connect ONNX to every deep learning accelerator

https://onnc.ai
<table>
<thead>
<tr>
<th>target</th>
<th>heterogeneous architecture</th>
<th>single architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming model</td>
<td>PetriNet(1)</td>
<td>CFG and DFG</td>
</tr>
<tr>
<td>IR type</td>
<td>ONNX IR (multiple outputs)</td>
<td>three address code (single output)</td>
</tr>
<tr>
<td>physical feature depends on</td>
<td>operand</td>
<td>opcode</td>
</tr>
</tbody>
</table>

https://onnc.ai
Heterogeneous System
to well control the data flow

- Xeon
- PCIe
- Xeon
- host interface
- DDR
- DDR IF
- weight FIFO
- 30 GB/s
- DSP or powerful MCU
- 140 GB/s
- Matrix operator
- Element-wise operator
Compulsory Spill
is easy to implement in the other compiler framework

flexible

cost effective

https://onnc.ai
Memory Spill
is what we already have in every compiler framework

https://onnc.ai
Operator Spill
is totally new and required for every accelerators

https://onnc.ai
<table>
<thead>
<tr>
<th>target</th>
<th>heterogeneous architecture system (HSA)</th>
<th>single architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>programming model</td>
<td>PetriNet(1)</td>
<td>CFG and DFG</td>
</tr>
<tr>
<td>IR type</td>
<td>ONNX IR (multiple outputs)</td>
<td>three address code (single output)</td>
</tr>
<tr>
<td>physical feature</td>
<td>operand</td>
<td>opcode</td>
</tr>
</tbody>
</table>

https://onnc.ai
Memory runtime consumption

Limited DLA
save x3.77 in avg.

paging system

- ONNC reuses local memory and save runtime memory consumption by life range analysis of tensors
- The runtime memory consumption includes inputs, outputs and weights

Experimental environment

Ubuntu Linux 16.04

Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz

32G RAM
ONNC supports various target devices

- Use LLVM-like triple to select compiler target backend
  - compiler
  - loader
  - calibration
- Use architecture description to select target platform
  - A platform contains multiple target backends
- Platform and Target instance register target-dependent passes into PassManager

```
use select target with quadruple
Platform
Target
```

Diagram:

```
select target with ADL
Driver

Platform
Target
NVDLA | BITMAIN | X86_64

TensorSel | MemAlloc | CodeEmit

cost models
lowering IR / optimizations
```
process control  Pass manager  None. Simple linked list
Transformation: from Compiler’s point of view

- **Target phases**
  - TensorSel - turn ONNX layer to target-specific operator
  - MemAlloc - turn symbols of tensor to memory address
  - CodeEmit - turn IR to machine instruction

- **Platform phases**
  - TensorPartition - map subgraph to hardware processing unit
  - OprSched - find the timing sequence of a subgraph
Four Kinds of Passes in ONNC

- **ModulePass**
  - The most general of all superclasses that you can use
  - Use entire network as an unit
- **TensorPass**
  - Use Tensor Graph as an unit
  - Tensor Graph bases on ONNX IR
- **RegionPass**
  - Use each single-entry-single-exit region in a tensor graph as an unit
  - For example, groups in GoogLeNet
- **ComputePass**
  - Use Compute Graph as an unit

```cpp
// methods in class Pass
bool run(Module& pModule);
virtual bool doInitialization(Module& pModule);
virtual bool doFinalization(Module& pModule);
```

```cpp
// methods in class ModulePass
virtual bool runOnModule(Module& pModule) = 0;
```

```cpp
// methods in class TensorPass
virtual bool runOnTensor(TensorGraph& pGraph) = 0;
```
AnalysisUsage describes the dependencies between Passes

- PassManager automatically creates all Passes that used by the other Passes.
- Similar to LLVM. Engineers who already familiar to LLVM can understand ONNC in a short time

```cpp
/// in A.cpp
INITIALIZE_PASS(A, "pass_a")

/// in B.cpp
INITIALIZE_PASS(B, "pass_b")
```

```cpp
/// methods in Pass D. Override
void D::getAnalysisUsage(AnalysisUsage &pUsage) const
{
    pUsage.addRequiredID(A::ID);
    pUsage.addRequiredID(B::ID);
}
```
Innate Iterative Compilation

traditional compiler

compilation model

ITERATIVE

sequential

Lattice

A
B
C
D

Add D, PassManager will add A and B automatically
Connect to both LLVM and ASIC

- No porting effort for LLVM compiler
- Support ASIC which can NOT be supported by LLVM

https://onnc.ai
CPUs + ARM Cortex-M + NVDLA

main processor

controller

DLA

x86 (LLVM)

ARM Cortex-A (LLVM)

ARM Cortex-M (uTensor + mbed + CMSIS)

ARC

RISC-V

CEVA

Tensilica

NVDLA

DLA
Refactoring uTensor

changes for uTensor

Refactoring uTensor • introduce RTBuilder

changes in ONNC

Linear Scan • uTensor backend

we’re here
Projects reside in https://repo.onnc.ai

The **Regression** project for testers

The **Umbrella** project for developers

https://onnc.ai
# Current Status

<table>
<thead>
<tr>
<th>Version</th>
<th>release date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.0</td>
<td>8/2</td>
<td>Initial release</td>
</tr>
<tr>
<td>0.9.1</td>
<td>8/3</td>
<td>Sophon DLA</td>
</tr>
<tr>
<td>0.9.2</td>
<td>9/18</td>
<td>memory allocation (linear scan algorithm)</td>
</tr>
</tbody>
</table>
| 0.9.3   | 10/1 (ETA)   | x86 interpreter  
            | Platform registry |
| 1.0.0   | 10/14 (ETA)  | x86 JIT and bundle  
            | uTensor/CMSIS interpreter |
| 1.1.0   | 10/31 (ETA)  | NVDLA bundle  
            | Platform passes |

# Next release

<table>
<thead>
<tr>
<th>Version</th>
<th>release date</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>