dMAzeRunner: Accelerating loop nests on dataflow accelerators

Aviral Shrivastava

Joint work with:

Shail Dave (ASU)
Sasikanth Avancha (Intel PCL)
Youngbin Kim and Kyoungwoo Lee (Yonsei)

Arizona State University

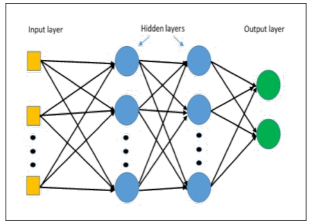
10/14/19

CML

Must-Accelerate Applications in ML Era

Widely Used ML Models

Multi Layer Perceptrons

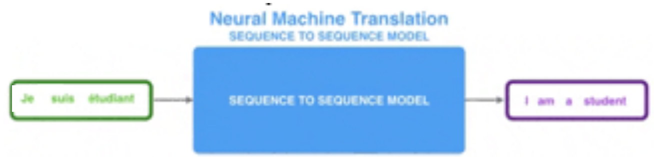


http://yann.lecun.com/exdb/lenet/

Convolution Neural Networks

http://vision03.csail.mit.edu/cnn_art/index.html
https://pjreddie.com/darknet/

Sequence Models


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL
Je suis étudiant
SEQUENCE TO SEQUENCE MODEL
I am a student

http://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
https://deeplearning.mit.edu/

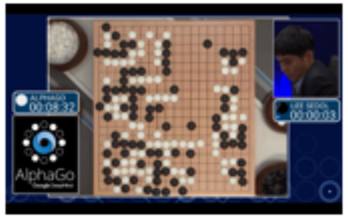
Graph Neural Networks

Points of Interest

Delaunay Triangulation

YOW! Data 2018 Conference.
https://www.youtube.com/watch?v=lDRb3CjESmM

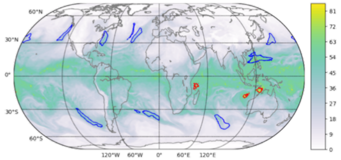
Reinforcement Learning



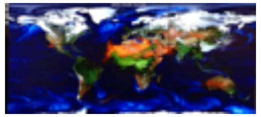
AlphaGo.
https://www.nature.com
/articles/nature24270

Popular Applications

- **Object Classification/Detection**
- **Media Processing/Generation**
- **Large-Scale Scientific Computing**



Tropical Cyclon Detection
https://insidehpc.com/2019/02/gordon-bell-prize-highlights-the-impact-of-ai/



https://giphy.com

- **Designing Software 2.0**

  Google shrinks language translation code from 500k LoC to 500

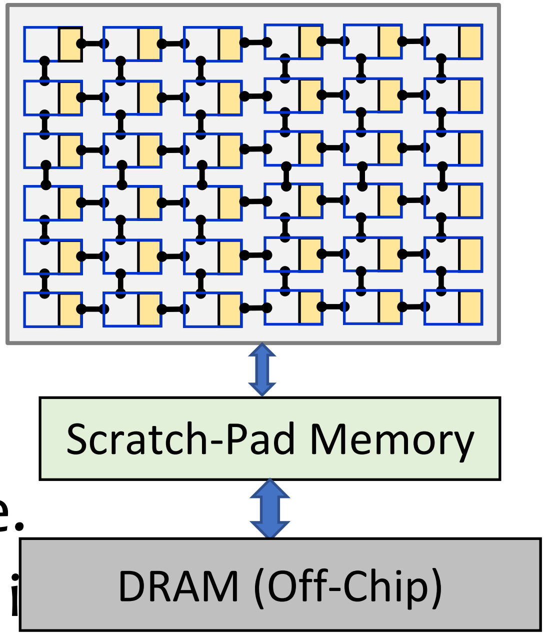
  https://jack-clark.net/2017/10/09/import-ai-63-google-shrinks-language-translation-code-from-500000-to-500-lines-with-ai-only-25-of-surveyed-people-believe-automationbetter-jobs/
  Kunle Olukotun, NeurIPS 2018 Invited talk.

- **and more …**

Dataflow Accelerators: Promising Solution

Known variations include - Systolic arrays,
- Spatially programmable architecture,
- Coarse-Grained Reconfigurable Arrays

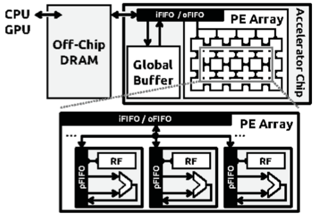
- Massive arrays of processing elements.
  - Simple: absence of complex OoO pipeline and decoding.
  - Programmable: accommodate executing all operations within loop.
- Private and shared memory for PEs sustain data reuse.
- PEs can be busy performing computations while data i being communicated from lower memories.
  - Taken care by effective data management i.e., software prefetching, data distribution, data allocation.



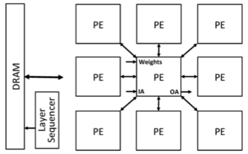
Scratch-Pad Memory

DRAM (Off-Chip)

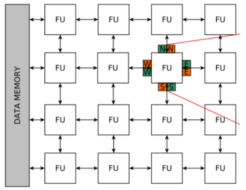
Eyeriss, MIT



SCNN, nVIDIA

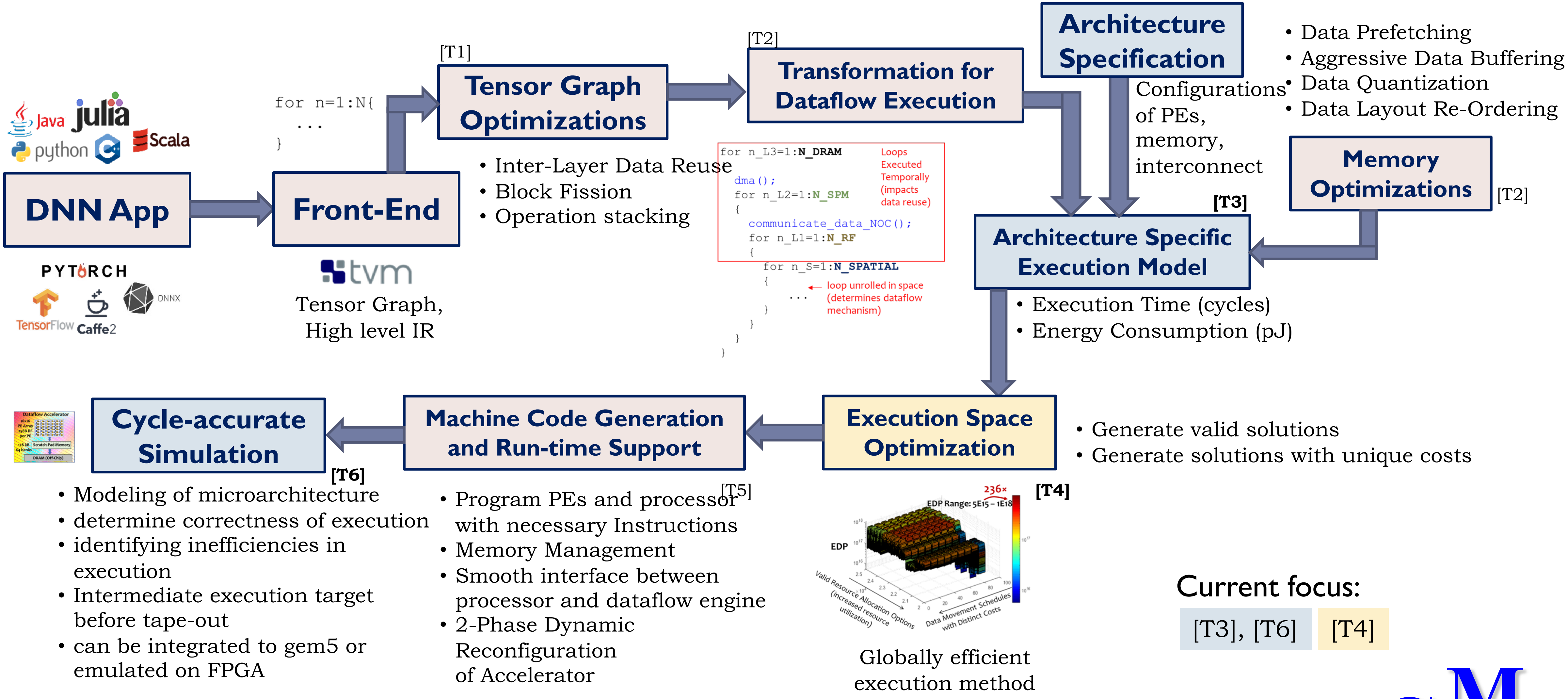


Hycube, NUS



[1] Norman Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In ISCA 2017.
[2] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep cnns. In JSSC 2016
[3] Dataflow Processing Unit from Wave Computing. In HOTCHIPS 2017.
[4] M. Thottethodi and T. N. Vijaykumar. Why the GPGPU is Less Efficient than the TPU for DNNs. ACM SIGARCH Blog, Jan 2019. (online)
[5] Bruce Fleischer et al., A Scalable Multi-TeraOPS Core for AI Training and Inference. In VLSI 2018.
[6] Manupa Karunaratne et al. Hycube: A cgra with reconfigurable single-cycle multi-hop interconnect. In DAC 2017.

Our current focus in the system stack



DNN App

Front-End

```
for n=1:N{
   ...
}
```

Tensor Graph,
High level IR

[T1]

Tensor Graph Optimizations

- Inter-Layer Data Reuse
- Block Fission
- Operation stacking

[T2]

Transformation for Dataflow Execution

```
for n_L3=1:N_DRAM                    Loops
                                     Executed
  dma();                             Temporally
  for n_L2=1:N_SPM                   (impacts
  {                                  data reuse)
    communicate_data_NOC();
    for n_L1=1:N_RF
    {
      for n_S=1:N_SPATIAL
      {
            ...   ← loop unrolled in space
      }             (determines dataflow
    }               mechanism)
  }
}
```

Architecture Specification

Configurations of PEs, memory, interconnect

- Data Prefetching
- Aggressive Data Buffering
- Data Quantization
- Data Layout Re-Ordering

Memory Optimizations [T2]

[T3]

Architecture Specific Execution Model

- Execution Time (cycles)
- Energy Consumption (pJ)

Execution Space Optimization

- Generate valid solutions
- Generate solutions with unique costs

[T4]

Globally efficient execution method

Machine Code Generation and Run-time Support

[T5]

- Program PEs and processor with necessary Instructions
- Memory Management
- Smooth interface between processor and dataflow engine
- 2-Phase Dynamic Reconfiguration of Accelerator

Cycle-accurate Simulation

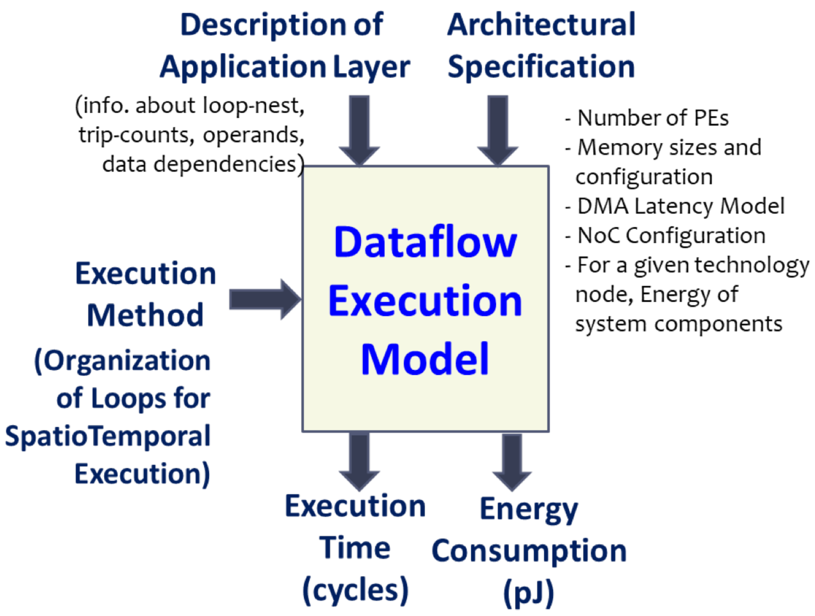
[T6]

- Modeling of microarchitecture
- determine correctness of execution
- identifying inefficiencies in execution
- Intermediate execution target before tape-out
- can be integrated to gem5 or emulated on FPGA

Current focus:

[T3], [T6] [T4]

Execution Modeling of Dataflow Accelerators



Description of Application Layer

(info. about loop-nest, trip-counts, operands, data dependencies)

Architectural Specification

- Number of PEs
- Memory sizes and configuration
- DMA Latency Model
- NoC Configuration
- For a given technology node, Energy of system components

Execution Method (Organization of Loops for SpatioTemporal Execution)

Dataflow Execution Model

Execution Time (cycles)

Energy Consumption (pJ)

Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, Aviral Shrivastava, dMazeRunner: Executing Perfectly Nested Loops on Dataflow Accelerators [CODES+ISSS, TECS 2019].

Features with detailed modeling of

✓ Analyze **arbitrary perfectly nested loops**.

✓ **miss penalty and stall cycles** (PE execution, managing PE/shared memory).

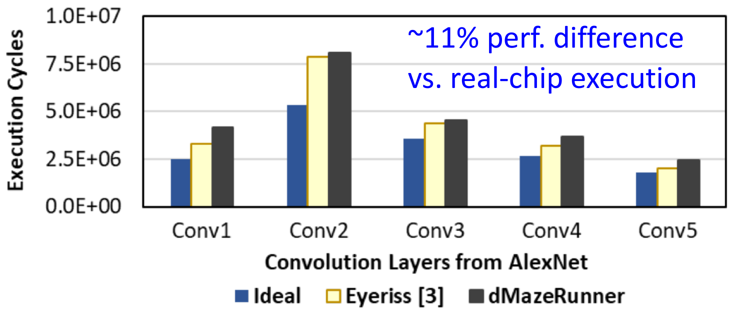
✓ **inter-PE communication.**

✓ **temporal/spatial data reuse.**

✓ Integrated **support common ML libraries** MXNet/Keras/Tensorflow/... (thanks TVM! – leveraging front-end)

Step-wise equations and analysis in the paper

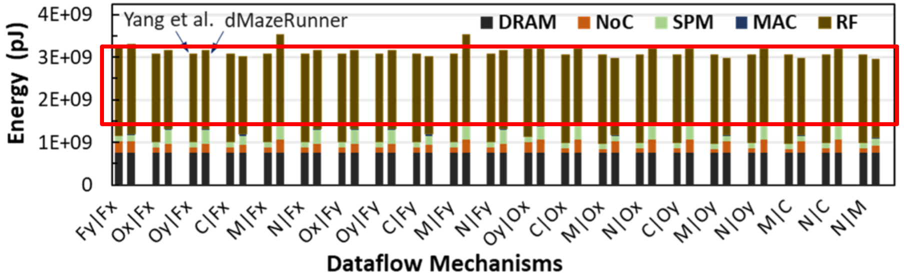
Validation of Dataflow Model against Eyeriss Chip



Chen, Yu-Hsin et al. "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks." [JSSC '17]

Validation against DNN Optimizer of Yang et al.

Yang, Xuan, M. Gao, J. Pu, A. Nayak, Q. Liu, S. Bell, J. Setter, K. Cao, H. Ha, Christos Kozyrakis, and Mark Horowitz. "DNN Dataflow Choice Is Overrated." [arXiv '18]



- Energy estimate differs by ~4.2% for variety of execution methods
- For efficient mappings, major energy spent in RF accesses

Web page: aviral.lab.asu.edu

DiRAC: Microarch and Cycle-accurate Simulator

Dynamically reconfigurable dataflow accelerator architecture template.

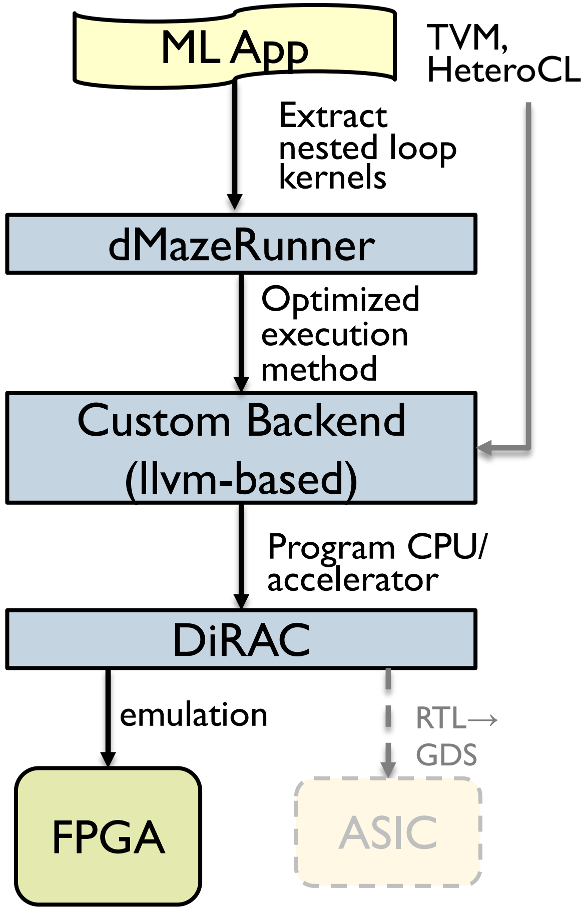
- **(a)synchronous execution of pipelined PEs,** double-buffered larger RFs
- **Programmable multicast network** for arbitrary dataflow
- 2D-mesh interconnect for fast inter-PE communications
- **Multi-bank**, conflict-free, software-directed **scratchpad management**
- **Architectural template serves as a kick-starter baseline.**
  Extend to support various interconnect/PEs/memory architecture

Cycle-accurate simulation of accelerator system.

- Explore FSM/ISA variations for PEs and controller, sensitivity analysis,…
- **Work-in-progress.** Release planned in Q4 end (Dec '19).
- **Next step: FPGA emulation for functional testing + rapid prototyping.**
- Develop + Integrate area/power model for comprehensive design exploration.

Outcomes

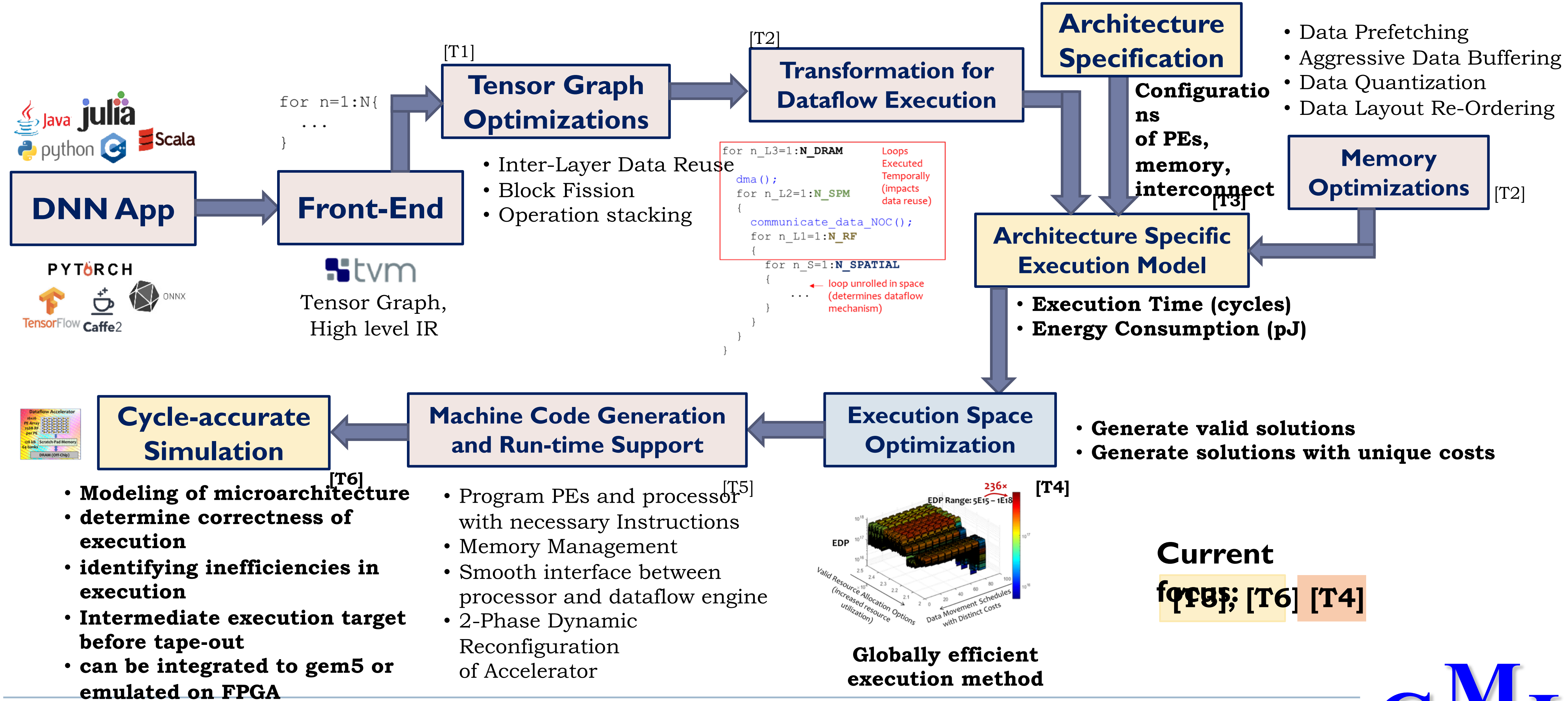
- **Validation of** optimizations achieved through **analytical modeling.**
- **Easy tool for prototyping** domain-specific accelerator architecture.
- **Educational/Training:** Tool for teaching and hands-on with ML accelerators.



Plan for Integration of DiRAC with dMazeRunner.

Web page: aviral.lab.asu.edu

Our current focus in the system stack



DNN App

Front-End

```
for n=1:N{
   ...
}
```

Tensor Graph, High level IR

[T1]

Tensor Graph Optimizations

- Inter-Layer Data Reuse
- Block Fission
- Operation stacking

[T2]

Transformation for Dataflow Execution

```
for n_L3=1:N_DRAM
{
  dma();
  for n_L2=1:N_SPM
  {
    communicate_data_NOC();
    for n_L1=1:N_RF
    {
      for n_S=1:N_SPATIAL
      {
          ...
      }
    }
  }
}
```

Loops Executed Temporally (impacts data reuse)

loop unrolled in space (determines dataflow mechanism)

Architecture Specification

Configurations of PEs, memory, interconnect

[T3]

- Data Prefetching
- Aggressive Data Buffering
- Data Quantization
- Data Layout Re-Ordering

Memory Optimizations

[T2]

Architecture Specific Execution Model

- **Execution Time (cycles)**
- **Energy Consumption (pJ)**

Execution Space Optimization

- **Generate valid solutions**
- **Generate solutions with unique costs**

[T4]

236×
EDP Range: 5E15 – 1E18
EDP
10^{18}
10^{17}
10^{16}
2.5
2.4
2.3
2.2
2.1
2
$\times 10^4$
Valid Resource Allocation Options (increased resource utilization)
0
20
40
60
80
100
Data Movement Schedules with Distinct Costs

Globally efficient execution method

Machine Code Generation and Run-time Support

[T5]

- Program PEs and processor with necessary Instructions
- Memory Management
- Smooth interface between processor and dataflow engine
- 2-Phase Dynamic Reconfiguration of Accelerator

Cycle-accurate Simulation

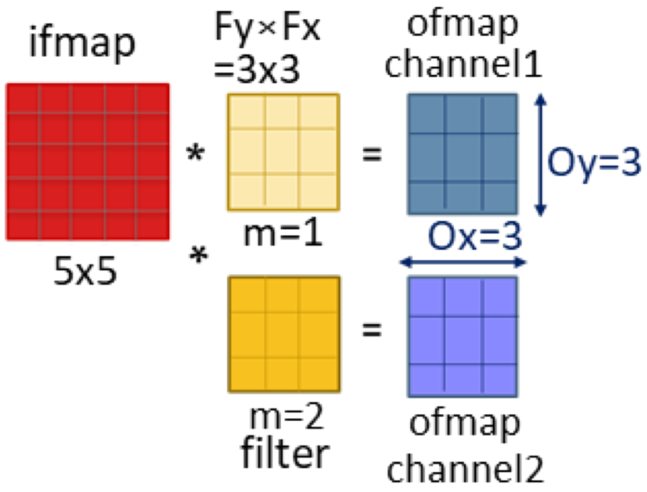
[T6]

- **Modeling of microarchitecture**
- **determine correctness of execution**
- **identifying inefficiencies in execution**
- **Intermediate execution target before tape-out**
- **can be integrated to gem5 or emulated on FPGA**

Current focus: [T5]; [T6] [T4]

Spatio-Temporal Execution on dataflow accelerators

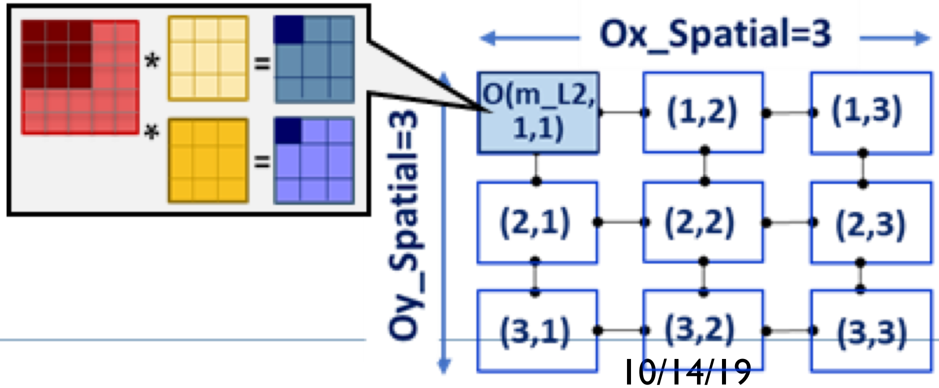
```
for m=1:2
 for oy=1:3
  for ox=1:3
   for fy=1:3
    for fx=1:3
     O[m][oy][ox]+=
      I[oy+fy-1][ox+fx-1]
      × W[m][fy][fx];
```



```
% access DRAM once (no L3 loops)
dma() % prefetch data in 256B SPM
for m_L2=1:2
 for fy_L2=1:3
   access_SPM_and_comm_NoC();
    for fx_L1=1:3
     for oy_S=1:3
       for ox_S=1:3
         O[m_L2][oy_S][ox_S]+=
          W[m_L2][fy_L2][fx_L1]×
          I[oy_S+fy_L2-1]
           [ox_S+fx_L1-1];
```

data in RF is I: 1x3
W: 1x1x3, O: 1x1x1

Ofmaps
Execute
Spatially



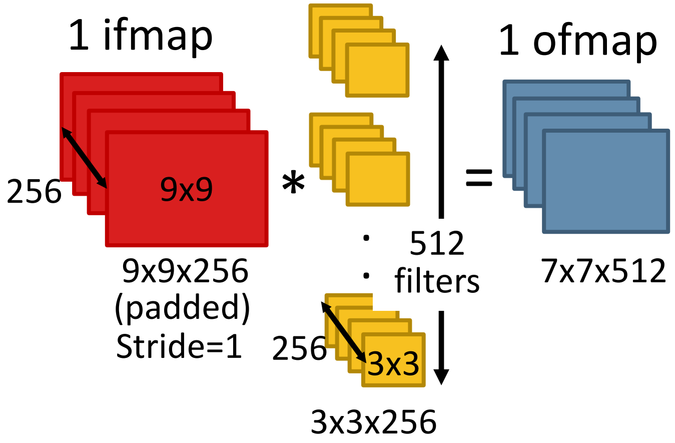
Vast "Execution Method" Space

- Many many ways to execute nested loops (of DNN) on a dataflow accelerator
  - Both software and hardware design space
  - **Hardware:** Size, layout and connectivity of PEs, SPM size, no. of regs, NOC params, etc.
  - **Software:** loop mappings, e.g., **Spatial:** output stationary, or row stationary, **Temporal:** order and tiling of loops, data buffering, etc.

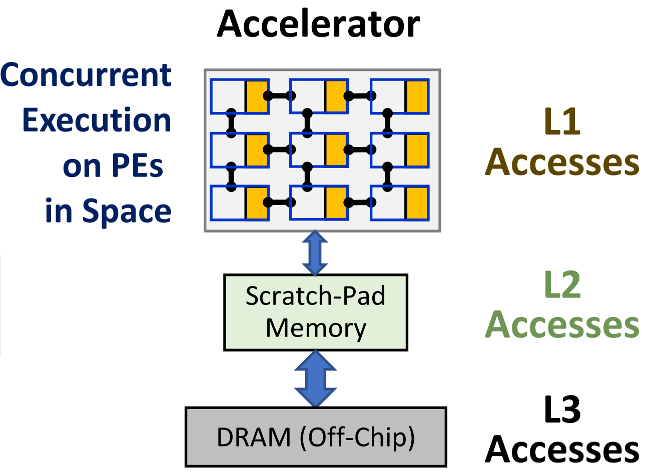
4D Convolution:

```
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];
```

```
<N,M,C,Ox,Oy,Fx,Fy> =
 <1,512,256,7,7,3,3>
```



Conv5_2 [ResNet]



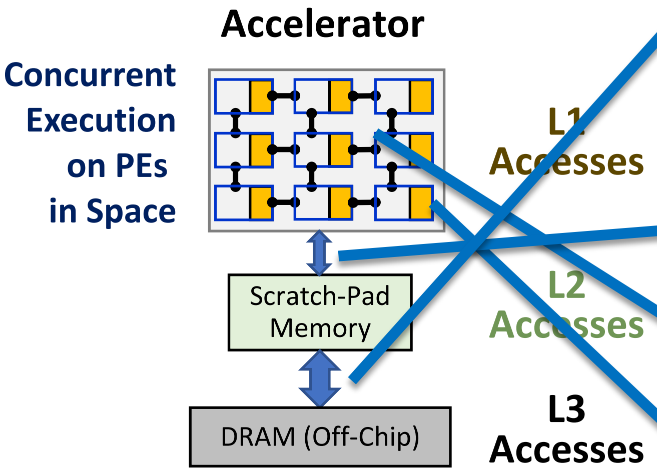
Orchestration of Loops

4D Convolution:

```
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];
```

```
<N, M, C,Ox,Oy,Fx,Fy> =
<8,64,32, 3, 3, 3, 3>
```

Ifmap: 8, 5x5x32
Filters: 64, 3x3x32
Ofmap: 8, 3x3x64



```
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[][][][];
      }
   }
}
```

All the tilings and re-orderings capture a vast space of "hardware-software execution methods" of the nested loops on the dataflow accelerator

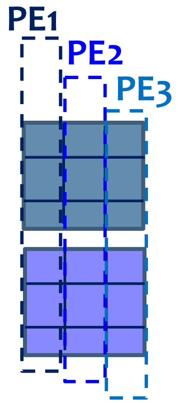
Config#1: 1D Spatial Execution

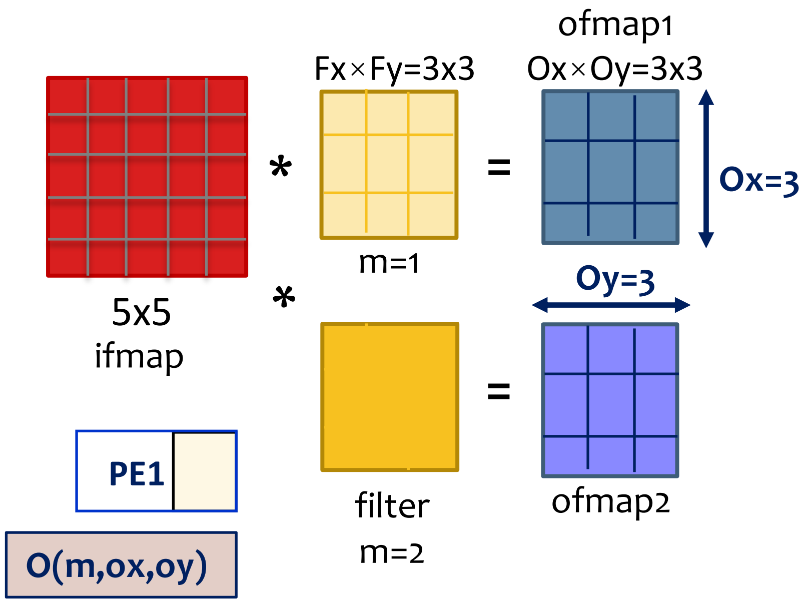
```
for m=1:2
 for ox=1:3
  for oy=1:3
   for fx=1:3
    for fy=1:3
      O[m][ox][oy]+= W[m][fx][fy]×
                I[ox+fx-1][oy+fy-1];
```

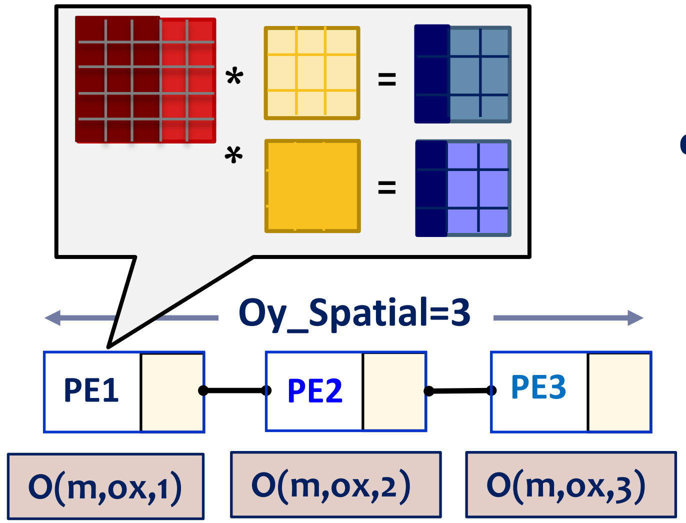
```
for m_T=1:2
 for fx_T=1:3
  for fy_T=1:3
   for ox_T=1:3
-------------------------------------
#pragma unroll spatial
 for oy_S=1:3
   O[m][ox][oy]+= W[m][fx][fy]×
           I[ox+fx-1][oy+fy-1];
```

Remaining loops execute Temporally on every PEs

Ofmap columns execute Spatially







execution on PE1

O[m_T][ox_T][1] +=
W [m_T][fx_T][fy_T]×
I [ox_T+fx_T-1][fy_T]

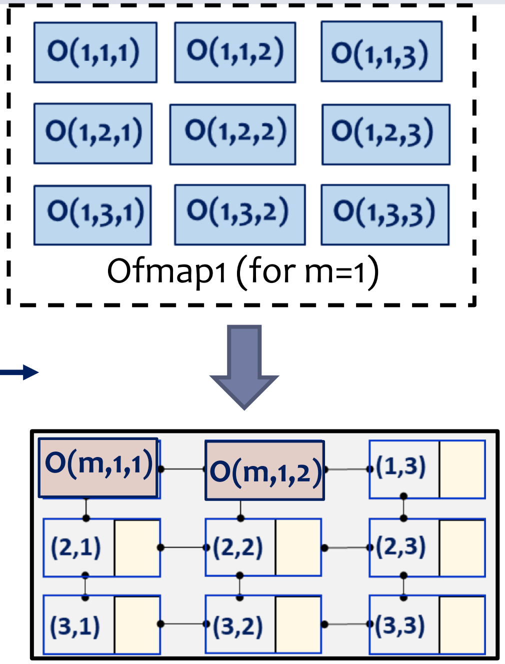
Config#2: 2D Spatial Execution

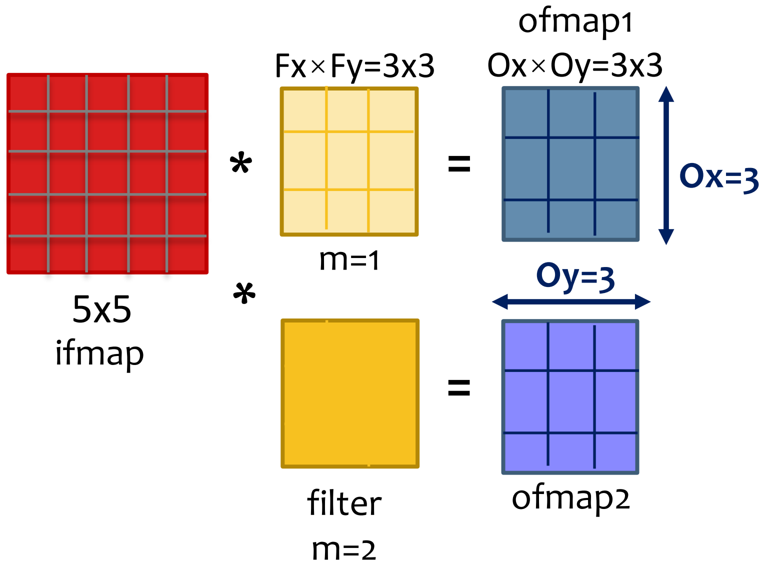
```
for m=1:2
 for ox=1:3
  for oy=1:3
   for fx=1:3
    for fy=1:3
      O[m][ox][oy]+= W[m][fx][fy]×
                  I[ox+fx-1][oy+fy-1];
```

```
for m_T=1:2
 for fx_T=1:3
  for fy_T=1:3
-----------------------------------
  #pragma unroll spatial
   for ox_S=1:3
    for oy_S=1:3
      O[m][ox][oy]+= W[m][fx][fy]×
               I[ox+fx-1][oy+fy-1];
```

Loops for Filter Weights execute Temporally on every PEs

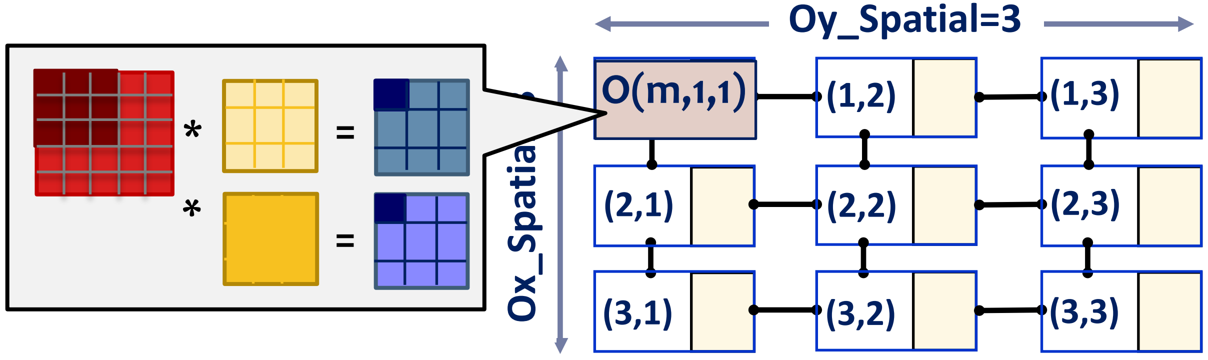
Ofmaps execute Spatially





execution on PE(1,1)

O[m_T][1][1] +=
W [m_T][fx_T][fy_T]×
I [fx_T][fy_T]



Output Stationary Dataflow Mechanism

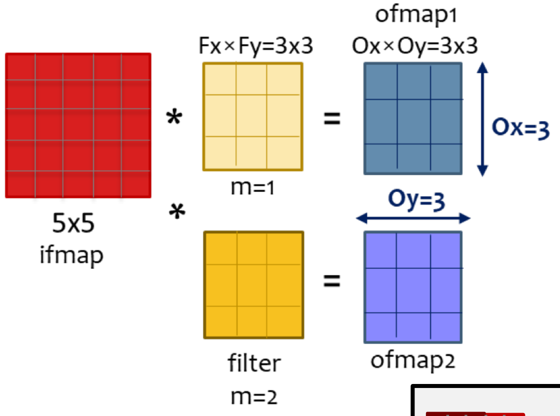
Config#3: 3D Spatial Execution

```
for m=1:2
 for ox=1:3
  for oy=1:3
   for fx=1:3
    for fy=1:3
      O[m][ox][oy]+= W[m][fx][fy]×
                  I[ox+fx-1][oy+fy-1];
```

```
for fx_T=1:3
 for fy_T=1:3
-----------------------------------------
  #pragma unroll spatial
  for m_S = 1:2
   for ox_S=1:3
    for oy_S=1:3
     O[m][ox][oy]+= W[m][fx][fy]×
            I[ox+fx-1][oy+fy-1];
```

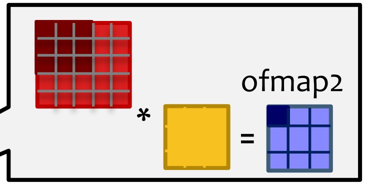
Loops for weights execute Temporally on every PEs

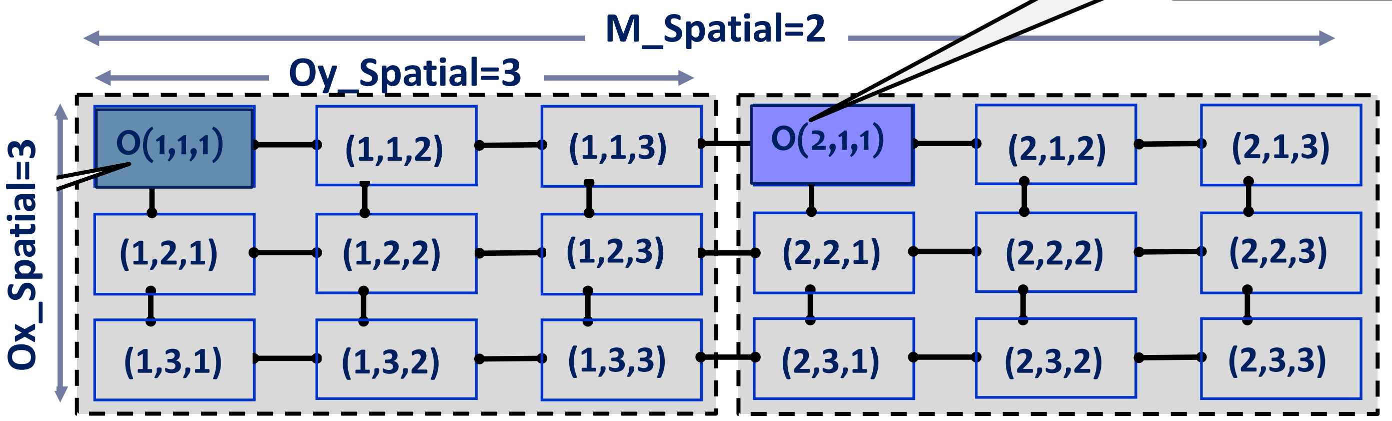
Entire ofmap execute Spatially

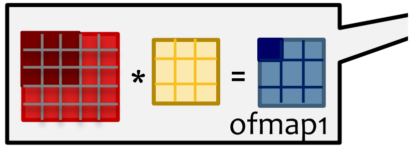


execution on PE(1,1,1) O [1][1][1] += W [1][fx_T][fy_T]× I [fx_T][fy_T]

execution on PE(2,1,1) O [2][1][1] += W [2][fx_T][fy_T]× I [fx_T][fy_T]







Reordering of the loops → Different dataflow

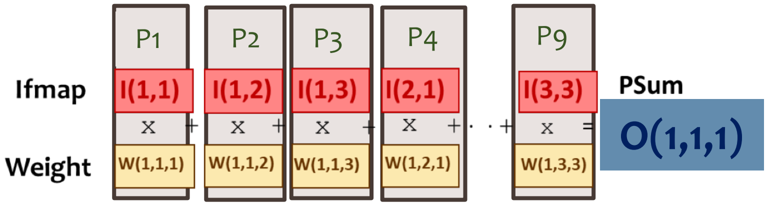
```
for m=1:2
 for ox=1:3
  for oy=1:3
   for fx=1:3
    for fy=1:3
      O[m][ox][oy]+=
              W[m][fx][fy]×
              I[ox+fx-1][oy+fy-1];
```

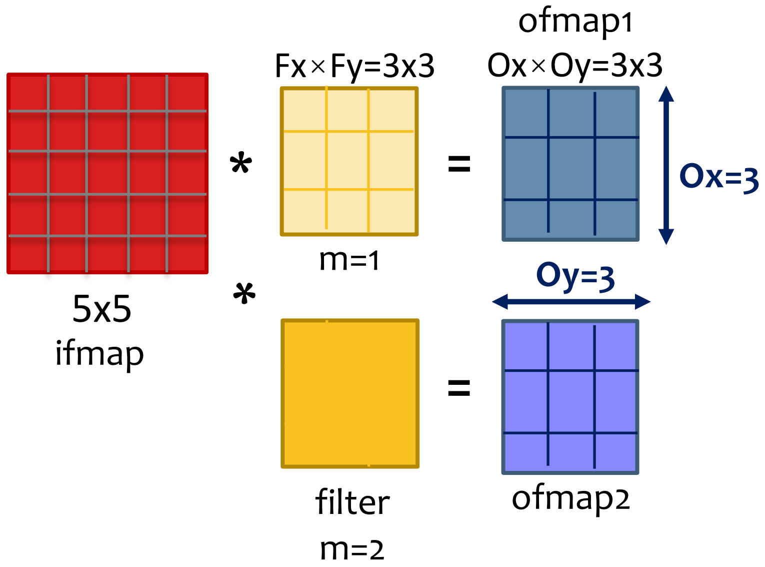
```
for m_T=1:2
 for ox_T=1:3
  for oy_T=1:3
---------------------------------------
   #pragma unroll spatial
   for fx_S=1:3
    for fy_S=1:3
      O[m][ox][oy]+=
              W[m][fx][fy]×
              I[ox+fx-1][oy+fy-1];
```

Loops for Filter Weights execute Temporally on every PEs

Weights laid out Spatially

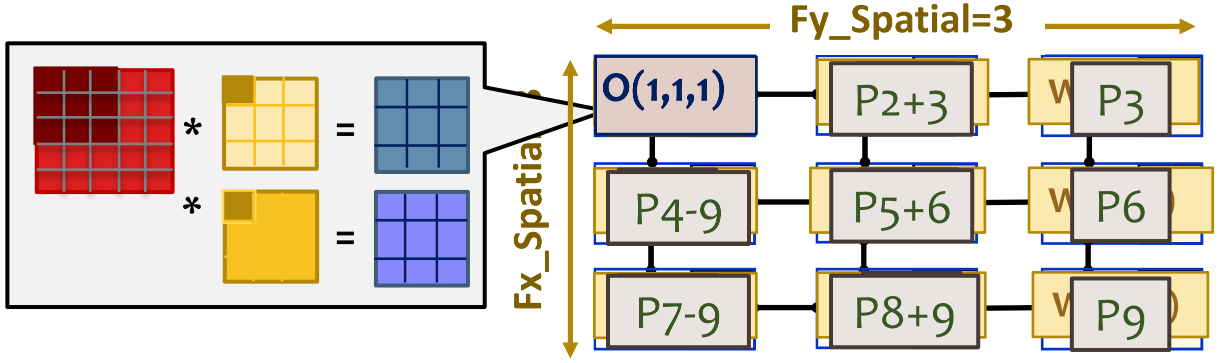
Unrolling some other two loops spatially





execution on PE(1,1)

O[m_T][ox_T][oy_T] +=
W [m_T][1][1]×
I [ox_T][oy_T]

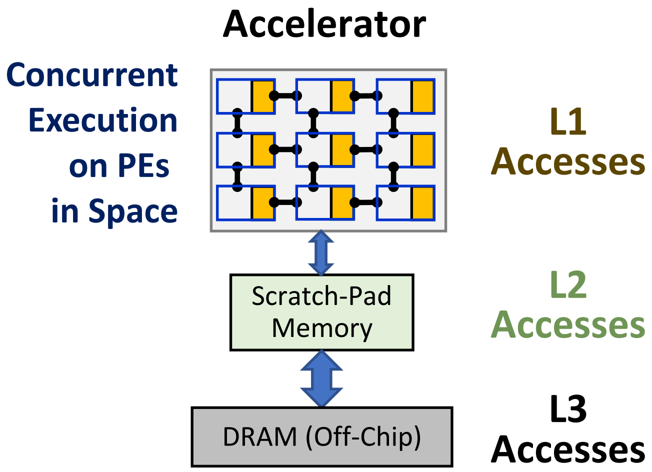


Weight Stationary Dataflow Mechanism

Exploration of “execution methods”

4D Convolution:

```
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];
```



```
<N, M, C,Ox,Oy,Fx,Fy> =
<8,64,32, 3, 3, 3, 3>
```

Ifmap: 8, 5x5x32

Filters: 64, 3x3x32

Ofmap: 8, 3x3x64

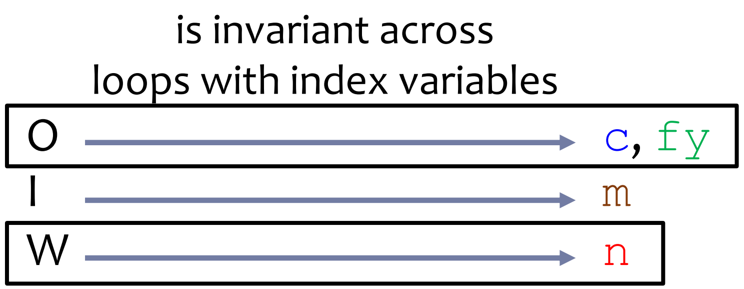
The problem of exploring the “execution methods” becomes the problem of exploring all the possibilities of tiling and ordering factors in the 28-dimensional loop

```
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[][][][];
                     }
              }
      }
```

Drastic pruning of Search Space

Example: Generating loop-orderings with unique data reuse factors

```
for n = 1:N=2
  for m = 1:M=8
    for c = 1:C=4
      for fy = 1:Fy=3
        O[n][m] += I[n][c][fy] * W[m][c][fy]
```



is invariant across
loops with index variables

O → c, fy

I → m

W → n

5 schedules with unique reuse costs as compared to 4! = 24 schedules

Interpretation:
2*8*4*3 accesses total
W is loaded only 8*4*3 times

Reuse Factor for Data Operands

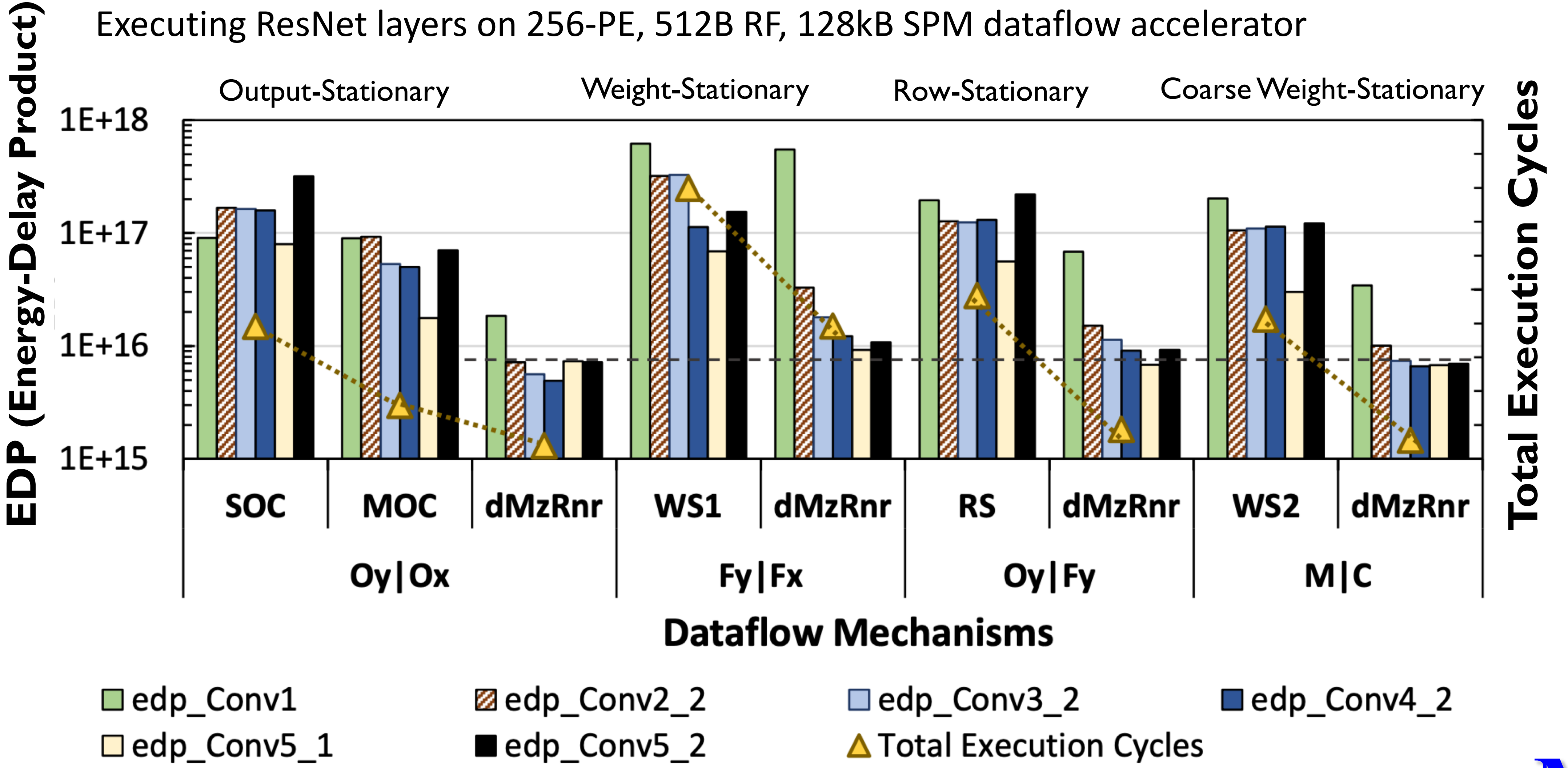
Schedule	I	W	O
{. . . , n}	-	N = 2	-
{. . . , m}	M = 8	-	-
{. . . , m, c}	-	-	C = 4
{. . . , m, fy}	-	-	Fy = 3
{. . . , m, fy, c}	-	-	C*Fy = 4*3

Loop with index variable ‘n’ is innermost

Operand ‘O’ is reused across both loops

Results: 9X reduction in EDP

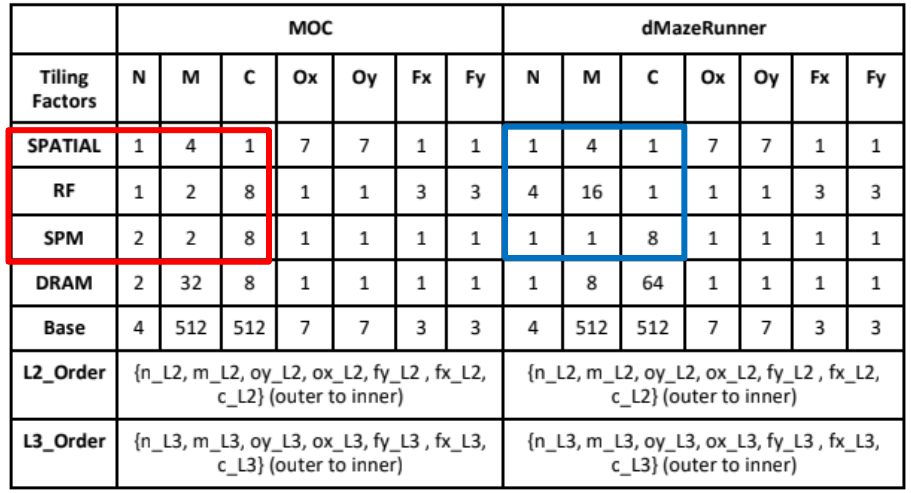
Executing ResNet layers on 256-PE, 512B RF, 128kB SPM dataflow accelerator



Adaptable Mappings Yield Better Results

- Adapts to kernel/arch characteristics
  - Scales for layers/tensors of different shapes
- Finds non-intuitive mappings that optimizes various factors e.g.,
  - ✓ High resource utilization
  - ✓ Maximized reuse of multiple data operands
  - ✓ Minimized DRAM accesses
  - ✓ Efficient interleaving of computation with communication latency

Example Mappings of ResNet Conv5_2 for Output-Stationary Dataflow



	MOC							dMazeRunner						
Tiling Factors	N	M	C	Ox	Oy	Fx	Fy	N	M	C	Ox	Oy	Fx	Fy
SPATIAL	1	4	1	7	7	1	1	1	4	1	7	7	1	1
RF	1	2	8	1	1	3	3	4	16	1	1	1	3	3
SPM	2	2	8	1	1	1	1	1	1	8	1	1	1	1
DRAM	2	32	8	1	1	1	1	1	8	64	1	1	1	1
Base	4	512	512	7	7	3	3	4	512	512	7	7	3	3
L2_Order	{n_L2, m_L2, oy_L2, ox_L2, fy_L2 , fx_L2, c_L2} (outer to inner)							{n_L2, m_L2, oy_L2, ox_L2, fy_L2 , fx_L2, c_L2} (outer to inner)						
L3_Order	{n_L3, m_L3, oy_L3, ox_L3, fy_L3 , fx_L3, c_L3} (outer to inner)							{n_L3, m_L3, oy_L3, ox_L3, fy_L3 , fx_L3, c_L3} (outer to inner)						

MOC: Simultaneous spatial processing of Multiple Output Channels [1, 2]

For data allocated in RFs of PEs,	**MOC**	**dMzRnr**
PE Compute vs. Data comm. Latency:	144 vs. 648	**576 vs. 576**
Total cycles:	~10,616,832	~2,459,648
Ideal execution cycles for output-stationary:	2,359,296	2,359,296
Reduction in DRAM accesses (ifmaps, weights):	(1x, 1x)	**(4.57x, 2x)**
Perf. improvement (normalized to MOC):	1x	**4.44x**
Energy-Delay-Product reduction (normalized):	1x	**9.86x**

[1] S. Gupta et al. Deep learning with limited numerical precision. In ICML, 2015.
[2] Y. Chen et al. Eyeriss: A spatial architecture for energy-efficient dataflow for CNNs. In ISCA 2016.

Achieving Close-to-Optimal Solutions in Seconds

- **Even domain non-experts can explore the space**

```
python run_optimizer.py --frontend mxnet
       --model resnet18 --layer-index 0
```

- **Does not preclude experts**/programmers from directing the search.
  - In-built support for a few common opt strategies.
- Quick exploration:
  - EDP ~2% higher vs. optimal of brute-force search (**seconds vs. days/hours**)
  - Implementation – **multi-threaded, caches commonly invoked routines** of analytical model.
  - Enables effective DSE of architecture.

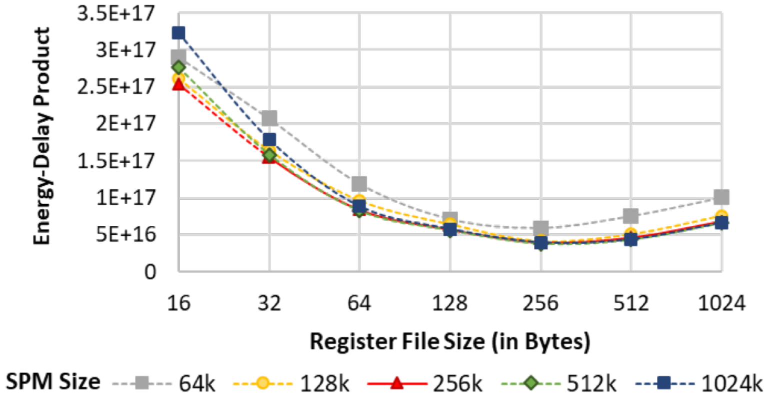
[Alpha Release] https://github.com/cmlasu/dMazeRunner

Search Space Exploration on an Intel i7-6700 Quad-core CPU
min: 1 second, ResNet conv5_2 (753 methods)
max: 122 seconds, ResNet conv2_2 (122092 methods)

[dMazeRunner, CODES+ISSS '19]

Optimizing Memory Sizes for ResNet18 Layers

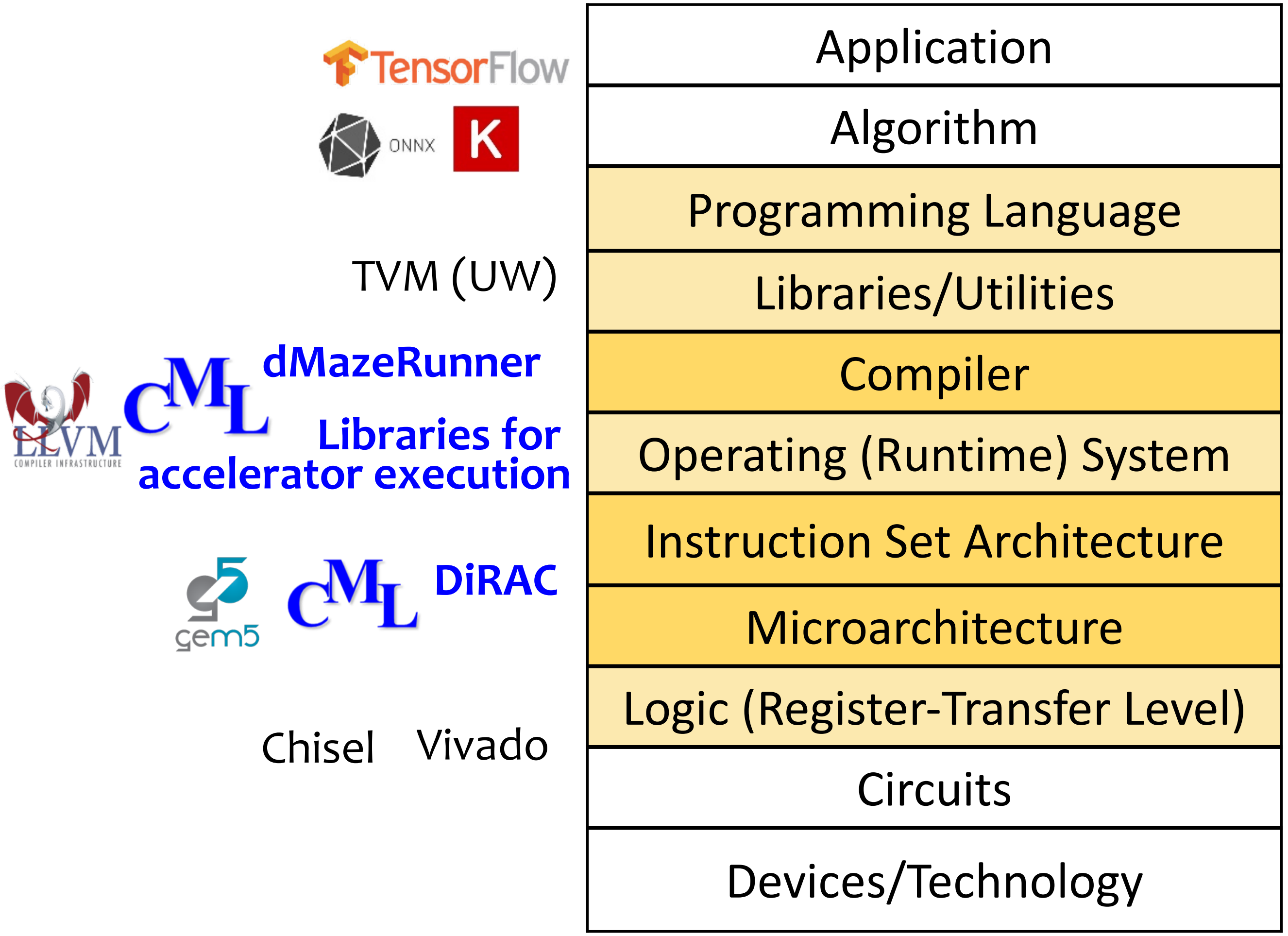
DSE for 256-PE CGRA



```
python tune_memory.py run <layer index>
```

Summary and Next Steps

- Coarse-grained dataflow accelerators promising for accelerating ML models.
  - Challenge: Programming the accelerators
  - System stack can extend the applicability.
- dMazeRunner App Mapping Framework
  - Analytical Power and performance model
  - Automated Design Space Exploration
- End-to-end system [WIP]
  - Programmable Microarch + Simulator
  - FPGA emulation
- Further Opportunities
  - Sparsity [WIP]: Support dynamic sparsity of varying levels (inference + training).
  - Multi-chip module accelerations
- Exciting times ahead!



TVM (UW)

dMazeRunner

Libraries for accelerator execution

DiRAC

Chisel Vivado

Application

Algorithm

Programming Language

Libraries/Utilities

Compiler

Operating (Runtime) System

Instruction Set Architecture

Microarchitecture

Logic (Register-Transfer Level)

Circuits

Devices/Technology

Web page: aviral.lab.asu.edu