Fusion of Operators of Computational Graphs via Greedy Clustering: The XNNC Experience

Michael Canesche canesche@cadence.com
Vanderson Rosário vrosario@cadence.com
Edson Borin edson@ic.unicamp.br
Fernando Pereira fernando@dcc.ufmg.br

Goal

DCC

1. Design and implement a new kernel fusion algorithm

Goal

DCC

1. Design and implement a new kernel fusion algorithm
2. Deploy this algorithm on the XNNC Compiler

Goal

1. Design and implement a new kernel fusion algorithm
2. Deploy this algorithm on the XNNC Compiler

Xtensa® Neural Network Compiler (XNNC)

XNNC

1. Design and implement a new kernel fusion algorithm
2. Deploy this algorithm on the XNNC Compiler

Xtensa® Neural Network Compiler (XNNC)

- Floating-point Convolutional Neural Network → Optimized, fixed-point code
- Cadence Tensilica® processors.

XNNC

DCC

1. Automotive Audio DSPs
2. Light Detection and Ranging
3. Consumer Electronics
4. etc

Xtensa® Neural Network Compiler (XNNC)

- Floating-point Convolutional Neural Network → Optimized, fixed-point code
- Cadence Tensilica® processors.

XNNC

DCC

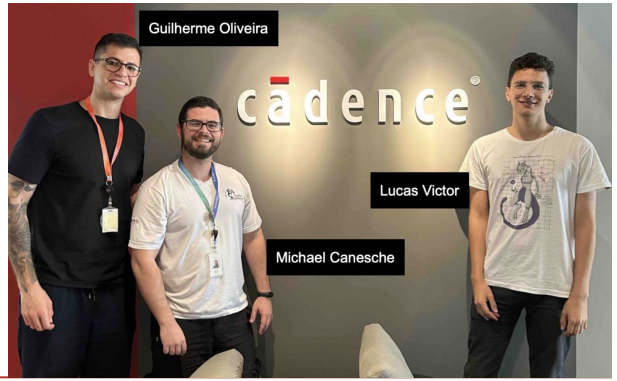
1. C++
2. 1.5 Million LoC
3. LLVM (MLIR)

Xtensa® Neural Network Compiler (XNNC)

- Floating-point Convolutional Neural Network → Optimized, fixed-point code
- Cadence Tensilica® processors.

XNNC

1. C++
2. 1.5 Million LoC
3. LLVM (MLIR)



Xtensa® Neural Network Compiler (XNNC)

- Floating-point Convolutional Neural Network → Optimized, fixed-point code
- Cadence Tensilica® processors.

XNNC

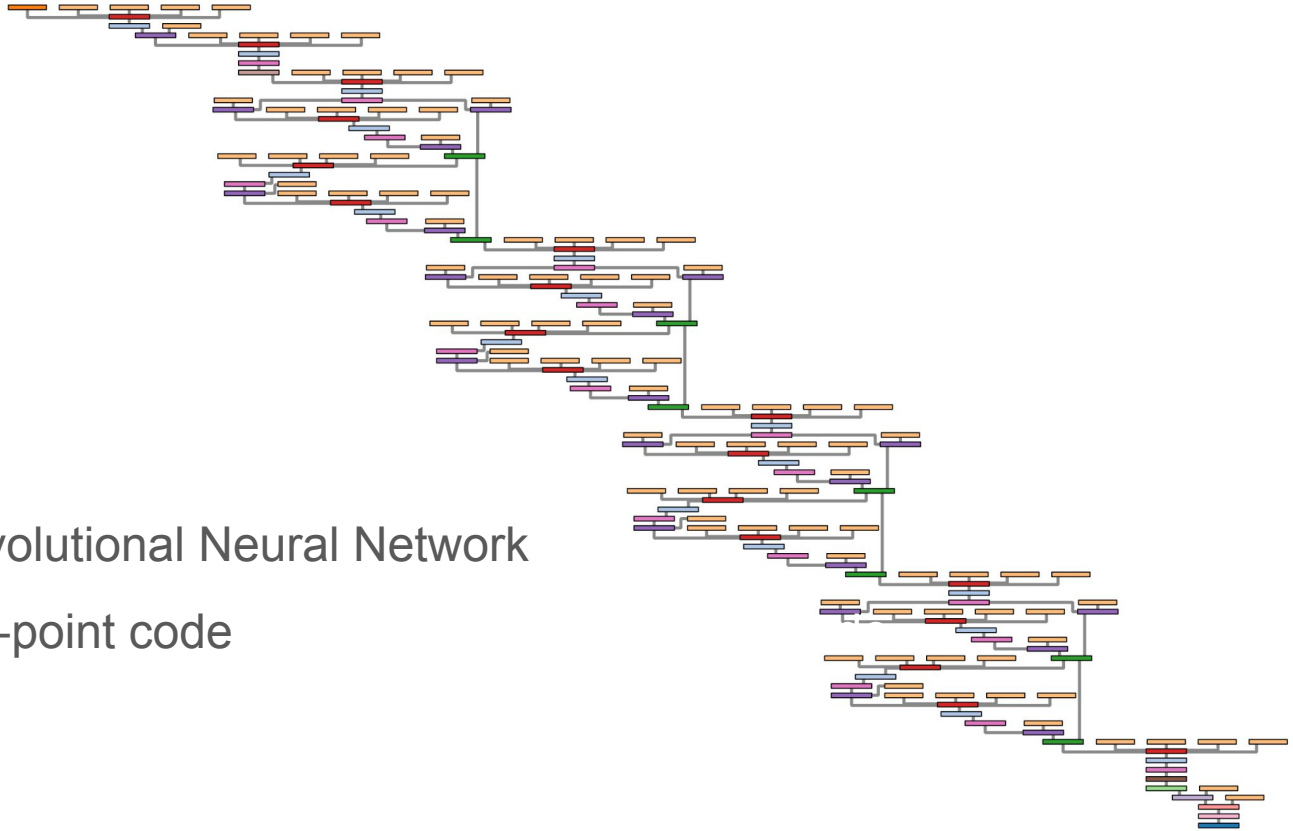
DCC

- Floating-point Convolutional Neural Network

  → Optimized, fixed-point code

XNNC

DCC

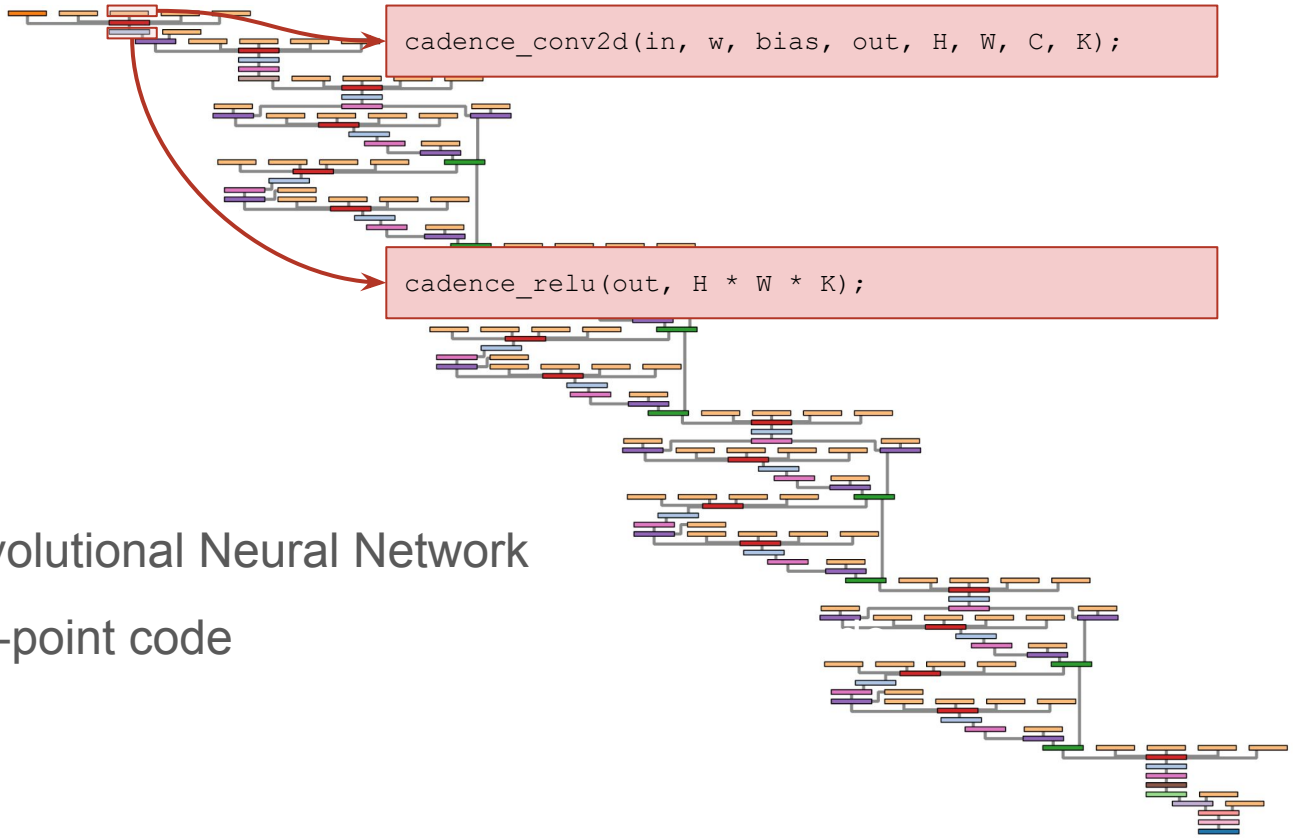


- Floating-point Convolutional Neural Network

  → Optimized, fixed-point code

XNNC

DCC

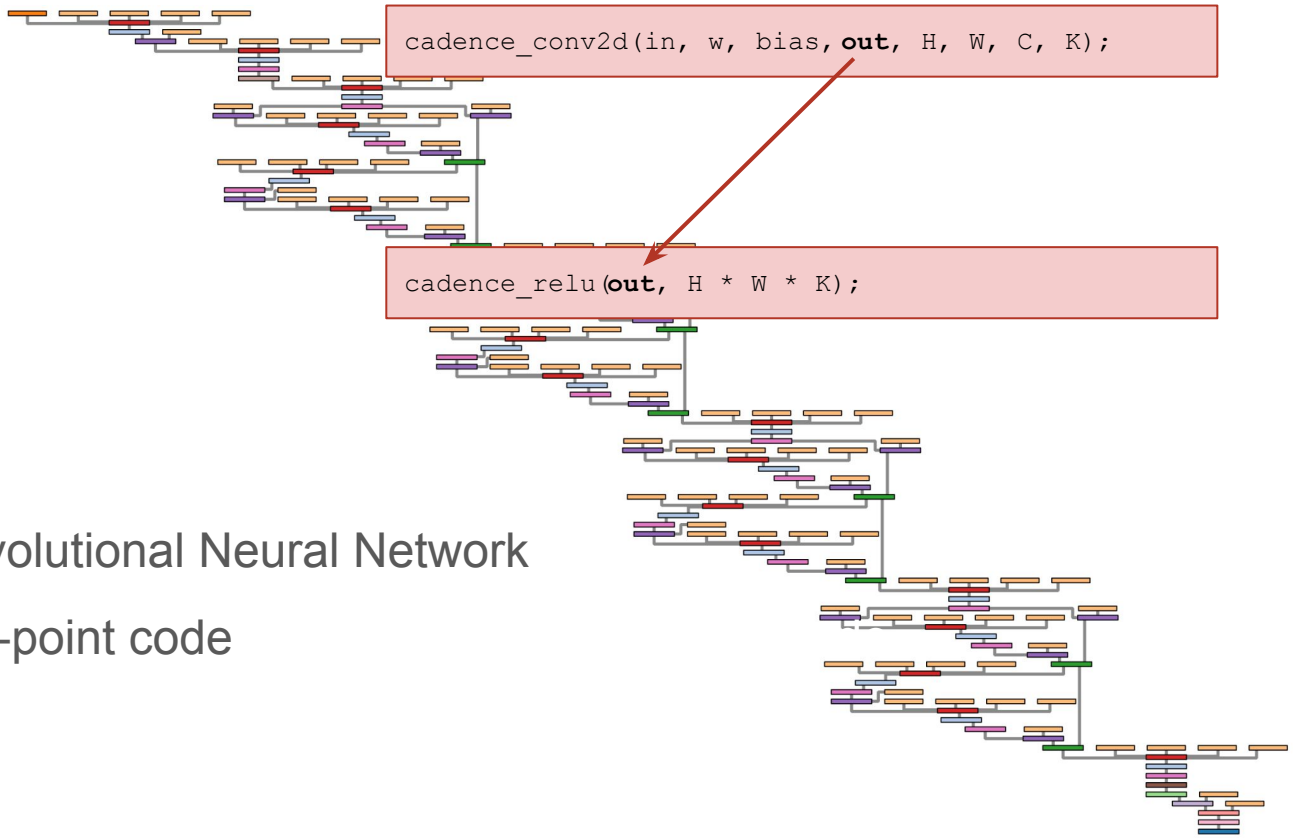


- Floating-point Convolutional Neural Network

  → Optimized, fixed-point code

XNNC

DCC



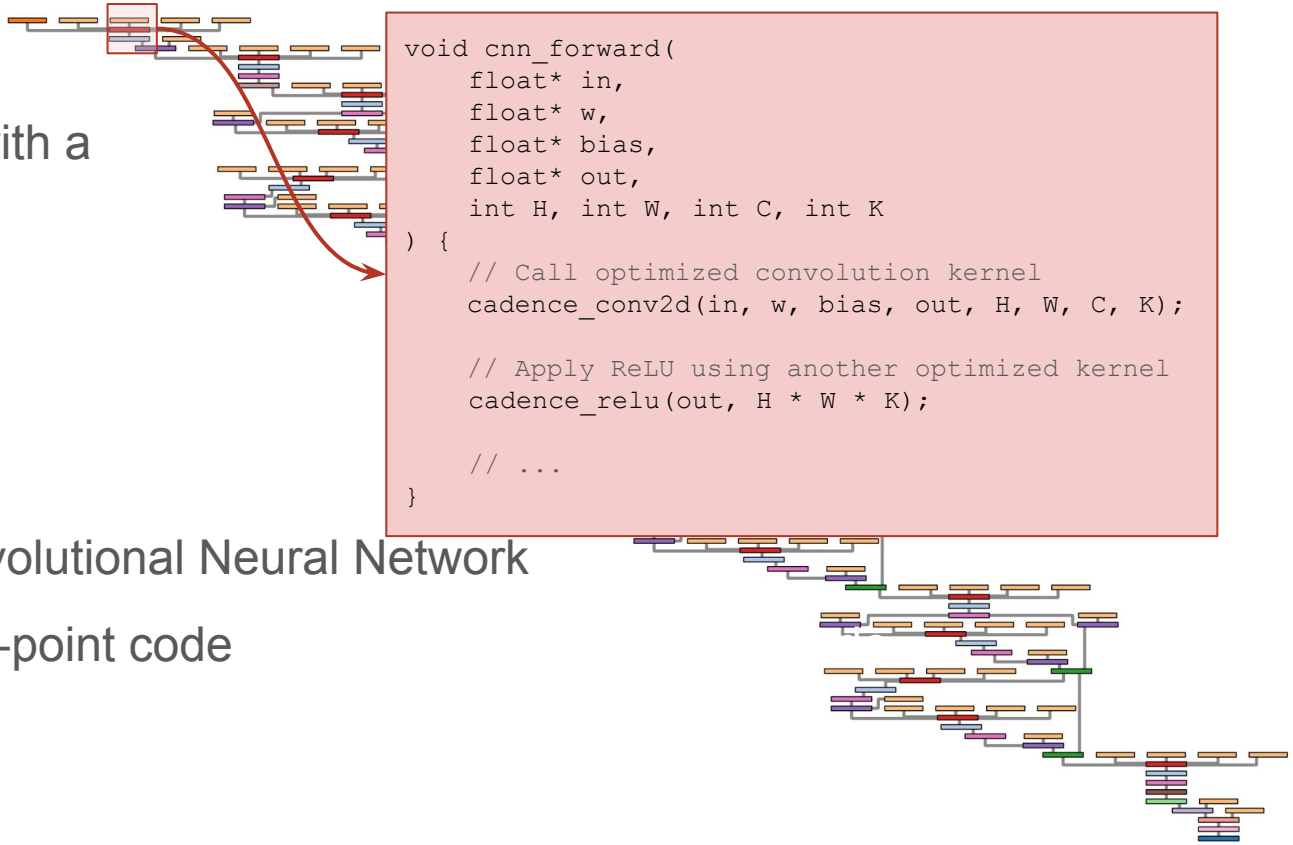
- Floating-point Convolutional Neural Network

  → Optimized, fixed-point code

XNNC

DCC

1. Pattern matching with a kernel library



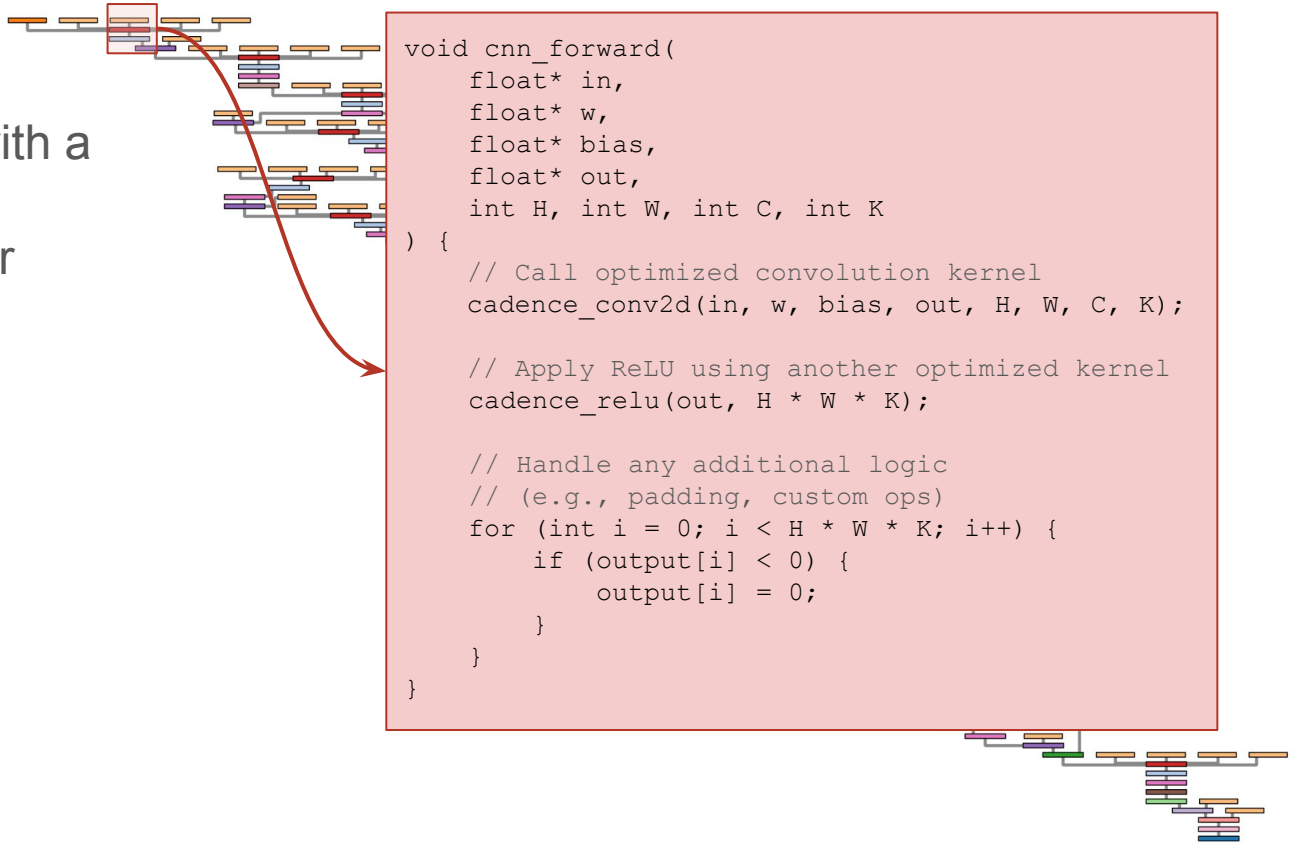
```
void cnn_forward(
    float* in,
    float* w,
    float* bias,
    float* out,
    int H, int W, int C, int K
) {
    // Call optimized convolution kernel
    cadence_conv2d(in, w, bias, out, H, W, C, K);

    // Apply ReLU using another optimized kernel
    cadence_relu(out, H * W * K);

    // ...
}
```

- Floating-point Convolutional Neural Network

  → Optimized, fixed-point code

XNNC

1. Pattern matching with a kernel library
2. Code generation for unmatched parts



```
void cnn_forward(
    float* in,
    float* w,
    float* bias,
    float* out,
    int H, int W, int C, int K
) {
    // Call optimized convolution kernel
    cadence_conv2d(in, w, bias, out, H, W, C, K);

    // Apply ReLU using another optimized kernel
    cadence_relu(out, H * W * K);

    // Handle any additional logic
    // (e.g., padding, custom ops)
    for (int i = 0; i < H * W * K; i++) {
        if (output[i] < 0) {
            output[i] = 0;
        }
    }
}
```

XNNC

1. Pattern matching with a kernel library
2. Code generation for unmatched parts
3. Final Translation to Hardware-Specific Instructions

```
void cnn_forward(
    float* in,
    float* w,
    float* bias,
    float* out,
    int H, int W, int C, int K
) {
    // Call optimized convolution kernel
    cadence_conv2d(in, w, bias, out, H, W, C, K);

    // Apply ReLU using another optimized kernel
    cadence_relu(out, H * W * K);

    // Handle any additional logic
    // (e.g., padding, custom ops)
    for (int i = 0; i < H * W * K; i++) {
        if (output[i] < 0) {
            output[i] = 0;
        }
    }
}
```

XNNC: "Instruction Selection"

DCC

↳ Computational graph → kernels + loops

XNNC: "Instruction Selection"

↳ Computational graph → kernels + loops

Traditional compiler: Instruction Selection

↳ AST → Instructions

DCC

But, what is "Operator Fusion"?

DCC

But, what is "Operator Fusion"?

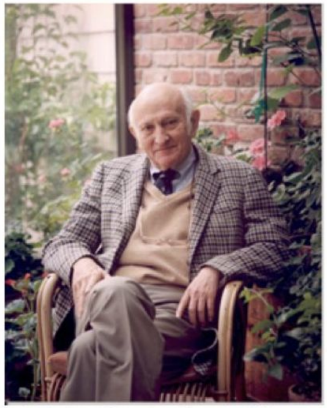
DCC

A CATALOGUE OF OPTIMIZING TRANSFORMATIONS

Frances E. Allen
John Cocke
IBM Thomas J. Watson Research Center
Yorktown Heights

Frances Allen and John Cocke's *"A Catalogue of Optimizing Transformations"* was written in 1971. More than fifty years later, the optimizations they listed in that work still form the backbone of optimizing compilers.

Frances E. Allen receiving the Erna Hamburger Distinguished Lecture Award at the EPFL



John Cocke in his garden. Computer History Museum archive, (c) Louis Fabian Bachrach

Example: Single-Head Attention

DCC

- Core of the Transformer Architecture

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. **Attention is All you Need**. In Neural Information Processing Systems. 2017

Example: Single-Head Attention

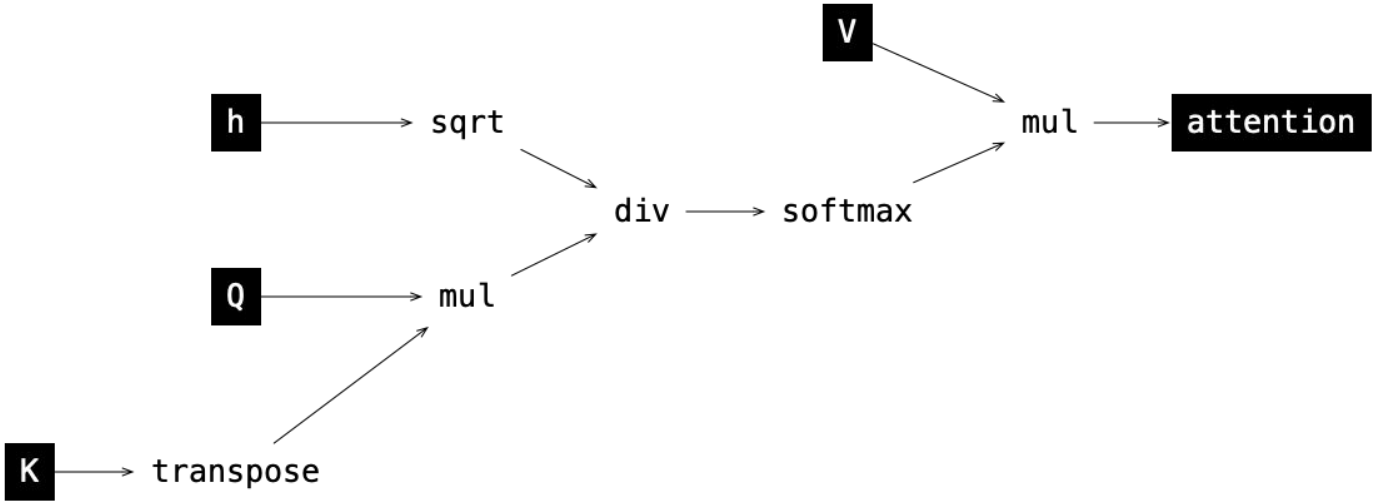
DCC

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. **Attention is All you Need**. In Neural Information Processing Systems. 2017

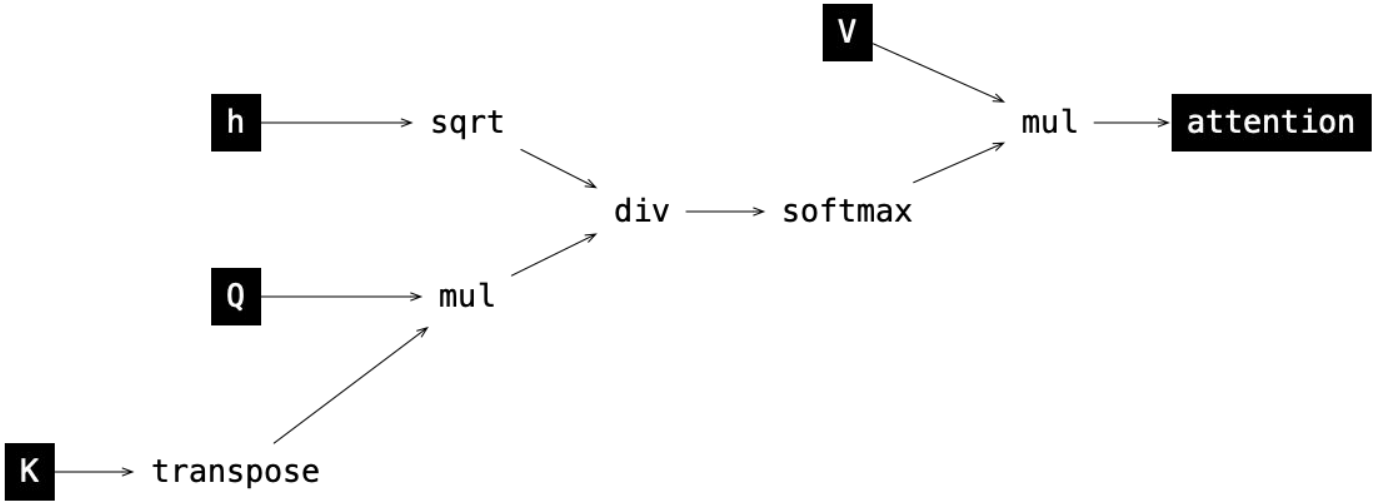
Example: Single-Head Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$



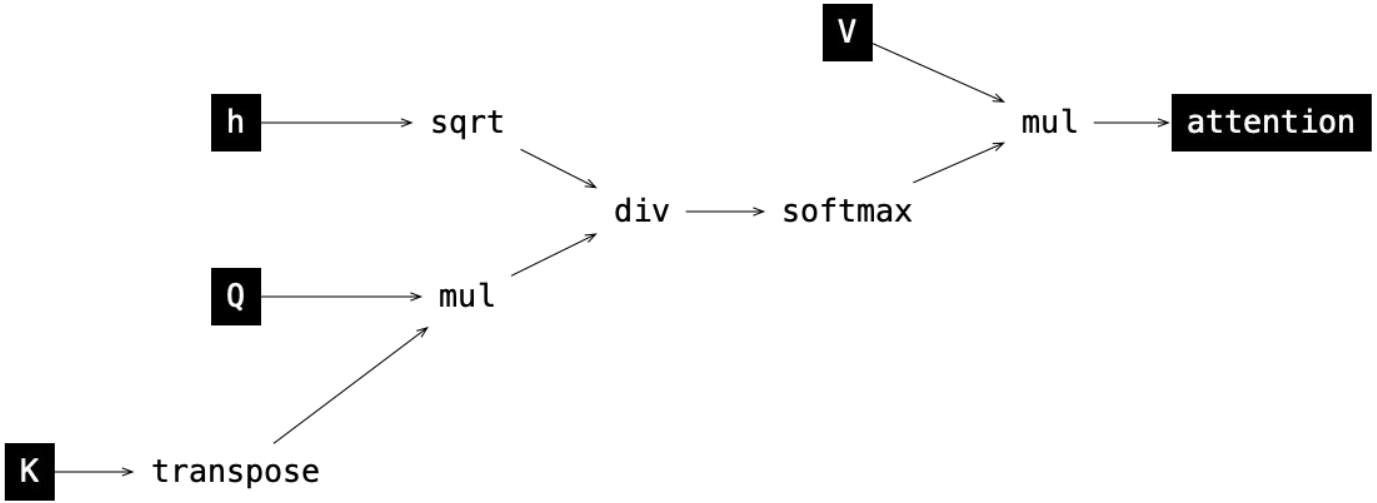
Example: Single-Head Attention

```
scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(h)
attention_weights = torch.softmax(scores, dim=-1)
attention_output = torch.matmul(attention_weights, V)
```



Example: Single-Head Attention

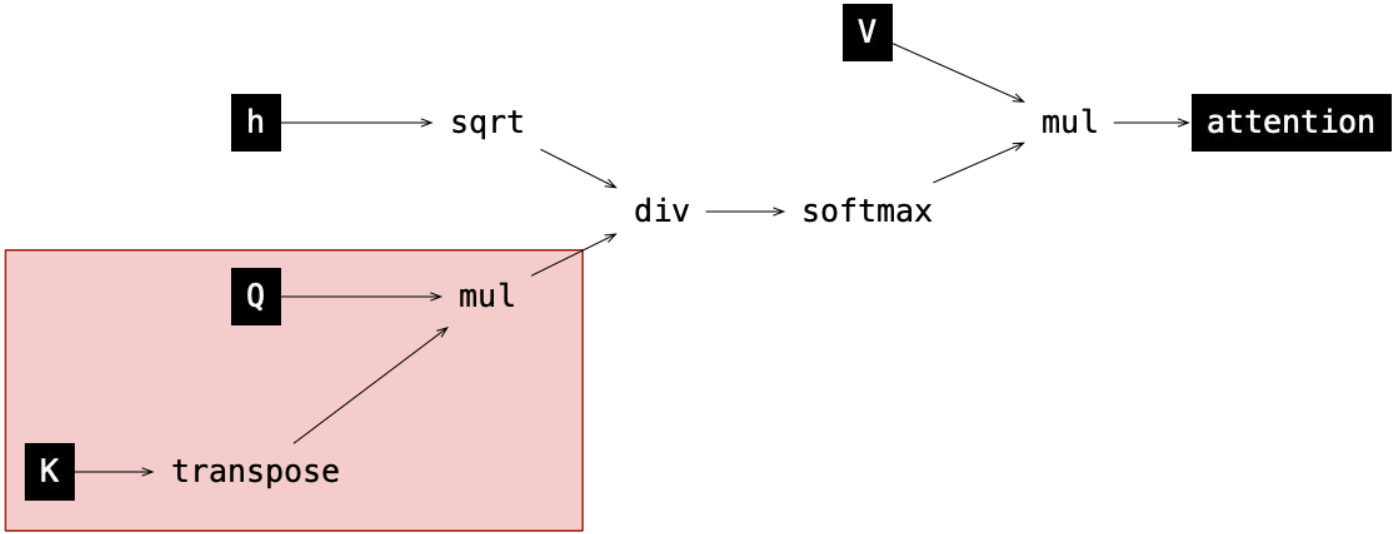
```
scores = torch.matmul(Q, K.transpose(-2, -1)) / math.sqrt(h)
attention_weights = torch.softmax(scores, dim=-1)
attention_output = torch.matmul(attention_weights, V)
```



Example: Single-Head Attention

DCC

```
scores = fused_matmul_transpose(Q, K, -2, -1) / math.sqrt(h)
attention_weights = torch.softmax(scores, dim=-1)
attention_output = torch.matmul(attention_weights, V)
```

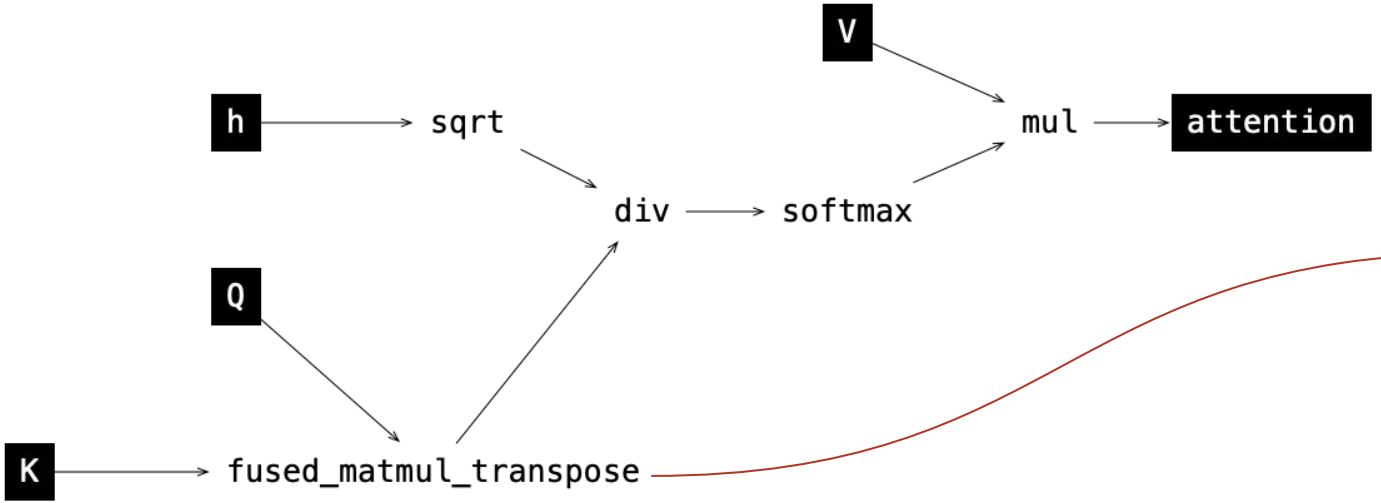


Example: Single-Head Attention

```
scores = fused_matmul_transpose(Q, K, -2, -1) / math.sqrt(h)
```

Compare with previous code:

```
scores = torch.matmul(Q, K.transpose()) / math.sqrt(h)
```



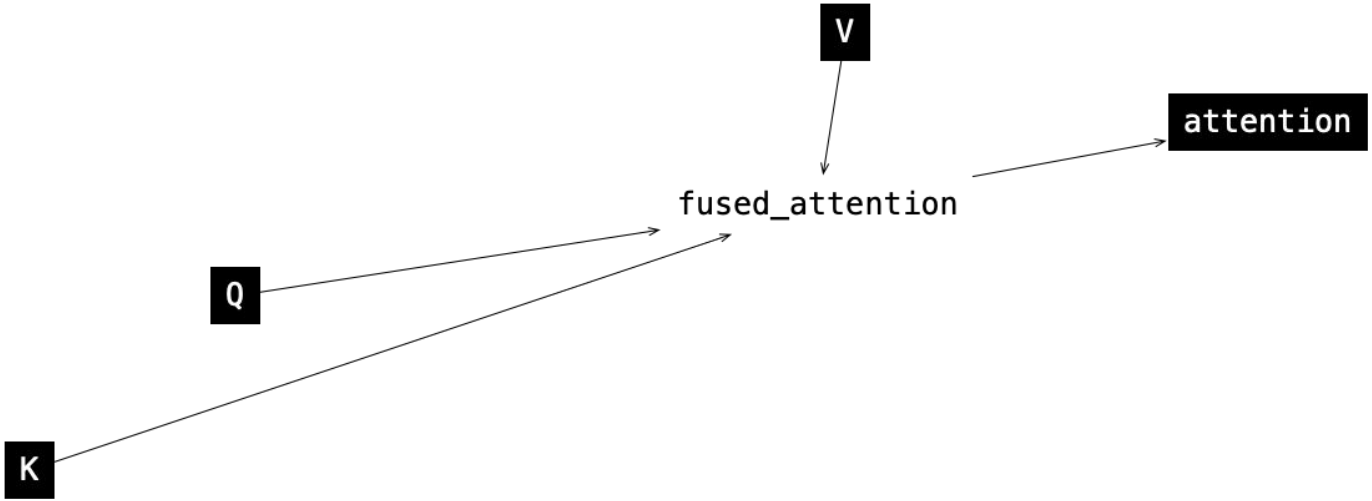
In memory:

- K
- Q

Example: Single-Head Attention

DCC

```
from flash_attention import fused_attention

attention_output = fused_attention(Q, K, V, scale=1.0 / math.sqrt(h))
```

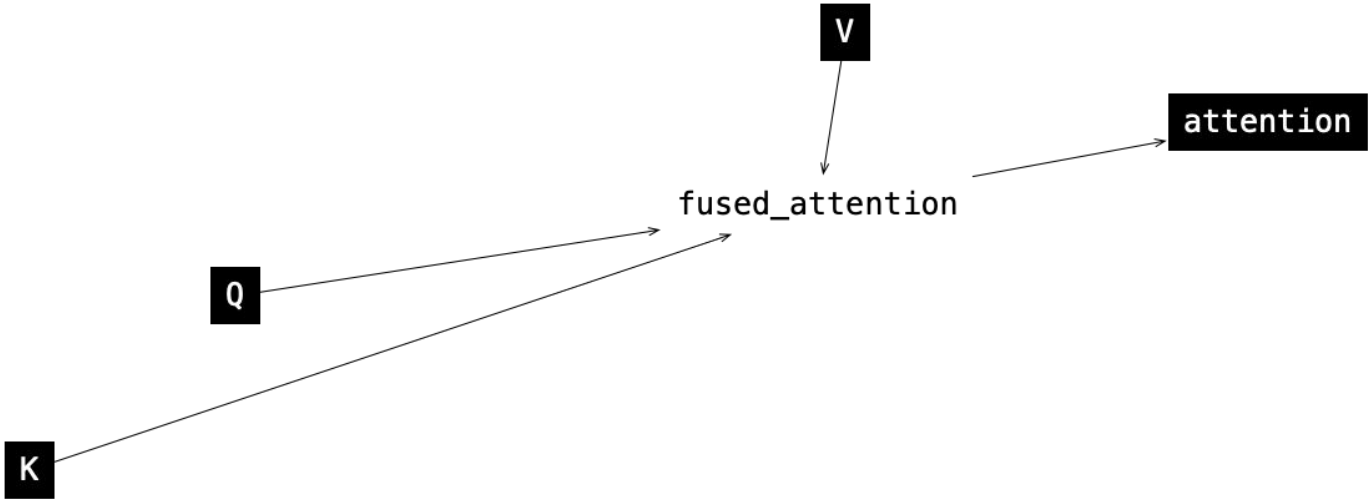


In memory:

- K
- Q

Example: Single-Head Attention

```
from flash_attention import fused_attention

attention_output = fused_attention(Q, K, V, scale=1.0 / math.sqrt(h))
```

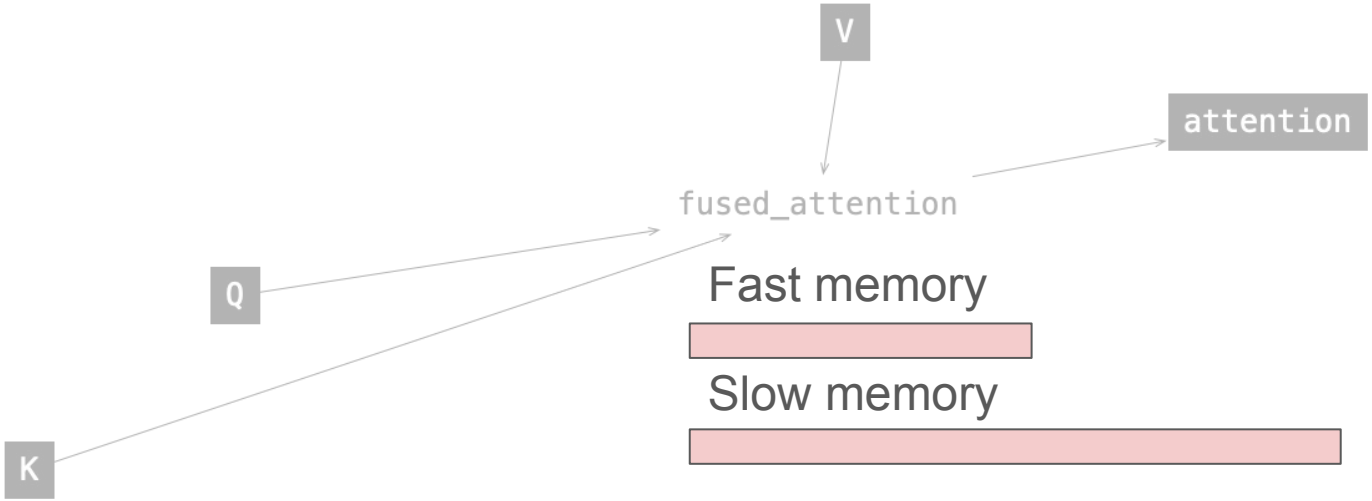


In memory:

- K
- Q
- V

Example: Single-Head Attention

```
from flash_attention import fused_attention

attention_output = fused_attention(Q, K, V, scale=1.0 / math.sqrt(h))
```



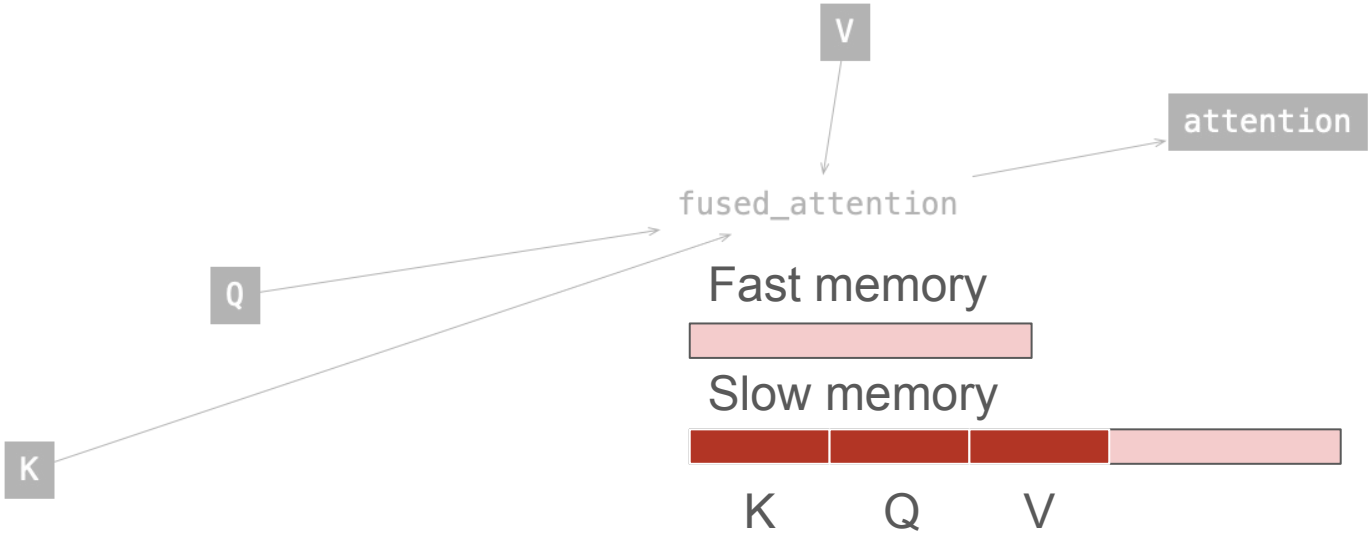
In memory:

- K
- Q
- V

Example: Single-Head Attention

DCC

```
from flash_attention import fused_attention

attention_output = fused_attention(Q, K, V, scale=1.0 / math.sqrt(h))
```



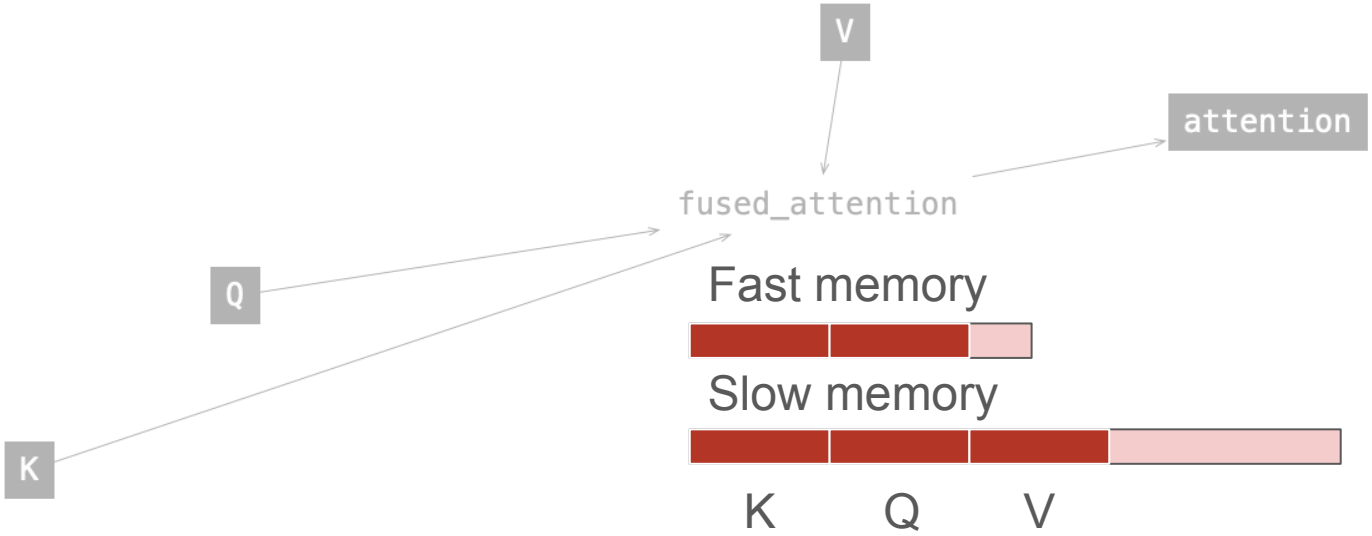
In memory:

- K
- Q
- V

Example: Single-Head Attention

DCC

```
from flash_attention import fused_attention

attention_output = fused_attention(Q, K, V, scale=1.0 / math.sqrt(h))
```

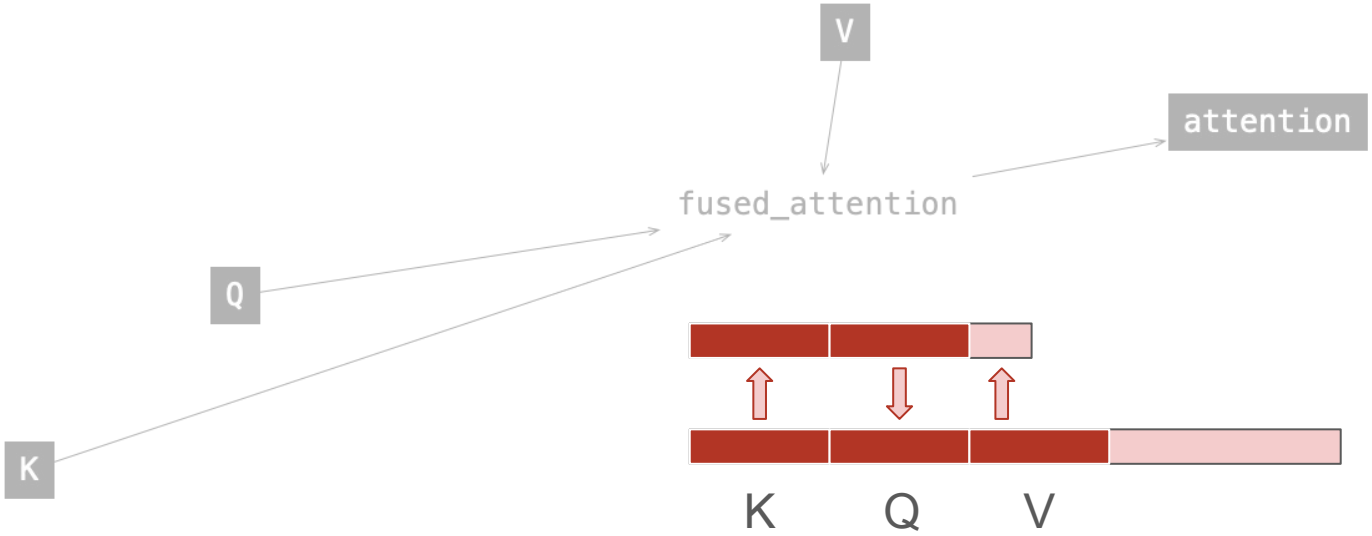


In memory:

- K
- Q

Example: Single-Head Attention

```
from flash_attention import fused_attention

attention_output = fused_attention(Q, K, V, scale=1.0 / math.sqrt(h))
```



In memory:

- K
- Q
- V

First Observation

DCC

1. Fusion improves performance, but not always

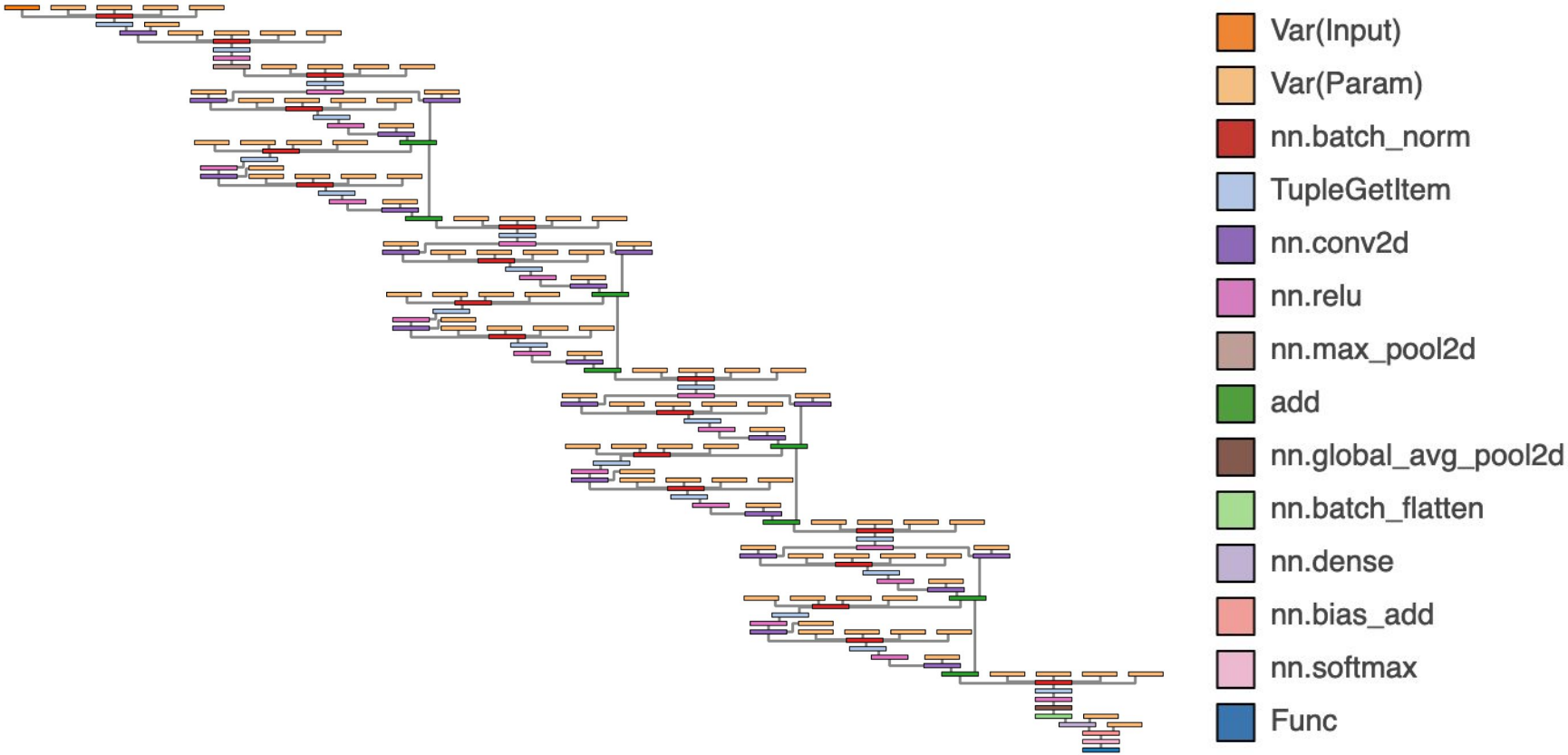
Second Observation

DCC

1. Fusion improves performance, but not always
2. Deciding which operators to fuse is a challenging problem

Example of a Computational Graph: ResNet18

DCC

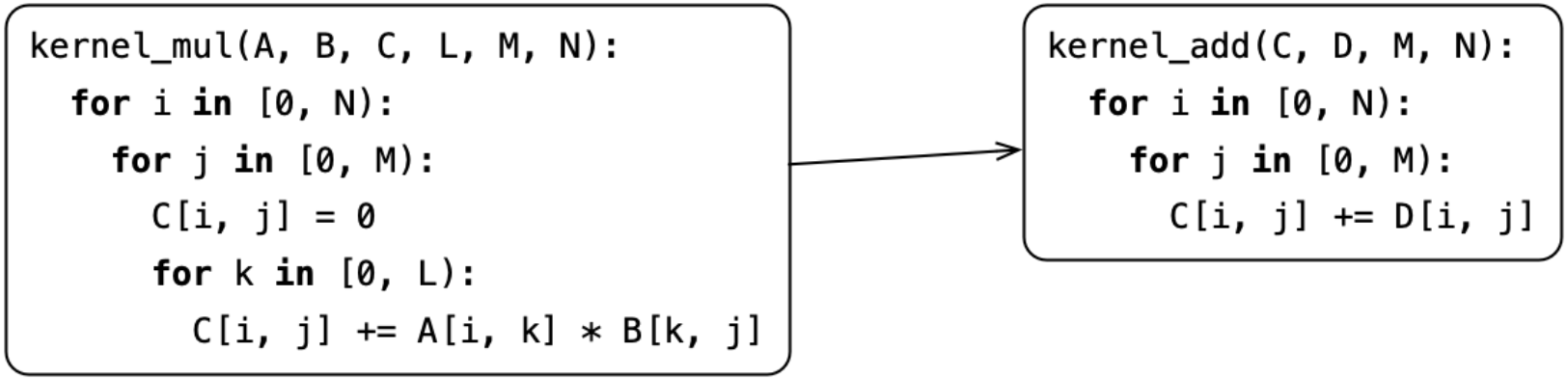


Which Operators does it Make Sense to Fuse?

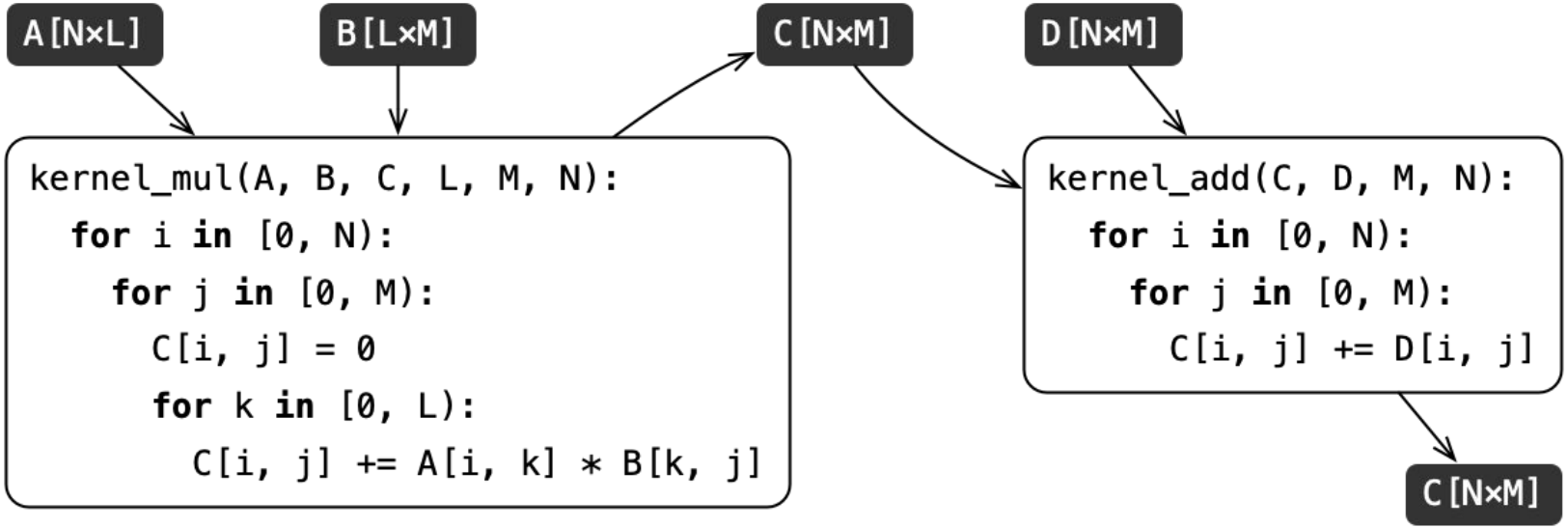
DCC

Which Operators does it Make Sense to Fuse?

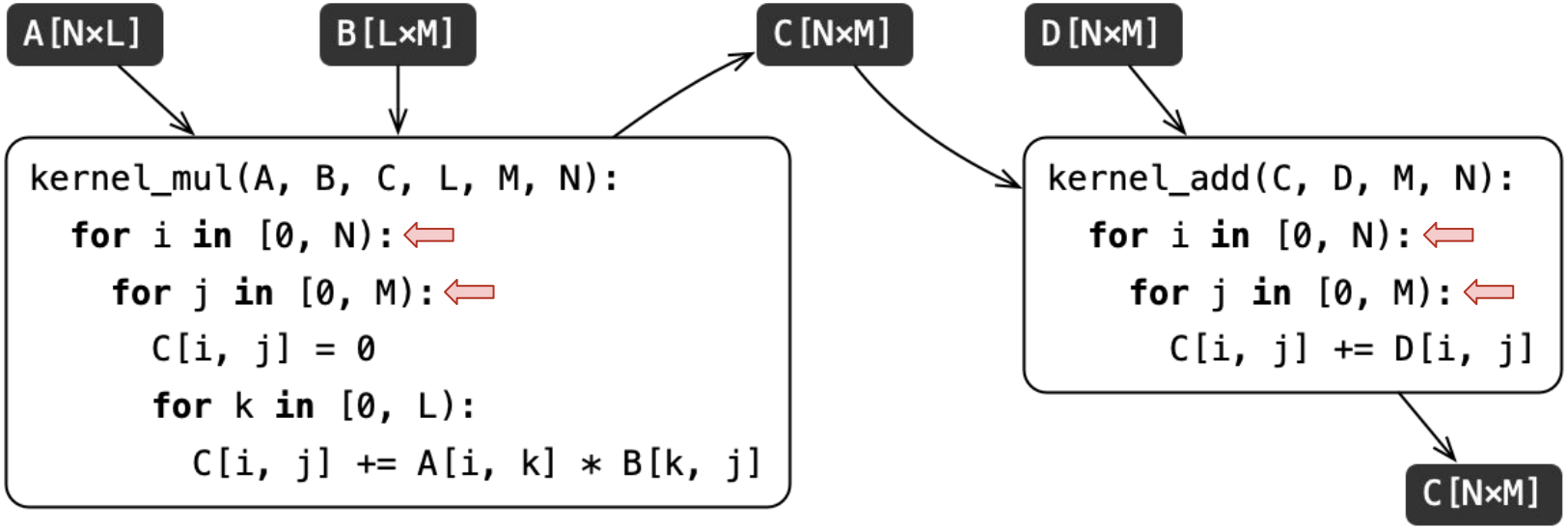
DCC



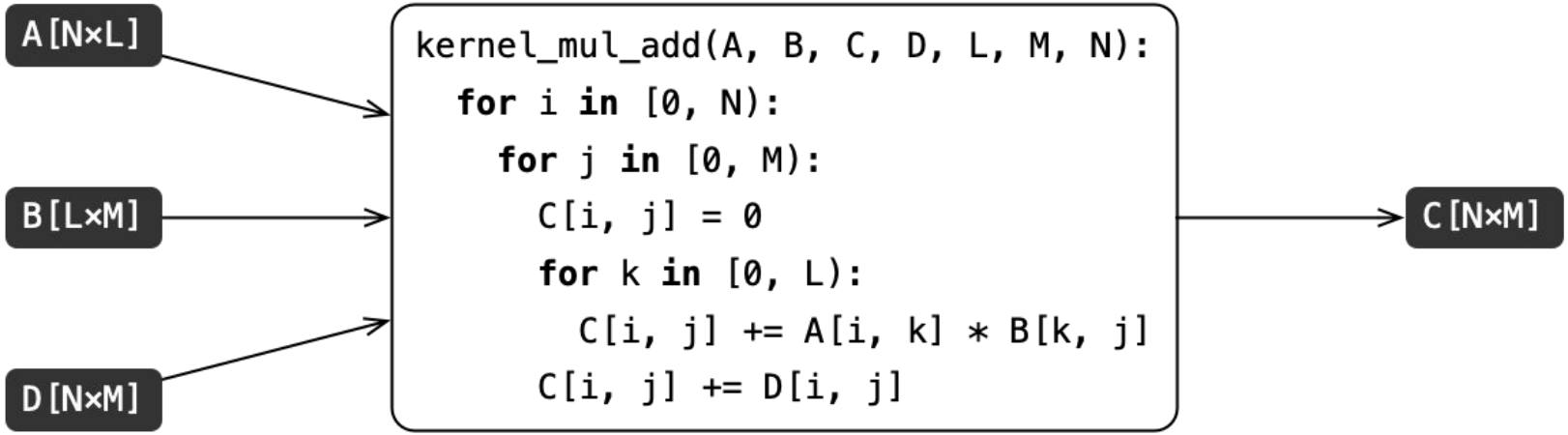
Which Operators does it Make Sense to Fuse?



Which Operators does it Make Sense to Fuse?



Which Operators does it Make Sense to Fuse?



Third Observation

DCC

1. Fusion improves performance, but not always
2. Deciding which operators to fuse is a challenging problem
3. Fusion, when seen as an optimization problem, is guided by a cost model

Third Observation

DCC

1. Fusion improves performance, but not always
2. Deciding which operators to fuse is a challenging problem
3. Fusion, when seen as an optimization problem, is guided by a cost model
    a. **Dynamic**: profile and decide

Third Observation

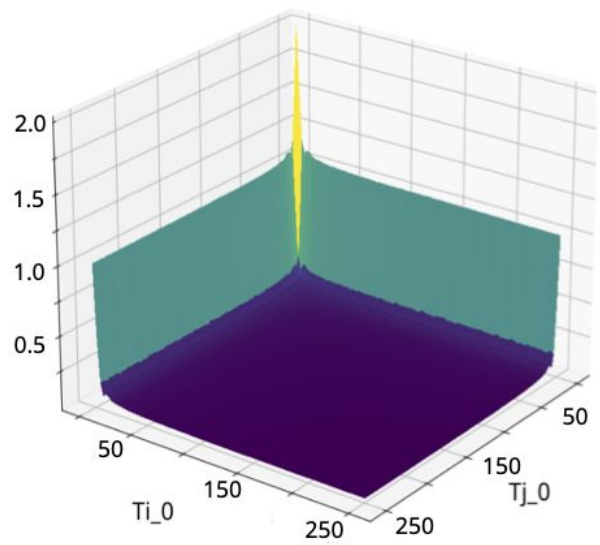
DCC

1. Fusion improves performance, but not always
2. Deciding which operators to fuse is a challenging problem
3. Fusion, when seen as an optimization problem, is guided by a cost model
    a. **Dynamic**: profile and decide
    b. **Static**: analytical function

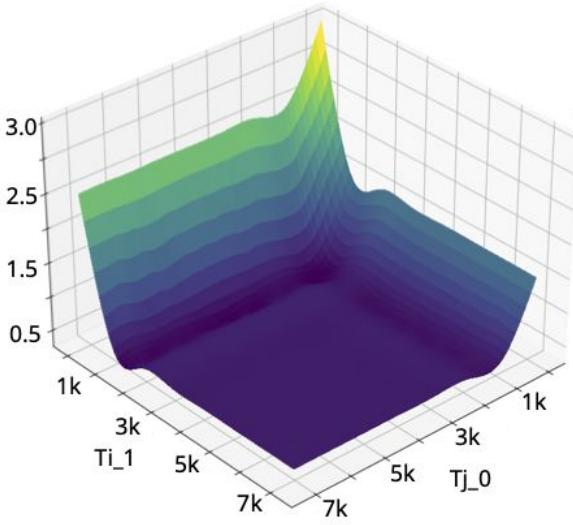
Static Cost Model: Program → Estimated Running Time

DCC

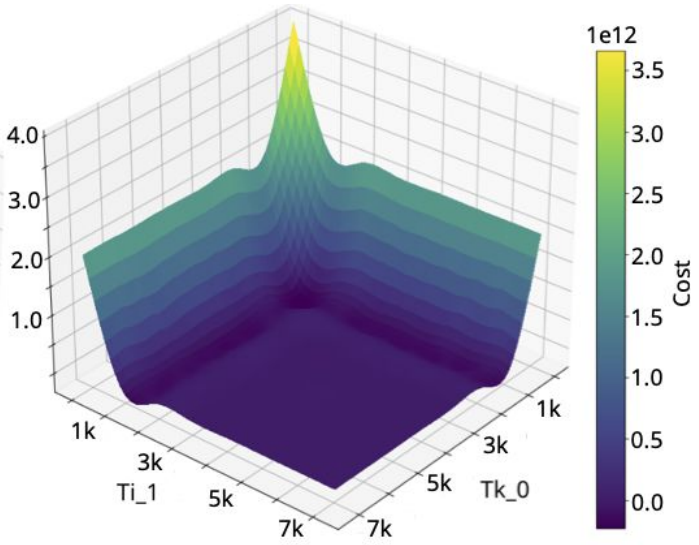
Static Cost Model: Program → Estimated Running Time



(a) Single cache L1.
Cost = Ni*Nj*Nk*(1/Tj_0 + 1/Ti_0 + 2/Nk)



(b) L1 and L2, varying Ti_1 and Tj_0.
Cost = Ni*Nj*Nk*(1/Tj_1 + 1/Ti_1 + 2/Nk) + Ni*Nj*Nk*(2/Tk_0 + 1/Tj_0 + 1/Ti_1)

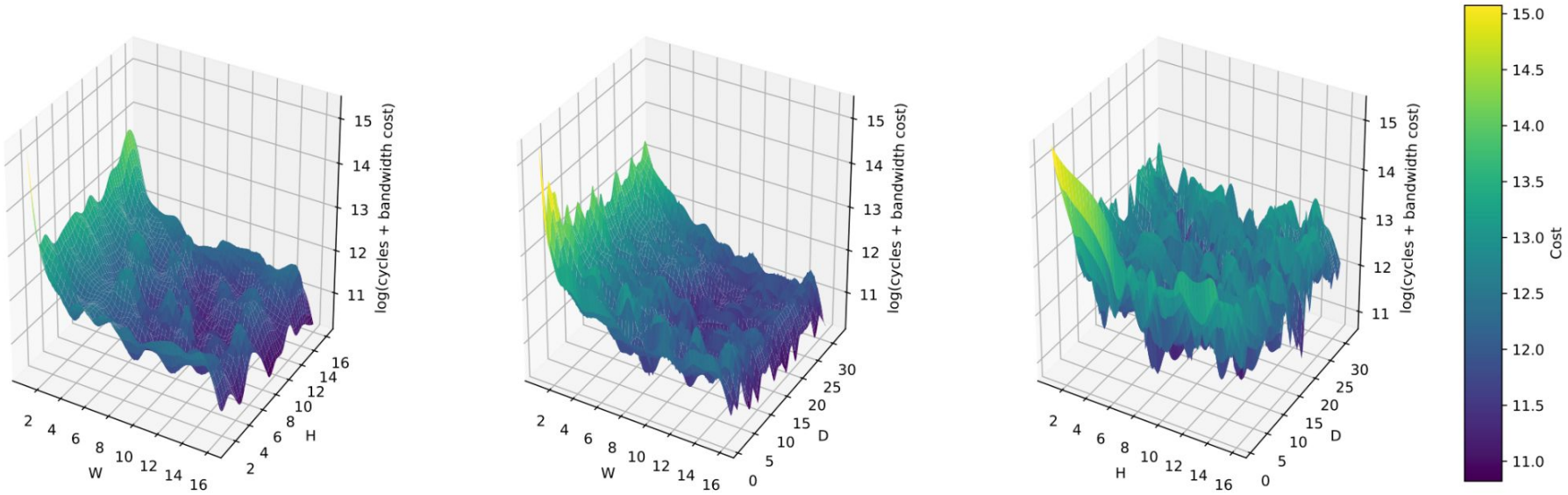


(c) L1 and L2, varying Ti_1 and Tk_0.

Auguste Olivry, Guillaume Iooss, Nicolas Tollenaere, Atanas Rountev, P. Sadayappan, Fabrice Rastello: **IOOpt: automatic derivation of I/O complexity bounds for affine programs**. PLDI 2021: 1187-1202

Static Cost Model: Program → Estimated Running Time

DCC

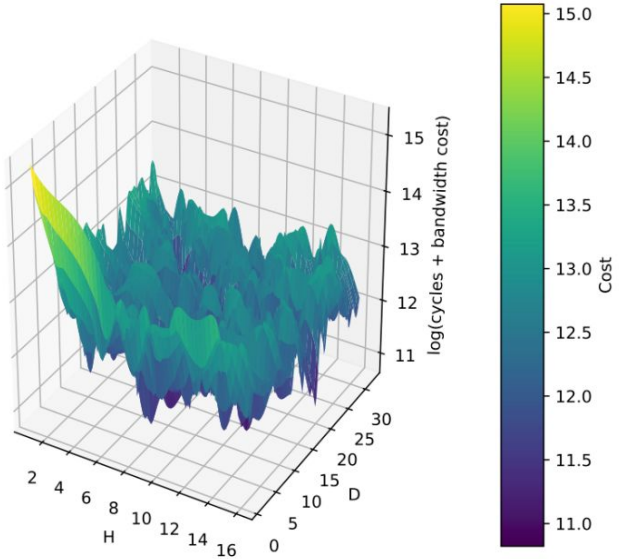


Michael Canesche, Vanderson Martins do Rosário, Edson Borin, Fernando Magno Quintão Pereira: **The Droplet Search Algorithm for Kernel Scheduling**. ACM Trans. Archit. Code Optim. 21(2): 35 (2024)

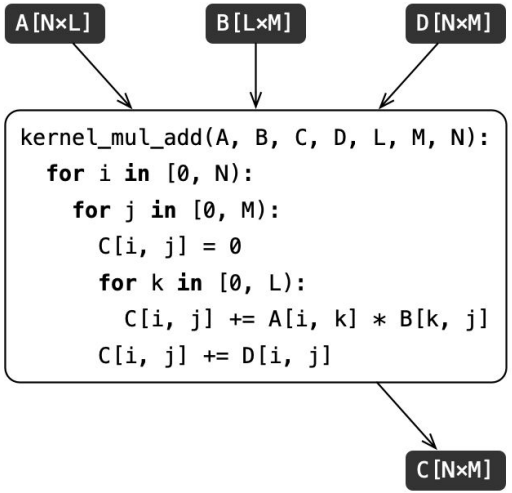
Static Cost Model: Program → Estimated Running Time

DCC

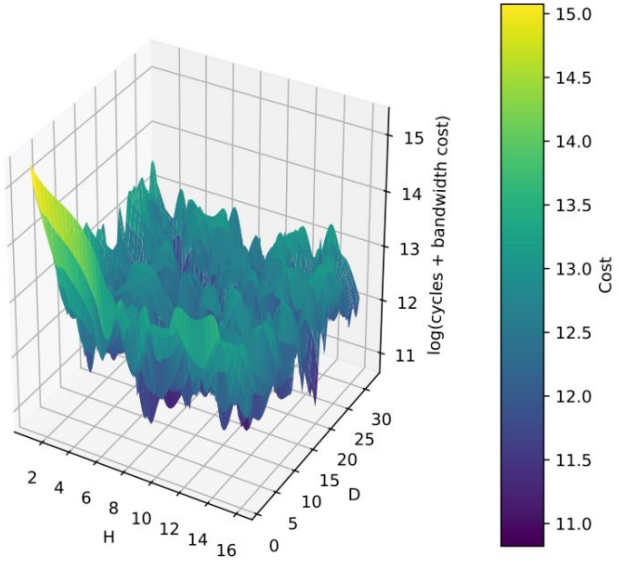
```
def perfModel(L, M, N, SIMD_WIDTH):
    oW, oH, oD = L, M, N
    vw = SIMD_WIDTH - 1
    return TreePerfModel(
        name,
        TPLoops(
            (oW.dim + vw - 1) // vw,
            TPSeq(
                TPLoops([oH, oD, oW]),
                TPCond(
                    cneq(oH % 2, 0),
                    TPLoops([(oD + 1, oH])
                ),
            ),
        ),
    )
```



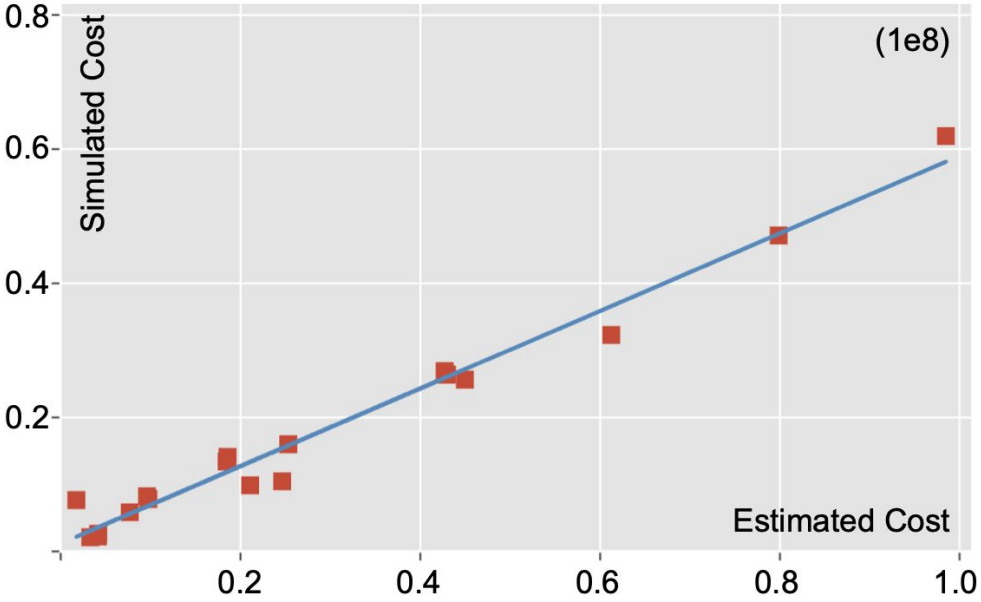
Static Cost Model: Program → Estimated Running Time

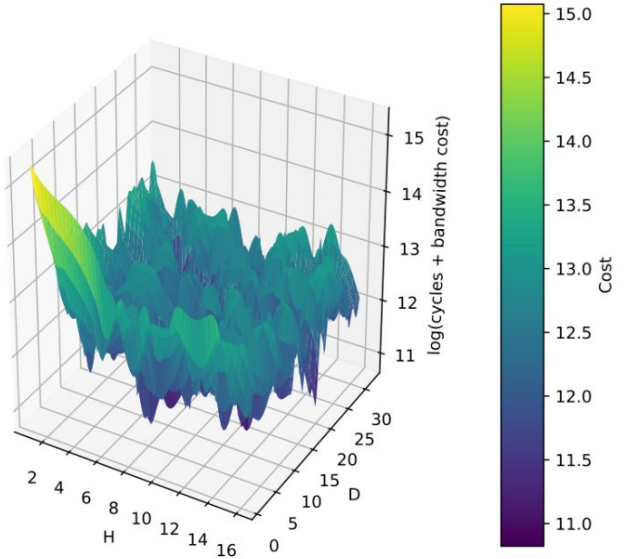



```
def perfModel(L, M, N, SIMD_WIDTH):
    oW, oH, oD = L, M, N
    vw = SIMD_WIDTH - 1
    return TreePerfModel(
        name,
        TPLoops(
            (oW.dim + vw - 1) // vw,
            TPSeq(
                TPLoops([oH, oD, oW]),
                TPCond(
                    cneq(oH % 2, 0),
                    TPLoops([(oD + 1, oH])
                ),
            ),
        ),
    )
```



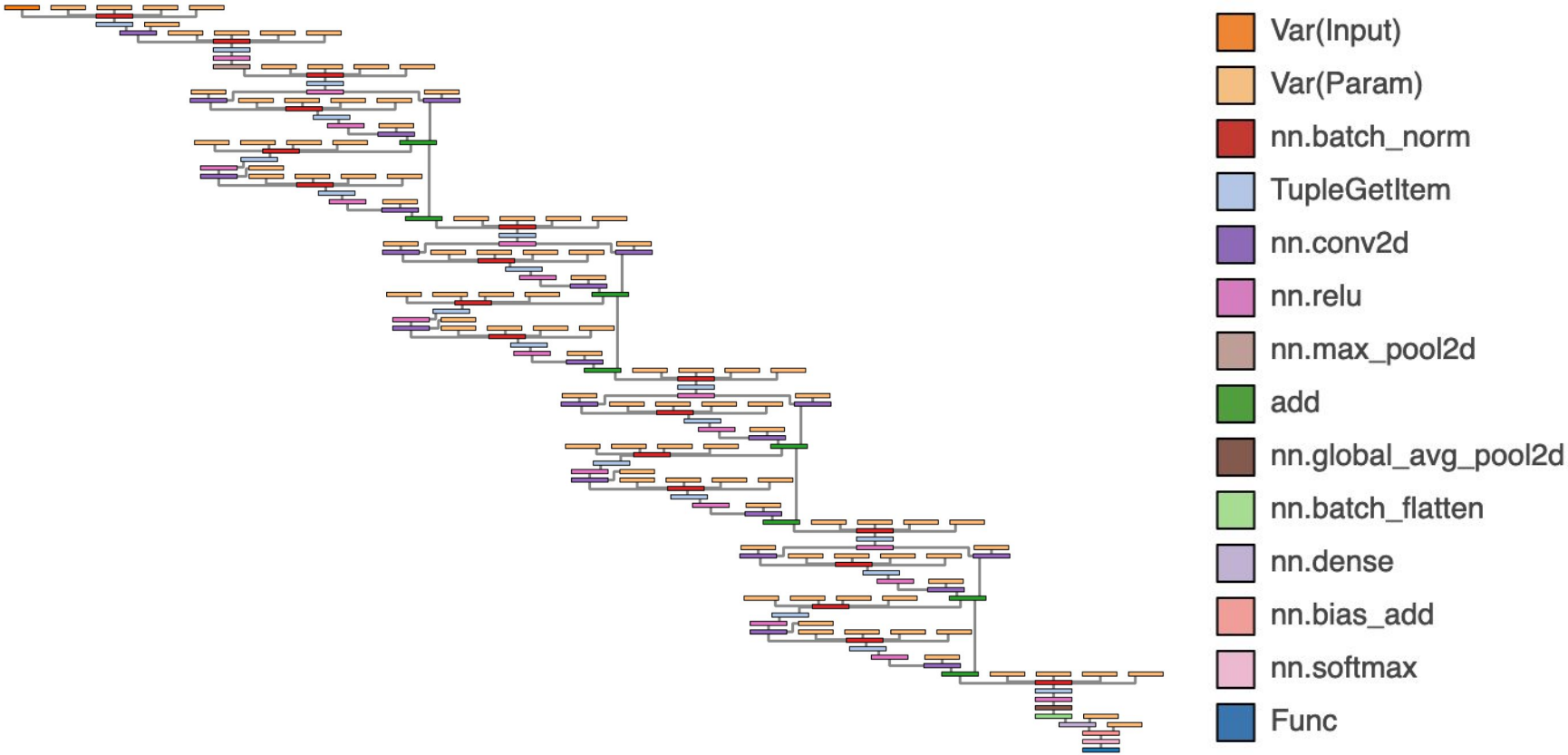
Static Cost Model: Program → Estimated Running Time





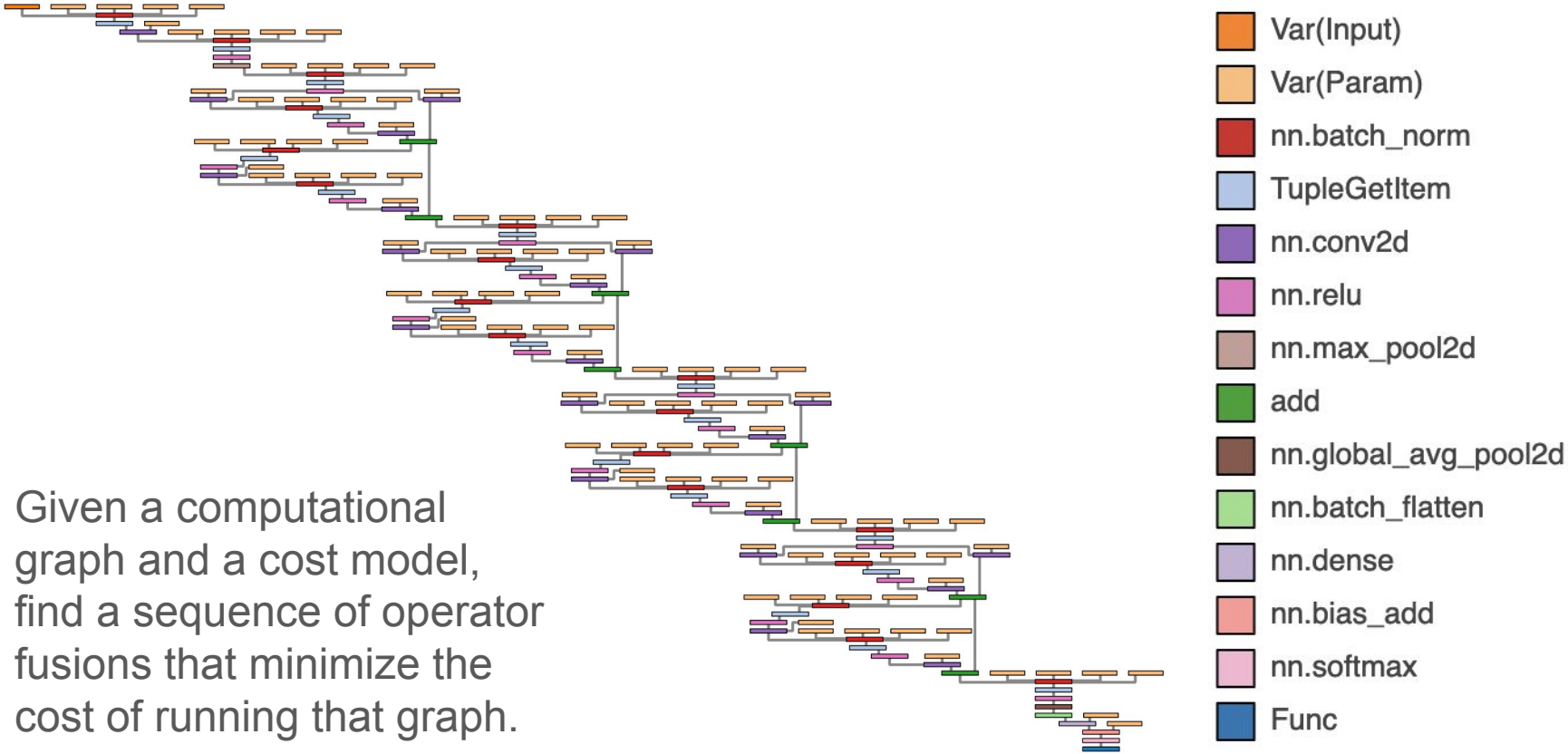
Fusion as an Optimization Problem

DCC



Fusion as an Optimization Problem

DCC



Given a computational graph and a cost model, find a sequence of operator fusions that minimize the cost of running that graph.

Fusion as an Optimization Problem

DCC

Exhaustive:

Xuyi Cai, Ying Wang, Lei Zhang: **Optimus: An Operator Fusion Framework for Deep Neural Networks**. ACM Trans. Embed. Comput. Syst. 22(1): 1:1-1:26 (2023)

Fusion as an Optimization Problem

DCC

Exhaustive:

Xuyi Cai, Ying Wang, Lei Zhang: **Optimus: An Operator Fusion Framework for Deep Neural Networks**. ACM Trans. Embed. Comput. Syst. 22(1): 1:1-1:26 (2023)

Evolutionary:

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, Ion Stoica: **Ansor: Generating High-Performance Tensor Programs for Deep Learning**. OSDI 2020: 863-879

Fusion as an Optimization Problem

DCC

Exhaustive:

Xuyi Cai, Ying Wang, Lei Zhang: **Optimus: An Operator Fusion Framework for Deep Neural Networks**. ACM Trans. Embed. Comput. Syst. 22(1): 1:1-1:26 (2023)

Evolutionary:

Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, Ion Stoica: **Ansor: Generating High-Performance Tensor Programs for Deep Learning**. OSDI 2020: 863-879

Minimum Spanning Tree:

Michael Canesche, Vanderson Rosário, Edson Borin, Fernando Magno Quintão Pereira: **Fusion of Operators of Computational Graphs via Greedy Clustering: The XNNC Experience**. CC 2025: 1-13

The Minimum Spanning Tree Approach

DCC

The Minimum Spanning Tree Approach

DCC

- Greedy algorithm guided by cost model

The Minimum Spanning Tree Approach

DCC

- Greedy algorithm guided by cost model
- Group nodes into clusters

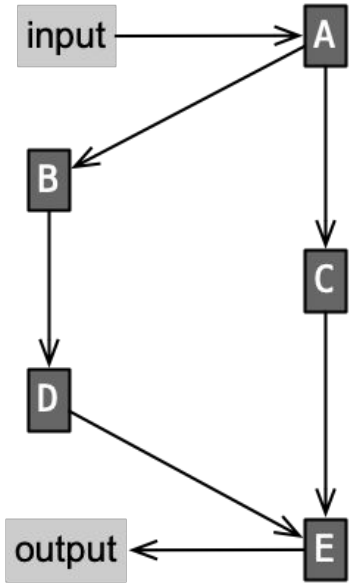
The Minimum Spanning Tree Approach

DCC

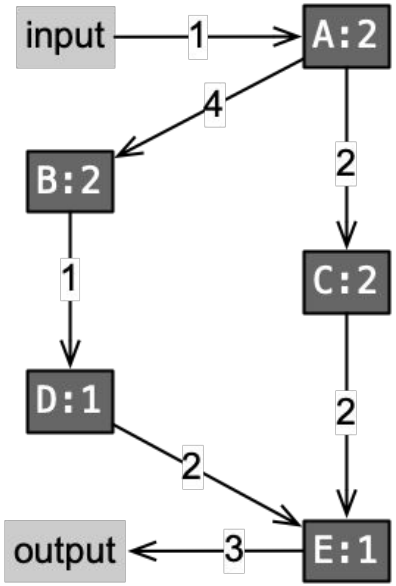
- Greedy algorithm guided by cost model
- Group nodes into clusters
- Fuse the two most profitable clusters (Until no more fusions are possible)

The Minimum Spanning Tree Approach

DCC



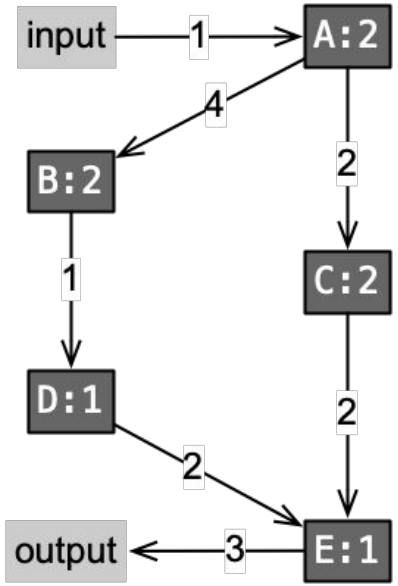
The Minimum Spanning Tree Approach



Cost model:

- Edges: cost of data transfer
- Vertices: cost of computation

The Minimum Spanning Tree Approach



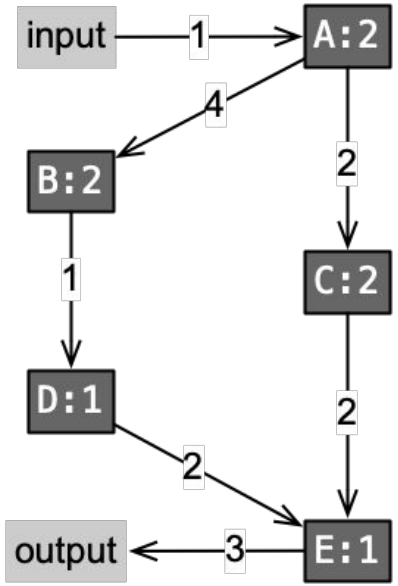
Cost model:

- Edges: cost of data transfer
- Vertices: cost of computation

(Non-) Properties:

- Non-associative
  - cost(fuse(A, fuse(B, C))) != cost(fuse(fuse(A, B), C))

The Minimum Spanning Tree Approach



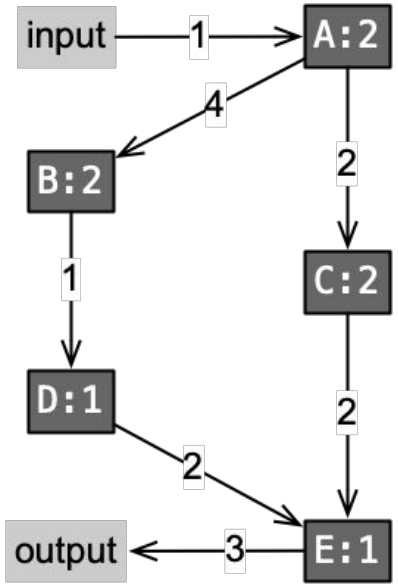
Cost model:

- Edges: cost of data transfer
- Vertices: cost of computation

(Non-) Properties:

- Non-associative
  - cost(fuse(A, fuse(B, C))) != cost(fuse(fuse(A, B), C))
- No Matroid Structure
  - Greedy choices are not guaranteed to be optimal

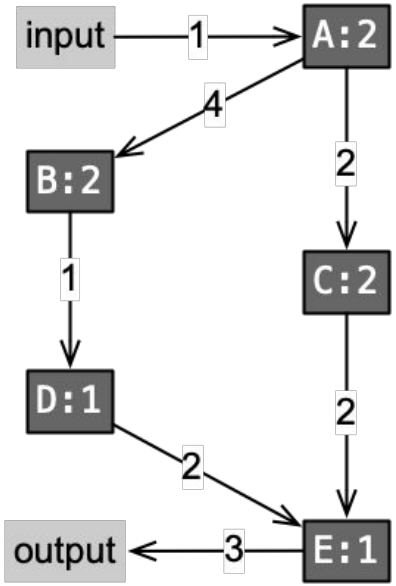
The Minimum Spanning Tree Approach



Assumption (example only!)

- Cost(edges) < 5

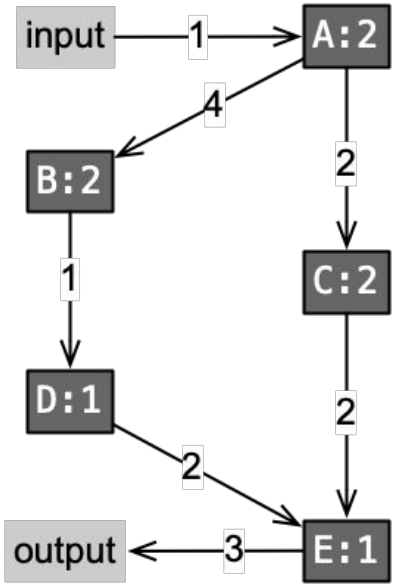
The Minimum Spanning Tree Approach



Assumption (example only!)

- Cost(edges) < 5
- Eg.: memory capacity == 4

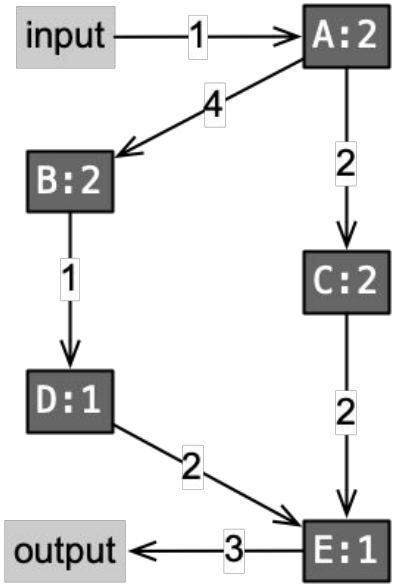
The Minimum Spanning Tree Approach



Assumption (example only!)

- Cost(edges) < 5
- Eg.: memory capacity == 4
- Cost(fuse(V_A, V_B)) == Cost(V_A) + Cost(V_B)

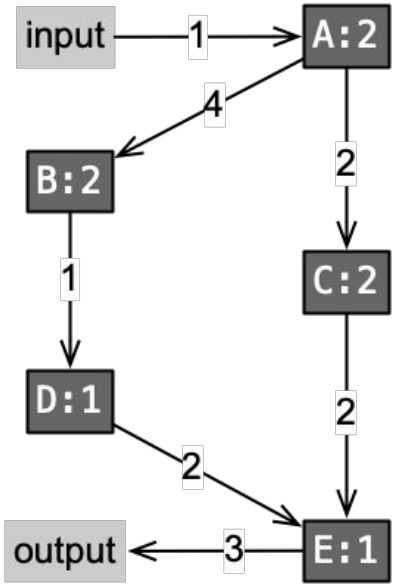
The Minimum Spanning Tree Approach



Assumption (example only!)

- Cost(edges) < 5
- Eg.: memory capacity == 4
- Cost(fuse(V_A, V_B)) == Cost(V_A) + Cost(V_B)
  - "cost is homomorphic with regards to fusion"

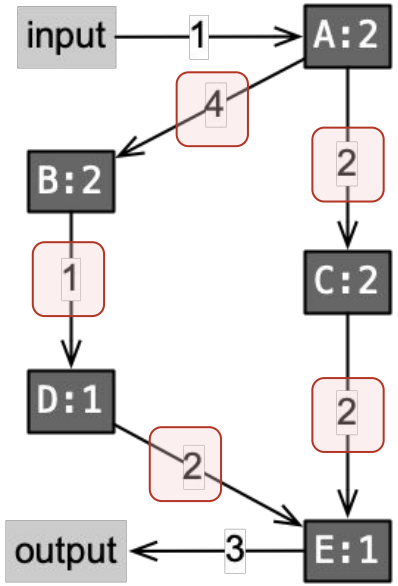
The Minimum Spanning Tree Approach



Assumption (example only!)

- Cost(edges) < 5
- Eg.: memory capacity == 4
- Cost(fuse(V_A, V_B)) == Cost(V_A) + Cost(V_B)
- Cost(fuse($V_A \rightarrow V_B$)) == 0

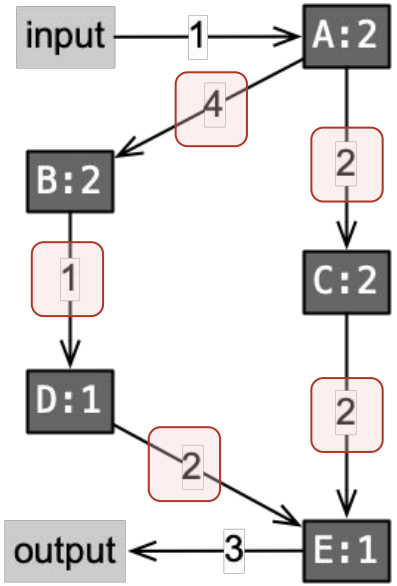
The Minimum Spanning Tree Approach



Edges: 15, Vertices: 8

- Five fusion candidates

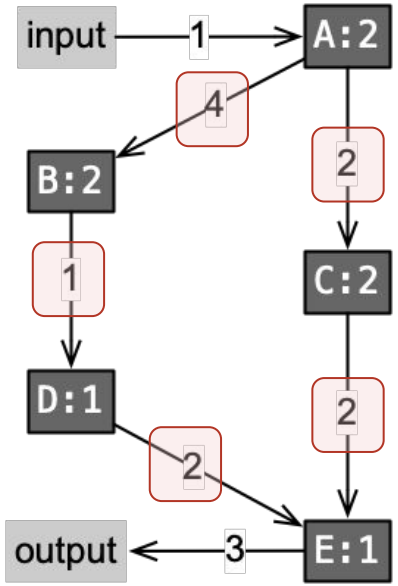
The Minimum Spanning Tree Approach



Edges: 15, Vertices: 8

- Five fusion candidates
- The most profitable?

The Minimum Spanning Tree Approach



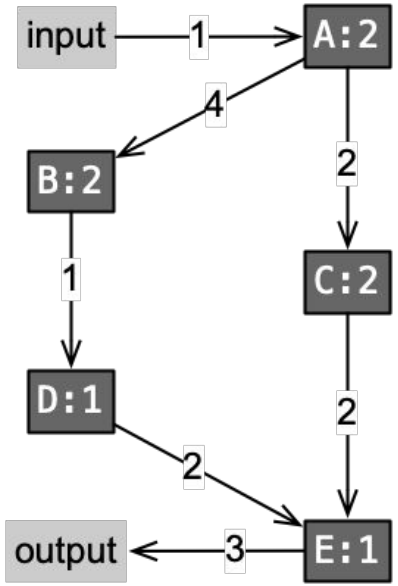
Edges: 15, Vertices: 8

Reduces total cost the most,

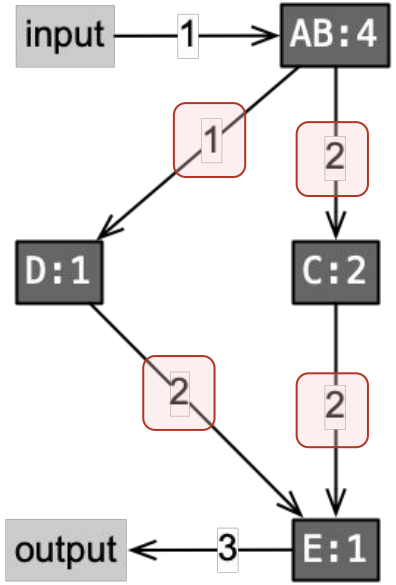
e.g.: C(edges) + C(vertices)

- Five fusion candidates
- The most profitable?

The Minimum Spanning Tree Approach



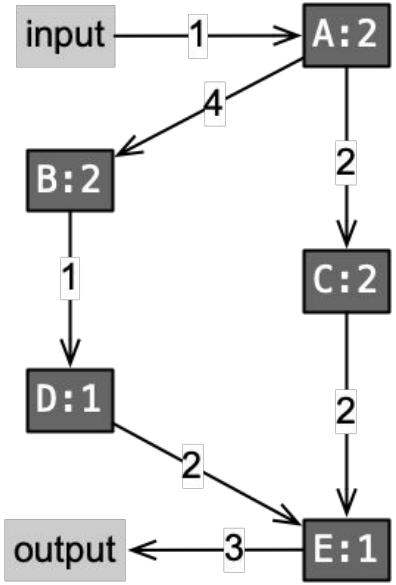
Edges: 15, Vertices: 8



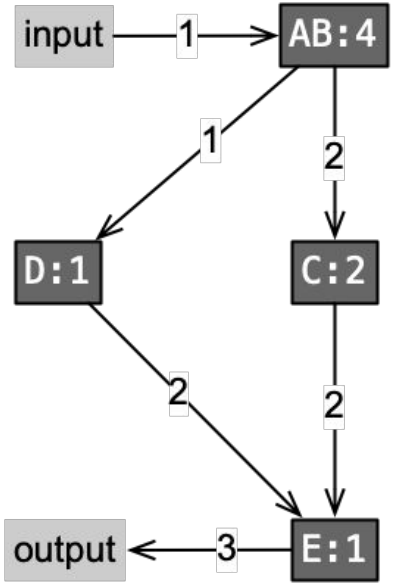
Edges: 11, Vertices: 8

- Four fusion candidates
- The most profitable?

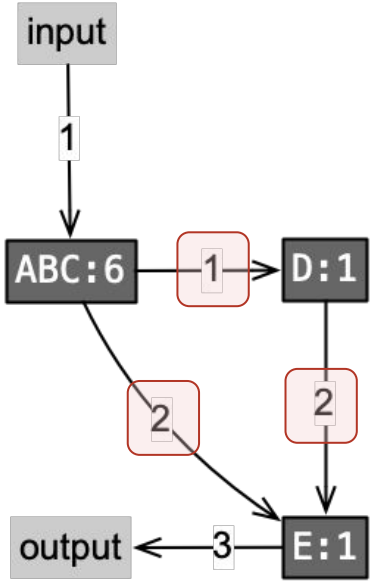
The Minimum Spanning Tree Approach



Edges: 15, Vertices: 8

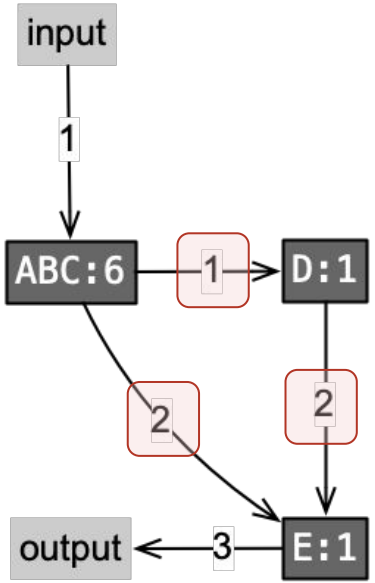


Edges: 11, Vertices: 8



Edges: 9, Vertices: 8

The Minimum Spanning Tree Approach



When no more fusions are possible, we are done

Edges: 9, Vertices: 8

How Good is the Cluster-Based Algorithm?

DCC

How Good is the Cluster-Based Algorithm?

DCC

- Benchmarks: 19 convolutional neural networks

How Good is the Cluster-Based Algorithm?

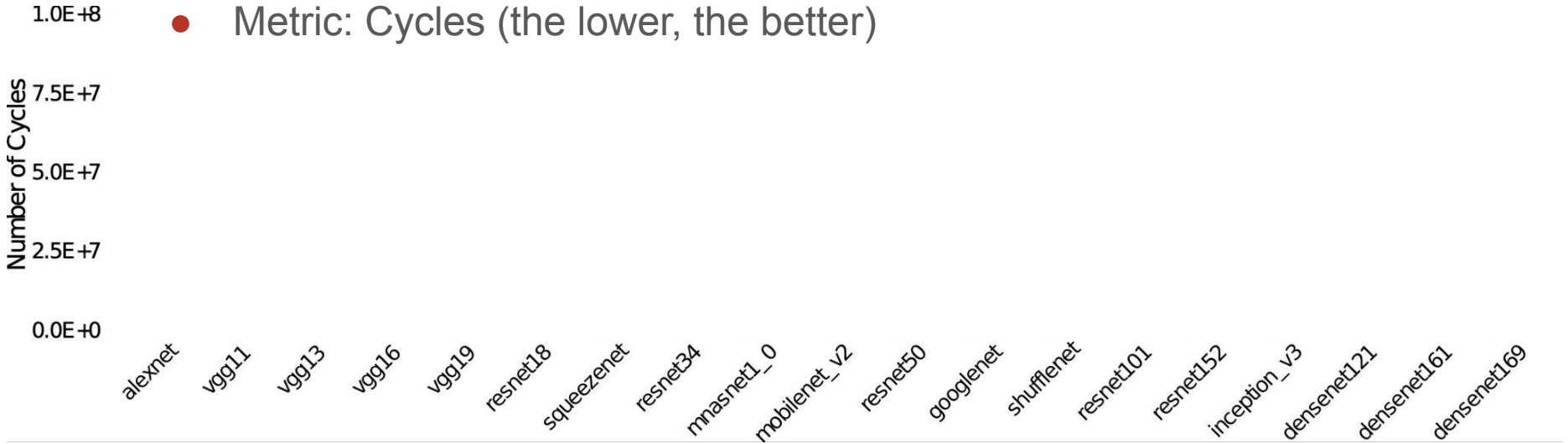
DCC

- Benchmarks: 19 convolutional neural networks
- Same image (batch=1, channels=3, height=224, width=224)

How Good is the Cluster-Based Algorithm?

DCC

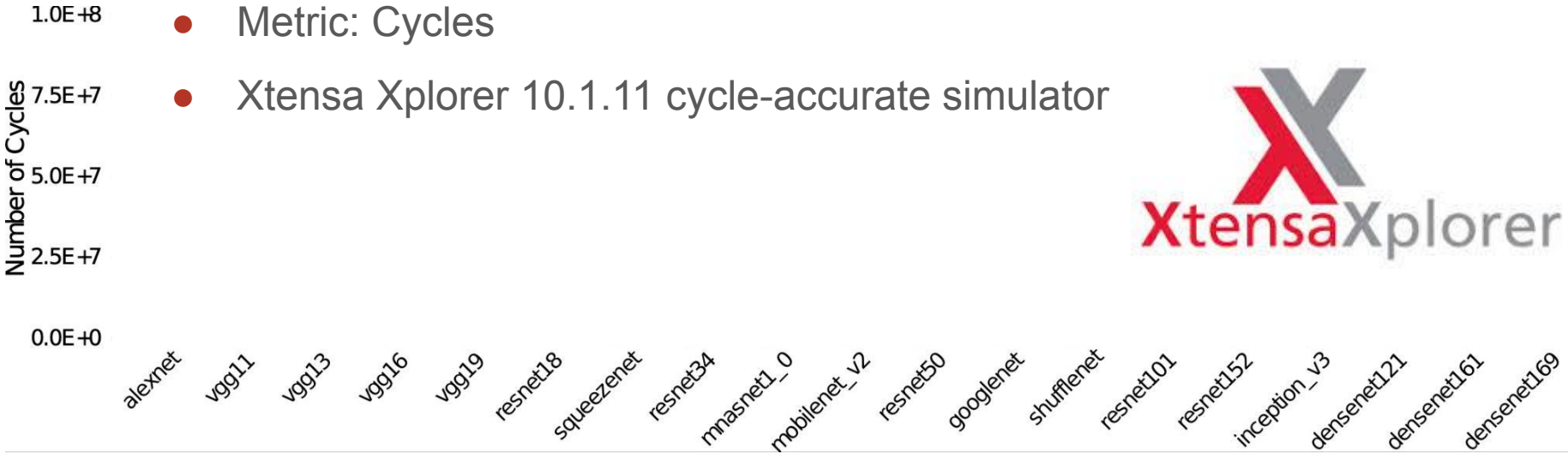
- Metric: Cycles (the lower, the better)



How Good is the Cluster-Based Algorithm?

DCC

- Metric: Cycles
- Xtensa Xplorer 10.1.11 cycle-accurate simulator

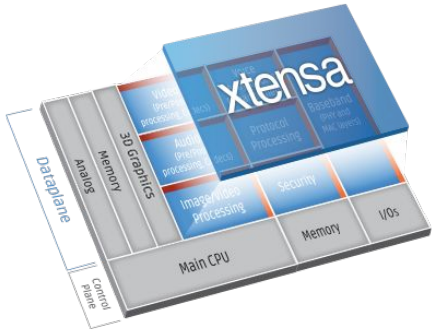


XtensaXplorer

How Good is the Cluster-Based Algorithm?

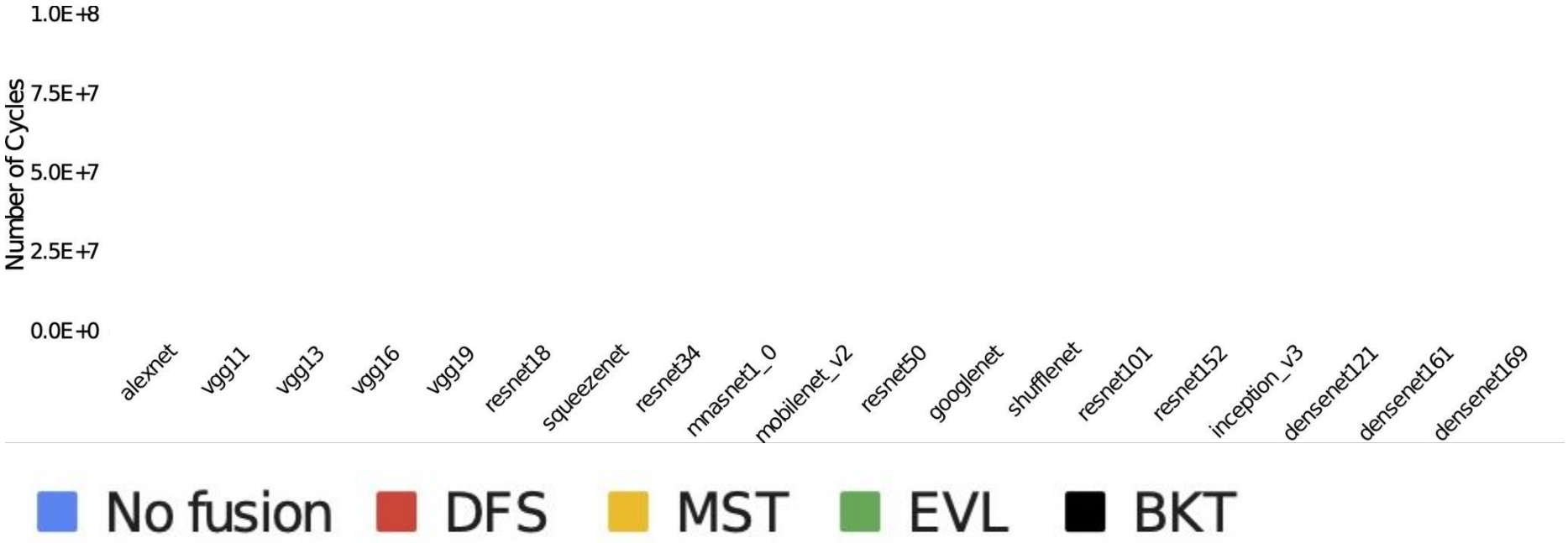
DCC

- Metric: Cycles
- Xtensa Xplorer 10.1.11 cycle-accurate simulator
- P6 Digital Signal Processor



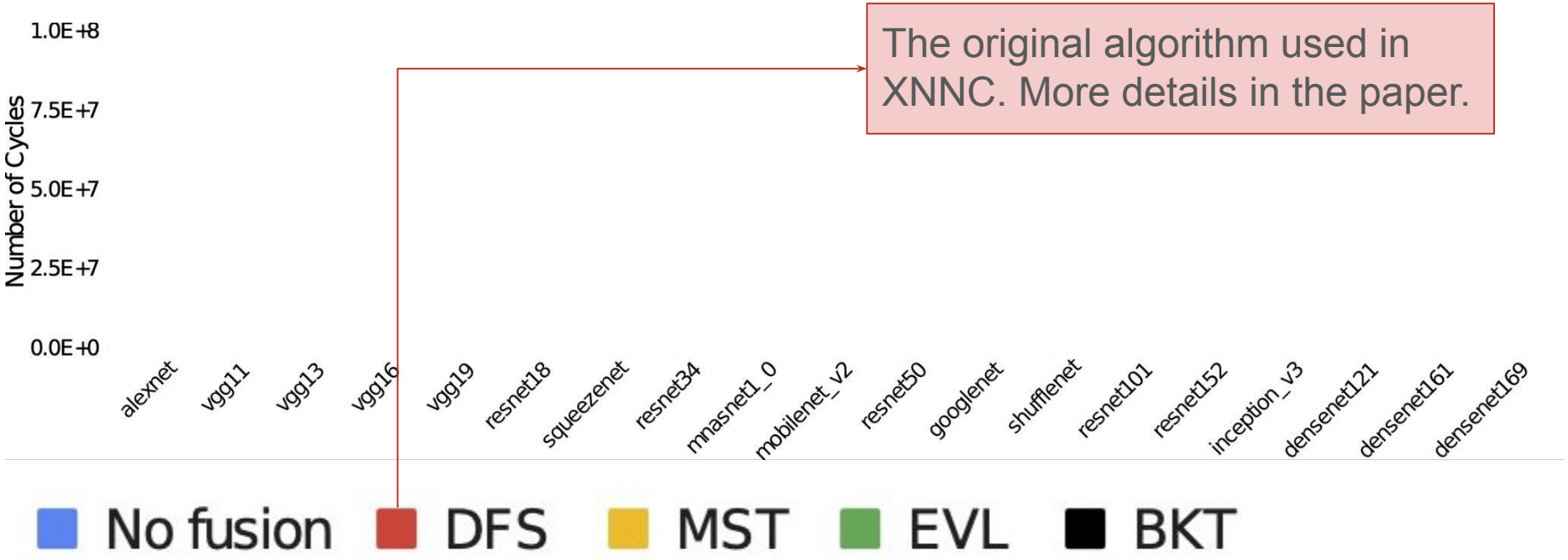
How Good is the Cluster-Based Algorithm?

DCC

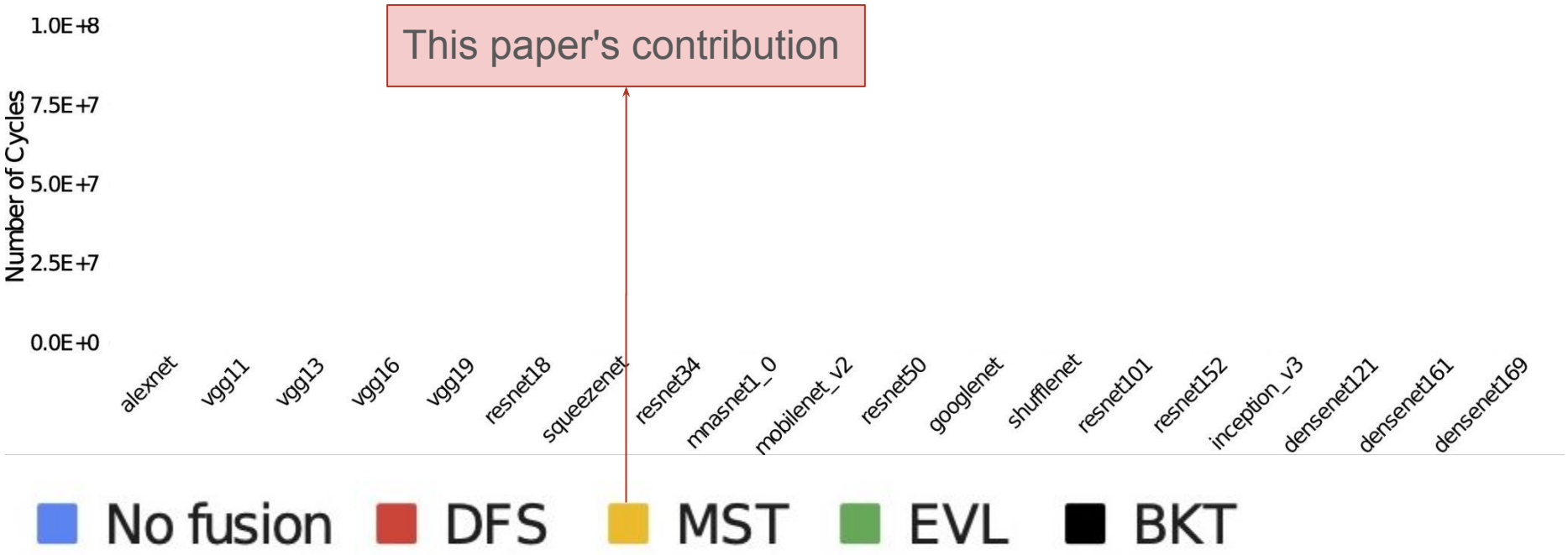


How Good is the Cluster-Based Algorithm?

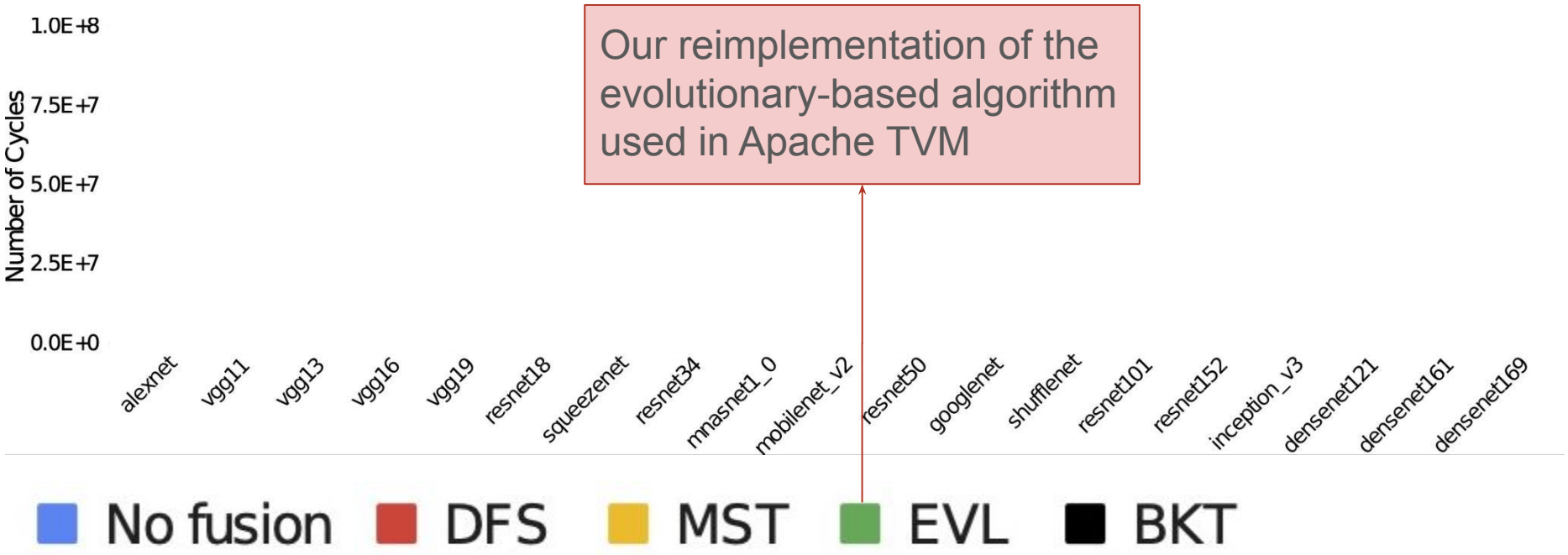
DCC



How Good is the Cluster-Based Algorithm?



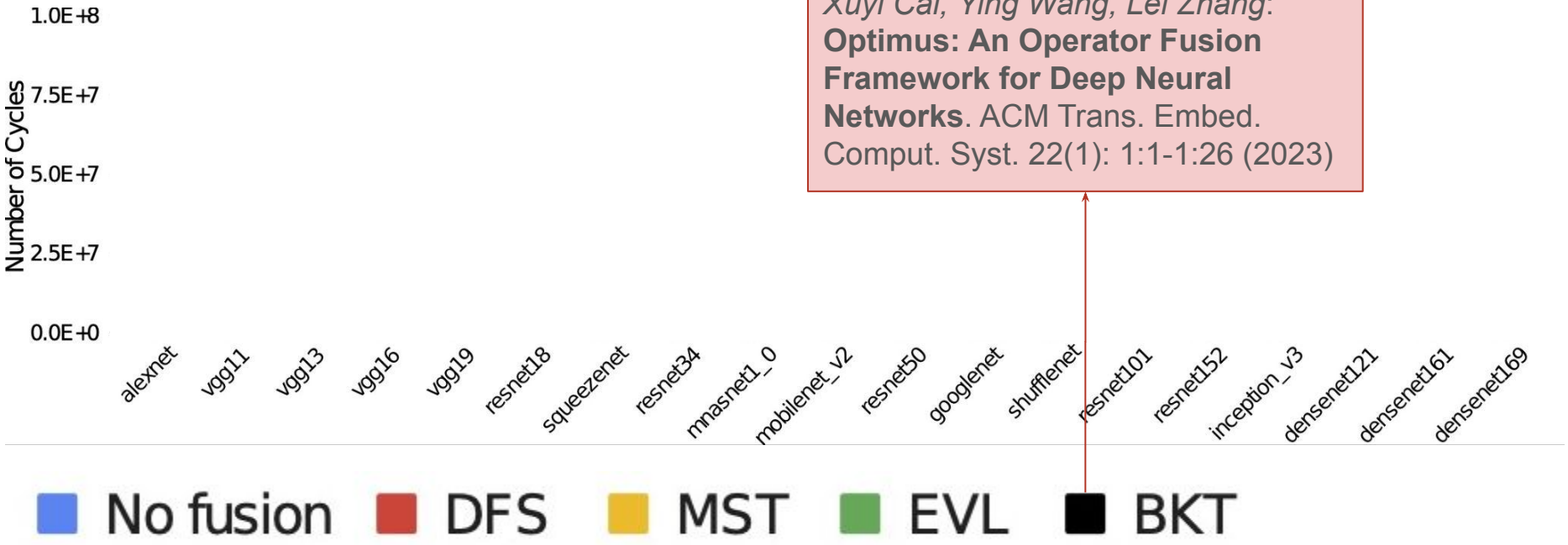
How Good is the Cluster-Based Algorithm?



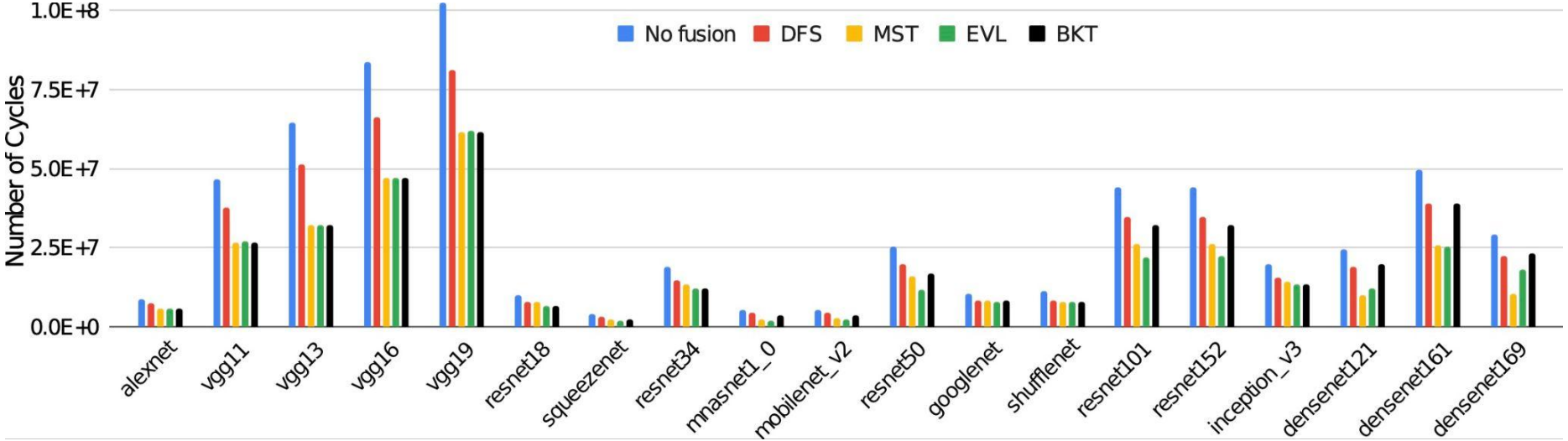
How Good is the Cluster-Based Algorithm?

DCC

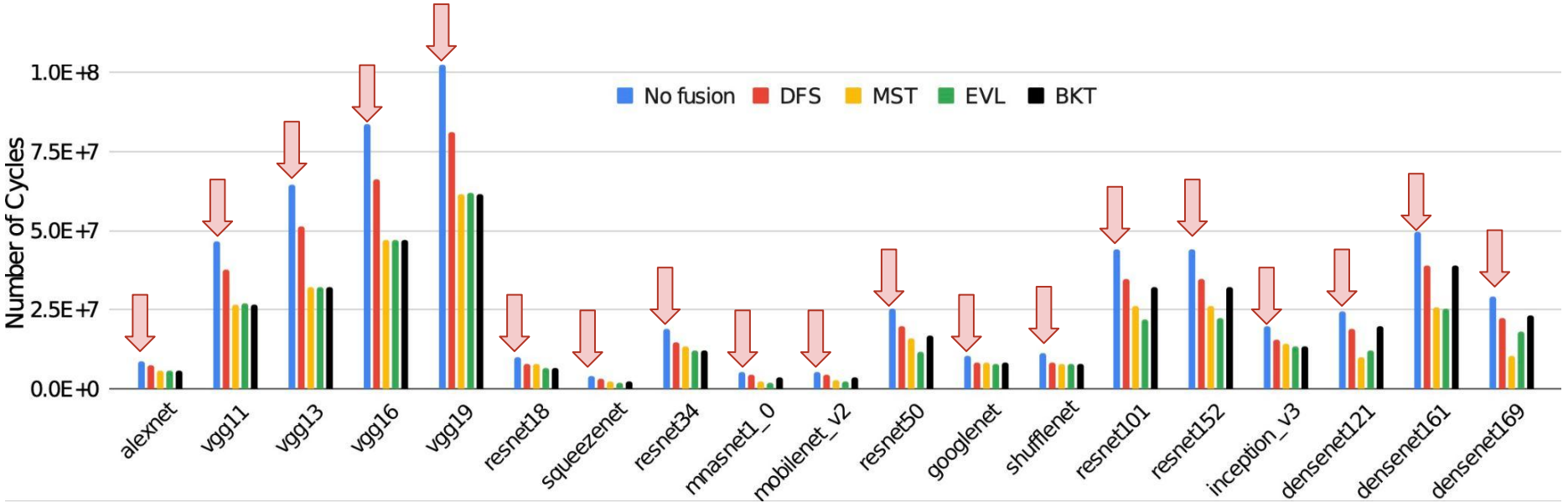
Xuyi Cai, Ying Wang, Lei Zhang: **Optimus: An Operator Fusion Framework for Deep Neural Networks**. ACM Trans. Embed. Comput. Syst. 22(1): 1:1-1:26 (2023)



How Good is the Cluster-Based Algorithm?



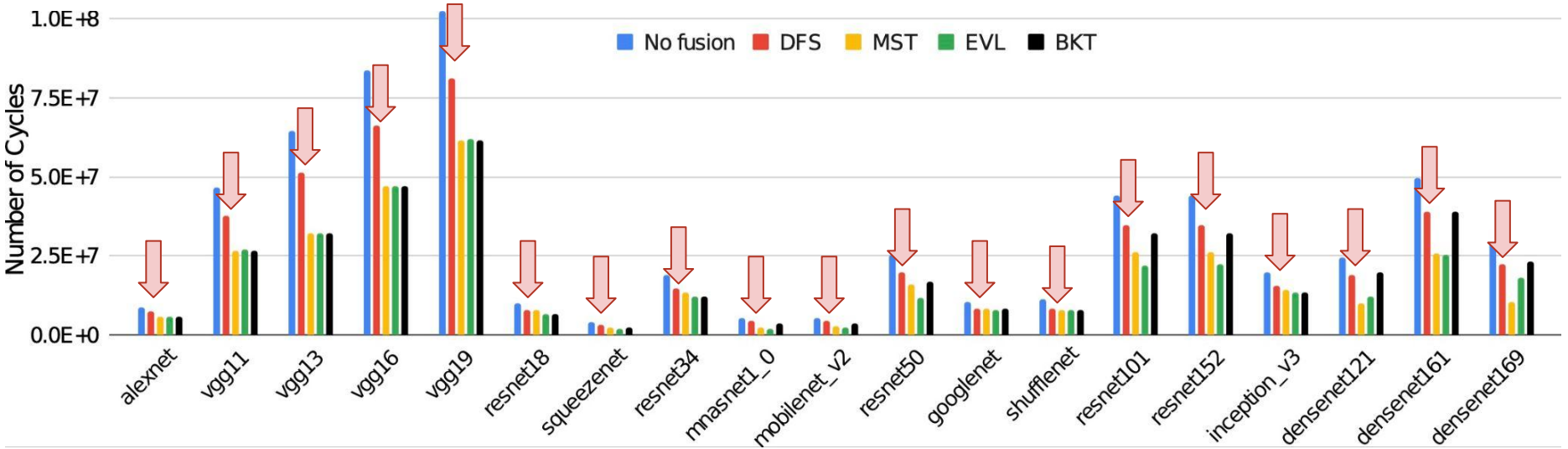
How Good is the Cluster-Based Algorithm?



Operator fusion, regardless of the approach, is usually good

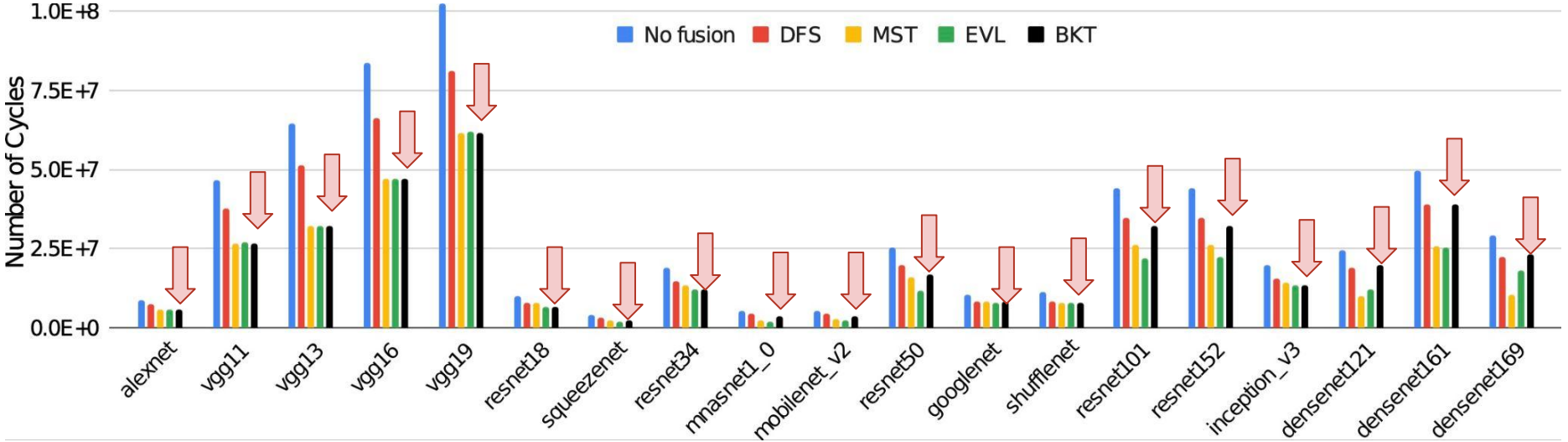
How Good is the Cluster-Based Algorithm?

DCC



The new algorithm, MST, improves on XNNC's old algorithm (DFS) by 39%

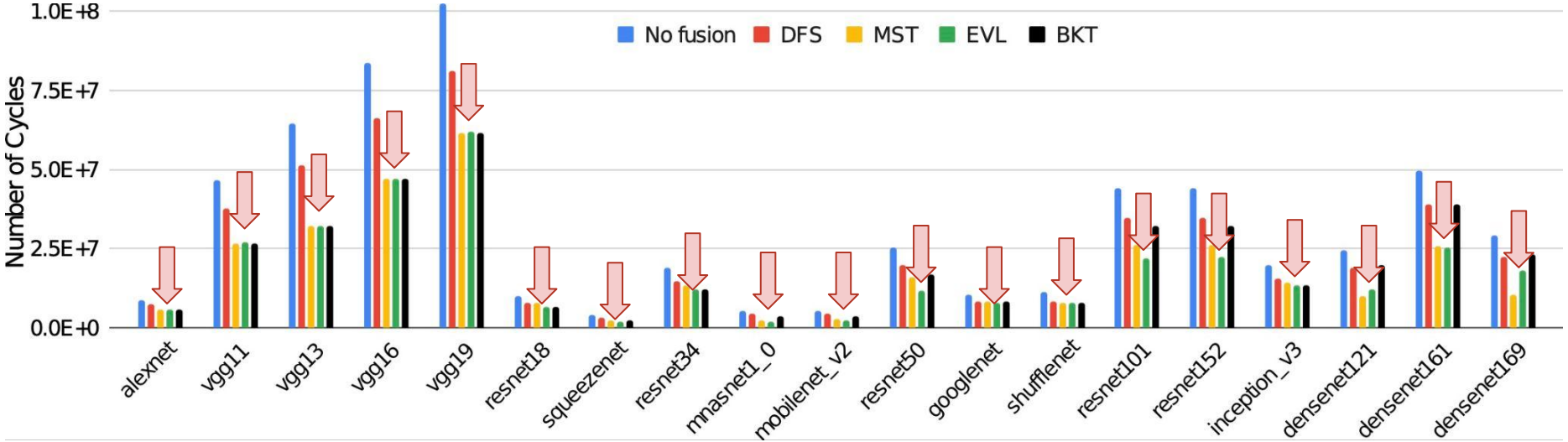
How Good is the Cluster-Based Algorithm?



The exhaustive BKT did not yield the best performance (due to time-outs)

How Good is the Cluster-Based Algorithm?

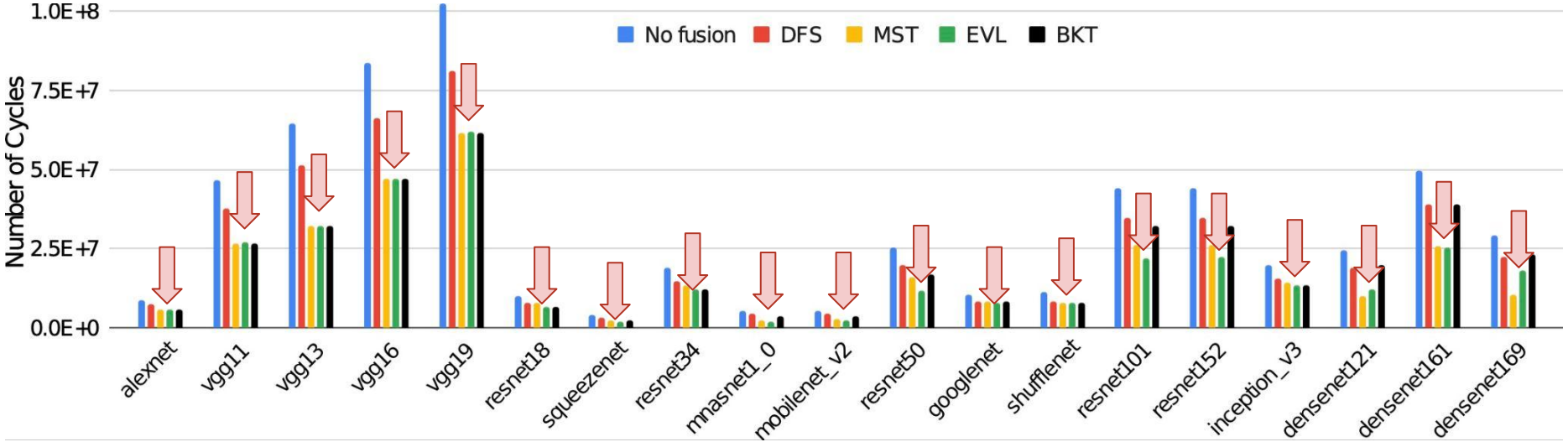
DCC



There was no statistical difference between MST and EVL

How Good is the Cluster-Based Algorithm?

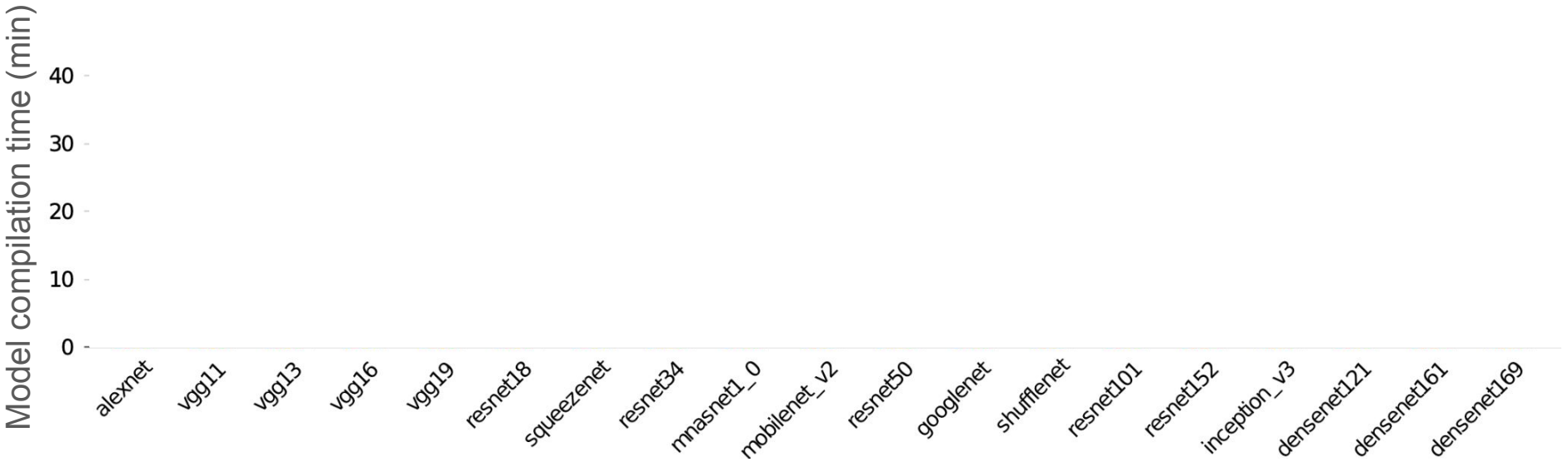
DCC



There was no statistical difference between MST and EVL.
But MST is usually faster.

How Good is the Cluster-Based Algorithm?

DCC

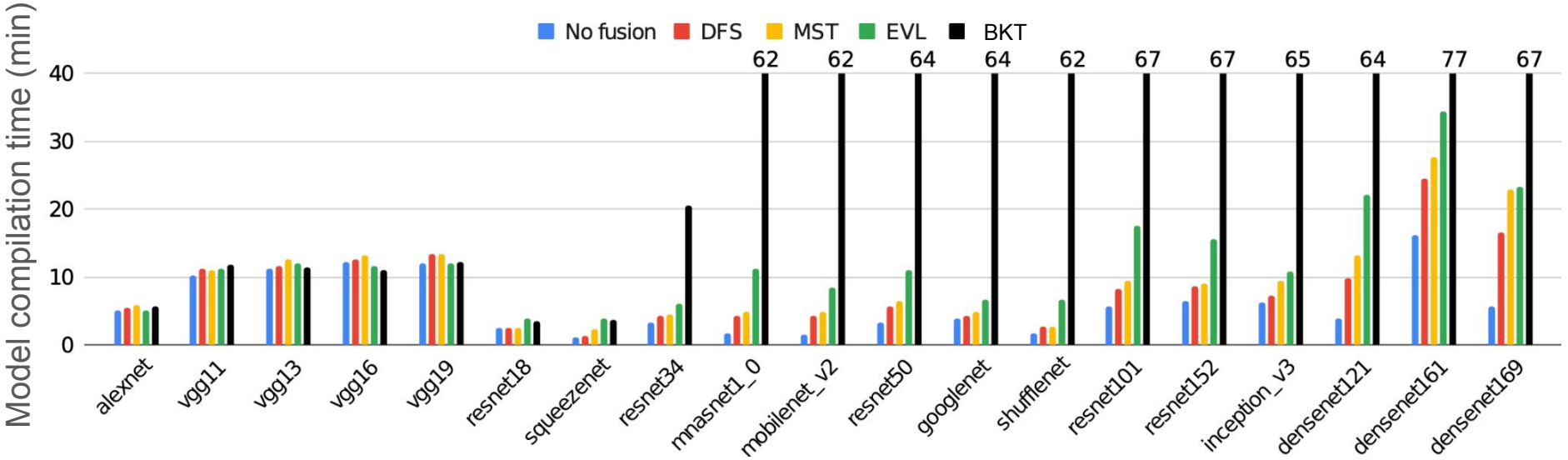


There was no statistical difference between MST and EVL.

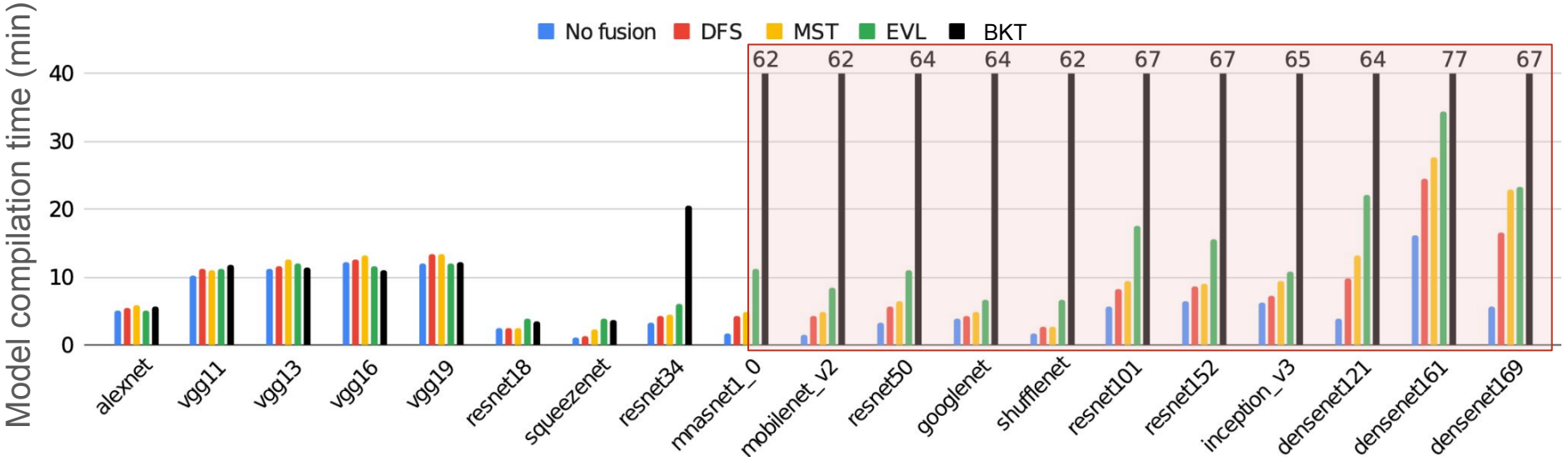
But MST is usually faster.

How Good is the Cluster-Based Algorithm?

DCC

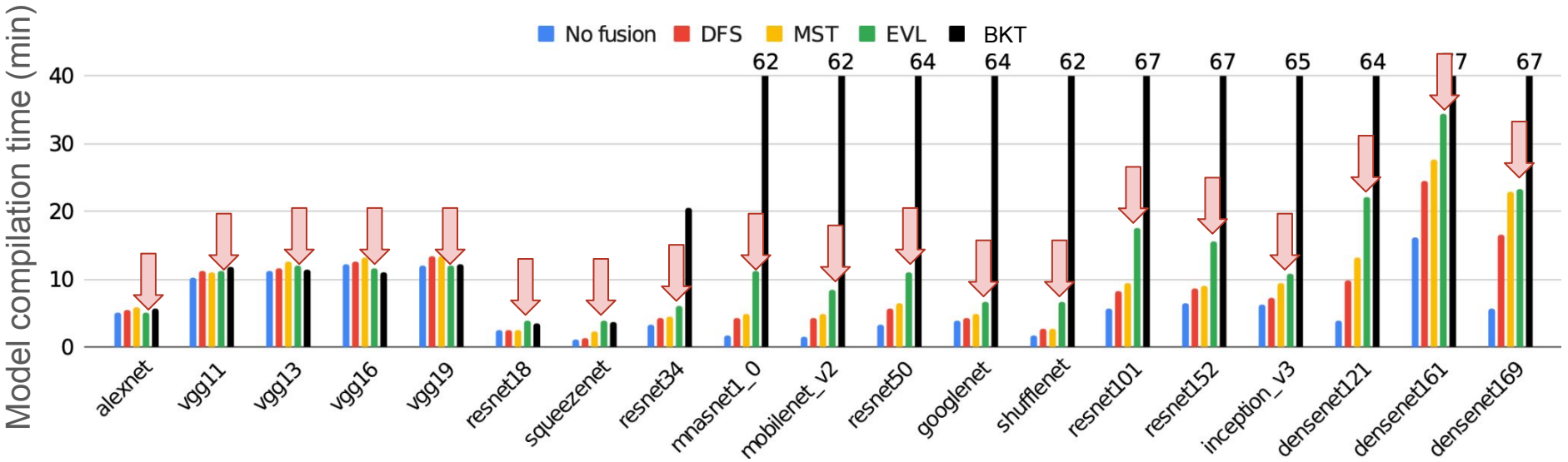


How Good is the Cluster-Based Algorithm?



The exhaustive approach experienced frequent timeouts (within one hour).

How Good is the Cluster-Based Algorithm?



The evolutionary algorithm (EVL) is 29% slower than MST

How Good is the Cluster-Based Algorithm?

DCC

- MST is 29% faster than EVL, but generates code equality as good

How Good is the Cluster-Based Algorithm?

DCC

- MST is 29% faster than EVL, but generates code equality as good
- MST is 11% slower than DFS, but generates code that is 39% faster

How Good is the Cluster-Based Algorithm?

DCC

- MST is 29% faster than EVL, but generates code equality as good
- MST is 11% slower than DFS, but generates code that is 39% faster
- MST's implementation is a bit larger than DFS's but much shorter than EVL's

How Good is the Cluster-Based Algorithm?

DCC

- MST is 29% faster than EVL, but generates code equality as good
- MST is 11% slower than DFS, but generates code that is 39% faster
- MST's implementation is a bit larger than DFS's but much shorter than EVL's
- The exhaustive approach seems unpractical in XNNC's environment

How Good is the Cluster-Based Algorithm?

DCC

- MST is 29% faster than EVL, but generates code equality as good
- MST is 11% slower than DFS, but generates code that is 39% faster
- MST's implementation is a bit larger than DFS's but much shorter than EVL's
- The exhaustive approach seems unpractical in XNNC's environment
  - Ex.: densenet169 has 1376 operators and 2231 edges

References & Contact

DCC

Fernando Pereira

fernando@dcc.ufmg.br

http://lac.dcc.ufmg.br

Michael Canesche, Vanderson Rosário, Edson Borin, Fernando Magno Quintão Pereira: **Fusion of Operators of Computational Graphs via Greedy Clustering: The XNNC Experience**. CC 2025: 1-13