SODA-OPT: System-Level Design with MLIR for HLS

Nicolas Bohm Agostini, David Kaeli, Antonino Tumeo

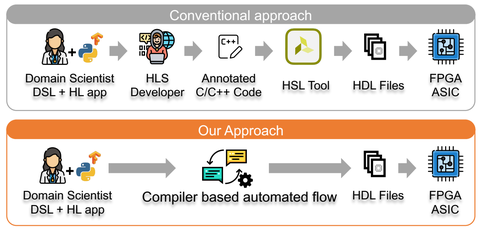
SC21 St. Louis, MO | science & beyond.

Motivation

- General purpose computing units are experiencing **progressively smaller performance gains** due to technology and power limitations
- **Architecture innovations are key** to keep increasing performance of current applications
- Custom **Domain Specific Accelerators** are now the main drivers of performance gains in different domains
- **High-Level Synthesis (HLS)** can be used for rapid generation of custom Domain Specific Accelerators

Challenges

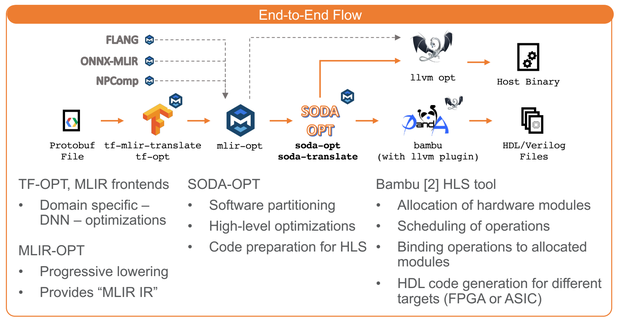
- HLS still **requires** considerable programming effort of knowledgeable **experts** to yield high performance designs
- **Most HLS flows** [3-4] must map into **annotated C/C++** to guide the synthesis
- **High-level (HL) information** of the application's dataflow, task or memory level parallelism is **often lost** when manual translation to C/C++ is performed

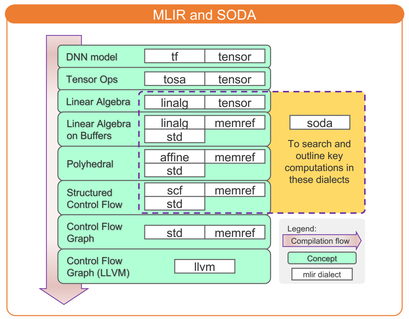


Contributions

- **MLIR [1] based** compiler flow for high-level synthesis
- MLIR **dialect to search and outline** MLIR code at suitable abstractions
- Compiler **passes and pipelines** to enhance HLS of arbitrary regions of a dataflow application
- Puts the domain scientist **in control** of custom accelerator generation with compiler passes

Approach





SODA Operations

- The SODA dialect borrows ideas of the GPU dialect
- Search step marks code regions to be outlined
- Outlining step moves selected regions into a special MLIR module
- `Operations` and Semantics are described below

Before outlining (used in the search step)

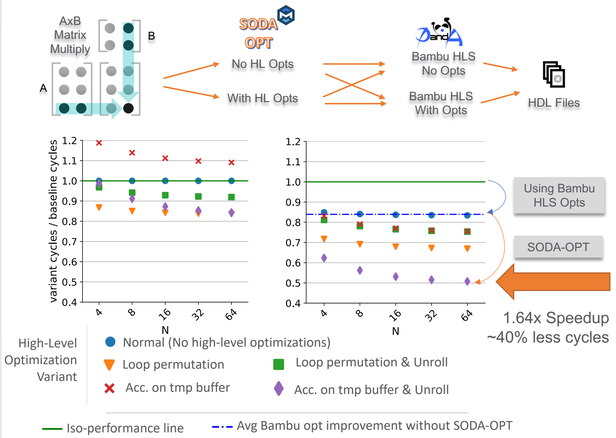
`soda.launch`	Marks code blocks to be outlined and extracted into separate kernel functions
`soda.terminator`	Indicates the end of a code region to be outlined

After outlining

`soda.launch_func`	Calls outlined functions and is replaced by accelerator call API in the host code
`soda.module`	Holds the list of outlined kernels to be later optimized
`soda.func`	Defines outlined kernels and their interface as functions
`soda.return`	Indicates the end of a kernel function

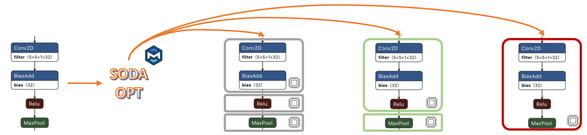
Benefits of High-Level Optimizations

- **High-level optimizations** can **increase performance** of final designs
- High-level optimizations, **paired** with standard HLS optimizations **generate even faster** designs



Outlining Dataflow Applications with SODA-OPT

- Experiment investigates the **impact of outlining** and generating accelerators for different granularities of a DNN model



Layer type	Layer params	Individual Ops no-opt	Individual Ops opt	Fused as in TF kernels no-opt	Fused as in TF kernels opt	Fused in coarser granularity no-opt	Fused in coarser granularity opt	Entire network no-opt	Entire network opt
Conv2D	5x5,6,same	2,423,374	2,353,598	2,462,602	2,388,122	2,526,225	2,443,073	6,806,682	5,965,844
Activation	Relu	39,230	34,526						
AveragePooling2D	2x2, s2x2	88,244	84,338	88,244	84,338				
Conv2D	5x5,16,same	4,917,022	4,835,522	4,917,022	4,835,522	5,175,970	4,853,938		
Activation	Relu	12,912	11,312						
AveragePooling2D	2x2, s2x2	30,092	28,842	30,092	28,842				
Dense	120 units	1,010,522	926,402	1,011,602	927,362	1,011,602	927,362		
Activation	Relu	962	842						
Dense	84 units	213,446	195,722	214,202	196,394	214,202	196,394		
Activation	Relu	674	590						
Dense	10 units	17,841	16,372	17,841	16,372	19,084	16,731		
Activation	Softmax	454	410	454	410				
Total		8,754,773	8,488,476	8,742,059	8,477,362	8,947,083	8,437,498	6,806,682	5,965,844

1.46x Speedup (~30% less cycles)

Conclusions

- We created a tool to **automatically** search, outline, optimize, dispatch, and accelerate kernels for HLS
  - It **eases programmer effort** to target FPGAs or generate code for ASICs
- **High-level optimizations** can further **improved HLS** synthesis performance
  - Our microkernels show 1.6x speedup over baseline designs that only leverage HLS optimizations
- SODA-OPT is flexible and can **outline** kernels from dataflow applications **at different granularities**
  - It enables the possibility of **fusing** multiple dataflow operations into a single accelerator
- It provides new HLS capabilities, essential to enable an "**agile hardware design**" approach

References

[1] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In CGO. 2–14.
[2] Bambu Developers. 2021. Bambu: A Free Framework for the HLS of Complex Applications.
[3] Skalicky, Sam, et al., 2018. Hot & Spicy: Improving Productivity with Python and HLS for FPGAs, In FCCM. 85-92.
[4] Ye, Hanchen, et al., 2021. ScaleHLS: Scalable HLS through MLIR, In arXiv 2107.11673

Northeastern University

Pacific Northwest NATIONAL LABORATORY
www.pnnl.gov

U.S. DEPARTMENT OF ENERGY | BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

For additional information, contact:
Nicolas Bohm Agostini
nicolas.agostini@pnnl.gov | bohmagostini.n@northeastern.edu

https://gitlab.pnnl.gov/sodalite/soda-opt

Acknowledgements

We thank the PNNL's SODALite team for the inspiring discussions and insightful suggestions.
This work was supported in part by PNNL's Laboratory Directed Research and Development (LDRD) and DARPA's Real-Time Machine Learning (RTML) initiatives.

10/10/21 | PNNL-SA-165850