Improving HLS with Shared Accelerators: A Retrospective

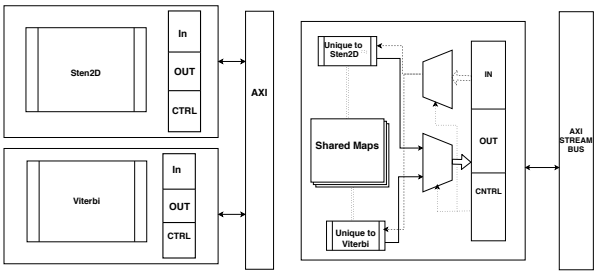
Parnian Mokri
Tufts University
USA

Mark Hempstead
Tufts University
USA

ABSTRACT

This paper is a retrospective paper about our previous ICCAD2020 paper, ReconfAST: An Early Stage identification tool to find Shared Accelerators (SAs). SAs are specialized hardware accelerators that execute very different software kernels but share the common hardware functions between them. Our early detection methodology identifies computationally similar and synthesize-able kernels that are used to build SAs. Our methodology, ReconfAST, transforms one of the compiler's intermediate output, the Abstract Syntax Trees (AST)s, into a new clustered AST (CAST) representation that further removes unneeded nodes and uses a regular expression to match common node configurations. SAs can provide increased coverage if both data flow and control flow similarities between - seemingly very different- workloads are detected. We saw a maximum reduction of above 200% in DSP, 75% reduction for LUTs, and 40% reduction for FFs compared to the smallest Dedicated Accelerators (DAs) with the best speedup.

In this paper, we briefly explain our tool and discuss some challenges we experienced in building a tool that designs accelerators independent of a specific language. One example is the nonintuitive results HLS generates based on optimized mapping heuristics. We suggest ways to improve HLS tools and utilize solutions from other communities to help designers design and evaluate their systems methodically.



(a) Dedicated Accelerators (DAs) **(b) Shared Accelerator (SA)**

Figure 1: Improving workload coverage and area efficiency in many-accelerator systems with Shared Accelerators (SAs).

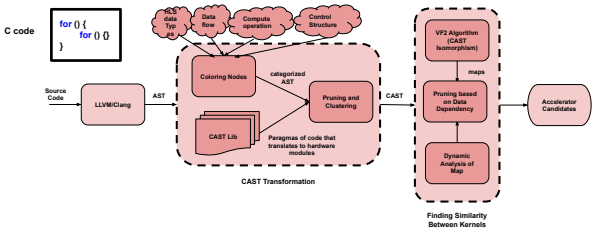


Figure 2: The ReconfAST methodology.

1 RECONFAST

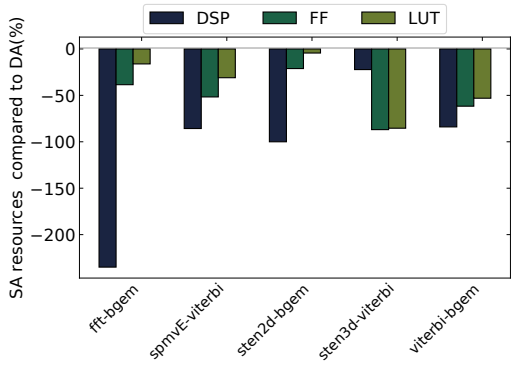
ReconfAST merges ideas from CAD, compiler, and graph theory to build an early-stage detection tool that identifies synthesize-able commonalities between seemingly different workloads from different domains that are used to build Shared Accelerators (SAs). Each SA resembles an ASIC implementation of one software kernel but can accelerate two or more distinctly different kernels. Figure 1 shows a simplified example of a system with accelerators for two MachSuite benchmarks, *Stencil2D* and *Viterbi*. Instead of building a separate Dedicated-Accelerator for each kernel, a single shared accelerator includes hardware for both kernels. Our automated ReconfAST identifies hardware the kernels have in common (a loop with an array multiplication and accumulation in this case) from application source code [4]. Figure 2 shows our methodology in more detail. We build our tool based on the front-end of the llvm-clang suit. Clang is used to generate the ASTs. All the workloads in our paper are written in C/C++, but the tool can support OpenCL and some functional languages. Our tool transforms ASTs into the CAST representations using python transformation scripts. The CAST of each workload is fed into a subgraph isomorphism, the VF2 library. VF2 (or other algorithms for graph isomorphism) has not been, to the best of our knowledge, ever applied to ASTs for use in high-level HLS hardware identification. We then validate the methodology by measuring the dynamic coverage using Valgrind/Callgrind and then analyze the hardware with Vivado HLS.

The main challenge we encountered was learning the heuristics that HLS tools use and excluding non-Pareto optimal results from our implementations to study our hypothesis. Including a full sweep of results reveals nonintuitive trends which baffles reviewers, resulting in rejection of our work. Over the past few years, there have been studies on unpredictable HLS heuristics, and some optimizations have been suggested [1],[2],[3].

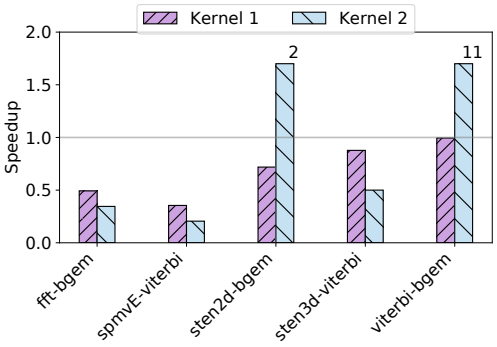
2 IMPLEMENTATION CHALLENGES

Although HLS tools have made considerable strides in recent years, they should be improved further to employ best practices from other communities such as compilers and EDA. For example, loop unrolling in compilers has long been optimized with heuristics that evaluate the cost and benefits of different unrolling factors. However, the HLS tools rely on the designer to discard inefficient designs [3]. When developing ReconfAST we struggled with the unpredictable nature of the mapping heuristic in HLS-based tools.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
LATTE '21, April 15, 2021, Virtual, Earth
© 2021 Copyright held by the owner/author(s).



Change in resources compares to the sum of both DAs.



The speedup of each kernel (best SA) is compared to the best performing DA with least speedup.

Figure 3: Change in area utilization normalized to the cost of the sum of the DAs with smallest number of DSPs; Speedup is calculated with respect to the Dedicated Accelerator

Figure 3 shows a partial comparison of SAs to the sum of their respective dedicated accelerators, where in some cases, the SA's speedup is considerably better than the DA and for not much increase in resource usage. All SAs have been normalized to the dedicated accelerator with the best speedup and smallest footprint on FPGA.

Figure 4 shows the effects optimizations have on different snippets of code like different $loop_i$ where $i \in (1, 2, 3, ..., 5)$ in bbgemm. We observed nonintuitive results for the stencil2d-bbgemm and viterbi-bbgemm SA when one workload (bbgemm) *engulfs* the other (viterbi). Further analysis showed similar patterns in SAs are directly linked to the size of the common subset of two workloads, data dependency, and whether the common subset is a large percentage of the workloads' total execution time. By designing about 10,000 accelerators, we noticed that the most critical factor in finding SAs with good speedup and efficient resource usage is the absence of data dependencies.

3 IMPROVING ACCELERATOR DESIGN PROCESS

Like any relatively new research area, the accelerator design process can improve in many aspects. In our opinion, research in this area is

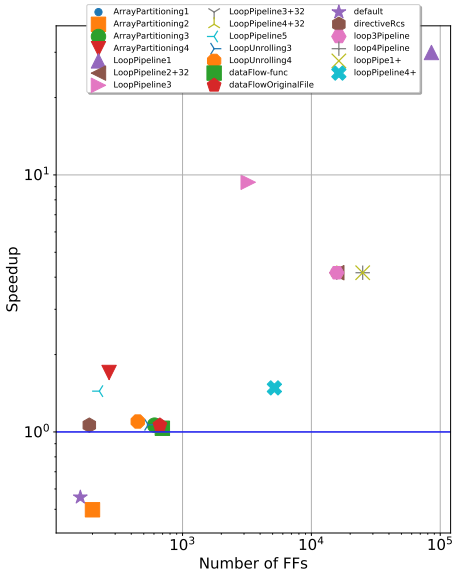


Figure 4: BBgemm dedicated accelerator optimizations affect on speedup and area (FFs) based on different loops (1..5)

hindered by the lack of 1) predictable and transparent tools; and 2) a research community respects and knows how to evaluate research that bisects traditional disciplines.

3.1 Improving Toolchains Toolchains for HLS designs are unpredictable because they do not include information that designers take into account while designing at RTL. We noticed that static analysis of HLS benchmarks, profilers like Valgrind, and compilers (such as Clang's) front-end intermediate outputs, i.e., Abstract Syntax Trees and DAGs provide a more comprehensive view of workloads and result in a more methodical design process. Many techniques from the compiler community can be applied to HLS tools and improve the design process; applying machine learning techniques is one of these approaches [5].

3.2 Interdisciplinary Research and Evaluation Considering the recent development in the field, it is crucial for the community to recognize that when evaluating work that contains ideas from a variety of fields, that the novelty of the work comes from the combination of these ideas for a novel problem, and it is fine if the individual ideas have been published before for other problems. Traditional venues in the Computer Architecture, CAD/EDA, and FPGA communities have different standards and focuses in the review process. A new research track for accelerators, including design methodologies, design tools, and accelerators' systems, needs to be a research focus group on its own with its standards. For example, the FPGA community is more focused on sharing datapaths rather than the idea of ASIC like hardware of Shared Accelerators. In comparison, the computer architecture wanted more evidence of the implementation and was skeptical of our references from the compiler community for using ASTs to find commonalities between source-codes. Or, at minimum, related communities need to be educated on the concerns of accelerator research and how to evaluate interdisciplinary and cross-cutting work properly.

REFERENCES

[1] Y. Choi, P. Zhang, P. Li, and J. Cong. 2017. HLScope+,: Fast and accurate performance estimation for FPGA HLS. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 691–698. https://doi.org/10.1109/ICCAD.2017.8203844

[2] K. Georgopoulos, G. Chrysos, P. Malakonakis, A. Nikitakis, N. Tampouratzis, A. Dollas, D. Pnevmatikatos, and Y. Papaefstathiou. 2016. An evaluation of vivado HLS for efficient system design. In *2016 International Symposium ELMAR*. 195–199. https://doi.org/10.1109/ELMAR.2016.7731785

[3] Rachit Nigam, Sachille Atapattu, Samuel Thomas, Zhijing Li, Theodore Bauer, Yuwei Ye, Apurva Koti, Adrian Sampson, and Zhiru Zhang. 2020. Predictable Accelerator Design with Time-Sensitive Affine Types. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 393–407. https://doi.org/10.1145/3385412.3385974

[4] B. Reagen, R. Adolf, Y.S. Shao, Gu-Yeon Wei, and D. Brooks. 2014. MachSuite: Benchmarks for accelerator design and customized architectures. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*. 110–119. https://doi.org/10.1109/IISWC.2014.6983050

[5] Z. Wang and M. O'Boyle. 2018. Machine Learning in Compiler Optimization. *Proc. IEEE* 106, 11 (2018), 1879–1901. https://doi.org/10.1109/JPROC.2018.2817118