Devito: Symbolic Math for Automated Fast Finite Difference Computations

Navjot Kukreja[1] M. Lange[1] M. Louboutin[2] F. Luporini[1] G. Gorman[1]

February 27, 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

The 1D acoustic wave equation

$$\frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0 \qquad (1)$$

Discretized:

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + \frac{c^2 dt^2}{h^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \qquad (2)$$

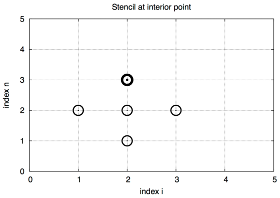


Figure 1: Mesh in space and time for a 1D wave equation

- Mathematically simple method for solving PDEs directly
- Calculate derivatives of any order with relative simplicity
- Easier to optimize performance than FEM codes

Pure Python Implementation

OPESCI

```
k = (c*dt/h)**2
for t in range(2,nsteps-1):
     p[xs,zs,t] = s[t]
     for z in range(1,nz-1):
         for x in range(1,nx-1):
             p[x,z,t] += 2*p[x,z,t-1] - p[x,z,t-2] + k*(p[x+1,z,t
                  -1]-4*p[x,z,t-1]+p[x-1,z,t-1]+p[x,z+1,t-1]+p[x,z-1,
                  t-1])
```

Imperial College London

Why does it need to be fast?

- Large number of operations: ≈ 6000 FLOPs per loop iteration of a 16th order TTI kernel
- Realistic problems have large grids: 1580 x 1580 x 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)

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

Why automated

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
- Change of physics
  - Anisotropy - VTI/TTI
  - Elastic equation
- Boundary conditions
- Continuous acquisition

SymPy - Symbolic computation in Python

OPESCI

- Symbolic computer algebra system written in pure Python
- Features
  - Complex symbolic expressions as Python object trees
  - Symbolic manipulation routines and interfaces
  - Convert symbolic expressions to numeric functions
    - Python or NumPy functions
    - C or Fortran kernels

For specialized domains generating C code is not enough!

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:

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \nabla u = 0 \tag{3}$$

can be written as

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

Devito auto-generates optimized C kernel code

Devito

Real-world applications need more than PDE solvers

- File I/O and support for large datasets
- Non-PDE kernel code e.g. sparse point interpolation
- 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

Devito Symbolic Engine (DSE)

- Symbolic manipulations to reduce the number of required flops
- Common subexpression elimination - enables further optimizations and improves compilation time
- Heuristic refactorization - $0.3 * a + ... + 0.3 * b => 0.3 * (a + b)$
- Approximations - Replace transient functions (e.g. trigonometric) with polynomial approximations
- Heuristic hoisting of time-invariants

Devito Loop Engine (DLE)

- Optimizations for parallelism and memory
- SIMD
- OpenMP - including *collapse*
- Loop blocking (only in spatial dimensions)
- Loop fission + elemental functions
- Non-temporal stores
- Padding + data alignment

Devito Architecture

OPESCI

sympy.Equation

SymbolicData

- **sympy.Function**
- Data allocation

Dimension

- **Iteration space**
- Function index

Expression

- SymPy expr traversal
- Symbolic properties (dims, vars, etc.)
- Substitution
- C code generation

Iteration

- **Loop instance**
- Loop limits (variable vs. fixed)
- Vectorisation
- OpenMP pragmas
- Substitution
- C code generation

Operator

- Universal entry-point: convert SymPy expressions
- Manage codegen, compilation and execution
- Manage cross expression data (eg. temporaries)
- Configurable by users (eg. DSE, blocking, etc.)
- Composable/mergeable

Devito Symbolic Engine

Devito Loop Engine

Introduction

Devito

Example

Example - 2D diffusion equation

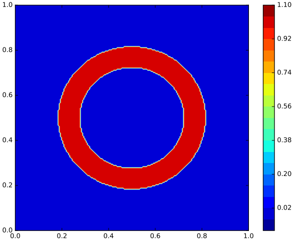
OPESCI

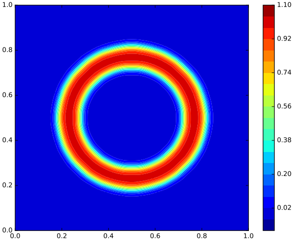
To illustrate let's consider the 2D diffusion equation:

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

Example: Smoothing a sharp-edged ring

- Finite difference with 5-point stencil





Imperial College London

OPESCI

We can solve this using Python (slow) . . .

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

Vectorized NumPy (faster) . . .

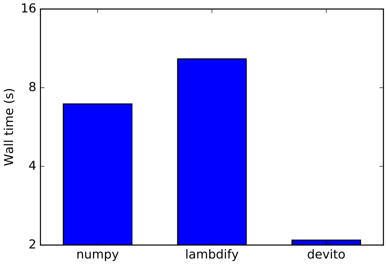
```python
for ti in range(timesteps):
  t0 = ti % 2
  t1 = (ti + 1) % 2
  # Vectorised version of the diffusion stencil
  uxx = (u[t0,2:,1:-1]-2*u[t0,1:-1,1:-1]+u[t0,:-2,1:-1])/dx2
  uyy = (u[t0,1:-1,2:]-2*u[t0,1:-1,1:-1]+u[t0,1:-1,:-2])/dy2
  u[t1,1:-1,1:-1] = u[t0,1:-1,1:-1] + a * dt * (uxx + uyy)
```

OPESCI

Solve symbolically in Devito:

```
u = TimeData(name='u', shape=(nx, ny),
             time_order=1, space_order=2)
u.data[0, :] = ring_initial(spacing=dx)
eqn = Eq(u.dt, a * (u.dx2 + u.dy2))
stencil = solve(eqn, u.forward)[0]
op = Operator(stencils=Eq(u.forward, stencil),
              subs={h: dx, s: dt}, nt=timesteps)
op.apply()
```

Single core benchmark:



Imperial College London

Typical loop nest - Devito code

```
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)
```

Generated Code

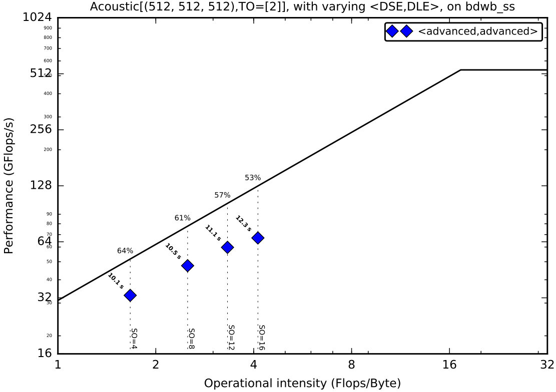
```
#pragma omp parallel
  {
    _MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
    _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
  }
  #pragma omp parallel
  {
    for (int i4 = 0; i4<329; i4+=1)
    {
      struct timeval start_main, end_main;
      #pragma omp master
      gettimeofday(&start_main, 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]);
            }
          }
```

Imperial College London

```
#pragma omp master
  {
    gettimeofday(&end_main, NULL);
    timings->main += (double)(end_main.tv_sec-start_main.tv_sec)+(double)(end_main.tv_usec-
          start_main.tv_usec)/1000000;
  }
  #pragma omp single
  {
    struct timeval start_post_stencils0, end_post_stencils0;
    gettimeofday(&start_post_stencils0, NULL);
    for (int p_src = 0; p_src < 1; p_src += 1)
    {
      u[i4][(int)(floor(5.0e-2F*src_coords[p_src][0])) + 40][(int)(floor(5.0e-2F*src_coords[
            p_src][1])) + 40][(int)(floor(5.0e-2F*src_coords[p_src][2])) + 40] = 9.2416F
            *(-1.25e-4F*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src][0])) +
            src_coords[p_src][0])*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src][1])
            ) + src_coords[p_src][1])*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src
            ][2])) + src_coords[p_src][2]) + 2.5e-3F*(float)(-2.0e+1F*(int)(floor(5.0e-2F*
            src_coords[p_src][0])) + src_coords[p_src][0])*(float)(-2.0e+1F*(int)(floor(5.0e
            -2F*src_coords[p_src][1])) + src_coords[p_src][1]) + 2.5e-3F*(float)(-2.0e+1F*(
            int)(floor(5.0e-2F*src_coords[p_src][0])) + src_coords[p_src][0])*(float)(-2.0e
            +1F*(int)(floor(5.0e-2F*src_coords[p_src][2])) + src_coords[p_src][2]) - 5.0e-2F
            *(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src][0])) + src_coords[p_src
            ][0]) + 2.5e-3F*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src][1])) +
            src_coords[p_src][1])*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src][2])
            ) + src_coords[p_src][2]) - 5.0e-2F*(float)(-2.0e+1F*(int)(floor(5.0e-2F*
            src_coords[p_src][1])) + src_coords[p_src][1]) - 5.0e-2F*(float)(-2.0e+1F*(int)(
            floor(5.0e-2F*src_coords[p_src][2])) + src_coords[p_src][2]) + 1)*src[i4][p_src
            ]/m[(int)(floor(5.0e-2F*src_coords[p_src][0])) + 40][(int)(floor(5.0e-2F*
            src_coords[p_src][1])) + 40][(int)(floor(5.0e-2F*src_coords[p_src][2])) + 40] +
            u[i4][(int)(floor(5.0e-2F*src_coords[p_src][0])) + 40][(int)(floor(5.0e-2F*
            src_coords[p_src][1])) + 40][(int)(floor(5.0e-2F*src_coords[p_src][2])) + 40];
       u[i4][(int)(floor(5.0e-2F*src_coords[p_src][0])) + 40][(int)(floor(5.0e-2F*src_coords
             [p_src][1])) + 41][(int)(floor(5.0e-2F*src_coords[p_src][2])) + 40] = 9.2416F
             *(1.25e-4F*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src][0])) +
             src_coords[p_src][0])*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[p_src
             ][1])) + src_coords[p_src][1])*(float)(-2.0e+1F*(int)(floor(5.0e-2F*src_coords[
             p_src][2])) + src_coords[p_src][2]) - 2.5e-3F*(float)(-2.0e+1F*(int)(floor(5.0e
             -2F*src_coords[p_src][0])) + src_coords[p_src][0])*(float)(-2.0e+1F*(int)(floor
             (5.0e-2F*src_coords[p_src][1])) + src_coords[p_src][1]) - 2.5e-3F*(float)(-2.0e
             +1F*(int)(floor(5.0e-2F*src_coords[p_src][1])) + src_coords[p_src][1])*(float)
```

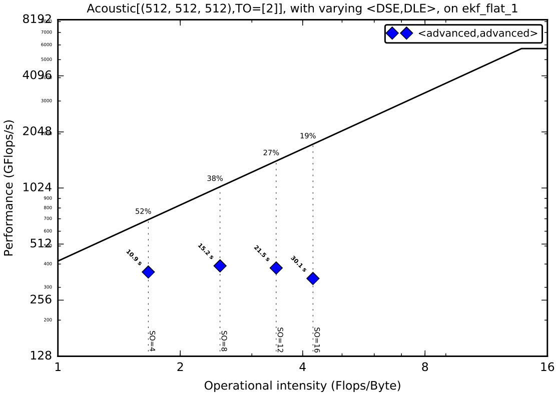
Performance

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



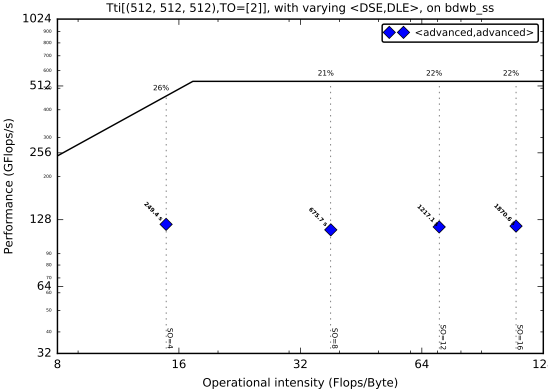
Imperial College London

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



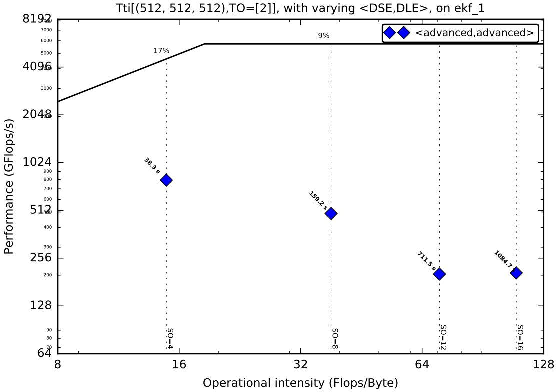
Performance

- Performance of TTI forward operator
- Intel®Xeon™E5-2620 v4 2.1Ghz Broadwell (8 cores per socket, single socket)
- 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 $512 \times 512 \times 512$, $t_n = 3000$



Conclusions

- 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
  - Integrate automated verification tools to catch compiler bugs [1]

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

Thank you

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

Poster

- Wednesday, March 1 4:30 PM - 6:30 PM
- Minisymposterium: Software Productivity and Sustainability for CSE and Data Science (*Galleria*)