Accelerating real-world stencil computations using temporal blocking: handling sparse sources and receivers

George Bisbas [1] Fabio Luporini [1](advisor) Mathias Louboutin [2](advisor) Gerard Gorman [1] (advisor) Paul H.J. Kelly [1](advisor)

[1]Imperial College London, UK [2]Georgia Institute of Technology, Atlanta, USA

Background

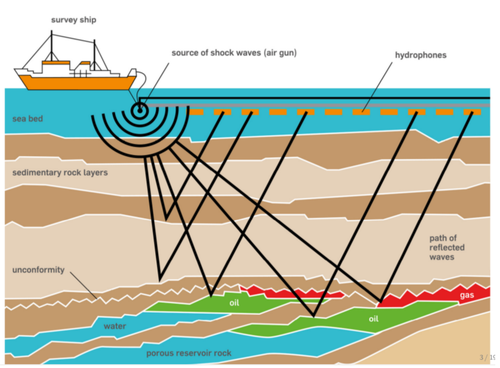
- Temporal blocking, also known as time-tiling is a well-known technique for accelerating stencil based computations by enhancing data locality.
- Solving partial differential equations with the finite difference method yields stencil codes that can be very complex for real world problems.
- Devito [1] is a tool for generating stencil codes from a high-level symbolic abstraction aiming at real-world applications targetting mostly seismic imaging.
- Seismic inversion related operators contain imperfect loops and include a mix of affine/non-affine operators due to sparse ones such as source injection and receiver interpolation as well as complex boundary conditions.

Motivation: temporal blocking for real-world applications

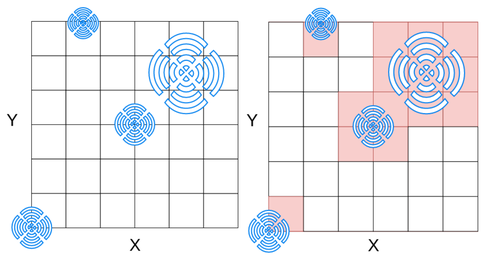
- Temporal blocking has proven efficient for imperfect loop nests while extensions to polyhedral tools claim to be able to handle non-affine loop nest transformations.
- Integration of such tools is limited to benchmarking purposes and not used in production code.
- Using time-tiling in production-level seismic imaging operators is challenging. One reason is the presence of non-trivial operators which elude finite differences (i.e., they are not classic stencils), including interpolation and non-trivial boundary conditions. Further, time-tiling should support distributed-memory parallelism. Optimizations to extend our work to high-order finite-difference will be investigated in future work.

Contributions:

- An inspector/executor scheme with low inspection cost in computation time and memory footprint for precomputing the effect of sparse operators (sources and receivers).
- Performance evaluation showing profitable results for low order stencils and competitive for higher order (that is under investigation).

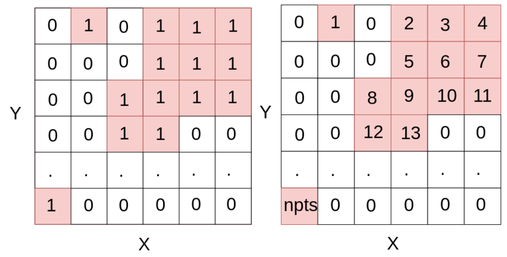


A seismic imaging survey where sources injection and receiver interpolation take place. Source: Open Learn



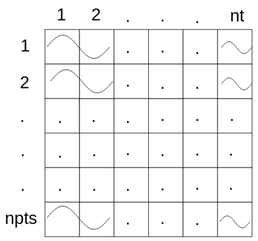
(a) Step 0: Sources are injected on a field.

(b) Step 1: Identify area affected from sources.

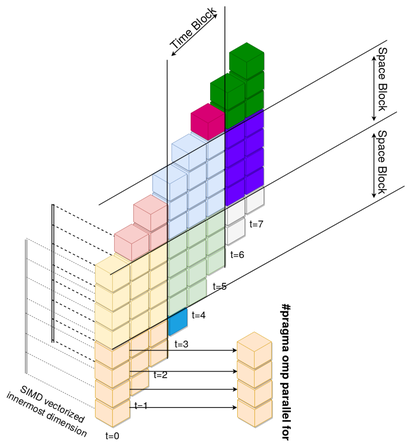


(a) Step 2: Binary mask with 1s on affected points.

(b) Step 3: A unique ID for each affected unique point.



(a) Step 4: Store efficiently all the precomputed source injection.



Temporal blocking parallelism levels.

The algorithm:

Our inspector/executor scheme for this loop nest transformation consists of the following 4 steps:

1. Inspection: Iterate over the sources (Fig:1a) for all timesteps to identify the unique points that are affected (Fig:1b) and then allocate space for a 2d array of size *unique_pts_affected* $\times$ *timesteps* .
2. Inspection: Two copies of our grid act as a binary mask of affected points (Fig:2a) and as an id mask (Fig:2b) for our points.
3. Execution: Iterate again over the sources, applying their effect (Fig:3a) to our allocated structure described in step 1
4. Execution: Transform the perfect loop nest for temporal blocking and adding the effect of the corresponding source using the binary mask created in step 1

Inspector/ Executor Code:

```
# Inspector
for (ti = 0; ti < timesteps; ti++)
  for (p_src = 0; p_src < p_src_M; p_src++)
    xx = src_coords[p_src][0]
    yy = src_coords[p_src][1]
    zz = src_coords[p_src][2]
    if (source_id[xx][yy][zz] == 0)
      source_id[xx][yy][zz] = id++
    source_mask[xx][yy][zz] = 1

double **save_src;
malloc2d(&save_src, id, timesteps);

# Executor
for (ti = 0; ti < timesteps; ti++)
  for (p_src = 0; p_src < p_src_M; p_src++)
    xx = src_coords[p_src][0]
    yy = src_coords[p_src][1]
    zz = src_coords[p_src][2]
    this_id = source_mask[xx][yy][zz]
    save_src[this_id][ti] += src[ti][p_src]

for (x_blk = x_m; x_blk < ; += x_blk_size)
  for (y_blk = ; y_blk < , ; += y_blk_size)
    for (t = ; t < ; t += sf)
      for (z_blk = ; z_blk < ; z_blk += z_blk_size)
        #pragma omp parallel for collapse(2)
        for (xi = ; xi <  ; xi++)
          for (yi = ; yi < ; yi++)
            #pragma omp simd
            for (zi = ; zi < ; zi++)
              grid[t1][xi - t][yi - t][zi] = PDE_stencil
            for (zi = ; zi < ; zi++)
              if(source_mask[xi-t][yi-t][zi])
              {grid[t1][xi-t][yi-t][zi]
              +=source_mask[xi-t][yi-t][zi]
              *save_src[(source_id[xi-t][yi-t][zi])][tw]}
```

Future work: automate temporal blocking in Devito

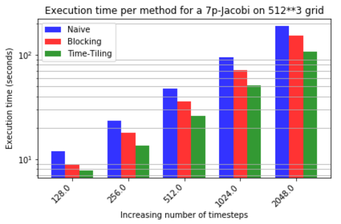
Devito allows to:

- Generate stencil code on-the-fly, allowing various types of. performance optimization to be performed during code generation.
- Use MPI/OpenMP parallelism and SIMD vectorization.
- Sophisticated loop transformations (e.g., blocking and auto-tuning).
- Domain-specific symbolic optimizations.

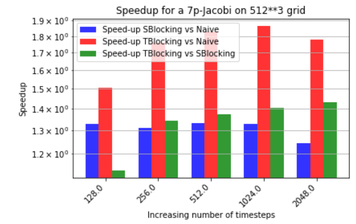
Our work aims to enhance this framework with temporal blocking optimizations that will be optimally tuned for increasing the performance of generated code, targetting real world applications.

Results

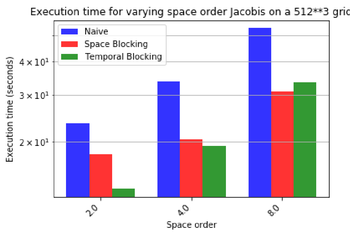
Experiments were executed for 7p, 13p and 25p Jacobis (Space orders 2, 4 and 8 respectively) on a 512**3 grid with 5 sources injecting for the whole time-domain of the experiment and block sizes of 32 x 32 x 256. Temporal blocking speedups over space blocking tend to decrease as the space order increases. The resolution of this issue is under investigation.



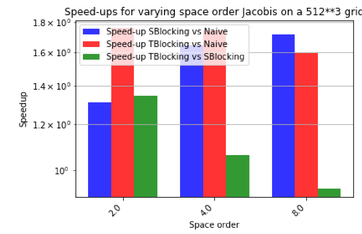
Execution time for different number of timesteps.



Comparing speed-ups for every method.



Execution time for a 256-step experiment for varying space order.



Comparing speed-ups for a 256-step experiment for varying space order.

References

- Luporini, Fabio, et al. "Architecture and performance of Devito, a system for automated stencil computation." arXiv preprint arXiv:1807.03032 (2018).
- Sim, Nicholas. "Optimising finite-difference methods for PDEs through parameterised time-tiling in Devito." arXiv preprint arXiv:1806.08299 (2018).
- Louboutin, Mathias et al. "Devito (v3.1.0): an embedded domain-specific language for finite differences and geophysical exploration." (2018).

Links:

www.devitoproject.org
https://github.com/opesci/devito
https://twitter.com/opesciproject

Imperial College London

EPSRC Engineering and Physical Sciences Research Council