



Skymizer | Porting ONNC to NVDLA

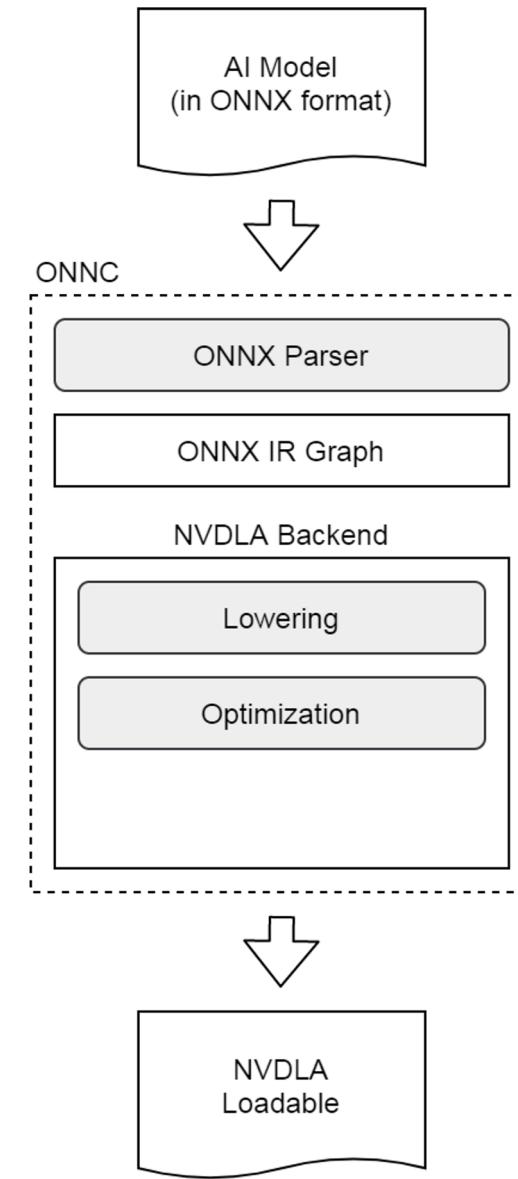
Po-Yen Chen (poyenc@skymizer.com)



Implement a NVDLA Backend

What do we need to do to port ONNC to NVDLA?

- Legalization – Convert unsupported operators and attributes to a set of supported operators.
- Map ONNX operators to NVDLA execution units
- Generate NVDLA loadable



Legalization

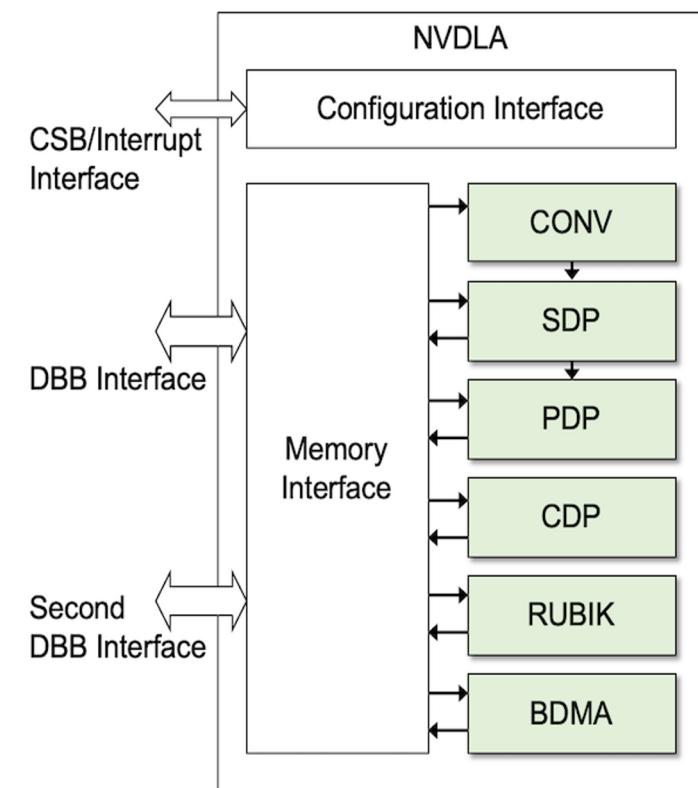
Convert unsupported operators and attributes to a set of supported operators.

- Replace *Gemm* to *Conv*
- Replace *GlobalAveragePool* by *AveragePool*
- Replace *Clip* to *Min & Max*
- Calibrate *AveragePool* result conditionally (without padding)
- Split *Conv* kernel

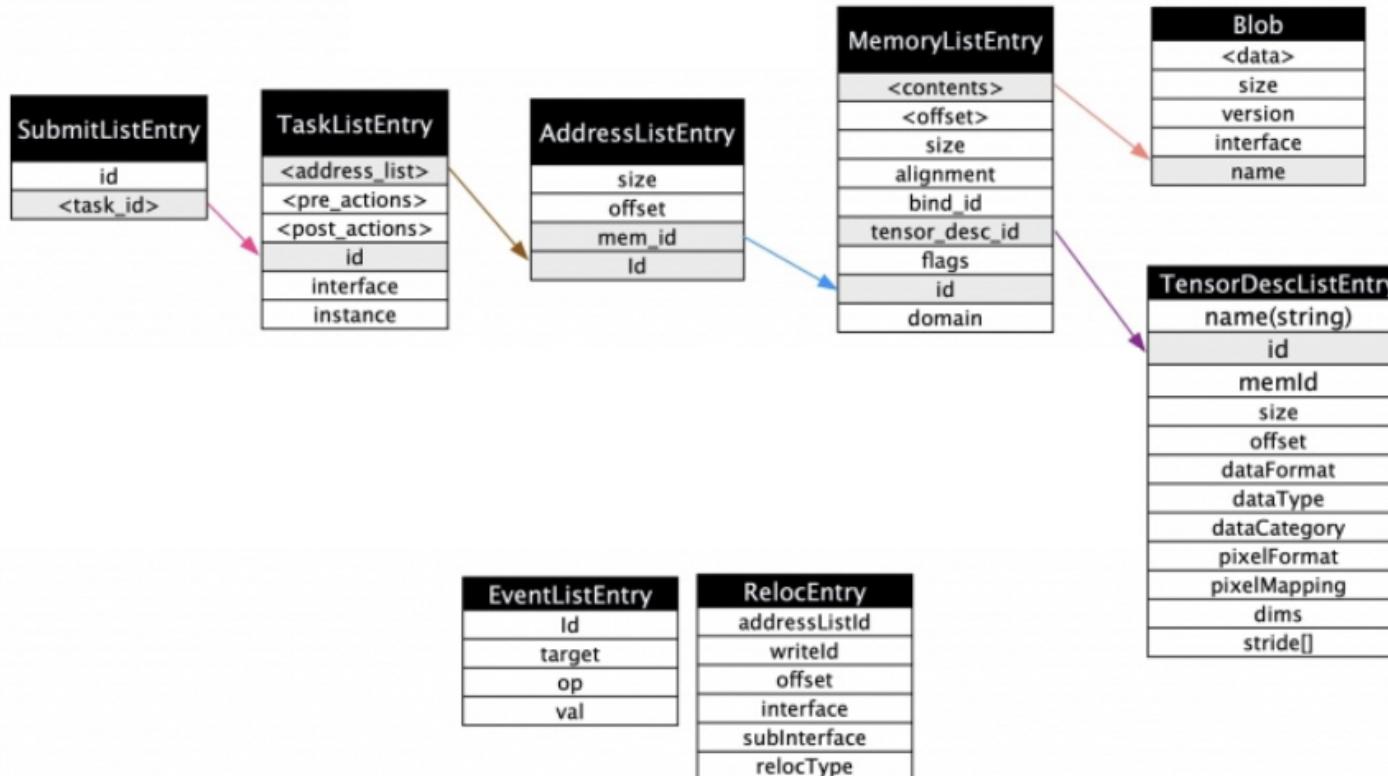


Mapping ONNC IRs to Execution Units

Hardware Execution Unit	Operators
CONV + SDP	<i>Conv (Gemm)</i>
SDP	<i>Relu, Add, Mul, Sum, Max, Min, PReLU (BatchNormalization, Clip)</i>
PDP	<i>MaxPool, AveragePool (GlobalAveragePool, ReduceMean)</i>
CDP	<i>LRN</i>
RUBIK	<i>Transpose</i>
CPU	<i>Softmax</i>
N/A	<i>Concat, Reshape, Unsqueeze, Identity</i>



Generate NVDLA Loadables



- NVDLA-supported vs fall-back-to-CPU operators
- UMD parses Loadables and KMD drives NVDLA based on the network information.



Programming Interface

NVDLA Open Source Software repository:

<https://github.com/nvdla/sw>

- NVDLA Loadable format defined in
[umd/core/src/common/include/priv/Loadable.h](https://github.com/nvdla/sw/blob/master/umd/core/src/common/include/priv/Loadable.h)
- Programming interface defined in
[kmd/firmware/include/dla_interface.h](https://github.com/nvdla/sw/blob/master/kmd/firmware/include/dla_interface.h)
- We use the *Loadable* structure generate NVDLA loadable file and configure each hardware layer through *dla_interface.h* types



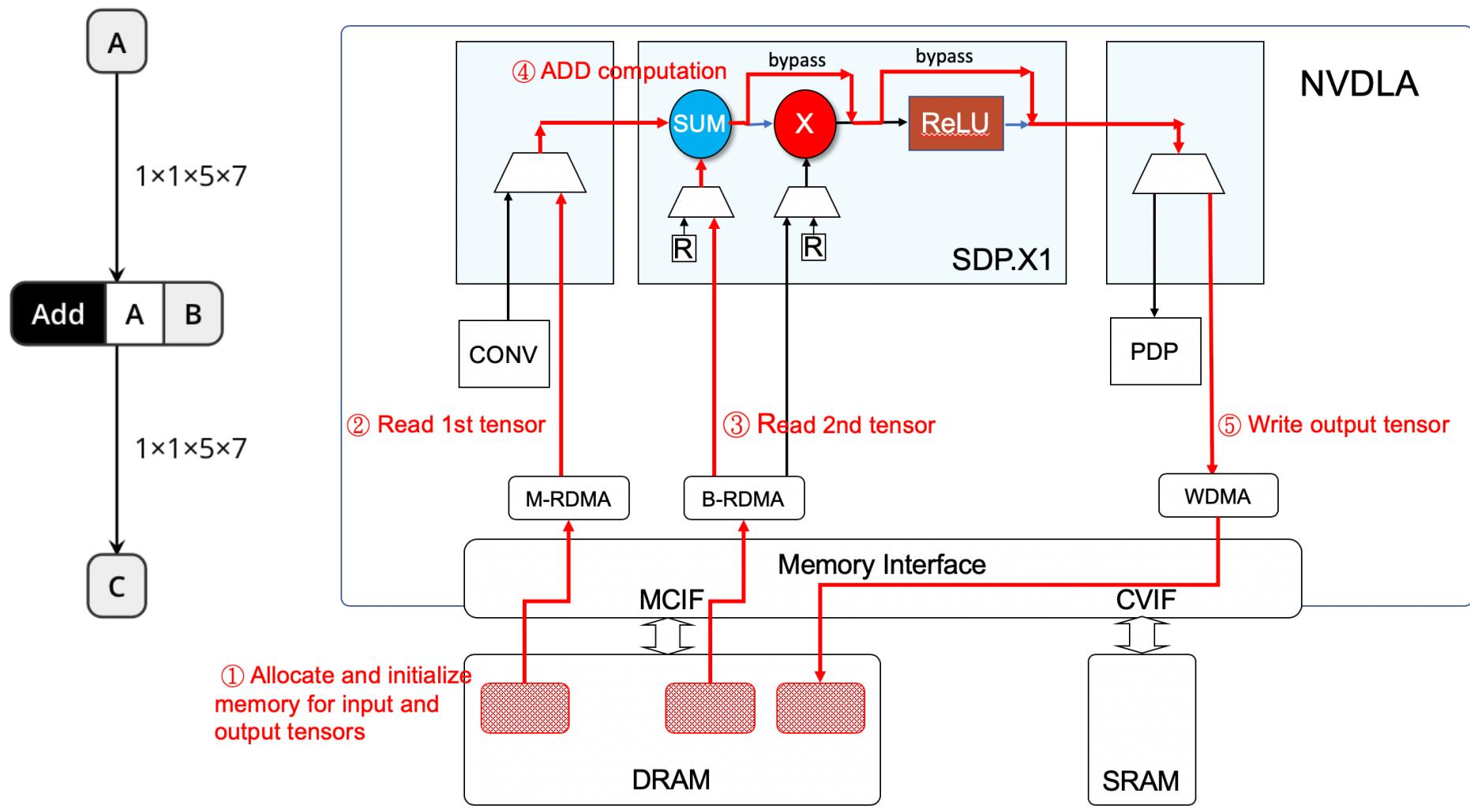
NVDLA Operation Descriptors

Operation descriptor & surface descriptor

- Common operation descriptor decide what to do for each hardware layers ([dla_common_op_desc](#))
- Operation descriptor decide how to do for each layer ([dla_operation_container](#))
- Surface descriptor decide the input/output data shape and location ([dla_surface_container](#))
- In backend method *addCodeEmit()*, we create descriptor instances for each ONNC IR (implemented in *CodeEmitVisitor*)



Support a New Operator -Add



Support a New Operator -Add

Register lower for Add IR in NVDLA backend

```
1 // NvDlaBackend.cpp
2 #include <onnc/Transforms/TensorSel/Standards/ConvLower.h>
3 // 1. Add include directive for AddLower declaration
4 #include <onnc/Transforms/TensorSel/Standards/AddLower.h>
5
6 void NvDlaBackend::RegisterLowers(LowerRegistry& pRegistry) const
7 {
8     pRegistry.emplace<ConvLower>();
9     // 2. Add AddLower for accepting ONNX Add IR
10    pRegistry.emplace<AddLower>();
11 }
```



Visitor Design Pattern

We dispatch ONNC IR to *CodeEmitVisitor*, and mapping ONNC IR to hardware execution units in its *visit()* methods. e.g. *Add*:

```
1 // CodeEmitVisitor.h
2 class CodeEmitVisitor : public CustomVisitor<CodeEmitVisitor>, private NvDlaConstants
3 {
4 public:
5     void visit(const Conv& pConv) override;
6     void visit(Conv& pConv) override;
7
8     // 1. add visit method for const Add IR
9     void visit(const Add& pOp) override;
10    // 2. add visit method for non-const Add IR, force it calls the previous version
11    void visit(Add& pOp) oroverride { visit(const_cast<const Add&>(pOp)); }
12 }
```



Descriptor Wrapper

The definition of *NvDlaDlaOperation* class, it wraps 3 descriptor structures

```
1 // NvDlaMeta.h
2 class NvDlaDlaOperation
3 {
4 public:
5     NvDlaDlaOperation() noexcept;
6
7 public:
8     struct dla_common_op_desc      op_dep;
9     union dla_operation_container op_desc;
10    union dla_surface_container   op_surf;
11};
```



Operation Type

All the NVDLA execution unit op type values

[kmd/firmware/include/dla_interface.h:34](#)

```
34  /**
35   * @ingroup Processors
36   * @name DLA Processors
37   * Processor modules in DLA engine. Each processor has it's
38   * own operation a.k.a. HW layer. Network is formed using
39   * graph of these operations
40   * @{
41   */
42 #define DLA_OP_BDMA          0
43 #define DLA_OP_CONV           1
44 #define DLA_OP_SDP            2
45 #define DLA_OP_PDP            3
46 #define DLA_OP_CDP            4
47 #define DLA_OP_RUBIK          5
48 /** @} */
```



Visit() –Setup Common Op Descriptor

A closer look to the Add *visit()* method (part 1/3)

- Setup common op descriptor

```
1 void CodeEmitVisitor::visit(const Add& pOp)
2 {
3     // Get tensor attributes.
4     const Tensor& first = *(pOp.getInput(0));
5     const Tensor& second = *(pOp.getInput(1));
6     const Tensor& output = *(pOp.getOutput(0));
7
8     //-----
9     // Configure hardware block
10    //-----
11    NvDlaDlaOperation* operation = new NvDlaDlaOperation();
12    // Set hardware block type.
13    operation->op_dep.op_type = DLA_OP_SDP;
14 }
```



Visit() –Setup SDP Op Descriptor

A closer look to the Add *visit()* method (part 2/3)

- Setup SDP op descriptor

```
15 struct dla_sdp_op_desc& desc = (struct dla_sdp_op_desc&)(operation->op_desc);  
16 desc.src_precision      = PRECISION_FP16;  
17 desc.dst_precision      = PRECISION_FP16;  
18 // No look up table is required.  
19 desc.lut_index          = -1;  
20  
21 // For this example, we only support batch == 1.  
22 desc.batch_num          = 1;  
23 desc.batch_stride       = 0;
```



Visit() –Setup SDP Op Descriptor

A closer look to the Add *visit()* method (part 2/3)

- Setup SDP op descriptor (cont'd)

```
25    // Enable X1 block.  
26    desc.x1_op.enable      = 1;  
27  
28    // X1 operation Options: Disable (SDP_OP_NONE) / ALU only (SDP_OP_ADD) /  
29    //                                Multiplier only (SDP_OP_MUL) / ALU+MUL (SDP_OP_BOTH)  
30    desc.x1_op.type       = SDP_OP_ADD;  
31  
32    // ALU type options: SUM/MIN/MAX  
33    desc.x1_op.alu_type   = SDP_ALU_OP_SUM;  
34  
35    // Disable ReLU  
36    desc.x1_op.act        = ACTIVATION_NONE;  
37  
38    // Set per_layer/per_channel/per_point mode based on the broadcasting type.  
39    // For this example we only support per_point mode.  
40    desc.x1_op.mode        = SDP_OP_PER_POINT;  
41  
42    // Set the datapath precision to be fp16.  
43    desc.x1_op.precision   = PRECISION_FP16;
```



Visit() –Setup SDP Surface Descriptor

A closer look to the Add *visit()* method (part 3/3)

■ Setup SDP surface descriptor

```
45 //-----
46 // Setup dataflow sources and destination
47 //-----
48 struct dla_sdp_surface_desc& surface = (struct dla_sdp_surface_desc&)(operation->op_surf);
49
50 // Setup 1st tensor source.
51 const NvDlaCubeInfo firstCubeInfo = makeCubeInfo(*this, NVDLA_CUBE_FEATURE, first);
52 // The 1st input tensor can be read from:
53 //   external DRAM via the interface of MCIF: DLA_MEM_MC
54 //   SRAM via the interface of CVIF: DLA_MEM_CV
55 //   the output of CONV hardware block: DLA_MEM_HW
56 // In this example, we only support the 1st input tensor is stored at external DRAM.
57 surface.src_data.type = DLA_MEM_MC;
58 // Setup memory allocation and DMA configuration for 1st input tensor.
59 surface.src_data.address = issueDlaAddr(first, firstCubeInfo);
60 surface.src_data.size = m_pMeta.getMemoryListEntrySize(first);
61 surface.src_data.width = firstCubeInfo.dim_w;
62 surface.src_data.height = firstCubeInfo.dim_h;
63 surface.src_data.channel = firstCubeInfo.dim_c;
64 surface.src_data.line_stride = firstCubeInfo.stride_line;
65 surface.src_data.surf_stride = firstCubeInfo.stride_surface;
```



Visit() –Setup SDP Surface Descriptor

A closer look to the Add *visit()* method (part 3/3)

- Setup SDP surface descriptor (cont'd)

```
67     // Setup 2nd tensor source.  
68     MemoryListEntryId    memoryId;  
69     const NvDlaCubeInfo secondCubeInfo = makeCubeInfo(*this, getSdpXSingleCubeType(second, DLA_PRECISION), second);  
70     // The 2nd input tensor is stored at DRAM and accessed through the interface of MCIF.  
71     surface.x1_data.type          = DLA_MEM_MC;  
72     // Setup memory allocation and DMA configuration for 2nd input tensor.  
73     // In addition, the 2nd tensor is constant so need be packed into a blob and becomes a part of loadable.  
74     surface.x1_data.address      = issueSDPOperand(second, secondCubeInfo, memoryId);  
75     surface.x1_data.size        = m_pMeta.getMemoryListEntrySize(memoryId);  
76     surface.x1_data.width       = secondCubeInfo.dim_w;  
77     surface.x1_data.height      = secondCubeInfo.dim_h;  
78     surface.x1_data.channel     = secondCubeInfo.dim_c;  
79     surface.x1_data.line_stride = secondCubeInfo.stride_line;  
80     surface.x1_data.surf_stride = secondCubeInfo.stride_surface;
```



Visit() –Setup SDP Surface Descriptor

A closer look to the Add *visit()* method (part 3/3)

■ Setup SDP surface descriptor (cont'd)

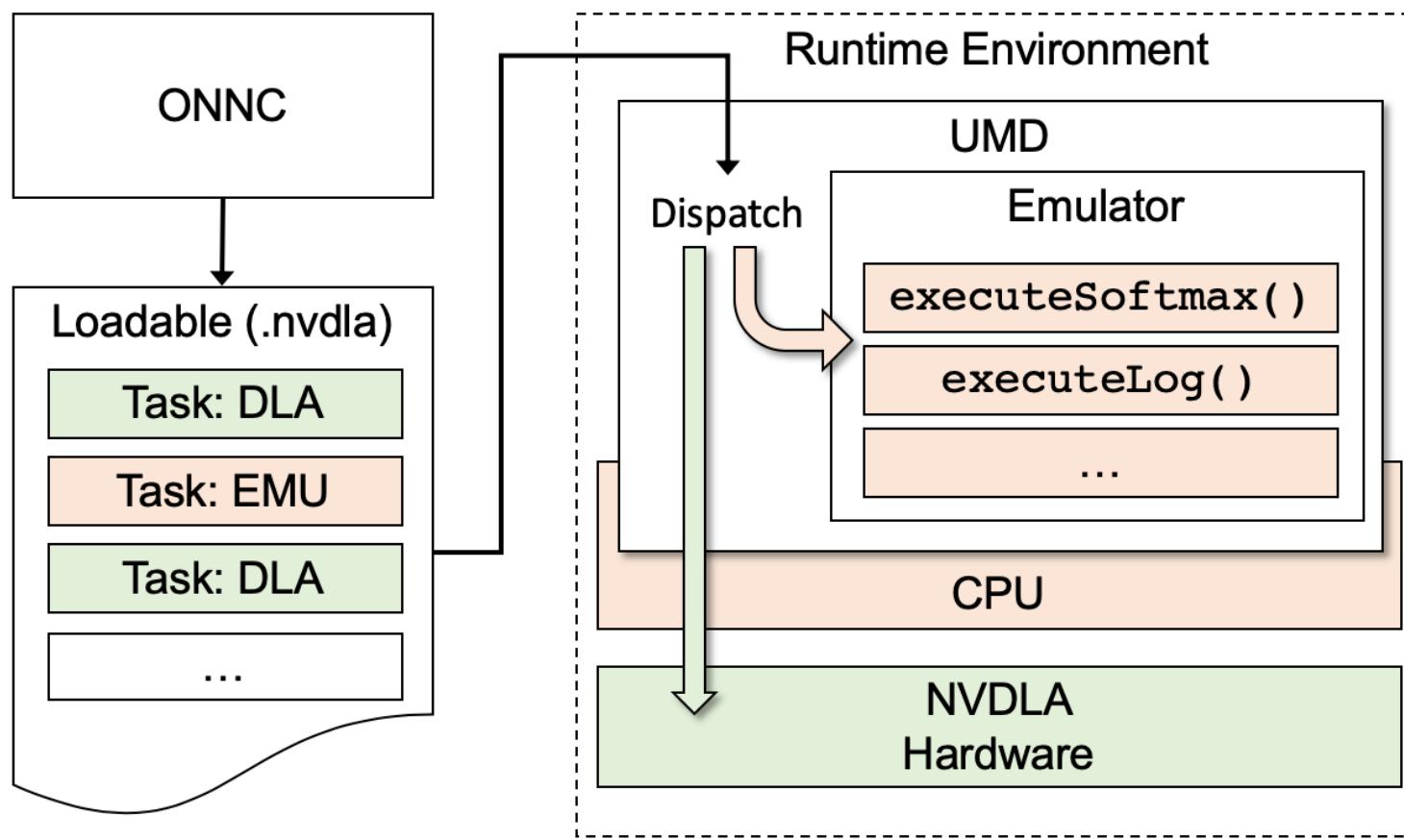
```
82    // Setup output tensor destination.  
83    const NvDlaCubeInfo outputCubeInfo = makeCubeInfo(*this, NVDLA_CUBE_FEATURE, output);  
84    // The output tensor is stored at DRAM.  
85    surface.dst_data.type      = DLA_MEM_MC;  
86    surface.dst_data.address   = issueDlaAddr(output, outputCubeInfo);  
87    surface.dst_data.size     = m_pMeta.getMemoryListEntrySize(output);  
88    surface.dst_data.width    = outputCubeInfo.dim_w;  
89    surface.dst_data.height   = outputCubeInfo.dim_h;  
90    surface.dst_data.channel  = outputCubeInfo.dim_c;  
91    surface.dst_data.line_stride = outputCubeInfo.stride_line;  
92    surface.dst_data.surf_stride = outputCubeInfo.stride_surface;  
93  
94    //-----  
95    //  enlist the operation  
96    //-----  
97    issueDlaOp(operation, NULL, m_pMeta.m_pPrevOp);  
98 }
```

Detail covered in [ONNC tutorial lab 4](#)



CPU Fallback Support

However, not all computations are supported by NVDLA execution units, for example: *Softmax*



Extend the Loadable Data Structure.

Add new emu operation for *Sofmax* in *emu_interface.h*

```
// emu_interface.h

#define NVDLA_EMU_OP_POWER      0
+/#define NVDLA_EMU_OP_SOFTMAX  1
```



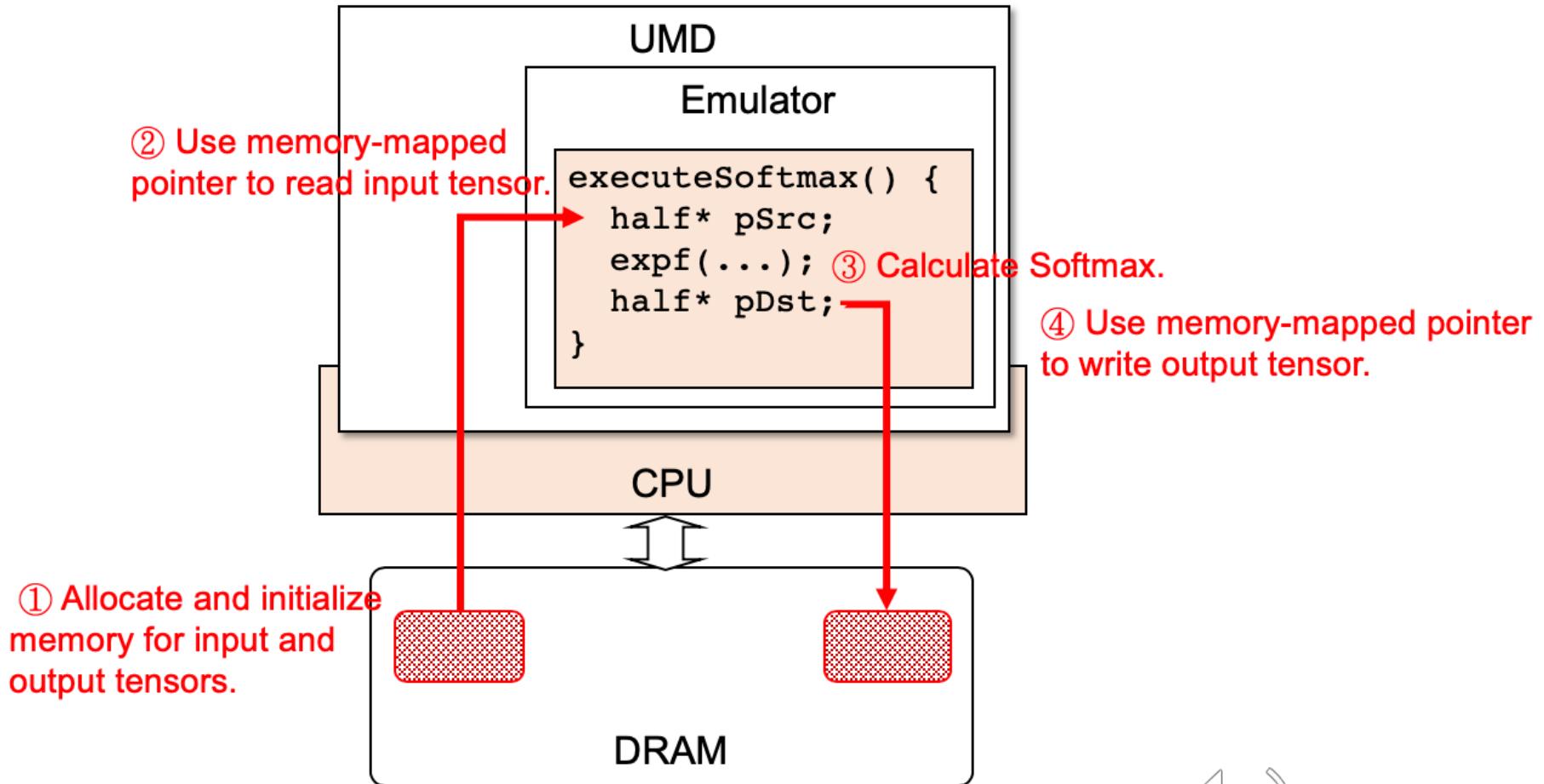
```
+struct emu_softmax_op_desc
+{
+    emu_common_op_desc common;
+    NvU8 axis;
+} __attribute__ ((packed, aligned(4)));

union emu_operation_container
{
    struct emu_power_op_desc power_op;
+    struct emu_softmax_op_desc softmax_op;
};
```

```
+struct emu_softmax_buffer_descs
+{
+    struct emu_buffer_desc src_data;
+    struct emu_buffer_desc dst_data;
+} __attribute__ ((packed, aligned(4)));

union emu_operation_buffer_container
{
    struct emu_power_buffer_descs power_buffe
+    struct emu_softmax_buffer_descs softmax_
};
```

Add Corresponding Code Emitting Function



Extend UMD Emulator

- <path/to/sw>/umd/core/common/EMUInterface.cpp
- <path/to/sw>/umd/core/common/EMUInterfaceA.cpp
- <path/to/sw>/umd/core/common/include/priv/EMUInterface.h
- <path/to/sw>/umd/core/common/include/priv/EMUInterfaceEnums.h
- <path/to/sw>/umd/core/runtime/Emulator.cpp
- <path/to/sw>/umd/core/runtime/include/priv/Emulator.h

```
// Emulator.cpp

bool Emulator::processTask(NvU8* task_mem, std::vector<NvU8*> addressList)
{
    if (opType == EMUOpType::POWER) {
        executePower(power_op_desc, power_op_buffer_descs, addressList);

    } else if (opType == EMUOpType::SOFTMAX) {
        executeSoftmax(softmax_op_desc, softmax_op_buffer_descs, addressList);

    }
}

bool Emulator::executeSoftmax(EMUSoftmaxOpDescAccessor opDesc,
                            EMUSoftmaxBufferDescsAccessor bufDescs,
                            std::vector<NvU8*> addressList)
```





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



skymizer

Boost deep learning accelerator
with compiler technology



<https://skymizer.com>

