Rapid development of seismic imaging applications using Symbolic mathematics

Navjot Kukreja[1] M. Lange[1] M. Louboutin[2] F. Luporini[1] J. Hueckelheim[1] P. Witte[2] C. Yount[3] F. Herrmann[2] G. Gorman[1]

October 2, 2017

[1]Department of Earth Science and Engineering, Imperial College London, UK
[2]Seismic Lab. for Imaging and Modeling, The University of British Columbia, Canada
[3]Intel Corporation

Seismic Imaging - Motivation

Use geophysics to understand the earth

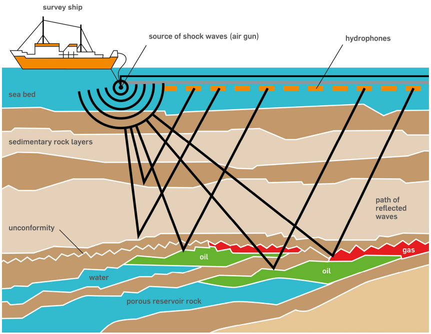


Figure 1: Offshore seismic survey

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

But finite difference is simple...

Acoustic wave equation with 2nd-order discretisation:

```
for ti in range(timesteps):
    t0 = ti % 3
    t1 = (ti + 1) % 3
    t2 = (ti + 2) % 3
    for i in range(1, nx-1):
        for j in range(1, ny-1):
            uxx = (u[t1, i+1, j] -2 * u[t1, i, j] + u[t1, i-1, j]) / dx2
            uyy = (u[t1, i, j+1] -2 * u[t1, i, j] + u[t1, i, j-1]) / dy2
            u[t2, i, j] = 2*u[t1, i, j] - u[t0, i, j] + dt * dt *
            (uxx + uyy)/m[i, j]
```

Not really...

OPESCI

12th-order acoustic wave equation:

```
for (int i4 = 0; i4<149; i4+=1) {
  for (int i1 = 6; i1<64; i1++) {
    for (int i2 = 6; i2<64; i2++) {
      for (int i3 = 6; i3<64; i3++) {
        u[i4][i1][i2][i3] = 6.01250601250601e-9F*(2.80896e+8F*damp[i1][i2][i3]*u[i4-2][i1][i2][
               i3]-3.3264e+8F*m[i1][i2][i3]*u[i4-2][i1][i2][i3]+6.6528e+8F*m[i1][i2][i3]*u[i4
               -1][i1][i2][i3]-2.12255421155556e+7F*u[i4-1][i1][i2][i3]-1.42617283950617e+2F*u[
               i4-1][i1][i2][i3-6]+2.46442666666667e+3F*u[i4-1][i1][i2][i3-5]-2.11786666666667e
               +4F*u[i4-1][i1][i2][i3-4]+1.25503209876543e+5F*u[i4-1][i1][i2][i3-3]-6.3536e+5F*u
               [i4-1][i1][i2][i3-2]+4.066304e+6F*u[i4-1][i1][i2][i3-1]+4.066304e+6F*u[i4-1][i1][
               i2][i3+1]-6.3536e+5F*u[i4-1][i1][i2][i3+2]+1.25503209876543e+5F*u[i4-1][i1][i2][
               i3+3]-2.11786666666667e+4F*u[i4-1][i1][i2][i3+4]+2.46442666666667e+3F*u[i4-1][i1
               ][i2][i3+5]-1.42617283950617e+2F*u[i4-1][i1][i2][i3+6]-1.42617283950617e+2F*u[i4
               -1][i1][i2-6][i3]+2.46442666666667e+3F*u[i4-1][i1][i2-5][i3]-2.11786666666667e+4F
               *u[i4-1][i1][i2-4][i3]+1.25503209876543e+5F*u[i4-1][i1][i2-3][i3]-6.3536e+5F*u[i4
               -1][i1][i2-2][i3]+4.066304e+6F*u[i4-1][i1][i2-1][i3]+4.066304e+6F*u[i4-1][i1][i2
               +1][i3]-6.3536e+5F*u[i4-1][i1][i2+2][i3]+1.25503209876543e+5F*u[i4-1][i1][i2+3][
               i3]-2.11786666666667e+4F*u[i4-1][i1][i2+4][i3]+2.46442666666667e+3F*u[i4-1][i1][
               i2+5][i3]-1.42617283950617e+2F*u[i4-1][i1][i2+6][i3]-1.42617283950617e+2F*u[i4
               -1][i1-6][i2][i3]+2.46442666666667e+3F*u[i4-1][i1-5][i2][i3]-2.11786666666667e+4F
               *u[i4-1][i1-4][i2][i3]+1.25503209876543e+5F*u[i4-1][i1-3][i2][i3]-6.3536e+5F*u[i4
               -1][i1-2][i2][i3]+4.066304e+6F*u[i4-1][i1-1][i2][i3]+4.066304e+6F*u[i4-1][i1+1][
               i2][i3]-6.3536e+5F*u[i4-1][i1+2][i2][i3]+1.25503209876543e+5F*u[i4-1][i1+3][i2][
               i3]-2.11786666666667e+4F*u[i4-1][i1+4][i2][i3]+2.46442666666667e+3F*u[i4-1][i1
               +5][i2][i3]-1.42617283950617e+2F*u[i4-1][i1+6][i2][i3])/(1.68888888888889F*damp[
               i1][i2][i3]+2*m[i1][i2][i3]);
      }
    }
  }
}
```

Imperial College London

Challenges

OPESCI

- Huge Problem sizes
  - Seismic surveys consist of thousands of individual experiments
  - Model wave propagation across large domains over thousands of timesteps
  - The code needs to be highly optimised for performance, including selecting appropriate discretisation for performance
- Performance optimisations are hardware-specific
  - Parallelisation/Vectorisation - Intrinsics
  - Cache-reuse (loop blocking)
  - Memory - alternate layouts, alignment, NUMA
  - Common sub-expression elimination
  - Others - Elemental functions/loop fission, Denormal numbers, streaming stores etc.
- Reverse mode requires the adjoint of the equation
  - Storing the entire forward wave-field in memory is prohibitively expensive
  - Solutions involving saving partial forward state and recomputing - e.g. checkpointing
- Various physical models
  - Inelastic - acoustic/VTI/TTI
  - Elastic

The real challenges

- Too much bespoke code being written
- Everyone who works on such a piece of code needs to be a polymath
- No suitable separation of concerns between the three kinds of expertise this code requires - Computer Science, Mathematics, Physics

Symbolic computation is a powerful tool

SymPy: Symbolic computer algebra system in pure Python[1]

Enables automation of stencil generation

- Complex symbolic expressions as Python object trees
- Symbolic manipulation routines and interfaces
- Convert symbolic expressions to numeric functions
  - Python (NumPy) functions; C or Fortran kernels
- For a great overview see A. Meurer's talk at SciPy 2016

For specialised domains generating C code is not enough!

- Compiler-level optimimizaton to leverage performance
- Stencil optimization is a research field of its own

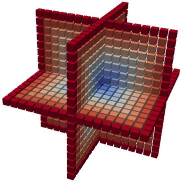
1

OPESCI

Devito: Finite difference DSL based on SymPy

Devito generates highly optimized stencil code...

- OpenMP threading and vectorisation pragmas
- Cache blocking and auto-tuning
- Symbolic stencil optimisation



... from concise mathematical syntax

Example: acoustic wave equation with dampening

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

can be written as

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

The power of symbolics

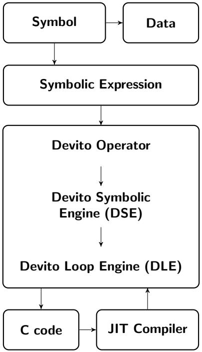
```
from devito import TimeData, DenseData
from sympy import solve

shape = (10, 10)
space_order = 2
time_order = 2
m = DenseData(name='m', shape=shape, space_order=space_order)
u = TimeData(name='u', shape=shape, space_order=space_order, time_order=
     time_order)

eqn = m * u.dt2 - u.laplace
stencil=solve(eqn, u.forward)

[In] print(stencil)
[Out] [(h_x**2*h_y**2*(2*u(t, x, y) - u(t - s, x, y))*m(x, y) + h_x**2*s
     **2*(-2*u(t, x, y) + u(t, x, y - h_y) + u(t, x, y + h_y)) + h_y**2*s
     **2*(-2*u(t, x, y) + u(t, x - h_x, y) + u(t, x + h_x, y)))/(h_x**2*
     h_y**2*m(x, y))]
```

Imperial College London




```
u = TimeData(name='u', shape=(nx, ny))
m = DenseData(name='m', shape=(nx, ny))
```

High-level function symbols associated with user data

```
eqn = m * u.dt2 - u.laplace
```

Symbolic equations that expand finite difference stencils

```
op = Operator(expressions)
op.apply(time=ntime)
```

Automatic code generation and execution from high-level expressions

Symbolic optimization to reduce computation per stencil point

Loop-level optimization for efficient parallel execution

Just-in-time compilation of optimized C code

Figure 2: Overview of the Devito architecture and associated example workflow. Devito's top-level API allows users to generate symbolic stencil expressions from data-carrying function objects that can be used to for symbolic expressions vis SymPy. From this high-level definition an operator then generates, compiles and executes optimized high-performance C code.

Example forward propagator

OPESCI

Wave propagators in less than 20 lines

```
def forward(model, m, eta, src, rec, order=2, save=True):
    # Create the wavefeld function
    u = TimeData(name='u', shape=model.shape, save=save,
                 time_order=2, space_order=order)

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

    # Inject wave as source term
    src_term = src.inject(field=u, expr=src * dt**2 / m)

    # Interpolate wavefield onto receivers
    rec_term = rec.interpolate(expr=u)

    # Create operator with source and receiver terms
    return Operator(update_u + src_term + rec_term,
                    subs={s: dt, h: model.spacing})
```

Example adjoint propagator

Wave propagators in less than 20 lines

```
def adjoint(model, m, eta, srca, rec, order=2):
    # Create the adjoint wavefeld function
    v = TimeData(name='v', shape=model.shape,
                 time_order=2, space_order=order)

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

    # Inject the previous receiver readings
    rec_term = rec.inject(field=v, expr=rec * dt**2 / m)

    # Interpolate the adjoint-source
    srca_term = srca.interpolate(expr=v)

    # Create operator with source and receiver terms
    return Operator(update_v + rec_term + srca_term,
                    subs={s: dt, h: model.spacing},
                    time_axis=Backward)
```

Example RTM setup

OPESCI

Reverse time migration in less than 100 lines

```
# Create the true and a smoothed model
m_true = Model(...)
m_smooth = Model(...)

# Create operators for forward and gradient
op_forward = forward(...)
op_gradient = forward(...)

# Create gradient field and loop over shots
grad = DenseData(name='grad', shape=model.shape)

for shot in shots:
    # Create receiver data from true model
    src = PointData(shot.source, ...)
    rec_true = PointData(shot.receiver.coordinates, ...)
    op_forward(src=src, rec=rec_true, m=m_true)

    # Run forward modelling operator with smooth model
    u = TimeData(name='u', shape=model.shape,
                 time_order=2, space_order=order)
    rec_smooth = PointData(shot.receiver.coordinates, ...)
    op_forward(u=u, src=src, rec=rec_smooth, m=m_smooth)

    # Compute gradient update from the residual
    v = TimeData(name='v', shape=model.shape,
                 time_order=2, space_order=order)
    residual = rec_true.data[:] - rec_smooth.data[:]
    op_gradient(u=u, v=v, grad=grad, rec=residual, m=m_smooth)
```

Rapid propagator development and integration

- Test and verify in Python
- Acoustic Operators in $<$ 20 lines
- TTI Operators in $<$ 100 lines [1]
- RTM setup in $<$ 100 lines
- Variable stencil order

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

Performance optimisations

- Common sub-expression elimination - C compilers do it already but it is quicker to do it at the higher level
- Heuristic refactorisation e.g. $0.3 * a + ... + 0.3 * b => 0.3 * (a + b)$
  - Impact: TTI, space order 16: 6680 5760
- Alias detection e.g. `sin(phi[i,j,k]), sin(phi[i-1,j-1,k-1])`
- Heuristic hoisting of time-invariants
- Loop fission + elemental functions (register locality)
- Padding + data alignment (split loads)
- Loop blocking in 1D/2D/3D (no time yet) - with autotuned block sizes
- SIMD vectorisation through pragmas (intrinsics not required)
- Thread parallelism through OpenMP pragmas
- YASK backend (WIP)

Verification

Verification of the generated code:

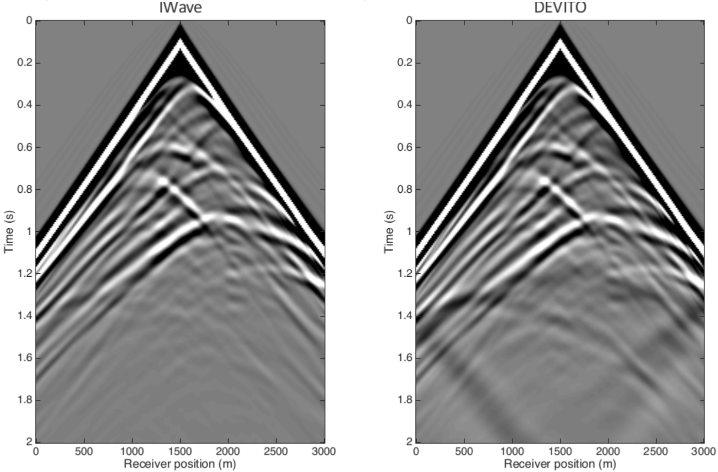
- Extensive unit-testing already in place with continuous integration (Travis)
- Adjoint test [1]
  - For any $x \in$span$(P_s A^T P_r^T)$, $y \in$span$(P_r A^T P_s^T)$
  - $< P_r A^T P_s^T x, y > - < x, P_s A^T P_r^T y >= 0$
  - Passes with at-least 8 matching significant digits for 2D and 3D with 2,4,6,8,10,12th 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 self-verification [2]

[1] Louboutin, M., Lange, M., Luporini, F., Kukreja, N., Herrmann, F., Velesko, P. and Gorman, G. (2017). Code generation from symbolic finite-difference for geophysical exploration. Geoscientific Model Development. (under review)

[2] Huckelheim, J., Luo, Z., Luporini, F., Kukreja, N., Lange, M., Gorman, G., Siegel, S., Dwyer, M. and Hovland, P. (2017). Towards built-in verification in high-performance stencil code generation. Correctness 2017: First International Workshop on Software Correctness for HPC Applications. (accepted for presentation)

Verification

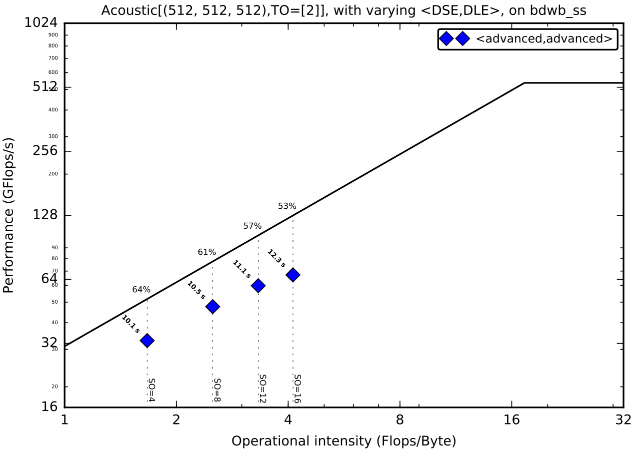
Comparison with a reference implementation - IWAVE [1]



[1] Symes, W.W., 2015. IWAVE structure and basic use cases. THE RICE INVERSION PROJECT, p.85.

Performance

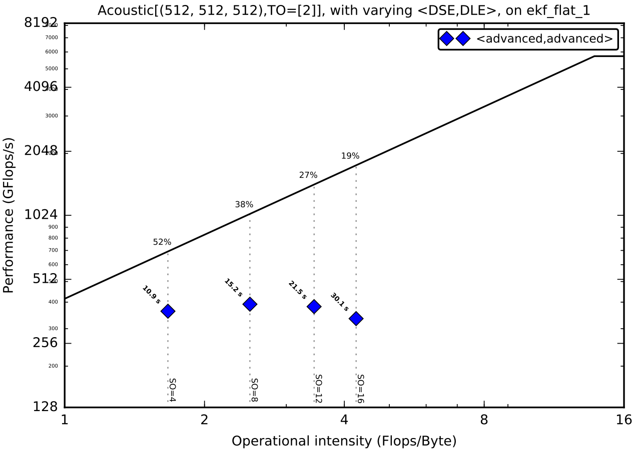
- Performance of acoustic forward operator
- Intel®Xeon™E5-2620 v4 2.1Ghz Broadwell (8 cores)
- Model size $512 \times 512 \times 512$, $t_n = 250$



Imperial College London

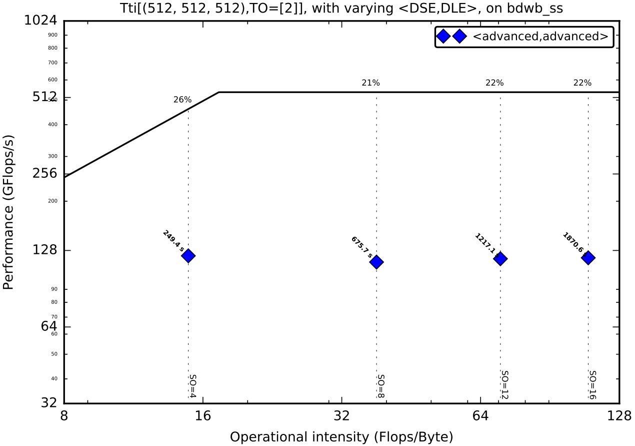
Performance

- Performance of acoustic forward operator
- Intel®Xeon Phi™7650 Knightslanding (68 cores) Quadrant Mode
- Model size $512 \times 512 \times 512$, $t_n = 3000$



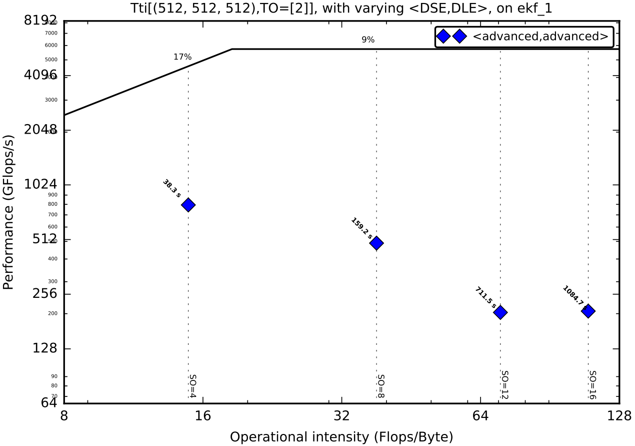
Performance

- Performance of TTI forward operator
- Intel®Xeon™E5-2620 v4 2.1Ghz Broadwell (8 cores)
- Model size $512 \times 512 \times 512$, $t_n = 250$



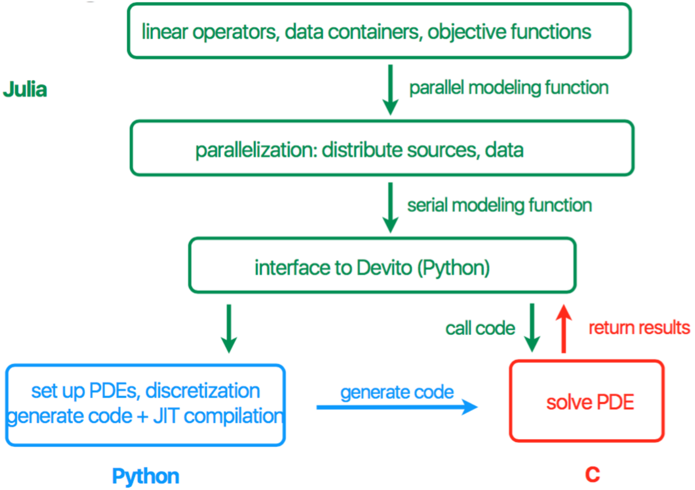
Performance

- Performance of TTI forward operator
- Intel®Xeon Phi™7650 Knightslanding (68 cores) Quadrant mode
- Model size $512 \times 512 \times 512$, $t_n = 3000$



Imperial College London

Integration with other codes



[1]

[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

Conclusions

OPESCI

- 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:
  - Yask Backend
  - MPI parallelism for larger models
  - Checkpointing
  - Better boundary conditions

Thank you

OPESCI

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

OPESCI

intel®

EPSRC
Pioneering research and skills

SINBAD

Imperial College London