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

George Bisbas
g.bisbas18@imperial.ac.uk
Imperial College
London, United Kingdom

Fabio Luporini (advisor)
f.luporini12@imperial.ac.uk
Imperial College
London, United Kingdom

Mathias Louboutin (advisor)
mloubout3@gatech.edu
Georgia Institute of Technology
Atlanta, Georgia, US

Gerard Gorman (advisor)
ggorman@imperial.ac.uk
Imperial College
London, United Kingdom

Paul H.J. Kelly (advisor)
paulhjkelly@imperial.ac.uk
Imperial College
London, United Kingdom

ABSTRACT

This paper concerns performance optimisation in finite-difference solvers found in seismic imaging. Tiling for locality across multiple time iterations is known to be profitable. We tackle a complicating factor that is arising in seismic inversion problems (as well as other similar contexts): the addition of waves injected from sources distributed sparsely over the 2D/3D domain, and the need for receivers that interpolate data measurements at a set of points, again distributed sparsely across the domain. The presence of code to handle sources and receivers prevents straightforward application of temporal blocking. In this work we show how to overcome this limitation.

In the current work, we introduce an algorithm for a loop nest transformation policy applied to wave propagation modelling with the finite difference method in order to improve data locality and optimize our cache memory use. Our algorithm uses an inspector/executor scheme capable of inspecting measurement operations at sparse locations and then compute the field updates. Our contribution is optimizing data locality through tiling and in particular using temporal blocking also known as time-tiling. A lot of work has been done for tiling loop nests with affine transformations, however our work is trying to apply temporal blocking on loop structures that have a non-affine nature and consist part of a large code production framework. We present the methodology and the algorithm followed by the improvements achieved in execution time and the results for FD kernels over a wide range of parameters. The ultimate goal of the work presented in this work is to automate this scheme for stencil codes generated by Devito.

KEYWORDS

temporal blocking, stencil computations, code-generation, PDEs, sparse operators

Conference'17, November 2019, Colorado, Denver, CO, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
George Bisbas, Fabio Luporini (advisor), Mathias Louboutin (advisor), Gerard Gorman (advisor), and Paul H.J. Kelly (advisor). 2019. Accelerating real-world stencil computations using temporal blocking: handling sparse sources and receivers. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Accelerating stencil computations has long been studied on a great variety of platforms and a lot of work has been done on enhancing data locality in order to optimize our computations through use of cache memory. 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 solve complex problems described by PDEs. Related work is partly solving the issues encountered in this research area however there is no framework that currently supports the modelling of real-world problems, a high-level syntax of PDE's, automatic parallel code generation (SIMD/OpenMP/MPI) and loop nest optimizations all together. We introduce an algorithm to apply temporal blocking to phenomena with source injection and is designed to be delivered, and automated as part of the Devito [4] framework.

2 MOTIVATION

Currently, there are quite a few frameworks trying solve and optimize stencil computations, in addition to Devito which is the working project of our research team, there are other frameworks for solving PDEs through code-generation like Firedrake [**?**]. Larger projects include Halide [6].

Considering automatic code-generation frameworks that are already bridged with Devito, it is worth to mention OPS [7] for GPU code generation and YASK [9].

While polyhedral tools manage to deal with most of the usual loop nest cases, the most important tools for polyhedral compilation like PLUTO [3], [2] and CLooG [1], are not able to deal with the non-affine nature of our loop nest structure. Code-generation with affine transformations has been studied well, however these tools are not in position to help our code-generation scheme for a number of reasons such as limited applicability to non-affine functions and technical difficulties in integrating with the existing compiler.

Spatial blocking schemes have long been explored and their performance gain is highly explored. For this reason temporal blocking algorithms have been developed to optimize data locality in the time dimension [5]. Temporal blocking has been shown to work for imperfect loop nests while some extensions to polyhedral tools claim to be able to handle non-affine loop nest transformations [8].

We focus on seismic imaging operators that differ from the typically encountered stencil codes. 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. Furthermore, time-tiling should support distributed-memory parallelism. It is also still unclear what optimizations will be necessary in the case of high-order seismic operators, however our work focuses on an algorithm that can be used for non-trivial stencil cases and is using skewed temporal blocking for its purposes. More temporal-blocking schemes are to be evaluated as future work helping in the direction of avoiding issues that are specific to different kinds of time-tiling (i.e. the pipelined start-up of skewed time-tiling).

3 METHODOLOGY AND IMPLEMENTATION

Our methodology is built upon an inspector/executor scheme trying to take advantage of all the data/parameters that are available at compile time and aiming to change the order of the computations in our problem. The parameters that are available at inspection time are the number of sources, the wavelet injected by each source, and the radius of the source impact. This data along with the size of grid and the number of simulation timesteps allow us to precompute the effect of the sources on an empty grid. Initially we perform a first pass on the sources for the whole time domain in order to identify all the unique points that may ever be affected from the source injection. A point is considered to be affected if at least one source injects to it for at least one timestep. The additional data structures that are going to be used for this computation are:

(1) A n-dimensional array, where usually $n = 3$ called $source\_id$. This array will store a unique id for every point that is affected from at least one source and at least for one timestep for the whole time-iteration space.
(2) A n-dimensional array, where usually $n = 3$ called $source\_mask$. This array will store 1s for every point that is affected from at least one source and at least for one timestep for the whole time-iteration space. This array will work as a binary mask in order to add (1) or not (0) source impact to a specific point.

The space overhead induced for the inspector scheme is introducing a space overhead compared but this is small compared to the space allocated for the execution of the seismic imaging operators.

In $step$ 2, after the identification of the affected points is over in $step1$, we allocate space for the precomputation of the source effect in every unique point. Let the allocated structure be named pre_comp_src. Array pre_comp_src is of size $id \times timesteps$.

The next step, $step$ 3, is the precomputation of the effect at every point. At the source effect pre-computation, array pre_comp_src is populated with all the required information concerning the source injection.

The main advantage of this pipeline is that now the loop used for computing the source injection is hoisted in a different loop of time compared to the state it was before. This leads to a loop nest structure where two different time loops are under an outer time loop. As a consequence, our main field computation loop is now in a format where we can easily apply time-tiling via skewing or other variations of temporal blocking.

4 EVALUATION AND RESULTS

The results presented in the poster consist of handwritten code that simulates source injection in a 512^3 grid with 13p, 25p, 37p and 49p Jacobi stencils. Experiments benefit from OpenMP parallelism at the inner x and y dimensions and SIMD vectorization in the innermost z dimension. Experiments were executed on 4 cores on a i7-7700K at 4.20GHz with thread pinning enabled.

5 FUTURE WORK

The ultimate goal of this research work is to automate the application of temporal blocking to finite difference solvers for real-world problems at extreme scales through the Devito [4] Compiler. The scheme presented in this work is going to be part of Devito delivering new optimizations. The results and the examples presented in this poster work are mainly motivated from the seismic imaging domain however the target applications are not limited to this scope.

REFERENCES

[1] C Bastoul. 2004. Code generation in the polyhedral model is easier than you think. *Proceedings 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04)* (2004), 7–16. https://doi.org/10.1109/PACT.2004.10018

[2] Uday Bondhugula. 2013. Compiling affine loop nests for distributed-memory parallel architectures. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '13* (2013), 1–12. https://doi.org/10.1145/2503210.2503289

[3] Uday Bondhugula, Albert Hartono, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. 2008. A practical automatic polyhedral parallelizer and locality optimizer. In *ACM SIGPLAN Notices*, Vol. 43. ACM, 101–113.

[4] Fabio Luporini, Michael Lange, Mathias Louboutin, Navjot Kukreja, Jan Hückelheim, Charles Yount, Philipp Witte, Paul H. J. Kelly, Gerard J. Gorman, and Felix J. Herrmann. 2018. Architecture and performance of Devito, a system for automated stencil computation. *CoRR* (2018), 1–27. arXiv:arXiv:1807.03032v1

[5] Takayuki Muranushi and Junichiro Makino. 2015. Optimal Temporal Blocking for Stencil Computation. In *ICCS*.

[6] Jonathan Ragan-Kelley, Andrew Adams, Dillon Sharlet, Connelly Barnes, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. 2017. Halide: decoupling algorithms from schedules for high-performance image processing. *Commun. ACM* 61, 1 (2017), 106–115.

[7] István Z. Reguly, Gihan R. Mudalige, Michael B. Giles, Dan Curran, and Simon McIntosh-Smith. 2014. The OPS domain specific abstraction for multi-block structured grid computations. *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* (2014), 58–67. https://doi.org/10.1109/WOLFHPC.2014.7

[8] Anand Venkat, Mary Hall, and Michelle Strout. 2015. Loop and data transformations for sparse matrix code. *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2015* (2015), 521–532. https://doi.org/10.1145/2737924.2738003

[9] Charles Yount. 2015. Vector folding: Improving stencil performance via multi-dimensional SIMD-vector representation. *Proceedings - 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security and 2015 IEEE 12th International Conference on Embedded Software and Systems, HPCC-CSS-ICESS 2015* (2015), 865–870. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.27