High Performance Stencil Code Generation with Lift

Bastian Hagedorn
University of Münster

Larisa Stoltzfus
The University of Edinburgh

Michel Steuwer
University of Glasgow

Sergei Gorlatch
University of Münster

Christophe Dubach
The University of Edinburgh

Introduction Stencils are a class of algorithms which update elements in a multi-dimensional grid based on neighboring values using a fixed pattern. They are used extensively in various domains such as medical imaging, numerical methods or machine-learning. Stencils are part of the original "seven dwarfs" [1] and are considered one of the most relevant classes of high-performance computing applications.

However, efficient programming of stencils for parallel accelerators such as Graphics Processing Units (GPUs) is challenging even for experienced programmers. Hand-optimized high-performance stencil code is usually written using low-level programming languages like OpenCL or CUDA. Achieving high-performance requires expert knowledge to manage every hardware detail. For instance, special care is required on how parallelism is mapped to GPUs or how data locality is exploited with local memory to maximize performance.

Domain Specific Languages (DSLs) and high-level library approaches have been successful at simplifying HPC application development. These approaches are based on algorithmic skeletons [3] which are recurring patterns of parallel programming. While these raise the abstraction level, they rely on hard-coded, not performance portable implementations. Alternative approaches are based on code generation, which places a huge burden on the implementers who have to reinvent the wheel for each new application domain.

Extending Lift for Stencil Computations Lift [5] is a novel code generation approach based on a high-level, data-parallel intermediate language whose central tenet is performance portability. It is designed as a target for DSLs and library authors, and exploits functional principles to produce high-performance GPU code. Applications are expressed using a small set of functional primitives and optimizations are all encoded as formal, semantics-preserving rewrite rules. These rules define an optimization space which is automatically searched for high-performance code [7]. Lift liberates DSL implementers from the tedious process of re-writing and tuning their code for each new domain or hardware.

Instead of expressing stencil computations using a single high-level *stencil* primitive, as often seen in other high-level approaches, e.g. [2, 6], in Lift we aim for composability and instead express stencil computations using smaller fundamental building blocks. Figure 1 shows how stencil computations are expressed in Lift. Stencil computations are decomposed into three fundamental parts, each represented by a primitive: (1) *boundary handling* is performed using *pad* which specifies how to handle neighboring values for elements at the borders of the input grid; (2) for every element of the input, a *neighborhood* is accessed specified by the shape of the stencil (*slide*); (3) *for each* neighborhood, a single output element is computed (reusing *map*).

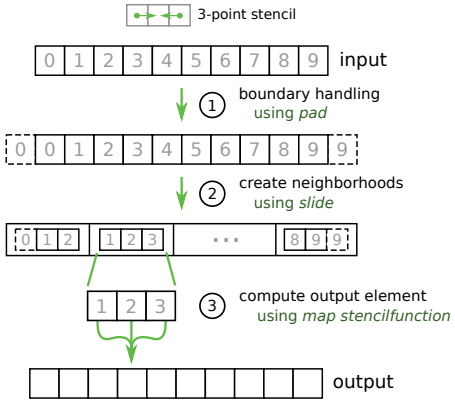

Figure 1. Expressing a stencil in Lift using *pad* for boundary handling, *slide* for creating the neighborhood and *map* to compute the output elements. These three logical steps are compiled into a single efficient OpenCL kernel by Lift.

By composing generic, intuitive 1D primitives, complex multi-dimensional stencils are expressible, demonstrating the extensibility of the Lift approach to new application domains. In Lift, optimizations are encoded as rewrite rules. Overlapped tiling, a stencil-specific optimization, is formalized and added as a new rewrite rule targeting the introduced new primitives. Besides adding two new primitives (*slide* and *pad*) and the overlapped tiling rule, we reuse Lift's existing set of rewrite rules and search space exploration.

Results In our experimental evaluation, we show that Lift automatically generates high-performance stencil code for AMD, NVIDIA and ARM GPUs. The results show that our approach is highly competitive with hand-written implementations and outperforms the state-of-the-art PPCG polyhedral compiler. For some benchmarks, the hand-written implementations show large performance differences across architectures. However, Lift is able to achieve good performance on all architectures, providing performance portability.

A full paper of this work has been presented at CGO'18 [4].

References

[1] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, et al. 2006. *The Landscape Of Parallel Computing Research: A View From Berkeley.* Technical Report. UCB/EECS-2006-183, EECS Department, University of California, Berkeley.

[2] Matthias Christen, Olaf Schenk, and Helmar Burkhart. 2011. PATUS: A Code Generation And Autotuning Framework For Parallel Iterative Stencil Computations on Modern Microarchitectures. In *IPDPS*. IEEE, 676–687.

[3] Murray I Cole. 1988. *Algorithmic Skeletons: A Structured Approach To The Management Of Parallel Computation.* Ph.D. Dissertation. University of Edinburgh.

[4] Bastian Hagedorn, Larisa Stoltzfus, Michel Steuwer, Sergei Gorlatch, and Christophe Dubach. 2018. High Performance Stencil Code Generation with Lift. In *CGO*. ACM.

[5] Michel Steuwer, Christian Fensch, Sam Lindley, and Christophe Dubach. 2015. Generating Performance Portable Code Using Rewrite Rules: From High-Level Functional Expressions To High-Performance OpenCL Code. In *ICFP*. ACM, 205–217.

[6] Michel Steuwer, Michael Haidl, Stefan Breuer, and Sergei Gorlatch. 2014. High-Level Programming Of Stencil Computations On Multi-GPU Systems Using The SkelCL Library. *Parallel Processing Letters* 24, 03 (2014), 1441005.

[7] Michel Steuwer, Toomas Remmelg, and Christophe Dubach. 2017. Lift: A Functional Data-Parallel IR For High-Performance GPU Code generation. In *CGO*. ACM, 74–85.