Geophysical Journal International

Geophys. J. Int. (2015) **203,** 1419–1435 doi: 10.1093/gji/ggv380
GJI Seismology

A nodal discontinuous Galerkin method for reverse-time migration on GPU clusters

A. Modave,[1] A. St-Cyr,[2] W.A. Mulder[2,3] and T. Warburton[1]

[1]*Rice University, Houston, Texas, USA. E-mail: modave@rice.edu*
[2]*Shell Global Solutions International B.V., Rijswijk, The Netherlands*
[3]*Delft University of Technology, Delft, The Netherlands*

Accepted 2015 September 8. Received 2015 August 26; in original form 2015 June 8

SUMMARY

Improving both accuracy and computational performance of numerical tools is a major challenge for seismic imaging and generally requires specialized implementations to make full use of modern parallel architectures. We present a computational strategy for reverse-time migration (RTM) with accelerator-aided clusters. A new imaging condition computed from the pressure and velocity fields is introduced. The model solver is based on a high-order discontinuous Galerkin time-domain (DGTD) method for the pressure–velocity system with unstructured meshes and multirate local time stepping. We adopted the MPI+X approach for distributed programming where X is a threaded programming model. In this work we chose OCCA, a unified framework that makes use of major multithreading languages (e.g. CUDA and OpenCL) and offers the flexibility to run on several hardware architectures. DGTD schemes are suitable for efficient computations with accelerators thanks to localized element-to-element coupling and the dense algebraic operations required for each element. Moreover, compared to high-order finite-difference schemes, the thin halo inherent to DGTD method reduces the amount of data to be exchanged between MPI processes and storage requirements for RTM procedures. The amount of data to be recorded during simulation is reduced by storing only boundary values in memory rather than on disk and recreating the forward wavefields. Computational results are presented that indicate that these methods are strong scalable up to at least 32 GPUs for a three-dimensional RTM case.

Key words: Image processing; Numerical solutions; Computational seismology.

1 INTRODUCTION

Reverse-time migration (RTM), introduced in the early 1980s (Baysal *et al.* 1983; Lailly 1983; Loewenthal 1983; McMechan 1983; Whitmore 1983; Tarantola 1984), is a migration method based on the wave equation. Although not the only means of obtaining images of the subsurface from seismic data, it is currently widely used in the oil and gas industry (Etgen & Michelena 2010). RTM constructs a subsurface image by correlating source wavefields with time-reversed receiver wavefields (e.g. Claerbout 1985; Bleistein *et al.* 2001). However, the procedure is very demanding in terms of computational and memory resources. It can be implemented as a single-step procedure or as an iterative scheme, minimizing the difference between observed primary reflections and synthetic data modeled by the Born approximation of the wave equation. At each iteration, the solutions of two numerical simulations have to be computed and cross-correlated: a forward simulation to determine the source wavefields in a background model that is reflection-free in the seismic bandwidth and a reverse-time wavefield generated by the data or data residuals at the receivers. In typical industrial applications, both simulations are expensive and must be performed on clusters with many-core CPUs and, optionally, with accelerators such as GPUs to further boost the performance. It is therefore critical to develop numerical methods that are both efficient and scalable on such clusters. Also, strategies have to be found to compute the RTM cross-correlation without penalizing the final run time.

Nowadays, finite-difference methods are the most widely used numerical schemes. A large literature is available, notably to reach high-order convergence rates and to reduce dispersion and dissipation errors (see e.g. Virieux *et al.* 2011, for a review). Alternatively, several kinds of finite-element methods have been proposed, such as spectral finite-element methods (Komatitsch & Vilotte 1998; Komatitsch & Tromp 1999), continuous mass-lumped finite-element methods (Chin-Joe-Kong *et al.* 1999; Cohen *et al.* 2001) or discontinuous Galerkin (DG) methods (Dumbser & Käser 2006; de la Puente *et al.* 2007; Collis *et al.* 2010; Etienne *et al.* 2010; Krebs *et al.* 2014; Mercerat & Glinsky 2015). When using unstructured meshes, finite-element methods can easily handle geometric or physical discontinuities without suffering from the loss of accuracy that afflicts most finite-difference schemes. In contrast to the standard finite-element approach, the mentioned methods do not require the solution of large sparse linear systems of equations, but if a lumping procedure or a block diagonal mass matrix is chosen, it is possible to use an explicit time-stepping scheme. In comparative work (Moczo *et al.* 2011; Baldassari *et al.* 2012; Minisini *et al.* 2012; Zhebel *et al.* 2014), they generally exhibit similar performance results and they surpass classical finite-difference schemes in cases with geometric or physical discontinuities. However, in three dimensions, spectral elements are currently only available for hexahedral meshes, which are less flexible than tetrahedral meshes. Continuous mass-lumped finite elements are available for tetrahedrons only up to third-degree polynomial basis functions, enabling at most a fourth-order spatial convergence of the scheme. The DG approach does not suffer from these limitations, but it is generally considered as expensive because the unknowns are duplicated at the interfaces between elements. This excessively large number of degrees of freedom can be offset by some advantages of the approach. In contrast with other methods, DG schemes are easily combined with local time-stepping strategies in order to improve computational efficiency and speed up of geophysical wave propagation (Dumbser & Käser 2009; Baldassari *et al.* 2011; Minisini *et al.* 2013). Moreover, hybrid discretizations can be deployed thanks to the flexibility in mixing several kinds of elements and different discretization orders (Kirby *et al.* 2000; Dumbser & Käser 2009; Etienne *et al.* 2010; Chan *et al.* 2015). Finally, the local element-to-element coupling and the dense algebraic operations required per element make DG methods suitable for parallel multithreading computations, especially with GPUs (Klöckner *et al.* 2009).

Accelerator-aided clusters require specialized algorithms and computational implementations to make full use of the hardware. In the computational geophysics community, several implementations have been proposed for the numerical modeling of seismic wave propagation problems with clusters of GPUs. They are based on finite-difference schemes (Michéa & Komatitsch 2010; Weiss & Shragge 2013), a spectral element scheme (Komatitsch *et al.* 2010) or a DG method (Mu *et al.* 2013). To our knowledge, only a few papers have been published for complete RTM procedures on GPUs, and only with finite-difference schemes (Abdelkhalek *et al.* 2012; Liu *et al.* 2012, 2013; Yang *et al.* 2014; Knibbe *et al.* 2014). On the other hand, Klöckner *et al.* (2009) have proposed GPU implementations for wave-like problems written with time-domain first-order wave systems and discretized with nodal DG schemes. This implementation has successfully been adapted for several applications (Gödel *et al.* 2010; Fuhry *et al.* 2014; Gandham *et al.* 2015; Modave *et al.* 2015), but does not yet address RTM imaging.

A supplementary difficulty for RTM procedures is to compute the cross-correlation of the solutions of source and receiver simulations. Because those simulations are performed in opposite time directions, it is common to either store the source wavefield at check points or to reproduce it backward in time, in order to compute the cross-correlation during the reverse-time simulation for the receiver wavefield. The source wavefield can be reproduced by using a check-pointing strategy or by saving boundary values during a first forward run (Symes 2007; Dussaud *et al.* 2008). However, in both cases, a substantial storage space is required as well as memory transfers during the runs, which can increase the total run time. Compression strategies can help to reduce these costs (e.g. Aguilar *et al.* 2013; Sun & Fu 2013). As an alternative, a strategy based on random boundaries can overcome this bottleneck, but the final image can be polluted by noise if the coverage of the survey is not dense enough (Clapp 2009; Liu *et al.* 2012). Several nonreconstructive imaging methods have been recently proposed with alternative imaging conditions (Nguyen & McMechan 2015).

In this paper, we propose a novel implementation, named *'RiDG'*, for RTM procedures with clusters of GPUs. It is based on a high-order nodal DG scheme for first-order wave systems (Hesthaven & Warburton 2002, 2007), unstructured meshes made of tetrahedras and a multirate Adams–Bashforth (MRAB) time stepping (Gödel *et al.* 2010). We describe and test the implementation for the pressure–velocity system with a novel imaging condition based on the knowledge of both pressure and velocity, which improves final RTM images in comparison with the classical condition. RiDG is coded in the C++ language, OCCA for multithreaded programming and MPI for parallel computing with distributed memory. OCCA (Medina *et al.* 2014) is a unified programming framework that abstracts major multithreading languages (OpenCL, CUDA, pThreads and OpenMP), offering flexibility to choose the hardware architecture and thread-model at run-time. The strategies of Klöckner *et al.* (2009) for GPU computing have been followed to optimize performances. We have particularly taken care of memory transfers and memory storage that are required for a complete RTM procedure. Despite those transfers, RiDG reaches an excellent strong scalability for a three-dimensional case with 32 GPUs.

This paper is organized as follows. In Section 2, the mathematical wave model and the novel imaging condition are introduced. Section 3 is dedicated to numerical schemes. The high-order nodal DG scheme, the MRAB time stepping and the numerical approximation of the imaging condition are detailed. In Section 4, the computational implementation with OCCA and MPI is described. In particular, the OCCA programming framework is introduced, and the management of computational work and memory transfer is discussed. Finally, computational results for three-dimensional benchmarks are provided in Section 5.

2 WAVE MODEL, IMAGING CONDITIONS AND RTM PROCEDURE

The RTM implementation described in this paper is based on the acoustic wave model formulated in terms of pressure $p(\mathbf{x}, t)$ and particle velocity $\mathbf{v}(\mathbf{x}, t)$,

$$\frac{\partial p}{\partial t} + \rho c^2\, \nabla \cdot \mathbf{v} = f(t)\, \delta(\mathbf{x} - \mathbf{x}_0), \tag{1}$$

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \nabla p = 0. \tag{2}$$

In this work, both the density ρ and phase velocity c are assumed to be positive and piecewise constant. In the source term, $f(t)$ is understood as a signal and $\delta(\mathbf{x} - \mathbf{x}_0)$ is the Dirac delta function. In the complete mathematical model, transparent boundaries are approximated with the boundary condition

$$p - \rho c\, \mathbf{n} \cdot \mathbf{v} = 0. \tag{3}$$

Note that outgoing waves with oblique incidence are partially reflected with the boundary condition (3), which is exact only for normal waves. Techniques such as perfectly matched layers (e.g. Komatitsch & Martin 2007; Meza-Fajardo & Papageorgiou 2008), sponge layers (e.g. Métivier *et al.* 2014; Tago *et al.* 2014) and high-order boundary conditions (e.g. Hagstrom & Warburton 2009; Baffet *et al.* 2014) can improve the solution close to the boundary. The use of these elaborate techniques is, however, out of the scope of this paper, and only the simple condition (3) is used in the remainder.

For each set of source and receiver signals, the RTM procedure requires the numerical solutions of two wave propagation problems using an initial model of the subsurface material parameters. The '*source*' solution is obtained by emitting the source signal forward-in-time and the '*receivers*' solution is computed by re-injecting backward-in-time the signals recorded at receivers. The key step of the method then consists in computing an imaging condition that correlates both solutions. The classical imaging condition (Claerbout 1985; Bleistein *et al.* 2001) is

$$I(\mathbf{x}) = \sum_{\text{shots}} \int_0^{t_f} p_S(\mathbf{x}, t)\, p_R(\mathbf{x}, t)\, dt, \tag{4}$$

where $p_S(\mathbf{x}, t)$ and $p_R(\mathbf{x}, t)$ are the pressure fields of the source and receivers wavefields for each shot, respectively and t_f is the final time.

Note that eq. (4) is expressed in its simplest form, without true-amplitude migration weights, as we want to focus on the computational rather than on the geophysical issues. High values of $I(\mathbf{x})$ correspond to geophysical features of interest, generally called reflectors.

We use here an alternative definition for the imaging condition, based on Claerbout's original interpretation and involving both pressure and velocity wavefields. In his foundation paper, Claerbout (1971) introduced the imaging condition as the cross-correlation of '*downgoing waves*' generated by the source and '*upgoing waves*' that reach the receivers. Formula (4) corresponds to this definition if no reflections occur in the underground. This assumption is valid when using a one-way wave equation, but the resulting imaging procedure suffers from dip limitations (McMechan 1983) because only waves propagating in certain directions are simulated. Two-way wave models, such as the pressure–velocity system (1) and (2) or the wave equation, do not suffer from dip limitations. However, they are valid only if the subsurface model is very smooth or has a constant impedance. Otherwise, a well-known disadvantage is the correlation of forward and time-reversed wavefields following the same path, generating long-wave artefacts in the image. Typical examples are diving waves and strong refractions. One strategy to overcome this difficulty consists in decomposing the source and receivers wavefields into upgoing and downgoing components and only keeping the components of interest in the imaging condition (e.g. Hu & McMechan 1987; Liu *et al.* 2011). In our case, it is easy to perform such a decomposition thanks to the characteristic fields of the first-order wave system (1) and (2). Recall that, in one dimension with z as the spatial coordinate, the system is equivalent to

$$\frac{\partial}{\partial t}(p + c\rho\, v_z) + c\frac{\partial}{\partial z}(p + c\rho\, v_z) = 0, \tag{5}$$

$$\frac{\partial}{\partial t}(p - c\rho\, v_z) - c\frac{\partial}{\partial z}(p - c\rho\, v_z) = 0, \tag{6}$$

where v_z is the vertical velocity. These equations correspond to one-way wave-equation models, where waves propagate upwards and downwards, respectively. By analogy with the previous works, we propose the imaging condition

$$I(\mathbf{x}) = \sum_{\text{shots}} \int_0^{t_f} q_S^-(\mathbf{x}, t)\, q_R^+(\mathbf{x}, t)\, dt, \tag{7}$$

where q^+ and q^- correspond to the upgoing and downgoing characteristic fields, defined as

$$q^{\pm}(\mathbf{x}, t) = p(\mathbf{x}, t) \pm c\rho\, \mathbf{e}_z \cdot \mathbf{v}(\mathbf{x}, t), \tag{8}$$

with the vertical unit vector $\mathbf{e}_z$, pointing upwards.

By using characteristic fields, the new imaging condition (7) takes into account upgoing information of the source wavefield and downgoing information of the receiver wavefield. Since both pressure p and velocity $\mathbf{v}$ are available with our first-order formulation, the use of the characteristic-based condition (7) instead of the classical one (4) imposes hardly any additional cost. When using the second-order wave equation, like in most RTM implementations, only the pressure is known and the velocity would be computed a posteriori. Let us mention that, for steep dips, the above can be easily generalized along the lines of Liu *et al.* (2011) by also considering decompositions in any direction. A complete study of imaging conditions based on characteristic variables is out of the scope of this paper and will be the subject of future work.

Approximating imaging conditions (4) and (7) with classical numerical integration requires the knowledge of both source and receivers solutions at each time step. As mentioned in the Introduction, this represents a difficulty because these solutions are simulated with opposite time directions. Following Dussaud *et al.* (2008), we perform both source and receivers simulations backward-in-time, together with a step-by-step computation of the imaging condition. The backward-in-time source simulation is made possible thanks to a preliminary forward-in-time run, where both the boundary values of fields and the final solution are saved to reproduce the solution in the other (reverse) time direction.

With a nodal DG scheme, we need to save the pressure p and the normal component of the velocity $\mathbf{n} \cdot \mathbf{u}$ (called traces of fields in the remainder) only at the boundary nodes, even with high-degree basis functions. By contrast, when using this strategy with a high-order finite-difference scheme, a thick enough halo of data must be saved to compute the thick stencil of the scheme at the boundary (see e.g. Yang *et al.* 2014). In three dimensions, the amount of data at boundaries increases linearly according to the stencil radius for finite-difference schemes whereas the halo zone for a nodal DG scheme is just one node deep. The thin halo inherent to the nodal DG method then allows for a storage requirement lower than with the widely used high-order finite-difference schemes.

3 NUMERICAL SCHEMES

In this section, we detail the numerical schemes used to solve for the source and receiver wavefields and to evaluate the imaging condition $I(\mathbf{x})$ from their correlation. The spatial discretization is based on a nodal DG scheme. The time stepping is performed through a multirate scheme with local time stepping where needed. These are detailed in Sections 3.1 and 3.2, respectively. The numerical evaluation of $I(\mathbf{x})$ is addressed in Section 3.3.

3.1 Nodal discontinuous Galerkin scheme

For each problem, the approximate fields are built on a spatial mesh of the computational domain $\Omega \subset \mathbb{R}^3$, made of non-overlapping tetrahedral cells,

$$\Omega = \bigcup_{k=1,\ldots,K} \mathsf{D}_k,$$

where K is the number of cells and D_k is the kth cell. With the nodal DG method, the pressure field and each Cartesian component of the velocity field are approached by piecewise polynomial functions. The discrete unknowns correspond to the values of fields at nodes of elements (Hesthaven & Warburton 2002, 2007). Over each cell

D_k, the approximate fields are then equal to

$$p_k(\mathbf{x},t) = \sum_{n=1}^{N_p} p_{k,n}(t)\,\ell_{k,n}(\mathbf{x}), \quad \forall \mathbf{x} \in \mathsf{D}_k, \tag{9}$$

$$v_{i,k}(\mathbf{x},t) = \sum_{n=1}^{N_p} v_{i,k,n}(t)\,\ell_{k,n}(\mathbf{x}), \quad \forall \mathbf{x} \in \mathsf{D}_k, \quad i = 1,2,3, \tag{10}$$

where N_p is the number of nodes per element, $p_{k,n}(t)$ and $v_{i,k,n}(t)$ are the values of fields at node n of element k and $\ell_{k,n}(\mathbf{x})$ is the associated multivariate Lagrange polynomial function. Denoting the position of the node with $\mathbf{x}_{k,n}$, we have

$$\ell_{k,n}(\mathbf{x}_{k,m}) = \begin{cases} 1, & \text{if } m = n, \\ 0, & \text{if } m \neq n, \end{cases}$$

for all $m, n = 1, \ldots, N_p$. In this work, the position of nodes is defined using the warp-and-blend technique (Warburton 2006). The number of nodes for each element is given by $N_p = (N+1)(N+2)(N+3)/6$, where N is the maximal order of polynomial functions.

The spatial discretization is obtained from the variational formulation of the problem. For each cell D_k, we are looking for $p(\mathbf{x},t) \in L_2(\mathsf{D}_k, \mathbb{R}^+)$ and $\mathbf{v}(\mathbf{x},t) \in \mathbf{L}_2(\mathsf{D}_k, \mathbb{R}^+)$ such that

$$\int_{\mathsf{D}_k} \left(\frac{1}{\rho c^2}\frac{\partial p}{\partial t} + \nabla \cdot \mathbf{v} \right) \psi(\mathbf{x})\, d\mathbf{x} + \int_{\partial \mathsf{D}_k} \mathbf{n} \cdot (\mathbf{v}^\star - \mathbf{v}^-)\, \psi(\mathbf{x})\, d\mathbf{x} = 0 \tag{11}$$

$$\int_{\mathsf{D}_k} \left(\rho \frac{\partial \mathbf{v}}{\partial t} + \nabla p \right) \cdot \boldsymbol{\psi}(\mathbf{x})\, d\mathbf{x} + \int_{\partial \mathsf{D}_k} (p^\star - p^-)\, \mathbf{n} \cdot \boldsymbol{\psi}(\mathbf{x})\, d\mathbf{x} = 0, \tag{12}$$

for all test functions $\psi(\mathbf{x}) \in L_2(\mathsf{D}^k)$ and $\boldsymbol{\psi}(\mathbf{x}) \in \mathbf{L}_2(\mathsf{D}^k)$, where $\mathbf{n}$ is the outward unit normal to the cell boundary $\partial\mathsf{D}_k$. In the boundary terms, we use upwind numerical fluxes provided by an exact Riemann solver (e.g. LeVeque 2002; Wilcox *et al.* 2010),

$$\mathbf{n} \cdot (\mathbf{v}^\star - \mathbf{v}^-) = \frac{1}{\rho^+ c^+ + \rho^- c^-} \left(\rho^+ c^+ \mathbf{n} \cdot [\![\mathbf{v}]\!] - [\![p]\!] \right),$$
$$p^\star - p^- = \frac{\rho^+ c^+ \rho^- c^-}{\rho^+ c^+ + \rho^- c^-} \left(\frac{1}{\rho^+ c^+}[\![p]\!] - \mathbf{n} \cdot [\![\mathbf{v}]\!] \right),$$

where $[\![p]\!] = p^+ - p^-$ and $[\![\mathbf{v}]\!] = \mathbf{v}^+ - \mathbf{v}^-$ are the jumps of fields at the interface. The plus and minus subscripts denote nodal values on the side of the neighboring and current cells, respectively.

The semi-discrete equations are obtained by injecting the approximate representations of fields (9) and (10) in eqs (11) and (12), and using the Lagrange polynomial functions as test functions. For each element k, a system of equations can then be written as

$$\frac{d\mathbf{q}_k^p}{dt} = -\rho_k c_k^2 \sum_{i=1}^{3}\sum_{j=1}^{3} g_{k,i,j}^{\mathrm{vol}} \mathbf{D}_j \mathbf{q}_k^{v_i} - \sum_{f=1}^{N_f} g_{k,f}^{\mathrm{sur}} \mathbf{L}_f \mathbf{p}_{k,f}^p, \tag{13}$$

$$\frac{d\mathbf{q}_k^{v_i}}{dt} = -\frac{1}{\rho_k} \sum_{j=1}^{3} g_{k,i,j}^{\mathrm{vol}} \mathbf{D}_j \mathbf{q}_k^p - \sum_{f=1}^{N_f} g_{k,f}^{\mathrm{sur}} \mathbf{L}_f \mathbf{p}_{k,f}^{v_i}, \tag{14}$$

where the vectors $\mathbf{q}_k^p$ and $\mathbf{q}_k^{v_j}$ contain the discrete unknowns associated to each field for all nodes of element k and the vectors $\mathbf{p}_{k,f}^p$ and $\mathbf{p}_{k,f}^{v_j}$ contain the penalty terms for all face nodes of face f. The first terms of the right-hand sides of these equations correspond to the volume integrals with spatial differential operators of the variational form, while the second terms correspond to the surface integrals. The matrices $\mathbf{D}_j$ and $\mathbf{L}_f$ are, respectively, the differentiation matrices and the lifting matrices of the reference element. The geometric factors $g_{k,i,j}^{\mathrm{vol}}$ and $g_{k,f}^{\mathrm{sur}}$ depend on the shape of each element. The definitions of the matrices and the factors and a complete derivation of the semi-discrete equations are given in Appendix A. A point source term is incorporated into the formulation by adding

$$\int_{\mathsf{D}_k} f(t)\, \tilde{\delta}(\mathbf{x} - \mathbf{x}_0)\, \psi(\mathbf{x})\, d\mathbf{x}, \tag{15}$$

to the right-hand-side of eq. (11). A corresponding term will enter eq. (13). In the term (15), the Dirac delta function is replaced by its L2-projection on the polynomial basis, $\tilde{\delta}(\mathbf{x} - \mathbf{x}_0) = \sum_n w_n \ell_{k,n}(\mathbf{x})$, where the coefficients w_n are obtained by solving the system

$$\int_{D_k} \ell_{k,m}(\mathbf{x}) \left[\delta(\mathbf{x} - \mathbf{x}_0) - \sum_{n=1}^{N_p} w_n \ell_{k,n}(\mathbf{x}) \right] d\mathbf{x} = 0, \quad \text{for } m = 1, \ldots, N_p.$$

For sake of clarity, we finally rewrite the semi-discrete equations as the abstract system

$$\frac{d\mathbf{q}_k}{dt} = \mathbf{r}_k, \tag{16}$$

where $\mathbf{q}_k$ is the vector of all discrete unknowns for element k and $\mathbf{r}_k$ combines the right-hand side terms of eqs (13) and (14). The global semi-discrete scheme is simply built by combining the local systems of all elements.

3.2 Multirate time-stepping scheme

The resolution in time is performed by using an explicit multirate scheme based on the third-order Adams–Bashforth method. Such schemes are advantageous in comparison to single-rate ones for problems with refined meshes or involving multiscale dynamics. The stability of explicit single-rate schemes is guaranteed by the global CFL condition

$$\Delta t \leq C \min_k \left\{ \frac{h_k}{c_k} \right\},$$

where C is a constant that depends on the time integration scheme and the spatial approximation, h_k is the characteristic size of element k and c_k is the corresponding phase velocity. The maximum stable time step Δt is thus determined by the element having the smallest ratio h_k/c_k. With multirate schemes, the update of approximate fields on each element is performed with a time step Δt_k that is local to the element, and the stability condition becomes local,

$$\Delta t_k \leq C \frac{h_k}{c_k}. \tag{17}$$

Larger local time steps are then allowed for elements having a large ratio h_k/c_k, thus providing a reduction of the computational cost of the global scheme.

The MRAB scheme is efficiently implemented by grouping the elements into levels with fixed time steps (e.g. Gödel *et al.* 2010; Gandham *et al.* 2015). For each element k of level l, the local time step is given by

$$\Delta t_l = 2^{N_{\mathrm{levels}} - l} \Delta t_{\mathrm{local}},$$

where N_{levels} is the number of levels and $\Delta t_{\mathrm{local}}$ is the smallest considered time step. The level of each element depends on the maximum local time step allowed by condition (17). After a first lumping of elements based on this condition, some of them are moved from coarse levels (with larger time steps) to finer levels (with small time steps) in order to have no more than one level of difference between two neighboring elements of the mesh. Therefore, the local time steps of two neighboring elements are the same or differ with a factor 2. Let us denote Δt_{global} the global time step, which corresponds to the local time step of the coarsest level (i.e. the level 1). For each level l, $N_l = 2^{l-1}$ local iterations are needed to achieve a global iteration.

At each local iteration, the discrete unknowns of elements are updated according to the classical third-order Adams–Bashforth formula,

$$\mathbf{q}_k^{n+m/N_l} = \mathbf{q}_k^{n+(m-1)/N_l} + \Delta t_l \sum_{s=1}^{3} a_s\, \mathbf{r}_k^{n+(m-s)/N_l}, \tag{18}$$

with $a_1 = 23/12$, $a_2 = -16/12$ and $a_3 = 5/12$, and where the indices n and m correspond to the global and local time steppings, respectively. This formula requires the knowledge of the right-hand side vector $\mathbf{r}_k$ at the three previous local steps. For each step, this vector is thus computed with the values of unknowns local to the element, as well as those of the neighboring element at each interface, according to eqs (13) and (14). However, at the interface between two elements belonging to two different levels, interface unknowns of the coarse-level element are available at only every two local steps of the fine-level element. For the intermediate step, the unknowns of the coarse-level element evaluated according to the modified Adams–Bashforth formula

$$\mathbf{q}_k^{n+(m-\frac{1}{2})/N_l} = \mathbf{q}_k^{n+(m-1)/N_l} + \Delta t_l \sum_{s=1}^{3} b_f\, \mathbf{r}_k^{n+(m-s)/N_l}, \tag{19}$$

with the coefficients $b_1 = 17/24$, $b_2 = -7/24$ and $b_3 = 2/24$.

This procedure is straightforwardly adapted for backward-in-time simulations. It suffices to consider that time decreases when the indexes n and m increase. No changes are then required in the update schemes, but the penalty parameters τ_p and $\tau_{\mathbf{v}}$ now must be strictly negative to stabilize this backward-in-time scheme.

3.3 Evaluation of imaging condition

We will now focus on the computation of the image $I(\mathbf{x})$, defined by eq. (4). The generalization to the imaging condition (7) is straightforward. For one sample of source–receivers signals, the time integral of $I(\mathbf{x})$ is evaluated time step by time step along a backward-in-time computation of both source solution $p_{\text{S}}(\mathbf{x}, t)$ and receivers solution $p_{\text{R}}(\mathbf{x}, t)$, as explained in Section 2. Each local time-step contribution to the time integral,

$$\int_{t^{n+(m-1)/N_l}}^{t^{n+m/N_l}} p_{\text{S}}(\mathbf{x}, t)\, p_{\text{R}}(\mathbf{x}, t)\, dt, \tag{20}$$

with $t^{n+m/N_l} = (n + m/N_l)\Delta t$, is computed at each node $\mathbf{x}$ and the values are accumulated over the steps to provide $I(\mathbf{x})$.

Integral (20) could typically be evaluated using the trapezoidal rule

$$\frac{\Delta t_l}{2}\left(p_{\text{S}}^{n+m/N_l}(\mathbf{x})\, p_{\text{R}}^{n+m/N_l}(\mathbf{x}) - p_{\text{S}}^{n+(m-1)/N_l}(\mathbf{x})\, p_{\text{R}}^{n+(m-1)/N_l}(\mathbf{x})\right), \tag{21}$$

which is a first-order approximation in time to the exact integral. We propose a better evaluation considering that the fields are obtained with a Adams–Bashforth scheme. Let us recall that this scheme uses on polynomial representations of fields

$$p_{\text{S}}(\mathbf{x}, t) = p_{\text{S}}^{n+(m-1)/N_l}(\mathbf{x}) + \sum_{s=1}^{3}\left(\int_0^{t^\star(t)} \ell_s(t')\, dt'\right) r_{\text{S}}^{n+(m-s)/N_l}(\mathbf{x}), \tag{22}$$

$$p_{\text{R}}(\mathbf{x}, t) = p_{\text{R}}^{n+(m-1)/N_l}(\mathbf{x}) + \sum_{s=1}^{3}\left(\int_0^{t^\star(t)} \ell_s(t')\, dt'\right) r_{\text{R}}^{n+(m-s)/N_l}(\mathbf{x}), \tag{23}$$

with $t^\star(t) = (t - t^{n+(m-1)/N_l})/\Delta t_l$, where $\ell_1(t)$, $\ell_2(t)$ and $\ell_3(t)$ are Lagrange interpolation functions with interpolation nodes at 0, -1 and -2, and $r_{\text{S}}^{n+(m-s)/N_l}$ and $r_{\text{R}}^{n+(m-s)/N_l}$ are the components of right-hand side vectors corresponding to pressure field. Injecting representations (22) and (23) in integral (20) gives

$$\begin{aligned}
&\Delta t_l\, p_{\text{S}}^{n+(m-1)/N_l}(\mathbf{x})\, p_{\text{R}}^{n+(m-1)/N_l}(\mathbf{x})\\
&+\Delta t_l^2 \sum_{s=1}^{3} a_s\, p_{\text{S}}^{n+(m-1)/N_l}(\mathbf{x})\, r_{\text{R}}^{n+(m-s)/N_l}(\mathbf{x})\\
&+\Delta t_l^2 \sum_{s=1}^{3} a_s\, r_{\text{S}}^{n+(m-1)/N_l}(\mathbf{x})\, p_{\text{R}}^{n+(m-s)/N_l}(\mathbf{x})\\
&+\Delta t_l^3 \sum_{s_1=1}^{3}\sum_{s_2=1}^{3} C_{s_1 s_2}\, r_{\text{S}}^{n+(m-s_1)/N_l}(\mathbf{x})\, r_{\text{R}}^{n+(m-s_2)/N_l}(\mathbf{x}),
\end{aligned} \tag{24}$$

with the coefficient matrix

$$\mathbf{C} = \begin{pmatrix} \frac{4703}{5040} & -\frac{457}{840} & \frac{52}{315} \\ -\frac{457}{840} & \frac{103}{315} & -\frac{251}{2520} \\ \frac{52}{315} & -\frac{251}{2520} & \frac{17}{560} \end{pmatrix}.$$

This formula gives the exact value of integral (20) for fields that are third-order approximation in time. Therefore, formula (24) theoretically keeps accuracy of numerical fields with an $\mathcal{O}(\Delta t^3)$ numerical error, while approximation (21) gives an $\mathcal{O}(\Delta t)$ error.

It is worthwhile to mention that this numerical evaluation is based on values of fields and right-hand side vectors that are available to update the fields with formula (18). The contribution (24) to the imaging condition is thus computed at each step right before the update of fields, without significant supplementary cost in comparison with approximation (21).

4 PARALLEL COMPUTATION AND IMPLEMENTATION

In this section, we describe the RiDG implementation for massively parallel computations on accelerator-aided clusters. The fundamentals and features of OCCA are described in Section 4.1. Our implementation for a single multithreading device is detailed in Section 4.2 and the adaptations for distributed-memory clusters are explained in Section 4.3. The memory transfers required for MPI communications and for the RTM procedure are discussed in Section 4.4.

4.1 Multithreading programming with OCCA

OCCA is a unified framework for multithreading programming that can be used for several shared-memory hardware architectures, such as CPUs, GPUs and Intel's Xeon Phi. The current version translates a single implementation of computational kernels to the OpenMP, OpenCL, pThreads, Intel COI and CUDA languages. It has been developed to maintain portability and performance together with platform-choice flexibility. Using this programming approach therefore allows customized implementations of algorithms for several computing devices with a single code.

The OCCA library provides a host API and a device API. The host API gives tools to generates a self-contained context and command queue from hardware devices, to allocate and manage device memory, and finally to compile and execute kernels. The device API allows us to write kernels that are compiled at run-time, and that send instructions to the device when executed. The kernel language of OCCA mirrors those used for GPU programming. Threads (work items) are grouped into groups (work groups), which compose the main grid. The work groups are queued for execution onto the available multiprocessors, and work items are executed in parallel. The optimum choices of the number of work groups, and the number of work items per work group depend on the algorithm and the characteristics of the device in use. These number are defined by the user, before the execution of each kernel, thanks to the host API.

Further information about OCCA can be found in a white paper (Medina *et al.* 2014). The latest developments are available on the website `http://www.libocca.org/`.

4.2 Implementation with one multithreading device

The complete computational procedure for the forward time stepping is given in Algorithm 1. It is straightforwardly adapted for the backward simulation. For each element k, $\mathtt{q}_k$ stores all the local values of fields (p and each Cartesian component of $\mathbf{v}$) and $\mathtt{qf}_k$ stores a copy of traces corresponding to face nodes (p and $\mathbf{n}\cdot\mathbf{v}$, where $\mathbf{n}$ is the outward unit normal to the face). When updating the right-hand-side vector, $\mathtt{q}_k$ is used to compute volume terms, while $\mathtt{qf}_k$ is used for the surface terms. The array $\mathtt{rhs}_k$ stores the right-hand side vectors corresponding to the three previous local time steps. n_{local} and N_{local} are the local step index and the number of local steps of the finest level.

This global procedure can be naturally decomposed into several computational kernels. These computational kernels are implemented in separate OCCA kernels, which allows us to optimize each task considering the characteristics of both the task and the device. Thereby, the loop iterations of Algorithm 1 are performed by the main process (on the 'host'), which calls the different kernels to execute operations on the device. Our implementation has three main kernels, which are called at each local time-step in this order:

(i) The *volume kernel* computes the volume terms and stores the result into array $\mathtt{rhs}_k$.

(ii) The *surface kernel* computes the surface terms and updates $\mathtt{rhs}_k$ with the result.

Algorithm 1: Forward time-stepping procedure

```
initialization
  initialize $\mathtt{q}_k$ for each element $k$
  initialize $\mathtt{qf}_k$ for each element $k$
  initialize $\mathtt{rhs}_k$ for each element $k$
for $n_{\text{global}} = 1, 2, \dots, N_{\text{global}}$ do
    for $n_{\text{local}} = 1, 2, \dots, N_{\text{local}}$ do
        $t_{\text{start}} = [(n_{\text{global}} - 1)N_{\text{local}} + (n_{\text{local}} - 1)] \cdot \Delta t_{\text{local}}$
        $t_{\text{new}} = t_{\text{start}} + \Delta t_{\text{local}}$
        for each element $k$ with unknowns evaluated at $t_{\text{start}}$ do
            update $\mathtt{rhs}_k$ for $t = t_{\text{start}}$
        for $l = N_{\text{levels}}, \dots, 2, 1$ do
            if the unknowns of level $l$ must be evaluated at $t_{\text{new}}$ then
                for each element $k$ of level $l$ do
                    update $\mathtt{q}_k$ and $\mathtt{qf}_k$ for $t = t_{\text{new}}$ with update scheme (18)
        for each element $k$ that has not been updated at this local time-step,
        but that is coarse-level neighbour of evaluated elements do
            update $\mathtt{qf}_k$ for $t = t_{\text{new}}$ with update scheme (19)
```

(iii) The *update kernel* performs the local time stepping with Adams–Bashforth schemes (18) and (19).

A supplementary kernel (the *source kernel*) generates signals at sources and receivers by adding a specific term in the right-hand side vector. For the RTM procedure, a specific kernel (the *image kernel*) updates an image array $\mathtt{I}_k$ with the contribution of the current local time-step to the imaging condition.

In implementing these kernels, we seek to optimize the use of the computational units of the hardware device and minimize waiting time when transferring data, in order to optimize run time. These questions have been studied by Klöckner *et al.* (2009, 2012, 2013) for the implementation of nodal DG schemes on GPU. The ideas have then been applied with an Adams–Bashforth multirate time stepping by Gödel *et al.* (2010) and Gandham *et al.* (2015) in different wave contexts.

4.2.1 *Memory on the device*

The locality of memory storage is addressed by storing all the data required by the numerical schemes on the device (e.g. in the global memory of a GPU). A floating-point array `q` stores all the discrete unknowns, and the array `qf` contains a copy of traces associated to face nodes. The array `rhs` is reserved for the right-hand side vectors associated to the three previous time-steps. In an RTM procedure, all these arrays are duplicated for source and receivers simulations, and the imaging condition is stored in the specific array `I`. When using an accelerator, such as a GPU, these arrays are only stored in its global memory and are never transferred to the RAM of the compute node, at any moment, except to export the final RTM image. Single precision is used for both storage and floating-point operations.

During an initialization procedure, the differentiation matrices $\mathbf{D}_j$ and the lifting matrices $\mathbf{L}_f$ are pre-computed and stored on the device in the aggregated arrays `Drst` and `Lift`. The Jacobian matrices $\partial\boldsymbol{\Psi}_k/\partial\mathbf{r}$ of all elements are stored in `vGeoFac`, and both Cartesian the components of normal vector and the Jacobian ratios $J_{k,f}/J_f$ are stored in `sGeoFac`. The physical parameters ρ_k and c_k are managed using a database principle: the array `phyDB` contains the parameters values for each kind of medium, and the mapping between medium indexes and mesh cells is stored in the array `phyMap`. Finally, several arrays list elements associated to each multirate time-stepping level, and to the coarse neighbors for the MRAB strategy.

In order to take full advantage of the cache memory, the different nodal values corresponding to a same field and a same element are stored contiguously in the array `q`. When fetching values from the memory to device registers, the memory bus operates by blocks of data instead of individual items. Since these nodal values must be used together to compute the first matrix-vector products of the right-hand sides of eqs (13) and (14), a contiguous storage is advantageous. We improve the utilization of the memory bus by padding the blocks of nodal values in `q`. N_{pad} array elements are thus reserved instead of N_p for each block, where $N_{\text{pad}} \geq N_p$ is chosen considering the characteristics of the bus. Then, the blocks of N_{pad} array elements corresponding to the different fields for a same element are stored contiguously. The coarsest granularity of storage thus corresponds to the element indexes. A similar granularity is used for `qf`, `rhs` and `I`. The dimensions and granularity of the main arrays are given in Table 1.

Table 1. List of main arrays stored in the global memory of the device. Dimensions are given from the coarsest to the finest granularity of storage. Symbols used are: the number of elements K, the number of nodes by element N_p, its padded version N_{pad}, the number of faces by element N_f, the number of nodes by face N_{fp}, the number of (scalar) fields N_{fields}, the number of traces N_{traces} and the spatial dimension N_{dim}.

Symbol	Dimensions	Definition
`q`	$K \cdot N_{\text{fields}} \cdot N_{\text{pad}}$	Values of fields at nodes
`qf`	$K \cdot N_{\text{traces}} \cdot N_f \cdot N_{fp}$	Values of traces at face nodes
`rhs`	$3 \cdot K \cdot N_{\text{fields}} \cdot N_{\text{pad}}$	Right-hand side terms
`I`	$K \cdot N_{\text{pad}}$	Imaging condition
`Drst`	$N_p^2 \cdot N_{\text{dim}}$	Differentiation matrices
`Lift`	$N_f \cdot N_{fp} \cdot N_p$	Lifting matrices
`vGeoFac`	$K \cdot N_{\text{dim}}^2$	Geometric factors for volume terms
`sGeoFac`	$K \cdot 4 \cdot N_f$	Geometric factors for surface terms

4.2.2 *Kernels*

The volume and surface kernels, shown in Algorithms 2 and 3, are conceived and optimized in a similar way. When one of them is executed, the volume or surface terms are computed and stored in `rhs` for all nodes that must be updated at the current local time step. For both kernels, each thread (work item) performs the operations for the N_{fields} items of array `rhs` associated to a given node and the current time step. Each work group processes the nodes of several elements, K_{blkV} for the volume kernel and K_{blkS} for the surface kernel. These parameters provide a way to tune the occupation of the device for each kernel, and then to adjust the load balancing. The volume kernel is then compiled for ceil(K/K_{blkV}) work groups of $K_{\text{blkV}} \cdot N_p$ work items and the surface kernel is compiled for ceil(K/K_{blkS}) work groups of $K_{\text{blkS}} \cdot \max(N_p, N_f N_{fp})$ work items.

The computation of volume and surface terms are efficiently processed in three subtasks, which correspond to the three OCCA inner loops of Algorithms 2 and 3. First, the geometric factors and the physical parameters, which are shared by all the nodes and face nodes of a given element, are transported from the global device memory to shared arrays. The second subtask consists in building the vectors that are used in the matrix-vector products of eqs (13) and (14). For the volume kernel, each work item fetches the field values of its assigned node, computes the linear combinations of velocity unknowns and stores the results in shared arrays. In addition, the right-hand-side vectors of the two previous time steps are updated. For the surface kernel, each work item fetches the field values of its assigned face node, both for the current and neighboring element. Then, it computes the penalty terms, multiplies them with the ratios of Jacobians, and stores the results in shared arrays. Finally, the matrix-vector products are computed in the third subtasks of kernels, and the results are added to `rhs`. Each work item effectively performs only one scalar product of vectors for each

Algorithm 3: Surface kernel

```
input
  pointers *qf, *rhs
  pointers *Lift, *sGeoFac, *phyMap, *phyDB, *typeMap
for each block b of elements do
    shared arrays pPena, uPena, vPena, wPena  (arrays $K_{\text{blkS}} \cdot N_f \cdot N_{fp}$)
    shared arrays sGeoFacWG  (array $K_{\text{blkS}} \cdot 4 \cdot N_f$)
    shared arrays phyParWG  (array $K_{\text{blkS}} \cdot 2$)
    for each element k of block b do
        for each node n of element k do
            load the geometric factors in sGeoFacWG for element k
            load the physical parameters in phyParWG for element k
    for each element k of block b do
        for each face node nf of element k do
            read the interior values of fields from qf
            read the type of face from typeMap
            if face at domain boundary then
                define the exterior values of fields with the condition
            else if face at interface then
                read the exterior values of fields from qf
            compute the components of penalty vectors
            multiply with the ratios of Jacobians
            store in pPena, uPena, vPena and wPena
    for each element k of block b do
        for each node n of element k do
            read the values of lifting matrices from Lift
            compute Lift · pPena, Lift · uPena, Lift · vPena and Lift · wPena
            add the resulting surface terms to rhs
```

Algorithm 2: Volume kernel

```
input
  pointers *q, *rhs
  pointers *Drst, *vGeoFac, *phyMap, *phyDB
for each block b of elements do
    shared arrays P, UdotGr, UdotGs, UdotGt  (arrays $K_{\text{blkV}} \cdot N_p$)
    shared arrays vGeoFacWG  (array $K_{\text{blkV}} \cdot 9$)
    shared arrays phyParWG  (array $K_{\text{blkV}} \cdot 2$)
    for each element k of block b do
        for each node n of element k do
            load the geometric factors in vGeoFacWG for element k
            load the physical parameters in phyParWG for element k
    for each element k of block b do
        for each node n of element k do
            read the values of fields from q
            compute the linear combinations for the $p$ equation
            store P, UdotGr, UdotGs and UdotGt
    for each element k of block b do
        for each node n of element k do
            read values of the differentiation matrices from Drst
            compute Dr · UdotGr + Ds · UdotGs + Dt · UdotGt for the $p$ equation
            compute Dr · P, Ds · P and Dt · P, and the linear combinations for the $v_i$'s equations
            compute the volume terms and store in rhs
```

Algorithm 4: Update kernel

```
input
  pointers *q, *qf, *rhs and *Fmask
for each block b of elements do
    shared array qNew  (array $N_{\mathrm{fields}} \cdot K_{\mathrm{blkS}} \cdot N_p$)
    for each element k of block b do
        for each node n of element k do
            compute the updated fields and store the values in qNew
    for each element k of block b do
        for each node n of element k do
            update q with the new values stored in qnew
    for each element k of block b do
        for each face node nf of element k do
            read node index n corresponding to face node nf in Fmask
            update qf with the new values stored in qnew
```

matrix-vector product. This operation is optimized thanks to the coherent granularity of storage and some supplementary fine-tuning (e.g. loop unrolls and storage of intermediates parameters in constant memories). Further details about these strategies can be found in the works of Klöckner *et al.* (2009, 2012, 2013).

The update kernel performs the time stepping (18) for all nodal values at a given time level l, stores the new values in arrays `q` and updates the traces in `qf`, as shown in Algorithm 4. Each work item updates the nodal values associated to a given node, and each work group deals with K_{blkS} elements. The operations are again achieved in subtasks. First, the values are updated and stored in a shared array. After a synchronization, they are transported into arrays `q` and `qf`, requiring respectively N_p and $N_f N_{fp}$ work items for the most inner loop. The size of the work group is thus $K_{\mathrm{blkS}} \cdot \max(N_p, N_f N_{fp})$. This kernel is also used to update coarse neighbor elements with the time stepping (19). The image kernel is completely analogous.

4.3 Towards an implementation for cluster of devices

The main challenge for large-scale computations with a large number of cores (with or without accelerators) is to reach a good parallel scalability for the final implementation. This requires to take care of two critical aspects: load balancing and managing the latency inherent to data transfers between compute nodes.

In our implementation, the device associated to each compute node of the cluster performs the whole computation for a subdomain corresponding to one part of the mesh. The procedure presented in the previous section is naturally applied to each device. At each local time step, MPI communications transfer the traces of fields situated at the interface between two subdomains when necessary. For the backward run of the RTM procedure, both source and receivers simulations are performed using the same mesh partition: this ensures the computation of the imaging condition without extra data transfers between processes.

The mesh partition is generated using METIS with a weighting strategy. The weight $N_l = 2^{l-1}$ is associated to each cell, where l is the time level of the cell. As explained in Section 3.2, N_l is also the number of local iterations required to achieve a global iteration for elements of level l. This number then is proportional to the compute load required for the corresponding element, which allows to balance the global load between the compute nodes.

Our experience has shown that the elements belonging to the finest time level were generally isolated in the mesh (see benchmark of Section 5.2). In the multirate time-stepping strategy, these elements are updated at every smallest time-step, and only those are updated for half smallest time steps. By preventing METIS to place them at interfaces between mesh parts, we remove the need for MPI communications when only those are updated, which represents half of all potential MPI communications. This is done by lumping each finest element (or group of finest elements) with his neighbors before METIS partition, and by separating them after the procedure.

4.4 Memory transfers for MPI communications and RTM procedure

We now detail our strategies for the memory transfers that are required to connect subdomains together, and to record/replay fields at the domain boundary for the source simulation in the RTM procedure. Both operations require different kinds of memory transfers:

(i) The connection of subdomains is obviously made through MPI communications (host–host transfers). Data to send must be pre-fetched from device memory, while received data are loaded on the device memory right after. Host-device transfers are then required.

(ii) Boundary data recorded for the source simulation must be stored in a memory large enough. The device memory usually being too small, the data are accumulated and stored either in the host memory—more precisely, in its random-access memory (RAM)—or on a hard disk drive (HDD). We considered both options. This then requires host-device transfers with optionally RAM–HDD transfers.

The flow chart of the complete RTM procedure with the memory transfers is shown in Fig. 1. Our goal was to hide latency inherent to transfers regardless of the kind of transfer. Therefore, we used non-blocking transfers whenever it was possible, in order to perform the different transfers alongside computations on the device. While our implementation runs on clusters with only CPUs, it is mainly conceived for clusters with accelerators. In that context, calling a kernel is a non-blocking operation for the compute (host) node. The operations mentioned in Fig. 1 are performed by the compute node, and data transfers from/to are written from that point of view. We describe hereafter several of the strategies that were employed.

The majority of data transfers are performed at the beginning of iterations, alongside kernels that do not need to immediately process transferred data: the volume kernel, the surface kernel for inner elements and the source kernel. On the flow chart (Fig. 1), these operations correspond to the two main groups of successive grey boxes. By calling twice the surface kernel, once for inner elements (that do not touch the border of the subdomain) and once for outer elements, we increase the compute time available to hide data transfers. Before starting MPI communications, data to be sent to neighboring subdomains are fetched from the device memory to the host memory. Because these data must already be on the host memory to start communications, blocking host-device transfers are used. By contrast, during the backward simulation, boundary data can be loaded onto the device memory with non-blocking host-device transfers. RAM–HDD exchanges are made in a blocking way after all the non-blocking operations, but occur alongside them. When all these operations are completed, data received with MPI communications is loaded on the device memory and surface terms of outer elements are computed. This ends the computation of the right-hand-side terms.

Different containers are used to store and to transfer boundary data during the RTM procedure. A specific buffer array is allocated

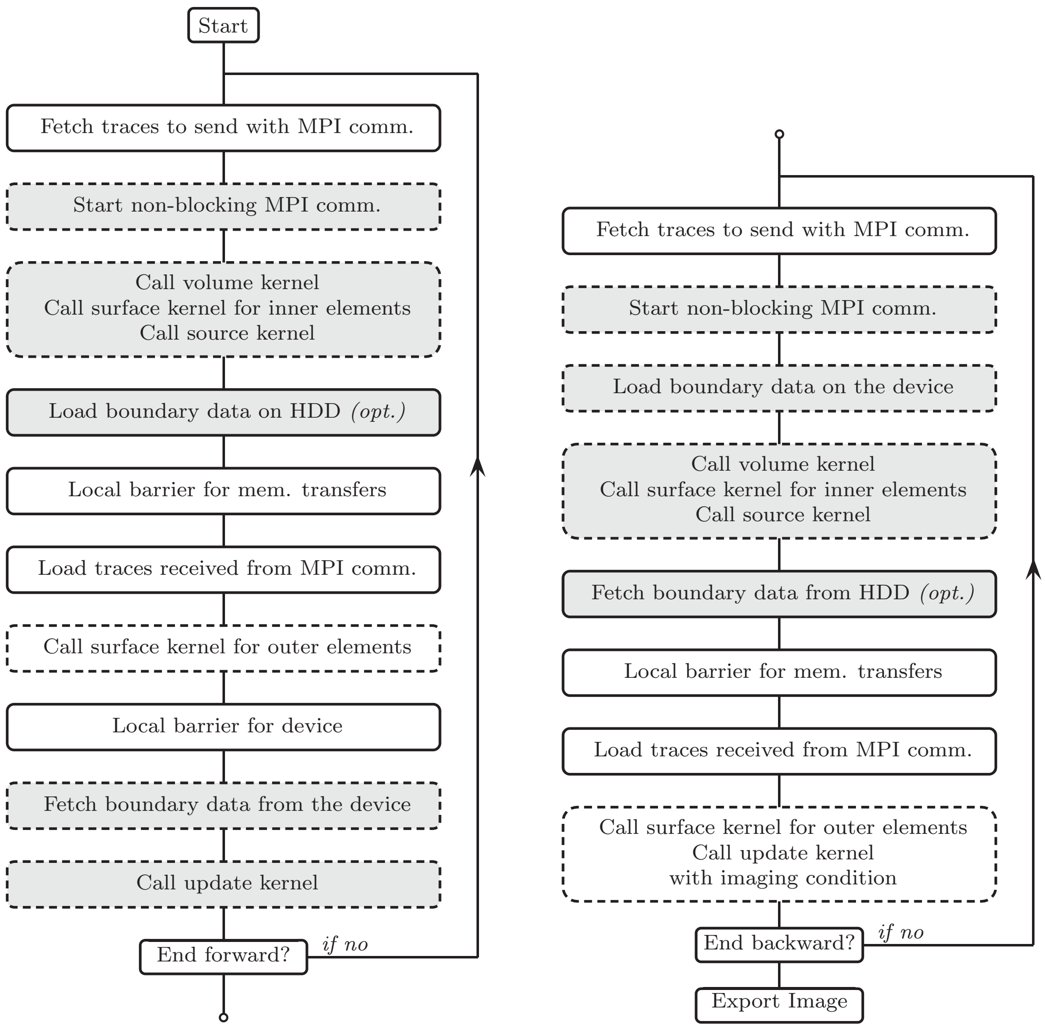

Start
Fetch traces to send with MPI comm.
Start non-blocking MPI comm.
Call volume kernel
Call surface kernel for inner elements
Call source kernel
Load boundary data on HDD *(opt.)*
Local barrier for mem. transfers
Load traces received from MPI comm.
Call surface kernel for outer elements
Local barrier for device
Fetch boundary data from the device
Call update kernel
End forward?
if no
Fetch traces to send with MPI comm.
Start non-blocking MPI comm.
Load boundary data on the device
Call volume kernel
Call surface kernel for inner elements
Call source kernel
Fetch boundary data from HDD *(opt.)*
Local barrier for mem. transfers
Load traces received from MPI comm.
Call surface kernel for outer elements
Call update kernel
with imaging condition
End backward?
if no
Export Image

Figure 1. Flow chart of the complete RTM procedure with the forward phase (left) and the backward phase (right). Each iteration of loops corresponds to a local time-step update (forward or backward). For the backward phase, each kernel performs the same operations for both source and receivers simulations. Boxes with dashed borders correspond to operations that are non-blocking for the host. Successive boxes with grey background correspond to independent operations that can be performed at the same time.

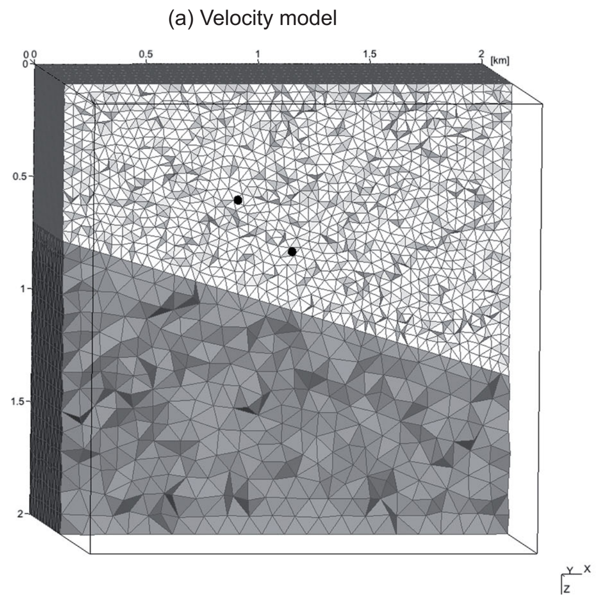
on the device memory for temporary storage. When the surface kernel is executed for outer elements, the traces corresponding to boundary nodes are copied into that buffer during the forward phase, and are read from it during the backward phase. This explain why, during the forward phase, boundary data are transferred from the device memory to the host memory at the end of each iteration. In the host memory (i.e. the RAM of the compute node), accumulated boundary data are stored in a sequence container with double ended queue. At each forward iteration, a new container is added at one end of the sequence with the current boundary data. When using a HDD to unload the RAM, data are transferred to the HDD by reading the sequence container from its other end, with a time delay of one global time step. This means that the sequence container always contains boundary data corresponding to one global time step. Since both host-device and RAM–HDD transfers involve different pieces of data, they can be performed simultaneously. The strategy is identical for the backward phase, except that HDD–RAM transfers are performed one global time step in advance.

5 COMPUTATIONAL RESULTS

5.1 Validation case

In order to validate our implementation, numerical convergence as well as run times, we use a reference benchmark proposed by Minisini *et al.* (2012) and Zhebel *et al.* (2014).

The computational domain of the benchmark is a cube $\Omega = [0, 2\,\text{km}] \times [0, 2\,\text{km}] \times [0, 2\,\text{km}]$ made of two media separated with a dipping plane interface (Fig. 2a). The density is $1\,\text{g}\,\text{cm}^{-3}$ in both media and the velocity is $1.5\,\text{km}\,\text{s}^{-1}$ in the upper one and $3\,\text{km}\,\text{s}^{-1}$ in the lower one. The plane interface runs from 0.7 km in depth at $x = 0$, to 1.3 km at $x = 2$ km. A Ricker pulse with a peak frequency of 12 Hz is generated at a point source located above the interface at position $\mathbf{x}_s = (779.7\,\text{m}, 1000\,\text{m}, 516.3\,\text{m})$, and creates a spherical wave (Fig. 2b). The peak of the Ricker pulse is generated at instant $t_p = 0.1127$ s and the final time of the simulation is $t_f = 0.8127$ s. During the simulation, the pressure is recorded at a receiver situated at position $\mathbf{x}_r = (1023.9\,\text{m}, 1000\,\text{m}, 746.2\,\text{m})$. The recorded signal is compared to the exact solution over the period $[t_p, t_f]$ using the



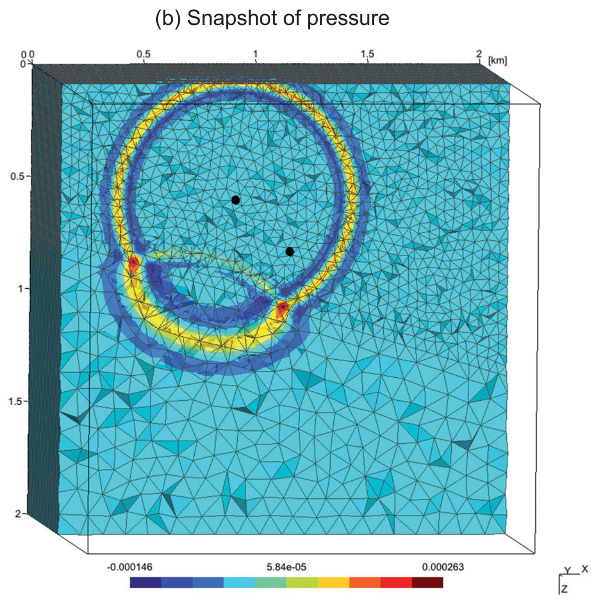


Figure 2. Representation of the half part of the computational domain with the coarsest mesh (a) and snapshot of the pressure at one instant (b). The velocity is 1.5 km s^{-1} in the upper medium (light grey) and 3 km s^{-1} in the lower one (dark grey). The source position (upper black point) and the receiver position (lower black point) are represented on both figures. The box indicates the total size of the computational domain.

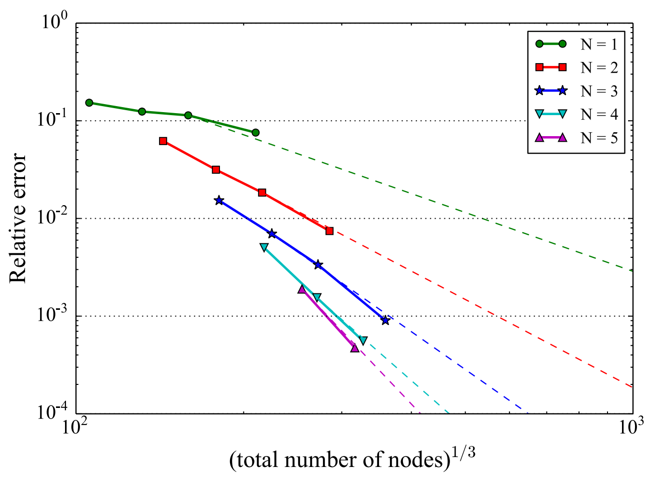


Figure 3. Numerical convergence of the numerical scheme for different polynomial degrees N with the validation case. For each degree, the relative error is plotted as a function of the cube root of the total number of node N_pK. Dashed lines correspond to h^{p+1} convergence.

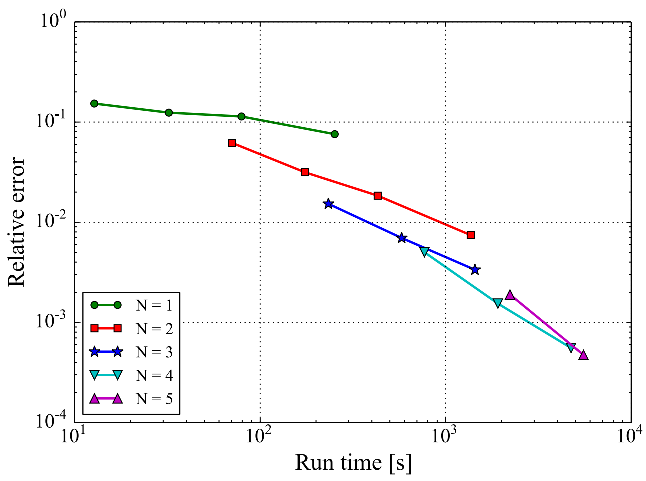


Figure 4. Computational performance of the implementation for different polynomial degrees N and four meshes with the validation case. The run times are obtained with one single Nvidia K20m GPU and CUDA through OCCA. For the highest degrees, simulations have not been performed with the finest meshes due to memory limitation of the GPU.

relative error defined as (Zhebel *et al.* 2014)

$$\sqrt{\frac{\int_{t_p}^{t_f} |p_{\text{num}} - p_{\text{ref}}|^2 \, dt}{\int_{t_p}^{t_f} |p_{\text{ref}}|^2 \, dt}}, \tag{25}$$

where $p_{\text{num}}(t)$ and $p_{\text{ref}}(t)$ are, respectively, the recorded and reference signals. Because of the chosen final time, the boundary conditions have no influence on the error.

The convergence of the error for polynomial basis functions with different degrees is shown in Fig. 3. For each degree N, relative errors are obtained with several meshes (with 294 508, 567 071, 1 005 575 and 2 320 289 tetrahedra). The time-stepping scheme is used with only one time level and the time step

$$\Delta t = 0.15 \min_k \left\{ \frac{\ell_k}{(N+1)^2 c_k} \right\},$$

where ℓ_k is the smallest perpendicular distance between a face and its opposite vertex for tetrahedron k. As expected, the error curve is close to the classical h^{p+1} convergence of upwind fluxes for all polynomial degrees.

Fig. 4 shows performance curves with a single Nvidia K20m GPU and the CUDA programming framework through OCCA. It confirms the interest of hp-refinement strategy. Indeed, an optimum polynomial degree exists depending on the desired accuracy. For instance, to reach relative error 10^{-2}, the third degree is more efficient than the second degree with a finer mesh. By contrast, for the relative error 10^{-3}, the fourth degree is worthwhile compared to the fifth with a coarser mesh. Therefore, the best strategy to improve accuracy of solution is to refine the mesh and to increase the polynomial degree together.

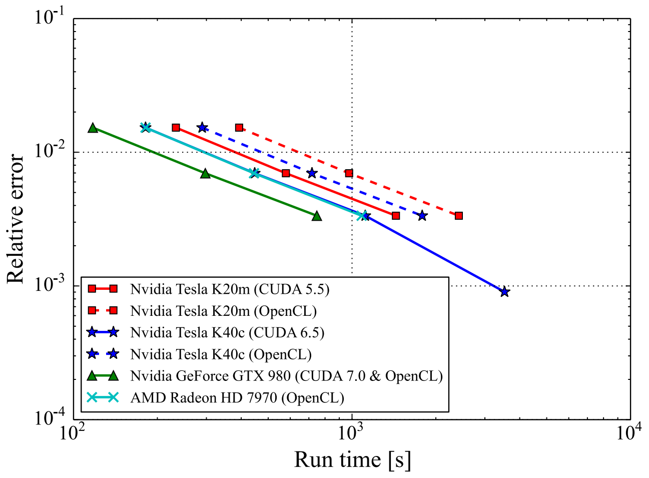


Figure 5. Comparison of run times obtained with different hardware devices. Third-degree polynomial basis functions are used with a single-rate time-stepping scheme.

Finally, Fig. 5 shows performance curves obtained with several GPUs: Nvidia Tesla K20m, Nvidia Tesla K40m, Nvidia GeForce GTX 980 and AMD Radeon HD 7970. All the Nvidia GPUs have been tested with CUDA and Nvidia OpenCL. The results are in accordance with the specifications of these GPUs. The floating-point performances of both Tesla K40c and Radeon HD 7970 are similar. Tesla K20m and GTX 980 have, respectively, the smallest and largest floating-point performances according to the specifications of the constructor. For both Nvidia Tesla GPUs, the implementation is faster with CUDA than with OpenCL (speedup: $\sim$1.65), while identical performances have been obtained with the GTX 980. Only the Tesla K40c has a enough large memory to deal the finest mesh, for which 6.16 GB must be stored on the GPU.

5.2 RTM case for a cluster of GPUs

This second benchmark deals with the complete RTM procedure for a larger case. The classical imaging condition (4) and the new one based on characteristics (7) are compared numerically. Both scalability and memory transfers of our implementation are studied using up to 32 Nvidia K20 GPUs of TACC's Stampede cluster.

The geometry consists in a salt dome embedded in a multilayered domain of size $8.9 \times 4.44 \times 5.1\,\mathrm{km}^3$. The top layer corresponds to sea water, while the other represent sediments and rocks. The velocity model is shown in Fig. 6(a). The discretization consists of a tetrahedral mesh made of 2 787 335 elements with a third-degree polynomial basis, which corresponds to 222 986 800 discrete unknowns per simulation. The mesh was generated by Kononov *et al.* (2012) and used by Minisini *et al.* (2013).

For this case, the multirate time-stepping scheme is used with three time levels. The coarse, intermediate and fine levels contain, respectively, 94.2 per cent, 5.7 per cent and less than 0.1 per cent of elements. The multirate time stepping is thus particularly interesting here. As shown in Fig. 6(b), most of the intermediate and fine elements are close to geometrical features (domain boundary and interfaces between layers). Since the smallest elements are isolated in the mesh, the lumping strategy in the partition procedure allows us to avoid MPI exchanges for the finest level without penalizing the load balancing between MPI processes.

5.2.1 *Synthetic survey and imaging conditions*

A synthetic survey has been performed with 159 receiver gathers situated in the top layer of the domain. The position of the first source is $\mathbf{x}_s = (8.9\,\mathrm{km}, 2.2\,\mathrm{km}, 8\,\mathrm{m})$ and the corresponding receivers are placed at depth $z_r = 10\,\mathrm{m}$ on the grid $(x_r, y_r) \in [5.9\,\mathrm{km}, 7.9\,\mathrm{km}] \times [2.1\,\mathrm{km}, 2.3\,\mathrm{km}]$ with regular steps of 25 m and 50 m. The same setting is used for the other gathers, but with positions moved by 50 m, 100 m, 150 m, etc. in the negative x-direction. In all cases, a Ricker pulse is emitted at the source with peak frequency 50 Hz, and the peak is generated at $t_p = 0.1127\,\mathrm{s}$. The total simulation time is $t_f = 3.6127\,\mathrm{s}$. The imaging conditions are computed over the period $[t_i, t_f]$ with $t_i = 0.75\,\mathrm{s}$. The backward run was only performed for that period. The transparent condition of eq. (3) is used at the domain boundary. The boundary values are saved and replayed for the source re-simulation, simultaneously with the reverse-time receiver wavefield computation.

Fig. 7 shows the final images obtained with the classical imaging condition (4) and the characteristics-based one (7). The main geometrical features of the reflectors are imaged with both conditions, but low-frequency noise contaminates the image obtained with the classical condition. This noise is suppressed by the new characteristics-based condition, providing a cleaner image. A similar observation has been made by Liu *et al.* (2011) with their condition based on a spatial Fourier transform to decompose the wavefields into up- and downgoing components. In contrast to that work, we use vertical characteristics to separate up- and downgoing waves at a negligible supplementary cost compared to the classical condition.

We should emphasize that wavefields can be straightforwardly decomposed locally along any oblique direction by choosing characteristics with the corresponding direction vector instead of $\mathbf{e}_z$ in formula (8).

5.2.2 *Memory and computational performance*

For each shot, the RTM procedure has been performed using 32 GPUs of TACC's Stampede cluster and the Gmsh software (Geuzaine & Remacle 2009) was used for post-processing operations. 14.5 GB of data must be allocated on GPUs, fairly well distributed between GPUs (from 353 to 498 MB by GPU). Boundary data are here saved only on RAM (no RAM-HDD transfers). They requires 240 GB in total or, on average, 7.5 GB by compute node. However, the distribution between the compute nodes is not well balanced: one compute node does not store data (the corresponding subdomain does not touch the domain boundary) and one node must store 14.34 GB. This is fortunately not a problem because compute node are configured with 32 GB of host memory and, as shown later in this section, latency due to memory transfers between host (RAM) and device (GPU memory) is quite well hidden by computations.

The run time for the complete procedure with one shot is 12.08 min, which includes a preparatory phase of 1.75 min (mesh partition, generation of lists, etc.), and, respectively, 4.21 and 6.12 min for forward and backward phases. The preparatory phase can be done only once for the complete survey, since it is the same for all shots. Let us mention that the backward phase has been made only for the period $[t_i, t_f]$.

While the backward phase requires two simulations alongside the computation of the imaging condition, the run time for a global time step is only 1.84–1.89 larger than for the forward phase, where only one simulation must be performed. We then have a factor smaller than two, while a larger one could be expected. This result is due

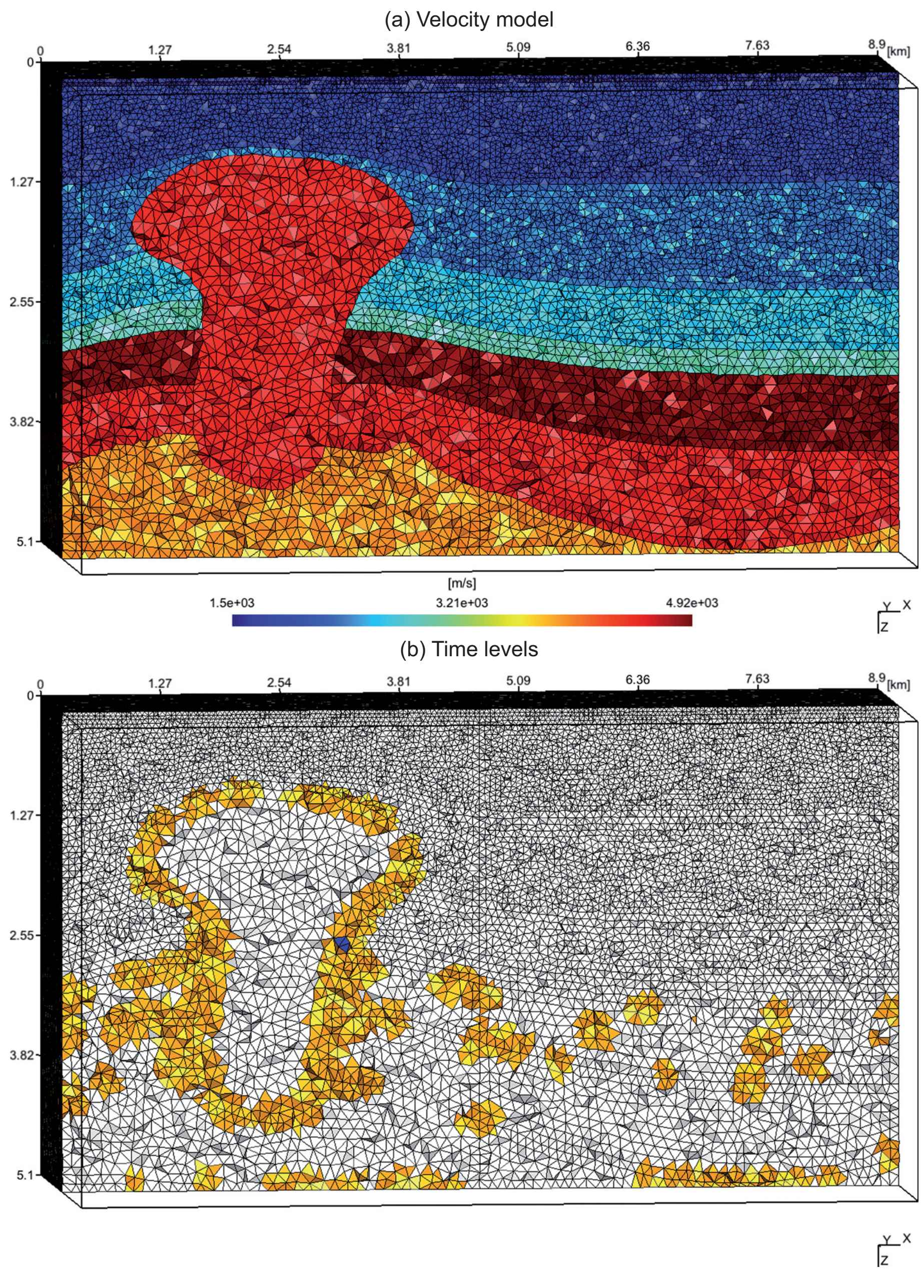


Figure 6. Velocity model (a) and time levels (b) for the RTM benchmark. Elements of the coarse, intermediate and fine levels are, respectively, white, orange and blue on figure (b). The box indicates the total size of the computational domain.

to the computation of both source and receivers wavefields with merged kernels during the backward phase (see Fig. 1). Memory transfers are saved when data used for both simulations are fetched only once, such as the differentiation/lifting matrices and the physical parameters. The update of the imaging condition, performed in the same kernel as the update of wavefields, is made at nearly no cost since the required wavefields values are already loaded (see Section 3.3).

5.2.3 *Scalability and memory transfers*

Our implementation exhibits good strong scalability, despite high latency data transfers, as shown in Fig. 8. Run times are obtained using 4–32 GPUs, considering both single-rate and multirate time stepping, and three alternative strategies to study memory transfers: boundary data are saved on an HDD, they are saved on the RAM associated to each compute node, and all slow communications are cut-off (boundary data transfers and MPI communications).

When using the HDD storage for boundary data, the scalability is lost for the forward phase with 24 and 32 GPUs. Let us recall that this storage requires supplementary RAM–HDD transfers, which are slower than transfers between memory device and RAM. In this case, the RAM-HDD transfers, take too long to be hidden by the run time of kernels. Fortunately, for a large enough number of GPUs, boundary data can be stored only in RAM, and scalability is recovered.

Using multirate time stepping instead of the single-rate version provides a speed up between 3.4 and 3.6, and preserves the scalability. MPI transfers and device-host boundary data transfers are not perfectly hidden by computations because of blocking operations,

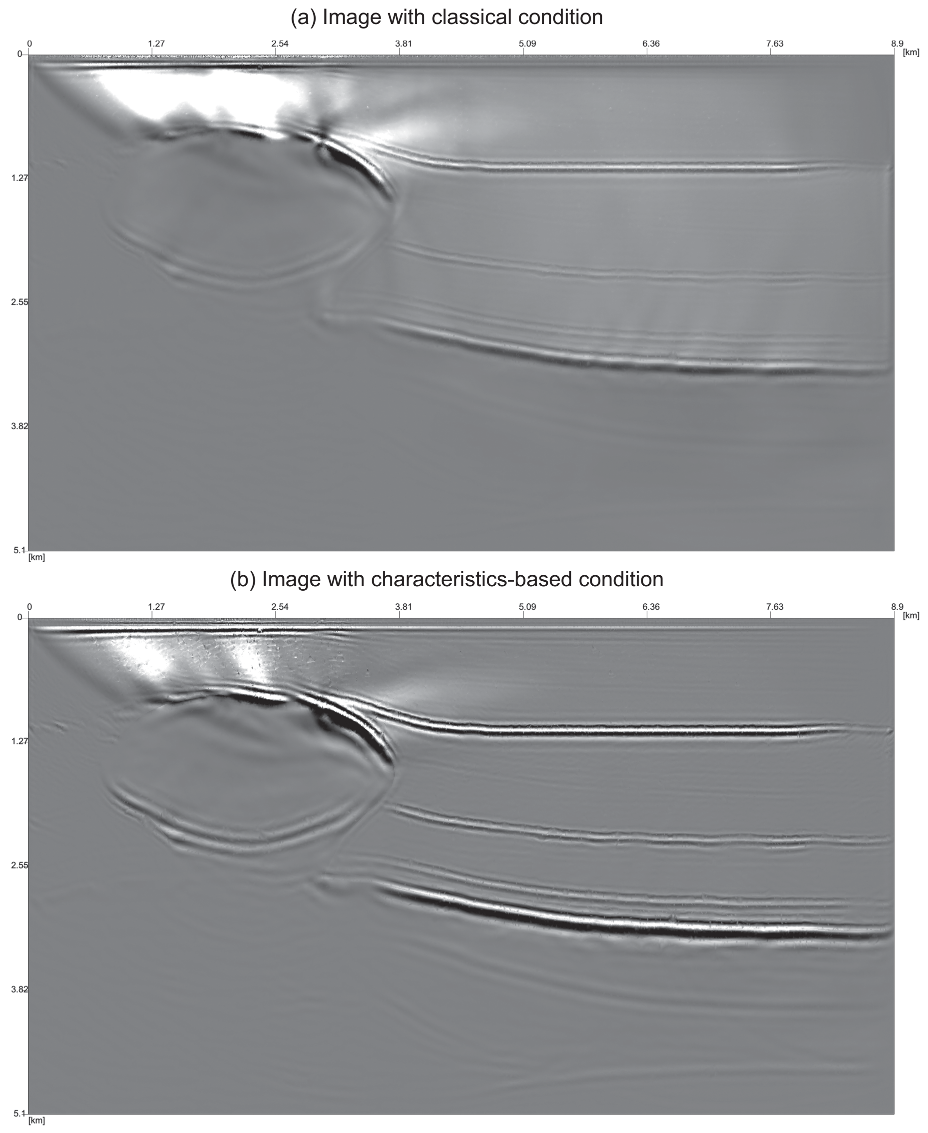


Figure 7. Final images in the *x*–*z* plane for the RTM benchmark obtained with the classical imaging condition (a) and the new one based on characteristics (b).

such as the loading of traces received from MPI communications (see Fig. 1). This however represent only few per cent of the total run time, and does not impact the strong scalability.

6 CONCLUSION

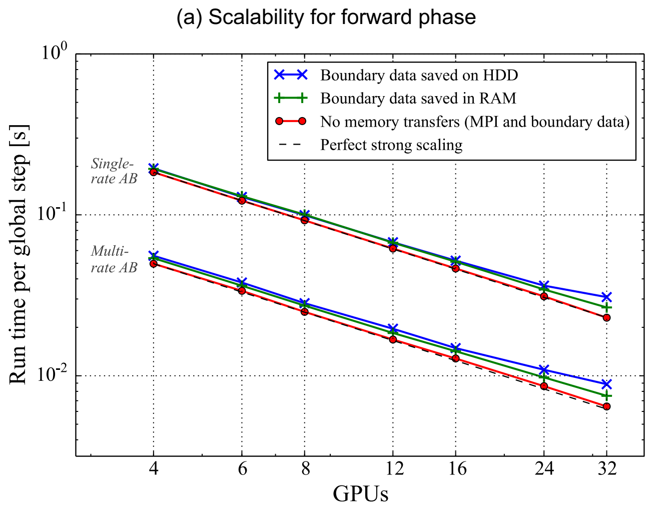
We have presented a high-performance computational strategy to compute RTM images with accelerator-aided clusters. It combines efficient numerical schemes, a flexible programming approach and a low memory storage/transfer strategy for RTM. Our final implementation exhibits excellent strong scalability with massively parallel GPUs. In addition, we have introduced a novel imaging condition that reduces noise in the final image, without significant extra computational cost in comparison with classical strategies. We summarize hereafter key features contributing to performance:

(i) The spatial scheme, based on a nodal DG method, enables high-order convergence rates. Since the weak element-to-element coupling and the dense algebraic operations required over each element, it is a suitable scheme for parallel computation on acceleration devices, especially with high-degree basis functions.

(ii) The levels-based multirate time-stepping scheme significantly improves run times for multiscale cases with unstructured meshes, which are common in seismic imaging. For parallel computations, mesh partition with METIS preserves good load balancing between cores thanks to strategies based on weights and lumping of elements.

(iii) The programming approach with MPI and OCCA provides a portable implementation that can be run on state-of-the-art computing architectures, and can be optimized for each of them. While the results presented herein are obtained with kernels optimized for GPUs, tuning for alternative architectures requires to modify only a few parameters and, eventually, only a few code lines.

(iv) While RTM procedure generally requires massive data storage with slow I/O, the thin halo regions inherent in DG discretizations overcome this bottleneck. Low storage requirements for DG boundary data allows halo trace data to be stored in RAM rather than relying on HDD, even for larger test cases. In the worst cases, where available RAM is not large enough, data exchange between RAM and HDD can be performed asynchronously with a limited, possibly insignificant, increase of run time.



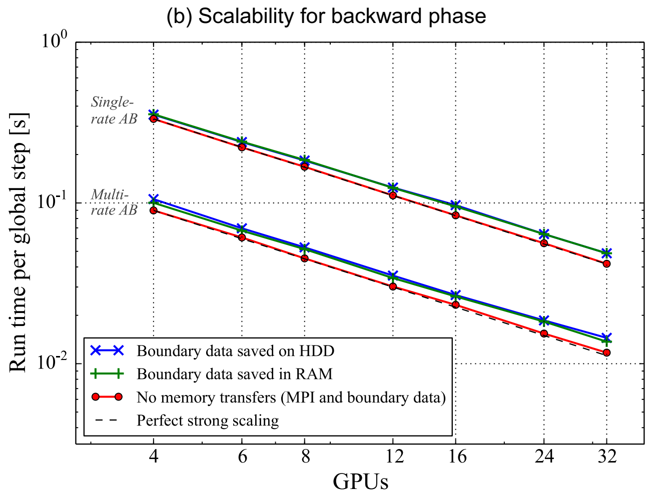


Figure 8. Strong scalability for the RTM benchmark with 4–32 Nvidia K20 GPUs. Run times are given separately for the forward phase (a) and the backward phase (b), both using the classical single-rate Adams–Bashforth (AB) scheme and using the multirate version with three time levels. In order to study the influence of memory transfers on performance, we consider cases where the boundary data are stored on an HDD, where they are only stored in the RAM associated to each compute node, and where both MPI node-to-node communications and boundary data transfers are disabled.

(v) Several optimization strategies for data storage and data movement improve run time: locality of storage, specific granularity of storage to enable cache reuse and hidden/asynchronous memory transfers for device-host and MPI node-to-node communications.

In the future, our implementation will be enriched with more realistic physical modules (i.e. spatially varying coefficients, anisotropic media and elastic waves) and more accurate domain truncation techniques (e.g. perfectly matched layers and high-order absorbing boundary conditions). It could be tricky to incorporate some of these enrichments in our computational strategy without penalizing too much both run time and scalability. Performances could be improved by using supplementary compression strategies (see e.g. Aguilar *et al.* 2013; Sun & Fu 2013) and DG formulations for meshes dominated by hexahedra (see e.g. Kirby *et al.* 2000; Chan *et al.* 2015).

In the perspective of industrial applications with finite-element approaches, another important issue to address is mesh generation (see e.g. Caumon *et al.* 2009; Kononov *et al.* 2012; Pellerin *et al.* 2014). For classical engineering applications, geometrical descriptions are carried out using Computer Aided Design (CAD) software and finite-element meshes can be directly generated on the basis of these descriptions (see e.g. Geuzaine & Remacle 2009). By contrast, geophysical domains are described through discrete data sets including voxels, triangulations and point clouds, which require processing steps before being suitable input to mesh generators. In addition, many configurations encountered in nature involve fractures, faults and hard contrasts of medium properties, which make mesh generation far more difficult. These geological features must indeed be detected and meshed and can lead to topological problems when they intersect. This currently prevents the use of existing mesh generators in an automatic fashion, and remains the main challenge in the acceptance of finite-element solvers in production seismic imaging. The work in this paper is envisioned as an exploratory step in anticipation of mature mesh generation tools for geophysical modeling.

ACKNOWLEDGEMENTS

This work has been funded by Shell Global Solutions International B.V. and Shell Oil Company. Axel Modave is partially supported by an excellence grant from Wallonie-Bruxelles International (WBI). He is an Honorary Fellow of the Belgian American Educational Foundation (BAEF) and a Postdoctoral Researcher on leave with the F.R.S-FNRS. The authors acknowledge Ty Mckercher and NVIDIA Corporation for providing GPU equipment, and the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper. The authors gratefully thank Rajesh Gandham, David Medina and Jean-François Remacle for helpful discussions and Josep de la Puente for his helpful comments on the manuscript.

REFERENCES

Abdelkhalek, R., Calandra, H., Coulaud, O., Latu, G. & Roman, J., 2012. Fast seismic modeling and reverse time migration on a graphics processing unit cluster, *Concurr. Comput.: Pract. Exp.,* **24**(7), 739–750.

Aguilar, G., Hanzich, M., Rubio, F., Gutierrez, N. & Cela, J.M., 2013. Efficient lossy compression for seismic processing, in *Proceedings of the 75th EAGE Conference & Exhibition*, Extended abstract.

Baffet, D., Hagstrom, T. & Givoli, D., 2014. Double absorbing boundary formulations for acoustics and elastodynamics, *SIAM J. Sci. Comput.,* **36**(3), A1277–A1312.

Baldassari, C., Barucq, H., Calandra, H. & Diaz, J., 2011. Numerical performances of a hybrid local-time stepping strategy applied to the reverse time migration, *Geophys. Prospect.,* **59**(5), 907–919.

Baldassari, C., Barucq, H., Calandra, H., Denel, B. & Diaz, J., 2012. Performance analysis of a high-order discontinuous Galerkin method, application to the reverse time migration, *Commun. Comput. Phys.,* **11**(2), 660–673.

Baysal, E., Kosloff, D. & Sherwood, J., 1983. Reverse time migration, *Geophysics,* **48**(11), 1514–1524.

Bleistein, N., Cohen, J.K. & Stockwell, J.W., 2001. *Mathematics of Multidimensional Seismic Imaging, Migration, and Inversion,* Vol. 13, Springer.

Caumon, G., Collon-Drouaillet, P., Le Carlier de Veslud, C., Viseur, S. & Sausse, J., 2009. Surface-based 3D modeling of geological structures, *Math. Geosci.,* **41**(8), 927–945.

Chan, J., Wang, J., Modave, A., Remacle, J.-F. & Warburton, T., 2015. GPU-accelerated discontinuous Galerkin methods on hybrid meshes, http://arxiv.org/abs/1507.02557.

Chin-Joe-Kong, M.J.S., Mulder, W.A. & Van Veldhuizen, M., 1999. Higher-order triangular and tetrahedral finite elements with mass lumping for solving the wave equation, *J. Eng. Math.,* **35**(4), 405–426.

Claerbout, J.F., 1971. Toward a unified theory of reflector mapping, *Geophysics,* **36**(3), 467–481.

Claerbout, J.F., 1985. *Imaging the Earth's Interior,* Blackwell Scientific Publications, Inc.

Clapp, R.G., 2009. Reverse time migration with random boundaries, in *Proceedings of the 79th SEG Annual Meeting,* Vol. 28, pp. 2809–2813.

Cohen, G., Joly, P., Roberts, J.E. & Tordjman, N., 2001. Higher order triangular finite elements with mass lumping for the wave equation, *SIAM J. Numer. Anal.,* **38**(6), 2047–2078.

Collis, S.S., Ober, C.C. & van Bloemen Waanders, B.G., 2010. Unstructured discontinuous Galerkin for seismic inversion, in *Proceedings of the 80th SEG Annual Meeting*, p. 2767.

de la Puente, J., Käser, M., Dumbser, M. & Igel, H., 2007. An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes—IV. Anisotropy, *Geophys. J. Int.,* **169**(3), 1210–1228.

Dumbser, M. & Käser, M., 2006. An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes—II. The three-dimensional isotropic case, *Geophys. J. Int.,* **167**(1), 319–336.

Dumbser, M. & Käser, M., 2009. A p-adaptive discontinuous Galerkin method with local time steps for computational seismology, in *High Performance Computing in Science and Engineering, Garching/Munich 2007,* pp. 569–584, Springer.

Dussaud, E., Symes, W.W., Williamson, P., Lemaistre, L., Singer, P., Denel, B. & Cherrett, A., 2008. Computational strategies for reverse-time migration, in *Proceedings of the 78th SEG Annual Meeting,* p. 2267.

Etgen, J. & Michelena, R., 2010. Introduction to this special section: reverse time migration, *Leading Edge,* **29**(11), 1363–1363.

Etienne, V., Chaljub, E., Virieux, J. & Glinsky, N., 2010. An hp-adaptive discontinuous Galerkin finite-element method for 3-D elastic wave modelling, *Geophys. J. Int.,* **183**(2), 941–962.

Fuhry, M., Giuliani, A. & Krivodonova, L., 2014. Discontinuous Galerkin methods on graphics processing units for nonlinear hyperbolic conservation laws, *Int. J. Numer. Methods Fluids,* **76**(12), 982–1003.

Gandham, R., Medina, D. & Warburton, T., 2015. GPU accelerated discontinuous Galerkin methods for shallow water equations, *Commun. Comput. Phys.,* **18**(1), 37–64.

Geuzaine, C. & Remacle, J.-F., 2009. Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities, *Int. J. Numer. Methods Eng.,* **79**(11), 1309–1331.

Gödel, N., Schomann, S., Warburton, T. & Clemens, M., 2010. GPU accelerated Adams-Bashforth multirate discontinuous Galerkin FEM simulation of high-frequency electromagnetic fields, *IEEE Trans. Magn.,* **46**(8), 2735–2738.

Hagstrom, T. & Warburton, T., 2009. Complete radiation boundary conditions: minimizing the long time error growth of local methods, *SIAM J. Numer. Anal.,* **47**(5), 3678–3704.

Hesthaven, J.S. & Warburton, T., 2002. Nodal high-order methods on unstructured grids. I. Time-domain solution of Maxwell's equations, *J. Comput. Phys.,* **181**(1), 186–221.

Hesthaven, J.S. & Warburton, T., 2007. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications,* Vol. 54, Springer.

Hu, L.-Z. & McMechan, G.A., 1987. Wave-field transformations of vertical seismic profiles, *Geophysics,* **52**(3), 307–321.

Kirby, R.M., Warburton, T.C., Lomtev, I. & Karniadakis, G.E., 2000. A discontinuous Galerkin spectral/hp method on hybrid grids, *Appl. Numer. Math.,* **33**(1), 393–405.

Klöckner, A., Warburton, T., Bridge, J. & Hesthaven, J.S., 2009. Nodal discontinuous Galerkin methods on graphics processors, *J. Comput. Phys.,* **228**(21), 7863–7882.

Klöckner, A., Warburton, T. & Hesthaven, J.S., 2012. Solving wave equations on unstructured geometries, in *GPU Computing Gems Jade Edition,* Applications of GPU Computing, pp. 225–242, ed. Hwu, W.W., Morgan Kaufmann.

Klöckner, A., Warburton, T. & Hesthaven, J.S., 2013. High-order discontinuous Galerkin methods by GPU metaprogramming, in *GPU Solutions to Multi-scale Problems in Science and Engineering,* Lecture Notes in Earth System Sciences, pp. 353–374, eds Yuen, D.A., Wang, L., Chi, X., Johnsson, L., Ge, W. & Shi, Y., Springer, Berlin Heidelberg.

Knibbe, H., Mulder, W.A., Oosterlee, C.W. & Vuik, C., 2014. Closing the performance gap between an iterative frequency-domain solver and an explicit time-domain scheme for 3D migration on parallel architectures, *Geophysics,* **79**(2), S47–S61.

Komatitsch, D. & Martin, R., 2007. An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation, *Geophysics,* **72**(5), SM155–SM167.

Komatitsch, D. & Tromp, J., 1999. Introduction to the spectral element method for three-dimensional seismic wave propagation, *Geophys. J. Int.,* **139**(3), 806–822.

Komatitsch, D. & Vilotte, J.-P., 1998. The spectral element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures, *Bull. seism. Soc. Am.,* **88**(2), 368–392.

Komatitsch, D., Erlebacher, G., Göddeke, D. & Michéa, D., 2010. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, *J. Comput. Phys.,* **229**(20), 7692–7714.

Kononov, A., Minisini, S., Zhebel, E. & Mulder, W.A., 2012. A 3D tetrahedral mesh generator for seismic problems, in *Proceedings of the 74th EAGE Conference & Exhibition,* p. B006.

Krebs, J.R., Collis, S.S., Downey, N.J., Ober, C.C., Overfelt, J.R., Smith, T.M., van Bloemen-Waanders, B.G. & Young, J.G., 2014. Full wave inversion using a spectral-element discontinuous Galerkin method, in *Proceedings of the 76th EAGE Conference & Exhibition*, Extended abstract.

Lailly, P., 1983. The seismic inverse problem as a sequence of before stack migration, in *Proceedings of the Conference on Inverse Scattering: Theory and Applications,* pp. 206–220, SIAM, Philadelphia (PA), USA.

LeVeque, R.J., 2002. *Finite Volume Methods for Hyperbolic Problems,* Vol. 31, Cambridge University Press.

Liu, F., Zhang, G., Morton, S.A. & Leveille, J.P., 2011. An effective imaging condition for reverse-time migration using wavefield decomposition, *Geophysics,* **76**(1), S29–S39.

Liu, G., Liu, Y., Ren, L. & Meng, X., 2013. 3D seismic reverse time migration on GPGPU, *Comput. Geosci.,* **59,** 17–23.

Liu, H., Li, B., Liu, H., Tong, X., Liu, Q., Wang, X. & Liu, W., 2012. The issues of prestack reverse time migration and solutions with graphic processing unit implementation, *Geophys. Prospect.,* **60**(5), 906–918.

Loewenthal, D., 1983. Reversed time migration in spatial frequency domain, *Geophysics,* **48**(5), 627–635.

McMechan, G.A., 1983. Migration by extrapolation of time-dependent boundary values, *Geophys. Prospect.,* **31**(3), 413–420.

Medina, D.S., St.-Cyr, A. & Warburton, T., 2014. *OCCA:* a unified approach to multi-threading languages, http://arxiv.org/abs/1403.0968.

Mercerat, E.D. & Glinsky, N., 2015. A nodal high-order discontinuous Galerkin method for elastic wave propagation in arbitrary heterogeneous media, *Geophys. J. Int.,* **201**(2), 1101–1118.

Métivier, L., Brossier, R., Labbé, S., Operto, S. & Virieux, J., 2014. A robust absorbing layer method for anisotropic seismic wave modeling, *J. Comput. Phys.,* **279,** 218–240.

Meza-Fajardo, K.C. & Papageorgiou, A.S., 2008. A nonconvolutional, split-field, perfectly matched layer for wave propagation in isotropic and anisotropic elastic media: stability analysis, *Bull. seism. Soc. Am.,* **98**(4), 1811–1836.

Michéa, D. & Komatitsch, D., 2010. Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards, *Geophys. J. Int.,* **182**(1), 389–402.

Minisini, S., Zhebel, E., Kononov, A. & Mulder, W.A., 2012. Efficiency comparison for continuous mass-lumped and discontinuous Galerkin finite-elements for 3D wave propagation, in *Proceedings of the 74th EAGE Conference & Exhibition,* p. A004.

Minisini, S., Zhebel, E., Kononov, A. & Mulder, W.A., 2013. Local time stepping with the discontinuous Galerkin method for wave propagation in 3D heterogeneous media, *Geophysics,* **78**(3), T67–T77.

Moczo, P., Kristek, J., Galis, M., Chaljub, E. & Etienne, V., 2011. 3-D finite-difference, finite-element, discontinuous-Galerkin and spectral-element schemes analysed for their accuracy with respect to P-wave to S-wave speed ratio, *Geophys. J. Int.,* **187**(3), 1645–1667.

Modave, A., St-Cyr, A., Warburton, T. & Mulder, W.A., 2015. Accelerated discontinuous Galerkin time-domain simulations for seismic wave propagation, in *Proceedings of the 77th EAGE Conference & Exhibition*, Madrid, Spain, 1–4 June 2015.

Mu, D., Chen, P. & Wang, L., 2013. Accelerating the discontinuous Galerkin method for seismic wave propagation simulations using multiple GPUs with CUDA and MPI, *Earthq. Sci.,* **26**(6), 377–393.

Nguyen, B.D. & McMechan, G.A., 2015. Five ways to avoid storing source wavefield snapshots in 2D elastic prestack reverse time migration, *Geophysics,* **80**(1), S1–S18.

Pellerin, J., Lévy, B., Caumon, G. & Botella, A., 2014. Automatic surface remeshing of 3D structural models at specified resolution: a method based on Voronoi diagrams, *Comput. Geosci.,* **62,** 103–116.

Sun, W. & Fu, L.-Y., 2013. Two effective approaches to reduce data storage in reverse time migration, *Comput. Geosci.,* **56,** 69–75.

Symes, W.W., 2007. Reverse time migration with optimal checkpointing, *Geophysics,* **72**(5), SM213–SM221.

Tago, J., Métivier, L. & Virieux, J., 2014. SMART layers: a simple and robust alternative to PML approaches for elastodynamics, *Geophys. J. Int.,* **199**(2), 700–706.

Tarantola, A., 1984. Inversion of seismic reflection data in the acoustic approximation, *Geophysics,* **49**(8), 1259–1266.

Virieux, J., Calandra, H. & Plessix, R.-É., 2011. A review of the spectral, pseudo-spectral, finite-difference and finite-element modelling techniques for geophysical imaging, *Geophys. Prospect.,* **59**(5), 794–813.

Warburton, T., 2006. An explicit construction of interpolation nodes on the simplex, *J. Eng. Math.,* **56**(3), 247–262.

Weiss, R.M. & Shragge, J., 2013. Solving 3D anisotropic elastic wave equations on parallel GPU devices, *Geophysics,* **78**(2), F7–F15.

Whitmore, N.D., 1983. Iterative depth migration by backward time propagation, in *Proceedings of the 53th SEG Annual Meeting,* pp. 382–385, Society of Exploration Geophysicists.

Wilcox, L.C., Stadler, G., Burstedde, C. & Ghattas, O., 2010. A high-order discontinuous Galerkin method for wave propagation through coupled elastic–acoustic media, *J. Comput. Phys.,* **229**(24), 9373–9396.

Yang, P., Gao, J. & Wang, B., 2014. RTM using effective boundary saving: A staggered grid GPU implementation, *Comput. Geosci.,* **68,** 64–72.

Zhebel, E., Minisini, S., Kononov, A. & Mulder, W.A., 2014. A comparison of continuous mass-lumped finite elements with finite differences for 3-D wave propagation, *Geophys. Prospect.,* **62**(5), 1111–1125.

APPENDIX A: DERIVATION OF THE NODAL DG SCHEME

The semi-discrete scheme is obtained from the formulation (11) and (12) by replacing the fields by their approximations (9) and (10), and by using the Lagrange polynomial functions as test functions. For each node n of element k, we then have

$$\sum_{n=1}^{N_p} M_{k,mn}\frac{dp_{k,n}}{dt} = -\sum_{i=1}^{3}\sum_{n=1}^{N_p} S_{k,i,mn}\,\rho_k c_k^2\, v_{i,k,n} - \sum_{f=1}^{N_f}\sum_{n'=1}^{N_{fp}} M_{k,f,mn'}\,P^p_{k,n'},$$

$$\sum_{n=1}^{N_p} M_{k,mn}\frac{dv_{j,k,n}}{dt} = -\sum_{n=1}^{N_p} S_{k,j,mn}\,\frac{1}{\rho_k}\,p_{k,n} - \sum_{f=1}^{N_f}\sum_{n'=1}^{N_{fp}} M_{k,f,mn'}\,P^{v_j}_{k,n'},$$

with $j = 1, 2, 3$. In this system, N_f is the number of faces by element, N_{fp} is the number of nodes by face, n' is the face node index and the corresponding node index in the list of all element nodes is denoted with $n(n')$. We have defined the penalties evaluated at each face node as

$$P^p_{k,f,n'} = \rho_k c_k^2 \left(\left(\sum_{i=1}^{3} n_{i,k,f}\left(v^\star_{i,k,n(n')} - v_{i,k,n(n')}\right)\right) - \tau_p \left(p^\star_{k,n(n')} - p_{k,n(n')}\right)\right),$$

$$P^{v_j}_{k,f,n'} = \frac{n_{j,k,f}}{\rho_k}\left(\left(p^\star_{k,n(n')} - p_{k,n(n')}\right) - \tau_{\mathbf{v}} \sum_{i=1}^{3} n_{i,k,f}\left(v^\star_{i,k,n(n')} - v_{i,k,n(n')}\right)\right),$$

where $n_{i,k,f}$ is the ith Cartesian component of the outward unit normal of the face $\partial \mathsf{D}_{k,f}$. The star subscript denotes the nodal values on the side of the neighboring element. The components of the so-called local mass, local stiffness and local face mass matrices are respectively

$$M_{k,mn} = \int_{\mathsf{D}_k} \ell_{k,m}\ell_{k,n}\,d\mathbf{x},$$

$$S_{k,i,mn} = \int_{\mathsf{D}_k} \ell_{k,m}\frac{d\ell_{k,n}}{dx_i}\,d\mathbf{x},$$

$$M_{k,f,mn'} = \int_{\partial\mathsf{D}_{k,f}} \ell_{k,m}\ell_{k,n(n')}\,d\mathbf{x}.$$

Inverting the local mass matrix, the system can then be written as

$$\frac{d\mathbf{q}^p_k}{dt} = -\rho_k c_k^2 \sum_{i=1}^{3} \mathbf{M}_k^{-1}\mathbf{S}_{k,i}\,\mathbf{q}^{v_i}_k - \sum_{f=1}^{N_f} \mathbf{M}_k^{-1}\mathbf{M}_{k,f}\,\mathbf{P}^p_{k,f},$$

$$\frac{d\mathbf{q}^{v_j}_k}{dt} = -\frac{1}{\rho_k}\,\mathbf{M}_k^{-1}\mathbf{S}_{k,j}\,\mathbf{q}^p_k - \sum_{f=1}^{N_f} \mathbf{M}_k^{-1}\mathbf{M}_{k,f}\,\mathbf{P}^{v_j}_{k,f},$$

where $\mathbf{q}^p_k$ and $\mathbf{q}^{v_j}_k$ contain the discrete unknowns associated to each field for all nodes of element k. The vectors $\mathbf{P}^p_{k,f}$ and $\mathbf{P}^{v_j}_{k,f}$ contain the penalties for all face nodes of face f. $\mathbf{M}_k$ and $\mathbf{S}_{k,i}$ are $N_p \times N_p$ matrices, while $\mathbf{M}_{k,f}$ is a $N_p \times N_{fp}$ matrix.

Since an affine transformation connects each element to a reference element, the local matrices can be expressed in terms of reference matrices that are the same for each element, combined with scaling and linear combinations. Denoting the transformation from the reference cell I to each cell D_k with $\boldsymbol{\Psi}_k : \mathbf{r} \in \mathsf{I} \to \mathbf{x} \in \mathsf{D}_k$, we have

$$\mathbf{M}_k = J_k\,\mathbf{M}_{\mathsf{I}},$$

$$\mathbf{S}_{k,i} = J_k \sum_{j=1}^{3} \frac{\partial \Psi_{k,i}}{\partial r_j}\mathbf{S}_{\mathsf{I},j},$$

where $J_k = \det(\partial\boldsymbol{\Psi}_k/\partial\mathbf{r})$ is the Jacobian of the transformation. Similarly, each face is related to a reference face with an *ad hoc* affine transformation. This permits to express all local face mass matrices in term of a reference two-dimensional mass matrix, involving the Jacobian of this transformation $J_{k,f}$ as well as a $N_p \times N_{fp}$ matrix which connects the face node indices with the node indices in the list of all nodes.

For each element, the system of equations can finally be written in the convenient form

$$\frac{d\mathbf{q}_k^p}{dt} = -\rho_k c_k^2 \sum_{i=1}^{3}\sum_{j=1}^{3} \frac{\partial \Psi_{k,i}}{\partial r_j} \mathbf{D}_j \mathbf{q}_k^{v_i} - \sum_{f=1}^{N_f} \frac{J_{k,f}}{J_k} \mathbf{L}_f \mathbf{P}_{k,f}^{p},$$

$$\frac{d\mathbf{q}_k^{v_i}}{dt} = -\frac{1}{\rho_k} \sum_{j=1}^{3} \frac{\partial \Psi_{k,i}}{\partial r_j} \mathbf{D}_j \mathbf{q}_k^{p} - \sum_{f=1}^{N_f} \frac{J_{k,f}}{J_k} \mathbf{L}_f \mathbf{P}_{k,f}^{v_i},$$

where $\mathbf{D}_j$ are the differentiation matrices and $\mathbf{L}_f$ are the lifting matrices for the reference element. Defining the geometric factors

$$g_{k,i,j}^{\text{vol}} = \frac{\partial \Psi_{k,i}}{\partial r_j},$$

$$g_{k,f}^{\text{sur}} = \frac{J_{k,f}}{J_k},$$

one obtains eqs (13) and (14).