Accelerators-II

Tushar Krishna  
Associate Professor @ Georgia Tech  
Visiting Professor @ MIT EECS and CSAIL

**Slide Acknowledgments:** Michael Pellauer, Angshuman Parashar, Joel Emer, Vivienne Sze, Hyoukjun Kwon, Felix Kao
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
• Advanced Dataflows
Outline

- Recap
- Dataflows for 1D Convolution
- Getting more realistic
- Advanced Dataflows
Recap

• Why domain-specific accelerators?
  – High Throughput requirements (workload constraint)
  – Energy costs of Data Movement (technology constraint)

• Why do accelerators help?
  – custom datapaths for the operator(s) of interest (e.g., matrix multiplication)
  – remove control overheads that Turing-complete engines (e.g., CPUs) have such as instruction fetch/decode, speculation, caches, ..
Accelerators

Off-Chip Memory

Custom Datapath

Global Buffer (100 – 500 kB)

DRAM
Why does this matter?

Attainable Performance (GFLOPS)

Floating Point Ops / Second

Peak Compute Performance (Depends on number of PEs)

Memory BW

Compute bound region

Mem bound region

FLOPs/Byte

Floating Point Ops / Byte

Energy Overheads

Normalized Energy Cost

1x (Reference)

1x

2x

6x

200x

December 6, 2023
How to reduce BW requirement?

<table>
<thead>
<tr>
<th>VGG16 conv 3_2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiply Add Ops</td>
<td>1.85 Billion</td>
</tr>
<tr>
<td>Weights</td>
<td>590 K</td>
</tr>
<tr>
<td>Inputs</td>
<td>803 K</td>
</tr>
<tr>
<td>Outputs</td>
<td>803 K</td>
</tr>
</tbody>
</table>

Data Reuse

How to exploit reuse? “Dataflow”

i.e., fine-grained schedule of computations within DNN accelerators

• Computation Order
• Parallelization Strategy
Dataflow Implication: Algorithm Reuse $\rightarrow$ HW Reuse

7-dimensional network layer

- **7D Computation Space**
  - $R \times S \times X \times Y \times C \times K \times N$

- **4D Operand/Result Data Spaces**
  - Weights - $R \times S \times C \times K$
  - Inputs - $X \times Y \times C \times N$
  - Outputs - $P \times Q \times K \times N$

2D hardware array

- **HW Design-space**
  - Number of PEs
  - Memory Hierarchy (Sizes and Bandwidths)
  - Interconnect Bandwidth

- **HW Reuse Structures**

  **Temporal**
  - DRAM
  - Buf
  - RF
  - $*$

  **Spatial (Multicast)**
  - Multicast

  **Spatial (Forwarding)**
  - Forwarding

How to describe and navigate?
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
• Advanced Dataflows
Output Stationary (OS) Dataflow

Computation

\[
\text{for} (\text{int } x = 0; x < X'; x++) \\
\text{for} (\text{int } s = 0; s < S; s++) \\
\text{Output}[x] += \text{Weight}[s] \times \text{Input}[x+s]
\]

PE0
PE1
PE2

\[X' = X - S\]

Data

PartialSum[X'][S] needs to access:
- Weight[s]
- Output[x']
- Input[x'+s]

Spatial multicast opportunity for weights

Output does not change over time => Temporal reuse opportunity

Computation Space

Data Space

Each point is a partial sum

Each point is a data access

December 6, 2023
Describing OS dataflow

Weights $S$ * Inputs $X$ = Outputs* $X' = X - S$

```c
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 $S$ 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

- 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, may have harmful “interference” (bursts)
    - Intermediate staging buffers reduce both bandwidth and energy
- Often corresponds to datatype that is “done with” earliest without further reloads
- **Note**: the “stationary” name is meant to give intuition, not to be a complete specification of all the behavior of a 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];
    }
}
```

- How many times will $x = 2$?
- How many times will $x+s = 2$?
- How many times will $s = 2$?

- Temporal distance between re-occurrence dictates buffer size to avoid re-load.
- How do you know if a buffer that size is worth it?
OS Dataflow Implementation

- Minimize partial sum R/W energy consumption
  - maximize local accumulation

- Broadcast/Multicast filter weights and reuse activations spatially across the PE array
Weight Stationary (WS) Dataflow

Computation

```c
for(int s = 0; s < S; s++)
    for(int x = 0; x < X'; x++)
        Output[x] += Weight[s] * Input[x+s]
```

Data

- PartialSum[X'][S] needs to access:
  - Weight[s]
  - Output[x']
  - Input[x'+s]

Computation Space

- Weight does not change over time => Temporal reuse opportunity
- Need Spatial reduction for output

Time = 0

Data Space

Each point is a partial sum

Need Spatial reduction for output
Describing WS 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 about the loop nest makes it weight stationary?

*outermost loop is S rank*
• **Minimize** weight read energy consumption
  – maximize convolutional and filter reuse of weights

• **Broadcast** activations and **accumulate** psums spatially across the PE array.
Simple Model for Mapping Dataflows to HW

\[ X' = X - \text{ceil}(S/2) \]

<table>
<thead>
<tr>
<th>Common metric</th>
<th>Weights</th>
<th>Inputs</th>
<th>Outputs / Partial Sums</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg. Min. accesses to backing store (MINALG)</td>
<td>S</td>
<td>X</td>
<td>X'</td>
</tr>
<tr>
<td>Maximum operand uses (MAXOP)</td>
<td>SX'</td>
<td>SX'</td>
<td>SX'</td>
</tr>
</tbody>
</table>
# 1D Convolution Summary

<table>
<thead>
<tr>
<th>Hardware Structure</th>
<th>Per Data Type</th>
<th>OS Dataflow Implication</th>
<th>WS Dataflow Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bandwidth to MAC</strong></td>
<td>Weight Fetch Rate</td>
<td>Every Cycle</td>
<td>Every S Cycles</td>
</tr>
<tr>
<td></td>
<td>Input Fetch Rate</td>
<td>Every Cycle</td>
<td>Every Cycle</td>
</tr>
<tr>
<td></td>
<td>Output Fetch Rate</td>
<td>Every S Cycles</td>
<td>Every Cycle</td>
</tr>
<tr>
<td><strong>Local Buffer Sizes for Temporal Reuse</strong></td>
<td>Weight Buffer Size</td>
<td>S</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Input Buffer Size</td>
<td>S</td>
<td>X’</td>
</tr>
<tr>
<td></td>
<td>Output Buffer Size</td>
<td>1</td>
<td>X’</td>
</tr>
<tr>
<td><strong>Total Local Buffer Accesses</strong></td>
<td>Weight Buffer</td>
<td>X’</td>
<td>SX’</td>
</tr>
<tr>
<td></td>
<td>Input Buffer</td>
<td>X’</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>Output Buffer</td>
<td>SX’</td>
<td>S</td>
</tr>
</tbody>
</table>
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
  – Multi-layer Buffering
  – Multiple PEs
  – Full Convolution
• Advanced Dataflows
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
  – Multi-layer Buffering
  – Multiple PEs
  – Full Convolution
• Advanced Dataflows
Multi-layer Buffering

How will this be reflected in the loop nest?  
New ‘level’ of loops
1D Convolution – “Tiled”

Weights * Inputs = Outputs

\[ X' = X - \text{ceil}(X/2) \]

```c
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:
\[ X'0 \times X'1 = X' \]
\[ S0 \times S1 = S \]
Buffer sizes

- Level 0 buffer size is volume needed in each Level 1 iteration.
- Level 1 buffer size is volume needed to be preserved and re-delivered in future (usually successive) Level 1 iterations.

- A **legal mapping** will fit into the hardware’s buffer sizes
Buffer sizes

// 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

<table>
<thead>
<tr>
<th></th>
<th>Level 0</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
<td>S0</td>
<td>S</td>
</tr>
<tr>
<td>Inputs</td>
<td>X’0+S0</td>
<td>S</td>
</tr>
<tr>
<td>Outputs</td>
<td>X’0</td>
<td>1</td>
</tr>
</tbody>
</table>
Energy Costs

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

Each buffer level’s energy is proportional to 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
// 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
// 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];
            }
        }
    }
}

s1=0
Inputs
X'0+S0

s1++
S0
X'0+S0

s1++
S0
X'0+S0

Next x1 iteration
Input halo!

Sliding window
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
// 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];
            }
        }
    }
}

Outputs

s1=0

X’0

s1++

X’0

Next x1 iteration

s1=0

X’0
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];
            }
        }
    }
}

<table>
<thead>
<tr>
<th></th>
<th>Level 0</th>
<th>Level 1 to 0 transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight Reads</td>
<td>SX’</td>
<td>SX’1</td>
</tr>
<tr>
<td>Input Reads</td>
<td>X’ * S1+ X’1 * S</td>
<td>X’+X’1*S</td>
</tr>
<tr>
<td>Output Reads</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Output Writes</td>
<td>X’ * S1</td>
<td>X’</td>
</tr>
</tbody>
</table>
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
  – Multi-layer Buffering
  – Multiple PEs
  – Full Convolution
• Advanced Dataflows
Spatial PEs

How will this be reflected in the loop nest? New ‘level’ of loops
1D Convolution – Partition Outputs

Weights | Inputs | Outputs
---------|--------|--------
S | X | X’ = X – ceil(S/2)

\[
\text{int } i[X]; \quad \# \text{ Input activations}
\text{int } w[S]; \quad \# \text{ Filter Weights}
\text{int } o[X’]; \quad \# \text{ Output activations}
\]

// 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
1D Convolution – Partition Outputs

Weights
S

* Inputs
X

= Outputs
X' = X - ceil(S/2)

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'; x0++) {
    for (s0 = 0; s0 < S; s0++) {
      o[x1*X'+x0] += i[x1*X'+x0 + s1*S+s0] * w[s1*S+s0];
    }
  }
}
Spatial PEs

**Implementation opportunity?**

Yes, single fetch and multicast
1D Convolution – Partition Outputs

How do we recognize multicast opportunities?

Indices independent of spatial index
Spatial PEs: Partitioned Outputs

L0 Weights
L0 Inputs
L0 Outputs

PE0

PE1

Implementation opportunity?

Parallel fetch

Assuming S=3

December 6, 2023
1D Convolution – Partition Weights

Weights \( \times \) Inputs = Outputs

\[
\begin{align*}
\text{int } i[X]; & \quad \# \text{ Input activations} \\
\text{int } w[S]; & \quad \# \text{ Filter Weights} \\
\text{int } o[X']; & \quad \# \text{ Output activations}
\end{align*}
\]

// Level 0
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:
\[X'0*X'1 = X' \quad S0*S1 = S\]
1D Convolution – Partition Weights

// 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 reduction

Other multicast opportunities? No
Spatial PEs: Partitioned Weights

Spatial sum needed? Yes

Assuming S=3
Spatial PEs with Spatial Summation

What if hardware cannot do a spatial sum?

Illegal mapping!
# NoC Support

<table>
<thead>
<tr>
<th>Hardware Structure</th>
<th>Per Data Type</th>
<th>Output-partitioned Dataflow Implication</th>
<th>Weight-partitioned Dataflow Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network-on-Chip for Spatial Reuse</strong></td>
<td>Weight Distribution</td>
<td>Spatial Multicast</td>
<td>Unicast</td>
</tr>
<tr>
<td></td>
<td>Input Distribution</td>
<td>Unicast/Spatial Multicast</td>
<td>Unicast</td>
</tr>
<tr>
<td></td>
<td>Output Collection</td>
<td>Temporal Reduction</td>
<td>Spatial Reduction</td>
</tr>
</tbody>
</table>
More Realistic Loop Nest

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

// Level 2
for (x2 = 0; x2 < X’2; x2++) {
    for (s2 = 0; s2 < S2; s2++) {
        // 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++) {
                        ...
                    }
                }
            }
        }
    }
}
```

December 6, 2023
Design-space of a DNN Accelerator

Mapping

HW Design-Space

Dataflow

Ordering (aka “stationary”)
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
  – Multi-layer Buffering
  – Multiple PEs
  – Full Convolution
• Advanced Dataflows
Mapping a Full Convolution

Weights

Inputs

Partial Sums

\[ X' = X - S \]

\[ Y' = Y - R \]
Reference Convolution Layer

```c
int i[C][Y][X];  // Input activation channels
int w[K][C][R][S];  // Filter weights (per channel pair)
int o[K][Y'][X'];  // Output activation channels

for (k = 0; k < K; k++) {
    for (y = 0; y < Y'; y++) {
        for (x = 0; x < X'; x++) {
            for (c = 0; c < C; c++) {
                for (r = 0; r < R; r++) {
                    for (s = 0; s < S; s++) {
                        o[k][y][x] += i[c][y+r][x+s] * w[k][c][r][s];
                    }
                }
            }
        }
    }
}
```
Describing a full accelerator

```c
// DRAM levels
for (g3=0; g3<G3; g3++) {
    for (n3=0; n3<N3; n3++) {
        for (m3=0; m3<M3; m3++) {
            for (f3=0; f3<F3; f3++) {
                // Global buffer levels
                for (g2=0; g2<G2; g2++) {
                    for (n2=0; n2<N2; n2++) {
                        for (m2=0; m2<M2; m2++) {
                            for (c2=0; c2<C2; c2++) {
                                for (s2=0; s2<S2; s2++) {
                                    // NoC levels
                                    parallel-for (g1=0; g1<G1; g1++) {
                                        parallel-for (n1=0; n1<N1; n1++) {
                                            parallel-for (m1=0; m1<M1; m1++) {
                                                parallel-for (f1=0; f1<F1; f1++) {
                                                    parallel-for (c1=0; c1<C1; c1++) {
                                                        parallel-for (s1=0; s1<S1; s1++) {
                                                            // SPad levels
                                                            for (f0=0; f0<F0; f0++) {
                                                                for (n0=0; n0<N0; n0++) {
                                                                    for (e0=0; e0<E0; e0++) {
                                                                        for (r0=0; r0<R0; r0++) {
                                                                            for (c0=0; c0<C0; c0++) {
                                                                                for (m0=0; m0<M0; m0++) {
                                                                                    o += I * W;
                                                                                }
                                                                            }
                                                                        }
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

December 6, 2023

MIT 6.5900 (ne 6.823) Fall 2023
A Mapping Representation

• For each temporal and spatial level:
  – Permutation order of all indices
  – Partitioning of data volume for all indices (factoring)
  – Data bypass flag per datatype (for temporal layers)

\[ \forall X \in \text{indices} \left( \prod_{l=0}^{\text{max level}} X_l \right) \geq X_{\text{total}} \]

\[
(N_l \ K_l \ C_l \ X'_l \ Y'_l \ R_l \ S_l) \ [I_l, W_l, O_l]
\]

How many permutations per level? \((# \ \text{Indices})!\)

How many bypass combinations per level? \(2^N\)

Total choices per temporal level? \((# \ \text{Indices})! * 2^N * \text{factorings}\)
Impact of Mappings

480,000 mappings shown

Spread: 19x in energy efficiency

Only 1 is optimal, 9 others within 1%

6,582 mappings have min. DRAM accesses but vary 11x in energy efficiency

VGG conv3 2 Layer. Source: Timeloop

1-level par. + 2-level par. + 3-level par.

O(10^{12}) + O(10^{24}) + O(10^{36})

Immense Search space
Exploring Mappings

- Gigantic space of potential loop orders & factorizations
- Cycle-accurate modeling of realistic dimensions and fabric sizes too slow
- Solution: use an analytic modeling

```c
int i[C][Y][X];  // Input activation channels
int w[K][C][R][S]; // Filter weights (per channel pair)
int o[K][Y'][X']; // Output activation channels

for (k = 0; k < K; k++) {
    for (y = 0; y < Y'; y++) {
        for (x = 0; x < X'; x++) {
            for (c = 0; c < C; c++) {
                for (r = 0; r < R; r++) {
                    for (s = 0; s < S; s++) {
                        o[k][y][x] += i[c][y+r][x+s] * w[k][c][r][s];
                    }
                }
            }
        }
    }
}
```

e.g.,: Timeloop (ISPASS 2019), MAESTRO (MICRO 2019), ..
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
• Advanced Dataflows
  – Fusion
  – Sparsity
Outline

• Recap
• Dataflows for 1D Convolution
• Getting more realistic
• Advanced Dataflows
  – Fusion
  – Sparsity
Not All GEMMs are Compute Bound

Even in the best case with infinite on-chip storage and large number of PEs,

\[ AI_{best\_GEMM} = \frac{M \times K \times N}{M \times K + K \times N + M \times N} \]

**Regular GEMM (M=1024, K=1024, K=1024)**

\[ AI_{best\_GEMM} = 341.33 \text{ ops/word} \]

**Skewed GEMM (M=1048576, N=32, K=32)**

\[ AI_{best\_GEMM} = 16 \text{ ops/word} \]
GEMMs in Attention Layers

Compute-bound $\Rightarrow$ Intra-operator dataflow to improve reuse is effective

Memory-bound $\Rightarrow$ Intra-operator dataflow is not effective

Activation – Activation GEMMs
Opportunity: Fusion

- Key Intuition: “Reuse” the intermediate output immediately

---

Kao et al, “FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks”, ASPLOS 2023

December 6, 2023
Opportunity: Fusion

Kao et al, “FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks”, ASPLOS 2023

December 6, 2023
Outline

- Recap
- Dataflows for 1D Convolution
- Getting more realistic
- Advanced Dataflows
  - Fusion
  - Sparsity
Sparsity in DNNs

Model Sparsity (per model)
- Weights
  - unstructured (fine-grained)
  - structured (blocked)
- Neurons
  - Neuron-like (Filters/Channels/Heads)
  - structured sparsity
- Affects inference + forward pass

Ephemeral Sparsity (per example)
- Dropout (Activations/Weights)
  - Gradient-based optimization
- Errors $e_1$
- Optimizer State
- Conditional computation
  - Route each example through a different sparse subnetwork

Source: Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks

Figure source: [https://htor.inf.ethz.ch/sparsity-in-dl/](https://htor.inf.ethz.ch/sparsity-in-dl/)

10-90% sparsity across ML Models today
Sparsity Patterns

DENSE  Block Balanced (Eg: N:M)  Unstructured

1D Blocks  2D Blocks
Sparse Accelerators

Trade-off Space

Dense

2:4 Structured Sparsity

Unstructured

Higher Regularity in Data
Lower Potential Speed-up
Lower Overhead for Sparsity Support

Lower Regularity in Data
Higher Potential Speed-up
Higher Overhead for Sparsity Support

Input Buffers

Weight Buffers

Output Buffers

Distribution Network

Collection Network

Flexible NoC

Flexible Buffers
Sparse Dataflows

- Inner Product
- Outer Product
- Gustavson’s

Active area of research!
Thank you!