

Leveraging symbolic math for rapid development of applications for seismic imaging

Navjot Kukreja¹ M. Louboutin² F. Luporini¹ P. Witte² M. Lange¹ F. Hermann² G. Gorman¹

March 16, 2017

¹Department of Earth Science and Engineering, Imperial College London, UK

²Seismic Lab. for Imaging and Modeling, The University of British Columbia, Canada

Introduction

Motivation

Example

Use geophysics to understand the earth

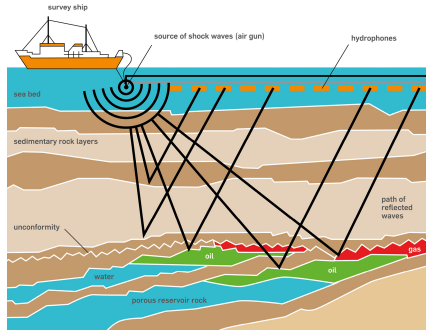
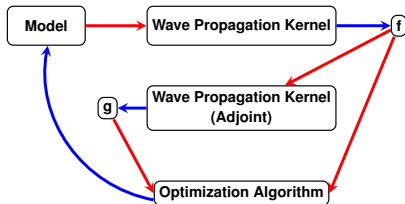
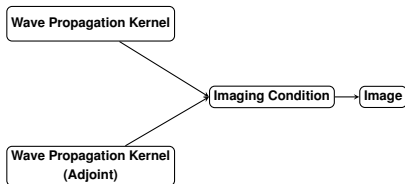


Figure 1: Offshore seismic survey

Source: <http://www.open.edu/openlearn/science-maths-technology/science/environmental-science/earths-physical-resources-petroleum/content-section-3.2.1>



Introduction

Motivation

Example

- Large number of operations: ≈ 6000 FLOPs per loop iteration of a 16th order TTI kernel
- Realistic problems have large grids: $1580 \times 1580 \times 1130 \approx 2.82$ billion points (SEAM benchmark ¹)
- $2.82 \times 10^9 \times 6000 \times 3000(t) \times 2$ (forward-reverse) $\approx 10^{17}$ FLOPs per shot
- Typically ≈ 30000 shots ($\approx 3 \times 10^{21} = 3 \times 10^9$ TFLOPs per FWI iteration)
- Typically ≈ 15 FWI iterations ($\approx 4.6 \times 10^{22} = 46$ billion TFLOPs total)

≈ 100 wall-clock days executing on the TACC Stampede (assuming linpack-level performance)

¹Michael Fehler and P. Joseph Keliher. *SEAM Phase I*. Society of Exploration Geophysicists, 2011

Computer science

- Fast code is complex
 - Loop blocking
 - OpenMP clauses
 - Vectorization - intrinsics
 - Memory - alignment, NUMA
 - Common sub-expression elimination
 - ADD/MUL balance
 - Denormal numbers
 - Elemental functions
 - Non temporal stores
- Fast code is platform dependent
 - Intrinsics
 - CUDA/OpenCL
 - Data layouts
- Fast code is error prone

Geophysics

- Change of discretizations - Numerical analysis
- Change of physics
 - Anisotropy - VTI/TTI
 - Elastic equation
- Boundary conditions

Computer science

- Fast code is complex
 - Loop blocking
 - OpenMP clauses
 - Vectorization - intrinsics
 - Memory - alignment, NUMA
 - Common sub-expression elimination
 - ADD/MUL balance
 - Denormal numbers
 - Elemental functions
 - Non temporal stores
- Fast code is platform dependent
 - Intrinsics
 - CUDA/OpenCL
 - Data layouts
- Fast code is error prone

Geophysics

- Change of discretizations - Numerical analysis
- Change of physics
 - Anisotropy - VTI/TTI
 - Elastic equation
- Boundary conditions

Not everyone is a polymath

Devito - A Finite Difference DSL for seismic imaging

- Aimed at creating fast high-order inversion kernels
- Development is driven by real-world problems

Based on SymPy expressions

- The acoustic wave equation:

$$m \frac{\partial^2 u}{\partial t^2} + \eta \frac{\partial u}{\partial t} - \nabla u = 0 \quad (1)$$

can be written as

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
```

Devito auto-generates optimized C code and provides the necessary plumbing to use it directly from Python

Real-world applications need more than PDE solvers

- File I/O and support for large datasets
- Non-PDE kernel code e.g. insert source, sample receivers
- Ability to easily interface with complex outer code

Devito follows the principle of graceful degradation

- Circumvent restrictions to the high-level API by customization
- Allows custom functionality in auto-generated kernels

Introduction

Motivation

Example

```
def forward(model, nt, dt, h, order=2):
    shape = model.shape
    m = DenseData(name="m", shape=shape, space_order=order)
    m.data[:] = model
    u = TimeData(name='u', shape=shape, time_order=2,
                  space_order=order)
    eta = DenseData(name='eta', shape=shape,
                    space_order=order)

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

    # Add source and receiver interpolation
    source = u.inject(src * dt^2 / m)
    receiver = rec.interpolate(u)

    # Create and execute operator kernel
    op = Operator(stencils=source + stencil + receiver,
                  subs={s: dt, h: h})
    op.apply(t=nt)
```

```
#pragma omp parallel
{
    _MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
    _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
}

#pragma omp parallel
{
    for (int i4 = 0; i4<329; i4+=1)
    {
        struct timeval start_main, end_main;
        #pragma omp master
        gettimeofday(&start_main, NULL);
        {
            #pragma omp for schedule(static)
            for (int i1 = 8; i1<122; i1++)
                for (int i2 = 8; i2<122; i2++)
                {
                    #pragma omp simd aligned(damp, m, u:64)
                    for (int i3 = 8; i3<122; i3++)
                    {
                        u[i4][i1][i2][i3] = ((3.04F*damp[i1][i2][i3] - 2*m[i1][i2][i3])*u[i4 - 2][i1][i2][i3] - 1.12198912198912e-7F*(u[i4 - 1][i1][i2][i3 - 8] + u[i4 - 1][i1][i2][i3 + 8] + u[i4 - 1][i1][i2 - 8][i3] + u[i4 - 1][i1][i2 + 8][i3] + u[i4 - 1][i1 - 8][i2][i3] + u[i4 - 1][i1 + 8][i2][i3]) + 2.34472828758543e-6F*(u[i4 - 1][i1][i2][i3 - 7] + u[i4 - 1][i1][i2][i3 + 7] + u[i4 - 1][i1][i2 - 7][i3] + u[i4 - 1][i1][i2 + 7][i3] + u[i4 - 1][i1 - 7][i2][i3] + u[i4 - 1][i1 + 7][i2][i3]) - 2.39357679357679e-5F*(u[i4 - 1][i1][i2][i3 - 6] + u[i4 - 1][i1][i2][i3 + 6] + u[i4 - 1][i1][i2 - 6][i3] + u[i4 - 1][i1][i2 + 6][i3] + u[i4 - 1][i1 - 6][i2][i3] + u[i4 - 1][i1 + 6][i2][i3]) + 1.60848360528361e-4F*(u[i4 - 1][i1][i2][i3 - 5] + u[i4 - 1][i1][i2][i3 + 5] + u[i4 - 1][i1][i2 - 5][i3] + u[i4 - 1][i1][i2 + 5][i3] + u[i4 - 1][i1 - 5][i2][i3] + u[i4 - 1][i1 + 5][i2][i3]) - 8.16808080808081e-4F*(u[i4 - 1][i1][i2][i3 - 4] + u[i4 - 1][i1][i2][i3 + 4] + u[i4 - 1][i1][i2 - 4][i3] + u[i4 - 1][i1][i2 + 4][i3] + u[i4 - 1][i1 - 4][i2][i3] + u[i4 - 1][i1 + 4][i2][i3]) + 3.48504781144781e-3F*(u[i4 - 1][i1][i2][i3 - 3] + u[i4 - 1][i1][i2][i3 + 3] + u[i4 - 1][i1][i2 - 3][i3] + u[i4 - 1][i1][i2 + 3][i3] + u[i4 - 1][i1 - 3][i2][i3] + u[i4 - 1][i1 + 3][i2][i3]) - 1.43758222222222e-2F*(u[i4 - 1][i1][i2][i3 - 2] + u[i4 - 1][i1][i2][i3 + 2] + u[i4 - 1][i1][i2 - 2][i3] + u[i4 - 1][i1][i2 + 2][i3] + u[i4 - 1][i1 - 2][i2][i3] + u[i4 - 1][i1 + 2][i2][i3]) + 8.21475555555556e-2F*(u[i4 - 1][i1][i2][i3 - 1] + u[i4 - 1][i1][i2][i3 + 1] + u[i4 - 1][i1][i2 - 1][i3] + u[i4 - 1][i1][i2 + 1][i3] + u[i4 - 1][i1 - 1][i2][i3] + u[i4 - 1][i1 + 1][i2][i3]) + 4*m[i1][i2][i3]*u[i4 - 1][i1][i2][i3] - 4.23474709115646e-1F*u[i4 - 1][i1][i2][i3])/ (3.04F*damp[i1][i2][i3] + 2*m[i1][i2][i3]));
                    }
                }
            }
        }
    }
}
```



```

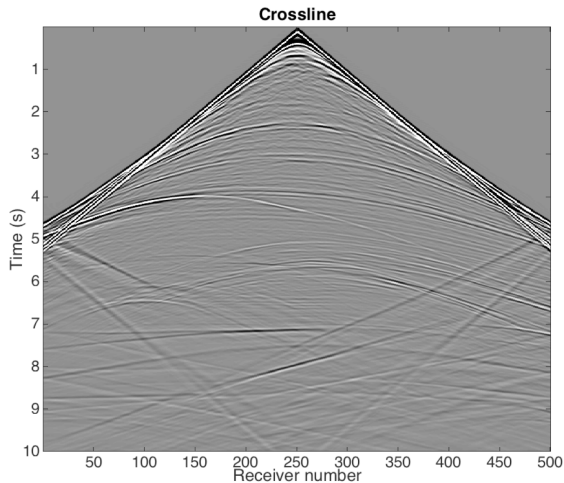
...+src[14]*p_src)/m[(int){
p_src[2]] + 40] + u[14
+ords[p_src[2]] + 40];
tp_src[2])) + 40] -

```

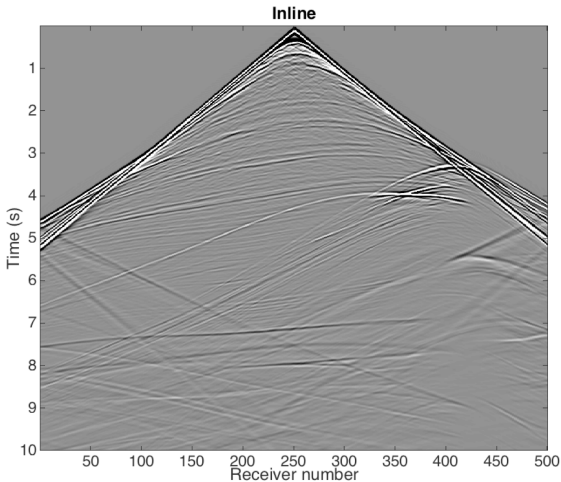
Full SEAM:

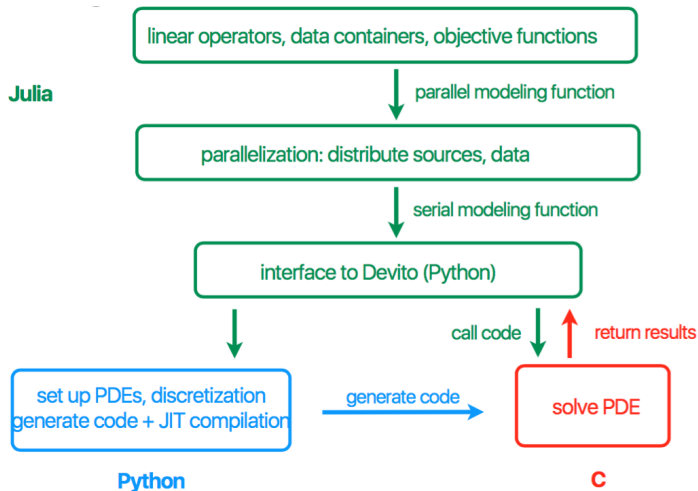
- 10m grid
- 5001 time steps (10sec modelling)
- 1575x1575x1125 domain
- 10Hz source in the middle of XY 40m depth
- 500x500 receivers
- Intel(R) Xeon(R) E5-2680 v2, 10 cores per socket, dual socket
- Time for a single modeling: 4h15min

Shot Record



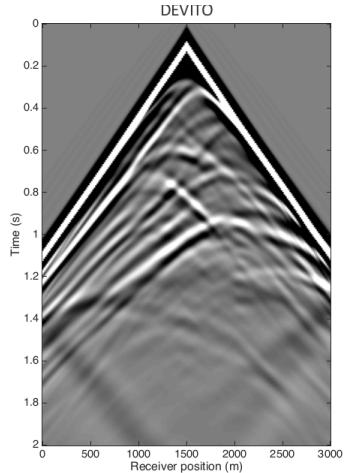
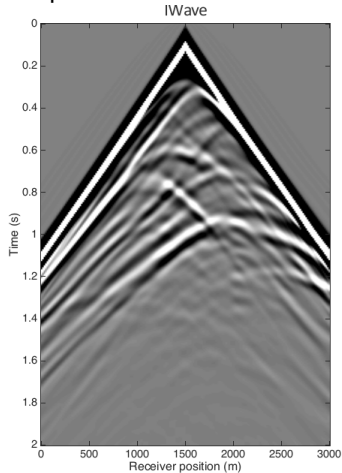
Shot Record





¹ Philipp Witte, Mathias Louboutin, and Felix J. Herrmann. Large-scale workflows for wave-equation based inversion in Julia. In *Domain-Specific Abstractions for Full-Waveform Inversion at SIAM CSE*, 2017

Comparison with a reference implementation - IWAVE

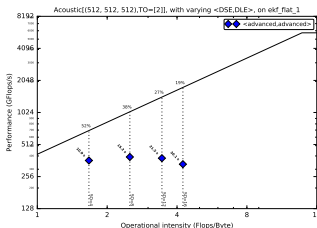
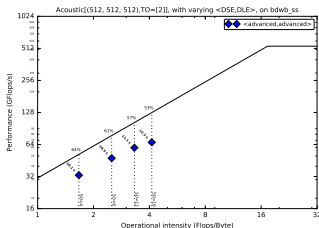


Verification of the generated code:

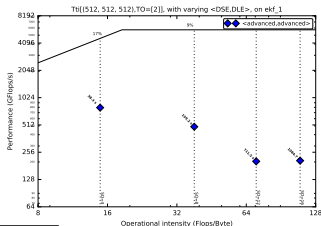
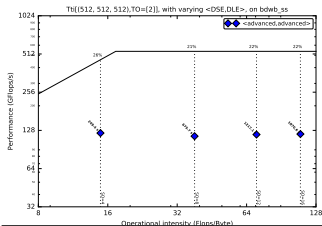
- Extensive unit-testing already in place with continuous integration (Travis)
- Adjoint test
 - For any $x \in \text{span}(P_s A^T P_r^T)$, $y \in \text{span}(P_r A^T P_s^T)$
 - $\langle P_r A^T P_s^T x, y \rangle - \langle x P_s A^T P_r^T y \rangle = 0$
 - Passes with at-least 8 matching significant digits for 2D and 3D with 2,4,6,8,10,12th order discretization
- Gradient test
 - For a small model perturbation dm , $\phi_s(m + hdm) = \phi_s(m) + \mathcal{O}(h)$ and $\phi_s(m + hdm) = \phi_s(m) + h(J[m]^T \delta d)dm + \mathcal{O}(h^2)$
 - Passes at the level of the machine's accuracy
- Automatic formal code verification being implemented ¹

¹Christopher Lidbury, Andrei Lascu, Nathan Chong, and Alastair F Donaldson. Many-core compiler fuzzing. In *ACM SIGPLAN Notices*, volume 50, pages 65–76. ACM, 2015

- Performance of acoustic forward operator
- Left:
 - Intel®Xeon™E5-2620 v4 2.1Ghz Broadwell (8 cores per socket, single socket)
 - Model size $512 \times 512 \times 512$, $t_n = 250$
- Right:
 - Intel®Xeon Phi™7650 Knightslanding (68 cores) Quadrant Mode
 - Model size $512 \times 512 \times 512$, $t_n = 3000$



- Performance of TTI¹ forward operator
- Left:
 - Intel®Xeon™E5-2620 v4 2.1Ghz Broadwell (8 cores per socket, single socket)
 - Model size $512 \times 512 \times 512$, $t_n = 250$
- Right:
 - Intel®Xeon Phi™7650 Knightslanding (68 cores) Quadrant mode
 - Model size $512 \times 512 \times 512$, $t_n = 3000$



¹Yu Zhang, Houzhu Zhang, and Guanquan Zhang. A stable tti reverse time migration and its implementation. *Geophysics*, 76(3):WA3–WA11, 2011

- Devito: A finite difference DSL for seismic imaging
 - Symbolic problem description (PDEs) via SymPy
 - Low-level API for kernel customization
 - Automated performance optimization
- Devito is driven by real-world scientific problems
 - Bring the latest in performance optimization closer to real science
- Future work:
 - Extend feature range to facilitate more science
 - MPI parallelism for larger models
 - Integrate stencil or polyhedral compiler backends like YASK

Publications

- N. Kukreja, M. Louboutin, F. Vieira, F. Luporini, M. Lange, and G. Gorman. Devito: automated fast finite difference computation. WOLFHPC 2016
- 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. PyHPC 2016
- M. Louboutin, M. Lange, N. Kukreja, F. Herrmann, and G. Gorman. Performance prediction of finite-difference solvers for different computer architectures. Submitted to Computers and Geosciences, 2016

Web links

- www.opesci.org
- github.com/opesci



BG GROUP



Imperial College
London