DSAGEN: Synthesizing Programmable Spatial Accelerators

Jian Weng, Sihao Liu, Vidushi Dadu, Zhengrong Wang, Preyas Shah, Tony Nowatzki
University of California, Los Angeles
May 15th, 2020
Specialized Accelerators

Specialized architecture often occupies 1/5~1/3 of publications in top conferences.

Existing Domain-Specific Approach:

- High-level Abstraction
- Compiler
- Sw/Hw Interface
- Specialized Mechanisms
- Idioms
- Apps
DSAGEN: Decoupled Spatial Accelerator Generator
Domain Specific

Apps → Design Space Explorer → Specialized Hardware
DSAGEN: Decoupled Spatial Accelerator Generator Domain Specific

Transformations with tradeoffs on performance and hardware cost
Outline

- **Design Space — Decoupled-Spatial Architecture**
  - Insight from Prior Work
  - The Programming Paradigm
  - Design Space: Hardware Primitives (& Composition)

- Compilation
- Design Space Exploration
- Evaluation
• Decoupled-Spatial Paradigm
  • Decoupled Compute/Memory
  • Spatially exposed resources

• Design Space
  • Composing hardware with simple primitives
  • Architecture Description Graph
Background: Decoupled-Spatial Architecture

```
for (int i = 0; i < n; ++i)
    c[i] += a[i] * b[i];
```

![Diagram of decoupled-spatial architecture](image-url)
## Hardware Primitives: Processing Element & Switch

<table>
<thead>
<tr>
<th>Hardware Cost: Low</th>
<th>Dedicated (=1)</th>
<th>Shared (&gt;1)</th>
</tr>
</thead>
</table>
| **Statically Scheduled** | **“Systolic” 1x Area**  
+ No contention  
- Harder to map  
- Higher power  
*Softbrain | **“CGRA” 2.6x Area**  
+ Better resource utilization  
- Harder to map  
*Conventional CGRA | |
| **Dynamically Scheduled** | **“Ordered Dataflow” 2.1x Area**  
+ Better flexibility  
*SPU | **“Tagged Dataflow” 5.8x Area**  
+ Better flexibility  
+ Better resource utilization  
*Triggered Instruction | |

<table>
<thead>
<tr>
<th>Hardware Cost: High</th>
<th>Instruction Buffer</th>
<th>Function Unit</th>
<th>Register File</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MUX</strong></td>
<td><strong>MUX</strong></td>
<td><strong>MUX</strong></td>
<td><strong>MUX</strong></td>
</tr>
</tbody>
</table>

- "Systolic": 1x Area
  - No contention
  - Harder to map
  - Higher power
  - Softbrain

- “CGRA”: 2.6x Area
  - Better resource utilization
  - Harder to map
  - Conventional CGRA

- "Ordered Dataflow": 2.1x Area
  - Better flexibility
  - SPU

- "Tagged Dataflow": 5.8x Area
  - Better flexibility
  - Better resource utilization
  - Triggered Instruction
Hardware Primitives: Memory

- Memory
  - Size
  - Bandwidth
  - Indirect Support
    - $a[b[i]]$
  - Atomic Update
    - $a[b[i]] += 1$
Examples of ADG

(a) Softbrain

(b) MAERI

(c) Diannao

(d) Data Path of Complex Mul.
Outline

• Decoupled-Spatial Architecture

• Compilation
  • High-Level Abstraction
  • Hardware-Aware Modular Compilation

• Design Space Exploration

• Evaluation
Compiling High-Level Lang. to Decoupled Spatial

How to abstract diverse underlying features with a **unified** high-level interface?

• Programmer Hints
  • Which code regions are offloaded onto the spatial accelerator.
  • Which memory accesses can be decoupled intrinsics.
  • Which offloaded regions should be concurrent.
An example of pragma annotation

```c
#pragma config ← The offloaded region in this compound body are concurrent
{
    #pragma stream ← The memory accesses below will be restricted
    for (i=0; i<n; ++i)
        #pragma offload ← The computational instructions below will be offloaded
        for (j=0; j<n; ++j)
            a[i*n+j] += b[c[j]] * d[i*n+j];
}
```

The computational instructions below will be offloaded
The memory accesses below will be restricted
The offloaded region in this compound body are concurrent
Compiling High-Level Lang. to Decoupled Spatial

How to hide the diversity of underlying hardware?

- Modular Transformation
  - Specialized Hardware features often dictate the code transformation
  - A fallback is required when the hardware feature is not available
Modular Transformation

```c
#pragma config
{
  #pragma stream
  for (i=0; i<n; ++i)
    #pragma offload
    for (j=0; j<n; ++j)
      a[i*n+j] += b[c[j]] * d[i*n+j];
}
```

Inspect the hardware features to generate corresponding version of indirect memory

// With indirect support
Read c[0:n], stream0
Indirect b, stream0, stream1

// Without indirect support
for (j=0; j<n; ++j)
  Scalar b[c[j]], stream0

```
```
Compiling High-Level Lang. to Decoupled Spatial Executable

How is the dependence graph of computational instructions mapped?
Spatial Mapping

1. Placement: Map instruction to PE’s with corresponding capability.
2. Routing: Routing the dependence edges thru the spatial network.
3. Timing: If necessary, balance the timing of data arrival
   • If one of 1-3 is not successful, revert some nodes and repeat 123

How is the dependence graph of computational instructions mapped?
Outline

• Decoupled-Spatial Architecture
• Compilation

• Design Space Exploration
  • Drive the Search
  • Evaluating Design Points
  • Repairing the Mapping

• Evaluation
Design Space Exploration

- Create a new ADG based on the current
- Evaluate the sw/hw pair
- Design Space Exp.

Architecture Description Graph (ADG)

Map
Remap
Multiple Xformed IR
Estimation Model

• Performance
  • Spatial architecture essentially enables hardware specialized sw-pipelining
  • The ratio of data availability determines the performance
  • Perf=#Inst * (Activity Ratio)

The model has mean performance error of 7%, and with maximum error 30%.

• Power/Area
  • Synthesis can be time consuming
  • A regression model can predict the trend of hardware cost

Model Validation

![Model Validation Graph]

- Dense NN
- MachSuite
- Sparse CNN

- Area
- Power

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The graph shows the comparison between model and synthesis for different models.
Repairing the Spatial Mapping

// Original Code
for (i=0; i<n; ++i)
c[i]+=a[i]*b[i];

No Unrolling:

Unroll by 2:

Sync
Sync
Sync
Sync
Hardware/Software Interface Generation

• How to configure accelerator with arbitrary topology?
  • Reuse the data path for configuration
  • Find path(s) that cover(s) all the components
  • A heuristic based heuristic algorithm to minimize the longest path of configuration

• For a graph with m nodes covered by n paths, the longest path cannot be shorter than $\left\lceil \frac{m}{n} \right\rceil$.
• We only introduces 40% overhead over the ideal bound.
Outline

• Decoupled-Spatial Architecture
• Compilation
• Design Space Exploration

• Evaluation
  • Methodology
  • Compiler
  • Design Space Exploration
Methodology

• Performance
  • Gem5 RISCV in-order core integrated with a cycle-accurate spatial accelerator simulator
    • The in-order core is extended with stream decoupled ISA

• Power/Area
  • All the components are implemented in Chisel RTL
  • Synthesized in Synopsys DC 28nm @1.00GHz
  • SRAM power/area are estimated by CACTI 7.0
Compiler Performance

• Softbrain — MachSuite
  • Versatile accelerator can handle moderate irregularity
• SPU — Histogram, and Key Join
  • Accelerator specialized for irregular workloads
• REVEL and Trigger — DSP
  • Accelerator specialized for imperfect loop body
• MAERI — PolyBench
  • Accelerator for neural network
Compiler Performance

MachSuite (Softbrain)

Irregular (SPU)

DSP (REVEL)

DSP (Trigger)

PolyBench (MAERI)

Compiled

Manual
Design Space Explorer

• Workloads
  • Dense Neural Network
  • MachSuite
  • Sparse Convolutional Neural Network

• Initial Design
  • A 5x5 mesh with all capability (arithmetic, control, and indirect)

• Objective: perf²/mm²
Design Space Explorer

Area Breakdown

Power Breakdown

Sparse CNN: 24h
MachSuite: 19.2h
Dense NN: 16h
# Conclusion

<table>
<thead>
<tr>
<th></th>
<th>HLS</th>
<th>Manual</th>
<th>DSAGEN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frontend</strong></td>
<td>C+Pragma</td>
<td>DSL/Intrinsics, etc.</td>
<td>C+Pragma</td>
</tr>
<tr>
<td><strong>Design Flow</strong></td>
<td>Nearly Automated</td>
<td>Manual</td>
<td>Nearly Automated</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td><strong>A Single Application</strong></td>
<td>Multiple Target Applications</td>
<td><strong>Multiple Target Applications</strong></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td><strong>Application-Specific</strong> Accel.</td>
<td>ASIC/Programmable Accel.</td>
<td><strong>A Programmable Accelerator</strong></td>
</tr>
<tr>
<td><strong>Design Space</strong></td>
<td>Limited</td>
<td>Rich</td>
<td>Rich</td>
</tr>
</tbody>
</table>
Q&A

• Our framework is working in progress at: https://github.com/PolyArch/dsa-framework

• All the questions and comments are welcomed