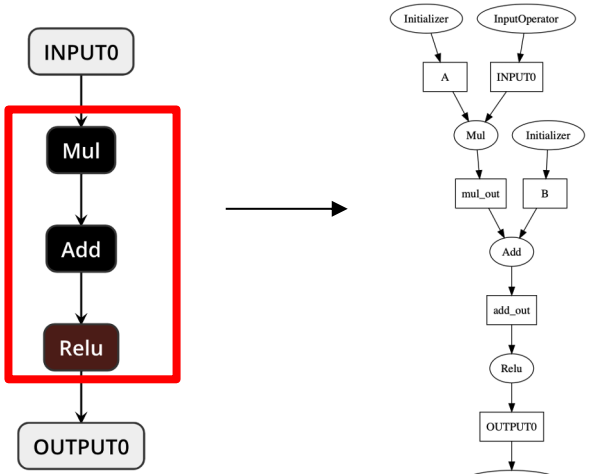
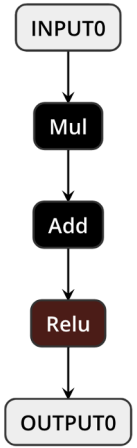
Hardware-Dependent Optimizations

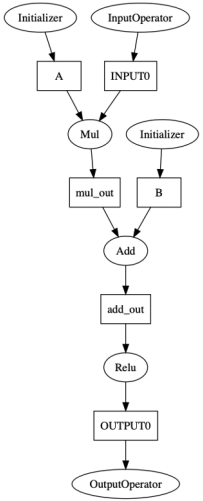
Cheng-Yi Chou, May 2020
Skymizer Taiwan Inc.

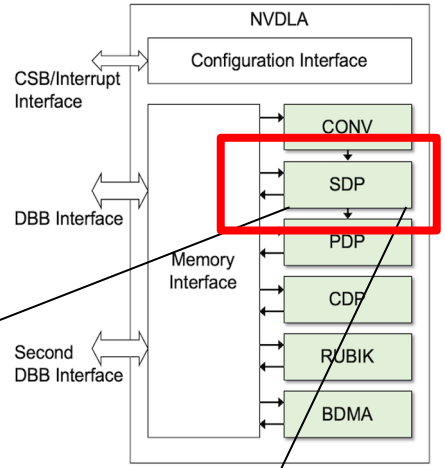
Mapping IR Graph To NVDLA Hardware

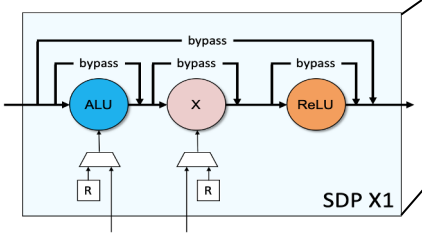


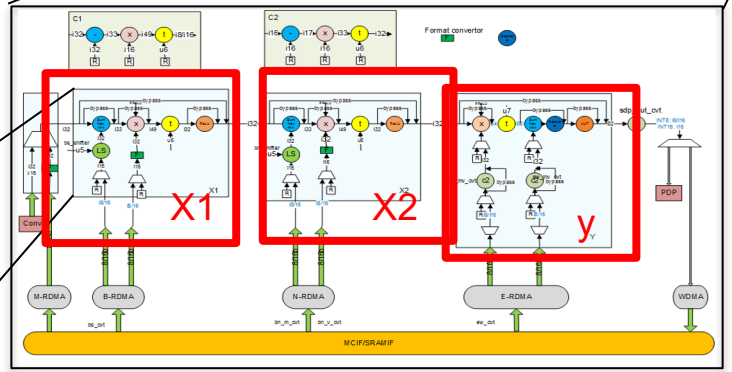
Mapping IR Graph To NVDLA Hardware









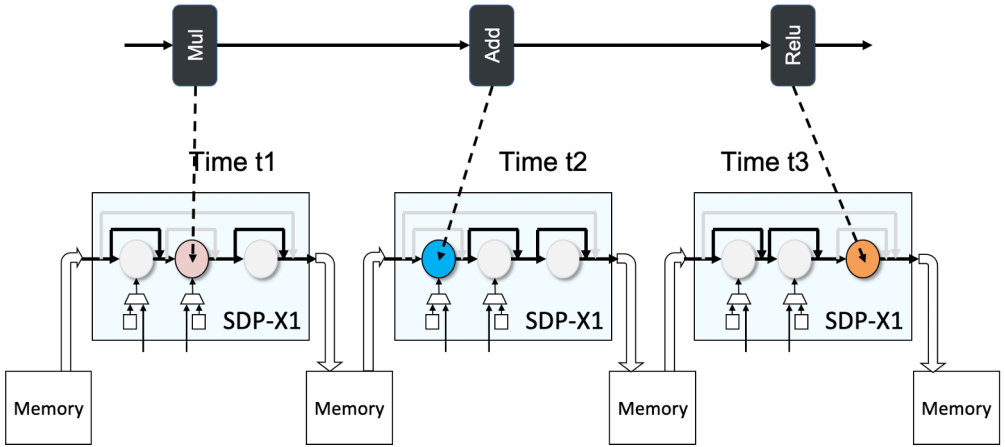


skymizer

Mapping Options

If a model contains a series of ***Mul-Add-Relu***, how to map operators into NVDLA hardware ?

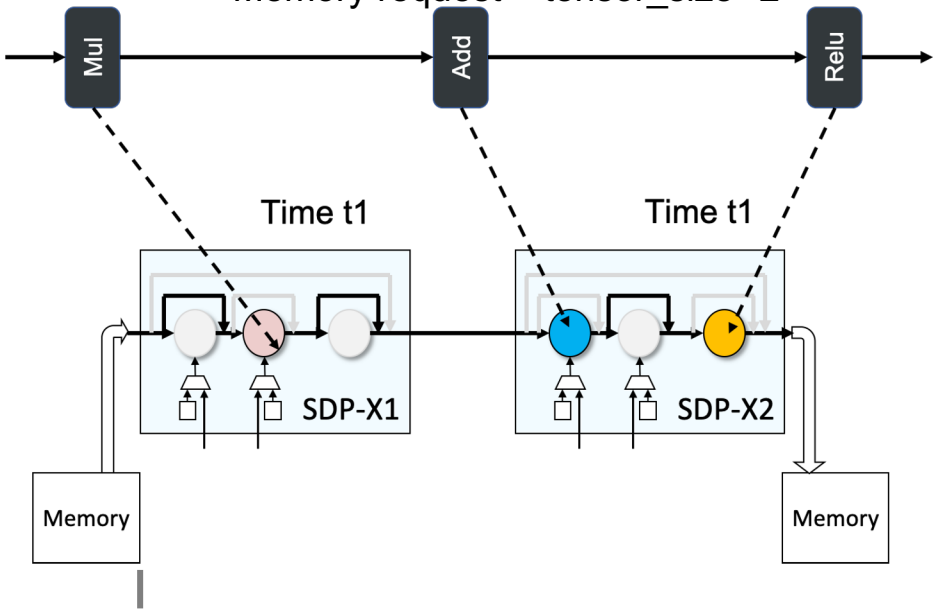
Option 1: Execution time = T
Memory request = tensor_size *6



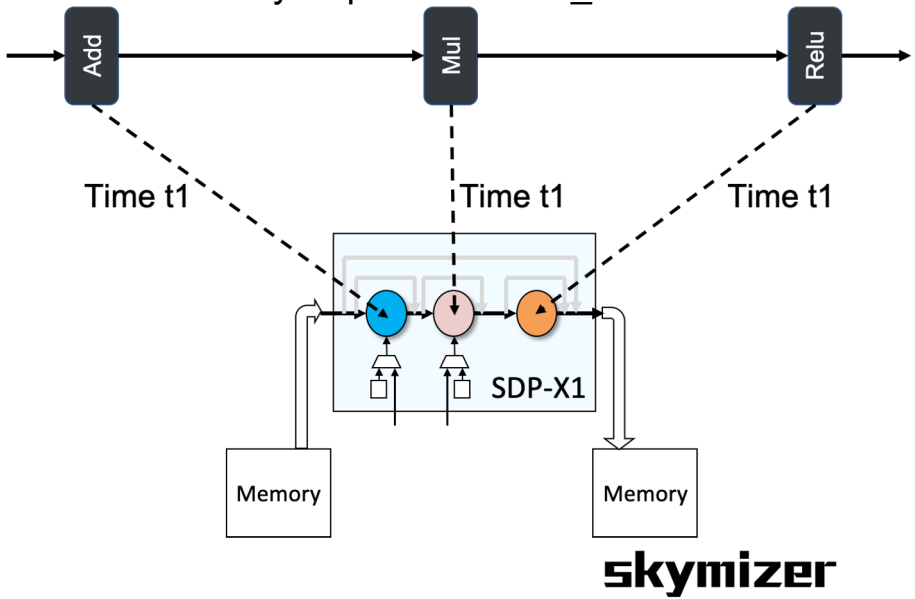
Mapping Options

If a model contains a series of ***Mul-Add-Relu***, how to map operators into NVDLA hardware ?

Option 2: Execution time ~= T/3
Memory request = tensor_size *2

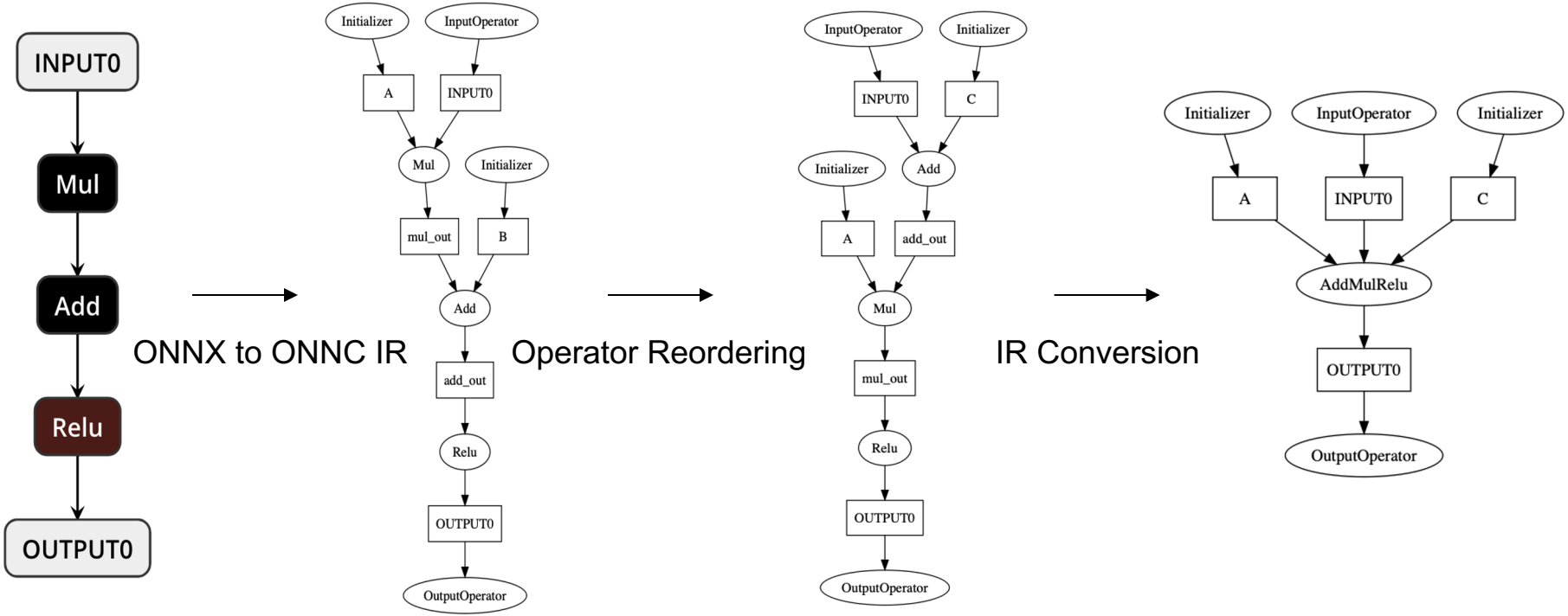


Option 3: Execution time ~= T/3
Memory request = tensor_size *2



skymizer

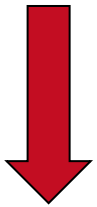
Hardware-Dependent Optimization For More Efficient Mapping



skymizer

Operator Reordering

$$Y = (X * alpha) + beta$$

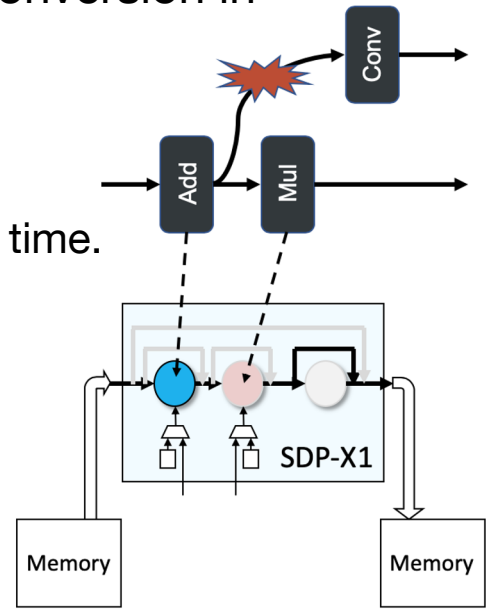


$$Y = (X + gamma) * alpha$$
$$gamma = beta / alpha$$

Operator Reordering

There are two pre-conditions to meet in order to have a valid conversion in this example.

1. The output of the **Add** operator can only have one consumer.
2. The values of **alpha** and **beta** need to be determined at compile time.



$$Y = (X * alpha) + beta$$

$$Y = (X + gamma) * alpha$$

$$gamma = beta / alpha$$

Lab - Mul And Add Re-Ordering And Fusion

In this lab, we will show how to create hardware-dependent optimization passes described in earlier slides

1. NvDlaReorderMulAddPass

2. NvDlaFuseAddMulReluPass

Setup Environment

```
# Skip this command if you have already had the Docker image.
$ docker pull onnc/onnc-community

# Prepare ONNC and tutorial source code.
$ git clone https://github.com/ONNC/onnc.git
$ cd onnc; git checkout tags/1.3.0; cd ..
$ git clone https://github.com/ONNC/onnc-tutorial.git

# Start the onnc/onnc-community Docker.
$ docker run -ti --rm \
-v <absolute/path/to/onnc>:/onnc/onnc \
-v <absolute/path/to/tutorial>:/tutorial \
onnc/onnc-community
```

Create New Backend - FooNvdla

```
$ cd /onnc/onnc
$ ./scripts/create-new-backend.sh FooNvdla

# Install the pre-built FooNvdla backend.
$ tar -zxvf <path/to/tutorial>/lab_4_Code_Emitting/src/FooNvdla.tar.gz \
-C <path/to/onnc>/lib/Target

$ cd /onnc/onnc-umbrella/build-normal
# Rebuild ONNC.
$ smake -j8 install

# Run ONNC to compile a DNN model.
$ onnc -mquadruple foonvdla \
/tutorial/models/test_group_Conv/test_group_Conv.onnx
FooNvdla is invoked
```

NvDlaReorderMulAddPass - Operator Reordering

- The re-ordering optimization is done in a pass named **NvDlaReorderMulAddPass**.

- We provide all of the related files in ***<path/to/tutorial>/lab_8_Mul_Add_Reordering_and_Fusion/src/*** directory.

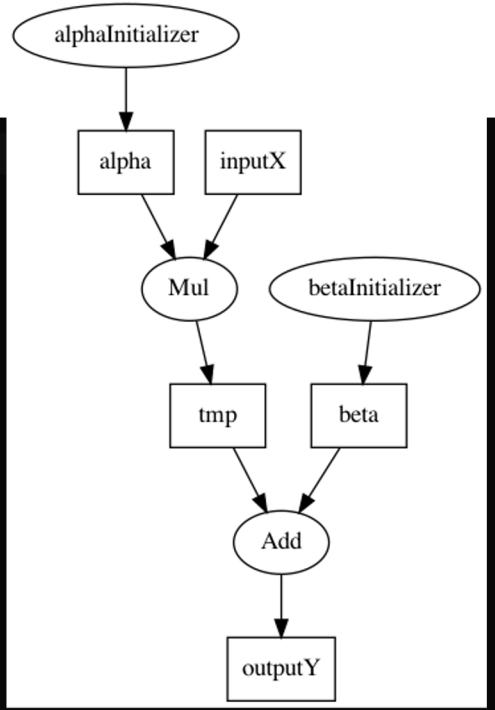
```
$ cp \
/tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/NvDlaReorderMulAddPass.* \
/onnc/onnc/lib/Target/FooNvdla
```

1. Search For The Matched Pattern

```
40  // File: NvDlaReorderMulAddPass.cpp
41  Pass::ReturnType NvDlaReorderMulAddPass::runOnComputeGraph(ComputeGraph& pCG)
42  {
43    Pass::ReturnType ret = Pass::kModuleNoChanged;
44    //------------------------------------------------------
45    // Search for the Mul-Add patterns that can be reordered.
46    //------------------------------------------------------
47    std::vector<ComputeOperator*> mulList;
48    for (ComputeOperator& node : pCG) {
49      if (canBeReordered(&node)) {
50        mulList.emplace_back(&node);
51        ret |= Pass::kModuleChanged;
52      }
53    }
54    ...
```

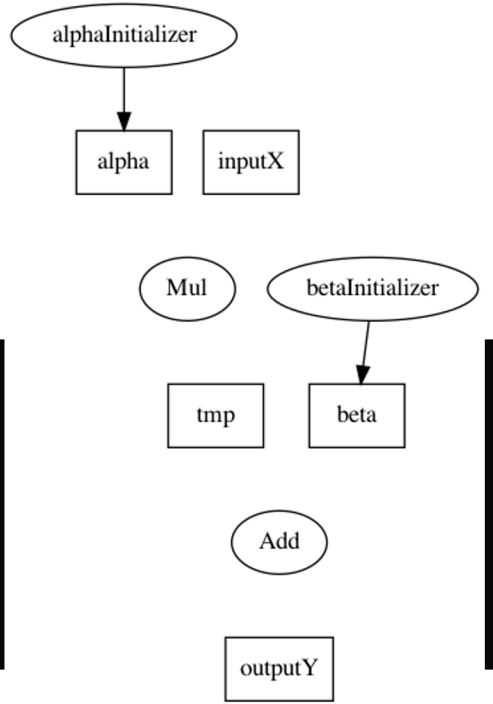
2. Iterate Through The Matched Patterns

```
54  // File: NvDlaReorderMulAddPass.cpp
55    //----------------------------------------
56    // Perform re-ordering on the found patterns.
57    //----------------------------------------
58    // The original pattern is:
59    //   outputY = (inputX * alpha) + beta
60    //
61    // We will re-arrange the above pattern by:
62    //   outputY = (inputX + gamma) * alpha, where
63    //     gamma = beta / alpha
64
65    for (ComputeOperator* node : mulList) {
66      Mul* mul = dyn_cast<Mul>(node);
67      Add* add = dyn_cast<Add>(node->getOutput(0)->getUses()[0].getUser());
68      ...
```



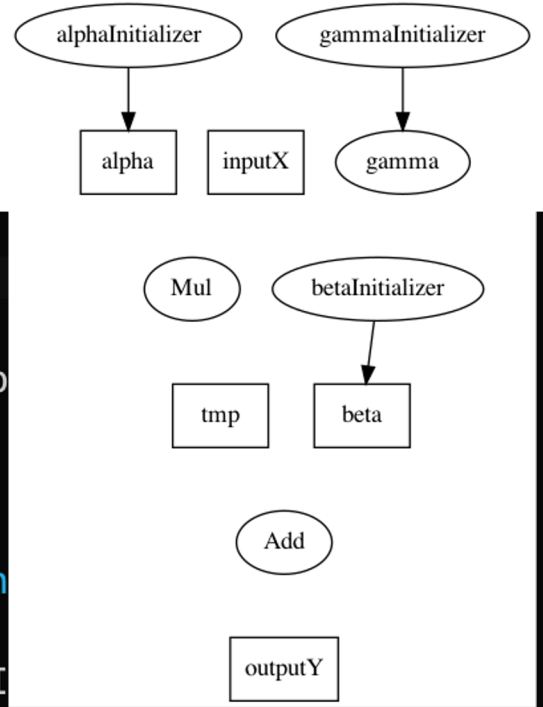
3. Remove Edges

```
120  // File: NvDlaReorderMulAddPass.cpp
121       mul->removeAllInputs();
122       mul->removeAllOutputs();
123       add->removeAllInputs();
124       add->removeAllOutputs();
```



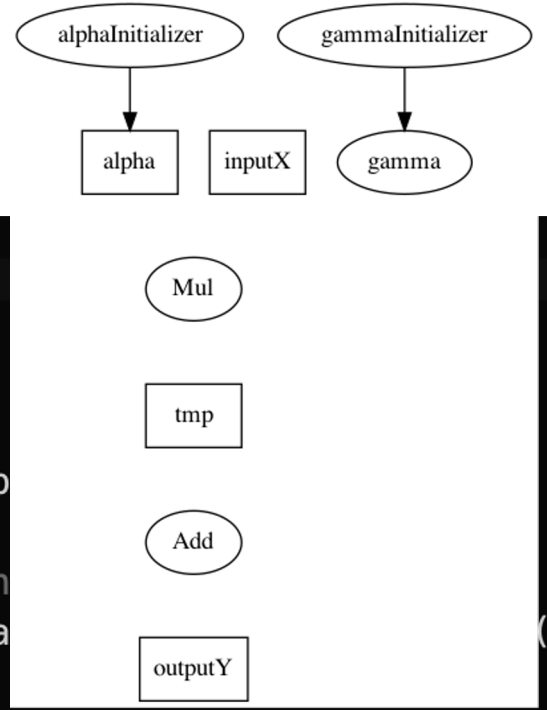
4. Create Gamma Tensor

```
143  // File: NvDlaReorderMulAddPass.cpp
144       // Create a new tensor gamma.
145       FloatTensor* gamma = dynamic_cast<FloatTensor*>(b
146       ...
147       // Add gamma into the ONNC IR graph.
148       gamma = pCG.addValue<FloatTensor>(gamma);
149       assert((gamma != nullptr) && "The name must be un
150       ...
151       Initializer* gammaInitializer = pCG.addOperator<I
152       gammaInitializer->setTensor(*gamma);
```



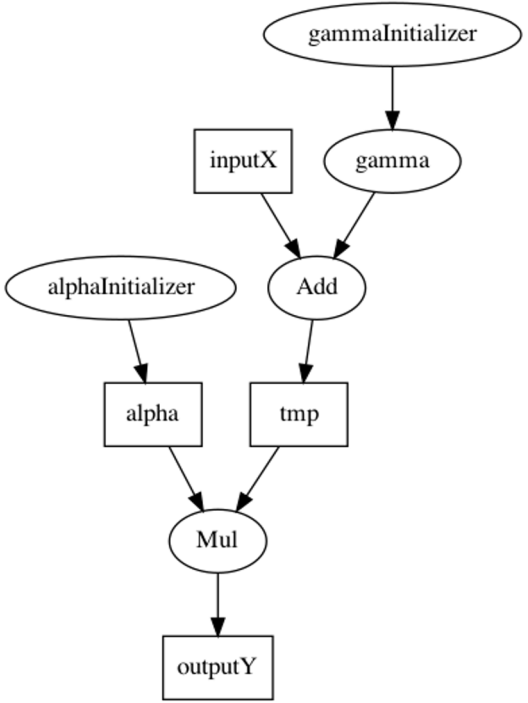
5. Remove Beta Tensor

```
184  // File: NvDlaReorderMulAddPass.cpp
185       // Calculate the constant data of gamma.
186       int tensorSize = beta->getValues().size();
187       for (int i = 0; i < tensorSize; i++) {
188         gamma->getValues().push_back( betaData[i] / alp
189       }
190       // Remove beta from the ONNC IR graph. We don't n
191       Initializer* betaInitializer = static_cast<Initia
192       pCG.erase(*betaInitializer);
193       pCG.erase(*beta);
```



6. Re-Connect The Operators

```
213  // File: NvDlaReorderMulAddPass.cpp
214       add->addInput(*inputX);
215       add->addInput(*gamma);
216       add->addOutput(*tmp);
217       mul->addInput(*tmp);
218       mul->addOutput(*outputY);
```



NvDlaFuseAddMulReluPass: Fuse Add-Mul-Relu Operators

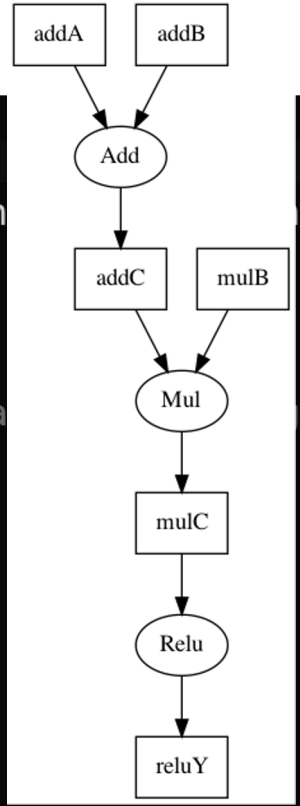
- The fuse optimization is done in a pass named **NvDlaReorderMulAddPass**.
- We provide all of the related files in ***<path/to/tutorial>/lab_8_Mul_Add_Reordering_and_Fusion/src/*** directory.

NvDlaFuseAddMulReluPass: Fuse Add-Mul-Relu Operators

```
$ mkdir -p /onnc/onnc/lib/Target/FooNvdla/Compute
# These files are about the definition of new IR.
$ cp /tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/NvDlaAddMulRelu.* \
/onnc/onnc/lib/Target/FooNvdla/Compute

# These files are about deploying the new IR into the model graph.
$ cp \
/tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/NvDlaFuseAddMulReluPass.* \
/onnc/onnc/lib/Target/FooNvdla

# These files are about the code emitting functions for the new IR.
$ cp /tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/CodeEmitVisitor.* \
/onnc/onnc/lib/Target/FooNvdla

# These files are about visualizing the optimization effect.
$ cp /tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/PrintONNCIRPass.* \
/onnc/onnc/lib/Target/FooNvdla
```

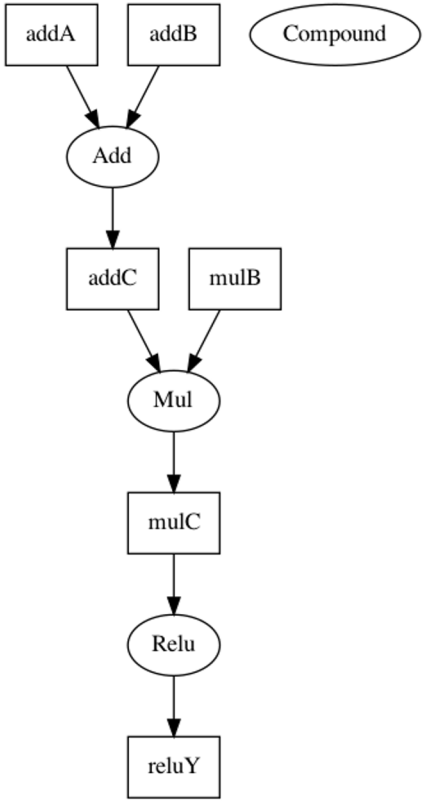
1. Find The Matched Pattern

```
37  // File: NvDlaFuseAddMulReluPass.cpp
38  Pass::ReturnType NvDlaFuseAddMulReluPass::runOnComputeGraph                & pCG)
39  {
40    Pass::ReturnType ret = Pass::kModuleNoChanged;
41
42    // Search for the Add-Mul-Relu patterns that can be repla                le Add
43    std::vector<ComputeOperator*> patternList;
44    for (ComputeOperator& node : pCG) {
45      if (isAddMulRelu(&node)) {
46        patternList.emplace_back(&node);
47        ret |= Pass::kModuleChanged;
48      }
49    }
```



2. Create A New Compound IR

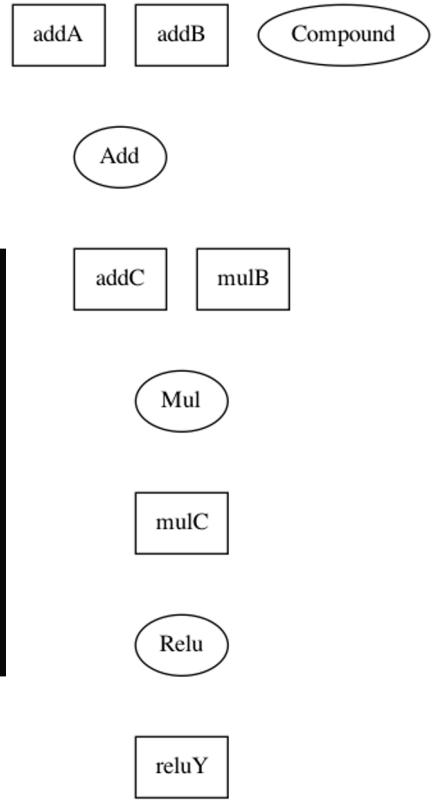
```
50  // File: NvDlaFuseAddMulReluPass.cpp
51    for (ComputeOperator* node : patternList) {
52      // Derive original IRs.
53      Add* add = dyn_cast<Add>(node);
54      Mul* mul = dyn_cast<Mul>(add->getOutput(0)->getUses()[0
55      Relu* relu = dyn_cast<Relu>(mul->getOutput(0)->getUses(
56      ...
57      // Create a new AddMulRelu IR.
58      NvDlaAddMulRelu* compound = pCG.addOperator<NvDlaAddMul
```



skymizer

3. Remove Edges

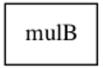
```
109  // File: NvDlaFuseAddMulReluPass.cpp
110       add->removeAllInputs();
111       add->removeAllOutputs();
112       mul->removeAllInputs();
113       mul->removeAllOutputs();
114       relu->removeAllInputs();
115       relu->removeAllOutputs();
```

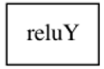


skymizer

4. Remove Original IRs


```
135  // File: NvDlaFuseAddMulReluPass.cpp
136       pCG.erase(*add);
137       pCG.erase(*mul);
138       pCG.erase(*relu);
139       pCG.erase(*addC);
140       pCG.erase(*mulC);
```

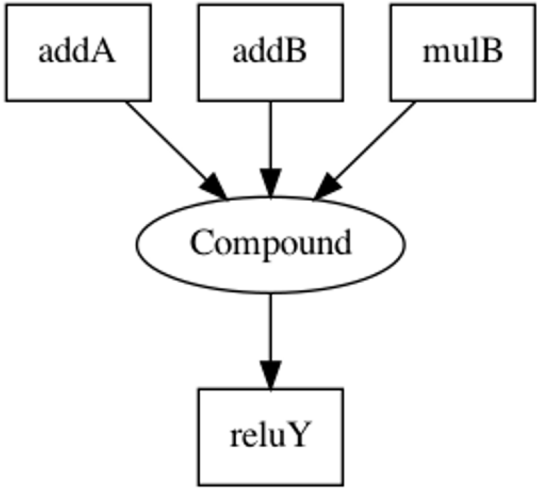




skymizer

5. Connect The New Compound IR

```
155  // File: NvDlaFuseAddMulReluPass.cpp
156       compound->addInput(*addA);
157       compound->addInput(*addB);
158       compound->addInput(*mulB);
159       compound->addOutput(*reluY);
```



Enable Passes In The Backend

```
$ cp /tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/FooNvdlaBackend.cpp \
/onnc/onnc/lib/Target/FooNvdla

$ cp /tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/CMakeLists.txt \
/onnc/onnc/lib/Target/FooNvdla
$ cp /tutorial/lab_8_Mul_Add_Reordering_and_Fusion/src/Makefile.am \
/onnc/onnc/lib/Target/FooNvdla
```

Rebuild ONNC & Compile Example Model

```
$ cd /onnc/onnc-umbrella/build-normal

# Rebuild ONNC.
$ smake -j8 install

# Execute ONNC to compile the model.
$ onnc -mquadruple foonvdla \
/tutorial/models/test_Mul_Add_Relu/test_Mul_Add_Relu.onnx
```

```
FooNvdla is invoked
=== PrintONNCIRPass ======
%A<float>[1, 1, 5, 5] = Initializer<unimplemented>()
%B<float>[1, 1, 5, 5] = Initializer<unimplemented>()
%INPUT0<float>[1, 1, 5, 5] = InputOperator<unimplemented>()
%mul_out<float>[1, 1, 5, 5] = Mul(%INPUT0<float>[1, 1, 5, 5], %A<float>[1, 1,
%add_out<float>[1, 1, 5, 5] = Add(%mul_out<float>[1, 1, 5, 5], %B<float>[1, 1,
%OUTPUT0<float>[1, 1, 5, 5] = Relu(%add_out<float>[1, 1, 5, 5])
 = OutputOperator<unimplemented>(%OUTPUT0<float>[1, 1, 5, 5])
==========================
NvDlaReorderMulAddPass is called...
=== PrintONNCIRPass ======
%A<float>[1, 1, 5, 5] = Initializer<unimplemented>()
%INPUT0<float>[1, 1, 5, 5] = InputOperator<unimplemented>()
%B__gamma_0)<float>[1, 1, 5, 5] = Initializer<unimplemented>()
%add_out<float>[1, 1, 5, 5] = Add(%INPUT0<float>[1, 1, 5, 5], %B__gamma_0)<flo
%mul_out<float>[1, 1, 5, 5] = Mul(%add_out<float>[1, 1, 5, 5], %A<float>[1, 1,
%OUTPUT0<float>[1, 1, 5, 5] = Relu(%mul_out<float>[1, 1, 5, 5])
 = OutputOperator<unimplemented>(%OUTPUT0<float>[1, 1, 5, 5])
==========================
NvDlaFuseAddMulReluPass is called...
=== PrintONNCIRPass ======
%A<float>[1, 1, 5, 5] = Initializer<unimplemented>()
%INPUT0<float>[1, 1, 5, 5] = InputOperator<unimplemented>()
%B__gamma_0)<float>[1, 1, 5, 5] = Initializer<unimplemented>()
%OUTPUT0<float>[1, 1, 5, 5] = AddMulRelu<>(%INPUT0<float>[1, 1, 5, 5], %B__gam
 = OutputOperator<unimplemented>(%OUTPUT0<float>[1, 1, 5, 5])
==========================
visit(NvDlaAddMulRelu) is called
```

kymizer

skymizer

Boost deep learning accelerator
with compiler technology

Skymizer Taiwan Inc.

CONTACT US

E-mail sales@skymizer.com **Tel** +886 2 8797 8337

HQ 12F-2, No.408, Ruiguang Rd., Neihu Dist., Taipei City 11492, Taiwan

BR Center of Innovative Incubator, National Tsing Hua University, Hsinchu Taiwan

https://skymizer.com