DeVito: Fast Finite Difference Computation

Marcos de Aguiar
SENAI CIMATEC
Salvador, Brazil
marcos.aguiar@fieb.org.br

Gerard Gorman
Imperial College
London
g.gorman@imperial.ac.uk

Felix Herrmann
University of
British Columbia
fherrmann@eos.ubc.ca

Navjot Kukreja
SENAI CIMATEC
Salvador, Brazil
navjotk@gmail.com

Michael Lange
Imperial College
London
michael.lange@imperial.ac.uk

Mathias Louboutin
University of
British Columbia
mloubout@eos.ubc.ca

Felippe Zacarias
SENAI CIMATEC
Salvador, Brazil
felippe.vieira@fieb.org.br

ABSTRACT

Entering the exascale era, High Performance Computing (HPC) architectures are rapidly changing and diversifying to continue delivering performance increases. These changes offer opportunities while also demanding disruptive software changes to harness the full hardware potential. An important beneficiary of these developments is the area of Seismic Imaging. Subsurface imaging techniques reduce the need for expensive exploratory drilling and enable reservoir monitoring throughout their life cycle to help maximize the value of discovered resources. However, seismic imaging techniques like 3D Full Waveform Inversion (FWI) are computationally demanding and have become feasible only because of recent advancements in HPC. The most computationally intensive part of FWI typically involves modelling the wave propagation numerically using Finite Difference (FD) formulations. Writing optimised code for FWI and other similar applications involves multiple man-years of effort that needs to be repeated every time a new development needs to be factored in – for every target platform. The question is how to achieve an acceptable degree of performance portability across diverse, rapidly evolving architectures, in spite of the sharp trade-off between easy-to-maintain, portable software written in high-level languages, and highly optimized, parallel codes for target architectures.

There have been multiple explorations into this problem of achieving performance portability across different architectures while maintaining a common code-base. This is usually solved by building code optimisation tools. An important question to be addressed while designing these tools is the abstraction level at which the application developer interacts with the tool. In the case of a code-to-code translation tool, for example, a large majority of the implementation details that could be used for optimisation are already frozen as implicit assumptions in code. Therefore it becomes imperative to have the application developer write as less code as possible - so that the rest of the code can be customised to a particular hardware platform.

DeVito is a new tool for performing optimised FD computation from high-level symbolic problem definitions. Given a differential equation in symbolic form, DeVito performs automated code generation and Just-In-time (JIT) compilation based on this symbolic equation. The generated code is highly optimised for the specific computer architecture. By delaying the generation of the actual code until the properties of the data set and the target platform are known, DeVito opens the door to an array of optimisations that aren't available to a code-to-code translation tool. An important role that DeVito plays is to provide a layer of separation between the domain expert and the HPC code tuning expert. While the domain expert can interact with DeVito using symbolic math without caring about the details of the performance optimisations, the HPC code tuning expert can focus on the lower layers of DeVito and concentrate their efforts on code optimisation without getting bogged down by the details of the target application. This separation of concerns enables performance portability, enhanced programmability and productivity, while providing the ability to quickly evaluate new numerical approaches without enormous development costs. DeVito has been designed to be used as part of complex workflows involving data flows across multiple applications over different nodes of a cluster.

1. REFERENCES

[1] L. Borges and P. Thierry. 3d finite differences on multi-core processors. *http://software. intel. com/en-us/articles/3d-finite-differences-on-multi-core-processors)*, 2(3):4–2, 2011.

[2] J. Jeffers and J. Reinders. *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches*. Morgan Kaufmann, 2015.

[3] D. Joyner, O. Čertík, A. Meurer, and B. E. Granger. Open source computer algebra systems: Sympy. *ACM Communications in Computer Algebra*, 45(3/4):225–234, 2012.

[4] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.