A Simulation of Seismic Wave Propagation at High Resolution in the Inner Core of the Earth on 2166 Processors of MareNostrum

Dimitri Komatitsch[1,2], Jesús Labarta[3], and David Michéa[1]

[1] Université de Pau et des Pays de l'Adour,
CNRS UMR 5212 & INRIA Sud-Ouest Magique-3D,
Avenue de l'Université, 64013 Pau Cedex, France
{dimitri.komatitsch,david.michea}@univ-pau.fr
http://www.univ-pau.fr
[2] Institut universitaire de France, 103 boulevard Saint-Michel, 75005 Paris, France
http://www.cpu.fr/Iuf
[3] Barcelona Supercomputing Center, Technical University of Catalonia,
c/ Jordi Girona, 31 - 08034 Barcelona, Spain
jesus@cepba.upc.es
http://www.bsc.es

Abstract. We use 2166 processors of the MareNostrum (IBM PowerPC 970) supercomputer to model seismic wave propagation in the inner core of the Earth following an earthquake. Simulations are performed based upon the spectral-element method, a high-degree finite-element technique with an exactly diagonal mass matrix. We use a mesh with 21 billion grid points (and therefore approximately 21 billion degrees of freedom because a scalar unknown is used in most of the mesh). A total of 2.5 terabytes of memory is needed. Our implementation is purely based upon MPI. We optimize it using the ParaVer analysis tool in order to significantly improve load balancing and therefore overall performance. Cache misses are reduced based upon renumbering of the mesh points.

Keywords: load balancing, cache misses, mesh partitioning, seismic wave propagation, global Earth.

1 Introduction

Modeling of seismic wave propagation resulting from large earthquakes in the three-dimensional (3D) Earth is of considerable interest in seismology because analyzing seismic wave propagation in the Earth is one of the few ways of studying the structure of the Earth's interior, based upon seismic tomography. Seismic waves resulting from earthquakes can be classified in two main categories: body waves, which propagate inside the medium and are of two types: compressional (pressure) waves, called P waves, and shear waves, called S waves; and surface waves, which travel along the surface of the medium and have an exponentially decreasing amplitude with depth. The analysis of the 3D geophysical structure of

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 364–377, 2008.

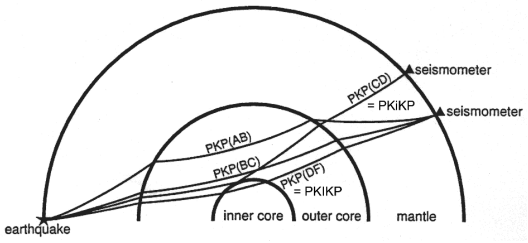

Fig. 1. Sketch of how PKP seismic phases propagate in the Earth after an earthquake, i.e. illustration of their ray paths. They therefore carry information about the anisotropic structure of the inner core.

the Earth therefore requires the ability to calculate accurate numerical seismograms (time series representing the three component of displacement at points located on the surface of the Earth). In particular, pressure waves can be used to study the solid inner core of the Earth and its anisotropy (i.e. varying seismic wave speed in different spatial directions). Figure 1 shows a sketch of PKP seismic phases, which are pressure waves that travel inside the core of the Earth. They travel through the Earth's mantle, then through its fluid outer core and solid inner core, then again in the mantle, and then reach the surface, where they are recorded.

The field of numerical modeling of seismic wave propagation in 3D geological media has significantly evolved in the last decade due to the introduction of the spectral-element method (SEM), which is a high-degree version of the finite-element method that is very accurate for linear hyperbolic problems such as wave propagation, having very little intrinsic numerical dispersion. It combines the flexibility of the finite-element method with the accuracy of the pseudospectral method. In addition, the mass matrix is exactly diagonal by construction, which makes it much easier to implement on parallel machines because no linear system needs to be inverted. The 3D SEM was first used in seismology for local and regional simulations (e.g., [1,2,3]) and then adapted to wave propagation at the scale of the full Earth (e.g., [4,5]). Until recently, at the scale of the global Earth available computer resources intrinsically limited such large calculations. For instance, on a PC cluster with 150 processors, Komatitsch and Tromp [4] reached a maximum seismic frequency of 0.0555 Hz, and on 1944 processors of the Japanese Earth Simulator Komatitsch et al. [6] reached a maximum seismic frequency of 0.2 Hz. Such frequencies are not high enough to capture important differential effects on seismic wave propagation related to anisotropy in the inner core of the Earth. Here we implement the SEM on MareNostrum, the world's number 13 supercomputer as of the November 2007 Top500 list of supercomputers, which is located in Barcelona, Catalonia, Spain. We show that on 2166 of its IBM PowerPC 970 processors we can simulate seismic waveforms accurately up to a maximum frequency of 0.5 Hz based upon message passing with MPI.

2 Spatial and Temporal Discretization of the Governing Equations

We consider a linear anisotropic elastic rheology for the heterogeneous solid Earth, and therefore the seismic wave equation can be written in the strong (i.e., differential) form as:

$$\begin{aligned} \rho\ddot{\mathbf{u}} &= \boldsymbol{\nabla}\cdot\boldsymbol{\sigma}+\mathbf{f}\ , \\ \boldsymbol{\sigma} &= \mathbf{C}:\boldsymbol{\varepsilon}\ , \\ \boldsymbol{\varepsilon} &= \tfrac{1}{2}[\boldsymbol{\nabla}\mathbf{u}+(\boldsymbol{\nabla}\mathbf{u})^T]\ , \end{aligned} \tag{1}$$

where $\mathbf{u}$ denotes the displacement vector, $\boldsymbol{\sigma}$ the symmetric, second-order stress tensor, $\boldsymbol{\varepsilon}$ the symmetric, second-order strain tensor, $\mathbf{C}$ the fourth-order stiffness tensor, ρ the density, and $\mathbf{f}$ an external force. The tensor product is denoted by a colon, a superscript T denotes the transpose, and a dot over a symbol indicates time differentiation. The physical domain of the model is denoted by Ω and its outer boundary by Γ. The material parameters of the solid, $\mathbf{C}$ and ρ, can be spatially heterogeneous. We can then rewrite the system (1) in a weak (i.e., variational) form by dotting it with an arbitrary test function $\mathbf{w}$ and integrating by parts over the whole domain:

$$\int_\Omega \rho\mathbf{w}\cdot\ddot{\mathbf{u}}\,\mathrm{d}\Omega+\int_\Omega \boldsymbol{\nabla}\mathbf{w}:\mathbf{C}:\boldsymbol{\nabla}\mathbf{u}\,\mathrm{d}\Omega=\int_\Omega \mathbf{w}\cdot f\,\mathrm{d}\Omega+\int_\Gamma(\boldsymbol{\sigma}\cdot\hat{\mathbf{n}})\cdot\mathbf{w}\,\mathrm{d}\Gamma\,. \tag{2}$$

The free surface (i.e., traction free) boundary condition is easily implemented in the weak formulation since the integral of traction $\boldsymbol{\tau}=\boldsymbol{\sigma}\cdot\hat{\mathbf{n}}$ along the boundary simply vanishes when we set $\boldsymbol{\tau}=\mathbf{0}$ at the free surface of the Earth.

This formulation is solved on a mesh of hexahedral elements in 3D, which honors both the free surface of the model and its main internal discontinuities (i.e., its geological layers). The unknown wave field is expressed in terms of high-degree Lagrange polynomials of degree N on Gauss-Lobatto-Legendre interpolation (GLL) points, which results in a diagonal mass matrix that leads to a simple time integration scheme (e.g., [1,3]). Because that matrix is diagonal, no linear system needs to be inverted and the method lends itself well to calculations on large parallel machines with distributed memory. Let $\mathbf{w}_N$, $\mathbf{u}_N$ denote the piecewise-polynomial approximations of the test functions and the displacement respectively. Making use of (2), the discrete variational problem to be solved can thus be expressed as: for all time t, find $\mathbf{u}_N$ such that for all $\mathbf{w}_N$ we have:

$$\int_\Omega \rho\mathbf{w}_N\cdot\ddot{\mathbf{u}}_N\,\mathrm{d}\Omega+\int_\Omega \boldsymbol{\nabla}\mathbf{w}_N:\mathbf{C}:\boldsymbol{\nabla}\mathbf{u}_N\,\mathrm{d}\Omega=\int_\Omega \mathbf{w}_N\cdot f\,\mathrm{d}\Omega\,. \tag{3}$$

We can rewrite this system (3) in matrix form as:

$$M\ddot{d}+Kd=F\ , \tag{4}$$

where M is the diagonal mass matrix, F is the source term, and K is the stiffness matrix. For detailed expression of these matrices, the reader is referred for instance to [3].

Time discretization of the second-order ordinary differential equation (4) with a time step Δt is achieved using the explicit Newmark central finite-difference scheme [7] which is second-order accurate and conditionally stable :

$$M\ddot{\mathbf{d}}_{n+1} + K\mathbf{d}_{n+1} = F_{n+1} \ , \tag{5}$$

where

$$\mathbf{d}_{n+1} = \mathbf{d}_n + \Delta t\dot{\mathbf{d}}_n + \frac{\Delta t^2}{2}\ddot{\mathbf{d}}_n \ \text{ and } \ \dot{\mathbf{d}}_{n+1} = \dot{\mathbf{d}}_n + \frac{\Delta t}{2}[\ddot{\mathbf{d}}_n + \ddot{\mathbf{d}}_{n+1}] \ . \tag{6}$$

At the initial time $t = 0$, null initial conditions are assumed i.e., $\mathbf{d} = \mathbf{0}$ and $\dot{\mathbf{d}} = \mathbf{0}$. The stability condition is $\Delta t(c_p/\Delta x)_{\max} \leq c$, where Δx is the distance between two adjacent grid points, c_p is the speed of the pressure waves in the geological medium, and c is a constant that is of the order of 0.5 [8].

3 Implementation of the Spectral-Element Method on MareNostrum

3.1 Acoustic/Elastic Formulation

We are interested in differential effects on very high frequency (0.5 Hertz) seismic phases when they propagate inside the solid inner core of the Earth, therefore to significantly reduce the computational cost we suppress the crust of the Earth and replace it with an extended upper mantle, and convert the whole mantle from elastic to acoustic, thus reducing the problem in that part of the model from a vectorial unknown to a scalar unknown, i.e. reducing memory usage and CPU cost by a factor of roughly three in 3D. In the acoustic mantle and crust we solve the acoustic wave equation in terms of a fluid potential [4]. We keep a (much more expensive to solve) elastic anisotropic medium in the inner core only. In that small part of the mesh we also model seismic attenuation (i.e., loss of energy by viscoelasticity), which has a significant impact on the cost of that small part of the simulation because memory requirements increase by a factor of roughly 2 and CPU time by a factor of roughly 1.5 [4].

3.2 Mesh Generation

Figure 2 shows a global view at the surface of the Earth of the spectral-element mesh we designed, which is accurate up to a frequency of 0.5 Hertz for pressure waves and which fits on 2166 processor cores (6 blocks of 19×19 slices). The sphere is meshed using hexahedra only, based upon an analytical mapping from the six sides of a unit cube to a six-block decomposition of the surface of the sphere called the 'cubed sphere' [9,4,5]. Each of the six sides of the 'cubed sphere' mesh is divided into 19×19 slices, shown with different colors, for a total of 2166 slices. We allocate one slice and therefore one MPI process per processor core (which means that in the remainder of the article when we say 'one processor' for simplicity we actually mean 'one processor core').

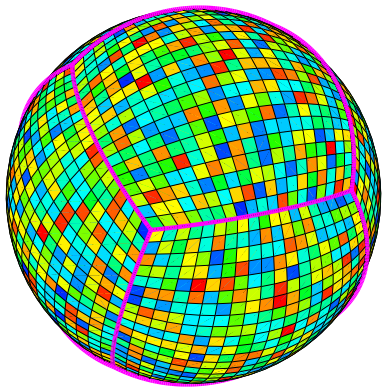

Fig. 2. The SEM uses a mesh of hexahedral finite elements on which the wave field is interpolated by high-degree Lagrange polynomials on Gauss-Lobatto-Legendre integration points. The figure shows a global view of the mesh at the surface, illustrating that each of the six sides of the so-called 'cubed sphere' mesh is divided into 19 × 19 slices, shown here with different colors, for a total of 2166 slices (i.e., one slice per processor core).

The total number of spectral elements in this mesh is 323 millions, which corresponds to a total of approximately 21 billion global grid points (the 'equivalent' of a 2770 × 2770 × 2770 grid), because each spectral element contains $(N+1)^3 = 5 \times 5 \times 5 = 125$ grid points since we use polynomial basis functions of degree $N = 4$, but with points on its faces shared by neighboring elements. This in turn also corresponds to approximately 21 billion degrees of freedom because a scalar unknown is used in most of the mesh (everywhere except in the inner core of the Earth, as mentioned above). Such simulations use a total of approximately 2.5 terabytes of memory.

Our SEM software package is called SPECFEM3D. Version 1.0 was released in 1999 and the current stable version is 3.6. In order to be able to run our large-scale calculations on MareNostrum, we had to fix some significant load balancing issues in version 3.6 and therefore produce a new version called 4.0. Below we discuss the main improvements made.

3.3 Type of Operations Performed at Each Time Step

At each iteration of the serial time loop, which are all identical, four main types of operations are performed:

- update of global arrays, for instance: for each i from 1 to Npoint,
displacement[i] += Δt velocity[i] + Δt^2 acceleration[i] / 2,
where displacement, velocity and acceleration are three global arrays, and Npoint is the number of grid points
- relatively large and purely local calculations of the product of predefined derivation matrices with a local copy of the displacement vector along cut planes

in the three directions (i, j and k) of a 3D spectral element, which contains $(N+1)^3 = 125$ points; therefore, each index i, j or k varies between 1 and N+1

- element-by-element products and sums of arrays of dimension (N+1,N+1, N+1,Nspec), where Nspec is the number of spectral elements, which involve a quadruple loop on the dimensions of the arrays

- sum of the contributions (which are elemental force vectors from a physical point of view) computed locally in arrays of dimension (N+1,N+1,N+1,Nspec) into global arrays of dimension Npoint using indirect addressing. This sum is called the 'assembly' process in finite elements.

3.4 Exploiting Locality

Increasing and exploiting locality of memory references is an important optimization technique. Locality must be optimized in loops that tend to reference arrays or other data structures by indices. The principle of locality deals with the process of accessing a single resource multiple times; in particular regarding temporal locality (a resource referenced at one point in time will be referenced again sometime in the near future) and spatial locality (the likelihood of referencing a resource is higher if a resource in the same neighborhood has been referenced). Memory should therefore be accessed sequentially as often as possible.

In the spectral-element method these are important issues because, as can be seen in Figure 3 drawn in 2D, each spectral element contains $(N+1)^2 = 25$ GLL points, and points lying on edges or corners (as well as on faces in 3D) are shared between elements. The contributions to the global system of degrees of freedom,

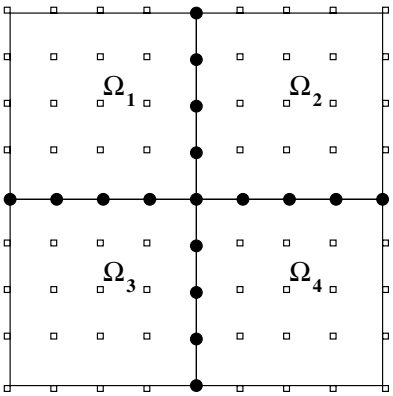

Fig. 3. Illustration of the local and global meshes for a four-element 2D spectral-element discretization with a polynomial degree of $N = 4$. Each element contains $(N+1)^2 = 25$ Gauss-Lobatto-Legendre points. Points lying on edges or corners (as well as on faces in 3D) are shared between elements. The contributions to the global system of degrees of freedom, computed separately on each element, have to be summed at the common points represented by black dots. Corners can be shared by any number of elements depending on the topology of the mesh, which is in most cases non structured.

computed separately on each element, therefore have to be summed at the common points, and the corners can be shared by any number of elements depending on the topology of the mesh, which is in most cases (included ours) non structured. For instance in 3D for a regular hexahedral mesh there are $(N+1)^3 = 125$ GLL integration points in each element, of which 27 belong only to this element (21.6%), 54 belong to two elements (43.2%), 36 belong to four elements (28.8%) and eight belong to 8 elements (6.4%). Hence, 78.4% of the points belong to at least two elements and it is therefore interesting to reuse these points by keeping them in the cache. During mesh creation we must therefore optimize the global numbering which maps point (i,j,k) of local element num_element to a unique global point number (after removal of the duplicated common points shared by more than one element) called global_addressing(num_element, i, j, k).

To do this, the mesh is created element by element, then the common points are identified, and a global addressing is created. This array must be reordered once and for all in the mesher to optimize the future memory access order of the points and elements in the solver, in order to maximize spatial and temporal locality and to access memory sequentially as often as possible. Simple renumbering based on looping on the elements in the mesher in the order in which they will be accessed in the solver and masking points already found works fine for this problem. A more sophisticated approach is to also change the order of the elements (in addition to the numbering of the points) based on the classical reverse Cuthill-McKee algorithm [10] to reduce the average memory strides to access the points shared by several elements, but improvements are very small in the case of the spectral-element method because one spectral element fits entirely in the Level 1 (L1) cache, several tens of spectral elements fit in the L2 cache and a large number of operations are performed on each spectral element because it contains 125 points. Detailed tests not shown here have shown us that it is not necessary in practice to use this algorithm for our problem.

The elementary contributions (internal mechanical forces) from each mesh element are computed based upon products of cut planes in the 3D memory block representing the element with a matrix called the 'derivation matrix' in order to compute the derivative of a given vector or scalar field. We therefore thought about using Basic Linear Algebra Subroutines (BLAS3) 'sgemm' calls in version 4.0 of the code instead of the Fortran loops used in version 3.6. However, this turned out to be inefficient for several reasons. First, these matrix-matrix products are performed on 2D cut planes that are extracted from different (orthogonal) directions of a given 3D memory block. Therefore, in order to use BLAS3 we need to perform some memory copies from the 3D blocks to 2D matrices for some (but not all) of the BLAS3 calls, in order for the input matrix to be correctly structured, which induces significant overhead. Second, these matrices are very small in each element (5×5 or 25×5) and therefore the matrix operations are too small to be efficiently replaced by BLAS3 calls because the overhead is large. Even if we compute all the elements (whose number is large in each mesh slice, typically more than 100,000) simultaneously with one BLAS3 call, we are still dealing with the multiplication of a 5×5 matrix with a

$5 \times 500{,}000$ matrix, a situation for which BLAS3 usually does not perform very well. Third, because we use static loop sizes in the solver (the only drawback being that we need to recompile the solver every time we change the size of the mesh), at compile time the compiler knows the size of all the loops and can therefore very efficiently optimize them (using unrolling for instance). Because the inner loops are very small (of size N+1 = 5), it is very difficult to do better than loop unrolling performed automatically by the compiler. Therefore it is better in terms of performance to let the compiler optimize the static loops rather than to switch to BLAS3.

One way of improving performance is to manually use the Altivec/VMX vector unit of the PowerPC, which can handle four single-precision floating-point operations in a vector and is therefore well suited for our small matrix products since we can load a vector unit with 4 floats, perform several 'multiply-and-add' (vec_MADD) operations to compute the matrix-matrix product, and store the results in four consecutive elements of the result matrix. Since our matrices are of size 5×5 and not 4×4, we use vector instructions for 4 out of each set of 5 values and compute the last one serially in regular Fortran. To improve performance and get correct results we align our 3D blocks of $5 \times 5 \times 5 = 125$ floats on 128 in memory using padding with three dummy values, which induces a negligible waste of memory of 128 / 125 = 2.4% (non aligned accesses lead to incorrect results in Altivec). We typically gain between 15% and 20% in CPU time with respect to version 4.0 without Altivec.

3.5 MPI Implementation and Load Balancing

Our SEM solver is based upon a pure MPI implementation. A few years ago on the Japanese Earth Simulator we implemented a mixed MPI – OpenMP solution, using MPI between nodes (i.e., between blocks of 8 processors with shared memory) and OpenMP inside each node. However, in practice, tests on a small number of processors gave a CPU time that was almost identical to a pure MPI run, and therefore we decided to permanently switch to pure MPI [6]. We do not claim that this conclusion is general; it might well be specific to our SEM algorithm, in particular we did not try the mixed OpenMP – MPI solution on a large number of nodes or on MareNostrum. Other groups have successfully implemented algorithms based upon mixed OpenMP – MPI models on large parallel machines.

I/O is not an issue in our simulations because we only output a small number of time series (called 'seismograms') to record seismic motion (the three components of the displacement vector) at a small number of points at the surface of the mesh. This means that the amount of data saved by our SEM is small.

Figure 4 shows that a regular mesh has the undesirable property that the size of the elements decreases dramatically with depth. To maintain a relatively constant number of grid points per wavelength, element size should increase with depth. In version 3.6 of SPECFEM3D, this was accomplished in three stages based on a simple 'doubling brick'. Each block has four sides that need to match up exactly with four other blocks to complete the cube. Schematically, these four

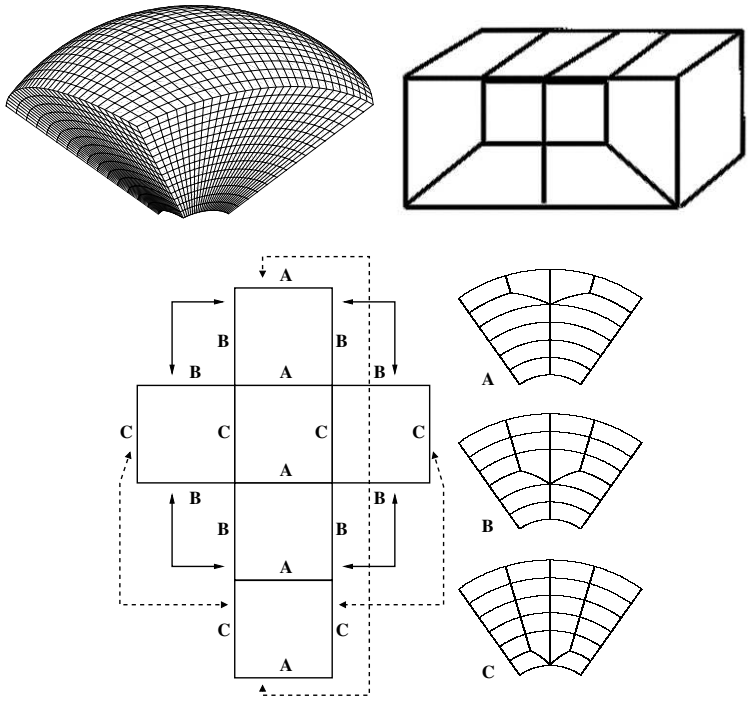

Fig. 4. A regular mesh (top left) has the undesirable property that the size of the elements decreases dramatically with depth. To maintain a relatively constant number of grid points per wave length, element size should increase with depth. In version 3.6 of SPECFEM3D, this is accomplished in three stages based on a simple 'doubling brick' (top right). Each block has four sides that need to match up exactly with four other blocks to complete the cube (bottom), as indicated by the arrows. Schematically, these four sides have one of three designs: A, B, or C, as illustrated on the right. When the six blocks are assembled to make the entire globe, they match perfectly. Unfortunately, the fact that the three types of blocks have a significantly different number of mesh elements induces significant load imbalance.

sides have one of three designs: A, B, or C. When the six blocks are assembled to make the entire globe, they match perfectly. However, because with that simple 'doubling brick' doubling the mesh in two directions in the same layer is topologically impossible, the three mesh types A, B and C contain a significantly (15% to 20%) different number of mesh elements, which in turn results in load imbalance in the same amount because in the SEM one performs the same number of elementary calculations in each element. In addition, an analysis of version 3.6 performed with the ParaVer analysis tool (Figure 5) also showed significant load imbalance in terms of the number of Level 2 (L2) data cache misses in each mesh slice (this will be further discussed below). ParaVer (see e.g. [11] and www.cepba.upc.es/paraver) is a tool developed at the Barcelona Supercomputing Center that is designed to analyze the number of data cache misses and of

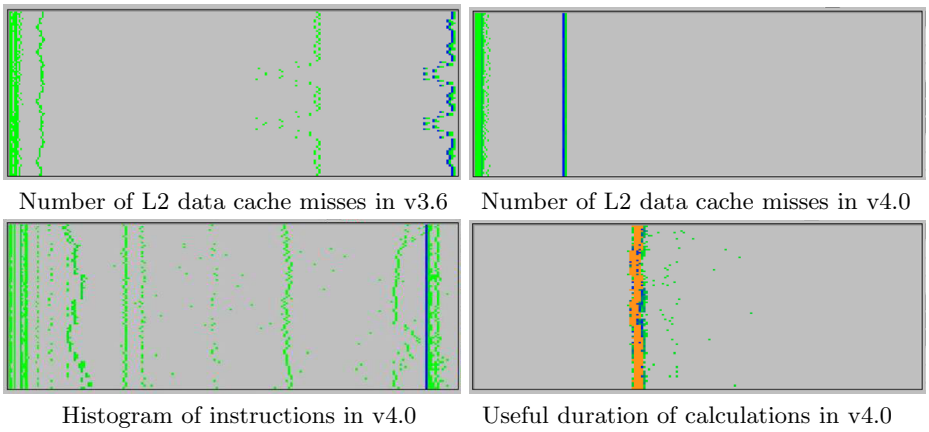

Fig. 5. ParaVer analysis of the code on 96 processors, from processor 1 at the top of each picture to processor 96 at the bottom. Top left: In version 3.6 of SPECFEM3D, L2 cache misses were very poorly balanced between mesh slices (irregular blue and green curve), thus inducing severe load imbalance. In version 4.0 (top right), L2 cache misses (represented on the same horizontal scale) have been drastically reduced and very well balanced (straight blue line). The number of instructions executed is also very well balanced (bottom left, straight blue line). As a result, useful duration of the calculations (bottom right, orange points) is well balanced too.

instructions of MPI processes as well as the useful duration of calculations performed, among many other things. Let us mention that the imbalance observed in terms of cache misses was also observed between slices belonging to the same chunk type (A, B or C) and was therefore mostly due to the numbering of the mesh points (i.e., the order in which they were accessed) and not only to the different mesh structure between different chunks.

In order to address the first issue of geometrical mesh imbalance, in version 4.0 the mesh doubling of Figure 4 is now accomplished in only one level instead of two based on a more efficient geometrical 'doubling brick' which is assembled in a symmetric block of four 'doubling bricks' based on mirror symmetry of the basic brick (Figure 6). This makes it possible to carry out the doubling in both directions in the same layer. As a result, while in the old mesh of version 3.6 there are three types of mesh chunks (labeled A, B and C in Figure 4), in version 4.0 of the code this poor property of the mesh is suppressed and all the mesh chunks have the same shape and exact same number of elements, thus resulting in perfect geometrical mesh balancing. Because the number of operations performed in each element is the same, load balancing is therefore very significantly improved.

In order to have more uniform mesh sampling in the inner core of the Earth, in version 4.0 we also slightly inflate the central cube in order to better balance the mesh angles compared to version 3.6 (Figure 7). When the central cube is not inflated, some elements can have a poor skewness and/or poor aspect ratio in the vicinity of the central cube. Inflating it significantly improves both the

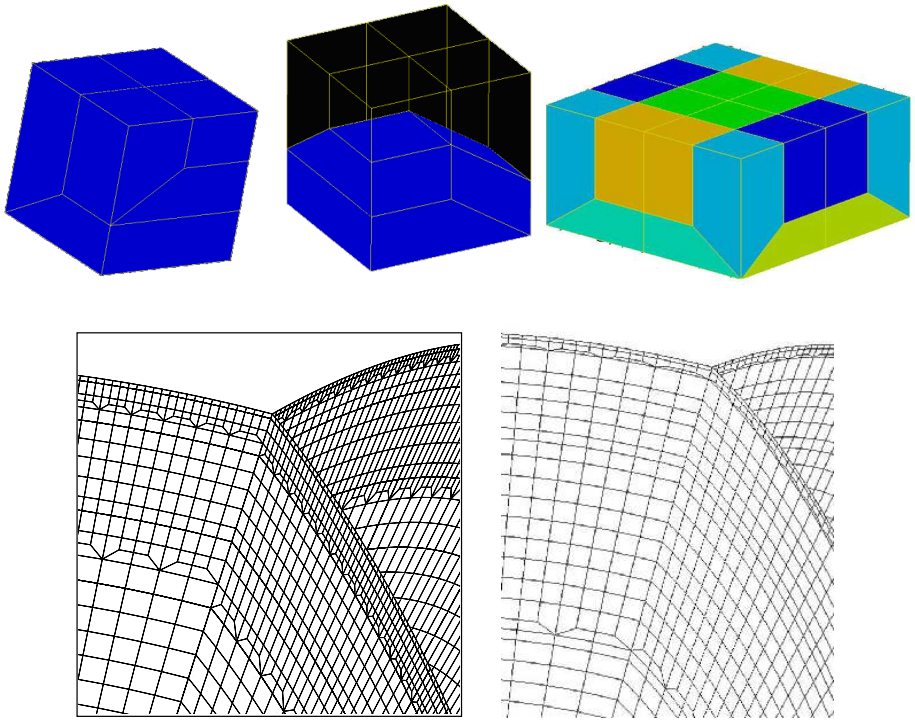

Fig. 6. In version 4.0 of SPECFEM3D, the mesh doubling of Figure 4 is accomplished in only one level instead of two in each mesh chunk and therefore three in the whole sphere, based on a more efficient geometrical 'doubling brick' (top, left and center) which is assembled in a symmetric block of four 'doubling bricks' based on mirror symmetry (top right). As a result, when we zoom on a region of contact between three of the six mesh chunks of Figure 2, we can see that while in the old mesh of version 3.6 (bottom left) there are three types of mesh chunks (labeled A, B and C in Figure 4), in version 4.0 of the code (bottom right) this poor property of the mesh has been suppressed and all the mesh chunks have the same shape and exact same number of elements, thus resulting in perfect geometrical mesh balancing.

skewness and the aspect ratio (both the average value for all the elements and the worst value for the most distorted element).

In the SEM one needs to assemble internal force contributions between neighboring slices, as mentioned above and in Figure 3. The pattern of communications needed to assemble such slices on the edges and corners of the six blocks of the cubed-sphere mesh can be determined from Figure 2 (for instance the valence of most surface points is 4, but it is three at the corners of the six blocks). Because the mass matrix is exactly diagonal, processors spend most of their time performing actual computations, and the amount of communications is comparatively small (in spite of the fact that the number of points to exchange increases approximately as N^2, but the polynomial degree N is always chosen

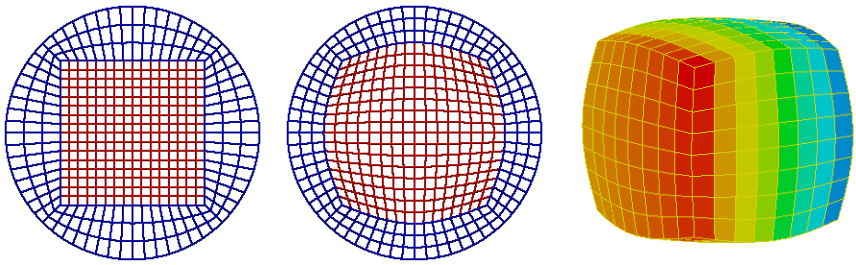

Fig. 7. In order to have more uniform mesh sampling in the inner core of the Earth, in version 4.0 of SPECFEM (middle) we slightly inflate the central cube in order to better balance the mesh angles compared to version 3.6 (left), as illustrated here in a 2D cut plane. In 3D this results in the inflated cube represented on the right.

small, between 4 and 7 in practice, see e.g. [3,8]). We thought about switching from the blocking MPI implementation used in version 3.6 to a non-blocking implementation in order to overlap the communications with calculations. This would imply first looping on the elements that are located on the edges of the mesh slices, computing their contributions, starting non-blocking SENDs of their contributions, and computing the rest of the elements inside the slices while the communications are being performed (see e.g. [12]). We implemented this strategy in a 2D version of our code (for simplicity) but did not notice any significant gain in terms of performance because the overall cost of communications is very small ($< 5\%$) compared to CPU time. We therefore concluded that there was no real need to switch to non-blocking MPI in the 3D version of the code.

3.6 Performance and Scaling Results

Let us perform an analysis of the improved code on 96 processors using ParaVer. Figure 5 shows that in the initial version 3.6, the number of L2 cache misses was very different between mesh slices, thus inducing severe load imbalance. In the improved version 4.0, L2 cache misses have been drastically reduced and very well balanced. The number of instructions executed is also very well balanced. As a result, useful duration of the calculations is well balanced too. In total, we gain a huge factor of 3.3 in terms of wall-clock time between both versions. This shows that the IBM PowerPC 970 is very sensitive to cache misses because the same run performed on an Intel Itanium and also on an AMD Opteron cluster shows a factor of 'only' 1.55 to 1.60.

Let us now analyze scaling by measuring wall-clock time per time step (averaged over 700 time steps in order for the measurement to be reliable) for a medium-size run performed on 24, 54, 96 and 216 processors (we also tried to run the test on 6 processors but it was too big to fit in memory). In Figure 8 we compare the scaling curve to the theoretical curve corresponding to perfect linear scaling, which we compute using the time measured on 96 processors and scaling it by the ratio of the number of processors used. The conclusion

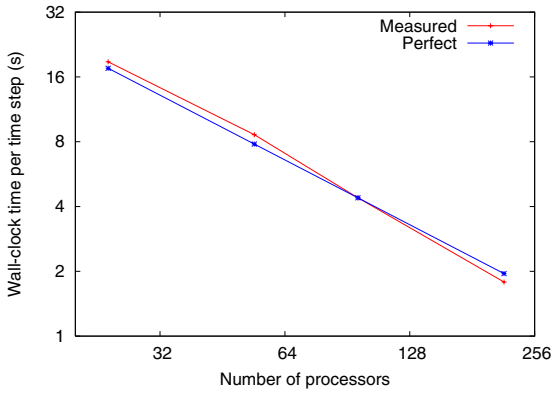

Fig. 8. Scaling in logarithmic scale measured (in red) for the same run performed on 24, 54, 96 and 216 processors, compared to perfect linear scaling (in blue) computed using the run on 96 processors as a reference. The two curves are very similar.

is that the scaling of the code is excellent. We therefore conclude that we are now ready to run the code for a real application on a very large number of processors of MareNostrum. MareNostrum has 2560 two-biprocessor blades, for a total of 10240 processor cores. Each blade has 8 gigabytes of memory, for a total of 20480 gigabytes of memory. For the final high-resolution run, we used 2166 processor cores and computed 50600 time steps of the explicit time integration scheme of the SEM algorithm. Total memory used was 2.5 terabytes. The code performed well and performance levels obtained were very satisfactory, the whole run took slightly less than 60 hours of wall-clock time (being the only user running on the corresponding dedicated blades). The geophysical analysis of the seismograms is currently under way.

4 Conclusions

MareNostrum has allowed us to reach unprecedented resolution for the simulation of seismic wave propagation resulting from an earthquake in the 3D inner core of the Earth using a spectral-element method implemented based upon MPI. A combination of better mesh design and improved point numbering has allowed us to balance the number of instructions very well, drastically reduce the number of L2 cache misses and also balance them very well, and as a result reach very good balancing in terms of the useful duration of the calculations in each mesh slice. BLAS3 or non-blocking MPI have not been required to achieve this, but using Altivec vector instructions such as multiply-and-add has allowed us to gain 20% in terms of CPU time.

Acknowledgments. The authors thank three anonymous reviewers for comments that improved the manuscript, and Jean Roman, Rogeli Grima, Nicolas

Le Goff, Roland Martin, Jean-Paul Ampuero and Jérémie Gaidamour for fruitful discussion. The idea of computing PKP seismic phases came from discussions with Sébastien Chevrot. The help of Sergi Girona, David Vicente, Judit Giménez and the BSC support group was invaluable to perform the calculations. This material is based in part upon research supported by HPC-Europa, European FP6 MIRG-CT-2005-017461 and French ANR-05-CIGC-002 NUMASIS.

References

1. Komatitsch, D., Vilotte, J.P.: The spectral-element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures. Bull. Seismol. Soc. Am. 88(2), 368–392 (1998)
2. Seriani, G.: 3-D large-scale wave propagation modeling by a spectral element method on a Cray T3E multiprocessor. Comput. Methods Appl. Mech. Engrg. 164, 235–247 (1998)
3. Komatitsch, D., Tromp, J.: Introduction to the spectral-element method for 3-D seismic wave propagation. Geophys. J. Int. 139(3), 806–822 (1999)
4. Komatitsch, D., Tromp, J.: Spectral-element simulations of global seismic wave propagation-I. Validation. Geophys. J. Int. 149(2), 390–412 (2002)
5. Chaljub, E., Capdeville, Y., Vilotte, J.P.: Solving elastodynamics in a fluid-solid heterogeneous sphere: a parallel spectral-element approximation on non-conforming grids. J. Comput. Phys. 187(2), 457–491 (2003)
6. Komatitsch, D., Tsuboi, S., Ji, C., Tromp, J.: A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator. In: Proceedings of the ACM/IEEE Supercomputing SC 2003 conference, CD-ROM (2003), `www.sc-conference.org/sc2003`
7. Hughes, T.J.R.: The finite element method, linear static and dynamic finite element analysis. Prentice-Hall International, Englewood Cliffs (1987)
8. De Basabe, J.D., Sen, M.K.: Grid dispersion and stability criteria of some common finite-element methods for acoustic and elastic wave equations. Geophysics 72(6), T81–T95 (2007)
9. Sadourny, R.: Conservative finite-difference approximations of the primitive equations on quasi-uniform spherical grids. Monthly Weather Review 100, 136–144 (1972)
10. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 24th National ACM Conference, pp. 157–172. ACM Press, New York (1969)
11. Jost, G., Jin, H., Labarta, J., Giménez, J., Caubet, J.: Performance analysis of multilevel parallel applications on shared memory architectures. In: Proceedings of the IPDPS 2003 International Parallel and Distributed Processing Symposium, Nice, France (April 2003)
12. Danielson, K.T., Namburu, R.R.: Nonlinear dynamic finite element analysis on parallel computers using Fortran90 and MPI. Advances in Engineering Software 29(3-6), 179–186 (1998)