DNN Dataflow Choice Is Overrated
Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qioyi Liu, Steven Emberton
Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark
Horowitz.
Stanford University, Tsinghua University
2018

Presenter: Derrick Blakely

https://qdata.github.io/deep2Read
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion
Systolic Arrays - Dataflow Analogous to Blood Circulation

Presented by: Derrick Blakely (University of Virginia)

Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qioyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. Stanford University, Tsinghua University 2018

https://qdata.github.io/deep2Read
Convolution - A Use Case for Systolic Arrays

\[ w_1 x_1 + w_2 x_2 + w_3 x_3 \]

Presenter: Derrick Blakely (University of Virginia)
Convolution - A Use Case for Systolic Arrays

\[ w_1 x_1 + w_2 x_2 + w_3 x_3 \]

\[ w_1 x_2 + w_2 x_3 + w_3 x_4 \]
Convolution - A Use Case for Systolic Arrays

\[ w_1 x_1 + w_2 x_2 + w_3 x \]
\[ w_1 x_3 + w_2 x_4 + w_3 x \]
\[ w_1 x_2 + w_2 x_3 + w_3 x \]
1D Systolic Convolution

Data Pool

Reads: 0
Writes: 0
1D Systolic Convolution

Data Pool

$w_3 \ w_2 \ w_1$
$x_2 \ x_3 \ x_4 \ x_5$

$y_3 \ y_2 \ y_1$

$w_3 \ w_2 \ w_1$

$x_1$

Reads: 1
Writes: 0
1D Systolic Convolution

Data Pool

Reads: 3
Writes: 0

W3
W2
W1

x3 x4 x5
y3 y2

x2
y1
x1

Presenter: Derrick Blakely (University of Virginia)

DNN Dataflow Choice Is Overrated

Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qioyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. Stanford University, Tsinghua University 2018

https://qdata.github.io/deep2Read
1D Systolic Convolution

Reads: 3
Writes: 0

Presenter: Derrick Blakely (University of Virginia)
1D Systolic Convolution

Data Pool

$y_1 = w_1 x_1 + w_2 x_2$

Reads: 3
Writes: 0

$x_3, x_4, x_5$

$y_3, y_2$

$x_2$

$w_3, w_2, w_1$
1D Systolic Convolution

\[ y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3 \]

\[ y_2 \]

\[ y_3 \]

Reads: 5
Writes: 0
1D Systolic Convolution

\[ y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3 \]

Reads: 6
Writes: 1

Presenter: Derrick Blakely (University of Virginia)

DNN Dataflow Choice Is Overrated

Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qioyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. Stanford University, Tsinghua University 2018

https://qdata.github.io/deep2Read
Why is this good?

- Avoids the von Neumann Bottleneck: number of memory accesses proportional to the number of inputs, not the number of computations
- Once finished: 8 reads, 3 writes
- Compare with SIMD: 18 reads, 3 writes
Why is this good?

- Avoids the von Neumann Bottleneck: number of memory accesses proportional to the number of inputs, not the number of computations
- Once finished: 8 reads, 3 writes
- Compare with SIMD: 18 reads, 3 writes
Why is this good?

- Avoids the von Neumann Bottleneck: number of memory accesses proportional to the number of inputs, not the number of computations
- Once finished: 8 reads, 3 writes
- Compare with SIMD: 18 reads, 3 writes
1D SIMD Convolution

Data Pool

Instructions

Reads: 0
Writes: 0
1D SIMD Convolution

Data Pool

Instructions

Reads: 6
Writes: 0

$w_1 x_1$

$w_1 x_2$

$w_1 x_3$
1D SIMD Convolution

Instructions

Data Pool

- $w_{2x2}$
- $w_{2x3}$
- $w_{2x4}$

Reads: 12
Writes: 0
1D SIMD Convolution

Data Pool

Instructions

Reads: 18
Writes: 3

$w_{3 \times 3}$

$w_{3 \times 4}$

$w_{3 \times 5}$
CNN Accelerator Overview
Want to:
- Maximize data reuse
- Minimize memory latency
- Minimize energy use
- Multiple things can be reused - balance reuse opportunities
Weight Stationary

Weight

Global Buffer

Psum
Output Stationary

Global Buffer

Activation → Weight

P0 → P1 → P2 → P3 → P4 → P5 → P6 → P7 → PE

Psum
Row Stationary (Eyeriss)
Row Stationary (Eyeriss)
Row Stationary (Eyeriss)
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion
Motivation

- Images are ubiquitous
- Optimizing image processing code yields high ROI
- Hard to do, creates unreadable code
Motivation

- Images are ubiquitous
- Optimizing image processing code yields high ROI
- Hard to do, creates unreadable code
Motivation

- Images are ubiquitous
- Optimizing image processing code yields high ROI
- Hard to do, creates unreadable code
Motivation

```c
void box_filter_3x3(const Image &in, Image &blurx) {
    Image blurx(in.width(), in.height()); // allocate blurx array

    for (int y = 0; y < in.height(); y++)
        for (int x = 0; x < in.width(); x++)
            blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;

    for (int y = 0; y < in.height(); y++)
        for (int x = 0; x < in.width(); x++)
            blurx(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;
}
```
Motivation

3x3 blur as a Halide algorithm:

```plaintext
Var x, y; Func blurx, blury;
blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;
```

Presenter: Derrick Blakely (University of Virginia)
Halide

- A language/compiler for image processing
- Key idea: decouple algorithm from computation schedule
- Algorithm: what is computed
- Schedule: where/when it’s computed
- Key idea: Don’t mix algorithm design with scheduling
- Allow compiler to safely optimize and schedule algorithm
Halide

- A language/compiler for image processing
- Key idea: decouple algorithm from computation schedule
  - Algorithm: what is computed
  - Schedule: where/when it’s computed
- Key idea: Don’t mix algorithm design with scheduling
- Allow compiler to safely optimize and schedule algorithm
Halide

- A language/compiler for image processing
- Key idea: decouple algorithm from computation schedule
- Algorithm: what is computed
- Schedule: where/when it’s computed
- Key idea: Don’t mix algorithm design with scheduling
- Allow compiler to safely optimize and schedule algorithm
A language/compiler for image processing
Key idea: decouple algorithm from computation schedule
Algorithm: what is computed
Schedule: where/when it’s computed
Key idea: Don’t mix algorithm design with scheduling
Allow compiler to safely optimize and schedule algorithm
Halide

- A language/compiler for image processing
- Key idea: decouple algorithm from computation schedule
- Algorithm: what is computed
- Schedule: where/when it’s computed
- Key idea: Don’t mix algorithm design with scheduling
- Allow compiler to safely optimize and schedule algorithm
Halide

- A language/compiler for image processing
- Key idea: decouple algorithm from computation schedule
- Algorithm: what is computed
- Schedule: where/when it’s computed
- Key idea: Don’t mix algorithm design with scheduling
- Allow compiler to safely optimize and schedule algorithm
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion
Proposal: A 3-Dimensional Design Space

1. Loop optimizations
2. Dataflow
3. Hardware resource allocation
Dimension 1: Loop Optimizations

A Single Convolutional Layer:

- input feature maps of size $X \times Y$
- $C$ channels
- $K$ filters of size $C \times F_X \times F_Y$
- Batch size of $B$

$$O[b][k][x][y] = \sum_{c=0}^{C-1} \sum_{f_y=0}^{F_Y-1} \sum_{f_x=0}^{F_X-1} l[b][c][x + f_x][y + f_y] \times W[k][c][f_x][f_y]$$
Dimension 1: Loop Optimizations

for $b = 0 : B$ do
  for $k = 0 : K$ do
    for $c = 0 : C$ do
      for $y = 0 : Y$ do
        for $x = 0 : X$ do
          for $f_y = -\frac{F_Y - 1}{2} : \frac{F_Y - 1}{2}$ do
            for $f_x = -\frac{F_X - 1}{2} : \frac{F_X - 1}{2}$ do
              $O[k][x][y] += I[c][x + f_x][y + f_y] \times W[k][c][f_x][f_y]$
Dimension 1: Loop Optimizations

1. \textbf{for} \ b = 0 : B \ \textbf{do}
2. \quad \textbf{for} \ k = 0 : K \ \textbf{do}
3. \quad \quad \textbf{for} \ c = 0 : C \ \textbf{do}
4. \quad \quad \quad \textbf{for} \ y = 0 : Y \ \textbf{do}
5. \quad \quad \quad \quad \textbf{for} \ x = 0 : X \ \textbf{do}
6. \quad \quad \quad \quad \quad \textbf{for} \ \frac{f_y}{2} = \frac{F_y - 1}{2} : \frac{F_y - 1}{2} \ \textbf{do}
7. \quad \quad \quad \quad \quad \quad \textbf{for} \ \frac{f_x}{2} = \frac{F_x - 1}{2} : \frac{F_x - 1}{2} \ \textbf{do}
8. \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad O[k][x][y] +=
9. \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad I[c][x + f_x][y + f_y] \times W[k][c][f_x][f_y]

Not how anyone does it!
Dimension 1: Loop Optimizations - Tiling

Original:
for (i=0; i<m; i++)
  for (j=0; j<n; j++)
    *=*b[j];

Tiled:
for (ii=0; ii<m; ii+=TILE)
  for (j=0; j<n; j++)
    for (i=ii; i<ii+TILE; i++)
      *=*b[j];

Cache size: 4
TILE=4
(must be tuned to cache size)

Cache hit rate without tiling: 0%
Cache hit rate with tiling: 50%
Dimension 1: Loop Optimizations - Reordering

Row-major order

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

Column-major order

\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} \\
  a_{21} & a_{22} & a_{23} \\
  a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]
Dimension 2: Dataflow

Some heuristics:

- Maximize filter reuse: “weight stationary” (WS)
- Maximize partial sum reuse: “output stationary” (OS)
- No local reuse (NLR)
- Row stationary (RS)
Dimension 2: Dataflow

Some heuristics:

- Maximize filter reuse: “weight stationary” (WS)
- Maximize partial sum reuse: “output stationary” (OS)
- No local reuse (NLR)
- Row stationary (RS)
Some heuristics:

- Maximize filter reuse: “weight stationary” (WS)
- Maximize partial sum reuse: “output stationary” (OS)
- No local reuse (NLR)
- Row stationary (RS)
Some heuristics:

- Maximize filter reuse: “weight stationary” (WS)
- Maximize partial sum reuse: “output stationary” (OS)
- No local reuse (NLR)
- Row stationary (RS)
Performance of Different Dataflows

![Graph showing the normalized energy per operation for different dataflow choices. The x-axis represents different dataflow options: RS, WS, OS_A, OS_B, OS_C, NLR. The y-axis represents the normalized energy. The graph illustrates the energy efficiency of ALU, DRAM, Buffer, Array, and RF.]
Dimension 3: Hardware Resource Allocation

- Dimensions of PE array
- Size of different layers of the memory hierarchy
- Energy cost and latency of accesses
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion

Presenter: Derrick Blakely (University of Virginia)

DNN Dataflow Choice Is Overrated

Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qioyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. Stanford University, Tsinghua University 2018

https://qdata.github.io/deep2Read
Halide Primitives

- Two new Halide primitives: Accelerate and Systolic
- One modified primitive: Unroll

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Scheduling primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall scope</td>
<td>accelerate</td>
</tr>
<tr>
<td>Loop blocking</td>
<td>tile, reorder</td>
</tr>
<tr>
<td>Dataflow</td>
<td>unroll, systolic</td>
</tr>
<tr>
<td>Resource allocation</td>
<td>in, compute_at</td>
</tr>
</tbody>
</table>
Halide Pseudocode

1 // Define (minX, extentX, minY, extentY, minK, extentK)
2 RDom r(-2, 5, -2, 5, 0, 3);
3
4 output(x, y, k) += input(x + r.x, y + r.y, r.z)
5   * w(r.x + 2, r.y + 2, r.z, k);
Halide Pseudocode

```halide
1 d = output.in()
2
3 output.tile(x, y, xo, yo, xi, yi, 28, 28)
   .reorder(xi, yi, xo, yo)
   .accelerate({input, w});

7 input.in().compute_at(output, xo);
8 w.in().compute_at(output, xo);
9 output.compute_at(d, xo);
```
for (k, 0, 64) 
  for (yo, 0, 4) 
    for (xo, 0, 4) 
      // Allocate local buffer for output. 
      alloc obuf[28, 28, 1] 

      // Allocate local buffer for input. 
      alloc ibuf[28 + 5 - 1, 28 + 5 - 1, 3] 
      // Copy input to buffer. 
      ibuf[...] = input[...] 

      // Allocate local buffer for w. 
      alloc wbuf[5, 5, 3, 1] 
      // Copy w to buffer. 
      wbuf[...] = w[...] 

    for (yi, 0, 28) 
      for (xi, 0, 28) 

        for (r.z, 0, 3) 
          for (r.y, -2, 5) 
            for (r.x, -2, 5) 
              obuf(xi, yi, 0) += 
              ibuf(xi + r.x, yi + r.y, r.z) 
              * wbuf(r.x + 2, r.y + 2, r.z, 0) 

      // Copy buffer to output. 
      output[...] = obuf[...]
New Primitive: Accelerate

- Accelerate: “Defines scope and interface to the rest of the system”
- Marks for transformation to some dataflow IR
- Dataflow IR is some kind of special C++ program
- IR compiled into Verilog using High-Level Synthesis (HLS) (probably involves additional optimizations)
- Nothing about latency or throughput
New Primitive: Accelerate

- Accelerate: “Defines scope and interface to the rest of the system”
- Marks for transformation to some dataflow IR
  - Dataflow IR is some kind of special C++ program
  - IR compiled into Verilog using High-Level Synthesis (HLS) (probably involves additional optimizations)
- Nothing about latency or throughput
New Primitive: Accelerate

- Accelerate: “Defines scope and interface to the rest of the system”
- Marks for transformation to some dataflow IR
- Dataflow IR is some kind of special C++ program
- IR compiled into Verilog using High-Level Synthesis (HLS) (probably involves additional optimizations)
- Nothing about latency or throughput
New Primitive: Accelerate

- Accelerate: “Defines scope and interface to the rest of the system”
- Marks for transformation to some dataflow IR
- Dataflow IR is some kind of special C++ program
- IR compiled into Verilog using High-Level Synthesis (HLS) (probably involves additional optimizations)
- Nothing about latency or throughput
New Primitive: Accelerate

- Accelerate: “Defines scope and interface to the rest of the system”
- Marks for transformation to some dataflow IR
- Dataflow IR is some kind of special C++ program
- IR compiled into Verilog using High-Level Synthesis (HLS) (probably involves additional optimizations)
- Nothing about latency or throughput
Custom Primitive: Systolic

- Systolic flag: PEs communicate during loop unrolling
- No systolic flag: unrolled loop performs tree reduction
Custom Primitive: Systolic

- Systolic flag: PEs communicate during loop unrolling
- No systolic flag: unrolled loop performs tree reduction
Overridden Primitive: Unroll

- Basic idea: loop unrollings correspond to different spatial architectures
- Example: unroll weights vertically and horizontally is “weight stationary” ($F_Y|F_X$)

```
\begin{array}{ccc}
  w_{11} & w_{12} & w_{12} \\
  w_{21} & w_{22} & w_{23} \\
  w_{31} & w_{32} & w_{33} \\
\end{array}
```

![Diagram](https://qdata.github.io/deep2Read)
Overridden Primitive: Unroll

- $F_Y|Y = \text{Eyeriss}$
- $C|K = \text{Google TPU}$

Diagram:

- $F_Y = 0$ W0
  - PE(0,0) -> O0
- $F_Y = 1$ W1
  - PE(1,0) -> PE(1,1) -> O1
- $F_Y = 2$ W2
  - PE(2,0) -> PE(2,1) -> O2

$C = 0$ I0
- $C = 1$ I1
- $C = 2$ I2
Recap

- Provide modified Halide with functional programming description of conv layer
- Modified Halide both generates a spatial architecture and schedules execution
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion
**Analysis Framework**

- Use parameters from AlexNet, MobileNet, and GoogleNet
- Use CACTI to analyze performance and energy
- RTL model
- Model with 28nm technology (Eyeriss uses 65nm, TPU 28nm)
- Model memory usage energy with weighted linear model
- Use their tool to model Eyeriss, TPU, etc.
Analysis Framework

- Use parameters from AlexNet, MobileNet, and GoogleNet
- Use CACTI to analyze performance and energy
- RTL model
- Model with 28nm technology (Eyeriss uses 65nm, TPU 28nm)
- Model memory usage energy with weighted linear model
- Use their tool to model Eyeriss, TPU, etc.
Analysis Framework

- Use parameters from AlexNet, MobileNet, and GoogleNet
- Use CACTI to analyze performance and energy
- RTL model
  - Model with 28nm technology (Eyeriss uses 65nm, TPU 28nm)
  - Model memory usage energy with weighted linear model
  - Use their tool to model Eyeriss, TPU, etc.
Analysis Framework

- Use parameters from AlexNet, MobileNet, and GoogleNet
- Use CACTI to analyze performance and energy
- RTL model
- Model with 28nm technology (Eyeriss uses 65nm, TPU 28nm)
  - Model memory usage energy with weighted linear model
  - Use their tool to model Eyeriss, TPU, etc.
Analysis Framework

- Use parameters from AlexNet, MobileNet, and GoogleNet
- Use CACTI to analyze performance and energy
- RTL model
- Model with 28nm technology (Eyeriss uses 65nm, TPU 28nm)
- Model memory usage energy with weighted linear model
- Use their tool to model Eyeriss, TPU, etc.
Analysis Framework

- Use parameters from AlexNet, MobileNet, and GoogleNet
- Use CACTI to analyze performance and energy
- RTL model
- Model with 28nm technology (Eyeriss uses 65nm, TPU 28nm)
- Model memory usage energy with weighted linear model
- Use their tool to model Eyeriss, TPU, etc.
Loop unrollings/dataflows are horizontal. 1 color = 1 register file size. Same color + same dataflow means different “replications” (parallelized loops).

(a) AlexNet

(b) MobileNet Depthwise
With good loop optimizations, reuse is high (high RF energy use). 2d = best blue points, Global = best red points from previous fig.
Outline

1. Background: Spatial Architectures
2. Background: Halide
3. DNN Dataflow is Overrated
4. Generating Hardware Designs Using Halide
5. Experiments
6. Conclusion
Issues

- They don’t successfully show that “dataflow choice is overrated”
- What about latency and throughput? Dataflow choice could be super important with respect to those
- Are they doing a good job modeling their competitors?
- Can Halide can actually model the space of “all possible systolic accelerators”?
- Uses very different nm process from Eyeriss
- Bad figures
Issues

- They don’t successfully show that “dataflow choice is overrated”
- What about latency and throughput? Dataflow choice could be super important with respect to those
- Are they doing a good job modeling their competitors?
- Can Halide can actually model the space of “all possible systolic accelerators”?
- Uses very different nm process from Eyeriss
- Bad figures
Issues

- They don’t successfully show that “dataflow choice is overrated”
- What about latency and throughput? Dataflow choice could be super important with respect to those
- Are they doing a good job modeling their competitors?
- Can Halide can actually model the space of “all possible systolic accelerators”?
- Uses very different nm process from Eyeriss
- Bad figures
Issues

- They don’t successfully show that “dataflow choice is overrated”
- What about latency and throughput? Dataflow choice could be super important with respect to those
- Are they doing a good job modeling their competitors?
- Can Halide can actually model the space of “all possible systolic accelerators”?

  - Uses very different nm process from Eyeriss
  - Bad figures
Issues

- They don’t successfully show that “dataflow choice is overrated”
- What about latency and throughput? Dataflow choice could be super important with respect to those
- Are they doing a good job modeling their competitors?
- Can Halide can actually model the space of “all possible systolic accelerators”?
- Uses very different nm process from Eyeriss
- Bad figures
Issues

- They don’t successfully show that “dataflow choice is overrated”
- What about latency and throughput? Dataflow choice could be super important with respect to those
- Are they doing a good job modeling their competitors?
- Can Halide can actually model the space of “all possible systolic accelerators”?
- Uses very different nm process from Eyeriss
- Bad figures
Lessons Learned

- Decoupling algorithm from processing details can be valuable for accelerating ML
- Idea of using Halide to generate designs is very interesting
- Good demonstration that loop optimizations are important
Lessons Learned

- Decoupling algorithm from processing details can be valuable for accelerating ML
- Idea of using Halide to generate designs is very interesting
- Good demonstration that loop optimizations are important
Lessons Learned

- Decoupling algorithm from processing details can be valuable for accelerating ML
- Idea of using Halide to generate designs is very interesting
- Good demonstration that loop optimizations are important