From High-Level Frameworks to custom Silicon with SODA

Serena Curzel, Nicolas Bohm Agostini, Reece Neff, Ankur Limaye, Jeff (Jun) Zhang, Vinay Amatya, Marco Minutoli, Vito Giovanni Castellana, Joseph Manzano, David Brooks, Gu-Yeon Wei, Fabrizio Ferrandi, Antonino Tumeo
Overview

• The SODA Synthesizer is a modular, multi-level, interoperable, extensible, open-source hardware compiler from high-level programming frameworks to silicon
  ✓ Compiler-based frontend, leveraging the MultiLevel Intermediate Representation (MLIR)
  ✓ Compiler-based backend, leveraging state-of-the-art High-Level Synthesis (HLS) techniques

• Generates synthesizable Verilog for a variety of targets, from Field Programmable Gate Arrays (FPGAs) to Application Specific Integrated Circuits (ASICs)

• Optimizations at all levels are performed as compiler optimization passes
Results

ASIC accelerators for LeNet layers

Useful links

SODA-OPT

Panda-Bambu HLS (v 0.9.7)

SODA Docker Image

SODA Tutorial: DATE 2022
Motivations

• Data Science algorithms, Machine Learning models and frameworks are quickly evolving


- Increasing number of layers and parameters (ResNet, VGG, Transformers...)
- New network architectures (GNN, LSTM, Reinforcement Learning...)
- Compression techniques (Quantization, pruning...)
- New programming environments (TensorFlow, PyTorch, MXNet...)

• Increased complexity and tight performance/power/area constraints (especially on edge devices) require domain-specific accelerators
Motivations

• Existing accelerators start from specific models (e.g., CNNs) or only try to accelerate specific computational patterns
  - Designing hardware accelerators by hand is complex and time-consuming
  - Hardware designers may want to explore different design trade-offs, depending on the application requirements

• Agile Hardware Design and Prototyping is required
  - Quickly transition from algorithm formulation to accelerator implementation
  - Sufficient design space exploration knobs
  - Minimal human interaction
Our solution: the SODA Synthesizer

- The SODA Synthesizer is a modular, multi-level, interoperable, extensible, open-source hardware compiler from high-level programming frameworks to silicon
  - Compiler-based **frontend**, leveraging the MultiLevel Intermediate Representation (MLIR)
  - Compiler-based **backend**, leveraging state-of-the-art High-Level Synthesis (HLS) techniques
- Generates **synthesizable Verilog** for a variety of targets, from Field Programmable Gate Arrays (FPGAs) to Application Specific Integrated Circuits (ASICs)
- Optimizations at all levels are performed as **compiler optimization** passes

---


Frontend: SODA-OPT

- **SODA-OPT**: Search, Outline, Dispatch, Accelerate frontend optimizer
- Employs and embraces the MLIR framework
  - MLIR: Multi-Level Intermediate Representation
  - Used in TensorFlow, TFRT, ONNX-MLIR, others
- Uses MLIR and compiler passes to:
  - Identify code regions for hardware generation
  - Perform high-level optimizations (dataflow transformations, data-level and instruction-level parallelism extraction)
  - Generate interfacing code and runtime calls for microcontroller
Frontend: SODA-OPT

- SODA-OPT implements optimizations as compiler passes

**Structural**
- Tiling
- Unrolling

**Memory**
- Temporary Buffer Allocation
- Alloca Buffer Promotion

**Avoid Redundancy and Promote Reuse**
- Scalar Replacement of Aggregates
- Early Alias Analysis
- Outlining

**Avoid Unnecessary Operations**
- Dead Code Elimination
- Common Sub-expression Elimination

- Single basic block containing the compute intensive part of the kernel
  More freedom to schedule operations

- Increased instruction-level parallelism
  Schedule independent arithmetic operations on the same cycle when their inputs are available

- Increased data-level parallelism
  Schedule operations into different memory units on the same cycle

- Avoid unnecessary reads from kernel arguments
  Reduce expensive accesses to external memory

- Reuse read results, aggregate on scalars
  Save scalar values loaded from memory and intermediate results in registers rather than performing repeated memory accesses

- Early alias analysis
  Schedule memory operations independently on regions that don’t alias

- Remove redundant or unnecessary operations
  Avoid wasting resources
Backend: High-Level Synthesis

• The synthesizer backend takes as input the optimized low-level IR and generates the hardware descriptions of the accelerators
• The main HLS backend is PandA-Bambu, an open-source state-of-the-art high-level synthesis (HLS) tool
  ✓ We are key contributors to Bambu, with parallel accelerator designs, modular HLS, and ASIC support
  ✓ Automated testing and verification
Backend: High-Level Synthesis

• We also support integration with Xilinx Vitis HLS through its open-source LLVM frontend
• The SODA Synthesizer has interfaces with multiple open-source and commercial backends
  ✓ Xilinx Vivado, Intel Quartus (FPGA)
  ✓ OpenROAD, Synopsys Design Compiler (ASIC)
• Automated path to FPGA bitstream or GDS2 files
Examples of generated accelerators

- LeNet model imported from TensorFlow
- Each operator is synthesized to an **ASIC accelerator** (OpenROAD FreePDK 45nm)
- SODA-OPT optimized accelerators are bigger, but also much **faster**
Examples of generated accelerators

- PolyBench kernels
- Outperforming state-of-the-art HLS tools and frontends

<table>
<thead>
<tr>
<th>Opt. Strategy</th>
<th>Kernel Size</th>
<th>No High Level_opts.</th>
<th>SODA-OPT Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>symm</td>
<td>421</td>
<td>2,928</td>
<td>21,400</td>
</tr>
<tr>
<td>three_mm</td>
<td>388</td>
<td>3,087</td>
<td>25,010</td>
</tr>
<tr>
<td>two_mm</td>
<td>315</td>
<td>2,475</td>
<td>20,258</td>
</tr>
<tr>
<td>gemm</td>
<td>186</td>
<td>1,446</td>
<td>11,922</td>
</tr>
<tr>
<td>dolgten</td>
<td>277</td>
<td>4,282</td>
<td>67,666</td>
</tr>
<tr>
<td>bircg</td>
<td>129</td>
<td>518</td>
<td>2,058</td>
</tr>
<tr>
<td>mvt</td>
<td>130</td>
<td>514</td>
<td>2,051</td>
</tr>
<tr>
<td>gemver</td>
<td>283</td>
<td>1,118</td>
<td>4,393</td>
</tr>
<tr>
<td>gesummv</td>
<td>162</td>
<td>578</td>
<td>4,172</td>
</tr>
<tr>
<td>atax</td>
<td>132</td>
<td>523</td>
<td>2,067</td>
</tr>
<tr>
<td>syrk2k</td>
<td>186</td>
<td>1,310</td>
<td>9,018</td>
</tr>
<tr>
<td>ssyrk</td>
<td>142</td>
<td>990</td>
<td>6,714</td>
</tr>
<tr>
<td>trmm</td>
<td>46</td>
<td>532</td>
<td>4,402</td>
</tr>
</tbody>
</table>

Conclusions

• The SODA toolchain provides an end-to-end compiler-based design flow from the formulation of an algorithm to the deployment of custom hardware accelerators
  • Multi-level, modular, and extensible
  • Promotes agile hardware design
  • Based on open-source technologies, and integrated with proprietary tools
• Start using SODA today with these links: