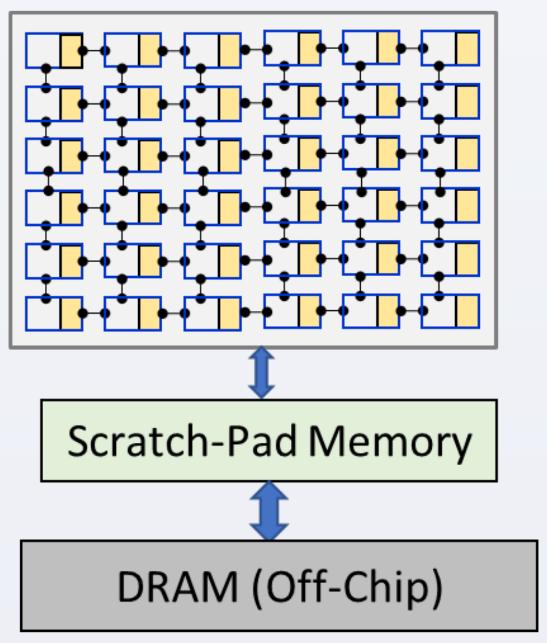
dMazeRunner: Executing Perfectly Nested Loops on Programmable Dataflow Accelerators

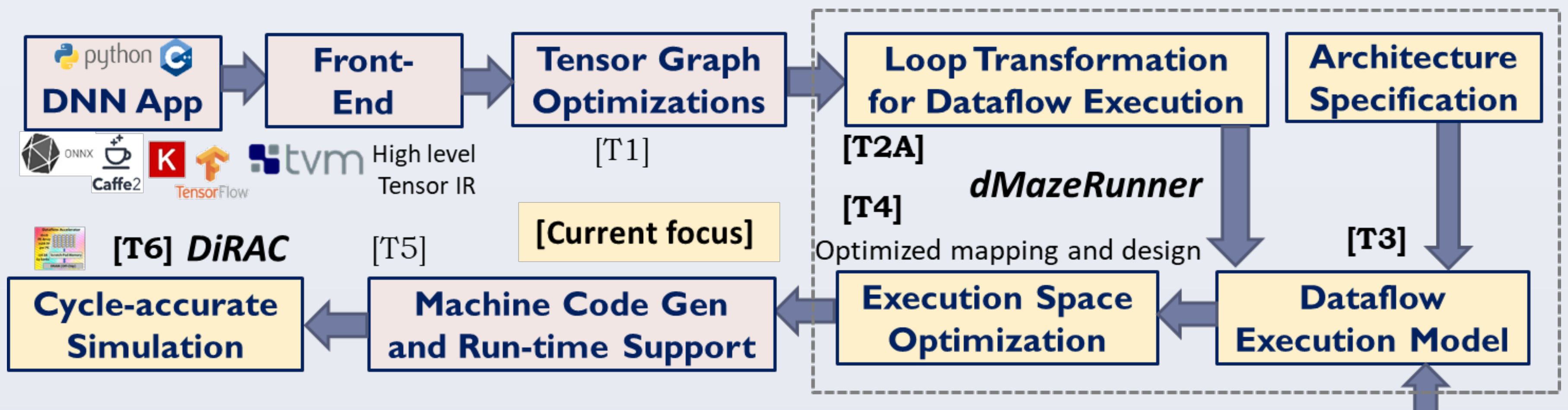
Shail Dave[1], Youngbin Kim[2], Sasikanth Avancha[3], Kyoungwoo Lee[2], Aviral Shrivastava[1]
1. Arizona State University 2. Yonsei University 3. Parallel Computing Lab, Intel Labs

Programmable Dataflow Accelerators



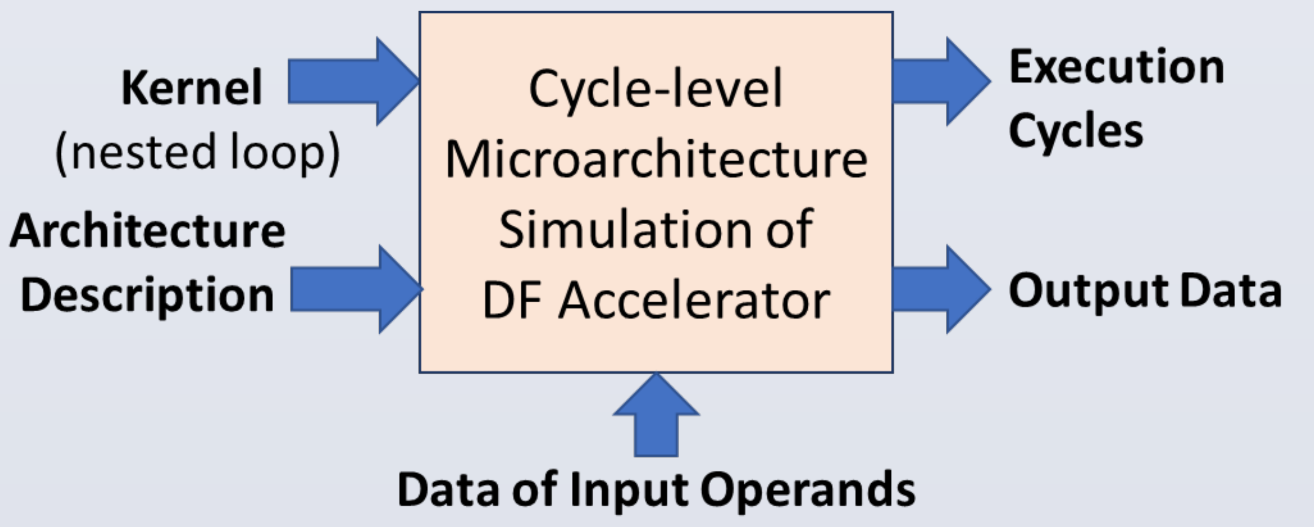
- **Massive array of Processing Elements (PEs)**; each PE has ALU-like functional unit to perform operation every cycle (**simple, programmable**).
- PE's Private + shared memory sustain data reuse.
- **Efficiently accelerate ML and media kernels.**
- Architecture Variations
  - Systolic arrays: TPU (Google), TensorCore (nVIDIA)
  - Spatially programmable architecture: Eyeriss (MIT), SCNN (nVIDIA), AI core (IBM), CSA (Intel)
  - Coarse-grained reconfig array: HyCUBE(NUS), DPU(Wave)

Current Focus in System Stack


python
DNN App
ONNX
Caffe2
TensorFlow
tvm
High level Tensor IR
Front-End
Tensor Graph Optimizations
[T1]
Loop Transformation for Dataflow Execution
Architecture Specification
[T2A]
dMazeRunner
[T4]
Optimized mapping and design
[T3]
[Current focus]
[T6] DiRAC
[T5]
Cycle-accurate Simulation
Machine Code Gen and Run-time Support
Execution Space Optimization
Dataflow Execution Model
[T2B]
Memory Optimizations

Code Release: https://github.com/cmlasu/dMazeRunner
https://github.com/cmlasu/DiRAC

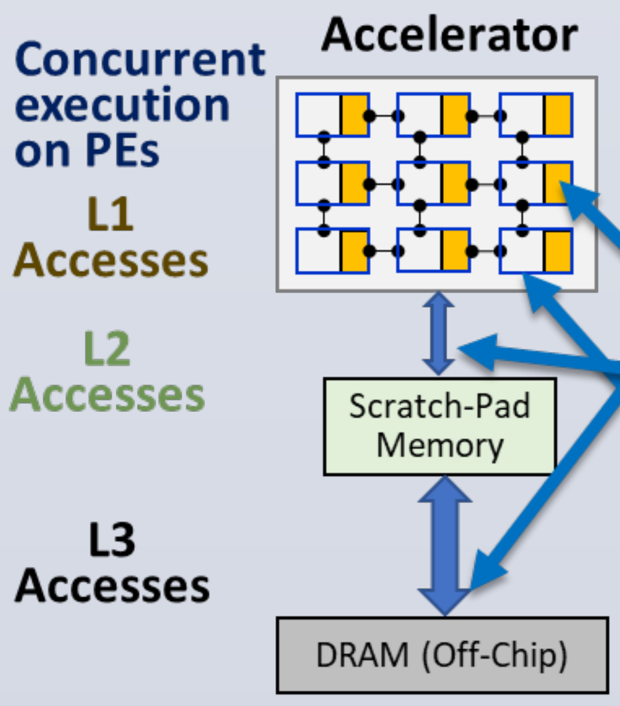
DiRAC: Cycle-level µarch Simulation



Need community infrastructure to learn and simulate DF accelerators.

- DiRAC: `--path test_conv_output_stationary --cmp-golden 1`
  - Architecture variations: PE grid layout, PE pipeline, size/buffering/partitioning of Register Files and Scratchpad memory, interconnect, DMA configuration
  - Simulate nested loops with no conditional statements and with MAC operations.

Loop Orchestration



```
for n_L3 = 1:N_DRAM
 for m_L3 = 1:M_DRAM
  for c_L3 = 1:C_DRAM
   for ox_L3 = 1:Ox_DRAM
    for oy_L3 = 1:Oy_DRAM
     for fx_L3 = 1:Fx_DRAM
      for fy_L3 = 1:Fy_DRAM
      {
       dma( );
       for n_L2 = 1:N_SPM
        for m_L2 = 1:M_SPM
         for c_L2 = 1:C_SPM
          for ox_L2 = 1:Ox_SPM
           for oy_L2 = 1:Oy_SPM
            for fx_L2 = 1:Fx_SPM
             for fy_L2 = 1:Fy_SPM
             {
              communicate_data_NoC( );
              for n_L1 = 1:N_RF
               for m_L1 = 1:M_RF
                for c_L1 = 1:C_RF
                 for ox_L1 = 1:Ox_RF
                  for oy_L1 = 1:Oy_RF
                   for fx_L1 = 1:Fx_RF
                    for fy_L1 = 1:Fy_RF
                    {
                     for n_S = 1:N_SPATIAL
                      for m_S = 1:M_SPATIAL
                       for c_S = 1:C_SPATIAL
                        for ox_S = 1:Ox_SPATIAL
                         for oy_S = 1:Oy_SPATIAL
                          for fx_L3 = 1:Fx_SPATIAL
                           for fy_L3 = 1:Fy_SPATIAL
                              O[][][][] +=
                               I[][][][] *
                               W[][][][];
             }}}
```

Spatial loops
What subset of data each PE executes
Temporal loops
Data Reuse in Memories.

Convolution layer in CNNs

```
for n=1:N                   % batch size
 for m=1:M                  % filters
  for c=1:C                 % channels
   for ox=1:Ox              % rows of ofmap
    for oy=1:Oy             % columns of ofmap
     for fx=1:Fx            % filter height
      for fy=1:Fy           % filter width
         O[n][m][ox][oy] +=
           I[n][c][ox+fx-1][oy+fy-1]*
           W[m][c][fx][fy];
```

Vast "Execution Method" Space

Problem of exploring "execution methods" becomes problem of exploring all the possibilities of tiling and ordering in 28-dimensional loop.

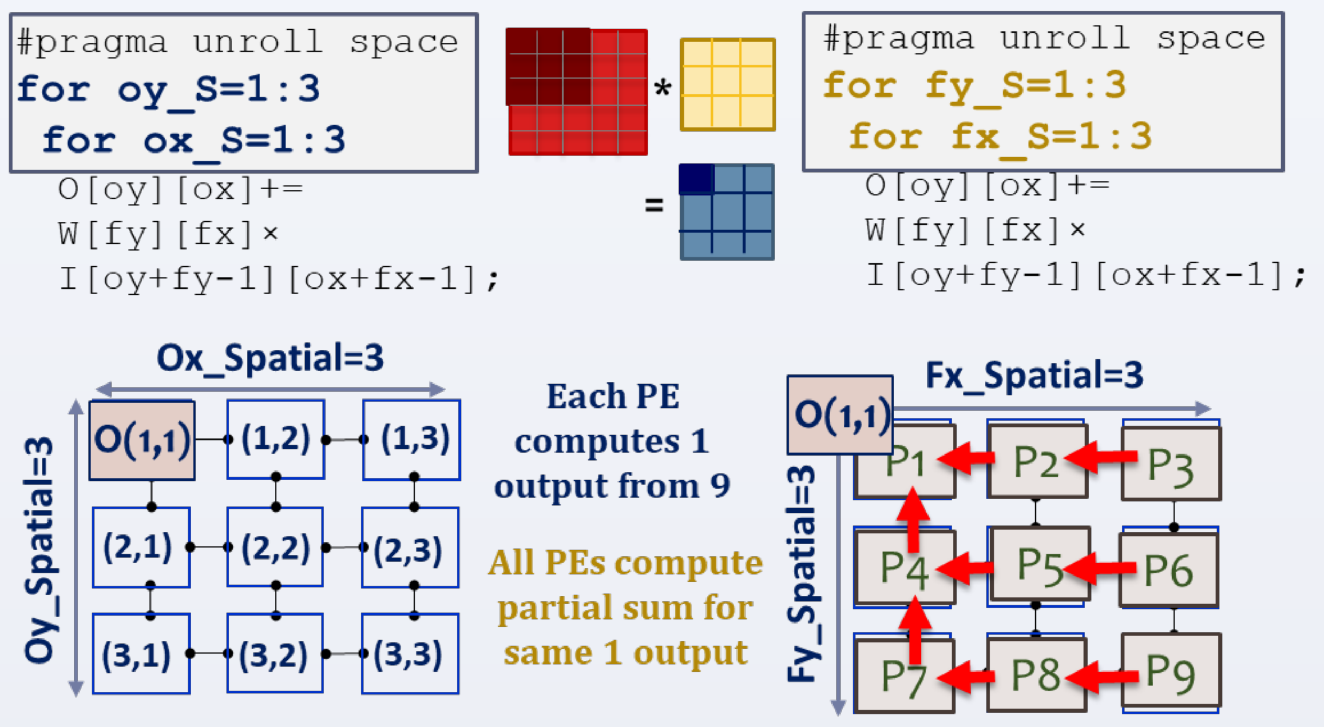
SpatioTemporal Execution of Loops

Output Stationary Dataflow

```
#pragma unroll space
for oy_S=1:3
 for ox_S=1:3
   O[oy][ox]+=
   W[fy][fx]×
   I[oy+fy-1][ox+fx-1];
```

Weight Stationary Dataflow

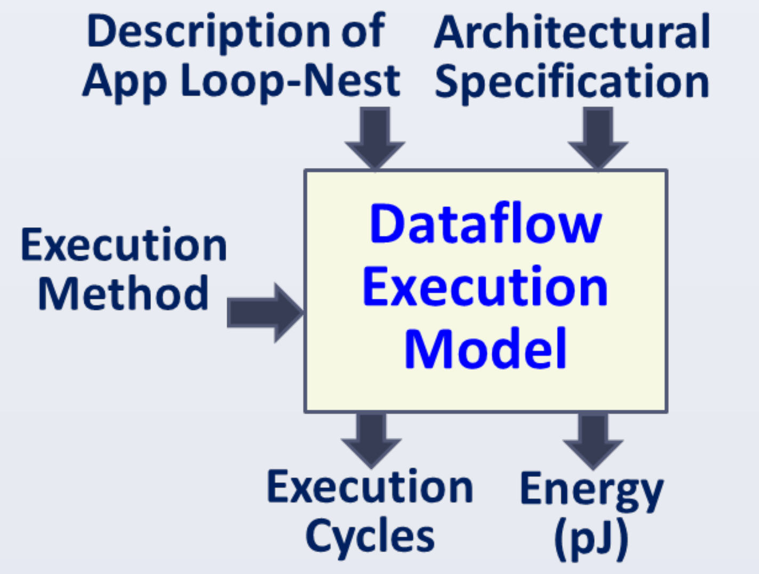
```
#pragma unroll space
for fy_S=1:3
 for fx_S=1:3
   O[oy][ox]+=
   W[fy][fx]×
   I[oy+fy-1][ox+fx-1];
```



Each PE computes 1 output from 9

All PEs compute partial sum for same 1 output

Analytical Model of Dataflow Execution



- ✓ arbitrary perfectly nested loops.
- ✓ miss penalty and stall cycles
- ✓ inter-PE communication.
- ✓ temporal/spatial data reuse.
- ✓ supports ML libraries MXNet, Keras.

Validation
- Eyeriss chip: ~11% difference in execution cycles (AlexNet, batch: 4)
- DNN Optimizer of X. Yang et al.: ~4.2% difference in Energy (ResNet Conv5_2)

Drastic Pruning of Search Space

```
for n = 1:N=2
 for m = 1:M=8
  for c = 1:C=4
   for fy = 1:Fy=3
   { comm_data();
     O[n][m]+=
       I[n][c][fy]×
       W[m][c][fy]  }
```

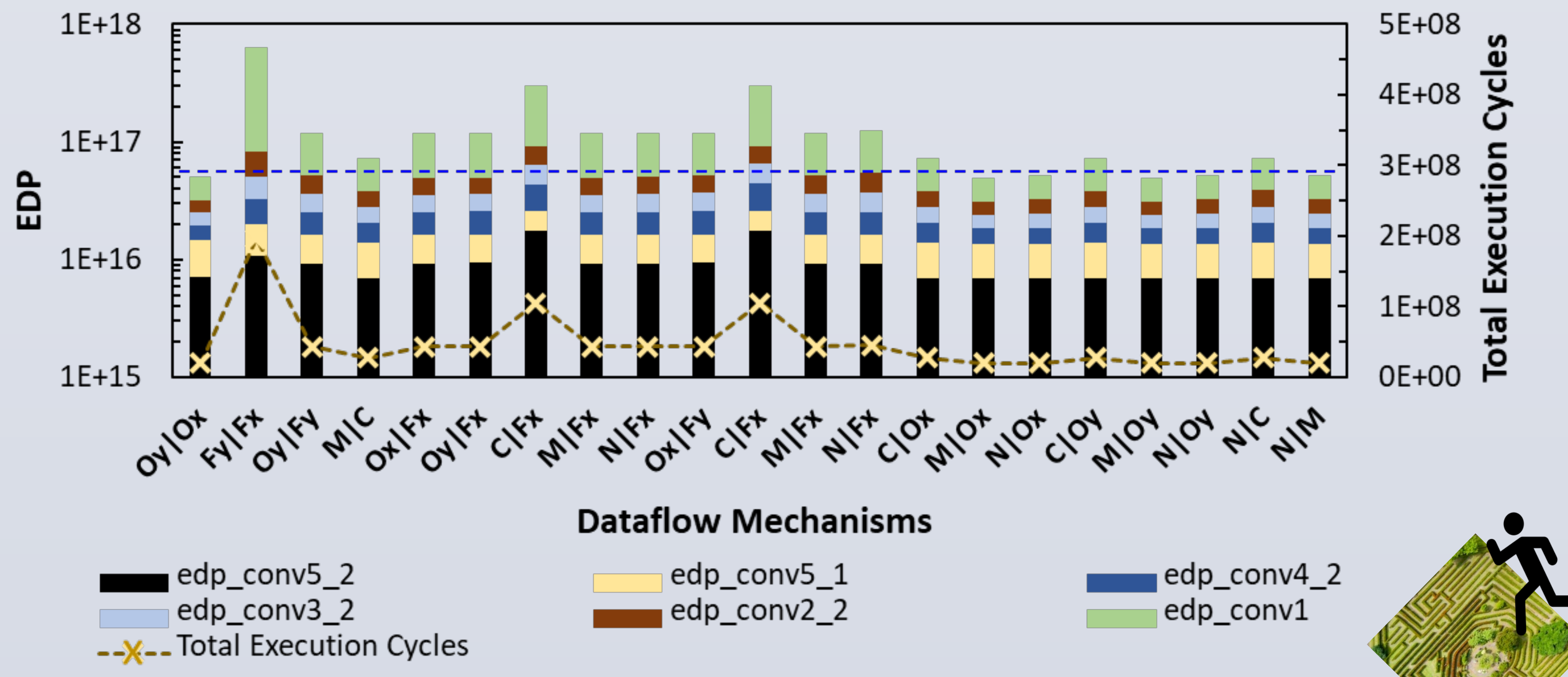
Unique Reuse Factors for Data Operands

Loop Order	I	W	O
{..., n}	–	N = 2	–
{..., m}	M = 8	–	–
{..., m, c}	–	–	C = 4
{..., m, fy}	–	–	Fy = 3
{..., fy, c}	–	–	C×Fy = 4×3

5 schedules vs. 4! = 24 orderings

Results: Optimized Mappings Across DFs and Layers

Executing ResNet layers on 256-PE, 512B RF, 128kB SPM accelerator



Adaptable Mappings = Better Results

Optimize various factors
- ✓ Very high resource utilization
- ✓ Reuse of multiple operands
- ✓ Minimize DRAM accesses.
- ✓ Efficiently interleave compute with communication latency

dMazeRunner Features

- Non-expert programmers can explore space in seconds.
- Domain experts can perform directed search.
- Explore efficient designs for models/layers through DSE

```
python run_optimizer.py
--frontend mxnet --model
resnet18_v1 -auto-optimize
```

Acknowledgements

This work was partially supported by funding from NSF grant CCF 1723476 – NSF/Intel joint research center for Computer Assisted Programming for Heterogeneous Architectures (CAPA) and from the grants NRF-2015M3C4A7065522 and 2014-3-00035 funded by MSIT.