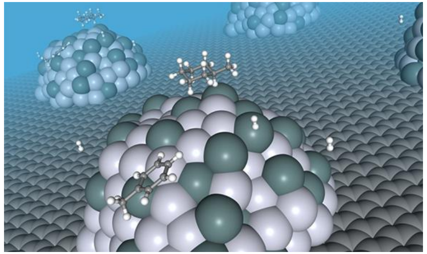
PFVM - A Neural Network Compiler that uses ONNX as its intermediate representation

Preferred Networks, Zijian Xu

About Preferred Networks

We are solving real-world problems by deep learning

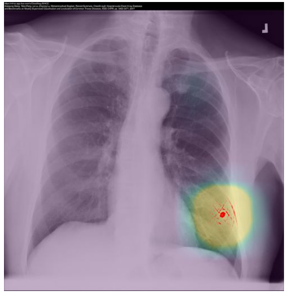


Material discovery

OPTUNA

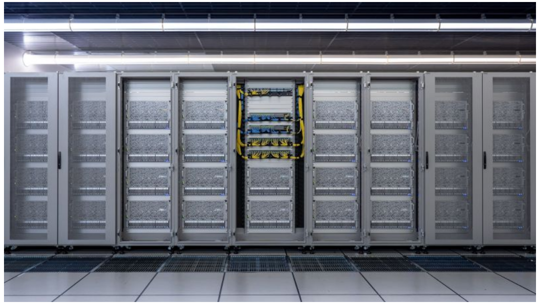
CuPy

Chainer



Medical image analysis

Character generation



MN-3 supercomputer

About Me

Zijian Xu

I optimize neural network models

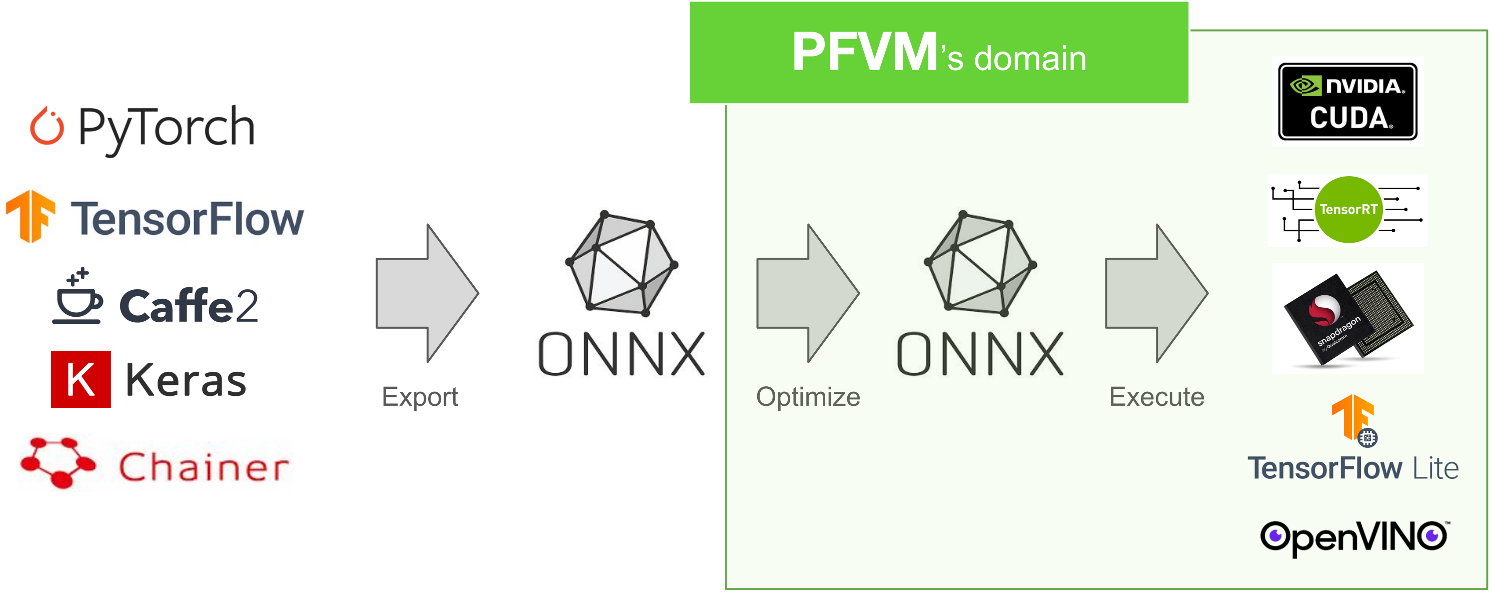
Today’s Topic

Want to share

- a use case of ONNX for model optimization
- our motivation to have a solid shape inference

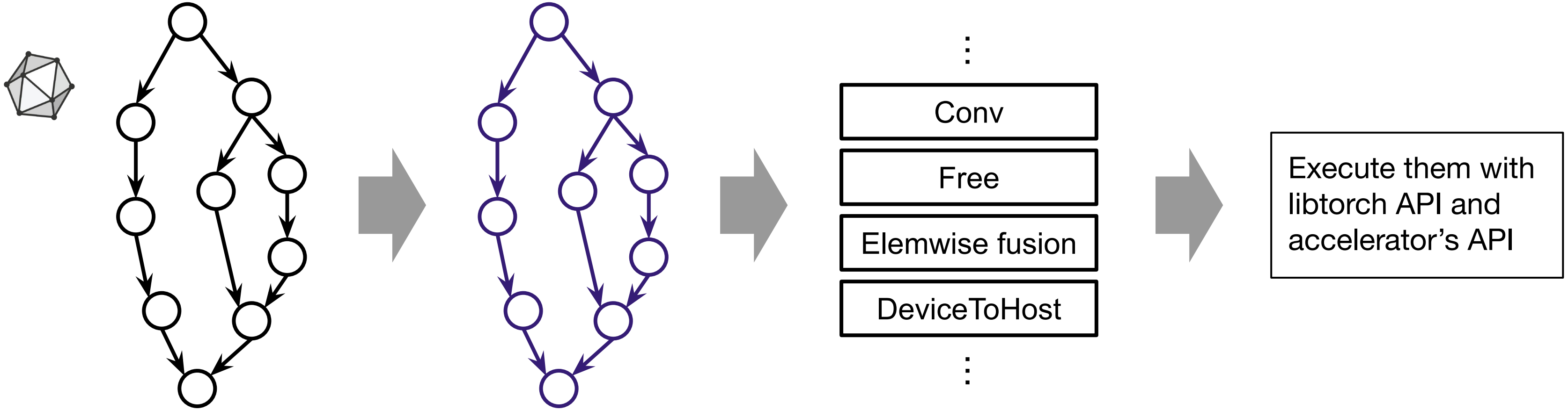
PFVM

PFN's inhouse neural network compiler and runtime



PFVM

PFN's inhouse neural network compiler and runtime



Input ONNX graph

Backward generation and model optimization

Schedule graph and emit operators

Why ONNX as IR?

- Stable and well documented
- Focused on interface and doesn’t do too much
- Shape inference
- Useful test cases

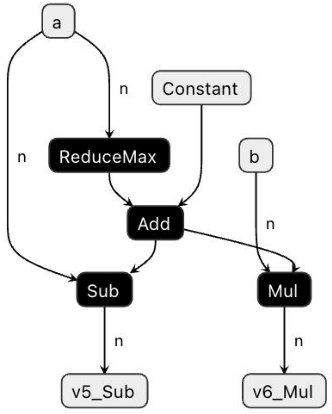
Why ONNX as IR?

- Stable and well documented
- Focused on interface and doesn’t do too much
- Shape inference
- Useful test cases

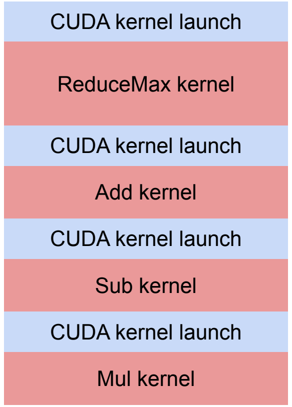
Let’s see how shape inference is important in model optimization

Case 1: Element-wise Fusion

To reduce overhead of kernel launch, PFVM fuses element-wise operators

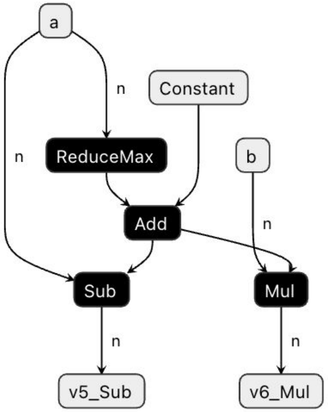


CUDA profile will look like…



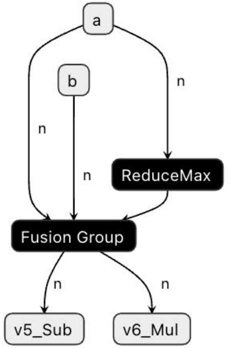
Case 1: Element-wise Fusion

To reduce overhead of kernel launch, PFVM fuses element-wise operators



CUDA profile will look like…

- CUDA kernel launch
- ReduceMax kernel
- CUDA kernel launch
- Add kernel
- CUDA kernel launch
- Sub kernel
- CUDA kernel launch
- Mul kernel

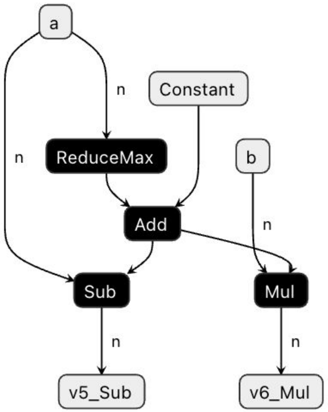


- CUDA kernel launch
- ReduceMax kernel
- CUDA kernel launch
- Fusion kernel

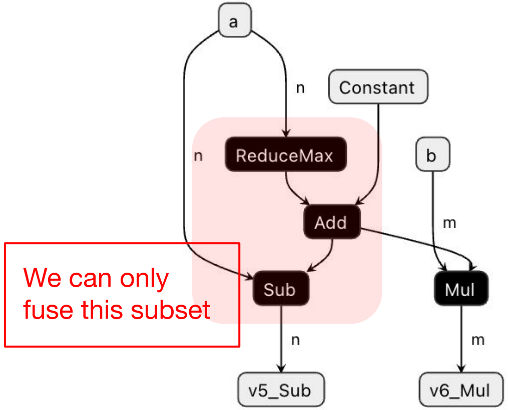
Case 1: Element-wise Fusion

Can we always fuse adjacent element-wise operators?

- No. Must be broadcastable



Can fuse

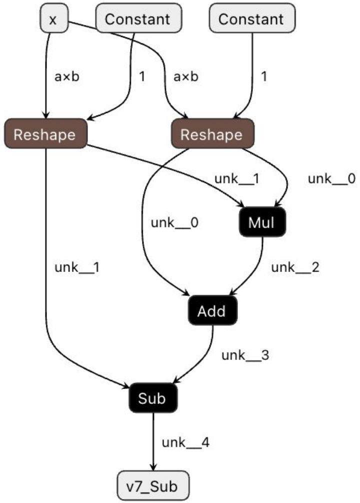


Can't fuse

Case 1: Element-wise Fusion

Can we always fuse adjacent element-wise operators?

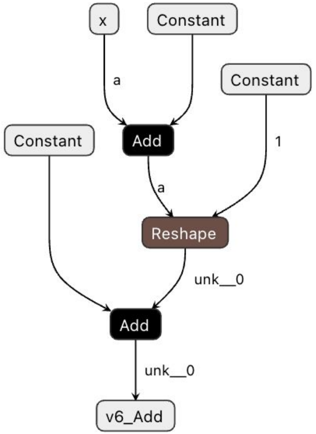
- No. Must be broadcastable



We lose fusion opportunities if we have unknown dims

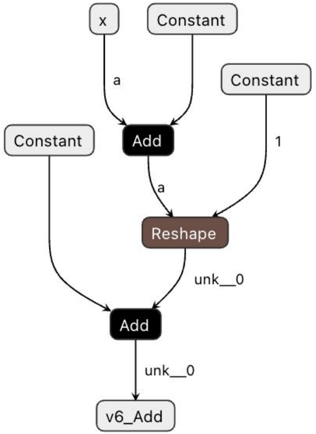
Case 2: Graph simplification

This Reshape can be Identity (and thus can be removed)



Case 2: Graph simplification

This Reshape can be Identity (and thus can be removed)



Then we can do element-wise fusion

Case 2: Graph simplification

Do models contain so many unnecessary operators?

- Not so many if the model is written by human

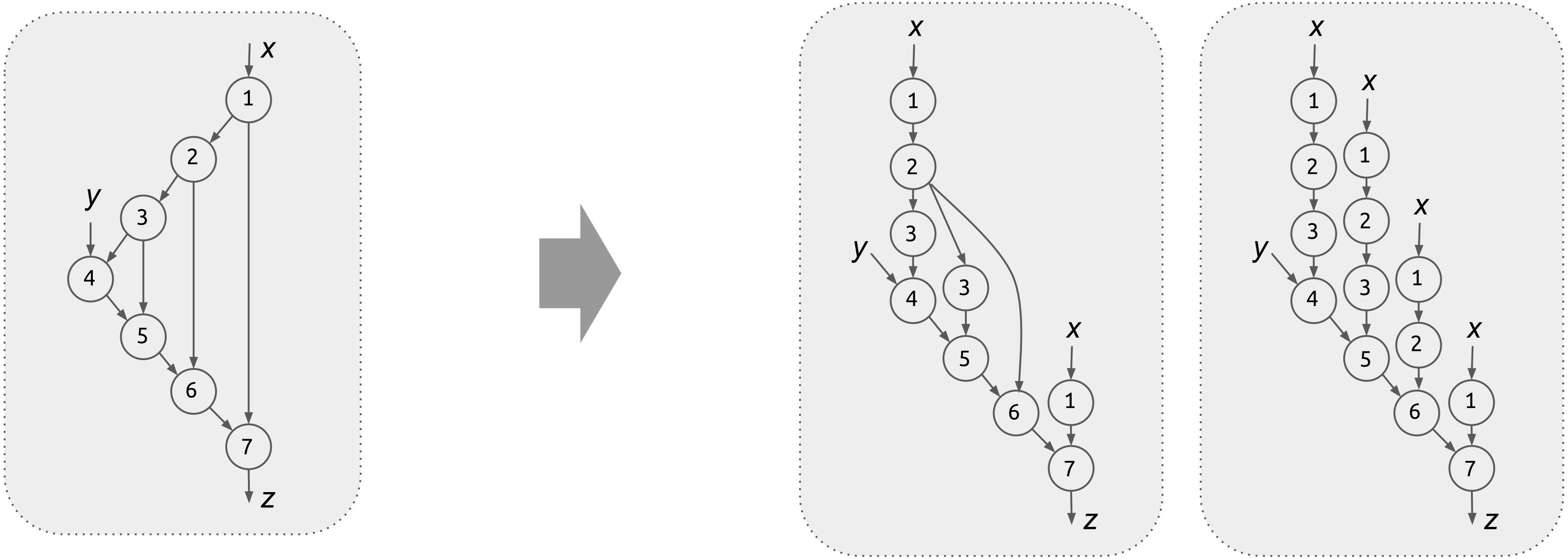
- But may contain **a lot** if it’s generated by programs
  - backward pass generation
  - neural architecture search

Case 2: Graph simplification

Let’s see example at Appendix

Case 3: Automatic checkpointing

Insert recomputation pass to reduce peak memory



Memory: ✕
Latency: ◎

Memory: ○
Latency: ○

Memory: ◎
Latency: ✕

Case 3: Automatic checkpointing

We need tensor size to estimate memory usage

When models contain dynamic axes (dim params)

- OK! Users can estimate them like n=100000

When models contain unknown dims

- Estimation won't work

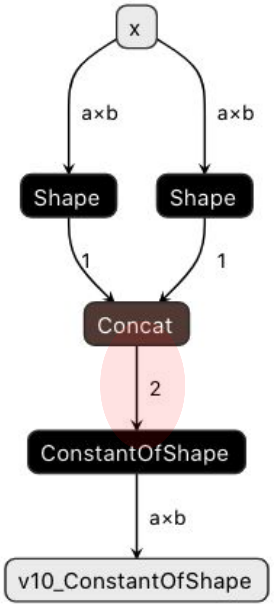
Why shape inference is important

Once we get *unknown* dim, we will have many unknown dims after that!

We lose many opportunities of optimization if shapes are unknown

Current ONNX Shape inference

Symbolic inference from 1.10 is great!!

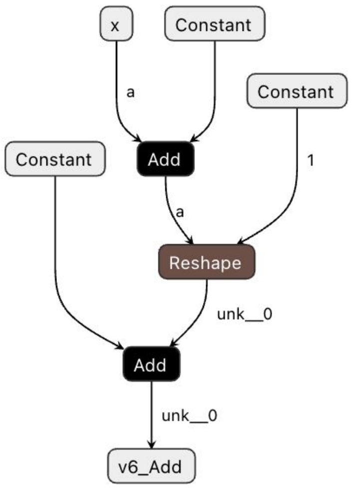


ConstantOfShape output can be inferred from input of shape [2] by data propagation!

Current ONNX Shape inference

For static case, it's already very nice

For dynamic case, we want more supports!



Q: Do we need to support case like Concat([M], [N])?

More optimized model execution

PFN's supercomputer for deep learning

MN-Core limits model dynamicity and gets great optimizations!

- MN-Core is a very large scale fast SIMD machine, and its scheduling is really challenging!
- Since MN-Core model is static, MN-Core programs know exact computation time or memory usage at compile-time!
- PFVM is integrated in MN-Core compiler and do some simplification and shape inference

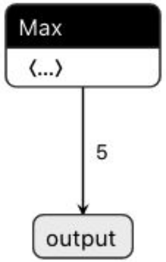
Thank you!

If you have any questions, please reach out to me!

Appendix: Example Optimization

Consider following example

Forward model: $y = \max(a, b)$



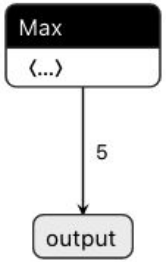
a, b: parameters

max: element-wise max with multidirectional broadcasting

Appendix: Example Optimization

Consider following example

Forward model: $y = \max(a, b)$



We need grad@a and grad@b from grad@y

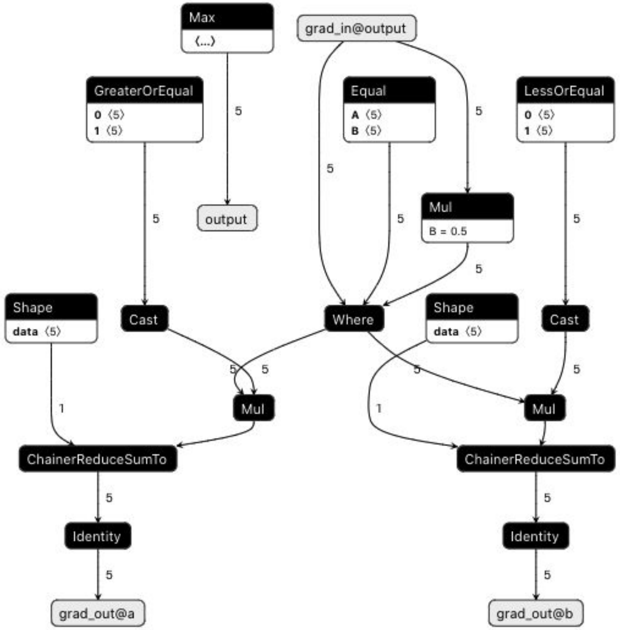
Appendix: Example Optimization

Forward model: y = max(a, b)

gy’ = where(a == b, grad@y / 2, grad@y)

grad@a = mul(gy’, cast(a >= b)).sum_to(a.shape)

grad@b = mul(gy’, cast(a <= b)).sum_to(b.shape)



Appendix: Example Optimization

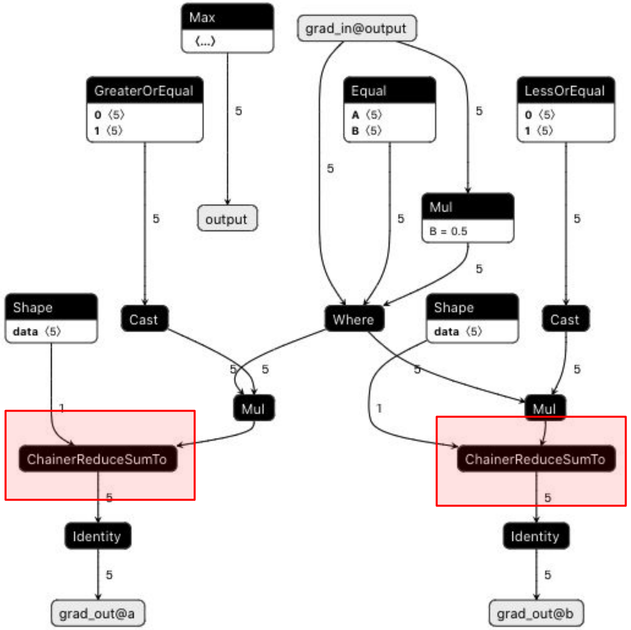
Forward model: y = max(a, b)

gy’ = where(a == b, grad@y / 2, grad@y)

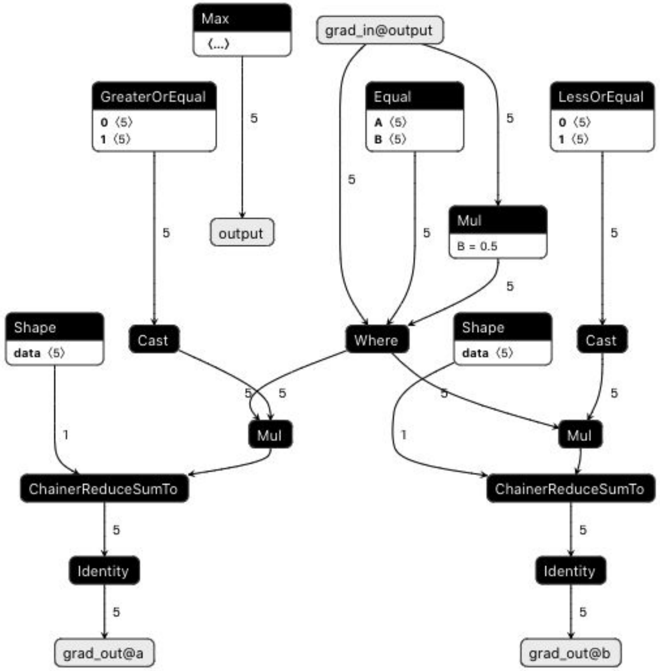
grad@a = mul(gy’, cast(a >= b)).sum_to(a.shape)

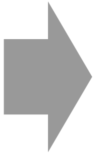
grad@b = mul(gy’, cast(a <= b)).sum_to(b.shape)

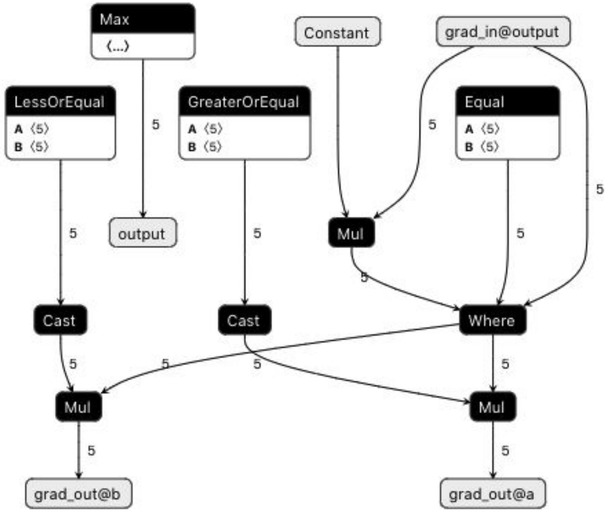
These reduction operators are heavy but can be removed when there is no broadcast



Appendix: Example Optimization

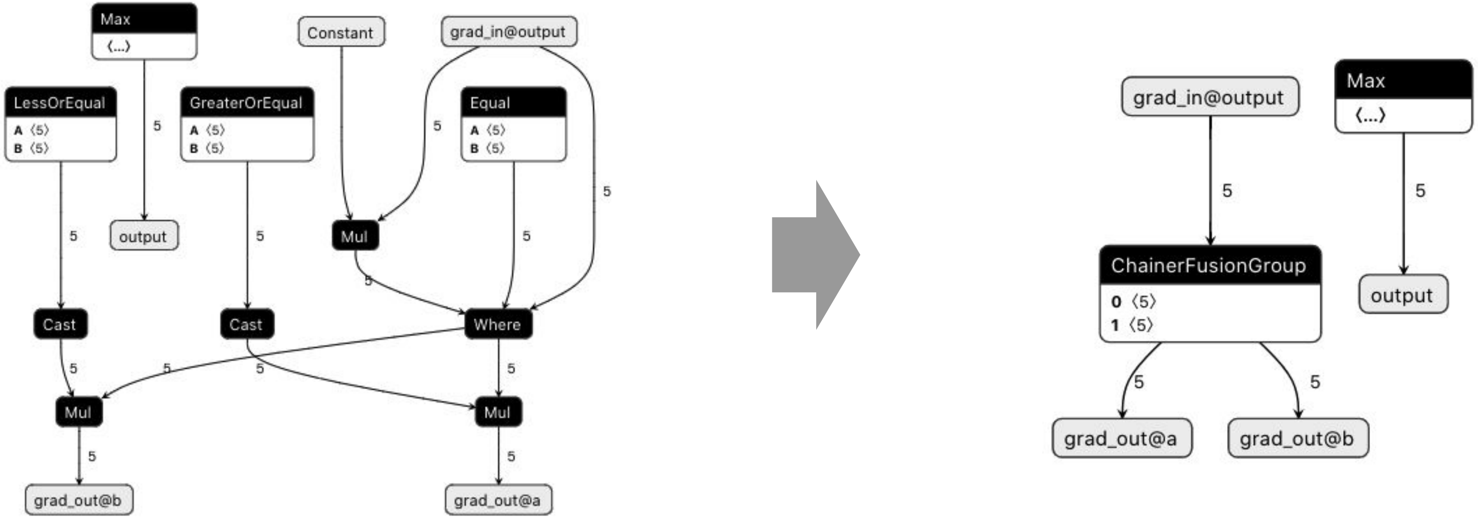






Simplified graph

Appendix: Example Optimization



Elementwise fusion