GPU Support for Automatic Generation of Finite-Differences Stencil Kernels

Vitor H M Rodrigues [1] Lucas Cavalcante [1] Maelso Pereira [1]
István Reguly [2] Fabio Luporini [3] Gerard Gorman [3]
Samuel Xavier de Souza [1]

[1]Universidade Federal do Rio Grande do Norte
[2]Pazmany Peter Catholic University
[3]Imperial College London

September 26, 2019

Motivation

- Many physic's problems involve the solution of Partial Difference Equations (PDE)

Poisson's equation: $\nabla^2 u = f(x, y, z)$

Heat flow equation: $k \frac{\partial^2 u}{\partial^2 x} = \frac{\partial u}{\partial t}, k > 0$

Wave equation: $\nabla^2 u = \frac{1}{v^2} \frac{\partial^2 u}{\partial^2 t}$

Problem

It is difficult to write computational solutions to these problems and solve them efficiently.

Looking for...

- A solution that will enable field specialists to solve their computational demands without digging into programming technicalities.

Domain Specific Language

1. Provides a high-level abstraction for a specific domain
2. Divides the matter of problem definition and problem computation
3. Enables execution of the same code in different architectures with, ideally, no modifications.
4. Examples of DSL for solving PDEs: Devito, Firedrake, FEniCS, and esys-Escript.

Devito

- PDE solver which uses Finite Difference method
- Uses DSL for problem specification in Python
- Automatically generates code in C
- Symbols and loop optimizations: basic, advanced and aggressive

Devito references Luporini et al. 2018[1] and Louboutin et al. 2019[2]

Devito - Example

3D Acoustic Wave Equation with damping

$$m(x, y, z)\ \frac{d^2u(t,x,y,z)}{dt^2} - \nabla^2 u(t, x, y, z) + \eta \frac{du(t,x,y,z)}{dt} = 0$$

Representation using Devito with Sympy

eqn = *model.m* * *u.dt2* - *u.laplace* + *model.damp* * *u.dt*

Devito - Generated C code

```
for (int x = x_m; x <= x_M; x += 1)
  {
    #pragma omp simd aligned(damp,m,u:32)
    for (int y = y_m; y <= y_M; y += 1)
    {
      float r0 = 1.0F*dt*m[x+2][y+2][z+2] +
                 5.0e-1F*(dt*dt)*damp[x+1][y+1][z+1];

      u[t1][x+2][y+2][z+2] =
        1.0F*(-dt*m[x+2][y+2][z+2]*u[t2][x+2][y+2][z+2]/r0 +
              (dt*dt*dt)*u[t0][x + 1][y + 2][z + 2]/r0 +
              (dt*dt*dt)*u[t0][x + 2][y + 1][z + 2]/r0 +
              (dt*dt*dt)*u[t0][x + 2][y + 2][z + 1]/r0 +
              (dt*dt*dt)*u[t0][x + 2][y + 2][z + 3]/r0 +
              (dt*dt*dt)*u[t0][x + 2][y + 3][z + 2]/r0 +
              (dt*dt*dt)*u[t0][x + 3][y + 2][z + 2]/r0) +
        2.0F*dt*m[x+2][y+2][z+2]*u[t0][x+2][y+2][z+2]/r0 +
        5.0e-1F*(dt*dt)*damp[x+1][y+1][z+1]*u[t2][x+2][y+2][z+2]/r0 -
        6.0F*dt*dt*dt*u[t0][x + 2][y + 2][z + 2]/r0;
      }
    }
```

Algorithm 1: A code segment auto generated from Devito using core backend with no optimization. Represents the propagation update for stencil of space order 2.

Stencil - 2^{nd} space order

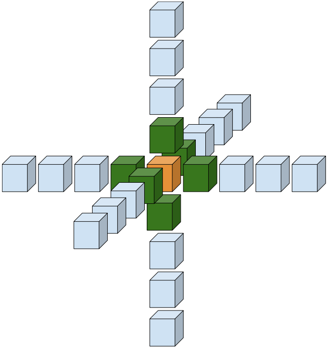


Figure: 2^{nd} order stencil

$$
\begin{aligned}
u_{t+1,x,y,z} = \\
&\alpha * u_{t-1,x,y,z} + \beta * u_{t,x,y,z} + \\
&\gamma * \big(u_{t,x+1,y,z} + u_{t,x-1,y,z} + \\
&\quad u_{t,x,y+1,z} + u_{t,x,y-1,z} + \\
&\quad u_{t,x,y,z+1} + u_{t,x,y,z-1} \big)
\end{aligned}
$$

Stencil - 8^{th} space order

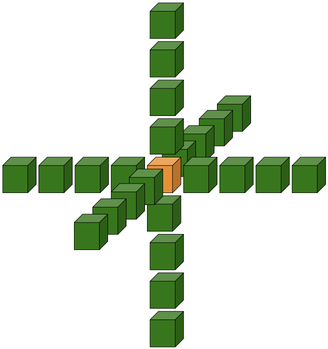
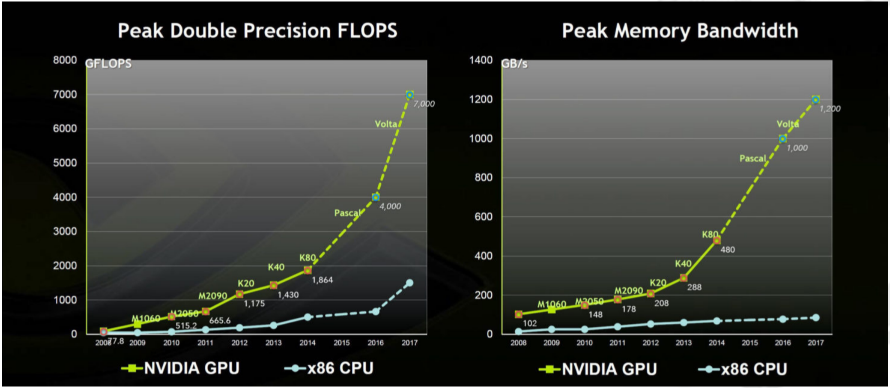


Figure: 8^{th} order stencil

$$
\begin{aligned}
u_{t+1,x,y,z} = \\
&\alpha * u_{t-1,x,y,z} + \beta * u_{t,x,y,z} + \\
&\gamma * (u_{t,x+1,y,z} + u_{t,x-1,y,z} + u_{t,x,y+1,z} + \\
&\qquad u_{t,x,y-1,z} + u_{t,x,y,z+1} + u_{t,x,y,z-1}) + \\
&\delta * (u_{t,x+2,y,z} + u_{t,x-2,y,z} + u_{t,x,y+2,z} + \\
&\qquad u_{t,x,y-2,z} + u_{t,x,y,z+2} + u_{t,x,y,z-2}) + \\
&\epsilon * (u_{t,x+3,y,z} + u_{t,x-3,y,z} + u_{t,x,y+3,z} + \\
&\qquad u_{t,x,y-3,z} + u_{t,x,y,z+3} + u_{t,x,y,z-3}) + \\
&\zeta * (u_{t,x+4,y,z} + u_{t,x-4,y,z} + u_{t,x,y+4,z} + \\
&\qquad u_{t,x,y-4,z} + u_{t,x,y,z+4} + u_{t,x,y,z-4})
\end{aligned}
$$

Devito works for CPU...



Comparison between x86 CPU and NVIDIA GPUs - (chart from NVDIA presentation)

How can we support GPU...

- So, we now can modify Devito to generate code for GPU architecture
- Which parallel programming platform: CUDA, OpenCL, OpenACC...?

OPS

- Oxford Parallel Structured Software (OPS)
- Framework for the execution of multi-block structured mesh applications
- Uses source-to-source translation
- Generates code for the following platforms
    - CUDA
    - OpenACC
    - OpenCL
    - MPI
    - OpenMP

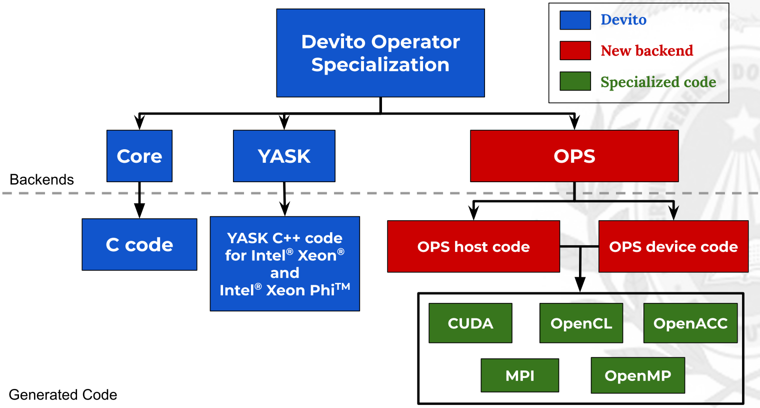
OPS reference Reguly et al. 2014[3]

Goal

To use the GPU's capability to solve PDEs problems

- To use Devito for problem specification and optimizations
- To use OPS framework to generate code targeted for GPU platform
- To integrate Devito and OPS to generate GPU code from a symbolic representation

Diagram displaying Devito and OPS integration



Devito-OPS integration

Steps implemented

1. Identified parallelizable code
2. Translated Devito expressions into OPS expressions
3. Generated kernel function
4. Partial generation of host code

Devito-OPS integration

Step 1 - Devito simple example

exemple.py

```
from devito import Eq, Grid, TimeFunction, Operator

grid_2d = Grid(shape=(4, 4))
v = TimeFunction(name='v', grid=grid_2d, time_order=2, save=10)
equation = Eq(v.forward, v+1)
operator = Operator(equation)

print(operator)
```

Devito-OPS integration

Step 2 - Code generated with backend core

```c
#define _POSIX_C_SOURCE 200809L
#include "stdlib.h"
#include "math.h"
#include "sys/time.h"
#include "xmmintrin.h"
#include "pmmintrin.h"

struct dataobj {
  void *restrict data;
  int * size;
  int * npsize;
  int * dsize;
  int * hsize;
  int * hofs;
  int * oofs;
};

struct profiler {
  double section0;
};


int Kernel(struct dataobj *restrict v_vec, const int time_M, const int time_m, struct profiler * timers, const int x_M, const int x_m, const int
y_M, const int y_m) {
  float (*restrict v)[v_vec->size[1]][v_vec->size[2]] __attribute__ ((aligned (64))) = (float (*)[v_vec->size[1]][v_vec->size[2]]) v_vec->data;
  /* Flush denormal numbers to zero in hardware */
  _MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
  _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
  for (int time = time_m; time <= time_M; time += 1){
    struct timeval start_section0, end_section0;
    gettimeofday(&start_section0, NULL);
    /* Begin section0 */
    for (int x = x_m; x <= x_M; x += 1) {
      #pragma omp simd aligned(v:32)
      for (int y = y_m; y <= y_M; y += 1) {
        v[time + 1][x + 1][y + 1] = v[time][x + 1][y + 1] + 1;
      }
    }
    /* End section0 */
    gettimeofday(&end_section0, NULL);
    timers->section0 += (double)(end_section0.tv_sec-start_section0.tv_sec)+(double)(end_section0.tv_usec-start_section0.tv_usec)/1000000;
  }
  return 0;
}
```

Devito-OPS integration

Step 3 - Identify parallelizable region

```
#define _POSIX_C_SOURCE 200809L
#include "stdlib.h"
#include "math.h"

int Kernel(float * v, const int time_M, const int time_m, const int x_M,
            const int x_m, const int y_M, const int y_m)
{
  for (int time = time_m; time <= time_M; time += 1)
  {
    for (int x = x_m; x <= x_M; x += 1)
    {
      for (int y = y_m; y <= y_M; y += 1)
      {
        v[time + 1][x + 1][y + 1] = v[time][x + 1][y + 1] + 1;
      }
    }
  }
  return 0;
}
```

Devito-OPS integration

Step 4 - Replacing with OPS API

```
#define _POSIX_C_SOURCE 200809L
#include "stdlib.h"
#include "math.h"

int Kernel(float * v, const int time_M, const int time_m, const int x_M,
            const int x_m, const int y_M, const int y_m)
{
  for (int time = time_m; time <= time_M; time += 1)
  {
```

```
  }
  return 0;
}
```

Devito-OPS integration

Step 5 - OPS API - ops_par_loop

```
#define _POSIX_C_SOURCE 200809L
#include "stdlib.h"
#include "math.h"

int Kernel(float * v, const int time_M, const int time_m, const int x_M,
            const int x_m, const int y_M, const int y_m)
{
  for (int time = time_m; time <= time_M; time += 1)
  {

    ops_par_loop(OPS_Kernel_0, "OPS_Kernel_0", block, 2, {x_m, x_M, y_m, y_M},
                 ops_arg_dat(v_dat[t0], 1, S2D_VT0_1PT, "float", OPS_READ),
                 ops_arg_dat(v_dat[t1], 1, S2D_VT1_1PT, "float", OPS_WRITE));

  }
  return 0;
}
```

Devito-OPS integration

Step 6 - Device Code

```
void OPS_Kernel_0(const ACC<float> & vt0, ACC<float> & vt1)
{
  vt1(0, 0) = vt0(0, 0) + 1;
}
```

Experiment

3D Acoustic Wave Propagation

- Time step 0.001 *s*, Total time: 30 *s*
- Domain size of: 1 *km* x 1 *km* x 1 *km* with grid points: 32^3, 64^3, 128^3, 256^3, 512^3
- Ricker source injection with peak frequency 10 *Hz*
- Space Order: 4, 8, 12, 16 and 24
- Single layer velocity model of 2 *km/s*
- Absorbing boundary

Compilation and Execution

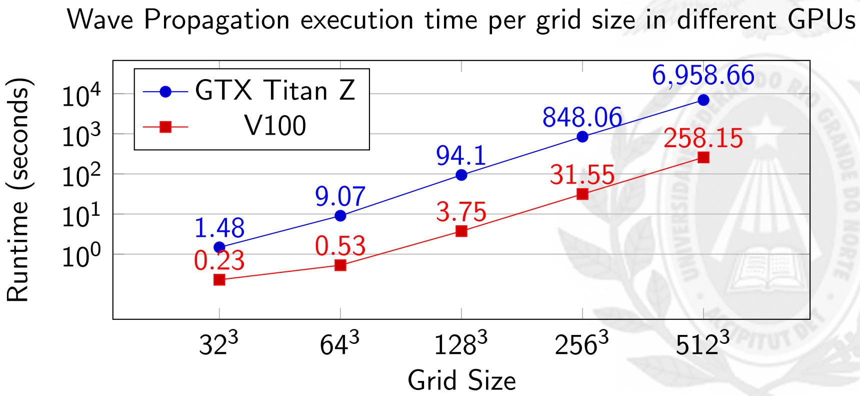
- Compiler: *nvcc 10.1*
- Flags: *-Xcompiler="-std=c99" -O3*
- Median of 5 executions

Architectures Evaluated

	Titan Z	Tesla V100
Memory Bandwidth (GB/s)	336 x2	900
Precision Peak Performance (GFLOPS)	4746	14000
Double Precision Peak Performance (GFLOPS)	1582	7000
Memory (GB)	6 x2	16

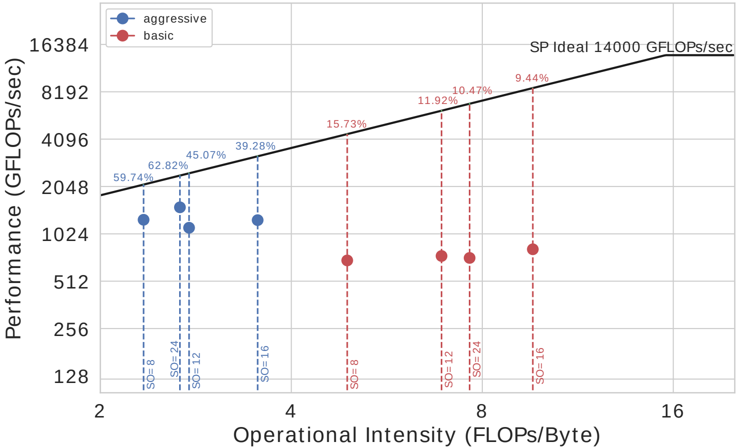
Graphical cards specification from vendors

Execution Time

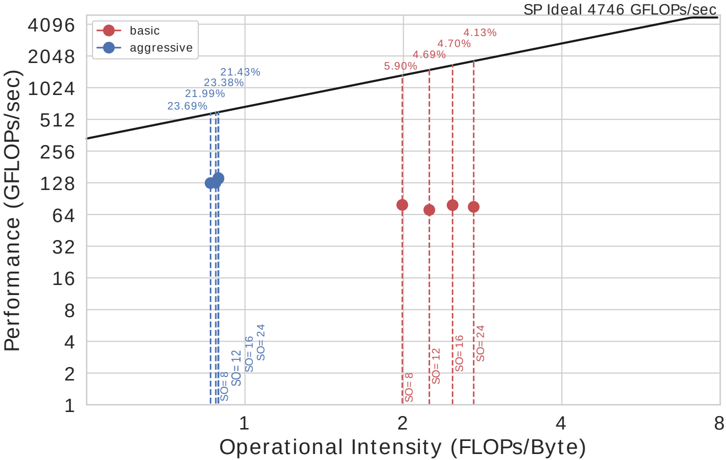


Execution time in different GPUs of an acoustic isotropic 3D wave propagation with 30,000 time steps for 4^{th} space order stencil.

Roofline Model for a Tesla V100 GPU



Roofline Model for a GTX Titan Z



Challenges

Challenges

- To avoid unnecessary memory copies from CPU to GPU
- To keep up with OPS and Devito upgrades

Conclusion

- Created an extension of Devito to enable code generation for the OPS syntax
- Successfully tested the solution with wave propagation algorithm
- Observed that the execution time from approximately 2 hours in GTX Titan Z took 4 minutes in V100
- Achieved up to 63% of the peak performance on V100

Where to find...

Source code

- Devito: https://github.com/opesci/devito
- OPS: https://github.com/OP-DSL/OPS

Website

- https://www.devitoproject.org/index.html
- https://op-dsl.github.io/

Talk to us

- Devito slack community: devitocodes.slack.com

References I

[1] 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," jul 2018.

[2] 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.

[3] I. Z. Reguly, G. R. Mudalige, M. B. Giles, D. Curran, and S. McIntosh-Smith, "The OPS domain specific abstraction for multi-block structured grid computations," in *Proceedings of WOLFHPC 2014: 4th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing - Held in Conjunction with SC 2014: The International Conference for High Performance Computing, Networking, Stor*, pp. 58–67, 2014.