SODA: Stencil with Optimized Dataflow Architecture

Yuze Chi, Jason Cong, Peng Wei, Peipei Zhou

University of California, Los Angeles

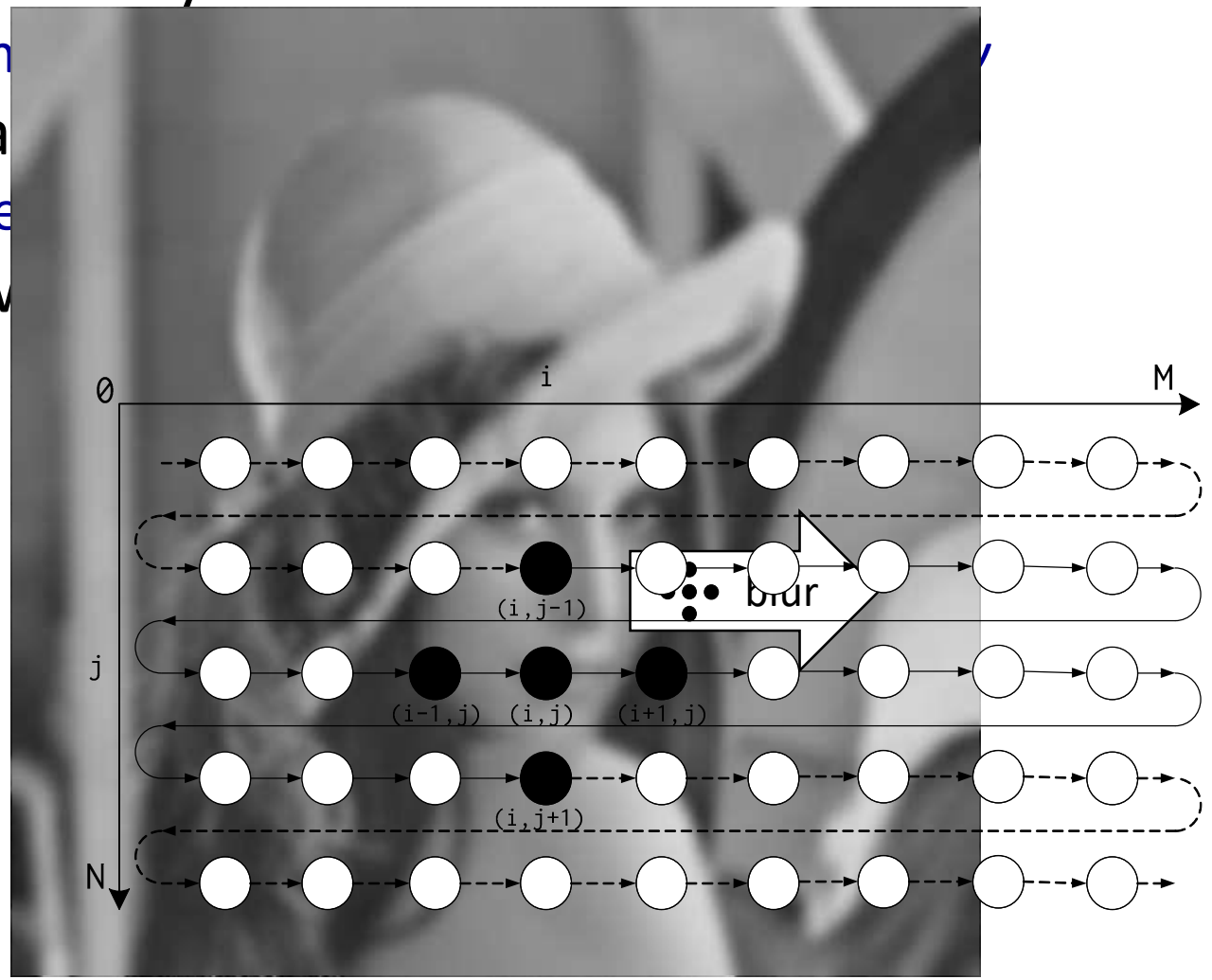
{chiyuze,cong,peng.wei.prc,memoryzpp}@cs.ucla.edu

What is stencil computation?

What is Stencil Computation?

- *A sliding window* applied on an array
  - Compute output according to som
- Extensively used in many area
  - Image processing, solving PDEs, ce
- Example: a 5-point blur filter v

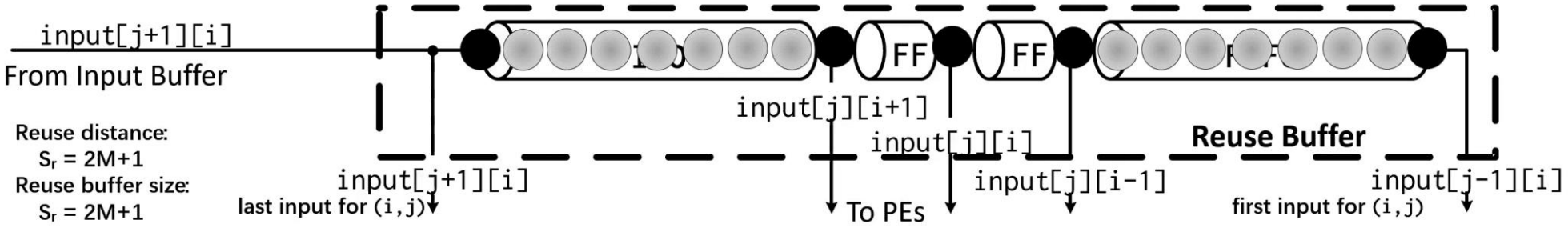
```
void blur(float input [N][M],
          float output[N][M]) {
  for(int j = 1; j < N-1; ++j) {
    for(int i = 1; i < M-1; ++i) {
      output[j][i] = (
        input[j-1][i  ] +
        input[j  ][i-1] +
        input[j  ][i  ] +
        input[j  ][i+1] +
        input[j+1][i  ] ) * 0.2f;
    }
  }
}
```



How do people do stencil computation?

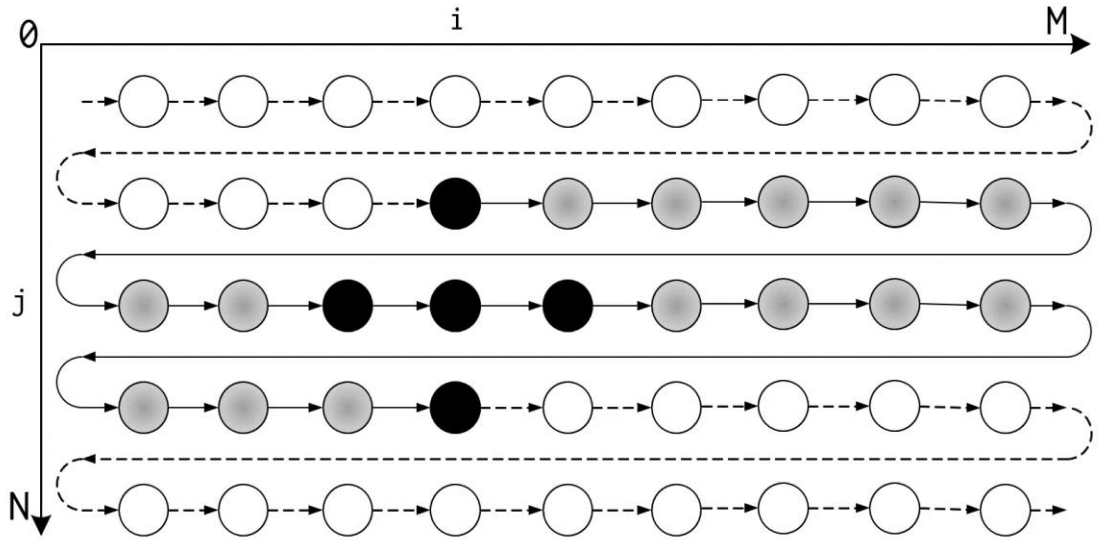
Stencil Optimization #1: Data Reuse

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



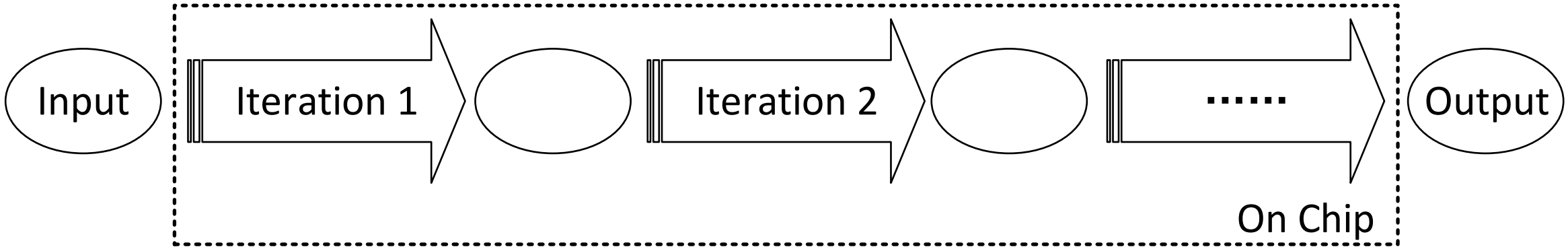
  - Full data reuse, 1 PE
  - Optimal size of reuse buffer
  - Optimal number of memory banks

- But how to parallelize?



Stencil Optimization #2: Temporal Parallelization

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



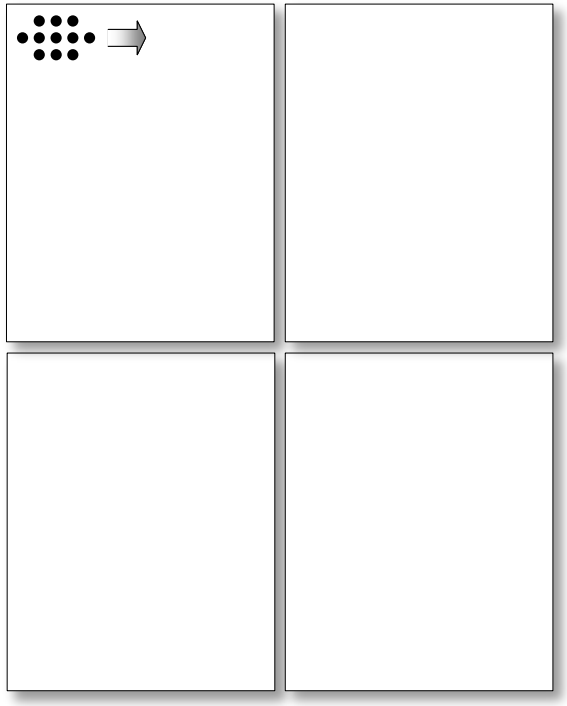
  - Parallelization within each iteration?

ICCAD'16: A Polyhedral Model-Based Framework for Dataflow Implementation on FPGA Devices of Iterative Stencil Loops

Stencil Optimization #3: Spatial Parallelization

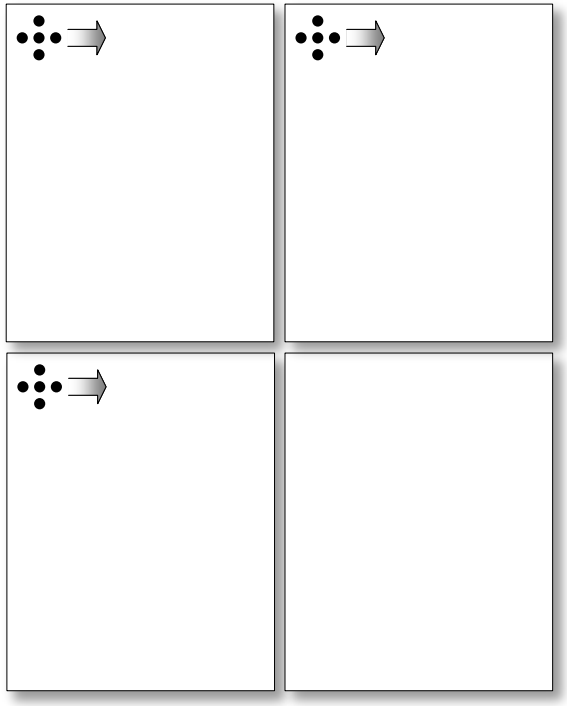
Element-Level Parallelization (FPGA'18)

- 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

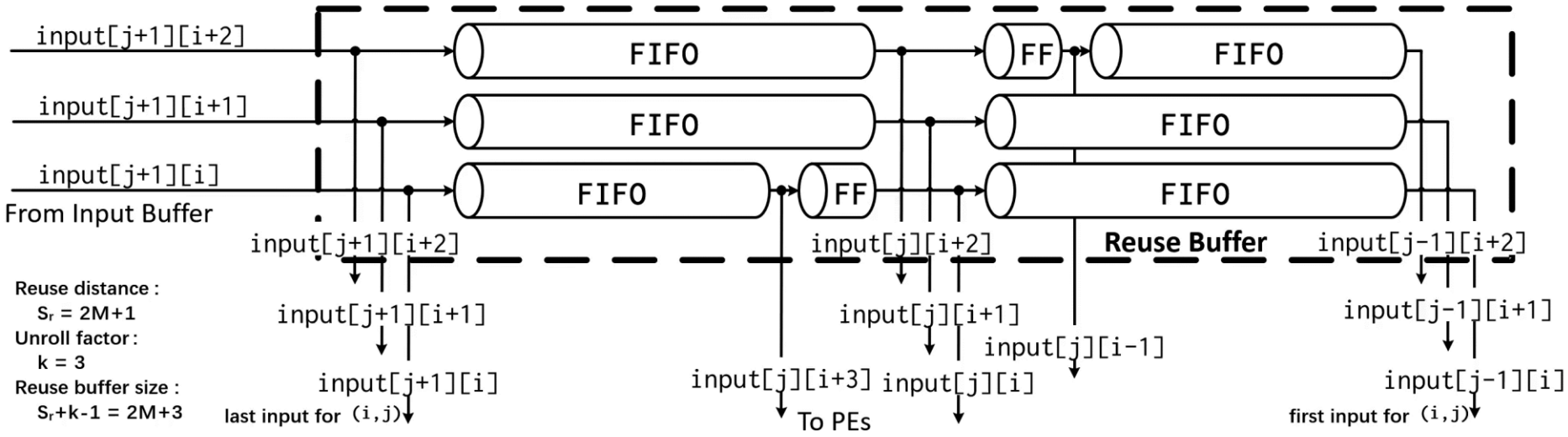
Stencil Optimization: Parallelization

- 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

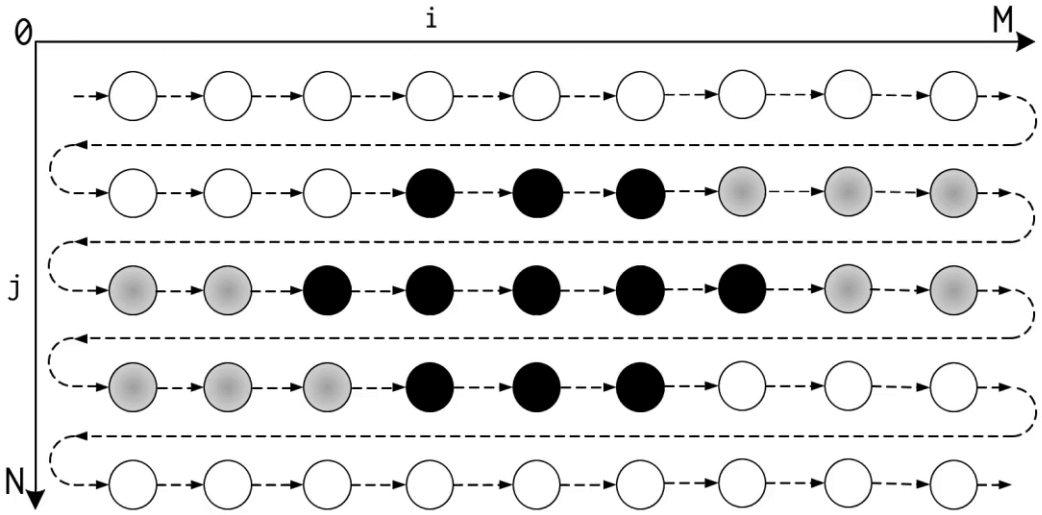
Can we do better?

SODA as a Microarchitecture: Data Reuse

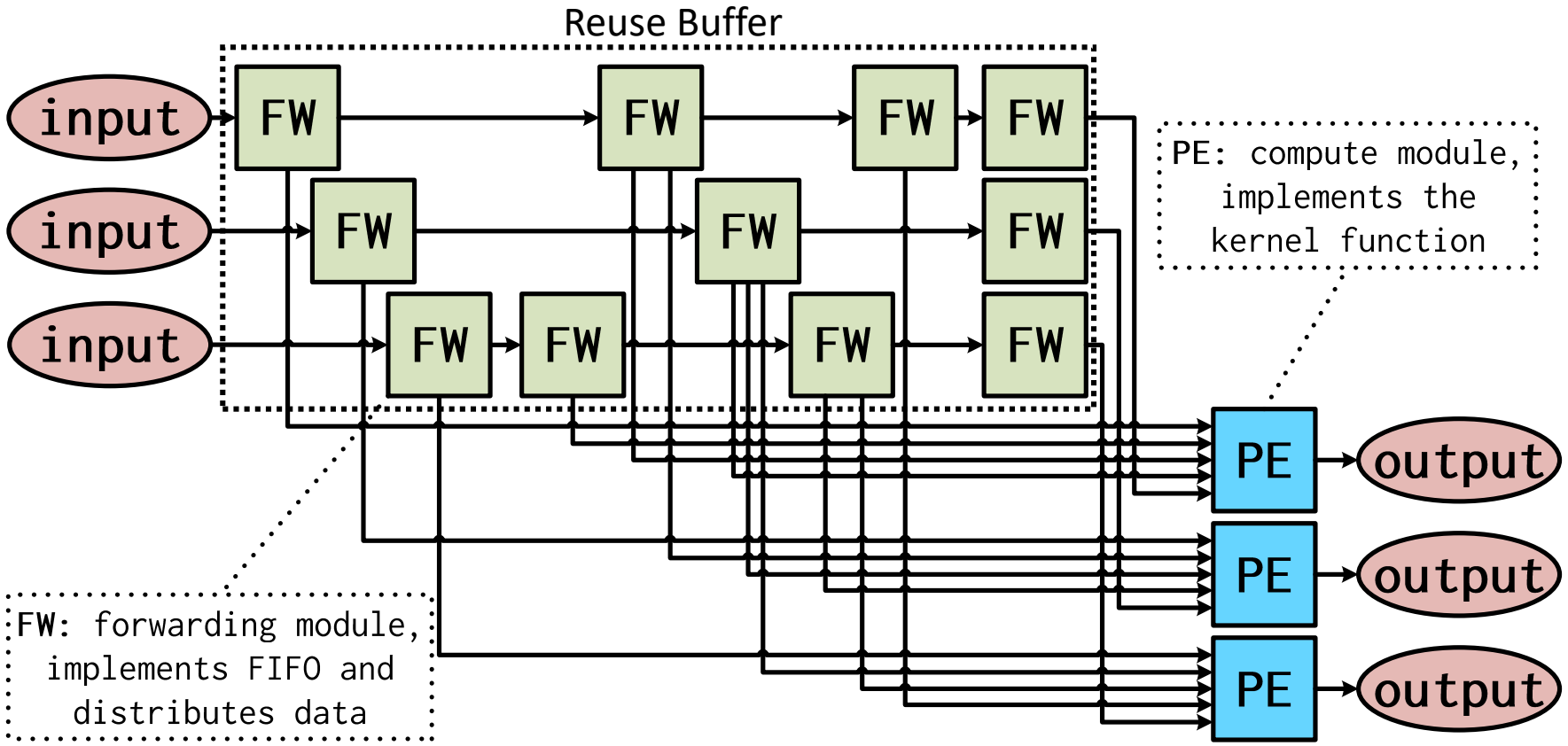
- For $k = 3$ PEs



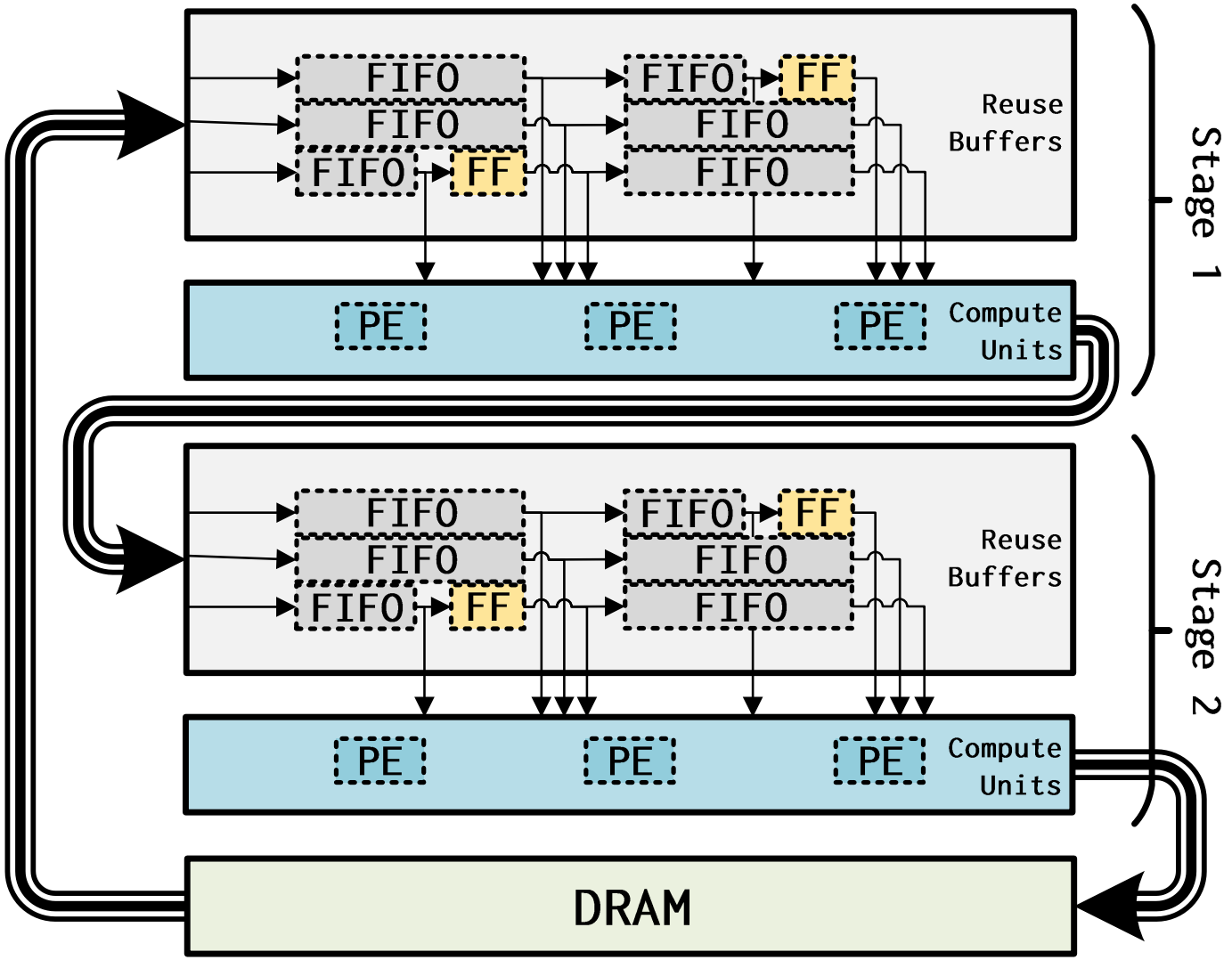
- k PEs only require $S_r + k - 1$ storage
- ***Full*** data reuse
- ***Optimal*** buffer size
- ***Scalable*** when k increases



SODA as a Microarchitecture: Spatial Parallelization



SODA as a Microarchitecture: Temporal Parallelization



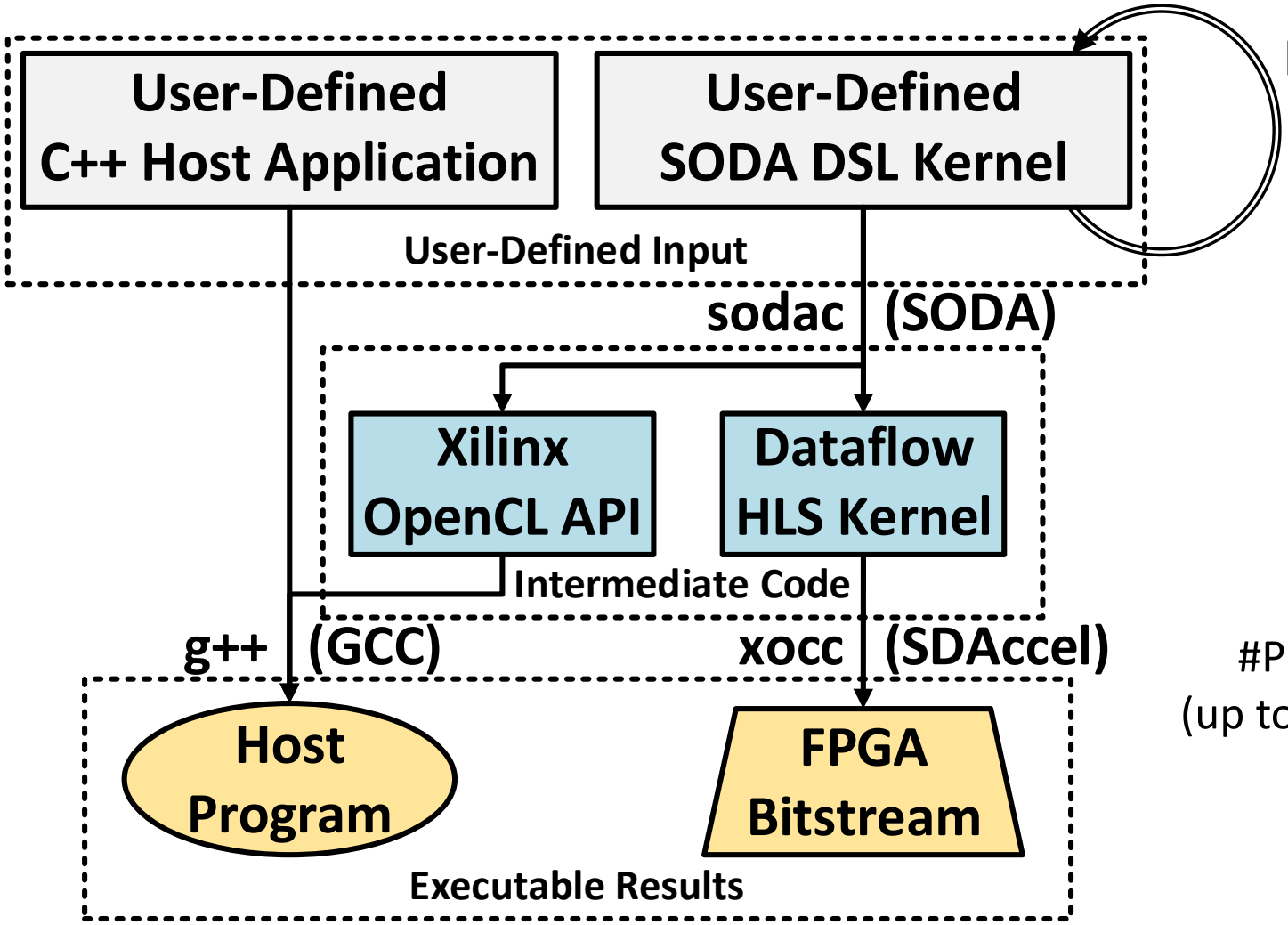
How do you program such a ~~messy~~ fancy architecture?

Stencil Optimization #4: Domain-Specific Language Support

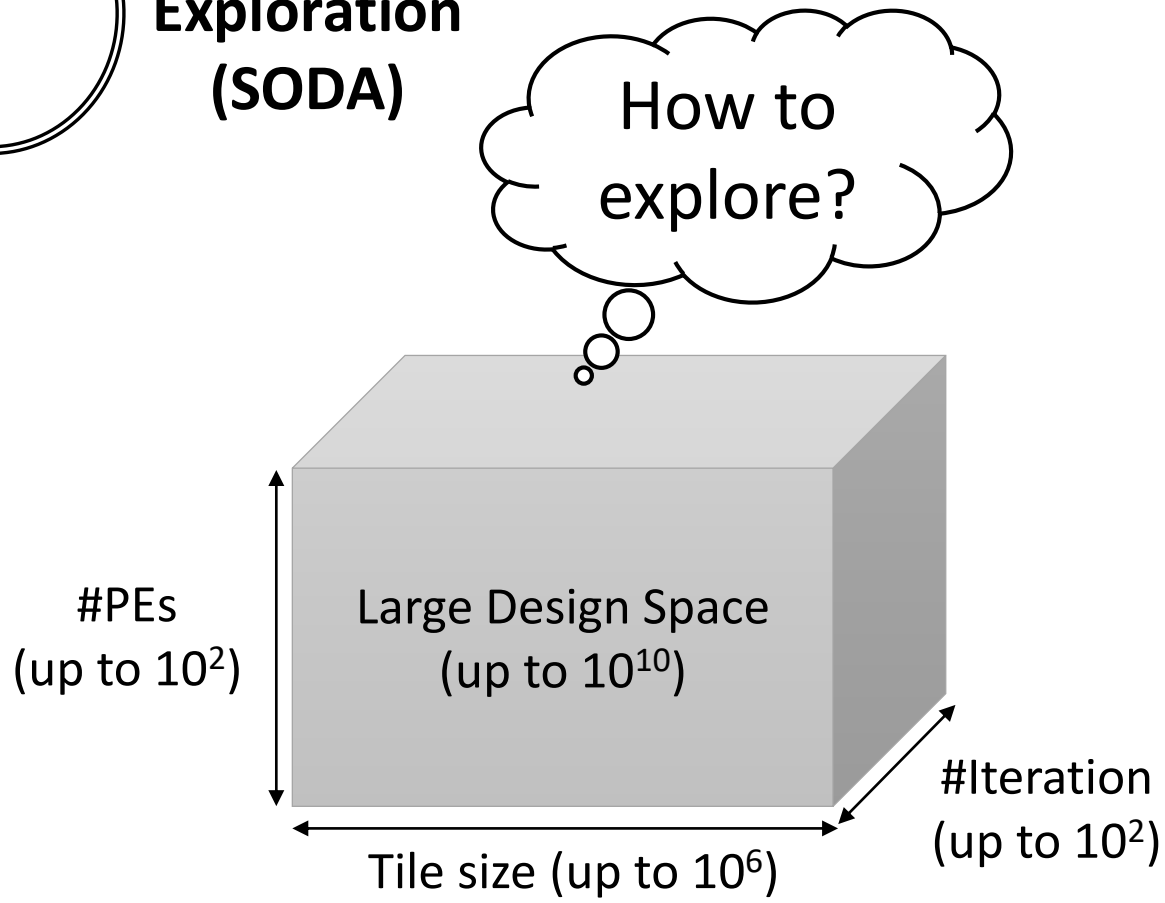
- Complex hardware architecture
- How to program?
  - Template-based
    - DAC'14, ICCAD'16, FPGA'18
  - Domain-specific language (DSL)
    - Darkroom, Halide, Hipacc…
- SODA uses a DSL
  - Flexible
  - Programmable

```
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)) * 0.2f
# 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

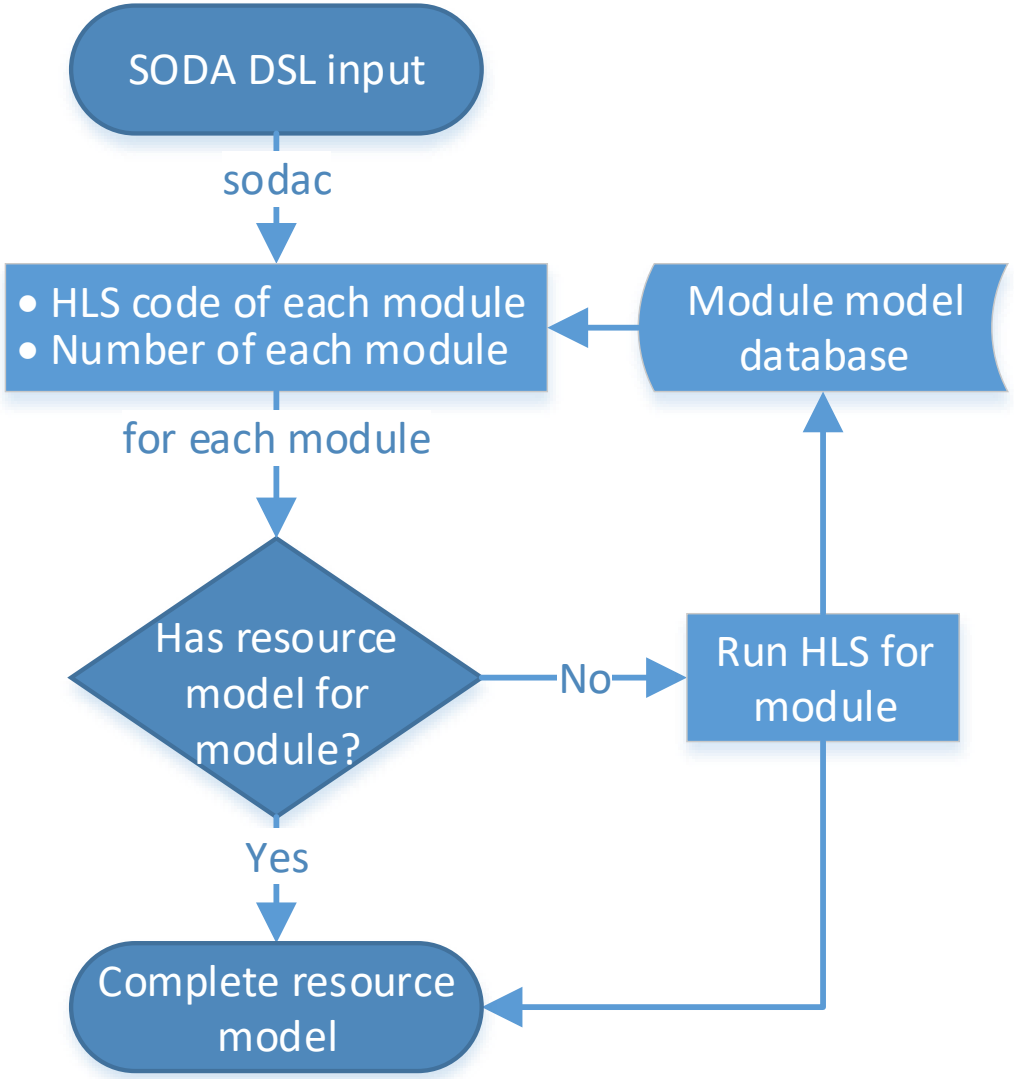


Design-Space Exploration (SODA)

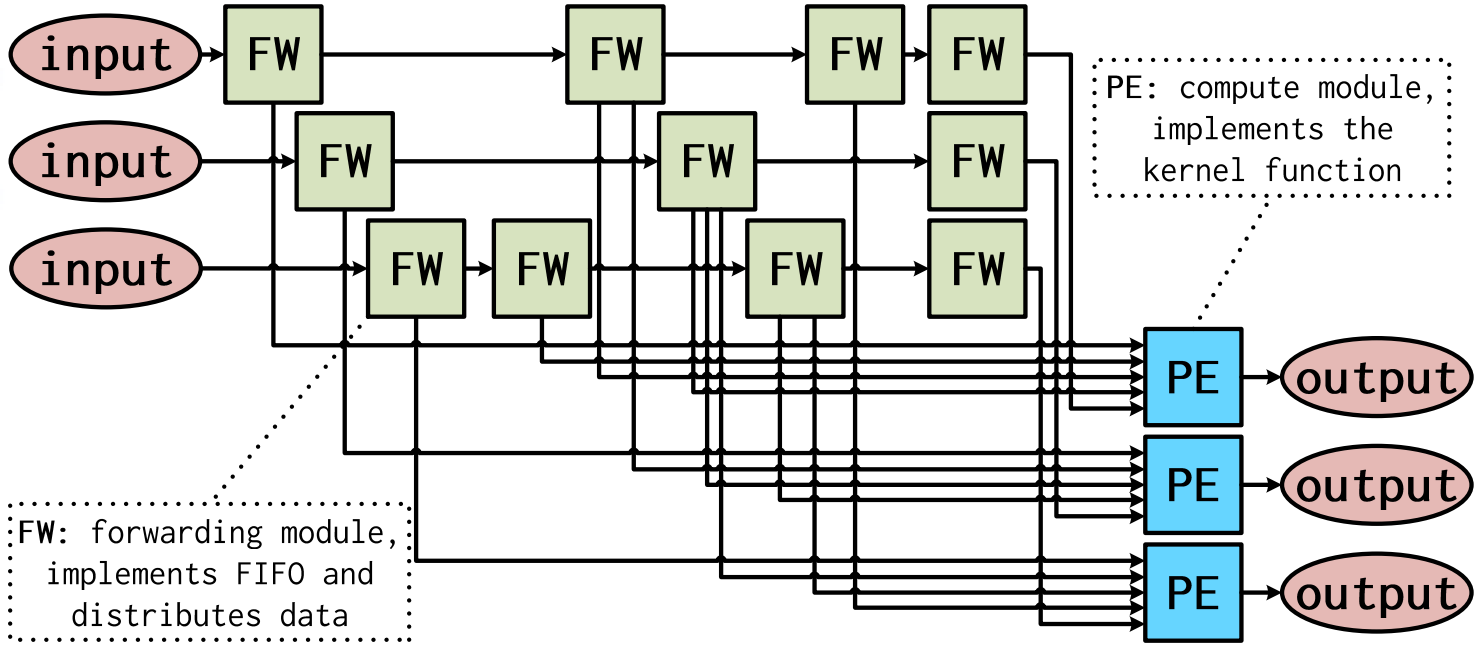


How do you explore such a huge design space?

SODA as an Exploration Engine: Resource Modeling

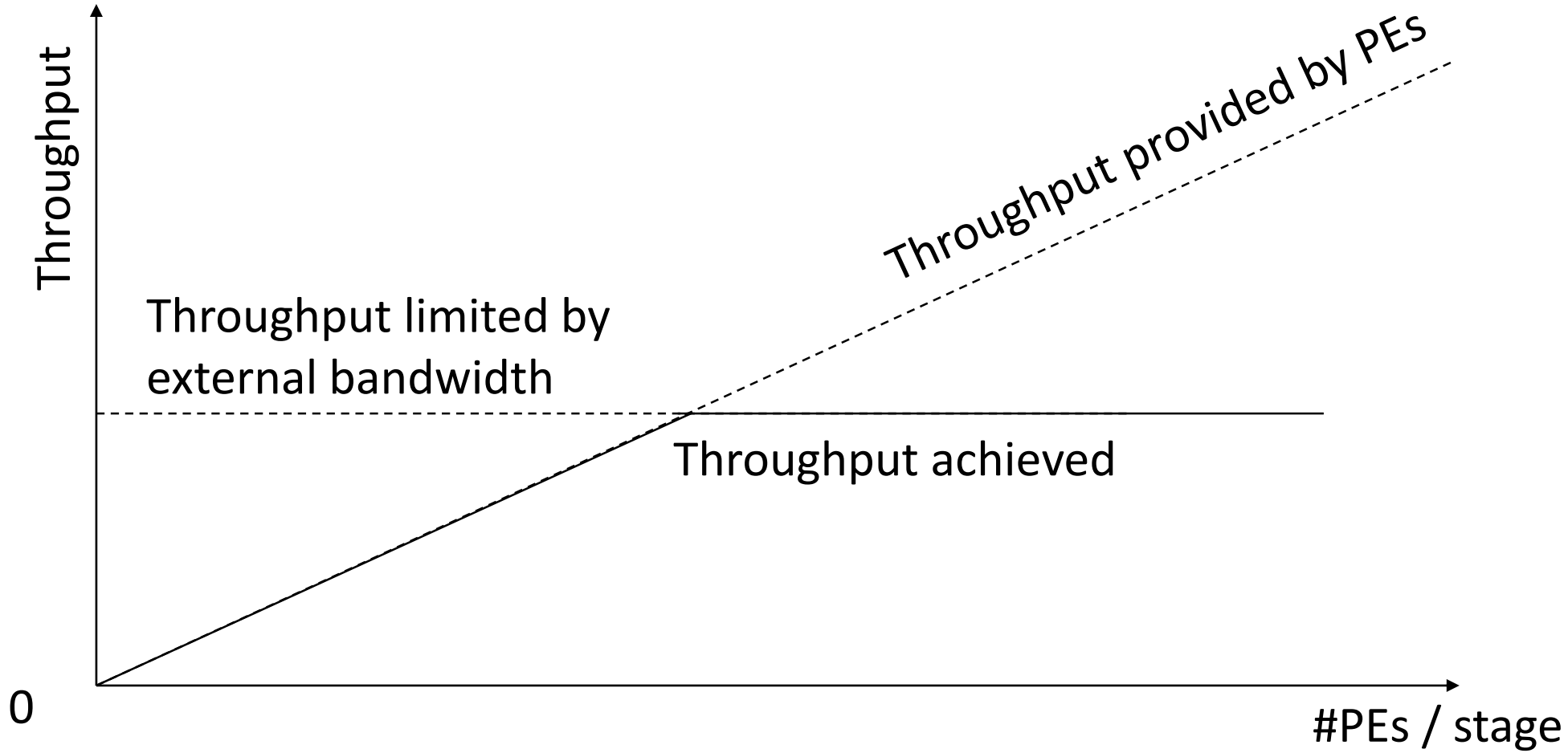


Resource Modeling Flow



Modularized Design Enabling Accurate Architecture-Specific Modeling

SODA as an Exploration Engine: Performance Modeling



Performance Roofline Model

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 $T_0, T_1, \ldots$
  - Bounded by available on-chip storage
  - Searched via branch-and-bound
- Can finish exploration in up to 3 minutes

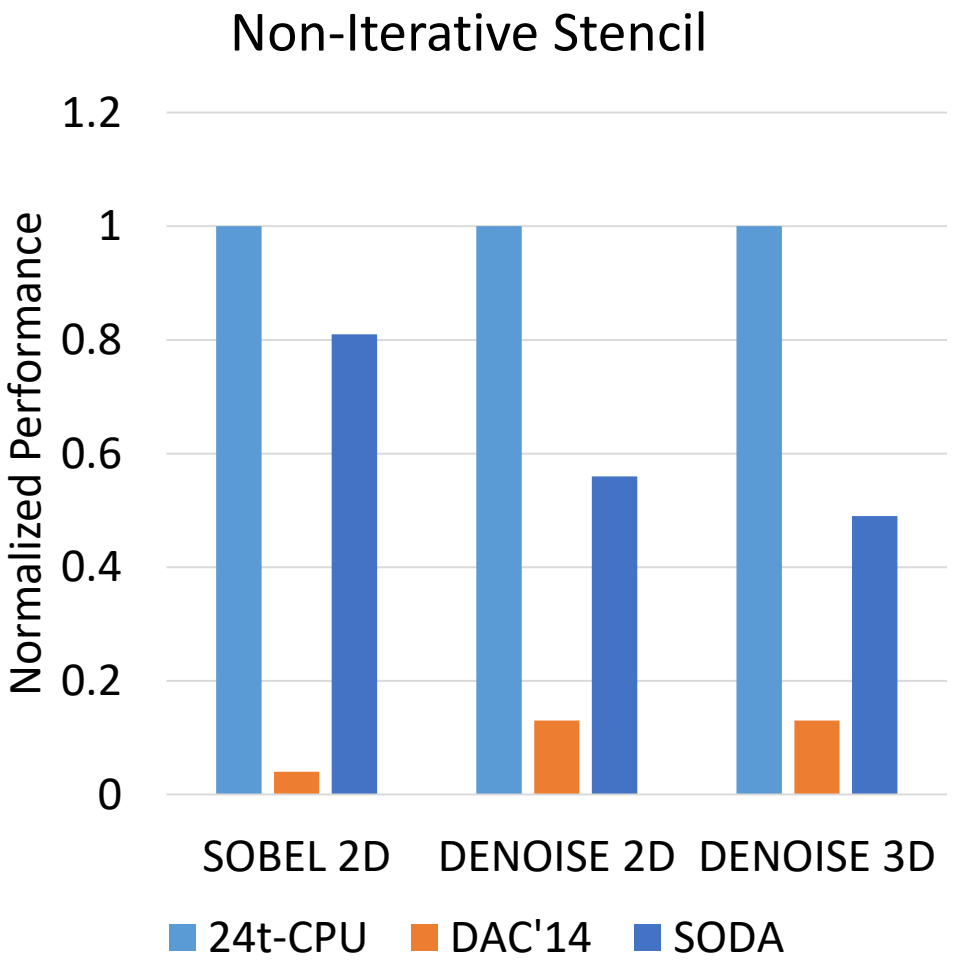
What does your result look like?

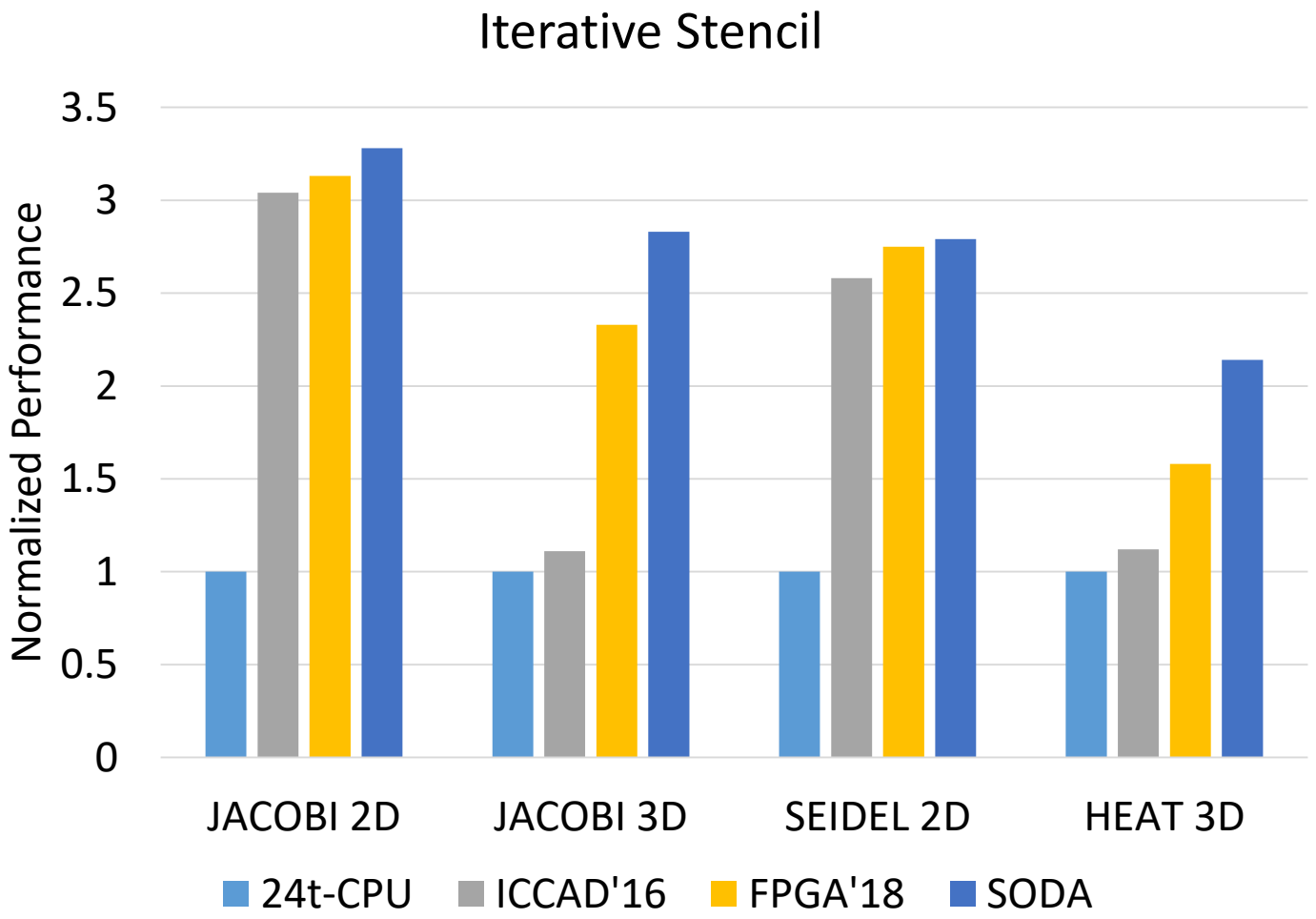
Experimental Results: Model Accuracy

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

Prediction Item	BRAM	DSP	LUT	FF	Throughput
Average Error	1.84%	0%	6.23%	7.58%	4.22%

Experimental Results: Performance Comparison





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.
- DAC'17: A Comprehensive Framework for Synthesizing Stencil Algorithms on FPGAs using OpenCL Model, Wang and Liang
- FPGA'18: Combined Spatial and Temporal Blocking for High-Performance Stencil Computation on FPGAs Using OpenCL, Zohouri et al.

Thank you!

Q&A

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.