Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 316012, 14 pages
http://dx.doi.org/10.1155/2015/316012

Research Article

A Performance Study of a Dual Xeon-Phi Cluster for the Forward Modelling of Gravitational Fields

Maricela Arroyo,[1] Carlos Couder-Castañeda,[1,2] Alfredo Trujillo-Alcantara,[3,4] Israel-Enrique Herrera-Diaz,[5] and Nain Vera-Chavez[4]

[1]*ABACUS-CINVESTAV-IPN, Apartado Postal 14-740, 07000 México City, DF, Mexico*
[2]*Centro de Desarrollo Aeroespacial del Instituto Politécnico Nacional, Belisario Domínguez 22, 06010 México City, DF, Mexico*
[3]*Escuela Superior de Física y Matemáticas, Av. Instituto Politécnico Nacional Edificio 9, Unidad Profesional Adolfo López Mateos, 07738 México City, DF, Mexico*
[4]*Instituto Mexicano del Petróleo, Eje Central Lázaro Cardenas No. 152, 07730 México City, DF, Mexico*
[5]*Department of Industrial Engineering, Campus Celaya-Salvatierra, University of Guanajuato, Mutualismo 303 Colonia Suiza, 38060 Celaya, Gto, Mexico*

Correspondence should be addressed to Carlos Couder-Castañeda; ccouder@ipn.mx

Received 31 December 2014; Revised 27 May 2015; Accepted 8 June 2015

Academic Editor: Enrique S. Quintana-Ortí

With at least 60 processing cores, the Xeon-Phi coprocessor is a truly multicore architecture, which consists of an interconnection speed among cores of 240 GB/s, two levels of cache memory, a theoretical performance of 1.01 Tflops, and programming flexibility, all making the Xeon-Phi an excellent coprocessor for parallelizing applications that seek to reduce computational times. The objective of this work is to migrate a geophysical application designed to directly calculate the gravimetric tensor components and their derivatives and in this way research the performance of one and two Xeon-Phi coprocessors integrated on the same node and distributed in various nodes. This application allows the analysis of the design factors that drive good performance and compare the results against a conventional multicore CPU. This research shows an efficient strategy based on nested parallelism using OpenMP, a design that in its outer structure acts as a controller of interconnected Xeon-Phi coprocessors while its interior is used for parallelyzing the loops. MPI is subsequently used to reduce the information among the nodes of the cluster.

1. Introduction

The Xeon-Phi coprocessor is one of the new architectures designed specifically for high performance computing (HPC). Currently, the Xeon-Phi is part of some supercomputers listed on the TOP 500 and will continue to be a fundamental component of these machines for the foreseeable future. The Xeon-Phi is a X86 multicore architecture with low power consumption and with a theoretical performance of one Teraflop, for which it utilizes 60 real processing cores, a 30 MB cache, and high band-width interconnection [1].

One of the current research activities is to analyse the features and benefits of each new technology that emerges across the field of supercomputers, which is based, today, on GPUs and coprocessors. Each new technology carries with it different programming paradigms. Nevertheless, the effort to migrate scientific applications to CUDA (for GPUs) or OpenCL (i.e., programming that requires programming low level kernels) is often much higher as compared to directive-based programming like OpenMP (for CPU or MIC) [2]. Prior experiments which test the functionality of the Xeon-Phi coprocessor show that migrating scientific software is relatively easy, thereby making the MICs a promising tool for HPC applications [3].

One important point to be taken into consideration in order to understand the performance expected from the Xeon-Phi is that the multithreading, implemented in each core, is the key to reduce the inherent latency to a microarchitecture multicore [4]. This multithreading should not be confused with hyperthreading which consists in rapidly commuting between two threads that use the same core and can be disabled via BIOS. Since the multithreading is inherent

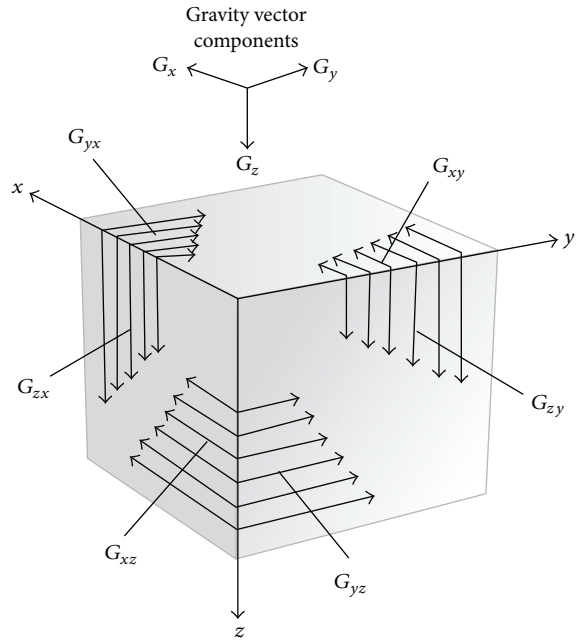

Figure 1: Vector and tensor relationships for the gravitational field in the three coordinate directions.

in the Xeon-Phi coprocessor (it cannot be disabled), the behaviour with different numbers of threads per processing core must be tested (from 1 to 4 threads per core).

Additionally, the numerical-based geophysical application that is migrated to the Xeon-Phi serves to extract more detailed information from the masses hidden in the subsoil by measuring the gravitational gradients. The gravity G has three elements that represent the components of gravity in the three orthogonal directions x, y, z (G_x, G_y, G_z). The gradient of G represents the gravitational field in tensor form and has nine components: $(G_{xx}, G_{yx}, G_{zx}, G_{xy}, G_{yy}, G_{zy}, G_{xz}, G_{yz}$, and $G_{zz})$ as shown in Figure 1.

The cartography of the gravitational gradients improves conventional gravitational studies by offering a better resolution of the subsoil. For example, an airplane equipped with gradiometers that is flying over a mountain (excess of mass) or a subterranean saline body (deficit of mass) will show only subtle variations in the global gravitational attraction. In contrast, the measurements of the corresponding gravitational gradients will reveal the geological changes with greater detail (see Figure 2). The erratic movement of the plane creates considerable noise in one gravitational profile, while the measurement of the difference between two sensors to obtain the gradient allows elimination of the error source [5].

The forward modelling of the gravity gradiometry consists in the direct conformation of the gravimetric data, where the initial model of the source body is constructed from the geological and geophysical intuition. We calculate the anomalies of the model and compare them to the observed anomaly, after which, the parameters are adapted in order to improve the adjustment between both anomalies. These three steps for adjusting the properties of the body, namely, calculation of the anomalies, comparison of the anomalies, and adjustments between both anomalies, are repeated until the observed and calculated anomalies are sufficiently similar with some criterion of error [6].

The numerical application basically consists in adapting a set of rectangular prisms that approximate the volume of the mass in the subsoil. If they are chosen sufficiently small, each prism can be considered to be of constant density. Then, by the principle of superposition, the gravitational anomaly of the body in any point can be approximated by adding the effects of all of the prisms on that point. Although this method seems simple, reducing the size of the prisms to adjust the source body causes a considerable increase in computation time. Besides, although there are other types of approximations such as points of mass or tesseroids, often for simplicity sake, researchers prefer to generate the prisms to model a body. This requires parallel computing that can carry out the forward modelling as fast as possible [7].

Thus, the computational challenge consists, in essence, in calculating the gravimetric response produced by a rectangular prismatic body with constant density or constant magnetization with respect to a group of observation points (see Figure 3). The set of prisms is known as an assembly of prisms and it is not necessarily regular. An assembly of prisms can be configured in any orientation with the only prerequisite that they cannot overlap. Since the gravitational field (or magnetic field) meets with the superposition property with respect to the observation, if f is the calculated response in a point (x, y), then the observed response in the point $f(x, y)$ is given as [8]

$$f(x, y) = \sum_{k=1}^{M} G(\rho_k, x, y), \tag{1}$$

where M is the total number of prisms and ρ is the density of the prism.

This problem was previously studied and it was determined that the best parallelization strategy is to carry out the solution via prisms since, in general, the number of prisms is much bigger than the number of observation points [9, 10]. The partitioning by observation points is efficient as long as there are not too many threads operating on the observation mesh at once. A false sharing could occur when the number of threads increases because the threads operate on the same memory locations. Another drawback of the observation points resides in the creating and closing of the parallel regions; in other words, for each prism, a function to calculate the anomalies is executed in parallel, but upon completion, the region is closed and it must be reopened for the next prism. This produces an unnecessary overload that affects performance (see Figure 4).

Therefore, if scalability is desired, the best parallelization is obtained via subdivision by prisms; in other words, the threads divide the work by the number of prisms. To avoid the problems of coherence in the cache, a different memory location must be created for each execution thread since it is no longer feasible to create only one memory space for only one observation mesh to be shared by all threads.

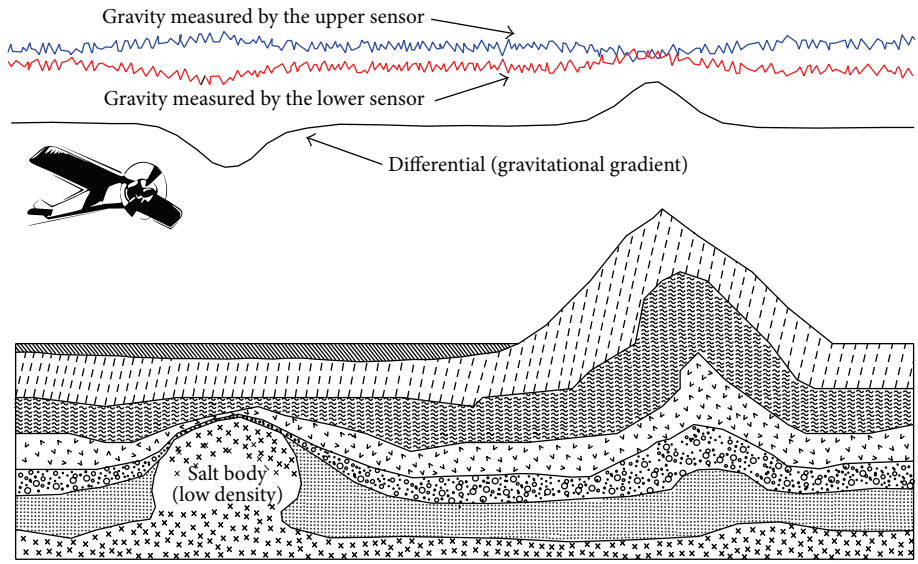

Figure 2: Simplified acquisition for recording the response of a geological signal by airborne gravity gradiometry.

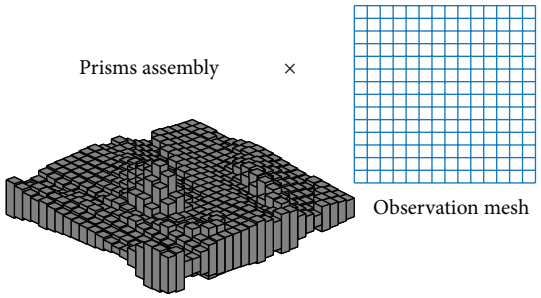

Figure 3: Configuration of the assembly of prisms against the observation mesh. For each prism in the assembly, an anomaly is calculated against each observation point.

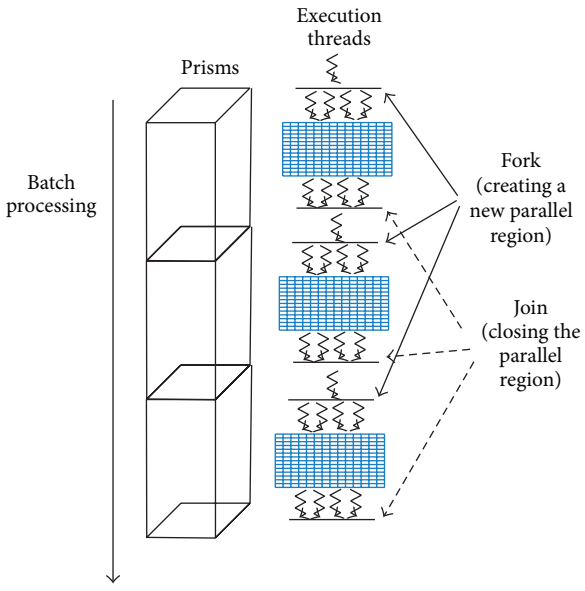

Figure 4: Partitioning by observation points. Each prism is calculated in parallel over the entire observation mesh.

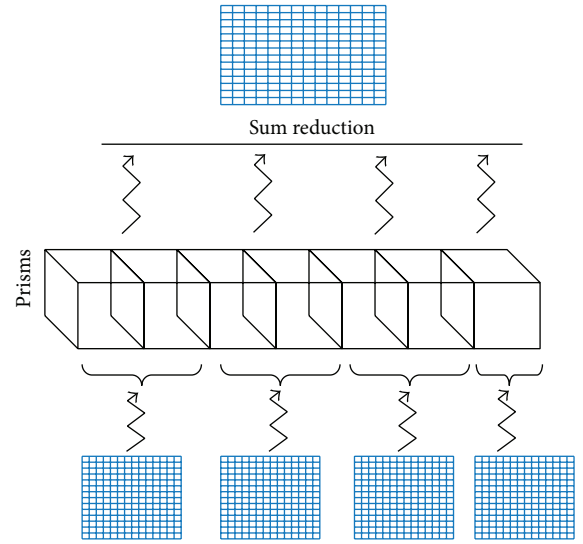

Figure 5: Partitioning by prisms. Each thread requires a separate memory location.

As shown in Figure 5, an observation mesh for each execution thread must be created, thereby avoiding the problems of consistency in the memory and false sharing in the accessing of the memory, since each thread writes to a different space in memory. If only one mesh was used, there would be problems in accessing shared variables, thereby causing inconsistencies.

One of the characteristics of OpenMP is that the division of computing is performed in an implicit way. Therefore, the partition of M prisms that conform the body in the subsoil is performed automatically by a balance algorithm included in OpenMP. In this case, the decision is left to the compiler, which in 99% of the cases is optimal [11].

The main contributions of this work are

(i) presenting a complete evaluation of the Phi coprocessors as integrated in the same node and in multiple nodes for the direct modelling of gravitational fields, thereby obtaining very acceptable performance results,

(ii) utilizing OpenMP as the controller and communicator for the integrated boards in the same node and also as a distributor of computing within a nested parallelism scheme. Additionally, we utilize MPI to distribute the computing among nodes,

(iii) constructing a geological model based on real data in order to test the computational tool and the resulting components delimit the dimensions of a saline body.

2. Description of the Forward Modelling Problem

The explorations via the gradiometry of gravity or full tensor gravity (FTG) are an exploration technique which is based on multiple variations of the gravitational field, variations which are measured via the local gradients in different directions. The main advantage of FTG is an improved resolution of the shallow sources. In this way, it is possible to construct more precise models that can detail objects such as gas, oil, or mineral hydrocarbon reservoirs and deposits.

The FTG consists of 9 components; however, only 5 are registered as independent measures from various perspectives. These measures are related by the fact that they can be recorded from the same geological source. Thus, the cojoint process can provide a data response with improved location of the anomaly source.

Each one of the tensors describes an object in a particular way. The G_{zz} tensor finds the source, G_{xx} and G_{yy} identify the limits of the source, G_{xz} identifies the boundaries, G_{yz} identifies the axes, both high and low, which in turn define the fault tendencies, and G_{xy} shows the anomalies that point toward the center of the source mass.

In geophysics, traditional methods designed to estimate the acceleration of gravity begin from seismic and geological information. This information is the basis for the construction of a subsoil domain with a density distribution. The integration of this information is often parametrized in prisms also called voxels and, depending on the complexity of the geometry, can be approximated by the form of the model with a number and size of determined parameters. Often, this final configuration is the basis for inverse models [12–14].

The gradiometric response, or direct modelling, is obtained by way of an algorithm that requires the calculation of a significantly large number of prisms. Thus, the computational cost is considerably high if the purpose is to approximate complex geometries with small prisms.

The gravitational field can be expressed in terms of the gravity potential $U_g(\mathbf{r})$ as

$$\mathbf{g}(\mathbf{r}) = \nabla U_g(\mathbf{r}). \tag{2}$$

The potential U_g by volume V has the following approximation [15]:

$$U_g(\mathbf{r}) = \gamma \iiint_V \frac{1}{|r - r'|} \rho(\mathbf{r}')\, dv, \tag{3}$$

where γ is the universal constant of gravity, ρ is the density in the volume, r is the position of the observation point, and r' is comprised of the elements of the volume given as dv.

The first spatial derivative of the potential U_g (3) can be expressed as

$$G(\mathbf{r})_\alpha = \gamma \iiint_V \frac{\partial}{\partial \alpha} \frac{1}{|r - r'|} \rho(\mathbf{r}')\, dv, \tag{4}$$

where α is any of the directions x, y, z; therefore, $G(\mathbf{r})$ in the different directions can be written as

$$\begin{aligned} G_x(\mathbf{r}) &= -\gamma \iiint_V (x - x') \frac{1}{|r - r'|^3} \rho(\mathbf{r}')\, dv, \\ G_y(\mathbf{r}) &= -\gamma \iiint_V (y - y') \frac{1}{|r - r'|^3} \rho(\mathbf{r}')\, dv, \\ G_z(\mathbf{r}) &= -\gamma \iiint_V (z - z') \frac{1}{|r - r'|^3} \rho(\mathbf{r}')\, dv. \end{aligned} \tag{5}$$

In order to calculate the tensors, we apply the second partial derivative to (4) to obtain

$$G_{\alpha\beta}(\mathbf{r}) = \frac{\partial^2}{\partial \alpha \partial \beta} U_g(\mathbf{r}), \quad \alpha, \beta = x, y, z, \tag{6}$$

therefore, the tensors can be approximated as

$$\begin{aligned} G_{xx}(\mathbf{r}) &= \gamma \iiint_V \frac{3(x - x')^2 - (r - r')^2}{|r - r'|^5} \rho(\mathbf{r}')\, dv, \\ G_{xy}(\mathbf{r}) &= \gamma \iiint_V \frac{3(x - x')(y - y')}{|r - r'|^5} \rho(\mathbf{r}')\, dv, \\ G_{xz}(\mathbf{r}) &= \gamma \iiint_V \frac{3(x - x')(z - z')}{|r - r'|^5} \rho(\mathbf{r}')\, dv, \\ G_{yy}(\mathbf{r}) &= \gamma \iiint_V \frac{3(y - y')^2 - (r - r')^2}{|r - r'|^5} \rho(\mathbf{r}')\, dv, \\ G_{yz}(\mathbf{r}) &= \gamma \iiint_V \frac{3(y - y')(z - z')}{|r - r'|^5} \rho(\mathbf{r}')\, dv, \end{aligned}$$

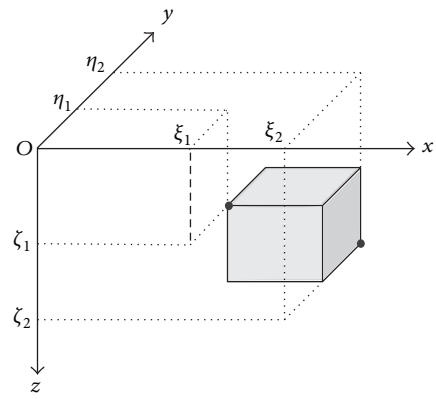

Figure 6: Delimitation of the rectangular prism in the Cartesian plane.

$$G_{zz}(\mathbf{r}) = \gamma \iiint_V \frac{3\left(z - z'\right)^2 - \left(r - r'\right)^2}{\left|r - r'\right|^5} \rho\left(\mathbf{r}'\right) dv. \tag{7}$$

For convenience, we can denote the gravitational tensor in matrix form in the following way:

$$\mathbf{\Gamma} = \begin{bmatrix} G_{xx} & G_{xy} & G_{xz} \\ G_{yx} & G_{yy} & G_{yz} \\ G_{zx} & G_{zy} & G_{zz} \end{bmatrix}, \tag{8}$$

where the trace satisfies the Laplace equation (Trace(Γ) = $G_{xx} + G_{yy} + G_{zz} = 0$).

The response of the tensor component G_{zz} can be approximated in a discrete way due to the fact that a rectangular prism of constant density (ρ) can be approximated as (see Figure 6)

$$G_{zz} = \gamma\rho \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2} \mu_{ijk} \arctan\left(\frac{x_i y_i}{z_k r_{ijk}}\right), \tag{9}$$

where $x_i = x_p - \xi_i$, $y_i = y_p - \eta_i$, $z_i = z_p - \zeta_i$, and $r_{ijk} = \sqrt{x_i^2 + y_i^2 + z_i^2}$ y $\mu_{ijk} = (-1)^i(-1)^j(-1)^k$. The discrete approximation for the remaining components can be reviewed in [9]. In Figure 7, we show the result of calculating the components of a prism measuring 200 m^3 at a height of 0.01 Km, with an observation mesh of 1 Km × 1 Km, and discretized every 20 m.

3. Design of the Parallel Algorithm

In terms of programmability, there are two ways to utilize the Xeon-Phi, (1) in offload mode where the main algorithm is executed in the conventional CPU or host, while the computationally expensive part of the code is transferred to the coprocessor, or (2) in native mode, where the algorithm is executed independently from the CPU or other coprocessors via the system bus. Although the performance is similar, the offload mode allows more flexibility in programming [16].

In order to program application software for the Xeon-Phi, a programming paradigm that allows parallelism is required; this is the case with a x86 SMP architecture. Thus, most of the same multicore CPU tools can be reused; specifically, tools such as Pthreads [17], OpenMP [18], Intel Cilk Plus [19], and OpenCL [20] are available in a transparent way for the architecture. An optimized version of MPI is also available.

Considering different alternatives that are available to develop parallel application software, and specifically for scientific applications, the OpenMP model has been proven to be an excellent tool [21, 22]. Even OpenMP transparency and its better performance in the MIC have been documented [23]. Moreover, as mentioned in the Introduction, the application software was previously migrated to a cluster of conventional CPUs using a hybrid OpenMP-MPI methodology; hence, OpenMP will be the development model for the migration.

Currently, it is possible to integrate more than one Xeon-Phi board per node with the purpose of benefiting from the fast interconnection speeds obtained by integrating various devices on the same PCI bus. By sharing the same mainboard, the devices avoid having to move information through an external network, thereby reducing the latency and overload associated with parallelism. The additional benefit obtained from utilizing OpenMP is that it can be employed as a controller and communicator for the integrated boards at the same node in a nested parallelism scheme [24].

The proposed design is shown in Figure 8; it consists of creating a general observation mesh labeled as G_{shared}, implemented as a bidimensional array in the memory of the CPU controlled by the master thread. Subsequently, two execution threads are created in order to address both Phi boards (one to control each board) with its own private observation mesh, all the while maintaining a shared reference in the global mesh G_{shared}. These private meshes in each thread are labeled as $G_{\text{shared}}^{T_0}$ and $G_{\text{shared}}^{T_1}$, and although they are private for the execution threads in the CPU, they will be shared for the threads which are created in the MIC. Next, the calculation threads in each of the MIC are created; each thread contains its own observation mesh G where the calculated gravimetric anomalies will be stored corresponding to the prisms assigned for processing. Once the calculation is completed, the sum over the $G_{\text{shared}}^{T_0}$ and $G_{\text{shared}}^{T_1}$ meshes is computed, followed finally by the reduction calculation over the global mesh G_{shared}. The nested parallelism design mounted on the MIC boards is shown in Figure 8 and in the pseudocode in Algorithm 1. Fragments of the code in FORTRAN to show how is implemented the Algorithm 1 is showed in Listing 1.

Once the algorithm has been designed for one node of the cluster, the implementation must be extended to be used for more than one node. The option used is MPI to carry out the reductions of data between nodes. Since the parallelization in MPI is explicit, we must manually distribute the number of prisms to be processed through a modular function and the distribution is carried out in the following way: M is the number of prisms to be calculated, n is the number of the MPI processes created, and p is the MPI process number

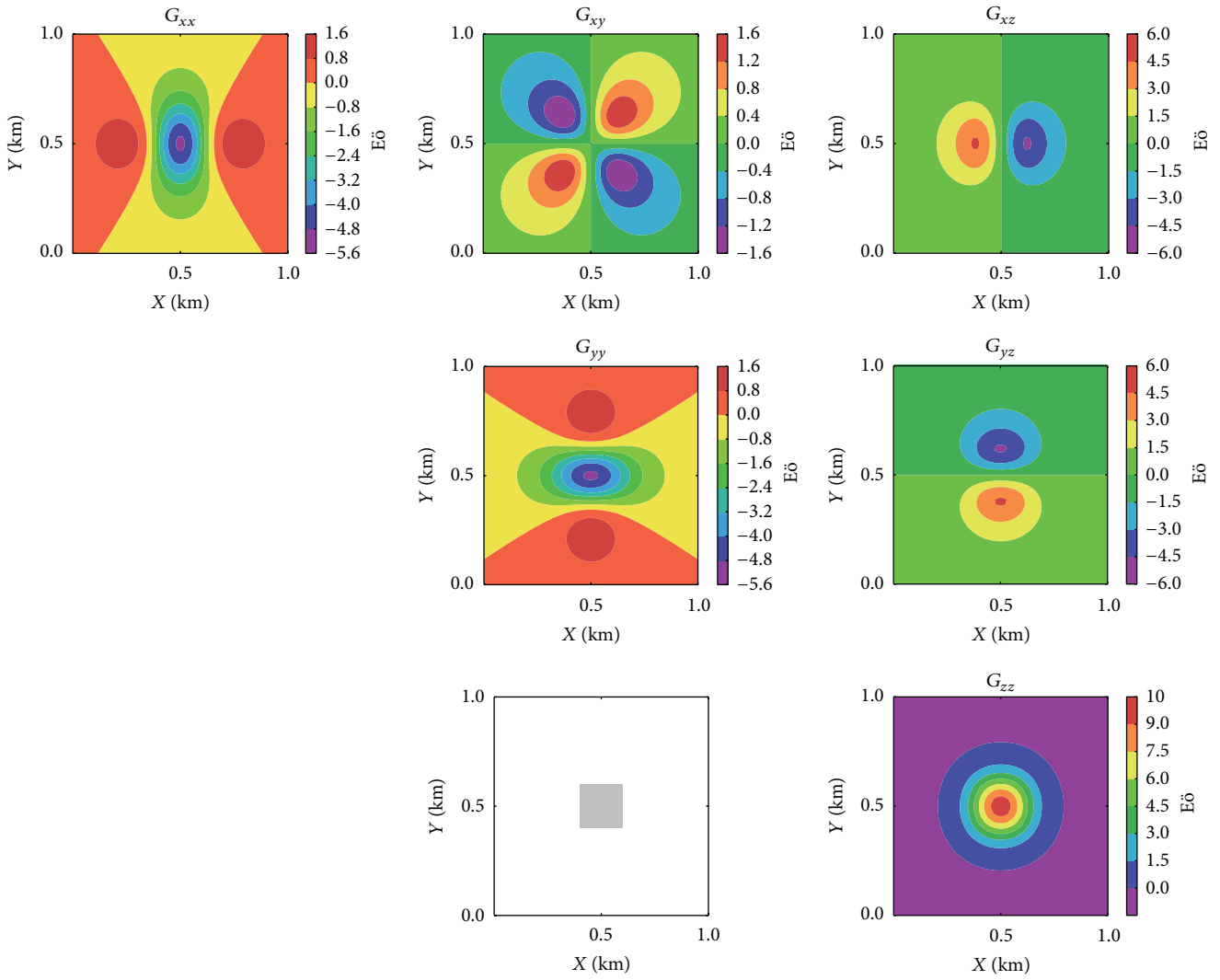

FIGURE 7: Gravity gradient response for a prism buried a depth of 100 m. Each side having a length of 200 m and constant density contrast of 100 kg/m^3.

$(p = 1, 2, 3, \ldots, (n-1))$. We define the initial number of prisms to be processed by the MPI process p as p_{start} and the final number as p_{end} and s is the quotient of the number of prisms M between the total number of processes n and r as the remainder; the procedure to determine p_{start} and p_{end} continues as follows:

$$[s] = \frac{M}{n}, \qquad r = \text{mod}\,(M, n). \tag{10}$$

Therefore,

$$p_{\text{start}} = p \times s + 1, \qquad p_{\text{end}} = (p+1) \times s. \tag{11}$$

If $r \neq 0$ and $p < r$, then we adjust as

$$p_{\text{start}} = p_{\text{start}} + p, \qquad p_{\text{end}} = p_{\text{end}} + (p+1). \tag{12}$$

If $r \neq 0$ and $p \geq r$, then we adjust as

$$p_{\text{start}} = p_{\text{start}} + r, \qquad p_{\text{end}} = p_{\text{end}} + r. \tag{13}$$

In this way, we can distribute the number of prisms M over n MPI processes in a balanced manner; once p_{start} and p_{end} have been determined for each node (one process to one node), the procedure of the Algorithm 1 is applied to internally process the prisms. The design of the algorithm is shown in Figure 9.

4. Performance Validation Experiments

This section reports the results obtained from the performance experiments, first on a conventional Xeon CPU in serial and parallel form with the purpose of having a baseline for comparison with regards to quality and performance of the numerical solution obtained by the Xeon-Phi coprocessor.

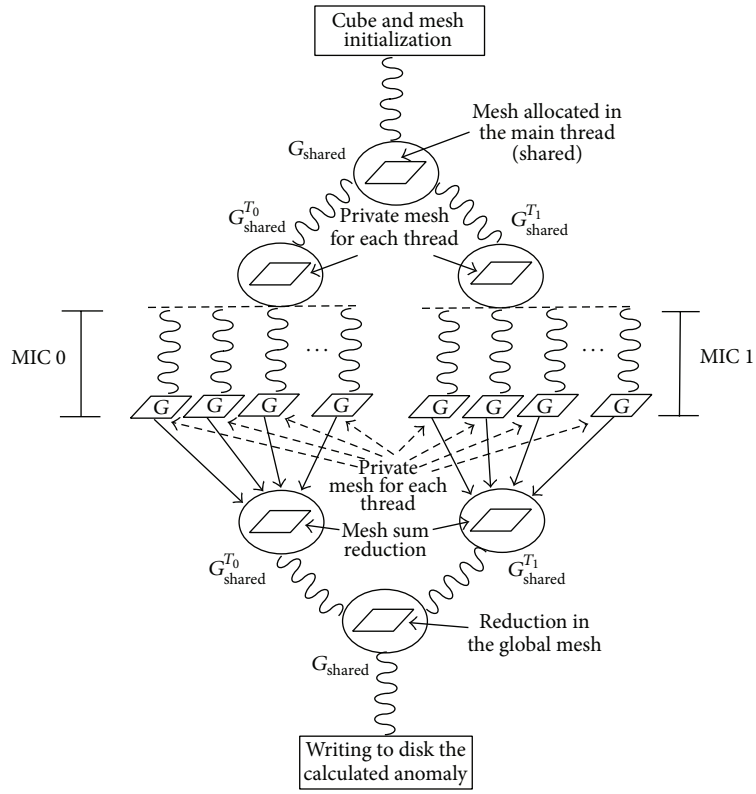

Figure 8: Design of the algorithm for only one node utilizing a nested parallelism model.

Data: The prism assembly
Result: The anomaly G
begin
  Parallel region with q threads
  Allocate($G(m, n)$) where $m \times n$ is the size of the observation mesh;
  begin
    Parallel region with 236 threads in the MIC;
    Allocate($G(m, n)$) for each thread;
    for p_{start} **to** p_{end} **do**
      Call GravAnomBoxMesh(Parameters, G)
    Critical (inner);
    PG = PG + G;
    % Reduction in the MIC
  Critical (outter);
  PGS = PGS + PG;
  % Reduction in the CPU

Algorithm 1: Calculation of the component G.

4.1. Modelling a Real Prisms Assembly. The application of the direct calculation of the FTG components is conducted on a portion of an exploration area in the Gulf of Mexico for which we have the geological information; however, we omit the exact location and the details of the zone due to business confidentiality. The objective is to locate the characteristics related to the distribution and geometry of the saline bodies while integrating the information from the zone (see Figure 10). It shows various horizons including the distribution of saline bodies, and in this way the geological information will determine the precise limits of the density, thereby determining the amplitudes of the anomalies in the area in question.

Basing the model on a Cartesian reference frame and positive depth, the geometry is composed of an assembly of prisms of variable density but with constant dimensions of 500 m in both x and y directions and 50 m in z direction. The domain measures 40.5 × 36.5 × 14.1 Km in the east-west, north-south, and depth directions, respectively (see Figure 11). The total number of prisms that set up the model is 1,667,466 with an observation mesh of 321 × 289 = 92,769 points. The number of prisms with nonzero density is 1,466,424.

The characteristics of the computing node are the following:

(i) Two Intel(R) Xeon(R) processors E5-2680 @ 2.70 Ghz.

(ii) 8 cores per processor (16 cores in total, two sockets).

(iii) 126 GB of RAM.

(iv) 172.8 Glops (double precision) single core, 1382.5 Glops expected using the 8 cores.

```
(1)  !$OMP PARALLEL DEFAULT (NONE) PRIVATE(G, P_G, TID, DEVICE, STATUS) &
         !$OMP & SHARED(P_G_shared,Xm,Ym,height) &
         !$OMP & SHARED(prisms,&
         !$OMP & Xa,Xb,Ya,Yb,Za,Zb, GType, X1p, Y1p, Z1p,posx,posy,posz) &
         !$OMP & SHARED (mic_start,mic_end,density) &
(6)      !$OMP & FIRSTPRIVATE(dxp,dyp,dzp,start,end,Typeoffile,Nx,Ny)&
         !$OMP & NUM_THREADS(n_of_devices)
          ALLOCATE(G(Ny,Nx),stat=status); !Global Shared Mesh
          CALL CHECKMEMORY(status);
          G = 0.0;
(11)      P_G => G;
          TID = OMP_GET_THREAD_NUM()
          !dir$ omp offload target(mic:TID) inout (P_G:alloc_if(.true.)
          free_if(.false.))

(16)      !$OMP PARALLEL DEFAULT(NONE) SHARED(P_G,MIC_INICIO,MIC_FINAL) PRIVATE(G)&
          !$OMP & PRIVATE(DEVICE, TID) FIRSTPRIVATE(Ny,Nx,STATUS,INICIO,FIN)&
          !$OMP & FIRSTPRIVATE(dxp,dyp,dzp,Typeoffile,GTIPO) SHARED(Xa,Xb,Ya,Yb,Za,
              Zb,X1p,Y1p,Z1p,posx,posy,posz)&
          !$OMP & SHARED(Xm,Ym,height,density,prisms) NUM_THREADS(num_threads)

(21)         !$OMP SINGLE
                    prisms = 0;
             !$OMP END SINGLE

             Allocate(G(Ny,Nx), Stat=status) !Local mesh for every thread created
                  in the Phi
(26)         Call CHECKMEMORY (status)
             G(:,:) = 0.0;
             TID = OMP_GET_THREAD_NUM()
             DEVICE = offload_get_device_number()

(31)         !$OMP DO PRIVATE(k)
                 Do k = mic_start(device+1), mic_end (device+1)
                 !Number of prisms to be processed
             If (TYPEOFFILE == 1) Then
                    Xa (k) = X1p (posx(k))
(36)                Xb (k) = Xa (k) + dxp
                    Ya (k) = Y1p (posy(k))
                    Yb (k) = Ya (k) + dyp
                    Za (k) = Z1p (posz(k))
                    Zb (k) = Za (k) + dzp
(41)         End If
             Call GravAnomBoxMalla (Xa(k), Ya(k), Za(k), Xb(k), Yb(k), Zb(k),
                 densidad(k),GTIPO,G,&
                     & Xm, Ym, height, Ny, Nx)
                     !$OMP ATOMIC
                     prismas = prisms + 1
(46)         !$OMP END DO

     !$OMP CRITICAL (INTERIOR)
                      P_G(:,:) = P_G(:,:) + G(:,:);
     !$OMP END CRITICAL (INTERIOR)
(51) !$OMP END PARALLEL
     !$OMP CRITICAL (EXTERIOR)
                     P_G_shared(:,:) = P_G_shared(:,:) + P_G(:,:);
     !$OMP END CRITICAL (EXTERIOR)
     !$OMP END PARALLEL
```

Listing 1: Code fragments in FORTRAN corresponding to the Algorithm 1. It shows how the nested parallelism is implemented to control the two Xeon-Phi coprocessors embedded in the same node using the INTEL LEO extensions.

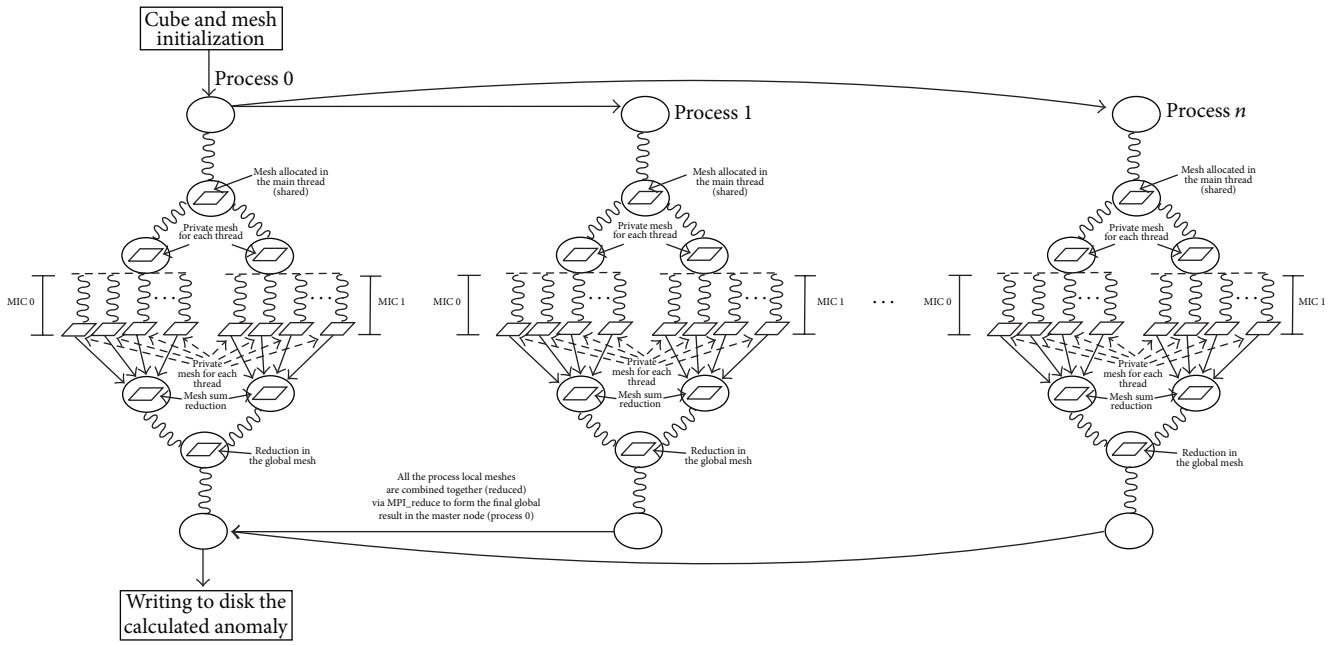

Figure 9: Design of the software application for clustering.

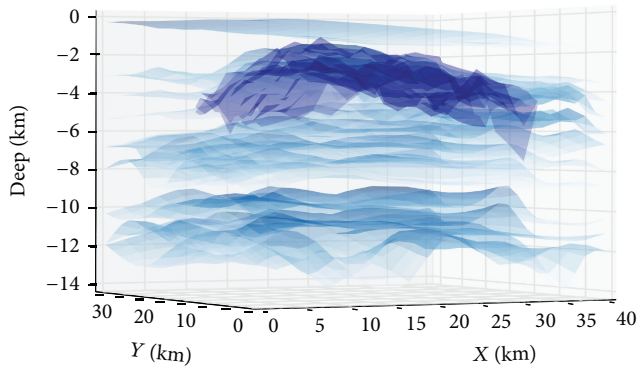

Figure 10: Geological horizons interpolated from previously interpreted seismic sections. Horizons belonging to saline bodies are shown in purple.

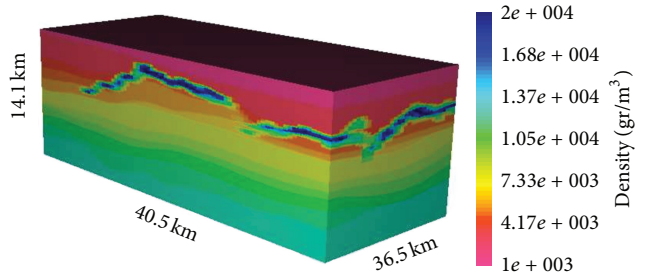

Figure 11: Synthetic 3D model created with prisms. The model is limited with geological horizons (Figure 10) and constant density layers were adjusted.

(v) Two Xeon-Phi coprocessors model 5110P with the following characteristics:

(a) 60 active cores.

(b) Frequency: 1053 MHz.

(c) Size of the GDDR3 memory: 7936 MB.

(d) Performance 1011 Glops.

4.2. Performance Experiments on One Node. The node has a total of 16 real cores of the CPU; therefore, the algorithm will spawn from 1 to 16 execution threads to analyse the performance. The problem that is tackled has a total of 1,466,424 prisms of variable density against an observation mesh of 321 × 289 points and at a height of 100 m above sea level, thereby translating to evaluate a calculation function of 136,038,688, and 056 times per each component of the gravitational tensor. Thus, this problem is a HPC problem. Figure 12 shows the computing times required for 1 thread up to 16 threads to calculate the G_{zz} component.

The serial computing time, in other words, the execution time obtained with only one execution thread, to compute G_{zz} is approximately 8 h 06 m. With this baseline time, if we calculated the entire tensor, we would need more than 48 h of time to compute all components. The required computing time would be unacceptable in an industrial environment where the delays have direct economic costs. Thus, the use of all available cores is necessary to reduce the computing time. Using all the available cores of the CPUs, we reduce the time by 14.11X (34 m 27 s) and consequently all the components could be calculated experimentally in about 3 h 30 min. Figures 12 and 13 show the behaviours in computing time and corresponding speed-ups.

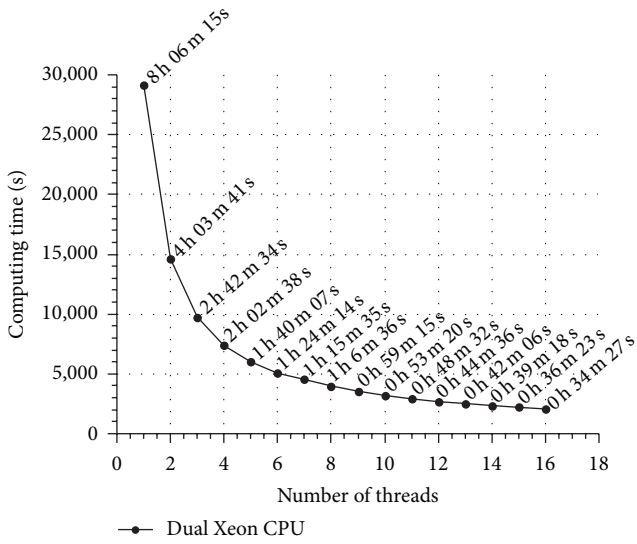

Figure 12: Behaviour of the computation times on the Xeon Dual CPU. One thread is mapped to one core using a compact affinity over the sockets.

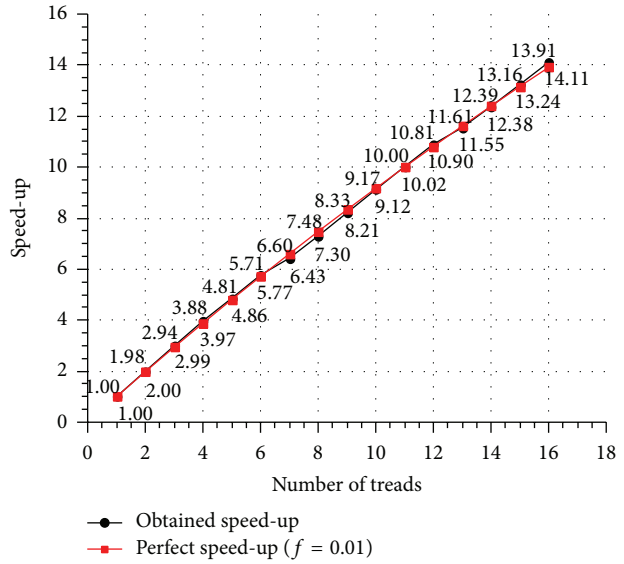

Figure 13: The speed-up corresponding to the execution times of the CPU. The perfect speed-up is calculated using the Amdahl's law $S(n) = n(1 + (n-1) \times 0.01)$; therefore, the fraction serial is estimated around 1%, thus obtaining a near-perfect speed-up.

Once we have conducted CPU experiments, the same data set is processed on the coprocessors. The first consideration is that the Xeon-Phi coprocessor supports the handling of 4 threads per processing core; nevertheless, it is always necessary to determine the optimal number of threads for the application under development since this optimality depends directly on both the type of algorithm and the memory handling within the application. In general, in the Phi architecture, more than one thread per processing core helps to hide the inherent latencies in the application. For

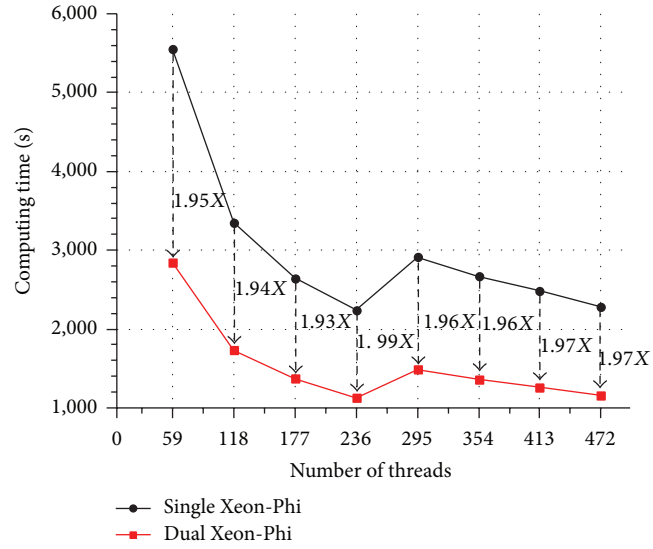

Figure 14: Behavior of the computing time used when calculating G_{zz} on the Xeon-Phi coprocessors, as we can observe the reduction of the time is almost 2X using the two coprocessors.

example, while one thread is waiting on the assignment of a memory resource, others can be attended by the core. In the conventional Xeon architecture, various programmers have found that some applications for HPC do not regularly benefit from the hyperthreading technology [25, 26] but the Phi technology is different, since the multithreading cannot be deactivated like the hyperthreading can.

The most recommended method for a program designed in offload mode is to begin with the creation of various threads from as few as $(n - 1)$ and up to $4 \times (n - 1)$, where n is the number of physical cores in the Phi. Thus, the experiments that must be executed, as recommended directly by Intel, are to create $(n-1)$, $2\times(n-1)$, $3\times(n-1)$, and $4\times(n-1)$ threads. This set of experiments will determine if, with an increase in the number of threads handled by the core, there is still an improvement in the performance of the algorithm. Multiples of $(n - 1)$ are employed rather than n, because one thread is left available for operating system services.

The experiments that are carried out start from $(n - 1)$ up to $8 \times (n - 1)$. Since in this case the Phi coprocessor employed has 60 cores, the experiments will be conducted with 59, 118, 177, 236, 295, 354, 413, and 472 execution threads, using a balanced affinity. First, we carry out the experiments using only one Phi board, followed by using two boards with both Phi boards integrated on one mainboard. The great advantage of the implementation designed in this research is that it permits a transparent communication between the boards, in addition to being scalable, thereby implying that virtually all the Phi boards integrated on the same mainboard could be used. Table 1 shows the resulting computing times obtained and the corresponding speed-up taking the 59 threads in only one coprocessor as the baseline. Figures 14 and 15 show the behavior of the computing time metric and of the speed-up of the Phi boards.

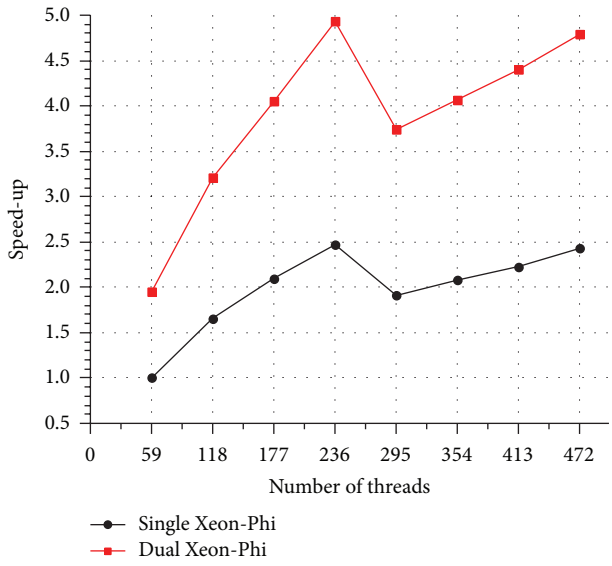

Figure 15: Behavior of the speed-up obtained when calculating G_{zz} on the Xeon-Phi coprocessors. We can observe a near-perfect speed-up 2.0X using the two coprocessors.

Table 1: Computation times when using the Xeon-Phi coprocessors. The best results are obtained when 236 threads are created per coprocessor; it means that we are obtaining a gain using the multithreading (4 threads per core), but when the number of threads is increasing, only an overhead is created.

Threads	Single Xeon-Phi	Dual Xeon-Phi
59	1 h 32 m 31 s	0 h 47 m 24 s
118	0 h 55 m 52 s	0 h 28 m 49 s
177	0 h 44 m 09 s	0 h 22 m 49 s
236	**0 h 37 m 24 s**	**0 h 18 m 46 s**
295	0 h 48 m 30 s	0 h 24 m 45 s
354	0 h 44 m 26 s	0 h 22 m 43 s
413	0 h 41 m 27 s	0 h 21 m 03 s
472	0 h 38 m 02 s	0 h 19 m 19 s

The results indicate, as expected, that the best result occurred with 4 threads per core. Therefore, this application does benefit from utilizing the multithreading since it achieves an improvement of a factor of 2.47X per card if 4 threads per core (optimal number per core) are used, as observed in Figure 15. If the number of threads is taken above 4, then the performance is influenced by the overloading of the threads per core and by the affinity.

Of the obtained results, and taking the best performance results shown in Table 1, we can observe that a Xeon-Phi coprocessor is 13X faster with respect to the serial version and 0.92X faster with respect to the original parallel version on the CPU. The two coprocessors working together are 25.91X faster with respect to the serial version and 1.84X faster with respect to the parallel CPU version. This implies that one Xeon-Phi coprocessor affords a similar performance as the 16 cores offered by the Dual Xeon CPU.

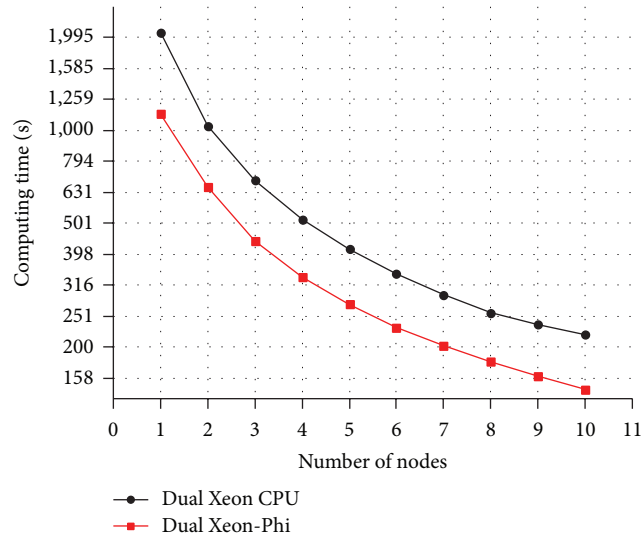

Figure 16: Behaviour of the computing time obtained for the estimation of G_{zz} when using 10 nodes of the cluster (the scale is semilogarithmic base 10 on the time axis). The curves compare the behaviour of the times obtained using 16 threads on the CPU versus 236 threads for each Xeon-Phi (these are the best results for both cases). As expected, the behaviour is not completely linear (but almost) as we can see when the speed-up is calculated (see Figure 17).

Table 2: Computing times and corresponding speed-up obtained when calculating G_{zz} on the Xeon Dual CPU and the Dual Xeon-Phi while using ten nodes of the cluster. The second speed-up listed in the Phi column is based on the time of the Dual Xeon CPU as a baseline.

Nodes	Dual Xeon CPU	Dual Xeon-Phi
1	34 m 27 s	18 m 46 s
2	17 m 12 s	10 m 56 s
3	11 m 28 s	07 m 19 s
4	08 m 36 s	05 m 36 s
5	06 m 53 s	04 m 34 s
6	05 m 44 s	03 m 51 s
7	04 m 55 s	03 m 22 s
8	04 m 18 s	02 m 59 s
9	03 m 56 s	02 m 41 s
10	**03 m 39 s**	**02 m 26 s**

4.3. Performance Experiments on the Cluster. Ten nodes of the cluster were employed to distribute the computing. Table 2 shows the times obtained from using the Dual Xeon CPU and the Dual Xeon-Phi.

Figures 16 and 17 show the behavior of the computing time and the speed-up corresponding to the information in Table 2. Figure 17 shows that the behaviour of the speed-up when using the Dual Xeon CPU in the nodes is quasiperfect meaning that the communication overload is virtually nil. When using the Dual Xeon-Phi, however, the overload is greater and the behaviour deviates from the perfect speed up of 14.16, thereby reducing the serial time by a factor of 199.83X; it is important to clarify that this final factor was

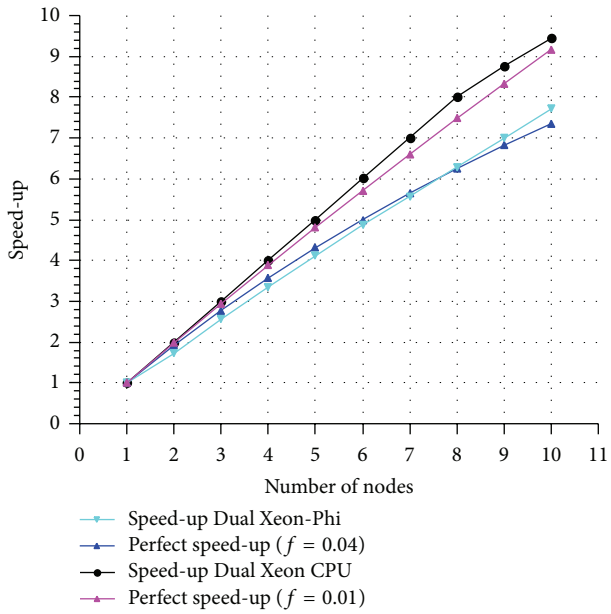

Figure 17: Behaviour of the speed-up when using ten nodes of the cluster. The serial fraction using the Xeon CPU is around 1% and for the Xeon-Phi, 4%; this means that the Xeon-Phi nodes introduce more overhead although they are faster; the speed-ups are calculated using their corresponding baseline, that is, the computation time used by one node.

Table 3: Errors of the L2 normal for different platforms with respect to the serial version.

Platform	Error L2 (G_{zz})
CPU multicore (16 threads)	$4.568821042594709e-13$
Dual Xeon-Phi (236×2 threads) Cluster CPU	$1.435794068840153e-12$
multicore (16 threads, 10 nodes) Cluster Dual Xeon	$4.543626871060080e-13$
Phi (236×2 threads, 10 nodes)	$5.417837921550926e-13$

calculated using one core of the CPU versus the 20 Xeon-Phi; therefore, it is not very formal but still could be useful.

4.4. Validation of the Numerical Code. The main objective of parallel programming is to decompose the program into appropriate components that can be executed simultaneously, thereby reducing the overall execution time. Although execution time is the primary goal, the implementation must be validated as well, since programming errors, inherent to parallelism, may occur. We use the error of the L2 norm to calculate the numerical differences between the different platforms with respect to the component G_{zz} taking as baseline the current serial version. We omit the differences for the remainder of the components since they are likely similar. The errors are shown in Table 3.

From the obtained errors, we show that there are no significant numerical differences; therefore, the version of the application that is migrated to the Xeon-Phi has a correct implementation.

5. Discussion about the Forward Modelling Results

The FTG data differ in many ways from the conventional high-resolution gravity data. One important difference is the high density during data collection. This increase in the sensitivity allows a much greater resolution and is the reason why the data can be incorporated successfully into the interpretation of a gas or oil hydrocarbon reservoir.

The tensor results obtained from G are shown in Figure 18; these results offer an important understanding into the definition of the geometry of bodies in order to characterize the structures of economic interest, since the FTG data were able to detect superficial bodies, bodies with measured and exact amplitudes in a range of −10 to 7 Eotvos. It is also possible to identify the principle tendencies of the geological structures in the study area and variations in the densities present in these anomalies. The resolution obtained with this technology is reasonably good, but we expect to improve these methods in the near future via better acquisition techniques and the ability to obtain even better resolution. In this way, we will be able to characterize the exploratory reservoirs in deep waters with better precision and definition.

6. Conclusions

This work implements and validates a parallel design based on a nested parallelization scheme for the calculation of gravitational components in OpenMP + MPI on a Xeon-Phi cluster architecture. The numerical experiments and the performance metrics validate that the implementation is efficient and leads to results which are very close to perfect speed-ups.

The results show a good design and adequate handling of the memory in order to exploit the benefits of the Xeon-Phi. In this case, the nested design in OpenMP allowed a transparent communication between the coprocessors, while the scheme of utilizing different regions of the memory per thread (observation mesh) was very advantageous with respect to the performance. This design benefits from the multithreading technology found on the Xeon-Phi, thereby improving the performance of this application by 147%.

With respect to the results obtained from the application software, the results provide very important insights into the geometry body definition. These insights help characterize the structures that are of financial importance since the algorithm is able to detect superficial bodies that are less than 300 m wide and with less than 7 Eotvos of anomaly. The results also identified the principle tendencies of the geological structures of the study area and the variations in density which these anomalies present. The resolution achieved via this technology is reasonably good. In the future, however, we expect that the processing and acquisition methods can be further improved and perfected in order to achieve even better resolution, thereby characterizing the exploratory oil reservoirs in deep waters with better precision and definition. Without a doubt, these high performance computing applications will be a requisite part of the solution to deep water exploration.

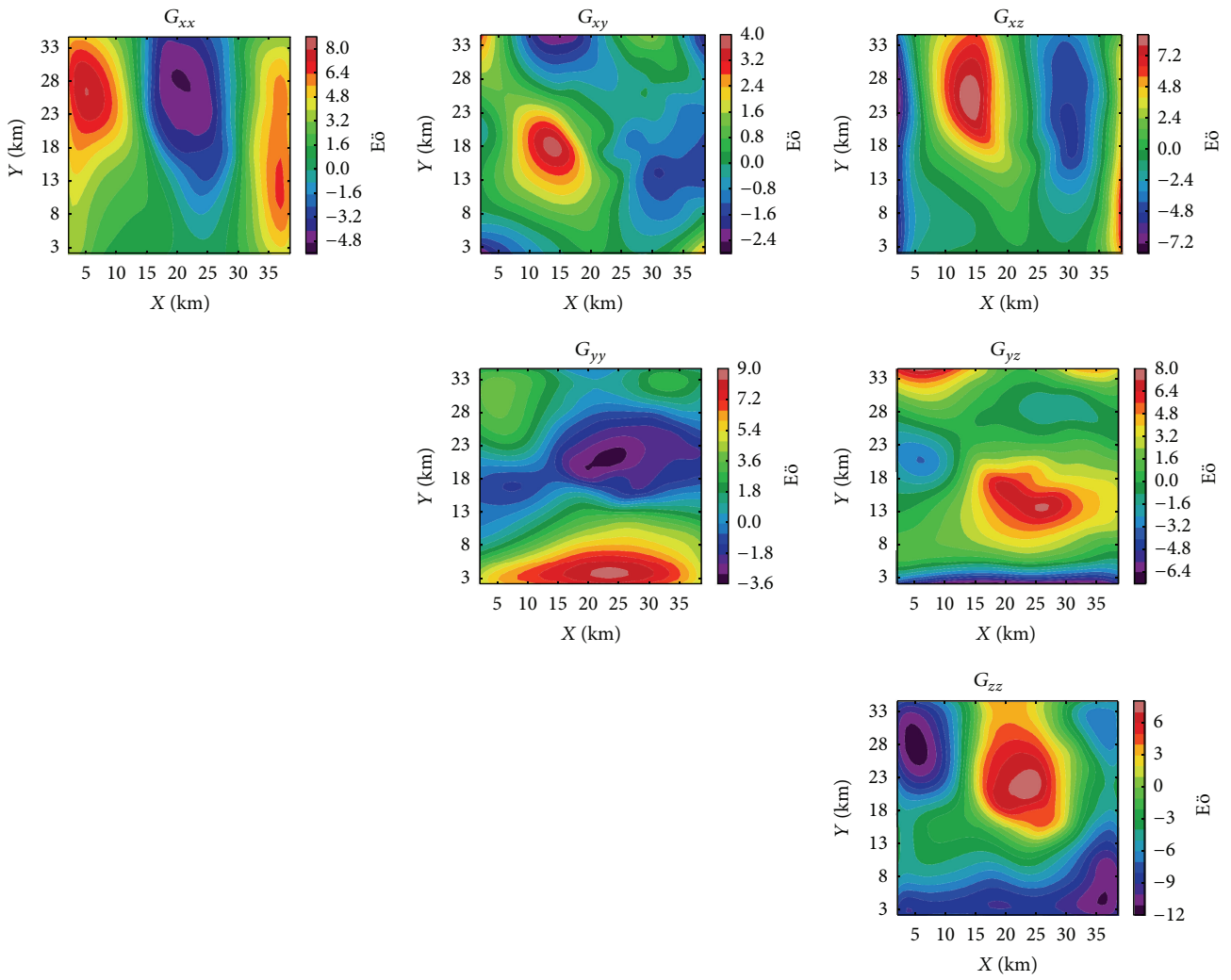

FIGURE 18: Free-air tensor components calculated from the geological horizons corresponding to Figure 11.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors wish to thank the support and opportunity offered by the CNS-IPYCIT to work on their equipment (http://www.cns-ipicyt.mx/) and the project ABACUS, CONACyT number EDOMEX-2011-C01-165873, for the grant given to the first two authors. Alfredo Trujillo-Alcantara wants to thank the support, in order to publish the results, provided by the program SENER-CONACYT and the hospitality of the doctoral program of the ESFM-IPN (http://esfm.ipn.mx/).

References

[1] V. W. Lee, C. Kim, J. Chhugani et al., "Debunking the 100X GPU vs. CPU Myth: an evaluation of throughput computing on CPU and GPU," in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*, vol. 38, pp. 451–460, ACM, Saint-Malo, France, June 2010.

[2] S. Wienke, D. Plotnikov, D. An Mey et al., "Simulation of bevel gear cutting with GPGPUs—performance and productivity," *Computer Science—Research and Development*, vol. 26, no. 3-4, pp. 165–174, 2011.

[3] K. W. Schulz, R. Ulerich, N. Malaya, P. T. Bauman, R. Stogner, and C. Simmons, "Early experiences porting scientific applications to the many integrated core (mic) platform," in *Proceedings of the TACC-Intel Highly Parallel Computing Symposium*, Austin, Tex, USA, 2012.

[4] G. Chrysos, "Intel Xeon Phi coprocessors—the architecture," Intel Whitepaper, 2014.

[5] R. Bell, "Gradiometr ía de gravedad," *Investigación y Ciencia: Edición Española de Scientific American*, no. 263, pp. 50–55, 1998.

[6] R. Tenzer and P. Novák, "Effect of crustal density structures on GOCE gravity gradient observables," *Terrestrial, Atmospheric and Oceanic Sciences*, vol. 24, no. 5, pp. 793–807, 2013.

[7] B. Heck and K. Seitz, "A comparison of the tesseroid, prism and point-mass approaches for mass reductions in gravity field

modelling," *Journal of Geodesy*, vol. 81, no. 2, pp. 121–136, 2007.

[8] D. Nagy, G. Papp, and J. Benedek, "The gravitational potential and its derivatives for the prism," *Journal of Geodesy*, vol. 74, no. 7-8, pp. 552–560, 2000.

[9] C. Couder-Castañeda, J. C. Ortiz-Alemán, M. G. Orozco-del-Castillo, and M. Nava-Flores, "Forward modeling of gravitational fields on hybrid multi-threaded cluster," *Geofísica Internacional*, vol. 54, no. 1, pp. 31–48, 2015.

[10] C. Couder-Castañeda, C. Ortiz-Alemán, M. G. Orozco-Del-Castillo, and M. Nava-Flores, "TESLA GPUs versus MPI with OpenMP for the forward modeling of gravity and gravity gradient of large prisms ensemble," *Journal of Applied Mathematics*, vol. 2013, Article ID 437357, 15 pages, 2013.

[11] Y. Zhang, M. Burcea, V. Cheng, R. Ho, and M. Voss, "An adaptive OpenMP loop scheduler for hyperthreaded SMPs," in *Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS '04)*, 2004.

[12] O. Boulanger and M. Chouteau, "Constraints in 3D gravity inversion," *Geophysical Prospecting*, vol. 49, no. 2, pp. 265–280, 2001.

[13] M. Čuma, G. A. Wilson, and M. S. Zhdanov, "Large-scale 3D inversion of potential field data," *Geophysical Prospecting*, vol. 60, no. 6, pp. 1186–1199, 2012.

[14] M. S. Zhdanov, R. Ellis, and S. Mukherjee, "Three-dimensional regularized focusing inversion of gravity gradient tensor component data," *Geophysics*, vol. 69, no. 4, pp. 925–937, 2004.

[15] L. B. Pedersen and T. M. Rasmussen, "The gradient tensor of potential field anomalies: some implications on data collection and data processing of maps," *Geophysics*, vol. 55, no. 12, pp. 1558–1566, 1990.

[16] J. Reinders, *An Overview of Programming for Intel Xeon Processors and Intel Xeon Phi Coprocessors*, Intel Corporation, Santa Clara, Calif, USA, 2012.

[17] G. E. Allen and B. L. Evans, "Real-time sonar beamforming on workstations using process networks and POSIX threads," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 921–926, 2000.

[18] M. Curtis-Maury, X. Ding, C. D. Antonopoulos, and D. S. Nikolopoulos, "An evaluation of OpenMP on current and emerging multithreaded/multicore processors," in *OpenMP Shared Memory Parallel Programming: Proceedings of the International Workshops, IWOMP 2005 and IWOMP 2006, Eugene, OR, USA, June 1–4, 2005, Reims, France, June 12–15, 2006*, vol. 4315 of *Lecture Notes in Computer Science*, pp. 133–144, Springer, Berlin, Germany, 2008.

[19] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou, "Cilk: an efficient multithreaded runtime system," *Journal of Parallel and Distributed Computing*, vol. 37, no. 1, pp. 55–69, 1996.

[20] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: a parallel programming standard for heterogeneous computing systems," *Computing in Science and Engineering*, vol. 12, no. 3, Article ID 5457293, pp. 66–72, 2010.

[21] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[22] M. Curtis-Maury, X. Ding, C. D. Antonopoulos, and D. S. Nikolopoulos, "An evaluation of openmp on current and emerging multithreaded/multicore processors," in *OpenMP Shared Memory Parallel Programming: International Workshops, IWOMP 2005 and IWOMP 2006, Eugene, OR, USA, June 1–4, 2005, Reims, France, June 12–15, 2006. Proceedings*, vol. 4315 of *Lecture Notes in Computer Science*, pp. 133–144, Springer, Berlin, Germany, 2008.

[23] T. Cramer, D. Schmidl, M. Klemm, and D. an Mey, "OpenMP programming on Intel Xeon Phi coprocessors: an early performance comparison," in *Proceedings of the Many-Core Applications Research Community Symposium (MARC '12)*, pp. 38–44, RWTH Aachen University, 2012.

[24] D. Schmidl, C. Terboven, D. an Mey, and M. Bücker, "Binding nested openmp programs on hierarchical memory architectures," in *Beyond Loop Level Parallelism in OpenMP: Accelerators, Tasking and More*, M. Sato, T. Hanawa, M. Müller, B. Chapman, and B. de Supinski, Eds., vol. 6132 of *Lecture Notes in Computer Science*, pp. 29–42, Springer, Berlin, Germany, 2010.

[25] U. Ranok, S. Kittitornkun, and S. Tongsima, "A multithreading methodology with OpenMP on multi-core CPUs: SNPHAP case study," in *Proceedings of the 8th Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI '11)*, pp. 459–463, May 2011.

[26] C. Leggett, S. Binet, K. Jackson, D. Levinthal, M. Tatarkhanov, and Y. Yao, "Parallelizing ATLAS reconstruction and simulation: issues and optimization solutions for scaling on multi- and many-CPU Platforms," *Journal of Physics: Conference Series*, vol. 331, no. 4, Article ID 42015, 2011.