Early-stage Automated Accelerator Identification Tool for Embedded Systems with Limited Area

Parnian Mokri

Mark Hempstead
Introduction

**Presenter:** Parnian Mokri Ph.D. candidate at Tufts University  
**Adviser:** Dr. Mark Hempstead  
**General research interests:**  
Applying techniques from Computer Architecture, Graph Theory, EDA, and Compilers, to address the challenges of heterogeneous architecture, especially in embedded systems with limited area and power.
Outline

Accelerators: Past and Future
  Designing Accelerator’s Challenge: Accelerator’s Coverage
Shared Accelerators: Our Solution to Coverage Problem
ReconfAST: Our early-stage methodology to identify Shared Accelerators (SAs)
  Methodology Overview
  Workload Expression with ASTs: Pro and Con
    CAST Transformation
    Tracking Data Dependency
Evaluation
ReconfAST’s Accelerators Selection: Dynamic Coverage Analysis
Shared Accelerators Implementation in Hardware
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.
What needs to be Accelerated?

1. Dedicated Accelerators cost resources
2. Most of them are very workload specific
3. Specifically designed for certain standards and domains
4. Covering workloads that users or companies care about
What needs to be Accelerated?

1. Dedicated Accelerators cost resources
2. Most of them are very workload specific
3. Specifically designed for certain standards and domains
4. Covering workloads that users or companies care about

**Increasing Accelerators’ Coverage**: Selecting accelerators and finding accelerators to cover many workloads from different domains is a challenging and open problem
Our Solution to Coverage Problem: Shared Accelerator

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: An Early Stage identification tool to find Shared Accelerators (SAs)

Find similarities + share resources + design hardware
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
Workload Representation

- CDFGs result in CGRAs that are too fine-grained [1].
  1. Fine-grained CGRAs increase the communication overhead [2]
  2. Finding isomorphism between them is computationally expensive [3]
  3. Literature shows DNN based on AST is more accurate than CDFG [4] [5]
Workload Representation

- CDFGs result in CGRAs that are too fine-grained [1].
  1. Fine-grained CGRAs increase the communication overhead [2]
  2. Finding isomorphism between them is computationally expensive [3]
  3. Literature shows DNN based on AST is more accurate than CDFG [4] [5]
- ASTs are more abstract which makes find larger commonalities between two ASTs easier.
Workload Representation

- CDFGs result in CGRAs that are too fine-grained [1].
  1. Fine-grained CGRAs increase the communication overhead [2]
  2. Finding isomorphism between them is computationally expensive[3]
  3. Literature shows DNN based on AST is more accurate than CDFG [4] [5]

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

- ASTs structure encode information about locality and instruction dependency.
Workload Representation

- CDFGs result in CGRAs that are too fine-grained [1].
  1. Fine-grained CGRAs increase the communication overhead [2]
  2. Finding isomorphism between them is computationally expensive[3]
  3. Literature shows DNN based on AST is more accurate than CDFG [4] [5]
- 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.
int main () {
    int a;
    int b=0,c=0;
    for(a = 10; a < 20; ++a) {
        c = callSum(a,b);
    }
    return 0;
}
Con: Expressing Workloads with AST (for Designing Hardware)

- ASTs are Syntax specific
- Shows lots of white space: Nodes that don’t translate to hardware like variable declaration
Con: Expressing Workloads with AST (for Designing Hardware)

- 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.
Con: Expressing Workloads with AST (for Designing Hardware)

- 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.

<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>Loop</td>
<td>For(i=0;i&lt;n;i++)</td>
<td>For,uOp:opr:val,BinOp:opr:int,bin:Opr,val,W*</td>
<td>Loop:For:I</td>
</tr>
<tr>
<td></td>
<td>For(i=0;i&lt;n;) i=add(i,1) while</td>
<td>For,uOp:opr:val,BinOp:opr:int,bin:Opr,val,W* while,opr:w,bInOp:opr:val,W8</td>
<td>Loop:For:ExplicitAdder</td>
</tr>
<tr>
<td></td>
<td>while,opr:w,bInOp:opr:val,W8</td>
<td>Loop:While</td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>Function Call ret ret if if</td>
<td>FuncCall:W*:returnStmt with single Literal with Expression (a function or expression) without else statement(not implicit) with else statement(implicit)</td>
<td>Branch:Call:Noret, Bt</td>
</tr>
<tr>
<td>WildCard</td>
<td>declarations, initialization</td>
<td>decl,Implicit, explicit, parantheisStmtnt, wildcard (W) node which will be removed</td>
<td></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

UniqueID:Line9:BinaryOp:+
:Operand1UniqueID:Operand2UniqueID

An operation node in CAST Adding unique identifier, code-line number, Type of operation, Operation, operands
ASTs of two functionally similar workloads

one CAST represents different ASTs.
Data Dependency between the acceleration candidate and the rest of the code

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. This data dependency creates irregular behavior especially for HLS unrolling pragmas [6]

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

Does ReconfAST solve the problem?

Evaluating ReconfAST’s SA candidates using Vivado HLS

We use Machsuite a HLSfied benchmark suit gathered by Harvard. Machsuite consists of workloads from different domains (from math, to genetics, to graphs)[7].
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 not 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.
Range of Speedup and Area of SAs

The size of Best-DA (FFs) is divided to size of all SAs 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 other 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.
• 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
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
• 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 workloas’ 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 workflows’ 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);
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
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; [8], [9], [10] Data annotated by Yuelin Liu
Computer architecture techniques offload Hotcode on arrays of processing elements. Sharing resources saves area and doesn’t slow-down workloads [8],[11]
Finding similarities between workloads in hardware is possible at different levels. Finding Similarity Saves Area but requires expert knowledge [12], [13].

<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>
</tbody>
</table>
**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 [14],[15]
### 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>
<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 [8].

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.

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 [12].
Partial Reconfiguration:

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

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

Multiple workloads can run on the same part of the hardware [14]

Partial reconfiguration finds abstract similarities between workloads, the area usage and input and output ports
Segmenting Source Code and Finding Similarities Based on Graph Isomorphism [16]

Finding similarities based on features and Neural Networks (DNN) [17]