Computers and Electrical Engineering 46 (2015) 190–201

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Evaluation of the 3-D finite difference implementation of the acoustic diffusion equation model on massively parallel architectures ☆

Mario Hernández [a,c], Baldomero Imbernón [b], Juan M. Navarro [b], José M. García [a], Juan M. Cebrián [a], José M. Cecilia [b,*]

[a] Dept. of Computer Engineering, University of Murcia, 30100 Murcia, Spain
[b] Universidad Católica San Antonio de Murcia (UCAM), 30107 Murcia, Spain
[c] Academic Unit of Engineering, Autonomous University of Guerrero, Chilpancingo, Mexico

ARTICLE INFO

Article history:
Received 15 November 2014
Received in revised form 2 July 2015
Accepted 3 July 2015
Available online 18 July 2015

Keywords:
NVIDIA GPUs
Intel Xeon Phi
Room acoustics simulation
Acoustic diffusion equation model
CUDA
OpenMP

ABSTRACT

The diffusion equation model is a popular tool in room acoustics modeling. The 3-D Finite Difference (3D-FD) implementation predicts the energy decay function and the sound pressure level in closed environments. This simulation is computationally expensive, as it depends on the resolution used to model the room. With such high computational requirements, a high-level programming language (e.g., Matlab) cannot deal with real life scenario simulations. Thus, it becomes mandatory to use our computational resources more efficiently. Manycore architectures, such as NVIDIA GPUs or Intel Xeon Phi offer new opportunities to enhance scientific computations, increasing the performance per watt, but shifting to a different programming model. This paper shows the roadmap to use massively parallel architectures in a 3D-FD simulation. We evaluate the latest generation of NVIDIA and Intel architectures. Our experimental results reveal that NVIDIA architectures outperform by a wide margin the Intel Xeon Phi co-processor while dissipating approximately 50 W less (25%) for large-scale input problems.

1. Introduction

The technology of both hardware and software has evolved considerably in the past ten years. Modern architectures use several cores with different functionality, performance and energy efficiency. Those cores can be divided into latency-oriented (for control-dominated tasks) and throughput-oriented (for data-driven tasks). Throughput-oriented cores are usually pseudo-independent external devices, connected through a PCIe bus to a host machine. In fact, NVIDIA Graphics Processing Units (GPUs) and Intel Many Integrated Core (MIC) architectures [1,2] are used in the most powerful supercomputers available nowadays [3].

NVIDIA GPUs are programmed through APIs such as OpenCL or CUDA [4], accessible from languages like C/C++, Fortran just to name a few. Intel accelerators are based on a simpler programming model, sharing many similarities with regular

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. J. Carretero.

* Corresponding author.
E-mail addresses: mario.hernandez4@um.es (M. Hernández), bimbernon@alu.ucam.edu (B. Imbernón), jmnvarro@ucam.edu (J.M. Navarro), jmgarcia@ditec.um.es (J.M. García), jcebrian@ditec.um.es (J.M. Cebrián), jmcecilia@ucam.edu (J.M. Cecilia).

http://dx.doi.org/10.1016/j.compeleceng.2015.07.001
0045-7906/

CPUs, making it more accessible for developers.However, developing applications on such architectures is still not a straight-forward task. Indeed, programmers in the scientific community need to use the latest breakthroughs in both; high performance computing and in the specific field of interest (e.g., image processing, computational modeling or acoustic diffusion) to deal with challenges of the next century. This vertical approach enables remarkable advances in computer-driven scientific simulations (the so-called hardware–software co-design).

A particular interest to us is the diffusion processes of the acoustic energy. This model is used to predict the sound field in arbitrary shape rooms and non-homogeneous distribution of the sound absorption [5]. A 3D Finite Differences (3D-FD) method was presented by Navarro et al. [6] as an alternative technique to solve the systems of equations for the acoustic diffusion equation model but in the time domain. The proposed 3D-FD method provides some schemes to solve the diffusion equation in a transitional regime. Therefore, the sound pressure level decay curve can be predicted in all receiver positions of the discretized room with only a single execution of the algorithm. However, the simulation of some room scenarios, e.g. long rooms and coupled rooms, requires a very small spatial discretization or cell size to obtain accurate results. If the spatial resolution of the discretization increases, the number of cells shows a cubic growth, and thus the computational requirements exceed the resources of traditional computing systems.

Although GPUs has been used for main room acoustics modeling techniques like ray-based modeling and wave based modeling [7], the parallelization of the 3D-FD acoustic diffusion implementation is a new trend [8]. Moveover, acceleration of the diffusion equation is a challenging mathematical problem both in 2D; ultrasonic and radar [9], heat equation [10] and flow model [11], and in 3D; reaction–diffusion problems [12] and explicit heat equation [12], however it has not been applied to the acoustic diffusion Dufort–Frankel approximation [6].

This paper describes the acceleration of a 3D-FD kernel within the room acoustics simulation to show the benefits of hardware–software co-design. Our starting point is a Matlab implementation that is rewritten in ANSI C to leverage sequential architectures. Then, we drive this application through the latest accelerator architectures in the market; i.e. Nvidia GPUs and Intel Xeon Phi architectures. By porting the code to these platforms, our simulations improve both elapsed time and energy consumption. Finally, we share our experiences with readers to let them know insights about programmability issues on both novel platforms. Our major contributions include the following:

1. A low level implementation (i.e., C/OpenMP) of the Matlab 3D-FD kernel. This implementation outperforms Matlab in a factor of **5**×, when running on a single thread. This shows that the programming effort is worth once the algorithm is mature enough and will not change over time.
2. Extension of the C implementation of the 3D-FD method to fully utilize the cores and Streaming SIMD Extensions (SSE)/Advanced Vector Extensions (AVX) vector instructions, in both Intel CPUs and Xeon Phi co-processors.
3. A data-parallel scheme on GPUs is deployed using Compute Unified Device Architecture (CUDA) programming model. Our design proposes a singular tiling technique to exploit data-locality via shared memory.
4. Current hardware generation based on NVIDIA Kepler/Maxwell GPUs and Intel MIC architecture are compared to reveal solid advantages of GPUs concerning energy efficiency and performance enhancements. The CUDA implementation shows much better performance (up to 2×) than the Intel Xeon Phi implementation and better energy efficiency; up to 5× better Energy Delay Product (EDP).
5. Our results nominate the NVIDIA's GPUs as the best accelerator option, giving the best performance and the lowest power consumption results. Moreover, considering pricing and performance-oriented metrics like the EDP, NVIDIA high-end GPUs (980GTX) are better suited for computing the 3D-FD Kernel.

The rest of the paper is organized as follows. Section 2 briefly introduces basic concepts. Section 3 describes the implementation details of the targeted 3D-FD kernel in both platforms. Section 4 reports our evaluation methodology before showing the main experimental results in Section 5. Finally, Section 6 concludes the paper and shows possible directions for future work.

2. Background

2.1. Discrete Element Method (DEM) simulation using finite differences

2.1.1. Acoustic diffusion equation model

Diffusion process has been successfully applied to predict room-acoustic parameters, such as the reverberation time and the sound pressure level, in different room scenarios [13]. Recently, a theoretical model was proposed based on the energy radiative transfer equation to generalize the modeling techniques, which make use of the sound particles propagation approach [14]. This energy transfer theory allows to formulate the foundations of the geometrical acoustics techniques [15] encompassing the diffusion equation model between them.

The acoustic diffusion equation model is stated to be accurate mostly in low absorption rooms and to predict the late part of the room impulse response [13]. From this estimated room impulse response, it is possible to calculate the sound pressure level and the reverberation time at every position of the enclosure.

The diffusion equation model for the sound energy density $w(\mathbf{r},t)$ at position $\mathbf{r}$ defined on a domain V and time t, which includes a sound source term $P(t)$ located at position $\mathbf{r}_s$, consists of a partial differential equation with mixed boundary conditions [13,16],

$$\frac{\partial w(\mathbf{r},t)}{\partial t} - D\nabla^2 w(\mathbf{r},t) + cmw(\mathbf{r},t) = P(t)\delta(\mathbf{r}-\mathbf{r}_s) \text{ in } V, \tag{1}$$

$$-D\frac{\partial w(\mathbf{r},t)}{\partial \mathbf{n}} = A_X(\mathbf{r},\alpha)cw(\mathbf{r},t) \text{ on } \partial V. \tag{2}$$

Eq. (1) is an homogeneous parabolic partial differential equation, where ∇^2 is the Laplace operator and $D = \lambda c/3$ is the diffusion coefficient with c being the speed of sound. This diffusion coefficient takes into account the room geometry through its *mean free path* λ, which indicates the average distance that a sound particle travels between two consecutive collisions [17]. In the classical acoustic theory, the mean free path in a room is given by $\lambda = 4V/S_t$, with volume V and total interior area S_t. The term $cmw(\mathbf{r},t)$ accounts for the atmospheric attenuation within the room, where m is the absorption coefficient of air [18].

Eq. (2) is a mixed boundary condition that models the local effects on the sound field induced by different degrees of absorption on the surfaces. The term $\mathbf{n}$ represents the unity vector normal to the boundary surface. This equation allows one to express the distribution of the surface absorption properties through the absorption factor $A_X = A_X(\mathbf{r},\alpha)$, where α is the absorption coefficient. Different definitions of A_X have been presented in the technical literature, each depending on the assumed physical theory. In this paper, the modified absorption factor [14,16] is adopted to perform the simulations in Eq. (3):

$$A_X = A_M(\mathbf{r},\alpha) = \frac{\alpha(\mathbf{r})}{2(2-\alpha(\mathbf{r}))}. \tag{3}$$

2.1.2. Finite difference implementation of the acoustic diffusion model

The finite difference method is a numerical technique used to solve a differential equation over a given region subject to the specified boundary conditions, based on a finite difference approach of the involved derivatives of a partial differential equation. When the finite difference approach is used, the problem domain is discretized so that the values of the unknown dependent variable are considered only at a finite number of nodal points or cells instead of at every point over the region. A discretized function is defined as follows in Eq. (4).

$$w(\mathbf{r},t) = w(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = w^n_{i,j,k}, \tag{4}$$

The temporal index n and the spatial indexes i, j and k have been introduced together with the temporal discretization Δt and spatial discretizations in the cartesian axis Δx, Δy and Δz. Latters are defined as the inverse of the temporal and spatial resolutions respectively. For example, low resolution implies large cell sizes and high resolution implies small cell sizes.

In this paper, the chosen 3D-FD solution for the acoustic diffusion equation will be the Dufort–Frankel scheme [19], published in [6], because its suitability and attractive features (e.g., it is unconditionally stable). For simplification, let us define $\beta_{0_v} = (2D\Delta t)/(\Delta v)^2$, where $v = [x,y,z]$, and $\beta_0 = \sum_{v=[x,y,z]}\beta_{0_v}$, so the finite difference scheme for Eq. (1) is as follows in Eq. (5),

$$\begin{aligned} w^{n+1}_{i,j,k}(1+\beta_0) = {} & w^{n-1}_{i,j,k}(1-\beta_0) - 2\Delta t cmw^n_{i,j,k} \\ & + \beta_{0_x}(w^n_{i+1,j,k} + w^n_{i-1,j,k}) \\ & + \beta_{0_y}(w^n_{i,j+1,k} + w^n_{i,j-1,k}) \\ & + \beta_{0_z}(w^n_{i,j,k+1} + w^n_{i,j,k-1}). \end{aligned} \tag{5}$$

The source term, included as a soft source, is added at the suitable position as $w^{n+1}_{i_s,j_s,k_s} = w^{n+1}_{i_s,j_s,k_s} + 2\Delta t P^n_{i_s,j_s,k_s}$.

Additional relations are needed to make the number of equations equal to the number of unknown variables. These variables are obtained from boundary conditions, explained as follows.

A second-order accurate difference of the boundary conditions [6] of Eq. (2) is used to ensure the accuracy of the approximation. For simplicity, the finite difference approximation of boundary surface oriented on the x-axis at both positions $x = 0$ and $x = l_x$ is presented.

$$w^{n+1}_{0,j,k} = \frac{4w^{n+1}_{1,j,k} - w^{n+1}_{2,j,k}}{3 + \frac{2A_{X_{0,j,k}}\Delta x}{D}}, \tag{6}$$

$$w^{n+1}_{L_x,j,k} = \frac{4w^{n+1}_{L_{x-1},j,k} + w^{n+1}_{L_{x-2},j,k}}{3 + \frac{2A_{X_{L_x,j,k}}\Delta x}{D}}. \tag{7}$$

The derivation for dimension y and dimension z is straightforward.

Thus, Eqs. (6) and (7), together with Eq. (5) are the complete finite difference approximations of the acoustic diffusion equation in 3D subjected to mixed boundary conditions. These approximations are implemented in Section 3 for different parallel architectures.

2.2. NVIDIA GPU and Intel MIC architectures

Heterogeneous computer architectures that combine general-purpose multicore CPUs with specialized accelerators have become a viable solution to build high performance supercomputers, as demonstrated by Titan at ORNL (NVIDIA GPGPUs), Tianhe-2 at NSCC (Intel Xeon Phi), and Stampede at TACC (Intel Xeon Phi) in the recent Top500 list [3].

NVIDIA introduced its CUDA architecture in 2006 [4]. CUDA allows users to program GPUs for general purpose computing. In a few years, the GPGPU (General Purpose for GPUs) field expanded and became one of the best ways to achieve high performance from commodity processors. CUDA is based on a hierarchy of abstraction layers; the *thread* is the basic execution unit; threads are grouped into *blocks*, each of which runs on a single multiprocessor, where they can share data on a small but extremely fast memory. A *grid* is composed of blocks, which are equally distributed and scheduled among all multiprocessors. The parallel sections of an application are executed as *kernels* in a Single Instruction Multiple Data (SIMD) fashion, that is, with all threads running the same code. A kernel is therefore executed by a grid of thread blocks, where threads run grouped in batches (warps), which are the scheduling units.

The Intel MIC architecture combines many Intel CPU cores onto a single chip [2]. The Xeon Phi co-processor is the first product based on this architecture. The main advantage of these accelerators is that they provide a general-purpose programming environment similar to that provided for x86 CPUs. This co-processor is capable of running applications written in industry-standard programming languages such as Fortran, C, and C++. Each co-processor consists of more than 50 cores clocked at 1 GHz or more. The number of processors actually depends on the generation and model of the specific co-processor. Each core contains a 512-bit wide vector unit (VPU) with vector register files (32 registers per thread context). It also features an in-order, dual-issue x86 pipeline, four-way hyper-threading, 32 KB of L1 data cache, and 512 KB of L2 cache that is kept fully coherent by a global-distributed tag directory.

The Intel Xeon Phi co-processor is delivered in form factor of a PCIe device. The second generation (Knights Landing) may, however, be used as stand-alone processor. The architecture runs a Linux operating system and can execute native applications. Nonetheless, binaries for this architecture are not compatible with other Intel CPUs. Native execution is not the only method to employ the co-processor. Offload models and hybrid message passing interface (MPI) jobs may be more suitable for complex applications. However, because our interest lies in the evaluation of performance and optimization methods, we restrict our discussion to native execution. Finally, to program applications on the Xeon Phi, users need to capture both functionality and parallelism. Being an x86 SMP-on-a-chip architecture, Xeon Phi offers full capability to use the same tools, programming languages, and programming models as a regular Intel Xeon processor. Specifically, tools like Pthreads, OpenMP, Intel Cilk Plus, and OpenCL are readily available. Given the large number of cores on the platform, a dedicated MPI version is also available.

3. 3D-FD implementations for massively parallel architectures

This section describes the implementation details of a 3D-FD acoustic diffusion equation model kernel. The algorithm proposed in this work are only applied to Eq. (5), i.e. to those cells within the simulated enclosure volume; which actually represents close to the 95% of total execution time. First we developed a sequential C version starting from the Matlab code. Next, we present two different parallel implementations based on the sequential code: (i) an OpenMP latency-oriented implementation optimized for the Xeon Phi architecture (with SIMD support), (ii) an optimized CUDA implementation for the NVIDIA GPU architecture.

3.1. Sequential baseline

Multigrid methods based on stencil patterns or computations are widely used in scientific applications, especially in physics, chemistry or signal processing. Much of the time consumed by a multigrid algorithm is due to the *smoother* used by it. In this work we focus on the 3D-FD implementation of the acoustic diffusion equation model kernel previously presented in Section 2.1.

Fig. 1 shows a multigrid method to implement a stencil computation based on this update equation, and Algorithm 1 shows the sequential baseline for this algorithm. It is implemented as a quad-nest loop traversing the complete computational domain by updating each grid point. Three 3D variable arrays are involved in the main computation loop (w_{n+1}, w_{n-1} and w_n). We consider the absorption coefficient of the medium (in this case the air inside the room (m)), the speed of sound (c) and the time discretization (dt) in addition to the constant equation inside the room (β_0). There is generally no need to store the entire space–time grid (namely w_{n+1}, w_{n-1} and w_n), and so the code uses three copies of the spatial grid, swapping their roles on alternate time steps. Although this loop nest is simple and fairly easy to understand, its performance may suffer from poor cache locality. Some basic transformations can be applied safely to improve the performance of the generated target code on any architecture without exploiting knowledge about stencil codes [20].

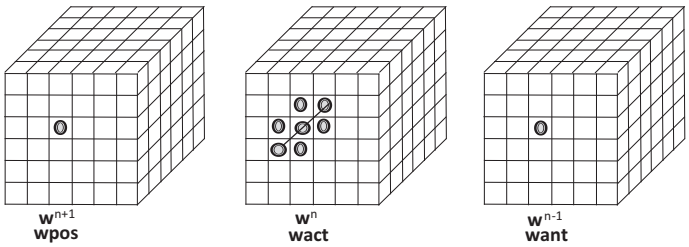

Fig. 1. Stencil pattern for 3D-FD of the acoustic diffusion equation model. Based on a multigrid method.

Algorithm 1. The sequential pseudocode for the calculation of the propagation of the acoustic energy density.

```
1: for i = 0; i ⩽ Iterations; i++ do
2:   for x = 0; x ⩽ XDIM; x++ do
3:     for y = 0; y ⩽ YDIM; y++ do
4:       for z = 0; z ⩽ ZDIM; z++ do
5:         w_{n+1}[x,y,z] = (w_{n-1}(x,y,z) * (1 − β_0)+
6:         β_{0_x} * (w_n(x+1,y,z) + w_n(x−1,y,z))+
7:         β_{0_y} * (w_n(x,y+1,z) + w_n(x,y−1,z))+
8:         β_{0_z} * (w_n(x,y,z+1) + w_n(x,y,z−1))−
9:         2 * dt * c * m * w_n(x,y,z)+
10:         2 * dt * P_n(x,y,z))/(1 + β_0);
11:      end for
12:    end for
13:  end for
14: end for
```

3.2. OpenMP implementation with vectorization in Intel Xeon Phi

3.2.1. OpenMP

This section describes the OpenMP implementation for the Intel architectures. OpenMP [21] is an API to develop parallel applications on shared-memory architectures mainly. The programming model is based on a set of compiler directives (*pragmas*) and a library of routines to query and tune some aspects of the execution.

The compiler provided by Intel to generate code for the Intel Xeon Phi architecture supports OpenMP 4.0 version. The execution model used by OpenMP is based on the fork-Join pattern. The sequential parts of the program are executed by a single thread while the parallel parts are executed by multiple threads. A very large number of parallel applications are based on for-loops, in which the same code is executed for multiple data elements. Such applications can be easily parallelized using the directive `#pragma omp parallel for`. This directive has some clauses that can be used to fine-tune the parallelization. Such optimizations include the scheduling model, variable privacy among threads or the reduction of the intermediate results for each iteration.

In our case, the parallelization is implemented by adding the: `#pragma omp parallel for private(y, z)`, where y and z are the indexes of two nested spatial loops. Those indexes need to be private for every thread to maintain the program semantics.

3.2.2. Vectorization

This section describes the vectorized (SIMD) implementation for Intel architectures. Almost all modern processors have SIMD units, which can be used to perform the same computation simultaneously over multiple data elements. Current x86 processors support AVX or, at least, SSE instructions. SSE provides 128-bit registers while AVX extends the functionality to 256-bit registers. The Intel Xeon Phi cores feature a vector processing unit (VPU) [22] that significantly increases its computing power. Each VPU supports a new 512-bit SIMD instruction set that can execute 16 float or 8 double elements per instruction.

When relying on auto-vectorization, the compiler looks for opportunities whenever the code is compiled using -O2 or higher. In our case, the parallelization process begins using the `O3 -vec-report2` compiler options, obtaining a report with some obstacles for vectorization due to non-contiguous memory accesses and loop data dependencies. Two code modifications were developed to solve this issue. Firstly, vectorization is forced by means of the `#pragma simd` feature within OpenMP 4.0. Secondly, the data was aligned for vectorization. Typically, this means both: (a) aligning the base-pointer where

the space is allocated for the arrays and, (b) making sure the starting indexes have good-alignment properties for each vectorized loop. The `_mm_malloc()` and `_mm_free()` instructions are used for this purpose. These routines take an alignment parameter (in bytes) in the second argument. In addition, for C/C++ arrays allocated dynamically, the `_assume_aligned(matrix, 64)` instruction is used before the vectorizable loop. Without this clause, the compiler may not be able to discover the specific alignment used when the matrix was allocated at runtime.

3.3. CUDA implementation

This section summarizes the data-parallelism approach developed for the CUDA architecture. We have studied several data-parallelism approaches but, for this paper, we only show our best empirically demonstrated CUDA design. Our main objective here is to provide a comparison among different platforms. Nevertheless, we refer the reader to, [23,24] for a optimization carving on similar stencil patterns in Nvidia architectures.

Fig. 2 shows our CUDA design. It is based on 2-dimensional block design dividing the cube into vertical blocks, one for each coordinate (x, y) of size z. A CUDA thread is assigned to each grid point $(x, y) \in [1, NZ]$. Those threads iterate over the z axis. Fig. 3 shows the data accessed by each CUDA 2-D block. We show the data lying on the x-axis. In this way, threads within the same block can share information through the on-chip shared memory and the register file, taking advantage of the data-locality that is available on this computational pattern.

4. Benchmarking environment

This section describes the hardware–software environment, the input data sets and the way power measurements are taken for later use in Section 5.

4.1. Hardware systems

4.1.1. Intel Xeon and Xeon Phi co-processor

The evaluation platform is equipped with two Ivy Bridge-EP Intel Xeon E5-2650 CPUs (2×8 cores in total), and a top-of-the-line Intel Xeon Phi 7120P co-processor (61 cores). The Intel(R) Xeon(R) CPU E5-2650 v2 at 2.6 GHz has a 20 MB L3 cache, 8 physical cores, 16 threads and 32 GB of DDR3-1600 main memory. It provides a theoretical bandwidth of 59.7 GB/s per processor. This machine is used as our evaluation CPU (*Xeon* in our figures).

The Xeon Phi 7120P has 61 cores working at 1.238 GHz, and each core can process 16 single-precision data elements at a time, with maximum 2 operations (multiply-add or mad) per cycle in each lane (i.e., a vector element). Therefore, the theoretical instruction throughput is 2416 GFlops. Moreover, the most important feature of the Xeon Phi co-processors is memory bandwidth. The evaluated Xeon Phi has 16 memory channels (32-bit wide). With up to 5 GT/s of transfer speed, the 7120P provides a theoretical bandwidth of 352 GB/s.

4.1.2. NVIDIA GPU

Kepler and Maxwell are the latest generations of the NVIDIA GPU architecture [25]. Compared to previous designs (i.e., Fermi), they extend the number of cores within a multiprocessor from 32 to 192 (128 SMM for Maxwell), and the scheduling units from 2 to up to 8 warps at a time. In addition the L2 cache doubles its size. Despite the resource additions (resulting in a far higher transistor count per unit area), modern GPUs are three (six) times more power-efficient than previous generations. This is mainly achieved keeping the frequency below 1 GHz and using a manufacturing process of 28 nm. We target a Kepler-based architecture (Tesla K40c) and a Maxwell GPU (980GTX). The K40c has 2880 cores running at boost clock of 0.88 GHz, giving a raw processing power up to 5068 GFLOPS. The memory speed is 3.0 GHz with a 384-bits memory bus width that provides a bandwidth of 288 GB/s. The memory size is 12 GB of GDDR5 with ECC capabilities. The 980GTX has 2048 cores running at 1.1 GHz, 4 GB of GDDR5 and a peak memory bandwidth of 224 GB/s.

4.2. Software environment

The Intel Xeon Phi was programmed using *C* with OpenMP. We used the Intel's ICC compiler (version 14.0.2). In order to produce an executable for the co-processor in the native execution mode, the code must be compiled with the flag `-mmic`. After compilation, the executable must be copied to the co-processor file system before execution. Sometimes, additional libraries must be copied to the co-processor or NFS-shared with it. In our evaluation, we transfer the files using the secure copy tools `scp`, log into the co-processor system using `ssh`, and run commands on the co-processor from the shell. The evaluated platform run CentOS 6.5 with MPSS 3.2.3. The CUDA programming model is used to program the NVIDIA GPUs (Tesla K40c and 980GTX). More precisely, we have used the CUDA SDK 6.5. The evaluated platforms run Ubuntu 14.04.1. The application is executed ten times in a row. We use a trimmed mean (0.3) to reduce the effects of outliers in the final measurements.

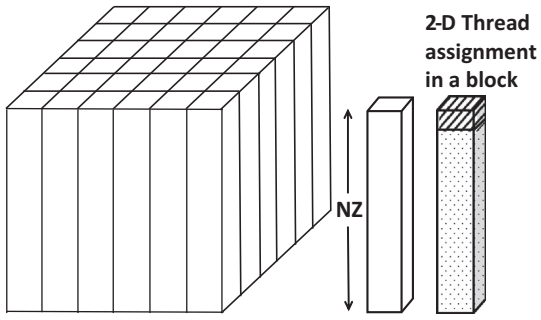

Fig. 2. 2D CUDA blocks design. Each CUDA thread need to loop over *Z*-axis.

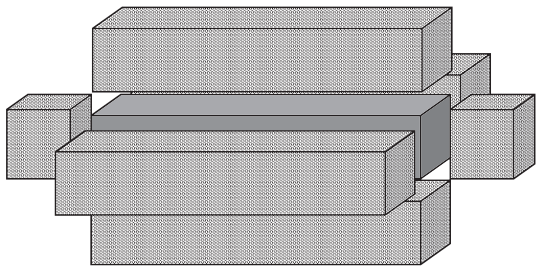

Fig. 3. Data accessed by a CUDA 2-D block.

4.3. Input datasets

This section describes the data of the different input parameters used in our experiments. The discretized enclosure was a cubical shape room with $8 \times 8 \times 8\text{m}^3$ dimensions. The spatial and temporal discretization were chosen to meet the conclusions enunciated in [6]. Accordingly, $(\Delta t)^2/(\Delta v)^2$ should be of the order of 10^{-8} to ensure that the 3D-FD implementation converges to a fixed value with a low error. Applying this convergence criterion, four different grid resolutions were created for the simulations; i.e. $128 \times 128 \times 128$ (32 MBytes), $256 \times 256 \times 256$ (256 MBytes), $384 \times 384 \times 384$ (864 MBytes) and $512 \times 512 \times 512$ (2048 MBytes) cells. These grid sizes eventually determine the memory requirements of the implementations. Memory usage comes mainly from four matrices required in the computations. Each one uses $(INPUT\_SIZE^3)$ *x size of*(*float*) *bytes* of storage. This estimation has been validated using Valgrind (-tool = memtest).

Previously to the performance experiments, several room scenarios have been targeted with different input parameters, e.g. absorption factor and diffusion coefficient, to check the validity and the accuracy of the output data. For the experiments discussed in Section 5, an absorption coefficient of 0.1, a speed of sound of 343 m/s, and an atmospheric attenuation of $0.0006\ \text{m}^{-1}$ were chosen. One sound source of 10^{-12} W and two receiver locations were defined to store the room impulse response of 1 s of duration.

4.4. Power measurement

For decades, power and energy estimations relied on complex models or expensive external devices. Nowadays, many hardware developers provide means to give energy/power feedback to application developers and computing centres. For Intel architectures, the second generation of Core i5 and i7 processors extended its MSRs[1] with energy information, enabling applications to examine their energy efficiency without using external devices. The studied Xeon processor includes three energy registers, one for the whole package (PKG), one for the cores (Power Plane 0 – PP0), and another one for the memory (DRAM). Latest versions of PAPI libraries (5.0 or higher) [26] include support for reading Intel's energy registers. Other platforms such as Intel Xeon Phi are also supported by PAPI relying on external components (micpower). This component allows developers to read instantaneous power readings for different parts of the Xeon Phi board, including core power, total board power, PCI-e power, etc. Real-time power measurement of individual GPU components is also supported by modern NVIDIA GPUs. This is done by using NVML (NVIDIA Management Library) [27], which reports the GPU power usage in real-time. PAPI provides a component to interact with NVML and retrieve power information from within the application. Unluckily, the evaluated NVIDIA 980GTX does not provide access to the power counters with our current setup (driver + NVML version). We have used an

[1] Model Specific Registers.

Table 1
Hardware specification summary.

System	Peak performance (GFlops)	Mem. bandwidth (GB/s)
Xeon 2650v2	2 × 166	2 × 59.7
Xeon Phi 7120P	2416	352
K40c	5068	288
980GTX	4612	224

external meter that measures the power dissipated by the whole system (at-the-wall power) for comparative purposes. We subtract the power used by the system running a single-threaded application without the GPU; to the power (steady power) dissipated by the system with the GPU to estimate the GPU power. Since the GPU version of the kernel only uses 80% of the CPU resources, we limit resource utilization of the CPU to measure our steady power.

We report PKG energy information for Intel Xeon, and total board power for both NVIDIA K40c, 980GTX and Intel Xeon Phi. This type of instrumentation introduces an overhead proportional to the sampling rate. Nevertheless, performance and power information were obtained in separate runs to minimize the aforementioned overhead. Temperature affects power readings as much as 5% every 8–10 °C [28]. Since we run the application ten times in a row we have ample time to warm up. Table 1 summarizes of the hardware characteristics of the analyzed systems.

5. Experimental results

This section presents our energy and performance evaluation of the 3D-FD kernel on the target platforms. The best configuration of thread count for each platform, 8 threads for Xeon (dual socket E5-2650v2@2.6 GHz, 4 threads per socket), 244 threads for Xeon Phi, and 256 threads per block for the GPUs is chosen for running the experiments. We show performance, energy and EDP (energy delay product) information for different cubic room resolutions previously explained in Section 4.3.

Fig. 4 shows runtime information (logarithmic scale) as well as speedup[2] (secondary *Y* axis) for all the evaluated platforms and optimal thread count. For small room sizes the performance of the Xeon CPU is very similar to that of the K40c, while the 980GTX is almost twice as fast. However, we feel that the implementation effort for GPU/accelerators does not pay off for this size. Nevertheless, as soon as the problem size increases, the accelerators start to pay off. Given the memory-bound nature of the kernel this is mostly because of the increased memory level parallelism (MLP). For all resolutions there is a clear advantage of the GPUs over both Xeon Phi and CPU implementations, however, the CUDA implementation effort is substantially higher when compared to the Xeon Phi. GPUs perform the computations approximately twice as fast as the Xeon Phi for all problem sizes, providing a peak speedup of 20× for sizes of 384 and 512 over the single threaded CPU implementation. Given the substantial performance increase of the Xeon Phi for problem sizes over 384 we think a different data organization can provide further improvements for this platform.

On the other hand, Figs. 5 and 6 show both energy and EDP as well as the average power dissipation (secondary *Y* axis) given the optimal thread count for each platform. It is important to note that the GPUs require a CPU to work, but CPU power was not accounted for the GPU computations. It is also worth mentioning that the second generation of Xeon Phi accelerators can work without a main CPU. Results show superior overall energy efficiency of the GPUs despite the higher technology factor (28 nm vs 22 nm), while providing half the run-times. Average power dissipation of the GPUs does not change much with problem sizes, and is close to 140 W. The Xeon CPU average power is reduced as we increase problem size, since it spends more time waiting for data than doing useful computations (around 115 W). For the Xeon Phi there is a better matching between the code and the input size/data organization for 384 and 512 sizes. This is translated both in better performance and higher CPU/Power dissipation (around 190 W).

The EDP metric gives additional importance to performance over power dissipation. For this metric (lower is better), the GPUs outperform the rest of the platforms, being 5× better than the Xeon Phi and 31× better than the Xeon CPU for 512. This advantage is slightly lower for 384, obtaining an impressive 4× better EDP than Xeon Phi and 20× better than the Xeon CPU. Nevertheless, the 980GTX GPUs is around 5 times cheaper than the K40c.

Table 2 provides a brief summary of the performance, energy and EDP results obtained in our evaluation for different room sizes.

5.1. Summary of results

Finally, we should promote efforts to using massively parallel architecture between several levels and components of existing software y hardware nowadays. According to our results, Table 3 shows comparative data through approaches to programmability.

Programmability looks to be a strong indicator of success in the adoption of the Xeon Phi architecture. Xeon Phi has clearly benefited from a more compatible programming paradigm even if it has slightly inferior performance statistics regarding to CUDA.

[2] Over the single threaded CPU implementation.

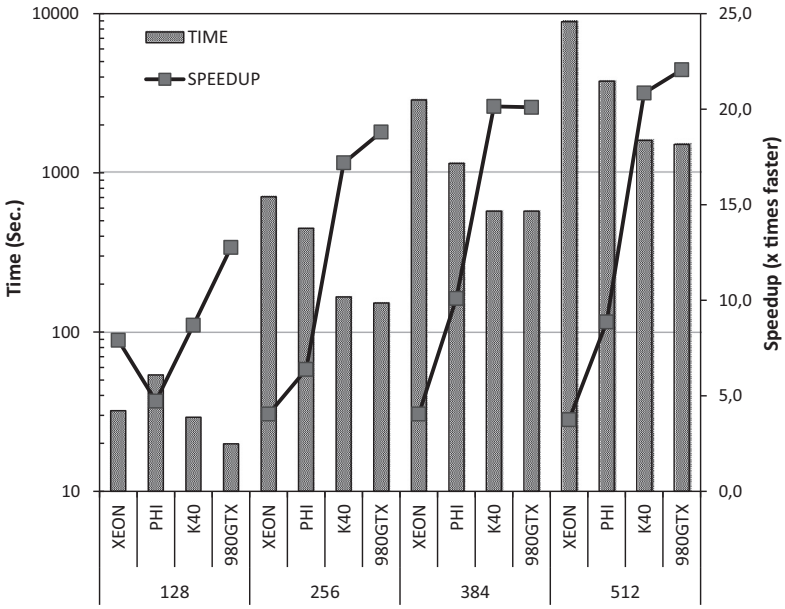

Fig. 4. Runtime (prim. *Y*) and Speedup (sec. *Y*) for different room resolutions i.e. $128 \times 128 \times 128$ (32 MBytes), $256 \times 256 \times 256$ (256 MBytes), $384 \times 384 \times 384$ (864 MBytes) and $512 \times 512 \times 512$ (2048 MBytes) cells.

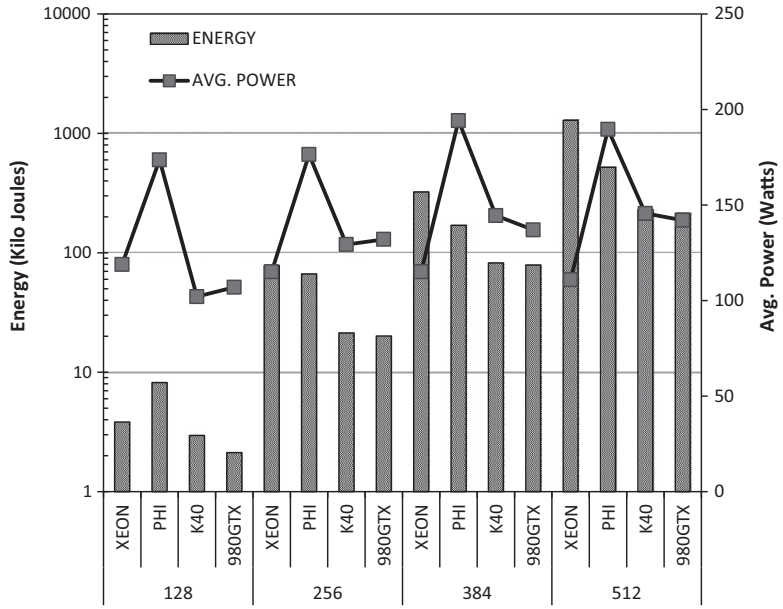

Fig. 5. Energy (prim. *Y*) and Average Power (sec. *Y*) for different room resolutions i.e. $128 \times 128 \times 128$ (32 MBytes), $256 \times 256 \times 256$ (256 MBytes), $384 \times 384 \times 384$ (864 MBytes) and $512 \times 512 \times 512$ (2048 MBytes) cells.

NVIDIA has the CUDA programming paradigm and accompanying tools. NVIDIAs CUDA programming environment is much more developed regarding to Xeon Phi.

6. Conclusions and future work

Applications with a real impact on the society can take advantage of the great advances in the field of high performance computing to overcome emerging challenges. When physical limitations of silicon-based architectures are threatening the

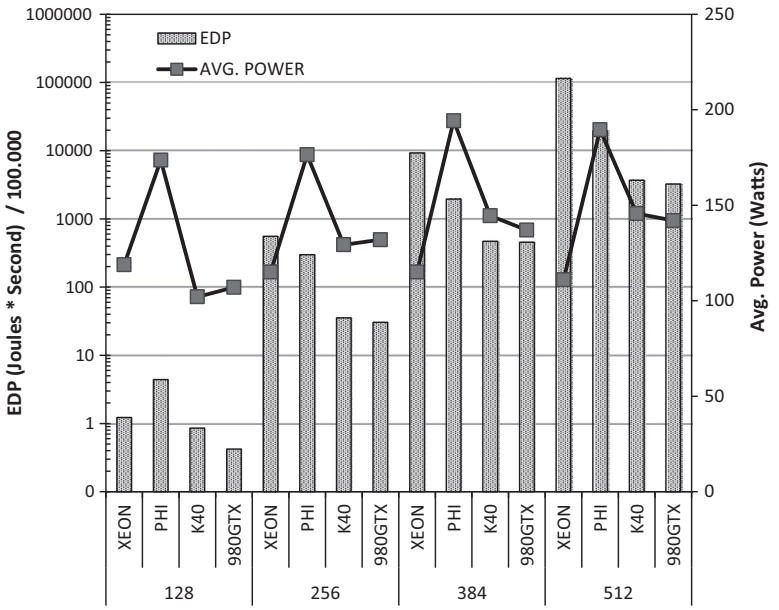

Fig. 6. EDP (prim. *Y*) and Average Power (sec. *Y*) for different room resolutions i.e. 128 × 128 × 128 (32 MBytes), 256 × 256 × 256 (256 MBytes), 384 × 384 × 384 (864 MBytes) and 512 × 512 × 512 (2048 MBytes) cells.

Table 2
Summary of the main empirical results.

Platforms	128			256			384			512		
	Time	Energy	EDP	Time	Energy	EDP	Time	Energy	EDP	Time	Energy	EDP
Xeon	32	4	1	708	78	574	2865	324	9277	8885	1286	114228
Phi	54	8	4	448	67	298	1146	170	1947	3759	520	19536
K40	29	3	1	166	21	35	574	82	471	1598	230	3683
980GTX	20	2	0	152	20	30	575	79	454	1510	214	3236

Time (s), Energy (kJ) and EDP (J* s)/100,000) for different room resolutions.

Table 3
Results comparative.

Approaches to programmability	Lenguaje C	Intel		NVIDIA	
		Xeon/OpenMP	Xeon Phi	K40	980GTX
Difficult of programing	Low	Low	Medium	High	High
Economic cost	–	–	3000€	3000€	600€
Time speed up[a]	5×	25×	50×	90×	110×
Energy savings[b]	1	2.63	2.71	3.96	7.81

[a] Speed up over the matlab sequential.
[b] Normalized against sequential C.

evolution of processors, massively parallel processors (GPUs and accelerators) come to the rescue. Programmability of these architectures can be a challenge for inexperienced developers, but the benefits of a hardware–software co-design will definitely pay off in the long term.

We have analyzed this synergy between application and hardware, benchmarking flagship processors from major vendors like Intel and NVIDIA. We analyze programmability, power, performance and energy using a 3-D Finite Difference (3D-FD) implementation of the diffusion equation model as our case study. We develop three different implementations for this room acoustic simulation kernel:

1. A single-threaded version of the Matlab code migrated to C.
2. A C/OpenMP version with vectorization to leverage the computational power of *CMPs* and *Intel Xeon Phi* accelerators.
3. A data-parallel scheme on *GPUs* using CUDA to target NVIDIA platforms, where we propose a tiling technique to exploit data locality using shared memory.

After our empirical evaluation, we nominate the NVIDIA's Maxwell GPUs as the most power efficient accelerator. Moreover, considering performance-oriented metrics like energy delay product (EDP), NVIDIA high-end GPUs are better suited for computing the 3D-FD Kernel, with up to 5× improvements over the Xeon Phi and 31× over the Xeon CPU for big room resolutions (512). For small problems, CPUs seem to be sufficient in terms of performance and energy efficiency. However, when problem size increases the independent memory interfaces/hierarchies (GDDR5) from both accelerators (Xeon Phi) and GPUs (K40c and 980GTX) offer clear benefits for this memory bounded kernel.

Given the memory boundless of the kernel, an interesting analysis would be to reduce the frequency of the computation elements (cores), looking for greater opportunities to reduce power in all platforms. Moreover, as the trend of parallel computing is to use HPC clusters, we aim to scale our evaluations to HPC clusters in the future.

Acknowledgements

This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 15290/PI/2010 and 18946/JLI/13, and by the Spanish MEC and European Commission FEDER under grant with references TEC2012-37945-C02-02 and TIN2012-31345, and the Nils Coordinated Mobility under grant 012-ABEL-CM-2014A, in part financed by the European Regional Development Fund (ERDF). We also thank NVIDIA for hardware donation under Professor Partnership 2008–2010, CUDA Teaching Center 2014–2015. Mario Hernández was supported by a research grant from the PROMEP under the Teacher Improvement Program (UAGro-197) Mexico.

References

[1] Garland M, Le Grand S, Nickolls J, Anderson J, Hardwick J, Morton S, et al. Parallel computing experiences with CUDA. IEEE Micro 2008;28:13–27.
[2] Rahman R. Xeon phi system software. In: Intel® Xeon Phi coprocessor architecture and tools. Springer; 2013. p. 97–112.
[3] Top 500 supercomputer site. <http://www.top500.org/> [last access 15.11.14].
[4] NVIDIA. NVIDIA CUDA C Programming Guide 6.5; 2014.
[5] Picaut J, Simon L, Polack J-D. A mathematical model of diffuse sound field based on a diffusion equation. Acta Acust United Acust 1997;83:614–21.
[6] Navarro JM, Escolano J, López JJ. Implementation and evaluation of a diffusion equation model based on finite difference schemes for sound field prediction in rooms. Appl Acoust 2012;73:659–65.
[7] Savioja L, Manocha D, Lin M. Use of gpus in room acoustic modeling and auralization. In: Proceedings of the international symposium on room acoustics.
[8] Lopez JJ, Navarro JM, Carnicero D, Escolano J. Some comments about graphic processing unit (gpu) architectures applied to finite-difference time-domain (fdtd) room acoustics simulation: present and future trends. In: Proceedings of meetings on acoustics, vol. 19. Acoustical Society of America; 2013. p. 070098.
[9] Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW, Skadron K. A performance study of general-purpose applications on graphics processors using cuda. J Parallel Distrib Comput 2008;68:1370–80.
[10] Heimlich A, Mol A, Pereira C. Gpu-based monte carlo simulation in neutron transport and finite differences heat equation evaluation. Prog Nucl Energy 2011;53:229–39.
[11] Kalyanapu AJ, Shankar S, Pardyjak ER, Judi DR, Burian SJ. Assessment of gpu computational enhancement to a 2d flood model. Environ Model Software 2011;26:1009–16.
[12] Molnár F, Izsák F, Mészáros R, Lagzi I. Simulation of reaction–diffusion processes in three dimensions using cuda. Chemometr Intell Lab Syst 2011;108:76–85.
[13] Valeau V, Picaut J, Hodgson M. On the use of a diffusion equation for room-acoustic prediction. J Acoust Soc Am 2006;119:1504–13.
[14] Navarro JM, Jacobsen F, Escolano J, López JJ. A theoretical approach to room acoustic simulations based on a radiative transfer model. Acta Acust United Acust 2010;96:1078–89.
[15] Savioja L. Modeling techniques for virtual acoustics. PhD thesis. Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory, Espoo, Finland; 1999.
[16] Jing Y, Xiang N. On boundary conditions for the diffusion equation in room acoustic predictions: theory, simulations, and experiments. J Acoust Soc Am 2008;123:145–53.
[17] Morse PM, Feshbach H. Methods of theoretical physics. New York: McGraw-Hill; 1953.
[18] Billon A, Picaut J, Foy C, Valeau V, Sakout A. Introducing atmospheric attenuation within a diffusion model for room-acoustic predictions. J Acoust Soc Am 2008;123(6):4040–3.
[19] Dufort EC, Frankel SP. Stability conditions in the numerical treatment of parabolic differential equations. Math Tables Others Aids Comput 1953;7:135–52.
[20] Kronawitter S, Lengauer C. Optimization of two Jacobi Smoother Kernels by domain-specific program transformation. In: Proceedings of the 1st international workshop on high-performance stencil computations. p. 75–80.
[21] Chandra R. Parallel programming in OpenMP. Morgan Kaufmann; 2001.
[22] Rahman SMF, Yi Q, Qasem A. Understanding stencil code performance on multicore architectures. In: Proceedings of the 8th ACM international conference on computing Frontiers, CF '11. New York (NY, USA): ACM; 2011. p. 30:1–30:10.
[23] Micikevicius P. 3D finite difference computation on GPUs using CUDA. In: Proceedings of 2nd workshop on general purpose processing on graphics processing units. ACM; 2009. p. 79–84.
[24] Cecilia JM, Abellán JL, Fernández J, Acacio ME, García JM, Ujaldón M. Stencil computations on heterogeneous platforms for the Jacobi method: GPUs versus cell BE. J Supercomput 2012;62:787–803.
[25] AMD. NVIDIA Corporation. The Kepler Architecture. <http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf> [last access 15.11.14].
[26] Mucci PJ, Browne S, Deane C, Ho G. PAPI: a portable interface to hardware performance counters. In: Proceedings of HPCMP users group conference; 1999. p. 7–10.

[27] Nvidia Corporation. NVML API reference. <http://developer.download.Nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf> [last access 15.11.14].
[28] Cebrian JM, Natvig L. Temperature effects on on-chip energy measurements. In: Proceedings IGCC 2013, Arlington, USA; 2013. p. 1–6.

Mario Hernández is a Ph.D. student at the University of Murcia (Spain) and Professor at the Academic Unit of Engineering, Autonomous University of Guerrero (Mexico). He received his L.I. degree in Computer Science from Technological Institute of Chilpancingo (Mexico, 1989), and M.C. degree Master in Computer Science from Autonomous University of Guerrero (Mexico, 2005).

Baldomero Imbernón is an M.Sc. Student in New Technologies in Computer Science in University of Murcia (Spain) specialism High Performance Architectures and Supercomputing. He received his B.S. degree in Computer Science from Catolic University of Murcia (Spain, 2013). In the last two years, he has authored several journal papers in the areas of high performance computing and bioinformatics.

Juan M. Navarro received the B.S. degree in Telecommunications and the M.S. degree in Telecommunication, from the Universidad Politécnica de Cartagena, Murcia, in 2006 and 2008, and the Ph.D. degree in Telecommunications from the Universitat Politécnica de Valéncia, Valencia, Spain, in 2012. He is Assistant Professor at the Polytechnic Science Department, Universidad Católica San Antonio de Murcia.

José M. García is full-professor of Computer Architecture and also the Head of the Research Group on Parallel Computer Architecture at the University of Murcia (Spain). He served as Director of the Computer Engineering Department from 1998 till 2004, and recently, he has served as the Dean of the School of Computer Science for seven years (2006–2013).

Juan M. Cebrian Born in Albacete, Spain, in 1982. Got his B.Sc. in Computer Engineering on July 2006 (University of Murcia), M.Sc. in July 2007 (University of Murcia) and finished his Ph.D. on September 2011 (University of Murcia) with the qualification of Summa Cum Laude founded by a four year grant from the Spanish Ministry of Education.

José M. Cecilia is Assistant Professor at the Computer Science Department, Catholic University of Murcia (Spain). He received his B.S. degree in Computer Science from Universidad de Murcia (Spain, 2005), M.S. degree in Computational Software Techniques in Engineering from Cranfield University (United Kingdom, 2007), and Ph.D. degree in Computer Science from the Universidad de Murcia (Spain, 2011).