ReconfAST: Early-Stage Identification Tool to Detect Similar Hardware Implementations

An HLS add-on to find Shared Accelerators

Parnian Mokri & Maziar Amiraski & Yuelin Liu & Mark Hempstead
Tufts University

Contact Information:
Electrical Engineering
Tufts University
161 College Ave, MA, USA
Email: pmokri01@tufts.edu

Abstract

We propose an early detection tool that complements existing High-level Synthesis tools by identifying computationally similar syntheziable kernels that are used to build Shared Accelerators (SAs). SAs are specialized hardware accelerators that execute very different software kernels but share the common hardware functions between them. SAs can provide increased coverage if both dataflow and control flow similarities between seemingly very different workloads are detected. Existing methods use either dynamic traces or analyze register transfer level (RTL) implementations to find these similarities which requires deep knowledge of RTL and time-consuming design process.

Introduction

Current design methodologies fail to efficiently assist computer architecture designers to maximize overall system performance and workload coverage, especially under a specific system-wide area constraint.

List of contributions:

1. Introduction of **shared accelerators** (SAs) and their architecture implications.
2. **ReconfAST**, a methodology for extracting SA candidates by detaching common kernels using AST representations of source-code.
3. Introducing Clustered-AST (**CAST**): A transformed AST representation that removes unnecessary syntax, summarizes common patterns, and aids in the efficient comparison of hardware similar workloads.
4. Study of ReconfAST on MachSuite demonstrating the potential of SAs to increase coverage.
5. ASIC and FPGA implementations of example SAs and analysis of hardware costs. Classification of workload patterns that result in good and poor performing SA implementations.

Problem: No Methodical Approach to Design Dedicated Accelerators

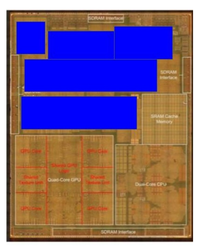

Figure 1: Apple A8 chip from 2016. 1/3 of the chip area is dedicated to Dedicated Accelerators(DA) [4]

Dedicated accelerators (DAs) accelerate only one application and are used extensively across a variety of applications such as Internet of Things, wearables, and implantable devices due to their high performance and low power characteristics.

Solution: Increase The Coverage of Each Accelerator: Shared Accelerator

Shared Accelerators (SAs) can execute multiple kernels by including all of the hardware for both kernels. Common hardware kernels are automatically discovered and shared, reducing area costs.

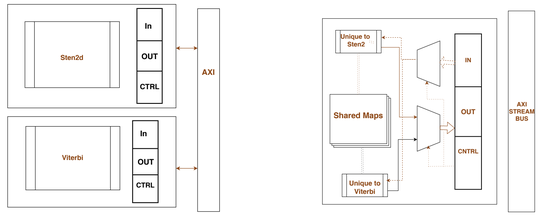

Figure 2: The Dedicated Accelerators on left transform to Shared Accelerators on right and save Area

Shared Accelerators resemble the structure of an ASIC implementation of one software kernel but can accelerate two or more distinctly different kernels.

Methodology

Capture abstract syntax trees from C code, transform trees to clustered representation and use tree-isomorphism algorithm to find similarities between workloads.

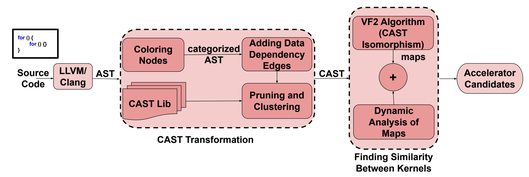

Figure 3: **ReconfAST** methodology, for finding similarities between applications

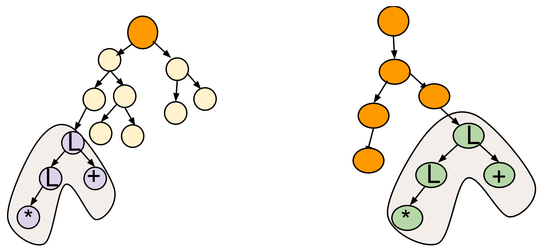

Figure 4: **ReconfAST** finds highlighted shared subtree between two CASTs of two different kernels.

Why use Abstract Syntax Trees?

1. **Simplicity**: ASTs abstract structure expresses the underlying computation clearly.
2. **Annotative**: To design efficient hardware, some additional pieces of information that is not intrinsic to ASTs — such as data-dependency— should be considered. Using the compiler's front-end, it is easy to add this information to ASTs with some post-processing.
3. **Adaptability**: Clang AST supports OpenCL, c,c++, and many functional programming languages such as Haskel and Scala. This adaptability by clang's AST means that our methodology is not bound to a specific language. Unlike EDA tools, this approach can run at the high-level before RTL is written or synthesized.
4. **Cost-efficiency**: Finding similarities between `trees` is much faster and less computationally expensive than between graphs.
5. **Ease of Evaluation**: Each node in AST can be mapped back to source code; it is much easier to pinpoint found patterns back to source code and use HLS tools and design an accelerator for the isomorphic pattern.

Choosing Effective Maps

Our methodology finds statically similar software kernels in pairs of workloads by evaluating if two subtrees are isomorphic. To judge the efficacy of Shared Accelerator candidates we need to estimate the shared subtree's fraction of total execution-time.

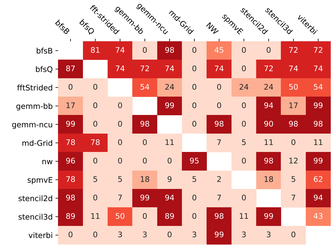

Figure 5: Maximum Dynamic Coverage (percentage of total execution time) measured of the matching (isomorphic) sub-graphs found be-tween the CASTs of each workload

To design shared accelerators, we utilize all the information from ASTs structure and apply the VF2 algorithm only on depth-first subtrees with leaves in a recursive manner starting from the root of the CAST trees.

Hardware Implementation

SAs accelerate workloads by 5x on average, and reduce Flipflop usage by 37%, DSPs by 16%, LUTs by 10% on average over a dedicated accelerators.

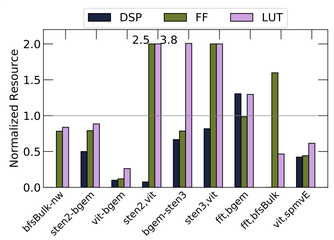

Figure 6: FPGA resource usage normalized to the sum of dedicated accelerators for both kernels. (Some kernels don't use DSPs)

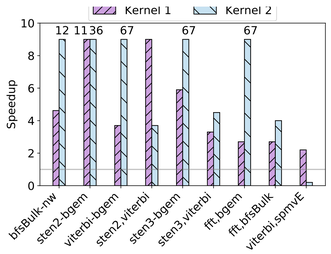

Figure 7: SA speedup normalized to the speedup of the kernel's DA*

Efficient/inefficient SAs:

- Absence or presence of a data dependency
- The relative size of accelerators
- Smaller isomorphic-subtrees with higher occurrence

Forthcoming Research

- *Redefining of functions in DA so they're compatible with SA structure
- Improving HLS tool by finding regularities between applications [2]
- System design
- Finding similarities between multiple workloads

References

[1] Jason Cong, Peng Li, Bingjun Xiao, and Peng Zhang. An Optimal Microarchitecture for Stencil Computation Acceleration Based on Non-Uniform Partitioning of Data Reuse Buffers. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 77:1–77:6, New York, NY, USA, 2014. ACM.

[2] Soha Hassoun and Carolyn McCreary. Regularity extraction via clan-based structural circuit decomposition. In *Proceedings of the 1999 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '99, pages 414–419, Piscataway, NJ, USA, 1999. IEEE Press.

[3] Q. Huang, R. Lian, A. Canis, J. Choi, R. Xi, S. Brown, and J. Anderson. The effect of compiler optimizations on high-level synthesis for FPGAs. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 89–96.

[4] Y.S. Shao and D. Brooks. ISA-independent workload characterization and its implications for specialized architectures. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 245–255, April 2013.

[5] K. Vipin and S. A. Fahmy. Mapping adaptive hardware systems with partial reconfiguration using copr for zynq. In *2015 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 1–8, June 2015.