On scalability issues of the elastodynamics equations on multicore platforms

Fabrice Dupros, Hiep-Thuan Do, Hideo Aochi

Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1226 – 1234

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

On scalability issues of the elastodynamics equations on multicore platforms

Fabrice Dupros[a,*], Hiep-Thuan Do[a], Hideo Aochi[a]

[a]*BRGM, BP 6009, 45060 Orléans Cedex 2, France*

Abstract

Understanding the wave propagation with respect to the structure of the Earth lies at the heart of many analysis both in the oil and gas industry and for quantitative seismic hazard assessment. One of the most widely used techniques to solve the elastodynamics equation is the finite difference method because of its simplicity and numerical efficiency. In the last two decades, the parallel efficiency of this numerical method has been demonstrated through many applications on various parallel platforms. The complexity induced by multicore platforms both in terms of fine-grained parallelism and considering the memory hierarchy justifies revisiting these conclusions. In this paper, we underline the impact of such platforms on standard implementations.

Keywords:
Elastodynamics; Finite Difference Method; Multicore; NUMA

1. Introduction

Owing to its simplicity and numerical efficiency, the finite difference method [1, 2] is one of the most popular techniques to solve the elastodynamics equations and to simulate the propagation of seismic waves (a review can be found for instance in [3]). Most of the parallel implementations for x86 cores is based on a distributed memory assumption and rely on the MPI library for coarse-grained parallelism with a possible use of OpenMP directives to distribute the triple nested loops [4, 5, 6, 7]. The overall parallel methodology is then based on a classical Cartesian grid partitioning with exchange of interface information on common edges. In spite of the good speedups usually reported, the performance levels obtained with standard implementations on multicore nodes remain poor and far from the peak performance. This is mainly due to the trend at the shared-memory level that is characterized by an increase in the number of cores and a slow improvement of the memory bandwidth.

To overcome this limitation, chip designers have introduced a hierarchy of memories for multiprocessor computing nodes, based for instance on Intel QuickPath Interconnect (QPI) or AMD HyperTransport technology to connect multicore processors. Most of the computing nodes available could therefore be considered as Non-Uniform Memory Access (NUMA) platforms. In this paper, we evaluate the impact of multicore architectures on standard implementation of the elastodynamics. We mainly focus on the issues coming from the increasing number of cores leading to a degradation of the load-balancing. We also underline the impact of the programming model on the communications ratio. In the last part, we underline the impact on the non-uniform memory access on the finite difference algorithm that is known to be memory-bound.

*Corresponding author. Tel.: +33-238644676 ; fax: +33-238644676
E-mail address: f.dupros@brgm.fr.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.289

2. Seismic wave modeling

2.1. Governing equations and numerical scheme

The seismic wave equation in the case of an elastic linear material is given in three dimensions by

$$\begin{cases} \rho \frac{\partial}{\partial t} v_x &= \frac{\partial}{\partial x}\sigma_{xx} + \frac{\partial}{\partial y}\sigma_{xy} + \frac{\partial}{\partial z}\sigma_{xz} + f_x \\ \rho \frac{\partial}{\partial t} v_y &= \frac{\partial}{\partial x}\sigma_{yx} + \frac{\partial}{\partial y}\sigma_{yy} + \frac{\partial}{\partial z}\sigma_{yz} + f_y \\ \rho \frac{\partial}{\partial t} v_z &= \frac{\partial}{\partial x}\sigma_{zx} + \frac{\partial}{\partial y}\sigma_{zy} + \frac{\partial}{\partial z}\sigma_{zz} + f_z \end{cases} \tag{1}$$

$$\begin{cases} \frac{\partial}{\partial t}\sigma_{xx} &= \lambda\left(\frac{\partial}{\partial x}v_x + \frac{\partial}{\partial y}v_y + \frac{\partial}{\partial z}v_z\right) + 2\mu\frac{\partial}{\partial x}v_x \\ \frac{\partial}{\partial t}\sigma_{yy} &= \lambda\left(\frac{\partial}{\partial x}v_x + \frac{\partial}{\partial y}v_y + \frac{\partial}{\partial z}v_z\right) + 2\mu\frac{\partial}{\partial y}v_y \\ \frac{\partial}{\partial t}\sigma_{zz} &= \lambda\left(\frac{\partial}{\partial x}v_x + \frac{\partial}{\partial y}v_y + \frac{\partial}{\partial z}v_z\right) + 2\mu\frac{\partial}{\partial z}v_z \\ \frac{\partial}{\partial t}\sigma_{xy} &= \mu\left(\frac{\partial}{\partial y}v_x + \frac{\partial}{\partial x}v_y\right) \\ \frac{\partial}{\partial t}\sigma_{xz} &= \mu\left(\frac{\partial}{\partial z}v_x + \frac{\partial}{\partial x}v_z\right) \\ \frac{\partial}{\partial t}\sigma_{yz} &= \mu\left(\frac{\partial}{\partial z}v_y + \frac{\partial}{\partial y}v_z\right). \end{cases} \tag{2}$$

v and σ represent the velocity and stress field respectively and f denotes a known external source force. ρ is the material density, λ and μ are the elastic coefficients known as Lamé parameters. We consider an isotropic medium leading to a symetric stress tensor.

The dominant numerical scheme to solve the above equations is certainly the explicit finite-difference method. It has been introduced in [2] for a second-order spatial approximation and has been extended in [8] to consider a fourth-order accurate stencil in space and a second-order stencil in time. One of the key features of this scheme is the introduction of a staggered-grid [1] for the discretization of the seismic wave equation.

For classical collocated methods over a regular Cartesian grid, all the unknowns are evaluated at the same

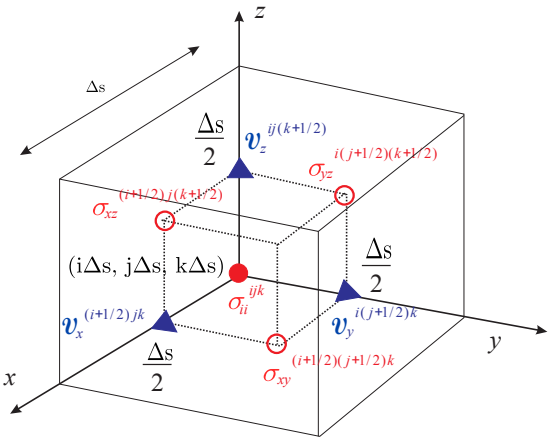

Fig. 1. Elementary 3D cell of the staggered grid and distribution of the stress (σ) and the velocity (v) components.

location, whereas the staggered grid leads to a shift of the derivatives by half a grid cell. The use of a staggered grid improves the overall quality of the scheme (in terms of numerical dissipation and stability), especially in the case of strong material heterogeneity. Figure 1 shows the elementary 3D cell and the distribution of the stress and velocity components. Exponents i, j, k indicate the spatial direction with ($\sigma^{ijk} = \sigma(i\Delta s, j\Delta s, k\Delta s)$, Δs corresponds to the space step and Δt to the time step. The off-diagonal stress tensor components are shifted in space by half

an interval in two directions and the velocity components are shifted both in time and in space by a time step and by half a space time. At the lateral and bottom sides of the model, we add a PML (Perfectly Matched Layer) [9] layer to absorb the outgoing energy. A fixed size of ten grid points is chosen for the thickness of this layer (in blue color in Figure 2)

2.2. Standard parallel implementation

Most of standard parallel implementations of the elastodynamics equations are based on MPI cartesian grid decomposition. The computational domain is decomposed into sub-domains D_i in such that each sub-domain is mapped to one process for computing. Figure 2 demonstrates this decomposition with 3× 3 subdomains. This

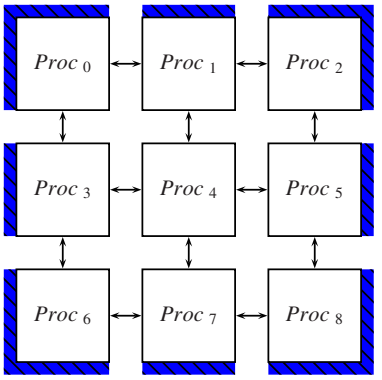

Fig. 2. Decomposition for global domain into 3 × 3 sub-domains

decomposition could be optimized using non-blocking communications and overlapping communication time by computation. One only needs to compute firstly the velocity and the stress components for the points p_{xyz} located at border B_i. Then, these values at border B_i are exchanged for Proc $_j$ corresponding. Finally, each Proc $_i$ determine the stress and velocity components for the other points $p_{xyz} \in \{D_i \Theta B_i\}$. Another popular way to extract the parallelism of such applications is to exploit the triple nested loops coming from the three dimensions of the problem under study. This allows a very straightforward use of OpenMP directives. This second level of decomposition could therefore be easily combined with a coarse-grained parallelism coming from MPI. This situation is detailed in algorithm 1.

3. Fine-grained parallelism

The simulations described in this section have been runned on `Jade` symmetric platform from the *Genci-Cines* French national computing center with 30 GB of memory and eight computing cores (Xeon-E5472) per node.

3.1. Load balancing

As shown in Figure 1 the standard Cartesian decomposition is based on a physical domain that involves several computing zones coming from the different numerical formulation. For instance, the border of the computational domain corresponds to the boundary conditions required to absorb the outgoing energy. The CPU-cost ratio observed between a boundary grid point and a physical domain point varies from two to four.
In our example, we consider a fixed problem size of 170 millions of grid points per node. This example allows us to saturate an average of 70% of the available memory on each node of the `Jade` platform. We consider a two-dimensional decomposition along the horizontal direction and we multiply the number of points in each direction (horizontal and vertical) by a factor of $\sqrt[3]{2}$ when we double the number of computing multicore nodes. This allows

Algorithm 1: Standard hybrid implementation of the elastodynamics equations based on MPI+OpenMP

```
1  for (∀ step of times Δt for simulating) do
2      Compute stress σ for all points p_xyz ∈ D_i begin
3          #pragma omp parallel for
4          for (x) do
5              for (y) do
6                  for (z) do
7                      Compute stress σ_xx, σ_yy, σ_zz, σ_xy, σ_xz, σ_yz at p_xyz;
8                  end
9              end
10         end
11         Exchange stress σ
12     end
13     Compute velocity for all points p_xyz ∈ D_i begin
14         #pragma omp parallel for
15         for (x) do
16             for (y) do
17                 for (z) do
18                     Compute velocity v_x, v_y, v_z at p_xyz;
19                 end
20             end
21         end
22         Exchange velocity
23     end
24 end
```

us to mimic the shape of the seismic basins encountered in standard studies. We define as the weak scaling the ratio between the elapsed time on one node and the elapsed time on a an increasing number of computing nodes.

Table 1 shows the degradation of the balance between the computing subdomains. On 256 cores, we reach a maximum imbalance of 45% between the MPI processes located inside the physical domain and those corresponding to the boundaries. This is coming from the fixed ratio between the PML grid points and the physical domain grid points for the subdomains located in the middle of computing domain. For the subdomains located at the border, the number of PML grid points is growing with respect to the number of physical grid points. The weak scaling results are quite good as they only rely on the slowest subdomains.

Figure 3 shows the idle time in the case of a standard Cartesian decomposition. For each MPI process, we can notice the communications with four neighbors as described in Figure 2. The green color rectangles correspond to the waiting time between the computing cores. The sizes of the rectangles vary because of the non-uniform computing costs for the velocity and the stress components depending on the location of the subdomain. The same situation could also be reported with standard OpenMP implementations as the decomposition is based on the outer loop corresponding to one-dimensional strip.

nodes (cores)	Weak scaling	Load imbalance (%)
1 (8)	1.00	9.8
2 (16)	1.23	14.1
4 (32)	1.14	15.4
8 (64)	1.17	16.7
16 (128)	1.32	21.4
32 (256)	1.01	45.0

Table 1. Load imbalance and weak scaling for the elastodynamics equations.

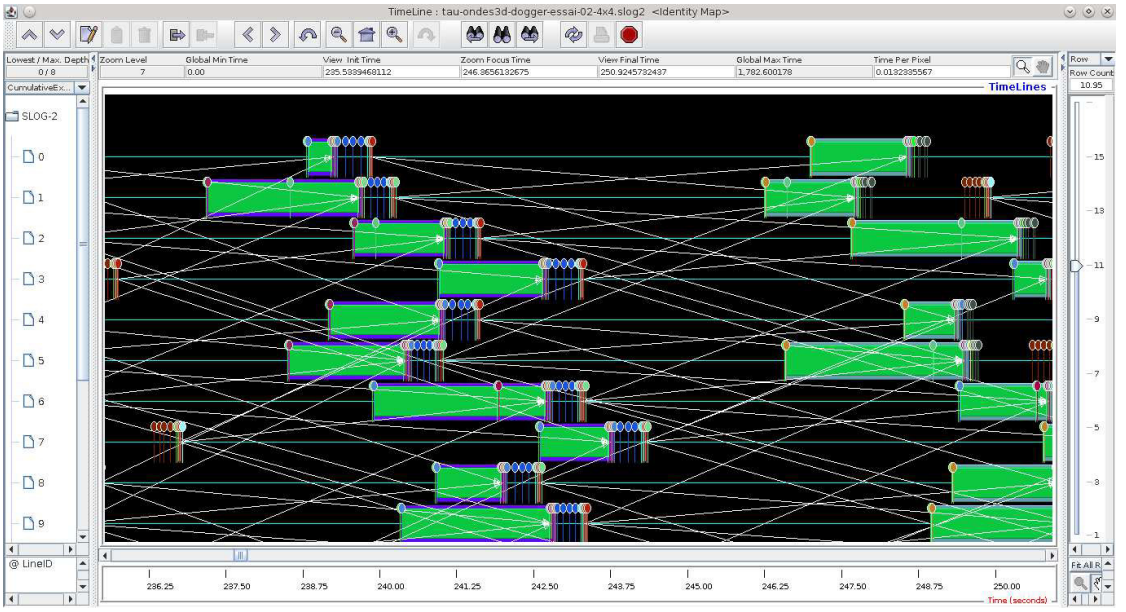

Fig. 3. Impact of the imbalance on the communication scheme between sixteen subdomains.

3.2. Programming model and communication strategies

The efficiency of the programming model selected to implement the elastodynamics equation can rather different depending on the granularity of the problem. For instance pure MPI or hybrid MPI+OpenMP implementation could lead to unexpected levels of performance (threads mapping, communication overlapping, efficient cooperation between MPI and OpenMP libraries.). In order to evaluate the impact of the size of the problem, we consider a standard situation with a problem that leads to a memory consumption of 2.6 GB per core.

t

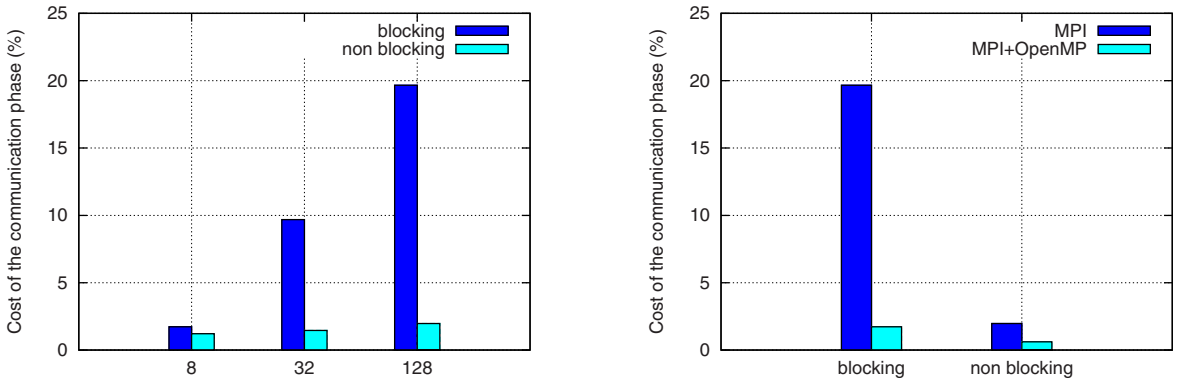

Fig. 4. Impact of the communications strategy with pure a MPI or a hybrid MPI+OpenMP implementation.

Figure 4 compares blocking and non-blocking communications. The directions and the volume of communication remain constant for each core but the total amount of communications is increasing. In the case of blocking communication, the contention on the network probably explains the degradation of the communication phase.

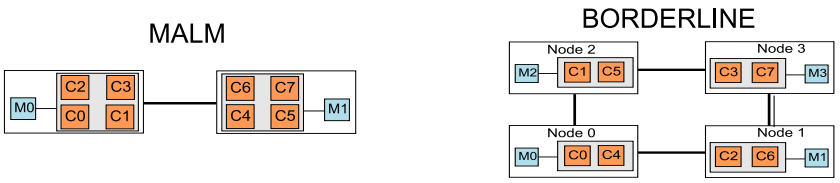

Fig. 5. Hierarchical and multicore architectures used as experimental platforms.

Characteristics	Malm platform	Borderline platform
Processor type	Opteron-2356	Opteron-2218
Cache memory (MB)	2	2
Total number of cores	8	8
Number of NUMA nodes	2	4
Memory per node (GB)	8	8

Table 2. Characteristics of the shared-memory architectures used in our experiments.

The non-blocking strategy provides very regular results with a maximum of 2% for the exchange cost of the boundaries between subdomains.
The plot on the right shows the results obtained on sixteen computing nodes. The hybrid programming model does not significantly impact the communication strategy. Using a three-dimensional decomposition makes the computation to communication ratio of the order of $\frac{V}{\sqrt[3]{P}}$ where V is the volume of the subdomain and P the number of computing cores. 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.

3.3. Related works and Discussions

Several approaches have been proposed to tackle the load balancing issues, particularly for the elastodynamics equation. One classical way of ensuring load balancing in the context of grid-based computation is the use of mesh partitioning techniques [10] 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 [11]. One could also exploit a quasi-static load-balancing algorithm based on zone costs [12]. The the main limitation is coming from the difficulties 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-grained computation.
Another techniques that have been introduced rely on domain overloading based on the efficiency of threads scheduling in order to balance the computation between the computing cores [13]. At the shared-memory level and for such a triple nested loop problem, OpenMP directives are added at the outer loop level to honor the unit-stride and reduce the number of costly OpenMP barriers. This induces a one-dimensional decomposition that is usually not the most efficient strategy for a three-dimensional problem. Specialized libraries such as Intel Threading Building Blocks (TBB) or Cilk [14] that provide work-stealing mechanisms can provide relevant solutions.

4. Memory affinity

4.1. Hardware characteristics

Our experiments have been carried out on two hierarchical multicore platforms that have different topologies regarding memory access. They are based on AMD Opteron processors with HyperTransport links between the different nodes. The NUMA factor induced by the memory hierarchy varies significantly. The topological links between the different nodes are described in Figure 5 for each platform and we summarize the characteristics of these different machines in Table 2.

4.2. Experiments

The experiments described in this section are based on problem sizes that saturate an average of 70% of the memory available on each NUMA node. In order to evaluate the impact of the hierarchical structure of the memory available on most of multicore platforms, we firstly consider the sequential implementation of the elastodynamics equations. We vary the mapping of the memory used by the application and the mapping of the computing thread by using *sched_setaffinity*. This allows us to occupy all the NUMA nodes of the architectures as threads are successively bind on the different cores.
In Tables 3 and 4 the mapping of the computing threads on the cores vary by columns and the memory mapping vary by lines. The zero values (**0**) are the references, i.e. the minimum time by lines. The results shows to the percentage of increase of the computing time in comparison with the reference time (i.e NUMA penalty).

On `Malm` platform, the maximum value of the penalty is 11.21%. This corresponds to an architecture based on a unique *Hypertransport* link. A closer look to the table reveals the global topology of the node as the penalty values inferior to 1% (for instance core #0, #1, #2, #3) correspond to a mapping of the memory and the computing thread on same NUMA node. These cores belong to the same multicore processor. In our case, the cross of one *Hypertransport* link leads to a NUMA penalty of an average of 10%. On `Borderline` platform, the maximum

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0.44	**0**	0.02	0.06	10.89	10.95	10.95	10.92
1	0.44	0.13	**0**	0.23	10.96	10.95	10.92	11.19
2	0.58	0.05	**0**	0.11	11.02	10.96	11.02	11.01
3	0.51	0.06	**0**	0.01	11.02	11.21	11.02	11.27
4	11.12	10.66	10.61	11.19	**0**	0.10	0.07	0.12
5	11.17	10.50	10.55	10.70	0.28	**0**	0.11	0.11
6	11.04	10.78	10.52	10.55	**0**	0.12	0.01	0.07
7	11.00	10.66	10.58	10.51	0.01	0.03	**0**	0.01

Table 3. Numa penalty for the elastodynamics equations on `Malm` platform.

value of the penalty is 20.88%. In this case, when data have to be exchanged through one *Hypertransport* link the average penalty is of 10% and of 20% for two links. The topology of the node is also revealed by our experiments showing that the core #0 and the core #4 belongs to the same dual-core processor. If we consider the parallel

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	**0**	8.97	8.20	18.67	1.14	8.74	7.43	18.73
1	8.01	**0**	18.87	8.24	9.26	0.04	18.39	7.92
2	7.62	18.96	0.62	8.43	9.07	18.91	**0**	7.88
3	19.12	9.10	9.56	**0**	20.88	9.06	8.62	0.38
4	**0**	9.14	8.21	18.85	1.37	9.04	7.55	18.53
5	9.09	**0**	20.12	9.45	10.47	1.60	19.61	8.89
6	7.65	19.33	0.17	8.64	8.43	19.35	**0**	8.46
7	19.67	9.78	9.34	0.75	20.70	9.50	9.13	**0**

Table 4. Numa penalty for the elastodynamics equations on `Borderline` platform.

OpenMP implementation of the elastodynamics equation, the maximum penalty observed on each platform is significantly increased. We compare two memory mapping strategies. The first one relies on the default Linux First-Touch memory policy. The second one exploits the parallel initialization of the data in order to maximize the locality between the data and the computing threads.
The penalty observed when all the computing cores are used is 53 % on `Borderline` platform and 19% on `Malm` platform. This is coming from the contention on the memory bus when all the computing threads access the data. This situation is worsened by the properties of the algorithm that is known to be memory-bound.

4.3. Related works and Discussions

Explicit approaches based on system calls (*mbind*()) or libraries and tools (*libnuma*) are difficult to handle at the application-level. Automatic approaches based on the use of memory policies are the simplest way of dealing with memory affinity. The performances measured strongly depend on the application under study [15]. User-level Application Programming Interface (API) like [16] provide tools to enhance memory placement with respect to the underlying architectures. One could also rely on more advanced strategies described for instance in [17]. In this case the idea is to define a recursive structure in order to group threads working on the same data.

5. Conclusion and future works

Exploiting multicore architectures efficiently is a critical topic for applications initially designed for distributed memory architectures or clusters of SMP. In this paper, we have underlined the limitations coming from the fine-grained parallelism leading to a degradation of the load balance. The impact of the programming model and the communication strategies can be significant depending on the ratio between the size of computing domain and the boundaries. We have also shown the NUMA penalty on two standard multicore platforms, the performance limitation can reach a maximum of 53% when all the available cores are used. Several improvements have been discussed in order to enhance the overall performance. These various levels of improvement probably need to be combined; auto-tuning strategies seem to be a promising way [18]. Another major limitation of the finite difference method is coming from the disadvantageous ratio between the limited pointwise computation and the intensive memory access required, leading to a memory-bound situation. Improvements at the algorithmic-level may be necessary to improve the peak performance level. For instance, spacetime decomposition appears like a promising strategy to tackle the poor pointwise computational intensity of the finite differences method [19].

Acknowledgements

This work is partially funded by the European FP7 IRSES project HPC-GA (High-Performance Computing for Geophysics Applications).

References

[1] R. Madariaga, Dynamics of an expanding circular fault, Bull. Seismol. Soc. Am. 66 (3) (1976) 639–666.
[2] J. Virieux, *P-SV* wave propagation in heterogeneous media: velocity-stress finite-difference method, Geophysics 51 (1986) 889–901.
[3] P. Moczo, J. Robertsson, L. Eisner, The finite-difference time-domain method for modeling of seismic wave propagation, in: Advances in Wave Propagation in Heterogeneous Media, Vol. 48 of Advances in Geophysics, Elsevier - Academic Press, 2007, Ch. 8, pp. 421–516.
[4] M. Chavez, E. Cabrera, R. Madariaga, N. Perea, C. Moulinec, D. Emerson, M. Ashworth, A. Salazar, Benchmark Study of a 3D Parallel Code for the Propagation of Large Subduction Earthquakes, in: Proceedings of the 15th Euro PVM/MPI Users' Group Meeting, Dublin, Ireland, 2008, pp. 303–310.
[5] T. Furumura, L. Chen, Parallel simulation of strong ground motions during recent and historical damaging earthquakes in Tokyo, Japan, Parallel Computing 31 (2) (2005) 149–165.
[6] S. E. Minkoff, Spatial Parallelism of a 3D Finite Difference Velocity-Stress Elastic Wave Propagation Code, SIAM J. Sci. Comput. 24 (1) (2002) 1–19.
[7] Y. Cui, K. B. Olsen, T. H. Jordan, K. Lee, J. Zhou, P. Small, D. Roten, G. Ely, D. K. Panda, A. Chourasia, J. Levesque, S. M. Day, P. Maechling, Scalable earthquake simulation on petascale supercomputers, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, 2010, pp. 1–20.
[8] A. R. Levander, Fourth-order finite-difference *P-SV* seismograms, Geophysics 53 (1988) 1425–1436.
[9] F. Collino, C. Tsogka, Application of the PML absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media, Geophysics 66 (1) (2001) 294–307.
[10] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[11] C. D. Norton, J. Z. Lou, 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, 2001, p. 120.
[12] S. Seguin, M. Cracraft, 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, Santa Clara, CA, USA, pp. 107–112.
[13] F. Dupros, H. Aochi, A. Ducellier, D. Komatitsch, J. Roman, Exploiting Intensive Multithreading for the Efficient Simulation of 3D Seismic Wave Propagation, in: Proceedings of the 11th IEEE CSE'08, International Conference on Computational Science and Engineering, São Paulo, Brazil, 2008, pp. 253–260.

[14] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, Y. Zhou, Cilk: an efficient multithreaded runtime system, SIGPLAN Not. 30 (8) (1995) 207–216.
[15] J. Antony, P. P. Janes, A. P. Rendell, Exploring Thread and Memory Placement on NUMA Architectures: Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport, in: Proceedings of HiPC 2006, Bangalore, India, 2006, pp. 338–352.
[16] C. P. Ribeiro, J.-F. Méhaut, A. Carissimi, M. B. Castro, L. G. Fernandes, Memory affinity for hierarchical shared memory multiprocessors, in: 21st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2009, São Paulo, Brazil, 2009, pp. 59–66.
[17] S. Thibault, R. Namyst, P.-A. Wacrenier, Building Portable Thread Schedulers for Hierarchical Multiprocessors: The BubbleSched Framework, in: Proceedings of Euro-Par 2007, 13th International Euro-Par Conference, Rennes, France, 2007, pp. 42–51.
[18] M. Christen, O. Schenk, Y. Cui, Patus for convenient high-performance stencils: evaluation in earthquake simulations, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, 2012, pp. 11:1–11:10.
[19] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, K. Yelick, Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures, in: SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, Austin, Texas, USA, 2008, pp. 1–12.