Leveraging symbolic math for rapid development of applications for seismic modeling

Navjot Kukreja, Mathias Louboutin, Michael Lange,
Fabio Luporini, Gerard Gorman

Abstract

Wave propagation kernels are the core of many commonly used algorithms for inverse problems in exploration geophysics. While they are easy to write and analyze for the simplified cases, the code quickly becomes complex when the physics needs to be made more precise or the performance of these codes is to be optimized. Significant effort is repeated every time new forms of physics need to be implemented, or a new computing platform to be supported. The use of symbolic mathematics as a domain specific language (DSL) for input, combined with automatic generation of high performance code customized for the target hardware platform is a promising approach to maximize code reuse. Devito is a DSL for finite difference that uses symbolic mathematics to generate optimized code for wave propagation based on a provided wave equation. It enables rapid application development in a field where the average time spent on development has historically been in weeks and months. The Devito DSL system is completely wrapped within the Python programming language and the fact that the running code is in C is completely transparent, making it simple to include Devito as part of a larger workflow including multiple applications over a large cluster.

1 Introduction

In recent years, petroleum exploration is increasingly being carried out in regions that have complex geological features. This has increased the demand for inversion based geophysical analysis which is made feasible with the parallel increase in the computational power. This increased computational power, however, comes with the caveat of increased programming complexity - not all code would necessarily be able to utilize modern processors to their full capabilities unless specifically optimized for performance on that particular hardware platform. The code written is often specific to different platforms - CPU vs Accelerators vs GPU vs FPGA. Similarly the choice of optimizations is also often specific to the platform as well as between different kinds of problems to be solved [3].

The common approach for the development of geophysical inversion algorithms starts with the geophysicists writing the first version of the code - followed by a phase of "code performance optimization" by computational experts. It is rare for the reuse of this optimized code for the development of newer algorithms - usually codes implementing new algorithms are written from scratch. Similarly, moving between computational platforms (e.g. CPU to GPU) also incurs significant development costs and repetition of effort.

A Domain specific language (DSL) would create a separation of concerns between the geophysical application developer and the computational scientist. The DSL enables "write once, run everywhere", *i.e.* the same code can be run on any current or future hardware platform with no additional effort from the application developer.

Devito [1] is a finite difference (FD) DSL for seismic imaging that accepts a high-level mathematical description of the equation to be solved in symbolic form. It then uses this symbolic description to automatically derive the finite difference kernel which is then used to generate optimized C code for multiple target architectures. However, these details are all abstracted away by providing a standard *Python* interface.

Since Devito is targeted specifically at seismic imaging problems, it provides features beyond a FD DSL. The most significant of these is the sparse-point interpolation which is an often used formulation to represent sources and receivers in code for seismic imaging. The fact that Devito's DSL is embedded within *Python* makes it easy to include Devito based code within a larger application running across a large cluster - something that is typical of inversion type problems. The large ecosystem of packages available to the python language reduces the development effort further.

2 Devito overview

Since Devito operators consist of purely symbolic expressions, low-level implementation choices that aid performance are left to Devito, which enables streamlined code generation bespoke to the target platform.

2.1 Symbolic stencil equations

The *SymPy* [4] Python package provides the basic symbol objects to represent complete equations in symbolic form. Devito utilizes these expressions as the basis for generating optimized finite difference stencil operators by providing two types of objects:

Symbolic Data Objects that associate grid data with *SymPy* symbols to form symbolic expressions.

Operator Objects that generate optimized C code to apply a given stencil expression to the associated data.

Devito's symbolic data objects decorate user data with the symbolic behaviour of `sympy.Function` objects and encapsulate time-varying functions as well as field data constant in time. These objects can be used directly within *SymPy* expressions.

Devito data objects also provide shorthand notation for common finite difference formulations. These consist primarily of automatically expanded symbolic expressions for first, second and cross derivatives in the time and space dimensions, where the order of the discretization is defined on the symbolic data object. For example `f.dx2` expands to `-2*f(x, y)/h**2 + f(-h + x, y)/h**2 + f(h + x, y)/h**2` for the `DenseData` object that has `space_order` set to 2.

The symbol `h` has been inserted to represent the grid spacing in x. It is important to note here that this API allows the user to change the spatial discretization of the problem by simply changing one parameter for the `DenseData` object `f`.

Another property of Devito's symbolic data objects is `f.laplace` that expands to `(f.dx2 + f.dy2)` for two-dimensional problems and `(f.dx2 + f.dy2 + f.dz2)` for three-dimensional ones. This allows the concise expression of PDEs, such as the acoustic wave equation: `wave_eqn = Eq(m * u.dt2, u.laplace)`

2.2 Automated performance optimization

To generate a performance optimized FD kernel, Devito uses properties of the target hardware and the dataset at the time of code generation. The performance optimization techniques that Devito uses currently are:

- Vectorization based on compiler hints
- OpenMP based thread parallelization
- Loop blocking with auto-tuning of block size
- Memory mapping - write large variables to disk automatically
- Common sub-expression elimination
- First touch initialization for effective utilization of NUMA

3 Seismic modeling

The simplest wave equation used is the acoustic case. For a spatially varying velocity model, c, the equation in the time domain is given by:

$$\begin{cases} m\frac{d^2u(x,t)}{dt^2} - \nabla^2 u(x,t) = q \\ u(.,0) = 0 \\ \frac{du(x,t)}{dt}|_{t=0} = 0 \end{cases} \tag{1}$$

where u is the wavefield, q is the source, ∇ is the Laplacian and $\frac{\partial^2 u}{\partial t^2}$ is the second-order time derivative. This equation is solved explicitly with a time marching scheme as part of a Reverse Time Migration scheme.

The code required to implement the forward solution of the equation is presented in listing 1 while listing 2 shows the code required to implement the solution to the adjoint equation.

Listing 1: Example code to solve the wave equation

```python
def forward(model, nt, dt, h, spc_order=2):
    m = DenseData("m", model.shape)
    m.data[:] = model
    u = TimeData(name='u',
        shape=model.shape, time_dim=nt,
        time_order=2,
        space_order=spc_order, save=True)
    damp = DenseData("damp", model.shape)

    # Derive stencil from symbolic equation
    eqn = m * u.dt2 - u.laplace + damp *
        u.dt
    stencil = solve(eqn, u.forward)[0]

    # Add spacing substitutions
    subs = {s: dt, h: h}
    op = Operator(stencils=Eq(u.forward,
        stencil), nt=nt, shape=model.shape,
        subs=subs)
    op.apply()
```

Listing 2: Example code to solve the adjoint of the wave equation

```python
def adjoint(model, nt, dt, h, spc_order=2):
    m = DenseData("m", model.shape)
    m.data[:] = model
    v = TimeData(name='v',
        shape=model.shape, time_dim=nt,
        time_order=2,
        space_order=spc_order, save=True)
    damp = DenseData("damp", model.shape)

    # Derive stencil from symbolic equation
    eqn = m * v.dt2 - v.laplace - damp *
        v.dt
    stencil = solve(eqn, v.backward)[0]

    # Add spacing substitutions
    subs = {s: dt, h: h}
    op = Operator(stencils=Eq(u.backward,
        stencil), nt=nt, shape=model.shape,
        subs=subs, forward=False)
    op.apply()
```

While the acoustic case is quite simple - it is not complex to write a finite difference formulation for the above equation by hand, this is not true for more complex forms of the equation like the one with tilted transverse isotropy (TTI). Writing a 16th order finite difference stencil for TTI would be considered an impossible feat by hand. However, when Devito was used to generate code for TTI, it only involved a change in the symbolic description of the original equation. Changing the spatial order of the discretization is trivial with Devito, as seen in the code listings.

3.1 Results

As part of our preliminary results, figure 1 depicts the performance of 8 different versions of the generated code on an Intel Xeon E5-2697v5 (Broadwell) @ 2.3GHz in comparison with the maximum achievable performance the platform.

More tests were carried out, comparing the numerical results of code generated by Devito with those of

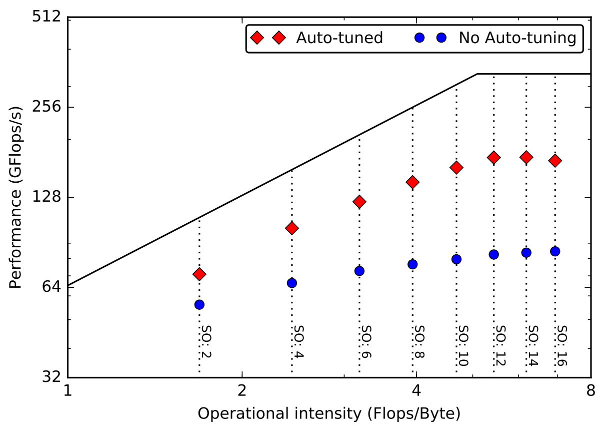

Figure 1: Roofline plot showing compute performance on an Intel Xeon Broadwell for different spatial orders

widely accepted codes for RTM in the industry - confirming the validity of the numerical results. These tests were extended to compare compute performance with industry leading codes as well. It was seen that code generated by Devito produces results in the same or slightly less time.

4 Conclusions

Despite being in its infancy, Devito already provides a prototype DSL as well as a minimal set of features that make it possible to solve simple yet realistic models based on wave equations. The research presented in this work has a multidisciplinary nature, which is reflected by the multi-layer structure of the framework. A necessary condition for the success of the project is the generation of highly optimized stencil code, as this is a strong requirement for the execution of real-world seismic inversion problems.

5 Acknowledgments

This work was financially supported in part by the Natural Sciences and Engineering Research Council of Canada Collaborative Research and Development Grant DNOISE II (CDRP J 375142-08) and the Imperial College London Intel Parallel Computing Centre. This research was carried out as part of the SINBAD II project with the support of the member organizations of the SINBAD Consortium.

References

[1] N. Kukreja, M. Louboutin, F. Vieira, F. Luporini, M. Lange, and G. Gorman. Devito: automated fast finite difference computation. *arXiv preprint arXiv:1608.08658*, 2016.

[2] M. Lange, N. Kukreja, M. Louboutin, F. Luporini, F. Vieira, V. Pandolfo, P. Velesko, P. Kazakas, and G. Gorman. Devito: Towards a generic finite difference dsl using symbolic python. *arXiv preprint arXiv:1609.03361*, 2016.

[3] M. Louboutin, M. Lange, F. Herrmann, N. Kukreja, and G. Gorman. Performance prediction of finite-difference solvers for different computer architectures. *arXiv preprint arXiv:1608.03984*, 2016.

[4] SymPy Development Team. *SymPy: Python library for symbolic mathematics*, 2016.