ARTICLE IN PRESS

Computer Physics Communications ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications

J. Francés [a,b,*], B. Otero [c], S. Bleda [a,b], S. Gallego [a,b], C. Neipp [a,b], A. Márquez [a,b], A. Beléndez [a,b]

[a] Dpto. de Física, Ingeniería de Sistemas y Teoría de la Señal, Universidad de Alicante, E-03080, Alicante, Spain

[b] Instituto Universitario de Física Aplicada a las Ciencias y las Tecnologías, Universidad de Alicante, E-03080, Alicante, Spain

[c] Dpt. d'Arquitectura de Computadors, Universitat Politènica de Catalunya, Barcelona, Spain

ARTICLE INFO

Article history:
Received 13 June 2014
Received in revised form
14 November 2014
Accepted 26 January 2015
Available online xxxx

Keywords:
FDTD
GPU computing
OMPI
SIMD extensions

ABSTRACT

The Finite-Difference Time-Domain (FDTD) method is applied to the analysis of vibroacoustic problems and to study the propagation of longitudinal and transversal waves in a stratified media. The potential of the scheme and the relevance of each acceleration strategy for massively computations in FDTD are demonstrated in this work. In this paper, we propose two new specific implementations of the bi-dimensional scheme of the FDTD method using multi-CPU and multi-GPU, respectively. In the first implementation, an open source message passing interface (OMPI) has been included in order to massively exploit the resources of a biprocessor station with two Intel Xeon processors. Moreover, regarding CPU code version, the streaming SIMD extensions (SSE) and also the advanced vectorial extensions (AVX) have been included with shared memory approaches that take advantage of the multi-core platforms. On the other hand, the second implementation called the multi-GPU code version is based on Peer-to-Peer communications available in CUDA on two GPUs (NVIDIA GTX 670). Subsequently, this paper presents an accurate analysis of the influence of the different code versions including shared memory approaches, vector instructions and multi-processors (both CPU and GPU) and compares them in order to delimit the degree of improvement of using distributed solutions based on multi-CPU and multi-GPU. The performance of both approaches was analysed and it has been demonstrated that the addition of shared memory schemes to CPU computing improves substantially the performance of vector instructions enlarging the simulation sizes that use efficiently the cache memory of CPUs. In this case GPU computing is slightly twice times faster than the fine tuned CPU version in both cases one and two nodes. However, for massively computations explicit vector instructions do not worth it since the memory bandwidth is the limiting factor and the performance tends to be the same than the sequential version with auto-vectorisation and also shared memory approach. In this scenario GPU computing is the best option since it provides a homogeneous behaviour. More specifically, the speedup of GPU computing achieves an upper limit of 12 for both one and two GPUs, whereas the performance reaches peak values of 80 GFlops and 146 GFlops for the performance for one GPU and two GPUs respectively. Finally, the method is applied to an earth crust profile in order to demonstrate the potential of our approach and the necessity of applying acceleration strategies in these type of applications.

1. Introduction

The new technology developments in computers produced in the last three decades not only has permitted new applications and faster computers, but also it has became the key point for the use of numerical methods in many scientific fields i.e. Electromagnetism, Optics, Acoustics or Astronomy amongst others. Regarding FDTD, it was not until the eighties when the scientific community shown interest in this method after his early publication in 1966 by Kane S. Yee [1]. This time gap between the publication, which established the basis of the finite difference schemes applied to solving Maxwell's equations, and the rising interest by researchers is mainly due to the evolution of modern computers. This new

* Corresponding author at: Dpto. de Física, Ingeniería de Sistemas y Teoría de la Señal, Universidad de Alicante, E-03080, Alicante, Spain. Tel.: +34 965903682; fax: +34 965909750.

E-mail address: jfmonllor@ua.es (J. Francés).

http://dx.doi.org/10.1016/j.cpc.2015.01.017
0010-4655/

scenario has allowed to develop new applications and also new formulations that cover a wide range of problems. The first attempts of FDTD in acoustics were published by Botteldooren [2], LoVetri [3], Wang [4] and its application to solid mechanics was performed by Virieux [5] and Cao [6] in the field of seismology. The application of finite-difference schemes to vibroacoustics can be performed mainly in two ways. The former is based on the discretisation of Newton's second law and Hook's law; the latter is based on the vectorial and scalar potentials derived from the two general laws of solid-mechanics mentioned below. In this work, the first solution has been considered due to its simplicity and applicability to inhomogeneous media. Interested readers on the scalar and vector potentials can find more information about it in the literature [7–10].

The unified treatment of vibrations in fluid and solid media by means of FDTD requires reduced values of the time and spatial resolutions. It is worth to note that the propagation in solids tends to be faster than in fluids such as air for instance. Moreover, FDTD computes the velocity and stress components as a function of both time and space. This characteristic of the method implies large simulation sizes, which is a common scenario in seismology or building acoustics that will also require a big number of time steps in order to ensure steady state. The reduction of spatial resolution also influences the grid size that would be greater for covering a finite extent of physical area. These factors make mandatory the application of acceleration strategies in order to reduce the time simulation costs. Regarding this aspect, some works related with GPU computing and FDTD in the field of Electromagnetics have been developed [11–13]. For FDTD and GPU computing applied to vibration problems there are some contributions related with seismology [14] and also for vibroacoustics [15]. The application of multi-GPU has been applied to FDTD and Electromagnetics in [16, 17] but an accurate performance analysis of multi-CPU FDTD code that uses SSE and AVX instructions compared to a multi-GPU version with Peer-to-Peer communication has not been carried out to the best of our knowledge.

In this work, the unified treatment of fluid and solid 2D FDTD analysis is carried out by means of a bi-processor station with two Intel Xeon E5-2360 and also with two NVIDIA GTX 670 GPUs. We focus on a hybrid approach to program multi-core based HPC systems, combining standardised programming models—MPI for distributed memory systems and OpenMP for shared memory systems. Although this approach has been utilised by many users to implement their applications, we study the performance characteristic of this approach using also vectorial instructions applied to FDTD for vibroacoustic problems. We analyse the performance of various hybrid configurations based on auto-vectorisation, implicit usage of SSE and also implicit addition of AVX instructions. More specifically, the SSE and AVX instructions are implicitly used in the algorithms and both are compared to the GPU versions. A similar analysis was performed in [18] for FDTD and Electromagnetics and using a single CPU with SSE and a GPU. In this work, an extension of that work is performed adding the AVX instructions and the effect of considering distributed memory approaches with Open MPI and Peer-to-Peer communications in CUDA GPUs. In addition, each CPU version exploits the resources of each core available in multi-core processor by means of a shared memory approach such as Open MP. As a result of this set up, an accurate and realistic comparison of the FDTD implementations is carried out between a highly tuned multi-CPU and a multi-GPU. The results here presented give evidence of that shared memory approaches such as OMPI can significantly improve the performance of 2D schemes that usually are not high memory demanding.

This paper is organised as follows: Section 2 gives a brief review of the numerical method and the programming techniques utilised in this work. Section 3 introduces the acceleration strategies followed for optimising the 2D FDTD scheme and show the proposals of the multi-CPU and multi-GPU code version implemented. Finally, the comparisons of the multi-CPU and multi-GPU with the application of FDTD to some examples are presented in Section 4. Conclusions are drawn in Section 5.

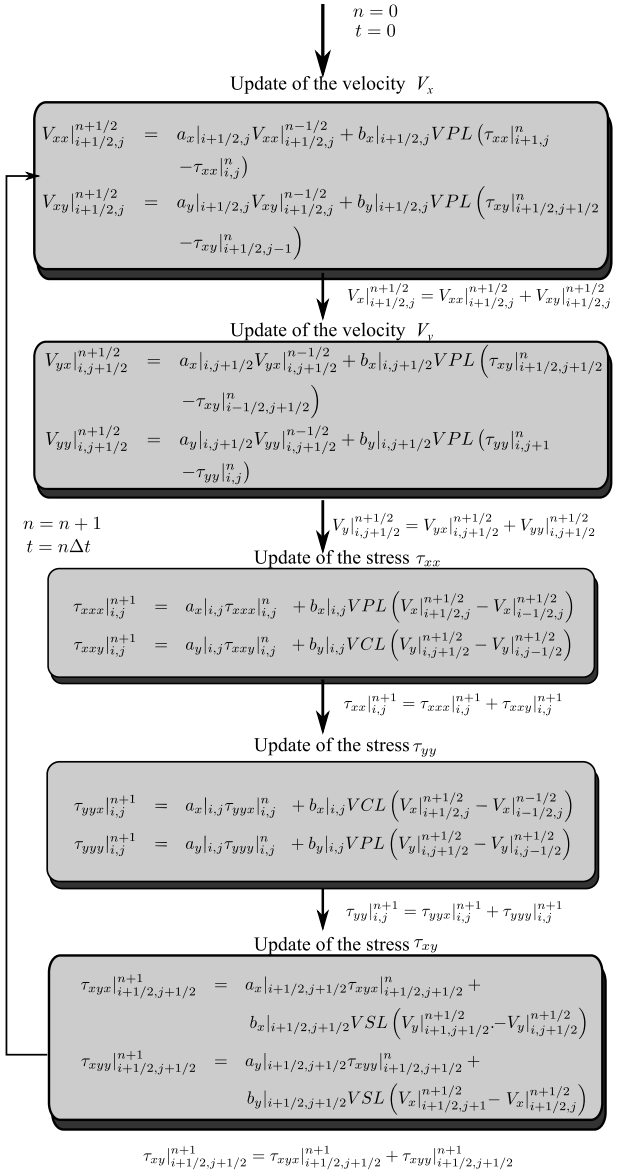

Fig. 1. Diagram of FDTD scheme for solving vibroacoustic problems.

2. Background review

2.1. Finite-difference time-domain method

In this work a modified set of equations based on a scaling of the velocity components are considered. The development of these equations is fully explained in [12]. The formulation and the update process of the finite difference time equations for this case are summarised in Fig. 1.

Some of the parameters in Fig. 1 are here explained

$$VPL = \frac{\Delta t c_p}{\Delta u}, \qquad VSL = VPL\left(\frac{c_s}{c_p}\right)^2, \qquad VCL = \frac{\Delta t \lambda}{\Delta u Z_0} \tag{1}$$

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

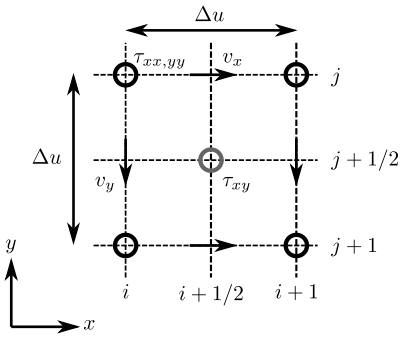

Fig. 2. FDTD lattice for solids.

where $V_i = v_i Z_0$ with $i = x, y$ is the normalised particle velocity and $Z_0 = \rho c_p = \sqrt{\rho\,(\lambda + 2\mu)}$ is the characteristic impedance of the medium. 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. 2). From Fig. 2(a) staggered distribution of the velocities and stresses along the Yee cell can be appreciated. This arrangement implies a simple average over the surrounding cells of the physical parameters that define the media on the boundaries between different kinds of materials [19].

The parameters a_i and b_i with $i = p, s$ introduce the effects of longitudinal (p-waves) and transverse (s-waves) losses in solid material [12]. As can be seen in the set of equations shown in Fig. 1 the operational intensity of the method is expected to be low since several data accesses should be done in order to update each field component. Thus, it is bound to the characteristic memory limiting factor of FDTD.

2.2. SIMD extensions: SSE and AVX instructions

The vector instructions have been introduced and gradually extended in multiple processor generations [18,20,21]. The SSE family vector instructions use 16 dedicated registers of 128 bits and were considered in this work since they are available in most modern processors. Programming AVX is quite similar varying vector length to 256 bits and also modifying slightly the instruction names respect SSE. Fig. 3 shows an illustration showing the paradigm of vector instructions compared with the standard way of programming sequential codes. As can be seen using vector instructions arithmetic operations can be performed in parallel using the same time that a single standard operation is carried out. There are different ways to use vector instructions in an application. In this work two alternatives have been considered. Firstly, an automatic vectorisation by the compiler that implies an arranging of the code so that the compiler can recognise possibilities for vectorisation. In `gcc`, vectorisation is enabled by `-O3`. This strategy is easy to use since no changes to the program code are needed. However, the code vectorisation is not guaranteed. Secondly, the usage of compiler intrinsic functions for vector operations in which functions that implement vector instructions are included in C++ language. This solution requires to exactly specify the operations that should be done on the vector values being closer to a low-level language programming. In both cases a specific memory alignment must be ensured. In order to successfully apply vector instructions, load operations must be done under a set of aligned bytes [21]. More specifically, arrays used in vector instructions should be 16-byte aligned at least since accessing to unaligned vector elements will fail.

2.3. Programming models

2.3.1. OpenMP

Open Multi-Processing (OpenMP) is a shared memory architecture API that provides multi thread capacity [22]. OpenMP is a portable approach for parallel programming on shared memory systems based on compiler directives, that can be included in order to parallelise a loop. In this way, a set of loops can be distributed along the different threads that will access to different data allocated in local shared memory. One of the advantages of OpenMP is its global view of application memory address space that allows relatively fast development of parallel applications with easier maintenance. However, it is often difficult to get high rates of performance in large scale applications. Although, in OpenMP a usage of threads ids and managing data explicitly as done in an MPI code can be considered, it defeats the advantages of OpenMP.

2.3.2. MPI

Message Passing Interface (MPI) is a specification for message passing operations [23]. In this case, MPI considers processes, instead of threads with in the OpenMP programming model. Since the functional behaviour of MPI calls is also standardised, your MPI calls should behave the same regardless of the implementation, thus ensuring the portability of your parallel programs. Performance, however, may vary from each implementation of the MPI standard. In this work we use the Open MPI [24] implementation since is a free software and it is available for many platforms i.e. Microsoft Windows and UNIX (including Linux and Mac OS X). Open MPI is usually included in cluster systems since it is a message-passing programming language. The most important advantages of this model are twofold: achievable performance and portability. A MPI program requires carefully thought out strategies for partitioning the data and managing the resultant communication. Secondly, global data must be duplicated for performance reasons, due to the distributed nature of the model. Thirdly, special considerations must be taken under account with MPI jobs running on large parallel systems since machine instabilities and inhomogeneous usage of resources such as memory and network can affect the overall performance.

2.3.3. Hybrid: OpenMP-MPI

The MPI library is often used for parallel programming in cluster systems since it is a message-passing programming language [25]. However, MPI is not the most appropriate programming model for multicore computers because of the fact that in many cases all slave processes must communicate directly with the master MPI process to receive new tasks. In large cluster environments, this set up can produce a bottleneck on system performance due to an excessive amount of communication. Therefore, hybrid schemes are commonly used in order to optimise applications in both

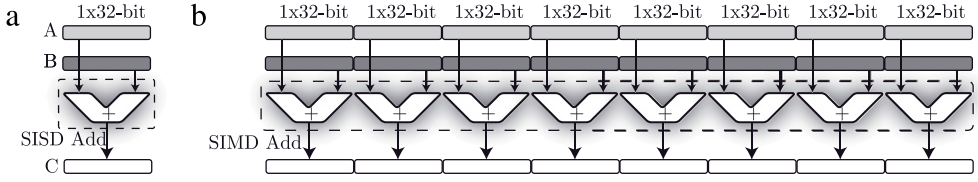

Fig. 3. (a) Scheme of a sequential arithmetic addition. (b) Scheme of an addition with AVX instructions.

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

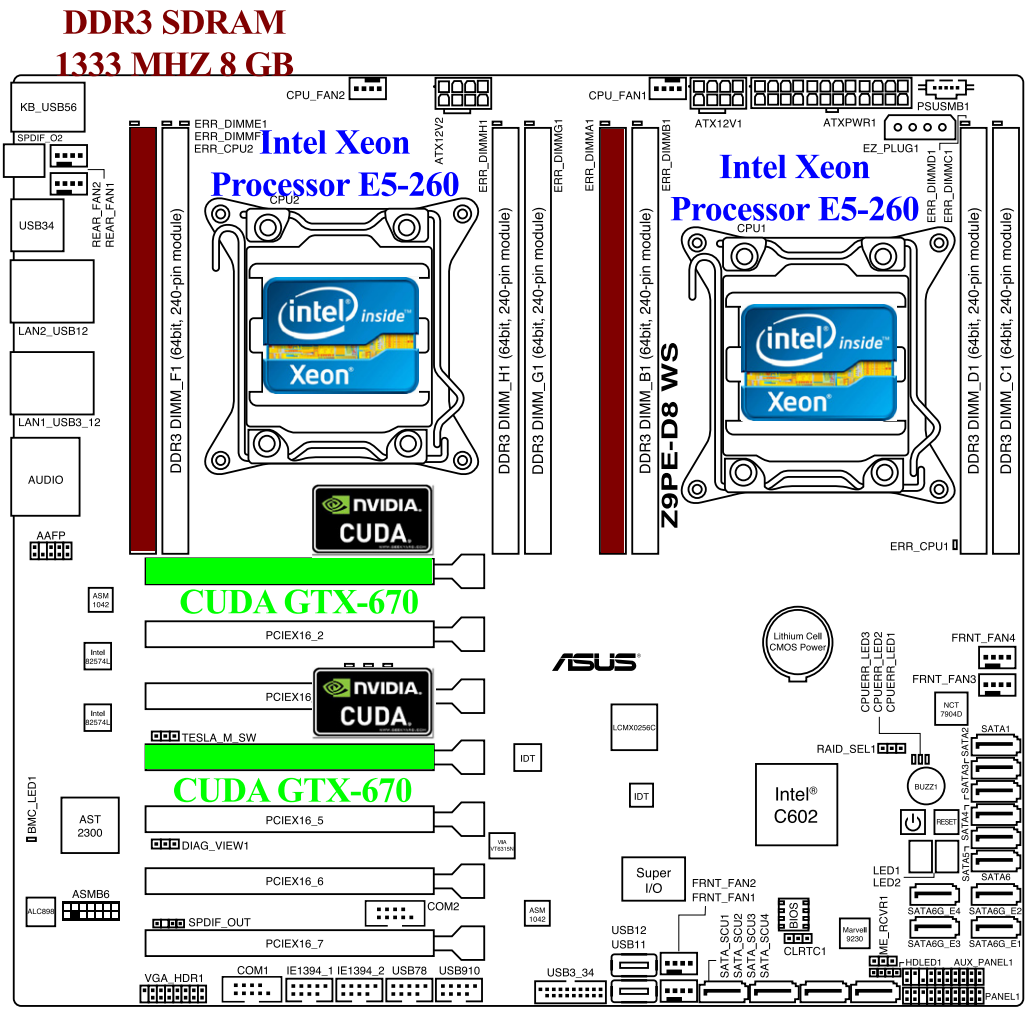

Fig. 4. Scheme of the biprocessor station used in this work.

levels shared and distributed memory [26,27]. This means that a MPI process can exploit the resources in its processor by means of OpenMP directives. In this work, this scheme is considered including also SSE and AVX instructions in the shared memory level in order to optimise all resources available in modern processors.

2.3.4. CUDA

CUDA (an acronym for Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA [28]. The unit of execution in CUDA is called a thread. Each thread executes the kernel by the streaming processors in parallel. In CUDA, a group of threads that are executed together is called thread blocks, and the computational grid consists of a grid of thread blocks. Additionally, a thread block can use the shared memory on a single multiprocessor as while as the grid executes a single CUDA program logically in parallel. Thus in CUDA programming, it is necessary to design carefully the arrangement of the thread blocks in order to ensure low latency and a proper usage of shared memory, since it can be shared only in a thread block scope. The effective bandwidth of each memory space depends on the memory access pattern. Since the global memory has lower bandwidth than the shared memory, the global memory accesses should be minimised. The global memory space is not cached, so it is important to follow the right access pattern to achieve maximum memory bandwidth. In recent revisions of the CUDA architecture new approaches for sharing global memory have been included. The communication between GPUs is possible by means of Peer-to-Peer communications. This approach is similar to that one offered by Open MPI, in which a message with data can be transferred between GPUs. Peer-to-Peer communication reduces the bottleneck produced by this communication since it avoids the inclusion of the CPU in this process. The PCIE bus is directly used between GPUs taking advantage of its high bandwidth. In this work two GPUs connected to the same PCIE bus are used in order to reduce the overall time processing costs in parallel applications.

3. Computational optimisation

3.1. Hardware scenario

In this section the strategies considered for accelerating FDTD for solid–fluid vibration analysis are detailed. In this work two Intel Xeon E5-2630 processors with 15 MB of cache, a clock speed of 2.3 GHz and the possibility of handle efficiently twelve threads each have been used. Regarding GPU computing, also two GTX660 GPU with Kepler architecture are considered. Fig. 4 shows the technical scheme with the two CPUs and the two GPUs considered in this work. The CUDA devices with compute capability 2.0 and higher can address each others memory via PCIE bus avoiding memory transfers between GPU and CPU host. This system hardware is considered for accelerating the parallel implementation of the FDTD for solid–fluid vibration analysis summarised in Fig. 1.

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

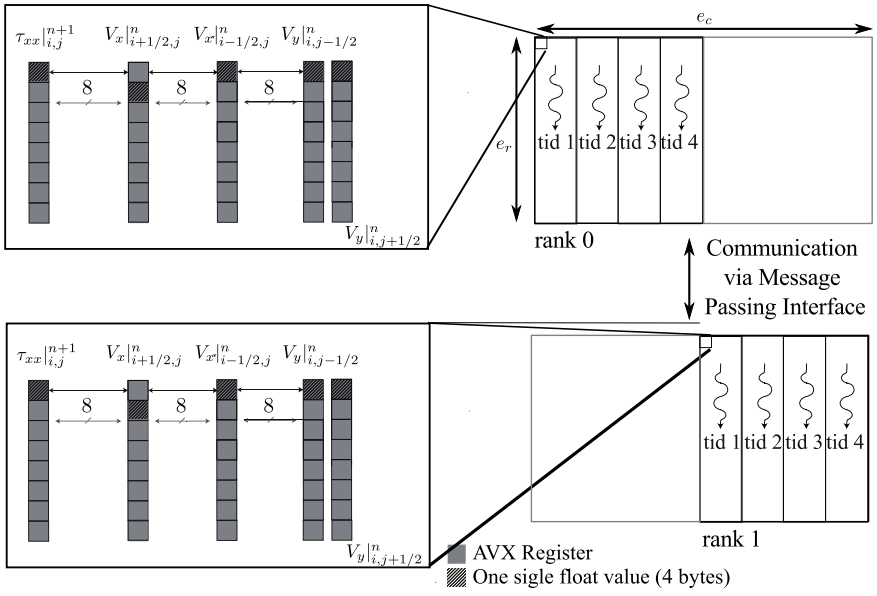

Fig. 5. Illustration showing the scheme for solving τ_{xx} component in two CPU using Open MPI, OpenMP and AVX vector instructions.

3.2. Implementations

3.2.1. Multi-CPU approach of the FDTD method

In this work, an hybrid approach has been considered in order to exploit all resources available in architectures with multiple CPUs such as that shown in Fig. 4. More specifically, OMPI is in charge of splitting the computational task in two parts that are assigned to the microprocessors available in Fig. 4. Each process takes advantage of the multi-core architecture of Intel Xeon processors by means of OpenMP whereas vector instructions (SSE and AVX) are also used for each thread. This scheme is illustrated in Fig. 5 specifically for AVX instructions. Note that the scheme is also valid for SSE modifying the number of registers used (4 instead of 8). Both SSE and AVX instructions follow the parallel computation model and is the most cost-effective way of accelerating single- or double-point performance in modern processors.

Here, only single precision has been considered for FDTD simulations, since single precision is accurate enough for FDTD applications. It is worth to note that the scaling factor and the relationship between spatial and time resolutions have been carefully chosen in order to avoid rounding and finite precision errors [15]. The usage of double precision was experimentally proven not to improve significantly the accuracy of the results obtained but it implied a dramatic downside effect in terms of computational resources.

In order to successfully apply the vector 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++ [13]. Thus, each position is reached taking into account that the storage order is by columns. For simplicity, the scheme of how the update of the stress component in x-axis is carried out by means of the AVX registers and OpenMP is shown in Fig. 5. The updating process for the rest of the components follow the same scheme. To simplify the notation, also the PML notation has been suppressed but they are also included in the optimisation process. For instance, the update of τ_{xx} requires several aligned loads that store 8 consecutive values of the terms involved: $V_x|^n_{i+1/2,j}$, $V_x|^n_{i-1/2,j}$, $V_y|^n_{i,j-1/2}$, $V_y|^n_{i,j+1/2}$. Note that the physical parameters that define the media are also stored in the vector registers. The next step is to perform the arithmetic operations by means of the vector registers. For the specific case of AVX some of the intrinsics functions used are: `_mm256_sub_ps(__m256 a, __m256 b)`, `_mm256_add_ps(__m256 a, __m256 b)` or `_mm256_mul_ps(__m256 a, __m256 b)` for instance. Summarising, OpenMP has been considered in order to parallelise the updating of each field component. Since, modern CPUs contain several cores the computational load has been distributed along them. As can be seen in Fig. 5, each thread is in charge of computing a set of columns of each field component, and each thread uses extensively the vector AVX instructions along the rows direction. The whole grid is split in two sections which are computed by different processes. The data dependencies are solved after updating each field component. For instance, here the code related with the transmission of the messages related with the updating of τ_{xx} component is shown:

```
#ifdef OPEN_MPI
  if (rank > 0){
    MPI::COMM_WORLD. Send(txx,rows, MPI::FLOAT,
        rank - 1, 2);
  }

  if (rank < numprocs-1){
    MPI::COMM_WORLD. Recv(&txx[local_col_*
        rows], rows,
    MPI::FLOAT,rank + 1, 2, Stat_);

  }
#endif}
```

This code shows the scheme used for sending and receiving the messages related with the updating of dependent data in different processes. It can be seen that all processes with a nonzero identifier (`rank`), which means that is not the master application, will send their first column of $\tau_{xx}|^{n+1}_{0,j}$ that have been recently updated in the prior process. In other words, a process with rank n will send the updated data to the rank $n-1$. This process will be ready for receiving this data and it will be stored in the last column of the matrix $\tau_{xx}|^{n+1}_{e_c/np,j}$. This process ensures that at the end of the field updating all processes have the latest data computed in their memory. This process can be extrapolated to the other components of the computation, being used at the same time as barriers for synchronisation. In next subsection more information about this process is detailed.

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

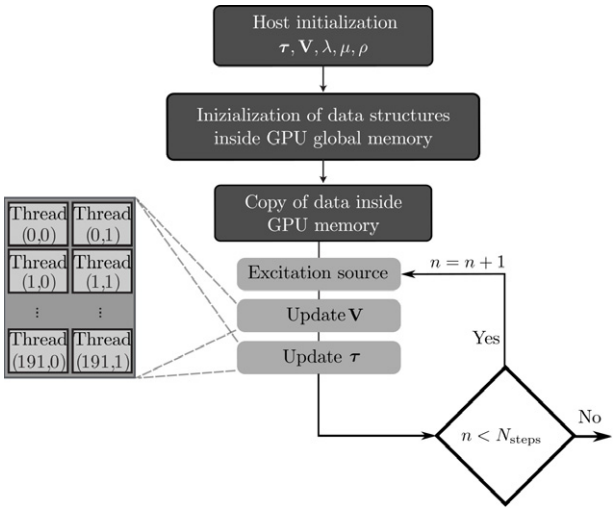

Fig. 6. Arrangement scheme of the grid of blocks invoked by CUDA Kernels with a detail of the threads that conform each block. Flowchart of GPU parallel programming for FDTD for fluid–solid vibration analysis.

3.2.2. Multi-GPU approach of the FDTD method

Regarding the 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 is launched per block [29]. This block size has been experimentally chosen and provides the best results for our application. This arrangement has been previously used in FDTD schemes providing successful results [12,13,18]. In these works was demonstrated that FDTD schemes relying on auto-caching feature of the GPU, instead of a direct shared memory approach, are enough to take advantage of the L1 cache memory. The scheme proposed is illustrated in Fig. 6 in which a detail of the threads and their arrangement inside the grid of blocks is shown.

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. Firstly, an initialisation in the host of the fields to be computed is performed. Secondly, the allocation of these components 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 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. The speed up is defined as the ratio between the parallel time and best sequential time costs of the same application. This parameter provides an estimation of how many times the parallel application is faster compared to the sequential code.

Regarding the 2 GPU version, the kernels focused on solving each field component are asynchronous, thus working in parallel in both GPUs. The synchronisation between GPUs is included by means of the synchronous `cudaMemcpyPeer` function that is in charge of updating the dependent data between GPUs. This means that both GPU will wait until the P2P communication ends for updating the next set of field components. For instance, GPU 1 needs the updated values of the first column of the x component of the velocity ($V_x|^{n+1/2}_{1/2,j}$). These values are computed by the GPU 0 ($V_x|^{n+1/2}_{e_c/2+1/2,j}$). Similarly, the same process must be performed for the last column of the y component of the velocity ($V_y|^{n+1/2}_{e_c/2,j+1/2}$) from the GPU 0, that must be sent to GPU 1 for updating the first column ($V_y|^{n+1/2}_{0,j+1/2}$). This process is performed once the velocity components are updated, since the updated values of velocities are needed for updating the new values of stress. For this case, once the stress component is obtained the GPU 1 must send $\tau_{xx}|^{n+1}_{0,j}$ to the GPU 0, that will store these values in $\tau_{xx}|^{n+1}_{e_c/2,j}$. The last column of $\tau_{xy}|^{n+1}_{e_c/2+1/2,j+1/2}$ in GPU 0 is also transferred to GPU 1 and stored in $\tau_{xy}|^{n+1}_{1/2,j}$. This scheme has been tested with the NVIDIA Profiler achieving a compute utilisation greater than 93% for both GPUs. The measurement of the bandwidth between global memory and the GPU close to 185 GB/s reveals that the speed of global memory access is the limiting factor, since the data-sheet of the GTX 670 provides an upper bandwidth of 192 GB/s. Moreover, the measurement of the averaged throughput of 2.5 GB/s between devices also corroborates the validity of the implementation here presented.

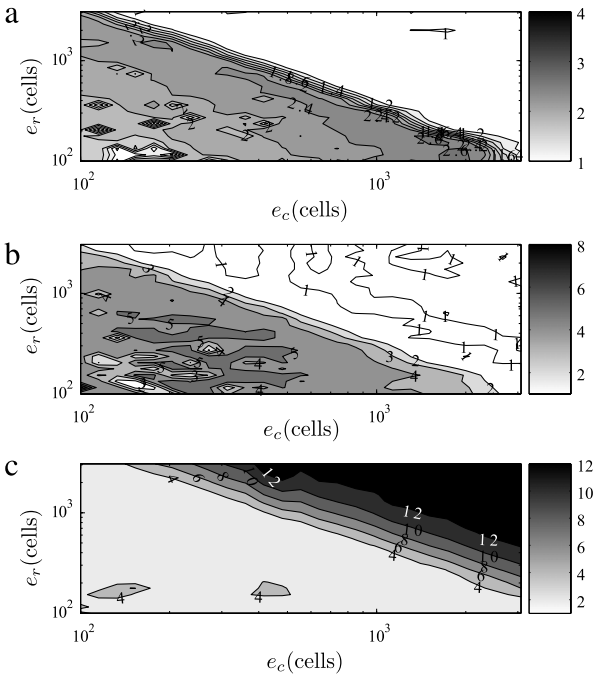

Fig. 7. Computational results as a function of the number of cells for 1 CPU and 1 GPU. (a) Speed up for the SSE+OpenMP version. (b) Speed up of the AVX+OpenMP version. (c) Speed up of the CUDA version.

4. Results

Firstly, the computational results for a single CPU and GPU are summarised in Fig. 7. The simulation grid is modified as a function of the number of rows (e_r) and columns (e_c). More specifically, Fig. 7 shows the speed up of the SSE+OpenMP (a), AVX+OpenMP (b), and CUDA (c) being the best sequential code the one obtained by means of autovectorization and also the addition of OpenMP directives. Although, this sequential code is not strictly sequential, it has been considered as the best implementation without any vector instructions or other acceleration strategies that imply a dramatic change in the programming paradigm. This assumption provides a more accurate analysis of the degree of improvement achieved by the accelerated versions, since they are compared with the best sequential code that a standard programmer could obtain with its sequential code. Fig. 7(a) shows the speed up for the SSE+OpenMP code, a maximum speed up close to 3 is obtained.

Fig. 7(b) represents the speed up for the AVX+OpenMP version. In this case the speed up is significantly improved respect of the

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

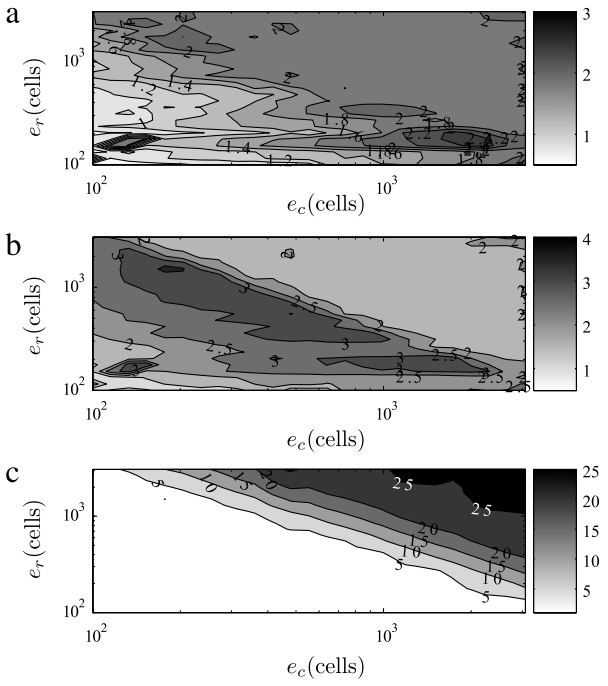

Fig. 8. Computational results as a function of the number of cells for 2 CPU and 2 GPU. (a) Speed up for the SSE+OpenMP+OMPI version. (b) Speed up of the AVX+OpenMP+OMPI version. (c) Speed up of the CUDA Peer-to-Peer version.

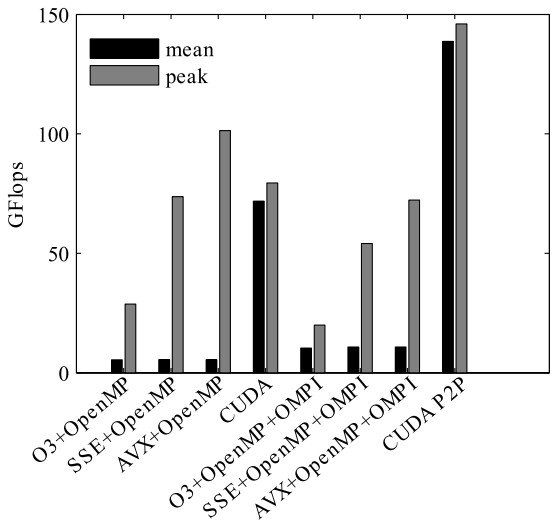

Fig. 9. Performance in GFLOPS for the FDTD implementations developed.

SSE code, being up to 8 in the best case. In both cases, the best performance is achieved in a region of relatively small simulation sizes. This behaviour is closely related with the size of the cache memory of the processors. For bigger simulation sizes, the amount of memory needed for storing all data is also increased, producing more cache misses and degrading the overall performance in vector instruction as well. As can be seen in Fig. 7(a)–(b) the upper right corner of both graphs shows that the performance of both SSE and AVX extensions do not provide a significant improvement compared to the autovectorized sequential version with also OpenMP. On the other hand, Fig. 7(c) illustrates the speed up of the CUDA version. The performance behaviour in this case varies in a significant way, since as can be seen the maximum speed up is obtained for high demanding computations with bigger simulation sizes. The overall performance of a GPU is more homogeneous than the CPU versions. However, for moderate simulation sizes the GPU is no more than 2 times faster than the sequential CPU version.

Secondly, the results derived from multi-CPU and multi-GPU are shown in Fig. 8. Following the analysis described the speed up for the vectorial code versions in two CPUs and the CUDA Peer-to-Peer implementation are shown in Fig. 8. For both the SSE and AVX versions there is a slight reduction of the performance in the lower area of both graphs (smaller simulation sizes). Whereas, there is a significant increase for high demanding simulations. This increase tend to be close to the double for the case of using a single CPU. Regarding the implementation of two GPUs, the speed up has risen up to 25 compared to the sequential version, being also close to be two times faster than the version using only one GPU.

Thirdly, the performance in GFLOPS for all code versions here presented is given in Fig. 9. For understanding properly the effect of computing with 2 CPUs and 2 GPUs no only the mean performance is given also the peak performance is included. The analysis of Fig. 9 gives some interesting evidences. For instance, the difference between peak and mean performance is considerably higher in CPUs than in GPUs, which behaves more homogeneous in all cases. Moreover, the 2 CPUs versions achieve an increment of the mean performance closer to be the double than the single CPU version, although the peak performance is dramatically reduced using two CPUs. This effect could be considered as a drawback of distributed memory approaches, but accelerating the method specifically for high demanding simulations is more relevant than achieving higher peaks in the performance for small simulation sizes.

For the specific case of multi-GPU computing, the homogeneous behaviour of this platforms is also corroborated. In both cases single GPU and two GPUs the peak and mean performance values are rather similar and the ratio of improvement of using two GPUs is also closer to be the double.

In spite of the fact that GPU computing in many cases is the best option, Fig. 10 shows the comparison of GPUs (single and Peer-to-Peer version) compared to the best CPUs versions. Note that in this specific case the relative speed up metric is used since the reference time cost is not a strictly the time simulation from a sequential version. In both cases the best option for CPU computing is that one that uses extensively the vector instructions AVX with OpenMP. Although, the relative speed up values between GPU and CPU in both cases (single and two CPUs and GPUs) remains more or less the same, a slight displacement of the upper values of the graphs can be identified. This effect demonstrates that using shared memory approaches extend in a significant value the region in which the CPU computing can be competitive compared to GPU computing.

Finally, an example of a simulation of a multi-layered structure similar to a section of the earth crust with a coast profile including water is included in Fig. 11. In this case a dense grid is used, thus GPU computing has been considered in order to reduce the overall time simulation costs up to 5 min with the Peer-to-Peer CUDA version. The parameters of the different layers are summarised in Table 1 and have been extracted from [30,31]. Fig. 11 shows a sequence of the simulation performed for different time steps. The aim of this simulation is to illustrate the complexity of these type of analysis, in which multiple reflections in the boundaries between layers are produced. It is worth to note that the amplitude of pressure is represented by means of a colour map and the velocity by means of arrows. The effect of reflections of different wavefronts in the coast profile can be easily identified in Fig. 11(d) in which the pressure and velocity are larger compared to the solid material in that time step. The delay between waves permit to obtain many parameters such as the situation of the epicentre for instance. In this specific case, an impact has been introduced in the deepest layer (−1900, −800) that corresponds approximately to 18.7 km from the surface. From Table 2 is easy to derive the physical area

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

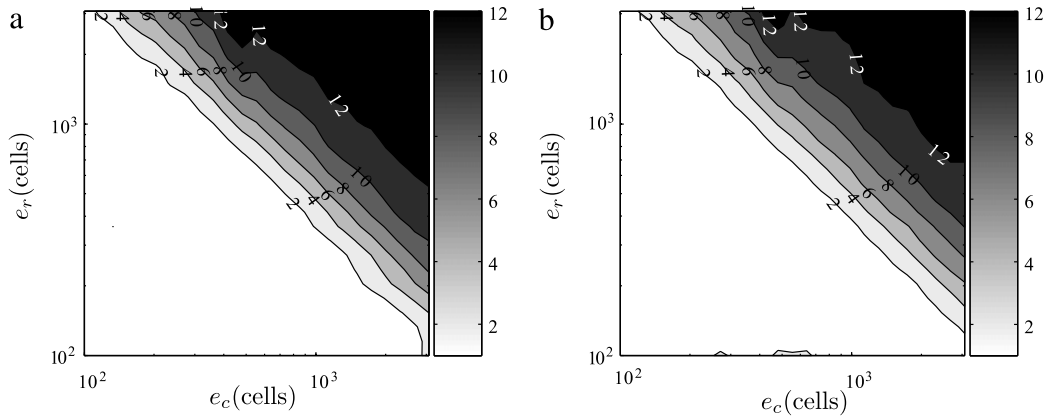

Fig. 10. Relative speed up between the GPU and the AVX code version over CPU. (a) 1 CPU vs. 1 GPU. (b) 2 CPUs vs. 2 GPUs.

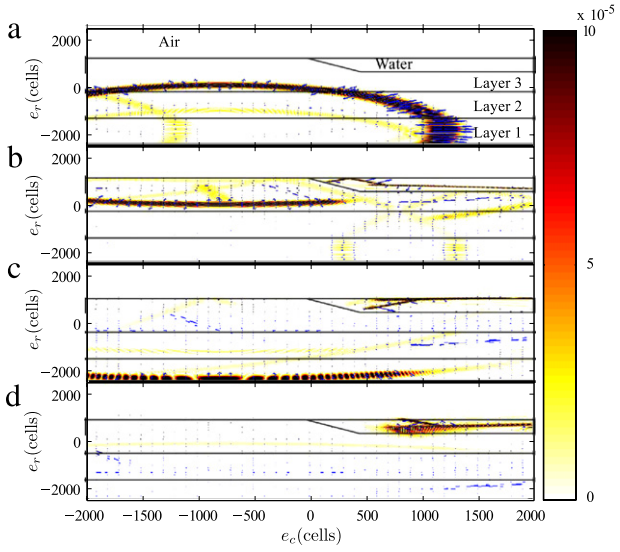

Fig. 11. Simulation sequence of the propagation of elastic waves along a stratified media. (a) $n = 5000$. (b) $n = 10\,000$. (c) $n = 15\,000$. (d) $n = 20\,000$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 1
Parameters of the materials used for obtaining the results in Fig. 11. The elastic parameters of air are given: $\lambda_0 = -0.142$ MPa, $\mu_0 = 0$ Pa and $\rho_0 = 1.21$ kg/m^3.

Material	λ	μ	ρ
Layer 1	52.22	21.00	2700
Layer 2	60.48	24.20	2900
Layer 3	74.50	29.80	2900
Water	2.3	0	1000

that is being simulated (21.25×17.00 km^2) and the whole time window being analysed (6.2 s).

Several analysis could be obtained from the results shown in Fig. 11 that would be out of the aim of this work. The target of this result is to show the potential of this method and the application of the acceleration strategies here presented for the simulation of high demanding applications such as those required in seismology. The data shown in Table 2 demonstrates the huge computational requirements of this kind of simulations.

5. Conclusions

In this work a unified scheme for FDTD analysis of vibrations on fluid and solid media has been accelerated in both CPU and GPU. For CPU computing vector instructions such as SSE and AVX were implicitly used in the algorithm with OpenMP and OMPI for the two CPUs version. Regarding GPU computing, two CUDA versions have been implemented one using a single GPU and another one that takes advantage of two GPUs connected to the same bus PCIE by means of Peer-to-Peer communication. The results reveal that a fine-tuned CPU version can be competitive in a wide range of situations. Furthermore, the results here presented show evidences of that shared memory approaches such as OMPI can significantly improve the performance of 2D schemes that usually are not high memory demanding. However, GPU computing gives the best performance for high demanding computing situations such as big simulation sizes with also a huge number of time steps.

Table 2
FDTD setup for the results shown in Fig. 11. The default units for the parameters here listed are measured in cells.

f_{max} (Hz)	Δ (m)	Δt (ms)	e_r	e_c	m_{PML}	N_{steps}
40	4.25	0.31	5000	4000	40	20 000

In order to accurately compare CPU and GPU computing, a comparison of both platforms with one and two nodes has been performed obtaining that in all cases the AVX extensions give the best performance in simulations that fit on cache memory. However, for high demanding simulations it has been demonstrated that there is no substantial effects on adding this instructions implicitly in FDTD codes compared to a fine-tuned sequential version with auto-vectorisation and also shared memory approaches. The addition of a second CPU with vector instruction reduces the maximum peak performance dramatically but increases significantly the overall performance of the implementation also for higher simulation sizes. This increase is close to be the double, thus validating our implementation. The same trend can be also identified for one and two GPUs, but it can be concluded that the impact of adding an extra CPU is larger than the effect of adding an extra GPU. This effect has been demonstrated obtaining the relative speed up between GPU and AVX CPU versions. The number of simulations in which the relative speed up is moderate becomes larger, thus implying that adding extra CPUs helps to use more efficiently the cache memories and thus helping to improve the performance of vector instructions. However, since an interesting trend has been corroborated with the comparison of CPU and GPU computing, the multi-GPU version becomes the best option for HPC since achieve upper performance limits closer to 146 GFLOPS. However, the authors consider mandatory to compare accurately these kind of platforms with a CPU version that takes advantage of its resources, hence both the standard and relative speedup here given are realistic in terms of how many times GPU computing is faster than CPU computing.

It is worth to mention that the averaged performance for large scale simulations achieved with the CPU versions establish an

Please cite this article in press as: J. Francés, et al., Multi-GPU and multi-CPU accelerated FDTD scheme for vibroacoustic applications, Computer Physics Communications (2015), http://dx.doi.org/10.1016/j.cpc.2015.01.017

upper value in these kind of applications being close to 5.5 and 11 GFLOPS with AVX and OpenMP for a single and two CPUs respectively.

The authors consider that the results here presented can be extrapolated to 3D in a straightforward manner. Therefore, as future works the authors are considering to corroborate the results here presented for 3D FDTD computing, to study the effect of double precision and also to include novel parallel approaches such as Intel Phi processors.

Acknowledgements

The authors would like to thank Sergio Orts Escolano for fruitful discussions of the GPU programming. 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, ISIC/2012/013 and GV/2014/076 and by the "Universidad de Alicante" of Spain under project GRE12-14.

References

[1] K. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, IEEE Trans. Antennas and Propagation 14 (3) (1966) 302–307. http://dx.doi.org/10.1109/TAP.1966.1138693.
[2] D. Botteldooren, Finite-difference time-domain simulation of low-frequency room acoustic problems, J. Acoust. Soc. Am. 98 (6) (1995) 3302–3308. http://dx.doi.org/10.1121/1.413817.
[3] J. LoVetri, D. Mardare, G. Soulodre, Modeling of the seat dip effect using the finite-difference time-domain method, J. Acoust. Soc. Am. 100 (4) (1996) 2204–2212. http://dx.doi.org/10.1121/1.417929.
[4] S. Wang, Finite-difference time-domain approach to underwater acoustic scattering problems, J. Acoust. Soc. Am. 99 (4) (1996) 1924–1931. http://dx.doi.org/10.1121/1.415375.
[5] J. Virieux, P-SV wave propagation in heterogeneous media: velocity-stress finite-difference method, Geophysics 51 (4) (1986) 889–901. http://dx.doi.org/10.1190/1.1442147.
[6] S. Cao, S. Greenhalgh, Finite-difference simulation of P-SV-wave propagation: a displacement-potential approach, Geophys. J. Int. 109 (3) (1992) 525–535. http://dx.doi.org/10.1111/j.1365-246X.1992.tb00115.x.
[7] M. Sato, Y. Takahata, M. Tahara, I. Sakagami, Expression of acoustic fields in solids by scalar and vector velocity potentials, in: Ultrasonics Symposium, 2001 IEEE, Vol. 1, 2001, pp. 851–854. http://dx.doi.org/10.1109/ULTSYM.2001.991853.
[8] M. Sato, Formulation of the FDTD method for separating the particle velocity vectors of an elastic wave field into longitudinal and shear wave components, Acoust. Sci. Technol. 25 (5) (2004) 382–385. http://dx.doi.org/10.1250/ast.25.382.
[9] M. Sato, Comparing three methods of free boundary implementation for analyzing elastodynamics using the finite-difference time-domain formulation, Acoust. Sci. Technol. 28 (1) (2007) 49–52.
[10] J. Francés, J. Ramis, J. Vera, A 3D FDTD scheme for analysis of the elastic wave fields in solids, in: Proceedings of the ICSV16, 5–9 July, Kraków, Poland, 2009, pp. 1–8.
[11] A. Shahmansouri, B. Rashidian, GPU implementation of split-field finite-difference time-domain method for Drude–Lorentz dispersive media, Prog. Electromagnetics Res. 125 (2012) 55–77.
[12] J. Francés, S. Bleda, M. Lázara, F.J. Martínez, A. Márquez, C. Neipp, A. Beléndez, Acceleration of split-field finite difference time-domain method for anisotropic media by means of graphics processing unit computing, Opt. Eng. 53 (1) (2013) 011005-1–011005-10.
[13] J. Francés, S. Bleda, S. Gallego, C. Neipp, A. Márquez, Pascual, A. Beléndez, Development of a unified FDTD-FEM library for electromagnetic analysis with CPU and GPU computing, J. Supercomput. 64 (1) (2013) 28–37.
[14] T. Okamoto, H. Takenaka, Large-scale simulation of seismic-wave propagation of the 2011 Tohoku-Oki M9 earthquake, in: Proceedings of the International Symposium on Engineering Lessons Learned from the 2011 Great East Japan Earthquake, 2011, pp. 349–360.
[15] J. Francés, S. Bleda, A. Márquez, C. Neipp, S. Gallego, B. Otero, A. Beléndez, Performance analysis of SSE and AVX instructions in multi-core CPUs and GPU computing on FDTD scheme for solid and fluid vibration problems, J. Supercomput. 70 (2) (2014) 514–526. http://dx.doi.org/10.1007/s11227-013-1065-x.
[16] L.-G. De, K. Li, F.-M. Kong, Y. Hu, Parallel 3D finite-difference time-domain method on multi-GPU systems, Internat. J. Modern Phys. C 22 (02) (2011) 107–121. http://dx.doi.org/10.1142/S012918311101618X.
[17] M.R. Zunoubi, J. Payne, M. Knight, FDTD multi-GPU implementation of Maxwell's equations in dispersive media, in: Proc. SPIE 7897, Optical Interactions with Tissue and Cells XXII, 78971S, February 18, 2011. http://dx.doi.org/10.1117/12.875528.
[18] J. Francés, S. Bleda, C. Neipp, A. Márquez, I. Pascual, A. Beléndez, Performance analysis of the FDTD method applied to holographic volume gratings: multi-core CPU versus GPU computing, Comput. Phys. Comm. 184 (3) (2013) 469–479. http://dx.doi.org/10.1016/j.cpc.2012.09.025.
[19] N.J. González, Simulación de tejidos vegetales mediante diferencias finitas (Master's thesis), EPSG-UPV, 2009.
[20] S. Páll, B. Hess, A flexible algorithm for calculating pair interactions on SIMD architectures, Comput. Phys. Comm. 184 (12) (2013) 2641–2650. http://dx.doi.org/10.1016/j.cpc.2013.06.003.
[21] S. Thakkur, T. Huff, Internet streaming simd extensions, Computer 32 (12) (1999) 26–34.
[22] OpenMP Application Program Interface, 4th edition, July 2013. URL: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.
[23] MPI: A Message-Passing Interface Standard, 3rd Edition, September 2012. URL: http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf.
[24] E. Gabriel, G.E. Fagg, G. Bosilca, T. Angskun, J.J. Dongarra, J.M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R.H. Castain, D.J. Daniel, R.L. Graham, T.S. Woodall, Open MPI: goals, concept, and design of a next generation MPI implementation, in: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, 2004, pp. 97–104.
[25] R. Dolbeau, S. Bihan, F. Bodin, HMPP: a hybrid multi-core parallel programming environment, in: Proceedings, Workshop on General Purpose Processing on Graphics Processing Units, Boston, Massachussets, USA, 2007, pp. 1–5.
[26] P. Alonso, R. Cortina, F. Martínez-Zaldívar, J. Ranilla, Neville elimination on multi- and many-core systems: OpenMP, MPI and CUDA, J. Supercomput. 58 (2) (2011) 215–225. http://dx.doi.org/10.1007/s11227-009-0360-z.
[27] C.-T. Yang, C.-L. Huang, C.-F. Lin, Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters, Comput. Phys. Comm. 182 (1) (2011) 266–269. http://dx.doi.org/10.1016/j.cpc.2010.06.035.
[28] NVIDIA Corporation, Whitepaper NVIDA's Next Generation CUDA™ Compute Architecture: Kepler™, 1st ed., 2012.
[29] J. Francés, S. Bleda, M.L. Álvarez, F.J. Martínez, A. Márquez, C. Neipp, A. Beléndez, Analysis of periodic anisotropic media by means of split-field FDTD method and GPU computing, in: Proc. SPIE 8498, Optics and Photonics for Information Processing VI, 84980K, October 15, 2012. http://dx.doi.org/10.1117/12.929545.
[30] A. Manglik, A.V. Thiagarajan, A.V. Mikhailova, Y. Rebetsky, Finite element modelling of elastic intraplate stresses due to heterogeneities in crustal density and mechanical properties for the Jabalpur earthquake region, central India, J. Earth Syst. Sci. 117 (2) (2008) 103–111.
[31] L. Andersen, Linear Elastodynamic Analysis, Aalborg University, 2006.