The Journal of Supercomputing. 2014, 70(2): 514-526. doi:10.1007/s11227-013-1065-x

Noname manuscript No.
(will be inserted by the editor)

Performance analysis of SSE and AVX instructions in multi-core CPUs and GPU computing on FDTD scheme for solid and fluid vibration problems

Jorge Francés · Sergio Bleda · Andrés Márquez · Cristian Neipp · Sergi Gallego · Beatriz Otero · Augusto Beléndez

Received: date / Accepted: date

Abstract In this work a unified treatment of solid and fluid vibration problems is developed by means of the Finite-Difference Time-Domain (FDTD). The scheme here proposed takes advantage from a scaling factor in the velocity fields that improves the performance of the method and the vibration analysis in heterogenous media. Moreover, the scheme has been extended in order to simulate both the propagation in porous media and the lossy solid materials. In order to accurately reproduce the interaction of fluids and solids in FDTD both time and spatial resolutions must be reduced compared with the set up used in acoustic FDTD problems. This aspect implies the use of bigger grids and hence more time and memory resources. For reducing the time simulation costs, FDTD code has been adapted in order to exploit the resources available in modern parallel architectures. For CPUs the implicit usage of the advanced vectorial extensions (AVX) in multi-core CPUs has been considered. In addition, the computation has been distributed along the different cores available by means of OpenMP directives. Graphic Processing Units (GPU) have been also considered and the degree of improvement achieved by means of this parallel architecture has been compared with the highly-tuned

J. Francés · S. Bleda · A. Márquez · C. Neipp · S. Gallego
Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal, Universidad de Alicante, Spain
Tel.: +34-965903682
Fax.: +34-965909750
E-mail: jfmonllor@ua.es

B. Otero
Dpt. d'Arquitectura de Computadors,Universitat Politècnica de Catalunya, Barcelona Spain
Tel.: +34-934054046
Fax.: +34-934017055
E-mail: botero@ac.upc.edu

A. Beléndez
Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal, Universidad de Alicante, Spain
Instituto Universitario de Física Aplicada a las Ciencias y las Tecnologías, Spain
Tel.: +34-965909951
Fax.: +34-965909951
E-mail: a.belendez@ua.es

CPU scheme by means of the relative speed up. The speed up obtained by the parallel versions implemented were up to 3 (AVX and OpenMP) and 40 (CUDA) times faster than the best sequential version for CPU that also uses OpenMP with auto-vectorization techniques, but non includes implicitely vectorial instructions. Results obtained with both parallel approaches demonstrate that massive parallel programming techniques are mandatory in solid-vibration problems with FDTD.

Keywords FDTD · GPU · CPU · OpenMP · AVX · vibration

PACS 47.11.Bc · 61.20.Ja · 79.20.Ap

Mathematics Subject Classification (2000) 68U20 · 65L12 · 68W10

1 Introduction

Kane S. Yee published in 1966 the initial FDTD scheme [1]. During the next two decades the scientific community do not show too much interest in this method due to its high computational requirements. However, the growing of the computer power in the last three decades has permitted that FDTD became a reference in different fields of science such as electromagnetism [2], optics and acoustics. This new scenario has allowed to develop new applications and also new formulations that cover a wider range of problems. The first attempts of FDTD in acoustics were published by Botteldooren [3], LoVetri [4], Wang [5] and its application to solid mechanics was performed by Virieux [6] and Cao [7] in the field of seismology. The formulation of FDTD for solids can be separated in two different schemes. The former is based on the discretization of the Newton's second law and the Hook's law; the latter is based on the vectorial and scalar potentials derived from the two general laws of solid-mechanics mentioned below. Regarding this second approach there are several contributions [8–10]. In this work, the first scheme is developed in order to model solid and fluid vibrations. The formulation based on the scalar and vectorial potentials has been developed efficiently only for homogeneous media, due to the difficulties derived from the boundary conditions in the interfaces between solid and fluid [11].

On the other hand, the direct application of the finite-difference approach to the Newton's second law and the Hook's law allows to model the interaction between fluids and solids, due to the fact that the derivation of the initial FDTD scheme for acoustics is a particular case from these laws. The vibration analysis in fluids and solids require reduced values for time and spatial resolutions, since the propagation in solids use to be faster than in fluids such as air for instance. In addition, FDTD scheme computes the field distribution as a function of time, thus sometimes a big number of time steps is required in order to ensure steady state. FDTD also requires a discretization of the simulation region. The size of this grid affects directly the time simulation costs and also the memory requirements of the method. In order to reduce the time simulation costs in seismology some works related with GPU computing have been developed [12].

In this work, a unified treatment of fluid and solid FDTD analysis is performed. The formulation has been slightly modified from [13,14] in order to model efficiently the vibration fields for fluid-solid interaction problems and porous and solid lossy materials. Moreover, a rigorous analysis of the performance of the 2D

FDTD in both parallel architectures multi-core CPU and GPU is performed. Both parallel implementations of the method are compared with a sequential code that takes advantage of OpenMP directives and also of the auto-vectorization available in modern compilers. It is worth to note that this version is faster than the usual sequential codes, since its implementation has been performed taking into consideration strategies that benefit the auto-vectorization features provided by modern compilers. On the other hand, the parallel CPU version takes advantage of the advanced vectorial extensions (AVX) available in modern microprocessors. The AVX instructions set has been directly used in order to exploit the potential of modern CPUs. In addition, parallel strategies have been considered in order to use all the available cores in the CPU, thus OpenMP directives have been also applied in all CPU versions. This fine-tuned CPU version of the FDTD scheme has been compared with a massively parallel CUDA code for GPU computing. The idea is to accurately estimate how fast is a GPU against a CPU that exploits all its available resources. This analysis has been also performed for the standard FDTD scheme applied to optics [15] and for the solid-fluid scheme and SSE [14], but it has not been done yet for the application of AVX in CPUs to the best of our knowledge.

The structure of the paper is as follows, in section 2 a two-dimensional FDTD scheme for solid and fluid media is briefly outlined, in section 3 the computational strategies considered for both GPU and CPU are given, in section 4 we present a comparison of the results obtained with the CPU code that takes advantage of the AVX and OpenMP directives and GPU computing. In this section the potential of the FDTD scheme here proposed is shown by means of the simulation of a common situation in building acoustics based on a cross-section with porous and lossy materials, finally section 5 describes the main conclusions of this paper.

2 Finite-Difference Time-Domain method for the analysis of vibration problems

The fundamental constitutional equations for the propagation of elastic waves in solids can be derived in vectorial notation from the Newton's second law and the Hook's law obtaining the following well known identities:

$$\frac{\partial \boldsymbol{\tau}}{\partial t} + \boldsymbol{\gamma}\boldsymbol{\tau} = \lambda \mathbf{I} \left(\nabla \cdot \mathbf{v}\right) + \mu \left(\nabla \mathbf{v} + \mathbf{v}\nabla\right), \tag{1}$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} = \nabla \cdot \boldsymbol{\tau} \tag{2}$$

where $\boldsymbol{\tau}$ is the stress tensor, $\boldsymbol{\gamma}$ is the resistivity tensor, $\mathbf{I}$ is the identity matrix, λ, μ and ρ are the two Lamé constants and the density respectively. The velocity vector is denoted by $\mathbf{v}$. The Eqs. (1)-(2) describe the propagation in linear, homogeneous and non-lossy solid media. Basically, in solid media there are two types of waves, the p-waves or longitudinal waves and the s-waves also known as the transversal waves. The velocity of propagation of both waves is different and defined as follows:

$$c_p = \sqrt{\tfrac{\lambda+2\mu}{\rho}}, \quad c_s = \sqrt{\tfrac{\mu}{\rho}}. \tag{3}$$

where c_p and c_s are the p-wave and s-wave velocities, respectively. It is important to note that those materials with null second Lamé parameter μ (shear modulus)

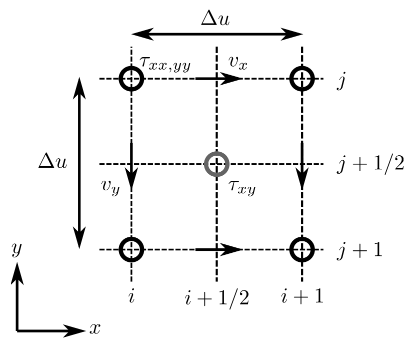

Fig. 1 FDTD lattice for solids.

behave as fluids with a compressibility modulus defined as $k=-\lambda$. The presence of heterogeneities and the p and s waves deal to different ondulatory phenomena such as the Rayleigh's and Love's waves.

In this work a modified set of equations are considered based on a scaling of the velocity components. For the specific case of 2D simulation (x, y) the normal stresses from Eq. (1) can be expressed as follows

$$\frac{\partial \tau_{xx}}{\partial t}+\gamma_p\tau_{xx}=c_p\frac{\partial V_x}{\partial x}+\frac{\lambda}{Z_0}\frac{\partial V_y}{\partial y}, \tag{4}$$

$$\frac{\partial \tau_{yy}}{\partial t}+\gamma_p\tau_{yy}=c_p\frac{\partial V_y}{\partial y}+\frac{\lambda}{Z_0}\frac{\partial V_x}{\partial x}, \tag{5}$$

whereas the shear stress can be defined as:

$$\frac{\partial \tau_{xy}}{\partial t}+\gamma_s\tau_{xy}=\frac{c_s^2}{c_p}\left(\frac{\partial v_x}{\partial y}+\frac{\partial v_y}{\partial x}\right), \tag{6}$$

with γ_p and γ_s being the longitudinal and shear resistivity parameters respectively. The velocity components can be split from Eq. (2) as follows:

$$\frac{\partial V_x}{\partial t}=c_p\frac{\partial \tau_{xx}}{\partial x}+c_p\frac{\partial \tau_{xy}}{\partial y}, \quad \frac{\partial V_y}{\partial t}=c_p\frac{\partial \tau_{yy}}{\partial y}+c_p\frac{\partial \tau_{yx}}{\partial x}, \tag{7}$$

where $V_i=v_iZ_0$ with $i=x,y$ and $Z_0=\rho c_p=\sqrt{\rho\left(\lambda+2\mu\right)}$. The scaling of the velocity components has many advantages such as the possibility of handle closer values for the velocities in both solids and fluids. Note that usually the movement of the particles in solids tend to be smaller than in a fluid. On the other hand, this normalization improves the finite error precision in FDTD equations since avoids the round-off errors committed by the processor due to handling numbers with huge differences in their modulus.

This formulation can be easily related with the pressure-velocity scheme for fluid media considering $k=-\lambda$ and $p=-1/2(\tau_{xx}+\tau_{yy})$.

Figure 1 shows the lattice configurations of the FDTD method for analysing the velocities and stress components. It is worth to note that FDTD scheme here proposed is based on an explicit scheme in which the calculation of the vibration fields at later time is obtained from the current time values of the vibration fields. Reformulating Eqs. (4)-(7) by the FDTD method gives

$$\begin{aligned}
\tau_{xx}|_{i,j}^{n+1} &= a_p|_{i,j}\tau_{xx}|_{i,j}^{n} + b_p|_{i,j}\left[VPL\left(V_x|_{i+1/2,j}^{n+1/2} - V_x|_{i-1/2,j}^{n+1/2}\right)+\right.\\
&\left.+VCL\left(V_y|_{i,j+1/2}^{n+1/2} - V_y|_{i,j-1/2}^{n+1/2}\right)\right], \qquad (8)\\
\tau_{yy}|_{i,j}^{n+1} &= a_p|_{i,j}\tau_{yy}|_{i,j}^{n} + b_p|_{i,j}\left[VPL\left(V_y|_{i,j+1/2}^{n+1/2} - V_y|_{i,j-1/2}^{n+1/2}\right)+\right.\\
&\left.+VCL\left(V_x|_{i+1/2,j}^{n+1/2} - V_x|_{i-1/2,j}^{n+1/2}\right)\right], \qquad (9)\\
\tau_{xy}|_{i+1/2,j+1/2}^{n+1} &= a_s|_{i+1/2,j+1/2}\tau_{xy}|_{i+1/2,j+1/2}^{n} + b_s|_{i+1/2,j+1/2}\left[VSL\times\right.\\
&\left.\left(V_x|_{i+1/2,j+1}^{n+1/2} - V_x|_{i+1/2,j}^{n+1/2} + V_y|_{i+1,j+1/2}^{n+1/2} - V_y|_{i,j+1/2}^{n+1/2}\right)\right], \qquad (10)\\
V_x|_{i+1/2,j}^{n+1/2} &= V_x|_{i+1/2,j}^{n-1/2} + VPL\left(\tau_{xx}|_{i+1,j}^{n} - \tau_{xx}|_{i,j}^{n}+\right.\\
&\left.+\tau_{xy}|_{i+1/2,j+1/2}^{n} - \tau_{xy}|_{i+1/2,j-1/2}^{n}\right), \qquad (11)\\
V_y|_{i,j+1/2}^{n+1/2} &= V_y|_{i,j+1/2}^{n-1/2} + VPL\left(\tau_{yy}|_{i,j+1}^{n} - \tau_{yy}|_{i,j}^{n}+\right.\\
&\left.+\tau_{yx}|_{i+1/2,j+1/2}^{n} - \tau_{yx}|_{i-1/2,j+1/2}^{n}\right), \qquad (12)
\end{aligned}$$

where

$$VPL = \frac{\Delta t c_p}{\Delta u}, \quad VSL = VPL\left(\frac{c_s}{c_p}\right)^2, \quad VCL = \frac{\Delta t \lambda}{\Delta u Z_0}. \qquad (13)$$

$$a_p|_{i,j} = \frac{1-\gamma_p|_{i,j}}{1+\gamma_p|_{i,j}}, \quad a_s|_{i+1/2,j+1/2} = \frac{1-\gamma_s|_{i+1/2,j+1/2}}{1+\gamma_s|_{i+1/2,j+1/2}}, \qquad (14)$$

$$b_p|_{i,j} = \frac{1}{1+\gamma_p|_{i,j}}, \quad b_s|_{i+1/2,j+1/2} = \frac{1}{1+\gamma_s|_{i+1/2,j+1/2}}, \qquad (15)$$

The spatial and time resolution of FDTD are denoted by Δu and Δt, where square cells are considered ($\Delta u = \Delta x = \Delta y$) (see Fig. 1). Regarding stability and dispersion, FDTD schemes must ensure the Courant-Friedrichs-Lewy (CFL) that for the specific case of solid-fluid interaction is defined as:

$$S = \frac{c_{l,max}\Delta t}{\sqrt{2}\Delta u} \leq 1, \quad R = \frac{c_{s,min}}{f_{max}\sqrt{2}\Delta u} \geq 10 \qquad (16)$$

where $c_{l,max}$ is the maximum value of the longitudinal velocity in the domain and $c_{s,min}$ the lowest tangential velocity [16,13]. The boundaries of the domain have been truncated by means of a Perfectly Matched Layer (PML) that solves the problems due to artifacts and reflections on the boundaries which are very common in finite-difference schemes [13]. More specifically, to derive the PML formulation for elastic waves using the FDTD method, the complex coordinate stretching method is applied to these governing equations [17,18]. In the absorbing layer, the complex coordinate setretching method replaces the original coordinate variable with a complex coordinate variable in both the equation of motion and Hooke's law, Eqs. (1) and (2) respectively. The imaginary part of the complex coordinate is related to the wave attenuation coefficient, thus waves in the boundary layer are attenuated. The formulation of the PML can follow the sintax detailed in (4)-(6) but considering a fictitious resistivity parameter that has nothing to do with the real γ which specify the medium.

Regarding porous materials, its modelling has been perfomed following the model proposed by Biot in which a porous material can be simulated as a solid material surrounded by a viscous fluid. Under this assumption a longitudinal wave appears with velocity

$$c_p' \simeq \sqrt{\frac{K_g + \frac{4}{3}\mu_m + \frac{1-\phi^2}{\phi}K_f}{\rho_m}} = \sqrt{\frac{\lambda_m + \frac{3}{4}\mu_m}{\rho_m}} \tag{17}$$

$$\rho_m = (1-\phi)\rho_s + \phi\rho_f, \tag{18}$$

with K_g, K_f being the bulk modulus of the frame and the fluid respectively, ϕ the porosity and ρ_s and ρ_f the density of the solid and the fluid surrounding material respectively. This model is fully detailed in [19,20] and has been directly included in the FDTD method here proposed simulating the porous media such as a solid material with a set or parameters defined in the literature.

3 Computational optimization

In this section the strategies considered for accelerating FDTD for solid-fluid vibration analysis are detailed. In this work an Intel Xeon E5-2360 processor with 15MB of cache, a clock speed of 2.3 GHz and the possibility of handle efficiently twelve threads has been used. Regarding GPU computing, a GTX660 GPU with Kepler architecture is considered for the parallel implementation of the FDTD for solid-fluid vibration analysis. The basic computing unit in this type of hardware consists of 32 threads and this arrangement is defined as a warp. The GPU is capable of swapping warps into and out of context without any performance overhead. This functionality provides an important method of hiding memory and instruction latency on the GPU hardware. The different warps invoked are also arranged into blocks of threads.

3.1 Multi-core CPU and AVX instructions

The AVX extensions were firstly supported by Intel processors with the Sandy Bridge processor in 2011. AVX instructions follow the parallel computation model and is the most cost-effective way of accelerating floating- or double-point performance in modern processors. Here, only single precison has been considered for FDTD simulations, since single precision is accurate enough for FDTD applications. In order to successfully apply the AVX instructions, load operations must be done under a set of aligned bytes [21]. For that reason, the allocation of the memory for a matrix with e_r rows and e_c columns is done as a single aligned column vector by means of a new array class fully implemented in C++ [22]. Thus each position is reached taking into account that the storage order is by columns. For simplicity, the scheme of how Eq. (8) is computed by means of the AVX registers and OpenMP is shown in Fig. 2a. The updating process for the rest of the components follow the same scheme. For simplifying the notation, also the PML

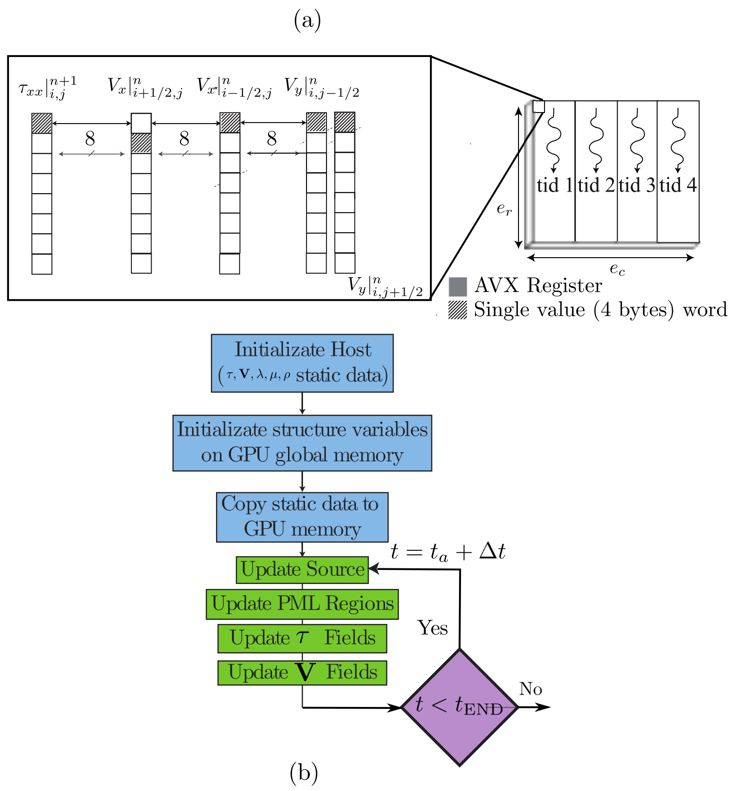

Fig. 2 a) Illustration showing the scheme for solving Eq. (8) in CPU. b) Flowchart of GPU parallel programming for FDTD for fluid-solid vibration analysis.

notation has been supressed but it is also included in the optimization process. For instance, the update of τ_{xx} requires several aligned loads that store 8 consecutive values of the terms involved: $V_x|_{i+1/2,j}^{n}$, $V_x|_{i-1/2,j}^{n}$, $V_y|_{i,j-1/2}^{n}$,$V_y|_{i,j+1/2}^{n}$. Note that the physical parameters that define the media are also stored in the AVX registers. The next step is to perform the arithmetic operations by means of the AVX registers using the intrinsics functions (`_mm256_sub_ps(__m256 a, __m256 b)`, `_mm256_add_ps(__m256 a, __m256 b)` or `_mm256_mul_ps(__m256 a, __m256 b)` for instace). In addition, OpenMP has been considered in order to parallelize the updating of each field component. Modern CPUs contain several cores that can be used by means of shared memory schemes in order to split the computational load amongst the different cores. By means of OpenMP directives, each component has been parallelized distributing the whole computation by columns. As can be seen in Fig. 2a, each thread is in charge of computing a set of columns of each field component, whereas each thread uses extensively the vectorial AVX instructions along the rows direction.

3.2 Graphic Processing Units (GPUs)

Regarding the SF-FDTD implementation and GPU computing, a number of blocks related with the number of rows and columns are invoked by means of the kernel functions and an array of 192×2 threads are launched per block [23].

Besides the potential of the CUDA kernel, it is necessary to divide the whole computation process in several kernels focused on computing each component of the vibration field. Fig. 2b summarizes the invocation path of the kernels related with the FDTD implementation. Firstly, an initialization in host of the fields to be computed is perfomed. Secondly, the allocation of these componentes is done inside the GPU, those fields that must be filled such as the physical parameters that models the media are copied to the GPU memory. Thirdly, the FDTD computation is performed by a set of kernels that update each component of the vibration field (stress and velocities). Finally, the fields are downloaded to the host. In this flow chart the post-process is omitted, but mandatory downloads of the τ and $\mathbf{V}$ components must be considered in order to compute the specific desired outputs. The time costs of this process has been also considered in order to compute the speed up in the results section.

4 Results

Firstly, the computational results are summarized in Fig. 3. The simulation grid is modified as a function of the number of rows (e_r) and columns (e_c). More specifically, Fig. 3a-b shows the time simulation cost and the speed up respectively for a set of simulations with $e_c = 500$, $e_c = 1000$ and $e_c = 1500$ varying the number of rows. The relative speed up has been computed considering as sequential version an auto-vectorized code with also OpenMP directives. The auto-vectorization provided by modern compilers (flag `-O3` in gcc) is based on predicting which loops can be automatically vectorized, or converted into vectorial instructions [24]. This auto-vectorized code is expected to be the fastest sequential code achievable by a programmer without advanced knowledge on parallelization techniques. The relative speed up obtained from the CPU parallel code optimized with AVX instructions and OpenMP behaves quite constant as a function of e_c and a slight maximum can be identified for the specific case of $e_r \approx 200$ cells. This local maximum belongs to an optimal usage of the cache memory available in the microprocessor for small computational sizes. As the simulation size becomes greater the simulation does not fit in the cache and the speed up remains constant. The overall speed up obtained by this version is closer to 3 compared with the auto-vectorized version. On the other hand, the GPU CUDA version remains more homogeneous as a function of the simulation size and the speed ups are quite near to 40 respect of the auto-vectorized sequential version. In order to accurately compare the GPU and the AVX+OpenMP CPU version, the relative speed up between them has been computed and shown in Fig. 3c. As can be seen the GPU version is near to 15 times faster than the fine-tuned CPU code and this behavior remains constant as a function of the simulation size. The effect of the cache size in the AVX+OpenMP CPU version can be also seen in Fig. 3c for $e_r \approx 200$ cells. The slight differences in the relative speed up curves reveals that the GPU version is more competitive for bigger simulation sizes, reaching values

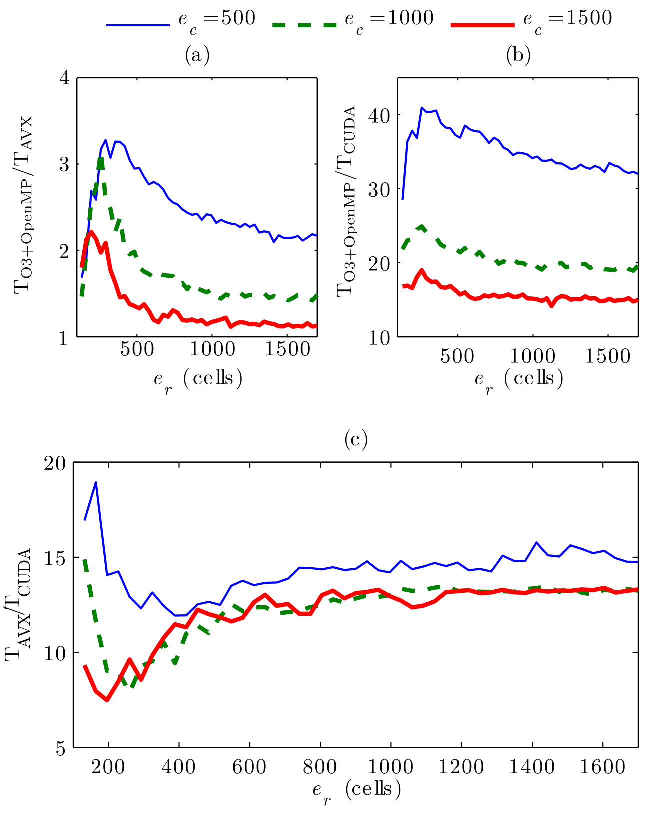

Fig. 3 Computational results as a function of the number of cells for the specific case of $e_c = 500$, $e_c = 1500$ and $e_c = 1500$. (a) Represents the relative speed up for the AVX version. (b) Relative speed up between the non-vectorial CPU version and GPU code. (c) Relative speed up between the AVX and CUDA codes.

of near to 15. This value is significantly different from the relative speed up values obtained in Fig. 3b. These results illustrates that AVX extensions in modern processors do not introduce a significant improvement in terms of computational costs compared to a massively computational architecture such as GPU in which the number of cores is a hundred times greater than in a single CPU. Nevertheless, significant improvements can be achieved by vectorization in modern processor by means of the auto-vectorization and also by means of the implicit usage of the vectorial registers available such as SSE or AVX. The full usage of these capabilities are needed in order to exploit all the resources of the processor and thus accurately analyze the real degree of improvement obtained by GPU computing. The results show that GPU codes are mandatory in this type of applications in which the requirements in terms of grid size and time steps can be unaffordable for sequential codes and even to multi-core processors. The low performance obtained by AVX compared to the relative speed up achieved by the conventional streamings SIMD extensions (SSE) in [14] is due to the overhead produced by unalinged loads required in FDTD codes. This overhead is minimized in SSE instructions and modern processors, whereas for AVX can not be neglected. It is espected that the

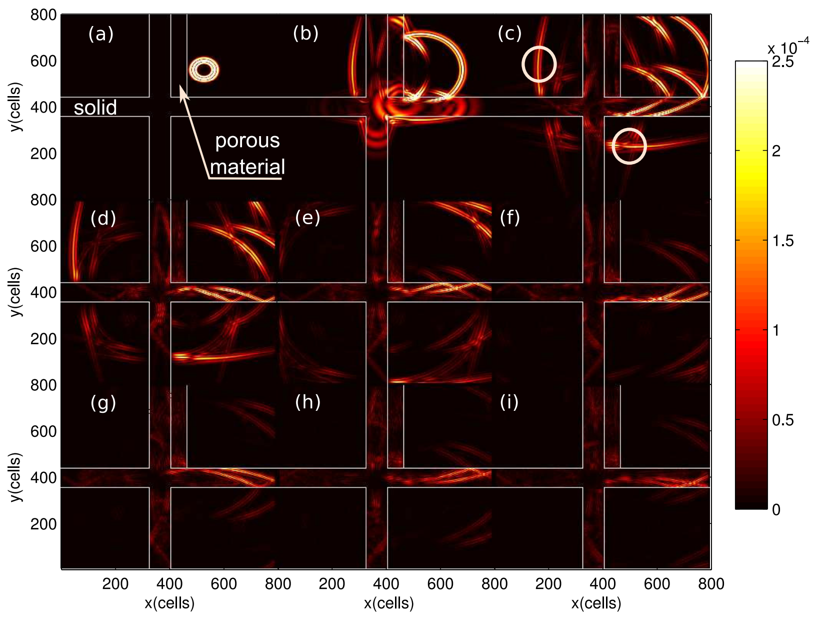

Fig. 4 Modulus of the the normalized velocity as a function of the space and time: (a) $n_{\text{step}} =$ 500, (b) $n_{\text{step}} = 1500$, (c) $n_{\text{step}} = 2500$, (d) $n_{\text{step}} = 3500$, (e) $n_{\text{step}} = 4500$, (f) $n_{\text{step}} = 5500$, (g) $n_{\text{step}} = 6500$, (h) $n_{\text{step}} = 7500$, (i) $n_{\text{step}} = 8500$

AVX-512 new compliant proposed by Intel in July 2013 minimizes this aspect and also expands the width of the vectorial registers to 512-bit amongst the 256-bit in current AVX.

Fig. 4 shows a sequence of the modulus of the scaled velocity as a function of the space and the time-simulation steps. The pressure source is located near the upper right corner. In the center of the domain there is a cross section filled with a solid material. In the upper right corner a porous material has been attached to the upper vertical wall in order to simulate a configuration close to a partition in building acoustics. It can be easily seen that the propagation inside the solid is faster rather in the fluid and also that different phenomena can be identified inside the different materials. More specifically, Fig. 4b-d shows that the longitudinal waves transmitted along the vertical wall are slightly attenuated compared with those transmitted to the right bottom partition. These waves are emphasized by means of two white circles in Fig. 4c. The shear waves cannot travel in fluid media thus, only longitudinal waves travel in the fluid outside the cross section whereas inside the solid material shear waves can be appreciated. For instance, Fig. 4b-c shows the sequence of the propagation of pressure waves in the right bottom partition that are mainly produced by the longitudinal waves that travel through the rigid structure along the x direction. As the time step grows, the transversal waves that travel slower than longitudinal waves remain in the solid cross section, but they do not contribute significantly to the velocity distribution inside the partition since shear waves can not travel in fluids. Hence, the contribution in terms of pressure is mainly due to the initial time steps as can be seen in the sequence given in Fig. 4.a-e.On the other hand, the dissipation of energy inside the porous layer can be identified in Fig. 4f-i.

A reduction in the amplitude in the waves inside the cross section can be percived due to the application of $\gamma_p = \gamma_s = 1 \cdot 10^{-5}$ s^{-1}. Although, the losses effect is small, it can be slightly identified in the absence of standing waves in the rigid structure, and it is expected to take advantage of these properties in large scale simulations. This sequence reveals in a qualitatively way the potential of this scheme and are consistent with those obtained in [13,14]. The set-up of the FDTD method and the values of the physical parameters that model the media are summarized in Table 1 and Table 2, respectively.

Table 1 Setup parameters of FDTD for results in Fig. 4.

f_0 (nm)	Δ (mm)	Δt (μs)	e_r (cells)	e_c (cells)	r_{PML} (cells)	e_{steps}
20 kHz	7.45	2.5	800	800	12	11313

Table 2 Solid (s) and porous (m) material parameters for results in Fig. 4. Note that for air λ_0 = -0.142 MPa, μ_0= 0 Pa and ρ_0 = 1.21 kg/m^3

λ_s (MPa)	μ_s (MPa)	ρ_s (kg/m^3)	λ_m (MPa)	μ_m (MPa)	ρ_m (kg/m^3)
$1.952 \cdot 10^3$	36	900	340	0.98	520

5 Conclusions

In this work a unified scheme for FDTD analysis of vibrations on fluid and solid media is considered. This scheme has been extended in order to simulate lossy solid materials and porous media has been also considered. A scaling factor has been introduced in the formulation in order to improve its implementation and also for optimizing the vibration analysis in heterogenous media. The formulation has been implemented in parallel hardware architectures such as multi-core CPU and GPU. The CPU optimized version takes advantage of the AVX instructions and also of the multiple cores available by means of OpenMP directives. Although, a fine-tuned CPU version can be competitive compared to GPU codes, since it reaches speed ups closer to 3 compared to the auto-vectorized sequential version, the effort necessary for including AVX extensions may not worth it compared to the code with auto-vectorization and OpenMP directives. GPU computing is mandatory for massively computations due to the fact that the speed up obtained is up to 40. It's worth to note that the speed up obtained from GPU codes can vary dramatically as a function of the sequential code selected. In this work a comparison between the multi-core CPU code acclerated with AVX and OpenMP is compared with the GPU CUDA based code in order to establish accurately the degree of improvement achieved, thus revealing that GPU is sligtly more than 15 times faster than the full vectorial and parallel CPU version.

Finally, FDTD applied to the analysis of elastic waves in solids and fluids has been demonstrated to have a low operational intensity, since the performance of FDTD is mostly limited by the memory bandwidth [25] thus the reduction of the simulation costs is attainable applying parallel strategies. The authors are considering to extend the current work to 3-D.

Acknowledgements

The work is partially supported by the “Ministerio de Economía y Competitividad” of Spain under project FIS2011-29803-C02-01, by the Spanish Ministry of Education (TIN2012-34557), by the “Generalitat Valenciana” of Spain under projects PROMETEO/2011/021 and ISIC/2012/013, and by the “niversidad de Alicante” of Spain under project GRE12-14.

References

1. K. Yee, Antennas and Propagation, IEEE Transactions on **14**(3), 302 (1966). DOI 10.1109/TAP.1966.1138693
2. A. Taflove, S.C. Hagness, *Computational electrodynamics: The Finite-Difference Time-Domain method* (Artech House, Norwood, MA, 2004)
3. D. Botteldooren, The Journal of the Acoustical Society of America **98**(6), 3302 (1995). DOI 10.1121/1.413817. URL http://link.aip.org/link/?JAS/98/3302/1
4. J. LoVetri, D. Mardare, G. Soulodre, The Journal of the Acoustical Society of America **100**(4), 2204 (1996). DOI 10.1121/1.417929. URL http://link.aip.org/link/?JAS/100/2204/1
5. S. Wang, The Journal of the Acoustical Society of America **99**(4), 1924 (1996). DOI 10.1121/1.415375. URL http://link.aip.org/link/?JAS/99/1924/1
6. J. Virieux, Geophysics **51**, 889 (1986). DOI 10.1190/1.1442147
7. S. Cao, S. Greenhalgh, Geophysical Journal International **109**(3), 525 (1992). DOI 10.1111/j.1365-246X.1992.tb00115.x. URL http://dx.doi.org/10.1111/j.1365-246X.1992.tb00115.x
8. M. Sato, Y. Takahata, M. Tahara, I. Sakagami, in *Ultrasonics Symposium, 2001 IEEE*, vol. 1 (2001), vol. 1, pp. 851–854 vol.1. DOI 10.1109/ULTSYM.2001.991853
9. M. Sato, Acoustical science and technology **25**(5), 382 (2004). DOI 10.1250/ast.25.382. URL http://ci.nii.ac.jp/naid/110003102921/en/
10. M. Sato, Acoustical Science and Technology **28**(1), 1346 (2007)
11. J. Francés, J. Ramis, J. Vera, in *Proceedings of the ICSV16, 5–9 July, Kraków, Poland* (2009), pp. 1–8
12. T. Okamoto, H. Takenaka, in *Proceedings of the International Symposium on Engineering Lessons Learned from the 2011 Great East Japan Earthquake* (2011), pp. 249–360
13. N.J. González, Simulación de tejidos vegetales mediante diferencias finitas. Master’s thesis, EPSG-UPV (2009)
14. J. Francés, S. Bleda, A. Márquez, C. Neipp, S. Gallego, B. Otero, A. Beléndez, in *Proceedings of the 2013 International Conference on CMMSE*, vol. 2 (2013), pp. 681–692
15. J. Francés, S. Bleda, C. Neipp, A. Márquez, I. Pascual, A. Beléndez, Computer Physics Communications **184**(3), 469 (2013). DOI http://dx.doi.org/10.1016/j.cpc.2012.09.025. URL http://www.sciencedirect.com/science/article/pii/S0010465512003128
16. C. Schroder, W. Scott, Geoscience and Remote Sensing, IEEE Transactions on **40**(2), 474 (2002). DOI 10.1109/36.992813
17. W.C. Chew, W.H. Weedon, Microwave and Optical Technology Letters **7**(13), 599 (1994). DOI 10.1002/mop.4650071304.
18. W.C. Chew, Q.H. Liu, Jouranl of Computational Acoustics **4**(4), 341 (1996). DOI 10.1142/S0218396X96000118. URL http://dx.doi.org/10.1142/S0218396X96000118

19. M.A. Biot, Geoscience and Remote Sensing, IEEE Transactions on **28**(2), 168 (1956)
20. M.A. Biot, Geoscience and Remote Sensing, IEEE Transactions on **28**(2), 179 (1956)
21. S. Thakkur, T. Huff, Computer **32**(12), 26 (1999)
22. J. Francés, S. Bleda, S. Gallego, C. Neipp, A. Márquez, I. Pascual, A. Beléndez, The Journal of Supercomputing **64**(1), 28. DOI 10.1007/s11227-012-0803-9. URL http://dx.doi.org/10.1007/s11227-012-0803-9
23. J. Francés, S. Bleda, M.L. Álvarez, F.J. Martínez, A. Márquez, C. Neipp, A. Beléndez, (2012), vol. 8498, pp. 84,980K–84,980K–9. DOI 10.1117/12.929545. URL http://dx.doi.org/10.1117/12.929545
24. I. Corporation, *Intel 64 and IA-32 Architectures: Optimization Reference Manual* (2011)
25. K-H. Kim, K. Kim, Q-H. Park, Computer Physics Communications **182**, 1201 (2011). DOI 10.1016/j.cpc.2011.01.025. URL http://dx.doi.org/10.1016/j.cpc.2011.01.025