibits load balancing. Moreover, a flat MPI model increases the number of MPI processes that are really necessary to exploit modern architectures composed of large interconnected shared memory nodes and generates an important number of potentially useless communications.

We therefore introduce a hierarchical partitioning approach

using a hybrid programming model and we rely on the properties of the thread library to provide an efficient load-balancing mechanism. We develop a methodology to enhance the performance level of the parallel finite-difference simulation of three-dimensional seismic wave propagation. An algorithm based on hybrid programming and domain overloading is used to reduce load imbalance. This approach is rather general and could be implemented to tackle parallel finite-difference problem using CPU consuming boundary conditions.

2. Numerical modeling of seismic wave propagation

The seismic wave equation in the case of an elastic material is:

$$\rho \frac{\partial v_i}{\partial t} = \frac{\partial \sigma_{ij}}{\partial j} + F_i \tag{1}$$

and the constitutive relation in the case of a isotropic medium is:

$$\frac{\partial \sigma_{ij}}{\partial t} = \lambda \delta_{ij} \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \right) + \mu \left(\frac{\partial v_i}{\partial j} + \frac{\partial v_j}{\partial i} \right) \tag{2}$$

where indices i, j, k represent a component of a vector or tensor field in cartesian coordinates (x, y, z), v_i and σ_{ij} represent the velocity and stress field respectively, and F_i denotes an external source force. ρ is the material density and λ and μ are the elastic coefficients known as Lamé parameters.
A time derivative is denoted by $\frac{\partial}{\partial t}$ and a spatial derivative with respect to the i-th direction is represented by $\frac{\partial}{\partial i}$. The Kronecker symbol δ_{ij} is equal to 1 if $i = j$ and zero otherwise. At the lateral and bottom sides of the model, we add a CPML layer to absorb the outgoing energy. A fixed size of ten grid points is chosen for the thickness of this layer. A scheme that is fourth-order accurate in space and second-order in time is used to discretize the equations. We use the classical staggered-grid represented in Figure 1.

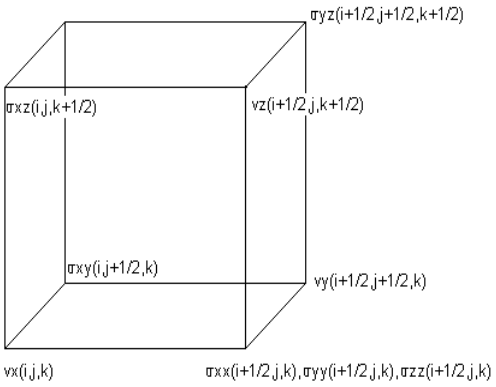

Figure 1. Elementary grid cell of the 3D staggered spatial finite-difference method classicaly used to discretize the equations of elastodynamics.

3 Parallel execution model

3.1 Cartesian-grid based decomposition using MPI

The parallelization strategy we use is based on data parallelism in which each processor solves its own subdomain problem. Each processor updates the wave field within its portion of grid and exchanges informations with its neighbors on common edges. The algorithm can be described as follows.

- A pre-processing phase, including allocation of memory necessary at the node level: Each MPI process is responsible for the initialization of physical variables corresponding to the relevant subset of computational domains and for the allocation of extra communication buffers.
- A time-dependent computational phase corresponding to the resolution of the first-order system of equations (1) and (2) : At each time step, the stress variables are computed first, then each domain exchanges interface information with its neighbors and then the velocity variables are updated with again an exchange phase to update common edges. The free surface or absorbing boundary conditions are implemented at this stage.

The processors at the top of the grid apply a free-surface boundary condition while the boundary subdomains in contact with the other edge apply absorbing conditions. Because of the high-order finite-difference stencil used to compute the spatial derivatives, an exchange of field values at grid points located on the edges is necessary between subdomains. The fourth-order operator we implement involves exchanging information from two grid points. Figure 2 represents the computational domain using different colours for the three regions. The implementation of the free surface condition is not an intensive computing phase, contrary to the physical domain and the bottom CPML layer. To take advantage of the natural horizontal symmetry in the geometry of the domain, a two-dimensional static decomposition is used to avoid additional imbalance in the computation owing to vertical partitioning.

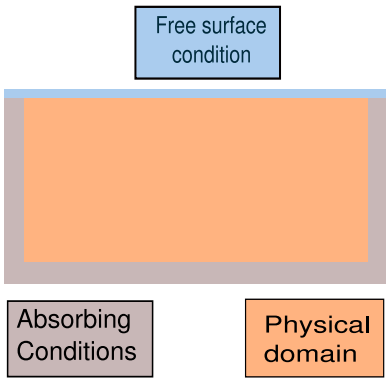

Figure 2. Lateral view of the three-dimensional computational domain with three different regions in terms of numerical formulation.

3.2 Limitations due to static and quasi-static partitioning

To illustrate the load imbalance in the context of static grid-based partitioning, we select three different discretizations based on grids having a different size (i.e, increasingly higher resolution).We consider 3D regional geological models, part of the French Riviera. The smallest grid is composed of 27 million grid points (330 x 330 x 255), the intermediate-sized grid contains 81 million grid points (500 x 500 x 325) and the largest model contains 780 million grid points (1000 x 1200 x 650). The CPU-cost ratio observed between a boundary grid point and a physical domain point varies from two to four. We therefore choose a ratio of three as an arbitrary average value for our theoretical representation in Figure 3.

One common remark is the increase of load imbalance with the number of processors. The largest test case exhibits an imbalance close to 50% on $O(1000)$ processors. Even moderate scale models could be inefficient with an average of 40% of imbalance for medium-sized model on 80 processors.

One classical way of ensuring load balancing in the context of grid-based computation is the use of mesh partitioning techniques [8] for initial distribution or dynamic re-distribution during the computation. Several redistribution toolkits have been successfully developed for MPI simulations, for example in the case of finite-element calculations and adaptive mesh refinement [9]. However, concerning finite-difference modeling, the use of such approaches changes the shape of the regular interface between distributed subdomains (north-south-east-west for two-dimensional decomposition) and therefore leads to far more complex message-passing schemes.

Considering two-dimensional partitioning in the horizontal directions of a 3D domain, one could suggest a quasi-static load-balancing algorithm based on zone costs. The idea would be to determine, from the numerical formulation or from runtime measurements, the suitable weights to shift forward or backward the boundaries of each MPI subdomain and minimize imbalance. This methodology has been applied in [10] with a moderate number of processors in the case of electromagnetic problems to address the PML overhead. Although this method appears attractive, a couple of limitations should be underlined. First of all, considering only the flops ratio provides incomplete hints, for example the number of components which need to be absorbed can vary from one to three in the boundary conditions formulation depending on the location of the point on the edge. This numerical consideration makes the imbalance quite irregular in the whole computational domain. On top of that, it is very difficult to evaluate a priori the execution time of various parts of a program accurately because of cache effects, arithmetic considerations or compiler behavior that can lead to unpredictable situations for such a fine grain computation.

Finally, shifting forward or backward the boundaries prevents from using a large number of processors. In fact, the minimum size authorized for any subdomain, in terms of grid point number per direction, is reached faster. The consequence is a very fine grain of computation and reduced opportunities for overlapping.

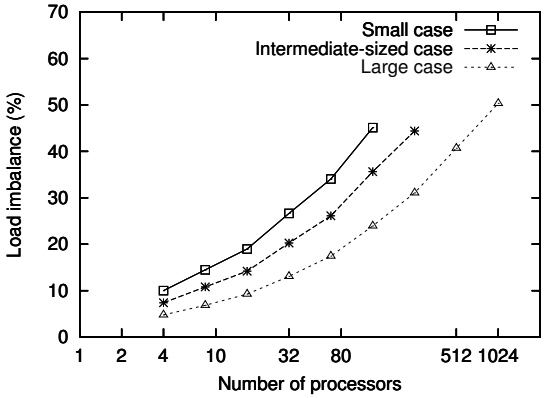

Figure 3. Theoretical load imbalance for a finite-difference parallel simulation and impact of CPU-costly boundary conditions. We have considered three different grid sizes.

3.3 Mixed-hybrid parallel approaches

Several references (e.g. [11], [12]) report implementations of a hybrid programming strategy for large-scale numerical simulations. One of the main objectives is to save memory consumption or communication volume at the shared-memory level, but also to introduce load balancing for complex parallel simulations. On the other hand, hybrid programming is considered as rather difficult to control in terms of performance prediction.
If we consider mixed MPI and OpenMP programming, the performance could be lowered by the implementation strategy of these standards [13]. Several aspects can be underlined such as the lack of support for distributed allocation of shared data or the difficulties to control the threads created by the OpenMP environment, especially in the context of non uniform memory architectures (NUMA).
MPI combined with POSIX threads programming is another way of handling a second level of parallelism. Unfortunately, kernel threads are typically preemptive with regular flush of cache memory. The use of a thread overloading strategy could be very inefficient, especially for cache-friendly algorithm.
One could also consider parallel programming environments described in [14] that provide dynamic load balancing for MPI applications and thread migration. Adaptive-MPI (AMPI) allows the straightforward conversion from legacy MPI applications to the AMPI environment, which is an important feature for programmers. The idea is to subdivide the problem into several smaller partitions with dynamic mapping over the set of physical processors. This is done using runtime measurements of computational loads and communication patterns. AMPI converts MPI processes into user-level threads with few possibilities of saving extra communication buffers on a shared-memory node. The decomposition also generates a number of communications between subdomains which represents a potential overhead.

4 Dynamic approach based on self-scheduling and thread overloading

4.1 Thread library

We choose the PM2 software and the Marcel POSIX-compliant thread library [15] to benefit from performance of thread management and several possibilities of scheduling policies. Marcel is a two-level thread library: it binds one kernel-level thread on each processor and then performs fast user-level context switches between user-level threads, hence getting complete control over thread scheduling in user space without any further help from the kernel. These features are important for the efficiency of our algorithm. We choose to use simple self-scheduling with non-preemptive thread execution. Using a single list of ready threads from which the scheduler simply picks up the next thread to be scheduled, the workload is automatically balanced between processors at the shared-memory level.

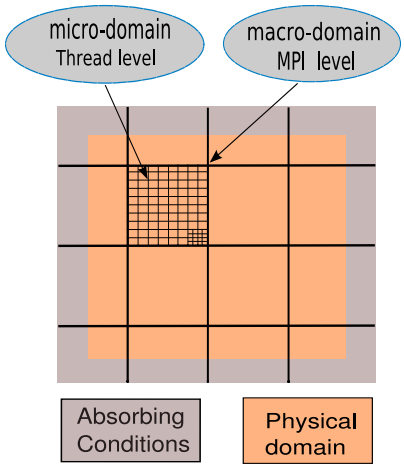

Figure 4. Top view - Hybrid partitioning of our 3D computational domain. Each MPI macro-domain creates a hierarchy of micro-domains.

4.2 Dynamic load-balancing algorithm

4.2.1 Hierarchical partitioning

A second level of parallelism is introduced by dividing each MPI domain into a grid of micro-domains. This new level of parallelism is designed to balance the computations inside a multiprocessor node and makes the global behavior of the code rely on macro-domain decomposition. This approach seems natural with the increasing power available inside current single shared-memory nodes. Multi-core or NUMA are the basic technologies on which large computing nodes, with a increasing number of computing units accessing the same memory space, are based. The idea is to benefit from this new flexibility and to define a moderate number of macro-domains in order to have a limited load imbalance.
Figure 4 illustrates the hierarchical partitioning used with MPI macro-domains and thread-level micro-domains. For example, geometrical considerations prevent an optimized static decomposition at the corner of the domain. Using virtualization at the node level, we define more threads than computing resources and use a global runqueue to ensure load balancing. Each macro-domain is divided independently by applying a first level of decomposition to get a

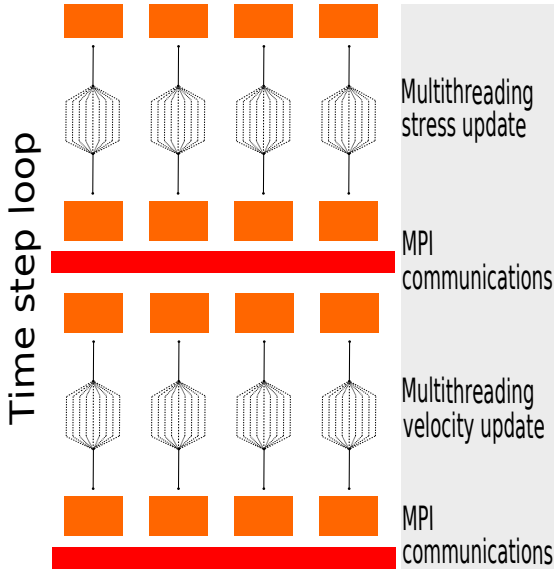

Figure 5. Two-level parallelism with four MPI processes. Each MPI task creates a pool of threads for the update of velocity and stress unknowns.

number of micro-domains greater than the number of virtual processors. Another level of decomposition is then applied to the lightest micro-domains. This hierarchy is finally used at the scheduling level. The definition of micro-domains could be inefficient without careful thread scheduling and meaningful information coming from the application. During the pre-processing phase, a weight is affected to each micro-domain, which allows the ranking of threads in different classes depending on their load and their level in the hierarchy. The fine grain of this parallelism level and the self-scheduling algorithm lead to natural load balancing, assuming for example scheduling of heavy threads in first position and the use of light threads as an adjustment strategy.

4.2.2 Thread management

The main differences with the classical 2D algorithm occur during the computational phase. At each time step, we create a huge number of threads for the parallel updates of the stress and velocity unknowns. These threads share data at the MPI process level as illustrated in Figure 5 for four MPI processes on four computing nodes. Each node is composed of a defined number of CPUs and a single MPI process is responsible for micro-domain intra-node creation and macro-domain inter-node communications. Typically we can multiply the number of physical processors by 100 to define the number of threads in the pool of threads without significant overhead in terms of thread context switching. These threads are destroyed before the communication phase. This strategy aims to maximum portability for the software.
Keeping only the main thread alive inside the MPI process during a communication phase makes the code independent of the thread-safe aspect of the MPI implementation. Nevertheless, other optimized approaches could be explored by considering together communication and computation threads and take them into account at the scheduler level to enhance overlap possibilities. Another direction could be to make these pools of threads blocked on a condition variable.

4.2.3 Computational efficiency

We can also notice that this hybrid approach maintains large MPI macro-domains and facilitates the overlapping of communications by computations. This strategy is widely used and very efficient for static grid-based decomposition and allows the use of persistent MPI communications with overlapping. Using a 2D decomposition makes the computation/communication ratio of the order of $\frac{V}{\sqrt{P}}$ where V is the volume of the subdomain and P the number of computing units. The hybrid computation approach increases the size of the MPI domain and decreases the number of MPI processes, making the previous ratio better in terms of overlapping. Different directions of partitioning could be selected at the thread level because they are independent of the macro-domain decomposition. However, cache behavior needs to be taken into account. Most arrays allocated in the code are 3D arrays. In order to avoid the generation of extra cache misses by cutting the local z direction, it is important in a C programming to keep that direction local with respect to the unit stride. This strategy is consistent with remarks in a previous section on the vertical direction. It is also a good way for maximizing the prefetching. We complete our hierarchical algorithm with local blocking as described in [16] in order to maximize performance and take full advantages of the size of the micro-domains. The gain obtained can be important for certain classes of stencils but the huge number of variables per grid point prevents us from getting very good performance in the context of the elastodynamics equation. Typically we did not gain more than 3 or 4% in practice with this blocking strategy.

5 Results and analysis

5.1 Description of the experimental platform

Our experimental platform is a cluster of ten IBM x3755 nodes. Each node is a quadri dual-core 2.6 Ghz AMD

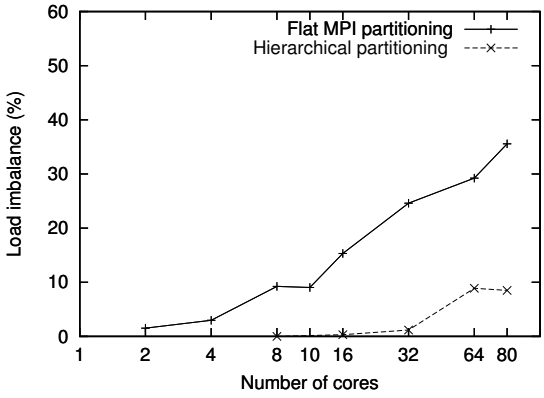

Figure 6. Load balancing comparison between a pure MPI approach and a mixed approach: the hybrid methodology is clearly superior.

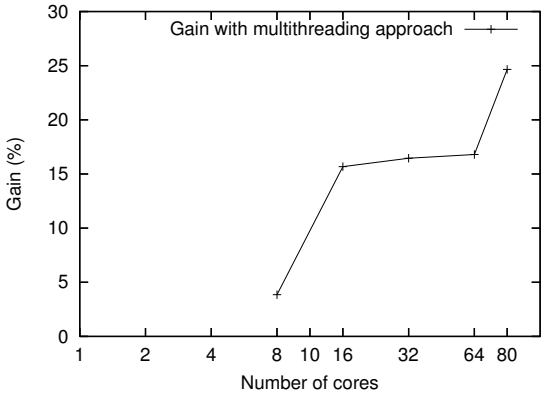

Figure 7. Gain in terms of elapsed time for the French Riviera example on 80 CPUs. The comparison with the pure MPI approach shows a maximum 25 % reduction of computation time.

Opteron processor with 32 GB of memory and 2 Mb of L2 cache for each processor. The nodes are interconnected with a Myrinet network. We use 80 cores to simulate a total of 6000 time steps.

5.2 Numerical experiments

We consider the intermediate-sized model described in section 3.2. The first concern is the behavior of the imbalance with our multithreaded programming methodology. Figure 6 shows the maximum imbalance between MPI subdomains and its evolution with the number of cores. As reported in the theoretical model of Figure 3, the imbalance increases significantly with the number of cores, reaching a maximum of 36% in the case of pure MPI partitioning. The experiments are close to this result but it is interesting to underline the difference between theory and experiments in the case of 8 cores for example: the experimental result (2.98%) is very far from the theoretical imbalance (7.05%). This is a good illustration of the difficulty to obtain realistic and useful static information to balance the computations. Inside each MPI process, we use 772 micro-domains and the same number of threads on 8 cores with no significant overhead in terms of management and scheduling. This demonstrates the efficiency of the Marcel thread library and makes intensive multithreading a powerful tool for computing purpose. For our implementation, the non-preemptive behavior of Marcel threads is also an advantage to obtain potential cache effects.
Figure 6 illustrates the interest of the hybrid approach over pure MPI partitioning, the imbalance is drastically reduced even for a moderate scale configuration. Using 80 cores we decrease the imbalance from 36% to 8%.

Figure 7 represents the elapsed time gain based on hybrid approach over MPI decomposition. This demonstrates the relation between load imbalance and parallel performance of the code. Each time one can decrease the imbalance, the MPI waiting idle time is also reduced and the global performance of the code is significantly enhanced. As the difference between MPI and hybrid decomposition increases, the elapsed time gain also increases: for example reducing the imbalance from 36% to 8% leads to a gain of 25% on the simulation time.

One of the main aspects of the hybrid methodology is to rely on the macro-domain imbalance behavior. The first requirement is to get nearly perfect micro-domain balance at the shared-memory node level. In our experiments, the maximum imbalance between virtual processors is 2% with 772 threads. That means that our strategy is well suited for geometry with different CPU-cost regions like the macro-domains at the corners of the three-dimensional domain. This class of macro-domains does not exhibit any horizontal symmetry and it is therefore difficult to balance computation using a grid-based decomposition. As the number of micro-domains increases, results improve until reaching the arithmetic limits of the decomposition of a subdomain by an increasing number of threads or the potential bottleneck coming from thread management overhead. Figure 8 shows a comparison of imbalance evolution with respect to the number of MPI processes. As explained in the algorithm description, the hybrid partitioning closely mimics the behavior of MPI partitioning at the macro-domain level and relies on the quality of this decomposition. Because we use a moderate number of macro-domains this imbalance remains very low. In case of a larger number of

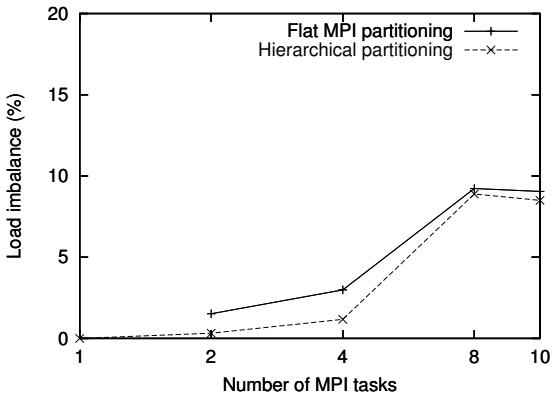

Figure 8. Comparison of pure MPI and hybrid partitioning behaviour in terms of load-balancing at the macro-domain level.

macro-domains, shifting forward or backward the computational domain boundaries could become attractive because it would lower the importance of having good static information.

6 Conclusions and future work

We have considered the load balancing problem in the context of 3D finite-difference modeling of seismic wave propagation. We have demonstrated the efficiency of a hierarchical partitioning coupled with a mixed-hybrid parallel programming approach and the use of a relevant thread model. Using a dedicated cost model to identify the load imbalance based on the numerical formulation used to solve the elastic wave equation, we have pointed out the poor performance of classical flat MPI partitioning for this class of problems. A two-level partitioning algorithm has been introduced, with MPI processes used at the higher level. These sets of macro-domains drive the global load-balance of the code and allow good communication/computation overlapping. At the lower level, we defined micro-domains using an overloading strategy. Using more threads than computing units is not only a way of benefiting from potential cache effects but is also useful to address the load-balancing problem at the node level. For a 3D test problem, we have obtained a gain of 25% in terms of elapsed time.
In future work, the hierarchical partitioning described here could be a good candidate to express memory affinity between macro-domains and/or micro-domains. Getting the maximum performance from hierarchical architectures by minimizing non local memory access is a difficult research subject and requires careful scheduling of threads with respect to affinity. In [17], the idea of controlling scheduling with bubbles has been introduced; it could be applied to our algorithm, considering the natural hierarchy between sub-domains in our partitioning. This NUMA-aware algorithm could be completed by hierarchical work stealing.
Another idea could be to use the thread-overloading methodology to implement a cache-efficient algorithm. Recent advances in the implementation of stencil computations [18] have demonstrated important speedup coming from a suitable organization and partitioning of the space-time domain. The first-order system of equations we use to model seismic wave propagation is well adapted to this particular implementation of grid-based computations.

Acknowledgments

Research supported by the French ANR under grant NUMASIS ANR-05-CIGC. The authors thank Jean-François Méhaut from LIG, France and Alexandre Carissimi from UFRGS, Brazil for discussions on thread overloading.

References

[1] J. Lysmer and L. A. Drake, "A finite element method for seismology," in *Methods in Computational Physics*, B. Alder, S. Fernbach, and B. A. Bolt, Eds. New York, USA: Academic Press, 1972, vol. 11, ch. 6, pp. 181–216.

[2] E. Tessmer and D. Kosloff, "3-D elastic modeling with surface topography by a Chebyshev spectral method," *Geophysics*, vol. 59, no. 3, pp. 464–473, 1994.

[3] D. Komatitsch and J. Tromp, "Introduction to the spectral-element method for 3-D seismic wave propagation," *Geophys. J. Int.*, vol. 139, no. 3, pp. 806–822, 1999.

[4] J. Virieux, "*P-SV* wave propagation in heterogeneous media: velocity-stress finite-difference method," *Geophysics*, vol. 51, pp. 889–901, 1986.

[5] P. Moczo, J. O. A. Robertsson, and L. Eisner, "Advances in wave propagation in heterogeneous earth, the finite-difference time-domain method for modelling of seismic wave propagation," *Advances in Geophysics*, vol. 48, pp. 421–516, 2007.

[6] D. Komatitsch and R. Martin, "An unsplit convolutional Perfectly Matched Layer improved at grazing incidence for the seismic wave equation," *Geophysics*, vol. 72, no. 5, pp. SM155–SM167, 2007.

[7] F. Collino and C. Tsogka, "Application of the PML absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media," *Geophysics*, vol. 66, no. 1, pp. 294–307, 2001.

[8] K. Schloegel, G. Karypis, and V. Kumar, "A unified algorithm for load-balancing adaptive scientific simulations," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing available on CDROM, November 04-10, Dallas, Texas, USA*, p. 59.

[9] C. D. Norton, J. Z. Lou, and T. A. Cwik, "Status and Directions for the PYRAMID Parallel Unstructured AMR Library," in *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS), San Francisco, CA, USA April 23-27, 2001*, p. 120.

[10] S. Seguin, M. Cracraft, and J. Drewniak, "Static and quasi-dynamic load balancing in parallel FDTD codes for signal integrity, power integrity, and packaging applications,," in *2004 IEEE International Symposium on Electromagnetic Compatibility, August 09-13, Santa Clara, CA, USA*, pp. 107–112.

[11] P. Hénon, P. Ramet, and J. Roman, "On using an hybrid MPI-Thread programming for the implementation of a parallel sparse direct solver on a network of SMP nodes," in *Proceedings of Sixth International Conference on Parallel Processing and Applied Mathematics, Workshop HPC Linear Algebra, Poznan, Pologne, LNCS 3911*, Sep. 2005, pp. 1050–1057.

[12] J. Corbalán, A. Duran, and J. Labarta, "Dynamic load balancing of MPI+OpenMP applications," in *33rd International Conference on Parallel Processing (ICPP 2004), 15-18 August 2004, Montreal, Quebec, Canada*, pp. 195–202.

[13] L. Adhianto and B. M. Chapman, "Performance modeling of communication and computation in hybrid MPI and OpenMP applications," in *12th International Conference on Parallel and Distributed Systems (ICPADS 2006), 12-15 July 2006, Minneapolis, Minnesota, USA*, pp. 3–8.

[14] M. A. Bhandarkar, L. V. Kalé, E. de Sturler, and J. Hoeflinger, "Adaptive Load Balancing for MPI Programs," in *Computational Science - ICCS 2001, International Conference, San Francisco, CA, USA, May 28-30, 2001*, pp. 108–117.

[15] R. Namyst and J.-F. Méhaut, "PM2: Parallel multithreaded machine. a multithreaded environment on top of PVM," in *Proc. 2nd Euro PVM Users' Group Meeting*, Lyon, France, Sep. 1995, pp. 179–184.

[16] G. Rivera and C.-W. Tseng, "Tiling optimizations for 3D scientific computations," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing available on CDROM, November 04-10, Dallas, Texas, USA*, p. 32.

[17] S. Thibault, R. Namyst, and P.-A. Wacrenier, "Building portable thread schedulers for hierarchical multiprocessors: The bubblesched framework," in *Euro-Par 2007, Parallel Processing, 13th International Euro-Par, Rennes, France*, pp. 42–51.

[18] S. Kamil, K. Datta, S. Williams, L. Oliker, J. Shalf, and K. Yelick, "Implicit and explicit optimizations for stencil computations," in *Proceedings of the 2006 workshop on Memory System Performance and Correctness, October 11, San Jose, California, USA*, pp. 51–60.