Scaling through abstractions – high-performance vectorial wave simulations for seismic inversion with Devito

1st Mathias Louboutin
Georgia Institute of Technology
Atlanta, GA
mlouboutin3@gatech.edu

2nd Fabio Luporini
Devito Codes
London, UK
fabio@devitocodes.com

3rd Rhodri Nelson
Imperial College London
London, UK
rhodri.nelson@imperial.ac.uk

4th Philipp Witte
Georgia Institute of Technology
Atlanta, GA
pwitte3@gatech.edu

5th George Bisbas
Imperial College London
London, UK
g.bisbas18@imperial.ac.uk

6th Jan Thorbecke
TU-Delft
Delft, NL
J.W.Thorbecke@tudelft.nl

7th Felix J. Herrmann
Georgia Institute of Technology
Atlanta, GA
felix.herrmann@gatech.edu

8th Gerard J. Gorman
Imperial College London
London, UK
g.gorman@imperial.ac.uk

***Abstract*— Devito is an open-source Python project based on domain-specific language and compiler technology. Driven by the requirements of rapid HPC applications development in exploration seismology, the language and compiler have evolved significantly since inception. Sophisticated boundary conditions, tensor contractions, sparse operations and features such as staggered grids and sub-domains are all supported; operators of essentially arbitrary complexity can be generated. To accommodate this flexibility whilst ensuring performance, data dependency analysis is utilized to schedule loops and detect computational-properties such as parallelism. In this article, the generation and simulation of MPI-parallel propagators (along with their adjoints) for the pseudo-acoustic wave-equation in tilted transverse isotropic media and the elastic wave-equation are presented. Simulations are carried out on industry scale synthetic models in a HPC Cloud system and reach a performance of 28TFLOP/s, hence demonstrating Devito's suitability for production-grade seismic inversion problems.**

I. Introduction

Seismic imaging methods such Reverse Time Migration (RTM) and Full-waveform inversion (FWI) rely on the numerical solution of an underlying system of partial differential equations (PDEs), most commonly some manifestation of the wave-equation. In the context of FWI, the finite-difference (FDM) and the spectral-element (SEM) methods are most frequently used to solve the wave-equation, with FDM methods dominating within the seismic exploration community [1]. Various forms of the wave-equation and modelling strategies for FWI are detailed in [2].

Despite the theory of FWI dating back to the 1980s [3], among the first successful expositions on real 3D data was presented in [4]. Other studies utilizing FDM within the FWI workflow include [5], [6]. The aforementioned studies approximate the underlying physics via the acoustic wave-equation; higher fidelity models solving the non-isotropic elastic wave-equation have been developed in, e.g., [7]–[11]. Owing to the flexibility of the mathematical discretizations that can be utilized, along with the ability to describe problems on complex meshes, there has also been a great deal of interest in utilizing SEM to solve inversion problems [12], [13]. The recent study [14] presents an efficient SEM based inversion scheme using a viscoelastic formulation of the wave-equation.

It is generally accepted that the PDE solver utilized within an inversion workflow must satisfy the following criteria [15]: - Efficient for multiple-source modelling - The memory requirement of the modelling - The ability of a parallel algorithm to use an increasing number of processors - Ability of the method to process models of arbitrary levels of heterogeneity - Reduce the nonlinearity of FWI - Feasibility of the extension of the modelling approach to more realistic physical descriptions of the media.

It is with these specifications in mind that Devito, a symbolic domain specific language (DSL) and compiler for the automatic generation of finite-difference stencils, has been developed. Originally deigned to accelerate research and development in exploration geophysics, the high-level interface, previously described in detail in [16], is built on top of `SymPy` [17] and is inspired by the underlying mathematical formulations and other DSLs such as FEniCS [18] and Firedrake [19]. This interface allows the user to formulate wave-equations, and more generally time-dependent PDEs in a simple and mathematically coherent way. The Devito compiler then automatically generates finite-difference stencils from these mathematical expressions. One of the main advantages of Devito over other finite-difference DSLs is that generic expressions such as sparse operations (i.e. point source injection or localized measurements) are fully supported and expressible in a high-level fashion. The second component of Devito is its compiler (c.f [20]) that generates highly optimized C code. The generated code is then compiled at runtime for the hardware at hand.

Previous work focused on the DSL and compiler to highlight the potential application and use cases of Devito. Here, we present a series of extensions and applications to large-scale three-dimensional problem sizes as encountered in exploration geophysics. These experiments are carried out in Cloud-based HPC systems and include elastic forward modelling using distributed-memory parallelism and imaging based on the tilted transverse isotropic (TTI) wave-equation ([21]–[25]). These proof of concepts highlight two critical features: first, the ability of the symbolic interface and the compiler to translate to large-scale adjoint-based inversion problems that require massive compute (since thousands of PDEs are solved) as well as large amounts of memory (since the adjoint state method requires the forward model to be saved in memory). Secondly, through the elastic modelling example, we demonstrate that Devito now fully supports and automates vectorial and second order tensorial staggered-grid finite-differences with the same high-level API previously presented for scalar fields defined on cartesian grids.

This paper is organized as follows: first, we provide a brief overview of Devito and its symbolic API and present the distributed memory implementation that allows large-scale modelling and inversion by means of domain decomposition. We then provide a brief comparison with a state of the art hand-coded wave propagator to validate the performance previously benchmarked with the roofline model ([16], [20], [26], [27]). Before concluding, results from the Cloud-based experiments discussed above are presented, highlighting the vectorial and tensorial capabilities of Devito.

II. Overview of Devito

Devito [16] provides a functional language built on top of `SymPy` [17] to symbolically express PDEs at a mathematical level and implements automatic discretization with the finite-difference method. The language is by design flexible enough to express other types of non finite-difference operators, such as interpolation and tensor contractions, that are inherent to measurement-based inverse problems. Several additional features are supported, among which are staggered grids, sub-domains, and stencils with custom coefficients. The last major building block of a solid PDE solver are the boundary conditions which for finite-difference methods are notoriously diverse and often complicated. The system is, however, sufficiently general to express them through a composition of core mechanisms. For example, free surface and perfectly-matched layers (PMLs) boundary conditions can be expressed as equations – just like any other PDE equations – over a suitable sub-domain.

It is the job of the Devito compiler to translate the symbolic specification into C code. The lowering of the input language to C consists of several compilation passes, some of which introduce performance optimizations that are the key to rapid code. Next to classic stencil optimizations (e.g., cache blocking, alignment, SIMD and OpenMP parallelism), Devito applies a series of FLOP-reducing transformations as well as aggressive loop fusion. For a complete treatment, the interested reader should refer to [20].

A. Symbolic language and compilation

In this section we illustrate the Devito language by demonstrating an implementation of the acoustic wave-equation in isotropic media

$$\begin{cases} m\frac{d^2u(t,x)}{dt^2} - \Delta u(t,x) = \delta(xs)q(t) \\ u(0,.) = \frac{du(t,x)}{dt}(0,.) = 0 \\ d(t,.) = u(t,xr). \end{cases} \quad (1)$$

The core of the Devito symbolic API consists of three classes:

- `Grid`, a representation of the discretized model.
- `(Time)Function`, a representation of spatially (and time-) varying variables defined on a `Grid` object.
- `Sparse(Time)Function` a representation of (time-varying) point objects on a `Grid` object, generally unaligned with respect to the grid points, hence called "sparse".

A `Grid` represents a discretized finite n-dimensional space and is created as follows

```
from devito import Grid
grid = Grid(shape=(nx, ny, nz),
            extent=(ext_x, ext_y, ext_z),
            origin=(o_x, o_y, o_z))
```

Listing 1: Grid creation

where `(nx, ny, nz)` are the number of grid points in each direction, `(ext_x, ext_y, ext_z)` is the physical extent of the domain in physical units (i.e `m`) and `(o_x, o_y, o_z)` is the origin of the domain in the same physical units. The object `grid` contains all the information related to the discretization such as the grid spacing. We use `grid` to create the symbolic objects that will be used to express the wave-equation. First, we define a spatially varying model parameter `m` and a time-space varying field `u`

```
from devito import Function, TimeFunction
m = Function(name="m", grid=grid,
             space_order=so)
u = TimeFunction(name="u", grid=grid,
                 space_order=so,
                 time_order=to)
```

Listing 2: Function definition

where `so` is the order of the spatial discretization and `to` the time discretization order used when generating the finite-difference stencil. Next, we define point source objects (`src`) located at the physical coordinates x_r, and the receiver (measurement) objects (`d`) located at the physical locations x_r

```
from devito import SparseTimeFunction
s = SparseTimeFunction(name="src",
```

```
                          grid=grid, npoint=1,
                          coordinates=x_s)
d = SparseTimeFunction(name="d", grid=grid,
                          npoint=1, nt=nt,
                          coordinates=x_r)
```

Listing 3: SparseFunction definition

The source term is handled separately from the PDE as a point-wise operation called injection, while measurement is handled via interpolation. By default, Devito initializes all `Function` data to 0, and thus automatically satisfies the zero Dirichlet condition at `t=0`. The isotropic acoustic wave-equation can then be implemented in Devito as follows

```
from devito import solve, Eq, Operator
eq = m * u.dt2 - u.laplace
update = Eq(u.forward, solve(eq, u.forward))
src_eqns = s.inject(u.forward, expr=s*dt**2/m)
d_eqns = d.interpolate(u)
```

Listing 4: Wave-equation symbolic definition

To trigger compilation one needs to pass the constructed equations to an `Operator`.

```
from devito import Operator
op = Operator(update + src_eqns + d_eqns)
```

Listing 5: Operator creation

The first compilation pass processes equations individually. The equations are lowered to an enriched representation, while the finite-difference constructs (e.g., derivatives) are translated into actual arithmetic operations. Subsequently, data dependency analysis is used to compute a performance-optimized topological ordering of the input equations (e.g., to maximize the likelihood of loop fusion) and to group them into so called "clusters". Basically, a cluster will eventually be a loop nest in the generated code, and consecutive clusters may share some outer loops. The ordered sequence of clusters undergoes several optimization passes, including cache blocking and FLOP-reducing transformations. It is then further lowered into an abstract syntax tree, and it is on such representation that parallelism is introduced (SIMD, shared-memory, MPI). Finally, all remaining low-level aspects of code generation are handled, among which the most relevant is data management (e.g., definition of variables, transfers between host and device).

The output of the Devito compiler for the running example used in this section is available at CodeSample in `acou-so8.c`.

B. Distributed-memory parallelism

We here provide a succinct description of distributed-memory parallelism in Devito; the interested reader should refer to the MPI tutorial at mpi-notebook for thorough explanations and practical examples.

Devito implements distributed-memory parallelism on top of MPI. The design is such that users can almost entirely abstract away from it and reuse non-distributed code as is. Given any Devito code, just running it as

```
DEVITO_MPI=1 mpirun -n X python ...
```

triggers the generation of code with routines for halo exchanges. The routines are scheduled at a suitable depth in the various loop nests thanks to data dependency analysis. The following optimizations are automatically applied:

- redundant halo exchanges are detected and dropped;
- computation/communication overlap, with prodding of the asynchronous progress engine by a designated thread through repeated calls to `MPI_Test`;
- a halo exchange is placed as far as possible from where it is needed to maximize computation/communication overlap;
- data packing and unpacking is threaded.

Domain decomposition occurs in Python upon creation of a `Grid` object. Exploiting the MPI Cartesian topology abstraction, Devito logically splits a grid based on the number of available MPI processes (noting that users are given an "escape hatch" to override Devito's default decomposition strategy). `Function` and `TimeFunction` objects inherit the `Grid` decomposition. For `SparseFunction` objects the approach is different. Since a `SparseFunction` represents a sparse set of points, Devito looks at the physical coordinates of each point and, based on the `Grid` decomposition, schedules the logical ownership to an MPI rank. If a sparse point lies along the boundary of two or more MPI ranks, then it is duplicated in each of these ranks to be accessible by all neighboring processes. Eventually, a duplicated point may be redundantly computed by multiple processes, but any redundant increments will be discarded.

When accessing or manipulating data in a Devito code, users have the illusion to be working with classic NumPy arrays, while underneath they are actually distributed. All manner of NumPy indexing schemes (basic, slicing, etc.) are supported. In the implementation, proper global-to-local and local-to-global index conversion routines are used to propagate a read/write access to the impacted subset of ranks. For example, consider the array

```
A = [[ 1,  2,  3,  4],
     [ 5,  6,  7,  8],
     [ 9, 10, 11, 12],
     [13, 14, 15, 16]])
```

which is distributed across 4 ranks such that `rank 0` contains the elements reading `1, 2, 5, 6`, `rank 1` the elements `3, 4, 7, 8`, `rank 2` the elements `9, 10, 13, 14` and `rank 3` the elements `11, 12, 15, 16`. The slicing operation `A[::-1, ::-1]` will then return

```
    [[ 16, 15, 14, 13],
     [ 12, 11, 10,  9],
     [  8,  7,  6,  5],
     [  4,  3,  2,  1]])
```

such that now `rank 0` contains the elements `16, 15, 12, 11` and so forth.

Finally, we remark that while providing abstractions for distributed data manipulation, Devito does not natively support any mechanisms for parallel I/O. However, the distributed NumPy arrays along with the ability to seamlessly transfer any desired slice of data between ranks provides a generic and flexible infrastructure for the implementation of any form of parallel I/O (e.g., see [28]).

III. Industry-scale 3D seismic imaging in anisotropic media

One of the main applications of seismic finite-difference modelling in exploration geophysics is reverse-time migration (RTM), a wave-equation based seismic imaging technique. Real-world seismic imaging presents a number of challenges that make applying this method to industry-scale problem sizes difficult. Firstly, RTM requires an accurate representation of the underlying physics via sophisticated wave-equations such as the tilted-transverse isotropic (TTI) wave-equation, for which both forward and adjoint implementations must to be provided. Secondly, wave-equations must be solved for a large number of independent experiments, where each individual PDE solve is itself expensive in terms of FLOPs and memory usage. For certain workloads, limited domain decomposition, which balances the domain size and the number of independent experiments, as well as checkpointing techniques must be adopted. In the following sections, we describe an industry-scale seismic imaging problem that poses all the aforementioned challenges, its implementation with Devito, and the results of an experiment carried out on the Azure Cloud using a synthetic data set.

A. Anisotropic wave-equation

In our seismic imaging case study, we use an anisotropic representation of the physics called Tilted Transverse Isotropic (TTI) modelling [22]. This representation for wave motion is one of the most widely used in exploration geophysics since it captures the leading order kinematics and dynamics of acoustic wave motion in highly heterogeneous elastic media where the medium properties vary more rapidly in the direction perpendicular to sedimentary strata [24], [25], [29]–[42]. The TTI wave-equation is an acoustic, low dimensional (4 parameters, 2 wavefields) simplification of the 21 parameter and 12 wavefields tensorial equations of motions [43]. This simplified representation is parametrized by the Thomsen parameters $\epsilon(x), \delta(x)$ that relate to the global (propagation over many wavelengths) difference in propagation speed in the vertical and horizontal directions, and the tilt and azimuth angles $\theta(x), \phi(x)$ that define the rotation of the vertical and horizontal axes around the cartesian directions. However, unlike the scalar isotropic acoustic wave-equation itself, the TTI wave-equation is extremely computationally costly to solve and it is also not self-adjoint as shown in [25].

The main complexity of the TTI wave-equation is that the rotation of the symmetry axis of the physics leads to rotated second-order finite-difference stencils. In order to ensure numerical stability, these rotated finite-difference operators are designed to be self-adjoint (c.f. [24], [41]). For example, we define the rotated second order derivative with respect to x as:

$$
\begin{aligned}
G_{\bar{x}\bar{x}} &= D_{\bar{x}}^T D_{\bar{x}} \\
D_{\bar{x}} &= \cos(\theta)\cos(\phi)\frac{\mathrm{d}}{\mathrm{d}x} + \cos(\theta)\sin(\phi)\frac{\mathrm{d}}{\mathrm{d}y} - \sin(\theta)\frac{\mathrm{d}}{\mathrm{d}z}.
\end{aligned} \tag{2}
$$

We enable the simple expression of these complicated stencils in Devito as finite-difference shortcuts such as `u.dx` where `u` is a `Function`. Such shortcuts are enabled not only for the basic types but for generic composite expressions, for example `(u + v.dx).dy`. As a consequence, the rotated derivative defined in 2 is implemented with Devito in two lines as:

```
dx_u = cos(theta) * cos(phi) * u.dx +
       cos(theta) * sin(phi) * u.dy -
       sin(theta) * u.dz
dxx_u = (cos(theta) * cos(phi) * dx_u).dx.T +
        (cos(theta) * sin(phi) * dx_u).dy.T -
        (sin(theta) * dx_u).dz.T
```

Listing 6: Rotated finite-difference with symbolic shortcuts

Note that while the adjoint of the finite-difference stencil is enabled via the standard Python `.T` shortcut, the expression needs to be reordered by hand since the tilt and azymuth angles are spatially dependent and require to be inside the second pass of first-order derivative. We can see from these simple two lines that the rotated stencil involves all second-order derivatives (`.dx.dx`, `.dy.dy` and `.dz.dz`) and all second-order cross-derivatives (`dx.dy`, `.dx.dz` and `.dy.dz`) which leads to a denser stencil support and higher computational complexity (c.f. [27]). For illustrative purposes, the complete generated code for tti modelling with and without MPI is made available at CodeSample in `tti-so8-unoptimized.c`, `tti-so8.c` and `tti-so8-mpi.c`.

Owing to the very high number of floating-point operations (FLOPs) needed per grid point for the weighted rotated Laplacian, this anisotropic wave-equation is extremely challenging to implement. As we show in Table I, and previously analysed in [27], the computational cost with high-order finite-difference is in the order of thousands of FLOPs per grid point without optimizations. The version without FLOP-reducing optimizations is a direct translation of the discretized operators into stencil expressions (see `tti-so8-unoptimized.c`). The version with optimizations employs transformations such as common sub-expressions elimination, factorization, and cross-iteration redundancy elimination – the latter being key in removing redundancies introduced by mixed derivatives. Implementing all of these techniques manually is inherently difficult and laborious. Further, to obtain the desired performance improvements it is necessary to orchestrate them with aggressive loop fusion (for data locality), tiling (for

spatial order	w/o optimizations	w/ optimizations
4	501	95
8	539	102
12	1613	160
16	5489	276

TABLE I: Per-grid-point FLOPs of the finite-difference TTI wave-equation stencil with different spatial discretization orders.

data locality and tensor temporaries), and potentially ad-hoc vectorization strategies (if rotating registers are used). While an explanation of the optimization strategies employed by Devito is beyond the scope of this paper (see [20] for details), what is emphasized here is that users can easily take full advantage of these optimizations without needed to concern themselves with the details.

It is evident that developing an appropriate solver for the TTI wave-equation, an endeavor involving complicated physics, mathematics, and engineering, is exceptionally time-consuming and can lead to thousands of lines of code even for a single choice of discretization. Verification of the results is no less complicated, any minor error is effectively untrackable and any change to the finite-difference scheme or to the time-stepping algorithm is difficult to achieve without substantial re-coding. Another complication stems from the fact that practitioners of seismic inversion are often geoscientists, not computer scientists/programmers. Low level implementations from non-specialists can often lead to poorly performing code. However, if research codes are passed to specialists in the domain of low level code optimization they often lack the necessary geophysical domain knowledge, resulting in code that may lack a key feature required by the geoscientist. Neither situation is conducive to addressing the complexities that come with implementing codes based on the latest geophysical insights in tandem with those from high-performance computing. With Devito on the other hand, both the forward and adjoint equations can be implemented in a few lines of Python code as illustrated with the rotated operator in Listing 6. The low level optimization element of the development is then taken care of under the hood by the Devito compiler.

The simulation of wave motion is only one aspect of solving problems in seismology. During wave-equation based imaging, it is also required to compute sensitivities (gradient) with respect to the quantities of interest. This requirement imposes additional constraints on the design and implementation of model codes as outlined in [21]. Along with several factors, such as fast setup time, we focused on correct and testable implementations for the adjoint wave-equation and the gradient (action of the adjoint Jacobian) [25], [44]; exactness being a mandatory requirement of gradient based iterative optimization algorithms.

B. 3D Imaging example on Azure

We now demonstrate the scalability of Devito to real-world applications by imaging an industry-scale three-dimensional TTI subsurface model. This imaging was carried out in the Cloud on Azure and takes advantage of recent work to port conventional cluster code to the Cloud using a serverless approach. The serverless implementation is detailed in [45], [46] where the steps to run computationally and financially efficient HPC workloads in the Cloud are described. This imaging project, in collaboration with Azure, demonstrates the scalability and robustness of Devito to large scale wave-equation based inverse problems in combination with a cost-effective serverless implementation of seismic imaging in the Cloud. In this example, we imaged a synthetic three-dimensional anisotropic subsurface model that mimics a realistic industry size problem with a realistic representation of the physics (TTI). The physical size of the problem is `10kmx10kmx2.8km` discretized on a `12.5m` grid with absorbing layers of width 40 grid points on each side leading to `881x881x371` computational grid points ($\approx$ 300 Million grid points). The final image is the sum of 1500 single-source images: 100 single-source images were computed in parallel on the 200 nodes available using two nodes per source experiment.

Computational performance

We briefly describe the computational setup and the performance achieved for this anisotropic imaging problem. Due to time constraints, and because the resources we were given access to for this proof of concept with Azure were somewhat limited, we did not have access to Infiniband-enabled virtual machines (VM). This experiment was carried out on `Standard_E64_v3` and `Standard_E64s_v3` nodes which, while not HPC VM nodes, are memory optimized thus allowing to the wavefield to be stored in memory for imaging (TTI adjoint state gradient [21], [25]). These VMs are Intel® Broadwell E5-2673 v4 2.3GH that are dual socket, 32 physical cores (with hyperthreading enabled) and 432Gb of DRAM. The overall inversion involved computing the image for 1500 source positions, i.e. solving 1500 forward and 1500 adjoint TTI wave-equations. A single image required, in single precision, 600Gb of memory. Two VMs were used per source and MPI set with one rank per socket (4 MPI ranks per source) and 100 sources were imaged in parallel. The performance achieved was as follows:

- 140 GFLOP/s per VM;
- 280 GFLOP/s per source;
- 28 TFLOP/s for all 100 running sources;
- 110min runtime per source (forward + adjoint + image computation).

We comment that if more resources were available, and because the imaging problem is embarrassingly parallel over sources and can scale arbitrarily,the imaging of all of the 1500 sources in parallel could have been attempted, which theoretically leads to a performance of 0.4PFLOP/s.

How performance was measured

The execution time is computed through Python-level timers

prefixed by an MPI barrier. The floating-point operations are counted once all of the symbolic FLOP-reducing transformations have been performed during compilation. Devito uses an in-house estimate of cost, rather than `SymPy`'s estimate, to take care of some low-level intricacies. For example, Devito's estimate ignores the cost of integer arithmetic used for offset indexing into multi-dimensional arrays. To calculate the total number of FLOPs performed, Devito multiplies the floating-point operations calculated at compile time by the size of the iteration space, and it does that at the granularity of individual expressions. Thanks to aggressive code motion, the amount of innermost-loop-invariant sub-expressions in an `Operator` is typically negligible and therefore the Devito estimate does not suffer from this issue, or at least not, to the best of our knowledge, in a tangible way. The Devito-reported GFLOP/s were also checked against those produced by Intel Advisor on several single-node experiments: the differences – typically Devito underestimating the achieved performance – were always at most in the order of units, and therefore negligible.

Imaging result

The subsurface velocity model used in this study is an artificial anisotropic model that is designed and built combining two broadly known and used open-source SEG/EAGE acoustic velocity models that each come with realistic geophysical imaging challenges such as sub-salt imaging. The anisotropy parameters are derived from a smoothed version of the velocity while the tilt angles were derived from a combination of the smooth velocity models and vertical and horizontal edge detection. The final seismic image of the subsurface model is displayed in Figure 1 and highlights the fact that 3D seismic imaging based on a serverless approach and automatic code generation is feasible and provides good results.

[45] describes the serverless implementation of seismic inverse problems in detail, including iterative algorithms for least-square minimization problems (LSRTM). The 3D anisotropic imaging results were presented as part of a keynote presentation at the EAGE HPC workshop in October 2019 [47] and at the Rice O&G HPC workshop [48] in which the focus was on the serverless implementation of seismic inversion algorithms in the Cloud. This work illustrates the flexibility and portability of Devito: we were able to easily port a code developed and tested on local hardware to the Cloud, with only minor adjustments. Further, note that this experiment included the porting of MPI-based code for domain decomposition developed on desktop computers to the Cloud. Our experiments are reproducible using the instructions in a public repository AzureTTI, which contains, among the other things, the Dockerfiles and Azure batch-shipyard setup. This example can also be easily run on a traditional HPC cluster environment using, for example, JUDI [28] or Dask [49] for parallelization over sources.

IV. Elastic modelling

While the subsurface image obtained in section III-B utilized anisotropic propagators capable of mimicking intricate physics, in order to model both the wave kinematics and amplitudes correctly, elastic propagators are required. These propagators are, for example, extremely important in global seismology since shear surface waves (which are ignored in acoustic models) are the most hazardous. In this section, we exploit the tensor algebra language introduced in Devito v4.0 to express an elastic model with compact and elegant notation.

The isotropic elastic wave-equation, parametrized by the so-called Lamé parameters λ, μ and the density ρ reads:

$$\begin{aligned} \frac{1}{\rho}\frac{dv}{dt} &= \nabla.\tau \\ \frac{d\tau}{dt} &= \lambda \mathrm{tr}(\nabla v)\mathbf{I} + \mu(\nabla v + (\nabla v)^T) \end{aligned} \tag{3}$$

where v is a vector valued function with one component per cartesian direction:

$$v = \begin{bmatrix} v_x(t,x,y) \\ v_y(t,x,y)) \end{bmatrix} \tag{4}$$

and the stress τ is a symmetric second-order tensor-valued function:

$$\tau = \begin{bmatrix} \tau_{xx}(t,x,y) & \tau_{xy}(t,x,y) \\ \tau_{xy}(t,x,y) & \tau_{yy}(t,x,y) \end{bmatrix}. \tag{5}$$

The discretization of (3) and (5) requires five equations in two dimensions (two equations for the particle velocity and three for the stress) and nine equations in three dimensions (three for the particle velocity and six for the stress). However, the mathematical definition only require two coupled vector/tensor-valued equations for any number of dimensions.

A. Tensor algebra language

We have augmented the Devito language with tensorial objects to enable straightforward and mathematically rigorous definitions of high-dimensional PDEs, such as the elastic wave-equation defined in (3). This implementation was inspired by [50], a functional language for finite element methods.

The extended Devito language introduces two new types, `VectorFunction`/`VectorTimeFunction` for vectorial objects such as the particle velocity, and `TensorFunction`/`TensorTimeFunction` for second-order tensor objects (matrices) such as the stress. These new objects are constructed in the same manner as scalar `Function` objects. They also automatically implement staggered grid and staggered finite-differences with the possibility of half-node averaging. Each component of a tensorial object – a (scalar) Devito `Function` – is accessible via conventional vector notation (e.g. `v[0]`, `t[0,1]`).

With this extended language, the elastic wave-equation defined in (3) and (5) can be expressed in only four lines of code:

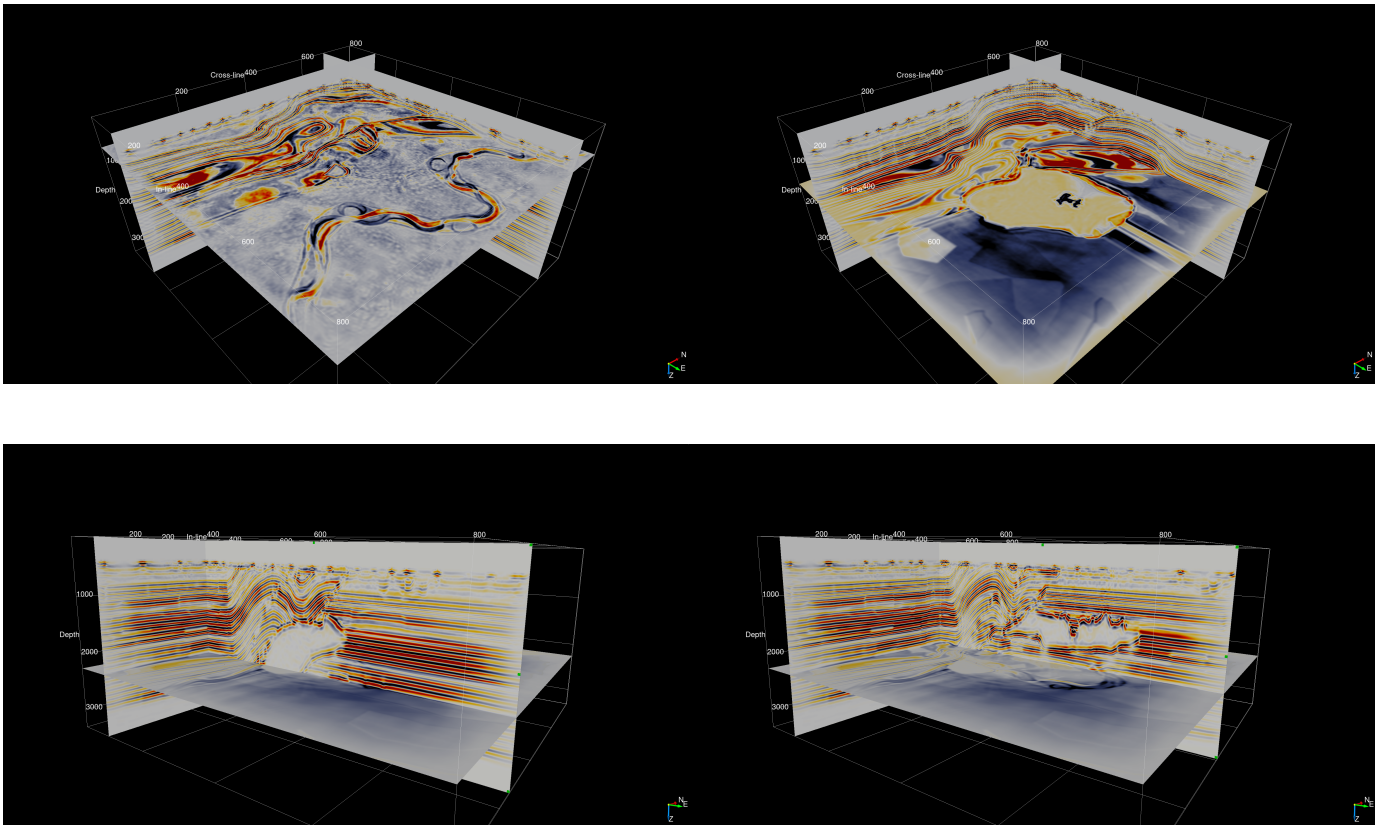

Fig. 1: 3D TTI imaging on a custom made model.

```
from devito import VectorTimeFunction,
                   TensorTimeFunction
v = VectorTimeFunction(name="v", grid=grid,
                       space_order=so,
                       time_order=1)
tau = TensorTimeFunction(name="t", grid=grid,
                         space_order=so,
                         time_order=1)
u_v = Eq(v.forward,
         damp * (v + s/rho*div(tau)))
u_t = Eq(tau.forward,
         damp * (tau + s * (l * diag(div(v.⟲
             forward)) + mu * (grad(v.forward) ⟲
             + grad(v.forward).T))))
```

Listing 7: Vectorial elastic wave-equation with Devito

The `SymPy` expressions created by these commands can be displayed with `sympy.pprint` as shown in Figure 2. This representation reflects perfectly the mathematics while maintaining computational portability and efficiency through the Devito compiler. The complete generated code for the elastic wave-equation with and without MPI is made available at CodeSample in `elastic-so12.c` and `elastic-so12-mpi.c`.

B. 2D example

To demonstrate the efficacy of the elastic implementation outlined above we utilized a broadly recognized 2D synthetic model, the elastic Marmousi-ii [51], [52] model. The wavefields are shown on Figure 3 and its corresponding elastic shot records are displayed in Figure 4. These two figures show that the wavefield is, as expected, purely acoustic in the water layer ($\tau_{xy} = 0$) and transitions correctly at the ocean bottom to an elastic wavefield. We can also clearly see the shear wave-front in the subsurface (at a depth of approximately 1km). Figures 3 and 4 demonstrate that this high-level Devito implementation of the elastic wave-equation is effective and accurate. Importantly, in constructing this model within the Devito DSL framework, computational technicalities such as the staggered grid analysis are abstracted away. We note that the shot records displayed in Figure 4 match the original data generated by the creator of this elastic model ([52]).

C. 3D proof of concept

Finally, three dimensional elastic data was modelled in the Cloud to demonstrate the scalability of Devito to cluster-size problems. The model used in this experiment mimics a reference model in geophysics known as the SEAM

Particle velocity update u_v :

$$\begin{bmatrix} \mathrm{v_x}\left(t+dt, x+\frac{h_x}{2}, y\right) \\ \mathrm{v_y}\left(t+dt, x, y+\frac{h_y}{2}\right) \end{bmatrix} = \begin{bmatrix} \left(dt\left(\frac{\partial}{\partial x}\,\mathrm{t_{xx}}\left(t,x,y\right) + \frac{\partial}{\partial y}\,\mathrm{t_{xy}}\left(t, x+\frac{h_x}{2}, y+\frac{h_y}{2}\right)\right)\mathrm{irho}\left(x,y\right) + \mathrm{v_x}\left(t, x+\frac{h_x}{2}, y\right)\right)\mathrm{damp}\left(x,y\right) \\ \left(dt\left(\frac{\partial}{\partial x}\,\mathrm{t_{xy}}\left(t, x+\frac{h_x}{2}, y+\frac{h_y}{2}\right) + \frac{\partial}{\partial y}\,\mathrm{t_{yy}}\left(t,x,y\right)\right)\mathrm{irho}\left(x,y\right) + \mathrm{v_y}\left(t, x, y+\frac{h_y}{2}\right)\right)\mathrm{damp}\left(x,y\right) \end{bmatrix}$$

Fig. 2: Particle velocity update stencil. The stress update is omitted for readability (the equation does not fit into a single page). However, it can be found in a Devito tutorial on elastic modelling.

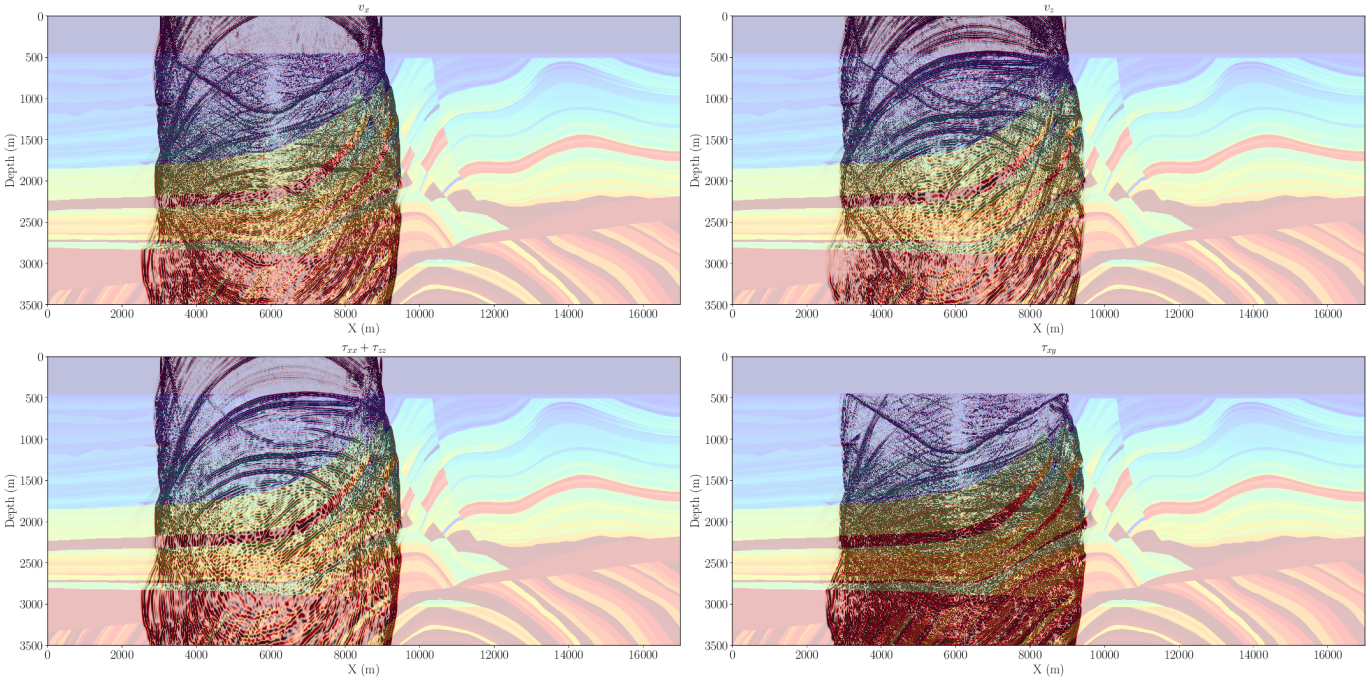

Fig. 3: Particle velocities and stress at time $t = 3$s for a 10m deep source and `x=5\text{km}` in the marmousi-ii model.

model [53], a three dimensional extreme-scale synthetic representation of a subsurface. The physical dimensions of the model are `45kmx35kmx15km` discretized with a grid spacing of `20mx20mx10m` leading to a computational grid of `2250x1750x1500` grid points (5.9 billion grid points). One of the main challenges of elastic modelling is the extreme memory cost owing to the number of wavefields (a minimum of 21 fields in a three dimensional propagator) that need to be stored:

- Three particle velocities with two time steps (`v.forward` and `v`)
- Six stress with two time steps (`tau.forward` and `tau`)
- Three model parameters `lambda`, `mu` and `rho`

These 21 fields, for the 5.9 billion point grid defined above, lead to a minimum memory requirement of 461Gb for modelling alone. For this experiment, access was obtained for small HPC VMs (on Azure) called `Standard_H16r`. These VMs contain 16 core Intel Xeon E5 2667 v3 chips, with no hyperthreading, and 32 nodes were used for a single source experiment (i.e. a single wave-equation was solved). We used a 12th order discretization in space that led to 2.8TFLOP/time-step being computed by this model and the elastic wave was propagated for 16 seconds (23000 time steps). Completion of this modelling run took 16 hours, converting to 1.1TFLOP/s. While these numbers may appear low, it should be noted that the elastic kernel is extremely memory bound, while the TTI kernel is nearly compute bound (see rooflines in [16], [20], [27]) making it more computationally efficient, particularly in combination with MPI. Future work will involve working on InfiniBand enabled and true HPC VMs on Azure to achieve Cloud performance on par with that of state of the art HPC clusters. Extrapolating from the performance obtained in this experiment, and assuming a fairly standard setup of 5000 independent source experiments, computing an elastic synthetic dataset would require 322 EFLOPs (23k time-steps x 2.8TFLOP/time-step x 5000 sources), or utilizing the full scalability and computing all sources in parallel this becomes 5.5PFLOP/s.

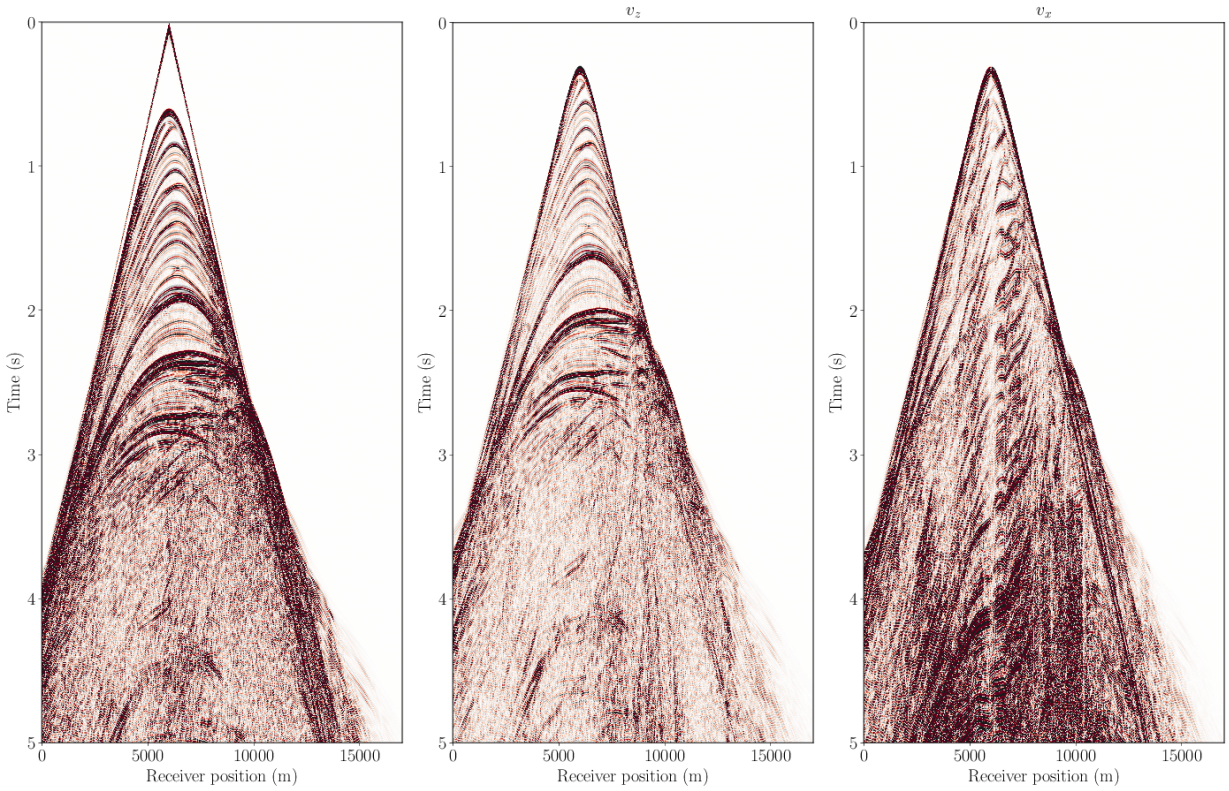

Fig. 4: Seismic shot record for 5sec of modelling. `a` is the pressure (trace of stress tensor) at the surface (5m depth), `b` and `c` display, respectively, the vertical and horizontal particle velocity at the ocean bottom (450m depth).

V. Performance comparison with other codes

Earlier performance benchmarks mainly focused on roofline model analysis. In this study, for completeness, the runtime of Devito is therefore compared to that of the open source hand-coded propagator fdelmodc. This propagator, described in [54], is a state of the art elastic kernel (Equation 3) and the comparisons presented here were carried out in collaboration with its author. To ensure a fair comparison, we ensured that the physical and computational settings were identical. The settings were as follows:

- 2001 by 1001 physical grid points.
- 200 grid points of dampening layer (absorbing layer [55]) on all four sides (total of 2401x1401 computational grid points).
- 10001 time steps.
- Single point source, 2001 receivers.
- Same compiler (GCC/ICC) to compile fdelmodc and run Devito.
- Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.8GHz.
- Single socket, four physical cores, four physical threads, thread pinning to cores and hyperthreading off.

The runtimes observed for this problem were essentially identical, showing less than a one percent of difference. Such similar runtimes were obtained with both the Intel and GNU compilers and the experiment was performed with both fourth and sixth order discretizations. Kernels were executed five times each and the runtimes observed were consistently very similar. This comparison illustrates the performance achieved with Devito is at least on par with hand-coded propagators. Considering we do not take advantage of the Devito compilers full capabilities in two dimensional cases, we are confident that the code generated will be at least on par with the hand-coded version for three dimensional problems and this comparison will be part of our future work.

VI. Conclusions

Transitioning from academic toy problems, such as the two-dimensional acoustic wave-equation, to real-world applications can be challenging, particularly if this transition is carried out as an afterthought. Owing to the fundamental design principles of Devito such scaling, however, becomes trivial. In this work we demonstrated the high-level interface provided by Devito not only for simple scalar equations but also for coupled PDEs. This interface allows, in a simple, concise and consistent manner, the expression of all-kinds of non-trivial differential operators. Next, and most importantly, we demonstrated that the compiler enables large-scale modelling with state-of-the art computational performance and programming paradigm. The single-node performance is on par with state of the

art hand-coded models, but packaged with this performance comes the flexibility of the symbolic interface and multi-node parallelism, which is integrated in the compiler and interface in a accessible way. Finally, we demonstrated that our abstractions provide the necessary portability to enable both on-premise and Cloud based HPC.

VII. Code availability

The code to reproduce the different examples presented in this work is available online in the following repositories:

- The complete code for TTI imaging is available at https://github.com/slimgroup/Azure2019/tree/v1.0 and includes the TTI propagators, the Azure setup for shot parallelism and a documentation.
- The elastic modelling can be run with the elastic example available in Devito at https://github.com/devitocodes/devito/blob/v4.2/examples/seismic/elastic/elastic_example.py and can be run with any size and spatial order. A standalone script to run the large 3D elastic modelling is also available at https://github.com/mloubout/SC20Paper/tree/master/codesamples

References

[1] C. Lyu, Y. Capdeville, and L. Zhao, "Efficiency of the spectral element method with very high polynomial degree to solve the elastic wave equation," *GEOPHYSICS*, vol. 85, no. 1, pp. T33–T43, 2020. [Online]. Available: https://doi.org/10.1190/geo2019-0087.1

[2] A. Fichtner, *Full Seismic Waveform Modelling and Inversion*. Springer Verlag, 2010.

[3] A. Tarantola, "Inversion of seismic reflection data in the acoustic approximation," *GEOPHYSICS*, vol. 49, no. 8, p. 1259, 1984. [Online]. Available: +http://dx.doi.org/10.1190/1.1441754

[4] L. Sirgue, O. I. Barkved, J. P. Van Gestel, O. J. Askim, and J. H. Kommedal, "3d waveform inversion on valhall wide-azimuth obc," 2009. [Online]. Available: https://www.earthdoc.org/content/papers/10.3997/2214-4609.201400395

[5] A. Ratcliffe, C. Win, V. Vinje, G. Conroy, M. Warner, A. Umpleby, I. Stekl, T. Nangoo, and A. Bertrand, "Full waveform inversion: A north sea OBC case study," jan 2011. [Online]. Available: https://doi.org/10.1190%2F1.3627688

[6] N. Petersson and B. Sjögreen, *SW4 v1.1 [software]*, Computational Infrastructure for Geodynamics, 2014.

[7] L. Preston, "Computation of kernels for full waveform seismic inversion using parelasti." 8 2018.

[8] ——, "Paraniso 1.0: 3-d full waveform seismic simulation in general anisotropic media." 9 2019.

[9] ——, "Parelastifwi 1.0 user guide." 9 2019.

[10] D. Köhn, O. Hellwig, D. De Nil, and W. Rabbel, "Waveform inversion in triclinic anisotropic media—a resolution study," *Geophysical Journal International*, vol. 201, no. 3, pp. 1642–1656, 04 2015. [Online]. Available: https://doi.org/10.1093/gji/ggv097

[11] B. Zehner, O. Hellwig, M. Linke, I. Görz, and S. Buske, "Rasterizing geological models for parallel finite difference simulation using seismic simulation as an example," *Computers & Geosciences*, vol. 86, pp. 83 – 91, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S009830041530073X

[12] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, and J. Tromp, "Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes," *Geophysical Journal International*, vol. 186, no. 2, pp. 721–739, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-246X.2011.05044.x

[13] J. Krebs, S. Collis, N. Downey, C. Ober, J. Overfelt, T. Smith, B. van Bloemen-Waanders, and J. Young, "Full wave inversion using a spectral-element discontinuous galerkin method," vol. 2014, no. 1, pp. 1–5, 2014. [Online]. Available: https://www.earthdoc.org/content/papers/10.3997/2214-4609.20140707

[14] P.-T. Trinh, R. Brossier, L. Métivier, L. Tavard, and J. Virieux, "Efficient time-domain 3d elastic and viscoelastic full-waveform inversion using a spectral-element method on flexible cartesian-based mesh," *Geophysics*, vol. 84, no. 1, pp. R75–R97, January-February 2019.

[15] J. Virieux, S. Operto, H. Ben-Hadj-Ali, R. Brossier, V. Etienne, F. Sourbier, L. Giraud, and A. Haidar, "Seismic wave modeling for seismic imaging," *The Leading Edge*, vol. 28, no. 5, pp. 538–544, 2009. [Online]. Available: https://doi.org/10.1190/1.3124928

[16] M. Louboutin, M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko, and G. J. Gorman, "Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration," *Geoscientific Model Development*, vol. 12, no. 3, pp. 1165–1187, 2019. [Online]. Available: https://www.geosci-model-dev.net/12/1165/2019/

[17] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[18] A. Logg, K.-A. Mardal, G. N. Wells *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[19] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G. Bercea, G. R. Markall, and P. H. J. Kelly, "Firedrake: automating the finite element method by composing abstractions," *CoRR*, vol. abs/1501.01809, 2015. [Online]. Available: http://arxiv.org/abs/1501.01809

[20] F. Luporini, M. Lange, M. Louboutin, N. Kukreja, J. Hückelheim, C. Yount, P. Witte, P. H. J. Kelly, G. J. Gorman, and F. J. Herrmann, "Architecture and performance of devito, a system for automated stencil computation," *CoRR*, vol. abs/1807.03032, jul 2018. [Online]. Available: http://arxiv.org/abs/1807.03032

[21] J. Virieux and S. Operto, "An overview of full-waveform inversion in exploration geophysics," *GEOPHYSICS*, vol. 74, no. 5, pp. WCC1–WCC26, 2009. [Online]. Available: http://library.seg.org/doi/abs/10.1190/1.3238367

[22] L. Thomsen, "Weak elastic anisotropy," *Geophysics*, vol. 51, no. 10, pp. 1964–1966, october 1986.

[23] Y. Zhang, H. Zhang, and G. Zhang, "A stable tti reverse time migration and its implementation," *GEOPHYSICS*, vol. 76, no. 3, pp. WA3–WA11, 2011. [Online]. Available: https://doi.org/10.1190/1.3554411

[24] E. Duveneck and P. M. Bakker, "Stable p-wave modeling for reverse-time migration in tilted ti media," *GEOPHYSICS*, vol. 76, no. 2, pp. S65–S75, 2011. [Online]. Available: https://doi.org/10.1190/1.3533964

[25] M. Louboutin, P. A. Witte, and F. J. Herrmann, "Effects of wrong adjoints for rtm in tti media," in *SEG Technical Program Expanded Abstracts*, 10 2018, pp. 331–335, (SEG, Anaheim). [Online]. Available: https://slim.gatech.edu/Publications/Public/Conferences/SEG/2018/louboutin2018SEGeow/louboutin2018SEGeow.html

[26] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[27] M. Louboutin, M. Lange, F. J. Herrmann, N. Kukreja, and G. Gorman, "Performance prediction of finite-difference solvers for different computer architectures," *Computers & Geosciences*, vol. 105, pp. 148–157, 08 2017.

[28] P. A. Witte, M. Louboutin, N. Kukreja, F. Luporini, M. Lange, G. J. Gorman, and F. J. Herrmann, "A large-scale framework for symbolic implementations of seismic inversion algorithms in julia," *Geophysics*, vol. 84, no. 3, pp. F57–F71, 03 2019, (Geophysics). [Online]. Available: https://slim.gatech.edu/Publications/Public/Journals/Geophysics/2019/witte2018alf/witte2018alf.pdf

[29] T. Alkhalifah, "An acoustic wave equation for anisotropic media," *Geophysics*, vol. 65, no. 4, pp. 1239–1250, 2000.

[30] E. Baysal, D. D. Kosloff, , and J. W. C. Sherwood, "Reverse time migration," *Geophysics*, vol. 48, no. 11, pp. 1514–1524, november 1983.

[31] K. P. Bube, T. Nemeth, J. P. Stefani, R. Ergas, W. Liu, K. T. Nihei, and L. Zhang, "On the instability in second-order systems for acoustic vti and tti media," *GEOPHYSICS*, vol. 77, no. 5, pp. T171–T186, 2012. [Online]. Available: https://doi.org/10.1190/geo2011-0250.1

[32] K. P. Bube*, R. Ergas, and T. Nemeth, *Stability and energy conservation for second-order acoustic systems for VTI andTTI media with positive shear wavespeeds*. SEG, 2014, pp. 3439–3443. [Online]. Available: https://library.seg.org/doi/abs/10.1190/segam2014-0986.1

[33] K. Bube, J. Washbourne, R. Ergas, and T. Nemeth, *Self-adjoint, energy-conserving second-order pseudoacoustic systems for VTI and TTI media for reverse time migration and full-waveform inversion*. SEG, 2016, pp. 1110–1114. [Online]. Available: e

[34] C. Chu, B. K. Macy, and P. D. Anno, "Approximation of pure acoustic seismic wave propagation in tti media," *GEOPHYSICS*, vol. 76, no. 5, pp. WB97–WB107, 2011. [Online]. Available: https://doi.org/10.1190/geo2011-0092.1

[35] R. P. Fletcher, X. Du, and P. J. Fowler, "Reverse time migration in tilted transversely isotropic (tti) media," *GEOPHYSICS*, vol. 74, no. 6, pp. WCA179–WCA187, 2009. [Online]. Available: https://doi.org/10.1190/1.3269902

[36] P. J. Fowler, X. Du, and R. P. Fletcher, "Coupled equations for reverse time migration in transversely isotropic media," *GEOPHYSICS*, vol. 75, no. 1, pp. S11–S22, 2010. [Online]. Available: https://doi.org/10.1190/1.3294572

[37] N. D. Whitmore, "Iterative depth migration by backward time propagation," *1983 SEG Annual Meeting, Expanded Abstracts*, 1983.

[38] P. A. Witte, C. C. Stolk, and F. J. Herrmann, "Phase velocity error minimizing scheme for the anisotropic pure p-wave equation," in *SEG Technical Program Expanded Abstracts*, 10 2016, pp. 452–457, (SEG, Dallas). [Online]. Available: https://slim.gatech.edu/Publications/Public/Conferences/SEG/2016/witte2016SEGpve/witte2016SEGpve.html

[39] S. Xu and H. Zhou, "Accurate simulations of pure quasi-p-waves in complex anisotropic media," *Geophysics*, vol. 79, no. 6, pp. 341–348, november-december 2014.

[40] L. Zhang, J. W. Rector III, and H. G. Micheal, "Finite-difference modelling of wave propagation in acoustic tilted ti media," *Geophysical Prospecting*, vol. 53, pp. 843–852, 2005.

[41] Y. Zhang, H. Zhang, and G. Zhang, "A stable tti reverse time migration and its implementation," *Geophysics*, vol. 76, no. 3, pp. WA3–WA11, may-june 2011.

[42] G. Zhan, R. C. Pestana, and P. L. Stoffa, "An efficient hybrid pdeudospectral/finite-difference scheme for solving the tti pure p-wave equation," *Journal of Geophyics and Engineering*, vol. 10, 2013.

[43] R. Hooke, D. Papin, S. Sturmy, and J. Young, *Lectures de potentia restitutiva*. London : Printed for John Martyn ..., 1678. [Online]. Available: http://lib.ugent.be/catalog/rug01:001559640

[44] M. Louboutin, "Modeling for inversion in exploration geophysics," phd, Georgia Institute of Technology, Atlanta, 03 2020, (PhD). [Online]. Available: https://slim.gatech.edu/Publications/Public/Thesis/2020/louboutin2020THmfi/louboutin2020THmfi.pdf

[45] P. A. Witte, M. Louboutin, H. Modzelewski, C. Jones, J. Selvage, and F. J. Herrmann, "An event-driven approach to serverless seismic imaging in the cloud," 2020.

[46] P. A. Witte, M. Louboutin, H. Modzelewski, C. Jones, J. Selvage, and F. J. Herrmann, "Event-driven workflows for large-scale seismic imaging in the cloud," in *SEG Technical Program Expanded Abstracts*, 09 2019, pp. 3984–3988, (SEG, San Antonio). [Online]. Available: https://slim.gatech.edu/Publications/Public/Conferences/SEG/2019/witte2019SEGedw/witte2019SEGedw.html

[47] F. J. Herrmann, C. Jones, G. Gorman, J. Hückelheim, K. Lensink, P. Kelly, N. Kukreja, H. Modzelewski, M. Lange, M. Louboutin, F. Luporini, J. Selvages, and P. A. Witte, "Accelerating ideation & innovation cheaply in the cloud the power of abstraction, collaboration & reproducibility," in *4th EAGE Workshop on High-performance Computing*, 10 2019, (EAGE HPC Workshop, Dubai).

[48] P. A. Witte, M. Louboutin, C. Jones, and F. J. Herrmann, "Serverless seismic imaging in the cloud," 2019, submitted to Rice Oil and Gas High Performance Computing Conference 2020 on November 27, 2019. [Online]. Available: https://slim.gatech.edu/Publications/Public/Submitted/2019/witte2019RHPCssi/witte2019RHPCssi.html

[49] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th Python in Science Conference*, K. Huff and J. Bergstra, Eds., 2015, pp. 130 – 136.

[50] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, "Unified Form Language: a domain-specific language for weak formulations of partial differential equations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 40, no. 2, p. 9, 2014.

[51] R. Versteeg, "The marmousi experience; velocity model determination on a synthetic complex data set," *The Leading Edge*, vol. 13, no. 9, pp. 927–936, 1994. [Online]. Available: http://tle.geoscienceworld.org/content/13/9/927

[52] G. S. Martin, R. Wiley, and K. J. Marfurt, "Marmousi2: An elastic upgrade for marmousi," *The Leading Edge*, vol. 25, no. 2, pp. 156–166, 2006. [Online]. Available: https://doi.org/10.1190/1.2172306

[53] M. Fehler and P. J. Keliher, *SEAM Phase 1: Challenges of subsalt imaging in tertiary basins, with emphasis on deepwater Gulf of Mexico*. Society of Exploration Geophysicists, 2011.

[54] J. W. Thorbecke and D. Draganov, "Finite-difference modeling experiments for seismic interferometry," *GEOPHYSICS*, vol. 76, no. 6, pp. H1–H18, 2011. [Online]. Available: https://doi.org/10.1190/geo2010-0039.1

[55] C. Cerjan, D. Kosloff, R. Kosloff, and M. Reshef, "A nonreflecting boundary condition for discrete acoustic and elastic wave equations," *GEOPHYSICS*, vol. 50, no. 4, pp. 705–708, 1985. [Online]. Available: https://doi.org/10.1190/1.1441945