Automated GPU Kernel Fusion with XLA

EuroLLVM'19, April 8 2019

Thomas Joerg, Google
Presenting work done by the XLA team

Outline

- TensorFlow
- Kernel fusion
- XLA compiler
- Automated kernel fusion

TensorBoard

EVENTS IMAGES GRAPH HISTOGRAMS

Fit to screen

Run cifar-train

Upload Choose File

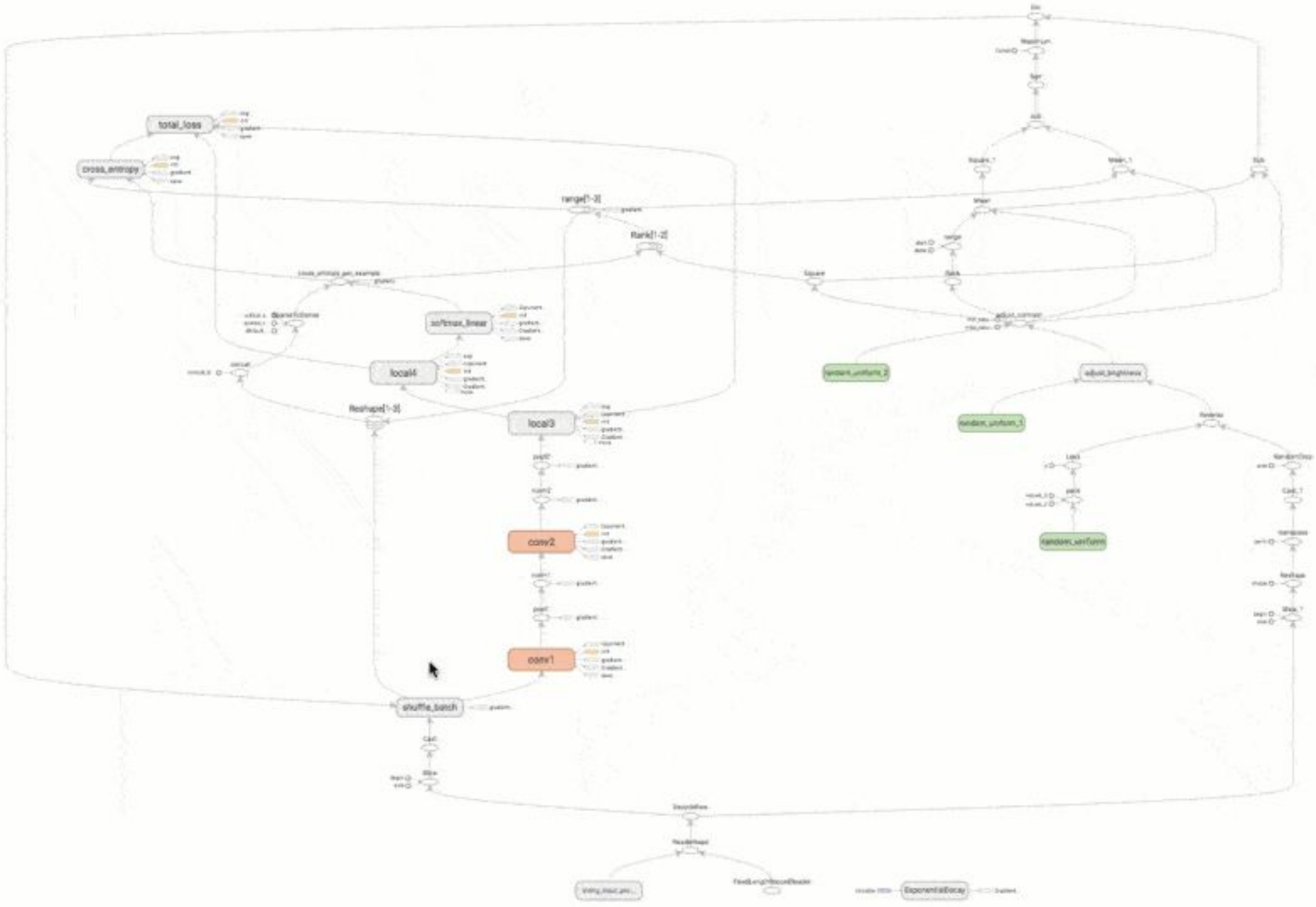
Color Structure

color: same substructure
gray: unique substructure

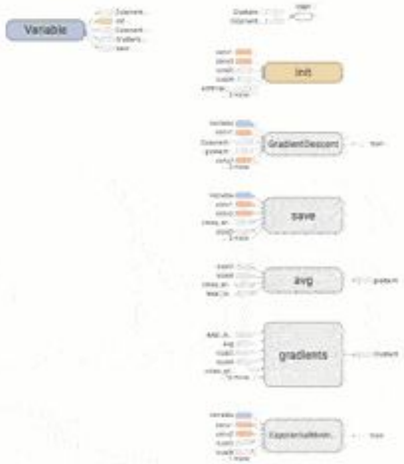
Graph (* = expandable)

- Namespace*
- OpNode
- Unconnected series*
- Connected series*
- Constant
- Summary
- Dataflow edge
- Control dependency edge
- Reference edge

Main Graph

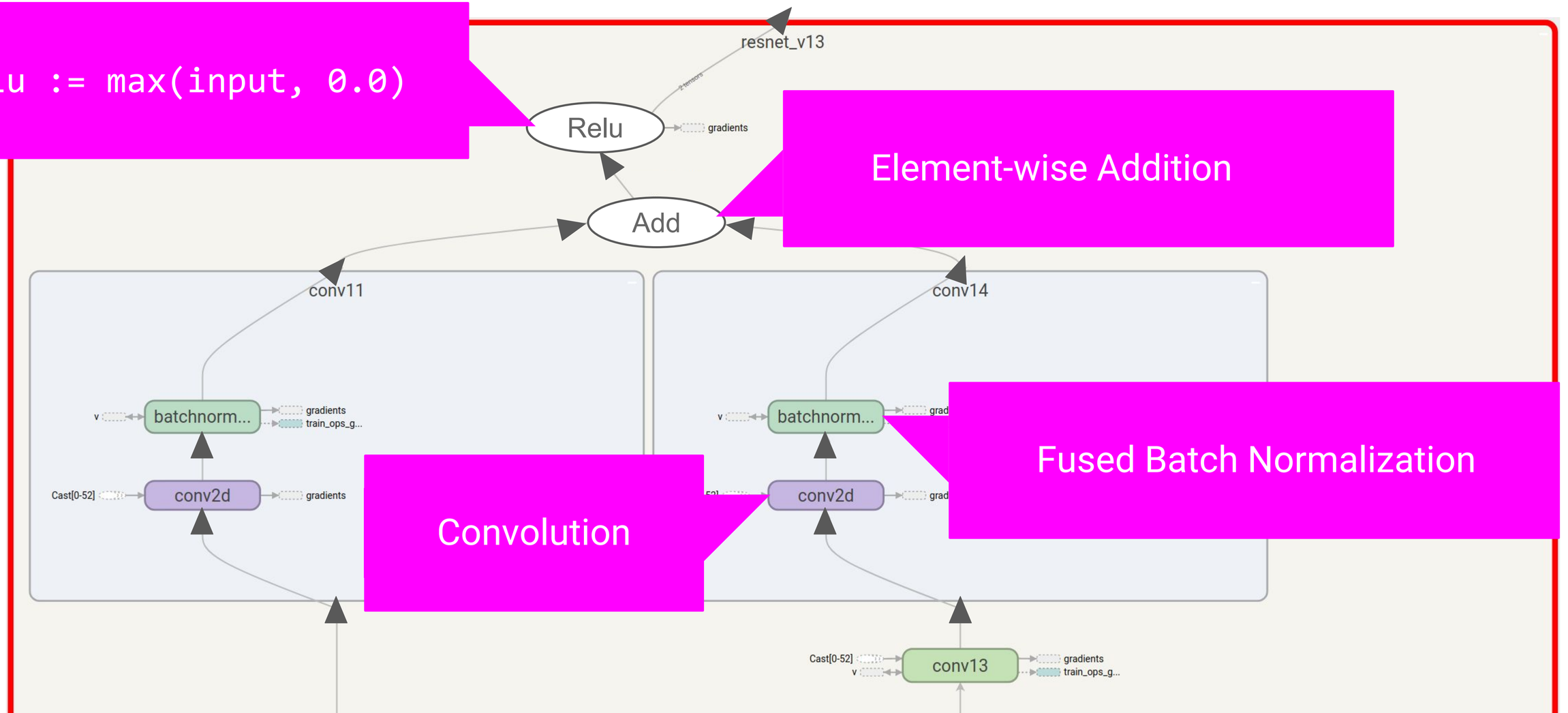


Auxiliary nodes



Example: ResNet block

```
ReLu := max(input, 0.0)
```



Fused Kernels

- Convenient
- Performant

```
// Compute a * x + y.
// a is a scalar, x and y are tensors.
tmp = tf.multiply(a, x)
out = tf.add(tmp, y)
```

```
// Compute a * x + y.
// a is a scalar, x and y are tensors.
tmp = tf.multiply(a, x)
out = tf.add(tmp, y)

__global__ void Multiply(int n, float a, float* x) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) x[i] = a * x[i];
}
```

```
// Compute a * x + y.
// a is a scalar, x and y are tensors.
tmp = tf.multiply(a, x)
out = tf.add(tmp, y)
```

Tensors read + written: 4

```
__global__ void Multiply(int n, float a, float* x) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) x[i] = a * x[i];
}
__global__ void Add(int n, float* x, float* y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) x[i] = x[i] + y[i];
}
```

```
// Compute a * x + y.
// a is a scalar, x and y are tensors.
tmp = tf.multiply(a, x)
out = tf.add(tmp, y)

__global__ void FusedMulAdd(int n, float a, float* x, float* y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) x[i] = a * x[i] + y[i];
}
```

```
// Compute a * x + y.
// a is a scalar, x and y are tensors.
out = tf.fused_multiply_add(a, x, y)
```

Tensors read + written: 3

25% reduction!

```
__global__ void FusedMulAdd(int n, float a, float* x, float* y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i < n) x[i] = a * x[i] + y[i];
}
```

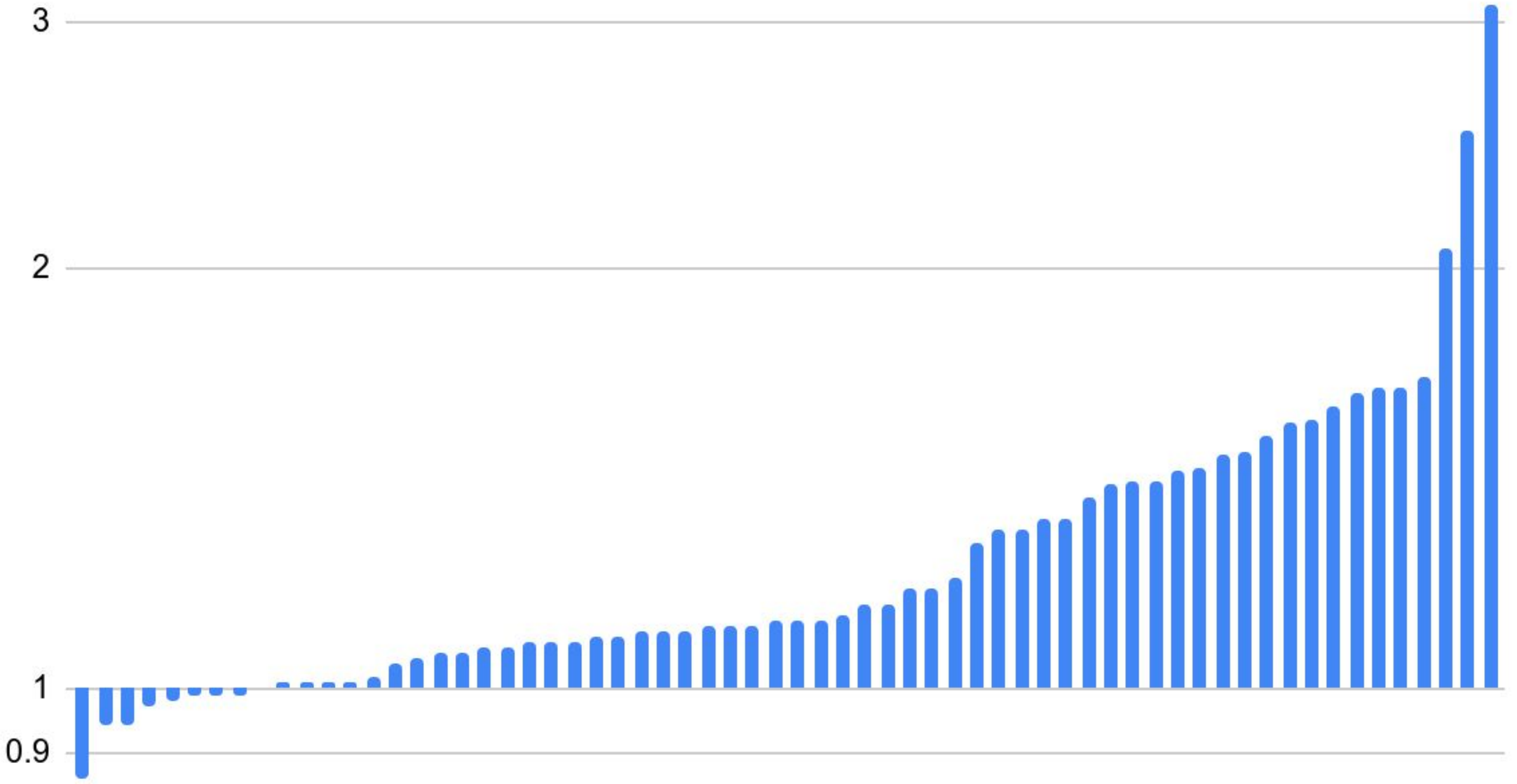
Fused Kernels

- Convenient
- Performant

But

- Development cost
- Inflexibel
- Hard to optimize

Speedup of TF with XLA vs TF without XLA

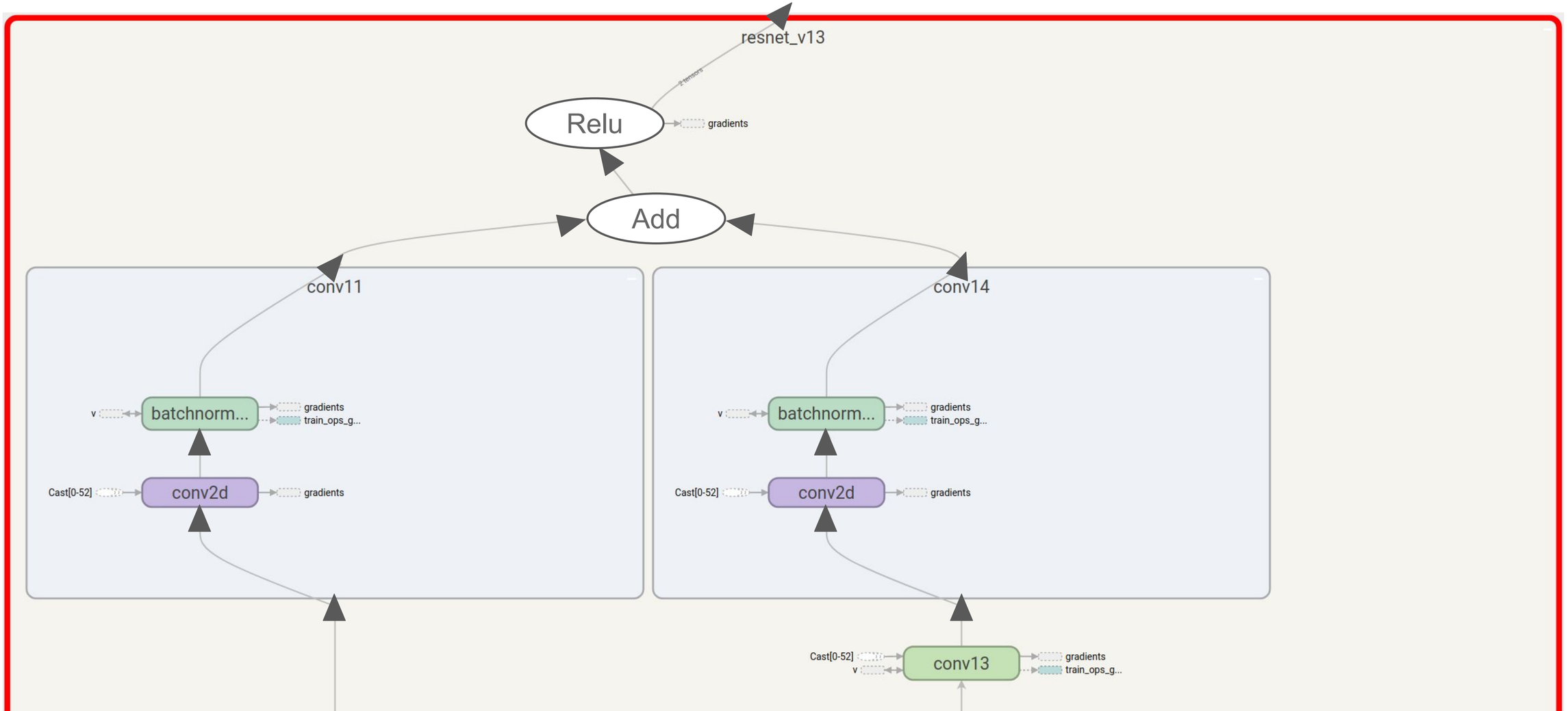


Submitter	Hardware	Chip count	Software	ResNet-50 v1.5 *
NVIDIA	DGX-1 (on premise)	8	ngc18.11_MXNet, cuDNN 7.4	65.6
Google	8x Volta V100 (Cloud)	8	TF 1.12, cuDNN 7.4	64.1

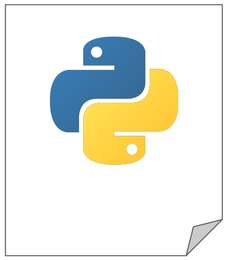
Full results: https://mlperf.org/results/

* speedup relative to reference implementation

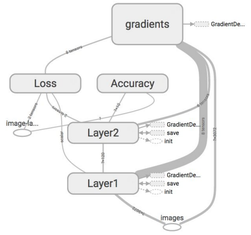
Example: ResNet block



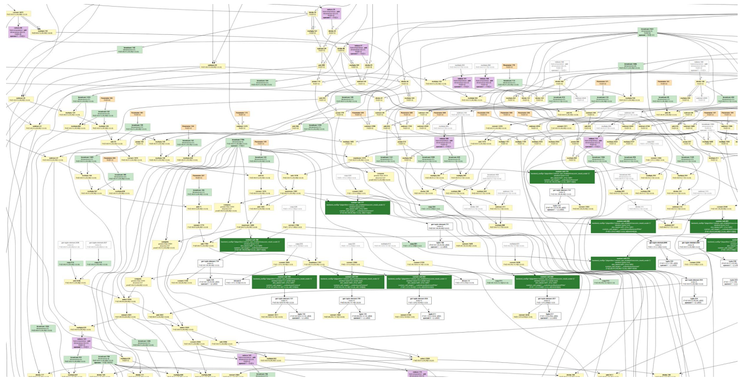
TensorFlow with XLA



TensorFlow Model



TensorFlow Graph



XLA Intermediate Representation: HLO

HLO Fusion happens here!

XLA target-independent & target-specific optimizations

TPU

GPU

CPU

Target-specific code generation

HLO IR

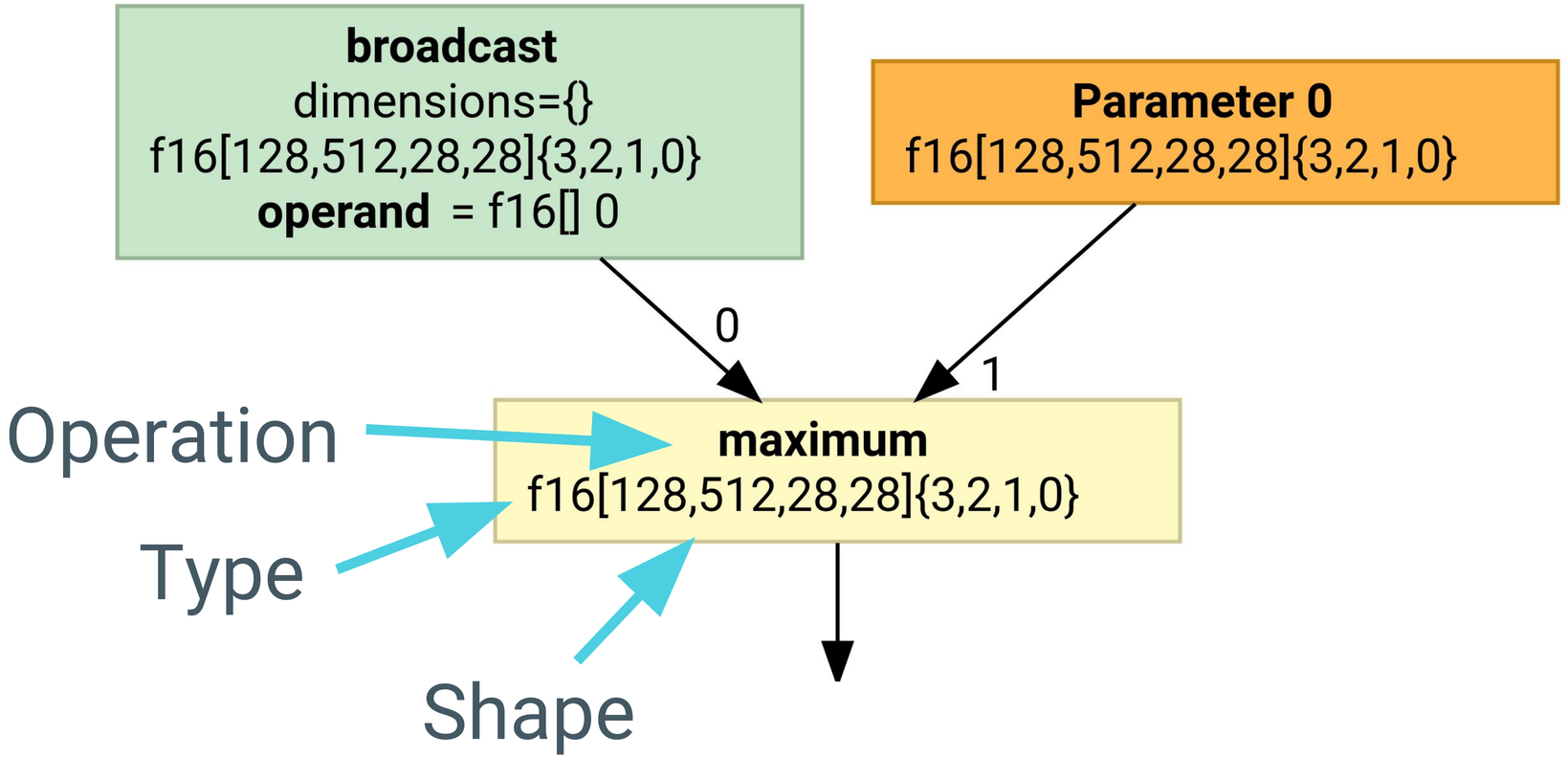
Sample HLO ops

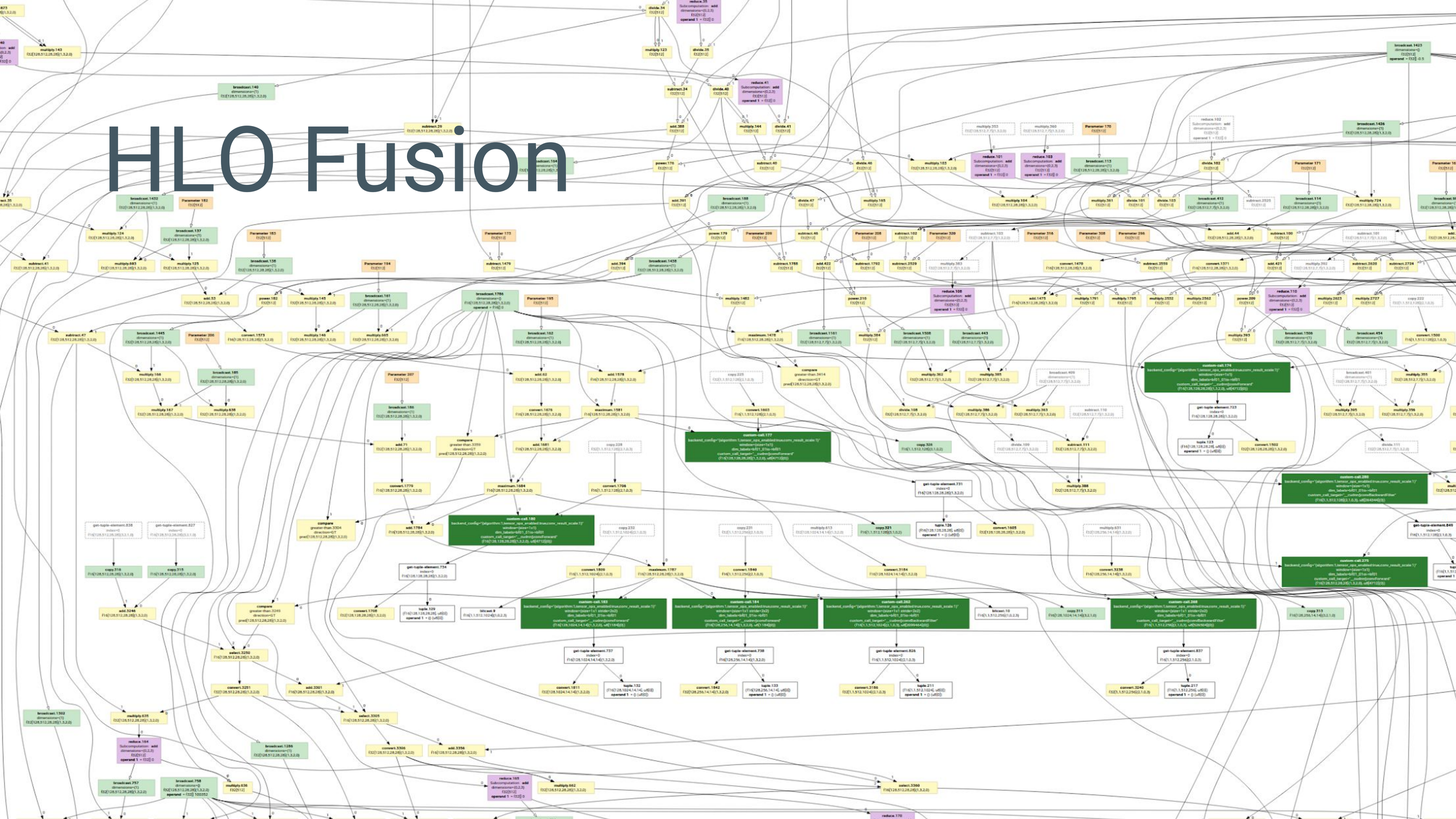
- Elementwise math
  - Add, Tanh, Map
- Spezialized math for neural nets
  - Dot, Convolution, Reduce
- Re-organize data
  - Reshape, Broadcast, Concat, Tuple
- Control flow
  - While, Call, CustomCall
- Data transfer
  - Parameter, Constant

Sample data types

- Primitive types
  - PRED
  - F16
  - F32
- Composite types
  - array: F32[2,3], F16[]
  - tuple: TUPLE(F32[16], F16)

ReLu in HLO


broadcast
dimensions={}
f16[128,512,28,28]{3,2,1,0}
operand = f16[] 0
Parameter 0
f16[128,512,28,28]{3,2,1,0}
0
1
maximum
f16[128,512,28,28]{3,2,1,0}
Operation
Type
Shape

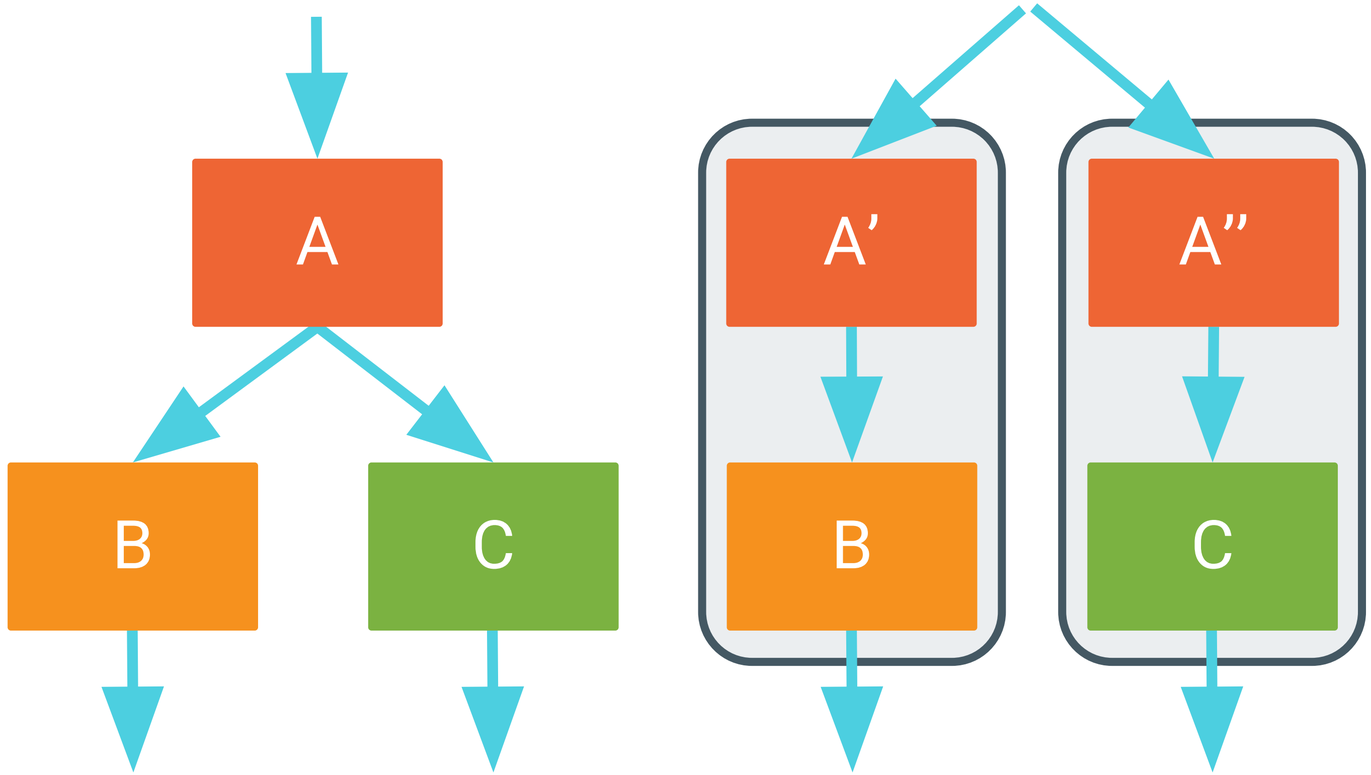

HLO Fusion

HLO Fusion

- Reduce memory bandwidth
- Compatible loop pattern
- Coalesced memory access

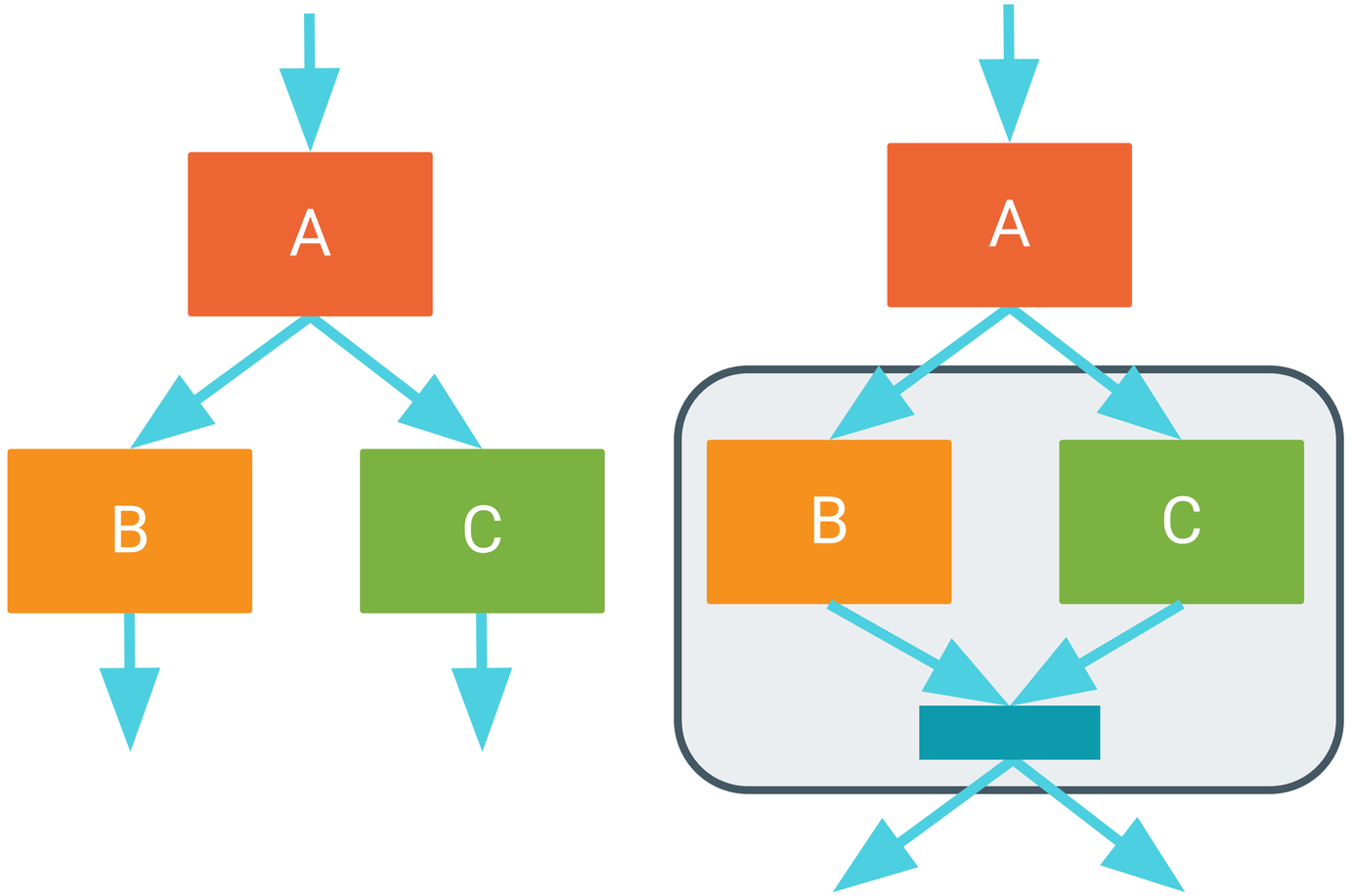
HLO Fusion

1) Fusion (with duplication)
2) Sibling fusion
3) Fusion with multiple outputs



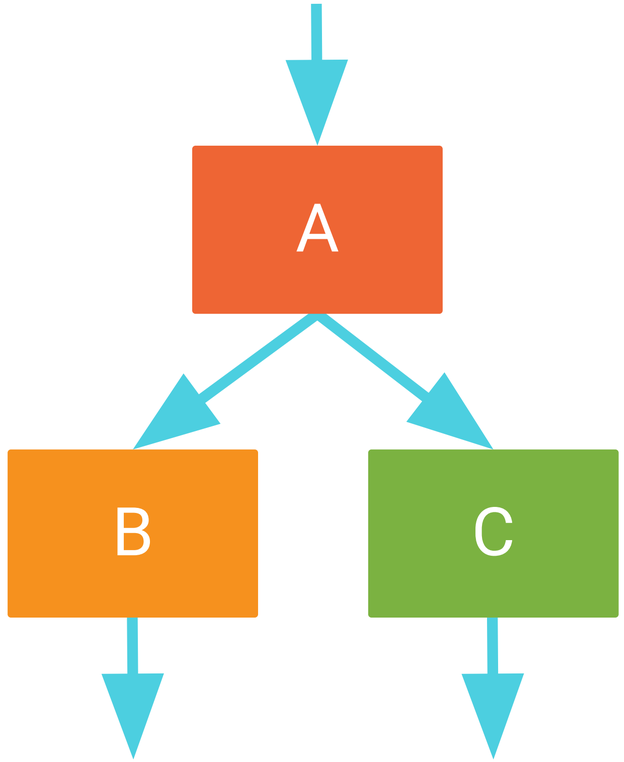
HLO Fusion

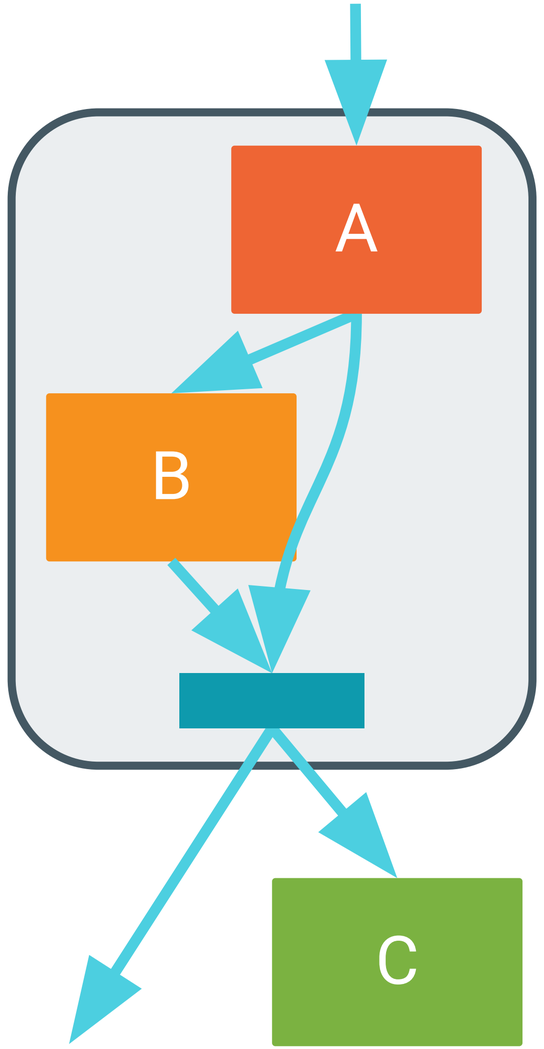
1) Fusion (with duplication)
2) Sibling fusion
3) Fusion with multiple outputs



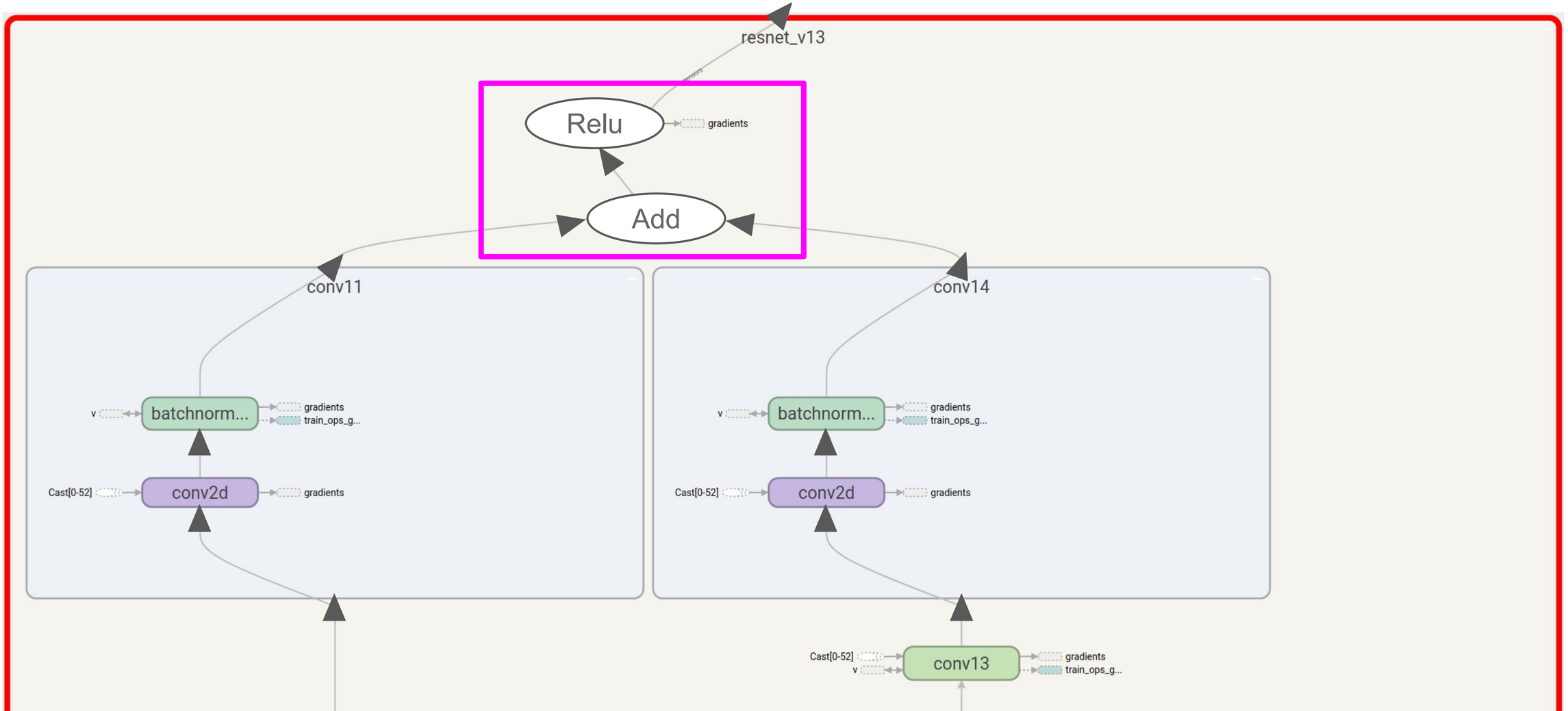
HLO Fusion

1) Fusion (with duplication)
2) Sibling fusion
3) Fusion with multiple outputs

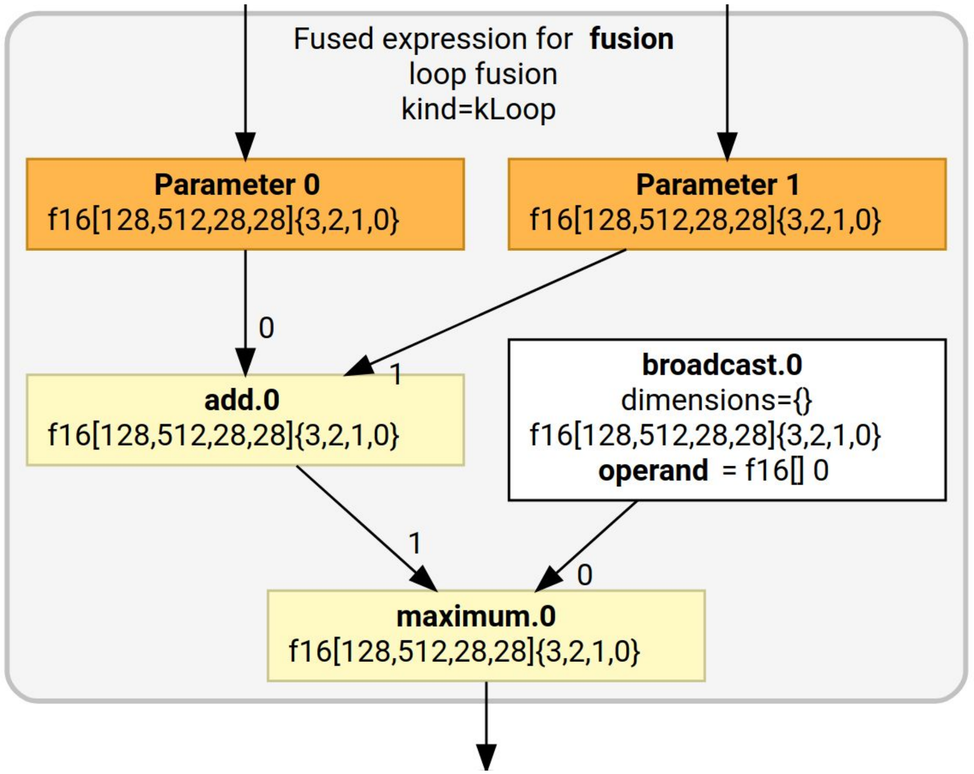




Example: ResNet block



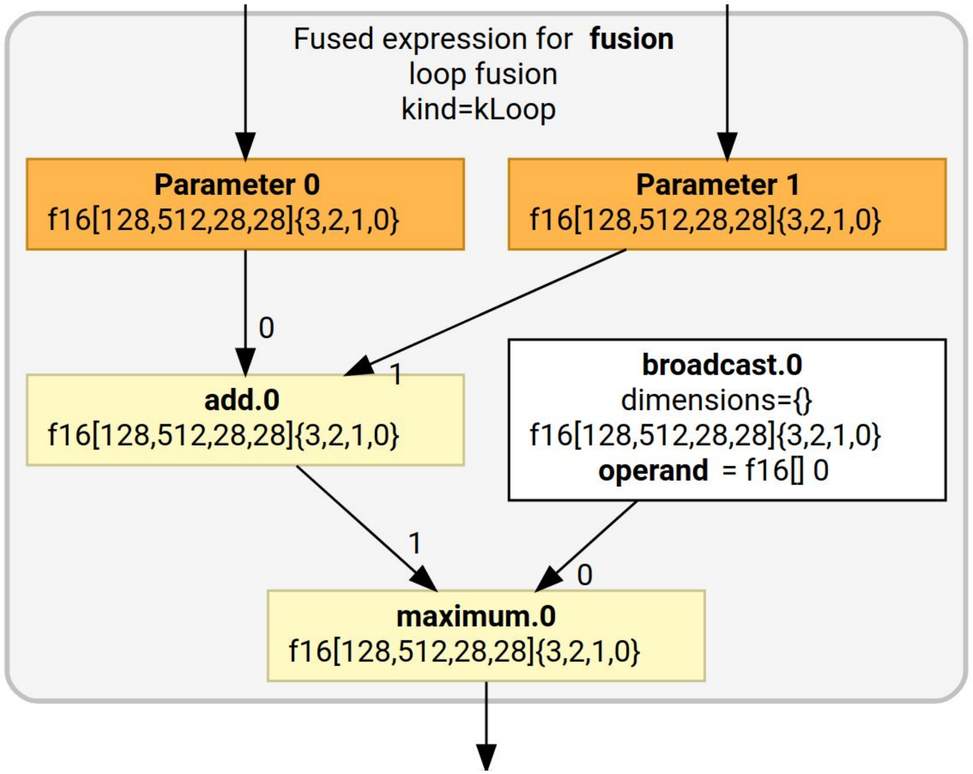
Fused Add + ReLu



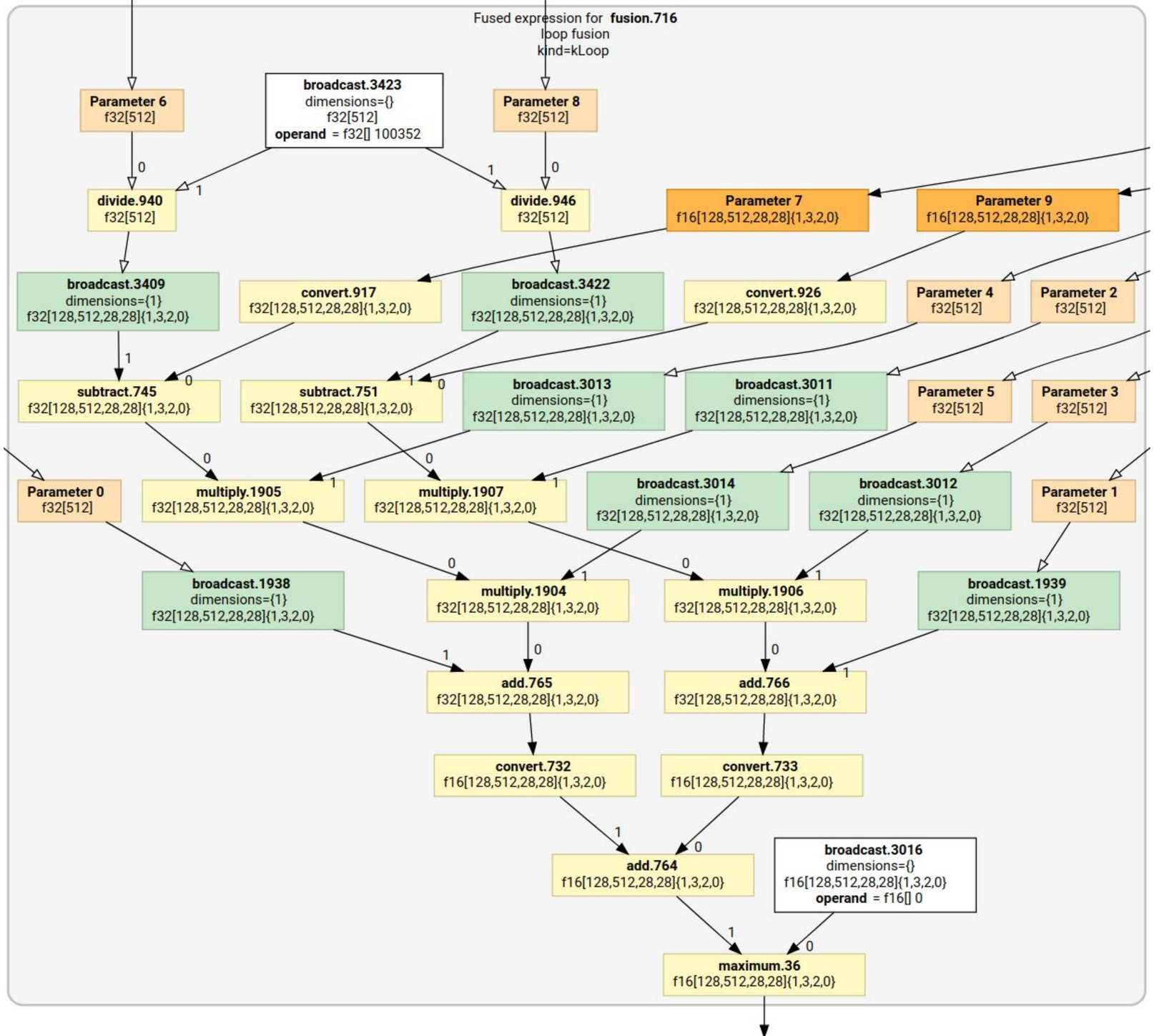

```
__global__ void fusion(float *lhs,
    float *rhs, float* output) {

  int i = blockIdx.x * blockDim.x +
          threadIdx.x;

  if (i < 128*512*28*28) {
    output[i] =
  }

}
```

```
std::function<llvm::Value*>(const IrArray::Index& index)
    MakeElementGenerator(const HloInstruction* hlo,
                         HloToElementGeneratorMap& operand_to_generator) {
  switch (hlo->opcode()) {
    case HloOpcode::kMaximum:
      return [...](const IrArray::Index& index) {
        llvm::Value* lhs =
            operand_to_generator.at(hlo->operand(0))(index);
        llvm::Value* rhs =
            operand_to_generator.at(hlo->operand(1))(index);
        auto cmp = b->CreateFCmpUGE(lhs, rhs);
        return ir_builder_->CreateSelect(cmp, lhs, rhs);
      };
    ...
  }
```

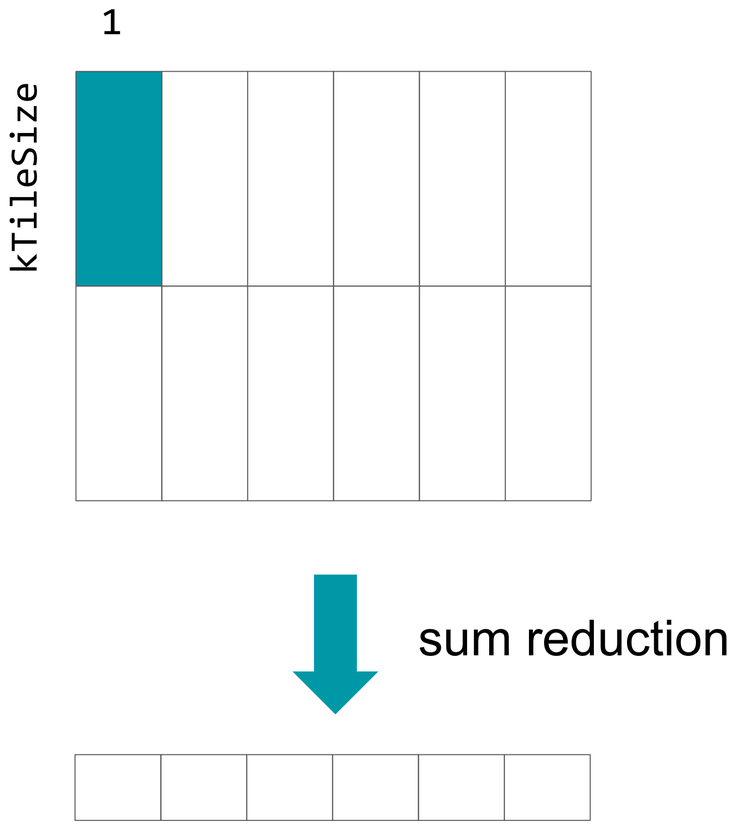
Fused Add + ReLu




```
__global__ void fusion(float *lhs,
    float *rhs, float* output) {

  int i = blockIdx.x * blockDim.x +
          threadIdx.x;

  if (i < 128*512*28*28) {
    output[i] =  max(0.0, lhs[i] + rhs[i]);
  }

}
```

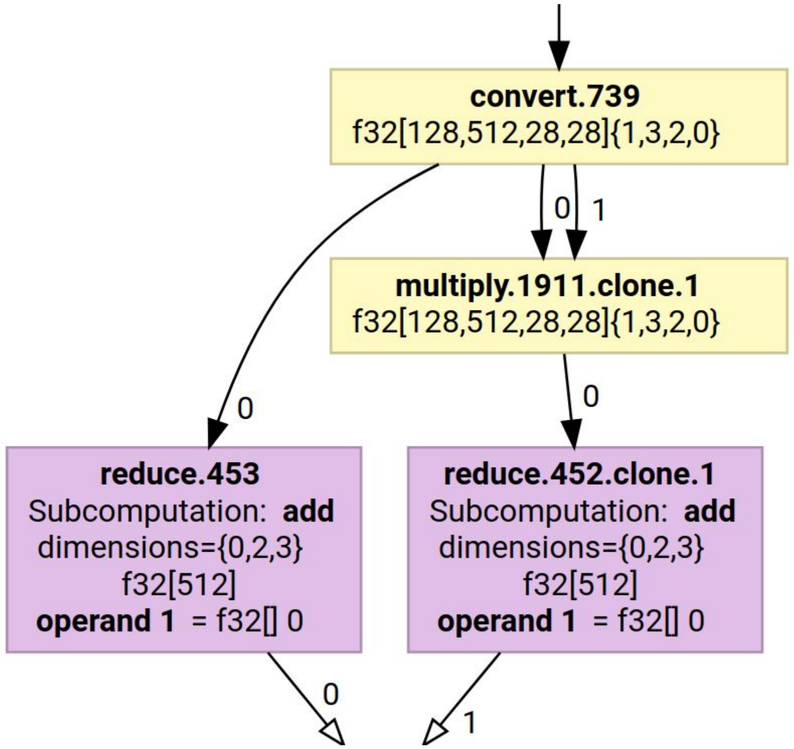

Fused expression for fusion.716
loop fusion
kind=kLoop
Parameter 6
f32[512]
broadcast.3423
dimensions={}
f32[512]
operand = f32[] 100352
Parameter 8
f32[512]
divide.940
f32[512]
divide.946
f32[512]
Parameter 7
f16[128,512,28,28]{1,3,2,0}
Parameter 9
f16[128,512,28,28]{1,3,2,0}
broadcast.3409
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
convert.917
f32[128,512,28,28]{1,3,2,0}
broadcast.3422
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
convert.926
f32[128,512,28,28]{1,3,2,0}
Parameter 4
f32[512]
Parameter 2
f32[512]
subtract.745
f32[128,512,28,28]{1,3,2,0}
subtract.751
f32[128,512,28,28]{1,3,2,0}
broadcast.3013
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
broadcast.3011
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
Parameter 5
f32[512]
Parameter 3
f32[512]
Parameter 0
f32[512]
multiply.1905
f32[128,512,28,28]{1,3,2,0}
multiply.1907
f32[128,512,28,28]{1,3,2,0}
broadcast.3014
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
broadcast.3012
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
Parameter 1
f32[512]
broadcast.1938
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
multiply.1904
f32[128,512,28,28]{1,3,2,0}
multiply.1906
f32[128,512,28,28]{1,3,2,0}
broadcast.1939
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
add.765
f32[128,512,28,28]{1,3,2,0}
add.766
f32[128,512,28,28]{1,3,2,0}
convert.732
f16[128,512,28,28]{1,3,2,0}
convert.733
f16[128,512,28,28]{1,3,2,0}
add.764
f16[128,512,28,28]{1,3,2,0}
broadcast.3016
dimensions={}
f16[128,512,28,28]{1,3,2,0}
operand = f16[] 0
maximum.36
f16[128,512,28,28]{1,3,2,0}

Reduction

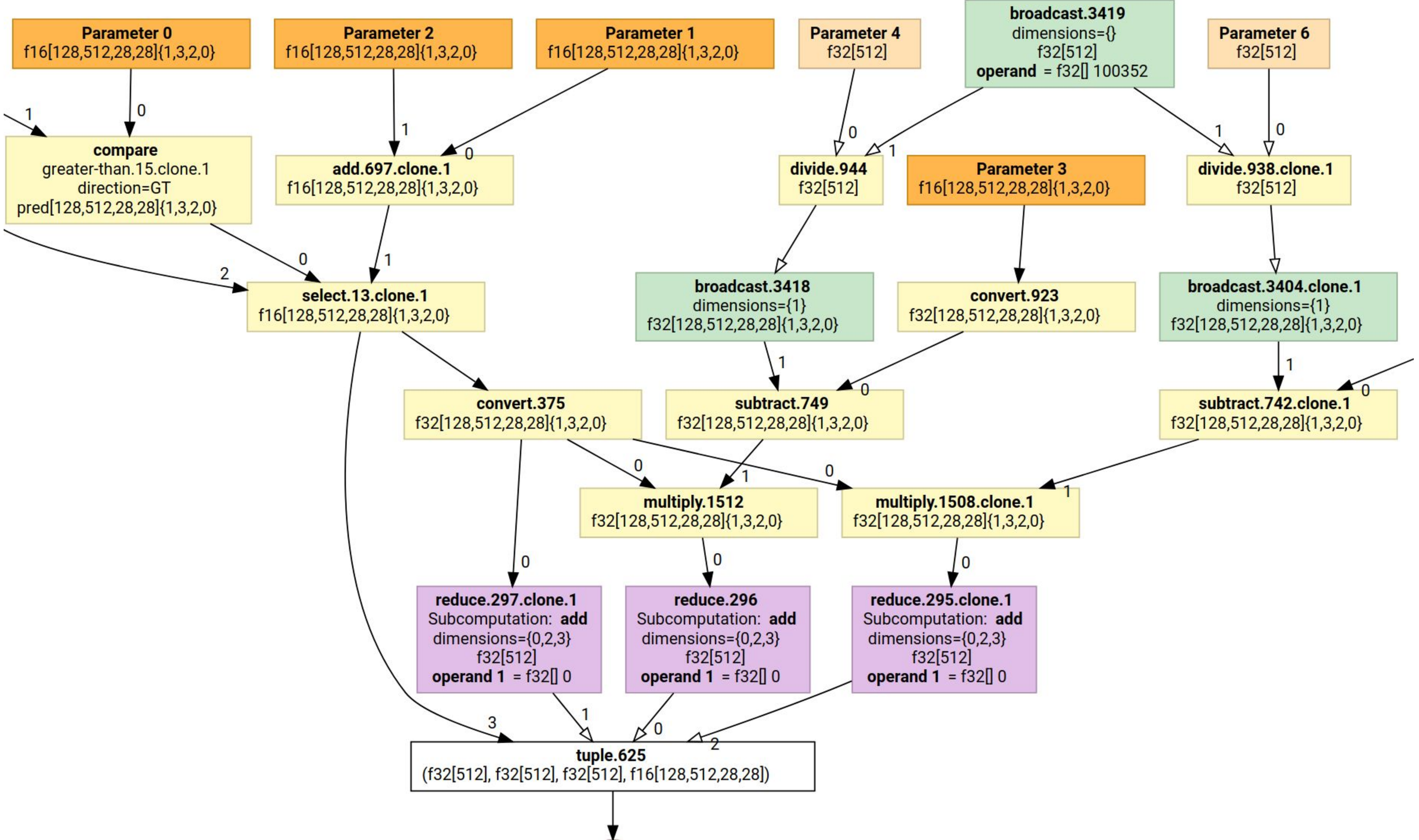



```
i = blockIdx.x * blockDim.x + threadIdx.x;
y_in_tiles = i / width;
x = i % width;

for (int j = 0; j < kTileSize: ++j) {
  y = y_in_tiles * kTileSize + j;
  if (y < height) {
    partial_sum += generator(y, x);
  }
}
atomicAdd(&output[x], partial_sum);
```

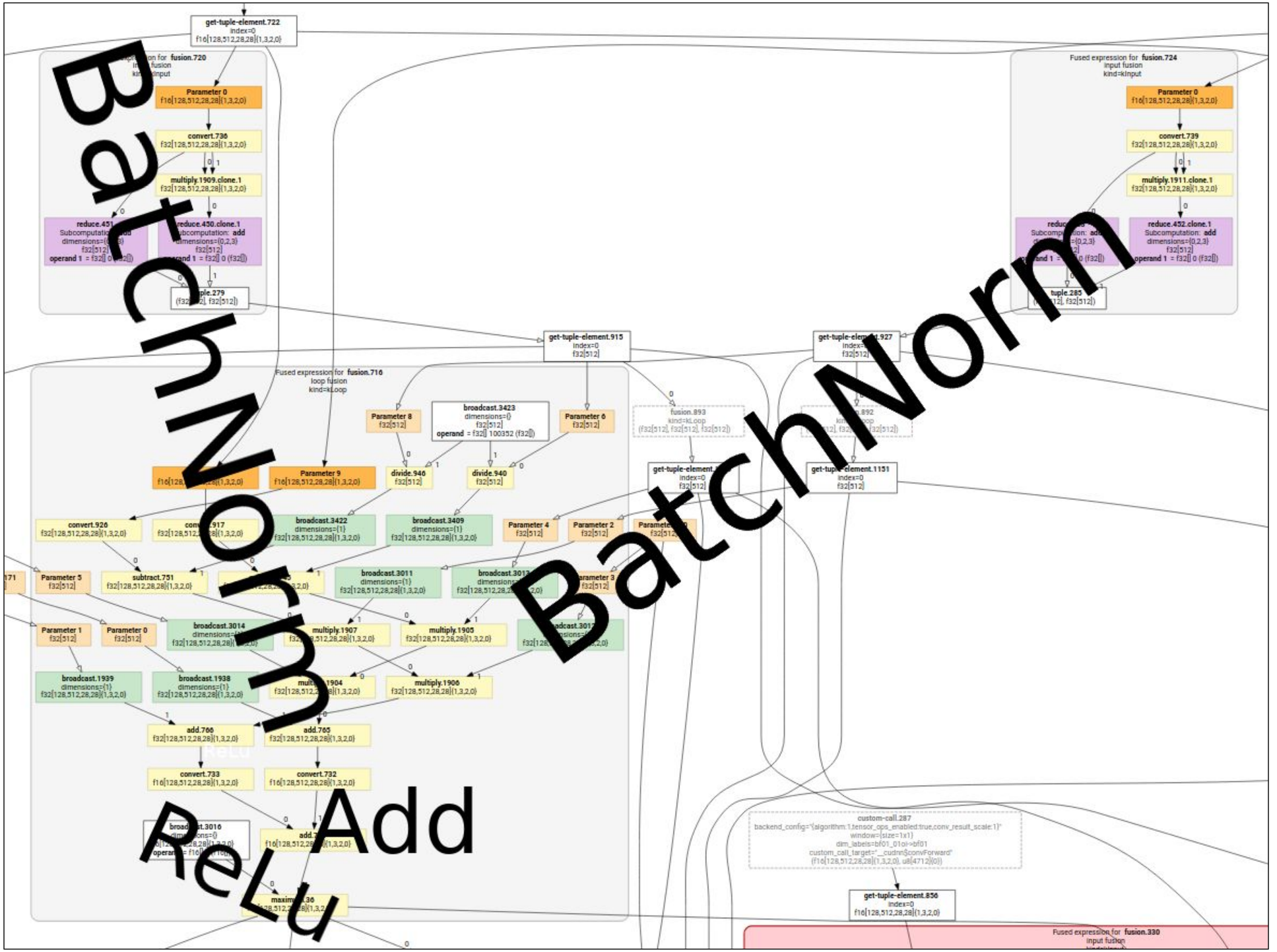
Multi-output fusion




```
i = blockIdx.x * blockDim.x + threadIdx.x;
y_in_tiles = i / width;
x = i % width;

for (int j = 0; j < kTileSize: ++j) {
  y = y_in_tiles * kTileSize + j;
  if (y < height) {
    partial_sum[0] += generator[0](y, x);
    partial_sum[1] += generator[1](y, x);
  }
}
atomicAdd(&output[0][x], partial_sum[0]);
atomicAdd(&output[1][x], partial_sum[1]);
```


Parameter 0
f16[128,512,28,28]{1,3,2,0}
Parameter 2
f16[128,512,28,28]{1,3,2,0}
Parameter 1
f16[128,512,28,28]{1,3,2,0}
Parameter 4
f32[512]
broadcast.3419
dimensions={}
f32[512]
operand = f32[] 100352
Parameter 6
f32[512]
compare
greater-than.15.clone.1
direction=GT
pred[128,512,28,28]{1,3,2,0}
add.697.clone.1
f16[128,512,28,28]{1,3,2,0}
divide.944
f32[512]
Parameter 3
f16[128,512,28,28]{1,3,2,0}
divide.938.clone.1
f32[512]
select.13.clone.1
f16[128,512,28,28]{1,3,2,0}
broadcast.3418
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
convert.923
f32[128,512,28,28]{1,3,2,0}
broadcast.3404.clone.1
dimensions={1}
f32[128,512,28,28]{1,3,2,0}
convert.375
f32[128,512,28,28]{1,3,2,0}
subtract.749
f32[128,512,28,28]{1,3,2,0}
subtract.742.clone.1
f32[128,512,28,28]{1,3,2,0}
multiply.1512
f32[128,512,28,28]{1,3,2,0}
multiply.1508.clone.1
f32[128,512,28,28]{1,3,2,0}
reduce.297.clone.1
Subcomputation: add
dimensions={0,2,3}
f32[512]
operand 1 = f32[] 0
reduce.296
Subcomputation: add
dimensions={0,2,3}
f32[512]
operand 1 = f32[] 0
reduce.295.clone.1
Subcomputation: add
dimensions={0,2,3}
f32[512]
operand 1 = f32[] 0
tuple.625
(f32[512], f32[512], f32[512], f16[128,512,28,28])

```
i = blockIdx.x * blockDim.x + threadIdx.x;
y_in_tiles = i / width;
x = i % width;

for (int j = 0; j < kTileSize: ++j) {
  y = y_in_tiles * kTileSize + j;
  if (y < height) {
    partial_sum[0] += generator[0](y, x);
    partial_sum[1] += generator[1](y, x);
    partial_sum[2] += generator[2](y, x);
    output[3][y, x] = generator[3](y, x);
  }
}
atomicAdd(&output[0][x], partial_sum[0]);
atomicAdd(&output[1][x], partial_sum[1]);
atomicAdd(&output[2][x], partial_sum[2]);
```


BatchNorm
BatchNorm
BatchNorm
Add
ReLu

Thank you! Questions?

XLA documentation
https://www.tensorflow.org/xla/overview

Public XLA mailing list
xla-dev@googlegroups.com

XLA on Github
https://github.com/tensorflow/tensorflow/tree/master/tensorflow/compiler