SODA: Stencil with Optimized Dataflow Architecture

Yuze Chi, Jason Cong, Peng Wei, Peipei Zhou
University of California, Los Angeles
{chiyuze, cong, peng.wei.prc, memoryzpp}@cs.ucla.edu
What is stencil computation?
What is Stencil Computation?

- A **sliding window** applied on an array
  - Compute output according to some fixed pattern using the stencil window
- Extensively used in many areas
  - Image processing, solving PDEs, cellular automata, etc.
- Example: a 5-point blur filter with uniform weights

```c
void blur(float input[N][M],
         float output[N][M]) {
    for(int j = 1; j < N-1; ++j) {
        for(int i = 1; i < M-1; ++i) {
            output[j][i] = (input[j-1][i] +
                            input[j][i-1] +
                            input[j][i  ] +
                            input[j][i+1] +
                            input[j+1][i] ) * 0.2f;
        }
    }
}
```
How do people do stencil computation?
Stencil Optimization #1: Data Reuse

- Non-uniform partitioning–based line buffer (DAC’14)

  - Full data reuse, 1 PE
  - Optimal size of reuse buffer
  - Optimal number of memory banks

- But how to parallelize?

DAC’14: An Optimal Microarchitecture for Stencil Computation Acceleration Based on Non-Uniform Partitioning of Data Reuse Buffers
Stencil Optimization #2: Temporal Parallelization

- Multiple iterations / stages chained together (ICCAD’16)
  - More iterations ⇒ better throughput
  - Communication-bounded ⇒ Computation-bounded

- Parallelization within each iteration?

ICCAD’16: A Polyhedral Model-Based Framework for Dataflow Implementation on FPGA Devices of Iterative Stencil Loops
Stencil Optimization #3: Spatial Parallelization

Element-Level Parallelization (FPGA’18)
- Fine-grained parallelism
- Private reuse buffers w/ duplication

Tile-Level Parallelization (DAC’17)
- Coarse-grained parallelism
- Private reuse buffers

DAC’17: A Comprehensive Framework for Synthesizing Stencil Algorithms on FPGAs using OpenCL Model
FPGA’18: Combined Spatial and Temporal Blocking for High-Performance Stencil Computation on FPGAs Using OpenCL
Stencil Optimization: Parallelization

- Previous works use private reuse buffers
  - $k$ PEs require $S_r \times k$ storage
    - $S_r$: reuse distance, the distance from the first data element to the last data element
  - *Sub-optimal* buffer size
  - *Not scalable* when $k$ increases
Can we do better?
SODA as a Microarchitecture: Data Reuse

- For $k = 3$ PEs

- $k$ PEs only require $S_r + k - 1$ storage
- **Full** data reuse
- **Optimal** buffer size
- **Scalable** when $k$ increases
SODA as a Microarchitecture: Spatial Parallelization

FW: forwarding module, implements FIFO and distributes data

PF: compute module, implements the kernel function

Reuse Buffer

input

output

input

input

output
SODA as a Microarchitecture: Temporal Parallelization
How do you program such a messy fancy architecture?
Stencil Optimization #4: Domain-Specific Language Support

◆ Complex hardware architecture

◆ How to program?
  ▪ Template-based
    • DAC’14, ICCAD’16, FPGA’18
  ▪ Domain-specific language (DSL)
    • Darkroom, Halide, Hipacc...

◆ SODA uses a DSL
  ▪ Flexible
  ▪ Programmable

```plaintext
kernel: jacobi2d
input float: in(3000, *) # specifies the tile size
output float: out(0, 0) = (in(0, -1) +
  in(-1, 0) + in(0, 0) + in (1, 0) +
  in(0, 1)) * 0.2 f

unroll factor: 3
iterate factor: 2
# SODA supports multiple stages:
# local float: tmp(0, 0) = (in(0, -1) +
#   in(-1, 0) + in(0, 0) + in(1, 0) +
#   in(0, 1)) * 0.2 f
# output float: out(0, 0) = (tmp(0, -1) +
#   tmp(-1, 0) + tmp(0, 0) + tmp(1, 0) +
#   tmp(0, 1)) * 0.2 f
# SODA supports multiple arrays as input:
# local float: t(0, 1) = in(0, 0) + tmp(0, 2)
```
SODA as an Automation Framework

User-Defined C++ Host Application
User-Defined SODA DSL Kernel

User-Defined Input
sodac (SODA)

Xilinx OpenCL API
Dataflow HLS Kernel

Intermediate Code
#PEs (up to $10^2$)
Tile size (up to $10^6$)

How to explore?

xocc (SDAccel)

Design-Space Exploration (SODA)

Large Design Space (up to $10^{10}$)

#Iteration (up to $10^2$)

Host Program
FPGA Bitstream

Executable Results

Large Design Space
(up to $10^{10}$)

#PEs
(up to $10^2$)

Tile size
(up to $10^6$)

#Iteration
(up to $10^2$)
How do you explore such a huge design space?
SODA as an Exploration Engine: Resource Modeling

Resource Modeling Flow

- SODA DSL input
  - sodac
  - HLS code of each module
  - Number of each module
  - Module model database

Has resource model for module?

- Yes: Complete resource model

- No: Run HLS for module
  - HLS code of each module
  - Number of each module for each module

Module model database

Modularized Design Enabling Accurate Architecture-Specific Modeling

PE: compute module, implements the kernel function
FW: forwarding module, implements FIFO and distributes data
SODA as an Exploration Engine: Performance Modeling

Performance Roofline Model

Throughput limited by external bandwidth

Throughput achieved

Throughput provided by PEs

Throughput

#PEs / stage

0
SODA as an Exploration Engine: Design-Space Pruning

- **Unroll factor** $k$
  - Only powers of 2 make sense due to the memory port

- **Iteration factor** $q$
  - Bounded by available resources, $kq \leq 10^2$

- **Tile size** $T_0, T_1, ...$
  - Bounded by available on-chip storage
  - Searched via branch-and-bound

- Can finish exploration in up to 3 minutes
What does your result look like?
Experimental Results: Model Accuracy

- Model prediction targets
  - Resource modeling target: **post-synthesis** resource utilization
  - Performance modeling target: **on-board** execution throughput

<table>
<thead>
<tr>
<th>Prediction Item</th>
<th>BRAM</th>
<th>DSP</th>
<th>LUT</th>
<th>FF</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Error</td>
<td>1.84%</td>
<td>0%</td>
<td>6.23%</td>
<td>7.58%</td>
<td>4.22%</td>
</tr>
</tbody>
</table>
Experimental Results: Performance Comparison

Non-Iterative Stencil

Normalized Performance

Normalized Performance

Iterative Stencil

Synthesis Tool: SDAccel / Vivado HLS 2017.2

FPGA: ADM-PCIE-KU3 w/ XCKU060

CPU: Intel Xeon E5-2620 v3 x2
What are the takeaways?
SODA: Stencil with Optimized Dataflow Architecture

◆ SODA is a Microarchitecture
  ▪ Flexible & scalable reuse buffers for multiple PEs

◆ SODA is an Automation Framework
  ▪ From DSL to hardware, end-to-end automation

◆ SODA is an Exploration Engine
  ▪ Optimal parameters via model-driven exploration
References

▪ DAC’14: An Optimal Microarchitecture for Stencil Computation Acceleration Based on Non-Uniform Partitioning of Data Reuse Buffers, Cong et al.

▪ ICCAD’16: A Polyhedral Model-Based Framework for Dataflow Implementation on FPGA Devices of Iterative Stencil Loops, Natale et al.


▪ FPGA’18: Combined Spatial and Temporal Blocking for High-Performance Stencil Computation on FPGAs Using OpenCL, Zohouri et al.
Acknowledgments
This work is partially supported by the Intel and NSF joint research program for Computer Assisted Programming for Heterogeneous Architectures (CAPA), and the contributions from Fujitsu Labs, Huawei, and Samsung under the CDSC industrial partnership program. We thank Amazon for providing AWS F1 credits.