Improving HLS with Shared Accelerators: A Retrospective

Parnian Mokri

Mark Hempstead
Outline

Accelerators: Past and Future
- Increasing coverage using Shared Accelerators
- Shared Accelerators: Our Solution to Coverage Problem
- ReconfAST: An Adds-on HLS Tool to Identify Shared Accelerators (SAs) at Early Stage

Take away 1: HLS Tools Irregularities Complicates Evaluation Process
Take away 2: Community Support for Accelerator Research

Extra
- Workload Expression with ASTs: Pro and Con
- CAST Transformation
- Tracking Data Dependency
- Evaluation
- ReconfAST results
- Categorizing example reviews and other types of support from the community
Specialized Hardware to Utilize Dark Silicon

Accelerators improve performance of workloads and utilize dark silicon

Application Specific Hardware: The core is turned on to run a specific workload with high speed and turn itself off.
Improving workload coverage and area efficiency in many-accelerator systems with **Shared Accelerators.** Instead of a system with an accelerator dedicated to each kernel, shared accelerators can execute multiple kernels by including all of the hardware for both kernels.
ReconfAST - Methodology

ReconfAST in Comparison to work in computer architecture, EDA, Compiler community

Expressing Workloads with AST
ReconfAST - Methodology
ReconfAST in Comparison to work in computer architecture, EDA, Compiler community

Transforming ASTs to CAST
ReconfAST - Methodology

ReconfAST in Comparison to work in computer architecture, EDA, Compiler community

Finding Similarities between Workloads
We use Machsuite, a HLSfied benchmark suit gathered by Harvard. Machsuite consists of workloads from different domains (from math, to genetics, to graphs).
Take away1: HLS tools have counter-intuitive results

BBgemm dedicated accelerator optimizations affect on speedup and area (FFs)
Take away1: HLS tools have counter-intuitive results

Counter-intuitive optimization results make evaluation of methodologies difficult [2]
Take away1: HLS tools have counter-intuitive results

Improving HLS toolchains with data from profilers: data dependencies, dynamic time. Slides in Extra
- Future work
1. SAs work was hard to publish, each community had different priorities and questions that wasn’t exactly suitable for accelerator research.

2. Providing an established way to evaluate results in accelerator community would help both authors and reviewers. Examples in Extra slides
Take away2: Community Support

1. SAs work was hard to publish, each community had different priorities and questions that wasn’t exactly suitable for accelerator research.

2. Providing an established way to evaluate results in accelerator community would help both authors and reviewers. Examples in Extra slides

3. Having a community for HLS/accelerators or having reviewers that are familiar with research from different communities would help authors.
Thank you for your attention
References


SAs Compared to CGRAs and ASIC

SAs share resources; SA’s Speedup closer to its DA while the energy consumption never gets worse than DA; [3], [4], [5] Data annotated by Yuelin Liu
### Background

ReconfAST in Comparison to work in computer architecture, EDA, Compiler community

<table>
<thead>
<tr>
<th>Technique</th>
<th>Code abstraction</th>
<th>Hardware type</th>
<th>Hardware abstraction</th>
<th>Finds Similarities</th>
</tr>
</thead>
<tbody>
<tr>
<td>DySER</td>
<td>CDFG</td>
<td>CGRA</td>
<td>RTL</td>
<td>No</td>
</tr>
<tr>
<td>Needle</td>
<td>CDFG</td>
<td>CGRA</td>
<td>RTL</td>
<td>No</td>
</tr>
</tbody>
</table>

Computer architecture techniques offload Hotcode on arrays of processing elements. Sharing resources saves area and doesn’t slow-down workloads [3],[6]
Finding similarities between workloads in hardware is possible at different levels. Finding Similarity Saves Area but requires expert knowledge [7], [8].
## Background

ReconfAST in Comparison to work in computer architecture, EDA, Compiler community

<table>
<thead>
<tr>
<th>Technique</th>
<th>Code abstraction</th>
<th>Hardware type</th>
<th>Hardware abstraction</th>
<th>Finds Similarities</th>
</tr>
</thead>
<tbody>
<tr>
<td>DySER</td>
<td>CDFG</td>
<td>CGRA</td>
<td>RTL</td>
<td>No</td>
</tr>
<tr>
<td>Needle</td>
<td>CDFG</td>
<td>CGRA</td>
<td>RTL</td>
<td>No</td>
</tr>
<tr>
<td>DAG-aware</td>
<td>DAG</td>
<td>CGRA</td>
<td>HLS/RTL</td>
<td>Yes (operations)</td>
</tr>
<tr>
<td>Partial Reconfiguration</td>
<td>NA</td>
<td>Traditional FPGAs</td>
<td>HLS/RTL</td>
<td>Somewhat</td>
</tr>
<tr>
<td>ABACUS</td>
<td>AST</td>
<td>Approximate computing circuit</td>
<td>RTL</td>
<td>Not explicitly</td>
</tr>
<tr>
<td>Finding Clones</td>
<td>AST/Feature extraction</td>
<td>NA</td>
<td>NA</td>
<td>YES</td>
</tr>
</tbody>
</table>

Workloads can be expressed in ASTs and ASTs can be used to find similarities between workloads [9],[10]
### Background

ReconfAST in Comparison to work in computer architecture, EDA, Compiler community

<table>
<thead>
<tr>
<th>Technique</th>
<th>Code abstraction</th>
<th>Hardware type</th>
<th>Hardware abstraction</th>
<th>Finds Similarities</th>
</tr>
</thead>
<tbody>
<tr>
<td>DySER</td>
<td>CDFG</td>
<td>CGRA</td>
<td>RTL</td>
<td>No</td>
</tr>
<tr>
<td>Needle</td>
<td>CDFG</td>
<td>CGRA</td>
<td>RTL</td>
<td>No</td>
</tr>
<tr>
<td>DAG-aware</td>
<td>DAG</td>
<td>CGRA</td>
<td>HLS/RTL</td>
<td>Yes (operations)</td>
</tr>
<tr>
<td>Partial Reconfig.</td>
<td>NA</td>
<td>Traditional FPGAs</td>
<td>HLS/RTL</td>
<td>Somewhat</td>
</tr>
<tr>
<td>ABACUS</td>
<td>AST</td>
<td>Approximate computing circuit</td>
<td>RTL</td>
<td>Not explicitly</td>
</tr>
<tr>
<td>Finding Clones</td>
<td>AST/Feature extraction</td>
<td>NA</td>
<td>NA</td>
<td>YES</td>
</tr>
<tr>
<td>ReconfAST</td>
<td>AST</td>
<td>FPGA/ASIC</td>
<td>HLS</td>
<td>YES</td>
</tr>
</tbody>
</table>

ReconfAST’s unique way for workload representation to design Shared Accelerators in Hardware.
DySER (2014): (Dynamically Specializing Execution Resources) architecture integrated into a processor pipeline highlighting the data-parallel extension [3].

Sharing resources is a promising way to utilize area and not lose speedup. Previous work have used Control Data Flow Graph to design Coarse Grained Reconfigurable Arrays.

Needle (2018) “what to Specialize”. NEEDLE uses dynamic profiles to identify hot paths and merges them to create Braids improving code coverage [6].
Finding similarities between workload’s Data flow graphs has been used to save on area but whether it is implemented at HLS or RTL level, focuses on finding computational similarities and overlooks control and communication overhead [7].
Partial Reconfiguration:

- **RTL level**
  - Time consuming
  - Needs expert knowledge

- **HLS level**
  - Time consuming
  - Needs expert knowledge
  - Doesn’t find computational similarities

Partial reconfiguration finds abstract similarities between workloads, the area usage and input and output ports.

Multiple workloads can run on the same part of the hardware [9]
Segmenting Source Code and Finding Similarities Based on Graph Isomorphism [11]

Finding similarities based on features and Neural Networks (DNN) [12]
Workload Representation

- CDFGs result in CGRAs that are too fine-grained [13].
  1. Fine-grained CGRAs increase the communication overhead [14]
  2. Finding isomorphism between them is computationally expensive [15]
  3. Literature shows DNN based on AST is more accurate than CDFG [16] [17]

- ASTs are more abstract which makes finding larger commonalities between two ASTs easier.
- ASTs structure encodes information about locality and instruction dependency.
- Correlating ASTs and source code is done by tracking code line number and marking it for further analysis.
• CDFGs result in CGRAs that are too fine-grained [13].
  1. Fine-grained CGRAs increase the communication overhead [14]
  2. Finding isomorphism between them is computationally expensive [15]
  3. Literature shows DNN based on AST is more accurate than CDFG [16] [17]

• ASTs are more abstract which makes find larger commonalities between two ASTs easier.
Workload Representation

- CDFGs result in CGRAs that are too fine-grained [13].
  1. Fine-grained CGRAs increase the communication overhead [14]
  2. Finding isomorphism between them is computationally expensive [15]
  3. Literature shows DNN based on AST is more accurate than CDFG [16] [17]
- ASTs are more abstract which makes find larger commonalities between two ASTs easier.
- ASTs structure encode information about locality and instruction dependency.
CDFGs result in CGRAs that are too fine-grained [13].

1. Fine-grained CGRAs increase the communication overhead [14]
2. Finding isomorphism between them is computationally expensive[15]
3. Literature shows DNN based on AST is more accurate than CDFG [16] [17]

ASTs are more abstract which makes find larger commonalities between two ASTs easier.

ASTs structure encode information about locality and instruction dependency.

Correlating ASTs and source code is done by tracking code line number and mark it for further analysis.
<table>
<thead>
<tr>
<th>Opcode Type</th>
<th>AST Statement</th>
<th>Reg Expression</th>
<th>CAST Node Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>For(i=0;i&lt;n; i=add(i,1)) while</td>
<td>For,uOp:opr:val,BinOp:opr:val,BinOp:opr:val,W8 while,opr:w,BinOp:opr:val,W8</td>
<td>Loop:For:ExplicitAdder</td>
</tr>
<tr>
<td></td>
<td>For(i=0;i&lt;n;) i=add(i,1)</td>
<td>For,uOp:opr:val,BinOp:opr:val,BinOp:opr:val,W8 while,opr:w,BinOp:opr:val,W8</td>
<td>Loop:While</td>
</tr>
<tr>
<td>Control</td>
<td>Function Call ret if if</td>
<td>FuncCall:W*:returnStmnt with single Literal with Expression (a function or expression) without else statement(not implicit) with else statement(implicit)</td>
<td>Branch:Call:Noret, Bt Branch:Ret:singleLit Branch:Ret:Exp Branch:If:NoElse Branch:If:Else</td>
</tr>
<tr>
<td></td>
<td>decl,Implicit, explicit, parantheisStmtnt,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WildCard</td>
<td>declarations, initialization</td>
<td></td>
<td>wildcard (W) node which will be removed</td>
</tr>
</tbody>
</table>

**AST library and CAST library**
Transforming AST to CAST

- **Adding hardware related information:** Parsing and annotating each AST node with Extra information

- Removing False Negatives
  - Pre-processing and Removal of White Spaces (variable declaration and statements)

- Correlating patterns to hardware modules
  - Building CAST by Categorizing and Clustering

- Accounting for Hardware Optimization:
  - Adding Data Dependencies
Transforming AST to CAST

- **Adding hardware related information:**
  Parsing and annotating each AST node with Extra information

- **Removing False Negatives**
  Pre-processing and Removal of White Spaces (variable declaration and statements)
Transforming AST to CAST

- **Adding hardware related information**: Parsing and annotating each AST node with Extra information
- **Removing False Negatives**
  Pre-processing and Removal of White Spaces (variable declaration and statements)
- **Correlating patterns to hardware modules** Building CAST by Categorizing and Clustering
Transforming AST to CAST

- **Adding hardware related information**: Parsing and annotating each AST node with extra information.
- **Removing False Negatives**: Pre-processing and removal of white spaces (variable declaration and statements).
- **Correlating patterns to hardware modules**: Building CAST by categorizing and clustering.
- **Accounting for Hardware Optimization**: Adding data dependencies.
A SA code candidate can be dependent on data outside of the code snippet in two forms. Data dependency on loop iterator or the computation operation.
Data Dependency inside the acceleration candidate

Data dependency inside the Acceleration candidate. This data dependency creates irregular behavior especially for HLS unrolling pragmas [2].

At this point we only take note of these dependencies to prune maps (isomorphic subtrees between two workloads).
Pros:

- ASTs are more abstract which makes find larger commonalities between two ASTs easier.

- ASTs structure encode information about locality and instruction dependency.

- Correlating ASTs and source code is done by tracking code line number and mark it for further analysis.

Cons:

- ASTs are Syntax specific

- Shows lots of white space: Nodes that don't translate to hardware like variable declaration

These features result in higher false negatives in comparisons: CAST library.
Pros:

- ASTs are more abstract which makes finding larger commonalities between two ASTs easier.
- ASTs structure encode information about locality and instruction dependency.
Pros:

- ASTs are more abstract which makes finding larger commonalities between two ASTs easier.
- ASTs structure encode information about locality and instruction dependency.
- Correlating ASTs and source code is done by tracking code line number and marking it for further analysis.

Cons:

- ASTs are Syntax specific
- Shows lots of white space: Nodes that don’t translate to hardware like variable declaration
Pros:

- ASTs are more abstract which makes finding larger commonalities between two ASTs easier.
- ASTs structure encode information about locality and instruction dependency.
- Correlating ASTs and source code is done by tracking code line number and marking it for further analysis.

Cons:

- ASTs are Syntax specific
- Shows lots of white space: Nodes that don’t translate to hardware like variable declaration

These features result in higher false negatives in comparisons: CAST library.
The size of Best-DA (FFs) is divided to size of all SAs, by applying, loop unrolling and pipe-lining, data flow, memory resources, and array partitioning with different parameters.

Latency of all SA optimizations is normalized to best DA.
Largest pairwise isomorphic subtree (Maps) are shown as an example.

Map1 is a shared map between NW and BFS. It is repeated 3 times in NW, which is why the map has 96% of execution time but only once in BFS.

Maximum Dynamic Coverage (percentage of total execution time) measured of the matching (isomorphic) sub-graphs found between the CASTs of each workload.
Conclusion

- ReconfAST is a high-level tool to find similarities between kernels that enables the design of **shared accelerators**
Conclusion

- ReconfAST is a high-level tool to find similarities between kernels that enables the design of **shared accelerators**
- SAs are application specific accelerators that can accelerate more than 1 workload
ReconfAST is a high-level tool to find similarities between kernels that enables the design of **shared accelerators**

SAs are application specific accelerators that can accelerate more than 1 workload

SAs are useful when there is a system-wide area constraint and DAs for all necessary kernels won’t fit
Conclusion

- ReconfAST is a high-level tool to find similarities between kernels that enables the design of shared accelerators.
- SAs are application specific accelerators that can accelerate more than 1 workload.
- SAs are useful when there is a system-wide area constraint and DAs for all necessary kernels won’t fit.
- Best cases for SAs Reduce Resources and do not slow down the workloads’ performance.
ReconfAST is a high-level tool to find similarities between kernels that enables the design of **shared accelerators**

SAs are application specific accelerators that can accelerate more than 1 workload

SAs are useful when there is a system-wide area constraint and DAs for all necessary kernels won’t fit

Best cases for SAs Reduce Resources and do not slow down the workload’s performance

For evaluation, we implemented different maps and applied 15 different optimizations to both SAs and DAs; (including loop pipelining and unrolling, array partitioning and reshaping, dataflow, resource, and their combinations);
Evaluation of ReconfAST

Does ReconfAST solve the problem?

We use Machsuite a HLSfied benchmark suit gathered by Harvard. Machsuite consists of workloads from different domains (from math, to genetics, to graphs)[1].
The size of Best-DA (FFs) is divided to size of all SAs. 
ReconfAST-ICCAD2020 10000 SAs, by applying, loop unrolling and pipe-lining, data flow, memory resources, and array partitioning with different parameters.

Latency of all SA optimizations is normalized to best DA.
Best Cases for SAs
Reduce Resource Usage and Doesn’t Slow Down the Workload’s Performance

Change in resources compares to the sum of both DAs.

If one kernel is large enough that the ther workload (especially its map) can unroll on it, then the second workload's speedup improves.

The speedup of each kernel (best SA) is compared to the best performing DA with least speedup.
1. A new track for accelerator research to avoid tunnel-vision review process

What it would look like if you’d use CDFGs

Finding similarities between ASTs is not novel

2. Standardizing the evaluation process
Future work

- Improving the methodology to find similarities between multiple workloads
- System design of Shared Accelerators
Future work

• Improving the methodology to find similarities between multiple workloads

• System design of Shared Accelerators
  1. Scheduling workloads on SAs
  2. Memory allocation of SAs
  3. Communication between SA and the rest of the system