Automated Temporal Blocking in the Devito DSL and Compiler framework

***George Bisbas*[1,2]**, Fabio Luporini[3], Mathias Louboutin[4], Rhodri Nelson[2], Gerard Gorman[2,3], Paul H.J. Kelly[1]

[1]Imperial College London, Dept. of Computing
[2]Imperial College London, Dept. of Earth Sciences and Engineering
[3]Devito Codes, UK
[4]Georgia Institute of Technology, Atlanta, USA

MS243: Stencil Computation for Scientific Applications, SIAM CSE23, Amsterdam

Our motivation:

- **Motivation**: speed up computationally expensive scientific simulations involving the solution of PDEs modelling wave equations through explicit finite-difference methods

- **Cache blocking has been profitable for stencil computations**

- **Temporal cache blocking has been even more profitable!**
    - **Rarely applied in production**
    - **Challenging to apply**
    - **Few libraries, not straightforward**
    - **Why miss out?**

- Through Devito framework we offer the opportunity to go from textbook-like math to HPC temporal blocking code

- **Improved performance** without the fuss!

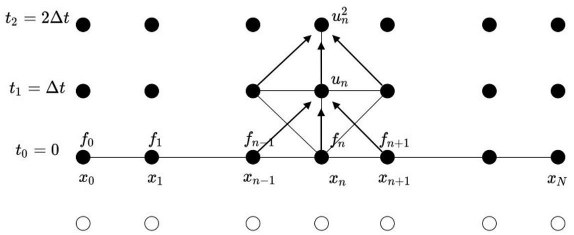
- **Q: Do I need to have CS skills to get perf?**

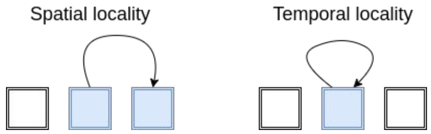
$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \qquad 0 < x < L$$

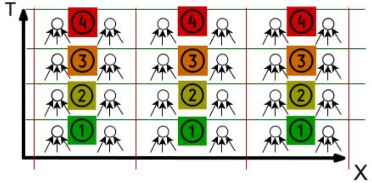
$$\text{Boundary Conditions}: \quad u(0,t) = 0; \quad u(L,t) = 0$$

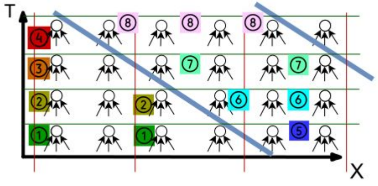
$$\text{Initial Conditions}: \quad u(x,0) = f(x)$$

$$\frac{\partial u}{\partial t}(x,0) = g(x)$$









Scientific simulations are demanding

🚧 **Very complex to model** (complicated PDEs, BCs, external factors, complex geometries)

✅Software offering **high-level, high-productivity DSLs**
✅Let **domain experts** navigate their design space

🚧 **Resource-demanding** ($O(10^3)$ FLOPs per loop iteration, high memory pressure, 3D grids with > 10^9 grid points, often $O(10^3)$ time steps, inverse problems, ≈O(billions) TFLOPs. Which means days, or weeks, or months on supercomputers!

✅**Offer** automated optimisations and **efficient codegen for HPC** workloads
✅**Higher resolution in space and time** opens up compelling new applications
✅**Unlocks** ever-increasing application **value**

- Complex FD-stencil
- Just a part of these codes!
- No one wants to write it
- No one wants to optimise it
- No one wants to debug

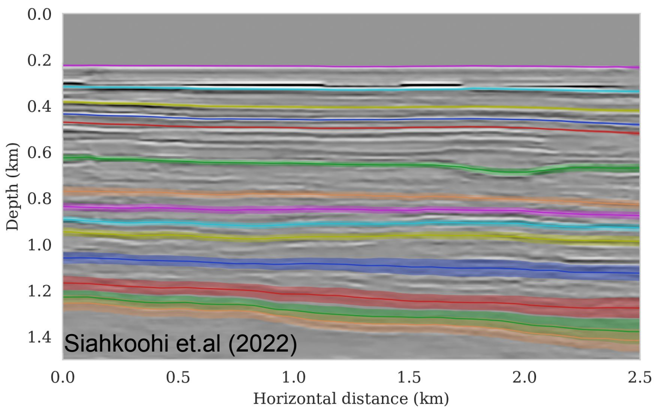
```
void kernel(…) {
  …
  <impenetrable code with aggressive performance optimizations, manually applied, full-time human resources, less reproducibility, debugging nightmares>
  …}
```

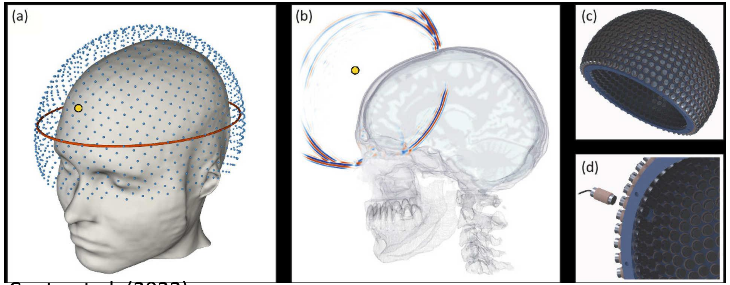
```
 for (int time = time_m, t0 = (time)%(3), t1 = (time + 2)%(3), t2 = (time + 1)%(3);
time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 2)%(3), t2 = (time + 1)%(3))
  {
    /* Begin section0 */
    START_TIMER(section0)
    for (int x0_blk0 = x_m; x0_blk0 <= x_M; x0_blk0 += x0_blk0_size)
    {
      for (int y0_blk0 = y_m; y0_blk0 <= y_M; y0_blk0 += y0_blk0_size)
      {
        for (int x = x0_blk0; x <= MIN(x_M, x0_blk0 + x0_blk0_size - 1); x += 1)
        {
          for (int y = y0_blk0; y <= MIN(y_M, y0_blk0 + y0_blk0_size - 1); y += 1)
          {
            #pragma omp simd aligned(damp,u,vp:32)
            for (int z = z_m; z <= z_M; z += 1)
            {
              float r10 = 1.0F/(vp[x + 12][y + 12][z + 12]*vp[x + 12][y + 12][z +
12]);
              u[t2][x + 12][y + 12][z + 12] = (r10*(-r8*(-2.0F*u[t0][x + 12][y + 12][z
+ 12]) - r8*u[t1][x + 12][y + 12][z + 12]) + r9*damp[x + 12][y + 12][z + 12]*u[t0][x +
12][y + 12][z + 12] + 2.67222496e-7F*(-u[t0][x + 6][y + 12][z + 12] - u[t0][x + 12][y
+ 6][z + 12] - u[t0][x + 12][y + 12][z + 6] - u[t0][x + 12][y + 12][z + 18] - u[t0][x
+ 12][y + 18][z + 12] - u[t0][x + 18][y + 12][z + 12]) + 4.61760473e-6F*(u[t0][x + 7]
[y + 12][z + 12] + u[t0][x + 12][y + 7][z + 12] + u[t0][x + 12][y + 12][z + 7] + u[t0]
[x + 12][y + 12][z + 17] + u[t0][x + 12][y + 17][z + 12] + u[t0][x + 17][y + 12][z +
12]) + 3.96825406e-5F*(-u[t0][x + 8][y + 12][z + 12] - u[t0][x + 12][y + 8][z + 12] -
u[t0][x + 12][y + 12][z + 8] - u[t0][x + 12][y + 12][z + 16] - u[t0][x + 12][y + 16][z
+ 12] - u[t0][x + 16][y + 12][z + 12]) + 2.35155796e-4F*(u[t0][x + 9][y + 12][z + 12]
+ u[t0][x + 12][y + 9][z + 12] + u[t0][x + 12][y + 12][z + 9] + u[t0][x + 12][y + 12]
[z + 15] + u[t0][x + 12][y + 15][z + 12] + u[t0][x + 15][y + 12][z + 12]) +
1.19047622e-3F*(-u[t0][x + 10][y + 12][z + 12] - u[t0][x + 12][y + 10][z + 12] - u[t0]
[x + 12][y + 12][z + 10] - u[t0][x + 12][y + 12][z + 14] - u[t0][x + 12][y + 14][z +
12] - u[t0][x + 14][y + 12][z + 12]) + 7.6190478e-3F*(u[t0][x + 11][y + 12][z + 12] +
u[t0][x + 12][y + 11][z + 12] + u[t0][x + 12][y + 12][z + 11] + u[t0][x + 12][y + 12]
[z + 13] + u[t0][x + 12][y + 13][z + 12] + u[t0][x + 13][y + 12][z + 12]) -
3.97703713e-2F*u[t0][x + 12][y + 12][z + 12])/(r10*r8 + r9*damp[x + 12][y + 12][z +
12]);
            }
          }
        }
      }
    }
    STOP_TIMER(section0,timers)
    /* End section0 */
```

Introducing Devito

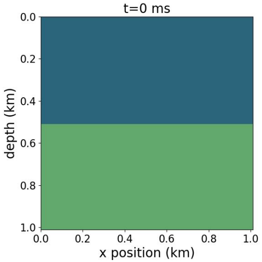
- **Devito is a DSL and compiler framework** for finite difference and stencil computations
- **Solving PDEs** using the **finite-difference method for structured grids** (but not limited to this!)
- Users model in the high-level DSL using symbolic math abstraction, and the compiler auto-generates HPC optimized code
- Inter(-national, -institutional,-disciplinary), lots of users from academia and industry
- Real-world problem simulations! (CFD, seismic/medical imaging, finance, tsunamis)

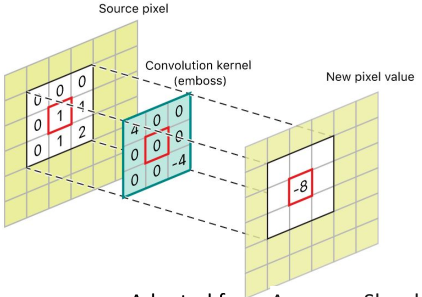
CI-core passing | CI-mpi passing | CI-gpu passing | codecov 88% | chat on slack | benchmarked by asv | pypi package 4.8.0 | launch binder | docker images





Cueto et.al. (2022)



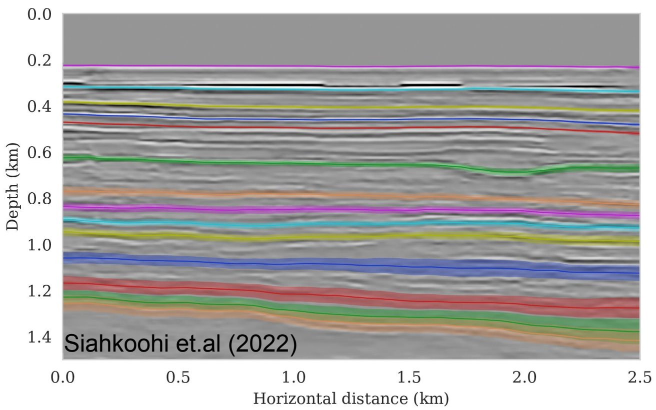


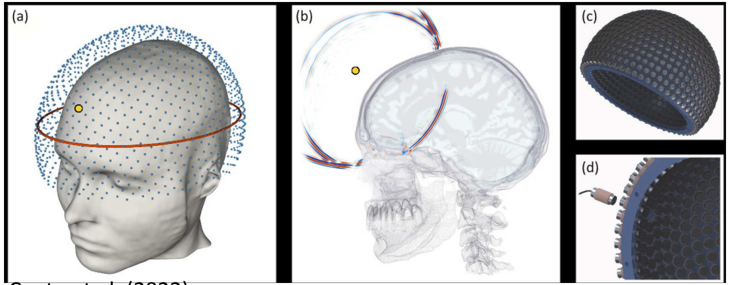
Adapted from Aryaman Sharda

Introducing Devito

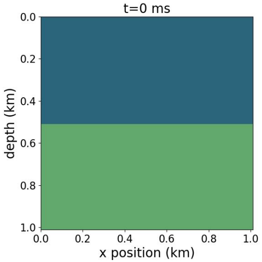
- **Open source** - MIT lic. - Try now! https://github.com/devitocodes/devito
- **Compose with** packages from the Python ecosystem (e.g. PyTorch, NumPy, Dask, TensorFlow)
- Best practices in **software engineering**: extensive software testing, code verification, CI/CD, regression tests, documentation, tutorials and PR code review
- Actual compiler technology (not a S2S translator or templates!)

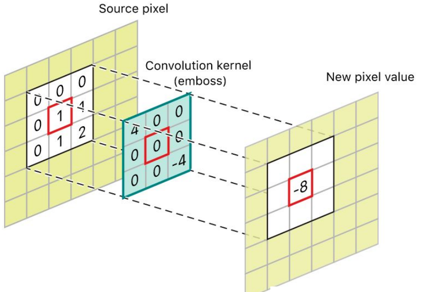
CI-core passing | CI-mpi passing | CI-gpu passing | codecov 88% | chat on slack | benchmarked by asv | pypi package 4.8.0
launch binder | docker images





Cueto et.al. (2022)





Adapted from Aryaman Sharda

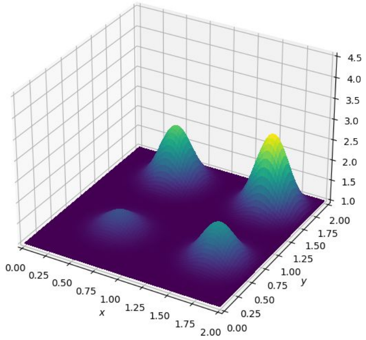
An example from textbook maths to via Devito DSL

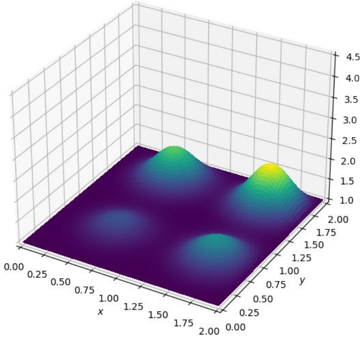
2D Heat diffusion modelling

$$u(x,1) = 0$$

$$u(0,y) = 0 \qquad \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = \frac{\partial u}{\partial t}(x,y) \qquad u(1,y) = 0$$

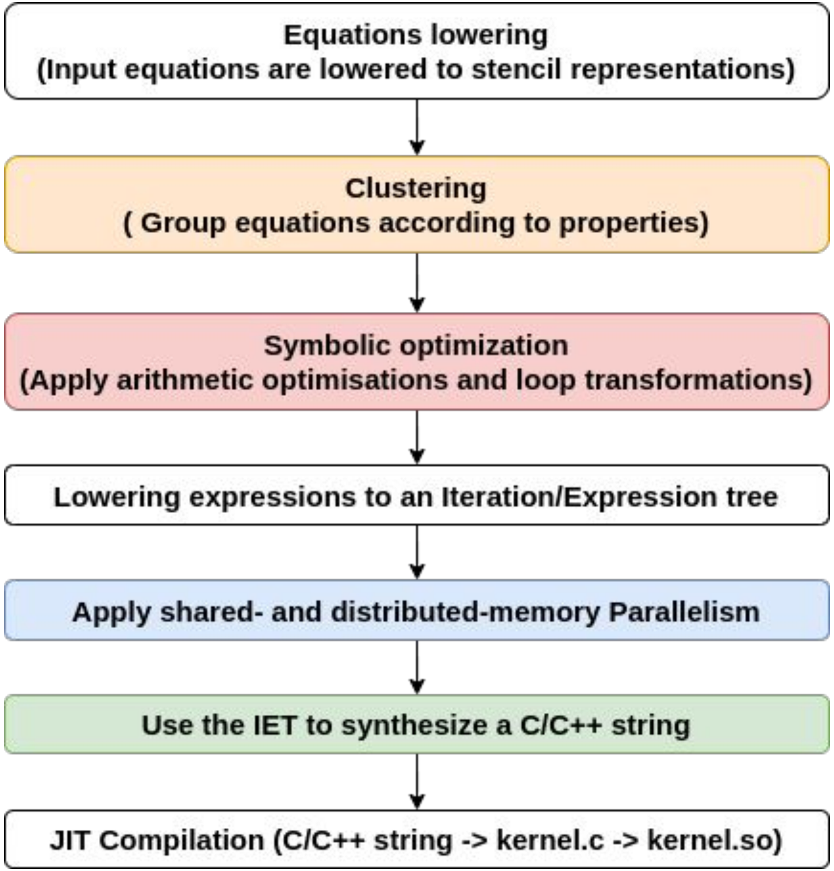
$$u(x,0) = 0$$

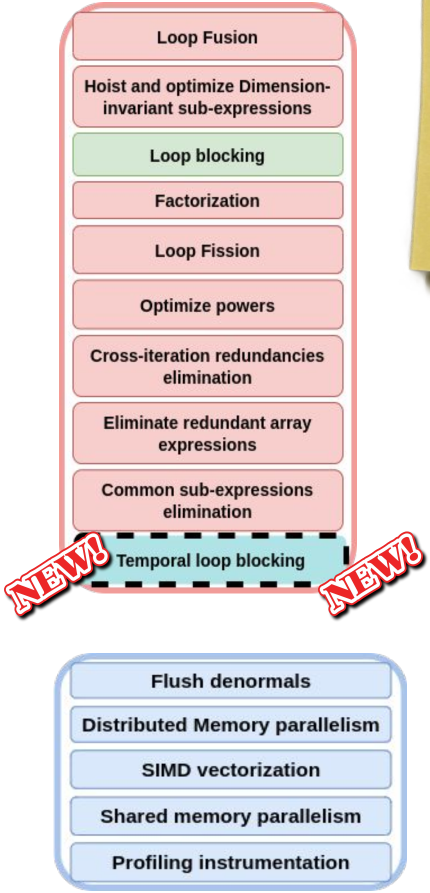




```python
from devito import Eq, Grid, TimeFunction, Operator, solve

# Define a structured grid
nx, ny = 10, 10
grid = Grid(shape=(10, 10))

# Define a field on the structured grid
u = TimeFunction(name='u', grid=grid, space_order=2)

# Define a forward time-stepping symbolic equation
eqn = Eq(u.dt, u.laplace)
eqns = [Eq(u.forward, solve(eqn, u.forward))]

# Define boundary conditions
x, y = grid.dimensions
t = grid.stepping_dim

bc_left = Eq(u[t + 1, 0, y], 0.)
bc_right = Eq(u[t + 1, nx-1, y], 0.)
bc_top = Eq(u[t + 1, x, ny-1], 0.)
bc_bottom = Eq(u[t + 1, x, 0], 0.)

eqns += [bc_left, bc_bottom, bc_right, bc_top]
op = Operator(eqns)

# Compute for 3 timesteps
op.apply(time_M=3, dt=0.1)
```

Devito's compiler optimisations overview





+ advanced combinations of them!
+ heuristics to tune them more!

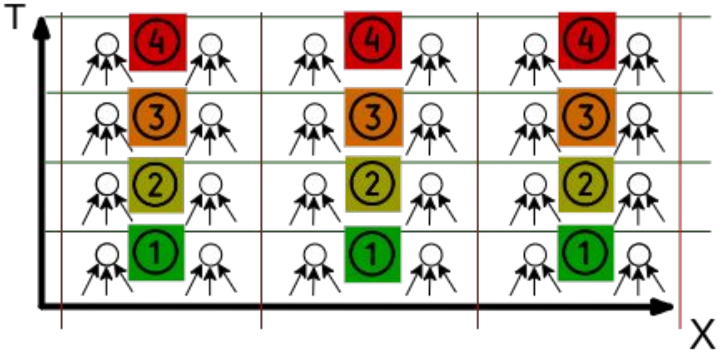
Write once,

Run everywhere!

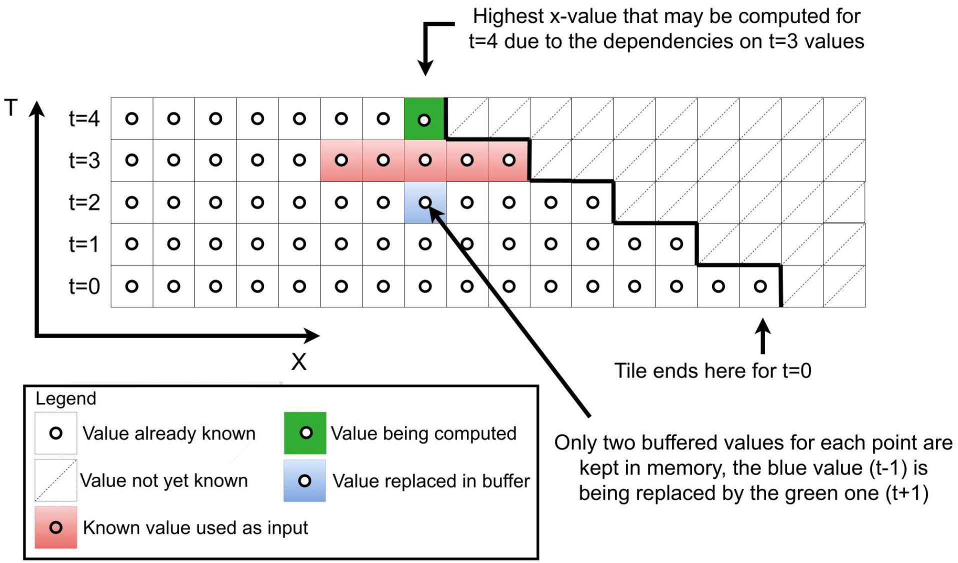
- Serial C/CPP code
- OpenMP parallel code
- MPI (+ OpenMP)
- OpenMP 5 GPU offloading via Clang
- OpenACC GPU offloading

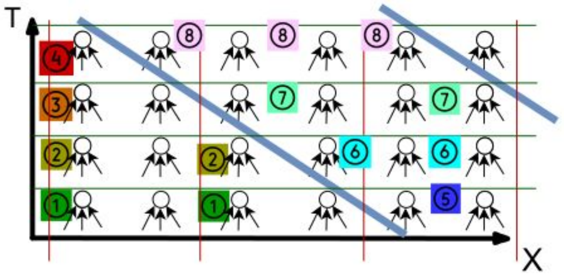
Standard loop blocking (enhancing spatial locality only!)

```
 for (int time = time_m, t0 = (time)%(2), t1 = (time + 1)%(2); time <= time_M; time += 1, t0 =
(time)%(2), t1 = (time + 1)%(2))
  {
    for (int x0_blk0 = x_m; x0_blk0 <= x_M; x0_blk0 += x0_blk0_size)
    {
      for (int y0_blk0 = y_m; y0_blk0 <= y_M; y0_blk0 += y0_blk0_size)
      {
        for (int x = x0_blk0; x <= x0_blk0 + x0_blk0_size - 1; x += 1)
        {
          for (int y = y0_blk0; y <= y0_blk0 + y0_blk0_size - 1; y += 1)
          {
            for (int z = z_m; z <= z_M; z += 1)
            {
              float r4 = -2.0F*u[t0][x + 2][y + 2][z + 2];
              u[t1][x + 2][y + 2][z + 2] = dt*(r0*u[t0][x + 2][y + 2][z + 2] + a*(r1*r4 + r1*u[t0][x + 1][y + 2][z
+ 2] + r1*u[t0][x + 3][y + 2][z + 2] + r2*r4 + r2*u[t0][x + 2][y + 1][z + 2] + r2*u[t0][x + 2][y + 3][z + 2]
+ r3*r4 + r3*u[t0][x + 2][y + 2][z + 1] + r3*u[t0][x + 2][y + 2][z + 3]) + 1.0e-1F);
            }
```

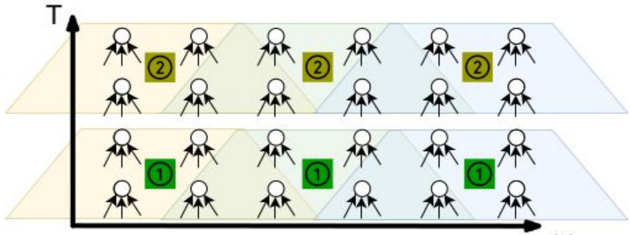


Wavefront temporal blocking

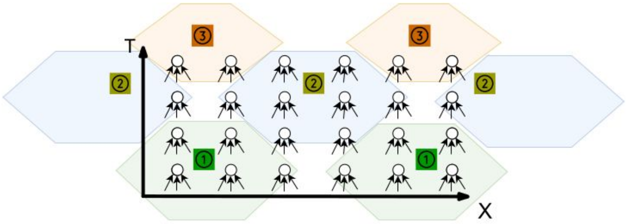




Other temporal blocking variants

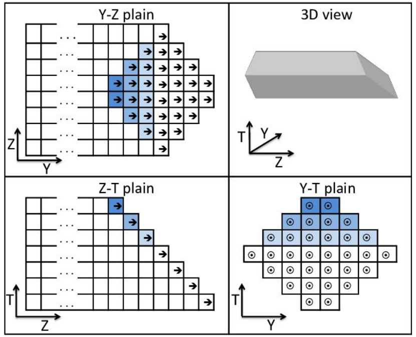


Overlapped temporal blocking

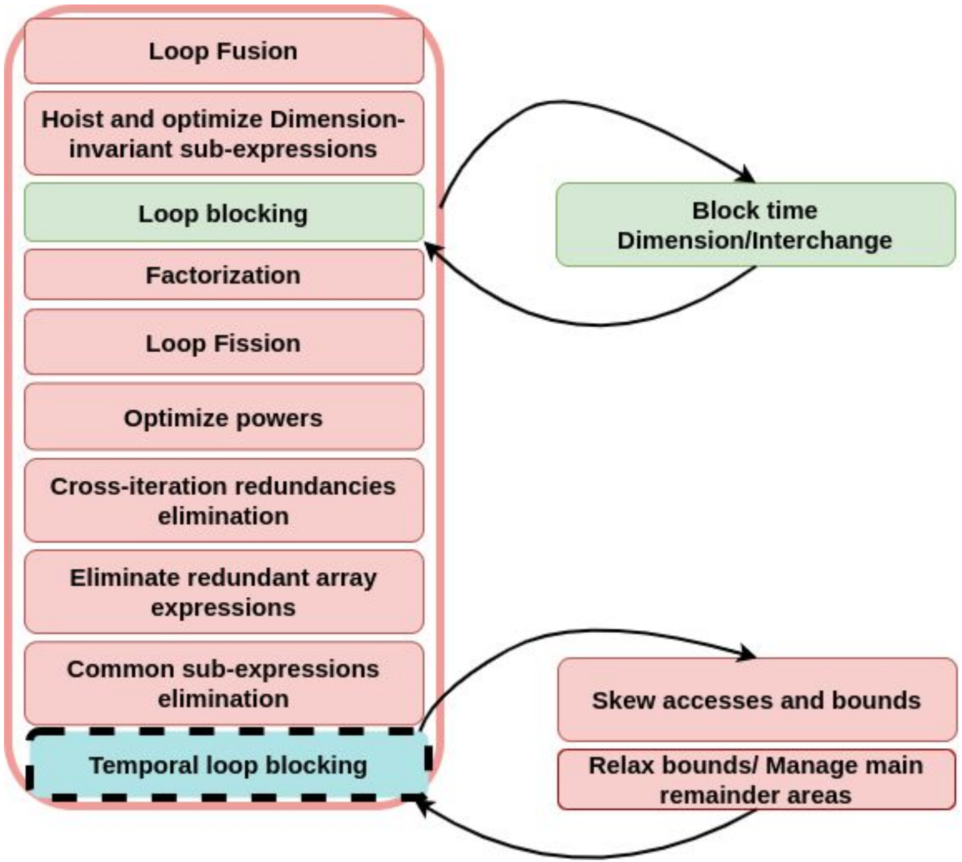


Hexagonal/Diamond temporal blocking

WDT [Malas et.al.]



Synthesizing Temporal blocking in the Devito optimisation pipeline

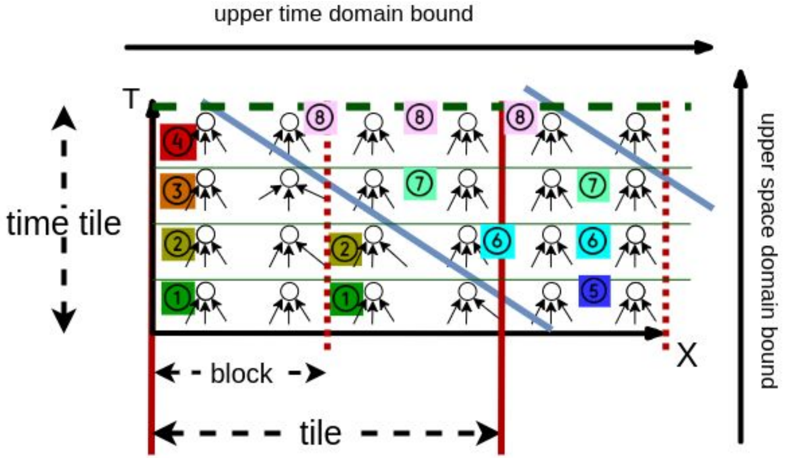


1. Tweak blocking pass to **produce an additional time loop** + space loops, **sort** them accordingly

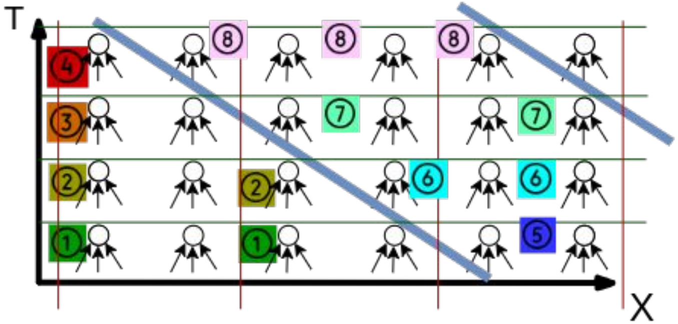
2. **Skew** time accesses and loop bounds

3. Take care of **main/remainder** areas, **time-space diagonals, domain bounds**

✅ Works in tandem with all other Devito opts!



Temporal loop blocking (Wavefront variant)



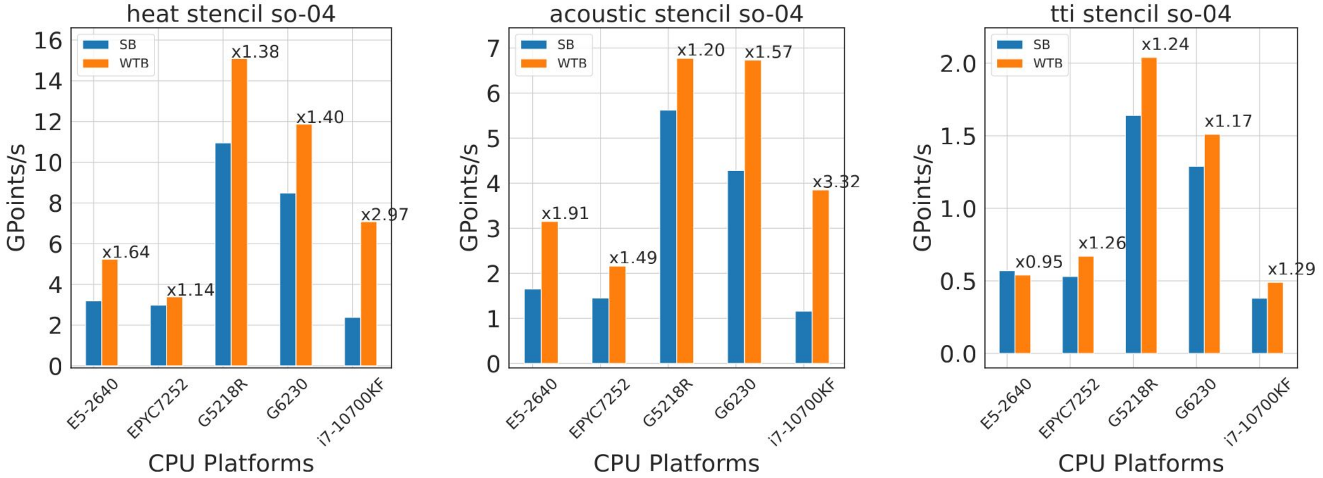
1

```
for (int time0_blk0 = time_m; time0_blk0 <= time_M; time0_blk0 += time0_blk0_size)
 {
   for (int x0_blk0 = x_m; x0_blk0 <= time_M - time_m + x_M; x0_blk0 += x0_blk0_size)
   {
     for (int y0_blk0 = y_m; y0_blk0 <= time_M - time_m + y_M; y0_blk0 += y0_blk0_size)
     {
       for (int time = time0_blk0, t0 = (time)%(2), t1 = (time + 1)%(2); time <= MIN(time0_blk0 + time0_blk0_size - 1, time_M); time += 1, t0 =
(time)%(2), t1 = (time + 1)%(2))
       {
         for (int x0_blk1 = MAX(x0_blk0, time + x_m); x0_blk1 <= MIN(x0_blk0 + x0_blk0_size - 1, time + x_M); x0_blk1 += x0_blk1_size)
         {
           for (int y0_blk1 = MAX(y0_blk0, time + y_m); y0_blk1 <= MIN(y0_blk0 + y0_blk0_size - 1, time + y_M); y0_blk1 += y0_blk1_size)
           {
             for (int x = x0_blk1; x <= MIN(MIN(x0_blk0 + x0_blk0_size - 1, time + x_M), x0_blk1 + x0_blk1_size - 1); x += 1)
             {
               for (int y = y0_blk1; y <= MIN(MIN(y0_blk0 + y0_blk0_size - 1, time + y_M), y0_blk1 + y0_blk1_size - 1); y += 1)
               {
                 for (int z = z_m; z <= z_M; z += 1)
                    {
                     float r4 = -2.0F*u[t0][-time + x + 2][-time + y + 2][z + 2];
                  u[t1][-time + x + 2][-time + y + 2][z + 2] = dt*(r0*u[t0][-time + x + 2][-time + y + 2][z + 2] + a*(r1*r4 + r1*u[t0][-time + x + 1][-time + y +
2][z + 2] + r1*u[t0][-time + x + 3][-time + y + 2][z + 2] + r2*r4 + r2*u[t0][-time + x + 2][-time + y + 1][z + 2] + r2*u[t0][-time + x + 2][-time + y + 3][z
+ 2] + r3*r4 + r3*u[t0][-time + x + 2][-time + y + 2][z + 1] + r3*u[t0][-time + x + 2][-time + y + 2][z + 3]) + 1.0e-1F);}
```

3

2

Experimental evaluation, low discretization orders



CPU characterstics					
	i7-10700KF	Gold 5218R	Gold 6230	EPYC7742	E5-2640
CPU(s)	16	80	40	256	32
Thread(s) per core:	2	2	1	2	2
Core(s) per socket:	8	20	20	64	8
Socket(s):	1	2	2	2	2
NUMA node(s):	1	2	2	8	2
L1d cache:	256KiB	1.3MiB	32KiB	32KiB	512KiB
L1i cache:	256KiB	1.3MiB	32KiB	32KiB	512KiB
L2 cache:	2MiB	40 MiB	1MiB	512KiB	4MiB
L3 cache:	16MiB	55 MiB	27.5MiB	16MiB	40MiB

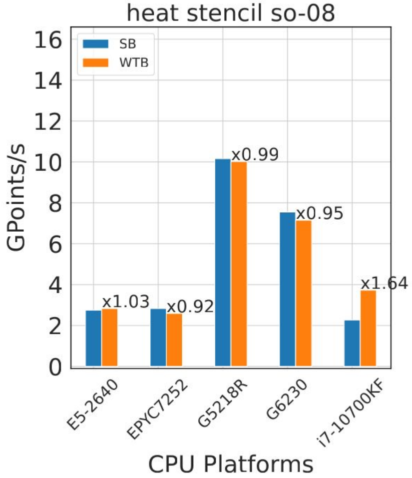
Table 4.1.: Characateristics of CPU platforms used for benchmarking

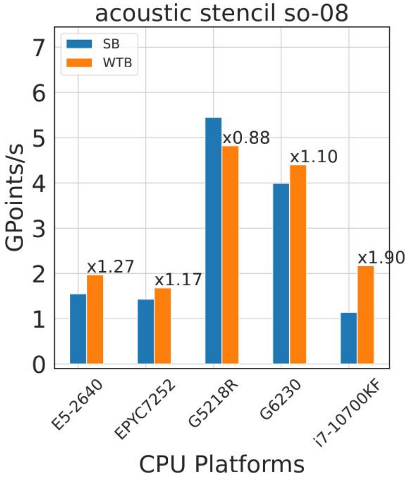
shape 1024 1024 1024, timesteps 512

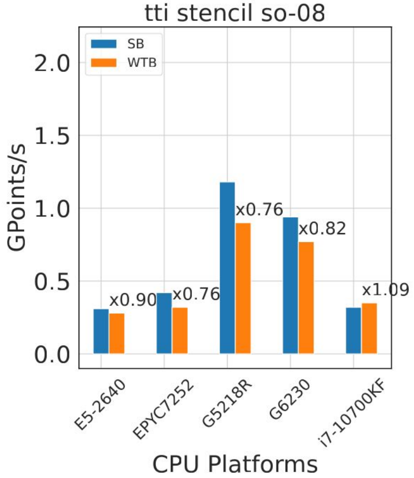
- Kernels are flop-optimized through Devito.
- Gpts/s aka Gcells/s: time to solution metric in stencil computations
- (!) High Gflops/s do not guarantee a faster solution.
- OMP thread pinning, SIMD
- Aggressive auto-tuning

Rooflines available

Experimental evaluation, higher discretization orders







CPU charactерstics					
	i7-10700KF	Gold 5218R	Gold 6230	EPYC7742	E5-2640
CPU(s)	16	80	40	256	32
Thread(s) per core:	2	2	1	2	2
Core(s) per socket:	8	20	20	64	8
Socket(s):	1	2	2	2	2
NUMA node(s):	1	2	2	8	2
L1d cache:	256KiB	1.3MiB	32KiB	32KiB	512KiB
L1i cache:	256KiB	1.3MiB	32KiB	32KiB	512KiB
L2 cache:	2MiB	40 MiB	1MiB	512KiB	4MiB
L3 cache:	16MiB	55 MiB	27.5MiB	16MiB	40MiB

Table 4.1.: Characateristics of CPU platforms used for benchmarking

shape 1024 1024 1024, timesteps 512

- Kernels are flop-optimized through Devito.
- Gpts/s aka Gcells/s: time to solution metric in stencil computations
- (!) High Gflops/s do not guarantee a faster solution.
- OMP thread pinning, SIMD
- Aggressive auto-tuning

Rooflines available

Conclusions

- We presented **Devito, a DSL and compiler framework** for explicit finite difference schemes for **solving PDEs** using the **FD method for structured grids (**but not limited to them!)
- The Devito Compiler supports a great variety of optimizations for stencil kernels, **we aim to add another one, to enhance temporal data reuse**
- Promising performance gains of ranging from 3x on low order (4) to 1.6x and 1.9x on higher order (8) problems

Current WIP

- Full integration to DSL (currently in a branch/fork)
- User will get out-of the box time tiled code for all PDEs!

Future plans

- Challenges with interpolations
- Automate more TB schemes
- Add MPI-aware scheme
- Extend TB to GPUs
- Performance for higher-order stencils

- Website
- Slack
- Code

DEVITOPROJECT

Join us, use Devito, work with us!

Imperial College London

EPSRC Engineering and Physical Sciences Research Council

HiPEDS

DEVITO