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 (ASTs), 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.

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 LLVM/Clang intermediate output, the Abstract Syntax Trees (ASTs), 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.

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 LLVM/Clang intermediate output, the Abstract Syntax Trees (ASTs), 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.

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

Figure 2: The Reconfast methodology.

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


