MODELING AND ANALYZING DEEP LEARNING ACCELERATOR DATAFLOWS WITH MAESTRO

Michael Pellauer*, Hyoukjun Kwon** and Tushar Krishna**

*Architecture Research Group, NVIDIA

**Georgia Institute of Technology

ACKNOWLEDGMENTS

Many slides from: Professors Joel Emer (NVIDIA, MIT) and Vivienne Sze's (MIT) class

- 6.S082/6.888 Hardware Architectures for Deep Learning (Fall, 2017)

Also includes slides from: Angshuman Parashar, Senior Research Scientist (NVIDIA)

Also influenced by:

Steve Keckler (NVIDIA)
Jason Clemons (NVIDIA)
Sophia Shao (NVIDIA)
Cristopher Fletcher (UIUC)

Former Nvidia interns:

Yu-Hsin Chen (MIT)
Anurag Mukkara (MIT)
Animesh Jain (U Mich.)

TALK OUTLINE

Motivation and Background

- Why architecture researchers should care about dataflow

Concepts in Tiling, Blocking, and Scheduling

- Basic example: 1D-convolution
- Adding intermediate staging buffers
- Extending to full CNN layers

MAESTRO: Modeling Accelerator Efficiency via Spatio-Temporal Resource Occupancy

- Analytic approach for architectural exploration
- Hands-on lab

MOTIVATION: DATA MOVEMENT

Why it's important

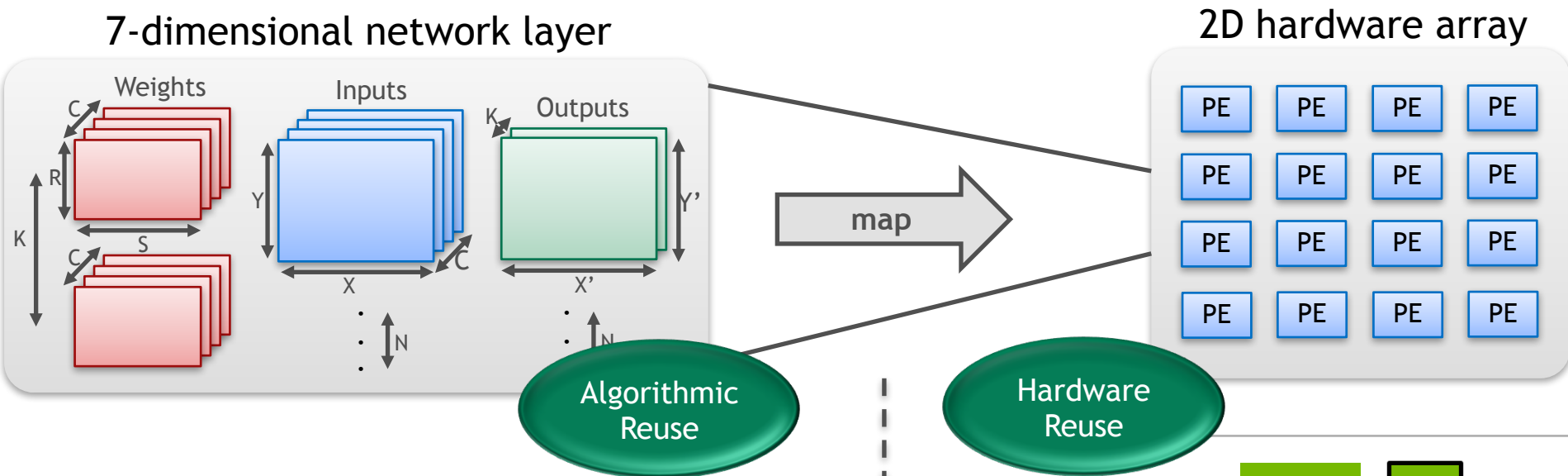
Energy costs	
8-bit Integer Multiply	1x
Fetch two 8-bit operands from DRAM	~100x
Fetch two 8-bit operands from large SRAM	~10x

Fortunately…

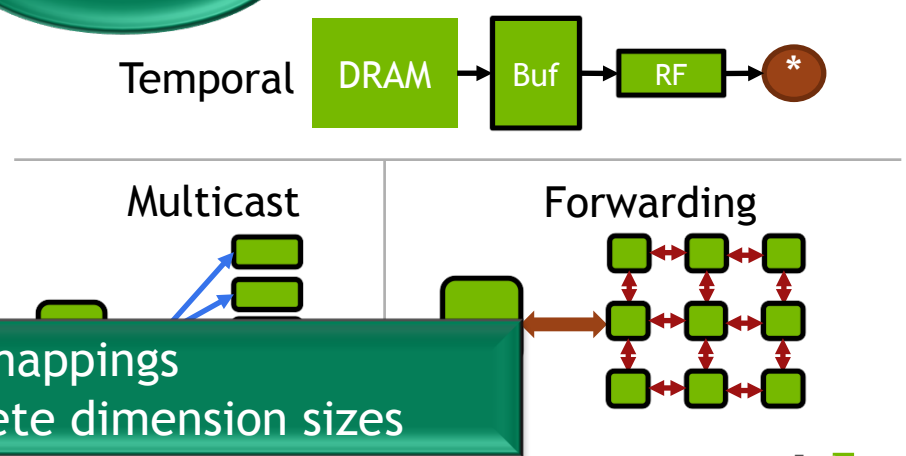
VGG16 conv 3_2	
Multiply Add Ops	1.85 Billion
Weights	590 K
Inputs	803 K
Outputs	803 K

Re-use

MAPPING REUSE TO HARDWARE



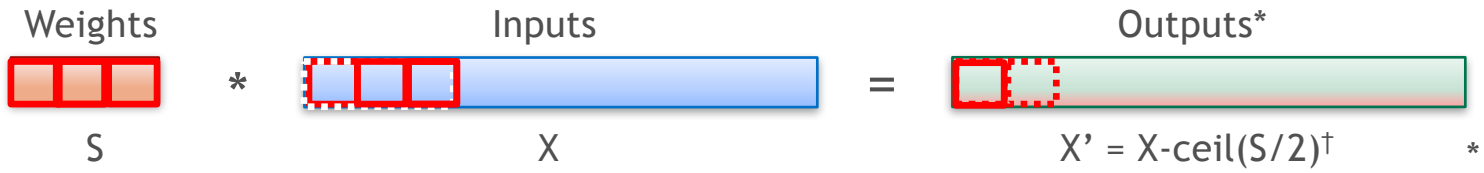
- **7D Computation Space**
  - R * S * X * Y * C * K * N
- **4D Operand / Result Spaces -**
  - Weights - R * S * C * K
  - Inputs - X * Y * C * N
  - Outputs



Millions of non-trivial mappings
Efficiency is dependent on concrete dimension sizes

LEARNING ABOUT DATAFLOWS

PEDAGOGAL EXAMPLE: 1-D CONVOLUTION



* The terms "Output" and "Partial Sum" used interchangeably

† Assuming: 'valid' style convolution

```
int i[X];      # Input activations
int w[S];      # Filter weights
int o[X'];     # Output activations

for (x = 0; x < X'; x++) {
        for (s = 0; s < S; s++) {
            o[x] += i[x+s]*w[s];
}
```

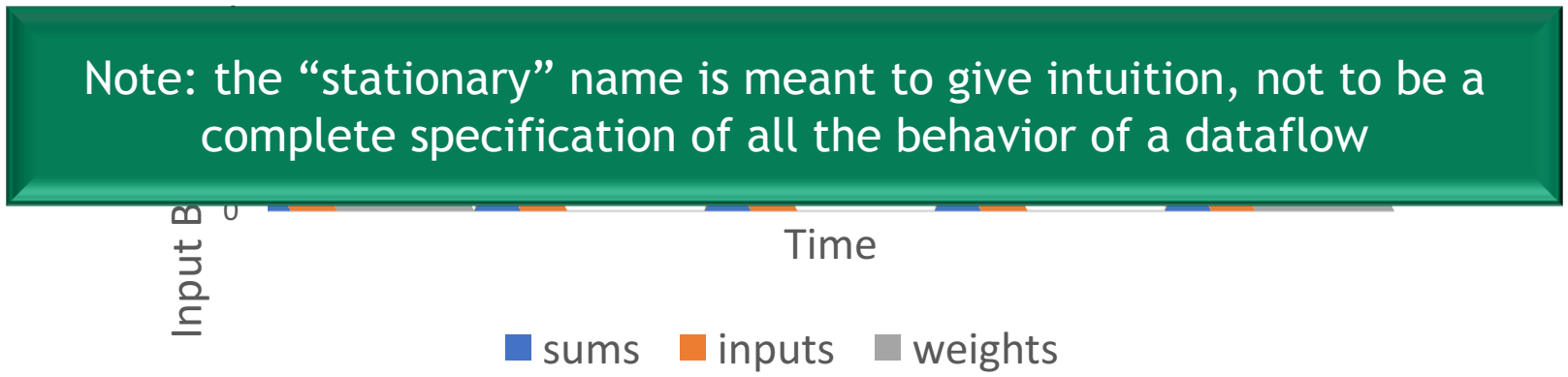
How often does the datapath change the weight and input? — Every cycle

Output? — Every R cycles: "Output stationary"

WHAT DO WE MEAN BY "STATIONARY"?

The datatype (and dimension) that changes most slowly

Sums: 1/10, Inputs: 3/10, Weights: 9/40



Note: the "stationary" name is meant to give intuition, not to be a complete specification of all the behavior of a dataflow

Imprecise analogy: think of data transfers as a wave with "amplitude" and "period"

- The stationary datatype has the **longest** period (locally held tile changes most slowly)
- Note: like waves, also can have harmful "constructive interference" (bursts)
- Often corresponds to datatype that is "done with" earliest without further reloads

Later we will see how intermediate staging buffers reduce both bandwidth and energy

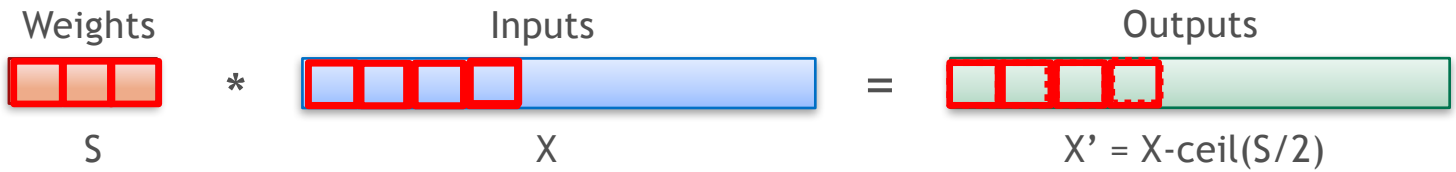
FROM "LOOP NEST" TO DATAFLOW

```
int i[X];       # Input activations
int w[S];       # Filter weights
int o[X'];      # Output activations

for (x = 0; x < X'; x++) {
        for (s = 0; s < S; s++) {
            o[x] += i[x+s]*w[s];
}
```

No constraints* on loop permutations!

* Because we don't care about where precision/saturation issues occur – usually choose data sizes such that it never happens [See NVDLA's 48-bit accumulators for 16-bit operands]

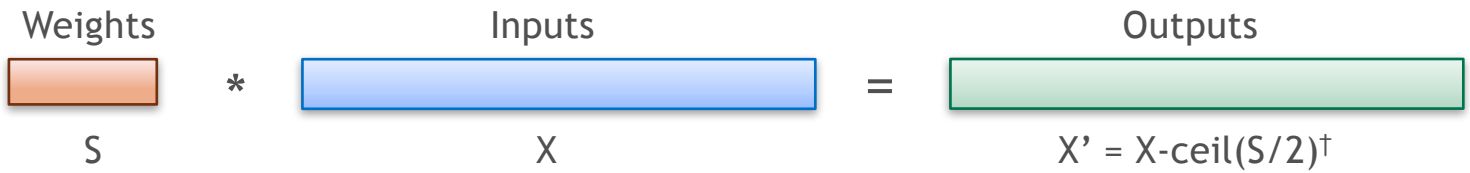
ANOTHER DATAFLOW




```
int i[X];       # Input activations
int w[S];       # Filter weights
int o[X’];      # Output activations

for (s = 0; s < S; s++) {
        for (x = 0; x < X’; x++) {
            o[x] += i[x+s]*w[s];
}
```

What dataflow is this? Weight stationary

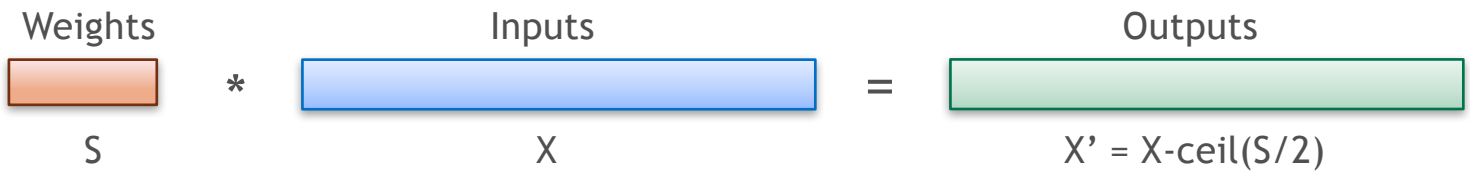
MORE DATAFLOWS




```
int i[X];      # Input activations
int w[S];      # Filter weights
int o[X’];      # Output activations

for (x = 0; x < X’; x++) {
        for (s = 0; s < S; s++) {
            o[x] += i[x+s]*w[s];
}
```

How can we implement input stationary with no input index?

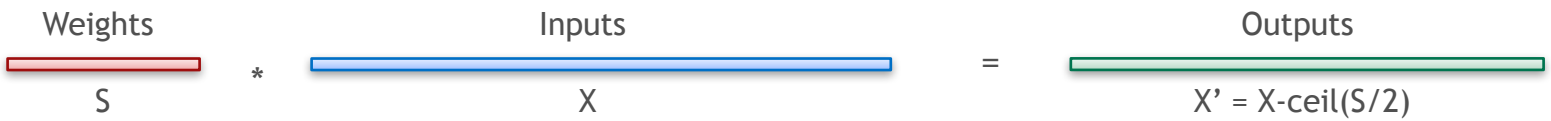
INPUT STATIONARY

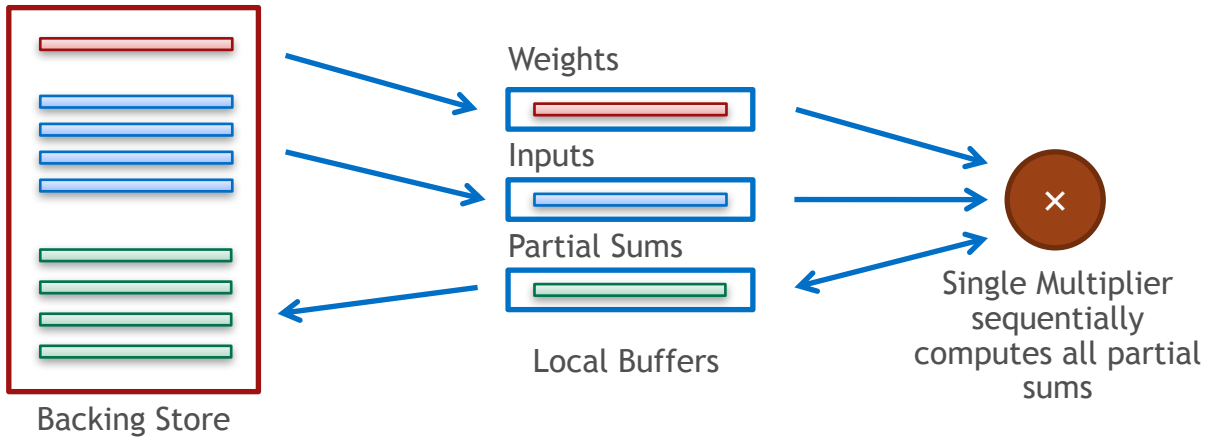



```
int i[X];      # Input activations
int w[S];      # Filter weights
int o[X’]      # Output activations

for (x = 0; x < X; x++) {
      for (s = 0; s < S; s++) {
         o[x-s] += i[x]*w[s];
}
```

Beware w-r must be >= 0 and <E

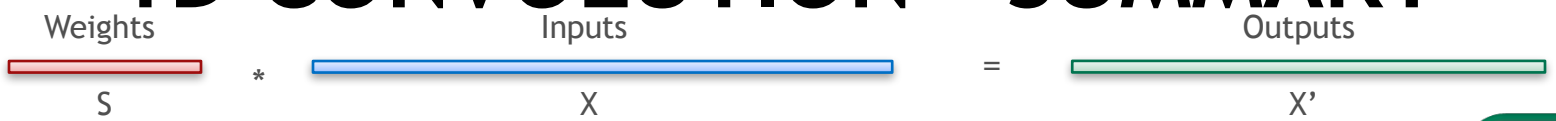
SIMPLE MODEL FOR MAPPING DATAFLOWS TO HARDWARE





Common metric	Weights	Inputs	Outputs / Partial Sums
Alg. Min. accesses to backing store (MINALG)	S	X	X'
Maximum operand uses (MAXOP)	SX'	SX'	SX'

1D CONVOLUTION - SUMMARY



Common metric	Weights	Inputs	Outputs / Partial Sums
Size = Alg. Min. accesses	S	X	X'
Maximum operand uses	SX'	SX'	SX'

Note: product always equals SX'

BUFSIZE-1D (Buffer size for zero re-fetch)

Dataflow	Weights	Inputs	Outputs
Weight-stationary	1	X'	X'
Input-stationary	S	1	S
Output-stationary	S	S	1

BUFMULT-1D (#times f[illegible]uffer is accessed)

Dataflow	Weights	Inputs	Outputs
Weight-stationary	SX'	S	S
Input-stationary	X'	SX'	X'
Output-stationary	X'	X'	SX'

BSACCESS-1D (Backing store access counts)

Dataflow	Weights	Inputs	Outputs
Weight-stationary	S	X	X'
Input-stationary	S	X	X'
Output-stationary	S	X	X'

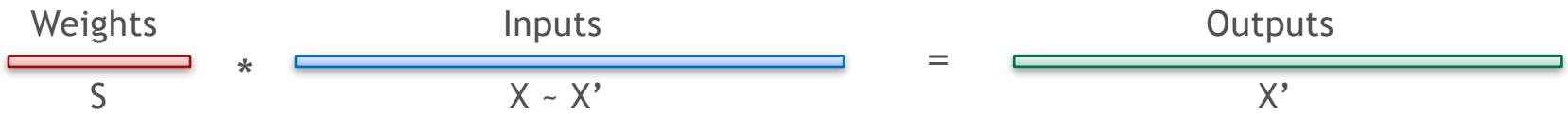
= Alg. Min. accesses

BUFACCESS-1D (Buffer access expense) = BUFMULT * BUFSIZE

Dataflow	Weights	Inputs	Outputs
Weight-stationary	SX'(1)	S(X')	S(X')
Input-stationary	X'(S)	SX'(1)	X'(S)
Output-stationary	X'(S)	X'(S)	SX'(1)

= Maximum operand uses!

1D CONVOLUTION - SUMMARY



Buffer size for zero re-fetch

Dataflow	Weights	Inputs	Outputs
Weight-stationary	1	X'	X'
Input-stationary	S	1	S
Output-stationary	S	S	1

Buffer accesses

Dataflow	Weights	Inputs	Outputs
Weight-stationary	SX'(1)	S(X')	S(X')
Input-stationary	X'(S)	SX'(1)	X'(S)
Output-stationary	X'(S)	X'(S)	SX'(1)

Buffer access energy:

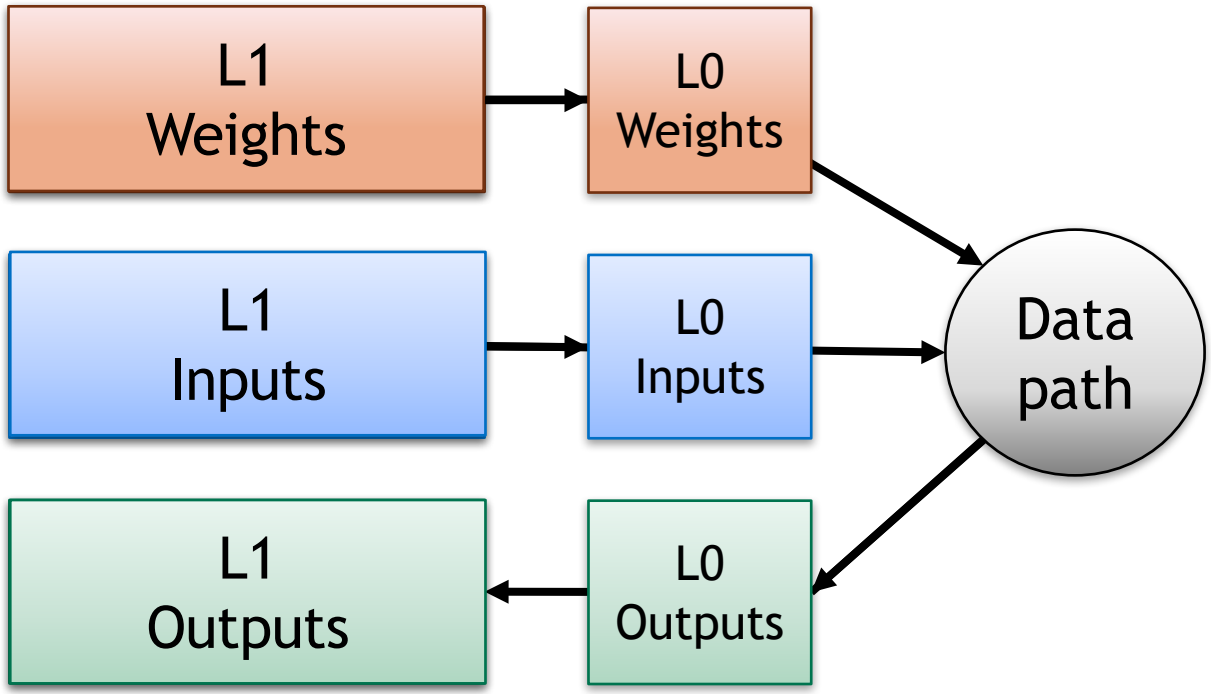
```
WS = SX'[f(1) + f(X') + 2f(X')]
IS = SX'[f(S) + f(1) + 2f(S)]
OS = SX'[f(S) + f(S) + 2f(1)]
```

f(x) = energy cost of accessing a RAM structure of size x

Significant difference in buffer access energy cost based on dataflow
But what if the provisioned buffering is smaller than required?

GETTING MORE REALISTIC

MULTI-LAYER BUFFERING



1-D CONVOLUTION - BUFFERED

```
int i[X];      # Input activations
int w[S];      # Filter Weights
int o[X’];     # Output activations

// Level 1
for (x1 = 0; x1 < X’1; x1++) {
  for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X’0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          x = x1 * X’0 + x0;
          s = r1 * R0 + r0;
          o[x] += i[x+s] * w[s];
}
```

Note X’ and S are factored so:
X’0 * X’1 = X’
S0 * S1 = S

ENERGY COSTS OF A MAPPING

```
// Level 1
for (x1 = 0; x1 < X'1; x1++) {
  for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
        o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0] * w[s1*S0+s0];
}
```

Constant over each level 1 iteration

Energy of a buffer access is a function of the size of the buffer

Each buffer level's energy is proportional the number of accesses at that level

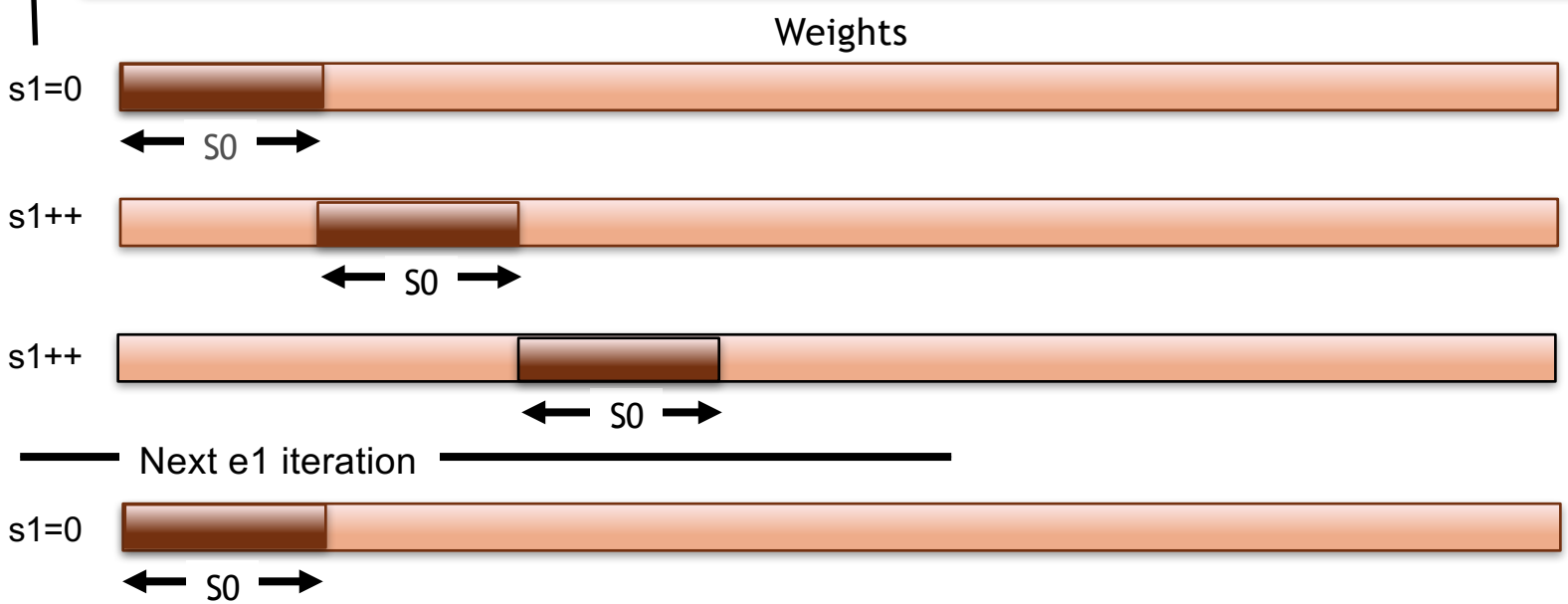
- For level 0 that is all the operands to the Datapath

For level L>0 there are three components:

- Data arriving from level L+1
- Data that needs to be transferred to level L-1
- Data that is returned from level L-1

MAPPING - WEIGHT ACCESS COSTS

```
// Level 1
for (x1 = 0; x1 < X'1; x1++) {
  for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0] * w[s1*S'0+s0];
}
```



MAPPING - WEIGHT ACCESS COSTS

Level 0 reads

Per level 1 iteration -> X’0*S0 weight reads

Times X’1*S1 level 1 iterations

Total reads = (X’0*S0)*(X’1*S1) = (X’0*X’1)*(S0*S1) = SX’ reads

Level 1 to 0 transfers

Per level 1 iteration -> S0 weights transferred

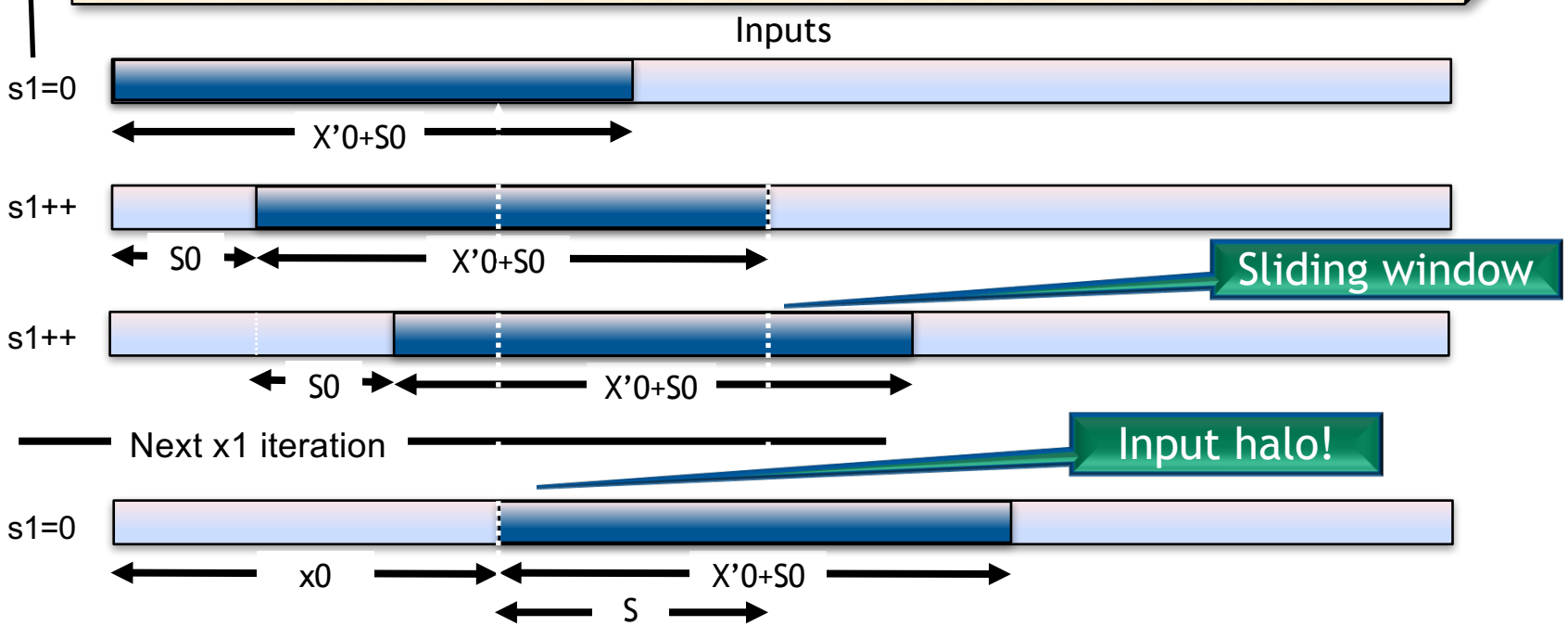
Times same number of level 1 iterations = X’1 * S1

Total transfers -> S0*(X’1*S1) = X’1*(S0*S1) = SX’1

Disjoint/partitioned reuse pattern

MAPPING - INPUT ACCESS COSTS

```
// Level 1
for (x1 = 0; x1 < X'1; x1++) {
  for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0] * w[s1*S0+s0];
}
```



MAPPING - INPUT ACCESS COSTS

Level 0 reads

Per level 1 iteration -> X’0+S0 inputs reads

Times X’1*S1 level 1 iterations

Total reads = X’1*S1*(X’0+S0) = ((X’1*X’0)*S1)+(X’1*(S1*S0)) = X’*S1+X’1*S reads

Level 1 to 0 transfers

For s=0, X’0+S0 inputs transferred

For each of the following S1-1 iterations another S0 inputs transferred

So total per x1 iteration is: X’0+S0*S1 = X’0+S inputs

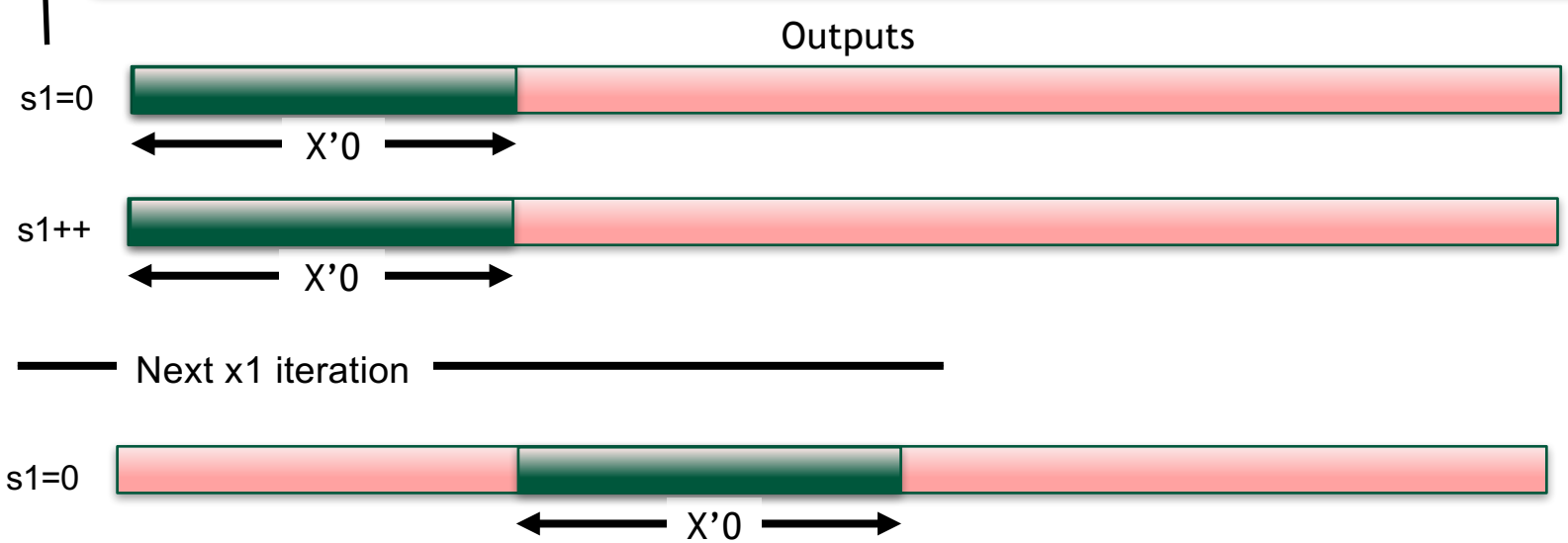
Times number of x1 iterations = X’1

So total transfers = X’1*(X’0+S) = (X’1*X’0)+X’1*S = X’+X’1*S

Sliding window/partitioned reuse pattern

MAPPING - OUTPUT ACCESS COSTS

```
// Level 1
for (x1 = 0; x1 < X'1; x1++) {
  for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0] * w[s1*S0+s0];
}
```



MAPPING - OUTPUT ACCESS COSTS

Level 0 writes

- Due to level 0 being ‘output stationary’ only X’0 writes per level 1 iteration
- Times X’1*S1 level 1 iterations
- Total writes = X’0*(X’1*S1) = (X’0*X’1)*S1 = X’*S1 writes

Level 0 to 1 transfers

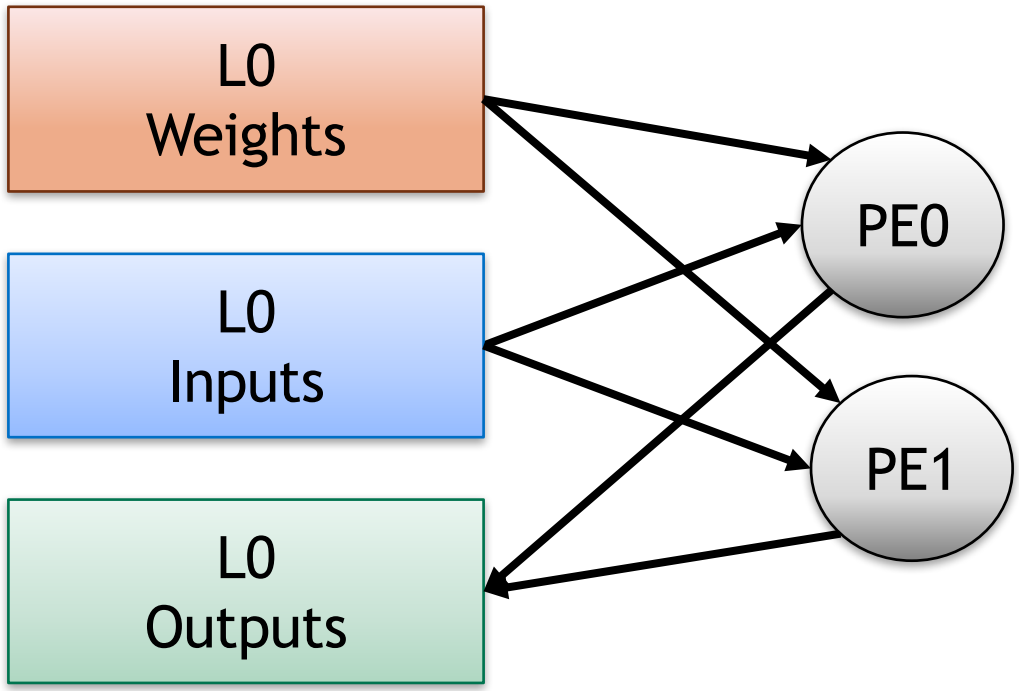
- After each S1 iterations a completed partial sum for X’0 outputs are transferred
- There are X’1 chunks of S1 iterations
- So total is X’1*X’0 = X’ transfers

MAPPING DATA COST SUMMARY

```
// Level 1
for (x1 = 0; x1 < X'1; x1++) {
  for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0]* w[s1*S0+s0];
}
```

	Level 0	Level 1 to 0 transfers
Weight Reads	SX'	SX'1
Input Reads	X' * S1+ X'1 * S	X'+X'1*S
Output Reads	N/A	N/A
Output Writes	X' * S1	X'

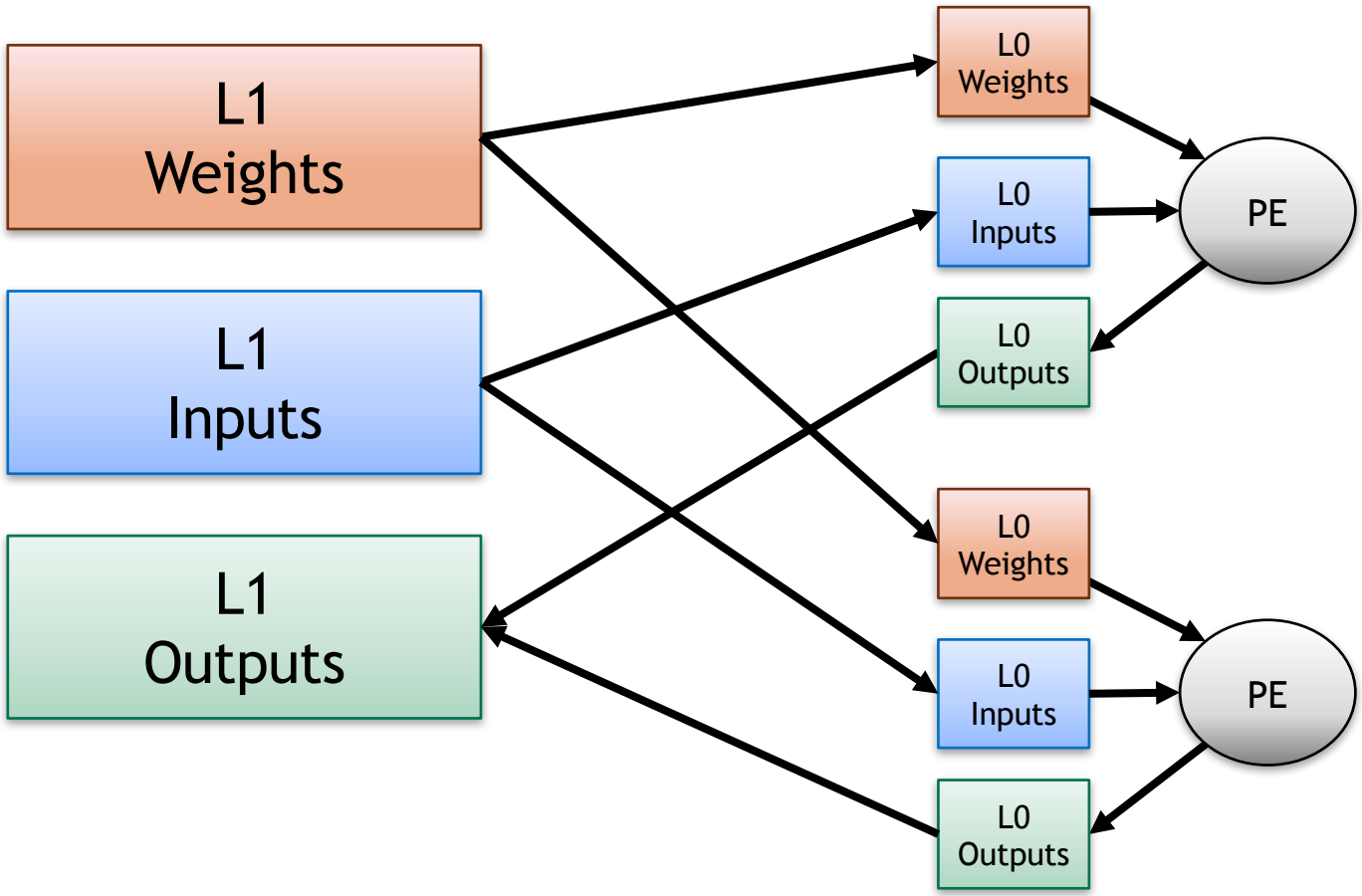
SPATIAL PES



How will this be reflected in the loop nest?

New ‘level’ of loops

INTERMEDIATE BUFFERING



1-D CONVOLUTION – SPATIAL

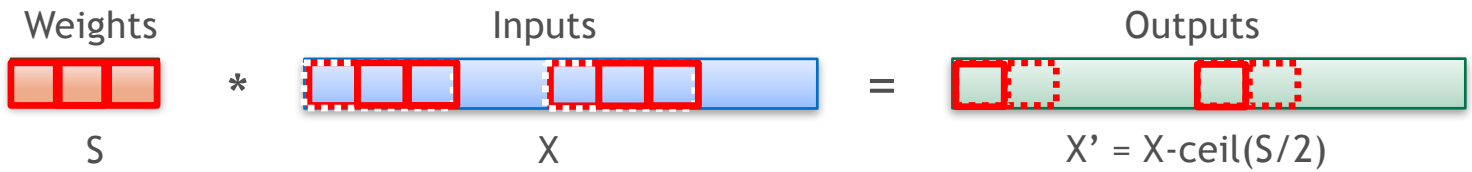
```
int i[X];       # Input activations
int w[S];       # Filter Weights
int o[X’];      # Output activations

// Level 1
parallel-for (x1 = 0; x1 < X’1; x1++) {
  parallel-for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X’0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X’0+x0] += i[x1*X’0+x0 + s1*S0+s0]
                          * w[s1*S0+s0];
}
```

Note:
X’0*X’1 = X’
S0*S1 = S

X’1 = 2

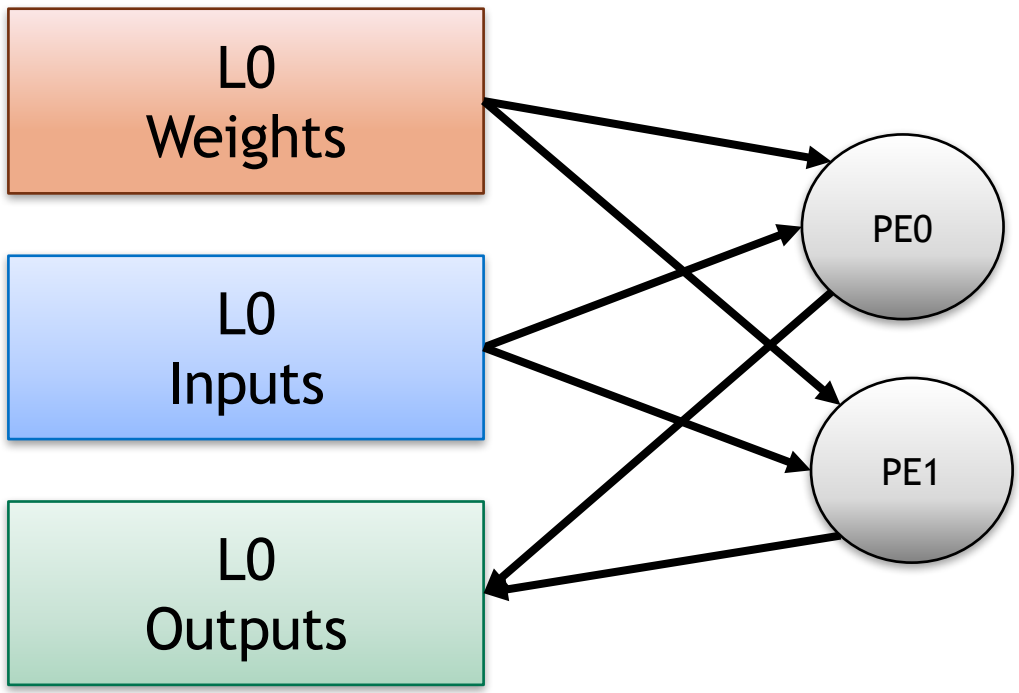
S1 = 1 => s1 = 0

1-D CONVOLUTION – SPATIAL




```
int i[X];      # Input activations
int w[S];      # Filter Weights
int o[X’];     # Output activations

// Level 1
parallel-for (x1 = 0; x1 < 2; x1++) {
// Level 0
    for (x0 = 0; x0 < X’0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X’0+x0] += i[x1*X’0+x0 + s1*S0+s0]
                          * w[s1*S0+s0];
}
```

SPATIAL PES




```
PE0        PE1
>>w[0]     >>w[0]
```

Implementation opportunity? Yes, single fetch and multicast

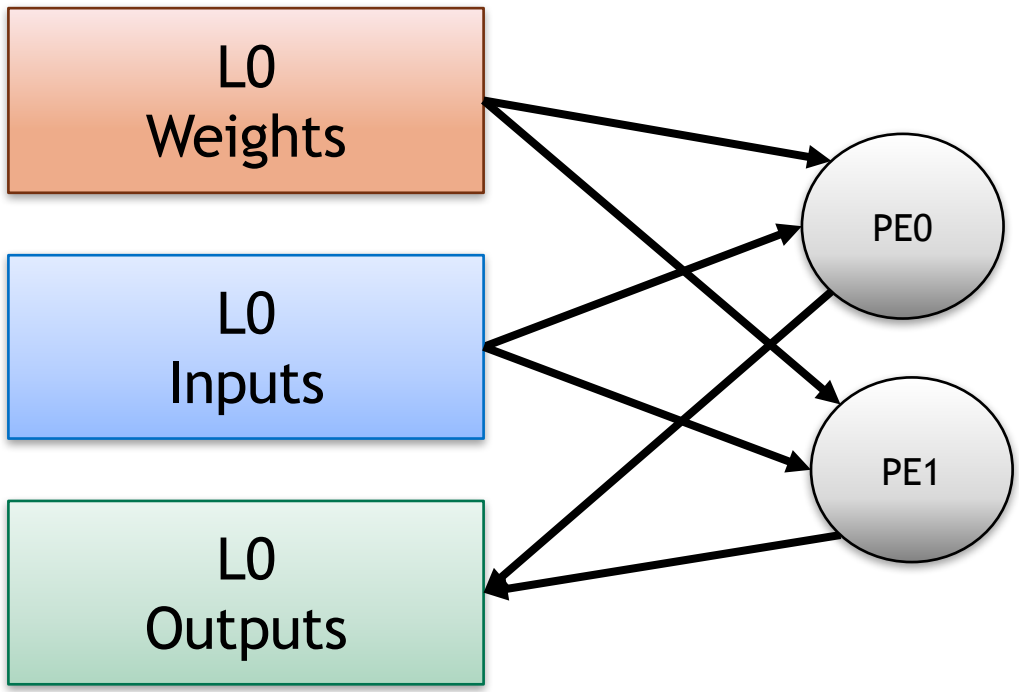
1-D CONVOLUTION – SPATIAL

```
// Level 1
parallel-for (x1 = 0; x1 < 2; x1++) {
// Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0]
                          * w[s1*S0+s0];
}
```

How do we recognize multicast opportunities?

Indices independent of spatial index

SPATIAL PES



PE0	PE1
>>w[0]	>>w[0]
>>i[0]	>>i[X'0+0]
>>w[1]	>>w[1]
>>i[1]	>>i[X'0+1]
>>w[2]	>>w[2]
>>i[2]	>>i[X'0+2]
<<o[0]	<<o[X'0+0]
>>w[0]	>>w[0]
>>i[1]	>>i[X'0+1]
>>w[1]	>>w[1]
>>i[2]	>>i[X'0+2]
>>w[2]	>>w[2]
>>i[3]	>>i[X'0+3]
<<o[1]	<<o[X'0+1]

Implementation opportunity? Parallel fetch

Assuming S=3

SPATIAL PARTITIONING WEIGHTS

```
int i[W];      # Input activations
int w[R];      # Filter Weights
int o[E];      # Output activations

// Level 1
parallel for (x1 = 0; x1 < X’1; x1++) {
  parallel-for (s1 = 0; s1 < S1; s1++) {
    // Level 0
    for (x0 = 0; x0 < X’0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X’0+x0] += i[x1*X’0+x0 + s1*S0+s0]
                          * w[s1*S0+s0];
}
```

Note:
X’0*X’1 = X’
S0*S1 = S

X’1 = 1 => x1 = 0

S0 = 1, S1 = 2

SPATIAL PARTITIONING WEIGHTS

```
int i[X];       # Input activations
int w[S];       # Filter Weights
int o[X’];      # Output activations

// Level 1
parallel-for (s1 = 0; s1 < 2; s1++) {
    // Level 0
    for (x0 = 0; x0 < X’0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
          o[x1*X’0+x0] += i[x1*X’0+x0 + s1*S0+s0]
                        * w[s1*S0+s0];
}
```

Note:
X’0*X’1 = X’
S0*S1 = S

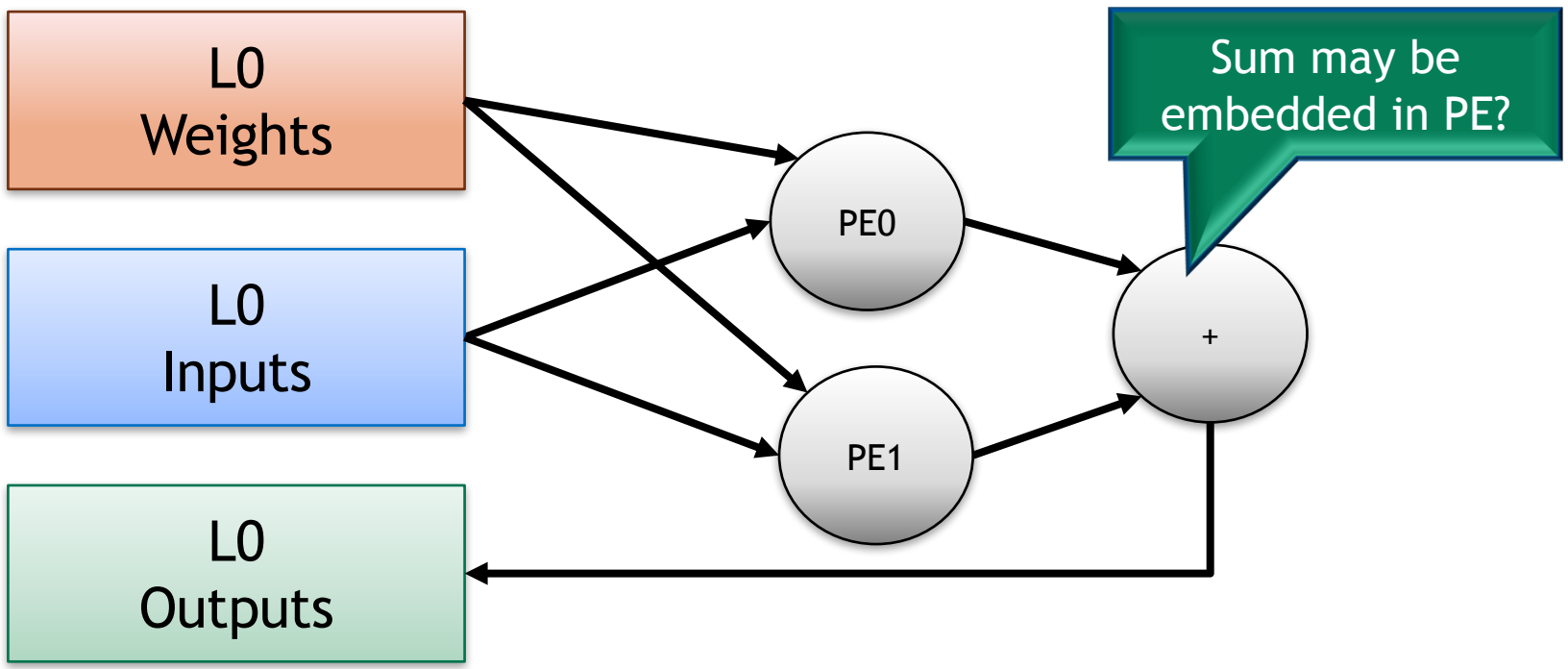
1-D CONVOLUTION – SPATIAL

```
// Level 1
parallel-for (s1 = 0; s1 < 2; s1++) {
    // Level 0
    for (x0 = 0; x0 < X'0; x0++) {
      for (s0 = 0; s0 < S0; s0++) {
        o[x1*X'0+x0] += i[x1*X'0+x0 + s1*S0+s0]
                      * w[s1*S0+s0];
}
```

How do we handle same index for output In multiple PEs? — Spatial summation…

Other multicast opportunities? — No

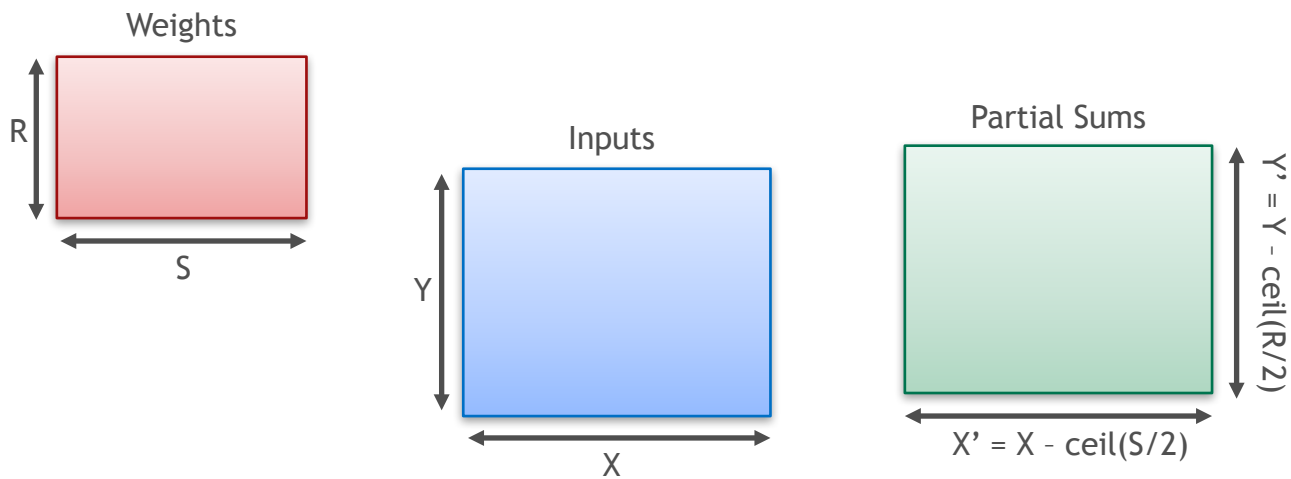
SPATIAL PES



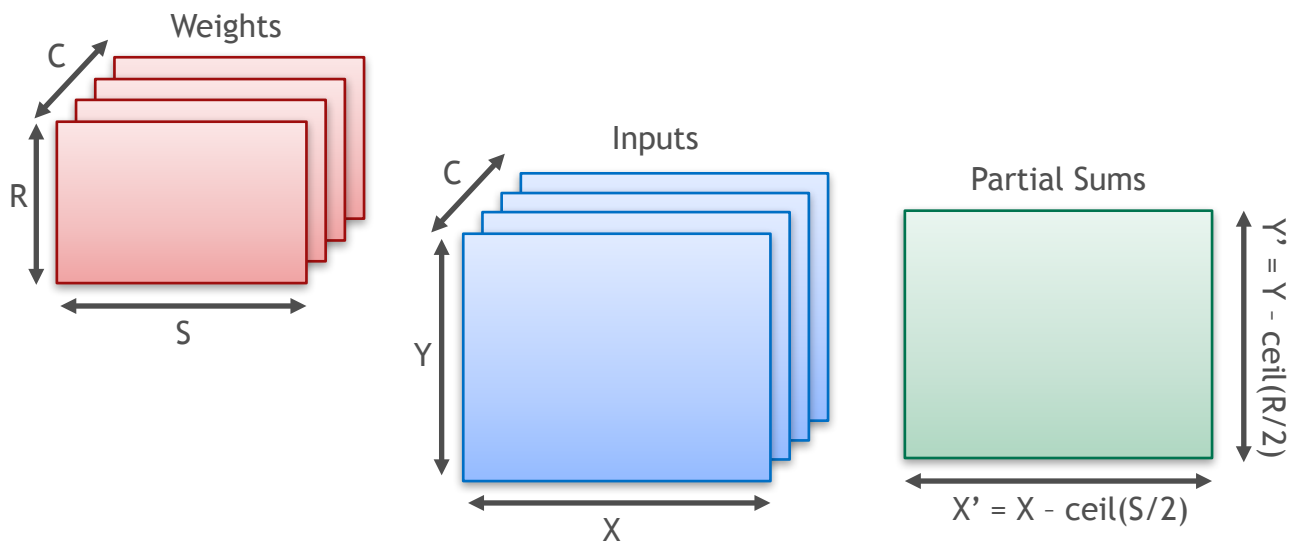
What if hardware cannot do a spatial sum?

Illegal mapping!

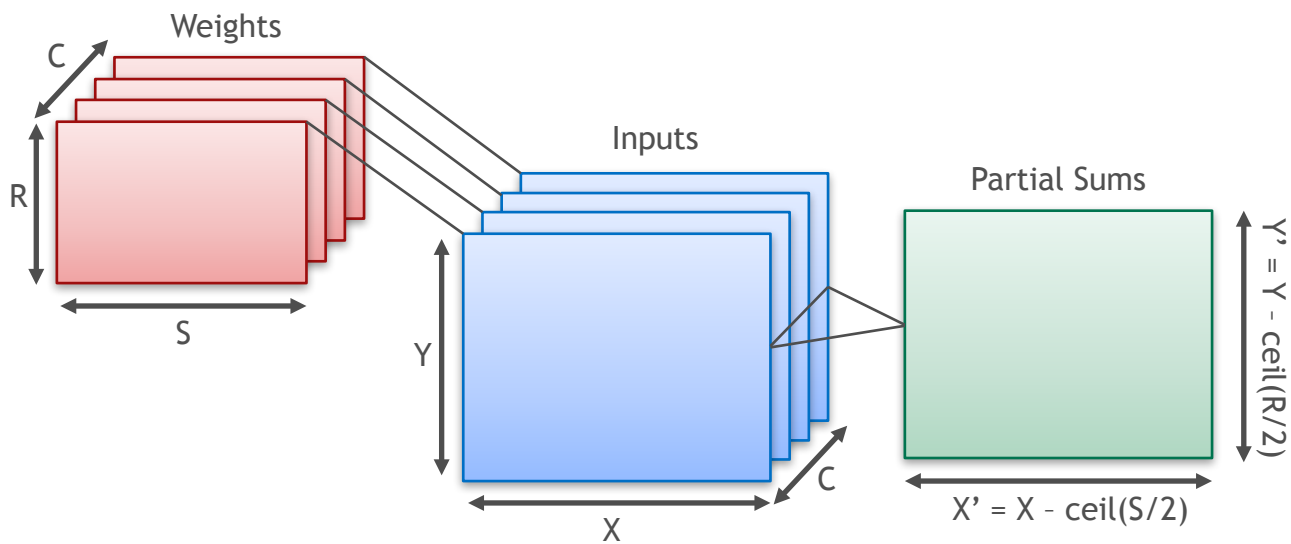
2D CONVOLUTION – ADDING DIMENSIONS



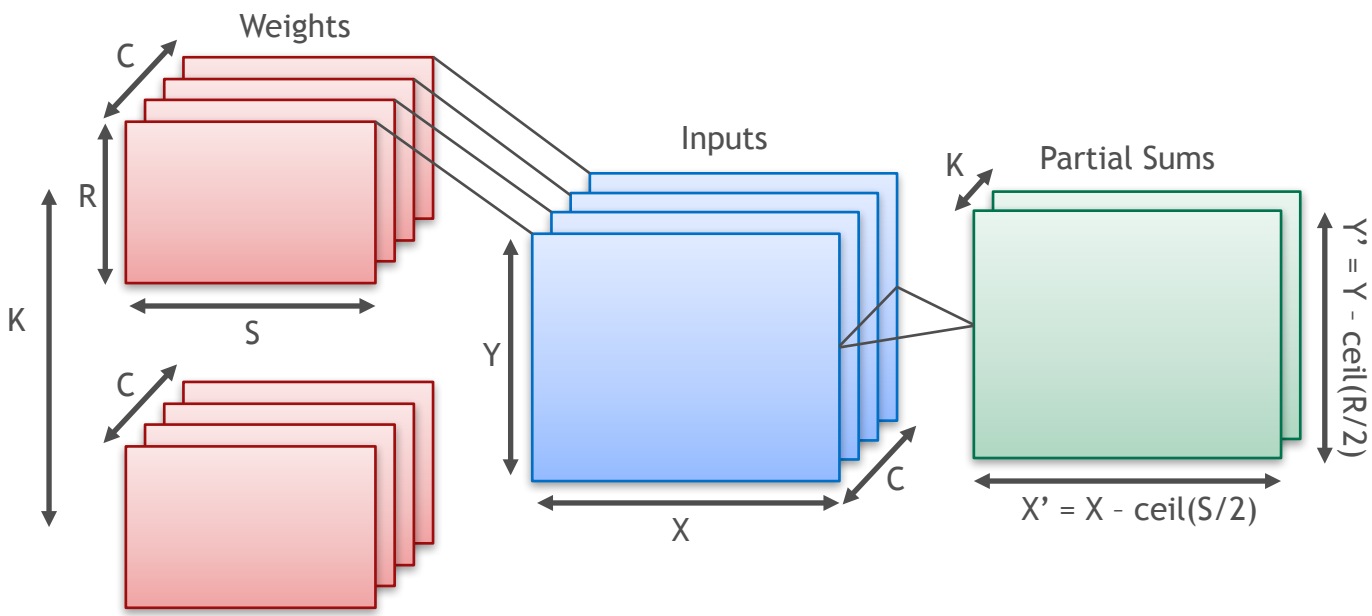
CNN PROBLEM FORMULATION – INPUT CHANNELS



CNN PROBLEM FORMULATION – INPUT CHANNELS



CNN PROBLEM FORMULATION - OUTPUT CHANNELS



CONV LAYER IMPLEMENTATION

Naïve 6-layer* for-loop implementation:

*Batch size would be a 7th loop (if available)

```
for(int ki=0; ki<K; ki++) //Loop_K
 for(int ci=0; ci<C; ci++) //Loop_C
  for(int yi=0; yi<Y; yi++) //Loop_Y
   for(int xi=0; xi<X; xi++) //Loop_X
    for(int ri=0; ri<R; ri++) //Loop_R
     for(int si=0; si<S; si++) //Loop_S
      O[k][x][y] += W[k][c][r][s] * I[c][y+r][x+s]
```

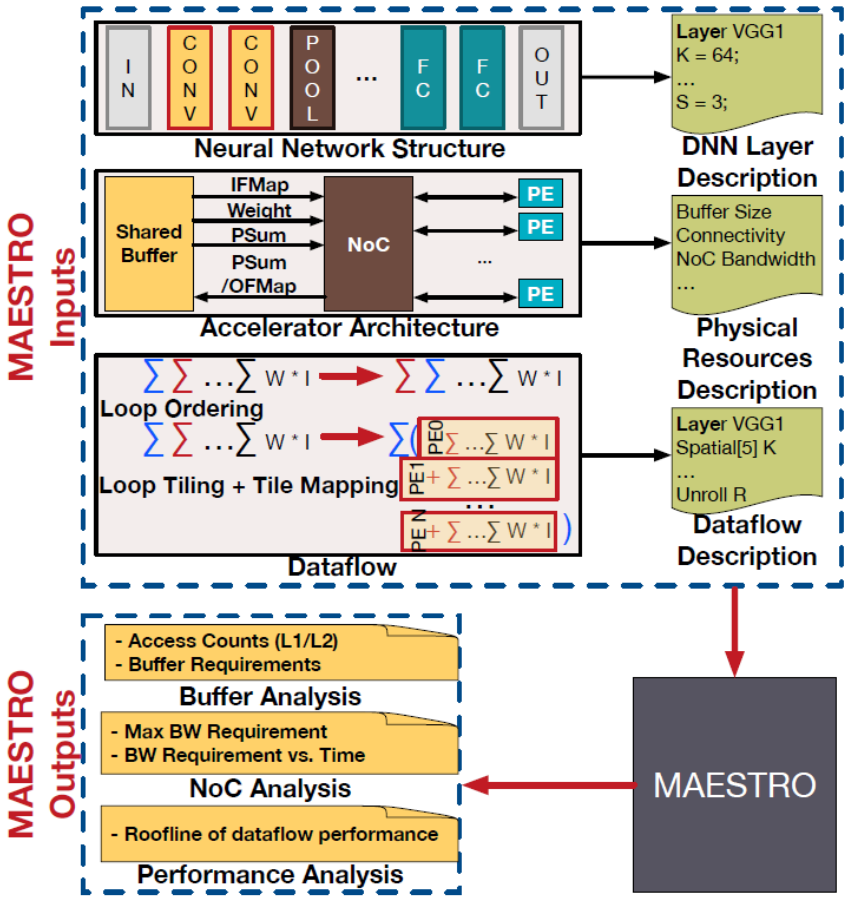
Gigantic space of potential tilings and dataflows – how to explore?

- Cycle-accurate modeling of realistic dimensions and fabric sizes too slow
- Solution: use an analytic model specialized for CNNs

MAESTRO: A DATAFLOW COST MODEL

OVERVIEW OF MAESTRO APPROACH

Modeling Accelerator Efficiency via Spatio-Temporal Resource Occupancy



Analytic model:

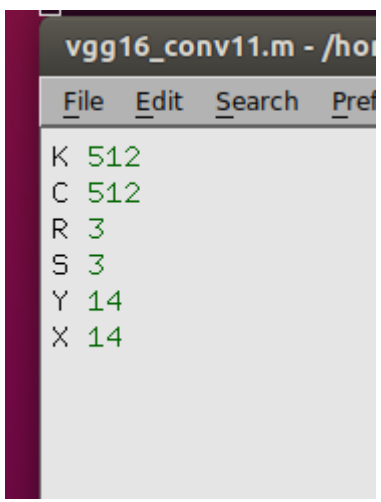
- Does not execute or simulate
- Answers obtained via mathematical equations
- First-cut estimate for focusing later simulation "deep dives"
- Validation is ongoing work

Not a general cost model:

- By specializing workloads and execution substrate we gain efficiency and productivity

ARCHITECTURE AND CNN LAYER DESCRIPTION

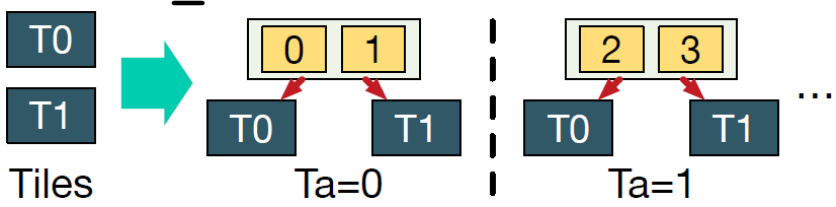
- Architecture assumptions:
  - Spatial array of PEs, each with L1 buffer
  - Shared L2 buffer, via on-chip network
- Model Input Parameters:
  - Number of PEs
  - Vector width
  - On-chip network bandwidth cap
  - Average number of hops
  - Does OCN support multicast?
- Layer:
  - Domain-specific convolutional layer description:



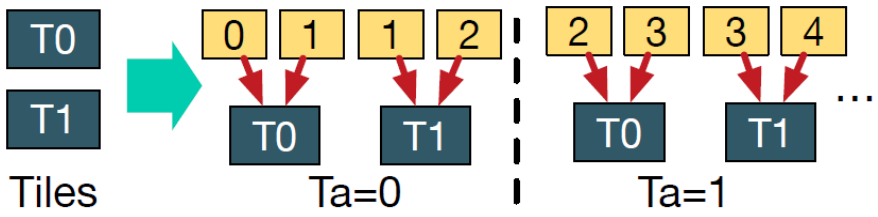
DESCRIBING DATAFLOWS IN MAESTRO

Combine 5 possible pragmas to represent CNN loop nest tiling

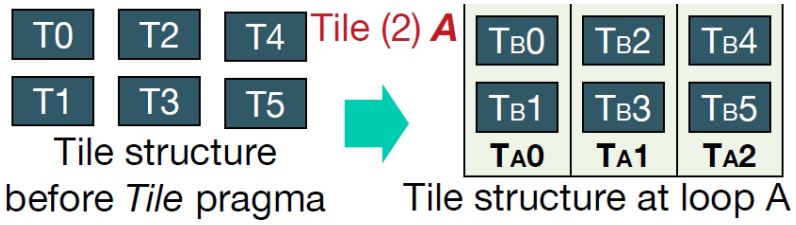
- **`Temporal_map(SIZE, OFFSET) Var`**



- **`Spatial_map(SIZE, OFFSET) Var`**



- **`Tile(SIZE) Var`**



- **`Merge Var`**

```
for(b=0; b<3 ; b++)
 for(a=0; a<2; a++)
```

→

```
for(c=0; c<3*2 ; c++)
 b = c /3; a = c%2;
```

- **`Unroll Var`**

```
for(b=0; b<B; b++)
 for(a=0; a<1; a++)
  f(a,b)
```

→

```
for(b=0; b<B; b++)
 f(0,b); f(1,b)
```

NOTE: all SIZE and OFFSET are in terms of tiles
Var = {K, C, X, Y, R, S}

DATAFLOW EXAMPLES

Our best effort to reproduce the representations in the literature

Accelerator	Dataflow Strategy	Dataflow
Example for this work	No Local Reuse (NLR)	TEMPORAL_MAP (4,4) **K** → TEMPORAL_MAP (1,1) **C** → TEMPORAL_MAP (1,1) **Y** → TEMPORAL_MAP (1,1) **R** → TEMPORAL_MAP (1,1) **S** → SPATIAL_MAP (1,1) **X**
Example for this work	Weight Stationary (WS)	TEMPORAL_MAP (1,1) **K** → TEMPORAL_MAP (1,1) **C** → TEMPORAL_MAP (3,3) **Y** → SPATIAL_MAP (1,1) **X** → UNROLL **R** → UNROLL **S**
ShiDiannao [9]	Output Stationary (OS)	TEMPORAL_MAP (1,1) **K** → TEMPORAL_MAP (1,1) **C** → TEMPORAL_MAP (1,1) **Y** → SPATIAL_MAP (3,1) **X** → UNROLL **R** → UNROLL **S**
Eyeriss [3]	Row-stationary	TEMPORAL_MAP (1,1) **K** → TEMPORAL_MAP (1,1) **C** → TEMPORAL_MAP (3,1) **Y** → SPATIAL_MAP (1,1) **X** → TEMPORAL_MAP (1,1) **R** → TILE (Sz(S)) **S** → TEMPORAL_MAP (SZ(S),Sz(S)) **S**
NVDLA [6]	Weight Stationary	TEMPORAL_MAP (1,1) **R** → TEMPORAL_MAP (1,1) **S** → TEMPORAL_MAP (64,64) **C** → TEMPORAL_MAP (1,1) **Y** → TEMPORAL_MAP (1,1) **X** → SPATIAL_MAP (1,1) **K**

ANALYSIS MODEL

Ignore this “eyechart” slide – Just intuition as to how it works

$$\text{NUMS(WEIGHT)} = Tsz(K) \times Tsz(C) \times Tsz(R) \times Tsz(S)$$

$$\text{NUMS(INPUT)} = Tsz(C) \times Tsz(Y) \times Tsz(X)$$

$$\text{NUMS(SUM)} = Tsz(K) \times Tsz(Y) \times TSz(X) \times TSz(R) \times TSz(S)$$

$$\text{NUMSU(WEIGHT)} = UTSz(K) \times UTsz(C) \times UTsz(R) \times UTsz(S)$$

$$\text{NUMSU(INPUT)} = UTSz(C) \times UTSz(Y) \times UTSz(X)$$

$$\text{NUMSU(SUM)} = UTSz(K) \times UTSz(Y) \times UTSz(X) \times UTSz(R) \times UTSz(S)$$

$$\text{L1 BUFFER REQ} = NumS(Weight) + NumS(Input) + NumS(Sum)$$

$$\text{L2 BUFFER REQ} = (NumSTile - 1) \times (NumSU(Weight) + NumSU(Input) + NumSU(Sum))) + NumS(Weight) + NumS(Input) + NumS(Sum)$$

$$\text{L1 RD WEIGHT} = NumTempFold \times NumSpatialFold \times NumSTile \times NumS(Weight) \times NumS(Input)$$

$$\text{L1 RD INPUT} = NumTempFold \times NumSpatialFold \times NumSTile \times NumS(Weight) \times NumS(Input)$$

$$\text{L1 RD SUM} = NumTempFold \times NumsTile \times (TSz[K] - 1 + (TSz[C] \times (TSz[R] \times TSz[S] - 1) + TSz[C] - 1)$$

$$\text{L1 WR WEIGHT} = NumTempFold \times NumSTile(\times(NumSpatialFold - 1) \times NumSU(Weight) + NumS(Weight)$$

$$\text{L1 WR INPUT} = NumTempFold \times NumSTile(\times(NumSpatialFold - 1) \times NumSU(Input) + NumS(Input)$$

$$\text{L1 WR SUM} = NumTempFold \times NumSTile(\times(NumSpatialFold - 1) \times NumSU(Sum) + NumS(Sum)$$

$$\text{L2 RD WEIGHT} = NumTempFold \times NumSTile \times (NumSU(Weight) + (NumSpatialFold - 1) \times NumSU(Weight))$$

$$\text{L2 RD INPUT} = NumTempFold \times NumSTile \times (NumSU(Input) + (NumSpatialFold - 1) \times NumSU(Input))$$

$$\text{L2 WR WEIGHT} = Sz(K) \times Sz(C) \times Sz(R) \times Sz(S)$$

$$\text{L2 WR INPUT} = Sz(C) \times Sz(Y) \times Sz(X)$$

$$\text{L2 WR SUM} = Sz(K) \times Sz(Y) \times Sz(X)$$

$$\text{(L2}\rightarrow\text{L1)NOCBW} = (NumSTile - 1) \times (NumSU(Wegihu) + NumSU(Input)) + NumS(Weight) + NumS(Input)$$

$$\text{(L1}\rightarrow\text{L2)NOCBW} = NumSTile \times NumSU(Sum)$$

$$\text{ROOFLINE_TH} = NumTotalPartialSums / (NumTempFold \times NumSpatialFold \times NumSTile \times NumS(Input) \times NumS(Weight)$$

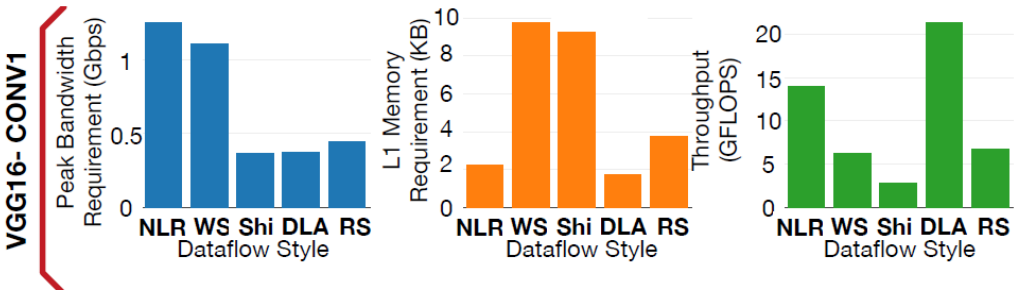
Definitions

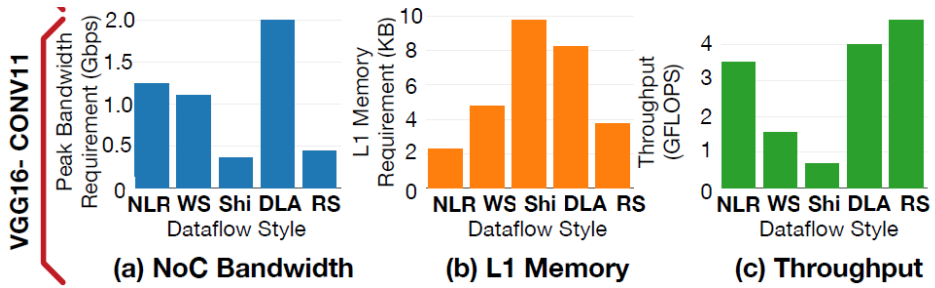
S-tile: Spatial tile; tiles at a the innermost spatially mapped loop
NumSTile: Total number of S-tiles
NumTempFold: Number of iterations of the entire innermost loop with spatial_map
NumSpatialFold: Number of implicit folding (due to insufficient number of tiles for a spatial mapping) within the innermost loop with spatial_map
NumS(Dataclass): Number of data points of the data class accessed in a S-tile
NumSU(Dataclass): Number of *unique* data points of the data class accessed in a S-tile
Sz(Var): The entire size of the Var dimension in the given neural network layer
TSz(Var): Number of assigned variable Var within a S-tile
UTSz(Var): Number of assigned *unique* variable Var within a S-tile (reuse considered)

$$\text{TSZ(VAR)} = \textbf{match}\ \text{getPragma(var)}$$
$$|\text{SPATIAL_MAP(Sz,Ofs), TEMPORAL_MAP(Sz,Ofs)} : Sz$$
$$|Unroll : \text{getUpperBound(var)}$$
$$|Merge : \text{getSize(var)} / \text{getSize(getUpLoopVar(var))}$$

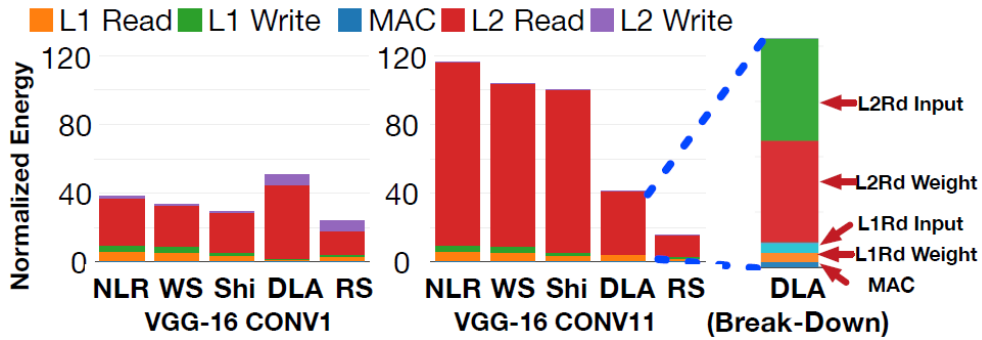
$$\text{UTSZ(VAR)} = \textbf{match}\ \text{getPragma(var)}$$
$$|\text{SPATIAL_MAP(Sz,Ofs), TEMPORAL_MAP(Sz,Ofs)} : Sz - Ofs + 1$$
$$|\text{default} : TSz(var);$$

EXAMPLES USING MAESTRO OUTPUT





Combining with an energy model:



NOTE: this represents the performance of the *dataflow* on normalized PE substrate

- Not representative of the performance of the original architecture

DEMO

MAESTRO command-line parameters:

```
./maestro <layer>.m <dataflow>.m <vector_width> <max_bandwidth <support_multicast?> <avg_hops> <num_pes>
```

Pre-provided layers:

```
pwc_fpe_conv1.m  pwc_fpe_conv6.m  vgg16_conv1.m  vgg16_conv4.m  vgg16_conv9.m
pwc_fpe_conv4.m  vgg16_conv11.m   vgg16_conv2.m  vgg16_conv6.m
```

Pre-provided dataflows:

```
bal.m  dla.m  nlr.m  nsd.m  prs.m  rs.m  sd.m  ws2.m  ws.m
```

Output example:

```
<< Buffer Analysis >>
L1BufferReq: 66.00Byte
L2BufferReq: 2.06KB
L1RdCount: 59305623552
  L1RdWeight: 29595009024
  L1RdInput: 29595009024
  L1RdSum: 115605504
L1WrCount: 3814981632
  L1WrWeight: 1849688064
  L1WrInput: 1849688064
  L1WrSum: 115605504
```

```
<<NoC Analysis>>
L1 to L2 NoCBW: 16 elements per cycle
L2 to L1 NocBW: 512 elements per cycle
<< Performance Analysis  >>
Roofline throughput: 1.000000 GFLOPS with 1GHz clock
L1 to L2 sum: 16.000000
L2 to L1 delay: 0.000000
L1 to L2 delay: 10.000000
Compute delay: 426.000000
total latency: 3150249984 cycles
```

MAESTRO HANDS-ON LAB

(Don't forget about the 10:00-10:30 coffee break!)

Exercise #1: Sweep throughput as a function of the number of PEs 1-256 (doubling)

- e.g. `./maestro layerDescription/vgg16_conv2.m dataflowDescription/dla.m 1 8 1 8 <N>`
- At what point do you see diminishing returns? Why? Can you fix it?

Exercise #2: Sweep numeric parameter sizes of a dataflow for a fixed architecture

- e.g. **`tmap (4, 4) K`** => **`tmap (8, 8) K`** => **`tmap (16, 16) K`**, etc.
- What effect does this have on results? Any differences for early/late layers?

Exercise #3: Sweep dataflows for a fixed layer and architecture

- e.g. `./maestro layerDescription/vgg16_conv2.m <dataflow>.m 1 8 1 8 64`
- Which dataflow has min/max reads for L1 weights/inputs/sums? Min/max transfers?

Challenge: Design a new dataflow with the following properties:

- No more than 16 PEs, bandwidth 4, L2 to L1 BW < 0.04 per cycle, 1 sMap and 5 tMaps
- Better than all provided dataflows on late layers, but can be worse on early layers

CONCLUSIONS

Understanding CNN dataflow and tiling concepts is critical for computer architecture researchers focusing on domain-specific accelerators

Caches are generally considered too expensive (area+power) but we can easily make up the difference using workload knowledge

For an example of exploiting advanced dataflow knowledge, come see:

UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition - Kartik Hegde (UIUC), Jiyong Yu (UIUC), Rohit Agrawal (UIUC), Mengjia Yan (UIUC), Michael Pellauer (NVIDIA), Christopher Fletcher (UIUC)

Session 8A: Machine Learning Systems 1 (Wednesday, 9:30 AM)